Design and Analysis of Algorithms CS218M

Dynamic Programming

Paritosh Pandya

Indian Institute of Technology, Bombay

Autumn, 2022

Maximum-Weight Non Overlapping Set of Intervals

Problem Given a set of intervals $I_1, \ldots, I_n$ where each interval I_i has a start time $s(I_i)$, end time $f(I_i)$ and a weight $w(I_i)$ aim is to find a subset S of the intervals such that no two intervals in S overlap and the sum of weights of interval $\Sigma_{I_i \in S} \; w(i)$ is maximum.

Maximum-Weight Non Overlapping Set of Intervals

Problem Given a set of intervals $I_1, \ldots, I_n$ where each interval I_i has a start time $s(I_i)$, end time $f(I_i)$ and a weight $w(I_i)$ aim is to find a subset S of the intervals such that no two intervals in S overlap and the sum of weights of interval $\Sigma_{I_i \in S}\, w(i)$ is maximum.

- Let $MWNOSI(S)$ denote the weight of maximum weight subset of non-overlapping intrvals from S.

$I \in S$

$MWNOSI(S) =$

Maximum-Weight Non Overlapping Set of Intervals

Problem Given a set of intervals $I_1, \ldots, I_n$ where each interval I_i has a start time $s(I_i)$, end time $f(I_i)$ and a weight $w(I_i)$ aim is to find a subset S of the intervals such that no two intervals in S overlap and the sum of weights of interval $\Sigma_{I_i \in S}\ w(i)$ is maximum.

- Let $MWNOSI(S)$ denote the weight of maximum weight subset of non-overlapping intrvals from S.

Optimal Recursive Substructure

Let $I \in S$. Then

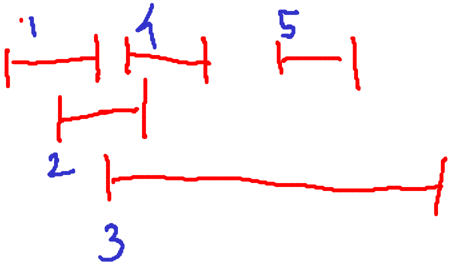
$$MWNOSI(S) \ = \ Max(\ MWNOSI(S - \{I\}),\ MWNOSI(S') + w(I))$$

where S' is the set of intervals from S which do not overlap with I.

Solution 1: Detecting Overlaps

Solution 1: Detecting Overlaps

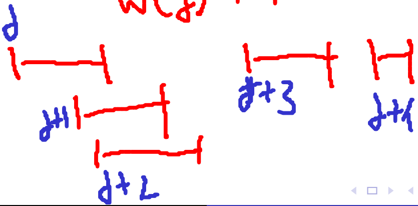
- Arrange intervals of S in increasing order of $s(i)$ to give $I_1, \ldots, I_n$.



Solution 1: Detecting Overlaps

- Arrange intervals of S in increasing order of $s(i)$ to give $I_1, \ldots, I_n$.
- Let $MWNOSI(j)$ denote value of solution for $MWNOSI(I_j, I_{j+1}, \ldots, I_n)$.

$$MWNOSI(j) = \max\left(MWNOSI(j+1),\; w(j) + MWNOSI(k)\right)$$



Solution 1: Detecting Overlaps

- Arrange intervals of S in increasing order of $s(i)$ to give $I_1, \ldots, I_n$.
- Let $MWNOSI(j)$ denote value of solution for $MWNOSI(I_j, I_{j+1}, \ldots, I_n)$.
- $$MWNOSI(j) = \; Max(MWNOSI(j+1), \; MWNOSI(k) + w(I_j))$$

  where $k > j$ with $s(I_k) \geq f(I_j)$ and $S(I_{k-1}) < f(I_j)$.

Solution 1: Detecting Overlaps

- Arrange intervals of S in increasing order of $s(i)$ to give $I_1, \ldots, I_n$.
- Let $MWNOSI(j)$ denote value of solution for $MWNOSI(I_j, I_{j+1}, \ldots, I_n)$.
- $$MWNOSI(j) = Max(MWNOSI(j+1),\ MWNOSI(k) + w(I_j))$$
  where $k > j$ with $s(I_k) \geq f(I_j)$ and $S(I_{k-1}) < f(I_j)$.
- Base case?
- What is the structure of memoization table T.

