Linear Data Structure

http://smartlead.hallym.ac.kr

Instructor: Jin Kim
010-6267-8189(033-248-2318)
jinkim@hallym.ac.kr
Office Hours:

늠름한 허스키

Non Linear Data Structure

- Data structure we will consider this semister:
  - Tree(트리)
  - Binary Search Tree(이진탐색트리)
  - Graph(그래프)
  - Weighted Graph(가중치그래프)
  - Sorting(정렬)
  - Balanced Search Tree(균형탐색트리)

가중치 그래프
Weighted Graph

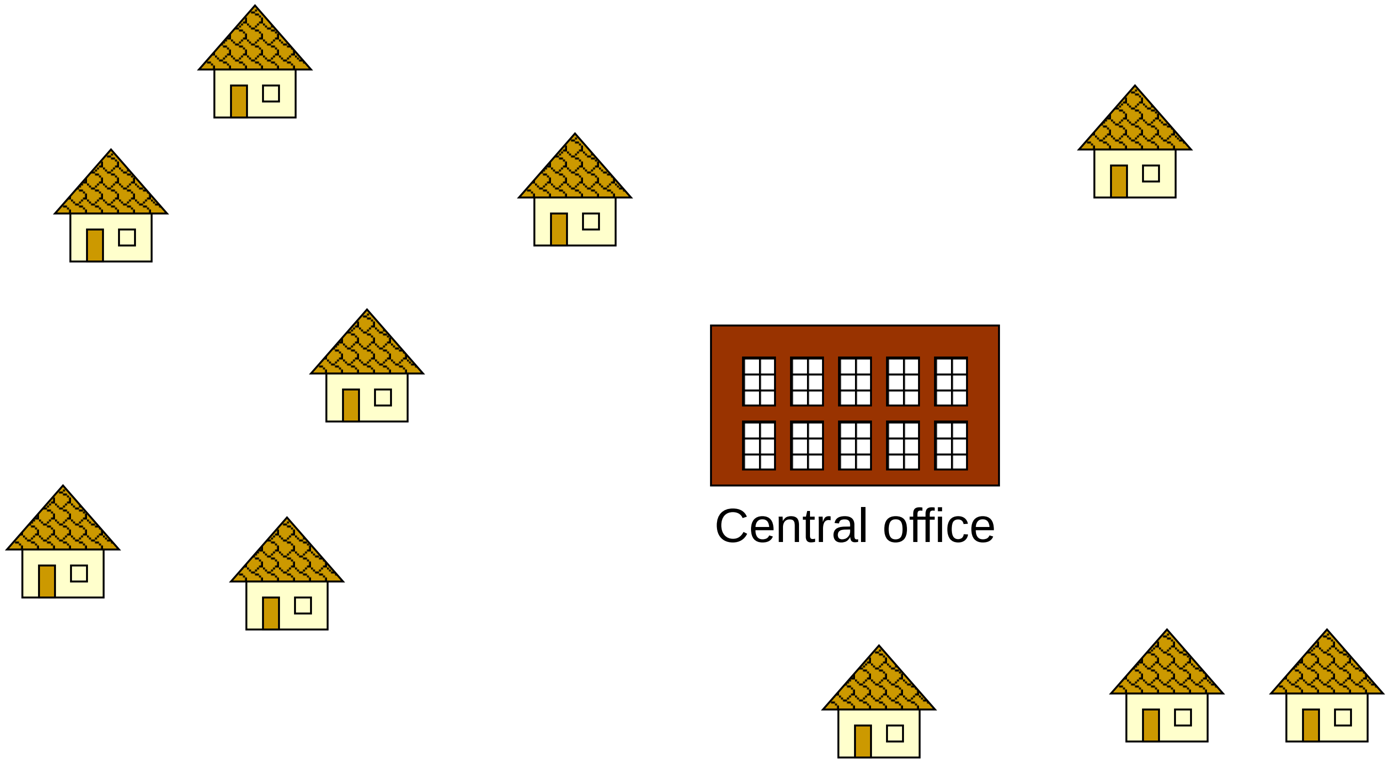
- Minimum Spanning Tree(최소비용간선트리)
- Shortest path(최단경로)
- Topological order(토폴로지컬(위상) 순서)
- Critical Path(임계경로)

Definition

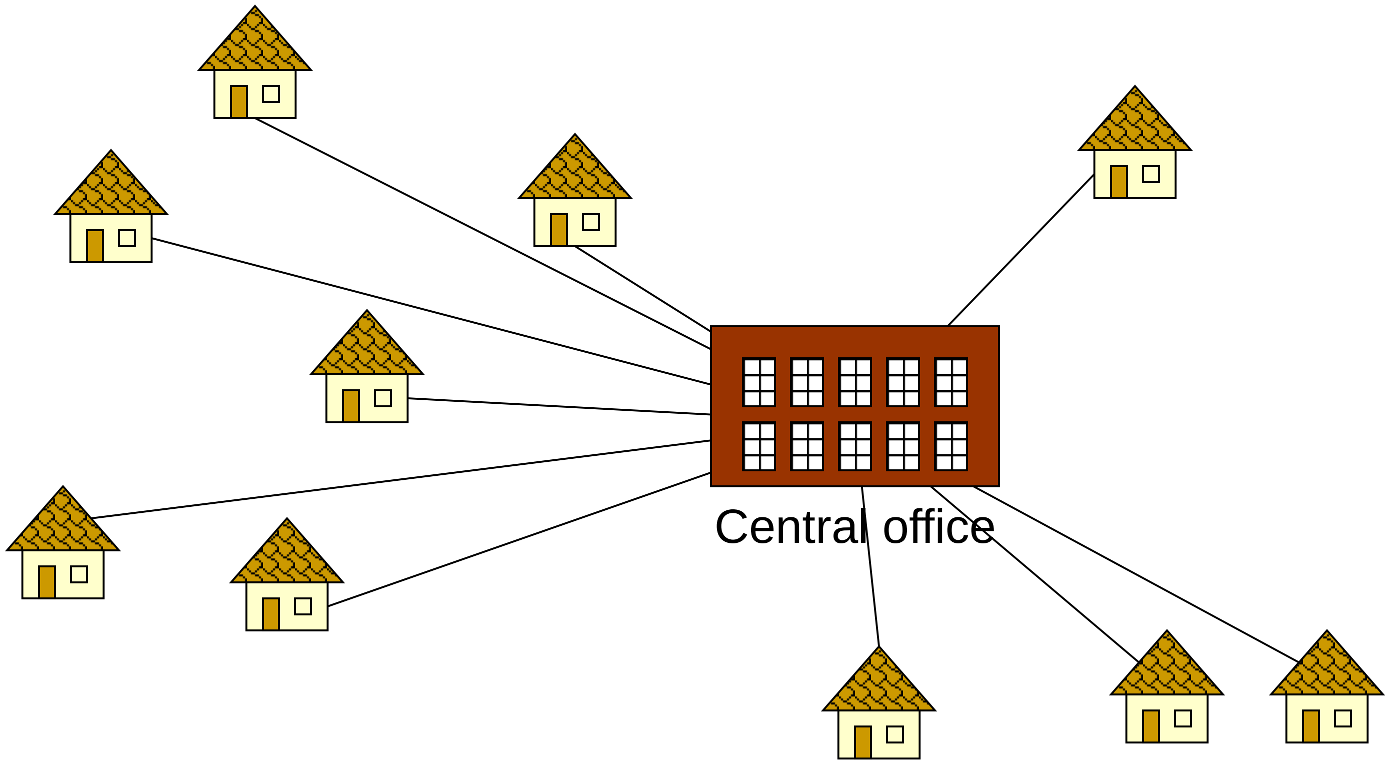
- A Minimum Spanning Tree (MST) is a subgraph of an undirected graph such that the subgraph spans (includes) all nodes, is connected, is acyclic, and has minimum total edge weight

Application(응용) of MST: 사례

Problem: Laying Telephone Wire(배선)

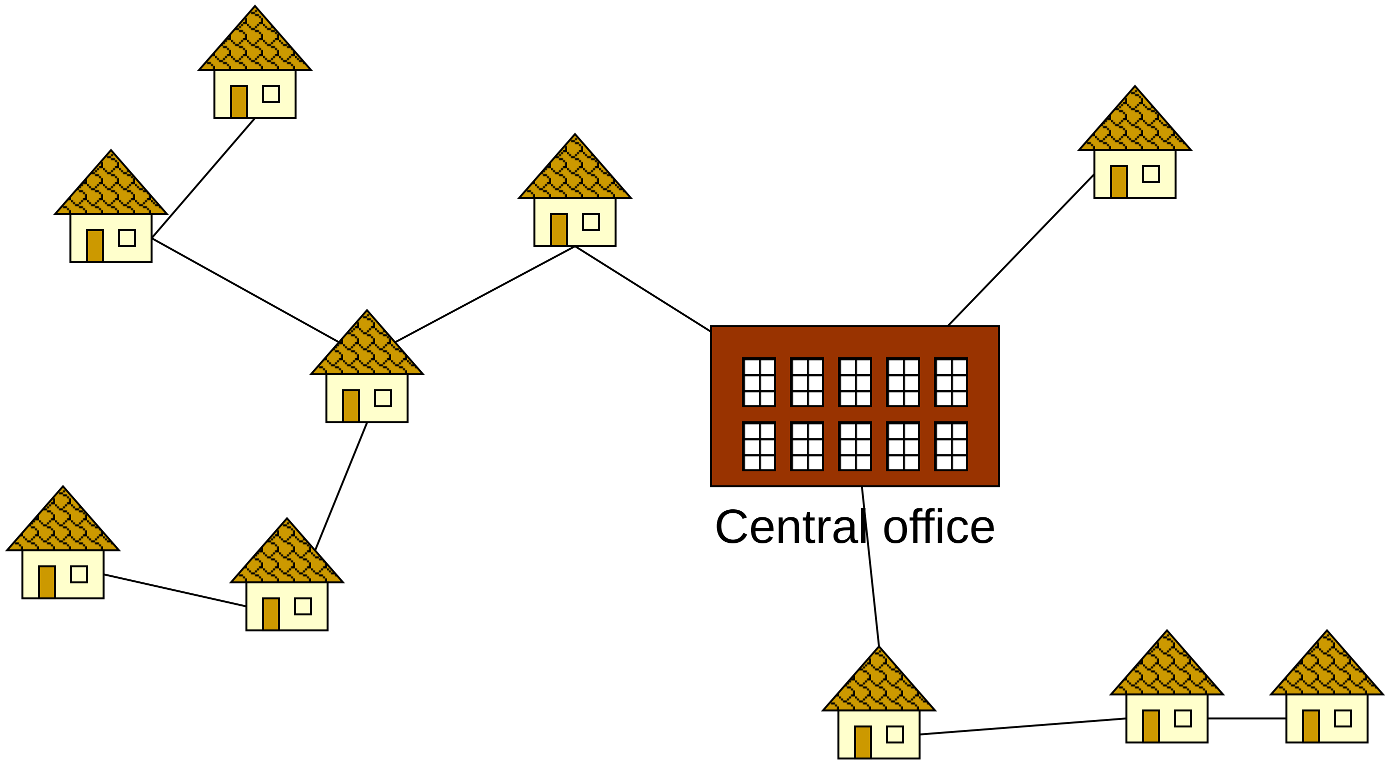


Wiring: Naive Approach(단순접근)



Expensive!(비싸다)

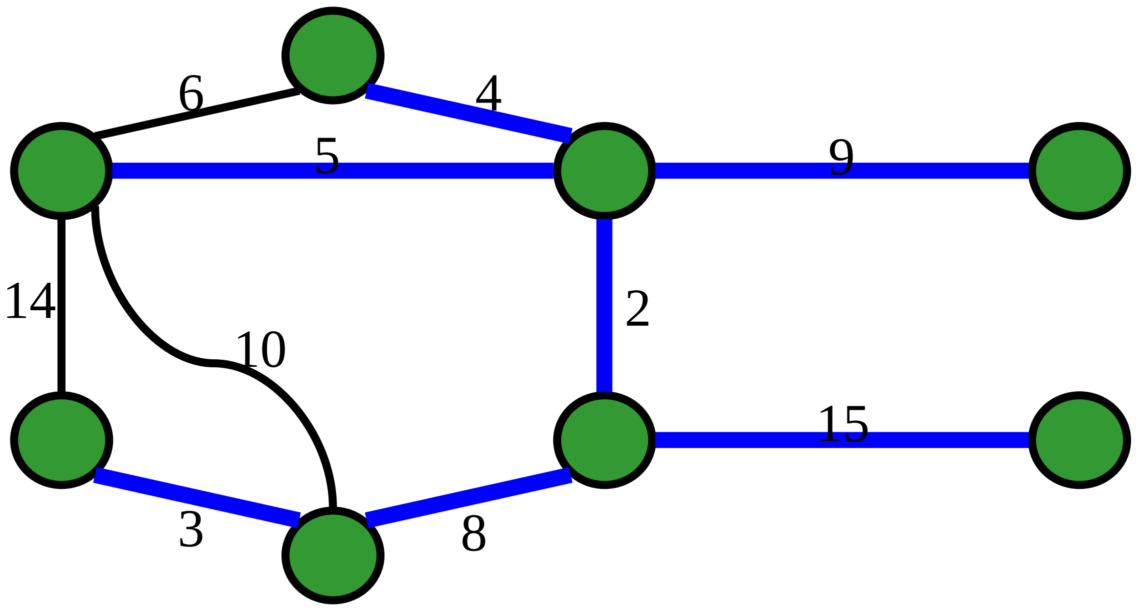
Wiring: Better Approach



Minimize the total length of wire connecting the customers
(전선의 길이의 합이 최소화되면 전선값을 줄일수 있다)

Minimum Spanning Tree (최소비용 간선트리)

- Problem: given a connected, undirected, weighted graph, find a *spanning tree(간선트리)* using edges that minimize the total weight
- 연결된 무방향 가중치 그래프가 주어졌을때, 가중치의 합이 최소가 되는 간선트리를 구함

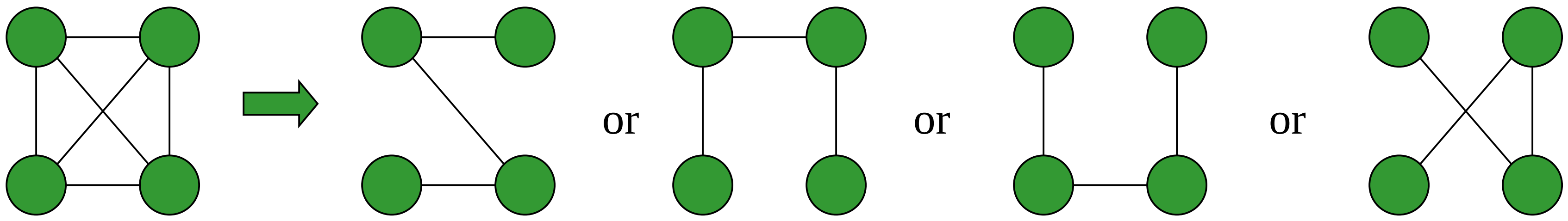


Spanning Trees신장트리

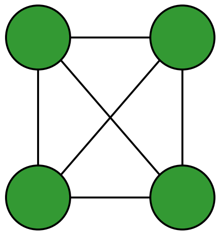
A spanning tree of a graph is just a subgraph that contains all the vertices and is a tree.신장트리는 어떤 그래프의 부분 그래프

A graph may have many spanning trees. (여러 개의 신장 트리 존재)

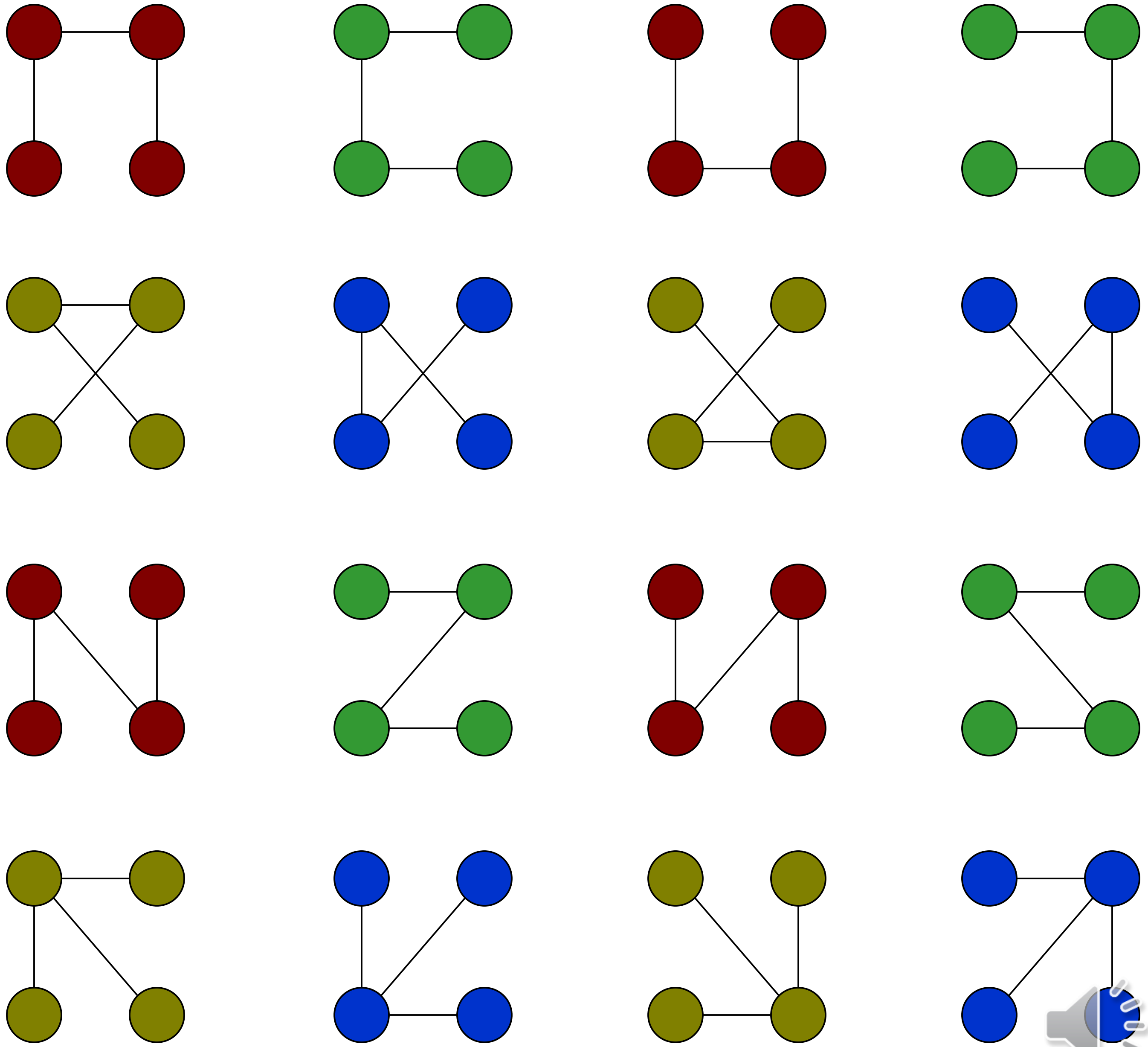
Graph A   Some Spanning Trees from Graph A



Complete Graph완전그래프

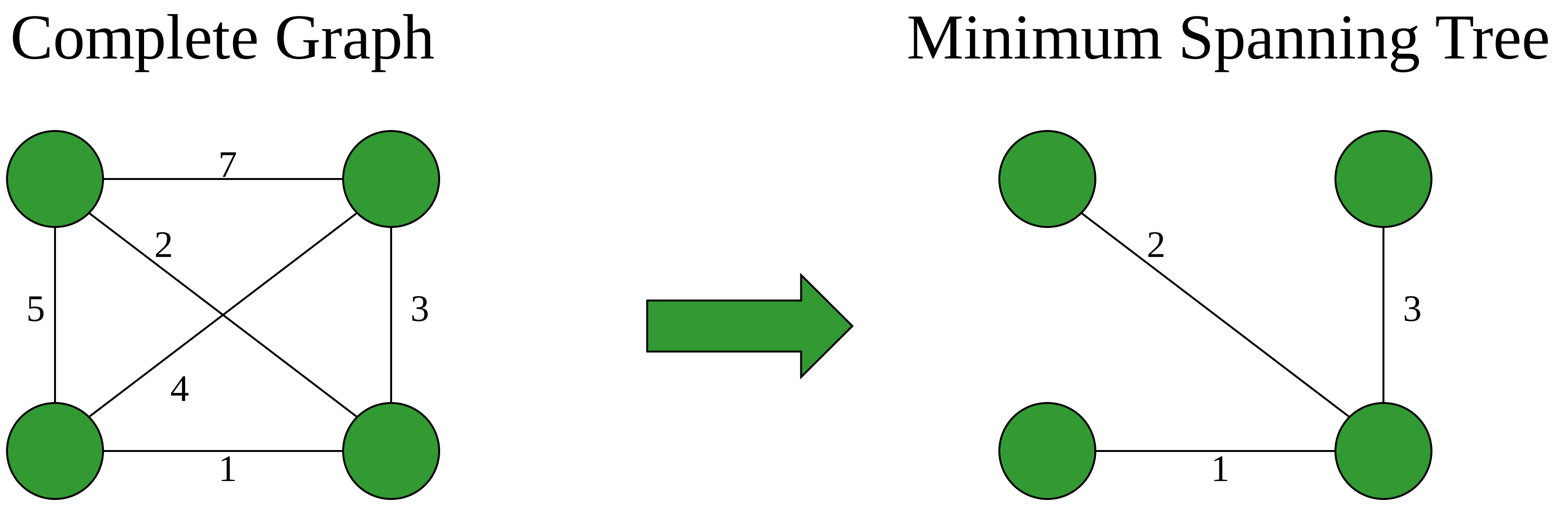


All 16 of its Spanning Trees



Minimum Spanning Trees

The Minimum Spanning Tree for a given graph is the Spanning Tree of minimum cost for that graph.최소비용신장트리는 가중치합이 최소



Algorithms for Obtaining the Minimum Spanning Tree
최소비용신장트리를 얻는 알고리즘들

- Kruskal's Algorithm
- Prim's Algorithm
- Sollin’s Algorithm

The algorithms of Kruskal and Prim

- In Kruskal's algorithm,
  - The set A is a forest.(숲에서 시작될 수 있다)
  - The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.
- In Prim's algorithm,
  - The set A forms a single tree.(계속 단일 트리)
  - The safe edge added to A is always a least-weight edge connecting the tree to a vertex not in the tree.

Prim's Algorithm

```
procedure prim(G(V, E)):
    choose any s∈V
    let L={s}, R=V-{s}
    T = {} // final tree edges
        for i = 1 to n:
          1. let (u, v) be the min cost edge
          crossing L-R
          2. remove v from R and add to L
                   3. add (u, v) to T
 return T
```

Prim’s Algorithm

```
 procedure prim(G(V, E)):
    choose any s∈V
    let L={s}, R=V-{s}
    let mWE store for each vertex
        in R, its min weight edge to L.
    T = {} // final tree edges
       for i = 1 to n:
        1. let v be min mWE vertex
           let (u, v) be the edge
           minimizing v’s mWE
        2. remove v from R & mWE;add to L
               3. add (u, v) to T
        4. update mWE for each neighbor of v in R
return T
```

Prim’s Algorithm Running Time

Exactly the same As Dijkstra’s Algorithm!

나중에배울 다익스트라 알고리즘과동일

Kruskal's algorithm

```
1    (Sort the edges in an increasing
     order)//간선정렬
2    A:={}
3    while E is not empty do {
3        take an edge (u, v) that is shortest in E
         and delete it from E
4        if  u and v are in different components then
                 add (u, v) to A
         }
```

Note: each time a shortest edge in E is considered.

Kruskal’s Algorithm

Work with edges, rather than nodes

초기에는 숲, 나중에는 하나의 트리

Two steps:

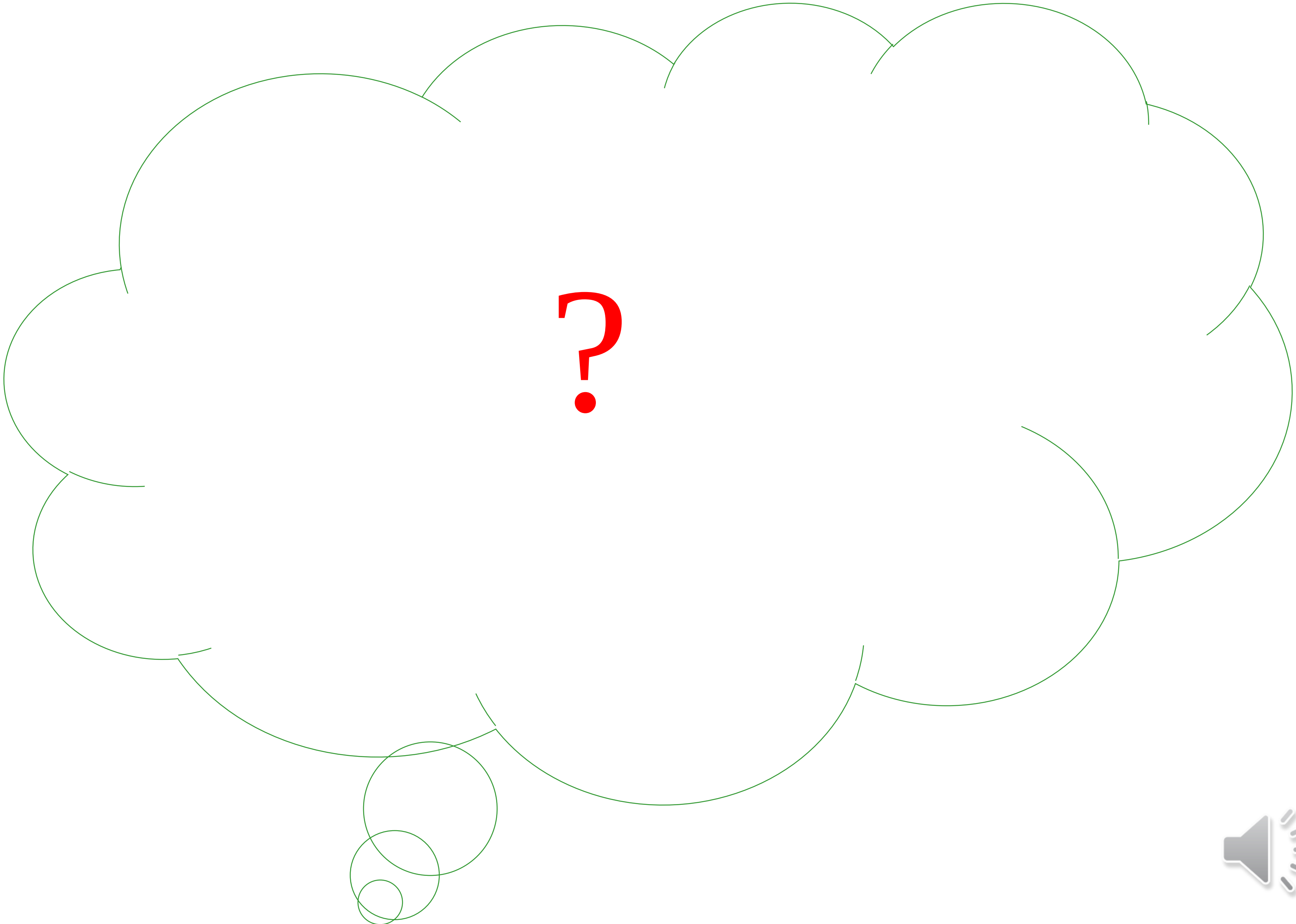
- Sort edges by increasing edge weight(간선을 작은순으로 정렬)
- Select the first |V| – 1 edges that do not generate a cycle(정점-1개의 간선을 선정)

Algorithm Characteristics
알고리즘특징

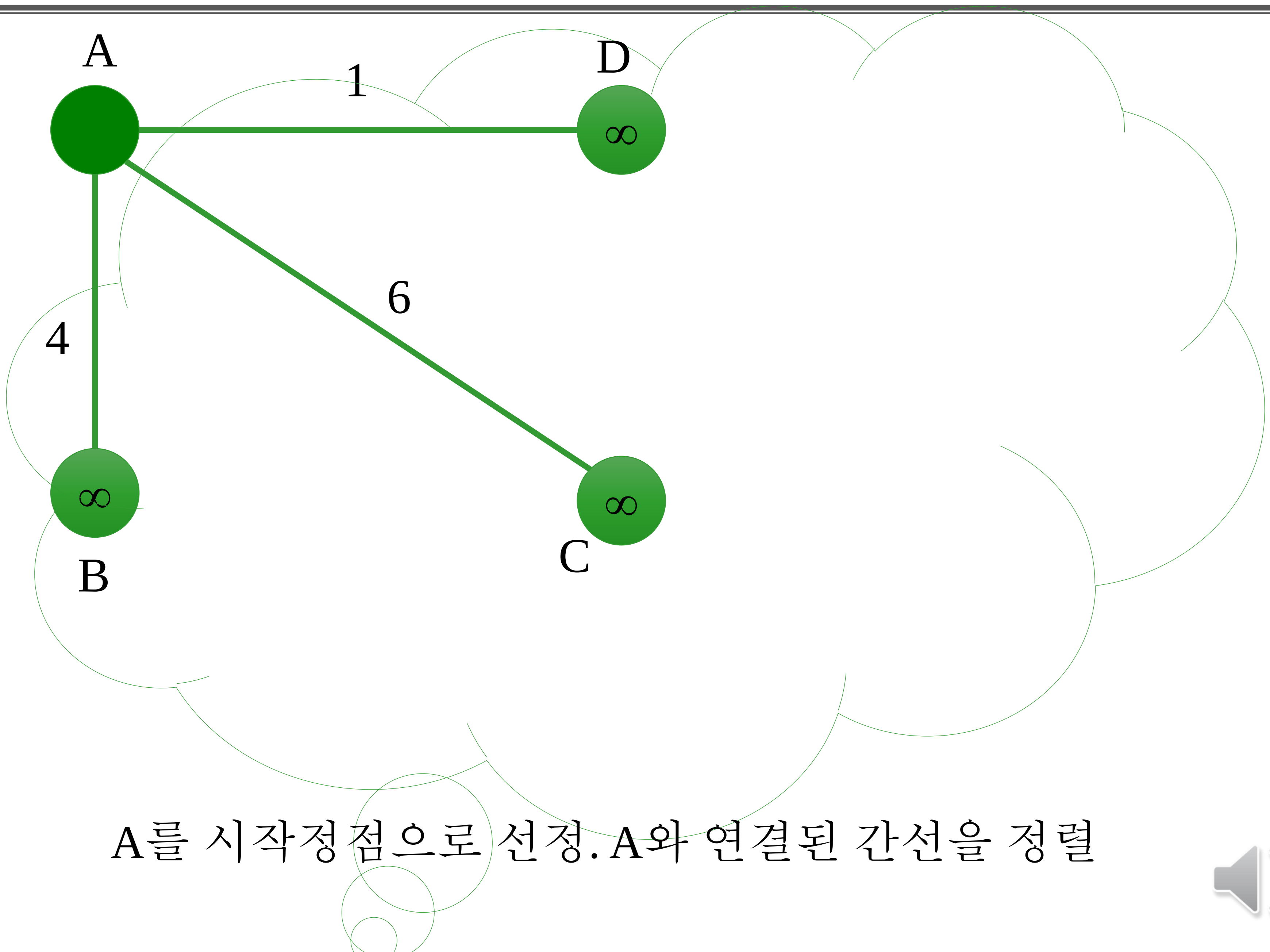
- Both Prim's and Kruskal's Algorithms work with undirected graphs(무방향그래프)
- Both work with weighted and unweighted graphs but are more interesting when edges are weighted(가중치가 없어도 간선트리를 얻을수있지만, 가중치간선트리가 더 흥미) 실제로는 가중치가 없는 그래프는 가중치가 모두 1이라 생각하면 편함.
- Both are greedy algorithms(탐욕적 알고리즘) that produce optimal solutions
- 두 알고리즘 모두 정답 제공.

