CSCI 270 Homework #3 Solutions

1. Write your name, student ID Number, and which lecture you attend (morning or afternoon). Multi-page submissions must be stapled.

2. You are given a weighted directed acyclic graph $G = (V, E)$. You want to find the longest path in G.

   (a) Split the problem up into a list of ordered decisions. What is the first decision your algorithm must make?
   Solution:
   At each point, we must decide what is the next node we visit. Our first decision is figuring out what node to start at.

   (b) Define $OPT(x)$ to be the length of the longest path in G that starts at node x. Give the recursive procedure to solve this problem (don't forget the base case!)
   Solution:
   If x has no outgoing edges, then $OPT(x) = 0$.
   Otherwise, $OPT(x) = max_{(x,y)\in E}[c_{xy} + OPT(y)]$.
   If you consider the case where there can be negative edge weights, then $OPT(x) = max(0, max_{(x,y)\in E}[c_{xy} + OPT(y)])$

   (c) What order do you need to fill the array in to transform this into an iterative procedure?
   Solution:
   Fill the array in reverse topological order (start with a node with no outgoing edges, end with a node with no incoming edges).

   (d) Specify where the answer is stored in your array.
   Solution:
   Answer $= max_x OPT(x)$, where x is any node with no incoming edges. If you consider the case where there can be negative edge weights, you have to consider x to be any node.

   (e) Analyze the running time of your iterative procedure.
   Solution:
   A first glance tells us that there are n nodes, and each takes m time to fill. But we can improve this by noting we look at each edge once, so the actual runtime is $\theta(n + m)$.

   (f) Explain how to reconstruct the actual path, instead of just stating the length of the path.
   Solution:
   Starting at the node where our answer is stored ($max_x OPT(x)$), determine which edge was the next visited by checking all edges $(x, y) \in E$, and seeing if $OPT(x) = c_{xy} + OPT(y)$. Once we find the y we visit next, repeat until we find $OPT(z) = 0$. This is the longest path.

3. Given a ribbon strip of length k, a tailor will cut it once, in any place you choose, for the price of k dollars. Suppose you are given a strip of length L, marked to be cut in n different locations labeled $1, 2, ..., n$. For simplicity, let indices 0 and $n + 1$ denote the left and right endpoints of the original ribbon strip of length L. Let the distance of mark i from the left end of the ribbon be d_i , and assume that $0 = d_0 < d_1 < d_2 < ... < d_n < d_{n+1} = L$.

(a) Suppose you have a ribbon length 10, and you want to make two cuts at $d_0 = 6$ and $d_1 = 3$. What is the total cost if you make cut d_0 first? What is the total cost if you instead make cut d_1 first?
Solution:
If you cut d_0 first, then you pay $10+6 = 16$. If you cut d_1 first, then you pay $10+7 = 17$.

(b) Give an efficient dynamic programming algorithm to determine the minimum payment required to make the specified cuts, and analyze the runtime of your algorithm.
Solution:
Let $OPT(i, j)$ = the cost to make all cuts on the sub-ribbon starting at d_i and ending at d_j. Our next decision will be to decide where the next cut goes. We have to consider all possible choices k where $i < k < j$.
$OPT(i, i + 1) = 0$, which is our base case.
$OPT(i, j) = d_j - d_i + min_{k:i<k<j}[OPT(i, k) + OPT(k, j)]$. $d_j - d_i$ is the cost of our cut, and k is the possible choices where we can make that cut: we choose the one that produces the smallest cost.
We want to fill in the array from smallest $x = j - i$ to largest x. The order is:
FOR $x = 1$ to $n - 1$
FOR $i = 1$ to $n - x$
Calculate $OPT(i, i + x)$
The answer is stored at $OPT(0, n + 1)$.
The size of the array is n^2, and each element takes n time to fill, so the runtime is $\theta(n^3)$.

4. Suppose a smuggler wants to sneak n units of contraband into a country by hiding them on k trucks that are driving across the border that day. The ith truck can hold c_i pieces of contraband. Each truck has a known probability p_i of being searched at the border. You need to decide which trucks to store contraband on. Give an efficient dynamic programming algorithm that determines the minimum possible probability that any of those trucks are searched. Analyze the runtime of your algorithm.
Solution:
Let $OPT(i, x)$ = the optimal way to sneak x units of contraband, using trucks $\geq i$.
$OPT(i, x) = \max(OPT(i + 1, x), (1 - p_i) \cdot OPT(i + 1, x - c_i)$.
For the base case, $OPT(i, x) = 1$ if $x \leq 0$.

for $i = k$ to 1
for $x = 0$ to n
Calculate $OPT[i, x]$

Answer is stored at $OPT[1, n]$. Runtime = $\theta(nk)$.

5. The year is 20XX. After the downfall of mankind, the Earth is now ruled by robots (whom were all created by the genius Dr. Ight). TGO, or "Turing Game Online", is very popular amongst robots.

Robots love powers of 5, that is, the numbers 1, 5, 25, 125, 625, and so on. In TGO, the player is given a bit string in binary. The ideal situation is when the string it represents is a power of 5 in binary, with no leading zeros. If that is not the case, the robot player tries to cut the given string into pieces, each piece being a binary representation of a power of 5,

with no leading zeros. Of course, it may be the case that this is impossible. In that case, the robot goes into a murderous rampage. You, as one of the surviving humans, are in charge of checking the bit strings to prevent this from happening.

You are given a String S that consists of characters '0' and '1' only. S represents the string given to a player of the Turing game. Return the smallest positive integer K such that it is possible to cut S into K pieces, each of them being a power of 5. Return -1 if there is no integer K that satisfies the constraints. Analyze the runtime of your algorithm.

Solution:

Let $OPT(i)$ = the fewest number of powers of 5 you can split the substring $S_iS_{i+1}...S_n$ into, and $= -1$ if there is no such k.

Let $TGO(S) = 1$ if S is a power of 5, and $= 0$ otherwise.

We must decide what is the leading power of 5 we are going to use, as our decision.

$OPT(i) = 1 + \min_{j:OPT(j)\geq 0} OPT(j)$, over all j such that $j > i$ and $TGO(S_i...S_{j-1}) = 1$, and $= -1$ if there are no valid j.

$OPT(n+1) = 0$.

for $i = n$ to 1

Calculate $OPT[i]$

Answer is stored at $OPT[1]$.

The runtime is $\theta(n^2x)$, where x is the time complexity of TGO. If we assume TGO finishes in constant time, then the runtime is $\theta(n^2)$