CS 600 Advanced Algorithms

Sri Dhanush Reddy Kondapalli

CWID: 10476150

Homework 7

1 C-15.6.16 Show how to modify the Prim-Jarnik algorithm to run in $O(n^2)$ time.

The key difference in this example is that the priority queue, Q, is implemented as an unordered doubly linked list. This allows us to insert and update the key in O(1) time for any vertex and remove the minimum in O(n) time. In the algorithm, we do O(m) insertions and updates and O(n) removeMin() operations. Because m is $O(n^2)$, the worst-case running time is $O(n^2)$.

Algorithm Solution: We investigated the Prim-Jarnik algorithm (Algorithm 15.8), which has a running time of O(m log n) (Theorem 15.6) and can extract vertex u in each iteration in O(log n) time. By modeling the graph G = (V,E) as an adjacency matrix A, we may implement the Prim-Jarnik algorithm in $O(n^2)$ time. To locate nearby vertices of vertex u, we must look in the adjacency matrix for the row u. We assume that the edge weights are stored in the adjacency matrix, and that unconnected edges have weight 0. The modified Prim-Jarnik algorithm is shown below.
Algorithm modifiedPrimJarnikMST(G, r):

```
Input: A weighted connected graph G(V,E) and
       r is the set of vertices in the current tree.
Output: A maximum spanning tree T for G

for each u ϵ in V[G] do
    key[u] ← ∞;
    π[u] ← NIL;

key[r] ← 0;
Q ← V[G];
while Q ≠ ϕ do
    u ← EXTRACT − MIN(Q);
```

```
for each v ϵ V[G] do
    if A[u,v] ≠ 0 and v ϵ Q and A[u,v] < key[v] then
        π[v] ← u;
        key[v] ← A[u,v];

return the tree T;
```

Running time: $O(n^2)$ because outer loop as well as inner loop has |V| variables. i.e. total number of vertices here n.

2 C-15.6.22 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?

Describe the maximum spanning tree algorithm. Consider the graph G, which has n vertices and m edges:

1. Sort all the edges in graph G in decreasing order by weight. Let S denote the set that contains the greatest weight spanning tree. Make S = phi
2. With the most weight, add the first edge to S.
3. If there is no cycle in S, add the next edge to S. If there are no more edges, leave the loop.
4. If S finishes n-1 edges, then stop and display the results in S; otherwise, continue the preceding step.

Running time: It will take m log m time to sort the edges in decreasing order. And doing the Kruskal method will take O minutes (m log n).
As a result, it will complete in O(m log n) time.

3 A-15.6.25 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.

We begin by creating all n vertices, which represent n rooms, in this procedure. The edges for constructing a minimum-cost spanning tree for G should then

be chosen. Let Q be a priority queue that stores all the edges in G in non-descending order, which implies we should sort the edges by cost weight. We can insert all the edges with 50-foot cables after the edges with 12-foot cables because there are only two types of wires, which takes O(m) time. The edge with the lowest cost in Q is then chosen and added to the current tree. If no cycle is generated, we shall include this edge.

4 C-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.

Because greatest residual capacity equals minimum weighted path, this issue can be solved using Dijkstra's method.
Each vertex has a variable d[v] linked with it. We set the d values of all vertices to 0, with the exception of the source (the vertex corresponding to a), which is set to Assume we have an advantage (u, v). If mind[u], w(u, v) $>$ d[v], we should change d[v] to mind[u], w(u, v) (since the path from a to u and then to v has bandwidth mind[u], w(u, v), which is more than the one we now have). The overall execution time is O((n + m)log n).

Alternate Solution:
The minimal capacity of one of an augmenting path's edges is its residual capacity. As a result, we want to discover the greatest capacity path from s to t. We may compute this using a maximum spanning tree approach (as shown in A-15.1 above), which is equivalent to a minimal spanning tree algorithm with all weights multiplied by 1. In this tree, the path from s to t will be a maximum capacity path (using a similar argument used to prove the important fact about minimum spanning trees).

5 A-16.7.30 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 n locals and m pups. Each n has a subset of m in which he or she is interested. Identifying and assigning pups to residents. Taking into account the disjoint sets n and m.

1. Create source and sink nodes sr and sk using bipartite matching, with sr having outgoing edges to all residents(n) and sk having incoming edges from the pups set labeled m.
2. A vertex in n from the source has capacity 3 whereas all edges from m to sk have capacity 1.
3. If k edges match, there exists a flow f of value k. Only pair the puppy with one of the residents.
4. In the given graph, calculate the maximum flow using Ford-Fulkerson.
5. Determine the increased route by reducing all residual capacity to zero. Adding flow from source to sink at each edge

Running time: The algorithm's execution time is O. (nm). Bipartite matching takes O(n+m) time, whereas Ford-Fulkerson takes O(n(n+m)) time.

6 A-16.7.34 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.

In the above description, there are n limousines and n pick-up locations. They only have one limo for each pick-up location. Now we must locate limos to pick up as many of the requests as possible while limiting trip costs and distance. This problem can be solved using Min-Cost Flow algorithm:

1. Initially, the flow f is empty, and then the maximum flow is built by a sequence of augmented paths with the lowest cost.

2. Assigning weight to the edges of the residual graph Rf and decreasing the time for the shortest path by modifying the weights in the residual graph Rf to be entirely non-negative.

3. Applying Dijkstra's Algorithm to the residual graph Rf to find the shortest path.

4. Determine an increased path's residual capacity. Checking for the rear and forward edges of a certain edge n

5. The computed flow f is now a maximum flow with the lowest 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 $O(|f|n \log n)$