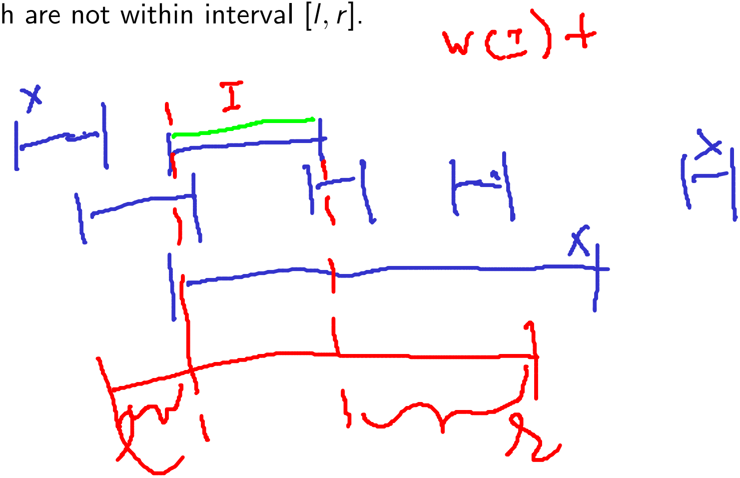
Solution 1: Detecting Overlaps

- Arrange intervals of S in increasing order of $s(i)$ to give $I_1, \ldots, I_n$.
- Let $MWNOSI(j)$ denote value of solution for $MWNOSI(I_j, I_{j+1}, \ldots, I_n)$.
- $$MWNOSI(j) = \ Max(MWNOSI(j+1),\ MWNOSI(k) + w(I_j))$$
  where $k > j$ with $s(I_k) \geq f(I_j)$ and $S(I_{k-1}) < f(I_j)$.
- Base case? $m[n+1] = 0$
- What is the structure of memoization table T. $m[1..n+1]$
- Complexity with memoization? $\Theta(1)$

$\Theta(n)$

Solution 2: Adding Extra Parameters

Let $EMWNOSI(I_1, \ldots, I_n;\ l;\ r)$ denote max weight subset of non-overlapping intervals from the list after removing intervals which are not within interval $[l, r]$.



Solution 2: Adding Extra Parameters

Let $EMWNOSI(I_1, \ldots, I_n;\ l;\ r)$ denote max weight subset of non-overlapping intervals from the list after removing intervals which are not within interval $[l, r]$.

Optimal Recursive Substructure

For $I \in S$ we have $EMWNOSI(S; l; r)\ =$

Solution 2: Adding Extra Parameters

Let $EMWNOSI(I_1, \ldots, I_n;\ l;\ r)$ denote max weight subset of non-overlapping intervals from the list after removing intervals which are not within interval $[l, r]$.

Optimal Recursive Substructure

For $I \in S$ we have $EMWNOSI(S; l; r)\ =$

- If $s(I) < l$ or $f(I) > r$ then $EMWNOSI(S - \{I\}; l; r)$

Solution 2: Adding Extra Parameters

Let $EMWNOSI(I_1, \ldots, I_n;\ l;\ r)$ denote max weight subset of non-overlapping intervals from the list after removing intervals which are not within interval $[l, r]$.

Optimal Recursive Substructure

For $I \in S$ we have $EMWNOSI(S; l; r)$ =

- If $s(I) < l$ or $f(I) > r$ then $EMWNOSI(S - \{I\}; l; r)$
- Otherwise *Max of*
  $EMWNOSI(S - \{I\}; l; r)$ and
  $EMWNOSI(S - \{I\}, l, s(I)) + w(I) + EMWNOSI(S - \{I\}, f(I), r)$

Solution 2: Adding Extra Parameters

Let $EMWNOSI(I_1, \ldots, I_n;\ l;\ r)$ denote max weight subset of non-overlapping intervals from the list after removing intervals which are not within interval $[l, r]$.

