Updated 4/16/2022

Topic 3: Efficient query evaluation
Unit 4: Optimization, Top-k, Ranked Enumeration
Lecture 24

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp22)

https://northeastern-datalab.github.io/cs7240/sp22/

4/15/2022

Pre-class conversations

- Recent topics: efficient algorithms need trees; what to do with problems that contain cycles?
- Project intermediate report: next week TUE, April 15
- 2nd iterations on scribes are optional

- Today
  - optimization problems, on trees and cycles
  - Dynamic Programming & friends

Topic 3: Efficient Query Evaluation & Factorized Representations

- **CONTINUED Lecture 16 (Fri 3/11):** T3-U1 Acyclic Queries
- Spring break
- **Lecture 17 (Tue 3/22):** T3-U1 Acyclic Queries
- **Lecture 18 (Fri 3/25):** T3-U1 Acyclic Queries
- **Lecture 19 (Tue 3/29):** T3-U1 Acyclic Queries / T3-U2 Cyclic Queries
- **Lecture 20 (Fri 4/1):** T3-U2 Cyclic Queries
- **Lecture 21 (Tue 4/5):** T3-U2 Cyclic Queries
- **Lecture 22 (Fri 4/8):** Factorized Representations / Top-k & Optimization Problems
- **Lecture 23 (Tue 4/12):** Top-k & Optimization Problems

Pointers to relevant concepts & supplementary material:

- **Unit 1. Acyclic Queries**: query hypergraph, Yannakakis algorithm, GYO reduction, dynamic programming, algebraic semirings, [Alice] Ch6.4, [Koutris'19] L4, enumeration, ranked enumeration:[Tziavelis+'20]
- **Unit 2. Cyclic Queries**: tree & hypertree decomposition, query widths, fractional hypertree width, AGM bound, worst-case optimal join algorithms, optimal algorithms, submodular width and multiple decompositions: [AGM'13], [NPRR'18], [KNR'17], [KNS'17]
- **Unit 3. Factorized Representations**: normalization, factorized databases [Olteanu, Schleich'16]
- **Unit 4. Top-k & Optimization Problems**: shortest paths, dynamic programming (DP), Yannakakis, semirings, rankings, top-k: [Roughgarden'10], [Ilyas+08], [Rahul, Tao'19], ranked enumeration [Tziavelis+'19]

Topic 4: Normalization, Information Theory & Axioms for Uncertainty

- **Lecture 24 (Fri 4/15):** Normal Forms & Information Theory
- **Lecture 25 (Tue 4/19):** Axioms for Uncertainty

Pointers to relevant concepts & supplementary material:

- **Unit 1. Normal Forms & Information Theory**: normal forms & their information-theoretic justification [Complete 18] Ch3, [Lee'87], [Arenas, Libkin'05]
- **Unit 2. Axioms for Uncertainty**: Uncertainty & Inconsistency, Maximum entropy principle [Cox'46], [Shannon'48], [Van Horn'03]

Topic 5: Linear Algebra & Iterative Graph Algorithms

- **Lecture 26 (Fri 4/22):** Graphs & Linear Algebra
- **Lecture 27 (Tue 4/26):** Graphs & Linear Algebra
- **Lecture 28 (Fri 4/29):** Computation Graphs

Pointers to relevant concepts & supplementary material:

- **Unit 1. Graphs & Linear Algebra**: graphs, linear algebra (LA), semirings, iterative algorithms, rankings on graphs, associative arrays: [Lay+ 21], [Kepner, Gilbert'11], [Kepner, Jananthan'18], [Klein'13], Random walks & PageRank axioms: [Newman'10], [Gleich'15], [AT'10], label propagation & back-propagation, semi-supervised learning [BDL'06], factorized graph representations: [KLG'20], belief propagation (BP): [Murphy'12], [GGKF'15]
- **Unit 2. Computation Graphs**: circuits, knowledge compilation [DM'02], [JS'13]

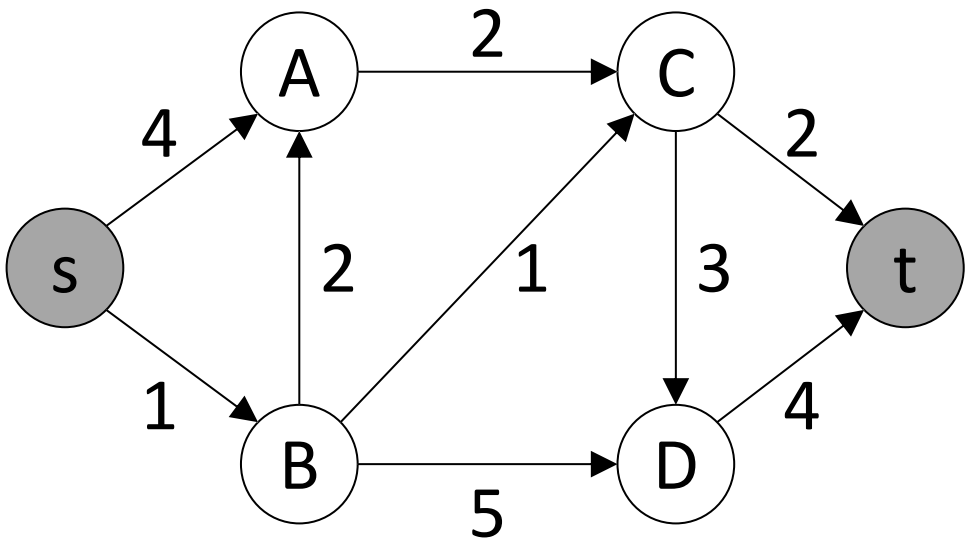
Outline: T3-4: Optimization, Top-k, Ranked Enumeration

- Dynamic Programming (DP)
  - Shortest path algorithms
  - DP & shortest path enumeration
  - Non-serial DP (NSDP) for solving SAT
  - Yannakakis and NSDP
  - Algebraic Structures (Semirings)
- Top-k
- Ranked Enumeration

Shortest-path algorithms on digraphs (directed graphs)

- Dijkstra
  - identical to uniform-cost-search (cost g from source)
  - only works for positive edges (can not handle negative edges)
- A*
  - variant of best-search Dijkstra (cost g from source + cost h to target)
  - visits even fewer nodes with good heuristics
- Bellman Ford
  - visits all nodes and edges (possibly multiple times)
  - more work, but negative weights are ok
  - becomes Dynamic Programming (DP) with topological sort (visits every node and edge exactly once)

Shortest-path algorithms



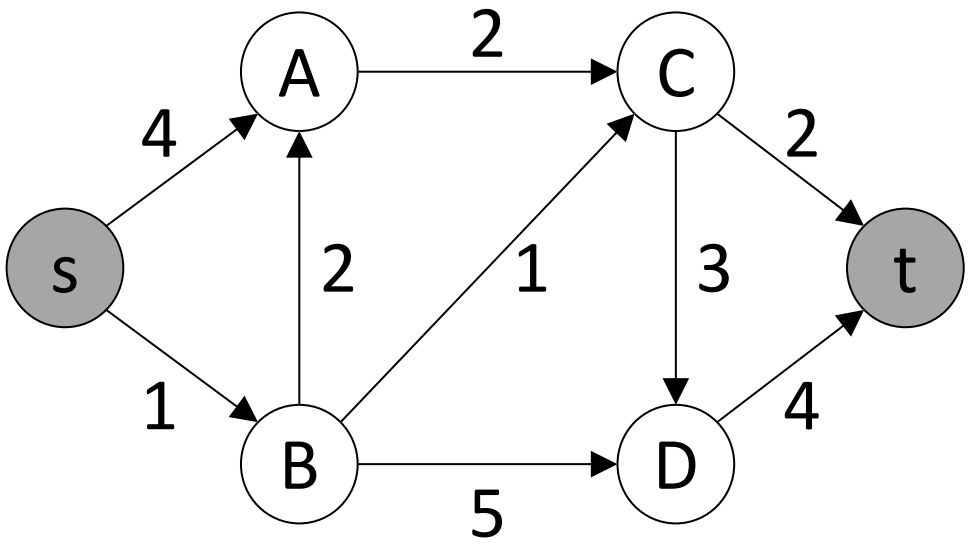
Our problem: find the shortest path from some source node s to some target node t in a directed graph (digraph).

We assume some cost on each edge. The shortest path is the one with minimum sum of costs.

Shortest-path algorithms: Dijkstra

$\min(x, \infty) = x \quad \forall x \in \mathbb{R}$

Currently best known value



vertex	distance from s (g)	previous vertex
s	0	
A	∞	
B	∞	
C	∞	
D	∞	
t	∞	

uses "relaxation": iteratively update approximate distance until correct solution is reached

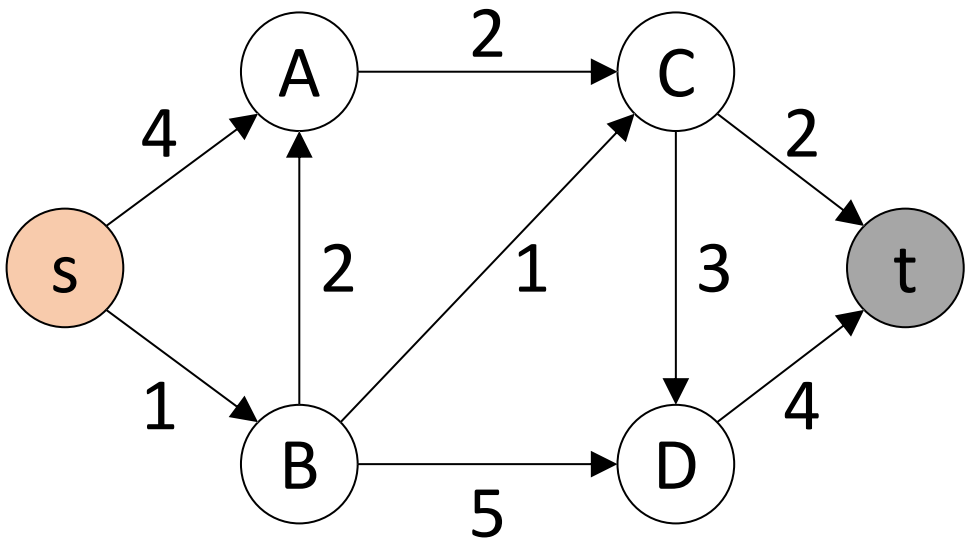
Open PQ = (s:0,A:∞, B:∞,C:∞,D:∞,t:∞) ← Priority Queue (PQ) sorted by distance g

Closed = {}

Dijkstra. "A note on two problems in connexion with graphs", Numerische Mathematik, 1959. https://doi.org/10.1007/BF01386390

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Shortest-path algorithms: Dijkstra



Sort of elements in Open

vertex	distance from s (g)	previous vertex
s	0	
A	∞	
B	∞	
C	∞	
D	∞	
t	∞	

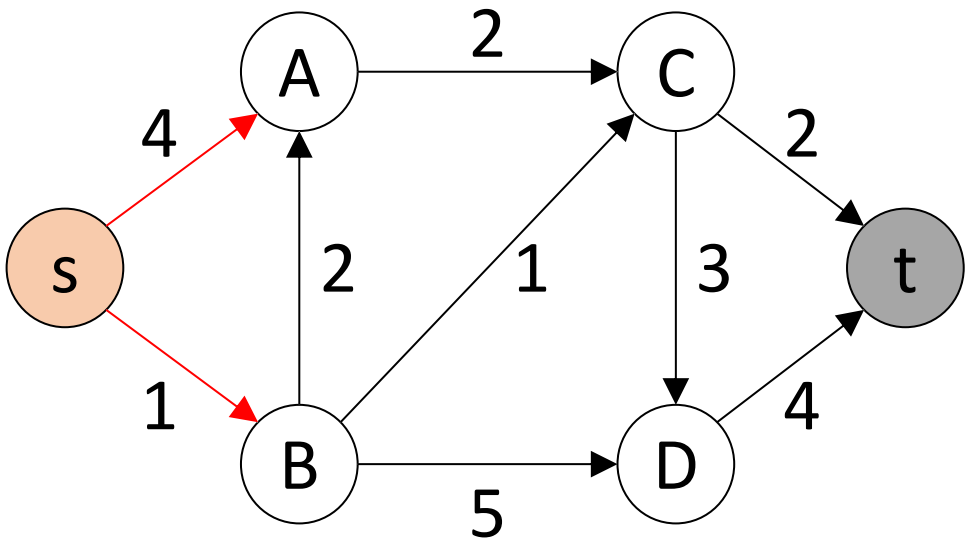
Current: s:0

Open PQ = (A:∞, B:∞,C:∞,D:∞,t:∞)

Closed = {}

Dijkstra. "A note on two problems in connexion with graphs", Numerische Mathematik, 1959. https://doi.org/10.1007/BF01386390
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Shortest-path algorithms: Dijkstra



vertex	distance from s (g)	previous vertex
s	0	
A	4	s
B	1	s
C	∞	
D	∞	
t	∞	

"decrease key"

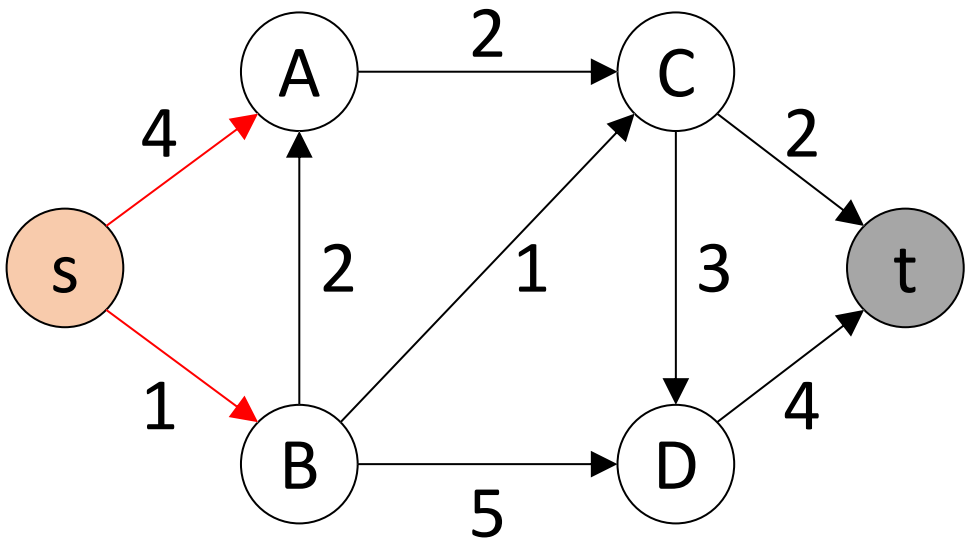
Current: s:0

Open PQ = (B:1,A:4,C:∞,D:∞,t:∞)

Closed = {}

Dijkstra. "A note on two problems in connexion with graphs", Numerische Mathematik, 1959. https://doi.org/10.1007/BF01386390
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Shortest-path algorithms: Uniform-cost-search



vertex	distance from s (g)	previous vertex
s	0	
A	4	s
B	1	s
C		
D		
t		

an uninformed search algorithm: use the lowest cumulative cost to find a path from s to t

Current: s:0

Open PQ = (B:1,A:4)

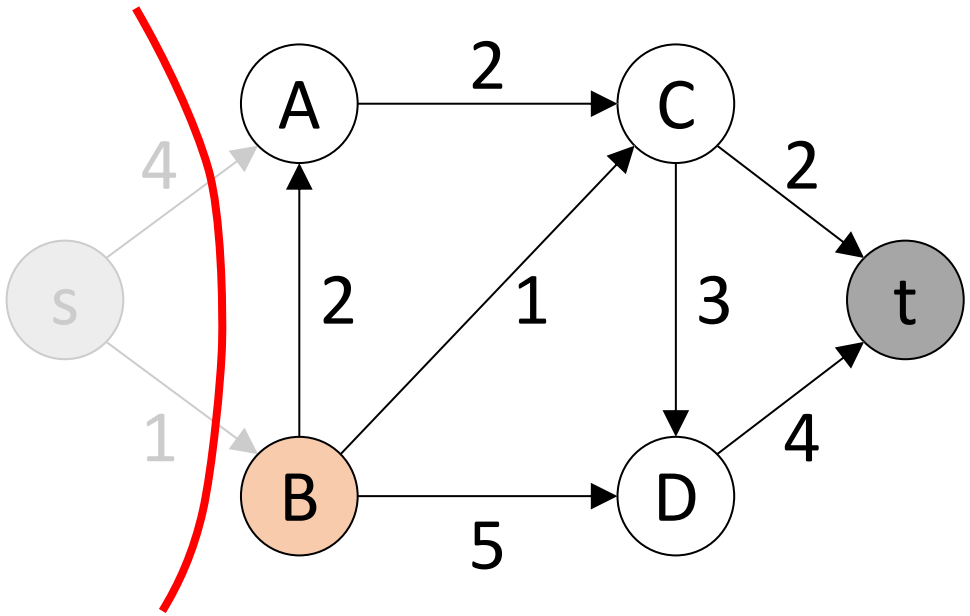
Closed = {}

Unknown = {C, D, t}

Minor improvement of "uniform-cost-search" over Dijkstra: initialize the PQ lazily only with discovered nodes (thus keep: Open, Closed, and Unknown). This results in a smaller PQ

Felner. "Position Paper: Dijkstra's Algorithm versus Uniform Cost Search or a Case Against Dijkstra's Algorithm", SoCS 2011. http://www.aaai.org/ocs/index.php/SOCS/SOCS11/paper/view/4017
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Shortest-path algorithms: Dijkstra



	vertex	distance from s (g)	previous vertex
✓	s	0	
	A	4	s
☞	B	1	s
	C	∞	
	D	∞	
	t	∞	

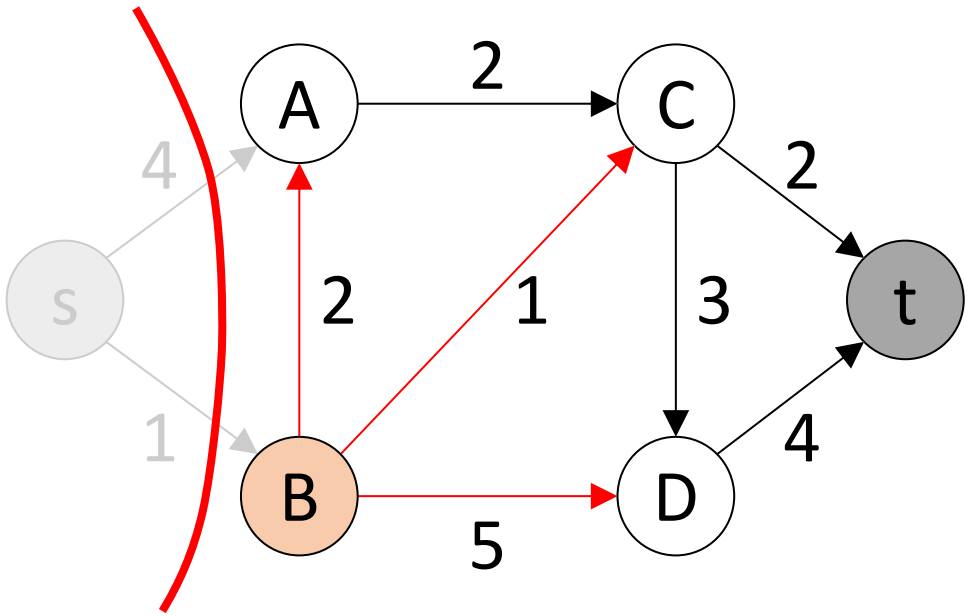
Current: B:1

Open PQ = (A:4,C:∞,D:∞,t:∞)

