Algorithms

Optimal Metro Network Handout

1 Problem Description

1.1 Problem Description

You are helping plan a new city metro that will help ferry tourists around in as efficient manner as possible. You are given a list of length n, the popularities of different destinations around the city, `popularity[]`. We define the congestion of a destination as the popularity of the destination multiplied by it's degree of separation from the central hub. The hub will be level one, and any stops it connect directly to is level two, and any stops those connect to is level three, and so on. The problem you face is trying to minimize the congestion in the metro. In other words you want to find the layout of the metro such that the total congestion is minimized.

1.2 Assumptions

1. All tourists stay in hotels nearest to the most popular destination and can be served by the one metro stop.
2. All the other destinations are connected by one metro line leading to it, and have at most two metro lines leading out of them.
3. There are no metro lines between metro stops other than the one leading to them, and at most two leading out of them.
4. In any diagram the destination will be labeled by it's popularity, while the recurrence relation will access the popularity by it's respective index in `popularity[]`.

2 Recurrence

2.1 Notation

An important step in dynamic programming is defining a system of notation that provides the necessary means to get you to a solution. Luckily for this problem, the notation is fairly simple. We only need to calculate the optimal cost for metro stop we are focusing on, and the sub-networks it leads to. Thus we can solve the sub-problems that arise from potential network layouts. We'll do this as follows:

- let r be the central hub of each sub network.
- Let $min\_congest(i, j)$ be the minimum congestion layout of destinations between indices $[i, j]$ in `popularity`. It tests each point r between as a potential central hub and recursive calculates the scores of the corresponding sub-networks.

2.2 Recurrence

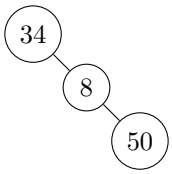
Solving the problem recursively

$$min\_congest(i,j) = \begin{cases} 0 & i > j \\ \texttt{popularity[i]} & i = j \\ \sum_{k=i}^{j} \texttt{popularity[k]} + min_{i \le r \le j}[min\_congest(i, r-1) + min\_congest(r+1, j)] & \text{otherwise} \end{cases} \quad (1)$$

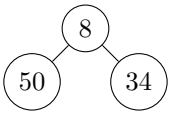
To gain an intuition for what this actually represents, each call to $min\_congest(i, j)$ you will sum all values in `popularity[]` and then find the value which best serves as the central stop on the network, i.e. it minimizes the congestion of it's two sub-networks. On the call of the sub-network $min_congest(i, r-1)$ you sum all values in `popularity[]` in indices $[i, r-1]$ and then choose the central stop on the sub-network that minimizes the congestion. This is repeated until the sub-networks are either empty or have just one value.
To use this to solve your problem with your lists of length n `destinations[0...n-1]` and `popularity[0...n-1]`, you need to solve the recurrence relation: $min\_congest(0, n-1)$.

3 Example and Primer Questions

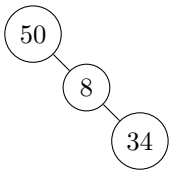
3.1 Example



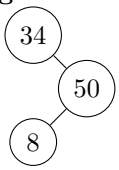
(I) **Congestion:** $1 * 34 + 2 * 8 + 3 * 50 = 200$



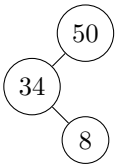
(II) **Congestion:** $1 * 8 + 2 * 50 + 2 * 34 = 150$



(III) **Congestion:** $1 * 50 + 2 * 8 + 3 * 34 = 168$



(IV) Congestion: $1 * 34 + 2 * 50 + 3 * 8 = 158$



(V) Congestion: $1 * 50 + 2 * 34 + 3 * 8 = 142$

Here we can see that the metro design in **(V)** has the least congestion and is thus the solution to our problem.

3.2 Primer Questions

1. In terms of the problem, what do the base cases represent?
2. What does the sum in the recurrence relation do? How does it change on subsequent calls to the recursive function?
3. If you have two or more stops with the same popularity, would they be guaranteed to have the same degree of separation from the central stop in the optimal network layout? Why or why not?