Assignment 2 Solutions

COMP3121/9101 21T3

Released September 29, due October 13

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

1. You are given a triangular grid of non-negative integers. The grid consists of n rows, the ith of which has i many entries. For $1 \leq j \leq i \leq n$, the jth entry in row i is denoted $T(i, j)$.

   Define a *route* to be any path that starts at the top entry and ends at any entry of the bottom row, with each step going either diagonally down to the left or diagonally down to the right. Your task is to find the largest sum of numbers that can be encountered on a route.

   (a) (6 points) Consider the following greedy algorithm which attempts to construct an optimal route.

   > Start at the top entry. When you reach an entry $T(i, j)$, there will be two entries immediately below it; $T(i+1, j)$ to the left and $T(i+1, j+1)$ to the right. Step down to row $i + 1$ in the direction of the larger of these two values.

   Construct an example for which this algorithm does *not* produce the correct answer. You must include the triangle of numbers, the answer produced by this algorithm and the correct answer.

   (b) (14 points) Design a dynamic programming algorithm which solves this problem and runs in $O(n^2)$ time.

   (a) Here is an example for which this algorithm is incorrect:

   $$\begin{array}{ccccc} & & 1 & & \\ & 3 & & 2 & \\ 4 & & 3 & & 100 \end{array}$$

   The algorithm will greedily follow $1 \to 3 \to 4$, producing the answer 8. The optimal path is $1 \to 2 \to 100$ which gives 103.

   (b) **Subproblems:** for $1 \leq j \leq i \leq n$, let $P(i, j)$ be the problem of determining $\text{opt}(i, j)$, the maximum sum of a route starting from entry $T(i, j)$.

   Recurrence: if $i < n$, we have

   $$\text{opt}(i, j) = T(i, j) + \max\left[\text{opt}(i + 1, j), \text{opt}(i + 1, j + 1)\right],$$

since the best route starting at (i, j) consists of the initial entry, then follows the best route starting from either the below-left entry or below-right entry.

Base cases: for $1 \leq j \leq n$, $\text{opt}(n, j) = T(n, j)$ since there are no entries below row n.

Since the solution to $P(i, j)$ depends on the solutions to $P(i + 1, j)$ and $P(i + 1, j + 1)$, we solve subproblems by decreasing order of i and any order of j.

Since all routes start at the top entry, the final answer is $\text{opt}(1, 1)$.

There are $O(n^2)$ subproblems, each solved in constant time, so the overall complexity is $O(n^2)$.

2. (20 points) You are given a non-negative integer m and an array A of length n, where each element $A[i]$ is a positive integer. Your task is to find the maximum sum of a subset S whose sum does not exceed m.

   Design a dynamic programming algorithm which solves this problem and runs in $O(mn)$ time.

   Subproblems: for $0 \leq i \leq n$ and $0 \leq j \leq m$, let $P(i, j)$ be the problem of determining $f(i, j)$, which is true if the subarray $A[1..i]$ has a subset that sums to j and false otherwise.

   Recurrence: for $i > 0$, we have

$$f(i,j) = \begin{cases} f(i-1,j) & \text{if } j < A[i] \\ f(i-1,j) \text{ OR } f(i-1,j-a_i) & \text{if } j \geq A[i]. \end{cases}$$

   Clearly, if $A[1..i-1]$ has a subset with sum j, then this same subset suffices. Alternatively, if $A[i] \leq j$, then we should also consider including $A[i]$ to a subset of $A[i..i-1]$ with sum $j - A[i]$.

   Base cases:

$$f(0,j) = \begin{cases} \text{true} & \text{if } j = 0 \\ \text{false} & \text{if } j > 0, \end{cases}$$

   since an empty array can only make a sum of zero.

   Since the solution to $P(i, j)$ depends on the solutions to $P(i-1, \cdot)$, we solve subproblems by increasing order of i and any order of j. To retrieve the maximum sum using entries from the entire array A, we check $f(n, 0), f(n, 1), \ldots, f(n, m)$ and find the largest index j where $f(n, j)$ is true.

   There are $O(nm)$ subproblems, each solved in constant time, so the overall complexity is $O(nm)$.

3. (20 points) You are given a positive integer n and a decimal digit k. Your task is to count the number of n-digit numbers (without leading zeros) in which the digit k appears an even number of times. Note that 0 is an even number.

   Design a dynamic programming algorithm which solves this problem and runs in $O(n)$ time.

   Note: The following solution is based on the interpretation that $0, 1, \ldots, 9$ are all 1-digit numbers. An alternative interpretation excludes 0 as a 1-digit number. Either interpretation is acceptable.

