Romil V. Shah
20008692

CS / CPE 600
Prof. Reza Peyrovian

Homework Assignment 7
Submission Date: 11/06/2022

Q1. No. 15.6.16

Show how to modify the Prim-Jarník algorithm to run in $O(n^2)$ time.

Sol.

Prim's Algorithm is a greedy algorithm that is used to find the minimum spanning tree from a graph. Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized. Prim's algorithm starts with the single node and explores all the adjacent nodes with all the connecting edges at every step. The edges with the minimal weights causing no cycles in the graph got selected.

Figure 15.4.1: The Prim-Jarnik algorithm for the MST problem.

Algorithm PrimJarníkMST(G):

Input: A weighted connected graph G with n vertices and m edges
Output: A minimum spanning tree T for G

Pick any vertex v of G
$D[v] \leftarrow 0$
for each vertex $u \neq v$ **do**
    $D[u] \leftarrow +\infty$
Initialize $T \leftarrow \emptyset$.
Initialize a priority queue Q with an item $((u, \text{null}), D[u])$ for each vertex u, where (u, null) is the element and $D[u]$ is the key.
while Q is not empty **do**
    $(u, e) \leftarrow Q$.removeMin()
    Add vertex u and edge e to T.
    for each vertex z adjacent to u such that z is in Q **do**
        // perform the relaxation procedure on edge (u, z)
        if $w((u, z)) < D[z]$ **then**
            $D[z] \leftarrow w((u, z))$
            Change to $(z, (u, z))$ the element of vertex z in Q.
            Change to $D[z]$ the key of vertex z in Q.
return the tree T

Here, we modify the Prim-Jarn´ık algorithm to run in $O(n^2)$ time. We use the adjacency matrix as the data structure, so that we can achieve the given time complexity.

1. We initialize the set, setMST which will keep track of all the vertices already on the minimum spanning tree.
2. In the input graph, we will assign key values to all the vertices. Then, we set one vertex as zero (initial) and others as Infinite.
3. If setMST contains all the vertices, then we do nothing.
4. Else, select the vertex z with minimum value and include that vertex in the setMST. Now update all the values of key that are adjacent to z.
5. For every adjacent vertex v, if weight of edge z-v is less than previous key value of v, update the key value as weight of z-v.

Key values are used only for vertices which are not present in the setMST.

We can use three arrays to represent the above algorithm, one setMST[] where for any vertex v setMST[v] is true include the vertex in MST, other one is for storing key values as keyMST[] and third one is the parentMST[] to store index of the parent nodes in minimum spanning tree. parentMST is the output array which will store the resulting MST.

Two outers for loops will result in time complexity as n^2. When we represent the above algorithm using adjacency matrix, it will run in $O(n^2)$ time.

Q2. No. 15.6.22

Suppose you are a manager in the IT department for the government of a corrupt dictator, who has a collection of computers that need to be connected together to create a communication network for his spies. You are given a weighted graph, G, such that each vertex in G is one of these computers and each edge in G is a pair of computers that could be connected with a communication line. It is your job to decide how to connect the computers. Suppose now that the CIA has approached you and is willing to pay you various amounts of money for you to choose some of these edges to belong to this network (presumably so that they can spy on the dictator). Thus, for you, the weight of each edge in G is the amount of money, in U.S. dollars, that the CIA will pay you for using that edge in the communication network. Describe an efficient algorithm, therefore, for finding a ***maximum spanning tree*** in G, which would maximize the money you can get from the CIA for connecting the dictator's computers in a spanning tree. What is the running time of your algorithm?

Sol.

We can solve this with the help of Kruskal's Algorithm.

Kruskal's Algorithm is used to find the minimum spanning tree for a connected weighted graph. The main aim of the algorithm is to find the subset of edges by using which we can traverse every vertex of the graph. It follows the greedy approach that finds an optimum solution at every stage instead of focusing on a global optimum.

Figure 15.3.1: Kruskal's algorithm for the MST problem.

Algorithm KruskalMST(G):
Input: A simple connected weighted graph G with n vertices and m edges
Output: A minimum spanning tree T for G

for each vertex v in G **do**
    Define an elementary cluster $C(v) \leftarrow \{v\}$.
Let Q be a priority queue storing the edges in G, using edge weights as keys
$T \leftarrow \emptyset$ // T will ultimately contain the edges of the MST
while T has fewer than $n - 1$ edges **do**
    $(u, v) \leftarrow Q$.removeMin()
    Let $C(v)$ be the cluster containing v
    Let $C(u)$ be the cluster containing u
    if $C(v) \neq C(u)$ **then**
        Add edge (v, u) to T
        Merge $C(v)$ and $C(u)$ into one cluster, that is, union $C(v)$ and $C(u)$
return tree T

Algorithm for maximum spanning tree.

Let us consider the undirected graph G with n vertices and m edges.

1. For all the edges in graph G, sort the edges in decreasing order by weight.
2. Let S be the set comprising the maximum weight spanning tree. Set S = ∅.
3. Add the first edge to S with the maximum weight,
4. Add the next edge to S if there is no cycle present in S.
5. Next, we check that if there are no edges remaining, we exit the loop.
6. If S completes n – 1 edge, then terminate and show results in S otherwise repeat the above 2 steps.

