Shortest paths algorithms

Further path problems

- Shortest paths in directed graphs with general edge weights
  → **Bellman-Ford Algorithm** determines shortest paths starting at some node s, or „there are negative cycles in the graph that are reachable from s“.
  → in general: NP-complete

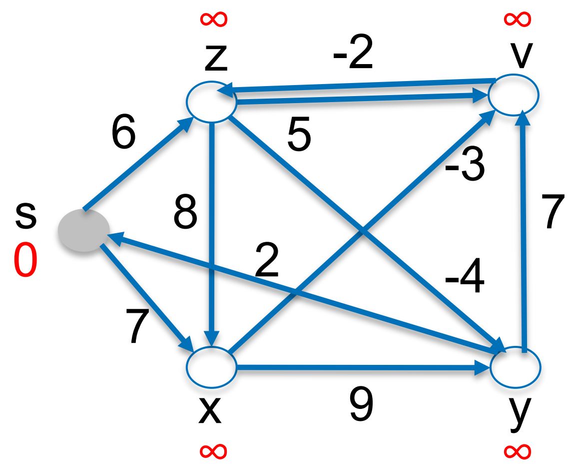
- Shortest paths in directed graphs without cycles.
  → there are fast algorithms, even with general edge weights

- **longest paths**
  - is there a simple path from some node s to naother one t in G such that each node of the graph is traversed exactly once? Problem is called: **Hamilton path problem**. → NP-complete
  - Is there a **Hamilton cycle**, i.e. a Hamiltom path from s to s in G? → NP-complete

Shortest paths algorithms

Bellman-Ford Algorithmus

```
1: Initialize(G,s)  // for all nodes v≠s: π[v]:=nil; dist[v]:=∞; dist[s]:=0;π[s]:=nil;
2: for i := 1 to |V| - 1 do
3:     for each edge (u,v)∈E do
4:         if dist[v] > dist[u] + f(u,v) then
5:             dist[v] := dist[u] + f(u,v);
6:             π[v] := u;
7:  for each edge (u,v)∈E do
8:      if dist[v] > dist[u] + f(u,v)
9:          return false
10: return true
```



Edge order, line 3: (v,z), (x,v), (x,y), (y,v), (y,s), (z,v), (z,x), (z,y), (s,x), (s,z)

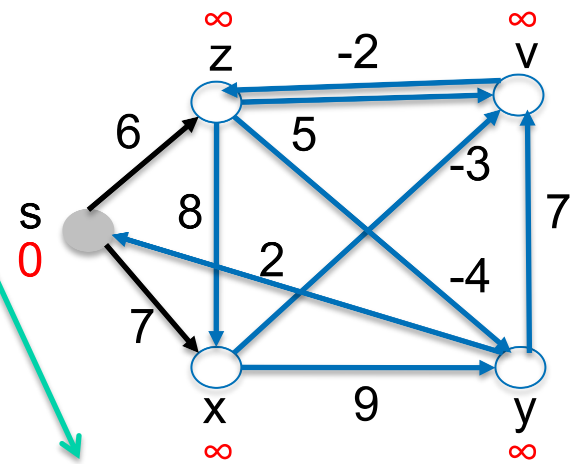
Rechenzeit: O(|V||E|)

Shortest paths algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Bellman-Ford Algorithmus

1: Initialize(G,s) // for all nodes v≠s: π[v]:=nil; dist[v]:=∞; dist[s]:=0;π[s]:=nil;
2: **for** i := 1 to |V| - 1 **do** → **i=1**
3: **for** each edge (u,v)∈E **do**
4: **if** dist[v] > dist[u] + f(u,v) **then**
5: dist[v] := dist[u] + f(u,v);
6: π[v] := u;
7: **for** each edge (u,v)∈E **do**
8: **if** dist[v] > dist[u] + f(u,v)
9: **return** false
10: **return** true



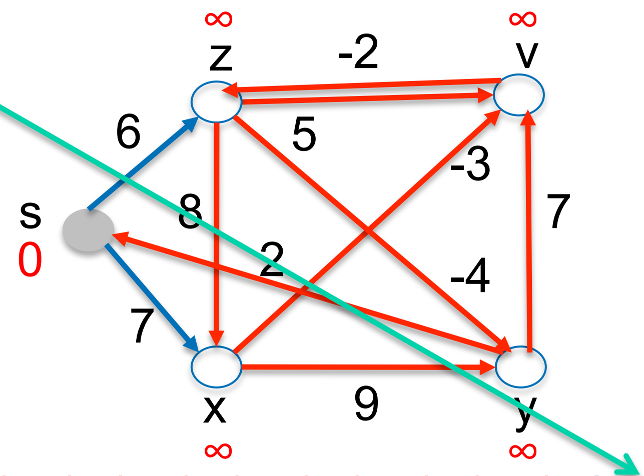
Edge order, line 3: **(v,z), (x,v), (x,y), (y,v), (y,s), (z,v), (z,x), (z,y)**, (s,x), (s,z)

Rechenzeit: O(|V||E|)

Shortest paths algorithms

Bellman-Ford Algorithmus

1: Initialize(G,s) // for all nodes v≠s: π[v]:=nil; dist[v]:=∞; dist[s]:=0;π[s]:=nil;
2: **for** i := 1 to |V| - 1 **do** → **i=1**
3: **for** each edge (u,v)∈E **do**
4: **if** dist[v] > dist[u] + f(u,v) **then**
5: dist[v] := dist[u] + f(u,v);
6: π[v] := u;
7: **for** each edge (u,v)∈E **do**
8: **if** dist[v] > dist[u] + f(u,v)
9: **return** false
10: **return** true



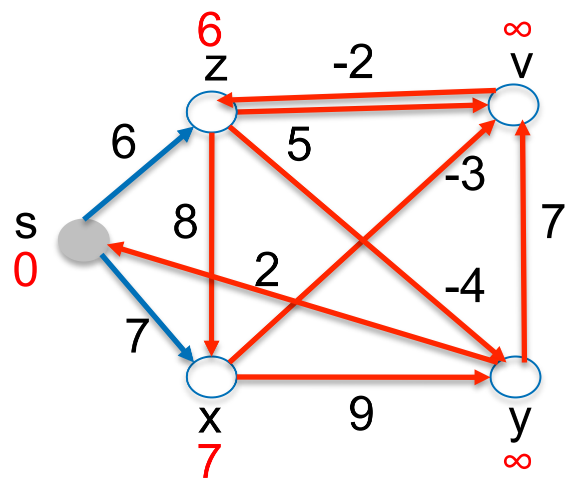
Edge order, line 3: (v,z), (x,v), (x,y), (y,v), (y,s), (z,v), (z,x), (z,y), **(s,x), (s,z)**

runtime: O(|V||E|)

