Graphs

Minimum Spanning Trees

Paolo Camurati and Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

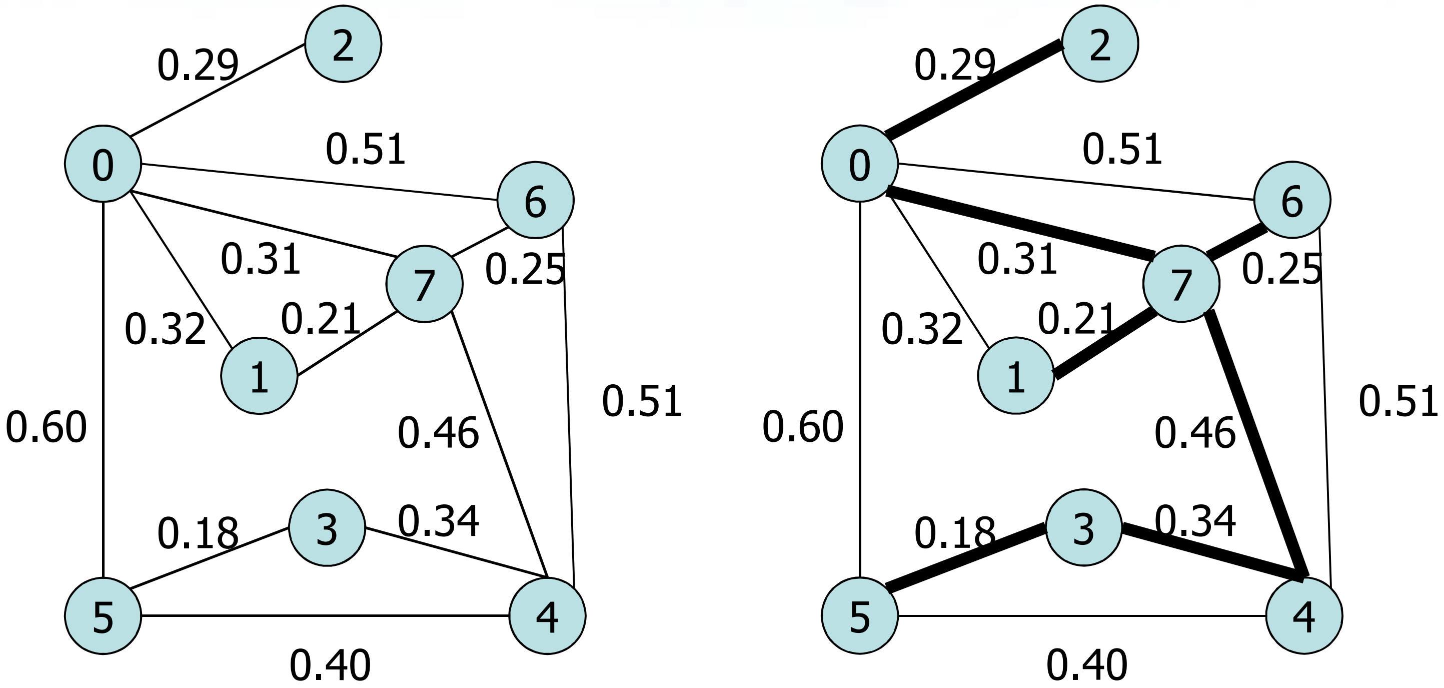
Problem definition

- Example
  - Given an electronic circuit, designers often need to make the pins of several components elettrically equivalent by wiring them togheter
  - To interconnect n pins we can use n-1 connections
  - Of all such arrangements the one that uses the least amount of wire is usually the most desiderable
- Such a problem can be mapped as a Minimum Spanning Tree problem

Minimum Spanning Trees

- Given a graph G=(V,E)
  - Connected
  - Undirected
  - Weighted
    - With a positive real-value weight function w: E→R
- A Minimum-weight Spanning Tree (MST) G' is a graph such that
  - G'=(V, T) with $T \subseteq E$
  - G' is acyclic
  - G' minimizes $w(T)=\Sigma_{(u,v)\in T}\, w(u,v)$

