Algorithm II

6. Dynamic Programming II

WU Xiaokun 吴晓堃

xkun.wu [at] gmail

Sequence alignment

String similarity

Q. How similar are two strings?

Ex. ocurrance and occurrence.

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

Edit distance

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

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

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

- cost = $\delta + \alpha_{CG} + \alpha_{TA}$.

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

Sequence alignment: cost

Goal. Given two strings $x_1 x_2 ... x_m$ and $y_1 y_2 ... y_n$, 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:

- $cost(M) = \sum_{(x_i, y_j) \in M} \alpha_{x_i y_j} + \sum_{i : x_i \, unmatched} \delta + \sum_{j : y_j \, unmatched} \delta$

Sequence alignment: problem structure

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

Goal. $OPT(m, n)$.

Case 1. $OPT(i, j)$ matches $x_i – y_i$.

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

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

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

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

- Pay gap for y_i + min cost of aligning $x_1 x_2 \ldots x_i$ and $y_1 y_2 \ldots y_{i-1}$.

≡

Sequence alignment: Bellman equation

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

Goal. $OPT(m, n)$.

Case 1. $OPT(i, j)$ matches $x_i – y_i$.

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

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

Bellman equation.

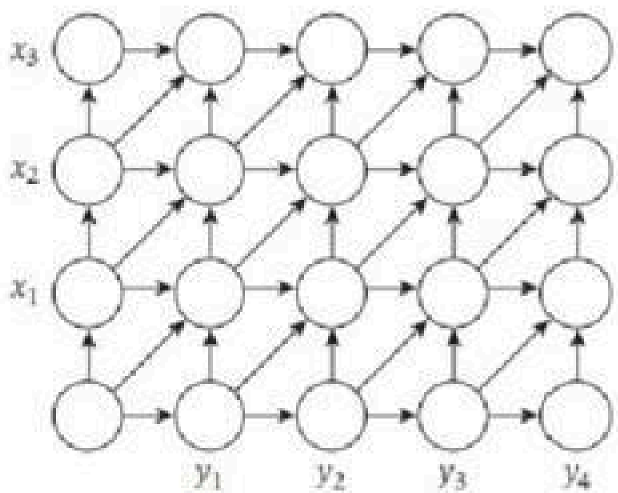
$$OPT(i,j) = \begin{cases} j\delta \\ i\delta \\ \min\{\alpha_{x_i,y_j} + OPT(i-1, j-1), \delta + OPT(i-1, j), \delta + OPT(i, j-1)\} \end{cases}$$

≡

Sequence alignment: Algorithm

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

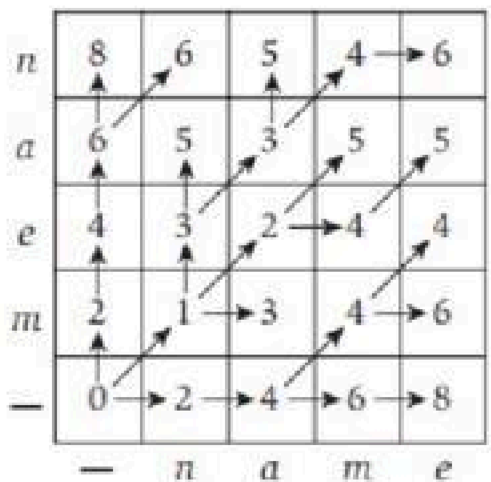
1. FOR $i = 0..m$: $M[i, 0] = i\delta$;
2. FOR $j = 0..n$: $M[0, j] = j\delta$;
3. FOR $i = 1..m$:
   1. FOR $j = 1..n$:
      1. $M[i, j] = \min\{\alpha x_i y_j + M[i-1, j-1], \delta + M[i-1, j], \delta + M[i, j-1]\}$;
4. RETURN $M[m, n]$;



Sequence alignment: trace-back

Ex. Matching *mean* and *name*, with:

- $\delta = 2$,
- mismatch vowels or consonants cost 1.
- matching vowel and consonant cost 3.



m	e	a	n	-
n	-	a	m	e
*	-		*	-
1	2	0	1	2

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.

Correctness.

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

Time. M has mn entries.

Hirschberg's algorithm

Sequence alignment in linear space

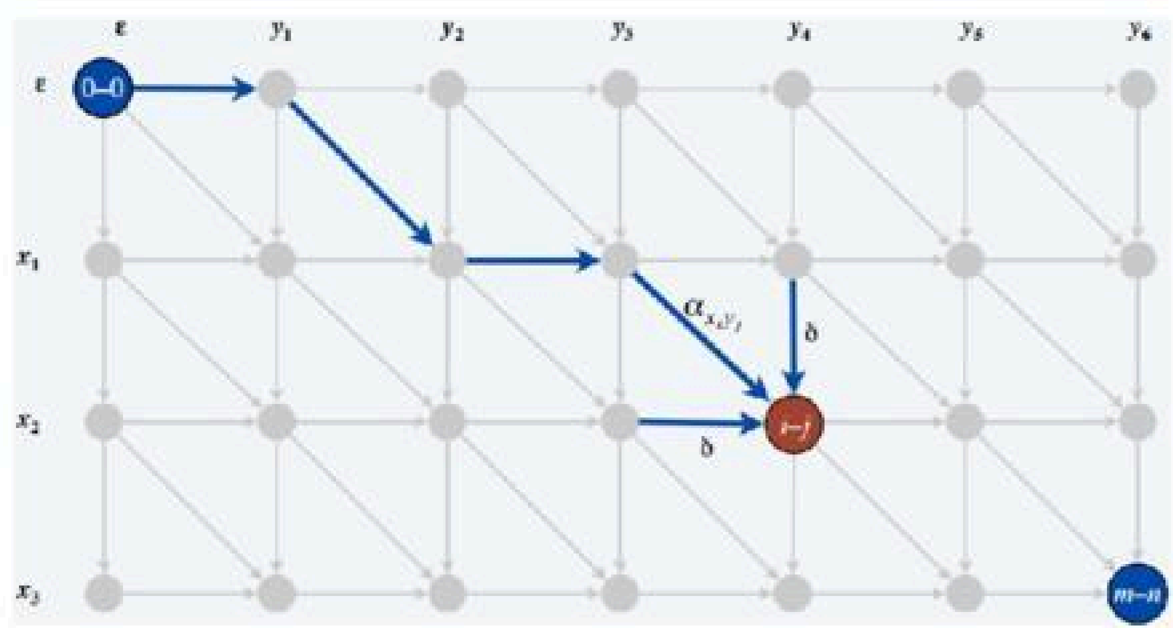
Theorem. [Hirschberg] There exists an algorithm to find an optimal alignment in $O(mn)$ time and $O(m + n)$ space.

