Homework No. 4 (Answers)

Due Monday October 30, 2017

There are 100 points in total.

1. (15 points) Imagine you are planning a driving trip to Seattle in the coming fall break! Your car has a tank to hold $G$ gallons of gas when it is full. There are $n$ gas stations along the planned route. The distances between these gas stations $d_{i,i+1}$ are all known (Athens can be designated station 0 and Seattle station $n + 1$). Assume your car drives $m$ miles per gallon. You would like to come up with an algorithm that allows you to make as fewer stops for gas as possible. Assume you will fill up the tank in Athens before setting off to the road. Show that the problem has a greedy choice property and prove it. (You do not need to write the algorithm.)

**Answers:**

(1) (5 points) Statement of the greedy-choice property:

There is an optimal strategy to first stop at the last gas station reachable with the full tank of gas. That is, there is an optimal strategy $S$ such that the smallest $i \in S$ satisfies

$$d_{0,i} \leq mG \text{ and } d_{0,i+1} > mG$$

where $d_{0,i}$ is the distance from Athens to gas station $i$.

(2) (10 points) Proof of the greedy-choice property:

Generic format of a proof is proof-by-contradiction.

Proof:
Let $S$ be an optimal solution and $i$ be the smallest in $S$.

(a) If $i$ satisfies the condition $d_{0,i} \leq mG$ and $d_{0,i+1} > mG$, then the property is proved;
(b) Otherwise, assume smallest $j$, $j \in S$ but $j < i$. We construct another solution $S'$ such that

$$S' = S - \{j\} \cup \{i\}$$

$S'$ is a valid solution because the vehicle can reach gas station $i$ without stopping at gas station $j$.

In addition, $S'$ is also optimal because $|S'| = |S|$. So $i$ is contains in optimal solution $S'$ and $i$ is the smallest.
2. **(15 points)** Consider a slightly different version of the “coin change” problem. You are given $m$ types of coins with different denominations $d_1 = 1$, $d_1 < d_2 < \cdots < d_m$, with weights $w_1, w_2, \ldots, w_m$. Note that weights may not be proportional to values of coins. You want to make changes for a given $M$ cents of money so that the total weight of the coins is minimized.

Define an objective function and formulate a recursive solution that is suitable for a dynamic programming algorithm (you do not need to write the algorithm however).

**Answers:**

(1) **(5 points)** Definition of an objective function:

Define $Wgt(M)$ to be the minimum total weight of coins to change for $M$ cents.

(2) **(10 points)** Recursive formulas for the objective function:

\[
Wgt(M) = \min\begin{cases}
Wgt(M - d_1) + w_1 & M \geq d_1 \\
Wgt(M - d_2) + w_2 & M \geq d_2 \\
\vdots & \\
Wgt(M - d_m) + w_m & M \geq d_m
\end{cases}
\]

Base case: $Wgt(0) = 0$. 

3
3. (30 points) Consider the following problem to find the maximum number of left-right parentheses pairs. The input is a sequence of left and right parenthesis symbols, like

```
) ( ( ) ( )
1 2 3 4 5 6 7 8
```

(where the indexes are just for illustration purpose). There may be more than one way to pair left and right parentheses. For example, one pairing is \{(2, 4), (5, 8)\}, and another is \{(2, 6), (3, 4), (7, 8)\}. However, we are only permitted to identify pairs that are combinations of nested pairs, e.g., like \{(2, 8)\} and \{(3, 4)\}, and parallel pairs, e.g., \{(2, 4)\} and \{(5, 6), (7, 8)\}. Crossing pairs, like \{(2, 4)\} and \{(3, 6)\} are not simultaneously allowed. The goal of the problem is to find the maximum number of non-crossing left-right parenthesis pairs.

The problem resembles the matrix chain multiplication problem as both problem require us to identify a best way to group data items on the input sequence and groupings can be in either parallel or nested but not crossing fashion. In light of the solution to the matrix chain multiplication problem, the following dynamic programming method has been proposed.

