Objectives

- Dynamic Programming
  - Sequence Alignment
  - Improving space requirements

Review: Dynamic Programming

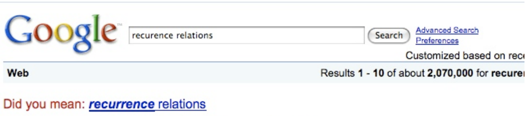
- Summarize each of the different templates we have used to do dynamic programming
  - Think about problems we have solved

Dynamic Programming Approaches

- Binary decision (weighted interval scheduling)
- Multiway decision (least segmented squares)
- Adding a parameter (knapsack)
- Intervals (RNA substructure)

SEQUENCE ALIGNMENT

Has This Ever Happened To You?



Customized based on rec

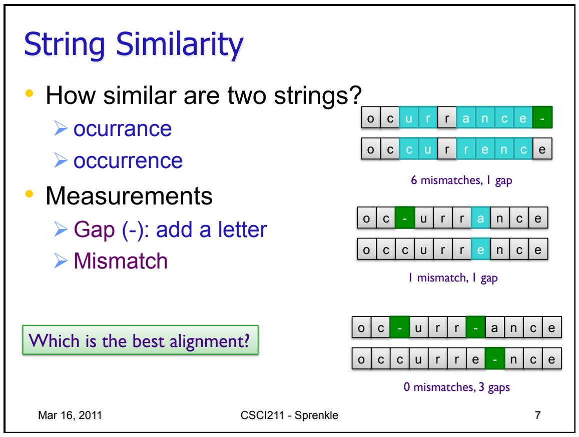
Results 1 - 10 of about 2,070,000 for recure

Did you mean: *recurrence* relations

How does Google know what I really meant?

String Similarity

- How similar are two strings?
  - ocurrance
  - occurrence
- We intuitively can tell that these two are similar
  - Systematic measurement?



String Similarity

- How similar are two strings?
  - ocurrance
  - occurrence
- Measurements
  - Gap (-): add a letter
  - Mismatch

o	c	u	r	r	a	n	c	e	-
o	c	c	u	r	r	e	n	c	e

6 mismatches, 1 gap

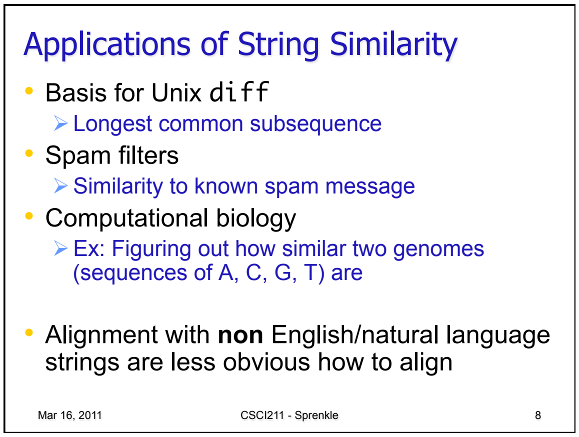
o	c	-	u	r	r	a	n	c	e
o	c	c	u	r	r	e	n	c	e

1 mismatch, 1 gap

o	c	-	u	r	r	-	a	n	c	e
o	c	c	u	r	r	e	-	n	c	e

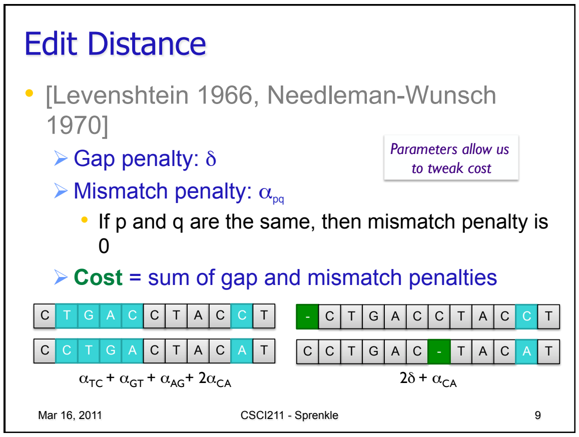
0 mismatches, 3 gaps

Which is the best alignment?



Applications of String Similarity

- Basis for Unix `diff`
  - Longest common subsequence
- Spam filters
  - Similarity to known spam message
- Computational biology
  - Ex: Figuring out how similar two genomes (sequences of A, C, G, T) are
- Alignment with **non** English/natural language strings are less obvious how to align



Edit Distance

- [Levenshtein 1966, Needleman-Wunsch 1970]
  - Gap penalty: δ
  - Mismatch penalty: α_{pq}
    - If p and q are the same, then mismatch penalty is 0
  - **Cost** = sum of gap and mismatch penalties

