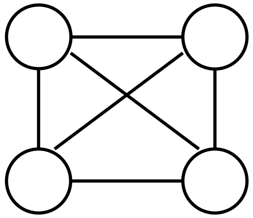
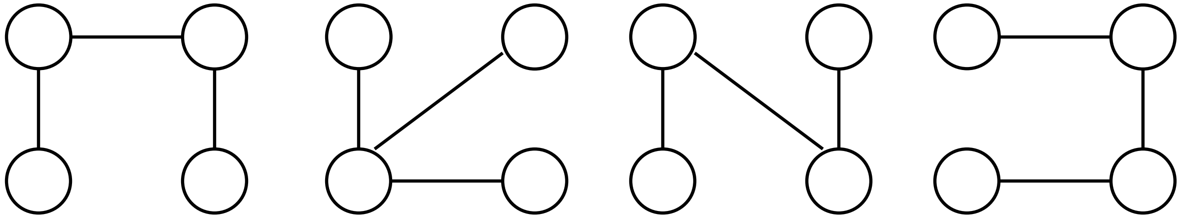
Spanning Trees生成树

Do you remember spanning subgraph?

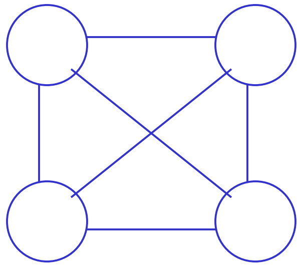
Spanning trees 生产树

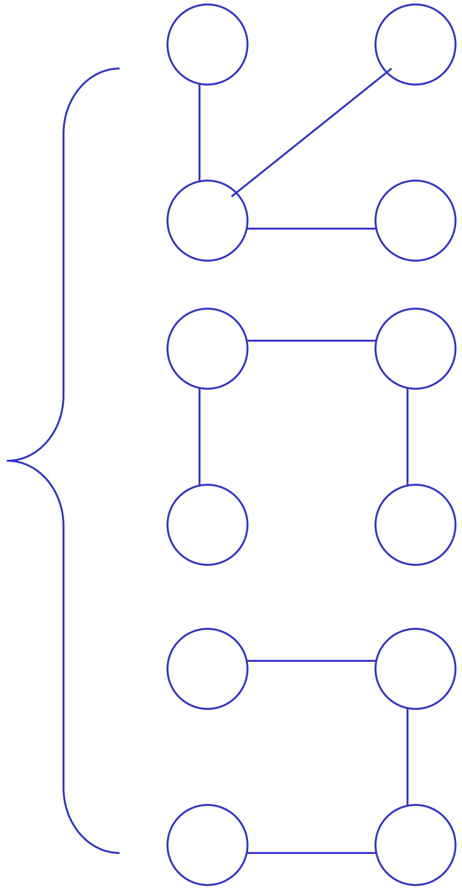
- **图的生成树：**简单连通图的一个*生成子图*，如果本身是一棵树，则称为生成树