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

Hirschberg's algorithm 1

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 1.1

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 $i + j$]

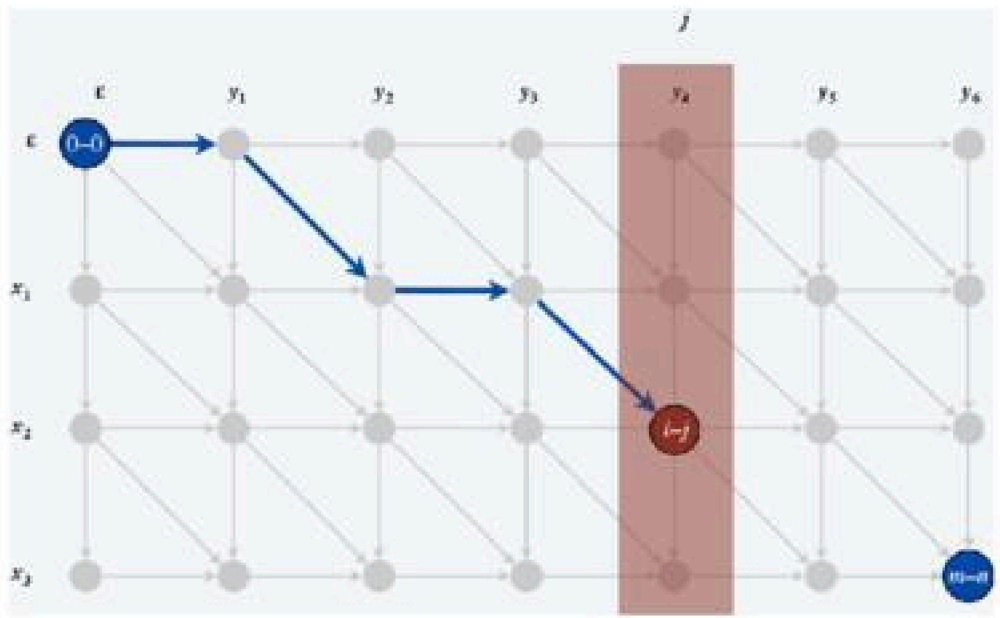
- Base case: $f(0, 0) = OPT(0, 0) = 0$.
- Inductive hypothesis: assume true for all (i', j') with $i' + j' < i + j$.
- Last edge on shortest path to (i, j) is from $(i–1, j–1)$, $(i–1, j)$, or $(i, j–1)$.
- Thus,

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

Hirschberg's algorithm 1.2

Edit distance graph.

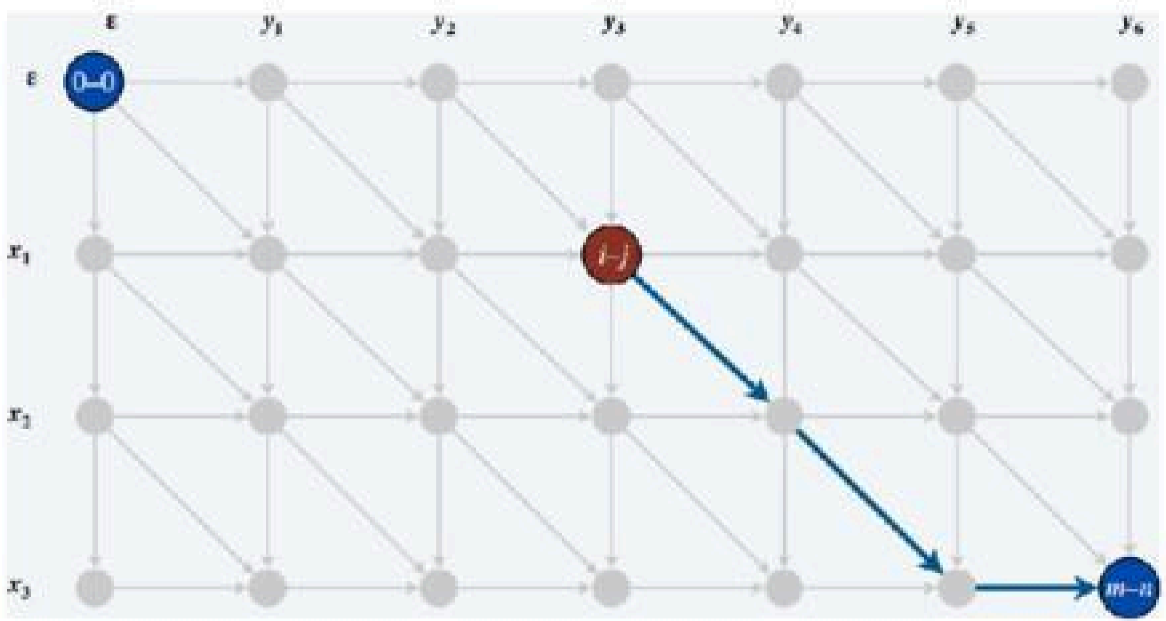
- Let $f(i, j)$ denote length of shortest path from $(0, 0)$ to (i, j).
- **Lemma**: $f(i, j) = OPT(i, j)$ for all i and j.
- Can compute $f(\cdot, j)$ for any j in $O(mn)$ time and $O(m + n)$ space.



Hirschberg's algorithm 2

Edit distance graph.