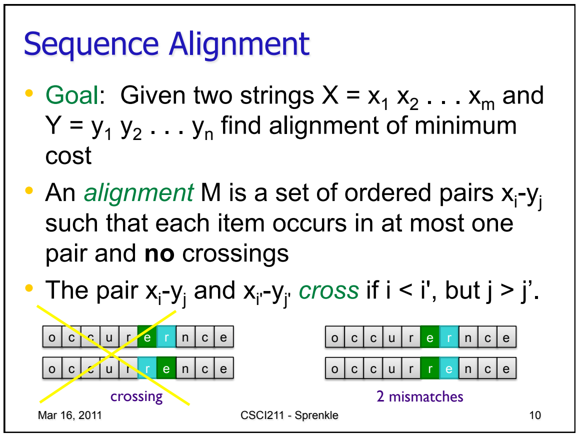
Parameters allow us to tweak cost

C	T	G	A	C	C	T	A	C	C	T
C	C	T	G	A	C	T	A	C	A	T

$\alpha_{TC} + \alpha_{GT} + \alpha_{AG} + 2\alpha_{CA}$

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

$2\delta + \alpha_{CA}$



Sequence Alignment

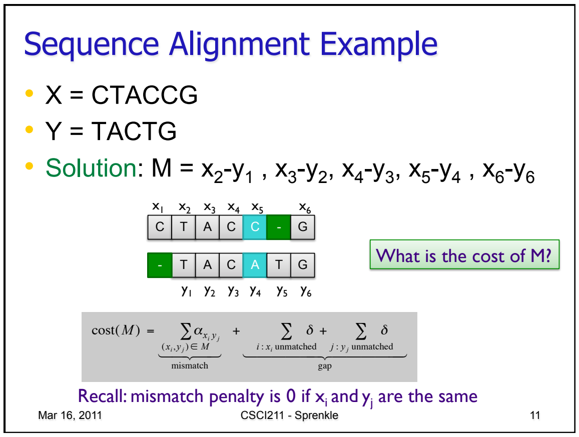
- Goal: Given two strings $X = x_1\ x_2 \ldots x_m$ and $Y = y_1\ y_2 \ldots y_n$ find alignment of minimum cost
- An *alignment* M is a set of ordered pairs x_i-y_j such that each item occurs in at most one pair and **no** crossings
- The pair x_i-y_j and $x_{i'}$-$y_{j'}$ *cross* if i < i', but j > j'.

o	c	c	u	r	e	r	n	c	e
o	c	c	u	r	r	e	n	c	e

crossing

o	c	c	u	r	e	r	n	c	e
o	c	c	u	r	r	e	n	c	e

2 mismatches



Sequence Alignment Example

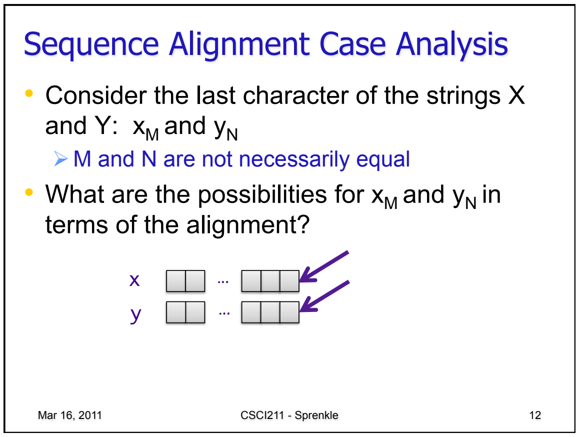
- X = CTACCG
- Y = TACTG
- Solution: M = x_2-y_1 , x_3-y_2, x_4-y_3, x_5-y_4 , x_6-y_6

x_1	x_2	x_3	x_4	x_5		x_6
C	T	A	C	C	-	G
-	T	A	C	A	T	G
	y_1	y_2	y_3	y_4	y_5	y_6

What is the cost of M?

$$\text{cost}(M) = \underbrace{\sum_{(x_i, y_j) \in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i : x_i \text{ unmatched}} \delta + \sum_{j : y_j \text{ unmatched}} \delta}_{\text{gap}}$$

Recall: mismatch penalty is 0 if x_i and y_j are the same

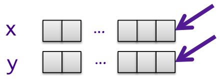


Sequence Alignment Case Analysis

- Consider the last character of the strings X and Y: x_M and y_N
  - M and N are not necessarily equal
- What are the possibilities for x_M and y_N in terms of the alignment?

Sequence Alignment Case Analysis

- Consider last character of strings X and Y: x_M and y_N
  - Case 1: x_M and y_N are aligned
  - Case 2: x_M is not matched
  - Case 3: y_N is not matched



Formulate the optimal solution's value

Sequence Alignment Case Analysis

- Consider last character of strings X and Y: x_M and y_N
  - Case 1: x_M and y_N are aligned
  - Case 2: x_M is not matched
  - Case 3: y_N is not matched

What are the costs for these cases?

x y

- OPT(i, j) = min cost of aligning strings $x_1\, x_2 \ldots x_i$ and $y_1\, y_2 \ldots y_j$

Sequence Alignment Cost Analysis

- Consider last character of strings X and Y: x_M and y_N
  - Case 1: x_M and y_N are aligned
    - Pay mismatch for x_M-y_N + min cost of aligning rest of strings
    - OPT(M, N) = α_{XmYn} + OPT(M-1, N-1)
  - Case 2: x_M is not matched
    - Pay gap for x_M + min cost of aligning rest of strings
    - OPT(M, N) = δ + OPT(M-1, N)
  - Case 3: y_N is not matched
    - Pay gap for y_N + min cost of aligning rest of strings
    - OPT(M, N) = δ + OPT(M, N-1)

Sequence Alignment Cost Analysis

- Base costs? → i or j is 0
  - What happens when we run out of letters in one string before the other?

```
X = CTACCG
Y = TACTG
```

Sequence Alignment: Problem Structure

Gaps for remainder of Y

$$OPT(i,j) = \begin{cases} j\delta & \text{if } i=0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{otherwise} \\ i\delta & \text{if } j=0 \end{cases}$$

Ran out of 1st string

Ran out of 2nd string

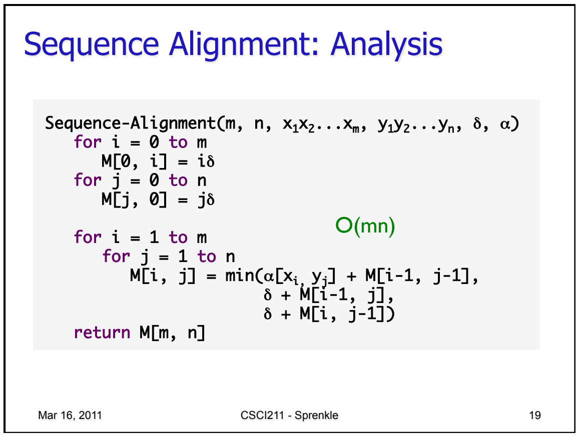
Gaps for remainder of X

Sequence Alignment: Algorithm

Cost parameters

```
Sequence-Alignment(m, n, $x_1x_2...x_m$, $y_1y_2...y_n$, δ, α)
   for i = 0 to m
      M[0, i] = iδ
   for j = 0 to n
      M[j, 0] = jδ

   for i = 1 to m
      for j = 1 to n
         M[i, j] = min(α[$x_i$, $y_j$] + M[i-1, j-1],
                       δ + M[i-1, j],
                       δ + M[i, j-1])
   return M[m, n]
```

