Assignment 3 - Hints and Clarifications

COMP3121/9101 21T3

Released October 13, due November 3

This document provides some hints to help you solve the problems in Assignment 3. You are *not* required to follow these hints, and there may be alternate solutions which are equally correct.

Also included are the clarifications listed in the Assignment 3 FAQ on the Ed forum. Further clarifications may be added after this document is released.

General clarifications:

- **How should a dynamic programming algorithm be formatted?**

  You should clearly identify the subproblems, recurrence, base cases, final answer calculation and time complexity. These can be written using subheadings as in the lecture slides, or in plain text. See this comment for more.

- **How do I justify the correctness of a dynamic programming algorithm?**

  In a dynamic programming algorithm, the correctness may be in question in several places.

  - Answers to base cases

    Usually these are trivial, and can be justified with at most a sentence of explanation.

  - Answers to other subproblems, achieved using the recurrence

    This is usually the least obvious part. Again, a worded explanation is sufficient; consider how the optimal solution to a large subproblem can be related to solutions of smaller subproblems, and look for a 'cut-and-paste' type argument for why we can use the (already computed) optimal solution of each smaller subproblem. Note that this makes use of the *optimal substructure* property.

  - Calculating the overall answer

    In some problems this is also trivial, and may require only a brief explanation. Make sure to state the order in which subproblems should be solved, and if necessary justify that this order satisfies any dependencies which arise from the recurrence. This is particularly important when the subproblem specification consists of two or more parameters.

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.

   Clarifications:

   - **What is the format of the grid?**
     As an example, when $n = 4$ the grid is of the form

$$\begin{array}{ccccccc} & & & T(1,1) & & & \\ & & T(2,1) & & T(2,2) & & \\ & T(3,1) & & T(3,2) & & T(3,3) & \\ T(4,1) & & T(4,2) & & T(4,3) & & T(4,4) \end{array}$$

     where all $T(i, j) \geq 0$.
   - **What is the output?**
     Only the largest sum of numbers on a route is necessary. You do not need to find a route which achieves this sum.
   - **Do the entries have to be in sorted order within a row?**
     No.
   - **Can a diagram be used to answer (a)?**
     Yes, along with a brief explanation of any information needed to interpret the diagram.

   Hint: For part (a), there is a counterexample with $n = 3$ and all grid entries equal to 0, 1 or 2.

   Hint: For path (b), observe that a path from the top down to $T(i, j)$ must take some entry from row $i - 1$. There are at most two options for which entry to take.

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.

   Clarifications:

   - **Can S include several elements of array A with the same value?**
     Yes.
   - **Do the elements of S have to be contiguous in A?**
     No.

   Hint: Refer to the 'Balanced Partition' problem from lectures.

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.

   Clarifications:

   - **What is the input?**
     The input consists of n and k only, where n is a positive integer and $k \in \{0, \ldots, 9\}$.
   - **Does an n-digit number with no instances of digit k contribute to the total being counted?**
     Yes.

   Hint: If the first instance of digit k occurs in the ith digit, we want to count the number of $(n-i)$-digit numbers where k appears an *odd* number of times.

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.

   Clarifications:

   - **Do the elements of B have to be contiguous in A?**
     No.
   - **Do the elements of B have to be in the same order that they appeared in A?**
     Yes.

   Hint: $B[i] \,\&\, B[i+1] \neq 0$ if these two integers have an 'on' bit in common. Each entry of A can be written as an m-bit number.

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.

   Clarifications:

   - **What is the variable n referred to in the time complexity?**
     n is the number of vertices, i.e. $|V|$.

   Hint: This is closely related to the all pairs shortest path problem, except that we are multiplying edge weights rather than adding them.