CS 473: Algorithms

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

University of Illinois, Urbana-Champaign

Fall 2009

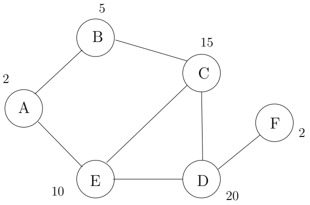
Part I

Maximum Weighted Independent Set in Trees

Maximum Weight Independent Set Problem

Input Graph $G = (V, E)$ and weights $w(v) \geq 0$ for each $v \in V$

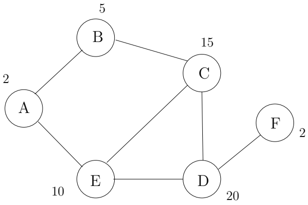
Goal Find maximum weight independent set in G



Maximum Weight Independent Set Problem

Input Graph $G = (V, E)$ and weights $w(v) \geq 0$ for each $v \in V$

Goal Find maximum weight independent set in G

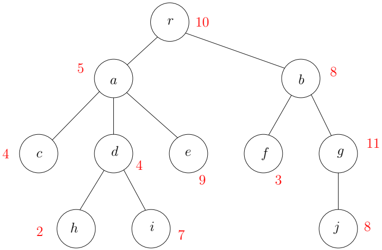


Maximum weight independent set in above graph: $\{B, D\}$

Maximum Weight Independent Set in a Tree

Input Tree $T = (V, E)$ and weights $w(v) \geq 0$ for each $v \in V$

Goal Find maximum weight independent set in T



Maximum weight independent set in above tree: ??

Towards a Recursive Solution

For an arbitrary graph G:

- Number vertices as $v_1, v_2, \ldots, v_n$
- Find recursively optimum solutions without v_n (recurse on $G - v_n$) and with v_n (recurse on $G - v_n - N(v_n)$ & include v_n).
- Saw that if graph G is arbitrary there was no good ordering that resulted in a small number of subproblems.

Towards a Recursive Solution

For an arbitrary graph G:

- Number vertices as $v_1, v_2, \ldots, v_n$
- Find recursively optimum solutions without v_n (recurse on $G - v_n$) and with v_n (recurse on $G - v_n - N(v_n)$ & include v_n).
- Saw that if graph G is arbitrary there was no good ordering that resulted in a small number of subproblems.

What about a tree?

Towards a Recursive Solution

For an arbitrary graph G:

- Number vertices as $v_1, v_2, \ldots, v_n$
- Find recursively optimum solutions without v_n (recurse on $G - v_n$) and with v_n (recurse on $G - v_n - N(v_n)$ & include v_n).
- Saw that if graph G is arbitrary there was no good ordering that resulted in a small number of subproblems.

What about a tree? Natural candidate for v_n is root r of T?

Towards a Recursive Solution

Natural candidate for v_n is root r of T? Let $\mathcal{O}$ be an optimum solution to the whole problem.

Case $r \notin \mathcal{O}$ Then $\mathcal{O}$ contains an optimum solution for each subtree of T hanging at a child of r.

Towards a Recursive Solution

Natural candidate for v_n is root r of T? Let $\mathcal{O}$ be an optimum solution to the whole problem.

Case $r \notin \mathcal{O}$ Then $\mathcal{O}$ contains an optimum solution for each subtree of T hanging at a child of r.

Case $r \in \mathcal{O}$ None of the children of r can be in $\mathcal{O}$. $\mathcal{O} - \{r\}$ contains an optimum solution for each subtree of T hanging at a grandchild of r.

Towards a Recursive Solution

Natural candidate for v_n is root r of T? Let $\mathcal{O}$ be an optimum solution to the whole problem.

Case $r \notin \mathcal{O}$ Then $\mathcal{O}$ contains an optimum solution for each subtree of T hanging at a child of r.

Case $r \in \mathcal{O}$ None of the children of r can be in $\mathcal{O}$. $\mathcal{O} - \{r\}$ contains an optimum solution for each subtree of T hanging at a grandchild of r.

Subproblems? Subtrees of T hanging at nodes in T.

