Foundations of Computer Science

ECE NTUA

3rd Section:
Graph and Network Algorithms

Slides:
Stathis Zachos, Aris Pagourtzis, Dimitris Fotakis

Acknowledgements: part of the slides comes from Stavros Nikolopoulos (University of Ioannina)

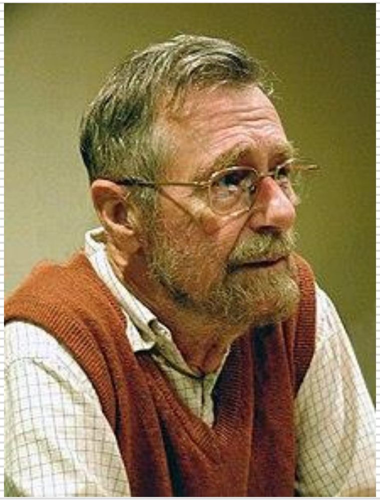
School of Electrical and Computer Engineering
National Technical University of Athens

Shortest Paths problem

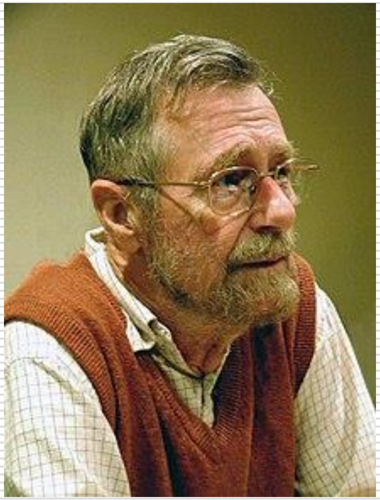
- In a weighted graph, we search for the **shortest paths** from a starting node s to all others
- Weights are non-negative numbers (e.g. distances)
- Applications: shorter / cheaper / faster journeys and many more.
- **Algorithmic ideas:**
  - What if all weights are the same? (e.g. = 1)
  - Modify BFS for integer weights?

Dijkstra Algorithm (concept)

- **Dijkstra (intuition)**
  - Starting from the origin node s we can easily find the node u that is closest to s
  - The second nearest (to s) will be a neighbour of s or u (why?)



Dijkstra Algorithm (concept)



- **Dijkstra (implementation)**
  - We keep temporary distance labels from s
  - For node u with minimum distance label, the path of least weight has been found!
  - Label of u becomes permanent
  - Update labels of u's neighbours
  - Repeat the above 3 steps on the neighbours of the permanent nodes

Dijkstra Algorithm

S := {s}; D(s) := 0 ; P(s) := NIL
for each v: (s,v) ∈ E **do** D(v) := cost(s,v); P(v) := s
for each v: (s,v) ∉ E **do** D(v) := ∞; P(v) := NIL

repeat n times
    select u from V\S with minimum D(u)
    S := S U {u}
    for all v in V\S: (u,v) ∈ E **do**
        if D(u) + cost(u,v) < D(v) **then**
            D(v) := D(u) + cost(u,v)
            P(v) := u

Complexity $O(|V|^2)$: for every repeat $O(|V|)$ to find minimum, $O(|V|)$ to update distances

Dijkstra: example

S := {s}; D(s) := 0 ; P(s) := NIL

for each v: (s,v) ∈ E **do** D(v) := cost(s,v); P(v) := s

for each v: (s,v) ∉ E **do** D(v) := ∞; P(v) := NIL

repeat n times

    select u from V\S with minimum D(u)

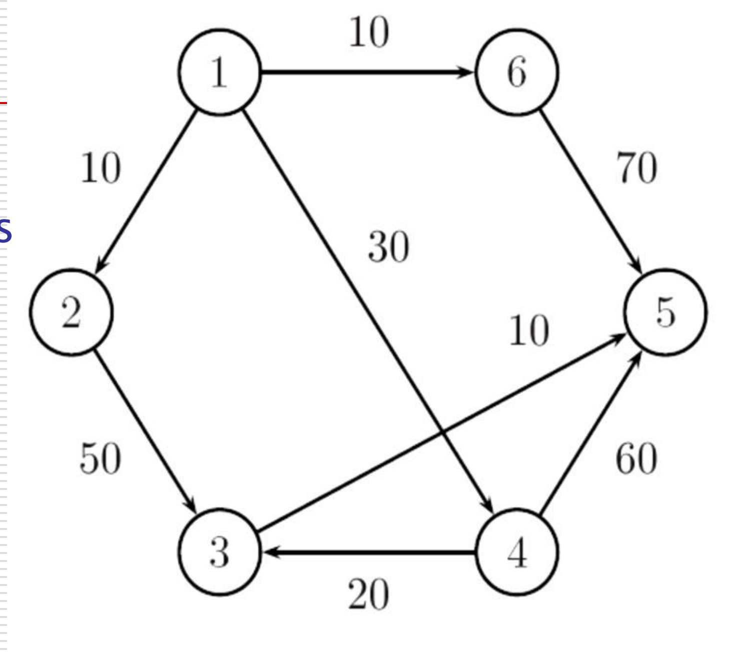
    S := S U {u}

    for all v in V\S: (u,v) ∈ E **do**

        if D(u) + cost(u,v) < D(v) **then**

            D(v) := D(u) + cost(u,v)

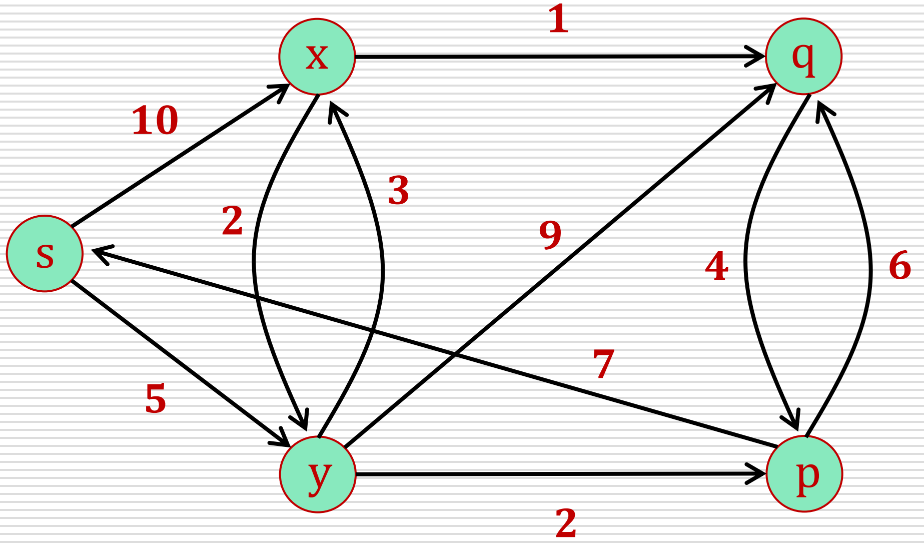
            P(v) := u



	S	w	D					P				
			2	3	4	5	6	2	3	4	5	6
-	{1}	-	10	∞	30	∞	10	1		1		1
2	{1,2}	2		60	30	∞	10		2			
3	{1,2,6}	6		60	30	80					6	
4	{1,2,6,4}	4		50		80			4			
5	{1,2,6,4,3}	3				60					3	
6	{1,2,6,4,3,5}	5										

Dijkstra: 2nd example

Initialize (G,s)