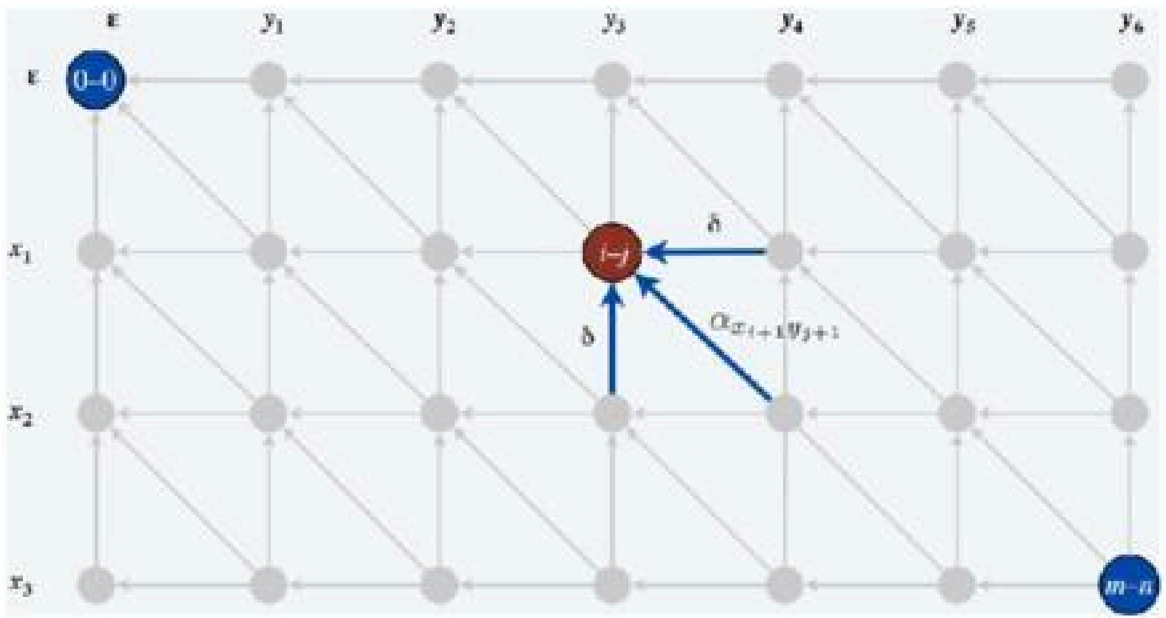
- Let $g(i, j)$ denote length of shortest path from (i, j) to (m, n).



Hirschberg's algorithm 2.1

Edit distance graph.

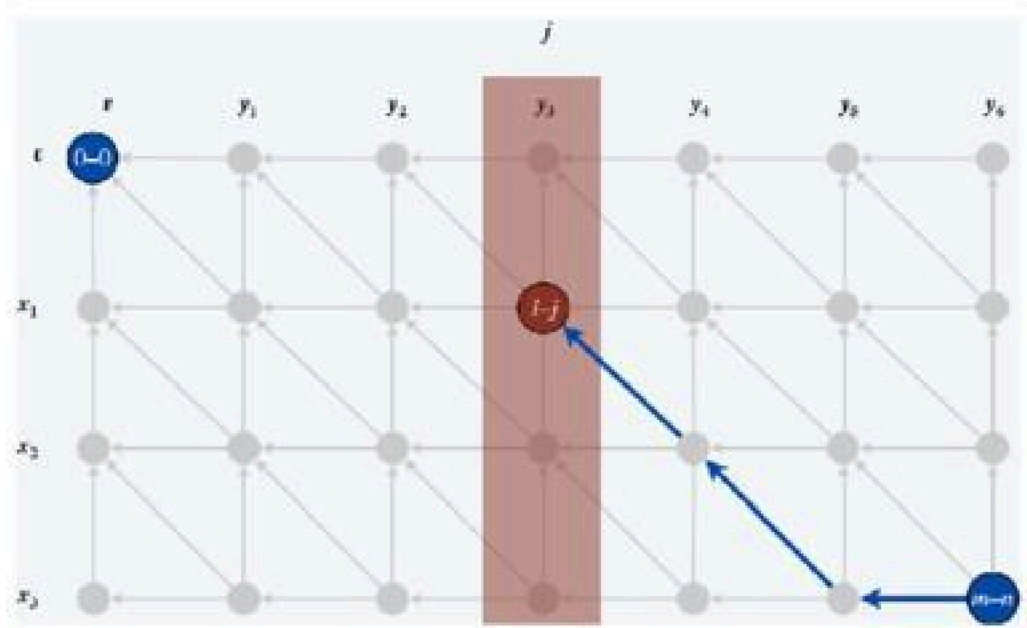
- Let $g(i, j)$ denote length of shortest path from (i, j) to (m, n).
- Can compute $g(i, j)$ by reversing the edge orientations and inverting the roles of $(0, 0)$ and (m, n).



Hirschberg's algorithm 2.2

Edit distance graph.

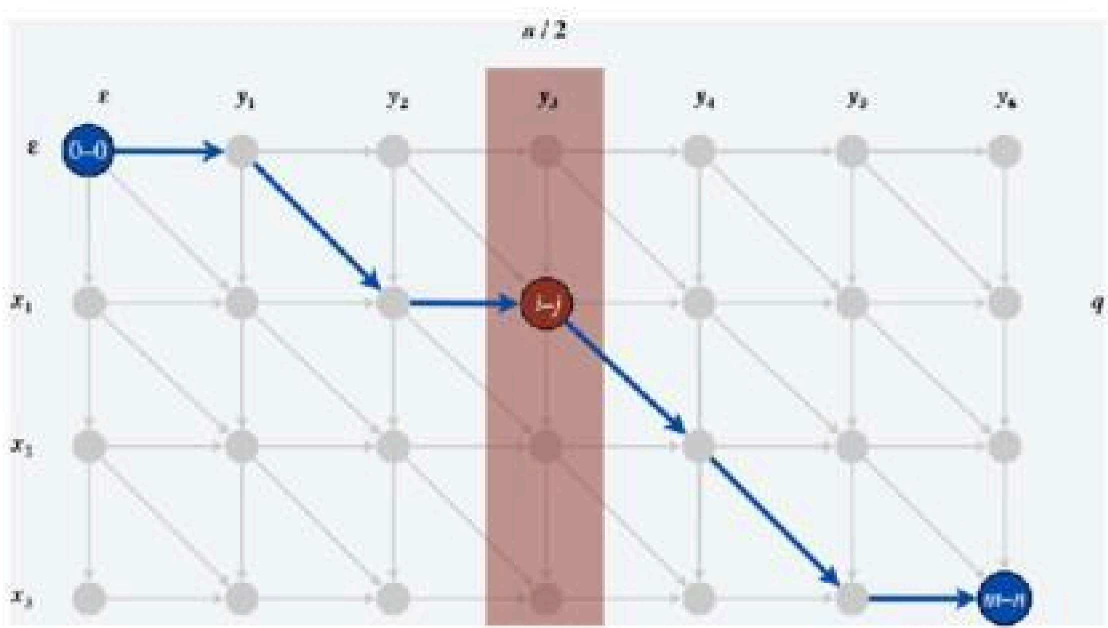
- Let $g(i, j)$ denote length of shortest path from (i, j) to (m, n).
- Can compute $g(\cdot, j)$ for any j in $O(mn)$ time and $O(m + n)$ space.