Prim's Algorithm
프림의 알고리즘 사용 예제
어느 정점이 출발정점이 되어도 무관

Prim’s Algorithm Simulation (2)

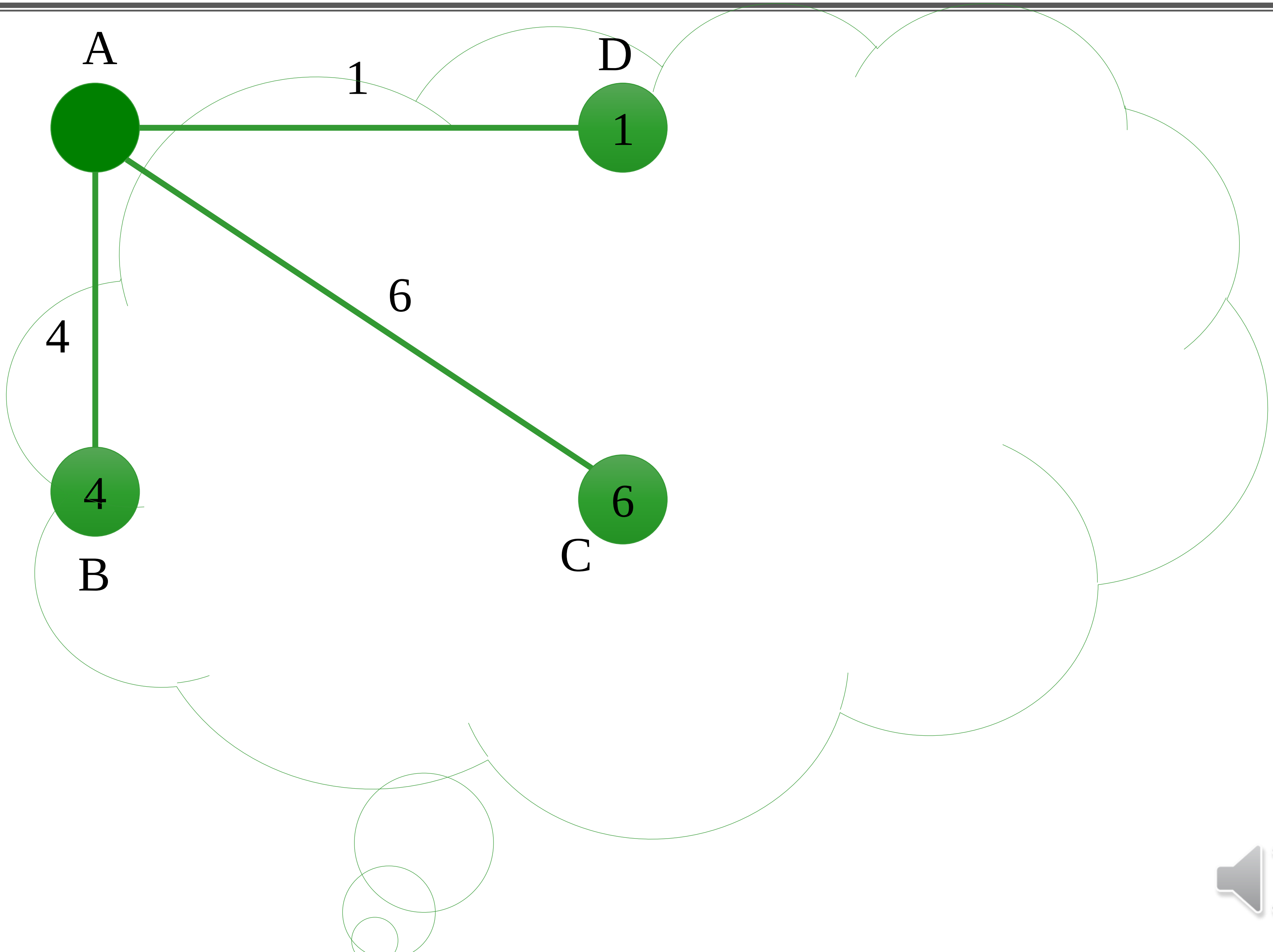


Prim's Algorithm Simulation

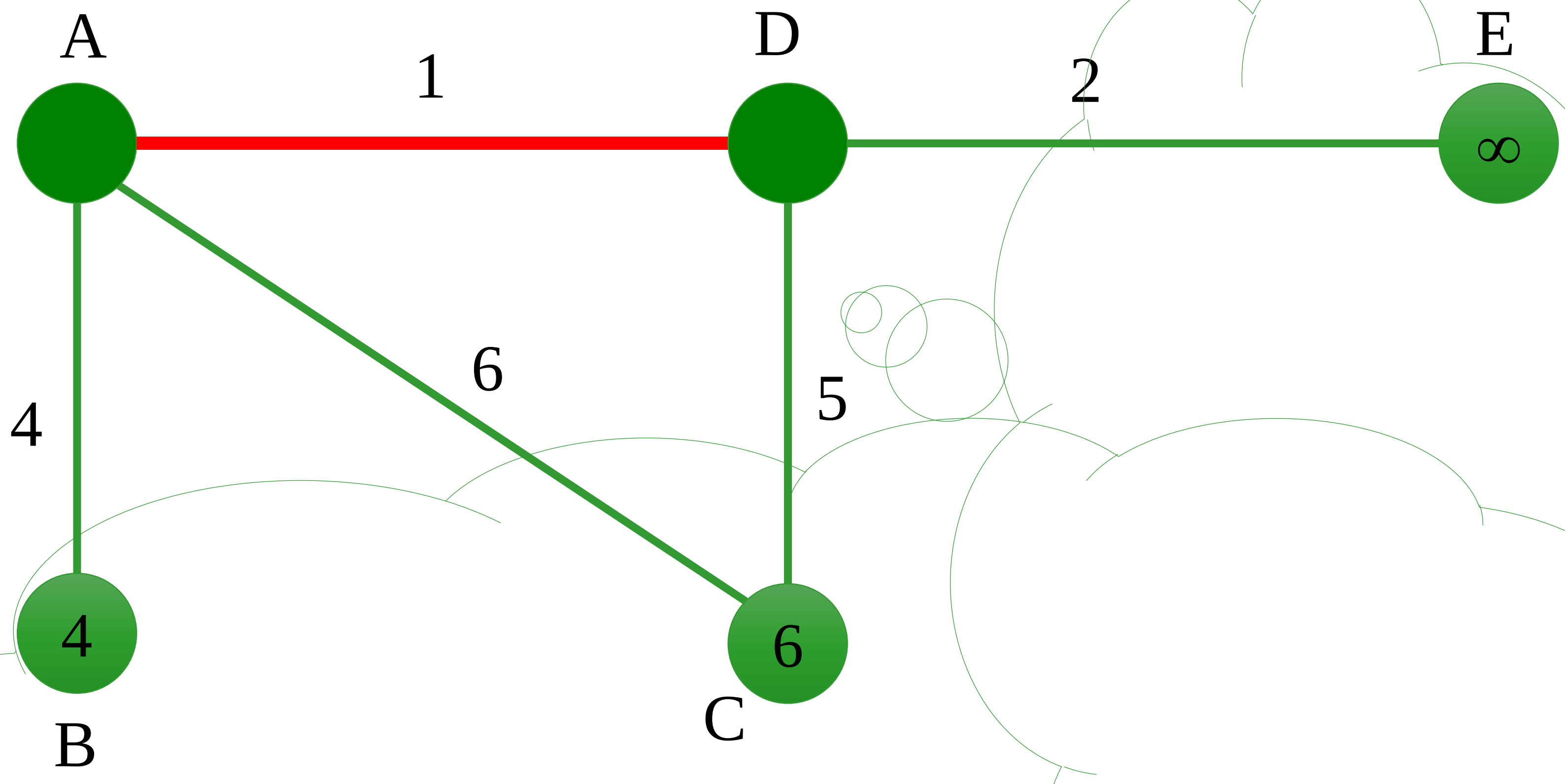


A를 시작정점으로 선정. A와 연결된 간선을 정렬

Prim’s Algorithm Simulation

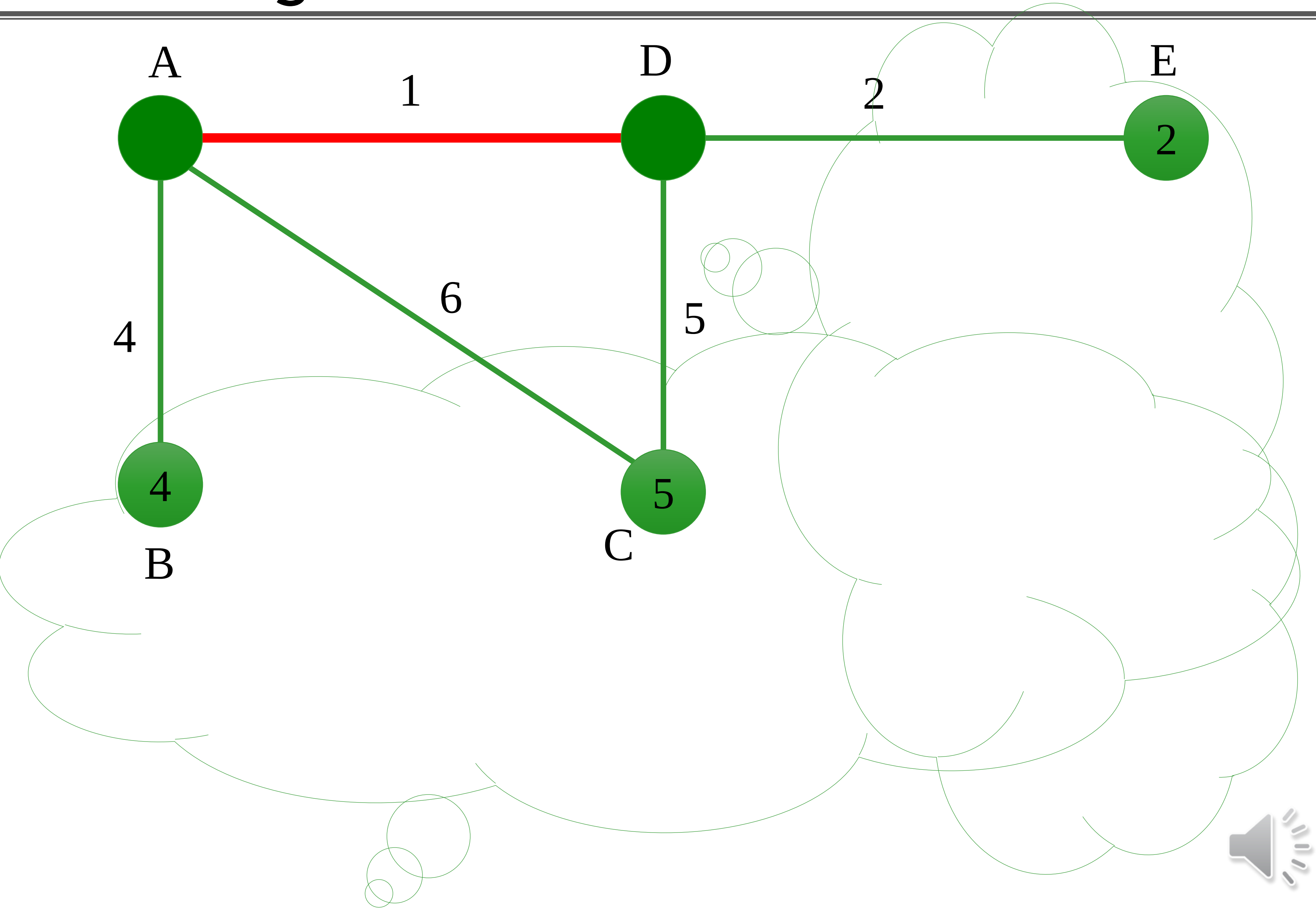


Prim's Algorithm Simulation

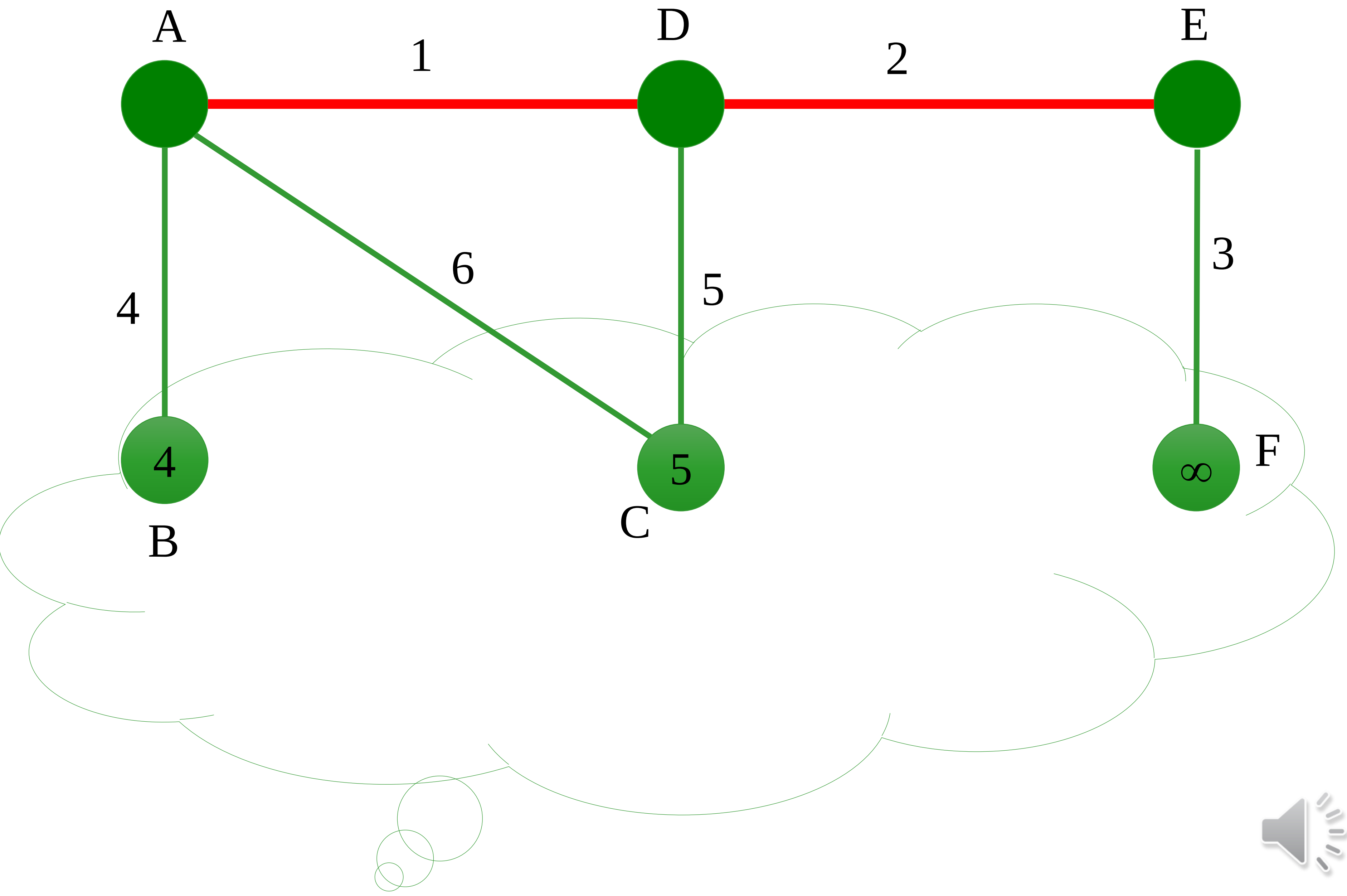


A, D와 관련된 간선을 조사, 가장 작은가중치간선선택

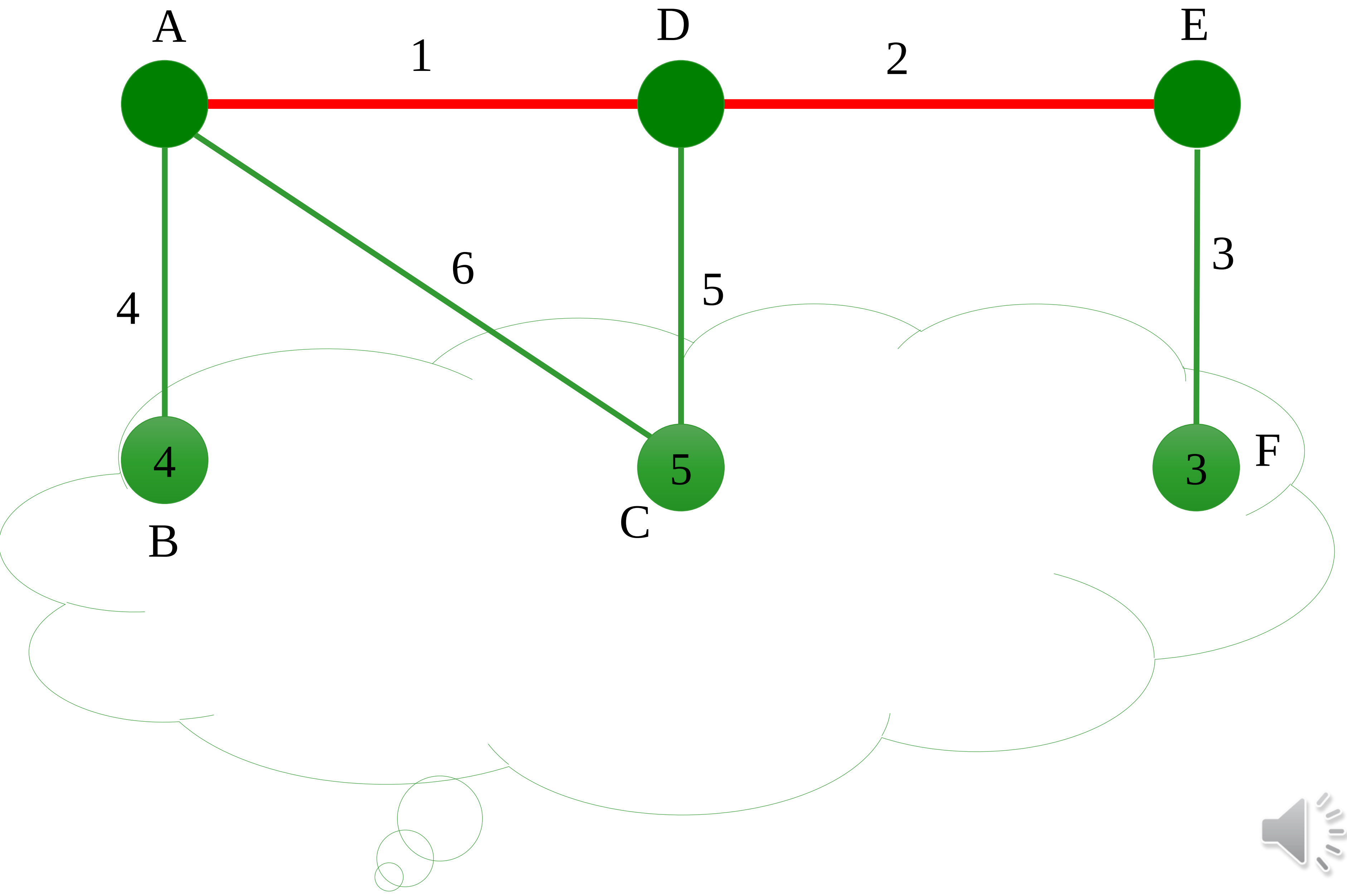
Prim’s Algorithm Simulation



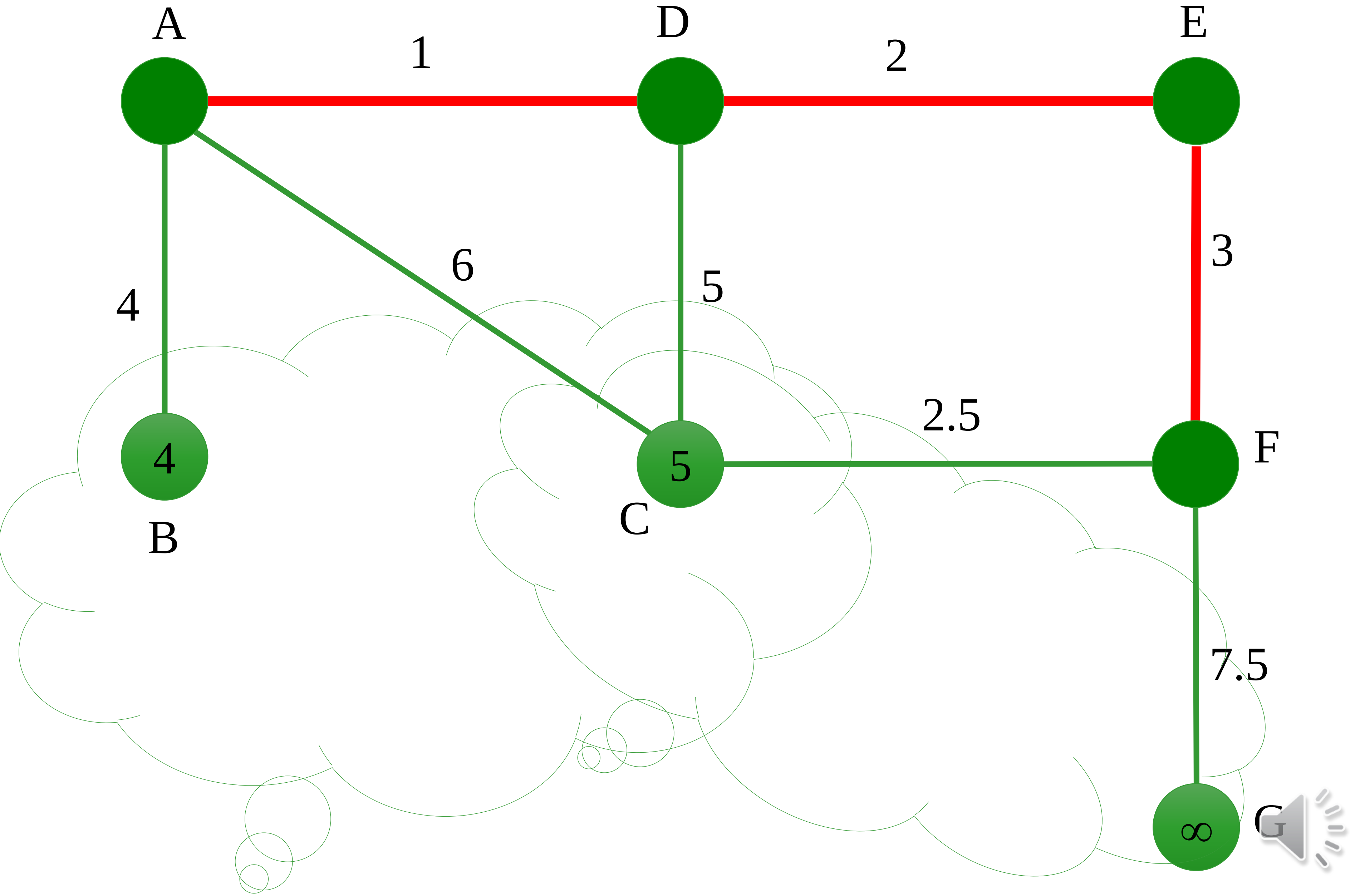
Prim’s Algorithm Simulation



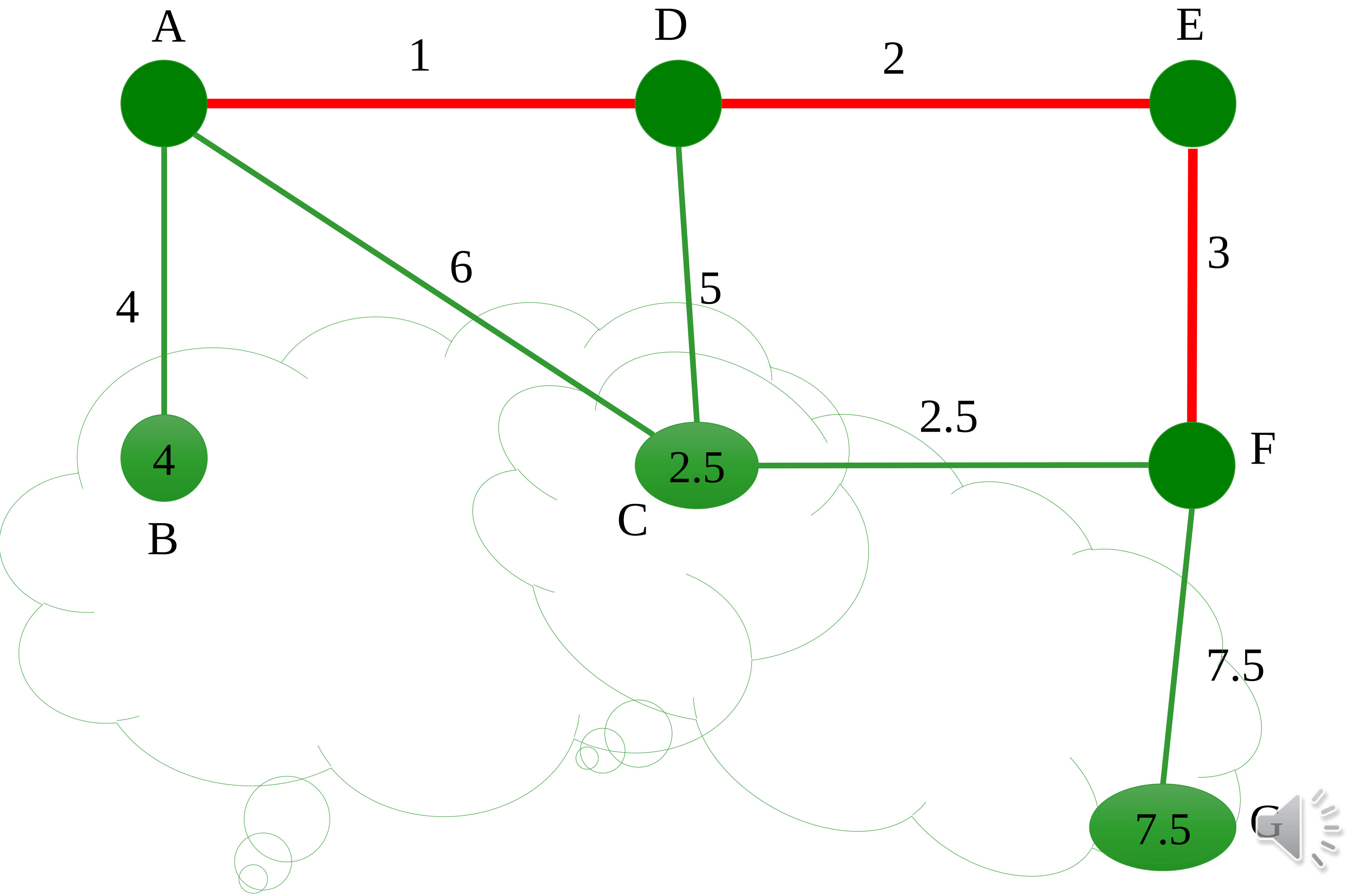
