CS 473ug: Algorithms

Chandra Chekuri
chekuri@cs.uiuc.edu
3228 Siebel Center

University of Illinois, Urbana-Champaign

Fall 2007

Part I

Longest Increasing Subsequence

Longest Increasing Subsequence

Sequence: an ordered list $a_1, a_2, \ldots, a_n$. *Length* of sequence is n

$a_{i_1}, a_{i_2}, \ldots, a_{ik}$ is a subsequence of $a_1, a_2, \ldots, a_n$ if $1 \leq i_1 < i_2 < \ldots < i_k \leq n$.

A sequence is *increasing* if $a_1 < a_2 < \ldots < a_n$. It is *non-decreasing* if $a_1 \leq a_2 \leq \ldots \leq a_n$. Similarly *decreasing* and *non-increasing*.

Longest Increasing Subsequence

Sequence: an ordered list $a_1, a_2, \ldots, a_n$. *Length* of sequence is n

$a_{i_1}, a_{i_2}, \ldots, a_{ik}$ is a subsequence of $a_1, a_2, \ldots, a_n$ if $1 \leq i_1 < i_2 < \ldots < i_k \leq n$.

A sequence is *increasing* if $a_1 < a_2 < \ldots < a_n$. It is *non-decreasing* if $a_1 \leq a_2 \leq \ldots \leq a_n$. Similarly *decreasing* and *non-increasing*.

Longest Increasing Subsequence

Input A sequence of numbers $a_1, a_2, \ldots, a_n$

Goal Find an *increasing subsequence* $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ of maximum length

Example

6 5 3 2 7 8 1 10

Subsequence: 5 2 1

Increasing Subsequence: 5 7 8

Longest Increasing Subsequence: 6 7 8 10

also 5 7 8 10

3 7 8 10 etc

A First Recursive Approach

$L(i)$: length of longest common subsequence in first i elements $a_1, a_2, \ldots, a_i$.

A First Recursive Approach

$L(i)$: length of longest common subsequence in first i elements $a_1, a_2, \ldots, a_i$.

Can we write $L(i)$ in terms of $L(1), L(2), \ldots, L(i-1)$?

Case 1 : $L(i)$ does not contain a_i, then $L(i) = L(i-1)$
Case 2 : $L(i)$ contains a_i, then $L(i) = ?$

A First Recursive Approach

$L(i)$: length of longest common subsequence in first i elements $a_1, a_2, \ldots, a_i$.

Can we write $L(i)$ in terms of $L(1), L(2), \ldots, L(i-1)$?

Case 1 : $L(i)$ does not contain a_i, then $L(i) = L(i-1)$

Case 2 : $L(i)$ contains a_i, then $L(i) = ?$ What is the element in the subsequence before a_i? If it is a_j then it better be the case that $a_j < a_i$ since we are looking for an increasing sequence.

A First Recursive Approach

$L(i)$: length of longest common subsequence in first i elements $a_1, a_2, \ldots, a_i$.

Can we write $L(i)$ in terms of $L(1), L(2), \ldots, L(i-1)$?

Case 1 : $L(i)$ does not contain a_i, then $L(i) = L(i-1)$

Case 2 : $L(i)$ contains a_i, then $L(i) =?$ What is the element in the subsequence before a_i? If it is a_j then it better be the case that $a_j < a_i$ since we are looking for an increasing sequence. Do we know which j? No!

A First Recursive Approach

$L(i)$: length of longest common subsequence in first i elements $a_1, a_2, \ldots, a_i$.

Can we write $L(i)$ in terms of $L(1), L(2), \ldots, L(i-1)$?

Case 1 : $L(i)$ does not contain a_i, then $L(i) = L(i-1)$

Case 2 : $L(i)$ contains a_i, then $L(i) =$? What is the element in the subsequence before a_i? If it is a_j then it better be the case that $a_j < a_i$ since we are looking for an increasing sequence. Do we know which j? No! So we try all possibilities

$$L(i) = 1 + \max_{j<i \text{ and } a_j<a_i} L(j)$$

A First Recursive Approach

$L(i)$: length of longest common subsequence in first i elements $a_1, a_2, \ldots, a_i$.

Can we write $L(i)$ in terms of $L(1), L(2), \ldots, L(i-1)$?

Case 1 : $L(i)$ does not contain a_i, then $L(i) = L(i-1)$

Case 2 : $L(i)$ contains a_i, then $L(i) =?$ What is the element in the subsequence before a_i? If it is a_j then it better be the case that $a_j < a_i$ since we are looking for an increasing sequence. Do we know which j? No! So we try all possibilities

$$L(i) = 1 + \max_{j<i \text{ and } a_j<a_i} L(j)$$

Is the above correct?

