TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Discrete Mathematics

Nguyễn Khánh Phương

Department of Computer Science
School of Information and Communication Technology
E-mail: phuongnk@soict.hust.edu.vn

PART 1
COMBINATORIAL THEORY
(Lý thuyết tổ hợp)

PART 2
GRAPH THEORY
(Lý thuyết đồ thị)

Content of Part 2

Chapter 1. Fundamental concepts

Chapter 2. Graph representation

Chapter 3. Graph Traversal

Chapter 4. Tree and Spanning tree

Chapter 5. Shortest path problem

Chapter 6. Maximum flow problem

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Chapter 4

Tree and Spanning Tree

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Contents

1. Tree and its properties

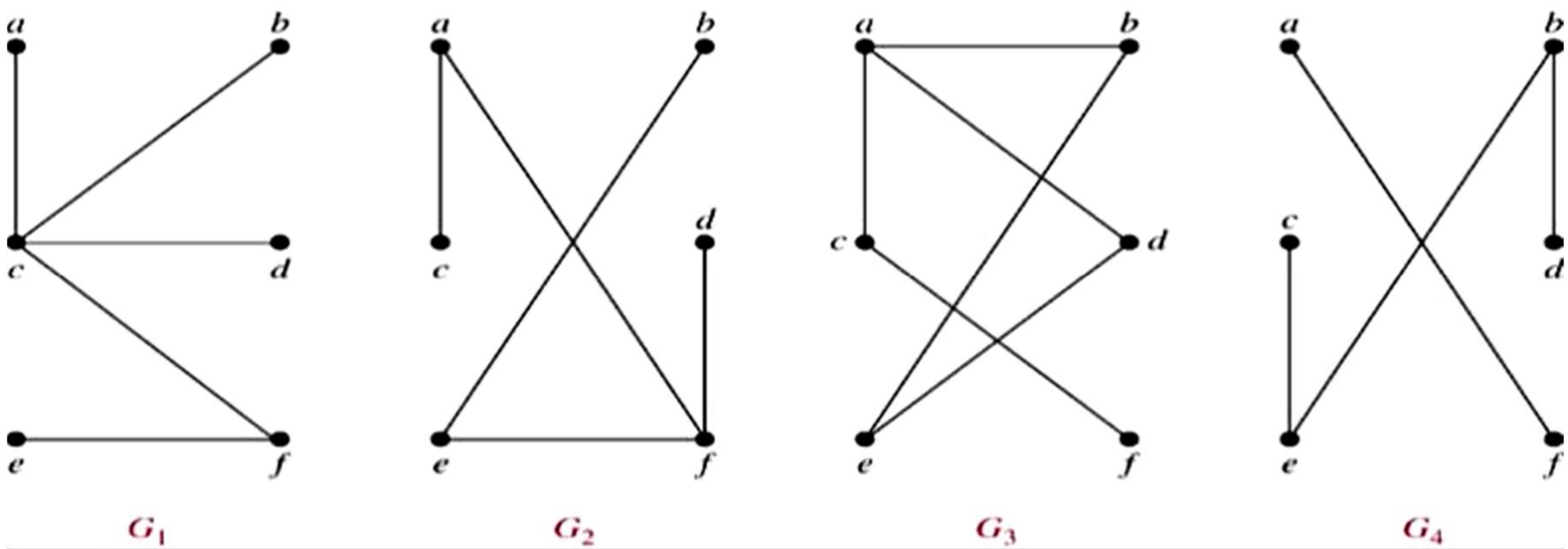
2. Spanning tree

3. The minimal spanning tree

1. Tree and its properties

A **tree** is an undirected connected graph with no cycles.

Example 1. Which of the graphs are trees?



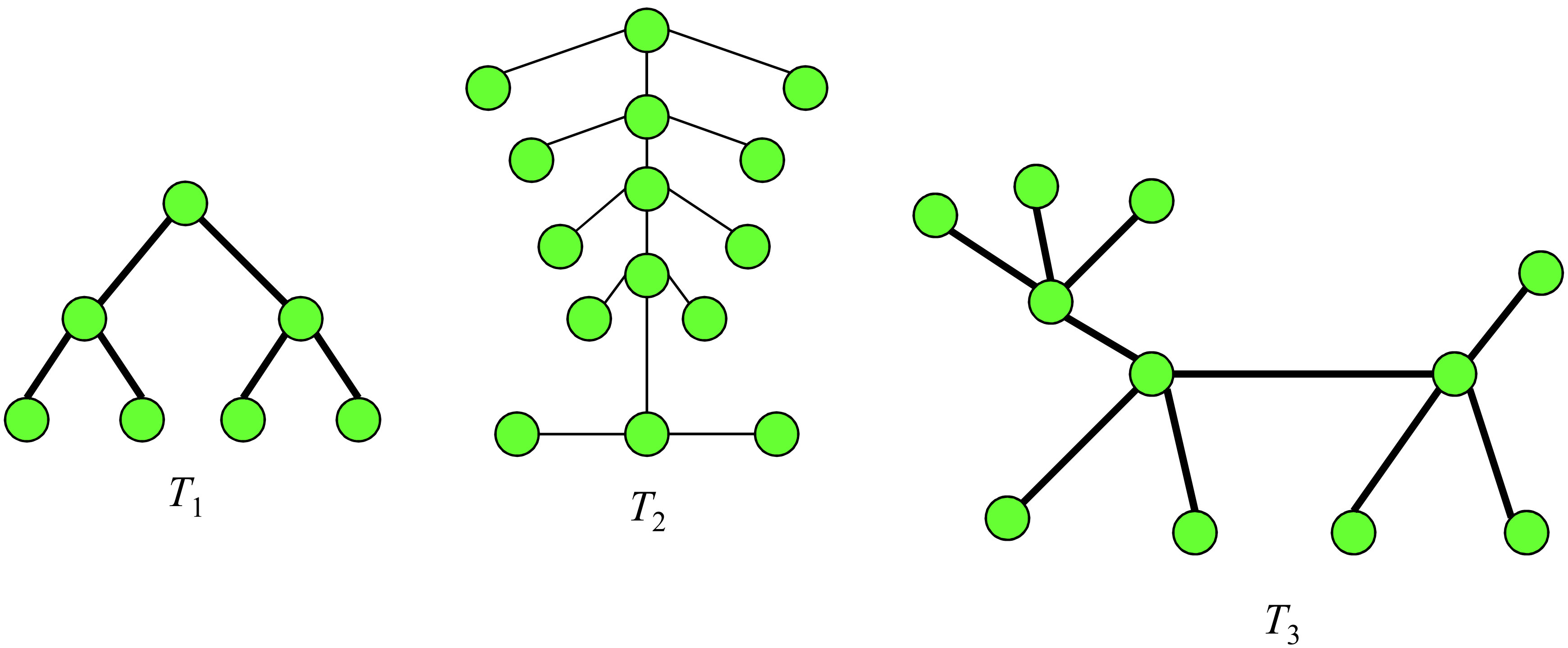
Solution: G_1, G_2

Note. G3: contains cycle {a,b,e,d,a}
G4: not connected

From the definition one could see that **a tree has no loops or multiple edges**, because any loop is a cycle by itself, and if edges e_i and e_j join the same pair of vertices then the sequence e_i, e_j is also a cycle

1. Tree and its properties

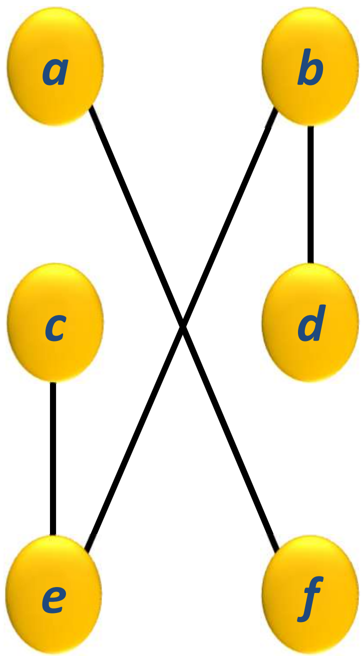
Forest: Graph containing no cycles that are not connected, but each connected component is a tree.



Forest *F* consists of 3 trees: T_1, T_2, T_3

Example

Forest: Graph containing no cycles that are not connected, but each connected component is a tree.



Forest

Not connected → not tree

Consists of 2 trees: {a,f} and {c,e,b,d}

1. Tree and its properties

Theorem. Given an undirected graph $G = (V,E)$, the following conditions are equivalent:

(1) G is a connected graph with no cycles. (Thus G is a tree by the above definition).

(2) For every two vertices $u, v \in V$, there exists exactly one simple path from u to v.

(3) G is connected, and removing any edge from G disconnects it (each edge of G is a bridge).

(4) G has no cycles, and adding any edge to G gives rise to a cycle. (Thus G is a maximal acyclic graph).

(5) G is connected and $|E| = |V| - 1$.

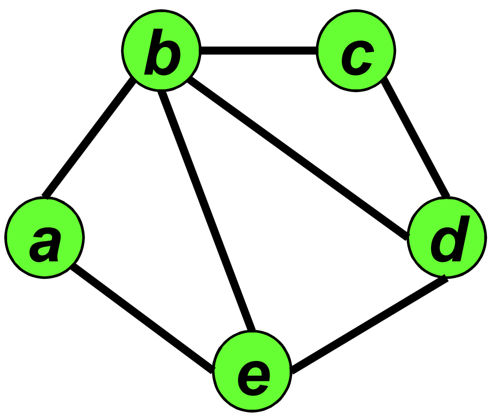
Contents

2.The spanning tree

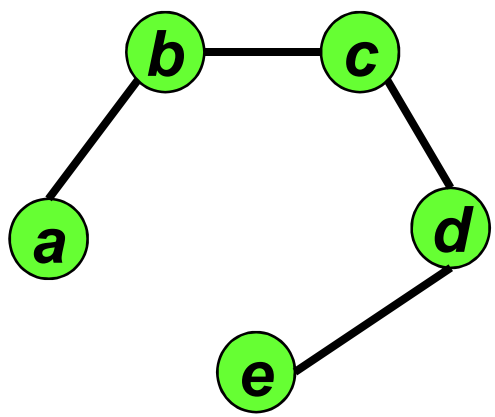
Let $G=(V, E)$ be an undirected connected graph with vertex set V.

Tree $T=(V,F)$ *where* $F \subseteq E$ *is called* ***spanning tree*** *of* G

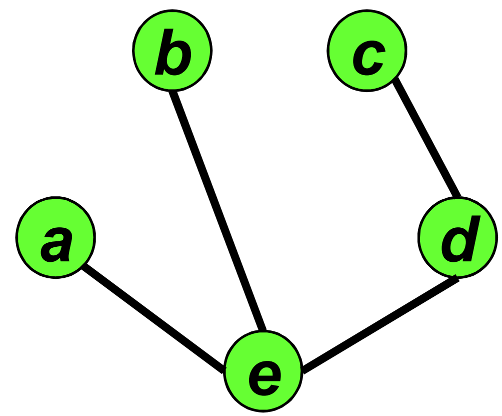
Undirected Connected graph without cycle



G



Spanning tree T_1



Spanning tree T_2

Graph G and its 2 spanning trees T_1 and T_2

2. The spanning tree

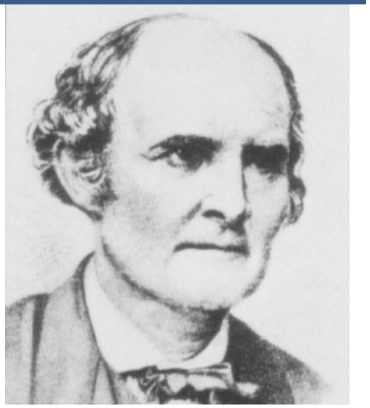
Theorem. Every undirected connected graph contains a spanning tree.

Proof. Let G be an undirected connected graph.

- If G contains no cycle then G is its own spanning tree.
- If G contains a cycle: Removing any edge from the cycle gives a graph which is still connected. If the new graph contains a cycle then again remove one edge of the cycle. Continue this process until the resulting graph T contains no cycles. We have not removed any vertices so T has the same vertex set as G, and at each step of the above process we obtain a connected graph. Therefore T is connected and it is a spanning tree for G.

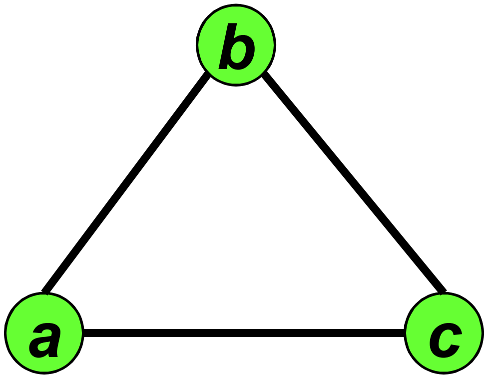
2. The spanning tree

Theorem (Cayley). A complete graph K_n has n^{n-2} spanning trees.

