Dynamic Programming
DPV Chapter 6, Part 2

Jim Royer

March 6, 2019
Optimal Substructure

A problem has **optimal substructure** when an optimal solution is made up of optimal solutions to its subproblems.

**Examples**

(a) Shortest paths in a graph.
(b) Making change.
(c) …

**Non-examples**

(a) Longest paths in a graph.
(b) Cheapest airline ticket from $A$ to $B$. 
Edit Distance, 1

Edit Operations:
- Insert a character
- Delete a character
- Substitute a character

The Edit Distance Problem
The edit distance between strings $x[1..m]$ and $y[1..n]$ = the minimal number of edit operations to change $x[1..m]$ to $y[1..n]$.

SNOWY $\xrightarrow{\text{insert}}$ SUNOWY $\xrightarrow{\text{subst}}$ SUNNWY $\xrightarrow{\text{del}}$ SUNNY

$\{S \quad N \quad O \quad W \quad Y\}$ edit distance as
$\{S \quad U \quad N \quad N \quad – \quad Y\}$ alignment

To solve this with dynamic programming, we have to figure out good subproblems.
Edit Distance, 2

The Edit Distance Problem
The \textit{edit distance} between strings $x[1..m]$ and $y[1..n]$
\hspace{2cm} = the minimal number of edit operations to change $x[1..m]$ to $y[1..n]$.

The $E(i, j)$ Problem
What is the edit distance of $x[1..i]$ and $y[1..j]$.

Goal: $E(m, n)$.
Strategy: Solve $E(i, j)$ for $i = 1, \ldots, m, j = 1, \ldots, n$.

$$E(i, j) = \begin{cases} 
\text{some combination of the solutions to smaller } E(i', j') \text{ problems} 
\end{cases}$$
The $E(i, j)$ Problem
What is the edit distance of $x[1..i]$ and $y[1..j]$.

Key Observation
In an optimal alignment for $E(i, j)$ the last column must look like:

```
<table>
<thead>
<tr>
<th>$x_i$</th>
<th>or</th>
<th>$-$</th>
<th>or</th>
<th>$x_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-$</td>
<td></td>
<td>$y_j$</td>
<td></td>
<td>$y_j$</td>
</tr>
</tbody>
</table>
```

So...?
Edit Distance, 4

The $E(i,j)$ Problem

What is the edit distance of $x[1..i]$ and $y[1..j]$ (where $i,j > 0$)?

\[
\begin{cases}
    \text{Case } x_i \rightarrow \_ : & \text{The cost is } 1 + E(i - 1, j).
    \\
    \text{Case } \_ \rightarrow y_j : & \text{The cost is } 1 + E(i, j - 1).
    \\
    \text{Case } x_i \downarrow y_j : & \\
    \quad \text{Subcase } x_i = y_j : & \text{The cost is } E(i - 1, j - 1).
    \\
    \quad \text{Subcase } x_i \neq y_j : & \text{The cost is } 1 + E(i - 1, j - 1).
\end{cases}
\]
The $E(i,j)$ Problem

What is the edit distance of $x[1..i]$ and $y[1..j]$ (where $i, j > 0$)?

$$
\begin{align*}
\text{Case } x_i - y_j: & \quad \text{The cost is } 1 + E(i-1,j). \\
\text{Case } - y_j: & \quad \text{The cost is } 1 + E(i,j-1).
\end{align*}
$$

$$
\begin{align*}
\text{Case } x_i \quad y_j: & \quad \text{Subcase } x_i = y_j: \quad \text{The cost is } E(i-1,j-1).
\quad \text{Subcase } x_i \neq y_j: \quad \text{The cost is } 1 + E(i-1,j-1).
\end{align*}
$$

$$
\therefore \quad E(i,j) = \min(1 + E(i-1,j), 1 + E(i,j-1), \text{diff}(i,j) + E(i-1,j-1))
$$

$$
\text{diff}(i,j) = \begin{cases} 
0, & \text{if } x_i = y_j; \\
1, & \text{otherwise.}
\end{cases}
$$
The $E(i,j)$ Problem
What is the edit distance of $x[1..i]$ and $y[1..j]$ (where $i,j > 0$)?

\[
\therefore E(i,j) = \min(1 + E(i-1,j), 1 + E(i,j-1), \text{diff}(i,j) + E(i-1,j-1))
\]

\[
\text{diff}(i,j) = \begin{cases} 
0, & \text{if } x_i = y_j; \\
1, & \text{otherwise.}
\end{cases}
\]

The base cases:

\[
E(i,0) = i. \quad E(0,j) = j.
\]

Why?
The $E(i, j)$ Problem

Compute $E(i, j) =$

the edit distance of $x[1..i]$ and $y[1..j]$.

```
function edcost(x[1..m], y[1..n])
  for i ← 0 to m do E[i, 0] ← i
  for j ← 1 to n do E[0, j] ← j
  for i ← 1 to m do
    for j ← 1 to n do
      E[i, j] ←
      min(1 + E[i - 1, j],
           1 + E[i, j - 1],
           diff(i, j) + E[i - 1, j - 1])
  return E[m, n]
```

Edit Distance, 7

\[ E(i, j) = \min(1 + E(i-1, j), 1 + E(i, j-1), \text{diff}(i, j) + E(i-1, j-1)) \]

→≡ insert    ↓≡ delete    \(\swarrow\)≡ copy/replace

\[
\begin{array}{cccccc}
\text{k} & \text{i} & \text{t} & \text{t} & \text{e} & \text{n} \\
\text{s} & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\text{i} & 1 & 1 & 2 & 3 & 4 & 5 & 6 \\
\text{t} & 2 & 2 & 1 & 2 & 3 & 4 & 5 \\
\text{t} & 3 & 3 & 2 & 1 & 2 & 3 & 4 \\
\text{i} & 4 & 4 & 3 & 2 & 1 & 2 & 3 \\
\text{n} & 5 & 5 & 4 & 3 & 2 & 2 & 3 \\
\end{array}
\]
Edit Distance, 7

\[ E(i, j) = \min(1 + E(i - 1, j), 1 + E(i, j - 1), \text{diff}(i, j) + E(i - 1, j - 1)) \]

→≡ insert    ↓≡ delete    ⊳≡ copy/replace

\[
\begin{array}{ccccccc}
& k & i & t & t & e & n \\
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
1 & 1 & 2 & 3 & 4 & 5 & 6 \\
2 & 2 & 1 & 2 & 3 & 4 & 5 \\
3 & 3 & 2 & 1 & 2 & 3 & 4 \\
4 & 4 & 3 & 2 & 1 & 2 & 3 \\
5 & 5 & 4 & 3 & 2 & 2 & 3 \\
6 & 6 & & & & & \\
\end{array}
\]
Edit Distance, 7

\[ E(i,j) = \min(1 + E(i-1,j), 1 + E(i,j-1), \text{diff}(i,j) + E(i-1,j-1)) \]

→ ≡ insert  ↓ ≡ delete  ↘ ≡ copy/replacement

<table>
<thead>
<tr>
<th></th>
<th>k</th>
<th>i</th>
<th>t</th>
<th>t</th>
<th>e</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>i</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>t</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>t</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>i</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>n</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>n</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
$$E(i, j) = \min(1 + E(i - 1, j), 1 + E(i, j - 1), \text{diff}(i, j) + E(i - 1, j - 1))$$

→≡ insert       ↓≡ delete      \equiv copy/replace
Edit Distance, 7

\[ E(i, j) = \min(1 + E(i - 1, j), 1 + E(i, j - 1), \text{diff}(i, j) + E(i - 1, j - 1)) \]

→≡ insert     ↓≡ delete     ↘≡ copy/replace

\[
\begin{array}{ccccccc}
\text{k} & \text{i} & \text{t} & \text{t} & \text{e} & \text{n} \\
0  & 1  & 2  & 3  & 4  & 5  & 6  \\
1  & 1  & 2  & 3  & 4  & 5  & 6  \\
2  & 2  & 1  & 2  & 3  & 4  & 5  \\
3  & 3  & 2  & 1  & 2  & 3  & 4  \\
4  & 4  & 3  & 2  & 1  & 2  & 3  \\
5  & 5  & 4  & 3  & 2  & 2  & 3  \\
6  & 6  & 5  & 4  & 3  &  &  \\
\end{array}
\]
Edit Distance, 7

\[ E(i,j) = \min(1 + E(i - 1,j), 1 + E(i,j - 1), \text{diff}(i,j) + E(i - 1,j - 1)) \]

→≡ insert \hspace{1cm} ↓≡ delete \hspace{1cm} \swarrow≡ copy/replace

\[
\begin{array}{cccccc}
  & k & i & t & t & e & n \\
 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
s & 1 & 1 & 2 & 3 & 4 & 5 & 6 \\
i & 2 & 2 & 1 & 2 & 3 & 4 & 5 \\
t & 3 & 3 & 2 & 1 & 2 & 3 & 4 \\
t & 4 & 4 & 3 & 2 & 1 & 2 & 3 \\
i & 5 & 5 & 4 & 3 & 2 & 2 & 3 \\
n & 6 & 6 & 5 & 4 & 3 & 3 & 3 \\
\end{array}
\]
Edit Distance, 7

\[ E(i, j) = \min(1 + E(i - 1, j), 1 + E(i, j - 1), \text{diff}(i, j) + E(i - 1, j - 1)) \]

→≡ insert \hspace{1cm} ↓≡ delete \hspace{1cm} \_\hspace{1cm}≡ copy/replace

\[
\begin{array}{ccccccc}
\text{k} & \text{i} & \text{t} & \text{t} & \text{e} & \text{n} \\
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
1 & 1 & 2 & 3 & 4 & 5 & 6 \\
2 & 2 & 1 & 2 & 3 & 4 & 5 \\
3 & 3 & 2 & 1 & 2 & 3 & 4 \\
4 & 4 & 3 & 2 & 1 & 2 & 3 \\
5 & 5 & 4 & 3 & 2 & 2 & 3 \\
6 & 6 & 5 & 4 & 3 & 3 & 2 \\
\end{array}
\]
Edit Distance, 7

\[ E(i, j) = \min(1 + E(i - 1, j), 1 + E(i, j - 1), \text{diff}(i, j) + E(i - 1, j - 1)) \]

→ ≡ insert, ↓ ≡ delete, ↘ ≡ copy/replace
Every dynamic program has an underlying dag structure: think of each node as representing a subproblem, and each edge as a precedence constraint on the order in which the subproblems can be tackled. Having nodes $u_1, \ldots, u_k$ point to $v$ means "subproblem $v$ can only be solved once the answers to $u_1, \ldots, u_k$ are known."

In our present edit distance application, the nodes of the underlying dag correspond to subproblems, or equivalently, to positions $(i, j)$ in the table. Its edges are the precedence constraints, of the form $(i - 1, j) \to (i, j)$, $(i, j - 1) \to (i, j)$, and $(i - 1, j - 1) \to (i, j)$ (Figure 6.5).

In fact, we can take things a little further and put weights on the edges so that the edit distances are given by shortest paths in the dag! To see this, set all edge lengths to 1, except for $\{(i - 1, j - 1) \to (i, j)\}$ (shown dotted in the figure), whose length is 0. The final answer is then simply the distance between nodes $s = (0, 0)$ and $t = (m, n)$. One possible shortest path is shown, the one that yields the alignment we found earlier. On this path, each move down is a deletion, each move right is an insertion, and each diagonal move is either a match or a substitution.

By altering the weights on this dag, we can allow generalized forms of edit distance, in which insertions, deletions, and substitutions have different associated costs.
The Knapsack Problem (KP)

Given:
- A knapsack with weight capacity $W$.
- Items $1, \ldots, n$ where item $i$ has weight $w_i$ and value $v_i$.

... with repetition

Find: a multiset $M \subseteq \{1, \ldots, n\}$ so that
- $\sum_{i \in M} w_i \leq W$ and
- $\sum_{i \in M} v_i$ is maximized.

... without repetition

Find: a set $S \subseteq \{1, \ldots, n\}$ so that
- $\sum_{i \in S} w_i \leq W$ and
- $\sum_{i \in S} v_i$ is maximized.
Knapsack with repetition

**Given:**
- A knapsack with capacity $W$.
- Items $1, \ldots, n$
- Item $i$ has weight $w_i$ & value $v_i$.

**Find:** a multiset $M \subseteq \{1, \ldots, n\} \ni$
- $\sum_{i \in M} w_i \leq W$ and
- $\sum_{i \in M} v_i$ is maximized.

$$K(w) = \begin{cases} \text{max. value gained from a} \\ \text{knapsack with cap. } w \end{cases}$$
$$= \max_{i: w_i \leq w} K(w - w_i) + v_i$$

\begin{array}{l}
\text{array } K[0..W] \\
K[0] \leftarrow 0 \\
\text{for } w \leftarrow 1 \text{ to } W \text{ do} \\
\quad K[w] \leftarrow 0 \\
\quad \text{for } i \leftarrow 1 \text{ to } n \text{ do} \\
\qquad \text{if } w_i \leq w \text{ then} \\
\qquad \quad K[w] \leftarrow \max(K[w], K[w - w_i] + v_i)
\end{array}

- This runs in $\Theta(n \cdot W)$ time.
- Since we usually measure the size of $W$ as $|W| = \text{the number of bits in the binary rep. of } W$.
  Hence, $\Theta(n \cdot W) = \Theta(n \cdot 2^{|W|})$.
- So this is only useful for small values of $W$. 
Knapsack without repetition

Given:
- A knapsack with capacity \( W \).
- Items 1, \ldots, n
- Item \( i \) has weight \( w_i \) & value \( v_i \).

Find: a set \( M \subseteq \{ 1, \ldots, n \} \) \( \exists \)
- \( \sum_{i \in M} w_i \leq W \) and
- \( \sum_{i \in M} v_i \) is maximized.

Problem
- \( K[w - w_n] \) is not useful since it does not tell you whether item \( n \) was used in an optimal solution.

Therefore, we refine things to:

\[
K[w, j] = \begin{cases} 
\text{the best value obtainable with capacity } w \\
\text{using items from } 1, \ldots, j \\
K[w, j - 1], & \text{if } w_j > w; \\
\max(K[w, j - 1], K[w - w_j, j - 1] + v_j), & \text{otherwise.}
\end{cases}
\]

(Why?)
Knapsack without repetition, 2

Knapsack w/o repetition

Given:
• A knapsack with capacity $W$.
• Items 1, . . . , $n$
• Item $i$ has weight $w_i$ & value $v_i$.

Find: a set $M \subseteq \{ 1, \ldots , n \}$ s.t.
• $\sum_{i \in M} w_i \leq W$ and
• $\sum_{i \in M} v_i$ is maximized.

Our recursive relation is:

$$K[w,j] = \begin{cases} 
\text{the best value obtainable with capacity } w \\
\text{using items from } 1, \ldots , j \\
K[w,j-1], \text{ if } w_j > w; \\
\max(K[w,j-1], K[w-w_j,j-1] + v_j), \text{ otherwise.}
\end{cases}$$

array $K[0..W, 0..n]$  // This is also $\Theta(n \cdot W)$ time.
for $w \leftarrow 0$ to $W$ do $K[w,0] \leftarrow 0$  // Hence only useful
for $j \leftarrow 1$ to $n$ do $K[0,j] \leftarrow 0$  // when $W$ is small.
for $j \leftarrow 1$ to $n$ do
  for $w \leftarrow 1$ to $W$ do
    if $w_i > w$ then $K[w,j] \leftarrow K[w,j-1]$
    else $K[w,j] \leftarrow \max(K[w,j-1], K[w-w_i,j-1] + v_i)$
Chain matrix multiplication, 1

Recall: Multiplying an $d_0 \times d_1$ matrix by an $d_1 \times d_2$ matrix results in a $d_0 \times d_2$ matrix and takes $(d_0 \cdot d_1 \cdot d_2)$-many scalar multiplies.

The Chain Matrix Multiplication Problem (CMMP)

Given: $d_0, \ldots, d_n \in \mathbb{N}^+$ and matrices $A_1, \ldots, A_n$ where $\dim(A_i) = d_{i-1} \times d_i$.
Find: The cheapest way to order the multiplications.

Example: $A \times B \times C \times D$

Suppose that
- $A$ is an $50 \times 20$ matrix.
- $B$ is an $20 \times 1$ matrix.
- $C$ is an $1 \times 10$ matrix.
- $D$ is an $10 \times 100$ matrix.

Then:

<table>
<thead>
<tr>
<th>Parenthesization</th>
<th>Cost computation</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \times (B \times (C \times D))$</td>
<td>$1 \cdot 10 \cdot 100 + 20 \cdot 1 \cdot 100 + 50 \cdot 20 \cdot 100$</td>
<td>103,000</td>
</tr>
<tr>
<td>$A \times ((B \times C) \times D)$</td>
<td>$20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 + 50 \cdot 20 \cdot 100$</td>
<td>120,200</td>
</tr>
<tr>
<td>$(A \times B) \times (C \times D)$</td>
<td>$50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 + 50 \cdot 1 \cdot 100$</td>
<td>7,000</td>
</tr>
<tr>
<td>$(A \times (B \times C)) \times D$</td>
<td>$20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 + 50 \cdot 10 \cdot 100$</td>
<td>60,200</td>
</tr>
<tr>
<td>$((A \times B) \times C) \times D$</td>
<td>$50 \cdot 20 \cdot 1 + 50 \cdot 1 \cdot 10 + 50 \cdot 10 \cdot 100$</td>
<td>151,000</td>
</tr>
</tbody>
</table>
Chain matrix multiplication, 2

<table>
<thead>
<tr>
<th>Parenthesization</th>
<th>Cost computation</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \times (B \times (C \times D))$</td>
<td>$1 \cdot 10 \cdot 100 + 20 \cdot 1 \cdot 100 + 50 \cdot 20 \cdot 100$</td>
<td>103,000</td>
</tr>
<tr>
<td>$A \times ((B \times C) \times D)$</td>
<td>$20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 + 50 \cdot 20 \cdot 100$</td>
<td>120,200</td>
</tr>
<tr>
<td>$(A \times B) \times (C \times D)$</td>
<td>$50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 + 50 \cdot 1 \cdot 100$</td>
<td>7,000</td>
</tr>
<tr>
<td>$(A \times (B \times C)) \times D$</td>
<td>$20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 + 50 \cdot 10 \cdot 100$</td>
<td>60,200</td>
</tr>
<tr>
<td>$((A \times B) \times C) \times D$</td>
<td>$50 \cdot 20 \cdot 1 + 50 \cdot 1 \cdot 10 + 50 \cdot 10 \cdot 100$</td>
<td>151,000</td>
</tr>
</tbody>
</table>

X

A X

B X

C D

X

A X

B C

D

X

A B C D

X D

X C

A B
The Chain Matrix Multiplication Problem (CMMP)

**Given:** $d_0, \ldots, d_n \in \mathbb{N}^+$ and matrices $A_1, \ldots, A_n$ where $\text{dim}(A_i) = d_{i-1} \times d_i$.

**Find:** The cheapest way to order the multiplications.

**The $C(i, k)$ subproblem, $1 \leq i \leq k \leq n$**

**Find:** $C(i, k) = \text{the min. cost of the } A_i \times \cdots \times A_k$.

$C(i, i) = 0.$

$$C(i, k) = \min_{j=i, \ldots, k-1} \left( C(i, j) + C(j + 1, k) + d_{i-1} \cdot d_j \cdot d_k \right), \text{ where } i < k.$$  

$C(1, n)$ is the minimal cost of the CMMP for $A_1, \ldots, A_n$.

**Question:** Why does optimal substructure hold?
Chain matrix multiplication, 4

The $C(i,k)$ subproblem, $1 \leq i \leq k \leq n$

**Find:** $C(i,k) = \text{the min. cost of the } A_i \times \cdots \times A_k$.  

$C(i,i) = 0$.  

$C(i,k) = \min_{j=i,\ldots,k-1} (C(i,j) + C(j+1,k) + d_{i-1} \cdot d_j \cdot d_k)$, where $i < k$.

$C(1,n)$ is the minimal cost of the CMMP for $A_1, \ldots, A_n$.

```
for i ← 1 to n do C[i,i] ← 0
for s ← 1 to n−1 do
  // Compute C[1,1+s], C[2,2+s],…,C[n−s,n]
  for i ← 1 to n−s do
    k ← i+s;  C[i,k] ← +∞
    for j ← i to k−1 do
      C[i,k] ← min(C[i,k],C[i,j] + C[j+1,k] + d_{i−1} \cdot d_j \cdot d_k)
  return C[1,n]
```

**Run Time:** $\Theta(n^3)$.

**Problem:** How to reconstruct the order of mult. from the $C[\cdot,\cdot]$ table?
Shortest paths: All-pairs, 1
Shortest paths: All-pairs, 2

All-Pairs Shortest Paths Problem (APSP)

**Given:** \( G = (\{1, \ldots, n\}, E) \) an undirected graph and \( \text{len}: E \to \mathbb{R}^+ \).

**Construct:** \( S[1..n, 1..n] \) so that \( S[i,j] = \) the length of a shortest \( G \)-path from \( i \) to \( j \).

**Assumption:** \( G \) is initially given by a matrix \( A[1..n, 1..n] \) so that

\[
A[i,j] = \begin{cases} 
\text{len}(i,j), & \text{if } (i,j) \in E; \\
+\infty, & \text{if } (i,j) \notin E.
\end{cases}
\]

**APSP, restated:** Given \( A[1..n, 1..n] \), compute \( S[1..n, 1..n] \).

**Question:** What are good subproblems?

- \( A \) is the approximation to \( S \) in which that paths have no intermediate vertices: \( \checkmark \ 1 \rightarrow 3 \quad \times \ 1 \rightarrow 2 \rightarrow 3 \)
- Allowing more and more intermediate vertices gives subproblems …
Shortest paths: All-pairs, 3

\[ dist[i,j,k] = \begin{cases} 
\text{the length of a shortest } G\text{-path from } i \text{ to } j \text{ using intermediate vertices from } \{1, \ldots, k\} \\
\end{cases} \]

\[ dist[i,j,0] = A[i,j] = \begin{cases} 
\text{len}(i,j), \text{ if } (i,j) \in E; \\
+\infty, \text{ if } (i,j) \notin E.
\end{cases} \]

\[ dist[i,j,n] = S[i,j] = \{\text{the length of a shortest } G\text{-path from } i \text{ to } j\} \]
Shortest paths: All-pairs, 3

\[ dist[i, j, k] = \begin{cases} 
\text{the length of a shortest } G \text{-path from } i \text{ to } j \text{ using intermediate vertices from } \{1, \ldots, k\} \\
\end{cases} \]

\[ dist[i, j, 0] = A[i, j] = \begin{cases} 
\text{len}(i, j), & \text{if } (i, j) \in E; \\
+\infty, & \text{if } (i, j) \notin E.
\end{cases} \]

\[ dist[i, j, n] = S[i, j] = \{\text{the length of a shortest } G \text{-path from } i \text{ to } j\} \]

**Question:** How do we go from \( dist[\cdot, \cdot, k - 1] \) to \( dist[\cdot, \cdot, k] \)?

**Note:** \( dist[i, j, k] \leq dist[i, j, k - 1] \).
Shortest paths: All-pairs, 4

Question: How do we go from \(\text{dist}[\cdot,\cdot,k-1]\) to \(\text{dist}[\cdot,\cdot,k]\), where:

\[
\text{dist}[i,j,k] = \begin{cases} 
\text{the length of a shortest } G\text{-path from } i \text{ to } j \text{ using intermediate vertices from } \{1,\ldots,k\} 
\end{cases}
\]

- \(\text{dist}[i,j,k-1] = \text{dist}[i,j,k]\) means: there is a shortest path from \(i\) to \(j\) using intermediate vertices from \(\{1,\ldots,k\}\) that does not use \(k\).
Question: How do we go from $\text{dist}[\cdot, \cdot, k - 1]$ to $\text{dist}[\cdot, \cdot, k]$, where:

$$\text{dist}[i, j, k] = \begin{cases} \text{the length of a shortest } G\text{-path from } i \text{ to } j \text{ using intermediate vertices from } \{1, \ldots, k\} \\
\end{cases}$$

- $\text{dist}[i, j, k - 1] > \text{dist}[i, j, k]$ means: shortest paths from $i$ to $j$ using intermediate vertices from $\{1, \ldots, k\}$ must use $k$.

- $\text{dist}[i, j, k] = \text{dist}[i, k, k - 1] + \text{dist}[k, j, k - 1]$.  \quad \text{(Why?)}
Question: How do we go from $\text{dist}[\cdot, \cdot, k - 1]$ to $\text{dist}[\cdot, \cdot, k]$?

$\text{dist}[i, j, k] = \begin{cases} 
\text{the length of a shortest } G \text{-path from } i \text{ to } j \text{ using intermediate vertices from } \{1, \ldots, k\} \\
\min(\text{dist}[i, j, k - 1], \text{dist}[i, k, k - 1] + \text{dist}[k, j, k - 1])
\end{cases}$

```plaintext
for i ← 1 to n do // Initialization
    for j ← 1 to n do
        dist[i, j, 0] ← A[i, j]
for k ← 1 to n do // Main iteration
    for i ← 1 to n do
        for j ← 1 to n do
            dist[i, j, k] ← min(dist[i, j, k - 1],
                               dist[i, k, k - 1] + dist[k, j, k - 1])
for i ← 1 to n do // Output
    for j ← 1 to n do
        S[i, j] ← dist[i, j, n]
return S
```
Shortest paths: All-pairs, 7

Time complexity: $\Theta(|V|^3)$. \textit{(Why?)}
Space complexity: Also $\Theta(|V|^3)$, but this is easy to improve to $\Theta(|V|^2)$.

\begin{verbatim}
for i ← 1 to n do  // Initialization
    for j ← 1 to n do
        dist[i, j, 0] ← A[i, j]
    for k ← 1 to n do  // Main iteration
        for i ← 1 to n do
            for j ← 1 to n do
                dist[i, j, k] ← min(dist[i, j, k − 1],
                                dist[i, k, k − 1] + dist[k, j, k − 1])
        for i ← 1 to n do  // Output
            for j ← 1 to n do
                S[i, j] ← dist[i, j, n]
    return S
\end{verbatim}
The traveling salesman problem

Given: $G$ a complete graph on verts $1, \ldots, n$ and $d(i,j) =$ (the distance of $i$ to $j$) $< \infty$

Find: A minimal cost of a complete tour of $G$.