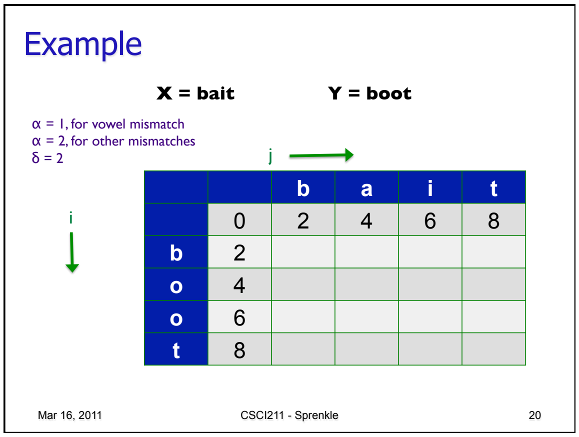
Costs?



Sequence Alignment: Analysis

```
Sequence-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α)
   for i = 0 to m
      M[0, i] = iδ
   for j = 0 to n
      M[j, 0] = jδ

   for i = 1 to m
      for j = 1 to n
         M[i, j] = min(α[xi, yj] + M[i-1, j-1],
                       δ + M[i-1, j],
                       δ + M[i, j-1])
   return M[m, n]
```

O(mn)



Example

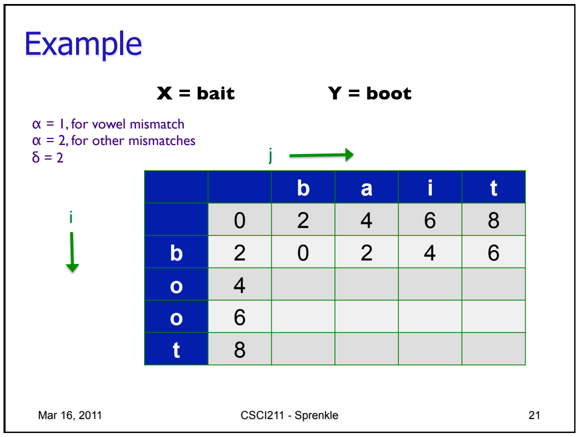
X = bait **Y = boot**

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

j

i

		b	a	i	t
	0	2	4	6	8
b	2				
o	4				
o	6				
t	8				



Example

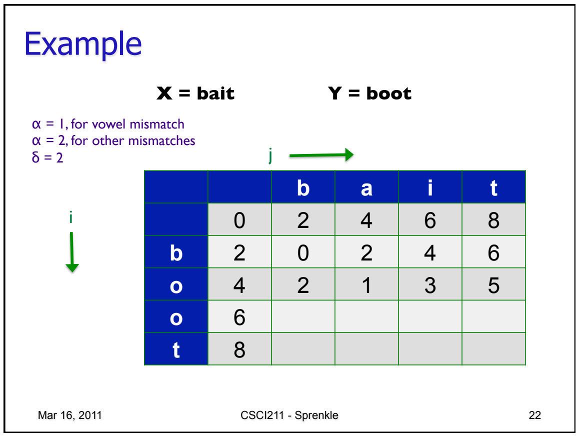
X = bait **Y = boot**

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

j

i

		b	a	i	t
	0	2	4	6	8
b	2	0	2	4	6
o	4				
o	6				
t	8				



Example

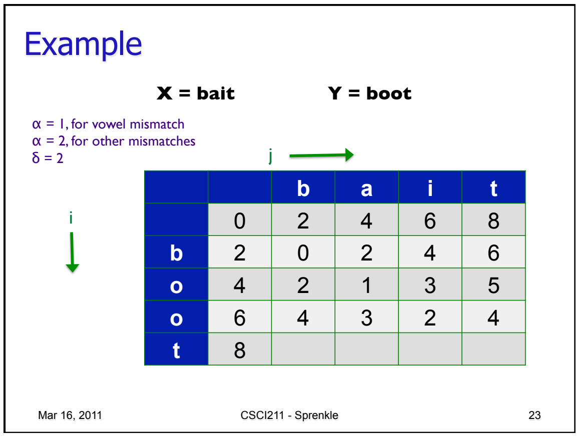
X = bait **Y = boot**

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

j

i

		b	a	i	t
	0	2	4	6	8
b	2	0	2	4	6
o	4	2	1	3	5
o	6				
t	8				



Example

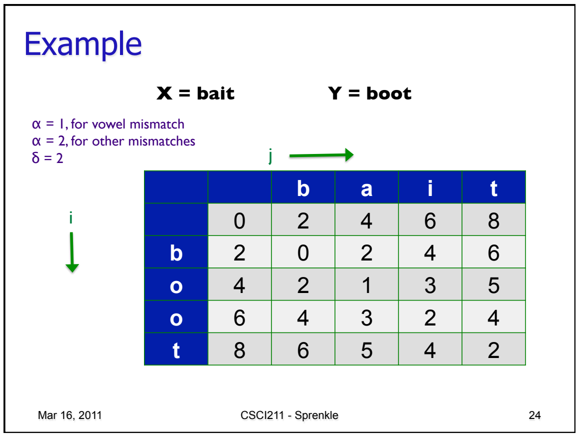
X = bait **Y = boot**

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

j

i

		b	a	i	t
	0	2	4	6	8
b	2	0	2	4	6
o	4	2	1	3	5
o	6	4	3	2	4
t	8				



Example

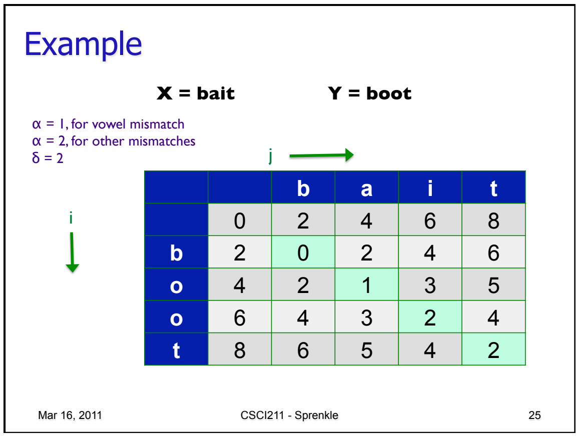
X = bait **Y = boot**

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

j

i

		b	a	i	t
	0	2	4	6	8
b	2	0	2	4	6
o	4	2	1	3	5
o	6	4	3	2	4
t	8	6	5	4	2



Example

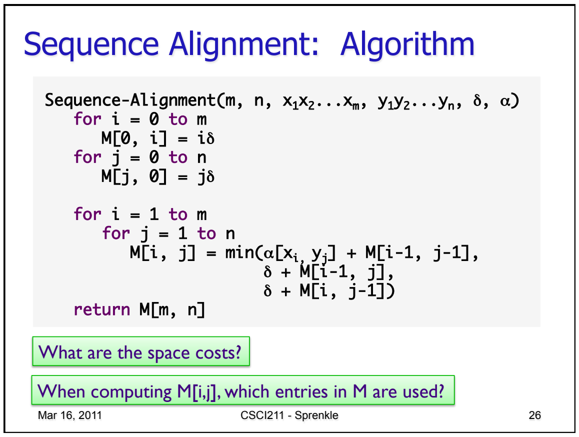
X = bait **Y = boot**

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

j →

i ↓

		b	a	i	t
	0	2	4	6	8
b	2	0	2	4	6
o	4	2	1	3	5
o	6	4	3	2	4
t	8	6	5	4	2

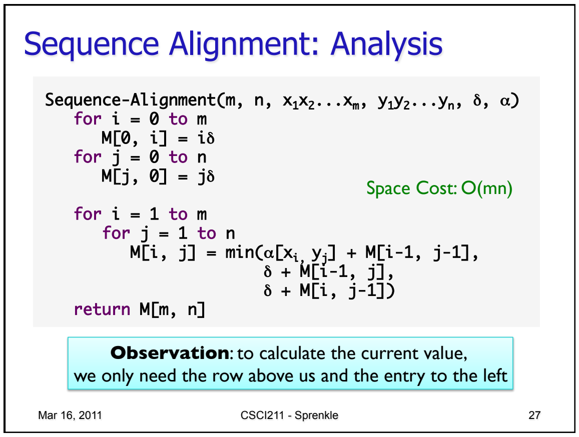


Sequence Alignment: Algorithm

