Graph Algorithms

Peter Lammich

3. Februar 2020

Outline

Outline

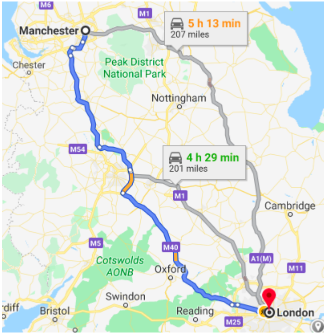
Outline

Motivation

- Shortest route between Manchester and London?

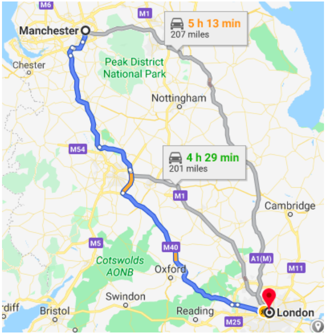
Motivation

- Shortest route between Manchester and London?



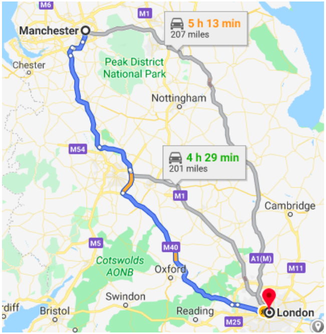
Motivation

- Shortest route between Manchester and London?
- Model map as graph: Nodes=Cities, Edges=Roads
    - Label each road with length (estimated travel time, ...)



Motivation

- Shortest route between Manchester and London?
- Model map as graph: Nodes=Cities, Edges=Roads
  - Label each road with length (estimated travel time, ...)
- Compute *shortest path* between two nodes



Formal Stuff

- Graph as *weight matrix* w
  - $E = \{(u, v) \mid w(u, v) \neq \infty\}$
  - $w(u, v) = d$ — Edge between u and v has weight d
  - $w(u, v) = \infty$ — No edge between u and v

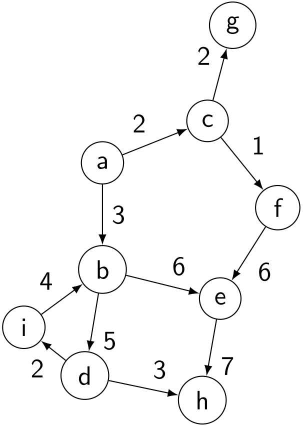
Formal Stuff

- Graph as *weight matrix* w
  - $E = \{(u, v) \mid w(u, v) \neq \infty\}$
  - $w(u, v) = d$ — Edge between u and v has weight d
  - $w(u, v) = \infty$ — No edge between u and v
- Recall: path p is list of nodes.
  - Path between u and v: upv
  - *Weight* of path: $|u_1 \ldots u_n| := \sum_{i=1}^{<n} w(u_i, u_{i+1})$
  - $|p| = \infty$ means path p not feasible!

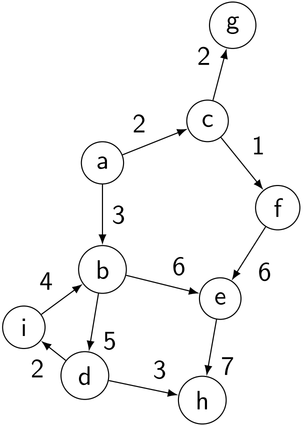
Formal Stuff

- Graph as *weight matrix* w
  - $E = \{(u, v) \mid w(u, v) \neq \infty\}$
  - $w(u, v) = d$ — Edge between u and v has weight d
  - $w(u, v) = \infty$ — No edge between u and v
- Recall: path p is list of nodes.
  - Path between u and v: upv
  - *Weight* of path: $|u_1 \ldots u_n| := \sum_{i=1}^{<n} w(u_i, u_{i+1})$
  - $|p| = \infty$ means path p not feasible!
- $\delta(u, v)$ — distance between u and v
  - $\delta(u, v) := \min |upv| \text{for all paths } p$
  - $\delta(u, v) = \infty$ — No path from u to v!

Example



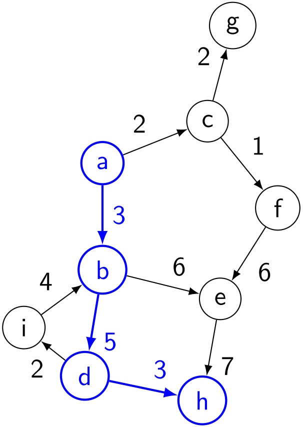
Example



Weight Matrix

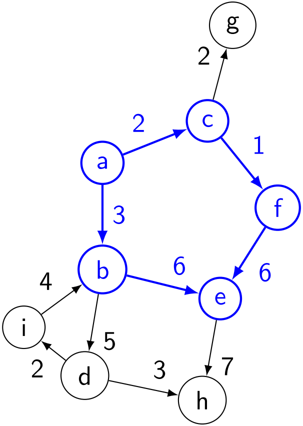
	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*
a	∞	3	2	∞	∞	∞	∞	∞	∞
b	∞	∞	∞	5	6	∞	∞	∞	∞
c	∞	∞	∞	∞	∞	1	2	∞	∞
d	∞	∞	∞	∞	∞	∞	∞	3	2
e	∞	∞	∞	∞	∞	∞	∞	7	∞
f	∞	∞	∞	∞	6	∞	∞	∞	∞
g	∞	∞	∞	∞	∞	∞	∞	∞	∞
h	∞	∞	∞	∞	∞	∞	∞	∞	∞
i	∞	4	∞	∞	∞	∞	∞	∞	∞

Example



Shortest path: *abdh*
Weight: $|abdh| = 11$
Distance: $\delta(a, h) = 11$ (weight of shortest path)

Example



Shortest path not always unique:
Shortest paths: *acfe*, *abe*
But distance is!
Weight: $|acfe| = |abe| = 9$
Distance: $\delta(a, h) = 9$ (weight of shortest path)

Setting the Scene

- Compute shortest paths from *start node s* to any node
  - Notation: $\delta(u) := \delta(s, u)$

Setting the Scene

- Compute shortest paths from *start node s* to any node
  - Notation: $\delta(u) := \delta(s, u)$
- Start with over-estimate D. $D(u) \geq \delta(u)$

Setting the Scene

- Compute shortest paths from *start node s* to any node
  - Notation: $\delta(u) := \delta(s, u)$
- Start with over-estimate D. $D(u) \geq \delta(u)$
  - $D(s) = 0$, $D(u) = \infty, u \neq s$

Setting the Scene

- Compute shortest paths from *start node s* to any node
  - Notation: $\delta(u) := \delta(s, u)$
- Start with over-estimate D. $D(u) \geq \delta(u)$
  - $D(s) = 0$, $D(u) = \infty, u \neq s$
- Improve estimate until precise

Setting the Scene

- Compute shortest paths from *start node s* to any node
  - Notation: $\delta(u) := \delta(s, u)$
- Start with over-estimate D. $D(u) \geq \delta(u)$
  - $D(s) = 0$, $D(u) = \infty, u \neq s$
- Improve estimate until precise
  - For edge (u, v) (*relax edge*)

procedure RELAX(u, v)
  if $D(u) + w(u, v) < D(v)$ **then**
    $D(v) \leftarrow D(u) + w(u, v)$

Setting the Scene

- Compute shortest paths from *start node s* to any node
  - Notation: $\delta(u) := \delta(s, u)$
- Start with over-estimate D. $D(u) \geq \delta(u)$
  - $D(s) = 0$, $D(u) = \infty, u \neq s$
- Improve estimate until precise
  - For edge (u, v) (*relax edge*)

    procedure RELAX(u, v)
        if $D(u) + w(u, v) < D(v)$ **then**
            $D(v) \leftarrow D(u) + w(u, v)$

    - If path over (u, v) better than current $D(v)$: adjust $D(v)$

Setting the Scene

- Compute shortest paths from *start node s* to any node
  - Notation: $\delta(u) := \delta(s, u)$
- Start with over-estimate D. $D(u) \geq \delta(u)$
  - $D(s) = 0$, $D(u) = \infty, u \neq s$
- Improve estimate until precise
  - For edge (u, v) (*relax edge*)

    procedure RELAX(u, v)
        if $D(u) + w(u, v) < D(v)$ **then**
            $D(v) \leftarrow D(u) + w(u, v)$

    - If path over (u, v) better than current $D(v)$: adjust $D(v)$
- Now: Strategies of relaxing edges, to reach precise estimate

Bellman Ford Algorithm

Assume no negative weight cycles exist. Negative weights are allowed!

Bellman Ford Algorithm

Assume no negative weight cycles exist. Negative weights are allowed!

procedure $\textsc{initEstimate}(s)$
    $D(u) \leftarrow \infty$ for all u
    $D(s) \leftarrow 0$ **return** D

procedure $\textsc{bellmanFord}(s)$
    $D \leftarrow \textsc{initEstimate}(s)$
    for $i \in 0.. < |V| - 1$ **do**
        for all (u, v) with $w(u, v) \neq \infty$ **do** $\textsc{relax}(u, v)$
    return D

- Relax each edge. Repeat (at most) $|V| - 1$ times.
    - If a round changes nothing, we can stop!

Bellman Ford Algorithm

Assume no negative weight cycles exist. Negative weights are allowed!

```
procedure initEstimate(s)
    D(u) ← ∞ for all u
    D(s) ← 0 return D
procedure bellmanFord(s)
    D ← initEstimate(s)
    for i ∈ 0..< |V| − 1 do
        for all (u, v) with w(u, v) ≠ ∞ do relax(u, v)
    return D
```

- Relax each edge. Repeat (at most) $|V| - 1$ times.
  - If a round changes nothing, we can stop!
- Claim: Returns $D = \delta$, i.e., precise estimate

Bellman Ford Algorithm

Assume no negative weight cycles exist. Negative weights are allowed!

procedure $\textsc{initEstimate}(s)$
  $D(u) \leftarrow \infty$ for all u
  $D(s) \leftarrow 0$ **return** D

procedure $\textsc{bellmanFord}(s)$
  $D \leftarrow \textsc{initEstimate}(\mathsf{s})$
  for $i \in 0..<|V|-1$ **do**
    for all (u, v) with $w(u, v) \neq \infty$ **do** $\textsc{relax}(u, v)$
  return D

- Relax each edge. Repeat (at most) $|V|-1$ times.
  - If a round changes nothing, we can stop!
- Claim: Returns $D = \delta$, i.e., precise estimate
- Idea: In step i, estimate for shortest path up to length i is precise

Bellman Ford Algorithm

Assume no negative weight cycles exist. Negative weights are allowed!

procedure $\textsc{initEstimate}(s)$
  $D(u) \leftarrow \infty$ for all u
  $D(s) \leftarrow 0$ **return** D

procedure $\textsc{bellmanFord}(s)$
  $D \leftarrow \textsc{initEstimate}(s)$
  for $i \in 0.. < |V| - 1$ **do**
    for all (u, v) with $w(u, v) \neq \infty$ **do** $\textsc{relax}(u, v)$
  return D

- Relax each edge. Repeat (at most) $|V| - 1$ times.
  - If a round changes nothing, we can stop!
- Claim: Returns $D = \delta$, i.e., precise estimate
- Idea: In step i, estimate for shortest path up to length i is precise
  - All shortest paths have length at most $|V| - 1$

Bellman Ford Algorithm

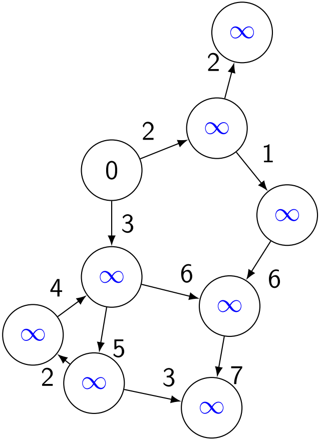
Assume no negative weight cycles exist. Negative weights are allowed!

procedure $\textsc{initEstimate}(s)$
  $D(u) \leftarrow \infty$ for all u
  $D(s) \leftarrow 0$ **return** D

procedure $\textsc{bellmanFord}(s)$
  $D \leftarrow \textsc{initEstimate}(s)$
  for $i \in 0.. < |V| - 1$ **do**
    for all (u, v) with $w(u, v) \neq \infty$ **do** $\textsc{relax}(u, v)$
  return D

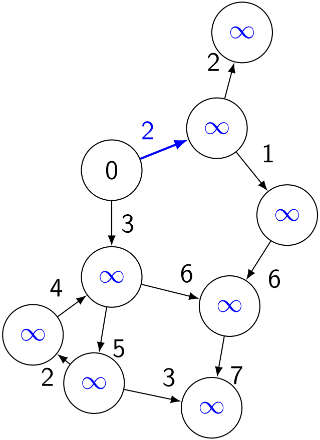
- Relax each edge. Repeat (at most) $|V| - 1$ times.
  - If a round changes nothing, we can stop!
- Claim: Returns $D = \delta$, i.e., precise estimate
- Idea: In step i, estimate for shortest path up to length i is precise
  - All shortest paths have length at most $|V| - 1$
  - Thus, D precise after algorithm

Example



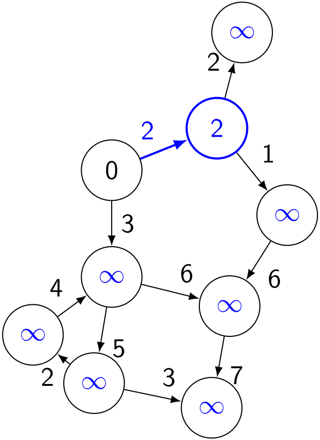
In each round, relax every edge once.

Example



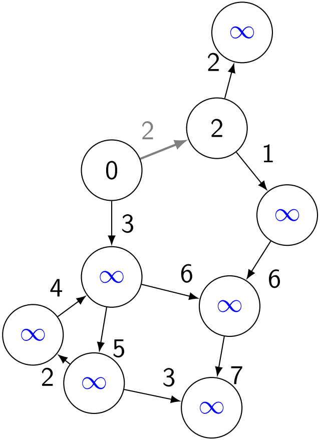
In each round, relax every edge once.

Example



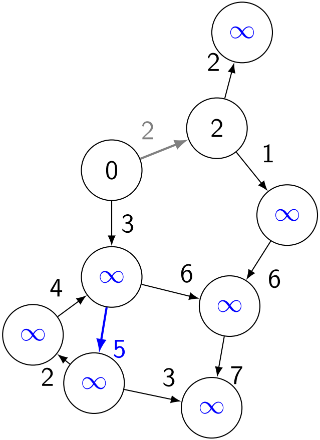
In each round, relax every edge once.

Example



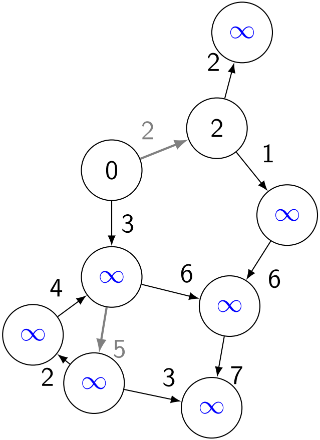
In each round, relax every edge once.

Example



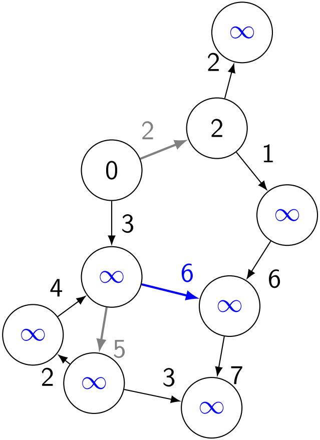
In each round, relax every edge once.

Example



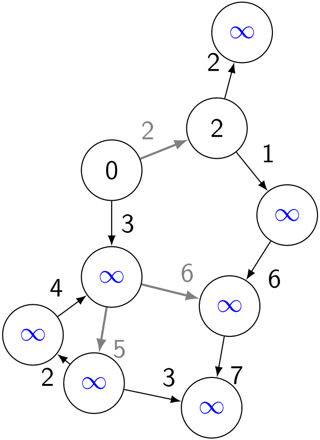
In each round, relax every edge once.

Example



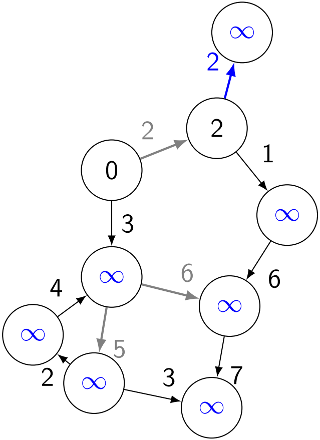
In each round, relax every edge once.

Example



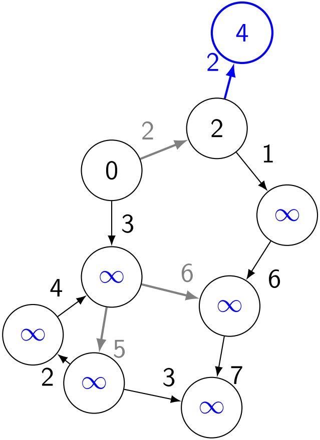
In each round, relax every edge once.

Example



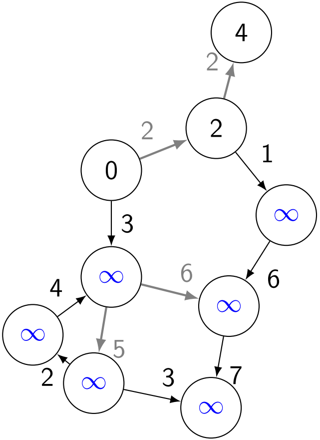
In each round, relax every edge once.

Example



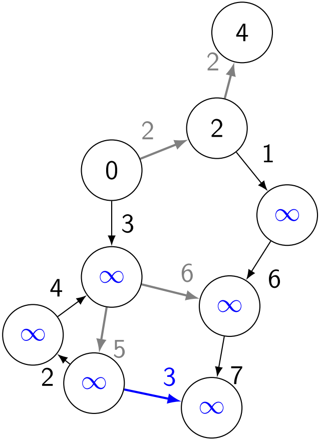
In each round, relax every edge once.

Example



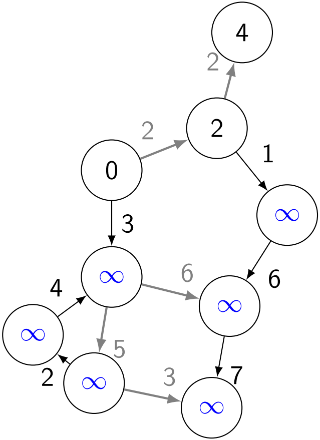
In each round, relax every edge once.

Example



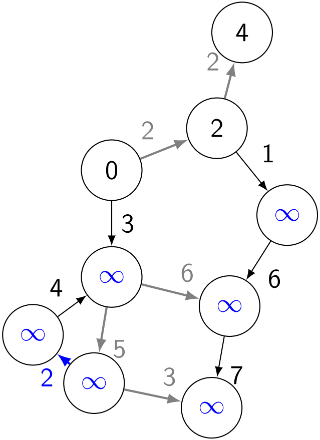
In each round, relax every edge once.

Example



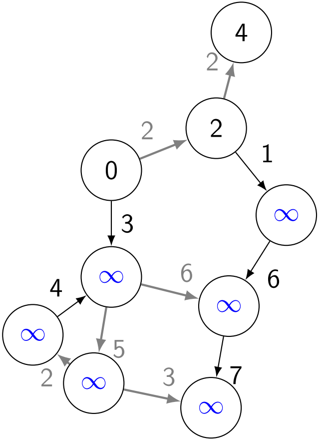
In each round, relax every edge once.

Example



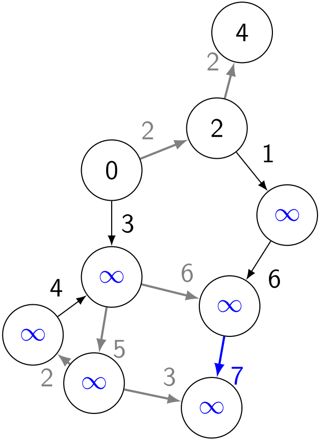
In each round, relax every edge once.

Example



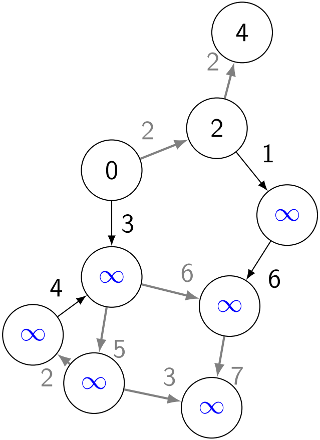
In each round, relax every edge once.

Example



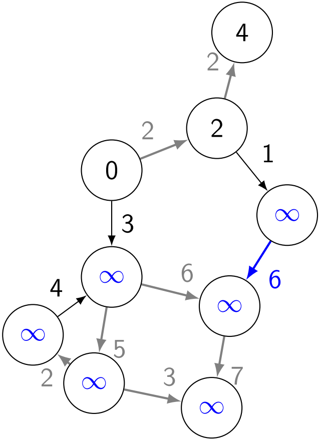
In each round, relax every edge once.

Example



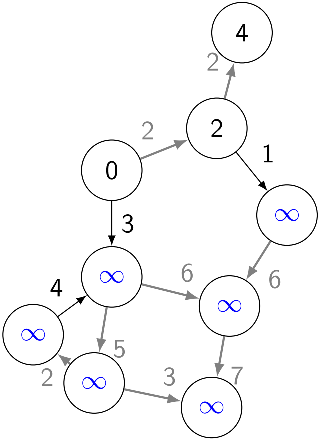
In each round, relax every edge once.

Example



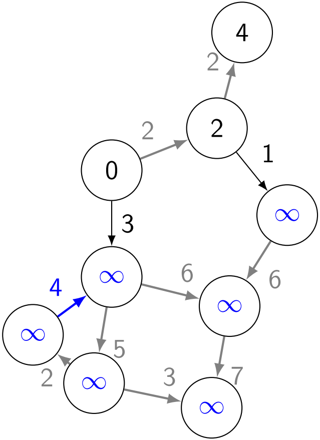
In each round, relax every edge once.

Example



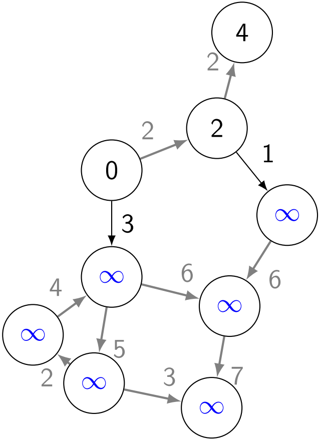
In each round, relax every edge once.

Example



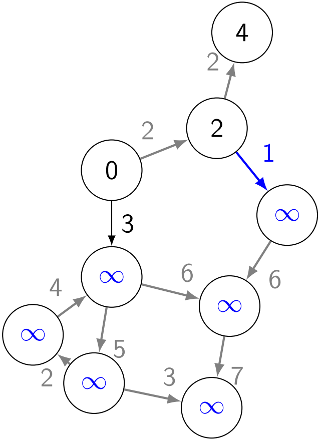
In each round, relax every edge once.

Example



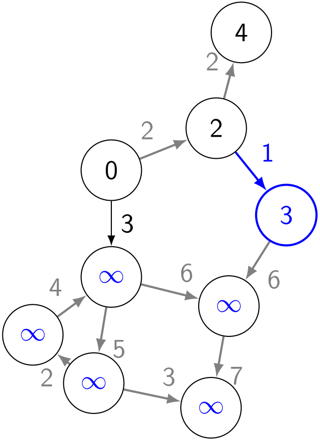
In each round, relax every edge once.

Example



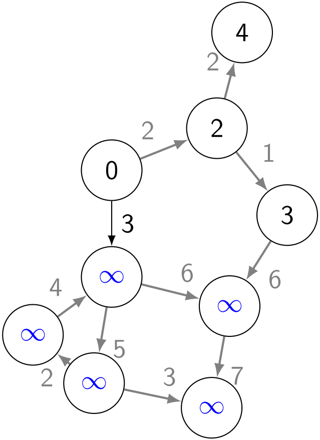
In each round, relax every edge once.

Example



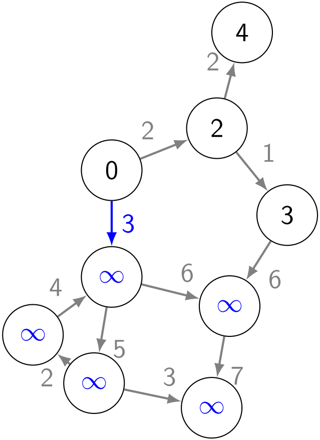
In each round, relax every edge once.

Example



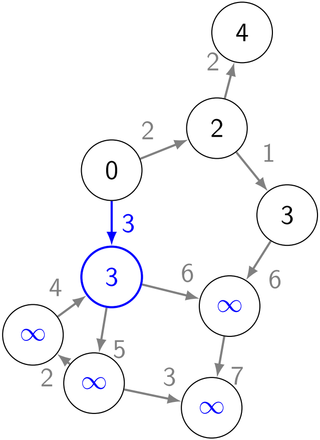
In each round, relax every edge once.

Example



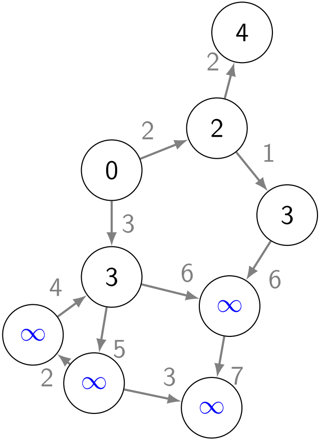
In each round, relax every edge once.

Example



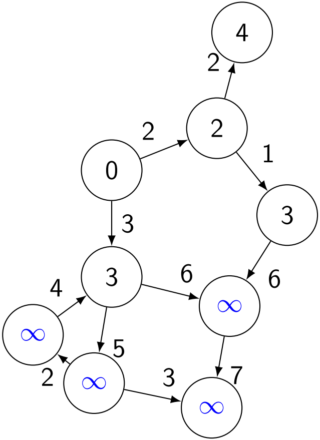
In each round, relax every edge once.

Example



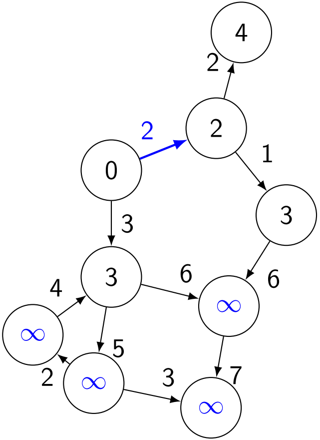
In each round, relax every edge once.

Example



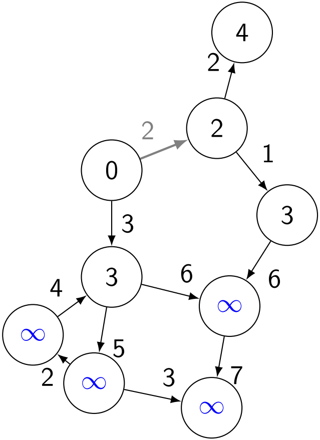
In each round, relax every edge once.
Next round ...

Example



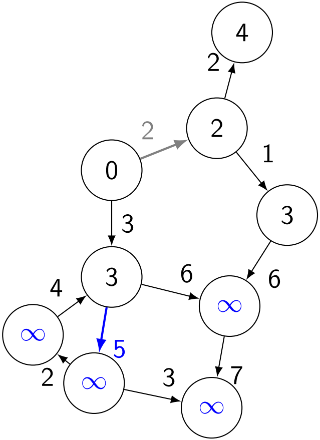
In each round, relax every edge once.

Example



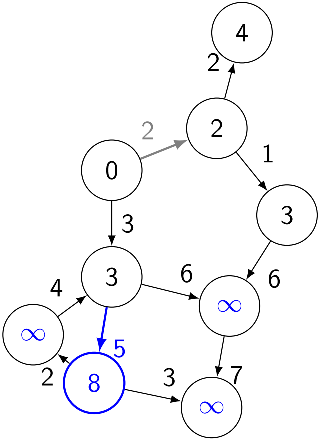
In each round, relax every edge once.

Example



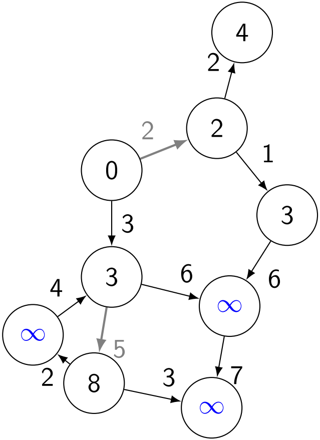
In each round, relax every edge once.

Example



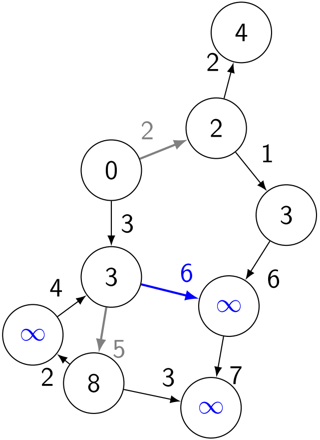
In each round, relax every edge once.

Example



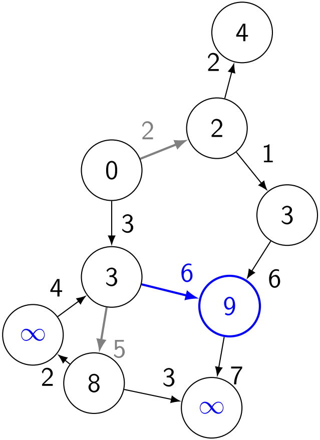
In each round, relax every edge once.

Example



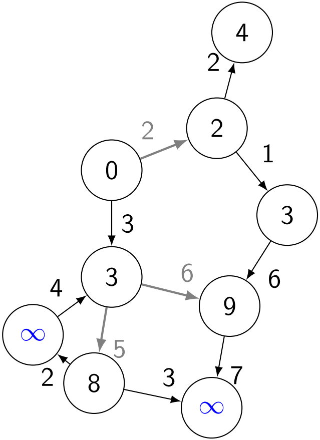
In each round, relax every edge once.

Example



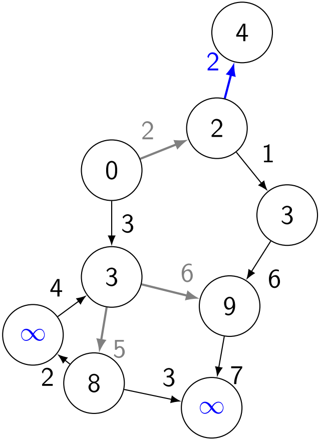
In each round, relax every edge once.

Example



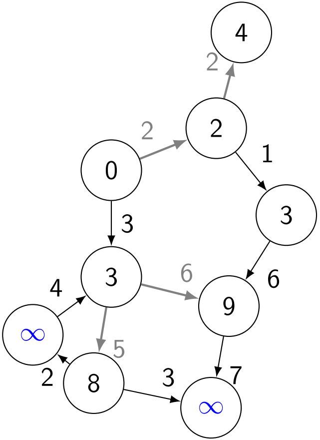
In each round, relax every edge once.

Example



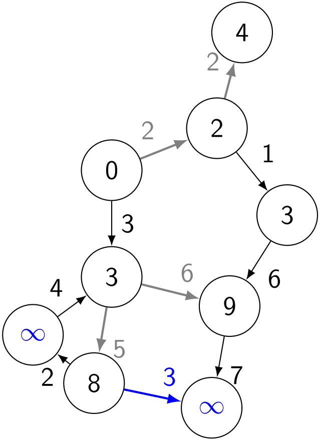
In each round, relax every edge once.

Example



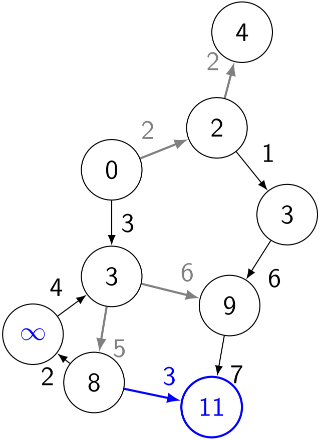
In each round, relax every edge once.

Example



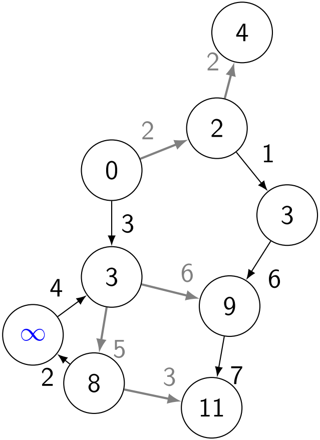
In each round, relax every edge once.

Example



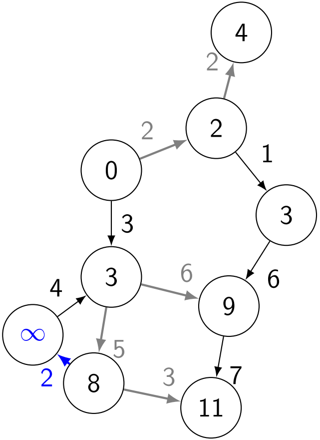
In each round, relax every edge once.

Example



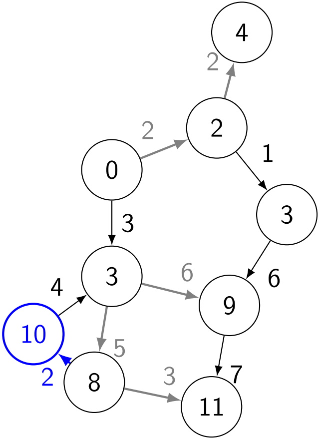
In each round, relax every edge once.

Example



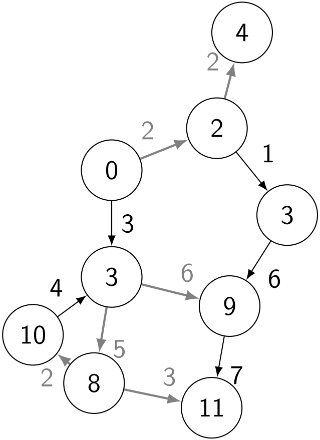
In each round, relax every edge once.

Example



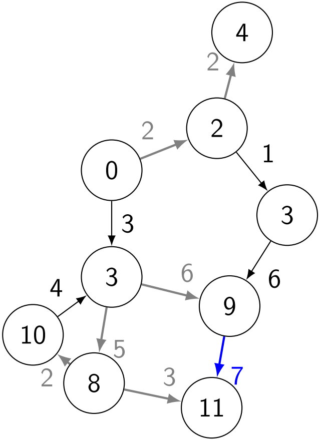
In each round, relax every edge once.

Example



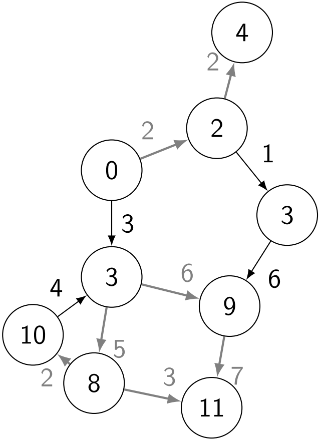
In each round, relax every edge once.

Example



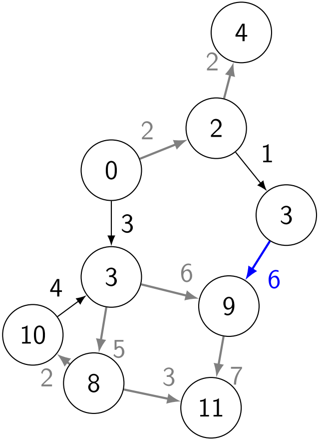
In each round, relax every edge once.

Example



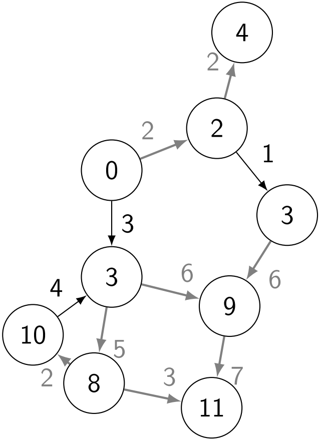
In each round, relax every edge once.

Example



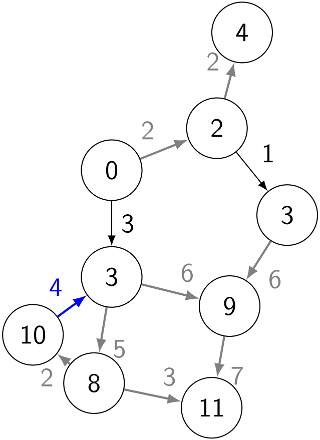
In each round, relax every edge once.

Example



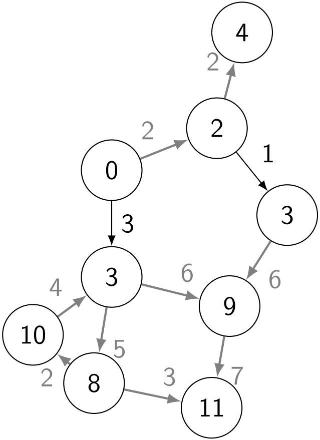
In each round, relax every edge once.

Example



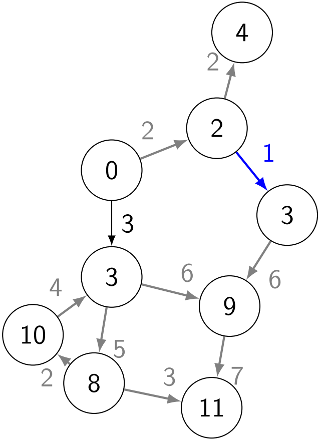
In each round, relax every edge once.

Example



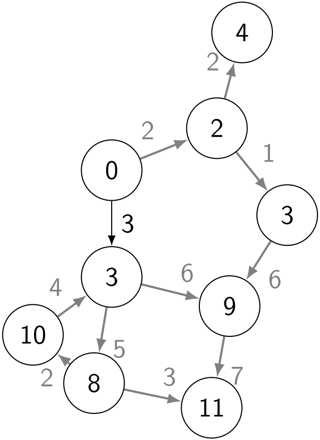
In each round, relax every edge once.

Example



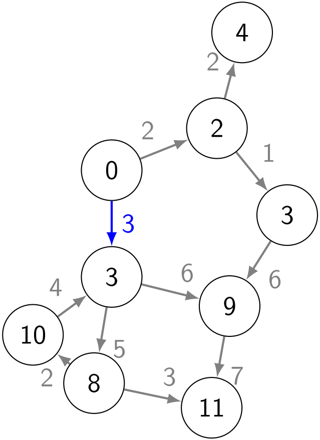
In each round, relax every edge once.

Example



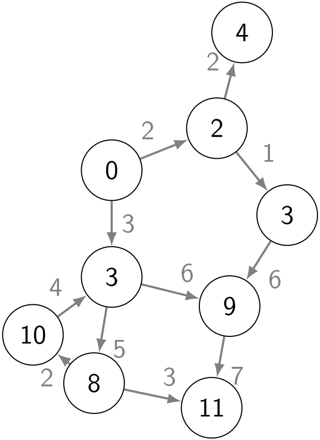
In each round, relax every edge once.

Example



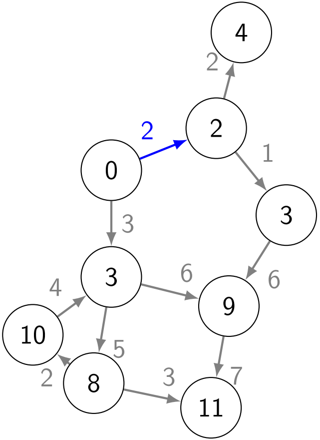
In each round, relax every edge once.

Example



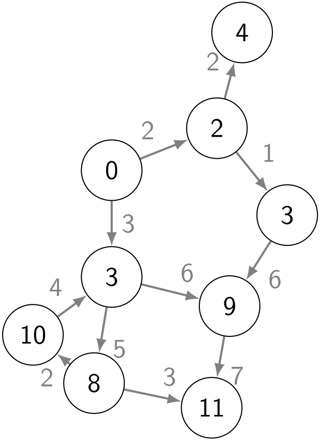
In each round, relax every edge once.
Next round ...

Example



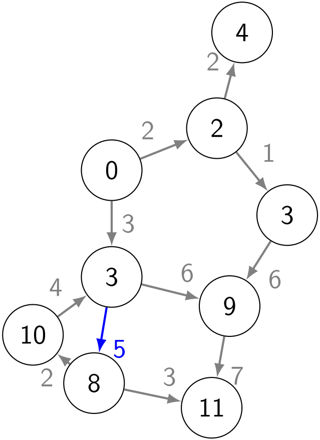
In each round, relax every edge once.

Example



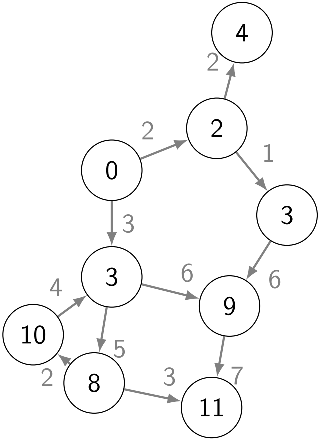
In each round, relax every edge once.

Example



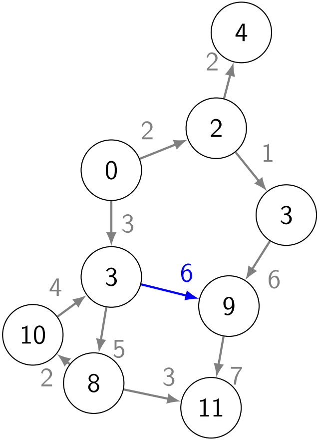
In each round, relax every edge once.

Example



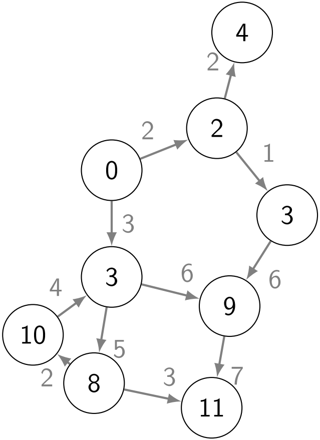
In each round, relax every edge once.

Example



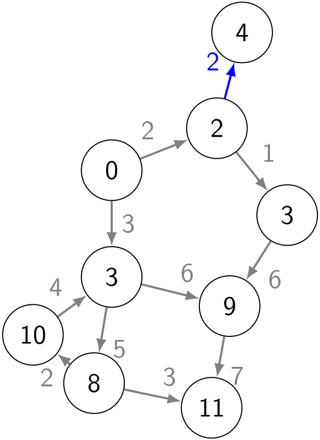
In each round, relax every edge once.

Example



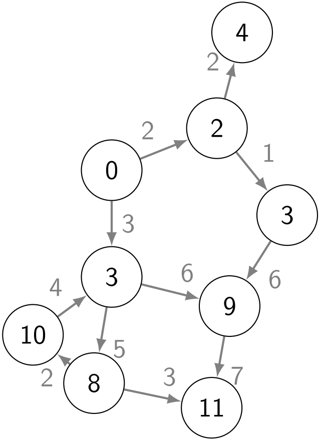
In each round, relax every edge once.

Example



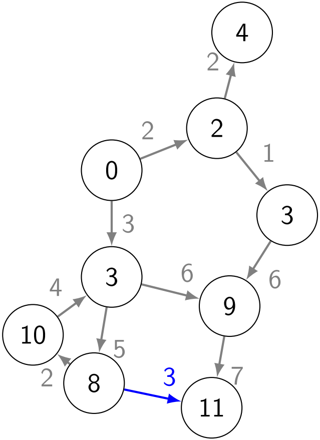
In each round, relax every edge once.

Example



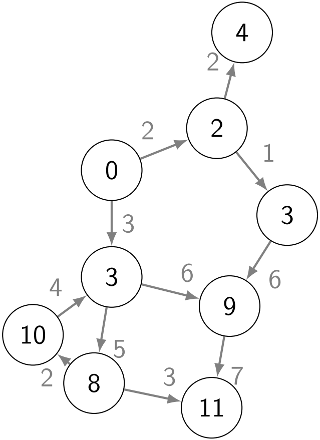
In each round, relax every edge once.

Example



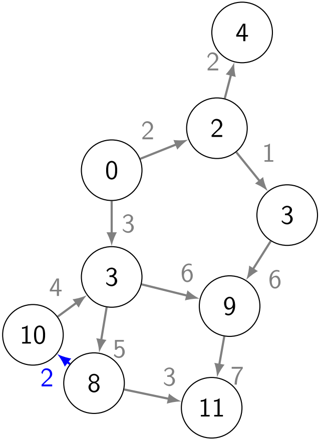
In each round, relax every edge once.

Example



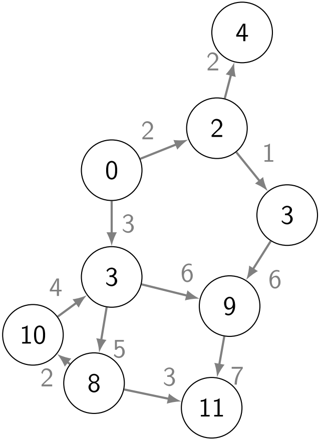
In each round, relax every edge once.

Example



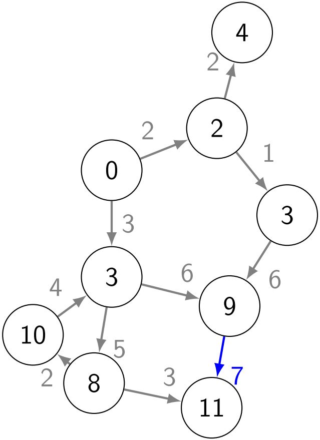
In each round, relax every edge once.

Example



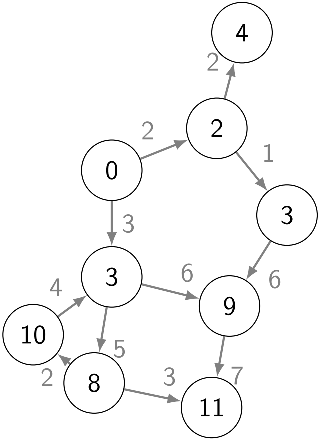
In each round, relax every edge once.

Example



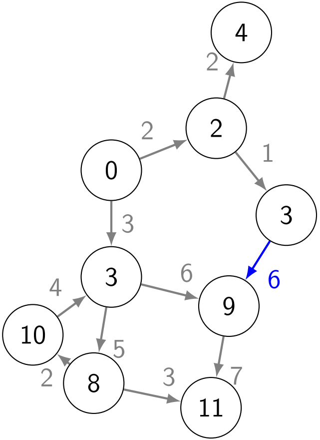
In each round, relax every edge once.

Example



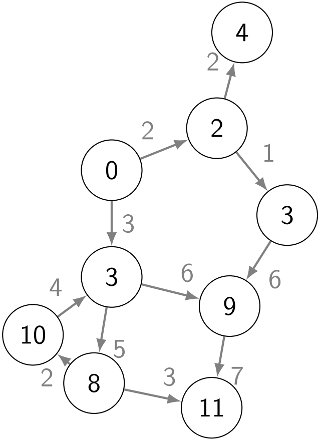
In each round, relax every edge once.

Example



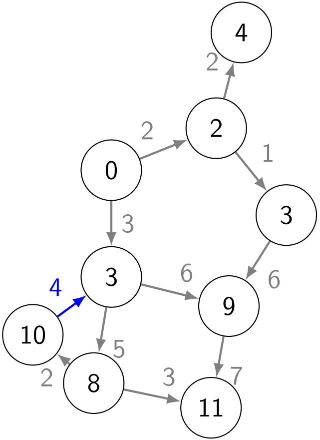
In each round, relax every edge once.

Example



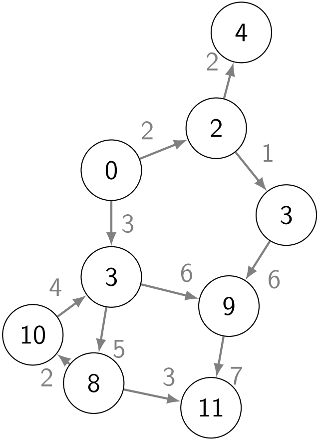
In each round, relax every edge once.

Example



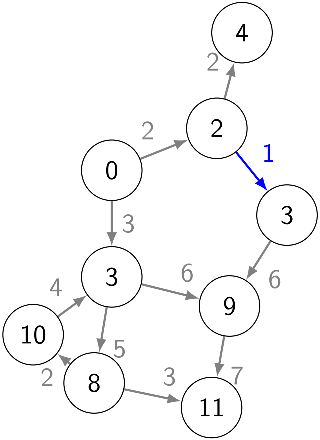
In each round, relax every edge once.

Example



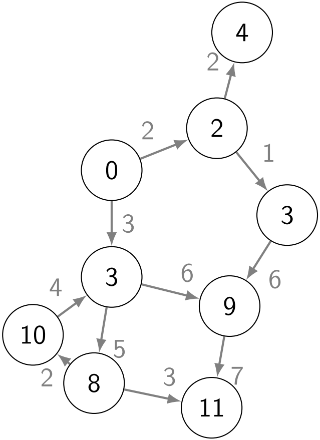
In each round, relax every edge once.

Example



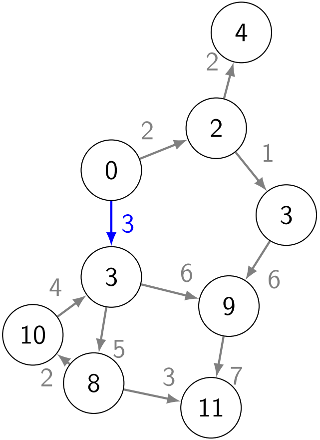
In each round, relax every edge once.

Example



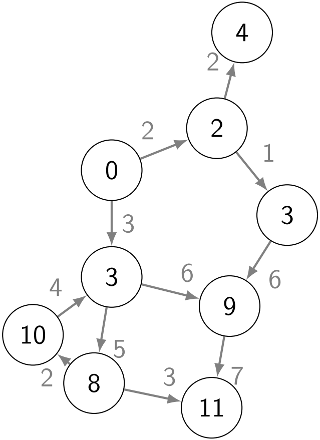
In each round, relax every edge once.

Example



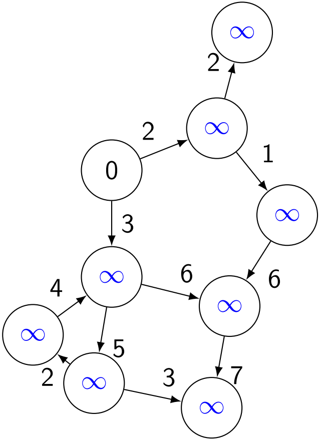
In each round, relax every edge once.

Example



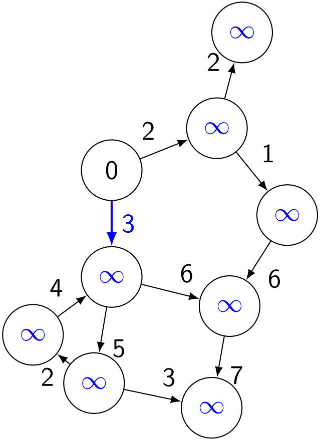
In each round, relax every edge once.
Round changed nothing: terminate

Example



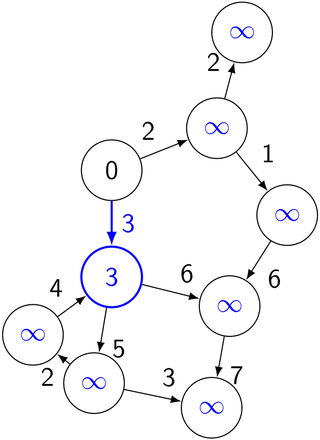
Order of edges can affects number of required rounds!

Example



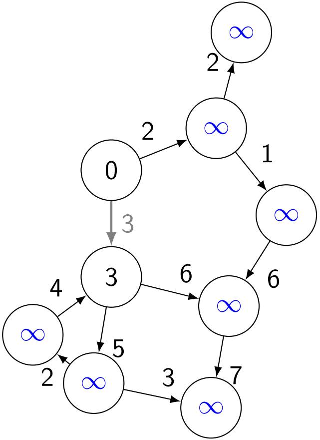
Order of edges can affects number of required rounds!

Example



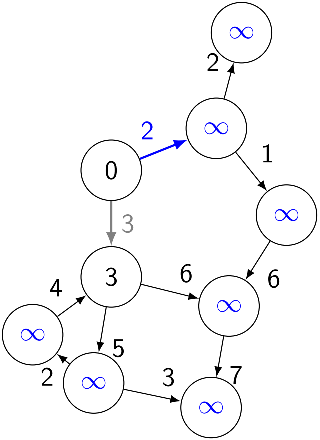
Order of edges can affects number of required rounds!

Example



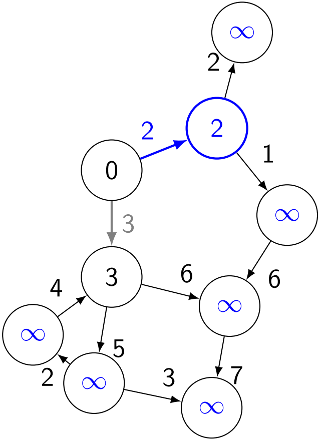
Order of edges can affects number of required rounds!

Example



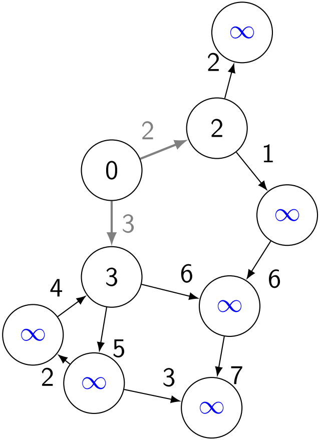
Order of edges can affects number of required rounds!

Example



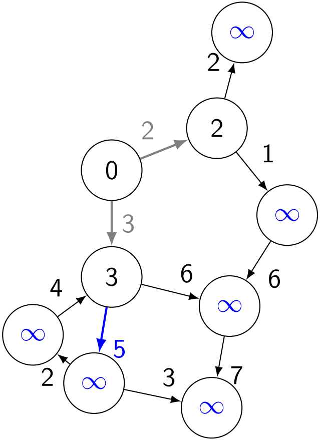
Order of edges can affects number of required rounds!

Example



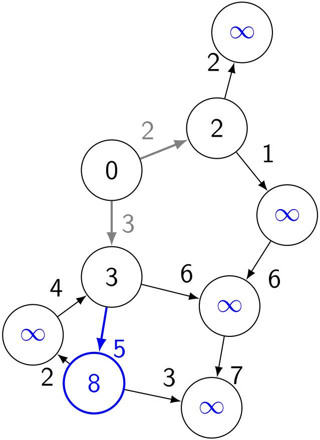
Order of edges can affects number of required rounds!

Example



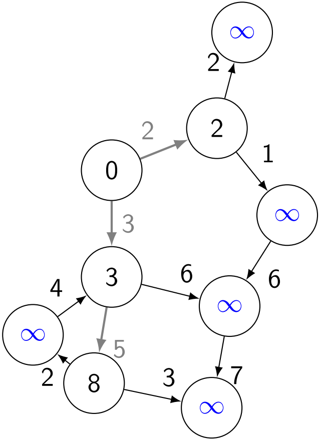
Order of edges can affects number of required rounds!

Example



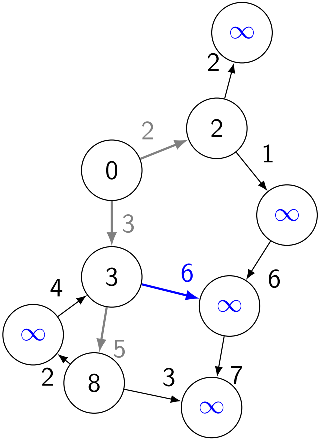
Order of edges can affects number of required rounds!

Example



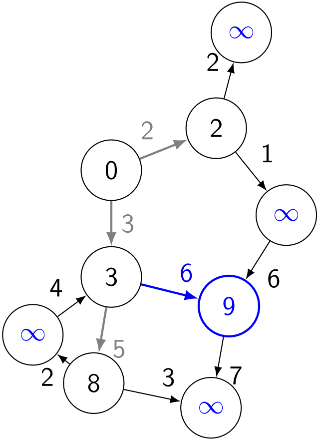
Order of edges can affects number of required rounds!

Example



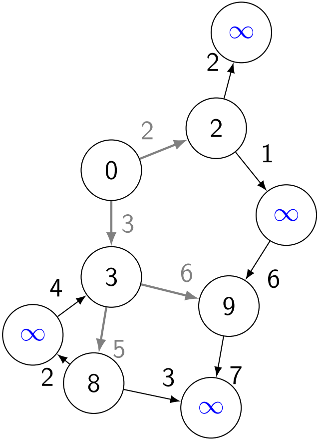
Order of edges can affects number of required rounds!

Example



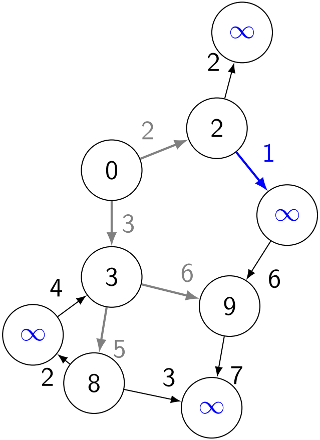
Order of edges can affects number of required rounds!

Example



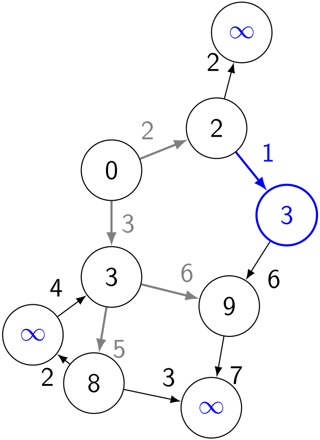
Order of edges can affects number of required rounds!

Example



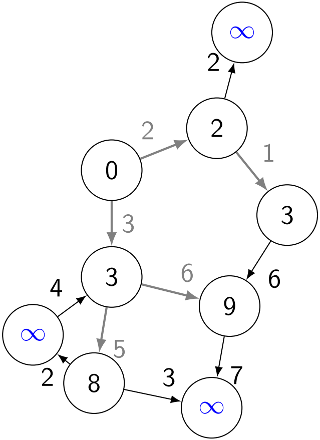
Order of edges can affects number of requi-
red rounds!

Example



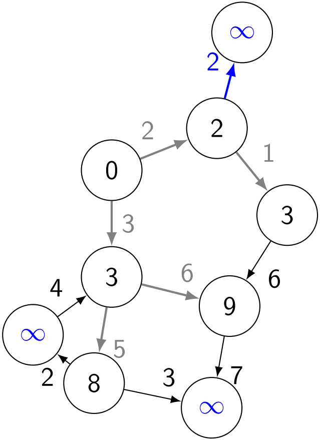
Order of edges can affects number of required rounds!

Example



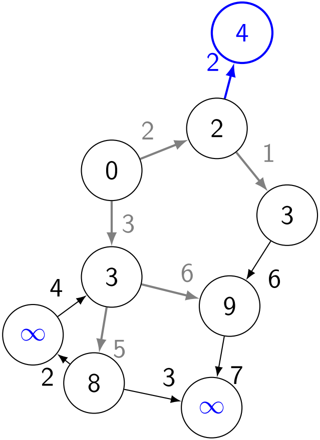
Order of edges can affects number of required rounds!

Example



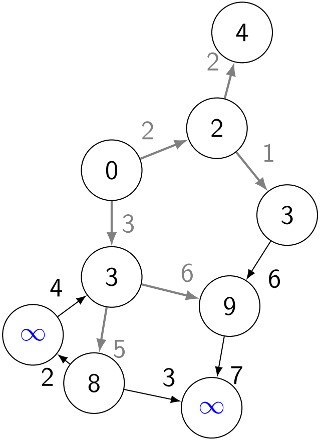
Order of edges can affects number of required rounds!

Example



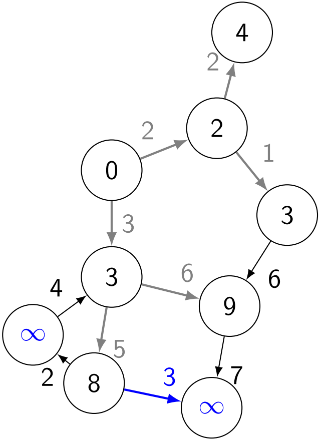
Order of edges can affects number of required rounds!

Example



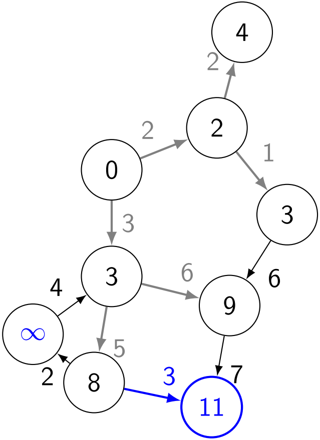
Order of edges can affects number of required rounds!

Example



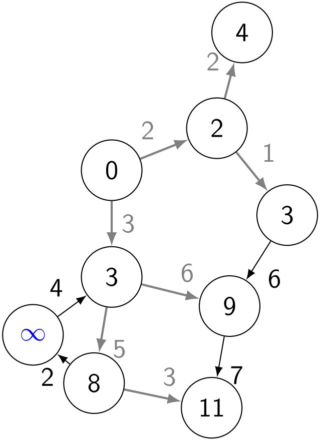
Order of edges can affects number of required rounds!

Example



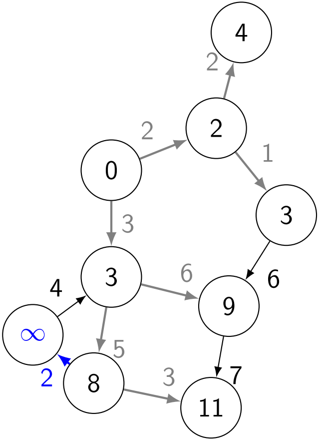
Order of edges can affects number of required rounds!

Example



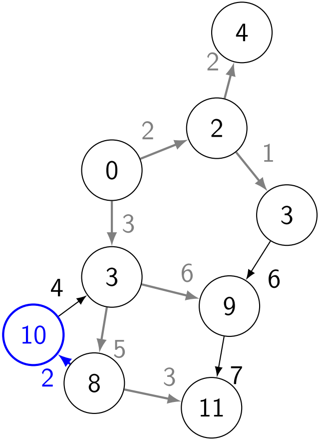
Order of edges can affects number of requi-red rounds!

Example



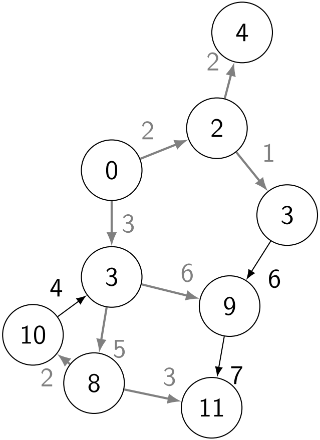
Order of edges can affects number of required rounds!

Example



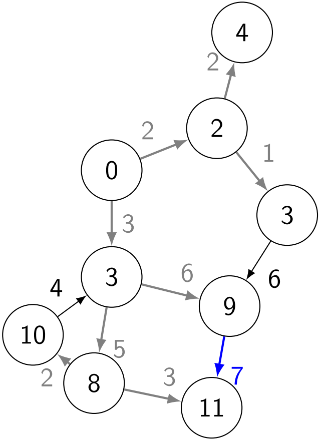
Order of edges can affects number of required rounds!

Example



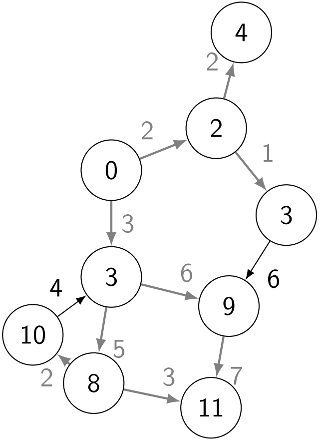
Order of edges can affects number of required rounds!

Example



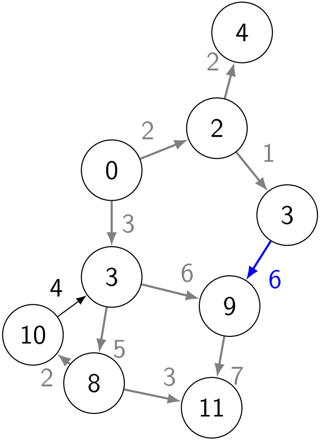
Order of edges can affects number of required rounds!

Example



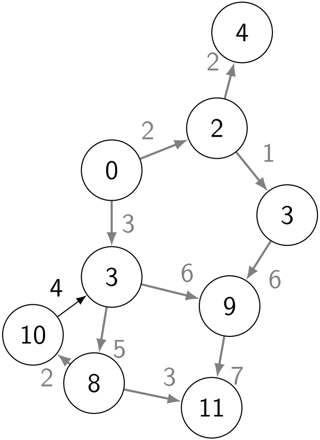
Order of edges can affects number of required rounds!

Example



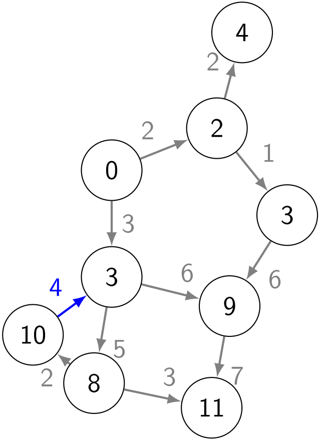
Order of edges can affects number of required rounds!

Example



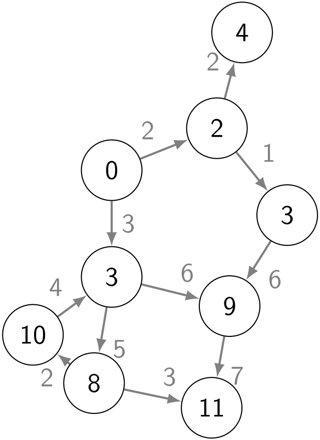
Order of edges can affects number of required rounds!

Example



Order of edges can affects number of required rounds!

Example



Order of edges can affects number of required rounds!
Nothing will change any more ...

Excursion: Loop Invariant

- A *loop invariant* is a statement that
  1. holds initially
  2. is *preserved* by loop iteration
  3. implies correctness when loop terminates

Excursion: Loop Invariant

- A *loop invariant* is a statement that
  1. holds initially
  2. is *preserved* by loop iteration
  3. implies correctness when loop terminates

In round i, estimate for shortest paths up to length i is precise and $\forall u \in V.\ D(u) \geq \delta(u)$

Excursion: Loop Invariant

- A *loop invariant* is a statement that
  1. holds initially
  2. is *preserved* by loop iteration
  3. implies correctness when loop terminates

In round i, estimate for shortest paths up to length i is precise and $\forall u \in V.\ D(u) \geq \delta(u)$

1. Initially, D precise up to length 0 (we have $D(s) = 0$)

Excursion: Loop Invariant

- A *loop invariant* is a statement that
  1. holds initially
  2. is *preserved* by loop iteration
  3. implies correctness when loop terminates

In round i, estimate for shortest paths up to length i is precise and $\forall u \in V.\ D(u) \geq \delta(u)$

1. Initially, D precise up to length 0 (we have $D(s) = 0$)
2. Assume D precise up to length i.

Excursion: Loop Invariant

- A *loop invariant* is a statement that
  1. holds initially
  2. is *preserved* by loop iteration
  3. implies correctness when loop terminates

In round i, estimate for shortest paths up to length i is precise and $\forall u \in V.\ D(u) \geq \delta(u)$

1. Initially, D precise up to length 0 (we have $D(s) = 0$)
2. Assume D precise up to length i.
   Show: After one more round, precise up to length $i + 1$

Excursion: Loop Invariant

- A *loop invariant* is a statement that
  1. holds initially
  2. is *preserved* by loop iteration
  3. implies correctness when loop terminates

In round i, estimate for shortest paths up to length i is precise and $\forall u \in V.\ D(u) \geq \delta(u)$

1. Initially, D precise up to length 0 (we have $D(s) = 0$)
2. Assume D precise up to length i.
   Show: After one more round, precise up to length $i + 1$
   prefix of shortest path is shortest path

Excursion: Loop Invariant

- A *loop invariant* is a statement that
  1. holds initially
  2. is *preserved* by loop iteration
  3. implies correctness when loop terminates

In round i, estimate for shortest paths up to length i is precise and $\forall u \in V.\ D(u) \geq \delta(u)$

1. Initially, D precise up to length 0 (we have $D(s) = 0$)
2. Assume D precise up to length i.
   Show: After one more round, precise up to length $i + 1$
   prefix of shortest path is shortest path
   assume: $spuv$ is s.p. of length $i + 1$. Thus spu is s.p. of length i

Excursion: Loop Invariant

- A *loop invariant* is a statement that
  1. holds initially
  2. is *preserved* by loop iteration
  3. implies correctness when loop terminates

In round i, estimate for shortest paths up to length i is precise and $\forall u \in V.\ D(u) \geq \delta(u)$

1. Initially, D precise up to length 0 (we have $D(s) = 0$)
2. Assume D precise up to length i.
   Show: After one more round, precise up to length $i + 1$
   prefix of shortest path is shortest path
   assume: $spuv$ is s.p. of length $i + 1$. Thus spu is s.p. of length i
   thus, $D(u)$ precise, and round relaxes edge (u, v)

Excursion: Loop Invariant

- A *loop invariant* is a statement that
  1. holds initially
  2. is *preserved* by loop iteration
  3. implies correctness when loop terminates

In round i, estimate for shortest paths up to length i is precise and $\forall u \in V.\ D(u) \geq \delta(u)$

1. Initially, D precise up to length 0 (we have $D(s) = 0$)
2. Assume D precise up to length i.
   Show: After one more round, precise up to length $i + 1$
   prefix of shortest path is shortest path
   assume: $spuv$ is s.p. of length $i + 1$. Thus spu is s.p. of length i
   thus, $D(u)$ precise, and round relaxes edge (u, v)
   thus, $D(v)$ precise after round

Excursion: Loop Invariant

- A *loop invariant* is a statement that
  1. holds initially
  2. is *preserved* by loop iteration
  3. implies correctness when loop terminates

In round i, estimate for shortest paths up to length i is precise and $\forall u \in V.\ D(u) \geq \delta(u)$

1. Initially, D precise up to length 0 (we have $D(s) = 0$)
2. Assume D precise up to length i.
   Show: After one more round, precise up to length $i+1$
   prefix of shortest path is shortest path
   assume: $spuv$ is s.p. of length $i+1$. Thus spu is s.p. of length i
   thus, $D(u)$ precise, and round relaxes edge (u,v)
   thus, $D(v)$ precise after round
3. Assume path up to length $|V|-1$ precise

Excursion: Loop Invariant

- A *loop invariant* is a statement that
  1. holds initially
  2. is *preserved* by loop iteration
  3. implies correctness when loop terminates

In round i, estimate for shortest paths up to length i is precise and $\forall u \in V.\ D(u) \geq \delta(u)$

1. Initially, D precise up to length 0 (we have $D(s) = 0$)
2. Assume D precise up to length i.
   Show: After one more round, precise up to length $i + 1$
   prefix of shortest path is shortest path
   assume: *spuv* is s.p. of length $i + 1$. Thus *spu* is s.p. of length i
   thus, $D(u)$ precise, and round relaxes edge (u, v)
   thus, $D(v)$ precise after round
3. Assume path up to length $|V| - 1$ precise
   as no negative-weight cycles exist: any shortest path is cycle free

Excursion: Loop Invariant

- A *loop invariant* is a statement that
  1. holds initially
  2. is *preserved* by loop iteration
  3. implies correctness when loop terminates

In round i, estimate for shortest paths up to length i is precise and $\forall u \in V.\ D(u) \geq \delta(u)$

1. Initially, D precise up to length 0 (we have $D(s) = 0$)
2. Assume D precise up to length i.
   Show: After one more round, precise up to length $i + 1$
   prefix of shortest path is shortest path
   assume: *spuv* is s.p. of length $i + 1$. Thus *spu* is s.p. of length i
   thus, $D(u)$ precise, and round relaxes edge (u, v)
   thus, $D(v)$ precise after round
3. Assume path up to length $|V| - 1$ precise
   as no negative-weight cycles exist: any shortest path is cycle free
   thus, length at most $|V| - 1$

Negative Weight Cycles

- What if negative weight cycle exists?

Negative Weight Cycles

- What if negative weight cycle exists?
- No shortest paths to nodes reachable from cycle!

Negative Weight Cycles

- What if negative weight cycle exists?
- No shortest paths to nodes reachable from cycle!
- Bellman-Ford can detect this:
  - Iterate until D does not change.
  - If D still changed in $|V|$th round: Report negative cycle

Complexity

- In worst case, we do $|V|$ rounds
- Each round inspects $|E|$ edges
- Time for relaxing edge: $O(1)$
- Complexity is $O(|V||E|)$