University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
Marc Fuchs

Algorithm Theory
Exercise Sheet 4

Due: Friday, 14th of November 2025, 10:00 am

Remark: You are required to use the principle of *dynamic programming* in all of your algorithms.

Exercise 1: USB drive *(7 Points)*

You have a special USB drive with a total capacity of T gigabytes (GB), where $1 \leq T$. Your goal is to fill the drive with as much data as possible **without (ever) exceeding** its capacity T.
You have access to an unlimited supply of two types of data files:

- **Text Packages (Type A):** Each is exactly A GB in size.
- **Video Packages (Type B):** Each is exactly B GB in size.

A, B and T are integers, and we know that $1 \leq A < B \leq T$. You can add these files one by one, starting from an empty drive (0 GB used). You also have a one-time-use **compression algorithm** that can be applied **at most once** at any point[1]. When you run it, it instantly compresses all data currently on the drive, reducing the total space used to **half** of its current amount, **rounded down** (e.g., $\lfloor \text{space\_used}/2 \rfloor$). After compressing, you can continue to add more files to the drive.
Your task is to determine the **maximum amount of space** you can fill on the drive without ever exceeding the T GB capacity. Your algorithm to solve this must run in $O(T)$ time. *(7 Points)*

Hint: Before you solve this problem, think of how you would use *dynamic programming* to solve the easier problem where you do not have a one-time-use compression algorithm[2], i.e., you should be able to tell for all integers $1 \leq x \leq T$ if you can exactly fill space x using the two package types. You can now extend this algorithm to solve the original problem.

Exercise 2: Weighted Independent Set on Trees *(6 Points)*

Let $G = (V, E)$ be a graph with **node** weights $\in \mathbb{N}$, denoted by $w(v)$ for node $v \in V$. We call a subset of the nodes $I \subseteq V$ a maximal weight independent set if for all nodes $u, v \in I$ with $u \neq v$, we have $\{u, v\} \notin E$ (in other words, no two nodes in I are neighbors in G) and furthermore the sum over the weights in I, denoted as $w(I)$,

$$w(I) := \sum_{v \in I} w(v)$$

is as large as possible among all such independent sets.
For this task, assume you are given a (node weighted) tree T of n nodes and you want to compute the maximum weight independent set of T. For simplicity, you can assume that the tree is rooted, i.e., there is a root node r and each node v has exactly one parent node (except for the root r) and all other neighbors of v are its children. Try to come up with a *dynamic programming* solution that

[1] To be clear, you can compress at any point where the currently stored data takes $0 \leq x \leq T$ GB.

[2] Note that this simpler problem can actually be solved without DP in time $O(T/B)$. However, it is not so clear if this solution can be extended to solve the original problem. Hence, try to construct an $O(T)$ time DP algorithm!

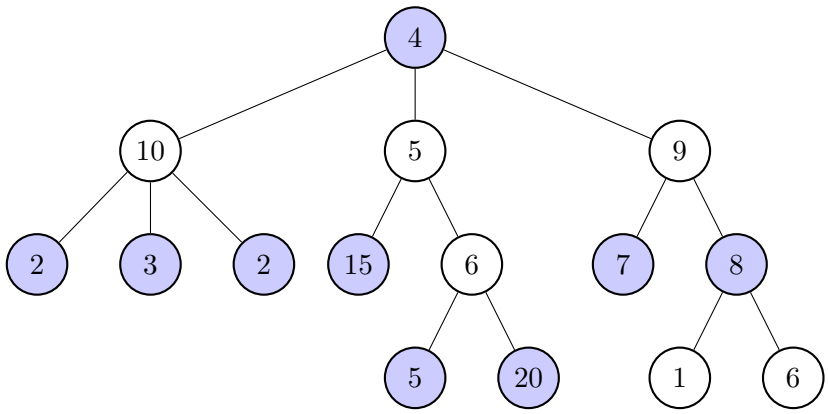

Figure 1: Example tree with a maximum weight independent set marked in blue.

runs in time $O(n)$. Argue that your running time is as stated. *(5 Points)*

Hint: For each node v consider the tree T_v, that is the subtree of T rooted at v. What is the maximum weight independent set of T_v if you already know the maximum weight independent sets of T_u for all children u of v?

Exercise 3: Longest Walk *(7 Points)*

Suppose you are given a graph $G = (V, E)$, where each node $v \in V$ is labeled with an elevation value $h(v) \in \mathbb{N}$. You can safely traverse an edge $(u, v) \in E$ from node u to node v if and only if the absolute difference between the elevation values of u and v is at most δ, that is, $|h(u) - h(v)| \leq \delta$, where $\delta > 0$ is an integer parameter.

Our goal now is to find a longest possible walk in the graph such that the walk starts at some node $s \in V$ and ends at another node $t \in V$. The walk should consist of two segments: an uphill segment, where the elevation must strictly increase in each step, followed by a downhill segment, where the elevation must strictly decrease in each step.

Note that a walk in a graph is path on which nodes are allowed to repeat. However, in the uphill segment, the elevation values are strictly increasing and in the downhill segment, the elevation values are strictly decreasing. Each node can therefore appear at most once in the uphill segment and at most once in the downhill segment (but can appear in both segments).

Your input consists of the graph $G = (V, E)$, the elevation function $h : V \to \mathbb{N}$, the starting node s, the target node t, and the parameter $\delta > 0$. Your task is to find a longest possible walk as described above or determine that no such walk exists.

Give an algorithm that solves the problem in time $O(nm)$, where as usual $n = |V|$ and $m = |E|$. Argue correctness and run time.

Hint: It makes sense to first solve the following problem: For every $v \in V$, find the longest path from s to v on which the elevation values strictly increase or determine that no such path from s to v exists.