Algorithms and Data Structures

Lecture 10

Graph Algorithms III: Shortest Paths

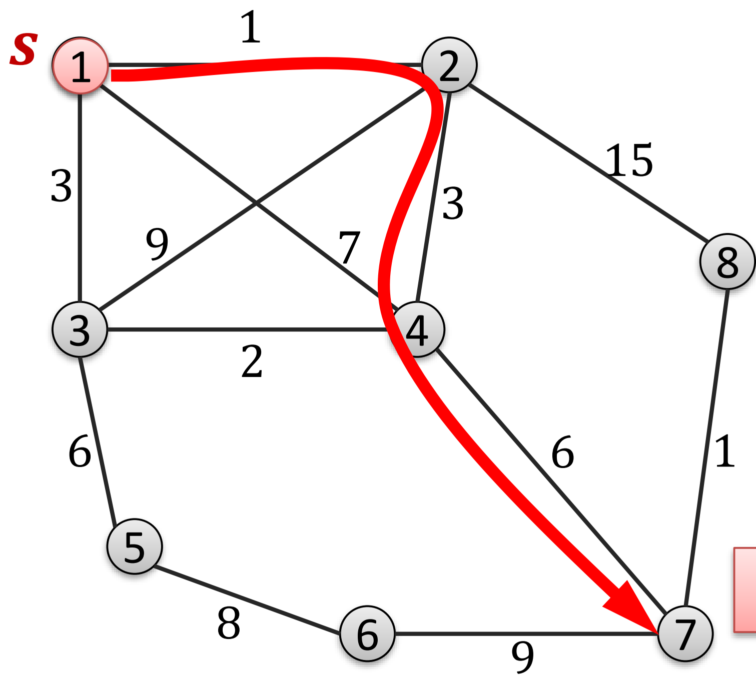
Fabian Kuhn
Algorithms and Complexity

UNI FREIBURG

Shortest Paths

Single Sourse Shortest Paths Problem

- Given: weighted graph $G = (V, E, w)$, start node $s \in V$
  - We denote the weight of an edge (u, v) by $w(u, v)$
  - Assumption for now: $\forall e \in E: w(e) \geq 0$
- Goal: Find shortest paths / distances from s to all nodes
  - Distance from s to v: $d_G(s, v)$ (length of a shortest path)

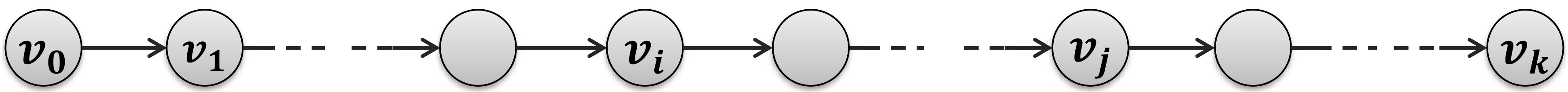


Distance from node 1 to node 7 : 10

Optimality of Subpaths

Lemma: If $v_0, v_1, \dots, v_k$ is a shortest path from v_0 to v_k, then it holds for all $0 \le i \le j \le k$ that the subpath $v_i, v_{i+1}, \dots, v_j$ is also a shortest path from v_i to v_j.

Shortest path from v_0 to v_k:

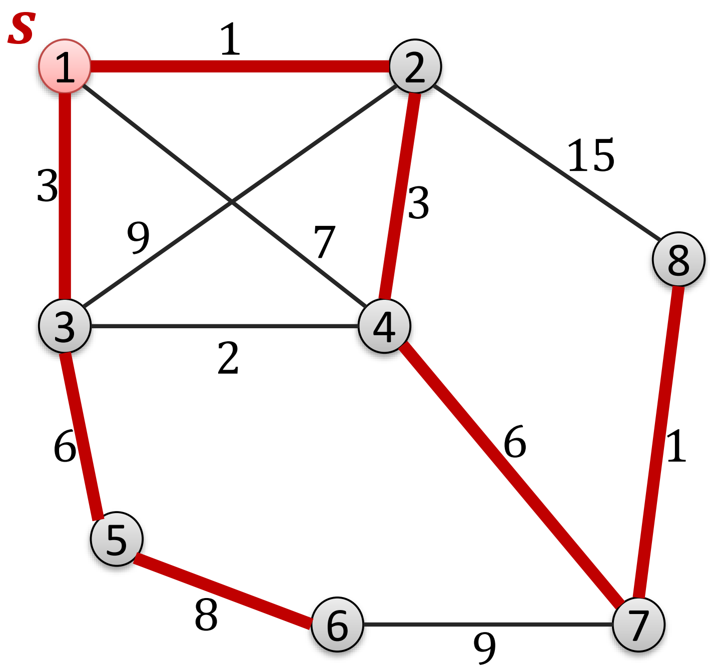


- Subpath from v_i to v_j is also a shortest path.
  - Otherwise, one could replace the path from v_i to v_j by the shortest path from v_i to v_j.
  - If by doing this, nodes are visited multiple time, one can cut out cycles and obtains an even shorter path.

- Lemma also holds for negative edge weights,
  - as long as the graph does not contain negative cycles.

Shortest-Path Tree

- Spanning tree that is rooted at node s and that contains shortest paths from s to all other nodes.
  - Such a tree always exists (follows from the optimality of subpaths)
- For unweighted graphs: BFS spanning tree
- **Goal:** Find a shortest path tree



Dijkstra's Algorithm: Idea

- Algorithm by Edsger W. Dijkstra (published in 1959)

Idea:

- We start at s and build the spanning tree in a step-by-step manner.

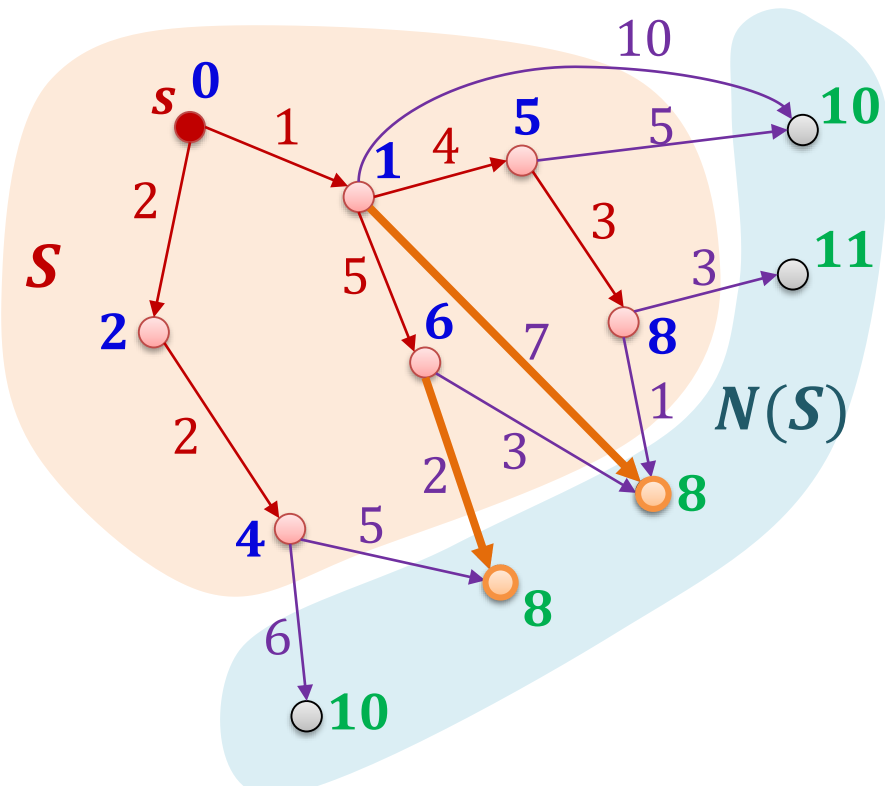
> **Invariant:**
> Algorithm always has a tree rooted at s, which is a subtree of a shortest path tree.

- Goal: In each step of the algorithm, add one node
  - Initially: subtree only consists of s
    (trivially satisfies invariant...)
  - 1st step: Because of the optimality of subpaths, there must be a shortest path consisting of a single edge...
  - Always add the remaining node at the smallest distance from s.