Hirschberg's algorithm 3

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

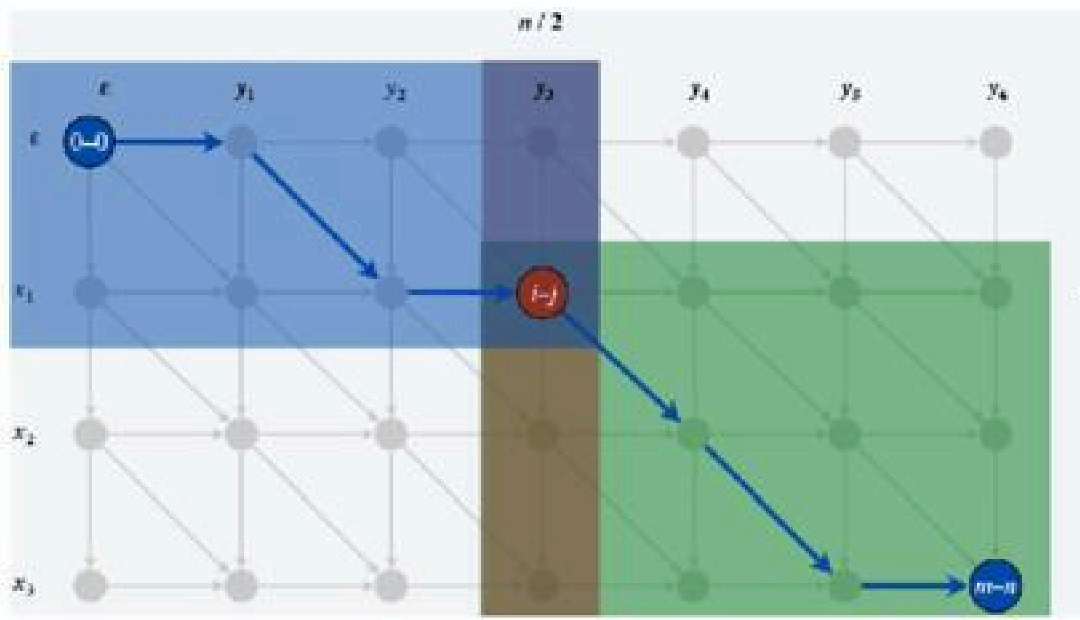
Observation 2. let q be an index that minimizes $f(q, n/2) + g(q, n/2)$. Then, there exists a shortest path from $(0, 0)$ to (m, n) that uses $(q, n/2)$.



Hirschberg's algorithm 4

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: 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: time analysis warmup

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

Pf.

- $T(m, n)$ is monotone non-decreasing in both m and n.
- $T(m, n) \leq 2T(m, n/2) + O(mn)$
  - $\Rightarrow T(m, n) = O(mn \log n)$.

Remark. Analysis is not tight because two sub-problems are of size $(q, n/2)$ and $(m-q, n/2)$. Next, we prove $T(m, n) = O(mn)$.

Hirschberg's: 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.
- $T(q, n/2) + T(m\text{–}q, n/2)$ time for two recursive calls.
- For some constant c:

$$T(m, n) \leq cmn + T(q, n/2) + T(m - q, n/2)$$
$$T(m, 2) \leq cm$$
$$T(2, n) \leq cn$$

Claim. $T(m, n) \leq 2cmn$.

Hirschberg's: time analysis (cont.)

$$T(m,n) \le cmn + T(q, n/2) + T(m-q, n/2)$$
$$T(m,2) \le cm$$
$$T(2,n) \le cn$$

Claim. $T(m,n) \le 2cmn$.
Pf. [by strong induction on $m+n$]

- Base cases: $m = 2$ and $n = 2$.

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

≡

Longest Common Subsequence

Problem. Given two strings $x_1 x_2 ... x_m$ and $y_1 y_2 ... y_n$, find a common subsequence that is as long as possible.

Alternative viewpoint. Delete some characters from x; delete some character from y; a common subsequence if it results in the same string.

Ex. LCS(GGCACCACG, ACGGCGGATACG) = GGCAACG.

Applications. Unix diff, git, bioinformatics.

Longest Common Subsequence: DP

Def. $OPT(i, j)$ = length of LCS of prefix strings $x_1 x_2 ... x_i$ and $y_1 y_2 ... y_j$.

Goal. $OPT(m, n)$.

Case 1. $x_i = y_j$.

- 1 + length of LCS of $x_1 x_2 ... x_{i-1}$ and $y_1 y_2 ... y_{j-1}$.

Case 2. $x_i \neq y_j$.

- Delete x_i: length of LCS of $x_1 x_2 ... x_{i-1}$ and $y_1 y_2 ... y_j$.
- Delete y_j: length of LCS of $x_1 x_2 ... x_i$ and $y_1 y_2 ... y_{j-1}$.

Bellman equation.

$$OPT(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ 1 + OPT(i-1, j-1) & \text{if } x_i = y_j \\ \max\{OPT(i-1, j), OPT(i, j-1)\} & \text{if } x_i \neq y_j \end{cases}$$

Longest Common Subsequence: DP II

Solution 2. Reduce to finding a min-cost alignment of x and y with

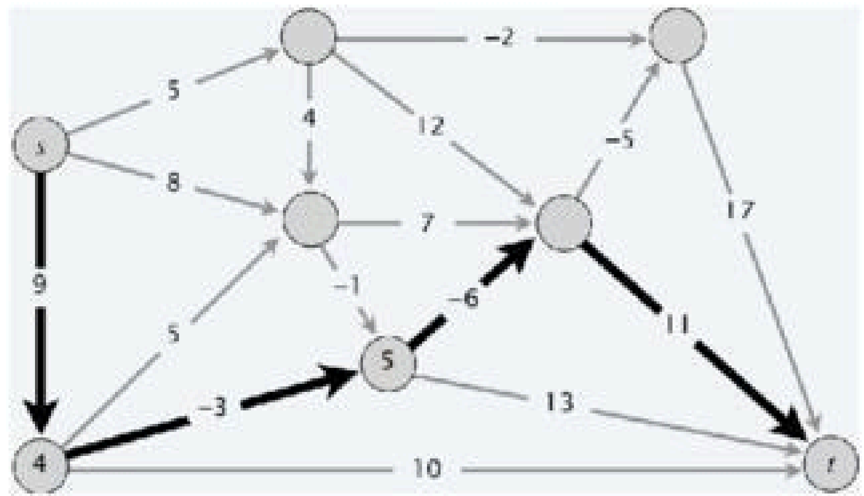
- Gap penalty $\delta = 1$
- Mismatch penalty α
  - $= 0, \text{if } p = q$
  - $= \infty, \text{if } p \neq q$
- Edit distance = # gaps = number of characters deleted from x and y.
- Length of LCS = ($m + n$ – edit distance) / 2.

