

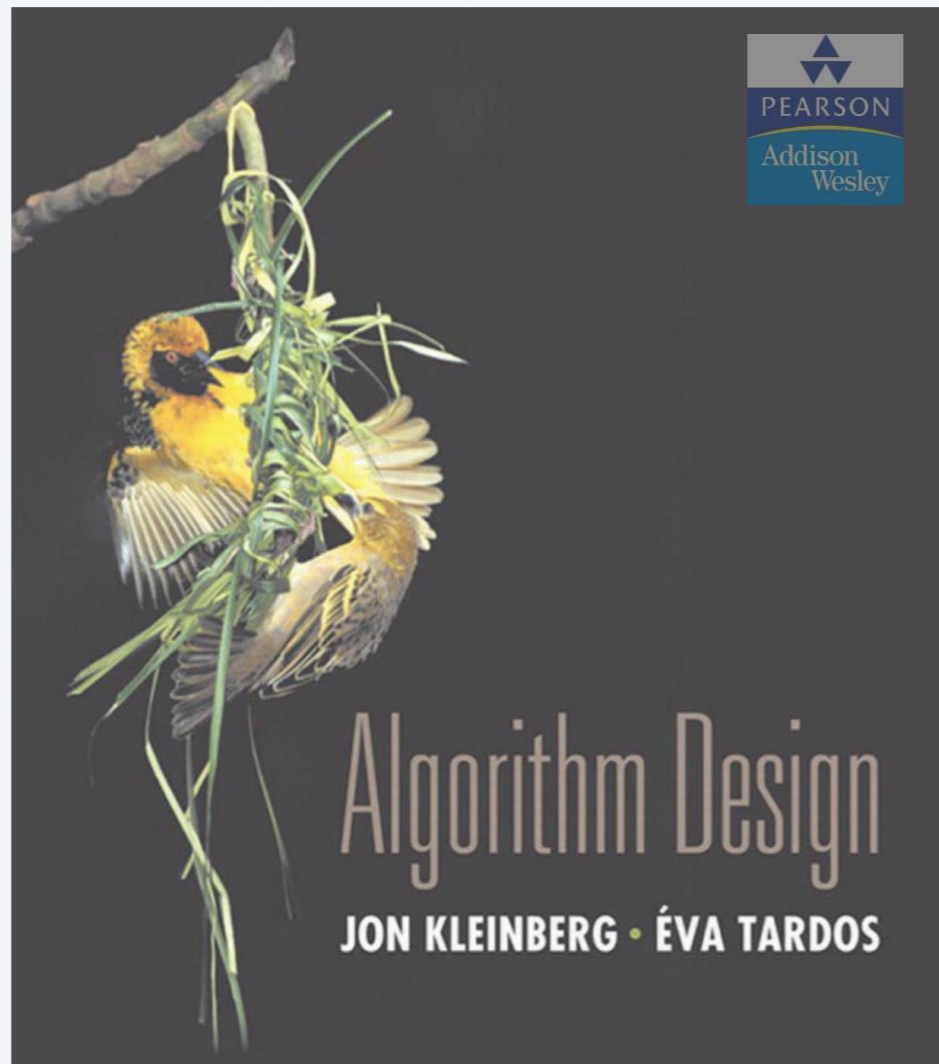
6. DYNAMIC PROGRAMMING II

- ▶ *sequence alignment*
- ▶ *Hirschberg's algorithm*
- ▶ *Bellman–Ford–Moore algorithm*

Lecture slides by Kevin Wayne

Copyright © 2005 Pearson-Addison Wesley

<http://www.cs.princeton.edu/~wayne/kleinberg-tardos>



SECTION 6.6

6. DYNAMIC PROGRAMMING II

- ▶ *sequence alignment*
- ▶ *Hirschberg's algorithm*
- ▶ *Bellman–Ford–Moore algorithm*

String similarity

Q. How similar are two strings?

Ex. occurrence and occurance.



6 mismatches, 1 gap



1 mismatch, 1 gap

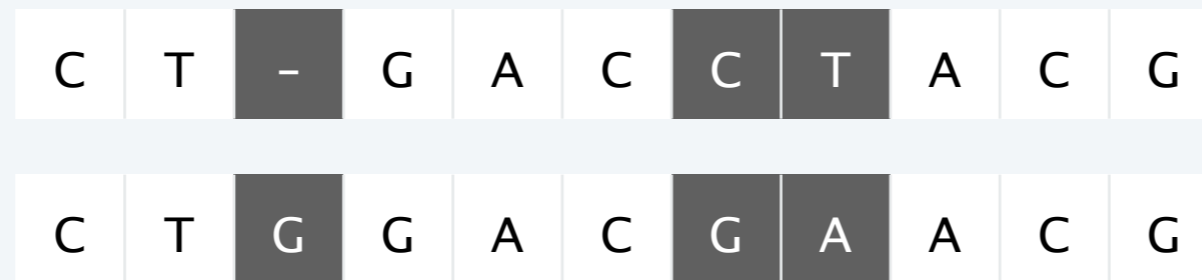


0 mismatches, 3 gaps

Edit distance

Edit distance. [Levenshtein 1966, Needleman–Wunsch 1970]

- Gap penalty δ ; mismatch penalty α_{pq} .
- Cost = sum of gap and mismatch penalties.



$$\text{cost} = \delta + \alpha_{CG} + \alpha_{TA}$$

← assuming $\alpha_{AA} = \alpha_{CC} = \alpha_{GG} = \alpha_{TT} = 0$

Applications. Bioinformatics, spell correction, machine translation, speech recognition, information extraction, ...

Confusion matrix for English text

Number of times one letter was substituted for another.

X	Y (correct)																									
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	0	7	1	342	0	0	2	118	0	1	0	0	3	76	0	0	1	35	9	9	0	1	0	5	0
b	0	0	9	9	2	2	3	1	0	0	0	5	11	5	0	10	0	0	2	1	0	0	8	0	0	0
c	6	5	0	16	0	9	5	0	0	0	1	0	7	9	1	10	2	5	39	40	1	3	7	1	1	0
d	1	10	13	0	12	0	5	5	0	0	2	3	7	3	0	1	0	43	30	22	0	0	4	0	2	0
e	388	0	3	11	0	2	2	0	89	0	0	3	0	5	93	0	0	14	12	6	15	0	1	0	18	0
f	0	15	0	3	1	0	5	2	0	0	0	3	4	1	0	0	0	6	4	12	0	0	2	0	0	0
g	4	1	11	11	9	2	0	0	0	1	1	3	0	0	2	1	3	5	13	21	0	0	1	0	3	0
h	1	8	0	3	0	0	0	0	0	0	2	0	12	14	2	3	0	3	1	11	0	0	2	0	0	0
i	103	0	0	0	146	0	1	0	0	0	0	6	0	0	49	0	0	0	2	1	47	0	2	1	15	0
j	0	1	1	9	0	0	1	0	0	0	0	2	1	0	0	0	0	0	5	0	0	0	0	0	0	0
k	1	2	8	4	1	1	2	5	0	0	0	0	5	0	2	0	0	0	6	0	0	0	4	0	0	3
l	2	10	1	4	0	4	5	6	13	0	1	0	0	14	2	5	0	11	10	2	0	0	0	0	0	0
m	1	3	7	8	0	2	0	6	0	0	4	4	0	180	0	6	0	0	9	15	13	3	2	2	3	0
n	2	7	6	5	3	0	1	19	1	0	4	35	78	0	0	7	0	28	5	7	0	0	1	2	0	2
o	91	1	1	3	116	0	0	0	25	0	2	0	0	0	0	14	0	2	4	14	39	0	0	0	18	0
p	0	11	1	2	0	6	5	0	2	9	0	2	7	6	15	0	0	1	3	6	0	4	1	0	0	0
q	0	0	1	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	0	14	0	30	12	2	2	8	2	0	5	8	4	20	1	14	0	0	12	22	4	0	0	1	0	0
s	11	8	27	33	35	4	0	1	0	1	0	27	0	6	1	7	0	14	0	15	0	0	5	3	20	1
t	3	4	9	42	7	5	19	5	0	1	0	14	9	5	5	6	0	11	37	0	0	2	19	0	7	6
u	20	0	0	0	44	0	0	0	64	0	0	0	0	2	43	0	0	4	0	0	0	0	2	0	8	0
v	0	0	7	0	0	3	0	0	0	0	0	1	0	0	1	0	0	0	8	3	0	0	0	0	0	0
w	2	2	1	0	1	0	0	2	0	0	1	0	0	0	0	7	0	6	3	3	1	0	0	0	0	0
x	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0
y	0	0	2	0	15	0	1	7	15	0	0	0	2	0	6	1	0	7	36	8	5	0	0	1	0	0
z	0	0	0	7	0	0	0	0	0	0	0	7	5	0	0	0	0	2	21	3	0	0	0	0	3	0

BLOSUM matrix for proteins

BLOcks SUBstitution Matrix: is a substitution matrix used for sequence alignment of proteins.

BLOSUM matrices are used to score alignments between evolutionarily divergent protein sequences.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	7	-3	-3	-3	-1	-2	-2	0	-3	-3	-3	-1	-2	-4	-1	2	0	-5	-4	-1
R	-3	9	-1	-3	-6	1	-1	-4	0	-5	-4	3	-3	-5	-3	-2	-2	-5	-4	-4
N	-3	-1	9	2	-5	0	-1	-1	1	-6	-6	0	-4	-6	-4	1	0	-7	-4	-5
D	-3	-3	2	10	-7	-1	2	-3	-2	-7	-7	-2	-6	-6	-3	-1	-2	-8	-6	-6
C	-1	-6	-5	-7	13	-5	-7	-6	-7	-2	-3	-6	-3	-4	-6	-2	-2	-5	-5	-2
Q	-2	1	0	-1	-5	9	3	-4	1	-5	-4	2	-1	-5	-3	-1	-1	-4	-3	-4
E	-2	-1	-1	2	-7	3	8	-4	0	-6	-6	1	-4	-6	-2	-1	-2	-6	-5	-4
G	0	-4	-1	-3	-6	-4	-4	9	-4	-7	-7	-3	-5	-6	-5	-1	-3	-6	-6	-6
H	-3	0	1	-2	-7	1	0	-4	12	-6	-5	-1	-4	-2	-4	-2	-3	-4	3	-5
I	-3	-5	-6	-7	-2	-5	-6	-7	-6	7	2	-5	2	-1	-5	-4	-2	-5	-3	4
L	-3	-4	-6	-7	-3	-4	-6	-7	-5	2	6	-4	3	0	-5	-4	-3	-4	-2	1
K	-1	3	0	-2	-6	2	1	-3	-1	-5	-4	8	-3	-5	-2	-1	-1	-6	-4	-4
M	-2	-3	-4	-6	-3	-1	-4	-5	-4	2	3	-3	9	0	-4	-3	-1	-3	-3	1
F	-4	-5	-6	-6	-4	-5	-6	-6	-2	-1	0	-5	0	10	-6	-4	-4	0	4	-2
P	-1	-3	-4	-3	-6	-3	-2	-5	-4	-5	-5	-2	-4	-6	12	-2	-3	-7	-6	-4
S	2	-2	1	-1	-2	-1	-1	-1	-2	-4	-4	-1	-3	-4	-2	7	2	-6	-3	-3
T	0	-2	0	-2	-2	-1	-2	-3	-3	-2	-3	-1	-1	-4	-3	2	8	-5	-3	0
W	-5	-5	-7	-8	-5	-4	-6	-6	-4	-5	-4	-6	-3	0	-7	-6	-5	16	3	-5
Y	-4	-4	-4	-6	-5	-3	-5	-6	3	-3	-2	-4	-3	4	-6	-3	-3	3	11	-3
V	-1	-4	-5	-6	-2	-4	-4	-6	-5	4	1	-4	1	-2	-4	-3	0	-5	-3	7

Edit distance

One more example.

What is edit distance between these two strings?

P A L E T T E

P A L A T E

Assume gap penalty = 2 and mismatch penalty = 1.



1 gap, 1 mismatch

Sequence alignment

Goal. Given two strings $x_1 x_2 \dots x_m$ and $y_1 y_2 \dots y_n$, find a min-cost alignment.

Def. An **alignment** M is a set of ordered pairs $x_i - y_j$ such that each character appears in at most one pair and no crossings.