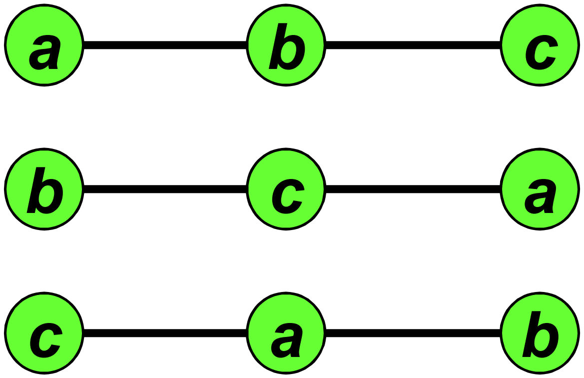


Arthur Cayley
(1821 – 1895)

(A **complete graph** is a simple undirected graph in which every pair of distinct vertices is connected by a unique edge)



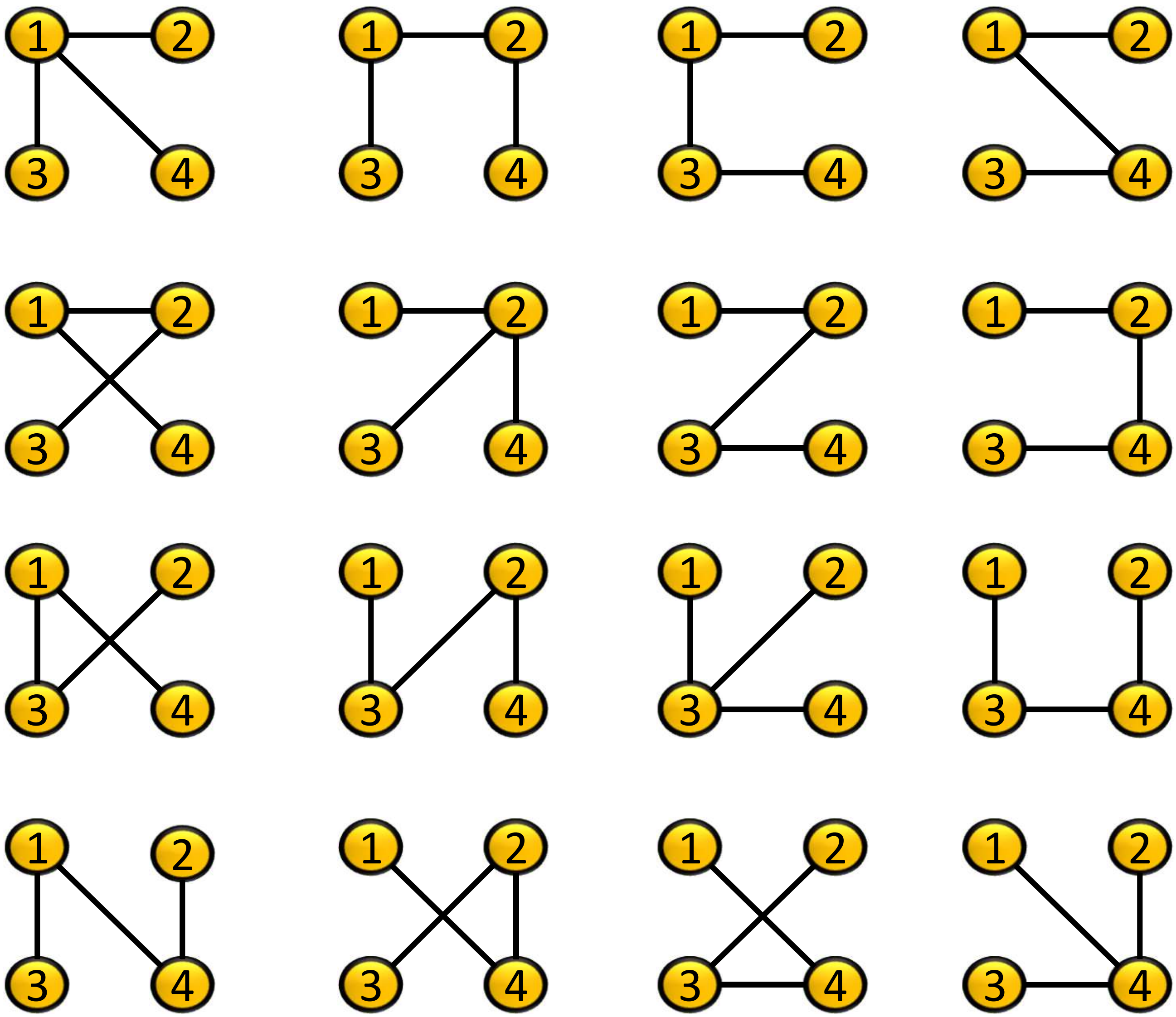
K_3



3 spanning trees of K_3

16 spanning trees of K_4

Theorem (Cayley). A complete graph K_n has n^{n-2} spanning trees.



Contents

Weighted Graphs and Minimum Spanning Trees

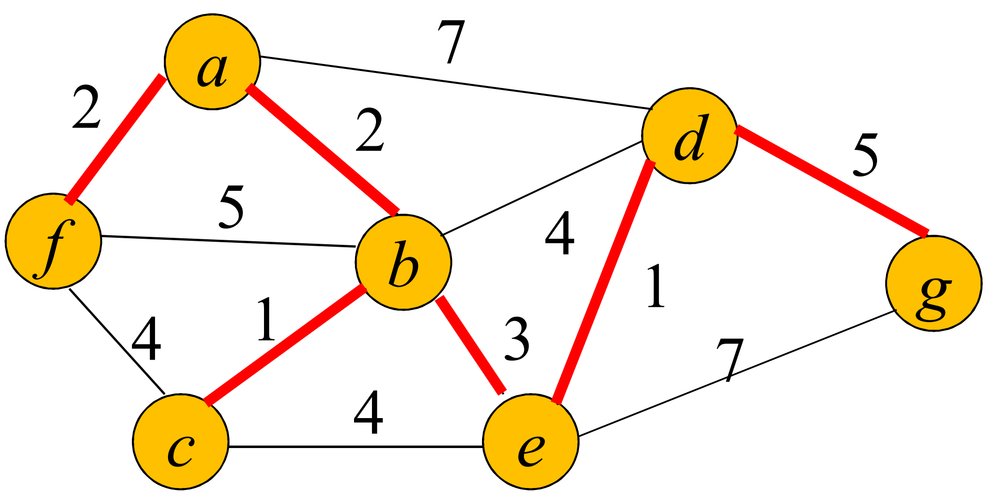
Let $G=(V, E)$ be an undirected connected graph with vertex set V:

- For each edge (u,v) in E, we have a weight $w(u,v)$ specifying the cost (length of edge) to connect u and v.

For any subgraph H of G, we define the *weight of H*, denoted by $w(H)$, to be the sum of its edge weights:

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

A ***minimum spanning tree*** for G is a spanning tree T which has the smallest weight



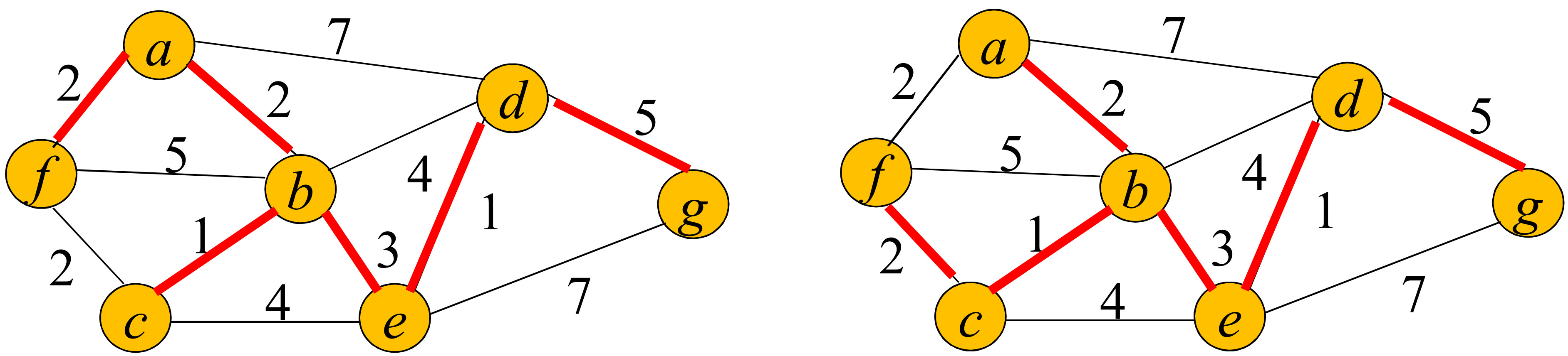
The weight of the spanning tree = 14

Weighted Graphs and Minimum Spanning Trees

Every undirected connected weighted graph *G* has a minimum spanning tree. Since *G* has only a finite number of spanning trees, one of them must have minimum weight.

Note: a given undirected connected weighted may have more than one minimum spanning tree.

Example: An undirected connected weighted graph with two minimal spanning trees, both of weight 14

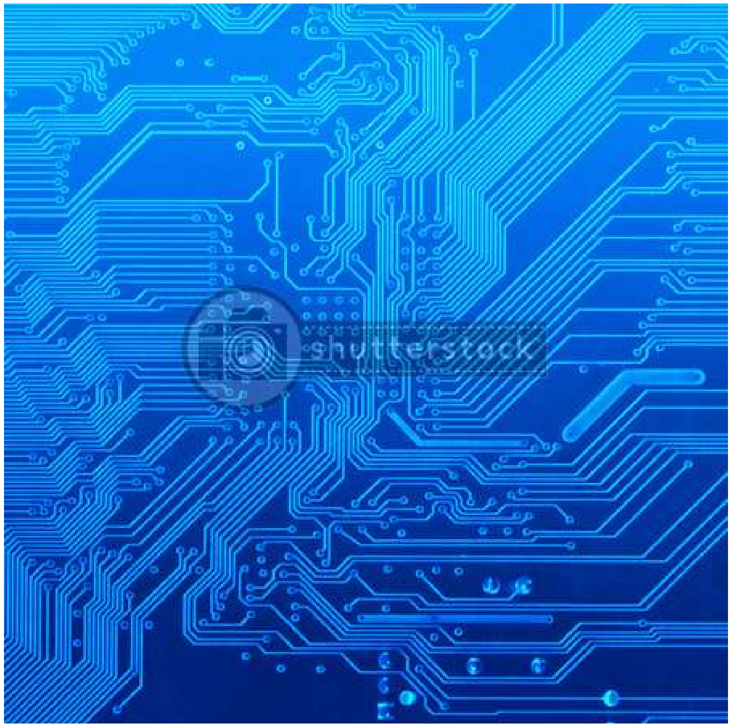


As the number of spanning trees of G is very large (see Cayley's theorem), we could not solve this problem by brute force.

Applications of Minimum Spanning Trees: an example

Network design: telephone, electrical, hydraulic, TV cable, computer, road.

- Phone network design. You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. It should be a spanning tree, since if a network isn't a tree you can always remove some edges and save money.
- In the design of electronic circuits, it is often necessary to connect pins by wiring them together:
  - To interconnect n pins, we can use n-1 wires, each connecting 2 pins.
  - We want to minimize the total length of the wires.
  - Minimum Spanning Trees can be used to model this problem.



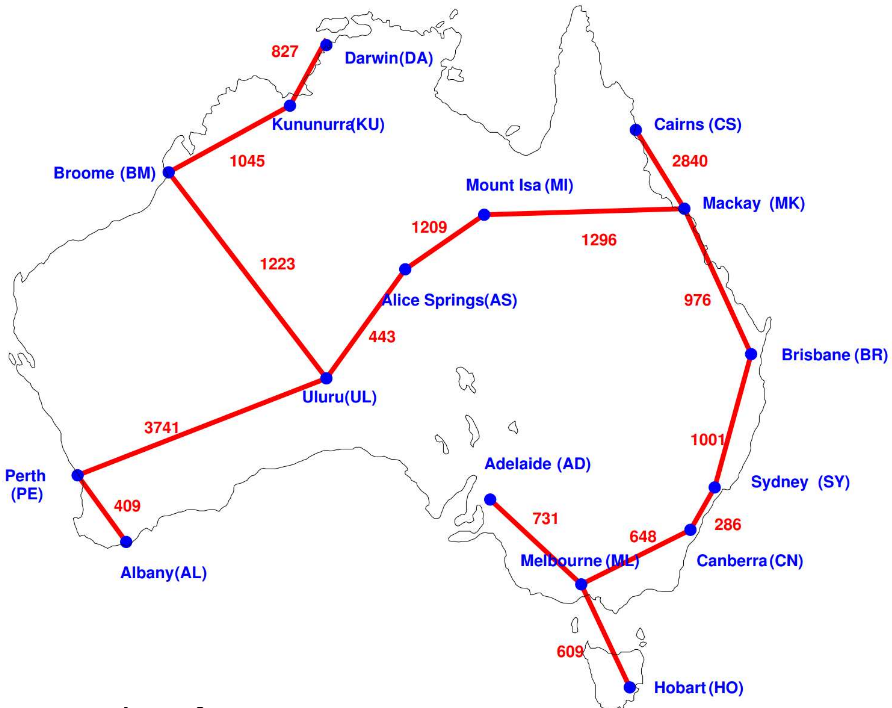
Applications of Minimum Spanning Trees: an example

Suppose we want to build a railway system that connects n cities so that passengers can travel between any two cities and the total cost of construction must be minimal.

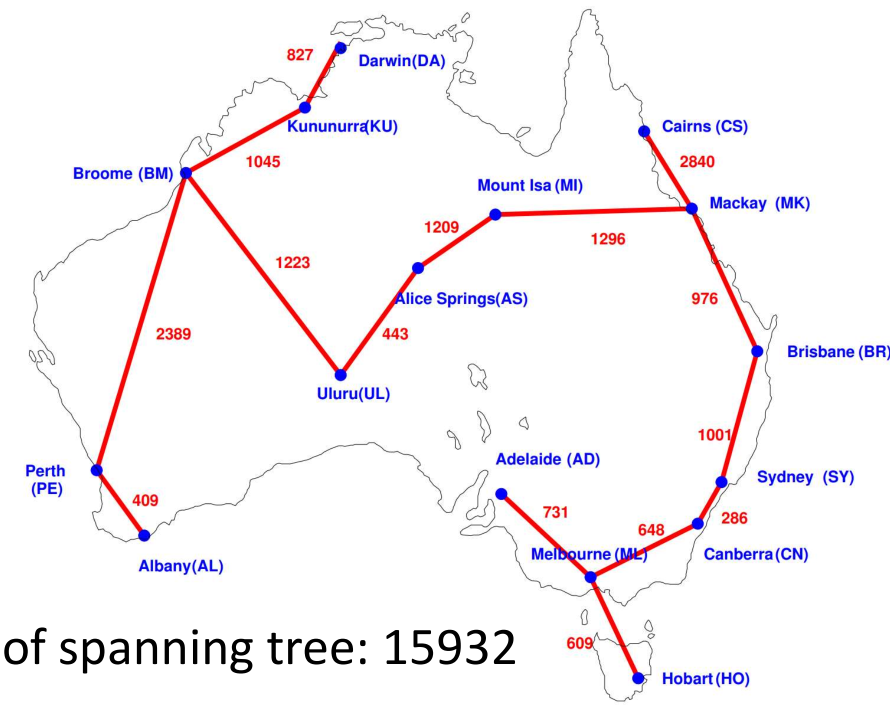
It is clear that it corresponds to a graph where vertices are the cities and the edges are the railroads connecting the cities, the length on each edge is the cost of building a railway connecting the two cities.

