Lecture #11
Dynamic Programming (DP)

General Approach to many problems:
Solve a big problem by breaking it into smaller subproblems, solve subproblems in order from “small” to “large”

Isn’t this recursion?

```plaintext
func Fib(n)
  if n ≤ 1 return n
  else Fib(n-1) + Fib(n-2)
  cost = O(Fib(n)) ~ 1.6^n
  fix-memoization

Fib(0) = 0, Fib(1) = 1
for i = 2 to n
  Fib(i) = Fib(i-1) + Fib(i-2)
  cost = O(n)
```
Dynamic Programming - Examples
• Shortest Path in DAG
• Longest Increasing Subsequence
  5, 2, 8, 6, 3, 6, 9, 7  \text{LIS length} = 4 \text{vertices}
• Edit Distance = fewest # edits to change one string to another (delete, insert, substitute)
  • Spell checking
  • How similar is my DNA to your DNA
• Cheating detection
• Knapsack: given \( n \) items with weights \( w_1, \ldots, w_n \)
  values \( v_1, \ldots, v_n \), total weight limit = \( W \)
  how to choose subset \( S \) of items with \( \max_{i \in S} v_i \) with \( \sum_{i \in S} w_i \leq W \)
• All-pairs-Shortest-Paths (better than \( n \times \text{Bellman-Ford} \))
• TSP - Traveling Salesperson Problem...
Shortest Paths in DAGs - DP point of view

- Given $G(V, E)$, $w(e) \in Z$, find shortest path from $s \in V$ to $v \in V$ (negative $w(e)$ OK, no cycles)

- Approach
  1) Define subproblems: find shortest path from $s$ to $v'$ for $v'$ "closer to" $s$ than $v$
  2) Show how to solve a problem given solutions to subproblems: $\text{dist}(v) = \min_{u : (u, v) \in E} \text{dist}(u) + w(u, v)$
  3) Base case: $\text{dist}(s) = 0$, $\text{dist}(w) = \infty$ if $w$ is a source
  4) Choose order to solve subproblems - topologically sort vertices starting at $s$

What if we wanted longest path?

$w \rightarrow -w$, $\min \rightarrow \max$
**Longest Increasing Subsequence (LIS)**

Given an unsorted sequence of numbers \( x_1, x_2, \ldots, x_n \) find LIS

1. **Subproblems**: \( f(i) = \text{LIS in } x_1, \ldots, x_i \)

2. **Solve using subproblems**

   \[
   L(i) = \max(1, \max_{j < i \land x_i \leq x_j}(L(j) + 1))
   \]

3. **Base**: \( L(1) = 1 \)

4. **Order to solve**: increasing \( i \)

Red edges form a DAG \( x_i \rightarrow x_j \) if \( x_i \leq x_j \): Longest Path in DAG

Cost \( T = \sum_{i=1}^{n} O(i) = O(n^2) \)
Edit Distance

• How similar are 2 strings \([x_1, \ldots, x_n], [y_1, \ldots, y_m]\)?
• How many “edits” needed to change \(x\) to \(y\)?

Edit means insert, delete or substitute a char.

\[
snowy \rightarrow snowy
\]
\[
sunny \rightarrow sunny
\]
\[
\text{\#edits} = 5
\]

Motivation:
- spell-checking - suggest fixes
- DNA matching
- cheat detection
- spam filtering
Edit Distance (ED) between $x = [x_1, ..., x_n]$ and $y = [y_1, ..., y_m]$

1) Subproblems: for all $1 \leq i \leq n$, $1 \leq j \leq m$
$$f(i, j) = ED([x_1, ..., x_i], [y_1, ..., y_j])$$

2) Look at last char in optimal alignment, could be:
- remove $x_i$
- insert $y_j$

Cost = $f(i-1, j) + 1$
$$f(i, j) = \min(f(i-1, j) + 1, f(i, j-1) + 1, f(i-1, j-1) + \delta(x_i, y_j))$$

3) Base case: $f(i, 0) = i$ (deletes), $f(0, j) = j$ (insert)

4) Order: for $f(i, j)$ need $f(i-1, j)$, $f(i, j-1)$, $f(i-1, j-1)$
Order of Subproblems for ED

DAG for computing \( f(i, j) \)

\[
f(i, j) = \min(f(i-1, j) + 1, \quad f(i, j-1) + 1, \quad f(i-1, j-1) + d_{ij})
\]

rowwise, columnwise

Cost = \( O(n \cdot m) \)

Memory = \( O(\min(m \text{ rowwise}, n \text{ columnwise})) \)

Implicitly shortest path in DAG with edge weight

\[
\delta_{ij}
\]
What about $O(m \cdot n)$ cost if $m, n = O(10^9)$?

Why? Sequencing DNA any 2 people have 99.9% same DNA

\[ E_D = O(.001 \cdot 10^a) = O(10^6) \]

\[ \text{only compute } f(i, j) \text{ for } (i-j) = O(10^6) \]

\[ \Rightarrow \text{cost } (10^0 \cdot 10^9) = O(10^{15}) \]

Smith-Waterman
Needleman-Wunsch
MetaHipMer
Knapsack Problem

Suppose you are deciding how to invest $W$ to invest $n$ investments likely payoffs $v_1, \ldots, v_n$. Which should you pick?

You have $n$ jewels, values $v_1, \ldots, v_n$ weights $w_1, \ldots, w_n$. You have to choose among $n\in \{1, 2, \ldots, n\}$.

Set $i \in \{1, 2, \ldots, n\}$ to maximize $\sum_{i \in S} v_i$ subject to $\sum_{i \in S} w_i \leq W$. 
Does a greedy algorithm work?

\[ W = 20 \quad w_1 = 11 \quad v_i = 15 \]
\[ w_2 = 10 \quad v_2 = 8 \]
\[ w_3 = 10 \quad v_3 = 8 \]

\[ W = 20 \quad v_2 + v_3 = 16 \]

Greedy: choose \( i \) to maximize \( \frac{v_i}{w_i} \) → \( v_i = 15 \)
Knapsack by Dynamic Programming

1) Subproblems: \( f(i, u) = \max \) value packing subset of \( 1, \ldots, i \) max weight \( u \leq W \)

2) Solve \( f(i, u) = \begin{cases} \text{if } w_i > u \quad \text{... } w_i \text{ too heavy} \\ f(i-1, u) \\ \text{else } \quad \text{could pack } w_i \\ \max( f(i-1, u), v_i + f(i-1, u-w_i) ) \\ \text{don't pack } w_i \quad \text{pack } w_i \end{cases} \)

3) Base Case: \( f(0, u) = 0 \), \( f(i, 0) = 0 \)

4) Order: for \( i = 1 \) to \( n \)
   for \( u = 1 \) to \( W \)
   \( f(i, u) = \ldots \) step 2
Cost of Knapsack

For $i = 0$ to $n$, $f(i, 0) = 0$; for $u = 0$ to $W$, $f(0, u) = 0$

For $i = 1$ to $n$

For $u = 1$ to $W$

if $w_i > u$

$f(i, u) = f(i-1, u)$

else

$f(i, u) = \max\{f(i-1, u), f(i-1, u-w_i)+v_i\}$

end if

Cost $= O(n \cdot W)$: Polynomial Time?

In “size of input” $v_1, \ldots, v_n, w_1, \ldots, w_m, W$

Size of input $O(n \log \max v_i + \log_2 W)$

$n \cdot W$ can be exponentially larger

Knapsack is $NP$-complete Chap 8