ISyE/Math/CS 425
Introduction to Combinatorial Optimization

2.2 Shortest Paths

Prof. Carla Michini
University of Wisconsin-Madison

Outline

- We will see the Minimum Driving Distances Problem and the Shortest (Di)Path Problem.
- We will study Ford's algorithm and the Ford-Bellman algorithm for solving them.
- We will study more efficient algorithms for important special cases:
  - Acyclic digraphs,
  - Nonnegative costs,
  - Unit costs.

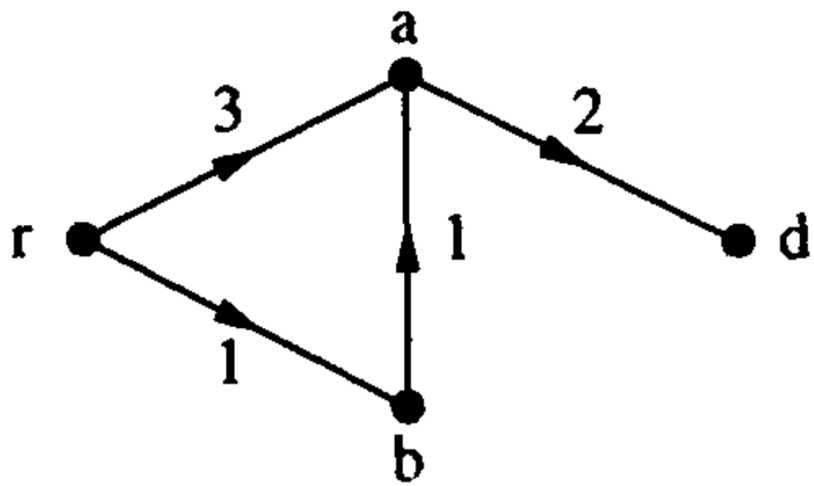
2.2 Shortest Paths

The Minimum Driving Distances Problem

- We want to make a table of the minimum driving distances from the corner of Walnut St and Pacific Ave to every other street corner in the city of San Francisco, CA.

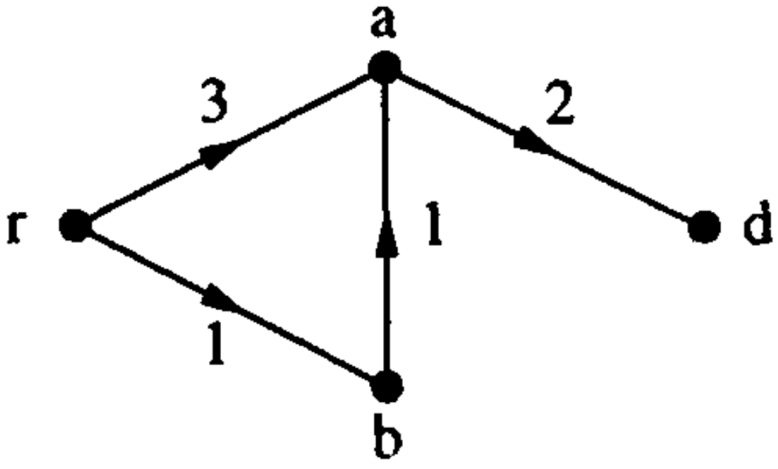
The Minimum Driving Distances Problem

- The routes must obey the directions on one-way streets.
- We need a graph where each edge has a direction.



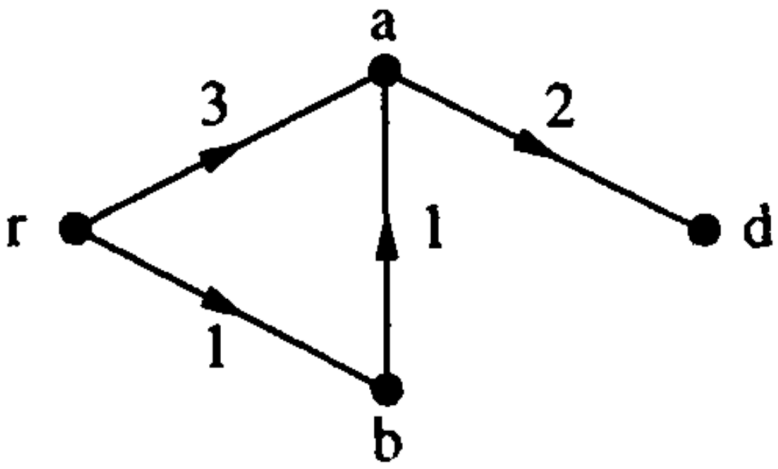
- If a street can be traversed in both directions, we use two "parallel" edges with opposite directions.

Digraphs



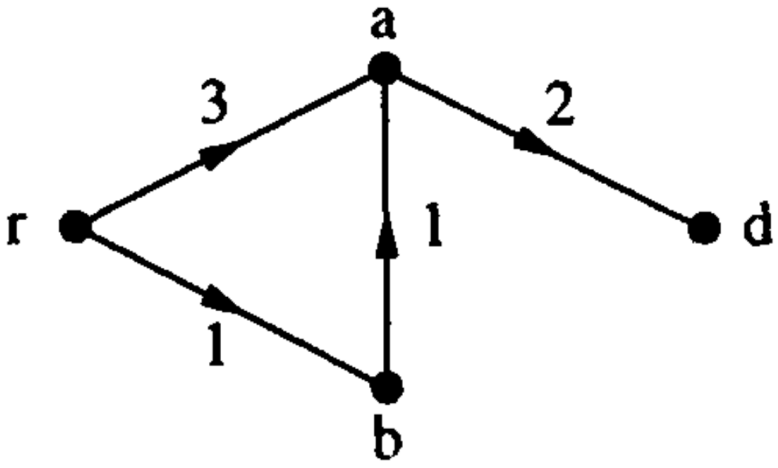
- A digraph $G = (V, E)$ consists of a set $V = V(G)$ of nodes, and $E = E(G)$ of arcs.
- Each arc consists of an ordered pair of distinct nodes.
- Given an arc $e = vw$, v is the tail $t(e) = v$, and w is the head $h(e) = w$.

Digraphs



- A path $P = v_0, e_1, v_1, \ldots, e_k, v_k$ is a sequence of nodes and arcs where $e_i = v_{i-1}v_i$, or $e_i = v_iv_{i-1}$.
- An arc e_i of path P is forward if $e_i = v_{i-1}v_i$ and is reverse otherwise.
- A dipath is a path in which every arc is forward.
- A dicircuit is a dipath that is also a circuit in the underlying graph.
- The cost of dipath P is $c(P) = \sum_{i=1}^{k} c_{e_i}$, where c_e is the cost of arc e.

Digraphs



- The street map of San Francisco defines a digraph:
  - The nodes are the street corners.
  - For each section of street joining two corners, and each direction in which it is legal to drive along it, there is an arc.

Digraphs

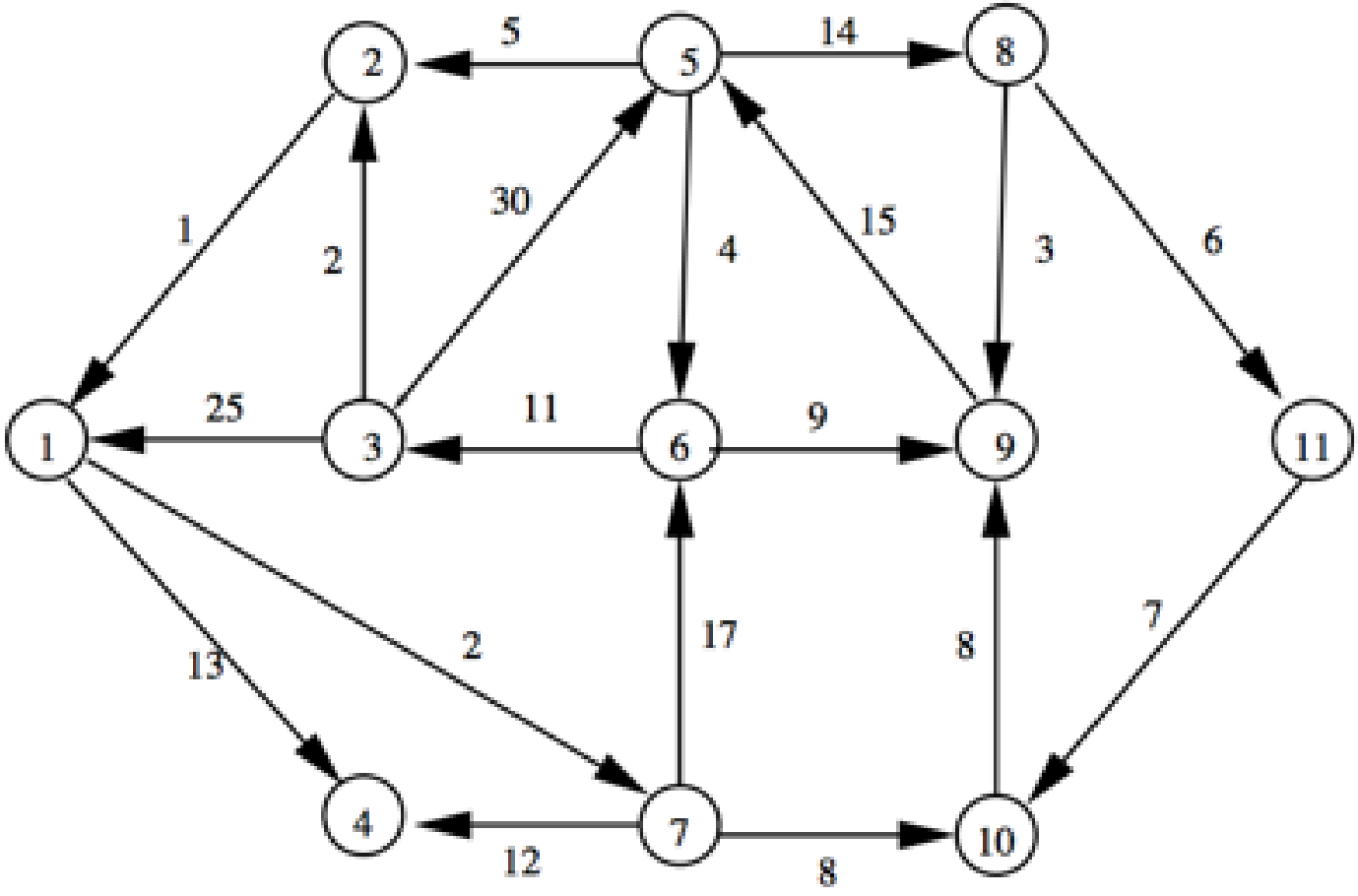
The Minimum Driving Distances Problem can be formulated as:

Shortest (Di)Path Problem
Input: A digraph G, a node $r \in V$, and costs c_e, $\forall e \in E$.
Objective: For each $v \in V$, find a dipath from r to v of least cost (if one exists).

- By modifying G, we can assume that dipaths exist from r to all the nodes. Why?

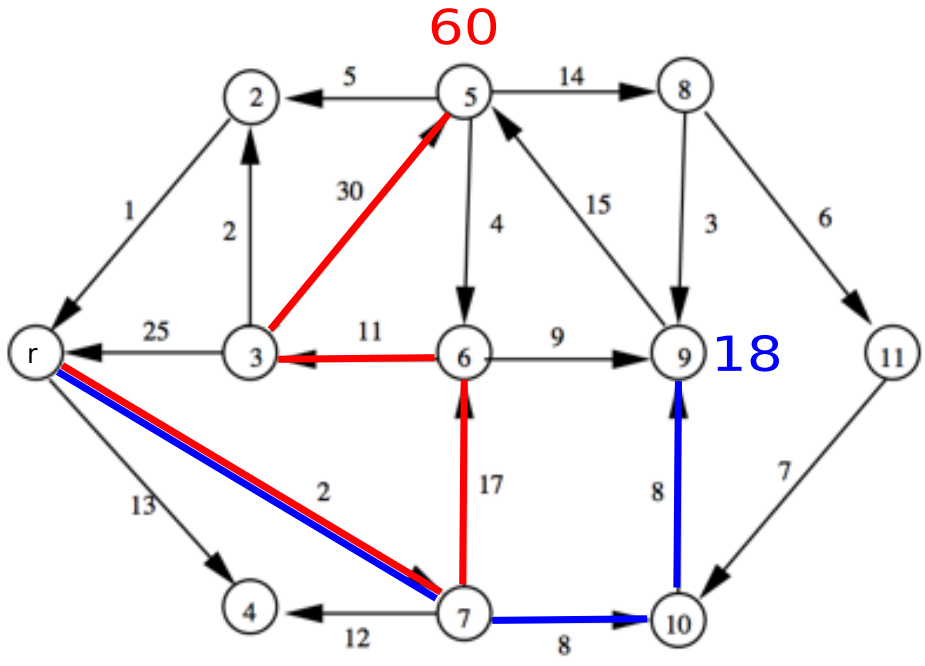
Exercise

Let's try to solve a Shortest Path problem!



Idea for algorithm:

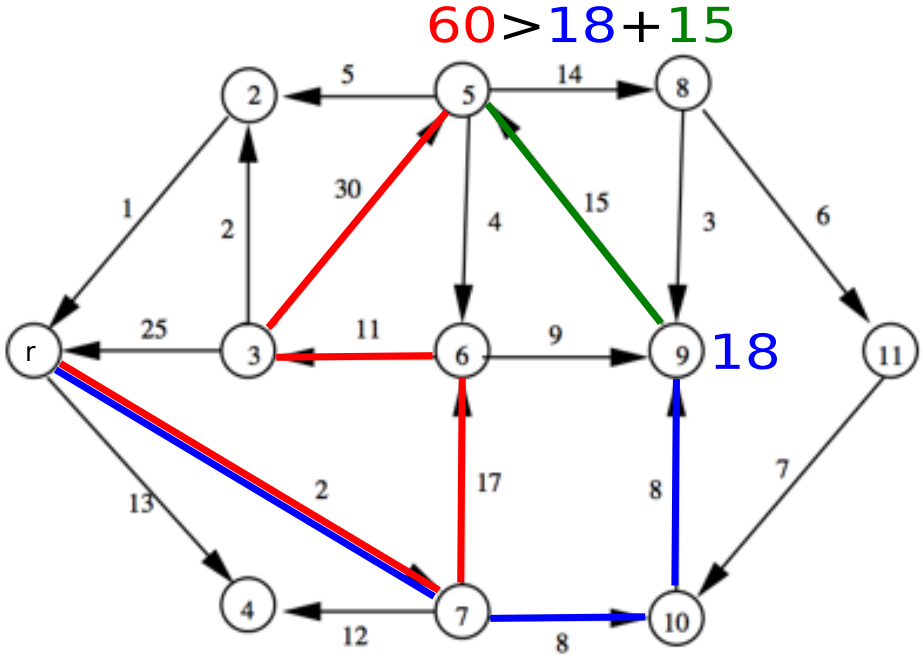
- Suppose we have a dipath P from r to v of cost $c(P)$.
- Suppose we have a dipath P' from r to w of cost $c(P')$.
- Suppose we find an arc $vw \in E$ satisfying $c(P') > c(P) + c_{vw}$.
- By appending vw to the dipath P, we get a cheaper dipath to w of cost $c(P) + c_{vw}$.



- If P' is a shortest path from r to w this cannot happen!

Idea for algorithm:

- Suppose we have a dipath P from r to v of cost $c(P)$.
- Suppose we have a dipath P' from r to w of cost $c(P')$.
- Suppose we find an arc $vw \in E$ satisfying $c(P') > c(P) + c_{vw}$.
- By appending vw to the dipath P, we get a cheaper dipath to w of cost $c(P) + c_{vw}$.



- If P' is a shortest path from r to w this cannot happen!

Feasible potential

- Let $y \in \mathbb{R}^V$ be a vector such that each entry of y corresponds to a node in V.
- A vector $y \in \mathbb{R}^V$ that satisfies

$$\begin{aligned} y_v + c_{vw} &\geq y_w \qquad \forall vw \in E \\ y_r &= 0 \end{aligned} \tag{*}$$

  is called feasible potential.
- If y^* is such that y^*_v is the least cost of a dipath to v, $\forall v \in V$, then y^* satisfies (*). Thus y^* is a feasible potential.

Feasible potential

Viceversa:

Proposition 2.9
Let y be a feasible potential and let P be a dipath from r to v. Then $c(P) \geq y_v$.

Let's prove it!

Structure of optimal solutions

- Subpaths of shortest dipaths are shortest dipaths. Why?
- Hence, for every node $v \neq r$, the only arc we need is the last arc of one least-cost dipath to v.
- There are $n-1$ nodes $v \neq r$, thus we collect $n-1$ such arcs in total. The subgraph induced by these arcs contains a dipath from r to every other node.
- So this is the arc-set of a "directed" spanning tree of G.

Ford's algorithm

- The stopping condition is provided by Proposition 2.9:
  If we have $\forall v \in V$ a feasible potential y_v and a dipath from r to v of cost y_v, then each dipath is of least cost.
- We will find a feasible potential y, and the predecessor $p(v)$ of v for each $v \in V$.
- Given a vector y, an arc vw is incorrect if

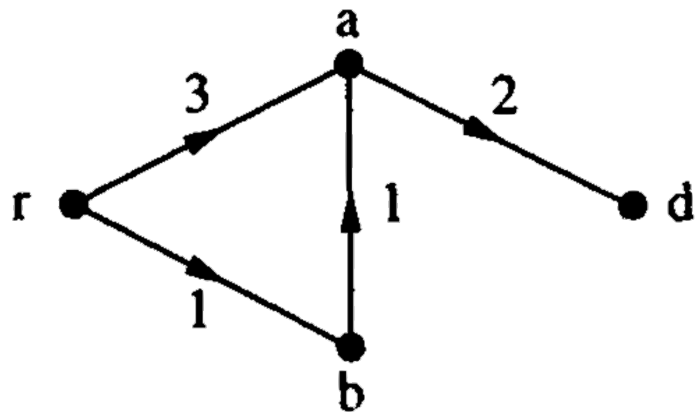
$$y_v + c_{vw} < y_w.$$

- To correct vw means to set

$$\begin{aligned} y_w &= y_v + c_{vw} \\ p(w) &= v. \end{aligned}$$

Ford's algorithm

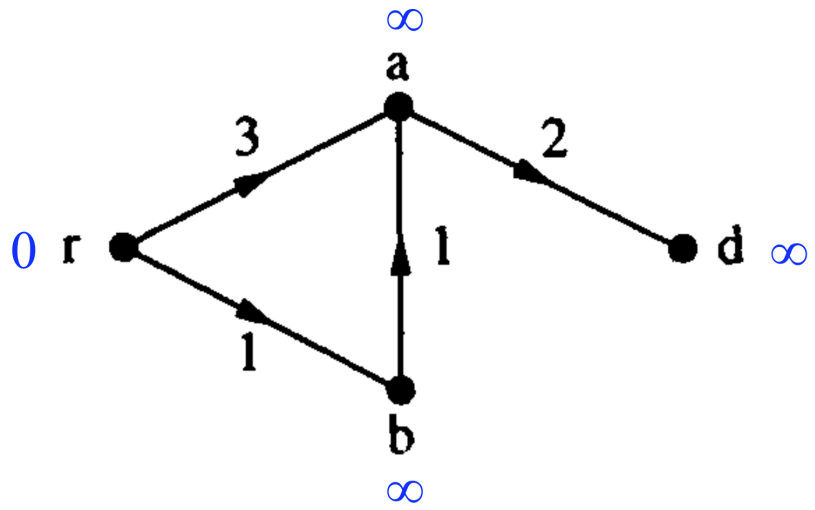
1. Initialize $y_r = 0$, $p(r) = 0$ and $y_v = \infty$, $p(v) = -1$ for every $v \neq r$
2. While y is not a feasible potential
   Find an incorrect arc vw and correct it



Ford's algorithm

1. Initialize $y_r = 0$, $p(r) = 0$ and $y_v = \infty$, $p(v) = -1$ for every $v \neq r$
2. While y is not a feasible potential
   Find an incorrect arc vw and correct it

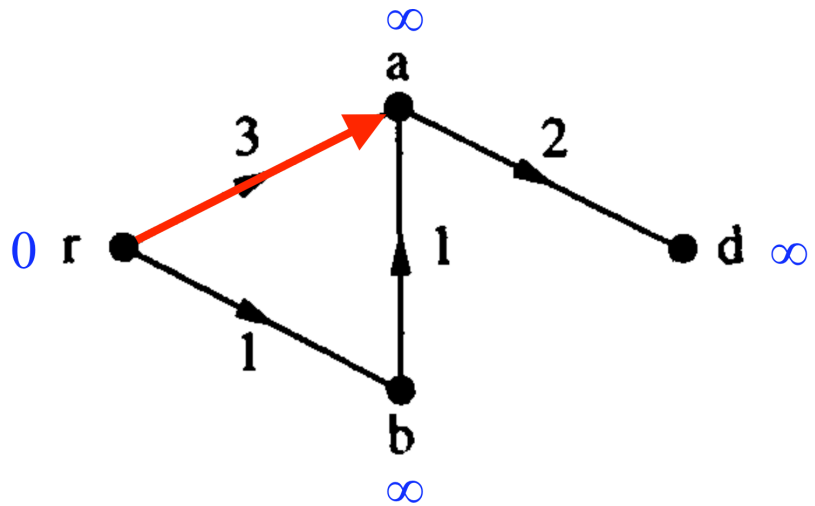
Initialization



Ford's algorithm

1. Initialize $y_r = 0$, $p(r) = 0$ and $y_v = \infty$, $p(v) = -1$ for every $v \neq r$
2. While y is not a feasible potential
   Find an incorrect arc vw and correct it

