# CSE 40113: Algorithms
## Homework 3

You may complete this homework in groups of 3 or less students. Note that the integrity policy applies: your group should write up your own work, although you're welcome to work on the problems in a larger group. If you have any questions, please re-read both the homework guidelines and the academic integrity policy carefully, and then come discuss any questions or concerns with me.

## Required Problems

1. You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, which you are able to make an accurate estimate of based on publicly available information on the internet. You need to decide which houses to rob, where the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and it will automatically contact the police if any three adjacent houses were broken into on the same night.

   (a) Give a (small) example of why choosing the largest value house first (in other words, being greedy) will not necessarily yield the most money.

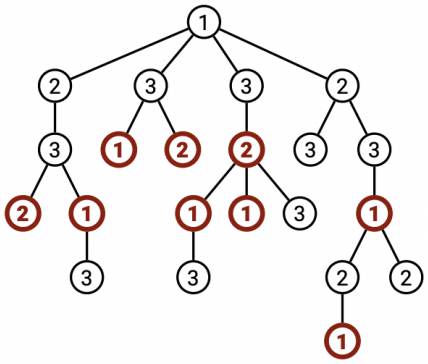
   (b) Write a recursive formula or expression for the maximum amount of money you can gain.

   (c) Given an array Houses$[1 \dots n]$, where Houses$[i]$ is the amount of money estimated in the $i^{th}$ house, design an algorithm to calculate the maximum amount of money you can rob tonight without alerting the police.

2. Since so few people came to last year's holiday party, the president of Giggle decides to give each employee a present instead this year. Specifically, each employee must receive on the three gifts: (1) an all-expenses-paid sixweek vacation anywhere in the world, (2) an all-the-pancakes-you-can-sort breakfast for two at Jumping Jack Flash's Flapjack Stack Shack, or (3) a burning paper bag full of dog poop. However, corporate regulations prohibit any employee from receiving exactly the same gift as his/her direct supervisor. Any employee who receives a better gift than his/her direct supervisor will almost certainly be fired in a fit of jealousy. As Giggle's social party czar, it's your job to decide which gift each employee receives.

   More formally, you are given a rooted tree $T$, representing the company hierarchy, stored as a data structure where you have saved the root as $T.root$. For each node $v$ in the tree, there is a list of children saved in an array. You want to label the nodes of T with integers 1, 2, or 3, so that every node has a different label from its parent. The cost of an labeling is the number of nodes with smaller labels than their parents.

   See below for an example of such a tree, which has cost 9, where all fired employees are shown in bold (since they have values lower than their direct supervisor.)



(a) Give a recursive formulation to calculate the minimum cost labeling rooted at a node $v$. (Hint: take inspiration from section 3.10 here.)

(b) Design and analyze an algorithm to compute the minimum-cost labeling of $T$, so that the fewest number of people are fired. (Yes, you may send the president a flaming bag of dog poop.)

3. Recall that a *subsequence* of an array is a subset of the elements in the array in the same order. (So: a subset in the same order, but they don't have to be next to each other in the array.) For example, 1, 1 5 3, and 2 1 9 5 8 7 2 6 5 3 are all subsequences of $A = [2\ 1\ 9\ 5\ 8\ 7\ 2\ 6\ 5\ 3\ ]$, where each $A[i]$ is a single digit.

   A sequence $X[1..n]$ is called *oscillating* if $X[i] < X[i+1]$ for all even $i$, and $X[i] > X[i+1]$ for all odd $i$. Describe an efficient algorithm to compute the length of the longest oscillating subsequence of an arbitrary array $A$ of integers.

4. Sample Solved Problem: A shuffle of two strings $X$ and $Y$ is formed by interspersing the characters into a new string, keeping the characters of $X$ and $Y$ in the same order. For example, the string BANANAANANAS is a shuffle of the strings BANANA and ANANAS in several different ways:
BANANAANANAS, BANANAANANAS, or BANANANANAS.

   Similarly, the strings PRODGYRNAMAMMIINCG and DYPRONGARMAMMICING are both shuffles of DYNAMIC and PROGRAMMING:
PRODGYRNAM AMMIINCG and DYPRONGARMAMMICING.

   Given three strings $A[1..m]$, $B[1..n]$, and $C[1..m+n]$, describe and analyze an algorithm to determine whether C is a shuffle of A and B.

   **Solution:**

**Recursive formulation:** We define a boolean function $Shuf(i, j)$, which is True if and only if the prefix $C[1..i+j]$ is a shuffle of the prefixes $A[1..i]$ and $B[1..j]$. This function satisfies the following recurrence:

- Shuf$(i, j)$ = true if $i = j = 0$
- Shuf$(0, j-1)$ AND $(B[j] = C[j])$ if $i = 0$ and $j > 0$
- Shuf$(i-1, 0)$ AND $(A[i] = C[i])$ if $i > 0$ and $j = 0$
- (Shuf$(i-1, j)$ AND $(A[i] = C[i+j])$) OR (Shuf$(i, j-1)$ AND $(B[j] = C[i+j])$) if $i > 0$ and $j > 0$

The proof that this formulation is correct can be shown via induction: if you're considering the $i + j^{th}$ character of $C$, it must be from either $A[i]$ or $B[j]$. We are trying both options, and returning true if either works. The base cases handle either A or B being empty, in which case either we've matched everything (and both are 0) or we must exactly match the rest of $C$ to which ever string is left.

**Dynamic programming:** We need to compute Shuf(m, n); if it is true, then we can shuffle the entire strings A and B into a string C. Since Shuf[i,j] needs to look up values Shuf[i-1,j] and Shuf[i,j-1] (as well as comparing some values from the 3 arrays), we can memoize all values into a two-dimensional array Shuf[0..m][0..n]. Each array entry Shuf[i, j] depends only on the entries immediately below and immediately to the right: Shuf[i-1, j] and Shuf[i, j-1]. Thus, we can fill the array in standard row-major order.

```
SHUFFLE?(A[1..m], B[1..n], C[1..m+n]):
  Shuf[0,0] ← TRUE
  for j ← 1 to n
      Shuf[0,j] ← Shuf[0,j−1] ∧ (B[j] = C[j])
  for i ← 1 to n
      Shuf[i,0] ← Shuf[i−1,0] ∧ (A[i] = B[i])
      for j ← 1 to n
          Shuf[i,j] ← FALSE
          if A[i] = C[i+j]
              Shuf[i,j] ← Shuf[i,j] ∨ Shuf[i−1,j]
          if B[i] = C[i+j]
              Shuf[i,j] ← Shuf[i,j] ∨ Shuf[i,j−1]
  return Shuf[m,n]
```

The algorithm runs in $O(mn)$ time, and (if we keep the entire 2d arrays) takes the same amount of space.

Note that this can be improved to $O(m)$ (or $O(n)$) by keeping only two rows: the one we are currently filling in, and the row immediately preceding it.