Shortest paths algorithms

Bellman-Ford Algorithmus

```
1: Initialize(G,s)  // for all nodes v≠s: π[v]:=nil; dist[v]:=∞; dist[s]:=0;π[s]:=nil;
2: for i := 1 to |V| - 1 do        →  i=1
3:     for each edge (u,v)∈E do
4:         if dist[v] > dist[u] + f(u,v) then
5:             dist[v] := dist[u] + f(u,v);
6:             π[v] := u;
7:  for each edge (u,v)∈E do
8:      if dist[v] > dist[u] + f(u,v)
9:          return false
10: return true
```



Edge order, line 3: (v,z), (x,v), (x,y), (y,v), (y,s), (z,v), (z,x), (z,y), **(s,x), (s,z)**

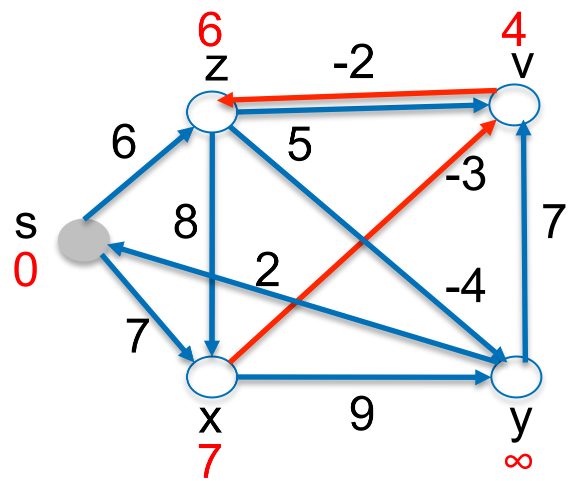
runtime: O(|V||E|)

Shortest paths algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Bellman-Ford Algorithmus

```
1: Initialize(G,s)  // for all nodes v≠s: π[v]:=nil; dist[v]:=∞; dist[s]:=0;π[s]:=nil;
2: for i := 1 to |V| - 1 do          → i=2
3:     for each edge (u,v)∈E do
4:         if dist[v] > dist[u] + f(u,v) then
5:             dist[v] := dist[u] + f(u,v);
6:             π[v] := u;
7:  for each edge (u,v)∈E do
8:      if dist[v] > dist[u] + f(u,v)
9:          return false
10: return true
```



Edge order, line 3: **(v,z)**, **(x,v)**, (x,y), (y,v), (y,s), (z,v), (z,x), (z,y), (s,x), (s,z)

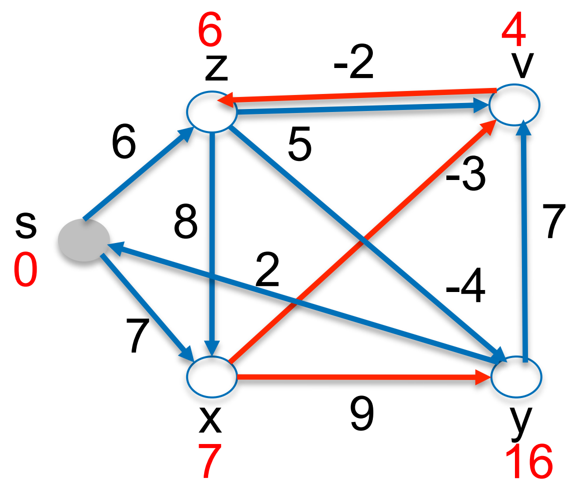
runtime: O(|V||E|)

Shortest paths algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Bellman-Ford Algorithmus

1: Initialize(G,s) // for all nodes v≠s: π[v]:=nil; dist[v]:=∞; dist[s]:=0;π[s]:=nil;
2: **for** i := 1 to |V| - 1 **do** → **i=2**
3: **for** each edge (u,v)∈E **do**
4: **if** dist[v] > dist[u] + f(u,v) **then**
5: dist[v] := dist[u] + f(u,v);
6: π[v] := u;
7: **for** each edge (u,v)∈E **do**
8: **if** dist[v] > dist[u] + f(u,v)
9: **return** false
10: **return** true



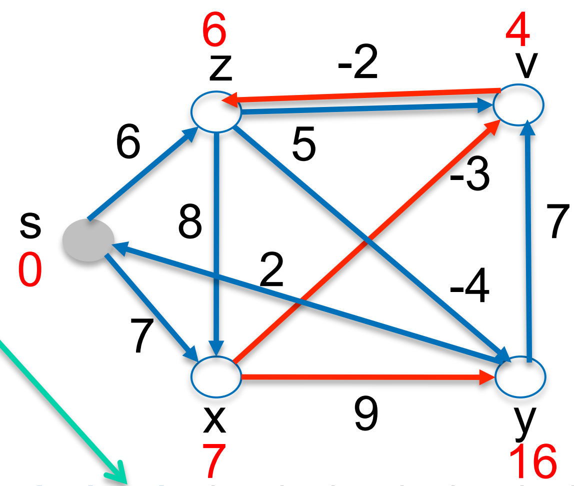
Edge order, line 3: (v,z), (x,v), **(x,y),** (y,v), (y,s), (z,v), (z,x), (z,y), (s,x), (s,z)

runtime: O(|V||E|)

Shortest paths algorithms

Bellman-Ford Algorithmus

```
1: Initialize(G,s)  // for all nodes v≠s: π[v]:=nil; dist[v]:=∞; dist[s]:=0;π[s]:=nil;
2: for i := 1 to |V| - 1 do        →  i=2
3:     for each edge (u,v)∈E do
4:         if dist[v] > dist[u] + f(u,v) then
5:             dist[v] := dist[u] + f(u,v);
6:             π[v] := u;
7:  for each edge (u,v)∈E do
8:      if dist[v] > dist[u] + f(u,v)
9:          return false
10: return true
```



Edge order, line 3 : (v,z), (x,v), **(x,y), (y,v), (y,s),** (z,v), (z,x), (z,y), (s,x), (s,z)