```
Sequence-Alignment(m, n, $x_1x_2...x_m$, $y_1y_2...y_n$, δ, α)
   for i = 0 to m
      M[0, i] = iδ
   for j = 0 to n
      M[j, 0] = jδ

   for i = 1 to m
      for j = 1 to n
         M[i, j] = min(α[$x_i$, $y_j$] + M[i-1, j-1],
                       δ + M[i-1, j],
                       δ + M[i, j-1])
   return M[m, n]
```

What are the space costs?

When computing M[i,j], which entries in M are used?

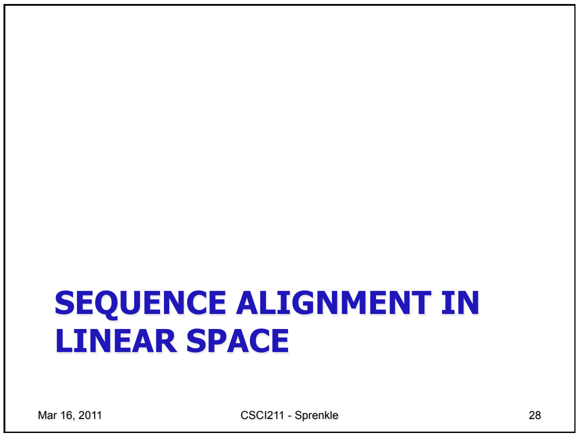


Sequence Alignment: Analysis

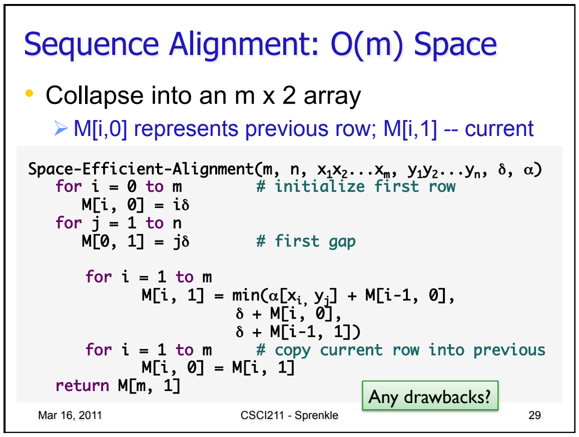
```
Sequence-Alignment(m, n, $x_1x_2...x_m$, $y_1y_2...y_n$, δ, α)
   for i = 0 to m
      M[0, i] = iδ
   for j = 0 to n
      M[j, 0] = jδ

   for i = 1 to m
      for j = 1 to n
         M[i, j] = min(α[$x_i$, $y_j$] + M[i-1, j-1],
                       δ + M[i-1, j],
                       δ + M[i, j-1])
   return M[m, n]
```

Space Cost: O(mn)

Observation: to calculate the current value, we only need the row above us and the entry to the left



SEQUENCE ALIGNMENT IN LINEAR SPACE

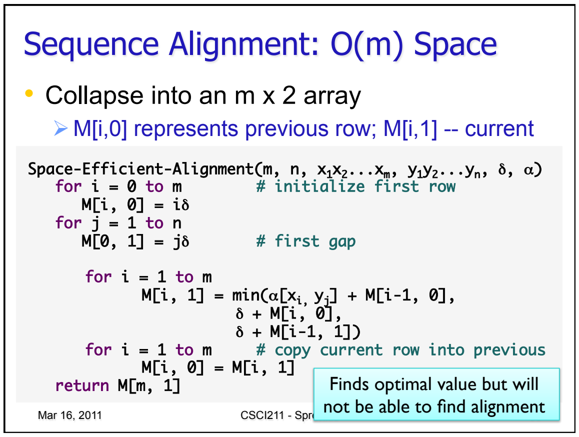


Sequence Alignment: O(m) Space

- Collapse into an m x 2 array
  - M[i,0] represents previous row; M[i,1] -- current

```
Space-Efficient-Alignment(m, n, $x_1x_2...x_m$, $y_1y_2...y_n$, δ, α)
   for i = 0 to m          # initialize first row
      M[i, 0] = iδ
   for j = 1 to n
      M[0, 1] = jδ         # first gap

      for i = 1 to m
            M[i, 1] = min(α[$x_i$, $y_j$] + M[i-1, 0],
                      δ + M[i, 0],
                      δ + M[i-1, 1])
      for i = 1 to m       # copy current row into previous
            M[i, 0] = M[i, 1]
   return M[m, 1]
```

Any drawbacks?



