Daniel Willard

CIS 315

Prof. Lei Jiao

Written Homework 4

Problem 1) ∞

1. Demonstrate the Floyd-Warshall algorithm (i.e., show the series of matrices) on the graph described by the following weight matrix:

$$W = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & \infty & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & 3 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{pmatrix}$$

for $k = 1$ to n

  for $i = 1$ to n $\qquad O(n^3)$

    for $j = 1$ to n

      $d_{ij}^{(k)} = \min\{ d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \}$

$$W0 = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & \infty & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & 3 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{pmatrix}$$

$$W1 = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{pmatrix}$$

$$W2 = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{pmatrix}$$

$$W3 = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{pmatrix}$$

$$W4 = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 4 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$$

$$W5 = \begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 4 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$$

$$W6 = \begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ -5 & -3 & 0 & -1 & -6 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 4 & \infty \\ 3 & 4 & 10 & 7 & 2 & 0 \end{pmatrix}$$

$$Wfinal = \begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ -5 & -3 & 0 & -1 & -6 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 4 & \infty \\ 3 & 4 & 10 & 7 & 2 & 0 \end{pmatrix}$$

Problem 2)

We are given a weighted graph $G = (V, E)$ with weights given by W. The nodes represent cities and the edges are (positive) distances between the cities. We want to get a car that can travel between any two cities: it can leave with a full tank of gas, but cannot purchase gas until it reached its destination. The *zero-stop capacity* of the car we purchase is defined to be the longest distance between any two cities that we may travel (but of course, between any two cities we would take the shortest path). A simple way to compute this is to run Floyd-Warshall, and then look at all pairs of vertices/cities and return the greatest distance.

Formally, Floyd-Warshall returns W^*, where $W^*[u, v]$ is the length of the shortest route from city u to v. The zero-stop capacity is thus

$$\max\{\ W^*[u,v] \mid u, v \in V\ \}.$$

Now suppose that you want to calculate the necessary *one-stop capacity* of a potential car: we are allowed to stop at just **one** intermediate city to purchase gas for the car. Please determine what capacity would be needed for our car to travel between any two cities with at most one refueling stop. For this problem, just provide the mathematical representation(s) (and you can refer to the above example of the zero-stop capacity); do not write codes or pseudocodes, and writing codes or pseudocodes will not receive any credits. **[5 points]**

Note that the first two steps of the development of a dynamic programming algorithm for a problem are as follows: (1) describe the structure of the subproblem; (2) find a recurrence for the optimal value of the subproblem in terms of smaller subproblems. For the two problems below, perform these two steps. Do not write codes or pseudocodes—just the subproblem and recurrence structure, including initial conditions when needed; writing codes or pseudocodes will not receive any credits.

1) Subproblem: Let C[u,v,k] represent the minimum fuel to travel form city u to city v with one stop at city k.
2) Recurrence: C[u,v,k] = min{C[u,k,k-1], C[k,v,k-1]} + W[u,v]

Where W is the shortest route form city U to city V.

Initialized at C[u,v,0] = W[u,v] for all u,v in V cities in the graph.

That would be the gas capacity needed for the given the distance if no stop between u,v. or the direct weight form u to v.

We the can calculate C for increasing values of k form 1-n for the nods inn the graph. We can then calculate the minimum gas capacity for one stop. After C[u,v,n] is calculated.

Problem 3)

You work for the OR state highway agency and must place warning signs along a certain road. Along the way there are n locations at which you may place a sign, at mile posts $m_1 < m_2 < \cdots < m_n$, where each m_i is measured from the starting point $m_1 = 0$. The only

places you are allowed to place a sign is at one of the given mileposts. In addition, you must place one at locations m_1 and m_n.

Your requirement is to place one every 100 miles, but this may not be possible (depending on the spacing of the mileposts). If you place two consecutive signs x miles apart, the penalty for that placement is $(100-x)^2$. You want to arrange a placement so as to minimize the total penalty - that is, the sum, over all locations, of the penalties. Perform the two steps above to start the process of determining the minimum possible penalty. **[6 points]**

1) Subproblem: let P[i,j] represent the minimum possible penalty for placement of warning signs at milepost I to j.
2) Recurrence relation: P[I,j] = min{P[I,k] +P[k+1,j] +(100-(m_k – m_i))^2 for all j in range [i,j-1]

The initial condition would be as given P[i,j] =0 for all I in range 1 to n as there is no penalty for placing a single sign at one location.

By calculation P for the increasing values of j-I form 1 to n-1 where n is the number of mileposts, we will find the minimum possible penalty.

Problem 4)

There is a small real estate firm which in some months maintains an office in Coquille, OR (code C) and in others in Drain, OR (code D), and moves back and forth between these two cities (they can only afford to have one location operating at a time). This company wants to have the cheapest possible location plan - the two cities have different operating costs and these costs can change from month to month.

We are given M, a fixed cost of moving between the two cities, and lists $C = (c_1, \ldots, c_n)$ and $D = (d_1, \ldots, d_n)$. Here c_i is the cost of operating out of Coquille in month i, and d_i is the cost of being in Drain that month. Suppose that $M = 10$, $C = (1, 3, 20, 30)$, and $D = (50, 20, 2, 4)$. If the location plan is (C, C, D, D), its cost will be $1 + 3 + 10 + 2 + 4 = 20$. On the other hand, the cost of the plan (D, D, C, D) is $50 + 20 + 10 + 20 + 10 + 4 = 114$. The goal here is to (start to) devise a dynamic programming algorithm which, given M, C, and D, determines the *cost* of the optimal plan. The plan can start in either city, and end in either city. Note that you will likely need two subproblems, which will be mutually recursive. [**8 points**]

um I think it is like but I don't have two subcases that I could think of.

1) Subproblem: let O[i][j] be the cost of the optimal location plan for the first I months with the location in the month of `i` being j where j is either C or D.
2) Recurrence 1) O[i][C] = min{O[i-1][C] +c[i], O[i-1][D]+M+c[i]}
3) Recurrence 2) O[i][D] = min{ O[i-1][D] +d[i], O[i-1][C]+M+d[i]}

Initial Conditions:

Since c[0]=1 is less then d[0] = 50

We start at location c.

O[0][C] = c[0]

O[0][D] = d[0] + M

Switch recurrence case based off staring location.