Analysis. $O(mn)$ time and $O(m + n)$ space.

Bellman–Ford–Moore algorithm

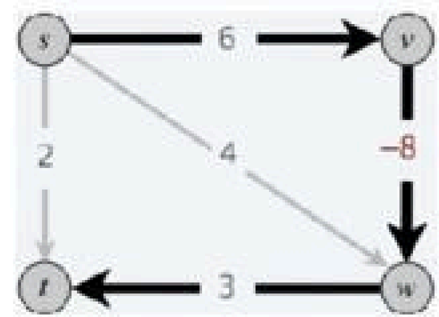
Shortest paths with negative weights

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.



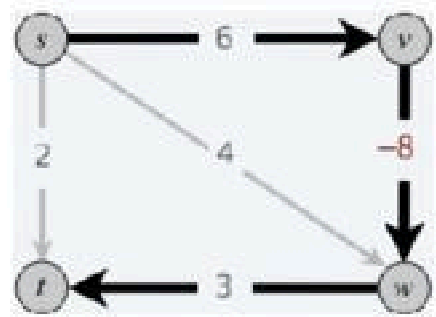
Greedy attempt

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

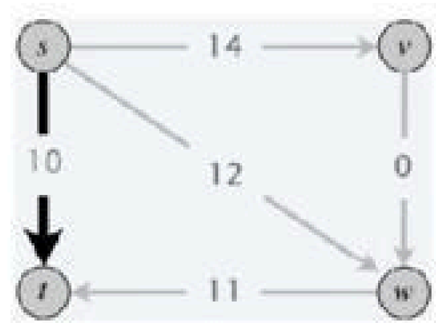


Greedy attempt

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

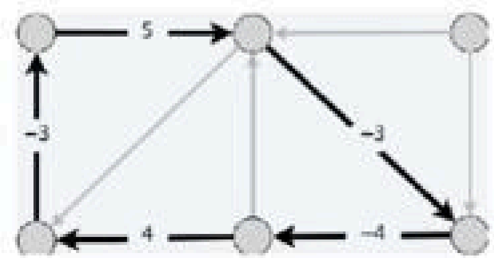


Re-weighting. Adding a constant to every edge length does not necessarily make Dijkstra's algorithm produce shortest paths.



Negative cycles

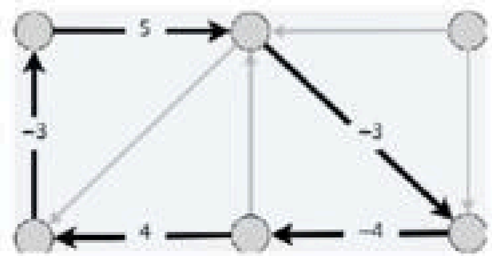
Def. A negative cycle is a directed cycle for which the sum of its edge lengths is negative.



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.

Negative cycles (cont.)

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



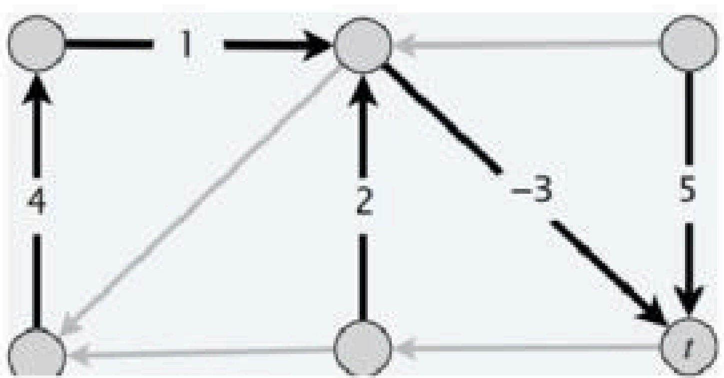
Lemma 2. If G has no negative cycles, then there exists a shortest $v \leadsto t$ path that is simple (and has n – 1 edges).
Pf.

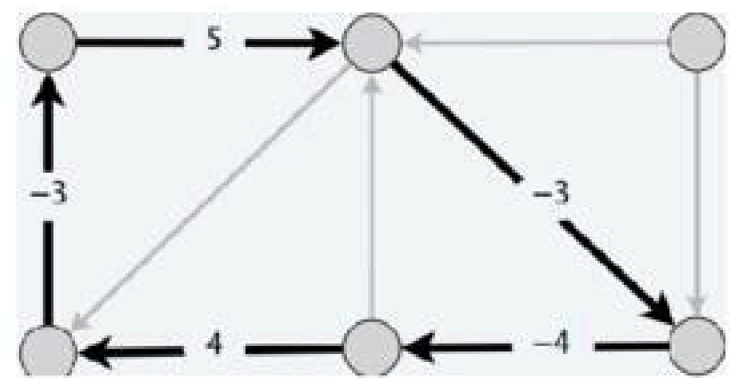
- Among all shortest $v \leadsto t$ paths, consider path P that uses the fewest edges.
- If that path P contains a directed cycle W, can remove the portion of P corresponding to W without increasing its length.

Two 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 $v \leadsto t$ path for every node v.

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





Quiz: shortest-paths via DP

Which sub-problems to find shortest $v \rightsquigarrow t$ paths for every node v?

A. $OPT(i, v)$ = length of shortest $v \rightsquigarrow t$ path that uses *exactly* i edges.
B. $OPT(i, v)$ = length of shortest $v \rightsquigarrow t$ path that uses *at most* i edges.
C. Neither A nor B.

