ITMO UNIVERSITY

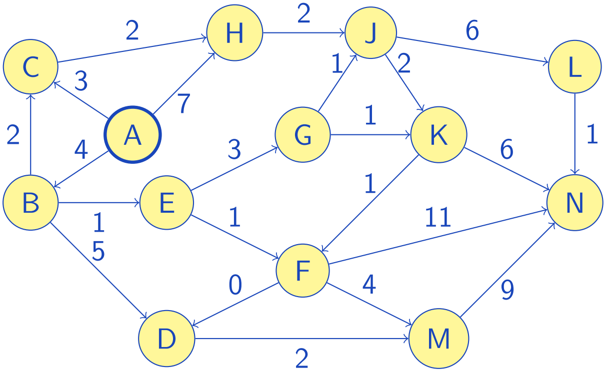
How to Win Coding Competitions: Secrets of Champions

Week 4: Algorithms on Graphs
Lecture 9: Single Source Shortest Paths

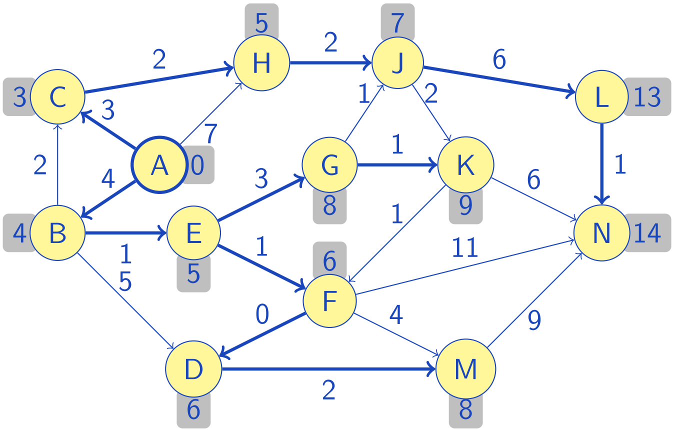
Maxim Buzdalov
Saint Petersburg 2016

Problem: for every vertex determine a shortest path from v_0

Problem: for every vertex determine a shortest path from v_0



Problem: for every vertex determine a shortest path from v_0



Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate

Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

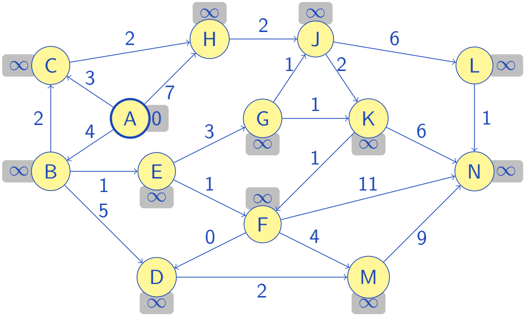
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example.



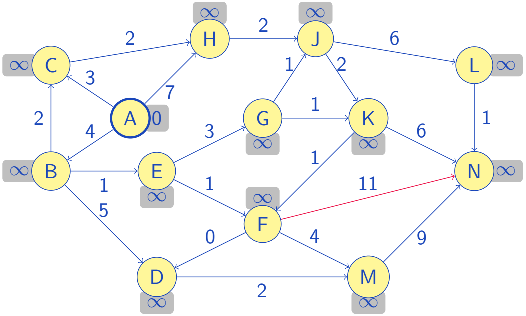
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



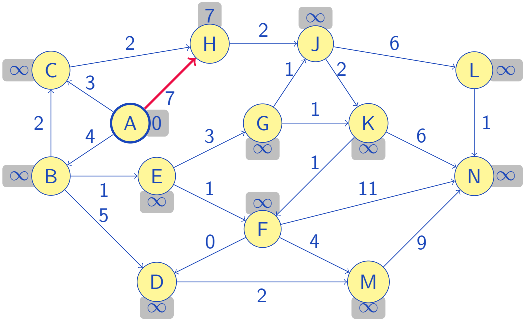
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



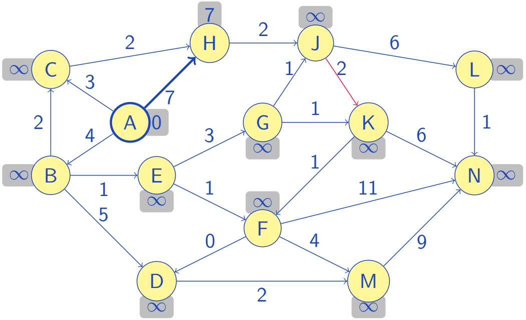
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



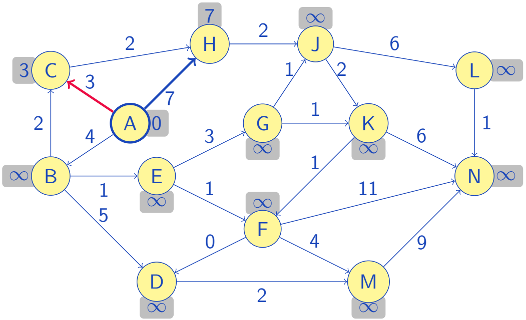
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



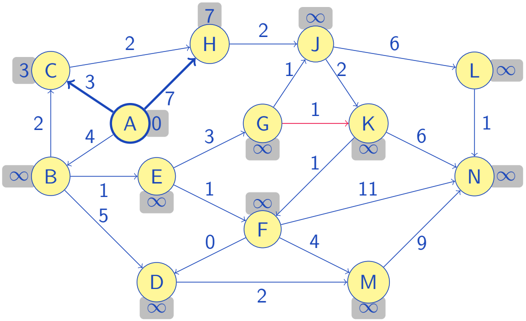
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



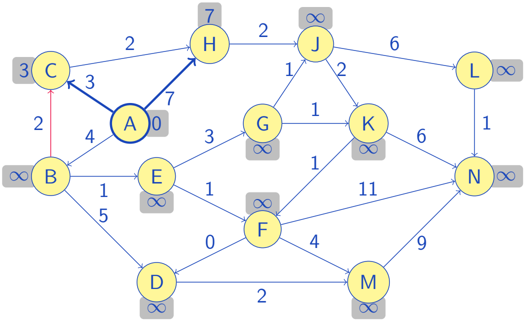
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



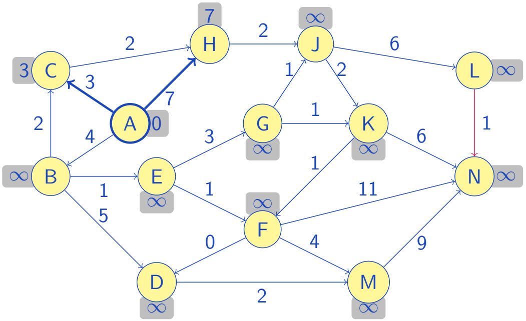
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



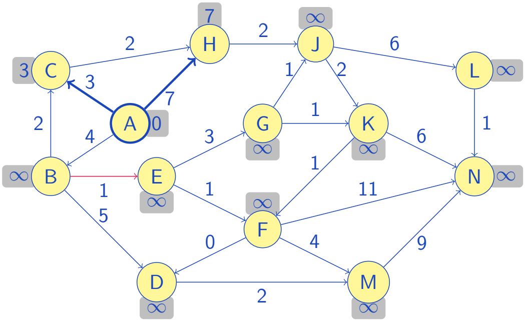
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



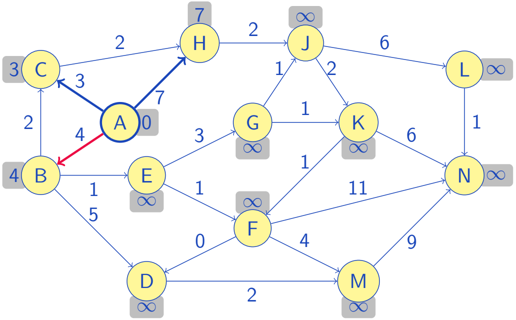
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



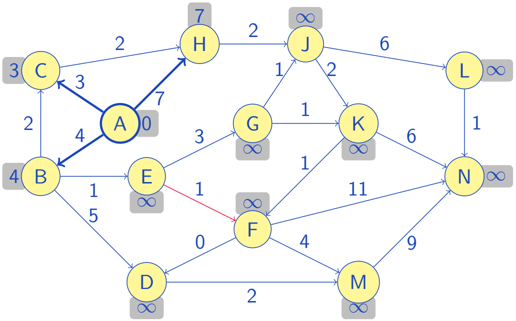
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



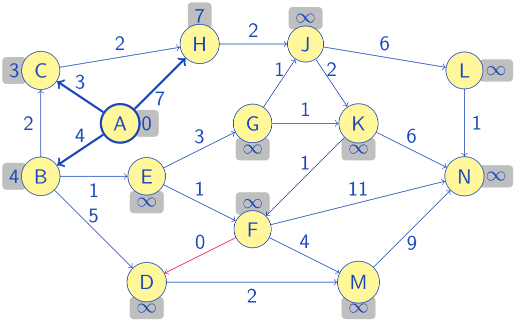
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



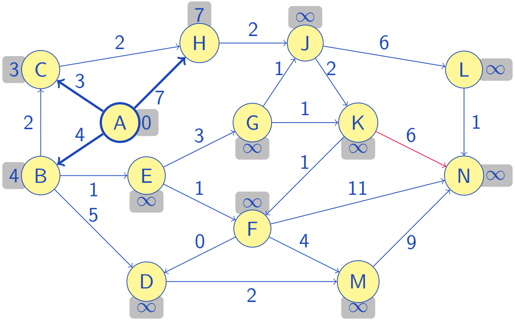
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



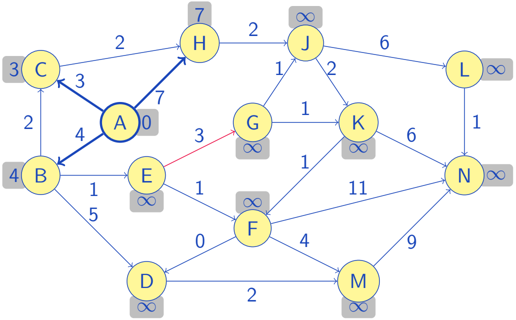
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



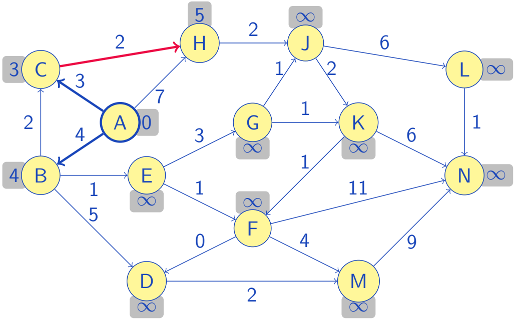
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



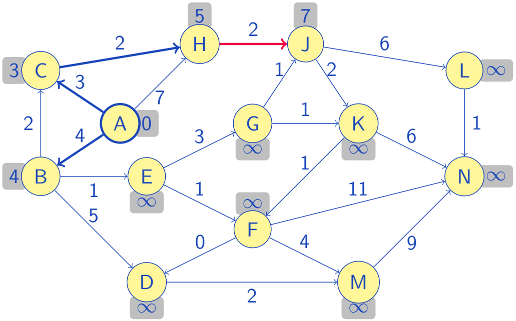
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



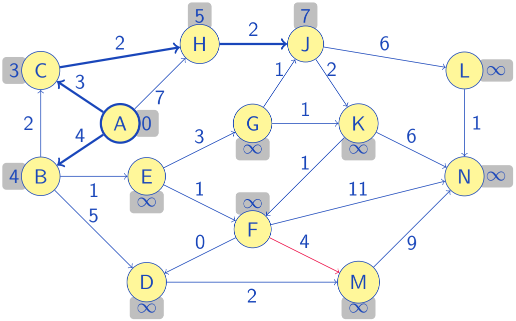
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



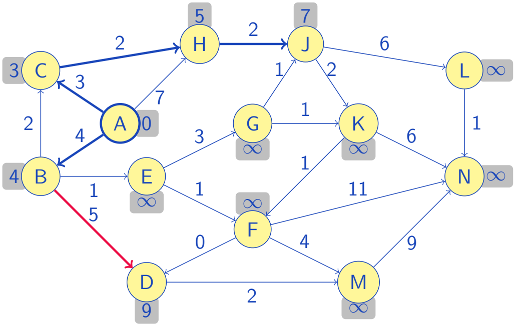
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



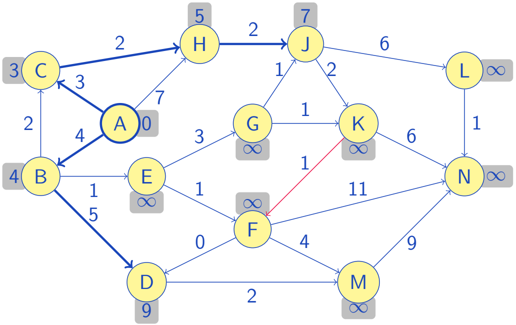
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



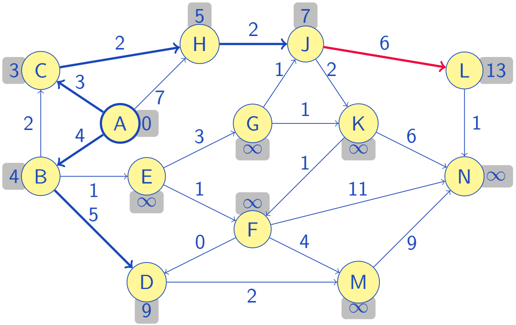
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



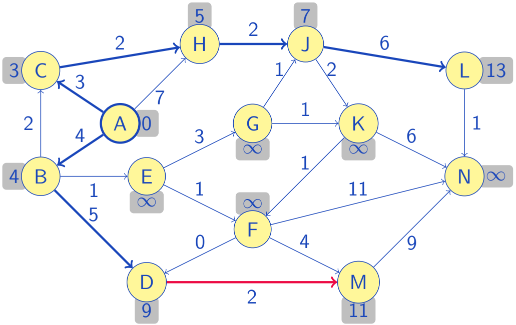
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



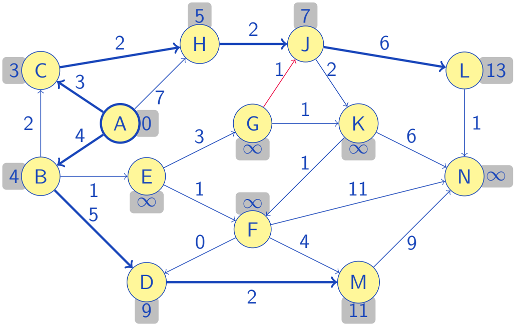
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