Optimal Recursive Substructure

For $I \in S$ we have $EMWNOSI(S; l; r) \; =$

- If $s(I) < l$ or $f(I) > r$ then $EMWNOSI(S - \{I\}; l; r)$
- Otherwise *Max* of

$$EMWNOSI(S - \{I\}; l; r) \text{ and}$$
$$EMWNOSI(S - \{I\}; l; s(I)) \; + \; EMWNOSI(S - \{I\}; f(I); r) \; + \; w(I)$$

Solution 2: Adding Extra Parameters

Let $EMWNOSI(I_1, \ldots, I_n;\ l;\ r)$ denote max weight subset of non-overlapping intervals from the list after removing intervals which are not within interval $[l, r]$.

Optimal Recursive Substructure

For $I \in S$ we have $EMWNOSI(S; l; r)\ =$

- If $s(I) < l$ or $f(I) > r$ then $EMWNOSI(S - \{I\}; l; r)$
- Otherwise *Max of*

$$EMWNOSI(S - \{I\}; l; r) \text{ and}$$

$$EMWNOSI(S - \{I\}; l; s(I)) + EMWNOSI(S - \{I\}; f(I); r) + w(I)$$

Base case:

Memoization Table m? dimensions?

Order of filling m?

Complexity with Memoization? $\Theta(n^3)$

Knapsack

Problem [KT6.4] Given a set of items $1..n$ items where item i has non-negative weight $w(i)$ and value $v(i)$, and a knapsack with capacity W, find a subset S of $1..n$ with total weight $(\Sigma_{i \in S}\ w(i)) \leq W$ such that the total value $(\Sigma_{i \in S}\ v(i))$ is maximized.
For simplicity assume that W as well as each $w(i)$ is an integer.

Knapsack

Problem [KT6.4] Given a set of items $1..n$ items where item i has non-negative weight $w(i)$ and value $v(i)$, and a knapsack with capacity W, find a subset S of $1..n$ with total weight $(\Sigma_{i \in S}\ w(i)) \leq W$ such that the total value $(\Sigma_{i \in S}\ v(i))$ is maximized.
For simplicity assume that W as well as each $w(i)$ is an integer.

- Greedy rule which gives optimal solution?

Knapsack

Problem [KT6.4] Given a set of items $1..n$ items where item i has non-negative weight $w(i)$ and value $v(i)$, and a knapsack with capacity W, find a subset S of $1..n$ with total weight $(\Sigma_{i \in S}\ w(i)) \leq W$ such that the total value $(\Sigma_{i \in S}\ v(i))$ is maximized.

For simplicity assume that W as well as each $w(i)$ is an integer.

- Greedy rule which gives optimal solution?
- Select items in order of weights (lightest item first).

Problem [KT6.4] Given a set of items $1..n$ items where item i has non-negative weight $w(i)$ and value $v(i)$, and a knapsack with capacity W, find a subset S of $1..n$ with total weight $(\Sigma_{i \in S}\ w(i)) \leq W$ such that the total value $(\Sigma_{i \in S}\ v(i))$ is maximized.
For simplicity assume that W as well as each $w(i)$ is an integer.

- Greedy rule which gives optimal solution?
- Select items in order of weights (lightest item first). Weight $(W/2+1,\ W/2,\ W/2)$ with value $(1,1,1)$.

Knapsack

Problem [KT6.4] Given a set of items $1..n$ items where item i has non-negative weight $w(i)$ and value $v(i)$, and a knapsack with capacity W, find a subset S of $1..n$ with total weight $(\Sigma_{i \in S}\ w(i)) \leq W$ such that the total value $(\Sigma_{i \in S}\ v(i))$ is maximized.

For simplicity assume that W as well as each $w(i)$ is an integer.

- Greedy rule which gives optimal solution?
- Select items in order of weights (lightest item first). Weight $(W/2 + 1,\ W/2,\ W/2)$ with value $(1, 1, 1)$.
- Select items in decreasing order of value (most expensive first).

