CSC 3100 Assignment 4 Report

Ziyu XIE
121090642

December 7, 2022

1 Special Shortest Path

This problem asks to find a special shortest path when the graph consists a special variable K and somtimes the consuming varies depending on K.

The source code is in A4_P1_121090642.cpp

For this problem, we can change the original graph G(E,V) to a new graph G(E',V') by adding some nodes and edges to G. For example, when K = 2, and we have a path **1**–>(2)–> **2**–>(4)–> **3**–>(8)–>**4**, then we can add one node5 between node1 and node2, add node6 between node2 and node3, and add node7 between node3 and node4. Then we can add edge with length $\frac{(K-1)*2}{2} = 1$ between node5 and node6, add edge with length $\frac{(K-1)*4}{2} = 2$ between node6 and node7. Then we use Dijkstra algorithm to solve the shortest path problem, and we get 8 as the shortest path for node4, the final result is 0 2 4 8.

If we can get such graph, then the problem solved. To get this graph, we need to consider each line of the input. Each line of the input tells us an edge's length and two nodes. For the two nodes, we have an additional new node in G', hence the total number in G' must be (m+n). However, when I first consider like this, I miss one point. That is, this is a graph with no direction, hence in fact there are two direcitve edges between two nodes. As a result, the total number of the node in G' must be (m+2*n). Then, we add two edge between them, which count for 4 edges.

Each time when we insert a line of data, we form such basic situation. Moreover, we can add our special path during this process. For each node, when we insert it, we find this node adds two edges to itself, and such edges are in pair. Then we can search all the previous edges the node have and consider whether we need to add the path.

In order to implement my algorithm, we must save both the edges to and from one node. For one node, there are two kinds of cases. The first is some edges' length, say l_1, is K time of the new added edge, then we need to let the new

edges' "node" to point to such edge. And the second is l_2, is 1/K time of the new added edge, then we need to let such edges' "node" to point to our new edge.

Such algorithm is done by the function **void insertGeneralEdge(int u, int v, lldouble w)**, and the function **void addsingleDirectEdge(int a, int b, lldouble w)** and **void addSpecialEdge(lldouble w, int a, int size, int atob, int btoa)** help it.

We do this for two new nodes, then we continuously insert all the nodes, then the graph has already been our expected G'. We do Dijkstra algorithm, say the function **void dijkstra()** to solve it directly.

Note that the number of nodes is very large, hence it is not convinient to use adj-matrix, so we use **vector<vector<> >** to save all the datas, similar to the linked list.

We also use heap to implement the algorithm, so the time complexity for Dijkstra is $O(V + VlgV + ElgV)$, and for the insertion, the time complexity is $O(V + E + 4 * E + E^2)$, but since the special path is not quite a lot, we have this E^2 would not be too large.

2 Median Search Tree

This problem asks to do some operations on the median 2k values among 2n numbers.

The source code is in A4_P2_121090642.cpp

This question is easy to solve. We can maintain two heaps to do it. The two heaps are big-top heap and small-top heap. If we can let all the big-top heap's element smaller than the top of the small-top heap, and the numbers are uniformly divided into these two heaps, then we can get the 2k median numbers and do the insertion and deletion efficiently.

Each time we insert, we maintain the numbers in two heaps are equal or the number of big-top heap is bigger than the small-top heap at most 1. The time compelixity for such insertion is $O(lgN)$, because the insertion for the heap is $O(lgN)$. The function in my code called **void insert(int w)** does this operation.

For output of the median 2k numbers, we need to pop all the k numbers in each heap and then put them back. the time compelixity is $O(4 * k * lgN)$. The function in my code called **void outputMedian()** does this operation.

For deletion, it is similar to the output operation. The time compelixity is $O(4 * k * lgN)$. The function in my code called **void deleteP(int p)** does this operation.

3 Football Match

This problem asks to find the maximum sum of the attractions of the (n-1) matches.

The source code is in A4_P3_121090642.cpp

This problem can be easily to be converted into a problem called the maximum spanning tree. Since this is a dense graph, we use the Prim's algorithm, the time complexity of this algorithm is $O(ElgV + V^2)$ in my code, called **long long int prim(long long int start)**.

For this question, we have to be concerned about the data range because the data may exceed the maximum integer limit, so in cpp we need to use long long int.

If we can get such a graph, then the problem solved. To form this graph, each time we insert a popularity about one node, we calculate the new edges formed by this node. The time complexity is $O(V^2)$, because each node form n-1 edges to other nodes. The function **void addNode(long long int cur_n, long long int value)** does this operation and the function **void insertEdge(long long int u, long long int v, long long int w)** helps with it.

4 Prefix

This problem asks to find out how many strings in $s_1, s_2, ..., s_n$ begins with t_i when given n strings $s_1, s_2, ..., s_n$ and q queries.

The source code is in A4_P4_121090642.cpp

This is a very classic Trie Tree model. We insert each small piece of string into a tree. In this tree, each node will point to up to 26 child nodes, representing the 26 letters of abc...z. In this way, for each of a string character, we can find the corresponding layer in this tree, if there is this string in the corresponding position in that layer, then it can be output, if not, then it does not exist.

When inserting, we find the pointer to the correct idx of our next string, then we update the counts in each node in order to do the search operation. The operation is implemented by the function **void insert(Node* root, string str)**. The time complexity is $O(n)$, where n is the length of the string. Or we can see the time complexity as $O(lgN)$, where N is the total number of the strings.

If you want to find a string based on the prefix, you can start from the root node and search downwards according to the traversal of the string. If the prefix string has not been traversed and the tree search has ended, then the character cannot be found. After the traversal, output the record value of the node, which is the quantity we found. The operation is implemented by the function **int getNum(Node* root, string prefix)**. The time complexity is $O(n)$, where n

is the length of the prefix string. Or we can see the time complexity as $O(lgN)$, where N is the total number of the strings.