   All numbers with $i > 0$ digits consist of a nonzero leading digit, then $i-1$ more digits that can be zero. We'll count how many numbers have an even number of digits k and how many have an odd number of digits k. We first deal with the case $k \neq 0$, and then we will discuss the modifications required when $k = 0$.

   Subproblems: for $1 \leq i \leq n$, let $P(i)$ be the problem of determining $\text{even}(i)$, the quantity of numbers with i decimal digits where k appears an even number of times, and $\text{odd}(i)$, the quantity of numbers with i decimal digits where k appears an odd number of times.

   Recurrence: for $1 < i < n$, we have

$$\begin{aligned} \text{even}(i) &= 9 \times \text{even}(i-1) + \text{odd}(i-1) \\ \text{odd}(i) &= 9 \times \text{odd}(i-1) + \text{even}(i-1). \end{aligned}$$

   If the leading digit is not k, then the parity of digits k in the remaining $i-1$ digits should be maintained. If instead the leading digit is k, then the parity is reversed. There are ten choices for the leading digit, namely k and the nine other digits. At the last step however, 0 is forbidden as the leading digit, so we have

$$\begin{aligned} \text{even}(n) &= 8 \times \text{even}(n-1) + \text{odd}(n-1) \\ \text{odd}(n) &= 8 \times \text{odd}(n-1) + \text{even}(n-1). \end{aligned}$$

   Base cases: we have

$$\text{even}(1) = 9 \text{ and } \text{odd}(1) = 1,$$

   since there is only one 1-digit number with an odd number of instances of digit k (namely k itself), and the remaining nine have no instances of k.

   The solution to $P(i)$ depends on the solution to $P(i-1)$, so we solve subproblems in increasing order of i.

   The final answer is simply $\text{even}(n)$, the quantity of n-digit numbers without leading zeros where digit k appears an even number of times.

   There are n subproblems, each solved in constant time, so the overall time complexity is $O(n)$.

   Special case: if $k = 0$, the final step of the recurrence is

$$\begin{aligned} \text{even}(i) &= 9 \times \text{even}(i-1) \\ \text{odd}(i) &= 9 \times \text{odd}(i-1) \end{aligned}$$

   since the leading digit must be nonzero.

4. (20 points) You are given a non-negative integer m and an array A of length n, where each element $A[i]$ is a non-negative integer less than 2^m. Your task is to find a subarray B of maximum length such that $B[i] \,\&\, B[i+1] \neq 0$ for every $1 \leq i < n$, where & denotes bitwise AND.

   Design a dynamic programming algorithm which solves this problem and runs in $O(mn)$ time.

   Note first that each $A[i]$ can be written as an m-bit number, potentially with leading zeros.

   Subproblems: For $1 \leq i \leq n$, let $P(i)$ be the problem of determining $\text{opt}(i)$, the length of the longest valid subsequence of $A[1..i]$ ending at $A[i]$ and $\text{pred}(i)$, the index in A of the penultimate entry in such a subsequence. Also, for $1 \leq i \leq n$ and $1 \leq j \leq m$, let $Q(i,j)$ be the problem of determining $f(i,j)$, the length of the longest valid subsequence of $A[1..i]$ where the jth bit[1] of the last selected element is 1, and $g(i,j)$, the index in A of the last entry in such a subsequence.

   Recurrence: For $i > 1$, let

$$j^* = \operatorname*{argmax}_j \{f(i-1,j) \mid \text{the } j\text{th bit of } A[i] \text{ is } 1\}.$$

   If $A[i]$ is the last entry of a valid subsequence, it must have a bit in common with the penultimate entry of the subsequence. There is no other constraint, so we pick the bit j^* which allows $A[i]$ to extend a valid subsequence of maximum length. Then

$$\text{opt}(i) = f(i-1, j^*) + 1$$

   and

$$\text{pred}(i) = g(i-1, j^*).$$

   Note that if $A[i]$ is zero, then j^* is undefined, so we set $\text{opt}(i) = 1$ and $\text{pred}(i)$ is undefined.

   Also, for $i > 1$ and all $1 \leq j \leq m$,

$$f(i,j) = \begin{cases} \text{opt}(i) & \text{if the } j\text{th bit of } A[i] \text{ is } 1 \\ f(i-1,j) & \text{if the } j\text{th bit of } A[i] \text{ is } 0 \end{cases}$$

   with $g(i,j) = i$ in the first case and $g(i,j) = g(i-1,j)$ in the second case. Clearly, if $A[i]$ has a 0 in bit j, then we should refer to the solution of $Q(i,j)$. However, if $A[i]$ has a 1 in bit j, it is in fact the best end index available. In this case, any solution to $Q(i,j)$ ending at $A[k]$ where $k < i$ is suboptimal, since appending $A[i]$ yields a subsequence which is:

   - longer by one,
   - valid, since $A[k]$ and $A[i]$ both have 1 in the jth bit, so $A[k] \,\&\, A[i]$ is nonzero, and
   - also has a 1 in the jth bit of its last entry.

