CS561 HW12

Ryan Scherbarth

November 2024

1. **<u>Exercise 23.2-1 CLRS</u> Run the Floyd-Warshall algorithm on the weighted, directed graph of figure 23.2. Show the matrix $D^{(k)}$ that results for each iteration of the outer loop.**

We start with an adjacency matrix W s.t. $w(i,j)$ is the weight of the edge from i to j. If no edge exists, we set $W_{i,j} = \infty$, and the value $w(i,j) = 0$ where $i == j$.

Base case: $D^{(0)} = W$.
When we have $D^{(0)}$, there are no intermediate vertices allowed, so the lowest weight for position $W_{i,j}$ will be $W = w(i,j)$.

Recursive Step:

$$d^{(k)}(i,j) = \min(d^{(k-1)}(i,j), d^{(k-1)}(i,k) + d^{(k-1)}(k,j))$$

For each $k \in 1..n$.

We have solved the equation after finding $D^{(6)}$ given there are 6 vertices in the graph. Looking at $D^{(6)}$, we have the shortest path distances between all pairs of vertices in the graph.

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

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

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

$$D^{(3)} = \begin{bmatrix} 0 & 1 & 3 & -4 & 0 & 13 \\ \infty & 0 & 2 & \infty & -1 & 12 \\ \infty & \infty & 0 & \infty & 7 & 10 \\ 2 & 3 & 5 & 0 & 2 & 15 \\ \infty & \infty & \infty & \infty & 0 & 5 \\ \infty & \infty & -8 & \infty & \infty & 0 \end{bmatrix}$$

$$D^{(4)} = \begin{bmatrix} 0 & 1 & 3 & -4 & 0 & 13 \\ 2 & 0 & 2 & 2 & -1 & 17 \\ \infty & \infty & 0 & \infty & 7 & 10 \\ 2 & 3 & 5 & 0 & 2 & 15 \\ \infty & \infty & \infty & \infty & 0 & 5 \\ \infty & \infty & -8 & \infty & \infty & 0 \end{bmatrix}$$

$$D^{(5)} = \begin{bmatrix} 0 & 1 & 3 & -4 & 0 & 13 \\ 2 & 0 & 2 & 2 & -1 & 4 \\ \infty & \infty & 0 & \infty & 7 & 10 \\ 2 & 3 & 5 & 0 & 2 & 7 \\ \infty & \infty & \infty & \infty & 0 & 5 \\ \infty & \infty & -8 & \infty & \infty & 0 \end{bmatrix}$$

$$D^{(6)} = \begin{bmatrix} 0 & 1 & 3 & -4 & 0 & 13 \\ 2 & 0 & 2 & 2 & -1 & 4 \\ -6 & -5 & -8 & -10 & 7 & 3 \\ 2 & 3 & 5 & 0 & 2 & 7 \\ \infty & \infty & \infty & \infty & 0 & 5 \\ -14 & -13 & -16 & -18 & -15 & -5 \end{bmatrix}$$

2. **<u>Recursion Cat:</u> You are given a tree with all nodes colored either red or black. Call a path *valid* if at any step of the path, the number of red nodes visited so far is greater than or equal to the number of black nodes visited so far. A cat starts at the root node of the tree and wants to find a valid path to some leaf node.**

   For each node v, let $f(v)$ be $-\infty$ if there is no valid path to v. Otherwise, let $f(v)$ be the number fo red nodes visited minus the number of black nodes for the path ending at v.

   (a) **Give a recurrence relation for f. Hint: you may find it useful to let $p(v)$ be the parent of v, for every node v that is not the root.**

   Let $p(v)$ be the parent of v for every node v that is not the root. Let G denote the graph containing all nodes. Let $f(v)$ represent the number of red nodes subtracted from the number of black nodes that have been visited so far.

   Base case:

$$\begin{aligned} f(x) &= 1 && \text{if } x \text{ is red} \\ &= 0 && \text{Otherwise} \end{aligned}$$

   Recursive Step:

$$\begin{aligned} f(x) &= f(p(x)) + (x) && \text{if } f(p(x)) + (x) \geq 0 \\ &= -\infty && \text{Otherwise} \end{aligned}$$

Where color$(x) = 1$ if x is red, and -1 otherwise. We will then introduce the base cases when x is the root, and when the path from x DNE to get our final recurrence;

$$\begin{aligned} f(x) &= 1 && \text{if } x \text{ is the red root value} \\ &= 0 && \text{if } x \text{ is the black root value} \\ &= f(p(x)) + color(x) \\ &= \infty && \text{Otherwise} \end{aligned}$$

At each step, we ensure that $f(x)$; the difference between red and black nodes, is greater than 0, and then jump adding 1 each time we touch a red node. This helps us iterate through and make sure we never access more black nodes than red nodes at any point.