$x_i - y_j$ and $x_{i'} - y_{j'}$ cross if $i < i'$, but $j > j'$

Def. The **cost** of an alignment M is:

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

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

an alignment of CTACCG and TACATG

$$M = \{ x_2 - y_1, x_3 - y_2, x_4 - y_3, x_5 - y_4, x_6 - y_6 \}$$

Sequence alignment: problem structure

Def. $OPT(i, j)$ = min cost of aligning prefix strings $x_1 x_2 \dots x_i$ and $y_1 y_2 \dots y_j$.

Goal. $OPT(m, n)$.

Case 1. $OPT(i, j)$ matches $x_i - y_j$.

Pay mismatch for $x_i - y_j$ + min cost of aligning $x_1 x_2 \dots x_{i-1}$ and $y_1 y_2 \dots y_{j-1}$.

Case 2a. $OPT(i, j)$ leaves x_i unmatched.

Pay gap for x_i + min cost of aligning $x_1 x_2 \dots x_{i-1}$ and $y_1 y_2 \dots y_j$.

Case 2b. $OPT(i, j)$ leaves y_j unmatched.

Pay gap for y_j + min cost of aligning $x_1 x_2 \dots x_i$ and $y_1 y_2 \dots y_{j-1}$.

optimal substructure property
(proof via exchange argument)

Bellman equation.

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

Sequence alignment: bottom-up algorithm

SEQUENCE-ALIGNMENT($m, n, x_1, \dots, x_m, y_1, \dots, y_n, \delta, \alpha$)

FOR $i = 0$ TO m

$M[i, 0] \leftarrow i \delta.$


FOR $j = 0$ TO n

$M[0, j] \leftarrow j \delta.$

FOR $i = 1$ TO m

FOR $j = 1$ TO n

$M[i, j] \leftarrow \min \{ \alpha_{x_i y_j} + M[i-1, j-1],$
 $\delta + M[i-1, j],$
 $\delta + M[i, j-1] \}.$



already
computed

RETURN $M[m, n].$

Sequence alignment: traceback

		P	A	L	A	T	E
	0	2	4	6	8	10	12
P	2	0	2	4	6	8	10
A	4	2	0	2	4	6	8
L	6	4	2	0	2	4	6
E	8	6	4	2	1	3	4
T	10	8	6	4	3	1	3
T	12	10	8	6	5	3	2
E	14	12	10	8	7	5	3



1 gap, 1 mismatch
(gap penalty = 2, mismatch penalty = 1)

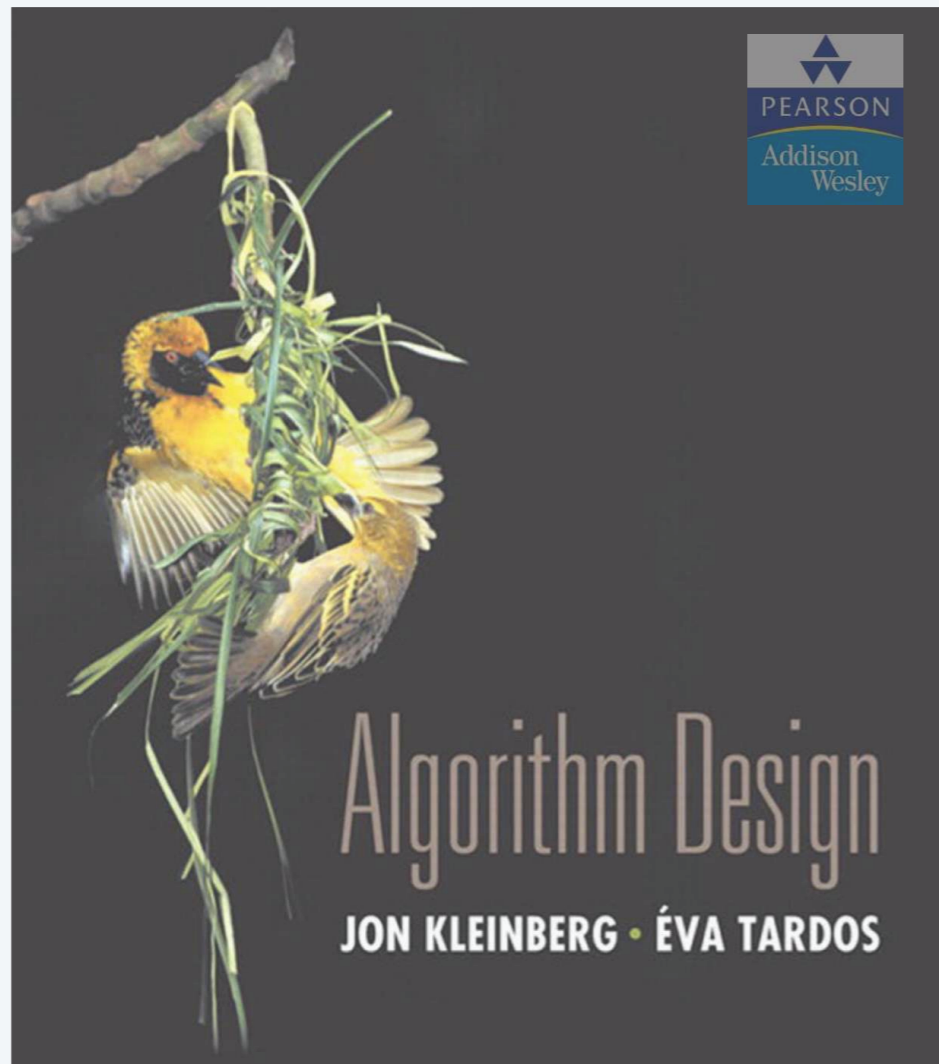
Sequence alignment: analysis

Theorem. The DP algorithm computes the edit distance (and an optimal alignment) of two strings of lengths m and n in $\Theta(mn)$ time and space.

Pf.

- Algorithm computes edit distance.
- Can trace back to extract optimal alignment itself. ▪

Can we improve the space used by the algorithm?



SECTION 6.7

6. DYNAMIC PROGRAMMING II

- ▶ *sequence alignment*
- ▶ *Hirschberg's algorithm*
- ▶ *Bellman–Ford–Moore algorithm*

Sequence alignment in linear space

Theorem. [Hirschberg] There exists an algorithm to find an optimal alignment in $O(mn)$ time and $O(m + n)$ space.

- Clever combination of divide-and-conquer and dynamic programming.

Programming
Techniques

G. Manacher
Editor

A Linear Space Algorithm for Computing Maximal Common Subsequences

D.S. Hirschberg
Princeton University

The problem of finding a longest common subsequence of two strings has been solved in quadratic time and space. An algorithm is presented which will solve this problem in quadratic time and in linear space.

Key Words and Phrases: subsequence, longest common subsequence, string correction, editing

CR Categories: 3.63, 3.73, 3.79, 4.22, 5.25



Improving the space: first trick

	P	A	L	A	T	E	
	0	2	4	6	8	10	12
P	2	0	2	4	6	8	10
A	4	2	0	2	4	6	8
L	6	4	2	0	2	4	6
E	8	6	4	2	1	3	4
T	10	8	6	4	3	1	3
T	12	10	8	6	5	3	2
E	14	12	10	8	7	5	3

To compute the next column/row of the matrix we need only the previous column/row



maintain only 2 column/row a time



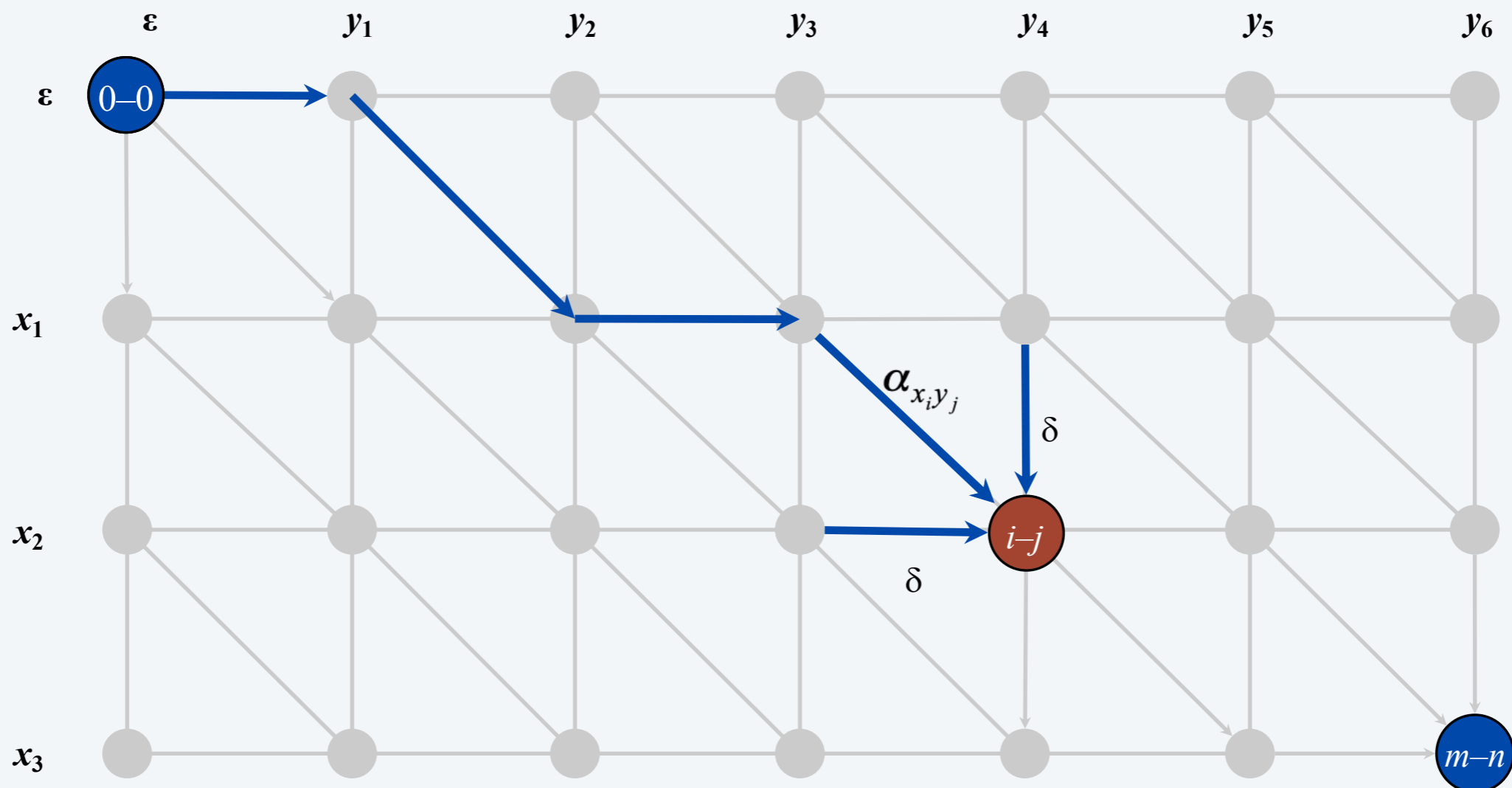
$O(m+n)$ space

notice: this allows to compute the edit distance but not the alignment

Hirschberg's algorithm

Edit distance graph.

- Let $f(i, j)$ denote length of shortest path from $(0,0)$ to (i, j) .
- Lemma: $f(i, j) = OPT(i, j)$ for all i and j .



Hirschberg's algorithm

Edit distance graph.

- Let $f(i, j)$ denote length of shortest path from $(0,0)$ to (i, j) .
- Lemma: $f(i, j) = OPT(i, j)$ for all i and j .