Example



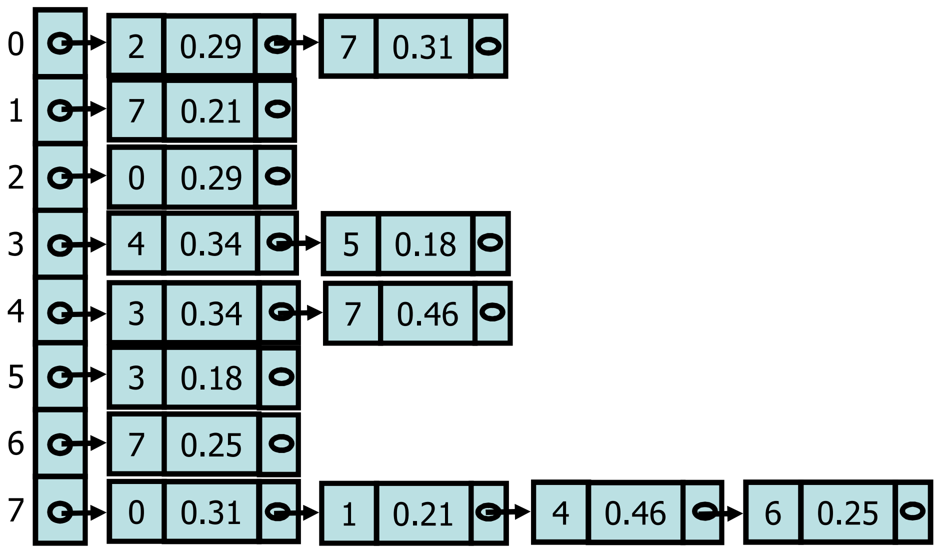
0
1
2
3
4
5
6
7
0.29
0.51
0.31
0.25
0.32
0.21
0.60
0.46
0.51
0.18
0.34
0.40
0
1
2
3
4
5
6
7
0.29
0.51
0.31
0.25
0.32
0.21
0.60
0.46
0.51
0.18
0.34
0.40

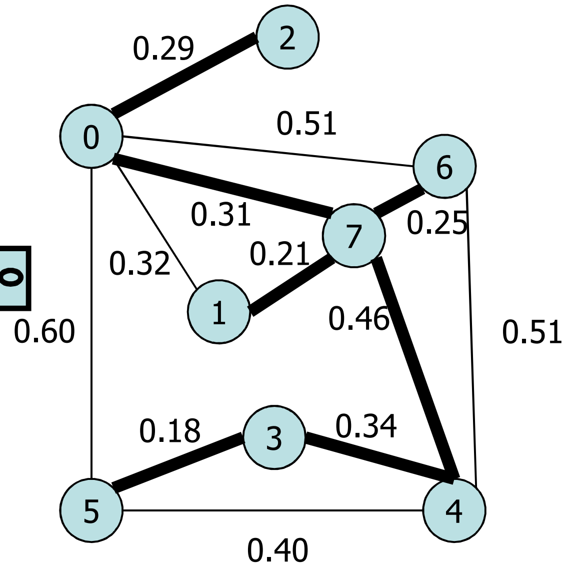
Properties

- MST properties
  - As G’ is acyclic and cover all vertices
    - G’ is a tree
  - The MST is generally not unique
    - It is unique only iff all weights are distinct
  - A MST may be represented as
    - An adjacency matrix or list
    - A list of edges plus weights
    - A list of parents plus weights