A Recursive Solution

$T(u)$: subtree of T hanging at node u
$OPT(u)$: max weighted independent set value in $T(u)$

$$OPT(u) =$$

A Recursive Solution

$T(u)$: subtree of T hanging at node u
$OPT(u)$: max weighted independent set value in $T(u)$

$$OPT(u) = \max\{\sum_{v \text{ child of } u} OPT(v), w(u) + \sum_{v \text{ grandchild of } u} OPT(v)\}$$

Iterative Algorithm

- Compute $OPT(u)$ bottom up. To evaluate $OPT(u)$ need to have computed values of all children and grandchildren of u
- What is an ordering of nodes of a tree T to achieve above?

Iterative Algorithm

- Compute $OPT(u)$ bottom up. To evaluate $OPT(u)$ need to have computed values of all children and grandchildren of u
- What is an ordering of nodes of a tree T to achieve above? Post-order traversal of a tree.

Iterative Algorithm

- Compute $OPT(u)$ bottom up. To evaluate $OPT(u)$ need to have computed values of all children and grandchildren of u
- What is an ordering of nodes of a tree T to achieve above? Post-order traversal of a tree.

```
Let v_1, v_2, ..., v_n be a post-order traversal of T
for i = 1 to n do
    M[v_i] = max(Σ_{v_j child of v_i} M[v_j], w(v_i) + Σ_{v_j grandchild of v_i} M[v_j])
return M[v_n] (* Note:  v_n is the root of T *)
```

Iterative Algorithm

- Compute $OPT(u)$ bottom up. To evaluate $OPT(u)$ need to have computed values of all children and grandchildren of u
- What is an ordering of nodes of a tree T to achieve above? Post-order traversal of a tree.

```
Let v_1, v_2, ..., v_n be a post-order traversal of T
for i = 1 to n do
    M[v_i] = max(∑_{v_j child of v_i} M[v_j], w(v_i) + ∑_{v_j grandchild of v_i} M[v_j])
return M[v_n] (* Note:  v_n is the root of T *)
```

Running time:

Iterative Algorithm

- Compute $OPT(u)$ bottom up. To evaluate $OPT(u)$ need to have computed values of all children and grandchildren of u
- What is an ordering of nodes of a tree T to achieve above? Post-order traversal of a tree.

```
Let v_1, v_2, ..., v_n be a post-order traversal of T
for i = 1 to n do
    M[v_i] = max(Σ_{v_j child of v_i} M[v_j], w(v_i) + Σ_{v_j grandchild of v_i} M[v_j])
return M[v_n] (* Note: v_n is the root of T *)
```

Running time:

- Naive bound: $O(n^2)$ since each $M[v_i]$ evaluation may take $O(n)$ time and there are n evaluations.

Iterative Algorithm

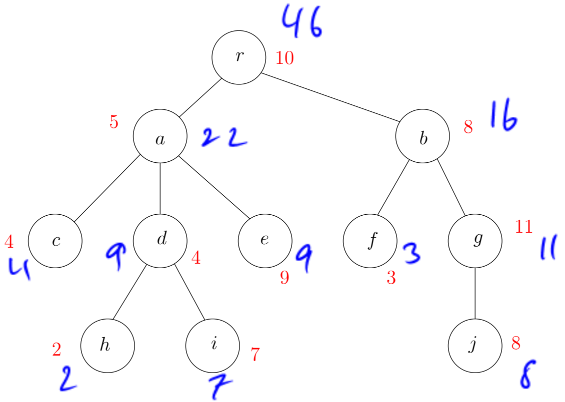
- Compute $OPT(u)$ bottom up. To evaluate $OPT(u)$ need to have computed values of all children and grandchildren of u
- What is an ordering of nodes of a tree T to achieve above? Post-order traversal of a tree.

```
Let v_1, v_2, ..., v_n be a post-order traversal of T
for i = 1 to n do
    M[v_i] = max(Σ_{v_j child of v_i} M[v_j], w(v_i) + Σ_{v_j grandchild of v_i} M[v_j])
return M[v_n] (* Note: v_n is the root of T *)
```

Running time:

- Naive bound: $O(n^2)$ since each $M[v_i]$ evaluation may take $O(n)$ time and there are n evaluations.
- Better bound: $O(n)$. A value $M[v_j]$ is accessed only by its parent and grand parent.

Example



Part II

DAGs and Dynamic Programming

Recursion and DAGs

Observation

Let A be a recursive algorithm for problem Π. For each instance I of Π there is an associated DAG $G(I)$.