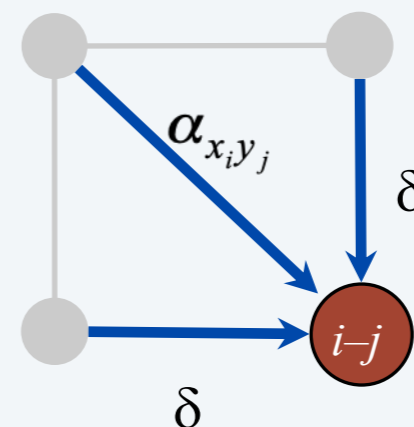
Pf of Lemma. [by strong induction on $i + j$]

- Base case: $f(0, 0) = OPT(0, 0) = 0$.
- Inductive hypothesis: assume true for all (i', j') with $i' + j' < i + j$.
- Last edge on shortest path to (i, j) is from $(i - 1, j - 1)$, $(i - 1, j)$, or $(i, j - 1)$.
- Thus,

$$\begin{aligned} f(i, j) &= \min\{\alpha_{x_i y_j} + f(i - 1, j - 1), \delta + f(i - 1, j), \delta + f(i, j - 1)\} \\ &= \min\{\alpha_{x_i y_j} + OPT(i - 1, j - 1), \delta + OPT(i - 1, j), \delta + OPT(i, j - 1)\} \\ &= OPT(i, j) \quad \blacksquare \end{aligned}$$

inductive hypothesis

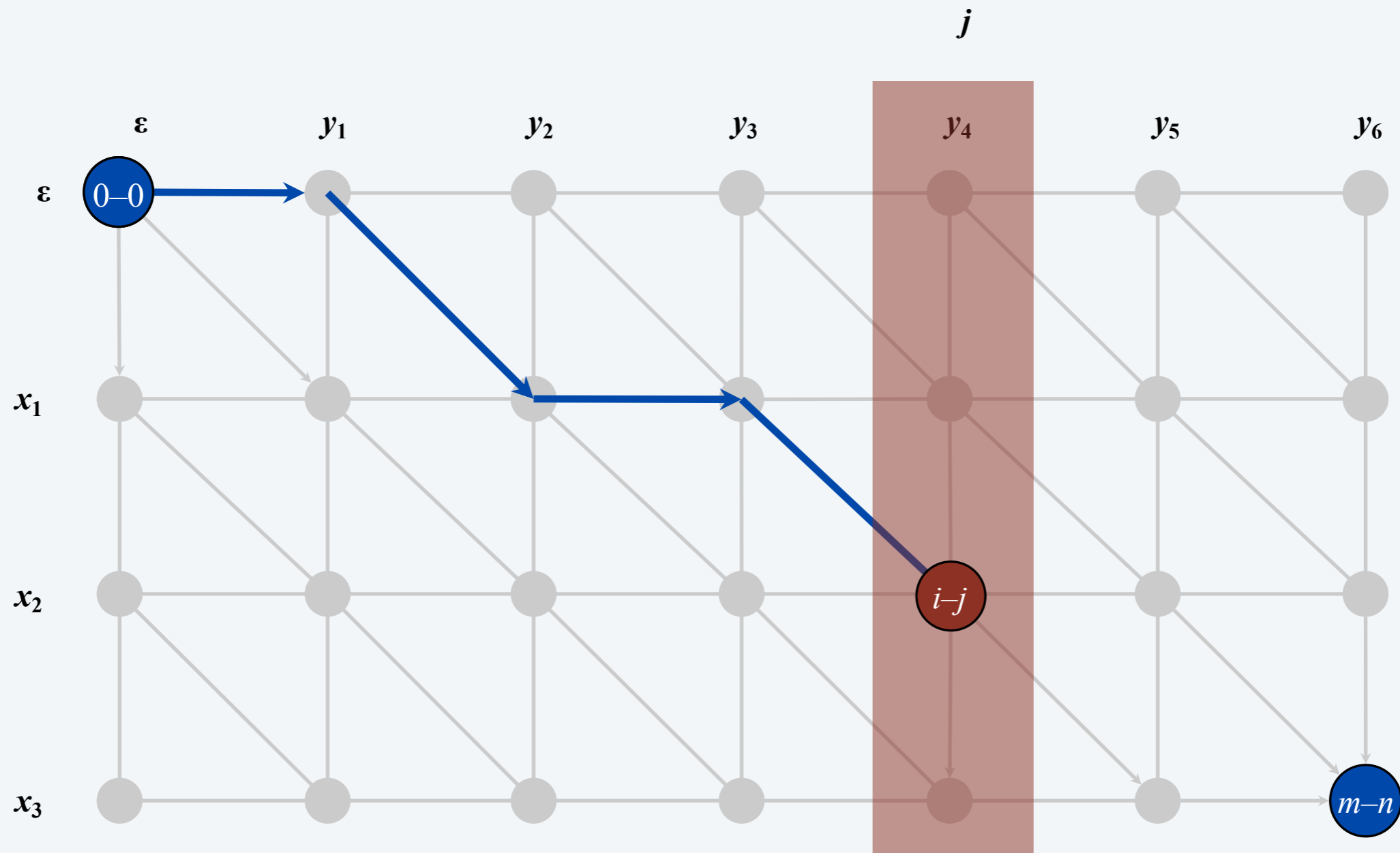
Bellman equation



Hirschberg's algorithm

Edit distance graph.

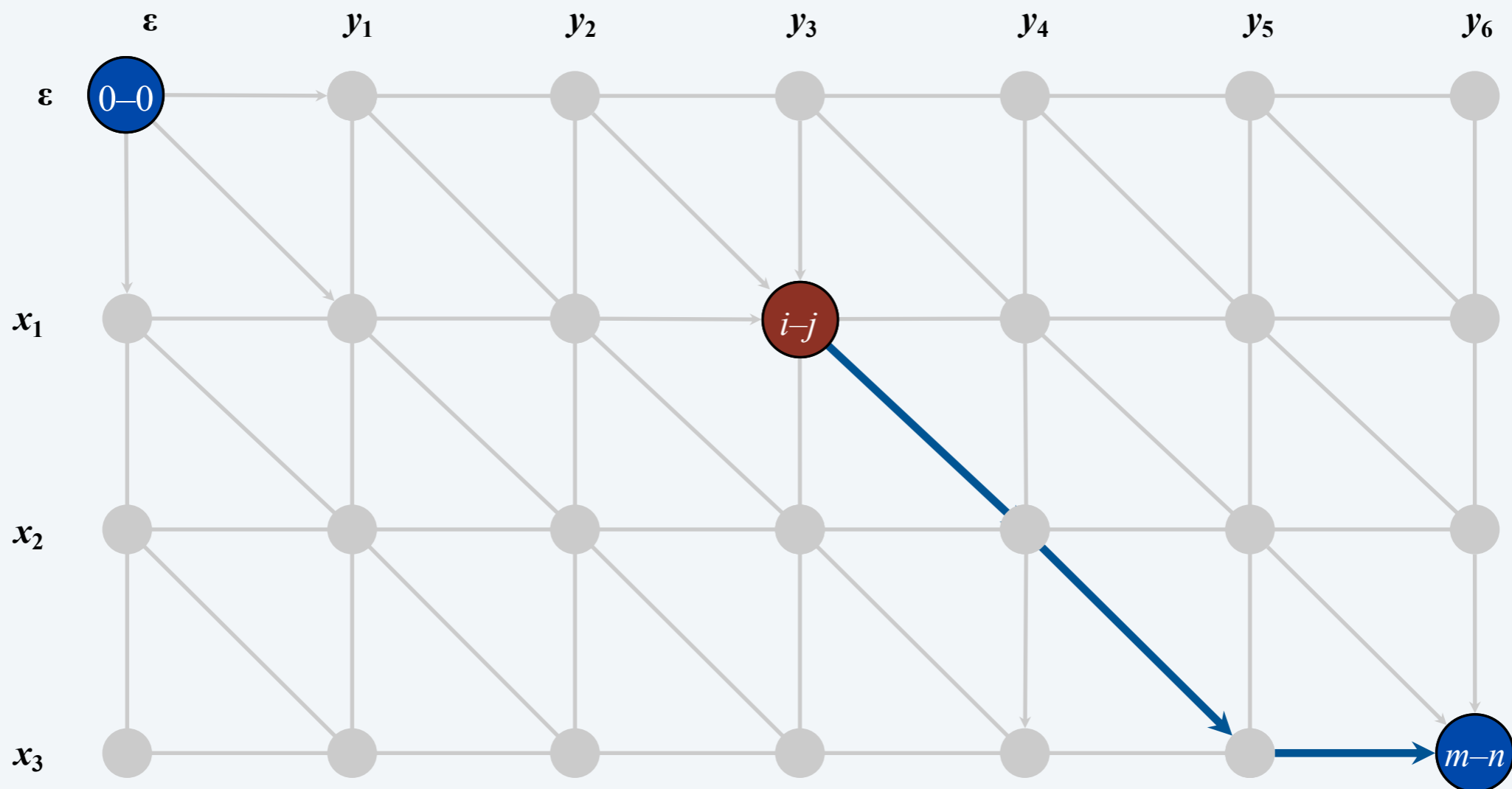
- Let $f(i, j)$ denote length of shortest path from $(0,0)$ to (i, j) .
- Lemma: $f(i, j) = OPT(i, j)$ for all i and j .
- Can compute $f(\cdot, j)$ for any j in $O(mn)$ time and $O(m + n)$ space.



Hirschberg's algorithm

Edit distance graph.

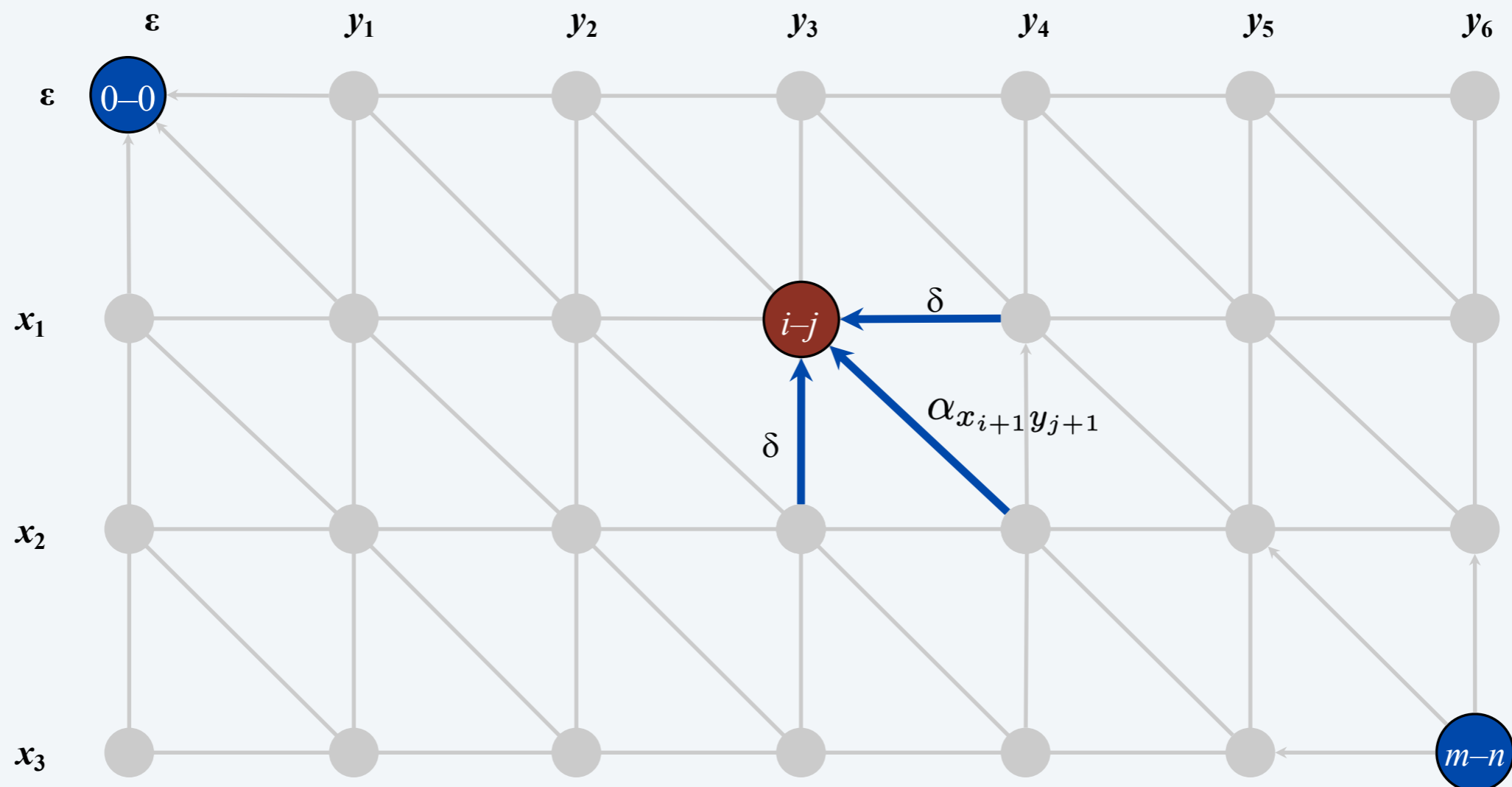
- Let $g(i, j)$ denote length of shortest path from (i, j) to (m, n) .



Hirschberg's algorithm

Edit distance graph.

