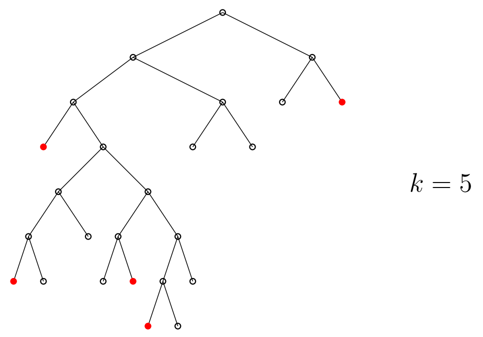
CS 473, Spring 2025
Homework 3 (due Feb 20 Thu 10am)

Instructions: As in previous homeworks.

Problem 3.1: *(Social distancing for koalas?)* We are given a binary tree T with n nodes, and a number $k \le n$. (You may assume that every non-leaf node has exactly 2 children.) We want to pick a largest subset S of *leaves* such that every two different leaves in S have distance at least k. (The distance between two nodes u and v refers to the length of the (unique) path from u to v in T.

In the example below, a feasible subset S for $k = 5$ is shown in red, of size 5. (Turn the picture upside down if you are a koala :-)

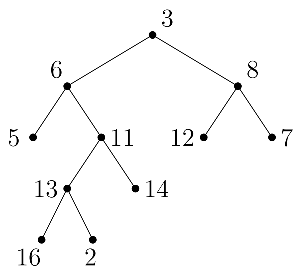


Describe a dynamic programming algorithm to solve this problem. Include the following steps: (i) first define your subproblems precisely, (ii) then derive the recursive formula (including base cases) with brief justifications, (iii) specify the evaluation order, and (vi) analyze the running time and space as a function of n and k (think of k as smaller than n but k is not necessarily a constant). For this problem, you do not need to write pseudocode if your recursive formula and evaluation order are described clearly. And you do not need to write pseudocode to output an optimal solution; just the optimal size suffices.

[Hint: in defining subproblems, use an additional parameter $i \in \{0, \ldots, k\}$ and add an extra constraint that all selected nodes must be of distance at least i from the root...]

Problem 3.2: *(Turning a sequence into a tree)* We are given a sequence of n numbers $\langle a_1, \ldots, a_n \rangle$. We want to compute a proper binary tree T with nodes $a_1, \ldots, a_n$ such that the *inorder traversal* is precisely $\langle a_1, \ldots, a_n \rangle$, while minimizing the cost function $c(T) = \sum_{(a_i, a_j) \text{ is an edge of } T} |a_i - a_j|$. (A proper binary tree refers to a rooted tree where each internal node of T has exactly 2 children.)

For example, for the input sequence $\langle 5, 6, 16, 13, 2, 11, 14, 3, 12, 8, 7\rangle$, one feasible solution is the following tree, with cost $|3-6| + |3-8| + |6-5| + |6-11| + |8-12| + |8-7| + |11-13| + |11-14| + |13-16| + |13-2| = 38$ (we do not claim that this is optimal).



Describe a dynamic programming algorithm to solve this problem. For full credit, the running time should be at most $O(n^4)$. Include the following steps: (i) first define your subproblems precisely, (ii) then derive the recursive formula (including base cases) with brief justifications, (iii) specify the evaluation order, and (vi) analyze the running time and space. For this problem, you do not need to write pseudocode if your recursive formula and evaluation order are described clearly. And you do not need to write pseudocode to output an optimal solution; just the optimal cost suffices.

[Hint: define $C(i, j, m)$ to be the minimum cost over all binary trees T for the subsequence $\langle a_i, a_{i+1}, \ldots, a_j\rangle$ with the extra constraint that the root is a_m. . .]

Problem 3.3: We are given a directed graph $G = (V, E)$ with n vertices and m edges (with $m \geq n$), where each edge e has a *weight* $w(e)$ and each vertex v has a *penalty* $p(v)$.

In addition, we are given a *source* vertex $s \in V$, and a number $K \leq n$.

We want to find a path π from s to some vertex v using at most K edges, minimizing $\text{cost}(\pi) := (\sum_{e\in\pi} w(e)) + p(v)$ (i.e., the weight of the path π plus the penalty at v). Note that in this problem, we want to output not just the optimal cost but also an optimal path.

(a) (50 pts) Describe a dynamic programming algorithm to solve this problem in $O(mK)$ time and $O(nK)$ space.

Include *all* of the following steps: (i) first define your subproblems precisely, (ii) then derive the recursive formula (including base cases) with brief justifications, (iii) specify the evaluation order, (iv) write pseudocode to output the optimal cost, (v) write pseudocode to output an optimal path, and (vi) analyze the running time and space.

[Hint: define $C(u, k)$ to be the minimum cost of a path from u using at most k edges. . .]

(b) (50 pts) Design and analyze a slightly slower but more space-efficient algorithm to solve this problem in $O(mK \log K)$ time and $O(n \log K)$ space.

[Hint: use divide-and-conquer in combination with your algorithm from (a) (somewhat similar to the space-saving trick from class but with a different recurrence).]