Assume input to be \(x_1 \ldots x_n\). Then a subproblem deals with a segment of the sequence \(x_i \ldots x_j\), where \(i \leq j\). There are a few situations to consider when it comes to further break this subproblem into smaller subproblems: (a) to pair \(x_i\) and \(x_j\) but only if \(x_i = '(' \) and \(x_j = ')'\); (b) to abandon either \(x_i\) or \(x_j\); (c) to split the subsequence at the position of \(x_k\), for some \(k\), \(i \leq k < j\).

**This question requires you:**

1. Define an objective function for the optimal solution;
2. Formulate recursively the objective function (don’t forget base cases);
3. Design an iterative algorithm to fill out DP table;

**Answers:**

1. (5 points) Definition of an objective function:

Let the input sequence be \(S[1..n]\).
Define \(P(i, j)\) to be the maximum number of left-right parentheses on the segment \(S[i..j]\) for \(i \leq j\).

2. (5 points) Recursive formula for the objective function:
\[
P(i, j) = \max \begin{cases} 
P(i + 1, j - 1) + 1 & \text{if } S[i] = '(', S[j] = ')'
\end{cases}
\]

Base case: \( P(i, i) = 0, \) for all \( i = 1, \ldots, n \).

(3) (10 points) Pseudocode: iterative process for DP table filling

Criteria for the pseudocode to be correct:
(a) \( n \times n \) table for the objective function \( P \);
(b) order of the cells in the table to be filled; in the diagonal fashion;
(c) information about how each cell is computed should be recorded in the table (or in another table) as well.
4. **(20 points)** Show how the algorithm **Recursive DFS** (along with **To-Start-DFS**) in the Lecture Note 4 works on the graph figure shown below. Assume for the graph, the adjacency list of every vertex is ordered alphabetically. Assume that the for loop of lines 4-5 of algorithm **To-Start-DFS** considers the vertices in the alphabetical order. Show the resulted depth-first-search tree (or forest). **Please show snapshot of the DFS search tree generated by the algorithm at every step when a new vertex is discovered.**

![Graph Diagram]

**Answers:**

Note that the algorithm the student needs to follow is DFS given in the lecture note, which does not color vertices nor assign discovery and finish time stamps. If student uses the one that colors vertices and assign time stamps, deduct at least 5 points.

The answer should contain snapshots of every step when a new vertex is discovered.
5. **(20 points)** Consider the following *Constrained Reachability* problem: the input consists of a directed graph $G(V, E)$, two vertices $s, t \in V$, and a subset $U \subseteq V$, where $U \cap \{s, t\} = \emptyset$. The output is “yes” if and only if there is a path from $s$ to $t$ in the graph *without* using any vertex in $U$. Apparently, the problem is the commonly known *Reachability* problem when $U = \emptyset$.

(1) Design a recursive DFS type of algorithm to *Constrained Reachability* solve problem. You may assume that the input graph is already stored in adjacency lists and $v.\mu$ indicates if vertex $v$ is in the set $U$ or not. The information about $\mu$ is also given along with the graph.

(2) Argue that your algorithm still runs in linear time (i.e., $O(|E| + |V|)$-time) though the set $U$ may contain a large number of vertices.

**Answers:**

(1) **(10 points)** Design a recursive DFS type of algorithm to solve the problem:

An answer should simply to modify a DFS search algorithm, in the place where

$$\text{for } v \in Adj[u] \text{ and } v.visit = FALSE$$

recursive call applied on $v$ should be avoided if $v.\mu = TRUE$.

A answer may propose a slightly different solution, but as long as it follows the basic of DFS and avoid the search tree to include vertices from $U$.

(2) **(10 points)** Argue for the complexity:

The student needs to know why an original DFS runs in $O(|E| + |V|)$-time, otherwise he/she would have trouble explain why the modified algorithm runs in the same amount of time. It is important for the student to justify that the part to avoid $v \in U$ is not more than a constant time since $v.\mu$ is given already for every $v \in U$.z