Assignment 4 : Report

1.

Trie

(a)

Approach

I have implemented a compressed trie with compact storage in the first problem, hence reducing the space complexity of a node from O(n) to O(1), where n is the length of a string. To achieve this, first a class List was created, which stores two array, one containing all the names and other containing the corresponding numbers. Each node of the trie is defined by three integers, one the word_index, which corresponds to the index of the word in the name array, next a begin variable, corresponding to the starting index of the substring in the node and an end variable, corresponding to the end of the substring. Hence, reducing the space complexity to O(1).

Apart from the index range variables, the node class contains an array of 26 pointer children, corresponding to each alphabet i.e. children[0] corresponds to a child node with substring starting with 'a', a parent pointer and a boolean variable "isEnd" denoting whether the node terminates a name.

Lastly, there is a Trie class, which contains a root to the compressed trie, word count (c), the list of names and numbers, match count, which is equal to the number of matches computed by a query, and an array of atmost 5 strings corresponding to the top 5 matches of a query. Since the word_index of each terminating node also corresponds to the index of the array containing the number of that person, displaying the number is also easy.

Observation

The insert function has a complexity of O(m) where m is the length of the word to be inserted in the trie. Since each node contains an array of 26 node pointers, and each pointer corresponds to an alphabet, child selection takes O(1) during the search operation. Splitting of node takes O(1) time, since only indices have to be changed.

The find function has a complexity of O(m*c), where m is the length of the longest word and c is the number of words in the trie, since in the worst case, the whole trie would be traversed.

How to Run

Keep the text file "name_number.txt", which contains the list of names and numbers, in the same folder as main.cpp. To compile, simply make, this would create an output file. To run, simply type:

```
$ ./output
```

1.

Graph

(a)

Approach

As mentioned in the question, I have implemented a directed graph using adjacency list. There is a separate vertex class, which contains the course name, the course duration, the in degree of the vertex, the index of the vertex in the adjacency list and a decoration object. The decoration class consists of additional variables which would help in better functioning of my algorithms. The edge class consists of two vertex variables corresponding to the two ends of an edge, and a weight variable denoting the weight of the edge, although in this problem, all weights are 1. Since the graph is directed, the "start" vertex of the edge points to the "end" vertex. All vertices are dynamically allocated.

The graph class contains two vectors, one vector of vertices and the other vector of list of edges (the adjacency list), where each index of the vector corresponds to a vertex. The graph class also contains a mapping from the course name to the vertex. Since in our version of C++ does not support unordered maps, in which all operations take O(1) time, C++ maps were used, which takes O(logn) time. However, this increase in complexity would only affect the add vertex and add edge operations, and not the detect cycle and longest path functions, since after the mapping is created, the index of the vertex in the adjacency list were stored in the vertex object itself.

To detect a cycle in the graph, I used DFS search. If an already visited node is still inside my recursion stack, there is a cycle in my graph.

To find the longest path, I used topological sort using BFS. I chose BFS over the usual DFS approach to topological sort to compute the longest path since I was adding the vertices in the longest path at the same time I was finding the length of the longest path. Initially the first course (having 0 in-degree) is pushed into a queue. Then, the in-degrees of all its adjacent nodes is reduced by one (BFS), if some node now has the in-degree 0, it is pushed into the queue. At the same time, the longest path till that node is computed and the vertices in that longest path are stored in a vector. In the end, the longest path ending at the last node is printed.

Observation

Insertion of vertices and edges take O(logn) time, because of the use of maps.

The detect cycle function takes O(V+E) time, where V and E are number of vertices and edges, since it uses the DFS algorithm.

The longest path function takes O(V+E) time, since it uses modified BFS (Kahn's) Algorithm.

How to Run

Keep the text files “courses.txt” and “dependencies.txt” , which contains the list of names and numbers, in the same folder as main.cpp. To compile, simply make, this would create an output file. To run, simply type:

```
$ ./output
```