Therefore, it leads to the problem of finding the smallest spanning tree on a complete graph K_n

Build a railway system



Length of spanning tree: 17284



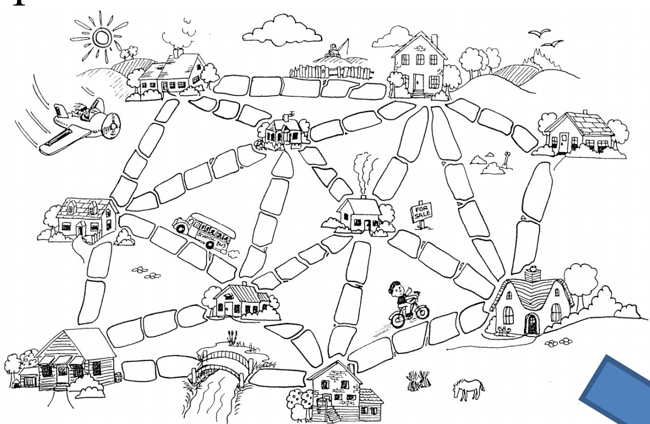
Length of spanning tree: 15932

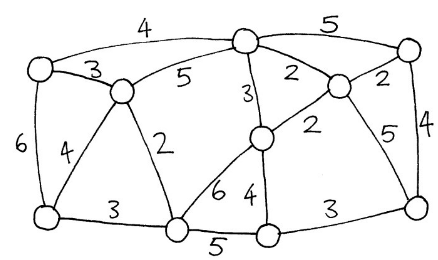
Applications of Minimum Spanning Trees: an example

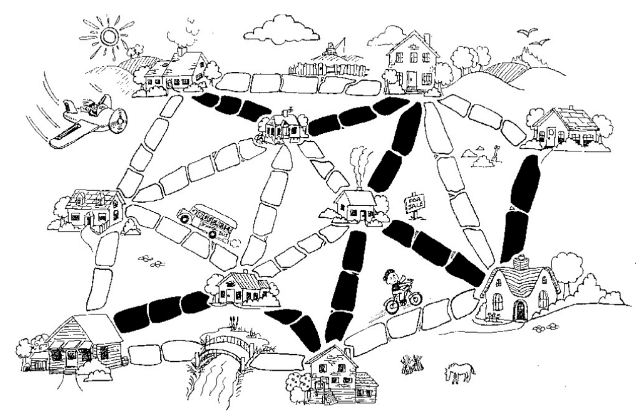
Once upon a time there was a city that had no roads. The mayor of the city decided that some of the streets must be paved, but didn't want to spend more money than necessary because the city also wanted to build a swimming pool. The mayor therefore specified two conditions:

- Enough streets must be paved so that it is possible for everyone to travel from their house to anyone else house only along paved roads,
- The paving should cost as little as possible. Here is the layout of the city. The number of paving stones between each house represents the cost of paving that route.

Find the best route that connects all the houses, but uses as few counters (paving stones) as possible







Applications of Minimum Spanning Trees: examples

Other practical applications based on minimal spanning trees include:

- Taxonomy
- Cluster analysis: clustering points in the plane, single-linkage clustering, graph-theoretic clustering, and clustering gene expression data.
- Constructing trees for broadcasting in computer networks. On Ethernet networks this is accomplished by means of the Spanning tree protocol.
- Image registration and segmentation.
- Curvilinear feature extraction in computer vision.
- Handwriting recognition of mathematical expressions.
- Circuit design: implementing efficient multiple constant multiplications, as used in finite impulse response filters.
- Regionalization of socio-geographic areas, the grouping of areas into homogeneous, contiguous regions.
- Comparing ecotoxicology data.
- Topological observability in power systems.
- Measuring homogeneity of two-dimensional materials.
- Minimax process control.

General scheme of the algorithm to find MST

Initialize: The minimum spanning tree T = ∅

Each step of the algorithm: one edge *e* which is the "safe" edge is chosen, subject only to the restriction that if adding edge *e* into T then T is still a tree (no cycle is created).

Generic-MST(G, c)

```
  T = ∅
  //T  is the subset edges of some minimum spanning tree
  while T is not the spanning tree do
       Finding edge (u, v) is "safe" edge for T
       T = T ∪{(u, v)}
  return T
```

Edge with smallest weight and insert it into T does not create cycle

Set *T* is always a subset of edges of some minimum spanning tree. This property is called the **invariant Property**.
An edge (*u*,*v*) is a **safe edge** for *T* if adding the edge to *T* does not destroy the invariant.

General scheme of the algorithm to find MST

Initialize: The minimum spanning tree T = ∅

Each step of the algorithm: one edge *e* which is the "safe" edge is chosen, subject only to the restriction that if adding edge *e* into T then T is still a tree (no cycle is created).

Generic-MST(G, c)

```
T = ∅
//T  is the subset edges of some minimum spanning tree
while T is not the spanning tree do
     Finding edge (u, v) is "safe" edge for T
     T = T ∪{(u, v)}
return T
```

Edge with smallest weight and insert it into T does not create cycle

How to find the "safe" edge ???:

The criteria to choose an edge at each step decides the process of two following minimum spanning tree algorithms (both use *greedy* strategies):

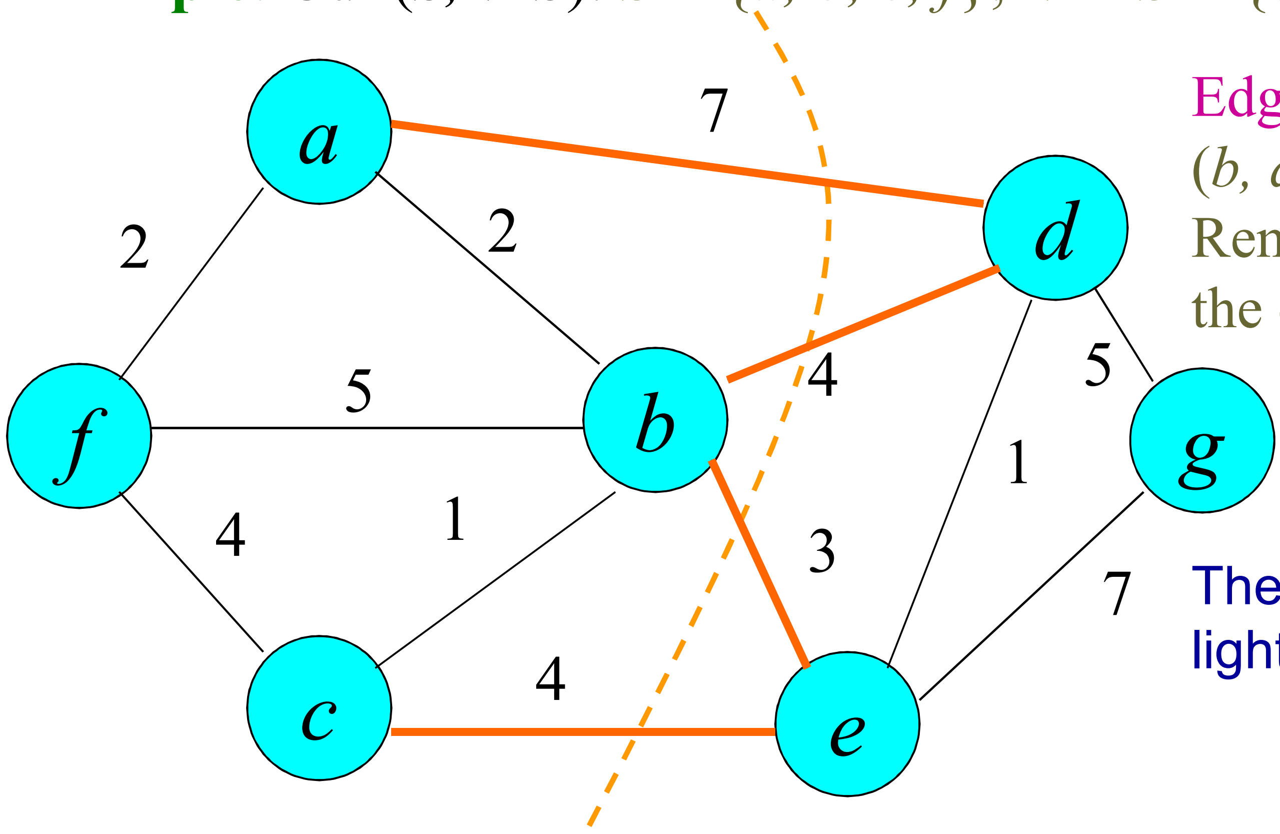
1. Kruskal
2. PRIM

How to find the "safe" edge ???:

We need some definitions and a theorem.

- A cut (S,V-S) of an undirected graph G=(V,E) is a partition of V into 2 sets S and V-S.
- An edge crosses the cut (S,V-S) if one of its endpoints is in S and the other is in V-S.
- An edge is a light edge crossing a cut if its weight is the minimum of any edge crossing the cut.

Example. Cut (S, V-S): $S = \{a, b, c, f\}$, $V - S = \{e, d, g\}$



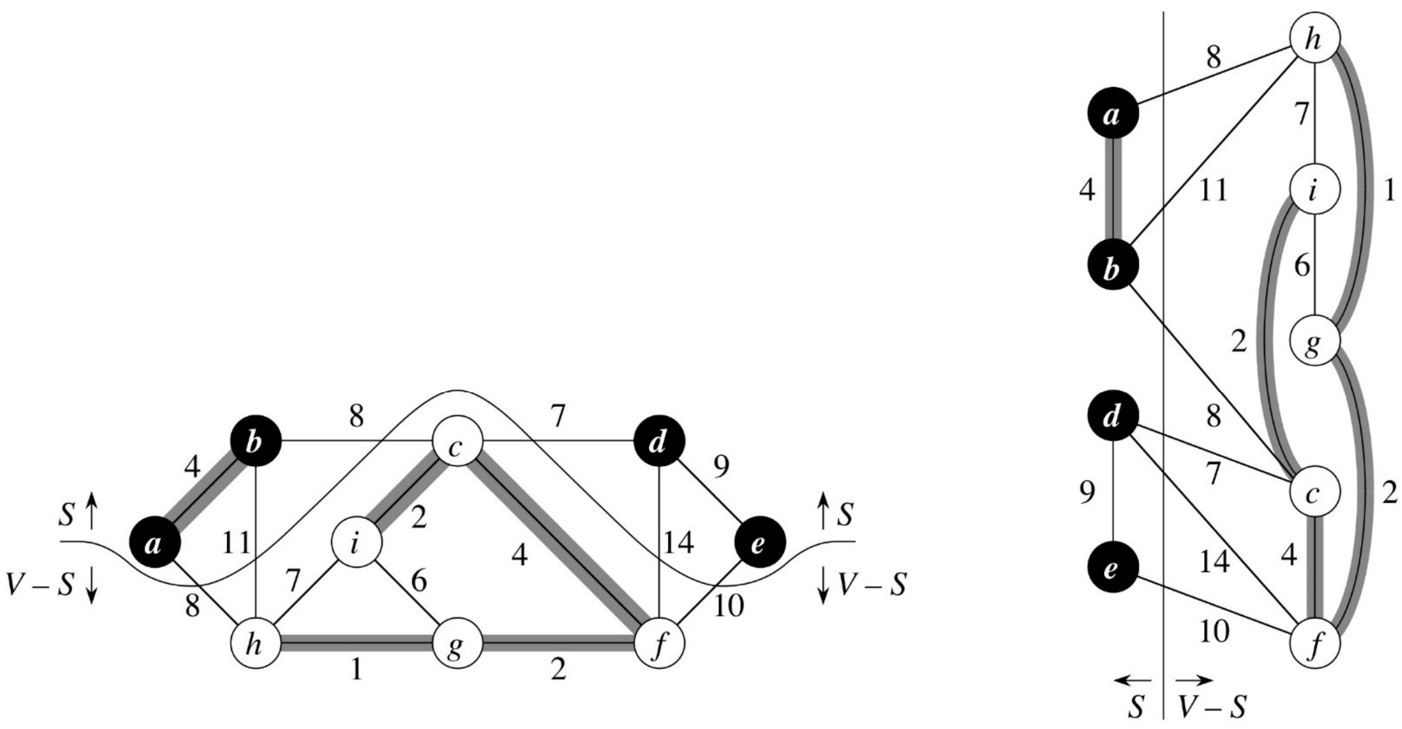
Edges crossing the cut :
(b, d), (a, d), (b, e), (c, e)
Remaining edges do not cross the cut

The edge (b,e) is the unique light edge crossing the cut.

How to find the “safe” edge ???:

We need some definitions and a theorem.

- Safe edge: an edge that may be added to T without violating the invariant that T is a subset of some minimum spanning tree (not create cycle in T)
- Respecting: A cut respects a set T of edges if no edge in T crosses the cut



(a)

(b)

T ={(a,b), (i, c), (c, f), (h, g)}
➔ Cut (S, V-S) repects T

T = {(a,b), (h,g), (g,f), (i, c), (c, f)}
➔ Cut (S, V-S) repects T

Light edge crossing the cut is the "safe" edge

Theorem.