Dijkstra's Algorithm : One Step

Given: A tree T that is rooted in s, such that T is a subtree of a shortest paths tree for node s in G. (nodes of T : S)

How can we extend T by a single node?



S : nodes in the tree T

$N(S)$: nodes that can be added to the tree directly.

To add $v \in N(S)$ it most hold that

$$d_G(s,v) = \min_{u \in S}\{d_G(s,u) + w(u,v)\}$$

We will see that this always holds for $v \in N(S)$ with minimum distance $d_G(s,v)$ from s.

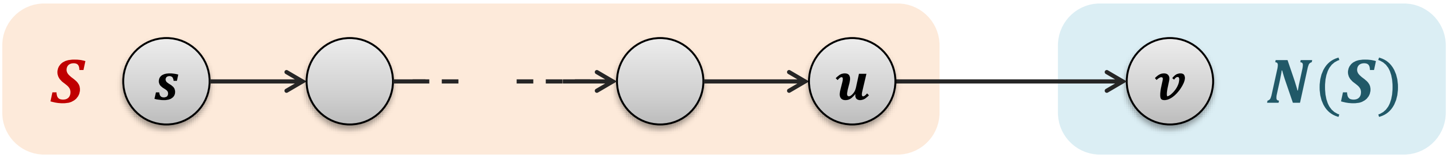
Dijkstra's Algorithm : One Step

Given: T is subtree of a shortest path tree for s in G.

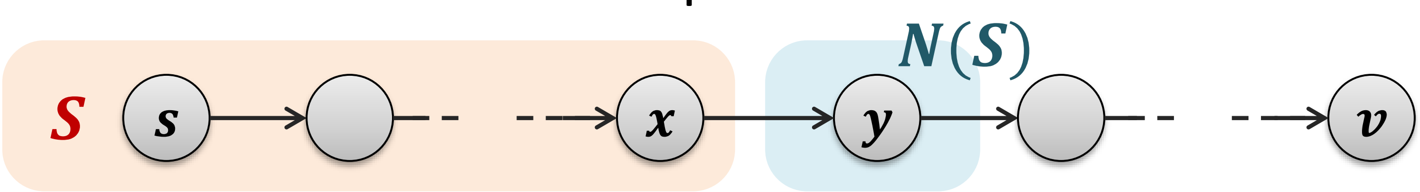
Lemma: For a node $v \in N(S)$ and an edge (u, v) with $u \in S$ such that $d_G(s, u) + w(u, v)$ is minimized, it holds that

$$\boldsymbol{d_G(s, v) = d_G(s, u) + w(u, v)}$$

Consider the s-v path that we obtain in this way:



Assume that there is a shorter path:



– Because there are no negative edge weights, we therefore have

$$d_G(s, x) + w(x, y) \leq d_G(s, v) < d_G(s, u) + w(u, v)$$

Dijkstra's Algorithm

Invariant:
Algorithm always has a tree $T = (S, A)$ rooted at s, which is a subtree of a shortest path tree of G.

- At the beginning, we have $T = (\{s\}, \emptyset)$
- For each node $v \notin S$, one at all times computes

$$\delta(s, v) := \min_{u \in S \cap N_{\text{in}}(v)} d_G(s, u) + w(u, v)$$

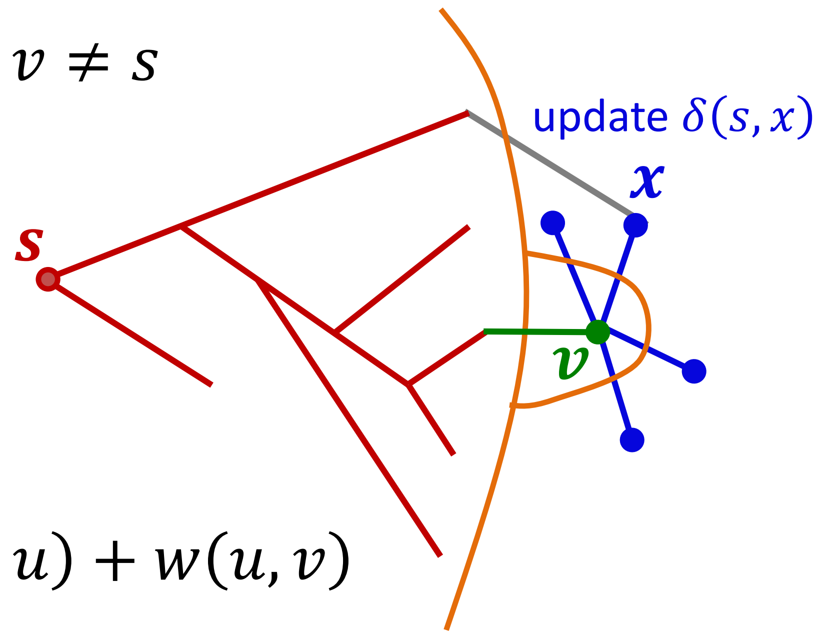
  - as well as the incoming neighbor $u =: \alpha(v)$ that minimized the expression...
- $\delta(s, v)$ corresponds to an s-v path $\implies \delta(s, v) \geq d_G(s, v)$
- Lemma on last slide:

  For minimum $\delta(s, v)$, we have: $\boldsymbol{\delta(s, v) = d_G(s, v)}$

Dijkstra's Algorithm

Initialization $\boldsymbol{T} = (\emptyset, \emptyset)$

- $\delta(s,s) = 0$, and $\delta(s,v) = \infty$ for all $v \neq s$
- $\alpha(v) = \text{NULL}$ for all $v \in V$



Iteration Step

- Choose a node v with smallest

$$\delta(s,v) := \min_{u \in S \cap N_{\text{in}}(v)} d_G(s,u) + w(u,v)$$

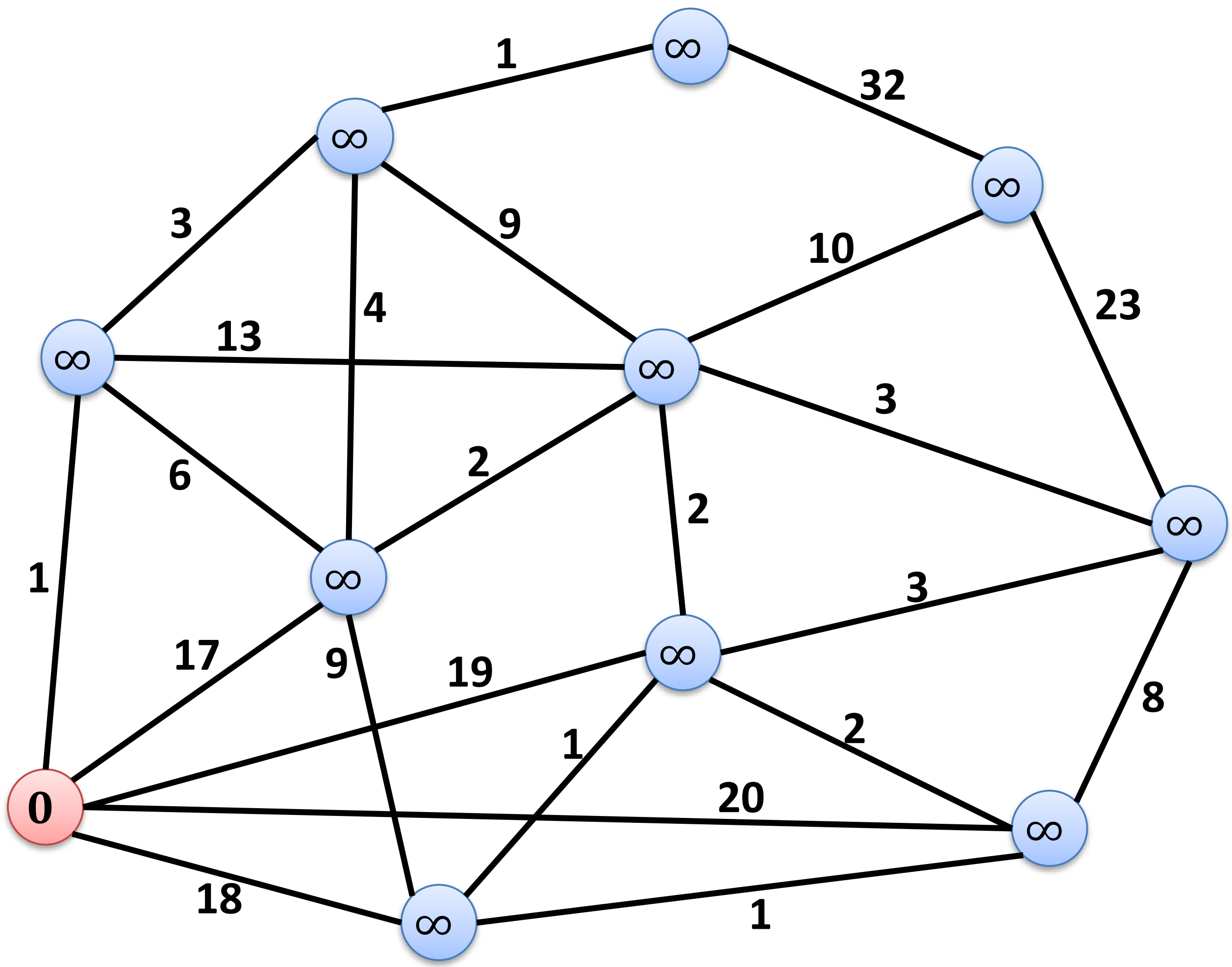
- Go through all out-neighbors $x \in V \setminus S$ and set

