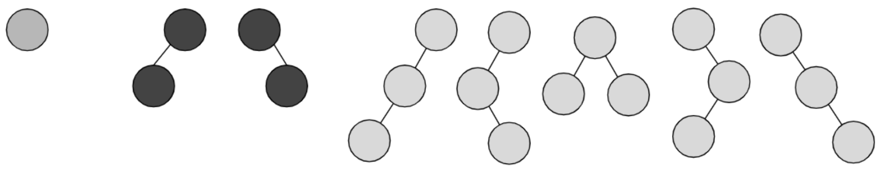
Homework 4

We may grade a **subset of the assigned questions**, to be determined after the deadline, so that we can provide better feedback on the graded questions.
Unless otherwise stated, each question requires sufficient justification to convince the reader of the correctness of your answer.
For bonus questions, we will not provide any insight during office hours or Piazza, and we do not guarantee anything about the difficulty of these questions.
We strongly encourage you to typeset your solutions in LaTeX.
If you collaborated with someone, you must state their name(s). You must write your own solution for all problems and may not look at any other student's write-up.

0. If applicable, state the name(s) and uniqname(s) of your collaborator(s).

Solution:

1. Let $T(n)$ be the number of (rooted) binary tree configurations with exactly n nodes. Examples of the first few values of T are shown below – two trees are distinct if a single node changes position, with right and left children considered distinguishable.



$$T(0) = 1, T(1) = 1, T(2) = 2, T(3) = 5$$

$T(n)$ can be described by the following recurrence:

$$T(n) = T(0)T(n-1) + T(1)T(n-2) + ... + T(n-1)T(0).$$

(a) Give a combinatorial argument for why this recurrence correctly counts the number of trees with n nodes.

(b) Write down pseudocode to compute $T(n)$, using dynamic programming. (You may choose to use a top-down or bottom-up approach.) Your implementation should have runtime $O(n^2)$; briefly justify why this is true.

(c) Now consider a variation on this problem: write down a recurrence to count the number of *non-empty* binary trees where every node has 0 or 2 children. Briefly explain why your recurrence is correct.

Solution:

2. Give recurrence relations (including base cases) that are suitable for dynamic programming solutions to the following problems. You do not need to prove correctness, but provide justification for each.

   (a) Visit-Texas($A[1 \dots n]$): Daphne is finally back home in Texas after a year! To celebrate, she will go on a hiking trip at Big Bend National Park. She will start her hike at trail marker 0. There are n campgrounds along the way at miles $a_1 < a_2 < \cdots < a_n$, where each a_i is measured with respect to the starting point. She can only stop at these specified campgrounds, but is able to choose which grounds she stops at. Additionally, she must end her hiking trip at the last campground, located at a_n.

   Ideally, she would like to hike 15 miles per day, however, due to the spacing between campgrounds, this may not always be achievable. If she hikes x miles on a given day, she will face a penalty of $(15 - x)^2$ for that day. Return the minimum total penalty, where the total penalty is defined as the sum of all daily penalties.

   For example, if $A = [5, 25, 27]$, then we have 4 possible hiking trips (the left side denotes which campgrounds she stopped at):

   - $1, 2, 3 \rightarrow (15-5)^2 + (15-20)^2 + (15-2)^2 = 294$
     Daphne stopped at campground 1, located at mile 5, after she hiked 5 miles, so the penalty is $(15-5)^2$. She then stopped at campground 2, located at mile 25, after she hiked 20 miles, so the penalty is $(15-20)^2$. She then ended at campground 3, located at mile 27, after she hiked 2 miles, so the penalty is $(15-2)^2$.
   - $2, 3 \rightarrow (15-25)^2 + (15-2)^2 = 269$
   - $1, 3 \rightarrow (15-5)^2 + (15-22)^2 = 149$
   - $3 \rightarrow (15-27)^2 = 144$

   Therefore, the the minimum total penalty is 144.

   Solution:

   (b) In today's age, many people rely on search engines such as Google as their primary source to get the information they need. When we type fast, we often forget to insert spaces between the words we enter into search bars. However, search engines are usually still able to look for the correct information. This is in part facilitated by the ability to check and see if the input string can be split into dictionary words. You can implement a function that does this yourself using dynamic programming!

   Word-Break(L, W): Given a string W and a set of words L, you want to decide if W can be segmented into words from L. (You may use a word multiple times.)