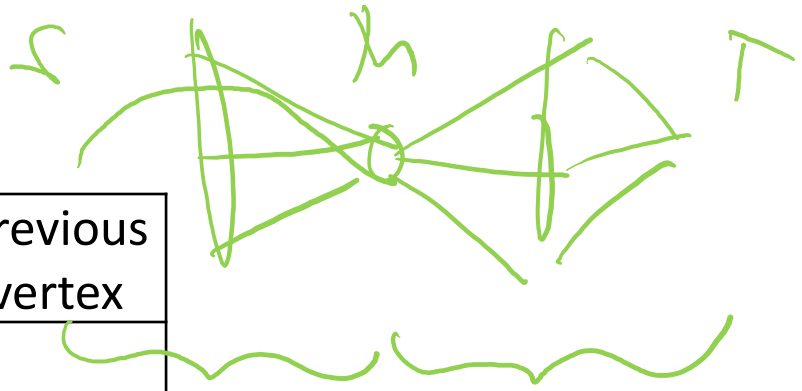
Closed = {s}

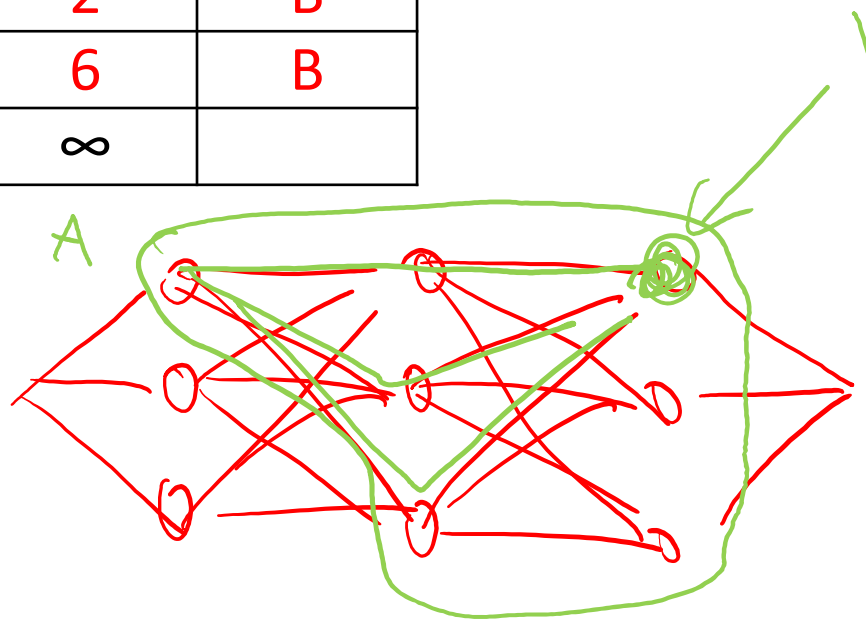
Dijkstra. "A note on two problems in connexion with graphs", Numerische Mathematik, 1959. https://doi.org/10.1007/BF01386390
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Shortest-path algorithms: Dijkstra



	vertex	distance from s (g)	previous vertex
✓	s	0	
	A	3	B
☞	B	1	s
	C	2	B
	D	6	B
	t	∞	





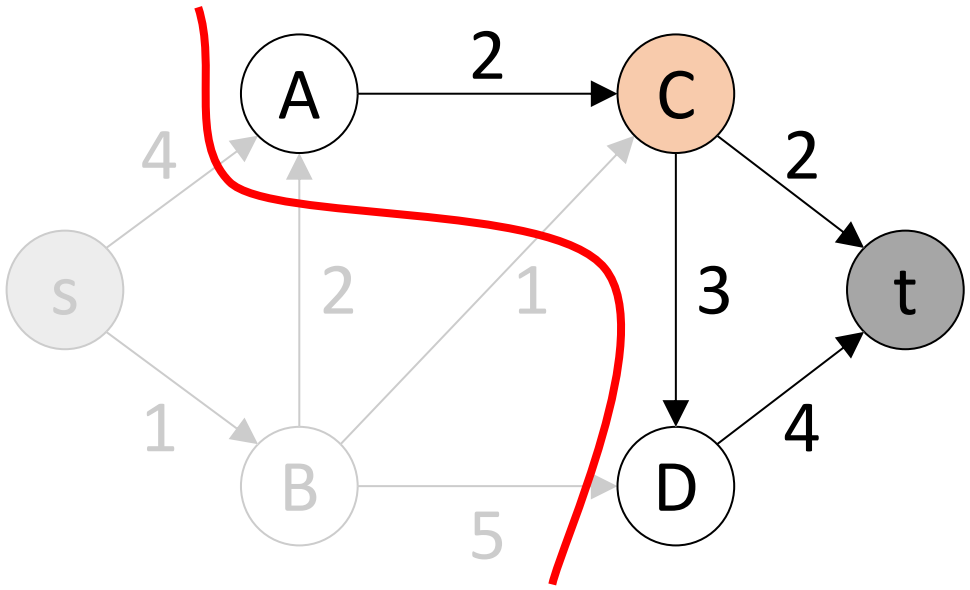
Current: B:1

Open PQ = (C:2,A:3,D:6,t:∞)

Closed = {s}

Dijkstra. "A note on two problems in connexion with graphs", Numerische Mathematik, 1959. https://doi.org/10.1007/BF01386390
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Shortest-path algorithms: Dijkstra



	vertex	distance from s (g)	previous vertex
✓	s	0	
	A	3	B
✓	B	1	s
☞	C	2	B
	D	6	B
	t	∞	

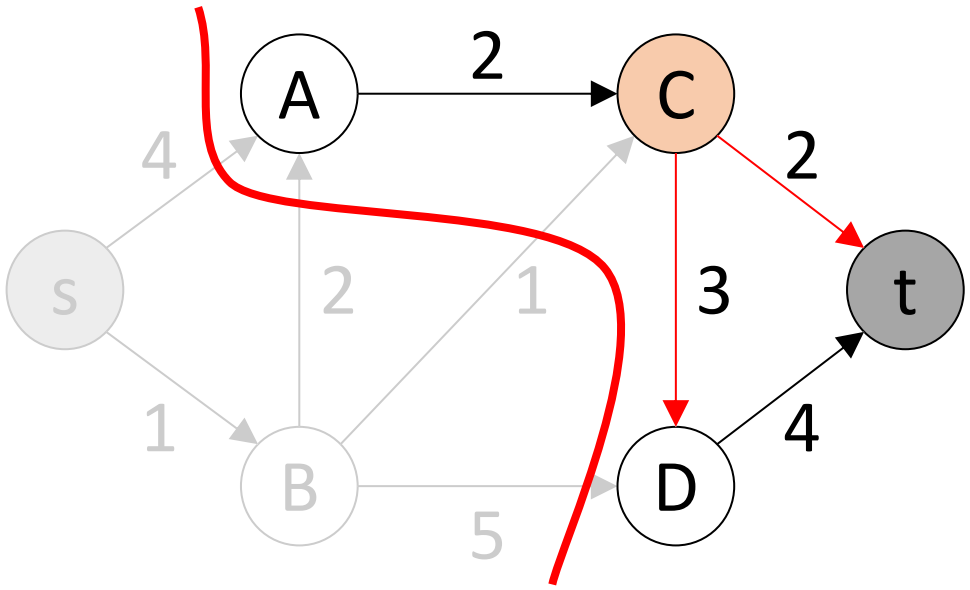
Current: C:2

Open PQ = (A:3,D:6,t:∞)

Closed = {s,B}

Dijkstra. "A note on two problems in connexion with graphs", Numerische Mathematik, 1959. https://doi.org/10.1007/BF01386390
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Shortest-path algorithms: Dijkstra



	vertex	distance from s (g)	previous vertex
✓	s	0	
	A	3	B
✓	B	1	s
☞	C	2	B
	D	5	C
	t	4	C

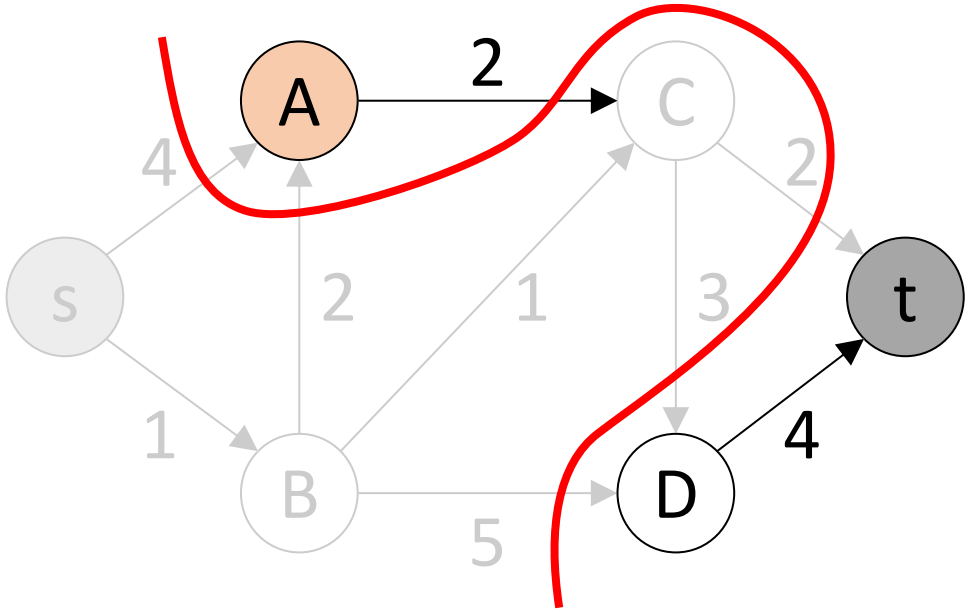
Current: C:2

Open PQ = (A:3,t:4,D:5)

Closed = {s,B}

Dijkstra. "A note on two problems in connexion with graphs", Numerische Mathematik, 1959. https://doi.org/10.1007/BF01386390
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Shortest-path algorithms: Dijkstra



	vertex	distance from s (g)	previous vertex
✓	s	0	
☞	A	3	B
✓	B	1	s
✓	C	2	B
	D	5	C
	t	4	C

Current: A:3

Open PQ = (t:4,D:5)

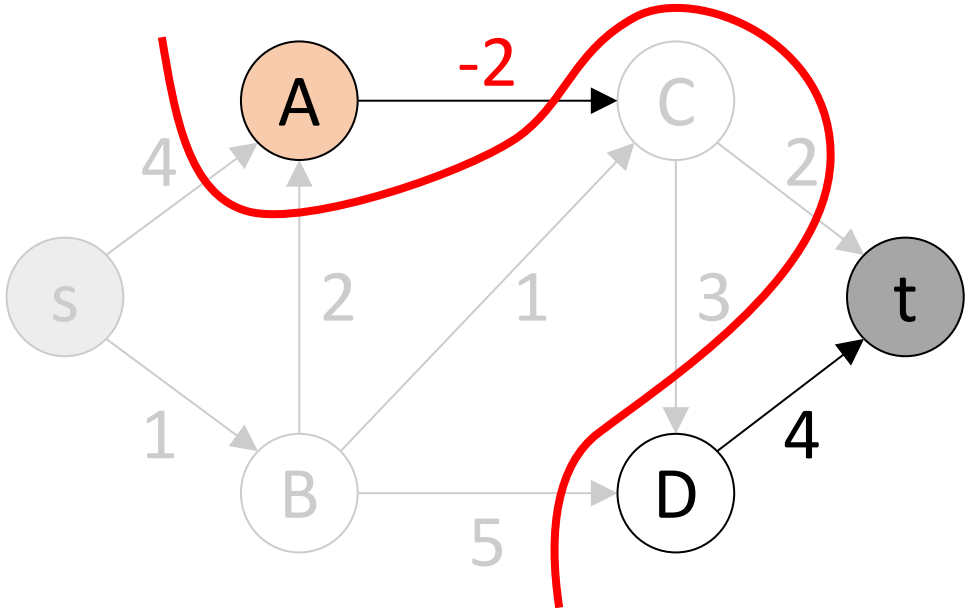
Closed = {s,B,C}

Notice that edge A-C is not used since C is already in Closed! Does not have to explore all edges from current node if all weights ≥ 0

Crucial difference to DP!

Dijkstra. "A note on two problems in connexion with graphs", Numerische Mathematik, 1959. https://doi.org/10.1007/BF01386390
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Shortest-path algorithms: Dijkstra



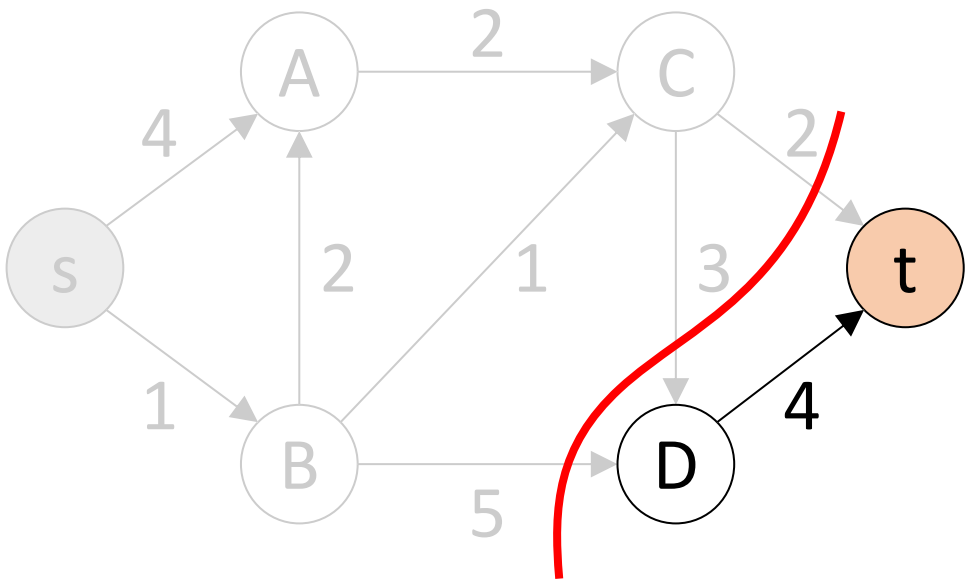
	vertex	distance from s (g)	previous vertex
✓	s	0	
☞	A	3	B
✓	B	1	s
✓	C	2	B
	D	5	C
	t	4	C

Current: A:3
Open PQ = (t:4,D:5)
Closed = {s,B,C}

Dijkstra would not find the shortest path with negative edge since C is closed before being visited from A

Dijkstra. "A note on two problems in connexion with graphs", Numerische Mathematik, 1959. https://doi.org/10.1007/BF01386390
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Shortest-path algorithms: Dijkstra



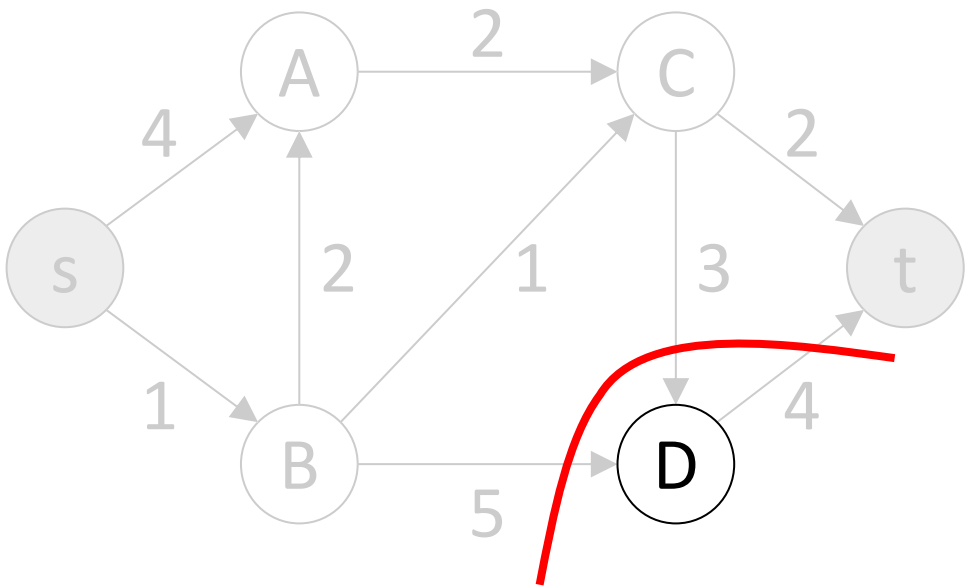
	vertex	distance from s (g)	previous vertex
✓	s	0	
✓	A	3	B
✓	B	1	s
✓	C	2	B
	D	5	C
☞	t	4	C

Current: t:4
Open PQ = (D:6)
Closed = {s,B,C,A}

Once t is popped from the PQ, we have found the shortest path. No need to visit D anymore

Dijkstra. "A note on two problems in connexion with graphs", Numerische Mathematik, 1959. https://doi.org/10.1007/BF01386390
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Shortest-path algorithms: Dijkstra



	vertex	distance from s (g)	previous vertex
✓	s	0	
✓	A	3	B
✓	B	1	s
✓	C	2	B
	D	5	C
✓	t	4	C

Current:
Open PQ = (D:6)
Closed = {s,B,C,A,t}

decrease-key

extract-minimum

Complexity determined by two operations in the PQ:

$\Theta(|E|t_{\mathrm{dk}}+|V|t_{\mathrm{em}}) =$

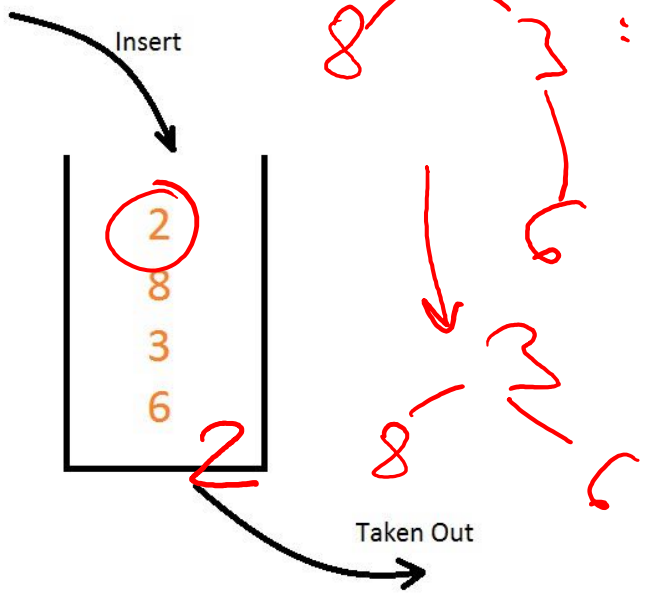
$\Theta((|E|+|V|)\cdot\log|V|)$ for binary heap

$\Theta(|E|\cdot\log|V|)$ for Fibonacci heap

Dijkstra. "A note on two problems in connexion with graphs", Numerische Mathematik, 1959. https://doi.org/10.1007/BF01386390
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Priority queues as black box

- Priority queue (PQ): an abstract data type in which each element has a "priority" associated with it. An element with highest priority is served before other elements



Summary of running times [edit]

Here are time complexities[5] of various heap data structures. Function names assume a min-heap. For the meaning of "$O(f)$" and "$\Theta(f)$" see Big O notation.