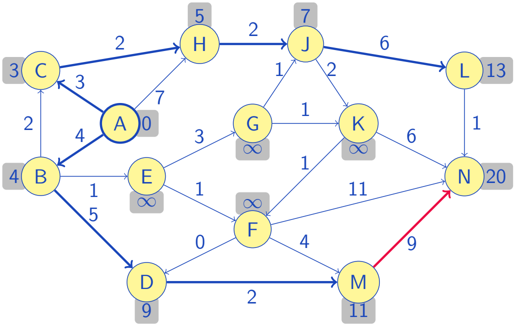
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 1



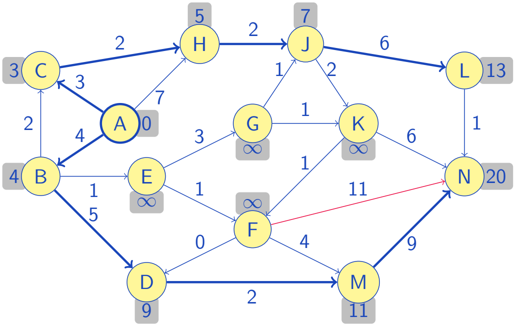
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



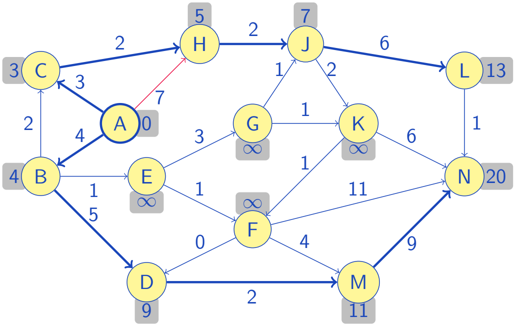
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



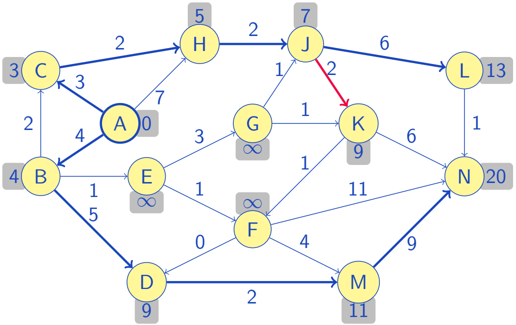
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



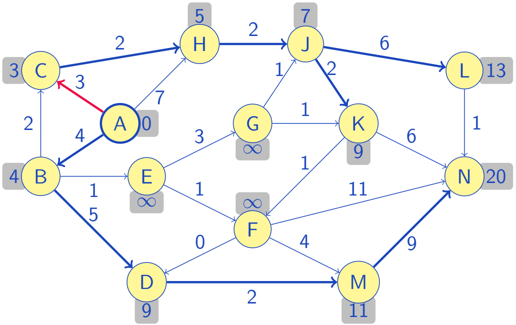
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



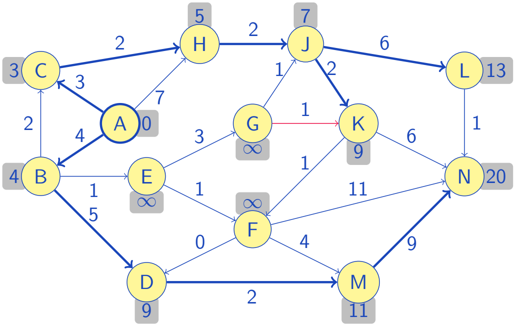
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



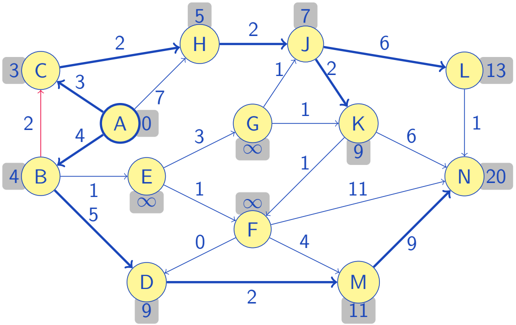
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



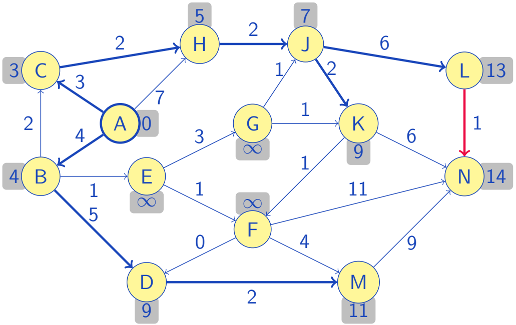
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



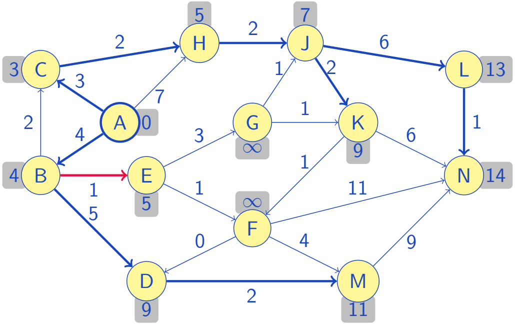
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



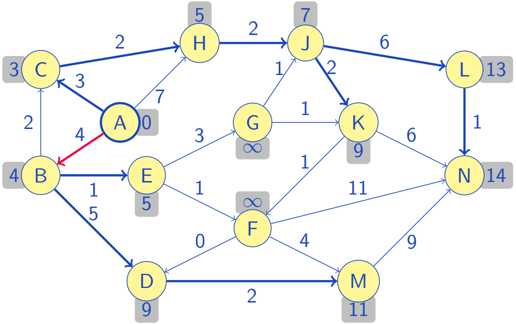
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



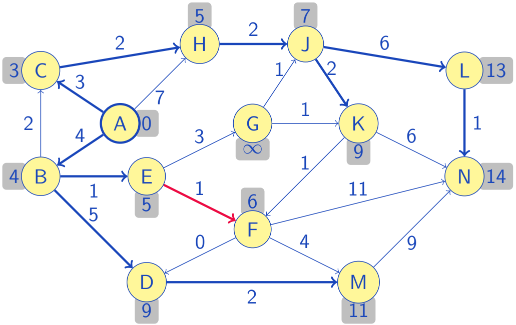
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



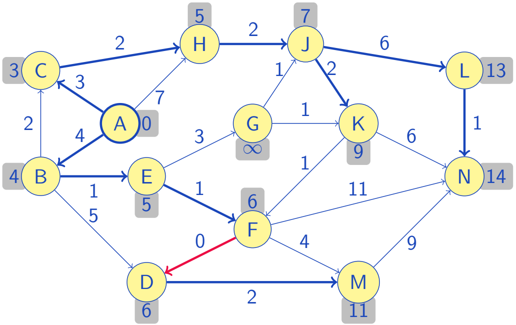
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



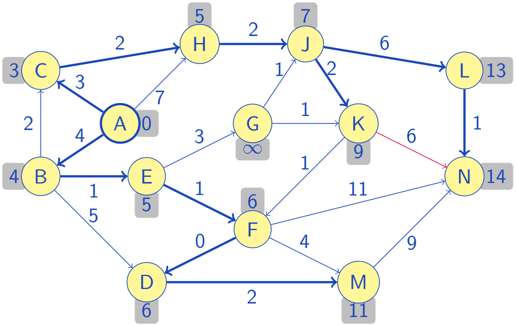
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



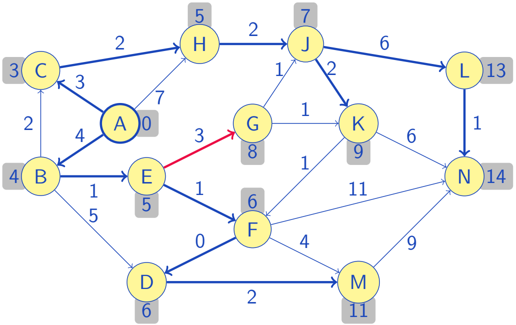
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



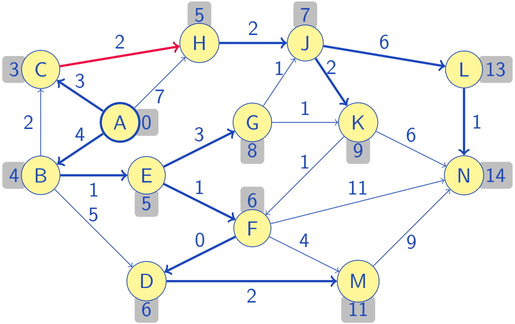
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



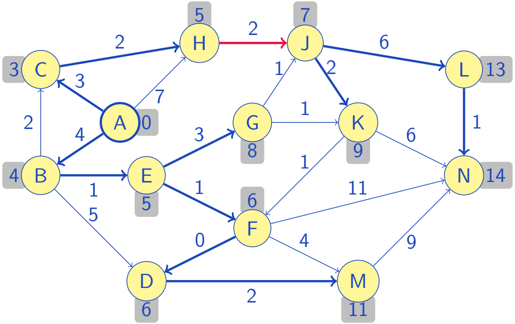
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



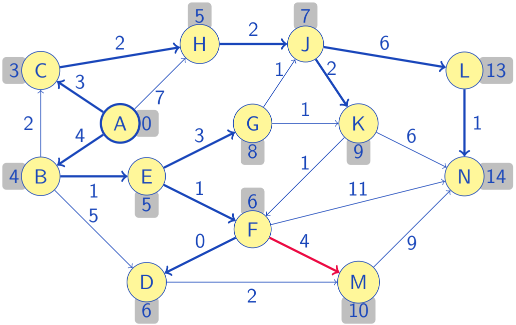
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



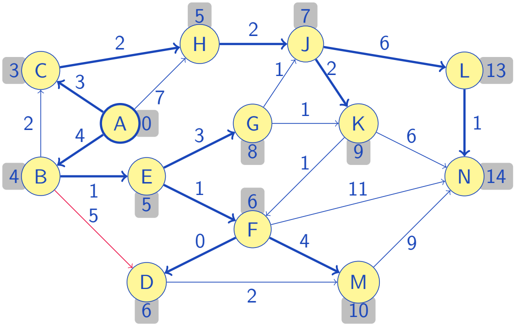
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



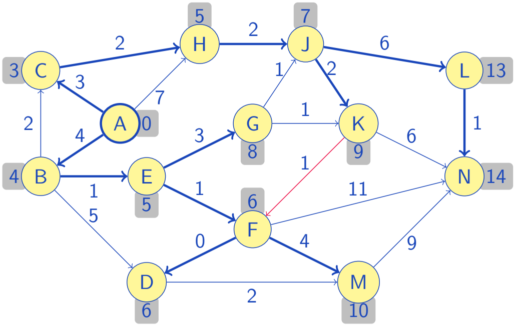
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



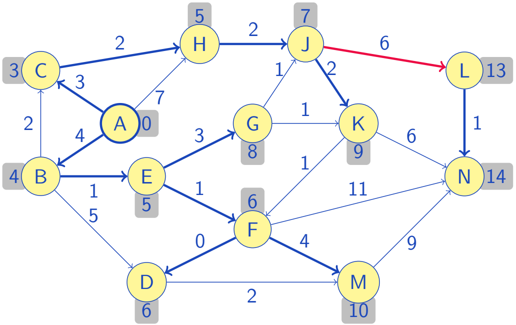
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



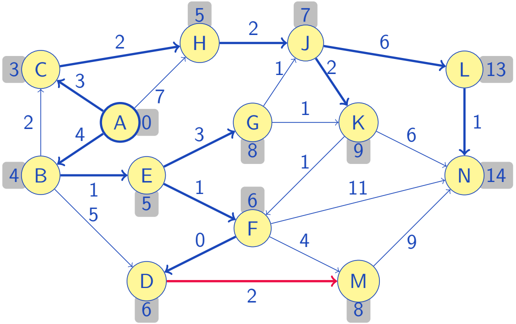
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



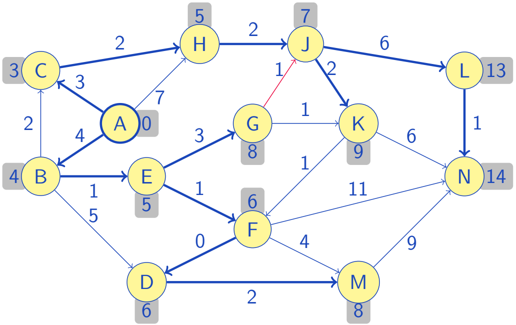
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



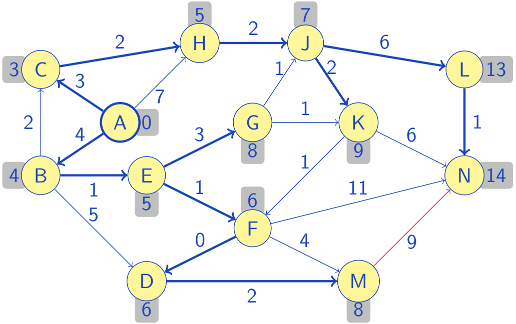
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 2



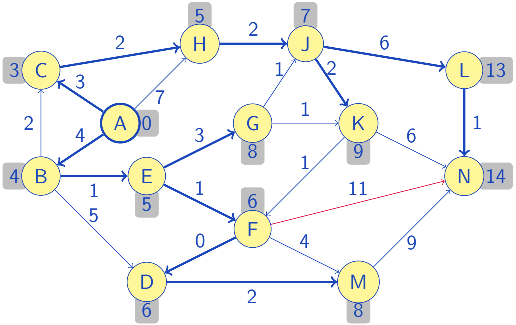
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



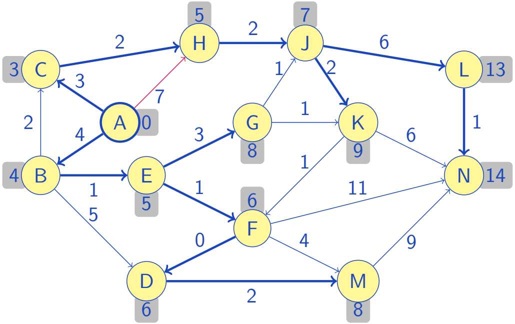
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



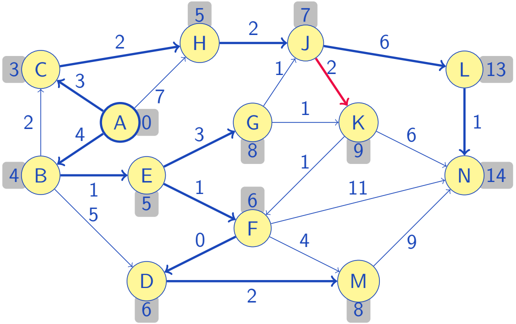
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



