

Abstract

The amount of information on the Web is growing rapidly, as well as the number of new users inexperienced in the art of Web research, which leads to the development of many Web applications called search engines specialized in helping the user in finding the information needed on the Web.

Web site search engine is software used for searching a specific Web site for a specific query.

This research aims to develop a Web site search engine that helps the user to find the most relevant Web pages with the user queries. The ranking part depends on the word attributes (such as font size, font style, font color, position of the word in the page, link text, title, header) and mixed with the indexing part. The index is spread in 36 binary files to reduce the system resources (Memory and Storage required) and the inverted index is created by sorting the index using the Improved Quick and Insertion sort methods to increase the indexing speed. The Lexicon is designed using a Multilayer structure with 4 layers.

The ranking part of the proposed Web site search engine based only on the word attributes, using the link structure of the Web will increase the ranking efficiency.

The proposed Web site search engine requires a computer installed as a server.

The programming tools used in developing the Web site search engine are: HyperText Markup Language, Visual Basic Script, Common Gateway Interface technique, Microsoft Visual Basic 6.0, and Windows operating system Socket.

Acknowledgment

I would like to express my sincere appreciation to my research supervisor, Dr. Moaid A. Fadhil, for giving me the major steps to go on to explore the subject, shearing with me the ideas in my research "Design and Implementation of an Internet Search Engine" And perform the points that I felt were important.

Also I wish to thank, Dr. Ban N. Al-kallak my supervisor and Dr. Loay A. George for their available advice and encouragement. Grateful thanks for the Head of Department of Computer Science Dr. Taha S. Bashaga.

I wish to thank the staff of Computer Science Department at the AL-Nahrain University for their help.

I would like to say "thank you" to my faithful friends for supporting and giving me advises.



Appendix A

The Internet

A.1 How the Internet work [Dav00]

In order to connect two dissimilar computers, they must be able to speak the same language. The Arpanet researchers realized this early on and developed a standard method for these machines to communicate, which they called Network Control Protocol (NCP). NCP provided a standardized format for how two different machines would exchange information. The early NCP was adequate to support further research activities performed by various researchers connected to the Arpanet, but it soon become apparent that NCP itself was in need of further research and development.

At about the same time that Ethernet was being developed in the labs of Palo Alto, ARPA (Advanced Research Projects Agency, a branch of the U.S. federal government) had began funding additional research activities on computer networks using various underlying methods of transmitting data, including radio and satellite communication. In 1973, to keep pace with the development of these new networking strategies, ARPA encouraged research into a new protocol that would supersede the Network Control Protocol and allow for host-to-host communication across any of these disparate networks.

As a result of this effort, a new suite of protocols was developed. This suite is referred to as the Internet Protocol Suit; however, this name is typically overshadowed by two of the more famous protocols produced as a

Appendix A: The Internet

result of this new research: Internet Protocol and Transmission Control Protocol (TCP/IP). ARPA once again made public the discoveries and the design of their new TCP/IP.

This new protocol suite was intended from the beginning to be a network of networks. This concept was labeled Internetworking, and the actual project became known as the Internet project.

Therefore, the Internet is not just a single network but a network of networks that now span the globe.

A.2 Internet Protocol Layers [Dav00]

The Internet protocol suite is based on four distinct and separate protocol layers. These layers are defined to work synergistically, allowing messages to flow cleanly across any number of dissimilar networks.

In early networks and, indeed, the networks of today, protocols exist for both the physical layer that connects machines together as well as for the logical layer that is used to transmit data across the physical layer.

In the Internet the IP serves as the delivery mechanism that moves these packets from source to destination. The IP layer routes each packet submitted to it by the TCP to its final destination. Each packet can be routed through a different path to reach its destination.

When discussing Internet communications, it is common to see these two protocols referred to as TCP/IP because TCP is generally layered on top of the lower level IP protocol, as shown in the following figure:

Appendix A: The Internet

Application layer	Various application protocols such as http, telnet, Ftp, tftp, bootp, snmp, etc
Transport layer	The TCP
Routing layer	The IP and ICMP protocols
Physical layer	Not specified by the Internet suite, can be most any independent networking standard (i.e., Ethernet, Token ring, FDDI)

These two protocols exist in top of some physical layer comprising the actual network hardware, such as Ethernet. At the highest level is some application layer such as HTTP, FTP, or Telnet.

A.3 The Internet Protocol (IP) [Dav00]

In order for the various networks connected to the Internet to share data, they must use a common data format once their messages leave their native network and travel out onto the Internet. This common format is IP, and any computer connected to the Internet must format its message according to the IP.

IP used what are known as datagrams to transmit data from one machine to another. A datagram is just a defined way of organizing the information before it is sent. When a computer on a particular network wishes to send a message to another machine, it packages an IP datagram and sends it out onto its local network. The datagram is actually contained in the data field of the lower-level data frame employed by the particular LAN (Local Area Network) technology being used.

IP is known as a connectionless, best-attempt delivery system. Connectionless means that there is no attempt made to verify that the destination machine exists before sending the message. Best-attempt delivery means that although successful delivery of the datagram is important, no guarantee is provided. If the message is not successfully delivered, no error status is returned to the sender. The detail of establishing a connection with the destination machine and verifying delivery are left up to higher-level protocols that ride on top of the IP delivery system.

A.4 The Transmission Control Protocol (TCP) [Dav00]

Riding on top of the routing layer is the transport layer, this layer is responsible for providing more robust delivery than can be provided for by the connectionless, best-attempt delivery Internet Protocol.

This layer is where Transmission Control Protocol comes in. TCP is the layer that applications typically interface with to send and receive message across the Internet.

TCP provides for a reliable, connection-oriented data transmission channel between two programs. These two programs may be running on the same computer or different machines separated by thousands of miles. Connection-oriented means a connection is established between the source and destination machines before any data is sent. Reliable means that data sent is guaranteed to reach its destination in the order sent or an error will be returned to the sender.

Whereas the IP address in an IP datagram specifies a particular machine on the Internet, the source port and destination port fields specify a particular process on that machine. This allows messages to be sent to specific applications running on a specific machine.

A.5 The Application Layer [Mer97]

This layer consists of a number of application protocols, that send its data to the transport layer (TCP or UDP), the following are some of them.

A.5.1 FTP

File Transfer Protocol is used for transferring documents and binary files. FTP uses TCP port 21 for initiating and controlling connections, and TCP port 20 for data transfer.

A.5.2 Telnet

It is an application that allows users to log in to (telnet to) a machine and use the equivalent of a direct console or terminal. Telnet uses TCP port 23.

A.5.3 Gopher

Gopher is a precursor to HTTP that organizes and presents information in a text-based menu format. Gopher has been largely obviated by HTTP. Gopher uses TCP port 70.

A.5.4 Finger

The Finger Protocol is used to find out information about users on a particular host, it can also provides information about currently logged-in users. Finger uses TCP port 79.

A.5.5 SMTP

Simple mail transfer protocol is the protocol used to transfer email between mail transport agents (MTA s) over the Internet. ESMTP is a new extended version of SMTP that supports additional commands for MTA communication. SMTP uses TCP port 25.

A.5.6 HTTP

Hypertext transfer protocol is the WWW protocol, and uses TCP port 80.

A.5.7 DNS

Domain Name Service which uses TCP port 53.

A.6 Internet Addresses

Every computer connected to the Internet has a unique address in the same way that every family in a given city can have a unique telephone number [Dav00].

IP addresses take the form X.X.X.X, where each X is one byte, such as 152.2.254.81. Every IP address falls into a network, which is a block of IP addresses grouped for administrative purpose [Mer97].

A.7 Domain Name System (DNS)

Most users never see or use IP addresses directly because the Domain Name System (DNS) provides a more meaningful and easier-to-remember name.

The host computer converts a DNS name to an IP address in the background, so you don't need to know the number [Fri97].

A DNS name is made up of a domain and one or more subdomains. For example, marauder.Millersv.edu uses the domain edu (educational institution) and has two subdomains, millersv and marauder.

The first subdomain is the name of the network (Millersv). The second subdomain, marauder, is the name of the computer system. If you read this address backward, it is the educational institution Millersville University, using the Marauder computer. There can be more than two subdomains [Fri97].

Appendix A: The Internet

DNS performs two functions. It provides lookup services, or name resolution, to hosts that are trying to find the IP number of a given hostname. It also provides the database that defines these mappings. Both of these functions are provided by name servers, which are hosts that provide name resolution services [Mer97].

Appendix B

CGI Library

This library is implemented using Visual Basic 6.0 for the proposed Web site search engine.

Option Explicit

'API Functions

Declare Function GetStdHandle Lib "kernel32" _

(ByVal nStdHandle As Long) As Long

Declare Function ReadFile Lib "kernel32" _

(ByVal hFile As Long, _

lpBuffer As Any, _

ByVal nNumberOfBytesToRead As Long, _

lpNumberOfBytesRead As Long, _

lpOverlapped As Any) As Long

Declare Function WriteFile Lib "kernel32" _

(ByVal hFile As Long, _

ByVal lpBuffer As String, _

ByVal nNumberOfBytesToWrite As Long, _

lpNumberOfBytesWritten As Long, _

lpOverlapped As Any) As Long

'Environment Variables

Public CGI_Accept As String

Appendix B: CGI Library

Public CGI_AuthType As String
Public CGI_ContentLength As String
Public CGI_ContentType As String
Public CGI_GatewayInterface As String
Public CGI_PathInfo As String
Public CGI_PathTranslated As String
Public CGI_QueryString As String
Public CGI_REFERER As String
Public CGI_RemoteAddr As String
Public CGI_RemoteHost As String
Public CGI_RemoteIdent As String
Public CGI_RemoteUser As String
Public CGI_RequestMethod As String
Public CGI_ScriptName As String
Public CGI_ServerSoftware As String
Public CGI_ServerName As String
Public CGI_ServerPort As String
Public CGI_ServerProtocol As String
Public CGI_UserAgent As String

'Length of incoming Data

Public lContentLength As Long

'Form's data of the client

Public formData As String

'Record to hold pair values

Type pairsRecord

 Name As String

 Value As String

End Type

'Array to hold pairs and Variable to hold number of pairs

Public pairs() As pairsRecord

Appendix B: CGI Library

Public pairsNo As Long

'Standard Input AND Standard Output

Public Const STD_INPUT_HANDLE = -10&

Public Const STD_OUTPUT_HANDLE = -11&

Public hStdIn As Long

Public hStdOut As Long

Sub Main()

 Initialize

 GetFormData

 CGI_Main

End Sub

Sub Initialize()

 pairsNo = 0

 hStdIn = GetStdHandle(STD_INPUT_HANDLE)

 hStdOut = GetStdHandle(STD_OUTPUT_HANDLE)

 CGI_Accept = Environ("HTTP_ACCEPT")

 CGI_AuthType = Environ("AUTH_TYPE")

 CGI_ContentLength = Environ("CONTENT_LENGTH")

 CGI_ContentType = Environ("CONTENT_TYPE")

 CGI_GatewayInterface = Environ("GATEWAY_INTERFACE")

 CGI_PathInfo = Environ("PATH_INFO")

 CGI_PathTranslated = Environ("PATH_TRANSLATED")

 CGI_QueryString = Environ("QUERY_STRING")

 CGI_Referer = Environ("HTTP_REFERER")

 CGI_RemoteAddr = Environ("REMOTE_ADDR")

 CGI_RemoteHost = Environ("REMOTE_HOST")

 CGI_RemoteIdent = Environ("REMOTE_IDENT")

 CGI_RemoteUser = Environ("REMOTE_USER")

 CGI_RequestMethod = Environ("REQUEST_METHOD")

 CGI_ScriptName = Environ("SCRIPT_NAME")

Appendix B: CGI Library

```
CGI_ServerSoftware = Environ("SERVER_SOFTWARE")
```

```
CGI_ServerName = Environ("SERVER_NAME")
```

```
CGI_ServerPort = Environ("SERVER_PORT")
```

```
CGI_ServerProtocol = Environ("SERVER_PROTOCOL")
```

```
CGI_UserAgent = Environ("HTTP_USER_AGENT")
```

```
lContentLength = Val(CGI_ContentLength)
```

```
End Sub
```

```
Sub GetFormData()
```

```
Dim tempBuffer As String
```

```
Dim bytesRead As Long
```

```
If CGI_RequestMethod = "POST" Then
```

```
tempBuffer = String(lContentLength, Chr(0))
```

```
ReadFile hStdIn, ByVal tempBuffer, lContentLength, bytesRead, ByVal 0&
```

```
formData = Left(tempBuffer, bytesRead)
```

```
GetPairs (formData)
```

```
ElseIf CGI_RequestMethod = "GET" Then
```

```
GetPairs (CGI_QueryString)
```

```
Else
```

```
'Put code for other requests like head
```

```
End If
```

```
End Sub
```

```
Sub GetPairs(data As String)
```

```
Dim pointer As Long
```

```
Dim position As Long
```

```
Dim length As Long
```

```
'If Data = "" Then Exit Sub
```

```
pointer = 1
```

```
Do
```

Appendix B: CGI Library

```
position = InStr(pointer, data, "=")
If position = 0 Then Exit Do
pairsNo = pairsNo + 1
ReDim Preserve pairs(1 To pairsNo)
pairs(pairsNo).Name = UCase(DecodeString(Mid(data, pointer, position - pointer)))
pointer = position + 1
position = InStr(pointer, data, "&")
If position = 0 Then
    length = Len(data)
    If pointer > length Then
        pairs(pairsNo).Value = ""
    Else
        pairs(pairsNo).Value = DecodeString(Mid(data, pointer, length))
    End If
    Exit Do
End If
pairs(pairsNo).Value = DecodeString(Mid(data, pointer, position - pointer))
pointer = position + 1
Loop
End Sub
```

```
Function EncodeString(data As String) As String
```

```
    Dim pointer As Long
```

```
    Dim position As Long
```

```
    'convert all (" ") to encoded (spaces)
```

```
    pointer = 1
```

```
    Do
```

```
        position = InStr(pointer, data, " ")
```

```
        If position = 0 Then Exit Do
```

```
        data = Mid(data, pointer, position - pointer) & "%20" & Mid(data, position + 1)
```

```
        pointer = position + 3
```

```
    Loop
```

Appendix B: CGI Library

```
EncodeString = data
```

```
End Function
```

```
Function DecodeString(data As String) As String
```

```
Dim pointer As Long
```

```
Dim position As Long
```

```
'convert all "+" to " " (space)
```

```
pointer = 1
```

```
Do
```

```
    position = InStr(pointer, data, "+")
```

```
    If position = 0 Then Exit Do
```

```
    Mid(data, position, 1) = " "
```

```
    pointer = position + 1
```

```
Loop
```

```
'Convert all %dd to character
```

```
pointer = 1
```

```
Do
```

```
    position = InStr(pointer, data, "%")
```

```
    If position = 0 Then Exit Do
```

```
    Mid(data, position, 1) = Chr("&H" & Mid(data, position + 1, 2))
```

```
    data = Left(data, position) & Mid(data, position + 3)
```

```
Loop
```

```
DecodeString = data
```

```
End Function
```

```
Function GetControlValue(ControlName As String, Value As String) As Boolean
```

```
Dim co As Long
```

```
If pairsNo = 0 Then
```

```
    GetControlValue = False
```

```
Exit Function
```

Appendix B: CGI Library

```
End If
    ControlName = UCase(ControlName)
For co = 1 To pairsNo
    If ControlName = pairs(co).Name Then
        Value = pairs(co).Value
        GetControlValue = True
        Exit Function
    End If
Next co
End Function
```

```
Sub SendData(data As String)
    Dim lpNumberOfBytesWritten As Long

    data = data & vbCrLf
    WriteFile hStdOut, data, Len(data), lpNumberOfBytesWritten, ByVal 0&
End Sub
```

```
Sub SendStatusOK()
    'HTTP Status header
    SendData "Status: 200 OK"
    'HTTP Content type header
    SendData "Content-type: text/html" & vbCrLf
End Sub
```

```
Sub SendThunderHed()
    'Head of the results page
    SendData "<HTML><HEAD><TITLE>Results"
    SendData "</TITLE></HEAD><BODY>"
    'Thunder Search Engine Shape
    SendData "<Center><Img Src=sthunder.jpg></Center>"
    SendData "<Form Name=MainForm Action=/cgi-bin/Search.exe Method=Get>"
    SendData "<Input Type=Text Name=ST size=50 " & "Value=" & hst & ">"
    SendData "<Input Type=Submit Name=Search Value=Search>"
End Sub
```


Appendix B: CGI Library

```
SendData "<Input Type=Hidden Name=hst Value=" & hst & ">"
End Sub

Sub SendError(errorTxt As String)
SendStatusOK
'Head of the results page
SendData "<HTML><HEAD><TITLE>Results"
SendData "</TITLE></HEAD><BODY>"
'Thunder Search Engine Shape
SendData "<Center><Img Src=sthunder.jpg></Center>"
SendData "<Form Name=MainForm Action=/cgi-bin/Search.exe Method=Get>"
SendData "<Input Type=Text Name=ST size=50 " & "Value="" & ">"
SendData "<Input Type=Submit Name=Search Value=Search>"
SendData "<Input Type=Hidden Name=hst Value="" & ">"

SendData "<Input Type=Hidden Name=hfp Value=" & 1 & ">"
SendData "<Input Type=Hidden Name=hnp Value=" & hnp & ">"
SendData "<Input Type=Hidden Name=htype Value=1>"
SendData "</Form>"
SendData "<HR>"
SendData "<Center><H2>ERROR</H2></Center>"
SendData "Text: " & hst & "<Br>"
SendData errorTxt
End Sub
```

Certification of the Examination Committee

We chairman and members of the examination committee, certify that we have studied the thesis entitled (**Development of a Web Site Search Engine**) presented by the student **Eihab Ahmed Muhammed Shaker Murjan** and examined him in its contents and in what is related to it, and we have found it worthy to be accepted for the degree of Master of Science in Computer Science with grade **Very Good**.

Signature:

Name: **Dr. Imad H. Al-Hussaini**

Title: **Assistant Professor**

Date: / / **2005**

(Chairman)

Signature:

Name: **Dr. Jamal M. Al-Ethawie**

Title: **Lectuerer**

Date: / / **2005**

(Member)

Signature:

Name: **Dr. Ahmed Tariq**

Title: **Assistant Professor**

Date: / / **2005**

(Member)

Signature:

Name: **Dr. Moaid A. Fadhil**

Title: **Senior Rreasercher**

Date: / / **2005**

(Supervisor)

Signature:

Name: **Dr. Ban N. Al-kallak**

Title: **Lectuerer**

Date: / / **2005**

(Supervisor)

Signature:

Name: **Dr. Laith Abdul Aziz Al-Ani**

Title: **Dean of College of Science**

Date: / / **2005**

Supervisor Certification

We certify that this thesis was prepared under our supervision at the Department of Computer Science/College of Science/ Al-Nahrain University, by **Eihab Ahmed Mohammed Shakir** as partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Signature:

Name: **Dr. Moaid A. Fadhil**

Title: **Senior Rreasercher**

Date: / / **2005**

Signature:

Name: **Dr. Ban N. Al-kallak**

Title: **Lectuerer**

Date: / / **2005**

In view of the available recommendations, I forward this thesis for debate by the examination committee.

Signature:

Name: **Dr. Taha S. Bashaga**

Title: **Head of the Department of Computer Science, Al-Nahrain
University.**

Date: / / **2005**

Chapter One

Introduction

1.1 Introduction

The Web creates new challenges for information retrieval. The amount of information on the Web is growing rapidly, as well as the number of new users inexperienced in the art of Web research. People are likely to surf the Web using its link graph, often starting with high quality human maintained indices such as *Yahoo!* or with search engines. Human maintained lists cover popular topics effectively but are subjective, expensive to build and maintain, slow to improve, and cannot cover all esoteric topics. Automated search engines that rely on keyword matching usually return too many low quality matches. To make matters worse, some advertisers attempt to gain people's attention by taking measures meant to mislead automated search engines [SL98].

So a real need exist for a search engine that can provide high quality search results.

1.2 The Internet

The Internet initiated a communications revolution where millions of users send messages, check live video cameras, read magazines, newspapers, participate in discussion groups spread around the world, and watch video

news segments as routinely as most of people turn on a television or talk on the telephone.

It is the Internet that bridges time, distance, and culture. It is where you can learn about almost any subject and communicate with almost anyone almost instantly [Fri96].

1.3 The Physical Internet [Fri97]

A network is a collection of computers linked together to achieve some common goal. In most cases, networks allow users to share information. In many businesses, networks enable one computer to send messages or get information from another computer. Conceptually, the Internet is no different from any other network, except it is bigger.

The Internet is a network of networks linked using very precise rules that allow any computer to connect to and exchange information with other networks or computers connected to the Internet. Connecting to the Internet means connecting to tens of thousands of other networks, millions of individual computers, and tens of millions of other computer users.

1.4 The Soft Internet [Fri97]

The Internet is a system that uses a communication language, called a **protocol**, to enable one computer network to speak to another. During the 1980s a new communication language emerged called **Transmission Control Protocol/Internet Protocol (TCP/IP)**. TCP/IP became a standard for the Internet. In short, any computer network could communicate with any other computer network as long as they both used the TCP/IP standard protocol.

The **Internet Society** is a voluntary organization and is not run by the government or by any individual. Rather, it is a board that meets to set

standards and determine resources. For example, it is the Internet Society, through the **Internet Architecture Board (IAB)**, that determines addresses for users, as well as the rules for accessing and using these addresses. For more information about the internet, see Appendix A.

1.5 The World Wide Web

The World Wide Web (**WWW**), was invented by *Tim Berners-Lee* in late 1990 while he was working at *CERN*, the *European laboratory for particle physics*. The WWW is a distributed hypermedia environment consisting of documents from around the world. The documents are linked using a system known as **Hypertext**, where elements of one document may be linked to specific elements of another document. The documents may be located on any computer connected to the Internet. In this context, the world “document” is not limited to text but may include video, audio, graphics, databases, and a host of other tools that can be accessed from any Web browser [Dav00].

These documents are created with a special language called **Hypertext Markup Language (HTML)**. This language allows the full use of hypermedia including text, images, graphics, sounds and other types of multimedia. Because HTML is a special language, it requires special software to access the Web. This type of access program is known as a **Browser** [Sab02].

The Internet has suffered for years from a reputation of being difficult to learn, hard to use. The WWW has changed all this. The Web has quickly become the graphical user interface to the Internet.

A Web site is a location on the WWW. Each Web site contains a **Home page**, which is the first document or Web page that users see when they enter the site. The site might also contain additional documents, files, or Web pages, which are sometimes called **Child pages** [Vir01].

1.6 Searching the Web

With so much data on the Internet, it can be difficult, frustrating, and seemingly impossible to find the exact information needed by the user [Bibf99].

Figure (1.1) [Bri00] present the growth of the Web according to a study presented by *Cyveillance, Inc.* Cyveillance make predictions regarding future growth of the Internet based on data collected over an eight-month period of time.

Number of unique pages on the Internet: 2.1 billion
Unique pages added per day: 7.3 million

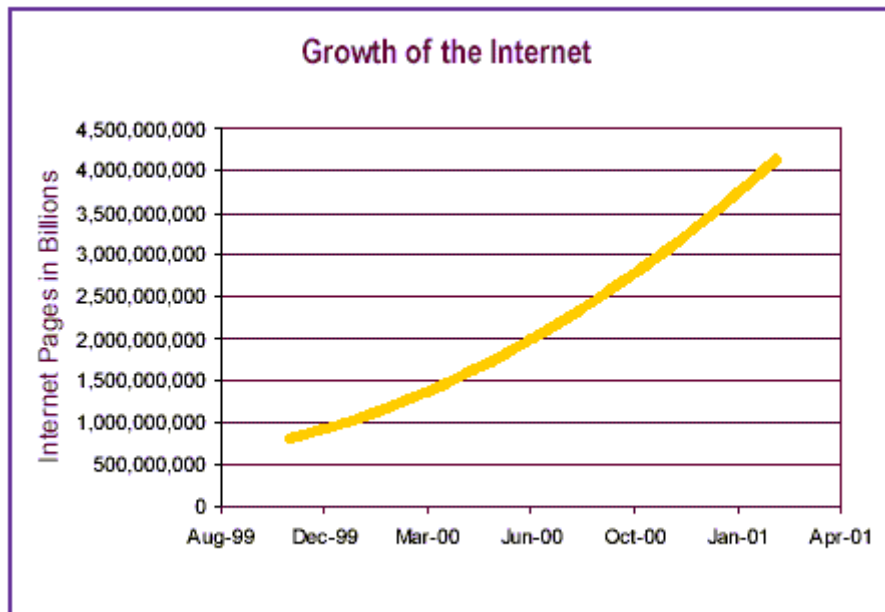


Figure (1.1) Size and Growth of the Internet

The Web is enormous and growing at an incredibly fast pace. It has been said that if the user spent only one minute per page, 10 hours a day, it would take four-and-a-half years to explore only 1 million Web pages. Thus a real need exists for some way to search this huge resource [Bibf99].

A few years ago a number of sites developed a means for locating information on the Web. These sites collect information on other Web sites then use a search engine (a piece of software for searching data) to locate

matches based on specific criteria. In addition, several of these sites provide categorical listing of relevant Web sites [Fri96].

1.6.1 Starting Points Pages

These pages provide a good point of entry for people accessing the Web for the first time.

They include lots of links that introduce the user to fundamental Web concepts, such as “What is a Browser?” the user can also find links to Subject Trees, Search Engines, Usenet **FAQs (Frequently Asked Questions)**, Internet documents of all kinds, and “What’s new” documents [Bry96].

1.6.2 Subject Trees [Bry96]

A subject tree is a subject-oriented catalog of URLs organized by topic. It is an alphabetically organized list of selected Web resources that is usually organized with major headings such as Arts and Humanities, Business, Economy, Government, and the like.

Within each category are found subheadings, which in turn display pages listing specific hyperlinks. Because subject trees are manually updated, they cannot hope to cover all of the Web. No subject tree is complete; it is a really big pain to keep one of these things updated, but they are very useful because they are selective; they list only those documents that would likely prove useful to the user.

1.6.3 Search Engine

It is a software used to help the Web user to search the Web for a specific information. According to a recent study sponsored by *Realnames* Corporation, 75% of frequent Internet users use search engines to navigate the Web [Trek01].

The search engine consists of the following parts:

1. **Crawler:** a program used to download the pages from the Web site and save it as files on the storage unit. But there are many practical complications in the crawler: sites may be busy or down during the crawling time, and come back to life later; pages may be duplicated at multiple sites; many pages have text that does not conform to the standards for HTML, HTTP redirection, robot exclusion; some information is hard to access because it is hidden behind a form, Flash animation or script program.
2. **Indexer:** A search engine's index is similar to the index in the back of a book: it is used to find the pages on which a word occurs [Isr05]. The indexer parse all the pages downloaded by the crawler and extract the words in the pages to build the index. The search engine index could be very large, so various of compression techniques could be used to reduce the size of the index.
3. **Ranker:** it is responsible for ranking the pages and decide which ones are most relevant to the user needs. There are many features that can be used to decide which pages contain the most relevant words required by the user such as (word font size, word font style, word font color, position of the word in the page).
4. **User Interface:** provide the interface as a Web page to the user with a text box and a search button. The user write his/her query in the text box and click on the search button, then a new page will be displayed containing the results of the required query.

1.6.4 Web Site Search Engine

Web site search engine has become increasingly important for the Web sites of education institutes, government and private companies, because it

provides more detailed information that general search engine usually can't offer.

Web site search engines are small compared with the Internet search engines, while the Internet search engine search the entire Web looking for answers to the user's query, the Web site search engine is dedicated to one Web site, searching only that Web site.

