Problem Set 4

This problem set is due at **10:00 am** on **Tuesday, March 7th**.

Problem 4- 1: Longest path

Given a directed acyclic graph (DAG) G give a dynamic programming algorithm to find the longest path in this graph. Prove that your algorithm is correct and analyze its runtime. (Hint: First topologically sort the DAG)

Problem 4- 2: OCD-2

Recall the problem 'OCD' from PS-2 where you had to store L gallons of oil in containers of capacities $1, 2, \ldots, 2^{1000}$. This time you need to store $L \geq 3$ gallons of oil and you are given access to a factory that can only make containers of capacities $1, 3, 4$. Once again, you would like to store the L gallons of oil in as few different containers as possible while ensuring that every container you store the oil in is full.

(a) Show that you cannot use the greedy algorithm from PS-2 to solve this problem. Give an L such that the greedy algorithm does not find the optimal number of containers.

(b) Give a dynamic programming algorithm to solve this problem. Prove that your algorithm is correct and analyze its runtime.

Solution to problem 1:

Recurrence relation:

$OPT(N) =$ number of edges in the longest path that ends in the N^{th} node of the topological sort
$OPT(1) = 0$
$OPT(N) = max_{(P,N) \in E}\{OPT(P) + 1\}$

Algorithm:

Result: Number of edges in the longest path in DAG G

TopologicalSort(G);

V $\leftarrow$ Nodes in G in topological order;
E $\leftarrow$ Edges in G;

MaxLength $\leftarrow$ array of maximum number of edges;
$\triangleright$ EX: MaxLength[i] = maximum number of edges in the path that ends in the i^{th} node

for *n = 1 ...* $|V|$ **do**
  MaxLength[n] = 0; $\triangleright$ Initialize MaxLength
end

for *n = 2 ...* $|V|$ **do**
  foreach *incoming edge* $\{u,v\}$ *of the* n^{th} *node* **do**
    p $\leftarrow$ topological order of u;
    MaxLength[n] = max(MaxLength[n], MaxLength[p] + 1);
end
return max(MaxLength);

Proof:

Claim: MaxLength[i] contains the number of edges in the longest path in that ends in the i^{th} node (when the i^{th} node refers to the topological ordering above)

Proof by Induction

Base Case: i = 1. MaxLength[1] = 0 because there are no incoming edges in the first node in a DAG.

Induction Hypothesis: Suppose the claim is true for all the nodes up to and including the i^{th} node.

Induction Step: for the $(i+1)^{th}$ node

The only way for a path to end at the $(i+1)^{th}$ node is if it came from a previous node in the topological order (by definition of topological ordering). Since it came from a previous node, the length of the longest path is (1 + the number of edges in the longest path to the connected previous node). One is added to account for the edge from the connected previous node to the $(i+1)^{th}$ node. By the induction hypothesis, MaxLength holds the correct value for every previous node. Since the path must have came from a previous node, the number of edges in the longest path that ends in the $(i+1)^{th}$ node is just the maximum of (1 + the number of edges in the longest path in the previous node) of all connected previous nodes. This is exactly That our algorithm calculates in the second for loop for MaxLength.

If there are no connected previous nodes, MaxLength should hold 0 for the $(i+1)^{th}$ node. This is covered in the initialization step in the first for loop.

Claim: The algorithm returns number of edges (weight) in the longest path in the given DAG.

Direct Proof: The longest path has to end in a node since the given graph is a DAG. MaxLength contains the number of edges (weight) in the longest path of that ends in each node. MaxLength is proved to be calculated correctly from the above proof, so the maximum value in MaxLength will be the number of edges in the overall longest path in the given DAG.

Runtime O($|V|$ + $|E|$) for topological sort, O($|V|$ + $|E|$) for looping through the nodes and the edges for the corresponding node.

Solution to problem 2:

Let $OPT(n)$ be the minimum sized set of containers such that the sum of the container sizes is n. Let C be the set of container sizes. Let $P(n)$ be the value of $c \in C$ such that $|OPT(n-c)|$ is minimized.

$$OPT(n) = \begin{cases} \emptyset & n = 0 \\ \{1\} & n = 1 \\ \{1,1\} & n = 2 \\ \{3\} & n = 3 \\ OPT(n - P(n)) \cup P(n) & n > 3 \end{cases}$$

The algorithm computes successive values of $OPT(n)$ until $n = L$.

Given $OPT(n')$ for $n' < n$, $P(n)$ can be computed in constant time, so $OPT(n)$ can be computed in constant time. L computations of $OPT(n)$ are required, so the total runtime of the algorithm is $O(L)$.

Theorem:
$OPT(n)$ is the minimum size set of containers that sums to n.

Proof:
Proof by strong induction on n.

Base case: $n \in [0, 3]$. The minimum set of containers to make 0 liters is the empty set. To make 1 liter, a size 1 container is needed. To make 2 liters, 2 size 1 containers are needed, since other containers are too large. To make 3 liters, a size 3 container is optimal.

Inductive hypothesis: suppose $OPT(n')$ is optimal for $n' < n$.

Inductive step: If $n > 3$, then the last case of the $OPT(n)$ definition applies.

First, we must prove that $OPT(n)$ is a valid set of containers (it sums to n). $OPT(n - P(n))$ sums to $n - P(n)$ liters, so adding a size $P(n)$ container sums to n liters, so $OPT(n)$ is valid.

Next, we must prove that $|OPT(n)|$ is minimal. Let C' be an optimal set of containers to fill n liters. Let c be the size of any container in C'. Consider $C' - \{c\}$. $C' - \{c\}$ must also be optimal, otherwise C' is not optimal. By the inductive hypothesis, $OPT(n - c)$ is optimal, so $|C' - \{c\}| = |C'| - 1 = |OPT(n - c)|$. $P(n)$ considers $OPT(n - c)$ in its minimization, so $|OPT(n - P(n))| \leq |OPT(n - c)|$. By the definition of $OPT(n)$, $|OPT(n)| = |OPT(n - P(n))| + 1$. Therfore $|OPT(n)| = |P(n)| + 1 \leq |OPT(n - c)| + 1 = |C'|$, so $OPT(n)$ is equally optimal as C'.