Algorithms Lab Project
Search Engine

Abdelrahman Abdelmonem
900192706

1. Pseudo Code

a. Indexing algorithm:

As far as I understand, Indexing algorithms is the algorithms used to store the data/content of the websites that we found. In our case, all the data and information about the websites are provided in csv files. Basically the indexing algorithms is the algorithm used to store and get the data from the files to be used in our program.

The indexing algorithms are mainly in a class called inputManager. While the handling of each file is a bit different, the main idea is the same. Open the file stream, read line by line using getline, store the data into suitable data structures.

```
fstream file("")
if(file.is_open())
While ( getline(file, myLineString ) )
      for ( from i=0 to myLineString.size() )
            If (myLineString.at(i) == ",")
                  Flag = 1;
                  //*do something depending on the file we are reading
            else
                  websiteName += myLineString.at(i)

      //**store the data into suitable data structures.
```

* this differs whether we are reading impressions file or the Graph file and so on.
** storing the data also differs depending on which file we are reading, in case of the graph file we store the data into a graph and also a map and update the number of Sites.. This will be illustrated in the data structures part.

b. Ranking algorithm:

The ranking algorithm compromises of different parts:

- Getting CTR.
- Getting pageRank and normPageRank.
- Getting final pageScore.

- Getting CTR:

```
for( int i=0 to numberSites )
            CTR[i] = clicks[i] / impressions[i]
```

- This is calculated every time we want to perform the search, this is to update the CTRs because they may change while using the program.
- NumberSites is a constant that indicates the number of sites we have available this is calculated from the indexing algorithm part.
- The impressions and clicks arrays are updated when the user performs a search or clicks on a website.

- Getting pageRank and normPageRank.

- The algorithm of pageRank was slightly modified from the video given in two main ways:

1. A damping factor was added which essentially represents the idea that the user may randomly visit any website at any time.

2. Sink nodes will usually cause problems that will lead sum of pageRank not equal to 1. The solution for this, according to resources, was to make the sink nodes point to every other node in the graph which can simply be done by adding another term in our equation.

- Final equation of the pageRank became:

$$PR(x) = (1 - \lambda) / N + \lambda \sum \frac{PR(y)}{out(y)} + \lambda \sum \frac{PR(z)}{N}$$

- Where:
  - λ is damping factor which is usually 0.85
  - N is the number of websites in the graph
  - y is the websites that point to x in the graph
  - z is the sink node websites.

- To calculate the pageRank an iterative algorithm was used, the algorithm terminates when either 100 iterations have been done or the difference between the prevRank and currentRank is less than 0.01 then we terminate the iterations:

```
//set base values and get the sinkNodes using a for loop.
for( from i =0 to size)
    prevRnk[i] = 1.0 / size;
    if (adjList[i].size() == 0)
               sinkNodes.push_back(i);

//iterations
for ( from int i = 0 to 100 ) {

       //get the sum of sinkNodes ranks here, this is used in the third term of the
equation.
       for (int j = 0; j < sinkNodes.size(); j++) {
              sinkNodesRnk += prevRnk[sinkNodes[j]];
       }

       //we will loop through the nodes and update the rank of each one.
       for(int j=0 to N) {

              //loop through the nodes that point to node j, and add their ranks divided by
their out degree.
              for( from k=0 to reverseAdjList[j].size() )
                  tempNode = reverseAdjList[j][k];
                      tempSum += (prevRnk[tempNode]) /  (adjList[tempNode].size());

              //Calculate the final sum using the above equation.
               tempSum = (1.0 - dampingFactor) + (dampingFactor * tempSum) + (dampingFactor
* (sinkNodesRnk / size));

               //update difference array and the currRank array and reset
               difference[j] = abs(prevRnk[j] - tempSum);
               currRnk[j] = tempSum;
               tempSum = 0;
       }

       //copy currRank to prevRank using a for loop
       //check if the difference is less than 0.01 and break if that is the case.

}

Return currRank;
```

- Getting final pageScore.

```
for( int i=0 to numberSites ) {
	double tempOne = ((0.1 * impressions[i]) / (1 + (0.1 * impressions[i])));
	pageScore[i] = 0.4 * normPageRank[i] + ((1 - tempOne) * normPageRank[i] + tempOne *
CTR[i]) * 0.6;
}
	//This is a temp array that will be sorted according to corresponding pageScore
values.
int* ordPageRank = new int[numSites];
for (int i = 0; i < numSites; i++) {
	ordPageRank[i] = i;
}

heap_sort(ordPageRank, numSites, pageScore);
```

- The Page Score is simply calculated using a for loop and then implementing the formula given in the slides.
- After calculating the pageScores for all the websites, then they are ordered ascendingly using heapSort, this is done by sorting a temp array that contains the ideas of the websites from 1 to N. This temp array is sorted according to the values of the pageScore array and not the values of the temp array itself.
- This was done by modifying heapSort and instead of comparing between the values of the temp array itself, we compare between the values of the pageScore arrays, and we make sure that if we swap any elements that they are swapped in both arrays. The resulting pageScore array will be sorted ascendingly and the temp array will contain the ids of the websites from smallest pageScore to highest page score (sorted according to pageScores).