A connected, undirected graph

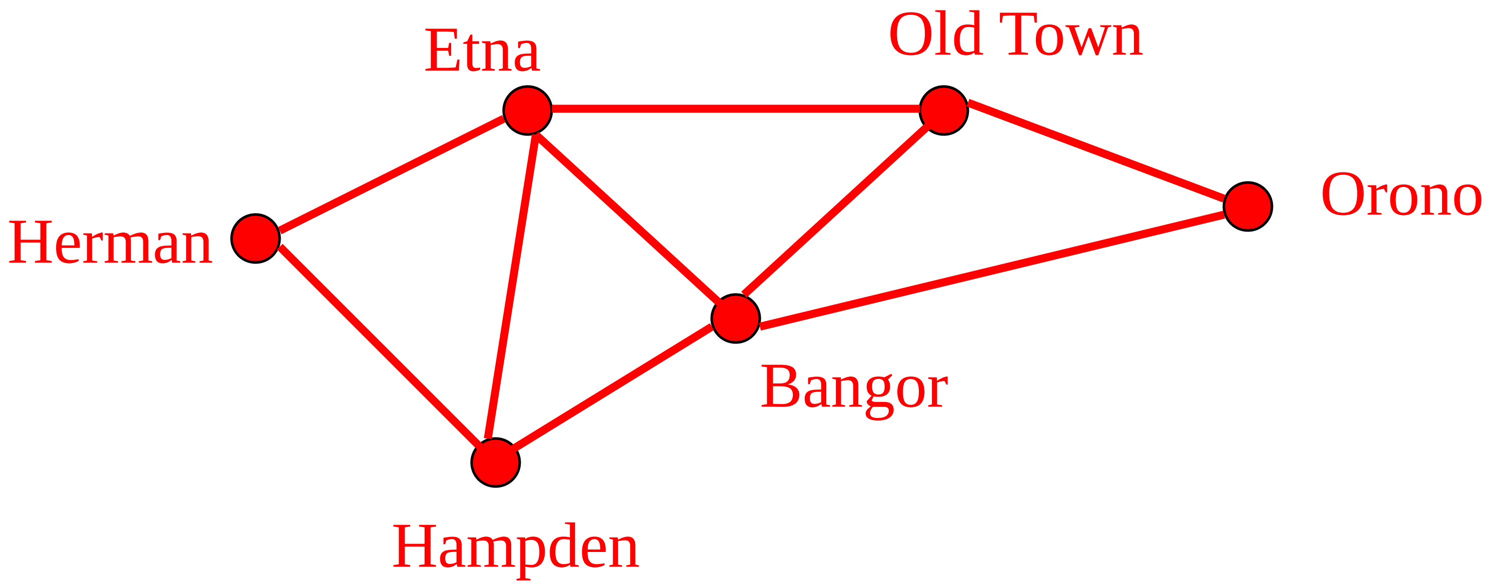

Four of the spanning trees of the graph

Undirected graph and 3 of its spanning trees


Undirected Graph

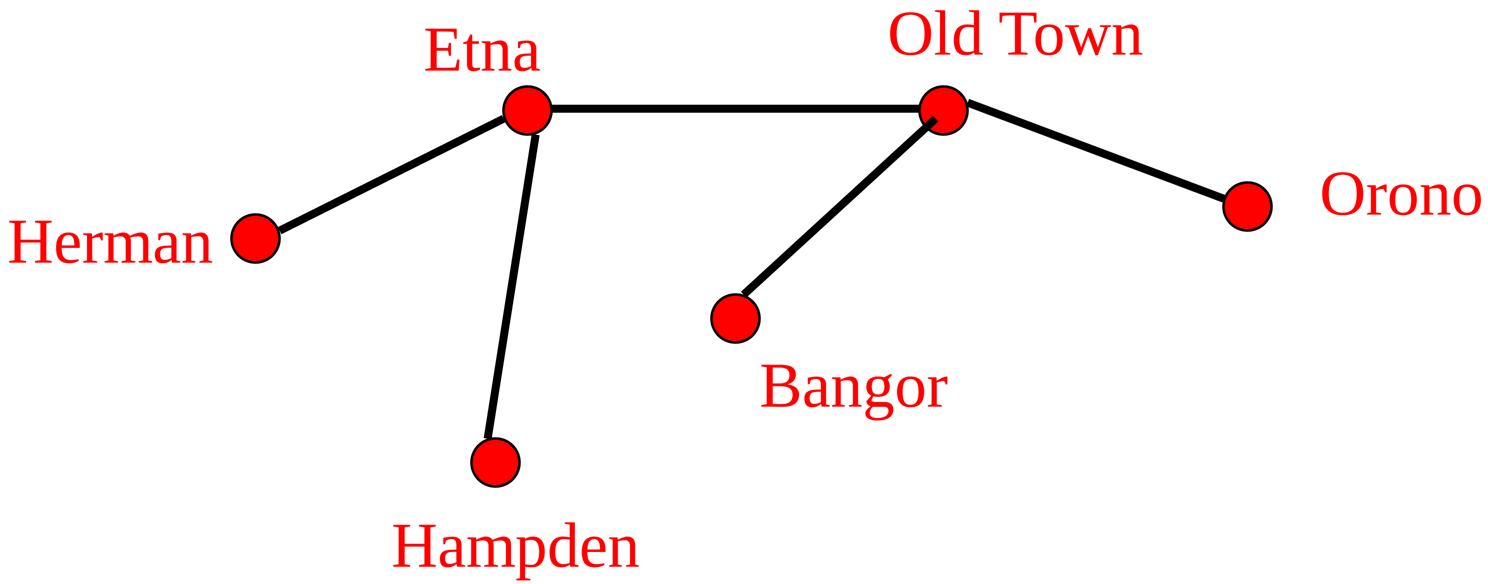


Spanning Tree 生成树



How do we plow the fewest roads so there will always be cleared roads connecting any two towns?