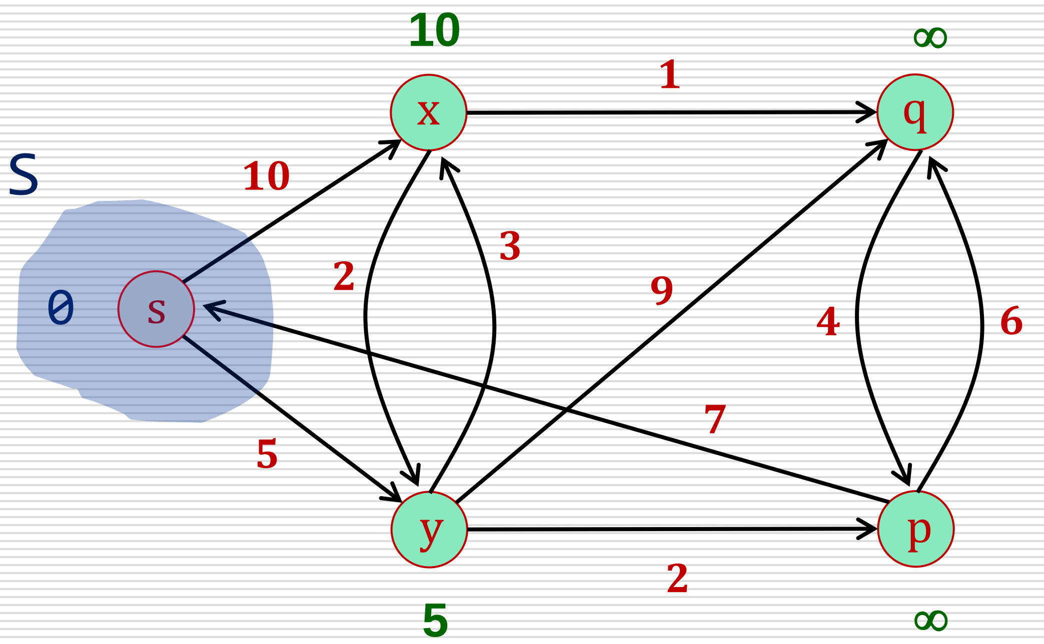


d(s)=0 d(x)= ∞ d(y)= ∞ d(p)= ∞ d(q)= ∞

prev(s)=NIL prev(x)= NIL prev(y)= NIL prev(p)=NIL prev(q)=NIL

Dijkstra: 2nd example

V\S = {y,x,q,p}



d(s)=0 prev(s)=NIL

d(x)=10 prev(x)=s

d(y)=5 prev(y)=s

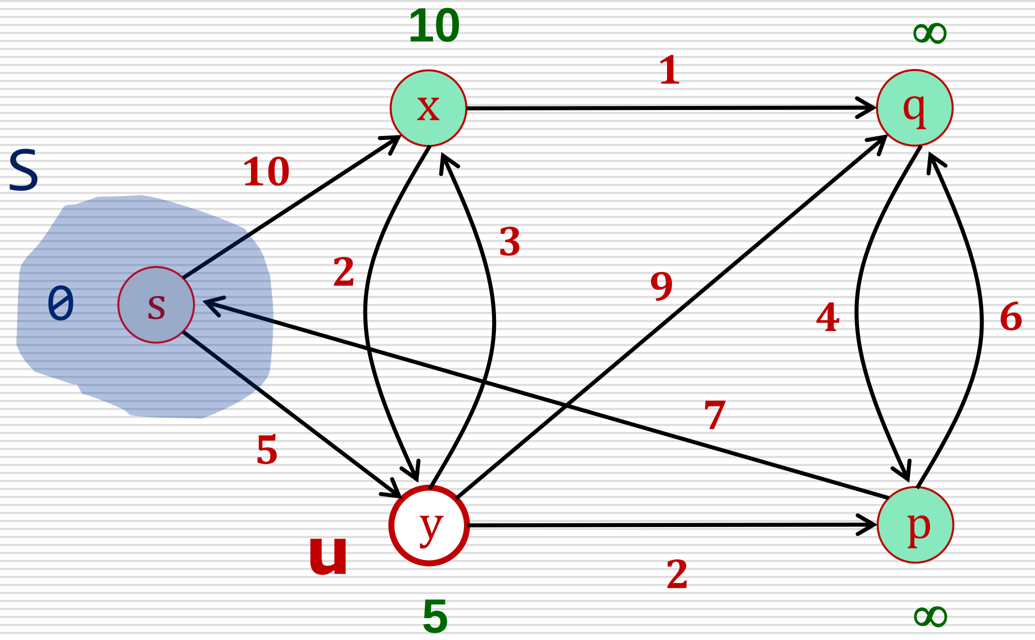
d(p)=∞ prev(p)=NIL

d(q)=∞ prev(q)=NIL

Dijkstra: 2nd example

V\S = {y,x,q,p}

u ← min$_{D(u)}$(V\S)
S ← S ∪ {u}



d(s)=0 prev(s)=NIL

d(x)=10 prev(x)=s

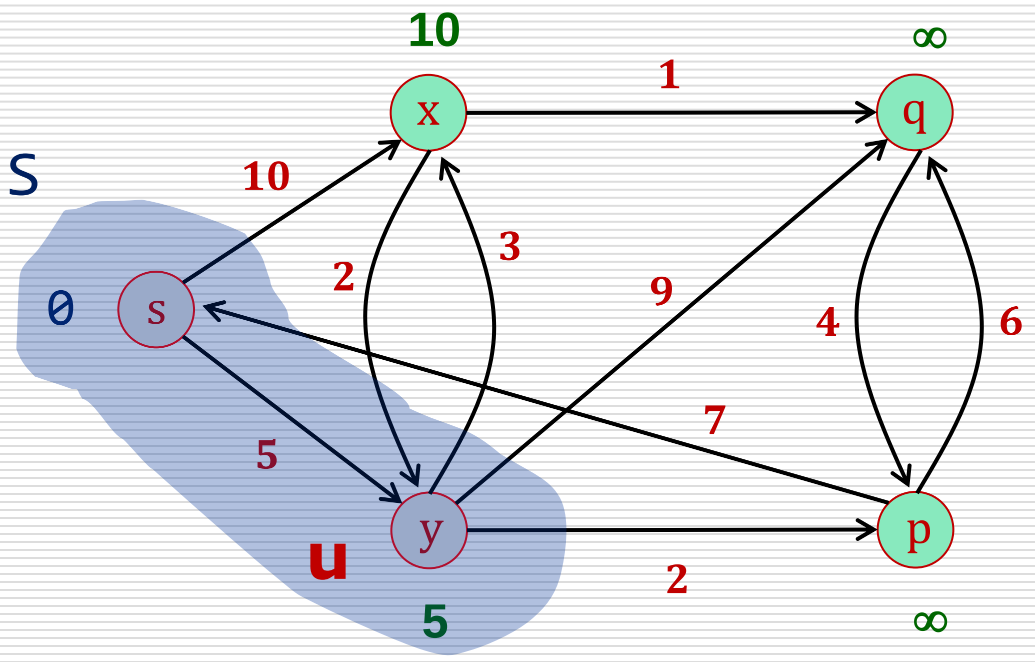
d(y)=5 prev(y)=s

d(p)=∞ prev(p)=NIL

d(q)=∞ prev(q)=NIL

Dijkstra: 2nd example

V\S = {x,q,p}

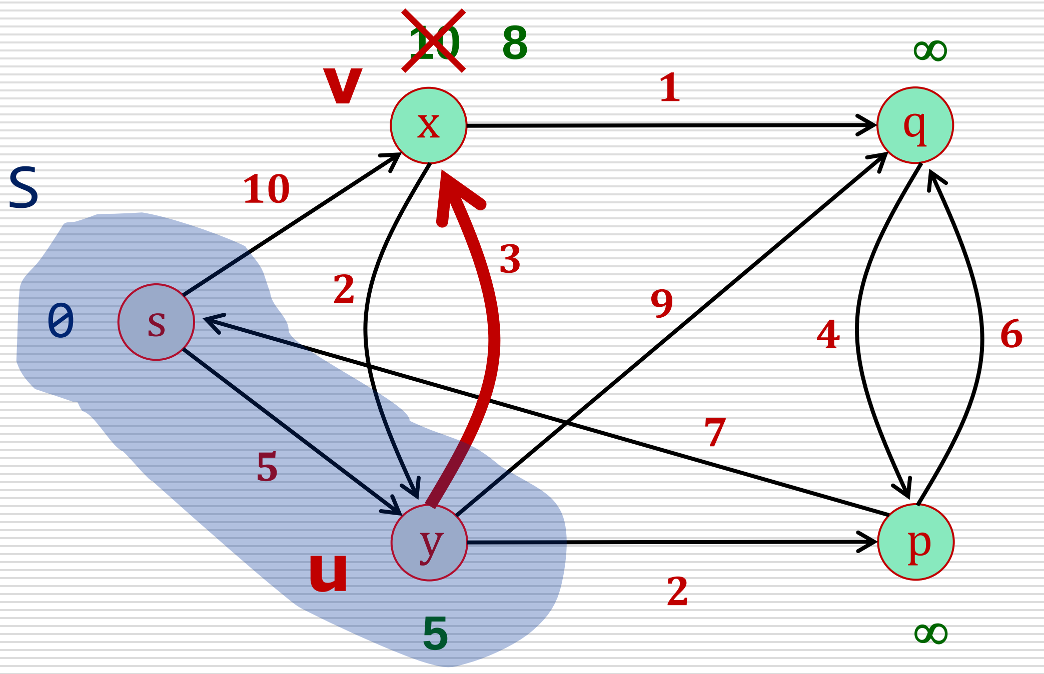


d(s)=0 d(x)=10 d(y)=5 d(p)=∞ d(q)=∞

prev(s)=NIL prev(x)=s prev(y)=s prev(p)=NIL prev(q)=NIL

Dijkstra: 2nd example

V\S = {x,q,p}



d(s)=0 prev(s)=NIL

d(x)=8 prev(x)=y

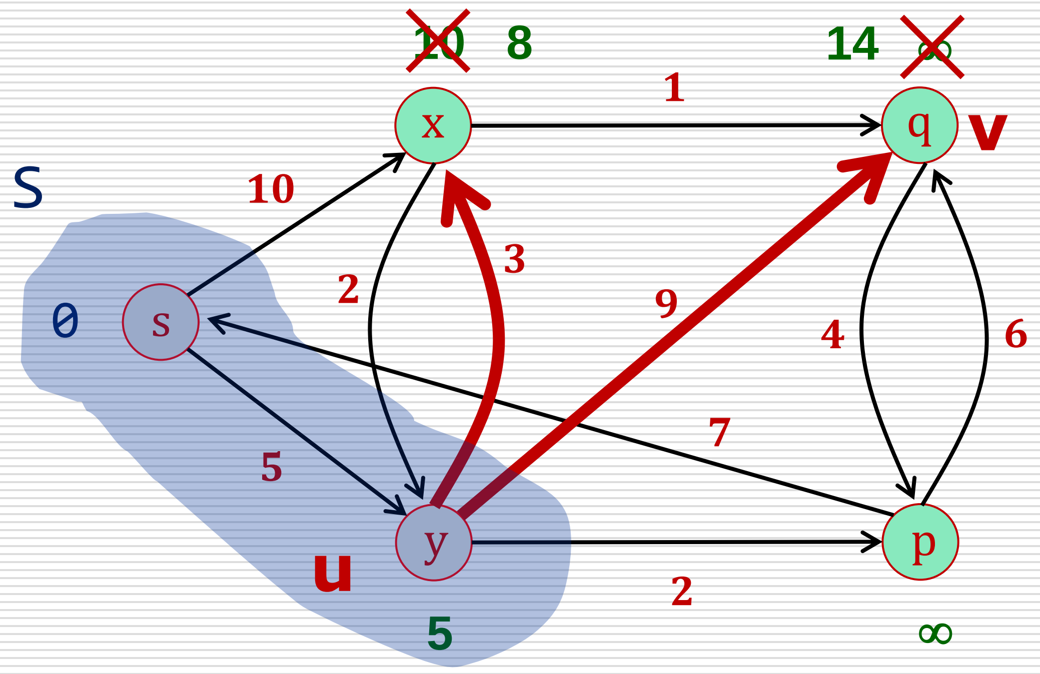
d(y)=5 prev(y)=s

d(p)=∞ prev(p)=NIL

d(q)=∞ prev(q)=NIL

Dijkstra: 2nd example

V\S = {x,q,p}

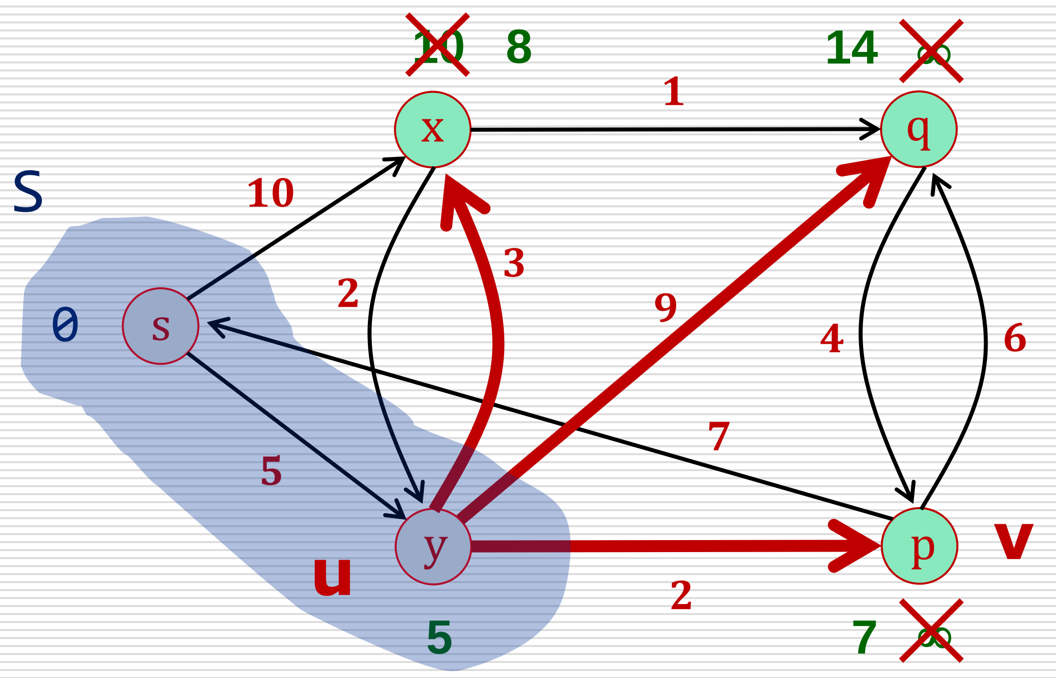


d(s)=0 d(x)=8 d(y)=5 d(p)=∞ d(q)=14

prev(s)=NIL prev(x)=y prev(y)=s prev(p)=NIL prev(q)=y

Dijkstra: 2nd example

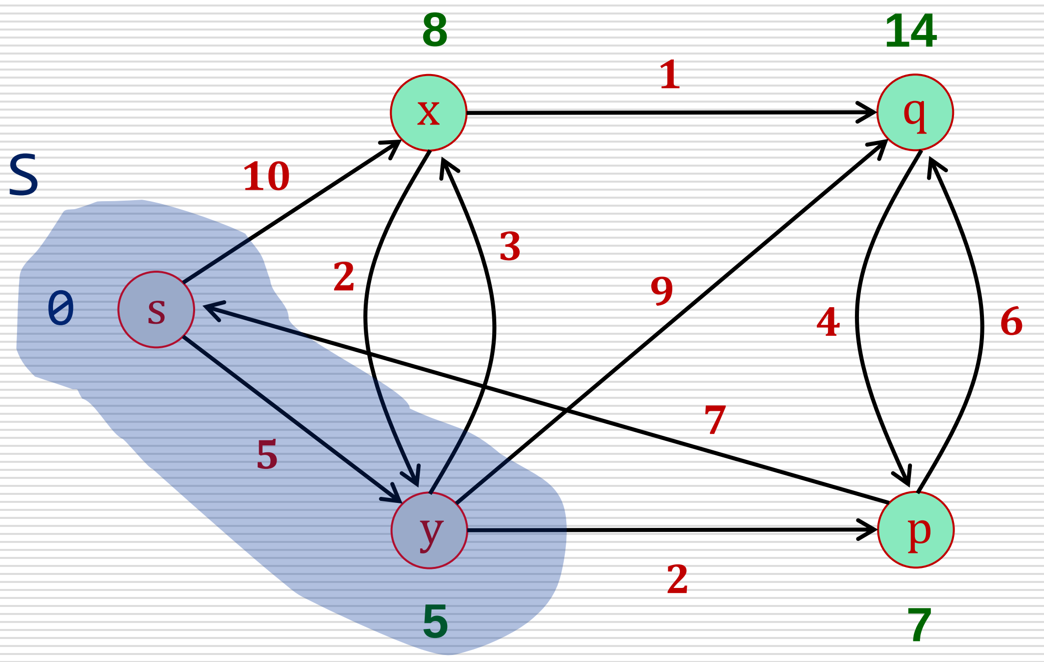
V\S = {p,x,q}



d(s)=0	d(x)=8	d(y)=5	d(p)=7	d(q)=13
prev(s)=NIL	prev(x)=y	prev(y)=s	prev(p)=y	prev(q)=y

Dijkstra: 2nd example

V\S = {p,x,q}



d(s)=0 prev(s)=NIL

d(x)=8 prev(x)=y

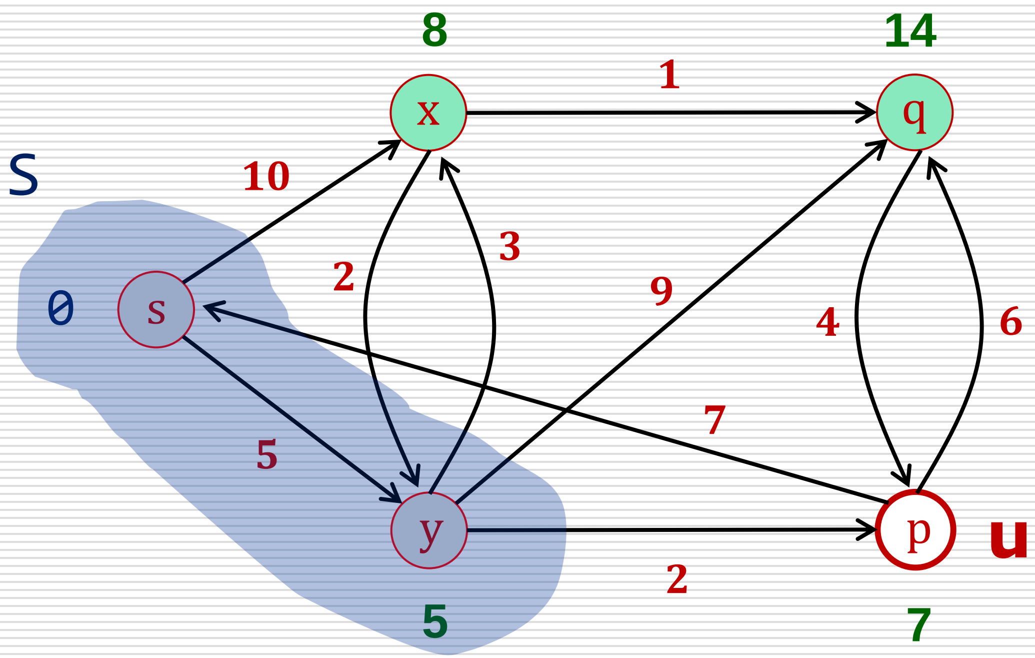
d(y)=5 prev(y)=s

d(p)=7 prev(p)=y

d(q)=13 prev(q)=y

Dijkstra: 2nd example

V\S = {p,x,q}

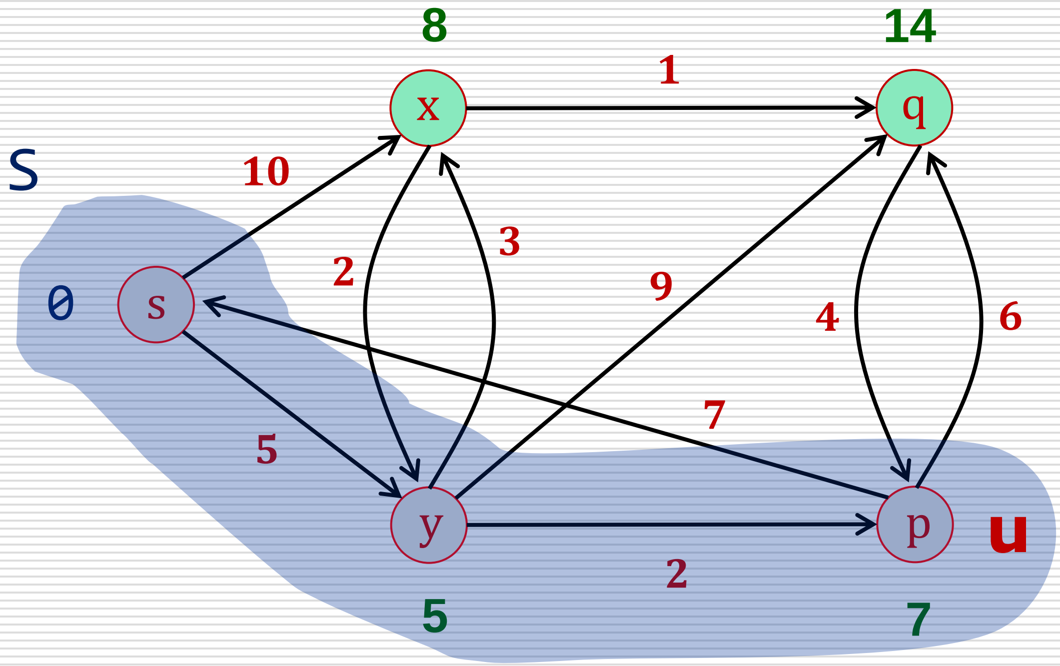


u ← $\min_{D(u)}$(V\S)
S ← S ∪ {u}

d(s)=0	d(x)=8	d(y)=5	d(p)=7	d(q)=13
prev(s)=NIL	prev(x)=y	prev(y)=s	prev(p)=y	prev(q)=y