1.7 Literature Survey

- 1- Isra'a Tahseen Ali Al-Attar [Isr05], "Internet Search Engine Design": this work gives a description of the design of Internet search engine and concentrates on the ranking of the search engine which consists of two subsystems: a term-based ranking subsystem that assign a rank score to the Web pages using the TF-IDF (Term Frequency-Inverse Document Frequency) algorithm and Vector Spreading Activation algorithm and a link-based ranking subsystem that assigns a rank score to the Web pages depending on their contained hyperlinks by implementing an improved PageRank algorithm.
- 2- Jianlin Cheng [Jia00], "Design and implementation of ICS (Information and Computer Science) Web Search Engine": presents the design of a site-specific search engine called ICS Web search engine, with a crawler that download about 12,000 pages and a ranker using the Roberson and Sparck Jones TF-IDF method.
- 3- Sunny Lam [Sun01], "The Overview of Web Search Engines": this paper gives a description of the search engines and information about the Web characteristics and the difficulties of search in the Web. Also gives many crawler examples.
- 4- Sergey Brin and Lawrence Page [SL98], "Google": present Google, a prototype of a large-scale search engine which makes heavy use of the structure present in hypertext, it is designed to crawl and index the

Web efficiently and produce much more satisfying search results. This paper provide an in-depth description of the Google search engine.

- 5- Allan Heydon and Marc Najork [All99], “Mercator”: this paper presents Mercator a scalable extensible web crawler, which describes all the basics and traps related to designing a crawler and gives many statistics about the Web.
- 6- WiseNut [Wis01]: this paper gives a description of the WiseNut search engine that consists of a crawler (named Zyborg), indexer that extract keywords and link information, context-sensetive ranking system , and query server.

1.8 Aim of Thesis

The objective of this thesis is to develop a Web site search engine to be used by any Web site by crawling, indexing, and ranking all the Web pages in that Web site to create the database, in which any user can use the Web site search engine to find the information nedded for his/her query by searching the database and return a Web page contains a list of links to all the pages that contain information about the user query.

1.9 Thesis Layout

In this section, the contents of individual chapters of the thesis are briefly reviewed.

- **Chapter two:** covers the theoretical basis of Web searching with the parts of the search engine.
- **Chapter three:** presents the practical work of the Web site search engine.
- **Chapter four:** presents the programming languages used in the proposed Web site search engine, user interface, features, system

requirement, experiment and results and tools used in the operation of the proposed Web site search engine.

- **Chapter five:** presents the conclusions on this work, with recommendation for future work.

Chapter Two

Web and Search Engines

2.1 Introduction

This chapter starts by talking about the Client-Server model, Web servers and URL which explains how the Web works, then how to create Web pages using the Web language (HTML) and how to view them using the Browser, then the Web characteristics and difficulties are explained.

After describing the Web, the importance of information retrieval, it's relationship with the search engines and the difference between information retrieval and data retrieval are discussed.

Then the types of search engines are listed, and the problems of search engines are discussed.

The last parts of the chapter describe the search engine architecture in details and give examples for Internet and Web site search engines.

2.2 The Client-Server Model

The WWW, as with most of the services available today, is based on the long-standing client-server model illustrated in figure (2.1):

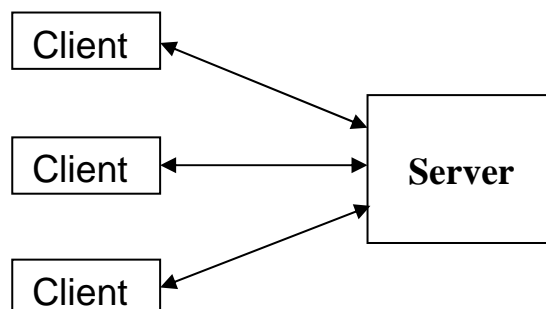


Figure (2.1) Client-Server Model

In this model there are normally many clients accessing a single server. One of the principal reasons for using the client-server model is the consolidation of resources in a central location. The server usually offers some service that many clients are interested in [Dav00].

2.3 Web Servers

Web server is a program that makes web pages available to people who are browsing the web [Bry96].

It is a program that monitors a particular port on the server computer and waits for incoming messages. Web server programs are specifically related to the WWW services that run on the Internet and by default listen to port 80.

These incoming messages are formatted according to the Hypertext Transfer Protocol (HTTP). These HTTP messages request information residing on the Web server's computer.

A Web server provides four major functions:

- Serving Web pages.
- Running gateway programs and returning their output.
- Controlling access to the server.
- Monitoring and logging server access statistics [Ric96].

If the request is for a simple static document that resides on the Web server computer, the Web server will retrieve the document and return it to the client that requests it. [Dav00]

The request may be for a static document that contains some embedded directives that must first be translated by the Web server before returning the document. [Dav00]

Lastly, the request may be to execute a CGI (Common Gateway Interface) program. The Web server will execute the specified CGI program and make the data passed from the web client available to the CGI program. When the

CGI program completes, the web server will pass back to the client any information that was returned by the CGI program [Dav00].

2.4 Uniform Resource Locator (URL)

URLs are used to point to resources of all types on the WWW, including HTTP documents.

A URL is made of **protocol part**, a **server designation** (Host name), a **port**, and a file (**path**) designation as shown below.

http :// www.nsa.gov :[80] /index.html

The diagram shows the URL 'http :// www.nsa.gov :[80] /index.html' with four horizontal lines underneath it. Below each line is a downward-pointing arrow. Under the first arrow is the label 'Protocol', under the second is 'Host name', under the third is 'Port', and under the fourth is 'Filename'.

Protocol Host name Port Filename

Each page on the Web has it's own address, which is it's URL.

2.5 The HTML Language

Back in the early 80's, IBM had a pretty good idea that was a little ahead of its time. This idea was to create documents with titles, addresses, headings, body text, and other elements that are all very similar from one document to the next.

The avenue that IBM explored was to develop a sort of pseudo-computer language that combines plain text and formatting instructions. Such a language is called a markup language, and IBM called its version **Generalized Markup Language**, or **GML**. IBM never did much with GML, so the **International Standards Organization (ISO)** derived its language from IBM's GML but called the new language **SGML**, for **Standard Generalized Markup Language**.

SGML defines many different types of documents. One of these documents was the hypertext document. HTML, which stands for Hypertext

Markup Language, is the subset of SGML that defines hypertext documents. People use the term HTML to refer to both the hypertext document itself (which is a specific type of SGML document) and the markup language that you use to create a hypertext document [Ric96].

Here is the HTML code for a simple Web page:

```
<HTML>  
<HEAD> <TITLE>Simple Web Page</TITLE> </HEAD>  
<BODY>  
    This is a simple HTML Web page  
</BODY>  
</HTML>
```

2.6 Browser

A Web browser is a **window** onto the WWW. With Web browsers, you can view Web documents containing integrated or linked graphics, or even video and audio clips. Browsers also can navigate **FTP (File Transfer Protocol)** sites and retrieve software and data files, read **Usenet** newsgroups, and send **E-Mail** messages [Ric96]. Figure (2.2) shows a browser.

The browser consists of the following main parts:

- 1- Window body: the contents of the Web page will be displayed in this part of the browser.
- 2- Title bar: the title of the Web page will be displayed in this part of the browser.
- 3- Address bar: in this part the user can write the address (URL) of the required Web page, and when the user press enter, the browser will connect to the server containing that page, request the page, and download it from the server to the client computer and display it in the

window body of the browser and display the page title in the title bar of the browser.

- 4- Toolbar: which contains a number of buttons such as: home, refresh, stop, back and next.

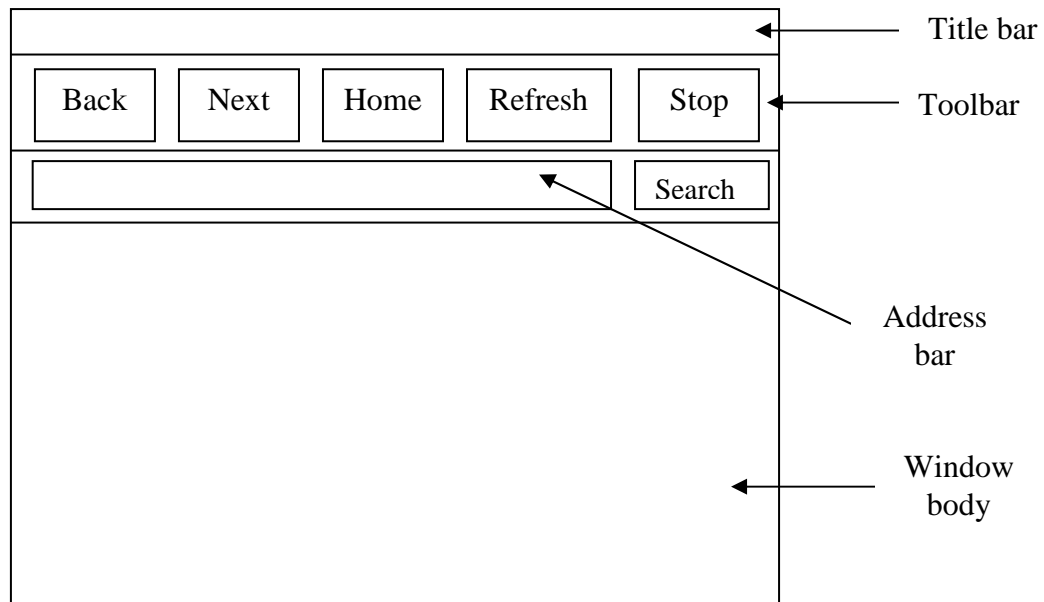


Figure (2.2) Browser

2.7 Characteristics of the Web

The most popular formats of Web documents are HTML, followed by GIF (Graphic Interchange Format) and JPG (Joint Photographic Group) (images format), ASCII files, Postscript and ASP (Active Server Page). Most HTML pages are not standard, because they do not comply with HTML specifications. HTML documents seldom start with a document type definition. Also they are typically small with an average of 5 KB and a median of 2 KB [Sun01].

On average, each HTML page contains one or two images and five to fifteen hyperlinks. Most of these hyperlinks are local, meaning the associated Web pages are mostly stored in the same Web server [Sun01].

2.8 Difficulties of the Web

The Web creates new challenges for the information retrieval. The amount of information on the Web is growing rapidly as well as the number of new users inexperienced in the art of Web research [SL98].

The Web has become increasingly commercial over time, from 1.5% of .com domain in 1993 to over 60% in 1997. At the same time, search engine development has moved from the **academic domain** to the **commercial domain**. Today, most search engine developments take place in companies without technical information to the public. Therefore, it is very difficult to study today's search engines [Sun01].

There are many problems for searching information on the Web, which can be divided into two classes: The first class contains problems with the **data** itself. The second class contains problems regarding how **users** use the information retrieval system.

The problems of the first class are:

- **Distributed data:** data is distributed widely in the world. It is located at different sites and platforms. The communication links between computers vary widely. Plus, there is no topology of data organization.
- **High percentage of volatile data:** documents can be added or removed easily in the World Wide Web. Changes to these documents go unnoticed by others. 40% of Web pages change every month. There is a very high chance of dangling links [Sun01].
- **Large volume:** the growth of data is exponential. It poses issues that are difficult to cope with.
- **Unstructured and redundant data:** the Web is not exactly a distributed hypertext. It is impossible to organize and add consistency to the data and the hyperlinks. Web pages are not well structured and

30% of all Web pages are duplicated. Semantic redundancy can increase traffic [Sun01].

- **Quality of data:** a lot of Web pages do not involve any editorial process. That means data can be false, inaccurate, outdated, or poorly written.
- **Heterogeneous data:** data on the Web are heterogeneous. They are written in different formats, media types, and natural languages.
- **Dynamic data:** the content of Web document changes dynamically. The content can be changed by a program such as hit counter that keep tracks of number of hits.
- **Low quality search engines results:** automatic search engines that rely on keyword matching usually return too many low quality matches. To make matters worse, some advertisers attempt to gain people's attention by taking measures meant to mislead automated search engines.

The second class of problems deals with interaction between the user and the search engine, there are two basic problems:

- 1- **How to specify a query:** the user needs to know how to pose a query, so that the results contain relevant information. Here are some of these problems
 - Many users do not understand how to provide a sequence of words for the search.
 - The users may get unexpected answers because the user is not aware of the input requirement of the search engine. For example, some search engines are case sensitive.
 - Many users do not understand Boolean logic: therefore, the user cannot perform advanced searching.

- Novice users do not know how to start working with search engines.

2- How to interpret the results: the user needs to know how to select the documents from the results.

Around 85% of users only look at the first page of the results, so relevant documents might be skipped [Sun01].

2.9 Information Retrieval

Before the search engine designer can understand search engines, the search engine designer needs to understand **Information Retrieval (IR)**, because Web searching is within the field of Information Retrieval [Sun01].

Search engine is the popular term for Information Retrieval systems. Since the 1940s, the problem of information storage and retrieval has attracted an increasing attention. It is simply stated: there is a vast amounts of information to which accurate and speedy access is becoming ever more difficult [Sab02].

Before the Internet was born, information retrieval was just index searching. For example authors, title, and subjects in library card catalog or computers. Today, among other things, IR includes modeling, document classification and categorization, systems architecture, user interface, data visualization, filtering, and languages. IR deals with the representation, storage, organization of, and access to information items [Sun01].

IR aims to provide fast, effective, and efficient methods of representing, managing, searching, retrieving and presenting such information. Suppose there is a store of documents and a person (user of the store) have a question (request or query) to which the answer is a set of documents satisfying the information needed by the question. The person can obtain the set by reading all the documents in the store, retaining the relevant documents and discarding all the others. In a sense, this constitutes perfect retrieval. This

solution is obviously impracticable. A user either does not have the time or does not wish to spend the time reading the entire document collection, a part from the fact that it may be physically impossible to do so.

When high-speed computers became available for **non-numerical work**, many thought that a computer would be able to read an entire document collection to extract the relevant documents. It soon became apparent that using the natural language text of a document not only caused input and storage problems, but also left unsolved the intellectual problem of characterizing the document content. But automatic characterization in which the software attempts to duplicate the human process of reading was very sticky problem indeed. More specifically, reading involves attempting to extract information, both **syntactic** and **semantic**, from the text and using it to decide whether each document is relevant or not to a particular request. The difficulty is not only knowing how to extract the information but also how to use it to decide relevance.

The automatic retrieval strategy is used to retrieve all the relevant documents at the same time retrieving as few of the non-relevant as possible.

In IR, the kind of search used is not the usual kind where the result of the search is clear-cut, either yes, the item is present, or no, the item is absent. IR is interested in search strategies in which the documents retrieved may be more or less relevant to the request.

2.10 Information Retrieval and Data Retrieval

Many users may not be able to distinguish between Data Retrieval (DR) and Information Retrieval (IR).

There is a difference between IR and DR. In DR, the result of a query must be accurate: it should return the exact match tuples of the query, no more and no less. If there is no change to the database, the result of a query executed at different times should be the same. On the other hand, IR can be

inaccurate as long as the error is insignificant. The main reason for this difference is that IR usually deals with natural language text, which is not always well structured and could be semantically ambiguous. DR deals with data that has a well-defined structure and semantics (e.g. a relational database). In addition, DR cannot provide a solution given a subject or topic but IR is able to do so [Sun01].

2.11 Types of Search Engines [Art05]

There are three main types of search engines:

-Directory-Based Search Services

For Directory-Based Search Services, the primary frame of reference is the subject matter. The site listings are compiled and reviewed manually. For example, Yahoo, the best-known Internet Directory, dedicates staff to review and categorize site suggestions and then adds them to a specific directory on Yahoo. The directory structure is hierarchical and starts with a general subject heading such as dentistry. Successive sub headings are more specific. For example under Dentistry topical areas such as dental implants, fluoridation, amalgam, organizations, orthodontics, and tooth whitening are included. These databases are comparatively small and the frequency of the updating is relatively low. Examples of Directory-Based Search Services are: Infoseek, Magellan.

-True Search Engine

In contrast, the unmanned Search Engine completely automates the process of indexing the sites and totally removes the human component. A software robot called a spider or crawler gathers sites from across the web as it scans pages and connects to associated links. One particular advantage is

that the spider will automatically return to the same site periodically to check for new content or new pages.

The results from this "spidering" are then saved in the engines index and serve as the basis to orient each query. Given the automation process and the size of the Internet, these indices grow to upwards of 250 to 500 million pages. These efficiencies enable the search engine to cover a wide variety and number of sites that are maintained current through regular visits by the robot. The information may not be as exacting and the quantities may be voluminous compared to a directory-based site. Examples of Search Engines are: Google, Altavista

-Hybrid Search Engines

Some search engines also maintain an associated directory. To be included in a search engine's directory. Examples are: Yahoo.

2.12 The problem with today's Search Engines

If the users type a few keywords in most of today's search engines they will retrieve pages that, in principle at least, are relevant to their query. This often results in the retrieval of thousands of Web pages that are related to the query only by the fact that the keyword appears somewhere on each of those pages.

For people with slow Internet connections, this quantity of information is overwhelming. Most are unwilling to wait for pages and pages of results to download. Those that do wait are left with the task of sorting through results one-by-one to find the information they want.

Frustrated Internet searchers will abandon sites that don't offer a satisfactory search experience, according to surveys. *Berrier Associates* found that 44 percent of users say they are frustrated with search engine use

and where unable to find what they are looking for, most users will try another search engine [Wis01].

Because simple keyword matching often returns thousands of results, most search engines sort pages based on where (position) the keyword appears on the page, giving more weight to pages in which the keyword appears in the title, at the top of the page or in the Meta tag - the HTML code that describes the content of a Web page. One such sorting method is known as "collection frequency weighting".

2.13 Architecture of Search Engine

The search engine usually consists of the following parts:

- 1- Crawling Part: in which all the Web pages are downloaded from the Web site and saved as files in the storage unit.
- 2- Indexing Part: in this part all the Web pages saved by the crawler are parsed to extract all the words to create the index.
- 3- Ranking Part: in which all the words in the Web pages are assigned a specific score.
- 4- User Interface Part: this part is responsible for interacting with the user and provide the interface required for collecting the user query and provide the results for the user.
- 5- Searching Part: this part is responsible for searching the lexicon for the user query and returns the results to the user.

Figure (2.3) shows the architecture of search engine.

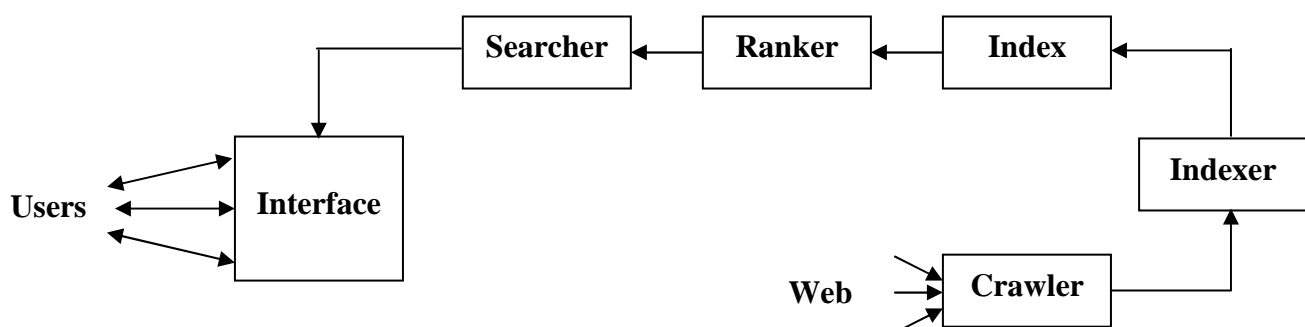


Figure (2.3) Search Engine Architecture

2.14 Crawling Part

The first step to create the search engine is to download all the Web pages of the Web site into the storage unit in the computer where the search engine will work on (Server).

Web crawling is a process to collect all the Web pages that are interested to search engine [Jia00].

Crawlers are also called robots, spiders, worms, wanderers, walkers, and knowbots. The first crawler, *Wanderer* was developed by *Matthew Gray* in 1993. Due to the competitive nature of the search engine business, the designs of these crawlers have not been publicly described. There are several crawling techniques available in public. The simplest one is to start with a set of URLs and from that extracts other URLs recursively in breadth-first or depth-first manner [Sun01].

The crawler consists of a URL list, downloading program and a URL extractor. Before running the crawler, the crawler designer will have to initialize the URL list with some URLs (Home page and other frequently requested URLs). The downloading program contains a pointer that points to one of the links in the URL list, which will be initialized to zero. When the crawler start running, the downloading program will increase the pointer by one and check if there is a link at the pointed position in the URL list. If there is a link, then the downloading program will request to download that link from the Internet and save it as a file on the storage unit. If there was no link then this means that there are no new links in the URL list, which is the end of the crawling process. After downloading a page from the Internet, it will be send to the URL extractor, which will extract all the links inside it and these links will be added to the URL list, but without duplicates (only the new links are added). The crawler complete it's work when there will be no more new links to be added to the URL list and the pointer of the

downloading program cross the last location in the URL list. Figure (2.4) shows the whole process:

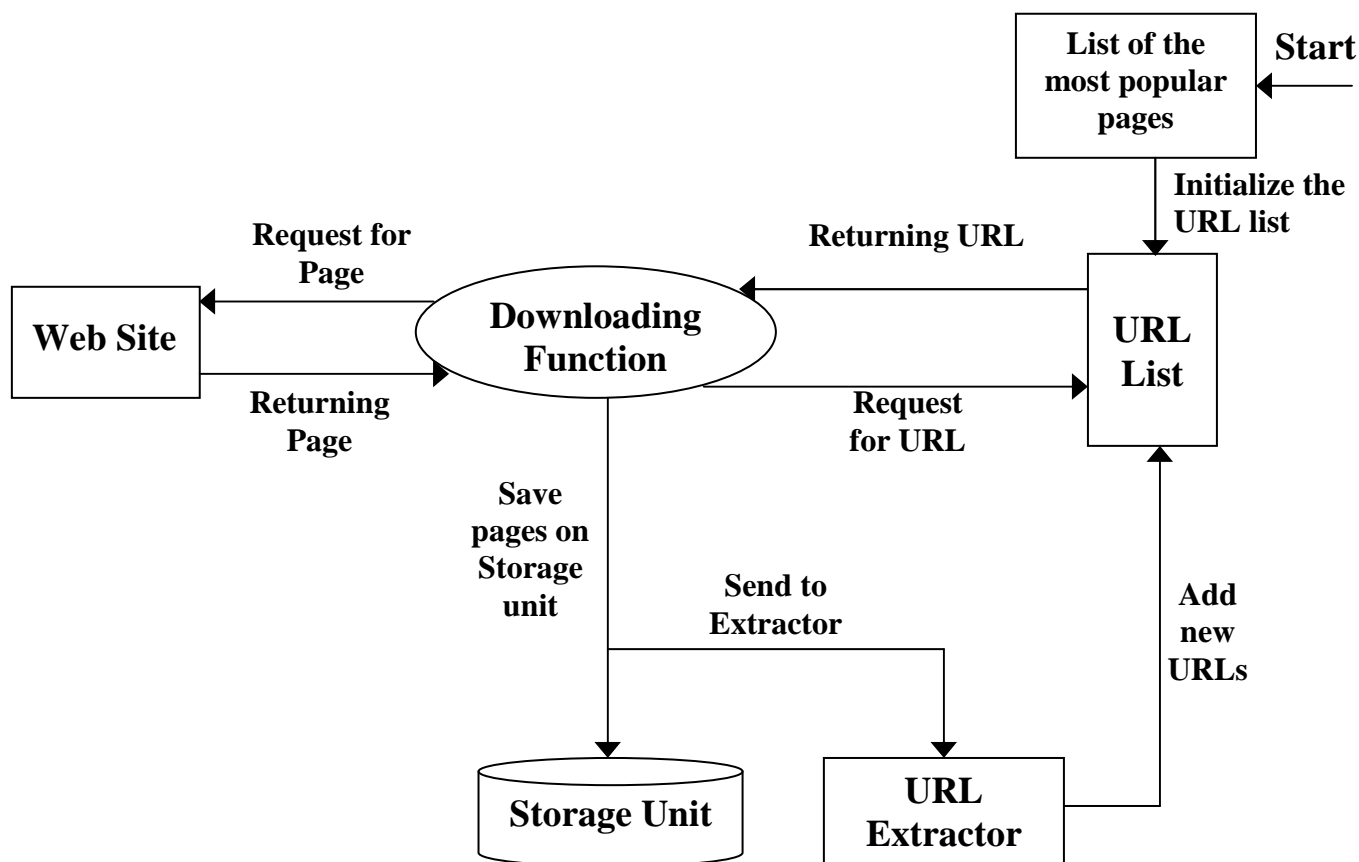


Figure (2.4) Crawling process

Running a Web crawler is a challenging task. There are tricky performance and reliability issues and even more importantly, there are social issues [SL98].

The design of a good crawler presents many challenges. Externally, the crawler must avoid overloading Web sites or network links as it goes about its business. Internally, the crawler must deal with huge volumes of data. Unless it has unlimited computing resources and unlimited time, it must carefully decide what URLs to scan and in what order [JHL98].

One of the problems that the crawler needs to cope with is that Web pages change dynamically, so the page that the index points to may not exist

anymore. Many search engines keep track of date, and shows the date to the query result.

2.14.1 Crawling Techniques

There are two policies used to traverse Web pages. The first one is **breadth-first policy**. It looks at all the pages linked by the current page and so on. The coverage will be wide but shallow. This may cause the Web server to have many rapid requests. The second is **depth-first policy**. We follow the first link of a page and we do the same on that page until we cannot go deeper. After that, it returns recursively. The advantage of using depth-first search is deep and space complexity is cheaper. But the disadvantage of using it is narrow.

Consider the Web site shown in figure (2.5)

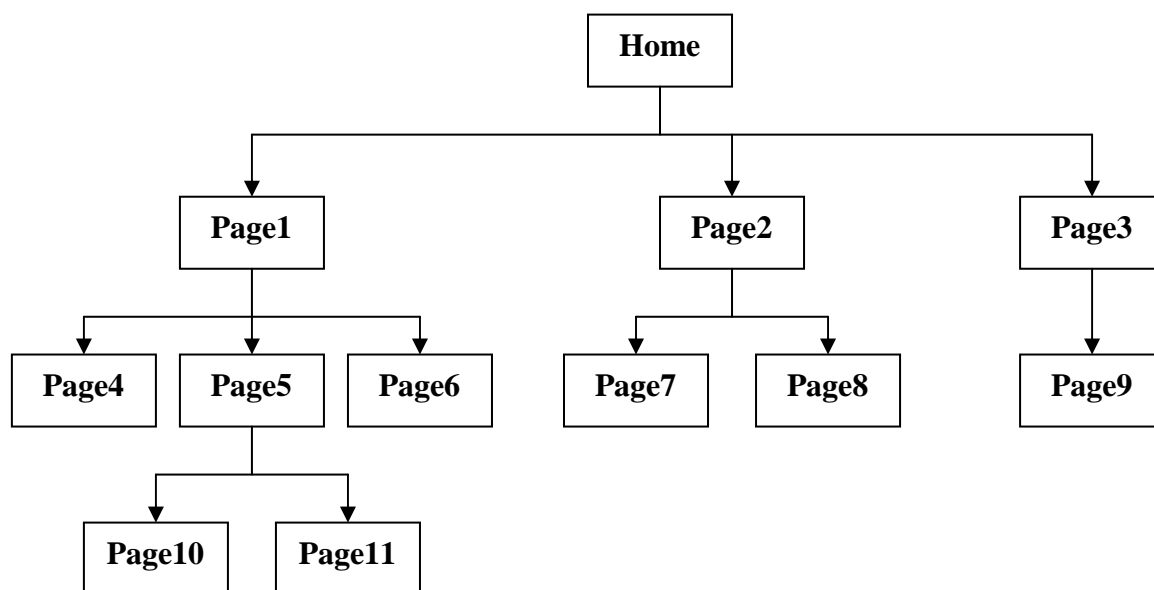


Figure (2.5) Website example

If the crawler is using a breadth-first policy then the pages will be downloaded in the order (Home, Page1, Page2, Page3, Page4, Page5, Page6, Page7, Page8, Page9, Page10, Page11).

If the crawler is using a depth-first policy then the pages will be downloaded in the order (Home, Page1, Page4, Page5, Page10, Page11, Page6, Page2, Page7, Page8, Page3, Page9).

2.14.2 Crawler Types

Crawlers are widely used today and there are many types of crawlers, the following are some of them:

- 1- Crawlers for the major search engines like AltaVista, InfoSeek, Excite, and Lycos attempt to visit most text pages, in order to build content indexes.
- 2- Crawlers that visit many pages looking for certain types of information (e.g. email addresses).
- 3- Personal crawlers that scan for pages of interest to a particular user.