(b) **Briefly describe a dynamic program that uses the recurrence above to return a valid path from root to some leaf, if such a path exists.**

Define a 2D array DP with side lengths equal to the number of nodes in the graph, and let $DP_{i,j}$ denote the largest value of the function $f(x)$ as the difference in red and black nodes visited in some sequence to get to node (i, j).

We will initialize the array s.t. the root node $r = (i, j)$ is given as $DP_r = color(r)$, where $color(r)$ is 1 when r is red, and 0 otherwise. We will set the parent of the first root node to $-\infty$.

For each iteration, we will fill out the parent as

$$\begin{aligned} DP(x) &= DP(p(x)) + color(x) && \text{if } DP(x) + color(x) \geq 0 \\ &= -\infty && \text{Otherwise} \end{aligned}$$

After the table has been completed, any given item $DP_{i,j}$ with a value other than $-\infty$ means there is a possible, valid path from i to j. Starting at a value $k = (i, j)$, the path can be reconstructed by following the parent pointers back to the root r, prepend each new value to the front of the list so that we don't need to reverse at the end.

Now recursion cat wants to find valid paths on any graph. Define a *red cycle* **to be a cycle that has more red than black nodes in it. Assume you are given a graph, G, with no red cycles. For any pair of nodes, you wan tot determine if there is a valid path from u to v. Taking inspiration fro Floyd-Warshall, you first assign labels $1..n$ to all n nodes in the graph. Then you consider paths from nodes u to v that visit intermediate nodes with label at most i. For a given path, let the** *black excess* **of that path be the maximum over all steps of the path of the number of black nodes minus the number of red nodes at any step.**
For example, a path of the form R, B, R, B, B, B, R, R has a black excess of 2.
Define $f(u, v, i, b) = -\infty$ if there is no path from u to v using intermediate nodes of label at most i with black excess at most b. Otherwise, define $f(u, v, i, b)$ to be the maximum, over all paths from u to v, with black excess at most b that visit intermediate nodes with label at most i, of the number of red nodes minus the number of black nodes in that path.
For example, if the only path from u to v has form R, B, R, B, B, B, R, R, then $f(u, v, n, 2) = 0$.

(c) **Write a recurrence relation for $f(u, v, i, b)$. It may help to assume that $-\infty + x = -\infty$ for any value x. Hint: Let the base case(s) be $f(u, v, 0, b)$ for any values of u, v and any b s.t. $0 \leq b \leq n$. It may help to define a node v, $color(v)$ to be 1 if the node is red, and -1 if the node is black.**

Base case: $i = 0$
When $i = 0$, we don't allow any intermediate nodes, so the only possible solution will be if there is already a valid path;

$$\begin{aligned} f(u, v, 0, b) &= color(v) && \text{if } u == v \text{ and } b \geq 0 \\ &= -\infty && \text{Otherwise.} \end{aligned}$$

case 1: $f(u, u, 0, b) = color(u)$, since a path from a node to itself with no intermediate nodes is the value of the node itself.
case 2: $f(u, v, 0, b) = -\infty$ if $u \neq v$ since we don't allow any intermediate nodes, it is not possible to get from u to v.

Recursive Step: $i > 0$
We add $f(u, v, i-1, b)$ which finds the best valid path from u to v without using i as an intermediate node, carrying over results from the parent iteration.

$$f(u, v, i, b) = \max(f(u, v, i-1, b), \max_{b' \leq b}(f(u, i, i-1, b') + f(u, i, i-1, b - b')))$$

The first max compares whether we use i as an intermediate node or not. The second max, corresponding to the case where we do use i as an intermediate node, finds the maximum value of the black nodes subtracted from the red nodes while staying within our budget b.

b gives the total black excess allowed in a path $u \to i \to v$, and b' the portion of the excess used in a segment starting at u.