$$\delta(s,x) := \min\{\delta(s,x), \delta(s,v) + w(v,x)\}$$

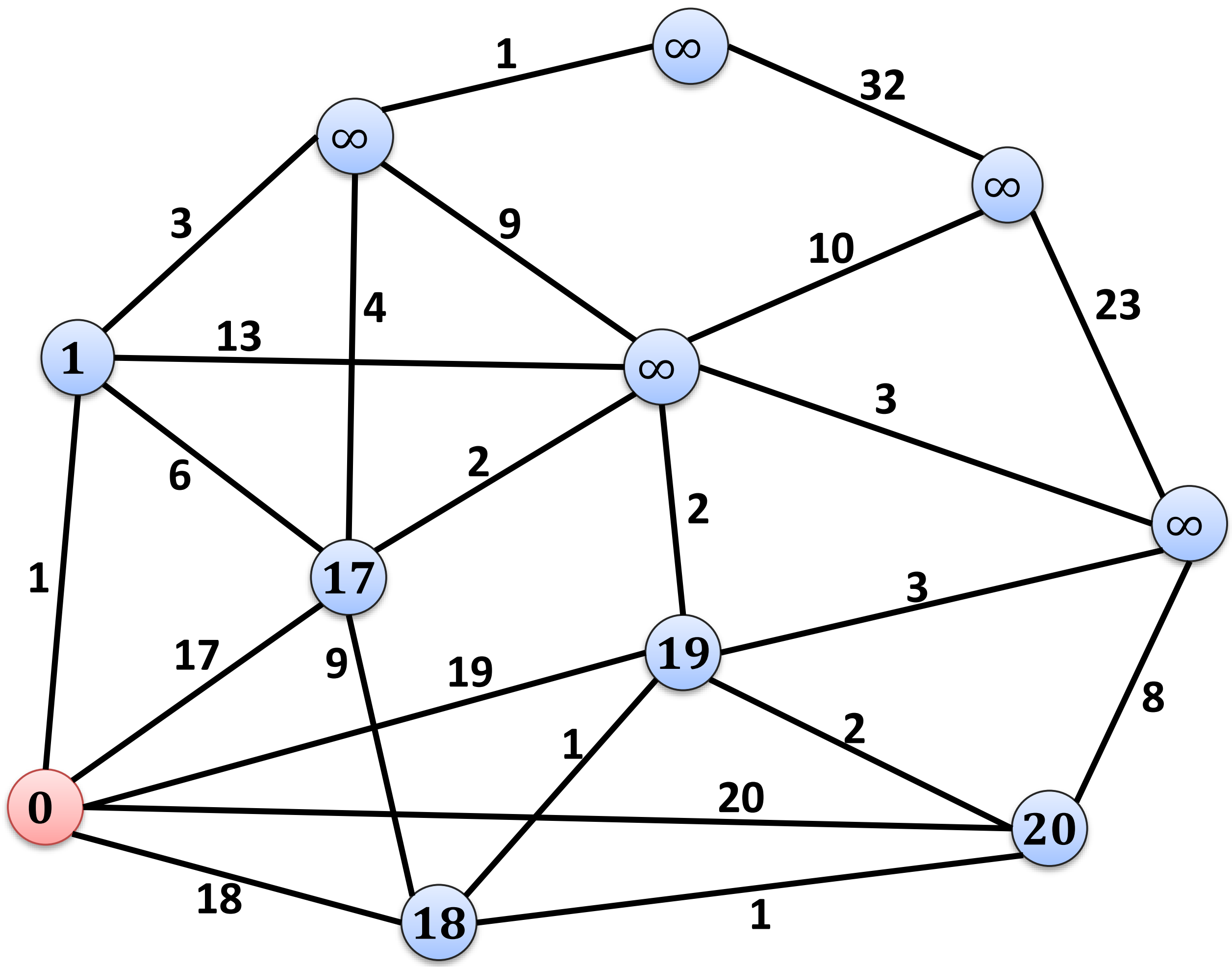
  - If $\delta(s,x)$ is decreased, set $\alpha(x) = v$

- Add node v and edge $(\alpha(v), v)$ to the tree T.