A First Recursive Approach

$L(i)$: length of longest common subsequence in first i elements $a_1, a_2, \ldots, a_i$.

Can we write $L(i)$ in terms of $L(1), L(2), \ldots, L(i-1)$?

Case 1 : $L(i)$ does not contain a_i, then $L(i) = L(i-1)$

Case 2 : $L(i)$ contains a_i, then $L(i) =$? What is the element in the subsequence before a_i? If it is a_j then it better be the case that $a_j < a_i$ since we are looking for an increasing sequence. Do we know which j? No! So we try all possibilities

$$L(i) = 1 + \max_{j<i \text{ and } a_j<a_i} L(j)$$

Is the above correct? No, because we do not know that $L(j)$ is a subsequence that actually ends at a_j!

A Correct Recursion

$L(i)$: longest common subsequence in first i elements $a_1, a_2, \ldots, a_i$ **that ends** in a_i

A Correct Recursion

$L(i)$: longest common subsequence in first i elements $a_1, a_2, \ldots, a_i$ **that ends** in a_i

$$L(i) = 1 + \max_{j < i \text{ and } a_j < a_i} L(j)$$

What is the final solution?

$$\begin{tikzcd} {} & a_j \arrow[l] & a_i \arrow[l, bend right] \end{tikzcd}$$

A Correct Recursion

$L(i)$: longest common subsequence in first i elements $a_1, a_2, \ldots, a_i$ **that ends** in a_i

$$L(i) = 1 + \max_{j<i \text{ and } a_j<a_i} L(j)$$

What is the final solution? $\max_{i=1}^{n} L(i)$

How many subproblems?

A Correct Recursion

$L(i)$: longest common subsequence in first i elements $a_1, a_2, \ldots, a_i$ **that ends** in a_i

$$L(i) = 1 + \max_{j<i \text{ and } a_j<a_i} L(j)$$

What is the final solution? $\max_{i=1}^{n} L(i)$

How many subproblems? $O(n)$

Table based Iterative Algorithm

Recurrence:

$$L(i) = 1 + \max_{j<i \text{ and } a_j<a_i} L(j)$$

Iterative algorithm:

```
for i = 1 to n do
    L[i] = 1
    for j = 1 to i − 1 do
        if a_j < a_i and 1 + L[j] > L[i]
            L[i] = 1 + L[j]
```

Output $\max_{i=1}^{n} L[i]$

Running Time:

Table based Iterative Algorithm

Recurrence:

$$L(i) = 1 + \max_{j<i \text{ and } a_j<a_i} L(j)$$

Iterative algorithm:

for $i = 1$ to n do
    $L[i] = 1$
    for $j = 1$ to $i - 1$ do
        if $a_j < a_i$ *and* $1 + L[j] > L[i]$
            $L[i] = 1 + L[j]$

Output $\max_{i=1}^{n} L[i]$

Running Time: $O(n^2)$
Space:

Table based Iterative Algorithm

Recurrence:

$$L(i) = 1 + \max_{j<i \text{ and } a_j<a_i} L(j)$$

Iterative algorithm:

for $i = 1$ to n do
    $L[i] = 1$
    for $j = 1$ to $i - 1$ do
        if $a_j < a_i$ *and* $1 + L[j] > L[i]$
            $L[i] = 1 + L[j]$

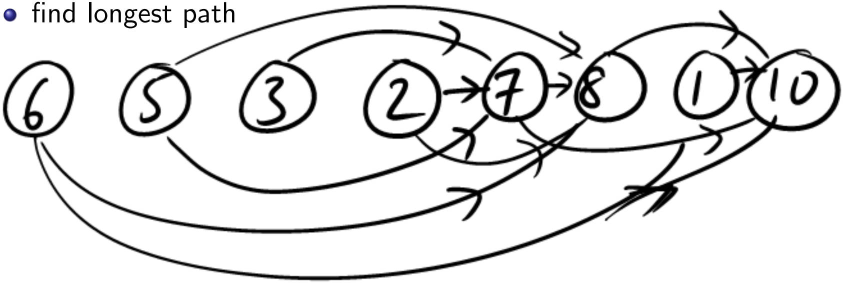
Output $\max_{i=1}^{n} L[i]$

Running Time: $O(n^2)$
Space: $O(n)$

A DAG Based Approach

Given sequence $a_1, a_2, \ldots, a_n$ create DAG as follows:

- for each i there is a node v_i
- if $i < j$ and $a_i < a_j$ add an edge (v_i, v_j)
- find longest path



Dynamic Programming Methods

There is always a DAG of computation underlying every dynamic program but it is not always clear.