2.15 Indexing Part

Indexing part is the most complicated and critical step in building the search engine. Simply, the indexer is used for parsing the web documents downloaded by the crawler to extract words and any features about these words (such as position in the page, font style, font color, and font size) to build the index (database) of the search engine.

Most indices use variants of inverted index files and lexicons. An inverted index file is a list of sorted words and their features; the inverted index can be created by sorting the index by word IDs.

The lexicon is a list of sorted words with pointers to the inverted index file. To find a word, the lexicon will be searched for that word and the results can be found from the inverted index file at the location pointed by the pointer in the lexicon list.

Consider a Website consisting of 3 pages (Page1, Page2, and Page3) as shown in figure (2.6). Each page contains 3 words as shown in table (2.1.a), table (2.1.b), and table (2.1.c). Each table contains the words that occur in the page with their positions.

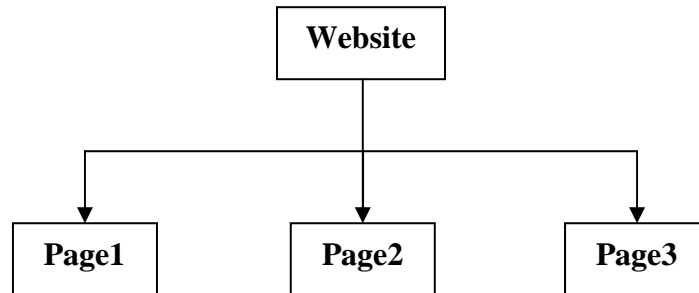


Figure (2.6) Simple Website

Table (2.1.a) Page1

Word	Position
Car	1
Train	5
Plane	11

Table (2.1.b) Page2

Word	Position
Home	1
Table	6
Car	12

Table (2.1.c) Page3

Word	Position
Table	1
Car	7
Plane	11

To index this Website, two tables are needed; table (2.2) for page IDs and table (2.3) for word IDs.

Table (2.2) Page IDs

Page Name	Page ID
Page1	P1
Page2	P2
Page3	P3

Table (2.3) Word IDs

Word	Word ID
Car	W1
Train	W2
Plane	W3
Home	W4
Table	W5

Then the index will be as shown in table (2.4)

Table (2.4) Index

Page ID	Word ID	Position
P1	W1	1
P1	W2	5
P1	W3	11
P2	W4	1
P2	W5	6
P2	W1	12
P3	W5	1
P3	W1	7
P3	W3	11

The inverted index is created by sorting the index by word IDs and position as shown in table (2.5)

Table (2.5) Inverted Index

Record Number	Word ID	Page ID	Position
1	W1	P1	1
2	W1	P3	7
3	W1	P2	12
4	W2	P1	5
5	W3	P1	11
6	W3	P3	11
7	W4	P2	1
8	W5	P3	1
9	W5	P2	6

The lexicon is the data structure used to search the Website will be as shown in table (2.6)

Table (2.6) Lexicon

Word ID	Number of pages	Pointer to inverted index
W1	3	1
W2	1	4
W3	2	5
W4	1	7
W5	2	8

To search for the word car, use table 2.3 to get the word ID, which is W1. By searching the lexicon for W1, the results will be 3 pages containing W1

and these pages found in the inverted index (table 2.5) starting from record number 1 to record number 3.

2.15.1 Indexing Steps

The indexing part consists of a number of steps that are different from search engine to another, but here are the general steps that can be found in most of the search engines:

- **Parsing:** In this step, each Web page is parsed into pure text without html tags. The pure text of each web page is used as the document to match against the user's query in the search part. In some search engines (as in the ICS Web search engine [Jia00]) the title of the Web pages is extracted out and used as the description of the Web link in the hitlist returned to user.
- **Deleting stop words:** This step helps save system resources (such as storage in hard disk and memory) by eliminating from further processing, as well as potential matching, those terms that have little value in finding useful documents in response to a customer's query. This step used to matter much more than it does now when memory has become so much cheaper and systems so much faster, but since stop words may comprise up to 40 percent of text words in a document, it still has some significance. A stop word list typically consists of those word classes known to convey little substantive meaning, such as articles (*a, the*), conjunctions (*and, but*), interjections (*oh, but*), prepositions (*in, over*), pronouns (*he, it*), and forms of the "to be" verb (*is, are*). To delete stop words, an algorithm compares index term candidates in the documents against a stop word list and eliminates certain terms from inclusion in the index for searching [Eli01]. A full list of stop words for general text is shown in table (2.7)

Table (2.7) Stop Word List [Wil03]

A	both	few	important	Much	parted	since	under
about	but	find	In	Must	parting	small	until
Above	by	finds	interest	My	parts	smaller	up
across	c	first	interested	Myself	per	smallest	upon
After	came	for	interesting	N	perhaps	so	us
Again	can	four	interests	Necessary	place	some	use
against	cannot	from	Into	Need	places	somebody	uses
All	case	full	Is	Needed	point	someone	used
almost	cases	fully	It	Needing	pointed	something	v
Alone	certain	further	Its	Needs	pointing	somewhere	very
Along	certainly	furthered	Itself	Never	points	state	w
already	clear	furthering	J	New	possible	states	want
also	clearly	furtheres	Just	Newer	present	still	wanted
although	come	g	K	Newest	presented	such	wanting
always	could	gave	Keep	Next	presenting	sure	wants
among	d	general	Keeps	No	presents	t	was
an	did	generally	Kind	Non	problem	take	way
and	differ	get	Knew	Not	problems	taken	ways
another	different	gets	Know	Nobody	put	than	we
any	differently	give	Known	Noone	puts	that	well
anybody	do	given	Knows	Nothing	q	the	wells
anyone	does	gives	L	Now	quite	their	went
anything	done	go	Large	nowhere	r	them	were
anywhere	down	going	Largely	number	rather	then	what
are	downed	good	Last	numbers	really	there	when
area	downing	goods	Later	o	right	therefore	where
areas	downs	got	Latest	of	room	these	whether
around	during	great	Least	off	rooms	they	which
as	e	greater	Less	often	s	thing	while
ask	each	greatest	let	old	said	things	who
at	early	group	lets	older	same	think	whole
away	either	grouping	like	oldest	saw	thinks	whose

b	end	groups	likely	on	say	this	why
back	ended	h	long	once	says	those	will
backed	ending	had	longer	one	second	though	with
backing	ends	has	longest	only	seconds	thought	within
backs	enough	have	m	open	see	thoughts	without
be	even	having	made	opened	sees	three	work
because	evenly	he	make	opening	seem	through	worked
become	ever	her	making	opens	seemed	thus	working
becomes	every	herself	man	or	seeming	to	works
became	everybody	here	many	order	seems	today	would
been	everyone	high	me	ordered	several	together	y
before	everything	higher	member	ordering	shall	too	year
began	everywhere	highest	members	orders	she	took	years
behind	f	him	men	other	should	toward	yet
being	face	himself	might	others	show	turn	you
beings	faces	his	more	our	showed	turned	young
best	fact	how	most	out	showing	turning	younger
better	facts	however	mostly	over	shows	turns	youngest
between	far	i	mr	p	side	two	your
big	felt	if	mrs	part	sides	u	yours

- **Term Stemming:** Stemming removes word suffixes, perhaps recursively in layer after layer of processing. The process has two goals. In terms of efficiency, stemming reduces the number of unique words in the index, which in turn reduces the storage space required for the index and speeds up the search process. In terms of effectiveness, stemming improves recall by reducing all forms of the word to a base or stemmed form. For example, if a user asks for *analyze*, they may also want documents which contain *analysis*, *analyzing*, *analyzer*, *analyzes*, and *analyzed*. Therefore, the document processor stems document terms to *analy-* so that documents which include various forms of *analy-* will have equal likelihood of being

retrieved; this would not occur if the engine only indexed variant forms separately and required the user to enter all. Of course, stemming does have a downside. It may negatively affect precision in that all forms of a stem will match, when, in fact, a successful query for the user would have come from matching only the word form actually used in the query [Eli01].

- **Extract index entries:** In this step the lexical tokens are generated from the remaining entries from the original document.
- **Term weight assignment:** Weights are assigned to terms in the index file. The more sophisticated the search engine, the more complex the weighting scheme. Measuring the frequency of occurrence of a term in the document creates more sophisticated weighting. Extensive experience in information retrieval research over many years has clearly demonstrated that the optimal weighting comes from use of "TF/IDF". This algorithm measures the frequency of occurrence of each term within a document. Then it compares that frequency against the frequency of occurrence in the entire database.
- **Create index:** The index or inverted file is the internal data structure that stores the index information and that will be searched for each query. Inverted files range from a simple listing of every alphanumeric sequence in a set of documents/pages being indexed along with the overall identifying numbers of the documents in which the sequence occurs, to a more linguistically complex list of entries, the TF/IDF weights, and pointers to where inside each document the term occurs. The more complete the information in the index, the better the search results [Eli01]. Inverted index is a mapping from keyword to the documents in which it appears [Jia00].

2.16 Ranking Part

Web ranking is very important to the searching quality of the search engine. Ideally the web page more relevant to user's query should have higher rank in the hitlist [Jia00].

Having determined which subset of documents or pages matches the query requirements to some degree, a similarity score is computed between the query and each document/page based on the scoring algorithm used by the system. Scoring algorithms rankings are based on the presence/absence of query term(s), term frequency, Boolean logic fulfillment, or query term weights. Some search engines use scoring algorithms not based on document contents, but rather, on relations among documents or past retrieval history of documents/pages.

After computing the similarity of each document in the subset of documents, the system presents an ordered list to the user. The sophistication of the ordering of the documents again depends on the model the system uses, as well as the richness of the document and query weighting mechanisms. For example, search engines that only require the presence of any alpha-numeric string from the query occurring anywhere, in any order, in a document would produce a very different ranking than one by a search engine that performed linguistically correct phrasing for both document and query representation.

Ranking is the heart of the search engine. In order to produce a good search engine, the search engine designer needs to know how to rank pages properly for the result documents. Today, most search engines use variations of the Boolean or vector model to do ranking. Recall that search engines do not allow access to the text, but only the indices, because it is too expensive in terms of time and space. So, when searching, ranking must use indices while not accessing the text. Besides that, there are also other difficulties as well. There might be too many relevant pages for a simple query [Sun01].

2.16.1 Difficulties in Determining Relevancy

Basing the rank of a Web page solely on the content of the page itself and in particular the content of the Meta tag, which does not even appear as part of the text of the page can cause problems for search engines. This is because savvy Web page authors can use a technique known as spamming, repeating a “hot” keyword many times in the title or the Meta tag to raise the rank of the page without adding any value to the content of that page. In an attempt to avoid falling victim to spamming, many search engines severely penalize pages that appear to be using this technique. Inevitably, legitimate pages are often unduly penalized. Recently, some search engines have begun to rely on the valuable information buried in the structure of the Web itself to rank pages in a more objective way.

2.16.2 Document Features

There are many document features that make a good match to a query, here are the most of them:

- **Term frequency:** How frequently a query term appears in a document is one of the good methods in determining a document’s relevance to a query. But in several situations this method fail, because many words have multiple meanings, they are polysemous. Many of the non-relevant documents presented to users result from matching the right word, but with the wrong meaning.
- **Location of term:** Many search engines give preference to words found in the title or lead paragraph or in the metadata of a document. Some studies show that the location in which a term occurs in a document or on a page indicates its significance to the document [Eli01].

- **Link:** Usually describes better a page than the page itself and makes possible to index non-text content. The link is associated not only to the page where it is found, but the one it points to. The only problem with this feature is that the destination of these links is not verified, so they may even not exist.
- **Date of Publication:** Some search engines assume that the more recent the information is, the more likely that it will be useful or relevant to the user. The engines therefore present results beginning with the most recent to the less current [Eli01].
- **Length:** Some search engines take the length of the page in consideration.
- **Presentation of words:** Some search engines use the attributes of the word as features like font size, font color, and font style.

2.17 User Interface Part

The user interface of search engines consists of three phases:

- 1- Query interface: provide the interface between the user and the search engine so that the user can write his/her query and send it to the search engine.
- 2- Query processor: which is responsible for parsing the query for search terms and perform some error checking.
- 3- Answer interface: which provide the results for the user or what is known as the hitlist.

2.17.1 Query Interface

Every query that result in a huge number of hits impossible to fit on one screen cannot be called a successful query. That is why it's necessary to create an interface with a larger configurability for user to use, giving it a

certain level of freedom to modify it by its own needs and wishes. Because of the fact that creating a user-accessible crawler would be unpractical, and allowing user access to main search engine database could be contra productive, interface remains as the only search engine element that can be offered to user as a tool. On the other hand, seeing that a large number of users are beginners, the search engine interface should have its simple form as well [Pet98].

A simple query interface page is shown in figure (2.7), which consists of a text box and a search button.

The diagram shows a rectangular frame containing the text "Welcome to Search Engine" centered at the top. Below this, there are two rectangular boxes side-by-side. The left box is labeled "Query field" and the right box is labeled "Search button".

Figure (2.7) Query interface

There are two types of query interface:

- 1- Basic query interface
- 2- Complex query interface

The basic query interface is a box where a sequence of words is entered. The sequence of words entered into different search engines produces different results. For example, AltaVista performs a search by the union of these words, whereas, HotBot performs a search by the intersection of these words (all words must appear in the result documents) [Sun01].

Some search engines support complex query interface, including Boolean operators (AND, OR, NOT) and other features, such as phrase search (like

“Text”), proximity search, URL searches, title search, date range, and data types search.

2.17.2 Query Processor

In query processing, a number of operations are performed on the input text (from the user) to transform it into a form understandable by the search engine.

The following steps are the possible operation performed on the input text

- **Tokenizing:** As soon as a user inputs a query, the search engine must tokenize the query stream, i.e., break it down into understandable segments. Usually a token is defined as an alphanumeric string that occurs between white space and/or punctuation.
- **Parsing:** Since users may employ special operators in their query, including Boolean, adjacency, or proximity operators, the system needs to parse the query first into query terms and operators. These operators may occur in the form of reserved punctuation (e.g., equation marks) or reserved terms in specialized formats (e.g., AND, OR). At this point, a search engine may take the list of query terms and search them against the inverted file. In fact, this is the point at which the majority of publicly available search engines perform the search.
- **Stop list and stemming:** Some search engines will go further and stop-list and stem the query. The stop list might also contain words from commonly occurring querying phrases, such as, “I’d like information about”. However, since most publicly available search engines encourage very short queries, as evidenced in the size of query window provided, the engines may drop these two steps.
- **Creating the query:** How each particular search engine creates a query representation depends on how the system does its matching. At

this point, a search engine may take the query representation and perform the search against the inverted file.

- **Query term weighting:** (assuming more than one query term). The final step in query processing involves computing weights for the terms in the query. Sometimes the user controls this step by indicating either how much to weight each term or simply which term or concept in the query matters most and must appear in each retrieved document to ensure relevance. Leaving the weighting up to the user is not common, because research has shown that users are not particularly good at determining the relative importance of terms in their queries. They can't make this determination for several reasons. First, they don't know what else exists in the database, and document terms are weighted by being compared to the database as a whole. Second, most users seek information about an unfamiliar subject, so they may not know the correct terminology. Few search engines implement system-based query weighting, but some do an implicit weighting by treating the first term(s) in a query as having higher significance. The engines use this information to provide a list of documents/pages to the user [Eli01].

After this final step the query is searched against the inverted file of documents.

2.17.3 Answer Interface

Answer interface or result page or hitlist is a list of links to web pages that contains the input text of the user. Search engines usually return pages in the order of relevance to the query. In other words, the most relevant pages appear on the top of the list. Typically, each result entry in the list includes a title of the page, an URL, a brief summary, a size, a date, and a written language.

2.18 Searching Part

Searching the inverted file for documents meeting the query requirements, referred to simply as "matching".

How systems carry out their search and matching functions differs according to which theoretical model of information retrieval underlies the system's design philosophy [Eli01].

While the computational processing required for simple, unweighted, non-Boolean query matching is far simpler than when the model is an NLP (natural language processing) -based query within a weighted, Boolean model, it also follows that the simpler the document representation, the query representation, and the matching algorithm, the less relevant the results, except for very simple queries, such as one-word, non-ambiguous queries seeking the most generally known information [Eli01].

The keywords in the query are matched against the inverted index to find all the documents ID's that contains the keywords. Then id is used to look up the URL and in some search engines the title of the corresponding Web pages and return the results to the ranking procedure.

2.19 Search Engines Examples

Two examples will be given for search engines, an Internet search engine and a Web site search engine.

-Internet Search Engine (Google) [Sun01]

The Google search engine (www.google.com) heavily uses the structure present in hypertext. It claims that it produces better results than other search engines today. The architecture is shown in the Figure (2.8).

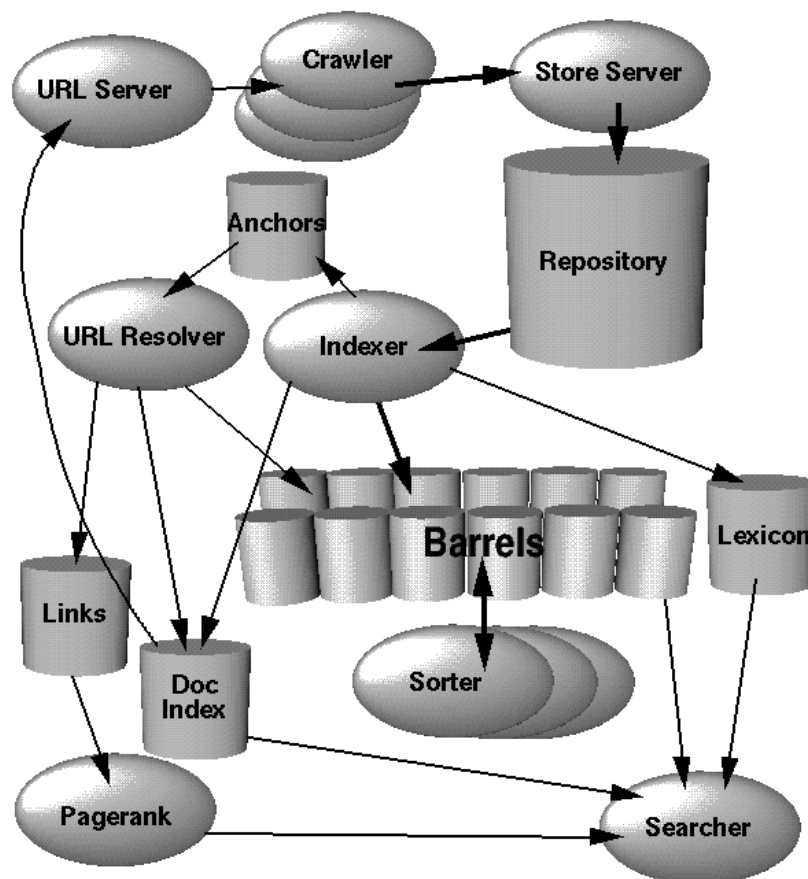


Figure (2.8) Google Architecture

The URL Server sends lists of URLs to be fetched by the crawlers. The crawlers download pages according to the list and send the downloaded pages to the Store Server. The Store Server compresses the pages and stores them in the repository. Every Web page has an associated ID number called a docID, which is assigned whenever a new URL is parsed out of a Web page. The index performs an indexing function. It reads the repository, uncompresses the documents, and parses them. Each page is converted into a Set of word occurrences called hits. The hits contain information about a word: position in document, an approximation of font size, and capitalization. The indexer distributes these hits into a set of “barrels” and creates a partially sorted forward index (like bucket sort). It parses out all the links in every Web page and stores important information about them in an anchors file.

The anchors file contains information about where each link points from and to and the text of the link. After that, the URL Resolver reads the anchors file and converts relative URLs into absolute URLs and in turn into docID. It puts the anchor text into the forward index, associated with the docID. It generates a links database for storing links and docIDs. The database is used to compute PageRanks for all the documents. The Sorter takes the barrels and resorts them by wordID instead of docID in order to generate the inverted index. Also, the Sorter produces a list of wordIDs and offsets into the inverted index. A program called DumpLexicon takes this list together with the lexicon produced by the indexer and generates a new lexicon to be used by the searcher. The searcher is run by a Web server and uses the lexicon built by DumpLexicon together with the inverted index and the PageRanks to answer queries.

-Web Site Search Engine (ICS) [Jia00]

In order to facilitate user's surfing experience on their Web sites, many institutes either license the searching tools from general search engine companies such as *Google* or create their own primitive search engines. Neither of these two options is ideal in some case because licensing search engine usually costs a lot of money and the searching quality of self-created primitive search engine is not satisfactory due to the lack of the expertise of applying modern technologies of building search engine, although these kind of technologies are available to public due to many researcher's hard work in the field of information retrieval in several decades [Jia00].

Figure (2.9) represents the architecture of the ICS Web site search engine.

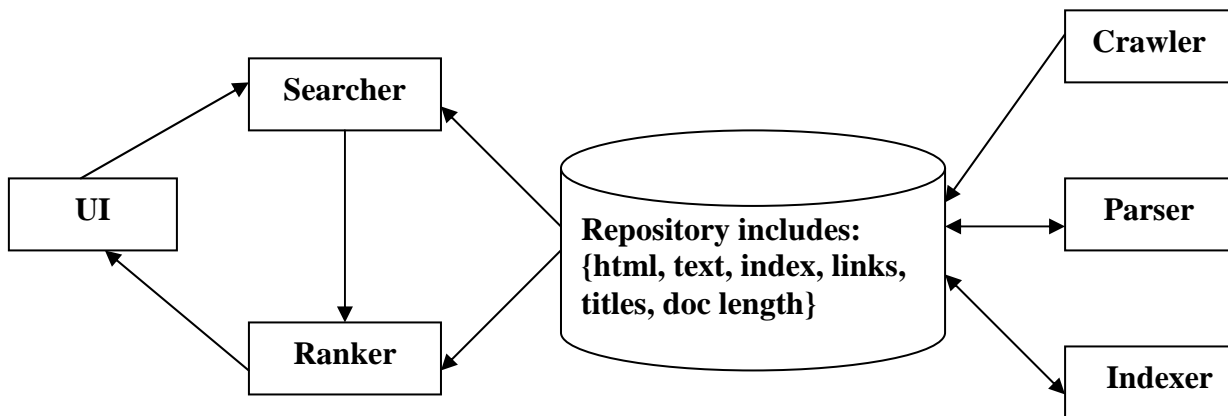


Figure (2.9) ICS Architecture

The crawler downloaded the web pages and saved the raw web pages into the main repository. Each web page is saved as an individual file. Each web page is assigned a unique id and the mappings between the web link and id is stored in a big file (links.txt). The parser parses each html file into pure text file and the title of the web page is also extracted. The mapping between web id and title are stored in one file (titles.txt). Given the parsed text files, the FOA indexer extracts the tokens from the text files, stem the tokens using Porter stemmer and generate keywords, and then scan the text files to create posting data structure for keywords. The inverted indexing is stored in InvertedIndex.txt. The web UI is used for user to submit the queries to search engine and receive the search results from ranker. Given the user's query, the search engine converts the query to keywords, match the query against the inverted index and retrieve the documents containing the keyword, and then pass the web page id, web links, and titles to ranker. The ranker will rank the web pages according the similarity between web page and user's query and return the web links and titles to UI. Web UI presents the results to user.

Chapter Three

Development of Web Site Search Engine

3.1 Introduction

The concern of this work is to develop a Web site search engine. This chapter begins by explaining the architecture of the proposed Web site search engine by dividing it in to two parts: Off-Line part and the On-Line part, then explain each part in details.

3.2 Architecture of Web Site Search Engine

The architecture of the proposed Web site search engine is shown in figure (3.1).

The Web site search engine consists of two parts: the **Off-Line** part and the **On-Line** part.

The Off-Line part consists of

- 1- Crawler: which will download all the pages in the Web site (Except for the non-HTML pages) and stores them as files in the storage unit.
- 2- Indexer: which will parse all the files (pages) downloaded by the crawler and index all the words in these files (except the stop words) and sort them to create the inverted index, then create the lexicon. The ranking is part of the indexing part.

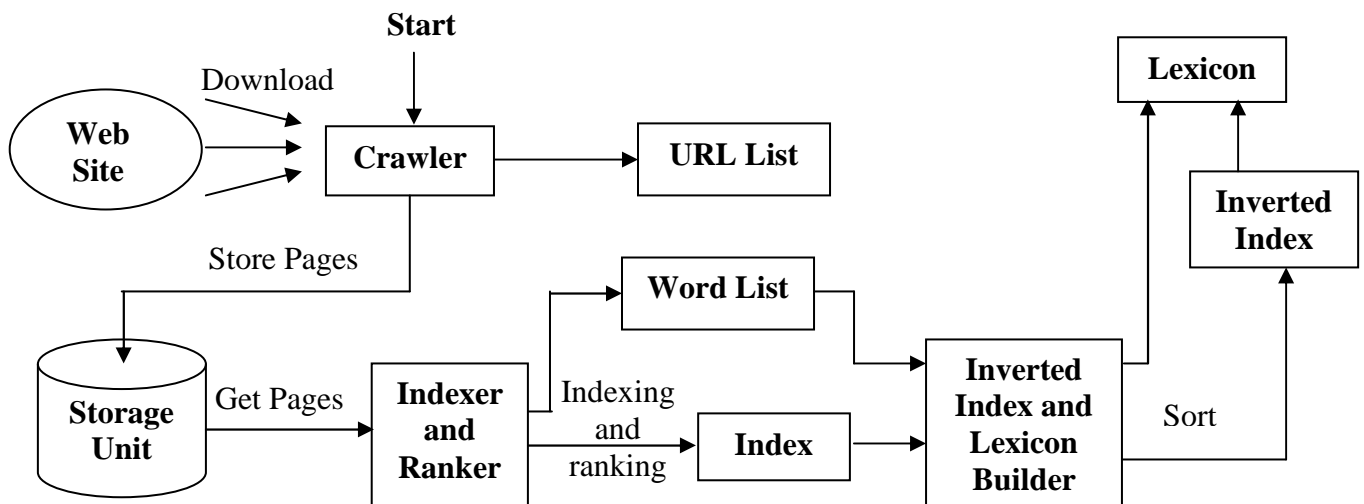


Figure (3.2) Off-Line Part

The Off-Line part is implemented on the search engine server and represent the first part as shown in figure (3.2), Off-Line means that till this moment the search engine is not ready to serve the users.

On-Line part consists of

- 1- Search Engine Interface: which consists of a Web page containing a text box and a button (search button), so that the user can write the query in the text box and click on the search button to start searching for the query and return the results. When the user write the address of the search engine on the address bar of the browser and press enter, the browser will display the search engine Web page by downloading it from the server of the search engine to the client computer (user).
- 2- CGI script: when the user click on the search button of the search engine Web page, the client computer will connect to the computer of the search engine and request to execute the CGI script of the search engine. The CGI script will perform a query processing to check the query and if there are no errors, the CGI script will pass the query to

the SES (Search Engine Server) using the CGI-SES (Common Gateway Interface-Search Engine Server) protocol.

- 3- SES: a program that runs on the server computer of the search engine and waits for requests from any search engine CGI script. When the SES receives a request from a CGI script, the SES will search the lexicon for the query and send the results to the CGI script, which will send them to the client computer (user).