- Create directed graph $G(I)$ as follows
- For each sub-problem in the execution of A on I create a node
- If sub-problem v *depends* on or *recursively calls* sub-problem u add *directed* edge (u, v) to graph
- $G(I)$ is a DAG. Why?

Recursion and DAGs

Observation

Let A be a recursive algorithm for problem Π. For each instance I of Π there is an associated DAG $G(I)$.

- Create directed graph $G(I)$ as follows
- For each sub-problem in the execution of A on I create a node
- If sub-problem v *depends* on or *recursively calls* sub-problem u add *directed* edge (u, v) to graph
- $G(I)$ is a DAG. Why? If $G(I)$ has a cycle then A will not terminate on I

Iterative Algorithm in Dynamic Programming and DAGs

Observation

An iterative algorithm B obtained from a recursive algorithm A for a problem Π does the following: for each instance I of Π, it computes a topological sort of $G(I)$ and evaluates sub-problems according to the topological ordering.

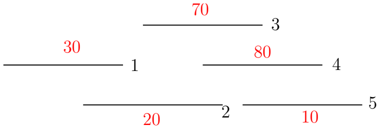
- Sometimes the DAG $G(I)$ can be obtained directly without thinking about the recursive algorithm A
- In some cases (not all) the computation of an optimal solution reduces to a shortest/longest path in DAG $G(I)$
- Topological sort based shortest/longest path computation is dynamic programming!

Weighted Interval Scheduling via Longest Path in a DAG

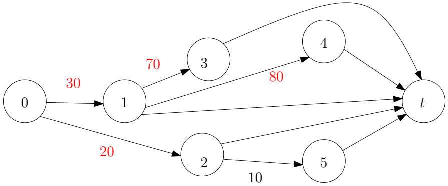
Given intervals, create a DAG as follows

- one node for each interval plus a dummy source node for interval 0 plus a dummy sink node t.
- for each interval i add edge $(p(i), i)$ of length/weight v_i.
- for each interval i add edge (i, t) of length 0

Example



$p(5) = 2,\ p(4) = 1,\ p(3) = 1,\ p(2) = 0,\ p(1) = 0$



Relating Optimum Solution

Given interval problem instance I let $G(I)$ denote the DAG constructed as described.

Claim: Optimum solution to weighted interval scheduling instance I is given by *longest* path from s to t in $G(I)$.

Relating Optimum Solution

Given interval problem instance I let $G(I)$ denote the DAG constructed as described.

Claim: Optimum solution to weighted interval scheduling instance I is given by *longest* path from s to t in $G(I)$.

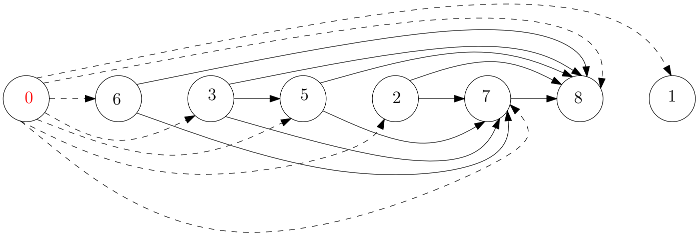
Assuming claim is true,

- If I has n intervals, DAG $G(I)$ has $n+2$ nodes and $O(n)$ edges. Creating $G(I)$ takes $O(n \log n)$ time: to find $p(i)$ for each i. How?
- Longest path can be computed in $O(n)$ time — recall $O(m+n)$ algorithm for shortest/longest paths in DAGs.

DAG for Longest Increasing Sequence

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

- add sentinel a_0 to sequence where a_0 is less than smallest element in sequence
- 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 from v_0



Part III

Edit Distance and Sequence Alignment

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

Definition

Edit distance between two words X and Y is the number of letter insertions, letter deletions and letter substitutions required to obtaine Y from X.

Example

The edit distance between FOOD and MONEY is at most 4

$$\underline{\mathrm{F}}\mathrm{OOD} \to \mathrm{MO}\underline{\mathrm{O}}\mathrm{D} \to \mathrm{MON} \sqcup \mathrm{D} \to \mathrm{MONE}\underline{\mathrm{D}} \to \mathrm{MONEY}$$

Edit Distance: Alternate View

Alignment

Place words one on top of the other, with gaps in the first word indicating insertions, and gaps in the second word indicating deletions.