Dynamic Programming Methods

There is always a DAG of computation underlying every dynamic program but it is not always clear.

Three ways to come up with dynamic programming solutions

- recursion followed by memoization with a polynomial number of subproblems
- bottom up via tables
- identify a DAG followed by a shortest/longest path in DAG

But there is always a recursion+memoization that is implicit in the other methods!

Dilworth's Theorem

Given sequence $a_1, a_2, \ldots, a_n$ the longest increasing subsequence can be of size 1.

Dilworth's Theorem

Given sequence $a_1, a_2, \ldots, a_n$ the longest increasing subsequence can be of size 1.

Example: $a_1 > a_2 > a_3 > \ldots > a_n$

Dilworth's Theorem

Given sequence $a_1, a_2, \ldots, a_n$ the longest increasing subsequence can be of size 1.

Example: $a_1 > a_2 > a_3 > \ldots > a_n$

In the above example the longest *decreasing* sequence is of size n.

Dilworth's Theorem

Given sequence $a_1, a_2, \ldots, a_n$ the longest increasing subsequence can be of size 1.
Example: $a_1 > a_2 > a_3 > \ldots > a_n$
In the above example the longest *decreasing* sequence is of size n.

Question: Given a sequence of length n, can the longest increasing sequence and the longest decreasing sequence be both small?

Dilworth's Theorem

Given sequence $a_1, a_2, \ldots, a_n$ the longest increasing subsequence can be of size 1.
Example: $a_1 > a_2 > a_3 > \ldots > a_n$
In the above example the longest *decreasing* sequence is of size n.

Question: Given a sequence of length n, can the longest increasing sequence and the longest decreasing sequence be both small?

at least

Dilworth's Theorem Consequence: In any sequence of length n, either the longest increasing sequence or the longest decresing sequence is of length $\lfloor \sqrt{n} \rfloor + 1$.

Exercise: give a sequence of length n in which both the longest increasing subsequence and longest decreasing susbequence are no more than $\lfloor \sqrt{n} \rfloor + 1$.

Part II

Edit Distance

Spell Checking Problem

Given a string "exponen" that is not in the dictionary, how should a spell checker suggest a *nearby* string?

Spell Checking Problem

Given a string "exponen" that is not in the dictionary, how should a spell checker suggest a *nearby* string?

What does nearness mean?

Question: Given two strings $x_1 x_2 \ldots x_n$ and $y_1 y_2 \ldots y_m$ what is a *distance* between them?

Spell Checking Problem

Given a string "exponen" that is not in the dictionary, how should a spell checker suggest a *nearby* string?

What does nearness mean?

Question: Given two strings $x_1 x_2 \ldots x_n$ and $y_1 y_2 \ldots y_m$ what is a *distance* between them?

Edit Distance: minimum number of "edits" to transform x into y.

Edit Distance

Edit Distance: minimum number of "edits" to transform x into y.

Edit operations:

- delete a letter
- add a letter
- substitute a letter with another letter

Edit Distance

Edit Distance: minimum number of "edits" to transform x into y.

Edit operations:

- delete a letter
- add a letter
- substitute a letter with another letter

Why is substitute not "delete plus add"?

In general different edits can have different *costs* and using substitution as a edit allows a single operation as far as distance is concerned

Example

FOO D

MONE Y

$\underline{F}OOD \rightarrow MOOD \rightarrow MOOD\underline{Y}$

$\rightarrow MONDY \rightarrow MONEY$

Edit Distance as Alignment

F O O _ D

M O N E Y

Edit Distance Problem

Input Two strings $x = x_1 x_2 \ldots x_n$ and $y = y_1 y_2 \ldots y_m$ over some fixed alphabet Σ

Goal Find edit distance between x and y: minimum number of edits to transform x into y

Edit Distance Problem

Input: Two strings $x = x_1 x_2 \ldots x_n$ and $y = y_1 y_2 \ldots y_m$ over some fixed alphabet Σ

Goal: Find edit distance between x and y: minimum number of edits to transform x into y

Note: EditDist(x,y) = EditDist(y,x)

Towards a Recursive Solution

Think of aligning the strings. Can we express the full problem as a function of subproblems?

$$x_1 \; - \; x_2 \; x_3 \; - \; x_4 \; \cdots \; x_n$$
$$y_1 \; - \; y_2 \; y_3 \; \cdots \; y_m \; -$$

AFOOD _

_FOODY

Towards a Recursive Solution

Think of aligning the strings. Can we express the full problem as a function of subproblems?