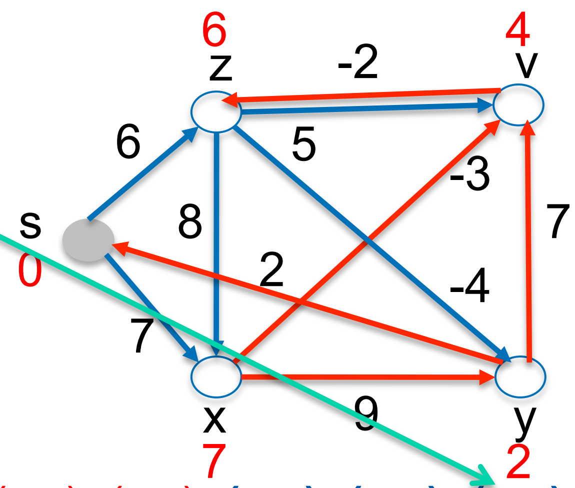
runtime: O(|V||E|)

Shortest paths algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Bellman-Ford Algorithmus

```
1: Initialize(G,s)  // for all nodes v≠s: π[v]:=nil; dist[v]:=∞; dist[s]:=0;π[s]:=nil;
2: for i := 1 to |V| - 1 do        → i=2
3:     for each edge (u,v)∈E do
4:         if dist[v] > dist[u] + f(u,v) then
5:             dist[v] := dist[u] + f(u,v);
6:             π[v] := u;
7:  for each edge (u,v)∈E do
8:      if dist[v] > dist[u] + f(u,v)
9:          return false
10: return true
```



Edge order, line 3: (v,z), (x,v), (x,y), (y,v), (y,s), **(z,v), (z,x), (z,y),** (s,x), (s,z)

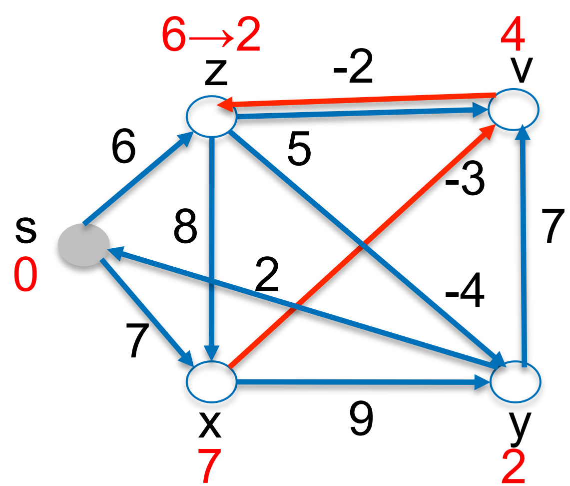
runtime: O(|V||E|)

Shortest paths algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Bellman-Ford Algorithmus

```
1: Initialize(G,s)  // for all nodes v≠s: π[v]:=nil; dist[v]:=∞; dist[s]:=0;π[s]:=nil;
2: for i := 1 to |V| - 1 do        → i=3
3:     for each edge (u,v)∈E do
4:         if dist[v] > dist[u] + f(u,v) then
5:             dist[v] := dist[u] + f(u,v);
6:             π[v] := u;
7:  for each edge (u,v)∈E do
8:      if dist[v] > dist[u] + f(u,v)
9:          return false
10:  return true
```



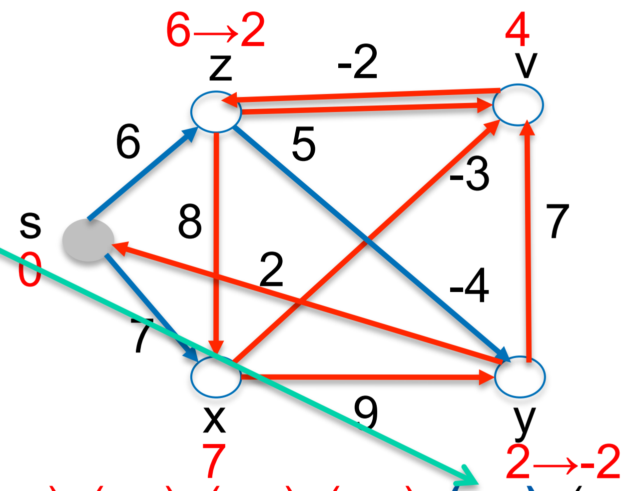
Edge order, line 3: **(v,z), (x,v),** (x,y), (y,v), (y,s), (z,v), (z,x), (z,y), (s,x), (s,z)

runtime: O(|V||E|)

Shortest paths algorithms

Bellman-Ford Algorithmus

```
1: Initialize(G,s)  // for all nodes v≠s: π[v]:=nil; dist[v]:=∞; dist[s]:=0;π[s]:=nil;
2: for i := 1 to |V| - 1 do        →  i=3
3:     for each edge (u,v)∈E do
4:         if dist[v] > dist[u] + f(u,v) then
5:             dist[v] := dist[u] + f(u,v);
6:             π[v] := u;
7:  for each edge (u,v)∈E do
8:      if dist[v] > dist[u] + f(u,v)
9:          return false
10: return true
```



Edge order, line 3: **(v,z), (x,v),** (x,y), (y,v), (y,s), (z,v), (z,x), **(z,y),** (s,x), (s,z)

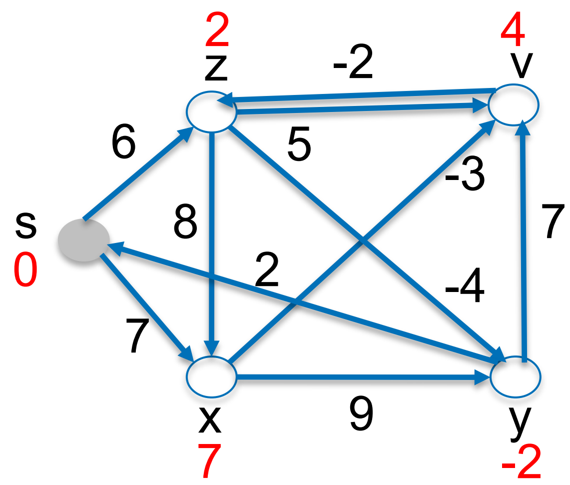
runtime: O(|V||E|)