The On-Line part is shown in figure (3.3)

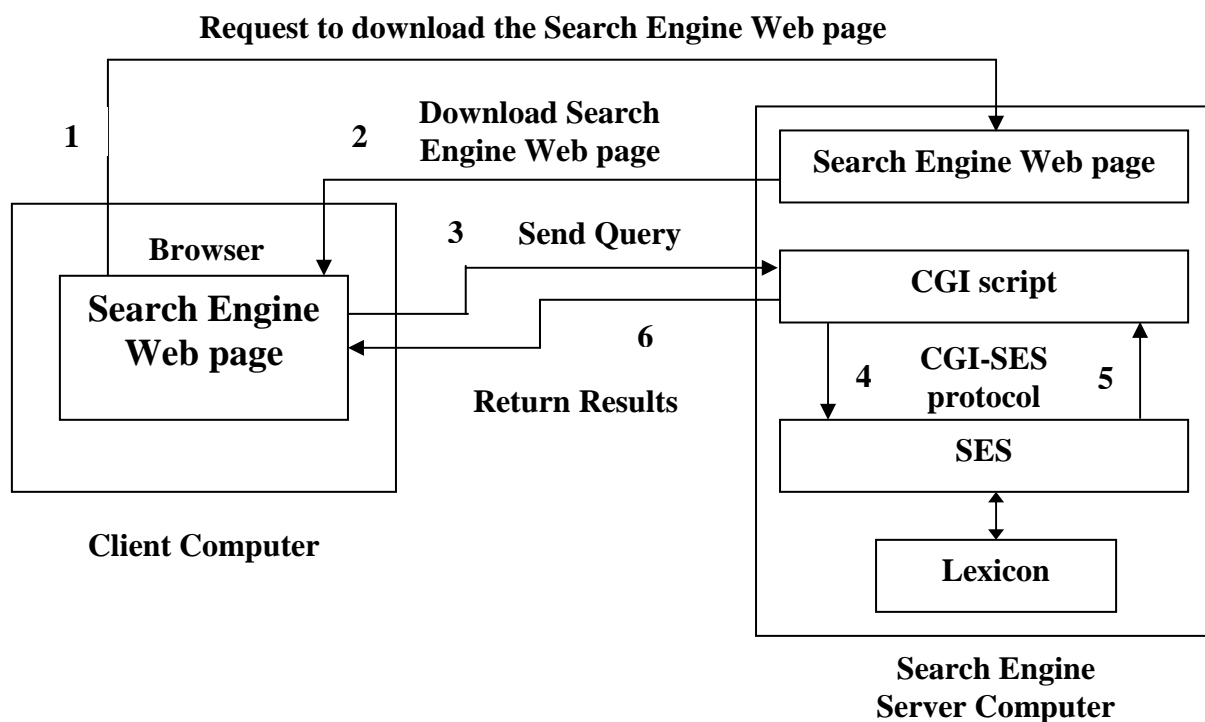


Figure (3.3) On-Line Part

In general, the Web site search engine consists of four parts:

- 1- Crawler
- 2- Indexer and ranker
- 3- Interface
- 4- Searcher

3.3 Crawler

The crawler consists of five parts; in the first part the information related to the crawling process is initialized, this information is important to know every thing about the crawling results. In the second part the data structure that contains the links to be downloaded from the Web site, which is called the **URL List**, is initialized with the **URL Startup File** links. Then the downloading and extracting part begin by downloading the pages from the URL List, save them as files on the storage unit, extract other links from them and add them to the URL List. In the Fourth part the crawling information is saved in a file named the **Crawler Results File**. Then the URL List is saved in a two files, one for the successfully downloaded links called the **True URL List** and the other for the unsuccessfully downloaded links called the **False URL List**.

A study by *Compaq systems research center* [Mar01], examines the average page quality over time of pages downloaded during a Web crawl of 328 million unique pages, it shows that traversing the Web graph in breadth-first search order is a good crawling strategy, as it tends to discover high-quality pages early on the crawl. So the breadth-first search order was used in the crawler of the proposed Web site search engine.

Figure (3.4) represents the five parts of the crawler.

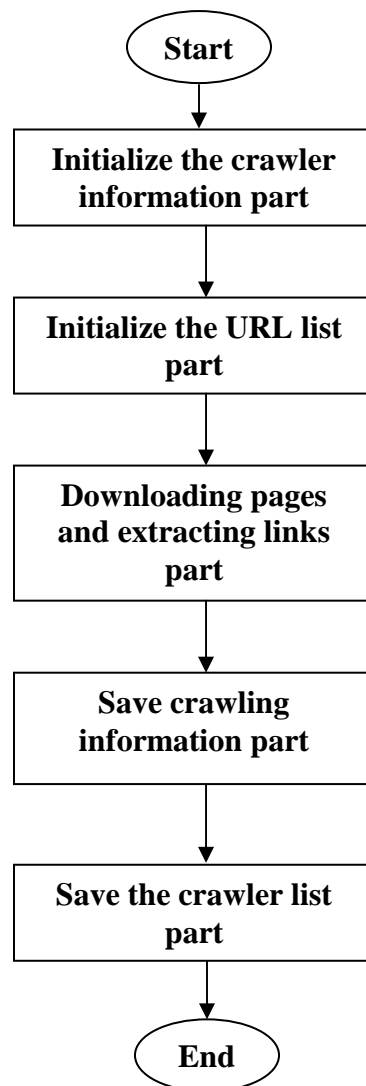


Figure (3.4) Crawling Parts

3.3.1 Initializing the Crawler Information

The first step in the crawler is to initialize the information about the crawling operation. The information of the crawling process contains the following variables, which will be initialized to zero.

- **File number:** each link from the URL List after downloaded will be saved as a file in the storage unit, each file name begins with the string “page” and ends with a number, the first file named **page0**, the second **page1** and so on, this variable represent the number of the file and increased by one each time a new page is downloaded.

- **Downloaded pages:** this variable keeps track of the number of links successfully downloaded.
- **External links:** this variable keeps track of the number of external links.
- **Non-HTML files:** this variable keeps track of the number of files that are not an html pages, like image files, sound files, E-Mail addresses, etc.
- **Connection error:** this variable keeps track of the number of failed connections.
- **Download error:** this variable keeps track of the number of failed downloading operations.
- **Save error:** this variable keeps track of the number of errors occurs during saving the pages on the storage unit.

If an **error occurs** during the initialization of the information of the crawling process, a **warning message** will be shown and the program will continue, because the initialization of the information is not a critical issue.

3.3.2 Initializing the URL list

The second part is used to initialize the URL List, but before the initialization, a text file called the **URL Startup File** should be created by the administrator of the search engine, which contains the most popular Web pages including the Home Page. The pages names entered into the file must be without the “**WWW**” or “**HTTP://**” and without the **Web site name** only the remaining string of the page name. The names must be separated by spaces. The URL List is initialized by the links in the URL Startup file.

The URL List is a **binary search tree** implemented as a **dynamic array** of records and each record represent a link in the URL List, which contains the following fields:

- **URL name:** which hold the name of the link without “**HTTP://**” and “**WWW.**” and without the **site name** because they are known and if stored will take a large space, so only the remaining name is hold. For example if we have the link “**http://www.microsoft.products.new.html**” then “**http://www.microsoft**” will be left and only “**products.new.html**” will be stored as the name of the link.
- **Left link:** a 4 bytes variable, which point to the link in the left direction as in the binary search tree.
- **Right link:** a 4 bytes variable, which point to the link in the right direction as in the binary search tree.
- **Downloading flag:** a 2 bytes variable, which represent the status of the link, if this link is downloaded successfully then it will be true, if not it will be false.

Table (3.1) represents an example of a URL List with 5 links:

Table (3.1) URL List

Cell number	URL name	Downloading flag	Left link	Right link
0	Index.html	True	2	1
1	Products.html	False	3	Null
2	About.htm	True	Null	Null
3	Map.html	True	Null	4
4	News.htm	False	Null	Null

Two variables are used to keep track of the URL list: one is the “**Current Link**” which keeps track of the current link that is to be downloaded and the other variable is the “**Last Link**” which always points to the last link in the URL list. The “**Current Link**” variable is initialized to point to the first link in the URL list which is location zero in the URL List and the “**Last Link**” variable is initialized to point to the last link in the URL list.

If an **error occur** during the initialization of the URL list a **critical error message** is shown and the program ends execution because it is a critical error, not a simple error.

3.3.3 Downloading Pages and Extracting Links

This part begins by downloading links from the URL list one by one and after each page is downloaded successfully all the links in that page is extracted out and only the **html** or **htm** links is added as a new links in the URL list if they are not already there. The crawler finishes his work when there is no more links to be extracted and the Current Link point to the Last Link. Figure (3.5) represents this part:

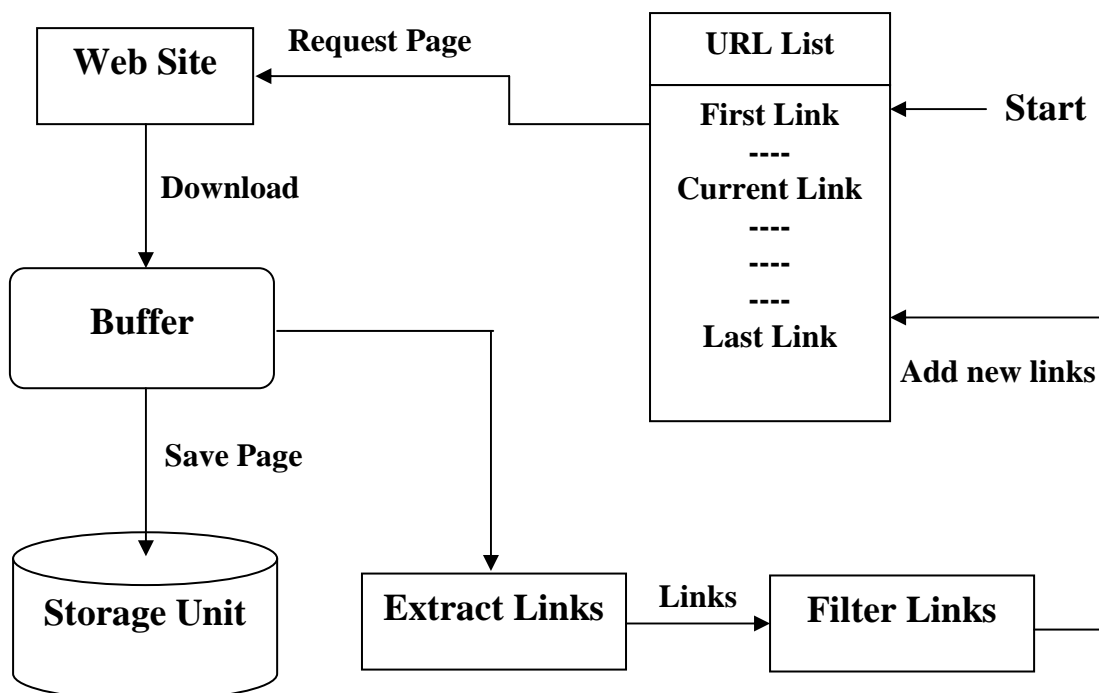


Figure (3.5) Crawling Process

Algorithm (3.1) is the **main** algorithm for crawling the Web pages

Algorithm (3.1)

Name : main crawling algorithm

Input : Pages of Web site

Output : Downloaded Pages

assign DownloadedPages **the value** 0

assign FileNumber **the value** 0

assign SaveErrors **the value** 0

assign DownloadedErrors **the value** 0

assign ConnectionErrors **the value** 0

while (current link is **less than or equal to** last link) **do**

(add "HTTP://" followed by Web site name to the current link

if (Web page downloaded correctly in memory)

then (**if** (Web page saved correctly on storage unit)

then (**assign** DownloadedPages **the value** DownloadedPages + 1

assign FileNumber **the value** FileNumber + 1

Execute algorithm 3.7 with Buffer containing the page as
input)

else (**assign** SaveErrors **the value** SaveErrors + 1)

)

else if (Web page could not be downloaded)

then (**assign** DownloadedErrors **the value** DownloadedErrors + 1)

else (**assign** ConnectionErrors **the value** ConnectionErrors + 1)

)

Algorithms (3.2) is used for **Adding New Link** to the URL List

Algorithm (3.2)

Name : Add New URL Link

Input : Link, URL list

Output : True or False

assign N **the value** 0

assign Condition **the value** "True"

assign ReturnValue **the value** "True"

Algorithm 3.2- (Continue)

```

while (Condition equal to "True") do
  (if (Link equal to the link of the Nth entry in URL list)
    then (assign Condition the value "False"
      assign ReturnValue the value "False")
    else if (Link less than link of the Nth entry in URL list)
      then (if (left link of Nth entry in URL list equal to Null)
        then (add Link as the left link of the Nth entry in URL list)
        else (assign N the value of left link of the Nth entry in URL list)
        )
      else if (Link greater than link of the Nth entry in URL list)
        then (if (right link of the Nth entry in URL list equal to Null)
          then (add Link as the right link of the Nth entry in URL list)
          else (assign N the value of right link of the Nth entry in URL list)
          )
        )
    )
  return the value ReturnValue

```

Algorithm (3.3) is used for **Extracting a Token** from the Web page

Algorithm (3.3)

Name : Get HTML Token

Input : Buffer containing the page, N (Pointer)

Output : HTML token

```

assign Token the value Null
assign N the value N + 1
while (Nth entry in Buffer equal to "Space" or "=" or "Quotation" or "Ascii13"
  or "Ascii10" and N is less than the length of the page) do
  (assign N the value N + 1)
while (Nth entry in Buffer not equal to ">" and "Space" and "=" and
  "Quotation" and "Ascii13" and N is less than the length of the page) do
  (assign Token the value of Token + Nth entry in Buffer
  assign N the value N + 1)
  Convert Token to lower case
return the value Token

```

Algorithm (3.4) is used for **Extracting a Link** from the Web page

Algorithm (3.4)

Name : Get URL Link

Input : Buffer containing the page, Html Token, N (pointer)

Output : URL link

assign Link the value Null

While (Nth entry in Buffer **not equal to** ">" and Link **not equal to** Token and N **Less than** the length of the page) **do**

(**Execute** algorithm 3.3 with Buffer containing the page, N as input

assign Link the Output of algorithm 3.3)

if (Link **equal to** Token)

then (**Execute** algorithm 3.3 with Buffer containing the page, N as input

assign Link the Output of algorithm 3.3

return the value of Link)

else (**return the value** Null)

Algorithm (3.5) is used for **Filtering a Link**

Algorithm (3.5)

Name : Filter URL Link

Input : URL Link, Web site name, BaseLink

**Output : Filtered URL Link if the Link is internal
or "1" if the Link is external
or "2" if the Link is not a HTML or HTM file**

assign LinkLen the value number of characters in Link

assign SiteLen the value number of characters in Web site name

if (LinkLen **less than** 5) **then** (**return the value** "2")

if (last 5 characters of Link **not equal to** ".html" and last 4 characters of Link **not equal to** ".htm") **then** (**return the value** "2")

assign Nu the value 1

if (first 7 characters of Link **equal to** "HTTP://")

then (**assign Nu the value** 8

if (characters from Nu to Nu + 4 of Link **equal to** "WWW.")

then (**assign Nu the value** 12)

)

Algorithm 3.5- (Continue)

```

else if (first 4 characters of Link equal to "WWW.")
    then (assign Nu the value 5)
if (Nu equal to 1)
    then (if (BaseLink equal to "1")
        then (return the value "1")
        else (return the value BaseLink + Link)
    )
else (if (characters from Nu to Nu + SiteLen in Link equal to Web site name)
    then (assign Nu the value Nu + SiteLen + 1
        return the value characters from Nu to LinkLen in Link)
    else (return the value "1")
    )

```

Algorithm (3.6) is used for **Filtering a base link**

Algorithm (3.6)

Name : Filter Base Link

Input : Base Link, Web site name

**Output : Filtered Base Link if the Base Link is internal
or "1" if the Base Link is external**

assign BaseLinkLen **the value** length of BaseLink

assign Nu **the value** 1

if (first 7 characters of BaseLink **equal to** "HTTP://")

then (**assign** Nu **the value** 8

if (characters form Nu to Nu + 4 of BaseLink **equal to** "WWW.")

then (**assign** Nu **the value** 12)

)

else if (first 4 characters of BaseLink **equal to** "WWW.")

then (**assign** Nu **the value** 5)

if (Nu **equal to** 1)

then (**if** (BaseLink **equal to** "1")

then (**return the value** "1")

else (**return the value** BaseLink + Link)

)

Algorithm 3.6- (Continue)

```

else (if (characters from Nu to Nu + SiteLen equal to Web site name)
    then (assign Nu the value Nu + SiteLen + 1
        return the value characters from Nu to BaseLinkLen in BaseLink)
    else (return the value "1")
    )

```

Algorithm (3.7) is used for **Extracting Links** from the Web page

Algorithm (3.7)

Name : HTML Link Extractor

Input : Buffer containing the HTML page

Output : HTML or HTML Links

assign BaseLink **the value** Null

assign N **the value** -1

assign PageLen **the value** length of the page

while (N **less than** PageLen)

(**assign** N **the value** N + 1

if (Nth entry in Buffer **equal to** "<" and Nth + 1 entry in Buffer **not equal to** "Space")

then (**Execute** algorithm 3.3 with Buffer, N as input

assign HtmlToken **the output of** algorithm 3.3

Execute one of the following blocks according to the value of HtmlToken

Case 1: HtmlToken equal to "base"

(**Execute** algorithm 3.4 with Buffer, "href", N as input

assign Link **the output of** algorithm 3.4

if (Link **not equal to** Null)

then (**Execute** algorithm 3.6 with Link, Web site name

assign BaseLink **the output of** algorithm 3.6)

)

Algorithm 3.7- (Continue)**Case 2:** HtmlToken equal to "a" or "area" or "link"

```
(Execute algorithm 3.4 with Buffer, "href", N as input
  assign Link the output of algorithm 3.4
  if (Link not equal to Null)
  then (Execute algorithm 3.5 with Link, Web site name, BaseLink as
    input
    assign Link the output of algorithm 3.5
    if (Link equal to "1")
    then (assign ExternalLinks the value ExternalLinks + 1)
    else if (Link equal to "2")
    then (assign NonHtmlFiles the value NonHtmlFiles + 1)
    else (Execute algorithm 3.2 with Link, URL list as input))
  )
```

Case 3: HtmlToken equal to "q"

```
(Execute algorithm 3.4 with Buffer, "cite", N as input
  assign Link the output of algorithm 3.4
  if (Link not equal to Null)
  then (Execute algorithm 3.5 with Link, Web site name, BaseLink as
    input
    assign Link the output of algorithm 3.5
    if (Link equal to "1")
    then (assign ExternalLinks the value ExternalLinks + 1)
    else if (Link equal to "2")
    then (assign NonHtmlFiles the value NonHtmlFiles + 1)
    else (Execute algorithm 3.2 with Link, URL list as input))
  )
```

Case 4: HtmlToken equal to "img"

```
(Execute algorithm 3.4 with Buffer, "longdesc", N as input
  assign Link the output of algorithm 3.4
  if (Link not equal to Null)
  then (Execute algorithm 3.5 with Link, Web site name, BaseLink as
    input
    assign Link the output of algorithm 3.5
    if (Link equal to "1")
    then (assign ExternalLinks the value ExternalLinks + 1)
    else if (Link equal to "2")
    then (assign NonHtmlFiles the value NonHtmlFiles + 1)
    else (Execute algorithm 3.2 with Link, URL list as input))
  )
```

Algorithm 3.7- (Continue)**Case 5:** HtmlToken **equal to** "iframe"

```

(Execute algorithm 3.4 with Buffer, "src", N as input
  assign Link the output of algorithm 3.4
  if (Link not equal to Null)
  then (Execute algorithm 3.5 with Link, Web site name, BaseLink as
    input
    assign Link the output of algorithm 3.5
    if (Link equal to "1")
    then (assign ExternalLinks the value ExternalLinks + 1)
    else if (Link equal to "2")
    then (assign NonHtmlFiles the value NonHtmlFiles + 1)
    else (Execute algorithm 3.2 with Link, URL list as input))
  )

```

Case 6: HtmlToken **equal to** "frame"

```

(assign HtmlToken the value Null
  while (Nth entry in Buffer not equal to ">" and HtmlToken not equal
    to "src" and HtmlToken not equal to "longdesc" and N less
    than PageLen) do
  (Execute algorithm 3.3 with Buffer, N as input
    assign HtmlToken the output of algorithm 3.3)
  if (HtmlToken equal to "src" or HtmlToken equal to "longdesc")
  then(Execute algorithm 3.3 with Buffer, N as input
    assign Link the output of algorithm 3.3
    if (Link not equal to Null)
    then (Execute algorithm 3.5 with Link, Web site name, BaseLink
      as input
      assign Link the output of algorithm 3.5
      if (Link equal to "1")
      then (assign ExternalLinks the value ExternalLinks + 1)
      else if (Link equal to "2")
      then (assign NonHtmlFiles the value NonHtmlFiles + 1)
      else (Execute algorithm 3.2 with Link, URL list as input))
    )
  )
)
)
)

```

3.3.4 Saving the Crawling Information

In this part the information about the crawling operation is saved in a file named the **Crawler Results File**.

If an **error occurs** during the saving operation of the crawling information, a **warning message** is shown and the program continues because it is not a critical error.

3.3.5 Saving the Crawler List

When the crawler finish his job, The URL list will contain the successfully downloaded links and the links that are not downloaded due to some error. Then the URL list will be divided into two lists: one for the successfully downloaded links called the **True URL List** and another for the unsuccessfully downloaded links called the **False URL List**. Each list will be saved on a file with a **header** and the header contains two variables:

- **Type:** which is a 2 bytes variable that is either true (represent the downloaded list) or false (represent the list with the links that are not downloaded).
- **Number of items:** a 4 bytes variable, which contains the number of items in the list.

The two lists are lists of string, which represent the URL link name, table (3.2) present an example of a True URL List file and table (3.3) present an example of a False URL List file taken from table 3.1.

Table (3.2) True URL list file

2 Bytes	True	}	Header
4 Bytes	3		
10 Bytes	Index.html	}	List of links
9 Bytes	About.htm		
8 Bytes	Map.html		

Table (3.3) False URL list file

2 Bytes	False	}	Header
4 Bytes	2		
13 Bytes	Products.html	}	List of links
8 Bytes	News.htm		

If an **error occur** during the saving operation of the crawling lists (true and false lists), a **critical error message** is shown and the program end execution.

3.4 Indexer and Ranker

The indexer consists of the following three parts:

- Create the index
- Create the inverted index
- Create the lexicon

The ranking part is mixed with the indexing part. During indexing, each word will have a rank status (priority).

3.4.1 Creating the Index

In this part all the word are extracted from the pages that are downloaded during the crawling part and all the words except the stop words are indexed in the search engine database.

Figure (3.6) shows the indexing parts:

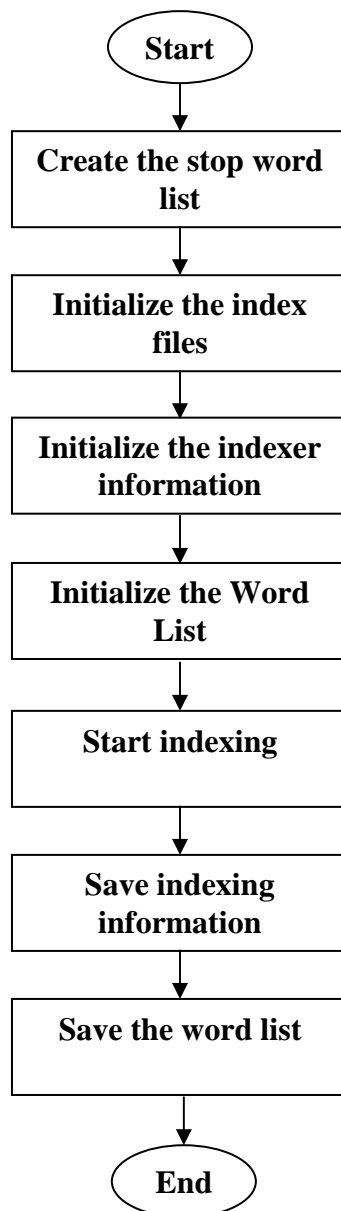


Figure (3.6) Indexing Parts

-Create the Stop Word List

The stop words are the words that are not important to the search engine like (to, is, are, etc.).

In this part the stop word list is created, first the administrator of the search engine creates a text file containing the stop words separated by spaces. Then the **Stop Word List** is initialized with the words of the stop word text file.

The Stop Word List is a **binary search tree** implemented as a **dynamic array** of records; each record represent a stop word which contains the following fields:

- **Stop Word:** represent the stop word, which is a string.
- **Left Link:** 2 bytes variable represent the left link in the Stop Word List.
- **Right Link:** 2 bytes variable represent the right link in the Stop Word List.

The Stop Word file contains a **Header**, which consists of:

- **Type:** 4 bytes variable represent the type of the list, which will be “Stop”.
- **Number of items:** 2 bytes variable represent the number of stop words in the list.

The stop word list used in this work is shown in table (3.4)

Table (3.4) Stop Word List

i	a	about	an	are
as	at	be	by	com
for	from	how	in	is
It	of	on	or	that
the	this	to	was	what
when	where	who	will	with
www				

After initializing the Stop Word List with the words in the stop word text file, the Stop Word List is saved on the storage unit as a file then loaded at the beginning of the indexing operation into the memory.

-Initialize the Index Files

The index files from A to Z (AindexFile.IF, BindexFile.IF, etc.) and from 0 to 9 (0IndexFile.IF, 1IndexFile.IF, etc.); are opened and initialized to Null. All the words that have the same first letter are saved in the same file that named on that first letter, for example the words (car, cup, coffee) are saved in the CindexFile.IF file.

The index file is a file of records, and each record contains the following fields:

- **Word ID:** 4 bytes variable represent the id of one of the indexed words.
- **Page ID:** 4 bytes variable represent the id of the page where the word occur.
- **Position:** 4 bytes variable represent the position where the word occurs in the page.
- **Rank Status:** 2 bytes variable represent the rank status of the word (priority of the word).

-Initialize the Indexer Information

In this part the information related to the indexing operation are initialized, these information are only the **Unindexed Files Number**. During the indexing operation an error may occur and many files may not indexed so this variable is important to know how many files are not indexed.

-Initialize the Word List

The Word List will contain all the words that occur in the Web pages. The Word List is a **binary search tree** implemented as a **dynamic array** of records and each record contains the following fields:

- **Word:** represents the word as a string.

- **Number:** the number of occurrences for the word in all the Web pages.
- **Left Link:** represents the left link of the binary search tree.
- **Right Link:** represents the right link of the binary search tree.

-Start Indexing

For each Web page, load it from the storage unit to the memory then parses the page looking for words. For each word, if it is not part of the **Stop Word List**, add the word to the **Word List**, if it is already in the Word List then increase the number of occurrences for that word. The Word List will return an ID for the word after inserting it. Then add the word to the **Index List** with its rank status and it's ID.

The Index List is a binary search tree implemented as a dynamic array of records, and each record contains the following fields:

- **Word:** represents the word as a string.
- **Word ID:** 4 bytes variable represents the ID of the word.
- **Position:** 4 bytes variable represents the position of occurrences for the word in the page.
- **Rank Status:** 4 bytes variable represents the priority for the word.
- **Left Link:** 4 bytes variable represents the left link for the binary search tree.
- **Right Link:** 4 bytes variable represents the right link for the binary search tree.

After finish the work with the current page, save the words from the Index List into the Index Files, each word in its correct Index File.

The following are the algorithms used in the indexing part:

Algorithm (3.8) is used for extracting a word from the file.

Algorithm (3.8)**Name : Get Word****Input : Buffer containing the page, N (pointer)****Output : Word****assign** Word **the value** Nth entry in Buffer**assign** N **the value** N + 1**while** ((Nth entry in Buffer **greater than or equal to** "a" and
Nth entry in Buffer **less than or equal to** "z") or
(Nth entry in Buffer **greater than or equal to** "A" and
Nth entry in Buffer **less than or equal to** "Z") or
(Nth entry in Buffer **greater than or equal to** "0" and
Nth entry in Buffer **less than or equal to** "9") and N **less than**
PageLen) **do**(**assign** Word **the value** Word + Nth entry in Buffer**assign** N **the value** N + 1)

Convert Word to upper case

return the value Word

Algorithm (3.9) is the parser algorithm.

