Assignment 1

Dr. Wael Abouelsaadat

Course Weight: 7.5% , **Score**: out of 395☺

Teams: to be done in teams of 3 (2< *3*< 4).

Release Date: *Nov. 7th, 2020*

Note: this document has been edited on Nov. 27th, 9:05PM. Changes colored in red.

Due Dates:

Part	*Date, time*
A	Nov. 17th, 2020, 11:50PM
B	Nov. 21st, 2020, 11:50PM
C	Nov. 27th, 2020, 11:50PM
D	Nov. 30th, 2020, 11:50PM

- ➔ Once Part A is submitted, you cannot change teams during course of assignment 1. If a team member is not participating, use the team member eval form that will be on MET/cms website, and individual evaluation sessions will be held.
- ➔ Part deadline is a hard one. Sample solution to parts submitted will be posted on MET/cms before midterm.
- ➔ Marks will be posted after complete submission of all parts of A1.
- ➔ Submission instructions will be posted by your TAs.
- ➔ Show your work! Even if you answer a problem incorrectly, you will be rewarded partial marks on how far you have progressed in your attempts towards the correct one.
- ➔ You can also present several solutions to the same problem.
- ➔ Follow the answers style presented in practice assignments for divide and conquer, greedy and dynamic programming.
- ➔ In each solution, start by writing your observations first if there is any, followed by brief high-level English description of your solution. Next, present the actual solution in pseudo-code implementation (pseudo code means it looks like code but not strictly following any programing language syntax – see https://en.wikipedia.org/wiki/Pseudocode).

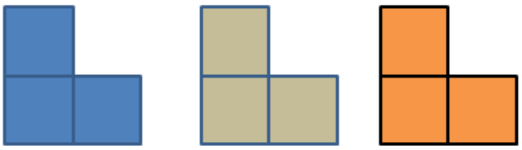
Part A: Divide and Conquer Algorithm Design

Assignment 1.1 [*30 marks*]

Imagine a marble slab of size 3m x 1m. One can tile any n x n room with such slab if n is divisible by 3. Is it true that for every n> 3 that is not divisible by 3, one can tile an n x n square with such slab and a single 1m x 1m slab? If it is possible, explain how, if it is not explain why. Use Divide and conquer in your answer.

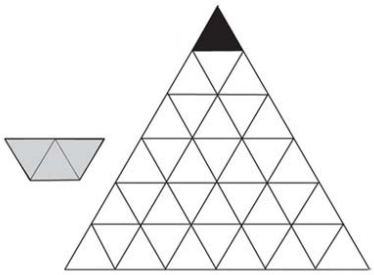
Assignment 1.2 [*35 marks*]

Imagine having a special type of ceramic that comes in an L-shape squares and in 3 distinct colors as shown below. Devise a divide and conquer solution for the following task: given a 2^n x 2^n (n>1) room with one missing square, tile it with this L-shaped ceramic so that no pair of ceramics that share an edge have the same color.



Assignment 1.3 [*35 marks*]

An equilateral triangle is partitioned into a smaller equilateral triangles by parallel lines dividing each of its sides into n> 1 equal segments. The topmost equilateral triangle is chopped off yielding a region such as the one shown below for n = 6. This region needs to be tiled with trapezoid tiles made of three equilateral triangles of the same size as the small triangles composing the region. (Tiles need not be oriented the same way, but they need to cover the region exactly with no overlaps.) Determine all values of n for which this can be done and device a divide and conquer tiling solution for such n's.



Part B: Greedy Algorithm Design

Assignment 1.4 [*25 marks*]

The tower in chess (tabyaa….) can move either horizontally to any square that is in the same row or vertically to any square that is in the same column as its current position. What is the minimum number of moves needed for the tower to pass over all the squares of an n x n chessboard? A tour does not have to start and end at the same squares, the squares where it starts, and ends are considered passed over by default.

Assignment 1.5 [*30 marks*]

There are n people, each in possession of a different part of a solution to a puzzle. They want to share the mini solutions they have with each other by sending electronic messages so that everyone can put them together to get the complete solution. What is the minimum number of messages they need to send to guarantee that everyone of them gets all the mini-solutions? Assume that the sender includes all the mini-solutions he or she already got and that a massage may only have one addressee.

