CS 600 HW 7
SHUCHI PARAGBHAI MEHTA
CWID: 20009083

- 15.6.16

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

To run Prim-Jarník algorithm in $O(n^2)$ runtime, we can use adjacency matrix as the data structure.

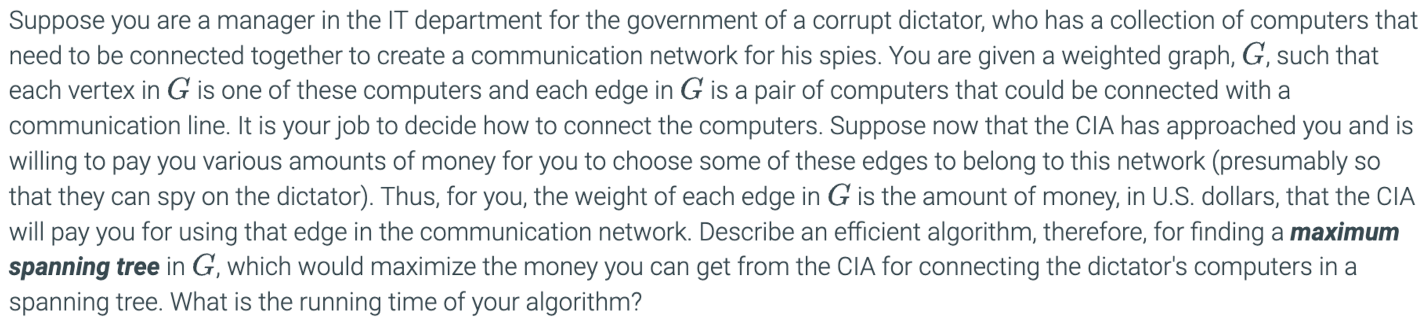
1. Create the set, setMST, which records all the vertices that are already in the minimal spanning tree.
2. Assign key values to each vertex in the input graph. Set all but one vertex to INFINITE and one to zero (the initial).
3. If *setMST* contains all the vertices, then do nothing
4. Else, choose the vertex z with the lowest value and add it to the setMST. Update all the key values that are close to z currently.
5. For each adjacent vertex v, update the key value to reflect the weight of the edge z-v if it is less than the previous key value of v.

Here, only vertices absent from the setMST are used as key values.

To describe the above algorithm, we can use three arrays:
one setMST[], where for any vertex v setMST[v] is true, include the vertex in MST,
another keyMST[], to storage of key values,
and a third one parentMST[], to storage of the index of the parent nodes in minimum spanning tree.
The output array, or parentMST, is where the final MST will be stored.

Running time:
The above procedure will run in $O(n^2)$ time when implemented by an adjacency matrix. The time complexity will be n^2 with two outer for loops.

- 15.6.22

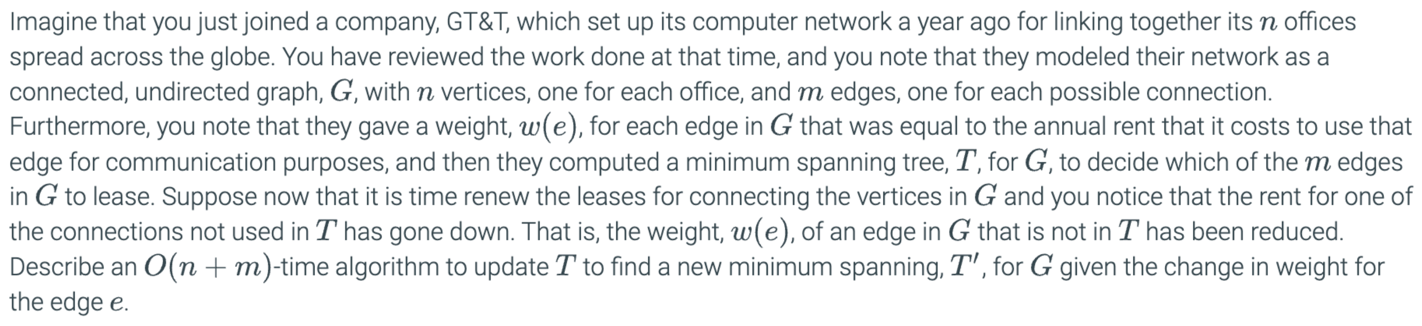

Suppose you are a manager in the IT department for the government of a corrupt dictator, who has a collection of computers that need to be connected together to create a communication network for his spies. You are given a weighted graph, G, such that each vertex in G is one of these computers and each 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?

Algorithm for maximum spanning tree. Let's consider an undirected graph G with n vertices and m edges.

1. Sort the edges in graph G in decreasing weight order for all the edges. Let S represent the collection that includes the maximum weight spanning tree. Set S = ∅
2. Add the first edge to S with maximum weight.
3. If S doesn't include a cycle, add the following edge to S. Check that the loop is over if there are no more edges.
4. If S has n-1 edges completed, then stop and display the results in S; otherwise, repeat the previous step.

Running time:
It will take m log m time to sort the edges in decreasing order. And it will take $O(m\ log\ n)$ time to run the Kruskal algorithm. Thus, it will take $O(m\ log\ n)$ time.

