Introduction to Algorithms

Lecture 10

Today's Topics

- Optimization Problems
- Greedy Algorithms
- Graph Representation & Algorithms
- Minimum Spanning Tree
  - Prim's Algorithm
  - Kruskal's Algorithm

Optimization Problems

- A problem that may have many feasible solutions.
- Each solution has a value
- In maximization problem, we wish to find a solution to maximize the value
- In the minimization problem, we wish to find a solution to minimize the value

Greedy Algorithms

- Many optimization problems can be solved more quickly using a greedy approach
  - The basic principle is that local optimal decisions may be used to build an optimal solution
  - But the greedy approach may not always lead to an optimal solution overall for all problems
  - The key is knowing which problems will work with this approach and which will not

Greedy algorithms

- A *greedy algorithm* always makes the choice that looks best at the moment
  - The hope: a locally optimal choice will lead to a globally optimal solution
  - For some problems, it works
- greedy algorithms tend to be easier to code

Graph Representation

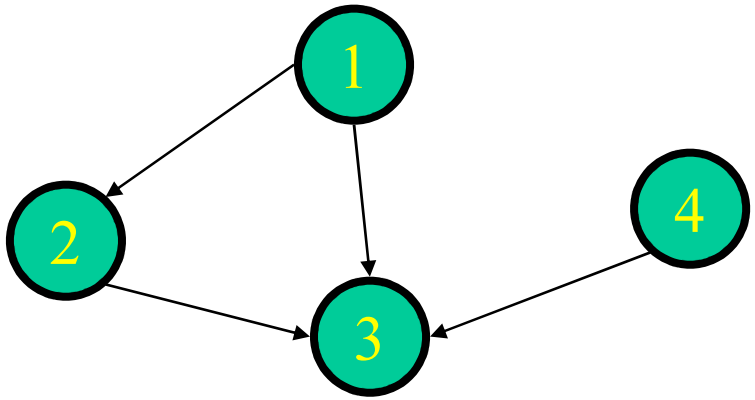
Graph $G = (V, E)$

V = set of vertices

E = set of edges = subset of $V \times V$

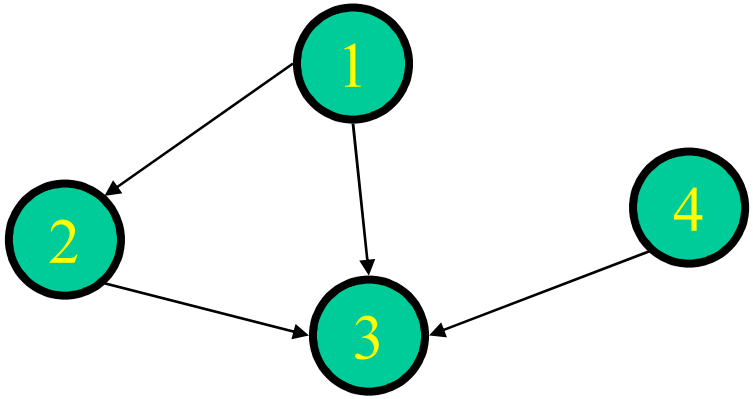
If G is connected, then

$|E| \geq |V| - 1 \Rightarrow \lg|E| = \Theta(\lg V)$



Graph Representation

Graph can be *directed* or *undirected*



Assume vertices $V = \{1,2,\ldots,n\}$

Define "Adjacency matrix" $A[1..n, 1..n]$

$$A[i,j] = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{if } (i,j) \notin E \end{cases}$$

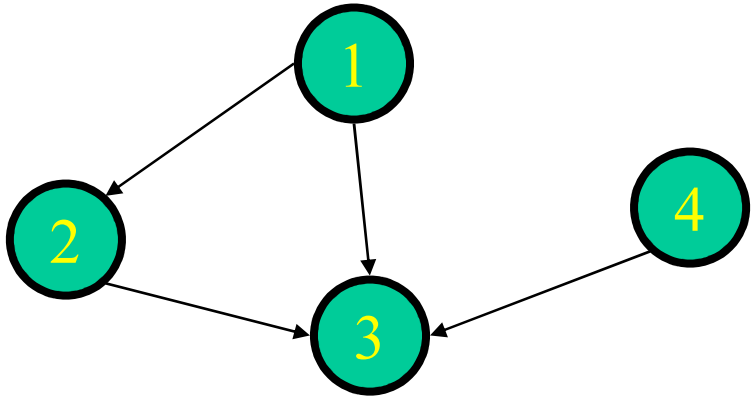
Graph Representation

	j=1	2	3	4
i=1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

Adjacency matrix:

$\Theta(V^2)$ storage $\rightarrow$ **dense** representation

Sparse Graph Representation



Adjacency list: $\Theta(V+E)$ storage $\rightarrow$ **sparse** representation

For each vertex v, keep a list Adj[v] of vertices adjacent to v.

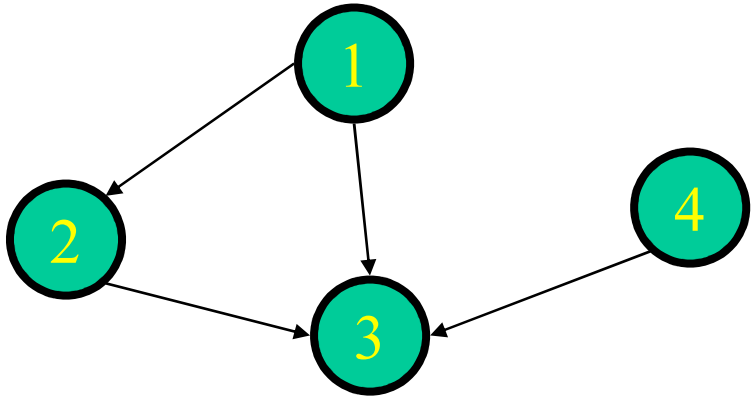
Sparse Graph Representation

Adj[1] = {2, 3}

Adj[2] = {3}

Adj[3] = {}

Adj[4] ={3}



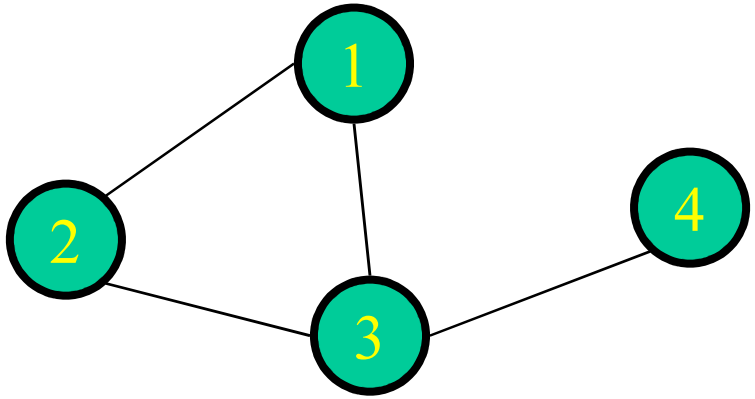
Adj[*v*] = degree(*v*) [for undirected graphs]

Adj[*v*] = out-degree(*v*) [for digraphs]

Handshaking Lemma

For undirected graphs,

$$\sum_{v \in V} \deg(v) = 2 \mid E \mid$$

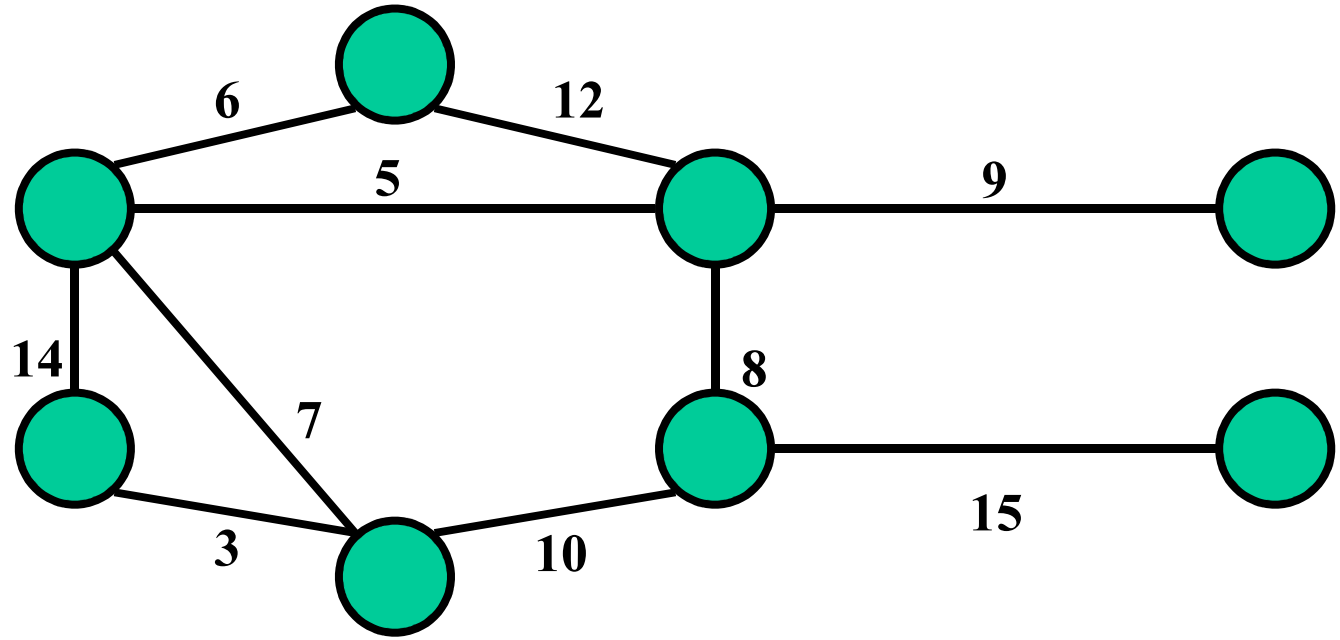


Thus adjacency list uses $\Theta(V+E)$ storage.