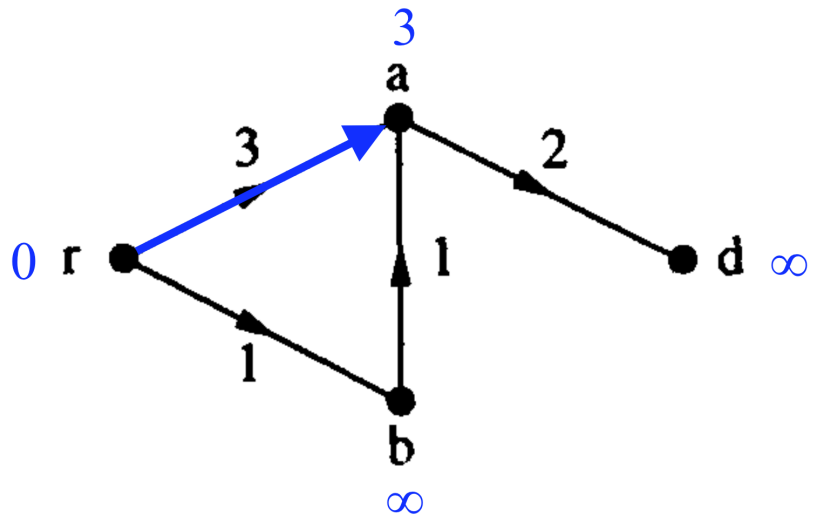
Iteration 1



Ford's algorithm

1. Initialize $y_r = 0$, $p(r) = 0$ and $y_v = \infty$, $p(v) = -1$ for every $v \neq r$
2. While y is not a feasible potential
   Find an incorrect arc vw and correct it

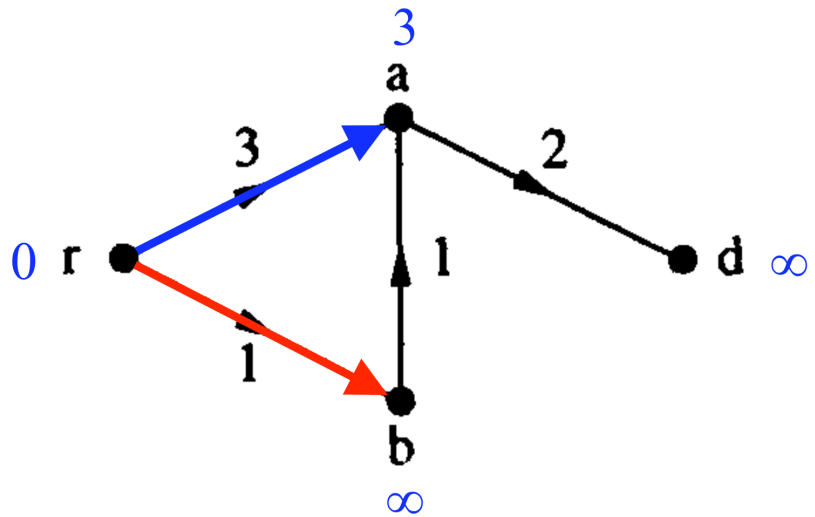
Iteration 1



Ford's algorithm

1. Initialize $y_r = 0$, $p(r) = 0$ and $y_v = \infty$, $p(v) = -1$ for every $v \neq r$
2. While y is not a feasible potential
   Find an incorrect arc vw and correct it

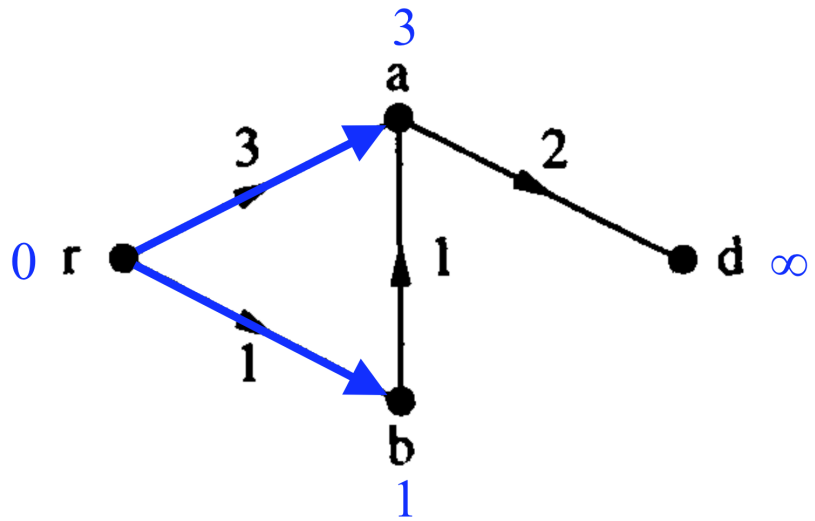
Iteration 2



Ford's algorithm

1. Initialize $y_r = 0$, $p(r) = 0$ and $y_v = \infty$, $p(v) = -1$ for every $v \neq r$
2. While y is not a feasible potential
   Find an incorrect arc vw and correct it

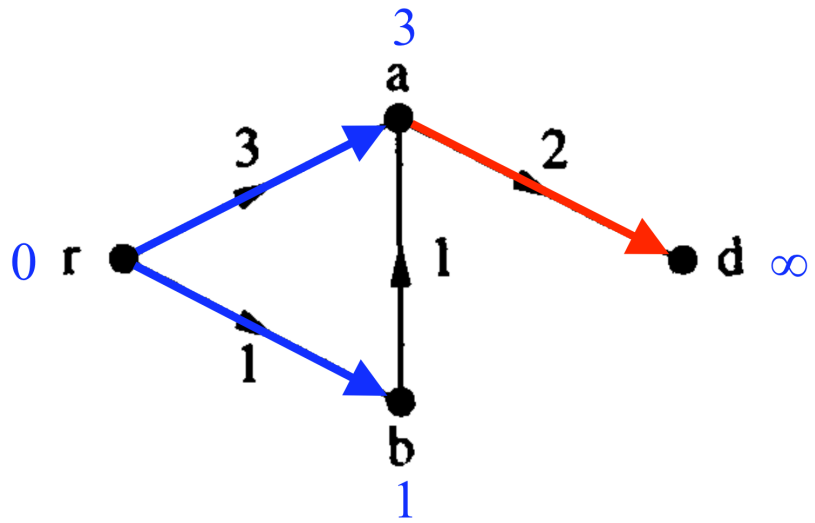
Iteration 2



Ford's algorithm

1. Initialize $y_r = 0$, $p(r) = 0$ and $y_v = \infty$, $p(v) = -1$ for every $v \neq r$
2. While y is not a feasible potential
   Find an incorrect arc vw and correct it

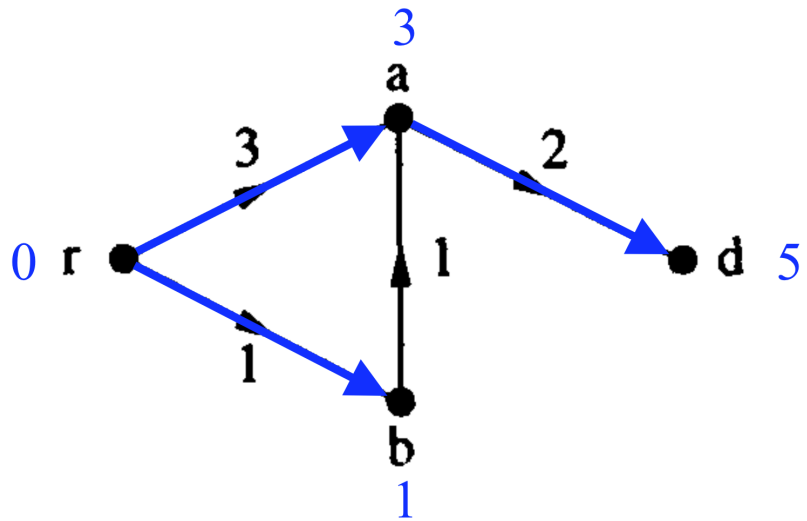
Iteration 3



Ford's algorithm

1. Initialize $y_r = 0$, $p(r) = 0$ and $y_v = \infty$, $p(v) = -1$ for every $v \neq r$
2. While y is not a feasible potential
   Find an incorrect arc vw and correct it

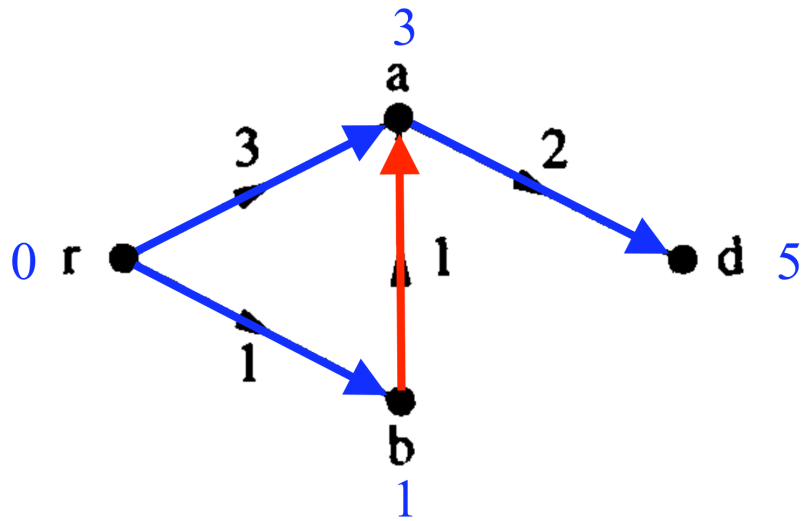
Iteration 3



Ford's algorithm

1. Initialize $y_r = 0$, $p(r) = 0$ and $y_v = \infty$, $p(v) = -1$ for every $v \neq r$
2. While y is not a feasible potential
   Find an incorrect arc vw and correct it

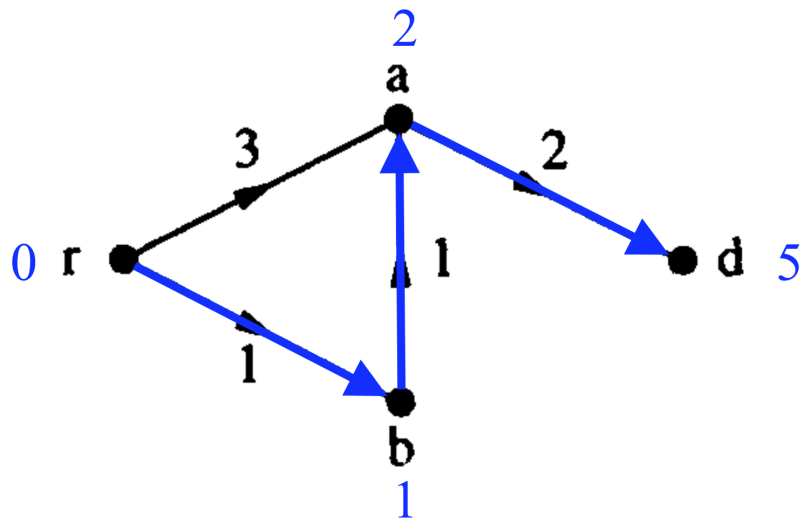
Iteration 4



Ford's algorithm

1. Initialize $y_r = 0$, $p(r) = 0$ and $y_v = \infty$, $p(v) = -1$ for every $v \neq r$
2. While y is not a feasible potential
   Find an incorrect arc vw and correct it

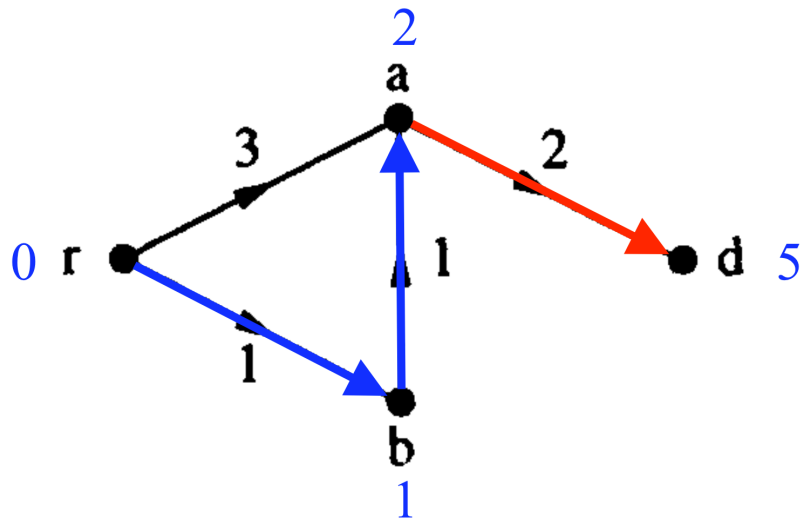
Iteration 4



Ford's algorithm

1. Initialize $y_r = 0$, $p(r) = 0$ and $y_v = \infty$, $p(v) = -1$ for every $v \neq r$
2. While y is not a feasible potential
   Find an incorrect arc vw and correct it

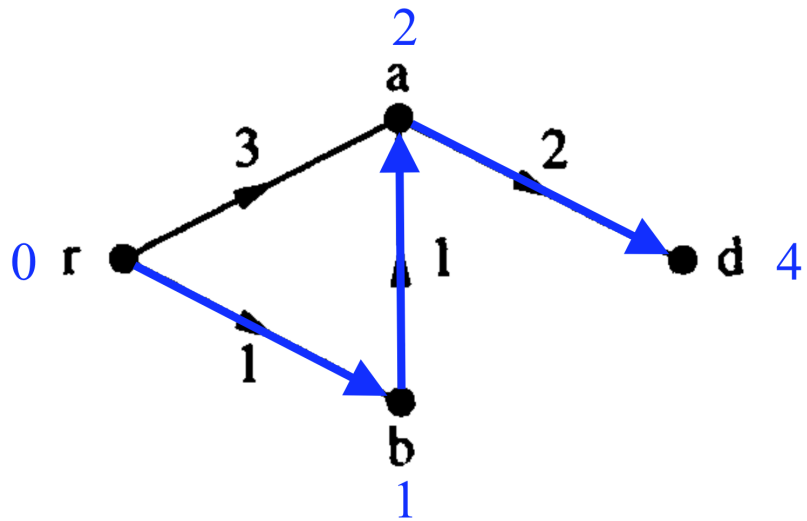
Iteration 5



Ford's algorithm

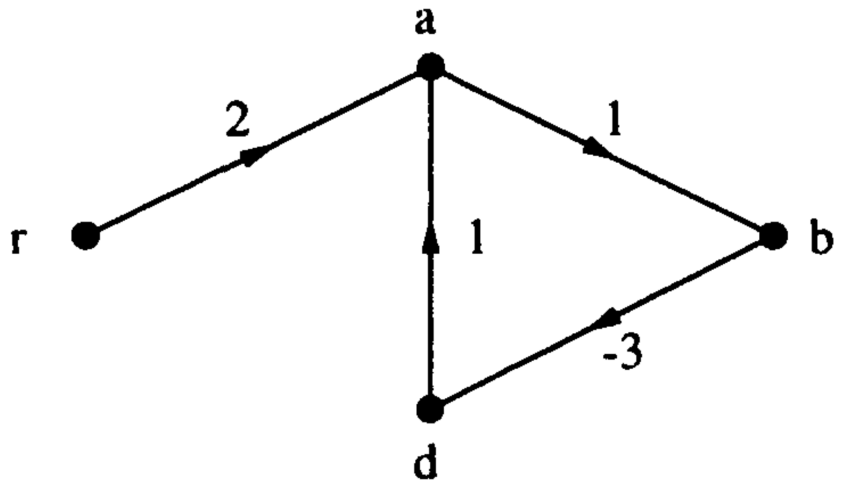
1. Initialize $y_r = 0$, $p(r) = 0$ and $y_v = \infty$, $p(v) = -1$ for every $v \neq r$
2. While y is not a feasible potential
   Find an incorrect arc vw and correct it

Iteration 5



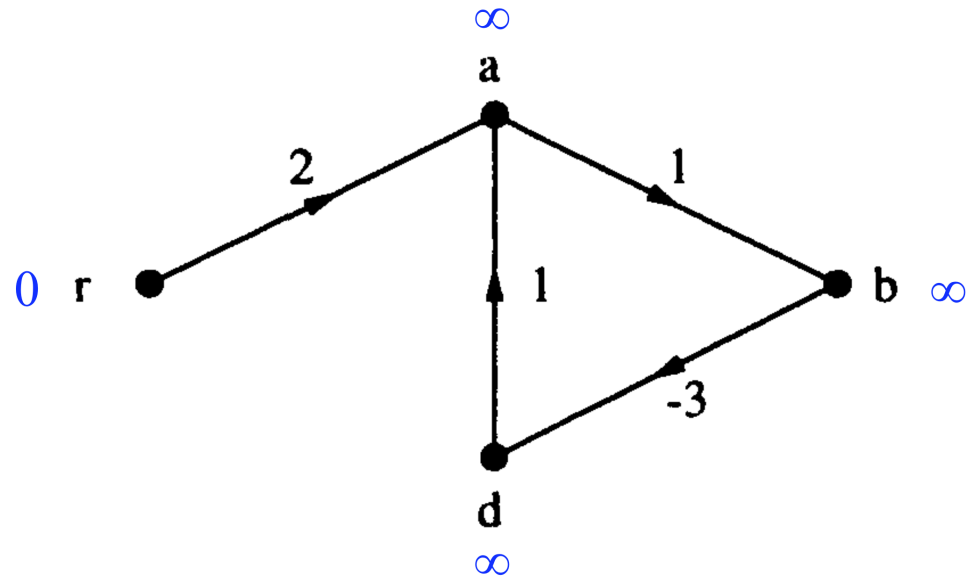
Negative-cost dicircuits

- If we have a negative-cost dicircuit the algorithm does not terminate.



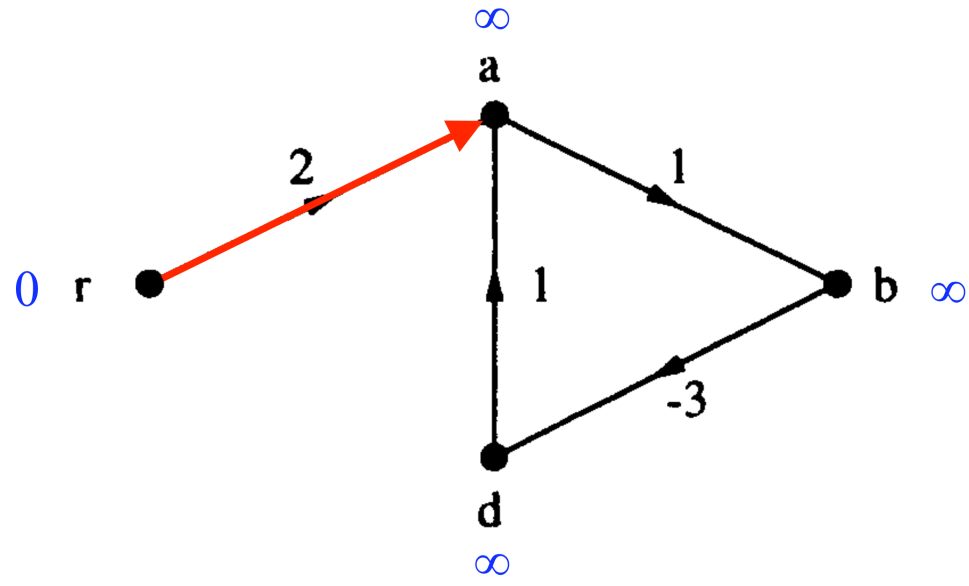
Negative-cost dicircuits

- If we have a negative-cost dicircuit the algorithm does not terminate.



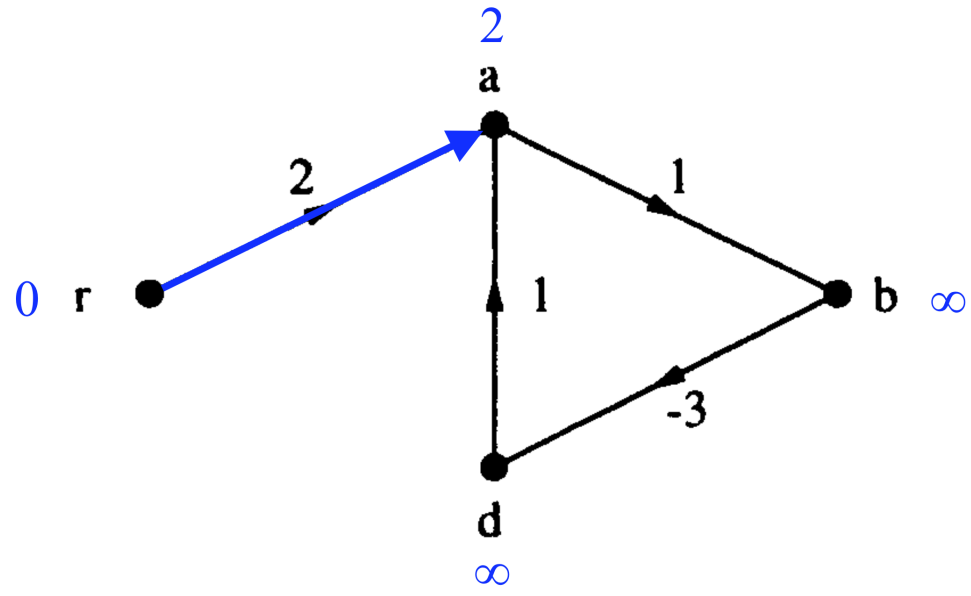
Negative-cost dicircuits

- If we have a negative-cost dicircuit the algorithm does not terminate.



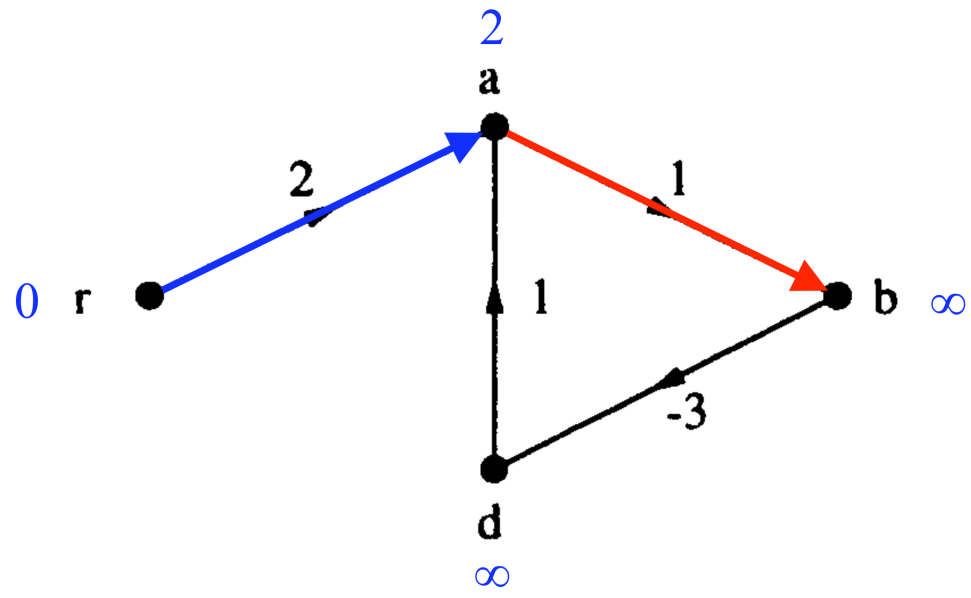
Negative-cost dicircuits

- If we have a negative-cost dicircuit the algorithm does not terminate.