Spanning Tree



定理

- **定理：** 一个简单图是连通的当且仅当它有一棵生成树
- Proof:…

Question 问题

- Question: what is the difference between connected graph and tree? 连通图跟树之间有何区别？

- How to get the spanning tree from a connected graph 如何求得一个连通图的生成树？

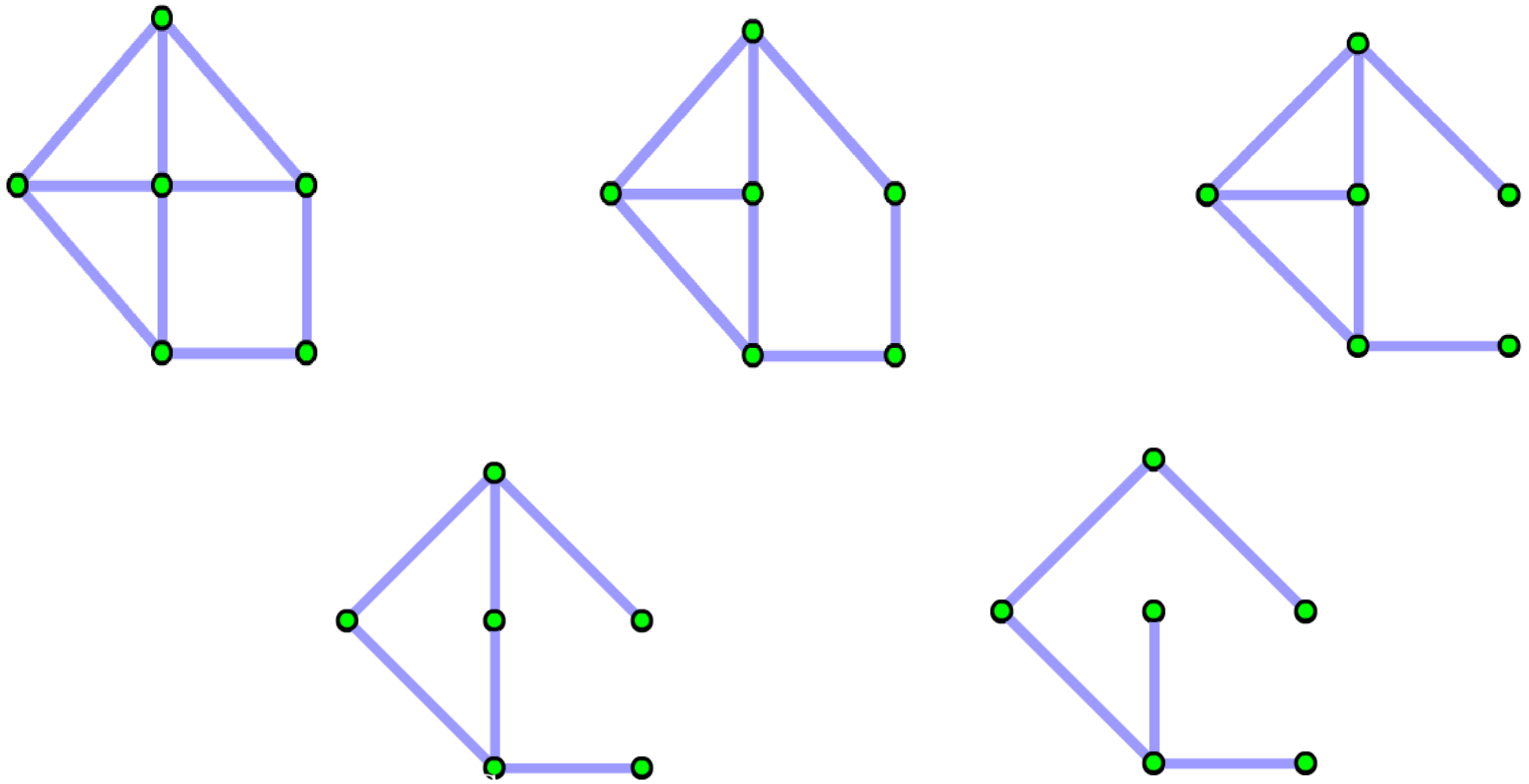
- 一个n个结点的连通图，至少有多少条边？

Finding a spanning tree 求生成树

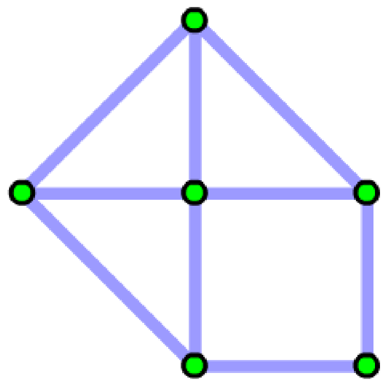
Finding a spanning tree of a connected graph