Algorithm (3.9)**Name : Parser****Input : Buffer containing the Web page****Output : List of words with their rank status****assign** N **the value** 0**assign** RankStatus **the value** 1**assign** PageLen **the value** length of the page**while** (N **less than** PageLen) **do**(**while** (Nth entry in Buffer **equal to** "Space" or "Ascii13" or "Ascii10" and N **less**
than PageLen) **do**(**assign** N **the value** N + 1)**if** (Nth entry in Buffer **equal to** "<" and Nth+1 entry in Buffer **equal to** "/")**then** (**assign** N **the value** N + 1

Algorithm 3.9- (Continue)

```

Execute algorithm 3.3 with Buffer, N as input
assign Token the output of algorithm 3.3
Execute one of the following blocks according to the value of Token
Case 1: Token equal to "b" or "strong" or "blink"
    (assign RankStatus the value RankStatus - 4)
Case 2: Token equal to "i" or "u" or "s" or "strike" or "code" or "samp" or
    "var" or "em" or "blockquote" or "tt" or "cite" or "address" or
    "sub" or "sup" or "kbd"
    (assign RankStatus the value RankStatus - 2)
Case 3: Token equal to "big"
    (assign RankStatus the value RankStatus - 12)
Case 4: Token equal to "marquee"
    (assign RankStatus the value RankStatus - 6)
Case 5: Token equal to "title"
    (assign RankStatus the value RankStatus - 128)
Case 6: Token equal to "a"
    (assign RankStatus the value RankStatus - 64)
Case 7: Token equal to "h1" or "h2" or "h3" or "h4" or "h5" or "h6"
    (assign RankStatus the value RankStatus - 32)
Case 8: Token equal to "font"
    (assign RankStatus the value RankStatus - 8)
while (Nth entry in Buffer not equal to ">" and N less than PageLen) do
    (assign N the value N + 1)
    assign N the value N + 1
    )
else if (Nth entry in Buffer equal to "<" and Nth+1 entry in Buffer not equal to
    "Space")
then (Execute algorithm 3.3 with Buffer, N as input
    assign Token the output of algorithm 3.3
    Execute one of the following blocks according to the value of Token
    Case 1: Token equal to "b" or "strong" or "blink"
        (assign RankStatus the value RankStatus - 4)
    Case 2: Token equal to "i" or "u" or "s" or "strike" or "code" or "samp" or
        "var" or "em" or "blockquote" or "tt" or "cite" or "address" or
        "sub" or "sup" or "kbd"
        (assign RankStatus the value RankStatus + 2)
    Case 3: Token equal to "big"
        (assign RankStatus the value RankStatus + 12)

```

Algorithm 3.9- (Continue)

```

Case 4: Token equal to "marquee"
      (assign RankStatus the value RankStatus + 6)
Case 5: Token equal to "title"
      (assign RankStatus the value RankStatus + 128)
Case 6: Token equal to "a"
      (assign RankStatus the value RankStatus + 64)
Case 7: Token equal to "h1" or "h2" or "h3" or "h4" or "h5" or "h6"
      (assign RankStatus the value RankStatus + 32)
Case 8: Token equal to "font"
      (assign RankStatus the value RankStatus + 8)
while (Nth entry in Buffer not equal to ">" and N less than PageLen) do
  (assign N the value N + 1)
  assign N the value N + 1
  )
else if ((Nth entry in Buffer greater than 64 and Nth entry in Buffer less than
  91) or (Nth entry in Buffer greater than 96 and Nth entry in Buffer
  less than 123) or (Nth entry in Buffer greater than 47 and Nth entry in
  Buffer less than 58))
  then (Execute algorithm 3.8 with Buffer, N as input
  assign Word the output of algorithm 3.8
  if (Word is not a stop word)
    then (add Word in the word list)
  )
else (assign N the value N + 1)
  )

```

Algorithm (3.10) is the main indexing algorithm.

Algorithm (3.10)

Name : Main indexing algorithm

Input : HTML Pages

Output : Index Files

assign PageNo **the value** 0

assign UnindexedPages **the value** 0

Algorithm 3.10 – (Continue)

```

repeat
  (if (page loaded in memory correctly)
    then (parse the page and put all the information relative to them in a list
      save extracted words from list to all index files)
    else (assign UnidexedPages the value UnidexedPages + 1)
  assign PageNo the value PageNo + 1
  )
until (PageNo greater than number of crawled pages)

```

The ranking status for each word depends on the characteristics of the word, which are: font attributes (size, color, style), the position of the word in the page and the status of the word (is it a link to other page?).

The ranking part is mixed with the indexing part, during indexing, some HTML Tags (commands) will affect the word rank status as shown in table (3.5):

Table (3.5) Tags Ranking

Tag	Description	Rank Score
B, STRONG, and BLINK	Change the word style	4
I, U, S, STRIKE, CODE, SAMP, VAR, EM, BLOCKQUOTE, TT, CITE, ADDRESS, SUB, SUP and KBD	Change the word style	2
BIG	Change the word style and size	12
MARQUEE	Convert the word to a scrolling word	6
TITLE	Change the word to a title	128

A	Change the word to a link	64
H1, H2, H3, H4, H5 and H6	Change the word to a heading	32
FONT	Change the word font style, color, or size	8

-Save Indexing Information

In this part, the indexing information is saved on a file; this information represents the number of unindexed pages. The file name will be “**indexFilesInfo.IFI**”.

-Save the Word List

In this part the Word List is saved on a file, which contains a **header**, and this header is as follows:

- **Type:** 8 bytes variable that represents the type of the file, which is a Word List file (“WordList”).
- **Number of items:** 4 bytes variable that represents the number of items in the Word List.

The Word List is saved without the Left Link and the Right Link, only the Word and the Number of occurrences.

3.4.2 Creating the Inverted Index

In this part each index file is loaded into the memory and sorted using the **Improved Quick sort** and **Insertion sort** methods then the index file is saved back to the storage unit.

Algorithm (3.11) is the improved quick sort algorithm:

Algorithm (3.11)

Name : Improved Quick Sort

Input : Buffer containing the Index file, Left pointer, Right pointer

Output : Sorted index file

Algorithm 3.11- (Continue)

```

If (Left less than Right)
  then (assign J the value Left
    assign K the value Right
    if (Word of entry Left in Buffer less than the Word of entry Right in Buffer)
      then (swap between Left and Right records of Buffer)
    else if (Word of entry Left in Buffer equal to the Word of entry Right in
      Buffer)
      then ( if (RankStatus of of entry Left in Buffer less than RankStatus of
        Entry Right in Buffer)
          then (swap between Left and Right records of Buffer))
    repeat
      (
        repeat (assign J the value J + 1)
        until (Word of Jth entry in Buffer less than Word of entry Left in Buffer
          or (Word of Jth entry in Buffer equal to Word of entry Left in
          Buffer and RankStatus of Jth entry in Buffer less than or equal to
          RankStatus of entry Left in Buffer))
        repeat (assign K the value K - 1)
        until (Word of Kth entry in Buffer greater than Word of entry Left in
          Buffer or (Word of Kth entry in Buffer equal to Word of entry Left
          In Buffer and RankStatus of Kth entry in Buffer greater than or
          equal to RankStatus of entry Left in Buffer))
        if (J less than K)
          then (swap between record of Jth entry and record of Kth entry in
            Buffer)
        ) until (J greater than K)
      swap between record of entry Left and record of Kth entry in Buffer
    if ((K - Left) greater than 10)
      then (Execute algorithm 3.11 with Buffer, Left, K - 1 as input)
    if ((Right - K) greater than 10)
      then (Execute algorithm 3.11 with Buffer, K + 1, Right as input)
    )

```


Algorithm (3.12) is the Insertion Sort algorithm

Algorithm (3.12)

Name : Insertion Sort

Input : Buffer containing the index file, First pointer, Last pointer

Output : Sorted index file

assign K the value Last - 1

repeat (**assign J the value** K + 1

assign Save **the value** record of Kth entry in Buffer

assign Flag **the value** "True"

while (J **less than** Last and Flag **equal to** "True") **do**

(**if** (Word of Save record **less than** Word of Jth entry in Buffer)

then (**assign** Jth - 1 entry in Buffer **the value** of Jth entry in Buffer
assign J **the value** J + 1)

else if (Word of Save record **equal to** Word of Jth entry in Buffer)

then (**if** (RankStatus of Save record **less than** RankStatus of Jth
entry in Buffer)

then (**assign** Jth - 1 entry in Buffer **the value** of Jth entry in
Buffer

assign J **the value** J + 1)

else (**assign** Flag **the value** "False"))

else (**assign** Flag **the value** "False")

)

if (Word of Save record **less than** Word of Jth entry in Buffer)

then (**assign** Jth - 1 entry in Buffer **the value** of Jth entry in Buffer
assign J **the value** J + 1)

else if (Word of Save record **equal to** Word of Jth entry in Buffer)

then (**if** (RankStatus of Save record **less than** RankStatus of Jth
entry in Buffer)

then (**assign** Jth - 1 entry in Buffer **the value** of Jth entry in
Buffer

assign J **the value** J + 1))

assign Jth - 1 entry in Buffer **the value** of Save

assign K **the value** of K - 1

) **until** (K **less than** 1)

3.4.3 Creating the Lexicon

The lexicon is a MultiLayer structure consists of 4 layers, Each layer is an array of records and each record is 4 bytes variable used to point to the position of the word in the inverted index file (the number of the record in the inverted index file), these layers are:

- **Layer1:** one dimension array that contains pointers to all the words with only one character like the word “A”.
- **Layer2:** two dimension array that contains pointers to all the words with only two characters like the word “TV”.
- **Layer3:** three dimension array that contains pointers to all the words with only three characters like the word “Car”.
- **Layer4:** four dimension array that contains pointers to all the words with only four characters like the word “Card”.

Algorithm (3.13) is used for building the lexicon and called for each inverted index file.

Algorithm (3.13)

Name : Building the lexicon

Input : Buffer containing the inverted index file

Output : lexicon

assign Max **the value** number of words in the index file

assign N **the value** 1

while (N **less than or equal to** Max) **do**

(**assign** Word **the value** of Word of Nth entry in index file

assign FirstRecordNu **the value** N

assign N **the value** N + 1

assign Flag **the value** “True”

while (N **less than or equal to** Max and Flag **equal to** “True”) **do**

(**if** (Word **equal to** Word of Nth entry in index file)

then (**assign** N **the value** N + 1)

else (**assign** Flag **the value** “False”))

Algorithm 3.13 (Continue)

```
If (Word is only one character long)
  then (assign Layer1 with index (Word) the value FirstRecordNu)
else if (Word is only two characters long)
  then (assign Layer2 with index (first character, second character) the value
    FirstRecordNu)
else if (Word is only three characters long)
  then (assign Layer3 with index (first character, second character, third character)
    the value FirstRecordNu)
else if (Word is only four characters long)
  then (assign Layer4 with index (first character, second character, third haracter,
    fourth character) the value FirstRecordNu)
assign Flag the value "True"
while (N less than Max and Flag equal to "True") do
  (if (first four character of word equal to first four character of word of Nth entry in
    index file)
    then (assign N the value N + 1)
    else (assign Flag the value "False"))
  )
```

3.5 Interface

The interface of the search engine is a simple Web page consists of a text box for entering the query and a button for sending the Query to the Web site. When the user enter the query into the text box and click on the search button, the query will be send to the Web site server which will run the Search.exe CGI program to receive the query, then the CGI program will check the query for errors, and if there was any error, the CGI program will send an error message back to the client. If no error was found, the CGI program will convert each AND to * and each OR to +, make a connection with the search engine server using the CGI-SES protocol and send the query to the search engine server as shown in figure (3.7).

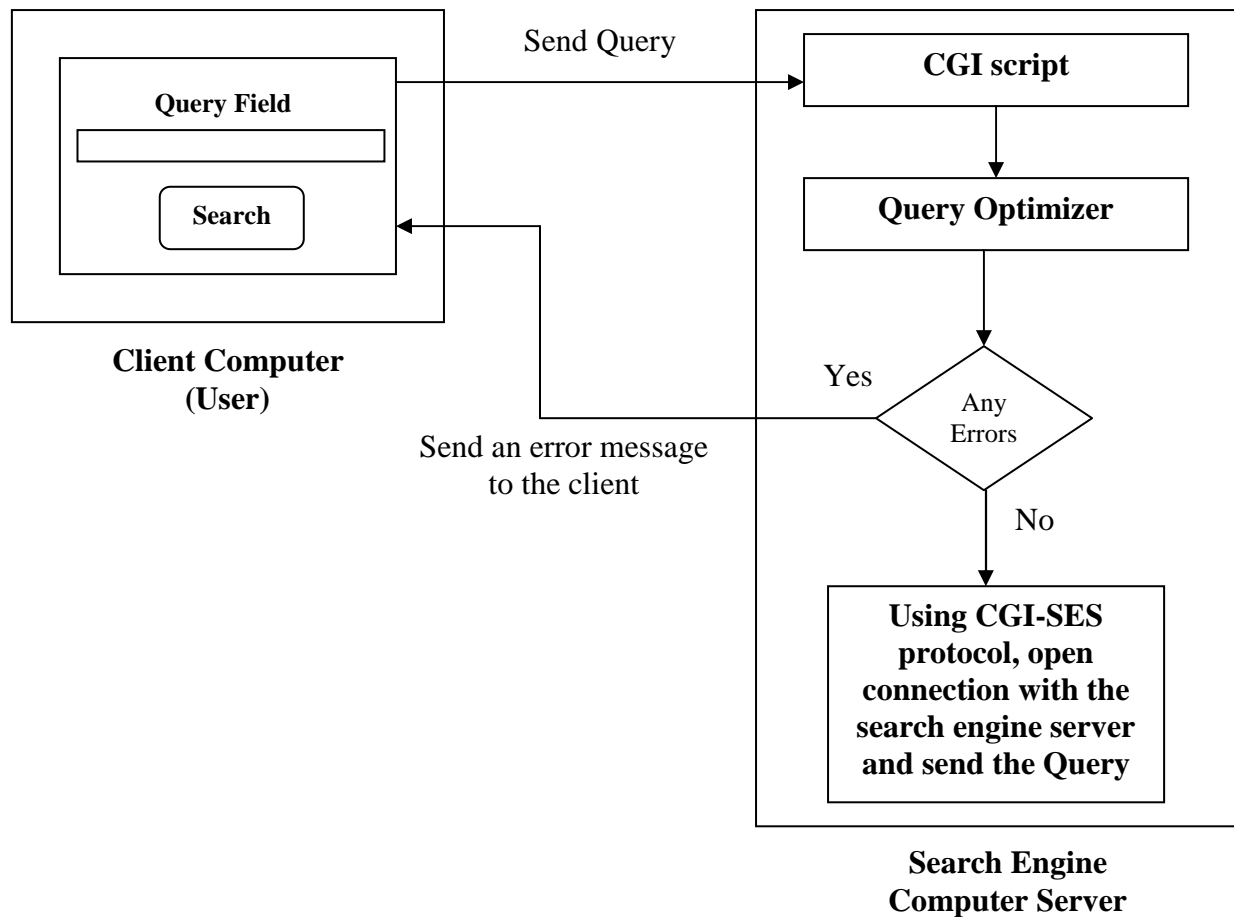


Figure (3.7) User-CGI Interface

3.6 Searcher

The searcher is the Search Engine Server (SES) that is responsible for serving any requests from the CGI programs by accepting the request, receive the query, search for the query and return the results back to the CGI program, which will return the results to the client through the Web site server.

When the SES starts running, it loads the lexicon, the word list, the URL list into the memory, then wait for any requests from CGI programs.

When a request arrives, the SES will accept the request by making a connection, take the query and search for the words in the query. There are two types of query that the SES can search for:

- **Normal Query:** which contains one word, the SES search for the word in its lexicon.
- **Boolean Query:** which contains Boolean operators (AND, OR) like (Car and station), the SES will create an array of counters and search for each word, increment the counter for that page ID for each word. For example if we are searching for the query (Car and Station) then we will search for the word “Car” and increment all the counters with an index equal to the page ID where that word was found, then search for the word “Station” and increment all the counters with an index equal to the page ID where that word was found, then all the counters that contains the number 2 (number of words in the query) will represent a page ID where the two words appears in.

For example if we want to search for the word “TV” then all the occurrences of that word will be found in the T index file because it starts with a T letter. But we don’t know the number of the first record in the T index file where the TV word will be found and the number of occurrences for that word. The second layer of the lexicon will contain the number of the first record in the T index file and in that record we will find the ID of the word so that we can supply it to the word list to get the number of occurrences for the word. So the word “TV” will be found in the T inverted index file starting at the record with the number found in the second layer and ending at the record with the number (first record number + number of occurrences for the word) as shown in figure (3.8).

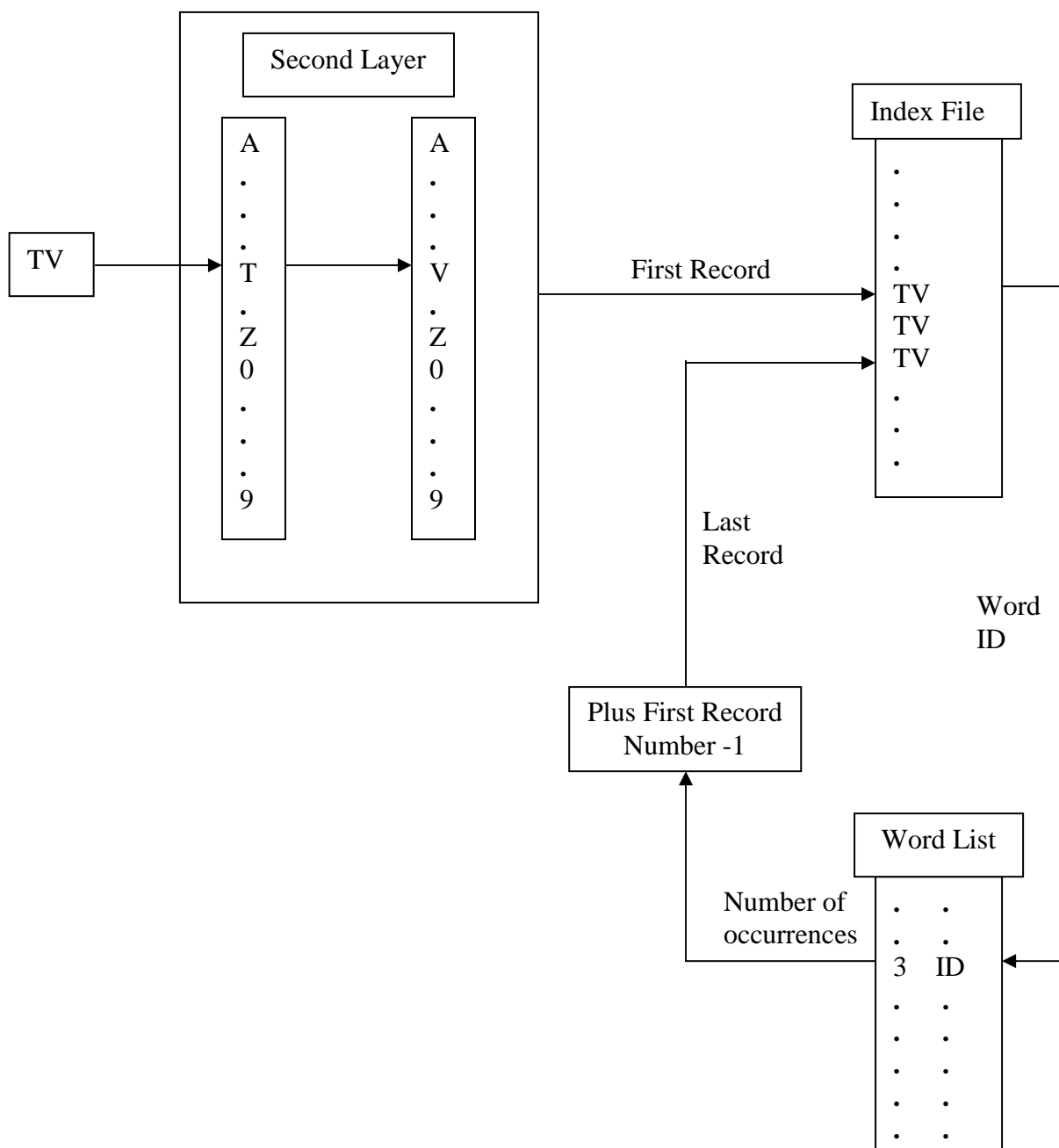


Figure (3.8) Searching-Phase1

All the records found in the index file containing the words “TV” will also contain the page ID where these words occur, so these page IDs will be used to retrieve the URLs string from the URL List and return these URLs to the CGI program which will return them after ordering them to the client as the final results as shown in figure (3.9).

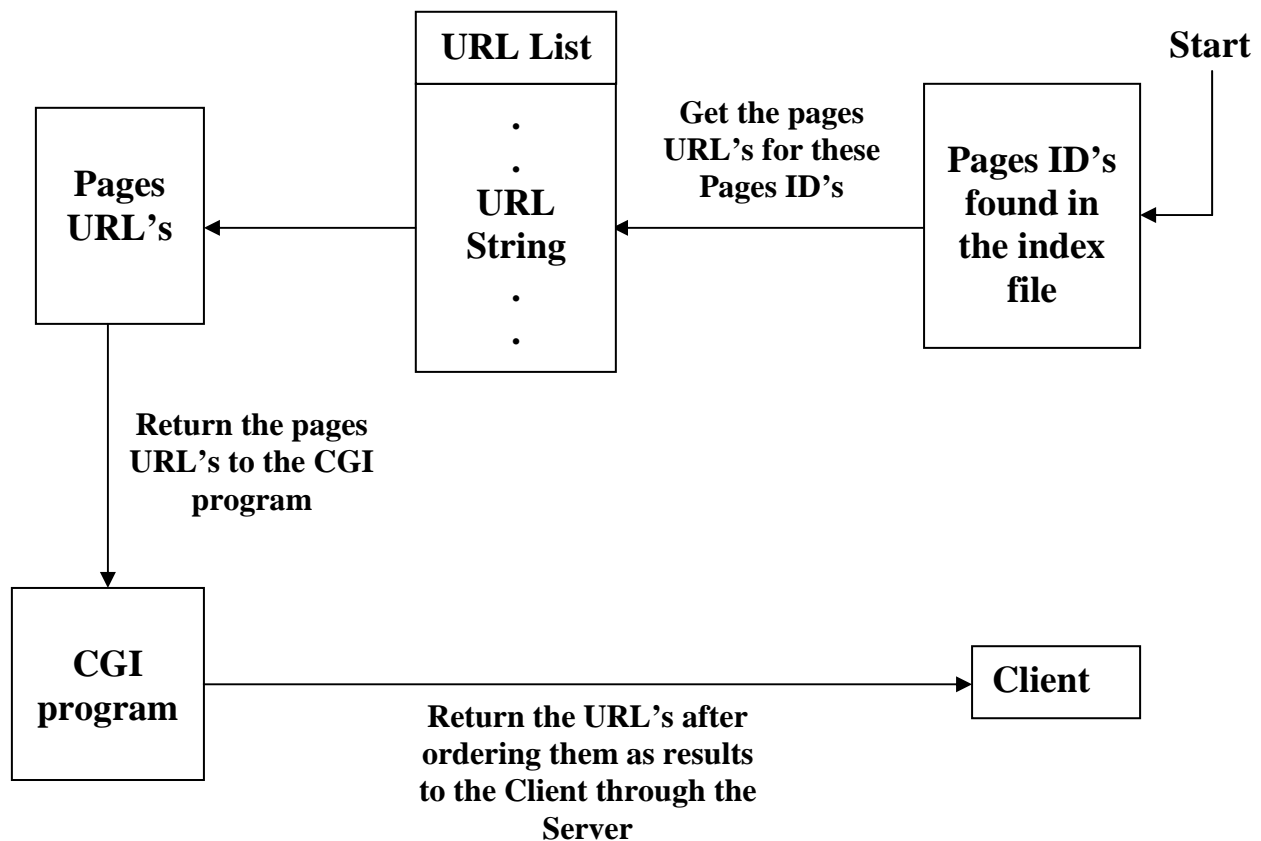


Figure (3.9) Searching-Phase2

3.7 The CGI-SES Protocol

The CGI talks to the SES using a protocol that was design for this project; this protocol is used for sending and receiving commands and data between both of them.

When the CGI program receives a request from the user for searching a specific text, the CGI program will send the **first page number**, **number of pages required** and the **text required** to the SES separated by spaces using the **Winsock** (a Socket Technique used by windows) connection.

The SES will respond with one of the following messages:

1. "ERROR: text": send back to the CGI program if an error was found in the data coming from the CGI program, and "text" is a description for the error.

2. “NOT FOUND”: send back to the CGI program if the user query was not found in the search engine database.
3. “FOUND NuOfResults FirstPage LastPage URL1 URL2 ...”: send back to the CGI program if the user query was found, where NuOfResults represent the number of results found, FirstPage and LastPage are like “from FirstPage to LastPage”, for example if the number of results to be displayed in the screen are 10 results, the user search for some word for the first time and the search database contain 30 results then NuOfResults will be 30, FirstPage will be 1 and LastPage will be 10, which means the search engine found 30 results and this message contains the results from 1 to 10 and the URL1 ... URL10 is the URL strings for these results.

The CGI will receive the message from the SES and respond to the client according to the contents of the message by generating a HTML page on the fly and send it to the client.

Chapter 4

Web Site Search Engine Operation

4.1 Introduction

This chapter consists of 6 parts, the first part explains the programming languages used in developing the Web site search engine, the second part explain how to run and use the proposed Web site search engine, the third part list the features of the proposed Web site search engine, the forth part discuss the system requirement, the fifth part discuss the experiment and results, and the last part show the tools used with the Web site search engine, which helps the Web site search engine administrator to check the search engine operations.

4.2 Programming languages

Visual Basic was used as the main programming language for implementing the crawler and the indexer.

VBScript and **HTML** were used as the programming languages in the implementation of the user interface.

CGI (Common Gateway Interface) was used in implementing the script (program) that is called by the Web site server.

The search engine is one of the new topics in computer science, the first search engine was designed in 1995, building a search engine is a challenging task requiring a lots of reading and learning many new subjects and techniques, so the time was the master factor in this project leaving little time for learning a visual language and being familiar with it's techniques and especially the Internet techniques. Visual Basic was used because it is simple, easy to learn than the other visual languages, leaving lots of time for the project design.

HTML is the basic language for building Web pages so it has been used, and VBScript is a **client-side** scripting language that is very simple and very close to the Visual Basic programming language.

CGI is the **server-side** scripting language that was used because of its capabilities than the other server-side scripting languages like **ASP (Active Server Page)** and **PHP (Personal Home Page)**. Appendix B contains an entire CGI library designed and written in visual basic for this work.

4.3 User Interface

The project is divided into 3 parts: the Crawler, the Indexer and the Search Engine Server (SES), each one have it's own interface. To put the search engine on line, the Search Engine Administrator (SEA) must crawl the Web site, index the pages to build the index, inverted index, and the lexicon then run the SES, which will be ready for receiving any request.

4.3.1 Crawler User Interface

To start the crawling process, first the SEA must enter some information to the crawler program, which is:

- **Output Path:** the crawler needs this path to store the true URL list file (TrueURLList.cul), the false URL list file (FalseURLList.cul) and the

crawling results file (CrawlerResults.CR). For example to set the output path to “c:\”, click on **options**, choose **Set Output Path** then write in the text box “c:\” as shown in figure (4.1).

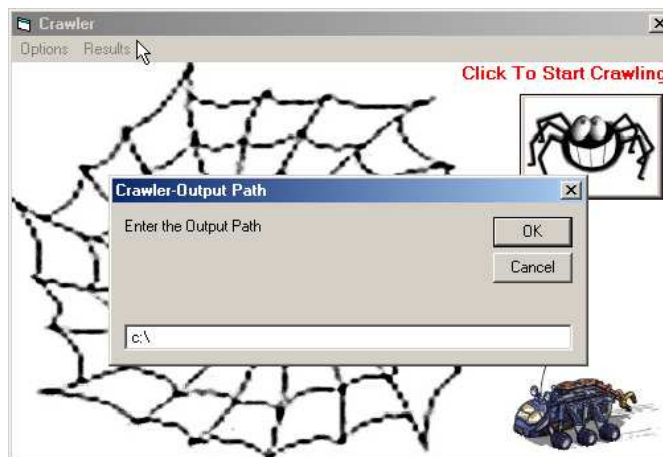
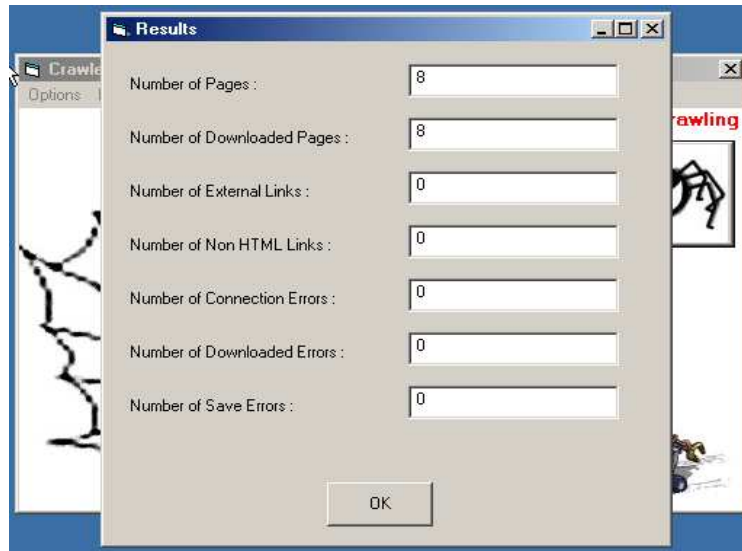


Figure (4.1) Crawler-Output Path

- **URL Startup File:** a text file that must be created by the SEA to store the most popular pages URL's including the Home page URL. The URLs must be separated by spaces. To set the path for this file to the crawler, from **options** choose **Set URL Startup File** and write the path for this file in the text box and click on ok.
- **Site Address:** which represent the site name of the Web site. For example if the Web site name was `http://www.mysite.com`, from **options** choose **Set the Site Address** and write “mysite” in the textbox and click on ok.
- **URL Prefix:** which is the prefix of the URL, for example the URL prefix for the site `http://www.mysite.com` is “`http://`”, to set this option then choose **Set the URL Prefix** from **Options** and write “`http://`”.