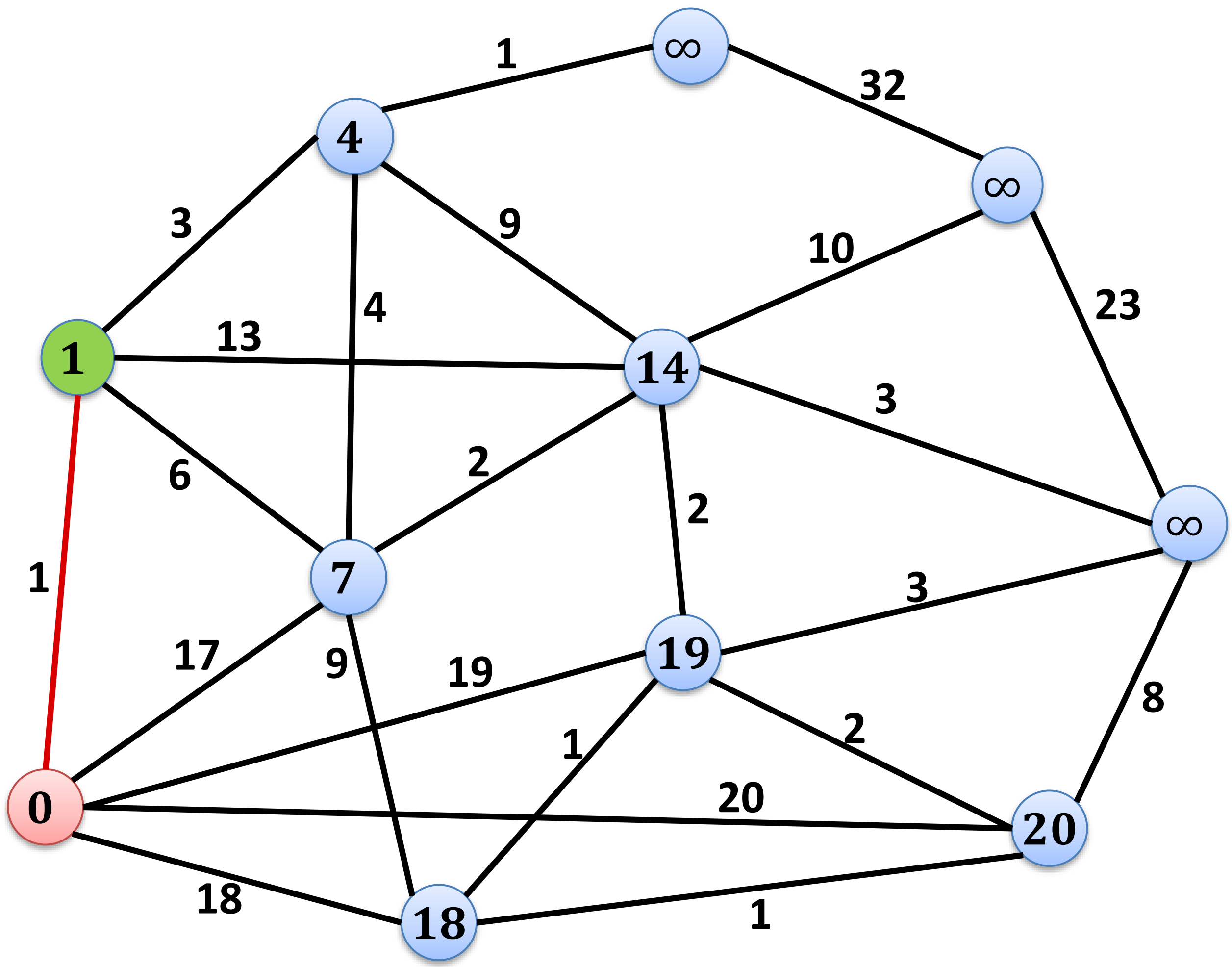
Dijkstra’s Algorithm: Example



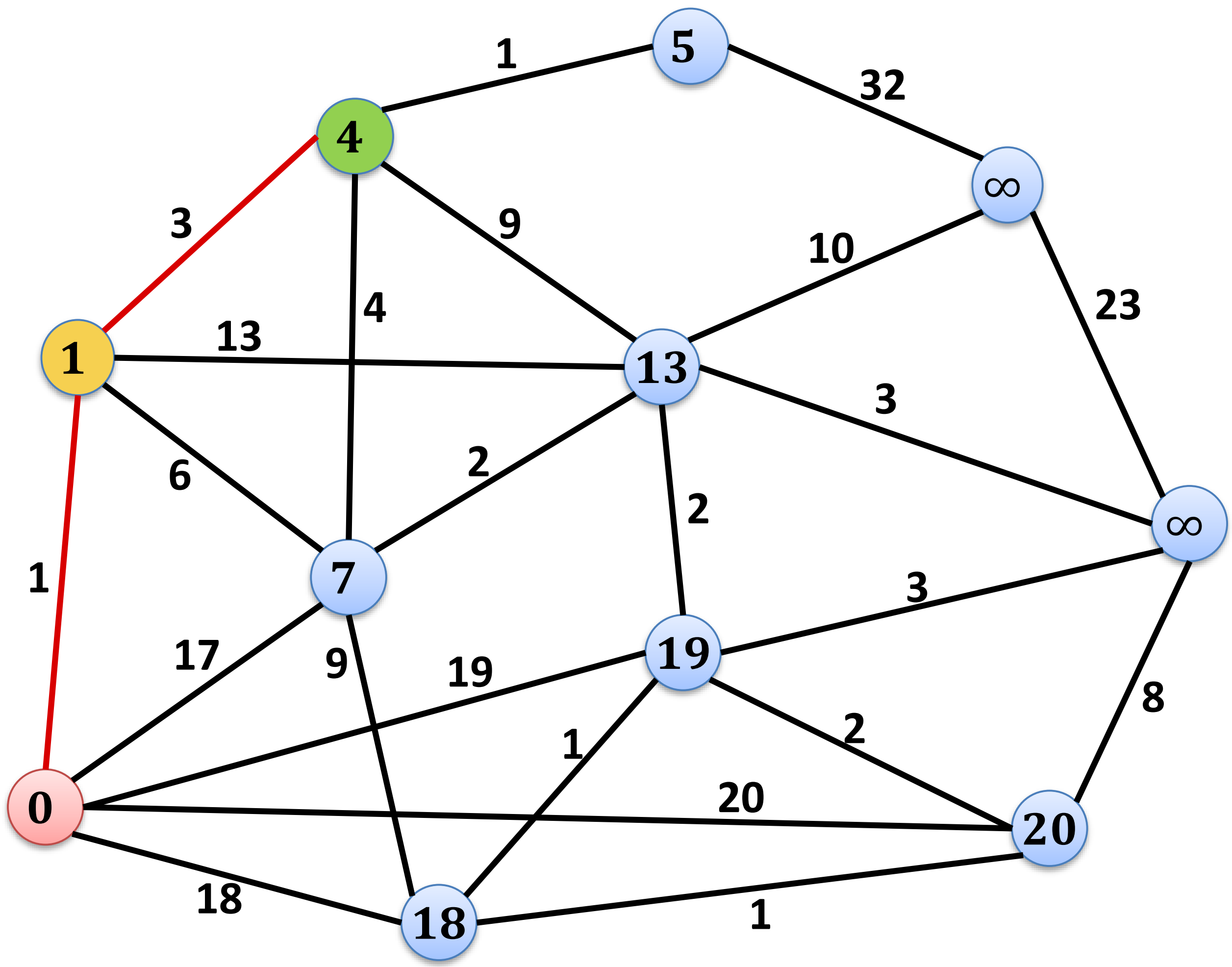
Dijkstra's Algorithm: Example



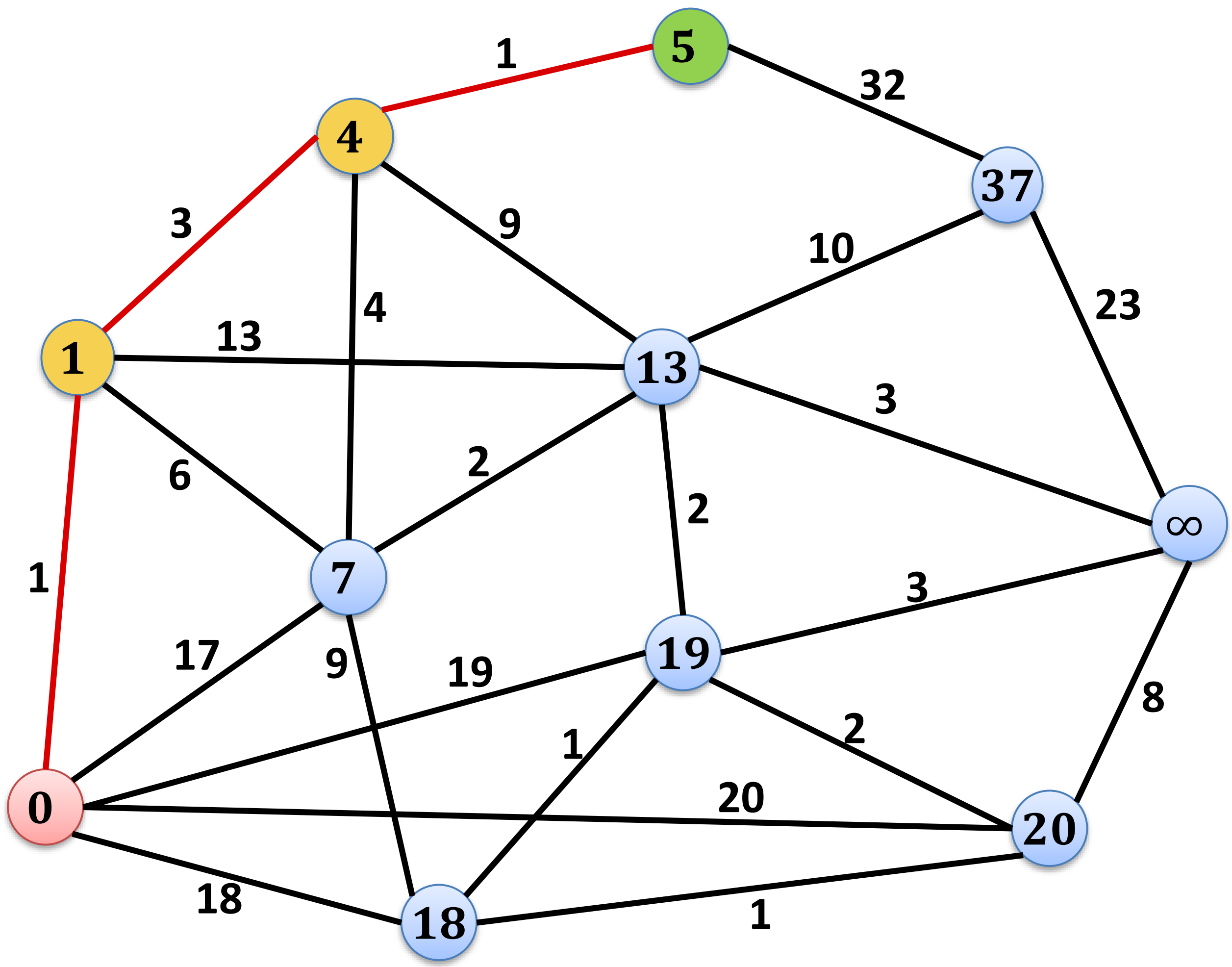
Dijkstra's Algorithm: Example



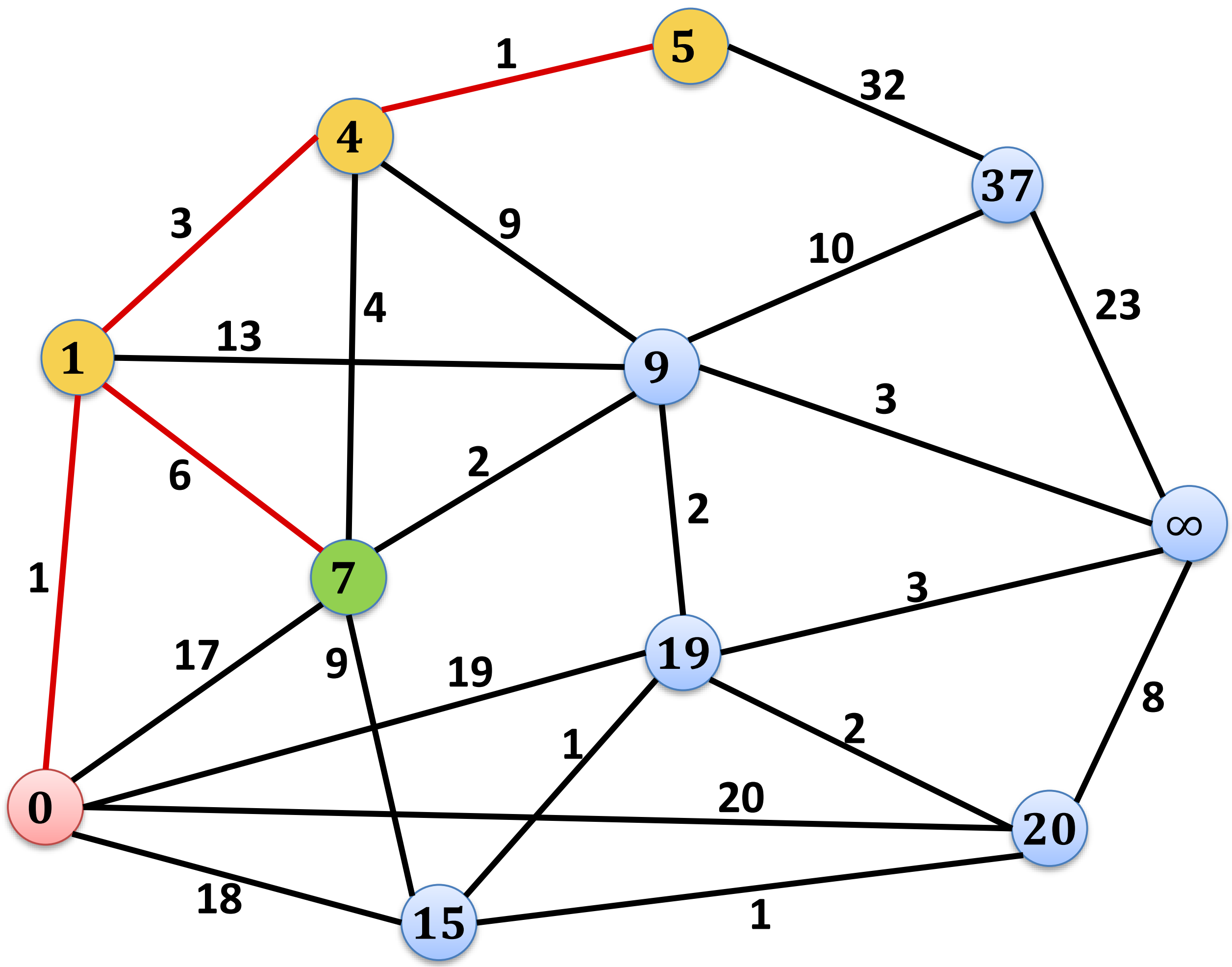
Dijkstra’s Algorithm: Example



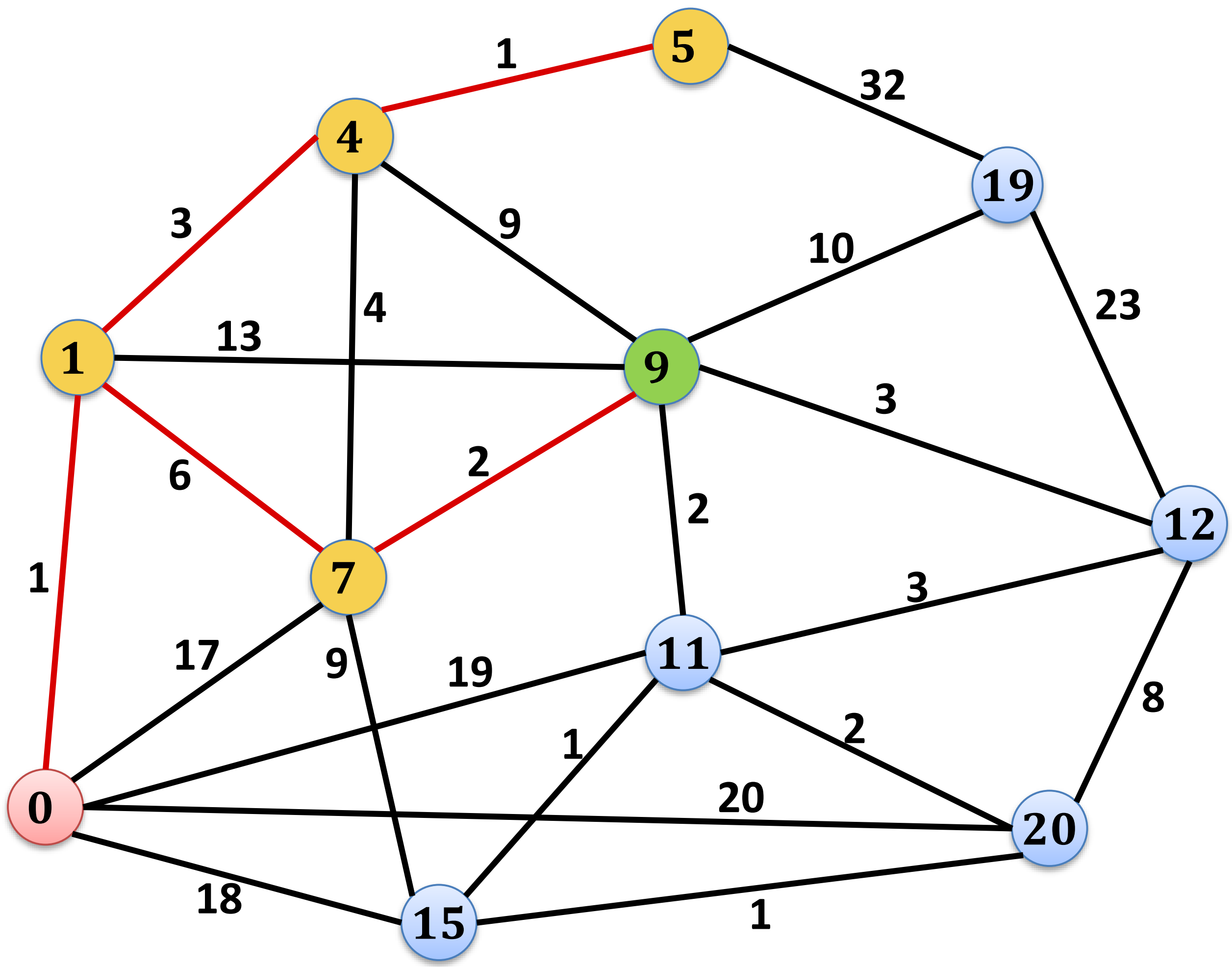
Dijkstra's Algorithm: Example



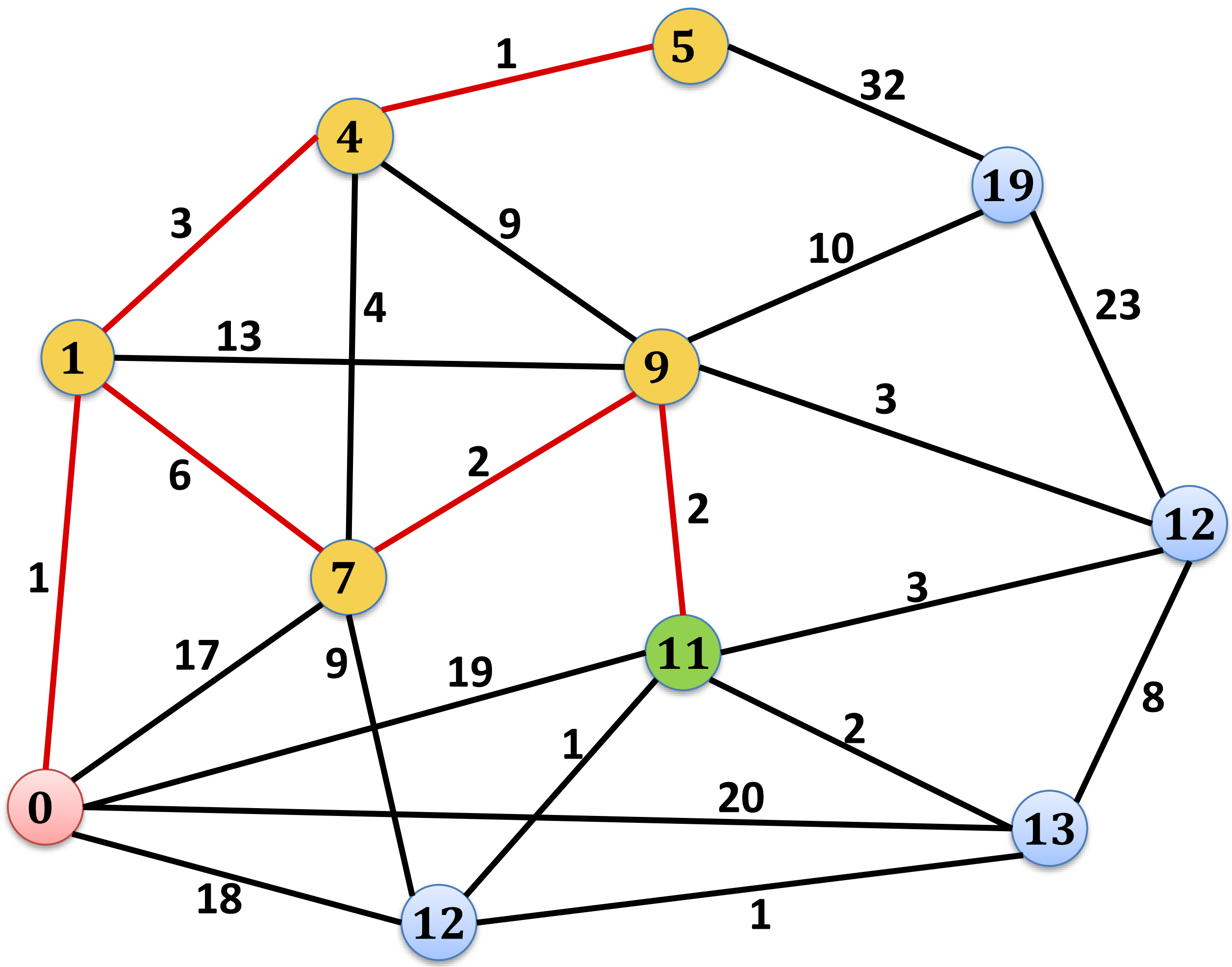
Dijkstra's Algorithm: Example



Dijkstra's Algorithm: Example



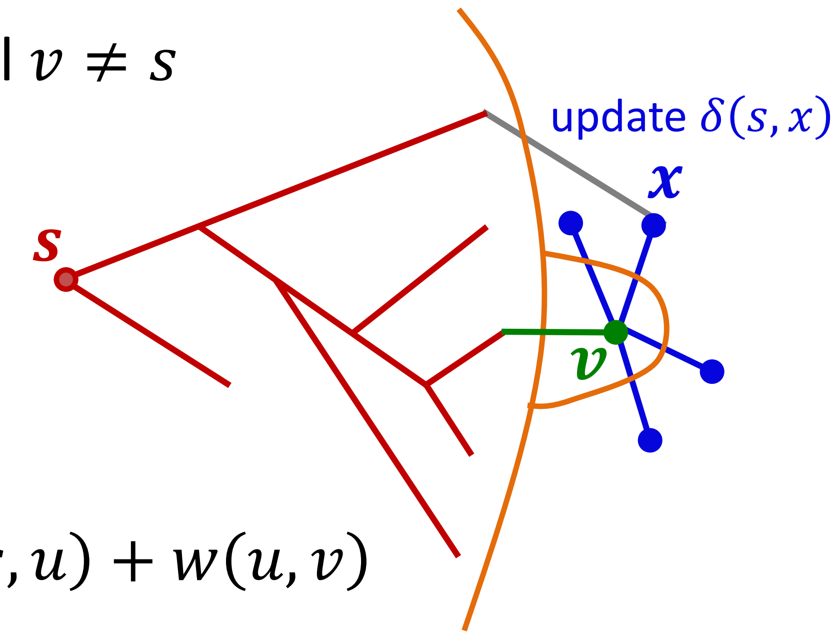
Dijkstra's Algorithm: Example



Dijkstra's Algorithm

Initialization $\boldsymbol{T} = (\emptyset, \emptyset)$

- $\delta(s,s) = 0, \quad$ and $\delta(s,v) = \infty$ for all $v \neq s$
- $\alpha(v) = \text{NULL}$ for all $v \in V$



