Dynamic Programming

Divide-and-Conquer works when a problem can be solved by solving smaller problems of the same type.

In DP, we typically have to solve a more general problem.

"I have just one question: What is the table?"

- shortest s-t path
- ______ with at most K edges
- longest increasing subsequence
- edit distance between two strings
- knapsack
- matrix multiplication chain

Find shortest s-t path in $G = (V,E)$
Given a connected s-t path with edge lengths $l(i,j) \geq 0$.

**Observation:** Shortest s-t path P is also shortest s-u path for all u on P.

**New general problem:** shortest s-u paths for all u.

**Algorithm.**

Maintain tentative distances $T(u)$.

- $T(s) = 0$
- $T(u) = \infty \quad \forall \ u \neq s$
- $S = \emptyset$

- Let j be the vertex not in S with smallest $T(j)$.

- Update $T(u) = \min \{ T(u), T(j) + l(j,u) \}$

- $S = S \cup \{ j \}$ + u neighbors of j.

- Repeat till S = V.

Let u the vertex with $T(u)$ in the shortest path.
**Lemma.** At every step $T(j)$ is the shortest path distance for all $j \in S$

**Proof.** By induction. True for all but last $j$ by hypothesis. For $j$, any other path must exit $S$ and will be longer.

**Theorem.** Dijkstra's algorithm correctly finds shortest $st$ paths from source $s$ to all $t$.

---

Longest Increasing Subsequence (LIS)

$1, 4, -1, 0, 6, 2, 4, 5$

$L(i), L(2), \ldots, L(i), \ldots$

More general problem: LIS ending with $S(i)$

For all $i$.

$L(i) = \max \{ 1 + S(j) : j < i, S(j) < S(i) \}$
\[ J(c) = \max \{ 1 + J(2) \} \]

Report overall max.

Matrix chain multiply:
\[
\begin{array}{c}
M_1(\cdot)
\end{array}
\begin{array}{c}
M_2(\cdot)
\end{array}
\begin{array}{c}
M_3(\cdot)
\end{array}
\ldots
\begin{array}{c}
M_K(\cdot)
\end{array}
\]

What is the fastest way to do it?

Assume \( n \times n \) matrix multiplication takes \( O(n^3) \).

\[ \sum_{\text{all } k} \quad \text{min} \quad C[i, k] + C[k+1, j] + m_i \cdot m_k \cdot m_j \]

E.g. \((2 \times 6) \cdot (6 \times 10) \cdot (10 \times 6)\)

\[ 0 \quad \rightarrow \quad 2 \cdot 6 \cdot 10 + 2 \cdot 10 \cdot 6 \quad \square \]

\[ 2 \quad 0 \quad \rightarrow \quad 6 \cdot 10 \cdot 6 + 2 \cdot 6 \cdot 6 \]

More general problem: For every \( i \leq k \leq j \)
What is the cost of multiplying from \( i \)th matrix to \( j \)th matrix.

\[ C[i, j] = \min_{i \leq k \leq j} \ C[i, k] + C[k+1, j] + m_i \cdot m_k \cdot m_j \]
$i < k < j$

$C[i, i+1] = m_i m_{i+1} m_{i+2}.$

$\delta$

$\text{Trie} = O(n)$ per entry of table

$= O(n^3).$

---

**Edit distance**

RAIN $\leftrightarrow$ WIND

Can delete, insert or swap.

4 swaps?

Or


RAIN $\rightarrow$ WIND

More general problem: edit distance between any two given strings.
two prefixes of the given strings.

\[ a_i \ldots a_m \quad a_i \ldots a_i \]
\[ b_i \ldots b_n \quad b_i \ldots b_j \]

\[ \text{ED}(i,j) : \begin{cases} \text{can delete } a_i \\ \text{or insert } b_j \\ \text{or map } a_i, b_j \text{ if } a_i \neq b_j \end{cases} \]

\[ \text{ED}(i,j) = \min \left\{ 1 + \text{ED}(i-1,j), 1 + \text{ED}(i,j-1), \text{ED}(i-1,j-1) + 1_{a_i \neq b_j} \right\} \]

\[ \sqrt[i]{\text{ED}(i,j)} \]

\[ \text{cost} = O(1) \text{ per entry} \]
\[ = O(nm) \text{.} \]

is this best possible?

\underline{Knapsack}

\underline{Items 1, 2, \ldots, n}
Items: $1, 2, \ldots, n$

weights: $w_1, w_2, \ldots$

values: $v_1, v_2, \ldots$

Capacity: $B$

Pack maximum total value in knapsack of capacity $B$.

Version 1. Can use many copies of each item.

More general problem: Maximum value for each capacity $1, 2, \ldots, B$

$$T(b) = \max \{ 0, v_i + T(b-w_i), b \geq w_i \}$$

Version 2. Can use each item at most once.

More general problem: Max value for capacity $b$

using items $1, 2, \ldots, i$ only.

\[ \begin{array}{ccccccc}
1 & 2 & \cdots & b & \cdots & B \\
1 & 2 & \end{array} \]
\[ T(i, b) = \max_{i \text{ is used}} V_i + T(i-1, b-w_i) \quad \text{if } i \text{ is used} \]
\[ T(i-1, b) \quad \text{if } i \text{ is not used} \]

\[ \text{Time} = O(nB) \]

\[ \text{Shortest path using at most } k \text{ edges} \]

\[ T(u, k) : \text{shortest } s-u \text{ path using at most } k \text{ edges.} \]

\[ T(u, k) = \min \{ T(u, k-1), \ T(v, k-1) + l(v, u) \} \quad \forall v \]
Time = $O(n)$ per entry = $O(n^3)$ total
OR, more carefully $O(m)$ per $k = O(mn)$. 