Operation	find-min	delete-min	insert	decrease-key	meld
Binary[5]	$\Theta(1)$	$\Theta(\log n)$	$O(\log n)$	$O(\log n)$	$\Theta(n)$
Leftist	$\Theta(1)$	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$	$\Theta(\log n)$
Binomial[5][6]	$\Theta(1)$	$\Theta(\log n)$	$\Theta(1)$[a]	$\Theta(\log n)$	$O(\log n)$[b]
Fibonacci[5][7]	$\Theta(1)$	$O(\log n)$[a]	$\Theta(1)$	$\Theta(1)$[a]	$\Theta(1)$
Pairing[8]	$\Theta(1)$	$O(\log n)$[a]	$\Theta(1)$	$o(\log n)$[a][c]	$\Theta(1)$
Brodal[11][d]	$\Theta(1)$	$O(\log n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Rank-pairing[13]	$\Theta(1)$	$O(\log n)$[a]	$\Theta(1)$	$\Theta(1)$[a]	$\Theta(1)$
Strict Fibonacci[14]	$\Theta(1)$	$O(\log n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
2–3 heap[15]	$O(\log n)$	$O(\log n)$[a]	$O(\log n)$[a]	$\Theta(1)$	?

a. ^ a b c d e f g h i Amortized time.

b. ^ n is the size of the larger heap.

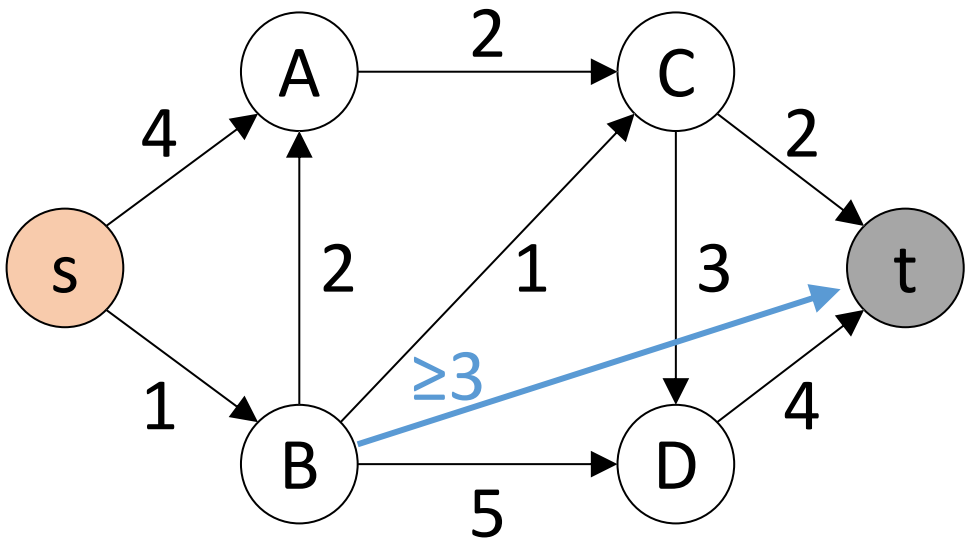
c. ^ Lower bound of $\Omega(\log \log n)$,[9] upper bound of $O(2^{2\sqrt{\log \log n}})$.[10]

d. ^ Brodal and Okasaki later describe a persistent variant with the same bounds except for decrease-key, which is not supported. Heaps with n elements can be constructed bottom-up in $O(n)$.[12]

Screenshot from Wikipedia: https://en.wikipedia.org/wiki/Priority_queue

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Shortest-path algorithms: A* (w/ good heuristics)



Lower bound heuristic

Sort of Open

vertex	distance from s (g)	previous vertex	distance to t (h)	$f=g+h$
s	0		4	4
A			4	
B			3	
C			2	
D			4	
t			0	

Current: s:0

Open PQ = ()

Closed = {}

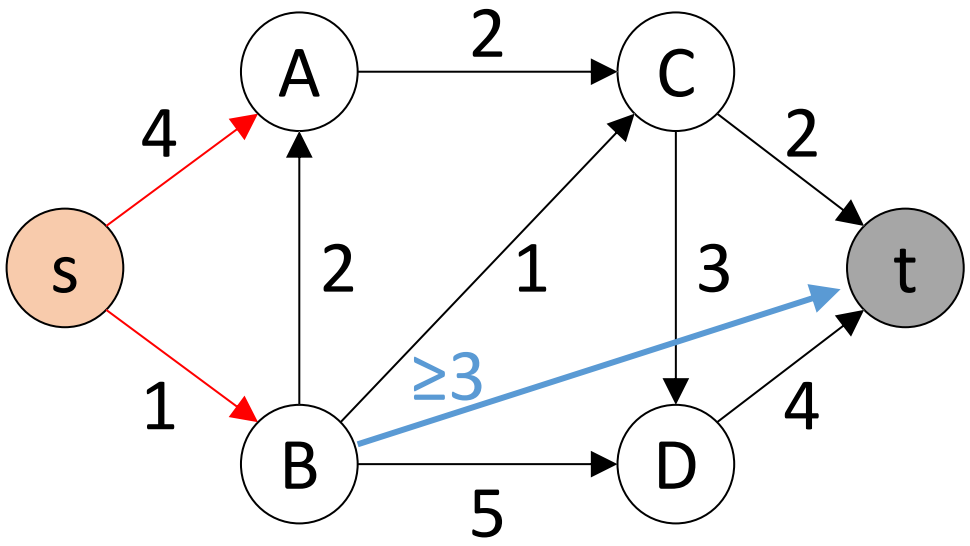
Unknown = {A,B,C,D,t}

To illustrate, the heuristic distance to t is here the actual distance.

A* improves over Dijkstra by not visiting every node and rather being "guided" through the graph (if the heuristics is good)

Hart, Nilsson, Raphael. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". IEEE TSSC, 1968. https://doi.org/10.1109/TSSC.1968.300136
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Shortest-path algorithms: A* (w/ good heuristics)



Lower bound heuristic

Sort of Open

vertex	distance from s (g)	previous vertex	distance to t (h)	$f=g+h$
s	0		4	4
A	4	s	4	8
B	1	s	3	4
C			2	
D			4	
t			0	

Current: s:0

Open PQ = (B:4,A:8)

Closed = {}

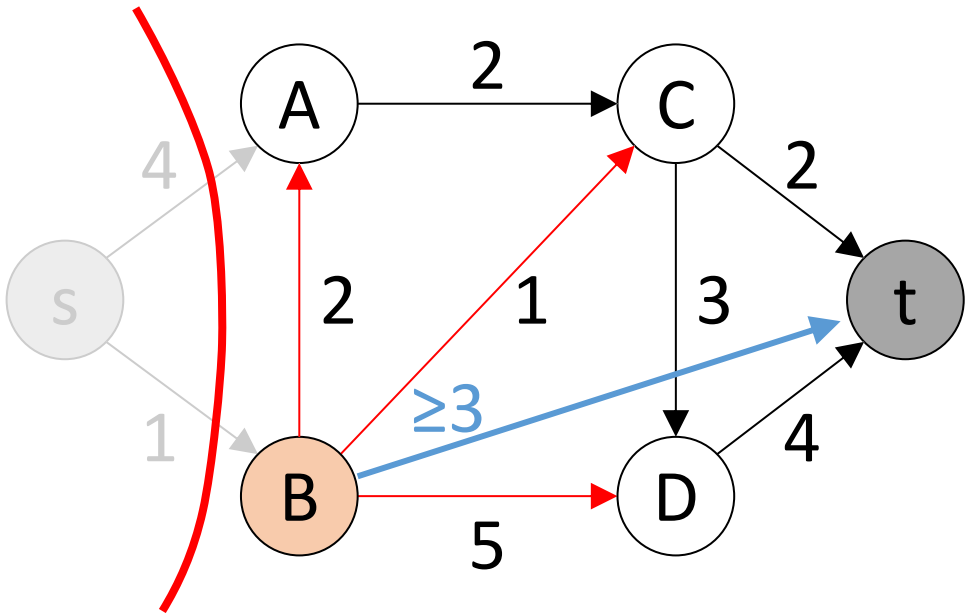
Unknown = {C,D,t}

To illustrate, the heuristic distance to t is here the actual distance.

A* improves over Dijkstra by not visiting every node and rather being "guided" through the graph (if the heuristics is good)

Hart, Nilsson, Raphael. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". IEEE TSSC, 1968. https://doi.org/10.1109/TSSC.1968.300136
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Shortest-path algorithms: A* (w/ good heuristics)



Lower bound heuristic

Sort of Open

	vertex	distance from s (g)	previous vertex	distance to t (h)	$f=g+h$
✓	s	0		4	4
	A	3	B	4	7
☞	B	1	s	3	4
	C	2	B	2	4
	D	6	B	4	10
	t			0	

Current: B:4

Open PQ = (C:4,A:7,D:10)

Closed = {s}

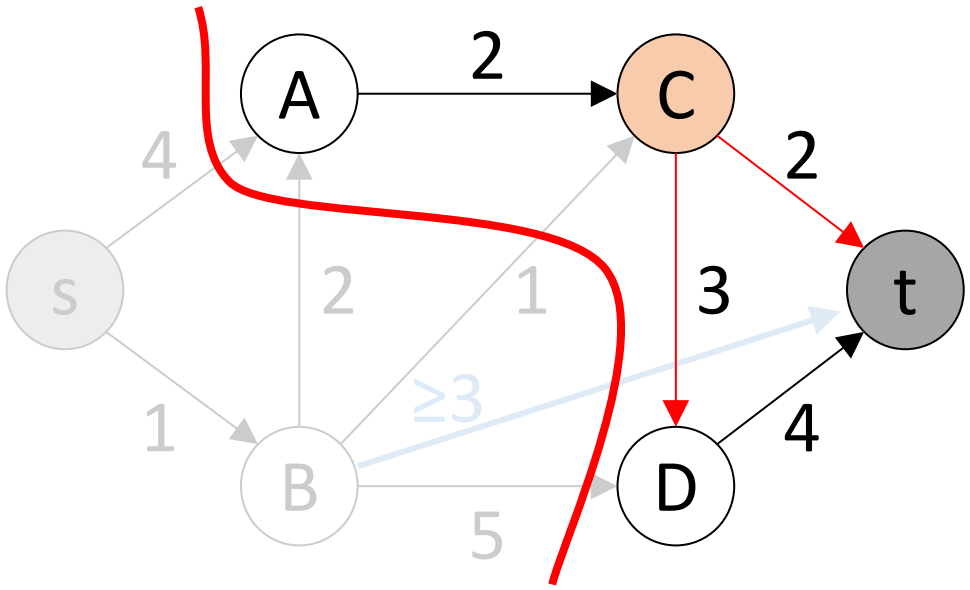
Unknown = {t}

To illustrate, the heuristic distance to t is here the actual distance.

A* improves over Dijkstra by not visiting every node and rather being "guided" through the graph (if the heuristics is good)

Hart, Nilsson, Raphael. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". IEEE TSSC, 1968. https://doi.org/10.1109/TSSC.1968.300136
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Shortest-path algorithms: A* (w/ good heuristics)



Lower bound heuristic

Sort of Open

	vertex	distance from s (g)	previous vertex	distance to t (h)	$f=g+h$
✓	s	0		4	4
	A	3	B	4	7
✓	B	1	s	3	4
☞	C	2	B	2	4
	D	5	C	4	9
	t	4	C	0	4

To illustrate, the heuristic distance to t is here the actual distance.

Current: C:4

Open PQ = (t:4,A:7,D:9)

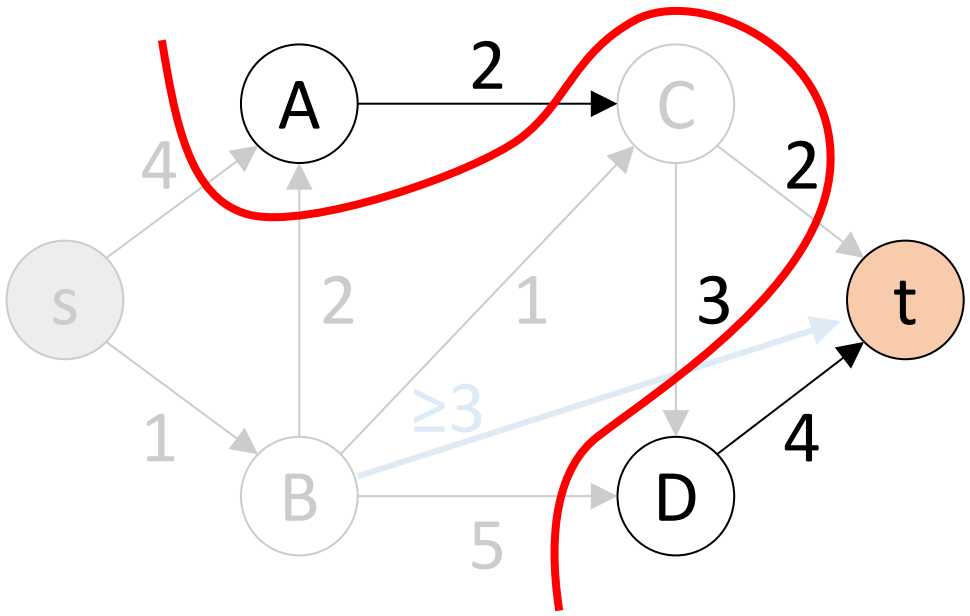
Closed = {s,B}

Unknown = {}

A* improves over Dijkstra by not visiting every node and rather being "guided" through the graph (if the heuristics is good)

Hart, Nilsson, Raphael. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". IEEE TSSC, 1968. https://doi.org/10.1109/TSSC.1968.300136

Shortest-path algorithms: A* (w/ good heuristics)



Lower bound heuristic

Sort of Open

	vertex	distance from s (g)	previous vertex	distance to t (h)	$f=g+h$
✓	s	0		4	4
	A	3	B	4	7
✓	B	1	s	3	4
✓	C	2	B	2	4
	D	5	C	4	9
☞	t	4	C	0	4

Current: t:4

Open PQ = (A:7,D:9)

Closed = {s,B,C}

Unknown = {}

To illustrate, the heuristic distance to t is here the actual distance.

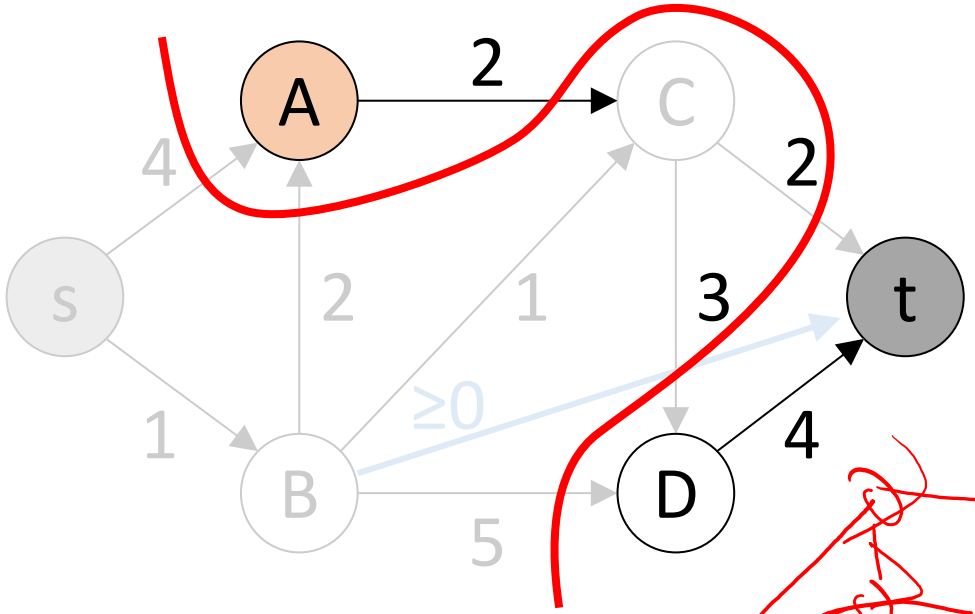
A* improves over Dijkstra by not visiting every node and rather being "guided" through the graph (if the heuristics is good)

Hart, Nilsson, Raphael. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". IEEE TSSC, 1968. https://doi.org/10.1109/TSSC.1968.300136

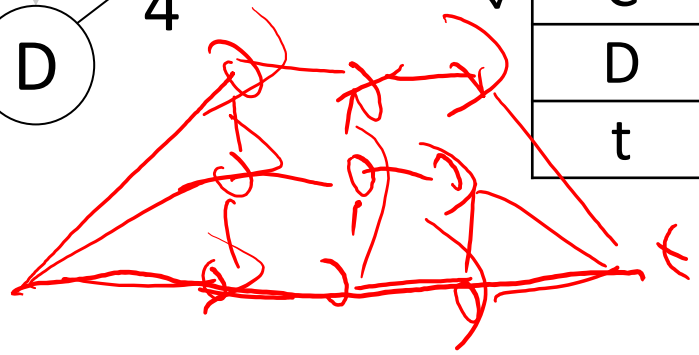
Shortest-path algorithms: A* (w/o heuristics)

if no good lower bound heuristic known

Sort of Open



	vertex	distance from s (g)	previous vertex	distance to t (h)	$f=g+h$
✓	s	0		0	0
☞	A	3	B	0	3
✓	B	1	s	0	1
✓	C	2	B	0	2
	D	5	C	0	5
	t	4	C	0	4



Current: A:3

Open PQ = (t:5,D:6)

Closed = {s,B,C}

Unknown = {}

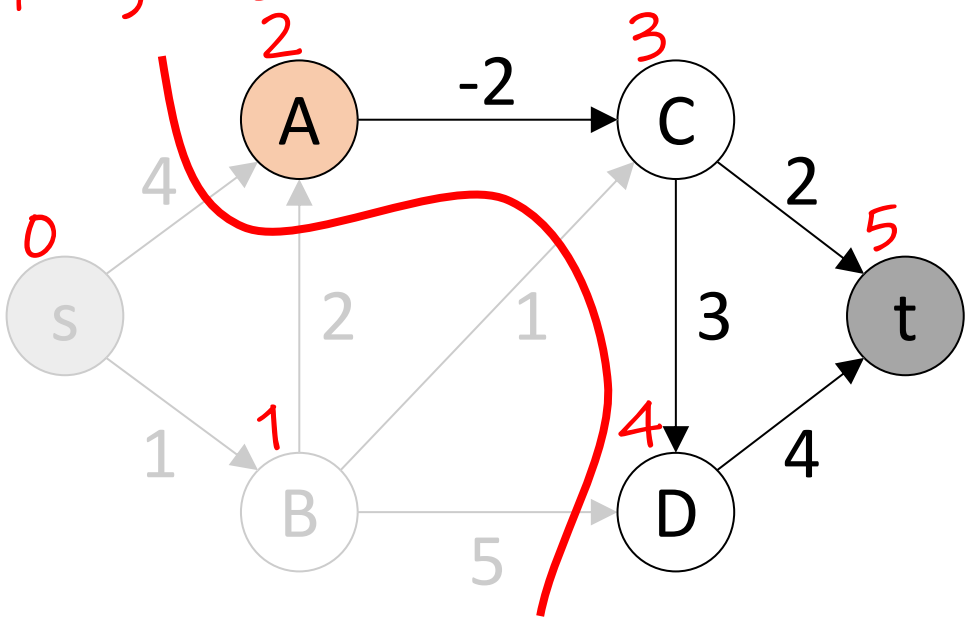
If we have no heuristic lower bound (other than 0), then A* becomes Uniform-cost-search = Dijkstra. We pick A over t since its known distance from s is smaller.