Representation

- Adjacency list



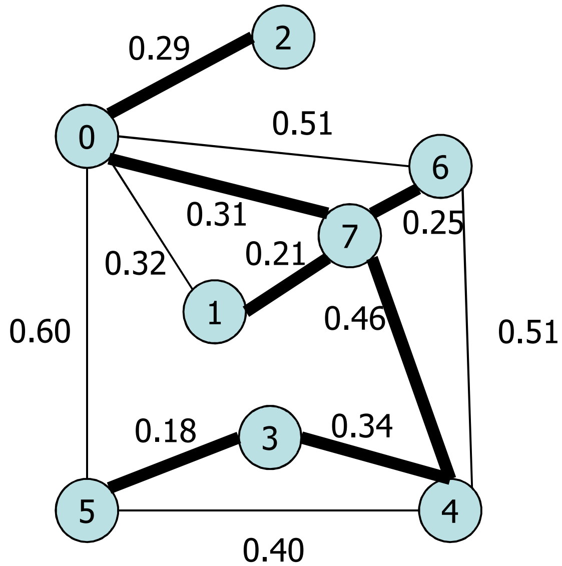


Representation

- List of edges (and weights) stored in a static (or dynamic) array

edge	weight
0-2	0.29
4-3	0.34
5-3	0.18
7-4	0.46
7-0	0.31
7-6	0.25
7-1	0.21

Specifically used for the Kruskal's algorithm

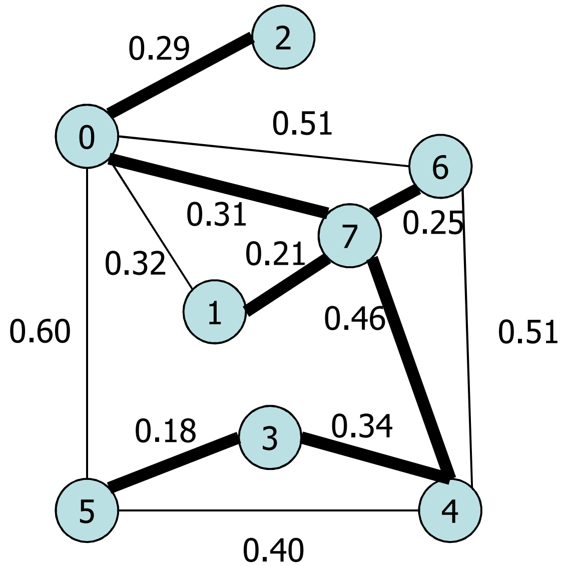


Representation

- List of parents (and weights) stored in a static (or dynamic) array

	parent	weight
0	0	0
1	7	0.21
2	0	0.29
3	4	0.34
4	7	0.46
5	3	0.18
6	7	0.25
7	0	0.31

Specifically used for the Prim's algorithm



Algorithms

- We will analyze two greedy algorithms
  - Greedy algorithms do not generally guarantee globally optimal soluzions, but for the MST they do
- Both algorithms
  - Kruskal's algorithm
  - Prim's algorithm

  are based on a generic method
- The generic method grows a spanning tree by adding one edge at a time

Generic algorithm

Pseudo-code

```
generic_MST (G, w)
  A = ϕ
  while A is not a MST do
    find a safe edge (u,v) for A
    A = A ∪ (u, v)
  return A
```

A is a subset of the MST (initially empty)

While A is not a MST

Add a safe edge (u,v) to A

IFF edge (u,v) is safe, adding (u,v) to a subset A of the MST let A as a subset of the MST

