Assignment 3 – Solutions

COMP3121/9101 22T1

Released March 17, due April 5

This document gives model solutions to the assignment problems. Note that alternative solutions may exist.

1. (20 points) You are given a 2 by n grid, where the cell on row i column j contains a non-negative number $a_{i,j}$. You can start at either cell in the leftmost column, and your goal is to reach either cell in the rightmost column by a sequence of moves. You can move to an adjacent cell (if it exists) in each of the 4 cardinal directions (up, down, left and right).

   A path achieves a score equal to the sum of values in its cells. Note that a cell which is used twice in a path only counts its value once to the score of that path.

   Design an algorithm which runs in $O(n)$ time and finds a path of minimum score from the leftmost column to the rightmost column.

Solution

Subproblems

Let $P(i, j)$ be the problem of determining $\text{opt}(i, j)$, the smallest sum of a path from the leftmost column to the cell in row i and column j, and $\text{pred}(i, j)$, the row in which this path enters column j.

Recurrence

For $j > 1$, we have

$$\text{opt}(i, j) = \min\left(\text{opt}(3 - i, j - 1) + a_{3-i,j} + a_{i,j}, \text{opt}(i, j - 1) + a_{i,j}\right),$$

with $\text{pred}(i, j) = 3 - i$ in the first case and $\text{pred}(i, j) = i$ in the second case. This accounts for the two ways to get from column $j - 1$ to column j:

- from the other row (row $3 - i$), step right and then up or down
- from the same row, step right.

We will never want to step left, as a path including a leftwards step must contain a cycle, and this is clearly not optimal. Note that this observation would not be true if there were more than two rows.

Base cases

In the first column, the paths are trivial, so $\text{opt}(1, 1) = a_{1,1}$ and $\text{opt}(2, 1) = a_{2,1}$. In both cases, $\text{pred}(i, 1) = i$.

Overall solution

The solution to $P(i, j)$ depends on the solutions to $P(i, j - 1)$ and $P(3 - i, j - 1)$, so we fill the grid from left to right. We begin with the base cases $P(\cdot, 1)$, then solve for both values $P(\cdot, 2)$, then $P(\cdot, 3)$ and so on.

The overall solution is the better of the two ways to reach the rightmost column, i.e.

$$\min(P(1, n), P(2, n)).$$

We can then reconstruct the path by backtracking through the pred array.

Time complexity

Each of $O(n)$ subproblems is solved in constant time, and the overall solution is found in constant time. Therefore the time complexity is $O(n)$.

2. (20 points) There are m towns in a straight line, with a road joining each pair of consecutive towns. Legends say that an ordinary person in one of these towns will become a hero by completing a sequence of n quests. The first quest will be completed in their home town, but after each quest they may complete their next quest either in the same town or after moving to a neighbouring town.

   For example, if $n = 5$ and $m = 4$, a resident of town 2 may become a hero as follows:

   - begin in town 2 for quest 1,
   - then move to town 3 for quest 2,
   - stay in town 3 for quest 3,
   - return to town 2 for quest 4, and
   - move to town 1 for quest 5.

   Design an algorithm which runs in $O(nm)$ time and finds the total number of ways to complete n quests.

Solution

If $m = 1$, the answer is clearly 1. Henceforth we assume $m > 1$.

Subproblems

Let $P(i, j)$ be the problem of determining $\text{num}(i, j)$, the number of ways to complete the first i quests finishing in town j.

Recurrence

For all $i > 1$ and all j, we have

$$\text{num}(i,j) = \begin{cases} \text{num}(i-1,j) + \text{num}(i-1,j+1) & \text{if } j = 1 \\ \text{num}(i-1,j-1) + \text{num}(i-1,j) & \text{if } j = n \\ \text{num}(i-1,j-1) + \text{num}(i-1,j) + \text{num}(i-1,j+1) & \text{otherwise.} \end{cases}$$

To complete the ith quest in town j, the hero must have completed their previous quest at town j or a neighbouring town. We therefore compute the sum of all ways to complete the first $j - 1$ quests ending at towns $i - 1$, i or $i + 1$ (if they exist).

Base cases

Clearly $\text{num}(1, j) = 1$ for all j, as there is only one way to complete the first quest in any particular town.