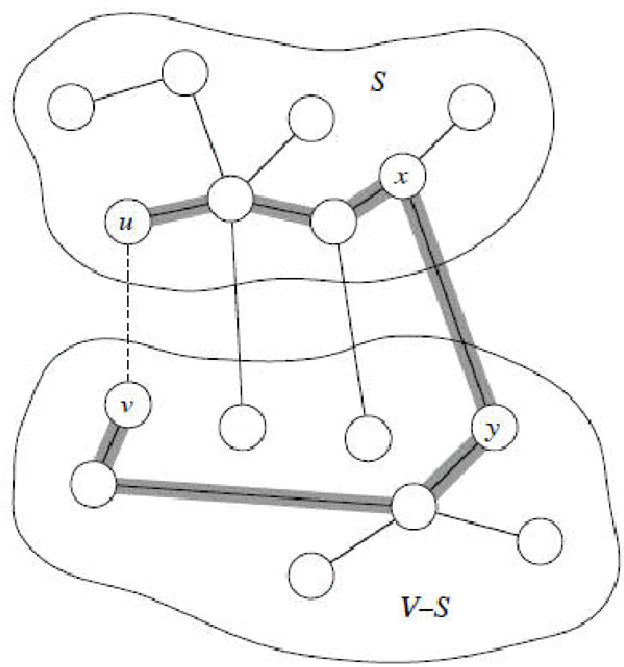
- Let G = (V, E) be a connected, undirected graph with a real-valued weight function ***w*** defined on E.
- Let T be a subset of E that is included in some minimum spanning tree for G,
- Let (S, V−S) be $\underbrace{\text{any cut of G that respects T,}}_{\text{for any edge } (x,\, y) \text{ in T, } \{x,\, y\} \subseteq \text{S or } \{x,\, y\} \subseteq \text{(V-S).}}$
- Let (u, v) be a light edge crossing (S, V−S).

Then $\underbrace{\text{edge } (u, v) \text{ is safe for T.}}_{T \cup \{(u,\, v)\} \text{ is still a subset of edges of some minimum spanning tree}}$

Proof: Let T_{opt} be an MST that includes T.

- If T_{opt} contains an edge (u, v), we are done.
- So now assume that T_{opt} does not contain (u, v). We will construct a different MST T'_{opt} that include $T = T \cup \{(u, v)\}$.
  - Recall: a tree has unique simple path between each pair of vertices. Since T_{opt} is an MST, it contains a unique path p between u and v. Path p must cross the cut $(S, V - S)$ at least once. Let (x, y) be an edge of p that crosses the cut. From how we chose (u, v), must have $w(u, v) < w(x, y)$.
  - Since the cut respects **T**, edge (x, y) is not in **T**. To form T'_{opt} from T_{opt}:
    - Remove the edge (x, y). Breaks T into two components.
    - Add edge (u, v). Reconnects.

.



[Note carefully: Except for the dashed edge (u, v), all edges shown are in T_{opt}. **T** is some subset of the edges of T_{opt}, but **T** cannot contain any edges that cross the cut $(S, V - S)$, since this cut respects **T**. Shaded edges are the path p.]

Proof: Let T_{opt} be an MST that includes T.

- If T_{opt} contains an edge (u, v), we are done.
- So now assume that T_{opt} does not contain (u, v). We will construct a different MST T'_{opt} that include $T = T \cup \{(u, v)\}$.
  - Recall: a tree has unique simple path between each pair of vertices. Since T_{opt} is an MST, it contains a unique path p between u and v. Path p must cross the cut $(S, V - S)$ at least once. Let (x, y) be an edge of p that crosses the cut. From how we chose (u, v), must have $w(u, v) < w(x, y)$.
  - Since the cut respects **T**, edge (x, y) is not in **T**. To form T'_{opt} from T_{opt}:
    - Remove the edge (x, y). Breaks T into two components.
    - Add edge (u, v). Reconnects.

  So $T'_{opt} = T_{opt} - \{(x, y)\} \cup \{(u, v)\}$ ➔ T'_{opt} is a spanning tree.

  $w(T'_{opt}) = w(T_{opt}) - w(x, y) + w(u, v)$

  $\leq w(T_{opt})$, since $w(u, v) \leq w(x, y)$.

Since T'_{opt} is a spanning tree, $w(T'_{opt}) \leq w(T_{opt})$, and T_{opt} is an MST, then T'_{opt} must be an MST.

- We need to show that edge (u, v) is safe for the set T:
  - $T \subseteq T_{opt}$ and $(x, y) \notin T \Rightarrow T \subseteq T'_{opt}$
  - $T \cup \{(u, v)\} \subseteq T'_{opt}$
- Since T'_{opt} is an MST, edge (u, v) is safe for the set T. This completes the proof.

How to find the "safe" edge?

T : set of edges of some spanning tree

Initialize: $T = \varnothing$

Kruskal algorithm

- T is **forest.**
- The "safe" edge added to T at each iteration is the edge with smallest weight **among edges connecting its connected components**.

Prim algorithm

- T is **tree**.
- The "safe" edge added to T at each iteration is the edge with smallest weight **among edges connecting the tree T to other vertex not in the tree**.

Kruskal algorithm

Joseph Kruskal
(1928 - ~)

```
Generic-MST(G, c)
   T = ∅
   //T  is the subset edges of a minimum spanning tree
   while T is not the spanning tree do
        Finding edge (u, v) is "safe" edge for T
        T = T ∪{(u, v)}
   return T
```

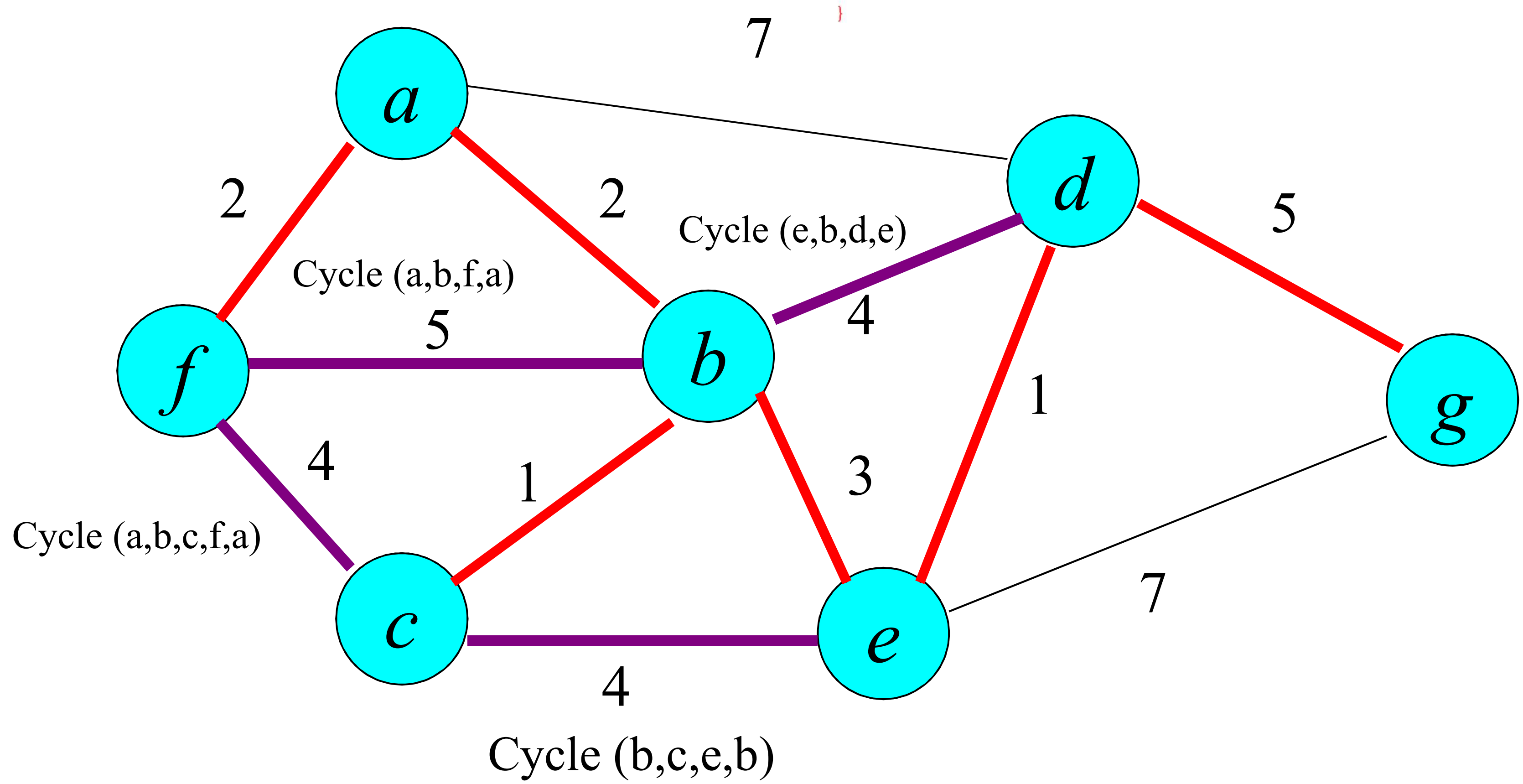
Kruskal algorithm:

- *T* is **forest (empty).**
- The "safe" edge included in *T* at each iteration is the edge with smallest weight **among edges connecting its connected components**.

```
void Kruskal ( )
{
    Sort m edges of the graph e1,e2, . . . , em in ascending order of weight;
    T = ∅;       // T: set of edges of the minimum spanning tree
    for (i = 1; i <=m; i++)
         if (T ∪{ei} does not contain cycle)
                       T =  T ∪ {ei};
}
```

Kruskal algorithm: an example

```
void Kruskal ( )
{
    Sort m edges of the graph e1,e2, . . . , em in ascending order of weight;
    T = ∅;        // T: set of edges of the minimum spanning tree
    for (i = 1; i <=m; i++)
        if (T ∪{ei} does not contain cycle)
                T =  T ∪ {ei};
}
```



Weight of the minimum spanning tree: 2+2+1+3+1+5=14

Computation time

```
void Kruskal ( )
{
    Sort m edges of the graph e1,e2, . . . , em in ascending order of weight;
    T = ∅;       // T: set of edges of the minimum spanning tree
    for (i = 1; i <=m; i++)
        if (T ∪{ei} does not contain cycle)
                    T =  T ∪ {ei};
}
```

- Step 1. Sort the edges in ascending order of weight
  - Could use heap sort/merge sort : $O(m \log m)$
- Iterations: Each iteration we need to check whether $T \cup \{ e_i \}$ contains the cycle?
  - Could use DFS to check with time $O(m+n)$.
  - Total time: $O(m(m+n))$

Computation time: $O(m \log m + m(m+n))$ where n, m is the number of vertices and edges of the graph, respectively.

Improved implementation:

- Each connected component C of forest F is setup as a set.
- Denote First(C) be the first vertex in connected component C.
- Each vertex j in C, set First(j) = First(C) = first vertex in C.
- Note: Adding edge (i,j) to forest F creates cycle iff i and j belongs to the same connected component, it means First(i) = First(j).
- When connecting connected components C and D together, we connect the smaller one (less number of vertices) to the larger one (more number of vertices):

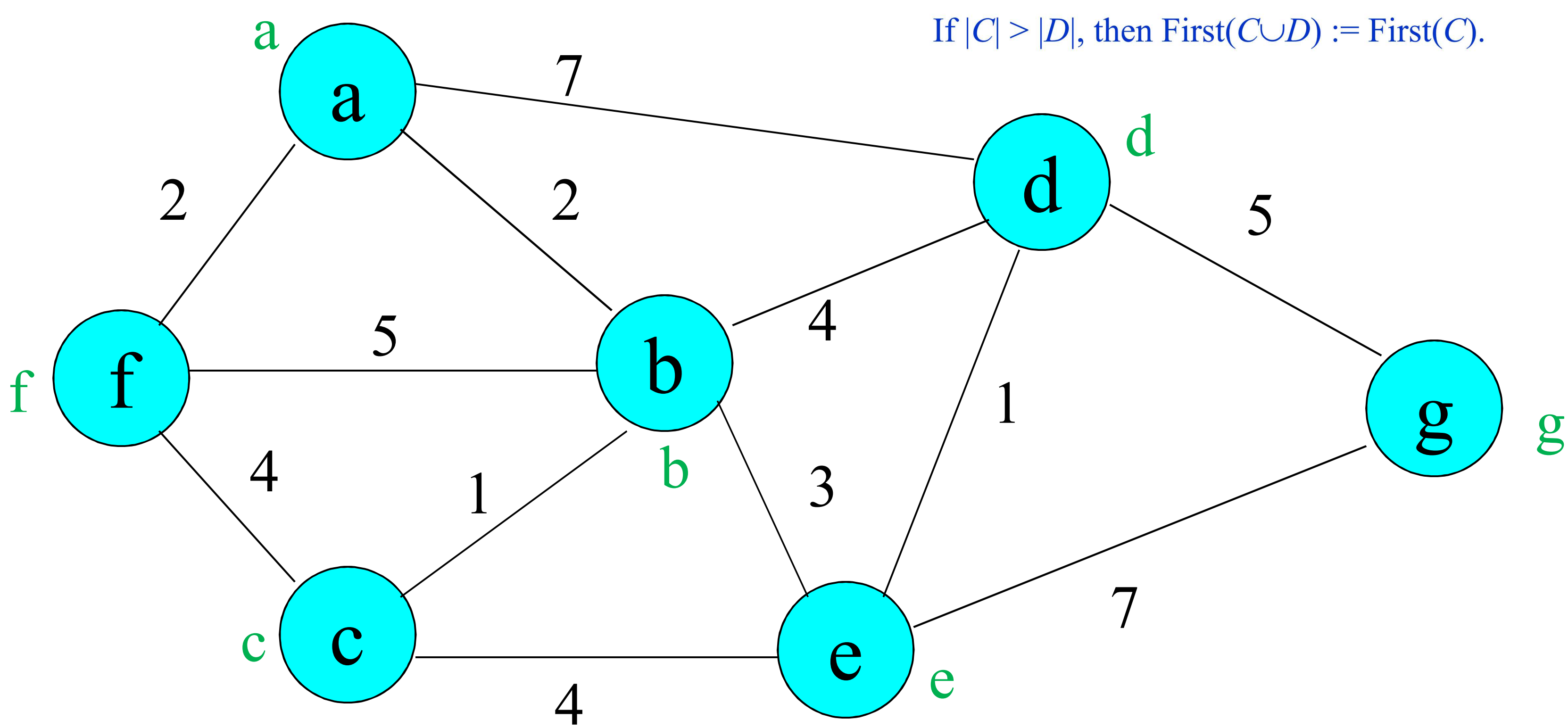
  If $|C| > |D|$, then First($C \cup D$) := First(C).