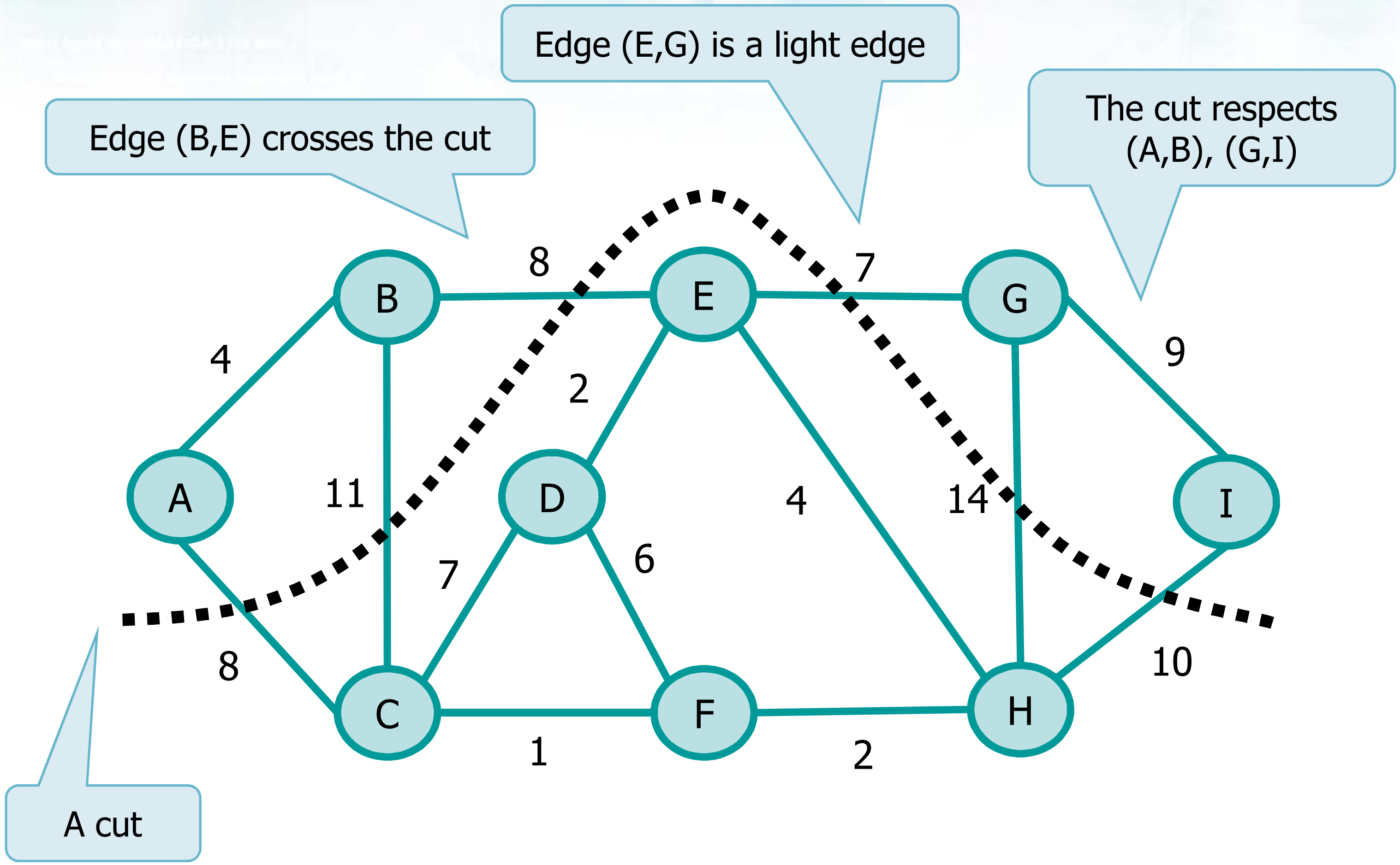
Generic algorithm

- Given a set A
  - Set of edges, i.e., a sub-set of a MST
  - Initially empty
- While A is not a MST
  - Find a **safe edge**
  - Add this edge to A
- Invariant
  - The edge (u,v) is **safe** if and only if added to a sub-set of the MST it produces another sub-set of the MST

Definitions

- G=(V,E) connected, undirected, and weighted
  - Cut
    - A partition of V into S and V-S such that
    - $V = S \cup (V\text{-}S)$ && $S \cap (V\text{-}S) = \varnothing$
  - Crossing edge
    - An edge $(u,v) \in E$ crosses the cut if and only if
    - $u \in S$ && $v \in (V\text{-}S)$ or vice-versa
  - A cut respecting a set of edges
    - A cut respect a set A of edges if no edge of A crosses the cut
  - A light edge
    - An edge if a light edge if its weight is minimum among the edges crossing the cut

Example


Edge (E,G) is a light edge
Edge (B,E) crosses the cut
The cut respects (A,B), (G,I)
A cut
A
B
C
D
E
F
G
H
I
4
8
11
8
7
2
6
1
4
7
14
2
9
10

Safe Edges: Theorem

- G=(V,E) connected, undirected, and weighted
- Let
  - A be a subset of E including a MST
    - Initially A is empty
  - (S, V-S) be any cut of G that respects A
  - (u, v) be a light edge crossing the cut (S, V-S)
- Then
  - Edge (u,v) is **safe** for A

