Lecture: Dynamic Programming III

Overview

- Subproblems for strings
- edit distance (& longest common subseq)
- knapsack

5 Easy Steps to Dynamic Programming:

(1) **define subproblems** | *count # subproblems*

(2) **guess (part of solution)** | *count # choices*

(3) **relate subproblems solutions** | *compute (time/subproblem)*

(4) **recurse or memoize or build DP table bottom up** | *time = (time/subproblem) * #subproblems*

(5) **solve original problem** | a subproblem or a combination of subproblems

Examples:

	Fibonacci	Shortest Paths
1) subproblems	**F_k for $1 \le k \le n$**	**$\delta_k(s, v)$ for $v \in V$, $0 \le k < \lvert V \rvert$**
# subproblems	n	V^2
2) guess	**nothing to guess**	**edge into V (if any)**
# choices	1	indegree(v) + 1
3) recurrence	**$F_k = F_{k-1} + F_{k-2}$**	**$\delta_k(s, v) =$** $\min\{\delta_{k-1}(s, u) + w(u, v)\}$
time / subproblem	O(1) to compute	O(1 + indegree(v))
4) topological order	**for k = 1, …, n**	**for k=0, 1, …, |V| - 1**
total time	**O(n)**	**O(VE)**
5) original subproblem	**F_n**	**$\delta_{\lvert V \rvert - 1}(s, v)$ for $v \in V$**
extra time	O(1)	O(V)

Subproblems for string and sequences:

(1) **suffixes x[i:]** | $O(|x|)$

(2) **prefixes x[:j]**| $O(|x|)$

(3) **substrings x[i:j]** | $O(|x|^2)$

Edit distance

Description:

Given two strings x & y, what is the cheapest possible sequence of character edits (insert c, delete c, replace c → c’) to transform x into y?

- cost of edit depends only on characters alphabet
- for example more common typo a -> s, then less cost
- cost of sequence = sum of costs of edits

Edit distance

Description:

Given two strings x & y, what is the cheapest possible sequence of character edits (insert c, delete c, replace c → c') to transform x into y?

- If *insert* & *delete* cost *1*, replace costs *0*, minimum ***edit distance*** equivalent to finding ***longest common subsequence***. Note that a subsequence is sequential but not necessarily contiguous.

- **H**I**E**ROG**L**YPHO**LO**GY vs. MIC**H**A**EL**ANGE**LO** $\Longrightarrow$ HELLO

Edit distance

Subproblems for multiple strings/sequences :

- combine suffix/prefix/substring subproblems
- multiply state spaces

Examples: Edit distance

1) subprograms
subproblems

$c(i, j)$ = edit-distance($x[i:]$, $y[j:]$) for $0 \leq i < |x|$, $0 \leq j < |y|$
$\Theta(|x| \cdot |y|)$ subproblems

2) guess
choices

guess whether, to turn x into y (3 choices):

- $x[i]$ deleted
- $y[j]$ inserted
- $x[i]$ replaced by $y[j]$

3) recurrence
time / subproblem

$c(i, j)$ = maximum of:

- *cost*(delete $x[i]$) + $c(i + 1, j)$, if $i < |x|$,
- *cost*(insert $y[j]$) + $c(i, j + 1)$, if $j < |y|$,
- *cost*(replace $x[i] \rightarrow y[j]$) + $c(i+1, j+1)$, if $i < |x|$ & $j < |y|$

Examples:

Edit distance

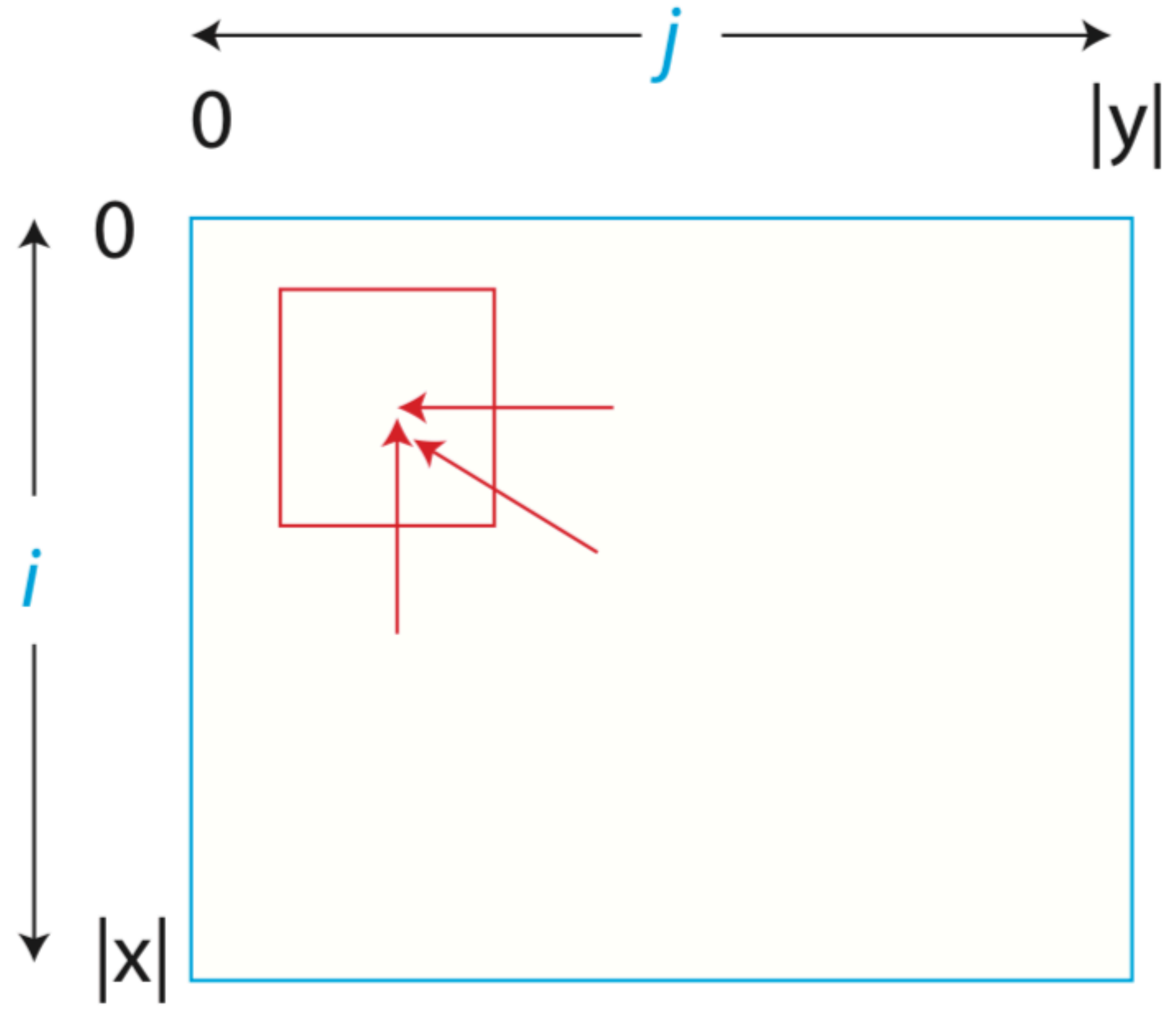
4) topological order
total time

DAG in 2D table:

- bottom-up OR right to left
- only need to keep last 2 rows/columns

$\Rightarrow$ linear space

- total time = $\Theta(|x| \cdot |y|)$



5) original subproblem **c(0, 0)**

Knapsack

Description:

Knapsack of size S you want to pack

- item i has integer size s_i & real value v_i
- goal: choose subset of items of maximum total value subject to total size $\leq S$

Examples:

1) subproblems
subproblems

2) guess
choices

3) recurrence
time / subproblem

Knapsack

value for *suffix i:*|
Θ(|number of items|) subproblems

guessing = whether to include item *i* or not
⇒ # choices = 2

DP[i] = max(DP[i + 1], v_i + DP[i + 1]) , if $s_i \leq S$)

- not enough information. Don't know how much space is left!

Examples:

Knapsack

1) subproblem
\# subproblems

***value for** suffix i **given knapsack of size** X:*
O(nS)

2) guess
\# choices

guessing = whether to include item *i* or not
⇒ # choices = 2

3) recurrence
time / subproblem

DP[i, X] = max(DP[i + 1, X], v_i **+ DP[i + 1, X -** s_i**]) , if** $s_i \leq X$**)**
DP[n, X] = 0

(4) topological order
total time

for i in n,...,0: for X in 0,...S
total time = O(nS)

Examples:

Knapsack

1) subproblems

DP [0, S]

5 Easy Steps to Dynamic Programming:

(1) **define subproblems**

(2) **guess (part of solution)**

(3) **relate subproblems solutions**

(4) **recurse or memoize or build DP table bottom up**

(5) **solve original problem**

Questions