❖ *T* is **forest (empty).**
❖ The "safe" edge included in *T* at each iteration is the edge with smallest weight **among edges connecting its connected components**.

Kruskal – Example

- Denote First(C) be the first vertex in connected component C.
- Each vertex j in C, set First(j) = First(C) = first vertex in C.
- Note: Adding edge (i,j) to forest F creates cycle iff i and j belongs to the same connected component, it means First(i) = First(j).
- When connecting connected components C and D together, we connect the smaller one (less number of vertices) to the larger one (more number of vertices):

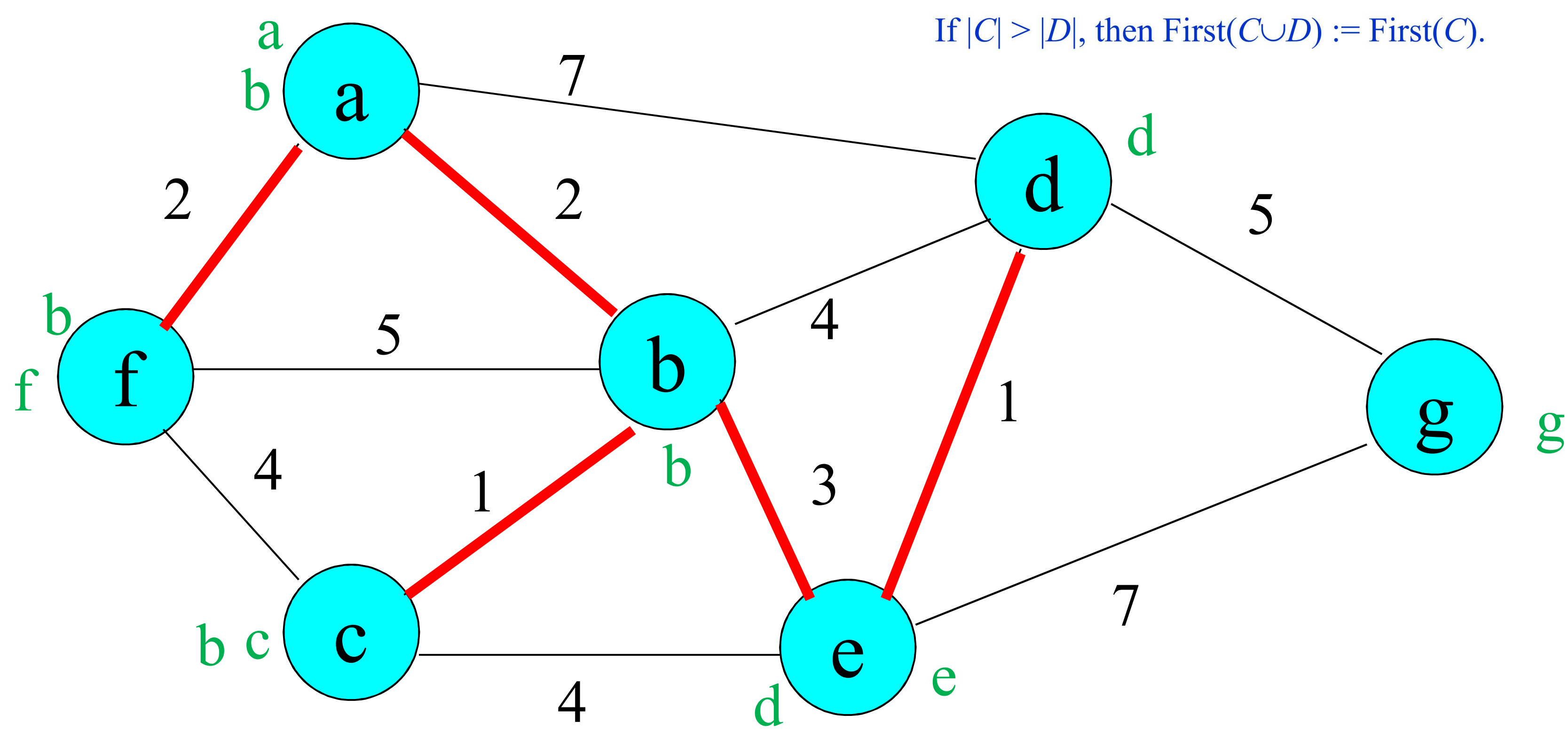
  If $|C| > |D|$, then First($C \cup D$) := First(C).



Kruskal – Example

- Denote First(C) be the first vertex in connected component C.
- Each vertex j in C, set First(j) = First(C) = first vertex in C.
- Note: Adding edge (i,j) to forest F creates cycle iff i and j belongs to the same connected component, it means First(i) = First(j).
- When connecting connected components C and D together, we connect the smaller one (less number of vertices) to the larger one (more number of vertices):

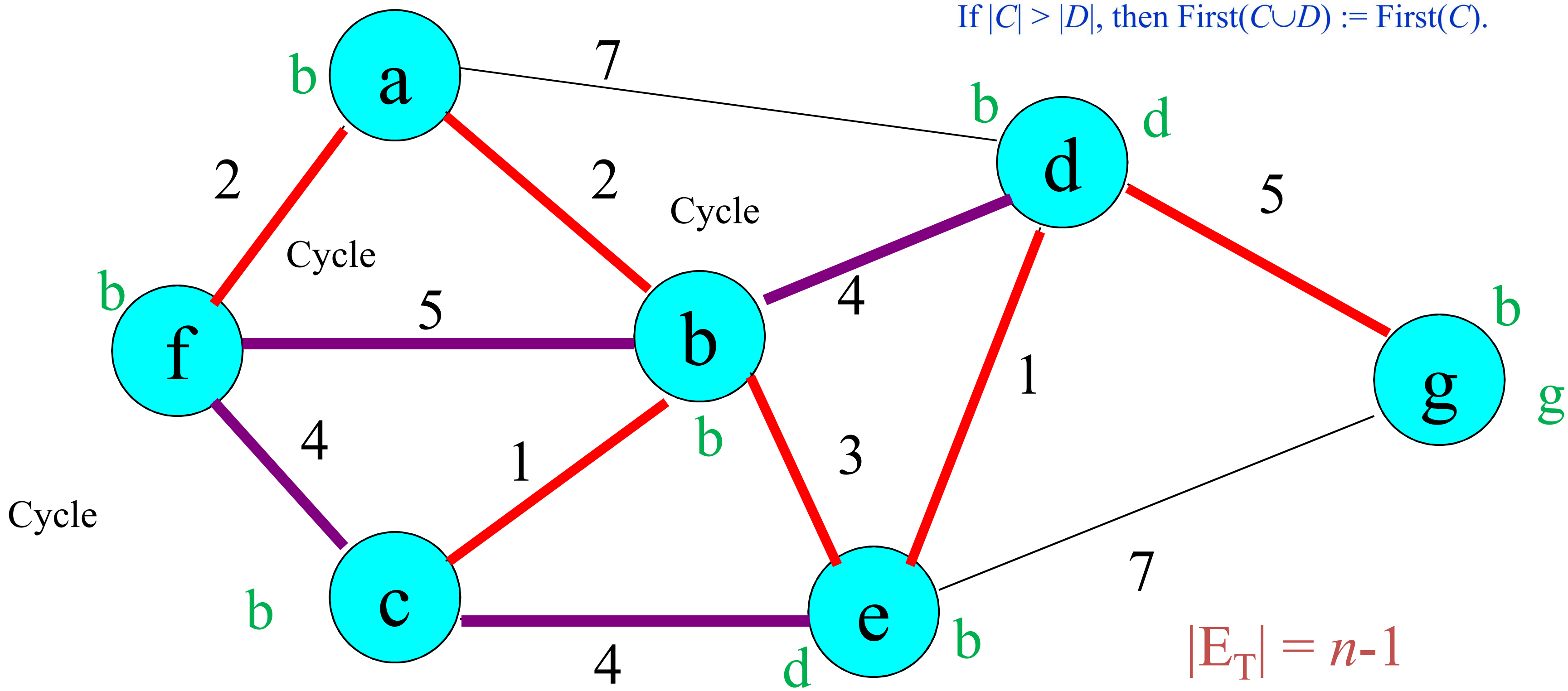
  If $|C| > |D|$, then First($C \cup D$) := First(C).



Kruskal – Example

- Denote First(C) be the first vertex in connected component C.
- Each vertex j in C, set First(j) = First(C) = first vertex in C.
- Note: Adding edge (i,j) to forest F creates cycle iff i and j belongs to the same connected component, it means First(i) = First(j).
- When connecting connected components C and D together, we connect the smaller one (less number of vertices) to the larger one (more number of vertices):

  If $|C| > |D|$, then First($C \cup D$) := First(C).



Cycle as vertex c and e belongs to the same connected component

$|E_T| = n-1$

Length of MST: 14

Improved implementation: Computation time

- Time to determine whether 2 vertices i, j belong to the same connected component: First(i) = First(j) : $O(1)$ for each i, j. There are m edges → Total: $O(m)$.
- Time to connect 2 connected components S and Q, where $|S| \geq |Q|$:
  - $O(1)$ for each vertex of Q (the one with smaller number of vertices)
  - Each vertex i in smaller connected component: connect $\log n$ times as maximum. (As the number vertices of connected component containing i is doubled after each connection.)

Total time to connect over all algorithm: $O(n \log n)$.

- Computation time:

$$O(m \log m + m + n \log n).$$

Improved implementation: Computation time

```
void Kruskal ( )
{
    Sort m edges of the graph e1,e2, . . . , em in ascending order of weight;
    T = ∅;       // T: set of edges of the minimum spanning tree
    for (i = 1; i <=m; i++)
        if (T ∪{ei} does not contain cycle)
                    T =  T ∪ {ei};
}
```

Time to connect 2 connected components S and Q, where $|S| \geq |Q|$:

- $O(1)$ for each vertex of Q (the one with smaller number of vertices)
- Each vertex i in smaller connected component: connect $\log n$ times as maximum. (As the number vertices of connected component containing i is doubled after each connection.)

➔ Total time to connect over all algorithm: $O(n \log n)$.

- Step 1. Sort the edges in ascending order of weight
  - Could use heap sort/merge sort : $O(m \log m)$
- Iterations: Each iteration we need to check whether $T \cup \{ e_i = (x, y)\}$ contains the cycle?
  - Could use DFS to check with time $O(m+n)$. ⟹ O(1): check First(x) = First(y)
  - Total time: $O(m(m+n))$ ⟹ O(m)

Computation time: $O(m \log m + m(m+n))$ where n, m is the number of vertices and edges of the graph, respectively.

⟹ $O(m \log m + m + n \log n)$.

PRIM algorithm

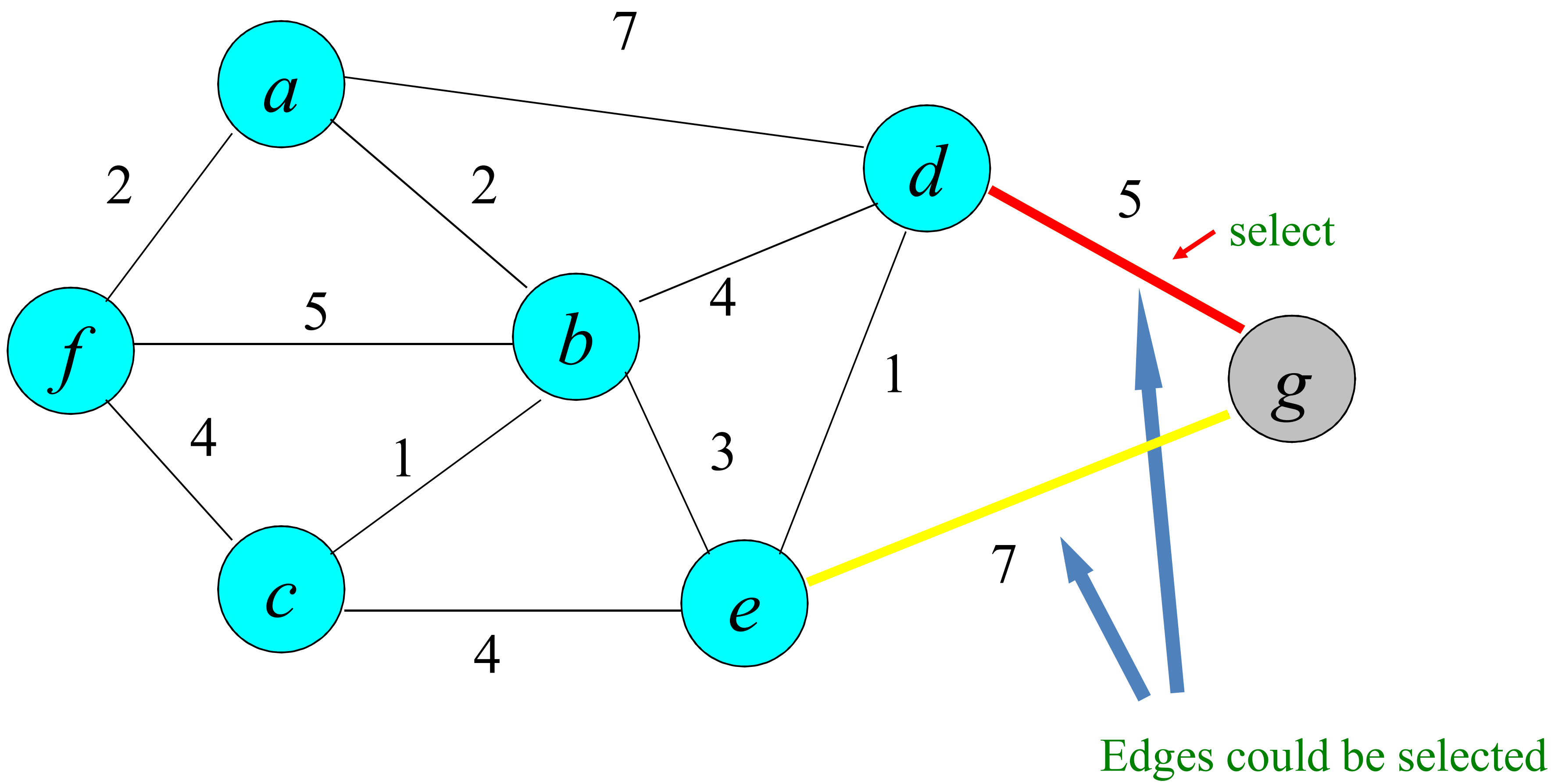
```
Generic-MST(G, c)
   T = ∅
   //T  is the subset edges of a minimum spanning tree
   while T is not the spanning tree do
        Finding edge (u, v) is "safe" edge for T
        T = T ∪{(u, v)}
   return T
```