Prim's Algorithm

- Known as DJP algorithm, Jarnik's algorithm, Prim-Jarnik algorithm, Prim-Dijkstra algorithm
  - Developed in 1930 by Vojtech Jarnik
  - Rediscovered in 1957 by Robert Prim
  - Redicoverd 1959 by Edsger Dijkstra
- Based on the generic algorithm
- Use the theorem to select the safe edge

Implementation

Pseudo-code

Source = starting vertex

```
mst_Prim (G, w, source)
  for each v ∈ V
    v.key = ∞
    v.pred = NULL
  source.key = 0
  Q = V
  while Q ≠ ϕ
    u = extract_min (Q)
    for each v ∈ adjacency list of u
      if v ∈ Q and w(u,v)<v.key
        v.pred = u
        v.key = w(u,v)
```

v.key is the minimum weight of any edge connecting v to a vertex in the tree

v.Pred is the vertex parent

Extract the vertex from Q and insert it in the MST

Update the key and pred fields of all adjacency nodes

Implementation

Pseudo-code

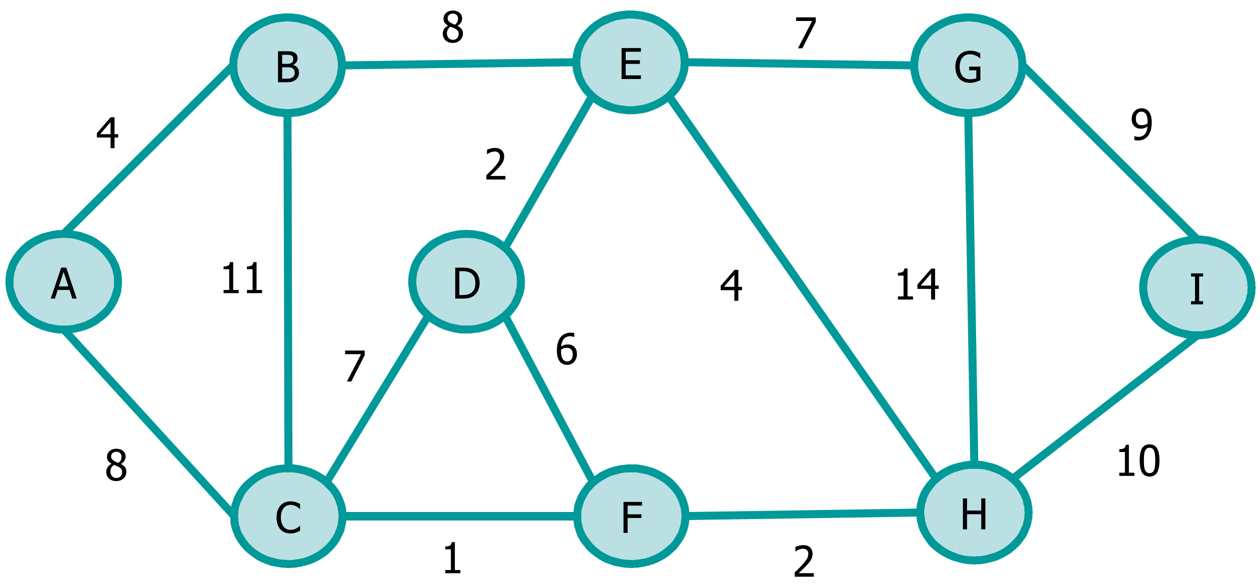
```
mst_Prim (G, w, source)
  for each v ∈ V
    v.key = ∞
    v.pred = NULL
  source.key = 0
  Q = V
  while Q ≠ ϕ
    u = extract_min (Q)
    for each v ∈ adjacency list of u
      if v ∈ Q and w(u,v)<v.key
        v.pred = u
        v.key = w(u,v)
```