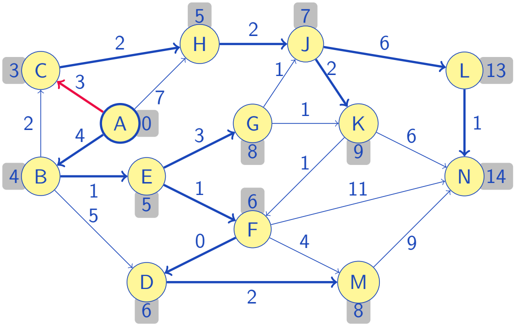
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



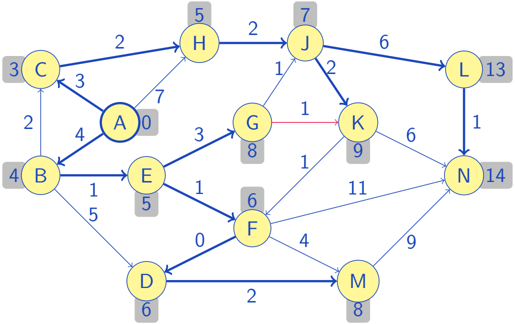
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



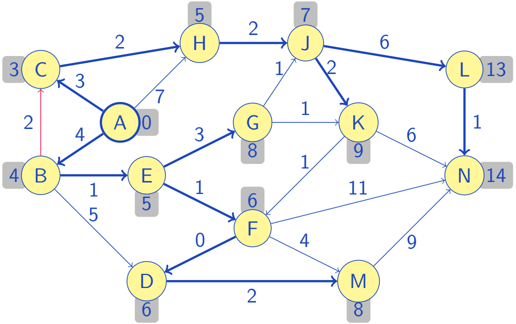
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



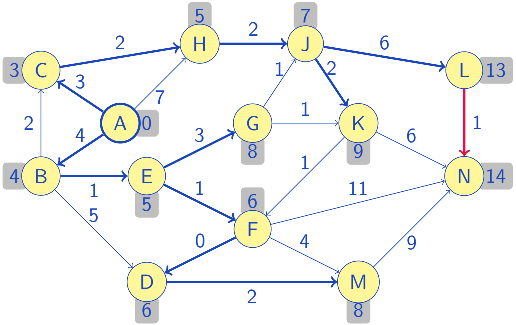
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



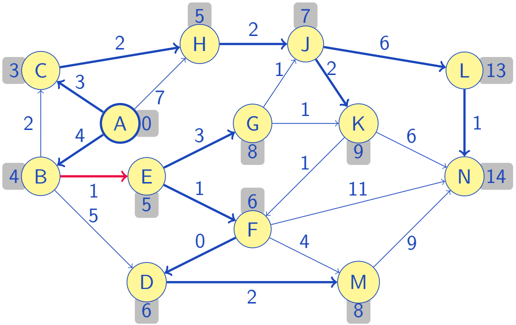
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



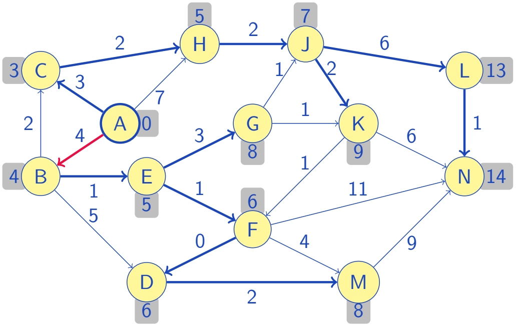
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



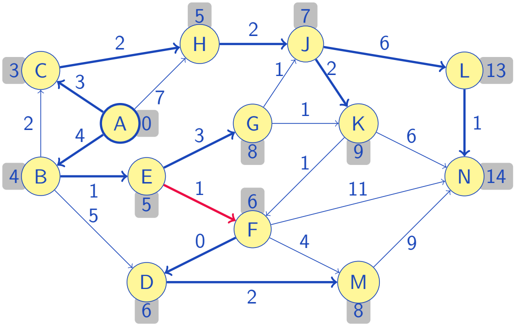
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



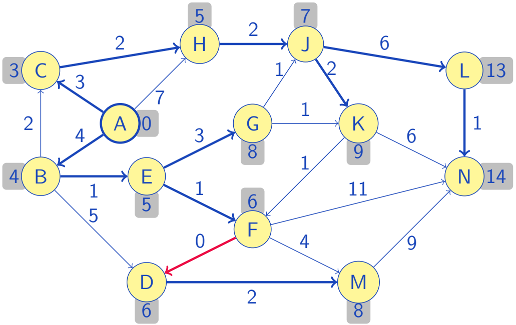
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



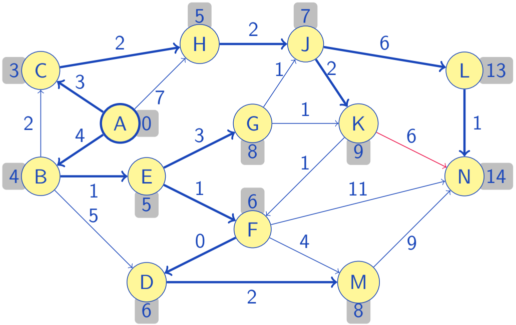
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



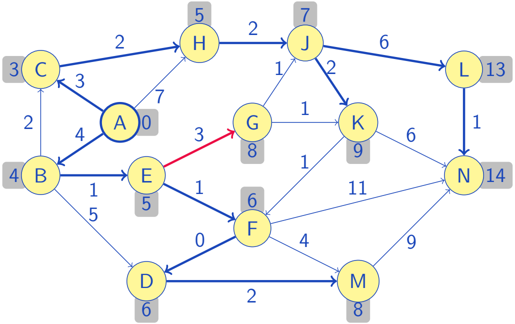
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



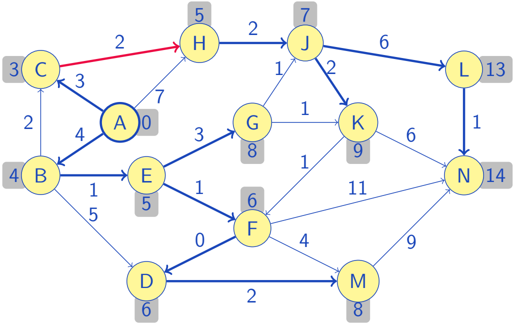
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



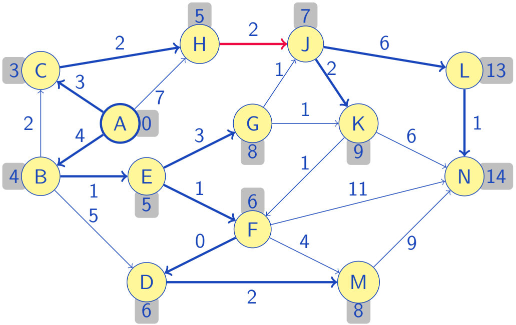
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



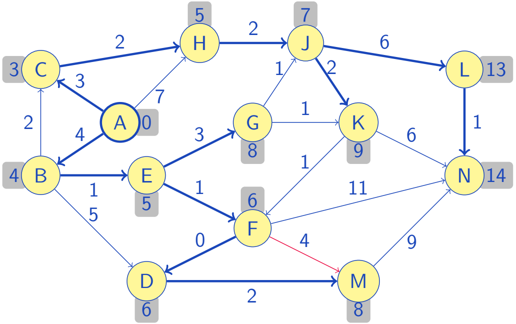
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



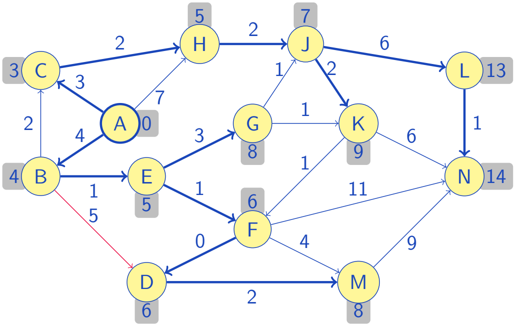
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



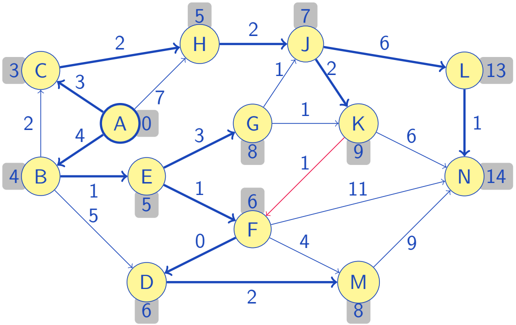
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



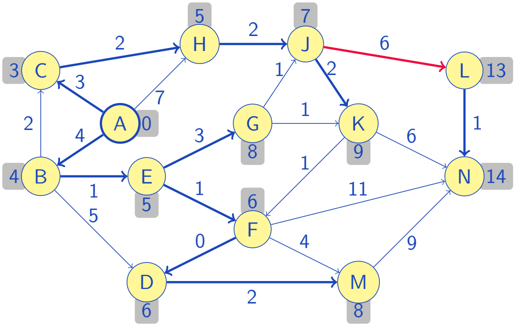
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



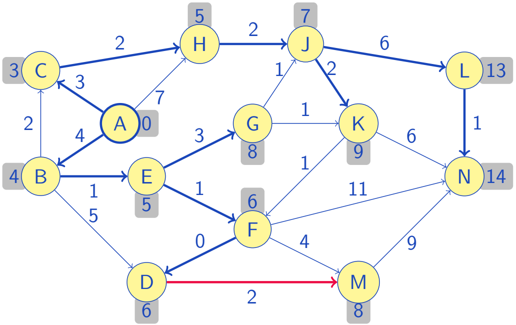
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



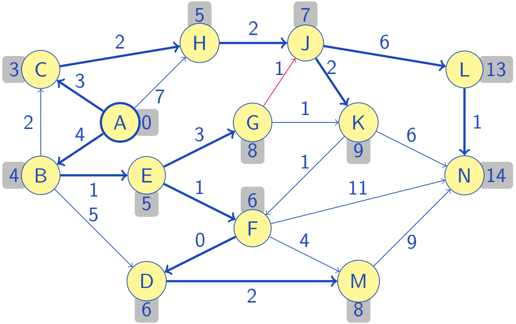
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



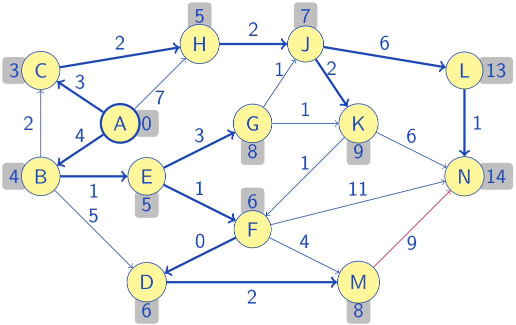
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3



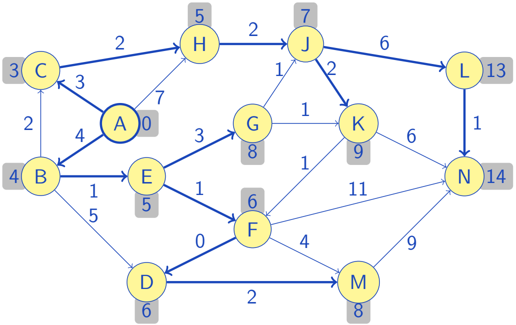
Bellman-Ford algorithm:

- Do $|V| - 1$ times:
  - Visit all edges
  - For (u, v), update shortest distance to v
  - If nothing updated, terminate
- Running time: $O(|V| \cdot |E|)$

Why does it work?

- First k iterations build first k segments in every shortest path

Example. Iteration 3. Nothing changed. We may stop



A negative cycle is a cycle of total negative length

- If any vertex of the negative cycle is reachable, then there is no shortest distance to any vertex of this cycle, and any vertex reachable from it

A negative cycle is a cycle of total negative length

- If any vertex of the negative cycle is reachable, then there is no shortest distance to any vertex of this cycle, and any vertex reachable from it

The Bellman-Ford algorithm can detect negative cycles. How?

- Update shortest distances along all edges once more
- If a shortest distance to v changes, then v is reachable from a negative cycle

An efficient algorithm for non-negative edge lengths

An efficient algorithm for non-negative edge lengths

- Idea: Maintain a set of vertices S with determined shortest distance from v_0
  - Initially, $S = \{v_0\}$
  - For all $v \notin S$, maintain shortest distance estimation: $D'[v] = \min_{u \in S} D[u] + L(u, v)$

An efficient algorithm for non-negative edge lengths

- Idea: Maintain a set of vertices S with determined shortest distance from v_0
  - Initially, $S = \{v_0\}$
  - For all $v \notin S$, maintain shortest distance estimation: $D'[v] = \min_{u \in S} D[u] + L(u, v)$
- Lemma: For a $v \notin S$ with the smallest $D'[v]$, the shortest distance is $D'[v]$

An efficient algorithm for non-negative edge lengths

- Idea: Maintain a set of vertices S with determined shortest distance from v_0
  - Initially, $S = \{v_0\}$
  - For all $v \notin S$, maintain shortest distance estimation: $D'[v] = \min_{u \in S} D[u] + L(u, v)$
- Lemma: For a $v \notin S$ with the smallest $D'[v]$, the shortest distance is $D'[v]$
  - Assume it is not true
  - There is another path which yields a distance strictly smaller than $D'[v]$

An efficient algorithm for non-negative edge lengths

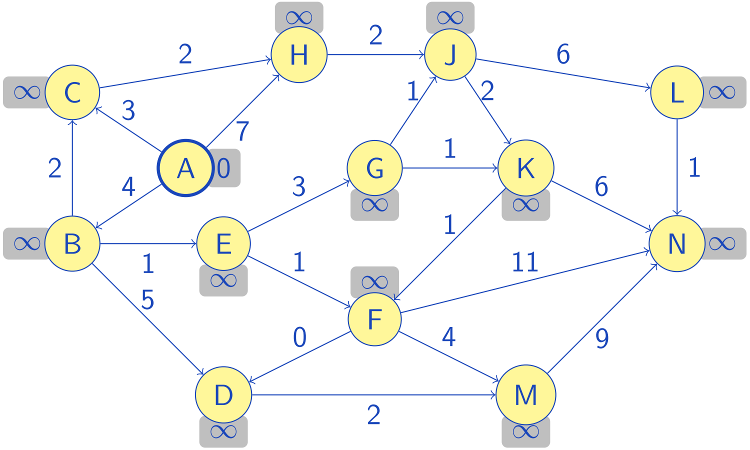
- Idea: Maintain a set of vertices S with determined shortest distance from v_0
  - Initially, $S = \{v_0\}$
  - For all $v \notin S$, maintain shortest distance estimation: $D'[v] = \min_{u \in S} D[u] + L(u, v)$
- Lemma: For a $v \notin S$ with the smallest $D'[v]$, the shortest distance is $D'[v]$
  - Assume it is not true
  - There is another path which yields a distance strictly smaller than $D'[v]$
  - It should go through other $v' \notin S$