（a）break the simple cycle （破环法）

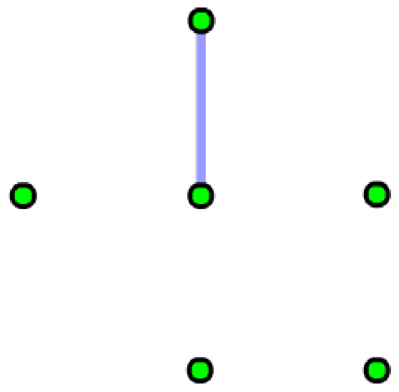
Example:

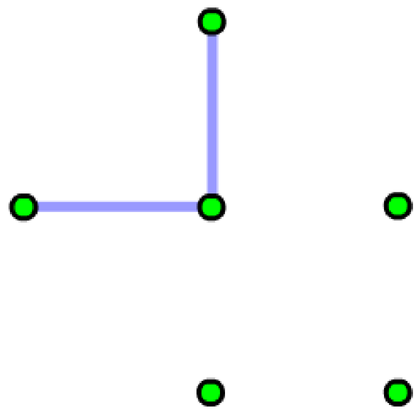


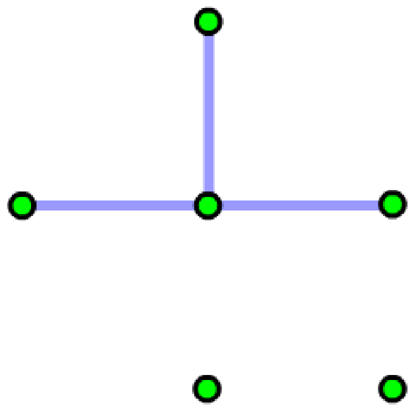
（b）Add edges without cycle （避环法）

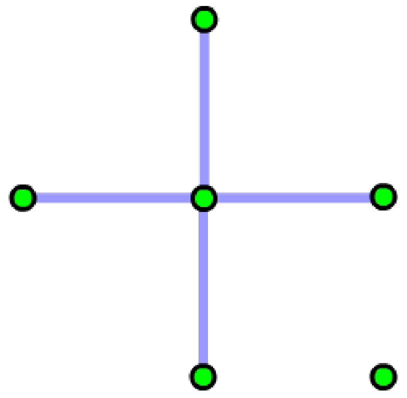


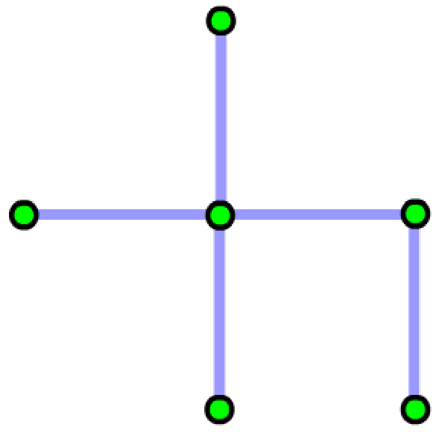
Example:





