Weekly Assignment 9

Munia Humaira
Student ID: 21116435
Algorithm Design and Analysis

November 14, 2024

Solution 1:

Alice's Claim is True. In a Minimum Spanning Tree (MST), removing any edge $\langle u, v\rangle$ creates two disjoint components. The edge $\langle u, v\rangle$ is indeed the lightest edge connecting these two components in the graph G. This conclusion follows from the properties of an MST: if there were a lighter edge than $\langle u, v\rangle$ that connected the components V_u and $V \backslash V_u$), then T would not be a minimum spanning tree.

Assume $\langle u, v\rangle$ is in T.

When we remove $\langle u, v\rangle$ from T, two components V_u (containing u) and $V \backslash V_u$ are formed.

Applying the Cut Property - The cut $(V_u, V \backslash V_u)$ separates u and v.

Since T is an MST, $\langle u, v\rangle$ must be the minimum-weight edge crossing this cut; otherwise, a lighter edge could replace $\langle u, v\rangle$, contradicting T's minimality.

So, Alice's claim holds true, as $\langle u, v\rangle$ must indeed be the light edge crossing the cut $(V_u, V \backslash V_u)$ due to the cut property of MSTs.

Solution 2:

(a) T remains an MST if $\alpha < w(u,v)$ and $\langle u,v\rangle \in T$:

The edge $\langle u,v\rangle \in T$, and we decrease its weight from $w(u,v)$ to α, where $\alpha < w(u,v)$.

Lowering the weight of an edge in the MST can't invalidate the MST, as all other edges in T still satisfy the MST properties, and T remains a spanning tree.

Since $\alpha < w(u,v)$, T remains an MST in $G\prime$, as no other configuration can produce a lower total weight without violating the tree structure.

(b) We can add $\langle u,v\rangle$ to T, creating a cycle (since T is a spanning tree).

Identify the maximum-weight edge in this cycle that is not $\langle u,v\rangle$.

If $w(x,y) > \alpha$ for this maximum-weight edge $\langle x,y\rangle$, remove $\langle x,y\rangle$ to break the cycle.

This substitution results in a new MST because it decreases the total weight while preserving connectivity.

Algorithm:

- Add $\langle u,v\rangle$ to T.
- Find the cycle created and identify the heaviest edge $\langle x,y\rangle$ in the cycle.
- If $w(x,y) > \alpha$, replace $\langle x,y\rangle$ with $\langle u,v\rangle$ in T.
- Return the updated tree T'.

This cycle detection and edge weight comparison can be done in $\mathcal{O}(|V|+|E|)$.

(c) Algorithm:

- Remove $\langle u,v\rangle$ from T, creating components C_u and C_v.
- Scan E to find the minimum-weight edge $\langle x,y\rangle$ such that $x \in C_u$ and $y \in C_v$.
- Add $\langle x,y\rangle$ to $T$to obtain the MST for G'.

The process of finding the lightest edge across the cut can be done in $\mathcal{O}(|V|+|E|)$.

Solution 3:

Let $m[j]$ represent the minimum number of coins needed to make the amount j.

- Base Case: $m[0] = 0$, no coins are needed to make the amount zero.
- Recurrence : To determine $m[j]$, consider each coin denomination c_i, to the optimal solution for $m[j - c_i]$ gives a valid solution for $m[j]$

Now, based on Dynamic Programming, initialize an array m where $m[0] = 0$ and other entries are initially set to infinity (or a large number representing an unreachable amount).

Fill in $m[j]$ for $j = 1$ to a using the recurrence relation.

The minimum number of coins required to make the amount a is $m[a]$.

The pseudocode:

```
FUNCTION min_coins(a, coins):
// Initialize DP array with "infinity" for amounts > 0, 0 for amount 0
CREATE array m of size (a + 1) and set all elements to infinity
SET m[0] = 0  // Base case: 0 coins needed to make amount 0

// Fill in DP array using the recurrence relation
FOR j FROM 1 TO a DO:
    FOR each coin c IN coins DO:
        IF j >= c THEN:
            SET m[j] = MIN(m[j], 1 + m[j - c])

// Return the minimum coins needed to make amount a
IF m[a] != infinity THEN:
    RETURN m[a]
ELSE:
    RETURN -1  // Return -1 if amount can't be reached
```

Time Complexity: $\mathcal{O}(ak)$, where a is the target amount and k is the number of coin denominations. This complexity arises because for each amount j from 1 to a, we check each coin denomination.

Solution 4:

Let $m[i]$ represent the maximum number of compatible requests that can be selected from the first i requests.

- Base Case: $m[0] = 0$ If there are no requests to consider, the maximum size of a compatible subset is zero.
- Recurrence: For each request i, there are two options:
  - Exclude request i: In this case, $m[i] = m[i-1]$.
  - Include request i: If we include i, then we add it to the optimal solution of requests compatible with i, which is given by $m[c(i)]$.
- Therefore, the recurrence relation is: $m[i] = max(m[i-1], 1 + m[c(i)])$

Now based on Dynamic Programming, initialize an array $m[0] = 0$. For each i from 1 to n, use the recurrence relation to compute $m[i]$.

The maximum size of a compatible subset is given by $m[n]$, the value computed for all n requests.

The Pseudocode:

```
FUNCTION max_compatible_requests(n, requests, c):
// Initialize DP array
CREATE array m of size (n + 1) and set all elements to 0
SET m[0] = 0  // Base case: 0 requests if no requests are considered

// Fill in DP array using the recurrence relation
FOR i FROM 1 TO n DO:
    // Use the recurrence to determine m[i]
    SET m[i] = MAX(m[i - 1], 1 + m[c[i]])

// Return the maximum size of the compatible subset
RETURN m[n]
```

Time Complexity: $\mathcal{O}(n)$, assuming $c(i)$ values are precomputed. Calculating each $m[i]$ depends only on $m[i-1]$ and $m[c(i)]$, resulting in a linear pass through the requests.