Vikram Thirumaran

Prof. Thanh H. Nguyen

CS 315

February 9th, 2025

HW4

Q1: Floyd-Warshall Algorithm

The Floyd-Warshall algorithm is a dynamic programming approach to finding the shortest paths between all pairs of vertices in a weighted graph. We update the distance matrix iteratively, considering each node as an intermediate vertex.

Algorithm Steps:

1. Initialize the distance matrix D(0) with the given weight matrix W.
2. Iterate over each node k (acting as an intermediate node):
   - For each pair of nodes (i,j), update:

     D(k)[i][j]=min(D(k−1)[i][j],D(k−1)[i][k]+D(k−1)[k][j])
3. Repeat until all nodes have been considered.

Step-by-Step Execution

Then I can compute matrices D(0),D(1),...,D(n) iteratively.

$$W = D(0) = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & \infty & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & 3 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{bmatrix} \rightarrow D(1) = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{bmatrix} \rightarrow D(2) = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{bmatrix}$$

$$D(3) = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 5 & 0 \end{bmatrix} \rightarrow D(4) = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 5 & 0 \end{bmatrix} \rightarrow D(5) = \begin{bmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 5 & 0 \end{bmatrix}$$

$$D(6) = \begin{bmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ -5 & -3 & 0 & -1 & -6 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 5 & 0 \end{bmatrix}$$

Q2: Dynamic Programming Problems (Detailed Breakdown)

Each problem below follows the two-step dynamic programming process:

1. Defining the subproblem
2. Finding a recurrence relation, including the base case and recursive step.

Q2.1: Highway Sign Placement

We need to place highway signs at designated mileposts while minimizing the penalty for spacing violations.

Step 1: Define the Subproblem

Let P(i) represent the minimum penalty for placing signs up to the i-th milepost.

- Given a list of mileposts $m_1,m_2,..., m_n$ (where $m_1 = 0$ and m_n is the last post), we must ensure signs are placed every 30 miles or fewer.
- The penalty for placing two consecutive signs x miles apart is $(30-x)^2$

Step 2: Recurrence Relation

We determine the best placement by choosing the previous sign at an optimal location j, minimizing the accumulated penalty.

Base Case:

$P(1) = 0$

(Since we must place a sign at $m_1 = 0$, there is no penalty.)

Recursive Step:

$P(i)=\min_{j<i}(P(j)+(30-(m_i-m_j))^2)$

- We iterate over all valid previous sign locations j, ensuring $m_i - m_j \leq 30$
- We minimize the total penalty accumulated from j to i.

Q2.2: Factory Location Planning

A company moves between two cities Brookings (B) and Chiloquin (C), incurring monthly operating costs and a fixed moving cost.

<u>Step 1: Define the Subproblem</u>

Define:

- $F_B(i)$ = Minimum cost of operating up to month i, ending in Brookings.
- $F_C(i)$ = Minimum cost of operating up to month i, ending in Chiloquin.

Given:

- Monthly costs:
  - B=(b1,b2,...,bn) (cost of being in Brookings each month)
  - C=(c1,c2,...,cn) (cost of being in Chiloquin each month)
- Moving cost M between the cities.

Step 2: Recurrence Relation

We consider two cases for each city:

1. Staying in the same city.
2. Moving to the other city (incurs a cost of M).

Base Case:

$F_B(1) = b1, F_C(1) = c1$

(Starting in either city has the direct cost of operating there for the first month.)

Recursive Step:

$F_B(i) = \min(F_B(i-1) + B[i], F_C(i-1) + M + B[i])$

$F_C(i) = \min(F_C(i-1) + C[i], F_B(i-1) + M + C[i])$

The cost of being in Brookings in month i comes from either staying in Brookings or moving from Chiloquin.

- The cost of being in Chiloquin in month i follows the same logic.

Q2.3: Maximum Coin Usage for Exact Change

We want to maximize the number of coins used to make exact change.

Step 1: Define the Subproblem

Let $M(Y)$ be the maximum number of coins that sum exactly to Y.

- Given a set of coin denominations $(z1, z2, \ldots, zn)$
- We can use unlimited copies of each coin.
- We need to find the maximum number of coins summing to Y (if possible).

Step 2: Recurrence Relation

Base Case:

$M(0)=0$ (Zero coins are needed to make value 0.)

$M(Y)=-\infty$, if $Y<\min(z1, \ldots, zn)$ (If Y is smaller than the smallest coin, it cannot be formed.)

Recursive Step:

$M(Y) = \max_{zsubi \leq Y} (1 + M(Y - z_i)$

Idk how to do a subscript of a subscript sorry 🙁

- We check all coins ziz_izi that do not exceed Y.
- We take the maximum value across all possibilities.