Sequence Alignment: O(m) Space

- Collapse into an m x 2 array
  - M[i,0] represents previous row; M[i,1] -- current

```
Space-Efficient-Alignment(m, n, $x_1x_2...x_m$, $y_1y_2...y_n$, δ, α)
   for i = 0 to m          # initialize first row
      M[i, 0] = iδ
   for j = 1 to n
      M[0, 1] = jδ         # first gap

      for i = 1 to m
            M[i, 1] = min(α[$x_i$, $y_j$] + M[i-1, 0],
                      δ + M[i, 0],
                      δ + M[i-1, 1])
      for i = 1 to m       # copy current row into previous
            M[i, 0] = M[i, 1]
   return M[m, 1]
```

Finds optimal value but will not be able to find alignment

Why Do We Care About Space?

- For English words or sentences, probably doesn't matter
- Matters for Biological sequence alignment
  - Consider: 2 strings with 100,000 symbols each
    - Processor can do 10 billion primitive operations
    - BUT dealing with a 10 GB array

Sequence Alignment: Linear Space

- Can we avoid using quadratic space?
  - Optimal value in O(m) space and O(mn) time.
    - Compute OPT(i, •) from OPT(i-1, •)
    - BUT, no simple way to recover alignment itself
- Theorem. [Hirschberg 1975] Optimal alignment in O(m + n) space and O(mn) time.
  - Clever combination of *divide-and-conquer* and *dynamic programming*

Recall Our Example

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

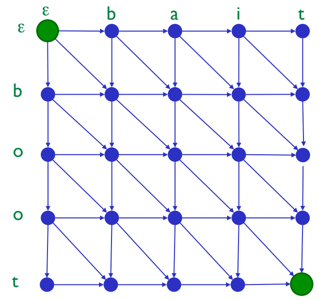
X = bait **Y = boot**

j →

i ↓

		b	a	i	t
	0	2	4	6	8
b	2	0	2	4	6
o	4	2	1	3	5
o	6	4	3	2	4
t	8	6	5	4	2

Mapping to a Graph Problem

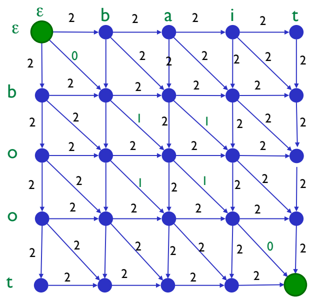


α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

- Horizontal and vertical edges cost δ
- Diagonal edges cost α

Goal: Find shortest path from top-left to bottom-right

Mapping to a Graph Problem



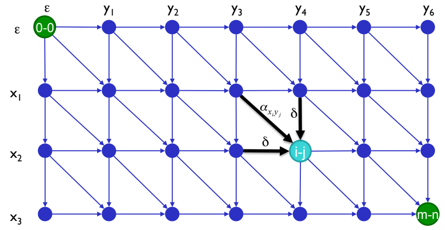
α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

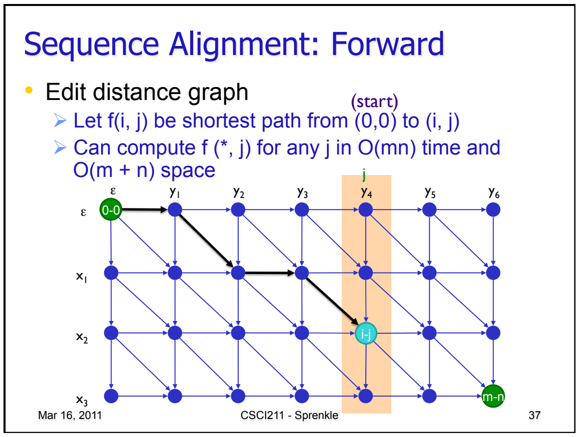
- Horizontal and vertical edges cost δ
- Diagonal edges cost α

Goal: Find shortest path from top-left to bottom-right

Sequence Alignment: Forward

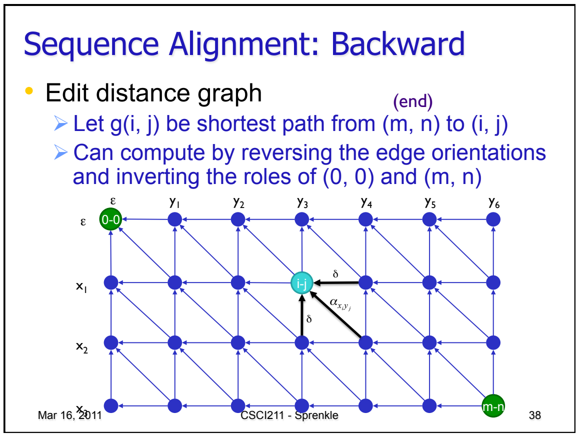
- Edit distance graph
  - Let f(i, j) be shortest path from (0,0) to (i, j) (start)
  - Observation: f(i, j) = OPT(i, j)