A special directory must be created in the Output Path and named “**DownloadedPages**”; all the Web pages will be store in that directory. To start the crawling process, click on the **spider** button on the crawler form.

To see the results of the crawling operation, choose **Show Results** from **Results** and write the path for the crawler results file and click on ok and the results will be shown as in figure (4.2).



To view the true (**Figure (4.2) Crawling Results**) he crawler, use the **Crawling List Viewer** program and click on **Get Crawling List** then enter the name of the crawling list file in the text box and click on ok as shown in figure (4.3) and the results in figure (4.4).

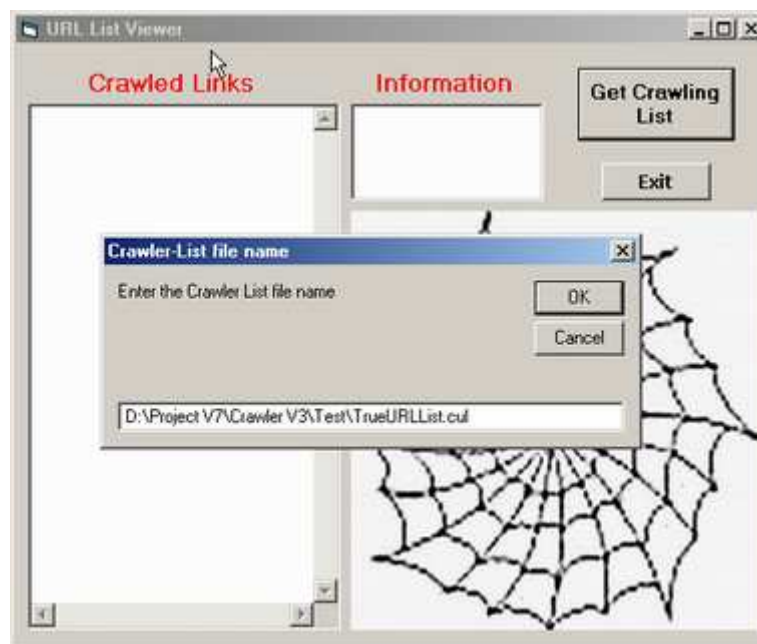


Figure (4.3) Crawling List Viewer

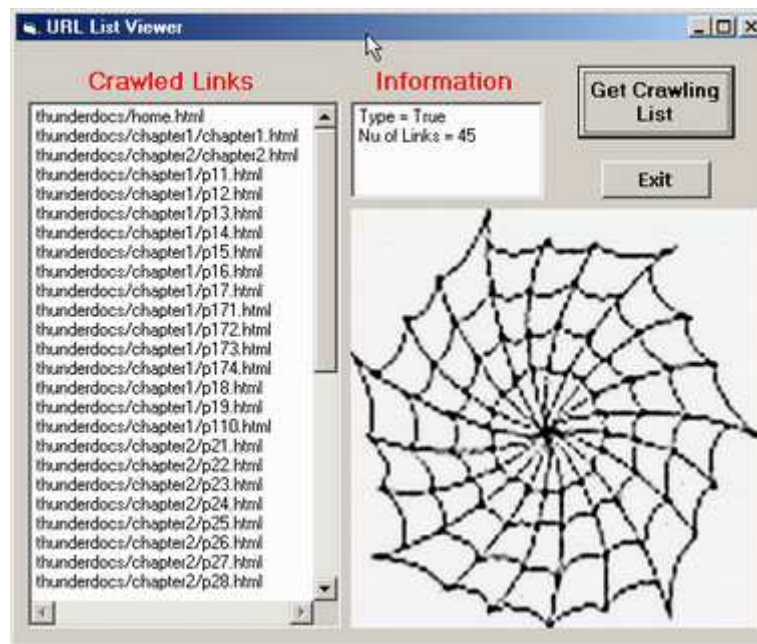


Figure (4.4) Crawling List Viewer-Results

4.3.2 Indexer User Interface

To run the indexer, first the SEA must create the stop word list, which is a text file containing the stop words separated by spaces. Then the SEA must convert the stop words text file to a stop words data file by selecting **Build Stop Words File** from the **Build** menu and write the stop words text file name in the text box then write the filename of the new stop words data file in the other text box as shown in figure (4.5) and figure (4.6).

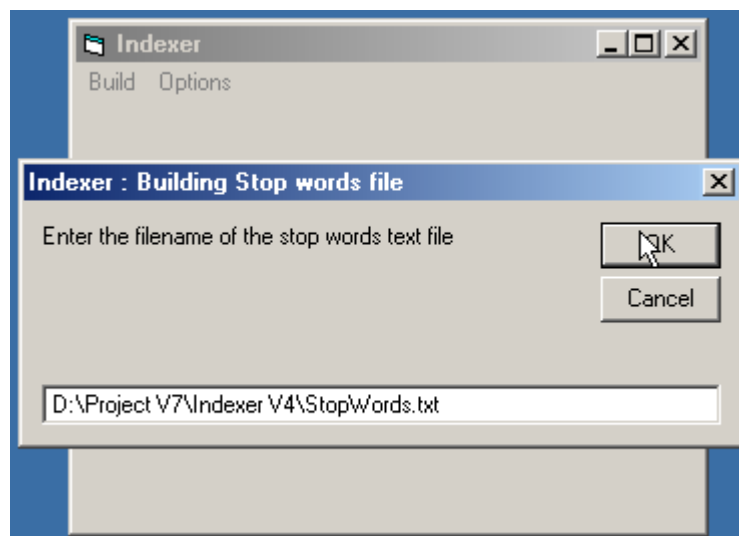


Figure (4.5) Indexer-Stop word text file

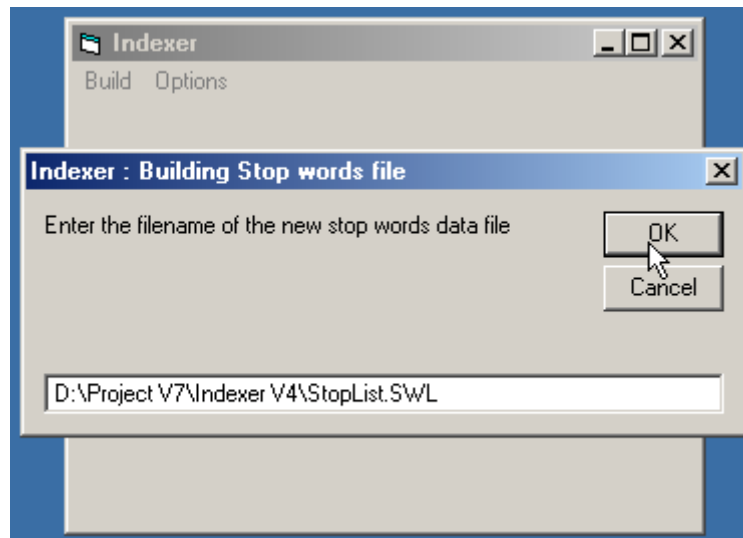


Figure (4.6) Indexer-Stop word data file

Now the SEA will have to set the Work Path by selecting **Set Work Path** from **Options** then write the work path in the text box and click on ok as shown in figure (4.7).

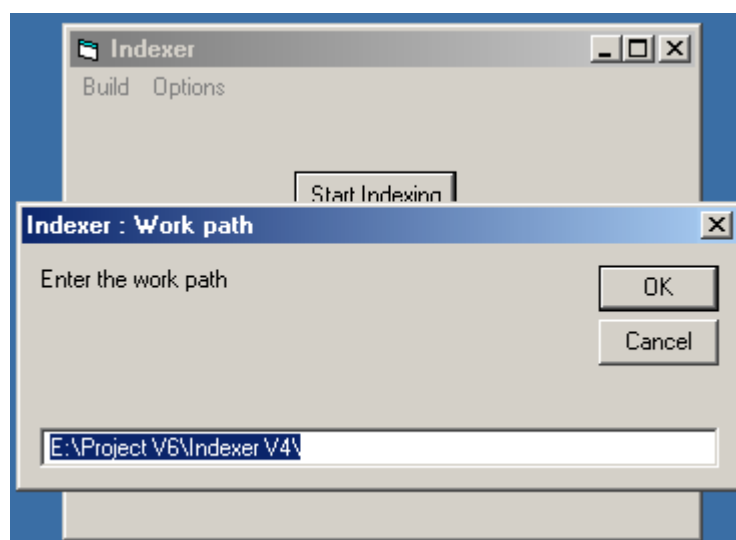


Figure (4.7) Indexer-Work Path

The SEA have to set the stop list file name by selecting **Set Stop List file name** from **Options** as shown in figure (4.8).

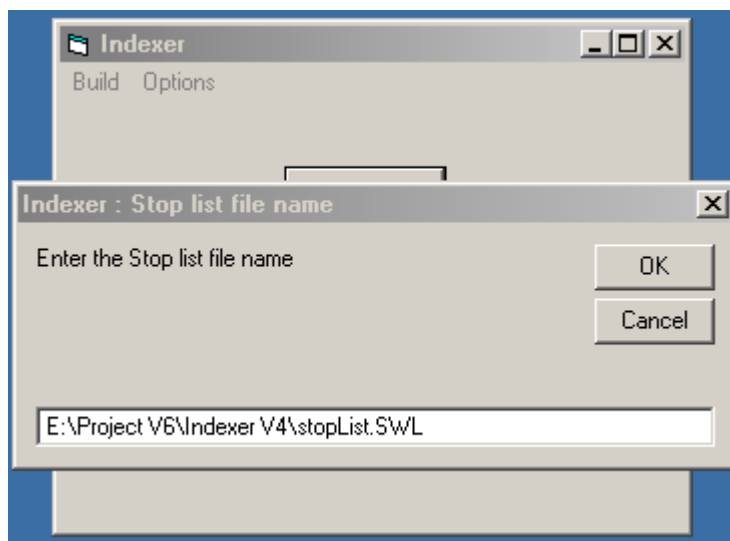


Figure (4.8) Indexer-Stop List

The **DownloadedPages** directory and the **TrueURLList** file must be in the same Work Path for the indexer. The **IndexFiles** directory must be created in the Work Path. To run the indexer, click on **Start Indexing** button then after finishing the indexing process click on the **Build Inverted Index** Button to build the inverted index and the lexicon.

4.3.3 Search Engine Server User Interface

The SES is responsible for serving all the requests coming from the clients. To put the SES on line, the SEA must choose **Change Path** from **Options** to set the path to the folder (directory) that contains the lexicon file, URL list file, word list file, inverted index files as shown in figure (4.9).

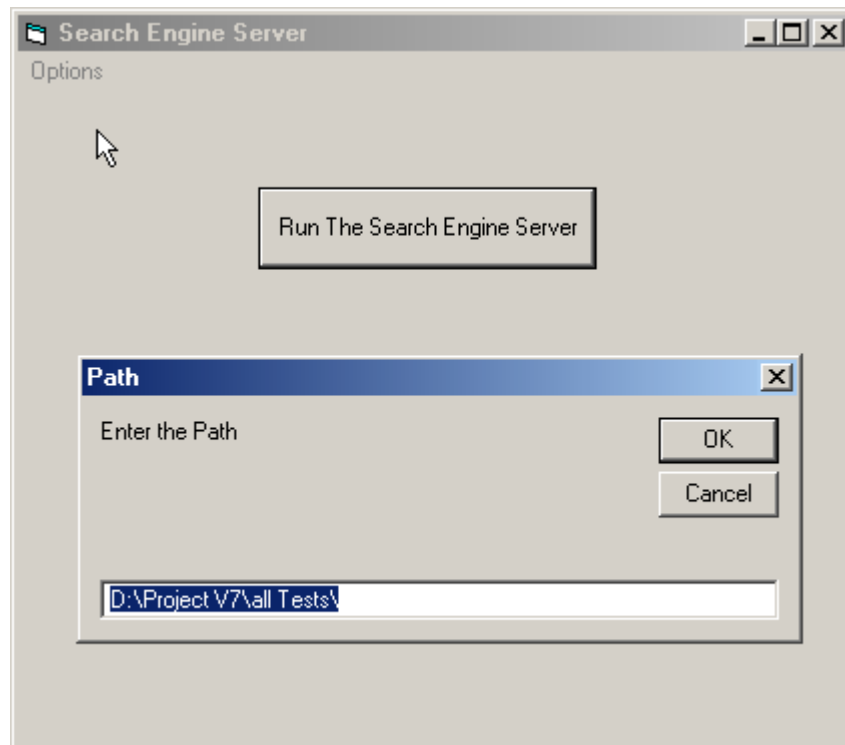


Figure (4.9) SES

To run the SES click on the **Run The Search Engine Server** button to put the SES on line and ready to any request.

4.3.4 Search Engine Web page Interface

The search engine Web page consists of a logo (thunder), text box, search button, and three links as shown in figure (4.10).



Figure (4.10) Search Engine Web page

The About page is used to give a description about the search engine as shown in figure (4.11)



Figure (4.11) About page

The Preferences page is used to change the number of results per page as shown in figure (4.12)



Figure (4.12) Preferences page

The documentation page give the documentation of the proposed system as a web site.

4.4 Features

There are many features a search engine can have, and the following are the most used features:

1. **Case Sensitive:** the proposed system is not case sensitive, because if the search engine is case sensitive then many results will be put aside because they are capital or small which will reduce the number of results.
2. **Deep Crawling:** the proposed system makes a deep crawling and downloads all the pages it can access (except for external and non-html links).
3. **Stop Word Indexing:** the proposed system excludes stop words from the index as a way to save storage space and to speed up the search process.
4. **Use of Meta tags:** the proposed system excludes meta tags from the index because many Web page designers put many keyword that are not related to the page in the meta tags to give their pages a higher ranking (priority).
5. **Links text indexing:** the proposed system indexes the links text.
6. **Ranking:** the proposed system uses the word frequency, word position, font attributes (size, color, style), and others (if the word is a link, title or a heading) to rank words.
7. **Spelling Correction, Stemming, and Abbreviation support:** some search engines have a spelling correction (type mathematic and the search engine will search for the correct word, which is mathematic), stemming (type swim and the search engine will search for swim, swims or swimming), and abbreviation support (type AI and the search

engine will search for AI or Artificial Intelligence). The proposed system does not use spelling correction, stemming or abbreviation.

8. **Filter support:** the proposed system has a filter used to ignore downloading the external and non-html links.
9. **Advance Search:** the proposed system support advance searching by providing Boolean search.
10. **Support for Language:** the proposed system search only English words.

4.5 System Requirement

There are two types of system requirement: Hardware Requirement and Software Requirement.

4.5.1 Hardware Requirement

The proposed Web site search engine requires to be online on the Internet or at least be implemented on a network.

To give an idea about the Hardware requirement of the server computer required for running the Web site search engine. Consider the following example:

The number of Web page in a Web site varies from tens of pages to a maximum of about 1,000,000 pages. In this example, the worst case (Maximum) will be taken.

Consider a Web site with 1,000,000 Web pages; the average length of URL string is 40 characters (URL string is like “home.html”, “products/new/newitems.htm”, “welcome.html”, “buildings/rooms/tables/newtables.htm”, and the like), average size of page s

1KB, average number of words in each page is 100 word, average number of characters in a word is 10 characters.

There are two parts in the search engine

Off-Line part:

The Off-Line part consists of the crawler and the indexer.

Crawler:

-Memory Requirement: the crawler requires memory space for the URL list during running time.

Memory space required by the crawler = (average size of URL string + 10) *
number of pages to be downloaded

$$\begin{aligned} &= (40 + 10) * 1000,000 \\ &= 50,000,000 \text{ Bytes} \\ &= 48 \text{ MB} \end{aligned}$$

The number 10 in the equation is the sum of: 4 Bytes for the left link, 4 Bytes for the right link, and 2 Bytes for the status of the URL link (downloaded or not).

-Disk Requirement: the crawler requires disk space to store the URL list and the downloaded pages (store as files).

Disk space required for storing the URL list = 6 + average size of URL string
* number of pages to be downloaded

$$\begin{aligned} &= 6 + 40 * 1000,000 \\ &= 40,000,00 \text{ Bytes} \\ &= 39 \text{ MB} \end{aligned}$$

Disk space required for storing the downloaded pages = number of pages to
be downloaded * average page size

$$\begin{aligned} &= 1000,000 * 1024 \\ &= 1,024,000,000 \text{ Bytes} \\ &= 977 \text{ MB} \end{aligned}$$

Disk space required by the crawler = Disk space required for the URL list +
Disk space required for the downloaded pages

$$= 39 \text{ MB} + 977 \text{ MB}$$

$$= 1016 \text{ MB (about 1 GB)}$$

Indexer:

The indexer consists of two parts: building the index and building the inverted index and lexicon.

Building the index:

-Memory Requirement:

Building the index requires memory space for the word list and additional memory space for another word list that will be stored in the disk and also used in building the inverted index and lexicon.

According to many research papers, the number of words in English language + scientific words is about 1000,000 word (see [Ask01]).

Memory space required for word list1 = (average word size + 12) * number of unique words in the Web site

$$= (10 + 12) * 1000,000$$

$$= 22,000,000$$

$$= 21 \text{ MB}$$

The number 12 in the equation is the sum of: 4 Bytes for the left link, 4 Bytes for the right link, and 4 Bytes for the number of word occurrences.

Memory space required for word list2 = (average word size + 4) * number of pages to be downloaded

$$= (10 + 4) * 1000,000$$

$$= 14,000,000 \text{ Bytes}$$

$$= 14 \text{ MB}$$

The number 4 in the equation is 4 Bytes for the number of word occurrences.

Memory space required to build the index = memory space required for word list1 + memory space required for word list2

$$= 21 \text{ MB} + 14 \text{ MB}$$

$$= 35 \text{ MB}$$

-Disk Requirement:

Building the index requires disk space for all the 36 index files (size of index), and for the word list.

Disk space required for storing the word list = (average word size + 4) * number of pages to be downloaded

$$= 14 * 1000,000$$

$$= 14,000,000 \text{ Bytes}$$

$$= 14 \text{ MB}$$

Disk space required for storing the index = 14 * average number of words in page * number of pages to be downloaded

$$= 14 * 100 * 1000,000$$

$$= 1,400,000,000 \text{ Bytes}$$

$$= 1336 \text{ MB}$$

Disk space required for the indexer = disk space required for storing the word list + disk space required for storing the index

$$= 14 \text{ MB} + 1336 \text{ MB}$$

$$= 1350 \text{ MB}$$

Building the inverted index and lexicon:

-Memory Requirement:

Building the inverted index and lexicon requires memory space for holding one of the index files to sort it, word list2 created while building the index (14 MB), and the multilayer structure used for the lexicon.

Memory space required by the multilayer structure requires = $(36 + 36^2 + 36^3 + 36^4) * 4$

$$= 6,910,416 \text{ MB}$$

$$= 7 \text{ MB}$$

The index is spread in to 36 files, which means that theoretically, each file size = index size / 36

$$= 1336 \text{ MB} / 36$$

$$= 38 \text{ MB}$$

But according to English language statistics, the words that begins with the letter ‘e’ take 11% of the number of English words as shown in table (4.1)

[Ask05]

Table (4.1) Letter frequency distributions in English

E	11.1607%	56.88	M	3.0129%	15.36
A	8.4966%	43.31	H	3.0034%	15.31
R	7.5809%	38.64	G	2.4705%	12.59
I	7.5448%	38.45	B	2.0720%	10.56
O	7.1635%	36.51	F	1.8121%	9.24
T	6.9509%	35.43	Y	1.7779%	9.06
N	6.6544%	33.92	W	1.2899%	6.57
S	5.7351%	29.23	K	1.1016%	5.61
L	5.4893%	27.98	V	1.0074%	5.13
C	4.5388%	23.13	X	0.2902%	1.48
U	3.6308%	18.51	Z	0.2722%	1.39
D	3.3844%	17.25	J	0.1965%	1.00
P	3.1671%	16.14	Q	0.1962%	(1)

Which means, that the index files will not be equal in size, in which the ‘e’ index file could be the largest index file according to table (4.1), which takes 11% of the index.

$$\text{Largest index file size} = \text{index size} * 11\%$$

$$= 1336 \text{ MB} * 11\%$$

$$= 147 \text{ MB}$$

Memory space required for building the inverted index and lexicon:

$$= 147 \text{ MB} + 14 \text{ MB} + 7 \text{ MB}$$

$$= 168 \text{ MB}$$

-Disk Requirement:

Building the inverted index and lexicon requires disk space for storing the lexicon.

Disk space required for building the inverted index and lexicon = 7 MB

Memory required for the Off-Line phase = 168 MB

Disk space required for the Off-Line phase = disk space required for the crawler + disk space required for the indexer

$$= 1016 \text{ MB} + (1350 \text{ MB} + 7 \text{ MB})$$

$$= 2373 \text{ MB}$$

On-Line part:

The On-Line phase consists of two parts: the searcher and the user interface page.

Searcher:

The searcher runs on the server-side (server computer).

-Memory Requirement:

The searcher requires memory space for lexicon, the word list, and the URL list.

Memory space required for the searcher = 7 MB + 14 MB + 39 MB

$$= 60 \text{ MB}$$

-Disk Requirement:

No disk space required.

User interface pages:

The user interface pages are viewed by the user in the client-side (user computer).

-Memory Requirement:

Only few Kilobytes are required for the Web site search engine Web page and for the results Web page.

-Disk Requirement:

No disk space required.

Full Hardware Requirement:

Server computer with 168 MB (256 MB) of memory and 2373 MB (3 GB) of disk space.

4.5.2 Software Requirement

The user side (it is the user computer) requires any Internet browser to open the Web site search engine page and to browse the results of the search.

4.6 Experiment and Results

Evaluating of search results is to measure how well the returned results meet the user's particular information need. Precision metric will be used to evaluate the performance of the proposed system.

Precision: the precision is to measure how much of what the users see is relevant. This measure is defines as:

$$\text{Precision} = \frac{C}{R}$$

Where C is the number of relevant documents retrieved by the search engine. R is the number of documents retrieved by the search engine.

For example, if the user search for the word “car”, and the results contain 40 pages, 20 of them are relevant then

$$\text{Precision} = \frac{20}{40} = 0.5 \text{ or } 50\%$$

For testing the proposed system, a Web site is created, which contains the first two chapter of this thesis (Chapter1 and Chapter2), this Web site is implemented on a local server, crawled, and indexed, which contains 45 pages. 5 different queries were tested and evaluated using the precision of the first 10 retrieved results. The results are shown in table (4.2).

Table (4.2) Experiment results

Query	Results	Precision
1. Internet	1, 2, 3, 9, 10 very Relevant 5, 6, 8 Relevant 4,7 Irrelevant	80%
2. Crawler	1, 2, 3, 7, 8, 9, 10 very Relevant 4, 5, 6 Relevant	100%
3. Search and Engine	1, 2, 3, 7, 8 very Relevant 4, 5, 9 Relevant 6, 10 Irrelevant	80%
4- HTML	1, 3, 4, 6, 9, 10 very Relevant 2, 5, 7, 8 Relevant	100%
5- Search and Query	1, 7, 8, 10 very Relevant 2, 5 Relevant 3, 4, 6, 9 Irrelevant	60%

4.7 Tools

Two tools are designed to be used with the proposed Web site search engine: URL list viewer and Word list viewer.

URL list viewer is a crawler tool used for displaying the URL list generated by the crawler, this tool is shown in figure (4.13).

Word list viewer is an indexer tool used for displaying the Word list generated by the indexer, this tool is shown in figure (4.14).

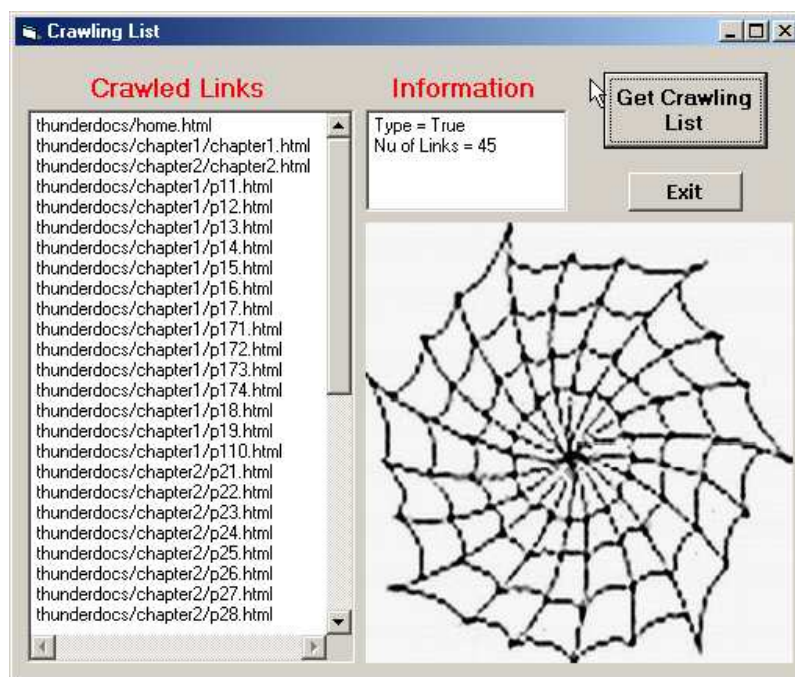


Figure (4.13) URL List Viewer

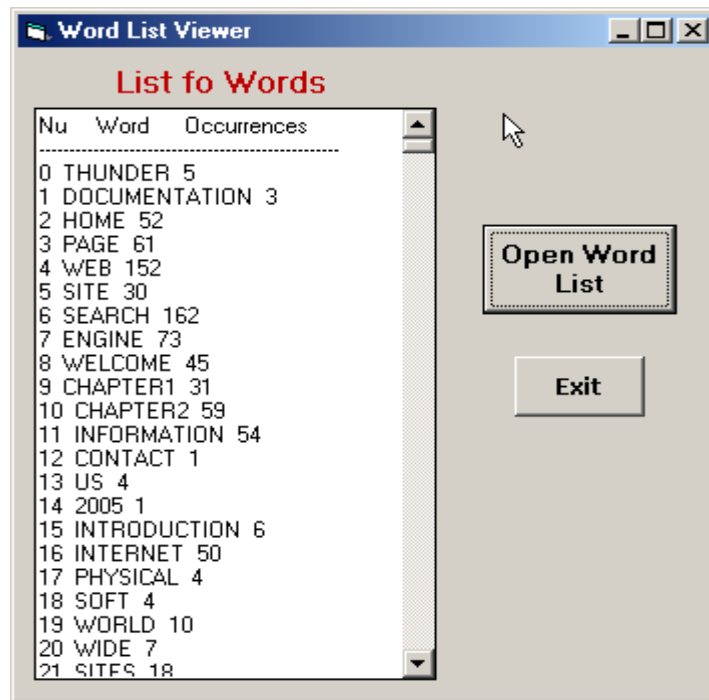


Figure (4.14) Word List Viewer

Chapter 5

Conclusions and Future Works

5.1 Conclusions

- 1- The index is spread in 36 files, which reduce the sort time and memory space required to build the inverted index.
- 2- The lexicon is created using a multilayer structure with 4 layers instead of the binary search tree.
- 3- The ranking part based only on the word attributes of the pages, which can be more efficient if the structure of the web (links) are used.