Dijkstra: 2nd example

V\S = {x,q}

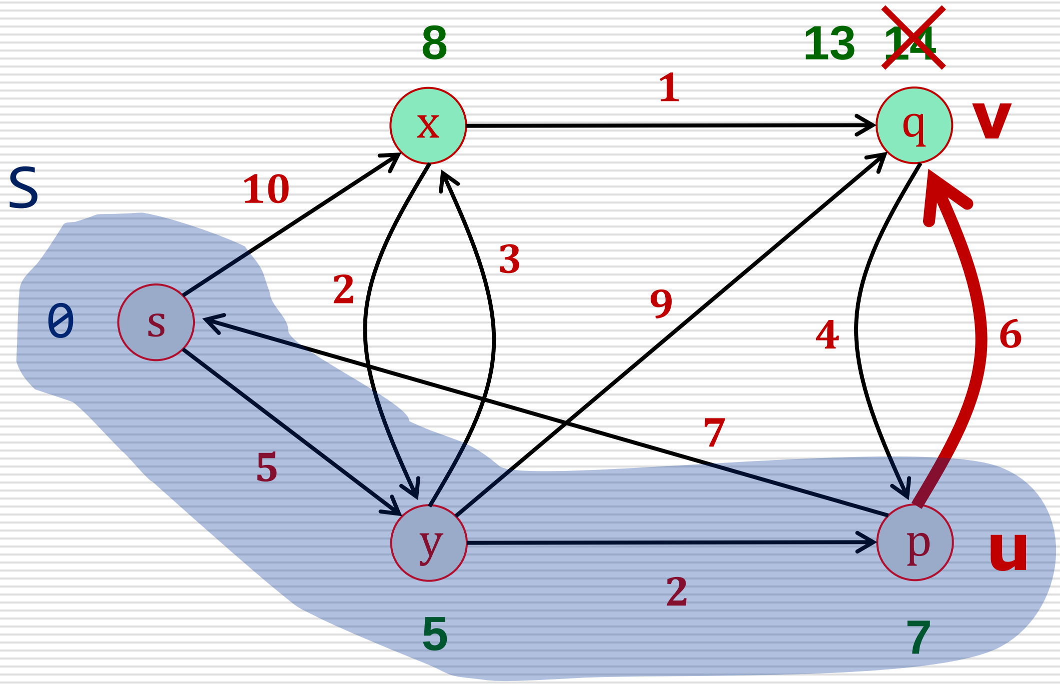


d(s)=0 d(x)=8 d(y)=5 d(p)=7 d(q)=13

prev(s)=NIL prev(x)=y prev(y)=s prev(p)=y prev(q)=y

Dijkstra: 2nd example

V\S = {x,q}



d(s)=0 prev(s)=NIL

d(x)=8 prev(x)=y

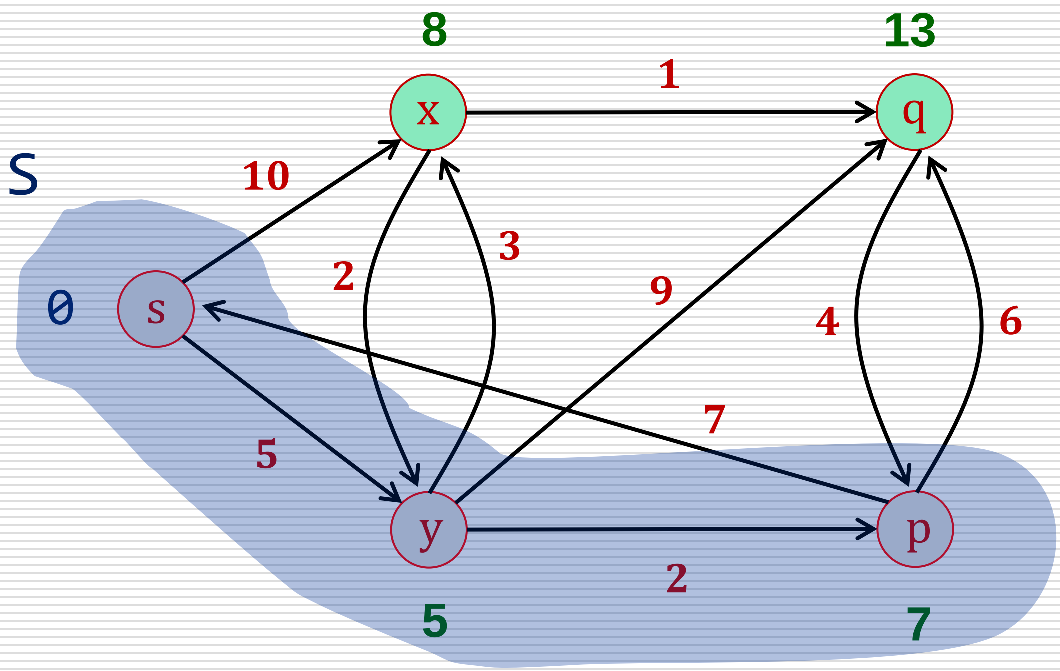
d(y)=5 prev(y)=s

d(p)=7 prev(p)=y

d(q)=13 prev(q)=p

Dijkstra: 2^{nd} example

V\S = {x,q}

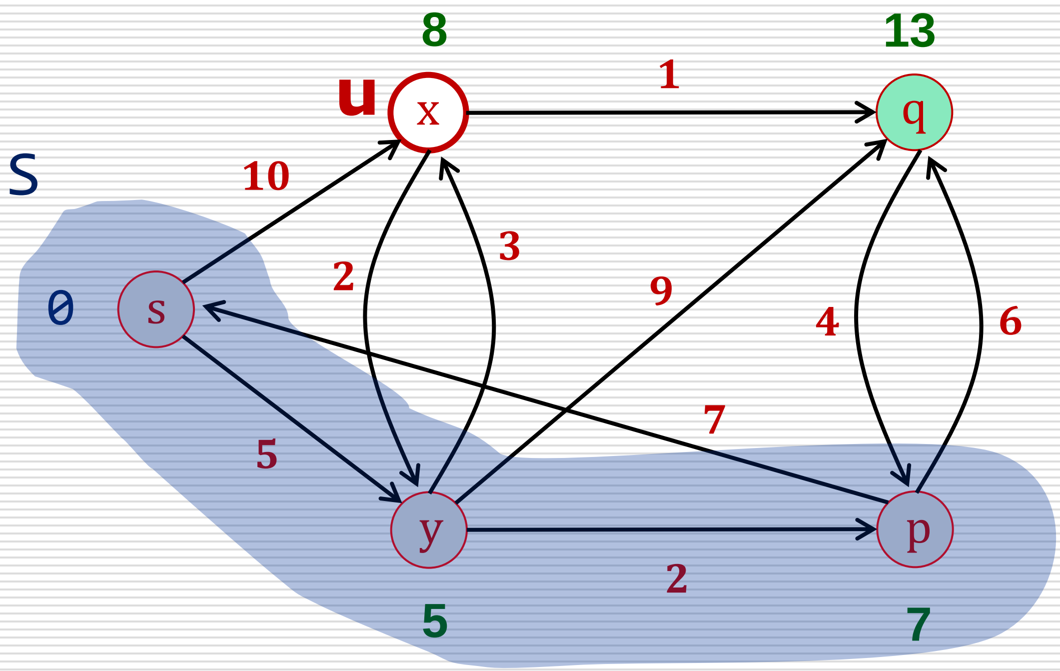


d(s)=0 d(x)=8 d(y)=5 d(p)=7 d(q)=13

prev(s)=NIL prev(x)=y prev(y)=s prev(p)=y prev(q)=p

Dijkstra: 2nd example

V\S = {x,q}



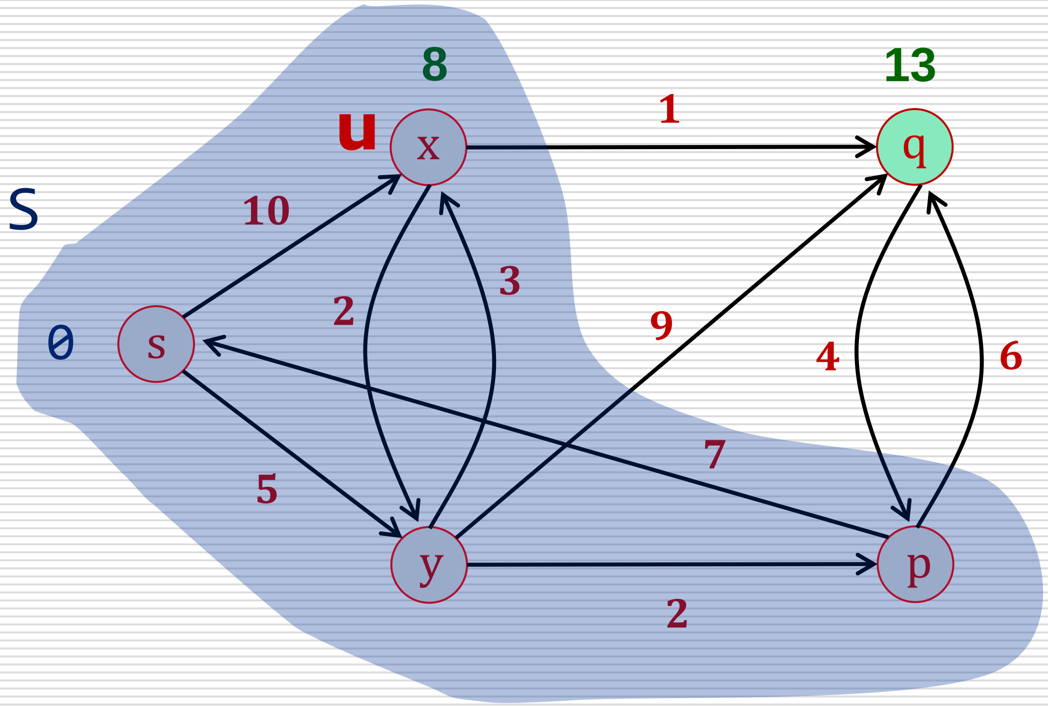
u ← min$_{D(u)}$(V\S)
S ← S ∪ {u}

d(s)=0 d(x)=8 d(y)=5 d(p)=7 d(q)=13

prev(s)=NIL prev(x)=y prev(y)=s prev(p)=y prev(q)=p

Dijkstra: 2nd example

V\S = {q}

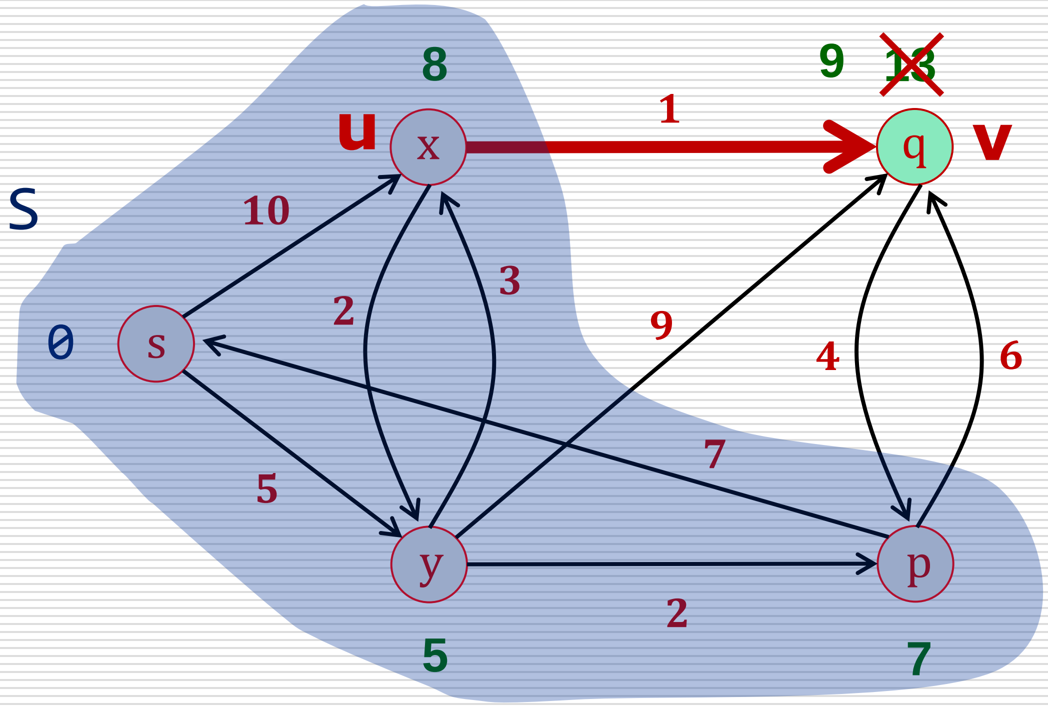


d(s)=0 d(x)=8 d(y)=5 d(p)=7 d(q)=13

prev(s)=NIL prev(x)=y prev(y)=s prev(p)=y prev(q)=p

Dijkstra: 2nd example

V\S = {q}

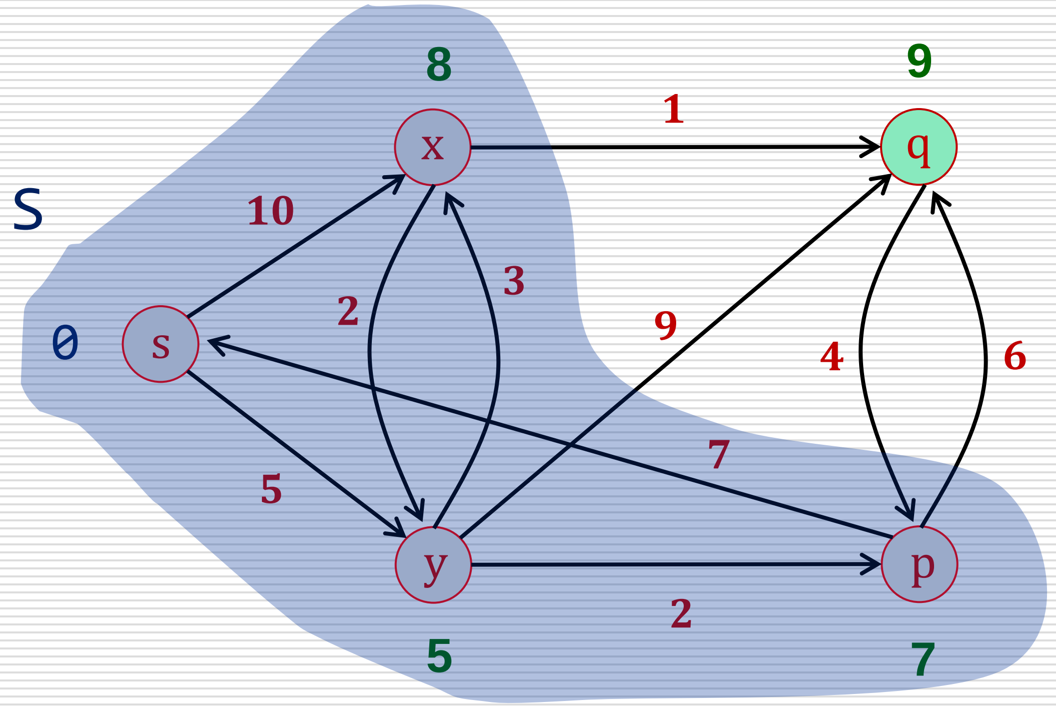


d(s)=0 d(x)=8 d(y)=5 d(p)=7 d(q)=9

prev(s)=NIL prev(x)=y prev(y)=s prev(p)=y prev(q)=x

Dijkstra: 2nd example

V\S = {q}

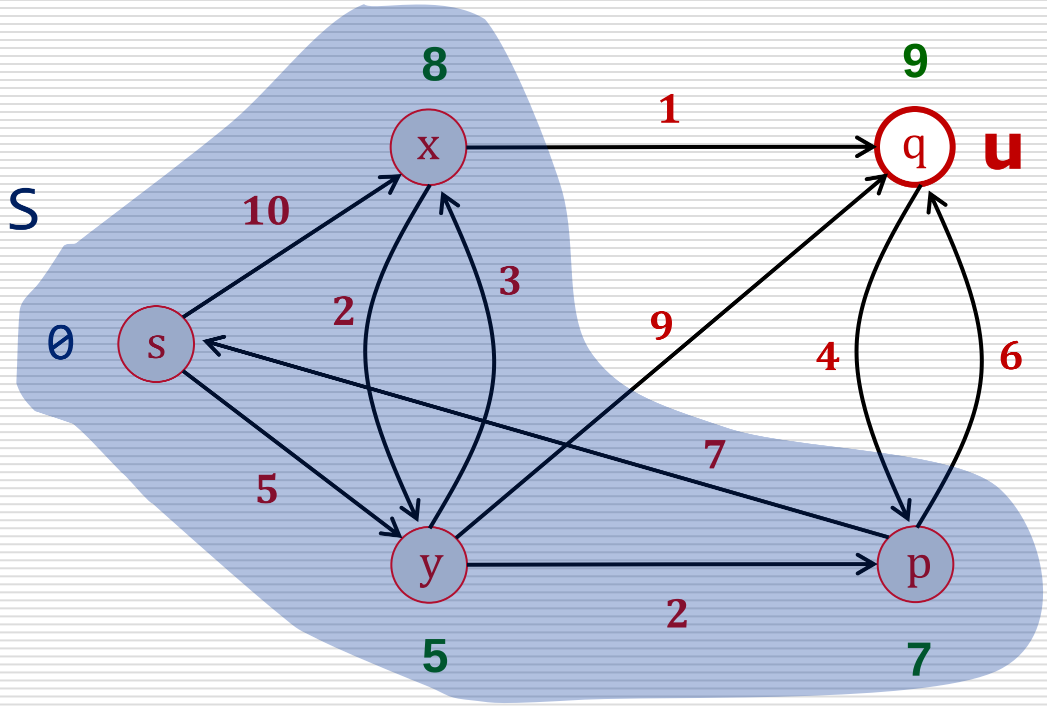


d(s)=0 d(x)=8 d(y)=5 d(p)=7 d(q)=9

prev(s)=NIL prev(x)=y prev(y)=s prev(p)=y prev(q)=x

Dijkstra: 2^{nd} example

$V \backslash S = \{q\}$



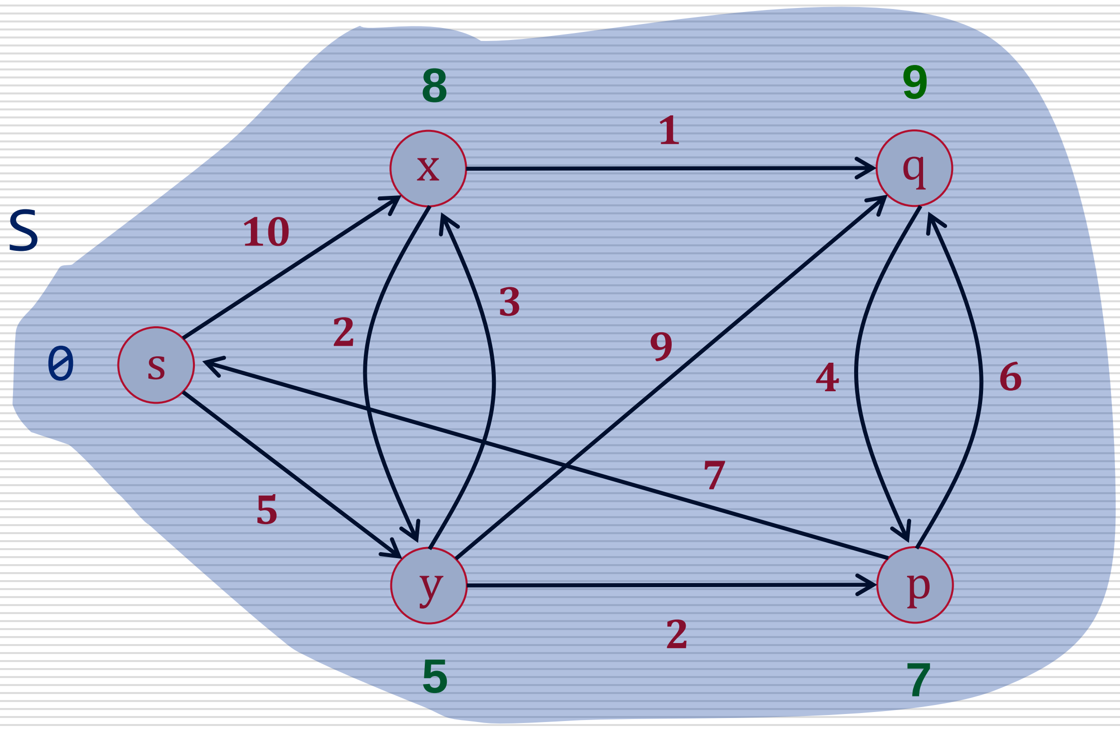
$u \leftarrow \min_{D(u)}(V \backslash S)$
$S \leftarrow S \cup \{u\}$

d(s)=0 d(x)=8 d(y)=5 d(p)=7 d(q)=9

prev(s)=NIL prev(x)=y prev(y)=s prev(p)=y prev(q)=x

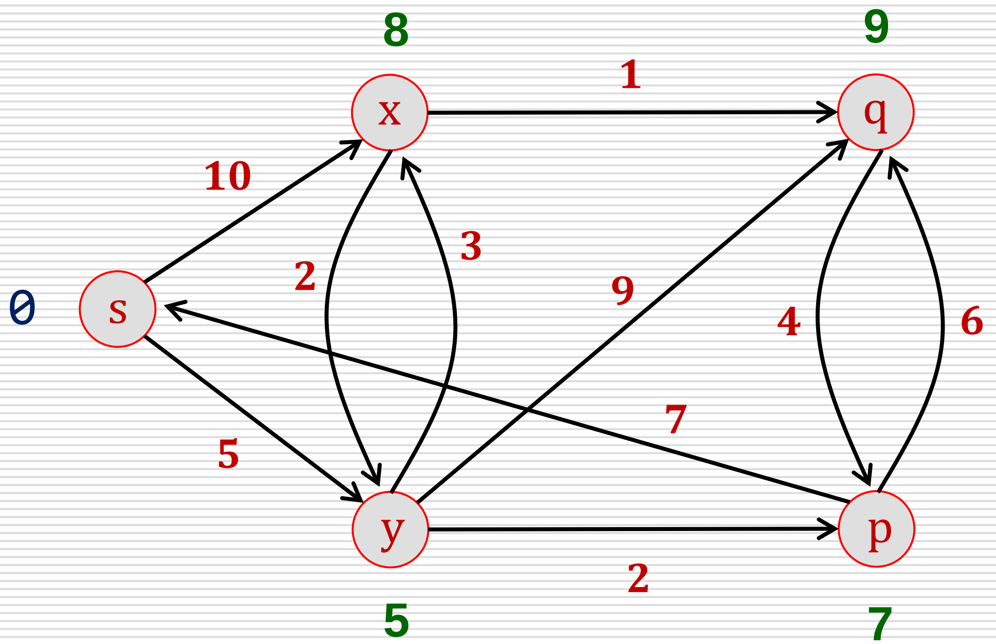
Dijkstra: 2nd example

V\S = {}



d(s)=0	d(x)=8	d(y)=5	d(p)=7	d(q)=9
prev(s)=NIL	prev(x)=y	prev(y)=s	prev(p)=y	prev(q)=x

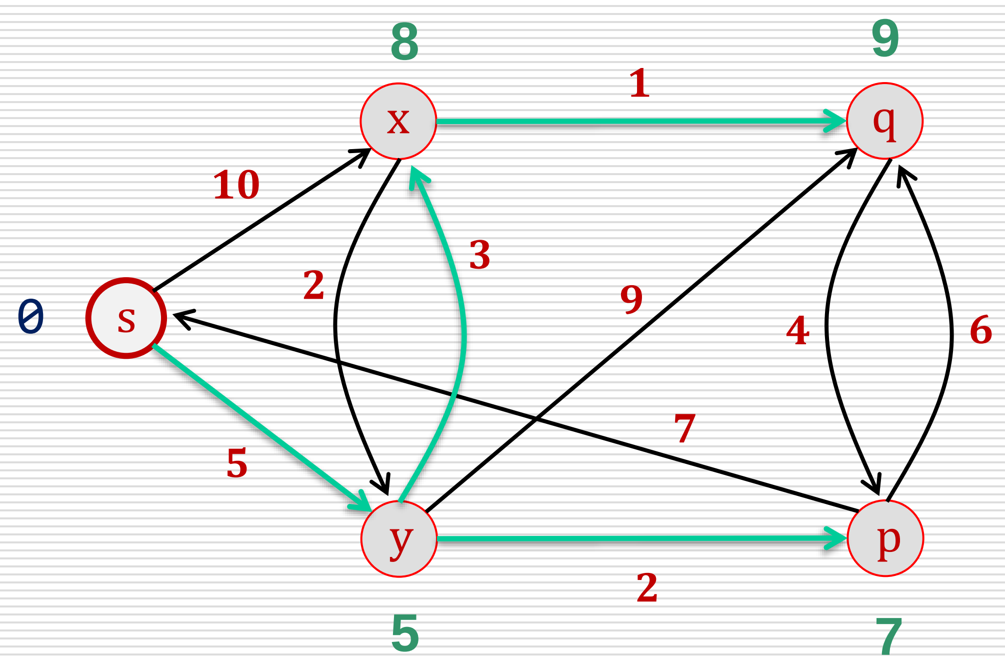
Dijkstra: 2^{nd} example



d(s)=0 d(x)=8 d(y)=5 d(p)=7 d(q)=9

prev(s)=NIL prev(x)=y prev(y)=s prev(p)=y prev(q)=x

Dijkstra: 2nd example



Output

Shortest Paths **cost**

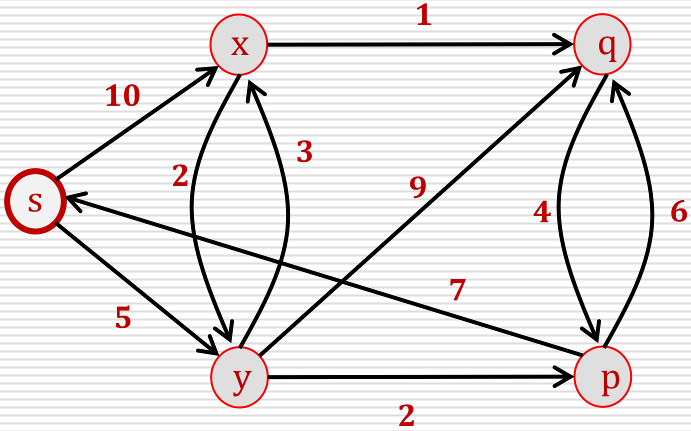
Previous nodes at shortest paths =>

Shortest Path Tree

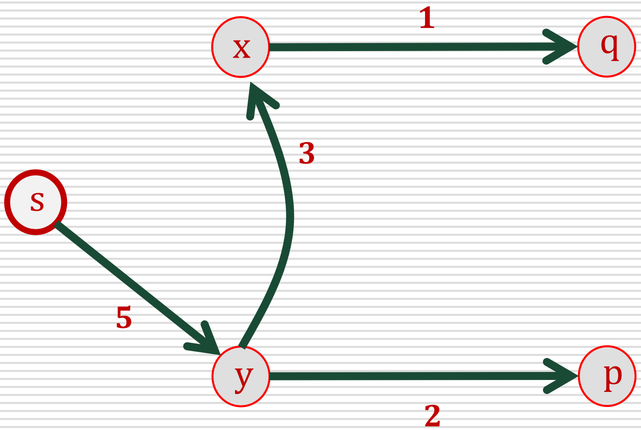
d(s)=0 **d(x)=8** **d(y)=5** **d(p)=7** **d(q)=9**

