ANNOUNCEMENTS

- Homework 1 due on Monday Sep 10, 11:59 PM
- Video issues last class
- Comment on notes – JeffE’s material
LAST CLASS

- Dynamic programming
  - Recursive algorithms – observe that same sub-problems solved many times
  - **Key idea:** avoid re-solving by storing answers!
  - How many “different problem instances” arise?
  - Examples: subset sum and coin change problems

\[
\begin{array}{c}
\text{denominations of coins:}
\end{array}
\]
OVERLAP IN RECURSIVE CALLS

A = {1, 2, 3, 5, 7, 11, 14, 15}, S = 20

Use first element
A' = {2, 3, 5, 7, 11, 14, 15}, S' = 19

Don't use first element
A'' = {5, 7, 11, 14, 15}, S' = 17

Key observation: every sub-problem has a suffix of A as the array, and some target sum in the range [0,...,S]. Thus only $n(S+1)$ sub-problems!
Today's Plan

- Conclude DP – a couple more examples from a high level
- Greedy algorithms
RECAP: DYNAMIC PROGRAMMING

- Design recursive algorithm
- Observe that number of distinct sub-problems is “small”
- Store answers! (memorize)
- Top-down vs. Bottom-up
- Challenge in using DP: formalizing sub-problems

(more in HW2)
**Problem:** given a sequence of numbers A[0], A[1], ..., A[n-1], find longest "sub-sequence" whose elements are increasing

▶ Natural recursive algorithm?

→ Split into cases based on whether the first element is in the sub-sequence or not.

Recursive call on remaining list.
procedure

\[ \text{MIS'}(A[0], A[1], \ldots, A[n-1]) : \text{find longest increasing sub-seq. that includes } A[0]. \]

Answer to original problem =

\[
\max_{\text{all } i} \text{MIS'}(A[i], A[i+1], \ldots, A[n-1]).
\]

\[ 4, 1, 1, 2, 3 \quad \text{MIS'}(4, 1, 2, 3) = 1 \]

\[ 1, 2, 3 \quad \text{MIS'}(1, 2, 3) = 3 \]

\[ (2, 3) = 2 \]

Goal: write recursion for \( \text{MIS'}(\quad) \).

\[ \text{MIS'}(A[0], A[1], \ldots, A[n-1]) ; \]

\[ \forall i \geq 1 \quad \text{if } A[i] > A[0], \text{ find } \]

\[ \text{MIS'}(A[i], A[i+1], \ldots, A[n-1]) \]

Output: \[ 1 + \max \]

Correctness: Take any subseq. that includes \( A[0] \); let it be \( A[0], A[i_1], A[i_2], \ldots, A[i_r] \).

We know that \( A[i_r] > A[0] \), so it will be considered in the recursive procedure.
Now, we can argue inductively that

\[ 1 + \text{MIS}'[A[i], A[i+1], \ldots] \geq r+1 \]

\[ 1, 4, -1, 0, 2, 3 \]

\[ 1, 4, 0, 2, 3 \]

\[ \frac{5}{4} \]

\[ 4 \]  

(*) Think about why student's suggestion does not work.

bseg.

\[ M_1 = 0, \ldots, -[n-1] \]
#(SUB-PROBLEMS) AND RUN TIME

\[ \text{# sub-problems} = n \]

\[ \text{Running time} = n^2. \]

Moral: \( \rightarrow \) defining MIS' is key (otherwise procedure isn't correct)

\( \rightarrow \) Should be careful about defining the right sub-problem
**Problem:** given undirected graph with edge wts, find a path of least total weight that visits every vertex precisely once and returns to start.
TRAVELING SALESMAN PROBLEM

Problem: given undirected graph with edge wts, find a path of least total weight that visits every vertex precisely once and returns to start.

Trivial algorithm? check all permutations, see which one has least cost

\[ \left( \frac{n}{2} \right)^{n/2} \approx 16 \approx n! = \left[ \frac{n(n-1)(n-2) \ldots 1} {4^n} \right] \cdot n \]

Dynamic prog algorithm: \( 2^n \cdot n^2 \)

Key: define sub-problem very carefully.
TRAVELING SALESMAN PROBLEM

**Problem:** given undirected graph with edge wts, find a path of least total weight that visits every vertex precisely once and returns to start.

- Trivial algorithm?
- Many heuristics for geometric cases
RECURSIVE ALGORITHM
SUB-PROBLEMS, RUNNING TIME
SUB-PROBLEMS, RUNNING TIME
CAN WE DO BETTER?

▸ Likely not – NP hard problem

▸ We will see that finding “approximately good” solution possible!
GREEDY ALGORITHMS

- Greedy == “myopic”

- Algorithm makes sequence of decisions – chooses best at each step, without taking future into account

→ Greedy algorithms almost never work!

→ Often, give insights about how to solve. (Sometimes, they actually work!)
**EXAMPLE**

**Scheduling:** Given set of jobs with processing times $p_1, p_2, \ldots, p_n$, find an order of processing so that sum of completion times is minimized.

1, 2, 4, 3

$C_1 = 1$; $C_2 = 3$; $C_3 = 7$; $C_4 = 10$

Intuitive idea: Do jobs with small $p_i$ first!

- Process in order of increasing $p_i$ (greedy algorithm)
NATURAL HEURISTIC?
CORRECTNESS

Claim: Processing jobs in this order is the best possible.

Proof: Suppose the optimal order is \( i_1, i_2, \ldots, i_n \). Suppose we swap jobs \( i_j \) and \( i_k \), where \( j < k \). Similar arguments show that it could not have been optimal.
CORRECTNESS — PROOF 2
**EXAMPLE — MATCHING**

**Matching:** suppose we have $n$ children, $n$ gifts and “happiness” values $H_{ij}$. Assign gifts to children to maximize “total happiness”
For each child, assign gift from the remaining gifts that has the most value.

Greedy does not always find best solution.
CORRECTNESS
CORRECTNESS