Simplified Page Ranking

1. Include

#include <iostream> ---->*use to do basic input and output operations*

#include<cstring> ---->*use to do basic string operation if needed*

#include <bits/stdc++.h> ---->*use to declare map and pair operations*

2. Data Structure Used

typedef map<string, int> web_ID_Map1 ---> *use to declare map with a unique name which use to store web link and corresponding assigned id*

3. Variable taken

int num1 , num2; ---> *Initial input i.e. number of input and number of power iteration*

web_ID_Map1 dataMap1 ---> *Map to store the Web link and map that link with a unique id*

int id = 1; ---> *Use to initialise web link with this id and later we have incremented this id whenever we got a new web link*

int a[2*num1][2*num1]; ---> *Adjacency Matrix with the size as num1 as this is the worse that a matrix can be created*

float r[num1]; ---> *Value to store power iteration*

float r2[num1]; ---> *temp storage of power iteration*

float sum = 0; ---> *Use this in various places for different purpose like calculating individual out-degree*

float M[2*num1][2*num1]; ---> *Store the required matrix which need for doing power iteration*

4. Operation Done

Making an Adjacency matrix and creating a map which will map each web link with a unique id

```
for (int i = 0; i < num1; i++) {
    int ii, jj;
    string str1 ,str2 ;
    cin>>str1>>str2;
    web_ID_Map1::iterator ele1 = dataMap1.find(str1);
    if (ele1 == dataMap1.end()) {
        dataMap1.insert(make_pair(str1, id));
        ii = id;       id++;
    } else
        ii = dataMap1[str1];

    web_ID_Map1::iterator ele2 = dataMap1.find(str2);
    if (ele2 == dataMap1.end()) {
        dataMap1.insert(make_pair(str2, id));
        jj = id;       id++;
    } else
        jj = dataMap1[str2];
    a[ii - 1][jj - 1] = 1;
}
```

In this Portion, we have taking web link from each input and storing it to str1 and str2.

The we are checking if the str1 and str2 is present in our dataMap. If the we link are not present, we are inserting that web link in our dataMap with a unique id and later incrementing that id for further use.

We are also creating an adjacency matrix that shows the linking of str1 with str2. From str1, we check first if the str1 in present in dataMap. And if the str1 is present, we take the ii= dataMap1[str1] and if the str1 is not present, then we take ii = id. And later we just increment the id;

Same thing we do with str2 and get the jj value.

We use this ii and jj value to assign 1 value to our adjacency matrix as it will show a link between

[ii-1] to [jj-1];

Total time complexity for the above operation is **O(n)**.

In above logic, we just took the id instead of link name. Example:

*Google gmail

*Facebook gmail

$\}$ Input

GENERATED MAP:

Google	1
Gmail	2
Facebook	3

$\}$ Link with there unique id assigned

So instead of

Google ->gmail

We did

(1-1) -> (2-1)

"-1" is cause the indexing in matrix starts from 0 instead of 1

Calculating the out degree from each node

```
 for (int i = 0; i < dataMap1.size(); i++) {
   sum = 0;
   for (int j = 0; j < dataMap1.size(); j++) {
      sum += a[i][j];
   }
   r[i] = sum;
}
```

We are iterating the adjacency matrix and one row at a time and storing the number of out degree in sum and later assigning that sum to r[i].

Just for time being, we are storing the out degree in r[i] just to minimise the use of extra variable. Later we will use r[i] to store the values of power matrix.

Total time complexity for the above operation is **O(nxn)**.

Matrix with out-degree value

```
for (int i = 0; i < dataMap1.size(); i++) {
   for (int j = 0; j < dataMap1.size(); j++)
      if (a[j][i] == 1)
         if (r[j])
            M[i][j] = 1 / r[j];
         else
            M[i][j] = 0;
}
```

In this, we are creating a matrix required for Power iterations. As instructed, we assign the

1/out-degree of j, in M[i][j] , if there is any link between the i and j node.

Otherwise, assigning 0.

Total time complexity for the above operation is **O(nxn)**.

Power column matrix

```
for (int i = 0; i < dataMap1.size(); i++) {
   r[i] = dataMap1.size();
   r[i] = 1 / r[i];
}
```

For the initial value of power matrix, we are initializing the r[i] with the

"**1/total number of node**"

And total number of node is nothing but just size of our dataMap.

Total time complexity for the above operation is **O(n).**

Power Iterations

```
while (num2--) {
   for (int i = 0; i < dataMap1.size(); i++) {
      sum = 0;
      for (int j = 0; j < dataMap1.size(); j++)
         sum += (M[i][j] * r[j]);
      r2[i] = sum;
   }
   for (int j = 0; j < dataMap1.size(); j++)
      r[j] = r2[j];
}
```

In this we are just multiplying the **M** matrix, storing the out degree as instructed, with the column matrix **r** i.e. power matrix.

