LECTURE #12
Last time:
• introduction to dynamic programming
• examples
  - shortest paths in a DAG
  - longest increasing subsequence
  - edit distance
  - knapsack (w/o repetition)

Today:
More examples of DP solutions:
- knapsack with repetition
- chain matrix multiplication
- all-pairs shortest paths
- traveling salesperson problem

\[ \approx \text{recursion + memoization (with explicit solving order)} \]

DP Recipe:
1. define problems
2. set boundaries
3. give recurrence
4. specify order (and give time)
Knapsack With Repetition

Given a maximum weight $W$ and value-weight pairs $((v_1, w_1), \ldots, (v_n, w_n))$, output a multi-set of weight $\leq W$ of highest value.

(OK to pick the same item more than once)

- Problems: $K(w) := \text{max value achievable with capacity } \leq w$
- Boundaries: $K(0) = 0$
- Recurrence: $K(w) := \max_{i=1, \ldots, n} \{ K(w-w_i) + v_i | w_i \leq w \}$
- Efficiency: - Solve in increasing weight $w$: $K(0), K(1), \ldots, K(W)$
  - Each computation takes $O(n)$ time
  - Total time is $O(n \cdot W)$

The algorithm is only weakly polynomial time.

Polynomial time would have been poly $(n, \log W)$, but we don't expect this because the problem is NP-complete.
Chain Matrix Multiplication

Multiplying \(x\)-by-\(y\) matrix \(A\) and \(y\)-by-\(z\) matrix \(B\) takes \(xyz\) ops. (We have also seen how to do better but this is not important for now.)

Today: find strategy to multiply matrices \(A_1, A_2, \ldots, A_n\) as cheaply as possible.

Example: \(50\)-by-\(1\) \(A\), \(1\)-by-\(50\) \(B\), \(50\)-by-\(1\) \(C\). The result is \(50\)-by-\(1\) \(R\).

• Option 1: \((A \cdot B) \cdot C = D \cdot C\)
  \(50 \cdot 50 + 50 \cdot 1 = 5000\)

• Option 2: \(A \cdot (B \cdot C) = A \cdot E\)
  \(1 \cdot 50 + 50 \cdot 1 = 100\)

The association order matters!
In general, the goal is to find the optimal association order for

\[ A_1, A_2, \ldots, A_n \]

where \( m_0 \)-by-\( m_1 \), \( m_1 \)-by-\( m_2 \), \ldots, \( m_{n-1} \)-by-\( m_n \).

**Input:** list of positive integers \( m_0, m_1, \ldots, m_n \) representing matrix dimensions (the matrices themselves don't matter).

**Output:** parenthesisation of the list (a tree on it).

We solve the problem via dynamic programming:

- **Subproblems:**
  \[ C(i,k) := \text{optimal cost for subsequence } m_i, \ldots, m_k \]
  [multiplying \( A_{i+1} \ldots A_k \)]

- **Boundaries:**
  \[ C(0,1) = C(1,2) = \ldots = C(n-1,n) = 0 \]

- **Recurrence:**
  \[ C(i,k) := \min_{i<j<k} \left\{ C(i,j) + C(j,k) + m_i m_j m_k \right\} \]

- **Cost:** \( O(n^2) \) subproblems, computing each is \( O(n) \) → \( O(n^3) \) cost to find \( C(0,n) \).
All-Pairs Shortest Paths

Given graph $G = (V,E)$ and lengths $l: E \to \mathbb{Z}$, find $\{\text{dist}(u,v)\}_{u,v \in V}$.

Idea: Run Bellman-Ford for every possible source: $|V| \cdot O(|V| \cdot |E|) = O(|V|^2 \cdot |E|)$.

Better: $O(|V|^3)$ via dynamic programming ($|E| \geq |V| - 1$ if $G$ is connected)

Floyd-Warshall algorithm

- The subproblems are:
  
  $$d(u,v,i) = \text{shortest path from } u \text{ to } v \text{ using intermediate nodes in } \{1, 2, \ldots, i\}$$

- Boundaries are:
  
  $$d(u,v,0) = \begin{cases} 
  \text{if } (u,v) \in E: l(u,v) \\
  \text{if } (u,v) \notin E: \infty
  \end{cases}$$

- The recurrence is:
  
  $$d(u,v,i) = \min \{ d(u,v,i-1), d(u,i,i-1) + d(i,v,i-1) \}$$

- Efficiency:
  
  - init boundaries: $O(|V|^2)$
  - table has $|V|^3$ entries and filling each is $O(1)$
  - total cost is $O(|V|^3)$.  

Traveling Salesperson Problem

Given graph \( G = (V,E) \) and lengths \( l: E \to \mathbb{Z} \), output a shortest tour of \( G \).

**Straightforward approach:** try all tours.

Without loss of generality (WLOG) start at vertex 1. There are \( \leq (n-1)! \) tours. Evaluating a tour costs \( O(n) \).

\[ \Rightarrow O(n!) \approx O\left( \frac{n^n}{e^n} \right). \]

**Better:** \( O(2^n \cdot n^2) \) via dynamic programming (cannot expect poly-time algo because TSP is NP-complete)

- **The subproblems are:**
  
  \[ \text{answer is } \min_{j=2,...,n} \left\{ C(\{2,...,n\}, j) + l(j, 1) \right\} \]

  \[ C(S, j) = \text{shortest path from } 1 \text{ to } j \in S \text{ visiting each vertex in } S \subseteq \{2,...,n\} \text{ once} \]

- **Boundaries:** \( C(\{j\}, j) = l(1, j) \)

- **Recurrence:** \( C(S, j) = \min_{i \in S \setminus \{j\}} \left\{ C(S \setminus \{j\}, i) + l(i, j) \right\} \)

- **Efficiency:** compute in order of increasing \(|S|\)

  table has \( O(2^n \cdot n) \) entries and each takes \( O(n) \) to compute \( \Rightarrow O(2^n \cdot n) \).