[1] "jth bit" means jth least significant bit throughout this solution.

Therefore we simply take the longest subsequence ending at $A[i]$, which is solved in $P(i)$.

Base cases: We have $\text{opt}(1) = 1$ and $\text{pred}(1)$ undefined, since the first entry alone is a valid subsequence with no penultimate element. For $1 \leq j \leq m$,

$$f(1, j) = \begin{cases} 1 & \text{if the } j\text{th bit of } A[1] \text{ is 1} \\ 0 & \text{if the } j\text{th bit of } A[1] \text{ is 0} \end{cases}$$

with $g(i, j) = 1$ in the first case and undefined otherwise, since the only candidate subsequences are either the sole element $A[1]$ or the empty subsequence.

Since the solution to $P(i)$ depends on the solutions to $Q(i-1, \cdot)$ *and* the solution to $Q(i, j)$ depends on the solutions to $P(i)$ and $Q(i-1, j)$, we solve subproblems by increasing order of i, at each stage solving $P(i)$ and then all $Q(i, j)$ in any order of j. The overall longest valid subsequence must end at some entry $A[i]$, so its length is given by

$$\max_{1 \leq i \leq n} \text{opt}(i).$$

The index of the last entry in such a subsequence is

$$i^* = \operatorname*{argmax}_{i : 1 \leq i \leq n} \text{opt}(i),$$

and the rest of the subsequence is found by backtracking; the second last entry is $\text{pred}(i^*)$, the third last is $\text{pred}(\text{pred}(i^*))$, and so on.

There are n subproblems $P(i)$, each solved in $O(m)$ time, and nm subproblems $Q(i, j)$, each solved in constant time, so the overall complexity is $O(nm)$.

5. (20 points) You are given a directed graph $G = (V, E)$, where each edge e has an associated weight $0 < w_e < 1$. You may assume that there is at least one path from every vertex u to every other vertex v, i.e. G has only one strongly connected component. Define the *safety* of a path consisting of edges $e_1, e_2, \ldots, e_k$ as $\prod_{i=1}^{k} w_{e_i}$. Your task is to find for each ordered pair of vertices (u, v) the maximum safety of a path from u to v.

   Design a dynamic programming algorithm which solves this problem and runs in $O(n^3)$ time.

   Define a new graph $G' = (V, E)$, where each edge e has the modified weight

$$w'_e = -\log w_e.$$

   Then for a path consisting of edges $e_1, e_2, \ldots, e_k$, the total weight in G' is

$$\sum_{i=1}^{k} (-\log w_{e_i}),$$

   which can be rewritten as

$$-\log \prod_{i=1}^{k} w_{e_i},$$

   the negative log of the safety of the corresponding path in G. Now $f(x) = -\log x$ is a decreasing function, so the shortest u–v path in G' corresponds to the path of *maximum* safety in G. Furthermore, each edge weight in G' is positive, so there are no negative cycles and therefore we can use the Floyd-Warshall algorithm to find the length of the shortest path in G' from every vertex u to every other vertex v.

   Subproblems: for $1 \le i, j \le n$ and $0 \le k \le n$, let $P(i, j, k)$ be the problem of determining $\text{opt}(i, j, k)$, the shortest path in G' from v_i to v_j using only intermediate vertices from $\{v_1, \ldots, v_k\}$.

   Recurrence: for all $1 \le i, j, k \le n$,

$$\text{opt}(i, j, k) = \min(\text{opt}(i, j, k-1), \text{opt}(i, k, k-1) + \text{opt}(k, j, k-1)).$$

   Base cases: for all $1 \le i, j \le n$, we have

$$\text{opt}(i, j, 0) = \begin{cases} 0 & \text{if } i = j \\ w'_e \,(= -\log w_e) & \text{if } (v_i, v_j) = e \in E \\ \infty & \text{otherwise.} \end{cases}.$$

   Since the solution to $P(i, j, k)$ depends on the solutions to $P(\cdot, \cdot, k-1)$, we solve the subproblems in increasing order of k and any order of i and j.

   To find the final answer, observe that for each pair $1 \le i, j \le n$, the shortest path from v_i to v_j in G' has length $\text{opt}(i, j, n)$ and therefore the maximum safety of a path from v_i to v_j in G is $\exp[-\text{opt}(i, j, n)]$.

   There are $O(n^3)$ subproblems, each solved in constant time, so the overall complexity is $O(n^3)$.