Lab 1

In this programming assignment, you will implement a system that uses various search algorithms to find information in a *pseudo* World-Wide Web. The *states* in this problem are web pages, and the actions are to follow a hyperlink.

We provide a skeleton code of the *main* method in Java. However you may use any programming language you feel comfortable with.

First download the intranets.zip file which contains:

- WebSearch.java: code that provides some hints and a skeleton of the *main* method you could use.
- results.txt: example results obtained
- Sample intranets.

In each of these intranets, *page1.html* is intended to be the START node, and the GOAL node is the one that contains `QUERY1 QUERY2 QUERY3 QUERY4`.

For ease in parsing web pages, all our pages will have a very simple structure. Besides simple words (eg, w6, QUERY2), there will be hyperlinks with the following syntax:

```
<A HREF = pageN.html > word word ... word </A>
```

There won't be any punctuation, quote marks, etc. There will always be spaces around words, the equal sign, and the HTML markers. Words on these pages are either *w#* or *QUERY#*, where # is some integer. Thus, you can simply walk through the source text of pages using Java's `StringTokenizer`, as illustrated in the provided hints.

Note that you can use your Web browser to *manually* search these intranets. This will give you a good feel for the task. Also, be aware that web browsers provide a menu option that allows you to view the *source* of the page being displayed. Feel free to make your own simpler intranets for debugging purposes.

Task 1

Write code that performs *breadth-first* and *depth-first* search for this task. Your code should report the number of nodes expanded and the solution path (ie, the path from the start state to the goal state - it is fine to print this path 'backwards,' from the goal to the start).

Task 2

The sample intranets 1, 5, and 7 have been randomly created, with the following propensities (from weakest to strongest):

- the more QUERY words on a page, the more likely the links on that page lead to the goal node
- the more QUERY words in the hypertext associated with a hyperlink, the more likely that hyperlink leads to the goal node
- the more *consecutive* and *in numerical order* QUERY words there are in a hyperlink, the more likely that hyperlink leads to the goal node (eg, seeing `QUERY1 QUERY2 QUERY3` is a very good indicator)

Since an intranet imitates the web, your surfing experience on the web might help you create good heuristics for your heuristic search. One obvious feature that could be incorporated into a heuristic function for this domain is to count how many special words appear on a given page. If special words show up a lot on a page, there is a high probability that the page is close to a goal page. That is, you can believe you are closer to a goal if you choose to go to a page with more special words on it.

The intuition about the special words on a page also applies to the hyperlinks on a page. That is, if a number of hyperlinks are sprinkled throughout a page, you probably will hesitate for a while in order to decide which one to click on first, given that you want to find the goal as quickly as possible. We often examine the words themselves in a hyperlink in order to guess what kind of page this link will lead to. Because the hypertext associated with the link is often a title, summary, or highlight of the page that the link is pointing to, it is reasonable to guess that the hypertext is similar to the page it leads to in terms of the content of special words. We prefer to first expand those links containing QUERY words because they are more similar to the goal than those links containing only plain words. This suggests a second feature that could be used in defining a heuristic function.

A third possible feature for a heuristic function is also based on the notion of similarity. Notice that the information we are searching for is the sequence of consecutive words QUERY1 QUERY2 QUERY3 QUERY4. Naturally, we think that the order and the consecutiveness is important. Thus we prefer to continue searching at links that have partial sequences of special words that are as similar as possible to the goal sequence. For example, Page4 contains four links. Because the second link contains QUERY2 QUERY3, which other links do not, the page associated with the second link is the best successor page. To implement this feature you must define similarity by quantifying a notion of distance from the goal sequence of words. Look at other pages to get ideas about how to formulate your similarity measure.

All of the above knowledge about web pages concentrates on the content of a page in terms of special words. Considering the positions of those special words on a page is also a good idea. That is, words at the beginning of a page include titles and main ideas, and thus are suggestive of the content of the page. Applying this idea to our intranet, assume that the earlier a special word appears on a page, the more promising the page is. This is a fourth possible feature to use in defining your heuristics.

Some or all of the above four features, and possibly others that you invent, can be combined to define heuristic functions for heuristic searches.

a. Use the above information to devise a *heuristic function* for use in best-first search. Describe your motivation for your heuristic. Note that unlike the standard approach where the heuristic is applied to the next state, here we want to use our heuristic to decide **which hyperlink to 'click on' (fetch) next.** This means that your heuristic function scores each **arc (hyperlink) coming out of the current node** and not each child node. Is your heuristic *admissible*? Explain why or why not. (You're not required to write an admissible heuristic.)
b. Write code that performs *best-first search* for this task, where your evaluation function is the heuristic above. Also, extend your code to handle *beam search*, with a beam width of 20 (but otherwise using best-first search). Beam search with a beam width of *n* keeps only the best *n* paths (as determined by your heuristic) found at any time during the search and discards the rest in order to reduce memory requirements. As in the previous part, your code should report the number of nodes expanded and the solution path.
c. How well did your heuristic work on the sample intranets?

Task 3 (www adventure, extra credit 20%)

You will now experiment with the real WWW search. Define a query word, and the initial website (i.e. computer science department, CNN, etc). You can use the code from the previous tasks to perform the search. You may find packages like html-parser useful.

What to Turn In

Run all of the search strategies implemented on intranets 1, 5, and 7, and turn in via Canvas a compressed file (.zip .rar or .tar.gz) containing the following:

- All of your source code (`e.g., .java` files. Make sure the debugging flag is set to `false`. **Do not turn in the results of running the code with debugging = true.)**
- **A `README` file explaining how to compile and run the program.**
- A short lab report that includes answers to questions listed above (questions 2a and 2c require separate answers) and the experimental results obtained (nodes visited and solution path found) with description of query words and the domain if you did WWW adventure.