F	O	O		D
M	O	N	E	Y

Edit Distance: Alternate View

Alignment

Place words one on top of the other, with gaps in the first word indicating insertions, and gaps in the second word indicating deletions.

$$\begin{array}{ccccc} \mathrm{F} & \mathrm{O} & \mathrm{O} & & \mathrm{D} \\ \mathrm{M} & \mathrm{O} & \mathrm{N} & \mathrm{E} & \mathrm{Y} \end{array}$$

Formally, an alignment is a set M of pairs (i, j) such that each index appears at most once, and there is no "crossing": $i < i'$ and i is matched to j implies i' is matched to $j' > j$. In the above example, this is $M = \{(1,1), (2,2), (3,3), (4,5)\}$.

Edit Distance: Alternate View

Alignment

Place words one on top of the other, with gaps in the first word indicating insertions, and gaps in the second word indicating deletions.

$$\begin{array}{ccccc} \text{F} & \text{O} & \text{O} & & \text{D} \\ \text{M} & \text{O} & \text{N} & \text{E} & \text{Y} \end{array}$$

Formally, an alignment is a set M of pairs (i, j) such that each index appears at most once, and there is no "crossing": $i < i'$ and i is matched to j implies i' is matched to $j' > j$. In the above example, this is $M = \{(1,1), (2,2), (3,3), (4,5)\}$. Cost of an alignment is the number of mismatched columns.

Edit Distance Problem

Problem

Given two words, find the edit distance between them, i.e., an alignment of smallest cost.

Applications

- Spell-checkers and Dictionaries

Applications

- Spell-checkers and Dictionaries
- Unix `diff`

Applications

- Spell-checkers and Dictionaries
- Unix `diff`
- DNA sequence alignment

Applications

- Spell-checkers and Dictionaries
- Unix `diff`
- DNA sequence alignment ... but, we need a new metric

Similarity Metric

Definition

For two strings X and Y, the cost of alignment M is

- [Gap penalty] For each gap in the alignment, we incur a cost δ
- [Mismatch cost] For each pair p and q that have been matched in M, we incur cost α_{pq}; typically $\alpha_{pp} = 0$

Similarity Metric

Definition

For two strings X and Y, the cost of alignment M is

- [Gap penalty] For each gap in the alignment, we incur a cost δ
- [Mismatch cost] For each pair p and q that have been matched in M, we incur cost α_{pq}; typically $\alpha_{pp} = 0$

Edit distance is special case when $\delta = \alpha_{pq} = 1$

An Example

Example

$$\begin{array}{cccccccccc} o & & c & u & r & r & a & n & c & e \\ o & c & c & u & r & r & e & n & c & e \end{array} \qquad \text{Cost} \;= \delta + \alpha_{ae}$$

$$\begin{array}{ccccccccccc} o & & c & u & r & r & & a & n & c & e \\ o & c & c & u & r & r & e & & n & c & e \end{array} \qquad \text{Cost} \;= 3\delta$$

Sequence Alignment

Input Given two words X and Y, and gap penalty δ and mismatch costs α_{pq}

Goal Find alignment of minimum cost

Problem Structure

Observation

Let $X = x_1x_2\cdots x_m$ and $Y = y_1y_2\cdots y_n$. If (m, n) are not matched then either the m'th position of X remains unmatched or the n'th position of Y remains unmatched.

Problem Structure

Observation

Let $X = x_1x_2\cdots x_m$ and $Y = y_1y_2\cdots y_n$. If (m, n) are not matched then either the m'th position of X remains unmatched or the n'th position of Y remains unmatched.

- Case x_m and y_n are matched.

Problem Structure

Observation

Let $X = x_1 x_2 \cdots x_m$ and $Y = y_1 y_2 \cdots y_n$. If (m, n) are not matched then either the m'th position of X remains unmatched or the n'th position of Y remains unmatched.

- Case x_m and y_n are matched.
  - Pay mismatch cost $\alpha_{x_m y_n}$ plus cost of aligning strings $x_1 \cdots x_{m-1}$ and $y_1 \cdots y_{n-1}$

Problem Structure

Observation

Let $X = x_1 x_2 \cdots x_m$ and $Y = y_1 y_2 \cdots y_n$. If (m, n) are not matched then either the m'th position of X remains unmatched or the n'th position of Y remains unmatched.