Robert Clay Prim
(1921 - ~)

PRIM algorithm:

- *T* is tree (initialize: T has one vertex).
- The "safe" edge included in *T* at each iteration is the edge with smallest weight **among edges connecting a vertex of T to other vertex not in T**

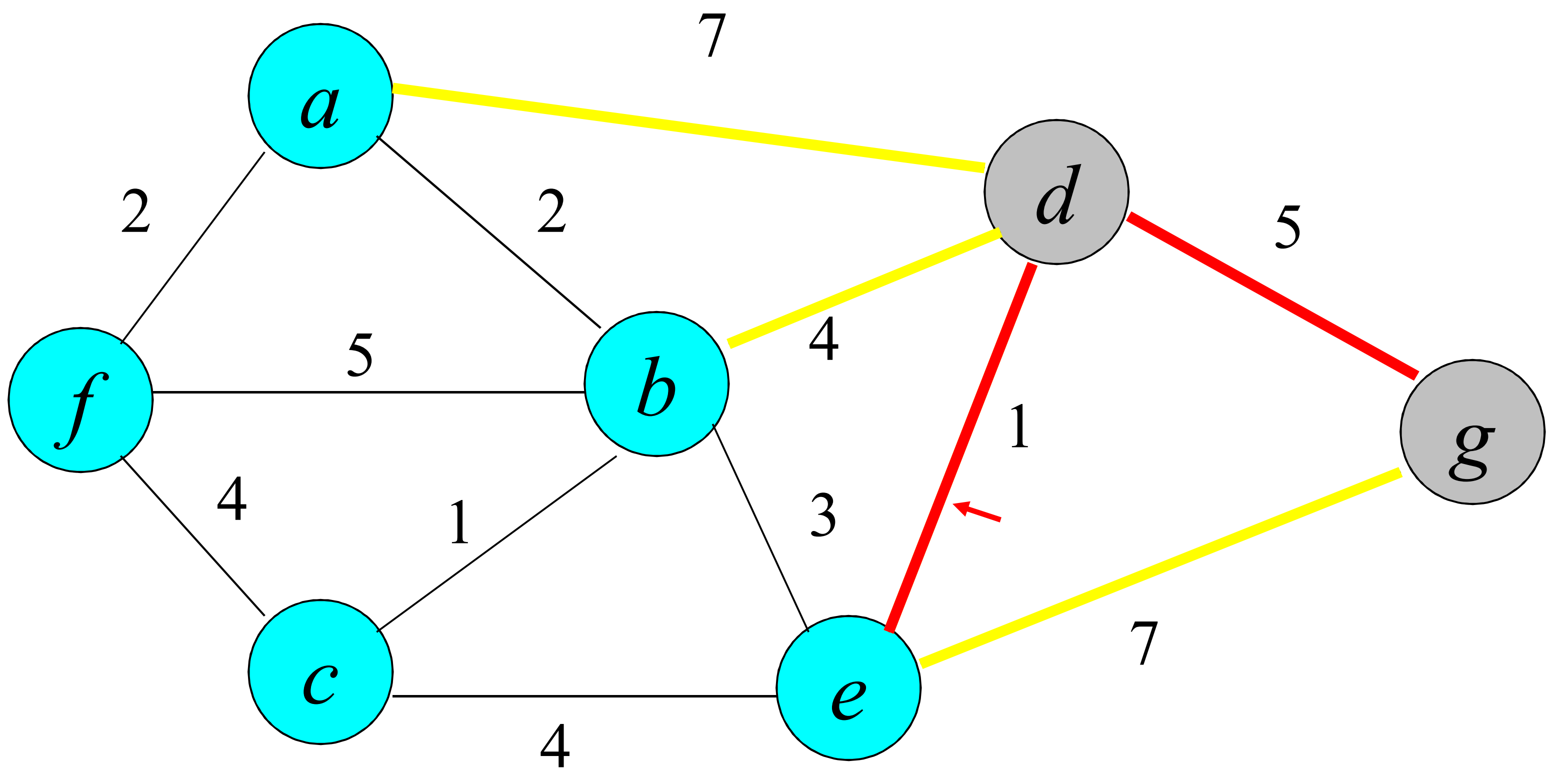
PRIM algorithm: an example



PRIM algorithm:

- *T* is tree (initialize: T has one vertex).
- The "safe" edge included in *T* at each iteration is the edge with smallest weight **among edges connecting a vertex of T to other vertex not in T**

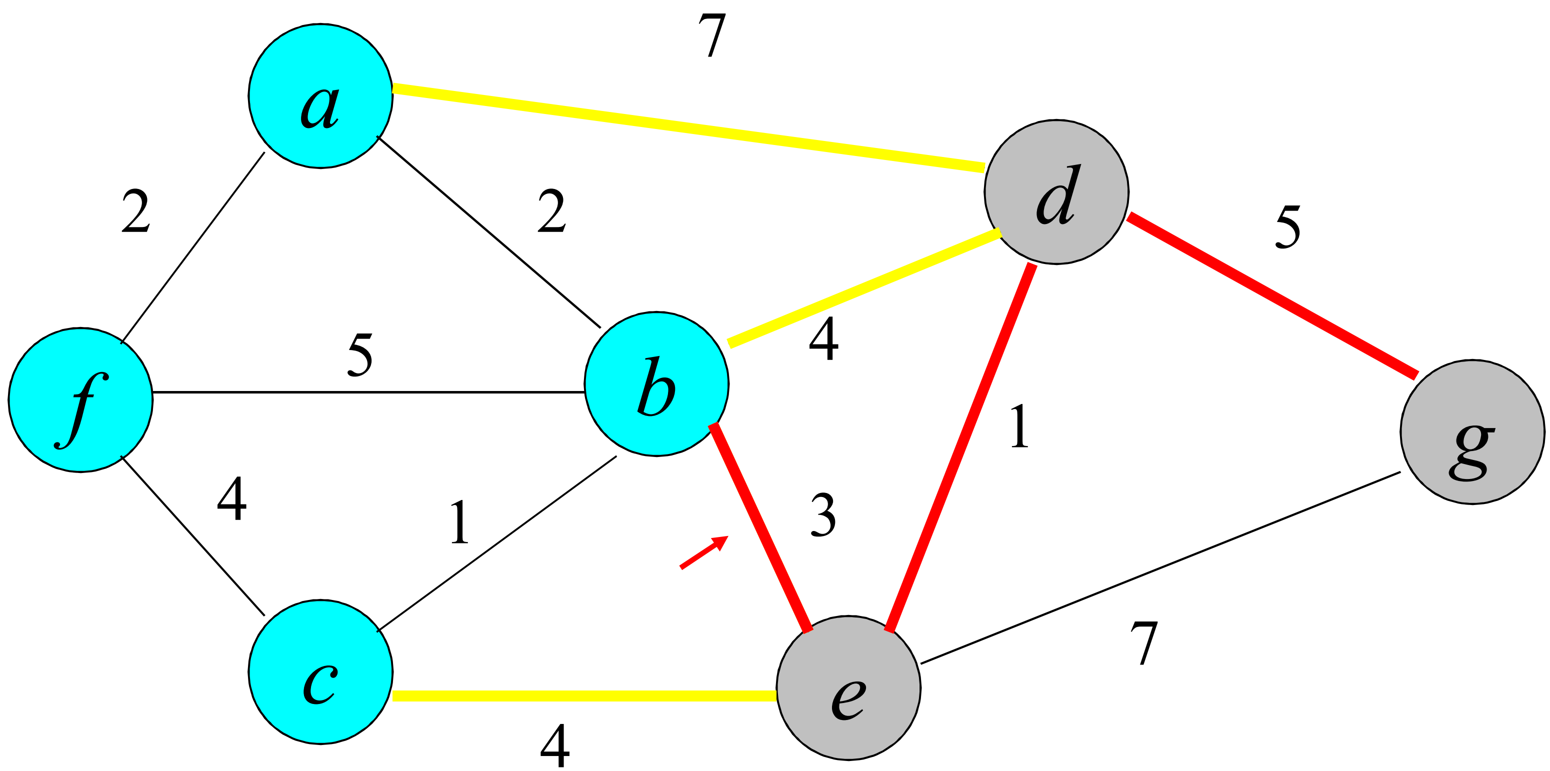
PRIM algorithm: an example



PRIM algorithm:

- *T* is tree (initialize: T has one vertex).
- The "safe" edge included in *T* at each iteration is the edge with smallest weight **among edges connecting a vertex of T to other vertex not in T**

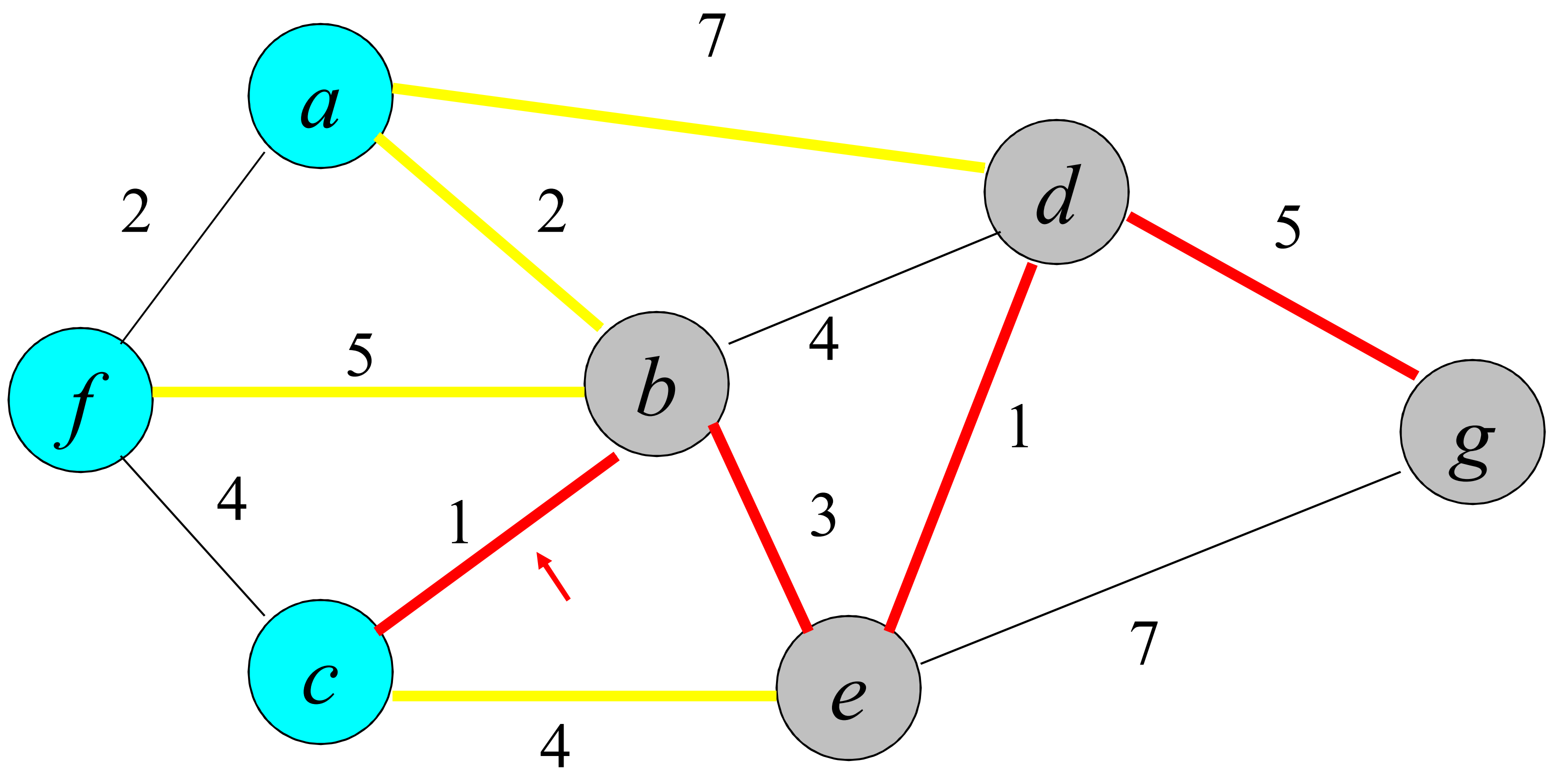
PRIM algorithm: an example



PRIM algorithm:

- *T* is tree (initialize: T has one vertex).
- The "safe" edge included in *T* at each iteration is the edge with smallest weight **among edges connecting a vertex of T to other vertex not in T**