Quiz: shortest-paths via DP

Which sub-problems to find shortest $v \leadsto t$ paths for every node v?

A. $OPT(i, v)$ = length of shortest $v \leadsto t$ path that uses *exactly* i edges.
B. $OPT(i, v)$ = length of shortest $v \leadsto t$ path that uses *at most* i edges.
C. Neither A nor B.

A: cannot eliminate shorter paths, since adding a negative edge may greatly reduce length and cancel previous effort, thus reduce to brute-force

DP for shortest-paths

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, simple path has $\leq n-1$ edges.

DP for shortest-paths

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

Goal. $OPT(n\text{–}1, v)$ for each v.

- by Lemma 2, simple path has $\leq n\text{–}1$ edges.

Case 1. Shortest $v \leadsto t$ path uses $\leq i\text{–}1$ edges.

- $OPT(i, v)$ = $OPT(i\text{–}1, v)$.

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

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

DP for shortest-paths: Bellman

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

Goal. $OPT(n\text{–}1, v)$ for each v.

Case 1. Shortest $v \leadsto t$ path uses $\leq i\text{–}1$ edges.

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

Bellman equation.

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

DP for shortest-paths: algorithm

SHORTEST-PATHS(V, E, l, t)

1. FOREACH node $v \in V$: $M[0, v] = \infty$;
2. $M[0, t] = 0$;
3. FOR $i = 1..n{-}1$:
    1. FOREACH node $v \in V$:
        1. $M[i, v] = M[i{-}1, v]$;
        2. FOREACH edge $(v, w) \in E$: $M[i, v] = min\{M[i, v], M[i{-}1, w] + l_{vw}\}$;

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

DP for shortest-paths: analysis

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.

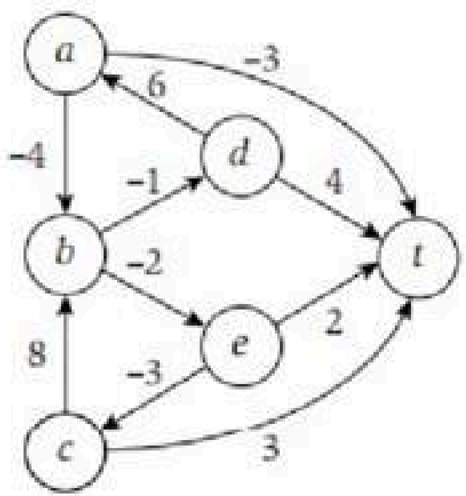
- Table requires $\Theta(n^2)$ space.
- Each iteration i takes $\Theta(m)$ time since we examine each edge once.

≡

DP for shortest-paths: trace-back

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 $M[i, v] = M[i{-}1, w] + l_{vw}$. Any directed path in this subgraph is a shortest path.



	0	1	2	3	4	5
t	0	0	0	0	0	0
a	∞	-3	-3	-4	-6	-6
b	∞	∞	0	-2	-2	-2
c	∞	3	3	3	3	3
d	∞	4	3	3	2	0
e	∞	2	0	0	0	0

Quiz: DP for shortest-paths

It is easy to modify the DP algorithm for shortest paths to ...

A. Compute lengths of shortest paths in $O(mn)$ time and $O(m+n)$ space.
B. Compute shortest paths in $O(mn)$ time and $O(m+n)$ space.
C. Both A and B.
D. Neither A nor B.

Shortest-paths: 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

BELLMAN–FORD–MOORE(V, E, c, t)

```
FOREACH node v:
   d[v] = INF;
   successor[v] = null;
d[t] = 0;
FOR i = 1 .. n – 1:
   FOREACH node w:
      IF (d[w] was updated in previous pass):
         FOREACH edge (v, w):
            IF (d[v] > d[w] + l vw) d[v] = d[w] + l vw;
```

≡

Quiz: Bellman–Ford–Moore

Which properties must hold after pass i of Bellman–Ford–Moore?

A. $d[v]$ = length of a shortest $v \leadsto t$ path using $\leq i$ edges.
B. $d[v]$ = length of a shortest $v \leadsto t$ path using exactly i edges.
C. Both A and B.
D. Neither A nor B.

Quiz: Bellman–Ford–Moore

Which properties must hold after pass i of Bellman–Ford–Moore?

A. $d[v]$ = length of a shortest $v \rightsquigarrow t$ path using $\leq i$ edges.
B. $d[v]$ = length of a shortest $v \rightsquigarrow t$ path using exactly i edges.
C. Both A and B.
D. Neither A nor B.

D. Now i is just a counter of $n - 1$ iterations.

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.

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, $d[w] \leq l(P')$, because P' is a $w \rightsquigarrow t$ path with $\leq i$ edges.
- After considering edge (v, w) in pass $i + 1$:

$$\begin{aligned} d[v] &\leq l_{vw} + d[w] \\ &\leq l_{vw} + l(P') \\ &= l(P) \end{aligned}$$

Bellman–Ford–Moore: analysis (cont.)

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.

- shortest path exists and has at most $n - 1$ edges
- after i passes, $d[v] \leq$ length of shortest path that uses $\leq i$ edges

Bellman–Ford–Moore: analysis (cont.)

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

Pf. Lemma 2 + Lemma 5.

- shortest path exists and has at most $n - 1$ edges
- after i passes, $d[v] \leq$ length of shortest path that uses $\leq i$ edges

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.

Quiz: Bellman–Ford–Moore trace-back

Assuming no negative cycles, which properties must hold throughout Bellman–Ford–Moore?

A. Following `successor`$[v]$ pointers gives a directed $v \rightsquigarrow t$ path.

B. If following `successor`$[v]$ pointers gives a directed $v \rightsquigarrow t$ path, *then* the length of that $v \rightsquigarrow t$ path is $d[v]$.