Hart, Nilsson, Raphael. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". IEEE TSSC, 1968. https://doi.org/10.1109/TSSC.1968.300136

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Shortest-path algorithms: Bellman-Ford

Topological sort



Sort of Open

	vertex	distance from s	previous vertex	topological sort
✓	s	0		0
☞	A	3	B	2
✓	B	1	s	1
	C	~~2~~ 1	~~C~~ A	3
	D	6	D	4
	t	∞		5

Current: A

Open PQ = (C:3,D:4,t:5)

Closed = {s,B}

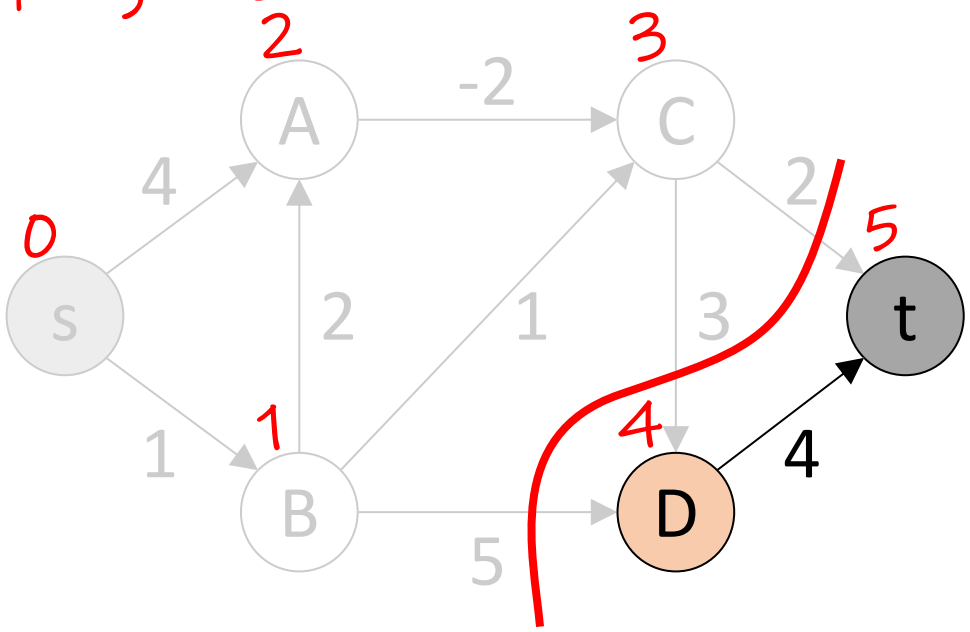
Nodes are visited in topological order (that is possible if graph is acyclic). Now the negative edge (A-C) does not cause problems.

[The general Bellman-Ford does not need a topological sort but visits each node |V-1| times.]

On the history (Shimbel, Bellman, Ford): Schrijver, "On the History of Combinatorial Optimization (Till 1960)", 2005. https://doi.org/10.1016/S0927-0507(05)12001-5

Shortest-path algorithms: Bellman-Ford

Topological sort



Sort of Open

	vertex	distance from s	previous vertex	topological sort
✓	s	0		0
✓	A	3	B	2
✓	B	1	s	1
✓	C	1	A	3
☞	D	4	C	4
	t	3	C	5

Current: D

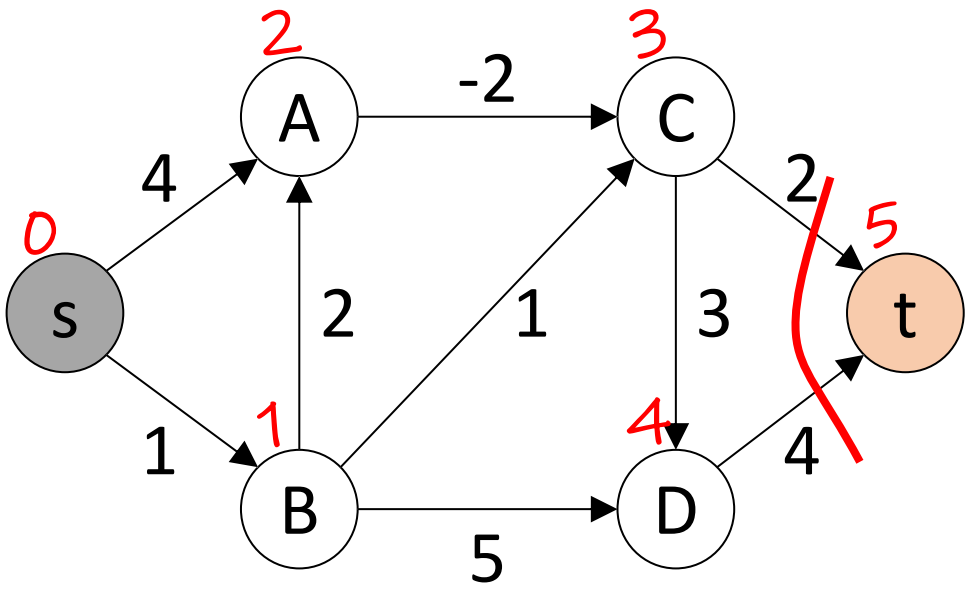
Open PQ = (t:5)

Closed = {s,B,A,C}

On the history (Shimbel, Bellman, Ford): Schrijver, "On the History of Combinatorial Optimization (Till 1960)", 2005. https://doi.org/10.1016/S0927-0507(05)12001-5

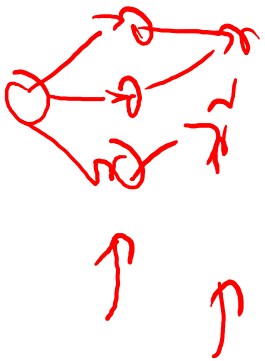
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Shortest-path algorithms: Bellman-Ford



Sort of Open

	vertex	distance from s	previous vertex	topological sort
✓	s	0		0
✓	A	3	B	2
✓	B	1	s	1
✓	C	1	A	3
✓	D	4	C	4
☞	t	3	C	5



Current: t

Open PQ = ()

Closed = {s,B,A,C,D}

Complexity:

$O(|E|)$ w/ topological sort (if we have a k-partite graph, then we have DP = Dynamic Programming)

[$O(|E| \cdot |V|)$ w/o topological sort (visits each node repeatedly)]

On the history (Shimbel, Bellman, Ford): Schrijver, "On the History of Combinatorial Optimization (Till 1960)", 2005. https://doi.org/10.1016/S0927-0507(05)12001-5

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

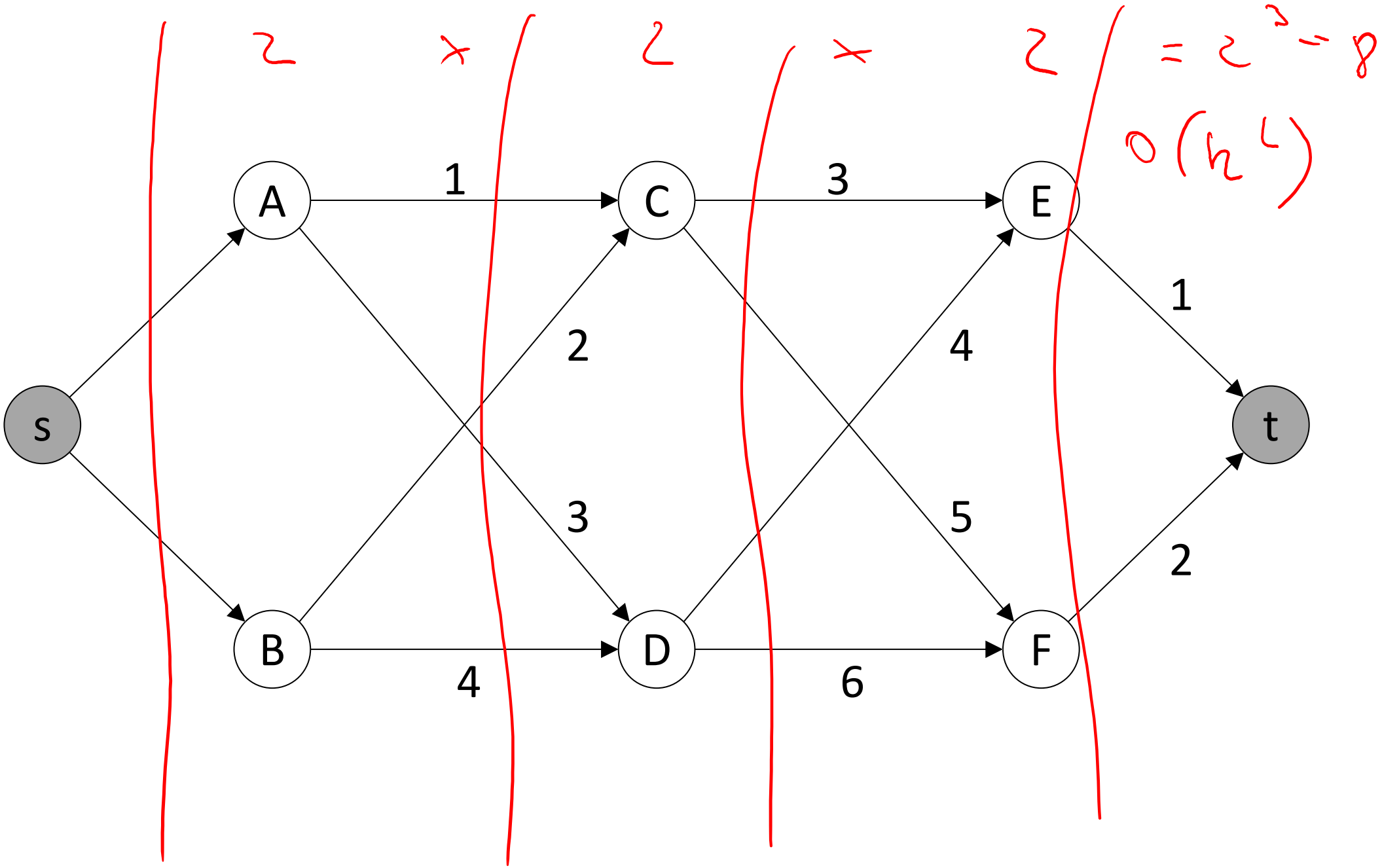
Shortest-path algorithms on digraphs (directed graphs)

- Dijkstra
  - identical to uniform-cost-search (cost g from source)
  - only works for positive edges (can not handle negative edges)
- A*
  - variant of best-search Dijkstra (cost g from source + cost h to target)
  - visits even fewer nodes with good heuristics
- Bellman Ford
  - visits all nodes and edges (possibly multiple times)
  - more work, but negative weights are ok
  - becomes Dynamic Programming (DP) with topological sort (visits every node and edge exactly once)

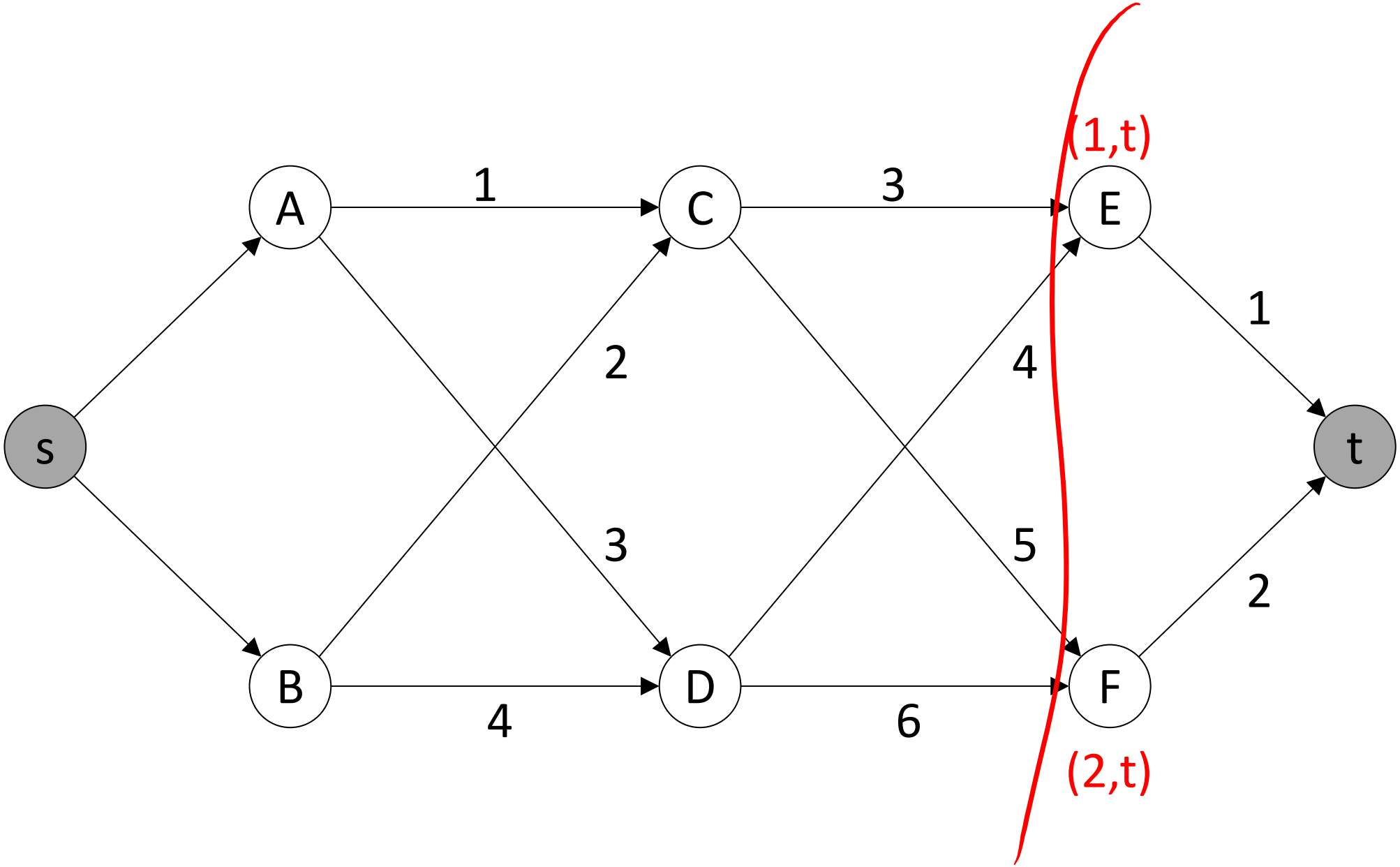
Outline: T3-4: Optimization, Top-k, Ranked Enumeration

- Dynamic Programming (DP)
  - Shortest path algorithms
  - DP & shortest path enumeration
  - Non-serial DP (NSDP) for solving SAT
  - Yannakakis and NSDP
  - Algebraic Structures (Semirings)
- Top-k
- Ranked Enumeration

