CS 473, Spring 2023
Homework 3 (due Feb 15 Wed 9pm)

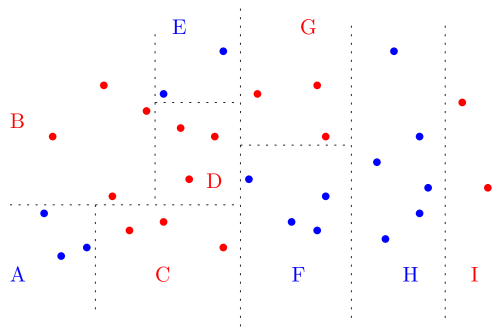
Instructions: As in Homework 1.

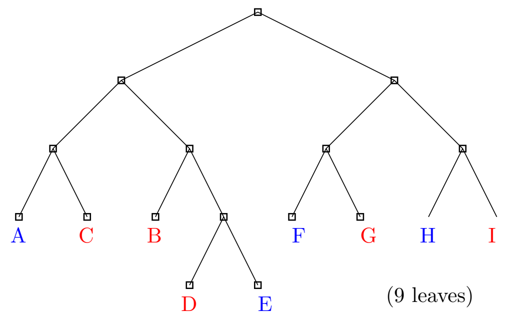
Problem 3.1: We are given a set P of n points in 2D, each colored red or blue. A *binary space partition (BSP)* is a binary tree where each node v is associated with a (possibly unbounded) rectangle $R(v)$. At the root v_r, the rectangle $R(v_r)$ is the entire plane. For every non-leaf node v with children v_1 and v_2, the rectangles $R(v_1)$ and $R(v_2)$ do not overlap and their union is $R(v)$ (in other words, $R(v_1)$ and $R(v_2)$ are obtained by cutting $R(v)$ into two by some vertical or horizontal line).

We say that a BSP is *proper* if at each leaf v, the points in $P \cap R(v)$ all have the same color.

Our problem is to find a proper BSP while minimizing the number of leaves. (The motivation comes from decision tree learning.)

Describe a polynomial-time dynamic programming algorithm to compute the optimal value. Include the following steps: (a) first define your subproblems precisely, (b) then derive the recursive formula (including base cases) with brief justifications (no need for long correctness proofs), (c) specify a valid evaluation order, and (d) analyze the running time and space (as a function of n). 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 the optimal tree itself.





Problem 3.2: We are given a set S of n real numbers between 0 and 1. We are also given an integer $L \leq n$. We want to select a subset $Q \subset S$ of L numbers to minimize the cost function

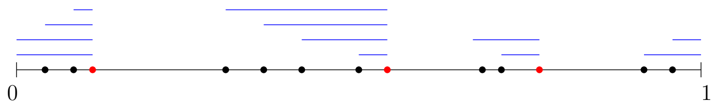
$$c(S,Q) = \sum_{x \in S} \min_{q \in Q} \delta(x,q),$$

where

$$\delta(x,q) = \begin{cases} q - x & \text{if } q \geq x \\ q - x + 1 & \text{if } q < x \end{cases}$$

(We can think of $\delta(x,q)$ as the distance from x to q but with "wrap-around". As an application, imagine we want to build L hospitals (again!) on a one-way loop, but this time, minimizing a sum rather than a max...)

In the following example, $L = 3$, and a feasible solution Q is drawn in red, where the cost is the sum of the lengths of the blue line segments.



(a) (70 pts) First describe an $O(n^3 \log L)$-time dynamic programming algorithm for this problem. Include the following steps: (a) first define your subproblems precisely, (b) then derive the recursive formula (including base cases) with brief justifications (no need for long correctness proofs), (c) specify a valid evaluation order, and (d) analyze the running time. 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 the optimal subset itself.

(b) (20 pts) Let $x_1, \ldots, x_n$ be the numbers of S in increasing order. For $i \leq j$, define

$$d(i,j) = \sum_{k=i}^{j-1} (x_j - x_k).$$

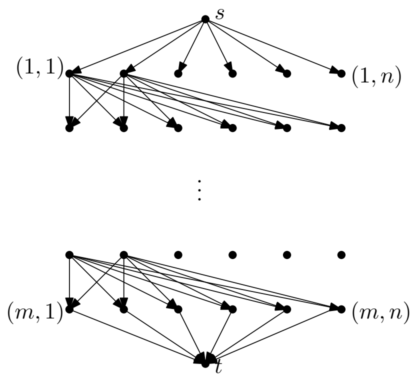
Prove that d satisfies the concave Monge property, i.e., for any $i \leq i' \leq j \leq j'$, we have $d(i,j) + d(i',j') \leq d(i,j') + d(i',j)$.

(c) (10 pts) Using results stated in class, argue that the running time of your algorithm from (a) can be improved to $O(n^2 \log L)$.

Problem 3.3: Consider the following weighted directed acyclic graph (DAG) G:

- The vertices are $\{(i,j) : i \in \{1,\ldots,m\},\ j \in \{1,\ldots,n\}\} \cup \{s,t\}$.
- For each $i \in \{1,\ldots,m-1\}$ and $j, j' \in \{1,\ldots,n\}$, there is an edge from (i,j) to $(i+1,j')$ with weight $f(i,j,j')$, for some function f that can be evaluated in constant time.
- For each $j \in \{1,\ldots,n\}$, there is an edge from s to $(1,j)$ with weight g_j, and an edge from (m,j) to t with weight h_j, for some given values $g_1,\ldots,g_n,h_1,\ldots,h_n$.

We want to compute a shortest path from s to t in G. Here, we really want to output an optimal path, not just the optimal total weight.



Since G has $O(mn)$ vertices and $O(mn^2)$ edges, we can apply a standard single-source shortest path algorithm in DAGs to solve this problem in $O(mn^2)$ time, using $O(mn)$ space.

Describe a more space-efficient algorithm to solve this problem, using space close to $O(m+n)$, up to some logarithmic factors. (Note that we can't afford to build and store the entire graph G explicitly, but we can call the function f whenever we need.) The running time should remain close to $O(mn^2)$, up to some logarithmic factors.

[Hint: combine DP with divide-and-conquer to save space. Unlike in the space-saving method for LCS from class, we may not be able to halve both the number of rows and the number of columns when we recurse. . .]