Prim's Algorithm Simulation



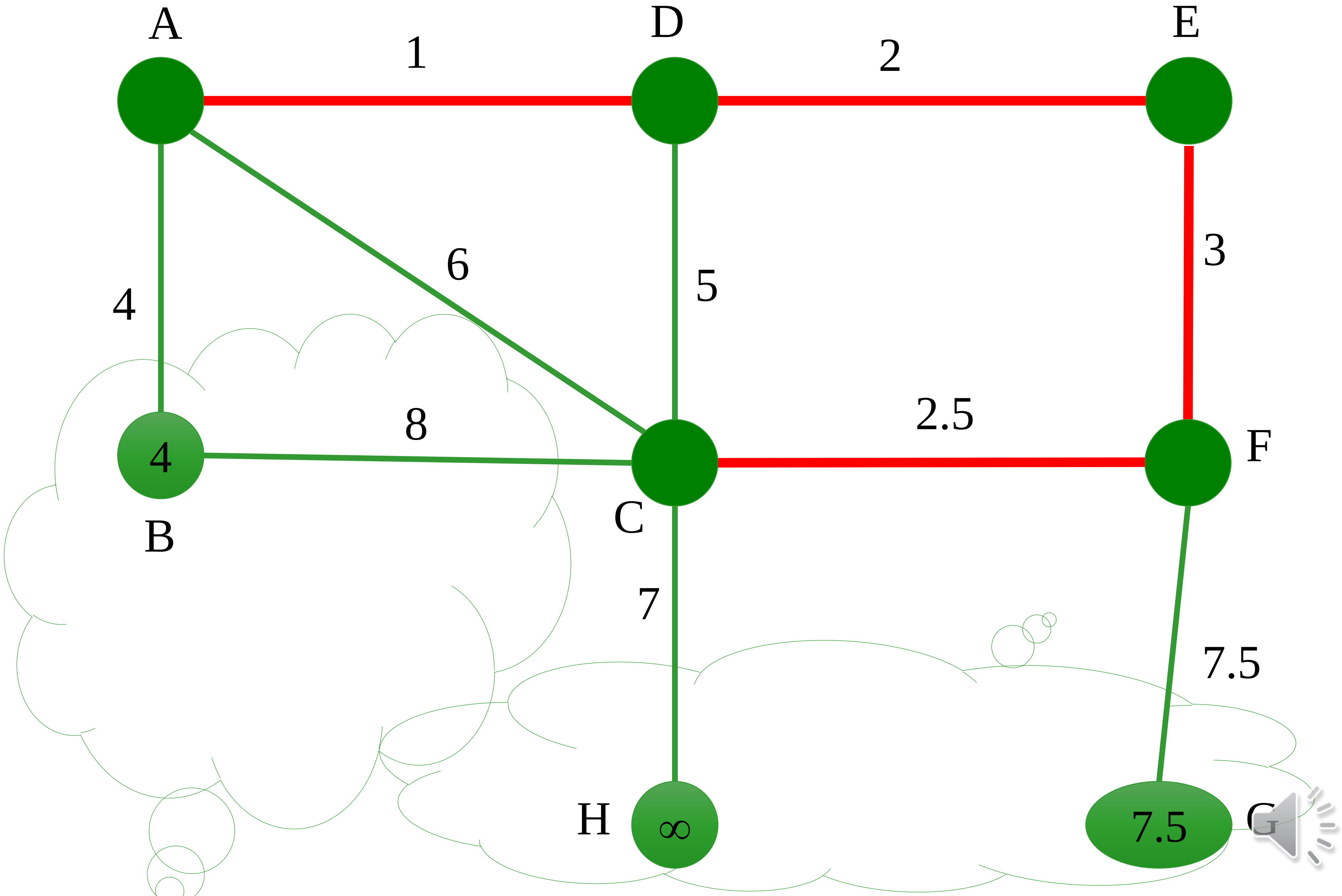
Prim's Algorithm Simulation



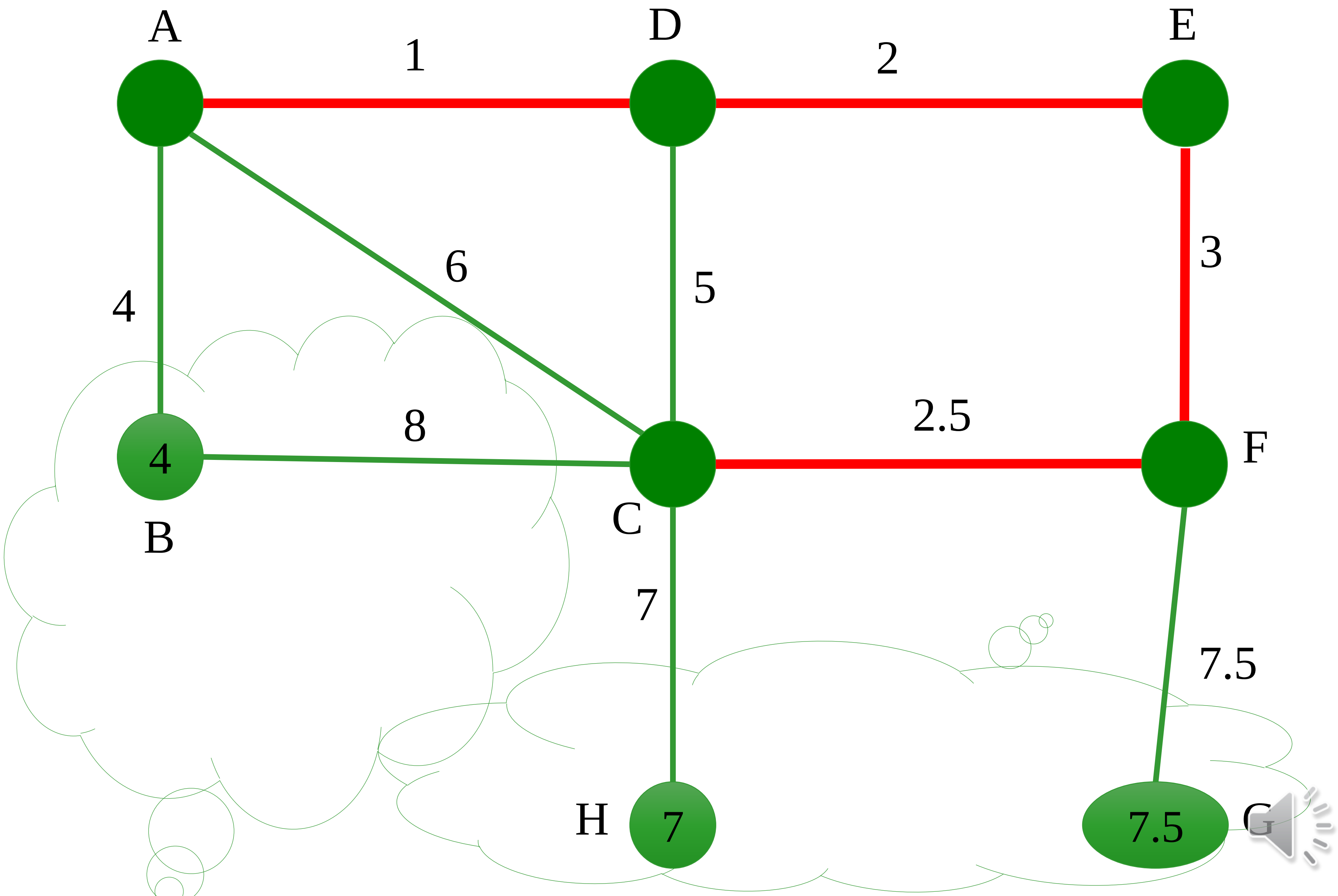
Prim's Algorithm Simulation



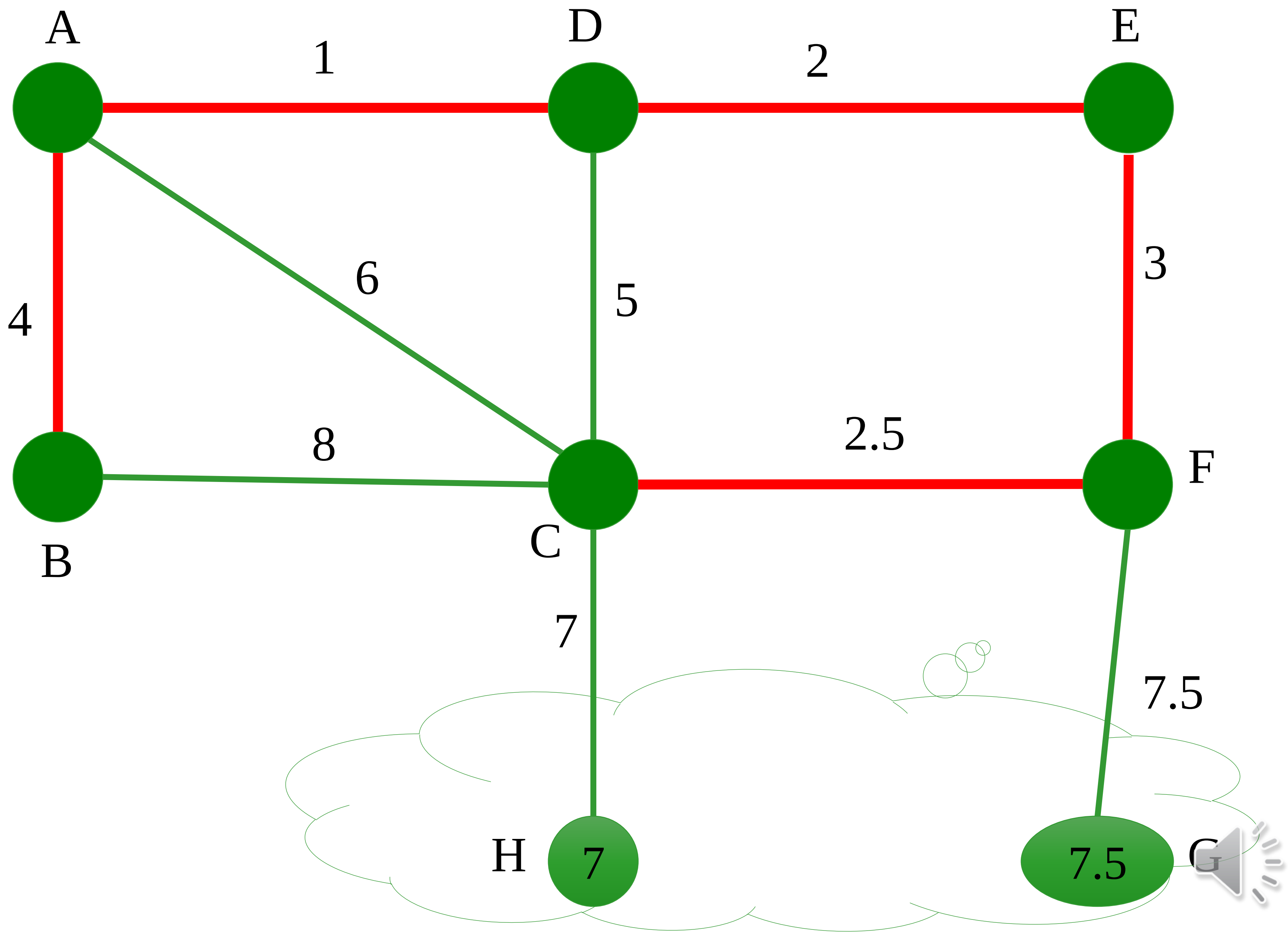
Prim's Algorithm Simulation



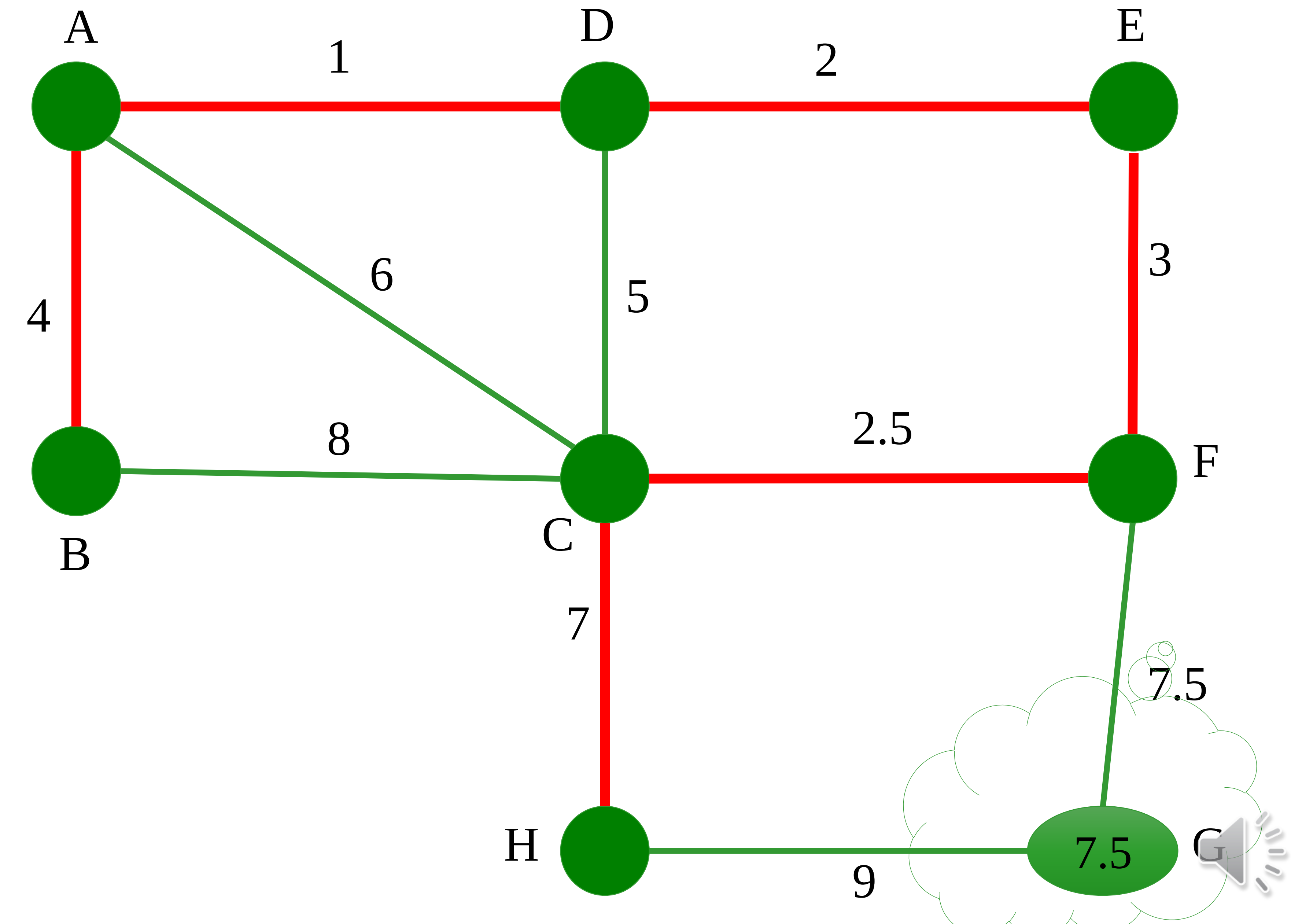
Prim's Algorithm Simulation



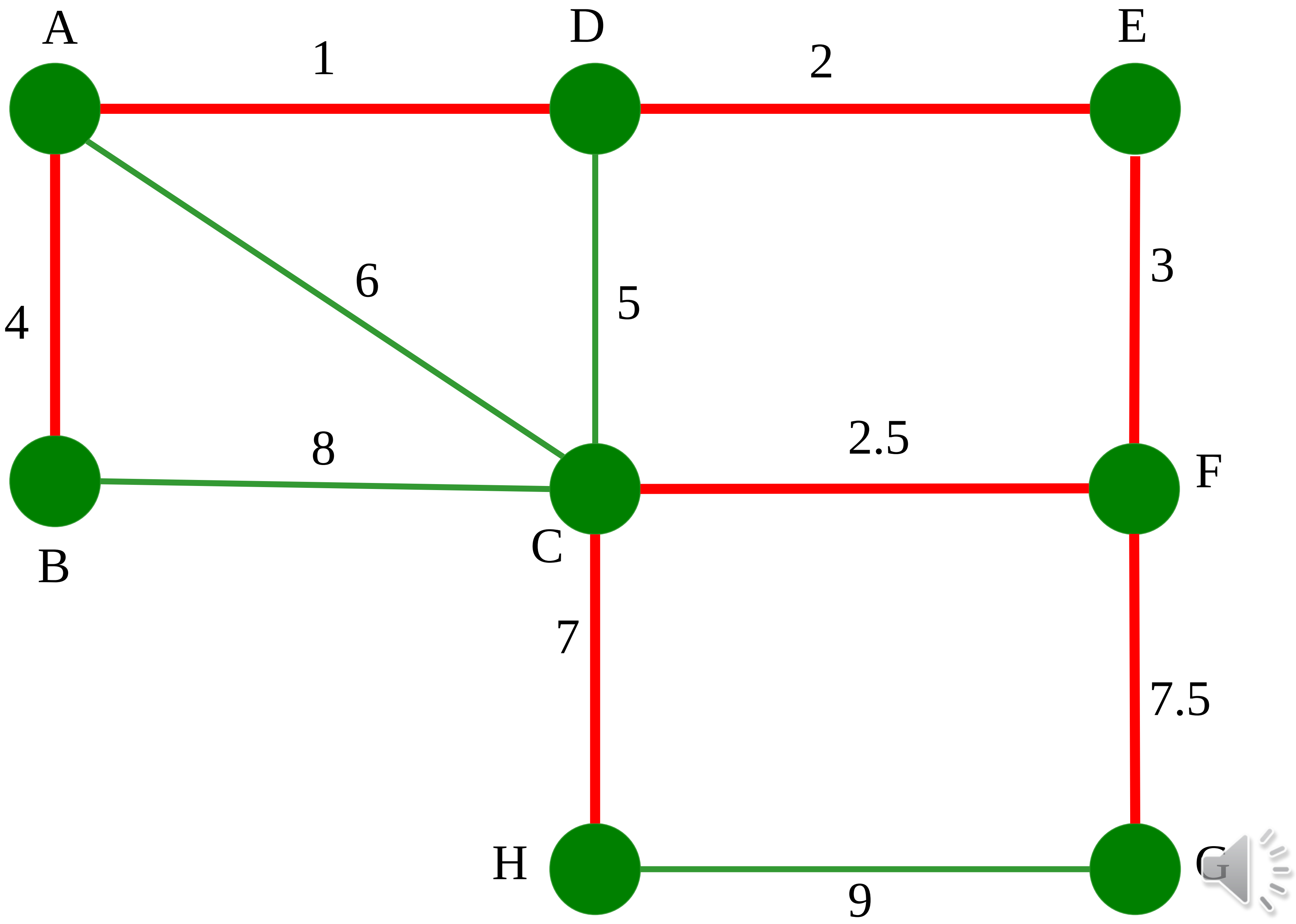
Prim's Algorithm Simulation



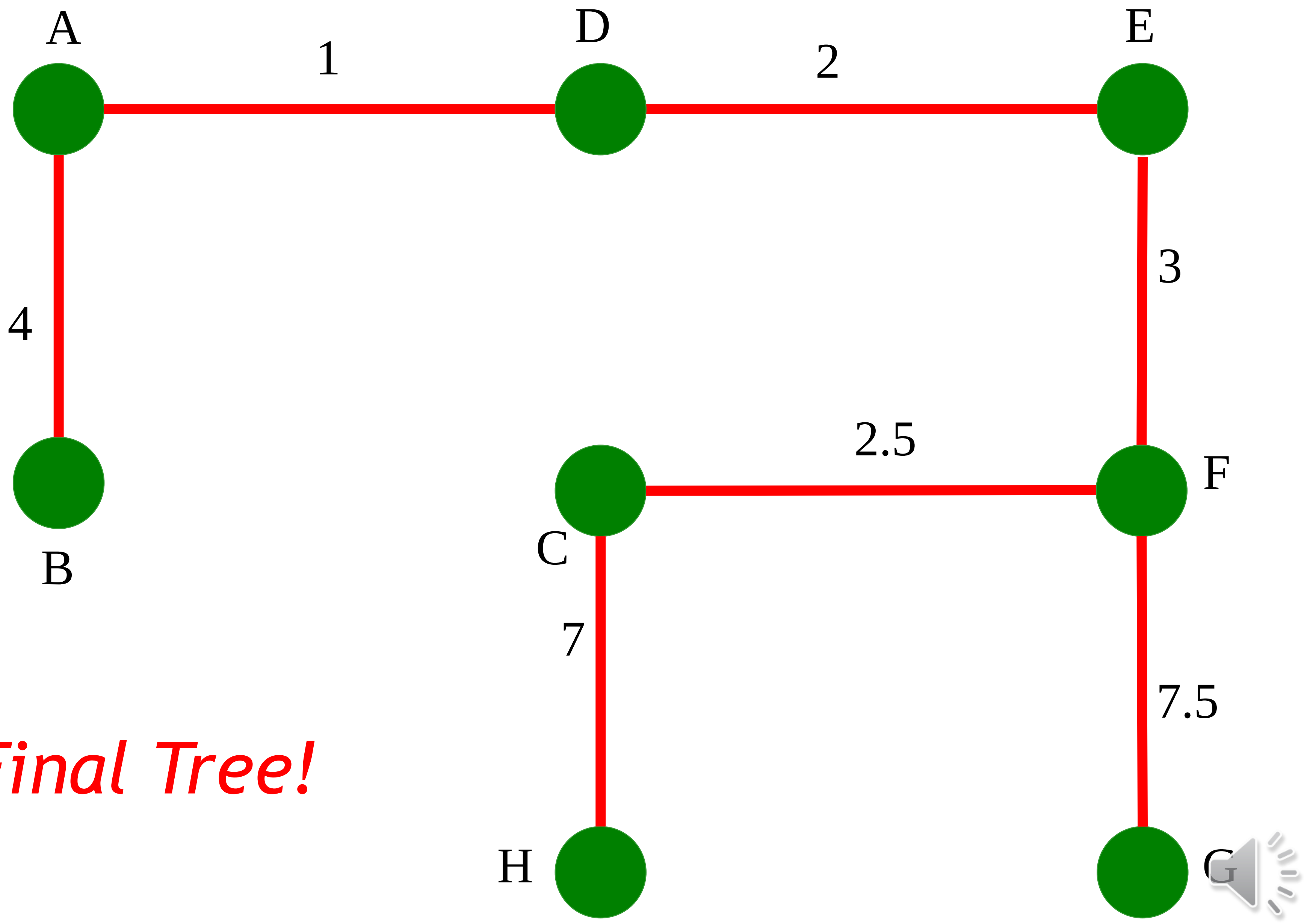
Prim’s Algorithm Simulation



Prim’s Algorithm Simulation



Prim's Algorithm Simulation



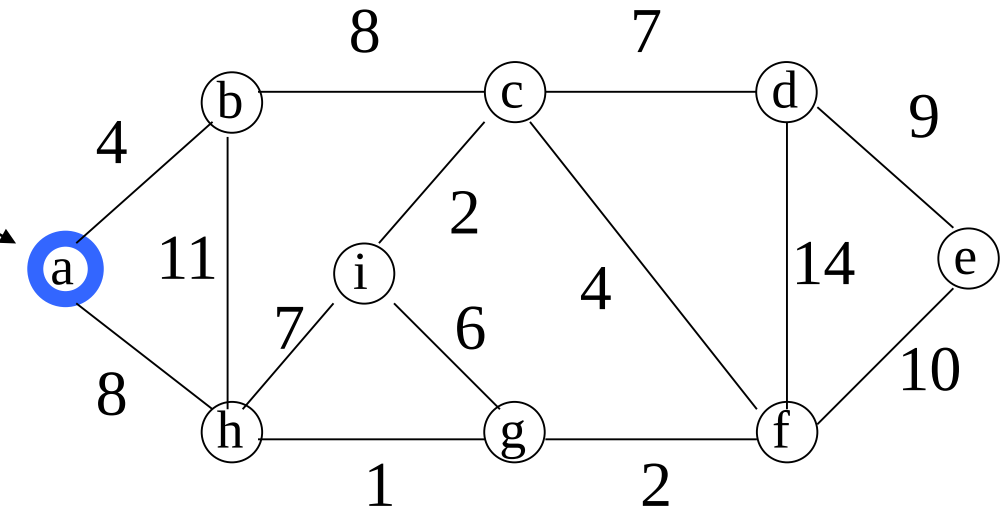
Final Tree!