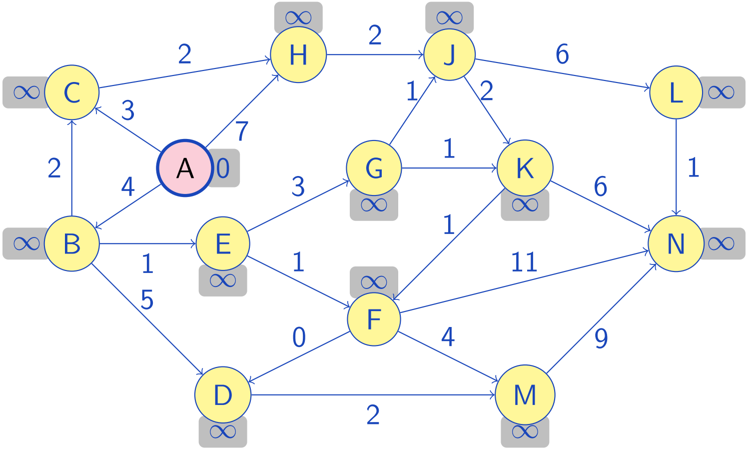
An efficient algorithm for non-negative edge lengths

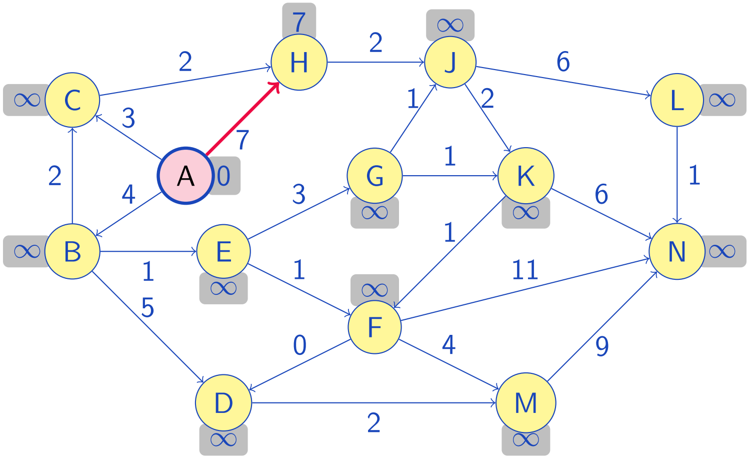
- Idea: Maintain a set of vertices S with determined shortest distance from v_0
  - Initially, $S = \{v_0\}$
  - For all $v \notin S$, maintain shortest distance estimation: $D'[v] = \min_{u \in S} D[u] + L(u, v)$
- Lemma: For a $v \notin S$ with the smallest $D'[v]$, the shortest distance is $D'[v]$
  - Assume it is not true
  - There is another path which yields a distance strictly smaller than $D'[v]$
  - It should go through other $v' \notin S$
  - But edge lengths are non-negative $\rightarrow$ contradiction

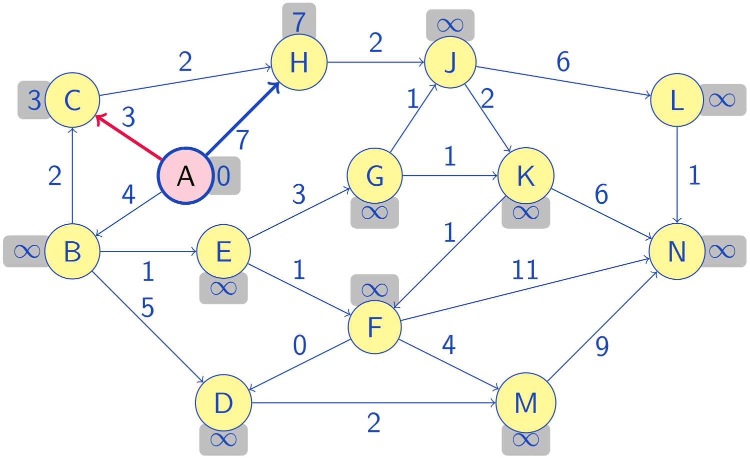
An efficient algorithm for non-negative edge lengths

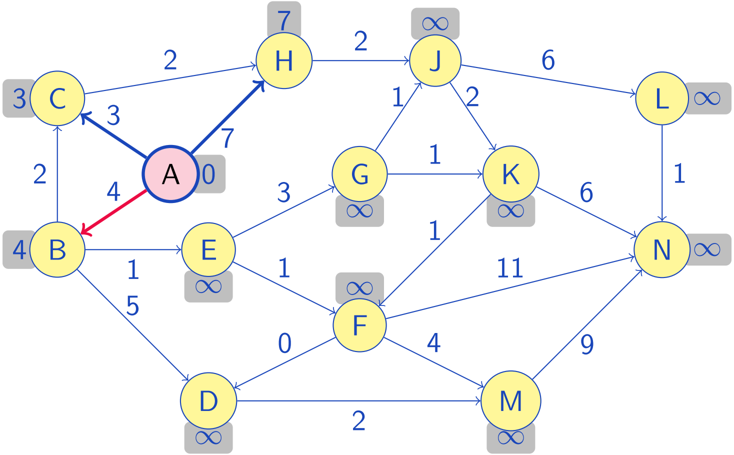
- Idea: Maintain a set of vertices S with determined shortest distance from v_0
  - Initially, $S = \{v_0\}$
  - For all $v \notin S$, maintain shortest distance estimation: $D'[v] = \min_{u \in S} D[u] + L(u, v)$
- Lemma: For a $v \notin S$ with the smallest $D'[v]$, the shortest distance is $D'[v]$
  - Assume it is not true
  - There is another path which yields a distance strictly smaller than $D'[v]$
  - It should go through other $v' \notin S$
  - But edge lengths are non-negative $\rightarrow$ contradiction
- How to update S:
  - Choose $v \notin S$ with the smallest $D'[v]$
  - Set $D[v] = D'[v]$
  - $S \leftarrow S \cup \{v\}$
  - For all edges $(v, v') \in E$, update $D'[v'] \leftarrow \min(D'[v'], D[v] + L(v, v'))$