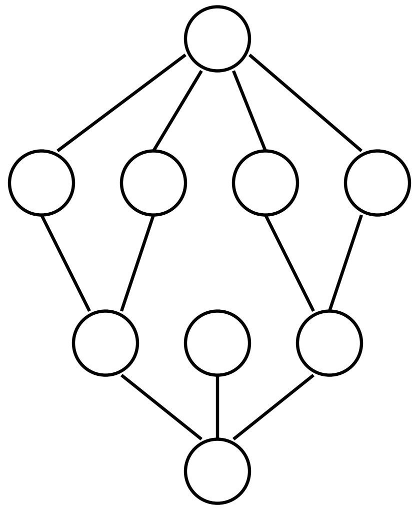


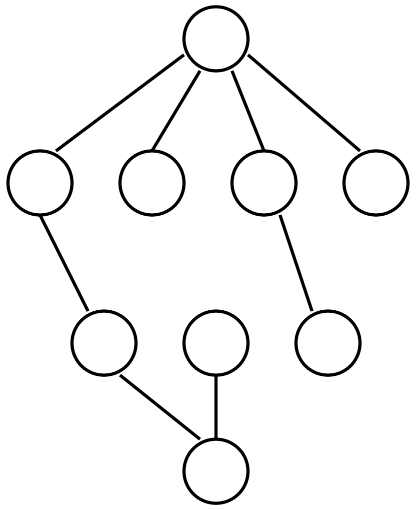


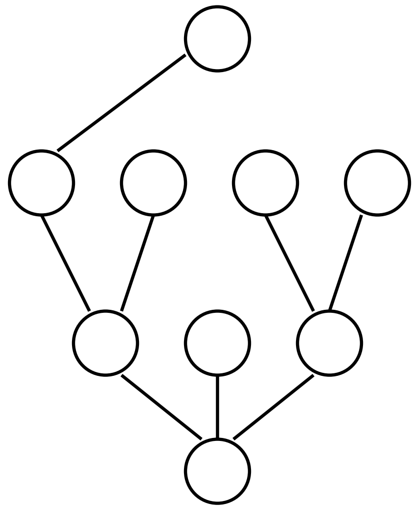


Finding a spanning tree 求生成树

- 前面的两种思路中，第一种思路很自然，但效率低，它需要找出所有的简单回路；
- 基于后面的添加边的思路，有两个经典的算法:BFS, DFS.


An undirected graph


Result of a BFS starting from top


Result of a DFS starting from top

Depth-First-Search (纵向优先搜索)

- **Depth-first search** (**DFS**) is an **algorithm** for traversing or searching a **tree, tree structure**, or **graph**. One starts at the root (selecting a node as the root in the graph case) and explores as far as possible along each branch before backtracking (回溯).
- Formally, DFS is an **uninformed search** that progresses by expanding the first child node of the search **tree** that appears and thus going deeper and deeper until a goal node is found, or until it hits a node that has no children. Then the search **backtracks**, returning to the most recent node it hasn't finished exploring. In a non-recursive implementation, all freshly expanded nodes are added to a **stack** for exploration.

DFS 纵向优先搜索

- 纵向优先搜索（或叫深度优先）是一种用来遍历或者搜索树（TREE）或图（GRAPH）结构的算法.搜索开始于某个根节点（从图中选取某个节点），然后在开始回溯前尽可能深的搜索树的分支。当结点所有子结点那一层都被搜索过，再回溯返回到当前结点的邻结点，继续搜索，直到遍历完整棵树。一般采用的是前序遍历，先根然后再左右结点的方式进行。