Shortest paths algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Bellman-Ford Algorithmus

```
1: Initialize(G,s)  // for all nodes v≠s: π[v]:=nil; dist[v]:=∞; dist[s]:=0;π[s]:=nil;
2: for i := 1 to |V| - 1 do   →   i=4,5
3:     for each edge (u,v)∈E do
4:         if dist[v] > dist[u] + f(u,v) then
5:             dist[v] := dist[u] + f(u,v);
6:             π[v] := u;
7:  for each edge (u,v)∈E do
8:      if dist[v] > dist[u] + f(u,v)
9:          return false
10:  return true
```



Edge order, line 3: (v,z), (x,v), (x,y), (y,v), (y,s), (z,v), (z,x), (z,y), (s,x), (s,z)

runtime: O(|V||E|)

Shortest paths algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Bellman-Ford Algorithmus

Lemma BF1: Let G=(V,E) a weighted directed graph with startnode s and weight function f: E→IR. Let G have no negative cycles that can be reached from s. Then: When BF algo is finished, for all nodes it is dist[v]=δ(s,v).

Proof: Let v be a node, reachable from s and let $p=<s=v_0,v_1,...,v=v_k>$ be a shortest path from s to v. p is a simple path and therefore: $k \leq |V|-1$.
We show by induction: $dist[v_i] = \delta(s,v_i)$ after i-th execution of lines 3-6.

ISta: at begin: dist[s] = 0 = δ(s,s). Because no negative cycle exists, dist[s] is never changed anymore.
IH: $dist[v_{i-1}] = \delta(s,v_{i-1})$ after (i-1)-st execution of lines 3-6.
ISte: i-1→i: analogously to proof of Lemma Dijk2

Shortest paths algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

Bellman-Ford Algorithmus

Claim BFCor: Let the Bellman-Ford Algorithm run on a weighted directed graph G with start node s. If G contains a from s reachable negative cycle, it will return false. Otherwise it will return true and for all nodes we have dist[v]=δ(s,v).

Proof: If there is no negative cycle reachable from s, for each node v will be valid: dist[v]=δ(s,v) because of Lemma BF1. If v is not reachable from s, dist[v] obviously stays ∞. Because after finishing, for all nodes is valid:

dist[v] = δ(s,v) ≤ δ(s,u) + f(u,v) = dist[u] + f(u,v),
line 9 was not executed.

always valid for shortest paths

...
7: **for** each edge (u,v)∈E **do**
8:     **if** dist[v] > dist[u] + f(u,v)
9:         **return** false
10: **return** true

TECHNISCHE UNIVERSITÄT DARMSTADT

Bellman-Ford Algorithmus

Claim BFKor: (cont.)
Now, let us assume that G contains a negative cycle, reachable from s: c=<$v_0,v_1,\ldots,v_k$>. Let $v_0=v_k$. Then:

$$\sum_{i=1}^{k} f(v_{i-1},v_i) < 0$$

Assumption: Bellman-Ford returns true, i.e., for all i=1,2,...,k: $dist[v_i] \le dist[v_{i-1}] + f(v_{i-1},v_i)$. Then, this also implies for the sums:

$$\sum_{i=1}^{k} dist[v_i] \le \sum_{i=1}^{k} dist[v_{i-1}] + \sum_{i=1}^{k} f(v_{i-1},v_i)$$

Because c is a cycle, each summand occurs in the first two sums. This implies:

$$\sum_{i=1}^{k} dist[v_i] = \sum_{i=1}^{k} dist[v_{i-1}] \text{ and thus } 0 \le \sum_{i=1}^{k} f(v_{i-1},v_i)$$

Contradiction to

Minimum Spanning Trees

TECHNISCHE UNIVERSITÄT DARMSTADT

- Definition MiSpa1

  – **Weighted undirected graph** (G,f): undirected graph G=(V,E) with weight function f: E → IR.

  – If H=(U,F), U⊆V, F⊆E, is a subgraph of G, then the weight f(H) of H is defined as

$$w(H) = \sum_{e \in F} w(e)$$

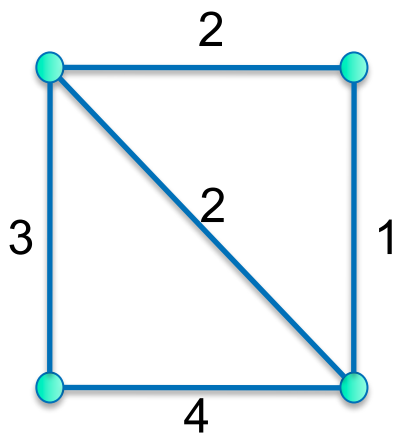
  – A subgraph H of an undirected graph G is called **spanning tree** of G if H is a tree containing all nodes if G.

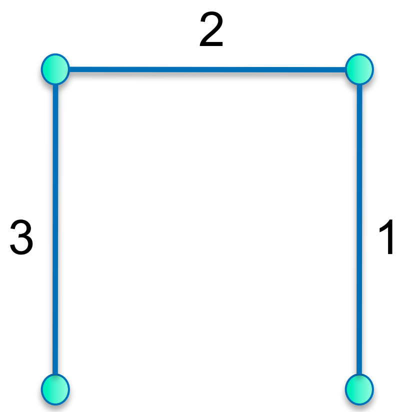
  – A spanning tree S of a weighted undirected graph is is called minimum spanning tree if S has smallest weight under all spanning trees of G.

