# CSE 417: Algorithms and Computational Complexity

Winter 2001

Lecture 7

Instructor: Paul Beame

TA: Gidon Shavit

1

## Three Steps to Dynamic Programming

- Formulate the answer as a recurrence relation or recursive algorithm
- Show that number of different parameter values in the recursive algorithm is bounded by a small polynomial
- Specify an order of evaluation for the recurrence so that already have the partial results ready when you need them.

2

## Sequence Comparison: Edit Distance

- Given:
  - Two strings of characters $A=a_1\ a_2\ \ldots\ a_n$ and $B=b_1\ b_2\ \ldots\ b_m$
- Find:
  - The minimum number of edit steps needed to transform A into B where an edit can be:
  - insert a single character
  - delete a single character
  - substitute one character by another

3

## Recursive Solution

- Sub-problems: Edit distance problems for all prefixes of A and B that don't include all of both A and B
- Let $D(i,j)$ be the number of edits required to transform $a_1\ a_2\ \ldots\ a_i$ into $b_1\ b_2\ \ldots\ b_j$
- Clearly $D(0,0)=0$

4

## Computing D(n,m)

- Imagine how best sequence handles the last characters $a_n$ and $b_m$
- If best sequence of operations
  - deletes $a_n$ then $D(n,m)=D(n-1,m)+1$
  - inserts $b_m$ then $D(n,m)=D(n,m-1)+1$
  - replaces $a_n$ by $b_m$ then $D(n,m)=D(n-1,m-1)+1$
  - matches $a_n$ and $b_m$ then $D(n,m)=D(n-1,m-1)$

5

## Recursive algorithm D(n,m)

- **if** n=0 **then**
  - **return** (m)
- **elseif** m=0 **then**
  - **return**(n)
- **else**
  - **if** $a_n=b_m$ **then**
    - replace-cost=0
  - **else**
    - replace-cost=1
  - **endif**
  - **return**(min{ D(n-1, m) + 1,
    D(n, m-1) +1,
    D(n-1, m-1) + replace-cost})

6

## Dynamic programming

```
for j = 0 to m;  D(0,j) ← j; endfor
for i = 1 to n;  D(i,0) ← i; endfor
for i = 1 to n
    for j = 1 to m
        if  a_i=b_j then
            replace-cost ← 0
        else
            replace-cost ← 1
        endif
        D(i,j) ←  min { D(i-1, j) + 1,
                        D(i, j-1) +1,
                        D(i-1, j-1) + replace-cost}
    endfor
endfor
```



## Example run with AGACATTG and GAGTTA

| | A | G | A | C | A | T | T | G |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |
| G | | | | | | | | |
| A | | | | | | | | |
| G | | | | | | | | |
| T | | | | | | | | |
| T | | | | | | | | |
| A | | | | | | | | |



## Example run with AGACATTG and GAGTTA

| | | A | G | A | C | A | T | T | G |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| G | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| A | 2 | | | | | | | | |
| G | 3 | | | | | | | | |
| T | 4 | | | | | | | | |
| T | 5 | | | | | | | | |
| A | 6 | | | | | | | | |



## Example run with AGACATTG and GAGTTA

| | | A | G | A | C | A | T | T | G |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| G | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| A | 2 | 1 | 2 | 1 | | | | | |
| G | 3 | | | | | | | | |
| T | 4 | | | | | | | | |
| T | 5 | | | | | | | | |
| A | 6 | | | | | | | | |



## Example run with AGACATTG and GAGTTA

| | | A | G | A | C | A | T | T | G |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| G | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| A | 2 | 1 | 2 | 1 | 2 | 3 | 4 | 5 | 6 |
| G | 3 | 2 | 1 | 2 | 2 | 3 | 4 | 5 | 5 |
| T | 4 | | | | | | | | |
| T | 5 | | | | | | | | |
| A | 6 | | | | | | | | |



## Example run with AGACATTG and GAGTTA

| | | A | G | A | C | A | T | T | G |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| G | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| A | 2 | 1 | 2 | 1 | 2 | 3 | 4 | 5 | 6 |
| G | 3 | 2 | 1 | 2 | 2 | 3 | 4 | 5 | 5 |
| T | 4 | 3 | 2 | 2 | 3 | 3 | 3 | 4 | 5 |
| T | 5 | 4 | 3 | 3 | 3 | 4 | 3 | 3 | 4 |
| A | 6 | 5 | 4 | 3 | 4 | 3 | 4 | 4 | 4 |

