Week 5 Tutorial Sheet

(To be completed during the Week 5 tutorial class)

Objectives: The tutorials, in general, give practice in problem solving, in analysis of algorithms and data structures, and in mathematics and logic useful in the above.

Instructions to the class: Aim to attempt these questions before the tutorial! It will probably not be possible to cover all questions unless the class has prepared them in advance. There are marks allocated towards active participation during the class. You **must** attempt the problems under **Assessed Preparation** section **before** your tutorial class and give your worked out solutions to your tutor at the start of the class – this is a hurdle and failing to attempt these problems before your tutorial will result in 0 mark for that class even if you actively participate in the class.

Instructions to Tutors:

1. The purpose of the tutorials is not to solve the practical exercises!
2. The purpose is to check answers, and to discuss particular sticking points, not to simply make answers available.

Supplementary problems: The supplementary problems provide additional practice for you to complete after your tutorial class, or as pre-exam revision. Problems that are marked as **(Advanced)** difficulty are beyond the difficulty that you would be expected to complete in the exam, but are nonetheless useful practice problems as they will teach you skills and concepts that you can apply to other problems.

Assessed Preparation

Problem 1. Implement the solution to the coin change problem described in the lectures.

(a) Use the bottom-up strategy to compute the solutions

(b) Use the top-down strategy to compute the solutions

Consult the notes if you are unclear on the difference between the two approaches.

Problem 2. Suppose that you are a door-to-door salesman, selling the latest innovation in vacuum cleaners to less-than-enthusiastic customers. Today, you are planning on selling to some of the n houses along a particular street. You are a master salesman, so for each house, you have already worked out the amount c_i of profit that you will make from the person in house i if you talk to them. Unfortunately, you cannot sell to every house, since if a person's neighbour sees you selling to them, they will hide and not answer the door for you. Therefore, you must select a subset of houses to sell to such that none of them are next to each other, and such that you make the maximum amount of money.

For example, if there are 10 houses and the profits that you can make from each of them are 50, 10, 12, 65, 40, 95, 100, 12, 20, 30, then it is optimal to sell to the houses $1,4,6,8,10$ for a total profit of \$252. Devise a dynamic programming algorithm to solve this problem.

(a) Describe in plain English, the optimal substructure present in the problem

(b) Define a set of overlapping subproblems that are based on the optimal substructure

(c) What are the base case subproblems and what are their values?

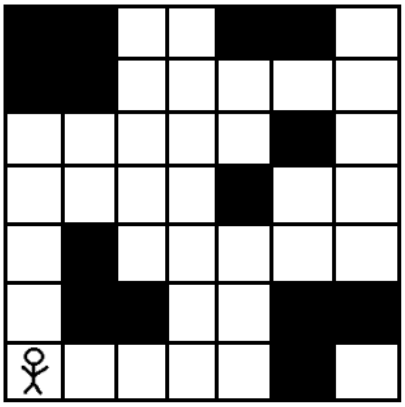
(d) Write a recurrence relation that describes the solutions to the subproblems

(e) Write psuedocode that implements all of this as a dynamic programming algorithm.

Tutorial Problems

Problem 3. Extend your solution to Problem 2 so that it returns an optimal subset of houses to sell to, in addition to the maximum possible profit.

Problem 4. You find yourself curiously stranded on an $n \times n$ grid, unsure of how you got there, or how to leave. Some of the cells of the grid are blocked and cannot be walked through. Anyway, while you're here, you decide to solve the following problem. You are currently standing at the bottom-left corner of the grid, and are only able to move up (to the next row) and to the right (to the next column). You wonder, how many ways can you walk to the top-right corner of the grid while avoiding blocked cells? You may assume that the bottom-left and top-right cells are not blocked. For example, in the following grid, the answer is 19.



Write a dynamic programming algorithm that given a grid as input, counts the number of valid paths from the bottom-left cell to the top-right cell. Your algorithm should run in $O(n^2)$ time.

Problem 5. You somehow find yourself on yet another $n \times n$ grid, but this time, it is more exciting. Each cell of the grid has a certain non-negative amount of money on it! Denote the amount of money in the cell of row i, column j by $c_{i,j}$. You are standing on the bottom-left corner $(1,1)$ of the grid. From any cell, you can only move up (to the next row), or right (to the next column). What is the maximum amount of money that you can collect?

Given $c_{i,j}$ for every cell, describe a dynamic programming algorithm to solve this problem. Your algorithm should run in $O(n^2)$ time.

Problem 6. Consider a sequence $a_1, a_2, ..., a_n$ of length n. A *subsequence* of a sequence a is any sequence that can be obtained by deleting any of the elements of a. Devise a dynamic programming algorithm that finds the length of a *longest increasing subsequence* of a. That is, a longest possible subsequence that consists of elements in strictly increasing order. Your algorithm should run in $O(n^2)$ time.

For example, given the sequence $\{\mathbf{0}, 8, 4, 12, \mathbf{2}, 10, \mathbf{6}, 14, 1, \mathbf{9}, 5, 13, 3, \mathbf{11}, 7, \mathbf{15}\}$, the longest increasing subsequence is $\{0, 2, 6, 9, 11, 15\}$ of length 6 (shown in **bold** in the original sequence).