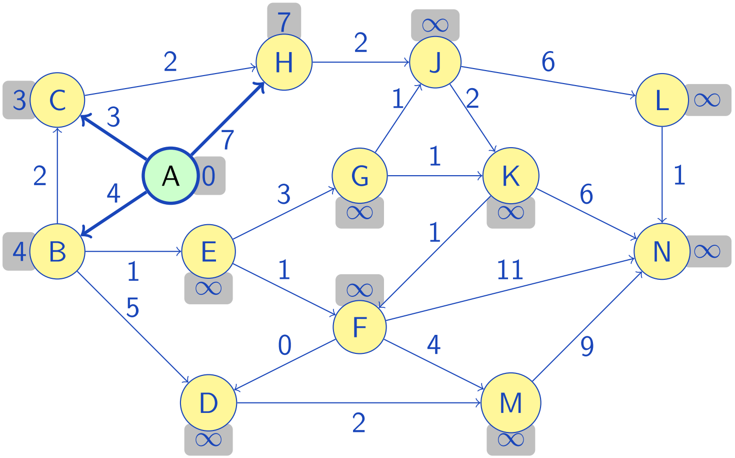

H
J
C
L
A
0
G
K
B
E
N
F
D
M
2
2
6
3
7
1
2
1
2
4
3
6
1
1
1
11
1
5
0
4
9
2

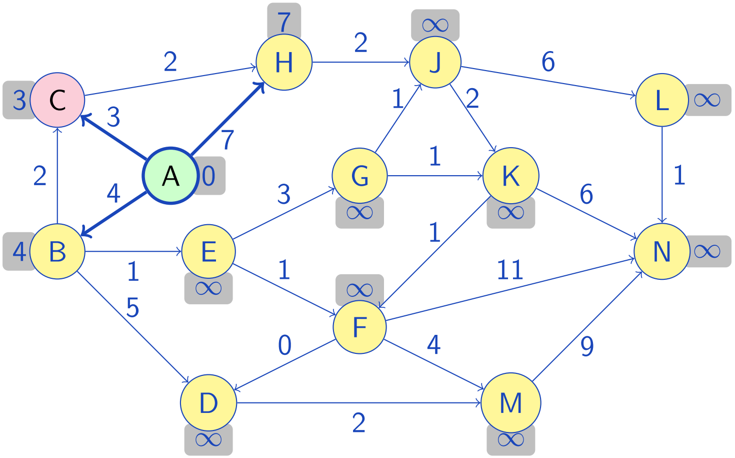

∞
H
2
∞
J
6
∞
C
2
L
∞
3
1
2
7
A
0
G
1
K
1
2
3
6
4
∞
∞
1
∞
B
E
N
∞
1
1
11
∞
∞
5
F
0
4
9
D
M
2
∞
∞

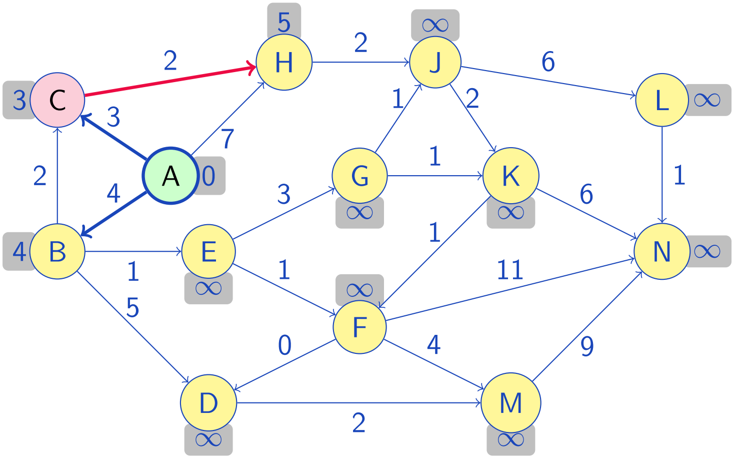

7
H
2
J
∞
6
L
∞
C
∞
2
3
7
1
2
A
0
G
1
K
1
2
4
3
∞
∞
6
B
∞
E
1
N
∞
1
1
11
∞
5
F
∞
0
4
9
D
M
2
∞
∞

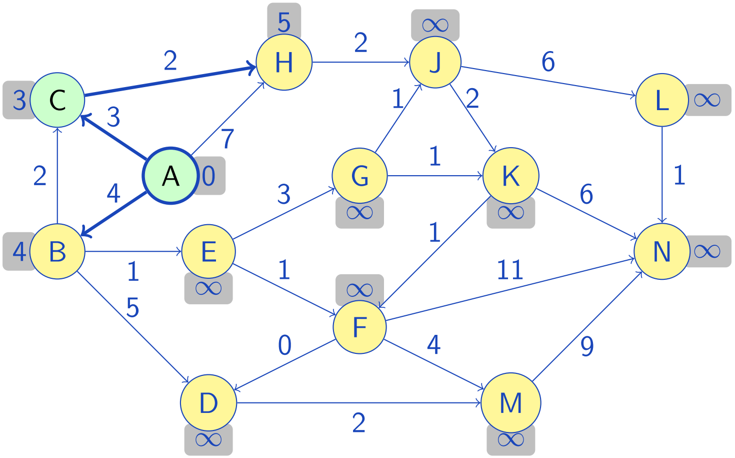

7
H
2
∞
J
6
3
C
2
3
1
2
L
∞
7
A
0
G
1
K
1
2
4
3
∞
∞
6
∞
B
E
1
1
N
∞
1
1
11
∞
∞
5
F
0
4
9
D
2
M
∞
∞

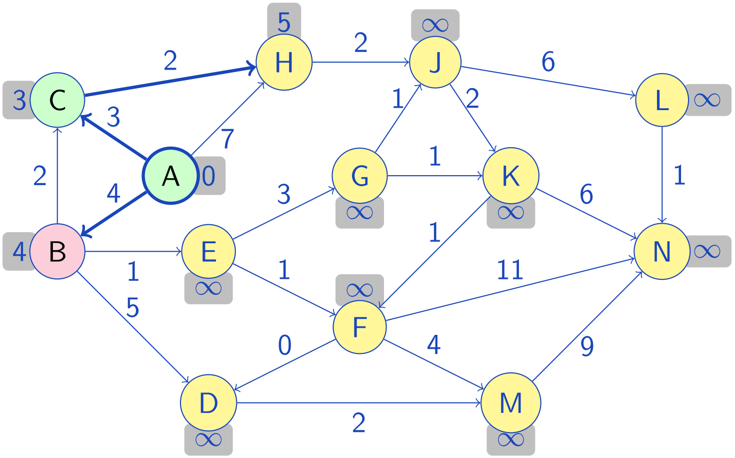

A
B
C
D
E
F
G
H
J
K
L
M
N
0
3
4
7
∞
2
3
7
4
2
1
5
3
1
0
2
1
1
1
2
2
6
6
1
11
4
9

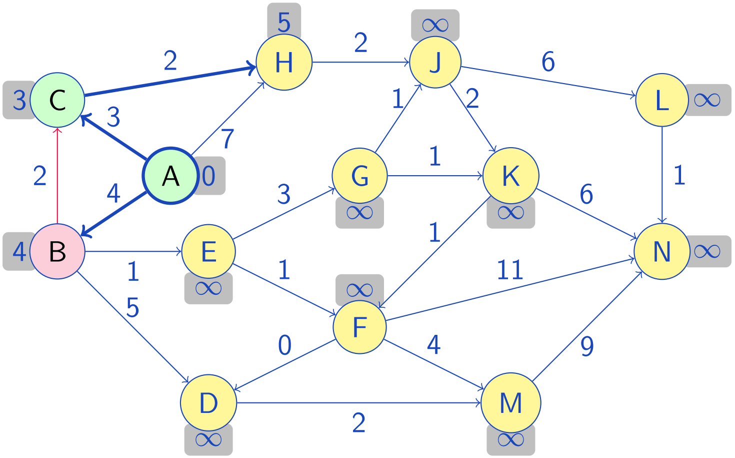

A
B
C
D
E
F
G
H
J
K
L
M
N
0
3
4
7
∞
2
3
7
4
2
1
5
2
6
1
2
1
3
1
1
0
4
11
6
1
9
2

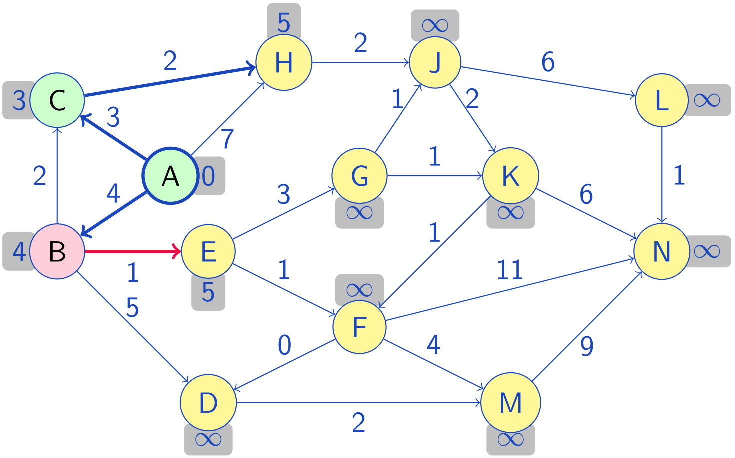

A
0
B
4
C
3
H
7
J
∞
L
∞
G
∞
K
∞
E
∞
F
∞
N
∞
D
∞
M
∞
2
2
6
3
7
1
2
2
1
1
4
3
6
1
1
1
11
5
0
4
9
2

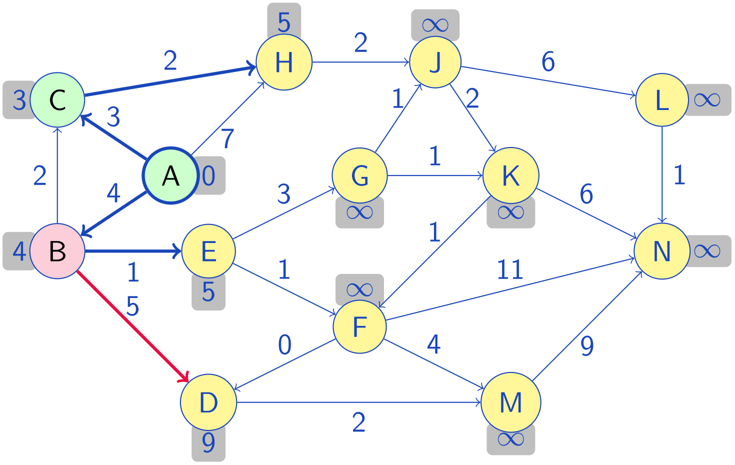

5
2
∞
H
J
6
2
3
C
3
1
2
L
∞
7
A
0
G
1
K
1
2
4
3
∞
∞
6
4
B
E
1
1
N
∞
1
11
∞
∞
5
F
0
4
9
D
M
2
∞
∞

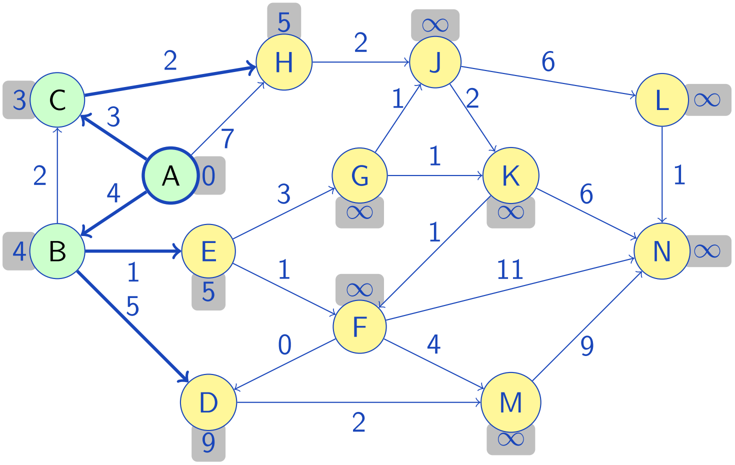

5
2
∞
6
H
J
3
C
2
1
2
L
∞
3
7
A
0
2
4
G
1
K
1
3
6
∞
∞
1
4
B
E
N
∞
1
1
11
∞
∞
5
F
0
4
9
D
M
2
∞
∞

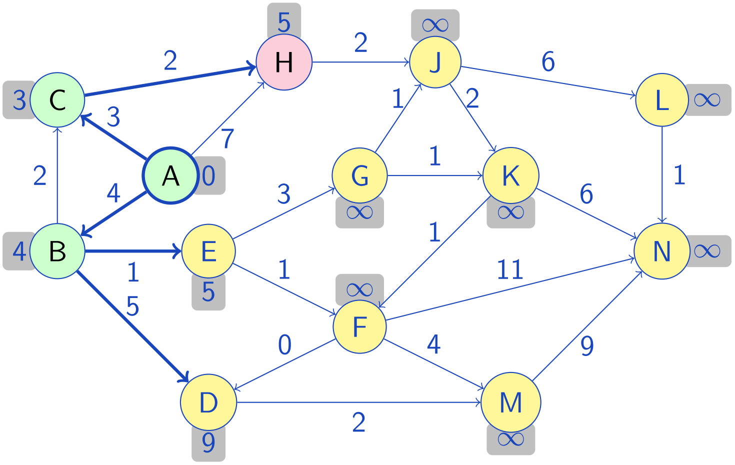

5
H
2
∞
J
6
L
∞
3
C
2
3
1
2
7
A
0
G
1
K
1
2
4
3
∞
∞
6
N
∞
4
B
E
1
∞
1
11
1
∞
5
F
0
4
9
D
2
M
∞
∞

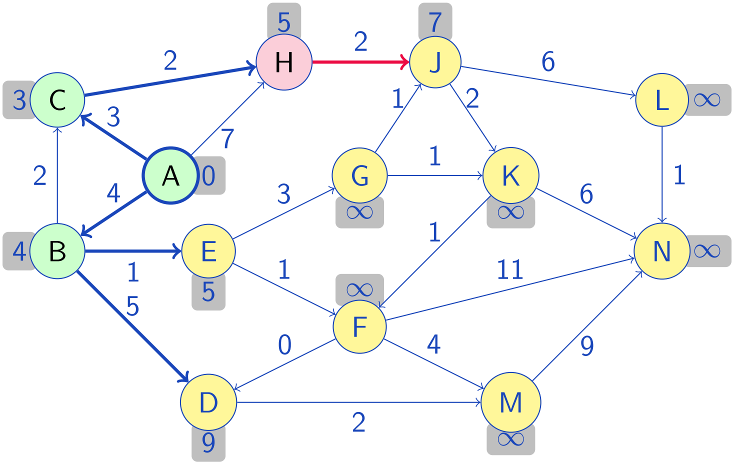

A
0
B
4
C
3
D
∞
E
∞
F
∞
G
∞
H
5
J
∞
K
∞
L
∞
M
∞
N
∞
2
3
7
4
2
1
5
2
6
1
2
1
3
1
1
6
1
11
0
4
2
9