Spanning Tree

Undirected, connected graph $G = (V, E)$

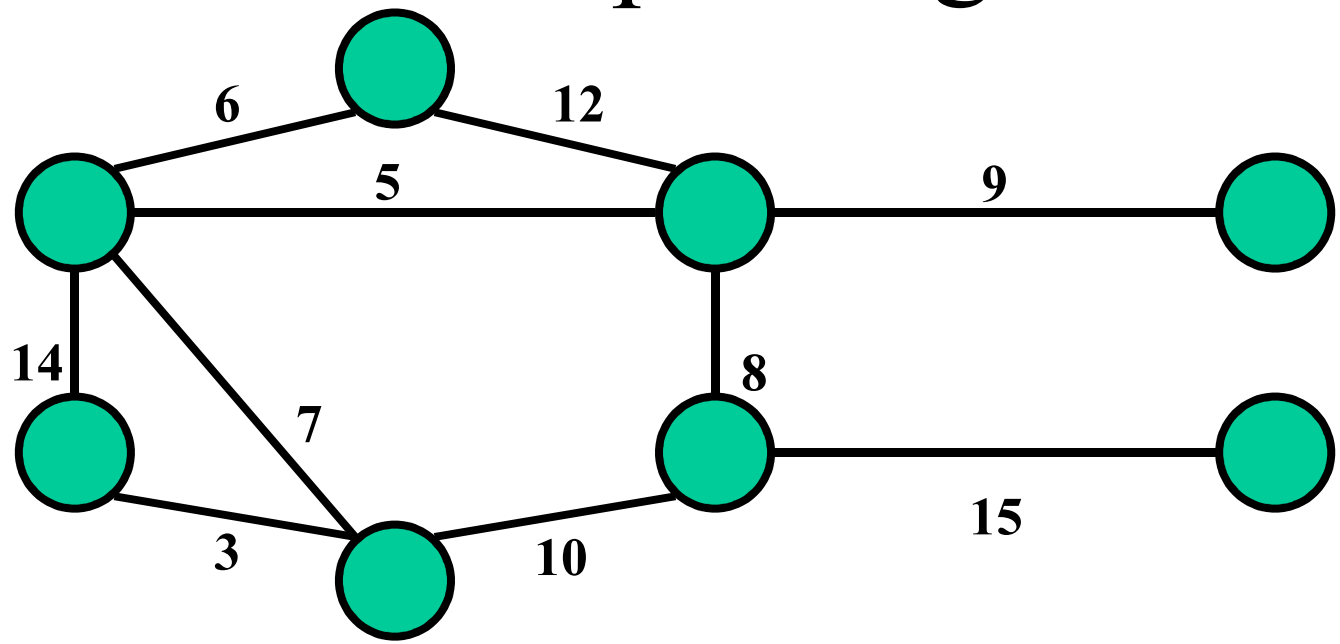
Weight function $W : E \rightarrow \mathcal{R}$



Spanning Tree: Tree that connects all vertices.

What is ST of the above graph?

Minimum Spanning Tree



Minimum Spanning Tree:

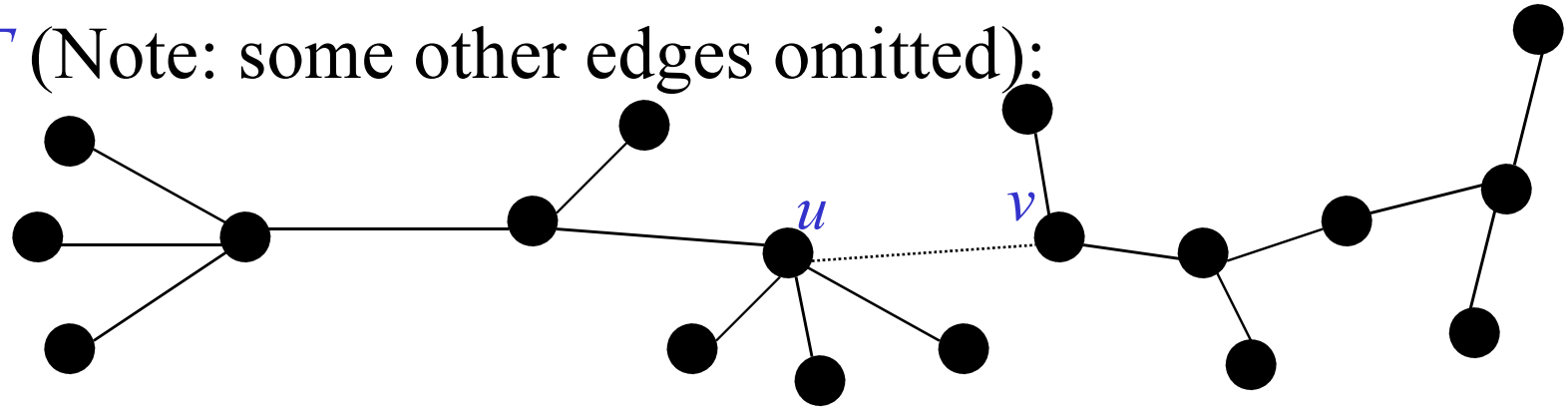
Tree that connects all vertices, and minimizes

$$w(T) = \sum_{(u,v)\in E} w(u,v)$$

What is MST of above graph?

Optimal Substructure

T (Note: some other edges omitted):



Removing (u, v) partitions T into T_1 and T_2.

Claim: T_1 is MST of $G_1 = (V_1, E_1)$, the subgraph of G induced by vertices in T_1.

V_1 = vertices in T_1

$E_1 = \{(x, y) \in E : x, y \in V_1\}$

T_2 is an MST of G_2.

Optimal Substructure

Proof: $w(T) = w(u,v) + w(T_1) + w(T_2)$

(There can't be a better tree than T_1 or T_2, or T would be suboptimal)

(Overlapping subproblems? Dynamic Programming? Yes, but…)

Greedy Choice

Greedy choice property:

Locally optimal (greedy) choice

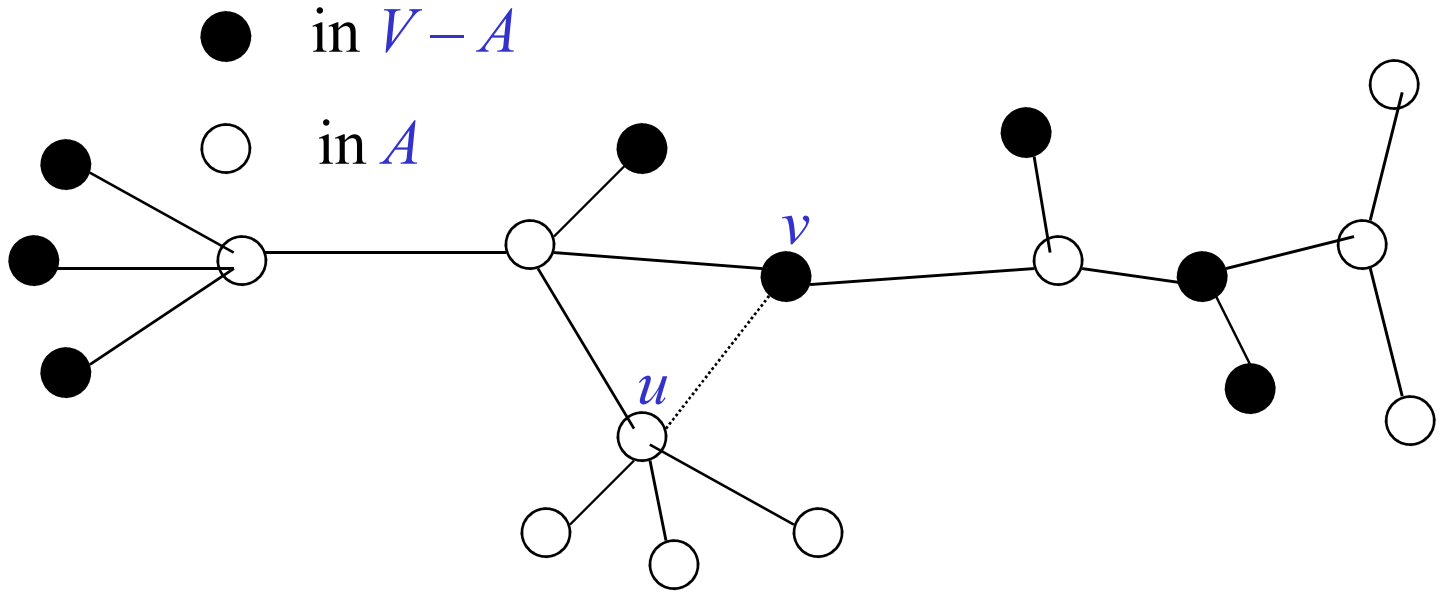
yields a globally optimal solution!

Theorem: Let T be MST of G, and let $A \subseteq V$.
Let (u, v) be min weight edge in G
connecting to $V - A$.

Then, $(u, v) \in T$.

Proof: "cut and paste"

Greedy Choice



Suppose $(u, v) \notin T$

Look at path from u to v in T. Swap (u, v) with first edge on path from u to v in T that crosses from A to $V - A$. This improves T!