Prim's Algorithm
프림의 알고리즘 예2

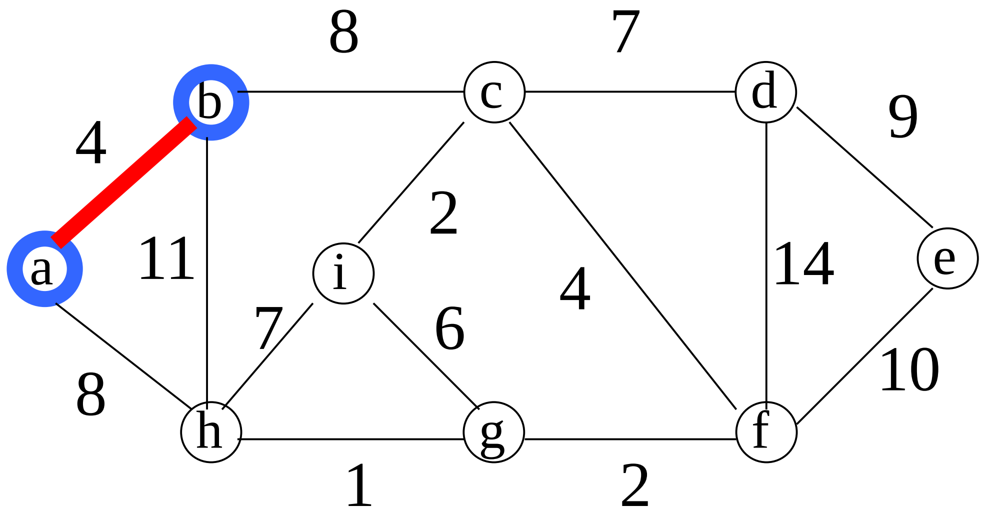
The execution of Prim's algorithm

the root vertex

a를 출발점으로 선택

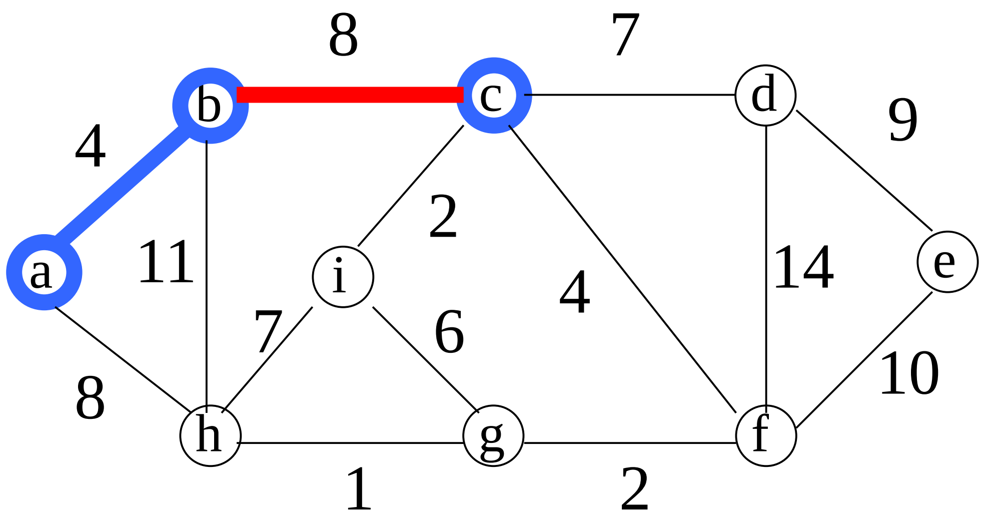


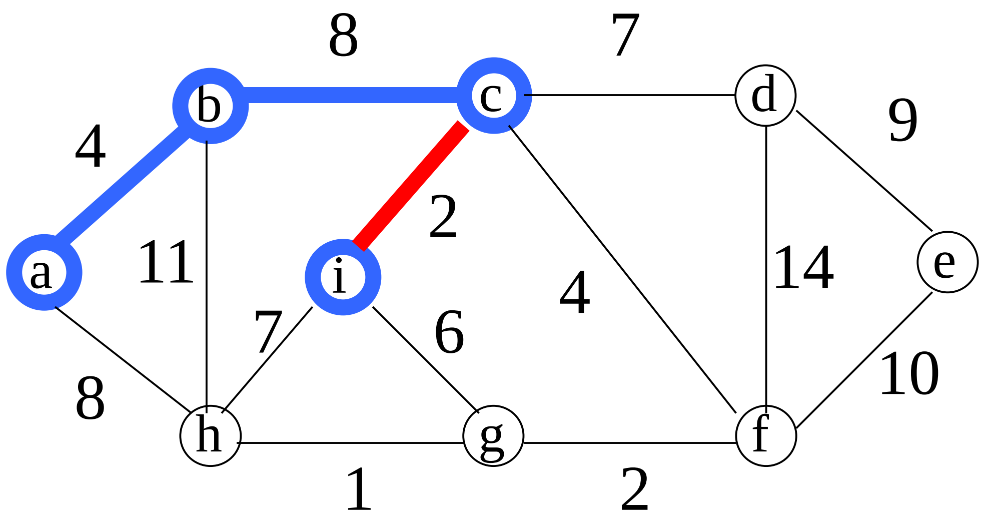
a-b:4
a-h :8
a-others :∞



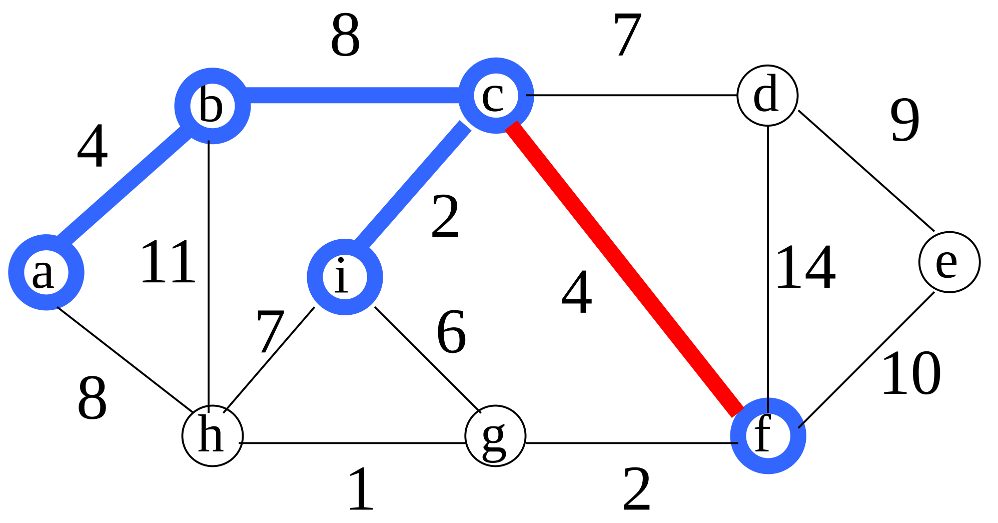
Choose b.
Parent(b)=a

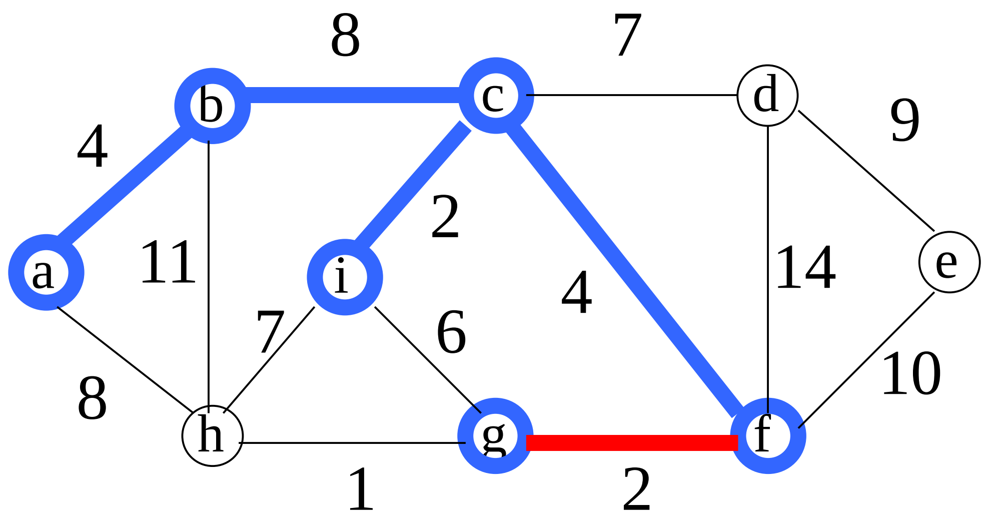
트리와 연결된 가장작은 간선선택


8
7
b
c
d
4
9
2
a
11
i
14
e
7
6
4
8
10
h
g
f
1
2

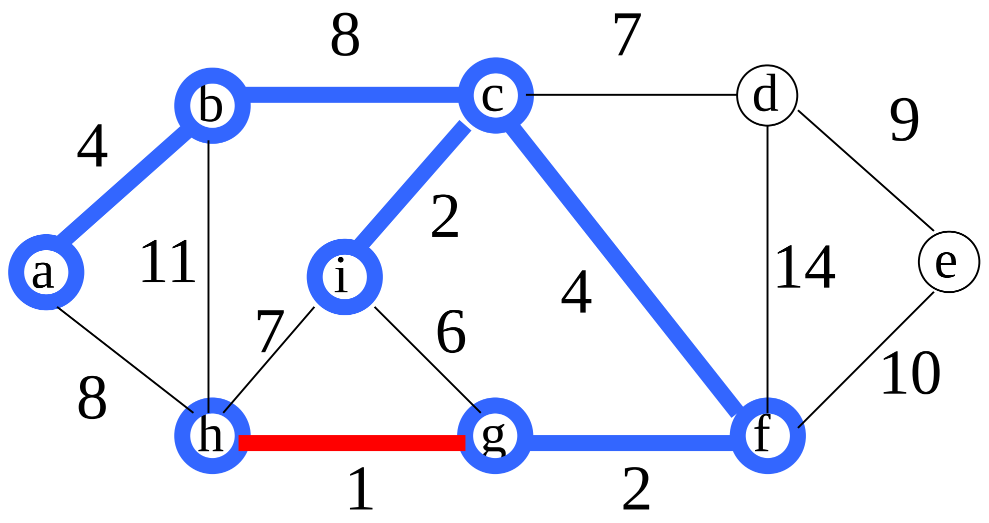

8
7
b
c
d
4
9
2
a
11
i
14
e
7
6
4
8
10
h
g
f
1
2

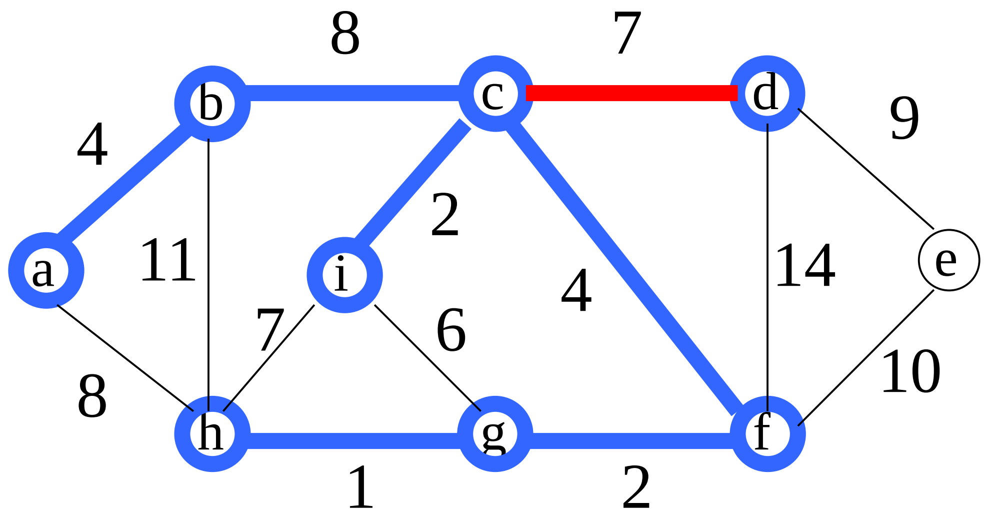
계속 하나의트리


8
7
4
9
2
11
14
4
7
6
10
8
1
2
a
b
c
d
e
f
g
h
i

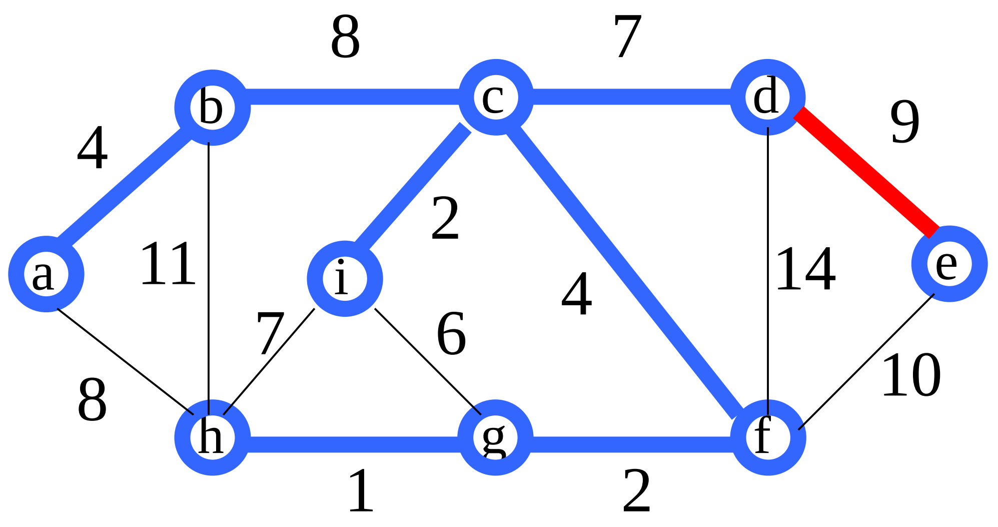

8
7
4
9
2
11
14
4
7
6
10
8
1
2
a
b
c
d
e
f
g
h
i

계속 하나의트리


8
7
b
c
d
9
4
2
a
11
i
4
14
e
7
6
10
8
h
g
f
1
2


8
7
b
c
d
9
4
2
a
11
i
4
14
e
7
6
10
8
h
g
f
1
2

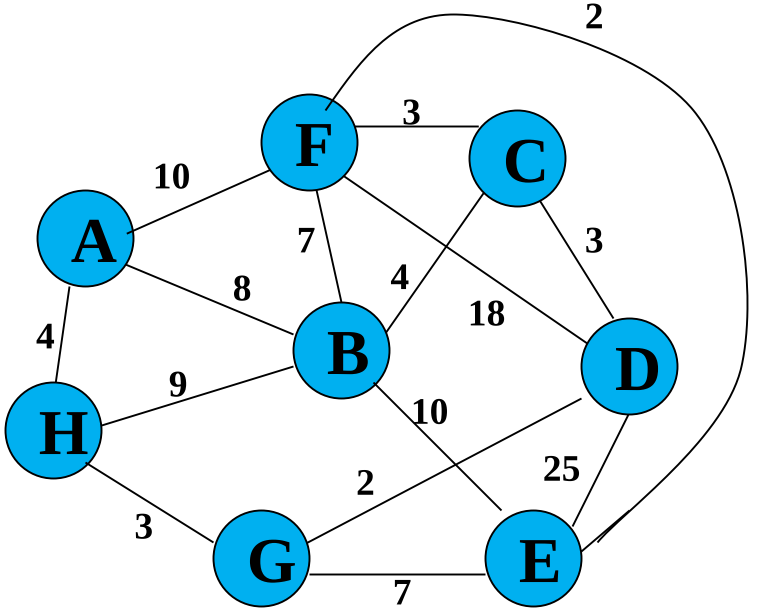
계속 하나의트리



마지막 최소 비용 간선 트리

Prim's Algorithm
프림의 알고리즘 예3

Walk-Through



우선순위큐

Initialize array

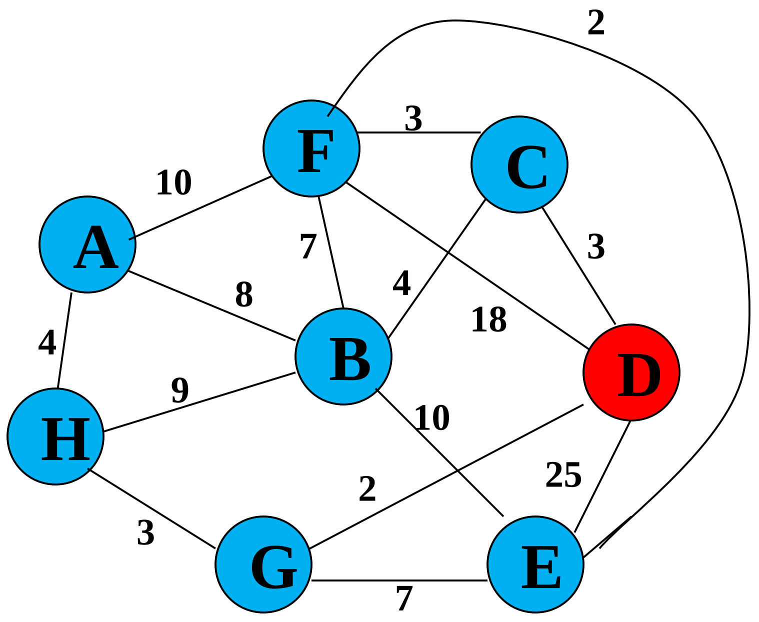
	K	d_v	p_v
A	F	∞	–
B	F	∞	–
C	F	∞	–
D	F	∞	–
E	F	∞	–
F	F	∞	–
G	F	∞	–
H	F	∞	–

P_v=출발점

d_v=거리

T : 연결됨
F : 연결되지 않음

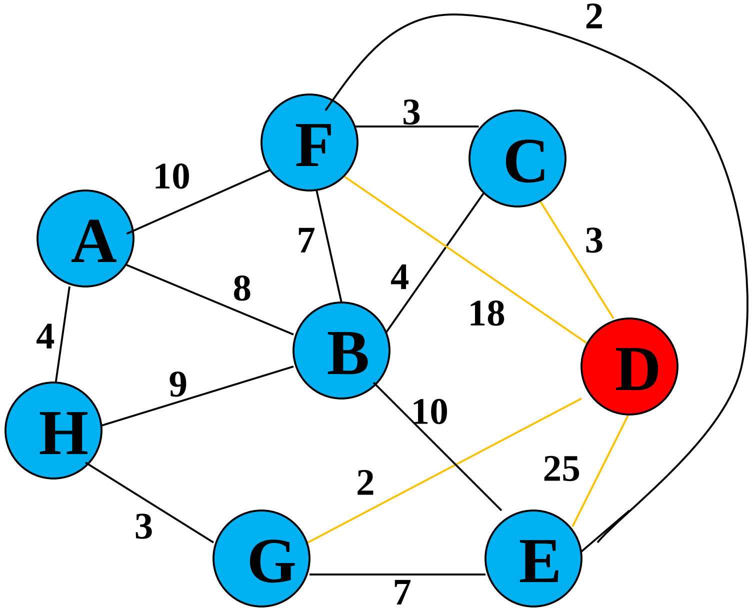
Starting node D(D시작노드)



Start with any node, say D

	K	d_v	p_v
A			
B			
C			
D	T	0	–
E			
F			
G			
H			

Starting node



Update distances of adjacent, unselected nodes

우선순위큐

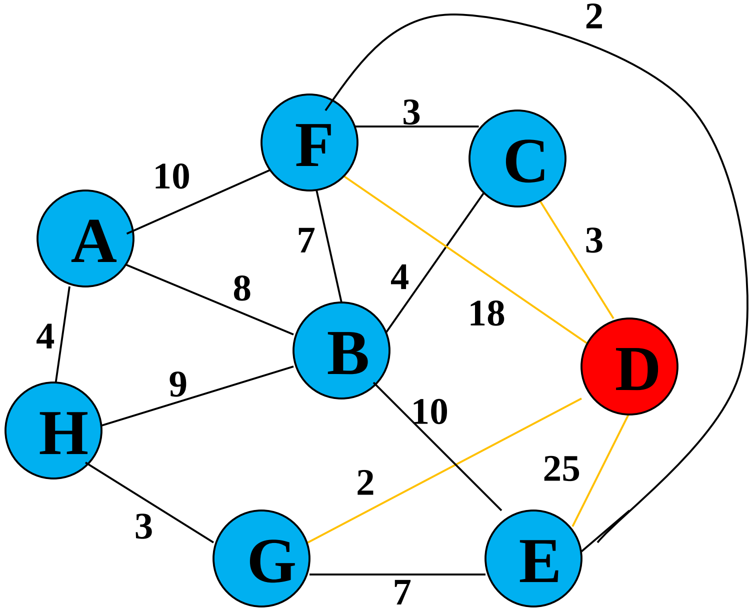
	K	d_v	p_v
A			
B			
C		3	D
D	T	0	–
E		25	D
F		18	D
G		2	D
H			

Update(수정)

Update(수정)

Update(수정)

Update(수정)

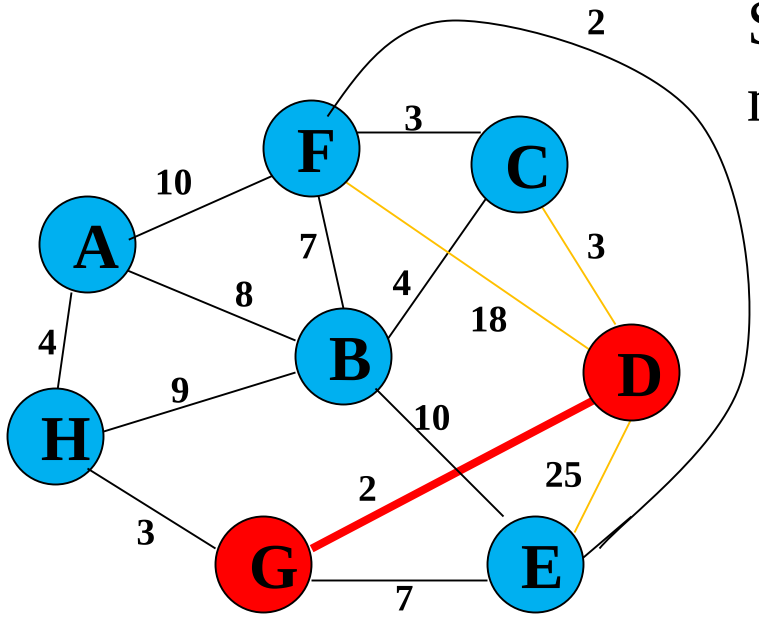


Update distances of
adjacent, unselected nodes

우선순위큐

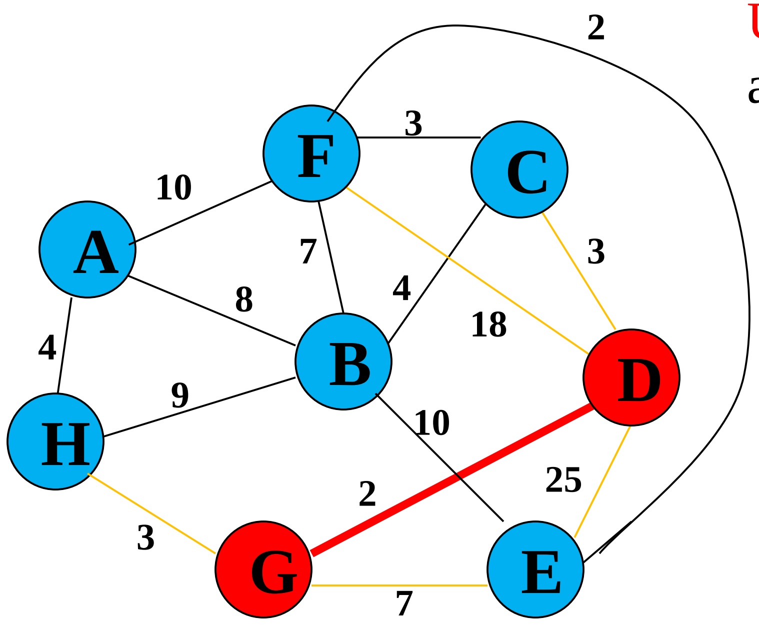
	K	d_v	p_v
A			
B			
C		3	D
D	T	0	–
E		25	D
F		18	D
G		2	D
H			

Min
가장작은가중치



Select node with minimum distance

	K	d_v	p_v
A			
B			
C		3	D
D	T	0	–
E		25	D
F		18	D
G	T	2	D
H			



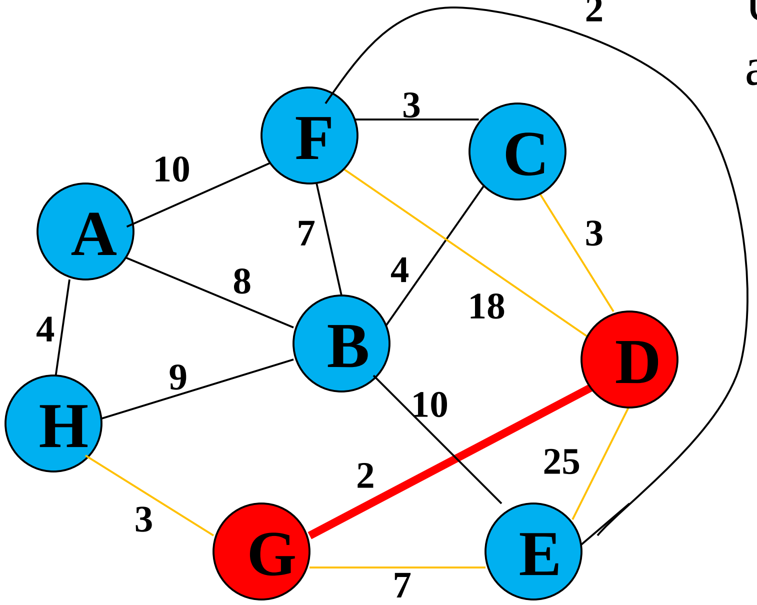
Update distances of adjacent, unselected nodes

	K	d_v	p_v
A			
B			
C		3	D
D	T	0	–
E		7/25	G
F		18	D
G	T	2	D
H		3/∞	G

Update(수정) (row E)

Update(수정) (row H)

∞ : no connection

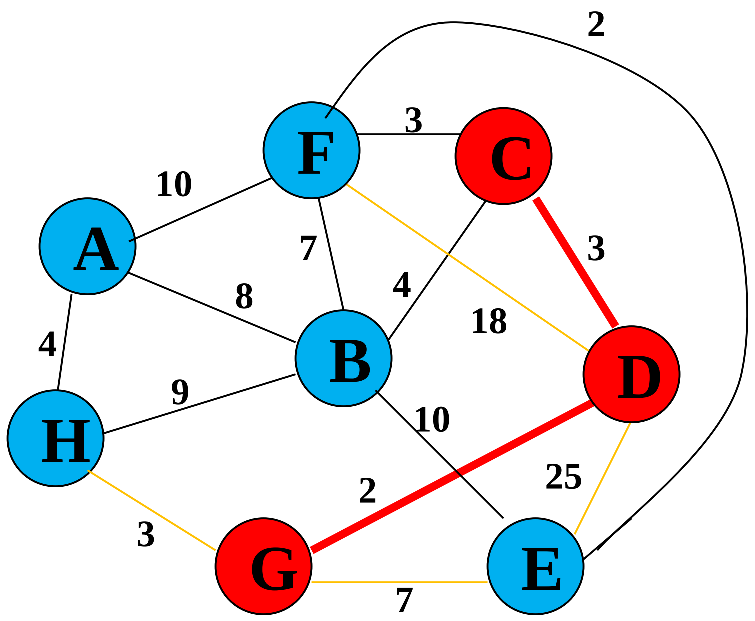


Update distances of adjacent, unselected nodes

	K	d_v	p_v
A			
B			
C		3	D
D	T	0	–
E		7	G
F		18	D
G	T	2	D
H		3	G

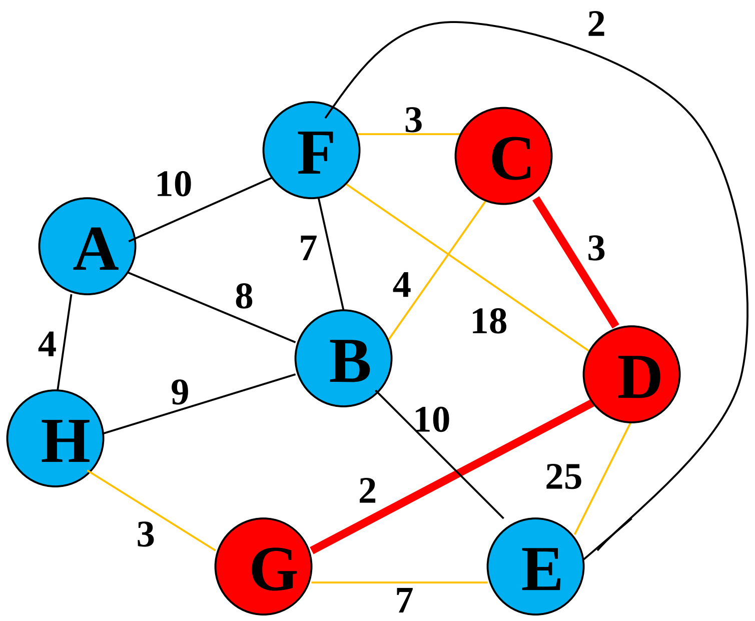
min

min



Select node with minimum distance

	K	d_v	p_v
A			
B			
C	T	3	D
D	T	0	–
E		7	G
F		18	D
G	T	2	D
H		3	G

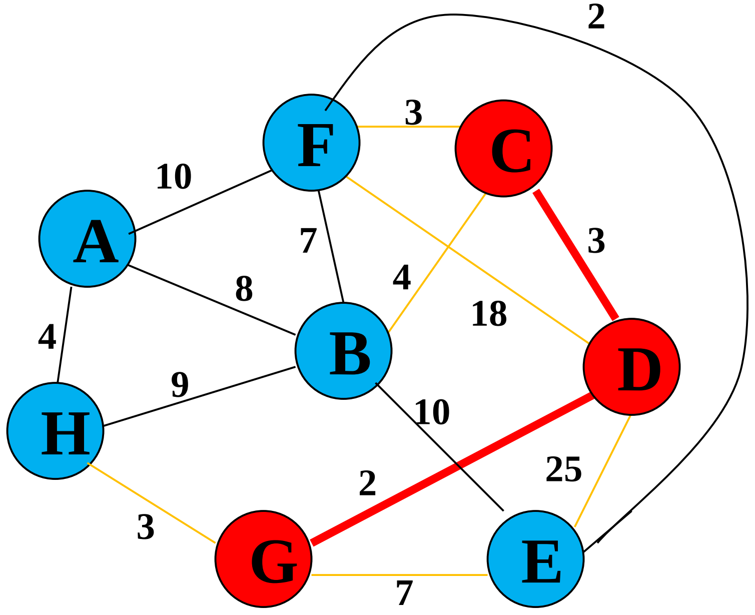


Update distances of
adjacent, unselected nodes

	K	d_v	p_v
A			
B		4/∞	C
C	T	3	D
D	T	0	–
E		7	G
F		3/18	C
G	T	2	D
H		3	G

Update(수정)

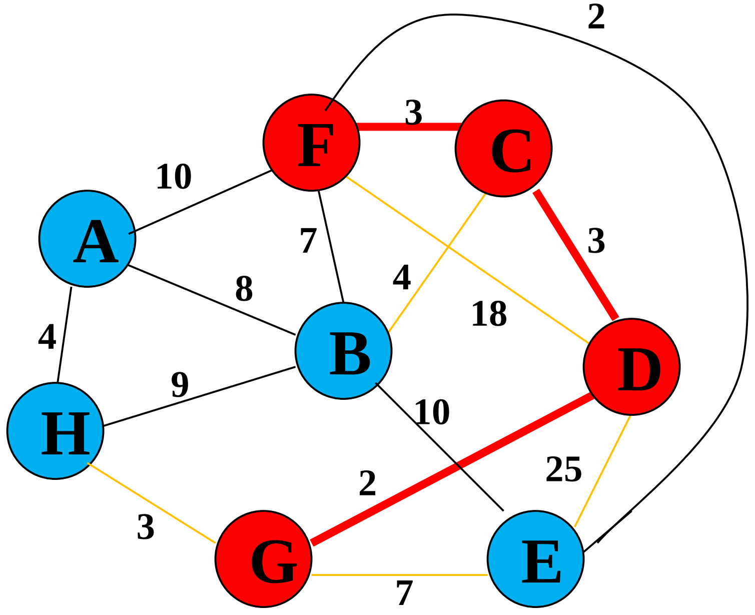
Update(수정)