- Case x_m and y_n are matched.
  - Pay mismatch cost $\alpha_{x_m y_n}$ plus cost of aligning strings $x_1 \cdots x_{m-1}$ and $y_1 \cdots y_{n-1}$
- Case x_m is unmatched.

Problem Structure

Observation

Let $X = x_1 x_2 \cdots x_m$ and $Y = y_1 y_2 \cdots y_n$. If (m, n) are not matched then either the m'th position of X remains unmatched or the n'th position of Y remains unmatched.

- Case x_m and y_n are matched.
  - Pay mismatch cost $\alpha_{x_m y_n}$ plus cost of aligning strings $x_1 \cdots x_{m-1}$ and $y_1 \cdots y_{n-1}$
- Case x_m is unmatched.
  - Pay gap penalty plus cost of aligning $x_1 \cdots x_{m-1}$ and $y_1 \cdots y_n$

Problem Structure

Observation

Let $X = x_1 x_2 \cdots x_m$ and $Y = y_1 y_2 \cdots y_n$. If (m, n) are not matched then either the m'th position of X remains unmatched or the n'th position of Y remains unmatched.

- Case x_m and y_n are matched.
  - Pay mismatch cost $\alpha_{x_m y_n}$ plus cost of aligning strings $x_1 \cdots x_{m-1}$ and $y_1 \cdots y_{n-1}$
- Case x_m is unmatched.
  - Pay gap penalty plus cost of aligning $x_1 \cdots x_{m-1}$ and $y_1 \cdots y_n$
- Case y_n is unmatched.

Problem Structure

Observation

Let $X = x_1 x_2 \cdots x_m$ and $Y = y_1 y_2 \cdots y_n$. If (m, n) are not matched then either the m'th position of X remains unmatched or the n'th position of Y remains unmatched.

$x_1\, x_2 \cdots x_m$

$y_1\, y_2 \cdots y_n$

- Case x_m and y_n are matched.
    - Pay mismatch cost $\alpha_{x_m y_n}$ plus cost of aligning strings $x_1 \cdots x_{m-1}$ and $y_1 \cdots y_{n-1}$
- Case x_m is unmatched.
    - Pay gap penalty plus cost of aligning $x_1 \cdots x_{m-1}$ and $y_1 \cdots y_n$
- Case y_n is unmatched.
    - Pay gap penalty plus cost of aligning $x_1 \cdots x_m$ and $y_1 \cdots y_{n-1}$

FOOD
MONEY

1 FOOD
MONEY

2 FOOD ⊔
MONEY

3 FOOD
MONEY ⊔

Subproblems and Recurrence

Optimal Costs

Let $\mathrm{Opt}(i,j)$ be optimal cost of aligning $x_1 \cdots x_i$ and $y_1 \cdots y_j$. Then

$$\mathrm{Opt}(i,j) = \min(\alpha_{x_i y_j} + \mathrm{Opt}(i-1,j-1), \delta + \mathrm{Opt}(i-1,j), \delta + \mathrm{Opt}(i,j-1))$$

Subproblems and Recurrence

Optimal Costs

Let $\mathrm{Opt}(i,j)$ be optimal cost of aligning $x_1 \cdots x_i$ and $y_1 \cdots y_j$. Then

$$\mathrm{Opt}(i,j) = \min(\alpha_{x_i y_j} + \mathrm{Opt}(i-1,j-1), \delta + \mathrm{Opt}(i-1,j), \delta + \mathrm{Opt}(i,j-1))$$

Base Cases: $\mathrm{Opt}(i,0) = \delta \cdot i$ and $\mathrm{Opt}(0,j) = \delta \cdot j$

Dynamic Programming Solution

```
for all i M[i,0] = iδ
for all j M[0,j] = 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])
```

Analysis

Dynamic Programming Solution

```
for all i M[i,0] = iδ
for all j M[0,j] = 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])
```

Analysis

- Running time is $O(mn)$

Dynamic Programming Solution

```
for all i M[i,0] = iδ
for all j M[0,j] = 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])
```