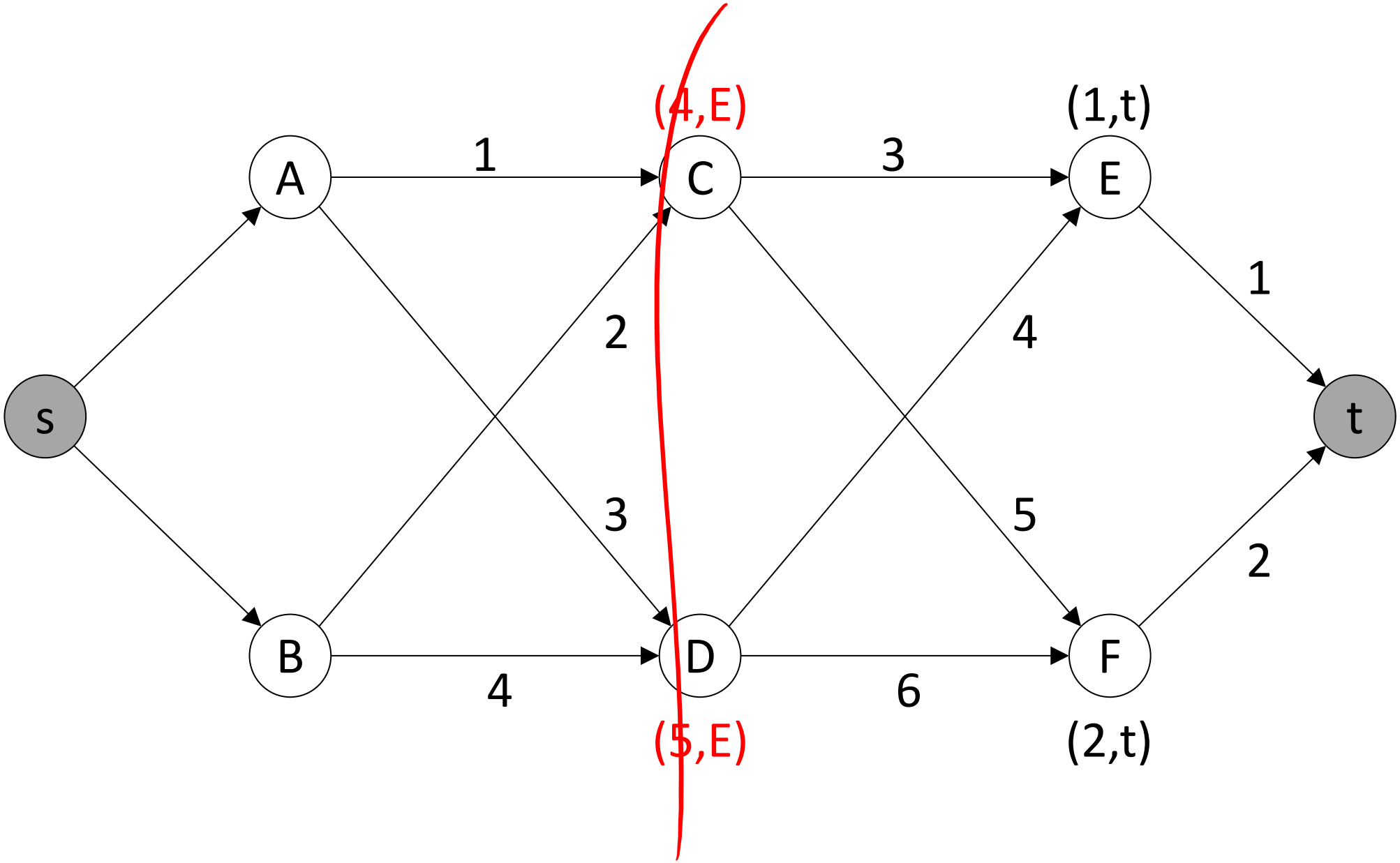
DP



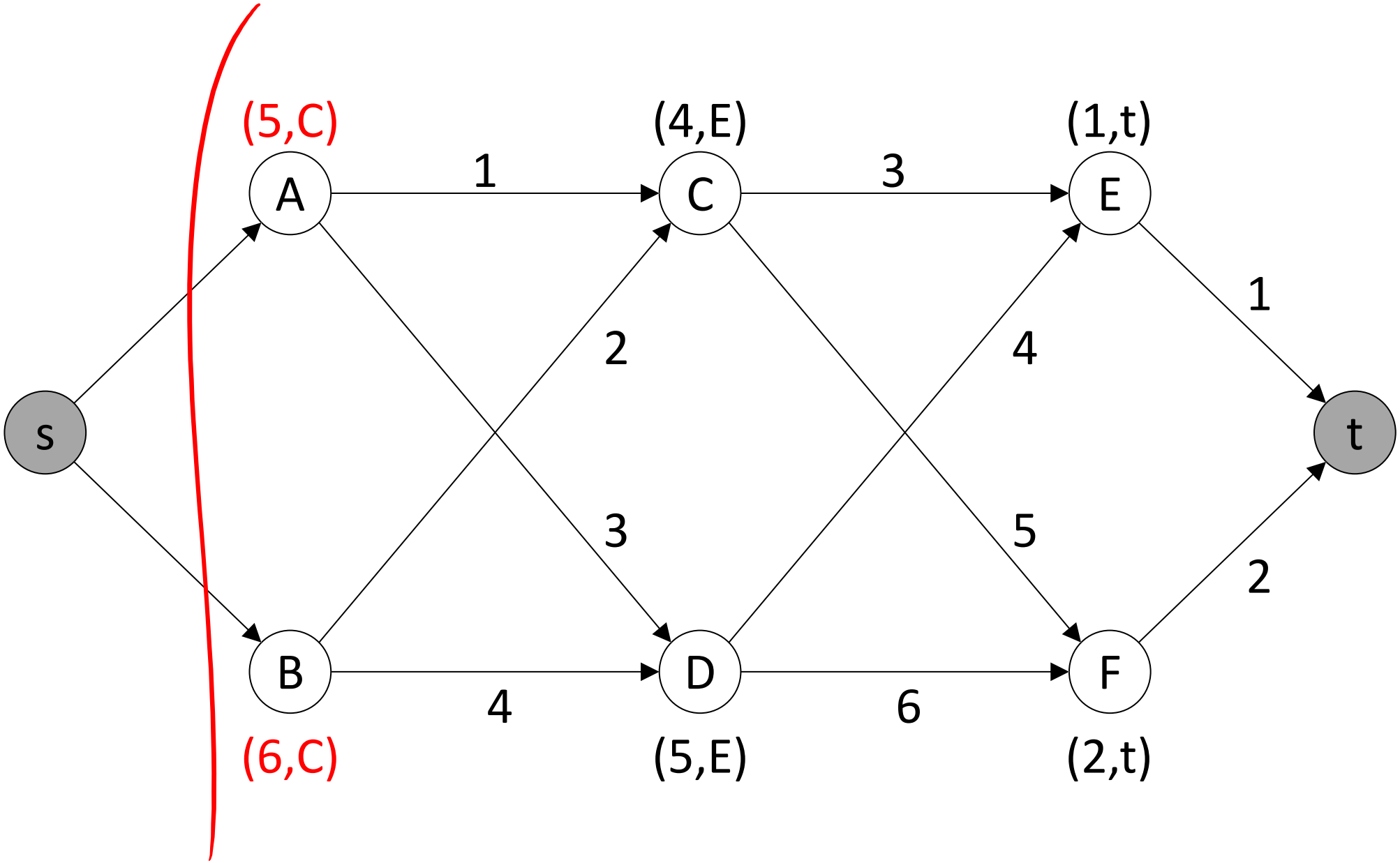
DP



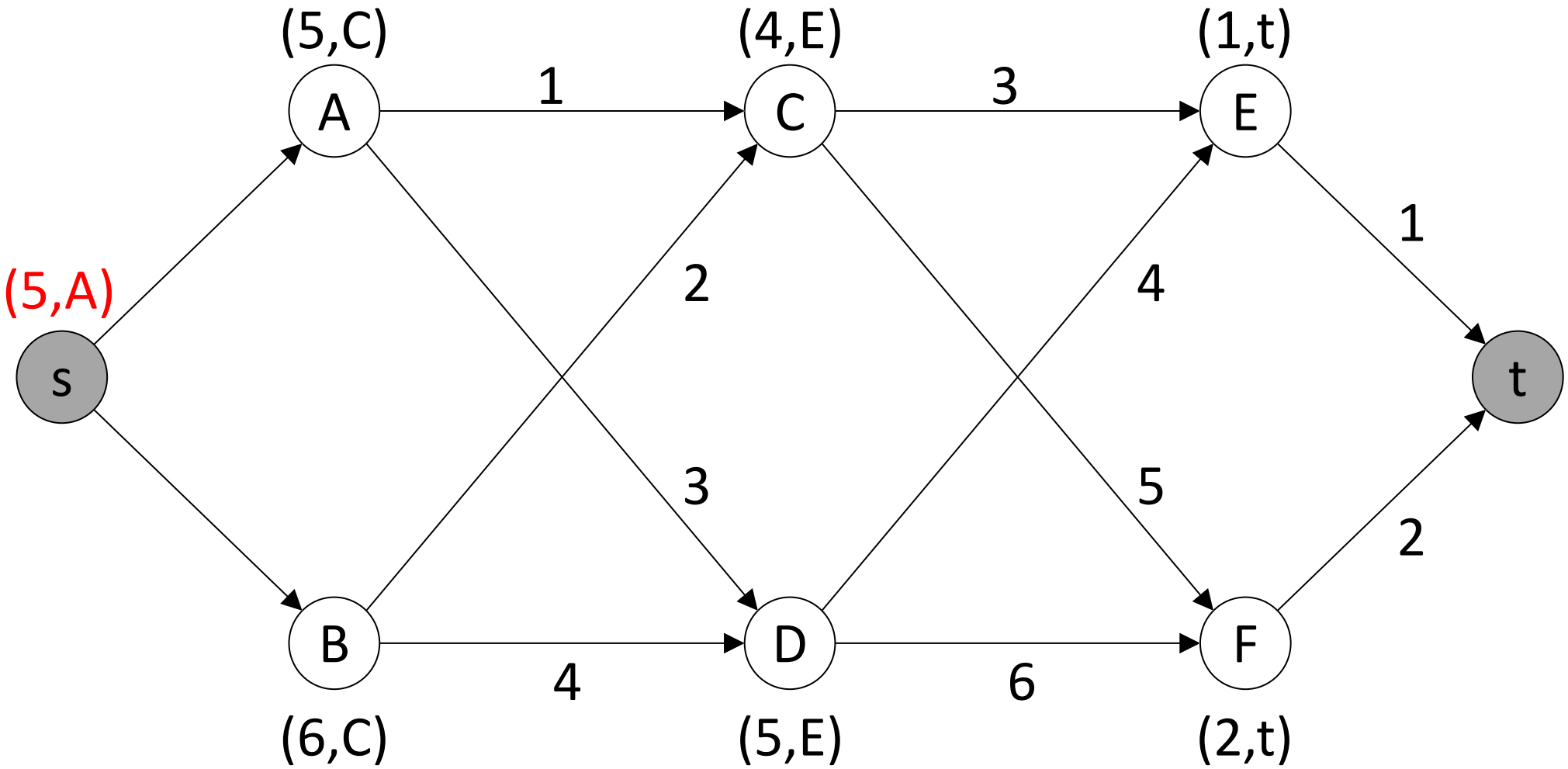
DP



DP

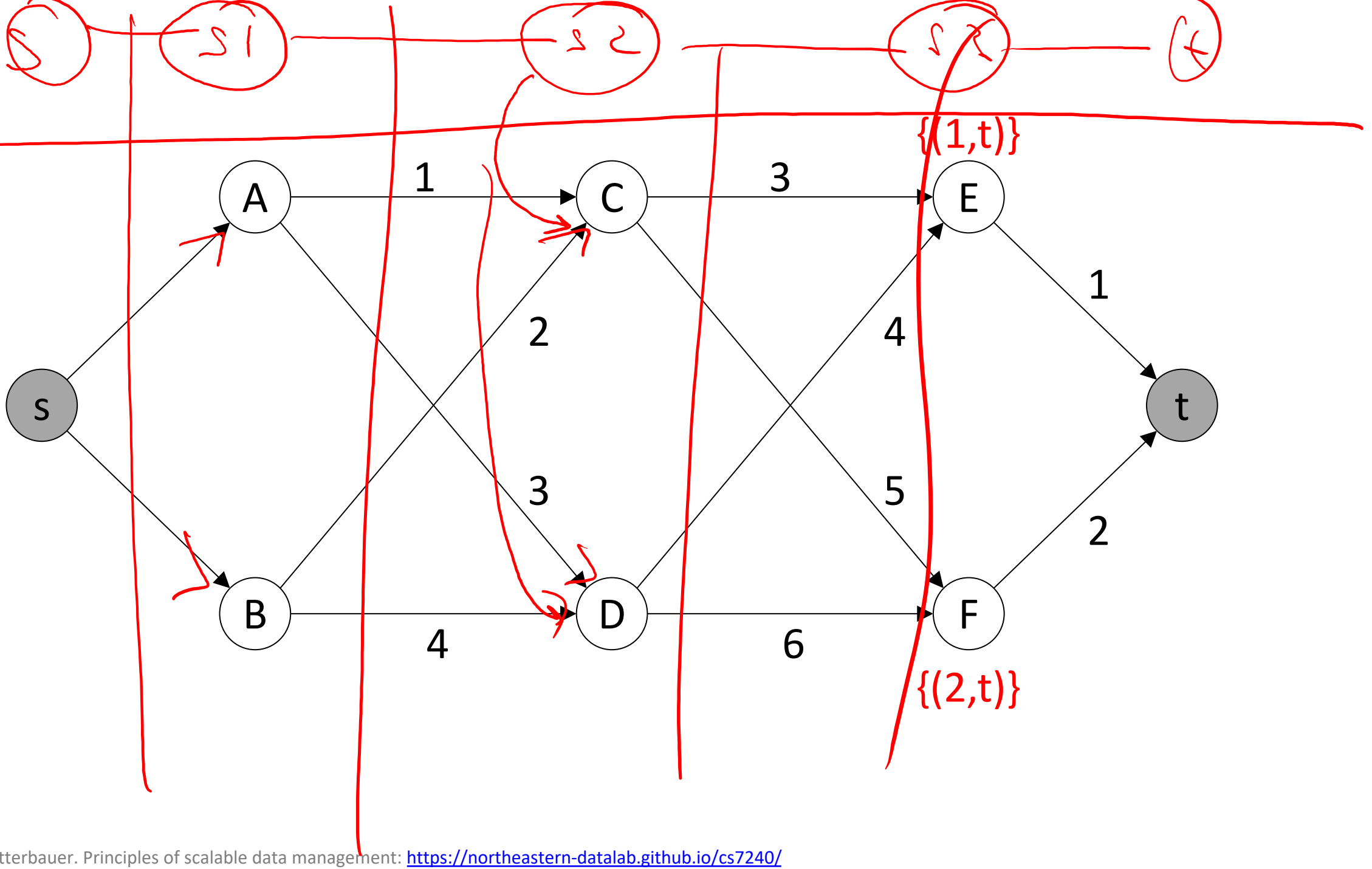


DP

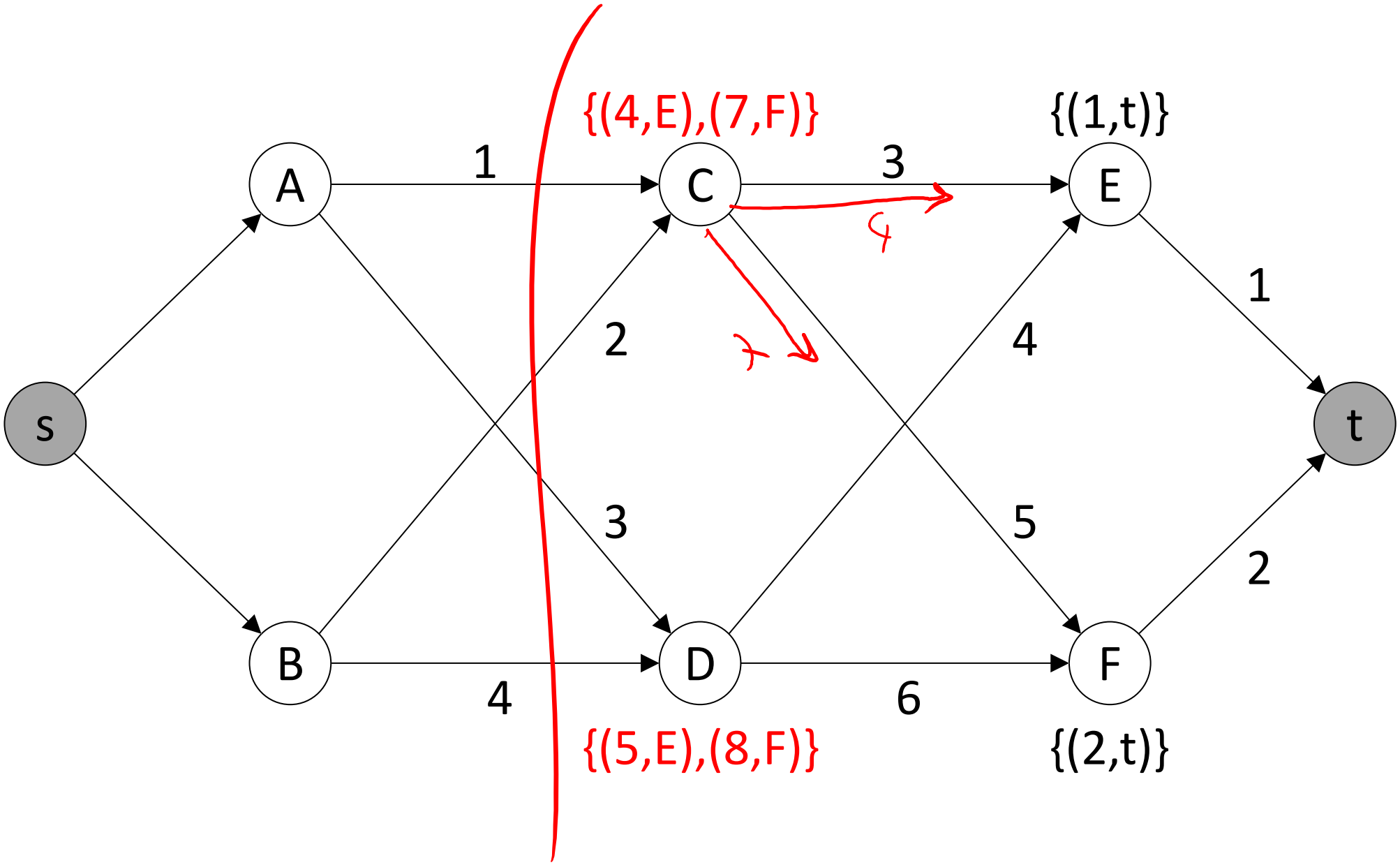


DP
w/ slight modification during bottom-up

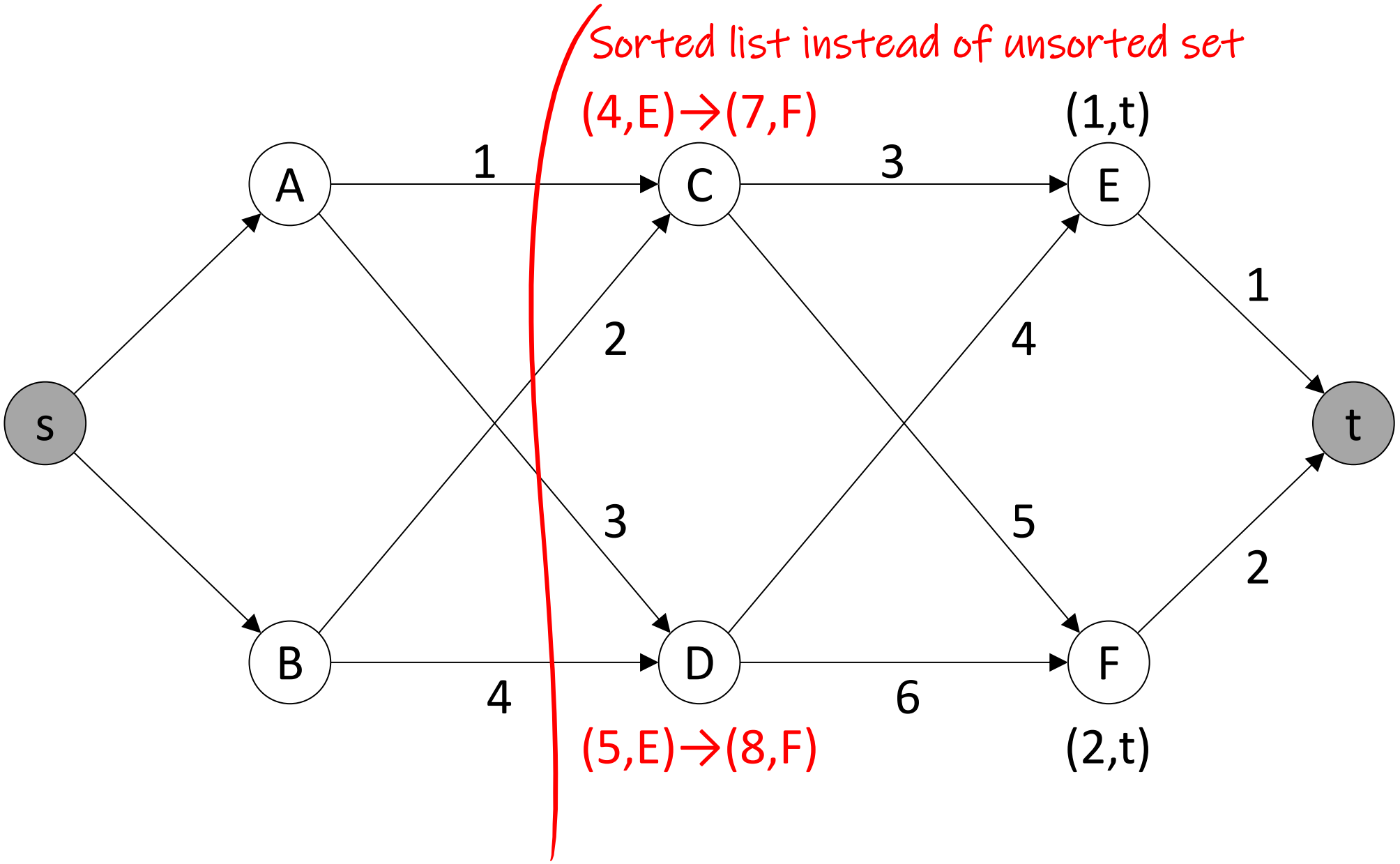
DP



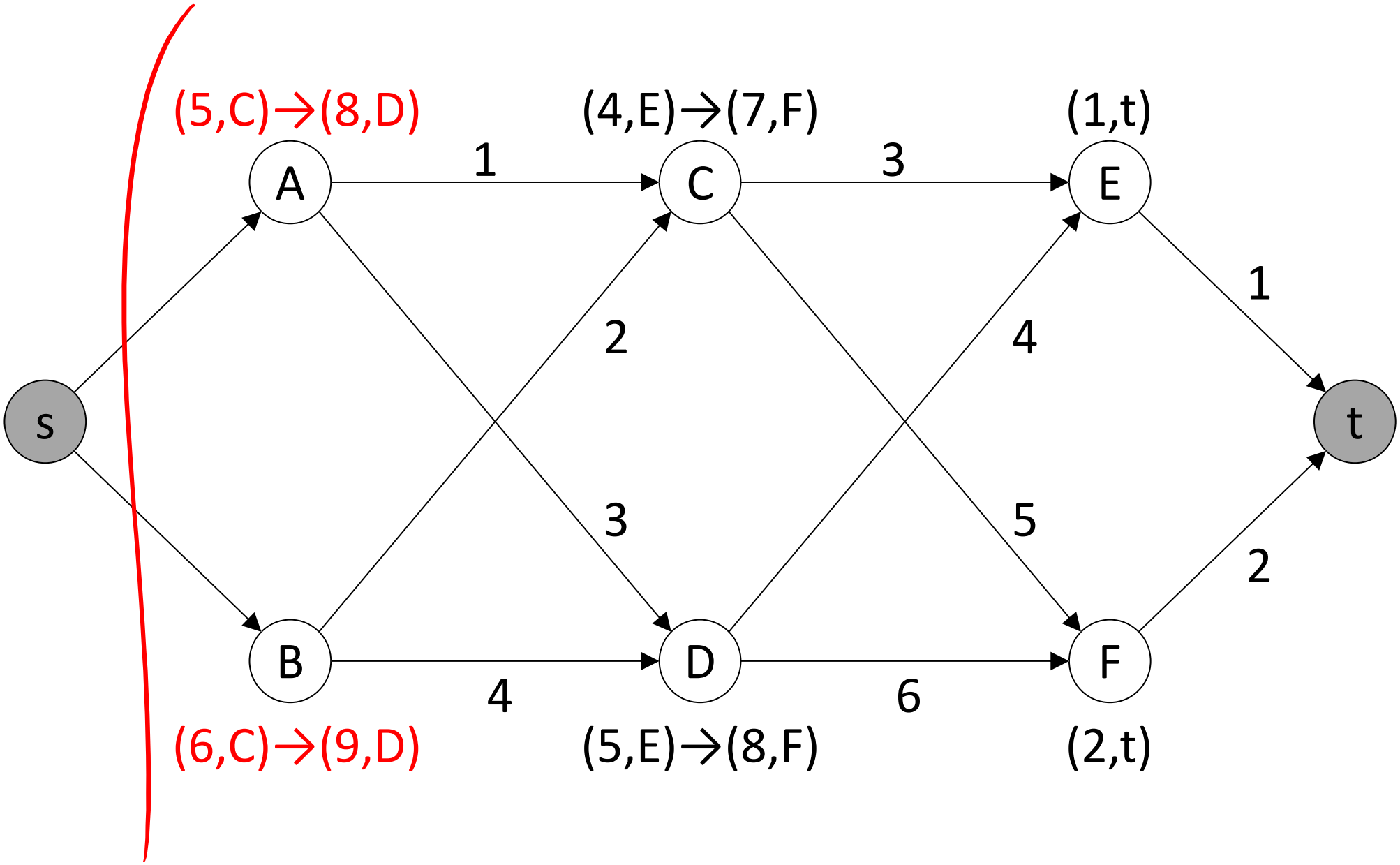
DP



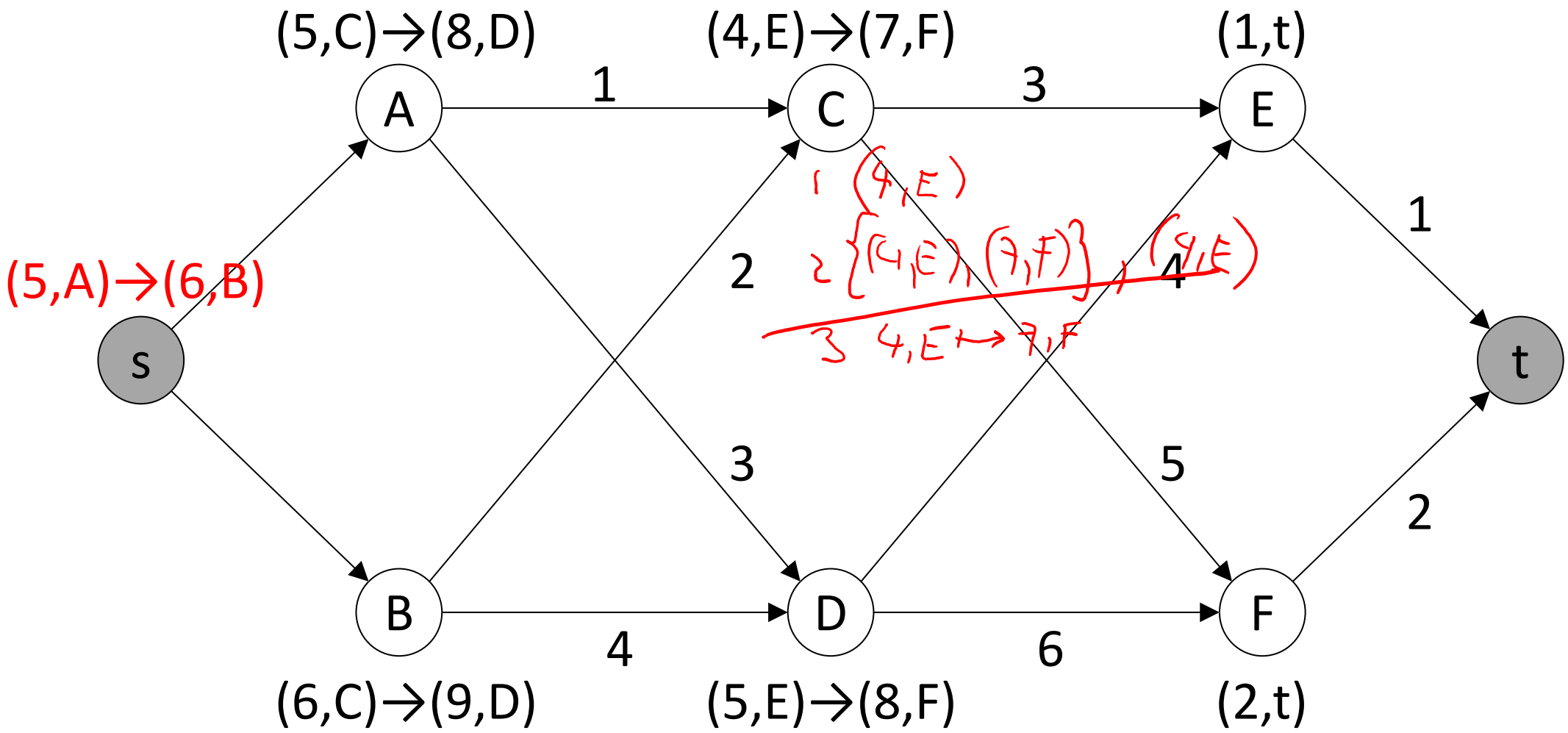
DP


Sorted list instead of unsorted set
(4,E)→(7,F)
(1,t)
A
C
E
1
3
1
2
4
s
t
3
5
2
B
D
F