Prim's Algorithm

Keep $V - A$ in priority queue Q, sorted by weight of lightest edge connecting to A

$Q \leftarrow V$

$key[v] \leftarrow \infty, \forall v \in V$

$key[s] \leftarrow 0$, for arbitrary $s \in V$

while $Q \neq \Phi$

  do $u \leftarrow Extract\text{-}Min(Q)$

    for each $v \in Adj[u]$

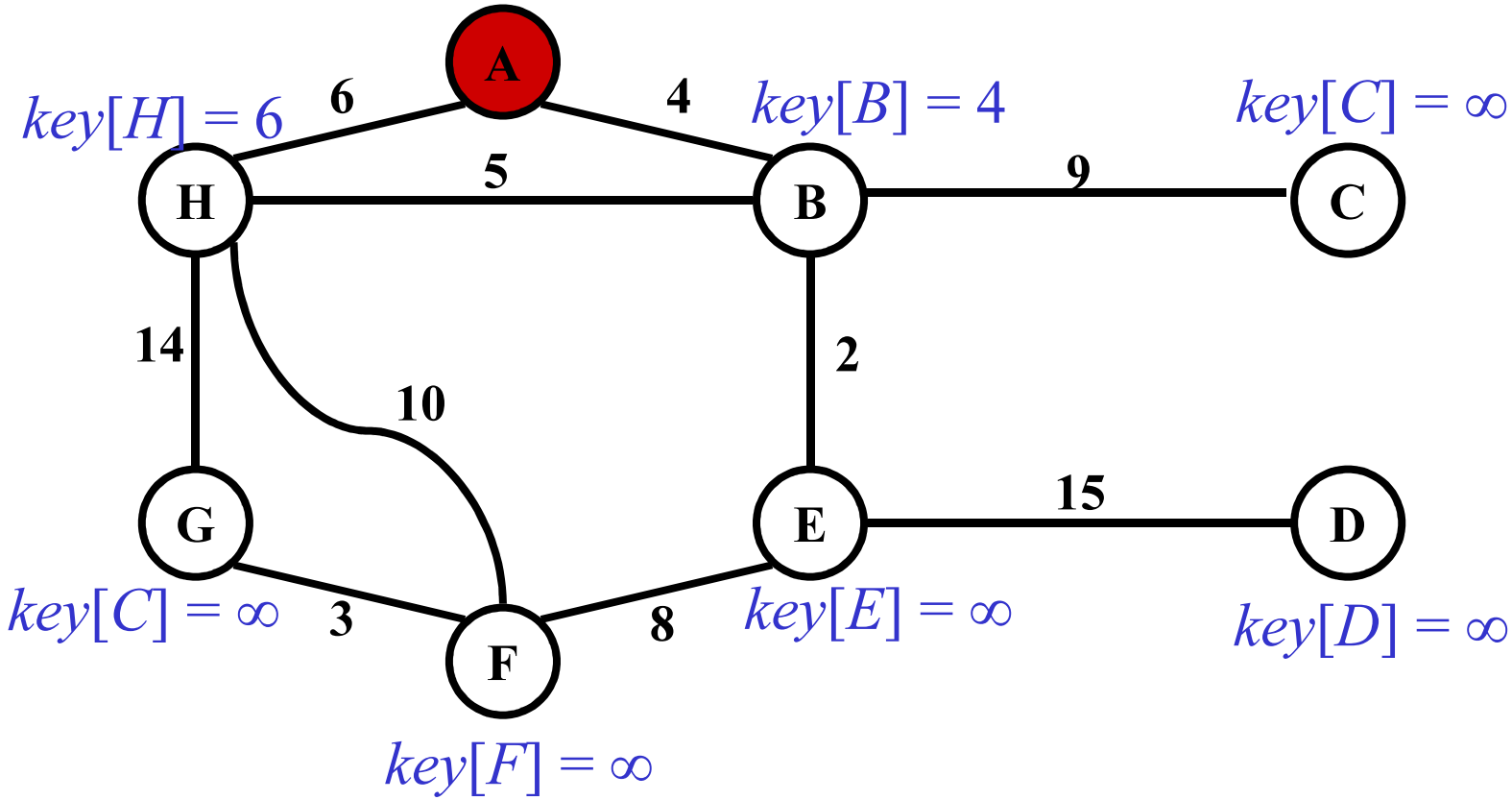
      do if $v \in Q$ and $w(u, v) \leq key[v]$

        then $key[v] \leftarrow w(u, v)$

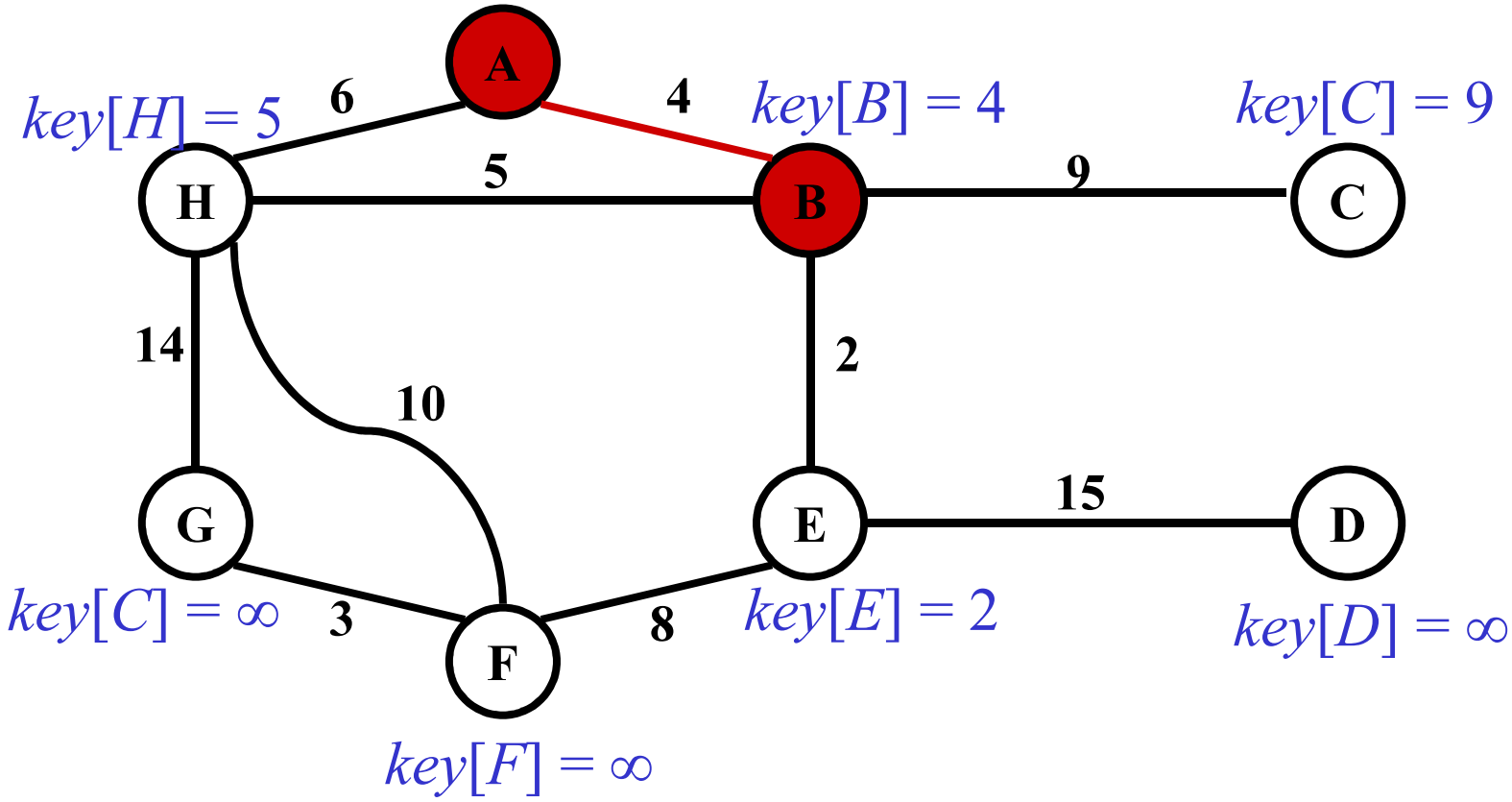
          $\pi[v] \leftarrow u$

At end, $\{(v, \pi[v])\}$ forms MST.

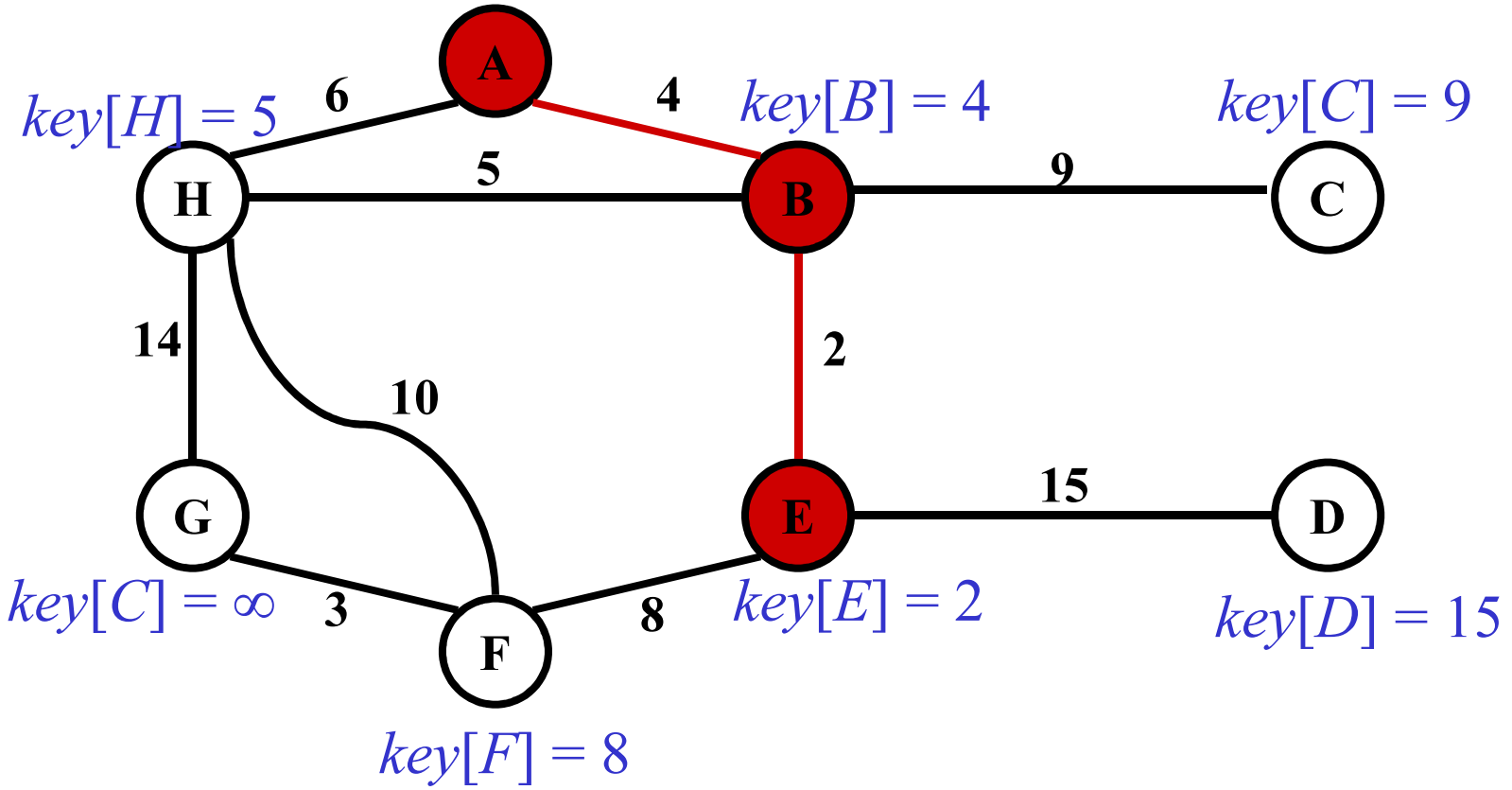
Example


A
H
B
C
G
E
D
F
6
4
5
9
14
2
10
15
3
8
key[H] = 6
key[B] = 4
key[C] = ∞
key[C] = ∞
key[E] = ∞
key[D] = ∞
key[F] = ∞