Case 1 x_n is aligned with y_m. Then $x_1 x_2 \ldots x_{n-1}$ is aligned with $y_1 y_2 \ldots y_{m-1}$

Towards a Recursive Solution

Think of aligning the strings. Can we express the full problem as a function of subproblems?

Case 1 x_n is aligned with y_m. Then $x_1 x_2 \ldots x_{n-1}$ is aligned with $y_1 y_2 \ldots y_{m-1}$

Case 2a x_n is aligned with _ and y_m is to left of _. Then $x_1 x_2 \ldots x_{n-1}$ is aligned with $y_1 y_2 \ldots y_{\cancel{n}}$. m

Towards a Recursive Solution

Think of aligning the strings. Can we express the full problem as a function of subproblems?

Case 1 x_n is aligned with y_m. Then $x_1 x_2 \ldots x_{n-1}$ is aligned with $y_1 y_2 \ldots y_{m-1}$

Case 2a x_n is aligned with $\_$ and y_m is to left of $\_$. Then $x_1 x_2 \ldots x_{n-1}$ is aligned with $y_1 y_2 \ldots y_n$.

Case 2b y_m is aligned with $\_$ and x_n is to left of $\_$. Then $x_1 x_2 \ldots x_n$ is aligned with $y_1 y_2 \ldots y_{m-1}$.

Towards a Recursive Solution

Think of aligning the strings. Can we express the full problem as a function of subproblems?

Case 1 x_n is aligned with y_m. Then $x_1 x_2 \ldots x_{n-1}$ is aligned with $y_1 y_2 \ldots y_{m-1}$

Case 2a x_n is aligned with _ and y_m is to left of _. Then $x_1 x_2 \ldots x_{n-1}$ is aligned with $y_1 y_2 \ldots y_n$.

Case 2b y_m is aligned with _ and x_n is to left of _. Then $x_1 x_2 \ldots x_n$ is aligned with $y_1 y_2 \ldots y_{m-1}$.

Subproblems involve aligning *prefixes* of the two strings.
Find edit distance between prefix $x[1..i]$ of x and prefix $y[1..j]$ of y
EditDist(x,y) is the distance between $x[1..n]$ and $y[1..m]$

A Recursive Solution

$E[i, j]$: edit distance between $x[1..i]$ and $y[1..j]$

A Recursive Solution

$E[i,j]$: edit distance between $x[1..i]$ and $y[1..j]$

Case 1 x_i aligned with $\cancel{x_j}$, y_j

$$E[i,j] = diff(x_i, y_j) + E[i-1, j-1]$$

where $diff(x_i, y_j) = 0$ if $x_i = y_j$, otherwise $diff(x_i, \cancel{x_j}) = 1$ y_j

A Recursive Solution

$E[i,j]$: edit distance between $x[1..i]$ and $y[1..j]$

Case 1 x_i aligned with x_j.

$$E[i,j] = \mathit{diff}(x_i, y_j) + E[i-1, j-1]$$

where $\mathit{diff}(x_i, y_j) = 0$ if $x_i = y_j$, otherwise $\mathit{diff}(x_i, x_j) = 1$

Case 2a x_i is mapped to _ and x_i is to right of $\cancel{x_j}$ y_j

$$E[i,j] = 1 + E[i-1, j]$$

A Recursive Solution

$E[i,j]$: edit distance between $x[1..i]$ and $y[1..j]$

Case 1 x_i aligned with x_j.

$$E[i,j] = \mathit{diff}(x_i, y_j) + E[i-1, j-1]$$

where $\mathit{diff}(x_i, y_j) = 0$ if $x_i = y_j$, otherwise $\mathit{diff}(x_i, x_j) = 1$

Case 2a x_i is mapped to _ and x_i is to right of x_j

$$E[i,j] = 1 + E[i-1, j]$$

Case 2b y_j is mapped to _ and x_i is to left of y_j

$$E[i,j] = 1 + E[i, j-1]$$

A Recursive Solution

$$E[i,j] = \min\{diff(x_i, y_j) + E[i-1, j-1], 1 + E[i-1, j], 1 + E[i, j-1]$$

Base cases:

A Recursive Solution

$$E[i,j] = \min\{diff(x_i, y_j) + E[i-1, j-1], 1 + E[i-1, j], 1 + E[i, j-1]$$

Base cases:

$$E[i,0] = i \text{ for } i \geq 0$$

$$E[0,j] = j \text{ for } i \geq 0$$

How many subproblems?