Update distances of adjacent, unselected nodes

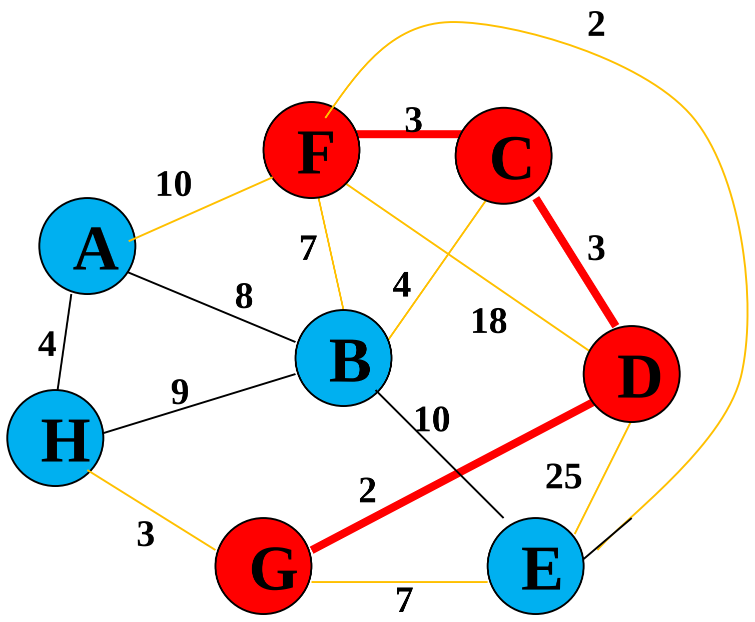
	K	d_v	p_v
A			
B		4	C
C	T	3	D
D	T	0	–
E		7	G
F		3	C
G	T	2	D
H		3	G

min



Select node with minimum distance

	K	d_v	p_v
A			
B		4	C
C	T	3	D
D	T	0	–
E		7	G
F	T	3	C
G	T	2	D
H		3	G

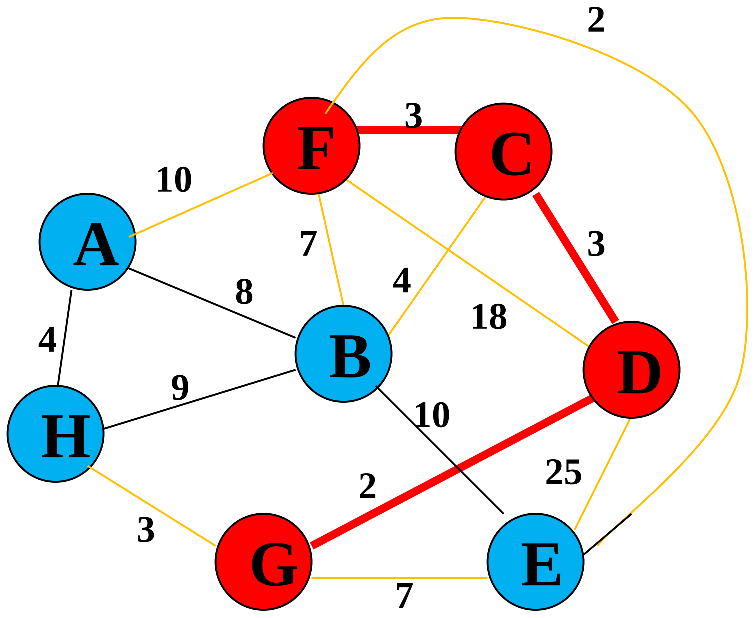


Update distances of
adjacent, unselected nodes

	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E		2	F
F	T	3	C
G	T	2	D
H		3	G

Update(수정)

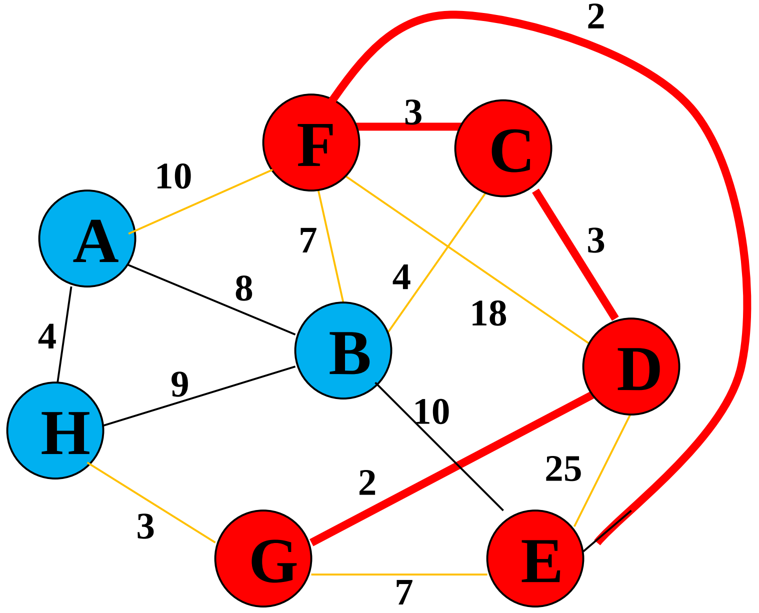
Update(수정)



Update distances of adjacent, unselected nodes

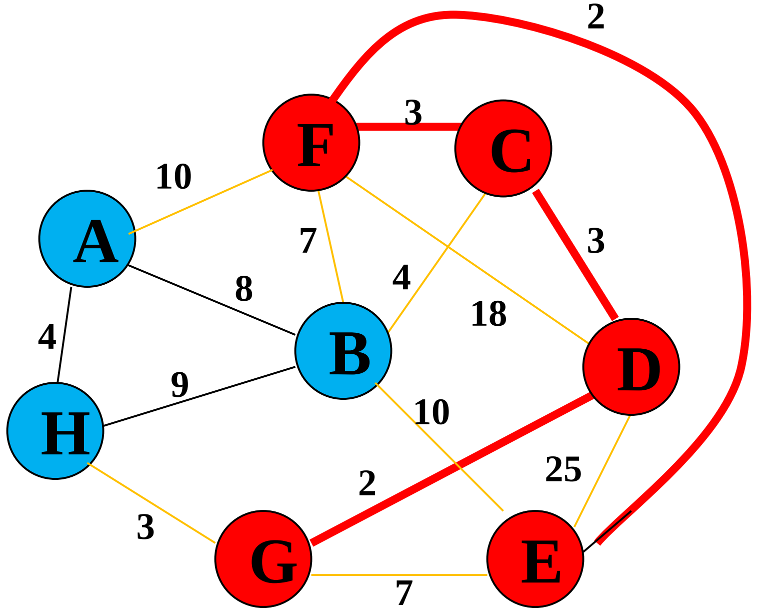
	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E		2	F
F	T	3	C
G	T	2	D
H		3	G

min



Select node with minimum distance

	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H		3	G

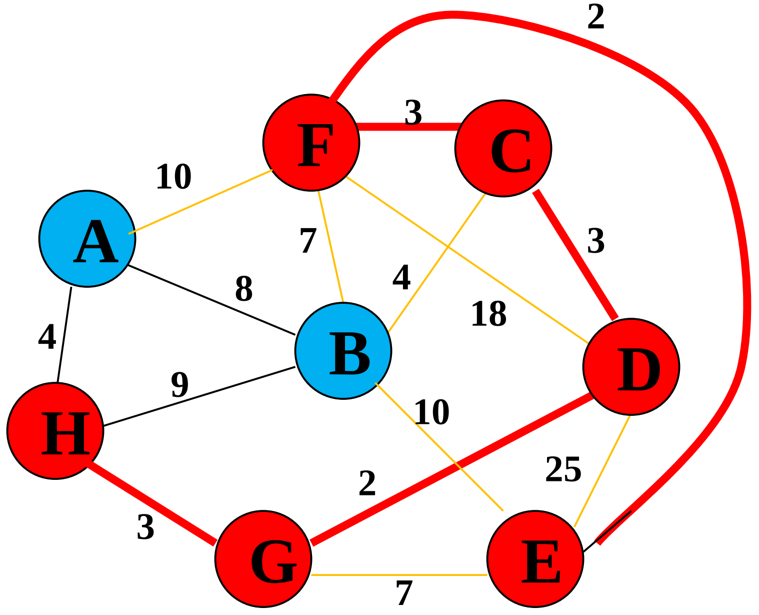


Update distances of adjacent, unselected nodes

	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H		3	G

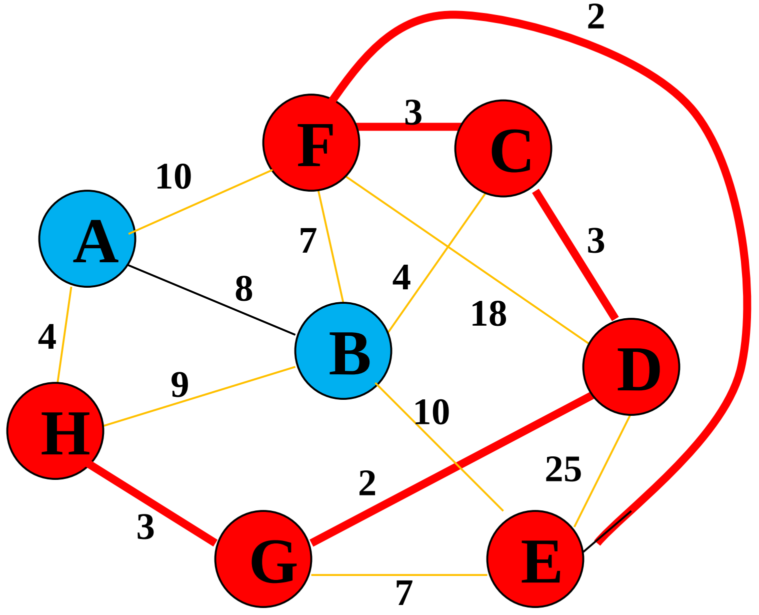
min

Table entries unchanged(변동없음)



Select node with minimum distance

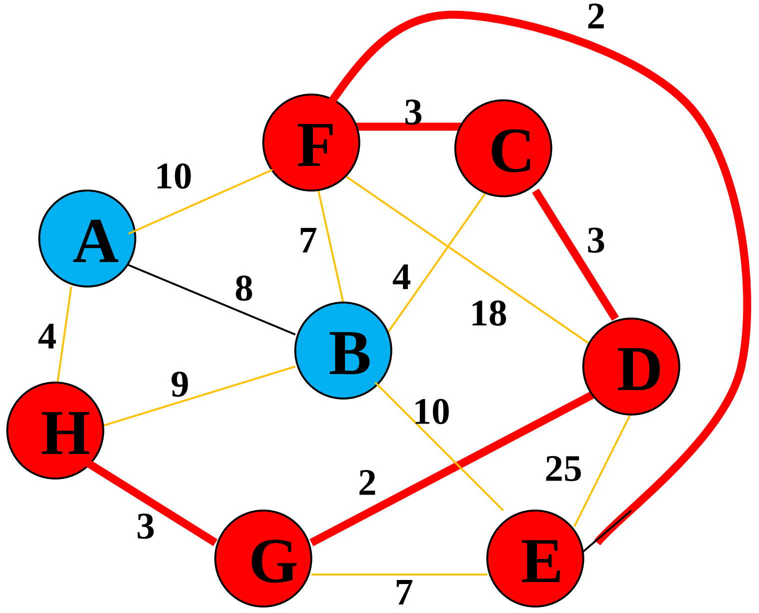
	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G



Update distances of
adjacent, unselected nodes

	K	d_v	p_v
A		4	H
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

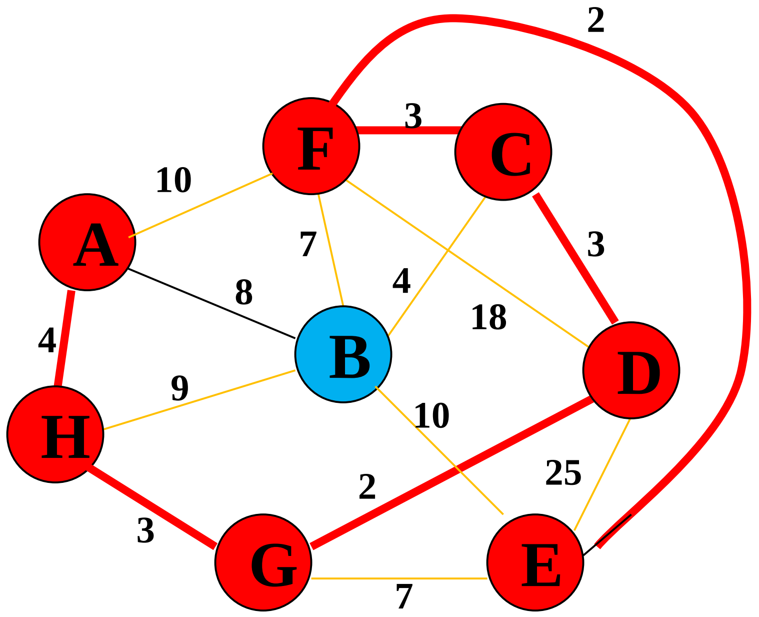
Update(수정)



Update distances of adjacent, unselected nodes

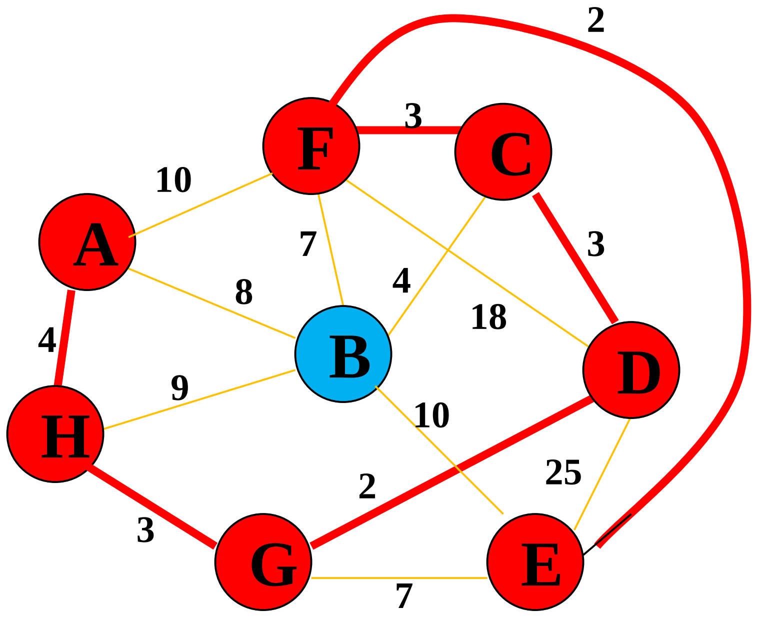
	K	d_v	p_v
A		4	H
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

min



Select node with minimum distance

	K	d_v	p_v
A	T	4	H
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

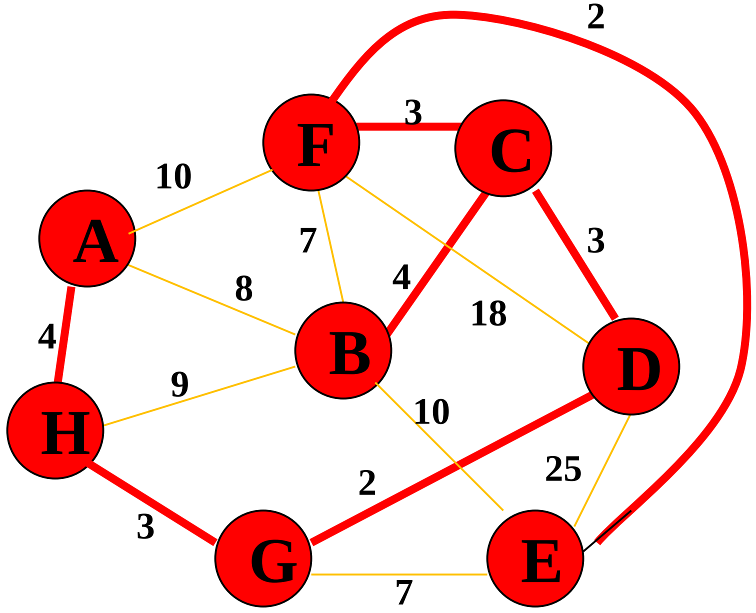


Update distances of
adjacent, unselected nodes

	K	d_v	p_v
A	T	4	H
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

min

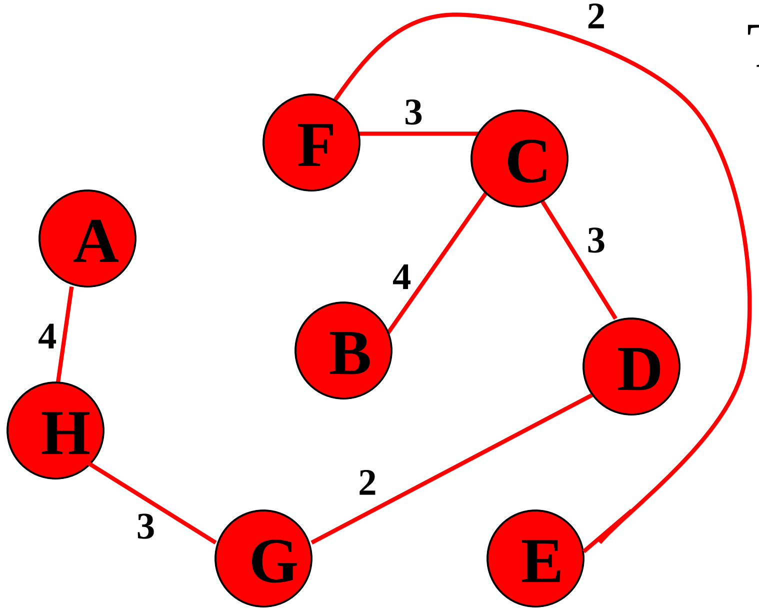
Table entries
unchanged



Select node with minimum distance

	K	d_v	p_v
A	T	4	H
B	T	4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

Cost of Minimum Spanning Tree = $\Sigma\, d_v$ = 21



	K	d_v	p_v
A	T	4	H
B	T	4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

= 21

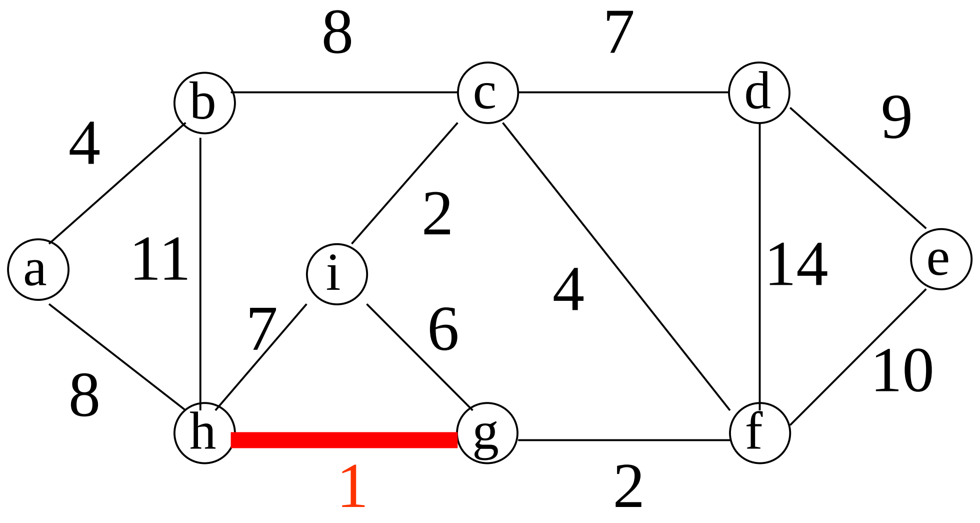
Done

최소비용신장트리
Minimum Spanning Tree

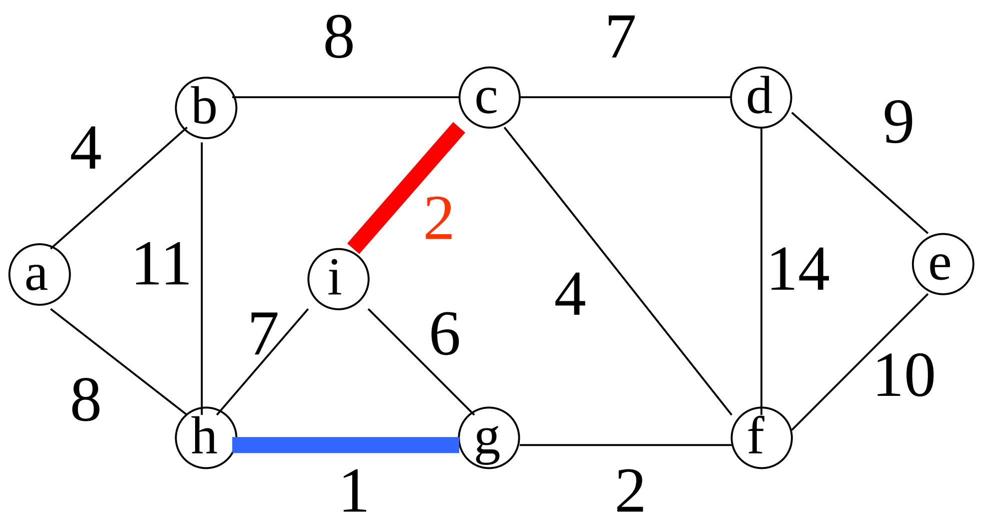
Kruskal’s Algorithm
크루스칼의 알고리즘 사용 예제

The execution of Kruskal's algorithm

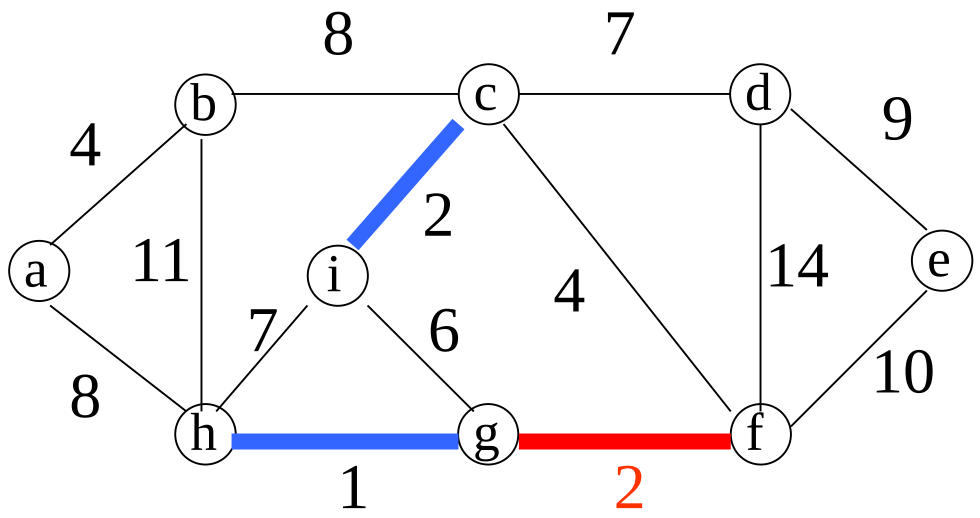
- The edges are considered by the algorithm in sorted order by weight. 간선을 가중치가 작은순으로 정렬
- The edge under consideration at each step is shown with a red weight number. 매 스텝마다 빨강간선이 가중치가 가장 작음



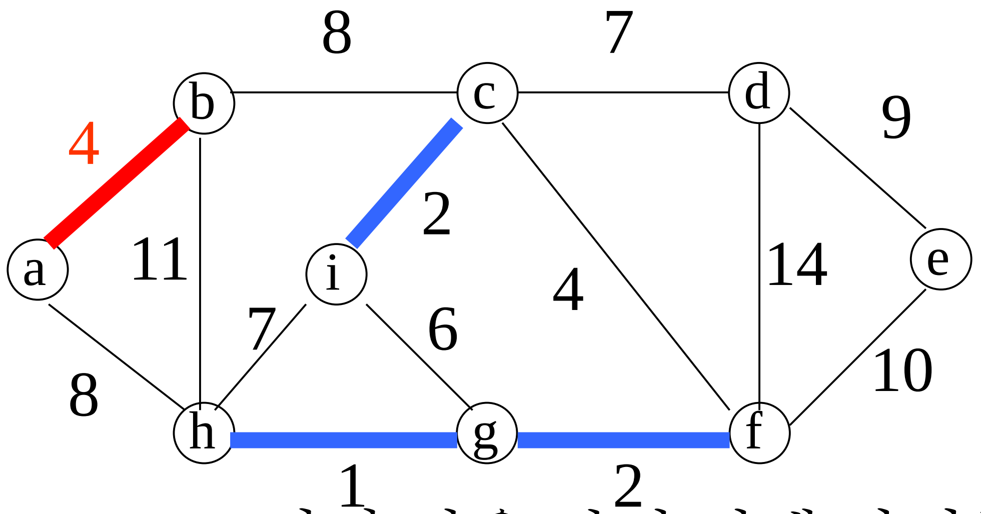
1. Select (h-g) (w=1)가장 작은 간선 선택:사이클만들지않음



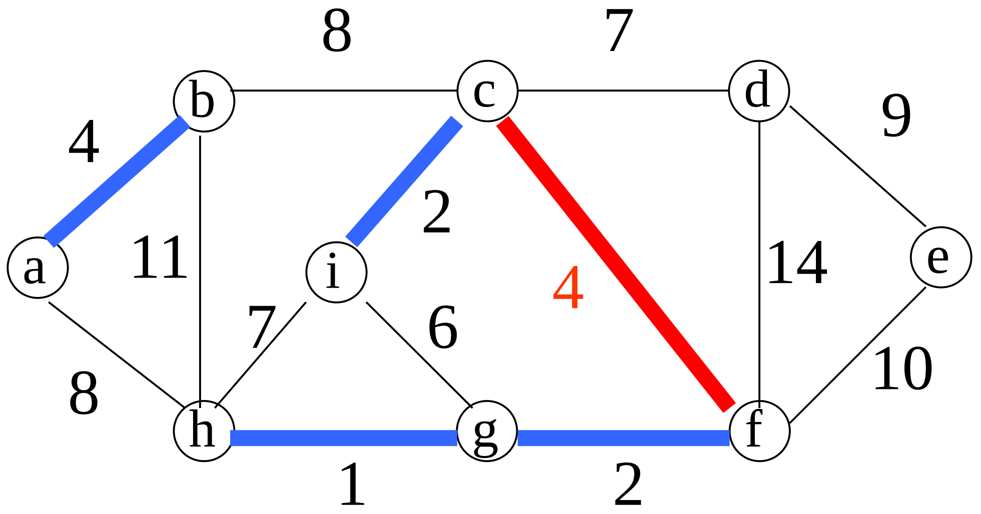
2. Select (c-i) (w=2)가장 작은 간선 선택 :사이클만들지않음
현재상태는 나무가 2개인 숲.