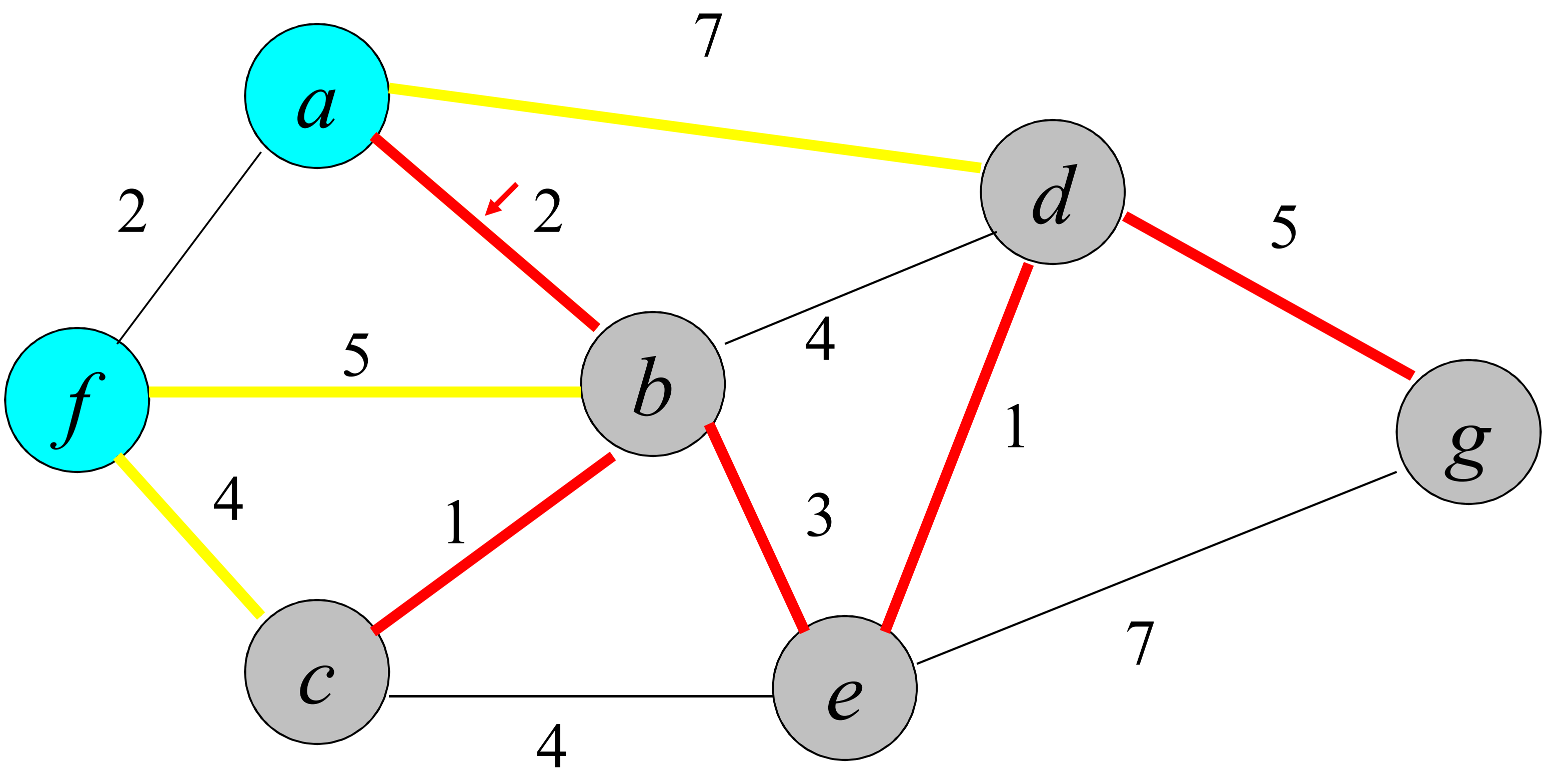
PRIM algorithm: an example



PRIM algorithm:

- *T* is tree (initialize: T has one vertex).
- The "safe" edge included in *T* at each iteration is the edge with smallest weight **among edges connecting a vertex of T to other vertex not in T**

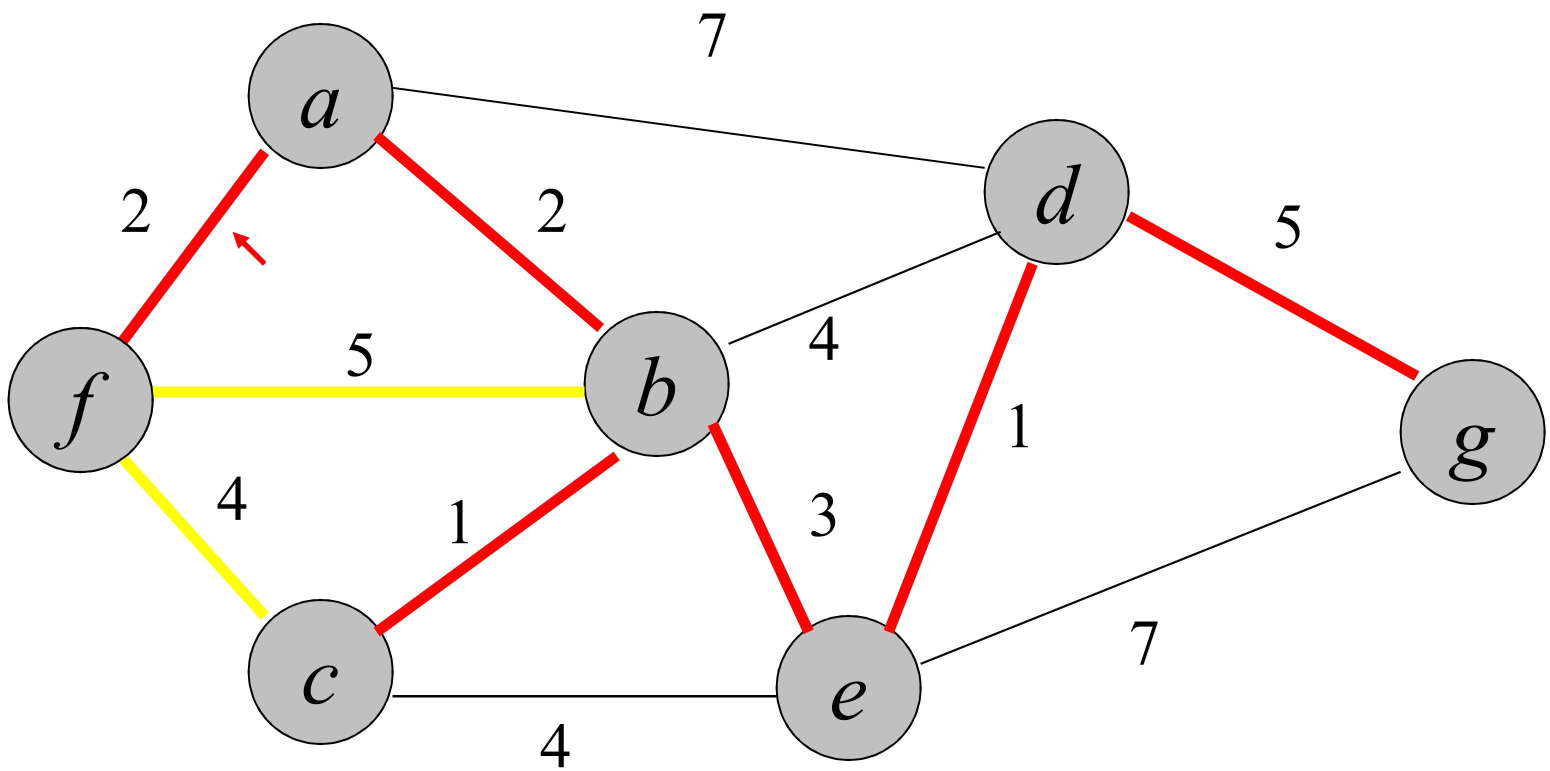
PRIM algorithm: an example



PRIM algorithm:

- *T* is tree (initialize: T has one vertex).
- The "safe" edge included in *T* at each iteration is the edge with smallest weight **among edges connecting a vertex of T to other vertex not in T**

PRIM algorithm: an example



Minimum spanning tree with edges: (*g*,*d*), (*d*,*e*), (*e*,*b*), (*b*,*c*), (*b*,*a*), (*a*,*f*)

The weight: 14 5+1+3+1+2+2 = 14

PRIM algorithm

void Prim(G, C) //G: graph; C: weight matrix

{

Tree T with only one vertex r

Select an arbitrary vertex $r \in V$;

Initialize: tree $T=(V(T), E(T))$ where $V(T)= \{r\}$ and $E(T)=\varnothing$;

while (T has $< n$ *vertices*)

Among edges connecting a vertex of T to other vertex not in T: find **the edge with minimum weight**

{

(u, v) is the minimum weight where $u \in V(T)$ and $v \in V(G) - V(T)$

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

$V(T) \leftarrow V(T) \cup \{ v \}$

add vertex v and edge (u,v) to tree T

}

}

PRIM: Implementation

- Graph with weight matrix $C = \{c[i,j],\ i, j = 1, 2, \ldots, n\}$.
- At each iteration: to select quickly a vertex and an edge to add to spanning tree, vertices of graph are labeled:

Label of a vertex $v \in V \setminus V(T)$ in the form $[d[v], near[v]]$:

- $d[v]$:use to record the distance from vertex v to vertex set $V(T)$:

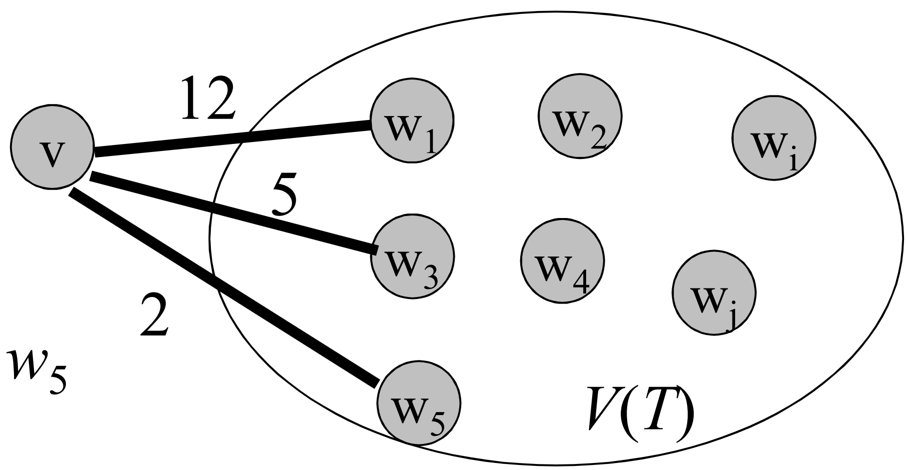
$$\boxed{d[v] := \min\{\ c[v, w] :\ w \in V(T)\ \}\ (= c[v, z])}$$

Edge with minimum edge among edges connecting vertex v to other vertex of $V(T)$

- $near[v] := z$ record the vertex of T that is nearest to vertex v

$d(v) = 2$

$near[v] = w_5$



```
void Prim(G, C) //G: graph; C: weight matrix
{
    Select an arbitrary vertex r ∈ V;
    Initialize: tree T=(V(T), E(T)) where V(T)= {r} and E(T)=∅;
    while (T has < n vertices)
    {
        (u, v) is the minimum weight where u ∈ V(T) and v ∈ V(G) – V(T)
        E(T) ← E(T) ∪ { (u, v) };
        V(T) ← V(T) ∪ { v }
    }
}
```

```
void  Prim ( ) {
    // Initialize:
    V(T) = { r }; E(T) = ∅ ;
    d[r] = 0; near[r] = r;
    for  v ∈ V \ V(T)
    {
        d[v] = c[r,v];  near[v] = r;
    }
    // Iteration:
    for k in range (2, n+1)
    {
        Find v ∈ V \ V(T) satisfying:  d[v] = min { d[i] : i ∈ V \ V(T) };
        V(T) = V(T) ∪ { v };   E(T) = E(T) ∪ { ( v, near[v] ) } ;
        for  v'∈ V \ V(T)
            if (d[v'] > c[v,v'])
            {
                d[v'] = c[v,v'] ;  near[v'] = v;
            }
    }
    T is the minimum spanning tree;
}
```

Prepare data for finding "safe" edge process

d[v]: the edge with minimum weight connecting vertex v *(not yet in the spanning tree* T to other vertex of T

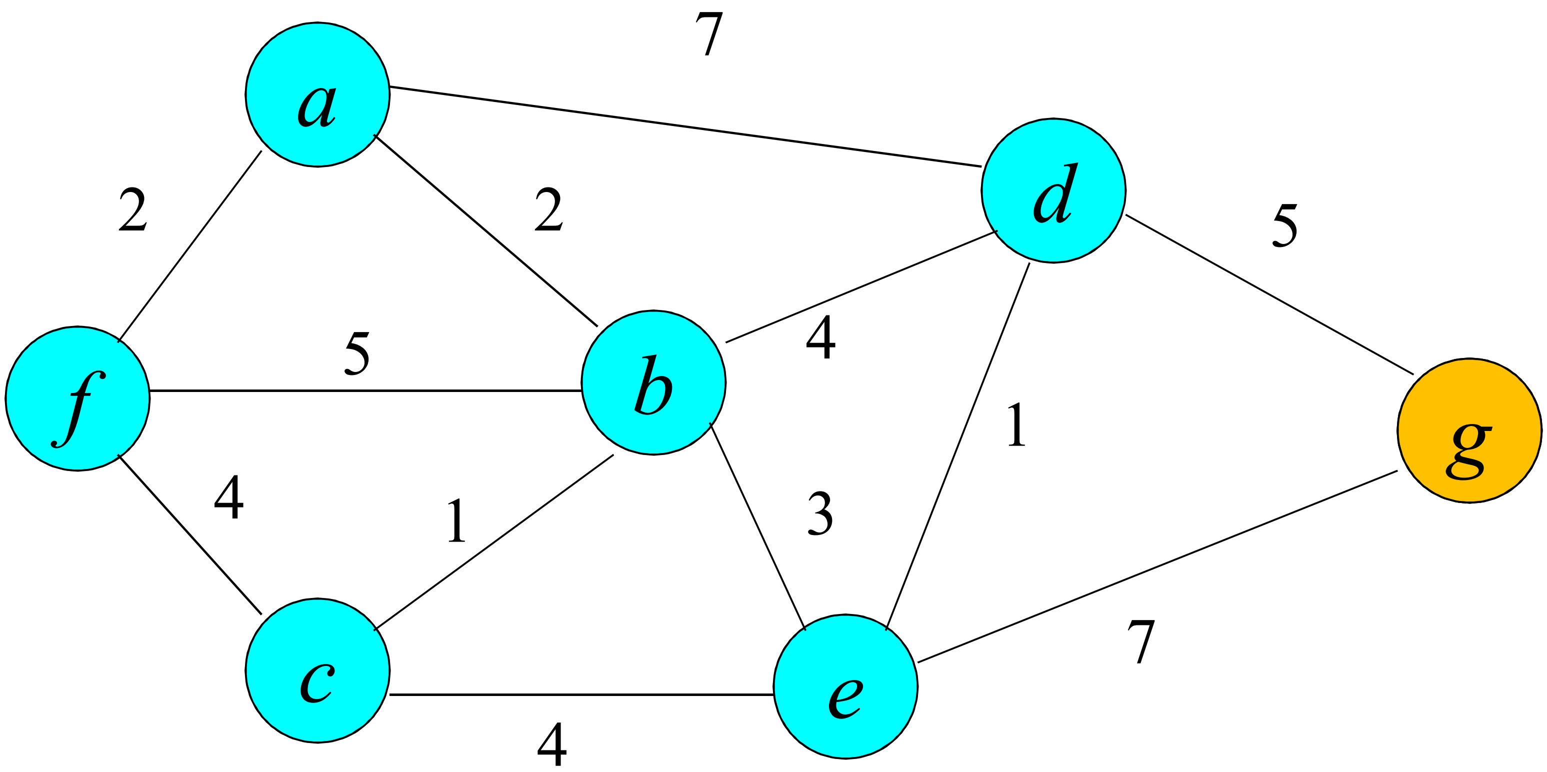
Because we have just changed the spanning tree T : vertex v has just been added to T

