

**WebGraph Project
Final Report
Spring 2006**

Shilpa Murarka
Spring Semester 2006
Submitted 5/18/06

Table of Contents

I.	Abstract.....	3
II.	Introduction.....	3
III.	Current Database Schema.....	5
IV.	Complexities and their Solutions.....	5
	1. Repeated Source Pages.....	5
	2. URL Equivalences.....	7
	3. Junk URL Problem.....	7
V.	Design and Implementation.....	8
	[A] WebGraph.....	8
	1. WebGraph.txt.....	8
	2. URLs.txt.....	8
	[B] Canonicalizer.....	10
VI.	Compiling code and running code on scidata1.....	10
VII.	Future Work.....	13
VIII.	References.....	13
IX.	Acknowledgements.....	13

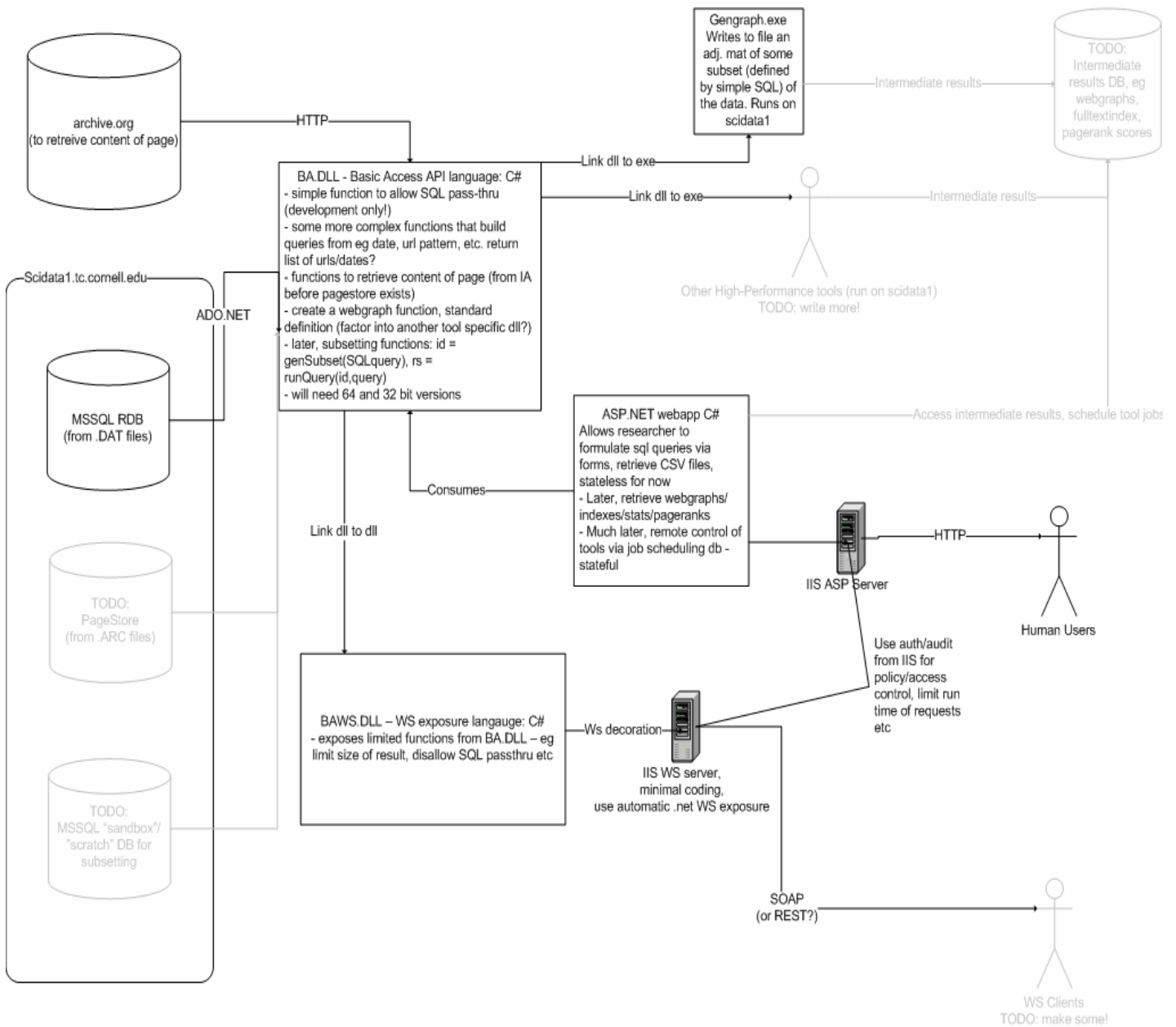
I. Abstract:

The WebLab project is a collaborative effort by researchers to create an intelligent archive of Internet web-pages. Among the user tools is functionality, WebGraph.exe, which writes to a file, some subset of the data (which is defined by a simple SQL). Webgraph writes to a file all source and destination URL pairs. This is a command-line high-performance tool, with very little graphical interface, for use by the researchers, to pull data from the database, given certain filters, very quickly. It runs on scidata1. Since, it is a high-performance tool; it is required to come up with optimal database queries to fast information retrieval. The URLs so obtained further need to be cannibalized for further processing, to remove certain redundancies in the data. The following report discusses a practical design and implementation of WebGraph and the Canonicalizer.

II. Introduction:

The WebLab project was started with the aim of helping researchers study the evolution of the World Wide Web. One can easily appreciate that it is difficult to conduct experiments on content that is spread across the globe and constantly evolving with time. The motivation behind this project is to provide researchers with a wealth of analysis and easily accessible information to allow them to study the patterns and trends of information change on the internet. Currently, implemented archival capability for internet data is restricted to verbatim snapshots of static content stored in online repositories, such as the Internet Archive. The WebLab project seeks to offer more intelligent analysis and classification tools to researchers.

A snapshot of the user tools architecture that has been designed is as follows:



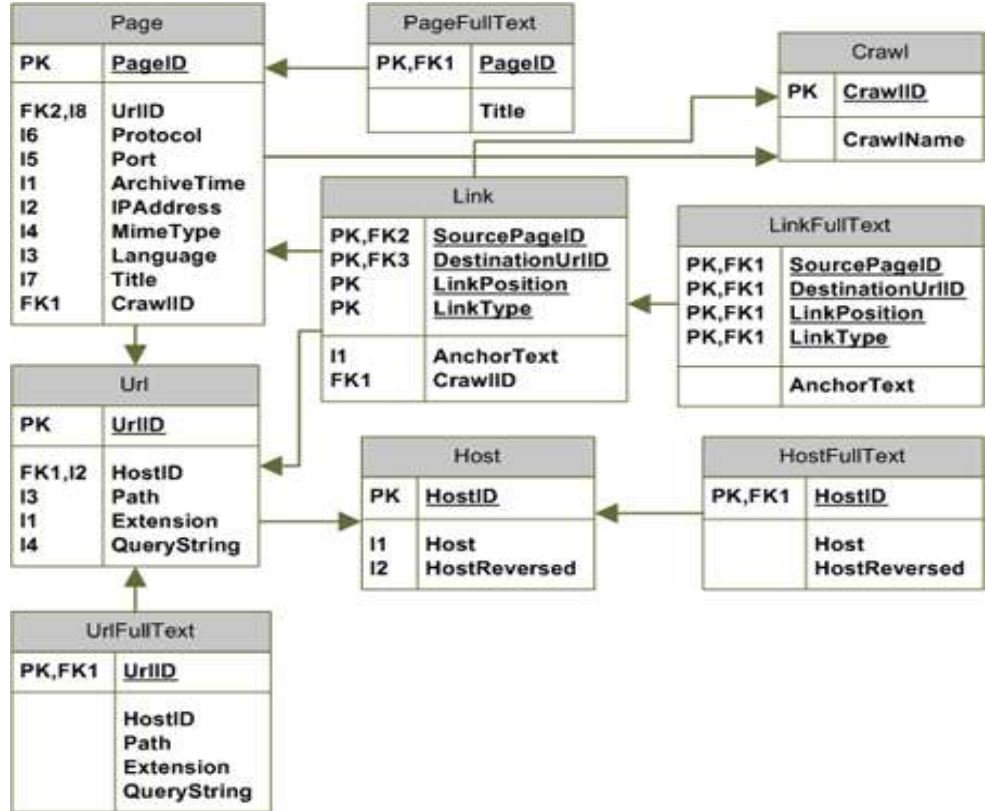
The Gengraph or the WebGraph is an application to fetch the source and the destination URL pairs of all the links in the database. These records can also be passed through filters:

1. Source domain name
2. Destination domain name
3. Crawl ID

These filters may assist in focused searching of the database. The source and the destination URL pairs are fetched as URL id's in the form of byte array from the database. They are stored in text files as well as printed on the screen for reference.

III. Current Database schema:

The current database scheme that has been used while designing the program is:



IV. Complexities and their Solutions:

There are certain complexities in the webgraph generation that have been identified and need to be addressed. These problems and their solutions have been listed as below:

1. Repeated source pages:

In the same crawl, a source page may be crawled more than once. The pages that a source page points to may change between crawls. It may also change between two instants in the same crawl. If we just randomly print out source and destination pairs we would get all source and destination pairs that ever existed and may not represent any state at a particular time. The output matrix should give the links which existed at some particular time instant, and not every link that ever existed during the crawl.

Consider an example.

Note: ts: this is the timestamp when the link between the source and destination was detected.

Consider that pages 1 and 2 were crawled on 17th march and then in the same crawl, happened to be crawled again on 18th March. The links present on these pages changed on 17th March, so we have different destination links for the two dates.

The output matrix would look like

Ts	source	destination
18 th mar	1	2
18 th mar	1	3
18 th mar	2	3
18 th mar	2	4
17 th mar	1	2
17 th mar	1	4
17 th mar	2	3
17 th mar	2	4

ie page 1 has changed on 17th march after the crawl. It was initially pointing to page 2 and 4, and on 18th march page 1 was pointing to pages 2 and 3.

If we just print distinct source and destination pairs, we would get,

Source	Destination
1	2
1	3
1	4
2	3
2	4

However, this neither represents the state of the the web on 17th march, nor that on 18th march. It says that page 1 was pointing to both 3 and 4. However, actually page 1 was pointing to either of them on different instants. So we need to print the webgraph as on 18th march like,

Source	Destination
1	2
1	3
2	3
3	4

or

that on 17th March like,

Source	Destination
1	2
1	4
2	3
2	4

We choose to print the first case as that is the most recent case.

Consider the page table. Its primary key is pageid and foreign key is urlid. It also has an archivetime. When the crawler crawls through a URL, and it is a valid one, then that URL is given a page id and saved along with this archive time in the database. When later, if the same URL is crawled again, it will receive a different page id and archive time. Thus, two page id's can have the same URL id's. They are just differentiated by their archive time. So, for each source URLid, for a particular crawlid, we find out the latest archive time when that URL was crawled and only that one is considered and the other pages with the same URL id and older archive times are ignored.

2. URL equivalences:

Different URL's may be equivalent . e.g. 'http://google.com' , 'www.google.com' , 'http://google.com:80/' , 'http://google.com/index.html' etc. There will be separate URL id's corresponding to these. However, since all these links mean the same, only one of these URL id's should be printed.

To remove URL equivalences, another program, Canonicaliser is written. This program reads the URL's printed out in the flat file by the earlier WebGraph program and canonicalises the URLs and replaces all the URL's by their canonicalised form. The definitions of rules of canonicalisation have been discussed later in the report.

3. Junk URL problem :

Some of the URL's may be very obviously invalid. Eg. 'http://', 'google.co' , etc. These URL id's should be removed from the output file. The invalid URLs are generally due to some discrepancy in the domain part of the URL. It may be missing, incomplete or incorrect.

In order to remove Junk URLs, a similar program like Canonicaliser needs to be built which reads all URLs from the flat file, and according to certain rules defined, classifies URLs as junk. It should then print out all the non-junk URL's to a separate file. This program has not been implemented and future work needs to be done on this.

V. Design and Implementation:

[A] WebGraph:

WebGraph is a command line application that gives the URLids in the webgraph as well as the URLs.