5.2 Suggestions for Future Works

1. Research on the user needs, the problems they face when using the search engine and all the ideas they have or want to be in the search engine.
2. Research on finding the best stop words to be used in the search engine.
3. Develop a NLP query interface for the search engine.
4. Develop a search engine that index non-html documents like PDF, Microsoft Word and other types of documents.
5. Design a search engine with a different interface technique between the client request and the search program that is faster than the socket interface.
6. Develop a ranking algorithm using the structure of the web (links).

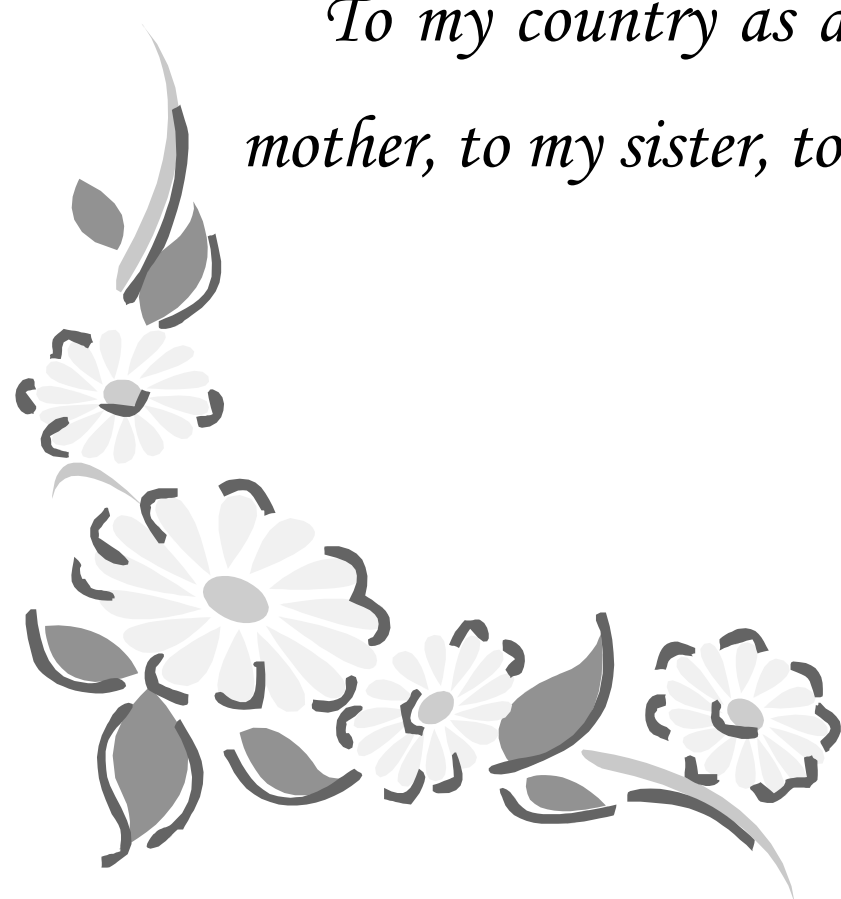
Dedication

I dedicate my work to all the researchers and scientists who use the science to make the world a better place.

To all the people who sacrifice in their lives for a better future for their country and for their children.

To my country as a simple gift, to my mother, to my sister, to my family.

Eihab



Keywords

Internet, Soft Internet, Web, WWW (World Wide Web), Web site Search engine, Crawler, Indexer, Searcher, Visual Basic, CGI (Common Gateway Interface), HTML (Hyper Text Markup Language), Browser, Web Growth, Starting Point Pages, Subject Trees, URL (Uniform Resource Locator), TCP/IP (Transmission Control Protocol/Internet Protocol), Client-Server Model, Web Servers, HTTP (Hyper Text Transfer Protocol), IR (Information Retrieval), DR (Data Retrieval), Characteristics of the Web, Difficulties of the Web, Types of search engines, Architecture of search engine, search engine user interface.

List of Abbreviations

Abbreviation	Meaning
AI	Artificial Intelligence
ARPA	Advanced Research Projects Agency
ASP	Active Server Page
BOOTP	Boot Strap Protocol
CERN	European laboratory for particle physics
CGI	Common Gateway Interface
CGI-SES	Common Gateway Interface-Search Engine Server
DNS	Domain Name System
DR	Data Retrieval
E-Mail	Electronic Mail
ESMTP	Extended Simple Mail Transfer Protocol
FAQs	Frequently Asked Questions
FDDI	Fiber Distributed Data Interface
FTP	File Transfer Protocol
GIF	Graphic Interchange Format
GML	Generalized Markup Language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IAB	Internet Architecture Board
ICMP	Internet Control Message Protocol
ICS	Information and Computer Science
IP	Internet Protocol
IR	Information Retrieval

ISO	International Standards Organization
JPG	Joint Photographic Group
LAN	Local Area Network
MTA	Mail Transport Agents
NCP	Network Control Protocol
NLP	Natural Language Processing
PDF	Portable Document Format
PHP	Personal Home Page
SEA	Search Engine Administrator
SES	Search Engine Server
SGML	Standard Generalized Markup Language
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
TF-IDF	Term Frequency-Inverse Document Frequency
TFTP	Trivial File Transfer Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WWW	World Wide Web

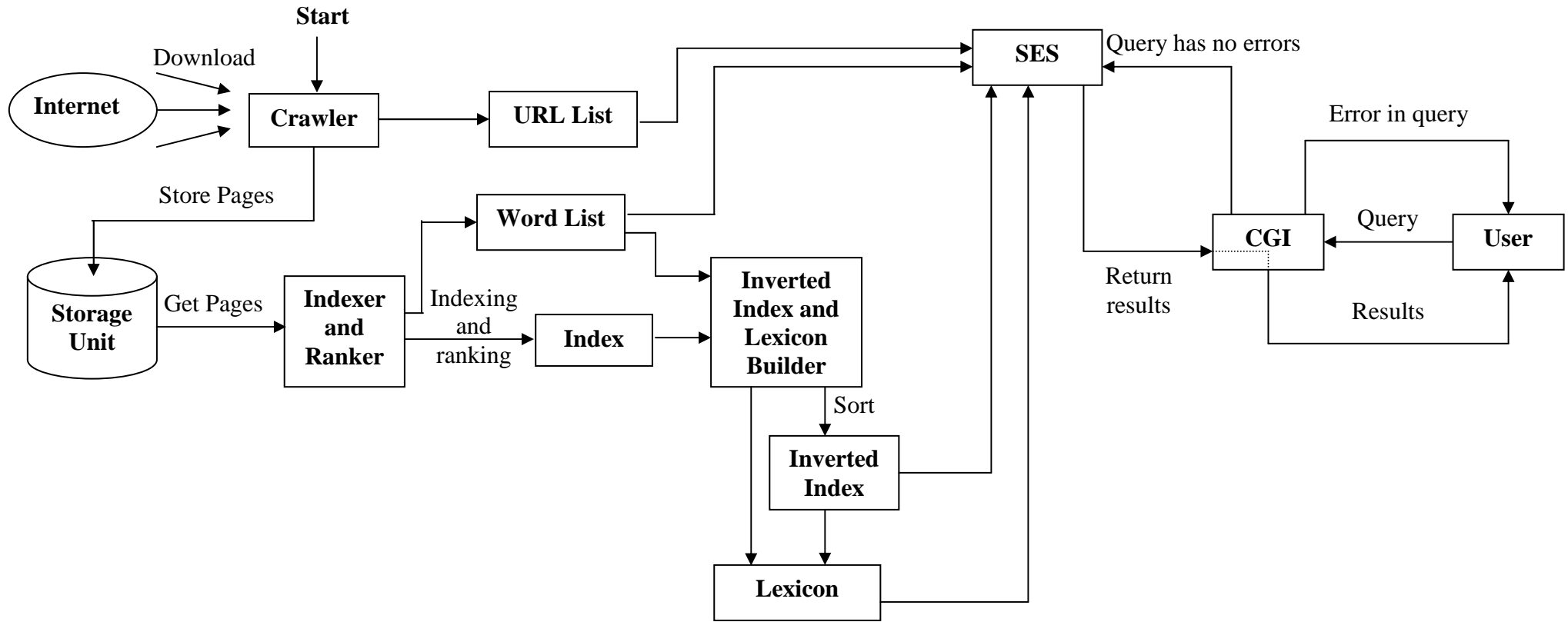


Figure (3.1) Proposed Web Site Search Engine Architecture

Design and Implementation of an Internet Search Engine





Introduction

The amount of information on the Web is growing rapidly, as well as the number of new users inexperienced in the art of Web research. The Web is enormous and growing at an incredibly fast pace. It has been said that if the user spent only one minute per page, 10 hours a day, it would take four-and-a-half years to explore only 1 million Web pages. Thus a real need exists for some way to search this huge resource.

Difficulties of the Web

The Web creates new challenges for the information retrieval. And these problems are:

1- Distributed data: data is distributed widely in the world. It is located at different sites and platforms.



The communication links between computers vary widely. Plus, there is no topology of data organization.

2- Large volume: the growth of data is exponential. It poses issues that are difficult to cope with.

3- High percentage of volatile data: documents can be added or removed easily in the World Wide Web. Changes to these documents go unnoticed by others. 40% of Web pages change every month. There is a very high chance of dangling links.

4- Unstructured and redundant data: the Web is not exactly a distributed hypertext. It is impossible to organize and add consistency to the data and the hyperlinks. Web pages are not well structured and 30% of all Web pages are duplicated. Semantic redundancy can increase traffic.



5- Quality of data: a lot of Web pages do not involve any editorial process. That means data can be false, inaccurate, outdated, or poorly written.

6- Heterogeneous data: data on the Web are heterogeneous. They are written in different formats, media types, and natural languages.

7- Dynamic data: the content of Web document changes dynamically. The content can be changed by a program such as hit counter that keep tracks of number of hits.



Web Search Solutions


There are three main solutions for searching the Web:

1- Starting Points Pages

These pages provide a good point of entry for people accessing the Web for the first time. They include lots of links that introduce the user to fundamental Web concepts, such as “What is a Browser?”, the user can also find links to Subject Trees, Search Engines, Usenet **FAQs (Frequently Asked Questions)**, Internet documents of all kinds, and “What’s new” documents.

2- Subject Trees

A subject tree is a subject-oriented catalog of URLs organized by topic. It is an alphabetically organized list of selected Web resources that is usually organized with



major headings such as Arts and Humanities, Business, Economy, Government, and the like.

3- Search Engines

A search engine is a software used to crawl the Web for downloading the Web pages, index them, rank them and build a database for these pages ready to be searched for satisfying the user queries.

There are two types of search engine on the Web:

1- Web Site Search Engine

2- Internet Search Engine

While the Internet search engine search the entire Web looking for answers to user's query, the Web Site search engine is dedicated to one Web Site, searching only that Web Site.



Web Site Search Engine

The Web Site search engine consists of the following components:

- 1- Crawler
- 2- Indexer
- 3- Ranker
- 4- Searcher

The Web Site search engine can be divided in to two phases:

- 1- Off-Line phase
- 2- On-Line phase



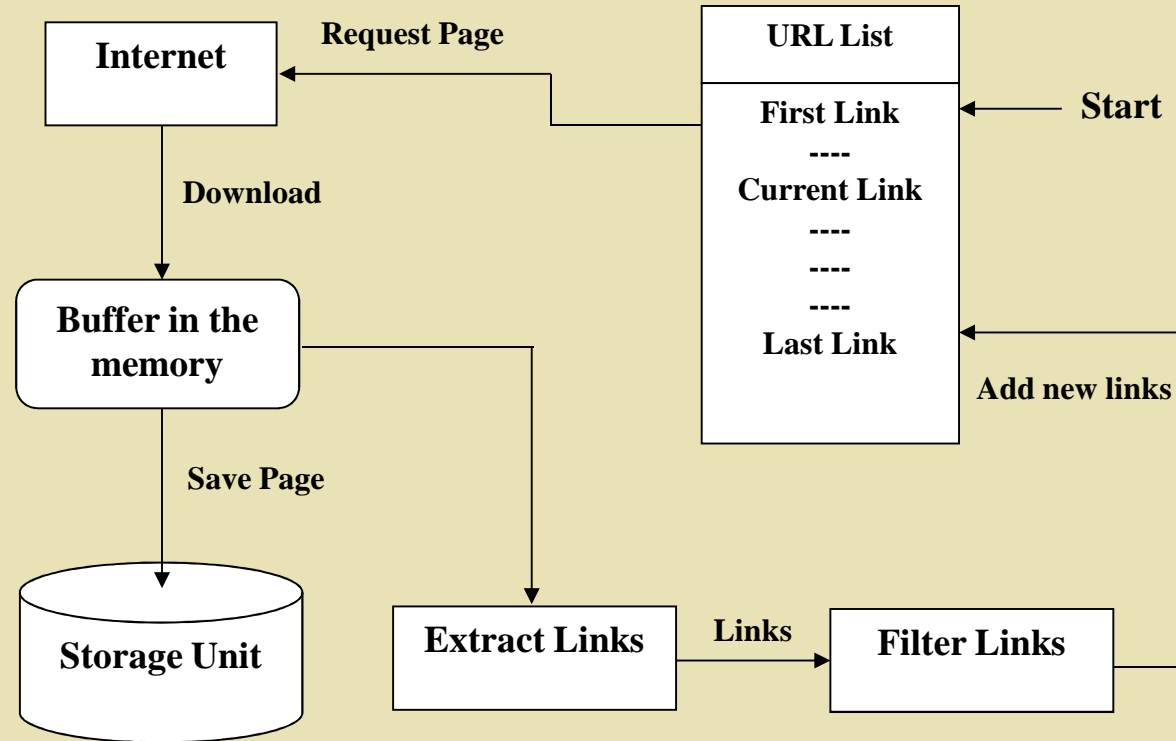
Off-Line phase

The Off-Line phase consists of:

Crawler

The crawler is responsible for downloading all the pages in the Web Site (Except for the non-HTML pages) and store them as files in the storage unit. The crawler consists of a URL list, downloading program and a URL extractor.

Before running the crawler, the crawler designer will have to initialize the URL list with some URLs (Home page and other frequently requested URLs) as shown in the figure in the next slide.



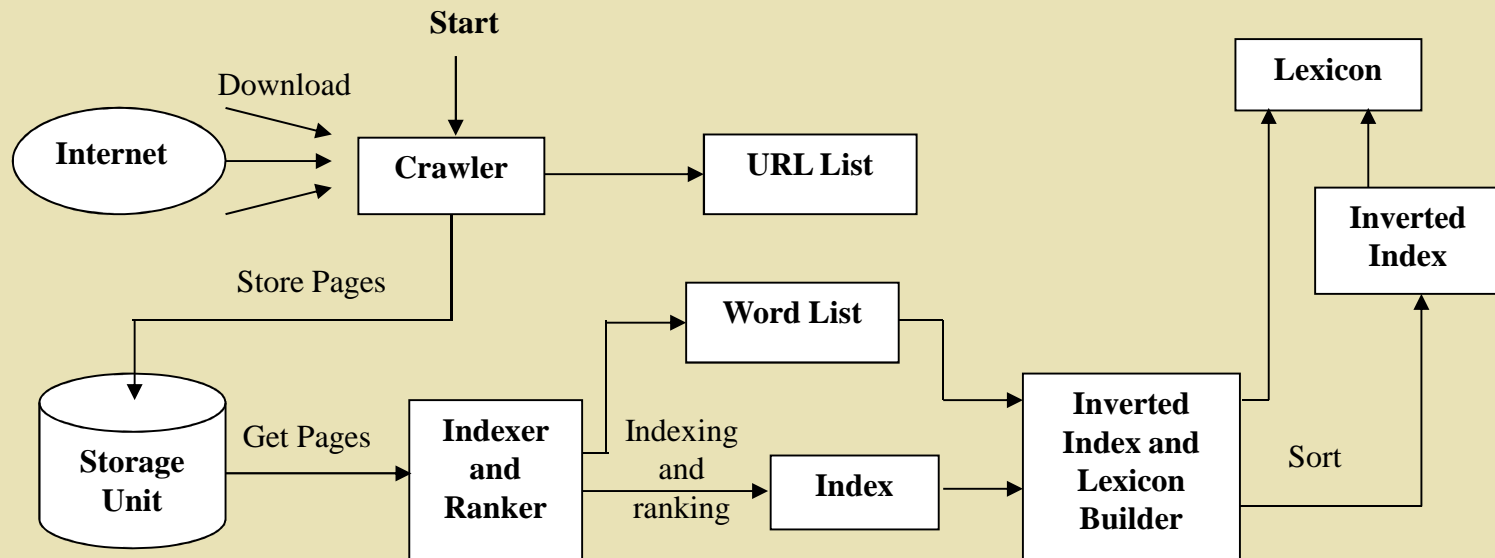
Indexer

The indexer is responsible for parsing all the files (pages) downloaded by the crawler and index all the words in these files (except the stop words) and sort them to create the inverted index, then create the lexicon. The ranking is part of the indexing phase.



Tag	Description	Rank Score
B, STRONG, and BLINK	Change the word style	4
I, U, S, STRIKE, CODE, SAMP, VAR, EM, BLOCKQUOTE, TT, CITE, ADDRESS, SUB, SUP and KBD	Change the word style	2
BIG	Change the word style and size	12
MARQUEE	Convert the word to a scrolling word	6
TITLE	Change the word to a title	128
A	Change the word to a link	64
H1, H2, H3, H4, H5 and H6	Change the word to a heading	32
FONT	Change the word font style, color, or size	8

Off-Line means that till this moment the search engine is not ready to serve the users. The Off-Line phase is implemented on the search engine server computer and represent the first phase as shown in the following figure:





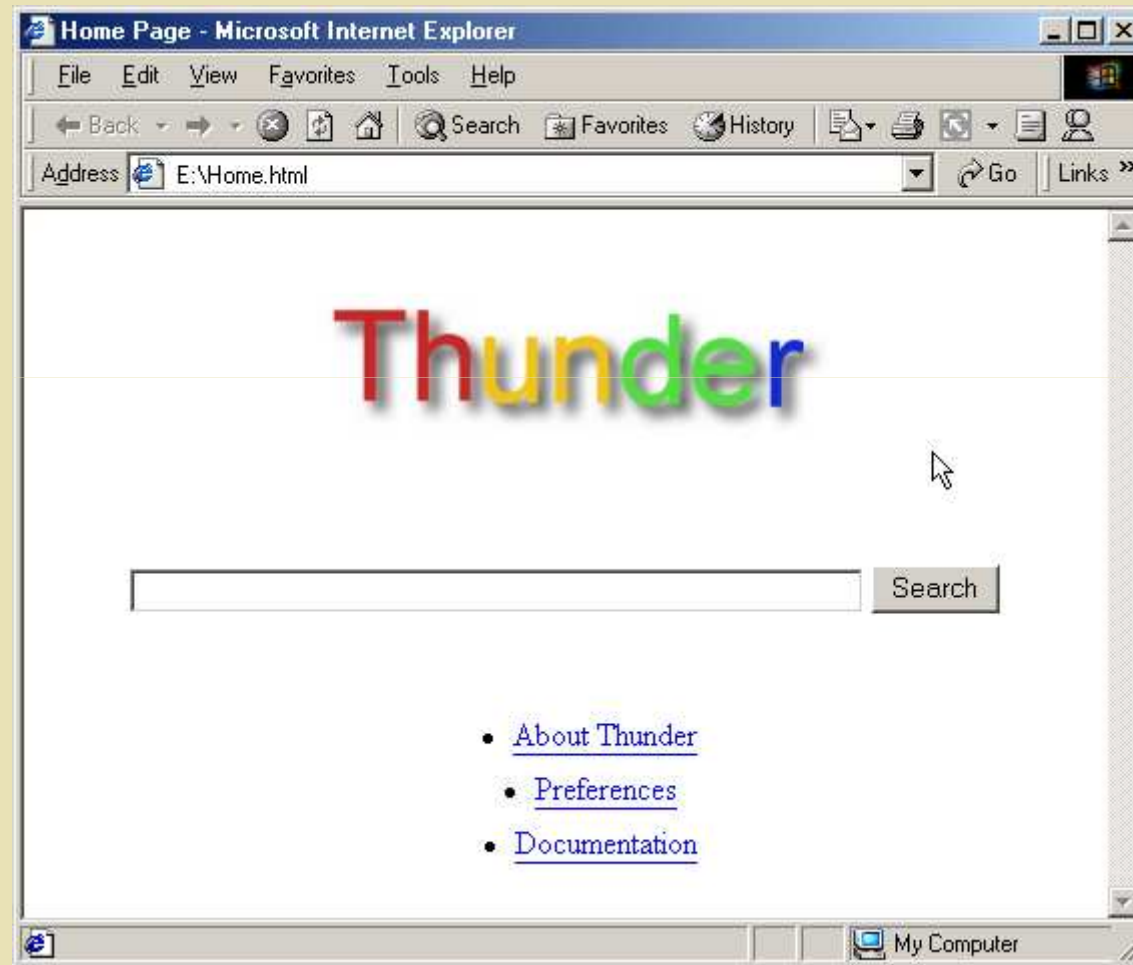
On-Line phase

The On-Line phase consists of:

Search Engine Interface

which consists of a Web page containing a text box and a button (search button), so that the user can write the query in the text box and click on the search button to start searching for the query and return the results. When the user write the address of the search engine on the address bar of the browser and press enter, the browser will display the search engine Web page by downloading it from the server of the search engine to the client computer (user).

The following figure is the user interface of the search engine:





CGI script

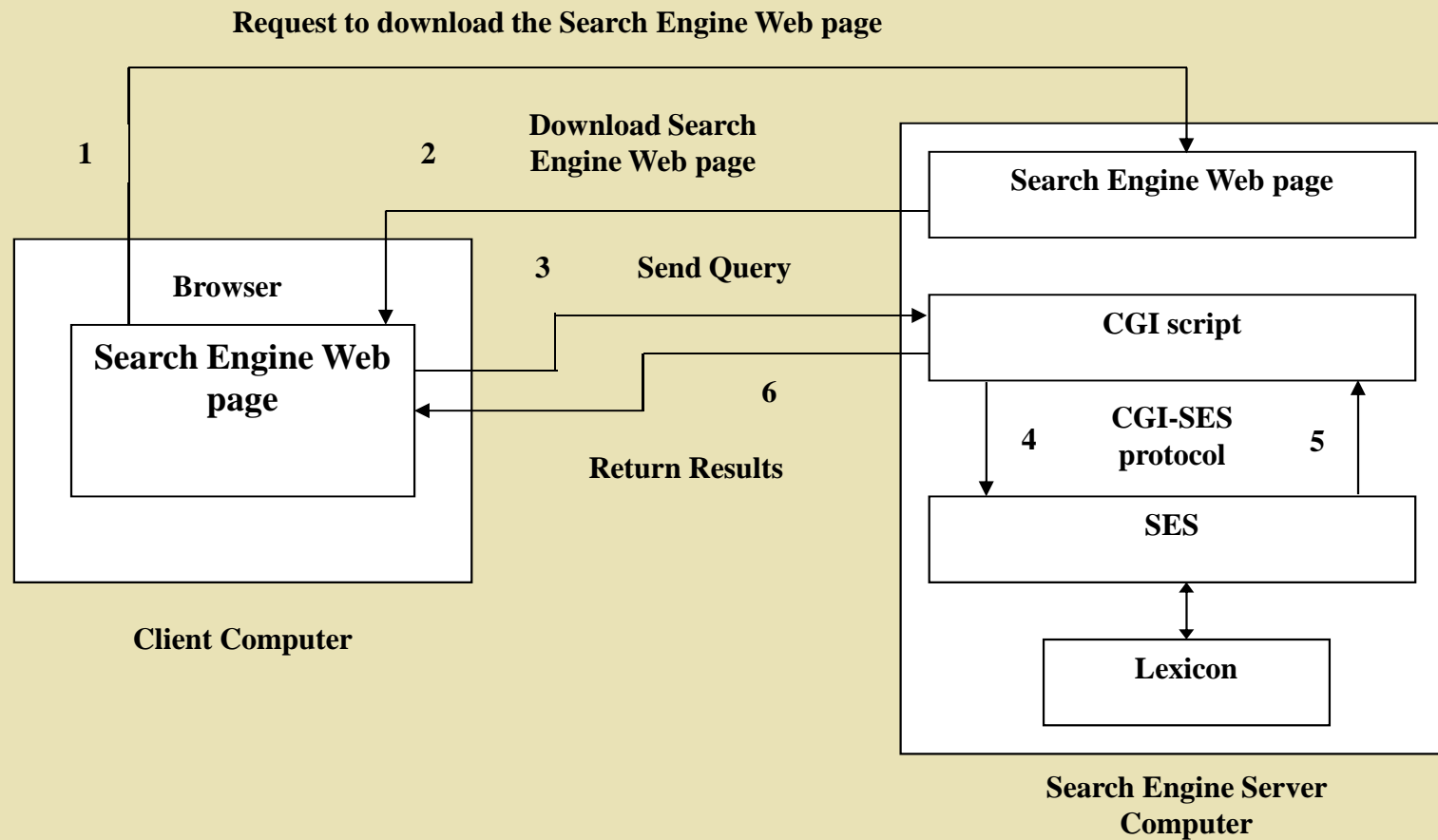
when the user click on the search button of the search engine Web page, the client computer will connect to the computer of the search engine and request to execute the CGI script of the search engine. The CGI script will perform a query processing to check the query and if there are no errors, the CGI script will pass the query to the SES (Search Engine Server) using the CGI-SES (Common Gateway Interface-Search Engine Server) protocol.



SES (Search Engine Server)

A program that runs on the server computer of the search engine and waits for requests from any search engine CGI script. When the SES receives a request from a CGI script, the SES will search the lexicon for the query and send the results to the CGI script, which will send them to the client computer (user).

The On-Line phase is shown in the following figure:





Conclusions

- 1- The crawler uses only one list with two pointers to keep track of the downloading process, which will add some extra speed over crawlers that uses two lists.
- 2- The index is spread in 36 files, which reduce the sort time and memory space required to build the inverted index.
- 3- The lexicon is created using a multilayer structure with 4 layers instead of the binary search tree, which provide a very fast access to the words required in the search process.
- 4- Using only a ranking system based on the word attributes on the page does not give the required results; the structure of the web (links) should be used to give more power to the ranking system.



Suggestions for Future Work

- 1- Research on the user needs, the problems they face when using the search engine and all the ideas they have or want to be in the search engine.
- 2- Research on finding the best stop words to be used in the search engine.
- 3- Research on spelling correction, stemming, abbreviation and phrase search features.
- 4- Develop a NLP query interface for the search engine.
- 5- Develop a search engine that index non-html documents like PDF, Microsoft Word and other types of documents.
- 6- Design a search engine with a different interface technique between the client request and the search program that is faster than the socket interface.



7- Add the title and date of publication to the URL list to this proposed system.

8- Design a search engine capable of searching in more than one language.

9- Design an Internet Crawler.

10- Develop a ranking algorithm using the structure of the web (links).

11- Develop a search engine that search for images (Content Based Image Retrieval).



Thank you
For
your listening

Design and Implementation of an Internet Search Engine





Introduction

The amount of information on the Web is growing rapidly, as well as the number of new users inexperienced in the art of Web research. The Web is enormous and growing at an incredibly fast pace. It has been said that if the user spent only one minute per page, 10 hours a day, it would take four-and-a-half years to explore only 1 million Web pages. Thus a real need exists for some way to search this huge resource.

Difficulties of the Web

The Web creates new challenges for the information retrieval. And these problems are:

1- Distributed data: data is distributed widely in the world. It is located at different sites and platforms.



The communication links between computers vary widely. Plus, there is no topology of data organization.

2- Large volume: the growth of data is exponential. It poses issues that are difficult to cope with.

3- High percentage of volatile data: documents can be added or removed easily in the World Wide Web. Changes to these documents go unnoticed by others. 40% of Web pages change every month. There is a very high chance of dangling links.

4- Unstructured and redundant data: the Web is not exactly a distributed hypertext. It is impossible to organize and add consistency to the data and the hyperlinks. Web pages are not well structured and 30% of all Web pages are duplicated. Semantic redundancy can increase traffic.



5- Quality of data: a lot of Web pages do not involve any editorial process. That means data can be false, inaccurate, outdated, or poorly written.