End when all vertices belong to the same tree

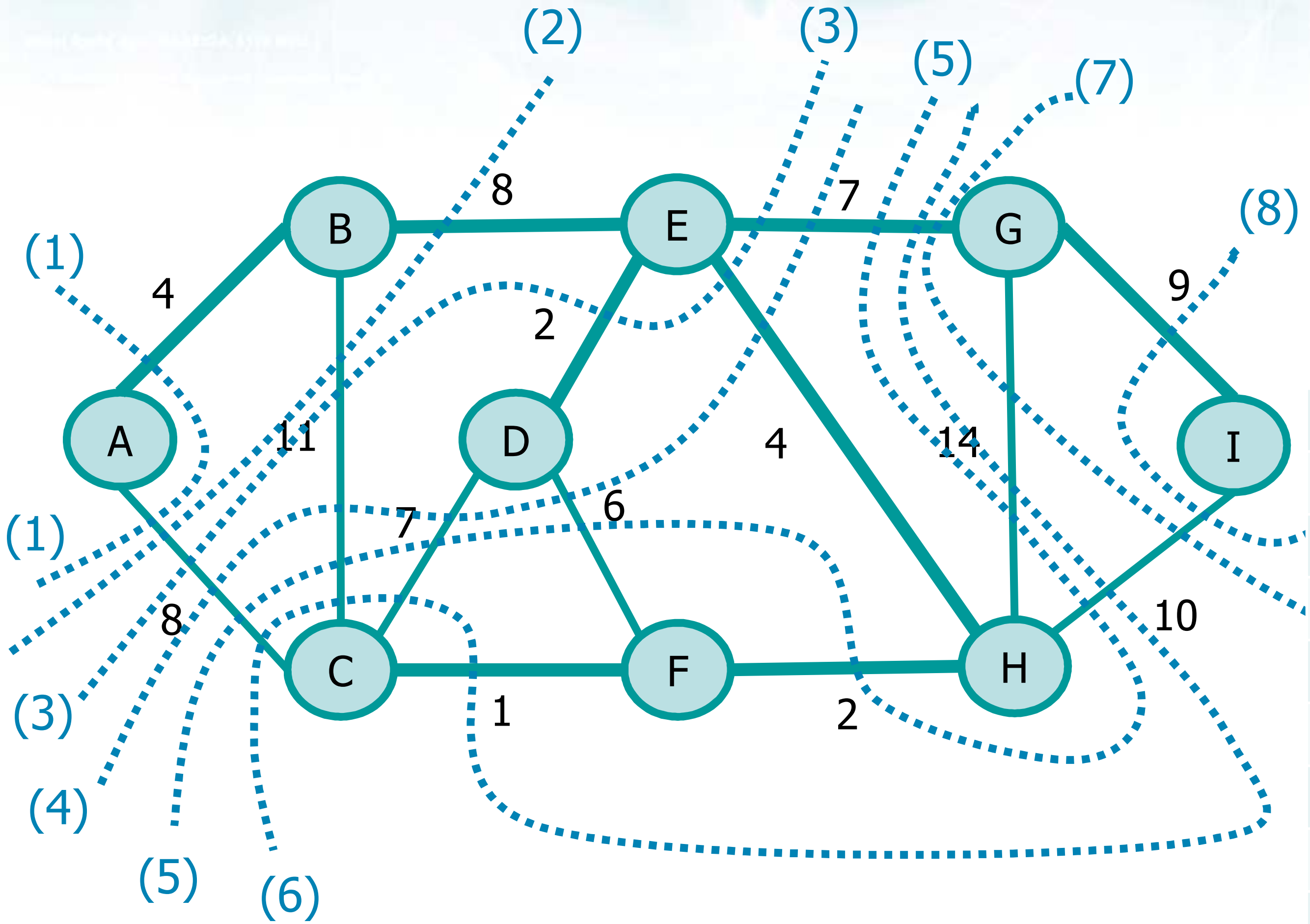
Select all edges crossing the cut
Among those, select the edge with minimun weight and add it to A

Adjust S and the set of edges crossing the cut depending on the selected edge

Example



Example



AB	4
BE	8
DE	2
EH	4
FH	2
CF	1
EG	7
GI	9
	37

Complexity

```
mst_Prim (G, w, source)
  for each v ∈ V
    v.key = ∞
    v.pred = NULL
  source.key = 0
  Q = V
  while Q ≠ ϕ
    u = extract_min (Q)
    for each v ∈ adjacency list of u
      if v ∈ Q and w(u,v)<v.key
        v.pred = u
        v.key = w(u,v)
```