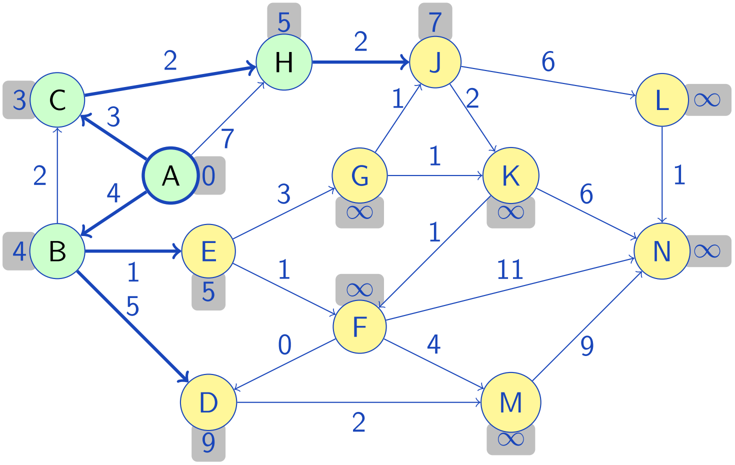

5
H
2
∞
J
6
L
∞
3
C
2
3
7
1
2
A
0
G
1
K
1
2
4
3
∞
∞
6
4
B
E
1
1
N
∞
1
5
∞
11
5
F
0
4
9
D
2
M
∞
∞

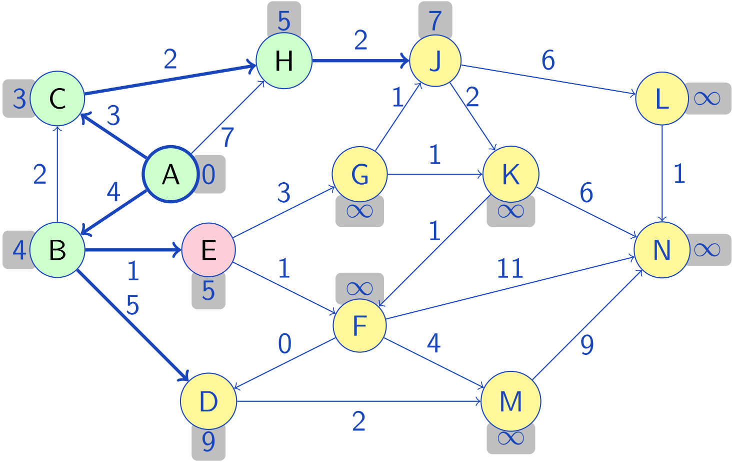

A
0
B
4
C
3
D
9
E
5
F
∞
G
∞
H
5
J
∞
K
∞
L
∞
M
∞
N
∞
2
3
7
2
4
1
5
3
1
0
2
2
1
1
2
6
1
1
4
11
6
1
9

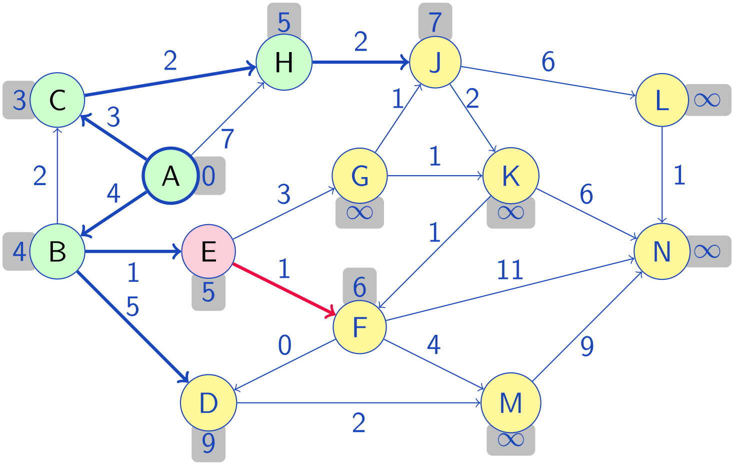

A
0
B
4
C
3
D
9
E
5
F
∞
G
∞
H
5
J
∞
K
∞
L
∞
M
∞
N
∞
2
3
7
2
4
1
5
2
3
1
0
1
2
1
2
6
1
1
6
11
4
9
2

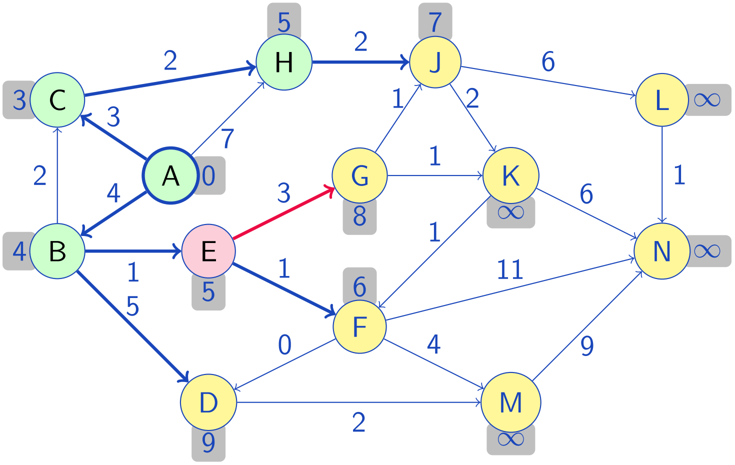

5
H
2
∞
J
6
L
∞
3
C
2
3
1
2
7
A
0
G
1
K
1
2
4
3
∞
∞
6
4
B
E
1
1
N
∞
1
11
5
5
∞
F
0
4
9
D
M
2
9
∞

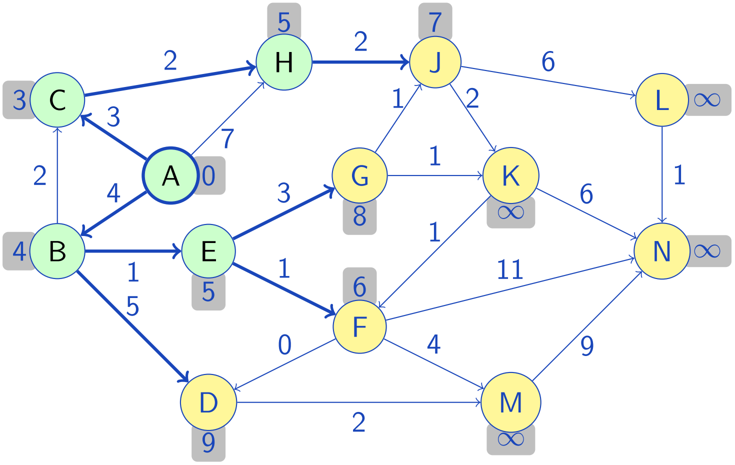

A
B
C
D
E
F
G
H
J
K
L
M
N
0
3
4
5
5
9
7
∞
2
3
7
4
2
1
5
2
6
1
2
3
1
1
1
0
2
4
11
6
1
9

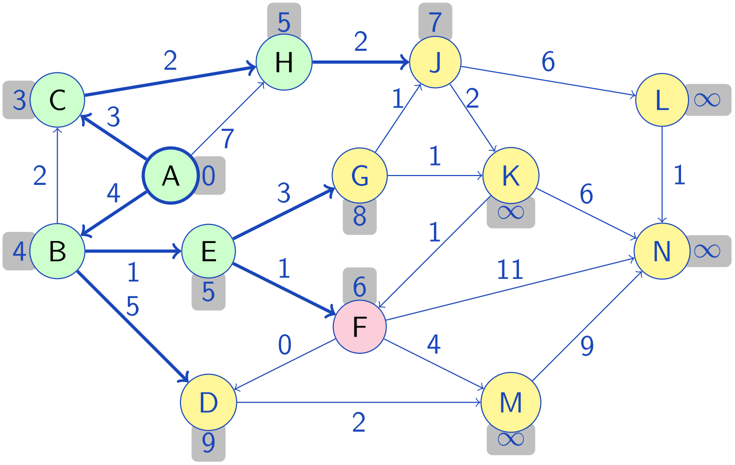

A
B
C
D
E
F
G
H
J
K
L
M
N
0
3
4
5
5
7
9
∞
2
3
4
7
1
5
2
2
6
1
2
1
3
1
1
0
2
4
11
6
1
9


A
B
C
D
E
F
G
H
J
K
L
M
N
0
3
4
5
5
7
9
∞
2
3
7
4
2
1
5
2
6
1
2
1
1
3
1
0
1
6
11
4
9
2

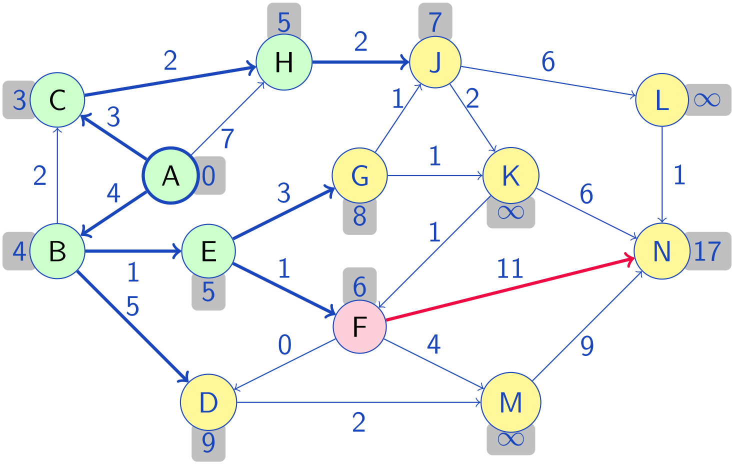

5
7
H
J
2
2
6
3
C
3
7
L
∞
1
2
A
0
G
1
K
2
1
4
3
6
∞
∞
1
4
B
E
N
∞
1
11
1
5
6
5
F
0
4
9
D
M
2
9
∞

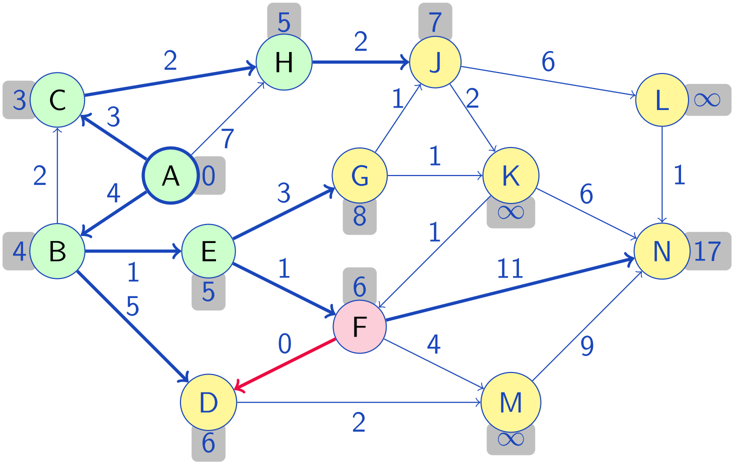

A
0
B
4
C
3
D
9
E
5
F
6
G
8
H
5
J
7
K
∞
L
∞
M
∞
N
∞
2
2
3
7
2
4
1
5
3
1
1
2
1
6
0
1
11
4
2
9
6
1

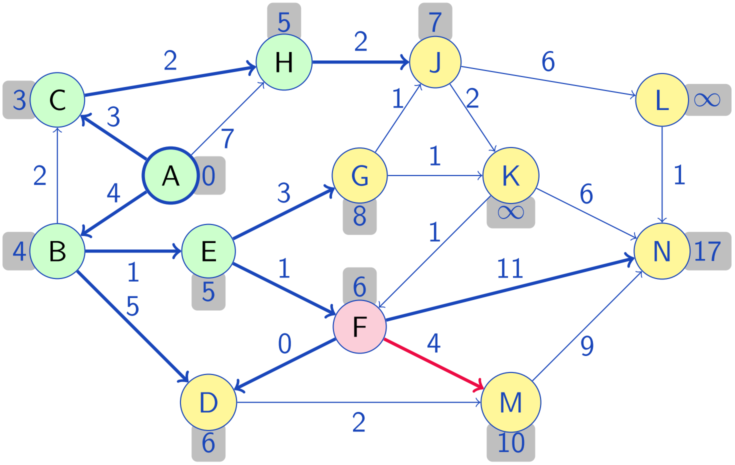

A
0
B
4
C
3
D
9
E
5
F
6
G
8
H
5
J
7
K
∞
L
∞
M
∞
N
∞
2
2
6
3
7
2
1
2
1
4
3
1
1
6
1
1
5
11
0
4
9
2

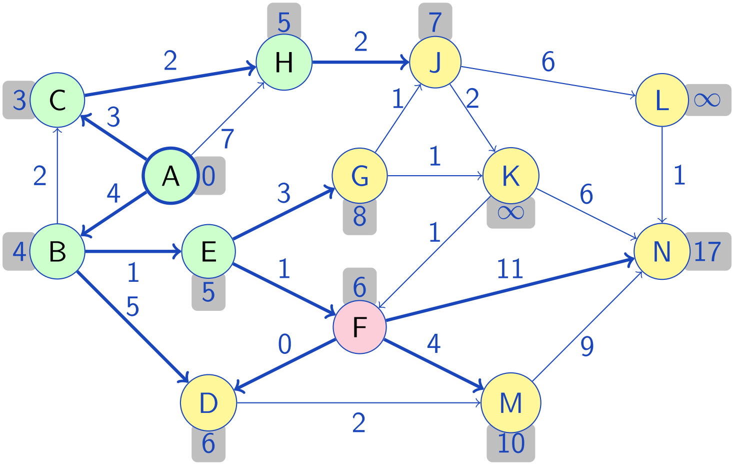


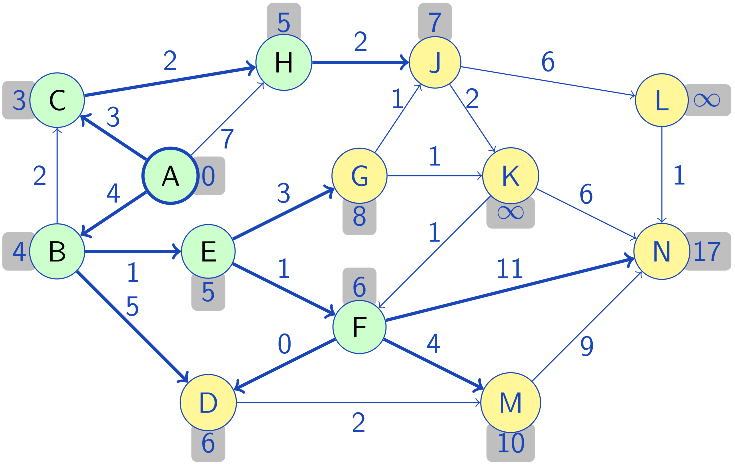
A
B
C
D
E
F
G
H
J
K
L
M
N
0
3
4
5
9
6
8
7
∞
2
3
7
4
2
1
5
3
1
0
2
1
1
2
6
1
1
6
11
4
9
1