- 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.

The weight of an edge in a graph G with n vertices and m edges is given as w(e). Let (u, v) represent the decreased edge that is not in T. To find the one unique simple path from u to v in T, we can use DFS or BFS. Find the edge with the greatest 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 DFS or BFS is $O(n+m)$.

OR

Given graph G, containing minimum spanning tree (MST) T, if we add a minimum weight edge (e) to it, it generates a cycle. In consideration of this, the edge whose weight exceeds that of all other edges in the cycle is not considered by the MST. Finding the edge(u,v) with the highest weight among all the edges in a cycle and replacing it with edge (e). The new minimum spanning tree is created by this.

Adding an edge, will take $O(1)$ and traversing through every edge and every vertex will take $O(m+n)$. Thus, the running time of the algorithm is $O(n+m)$.

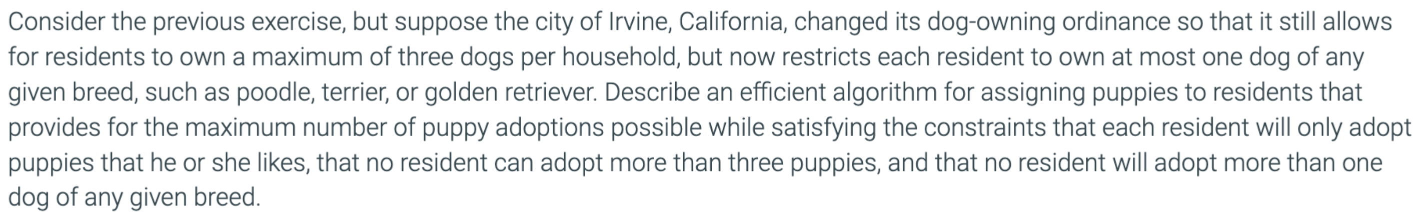
- 16.7.19

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.

Maximum spanning tree can be used to compute an augmented path. By increasing each weighted edge in the least spanning tree by (-1), followed by the Kruskal Algorithm, we can achieve this goal and provide an augmented path with the highest residual capacity.

Running time:
Every weighted edge will need to be multiplied by (-1), which will take $O(m)$ time, and the Kruskal method will run in $O((n+m)\ logn)$ time. Thus, the total time required is $O((n+m)\ logn)$.

- 16.7.30

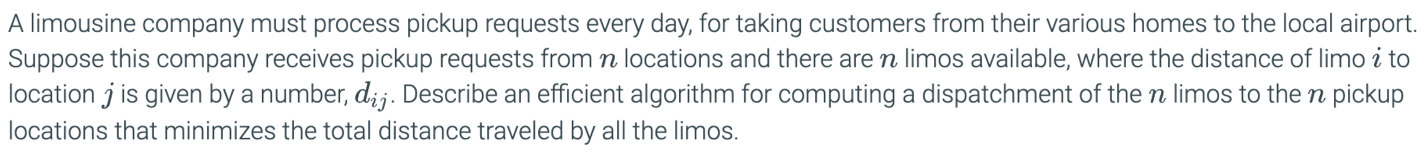

Consider the previous exercise, but suppose the city of Irvine, California, changed its dog-owning ordinance so that it still allows for residents to own a maximum of three dogs per household, but now restricts each resident to own at most one dog of any given breed, such as poodle, terrier, or golden 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.

There are m puppies and n residents. Each n is interested in a specific subset of m. finding the residents to whom puppies were assigned. Considering the sets n and m to be distinct.

1. Add source and sink nodes, sr and sk, using bipartite matching, with sr having outward edges to all residents(n) and sk having incoming edges from the puppies set referred to as m.
2. From the source, a vertex in n has capacity 3, whereas from m to sk, all edges have capacity as 1.
3. If k edges match, then there is a flow f with value k. Match just one resident with the puppy.
4. In the graph above, use Ford-Fulkerson to determine the maximum flow.
5. By setting all residual capacity to 0, locate the increased path. Increasing flow from source to sink at every edge

Running Time:
The algorithm's execution time is $O(nm)$. Ford-Fulkerson and bipartite matching will both take $O(n+m)$ and $O(n(n+m))$ time, respectively.

- 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.

As there are n limos and n pick-up locations in the description above. There is only one limo available for each pickup location. Now that we have a good number of requests that match, we need to find limos to pick up locations while keeping travelling cost and distance to a minimum.

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

1. The flow f starts out empty and builds up to its maximum flow by using a sequence of augmented paths while maintaining a low cost.
2. Assigning weight to the edges of the residual graph R_f and speeding up the shortest path by making all the weights in the residual graph R_f positive.
3. Using Dijkstra's Algorithm to determine the shortest path on the residual graph R_f.
4. Calculate an upgraded path's remaining capacity. checking the forward and backward edges of a specific edge in π.
5. The calculated flow, f, is now the highest flow at the lowest cost.

Running Time:
The Dijkstra Algorithm will take $O(n\ log\ n)$ running time, while the runtime of minimum-cost maximum flow f will be $O(|f\ |\ n\ log\ n)$. Therefore, the algorithm's overall running time is $O(|f\ |\ n\ log\ n)$.