C -15.6 Show how to modify the Prim-Jarn ́ık algorithm to run in O(n2) time.

**Solution:**

Running time O(n2) of the Prim- Jarn ́ık algorithm. Using adjacency matrix as the data structure we can achieve the given time complexity.

1. Initialize the set, *setMST* which keep tracks of all the vertices already in the minimum spanning tree.
2. In the input graph assign key values to all the vertices. Set one vertex as zero(initial) and others as INFINITE.
3. If *setMST* contains all the vertices, then 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 for *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.   
**Running time:** Representing above algorithm using adjacency matrix will run in O(n2) time. Two outers for loops will result in the time complexity as n2.

A 15.1 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?

**Solution:**

Describing the algorithm for maximum spanning tree. 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. Let S be the set comprising the maximum weight spanning tree. Set S = ∅
2. Add the first edge to S with maximum weight.
3. Add the next edge to S if there is no cycle present in S. Check if there are no edges remaining exit the loop.
4. If S completes *n-1* edges, then terminate and show results in S otherwise repeat above step.

**Running time:** Sorting the edges according to decreasing order will take m log m time. And performing Kruskal algorithm will take O(m log n).

So, together it will run in O(m log n) time.

A-15.4 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.

**Solution:**

Given a graph G, having *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. We can perform DFS or BFS, 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)*.

**Running time**: The running time of the algorithm to perform DFS or BFS is *O(n+m)*

OR

Given graph G, containing minimum spanning tree (MST) T but if we add a minimum weight edge (e) in MST (T) then it forms a cycle. Now considering that, the edge whose weight is greater than all other edges in cycle is not consider in the MST. Going through all the edges in a cycle find an edge(u,v) with the largest weight and replace this edge with edge (e). This forms the new minimum spanning tree.

**Running time**: The running time of the algorithm is *O(n+m).* Adding an edge in the will take O(1) and traversing through every edge and every vertex will take O(m+n).

C-16.3 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.

**Solution:**

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.

**Running time**: The running time of multiplying every weighted edge with (-1) will take O(m) time and Kruskal algorithm will take O((n+m) logn). This together will take O((n+m) logn).

A-16.3 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.

**Solution:**

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

**Running Time**: The running time of the algorithm is O(nm). Bipartite matching will take O(n+m) and Ford-Fulkerson will take O(n(n+m)) time.

A-16.7 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, dij . 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.

**Solution:**

Consider the above description there are n limos and n pick up location. For each pick up location they have only 1 limo. Now we have to 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 building maximum flow by a series of augmented path along with minimum cost.
2. Assigning the weight to the edges of the residual graph Rf and reducing the time for the shortest path by changing the weight in residual graph Rf so that they are all non-negative.
3. Then applying Dijkstra’s Algorithm on residual graph Rf 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.
5. Now the calculated flow *f* is a maximum flow of minimum cost.

**Running Time:** 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 total running time of the algorithm is O(|*f* | n log n)