Knapsack

Problem [KT6.4] Given a set of items $1..n$ items where item i has non-negative weight $w(i)$ and value $v(i)$, and a knapsack with capacity W, find a subset S of $1..n$ with total weight $(\Sigma_{i \in S}\ w(i)) \leq W$ such that the total value $(\Sigma_{i \in S}\ v(i))$ is maximized.

For simplicity assume that W as well as each $w(i)$ is an integer.

- Greedy rule which gives optimal solution?
- Select items in order of weights (lightest item first). Weight $(W/2+1,\ W/2,\ W/2)$ with value $(1,1,1)$.
- Select items in decreasing order of value (most expensive first). Value $(3,2,2)$ with weight $(W/2+1,\ W/2,\ W/2)$.

Knapsack

Problem [KT6.4] Given a set of items $1..n$ items where item i has non-negative weight $w(i)$ and value $v(i)$, and a knapsack with capacity W, find a subset S of $1..n$ with total weight $(\Sigma_{i \in S}\ w(i)) \leq W$ such that the total value $(\Sigma_{i \in S}\ v(i))$ is maximized.

For simplicity assume that W as well as each $w(i)$ is an integer.

- Greedy rule which gives optimal solution?
- Select items in order of weights (lightest item first). Weight $(W/2+1,\ W/2,\ W/2)$ with value $(1,1,1)$.
- Select items in decreasing order of value (most expensive first). Value $(3,2,2)$ with weight $(W/2+1,\ W/2,\ W/2)$.
- No known Greedy rule gives optimal solution.

Optimal Substructure of Knapsack

Optimal Substructure of Knapsack

- Let $opt(i, w)$ denote value of maximum value subset S of $1..i$ where $w(S) \leq w$. Define this for $0 \leq w \leq W$.

Optimal Substructure of Knapsack

- Let $opt(i, w)$ denote value of maximum value subset S of $1..i$ where $w(S) \leq w$. Define this for $0 \leq w \leq W$.
- If $w < w(i)$ then
  $opt(i, w) =$

Optimal Substructure of Knapsack

- Let $opt(i, w)$ denote value of maximum value subset S of $1..i$ where $w(S) \leq w$. Define this for $0 \leq w \leq W$.
- If $w < w(i)$ then

$$opt(i, w) \;=\; opt(i-1, w)$$

Optimal Substructure of Knapsack

- Let $opt(i, w)$ denote value of maximum value subset S of $1..i$ where $w(S) \leq w$. Define this for $0 \leq w \leq W$.
- If $w < w(i)$ then
$$opt(i, w) \; = \; opt(i-1, w)$$
- Otherwise $opt(i, w) \; =$ $\max\big(opt(i-1, w),\; w(i) + opt(i-1, w - w(i))\big)$

Optimal Substructure of Knapsack

- Let $opt(i, w)$ denote value of maximum value subset S of $1..i$ where $w(S) \leq w$. Define this for $0 \leq w \leq W$.
- If $w < w(i)$ then
  $$opt(i, w) \;=\; opt(i-1, w)$$
- Otherwise $opt(i, w) \;=\; max$ of
  $opt(i-1, w)$, and
  $w(i) \;+\; opt(i-1, w-w(i))$

$i \leftarrow 0..n$

$0 \leq w \leq W$

Optimal Substructure of Knapsack

- Let $opt(i, w)$ denote value of maximum value subset S of $1..i$ where $w(S) \leq w$. Define this for $0 \leq w \leq W$.
- If $w < w(i)$ then
$$opt(i, w) = opt(i-1, w)$$
- Otherwise $opt(i, w) = max$ of
$opt(i-1, w)$, and
$w(i) + opt(i-1, w-w(i))$
- Base cases: $opt(0, w) = 0$.

Optimal Substructure of Knapsack

- Let $opt(i, w)$ denote value of maximum value subset S of $1..i$ where $w(S) \leq w$. Define this for $0 \leq w \leq W$.
- If $w < w(i)$ then
$$opt(i, w) = opt(i-1, w)$$
- Otherwise $opt(i, w) = max$ of
$opt(i-1, w)$, and
$w(i) + opt(i-1, w-w(i))$
- Base cases: $opt(0, w) = 0$.
- Dimensions of memoization table m?

