

Dynamic Programming

DPV Chapter 6, Part 2

Jim Royer

March 6, 2019

1 / 30

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 .

2 / 30

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 $\xrightarrow{\text{subst}}$ SUNN~~W~~Y $\xrightarrow{\text{del}}$ SUNNY

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

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

3 / 30

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

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

4 / 30

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:

$$\left| \begin{array}{c} x_i \\ - \end{array} \right| \quad \text{or} \quad \left| \begin{array}{c} - \\ y_j \end{array} \right| \quad \text{or} \quad \left| \begin{array}{c} x_i \\ y_j \end{array} \right|$$

So...?

5 / 30

Edit Distance, 4

The $E(i, j)$ Problem

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

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

6 / 30

Edit Distance, 4

The $E(i, j)$ Problem

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

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

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

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

6 / 30

Edit Distance, 5

The $E(i, j)$ Problem

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

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

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

The base cases:

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

Why?

7 / 30

Knapsack with repetition

Knapsack with repetition

Given:

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

Find: a multiset $M \subseteq \{1, \dots, n\} \ni$

- $\sum_{i \in M} w_i \leq W$ 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 $|W| = \text{the number of bits in the binary rep. of } W$. Hence, $\Theta(n \cdot W) = \Theta(n \cdot 2^{|W|})$.
- So this is only useful for small values of W .

12 / 30

Knapsack without repetition, 1

Knapsack without repetition

Given:

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

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

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

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

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

(Why?)

13 / 30

Knapsack without repetition, 2

Knapsack w/o repetition

Given:

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

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

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

Our recursive relation is:

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

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

```
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_j > w then K[w, j] ← K[w, j - 1]
    else K[w, j] ← max(K[w, j - 1], K[w - w_j, j - 1] + v_j)
```

14 / 30

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: $d_0, \dots, d_n \in \mathbb{N}^+$ and matrices A_1, \dots, 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

15 / 30

Shortest paths: All-pairs, 2

All-Pairs Shortest Paths Problem (APSP)

Given: $G = (\{1, \dots, n\}, E)$ an undirected graph and $len: E \rightarrow \mathbb{R}^+$.

Construct: $S[1..n, 1..n]$ so that $S[i, j]$ is 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: $\checkmark \textcircled{1} \rightarrow \textcircled{2}$ $\times \textcircled{1} \rightarrow \textcircled{2} \rightarrow \textcircled{3}$
- Allowing more and more intermediate vertices gives subproblems ...

20 / 30

Shortest paths: All-pairs, 3

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

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

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

21 / 30

Shortest paths: All-pairs, 3

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

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

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

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

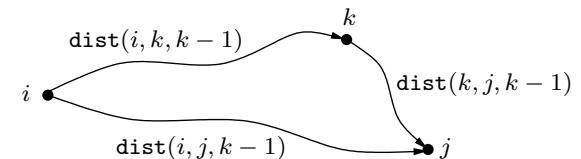
Note: $dist[i, j, k] \leq dist[i, j, k-1]$.

21 / 30

Shortest paths: All-pairs, 4

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

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



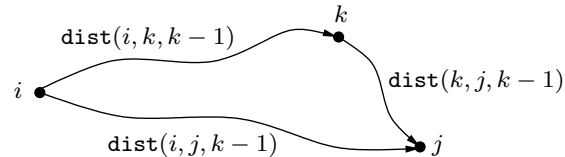
- $dist[i, j, k-1] = dist[i, j, k]$ means: there is a shortest path from i to j using intermediate vertices from $\{1, \dots, k\}$ that **does not** use k .

22 / 30

Shortest paths: All-pairs, 5

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

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



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

23 / 30

Shortest paths: All-pairs, 6

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

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

```

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
    
```

24 / 30

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
    
```

25 / 30

The traveling salesman problem

The traveling salesman problem

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

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

Subproblems

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

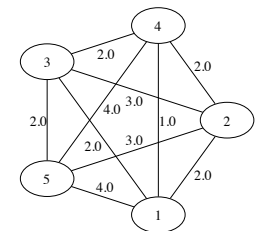
$$C[S, j] = \begin{cases} \text{the minimal cost of a path from} \\ 1 \text{ to } j \text{ using just nodes in } S. \end{cases}$$

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

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

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

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



⇐ 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!!!

26 / 30

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

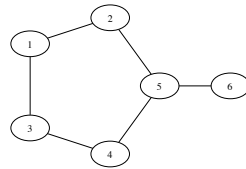
Given: T a tree.

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.



27 / 30

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?

28 / 30

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

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

29 / 30

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.

30 / 30