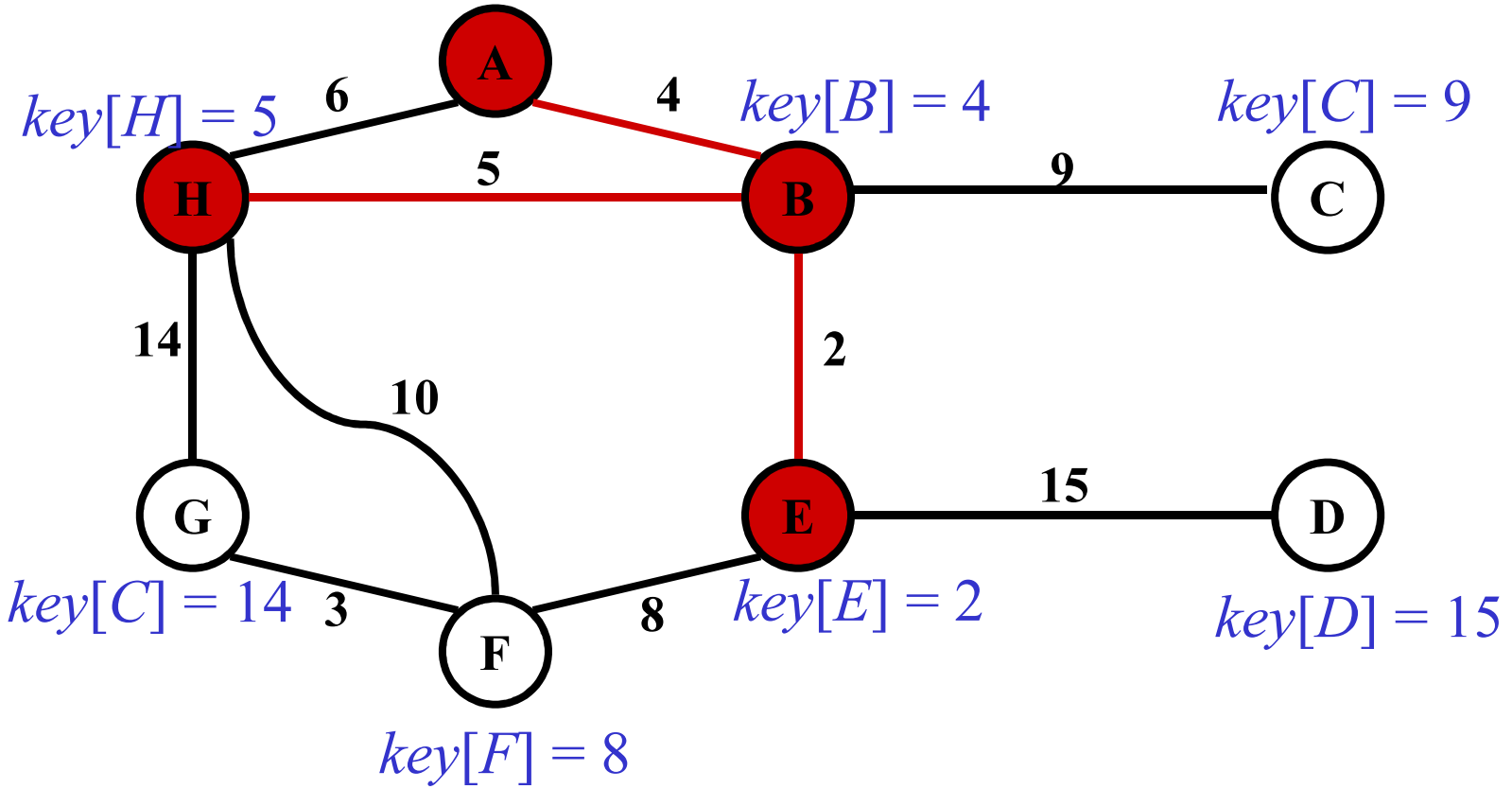
Example


A
H
B
C
G
E
D
F
6
4
5
9
14
10
2
15
3
8
key[H] = 5
key[B] = 4
key[C] = 9
key[C] = ∞
key[E] = 2
key[D] = ∞
key[F] = ∞

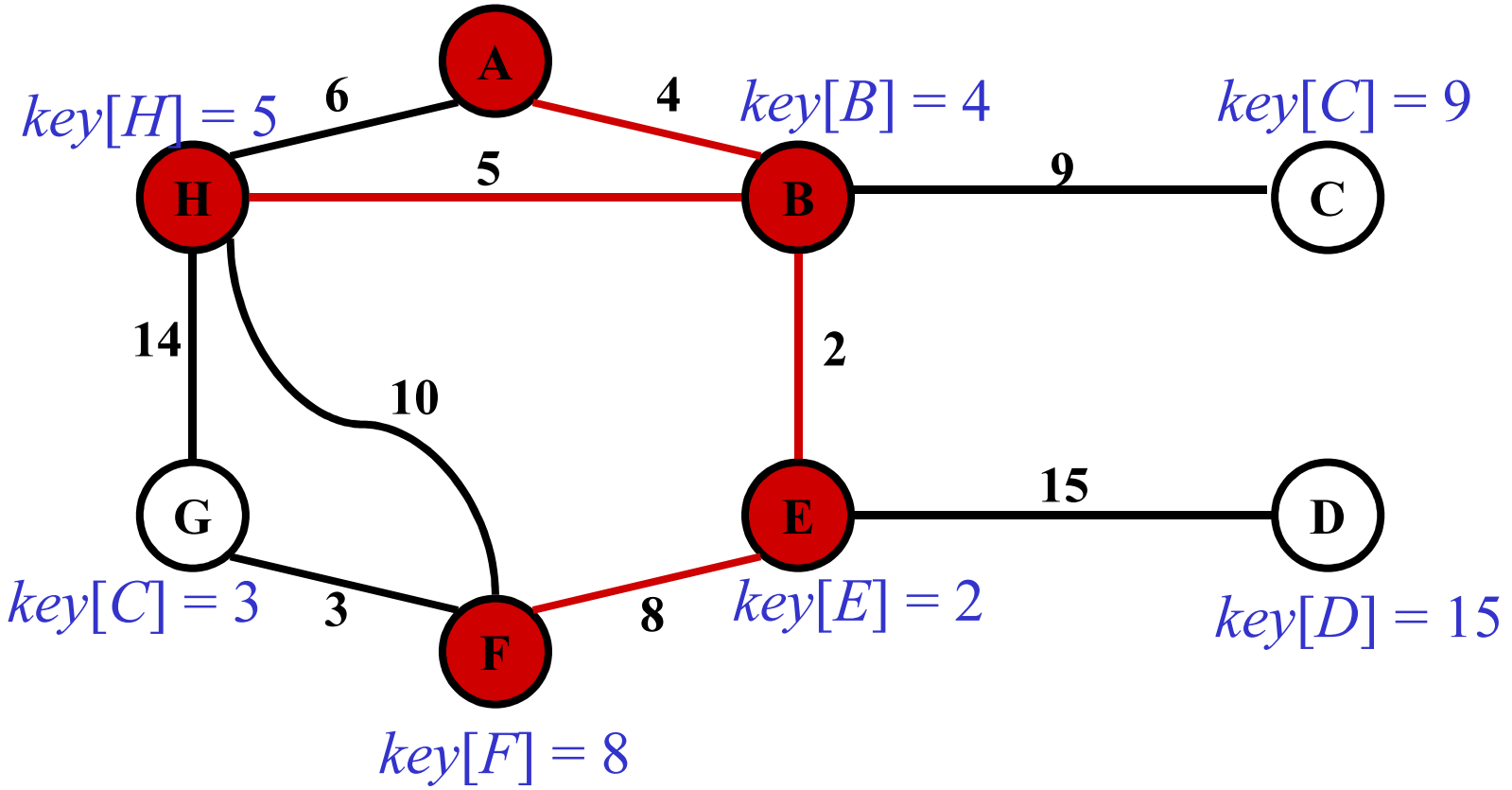
Example


A
H
B
C
G
E
D
F
6
4
5
9
14
2
10
15
3
8
key[H] = 5
key[B] = 4
key[C] = 9
key[C] = ∞
key[E] = 2
key[D] = 15
key[F] = 8