prev(s)=NIL **prev(x)=y** **prev(y)=s** **prev(p)=y** **prev(q)=x**

Dijkstra: 2nd example

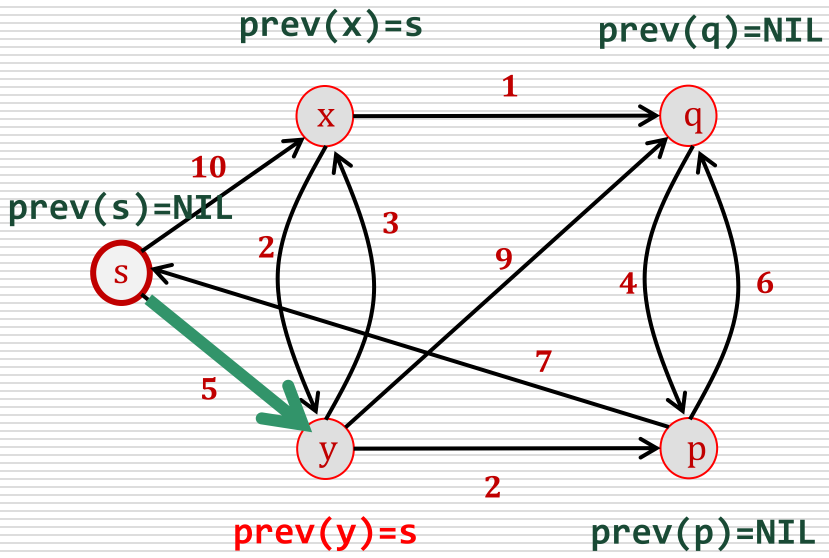


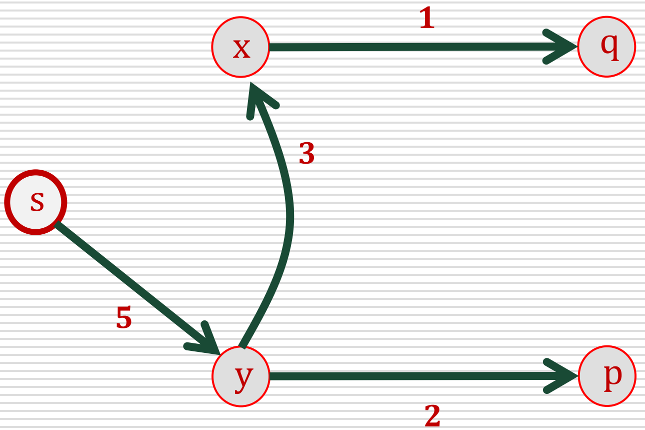
Shortest Path Tree



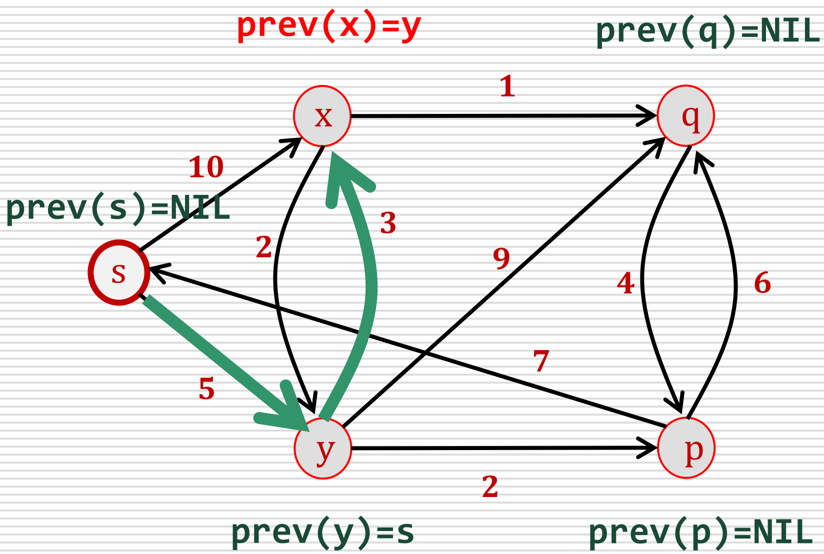
Dijkstra: 2nd example

Shortest Path Tree
Construction

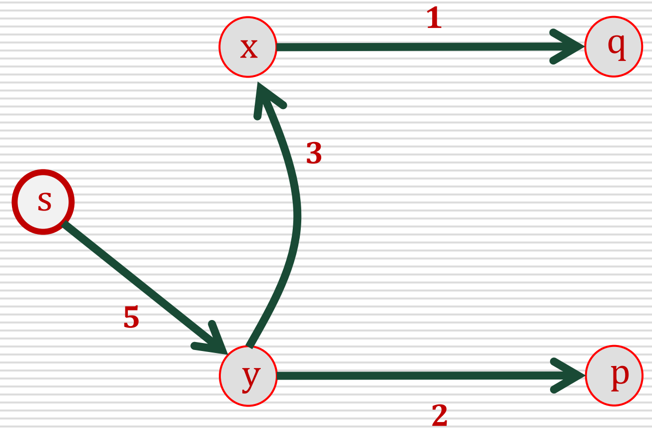




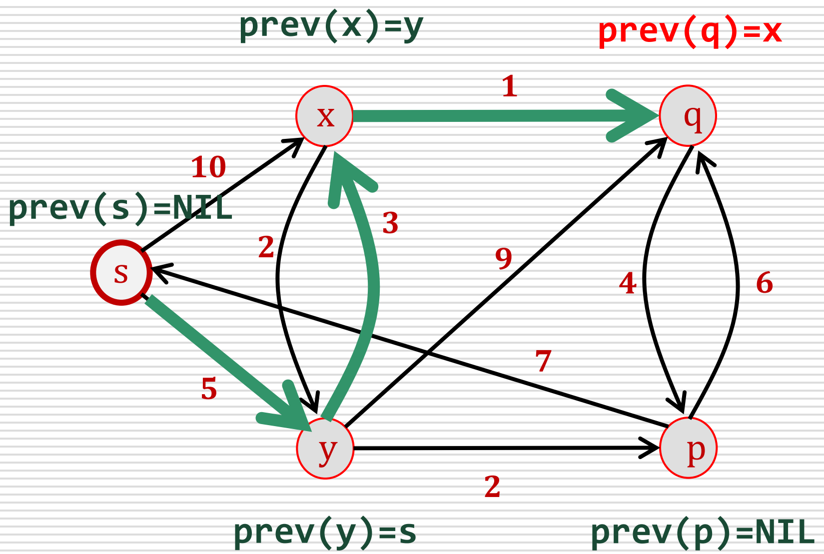
Dijkstra: 2nd example



Shortest Path Tree Construction

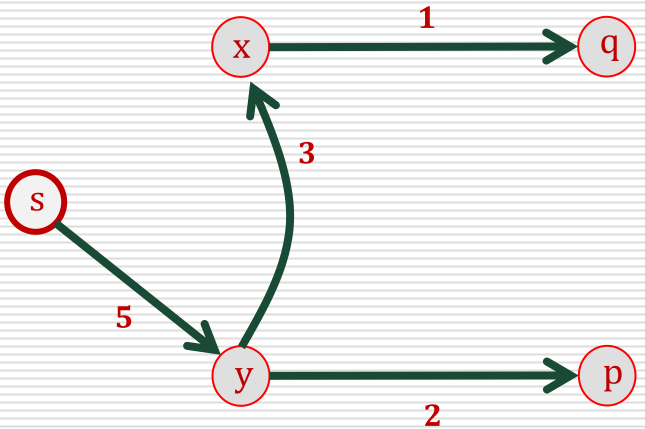


Dijkstra: 2^{nd} example



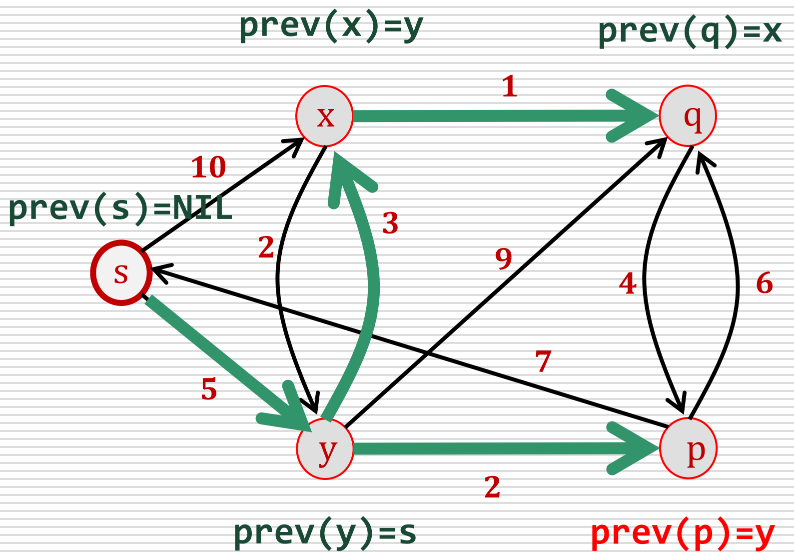
Shortest Path Tree

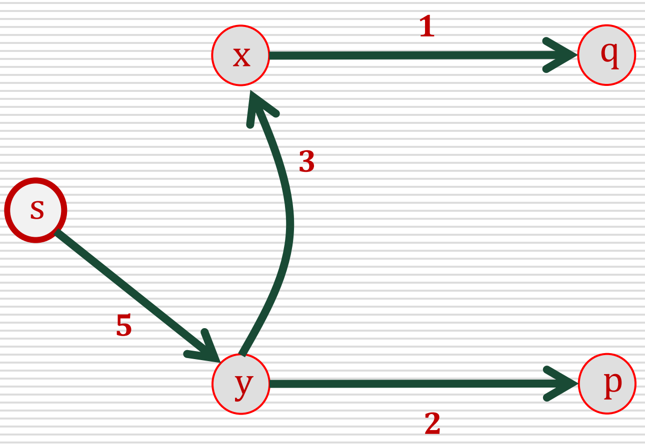
Construction



Dijkstra: 2nd example

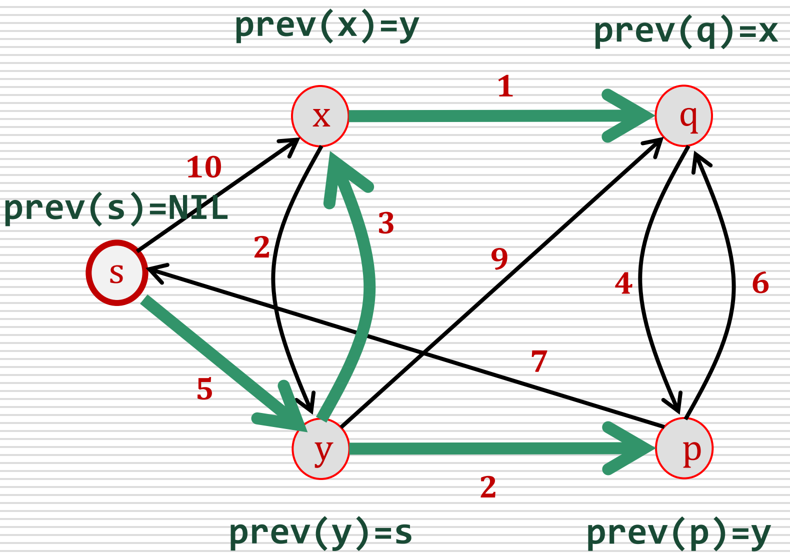
Shortest Path Tree Construction

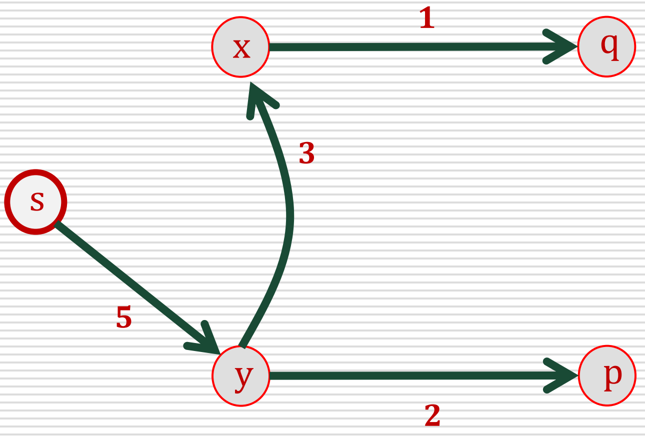




Dijkstra: 2nd example

Shortest Path Tree
Construction

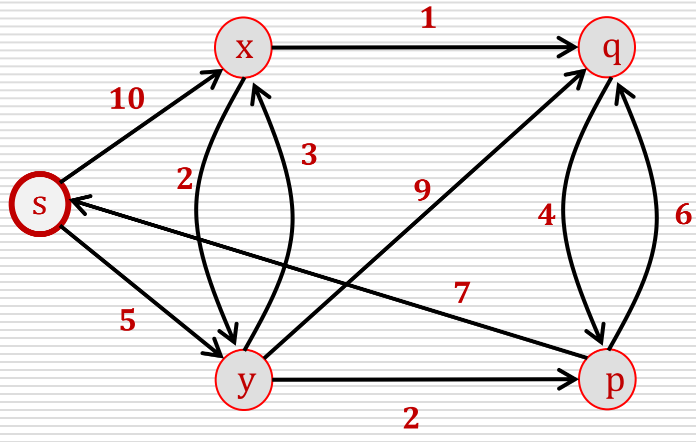


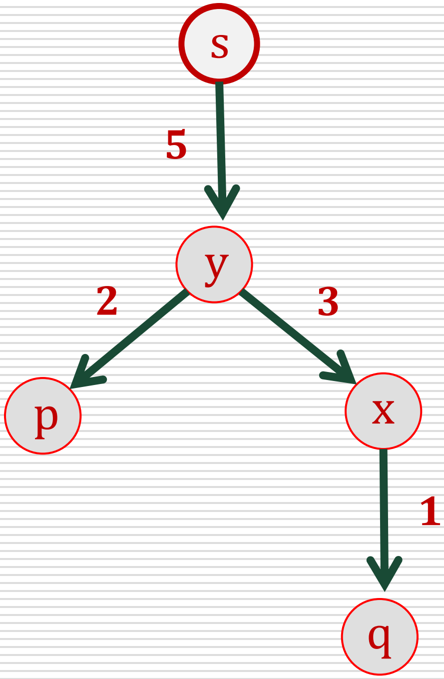


Dijkstra: 2nd example

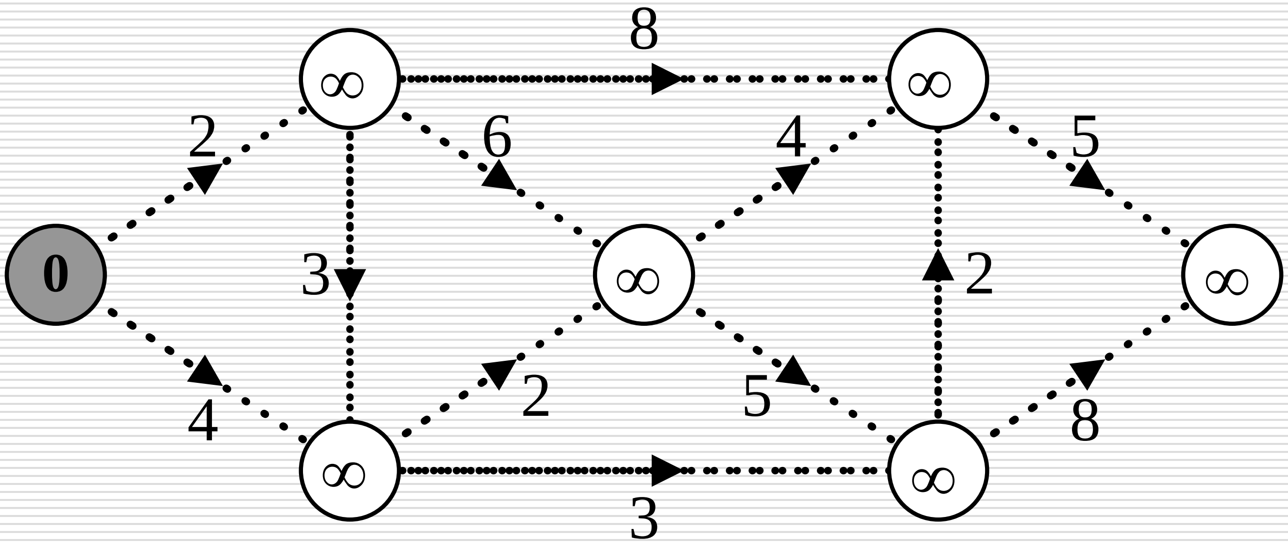
Shortest Path Tree

Construction



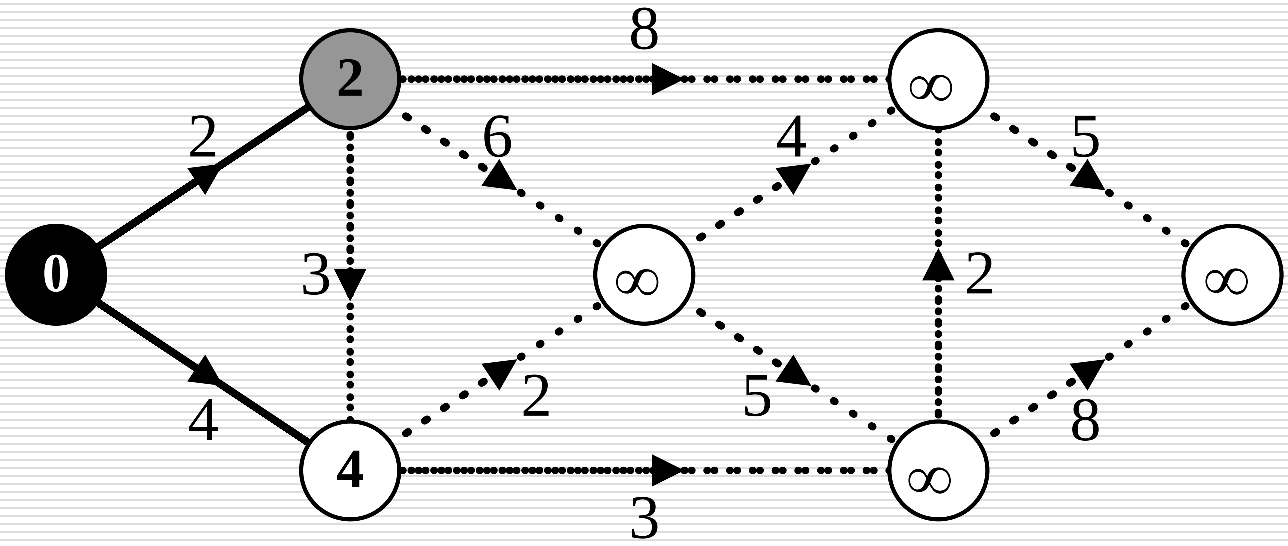


Dijkstra: 3rd example



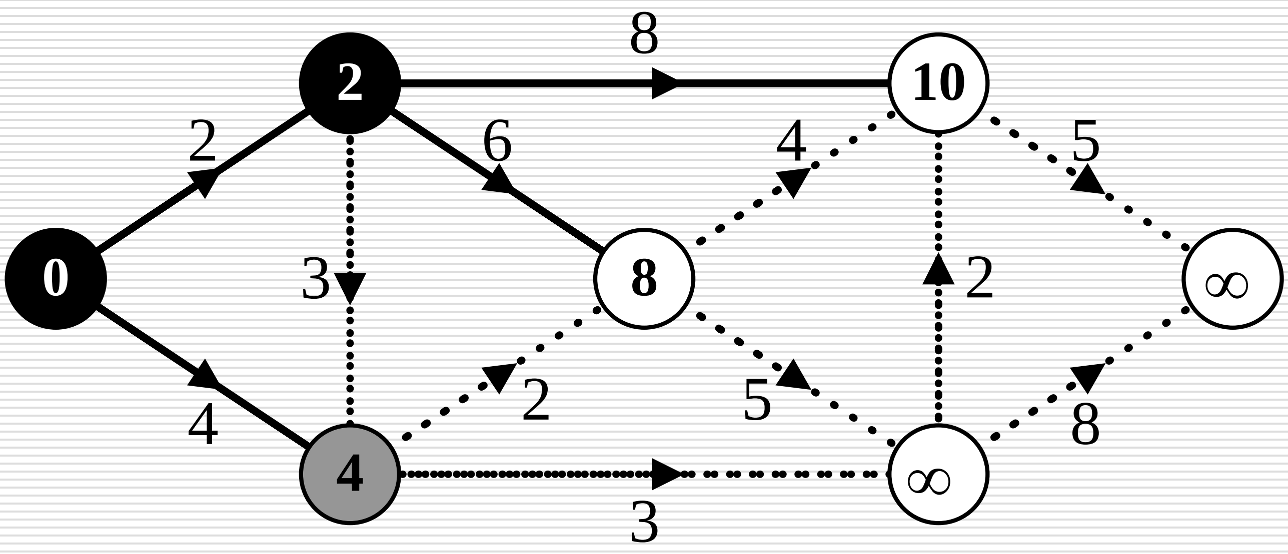
The node labels show the minimum distance from the original node so far (table D).

Dijkstra: 3rd example



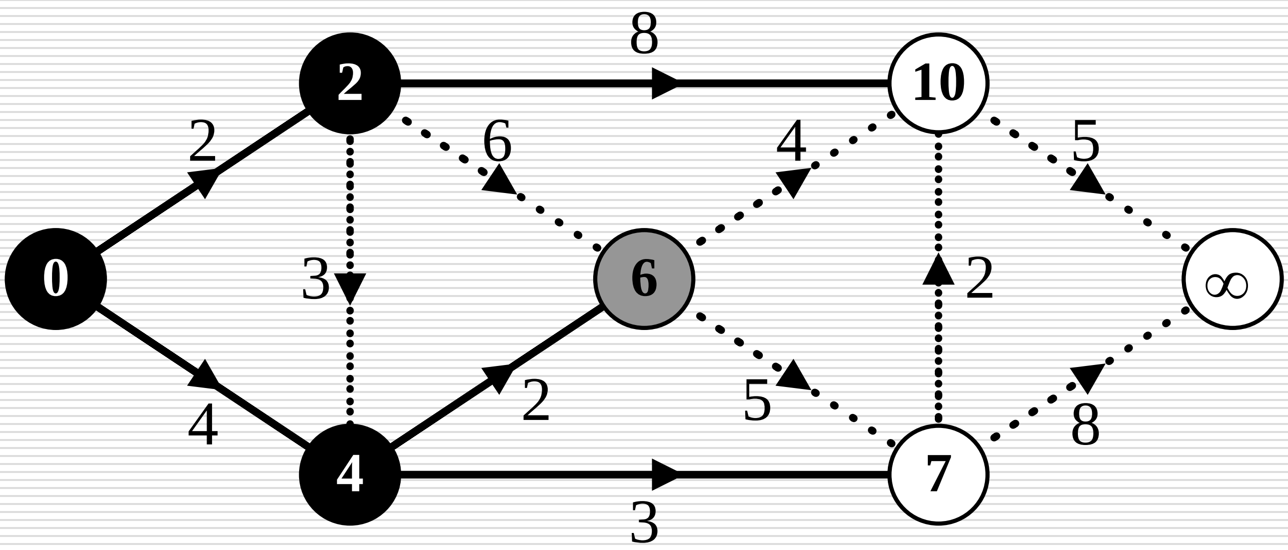
The node labels show the minimum distance from the original node so far (table D), the continuous edges indicate which is the corresponding previous node (table P).