4
6
(5,E)→(8,F)
(2,t)

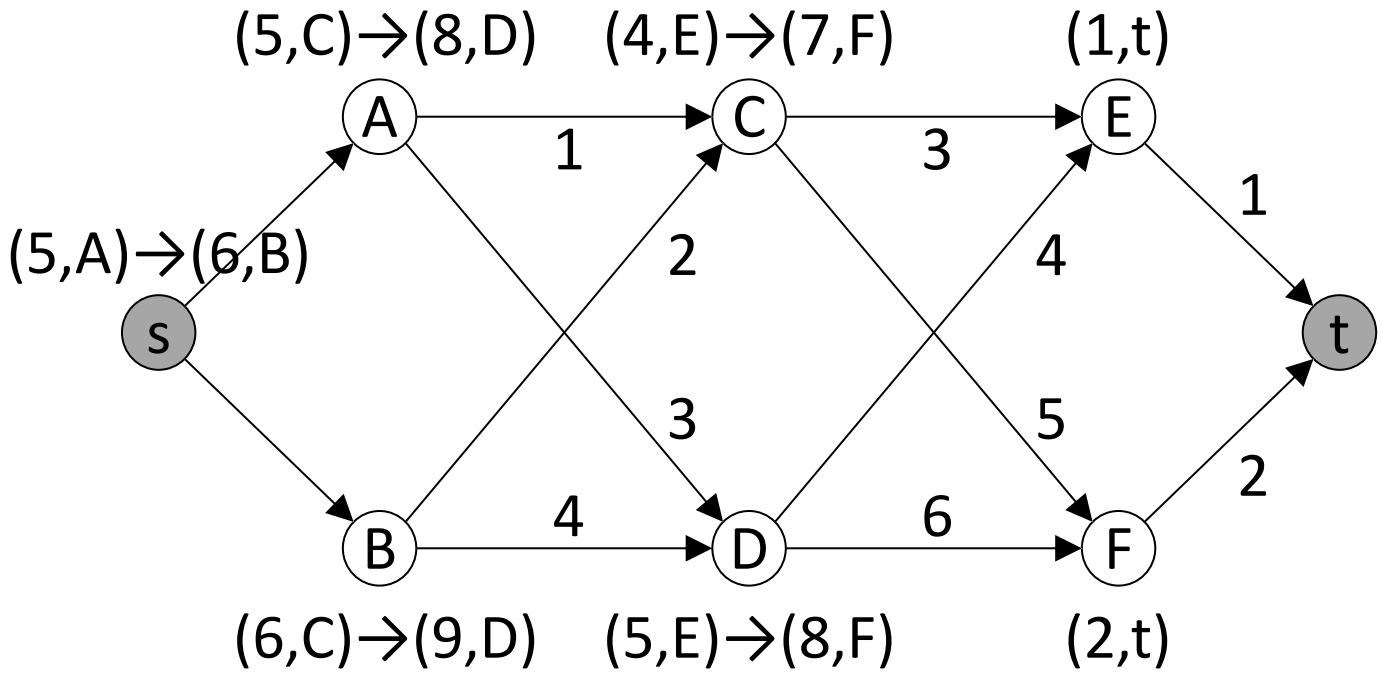
DP



DP



Adopted DP w/
another slight modification
now during top-down
= Any-k



Initialize PQ of prefixes with pointers from s

Repeat {

p = PQ.pop

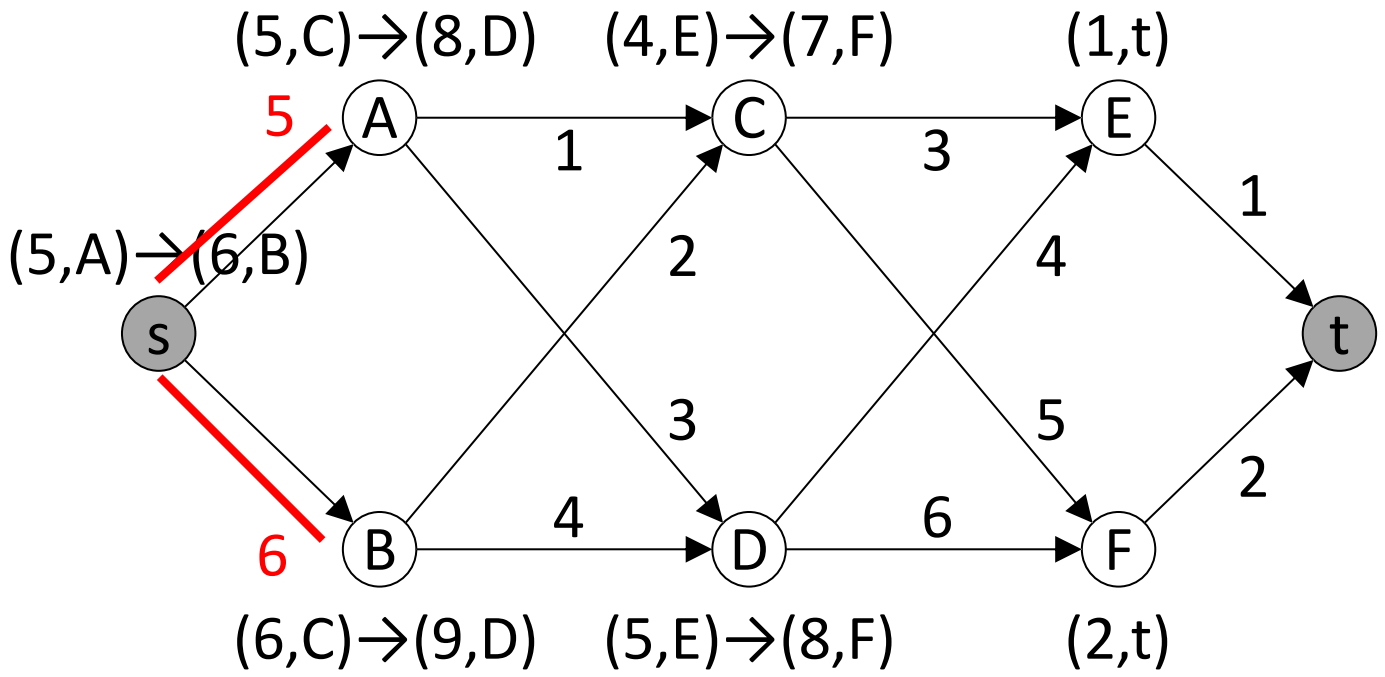
If p is a full path then output(p)

else expand p and push to PQ

until PQ = ∅}

PQ

(5,A)→(6,B)



Initialize PQ of prefixes with pointers from s

Repeat {

p = PQ.pop

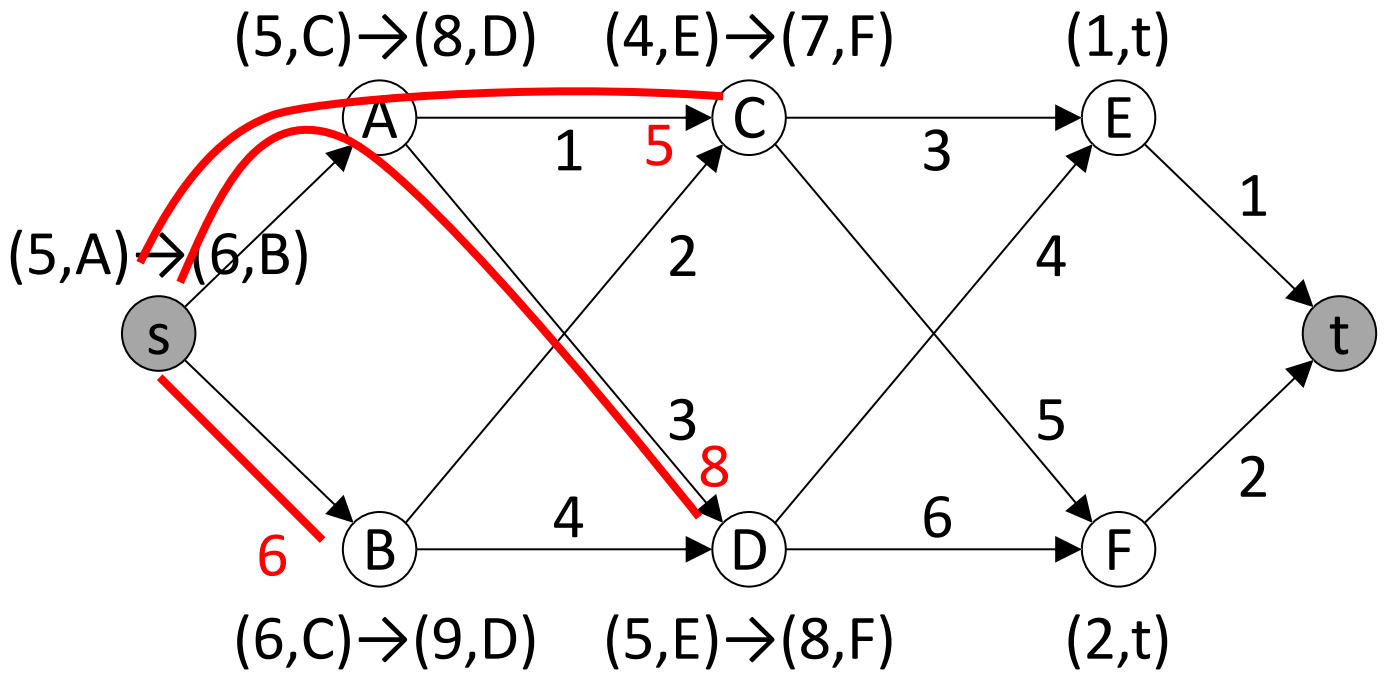
If p is a full path then output(p)