It prints out two text files:

1. webgraph.txt:

This file contains the source and the destination URL ids in the pairs of the Webgraph. It asks the user to input values for the following filters:

1. Source Domain name
2. Destination Domain name
3. Crawl ID

The user can either input these values if he wants to or can just press enter if these filters are not required.

The SQL query used for this file is:

```
select distinct p.urlid, u1.urlid
from page p, url u1, link l, url u2, host h2, host h1
where p.pageid = l.sourcepageid
and l.destinationurlid = u1.urlid
and p.pageid in
    (select p.pageid from page p,
        (select urlid as url, crawlid as crawl, max(archivetime) as
            archtime    from page group by urlid, crawlid) as temp2
    where p.urlid = temp2.url
    and p.archivetime = temp2.archtime
    and (temp2.crawl = @Crawlid or @Crawlid = '' )
    and p.urlid= u2.urlid
    and u2.hostid = h2.hostid
    and (h2.host = @Srcdomainname or @Srcdomainname = '')
    and u1.hostid=h1.hostid
    and (h1.host = @Dstdomainname or @Dstdomainname = ''));
```

This query takes approximately a minute to fetch all records, when running on scidata1.

2. URLs.txt

This file contains all the URLs which were either a source or a destination in the pairs above. Each row consists of one URL. It consists of each of the following parts of a URL, separated by a space:

1. URL id
2. Protocol

3. Domain Name
4. Port Number
5. Path
6. File Extension

The URLs are written to this separate file, so that they can be canonicalised, by the next program ie Canonicaliser.

The SQL query for this is:

```

select distinct p.urlid, p.protocol, h2.host, p.port,
u2.path,u2.extension
from page p, url u1, link l, url u2,host h2, host h1
where p.pageid = l.sourcepageid
and l.destinationurlid = u1.urlid
and p.pageid in
    (select p.pageid
     from page p, (select urlid as url, crawlid as crawl,
max(archivetime)
                    as archtime
                    from page
                    group by urlid, crawlid) as temp2
    where p.urlid = temp2.url
    and p.archivetime = temp2.archtime
    and (temp2.crawl = @CrawlID or @CrawlID = '' ))
    and p.urlid= u2.urlid
    and u2.hostid = h2.hostid
    and (h2.host = @SrcDomainname or @SrcDomainname = '')
    and u1.hostid=h1.hostid
    and (h1.host = @DstDomainname or @DstDomainname = '')

union

select distinct u1.urlid, p.protocol, h1.host, p.port,
u1.path,u1.extension
from page p, url u1, link l, url u2,host h2, host h1
where p.pageid = l.sourcepageid
and l.destinationurlid = u1.urlid
and p.pageid in
    (select p.pageid
     from page p, (select urlid as url, crawlid as crawl,
max(archivetime)
                    as archtime
                    from page
                    group by urlid, crawlid) as temp2
    where p.urlid = temp2.url
    and p.archivetime = temp2.archtime
    and (temp2.crawl = @CrawlID or @CrawlID = '' ))
    and p.urlid= u2.urlid
    and u2.hostid = h2.hostid
    and (h2.host = @SrcDomainname or @SrcDomainname = '')
    and u1.hostid=h1.hostid
    and (h1.host = @DstDomainname or @DstDomainname = '');

```

[B] Canonicaliser:

A URL is composed of six components: the protocol, host, port, path, query, and fragment components. The protocol (Hypertext Transfer Protocol) is used for communicating between a web server and a client. The host component contains a location of a web server.

The location

can be described as either a domain name or IP (Internet Protocol) address. A port number can be specified in the component. The colon symbol (":") should be prefixed prior to the port number. The path component contains directories including a web page and a file name of the page. A directory and a file are separated by the slash symbol ("/"). The query component contains parameter names and values that may be supplied to web applications. The query string starts with the question symbol ("?"). A parameter name and a parameter value are separated by the equal symbol ("="). A pair of parameter name and value is separated each other by the ampersand symbol ("&"). The fragment component is used for indicating a particular part of a document. The fragment string starts with the sharp symbol ("#"). Fig. 1 shows all the components of a URL

eg. `http://example.com:8042/over/there?name=ferret#nose`

The URL normalization is a process that transforms a URL into a canonical form. During the URL normalization, syntactically different URLs that are equivalent should be transformed into a syntactically identical URL, and URLs that are not equivalent should not be transformed into a syntactically identical URL.

