Ali Abdullah Ahmad
20031246
CS 600

Q1. No. 15.6.16
Show how to modify the Prim-Jarník algorithm to run in $O(n^2)$time
Ans.

Prim-Jarník Algorithm ($O(n^2)$ time complexity)
Modifications

1. Use an adjacency matrix instead of an adjacency list for storing edge weights. This allows O(1) edge lookups.
2. Replace the priority queue with a simple array that keeps track of the minimum weight edge connecting each vertex to the MST.
3. Find the minimum edge in O(n) time by iterating over all vertices instead of using a priority queue.

```
def prim_jarnik_n2(graph, n):
    in_mst = [False] * n                    # Tracks if a node is in MST
    min_weight = [float('inf')] * n         # Min edge weight to MST
    parent = [-1] * n                       # Stores MST structure

    # Start from the first vertex
    min_weight[0] = 0

    for _ in range(n):
        # Find the vertex u with the smallest min_weight value
        u = -1
        for v in range(n):
            if not in_mst[v] and (u == - 1 or min_weight[v] < min_weight[u]):
                u = v

        in_mst[u] = True                    # Include vertex u in MST

        # Update weights of the adjacent vertices
        for v in range(n):
            if graph[u][v] and not in_mst[v] and graph[u][v] < min_weight[v]:
                min_weight[v] = graph[u][v]
                parent[v] = u

    # Return MST as a list of edges
    return [(parent[v], v, graph[parent[v]][v]) for v in range(1, n) if parent[v] != -1]
```

For each vertex, it finds the minimum key vertex in O(n) time.
It updates the min edge weights for O(n) vertices in O(n) time.
Overall complexity: **$O(n) * O(n) = O(n^2)$**.

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?

Ans.

Modified Kruskal's Algorithm for Maximum Spanning Tree

A simple way to compute the Maximum Spanning Tree (MaxST) is by modifying Kruskal's Algorithm, which is originally designed for the Minimum Spanning Tree (MST):

1. Sort all edges in descending order (instead of ascending order as in standard Kruskal's).
2. Use a Union-Find data structure to iteratively add the highest-weight edges while avoiding cycles.
3. Continue until n - 1 edges are included, forming a spanning tree.

Algorithm Steps

1. Sort the edges of the graph in descending order of weight.
2. Initialize a Union-Find (Disjoint Set) data structure to track connected components.
3. Iterate through the sorted edges and add the largest edge to the spanning tree if it does not form a cycle.
4. Stop when n - 1 edges have been added (where n is the number of vertices).
5. The sum of the selected edges is the maximum amount of money the CIA will pay.

Time Complexity:

Sorting the edges: O(mlogm), where m is the number of edges.
Processing each edge (Union-Find operations): O(mα(n)), where α(n) is the inverse Ackermann function, which is practically constant.
Overall Complexity**: O(mlogm),** since sorting dominates.

Q3. No. 15.6.25

(a) 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.

Ans.

Algorithm Explanation

1. Find the unique path from u to v in T:
    - Since T is a tree (a connected acyclic graph), there is a unique simple path between any two vertices.
    - We can use DFS or BFS to find this path efficiently in O(n) time.
2. Find the edge with the maximum weight on this path:
    - As we traverse the path, we track the edge with the maximum weight in O(n) time.
3. Compare the maximum-weight edge with the new edge (u,v):
    - If the new edge (u,v) has a smaller weight than the heaviest edge on the path, replace the heaviest edge with (u,v).
    - Otherwise, the MST remains unchanged.

Time Complexity Analysis

- Finding the unique path (DFS or BFS traversal): O(n)
- Finding the heaviest edge on the path: O(n)
- Replacing the edge (if needed): O(1)

Thus, the total time complexity is **O(n+m)**

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.

Algorithm:

1. Initialize a priority queue (max heap) where each entry is of the form (capacity, vertex).
2. Start from the source vertex with an initial capacity of ∞ (or a very large number).
3. Process each vertex by selecting the edge with the largest residual capacity (greedy approach).
4. For each neighbor v of the current vertex u, update v's capacity if:
    - The residual capacity of edge (u,v) is larger than the current known capacity of v.
    - The new bottleneck capacity (minimum of the edge capacity and u's capacity) is greater than the previous value for v.
5. Continue this process until reaching the sink.
6. The value recorded at the sink is the maximum possible residual capacity of an augmenting path.

Time Complexity Analysis

- The algorithm is essentially Dijkstra's algorithm, which runs in O((n+m)logn) time when implemented with a priority queue (heap).

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.

Ans.

Graph Construction for Maximum Flow

We construct a flow network as follows:

1. Create a source node s and a sink node t.
2. Create a set of resident nodes R (one for each resident).
3. Create a set of puppy nodes P (one for each puppy).
4. Create an intermediate node for each resident-breed pair B to handle the "one dog per breed" constraint.
5. Add directed edges with capacities:
    - From s to each resident node r with capacity 3 (max three puppies per resident).
    - From each resident node r to their breed nodes B with capacity 1 (ensuring only one dog per breed).
    - From each breed node B to the puppy nodes P with capacity 1 (ensuring each puppy is assigned to one resident).
    - From each puppy node p to the sink t with capacity 1 (ensuring each puppy is adopted at most once).

Finding the Maximum Matching

- Compute the maximum flow in this network using the Edmonds-Karp algorithm (which is a BFS-based implementation of Ford-Fulkerson).
- Each unit of flow represents a successful puppy adoption.

Time Complexity Analysis

- Constructing the flow graph: O(n+m).
- Maximum flow computation using Edmonds-Karp (Ford-Fulkerson with BFS): O (VE^2)
    - Here, V (vertices) is O(n+m).
    - E (edges) is also O(n+m).
    - Thus, the time complexity is O((n+m)(n+m))=**O(nm)**

Q6. No. 16.7.34

(a) 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.

Ans.

Graph Construction for Min-Cost Flow

1. Create a source node s and a sink node t.
2. Create nodes for limos (set L) and pickup locations (set P).
3. Add edges:
    - From source s to each limo li with capacity 1 (each limo is used only once).
    - From each limo li to each pickup location pj with capacity 1 and cost dij (distance from limo to pickup location).
    - From each pickup location pj to sink t with capacity 1 (each pickup location can only have one limo).

Algorithm Steps

1. Initialize flow f=0
2. Find the minimum-cost augmenting path from s to t
    - Assign weights to edges based on distance di.
    - Ensure all edge weights are non-negative (potentially use the successive shortest path algorithm or cycle-canceling method).
3. Run Dijkstra's algorithm on the residual graph Rf to find the shortest augmenting path.
4. Augment flow along the found path.
5. Repeat until maximum possible flow is achieved.

Time Complexity Analysis

- Dijkstra's algorithm for finding the shortest path in the residual graph: O(nlogn).
- Total flow units (|f|) = n, since each limo is assigned once.
- Each flow augmentation takes O(nlogn) time.
- Overall, the min-cost max-flow algorithm runs in **$O(n^2 \log n)$ time.**