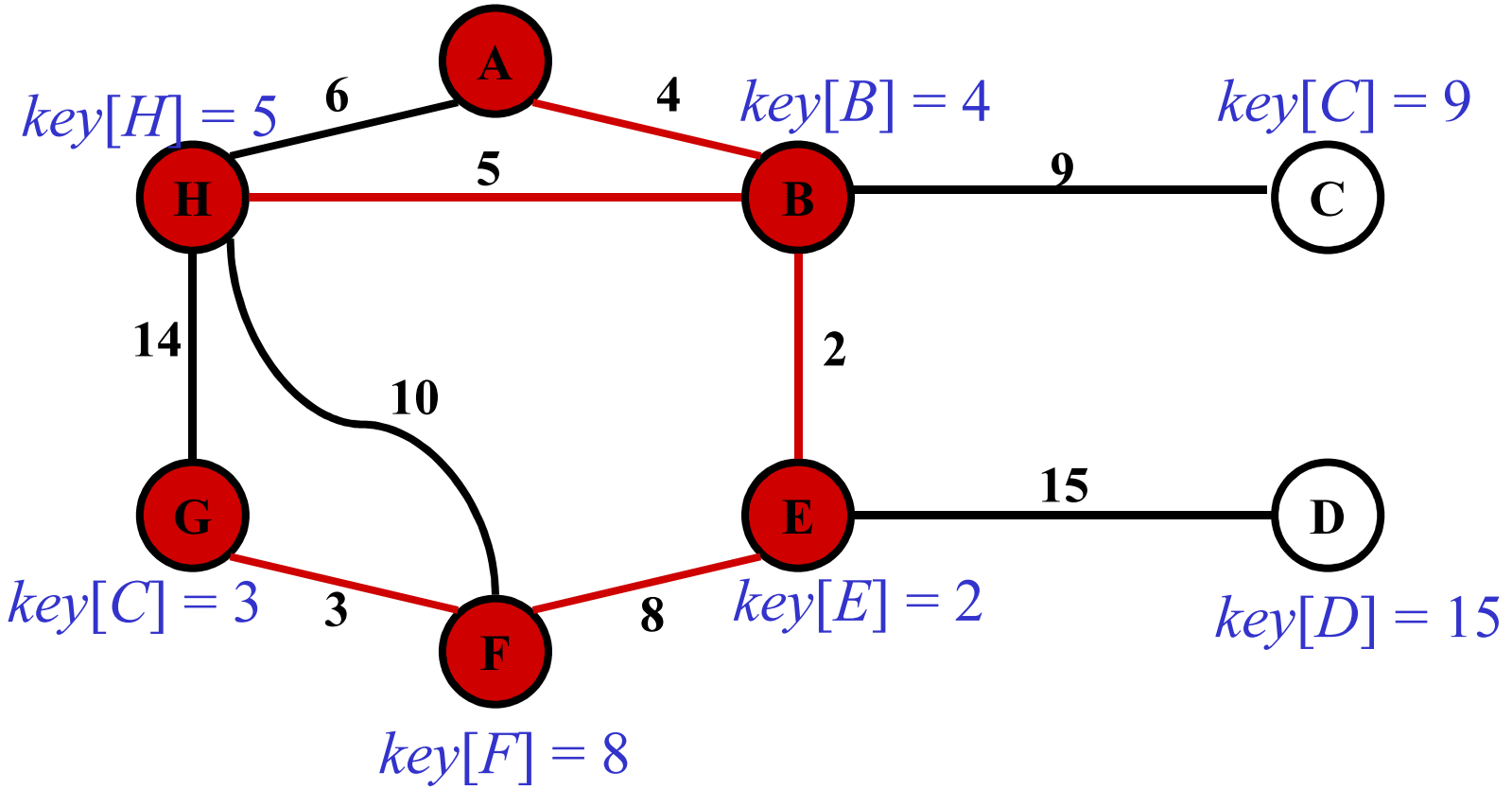
Example


A
H
B
C
G
E
D
F
6
4
5
9
14
10
2
15
3
8
key[H] = 5
key[B] = 4
key[C] = 9
key[C] = 14
key[E] = 2
key[D] = 15
key[F] = 8

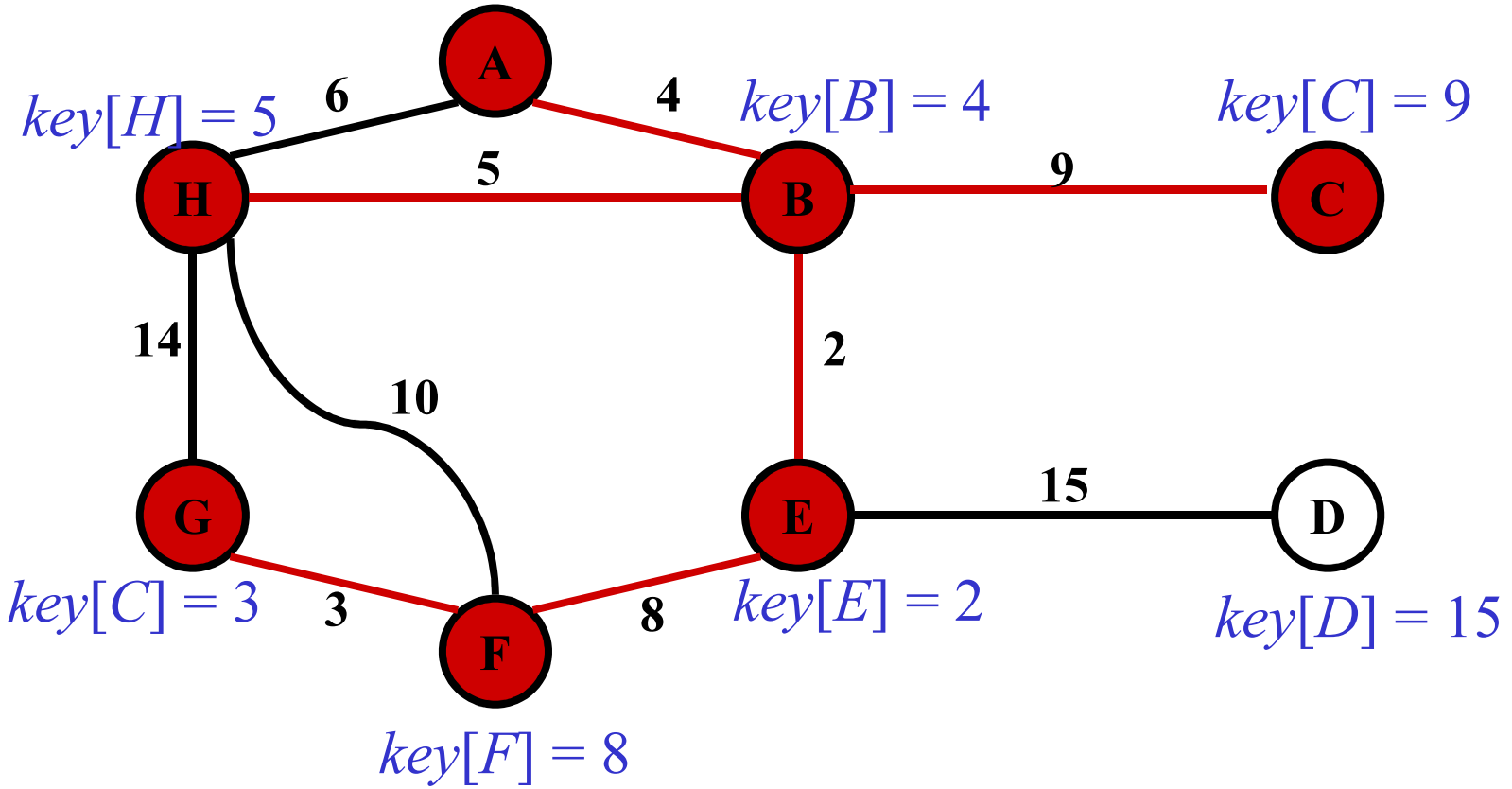
Example


A
H
B
C
G
E
D
F
6
4
5
9
14
10
2
15
3
8
key[H] = 5
key[B] = 4
key[C] = 9
key[C] = 3
key[E] = 2
key[D] = 15
key[F] = 8

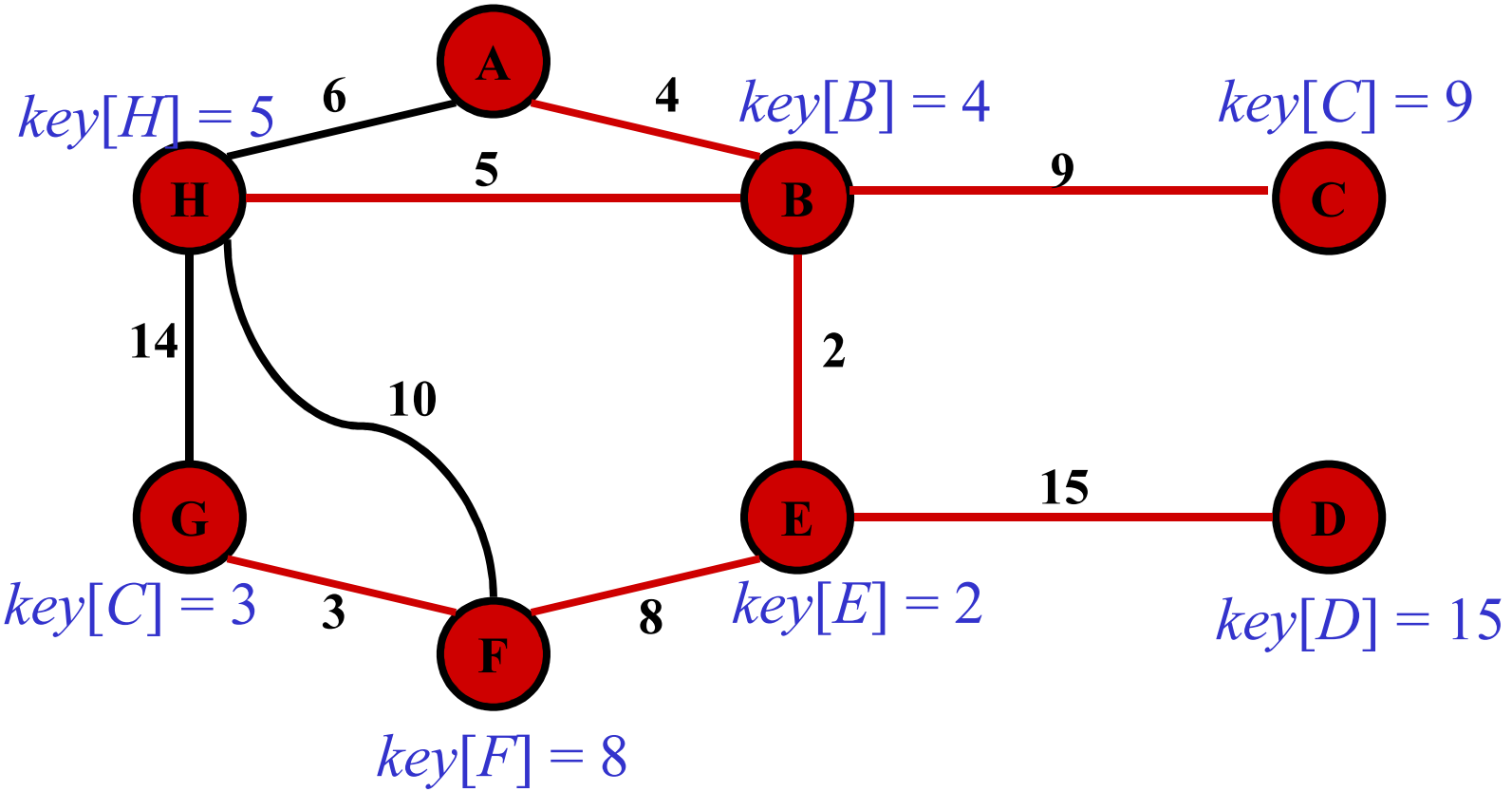
Example


A
H
B
C
G
E
D
F
key[H] = 5
6
4
key[B] = 4
key[C] = 9
5
9
14
2
10
15
key[C] = 3
3
8
key[E] = 2
key[D] = 15
key[F] = 8