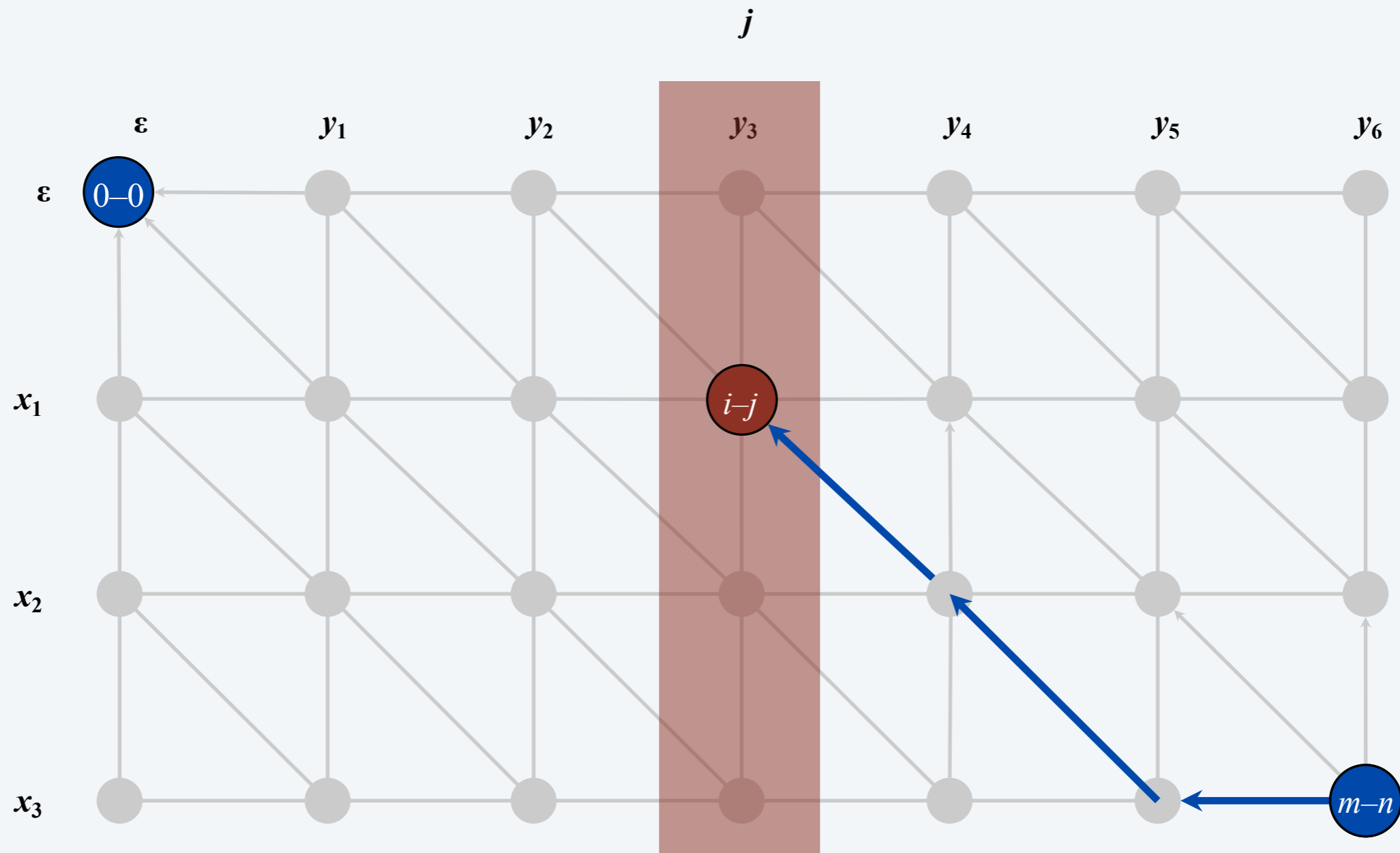
- Let $g(i, j)$ denote length of shortest path from (i, j) to (m, n) .
- Can compute $g(i, j)$ by reversing the edge orientations and inverting the roles of $(0, 0)$ and (m, n) .



Hirschberg's algorithm

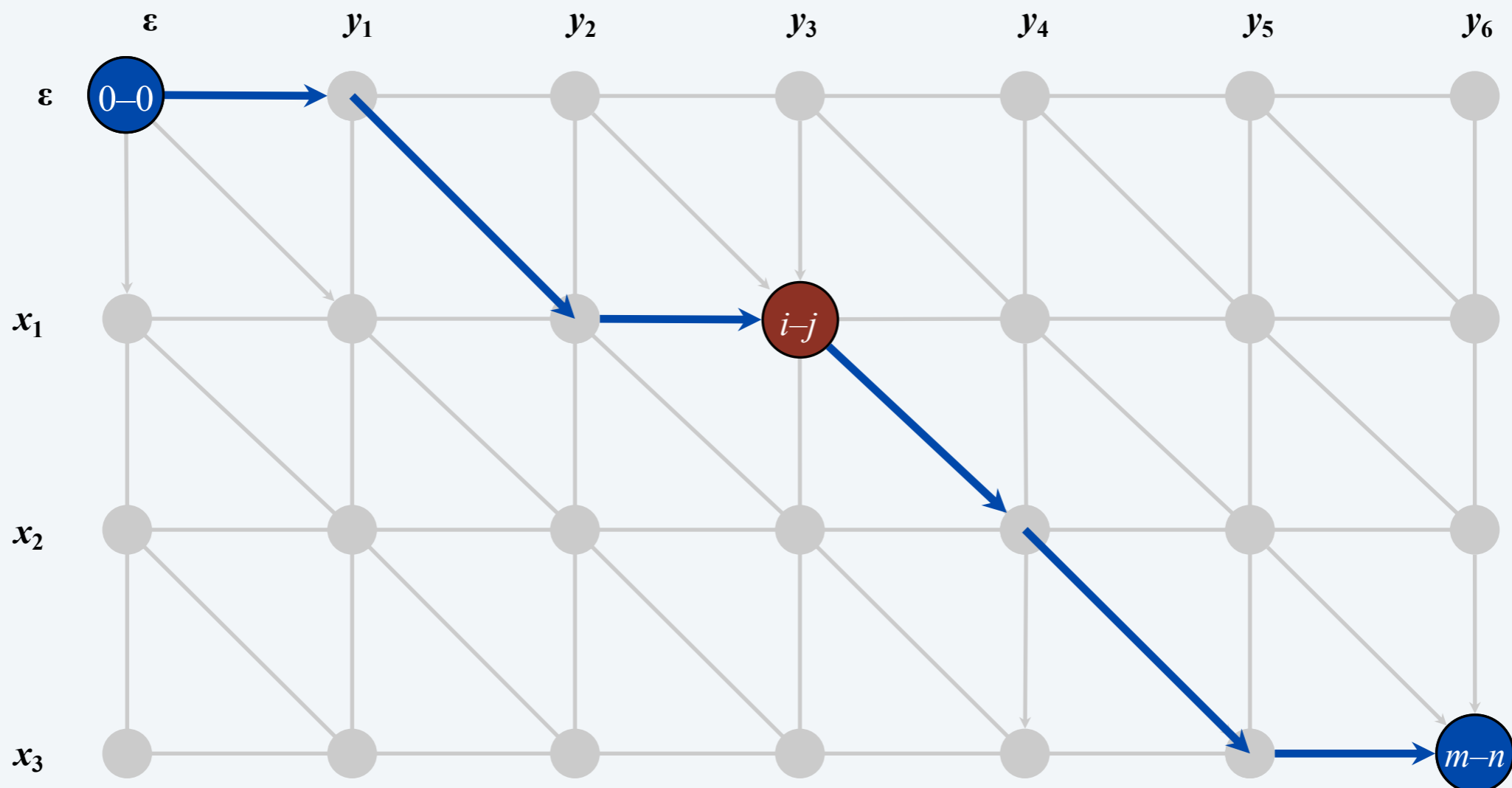
Edit distance graph.

- Let $g(i, j)$ denote length of shortest path from (i, j) to (m, n) .
- Can compute $g(\cdot, j)$ for any j in $O(mn)$ time and $O(m + n)$ space.



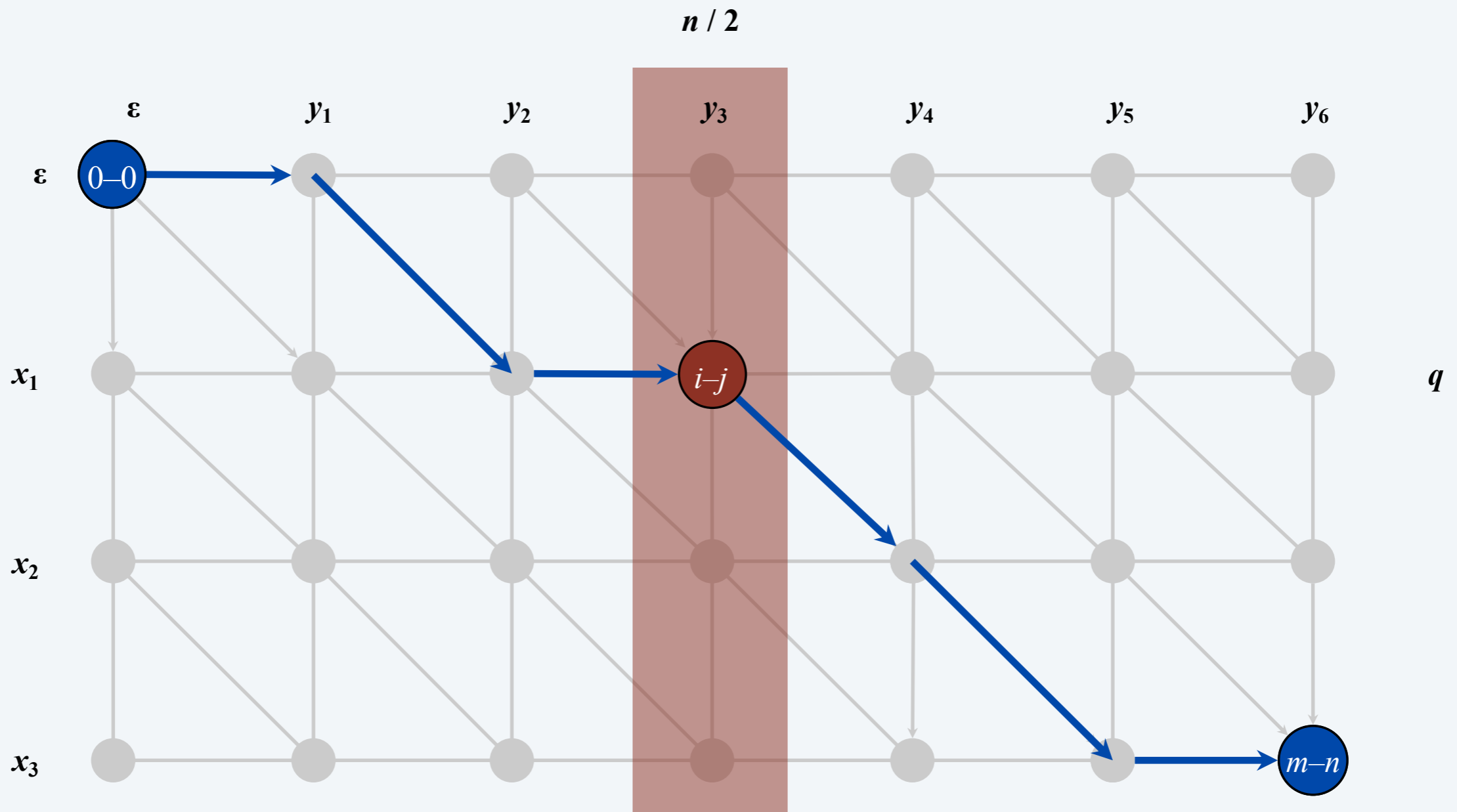
Hirschberg's algorithm

Observation 1. The length of a shortest path that uses (i, j) is $f(i, j) + g(i, j)$.