Dijkstra: 3rd example



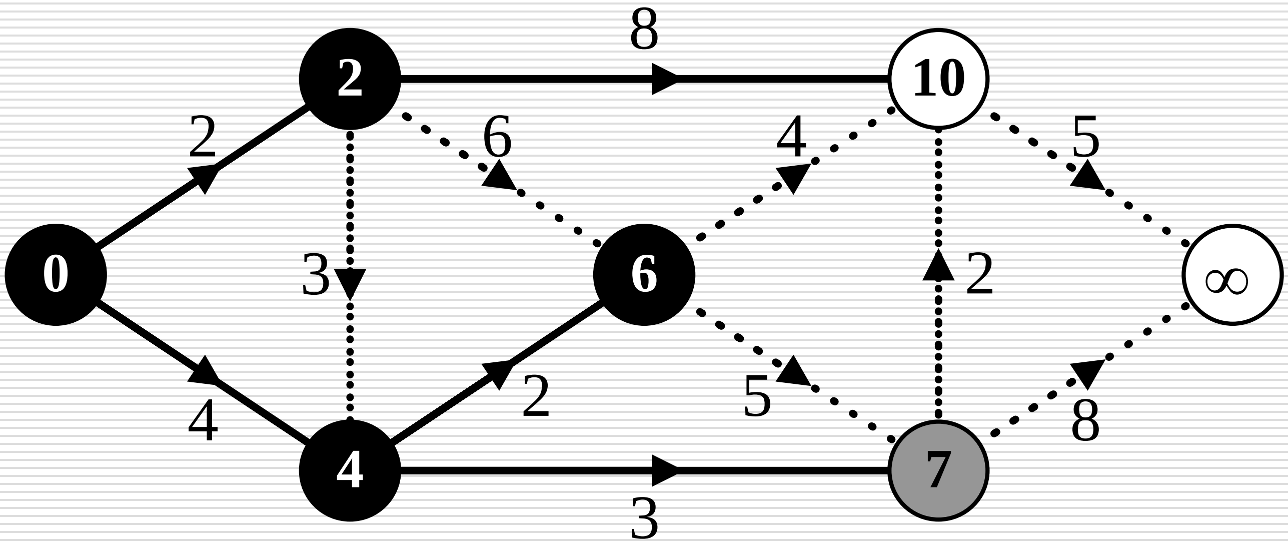
The node labels show the minimum distance from the original node so far (table D), the continuous edges indicate which is the corresponding previous node (table P).

Dijkstra: 3rd example



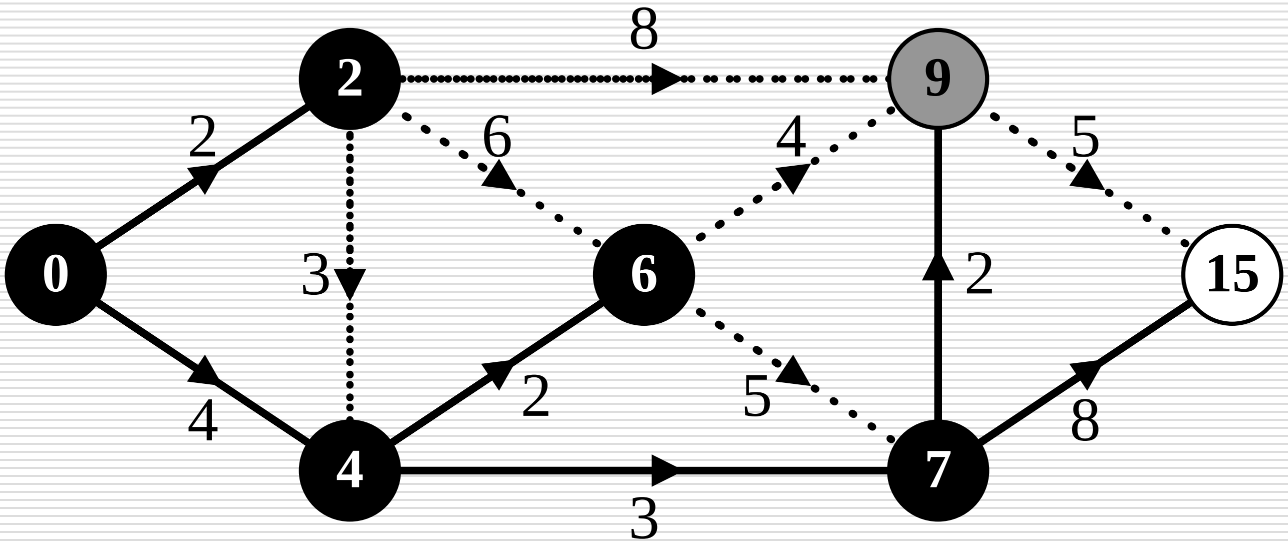
The node labels show the minimum distance from the original node so far (table D), the continuous edges indicate which is the corresponding previous node (table P).

Dijkstra: 3rd example



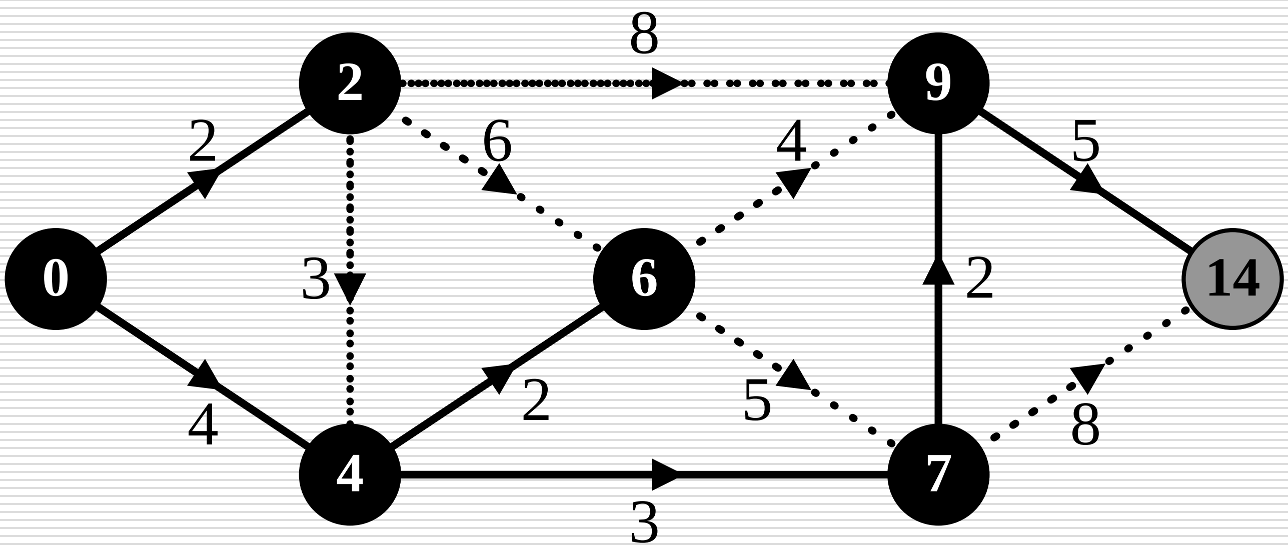
The node labels show the minimum distance from the original node so far (table D), the continuous edges indicate which is the corresponding previous node (table P).

Dijkstra: 3rd example



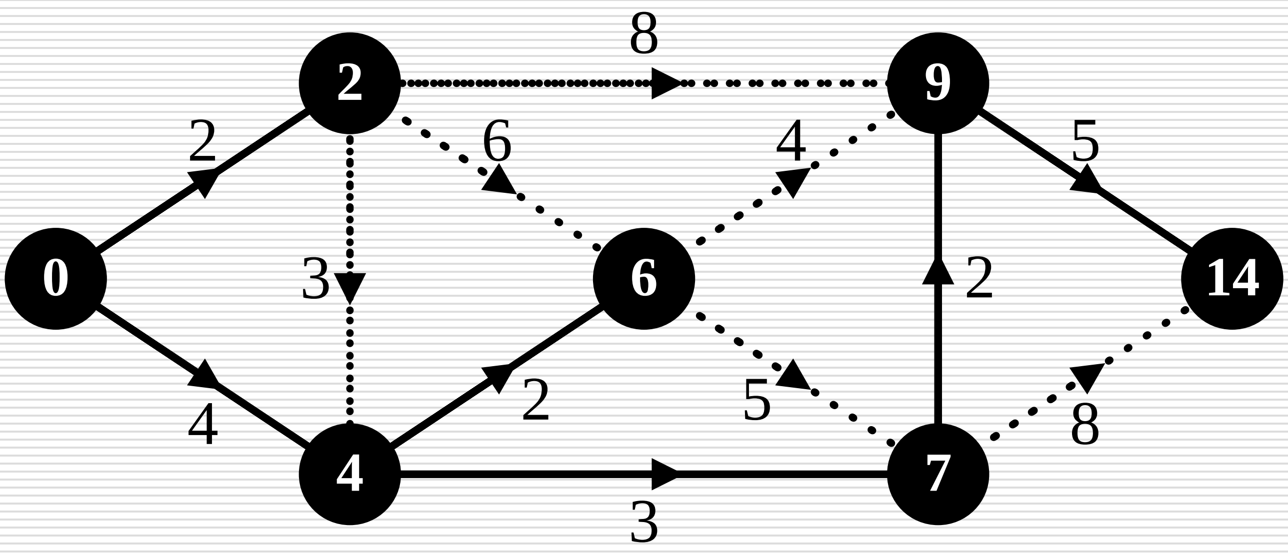
The node labels show the minimum distance from the original node so far (table D), the continuous edges indicate which is the corresponding previous node (table P).

Dijkstra: 3rd example



The node labels show the minimum distance from the original node so far (table D), the continuous edges indicate which is the corresponding previous node (table P).

Dijkstra: 3rd example



The node labels show the minimum distance from the original node so far (table D), the continuous edges indicate which is the corresponding previous node (table P).

Dijkstra Complexity

- Similarities with BFS algorithm?
- Slower than BFS (priority queue vs simple queue)
- It requires **n=|V|** insert operations in the queue, and **m=|E|** update operations:

  |V| insert / extract-min

  |E| update

- Implementing a queue with a matrix $\Rightarrow$ **O(|V|²)**
- Implementing a queue with a binary heap $\Rightarrow$ **O(|E| log|V|)**
- Implementing a queue with Fibonacci heap$\Rightarrow$**O(|E|+|V|log|V|)**

Dijkstra - Correctness

- The algorithm gradually builds a shortest paths tree. The tree is initialized with the initial node *s*.
- At each iteration, the node ***w*** with the minimum temporary label (current distance) from node *s* is selected.
- The proof is based on two invariant loop conditions.

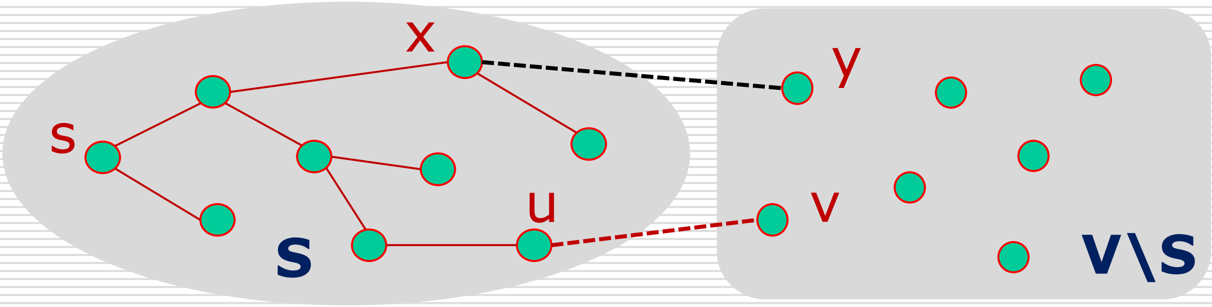
Dijkstra - Correctness

- 1st loop invariant: after each iteration of the outer loop, the label D(u) of each node u of S is equal to the cost of the shortest path from s to u, and the label D(v) of each node v of V\S is equal to the cost of the shortest path from s to v, *among all paths that pass only through nodes of the set* S.
- 2nd loop invariant: after each iteration of the outer loop, for node w in V\S with *minimum* (between nodes of V\S) label D(w), this is equal to the cost of the shortest path from s to w.
- Proof: by induction.

Dijkstra - Correctness

Lemma: 1^{st} invariant => 2^{nd} invariant

Let G = (V, E) be a graph with non-negative weights, S∈V, and (S, V\S) a partition of V s.t. s∈S and ∀u∈S the label of u equals the minimum distance D(s, u).



Let v∈V\S the node with the minimum label in V\S. Then there exists an edge (u, v) that minimizes the quantity

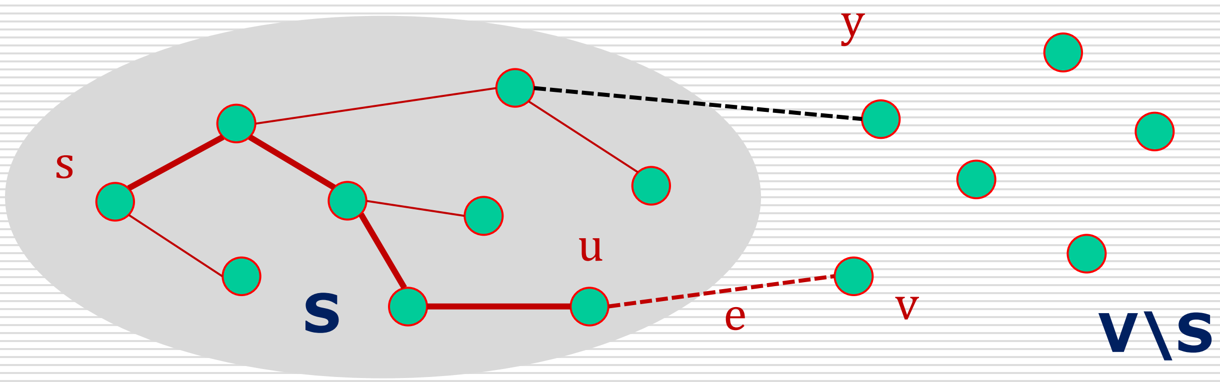
$$\mathbf{D(s, u) + c(u, v)}$$

between all edges (x, y) with x∈S and y∈V\S. W.T.S.

P = (s, …, u, v) is s.p. from s to v.

Dijkstra - Correctness

Proof



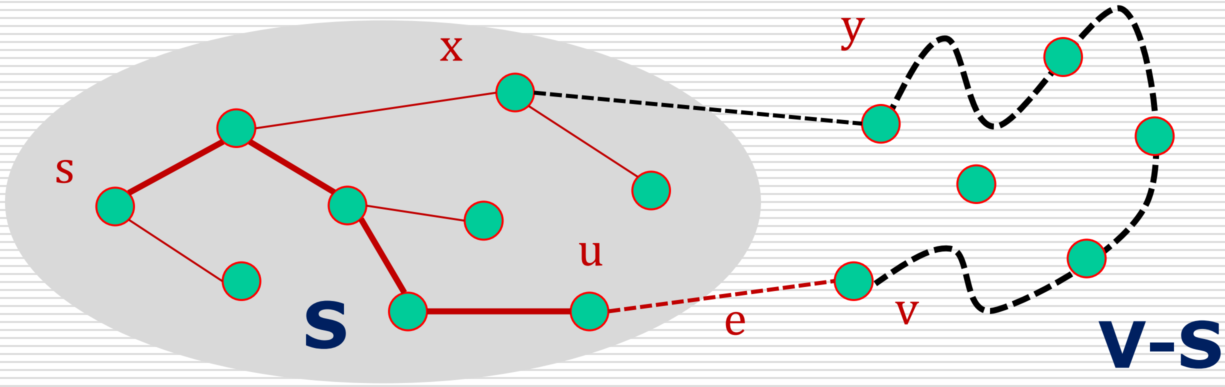
Let e = (u, v) and let (s, …, u) be the shortest path from s to u.

For the path P = (s, …, u, v) from s to v we have :

$$c(P) = D(s,u) + c(u,v) \qquad (1)$$

Dijkstra - Correctness

Proof



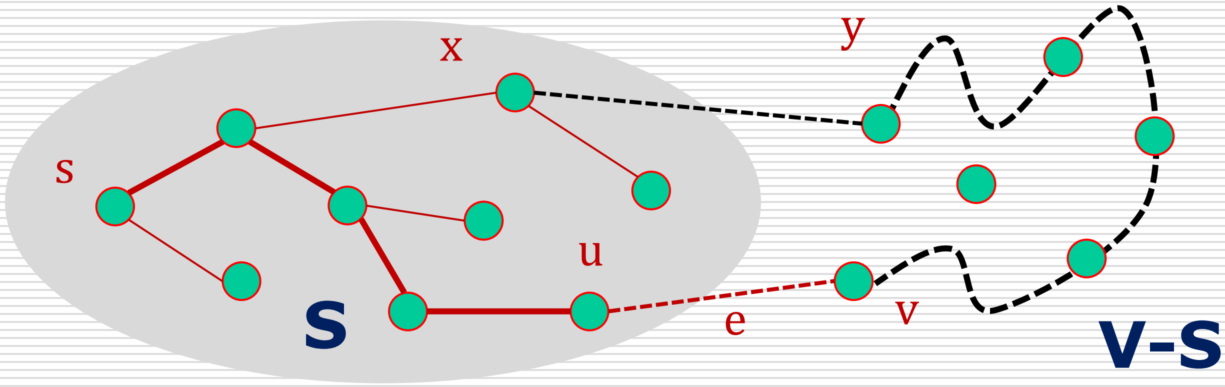
Let Q = (s, …, x, y, …, v) a shortest path from s to v, and

let y the first node of path Q : y∈V\S

We will demonstrate that **c(P) ≤ c(Q)**

Dijkstra - Correctness

Proof



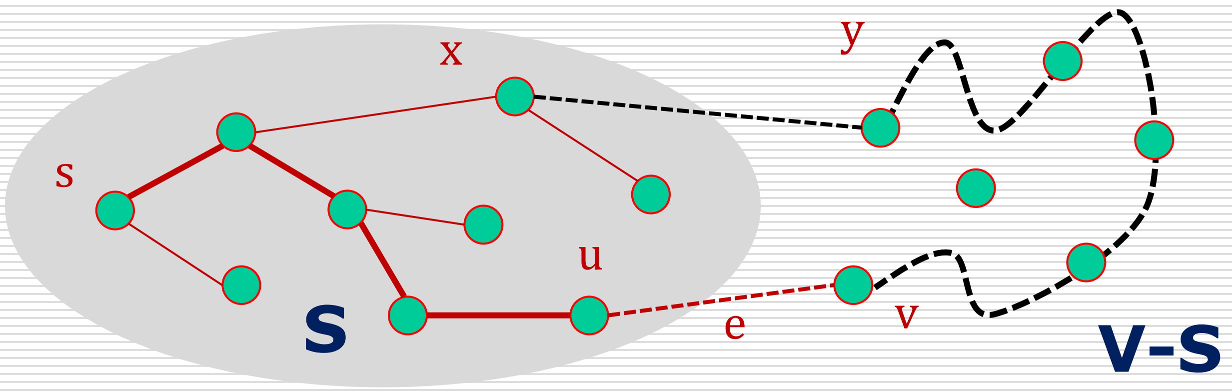
From (1) and by choosing the edge e = (u, v), we obtain:

$$\mathbf{c(P)} = D(s,u) + c(u,v) \le D(s, x) + c(x, y) \le \mathbf{c(Q)}$$

Exercise: *complete the proof.*

Dijkstra - Correctness

Proof



From (1) and by choosing the edge e = (u, v), we obtain:

c(P) = D(s,u) + c(u,v) ≤ D(s, x) + c(x, y) ≤ **c(Q)**

Exercise: *complete the proof.*

What about negative weights?