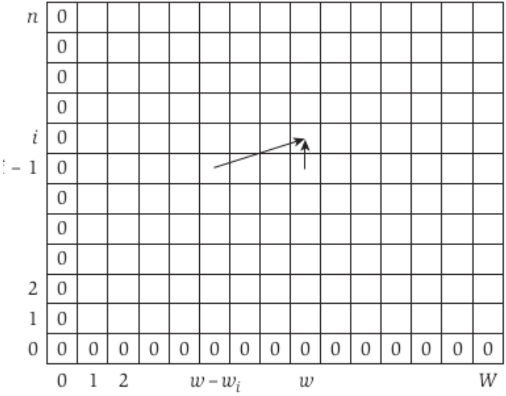
Optimal Substructure of Knapsack

- Let $opt(i, w)$ denote value of maximum value subset S of $1..i$ where $w(S) \leq w$. Define this for $0 \leq w \leq W$.
- If $w < w(i)$ then
  $$opt(i, w) = opt(i-1, w)$$
- Otherwise $opt(i, w) = max$ of
  $opt(i-1, w)$, and
  $w(i) + opt(i-1, w-w(i))$
- Base cases: $opt(0, w) = 0$.
- Dimensions of memoization table m?
- Complexity?

Optimal Substructure of Knapsack

- Let $opt(i, w)$ denote value of maximum value subset S of $1..i$ where $w(S) \leq w$. Define this for $0 \leq w \leq W$.
- If $w < w(i)$ then
$$opt(i, w) = opt(i-1, w)$$
- Otherwise $opt(i, w) = max$ of
  $opt(i-1, w)$, and
  $w(i) + opt(i-1, w-w(i))$
- Base cases: $opt(0, w) = 0$.
- Dimensions of memoization table m?
- Complexity? $O(n \cdot W)$
  Pseudo-polynomial – proportional to value of constant occuring in input.

Knapsack Bottom Up Schedule



Detecting Subsequence

Problem Given sequences (arrays) $X = \langle x_1, x_2, \ldots, x_m \rangle$ and sequence $Z = \langle z_1, \ldots, z_k \rangle$, determine whether Z is a subsequence of X, that is there exists a sequence of indices $\langle i_1, \ldots, i_k$ such that $Z_j = X_{i_j}$.

Detecting Subsequence

Problem Given sequences (arrays) $X = \langle x_1, x_2, \ldots, x_m \rangle$ and sequence $Z = \langle z_1, \ldots, z_k \rangle$, determine whether Z is a subsequence of X, that is there exists a sequence of indices $\langle i_1, \ldots, i_k$ such that $Z_j = X_{i_j}$.
Solution: Greedy Algorithm of Complexity $\Theta(m)$

Longest Common Subsequence (LCS)

Problem Given sequences (arrays) $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n$, determine the longest length sequence $Z = \langle z_1, \ldots, z_k \rangle$ which is a subsequence of both X and Y.

Longest Common Subsequence (LCS)

Problem Given sequences (arrays) $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n$, determine the longest length sequence $Z = \langle z_1, \ldots, z_k \rangle$ which is a subsequence of both X and Y.

Brute Force Solution

Systematically generate all subsequences Z of Y. For each check if Z is a subseqeunce of X. Also remember the maximum of the length of "yes" subsequences examined so far.

Complexity?

Optimal Substructure

Optimal Substructure

Notation: Given $X = \langle x_1, x_2, \ldots, x_m \rangle$, the ith prefix of X is $X_i = \langle x_1, x_2, \ldots, x_i \rangle$.

Optimal Substructure

Notation: Given $X = \langle x_1, x_2, \ldots, x_m \rangle$, the ith prefix of X is $X_i = \langle x_1, x_2, \ldots, x_i \rangle$.

Theorem

Given sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, if $Z = \langle z_1, \ldots, z_k \rangle$ is their LCS, then

Optimal Substructure

Notation: Given $X = \langle x_1, x_2, \ldots, x_m \rangle$, the ith prefix of X is $X_i = \langle x_1, x_2, \ldots, x_i \rangle$.