Example


A
H
B
C
G
E
D
F
key[H] = 5
key[B] = 4
key[C] = 9
key[C] = 3
key[E] = 2
key[D] = 15
key[F] = 8
6
4
5
9
14
2
10
15
3
8

Example


A
H
B
C
G
E
D
F
key[H] = 5
key[B] = 4
key[C] = 9
key[C] = 3
key[E] = 2
key[D] = 15
key[F] = 8
6
4
5
9
14
2
10
15
3
8

Complexity of Prim

$$\text{Time} = |V| \times T(\textit{Extract-Min}) + \Theta(E) \times T(\textit{Decrease-Key})$$

Q	T(*Extract-Min*)	T(Descreas-Key)	
array	$\Theta(V)$	$\Theta(1)$	1
binary heap	$\Theta(\lg V)$	$\Theta(\lg V)$	1
Fibonacci heap	$\Theta(\lg V)$	$\Theta(1)$	$\Theta(V \lg V + E)$

Kruskal's algorithm for MST

Disjoint-set data structure

Sets $S = \{S_i\}$, S_i intersects S_j = empty set

Operations:

- Insert(x): $S \leftarrow \S \cup \{\{x\}\}$
- Union(S_i, S_j): $S \leftarrow S - \{S_i, S_j\} \cup \{S_i \cup S_j\}$
- FindSet(x): return unique $S_i \in S$ where $x \in S_i$

Kruskal's algorithm for MST

$T \leftarrow$ empty set

for each $v \in V$

  do Insert(v)

Sort E by edge weight

for each edge $(u, v) \in E$

  do if $FindSet(u) \neq FindSet(v)$

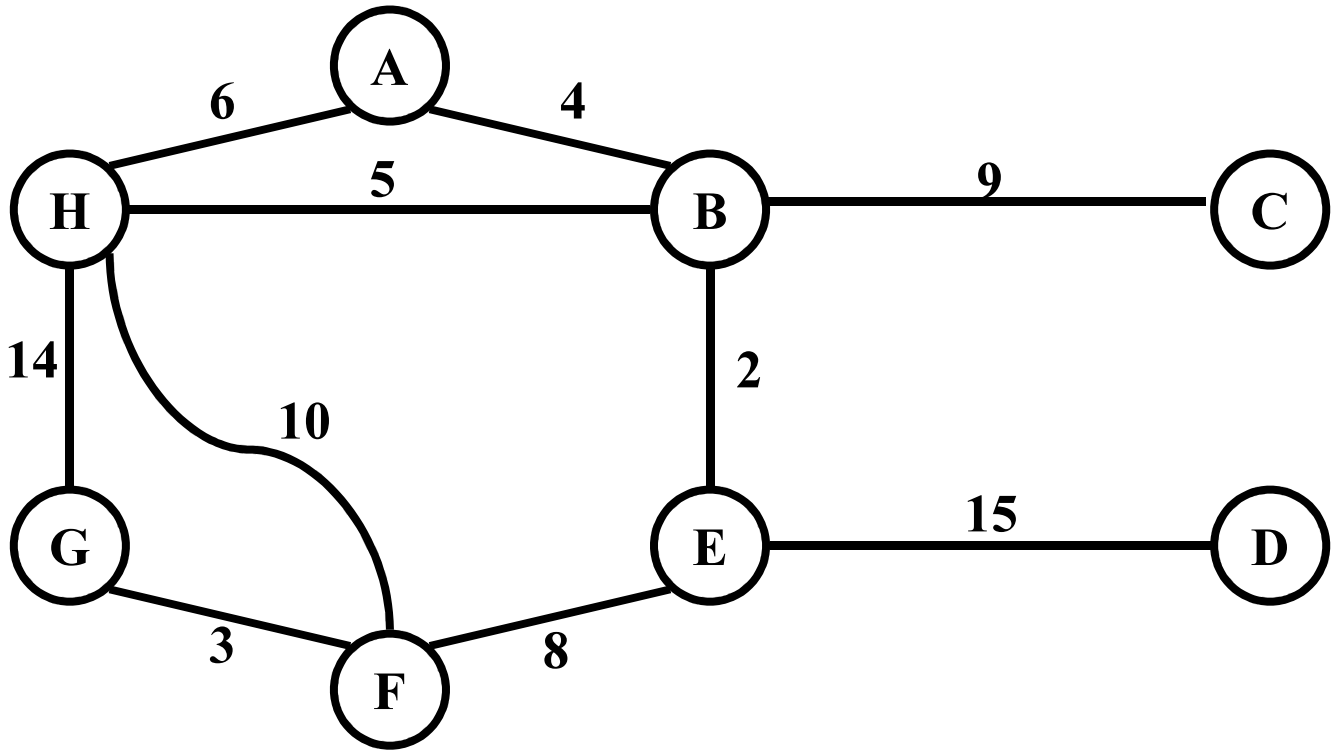
    then $T \leftarrow T \cup \{(u, v)\}$

      $Union(FindSet(u), FindSet(v))$

That is adds cheapest edge that connects two trees of "forest".

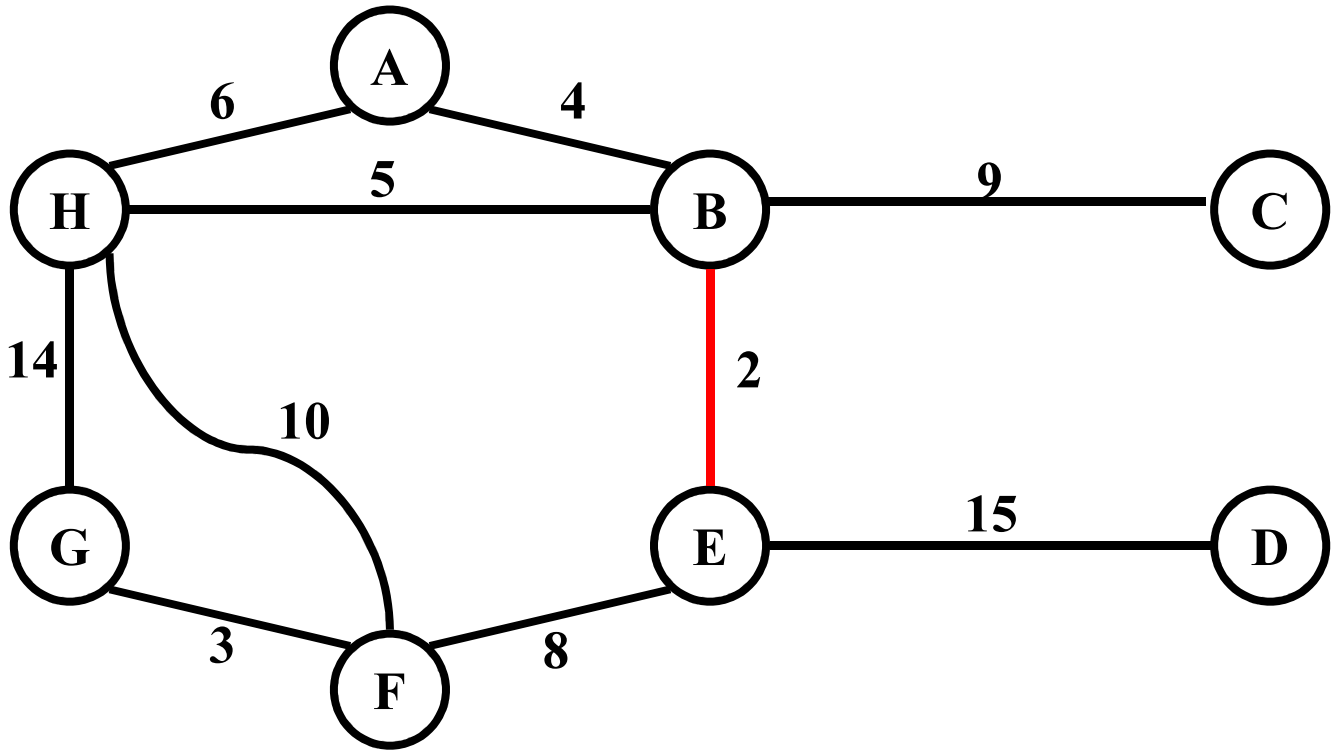
Why is this algorithm correct?