Minimum Spanning Trees

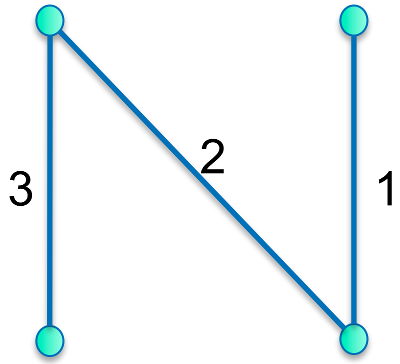
TECHNISCHE UNIVERSITÄT DARMSTADT

Graph G

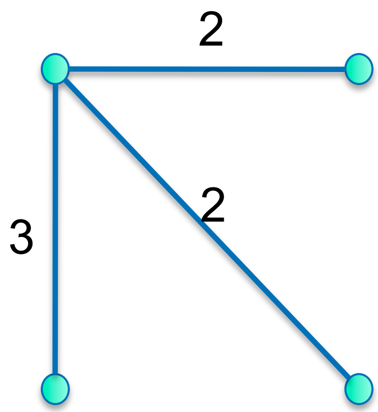




Spanning tree for G, minimum



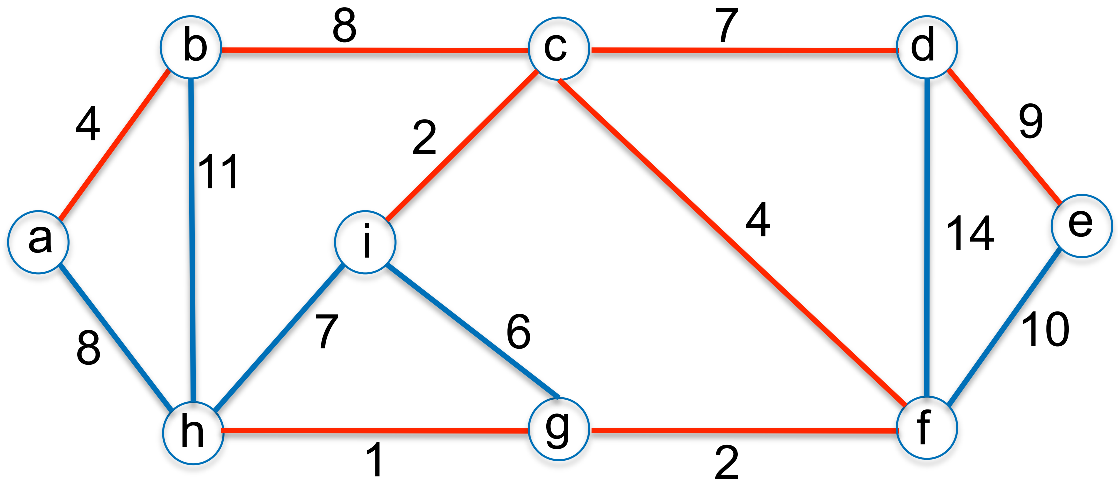
Spanning tree for G, minimum



Spanning tree for G, not minimum

Minimum Spanning Trees

TECHNISCHE
UNIVERSITÄT
DARMSTADT



Minimum Spanning Trees

- Aim: Let a weighted undirected graph (G,f) with G=(V,E) be given. Efficiently find a minimum spanning tree of (G,f).

- General idea: Iteratively expand an edge-set A⊆E to a mimimum spanning tree:
  - Def. MiSpa2: (u,v) is called **A-safe**, iff A ∪ {(u,v)} can be expanded to a minimum spanning tree.

  - At beginning: A={}
  - Step by step replace A with A ∪ {(u,v)}, where (u,v) is an A-safe edge
  - Repeat 1. and 2. until |A| = |V| - 1

Minimum Spanning Trees

TECHNISCHE UNIVERSITÄT DARMSTADT

- Generic MST-Algorithm (MST = Minimum Spanning Tree)

Generic-MST(G,f)

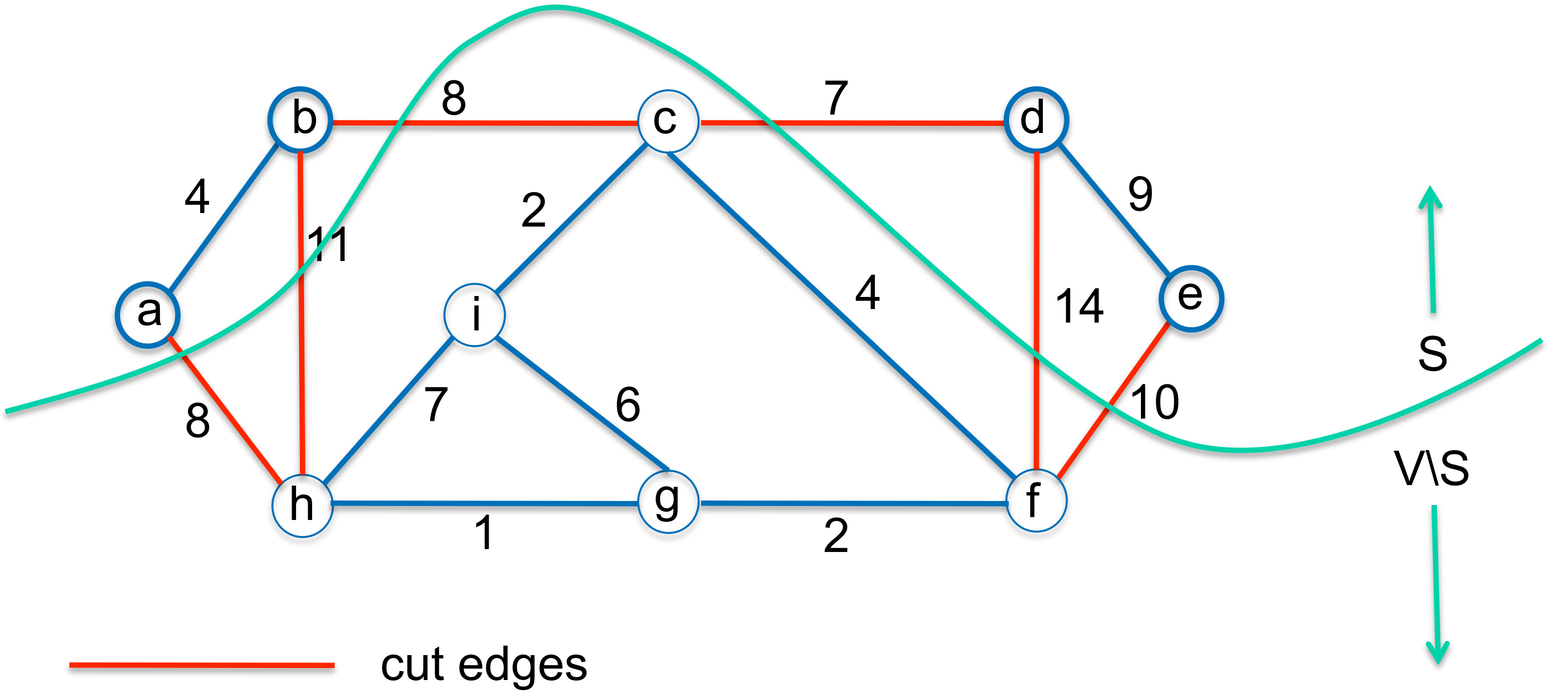
1: A := {}
2: **while** A is not yet MST
3: **for** each edge (u,v)∈E **do**
4: find A-safe edge (u,v)
5: A := A ∪ {(u,v)}
6: return A;

Minimum Spanning Trees

- Def. MiSpa3:
  - A **cut** (C,V \ C) in a graph G=(V,E) is a partition of the node set V of G.
  - An edge of G **crosses** a cut (C,V \ C), when one node of the edge is in C, the other one in V \ C.
  - A cut (C, V \ C) **respects** a subset A⊆E, iff no element of A crosses the cut.
  - An edge, crossing the cut (C, V \ C), is called **a light edge**, iff it is a cut crossing edge with minimum weight.