Analysis

- Running time is $O(mn)$
- Space used is $O(mn)$

Matrix and DAG of Computation

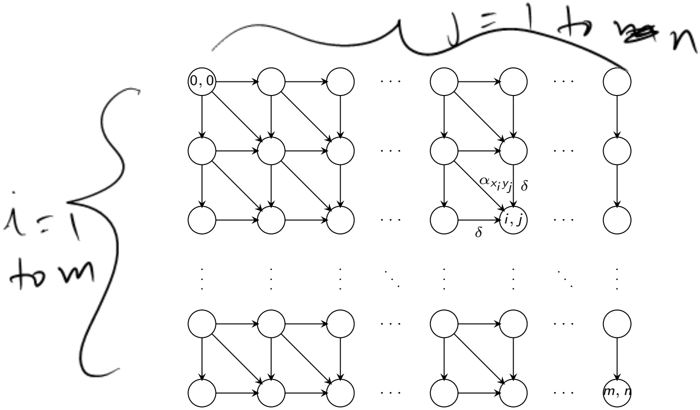


Figure: Iterative algorithm in previous slide computes values in row order. Optimal value is a shortest path from $(0, 0)$ to (m, n) in DAG.

Sequence Alignment in Practice

- Typically the DNA sequences that are aligned are about 10^5 letters long!

Sequence Alignment in Practice

- Typically the DNA sequences that are aligned are about 10^5 letters long!
- So about 10^{10} ops and 10^{10} bytes needed

Sequence Alignment in Practice

- Typically the DNA sequences that are aligned are about 10^5 letters long!
- So about 10^{10} ops and 10^{10} bytes needed
- The killer is the 10GB storage

Sequence Alignment in Practice

- Typically the DNA sequences that are aligned are about 10^5 letters long!
- So about 10^{10} ops and 10^{10} bytes needed
- The killer is the 10GB storage
- Can we reduce space requirements?

Optimizing Space

- Recall

$$M(i,j) = \min(\alpha_{x_i y_j} + M(i-1, j-1), \delta + M(i-1, j), \delta + M(i, j-1))$$

Optimizing Space

- Recall

$$M(i,j) = \min(\alpha_{x_i y_j} + M(i-1, j-1), \delta + M(i-1, j), \delta + M(i, j-1))$$

- Entries in jth column only depend on $(j-1)$'st column and earlier entries in jth column

Optimizing Space

- Recall

$$M(i,j) = \min(\alpha_{x_i y_j} + M(i-1,j-1), \delta + M(i-1,j), \delta + M(i,j-1))$$

- Entries in jth column only depend on $(j-1)$'st column and earlier entries in jth column
- Only store the current column and the previous column reusing space; $N(i,0)$ stores $M(i,j-1)$ and $N(i,1)$ stores $M(i,j)$

Computing in column order to save space

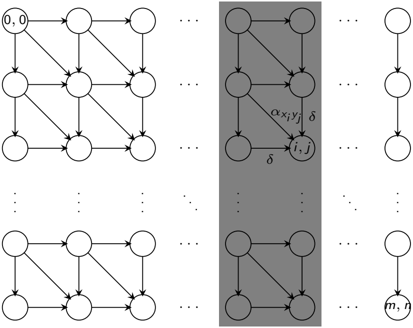


Figure: $M(i, j)$ only depends on previous column values. Keep only two columns and compute in column order.

Space Efficient Algorithm

```
for all i N[i,0] = iδ
for j = 1 to n
    N[0,1] = jδ (* corresponds to M(0,j) *)
    for i = 1 to m
        N[i,1] = min (α_{x_i y_j} + N[i-1,0], δ + N[i-1,1], δ + N[i,0])
    update N[i,0] = N[i,1]
```

Analysis

Running time is $O(mn)$ and space used is $O(2m) = O(m)$

Analyzing Space Efficiency

- From the $m \times n$ matrix M we can construct the actual alignment (exercise)

Analyzing Space Efficiency

- From the $m \times n$ matrix M we can construct the actual alignment (exercise)
- Matrix N computes cost of optimal alignment

Analyzing Space Efficiency

- From the $m \times n$ matrix M we can construct the actual alignment (exercise)
- Matrix N computes cost of optimal alignment but no way to construct the actual alignment

Analyzing Space Efficiency

- From the $m \times n$ matrix M we can construct the actual alignment (exercise)
- Matrix N computes cost of optimal alignment but no way to construct the actual alignment
- Space efficient computation of alignment? More complicated algorithm — see text book.