Depth-First-Search Example

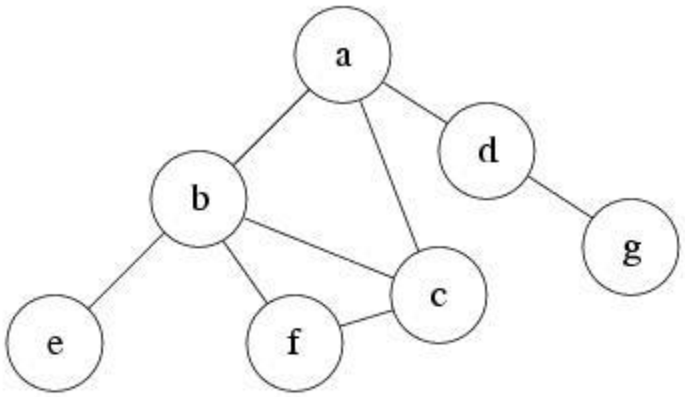


Figure 1: depth-first search 1

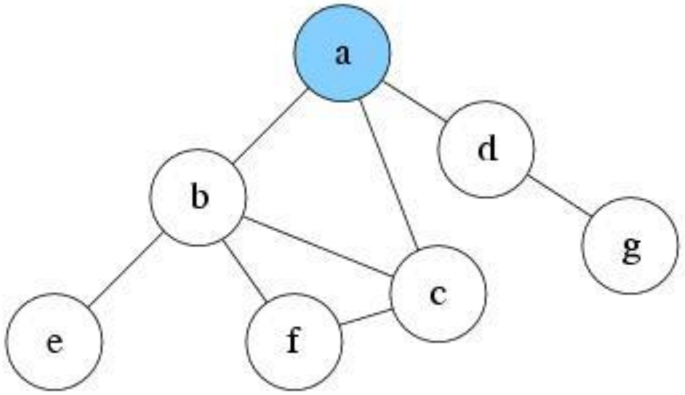


Figure 2: depth-first search 2

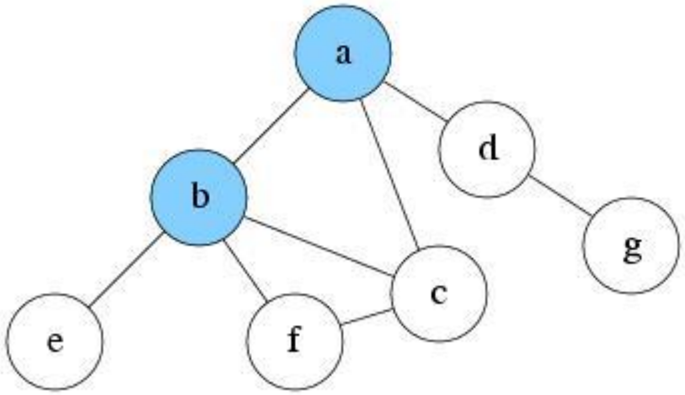


Figure 3: depth-first search 3

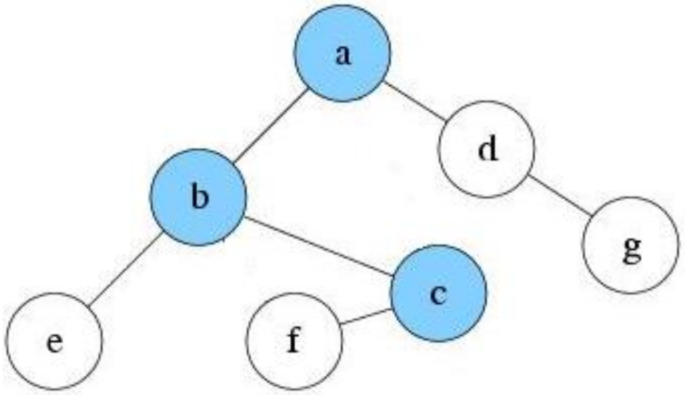


Figure 4: depth-first search 4

Depth-First-Search Example

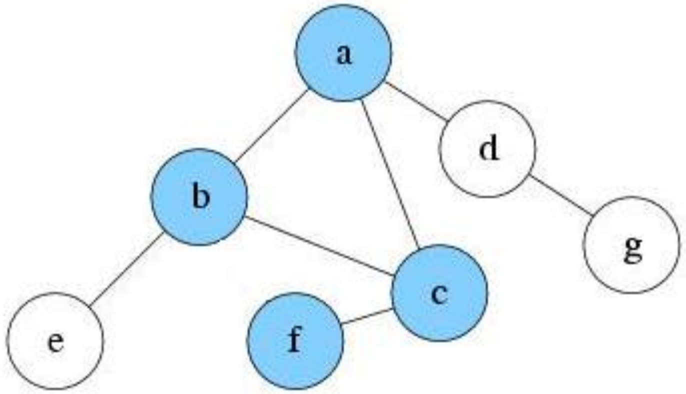


Figure 5: depth-first search 5

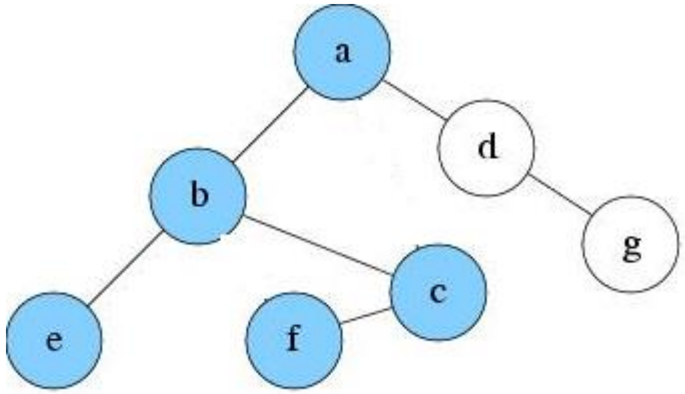


Figure 6: depth-first search 6

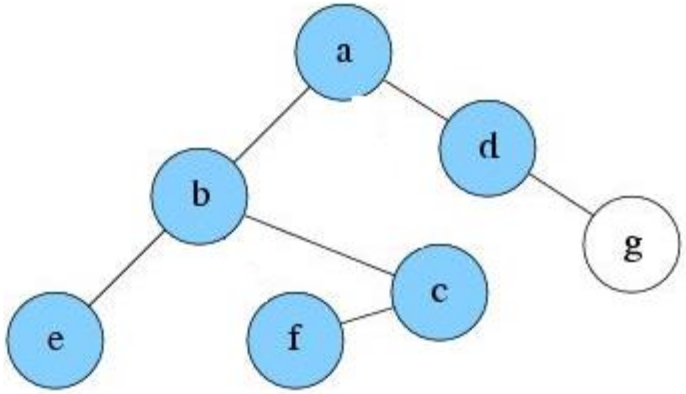


Figure 7: depth-first search 7

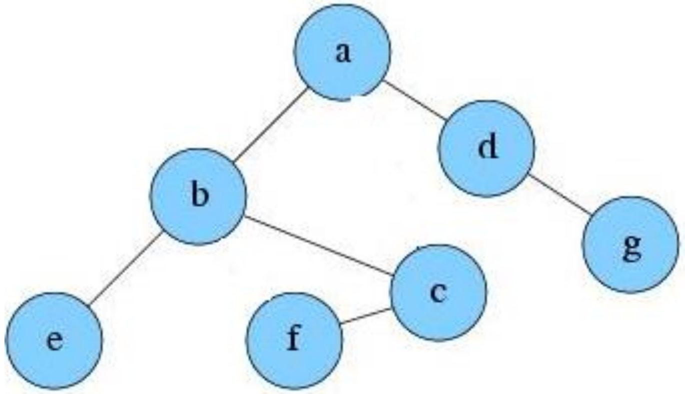


Figure 8: depth-first search 8

Depth-First-Search

- DFS Animation

Breadth-**First-Search (横向优先搜索)**

- In graph theory, **breadth-first search** (**BFS**) is a graph search algorithm that begins at the root node and explores all the neighboring nodes. Then for each of those nearest nodes, it explores their unexplored neighbor nodes, and so on, until it finds the goal.

- BFS is an uniformed search method that aims to expand and examine all nodes of a graph or combination of sequences by systematically searching through every solution. In other words, it exhaustively searches the entire graph or sequence without considering the goal until it finds it. It does not use a heuristic algorithm.

BFS横向优先

- 横向优先（或广度优先）搜索(breadth first search, BFS)是图的一种遍历策略，搜索过程：先访问节点v；再依次访问与v相邻的节点；访问这些节点之后，再访问与之相邻的节点。也就是说，从广度上进行搜索。