Dijkstra - Correctness

- Correctness applies **only** to graphs **without negative weights**

- *Counterexample?*

Bellman-Ford Algorithm

dist(s) := 0 ; p(s) := NIL

for each v<>s **do** dist(v) := ∞; p(v) := NIL

repeat n-1 times

    for each edge e = (u,v) **do**

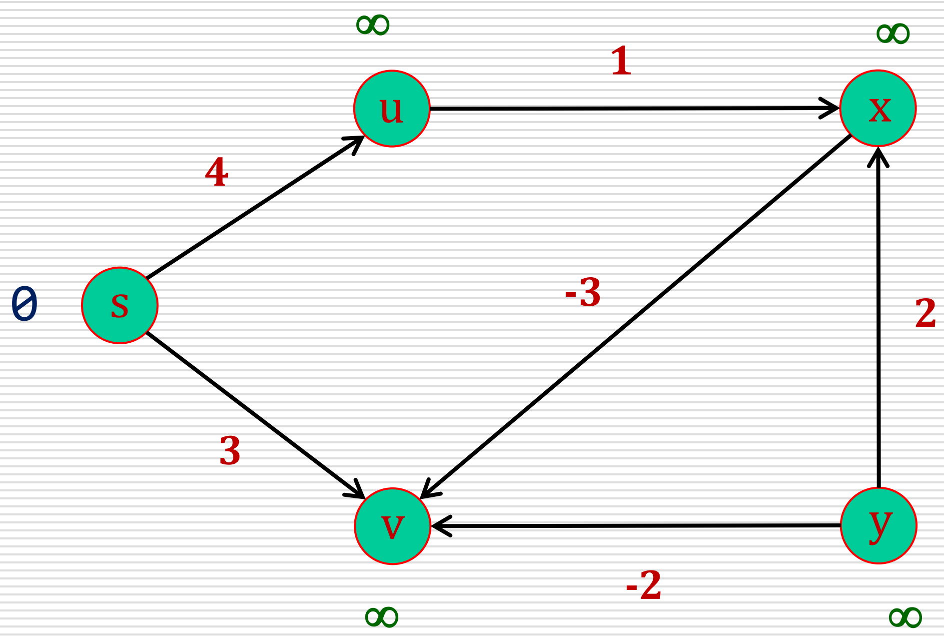
        if dist(u) + cost(u,v) < dist(v) **then**

            dist(v) := dist(u) + cost(u,v)

            p(v) := u

Bellman-Ford Algorithm

Examples



n = 5
m = 6

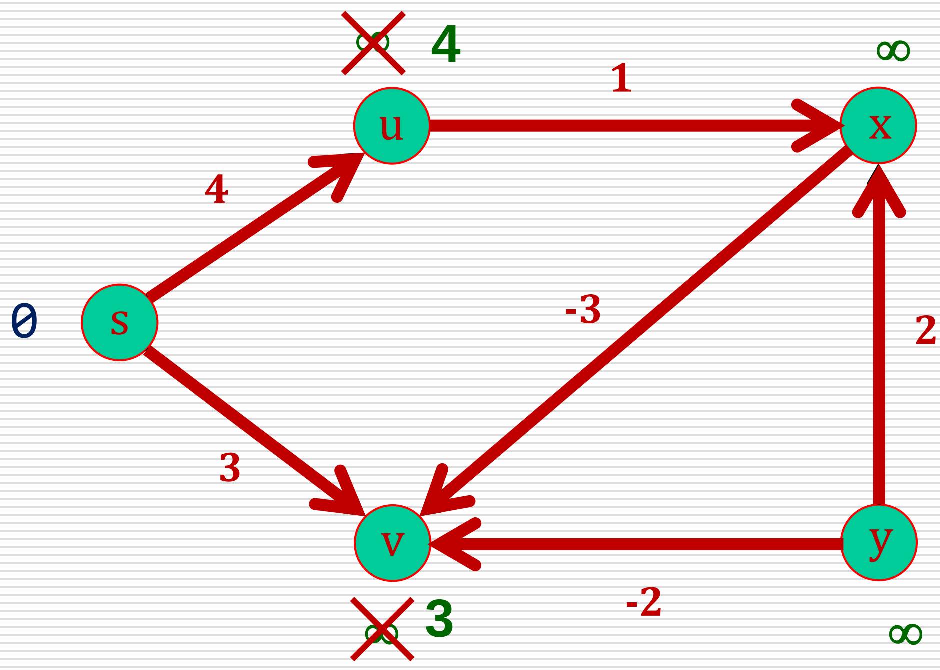
Update
6 edges
4 times

V = {s, u, v, x, y}

E = {(y,x),(u,x),(y,v),(s,u),(x,v),(s,v)}

Bellman-Ford Algorithm

Example 1



n = 5
m = 6

Update
6 edges
4 times

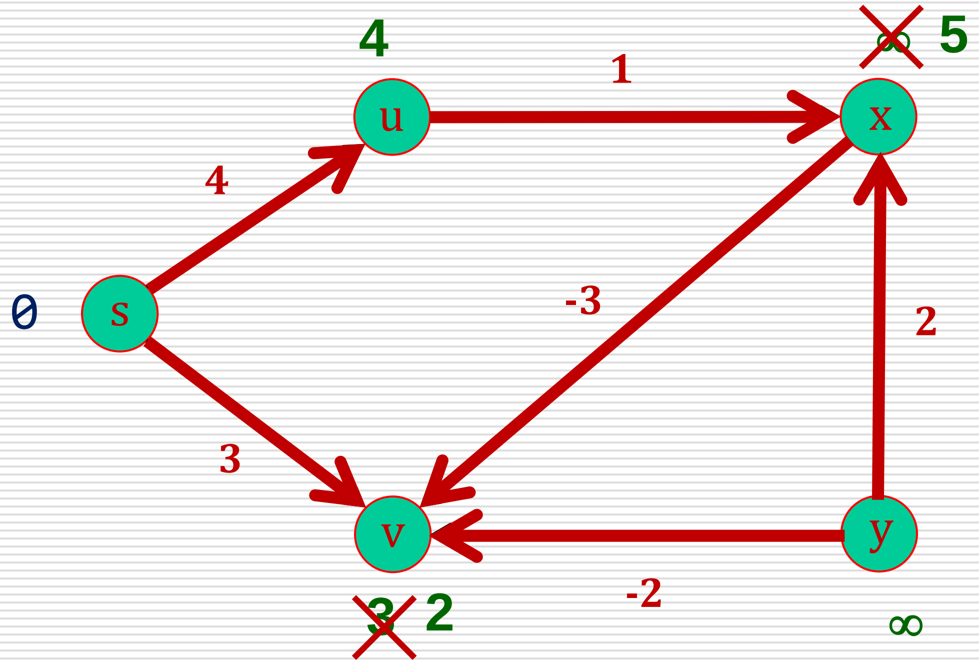
1^{st} Iteration

S1: (y,x), (u,x), (y,v), (s,u), (x,v), (s,v)

✓ ✓ ✓ ✓ ✓ ✓

Bellman-Ford Algorithm

Example 1



n = 5
m = 6

Update
6 edges
4 times

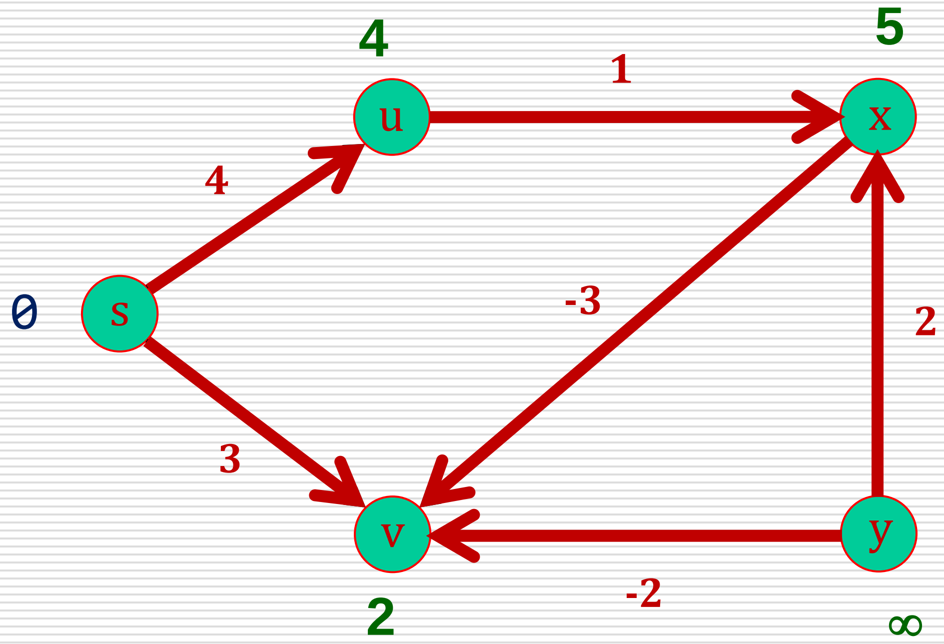
2nd Iteration

S1: (y,x), (u,x), (y,v), (s,u), (x,v), (s,v)

✓ ✓ ✓ ✓ ✓ ✓

Bellman-Ford Algorithm

Example 1



n = 5
m = 6

Update
6 edges
4 times

3rd Iteration

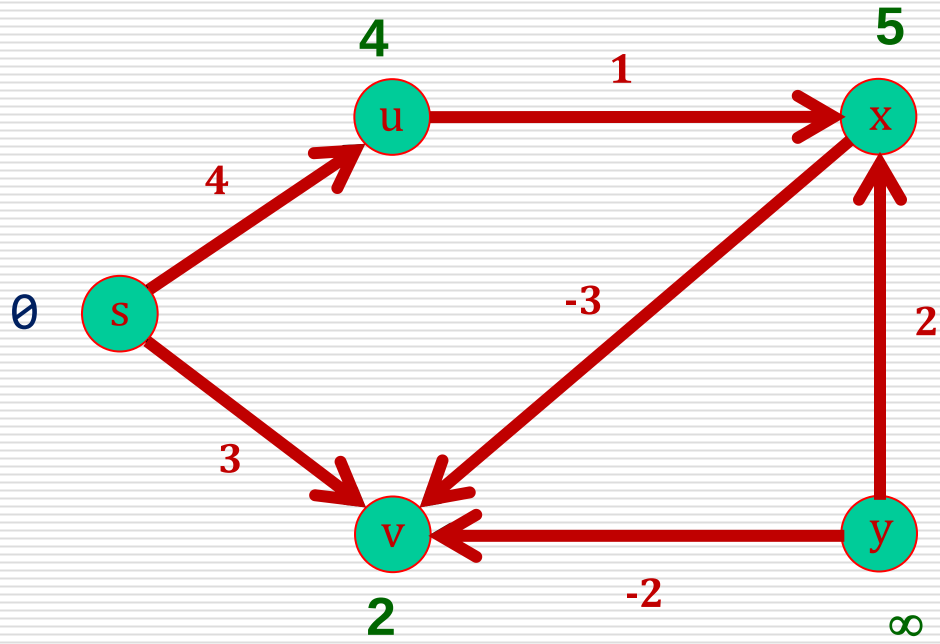
No change
why ?

S1: (y,x), (u,x), (y,v), (s,u), (x,v), (s,v)

✓ ✓ ✓ ✓ ✓ ✓

Bellman-Ford Algorithm

Example 1



n = 5
m = 6

Update
6 edges
4 times

4th Iteration

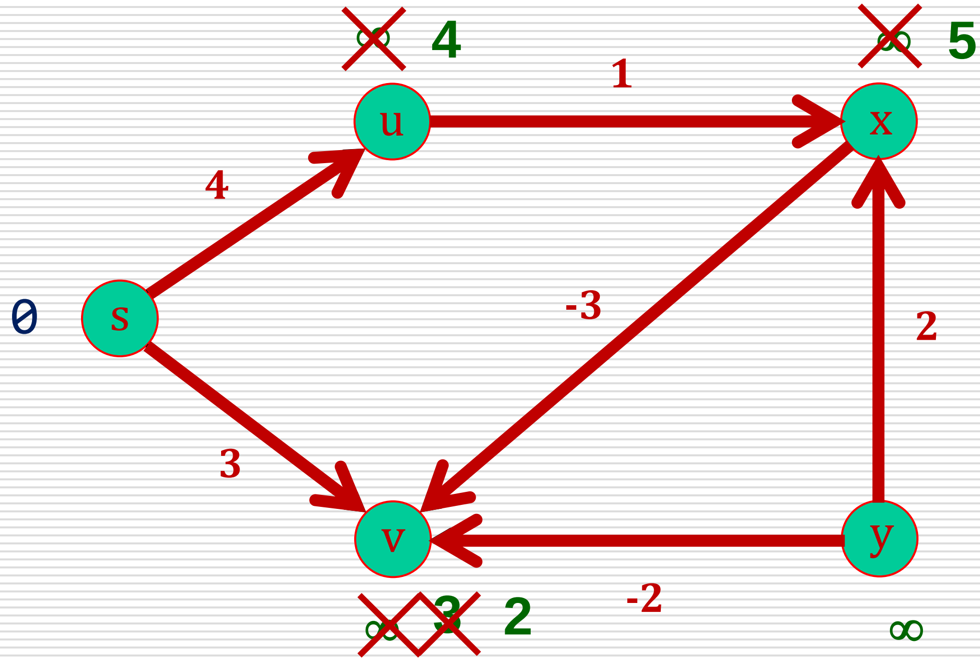
No change

S1: (y,x), (u,x), (y,v), (s,u), (x,v), (s,v)

✓ ✓ ✓ ✓ ✓ ✓

Bellman-Ford Algorithm

Example 2 (*different ordering of edges*!)



n = 5
m = 6

Update
6 edges
4 times

1^{st} Iteration

S2: (s,u), (u,x), (y,x), (s,v), (x,v), (y,v)

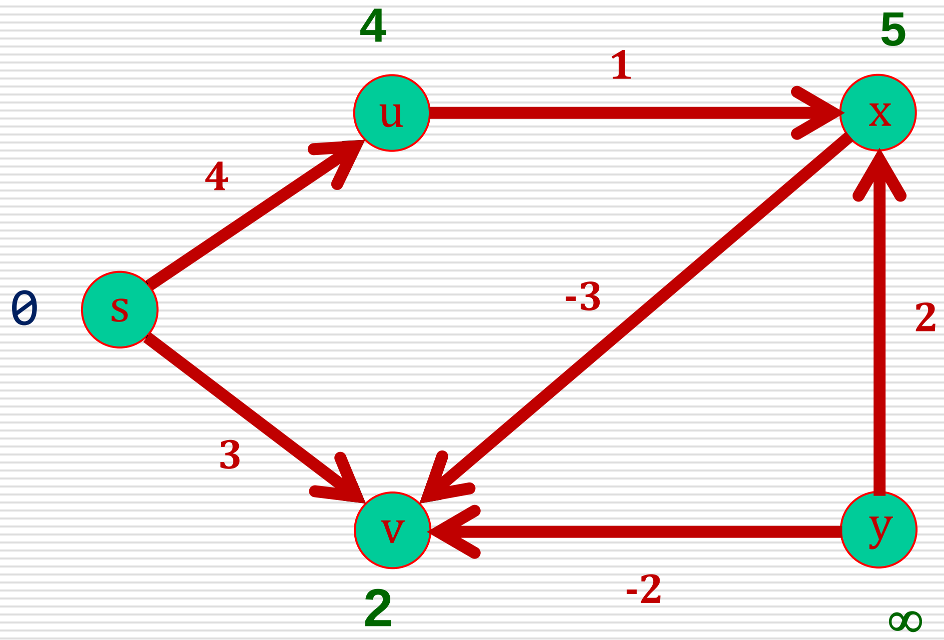
✓ ✓ ✓ ✓ ✓ ✓

Bellman-Ford Algorithm

Example 2 (*different ordering of edges!*)

n = 5
m = 6

Update
6 edges
4 times



2nd Iteration

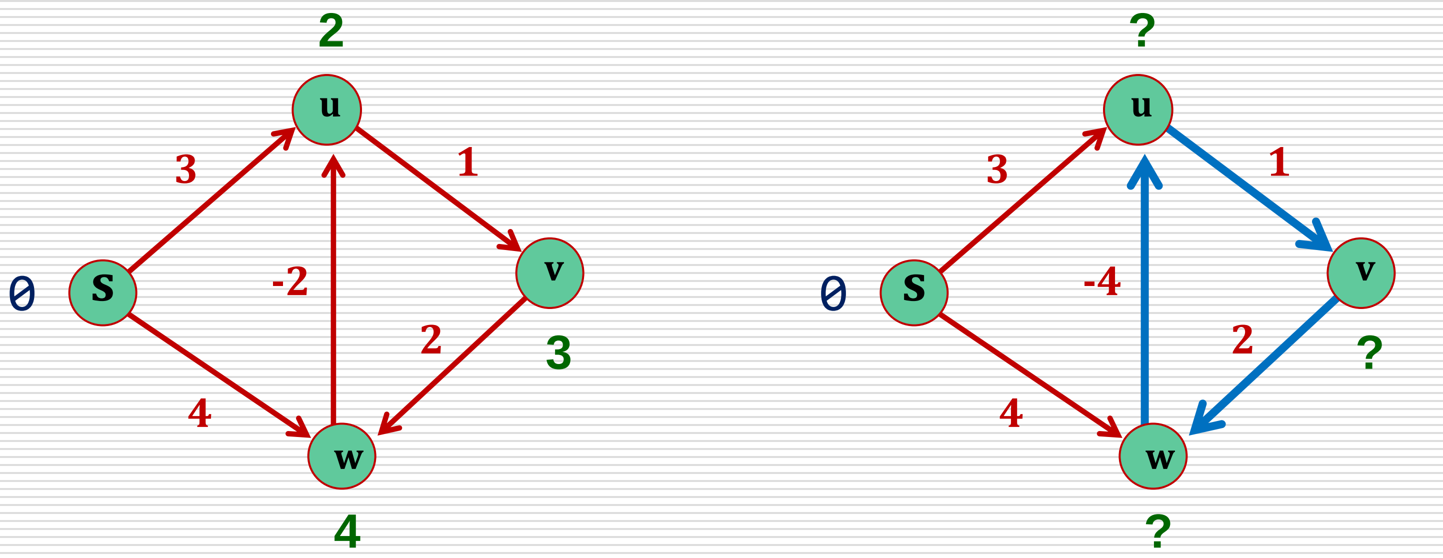
No change
why ?

S2: (s,u), (u,x), (y,x), (s,v), (x,v), (y,v)

✓ ✓ ✓ ✓ ✓ ✓

Bellman-Ford Algorithm

- **Negative Cycles**



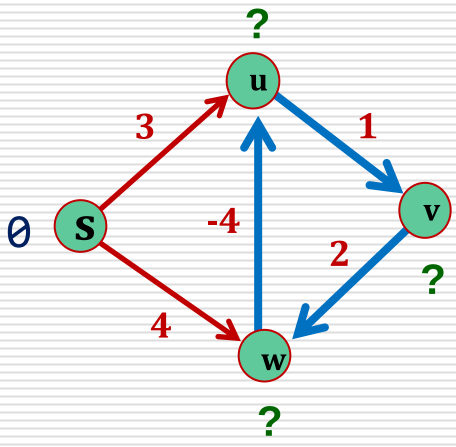
If there is a negative cycle, it makes no sense to search for shortest paths!!!