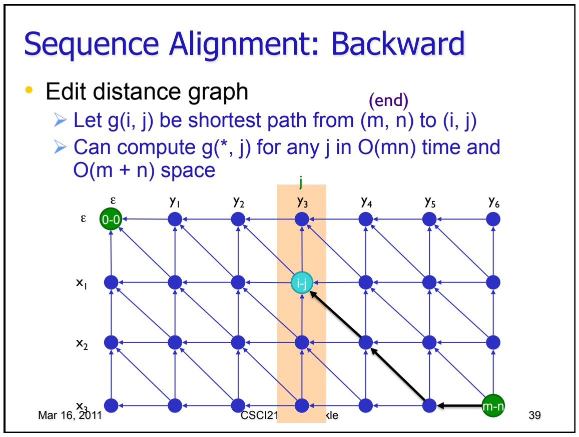
Sequence Alignment: Forward

- Edit distance graph
  - Let f(i, j) be shortest path from (0,0) to (i, j)
  - Can compute f (*, j) for any j in O(mn) time and O(m + n) space



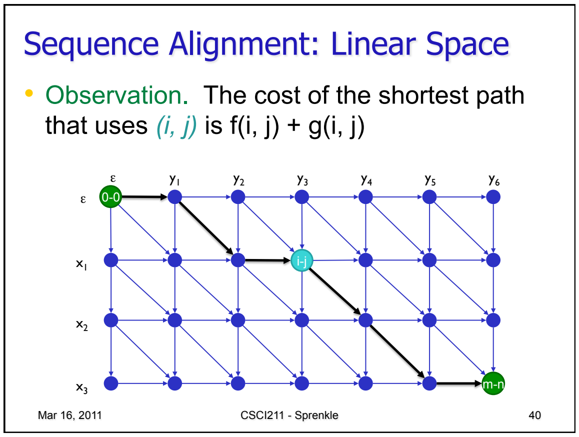
Sequence Alignment: Backward

- Edit distance graph
  - Let g(i, j) be shortest path from (m, n) to (i, j)
  - Can compute by reversing the edge orientations and inverting the roles of (0, 0) and (m, n)



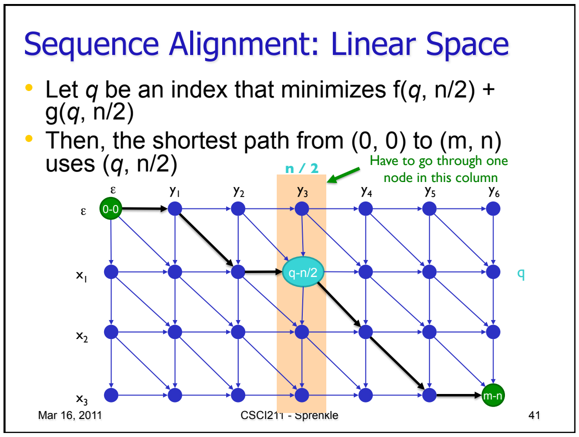
Sequence Alignment: Backward

- Edit distance graph
  - Let g(i, j) be shortest path from (m, n) to (i, j)
  - Can compute g(*, j) for any j in O(mn) time and O(m + n) space



Sequence Alignment: Linear Space

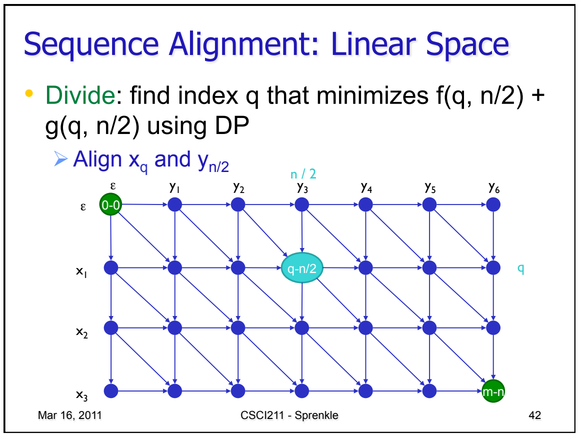
- Observation. The cost of the shortest path that uses *(i, j)* is f(i, j) + g(i, j)



Sequence Alignment: Linear Space

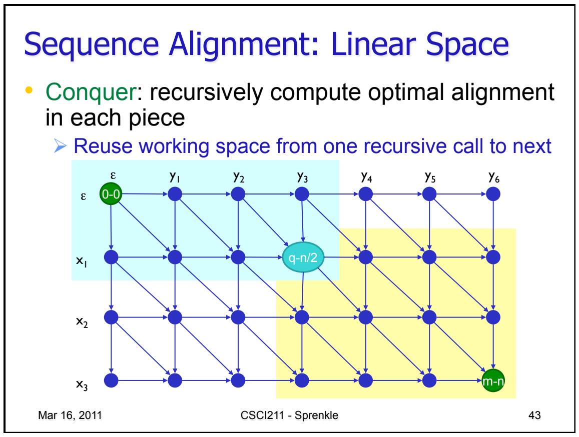
- Let *q* be an index that minimizes f(*q*, n/2) + g(*q*, n/2)
- Then, the shortest path from (0, 0) to (m, n) uses (*q*, n/2)

Have to go through one node in this column



Sequence Alignment: Linear Space

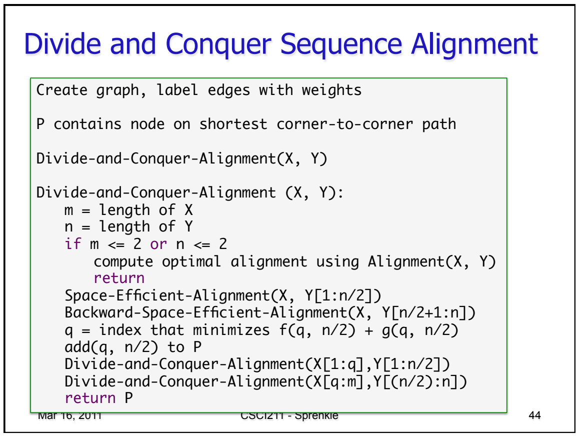
- Divide: find index q that minimizes f(q, n/2) + g(q, n/2) using DP
  - Align x_q and $y_{n/2}$



Sequence Alignment: Linear Space

- Conquer: recursively compute optimal alignment in each piece
  - Reuse working space from one recursive call to next