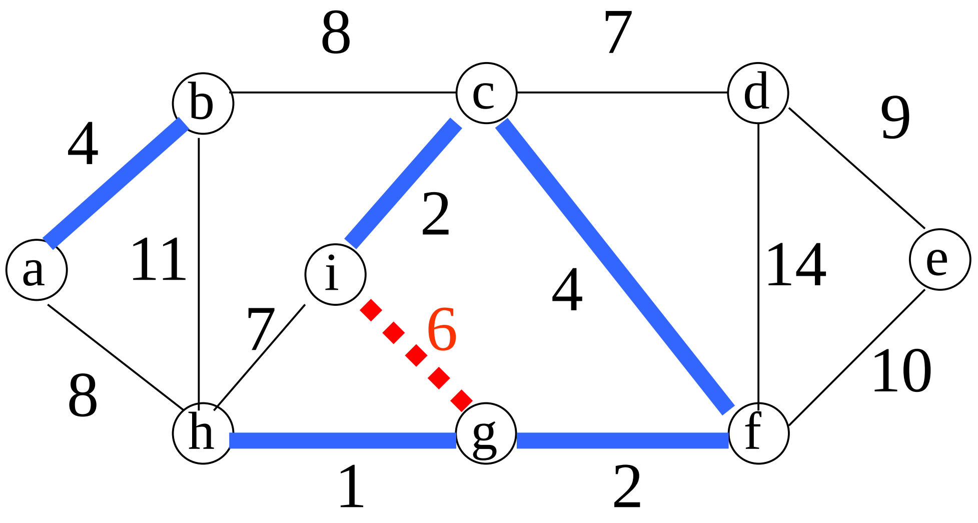
3. Select (f-g) (w=2)가장 작은 간선 선택: 사이클만들지 않음



4. Select (a-b) (w=4)가장 작은 간선 선택:사이클만들지않음
나무가 세 개인 숲

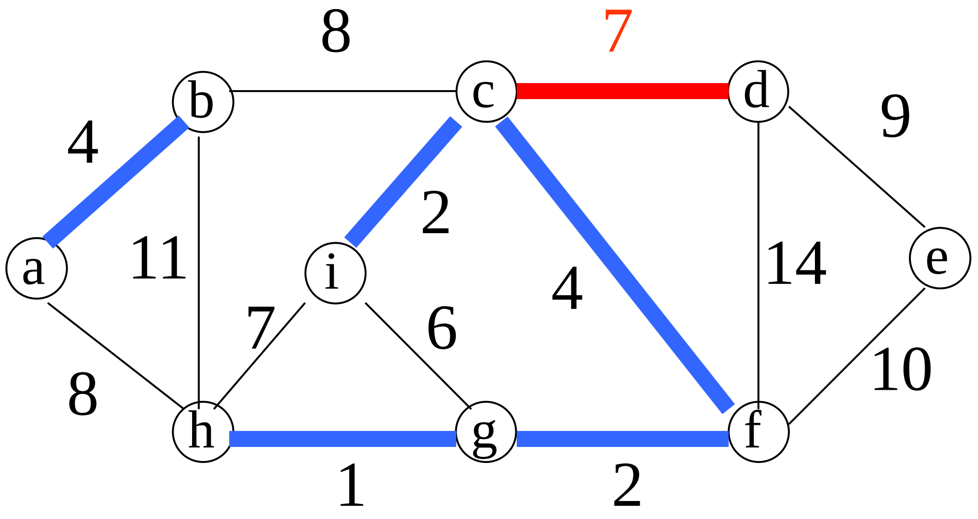


5. Select (c-f) (w=4)가장 작은 간선 선택:사이클만들지않음

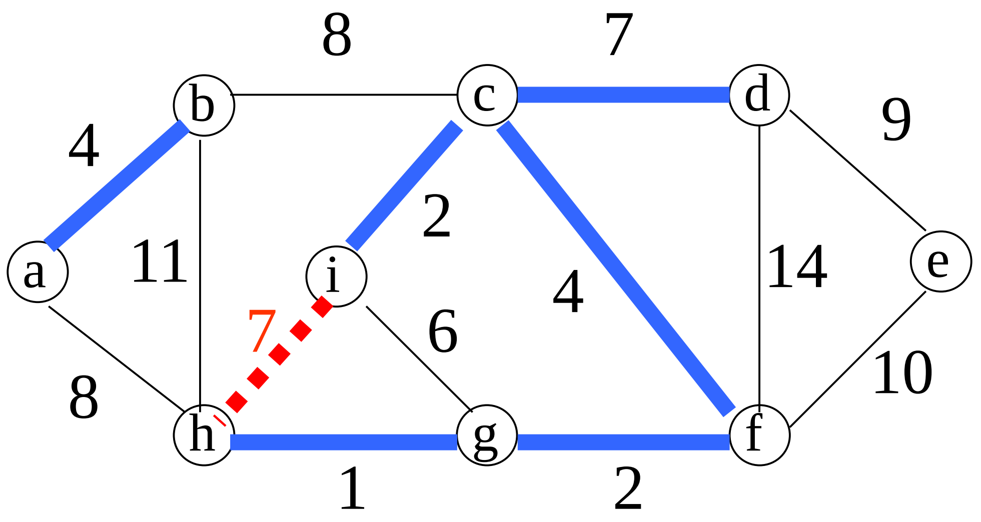


사이클생성
버린다

6. Not select (g-i) (w=4) cycle(사이클만듬:답이아님)

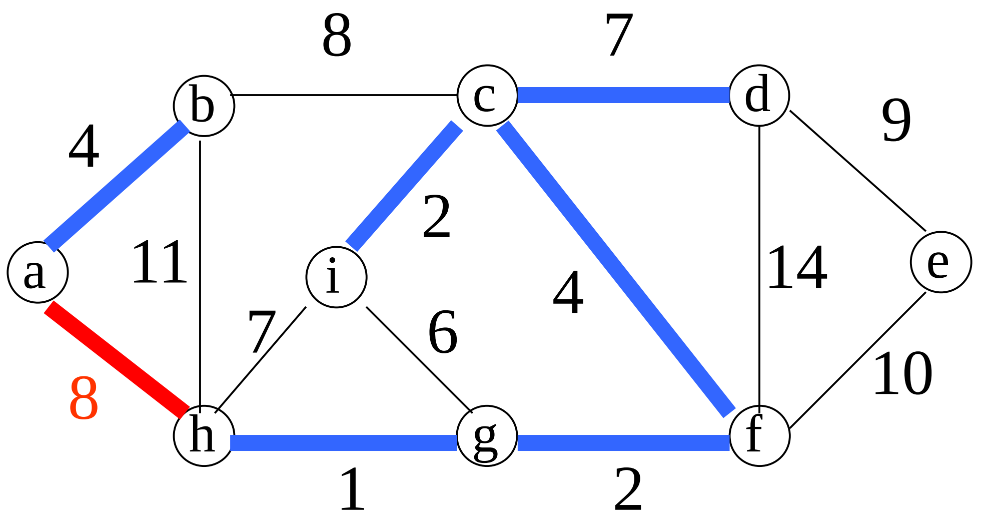


7. Select (c-d) (w=7) : 사이클만들지않음

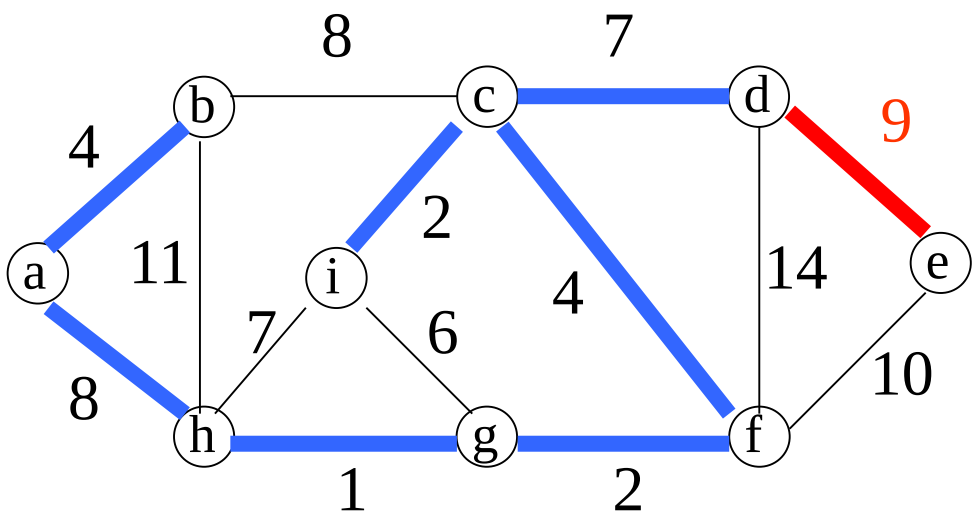


사이클생성
버린다

8. Not select (h-i) cycle(사이클만듬) :답이아님



8. select (h-a) :사이클만들지않음



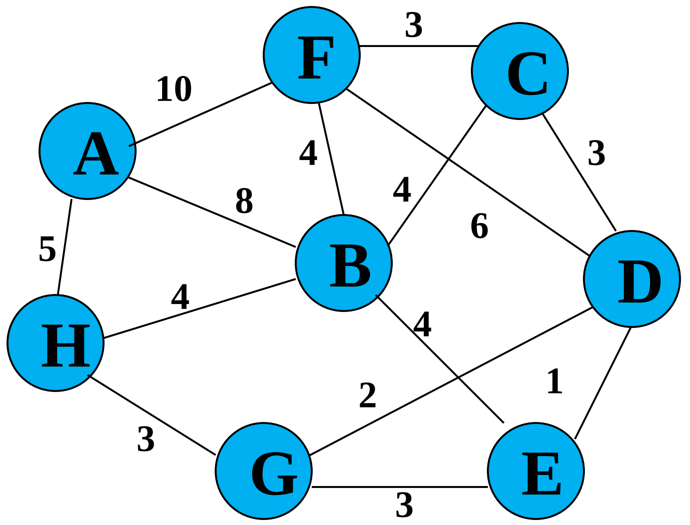
9. select (d-e) : 사이클만들지않음. 정점8개,7개의간선얻음 알고리즘 끝

Cost of Minimum Spanning Tree = Σ d_v = **37**

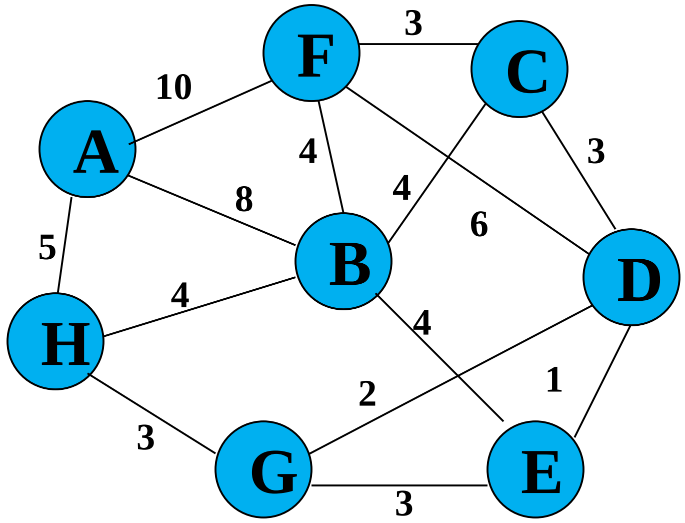
가중치의 합이 37인 최소비용신장트리를 얻음.

Kruskal’s Algorithm
크루스칼의 알고리즘 예 2

Walk-Through(검토)



Consider an undirected, weight graph

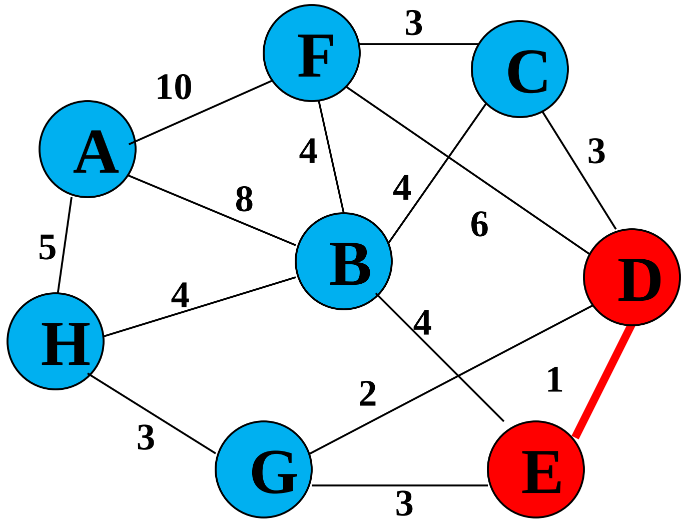


Sort the edges by increasing edge weight

간선의 길이순으로 정렬

edge	d_v	
(D,E)	1	
(D,G)	2	
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

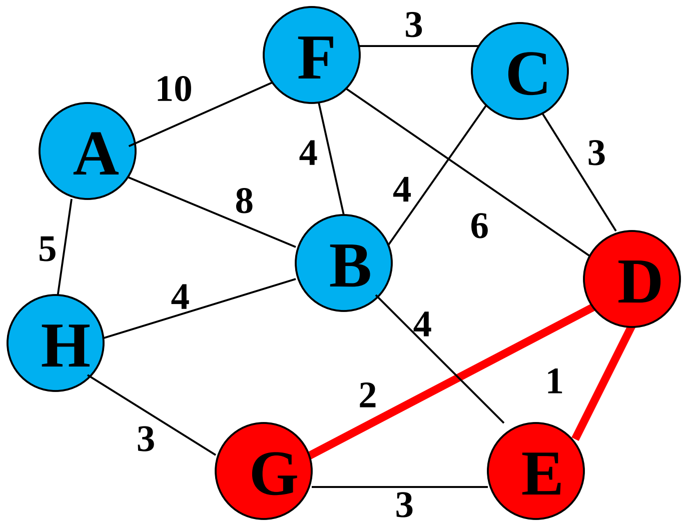
edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

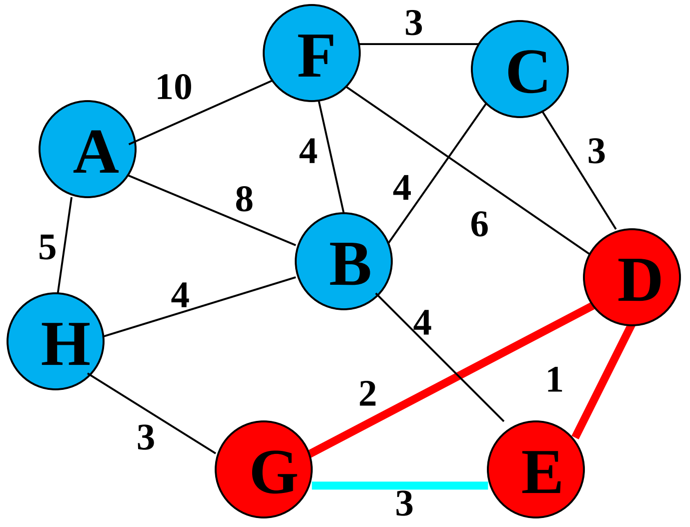
edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



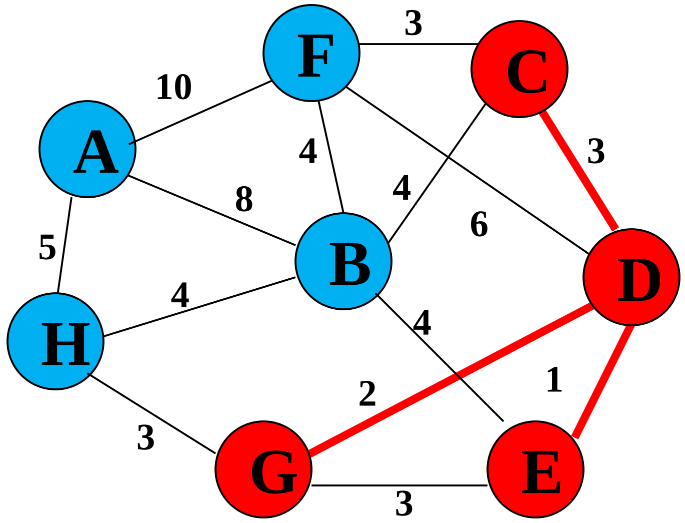
Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Accepting edge (E,G) would create a cycle

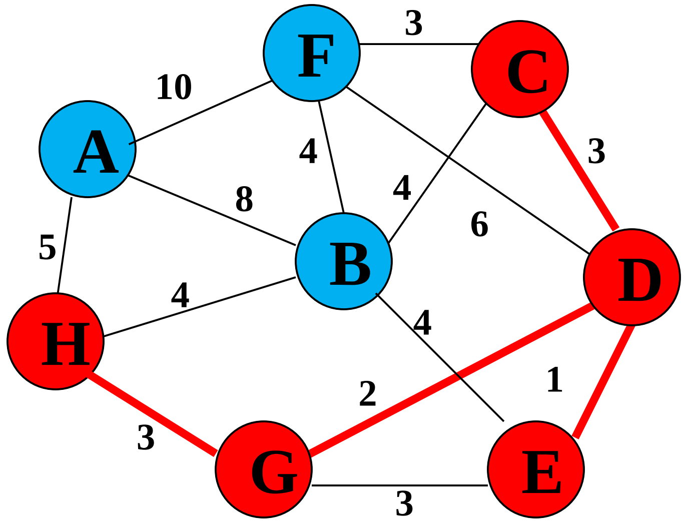
χ 사이클을 만들기 때문에 해가 안됨



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	
(C,F)	3	
(B,C)	4	

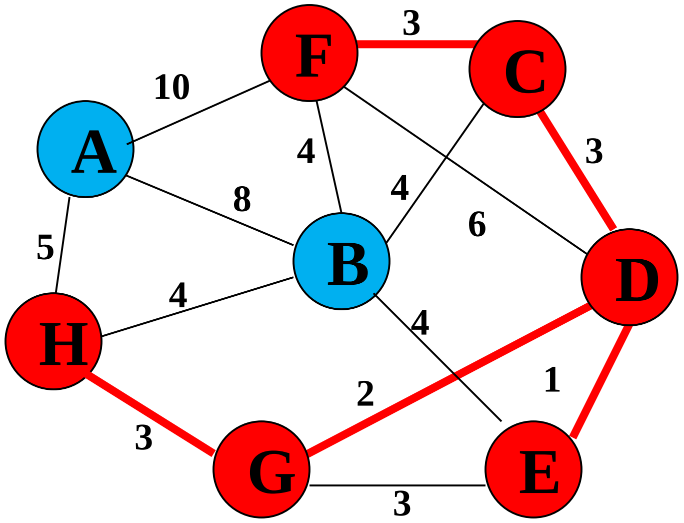
edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	
(B,C)	4	

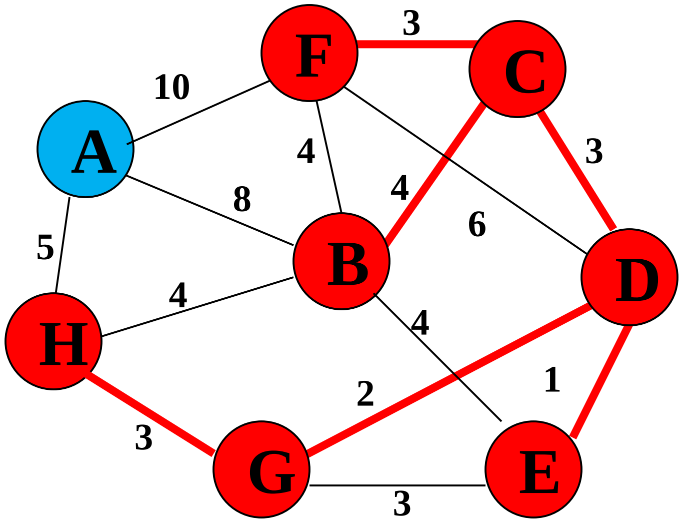
edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	

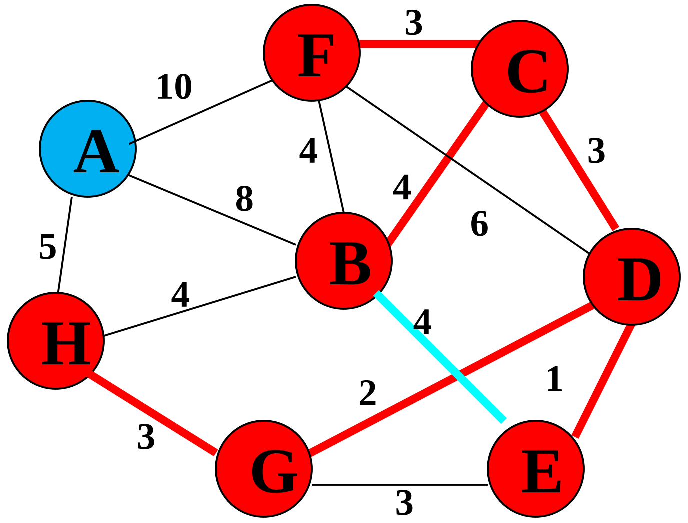
edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

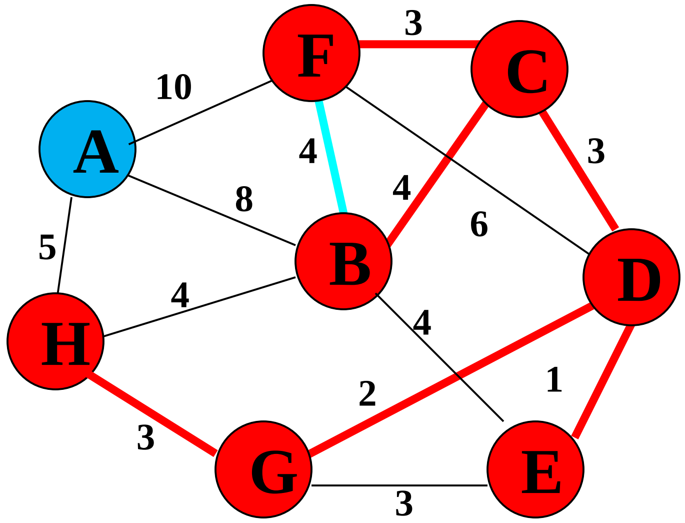


Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√

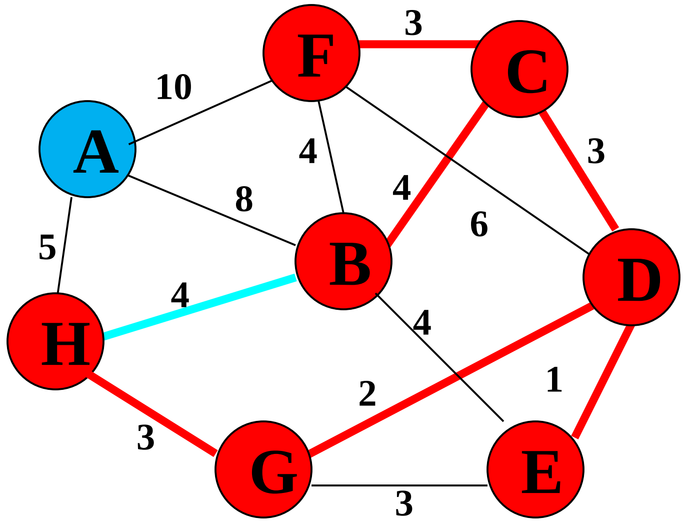
edge	d_v	
(B,E)	4	χ
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Select first |V|–1 edges which do not generate a cycle



edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√

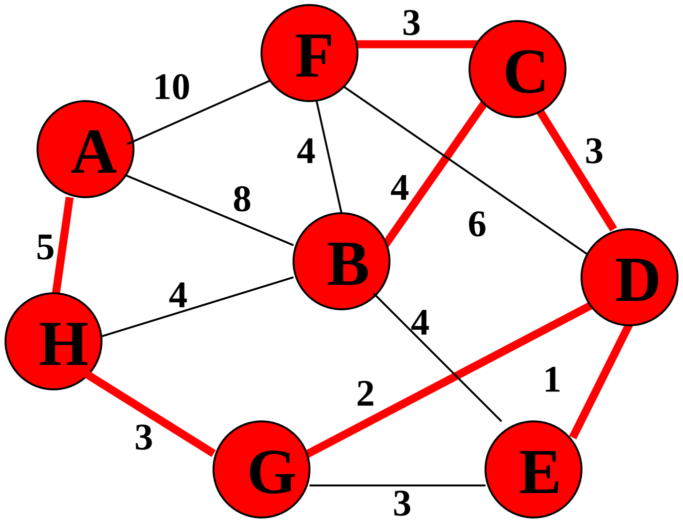
edge	d_v	
(B,E)	4	χ
(B,F)	4	χ
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√

edge	d_v	
(B,E)	4	χ
(B,F)	4	χ
(B,H)	4	χ
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