C. Both A and B.

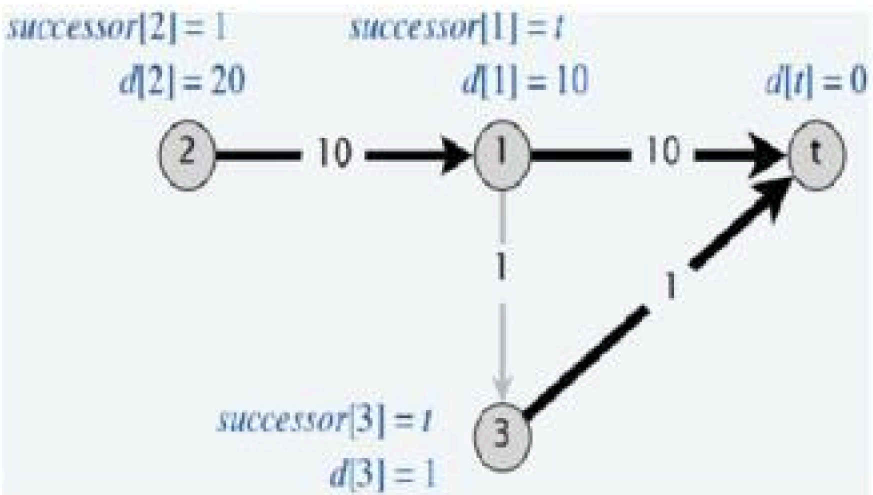
D. Neither A nor B.

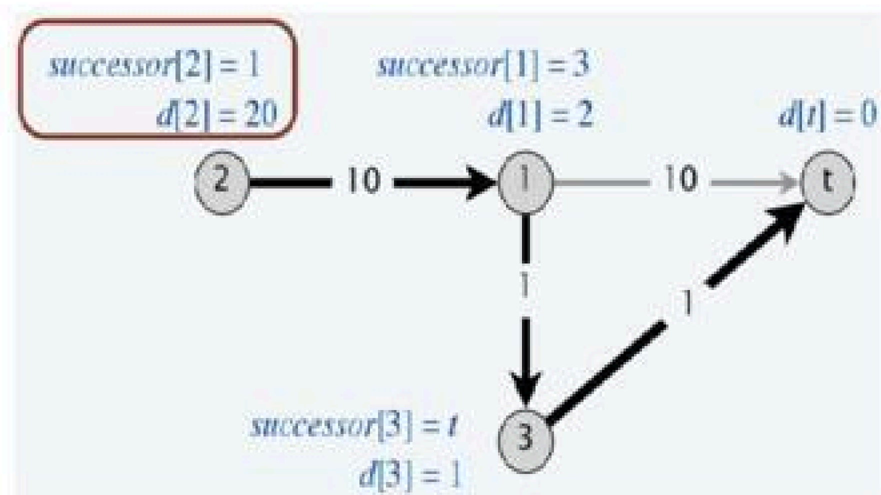
Bellman–Ford–Moore: trace-back

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 $d[v]$.



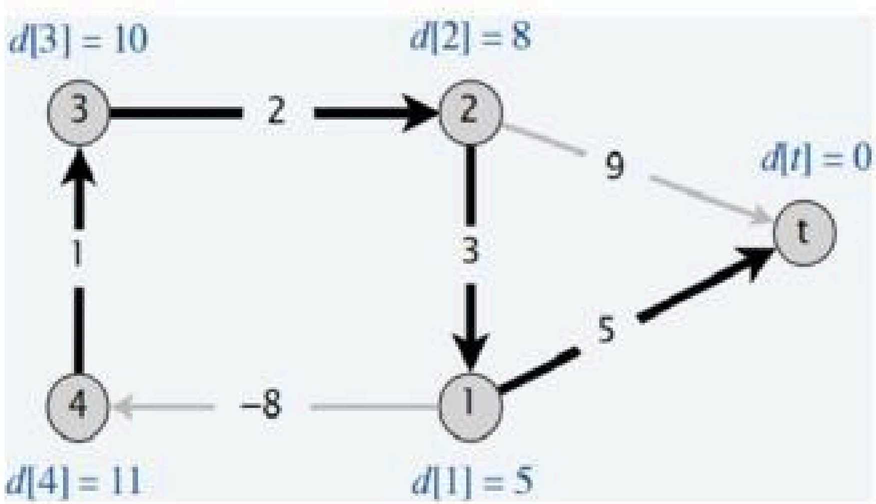


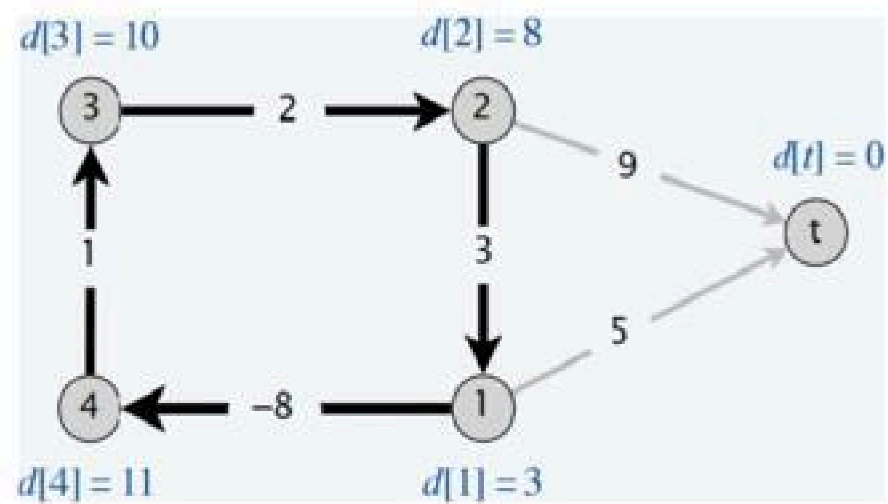
Bellman–Ford–Moore: trace-back

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 $d[v]$.
- With negative cycle, successor graph may have directed cycles.





Bellman–Ford–Moore: shortest paths

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

Pf.

- If `successor`$[v]$ = w, we must have $d[v] \geq d[w] + l_{vw}$.
- Let $v_1 \to v_2 \to \ldots \to v_k \to v_1$ be sequence in a directed cycle W.
- Assume that (v_k, v_1) is the last edge in W added to successor graph.
- Just prior to that:

$$
\begin{aligned}
d[v_1] &\geq l(v_1, v_2) + d[v_2] \\
&\vdots \\
d[v_{k\text{-}1}] &\geq l(v_{k\text{-}1}, v_k) + d[v_k] \\
d[v_k] &> l(v_k, v_1) + d[v_1] \text{ strict less: updating now}
\end{aligned}
$$

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

BFM: shortest paths (cont.)

Theorem 3. Assuming no negative cycles, Bellman–Ford–Moore finds shortest $v \rightsquigarrow t$ paths for every node v in $O(mn)$ time and $\Theta(n)$ extra space.

