College of Computer & Information Science
Northeastern University
Fall 2017

CS5800
Algorithms
29 November 2017

Sample Solutions to Midterm II

Problem 1. (2 + 4 = 6 points) Depth-First Search

(a) Give an example of a directed graph $G = (V, E)$, two vertices $s, t \in V$, and a DFS run on the graph G such that:

1. G has four vertices, so $|V| = 4$.
2. For every vertex v in V, there exists a path in G from s to v.
3. In the DFS forest there is no path from s to t.

Answer: A simple example is a cycle with 4 vertices: $s \to u \to v \to t \to s$. Clearly, for every vertex v, s has a path to v. If the DFS run, if we start the DFS from u, we would obtain the DFS forest as the single tree $u \to v \to t \to s$, satisfying the last requirement.

(b) An *out-tree* of a directed graph G is a rooted tree in which there is a path from the root to every vertex in G. Design an efficient algorithm to determine whether a given directed graph has an out-tree. State the worst-case running time of the algorithm, in terms of n and m, the number of vertices and the number of edges, respectively, of G.

Answer: Here are two algorithms.

Algorithm 1:

1. Run DFS.
2. Let C be last DFS component, and let v be the vertex in C from where the DFS was initiated (i.e., with the lowest discovery time).
3. Run DFS from v to see if every vertex is reachable from v. If yes, then return "YES" else return "NO".

Algorithm 2:

1. Find SCCs and DAG D on SCCs.
2. Compute topological sort on D. Let v be first node of this topological sort.
3. Run DFS from v to see if every node in D is reachable from v. If yes, then return "YES" else return "NO".

Both the above algorithms take time $O(m + n)$.

A more inefficient algorithm is to run DFS from every vertex (separately) to check if any of them could possibly be a root of the desired out-tree.

Problem 2. (6 points) MSTs and shortest paths

For each of the following claims, indicate whether it is true or false. Justify your answers.

(a) Let G be an undirected connected graph with distinct positive weights on edges. Let T be a minimum spanning tree of G and let u be an arbitrary vertex of G. Let the edge (u, v) denote the edge whose weight is minimum among all edges adjacent to u. Then, (u, v) belongs to T.

Answer: True. Proof by contradiction. If (u, v) is not in T, then adding it would form a cycle C. This cycle C must contain another edge (u, x) adjacent to u. The weight of (u, x) exceeds that of (u, v). Removing (u, x) yields a new spanning tree of weight less than that of T, a contradiction.

(b) Let G be a directed graph with arbitrary weights (nonnegative or negative) on edges, and s and t be two vertices in G. Let p denote a shortest path from s to t. If we multiply the weight of every edge in the graph by 2, then p remains a shortest path from s to t.

Answer: True. Multiplying every edge weight by 2 doubles the weight of every path. Hence, for any two paths P_1 and P_2, if weight of P_1 was at most that of P_2 earlier, then the revised weight of P_1 will be at most the revised weight of P_2. So p remains a shortest path.

Problem 3. (6 points) Covering with intervals

You are given a set $A = \{a_1, a_2, \ldots, a_n\}$ of n integer points on the number line and a positive integer L, and are asked to determine the smallest set of closed intervals, each of length L, that contains all of the n points. The length of an interval $[x, y]$ is $y - x$.

Design an efficient algorithm for the above problem. Prove that your algorithm is correct. State the worst-case running time of your algorithm.

For example, consider the input instance with $n = 5$, $A = \{5, 2, -1, 7, 10\}$, and $L = 3$. One set of closed intervals of length 3 that covers A is $\{[-3, 0], [1, 4], [5, 8], [9, 12]\}$ since -1 is covered by $[-3, 0]$, 2 is covered by $[1, 4]$, 5 and 7 are covered by $[5, 8]$, and 10 is covered by $[9, 12]$. This set has size 4. A better, optimal, solution is the set $\{[-3, 0], [2, 5], [7, 10]\}$ with 3 intervals.

Answer: Here is a greedy algorithm.

1. Sort the a_is in nondecreasing order. Let $b_1 \le b_2 \le \ldots \le b_n$ be this list.
2. Set i to 1 and S to empty set.
3. Repeat until $i > n$:
   (a) Add interval $[b_i, b_i + L]$ to S.
   (b) Let $M = b_i + L$. Increment i until either $i > n$ or $b_i > M$.

Running time $= O(n \log n)$.

Proof of correctness: We will prove the greedy choice property and establish correctness by induction. We claim $[b_1, b_1 + L]$ is in an optimal solution. Consider an optimal solution S^*. If $[b_1, b_1 + L]$ is not in S^*, then there is an interval $[b, b + L]$ with $b < b_1$. Replacing this interval by $[b_1, b_1 + L]$ yields a new solution with the same number of intervals as S^* that also covers all points. This new solution is optimal. The greedy choice property is established.