O(|V|)

Executed |V| times

O(lg |V|) → **O(|V| log|V|)**

Executed |E| times altogether

O(lg |V|) → **O(|E| log|V|)**

Overall running time complexity
T(n) = O(|V| · log|V|+|E| · lg |V|)

Complexity

- In general
  - $T(n) = O(|V| \cdot \lg |V| + |E| \cdot \lg |V|)$

  that is
  - $T(n) = O(|E| \cdot \lg |V|)$
- With an efficient data structure, such as a Fibonacci-Heap the running time can be improved to
  - $T(n) = O(|E| + |V| \cdot \lg |V|)$

Safe Edges: Corollary

- G=(V,E) connected, undirected, and weighted
- Let A be such that
  - A is a subset of E that is included in a MST
    - Initially A is empty
  - C is a tree in the forest G_A= (V, A)
  - (u,v) is a light edge connecting C to another component of G_A
- Then
  - Edge (u,v) is **safe** for A

Kruskal's Algorithm

- Algorithm proposed by Joseph Kruskal in 1956
- Based on the generic algorithm
- Use the corollary to select the safe edge
  - Forest of tree, initially single vertices
  - Sort edges into nondecreasing order by weigth w
  - Iteration
    - Select a safe edge, i.e., an edge with minimum weight connecting two trees and generating one single tree (Union-Find)
  - End
    - All vertices belong to the same tree

Implementation

Pseudo-code

```
mst_Kruskal (G, w)
  A = ϕ
  for each vertex v ∈ V
    make_set (v)
sort E into non-decreasing order by weight w
for each edge (u,v)∈E
  if find (u) ≠ find (v)
    A = A ∪ (u,v)
    union (u,v)
return A
```