The following rules are applicable when URLs are normalised:

1. A URL with a default port number (80 for the HTTP protocol) and a URL without the port number represent the same page. For instance, "`http://example.com:80/`" and "`http://example.com/`" represent the same page. During the normalization, the default port number is truncated from a URL.
2. The host domain name component is case insensitive. For example, "`http://EXAMPLE.com`" and "`http://example.com`" represent the same page. During the normalization, all the letters in the components are changed into lower-case letters.
3. A URL with path string null and a URL with path string "/" represent the same page. For instance, "`http://example.com`" and "`http://example.com/`" represent the same page. If a path string is null during the normalization, then the path string is transformed into "/".
4. The path component is case insensitive. For example, "`http://example.com/FOLDER/File`" and "`http://example.com/folder/file`" represent the same page. During

the normalization, all the letters in the path components are changed into lower-case letters.

5. The three file names, “index.htm”, “index.html”, and “default.htm”, are considered as default pages by the Apache Web Server. The URLs with and without the default page names consist of a set of equivalent URL candidates. This means that a URL with a default page specified in the path component is normalized by eliminating the default page name.

6. The protocol component is case insensitive. For example, “HTTP://example/.com” and “http://example.com” represent the same page. During the normalization, all the letters in the protocol components are changed into lower-case letters.

7. A URL with a leading ‘www’ In the domain name is equivalent to a domain name without it. So the leading ‘www’ is truncated to normalize the URL.

X. Compiling code and Running it on scidata1.

Scidata1 is an Itanium machine. So the code has to be compiled for itanium. The steps for compiling code and running it on scidata1 are as follows:

1. The .sln and .csproj files need to be modified. Some text needs to be added for necessary Itanium support.

The following highlighted text shows changes in the .csproj file. The text in blue was added to the original file.

```
<Project DefaultTargets="Build" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <Configuration Condition="'$(Configuration)' == "">Debug</Configuration>
    <Platform Condition="'$(Platform)' == "">AnyCPU</Platform>
    <ProductVersion>8.0.50727</ProductVersion>
    <SchemaVersion>2.0</SchemaVersion>
    <ProjectGuid>{8BF5B062-4D7F-42E0-9892-26919EF9DC9C}</ProjectGuid>
    <OutputType>Exe</OutputType>
    <AppDesignerFolder>Properties</AppDesignerFolder>
    <RootNamespace>WebLab</RootNamespace>
    <AssemblyName>WebLab</AssemblyName>
  </PropertyGroup>
  <PropertyGroup Condition="'$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>>false</Optimize>
    <OutputPath>bin\Debug</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <PropertyGroup Condition="'$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
```

```

<OutputPath>bin\Release\</OutputPath>
<DefineConstants>TRACE</DefineConstants>
<ErrorReport>prompt</ErrorReport>
<WarningLevel>4</WarningLevel>
</PropertyGroup>
<PropertyGroup Condition=" $(Configuration)|$(Platform) == 'Debug|Itanium' ">
  <DebugSymbols>true</DebugSymbols>
  <DebugType>full</DebugType>
  <Optimize>>false</Optimize>
  <OutputPath>bin\Debug\</OutputPath>
  <DefineConstants>DEBUG;TRACE</DefineConstants>
  <ErrorReport>prompt</ErrorReport>
  <WarningLevel>4</WarningLevel>
</PropertyGroup>
<PropertyGroup Condition=" $(Configuration)|$(Platform) == 'Release|Itanium' ">
  <DebugType>pdbonly</DebugType>
  <Optimize>>true</Optimize>
  <OutputPath>bin\Release\</OutputPath>
  <DefineConstants>TRACE</DefineConstants>
  <ErrorReport>prompt</ErrorReport>
  <WarningLevel>4</WarningLevel>
</PropertyGroup>
<ItemGroup>
  <Reference Include="System" />
  <Reference Include="System.Data" />
  <Reference Include="System.Xml" />
</ItemGroup>
<ItemGroup>
  <Compile Include="Program.cs" />
  <Compile Include="Properties\AssemblyInfo.cs" />
</ItemGroup>
<Import Project="$(MSBuildBinPath)\Microsoft.CSharp.targets" />
<!-- To modify your build process, add your task inside one of the targets below and uncomment it.
      Other similar extension points exist, see Microsoft.Common.targets.
-->
<Target Name="BeforeBuild">
</Target>
<Target Name="AfterBuild">
</Target>
-->
</Project>