And this operation is takes place total of num2 -1 times to get our required power matrix.

Total time complexity for the above operation is **O(nxn + n) = O(nxn).**

Printing the Output

```
web_ID_Map1::iterator itr = dataMap1.begin();
for (; itr != dataMap1.end(); itr++) {
   cout << itr->first << " " << fixed << setprecision(2)<<r[itr->second - 1] << endl;
}
```

Printing just an output of the code

5. Actual Code

```
#include <iostream>

#include<cstring>

#include <bits/stdc++.h>


using namespace std;


typedef map<string, int> web_ID_Map1;


int main()

{

   int num1 , num2;                         //Initial input i.e. number of input and number of
power iteration

   web_ID_Map1 dataMap1;      //Map to store the Web link and map that link with a
unique id

   int id = 1;                                        //Use to initialize web link with this id and later
we have incremented this id when ever we got a new web link

   cin>>num1>>num2;

   num2--;                                                    //For expected output, we reduce it to
one less power iteration

   int a[2*num1][2*num1];              //Adjacency Matrix

   float r[num1];                                 //Value to store power iteration

   float r2[num1];                               //temp storage of power iteration

   float sum = 0;                                //Use this in various places for different
purpose like calculating individual out degree

   float M[2*num1+1][2*num1];                //Store the required matrix which need for
doing power iteration

   memset(&M, 0, sizeof (M));

   memset(&a, 0, sizeof (a));

   memset(&r2, 0, sizeof (r2));

  memset(&r, 0, sizeof (r));
```

```
        //Making an Adjacency matrix and creating a map which will map each web link with
a unique id
  for (int i = 0; i < num1; i++) {
    int ii, jj;
    string str1 ;
    string str2 ;
    cin>>str1>>str2;
    web_ID_Map1::iterator ele1 = dataMap1.find(str1);
    if (ele1 == dataMap1.end()) {
      dataMap1.insert(make_pair(str1, id));
      ii = id;
      id++;
    } else {
      ii = dataMap1[str1];
    }

    web_ID_Map1::iterator ele2 = dataMap1.find(str2);
    if (ele2 == dataMap1.end()) {
      dataMap1.insert(make_pair(str2, id));
      jj = id;
      id++;
    } else {
      jj = dataMap1[str2];
    }
    a[ii - 1][jj - 1] = 1;
  }

  //Calculating the out degree from each node
  for (int i = 0; i < dataMap1.size(); i++) {
    sum = 0;
    for (int j = 0; j < dataMap1.size(); j++) {
        sum += a[i][j];
    }
    r[i] = sum;
}

//Matrix with out degree value
for (int i = 0; i < dataMap1.size(); i++) {
    for (int j = 0; j < dataMap1.size(); j++)
        if (a[j][i] == 1)
            if (r[j])
                M[i][j] = 1 / r[j];
            else
                M[i][j] = 0;
}

//Power column matrix
for (int i = 0; i < dataMap1.size(); i++) {
    r[i] = dataMap1.size();
    r[i] = 1 / r[i];
}

//Power Iterations
while (num2--) {
    for (int i = 0; i < dataMap1.size(); i++) {
        sum = 0;
        for (int j = 0; j < dataMap1.size(); j++)
            sum += (M[i][j] * r[j]);
            r2[i] = sum;
        }
        for (int j = 0; j < dataMap1.size(); j++)
            r[j] = r2[j];
    }
    web_ID_Map1::iterator itr = dataMap1.begin();
    for (; itr != dataMap1.end(); itr++) {
        cout << itr->first << " " << fixed << setprecision(2)<<r[itr->second - 1] << endl;
    }
    return 0;
}
```

Total Time Complexity for the above Code is O(nxn).

What’s new we learnt

We learnt about a new data Structure i.e. maps and few new features of map like it store data in a pair format, the data is sorted by default, and only one data can be map with the hash key hence giving the time complexity O(1) for accessing the data corresponding to any hash key and we also need an iterator to print all the values in map.

We also learnt the concept of adjacency matrix and how we can create it.

We got to know about few terms like out degree, power matrix, power iteration and there uses.

We have even learnt how to set values in array using memset and memcpy. And also got to know how to print a floating point value up to a particular decimal point.

Also, it’s helps in our logic building ability.

At last, we learnt that whenever we need to iterate an adjacency matrix, the time complexity of the logic must be at least o(nxn);

References:

www.geeksforgeeks.com -> for referring the syntax

https://stepik.org/lesson/241836/step/1 -> for testing the code