Iteration Step

- Choose a node v with smallest

$$\delta(s,v) := \min_{u \in S \cap N_{\text{in}}(v)} d_G(s,u) + w(u,v)$$

- Go through all out-neighbors $x \in V \setminus S$ and set

$$\delta(s,x) := \min\{\delta(s,x), \delta(s,v) + w(v,x)\}$$

  – If $\delta(s,x)$ is decreased, set $\alpha(x) = v$

- Add node v and edge $(\alpha(v), v)$ to the tree T.

Similar to the MST algorithm of Prim!

Reminder : Prim's MST Algorithm

```
H = new priority queue; A = ∅
for all u ∈ V \ {s} do
    H.insert(u, ∞); α(u) = NULL
H.insert(s, 0)

while H is not empty do
    u = H.deleteMin()
    for all unmarked neighbors v of u do
        if w({u,v}) < d(v) then
            H.decreaseKey(v, w({u,v}))
            α(v) = u
    u.marked = true
    if u ≠ s then A = A ∪ {{u, α(u)}}
```

Dijkstra's Algorithm : Implementation

```
H = new priority queue; A = ∅
for all u ∈ V \ {s} do
    H.insert(u, ∞); δ(s,u) = ∞; α(u) = NULL
H.insert(s, 0)

while H is not empty do
    u = H.deleteMin()
    for all unmarked out-neighbors v of u do
        if δ(s,u) + w(u,v) < δ(s,v) then
            δ(s,v) = δ(s,u) + w(u,v)
            H.decreaseKey(v, δ(s,v))
            α(v) = u
    u.marked = true
    if u ≠ s then A = A ∪ {(α(u),u)}
```

Dijkstra's Algorithm: Running Time

- Algorithm implementation is almost identical to the implementation of Prim's MST algorithm.

- **Number of heap operations:**

  create: 1, insert: n, deleteMin: n, decreaseKey: $\leq m$

  – Or alternatively without decrease-key: $O(m)$ insert and deleteMin Op.

