Lab 8: Ordered Search Results

Due: April 16th at 11:59pm

Overview

The goal of this lab is to extend your web page index implementation using a priority queue to order the search results. Additionally, you will add functionality to search for multiple words, using the total matches across all words to order results. **This is a group lab, groups of two, but you must work with someone in your lab section.**

Deliverables

Your submission should include requisite code to run your program, but the following files will be the focus of grading:

- `Makefile`
- `README`
- `sorted_search.cpp`
- `search_engine.[cpp|h]`
- `priority_queue.h`
- `binary_heap.h`
- `binary_heap.inl`
- `test_heap.cpp`
- `library/`
- `inputFiles/`

All starter code is provided via `update35`.

Submission

You will submit using `handin35`. Be sure to place all relevant files in `cs35/labs/08`.

Sorted Search Results and Multiple Keywords and Phrases

In this lab, you will extend your previous lab by implementing a priority queue using a binary heap to order search results. You must also add in features for searching multiple words where the sum across matches of each keyword is used to order results. For example, consider the trace below. A search for "swarthmore" alone produced a number of matches, but many instances of "swarthmore" is accompanied by "college", this added to the total wherever both "swarthmore" and "college" appear.

```
./sorted_search inputFiles/urls.txt  inputFiles/ignore.txt
        www.cs.swarthmore.edu/~meeden/index.html
        www.cs.swarthmore.edu/~newhall/index.php
        www.cs.swarthmore.edu/~richardw/index.html
        www.cs.swarthmore.edu/~adanner/index.php
        www.cs.swarthmore.edu/~soni/index.php
        www.cs.swarthmore.edu/~aviv/index.html
       ( More sites omitted)

Enter search query: swarthmore
Search results for "swarthmore":
        www.cs.swarthmore.edu/~soni/cs35/f12/Labs/lab08.php: 69
        www.cs.swarthmore.edu/~meeden/index.html: 4
        www.cs.swarthmore.edu/~soni/index.php: 4
        www.cs.swarthmore.edu/~richardw/index.html: 4
        www.cs.swarthmore.edu/~aviv/index.html: 4
        www.cs.swarthmore.edu/~newhall/index.php: 4
        www.cs.swarthmore.edu/~adanner/index.php: 3
       ( More sites omitted)

Enter search query: swarthmore college
Search results for "swarthmore college":
        www.cs.swarthmore.edu/~soni/cs35/f12/Labs/lab08.php: 69
        www.cs.swarthmore.edu/~meeden/index.html: 8
        www.cs.swarthmore.edu/~richardw/index.html: 8
        www.cs.swarthmore.edu/~soni/index.php: 7
        www.cs.swarthmore.edu/~aviv/index.html: 7
        www.cs.swarthmore.edu/~newhall/index.php: 7
        www.cs.swarthmore.edu/~adanner/index.php: 6
       ( More sites omitted)

Enter search query: (CTRL^D or EOF to exit)
```

Starter's Abstract As usual, this is an involved lab requiring you to touch many pieces of code, so below is an abstract of the work required. A more complete description of each of the provided files is found at the end of this document.

1. Fix any bugs with your AVL implementation and copy the requisite code into the library. Be sure to update the include statements where appropriate.
2. Complete your implementation of Binary Heap in `binary_heap.inl`
3. Write a sequence of tests in `test_heap.cpp`, with at least one test for each binary heap function.
4. Work in `search_engine.[h|cpp]` to implement a class representation of a search engine, and instantiate and use your search engine in `sorted_search.cpp`.
5. (**Extra Credit**) Add additional results for matching arbitrary search phrases.
6. (**Extra Credit**) Add PageRank priority to your search engine.

Implementing a Binary Heap

Your first step is to implement the BinaryHeap class in `binary_heap.h` and `binary_heap.inl`. You will use an array based implementation. Consider the header file below:

```
template <typename P, typename V>
class BinaryHeap : public PriorityQueue<P,V> {
  private:

    //array of priority-value pairs
    Pair<P,V>* items;

    //number of elements in the heap
    int size;

    //the current capacity of the heap
    int capacity;
```

The base underlying structure is an array of Pair objects; note that the pair objects are stack references. The BinaryHeap class is responsible for managing this array by maintaining heap order, completeness, and ensuring the capacity. Like in class, you should use 1-based indexing for your heap which will reduce the complication of the math, but you should be mindful of boundary checks.

SearchEngine Class and the Sorted Search

The purpose of the SearchEngine class is to encapsulate all the functionality of web indexing and search. In many ways its interface is similar to the main search program from Lab 07:

- The SearchEngine is constructed with two file names: one file containing the URLs to search and one file containing the search terms to ignore when indexing web pages. When the SearchEngine is constructed, it should immediately index all pages in the URL file.
- After being constructed, the `executeSearch()` function parses a search phrase and executes the search, returning a pointer to a PriorityQueue of the search results. Each item in the PriorityQueue contains the priority of that query result and its URL.

