This assignment is **due on April 5** and should be submitted on Gradescope. All submitted work must be *done individually* without consulting someone else's solutions in accordance with the University's "Academic Dishonesty and Plagiarism" policies.

As a first step go to the last page and read the section: "Advice on how to do the assignment". For help with structuring your solution, please also take a look at the section: "Exemplar for Knapsack".

Background.

You found a part-time job working for a mad scientist. They're currently working on special shoes to walk through lava and guess who's tasked with testing them out? You guessed it: you! The shoes are still a prototype and after stepping in lava for a bit, we need to cool them off in water. Unfortunately, the material of the shoes is porous and so standing in water for too long isn't nice either. Ideally, you take one step in lava and then one in water, alternating between the two.

To make things a bit more interesting, your employer has designed a parkour for you to walk in the form of a binary tree and you're supposed to start at the root and walk to one of the leaves of the tree. Every node of this tree is either red (lava) or blue (water) and has a value associated with it, indicating how much you get paid if you step on it. Needless to say, your goal is to 1. survive this ordeal, and 2. make sure your boss pays you as much as possible for using you as a test subject.

For science!!!

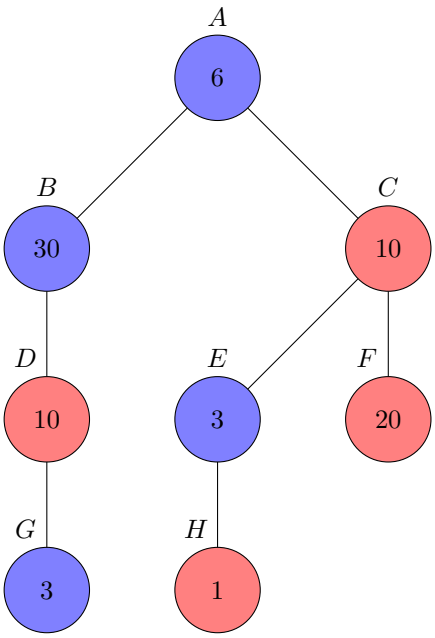

Figure 1: Example

Problem 1. (20 points)

Let's start of with a slightly simplified version of the problem: all nodes of the binary tree are regular stone and for now you just want to figure out which root to leaf path maximizes your pay. More formally, given a rooted binary tree consisting

of n nodes, where every node has a positive integer value, design a DP that finds the value of the root to leaf path that maximizes the summed value of the nodes on the path. For full marks, your algorithm should run in $O(n)$ time.

For the example in Figure 1, your algorithm should return 49, the value of the root to leaf path $ABDG$.

Your task is to:

a) Define the subproblems needed by your algorithm.
What we expect to see: Definition of OPT(...), and its parameters

b) Define the recurrence.
What we expect to see: A recurrence that relates each subproblem to "smaller" subproblems. We just want to see a clear description of the recurrence here. The justification for the recurrence goes into (e).

c) Define the base cases.
What we expect to see: A description of the base cases and the values that OPT(..) takes on these base cases.

d) Describe how the maximum profit is obtained from $OPT(..)$.
What we expect to see: A description of how to obtain the optimal value to the original problem using the optimal values of the subproblems.

e) Justify your recurrence and base cases, and that your answer to part (d) correctly returns the optimal value.
What we expect to see: The sufficiency of the base cases, i.e. the recurrence will always end up at a base case, the correctness of the recurrence and the values for the base cases, and that you correctly obtained the optimal revenue from $OPT(..)$.

f) Prove the time complexity of your algorithm.
What we expect to see: An analysis of the running time required to compute all the $OPT(..)$ values and to obtain the maximum revenue from $OPT(..)$

Problem 2. (80 points)

Back to our original experiment: The mad scientist has set up the parkour, but the lava and water nodes aren't always alternating, so we may have to jump over some nodes in order to ensure that our feet neither burn nor soak. More formally, given a rooted binary tree on n nodes, where every node has a positive integer value and is colored either red or blue, an alternating path is a subsequence of a root to leaf path such that along this subsequence we alternatingly have red and blue nodes. Recall that a subsequence can skip nodes, so for example [2,4] is a subsequence of [1,2,3,4]. Design a DP that computes the value of the alternating root to leaf path with the highest summed value. For full marks, your algorithm should run in $O(n)$ time.

For the example in Figure 1, the maximum alternating subsequence is $[B, D, G]$ with value $30 + 10 + 3 = 43$. The subsequence $[A, F]$ is also alternating, but since its value is only $6 + 20 = 26$, it isn't the maximum. The subsequence $[A, B, D]$ isn't

alternating, since A and B are consecutive and have the same color.

Your task is to:

a) Define the subproblems needed by your algorithm.

b) Define the recurrence.

c) Define the base cases.

d) Describe how the maximum profit is obtained from $OPT(..)$.

e) Justify your recurrence and base cases, and that your answer to part (d) correctly returns the optimal value.

f) Prove the time complexity of your algorithm.