Hirschberg's algorithm

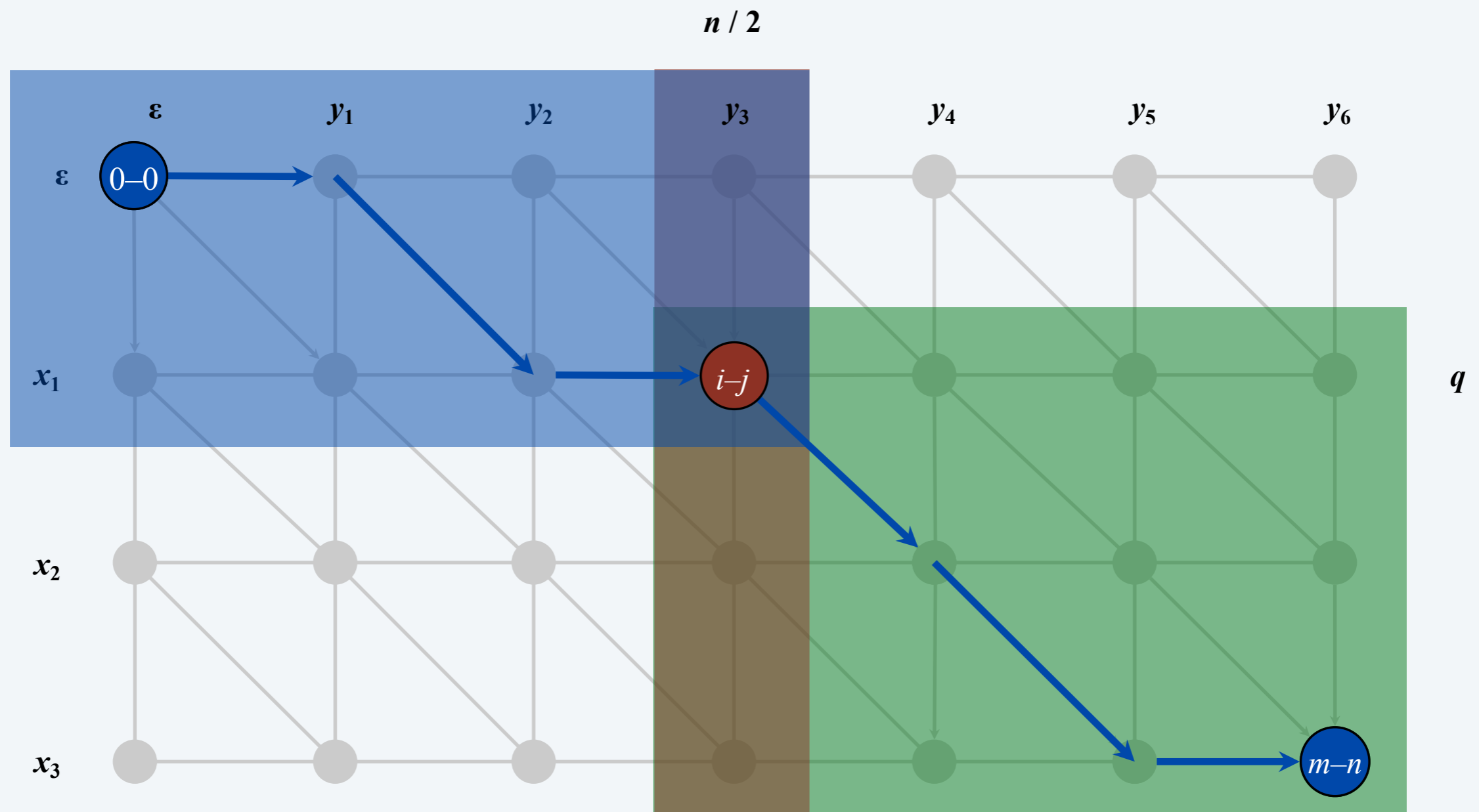
Observation 2. let q be an index that minimizes $f(q, n/2) + g(q, n/2)$.
Then, there exists a shortest path from $(0, 0)$ to (m, n) that uses $(q, n/2)$.



Hirschberg's algorithm

Divide. Find index q that minimizes $f(q, n/2) + g(q, n/2)$; save node $i-j$ as part of solution.

Conquer. Recursively compute optimal alignment in each piece.



Hirschberg's algorithm: space analysis

Theorem. Hirschberg's algorithm uses $\Theta(m + n)$ space.

Pf.

- Each recursive call uses $\Theta(m)$ space to compute $f(\cdot, n/2)$ and $g(\cdot, n/2)$.
- Only $\Theta(1)$ space needs to be maintained per recursive call.
- Number of recursive calls $\leq n$. ▪


Hirschberg's algorithm: running time analysis

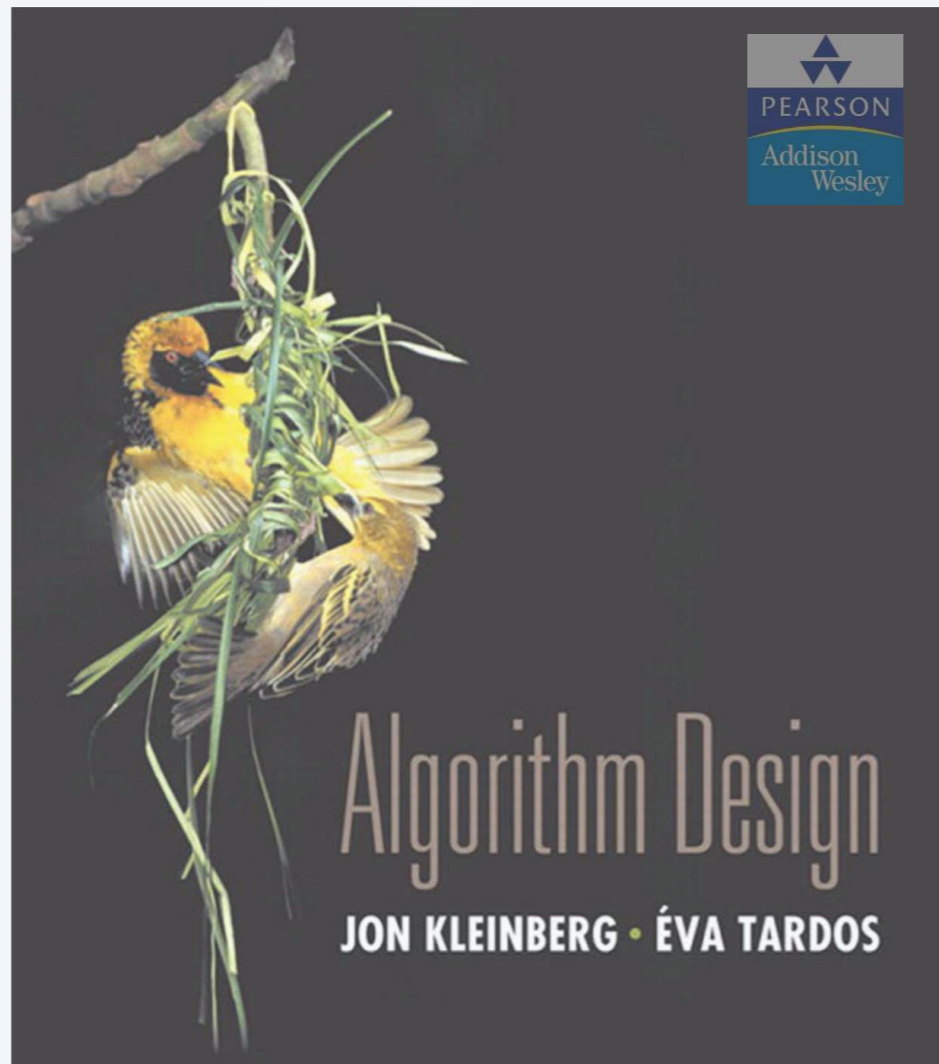
Theorem. Let $T(m, n)$ = max running time of Hirschberg's algorithm on strings of lengths at most m and n . Then, $T(m, n) = O(mn)$.

Pf. [by strong induction on $m + n$]

- $O(mn)$ time to compute $f(\cdot, n/2)$ and $g(\cdot, n/2)$ and find index q .
- $T(q, n/2) + T(m - q, n/2)$ time for two recursive calls.
- Choose constant c so that:
$$T(m, 2) \leq cm$$
$$T(2, n) \leq cn$$
$$T(m, n) \leq cmn + T(q, n/2) + T(m - q, n/2)$$
- Claim. $T(m, n) \leq 2cmn$.
- Base cases: $m = 2$ and $n = 2$.
- Inductive hypothesis: $T(m', n') \leq 2cm'n'$ for all (m', n') with $m' + n' < m + n$.

$$\begin{aligned} T(m, n) &\leq T(q, n/2) + T(m - q, n/2) + cmn \\ &\leq 2cq n/2 + 2c(m - q) n/2 + cmn \\ &= cq n + cmn - cq n + cmn \\ &= 2cmn \quad \blacksquare \end{aligned}$$

inductive hypothesis 



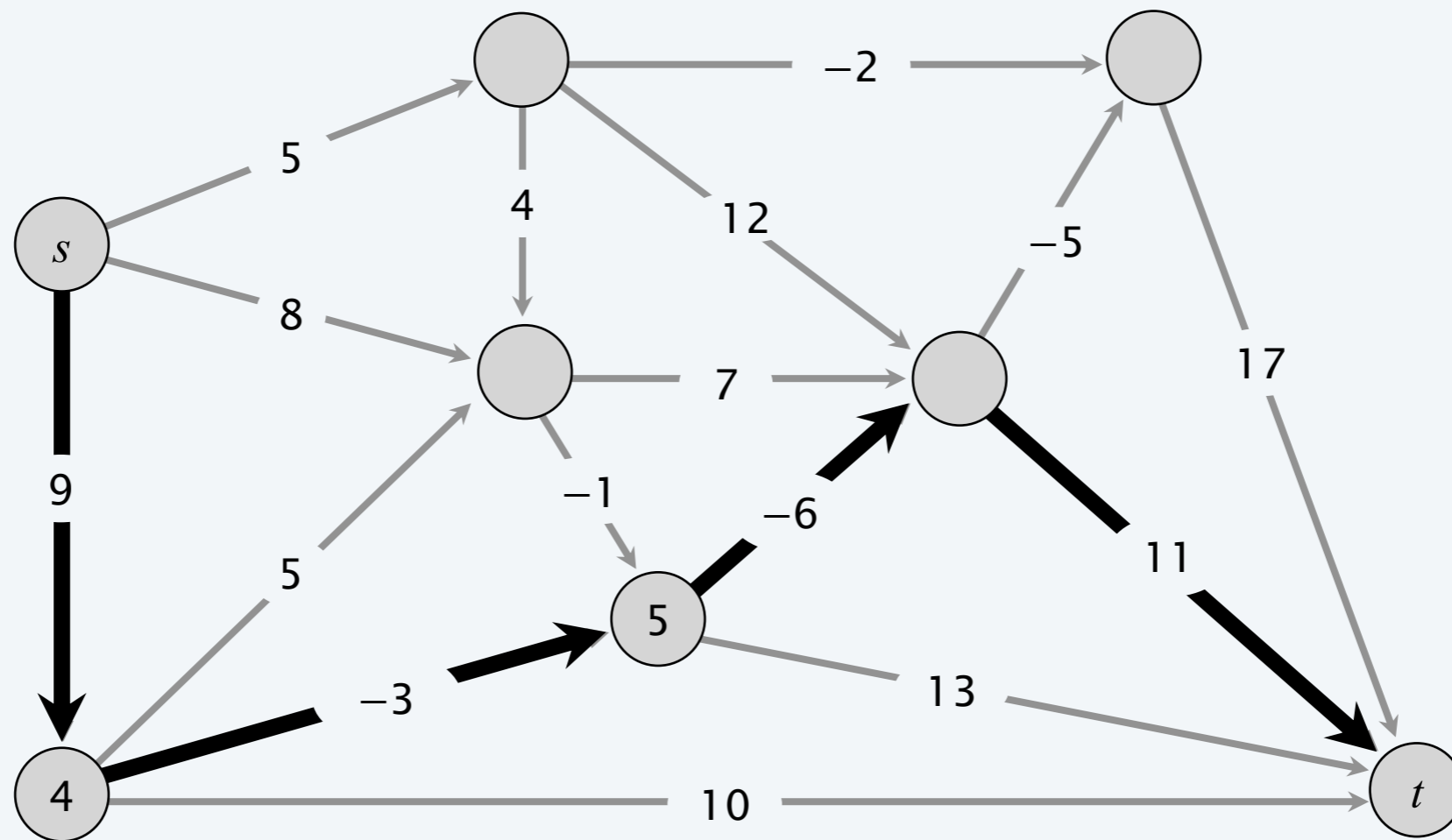
SECTION 6.8

6. DYNAMIC PROGRAMMING II

- ▶ *sequence alignment*
- ▶ *Hirschberg's algorithm*
- ▶ *Bellman–Ford–Moore algorithm*