   Write a recurrence for the Word-Break problem. For simplicity, you can assume there is a function $L(w)$ that returns true iff word $w \in L$.
   Examples:

   - Input: $L =$ ['EECS', '376', 'Is', 'Awesome'], $W =$ 'AwesomeEECS376IsAwesome'.
     Output = true

- Input: $L =$ [‘cats’, ‘and’, ‘dogs’], $W =$ ‘catsanddogs’.
  Output = true
- Input: $L =$ [‘cats’, ‘and’, ‘dogs’], $W =$ ‘catsandogs’.
  Output = false
- Input: $L =$ [‘cats’, ‘and’, ‘dogs’], $W =$ ‘catsand’.
  Output = true

Solution:

3. You and your team of ice sculptors are sculpting an ice castle. Unfortunately, it is melting! You want to split your sculptors in half to fit through the two available exits. For maximum safety, you will want to balance the total usage time of the two exits, to get sculptors out of the room as fast as possible. The (usage) time of an exit is defined as the total of all of the exit times of the sculptors assigned to that exit. To simplify this problem, you assume that the sculptors have (positive) integer exit times.

   Concretely, assume that there are n sculptors in the room, where sculptor i takes time $w_i \in \mathbb{N}$ to leave (note that w_i is assumed to be an integer). Given the non-sorted list $(w_1, \cdots, w_n)$, pick out the sculptors ($S \subseteq \{1, \cdots, n\}$) who leave through the first exit. The rest, $\bar{S} = \{1, \cdots, n\} \backslash S$, will automatically leave through the second exit. Your goal is to minimize

$$|(\text{exit 1's time}) - (\text{exit 2's time})| = \left| \Big(\sum_{i \in S} w_i \Big) - \Big(\sum_{i \in \bar{S}} w_i \Big) \right|$$

   (a) The naïve algorithm is given by: “exhaust all possible combinations of sculptors that leave through exit 1; in each combination, find the absolute value of the difference between exit 1’s time and exit 2’s time. Return the combination with the smallest absolute value.” Find the complexity of the naïve algorithm in big-O notation.

   (b) Write a recurrence, using DP, that can be used to determine the smallest possible difference between the exit times of the two groups. Briefly explain why your recurrence is correct.

   Note: It should be possible to implement your recurrence in time $O(n \cdot C)$, where $C = \sum_{i=1}^{n} w_i$ is the total time it takes for all sculptors to exit (but you do not have to write the pseudocode).

   Hint: One potential approach is to write a recurrence to check if it is possible to take a subset whose sum is j, for some j passed into the recurrence. (But note that there are also other ways to solve it with a different DP recurrence.)

Solution:

4. An environment investigation group is testing the oxygen density along the Huron River. Consider a 1-D river of length n discretized as $n+1$ evenly spaced points, denoted as $(0, 1, 2, \dots, n)$. The oxygen density at point k is a_k. Assume $a_0 > a_1 > \dots > a_n$ and $a_0 > 1 > a_n$, where $a_i \neq 1$ for any a_i. The task is to find the point where the density drops under 1, i.e. find k such that $a_k > 1 > a_{k+1}$, by a *programmable drone*. The drone can both travel from point to point and take measurements at its current position.

   - The drone can fly from i to j using time $|i-j|t_1$.
   - The drone can measure the oxygen density at its current position using time t_2.

   The *search time* is the time the drone spends in both traveling and measuring until it finds the critical point k where $a_k > 1 > a_{k+1}$. Initially the drone is placed at position 0.

   (a) Describe the optimal search strategy when $t_2 = 0$ (and $t_1 \neq 0$). That is, find the optimal search strategy for the case where it takes no time to take measurements.

   (b) Describe the optimal search strategy for when $t_1 = 0$ (and $t_2 \neq 0$). That is, find the optimal strategy for the case where it takes no time to jump from one location to another location.

   In Parts (a) and (b), just provide a description; you do not need to write any recurrence or pseudocode.

   (c) The previous two parts showed that there are many possible search algorithms which will fall somewhere between the two extremes you found in parts (a) and (b). Each of these algorithms will have a worst-case search time, and the optimal algorithm depends on the values of t_1 and t_2. Write a recurrence relation for the cost of an optimal search strategy. Explain why your recurrence is correct.

   Hint: During the search, if the search area shrinks to $[k, j]$ where $a_k > 1 > a_j$ and the drone is either at k or j, then the optimal strategy from this moment on should be the same as the optimal strategy for searching over a shorter river $(0, 1, 2, \dots, j-k)$ with the drone placed at 0.

Solution: