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 $y[1..n]$.

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

$$\left.\begin{array}{cccccc} S & - & N & O & W & Y \\ S & U & N & N & - & Y \end{array}\right\} \text{ edit distance as alignment}$$

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

Edit Distance, 2

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 $y[1..n]$.

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 $i = 1,\ldots,m,\ j = 1,\ldots,n$.

$$E(i,j) \ = \ \begin{cases} \text{some combination of the solutions} \\ \text{to smaller } E(i',j') \text{ problems} \end{cases}$$

Edit Distance, 3

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:

$$\left|\begin{matrix} x_i \\ - \end{matrix}\right| \quad or \quad \left|\begin{matrix} - \\ y_j \end{matrix}\right| \quad or \quad \left|\begin{matrix} x_i \\ y_j \end{matrix}\right|$$

So...?

Edit Distance, 4

The $E(i,j)$ Problem

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

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

Edit Distance, 4

The $E(i,j)$ Problem

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

$$\begin{cases} \text{Case } \left|\begin{matrix} x_i \\ - \end{matrix}\right| : & \text{The cost is } 1 + E(i-1,j). \\ \text{Case } \left|\begin{matrix} - \\ y_j \end{matrix}\right| : & \text{The cost is } 1 + E(i,j-1). \\ \text{Case } \left|\begin{matrix} x_i \\ y_j \end{matrix}\right| : & \begin{cases} \text{Subcase } x_i = y_j\text{:} & \text{The cost is } E(i-1,j-1). \\ \text{Subcase } x_i \neq y_j\text{:} & \text{The cost is } 1 + E(i-1,j-1). \end{cases} \end{cases}$$

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

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

Edit Distance, 5

The $E(i,j)$ Problem

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

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

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

The base cases:

$$E(i,0) = i. \qquad E(0,j) = j.$$

Why?

Edit Distance, 6

The $E(i,j)$ Problem

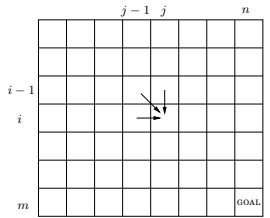
Compute $E(i,j) =$
the edit distance of $x[1..i]$ and $y[1..j]$.

function edcost($x[1..m], y[1..n]$)
  for $i \leftarrow 0$ **to** m **do** $E[i,0] \leftarrow i$
  for $j \leftarrow 1$ **to** n **do** $E[0,j] \leftarrow j$
  for $i \leftarrow 1$ **to** m **do**
    for $j \leftarrow 1$ **to** n **do**
      $E[i,j] \leftarrow$

$$\min\Big(1 + E[i-1,j],\ 1 + E[i,j-1],\ diff(i,j) + E[i-1,j-1]\Big)$$

  return $E[m,n]$

See `http://www.miislita.com/searchito/levenshtein-edit-distance.html` for a nice animation.



→≡ insert ↓≡ delete ↘≡ copy/replace

		P	O	L	Y	N	O	M	I	A	L
	0	1	2	3	4	5	6	7	8	9	10
E	1	1	2	3	4	5	6	7	8	9	10
X	2	2	2	3	4	5	6	7	8	9	10
P	3	2	3	3	4	5	6	7	8	9	10
O	4	3	2	3	4	5	5	6	7	8	9
N	5	4	3	3	4	4	5	6	7	8	9
E	6	5	4	4	4	5	5	6	7	8	9
N	7	6	5	5	5	4	5	6	7	8	9
T	8	7	6	6	6	5	5	6	7	8	9
I	9	8	7	7	7	6	6	6	6	7	8
A	10	9	8	8	8	7	7	7	7	6	7
L	11	10	9	8	9	8	8	8	8	7	6

Edit Distance, 7

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

$\rightarrow \equiv$ insert $\quad \downarrow \equiv$ delete $\quad \searrow \equiv$ copy/replace

		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						

Edit Distance, 7

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

$\rightarrow \equiv$ insert $\quad \downarrow \equiv$ delete $\quad \searrow \equiv$ copy/replace

		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					

Edit Distance, 7

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

$\rightarrow \equiv$ insert $\quad \downarrow \equiv$ delete $\quad \searrow \equiv$ copy/replace

		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				

Edit Distance, 7

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

$\rightarrow \equiv$ insert $\quad \downarrow \equiv$ delete $\quad \searrow \equiv$ copy/replace

		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			

Edit Distance, 7

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

$\rightarrow \equiv$ insert $\quad \downarrow \equiv$ delete $\quad \searrow \equiv$ copy/replace

		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		

Edit Distance, 7

$$E(i,j) = \min(1 + E(i-1,j), 1 + E(i,j-1), diff(i,j) + E(i-1,j-1))$$

$\rightarrow \equiv$ insert $\quad \downarrow \equiv$ delete $\quad \searrow \equiv$ copy/replace

		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	

Edit Distance, 7

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

$\rightarrow \equiv$ insert $\quad \downarrow \equiv$ delete $\quad \searrow \equiv$ copy/replace

		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	2

Edit Distance, 7

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

$\rightarrow \equiv$ insert $\quad \downarrow \equiv$ delete $\quad \searrow \equiv$ copy/replace

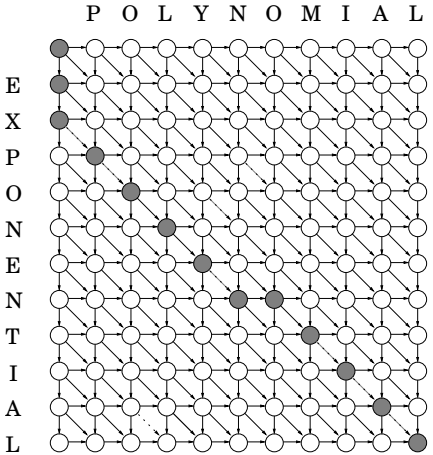
		k	i	t	t	e	n
	0	1	2	3	4	5	6
s	1	1	2	3	4	5	6
i	2	2	1	2	3	4	5
t	3	3	2	1	2	3	4
t	4	4	3	2	1	2	3
i	5	5	4	3	2	2	3
n	6	6	5	4	3	3	2
g	7	7	6	5	4	4	3

Edit Distance, 8

$\rightarrow \equiv$ insert

$\downarrow \equiv$ delete

$\searrow \equiv$ copy/replace



The underlying dependencies dag and a cost-6 path.

Knapsack—Again

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.

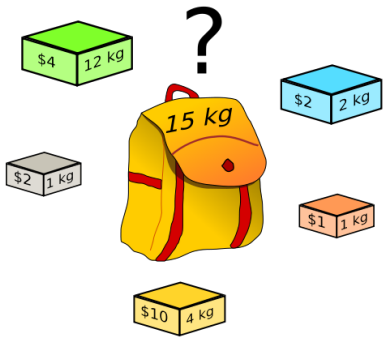


Image from: http://commons.wikimedia.org/wiki/File:Knapsack.svg

Knapsack with repetition

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.

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

(Why?)

```
array K[0..W]
K[0] ← 0
for w ← 1 to W do
    K[w] ← 0
    for i ← 1 to n do
        if w_i ≤ w then
            K[w] ← max(K[w], K[w − w_i] + v_i)
```

- This runs in $\Theta(n \cdot W)$ time.
- Since we usually measure the size of W as $|W|$ = 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, 1

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 $M \subseteq \{1, \ldots, n\} \ni$

- $\sum_{i \in M} w_i \leq W$ 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:

$$K[w, j] = \begin{cases} \text{the best value obtainable with capacity } w \\ \text{using items from } 1, \ldots, j \end{cases}$$

$$= \begin{cases} K[w, j-1], & \text{if } w_j > w; \\ \max(K[w, j-1], K[w - w_j, j-1] + v_j), \\ \qquad \text{otherwise.} \end{cases}$$

(Why?)

Knapsack without repetition, 2

Knapsack w/o repetition

Given:

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

Find: a set $M \subseteq \{1, \ldots, n\} \ni$

- $\sum_{i \in M} w_i \leq W$ and
- $\sum_{i \in M} v_i$ is maximized.

Our recursive relation is:

$$K[w,j] = \begin{cases} \text{the best value obtainable with capacity } w \\ \text{using items from } 1, \ldots, j \end{cases}$$

$$= \begin{cases} K[w, j-1], & \text{if } w_j > w; \\ \max(K[w, j-1], K[w - w_j, j-1] + v_j), & \\ \qquad \text{otherwise.} & \end{cases}$$

```
array K[0..W, 0..n]                          // This is also Θ(n · W) time.
for w ← 0 to W do K[w, 0] ← 0                // Hence only useful
for j ← 1 to n do K[0, j] ← 0                //  when W is small.
for j ← 1 to n do
    for w ← 1 to W do
        if w_i > w then K[w, j] ← K[w, j − 1]
                   else  K[w, j] ← 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: $A \times B \times C \times D$

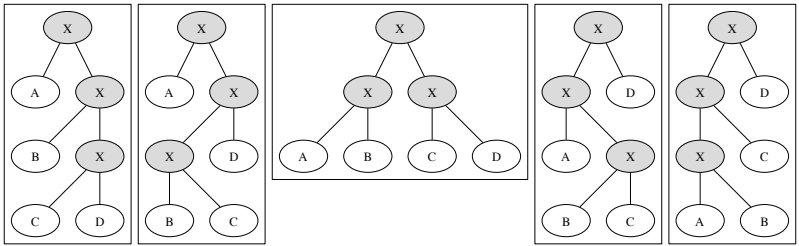
Suppose that
- A is an 50×20 matrix.
- B is an 20×1 matrix.
- C is an 1×10 matrix.
- D is an 10×100 matrix.

Then:

Parenthesization	Cost computation	Cost
$A \times (B \times (C \times D))$	$1 \cdot 10 \cdot 100 + 20 \cdot 1 \cdot 100 + 50 \cdot 20 \cdot 100$	103,000
$A \times ((B \times C) \times D)$	$20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 + 50 \cdot 20 \cdot 100$	120,200
$(A \times B) \times (C \times D)$	$50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 + 50 \cdot 1 \cdot 100$	7,000
$(A \times (B \times C)) \times D$	$20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 + 50 \cdot 10 \cdot 100$	60,200
$((A \times B) \times C) \times D$	$50 \cdot 20 \cdot 1 + 50 \cdot 1 \cdot 10 + 50 \cdot 10 \cdot 100$	151,000

Chain matrix multiplication, 2

Parenthesization	Cost computation	Cost
$A \times (B \times (C \times D))$	$1 \cdot 10 \cdot 100 + 20 \cdot 1 \cdot 100 + 50 \cdot 20 \cdot 100$	103,000
$A \times ((B \times C) \times D)$	$20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 + 50 \cdot 20 \cdot 100$	120,200
$(A \times B) \times (C \times D)$	$50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 + 50 \cdot 1 \cdot 100$	7,000
$(A \times (B \times C)) \times D$	$20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 + 50 \cdot 10 \cdot 100$	60,200
$((A \times B) \times C) \times D$	$50 \cdot 20 \cdot 1 + 50 \cdot 1 \cdot 10 + 50 \cdot 10 \cdot 100$	151,000



Chain matrix multiplication, 3

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.

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

$$
\begin{aligned}
\textbf{Find: } C(i,k) &= \text{the min. cost of the } A_i \times \cdots \times A_k. \\
C(i,i) &= 0. \\
C(i,k) &= \min_{j=i,\ldots,k-1} \Big(C(i,j) + C(j+1,k) + d_{i-1} \cdot d_j \cdot d_k \Big), \quad \text{where } i < k.
\end{aligned}
$$

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

$$\begin{aligned} \textbf{Find: } C(i,k) &= \text{the min. cost of the } A_i \times \cdots \times A_k. \\ C(i,i) &= 0. \\ C(i,k) &= \min_{j=i,\ldots,k-1} \big(C(i,j) + C(j+1,k) + d_{i-1} \cdot d_j \cdot d_k\big), \quad \text{where } i < k. \end{aligned}$$

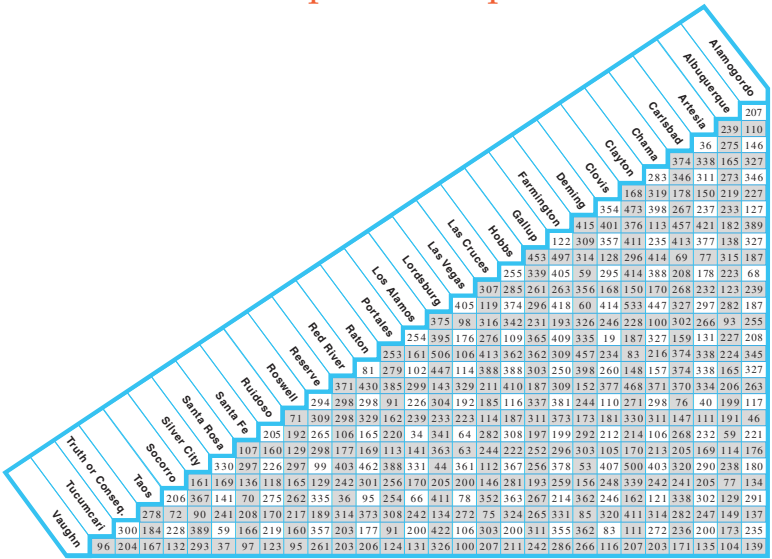
$C(1,n)$ is the minimal cost of the CMMP for $A_1, \ldots, A_n$.

```
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} · d_j · 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 $len{:}E \to \mathbb{R}^+$.
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

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

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: ✓ ❶→❸ ✗ ❶→❷→❸
- Allowing more and more intermediate vertices gives subproblems ...

Shortest paths: All-pairs, 3

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

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

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

Shortest paths: All-pairs, 3

$$
\begin{aligned}
dist[i,j,k] &= \begin{cases} \text{the length of a shortest } G\text{-path from } i \text{ to } j \text{ using inter-} \\ \text{mediate vertices from } \{1,\ldots,k\} \end{cases} \\
dist[i,j,0] &= A[i,j] = \begin{cases} len(i,j), & \text{if } (i,j) \in E; \\ +\infty, & \text{if } (i,j) \notin E. \end{cases} \\
dist[i,j,n] &= S[i,j] = \{\text{the length of a shortest } G\text{-path from } i \text{ to } j
\end{aligned}
$$

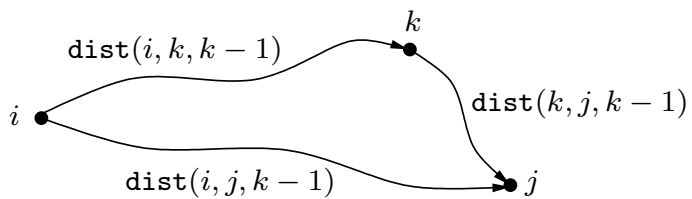
Question: How do we go from $dist[\cdot,\cdot,k-1]$ to $dist[\cdot,\cdot,k]$?

Note: $dist[i,j,k] \leq dist[i,j,k-1]$.

Shortest paths: All-pairs, 4

Question: How do we go from $dist[\cdot,\cdot,k-1]$ to $dist[\cdot,\cdot,k]$, where:

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

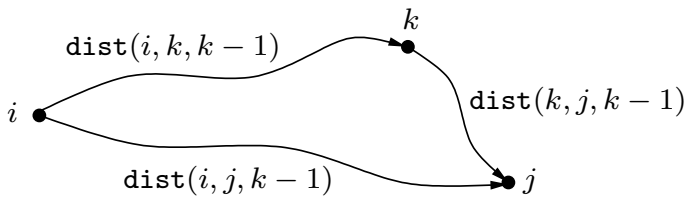


- $dist[i,j,k-1] = dist[i,j,k]$ means: there is a shortest path from i to j using intermediate vertices from $\{1,\ldots,k\}$ that does not use k.

Shortest paths: All-pairs, 5

Question: How do we go from $dist[\cdot,\cdot,k-1]$ to $dist[\cdot,\cdot,k]$, where:

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



- $dist[i,j,k-1] > dist[i,j,k]$ means: shortest paths from i to j using intermediate vertices from $\{1,\ldots,k\}$ must use k.
  $\therefore dist[i,j,k] = dist[i,k,k-1] + dist[k,j,k-1]$. *(Why?)*

Shortest paths: All-pairs, 6

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

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

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

```
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
```

The traveling salesman problem

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

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

Subproblems

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

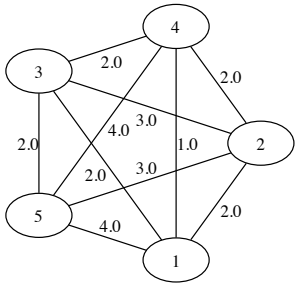
$$C[S,j] = \begin{cases} \text{the minimal cost of a path from} \\ 1 \text{ to } j \text{ using just nodes in } S. \end{cases}$$

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

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

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

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.

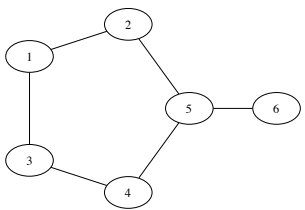
The Indep. Set Problem for Trees

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 $I(u) = 1 + \sum\{ I(v) : v \text{ is a grandchild of } u \}$.

Case: u is in no maximal sized indep. set for u's subtree

Then $I(u) = \sum\{ I(v) : v \text{ is a child of } u \}$.

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

Independent sets in trees, 4

Strategy

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

$$\begin{aligned} I(u) &= \text{the size of a largest indep. set in } u\text{'s subtree} \\ &= \max\left(1 + \sum_{v \text{ is a grandchild of } u} I(v), \sum_{v \text{ is a child of } u} I(v)\right) \end{aligned}$$

- 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.