Assignment 1.6 [*30 marks*]

You would like to help a veggie merchant by giving him a set of weights he can use to weight his products for the customers. Find an optimal set of n weights $\{w_1, w_2,...,w_n\}$ so that it would be possible to weigh on a two-pan balance scale an integral load in the largest possible range from 1 to W, assuming the following:

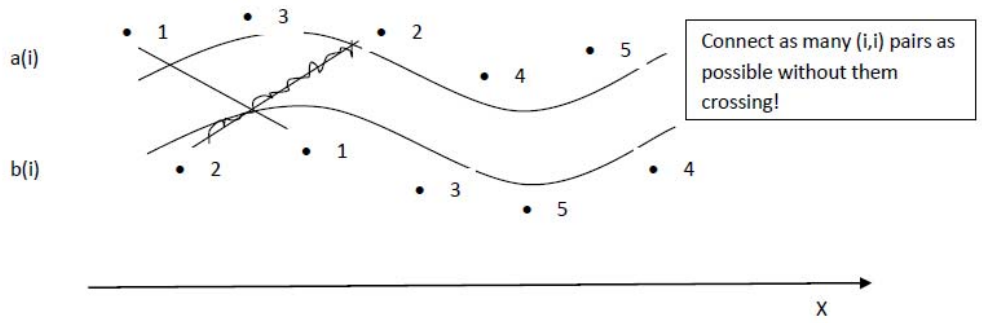
(a) weights can be put only on the free pan of the scale.

(b) weights can be put on both pans of the scale.

Part C: Dynamic Programming Algorithm Design

Assignment 1.7 [*20 marks*]

Consider a 2-D map with a horizontal river passing through its center. There are *n* cities on the northern bank with x-coordinates a(1) ... a(*n*) and n cities on the southern bank with x-coordinates b(1) ... b(*n*). You want to connect as many north-south pairs of cities as possible with bridges such that no two bridges cross. When connecting cities, you can only connect city *i* on the northern bank to city *i* on the southern bank. Using DP problems studied in class(if you wish), give a dynamic programming solution to solve this problem.



Tip: the goal behind this problem is to learn to identify the class a problem belongs to. This is just another incarnation of a problem we discussed in class.

Assignment 1.8 [*25 marks*]

Consider the case of 3 knapsacks, all of the same maximum weight *w*, where *w* is greater than the sum of some 1/3 of the items given. Your task is to divide a given set of items into 3 sets of equal value to be able to put each set in one of the bags. You can assume that the following invariant holds on the items:

The input items can be split into 3 subsets of equal value and equal weight.

Design a dynamic programming solution which identifies the subsets.

Tip: the goal behind this problem is to learn to develop thoughtful observations before jumping to write a solution. There is one important observation that will make the solution straight forward!

You are told that it is guaranteed to get an input splitable to 3 subsets of the same weight and equal value, so why bother about both. Just work on the weight or the value!. There is a case where the subsets are of equal weight but not equal value, or the reverse, but you are not asked to solve that here.

Assignment 1.9 [*30marks*]

John, a tourist currently in Cairo, decided to go on a road trip from Cairo to Aswan! He has a saving of 2,000EGP and is ready to spend it all. There are *n* poll stations connecting Cairo to Aswan along the river side, referred to as r_1, r_2, … r_n. There are also *n* poll stations connecting Cairo to Aswan through oasis in the desert, referred to as o_1, o_2,…, o_n. To reach his destination, he has to pass by *n* poll stations, where poll station *i* is either is either r_i or o_i. In other words, John has to pass by *n* poll stations, where poll station*i* (1 <= *i*<= *n*) can be located either on the river side or beside an oasis.

The total amount of money spent in his journey will be the cost of the fuel from one poll station to the next in addition to the taxes paid at every poll station he passes by (which differs from one poll station to the other). John wants to minimize the cost of his journey. He drew a map of all the possible routes between poll station *i* and poll station *j* where *j* = *i* + 1, and calculated the fuel cost for those routes.

He also inquired about the amount of taxes paid at each poll station. John wants to calculate the cost of the cheapest route from Cairo to Aswan, to know whether he can afford the trip or not.

Your are given:

riverToRiver(*i,j*): the cost of fuel needed to go from poll station r_i to poll station r_j .

oasisToOasis(*i,j*): the cost of fuel needed to go from poll station o_i to poll station o_j .

riverToOasis(*i,j*): the cost of fuel needed to go from poll station r_i to poll station o_j .

oasisToRiver(*i,j*): the cost of fuel needed to go from poll station o_i to poll station r_j .

riverTaxes(*i*): the amount of taxes paid at poll station r_i.

oasisTaxes(*i*): the amount of taxes paid at poll station oi.

Using dynamic programming, develop a solution that decides whether John the tourist can hit the road, or should stay at hotel.

Assignment 1.10 [*35 marks*]

Design a dynamic programming algorithm to count how many different ways the palindrome

Was it a cat isaw

can be read in the diamond-shaped arrangement shown below.

```
          W
        W A W
      W A S A W
    W A S I S A W
  W A S I T I S A W
W A S I T A T I S A W
  W A S I T A T I S A W
    W A S I T I S A W
      W A S I S A W
        W A S A W
          W A W
            W
```

You may start at W and go in any direction on each step, up, down, left, right through adjacent letters. The same letter can be used more than once in the same sequence.

Tip: It is easier to count first the number of ways to spell CAT I SAW.

Part D: Open Problems

The decision is yours regarding which algorithm design technique should be used to solve each of the below problems.

Assignment 1.11 [*30 marks*]

A firm has invented a super-strong glass sheet. For publicity purposes, it wants to determine the highest floor in a 100-story building from which such a glass can fall without breaking. The firm has given a tester two identical glass sheets to experiment with. of course, the same glass sheet can be dropped multiple lines unless it breaks.

What is the minimum number of droppings that is guaranteed to determine the highest safe floor in all cases?

Tip: Consider the function H(k), the maximum number of floors for which the problem can be solved in k drops.

Assignment 1.12 [*30 marks*]

Given a list of n integers, $v_1 \ldots v_n$, the product-sum is the largest sum that can be formed by multiplying adjacent elements in the list. Each element can be matched with at most one of its neighbors. For example, given the list 1, 2, 3, 1 the product sum is 8 = 1 + (2 × 3) + 1, and given the list 2, 2, 1, 3, 2, 1, 2, 2, 1, 2 the product sum is 19 = (2 × 2) + 1 + (3 × 2) + 1 + (2 × 2) + 1 + 2. Develop an algorithm to find the largest product sum [the 8 or the 19 in examples].

Assignment 1.13 [*40 marks*]

Consider nested function calls, such as f(g(x), h(y, k())), and how they are evaluated by an interpreter. Suppose that functions require one time step for each nested function call they make, and only one function call can be made at each time step. Then the example expression above can be evaluated in four steps, as follows:

- time 0: call f() - now we know that we have to call g() and h(),
- time 1: f calls g(),
- time 2: f calls h() - now we know that we have to call k(),
- time 3: h calls k(),
- time 4: all done!

You should have no trouble convincing yourself that every order of calls will require the same number of steps (for example, if f calls h() first, it does not change the other calls that must be made, just their order).

Now suppose that we have access to a parallel computer that can execute multiple functions calls at the same time. It is still the case that functions require one time step

for each nested function call they make but now different functions can make calls at the same time. Then, the example above can be evaluated in only
three steps - we cannot do better, because f requires two time steps (one to call h and one to call g):

- time 0: call f() - now we know that we have to call g() and h(),
- time 1: f calls h() - now we know that we have to call k(),
- time 2: f calls g() and h calls k(), in parallel,
- time 3: all done!

This situation can be represented by the following abstract "Efficient Function Calls" problem.

> Input: A nested function call expression E of the form "$f(e_1,\ldots e_k)$," where f is a function name and each e_i is either a variable or another nested function call expression (it is possible to have $k = 0$).
>
> Output: The order of nested calls for each function in E, to minimize the total number of steps required to evaluate the expression on the parallel computer.

Give an algorithm to solve this problem efficiently: include a brief high-level English description, as well as a detailed pseudo-code implementation. Justify that your algorithm is correct, and analyze its worst-case running time.

Tip: Start by defining an array that stores optimal values for arbitrary sub-problems. There is no simple linear structure to specify sub-problems. However, every function call expression E can be represented as a tree, with one node for each function symbol in E, where the children of node f are exactly the functions called directly by f.