HW 6

Due on **Wednesday, October 25, 2017 at 10am**

CS/ECE 374: Algorithms & Models of Computation, Fall 2017 Version: **1.41**

Submission instructions as in previous homeworks.

Unless stated otherwise, for all questions in this homework involving dynamic programming, you need to provide a solution with explicit memoization.

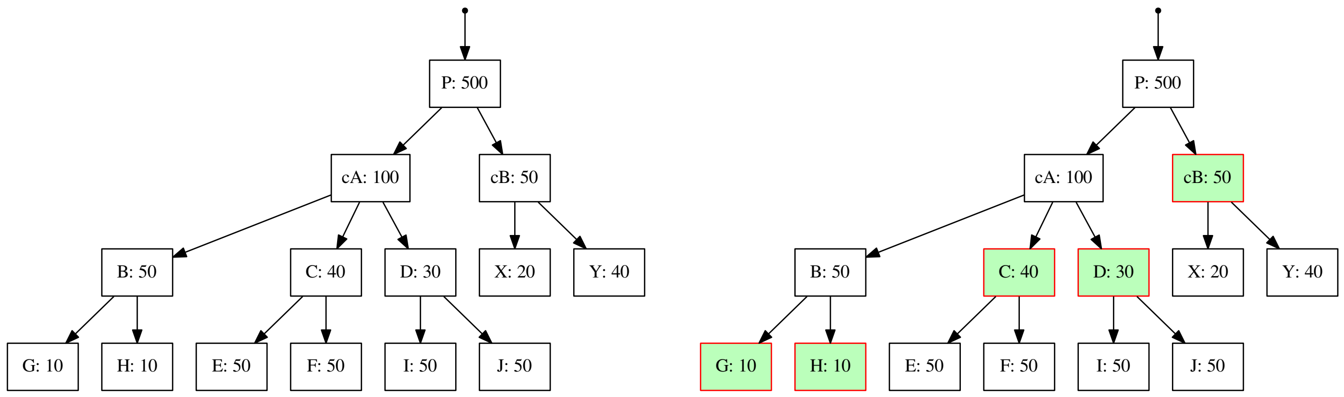
1 (100 PTS.) No one expects the Spanish inquisition.

(This problem is somewhat easier than the usual homeworks problems.)

It is 1492 in Spain (the Jews were just kicked out of Spain, and Columbus just landed in the Bahamas [an interesting year]) – unfortunately, the people in the town that you live in decided that you are a witch, and it is only a question of time till the inquisition would come to investigate[1]. You can avoid the investigation by paying bribes.

Fortunately, you got your hands on the organizational chart of the inquisition, which is a tree. The leafs are the investigators, and the internal nodes are the supervisors. A person that corresponds to a node u of this tree T, can be bought off by paying b_u Maravedís (the Spanish money at the time), and then they would suppress any investigation by anybody in their subtree.

Given a tree T as above with n nodes, the problem is to compute the set of people that you should bribe, so that all possible investigations are suppressed, and that the total sum paid is minimal. As example, consider the following tree (on the left, and a solution on the right).



1.A. (30 PTS.) State the recursive formula (or formulas if needed) for computing the desired quantity – and explain the logic behind it (in one sentence). What is the running time of your algorithm? (The running time should be as good as possible, as usual.)

Note, that you need only to compute the price of the optimal solution – there is no need to output the solution itself.

1.B. (70 PTS.) Describe a dynamic program to solve the problem. Your algorithm should be implemented using explicit memoization, and should not use recursion. Analyze the running time of your algorithm (you need to provide pseudo-code). What is the running time and space of your algorithm?

Here, assume the input is given in an array $Z[1 \,..\, n]$ of nodes. The first entry (i.e., $Z[1]$) is the root. Every node has a field b specifying the bribe, and a field ℓ specifying the number of children it has, and an array/list C of the indices of the children nodes. You can assume that for a node $Z[i]$, all its children are stored in locations in Z with index strictly larger than i.

(Again, no need to output the set realizing the solution – just the price of this solution – you should think about how to print out the optimal solution.)

[1]For once, somebody is expecting the Spanish inquisition: `https://www.youtube.com/watch?v=QqreRufrkxM`.

1.C. (**Not for submission**, and we are not going to provide a solution for this part.) What if you need to bribe k people on each path from the root to a leaf of the tree (think about the case $k = 2$ or $k = 3$). Indeed, if each person getting bribed knows that you can manage without them, the amount they are going to ask for is going to be significantly lower. How do you compute the minimum cost set of people to bribe in this case? What is the resulting running time?

2 (100 PTS.) Fishing for staircases.

A DAG is a directed graph with no cycles. Given a DAG G with n vertices and m edges, one can compute in $O(n+m)$ time an ordering of its vertices $v_1, \ldots v_n$, such that if there is an edges $(v_i, v_j) \in \mathsf{E}(\mathsf{G})$, then $i < j$. This is known as ***topological sort*** (or ***topological ordering***) of G, and you can assume you are given a black box that can implement this operation in $O(n+m)$ time. A vertex in a DAG is a ***sink*** if it has no outgoing edges.

2.A. (25 PTS.) The ***value*** of a vertex v in a DAG G, is the length of the longest path in DAG that starts in v. Describe a linear time algorithm (in the number of edges and vertices of G) that computes for all the vertices in G their value.