Shortest paths with negative weights

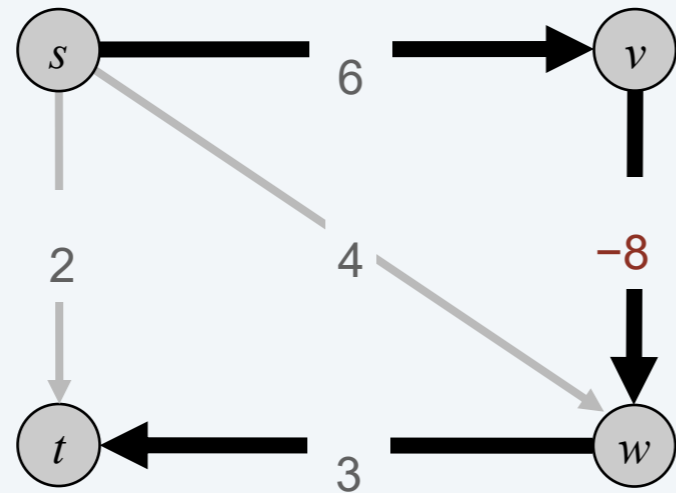
Shortest-path problem. Given a digraph $G = (V, E)$, with arbitrary edge lengths ℓ_{vw} , find shortest path from source node s to destination node t .



length of shortest $s \rightsquigarrow t$ path = $9 - 3 - 6 + 11 = 11$

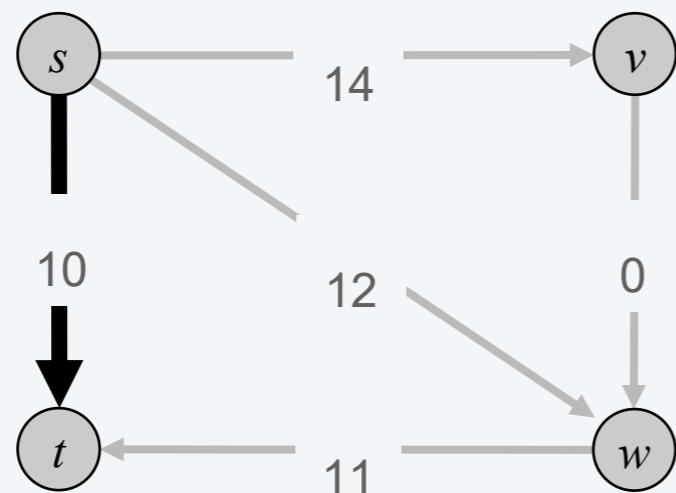
Shortest paths with negative weights: failed attempts

Dijkstra. May not produce shortest paths when edge lengths are negative.



Dijkstra selects the vertices in the order s, t, w, v
But shortest path from s to t is $s \rightarrow v \rightarrow w \rightarrow t$.

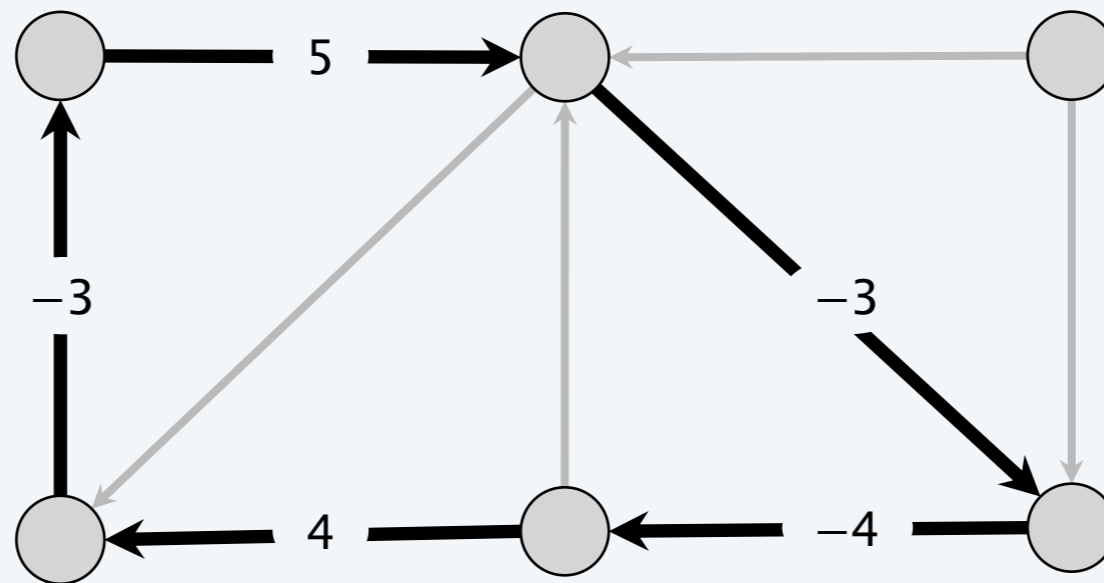
Reweighting. Adding a constant to every edge length does not necessarily make Dijkstra's algorithm produce shortest paths.



Adding 8 to each edge weight changes the shortest path from $s \rightarrow v \rightarrow w \rightarrow t$ to $s \rightarrow t$.

Negative cycles

Def. A **negative cycle** is a directed cycle for which the sum of its edge lengths is negative.

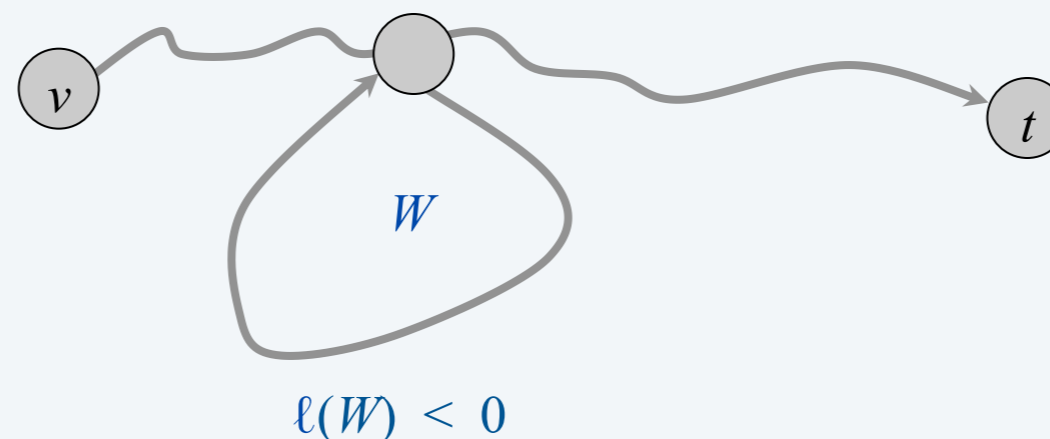


a negative cycle W : $l(W) = \sum_{e \in W} l_e < 0$

Shortest paths and negative cycles

Lemma 1. If some $v \rightsquigarrow t$ path contains a negative cycle, then there does not exist a shortest $v \rightsquigarrow t$ path.

Pf. If there exists such a cycle W , then can build a $v \rightsquigarrow t$ path of arbitrarily negative length by detouring around W as many times as desired. ▀

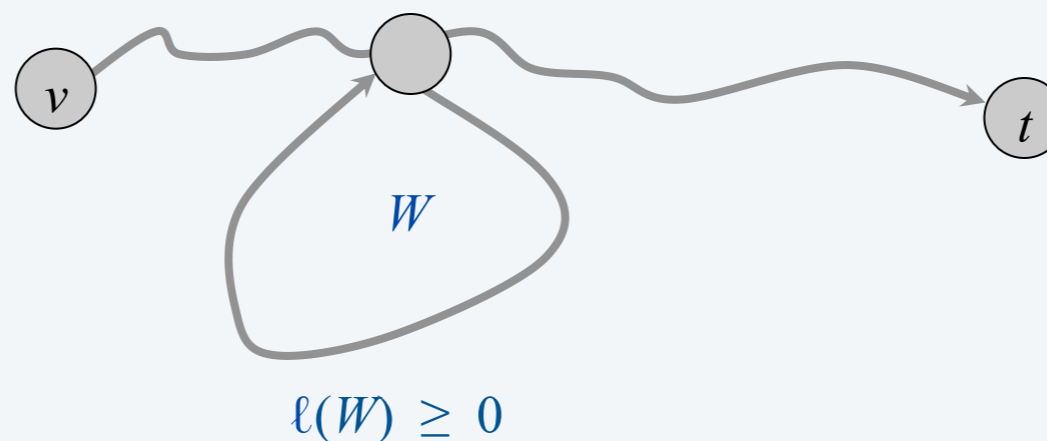


Shortest paths and negative cycles

Lemma 2. If G has no negative cycles, then there exists a shortest $v \rightsquigarrow t$ path that is simple (and has $\leq n - 1$ edges).

Pf.

- Among all shortest $v \rightsquigarrow t$ paths, consider one that uses the fewest edges.
- If that path P contains a directed cycle W , can remove the portion of P corresponding to W without increasing its length. ▪

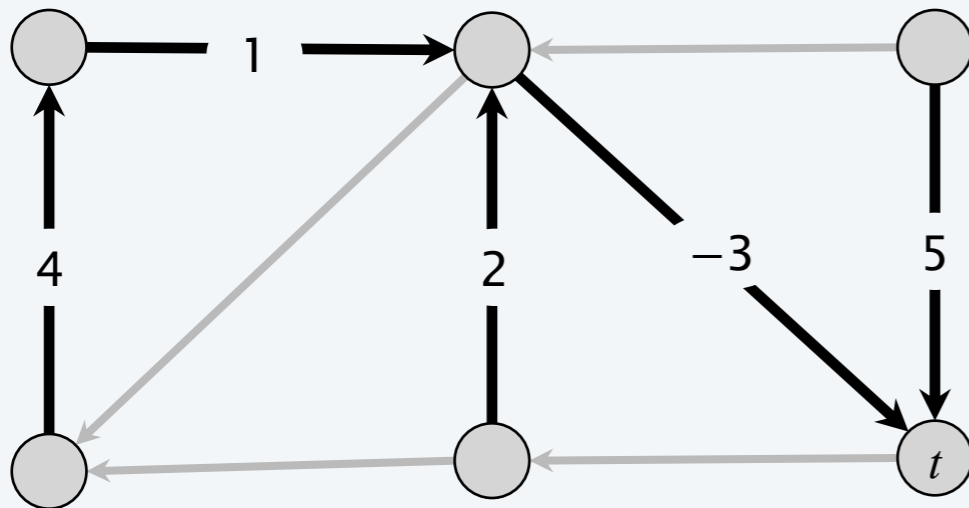


Shortest-paths and negative-cycle problems

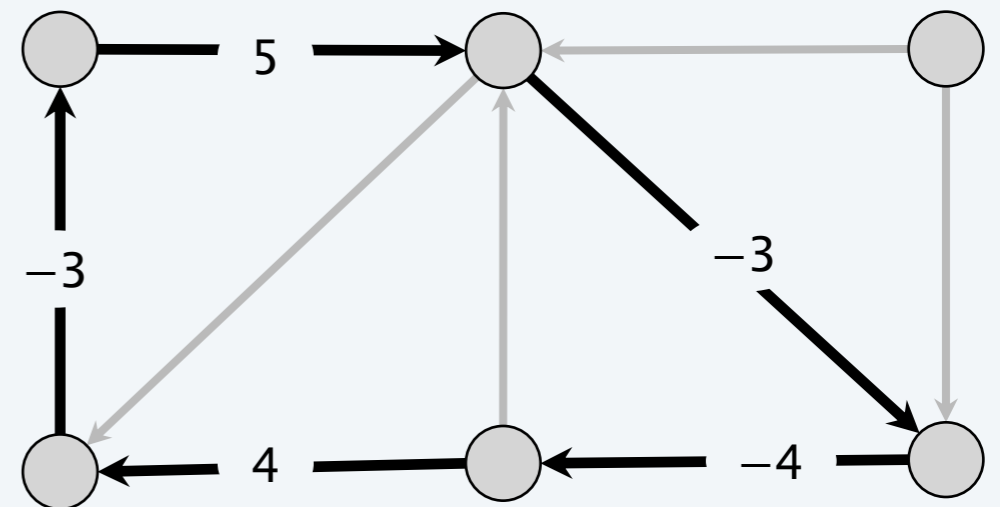
Single-destination shortest-paths problem. Given a digraph $G = (V, E)$ with edge lengths ℓ_{vw} (but no negative cycles) and a distinguished node t , find a shortest $v \rightsquigarrow t$ path for every node v .

← equivalent to the single-source shortest-paths problem

Negative-cycle problem. Given a digraph $G = (V, E)$ with edge lengths ℓ_{vw} , find a negative cycle (if one exists).