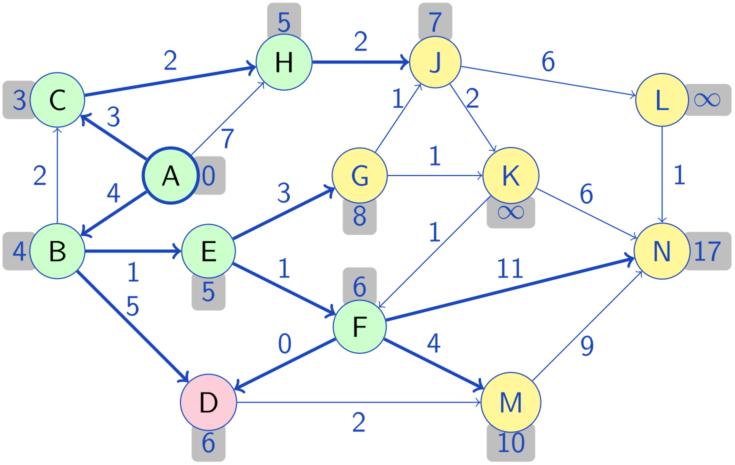

A
B
C
D
E
F
G
H
J
K
L
M
N
0
3
4
5
5
6
7
8
9
17
∞
∞
∞
2
3
7
4
2
1
5
3
1
0
2
2
1
2
1
1
6
6
1
11
4
9

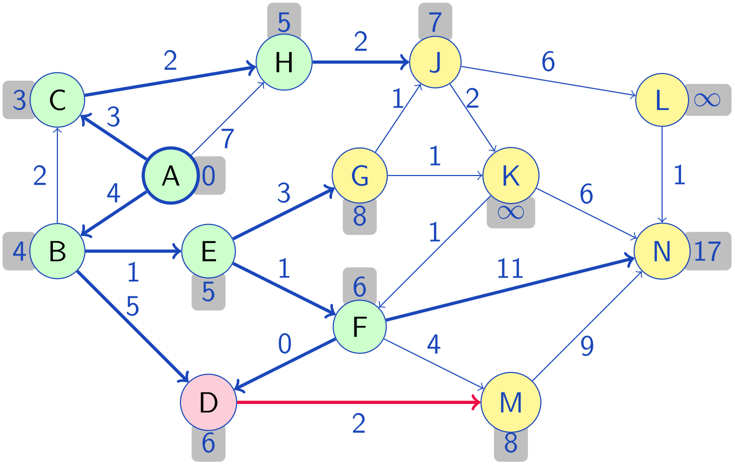

5
7
2
H
J
6
3
C
3
L
∞
1
2
7
2
A
0
G
1
K
1
2
4
3
8
∞
6
4
B
E
1
N
17
1
1
11
5
5
6
F
0
4
9
D
M
6
2
∞

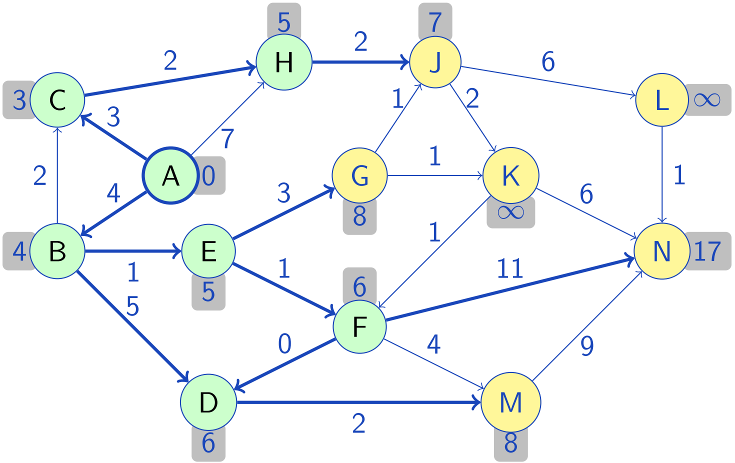

5
2
7
H
J
6
3
C
2
L
∞
3
7
1
2
A
0
1
G
K
1
2
4
3
8
∞
6
4
B
E
1
N
17
1
11
5
1
6
5
F
0
4
9
D
M
2
6
10

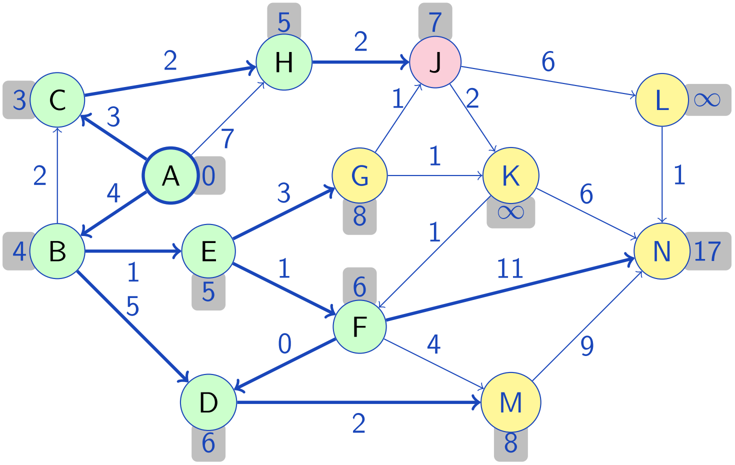

A
B
C
D
E
F
G
H
J
K
L
M
N
0
3
4
5
5
6
6
7
8
10
17
∞
∞

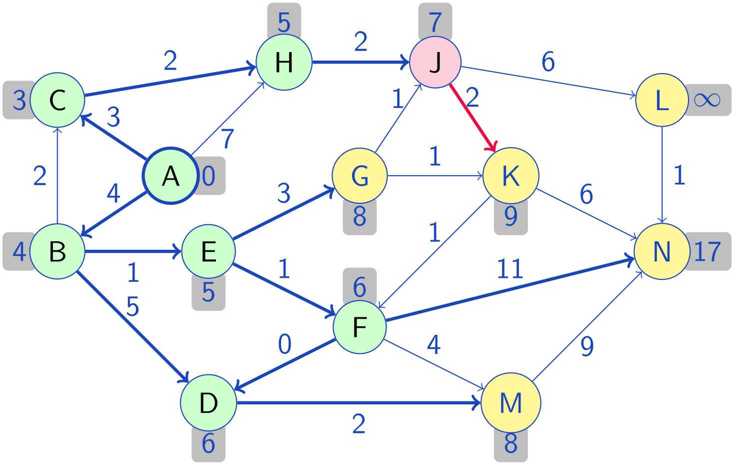

A
B
C
D
E
F
G
H
J
K
L
M
N
0
4
3
6
5
6
8
5
7
∞
∞
10
17
2
2
6
3
7
1
2
2
1
4
3
1
1
1
1
6
5
11
0
4
9
2

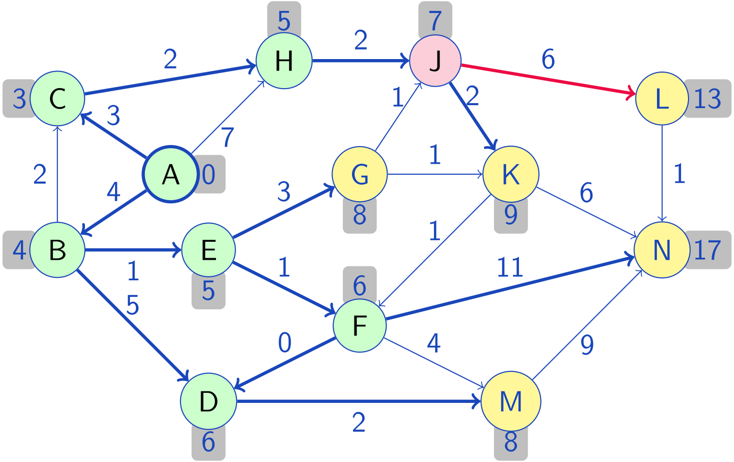

5
H
2
7
J
6
L
∞
3
C
2
3
7
1
2
A
0
1
G
K
1
2
4
3
6
8
∞
4
B
E
1
N
17
1
1
11
5
5
6
F
0
4
9
D
M
2
6
10

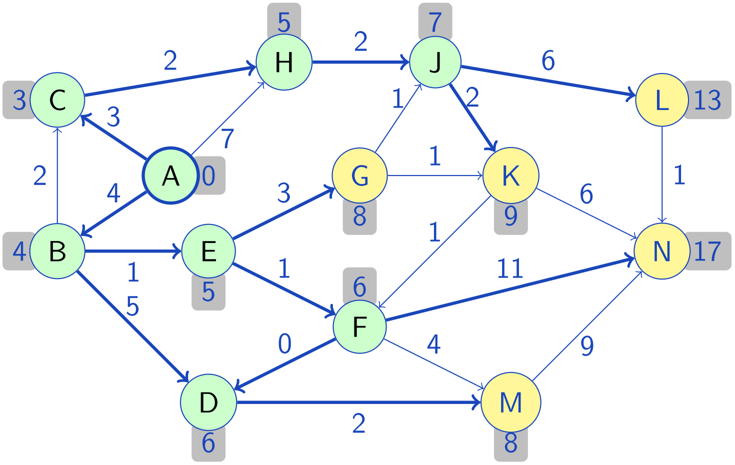

5
7
2
6
H
J
2
3
C
L
∞
3
1
2
7
A
0
G
1
K
2
4
3
8
∞
6
1
1
B
4
E
N
17
1
1
11
5
5
6
F
0
4
9
D
M
2
6
8

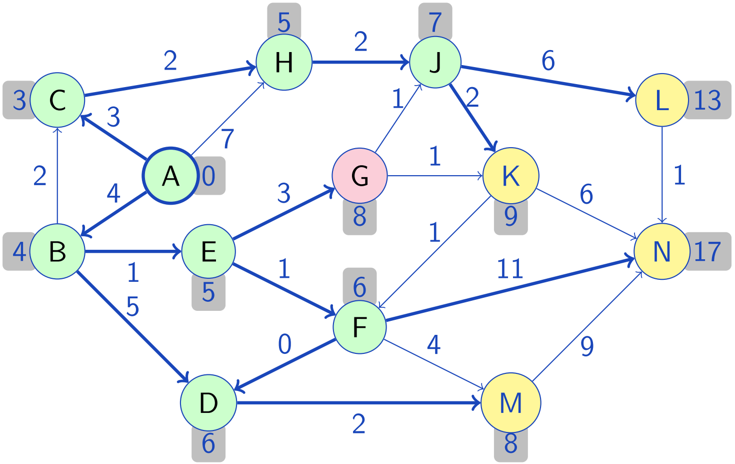

A
0
B
4
C
3
D
6
E
5
F
6
G
8
H
5
J
7
K
∞
L
∞
M
8
N
17
2
2
6
3
7
1
2
2
1
1
4
3
6
1
1
1
11
5
0
4
9
2

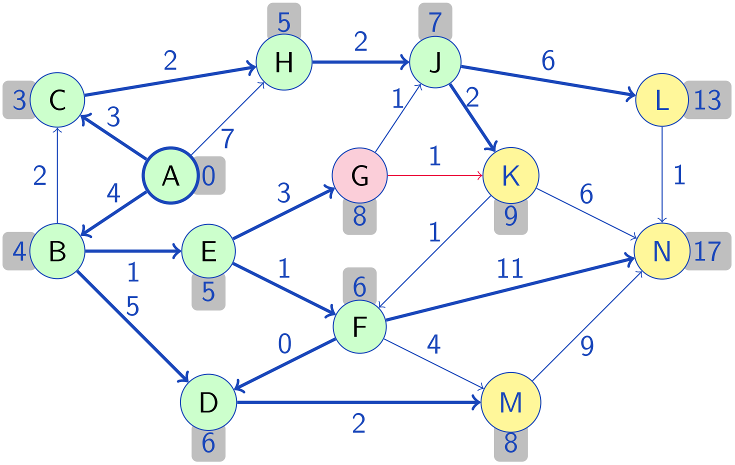

5
H
2
7
J
6
L
∞
3
C
3
2
7
1
2
A
0
G
1
K
2
4
3
8
∞
6
1
4
B
E
1
N
17
1
1
11
5
5
6
F
0
4
9
D
M
2
6
8

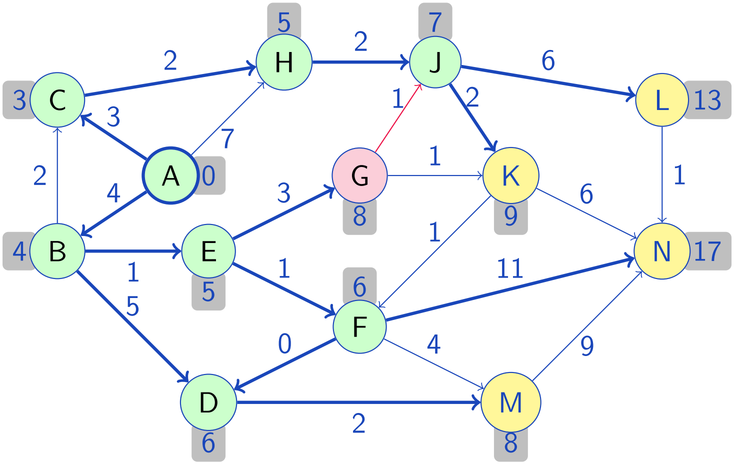

5
H
2
7
J
6
L
∞
3
C
2
3
7
1
2
G
1
K
1
2
A
0
4
3
8
9
6
4
B
1
E
5
1
1
6
F
11
N
17
5
0
4
9
D
6
2
M
8

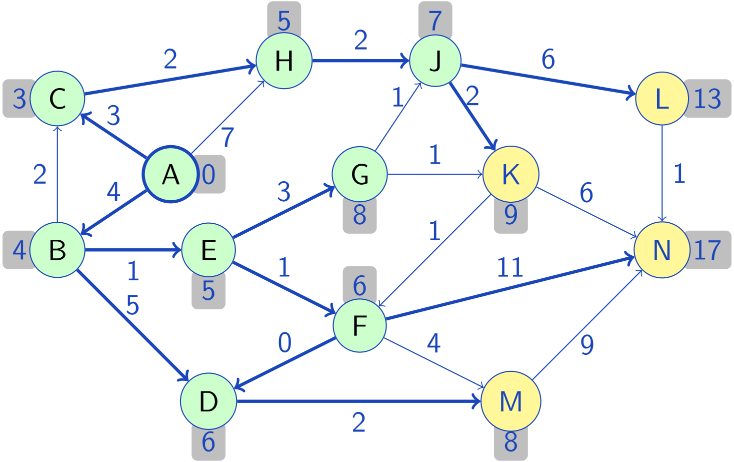

A
B
C
D
E
F
G
H
J
K
L
M
N
0
3
4
5
6
5
6
8
5
7
9
13
8
17

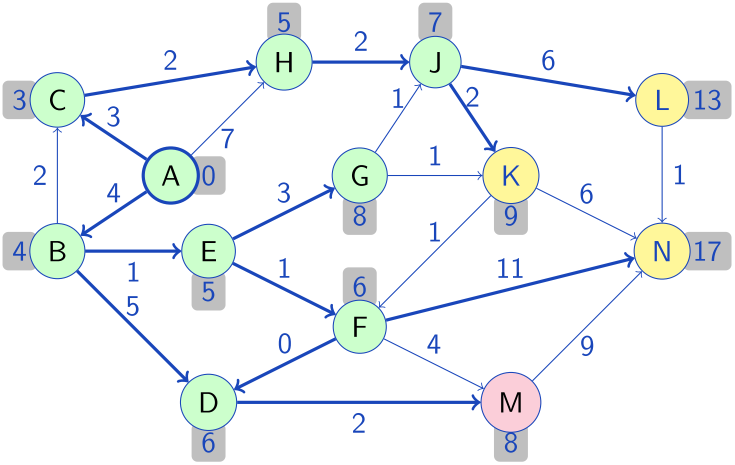

A
0
B
4
C
3
D
6
E
5
F
6
G
8
H
5
J
7
K
9
L
13
M
8
N
17
2
3
7
2
4
1
5
3
1
0
2
2
1
1
1
2
6
6
1
11
4
9

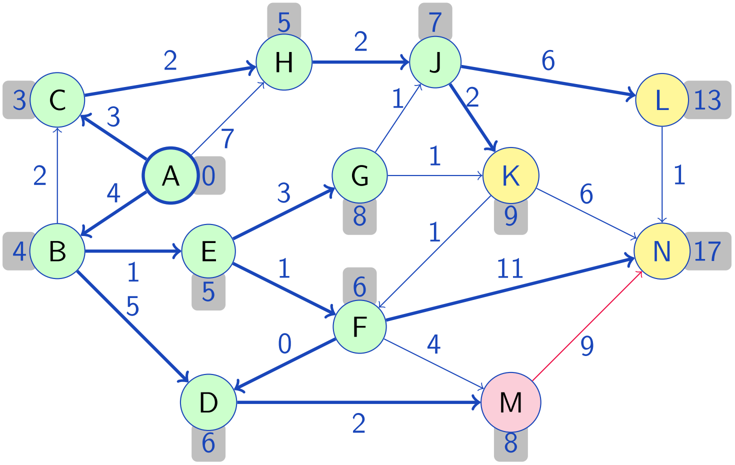

5
7
2
2
6
3
C
H
J
L
13
3
7
1
2
A
0
2
4
G
1
K
1
6
3
8
9
4
B
E
1
N
17
1
1
11
5
5
6
F
0
4
9
D
M
2
6
8

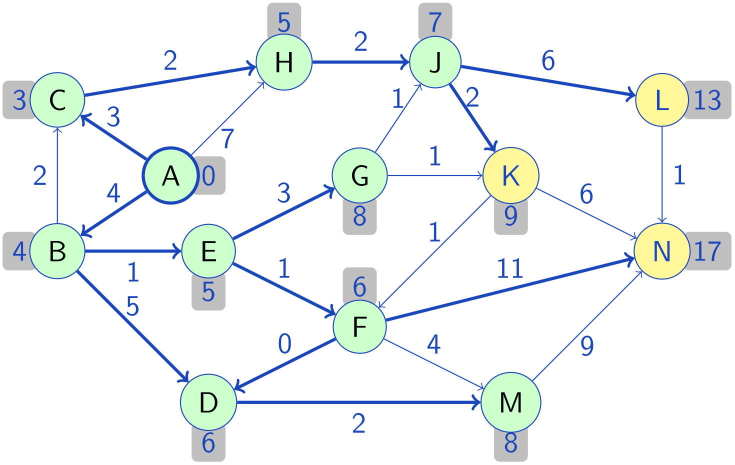

A
0
B
4
C
3
D
6
E
5
F
6
G
8
H
5
J
7
K
9
L
13
M
8
N
17
2
3
7
2
4
1
5
3
1
0
2
2
1
2
1
6
1
4
11
6
1
9

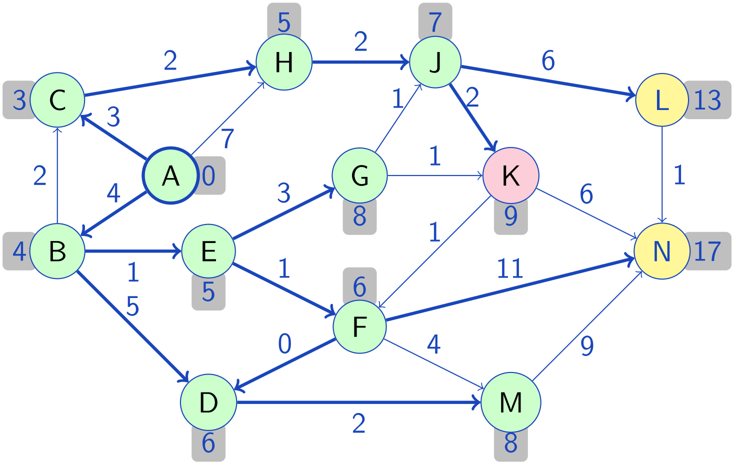

A
0
B
4
C
3
D
6
E
5
F
6
G
8
H
5
J
7
K
9
L
13
M
8
N
17
2
3
7
2
4
1
5
3
1
0
2
2
1
1
2
6
1
1
11
4
6
1
9

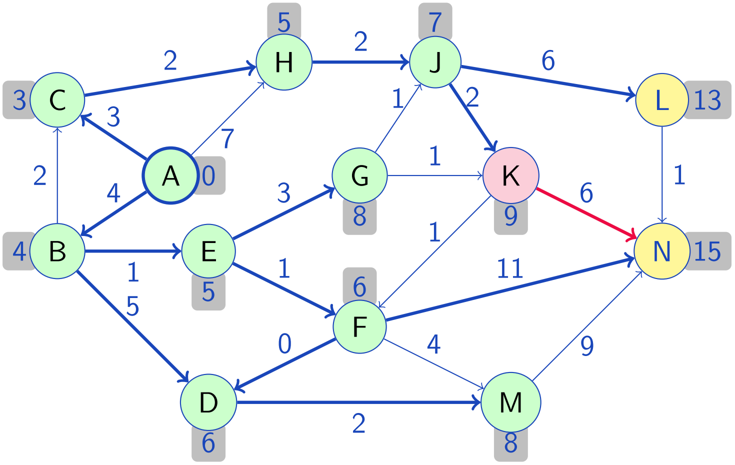

A 0
C 3
B 4
H 5
E 5
D 6
F 6
J 7
G 8
M 8
K 9
L 13
N 17

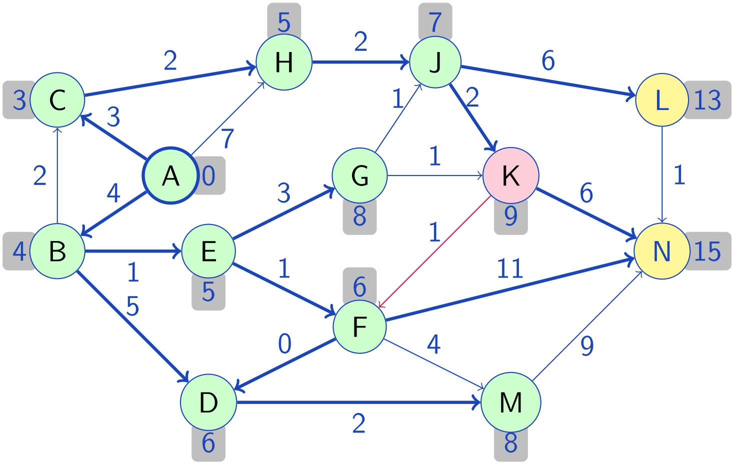

A
0
B
4
C
3
D
6
E
5
F
6
G
8
H
5
J
7
K
9
L
13
M
8
N
17
2
3
7
2
4
1
5
2
3
1
0
2
1
1
2
6
1
1
11
4
6
1
9

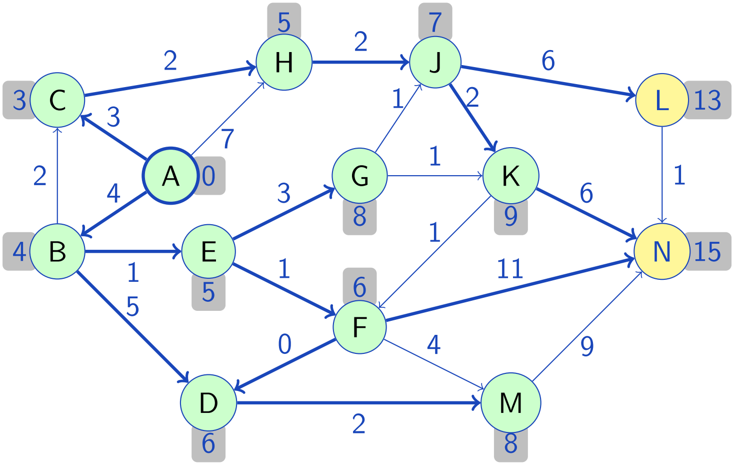

A
B
C
D
E
F
G
H
J
K
L
M
N
0
4
3
6
5
6
8
5
7
9
13
8
17
2
3
7
2
4
1
5
3
1
0
2
2
1
1
2
6
1
6
1
11
4
9

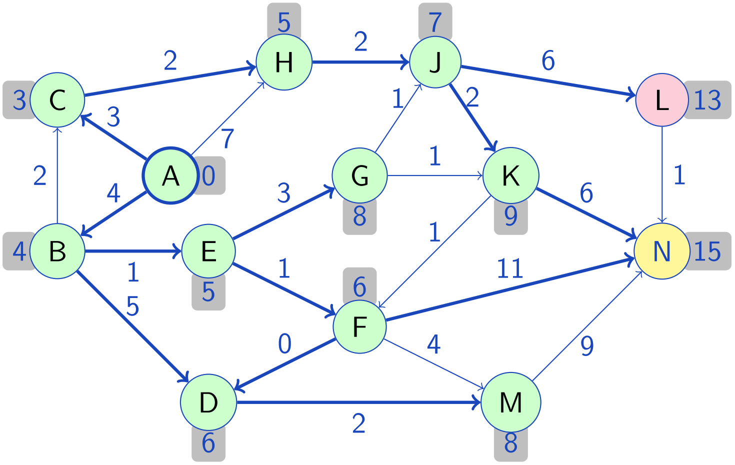

A
0
B
4
C
3
D
6
E
5
F
6
G
8
H
5
J
7
K
9
L
13
M
8
N
17
2
3
7
2
4
1
5
3
1
0
2
2
1
1
1
6
2
1
6
11
4
9
1

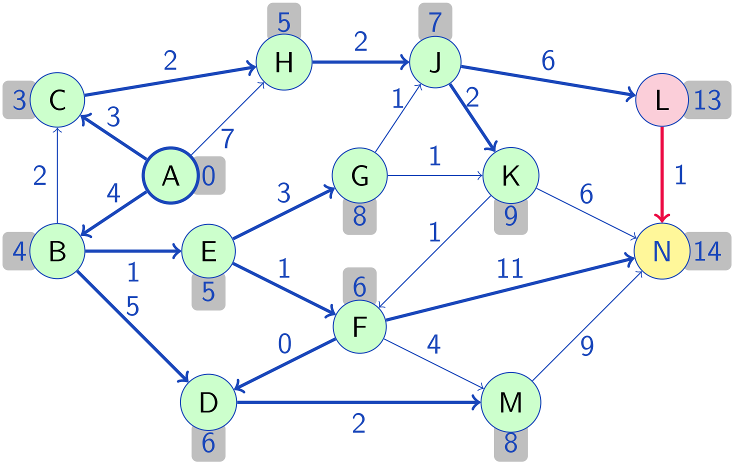

5
7
2
6
2
H
J
3
C
3
7
1
2
L
13
2
A
0
4
3
G
1
K
6
1
8
9
1
4
B
E
1
N
17
1
5
11
5
6
F
0
4
9
D
2
M
6
8

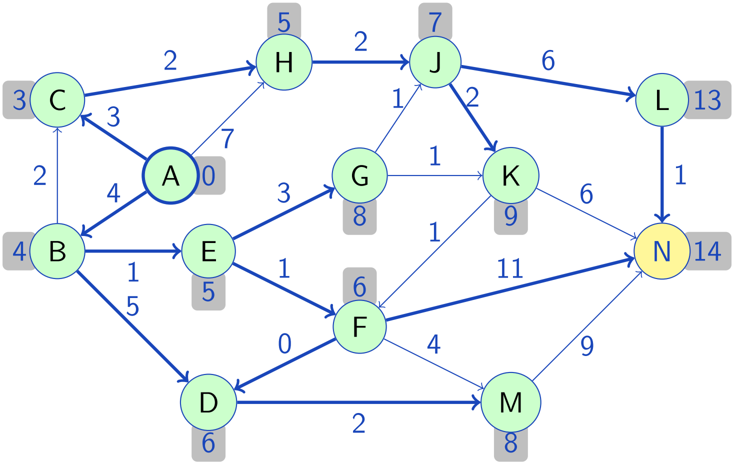

A
0
B
4
C
3
D
6
E
5
F
6
G
8
H
5
J
7
K
9
L
13
M
8
N
15
2
2
6
3
7
1
2
2
4
1
1
6
1
3
1
1
11
5
1
0
4
9
2

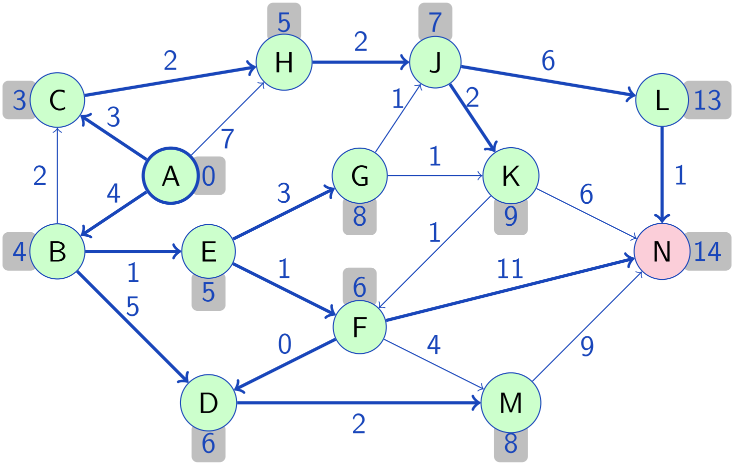

5
7
2
H
J
6
3
C
2
L
13
3
1
2
7
A
0
G
1
K
2
4
1
6
3
8
9
1
4
B
E
N
15
1
11
1
5
5
6
F
0
4
9
D
M
2
6
8

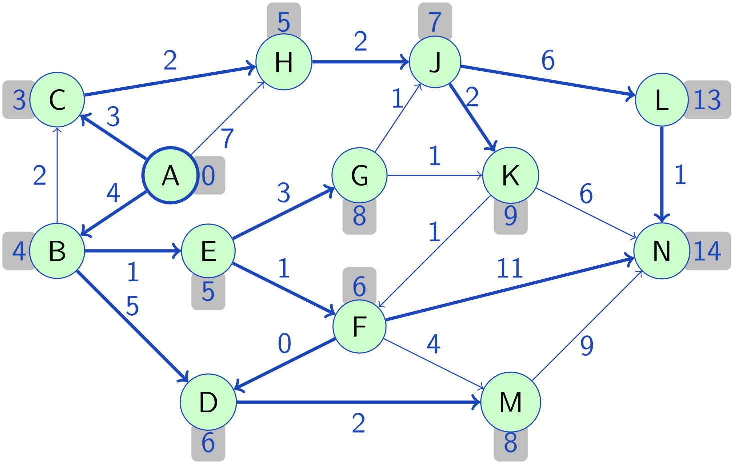

A
0
B
4
C
3
D
6
E
5
F
6
G
8
H
5
J
7
K
9
L
13
M
8
N
15
2
3
7
2
4
1
5
3
1
0
2
2
1
2
6
1
1
4
11
6
1
9


A
B
C
D
E
F
G
H
J
K
L
M
N
0
4
3
6
5
6
8
5
7
9
13
8
15
2
3
7
2
4
1
5
2
1
3
1
0
1
1
2
6
2
1
6
1
11
4
9
2


A
0
B
4
C
3
D
6
E
5
F
6
G
8
H
5
J
7
K
9
L
13
M
8
N
14
2
3
7
2
4
1
5
3
1
0
2
2
1
1
1
2
6
6
1
11
4
9


A
0
B
4
C
3
D
6
E
5
F
6
G
8
H
5
J
7
K
9
L
13
M
8
N
14
2
3
7
2
4
1
5
2
6
1
2
1
3
1
1
6
1
11
0
4
2
9


A
0
B
4
C
3
D
6
E
5
F
6
G
8
H
5
J
7
K
9
L
13
M
8
N
14
2
2
6
3
7
1
2
2
1
1
4
3
6
1
1
1
11
5
0
4
9
2


5
H
2
7
J
6
L
13
3
C
2
3
7
1
2
A
0
1
G
K
2
4
3
1
6
8
9
4
B
E
1
N
14
1
11
1
5
6
5
F
0
4
9
D
M
2
6
8

- Recall: How to update S
  - Choose $v \notin S$ with the smallest $D'[v]$
  - Set $D[v] = D'[v]$
  - $S \leftarrow S \cup \{v\}$
  - For all edges $(v, v') \in E$, update $D'[v'] \leftarrow \min(D'[v'], D[v] + L(v, v'))$

- Recall: How to update S
  - Choose $v \notin S$ with the smallest $D'[v]$
  - Set $D[v] = D'[v]$
  - $S \leftarrow S \cup \{v\}$
  - For all edges $(v, v') \in E$, update $D'[v'] \leftarrow \min(D'[v'], D[v] + L(v, v'))$
- How to choose vertices with smallest distance?

- Recall: How to update S
  - Choose $v \notin S$ with the smallest $D'[v]$
  - Set $D[v] = D'[v]$
  - $S \leftarrow S \cup \{v\}$
  - For all edges $(v, v') \in E$, update $D'[v'] \leftarrow \min(D'[v'], D[v] + L(v, v'))$
- How to choose vertices with smallest distance?
  1. Naïve way. Iterate each time over vertices