A Recursive Solution

$$E[i,j] = \min\{diff(x_i, y_j) + E[i-1, j-1], 1 + E[i-1, j], 1 + E[i, j-1]$$

Base cases:

$$E[i,0] = i \text{ for } i \geq 0$$

$$E[0,j] = j \text{ for } i \geq 0$$

How many subproblems? $O(mn)$

Iterative Solution

What is table?

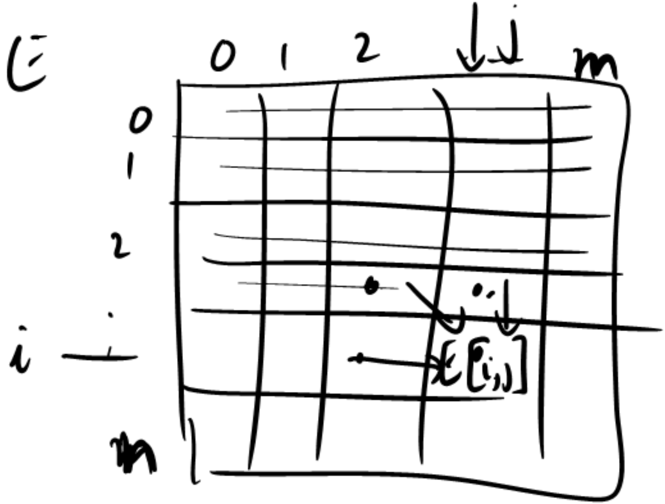
Iterative Solution

What is table? E is a two-dimensional array of size $(n+1)(m+1)$
How do we order the computation?

Iterative Solution

What is table? E is a two-dimensional array of size $(n+1)(m+1)$
How do we order the computation?
To compute $E[i,j]$ need to have computed $E[i-1,j-1], E[i-1,j], E[i,j-1]$.

Iterative Solution



Iterative Solution

```
for i = 0 to n
    E[i, 0] = i

for j = 0 to m
    E[0, j] = j

for i = 1 to n do
    for j = 1 to m do
        E[i, j] = min{diff(x_i, y_j) + E[i, j], 1 + E[i − 1, j], 1 + E[i, j − 1]}
```

for $i = 0$ to n
 $E[i, 0] = i$

for $j = 0$ to m
 $E[0, j] = j$

for $i = 1$ to n do
 for $j = 1$ to m do
 $E[i, j] = \min\{diff(x_i, y_j) + E[i, j], 1 + E[i-1, j], 1 + E[i, j-1]\}$

Running Time:

Iterative Solution

```
for i = 0 to n
    E[i,0] = i

for j = 0 to m
    E[0,j] = j

for i = 1 to n do
    for j = 1 to m do
        E[i,j] = min{diff(x_i, y_j) + E[i,j], 1 + E[i-1,j], 1 + E[i,j-1]}
```

Running Time: $O(nm)$
Space:

Iterative Solution

for $i = 0$ to n
    $E[i, 0] = i$

for $j = 0$ to m
    $E[0, j] = j$

for $i = 1$ to n do
    for $j = 1$ to m do
        $E[i, j] = \min\{diff(x_i, y_j) + E[i, j], 1 + E[i-1, j], 1 + E[i, j-1]\}$

Running Time: $O(nm)$
Space: $O(nm)$

Iterative Solution

```
for i = 0 to n
    E[i,0] = i

for j = 0 to m
    E[0,j] = j

for i = 1 to n do
    for j = 1 to m do
        E[i,j] = min{diff(x_i, y_j) + E[i,j], 1 + E[i-1,j], 1 + E[i,j-1]}
```

Running Time: $O(nm)$
Space: $O(nm)$
Can reduce space to $O(n+m)$ if only distance is needed but not obvious how to actually compute the edits. Can do it with a clever scheme (see Jeff's notes).

Example

Where is the DAG?

- one node for each (i, j), $1 = 0 \leq i \leq n$, $0 \leq j \leq m$.
- Edges for node (i, j): from $(i-1, j-1)$ of cost $diff(x_i, y_j)$, from $(i-1, j)$ of cost 1, from $(i, j-1)$ of cost 1
- find *shortest* path from $(0, 0)$ to (n, m)