Once your SearchEngine is complete, your main `sortedSearch` will merely process the command-line arguments and user input, using each user-input query as an argument to the `executeSearch()` function and printing the highest-ranked URLs returned. Below is the class description:

```
class SearchEngine {
  private:
     //TODO: Add private fields

  public:
    SearchEngine(char urlFile[], char ignoreFile[]);

    SearchEngine();              // Default constructor, does nothing.
    ~SearchEngine();

    PriorityQueue<int, std::string>* executeSearch(std::string searchPhrase);
  //...
};
```

Handling User Input

Note that in this lab, you can now search multiple-word phrases. You will want to obtain the entire user phrase at once (e.g., using `getline()`) and then query each word individually (*hint: see the instructions below on how to split a string*). The final result is the union of all occurences of each word in the query. So, for example, if we were searching for two keywords, $k1$ and $k2$ and $k1$ occured 21 times on a URL and $k2$ occured 10 times, than the resulting priority is 31, or the union of the occurances or the sum of occurances.

Parsing User Input

Parsing user input in C++ can be less pleasant than in other languages. In the SearchEngine we have provided a private function, `split()`, that behaves much like the split function in Python. Specifically, given a string as input, it returns a pointer to a heap-allocated list of the whitespace-separated tokens from that string. For example,

```
split("This is    a test");
```

would return a pointer to list containing ”This”, ”is”, ”a”, and ”test”. The caller is responsible for freeing the memory referred to by that pointer.

Error Handling

The requirements for the SearchEngine are the same as for last week's lab. That is, you should handle files properly including in the case that an incorrect file name is given or when a URL cannot be parsed by swatHTMLStream.

Grading and Extra Credit

Rubric

This lab will be graded on a 45 point scale:

- (15 Points) Heap Implementation
- (10 Points) Heap Test routines
- (15 Points) Web page index search and traversal
- (10 Points) Coding Style and memory

Note that there are points dedicated to aspects of your lab that are not part of the solution. Do pay attention to coding style and managing memory, design and lay out your code effectively so that it is readable and easily debugged. Run your program through `valgrind` and plug any memory leaks.

Challenge and Extra Credit

There are two significant extra credit challenges in this lab. Be mindful, these are very non-trivial.

- **(Extra Credit 5pt)** : Add in functionality to match arbitrary phases. For example, if a user searched for "swarthmore college", your search engine would report the number of occurases of the complete phrase "swarthmore college" on each URL as well as the union of searching for each keyword individually. (*HINT: If you stored the index of a word within a web page along with the word, you could check for consecutive keywords. That is the first word on the page has index 0 and the second word has index 1, and so on.*)
- **(Extra Credit 10pt)** : Add an additional ranking feature using PageRank. You can read more about PageRank on the wikipedia page: `http://en.wikipedia.org/wiki/PageRank`. The key to PageRank is that you must implement a tool to parse links and track which webpages link to whom. You should constrain your PageRank-ing to internal URLs only; that is, a link to URLs outside of the `www.cs.swarthmore.edu` domain is not considered. (*HINT: You will need to handle relative and absolute web page links. Partial credit will be award for only using absolute web page links.*)

Provided Code

- `Makefile` : The makefile for compilation, you will not need to edit this file unless you want to compile more code.
- `README` : The readme file, you will need to edit this file
- `sorted_search.cpp` : The main part of the program, you will need to edit this file.
- `search_engine.[cpp|h]` : The class and source file for the SearchEngine class, you will need to edit this file
- `priority_queue.h` : The header file describing the abstract class for a Priority Queue. You will not need to edit this file, but you should read it.
- `binary_heap.h` : The header file for the binary heap. You will not need to edit this file, but you should read it.
- `binary_heap.inl` : The inline implementation file for a binary heap. You will need to edit this file/
- `test_heap.cpp` : The test file for the binary heap. You will need to edit this file.
- `library/` : The library for the search engine. You will need to copy your AVL implementation here. Be sure to update the include statements.
- `inputFiles/` : A directory containing various URL lists that you can use to test your program.

Compilation

Compilation will occur using `make`. To compile your program

```
bash> make
```

If you add extra functionality contained in other files, you should edit the `Makefile`, but it is not necessary. To compile the test code:

```
bash> make test
```

Which will compile an executable `test_heap` at first, you can add more tests as you develop.

Execution

Here are the command line arguments for `crack`:

```
>./sorted_search
Usage: ./sorted_search <url-file> <ignore-file>
Required options:
  <url-file>:     a file of urls to index and search
  <ignore-file>:  a file of words to ignore
```