Theorem

Given sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, if $Z = \langle z_1, \ldots, z_k \rangle$ is their LCS, then

- If $x_m = y_n$ then

Optimal Substructure

Notation: Given $X = \langle x_1, x_2, \ldots, x_m \rangle$, the ith prefix of X is $X_i = \langle x_1, x_2, \ldots, x_i \rangle$.

Theorem

Given sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, if $Z = \langle z_1, \ldots, z_k \rangle$ is their LCS, then

- If $x_m = y_n$ then $z_k = x_m = y_n$ and Z_{k-l} is LCS of X_{m-1}, Y_{n-1}.

Notation: Given $X = \langle x_1, x_2, \ldots, x_m \rangle$, the ith prefix of X is $X_i = \langle x_1, x_2, \ldots, x_i \rangle$.

Theorem

Given sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, if $Z = \langle z_1, \ldots, z_k \rangle$ is their LCS, then

- If $x_m = y_n$ then $z_k = x_m = y_n$ and Z_{k-l} is LCS of X_{m-1}, Y_{n-1}.
- If $x_m \neq y_n$ and $z_k \neq x_m$ then

Optimal Substructure

Notation: Given $X = \langle x_1, x_2, \ldots, x_m \rangle$, the ith prefix of X is $X_i = \langle x_1, x_2, \ldots, x_i \rangle$.

Theorem

Given sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, if $Z = \langle z_1, \ldots, z_k \rangle$ is their LCS, then

- If $x_m = y_n$ then $z_k = x_m = y_n$ and Z_{k-l} is LCS of X_{m-1}, Y_{n-1}.
- If $x_m \neq y_n$ and $z_k \neq x_m$ then Z is LCS of X_{m-1}, Y.

Notation: Given $X = \langle x_1, x_2, \ldots, x_m \rangle$, the *i*th prefix of X is $X_i = \langle x_1, x_2, \ldots, x_i \rangle$.

Theorem

Given sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, if $Z = \langle z_1, \ldots, z_k \rangle$ is their LCS, then

- If $x_m = y_n$ then $z_k = x_m = y_n$ and Z_{k-l} is LCS of X_{m-1}, Y_{n-1}.
- If $x_m \neq y_n$ and $z_k \neq x_m$ then Z is LCS of X_{m-1}, Y.
- If $x_m \neq y_n$ and $z_k \neq y_n$ then Z is LCS of X_{m-1}, Y.

Recursive Solution

Let $c[i,j]$ denote the length of LCS of X_i and Y_j. Then,
$c[i,j] \ =$

Recursive Solution

Let $c[i, j]$ denote the length of LCS of X_i and Y_j. Then,
$c[i, j] =$

- 0 if $i = 0$ or $j = 0$

Recursive Solution

Let $c[i,j]$ denote the length of LCS of X_i and Y_j. Then,
$c[i,j] =$

- 0 if $i = 0$ or $j = 0$
- $c[i-1, j-1] + 1$ if $i > 0 \wedge j > 0$ and $x_i = y_j$

Recursive Solution

Let $c[i,j]$ denote the length of LCS of X_i and Y_j. Then,
$c[i,j] \ =$

- 0 if $i = 0$ or $j = 0$
- $c[i-1, j-1] + 1$ if $i > 0 \wedge j > 0$ and $x_i = y_j$
- $max(c[i-1, j], c[i, j-1]$ if $i > 0 \wedge j > 0$ and $x_i \neq y_j$

Let $c[i,j]$ denote the length of LCS of X_i and Y_j. Then,
$c[i,j] \ =$

- 0 if $i = 0$ or $j = 0$
- $c[i-1, j-1] + 1$ if $i > 0 \wedge j > 0$ and $x_i = y_j$
- $max(c[i-1,j], c[i,j-1]$ if $i > 0 \wedge j > 0$ and $x_i \neq y_j$

Designing the DP algorithm

- Memoization table $c[0..m, 0..n]$.