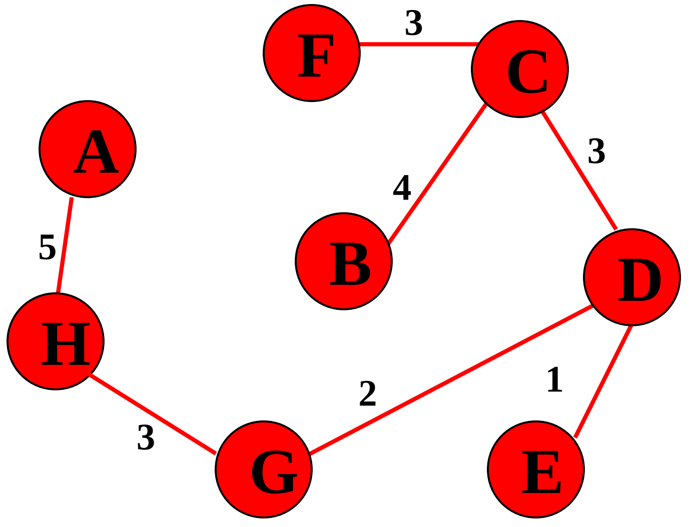


Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√

edge	d_v	
(B,E)	4	χ
(B,F)	4	χ
(B,H)	4	χ
(A,H)	5	√
(D,F)	6	
(A,B)	8	
(A,F)	10	

Select first |V|–1 edges which do not generate a cycle



edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√

edge	d_v	
(B,E)	4	χ
(B,F)	4	χ
(B,H)	4	χ
(A,H)	5	√
(D,F)	6	
(A,B)	8	
(A,F)	10	

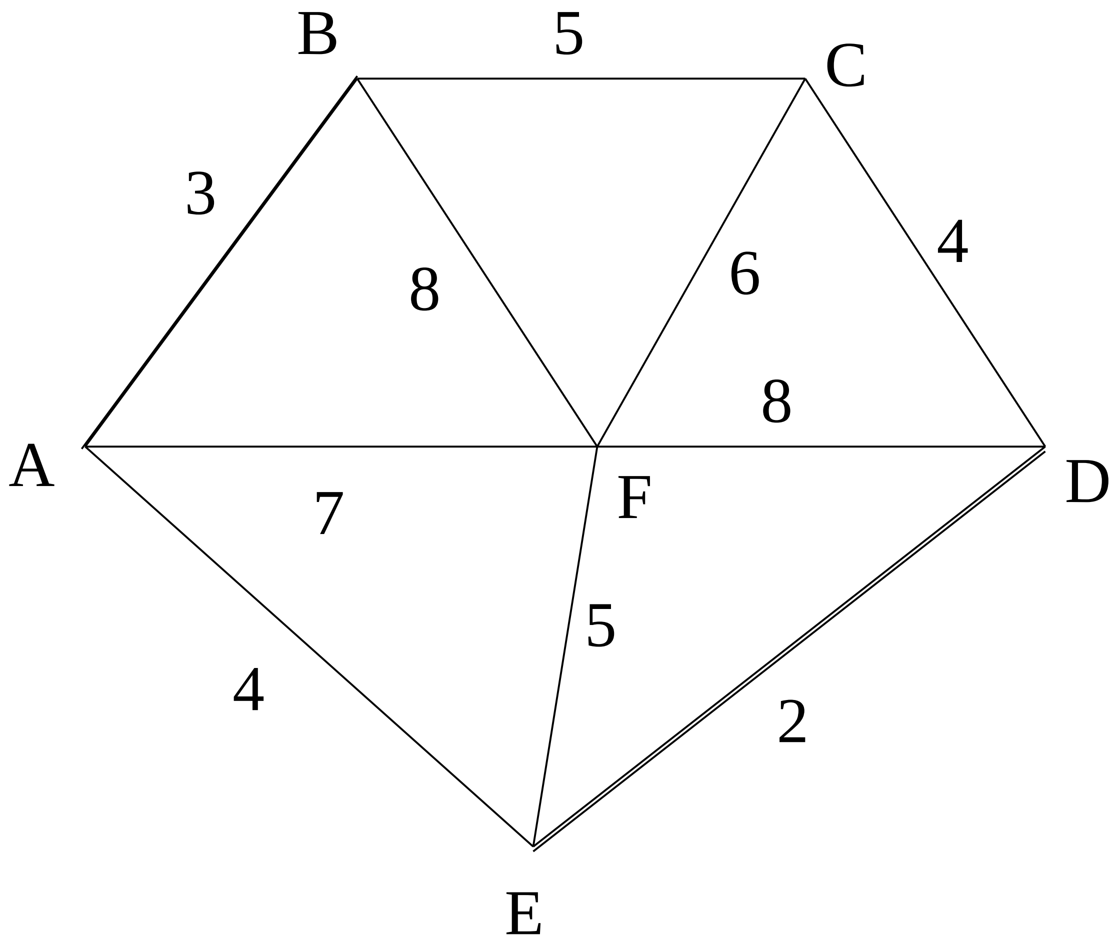
} not considered

Done

Total Cost = $\Sigma\, d_v = 21$

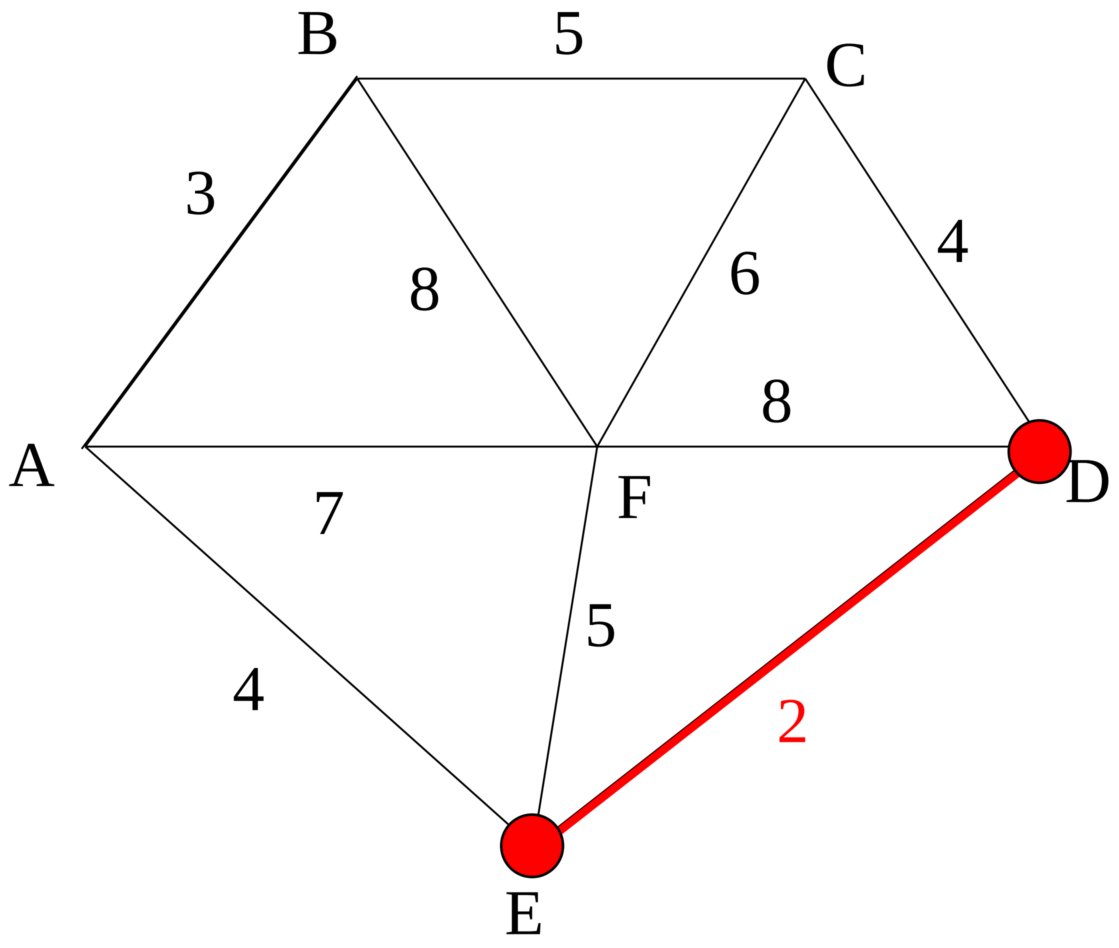
두 알고리즘의 예3

Kruskal's Algorithm



Select the next shortest edge which does not create a cycle

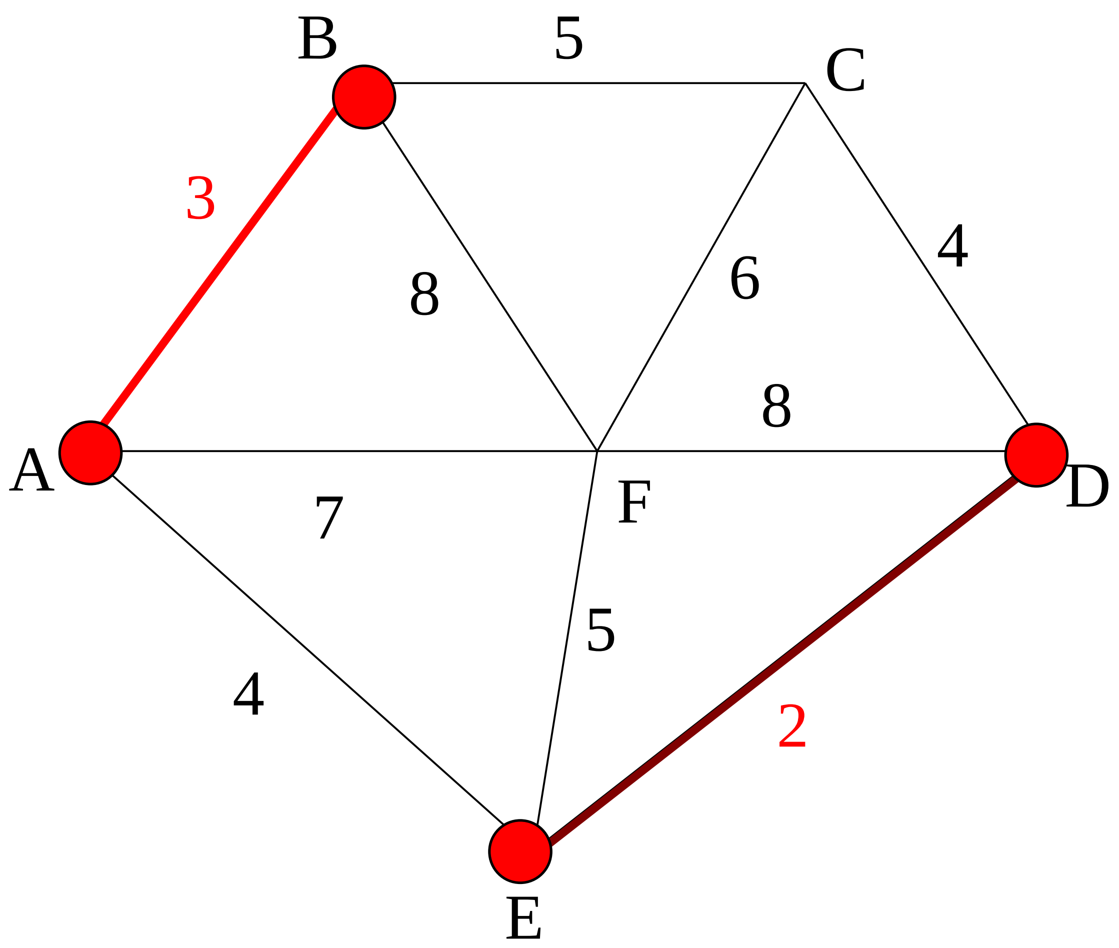
Kruskal's Algorithm



Select the next shortest edge which does not create a cycle

ED 2

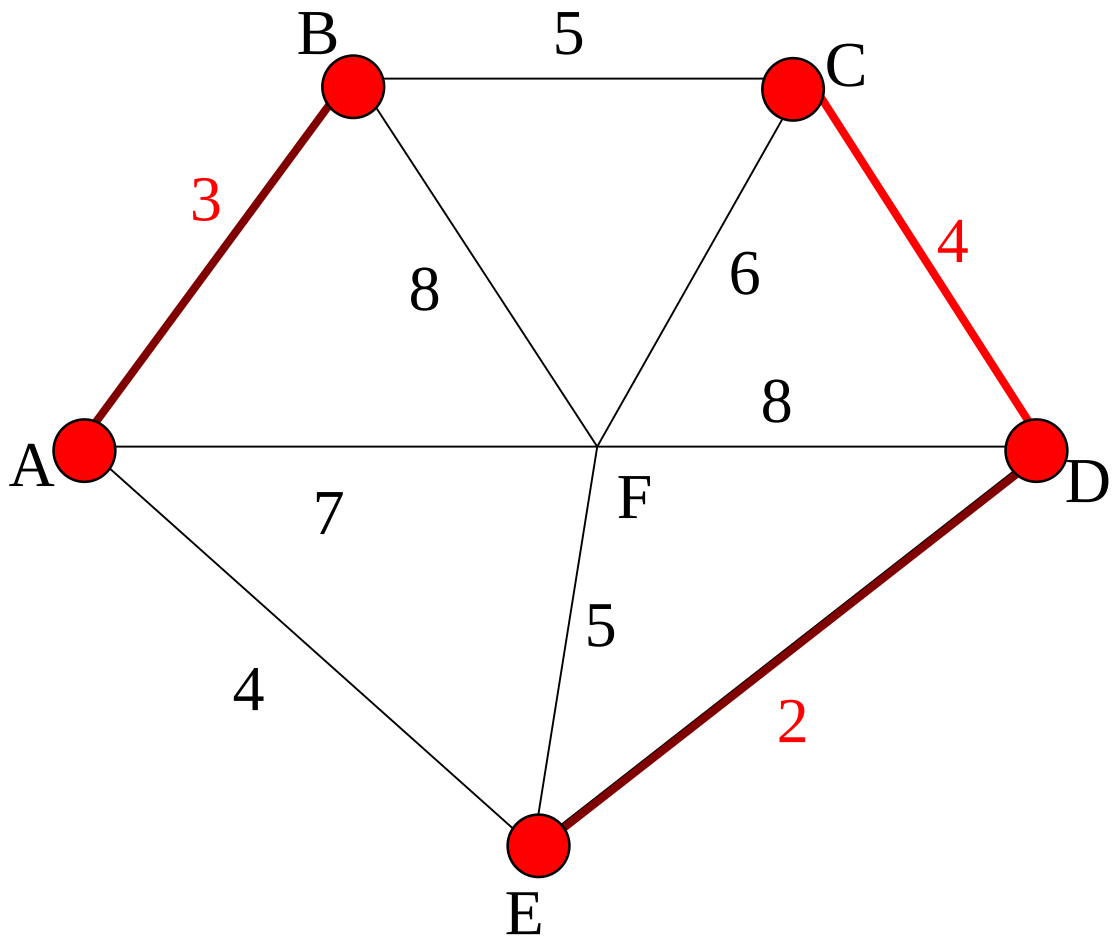
Kruskal's Algorithm



Select the next shortest edge which does not create a cycle

ED 2
AB 3

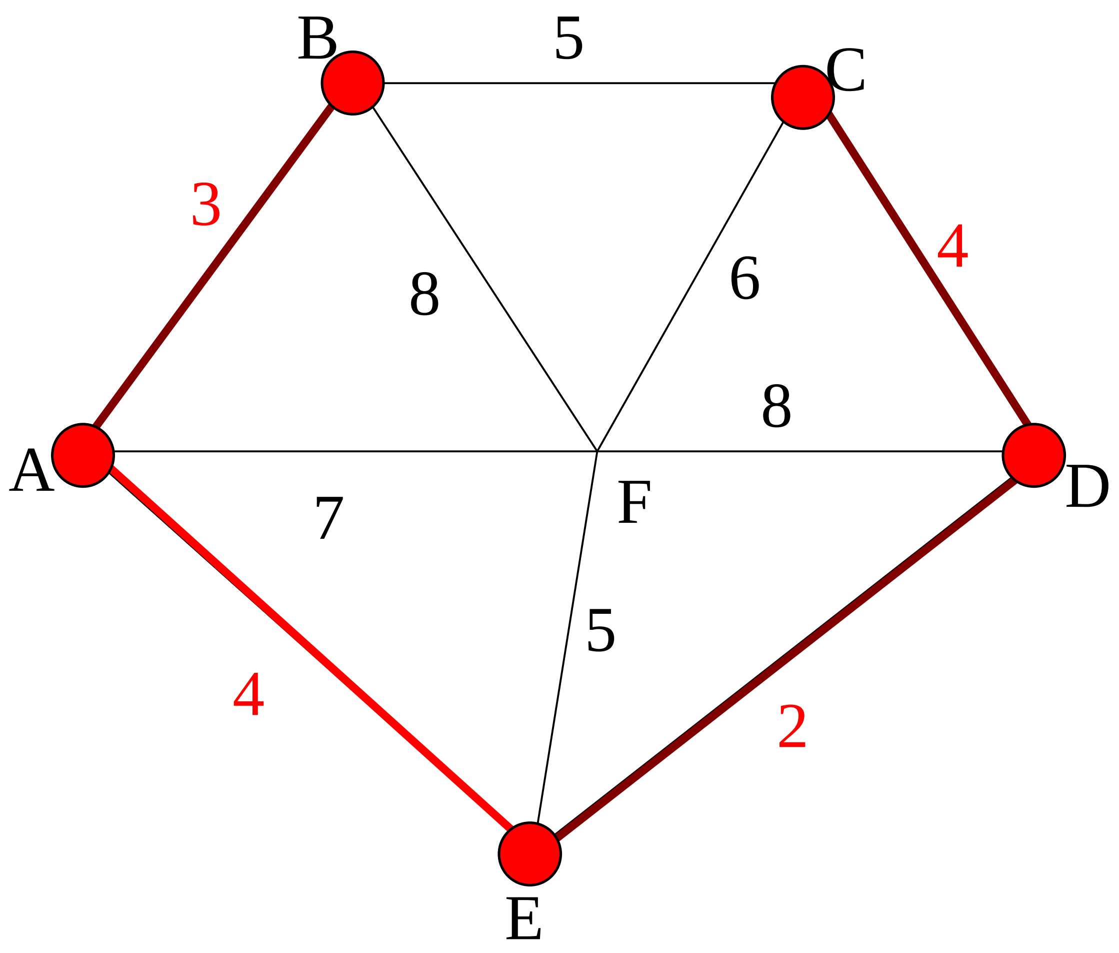
Kruskal's Algorithm



Select the next shortest edge which does not create a cycle

ED 2
AB 3
CD 4 (or AE 4)

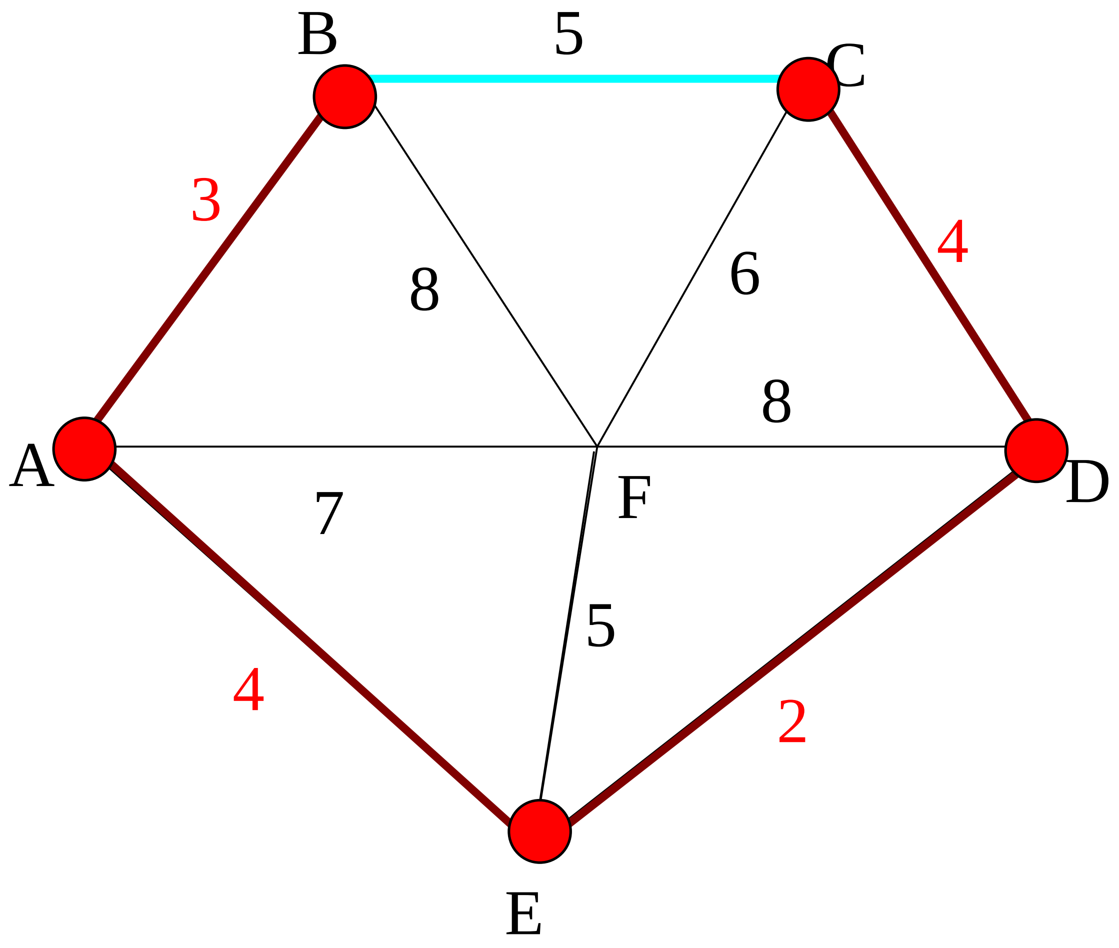
Kruskal’s Algorithm



Select the next shortest edge which does not create a cycle

ED 2
AB 3
CD 4
AE 4

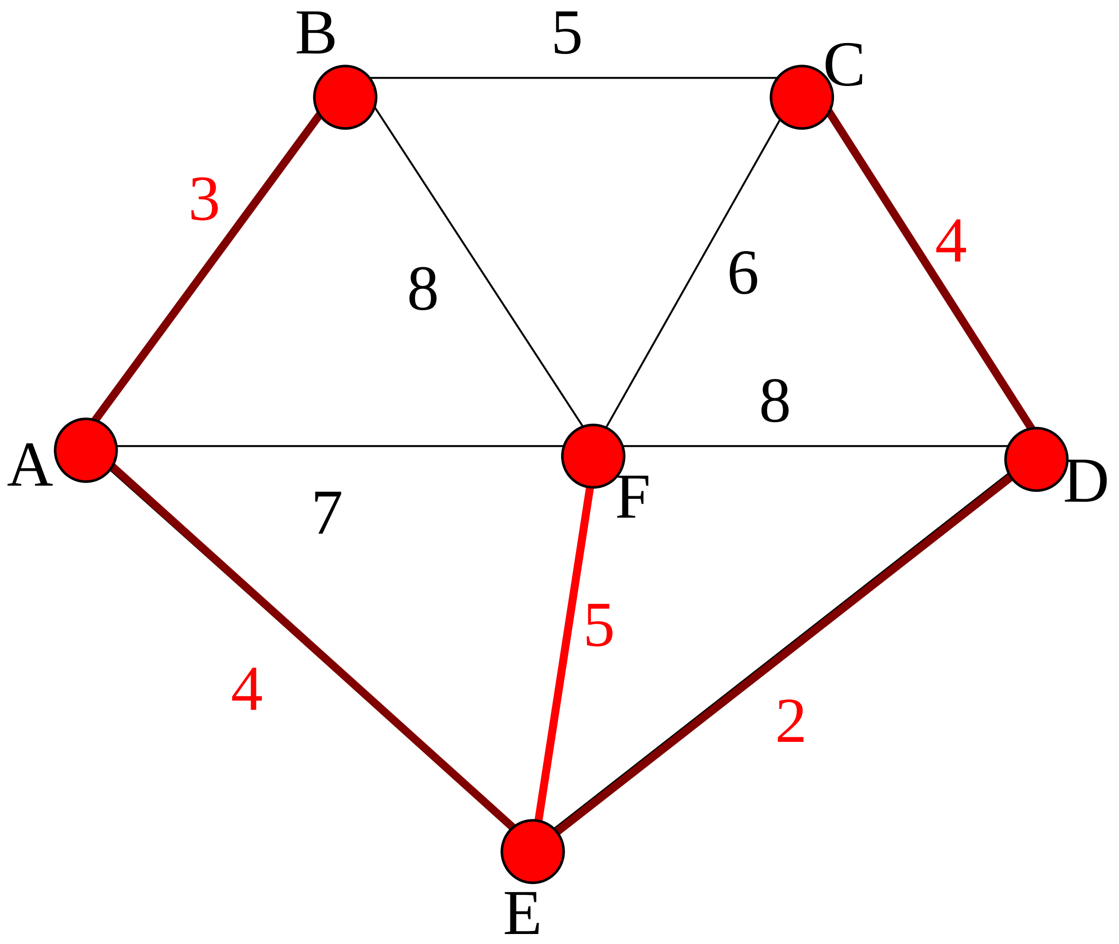
Kruskal's Algorithm



Select the next shortest edge which does not create a cycle

ED 2
AB 3
CD 4
AE 4
BC 5 – forms a cycle
안됨

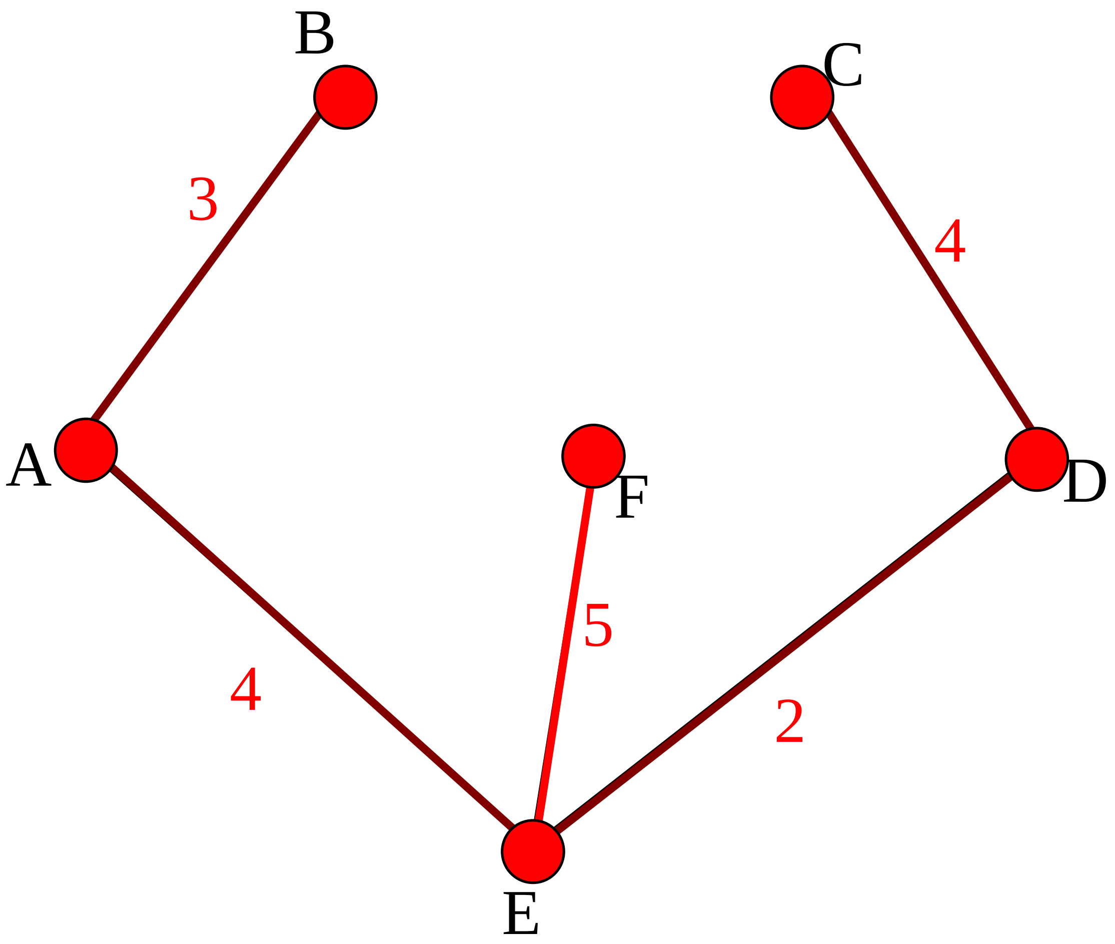
Kruskal's Algorithm



Select the next shortest edge which does not create a cycle

ED	**2**
AB	**3**
CD	**4**
AE	**4**
BC	**5 – forms a cycle**
EF	**5**

Kruskal’s Algorithm



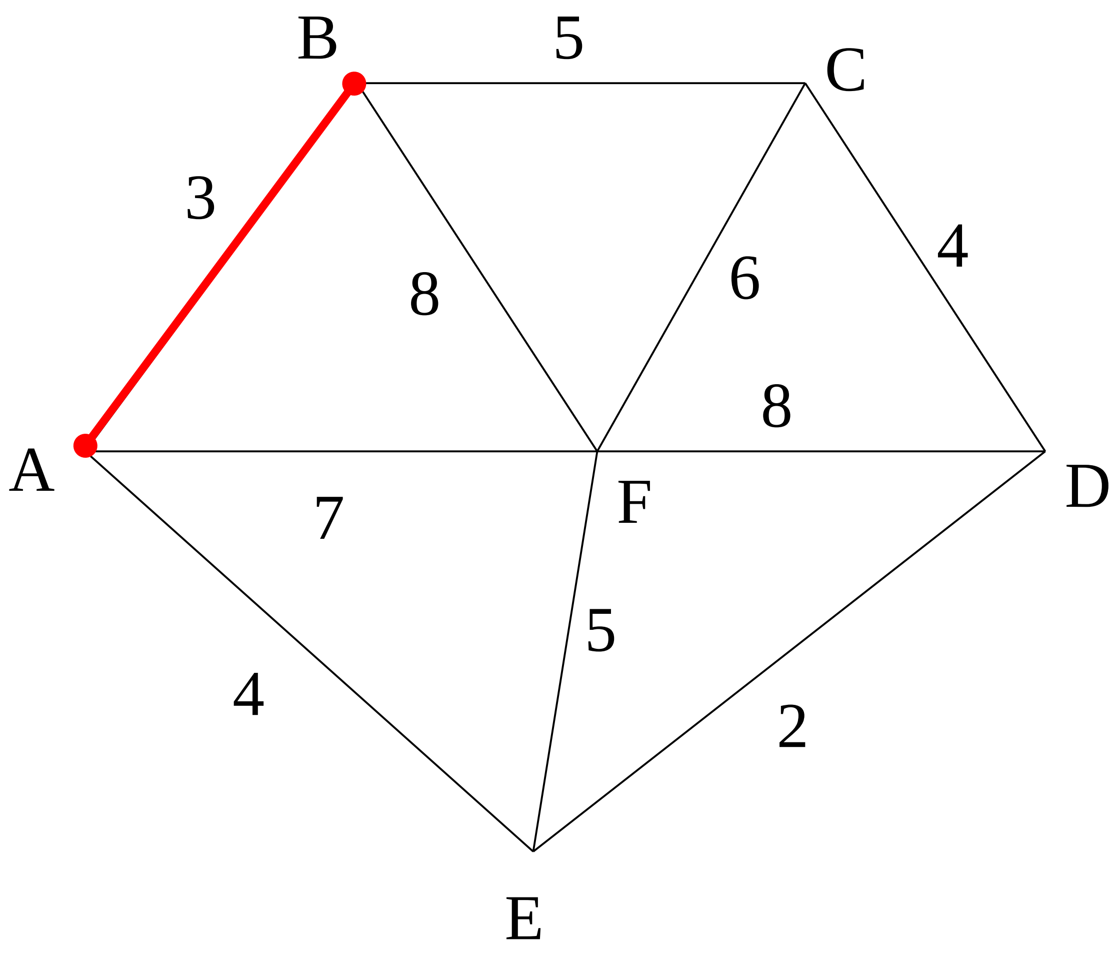
All vertices have been connected.

The solution is

ED	**2**
AB	**3**
CD	**4**
AE	**4**
EF	**5**

Total weight of tree: 18

Prim’s Algorithm

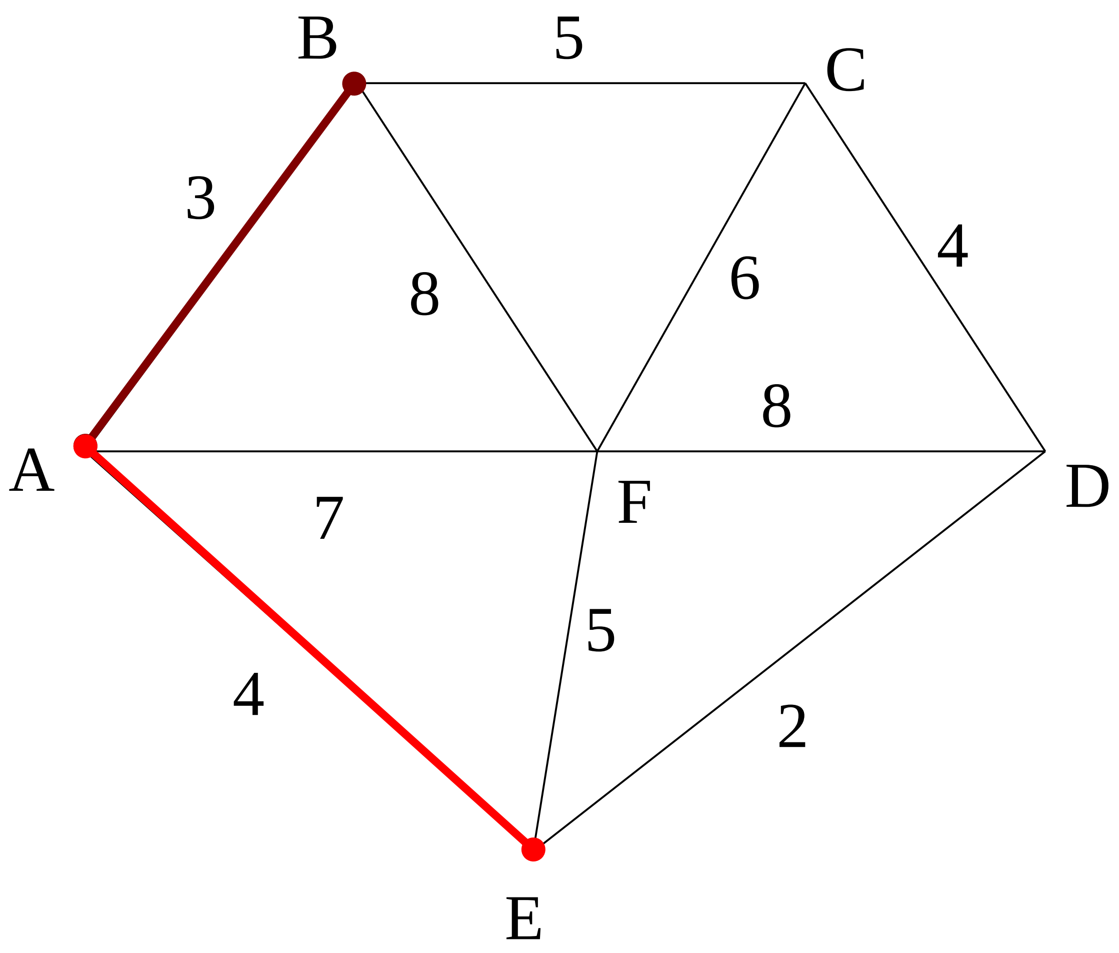


Select any vertex
A A를 출발정점으로 선택.
다른정점선택해도 상관없음.