## Example run with AGACATTG and GAGTTA

| | | A | G | A | C | A | T | T | G |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| G | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| A | 2 | 1 | 2 | 1 | 2 | 3 | 4 | 5 | 6 |
| G | 3 | 2 | 1 | 2 | 2 | 3 | 4 | 5 | 5 |
| T | 4 | 3 | 2 | 2 | 3 | 3 | 3 | 4 | 5 |
| T | 5 | 4 | 3 | 3 | 3 | 4 | 3 | 3 | 4 |
| A | 6 | 5 | 4 | 3 | 4 | 3 | 4 | 4 | 4 |



## Example run with AGACATTG and GAGTTA

| | | A | G | A | C | A | T | T | G |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| G | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| A | 2 | 1 | 2 | 1 | 2 | 3 | 4 | 5 | 6 |
| G | 3 | 2 | 1 | 2 | 2 | 3 | 4 | 5 | 5 |
| T | 4 | 3 | 2 | 2 | 3 | 3 | 3 | 4 | 5 |
| T | 5 | 4 | 3 | 3 | 3 | 4 | 3 | 3 | 4 |
| A | 6 | 5 | 4 | 3 | 4 | 3 | 4 | 4 | 4 |



## Reading off the operations

- Follow the sequence and use each color of arrow to tell you what operation was performed.



## Longest Increasing Subsequence

- Given a sequence of integers $s_1,\ldots,s_n$ find a subsequence $s_{i_1} < s_{i_2} < \ldots < s_{i_k}$ with $i_1<\ldots<i_k$ so that $k$ is as large as possible.
- e.g. Given 9,5,2,8,7,3,1,6,4 as input,
  - possible increasing subsequence is 5,7
  - better is 2,3,6 or 2,3,4 (either or which would be a correct output to our problem)



## Find recursive algorithm

- Solve sub-problem on $s_1,\ldots,s_{n-1}$ and then try to extend using $s_n$
- Two cases:
  - $s_n$ is not used
    - answer is the same answer as on $s_1,\ldots,s_{n-1}$
  - $s_n$ is used
    - answer is $s_n$ preceded by the longest increasing subsequence in $s_1,\ldots,s_{n-1}$ that ends in a number smaller than $s_n$



## Refined recursive idea (stronger notion of subproblem)

- Suppose that we knew for each $i<n$ the longest increasing subsequence in $s_1,\ldots,s_n$ that ends in $s_i$.
- Now to compute value for $i=n$ find
  - $s_n$ preceded by the maximum over all $i<n$ such that $s_i<s_n$ of the longest increasing subsequence ending in $s_i$
- First find the best **length** first rather than trying to actually compute the sequence itself.

## Recurrence

- Let L[j]=length of longest increasing subsequence in $s_1,\ldots,s_n$ that ends in $s_j$.
- L[j]=1+max{L[i] **:** i<j and $s_i<s_j$}
  (where max of an empty set is 0)
- Length of longest increasing subsequence:
  - max{L[i]: $1\le i \le n$}

19

## Computing the actual sequence

- For each j, we computed
  L[j]=1+max{L[i] **:** i<j and $s_i<s_j$}
  (where max of an empty set is 0)
- Also maintain P[j] the value of the i that achieved that max
  - this will be the index of the predecessor of $s_j$ in a longest increasing subsequence that ends in $s_j$
  - by following the P[j] values we can reconstruct the whole sequence in linear time.

20

## Longest Increasing Subsequence Algorithm

- **for** j=1 **to** n **do**
  L[j]←1
  P[j]←0
  **for** i=1 **to** j-1 **do**
  **if** ($s_i$<$s_j$ & L[i]+1>L[j]) **then**
  P[j] ←i
  L[j] ←L[i]+1
  **endfor**
  **endfor**
- Now find **j** such that **L[j]** is largest and walk backwards through **P[j]** pointers to find the sequence

21