For the remainder of the proof, we use induction on the number of points. If $n = 1$, our algorithm is optimal since it selects exactly one interval, as required. For the induction hypothesis, suppose the algorithm is optimal for $n < m$. Consider $n = m$. By the greedy choice property, the first interval selected, $[b_1, b_1 + L]$, is part of an optimal solution, say OPT_m. We know that $S' \setminus \{[b_1, b_1 + L]\}$ is an optimal solution for all points in A that exceed $b_1 + L$, since otherwise we can replace the remaining intervals in S' by fewer intervals from this purportedly better optimal solution for this subproblem. By our induction hypothesis, our algorithm computes an optimal solution for the remaining subproblem (with fewer than m points). Hence, the solution returned for m points is that of the same size as OPT_m.

Problem 4. Minimum weight vertex cover for a path

A path graph G is simply an undirected graph consisting of the vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and the edge set

$$E = \{(v_i, v_{i+1}) : 1 \leq i \leq n-1\}.$$

A *vertex cover* of G is a subset D of V such that for every edge (v_i, v_{i+1}), either v_i is in D or v_{i+1} is in D. (That is, a vertex cover covers every edge of the graph.) We also associate a positive weight w_i with vertex v_i. The weight of any subset S of vertices is simply the sum of the weights of the vertices in S.

For example, consider a path graph G with five vertices v_1, v_2, v_3, v_4, v_5, with weights 1, 9, 6, 3, and 7, respectively and four edges $(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_5)$. The set $\{v_1, v_3, v_4\}$ is a vertex cover with weight 10, while the set $\{v_1, v_4\}$ is not an vertex cover.

Give an efficient algorithm to determine a minimum weight vertex cover of a given weighted path graph G. State the worst-case running time of your algorithm. You need not prove the correctness of your algorithm.

Answer: We use dynamic progamming. Let $VCW[i]$ denote the weight of an optimal vertex cover for the subpath from v_i to v_n, and $VC[i]$ denote an optimal vertex cover for the subpath from v_i to v_n. We have the following recurrence for $VCW[i]$.

$VCW[n] = 0$.

For $1 \leq i < n$, $VCW[i] = \min\{w_i + VCW[i+1], w_{i+1} + VCW[i+2]\}$.

Here is the algorithm.

```
1. $VCW[n] = 0$;
2. For $i$ from $n-1$ down to 1:
    (a) if $w_i + VCW[i+1] < w_{i+1} + VCW[i+2]$, then $C[i] = i$; $VCW[i] = w_i + VCW[i+1]$
    (b) else $C[i] = i+1$; $VCW[i] = w_{i+1} + VCW[i+2]$
3. $VC = \emptyset$
4. While $i < n$:
    (a) Add $C[i]$ to $VC$ and set $i$ to $C[i] + 1$.
```

Running time is $O(n)$.

5. Decomposing an integer in terms of given integers

You are given an integer number $M \in \mathbb{N}$ and an array of n **distinct** integer numbers:

$$a_1, a_2, ..., a_n \quad (a_i \in \mathbb{N} \text{ for all } 1 \leq i \leq n).$$

such that $1 \in \{a_1, ..., a_n\}$.

Design an algorithm that finds the smallest number $k \in N$ such that there exists a sequence:

$$x_1, x_2, ..., x_k$$

that adds up to M, that is,

$$x_1 + x_2 + ... + x_k = M$$

and $x_i \in \{a_1, ..., a_n\}$ for all $1 \leq i \leq k$. Note that (i) the x_i's are not necessarily distinct, (ii) the ordering of the elements in the sequence is immaterial, and (iii) the main objective is to minimize the number of elements in the sequence.

For example, consider the instance with $M = 13$, $n = 3$, $a_1 = 2$, $a_2 = 1$, $a_3 = 5$. Then, one possible sequence is $2, 5, 2, 2, 2$ since these five numbers add up to 13. An optimal sequence is $5, 2, 5, 1$ with four elements.

Give a clear explanation of the algorithm and state the worst-case running time of your algorithm. You need not prove the correctness of your algorithm.

Answer: Let $OPT[j]$ denote the smallest sequence of numbers from the array that add up to j. We get the following recurrence.

$$OPT[j] = \min_{1 \leq i \leq n, a_i \leq j} OPT[j - a_i] + 1.$$

For the base case, we have $OPT[0] = 0$.

For the algorithm, we set the base case and have a for-loop for j from 1 to M. The minimization requires a for-loop, i going from 1 to n. Finally, we return $OPT[M]$.

Running time is $O(Mn)$.