Bellman-Ford Algorithm

Negative cycle detection

```
Bellman-Ford-detection(G,w,s)

1.  initialize(G,s)
2.  repeat n-1 times:
          for all edges (u,v)∈E:
               Update(u,v,c)
3.  for all edges (u,v)∈E:
          if d(v) > d(u) + c(u,v) then
               return FALSE
4.  return d(.)
```



Bellman-Ford Algorithm

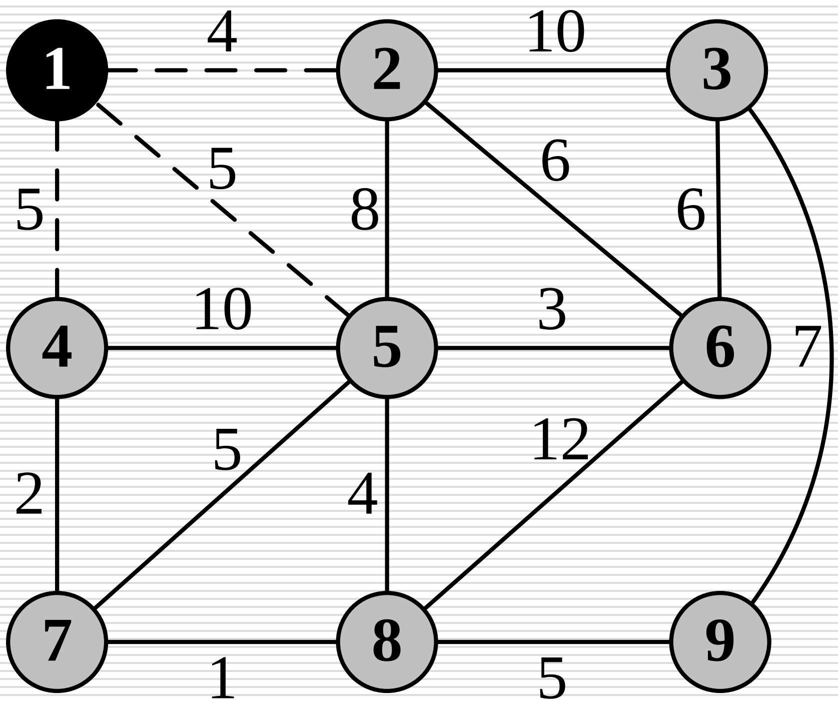
- **Correctness:** by the end of k-th iteration, the shortest paths consisting of at most k edges have been correctly computed (exercise: prove it).
- **Complexity**: **O(|V||E|)**

Minimum Spanning Tree (MST)

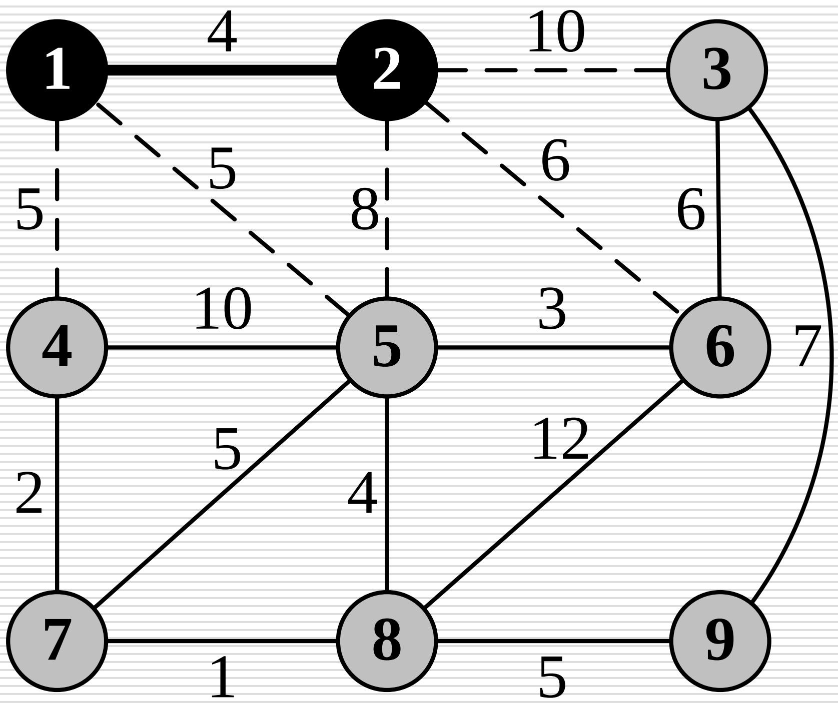
- **Criterion - Prim:** At each step, we select the edge with the minimum cost such that the new subgraph remains a tree (starting from any node)

- **Criterion - Kruskal:** At each step, we select the edge with the minimum cost such that the new subgraph remains cycle-free.

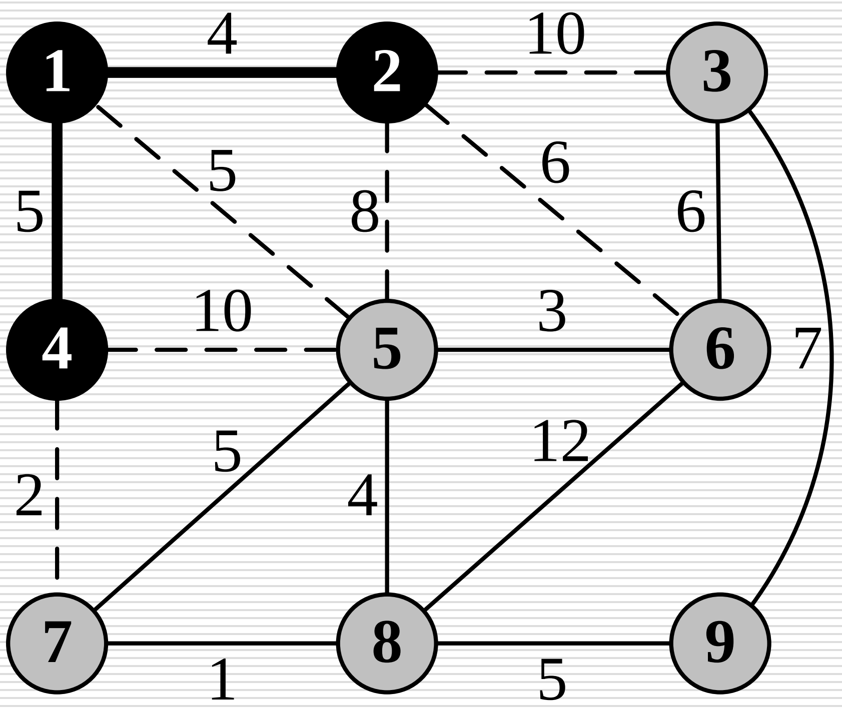
Prim Algorithm: example



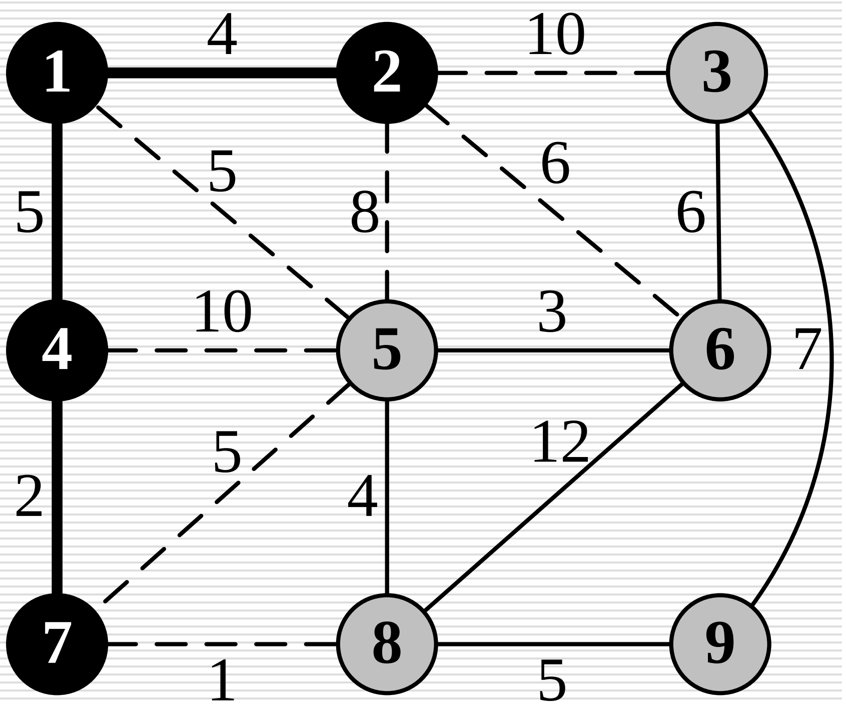
Prim Algorithm: example



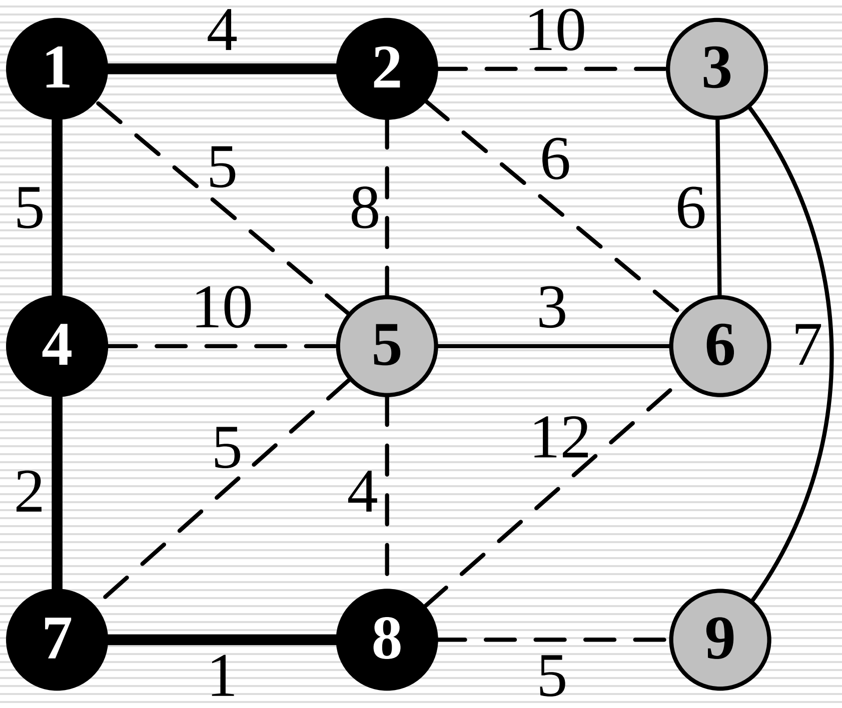
Prim Algorithm: example



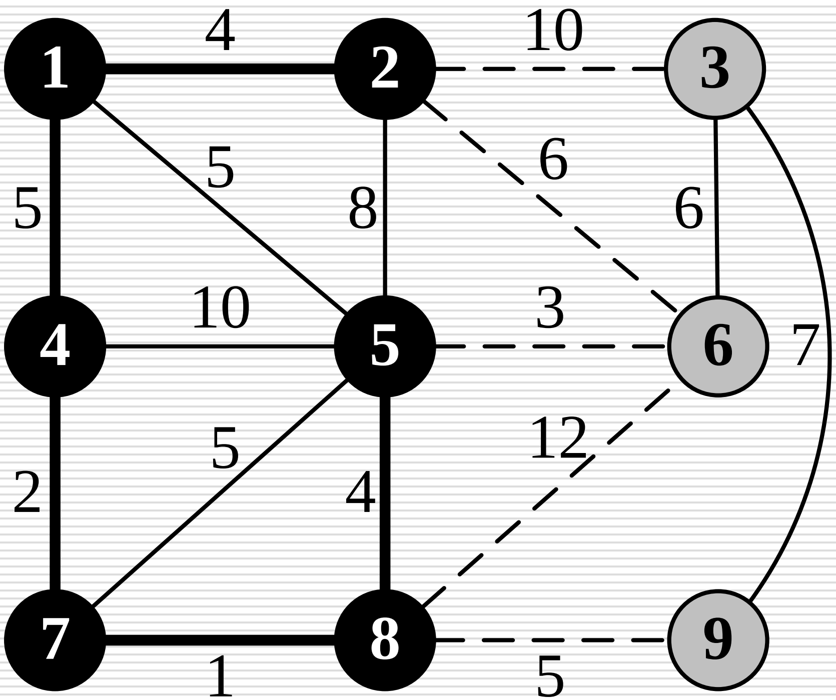
Prim Algorithm: example



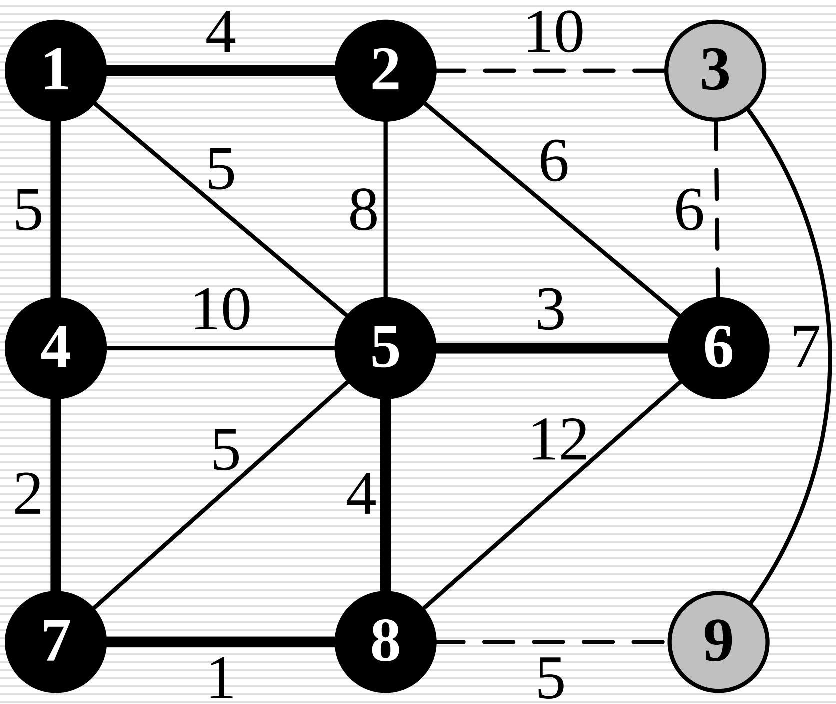
Prim Algorithm: example



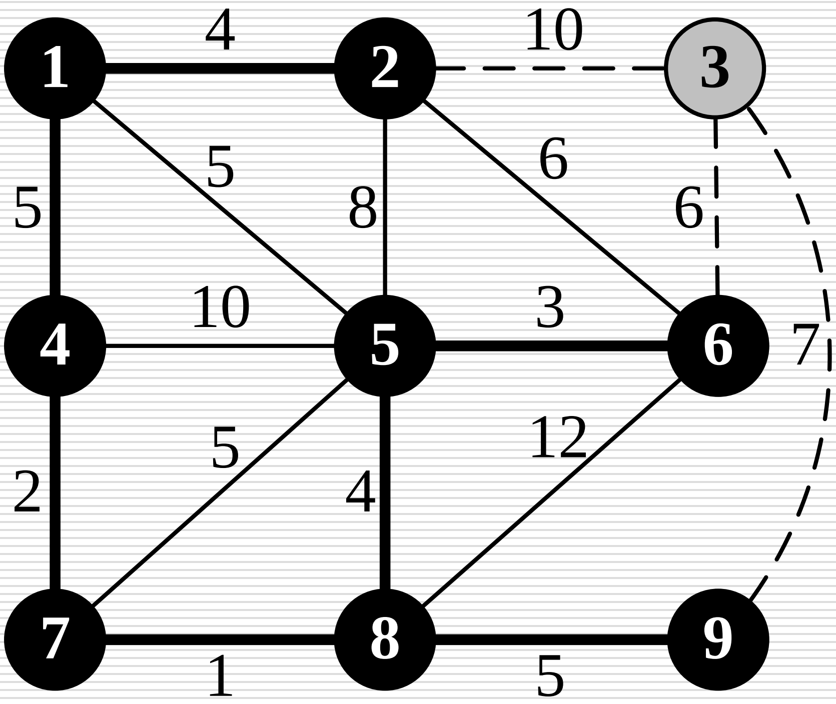
Prim Algorithm: example



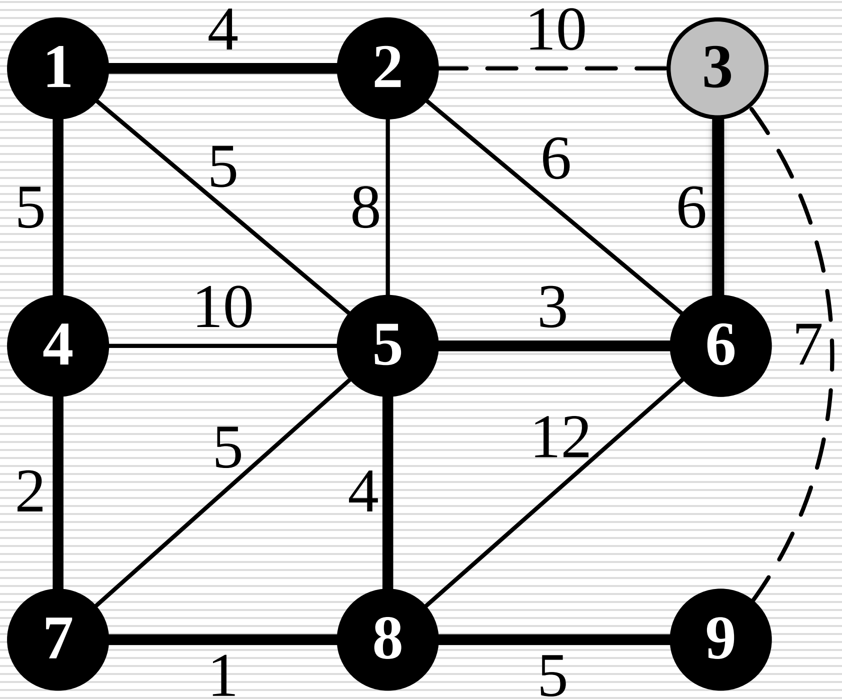
Prim Algorithm: example



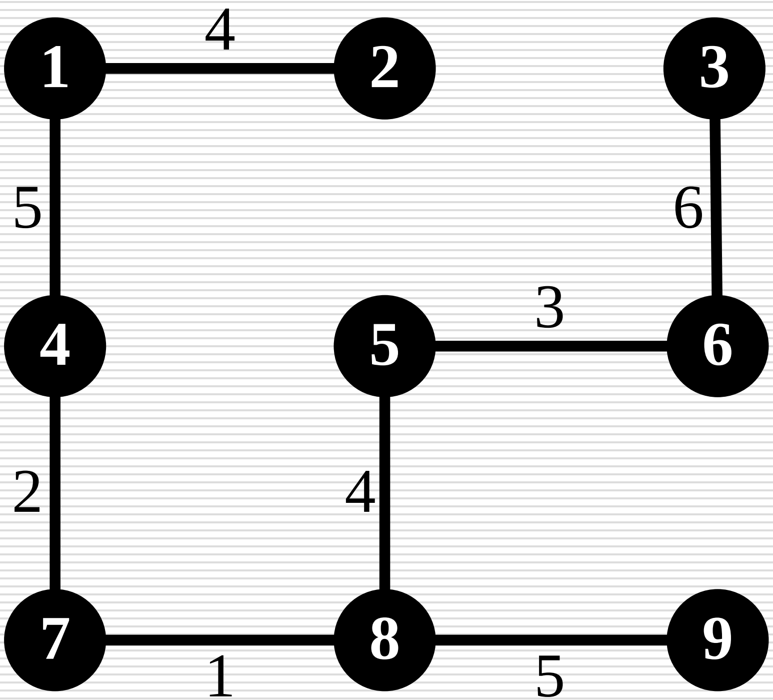
Prim Algorithm: example



Prim Algorithm: example



Prim Algorithm: example



Prim Algorithm

- An initial node, say *s*, is selected. Initialization:
  `dist(s):= 0;` **for each** `v<>s` **do** `dist(v) := ∞`