Minimum Spanning Trees

Cut:



Minimum Spanning Trees

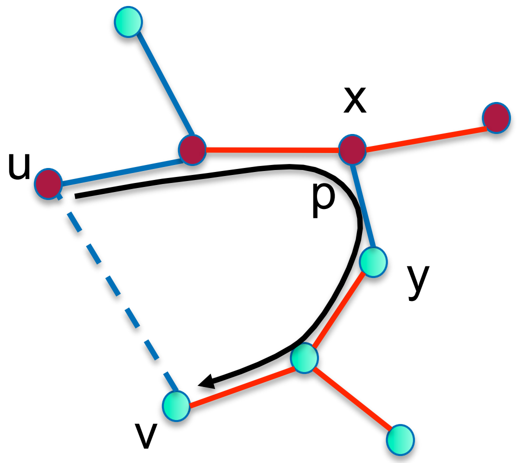
- Claim MiSpa1:
  Let (G,f) be a weighted undirected and connected graph. Let us assume that there is a minimum spanning tree T in G which contains the edge set A⊆E. Let (S, V \ S) be a cut respecting A

  [...respecting: no element of A crosses the cut]

  and let (u,v) be a light edge, crossing (S, V \ S).

  [... light: (C, V \ C) crossing edge with minimum weight (minimum over all edge crossing edges)]

  Then: (u,v) is an A-safe edge.



- S V\S
- A
- (u,v) light edge from S to V \ S
  Only edges of T are in the picture
- spanning tree T‘, containing (u,v), is constructed as follows:
  remove (x,y) and add (u,v) to A

Minimum Spanning Trees

- Claim MiSpa2 (ends in Kruskal-algorithm):
  Let (G,f) be a weighted undirected graph. Let a minimum spanning tree exist in G that contains the edge set $A \subseteq E$. If (u,v) is a light edge of minimum weight which connects a tree B of the forest G_A =(V,A) with another tree of G_A, then (u,v) is A-safe.

  Proof:
  - The cut (B, V \ B) respects A (why?)
    - B is a connecting component of G_A = (V,A)
    - therefore: B is a tree without edges in G_A going from B to V\B
    - thus the cut respects A (Def)
  - Therefore, (u,v) is a light edge for the cut (why?)
    - (u,v) connects two trees, which are part of a minimum spanning tree.
    - two subtrees cannot be connected cheaper.
  - Therefore (u,v) is A-safe with claim MiSpa1

Minimum Spanning Trees

- Prim‘s Algorithm -- Idea
  - At each point of time when the algorithm is running, the graph $G_A = (V,A)$ consists of a tree T_A and a set of isolated nodes I_A
  - An edge of minimum weight that connects a node of I_A with T_A is added to A
  - The nodes in I_A are organized in a Min-Heap. The key key[v] of a node $v \in I_A$ is given by the minimum weight of an edge which connects v with T_A.

Minimum Spanning Trees

TECHNISCHE UNIVERSITÄT DARMSTADT

- Prim‘s Algorithm

Prim-MST(G,f,w)

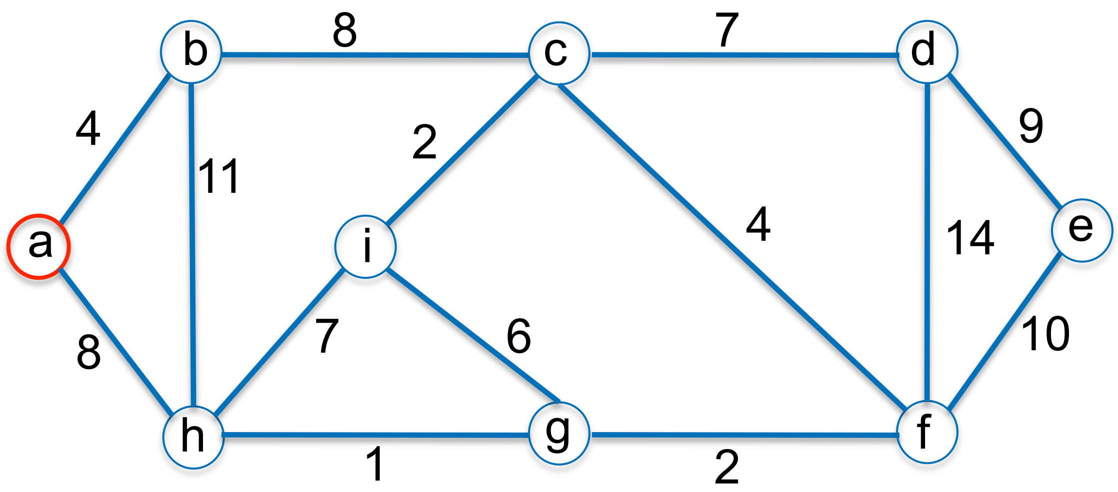
1: **for** all v ∈ V **do**
2:     key[v] := ∞
3:     π(v) := nil
4: key[w] := 0
5: Q:= Build-Heap(V)
6: **while** Q ≠ {} **do**
7:     u :=Extract-Min(Q)
8:     **for** all v ∈ Adjazenzliste[u] **do**
9:         **if** v ∈ Q **and** f(u,v) < key[v] **then**
10:             π[v] := u
11:             key[v] := f(u,v)
12:             Decrease-Key(Q,v,key[v])

runtime:

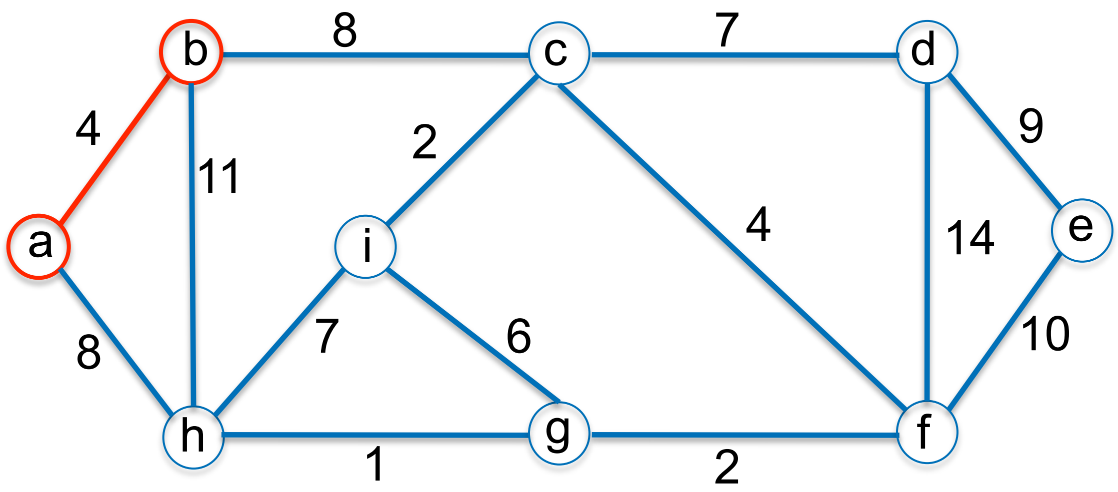
- |V|-many while-loop-executions
- line7: O(log |V|)
- lines 9-12: O(2 · |E|) executions
- line 12: O(log |V|)

- totally: O(|E| · log |V|)
- with Fibonacci-heaps
  O(|V| log |V| + |E|) is possible