→ Need to update label of vertices not yet in T if necessary

Computation time: $O(|V|^2)$

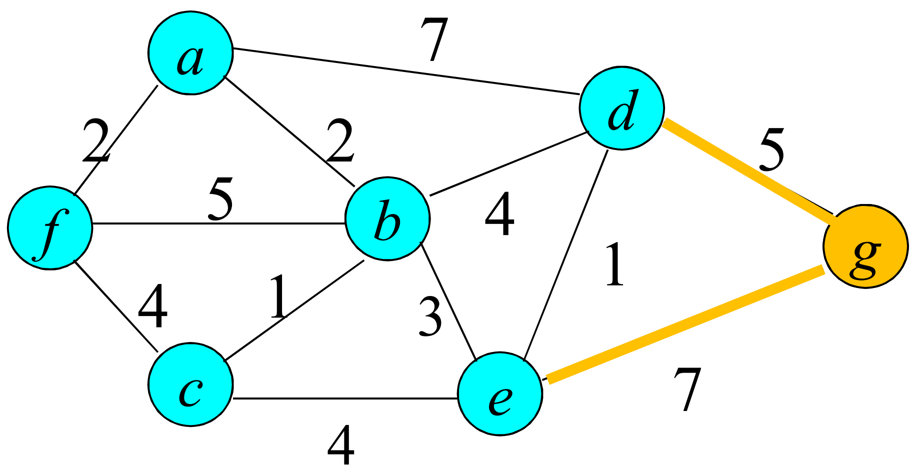
PRIM: an example

Find the minimum spanning tree of the graph



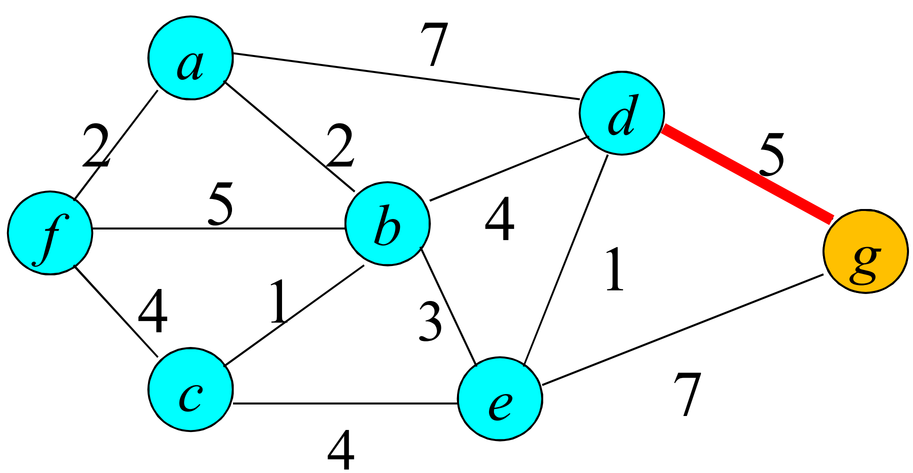
Vertex a	Vertex b	Vertex c	Vertex d	Vertex e	Vertex f	Vertex g	$V(T)$
[∞, g]	[∞, g]	[∞, g]	[5, g]	[7, g]	[∞, g]	[0, g]	g

Find $v \in V \setminus V(T)$ *satisfying*: $\mathrm{d}[v] = \min \{ \mathrm{d}[i] : i \in V \setminus V(T) \}$;



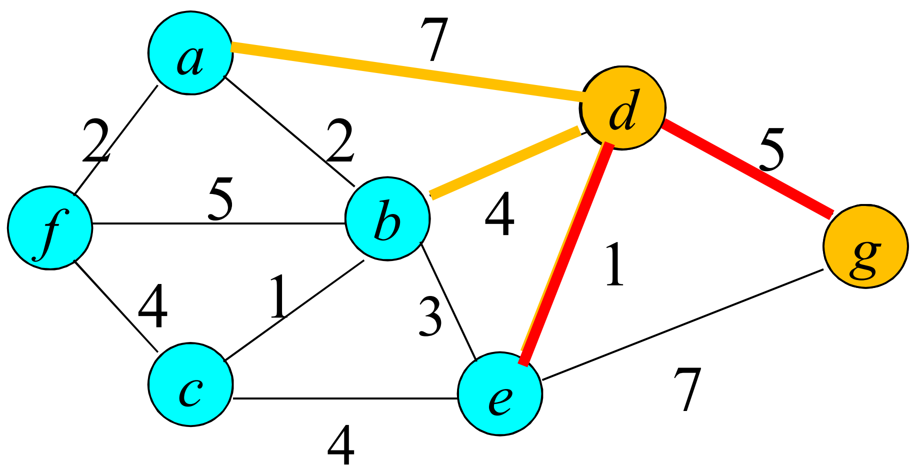
Vertex a	Vertex b	Vertex c	Vertex d	Vertex e	Vertex f	Vertex g	$V(T)$
[∞, g]	**[∞, g]**	**[∞, g]**	**[5, g]**	**[7, g]**	**[∞, g]**	**[0, g]**	**g**

Find $v \in V \setminus V(T)$ *satisfying*: $\mathrm{d}[v] = \min \{ \mathrm{d}[i] : i \in V \setminus V(T) \}$;



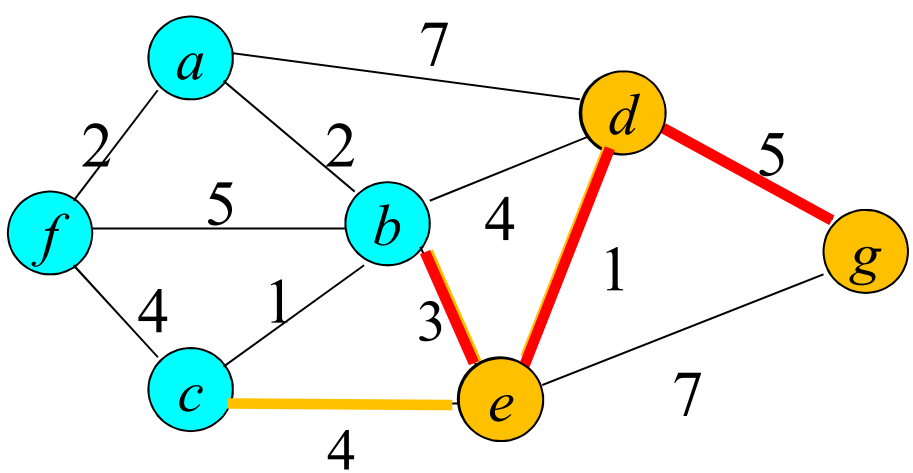
Vertex a	Vertex b	Vertex c	Vertex d	Vertex e	Vertex f	Vertex g	$V(T)$
[∞, g]	[∞, g]	[∞, g]	[5, g]	[7, g]	[∞, g]	[0, g]	g
[7, d]	[4, d]	[∞, g]	-	[1, d]	[∞, g]	-	g, d

Find $v \in V \setminus V(T)$ *satisfying*: $d[v] = \min \{ d[i] : i \in V \setminus V(T) \}$;



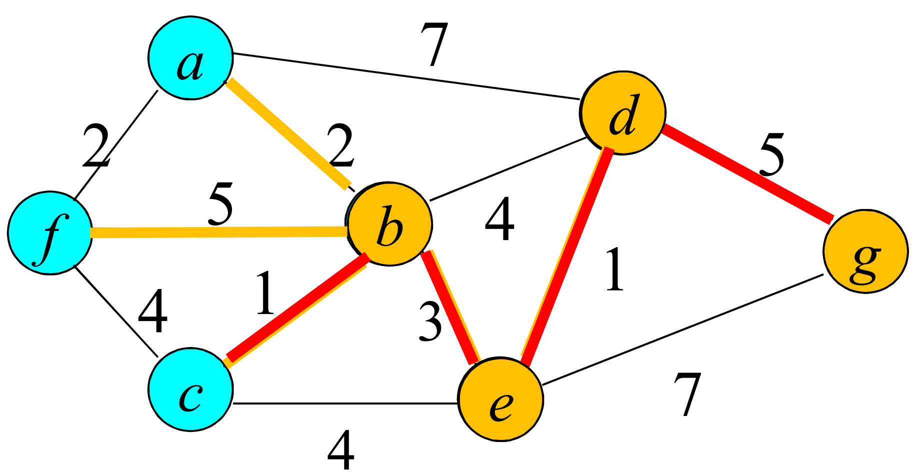
Vertex a	Vertex b	Vertex c	Vertex d	Vertex e	Vertex f	Vertex g	$V(T)$
[∞, g]	[∞, g]	[∞, g]	[5, g]	[7, g]	[∞, g]	[0, g]	g
[7, d]	[4, d]	[∞, g]	-	[1, d]	[∞, g]	-	g, d
[7, d]	[3, e]	[4, e]	-	-	[∞, g]	-	g, d, e

Find $v \in V \setminus V(T)$ *satisfying*: $d[v] = \min \{ d[i] : i \in V \setminus V(T) \}$;



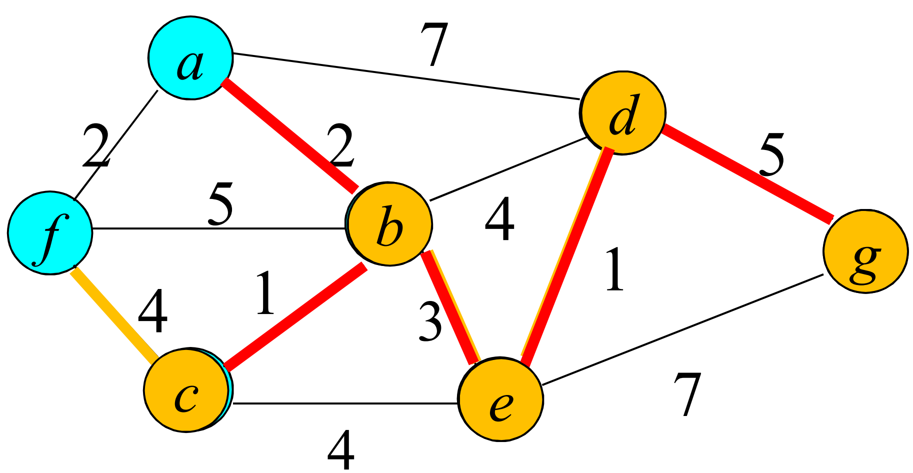
Vertex a	Vertex b	Vertex c	Vertex d	Vertex e	Vertex f	Vertex g	$V(T)$
[∞, g]	[∞, g]	[∞, g]	[5, g]	[7, g]	[∞, g]	[0, g]	g
[7, d]	[4, d]	[∞, g]	-	[1, d]	[∞, g]	-	g, d
[7, d]	[3, e]	[4, e]	-	-	[∞, g]	-	g, d, e
[2, b]	-	[1, b]	-	-	[5, b]	-	g, d, e, b

Find $v \in V \setminus V(T)$ *satisfying*: $d[v] = \min \{ d[i] : i \in V \setminus V(T) \}$;



Vertex a	Vertex b	Vertex c	Vertex d	Vertex e	Vertex f	Vertex g	$V(T)$
[∞, g]	[∞, g]	[∞, g]	[5, g]	[7, g]	[∞, g]	[0, g]	g
[7, d]	[4, d]	[∞, g]	-	[1, d]	[∞, g]	-	g, d
[7, d]	[3, e]	[4, e]	-	-	[∞, g]	-	g, d, e
[2, b]	-	[1, b]	-	-	[5, b]	-	g, d, e, b
[2, b]	-	-	-	-	[4, c]	-	g, d, e, b, c

Find $v \in V \setminus V(T)$ *satisfying*: $d[v] = \min \{ d[i] : i \in V \setminus V(T) \}$;



Vertex a	Vertex b	Vertex c	Vertex d	Vertex e	Vertex f	Vertex g	$V(T)$
[∞, g]	[∞, g]	[∞, g]	[5, g]	[7, g]	[∞, g]	[0, g]	g
[7, d]	[4, d]	[∞, g]	-	[1, d]	[∞, g]	-	g, d
[7, d]	[3, e]	[4, e]	-	-	[∞, g]	-	g, d, e
[2, b]	-	[1, b]	-	-	[5, b]	-	g, d, e, b
[2, b]	-	-	-	-	[4, c]	-	g, d, e, b, c
-	-	-	-	-	[2, a]	-	g, d, e, b, c, a
							g, d, e, b, c, a, f

Minimum spanning tree edges:

Weight : 14