A is initially the empty set

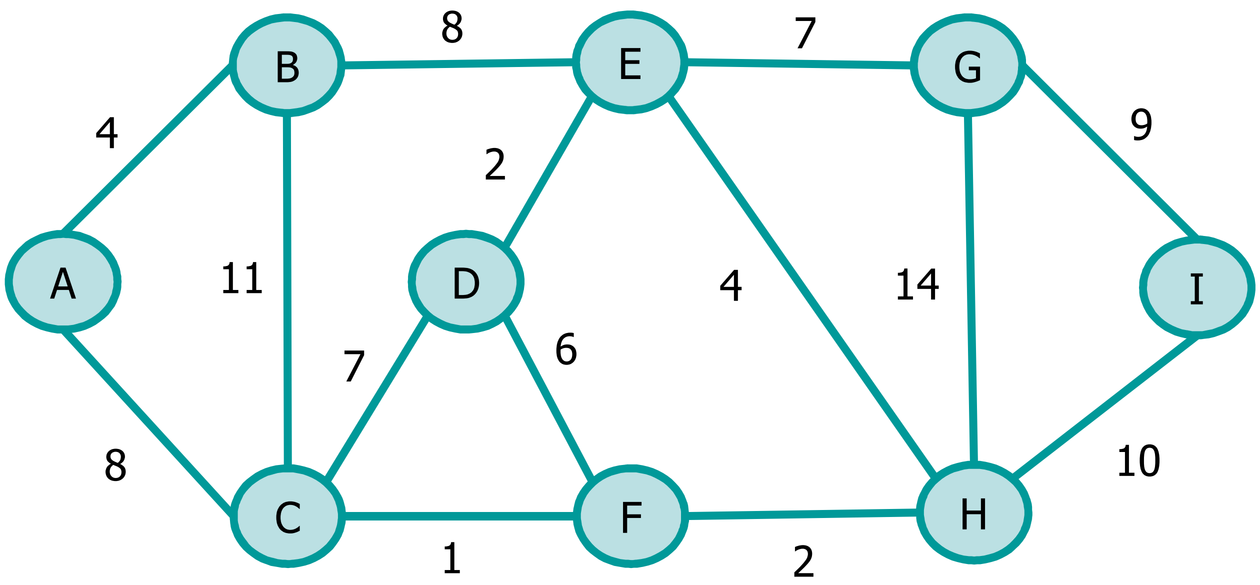
For each v create a set

taken in nondecreasing order by weight

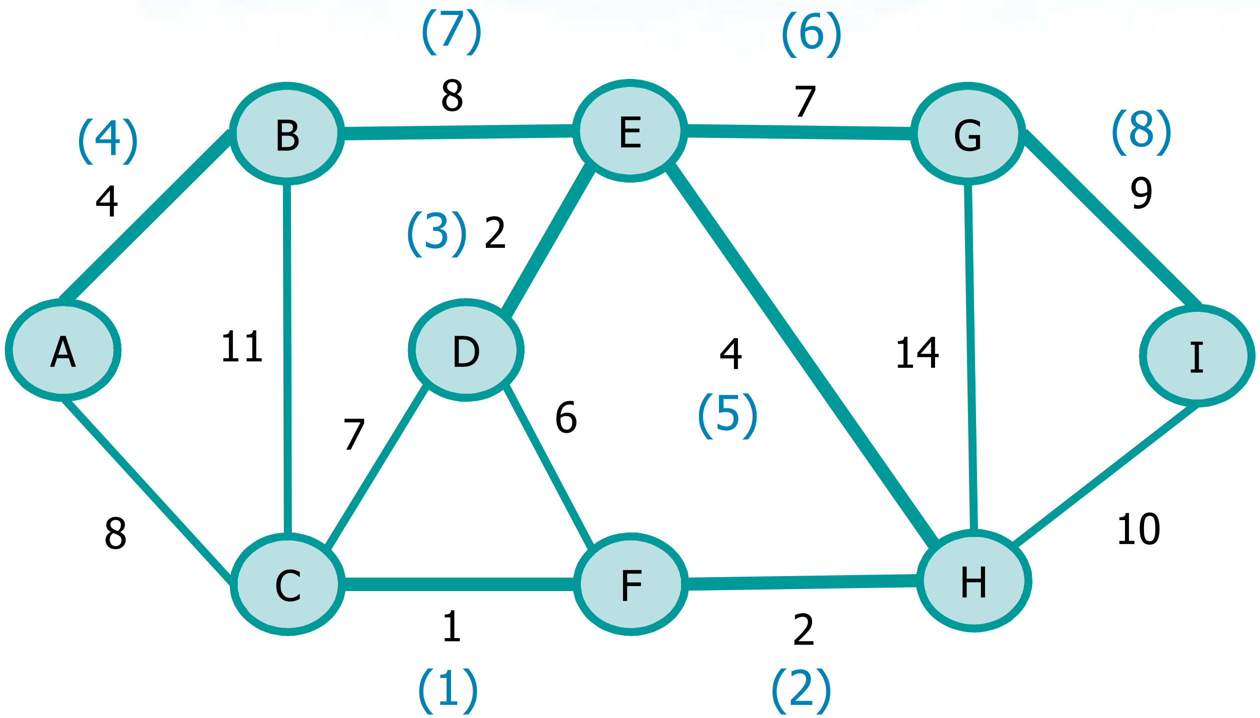
Find representative of u and v

Union set

Example



Example



CF	1
FH	2
DE	2
AB	4
EH	4
EG	7
BE	8
GI	9
	37

Complexity

```
mst_Kruskal (G, w)
  A = ϕ
  for each vertex v ∈ V
    make_set (v)
sort E into non-decreasing order by weight w
for each edge (u,v)∈E
  if find (u) ≠ find (v)
    A = A ∪ (u,v)
    union (u,v)
return A
```

O(1)

Executed V times

O(1) → **O(|V|)**

O(|E| lg |E|)

Executed E times

Union and find takes O(lg |E|)
→ **O(E log |E|)**

Overall running time complexity
T(n) = O(|E| · lg |E|)

Complexity

- In general
  - T(n) = (|E| · lg |E|)
- Asintotically, for dense graph, Prim is more efficient than Kruskal
  - Prim
    - T(n) = (|E| + |V|· lg |V|)
  - Kruskal
    - T(n) = (|E| · lg |E|)

For dense graph

$$E = \frac{|V| \cdot (|V| - 1)}{2}$$

then

|E| > |V|