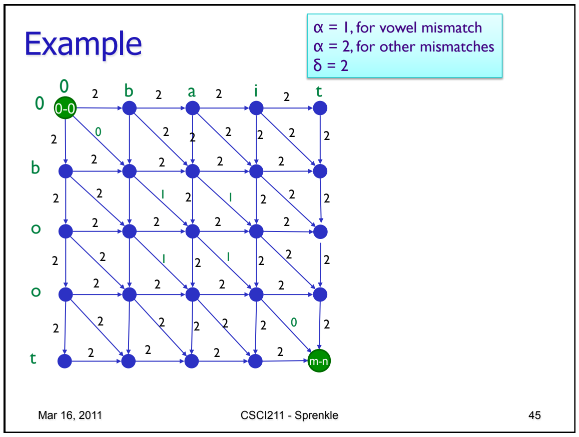
Mar 16, 2011 CSCI211 - Sprenkle 43



Divide and Conquer Sequence Alignment

```
Create graph, label edges with weights

P contains node on shortest corner-to-corner path

Divide-and-Conquer-Alignment(X, Y)

Divide-and-Conquer-Alignment (X, Y):
    m = length of X
    n = length of Y
    if m <= 2 or n <= 2
        compute optimal alignment using Alignment(X, Y)
        return
    Space-Efficient-Alignment(X, Y[1:n/2])
    Backward-Space-Efficient-Alignment(X, Y[n/2+1:n])
    q = index that minimizes f(q, n/2) + g(q, n/2)
    add(q, n/2) to P
    Divide-and-Conquer-Alignment(X[1:q],Y[1:n/2])
    Divide-and-Conquer-Alignment(X[q:m],Y[(n/2):n])
    return P
```

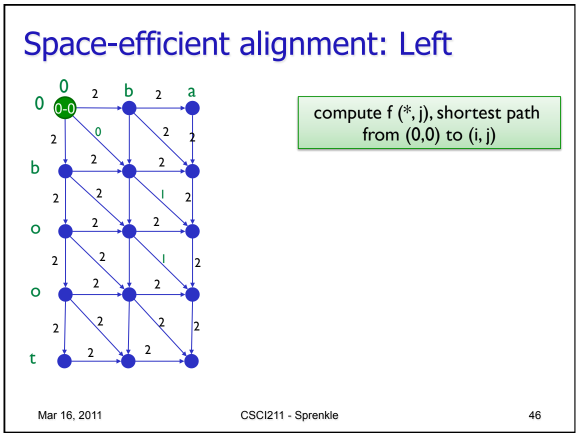
Mar 16, 2011 CSCI211 - Sprenkle 44



Example

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

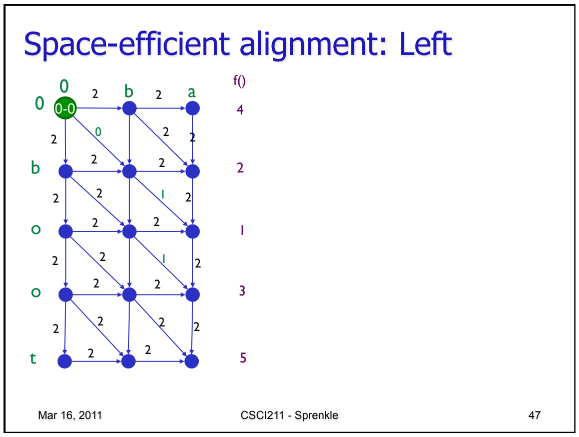
Mar 16, 2011 CSCI211 - Sprenkle 45



Space-efficient alignment: Left

compute f (*, j), shortest path from (0,0) to (i, j)

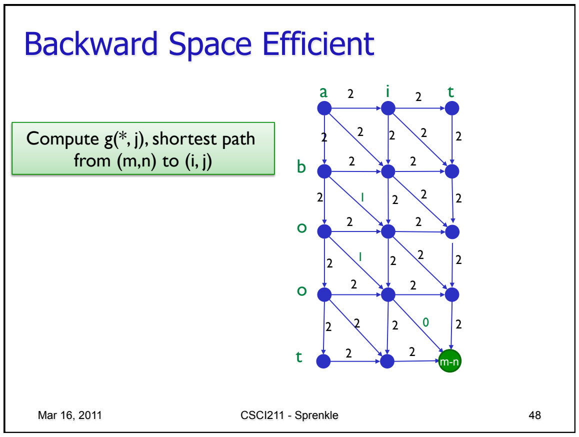
Mar 16, 2011 CSCI211 - Sprenkle 46



Space-efficient alignment: Left

f()
4
2
1
3
5

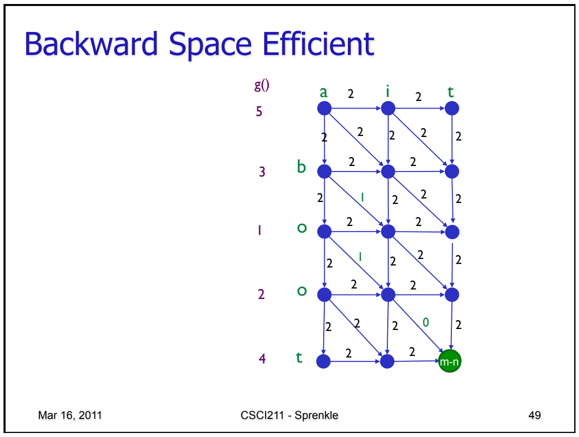
Mar 16, 2011 CSCI211 - Sprenkle 47



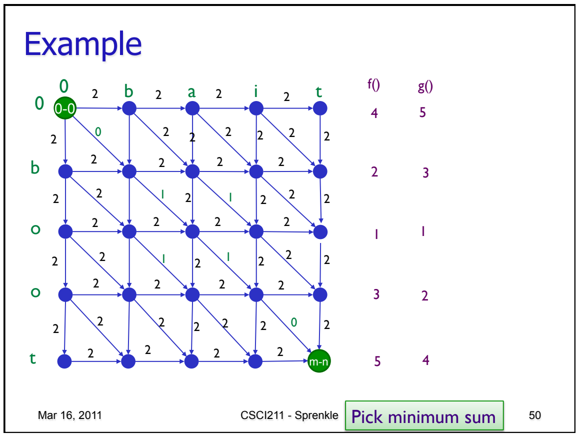
Backward Space Efficient

Compute g(*, j), shortest path from (m,n) to (i, j)