```

The following highlighted text shows changes in the .csproj file. The text in blue was added to the original file.

```

Microsoft Visual Studio Solution File, Format Version 9.00
# Visual Studio 2005
Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") = "WebLab", "WebLab\WebLab.csproj", "{8BF5B062-4D7F-42E0-9892-26919EF9DC9C}"
EndProject
Global
  GlobalSection(SolutionConfigurationPlatforms) = preSolution
    Debug|Any CPU = Debug|Any CPU
    Debug|Itanium = Debug|Itanium
    Release|Any CPU = Release|Any CPU
    Release|Itanium = Release|Itanium
  EndGlobalSection
  GlobalSection(ProjectConfigurationPlatforms) = postSolution
    {8BF5B062-4D7F-42E0-9892-26919EF9DC9C}.Debug|Any CPU.ActiveCfg = Debug|Any CPU
    {8BF5B062-4D7F-42E0-9892-26919EF9DC9C}.Debug|Any CPU.Build.0 = Debug|Any CPU
    {8BF5B062-4D7F-42E0-9892-26919EF9DC9C}.Debug|Itanium.ActiveCfg = Debug|Itanium
    {8BF5B062-4D7F-42E0-9892-26919EF9DC9C}.Debug|Itanium.Build.0 = Debug|Itanium
    {8BF5B062-4D7F-42E0-9892-26919EF9DC9C}.Release|Any CPU.ActiveCfg = Release|Any CPU
    {8BF5B062-4D7F-42E0-9892-26919EF9DC9C}.Release|Any CPU.Build.0 = Release|Any CPU
    {8BF5B062-4D7F-42E0-9892-26919EF9DC9C}.Release|Itanium.ActiveCfg = Release|Itanium
    {8BF5B062-4D7F-42E0-9892-26919EF9DC9C}.Release|Itanium.Build.0 = Release|Itanium
  EndGlobalSection
  GlobalSection(SolutionProperties) = preSolution
    HideSolutionNode = FALSE
  EndGlobalSection

```

EndGlobal

2. When Visual Studio is run, Itanium is listed as one of the platform options along with 'Any CPU'. The dropdownlist is switched to cross compile and the output goes in the bin/Itanium/Release directory.
3. The .exe from the bin/Itanium/Release directory should be copied from the local machine to H: (ie to [\\tc.cornell.edu\tc](http://tc.cornell.edu/tc)) in the Users folder.
4. On scidata1, this .exe can be copied from H: to the local T:. It can be executed from command line.

VII. Future Work:

The Canonicalizer gives us a file with many canonicalised forms of all URLs. These may have duplicates which need to be removed. For that, every URL is required to be compared to every other URL, which essentially makes it an n^2 problem. The URLs may be sorted to help remove duplicates, but this might be an n^2 operation too. Hence, removing all duplicate URLs from the file, will be a complex operation and will require some research. Also, research needs to be done to define a Junk URL and remove these URLs if they are listed as destination URLs in the webgraph.

VIII. References:

1. Web Laboratory Project - <https://gforge.cis.cornell.edu/projects/wri/>
2. Sang Ho Lee, Sung Jin Kim, and Seok Hoo Hong, On URL Normalization: <http://dmlab.ssu.ac.kr/publication/LeKi05a.pdf>

IX. Acknowledgements:

This work is a part of the Web Laboratory, which is a joint project of Cornell University and the Internet Archive. Other members of the team are: William Arms, Blazej Kot, Nick Gerner and Manuel Calimlim This work is funded in part by National Science Foundation grants 0403340, 0127308, and 0537606.