(d) **Briefly describe a dynamic program that uses the recurrence above to determine if a valid path exists from u to v for every u and v. What is the runtime as a function of n, the number of nodes, and m, the number of edges.**

Let DP be a 4D array where $DP[u][v][i][b] = 1$ means a valid path exists from u to v, visiting no more than i intermediate nodes, with a black excess of no more than b.

Base case: $i = 0$

$$\begin{aligned} DP[u][v][0][b] &= 1 && \text{if } u = v \text{ and } b \geq 0 \\ &= 0 && \text{Otherwise} \end{aligned}$$

A valid path exists from u to itself if $b \geq 0$, since the path has only u. No valid path can exist otherwise since we do not allow any intermediate nodes.

Recursive Step: $i > 0$

$$DP[u][v][i][b] = \max(DP[u][v][i-1][b], \max_{b' \leq b}(DP[u][i][i-1][b] + DP[u][i][i-1][b - b']))$$

We will iterate through each position in DP evaluating our recurrence relation above. Once complete, we know there exists a valid path from u to v if $DP[u][v][i][b] = 1$.

It will take $O(n^2)$ to check all pairs of nodes, (u, v). It will take $O(n)$ for each of these to check each possible intermediate node i. It will take another $O(n)$ for each of these to check each black excess value b, so we will have a total $O(n^4)$ runtime to build the table, and despite having a runtime of $O(1)$ to access the table after built, $O(n^4)$ will still bind our runtime.

3. **The Subgraph Isomorphism problem takes as input two undirected graphs, G_1 and G_2 and returns TRUE iff G_1 is isomorphic to a subgraph of G_2. Prove that the subgraph isomorphism problem is NP-complete.**

   A problem is NP-complete if it is NP-Hard and NP-Easy. Therefore, a problem must

   (a) it's "yes" answers can be verified in polynomial time

   (b) if solving one solution in polynomial time would guaranteed ability to solve every other valid solution in polynomial time.

   For (a), we need to check that f maps each vertex from G_1 to G_2 one-to-one, and also that for each edge (u, v) in G_1, there is an edge s.t. $(f(u), f(v))$ in G_2. We can verify both of these attributes in polynomial time, and conclude that the problem belongs to NP and is NP-Easy.

   We can use the Clique problem to prove (b). With graph G and some number x as inputs, we can create $G_1 = K_x$ as a complete graph on k vertices. $G_2 = G$ will be the input graph for the clique.

   If $G_1 = K_x$ is isomorphic, to some subgraph of G_2, then G_2 must contain a clique of size x. It will take $O(x^2)$ time to construct the graph, which verifies we are running in polynomial time and thus, we can conclude the problem is also NP-Hard.

   Since we have shown that the problem satisfies the properties of both NP-Easy and NP-Hard, we can then conclude that it is NP-Complete.

4. **In the MIN-INSIDE-EDGES problem, you are given a graph $G = (V, E)$, and a number $x \leq |V|$, and you must choose a subset $V' \subseteq V$ of size x. Call an edge E *inside* if both endpoints of the edge are nodes in V'. Your goal is to output the minimum number of inside edges for any set of V' of size x.**

   (a) **Show that MIN-INSIDE-EDGES is NP-Hard by a reduction from one of the following: 3-SAT, VERTEX-COVER, CLIQUE, SUBGRAPH-ISOMORPHISM, INDEPENDENT-SET, 3-COLORABLE, HAMILTONIACYCLE, or TSP.**

   We can prove that MIN-INSIDE-EDGES by reduction using the Clique problem. Let the input for the starting clique problem be a graph $G = (V, E)$ and an int k, where we need to find a subset $V' \subseteq V$ of size k s.t. every vertex pair in V' is connected by an edge.

   We start by solving the MIN-INSIDE-EDGES problem for G and $x = k$. A subset $V' \subseteq V$ is a clique of size k in G iff the number of inside edges in V is the maximum possible, which will be given by $\binom{k}{2} = \frac{k(k-1)}{2}$.

   When we solve the MIN-INSIDE-EDGES problem, we find the minimum number of inside edges for a subset of size $x = k$, through which we can also determine if a clique exists by checking on the last step. If taking the inverse of the solution to the MIN-INSIDE-EDGES problem gives the maximum, we know there exists a clique as well.