Pf.

- The successor graph cannot have a directed cycle. [Lemma 6]
- Let P: $v = v_1 \to v_2 \to \ldots \to v_k = t$ be following successor pointers.
- Upon termination, if `successor`[v] = w, we must have $d[v] = d[w] + l_{vw}$.

$$
\begin{aligned}
d[v_1] &= l(v_1, v_2) + d[v_2] \\
d[v_2] &= l(v_2, v_3) + d[v_3] \\
&\vdots \\
d[v_{k\text{-}1}] &= l(v_{k\text{-}1}, v_k) + d[v_k]
\end{aligned}
$$

- Adding equations yields $d[v] = d[t] + l(v_1, v_2) + l(v_2, v_3) + \ldots + l(v_{k\text{-}1}, v_k)$.

Distance-vector protocols

Communication network

Communication network.

- Node ≈ router.
- Edge ≈ direct communication link.
- Length of edge ≈ latency of link.

Dijkstra's algorithm. Requires global information of network.

Bellman–Ford–Moore. Uses only local knowledge of neighboring nodes.

Synchronization. We don't expect routers to run in lockstep.

- The order in which each edges are processed in Bellman–Ford–Moore is not important.
- Moreover, algorithm converges even if updates are asynchronous.

Distance-vector routing protocols

Distance-vector routing protocols. ["routing by rumor"]

- Each router maintains a vector of shortest-path lengths to every other node (**distances**) and the *first hop* on each path (**directions**).
- **Algorithm**: each router performs n separate computations, one for each potential destination node.

Ex. Routing Information Protocol (RIP), Xerox XNS RIP, Novell's IPX RIP, Cisco's IGRP, DEC's DNA Phase IV, AppleTalk's RTMP.

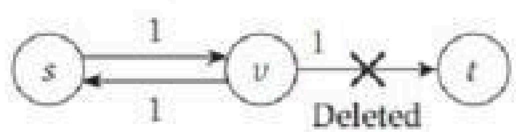
Distance-vector routing protocols

Distance-vector routing protocols. [“routing by rumor”]

- Each router maintains a vector of shortest-path lengths to every other node (**distances**) and the *first hop* on each path (**directions**).
- **Algorithm**: each router performs n separate computations, one for each potential destination node.

Ex. Routing Information Protocol (RIP), Xerox XNS RIP, Novell’s IPX RIP, Cisco’s IGRP, DEC’s DNA Phase IV, AppleTalk’s RTMP.

Caveat. Edge lengths may *change* during algorithm (or fail completely).



Path-vector routing protocols

Link-state routing protocols.

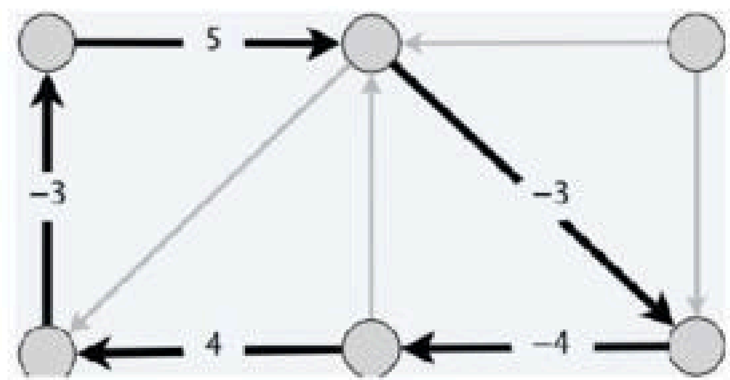
- Each router stores the *whole path* (or network topology).
- Based on Dijkstra's algorithm.
- Avoids "counting-to-infinity" problem and related difficulties.
- Requires significantly more storage.

Ex. Border Gateway Protocol (BGP), Open Shortest Path First (OSPF).

Negative cycles

Detecting negative cycles

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



Detecting negative cycles: application

Currency conversion. Given n currencies and exchange rates between pairs of currencies, is there an arbitrage opportunity?

Remark. Fastest algorithm very valuable!

Detecting negative cycles - I

Lemma 7. If $OPT(n, v) = OPT(n\text{–}1, v)$ for every node v, then no negative cycles.
Pf. The $OPT(n, v)$ values have converged $\Rightarrow$ shortest $v \rightsquigarrow t$ path exists.

Lemma 8. If $OPT(n, v) < OPT(n\text{–}1, v)$ for some node v, then (any) shortest $v \rightsquigarrow t$ path of length $\leq n$ contains a cycle W. Moreover W is a negative cycle.
Pf. [by contradiction]

- Since $OPT(n, v) < OPT(n\text{–}1, v)$, we know that shortest $v \rightsquigarrow t$ path P has exactly n edges.
- By *pigeonhole principle*, the path P must contain a repeated node x.
- Let W be any cycle in P.
- Deleting W yields a $v \rightsquigarrow t$ path with $< n$ edges $\Rightarrow$ W is a negative cycle.

Detecting negative cycles - II

Theorem 4. Can find a negative cycle in $\Theta(mn)$ time and $\Theta(n2)$ space.
Pf.

Construct **Augmented graph** G': Add new sink node t and connect all nodes to t with 0-length edge.

- G has a negative cycle iff G' has a negative cycle.

Case 1. [$OPT(n, v) = OPT(n–1, v)$ for every node v]

- By Lemma 7, no negative cycles.

Case 2. [$OPT(n, v) < OPT(n–1, v)$ for some node v]

- Using proof of Lemma 8, can extract negative cycle from $v \rightsquigarrow t$ path. (cycle cannot contain t since no edge leaves t)

≡

Detecting negative cycles - III

Theorem 5. Can find a negative cycle in $O(mn)$ time and $O(n)$ extra space.

Pf.

- Run `Bellman-Ford-Moore` on G' for $n' = n + 1$ passes (instead of n'–1).
- If no $d[v]$ values updated in pass n', then no negative cycles.
- Otherwise, suppose $d[s]$ updated in pass n'.
- Define `pass`(v) = last pass in which $d[v]$ was updated.
- Observe `pass`(s) = n' and `pass`(`successor`[v]) $\geq$ `pass`(v) –1 for each v.
- Following successor pointers, we must eventually repeat a node.
- Lemma 6 $\Rightarrow$ the corresponding cycle is a negative cycle.