Example



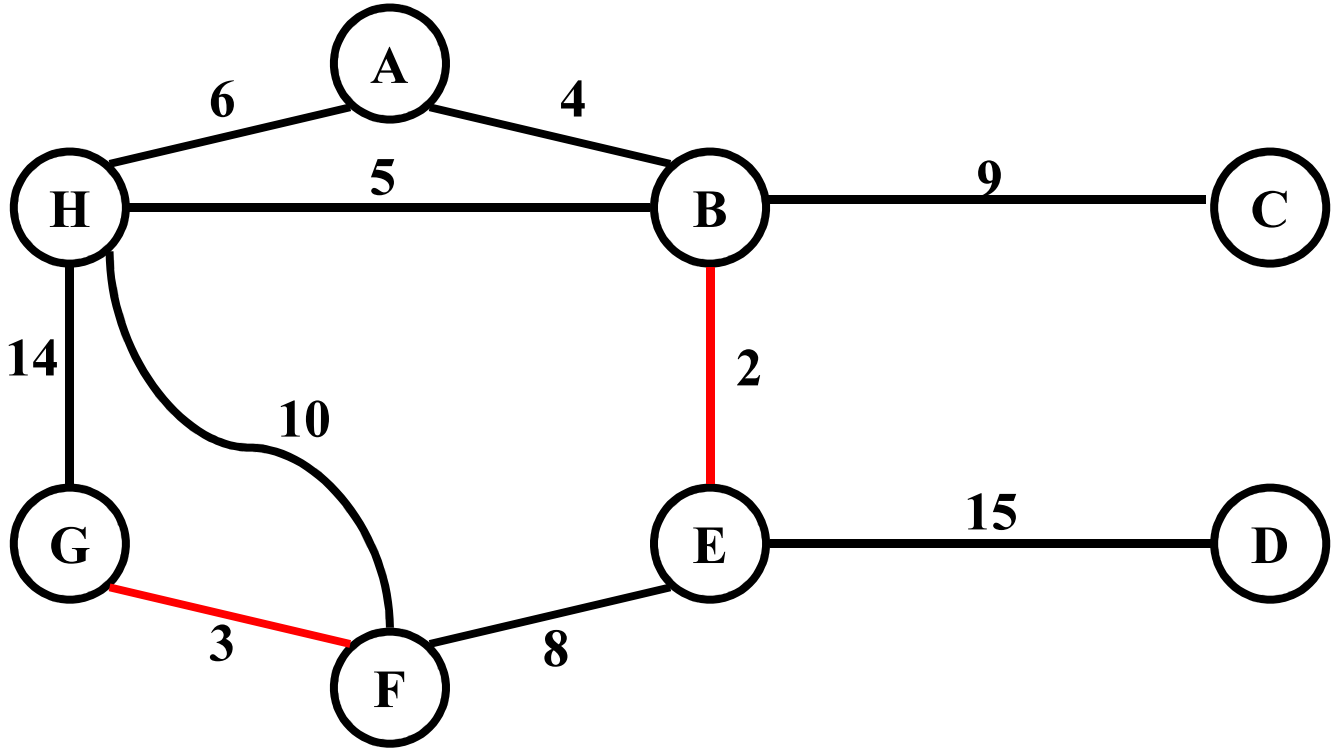
$\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{F\}, \{G\}, \{H\}$

Example



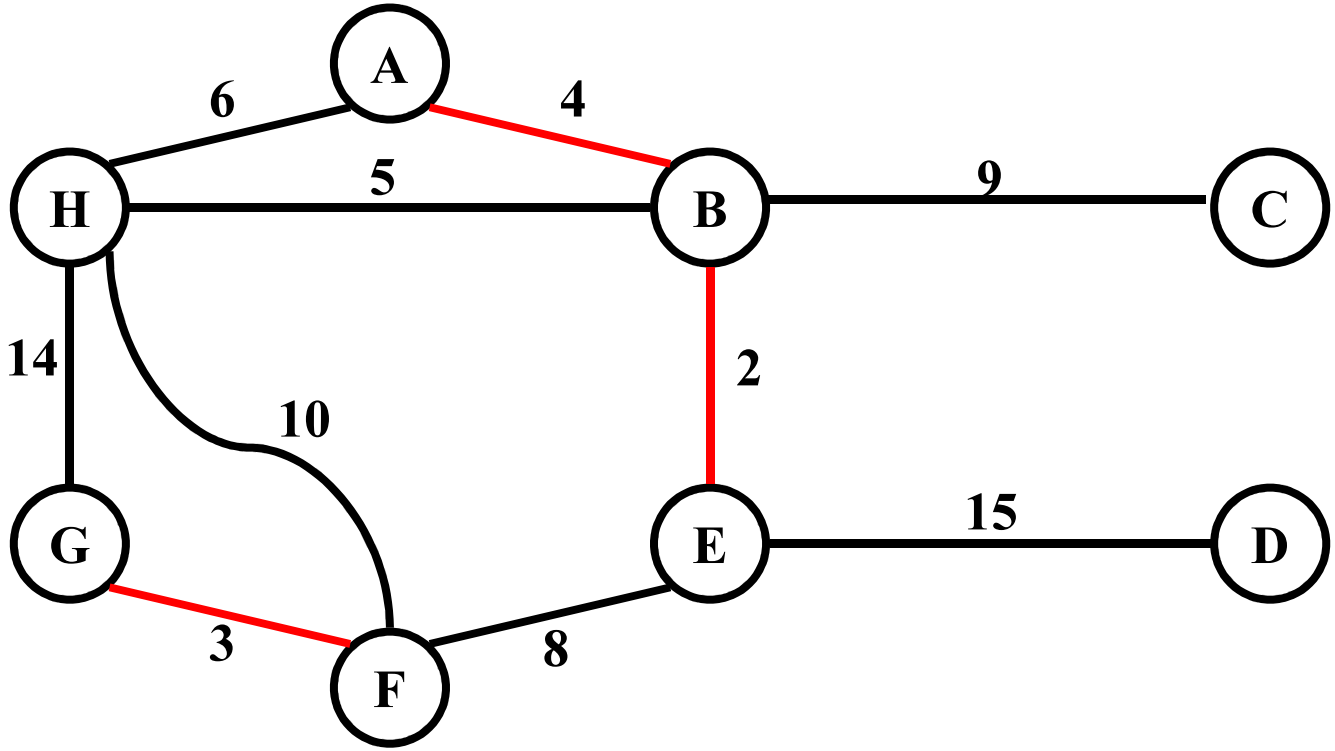
$\{A\}, \{B, E\}, \{C\}, \{D\}, \{F\}, \{G\}, \{H\}$

Example



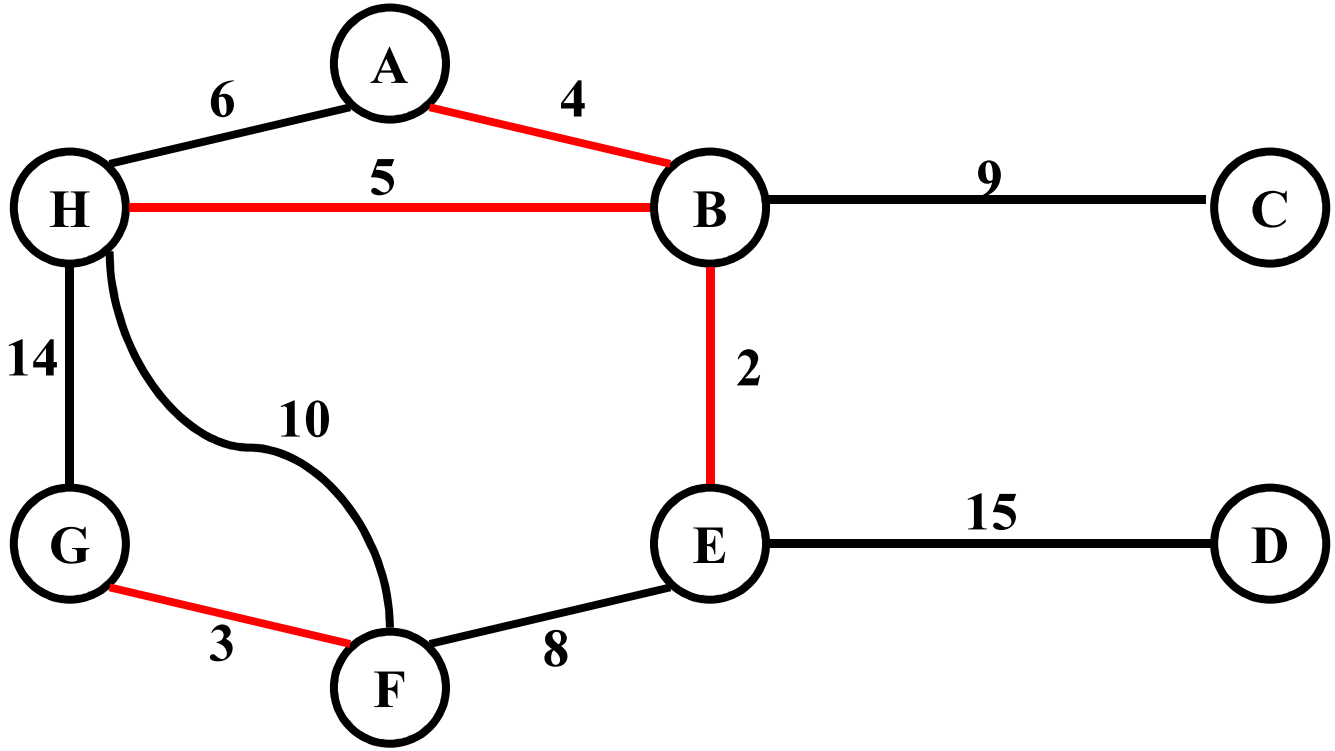
$\{A\}, \{B, E\}, \{C\}, \{D\}, \{F, G\}, \{H\}$

