Algorithm Assignment4

Xinyu Jiang

September 2018

1

(15 pts) As discussed in the class, the single-link k clustering problem can be solved based on minimum spanning tree (MST) algorithms. There are two natural algorithms:

(a) Apply Kruskal's algorithm and stop when there are k components, and

(b) Apply Prim's algorithm to obtain a MST and then delete $k - 1$ longest edges.

Are the above two algorithms equally good, or one is better than the other? Please justify your answer.
For Kruskal's algorithm, it needs to go through all the edges and sort all of them, then it need to check the shortest path that could not connect the whole graph, if it would connect the whole graph, choose the next shortest therefore the run time for Kruskal's algorithm is O(log(n)). Because Kruskal's algorithm need to sort all the edges, so if there are a lot of edges will affect the run time, therefore, Kruskas's algorithm for MST is good to use in graph that does not have a lot of edges.
For Prim's algorithm, it find one point as its start point, then find the shortest edge that would not cause the graph connected, therefore the run time is O(n^2). Because then Prim's algorithm does not care about how many edges in total, so it is more efficient using in graph that contains a lot of edges.
Therefore, Kruskal's algorithm is better using in graph that does not contain a lot of edges, and Prim's algorithm is better using in graph that contain a lot of edges.

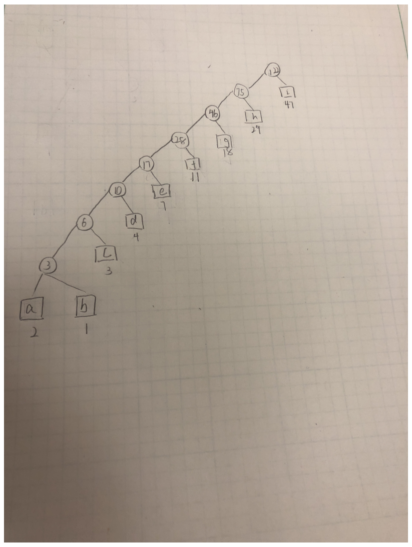
2

(15 pts) Alice is writing a secret message to Bob and wants to prevent it from being understood by eavesdroppers. She decides to use Huffman encoding to encode the message, based on the symbol frequencies that are given by the *Lucas numbers*. The nth Lucas number is defined as $L_n = L_{n-1} + L_{n-2}$ for $n > 1$ with base cases $L_0 = 2$ and $L_1 = 1$.

1. For an alphabet of $\Sigma = \{a, b, c, d, e, f, g, h, i\}$ with frequencies given by the first $|\Sigma|$ Lucas numbers, give an optimal Huffman code and the corresponding encoding tree for Alice to use.

2. Generalize your answer to (1) and give the structure of an optimal code when the frequencies are the first n Lucas numbers.

a).Because $L_0 = 2$ and $L_1 = 1$, a appears 2 times and b appears 1 time, so that according to the function $L_n = L_{n-1} + L_{n-2}$, c appears 3 times, d appears 4 times ,e appears 7 times, f appears 11 times, g appears 18 times, h appears 29 times, i appears 47 times
Graph for corresponding encoding tree.



Suppose going to the left is 0 and going to the right is 1. The Huffman code for the set is

```
a 00000000
b 00000001
c 0000001
d 000001
e 00001
f 0001
g 001
h 01
i 1
```

b). Because $L_n = L_{n-1} + L_{n-2}$, then, $L_3 = L_2 + L_1$ and $L_4 = L_3 + l_2$, so $L_n = L_n - 1 + L_n - 2 + ... + L_1,$. For the Huffman coding, suppose there are n number in the set, then the most frequent letter is 1, and the next frequent letter is is 01, the third frequent is 001, (based on the Huffman coding on 2a), the second last frequent number is (n-1) '0's+1, the least frequent is (n-1) '0's.