Subproblems
For $S \subseteq \{1, \ldots, n\}$ with $1 \in S$ and $j \in S$:

$$C[S,j] = \begin{cases} 
\text{the minimal cost of a path from} \\
\text{1 to } j \text{ using just nodes in } S.
\end{cases}$$

$$C[\{1\}, 1] = 0.$$  

$$C[S, 1] = \infty, \text{ when } |S| > 1. \text{ Why?}$$

$$C[S,j] = \min_{i \in (S - \{j\})} C[S - \{j\}, i] + dist(i,j).$$

Note: This is a set up for NP-completeness.

$\Leftarrow$ There are at most $2^n \cdot n$-many subproblems, and each one takes $O(n)$ time to solve.

$!!!$ This takes $O(n^2 2^n)$ time!!!
Independent sets in trees, 1

Definition
Suppose \( G = (V, E) \) is an undirected graph.

(a) \( u \) and \( v \in V \) are independent when \( (u, v) \notin E \).

(b) \( U \subseteq V \) is an independent set when every pair of elements from \( U \) are independent.

The Independent Set Problem (ISP)
Given: \( G \) an undirected graph.
Find: A max-sized independent set for \( G \).

The Indep. Set Problem for Trees
Given: \( T \) a tree.
Find: A max-sized independent set for \( T \).

Example
In the graph below:
• \( \{1, 4, 5\} \) is not an independent set.
• \( \{1, 5\} \) and \( \{1, 6\} \) are independent sets.
• \( \{2, 3, 6\} \) and \( \{1, 4, 6\} \) are max-sized independent sets.
Independent sets in trees, 2

Definition
Suppose $G = (V, E)$ is an undirected graph.

(a) $u$ and $v \in V$ are independent when $(u, v) \notin E$.

(b) $U \subseteq V$ is an independent set when every pair of elements from $U$ are independent.

The Indep. Set Problem for Trees
Given: $T$ a tree.
Find: A max-sized independent set for $T$.

Strategy
- Pick some vertex of $T$ as the root, $r$.
- Now each vertex of $T$ is the root of a subtree.
- For each $v$ in $T$, define $I(v) =$ the size of a largest independent set in $v$’s subtree.
- $I(v) = 1$ when $v$ is a leaf.
- $I(r) =$ the size of a largest indep. set in $T$.
- So, what is the recursion?
Independent sets in trees, 3

Strategy

• Pick some vertex of $T$ as the root, $r$.
• For each $v$ in $T$: $I(v) =$ the size of a largest indep. set in $v$’s subtree.
• $I(v) = 1$ when $v$ is a leaf and $I(r) =$ the size of a largest indep. set in $T$.
• What is the recursion for $I(u)$?

Case: $u$ is in some maximal sized indep. set for $u$’s subtree
Then $I(u) = 1 + \sum \{ I(v) : v$ is a grandchild of $u \}$.

Case: $u$ is in no maximal sized indep. set for $u$’s subtree
Then $I(u) = \sum \{ I(v) : v$ is a child of $u \}$.

$$I(u) = \max \left( 1 + \sum_{v \text{ is a grandchild of } u} I(v), \sum_{v \text{ is a child of } u} I(v) \right)$$
Independent sets in trees, 4

Strategy

- Pick some vertex of $T$ as the root, $r$.
- For each $u$ in $T$ compute:

$$I(u) = \text{the size of a largest indep. set in } u\text{'s subtree}$$

$$= \max \left( 1 + \sum_{v \text{ is a grandchild of } u} I(v), \sum_{v \text{ is a child of } u} I(v) \right)$$

- This can be done in $\Theta(|V|)$ time.
- For the general graph case, $O(2^{|V|})$ is the best known time.

**Note:** This is another set up for NP-completeness.