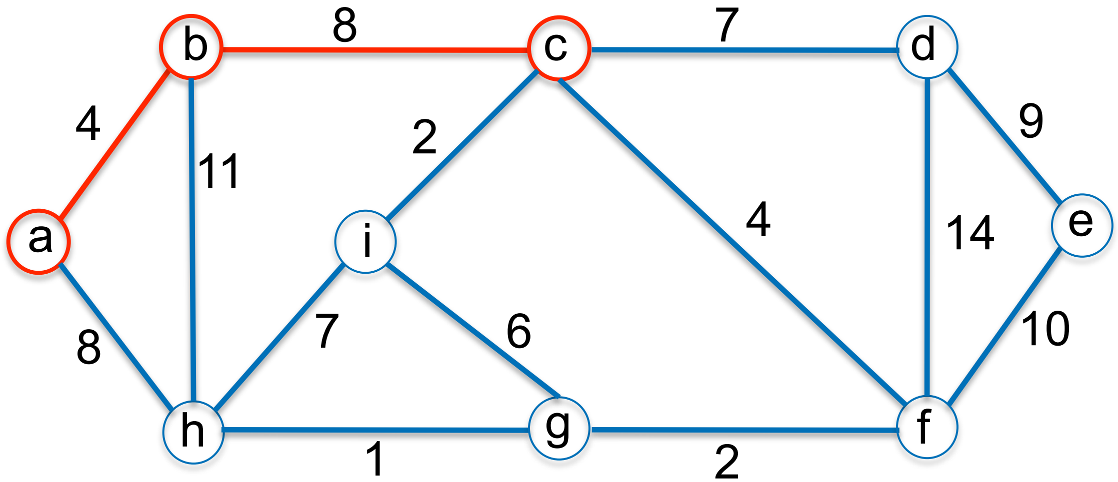
Minimum Spanning Trees



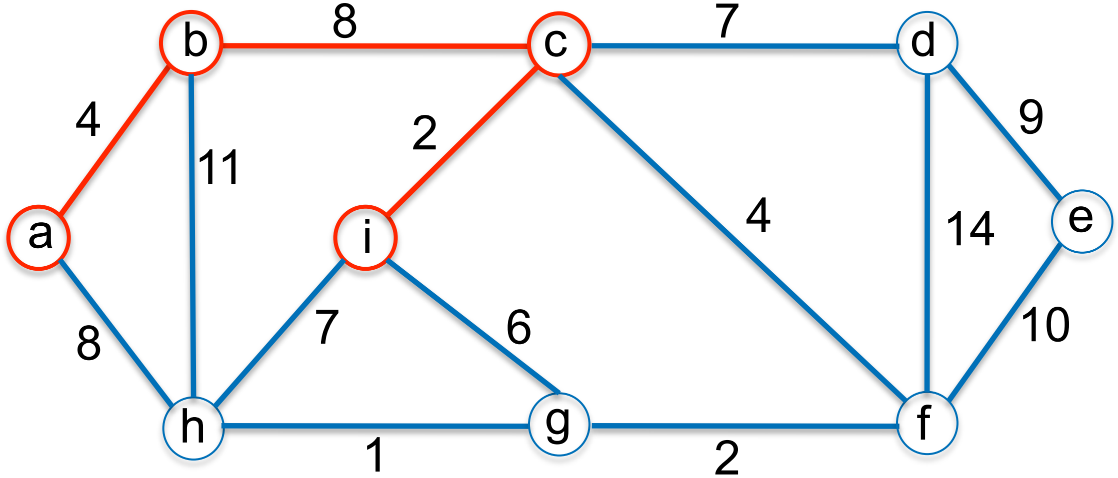
Minimum Spanning Trees



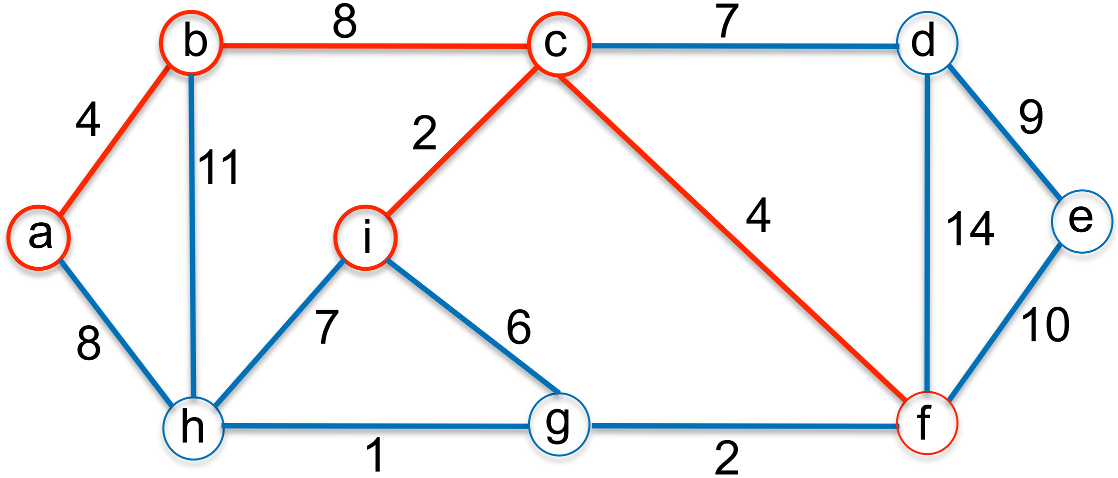
Minimum Spanning Trees



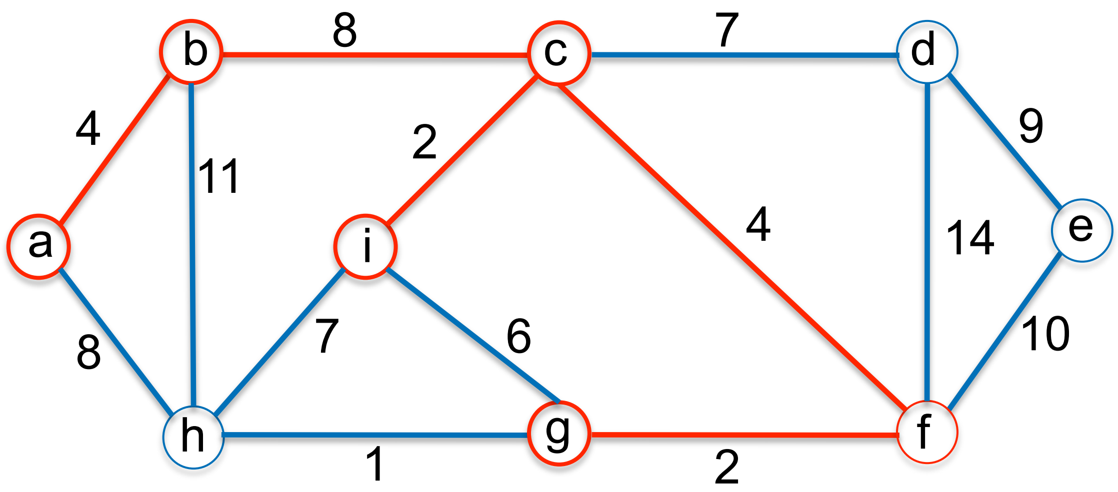
Minimum Spanning Trees



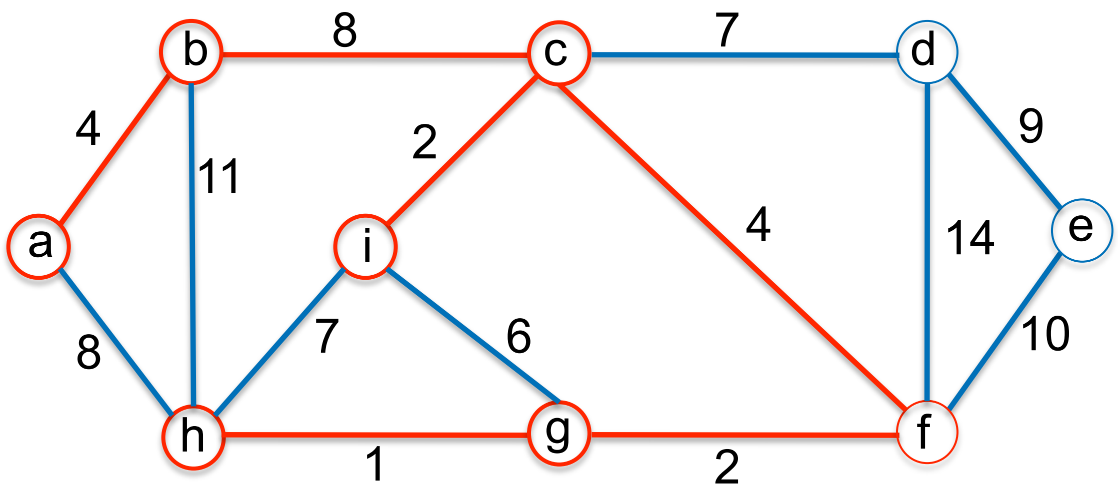
Minimum Spanning Trees



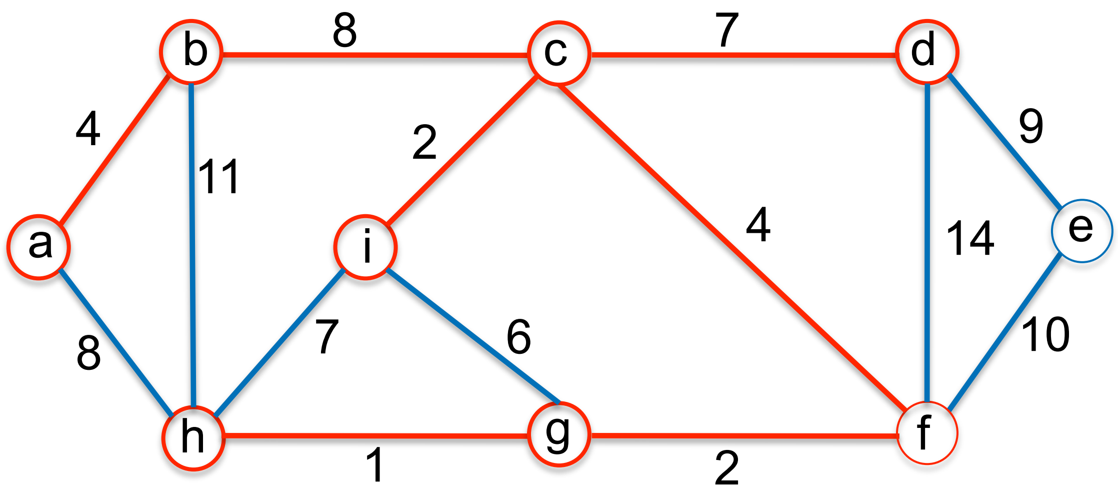
Minimum Spanning Trees



Minimum Spanning Trees



Minimum Spanning Trees



Minimum Spanning Trees