- **Running time with binary heap:**

$$\boldsymbol{O(m \log n)}$$

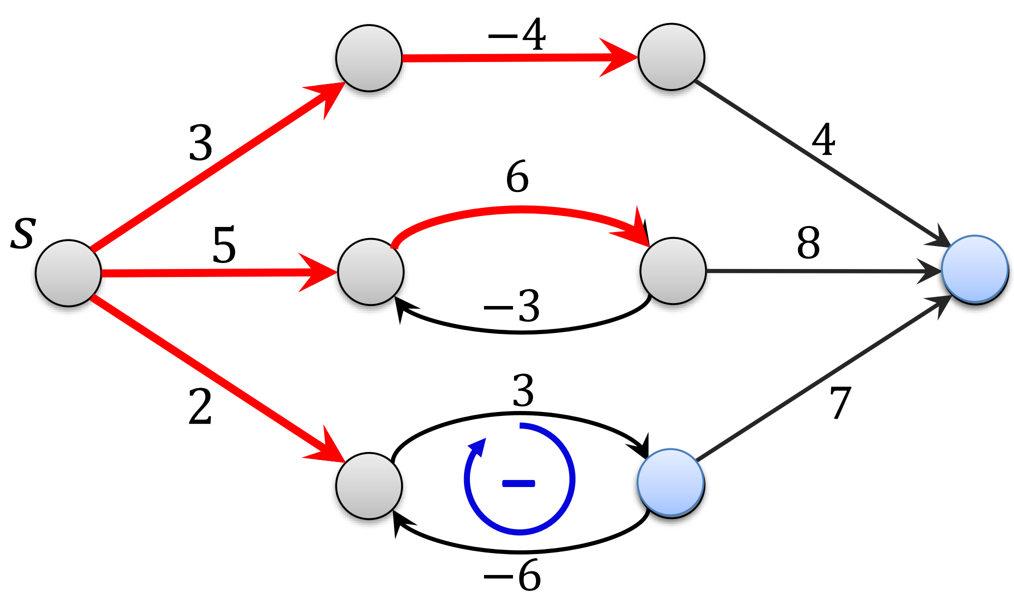
- **Running time with Fibonacci heap:**

$$\boldsymbol{O(m + n \log n)}$$

Negative Edge Weights

- Shortest paths can also be defined for graphs with negative edge weights.
  - Shortest path is defined if there no shorter way, even if nodes can be visited multiple times.

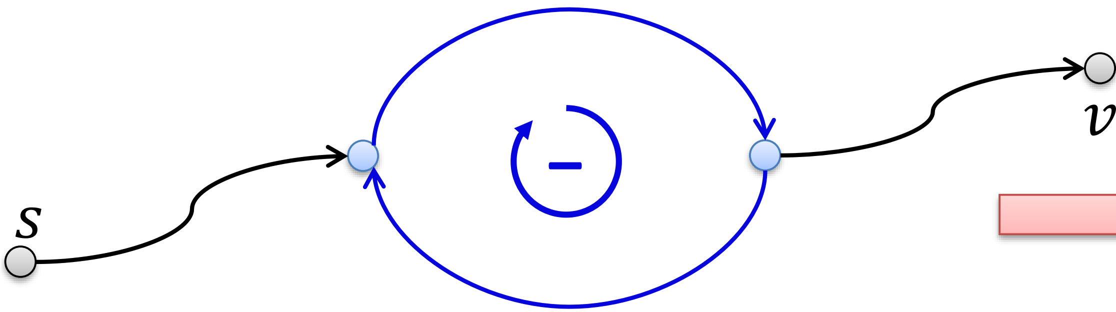
Example



Negative Edge Weights

Lemma: In a directed, weighted graph G, there is a shortest path from s to v if and only if there is no there is no negative cycle that is reachable from s and from which one can reach v.

- Also holds for undirected graphs if edges $\{u, v\}$ are considered as 2 directed edges (u, v) and (v, u).



→ no shortest path from u to v

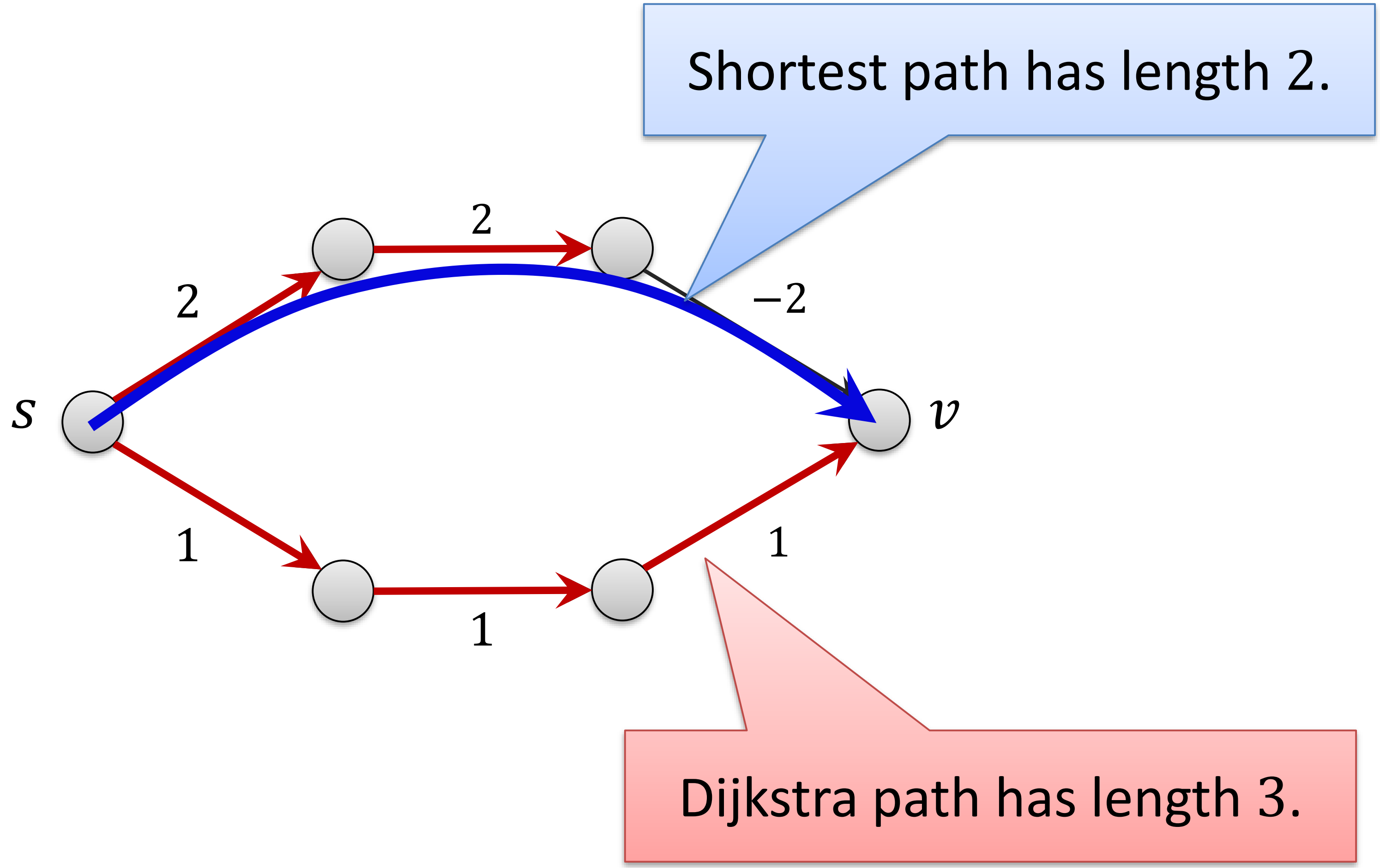
Nodes are not visited multiple times.

no reachable negative cycle → We can restrict our attention to simple path. There are only finitely many such path.

Dijkstra's Algorithm and Negative Weights

Does Dijkstra's algorithm work with negative edge weights?