2. Complexity Analysis

a. Indexing algorithms:

I will again assume that the indexing algorithms are the algorithms used to store the data from the given files.

- Space Complexity:
  - When reading the Web graph file, each line represents an edge which is stored in a vector of edges. From this vector of edges an adjacency list is created which is a vector of vectors. In the worst case, if the number of websites is n and all of them point to each other then we will have space complexity **O(n^2)** due to the adjacency list.

  - When reading both the impressions file and the clicks file (if it exists) we will need an array or a vector, so worst case **O(n)**,

  - When reading the keywords file, we will store it in a vector of vectors. The columns will be the keywords for each website. Worst case, if the max number of keywords is m and the number of websites is n then we will have **O(nm)**.

- Time Complexity:
  - When Reading any of the files, we loop the file line by line and loop through the line character by character.
  - Worst case, *if the max number of lines is n and the maximum number of characters is m then we will have complexity **O(nm).***

b. Ranking algorithms:

- Space Complexity:
    - Getting CTR:
        - Two arrays are used so complexity will **θ(n),** where n is the number of sites.
    - Getting pageRank and normPageRank:
        - Two arrays and one vector are used, they are all one dimensional so complexity is **O(n).** Where n is the number of websites.
    - Getting final pageScore:
        - Two arrays are used, all of them are one dimensional. So complexity is **θ(n).** Where n is the number of websites.

- ***Total final space complexity will be O(n).***

- Time Complexity:

    - Getting CTR:
        - Complexity **θ(n),** where n is the number of sites, this is due to the one for loop.
    - Getting pageRank and normPageRank:
        - Complexity **O(n^2),** where n is the number of sites, this is because we loop through all the nodes and then loop though the nodes that point to that node, in worst case the graph will be complete.
    - Getting final pageScore:
        - Complexity **θ(n log(n)),** where n is the number of sites. this is due to heapSort.

- ***Total final time complexity will be O(n^2).***

3. Main data structures

- Arrays:
    - Arrays more specifically dynamic arrays are used throughout the program to store a lot of information, like the number of clicks and impressions and pageRank and so on.
- Vectors STL:
    - Vectors are also used as they provide more flexibility and functionality than arrays, more importantly they are part of STL and are compatible with the algorithms library, they are used with the find method a lot which has complexity nlogn.

    - They are also used for adjacency lists and to store the keywords of the websites and to store the Websites names given their IDs and store the search results and so on.
- Unordered_map STL (HashTables):
    - One map is used and it plays an important role in the project. FromWebsitestoID map, the main idea of this map is to store the ids of each website. When reading the impressions and the graph file, each website is given a unique ID. This is important because throughout the program we will need to access the id of a website from its name and vice versa. This map allows us to access the id of a website given its name.
    - Internally, the unordered_maps are implemented using Hash tables.
- Heaps (Trees):
    - Heaps were created and implemented internally using arrays, these are used only once, they are used to sort the websites according to their pageRank values using heapSort.

- Graph Class (vectors):
    - This is a custom class which is used to store the information given in the Graph file. A vector of vectors is used to store the data in adjacency list, another vector of vectors is used to create a reverse adjacency list which is used in the pageRank algorithm.
    - The class contains a default constructor and another constructor which uses a vector of edges, a print graph function and a getRank function and a reSize function.

- Edge (struct):
    - Mainly used in the Graph class.

4. Design tradeoffs/limitations

- I assumed that all the websites will be given either in the ***Web graph file*** or the ***Number of impressions file***. Meaning, that it may be possible for disconnected websites to exist if they only appear in the number of impressions file. Moreover, it means that the keywords file will not contain any websites that didn't appear in either the web graph file or the number of impressions file. This was done because I thought it may be possible that not all websites will appear in the web graph file.

- In the UI version of the project there is a character limit to the length of the search query, this was done purely out of convenience and looks. Since I am not actually using a UI but instead a graphics library, it would have been very difficult to implement an actual textbox that expands with the length of the search query.

- The search Query is not full proof, if you use only one quote then the search results will probably be empty. Moreover, a combination of ANDs and ORs and Quotes will probably create unexpected results. However, the main assumption is that each word is determined by the previous word, meaning that if the word “ice” is preceded by an OR then it will be considered an OR word and won’t be required to appear in the keywords However if it is preceded by an AND then it will be considered an AND word and must and will be required to appear in the keywords of the site. This was done because it was the easiest to implement and made sense to me.

5. References

- https://moz.com/beginners-guide-to-seo/how-search-engines-operate
- https://www.youtube.com/watch?v=_Wc9OkMKS3g
- https://domino.mpi-inf.mpg.de/intranet/ag5/ag5publ.nsf/0/31deb7636690f704c125729c003181ef/$file/www2007.pdf
- https://www.wordstream.com/click-through-rate
- https://en.wikipedia.org/wiki/Click-through_rate