(reverse) shortest-paths tree



negative cycle


Shortest paths with negative weights: dynamic programming

Def. $OPT(i, v)$ = length of shortest $v \rightsquigarrow t$ path that uses $\leq i$ edges.

Goal. $OPT(n - 1, v)$ for each v .  by Lemma 2, if no negative cycles, there exists a shortest $v \rightsquigarrow t$ path that is simple

Case 1. Shortest $v \rightsquigarrow t$ path uses $\leq i - 1$ edges.

- $OPT(i, v) = OPT(i - 1, v)$.

 optimal substructure property
(proof via exchange argument)

Case 2. Shortest $v \rightsquigarrow t$ path uses exactly i edges.

- if (v, w) is first edge in shortest such $v \rightsquigarrow t$ path, incur a cost of ℓ_{vw} .
- Then, select best $w \rightsquigarrow t$ path using $\leq i - 1$ edges.

Bellman equation.

$$OPT(i, v) = \begin{cases} 0 & \text{if } i = 0 \text{ and } v = t \\ \infty & \text{if } i = 0 \text{ and } v \neq t \\ \min \left\{ OPT(i - 1, v), \min_{(v,w) \in E} \{OPT(i - 1, w) + \ell_{vw}\} \right\} & \text{if } i > 0 \end{cases}$$

Shortest paths with negative weights: implementation

SHORTEST-PATHS(V, E, ℓ, t)

FOREACH node $v \in V$:

$M[0, v] \leftarrow \infty$.

$M[0, t] \leftarrow 0$.

FOR $i = 1$ TO $n - 1$

FOREACH node $v \in V$:

$M[i, v] \leftarrow M[i - 1, v]$.

FOREACH edge $(v, w) \in E$:

$M[i, v] \leftarrow \min \{ M[i, v], M[i - 1, w] + \ell_{vw} \}$.

Shortest paths with negative weights: implementation

Theorem 1. Given a digraph $G = (V, E)$ with no negative cycles, the DP algorithm computes the length of a shortest $v \rightsquigarrow t$ path for every node v in $\Theta(mn)$ time and $\Theta(n^2)$ space.

Pf.

- Table requires $\Theta(n^2)$ space.
- Each iteration i takes $\Theta(m)$ time since we examine each edge once. ▪

Finding the shortest paths.

- Approach 1: Maintain $successor[i, v]$ that points to next node on a shortest $v \rightsquigarrow t$ path using $\leq i$ edges.
- Approach 2: Compute optimal lengths $M[i, v]$ and consider only edges with $M[i, v] = M[i - 1, w] + \ell_{vw}$. Any directed path in this subgraph is a shortest path.

Shortest paths with negative weights: practical improvements

Space optimization. Maintain two 1D arrays (instead of 2D array).

- $d[v]$ = length of a shortest $v \rightsquigarrow t$ path that we have found so far.
- $successor[v]$ = next node on a $v \rightsquigarrow t$ path.

Performance optimization. If $d[w]$ was not updated in iteration $i - 1$, then no reason to consider edges entering w in iteration i .

Bellman–Ford–Moore: efficient implementation

BELLMAN–FORD–MOORE(V, E, ℓ, t)

FOREACH node $v \in V$:

$d[v] \leftarrow \infty$.

$successor[v] \leftarrow null$.

$d[t] \leftarrow 0$.

FOR $i = 1$ **TO** $n - 1$

FOREACH node $w \in V$:

IF ($d[w]$ was updated in previous pass)


FOREACH edge $(v, w) \in E$:

IF ($d[v] > d[w] + \ell_{vw}$)

$d[v] \leftarrow d[w] + \ell_{vw}$.

$successor[v] \leftarrow w$.

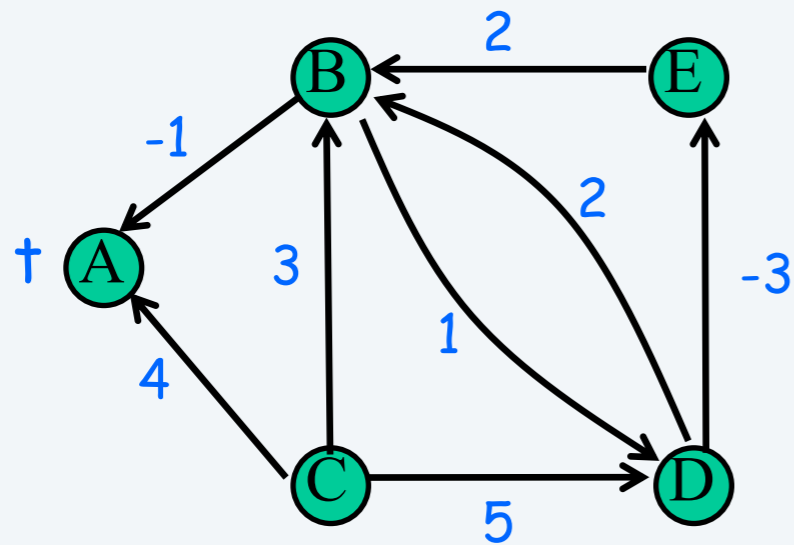
IF (no $d[\cdot]$ value changed in pass i) **STOP**.



pass i
 $O(m)$ time

Example: try to run Bellman-Ford algorithm on the following graph.

Order of the nodes: t, D, C, B, E



Bellman–Ford–Moore: analysis

Lemma 3. For each node v : $d[v]$ is the length of some $v \rightsquigarrow t$ path.

Lemma 4. For each node v : $d[v]$ is monotone non-increasing.

Lemma 5. After pass i , $d[v] \leq$ length of a shortest $v \rightsquigarrow t$ path using $\leq i$ edges.

Pf. [by induction on i]

- Base case: $i = 0$.
- Assume true after pass i .
- Let P be any $v \rightsquigarrow t$ path with $\leq i + 1$ edges.
- Let (v, w) be first edge in P and let P' be subpath from w to t .
- By inductive hypothesis, at the end of pass i , $d[w] \leq \ell(P')$ because P' is a $w \rightsquigarrow t$ path with $\leq i$ edges.
- After considering edge (v, w) in pass $i + 1$:

and by Lemma 4,
 $d[w]$ does not increase

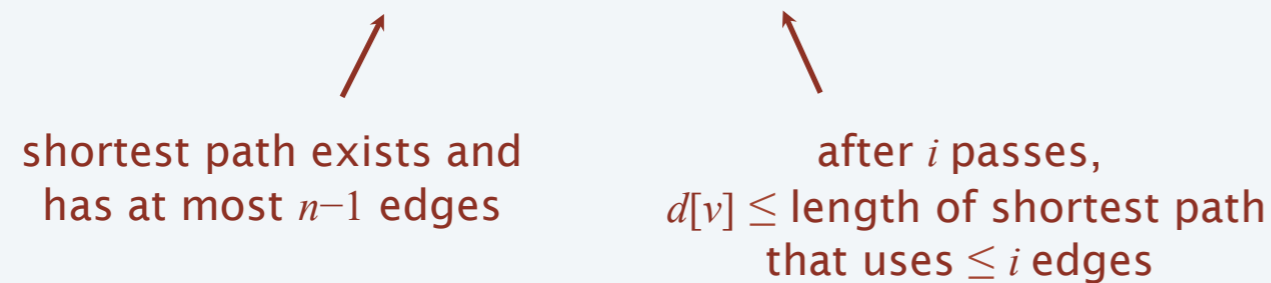
$$\begin{aligned} d[v] &\leq \ell_{vw} + d[w] \\ &\leq \ell_{vw} + \ell(P') \\ &= \ell(P) \quad \blacksquare \end{aligned}$$

and by Lemma 4,
 $d[v]$ does not increase

Bellman–Ford–Moore: analysis

Theorem 2. Assuming no negative cycles, Bellman–Ford–Moore computes the lengths of the shortest $v \rightsquigarrow t$ paths in $O(mn)$ time and $\Theta(n)$ extra space.

Pf. Lemma 2 + Lemma 5. ▀



Remark. Bellman–Ford–Moore is typically faster in practice.

- Edge (v, w) considered in pass $i + 1$ only if $d[w]$ updated in pass i .
- If shortest path has k edges, then algorithm finds it after $\leq k$ passes.

what about the shortest paths?

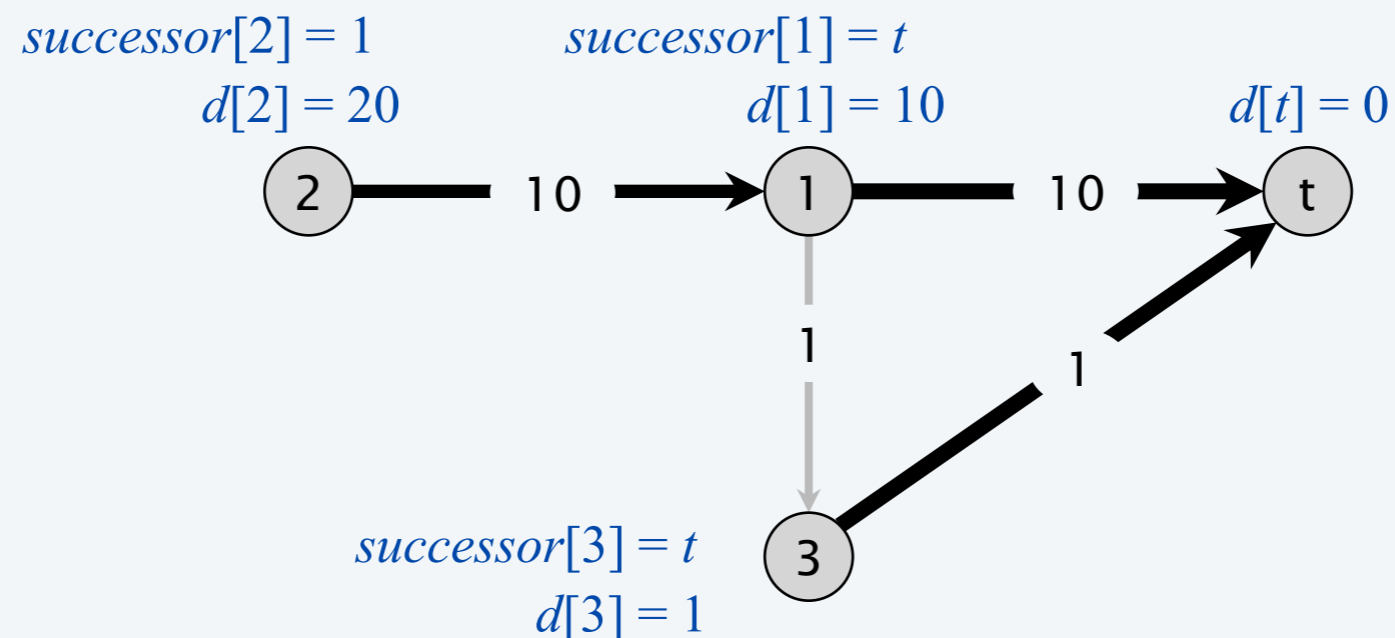
Bellman–Ford–Moore: analysis

Claim. ~~Throughout Bellman–Ford–Moore, following the $successor[v]$ pointers gives a directed path from v to t of length $d[v]$.~~

Counterexample. Claim is false!

