THE CHINESE UNIVERSITY OF HONG KONG, SHENZHEN

CSC 3100

DATA STRUCTURE

Assignment 4 Report

Author:

Ma Kexuan

Student Number:

ID 120090651

December 9, 2022

Problem 1

1. **Possible solutions:** We can use DFS algorithm to preprocess the whole graph by adding middle nodes, and use the original Dijkstra algorithm to get the answer. In this problem, I use adjcency list to store the whole graph, we may also use adjcency matrix to store the whole graph.

2. **I solve the problem by the following logic:** First, for each edge inserted, create two middle nodes corresponding to each directed edge since this is a undirectional graph. Initialize the weight between the original nodes and the middle nodes as the input weight. Then, initialize the distance array and two indicator array used in DFS and Dijkstra. After that, update the edge by DFS algorithm. If the weight from original node1 to middle node is 1/K of the weight from middle node to node2, then we add an edge between node1 and node2 with weight K-1. After the DFS, we just simply use the Dijkstra algorithm to find the shortest path. At last, when we output the result, since we actually double the weight of each original edge, we need to output them by half of the value stored in the distance nodes.

3. **My solution** is better on the space complexity since I use the adjcency list to store the edges, if we use adjcency matrix, since the number of nodes is quite large, and the graph is quite sparse, using matrix will allocate more unused space.

Problem 2

1. **Possible solutions:** We can use two heaps plus an array(linked list, vector) to store the whole tree. We can also use three c++ vectors to hold the case.

2. **I solve the problem by the following logic:** First, I use maxheap and minheap provided by c++ STL, and define an array. I use arrinsert and arrdelete to move the elements in the array when doing insertion and deletion. When doing insertion, if the value is less than the front of the array, push it into the maxheap, check whether size of minheap is differed with maxheap by 2, if so, push back value of array to minheap, insert the max element of maxheap to the front of the array. If value is larger than the back of the array, push it to the minheap, check whether size of minheap is larger than maxheap, if so, doing reverse operation stated above. If the value is between the median array, first use insertion sort like algorithm to insert it into the array, and check whether size of maxheap equals to that of minheap, if so, push the front of array into the maxheap, otherwise push the back of array into the minheap. For the deletion, first delete it from the array, check whether size of maxheap equals to that of minheap, if so, insert the top element of minheap to the back of the array, otherwise insert the top element of maxheap to the front of the array.

3. **My solution** is better than the three vectors version since doing insertion on heap only require logn time complexity, if we use 3 vectors, and after insertion always use

quicksort to sort the left and right array, the complexity would be nlogn.

Problem 3

1. **Possible solutions:** We can use Prim or Kruskal algorithm to finish this task. The task is equivalent to find a maximum spanning tree.

2. **I solve the problem by the following logic:** I use Kruskal algorithm to finish this question. I define two functions, which are find and build, to use union-find disjoint sets to reduce the time complexity of the algorithm. First, I calculate all of the weight by the formula given by the problem, and use distance list to store the values since Kruskal algorithm use the length of each edge to simultaneously create several trees, and merge them together at last. Then I sort the initialize distance array by the sorting algorithm provided by c++. After that, I use the Kruskal algorithm to select edges, and output the final result.

3. **My solution** may be not as good as the Prim algorithm since this is an complete graph, so Prim may actually be better than Kruskal. However, since selecting maximum edges is easy to implement and the union-find disjoint sets are also easier to implement, so I actually sacrifice the time complexity to make the whole question easier to implement.

Problem 4

1. **Possible solutions:** We can use the Trie tree structure to finish the task. We can also use Brute-Force algorithm, which means using a single big array to store all the strings and search by a for loop.

2. **I solve the problem by the following logic:** I define a class called Trie, which has two private attributes, count and a pointer to an array of size 26, which stores the 26 alphabets. When doing insertion, we start from the root, and scan the characters in the string one by one, if we find the space hasn't been used yet, we create a node there, and increase count by 1, else only do the increase, and make the pointer pointing at the deeper location. When doing searching, we also start at the root, and scan the characters in the string one by one. If we haven't encounter the null pointer, we continue to search deeper, if after the whole search, we end at a null position, we simply return 0 since there is no this prefix in the whole tree, otherwise we return the count, which represents how many times this end letter of the prefix string has been used.

3. **My solution** is better than the brute-force one since it requires lots of time when searching one by one in lots of for loops. If the sum of the length of the strings is n, we need O(n) to insert, if the length of the prefix is k, we need O(k) in each searching process, it is quite better than the simple algorithm.