Homework Assignment #2

Due: Oct 9th, 2017

CMPUT 304
Department of Computing Science
University of Alberta

Note: All logarithms are in base 2 unless specified otherwise.
Exercises are for you and you alone. You need not submit them and they will not be graded.
You may submit as many answers to as many problems as you like.

Exercise I. Let $k_1, k_2, ..., k_7, k_8$ be 8 distinct keys in sorted order (i.e. $k_1 < k_2 < ... < k_7 < k_8$). The number of calls r_i for an element i is shown in Table 1. Show how to find an optimal BST for these sequence of keys.

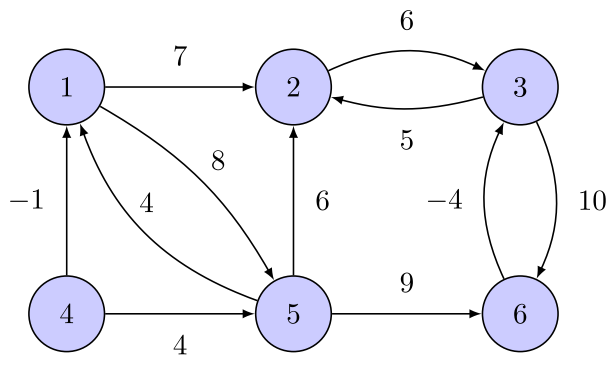
i	1	2	3	4	5	6	7	8
r_i	4	6	8	2	10	12	14	5

Table 1: Number of calls for each element.

Exercise II. Show how to compute the edit distance of strings "practicable" and "prescribing" where operations costs are as follows:

- The cost of replacing a vowel with another vowel is 3
- The cost of replacing a vowel with a constant or vice verse is 2.
- The cost of replacing a constant with a constant is 1.
- The cost of adding or removing a vowel is 2.
- The cost of adding or removing a constant is 1.

Problem 1. (1 pts) Show how to find all pairs shortest-paths on the following graph.



Problem 2. (3 pts) In the problem of **shortest bitonic tour**, the input consists of n points in the 2-dimensional plane. (So each point is given by two coordinates (x_i, y_i).) Bitonic tours are composed of two parts: the first part begins at the leftmost point (of smallest x coordinate) and *only* moves rightwards between points until this visit the rightmost point (of largest x coordinate); the second part starts at the rightmost point, and only moves leftwards through the points that weren't yet visited in

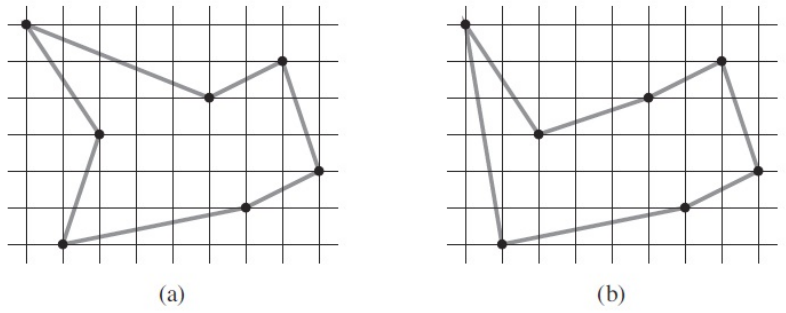


Figure 1: An example of two possible tours over the same set of points. Note how (a) isn't a bitonic tour whereas (b) is a bitonic tour.

the first part until it reaches back to the leftmost point. So at the first part one is allowed to move from (x_i, y_i) to (x_j, y_j) only if $x_i < x_j$; and in the second part one is only is allowed to move from (x_i, y_i) to (x_j, y_j) only if $x_i > x_j$. Figure 1 illustrates a possible bitonic tour.

Describe an $O(n^2)$-time algorithm for determining an optimal bitonic tour on a given set of $n + 2$ points in the plain, with (x_L, y_L) and (x_R, y_R) being the leftmost and rightmost points resp., and $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ denoting the n intermediate points in between the two. You may assume that no two points have the same x-coordinate and that all operations on real numbers take unit time.

Hint: You recursion should add the n intermediate points one by one, from left to right. Each time, maintain the cost of the optimal bitonic tour by characterizing any two possible right-most points for the two parts of the tour.
(See problem 15-3 in the text for more information.)

Problem 3. (3 pts) $2n$ cards, each with different value v_i, are placed in a row from left to right, and form the basis for a 2 player game. Each player, in her turn, picks of the two extreme cards (either the left-most card or the right-most card), removes it off the board and puts it to her pile. Once all cards have been collected, the score of each player is computed as the sum of all card-values in her pile.