- Length of successor $v \rightsquigarrow t$ path may be strictly shorter than $d[v]$.

consider nodes in order: $t, 1, 2, 3$



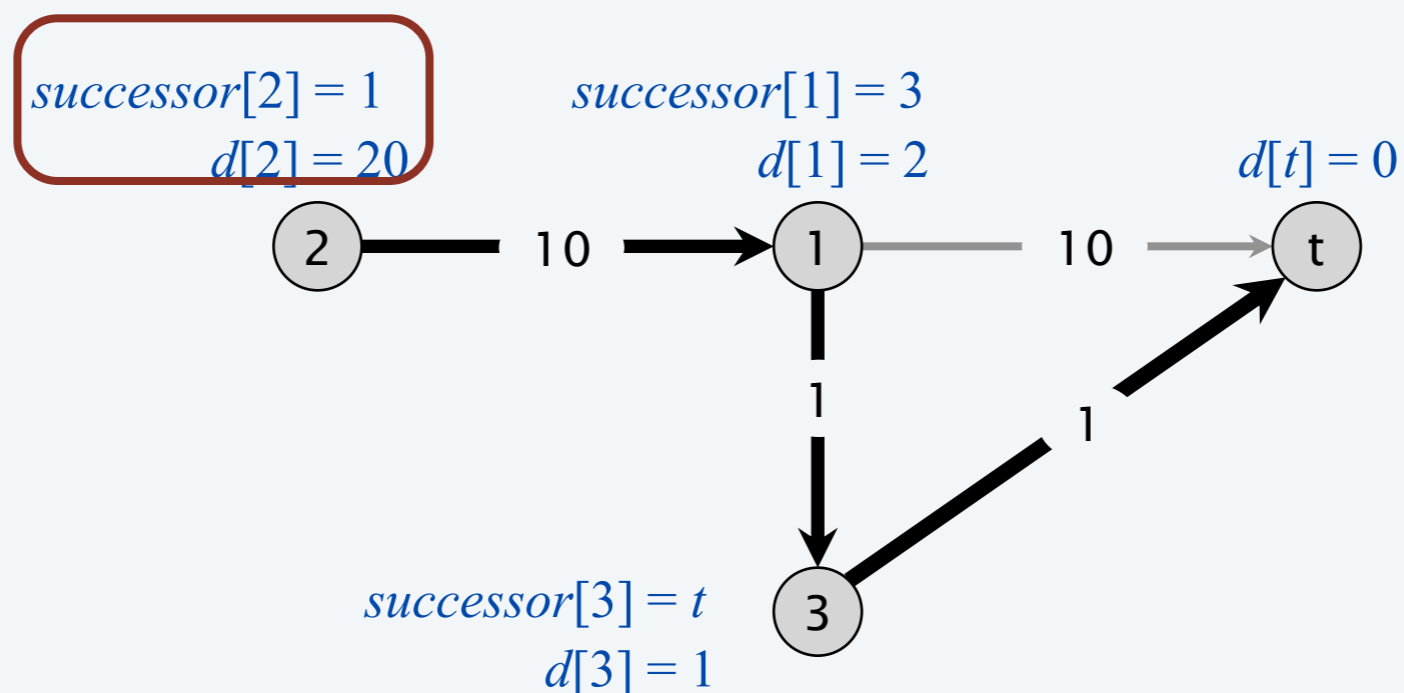
Bellman–Ford–Moore: analysis

Claim. ~~Throughout Bellman–Ford–Moore, following the $successor[v]$ pointers gives a directed path from v to t of length $d[v]$.~~

Counterexample. Claim is false!

- Length of successor $v \rightsquigarrow t$ path may be strictly shorter than $d[v]$.

consider nodes in order: $t, 1, 2, 3$



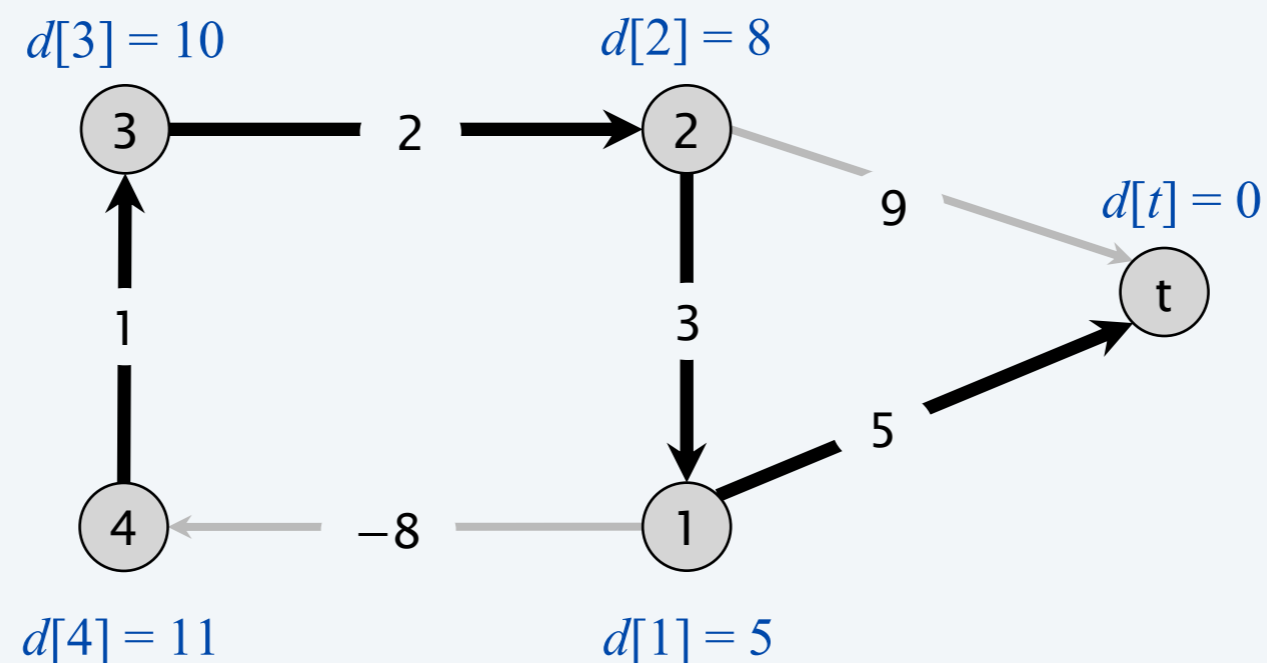
Bellman–Ford–Moore: analysis

Claim. ~~Throughout Bellman–Ford–Moore, following the $successor[v]$ pointers gives a directed path from v to t of length $d[v]$.~~

Counterexample. Claim is false!

- Length of successor $v \rightsquigarrow t$ path may be strictly shorter than $d[v]$.
- If negative cycle, successor graph may have directed cycles.

consider nodes in order: $t, 1, 2, 3, 4$



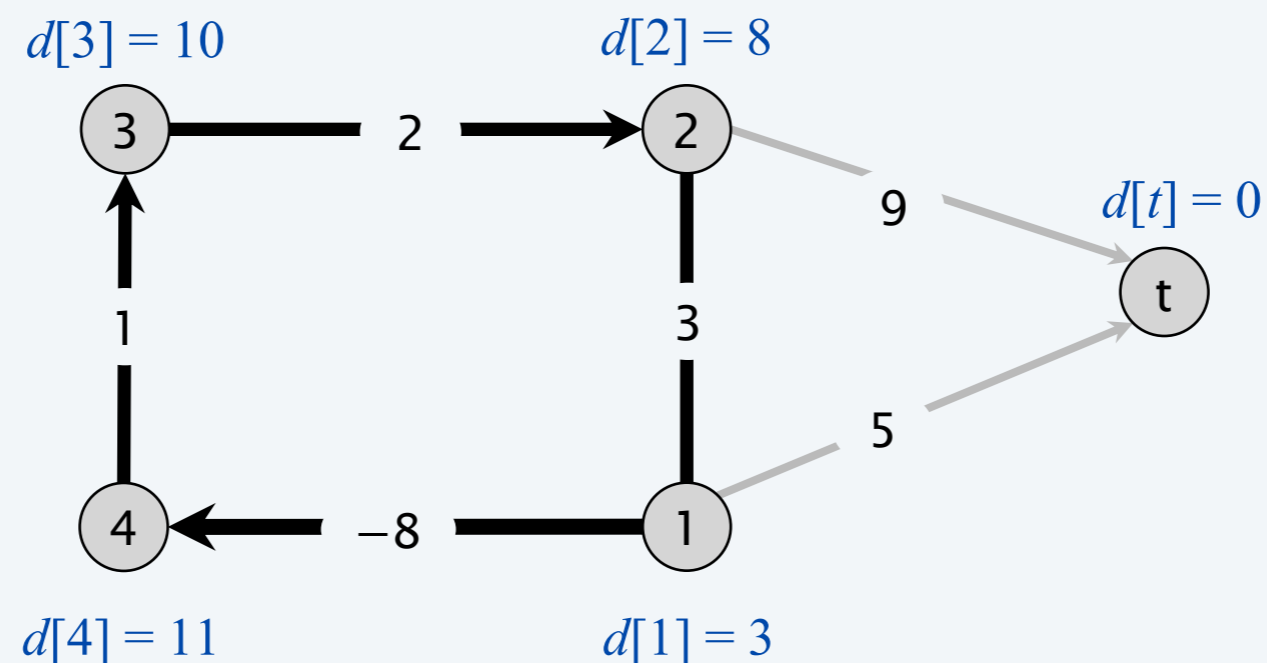
Bellman–Ford–Moore: analysis

Claim. ~~Throughout Bellman–Ford–Moore, following the $successor[v]$ pointers gives a directed path from v to t of length $d[v]$.~~

Counterexample. Claim is false!

- Length of successor $v \rightsquigarrow t$ path may be strictly shorter than $d[v]$.
- If negative cycle, successor graph may have directed cycles.

consider nodes in order: $t, 1, 2, 3, 4$



Bellman–Ford–Moore: finding the shortest paths


Lemma 6. Any directed cycle W in the successor graph is a negative cycle.

Pf.

- If $successor[v] = w$, we must have $d[v] \geq d[w] + \ell_{vw}$.
(LHS and RHS are equal when $successor[v]$ is set; $d[w]$ can only decrease; $d[v]$ decreases only when $successor[v]$ is reset)
- Let $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k \rightarrow v_1$ be the sequence of nodes in a directed cycle W .
- Assume that (v_k, v_1) is the last edge in W added to the successor graph.

- Just prior to that:
$$\begin{array}{rcl} d[v_1] & \geq & d[v_2] + \ell(v_1, v_2) \\ d[v_2] & \geq & d[v_3] + \ell(v_2, v_3) \\ \vdots & & \vdots \\ d[v_{k-1}] & \geq & d[v_k] + \ell(v_{k-1}, v_k) \\ d[v_k] & > & d[v_1] + \ell(v_k, v_1) \end{array}$$

← holds with strict inequality since we are updating $d[v_k]$

- Adding inequalities yields $\ell(v_1, v_2) + \ell(v_2, v_3) + \dots + \ell(v_{k-1}, v_k) + \ell(v_k, v_1) < 0$.


W is a negative cycle

what if the graph has a negative cycle (the can reach t)?

BELLMAN–FORD–MOORE(V, E, ℓ, t)

FOREACH node $v \in V$:

$d[v] \leftarrow \infty$.

$successor[v] \leftarrow null$.

$d[t] \leftarrow 0$.

FOR $i = 1$ TO $n - 1$

FOREACH node $w \in V$:

IF ($d[w]$ was updated in previous pass)

FOREACH edge $(v, w) \in E$:

IF ($d[v] > d[w] + \ell_{vw}$)

$d[v] \leftarrow d[w] + \ell_{vw}$.

$successor[v] \leftarrow w$.

IF (no $d[\cdot]$ value changed in pass i) STOP.

Bellman–Ford–Moore: checking for negative cycle

BELLMAN–FORD–MOORE(V, E, ℓ, t)

FOREACH node $v \in V$:

$d[v] \leftarrow \infty$.

$successor[v] \leftarrow null$.

$d[t] \leftarrow 0$.

FOR $i = 1$ TO $n - 1$

FOREACH node $w \in V$:

IF ($d[w]$ was updated in previous pass)

FOREACH edge $(v, w) \in E$:

IF ($d[v] > d[w] + \ell_{vw}$)

$d[v] \leftarrow d[w] + \ell_{vw}$.

$successor[v] \leftarrow w$.

IF (no $d[\cdot]$ value changed in pass i) STOP.

FOREACH edge $(v, w) \in E$

IF ($d[v] > d[w] + \ell_{vw}$) THEN return “there is a negative cycle”

pass n
 $O(m)$ time

Lemma 6. If there is a negative cycle (that can reach t) the (modified) algorithm report it.

Pf.

- If there is no negative cycle the pass $\#n$ do nothing
- Let $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k \rightarrow v_1$ a directed negative cycle W .
- assume by contradiction that the algorithm does not return it
- Then: condition of the last **IF** is always false.

- Hence:
$$\begin{array}{rcl} d[v_1] & \leq & d[v_2] + \ell(v_1, v_2) \\ d[v_2] & \leq & d[v_3] + \ell(v_2, v_3) \\ \vdots & & \vdots \\ d[v_{k-1}] & \leq & d[v_k] + \ell(v_{k-1}, v_k) \\ d[v_k] & \leq & d[v_1] + \ell(v_k, v_1) \end{array}$$

- Adding inequalities yields $\ell(v_1, v_2) + \ell(v_2, v_3) + \dots + \ell(v_{k-1}, v_k) + \ell(v_k, v_1) \geq 0$. ▪



W is cannot be a negative cycle: a contradiction