Let $c[i,j]$ denote the length of LCS of X_i and Y_j. Then,
$c[i,j] =$

- 0 if $i = 0$ or $j = 0$
- $c[i-1, j-1] + 1$ if $i > 0 \wedge j > 0$ and $x_i = y_j$
- $max(c[i-1, j], c[i, j-1]$ if $i > 0 \wedge j > 0$ and $x_i \neq y_j$

Designing the DP algorithm

- Memoization table $c[0..m, 0..n]$.
- Base case?, Order of computing $c[i,j]$?

Recursive Solution

Let $c[i, j]$ denote the length of LCS of X_i and Y_j. Then,
$c[i, j] \ =$

- 0 if $i = 0$ or $j = 0$
- $c[i-1, j-1] + 1$ if $i > 0 \wedge j > 0$ and $x_i = y_j$
- $max(c[i-1, j], c[i, j-1]$ if $i > 0 \wedge j > 0$ and $x_i \neq y_j$

Designing the DP algorithm

- Memoization table $c[0..m, 0..n]$.
- Base case?, Order of computing $c[i, j]$?
- Complexity?

Recursive Solution

Let $c[i,j]$ denote the length of LCS of X_i and Y_j. Then,
$c[i,j] \ =$

- 0 if $i = 0$ or $j = 0$
- $c[i-1, j-1] + 1$ if $i > 0 \wedge j > 0$ and $x_i = y_j$
- $max(c[i-1, j], c[i, j-1]$ if $i > 0 \wedge j > 0$ and $x_i \neq y_j$

Designing the DP algorithm

- Memoization table $c[0..m, 0..n]$.
- Base case?, Order of computing $c[i,j]$?
- Complexity? $\Theta(m \cdot n)$

Bottom UP Procedure

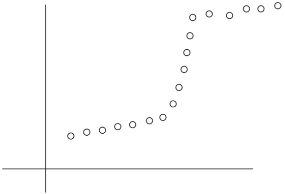
```
LCS-LENGTH(X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```

Constructing the Solution

i	j	0	1	2	3	4	5	6
	y_j		B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4

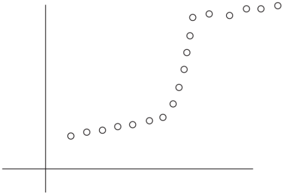
Segmented Least Square

Problem [KT6.3] Given a set of n points $(x_1, y_1), \ldots, (x_n, y_n)$ in x, y-plane in order $x_1 < x_2 < \ldots < x_n$, find a small set of line segments such that the soln. gives the least error squared.

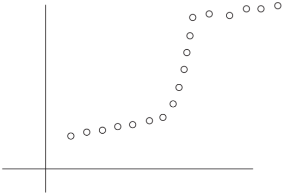


Segmented Least Square

Problem [KT6.3] Given a set of n points $(x_1, y_1), \ldots, (x_n, y_n)$ in x, y-plane in order $x_1 < x_2 < \ldots < x_n$, find a small set of line segments such that the soln. gives the least error squared.



Problem [KT6.3] Given a set of n points $(x_1, y_1), \ldots, (x_n, y_n)$ in x, y-plane in order $x_1 < x_2 < \ldots < x_n$, find a small set of line segments such that the soln. gives the least error squared.



.

Formulating the Problem

- if $p_i, p_{i+1}, \ldots, p_j$ belongs to a line segment then $e_{i,j}$ denotes the squared error from these points after fitting the best line through them.
- Trade off between reducing error and reducing number of line segments.
- Each line segment incurs a cost of C.

Formulating the Problem

- if $p_i, p_{i+1}, \ldots, p_j$ belongs to a line segment then $e_{i,j}$ denotes the squared error from these points after fitting the best line through them.
- Trade off between reducing error and reducing number of line segments.
- Each line segment incurs a cost of C.

Formulating the Problem

- if $p_i, p_{i+1}, \ldots, p_j$ belongs to a line segment then $e_{i,j}$ denotes the squared error from these points after fitting the best line through them.
- Trade off between reducing error and reducing number of line segments.
- Each line segment incurs a cost of C.

Optimal Substructure

For the subproblem $p_1, \ldots, p_j$

$$opt(j) \;=\; min_{1 \leq i \leq j} \;\; e_{i,j} + C + opt(i-1)$$