else expand p and push to PQ

until PQ = ∅}

PQ

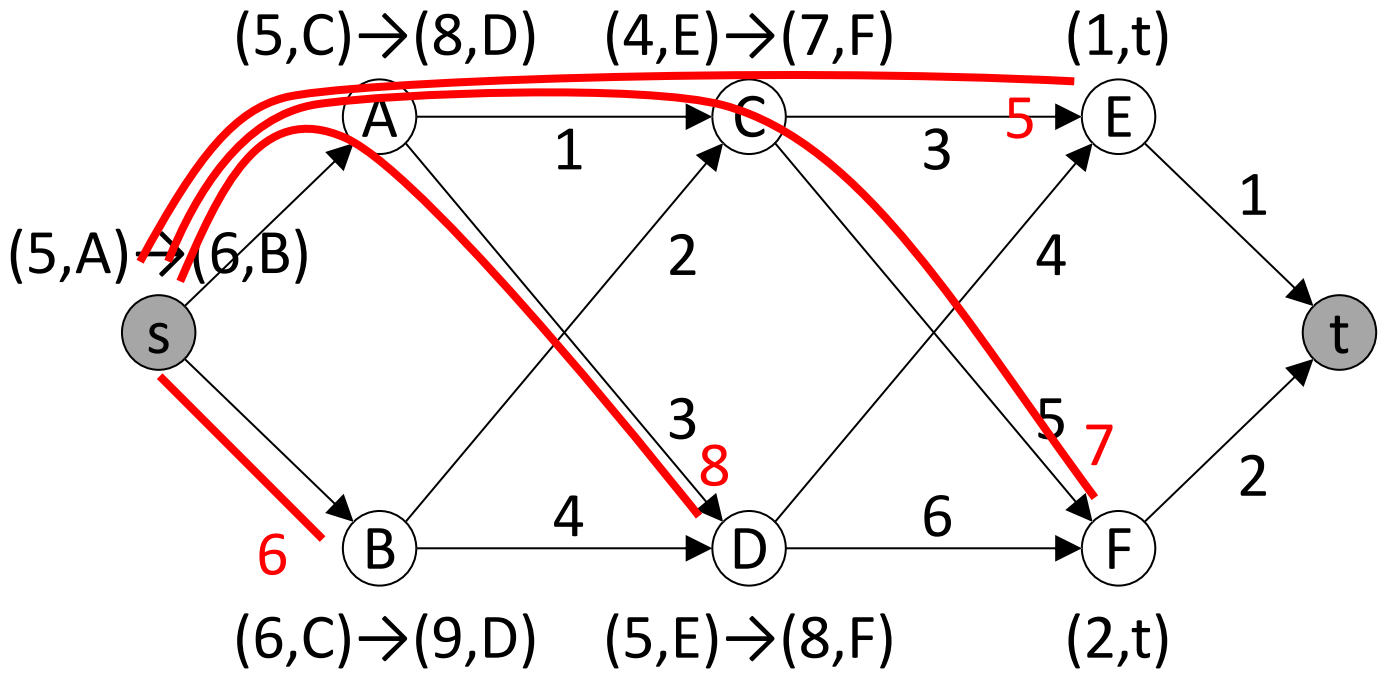
(5,A)→(6,B)
(5,AC)→(6,B)→(8,AD)



Initialize PQ of prefixes with pointers from s
Repeat {
    p = PQ.pop
    If p is a full path then output(p)
        else expand p and push to PQ
until PQ = ∅}

PQ

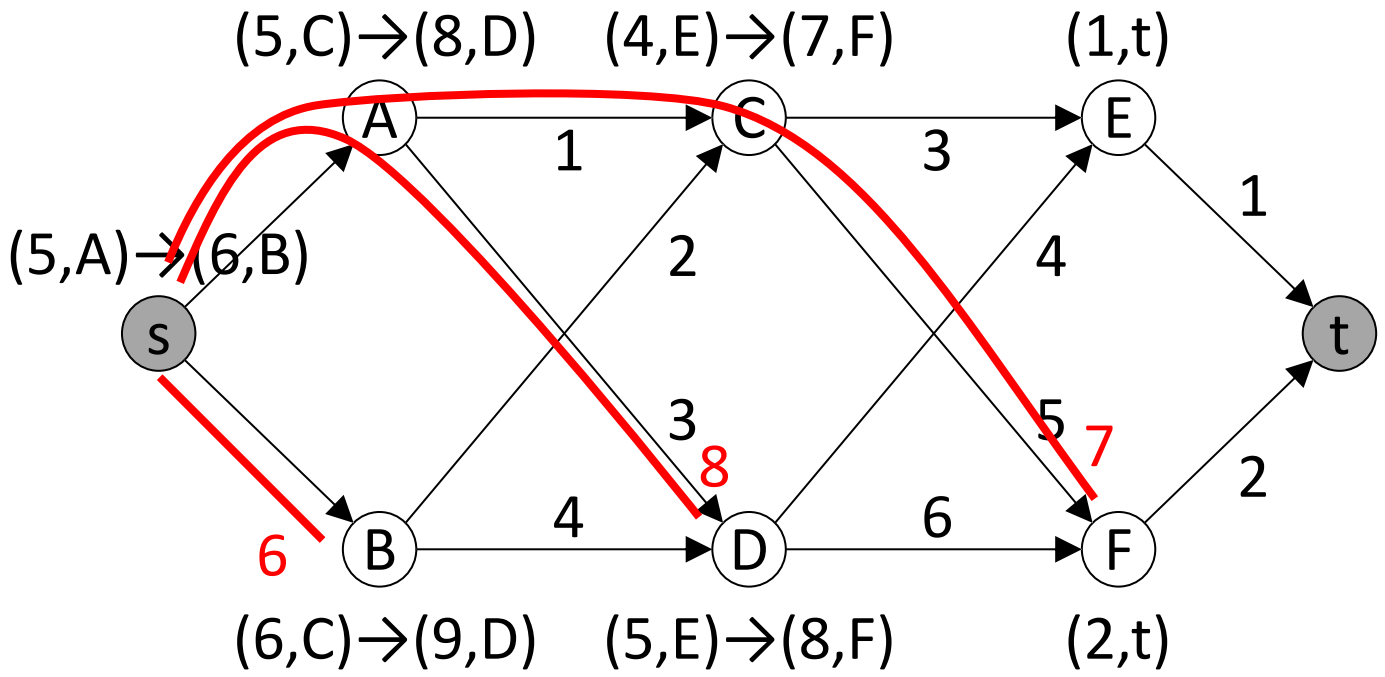
(5,A)→(6,B)
(5,AC)→(6,B)→(8,AD)
(5,ACE)→(6,B)→(7,ACF)→(8,AD)



Initialize PQ of prefixes with pointers from s
Repeat {
    p = PQ.pop
    If p is a full path then output(p)
        else expand p and push to PQ
until PQ = ∅}

PQ

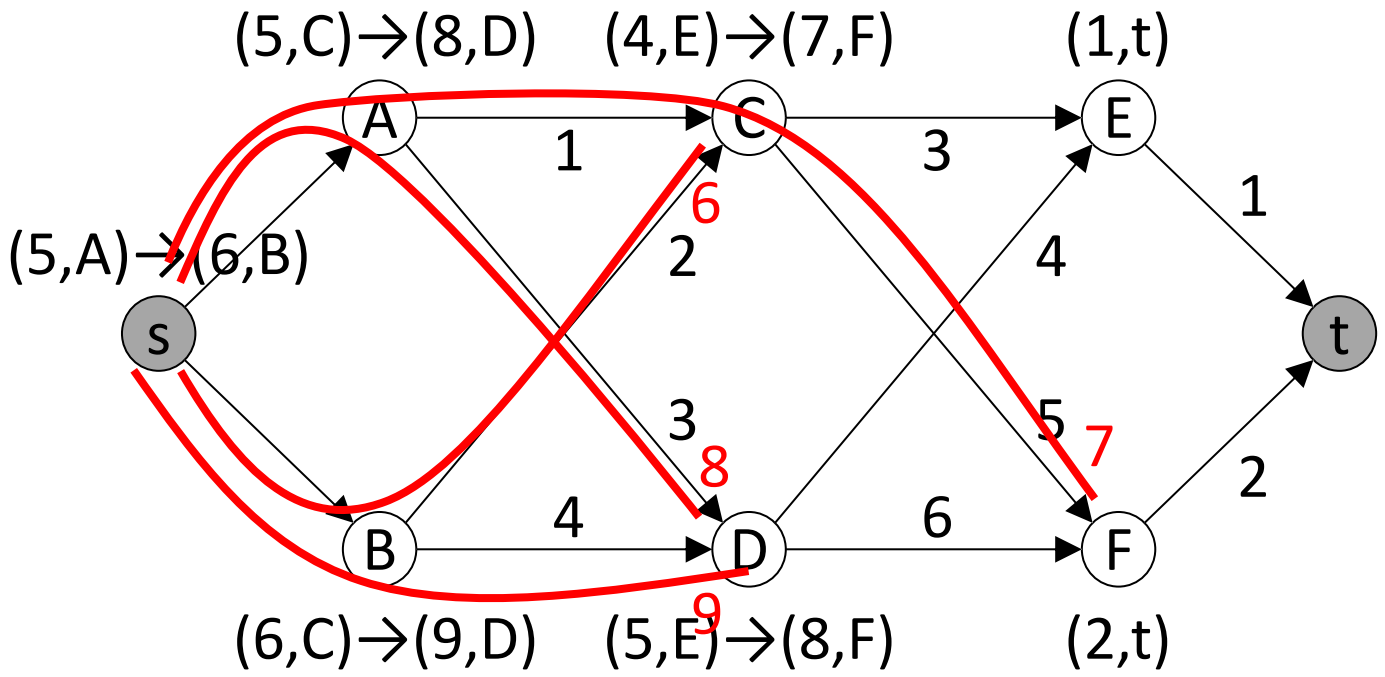
(5,A)→(6,B)
(5,AC)→(6,B)→(8,AD)
(5,ACE)→(6,B)→(7,ACF)→(8,AD)
(6,B)→(7,ACF)→(8,AD)



Initialize PQ of prefixes with pointers from s
Repeat {
    p = PQ.pop
    If p is a full path then output(p)
        else expand p and push to PQ
until PQ = ∅}

PQ

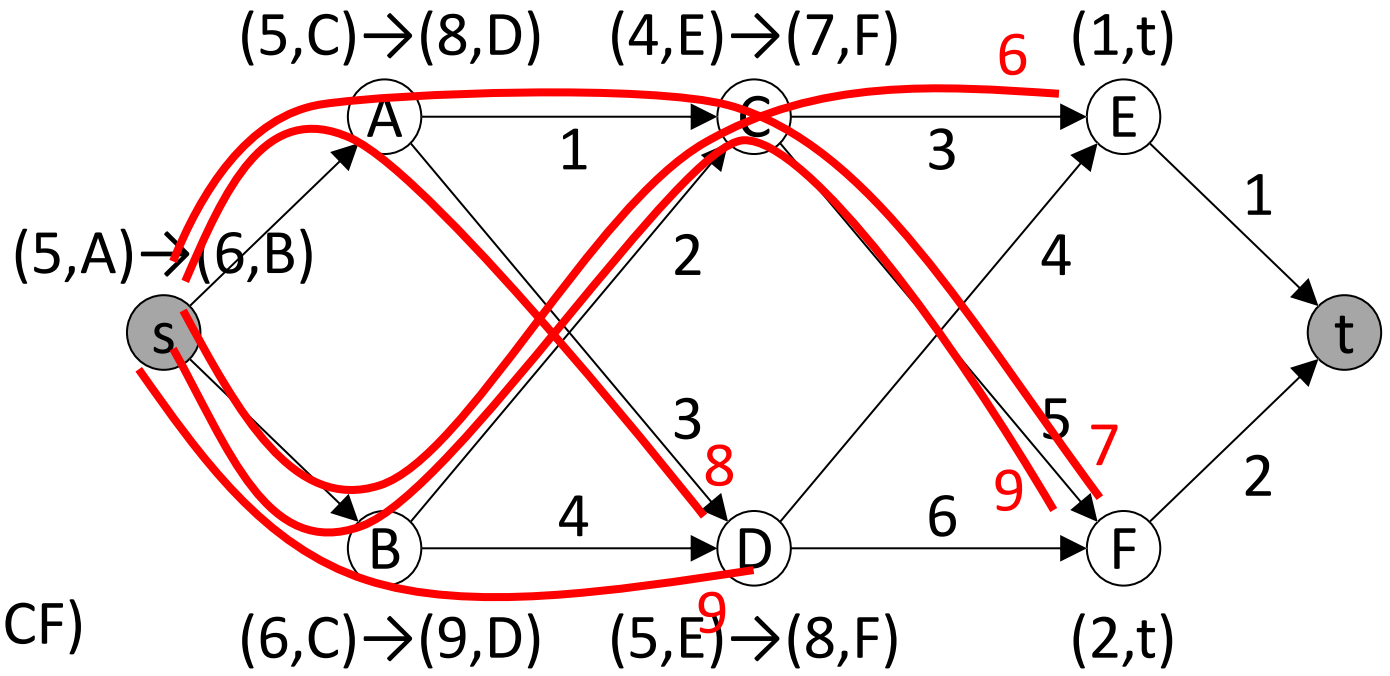
(5,A)→(6,B)
(5,AC)→(6,B)→(8,AD)
(5,ACE)→(6,B)→(7,ACF)→(8,AD)
(6,B)→(7,ACF)→(8,AD)
(6,BC)→(7,ACF)→(8,AD)→(9,BD)



Initialize PQ of prefixes with pointers from s
Repeat {
    p = PQ.pop
    If p is a full path then output(p)
        else expand p and push to PQ
until PQ = ∅}

PQ

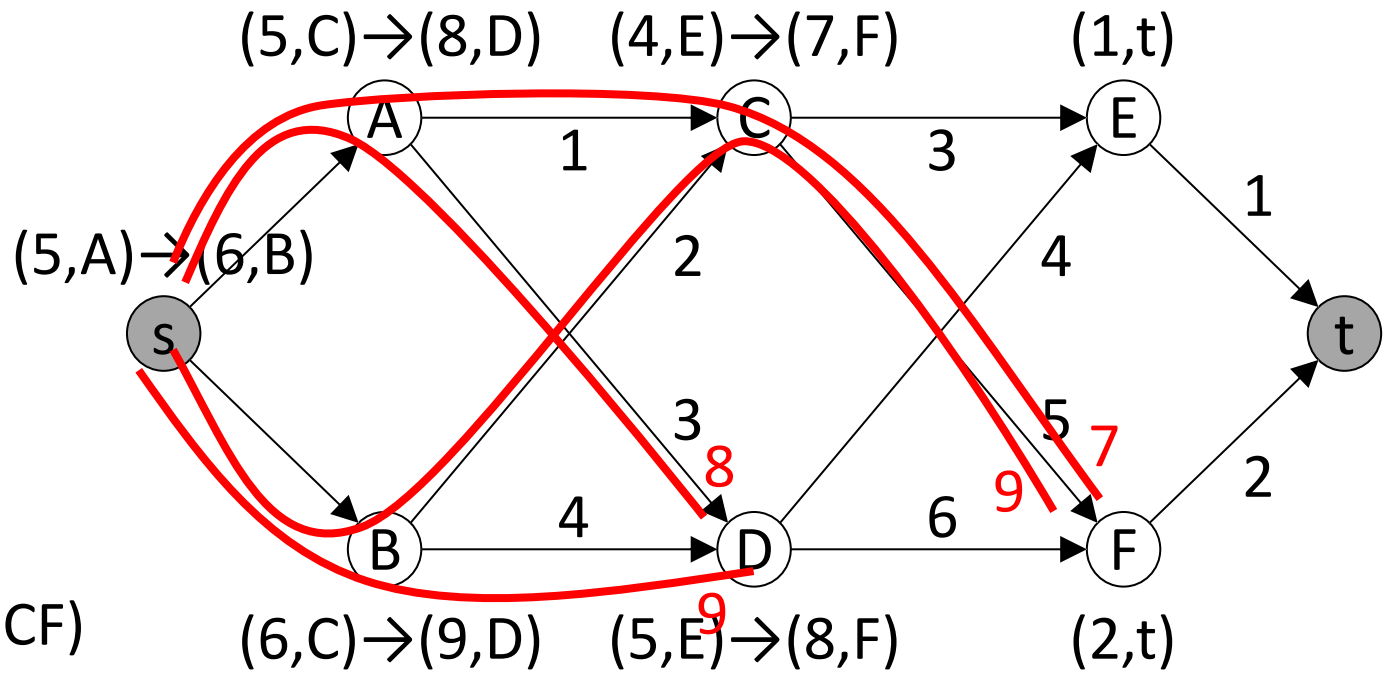
(5,A)→(6,B)
(5,AC)→(6,B)→(8,AD)
(5,ACE)→(6,B)→(7,ACF)→(8,AD)
(6,B)→(7,ACF)→(8,AD)
(6,BC)→(7,ACF)→(8,AD)→(9,BD)
(6,BCE)→(7,ACF)→(8,AD)→(9,BD)→(9,BCF)



Initialize PQ of prefixes with pointers from s
Repeat {
    p = PQ.pop
    If p is a full path then output(p)
        else expand p and push to PQ
until PQ = ∅}