Overall solution

The solution to $P(i, j)$ depends only on the solutions to $P(i-1, \cdot)$, so we fill in the grid by rows from top to bottom. First $P(1, \cdot)$ are the base cases, then we solve all $P(2, \cdot)$, then all $P(3, \cdot)$, and so on.

The hero's path may finish at any of the towns, so the overall solution is

$$\sum_{j=1}^{m} \text{num}(n, j).$$

Time complexity

Each of $O(nm)$ subproblems is solved in constant time, and the overall solution is computed in $O(m)$, so the total time complexity is $O(nm)$.

3. (20 points) There are n different sizes of boxes, from 1 to n. There is an unlimited supply of boxes of each size t, each with a value v_t. A box of size t can hold several smaller boxes of sizes $a_1, a_2, \ldots, a_k$ as long as the sum of sizes $a_1 + a_2 + \ldots + a_k$ is *strictly less than* t. Each of these boxes may be filled with yet more boxes, and so on.

   Design an algorithm which runs in $O(n^2)$ time and finds the maximum value that can be attained by taking one box, potentially with smaller boxes nested inside it.

Solution

Subproblems

Let $P(i)$ be the problem of determining $\text{opt}(i)$, the maximum value that can be stored in *the space occupied by* a box of size i, which we will call i *units of space*.

Recurrence

For all $i > 1$, we have

$$\text{opt}(i) = \max\left(v_i + \text{opt}(i-1), \max_{1 \le j \le \frac{i}{2}} [\text{opt}(j) + \text{opt}(i-j)]\right).$$

The best way to fill i units of space is to either take a box of size i, with value v_i, and fill its interior in the best possible way, or to subdivide the i units of space into some j and $i - j$, each of which could be made up of one or more boxes.

Base cases

For $i = 1$, the only way to fill one unit of space is with a single box of size 1, so we have $\text{opt}(1) = v_1$.

Overall solution

Claim: The best solution must take the largest box.

Proof: Suppose the best solution does not take the largest box. Then putting it inside the largest box yields an even greater value, which is a contradiction. Therefore, we

take a box of size n and fill its interior in the best possible way, to get an answer of $v_n + \text{opt}(n-1)$.

Time complexity

Each of n subproblems is solved in $O(n)$ time. The overall solution is found in $O(1)$. Therefore the total time complexity is $O(n^2)$.

4. (20 points) You are given a tree T with n vertices, rooted at vertex 1. Each vertex i has an associated value a_i, which may be negative. You wish to colour each vertex either red or black. However, you must ensure that for each pair of red vertices, the path between them in T consists only of red vertices.

   Design an algorithm which runs in $O(n)$ time and finds the maximum possible sum of values of red vertices, satisfying the constraint above.

Solution

Subproblems

Let $P(i)$ be the problem of determining $\text{opt}(i)$, the maximum sum of red vertices within the subtree rooted at vertex i, with vertex i coloured red.

Recurrence

Letting C_i be the set of children of vertex i, we have

$$\text{opt}(i) = v_i + \sum_{j \in C_i} \max(\text{opt}(j), 0).$$

For each child j, it may be coloured either red or black:

- if vertex j is coloured red, we use the optimal solution $\text{opt}(j)$ for its subtree;
- otherwise, its subtree must all be black, contributing zero to $\text{opt}(i)$.

We take the better of these two options.

Base cases

If vertex i is a leaf, then the only vertex in the subtree is vertex i itself, so $\text{opt}(i) = v_i$.

Overall solution

Note that the solution to $P(i)$ depends on the solution to $P(j)$ for all children j of i. Therefore we first perform a depth-first search from the root, recording the order in which we see the vertices, and then iterate through the subproblems in the reverse of this order. This ensures that each child is solved before its parent, as required.

The overall solution is

$$\max\left(\max_{1 \le i \le n} \text{opt}(i), 0\right).$$

This simply decides the best vertex to be the root of the red subtree, and colours its descendants optimally. We also account for the special case where all $\text{opt}(i)$ values are negative (i.e. when all v_i are negative), where it is better to not colour any vertices red.

Time complexity

The initial DFS takes $O(n)$ time. Each vertex is the child of at most one other vertex, so the total time taken to solve all n subproblems is $O(n)$. Calculating the final answer also takes $O(n)$ time, so the total time complexity is $O(n)$.