Select the shortest edge connected to that vertex

AB 3

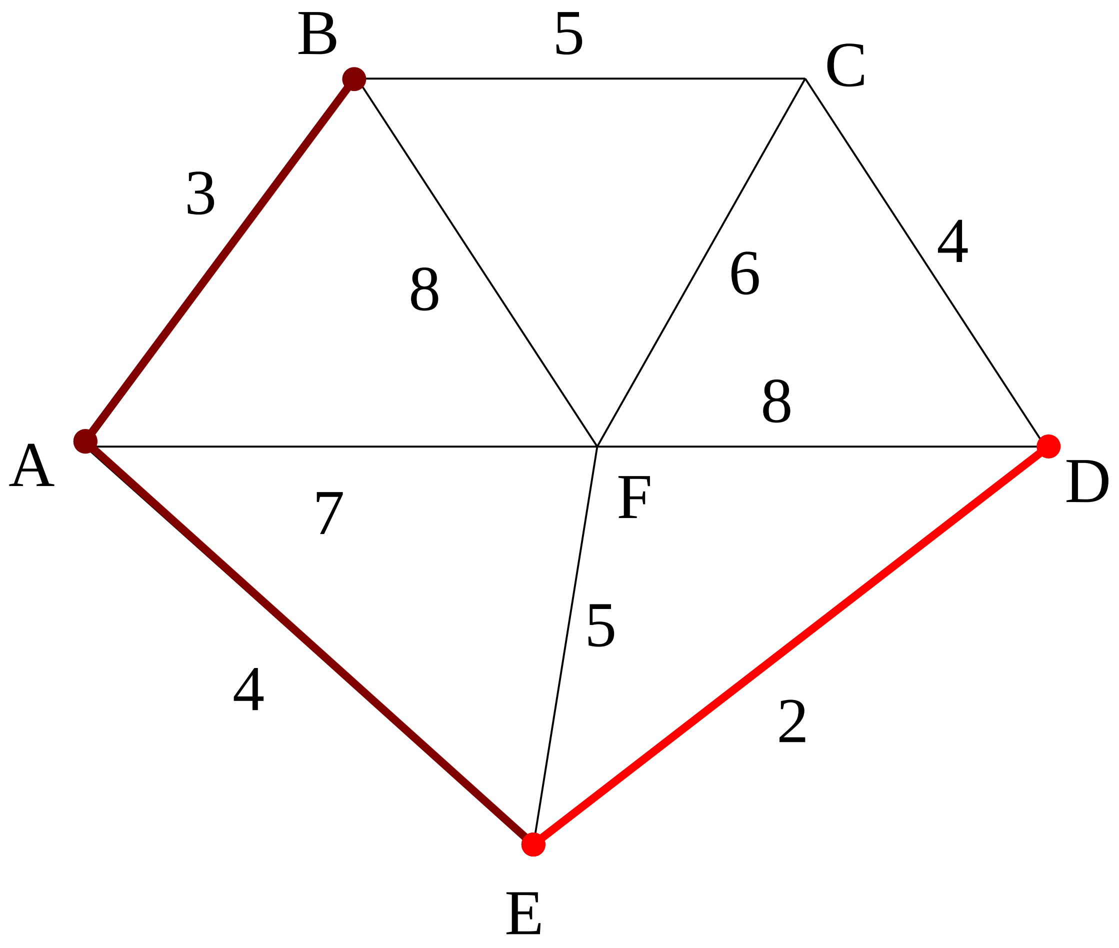
Prim's Algorithm



Select the shortest edge connected to any vertex already connected.

AE 4

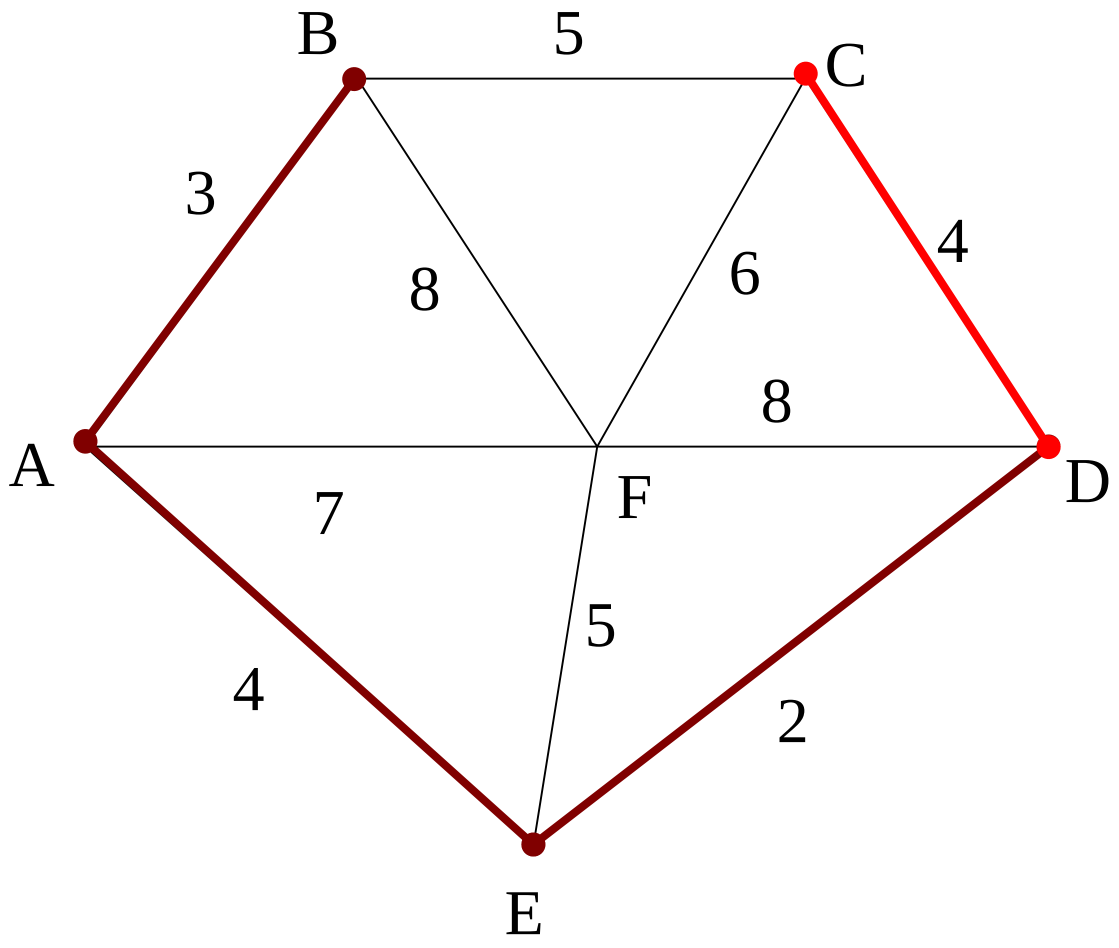
Prim's Algorithm



Select the shortest edge connected to any vertex already connected.

ED 2

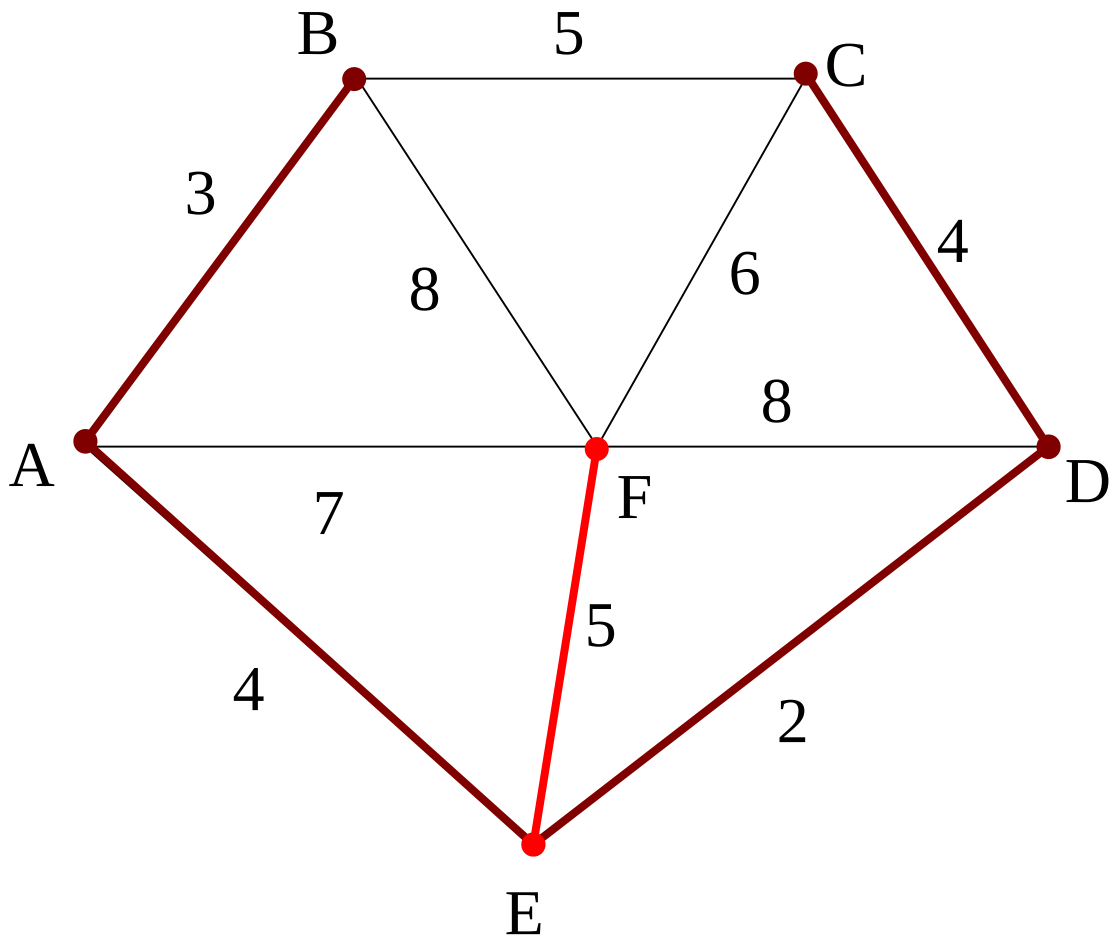
Prim's Algorithm



Select the shortest edge connected to any vertex already connected.

DC 4

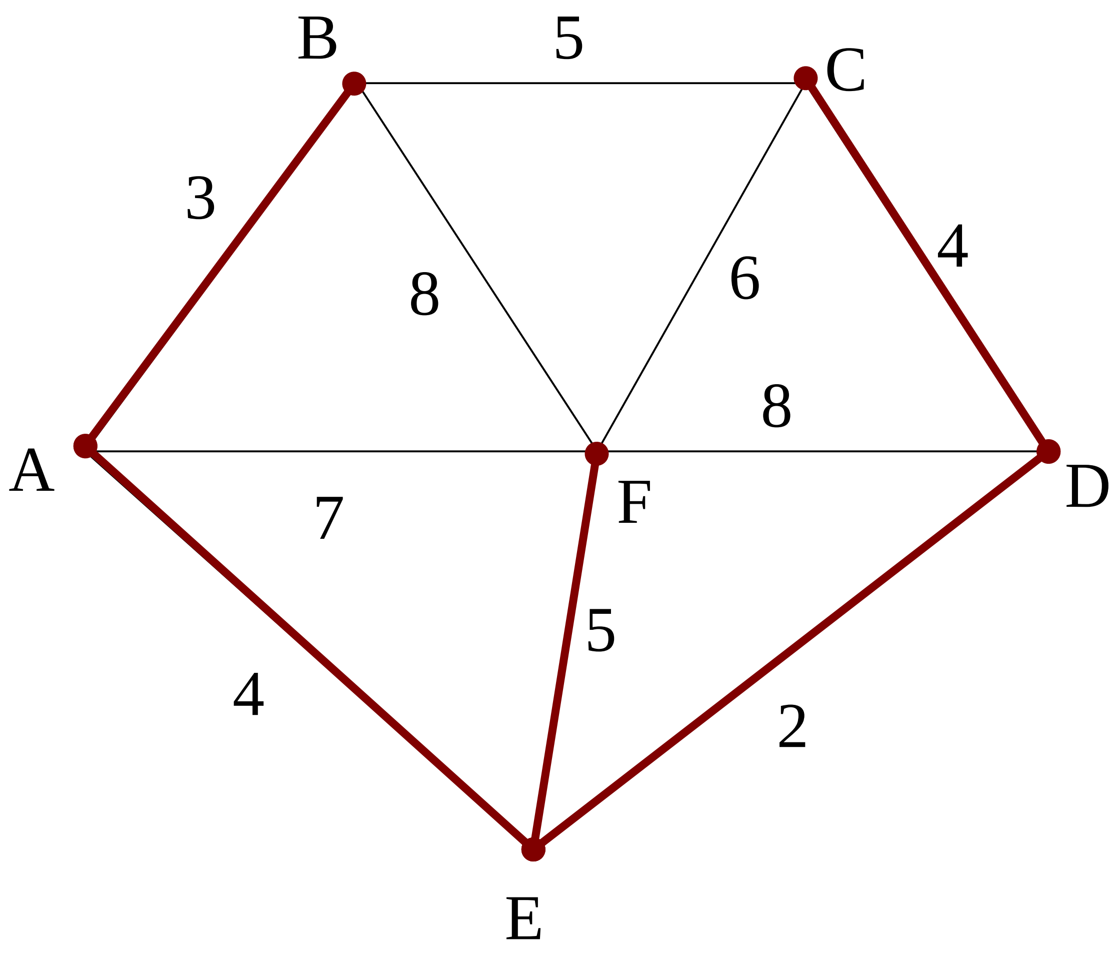
Prim's Algorithm



Select the shortest edge connected to any vertex already connected.

EF 5

Prim's Algorithm



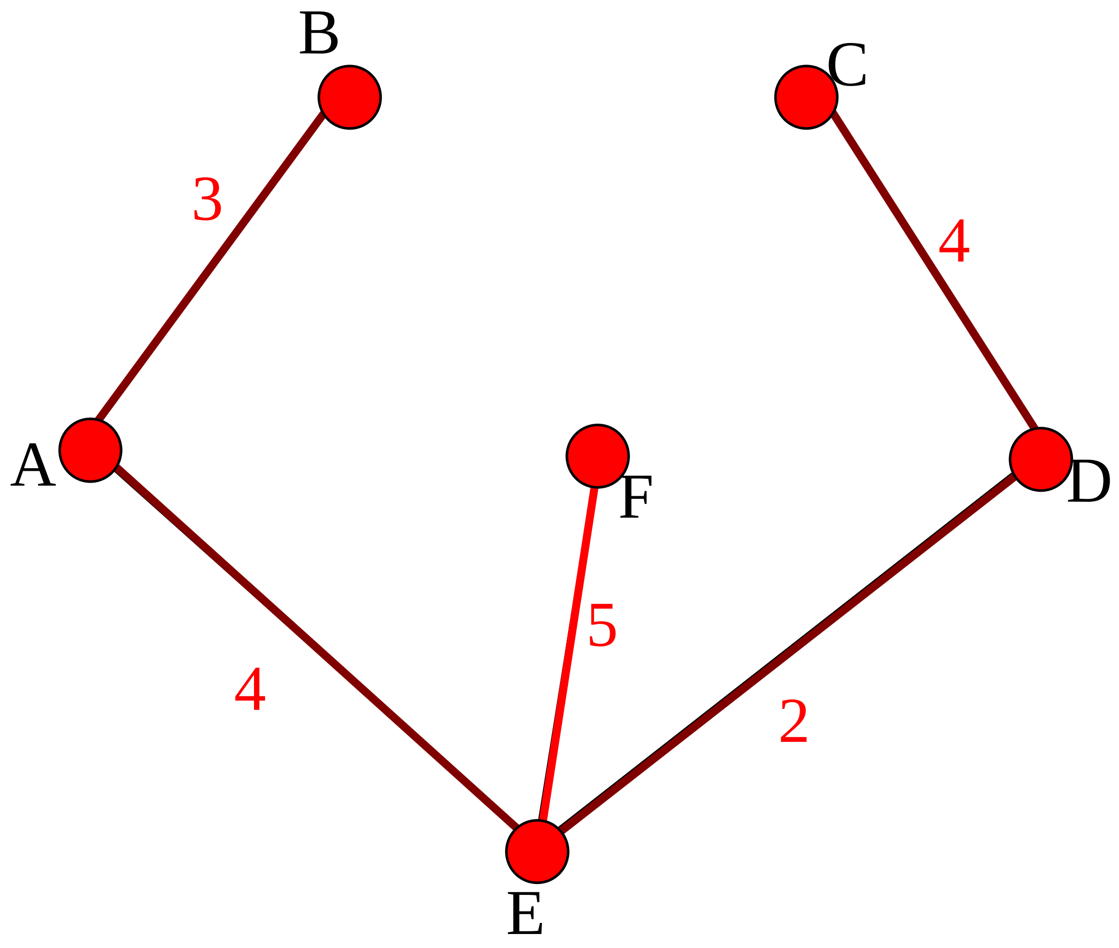
All vertices have been connected.

The solution is

AB 3
AE 4
ED 2
DC 4
EF 5

Total weight of tree: 18

Prim’s Algorithm



All vertices have been connected.

The solution is

ED 2
AB 3
CD 4
AE 4
EF 5

Total weight of tree: 18

Some points to note

- Both algorithms will always give solutions with the same length. (두 알고리즘 모두 항상 정답 제공)

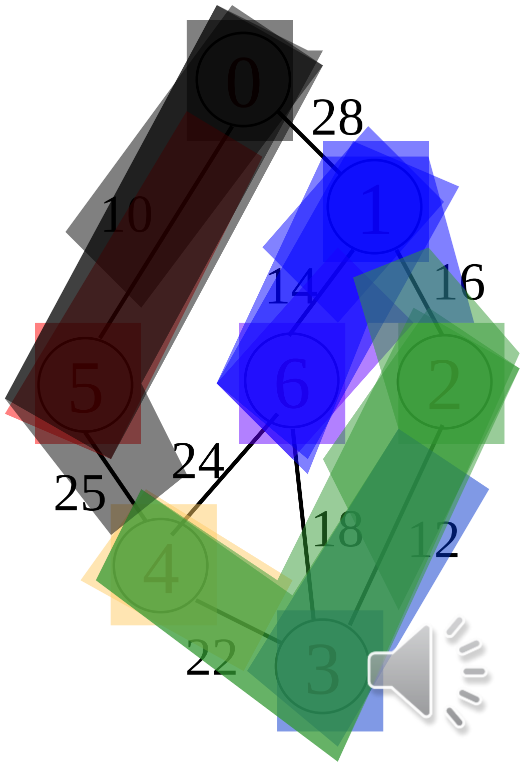
- They will usually select edges in a different order – you must show this in your workings. 자네들의 능력으로 찾으라.

- Occasionally they will use different edges – this may happen when you have to choose between edges with the same length. In this case there is more than one minimum connector for the network.(동일한 가중치의 간선이 존재하면, 서로 다른 최소 비용 신장 트리 존재)

Sollin's Algorithm
솔린의 알고리즘
소개만 하고 시험에는 출제하지 않겠음.

Minimum Cost Spanning Trees (7/7)

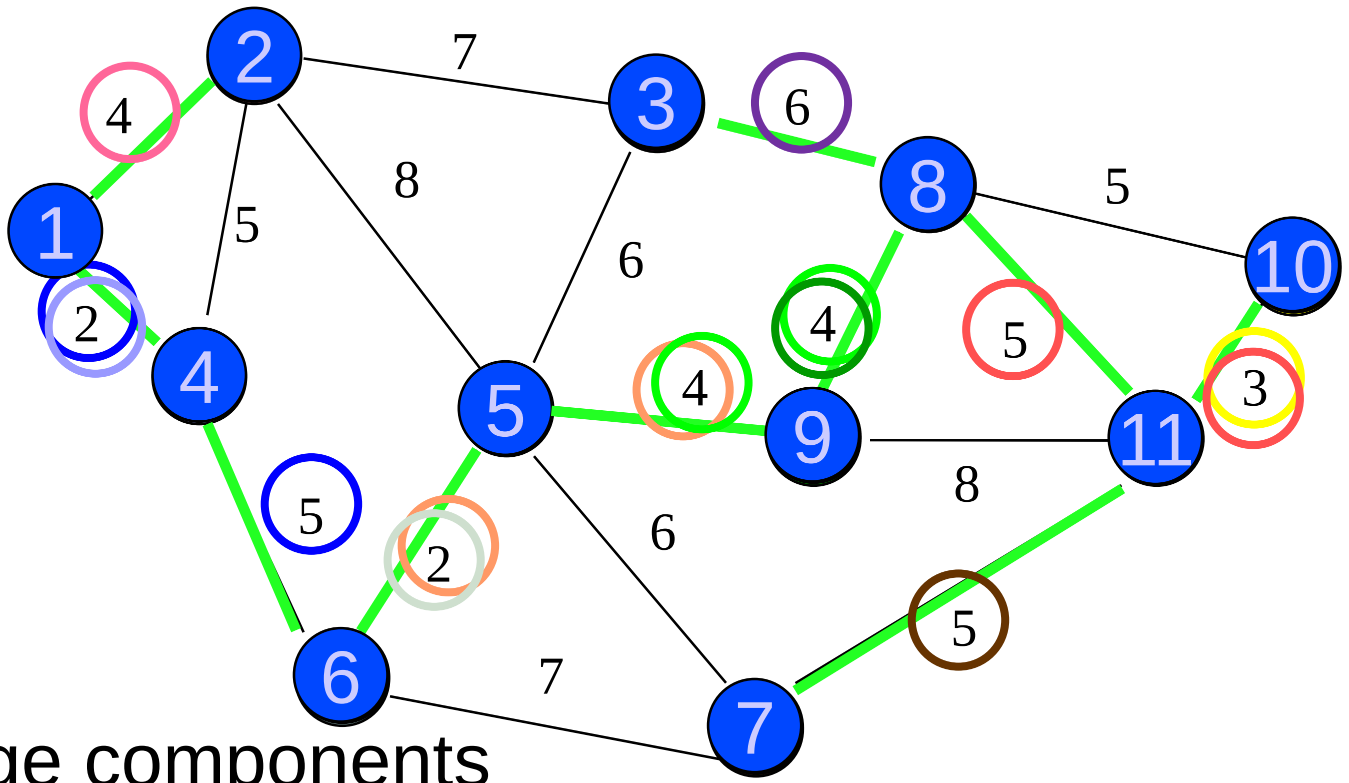
- Sollin's Algorithm
  - Selects several edges for inclusion in T at each stage.
  - At the start of a stage, the selected edges forms a spanning forest.
  - During a stage we select a minimum cost edge that has exactly one vertex in the tree edge for each tree in the forest.
  - Repeat until only one tree at the end of a stage or no edges remain for selection.
    - Stage 1: (0, 5), (1, 6), (2, 3), (3, 2), (4, 3), (5, 0), (6, 1)⇨{(0, 5)}, {(1, 6)}, {(2, 3), (4, 3)}
    - Stage 2: {(0, 5), (5, 4)}, {(1, 6), (1, 2)}, {(2, 3), (4, 3), (1, 2)}
    - Result: {(0, 5), (5, 4), (1, 6), (1, 2), (2, 3), (4, 3)}



Sollin's Algorithm

- Start with each node a component모든정점에서 동시출발
- Start with tree edges M = {}
- While |M| < V-1
    - For each component add least cost edge to distinct component(각각의 요소에 최소의 간선을 추가)
    - Merge components(합병)

Boruvka's Algorithm



Merge components

Two components may pick the same edge

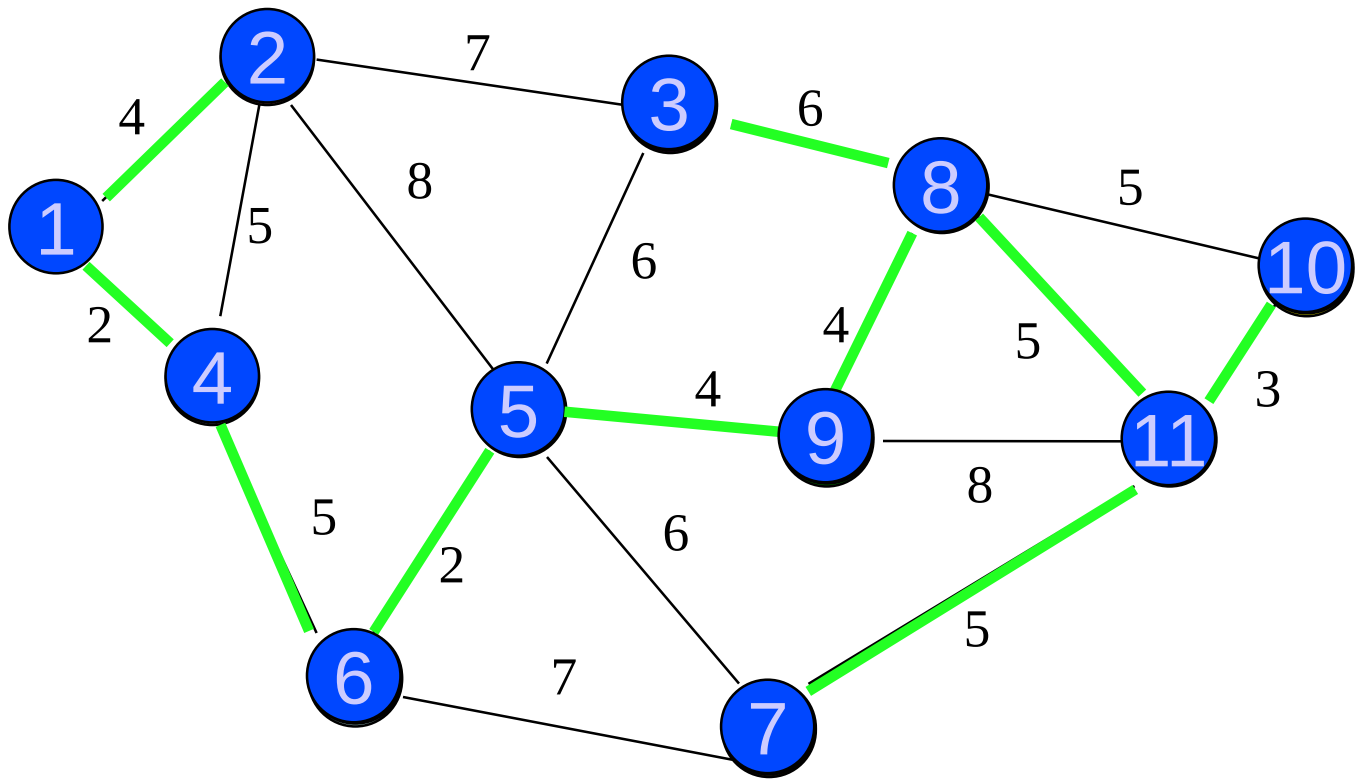
All nodes are tiny trees....each with a color

Grow each by adding nearest neighbor

Sollin's Algorithm

- Note that while the animation showed one edge added at a time, in fact each component can add an edge in parallel 동시에 이루어진다고 생각
- Makes for nice parallel (병렬)MST algorithm (Sollin)

Sollin's Algorithm



Grow components in parallel...

Done in two passes!

Sollin's Algorithm

- Grows components like Prim, but in parallel
- Adds edges connecting components like Kruskal

감사합니다.