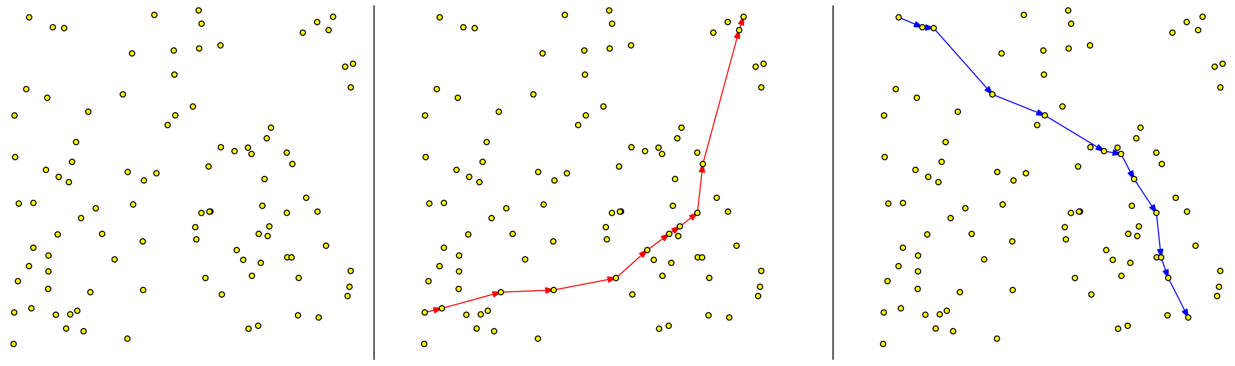
2.B. (25 PTS.) Prove that if two vertices $u, v \in \mathsf{V}(\mathsf{G})$ have the same value (again, G is a DAG), then the edges (u, v) and (v, u) are not in G.

2.C. (25 PTS.) Using (B), prove that in any DAG G with n vertices, for any k, either there is a path of length k, or there is a set X of $\lfloor n/k \rfloor$ vertices in G that is ***autonomous***; that is, there is no edge between any pair of vertices of X.

2.D. (25 PTS.) Consider a set of P of n points in the plane. The points of P are in general position – no two points have the same x or y coordinates. Consider a sequence S of points $p_1, p_2, \ldots, p_k$ of P, where $p_i = (x_i, y_i)$, for $i = 1, \ldots, k$. The sequence S is ***staircase***, if either

- for all $i = 1, \ldots, k-1$, we have $x_i < x_{i+1}$ and $y_i < y_{i+1}$, or
- for all $i = 1, \ldots, k-1$, we have $x_i < x_{i+1}$ and $y_i > y_{i+1}$.

Prove using (C) that there is always a staircase of length $\lfloor \sqrt{n} \rfloor$ in P. Describe an algorithm, as fast as possible, that computes the longest staircase in P.

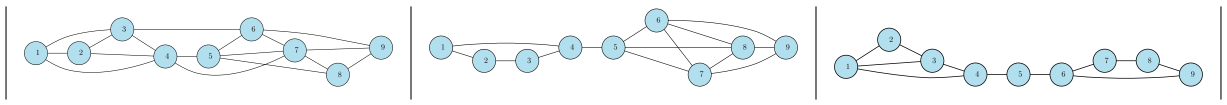


2.E. **(Harder + not for submission.)** Using the algorithm of (D), describe a polynomial time algorithm that decomposes P into a set of $O(\sqrt{n})$ disjoint staircases. Prove the correctness of your algorithm.

3 (100 PTS.) Independence in orderly graphs.

For a graph G with n vertices, assigning each vertex of G a unique number in $\{1, \ldots, n\}$ is a ***numbering*** of its vertices.

A graph G with a numbering $\mathsf{V}(\mathsf{G}) = \{1, \ldots, n\}$ is ***k-orderly***, if for every edge $ij \in \mathsf{E}(\mathsf{G})$, we have that $-k \leq i - j \leq k$ (note, that it is not true that if $|i - j| \leq k$ then ij must be an edge in the graph!). A different numbering of the vertices of G might change the value of k for which G is k-orderly. Here are a few examples of a 3-orderly graph:



3.A. (20 PTS.) For any given value of k, show an example of a graph that is not k-orderly for any numbering of its vertices. Prove the correctness of your example. For credit your graph should have a minimal number of edges (as a function of k).

3.B. (80 PTS.) Given a k-orderly graph G (here you are given the numbering of the vertices, and the value k [or, you can compute it directly from the numbering]), consider the problem of computing the largest independent set of G. As a reminder, a set of vertices $X \subseteq \mathsf{V}(\mathsf{G})$ is ***independent***, if no pair of vertices of X are connected by an edge of G.

Provide an algorithm, as fast as possible, for computing the size of the largest independent set in G. What is the dependency of the running time of your algorithm on the parameter k?

In particular, for credit, your solution for this problem should have polynomial time for k which is a constant. For full credit, the running time of your algorithm should be $O(f(k)n)$, where $f(k)$ is some (potentially large) function of k.

For this question you can use implicit memoization.

Hint: (A) Think about the vertices as ordered from left to right as above. Start with $k = 1$. Then, solve the problem for $k = 2, 3, 4, \ldots$. Hopefully, by the time you hit $k = 5$ you would be able to describe an algorithm for the general case. Think about defining a recursive function for this problem that has $\approx k$ parameters.