6- Heterogeneous data: data on the Web are heterogeneous. They are written in different formats, media types, and natural languages.

7- Dynamic data: the content of Web document changes dynamically. The content can be changed by a program such as hit counter that keep tracks of number of hits.



Web Search Solutions


There are three main solutions for searching the Web:

1- Starting Points Pages

These pages provide a good point of entry for people accessing the Web for the first time. They include lots of links that introduce the user to fundamental Web concepts, such as “What is a Browser?”, the user can also find links to Subject Trees, Search Engines, Usenet **FAQs (Frequently Asked Questions)**, Internet documents of all kinds, and “What’s new” documents.

2- Subject Trees

A subject tree is a subject-oriented catalog of URLs organized by topic. It is an alphabetically organized list of selected Web resources that is usually organized with



major headings such as Arts and Humanities, Business, Economy, Government, and the like.

3- Search Engines

A search engine is a software used to crawl the Web for downloading the Web pages, index them, rank them and build a database for these pages ready to be searched for satisfying the user queries.

There are two types of search engine on the Web:

1- Web Site Search Engine

2- Internet Search Engine

While the Internet search engine search the entire Web looking for answers to user's query, the Web Site search engine is dedicated to one Web Site, searching only that Web Site.



Web Site Search Engine

The Web Site search engine consists of the following components:

- 1- Crawler
- 2- Indexer
- 3- Ranker
- 4- Searcher

The Web Site search engine can be divided in to two phases:

- 1- Off-Line phase
- 2- On-Line phase



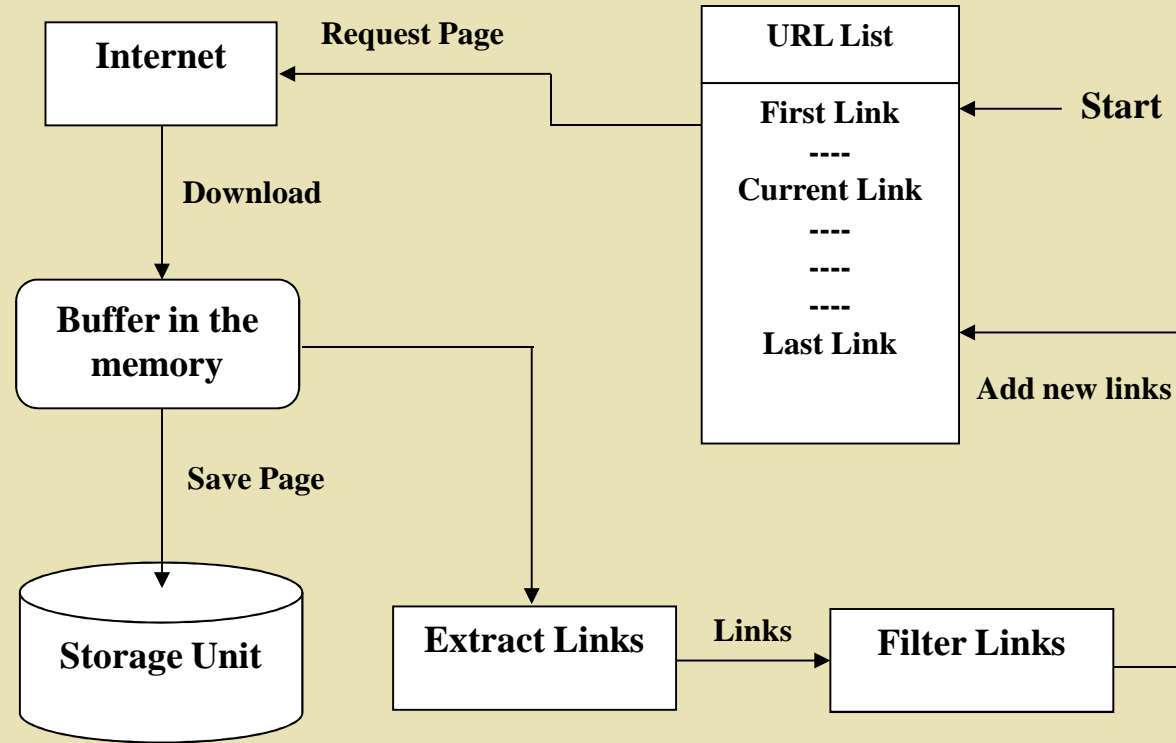
Off-Line phase

The Off-Line phase consists of:

Crawler

The crawler is responsible for downloading all the pages in the Web Site (Except for the non-HTML pages) and store them as files in the storage unit. The crawler consists of a URL list, downloading program and a URL extractor.

Before running the crawler, the crawler designer will have to initialize the URL list with some URLs (Home page and other frequently requested URLs) as shown in the figure in the next slide.



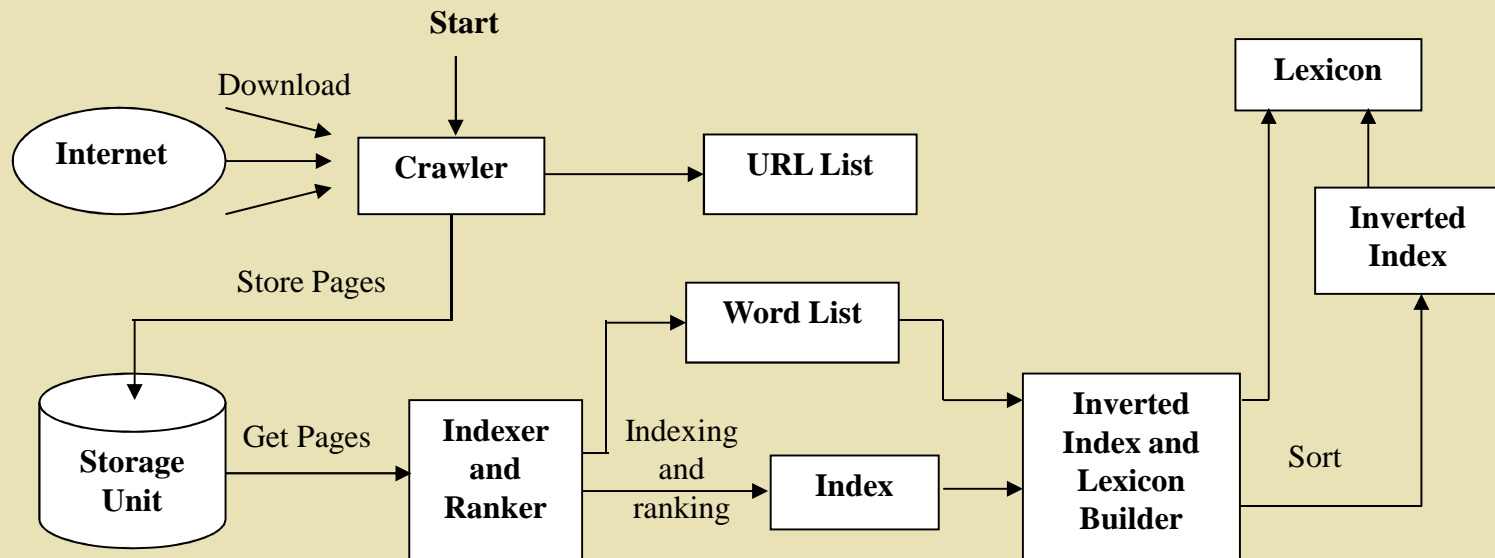
Indexer

The indexer is responsible for parsing all the files (pages) downloaded by the crawler and index all the words in these files (except the stop words) and sort them to create the inverted index, then create the lexicon. The ranking is part of the indexing phase.



Tag	Description	Rank Score
B, STRONG, and BLINK	Change the word style	4
I, U, S, STRIKE, CODE, SAMP, VAR, EM, BLOCKQUOTE, TT, CITE, ADDRESS, SUB, SUP and KBD	Change the word style	2
BIG	Change the word style and size	12
MARQUEE	Convert the word to a scrolling word	6
TITLE	Change the word to a title	128
A	Change the word to a link	64
H1, H2, H3, H4, H5 and H6	Change the word to a heading	32
FONT	Change the word font style, color, or size	8

Off-Line means that till this moment the search engine is not ready to serve the users. The Off-Line phase is implemented on the search engine server computer and represent the first phase as shown in the following figure:





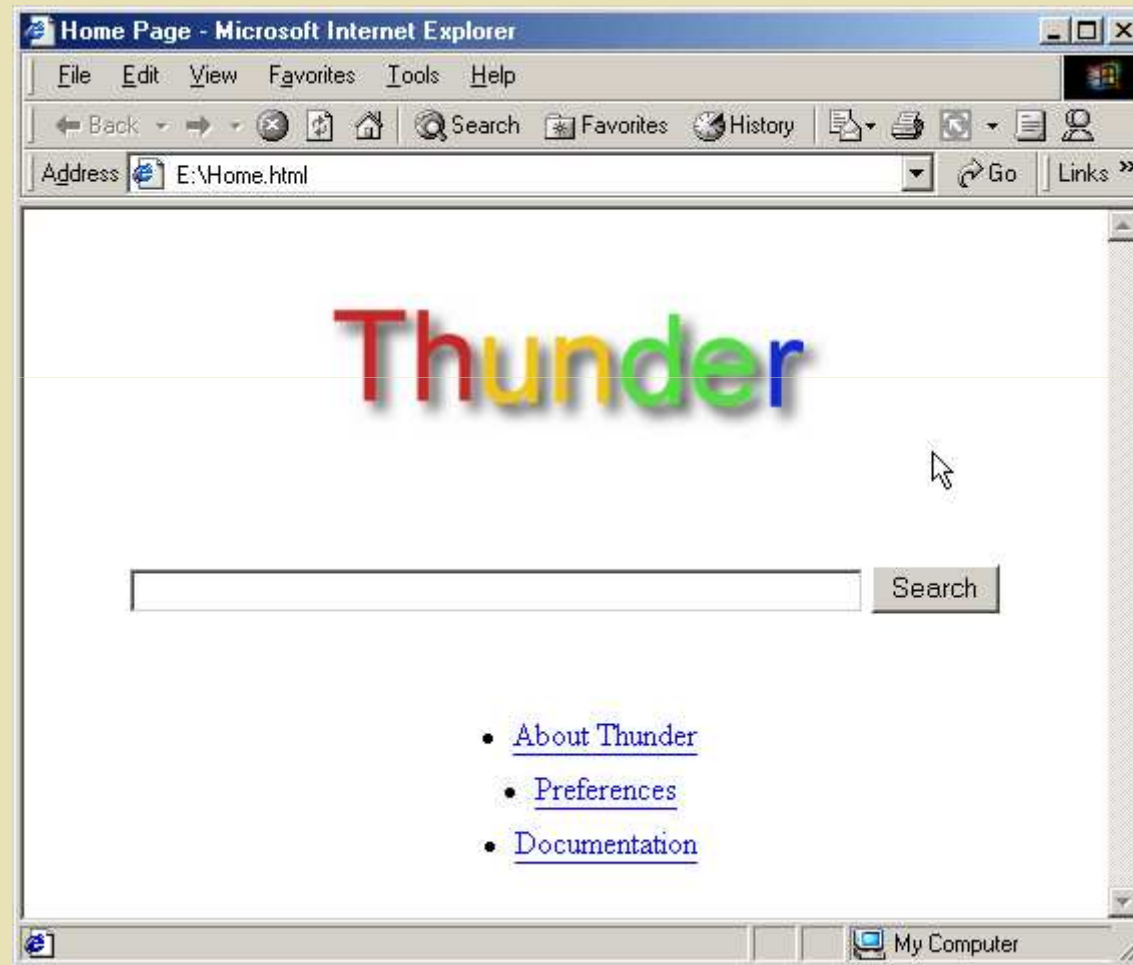
On-Line phase

The On-Line phase consists of:

Search Engine Interface

which consists of a Web page containing a text box and a button (search button), so that the user can write the query in the text box and click on the search button to start searching for the query and return the results. When the user write the address of the search engine on the address bar of the browser and press enter, the browser will display the search engine Web page by downloading it from the server of the search engine to the client computer (user).

The following figure is the user interface of the search engine:





CGI script

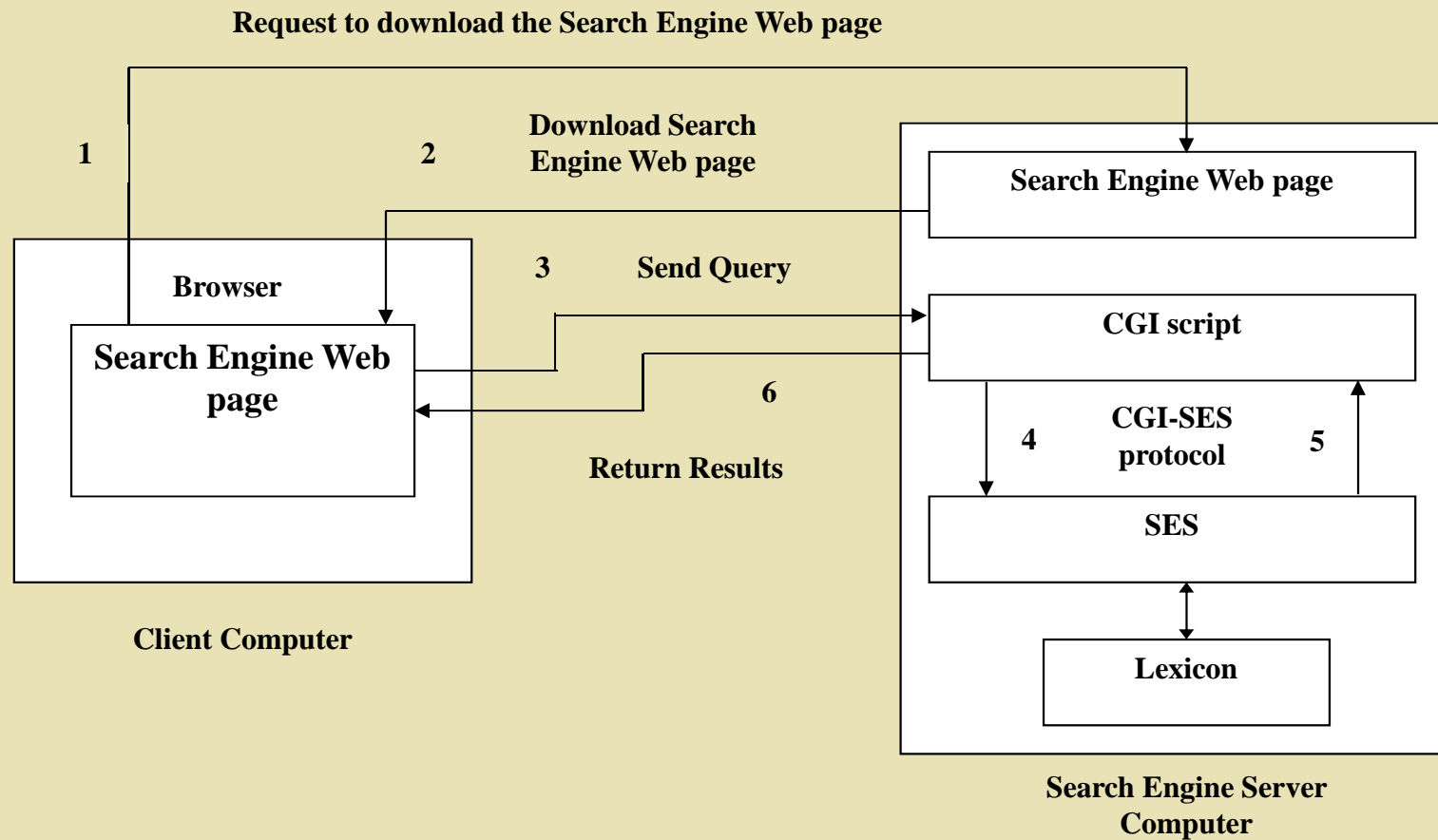
when the user click on the search button of the search engine Web page, the client computer will connect to the computer of the search engine and request to execute the CGI script of the search engine. The CGI script will perform a query processing to check the query and if there are no errors, the CGI script will pass the query to the SES (Search Engine Server) using the CGI-SES (Common Gateway Interface-Search Engine Server) protocol.



SES (Search Engine Server)

A program that runs on the server computer of the search engine and waits for requests from any search engine CGI script. When the SES receives a request from a CGI script, the SES will search the lexicon for the query and send the results to the CGI script, which will send them to the client computer (user).

The On-Line phase is shown in the following figure:





Conclusions

- 1- The crawler uses only one list with two pointers to keep track of the downloading process, which will add some extra speed over crawlers that uses two lists.
- 2- The index is spread in 36 files, which reduce the sort time and memory space required to build the inverted index.
- 3- The lexicon is created using a multilayer structure with 4 layers instead of the binary search tree, which provide a very fast access to the words required in the search process.
- 4- Using only a ranking system based on the word attributes on the page does not give the required results; the structure of the web (links) should be used to give more power to the ranking system.



Suggestions for Future Work

- 1- Research on the user needs, the problems they face when using the search engine and all the ideas they have or want to be in the search engine.
- 2- Research on finding the best stop words to be used in the search engine.
- 3- Research on spelling correction, stemming, abbreviation and phrase search features.
- 4- Develop a NLP query interface for the search engine.
- 5- Develop a search engine that index non-html documents like PDF, Microsoft Word and other types of documents.
- 6- Design a search engine with a different interface technique between the client request and the search program that is faster than the socket interface.



7- Add the title and date of publication to the URL list to this proposed system.

8- Design a search engine capable of searching in more than one language.

9- Design an Internet Crawler.

10- Develop a ranking algorithm using the structure of the web (links).

11- Develop a search engine that search for images (Content Based Image Retrieval).



Thank you
For
your listening

References

- [All99] Allan Heydon and Marc Najork, “Mercator: A Scalable, Extensible Web Crawler”, Compaq System Research Center, 1999, Available at <http://research.compaq.com/src/mercator/papers/www/paper.html>
- [Art05] Articles, “Search Engine Types”, 2005, paper available at <http://www.thejcdp.com>
- [Ask01] Ask Jeeves, “Number of words in English language”, 2001, paper available at <http://www.tm.wc.ask.com>
- [Ask05] Ask Oxford, “frequency of the letters of the alphabet in English”, 2005, paper available at <http://www.askoxford.com/asktheexperts/faq/aboutwords/frequency?view=uk>
- [Bibf99] Internet Certification Institute International, “Basic Internet Business Fundamentals”, 1999.
- [Bri00] Brian H. Murray, "Sizing the Internet", Cyveillance, Inc, 2000. Available at http://www.cyveillance.com/web/downloads/Sizing_the_Internet.pdf
- [Bry96] Bryan Pfaffenberger, “Netscape Navigator 3.0, surfing the web and exploring the Internet”, AP Professional, 1996.
- [Dav00] David Maggiano, “CGI programming with Tcl”, Addison-Wesley, 2000.
- [Eli01] Elizabeth Liddy, “How a Search Engine Works”, School of Information Studies, Syracuse University, 2001. Available at <http://www.infoday.com/searcher/may01/liddy.htm>
- [Fri96] Fritz J. Erickson and John A. Vonk, “Effective Internet”, Irwin McGraw-Hill, 1996.
- [Fri97] Fritz J. Erickson and John A. Vonk, “Netscape Navigator and the World Wide Web”, Irwin McGraw-Hill, 1997.

References

- [Isr05] Isra'a Tahseen Ali Al-Attar, M.Sc. thesis, "Internet Search Engine Design", 2005.
- [JHL98] Junghoo Cho, Hector Garcia-Molina, Lawrence Page, "Efficient Crawling Through URL Ordering", Department of Computer Science, Stanford University. Available at <http://decweb.ethz.ch/www7/1919/com1919.htm>
- [Jia00] Jianlin Cheng, "Design and implementation of ICS Web Search Engine", information and Computer Science Department, University of California, Irvine, 2000. Available at <http://contact.ics.edu/download/cheng-report.pdf>
- [Mar01] Marc Najork, Janet L. Wiener, "Breadth-first search crawling yields high-quality pages", Compaq Systems Research Center, 2001, Available at <http://www10.org/cdrom/papers/208/>
- [Pet98] Petar Perkovic, "Search engines: do you speak their language? A search for ideal interface", Faculty of Philosophy, Department of information science, 1998.
- [Ric96] Rick Stout, "The World Wide Web Complete Reference", Osborne McGraw-Hill, 1996.
- [Sab02] Saba Abdul Khaliq Abdullah Al-Khadady, M.Sc. thesis, "Internet and Arabic search engines", 2002.
- [SL98] Sergey Brin and Lawrence Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine", Computer Science Department, Stanford University, 1998. Available at <http://www-db.stanford.edu/~backrub/google.html>
- [Sun01] Sunny Lam, "The Overview of Web Search Engines", Department of Computer Science, University of Waterloo, Ontario Canada, February, 9,2001. Available at <http://db.uwaterloo.ca/~tozsu/courses/cs748t/surveys/sunny.pdf>
- [Trek01] "Introduction to TREK (Text Retrieval Conference)", 2001. Available at <http://www10.org/cdrom/papers/317/node1.html>

References

- [Vir01] “Virtualis Glossary”, Virtualis Systems, 2001. Available at http://www.virtualis.com/guides_glossary.html
- [Wi103] WilsonWeb Stop Words paper. Available at <http://www.hwwilson.com/default.cfm>
- [Wis01] “WiseNut Search Engine White Paper”, September 2001. Available at <http://www.wisenut.com/pdf/wisenutwhitepaper.pdf>

Table of Contents

ABSTRACT.....	II
LIST OF ABBREVIATIONS	III
TABLE OF CONTENTS	V
CHAPTER ONE: INTRODUCTION.....	1
1.1 Introduction.....	1
1.2 The Internet.....	1
1.3 The Physical Internet	2
1.4 The Soft Internet	2
1.5 The World Wide Web.....	3
1.6 Searching the Web	4
1.6.1 Starting points pages	5
1.6.2 Subject Trees.....	5
1.6.3 Search Engine	5
1.6.4 Web Site Search Engine.....	6
1.7 Literature Survey	7
1.8 Aim of Thesis.....	8
1.9 Thesis Layout.....	8
CHAPTER TWO: WEB AND SEARCH ENGINES.....	10
2.1 Introduction.....	10
2.2 The Client-Server Model	10
2.3 Web Servers.....	11
2.4 Uniform Resource Locator (URL).....	12
2.5 The HTML Language	12

2.6 Browser	13
2.7 Characteristics of the Web	14
2.8 Difficulties of the Web	15
2.9 Information Retrieval.....	17
2.10 Information Retrieval and Data Retrieval	18
2.11 Types of Search Engines	19
2.12 The Problem with today’s Search Engines.....	20
2.13 Architecture of Search Engine.....	21
2.14 Crawling Part	22
2.14.1 Crawling Techniques	24
2.14.2 Crawler Types	25
2.15 Indexing Part.....	25
2.15.1 Indexing Steps.....	29
2.16 Ranking Part	33
2.16.1 Difficulties in Determining Relevancy	34
2.16.2 Document Features	34
2.17 User Interface Part	35
2.17.1 Query Interface	35
2.17.2 Query Processor	37
2.17.3 Answer Interface	38
2.18 Searching Part	39
2.19 Search Engines Examples.....	39
CHAPTER THREE: DEVELOPMENT OF WEB SITE SEARCH ENGINE.....	43
3.1 Introduction.....	43
3.2 Architecture of Web Site Search Engine	43
3.3 Crawler.....	47
3.3.1 Initializing the Crawler Information	48
3.3.2 Initializing the URL List	49
3.3.3 Downloading Pages and Extracting Links	51
3.3.4 Saving the Crawling Information.....	59
3.3.5 Saving the Crawler List	59

3.4 Indexer and Ranker	60
3.4.1 Creating the Index	60
3.4.2 Creating the Inverted Index	69
3.4.3 Creating the Lexicon	72
3.5 Interface	73
3.6 Searcher	74
3.7 The CGI-SES Protocol	77
CHAPTER FOUR: WEB SITE SEARCH ENGINE OPERATION.....	79
4.1 Introduction	79
4.2 Programming Languages	79
4.3 User Interface	80
4.3.1 Crawler User Interface	80
4.3.2 Indexer User Interface	83
4.3.3 Search Engine Server User Interface	85
4.3.4 Search Engine Web page Interface	86
4.4 Features	88
4.5 System Requirement	89
4.5.1 Hardware Requirement	89
4.5.2 Software Requirement	95
4.6 Experiment and Results	95
4.7 Tools	97
CHAPTER FIVE: CONCLUSIONS AND FUTURE WORKS.....	99
5.1 Conclusions	99
5.2 Suggestions for Future Works	99
REFERENCES.....	100
APPENDIX A: THE INTERNET	A-1
APPENDIX B: CGI LIBRARY.....	B-1

Republic of Iraq
Al-Nahrain University
College of Science



Development of a Web Site Search Engine

A THESIS
SUBMITTED TO THE
COLLEGE OF SCIENCE, AI-NAHRAIN UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE IN
COMPUTER SCIENCE

By

Eihab Ahmed Mohammed Shakir

(B.Sc. 2002)

SUPERVISORS

Dr. Moaid A. Fadhil

Dr. Ban N. Al-kallak

March 2005

Moharam 1425

الخلاصة

ان كمية المعلومات الموجودة على شبكة الانترنت تزداد بشكل سريع جدا، اضافة لعدد المستخدمين التي تنقصهم الخبرة في مجال البحث في الشبكة، مما ادى الى تطوير الكثير من التطبيقات الخاصة بالشبكة و التي تسمى محركات البحث المتخصصة في مساعدة المستخدم في ايجاد المعلومات المطلوبة من على الشبكة.

محرك بحث الموقع هو برنامج يستخدم للبحث في موقع معين في الشبكة عن استفسار معين. ان هذا البحث يهدف الى تصميم محرك بحث للموقع يساعد المستخدم في استرجاع الصفحات الاكثر مطابقة مع الاستفسارات المقدمة من قبل المستخدمين. يعتمد الجزء المسؤول عن الرتب على صفات الكلمة (مثل حجم الخط، نوع الخط، لون الخط، مكان الكلمة في الصفحة، نص ارتباطي، عنوان، و عنوان خاص) و هو مدمج مع الجزء الخاص بالفهرسة. تم تقسيم الفهرس الى ٣٦ فايل للتقليل من مصادر النظام (الذاكرة و وحدة الخزن المطلوبة)، كما تم تصميم الفهرس المعاكس عن طريق ترتيب الفهرس باستخدام طريقتي الترتيب السريع المحسنة و الادخال (Quick and Insertion sort methods) وذلك لزيادة سرعة الفهرسة. تم تصميم القاموس باستخدام هيكل متعدد الطبقات يتكون من اربعة طبقات.

نظام الرتب المستخدم في محرك بحث الموقع المقترح يعتمد فقط على صفات الكلمة، ممكن استخدام الهيكل الارتباطي الموجود في شبكة الانترنت لزيادة كفاءة نظام الرتب.

ان محرك بحث الموقع المقترح يتطلب جهاز حاسوب يعمل كخادم، كما تم استخدام الادوات البرمجية التالية: HyperText Markup Language, Visual Basic Script, Common Gateway Interface technique, Microsoft Visual Basic 6.0, and Windows operating system Socket.



الجمهورية العراقية
وزارة التعليم العالي و البحث العلمي
جامعة النهرين

تطوير محرك بحث الموقع

رسالة

مقدمة الى قسم علوم الحاسبات في جامعة النهرين
كجزء من متطلبات نيل درجة الماجستير في علوم
الحاسبات

من قبل

إيهاب أحمد محمد شاكر

(بكالوريوس جامعة النهرين ٢٠٠٢)

أشرف

د. بان نديم الكلاك

د. مؤيد عبد الرزاق

محرم ١٤٢٥

آذار ٢٠٠٥

الأسم: ايهاب احمد محمد شاكر مرجان
البريد الالكتروني: Eihabmurjan@yahoo.com
رقم الهاتف الارضي: ٥٥٢٤٧٢٢
رقم هاتف المحمول: ٠٧٩٠٢١٩٠٥٨٩
العنوان: حي الجهاد/محطة ٨٩١/زقاق ١٠/دار ٢٥
تاريخ المناقشة: ٩/٥/٢٠٠٥

desktop

[.ShellClassInfo]

LocalizedResourceName=@%SystemRoot%\system32\shell32.dll,-21815