     - Running time of one iteration: $O(|V|)$
     - $|V|$ iterations, $O(|V|^2)$ total time. Good for dense graphs, bad for sparse ones

- Recall: How to update S
  - Choose $v \notin S$ with the smallest $D'[v]$
  - Set $D[v] = D'[v]$
  - $S \leftarrow S \cup \{v\}$
  - For all edges $(v, v') \in E$, update $D'[v'] \leftarrow \min(D'[v'], D[v] + L(v, v'))$
- How to choose vertices with smallest distance?
  1. Naïve way. Iterate each time over vertices
     - Running time of one iteration: $O(|V|)$
     - $|V|$ iterations, $O(|V|^2)$ total time. Good for dense graphs, bad for sparse ones
  2. Using binary heap
     - Choosing – "extract minimum": $O(\log |V|)$, at most $|V|$ operations
     - Updating – "decrease key": $O(\log |V|)$, at most $|E|$ operations
     - Total running time: $O(|E| \log |V|)$. Good for sparse graphs, bad for dense ones

- Recall: How to update S
  - Choose $v \notin S$ with the smallest $D'[v]$
  - Set $D[v] = D'[v]$
  - $S \leftarrow S \cup \{v\}$
  - For all edges $(v, v') \in E$, update $D'[v'] \leftarrow \min(D'[v'], D[v] + L(v, v'))$
- How to choose vertices with smallest distance?
  1. Naïve way. Iterate each time over vertices
     - Running time of one iteration: $O(|V|)$
     - $|V|$ iterations, $O(|V|^2)$ total time. Good for dense graphs, bad for sparse ones
  2. Using binary heap
     - Choosing – "extract minimum": $O(\log |V|)$, at most $|V|$ operations
     - Updating – "decrease key": $O(\log |V|)$, at most $|E|$ operations
     - Total running time: $O(|E| \log |V|)$. Good for sparse graphs, bad for dense ones
  3. Using Fibonacci heap: "decrease key" in amortized $O(1)$ time
     - Total running time: $O(|V| \log |V| + |E|)$. However, impractical :(

Contest trick: How to implement Dijkstra with heap using standard libraries?

- Libraries of C++ and Java do not support heaps with "decrease key" operation

Contest trick: How to implement Dijkstra with heap using standard libraries?

- Libraries of C++ and Java do not support heaps with “decrease key” operation
- Basically two choices:

Contest trick: How to implement Dijkstra with heap using standard libraries?

- Libraries of C++ and Java do not support heaps with "decrease key" operation
- Basically two choices:
  1. Simulate binary heap with a binary tree of vertices ordered by distance
     - "Extract minimum": remove the minimum (leftmost) vertex
     - "Decrease key": remove vertex, change distance, insert vertex

Contest trick: How to implement Dijkstra with heap using standard libraries?

- Libraries of C++ and Java do not support heaps with "decrease key" operation
- Basically two choices:
  1. Simulate binary heap with a binary tree of vertices ordered by distance
     - "Extract minimum": remove the minimum (leftmost) vertex
     - "Decrease key": remove vertex, change distance, insert vertex
     - $O(|E| \log |V|)$ but quite slow due to tree's hidden constant

Contest trick: How to implement Dijkstra with heap using standard libraries?

- Libraries of C++ and Java do not support heaps with "decrease key" operation
- Basically two choices:
  1. Simulate binary heap with a binary tree of vertices ordered by distance
     - "Extract minimum": remove the minimum (leftmost) vertex
     - "Decrease key": remove vertex, change distance, insert vertex
     - $O(|E| \log |V|)$ but quite slow due to tree's hidden constant
  2. Simulate binary heap with a priority queue of vertex-distance pairs
     - "Extract minimum": remove the minimum record,
       discard and continue if distance in pair differs from current distance to that vertex
     - "Decrease key": just insert a vertex-distance pair with the new distance

Contest trick: How to implement Dijkstra with heap using standard libraries?

- Libraries of C++ and Java do not support heaps with "decrease key" operation
- Basically two choices:
  1. Simulate binary heap with a binary tree of vertices ordered by distance
     - "Extract minimum": remove the minimum (leftmost) vertex
     - "Decrease key": remove vertex, change distance, insert vertex
     - $O(|E| \log |V|)$ but quite slow due to tree's hidden constant
  2. Simulate binary heap with a priority queue of vertex-distance pairs
     - "Extract minimum": remove the minimum record, discard and continue if distance in pair differs from current distance to that vertex
     - "Decrease key": just insert a vertex-distance pair with the new distance
     - $O(|E| \log |E|)$ but rather fast, generally better than tree