Mar 16, 2011 CSCI211 - Sprenkle 48

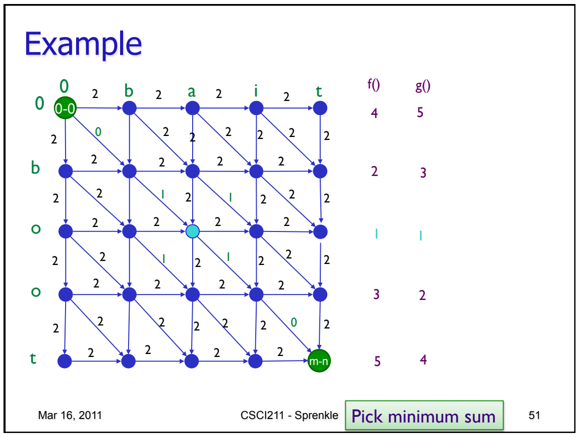


Backward Space Efficient



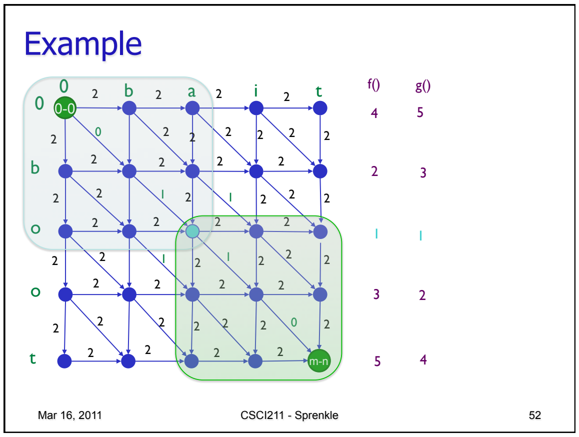
Example

Pick minimum sum

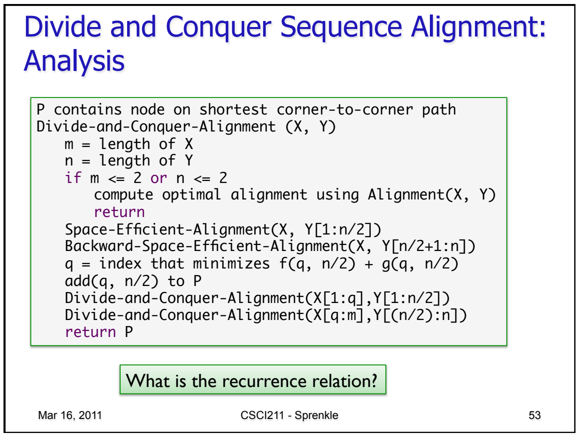


Example

Pick minimum sum



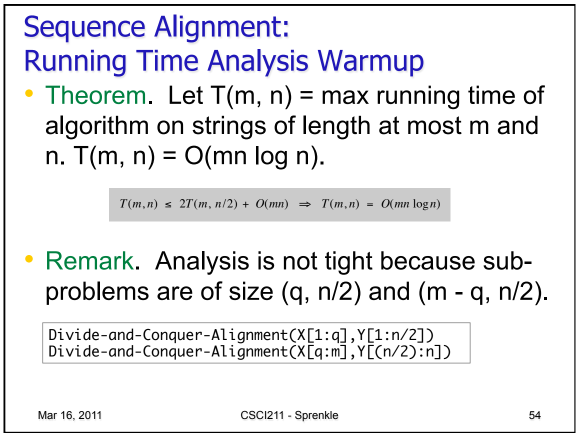
Example



Divide and Conquer Sequence Alignment: Analysis

```
P contains node on shortest corner-to-corner path
Divide-and-Conquer-Alignment (X, Y)
    m = length of X
    n = length of Y
    if m <= 2 or n <= 2
        compute optimal alignment using Alignment(X, Y)
        return
    Space-Efficient-Alignment(X, Y[1:n/2])
    Backward-Space-Efficient-Alignment(X, Y[n/2+1:n])
    q = index that minimizes f(q, n/2) + g(q, n/2)
    add(q, n/2) to P
    Divide-and-Conquer-Alignment(X[1:q],Y[1:n/2])
    Divide-and-Conquer-Alignment(X[q:m],Y[(n/2):n])
    return P
```

What is the recurrence relation?



Sequence Alignment: Running Time Analysis Warmup

- **Theorem**. Let T(m, n) = max running time of algorithm on strings of length at most m and n. T(m, n) = O(mn log n).

$$T(m,n) \le 2T(m,n/2) + O(mn) \Rightarrow T(m,n) = O(mn\log n)$$

- **Remark**. Analysis is not tight because sub-problems are of size (q, n/2) and (m - q, n/2).

```
Divide-and-Conquer-Alignment(X[1:q],Y[1:n/2])
Divide-and-Conquer-Alignment(X[q:m],Y[(n/2):n])
```

Sequence Alignment: Running Time Analysis

- Theorem. Let T(m, n) = max running time of algorithm on strings of length m and n. T(m, n) = O(mn)
- Recurrence Relation:

$$\begin{aligned} T(m,\ 2) &\le cm \\ T(2,\ n) &\le cn \\ T(m,\ n) &\le cmn + T(q,\ n/2) + T(m-q,\ n/2) \end{aligned}$$

- Solve using substitution:

$$\begin{aligned} T(m,n) &\le T(q,n/2) + T(m-q,n/2) + cmn \\ &\le 2cqn/2 + 2c(m-q)n/2 + cmn \\ &= cqn + cmn - cqn + cmn \\ &= 2cmn \end{aligned}$$

This Week

- Problem Set 7 due Friday
  - Looks short but lots of parts
- Jan Cuny’s visit
  - 3 p.m. – reception to meet Jan
  - 4 p.m. – Broadening Participation in Computing
  - Recorded using Tegrity, should be able to watch later
- Keep reading Chapter 6