# Announcements

PLEASE fill out course reviews before Sunday

I made moderate changes for this version; I'm going to make major changes before I teach this course again. Your feedback will help!

HW6 will be back soon

HW7 won't be back before the final ends, but we will release solutions by sometime Monday.

Fill out the poll everywhere for Activity Credit!
Go to pollev.com/cse417 and login with your UW identity

# Longest Common Subsequence

Given two arrays $A_1$ and $A_2$ find the length of the longest subsequence that appears on both $A_1$ and $A_2$.

For example, if $A_1$ is $a, b, c, d$

And $A_2$ is $a, c, b, a, d$

The correct answer is $3$ (corresponding to $a, b, d$ or $a, c, d$).

Notice the subsequences are in the order of the original array.

What's one step?

Or said differently, if you were going to try to write a recursive version, what would you consider checking?

# DP Practice

The sequence C = c1, . . . , ck is a non-adjacent subsequence of A = a1, . . . , an, if C can be formed by selecting non-adjacent elements of A, (in order). The non-adjacent LCS problem is given sequences A and B, find a maximum length sequence C which is a non-adjacent subsequence of both A and B.

This problem can be solved with dynamic programming. Give a recurrence that is the basis for a dynamic programming algorithm. You should also give the appropriate base cases, and explain why your recurrence is correct.

# Maximum Subarray Sum

We saw an $O(n \log n)$ divide and conquer algorithm.

Can we do better with DP?

Given: Array $A[]$

Output: $i, j$ such that $A[i] + A[i+1] + \cdots + A[j]$ is maximized.

Is it enough to know OPT()?

# Remember maximum Subarray Sum?

$$INCLUDE(i) = \begin{cases} \max\{A[i], A[i] + INLCUDE(i-1)\} & \text{if } i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$OPT(i) = \begin{cases} \max\{INCLUDE(i), OPT(i-1)\} & \text{if } i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

If we include $i$, the subarray must be either just $i$ or also include $i-1$.

Overall, we might or might not include $i$. If we don't include $i$, we only have access to elements $i-1$ and before. If we do, we want $INCLUDE(i)$ by definition.

# Remember Maximum Subarray Sum?

Long subarrays only: Describe and analyze an algorithm that finds a contiguous subarray of $A$ of length at least $X$ (i.e. including at least $X$ elements) that has the largest sum.

You may assume $X \leq n$.

# Reductions

1. Figure out what you're reducing from and to

The known NP-hard problem is the source, the new problem is the target.

2. Understand both your input types (are they both graphs? Is one a graph and the other a list of variables and constraints?)

3. Understand the "certificates" of each – what are you looking for?

# Actually designing the reduction

Your goal is to transform the certificate of the source problem into the certificate of the target problem

AND to not create any “false positive” certificates.

You are given a directed graph G = (V, E) with weights we on its edges e ∈ E. The weights can be negative or positive. The Zero-Weight-Cycle Problem is to decide if there is a simple cycle in G so that the sum of the edge weights on this cycle is exactly 0. Prove that the Zero-Weight-Cycle problem is NP-Complete. (Hint: Hamiltonian PATH)

Thinking under pressure.

What is being CS academia/algorithm-researcher like? Do you just sit there staring at questions and reading books until you figure out an algorithm?

Chess moves are a problem that's beyond NP, but now we're able to develop AI that plays the game better than humans can. Does that mean that Non-NP = NP? What are the implications of that?