   Starting with an instance of clique, we start by transforming it into an instance of MIN-INSIDE-EDGES s.t. we have the same graph, G, and some $x = k$ as the size of the subset. solving MIN-INISIDE-EDGES will then be 0 when V' does not contain any clique of size k.

   Therefore, solving MIN-INSIDE-EDGES for G and $x = k$ can determine whether a clique size k exists in graph G based off of the number of inside edges. Since the clique problem is NP-Hard, and we have shown a polynomial-time reduction from clique to MIN-INSIDE-EDGES, we can conclude through reduction that MIN-INSIDE-EDGES is also NP-Hard.

(b) **Consider the randomized algorithm that pricks a subset V' of size x uniformly at random from all subsets of V of size x, when given graph $G = (V, E)$ and number x. Compute the expected number of inside edges for thsi algorithm using indicator random variables and linearity of expectation. Let $n = |V|$ and $m = |E|$.**

Given a graph $G = (V, E)$ with $n = |V|$ and $m = |E|$, $V' \subseteq V$ as a subset of size x chosen uniformly at random, Let $e = (u, v)$ be an inside edge iff $u, v \in V'$. Then, for each edge $e \in E$ can be defined by the indicator random variable;

$$\begin{aligned} I_e &= 1 && \text{when both endpoints are in } V' \\ &= 0 && \text{Otherwise} \end{aligned}$$

Given an indicator random variable I, we can calculate the expected number of inside edges as the summation of $1..n$ multiplied by the probability of i;

$$E[x] = \sum_{e \in E} I_e$$

We can calculate I_e for edge $e = (u, v)$ as the probability that both u and v are in V'. There are $\binom{n-2}{x-2}$ possibilities to choose the remaining $x - 2$ vertices, so the probability will be

$$\begin{aligned} I_e &= \frac{\binom{n-2}{x-2}}{\binom{n}{2}} \\ &= \frac{x(x-1)}{n(n-1)} \end{aligned}$$

So we can now solve the expected number of inside edges;

$$\begin{aligned} E[x] &= \sum_{e \in E} \frac{x(x-1)}{n(n-1)} \\ &= m\frac{x(x-1)}{n(n-1)} \end{aligned}$$

Therefore, the expected number of inside edges can be expressed in terms of the number of edges, m, the number of vertices, n, and the size of the subset V', n, as $E[x] = m\frac{x(x-1)}{n(n-1)}$.

(c) **Let μ be the expected number of inside edges for the randomized algorithm (i.e. your answer from part (b)). Now use Markov's inequality to bound the probability that there are greater than or equal to $\frac{11}{10}\mu$ inside edges after running the algorithm in part (b).**

Markov's inequality says;

$$\begin{aligned} P(X \geq a) &\leq \frac{E[X]}{a} && \text{for all } a > 0 \\ P(X \geq \frac{11}{10}\mu) &\leq \frac{E[X]}{\frac{11}{10}\mu} \\ &\leq \frac{\mu}{\frac{11}{10}\mu} \\ &\leq \frac{10}{11} \end{aligned}$$

The probability there are greater than or equal to $\frac{11}{10}\mu$ inside edges is bounded by $P(X \geq \frac{11}{10}\mu) \leq \frac{10}{11}$

(d) **Using your result from part (c), bound the expected number of times you would need to run the randomized algorithm before you get a solution that has less than $\frac{11}{10}\mu$ inside edges. Hint: Recall that for a random variable Y taking on positive integer values, $E(Y) = \sum_{i=1}^{\infty} p(Y \geq i)$ (see slides 32-33 from "Randomized Data Structures" lecture).**

If the expected value of Y; the expected number of runs, is given by $E[Y] = \sum_{i=1}^{\infty} P(Y \geq i)$. We know the probability is bounded s.t. $P(X < \frac{11}{10}\mu) \leq \frac{10}{11}$.

We can then derive the successful run probability as $1 - \frac{10}{11} = \frac{1}{11}$.

Since each run is independent, we have a geometric distribution Y with success probability given by $p = \frac{1}{11}$, the probability for the number of successive successes could be defined as $P(Y \geq i) = (\frac{10}{11})^{i-1}$.

The expected value of Y is then defined as $E[Y] = \frac{1}{1-\frac{10}{11}} = \frac{1}{\frac{1}{11}} = 11$. Therefore, the expected number of times the randomized algorithm needs to be run given each run is linearly independent with a uniquely random uniform distribution is $E[Y] = 11$.