Here is an example of a play with 4 card: $\boxed{1}$ $\boxed{5}$ $\boxed{2}$ $\boxed{-1}$.

1. Out of the two most extreme cards, Player one picks the -1 card.
2. Out of the two most extreme cards left (i.e. $\boxed{1}$ and $\boxed{2}$), Player two picks the 2 card.
3. Out of the two most extreme cards left (i.e. $\boxed{1}$ and $\boxed{5}$), Player one picks the 5 card.
4. As there is a single card left, Player two picks the 1 card and the game ends. Player one ends with $(-1) + 5 = 4$ points, and Player two has $2 + 1 = 3$ points.

(i) (3 pts) Give a $O(n^2)$-time algorithm that finds the max-score of each player and a strategy that obtains that score.

(ii) (Bonus exercise, 0 pts) Give a $O(n)$-time algorithm for the first player to guarantee a score which is $\geq$ the score of the second player.
Hint: This question has nothing to do with dynamic programming. Here's a little hint:

$$\underbrace{\text{red, blue, red, blue, red, blue,..., red, blue}}_{2n}$$

Problem 4. (5 pts) <u>k-means Clustering on the Line</u>
Before we describe the problem of k-means clustering, we give a definition and discuss some of its properties.
<u>Definition:</u> Given a set X of m real numbers, $x_1, ..., x_m$, we denote their cost as

$$\phi(X) = \min_{y\in\mathbb{R}} \left\{ \sum_{x\in X} (x-y)^2 \right\}$$

(i) (1 pts) Denote $\mu_X = \frac{1}{m}\sum\limits_{x\in X} x$. Prove that for any real Δ it holds that

$$\sum_{x\in X} (x - (\mu_X + \Delta))^2 - \sum_{x\in X} (x-\mu_X)^2 = m\Delta^2$$

and use this identity to infer that the point y on which the cost $\phi(X)$ is obtained is in fact μ_X. I.e. $\phi(X) = \frac{1}{m}\sum_{x\in X}(x-\mu_X)^2$.

(ii) (1 pts) Use the above fact to give a $O(1)$-time algorithm that takes as input X, μ_X and $\phi(X)$ as defined above and in addition another new input point x_{m+1} and returns the new center $\mu_{X\cup\{x_{m+1}\}}$ and the new cost of the $m+1$ points $\phi(X\cup\{x_{m+1}\})$.

We can now discuss the problem of k-means clustering in one-dimension. Your input is composed of n datapoints $x_1, ...x_n$ and your goal is to partition the n real-value points into k clusters $C_1, ..., C_k$ as to minimize $\sum_i \phi(C_i)$.

(iii) (3 pts) Give a $O(n^2k)$-time algorithm for computing the best k-means clustering cost for a given input composed of n real-value datapoints.
Hint: First prove that for any 3 distinct points in X: $x < y < z$ there cannot be two distinct clusters in the optimal solution C_i^*, C_j^* such that x and z both belong to C_i^* while y belongs to cluster C_j^*. What does this claim imply about the structure of the optimal clusters? This structure should enable you to devise a dynamic programming algorithm for the problem with an underlying array of size $n\times k$, where filling each cell requires we find the min over $O(n)$-cells. Use the previous article to assure that each one of these $O(n)$ options is found in $O(1)$-time.

Problem 5. (1 pts) In this question, you are asked to go through a thought experiment. Suppose Problems 2-4 were not given to you as part of an assignment centered on dynamic programming but rather as open ended questions. What would you do then? Chances are that for each problem you would try a few different approaches for designing an algorithm that solves it, rule these approaches out one by one, until finally it would dawn upon you to try the dynamic programming approach and that would work. In this problem, you are asked to do just this — to rule out a <u>greedy approach</u> for solving these problems.

For each of the above mentioned three problems (2, 3(i), 4(iii)), (a) suggest a reasonable greedy approach that one is likely to try to use in order to solve the problem (there are likely multiple greedy ideas for each problem, so just name one), and (b) illustrate how the greedy approach fails to solve the problem show by *constructing a specific example* on which the result of the greedy approach isn't the optimal solution.

For example: in Problem 3(i), a highly non-intelligent approach is for Player I to always pick the leftmost card. However, the following example on two cards: $\boxed{1}$ $\boxed{2}$ illustrates that in this case Player I fails to maximize her score, as she ends up with a score of 1 rather than taking the right-most of the two cards and obtaining a score of 2.