3

(15 pts) Suppose you are given the minimum spanning tree T of a given graph G (with n vertices and m edges) and a new edge $e = (u, v)$ of weight w that will be added to G. Give an efficient algorithm to find the MST of the graph $G \cup e$, and prove its correctness. Your algorithm should run in $O(n)$ time.

```
addEdge(MST,newEdge){
    tempMaxEdge.weigth=0
    while(i to all edges in the graph){
        if(MST[i].weight>tempMaxEdge.weight){
            tempMaxEdge=MST[i]
        }
    }
    //find the maxWeight in the MST
    if(tempMaxEdge.weight>newEdge){
        delete the maxWeight edge in MST
        add newEdge
        //It is not necessary to replace the max edge
        //the added edge have to be smaller than the biggest weight in the original grap
    }
}
```

Because the graph is MST, so that adding an edge will cause the graph be connected which is no longer MST
Proof: By the definition of greed algorithm , all sub-graph of MST is optimal, so that the whole graph is optimal. By the definition of MST, adding one edge will cause a sub-graph connected, so that it is not optimal. Therefore, in order to keep the the sub-graph optimal, we have to delete the highest weight of the original graph so that every sub-graph is optimal, since every sub-graph is optimal, the graph is MST and the algorithm is correct.

4

(20) Let $G = (V, E)$ be a directed weighted graph of the US highway system, with edge weights being distances between different cities/intersections. Boulder and Denver are two vertices of G, and $k > 0$ is a given integer. Assume that you will stop at every city/intersection you pass by. Design an algorithm to find the shortest path from Boulder to Denver that contains exactly k stops (excluding Boulder and Denver), and prove its correctness. Notice that a k-stop path from Boulder to Denver may not exist. So, your algorithm should also take care of such possibility.

```
findPath(graph, source, destination, k){
    destinationPath[]
    shortestKPath.distance=max_int;
bool findPath=false;
i=0;
while(did not find all path to dest){
    tempPath
    if(vertex going to destinaton){
        update tempPath
        add to destionition[i]
        i++;
    }
}
for(i=0 to all elements in destinationPath){
    if(destationPath[i].stops==k){
        findPath=true;
       if(distationPath[i].distance<shortestKPath.distance){
             shortestKPath=distationPath[i]
        }
    }
}
if(findPath==false){
    print("Can not find path that stops k times");
    return NULL;
}else{
    return shortestKPath;
}
}
```

Logic for algorithm:
Find all path from Boulder to Denver and track the how many times it stops, save all path to an array, then set a flag to check if there are paths that have k stops in the array, if there are no path has k stops, return NULL, if there are paths stops time equal to k, find the path that have stop times equal to k, then find the shortest path that have k stops

Proof by induction:
Base case: Suppose there are only one edge in the graph, then the algorithm would just add the edge to destinationPath array, since it stops 0 times, if k=0 the shortest path is just the one edge, if k is not equal to 0, the function will return NULL,which the algorithm holds.
Induction hypothesis:Suppose the algorithm holds for n edges in the graph
Induction steps: We need to show that for k+1 edges in the graph, the algorithm also holds.
For n+1 edges, the function will go through all the edges in the graph and find all the paths from source to destination and save it to an destinationPath array, then in the destinationPath array, check if there are paths have k stops, if there are no paths that stops k times, then return NULL. If there are paths that stops k times return the shortest distance path that have k stops. Therefore,

the algorithm holds for n+1 edges in the graph so that the algorithm is correct.

5

(15 pts) You're asked to compute the in- and out- degrees of all vertices in a directed multigraph $G = (V, E)$. But you're not sure how to represent the graph so that the calculation is most efficient. For each of the following three possible representations, express your answers in asymptotic notation in terms of $|V|$ and $|E|$, and justify your answers.

1. An *edge list* representation. Assume vertices have arbitrary labels.
2. An *adjacency list* representation. Assume the vector's length is known.
3. An *adjacency matrix* representation. Assume the size of the matrix is known.

a).For edge list representation, because it go through all the vertexes in the list, which means the in and out run time is O(V)
b). For adjacency list representation, because it always find edges connect to one vertex, so the in and out run time is O(V+E)
c) For adjacency matrix representation, because it shows all vertex's connection status in a matrix, so that the in and out run time is O(V^2)