Example



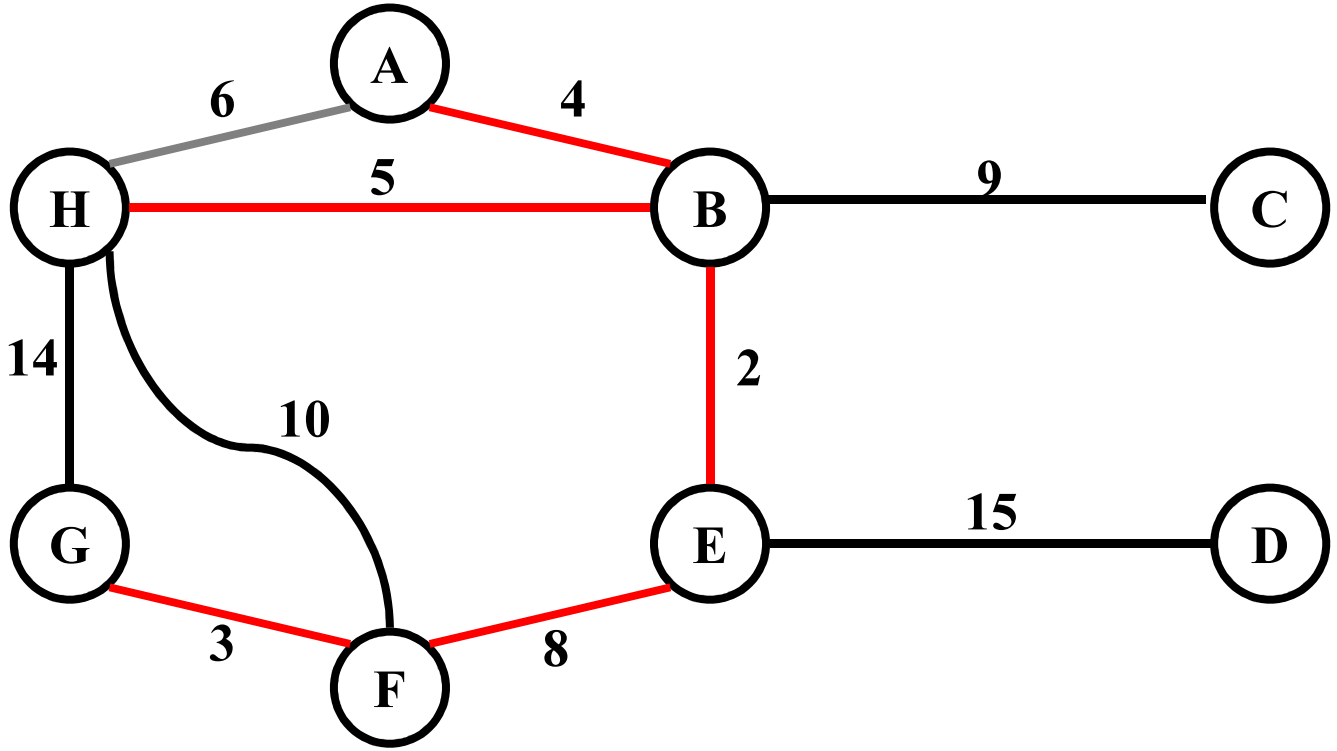
$\{A, B, E\}$, {C}, $\{D\}$, $\{F, G\}$, $\{H\}$

Example



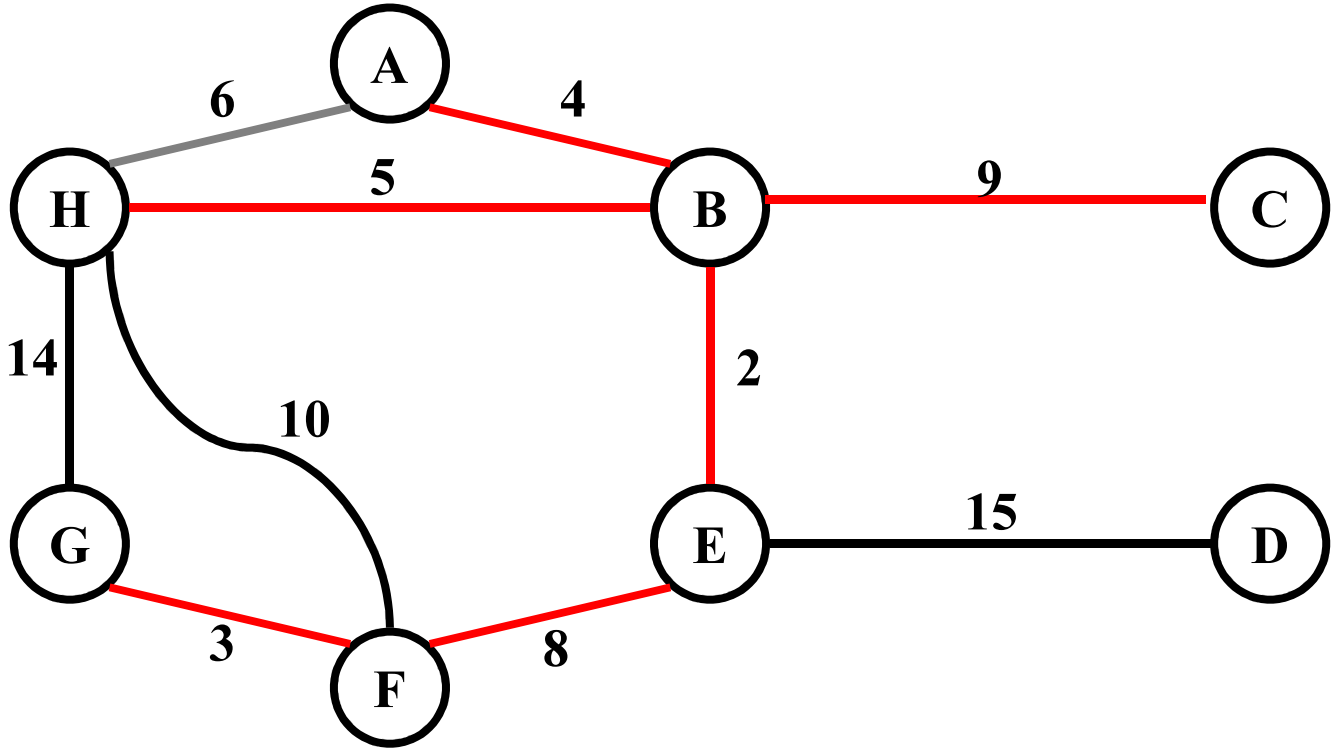
$\{A, B, E, H\}$, {C}, $\{D\}$, $\{F, G\}$

Example



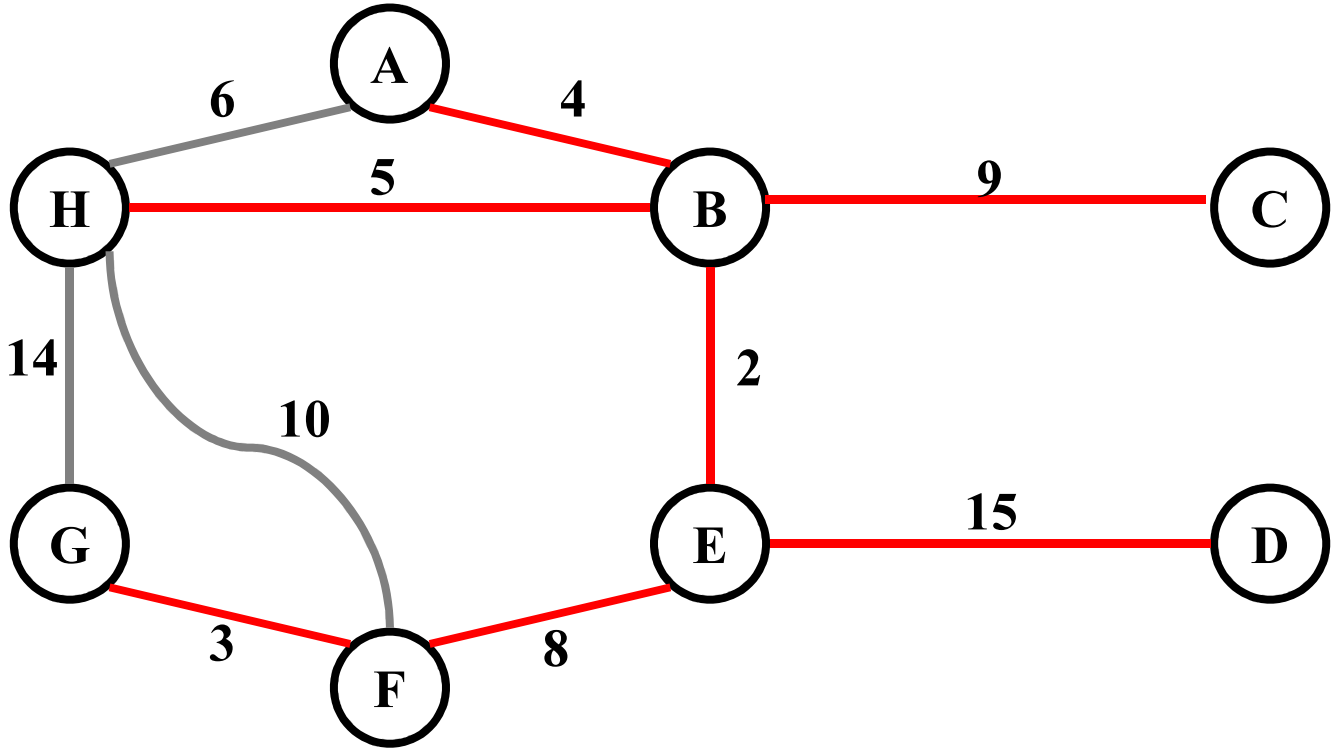
{*A*, *B*, *E*, *H*, *F*, *G*}, {C}, {*D*}

Example



$\{A, B, E, H, F, G, \mathrm{C}\}, \{D\}$

Example



{*A*, *B*, *E*, *H*, *F*, *G*, C, *D*}

Kruskal's Running Time

Sort: $\Theta(E \times \lg E) = \Theta(E \times \lg V)$, since $|E| \leq |V|^2$

$\Theta(V)$ calls to Insert

$\Theta(E)$ calls to FindSet

$\Theta(V)$ calls to Union

Thus $\Theta(E \times \alpha(E, V))$ time using best data structure.

m operations on n sets require $\Theta(m \cdot \alpha(m, n))$ time.

$\alpha(m, n)$ is "functional inverse" of Ackermann's fn.

$\alpha(m, n) \leq 4$ even for $m, n = 10^{80}$

$\alpha(m, n)$ grows very slowly, but not constant!

Best MST to date: 1993 Karger, Klein, Tarjan $\Theta(n)$ time randomized.