Sorting the edges according to decreasing order will take m log n time. Performing Kruskal Algorithm will take O(m log n).
So, together the algorithm will run in O(m log n) time.

Q3. No. 15.6.25

Imagine that you just joined a company, GT&T, which set up its computer network a year ago for linking together its n offices spread across the globe. You have reviewed the work done at that time, and you note that they modeled their network as a connected, undirected graph, G, with n vertices, one for each office, and m edges, one for each possible connection. Furthermore, you note that they gave a weight, $w(e)$, for each edge in G that was equal to the annual rent that it costs to use that edge for communication purposes, and then they computed a minimum spanning tree, T, for G, to decide which of the m edges in G to lease. Suppose now that it is time renew the leases for connecting the vertices in G and you notice that the rent for one of the connections not used in T has gone down. That is, the weight, $w(e)$, of an edge in G that is not in T has been reduced. Describe an $O(n + m)$-time algorithm to update T to find a new minimum spanning, T', for G given the change in weight for the edge e.

Sol.

We are given a graph G, which has n vertices and m edges, weight of the edge as w(e). Let (u, v) be the edge not in T but has been reduced.

Breadth first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes. Then, it selects the nearest node and explores all the unexplored nodes. While using BFS for traversal, any node in the graph can be considered as the vertices of a tree data structure or a graph.

Depth first search algorithm is a recursive algorithm to search all the initial node of graph G and goes deeper until we find the goal node or the node with no children. Because of the recursive nature, stack data structure can be used to implement the DFS algorithm. The process of implementing the DFS is like the BFS algorithm.

We can perform Depth First Search or Breadth First Search, to find the unique simple path from u to v in T. Find an edge e of maximal weight on that path. If the weight of an edge e is greater than that of (u, v), then replace the edge e which is in T with (u, v).

The running time of the algorithm to perform a Depth First Search or Breadth First Search is O(n + m) time.

Q4. No. 16.7.19

Let N be a flow network with n vertices and m edges. Show how to compute an augmenting path with the largest residual capacity in $O((n+m)\ \log\ n)$ time.

Sol.

We can solve this using Kruskal's Algorithm

Kruskal's Algorithm is used to find the minimum spanning tree for a connected weighted graph. The main aim of the algorithm is to find the subset of edges by using which we can traverse every vertex of the graph. It follows the greedy approach that finds an optimum solution at every stage instead of focusing on a global optimum.

Augmented path can be computed using maximum spanning tree. We can do this by multiplying each weighted edge present in the minimum spanning tree by (-1) and then applying Kruskal algorithm will give augmented path with maximum residual capacity.

The multiplication of every weighted edge with (-1) will take O(m) time and Kruskal algorithm will take O((n + m) log n). So, this together will take O((n + m) log n) time.

Q5. No. 16.7.30

Consider the previous exercise, but suppose the city of Irvine, California, changed its dog-owning ordinance so that it still allows for residents to own a maximum of three dogs per household, but now restricts each resident to own at most one dog of any given breed, such as poodle, terrier, or golden retriever. Describe an efficient algorithm for assigning puppies to residents that provides for the maximum number of puppy adoptions possible while satisfying the constraints that each resident will only adopt puppies that he or she likes, that no resident can adopt more than three puppies, and that no resident will adopt more than one dog of any given breed.

Sol.

There are n residents and m puppies. Each n has subset of m that he/she interested in. Finding the assignment of puppies to residents. Considering to disjoint sets n and m.

1. Using bipartite matching, add source and sink nodes sr and sk respectively where sr will have outgoing edges to all residents(n) and sk has incoming edges from the puppies set called m.
2. A vertex in n from source has capacity 3 while from m to sk capacity to all the edges as 1.
3. If there is a matching of k edges, there is a flow f of value k. Match the puppy with only one resident.
4. Now calculating the maximum flow using Ford-Fulkerson in the above graph.
5. Find the augmented path by setting all residual capacity to 0. Adding flow at every edge from source to sink.

Bipartite matching will take O(n + m) time and Ford - Fulkerson will take O(n (n + m)) time. So, the run time of the algorithm is O(nm) time.

Q6. No. 16.7.34

A limousine company must process pickup requests every day, for taking customers from their various homes to the local airport. Suppose this company receives pickup requests from n locations and there are n limos available, where the distance of limo i to location j is given by a number, d_{ij}. Describe an efficient algorithm for computing a dispatchment of the n limos to the n pickup locations that minimizes the total distance traveled by all the limos.

Sol.

Let us consider the above description, where there are n limos and n pickup locations. For each pickup location they have only 1 limo. Now we must find limos to pick up location as many of the requests matched while minimizing the travelling cost and distance.

This problem can be solved using Min-Cost Flow algorithm.

1. Initially the flow f is empty and then, we build maximum flow by a series of augmented path along with minimum cost.
2. Assigning the weight to the edges of the residual graph R_f and reducing the time for the shortest path by changing the weight in residual graph R_f so that they are all non-negative.
3. Then applying Dijkstra's Algorithm on residual graph R_f to calculate the shortest path.
4. Compute the residual capacity of an augmented path π. Checking for backward and forward edge for a particular edge in π.
1. Now, the calculated flow f is a maximum flow of minimum cost.

The running time of Dijkstra Algorithm is O(n log n) and the minimum-cost maximum flow f will be O(|f| n log n). So, the run time of the algorithm is O(|f| n log n).