Martina Muawad

900193032

Project Report

**1)PseudoCodes**

1.Indexing:🡪 forward indexing type

Graph createGraph()

{ fstream webGraph

webGraph opens( file that has websites connections)  
reads every website name until a comma and pushes it into vector<string> websites if not repeated.  
close Webgraph  
for( int i=0 to size of websites vector)  
{ indexes in an map of type <string, int> every website to a corresponding int value  
increases a counter for every website inserted to map}  
creates a graph of size counter  
for(int i=0 to size of websites vector, i+=2)  
{ add edges to created graph using the the vector where each i is the source and the next value is its destination.  
jumps 2 steps and does the same // each value is the website that was in the next column in the csv file which means it was connected to another site}  
returns final graph  
} //to access any website value in the future, it is enough to know its index and get the key which is the string hence O(1)  
void createInverseMap(){  
for (pointer it= beginning of key map end of map)  
{fill the first element (key) of the inverse map with the index of the first unordered map  
fills the second element(value) of the inverse map with the string of the first map}

} //the purpose of this function is to facilitate the retrieving of the website name.  
  
2.Ranking Algorithm:  
void compute Ranks() //function is called whenever the program is initialized

{calls the returned final graph from the indexing   
creates two vectors of type double // prev and current vectors  
for(int i=0 to size of graph vertices)  
{ initialize prev[i] = 1.0/numberOfVertices //1/N  
while(bool iterated) // changes when the prev=current and no need to iterate to get different values  
{ for(int j=0 to number of vertices of graph)  
{ for(int k=0 to number of vertices of graph)  
 { for(int m=0 to number of websites connected to that vertex  
 if(j==webgraph.list[k][m]) // a node points to our website temp=prev[k]/webgraph.list[k].size()} //number of nodes that node is pointing to

}}

Fills the current[j] with the temp;}

For(int k=0 to size of current)

{ if( all values in prev == values of current)  
iterate =false //stops iterating  
else fill prev with current values and iterate again}}

For(int i=0 to number of graph vertices)  
{ push all values of current to a global vector pagerank with the same indexing of the graph}}

**2)Time and Space complexity analysis**

1. Indexing:  
- If we assume the number or vertices(websites) and edges we enter is V,\*E then the indexing function loops V\*E times to fill the first array with the file’s input.  
Then to add the vertices into the unordered map keying is takes V/E as we only fill it with the number of vertices minus the edges.   
And because the time complexity of the unordered map is O(1)  
Therefore the time complexity of the algorithm is O(V\*E).

Meanwhile, the space complexity of the unordered map is O(its size) = O(vertices) and the space complexity of the for loops is O(1) since it reuses the same space in every loop.  
Therefore overall space complexity is O(V).

2. Page Ranaking:  
The time complexity of the page rankin algorithm if we have V vertices, we loop over all the vertices hence V times, then we loop at every edge of the vertices to see how many are connected so we loop E times. We iterate until the numbers stop changing which is approximately equal to the V of the graph.   
Therefore, the time complexity of the function is O(V\*E)

The space complexity for the algorithm is O(V) as it computes and saves the rank of each website which is total number of V.

**3) Main data structures:**I mainly used 3 main data structures throughout:  
A) Vectors.  
B) Unordered map (hashing table).  
C) Graph.

**4) Design tradeoffs**1. I used unordered map even though it has space complexity of its size because its time complexity is O(1) so finding and retrieving the key and data from it like the keywords would be a lot faster.  
2. I used adjacency list instead of matrix to iterate over the edges connected to every vertex instead of iterating over the whole matrix to calculate the sum of vertices it is connected with.