Problem 7. Consider a pair of sequences $a_1, a_2, ..., a_n$ and $b_1, b_2, ..., b_m$ of length n and m. Devise a dynamic programming algorithm that finds the length of a *longest common subsequence* of a and b. A common subsequence is a sequence that is both a subsequence of a and a subsequence of b. Your algorithm should run in $O(nm)$. [Hint: This problem is very similar to the edit distance problem]

Problem 8. Consider a sequence $a_1, a_2, ..., a_n$ of length n. Devise a dynamic programming algorithm that finds the maximum sum of all subarrays of a. A subarray or substring of a is a contiguous subsequence, ie. a subsequence consisting of consecutive elements.

(a) Your algorithm should run in $O(n^2)$ time

(b) Your algorithm should run in $O(n)$ time [Hint: Be greedy]

Supplementary Problems

Problem 9. A ferry that is going to carry cars across the bay has two lanes in which cars can drive onto. Each lane has a total length of L. The cars that wish to board the ferry line up in a single lane, and are directed one by one onto one of the two lanes of the ferry until the next car cannot fit in either lane. Depending on which lanes the cars are directed onto, fewer or greater cars may fit on the ferry. Given the lengths of each of the cars (which will not be longer than the length of the ferry L), and assuming that the distance between cars when packed into the ship is negligible, write a dynamic programming solution that determines the maximum number of cars that can be loaded onto the ferry if distributed optimally.

(a) Solve the problem in whatever time complexity you can (without resorting to brute force)

(b) Improve your solution to $O(nL)$ time complexity (if it is not already)

For example, suppose that the car lengths are 2, 2, 7, 4, 9, 8, 1, 7, 3, 3 and the ferry is 20 units long. An optimal solution is to load 8 cars in lanes arranged like 2, 2, 7, 9 and 4, 8, 1, 7.

Problem 10. You and your friend are going to play a game. You start with a row of n coins with values $a_1, a_2, ..., a_n$. You will each take turns to remove either the first or last coin. You continue until there are no coins left, and your score is the total value of the coins that you removed. You will make the first move of the game.

(a) Show that the greedy strategy in which you simply pick the most valuable coin every time is not optimal

(b) Devise a dynamic programming algorithm that determines the maximum possible score that you can achieve if both you and your friend make the best possible moves.

For example, given the coins 6, 9, 1, 2, 16, 8, the maximum value that you can achieve when going first is 23.

Problem 11. Suppose you have n jobs where each job has a start time, a finish time and a profit value associated with it. A valid subset of the jobs is a subset of jobs such that no two jobs overlap. Your goal is to find a valid subset of jobs with maximum profit value. Below is an example.

Suppose you have following jobs where the first two values in each job represent start and finish time and the third value represents the profit of the job.

```
Job1 = [1, 10, 50]
Job2 = [8, 12, 100]
Job3 = [10, 45, 100]
Job4 = [18, 55, 150]
Job5 = [50, 60, 50]
```

The optimal schedule is Job2 and Job4 with total profit of 250. Note that Job1, Job3, Job5 is a valid schedule but its total profit is 200. Your goal is to write a dynamic programming solution to find the total profit of the optimal schedule.

1. Your algorithm should run in $O(n^2)$ time
2. **(Advanced)** Your algorithm should run in $O(n \log(n))$ time

Problem 12. A sequence is a *palindrome* if it is equal to its reversal. For example, `racecar` and $5, 3, 2, 3, 5$ are palindromes. Given a sequence $a_1, a_2, ..., a_n$ of length n, solve the following problems.

(a) Devise a dynamic programming algorithm that determines the length of the longest palindromic **subsequence** of a. Your algorithm should run in $O(n^2)$ time.

(b) Devise an algorithm that determines the length of the longest palindromic **substring** of a. Your algorithm should run in $O(n^2)$ time. (You do not have to use dynamic programming).

(c) **(Advanced)** Devise a dynamic programming algorithm that determines the length of the longest palindromic **substring** of a. Your algorithm should run in $O(n)$ time.

Problem 13. (Advanced) Consider a pair of sequences $a_1, a_2, ..., a_n$ and $b_1, b_2, ..., b_m$ of length n and m. A supersequence of a sequence a is a sequence that contains a as a subsequence. Devise an algorithm that finds the length of a *shortest common supersequence* of a and b. A common supersequence is a sequence that is both a supersequence of a and a supersequence of b. Your algorithm should run in $O(nm)$.

Problem 14. (Advanced) Suppose that I give you a string S of length n and a list L consisting of m words with length at most n. Devise a dynamic programming algorithm to determine the minimum number of strings from L that must be concatenated to form the string S. You may use a word from L multiple times. Your algorithm should run in $O(n^2m)$ time.

Problem 15. (Advanced) Given three strings A, B, and C of length n, m, and $n+m$ respectively, determine whether C can be formed by interleaving the characters of A and B. In other words, determine whether or not A and B can be found as disjoint subsequences of C. Your algorithm should run in $O(nm)$ time.