Advice on how to do the assignment

- Assignments should be typed and submitted as pdf (no pdf containing text as images, no handwriting).
- Start by typing your student ID at the top of the first page of your submission. Do **not** type your name.
- Submit only your answers to the questions. Do **not** copy the questions.
- When asked to give a plain English description, describe your algorithm as you would to a friend over the phone, such that you completely and unambiguously describe your algorithm, including all the important (i.e., non-trivial) details. It often helps to give a very short (1-2 sentence) description of the overall idea, then to describe each step in detail. At the end you can also include pseudocode, but this is optional.
- In particular, when designing an algorithm or data structure, it might help you (and us) if you briefly describe your general idea, and after that you might want to develop and elaborate on details. If we don't see/understand your general idea, we cannot give you marks for it.
- Be careful with giving multiple or alternative answers. If you give multiple answers, then we will give you marks only for "your worst answer", as this indicates how well you understood the question.
- Some of the questions are very easy (with the help of the slides or book). You can use the material presented in the lecture or book without proving it. You do not need to write more than necessary (see comment above).
- When giving answers to questions, always prove/explain/motivate your answers.
- When giving an algorithm as an answer, the algorithm does not have to be given as (pseudo-)code.
- If you do give (pseudo-)code, then you still have to explain your code and your ideas in plain English.
- Unless otherwise stated, we always ask about worst-case analysis, worst case running times, etc.
- As done in the lecture, and as it is typical for an algorithms course, we are interested in the most efficient algorithms and data structures.
- If you use further resources (books, scientific papers, the internet,...) to formulate your answers, then add references to your sources and explain it in your own words. Only citing a source doesn't show your understanding and will thus get you very few (if any) marks. Copying from any source without reference is considered plagiarism.

Exemplar for Knapsack

The following exemplar is designed to help you understand the requirements of the assignment. You should still write your submission in your own words and not plagiarize the exemplar.

a) Define the subproblems needed by your algorithm.
Consider the items in some arbitrary order. Define $\text{OPT}(i, w)$ to be the optimal solution to the knapsack problem consisting of the first i items and weight limit w for $0 \leq i \leq n$ and $0 \leq w \leq W$. When $i = 0$, this corresponds to a knapsack problem with no items.

b) Define the recurrence and base cases.

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i-1, w) & \text{if } w_i > w, \\ \max\{v_i + \text{OPT}(i-1, w-w_i), \text{OPT}(i-1, w)\} & \text{otherwise} \end{cases}$$

c) Define the base cases
The base cases are when $i = 0$. In this case, $\text{OPT}(0, w) = 0$ for all $0 \leq w \leq W$.

d) Describe how to obtain the most value achieved by any feasible subset of items from the subproblems.
The most value achieved is in the subproblem $\text{OPT}(n, W)$.

e) Justify your recurrence, base cases, as well as how you obtain the optimal value to the original problem from $OPT(..)$.
The base cases are sufficient as each $\text{OPT}(i, w)$ depends on some $\text{OPT}(i-1, w)$. Thus, the recurrence will always end up at $\text{OPT}(0, w)$ for some w. We have $\text{OPT}(0, w) = 0$ since the most value that can be achieved with no items is 0.
We now justify the recurrence. Let $S(i, w)$ be the optimal solution for the first i items with weight limit w. The recurrence for $\text{OPT}(i, w)$ is correct because:

- if $w_i > w$, then $S(i, w)$ cannot contain item i and so is a subset of the first $i-1$ items with weight at most w, so $S(i, w) = S(i-1, w)$ and $\text{OPT}(i, w) = \text{OPT}(i-1, w)$
- if $w_i \leq w$, then either item i is in $S(i, w)$ or not:
  - if item i is in $S(i, w)$, then the remaining items of $S(i, w)$ is a subset of the first $i-1$ items with weight at most $w - w_i$ so $S(i, w) = \{i\} \cup S(i-1, w-w_i)$ and so $\text{OPT}(i, w) = v_i + \text{OPT}(i-1, w-w_i)$.
  - else, $S(i, w)$ is a subset of the first $i-1$ items with weight at most w, so $S(i, w) = S(i-1, w)$ and $\text{OPT}(i, w) = \text{OPT}(i-1, w)$.

Finally, the optimal value for the original problem is $\text{OPT}(n, W)$ as by definition of OPT, this is the most value that can be achieved by a subset of all n items with a weight limit of W.

f) Prove the time complexity of your algorithm.
The base cases take time $O(W)$ as there are $W + 1$ of them and setting them to 0 takes constant time. For the recursive cases, there are $O(nW)$ iterations and each iteration takes $O(1)$ time since it only requires a constant number of comparisons, arithmetic operations and array lookups. Thus, the recursive cases takes time $O(nW)$. Returning the optimal value takes $O(1)$ time for an array lookup. Thus, the total time is $O(W) + O(nW) + O(1) = O(nW)$.