- Answer: no



Bellman-Ford Algorithm

- To simplify, we only compute the distances $d_G(s, v)$

Assumption:

- For all nodes v: algorithm has dist. estimate $\boldsymbol{\delta(s, v) \geq d_G(s, v)}$
- Initialization: $\delta(s, s) = 0, \;\; \delta(s, v) = \infty$ for $v \neq s$

Observation:

- If $(u, v) \in E$ such that $\delta(s, u) + w(u, v) < \delta(s, v)$, then we can decrease (and thus improve) $\delta(s, v)$ because

$$\boldsymbol{d_G(s, v)} \leq d_G(s, u) + w(u, v)$$
$$\leq \boldsymbol{\delta(s, u) + w(u, v)}$$

Bellman-Ford Algorithm

- Consider all edges (u, v) and try to improve $\delta(s, v)$,
  - until all distances are correct ($\forall v \in V: \delta(s, v) = d_G(s, v)$)

$\delta(s, s) \coloneqq 0;\ \ \forall v \in V \setminus \{s\} : \delta(s, v) \coloneqq \infty$

repeat

 for all $(u, v) \in E$ **do**

  if $\delta(s, u) + w(u, v) < \delta(s, v)$ **then**

   $\delta(s, v) \coloneqq \delta(s, u) + w(u, v)$

until $\forall v \in V\colon\ \delta(s, v) = d_G(s, v)$

- How many repetitions are necessary?
  - Shortest paths consisting of one edge $\Longrightarrow$ 1 repetitions
  - Shortest paths consisting of two edges $\Longrightarrow$ 2 repetitions
  - …
  - Shortest paths consisting of k edges $\Longrightarrow$ k repetitions

Bellman-Ford Algorithm

$\delta(s,s) \coloneqq 0;\;\; \forall v \in V \setminus \{s\} : \delta(s,v) \coloneqq \infty$

for i := 1 to n-1 **do**

    for all $(u,v) \in E$ **do**

        if $\delta(s,u) + w(u,v) < \delta(s,v)$ **then**

            $\delta(s,v) \coloneqq \delta(s,u) + w(u,v)$

After i repetitions, we have $\boldsymbol{\delta(s,v) \leq d_G^{(i)}(s,v)}$, where $d_G^{(i)}(s,v)$ is the length of a shortest path consisting of at most i edges.

- **Follows by induction on i:**
  - $i = 0$: $\delta(s,s) = d_G^{(0)}(s,s) = 0,\; v \neq s \implies \delta(s,v) = d_G^{(0)}(s,v) = \infty$
  - $i > 0$:

    $$d_G^{(i)}(s,v) = \min\left\{d_G^{(i-1)}(s,v),\; \min_{u \in N^{in}(v)} d_G^{(i-1)}(s,u) + w(u,v)\right\}$$

    (shortest path consists of $\leq i-1$ edges or of exactly i edges)

Bellman-Ford Algorithm

$\delta(s,s) \coloneqq 0;\ \ \forall v \in V \setminus \{s\} : \delta(s,v) \coloneqq \infty$

```
for i := 1 to n-1 do
    for all (u,v) ∈ E do
        if δ(s,u) + w(u,v) < δ(s,v) then
            δ(s,v) := δ(s,u) + w(u,v)
```

Theorem: If the graph has no negative cycles that are reachable from s, at the end all distances are computed correctly.

- At the end, we have for all $v \in V$:

$$\delta(s,v) \leq d_G^{(n-1)}(s,v)$$

- Because every path consists of $\leq n-1$ edges, we also have

$$d_G^{(n-1)}(s,v) = d_G(s,v)$$

Detecting Negative Cycles

- We will see: If there is a (from s reachable) negative cycle, then there is an improvement for some edge:

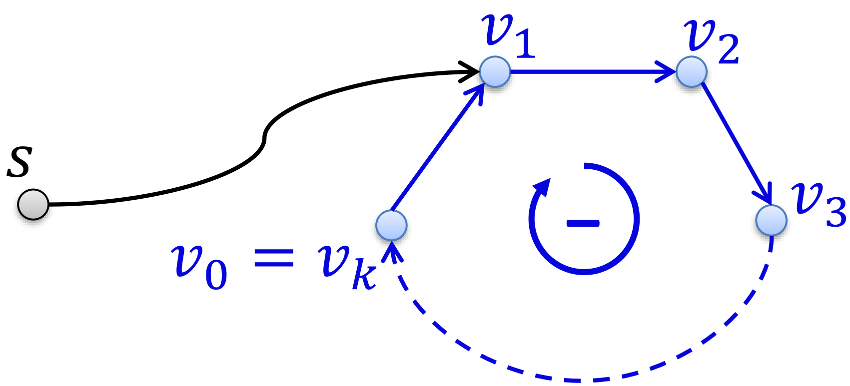
$$\exists (u, v) \in E : \delta(s, u) + w(u, v) < \delta(s, v)$$

Bellman-Ford Algorithm

for i := 1 to n-1 **do**
  for all $(u, v) \in E$ **do**
    if $\delta(s, u) + w(u, v) < \delta(s, v)$ **then**
      $\delta(s, v) := \delta(s, u) + w(u, v)$
for all $(u, v) \in E$ **do**
  if $\delta(s, u) + w(u, v) < \delta(s, v)$ **then**
    return false
return true

Detecting Negative Cycles

Lemma: If G contains a negative cycles that is reachable from s, then the Bellman-Ford algorithm returns false.



neg. cycle $\Longrightarrow \displaystyle\sum_{i=1}^{k} w(v_{i-1}, v_i) \overset{!}{<} 0$

reachable from $s \Longrightarrow \delta(s, v_i) \neq \infty$

Proof by contradiction:

- Assumption : $\forall i \in \{1, \dots, k\} : \delta(s, v_{i-1}) + w(v_{i-1}, v_i) \geq \delta(s, v_i)$

$$\sum_{i=1}^{k} \delta(s, v_i) \leq \sum_{i=1}^{k} \left(\delta(s, v_{i-1}) + w(v_{i-1}, v_i)\right)$$

$$= \sum_{i=1}^{k} \delta(s, v_{i-1}) + \underbrace{\sum_{i=1}^{k} w(v_{i-1}, v_i)}_{< 0}$$

$\sum_{i=1}^{k} \delta(s, v_i) = \sum_{i=1}^{k} \delta(s, v_{i-1})$

Bellman-Ford Algorithm : Shortest Paths

A shortest path tree can be computed in the usual way.

Initialization:

- $\delta(s,s) = 0$, für $v \neq s : \delta(s,v) = \text{NULL}$
- $\alpha(s) = \text{NULL}$, for $v \neq s : \alpha(v) = \text{NULL}$

In every loop iteration:

…

$$\textbf{if } \delta(s,u) + w(u,v) < \delta(s,v) \textbf{ then}$$
$$\delta(s,v) \coloneqq \delta(s,u) + w(u,v)$$
$$\alpha(v) \coloneqq u$$

- At the end, $\alpha(v)$ points to a parent in the shortest path tree
  - if there are no negative cycles…

Bellman-Ford Algorithm : Summary

Theorem: If there is a negative cycle that is reachable from s, the Bellman-Ford algorithm detects this. If no such cycle exists, the Bellman-Ford algorithm computes a shortest path tree in time $O(|V| \cdot |E|)$.

- **Correctness:** already proven
- **Running time:**
  - $n - 1 + 1$ loop iterations
  - In every loop iteration, we go once through all the edges.
- **Remark:** One can adapt the algorithm such that it computes a shortest path for all v, for shich such a path from s existsts (and it detects if no shortest path exists).
  - in the same asymptotic running time

Routing Paths in Networks

Goal: Optimal routing paths for some destination t

- For every node, we want to know to which neighbor one has to send a message destined at node t.
- This corresponds to computing a shortest path tree if all edges are reversed (transpose graph)

Algorithm:

- Nodes remember tha current distance $\delta(u,t)$ and the currently best neighbor.
- All nodes in parallel check if there is an improvement for some neighbor:
$$\exists(u,v) \in E : w(u,v) + \delta(v,t) < \delta(u,t)$$
- Corresponds to a parallel variant of the Bellman-Ford algorithm

Shortest Paths Between All Node Pairs

- **all pairs shortest paths problem**

Compute single-source shortest paths for all nodes

- Dijkstra algorithm with all nodes:

  Running time: $n \cdot O(\text{Running time Dijkstra}) \in O(mn + n^2 \log n)$
  - Problem: only works for non-negative edge weights

- Bellman-Ford algorithm with all nodes:

  Running time: $n \cdot O(\text{Running time BF}) \in O(mn^2) \in O(n^4)$
  - Problem: slow...
  - If the Bellman-Ford algorithm is carried out for all nodes, the running time can be improved to $O(n^3 \cdot \log n)$.
  - If all $d_G^{(i)}(u,v)$-distances are known, one can directly compute the $d_G^{(2i)}(u,v)$-distances in one iteration.
    - Further details and discussion of other algorithms in various text books.