PQ

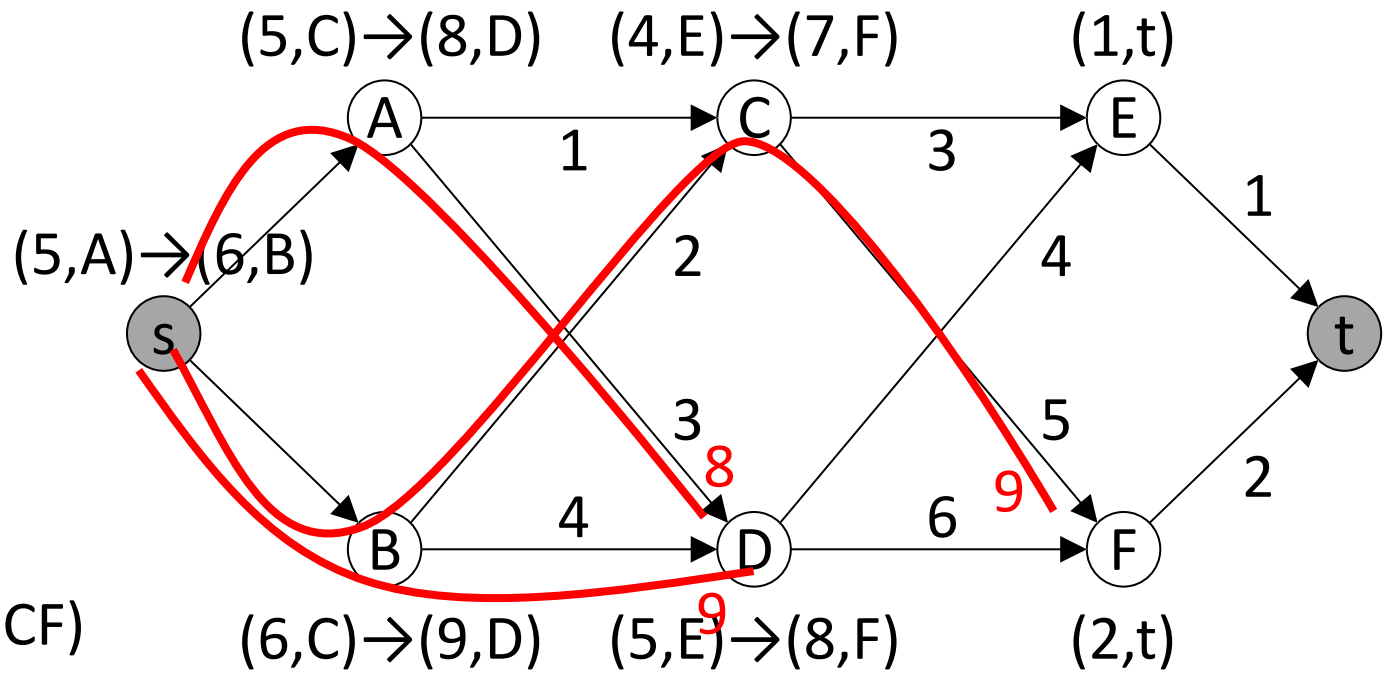
(5,A)→(6,B)
(5,AC)→(6,B)→(8,AD)
(5,ACE)→(6,B)→(7,ACF)→(8,AD)
(6,B)→(7,ACF)→(8,AD)
(6,BC)→(7,ACF)→(8,AD)→(9,BD)
(6,BCE)→(7,ACF)→(8,AD)→(9,BD)→(9,BCF)
(7,ACF)→(8,AD)→(9,BD) →(9,BCF)



Initialize PQ of prefixes with pointers from s
Repeat {
    p = PQ.pop
    If p is a full path then output(p)
        else expand p and push to PQ
until PQ = ∅}

PQ

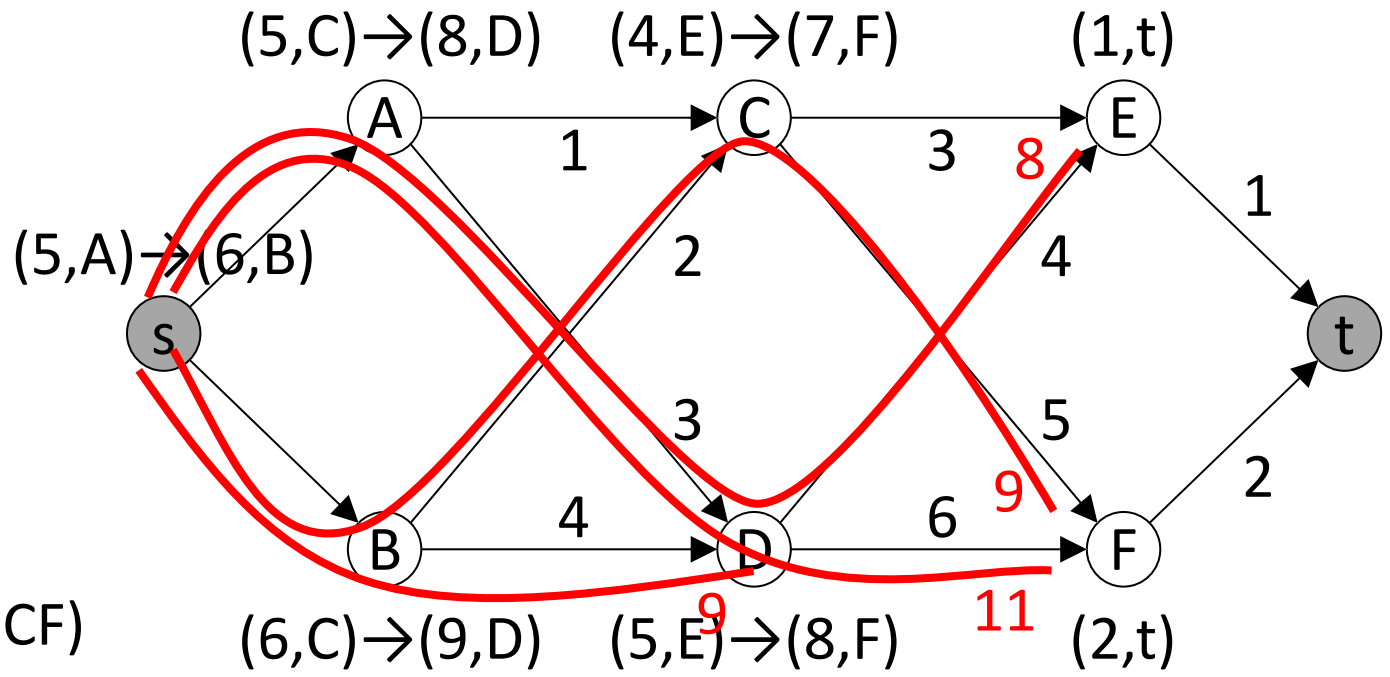
(5,A)→(6,B)
(5,AC)→(6,B)→(8,AD)
(5,ACE)→(6,B)→(7,ACF)→(8,AD)
(6,B)→(7,ACF)→(8,AD)
(6,BC)→(7,ACF)→(8,AD)→(9,BD)
(6,BCE)→(7,ACF)→(8,AD)→(9,BD)→(9,BCF)
(7,ACF)→(8,AD)→(9,BD) →(9,BCF)
(8,AD)→(9,BD)→(9,BCF)



Initialize PQ of prefixes with pointers from s
Repeat {
    p = PQ.pop
    If p is a full path then output(p)
        else expand p and push to PQ
until PQ = ∅}

PQ

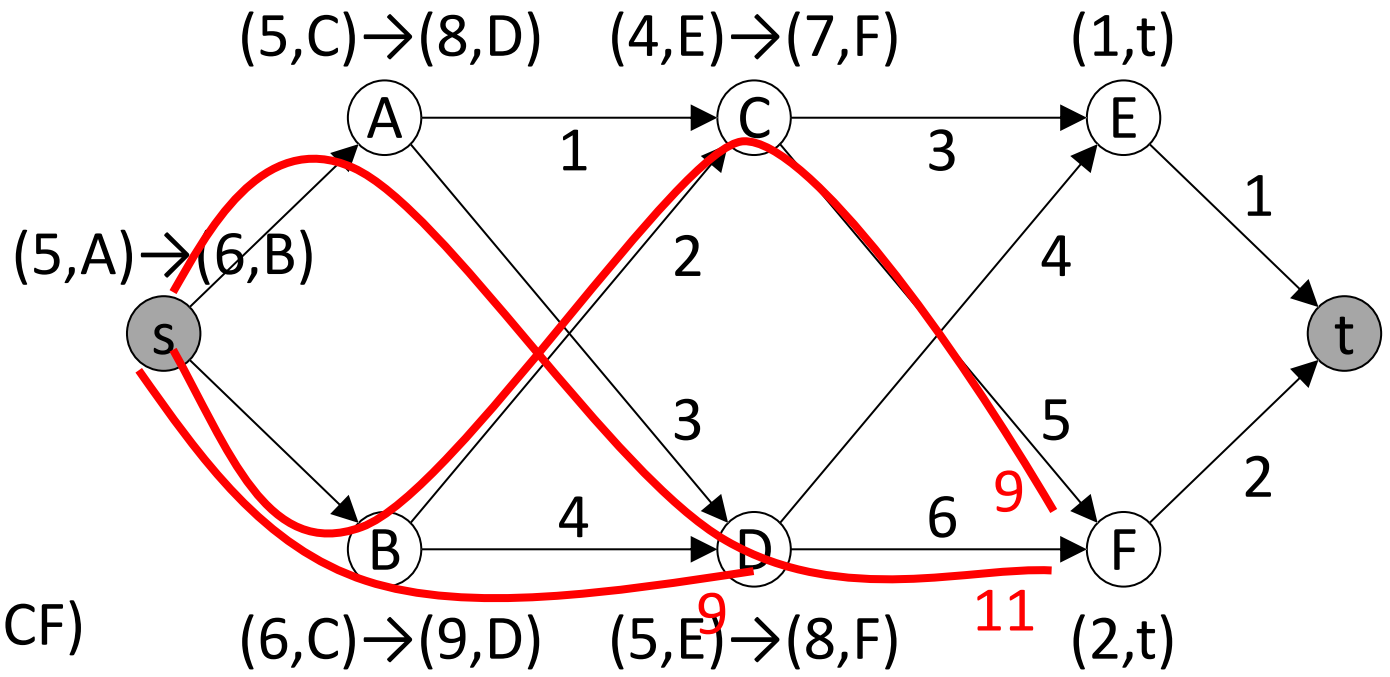
(5,A)→(6,B)
(5,AC)→(6,B)→(8,AD)
(5,ACE)→(6,B)→(7,ACF)→(8,AD)
(6,B)→(7,ACF)→(8,AD)
(6,BC)→(7,ACF)→(8,AD)→(9,BD)
(6,BCE)→(7,ACF)→(8,AD)→(9,BD)→(9,BCF)
(7,ACF)→(8,AD)→(9,BD) →(9,BCF)
(8,AD)→(9,BD)→(9,BCF)
(8,ADE)→(9,BD)→(9,BCF)→(11,ADF)



Initialize PQ of prefixes with pointers from s
Repeat {
    p = PQ.pop
    If p is a full path then output(p)
        else expand p and push to PQ
until PQ = ∅}

PQ

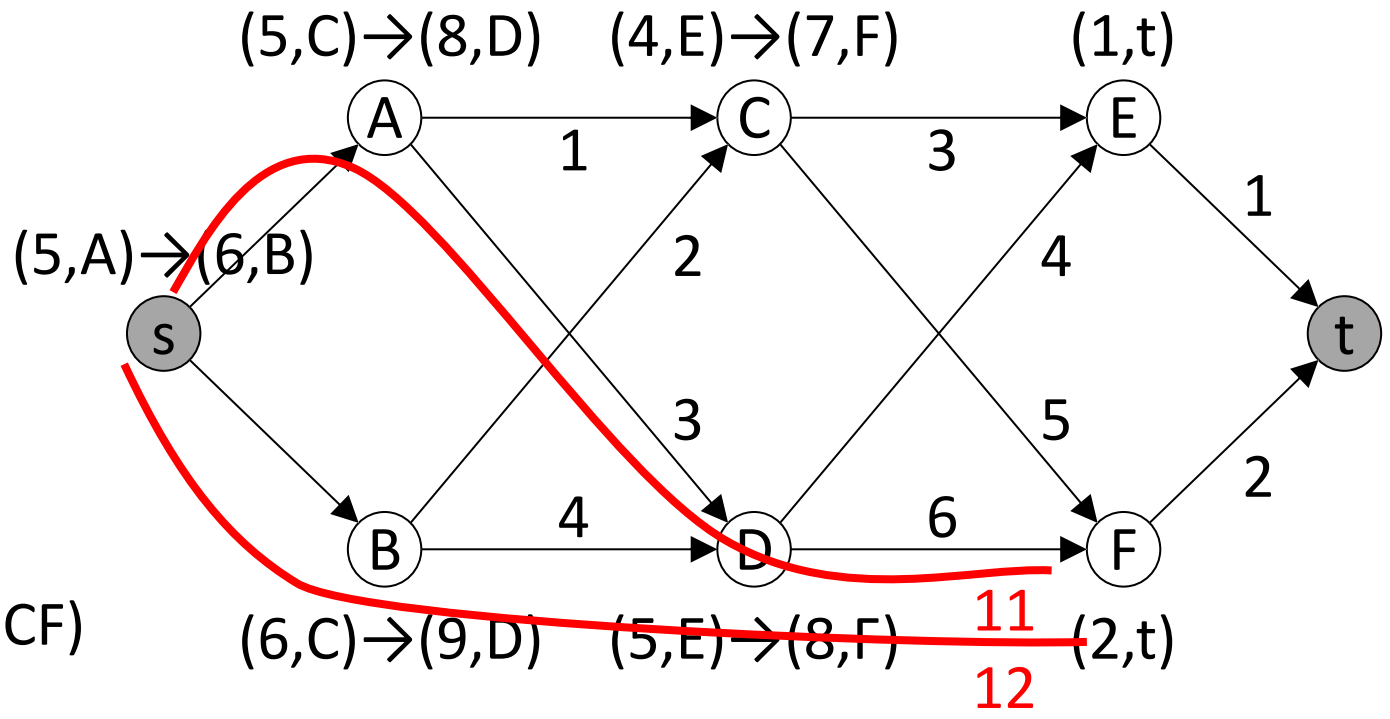
(5,A)→(6,B)
(5,AC)→(6,B)→(8,AD)
(5,ACE)→(6,B)→(7,ACF)→(8,AD)
(6,B)→(7,ACF)→(8,AD)
(6,BC)→(7,ACF)→(8,AD)→(9,BD)
(6,BCE)→(7,ACF)→(8,AD)→(9,BD)→(9,BCF)
(7,ACF)→(8,AD)→(9,BD) →(9,BCF)
(8,AD)→(9,BD)→(9,BCF)
(8,ADE)→(9,BD)→(9,BCF)→(11,ADF)
(9,BD)→(9,BCF)→(11,ADF)



Initialize PQ of prefixes with pointers from s
Repeat {
    p = PQ.pop
    If p is a full path then output(p)
        else expand p and push to PQ
until PQ = ∅}

PQ

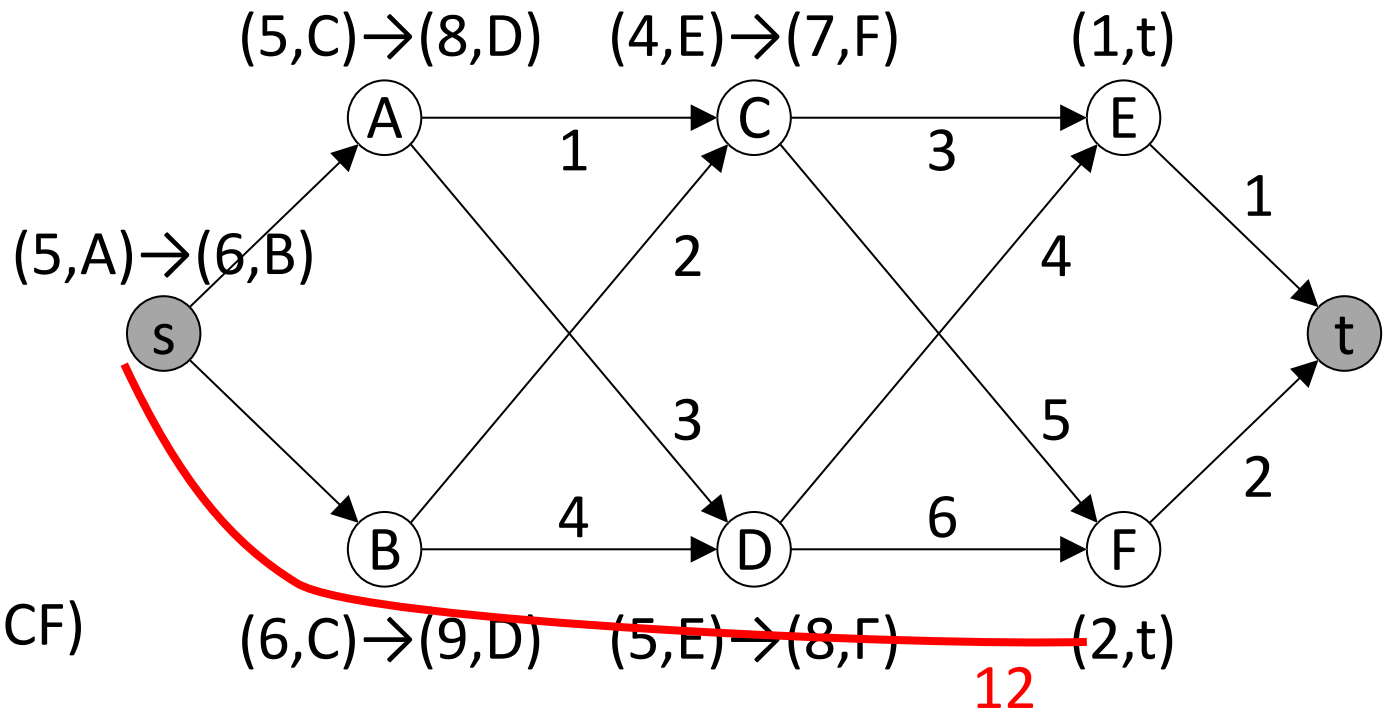
(5,A)→(6,B)
(5,AC)→(6,B)→(8,AD)
(5,ACE)→(6,B)→(7,ACF)→(8,AD)
(6,B)→(7,ACF)→(8,AD)
(6,BC)→(7,ACF)→(8,AD)→(9,BD)
(6,BCE)→(7,ACF)→(8,AD)→(9,BD)→(9,BCF)
(7,ACF)→(8,AD)→(9,BD) →(9,BCF)
(8,AD)→(9,BD)→(9,BCF)
(8,ADE)→(9,BD)→(9,BCF)→(11,ADF)
(9,BD)→(9,BCF)→(11,ADF)
(9,BDE)→(9,BCF)→(11,ADF)→(12,BDF)
(9,BCF)→(11,ADF)→(12,BDF)
(11,ADF)→(12,BDF)



Initialize PQ of prefixes with pointers from s
Repeat {
    p = PQ.pop
    If p is a full path then output(p)
        else expand p and push to PQ
until PQ = ∅}

PQ

(5,A)→(6,B)
(5,AC)→(6,B)→(8,AD)
(5,ACE)→(6,B)→(7,ACF)→(8,AD)
(6,B)→(7,ACF)→(8,AD)
(6,BC)→(7,ACF)→(8,AD)→(9,BD)
(6,BCE)→(7,ACF)→(8,AD)→(9,BD)→(9,BCF)
(7,ACF)→(8,AD)→(9,BD) →(9,BCF)
(8,AD)→(9,BD)→(9,BCF)
(8,ADE)→(9,BD)→(9,BCF)→(11,ADF)
(9,BD)→(9,BCF)→(11,ADF)
(9,BDE)→(9,BCF)→(11,ADF)→(12,BDF)
(9,BCF)→(11,ADF)→(12,BDF)
(11,ADF)→(12,BDF)
(12,BDF)



Initialize PQ of prefixes with pointers from s
Repeat {
    p = PQ.pop
    If p is a full path then output(p)
        else expand p and push to PQ
until PQ = ∅}