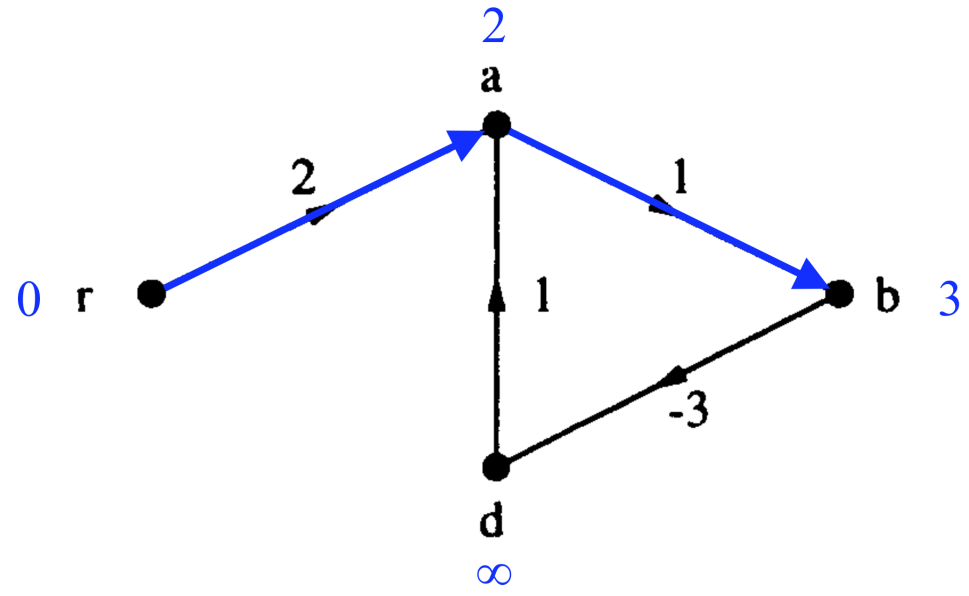
Negative-cost dicircuits

- If we have a negative-cost dicircuit the algorithm does not terminate.



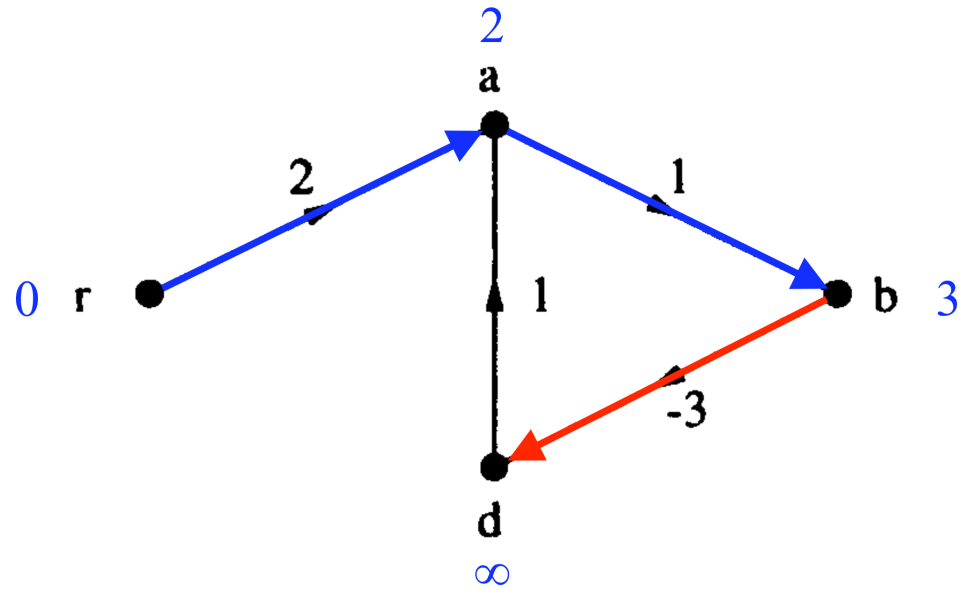
Negative-cost dicircuits

- If we have a negative-cost dicircuit the algorithm does not terminate.



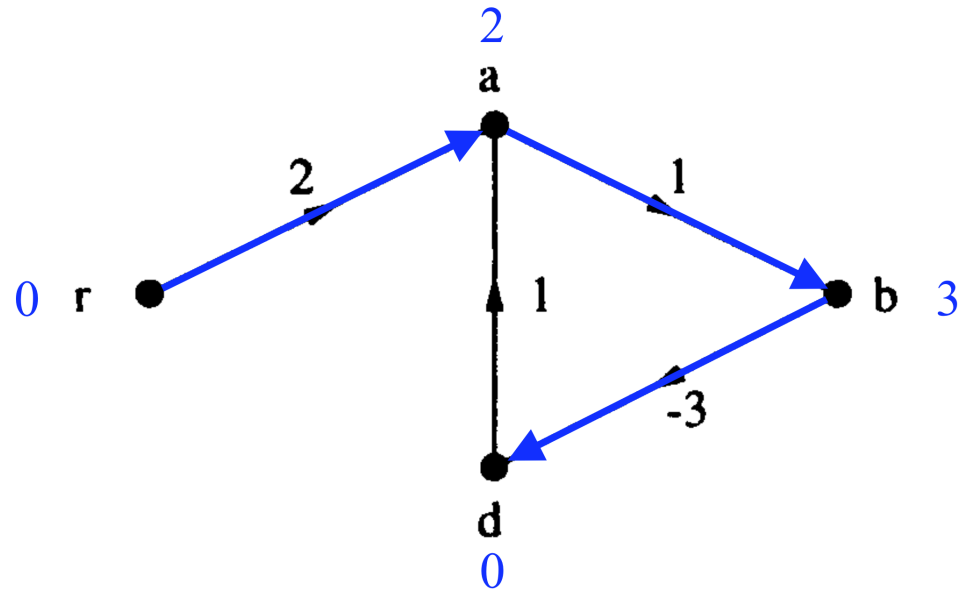
Negative-cost dicircuits

- If we have a negative-cost dicircuit the algorithm does not terminate.



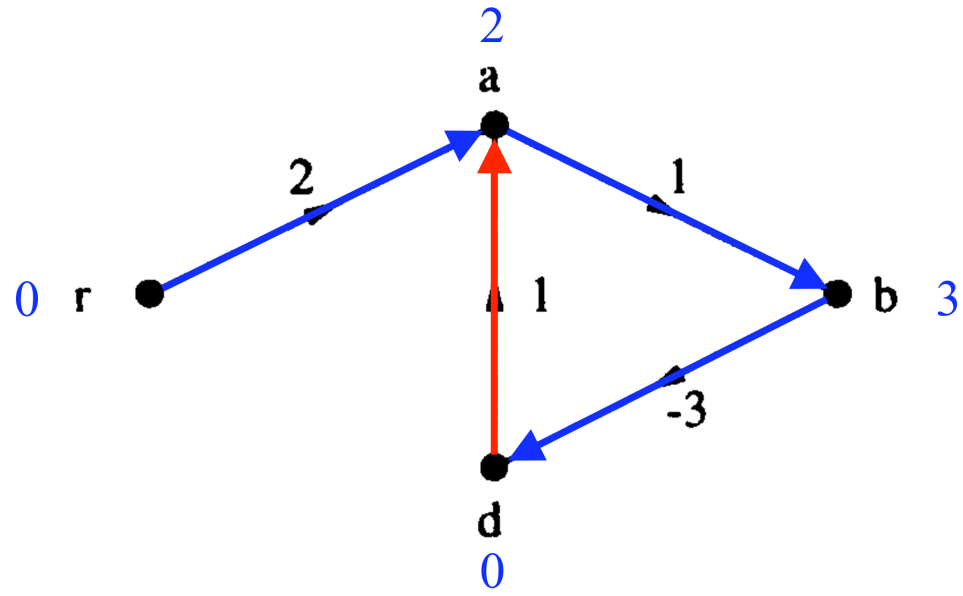
Negative-cost dicircuits

- If we have a negative-cost dicircuit the algorithm does not terminate.



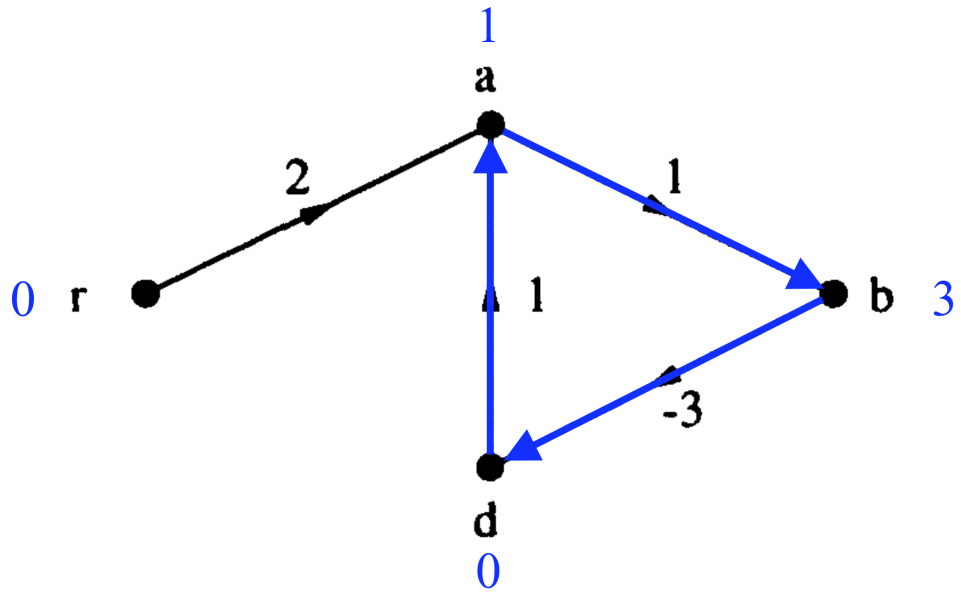
Negative-cost dicircuits

- If we have a negative-cost dicircuit the algorithm does not terminate.



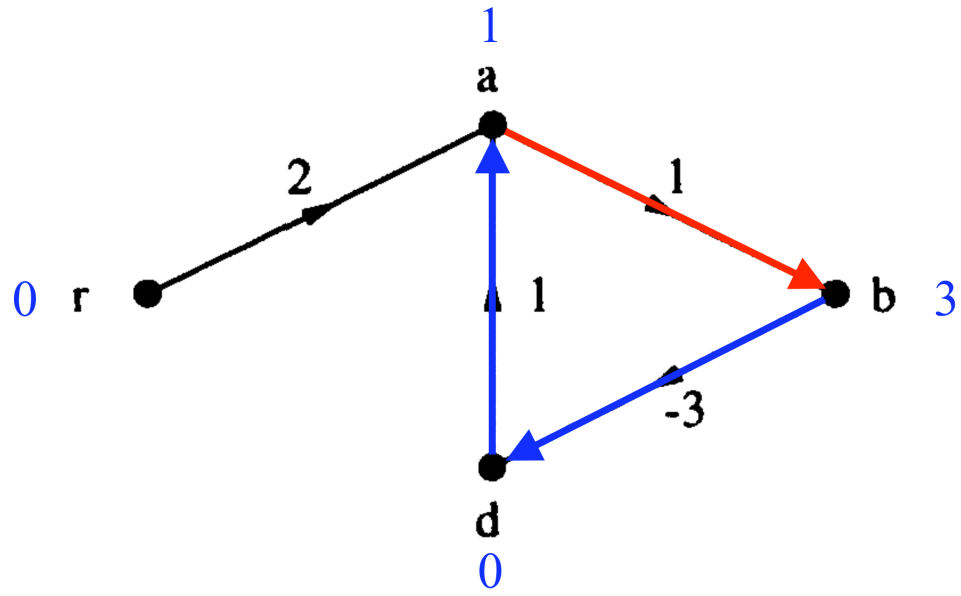
Negative-cost dicircuits

- If we have a negative-cost dicircuit the algorithm does not terminate.



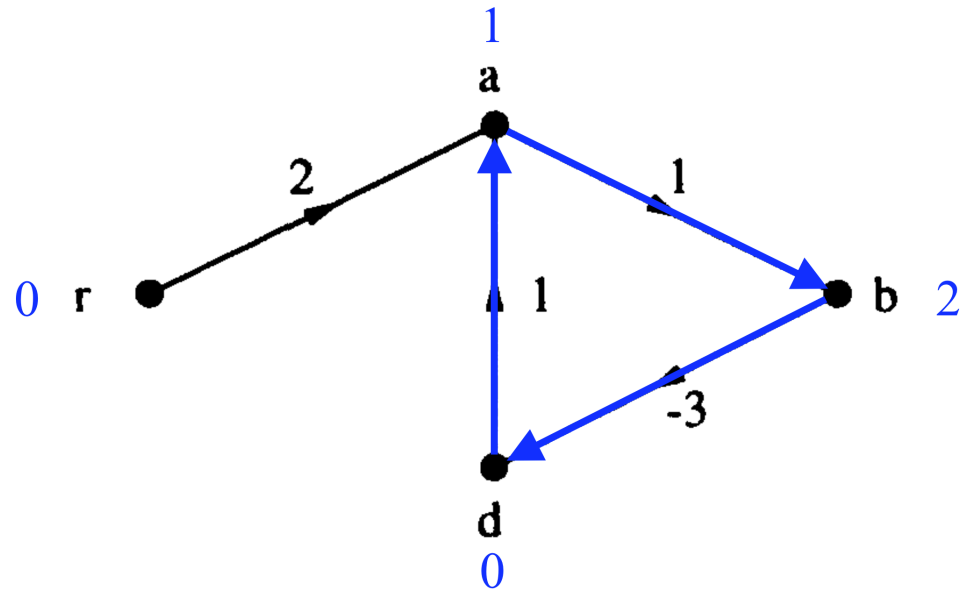
Negative-cost dicircuits

- If we have a negative-cost dicircuit the algorithm does not terminate.



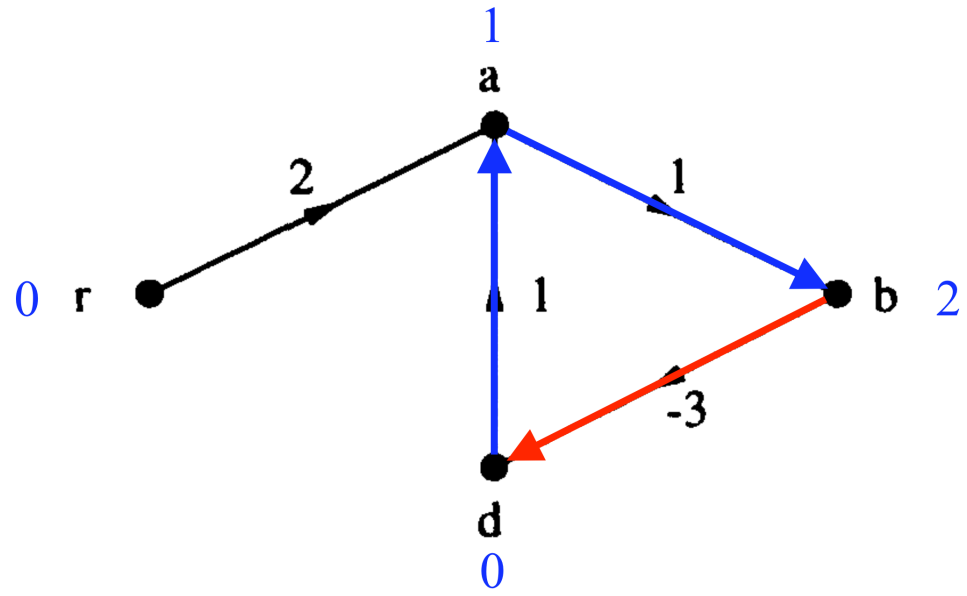
Negative-cost dicircuits

- If we have a negative-cost dicircuit the algorithm does not terminate.



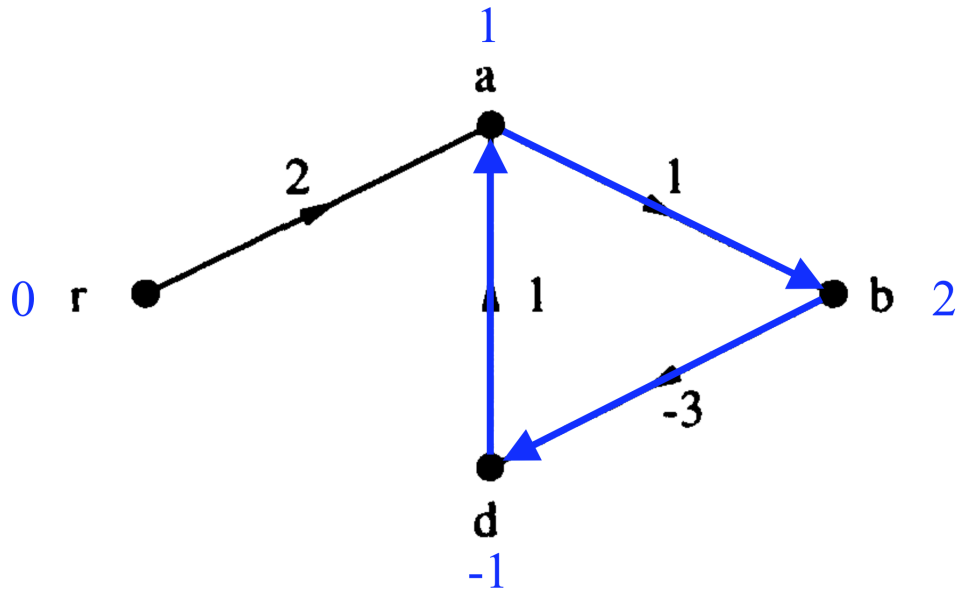
Negative-cost dicircuits

- If we have a negative-cost dicircuit the algorithm does not terminate.



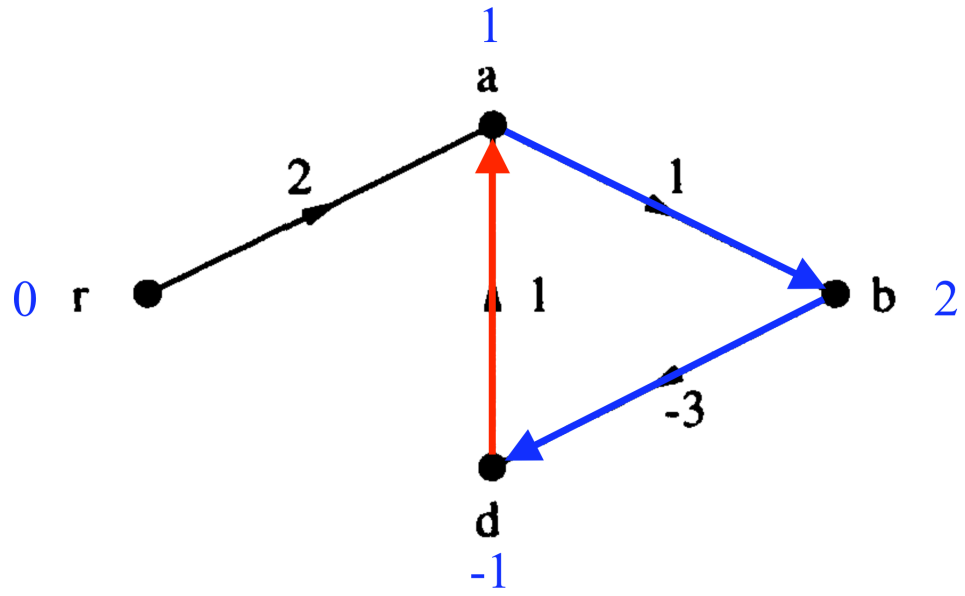
Negative-cost dicircuits

- If we have a negative-cost dicircuit the algorithm does not terminate.



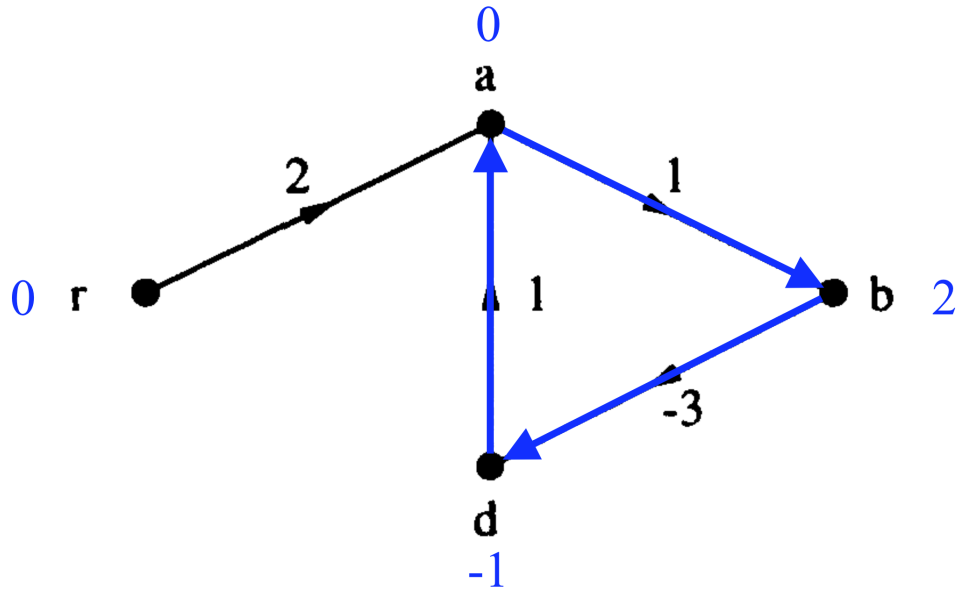
Negative-cost dicircuits

- If we have a negative-cost dicircuit the algorithm does not terminate.



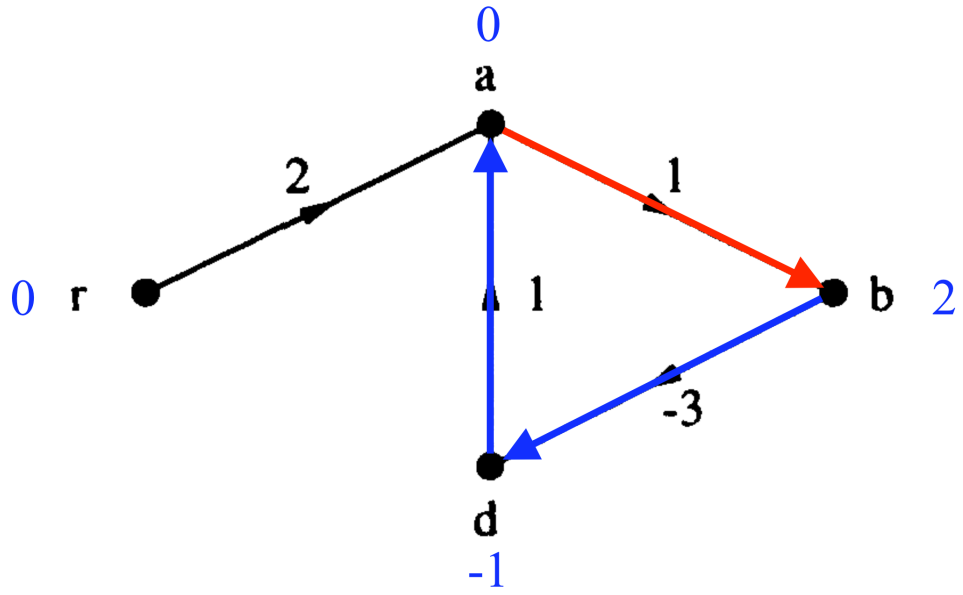
Negative-cost dicircuits

- If we have a negative-cost dicircuit the algorithm does not terminate.



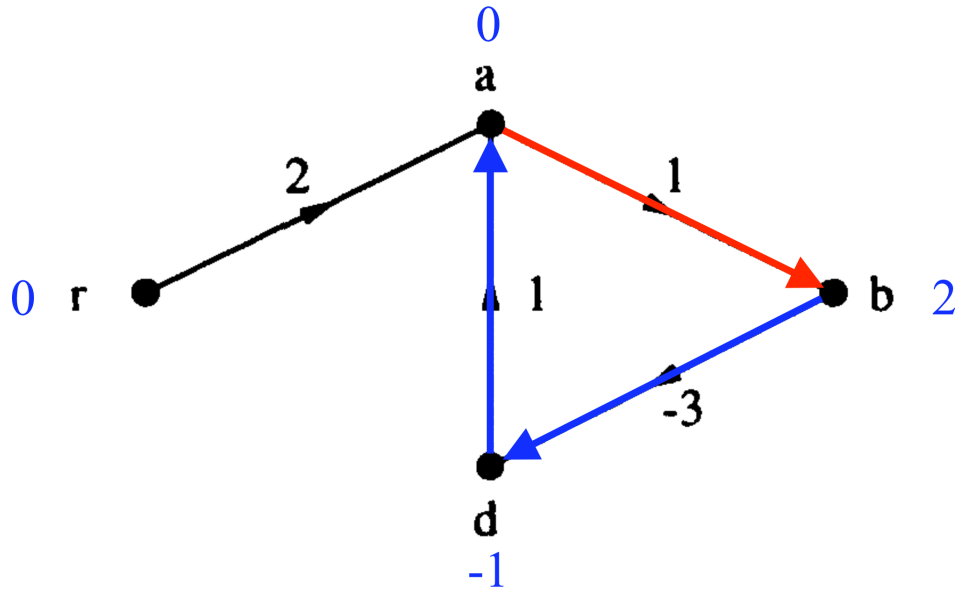
Negative-cost dicircuits

- If we have a negative-cost dicircuit the algorithm does not terminate.



Negative-cost dicircuits

- If we have a negative-cost dicircuit the algorithm does not terminate.



- We will see algorithms that also recognize when a negative-cost dicircuit exists.
- In several applications one searches for negative-cost dicircuits.

Negative-cost dicircuits: Example

Money-making sequence of currency exchanges.

- Let r_{vw} be the amount of currency w that can be purchased with one unit of currency v.
- A sequence $v_0, v_1, \ldots, v_k = v_0$ of currency exchanges is money-making if and only if

$$\prod_{i=1}^{k} r_{v_{i-1}v_i} > 1$$

Negative-cost dicircuits: Example

Money-making sequence of currency exchanges.

- Let r_{vw} be the amount of currency w that can be purchased with one unit of currency v.
- A sequence $v_0, v_1, \ldots, v_k = v_0$ of currency exchanges is money-making if and only if

$$\prod_{i=1}^{k} r_{v_{i-1}v_i} > 1 \quad \Leftrightarrow \quad \log(\prod_{i=1}^{k} r_{v_{i-1}v_i}) > 0$$

$$\Leftrightarrow \quad \sum_{i=1}^{k} \log r_{v_{i-1}v_i} > 0 \quad \Leftrightarrow \quad \sum_{i=1}^{k} (-\log r_{v_{i-1}v_i}) < 0.$$

Negative-cost dicircuits: Example

Money-making sequence of currency exchanges.

- Let r_{vw} be the amount of currency w that can be purchased with one unit of currency v.
- A sequence $v_0, v_1, \ldots, v_k = v_0$ of currency exchanges is money-making if and only if

$$\prod_{i=1}^{k} r_{v_{i-1}v_i} > 1 \quad \Leftrightarrow \quad \log(\prod_{i=1}^{k} r_{v_{i-1}v_i}) > 0$$

$$\Leftrightarrow \quad \sum_{i=1}^{k} \log r_{v_{i-1}v_i} > 0 \quad \Leftrightarrow \quad \sum_{i=1}^{k} (-\log r_{v_{i-1}v_i}) < 0.$$

- Using costs $c_{vw} = -\log r_{vw}$, dicircuits of negative cost correspond to money-making sequences.

Correcness of Ford's algorithm

We now show that Ford's algorithm terminates with least-cost dipaths, if there are no negative-cost dicircuits...

Intermediate results

For every $v \in V(G)$, we denote by y_v^j the value of y_v at iteration j, and by $p^j(v)$ be the predecessor of v at iteration j.

Key observation: If at iteration j we correct arc ab, we have $y_b^j < y_b^{j-1}$, and $y_v^j = y_v^{j-1}$ for every other node v (Why?).

Intermediate results

For every $v \in V(G)$, we denote by y_v^j the value of y_v at iteration j, and by $p^j(v)$ be the predecessor of v at iteration j.

Proposition 2.10 (part 1)

At each iteration j of Ford's algorithm, $\forall v \in V$ with $y_v^j \neq \infty$:

1. There is a dipath from r to v of cost equal to y_v^j.
2. If (G, c) has no negative-cost dicircuit, there is a simple dipath from r to v of cost equal to y_v^j.

Proposition 2.10 (part 2)

At each iteration j of Ford's algorithm, $\forall v \in V$ with $p^j(v) \neq -1$:

If (G, c) has no negative-cost dicircuit, then p^j defines a simple dipath from r to v of cost at most y_v^j.

Note: the path in part 2 is defined by the predecessors p^j, the path in part 1 might be different.

Part 1: example

Proposition 2.10 (part 1)

At each iteration j of Ford's algorithm, $\forall v \in V$ with $y_v^j \neq \infty$:

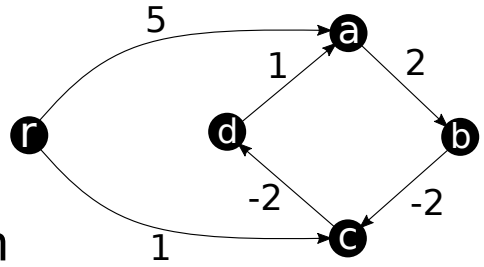
1. There is a dipath from r to v of cost equal to y_v^j.
2. If (G, c) has no negative-cost dicircuit, there is a simple dipath from r to v of cost equal to y_v^j.

Consider the example in the next slide.

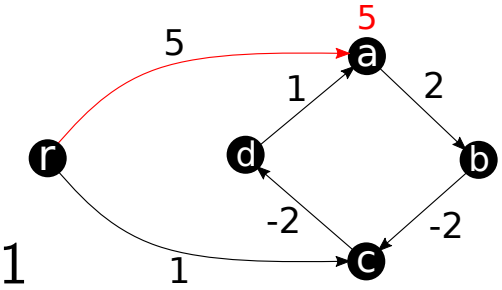
- At iteration 7 there exists a dipath to node a of length $y_a^7 = 4$. This is r, a, b, c, d, a (note that the dipath is not the path given by the predecessors).
- This dipath is not simple, since it contains the sequence a, b, c, d, a. In fact, G contains a negative cost dicircuit.

We now prove Proposition 2.10, part 1.

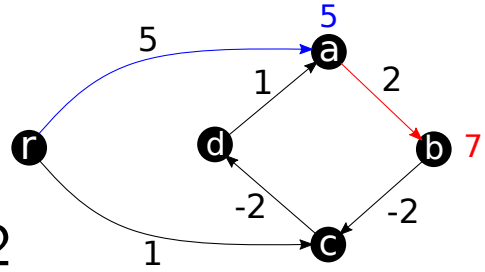
Graph



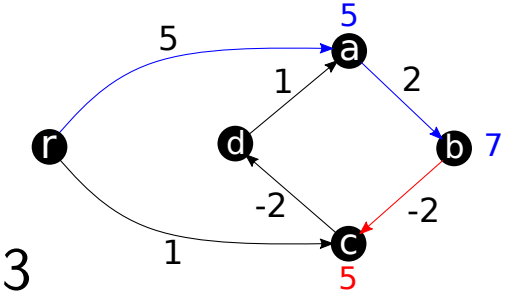
Iteration 1



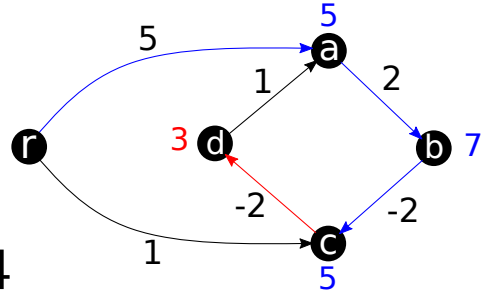
Iteration 2



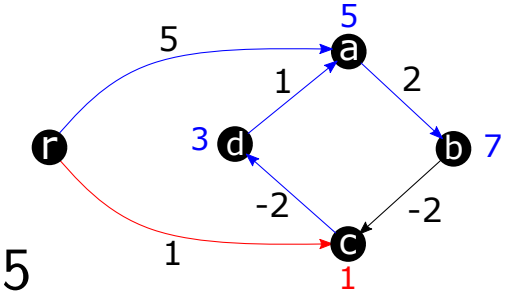
Iteration 3



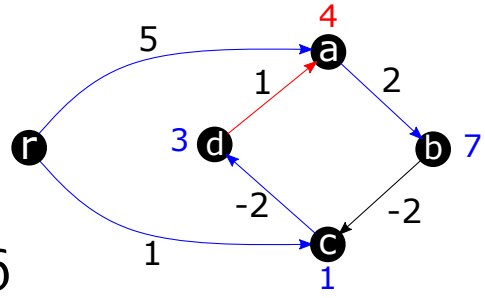
Iteration 4



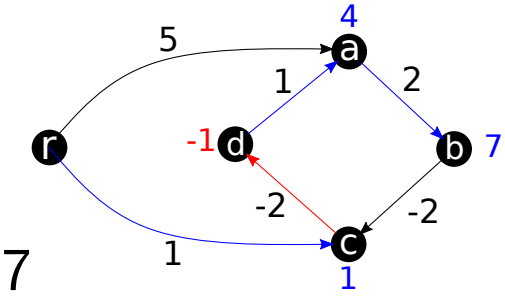
Iteration 5



Iteration 6



Iteration 7



Part 2

To prove Proposition 2.10 (part 2) we need an additional lemma.

Lemma *

Consider an iteration j of Ford's algorithm. We have

(i) $y_u^j + c_{uv} \leq y_v^j \quad \forall\, uv \in E : u = p^j(v)$.

Moreover, if at iteration j we have corrected arc $ab \in E$, we have:

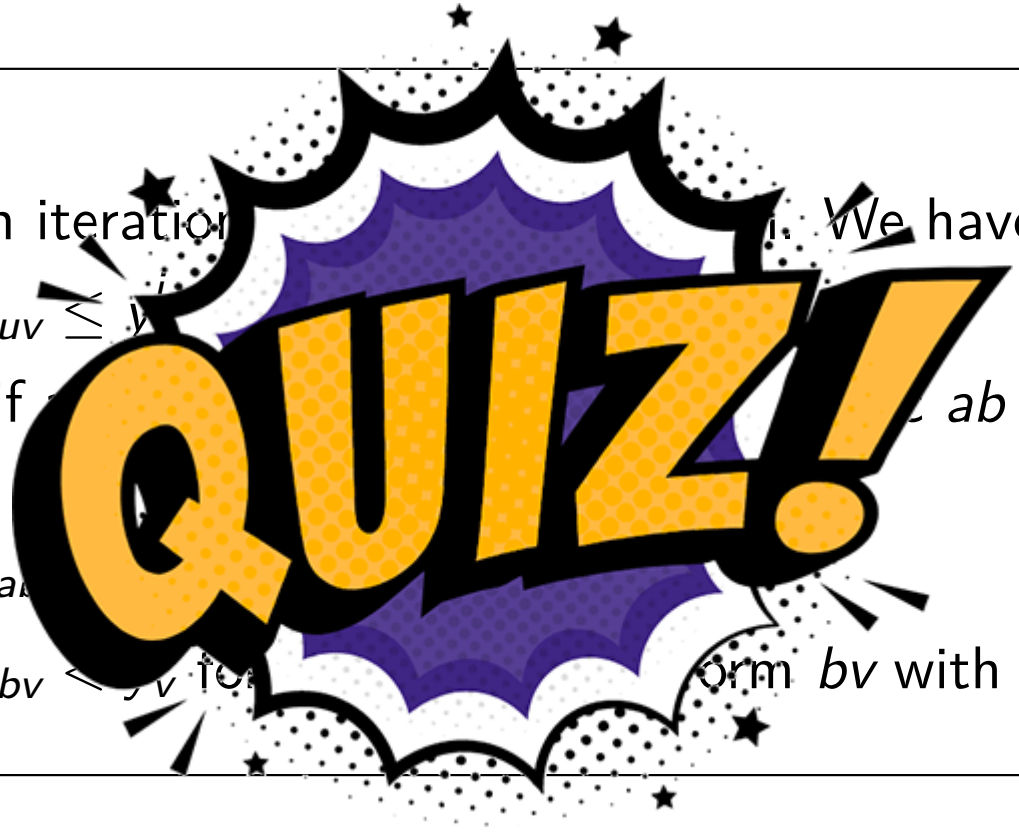
(ii) $y_a^j + c_{ab} = y_b^j$ and $a = p^j(b)$.

(iii) $y_b^j + c_{bv} < y_v^j$ for each arc of the form bv with $b = p^j(v)$.

Main remark: Since we correct arc ab, we have $y_b^j < y_b^{j-1}$, and $y_v^j = y_v^{j-1}$ for all $v \neq b$.

Part 2

To prove Proposition 2.10 (part 2) we need an additional lemma.



Lemma *

Consider an iteration j of the algorithm. We have

(i) $y_u^j + c_{uv} \geq y_v^j$ for all $uv \in A$.

Moreover, if $y_u^j + c_{uv} = y_v^j$ for all $uv \in A$ and $ab \in E$, we have:

(ii) $y_a^j + c_{ab} = y_b^j$ and

(iii) $y_b^j + c_{bv} < y_v^j$ for all arcs bv with $b = p^j(v)$.

Main remark: Since we correct arc ab, we have $y_b^j < y_b^{j-1}$, and $y_v^j = y_v^{j-1}$ for all $v \neq b$.

Part 2: example

Proposition 2.10 (part 2)

At each iteration j of Ford's algorithm, $\forall v \in V$ with $p^j(v) \neq -1$:

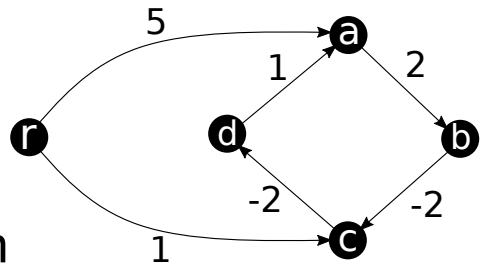
If (G, c) has no negative-cost dicircuit, then p^j defines a simple dipath from r to v of cost at most y_v^j.

Consider the example in the next slide.

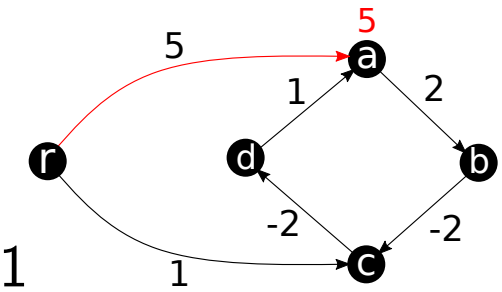
- At iteration 5 the dipath to node a defined by the predecessors p^5 is a, b, c, d, a. The dipath has cost $-1 < 4 = y_a^5$.
- This dipath is not simple, since it contains the sequence a, b, c, d, a. In fact, G contains a negative cost dicircuit.

We now prove Proposition 2.10, part 2.

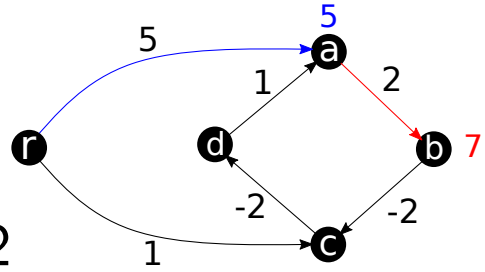
Graph



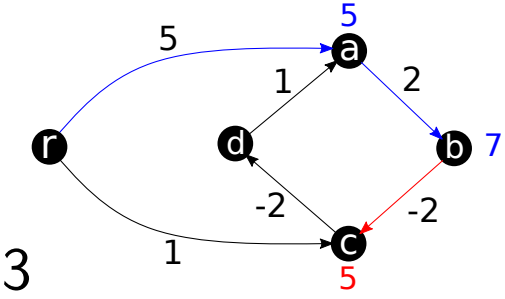
Iteration 1



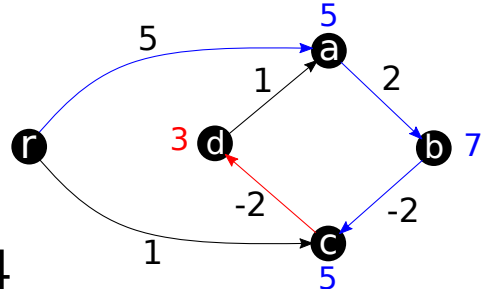
Iteration 2



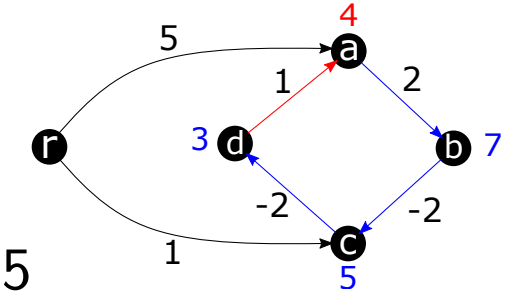
Iteration 3



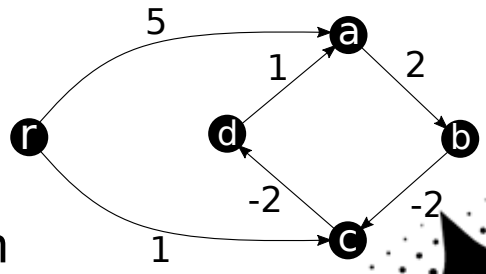
Iteration 4



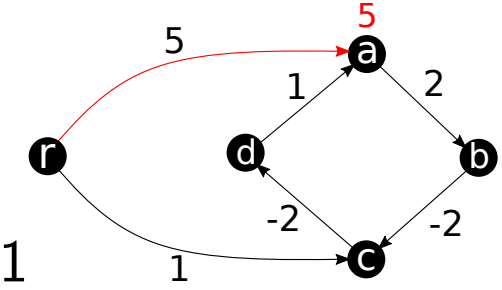
Iteration 5



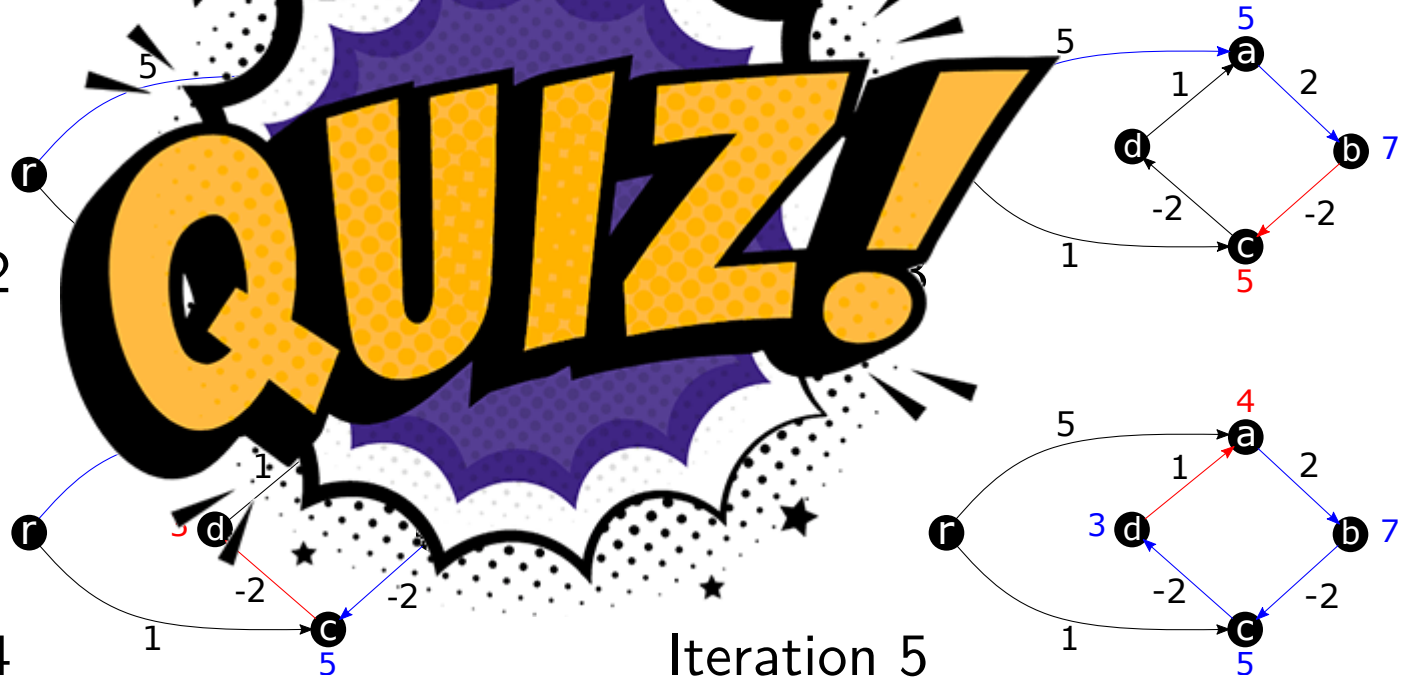
Graph



Iteration 1



Iteration 2



Iteration 4

Iteration 5

Correctness of Ford's algorithm

Theorem 2.11

If (G, c) has no negative-cost dicircuit, then Ford's algorithm terminates after a finite number of iterations. At termination, for each $v \in V$, p defines a least-cost dipath from r to v of cost y_v.

Proof: see Notes.

Refinements of Ford's Algorithm

Basic step of Ford's algorithm

1. Choose an arc e.
2. If e is incorrect, then correct it.

- We can perform each basic step in constant time.
- The nb of steps depends on the order in which arcs are chosen.
- Suppose that arcs of the graph are chosen in sequence S
- Let $S = \{e_1, \ldots, e_{|S|}\}$ (there might be repetitions!!).
- There are choices for S that result in very bad performance of the algorithm.

Refinements of Ford's Algorithm

Basic step i of Ford's algorithm

1. Choose arc e_i from S.
2. If e_i is incorrect, then correct it.

- We can perform each basic step in constant time.
- The nb of steps depends on the order in which arcs are chosen.
- Suppose that arcs of the graph are chosen in sequence S
- Let $S = \{e_1, \ldots, e_{|S|}\}$ (there might be repetitions!!).
- There are choices for S that result in very bad performance of the algorithm.

Refinements of Ford's Algorithm

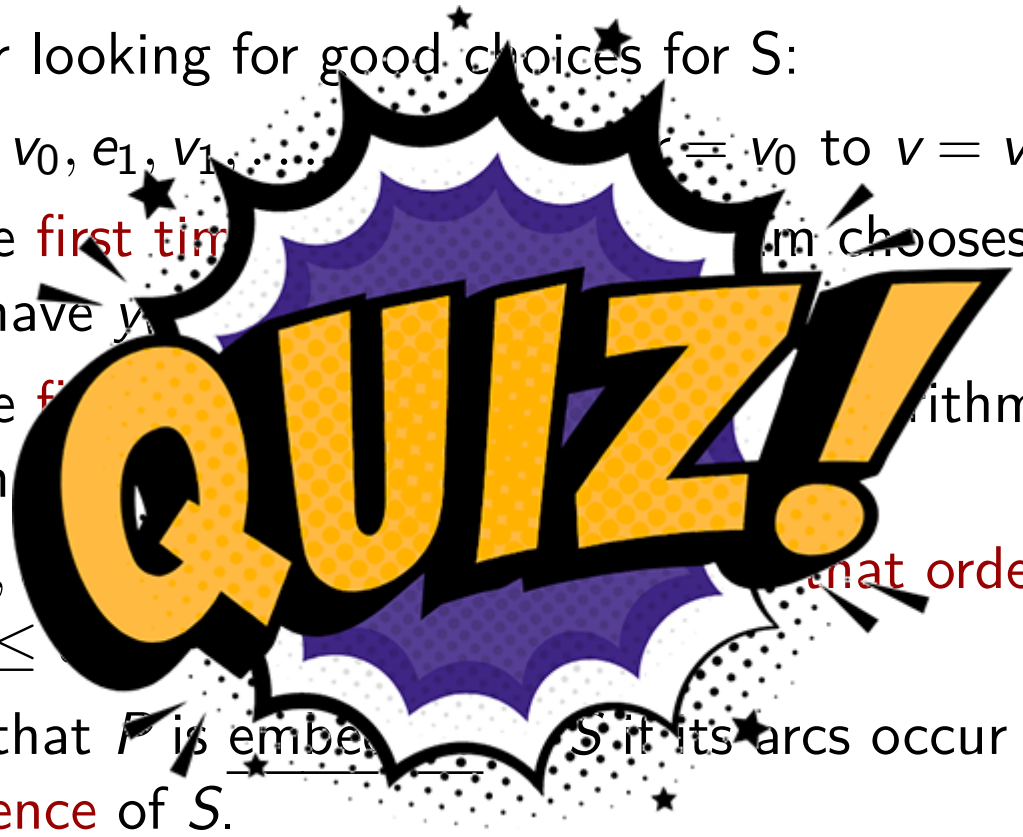
Basic idea for looking for good choices for S:

- Let $P = v_0, e_1, v_1, \ldots, e_k, v_k$ from $r = v_0$ to $v = v_k$.
- After the first time that Ford's algorithm chooses the arc e_1 we will have $y_{v_1} \leq y_r + c_{e_1} \leq c_{e_1}$.
- After the first subsequent time that the algorithm chooses e_2 we will have $y_{v_2} \leq y_{v_1} + c_{e_2} \leq c_{e_1} + c_{e_2}$.
- Once $e_1, e_2, \ldots, e_k$ have been chosen in that order, we will have $y_v \leq c(P)$.
- We say that P is embedded in S if its arcs occur as a subsequence of S.

Refinements of Ford's Algorithm

Basic idea for looking for good choices for S:

- Let $P = v_0, e_1, v_1, \ldots$ $= v_0$ to $v = v_k$.
- After the first tim m chooses the arc e_1 we will have y
- After the fi ithm chooses e_2 we will h
- Once e_1, hat order, we will have $y_v \leq$
- We say that P is embe S if its arcs occur as a subsequence of S.



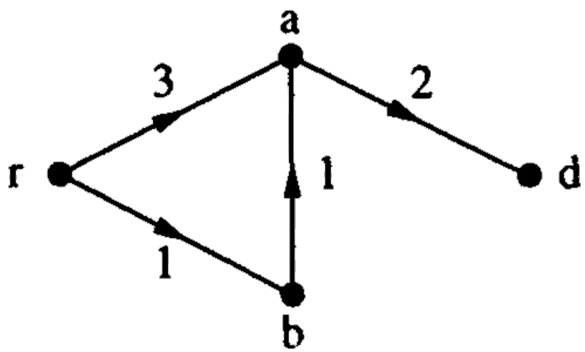
Refinements of Ford's Algorithm

Proposition 2.16
Suppose that Ford's algorithm uses sequence S. Let P be a path from r to $v \in V$ embedded in S. After scanning S we have $y_v \leq c(P)$.

- Let S be such that there is a least-cost dipath P^* to v embedded in S. After scanning S, $y_v \leq c(P^*)$. By Proposition 2.10(part 2), the dipath $\bar{P}$ to v given by the vector of predecessors is such that $c(\bar{P}) \leq y_v$. Thus $c(\bar{P}) = c(P^*)$, i.e. $\bar{P}$ is a least-cost dipath to v.
- If S is such that for every node v there is a least-cost dipath to v embedded in S, then S will bring the algorithm to termination.
- We want S to have this property, and we want S to be short, since its length is the running time of the algorithm.

The Ford-Bellman algorithm

- Every simple dipath in G is embedded in $S_1, S_2, \ldots, S_{n-1}$, where for each i, S_i is an arbitrary ordering of E.
- We speak of a sequence of "passes" through E.
- Since each arc is handled in constant time per pass, by Proposition 2.16 we get a shortest path algorithm that runs in time $O(mn)$.



G has 4 nodes, thus we only need three passes. For example:

$S_1 = ad, ba, ra, rb.$
$S_2 = ba, ad, rb, ra.$
$S_3 = rb, ba, ra, ad.$

The Ford-Bellman algorithm

- If there is no negative-cost dicircuit, then $n-1$ passes are sufficient to determine a feasible potential.
- If y is not a feasible potential after $n-1$ passes, then there exists a negative-cost dicircuit.

Ford-Bellman algorithm

1. Initialize y, p
2. Set $i = 0$
3. While $i < n$ and y is not a feasible potential
   replace i by $i + 1$
   for each $e \in E$
   if e is incorrect, correct e

The Ford-Bellman algorithm

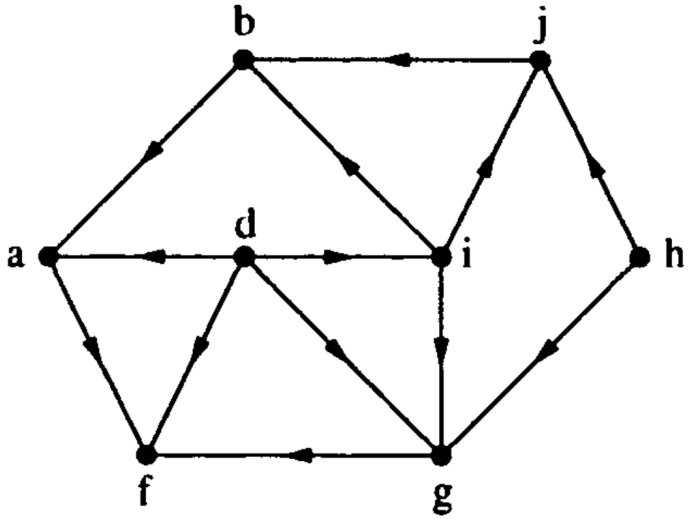
Theorem 2.17

- If $i < n$ at termination, the Ford-Bellman algorithm computes a least-cost dipath from r to v for all $v \in V$.
- Otherwise, if $i = n$ at termination, it detects that there is a negative-cost dicircuit.
- In either case it runs in time $O(mn)$.

Acyclic digraphs

- Suppose that G has a topological sort, i.e., an ordering $v_1, v_2, \ldots, v_n$ of V so that $v_i v_j \in E$ implies $i < j$.

Example:

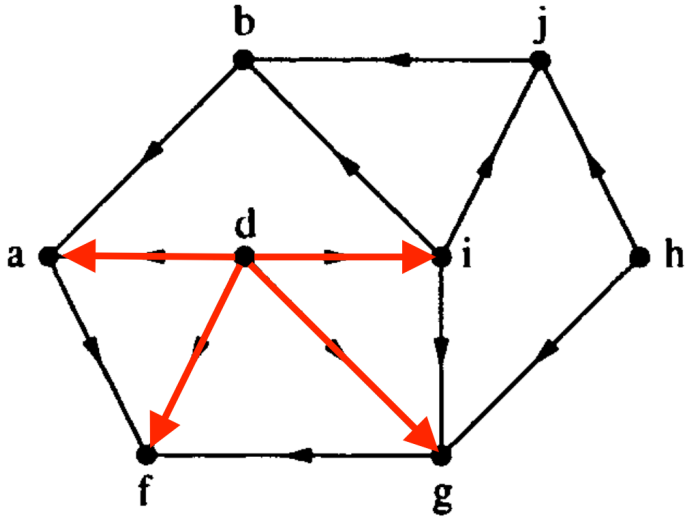


Topological sort:
d, h, i, g, j, b, a, f.

Acyclic digraphs

- Suppose that G has a topological sort, i.e., an ordering $v_1, v_2, \ldots, v_n$ of V so that $v_i v_j \in E$ implies $i < j$.
- If we order E in the sequence S so that $v_i v_j$ precedes $v_k v_l$ if $i < k$, then every dipath of G is embedded in S.

Example:



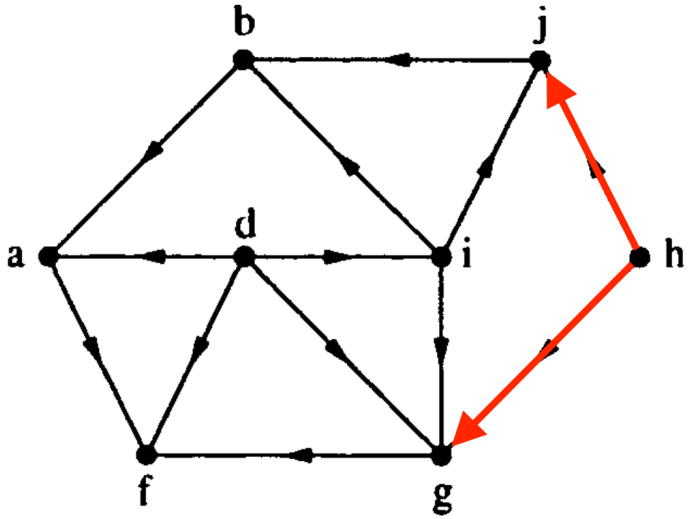
Topological sort:
d, h, i, g, j, b, a, f.

$S = da, df, dg, di,$

Acyclic digraphs

- Suppose that G has a topological sort, i.e., an ordering $v_1, v_2, \ldots, v_n$ of V so that $v_i v_j \in E$ implies $i < j$.
- If we order E in the sequence S so that $v_i v_j$ precedes $v_k v_l$ if $i < k$, then every dipath of G is embedded in S.

Example:



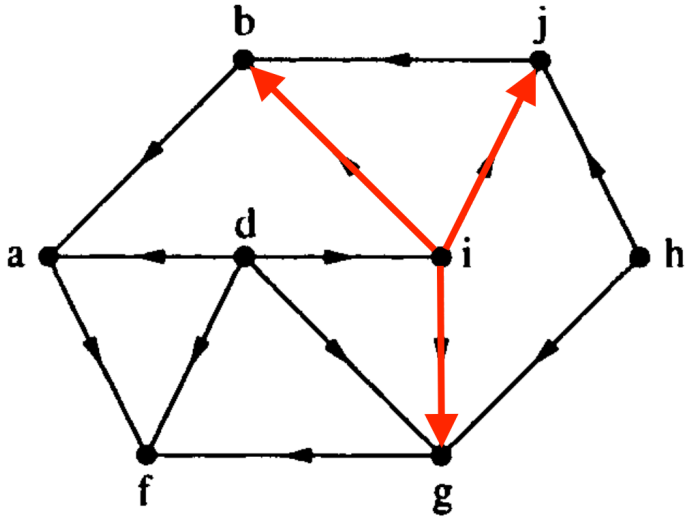
Topological sort:
d, h, i, g, j, b, a, f.

$S = da, df, dg, di,$
$hg, hj,$

Acyclic digraphs

- Suppose that G has a topological sort, i.e., an ordering $v_1, v_2, \ldots, v_n$ of V so that $v_i v_j \in E$ implies $i < j$.
- If we order E in the sequence S so that $v_i v_j$ precedes $v_k v_l$ if $i < k$, then every dipath of G is embedded in S.

Example:



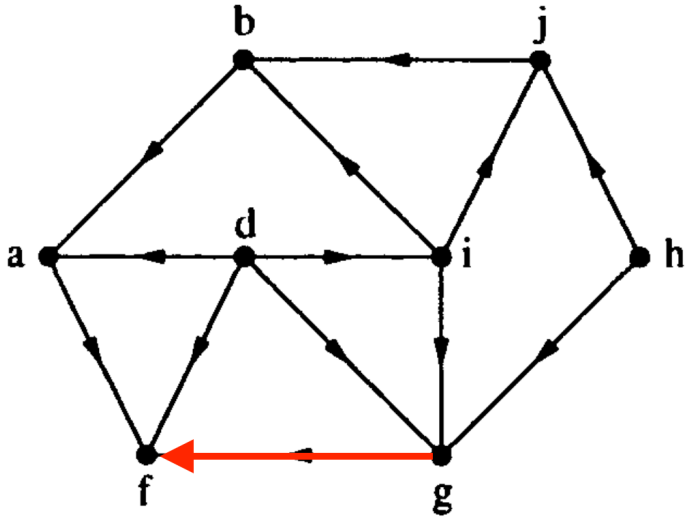
Topological sort:
d, h, i, g, j, b, a, f.

$S = da, df, dg, di,$
$hg, hj,$
$ib, ig, ij,$

Acyclic digraphs

- Suppose that G has a topological sort, i.e., an ordering $v_1, v_2, \ldots, v_n$ of V so that $v_i v_j \in E$ implies $i < j$.
- If we order E in the sequence S so that $v_i v_j$ precedes $v_k v_l$ if $i < k$, then every dipath of G is embedded in S.

Example:



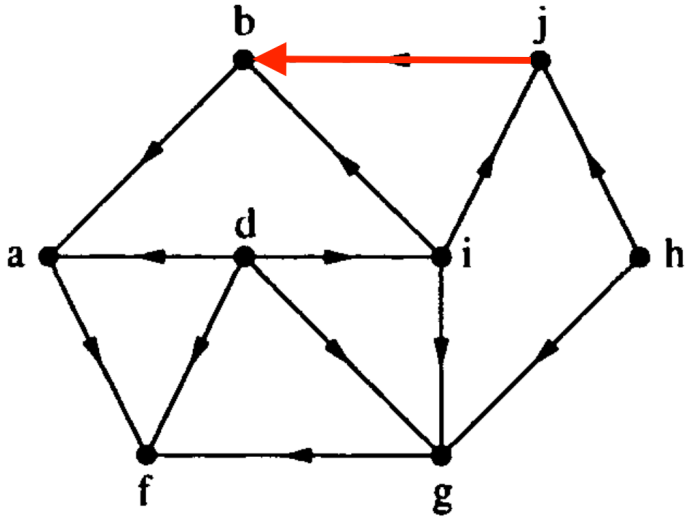
Topological sort:
d, h, i, g, j, b, a, f.

$S = da, df, dg, di,$
$hg, hj,$
$ib, ig, ij,$
$gf,$

Acyclic digraphs

- Suppose that G has a topological sort, i.e., an ordering $v_1, v_2, \ldots, v_n$ of V so that $v_i v_j \in E$ implies $i < j$.
- If we order E in the sequence S so that $v_i v_j$ precedes $v_k v_l$ if $i < k$, then every dipath of G is embedded in S.

Example:



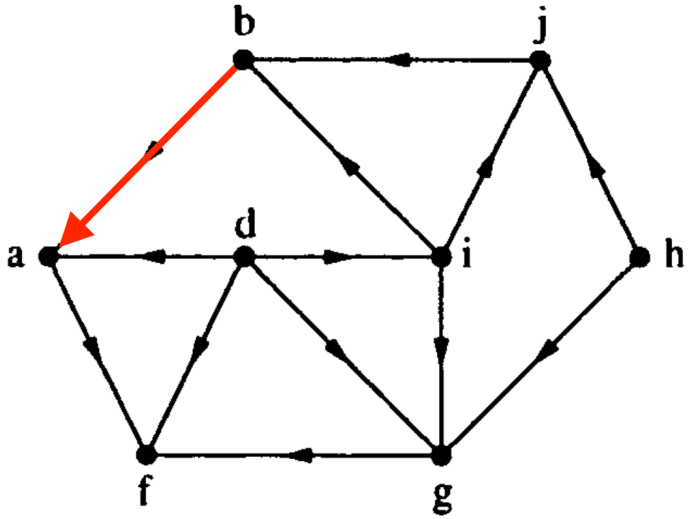
Topological sort:
d, h, i, g, j, b, a, f.

$S = da, df, dg, di,$
$hg, hj,$
$ib, ig, ij,$
$gf, jb,$

Acyclic digraphs

- Suppose that G has a topological sort, i.e., an ordering $v_1, v_2, \ldots, v_n$ of V so that $v_i v_j \in E$ implies $i < j$.
- If we order E in the sequence S so that $v_i v_j$ precedes $v_k v_l$ if $i < k$, then every dipath of G is embedded in S.

Example:



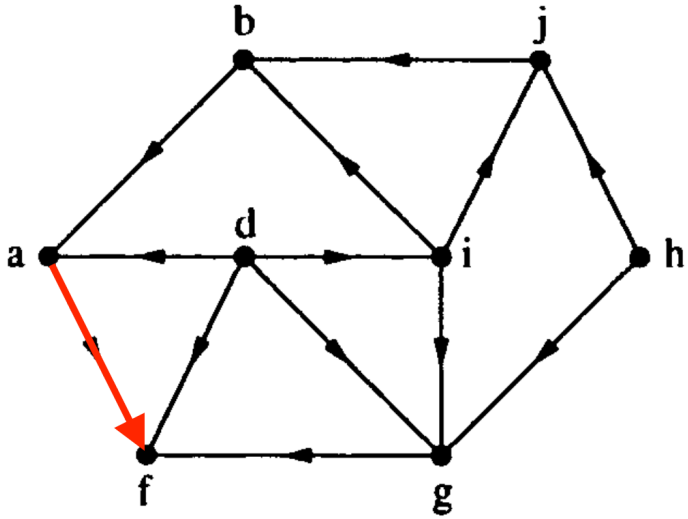
Topological sort:
d, h, i, g, j, b, a, f.

$S = da, df, dg, di,$
$hg, hj,$
$ib, ig, ij,$
$gf, jb, ba,$

Acyclic digraphs

- Suppose that G has a topological sort, i.e., an ordering $v_1, v_2, \ldots, v_n$ of V so that $v_i v_j \in E$ implies $i < j$.
- If we order E in the sequence S so that $v_i v_j$ precedes $v_k v_l$ if $i < k$, then every dipath of G is embedded in S.

Example:



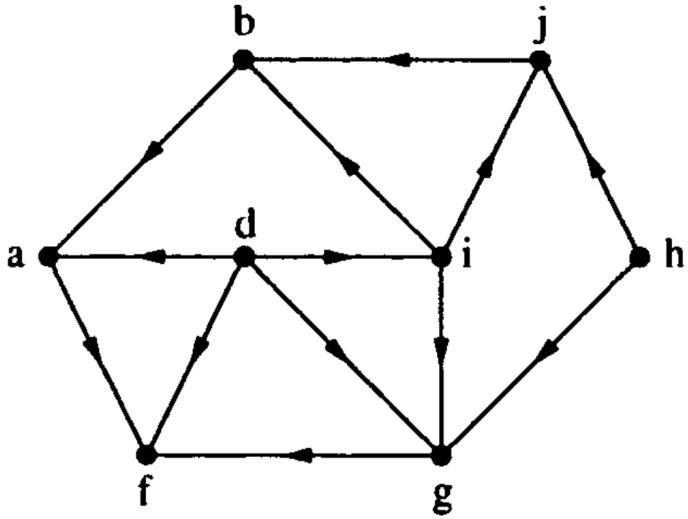
Topological sort:
d, h, i, g, j, b, a, f.

$S = da, df, dg, di,$
$hg, hj,$
$ib, ig, ij,$
$gf, jb, ba, af.$

Acyclic digraphs

- Suppose that G has a topological sort, i.e., an ordering $v_1, v_2, \ldots, v_n$ of V so that $v_i v_j \in E$ implies $i < j$.
- If we order E in the sequence S so that $v_i v_j$ precedes $v_k v_l$ if $i < k$, then every dipath of G is embedded in S.

Example:



Topological sort:
d, h, i, g, j, b, a, f.

$S = da, df, dg, di,$
$hg, hj,$
$ib, ig, ij,$
$gf, jb, ba, af.$

- It follows that Ford's Algorithm will solve the shortest path problem in just one pass through E.

Acyclic digraphs

When does a digraph have a topological sort?

We say that a digraph G is acyclic if it has no dicircuit.

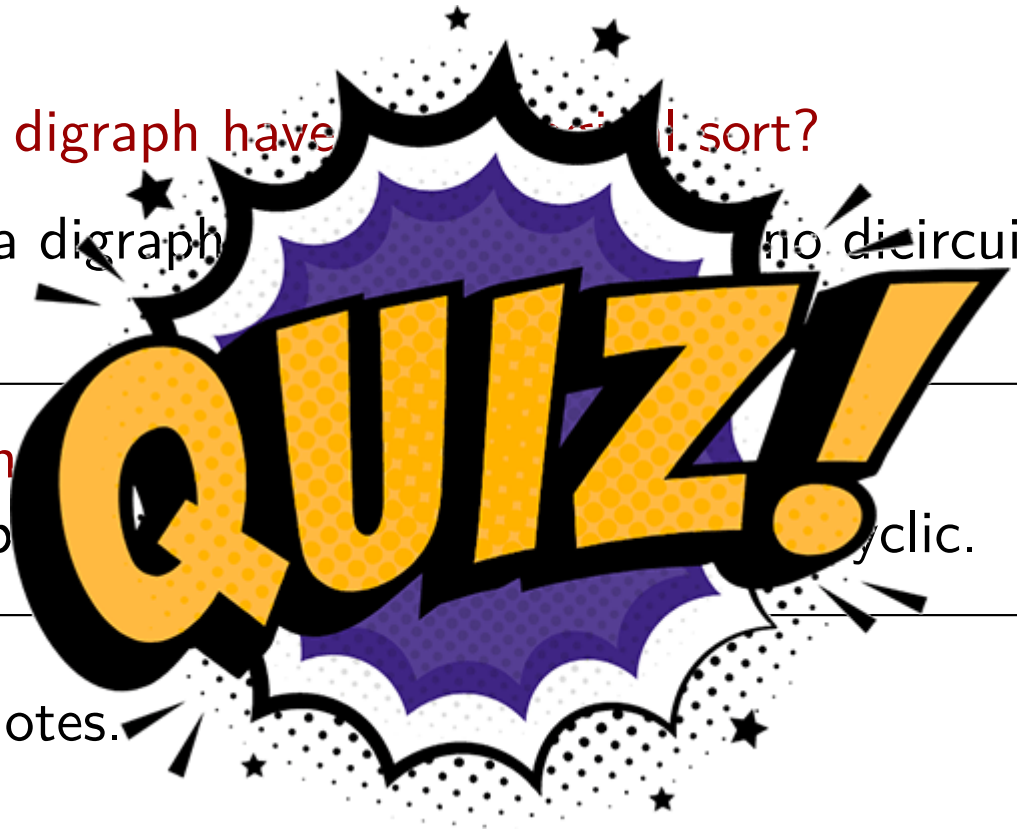
> **Observation**
> G has a topological sort if and only if G is acyclic.

Proof: see Notes.

Acyclic digraphs

When does a digraph have a topological sort?

We say that a digraph is acyclic if it has no dicircuit.



Observation
G has a topological sort if and only if it is acyclic.

Proof: see Notes.

Acyclic digraphs

- A usual representation of a digraph is to store all the arcs having tail v in a list L_v. To scan v means to do the following: For each $vw \in L_v$: if vw is incorrect, correct vw.

Shortest paths in an acyclic digraph

1. Find a topological sort $v_1, \ldots, v_n$ of G with $r = v_1$
2. Initialize y, p
3. For $i = 1$ to n
   Scan v_i

Notice that, if $r = v_i$ with $i > 1$, then there can be no dipath from r to $v_1, \ldots, v_{i-1}$ so these nodes can be deleted. Hence we may assume that $v_1 = r$.

Acyclic digraphs

- Acyclic digraphs admit a topological sort
- There is a $O(m)$ algorithm to find a topological sort
- The graph is acyclic, so all dipaths are simple
- We can order the arcs in sequence S with no repetitions, so that each dipath is embedded is S.
- $|S| = m$, i.e. we have a "one pass" algorithm!

Theorem 2.18

The shortest path problem on an acyclic digraph can be solved in time $O(m)$.

Nonnegative Costs

- In many applications of the shortest path problem we know that $c \geq 0$.
- It is possible to design a "one-pass" algorithm based on an ordering of the nodes.
- This ordering is computed during the course of execution.
- If $v_1, v_2, \ldots, v_i$ have been determined and scanned, then v_{i+1} is chosen to be the unscanned node v for which y_v is minimum.

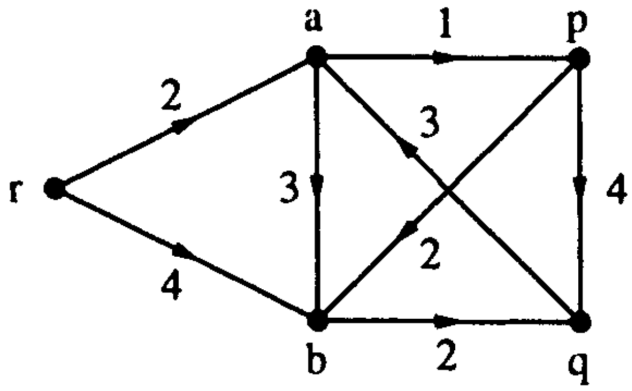
Dijkstra's algorithm

1. Initialize y, p
2. Set $Q = V$
3. While $Q \neq \emptyset$

   Choose $v \in Q$ with y_v minimum

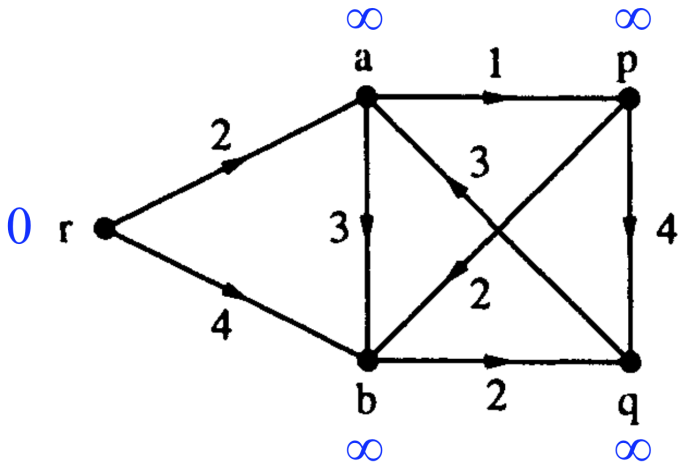
   Delete v from Q

   Scan v



Dijkstra's algorithm

1. Initialize y, p
2. Set $Q = V$
3. While $Q \neq \emptyset$
   - Choose $v \in Q$ with y_v minimum
   - Delete v from Q
   - Scan v



$Q = \{ \ r, \ \ a, \ \ b, \ \ p, \ \ q\}$

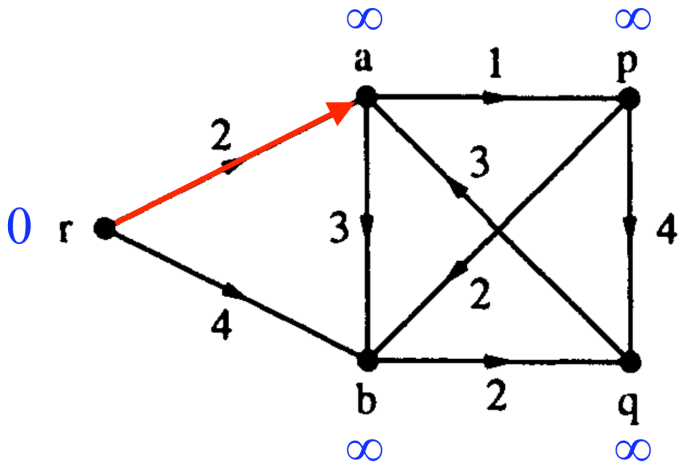
Dijkstra's algorithm

1. Initialize y, p
2. Set $Q = V$
3. While $Q \neq \emptyset$

   Choose $v \in Q$ with y_v minimum

   Delete v from Q

   Scan v



$Q = \{ \not{r}, \quad a, \quad b, \quad p, \quad q \}$

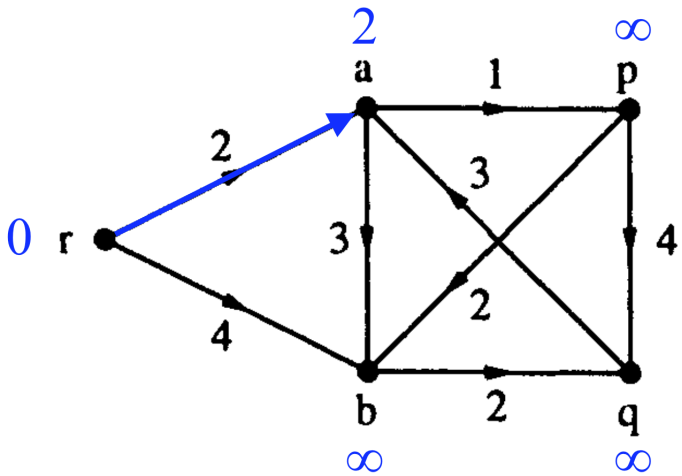
Dijkstra's algorithm

1. Initialize y, p
2. Set $Q = V$
3. While $Q \neq \emptyset$

   Choose $v \in Q$ with y_v minimum

   Delete v from Q

   Scan v



$Q = \{ \not{r}, \quad a, \quad b, \quad p, \quad q \}$

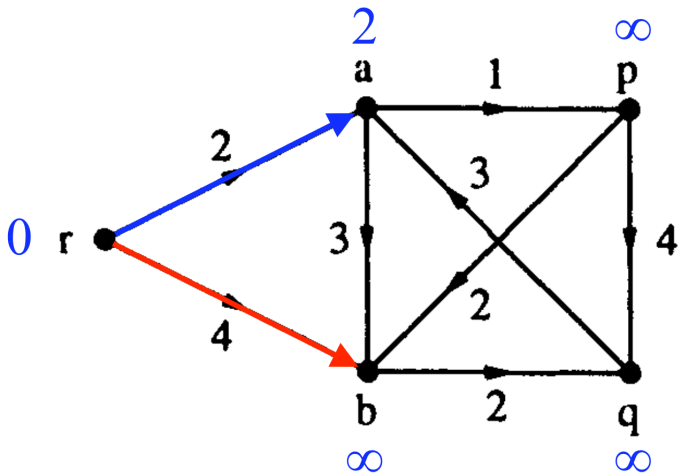
Dijkstra's algorithm

1. Initialize y, p
2. Set $Q = V$
3. While $Q \neq \emptyset$

   Choose $v \in Q$ with y_v minimum

   Delete v from Q

   Scan v



$Q = \{ \not{r}, \quad a, \quad b, \quad p, \quad q \}$

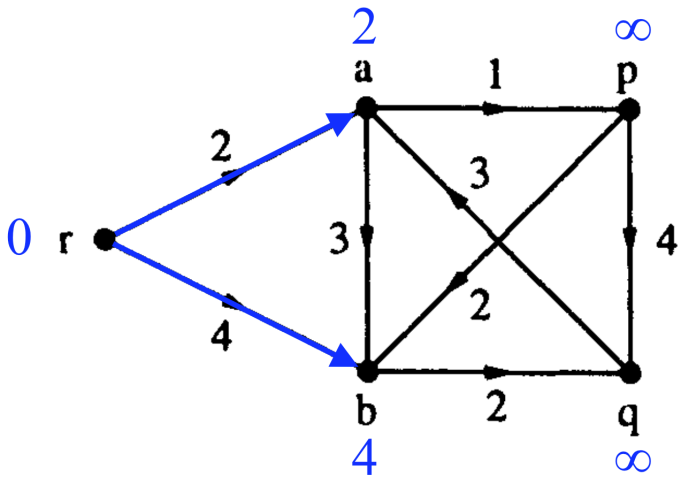
Dijkstra's algorithm

1. Initialize y, p
2. Set $Q = V$
3. While $Q \neq \emptyset$

   Choose $v \in Q$ with y_v minimum

   Delete v from Q

   Scan v



$Q = \{ \not{r}, \quad a, \quad b, \quad p, \quad q \}$

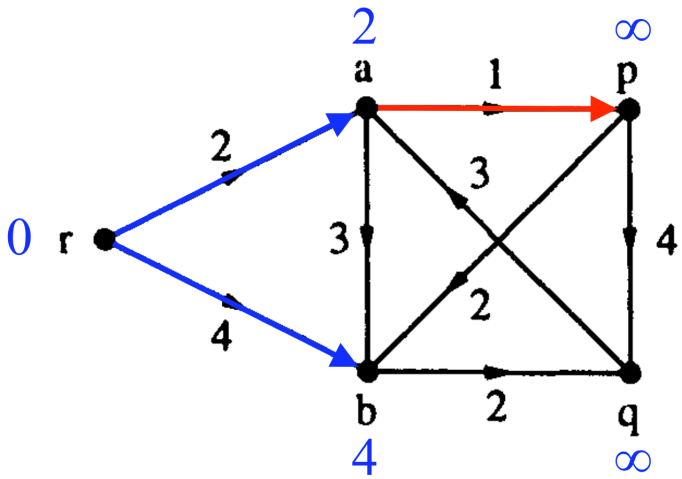
Dijkstra's algorithm

1. Initialize y, p
2. Set $Q = V$
3. While $Q \neq \emptyset$

   Choose $v \in Q$ with y_v minimum

   Delete v from Q

   Scan v



$Q = \{ \not{r}, \not{a}, b, p, q\}$

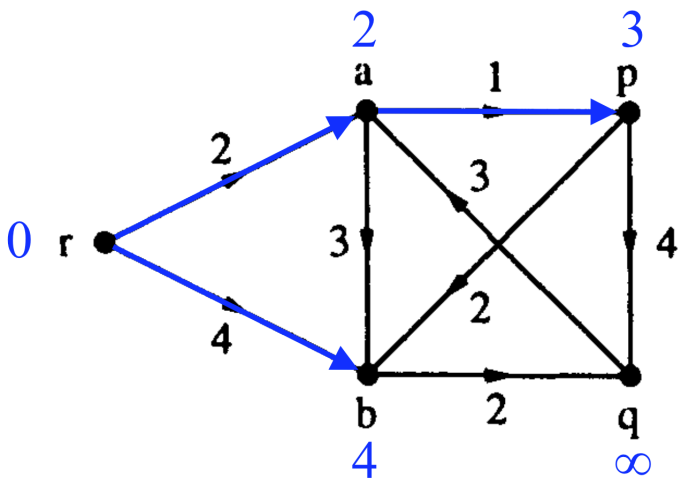
Dijkstra's algorithm

1. Initialize y, p
2. Set $Q = V$
3. While $Q \neq \emptyset$

   Choose $v \in Q$ with y_v minimum

   Delete v from Q

   Scan v



$$Q = \{ \not{r},\ \not{a},\ b,\ p,\ q \}$$

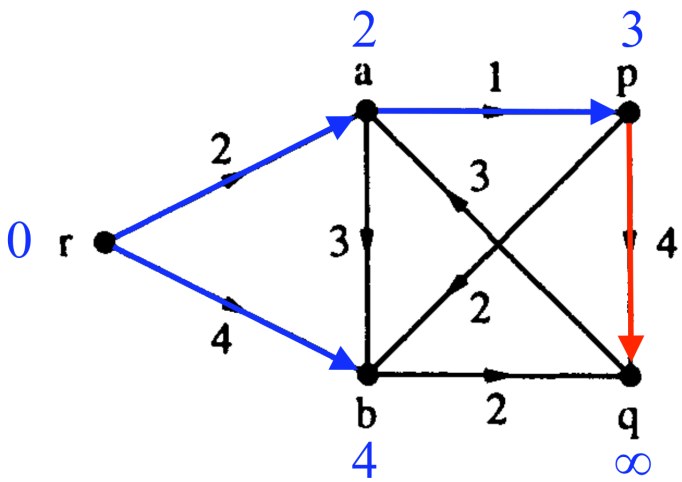
Dijkstra's algorithm

1. Initialize y, p
2. Set $Q = V$
3. While $Q \neq \emptyset$

   Choose $v \in Q$ with y_v minimum

   Delete v from Q

   Scan v



$$Q = \{ \not{r},\ \not{a},\ b,\ \not{p},\ q \}$$

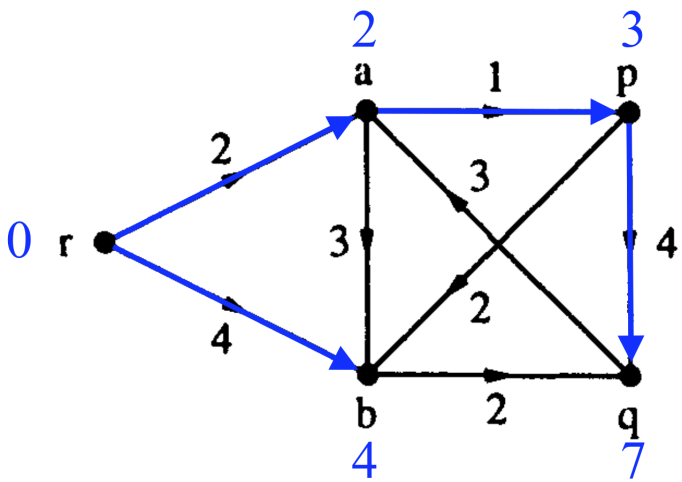
Dijkstra's algorithm

1. Initialize y, p
2. Set $Q = V$
3. While $Q \neq \emptyset$

   Choose $v \in Q$ with y_v minimum

   Delete v from Q

   Scan v



$Q = \{ \not{r}, \not{a}, b, \not{p}, q\}$

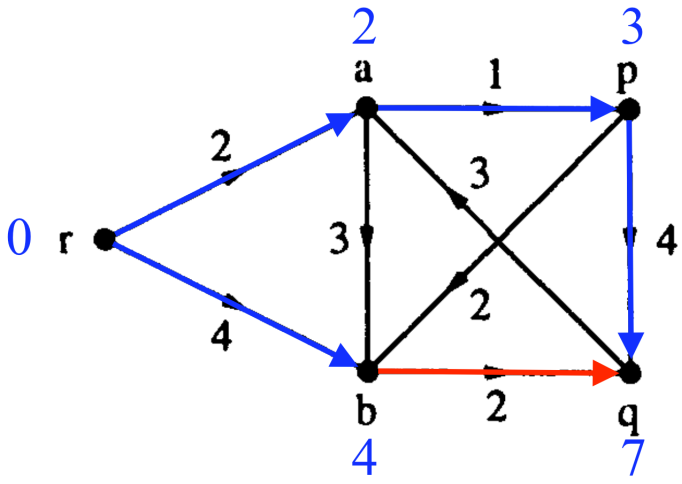
Dijkstra's algorithm

1. Initialize y, p
2. Set $Q = V$
3. While $Q \neq \emptyset$

   Choose $v \in Q$ with y_v minimum

   Delete v from Q

   Scan v



$Q = \{ \not{r},\ \not{a},\ \not{b},\ \not{p},\ q\}$

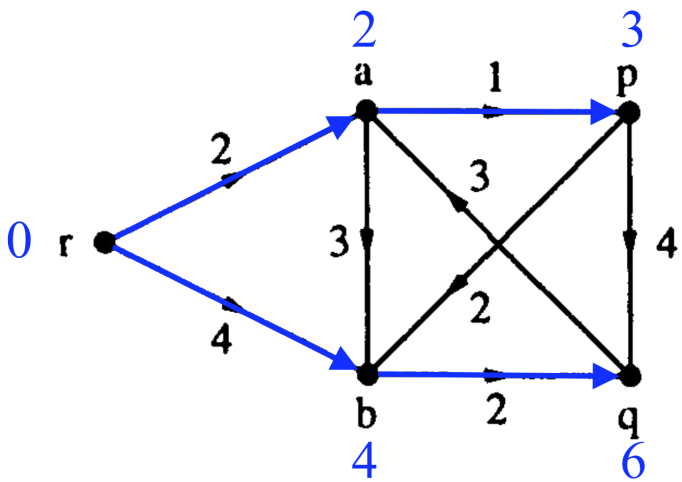
Dijkstra's algorithm

1. Initialize y, p
2. Set $Q = V$
3. While $Q \neq \emptyset$

   Choose $v \in Q$ with y_v minimum

   Delete v from Q

   Scan v



$Q = \{ \not{r},\ \not{a},\ \not{b},\ \not{p},\ q\}$

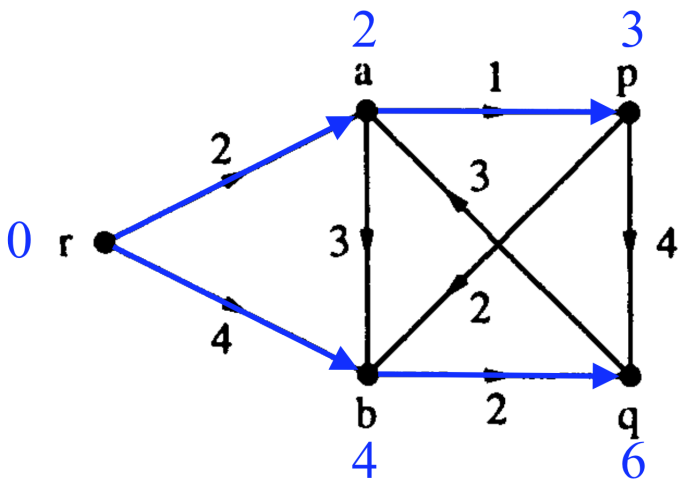
Dijkstra's algorithm

1. Initialize y, p
2. Set $Q = V$
3. While $Q \neq \emptyset$

   Choose $v \in Q$ with y_v minimum

   Delete v from Q

   Scan v



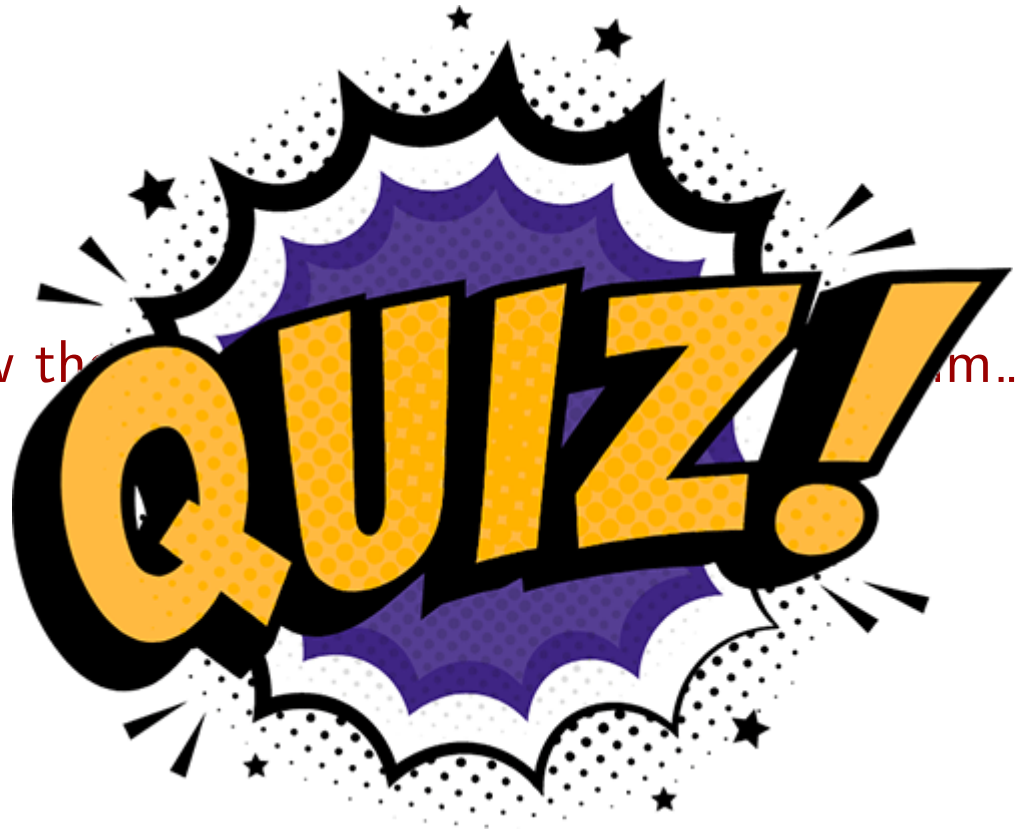
$Q = \{ \not r, \not a, \not b, \not p, \not q \}$

Correctness of Dijkstra's algorithm

We now show the correctness of Dijkstra's algorithm...

Correctness of Dijkstra's algorithm

We now show th m...



Correctness of Dijkstra's algorithm

For each $w \in V$, let y'_w be the value of y_w when w is chosen to be scanned.

Proposition 2.19
If u is scanned before v, then $y'_u \leq y'_v$.

Proof: see Notes.

Theorem
Dijkstra's algorithm is valid.

Proof: see Notes.

Dijkstra's algorithm

- The running time of the algorithm is given by the time to find v plus the time to scan v.
- We need $O(m)$ to scan all given nodes.
- Choosing v requires considerable time:
  $1 + 2 + \cdots + (n-2) + (n-1) = O(n^2)$ comparisons.
- The running time is $O(n^2)$.

Theorem 2.20

If $c \geq 0$, then the shortest path problem can be solved in time $O(n^2)$.

Unit costs

- All arc-costs are 1: problem of finding a dipath from r to v having as few arcs as possible.

> Proposition 2.21
> If each $c_e = 1$, then in Dijkstra's algorithm the final value of y_v is the first finite value assigned to it. Moreover, if v is assigned its first finite y_v before q is, then $y_v \leq y_q$.

Proof: see Notes.

Unit costs

- When picking $v \in Q$ such that y_v is minimum, we choose among the set Q of those unscanned nodes v having y_v finite.
- Proposition 2.21 tells us that we can simply choose the element of Q that was added to Q first, that is, we can keep Q as a queue, and v can be found in constant time.
- So Dijkstra's Algorithm has a running time of $O(m)$ in the case of unit costs.
- We no longer need to maintain the y_v.

Breadth-first search

1. Initialize p
2. Set $Q = \{r\}$
3. While $Q \neq \emptyset$

    Delete v from the front of Q

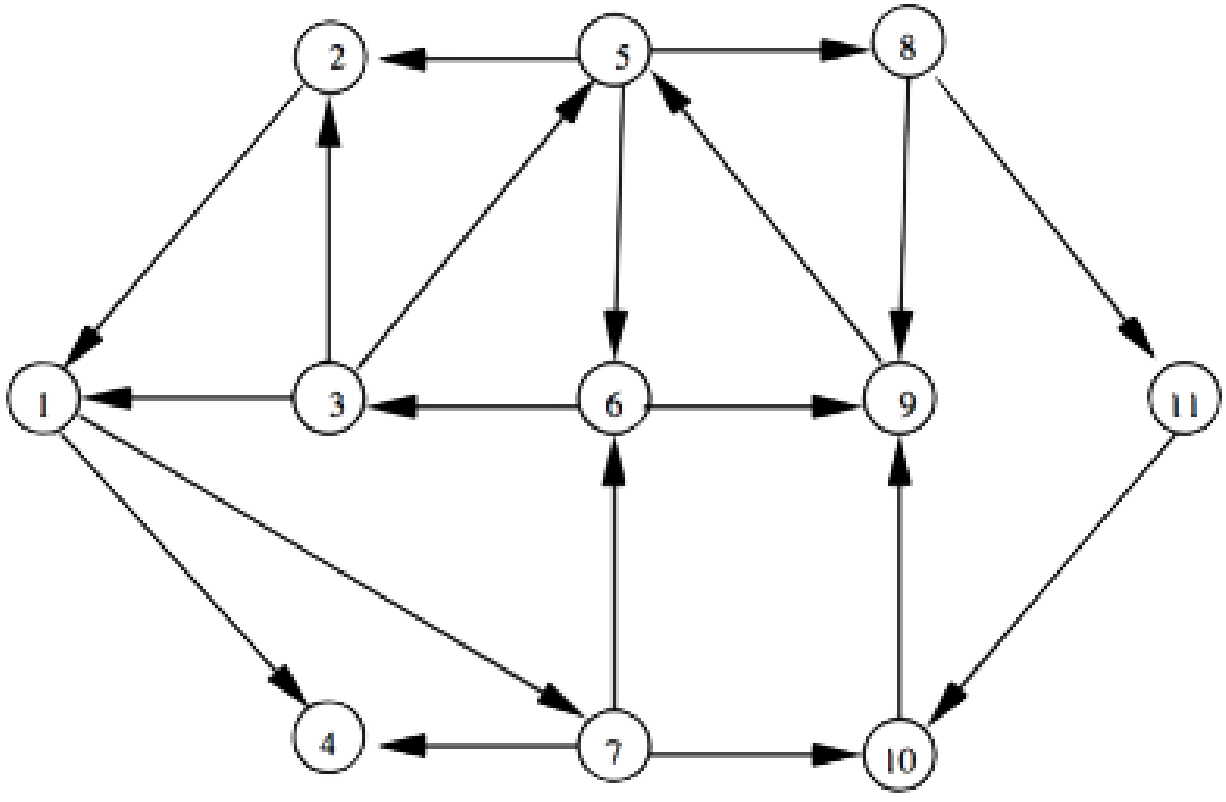
    For each $vw \in L_v$

        If $p(w) = -1$

            Add w to the back of Q

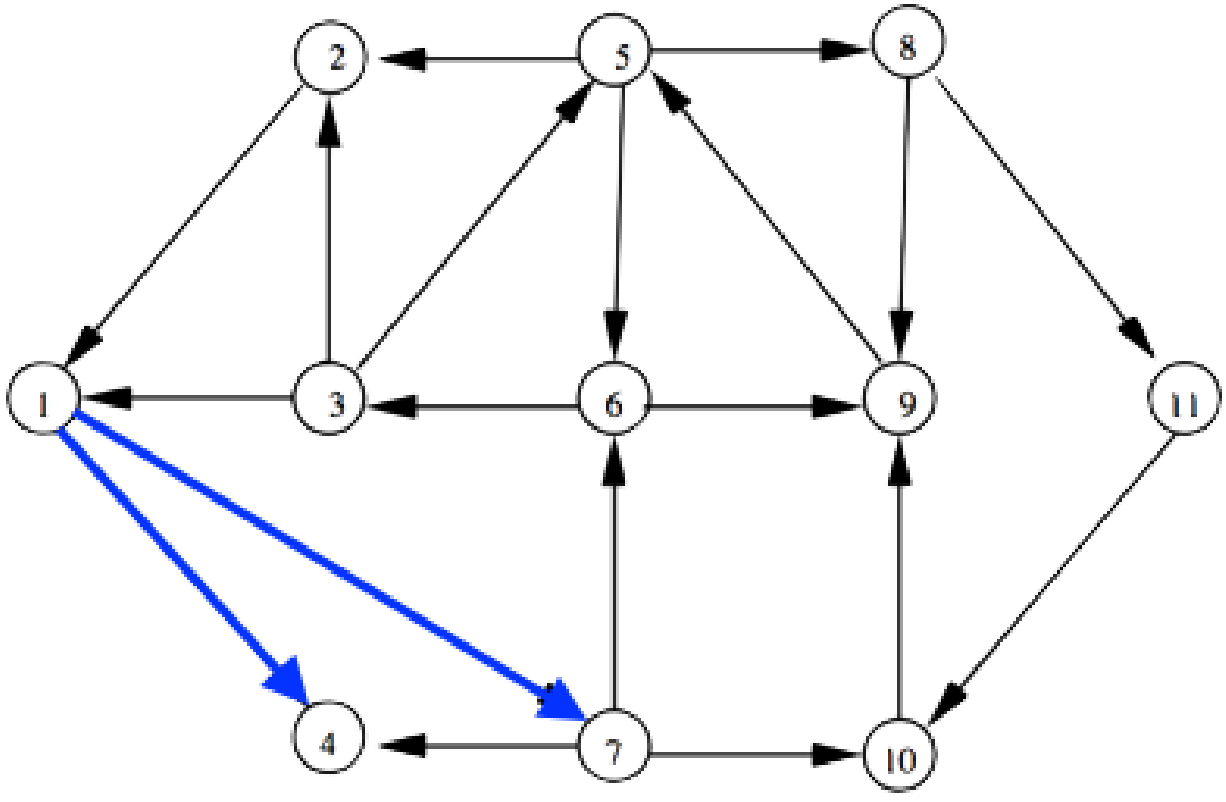
            Set $p(w) = v$

Example



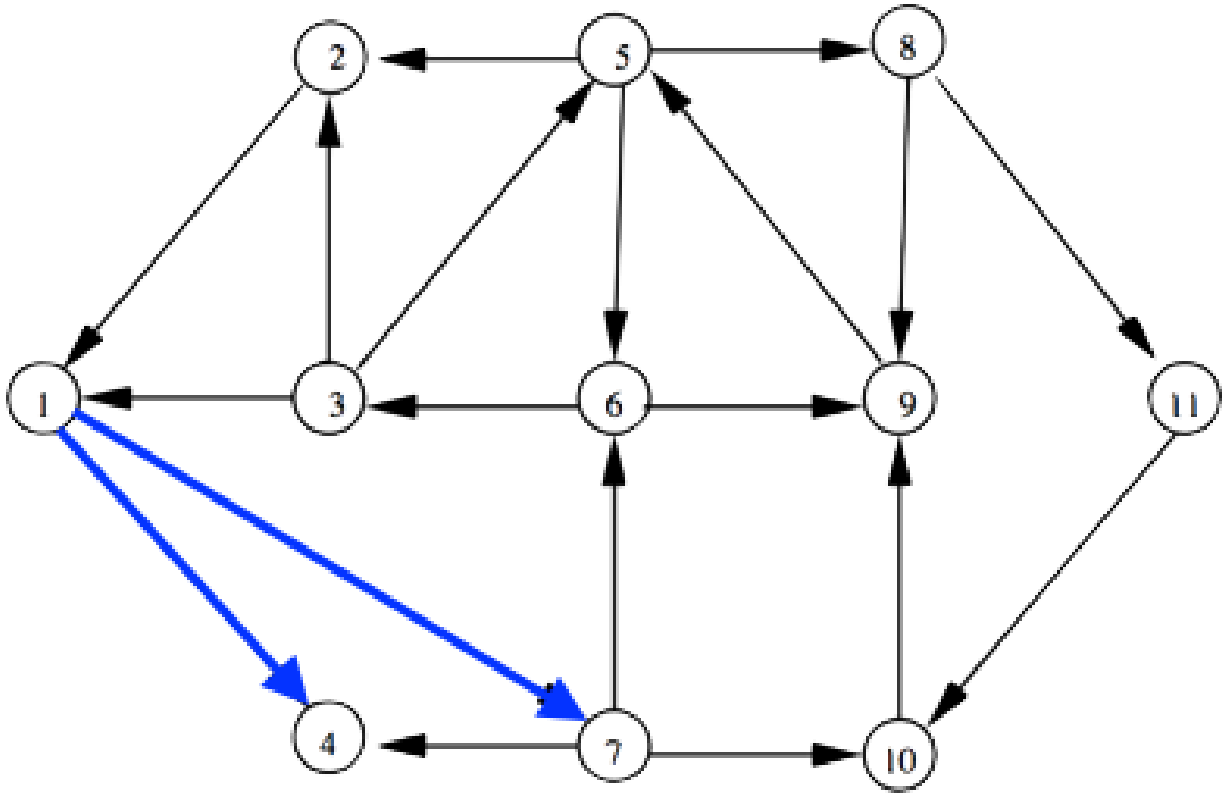
$Q = \{\ 1,$ $\}$

Example



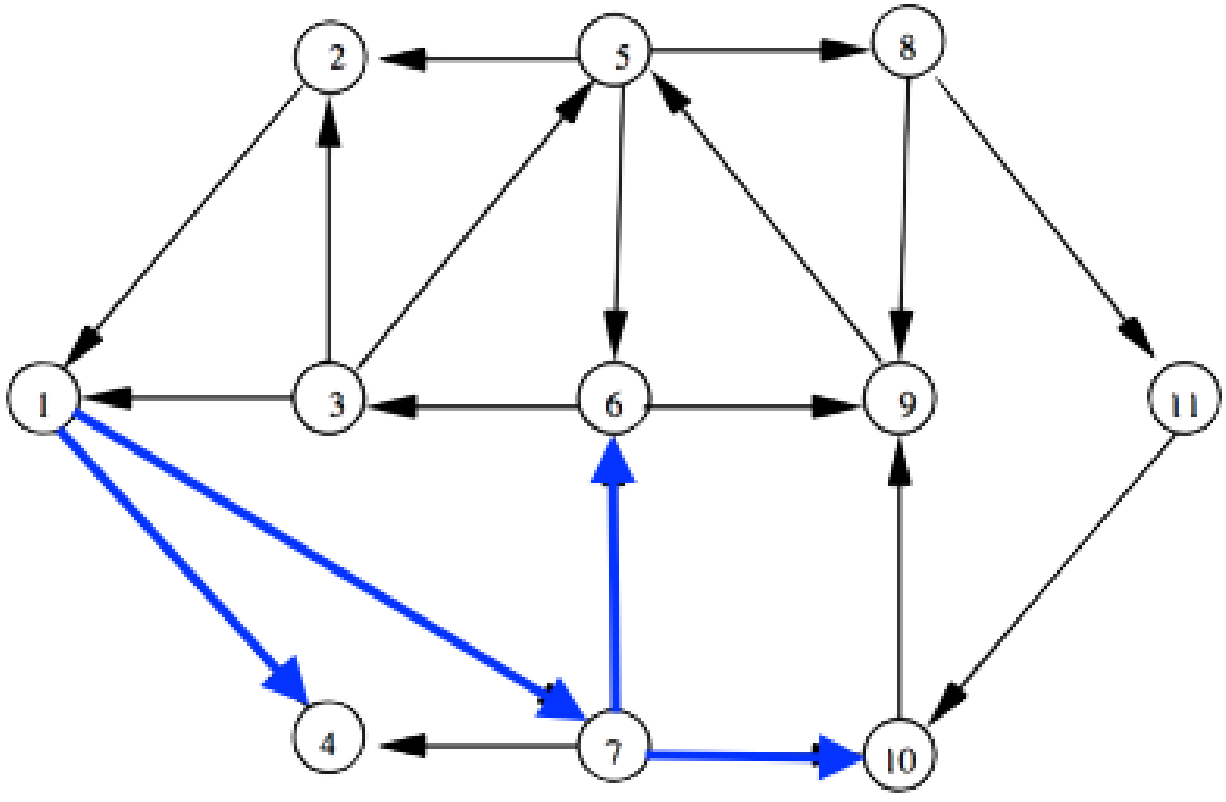
$Q = \{\ 1,\ \ 4,\ \ 7,\ \ \}$

Example



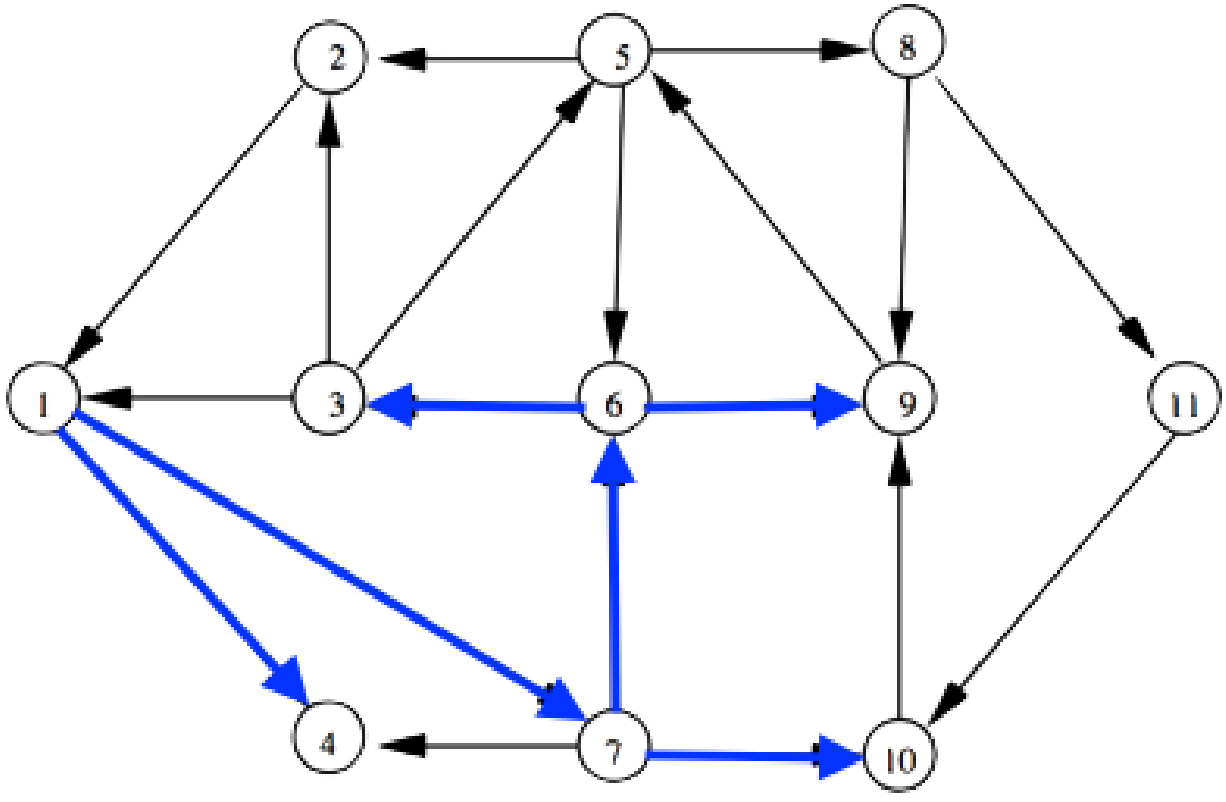
$Q = \{\ 1,\ 4,\ 7,\ \}$

Example



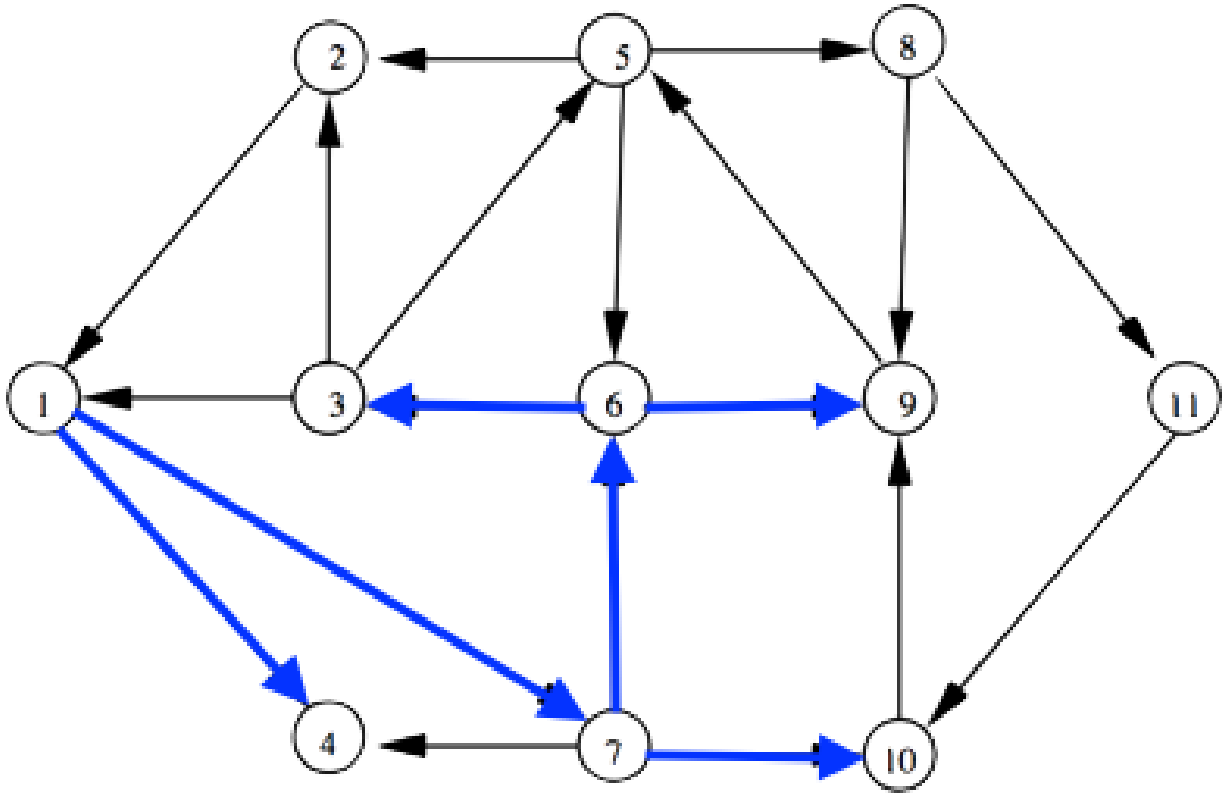
$Q = \{\ \not{1},\ \not{4},\ \not{7},\ 6,\ 10,\ \}$

Example



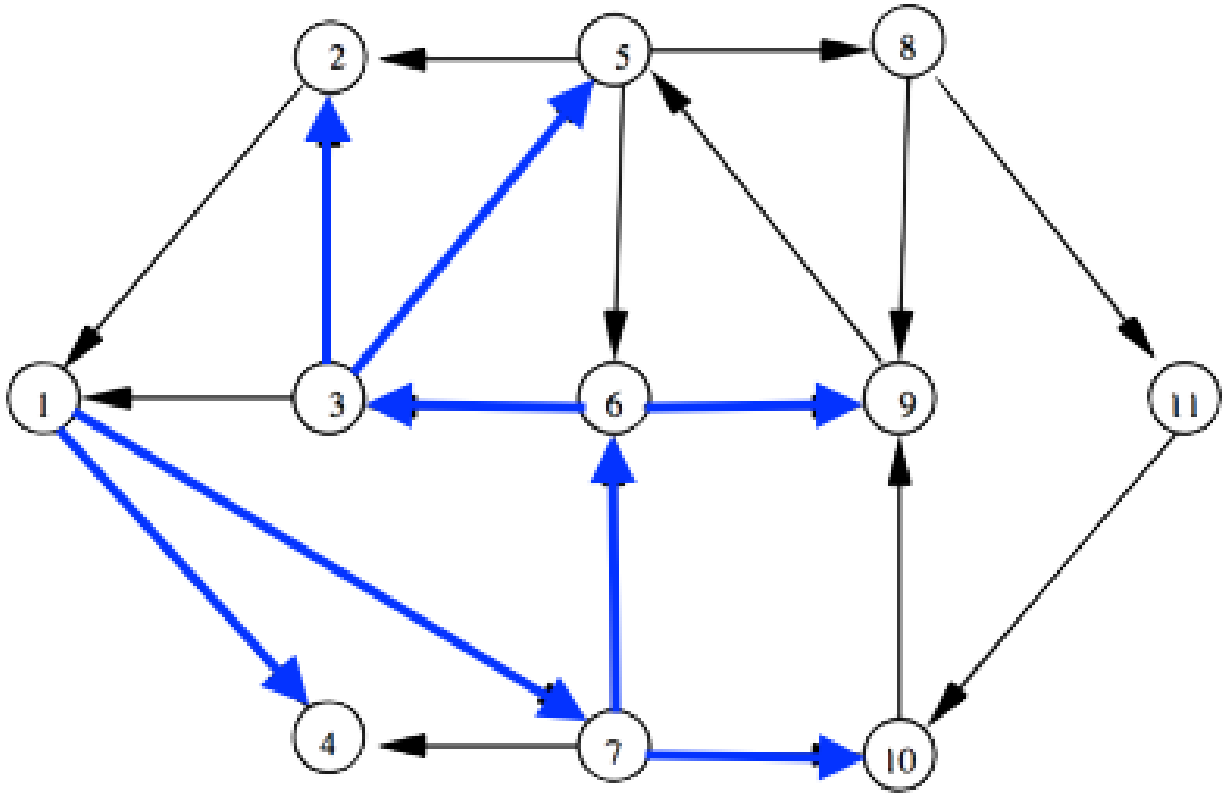
$Q = \{\ \not{1},\ \not{4},\ \not{7},\ \not{6},\ 10,\ 3,\ 9,\qquad\qquad\qquad\qquad\ \}$

Example



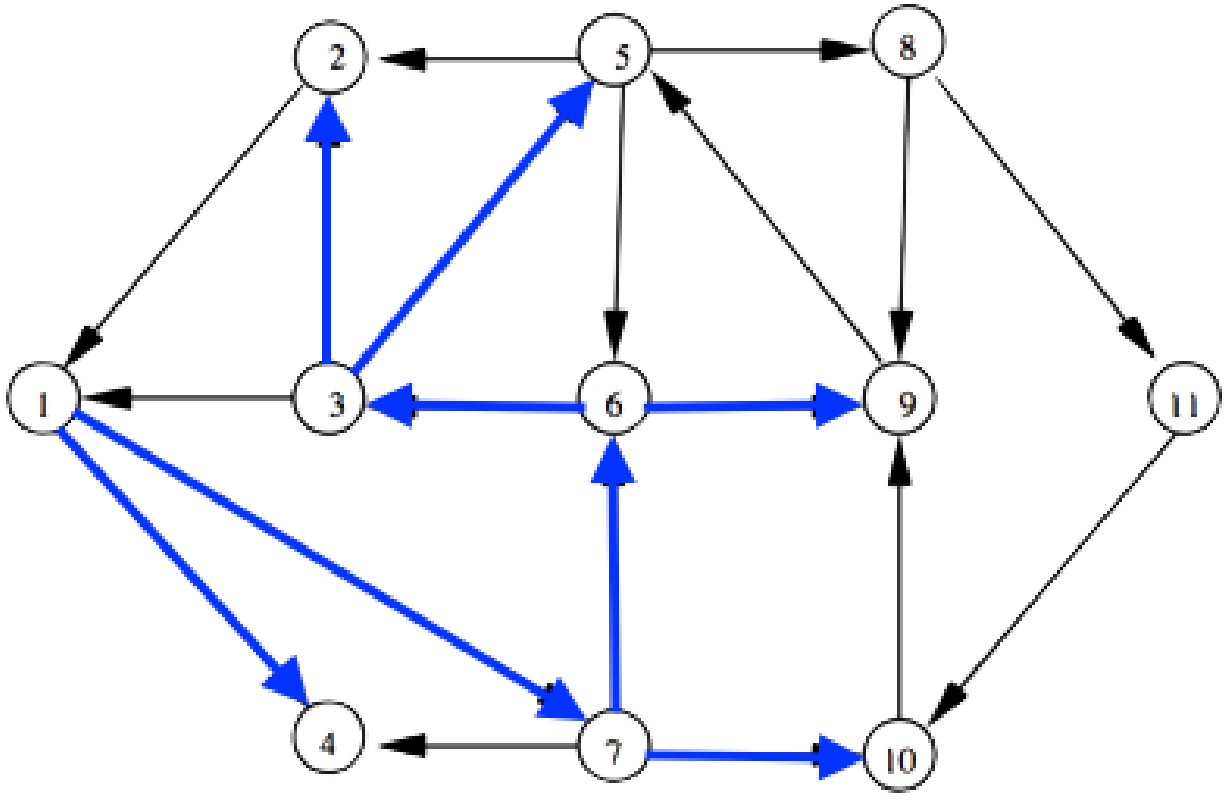
$Q = \{\ \not{1},\ \not{4},\ \not{7},\ \not{6},\ \not{10},\ 3,\ 9,\qquad\qquad\qquad\qquad\ \}$

Example



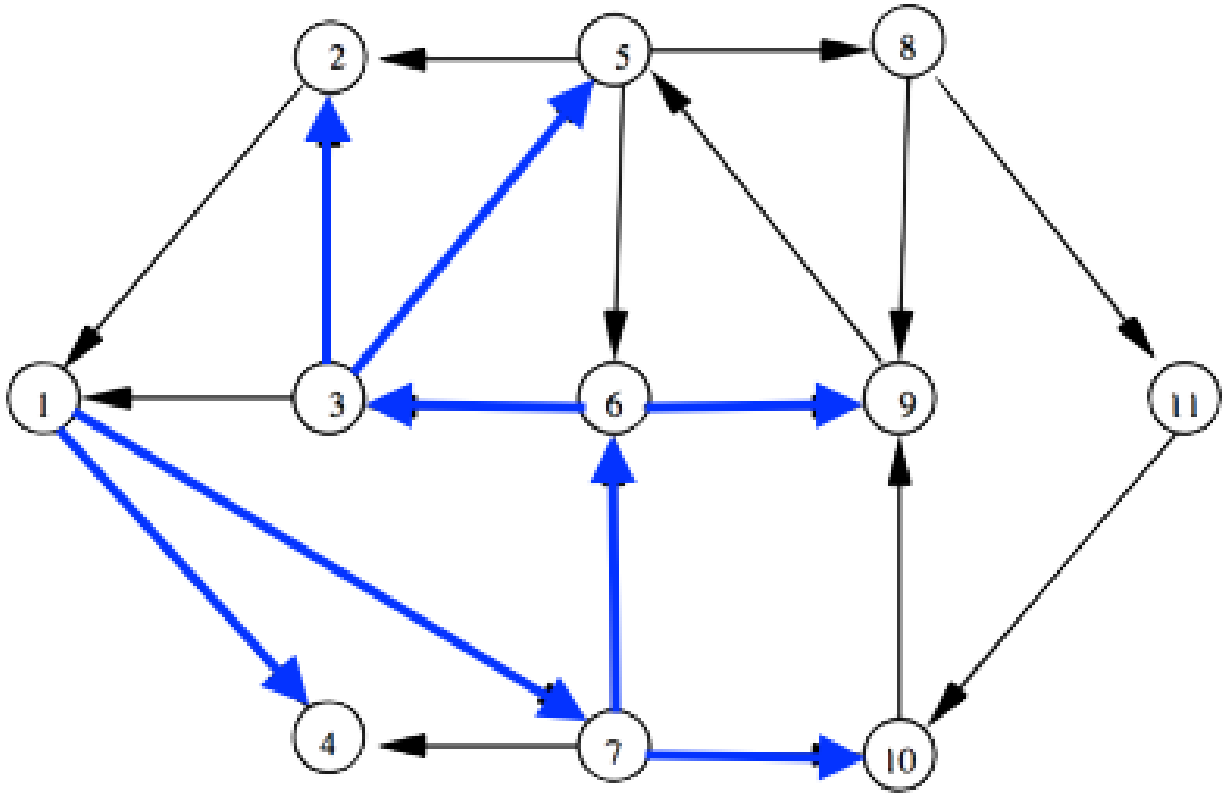
$Q = \{\ \not{1},\ \not{4},\ \not{7},\ \not{6},\ \not{10},\ \not{3},\ 9,\ 2,\ 5,\ \ \ \ \ \ \ \ \}$

Example



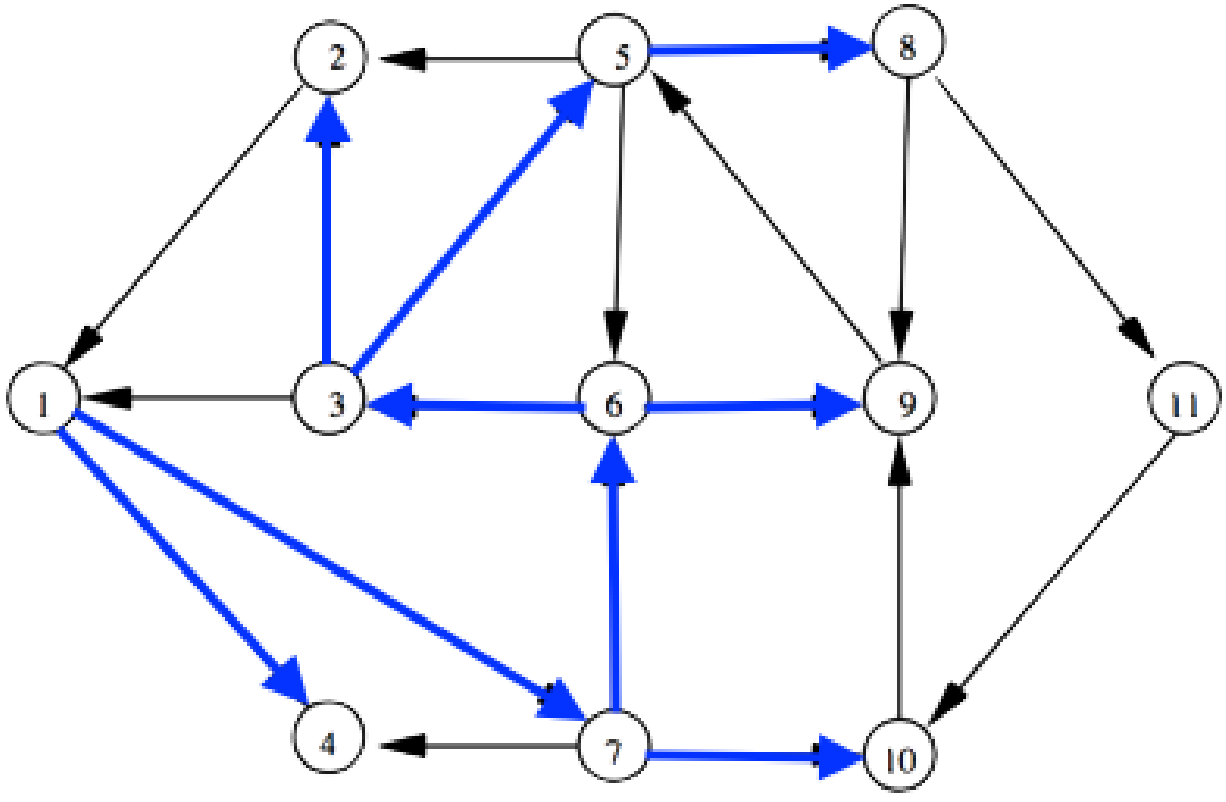
$Q = \{\ \not{1},\ \not{4},\ \not{7},\ \not{6},\ \not{10},\ \not{3},\ \not{9},\ 2,\ 5,\qquad\qquad\}$

Example



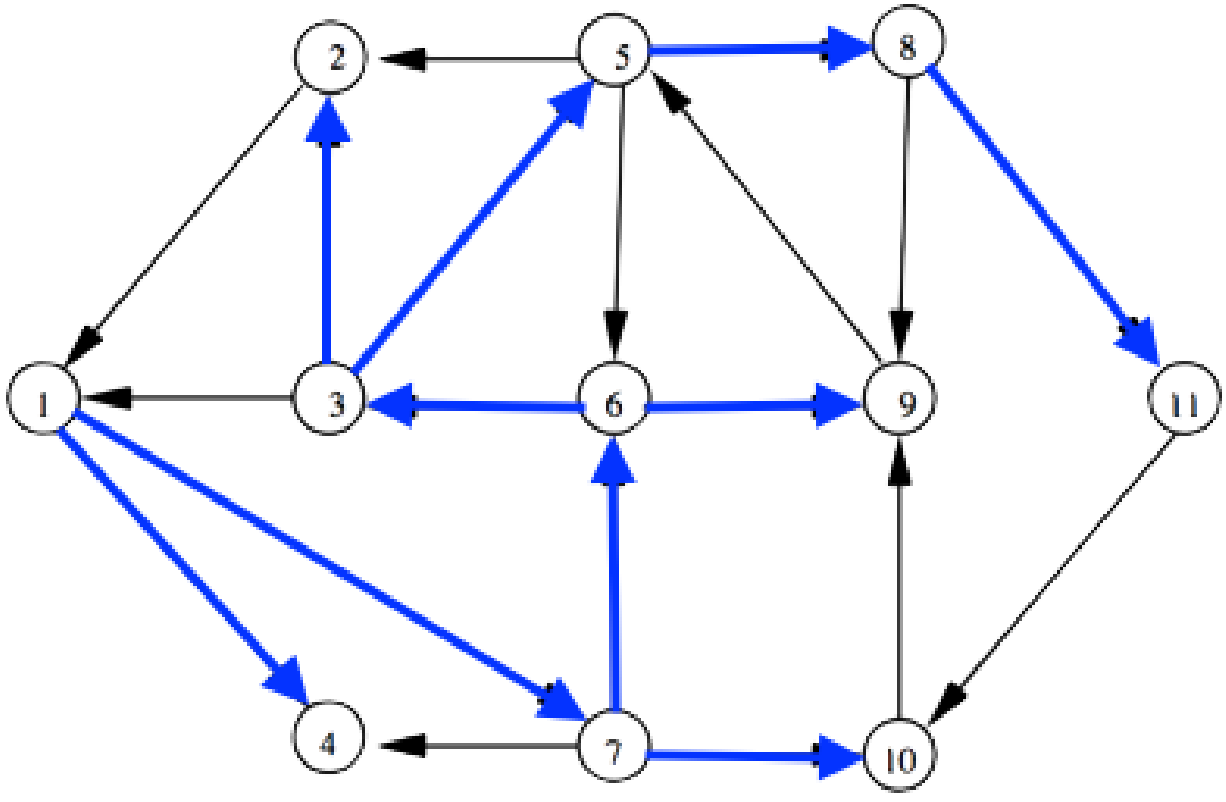
$Q = \{\ \not{1},\ \not{4},\ \not{7},\ \not{6},\ \not{10},\ \not{3},\ \not{9},\ \not{2},\ 5,\qquad\qquad\}$

Example



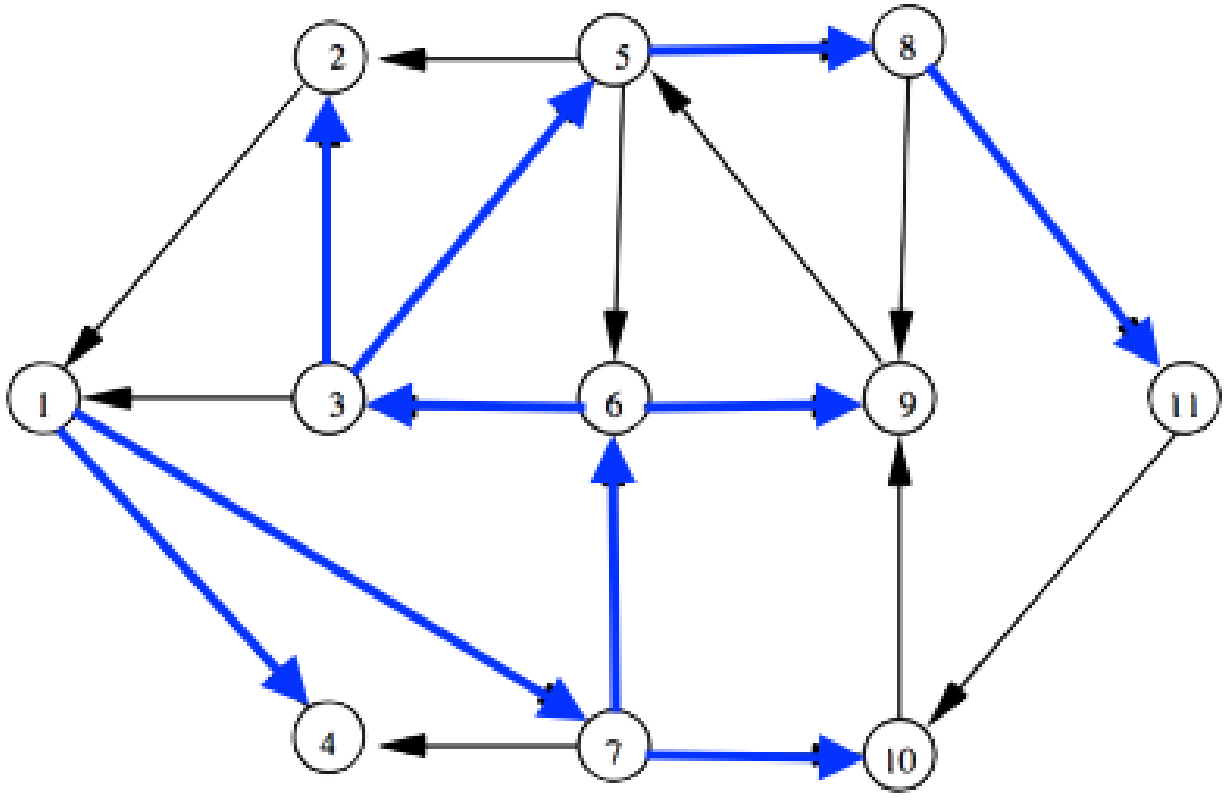
$$Q = \{\ \not{1},\ \not{4},\ \not{7},\ \not{6},\ \not{10},\ \not{3},\ \not{9},\ \not{2},\ \not{5},\ 8,\quad \}$$

Example



$$Q = \{\ 1,\ 4,\ 7,\ 6,\ 10,\ 3,\ 9,\ 2,\ 5,\ 8,\ 11\}$$

Example



$$Q = \{\ 1,\ 4,\ 7,\ 6,\ 10,\ 3,\ 9,\ 2,\ 5,\ 8,\ 11\}$$