CS 161 (Stanford, Winter 2023) Section 6

1 Dynamic Programming: Longest Increasing Subsequence

In this exercise we'll practice designing and analyzing dynamic programming algorithms. Let A be an array of length n containing real numbers. A *longest increasing subsequence* (LIS) of A is a sequence $0 \leq i_0 < i_1 < \cdots < i_{\ell-1} < n$ so that $A[i_0] < A[i_1] < \cdots < A[i_{\ell-1}]$, so that ℓ is as large as possible. For example, if $A = [6, 3, 2, 5, 6, 4, 8]$, then a LIS is $i_0 = 1$, $i_1 = 3$, $i_2 = 4$, $i_3 = 6$ corresponding to the subsequence $3, 5, 6, 8$. (Notice that a longest increasing subsequence doesn't need to be unique).

In the following parts, we'll walk through the recipe that we saw in class for coming up with DP algorithms to develop an $O(n^2)$-time algorithm for finding an LIS.

1. **(Identify optimal sub-structure and a recursive relationship).** We'll come up with the sub-problems and recursive relationship for you, although you will have to justify it. Let $D[i]$ be the length of the longest increasing subsequence of $[A[0], \ldots, A[i]]$ that ends on $A[i]$. Explain why

$$D[i] = \max\left(\{D[k] + 1 \,:\, 0 \leq k < i, A[k] < A[i]\} \cup \{1\}\right).$$

2. **(Develop a DP algorithm to find the value of the optimal solution)** Use the relationship above to design a dynamic programming algorithm returns the *length* of the longest increasing subsequence. Your algorithm should run in time $O(n^2)$ and should fill in the array D defined above.

3. **(Adapt your DP algorithm to return the optimal solution)** Adapt your algorithm above to return an actual LIS instead of its length. Your algorithm should run in time $O(n^2)$.

Note: Actually, there is an $O(n \log n)$-time algorithm to find an LIS, which is faster than the DP solution in this exercise!

2 Housing Layout

You own $n \geq 1$ consecutive plots of lands that you can build on, and you want to build a number of houses on these plots, with each plot having at most one house. However, due to some strange laws in your city, you cannot build two houses on two consecutive plots of lands. (The other plots of lands will just be wasted, unused.)

Each plot of land is different, so building the house in the right plots is important. You have estimated the profit you would get from building a house on each plot of land to be

$p[1], \ldots, p[n]$, where $p[i]$ is a positive integer representing the profit, in dollars, you would get from building a house on the i^{th} plot of land.

Example if the input was $p = [21, 4, 6, 20, 2, 5]$, then you should build houses in the pattern

and you would profit by $21 + 20 + 5 = 46$ dollars. You would **not** be allowed to build houses in the pattern

because there are two houses next to each other.

In this question, you will design a dynamic programming algorithm which runs in time $O(n)$ which takes as input the array p and returns the maximum profit possible given p. Do this by answering the two parts below.

2.1 Sub-problems

What sub-problems will you use in your dynamic programming algorithm? What is the recursive relationship which is satisfied between the sub-problems? What are the base cases for this recursive relationship?

2.2 The algorithm

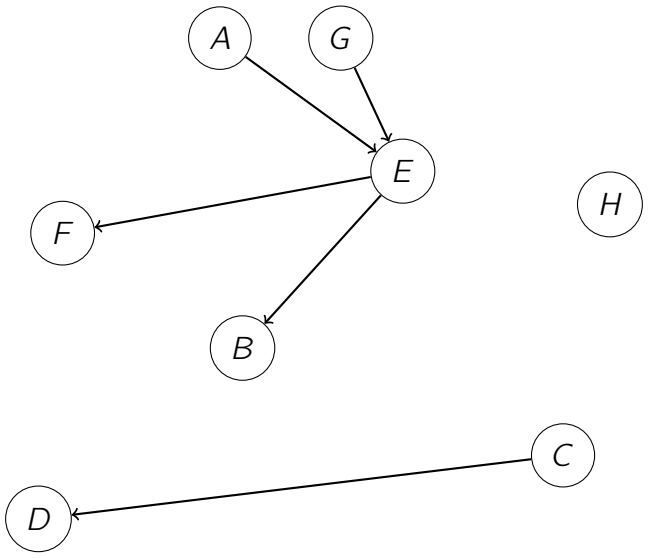
Write pseudocode for your algorithm. Your algorithm should take as input the array p, and return a single number which is the maximum profit possible. Your algorithm does not need to output the optimal way to build houses.

3 Hyperlinks Can Go Backward?

On the internet, many pages have links pointing to other pages, but sometimes it's not possible to reach a site you were on previously without clicking the "back" button in your browser. Elgoog can model their website as a directed graph G with n pages, and each page has some number of links to other pages. There are a total of m links over all these pages. Currently, it's not possible to get from some pages to some other pages without clicking the back button, and sometimes not possible at all! They want your help in designing an algorithm which can output the **minimum** total number of extra links they need to add so that every page is reachable from every other page.

3.1 Example Graph

In the given graph, find the minimum number of links that Elgoog must add.



3.2 Proof 1

Suppose G is a directed, acyclic graph (DAG) with at least two vertices and is connected in the undirected sense (for example, ABEFG are connected and CD are connected). Define $S \subseteq V$ as the set of *source* nodes: those vertices with no incoming edges and $T \subseteq V$ as the set of *sink* nodes: those vertices with no outgoing edges. **Prove** that the minimum number of links which have to be added is $\max(|S|, |T|)$.

3.3 Proof 2

Suppose G is a DAG which is not necessarily connected. **Prove** an expression for the minimum number of links which must be added.

Hint: You will have to case on whether G has exactly one vertex or not.

3.4 Algo Design

Write an algorithm which runs in time $O(m+n)$ and computes the minimum number of links to be added in the case when G is a general directed graph instead of just a DAG.