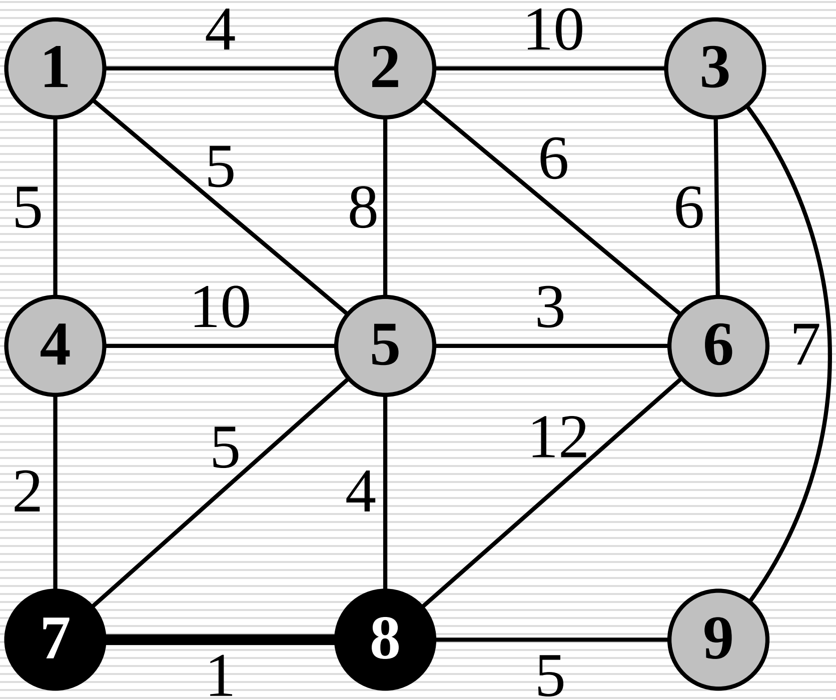
- Iteratively, the node w with the minimum distance from the tree constructed so far is selected and added to the tree. The distances of w's neighbors from the tree are updated based on the cost of the edges (w,u_i):

  if $cost(w,u_i) < d(u_i)$ **then**
  $d(u_i) := cost(w,u_i)$

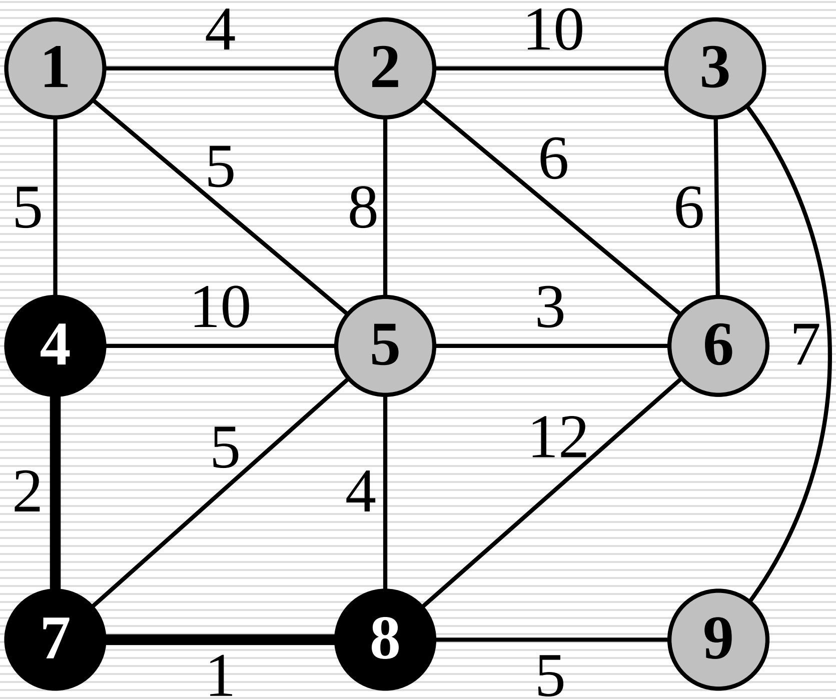
- A lot of similarities with Dijkstra (differences?)

- **Complexity**: **$O(|V|^2)$, $O(|E|\log|V|)$, $O(|E|+|V|\log|V|)$**

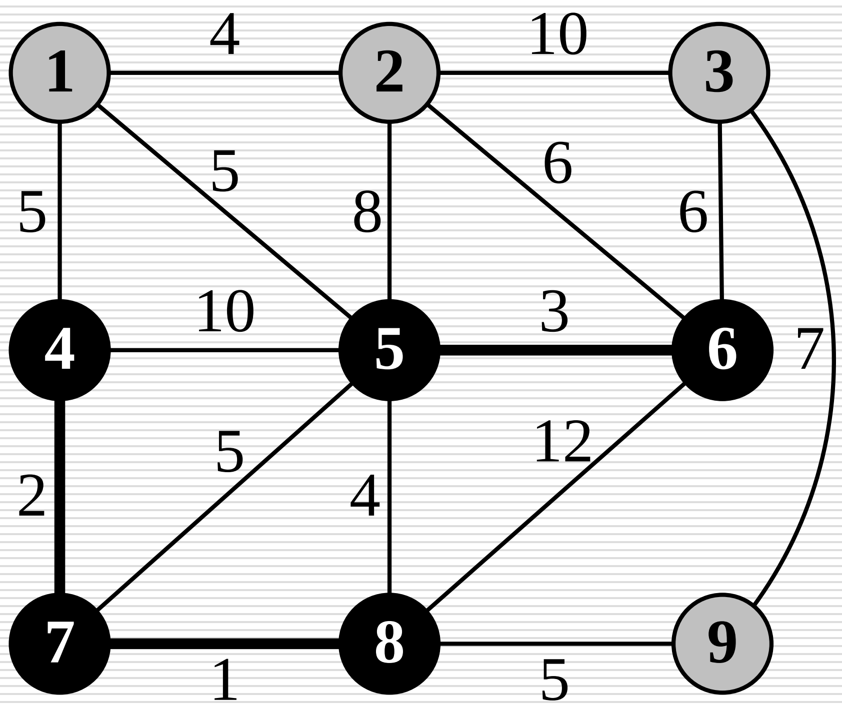
Kruskal Algorithm: example



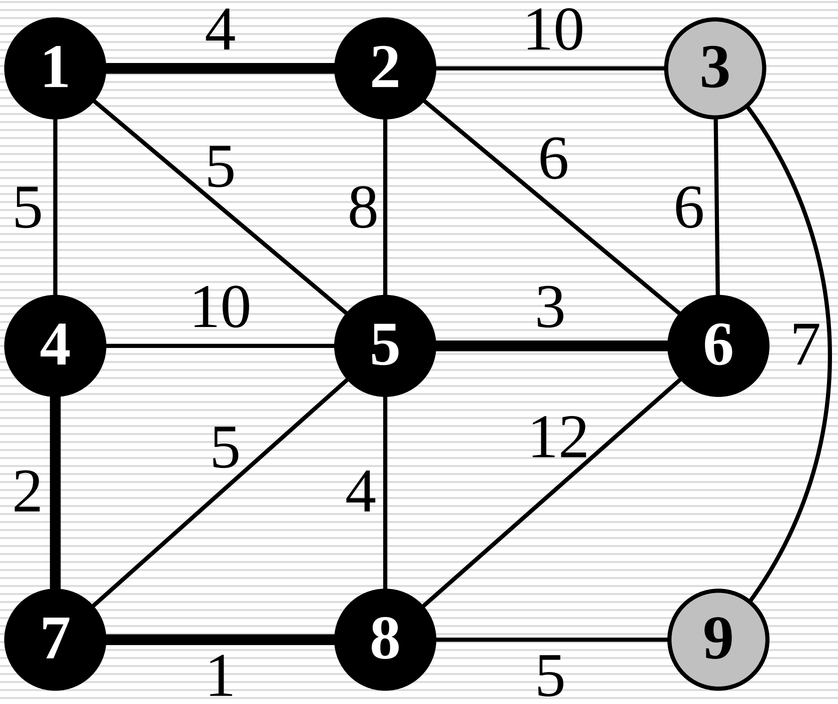
Kruskal Algorithm: example



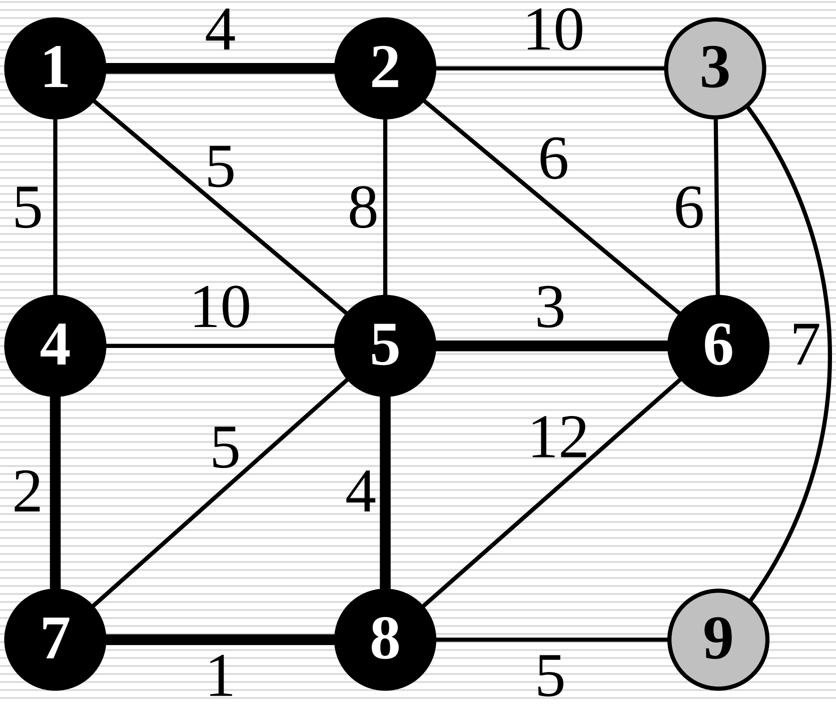
Kruskal Algorithm: example



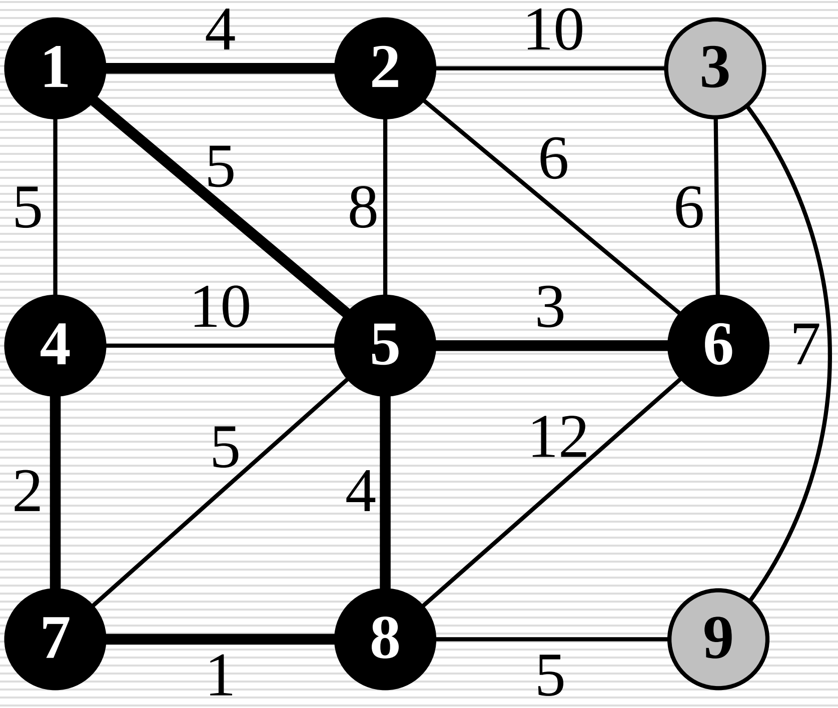
Kruskal Algorithm: example



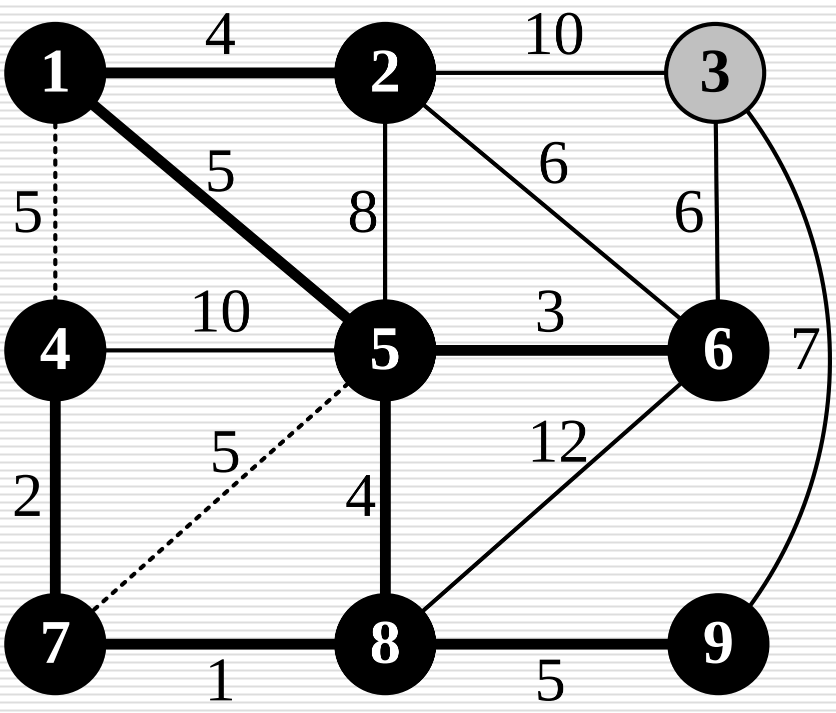
Kruskal Algorithm: example



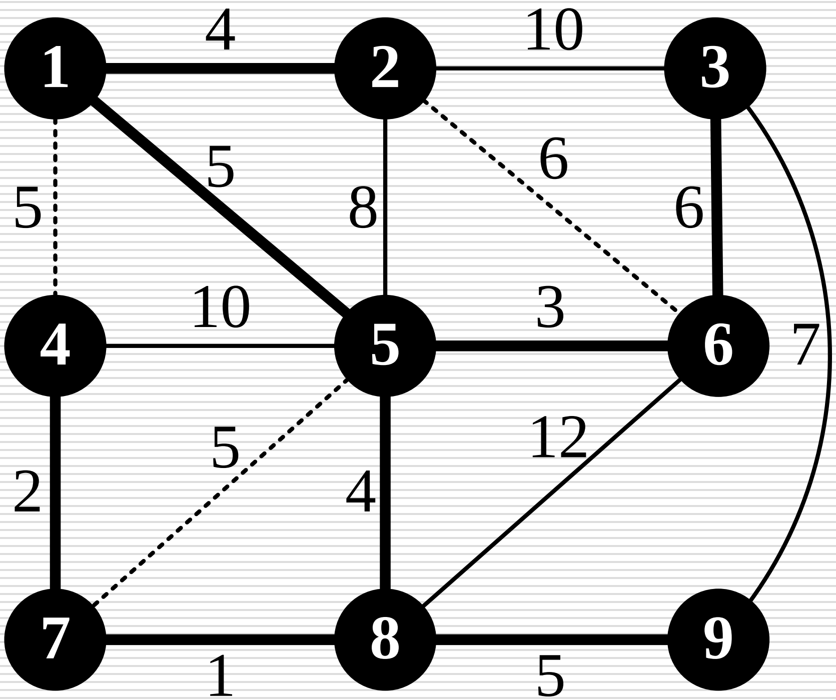
Kruskal Algorithm: example



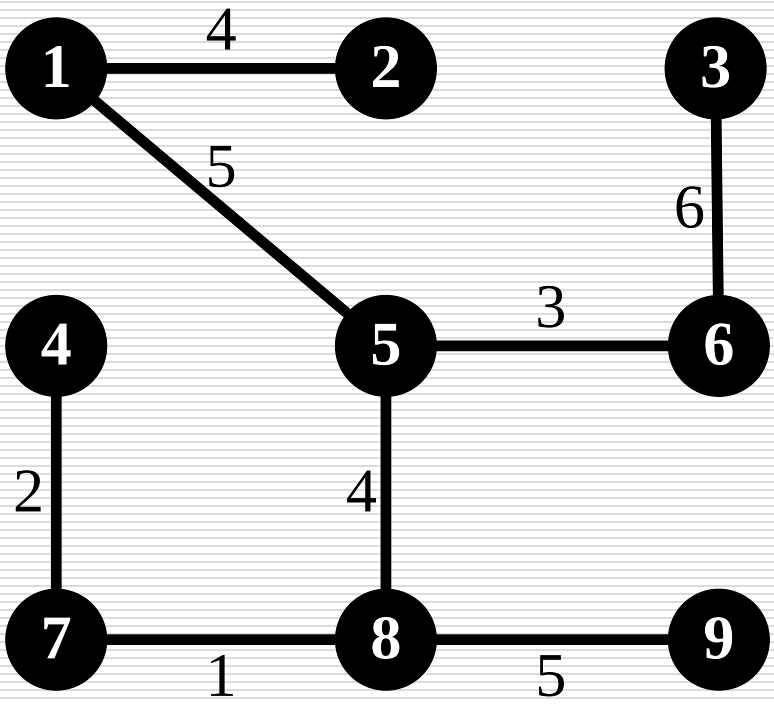
Kruskal Algorithm: example



Kruskal Algorithm: example



Kruskal Algorithm: example



Kruskal Algorithm

- The edges are sorted in ascending order of cost. At each step, the edge with the minimum cost is selected. If it does not form a cycle in the forest constructed so far, it is added to the forest; otherwise, it is discarded.

- For efficient implementation, the existence of a cycle is checked using set operations (UNION-FIND), which requires only the use of Union by Size/Rank.

- **Complexity**: **O(|E|log|V|)**

Common idea: Prim-Kruskal

- Let initial graph G=(V, E)
- starting with the graph G'=(V, ∅) which contains all the vertices of G but no edges,
- iteratively connecting any complementary subsets of vertices S and V\S that currently have no edge between them, using the lightest possible edge from E.
- We eventually obtain a minimum spanning tree

Why the idea works?

Theorem. A set of edges A, which is ***promising*** (i.e., a subset of a Minimum Spanning Tree), remains promising if we add the lightest edge e=(u,v) that connects a connected component V_i of the current subgraph (defined by the vertices V and the edges in A) to the rest of the graph $V \setminus V_i$.

Proof. We consider an MST *T that is a superset of* A (*always exists?*). Suppose that e does not belong to *T*. Then the path p in *T* connecting u and v contains an edge e′ that crosses the cut (V_i, $V \setminus V_i$). It holds that

cost(e) <= cost(e′), therefore:

replacing(e , e′) => ∃ MST *T′* that contains e

Bonus: Boruvka Algorithm

- It operates in rounds. Initially, each vertex is its own component.
- In each round, *every* connected component is joined to one of the other components using the **lightest** possible edge. A method for resolving ties is required.

Complexity: **O(|E| log|V|)** (n each round, the number of components is halved).

It is well-suited for parallel or distributed implementation.

Key Techniques

- **Greedy** algorithms: Gradually "build" the solution from smaller to larger subproblems. At each stage, an irrevocable choice is made, providing the optimal solution for the corresponding subproblem.
  Dijkstra, Prim, Kruskal, Boruvka
- **Dynamic programming**: Gradually "build" the solution by combining optimal solutions of smaller subproblems to derive the optimal solution for larger problems (principle of optimal substructure).
  Bellman-Ford
- Comparison with "**Divide and Conquer**": In Divide and Conquer, subproblems are *independent*, while in Dynamic Programming and Greedy Algorithms, subproblems *overlap* and share solutions.