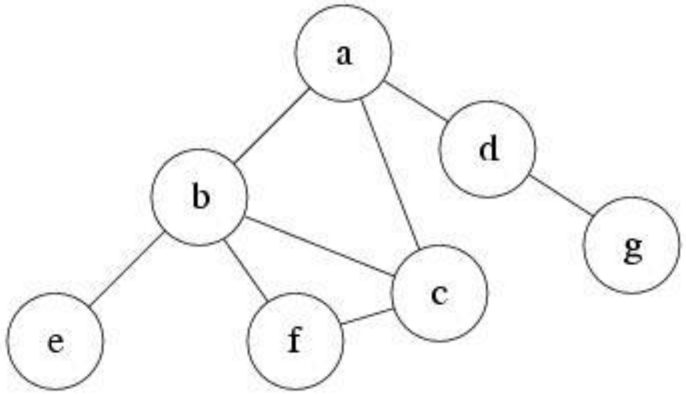


Figure 1: breadth-first search 1

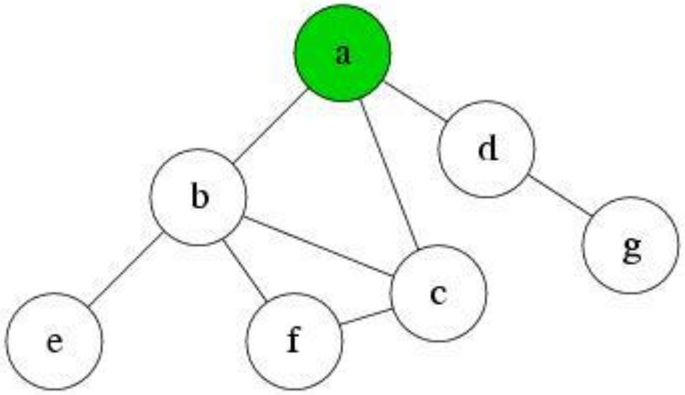


Figure 2: breadth-first search 2

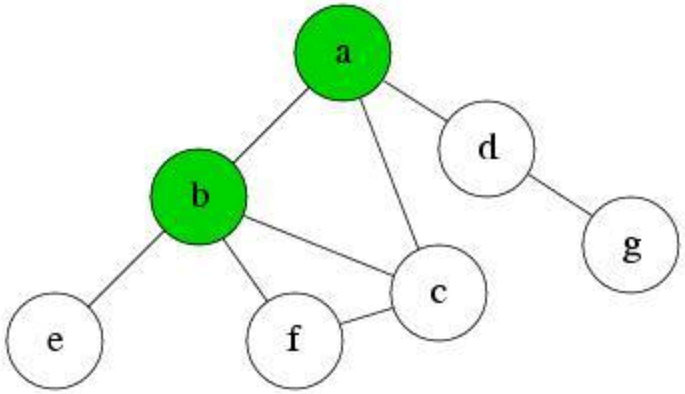


Figure 3: breadth-first search 3

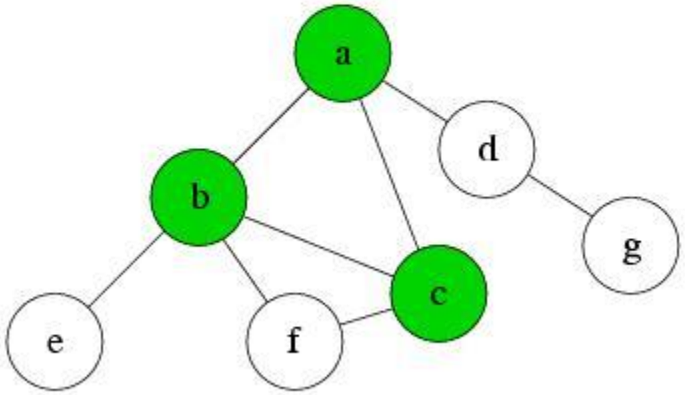


Figure 4: breadth-first search 4

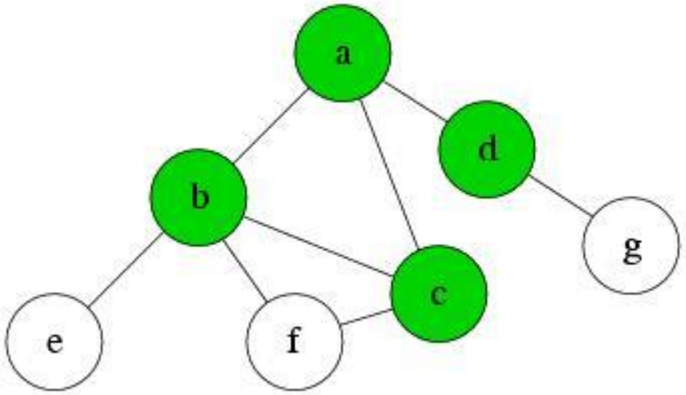


Figure 5: breadth-first search 5

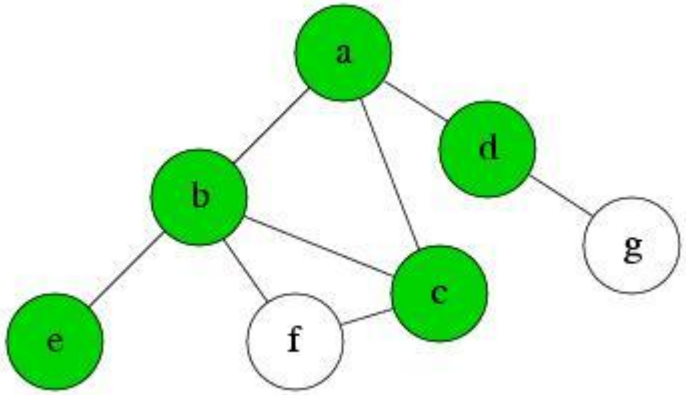


Figure 6: breadth-first search 6

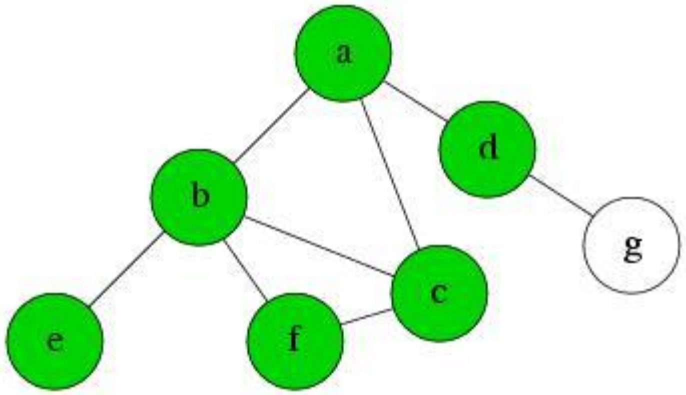


Figure 7: breadth-first search 7

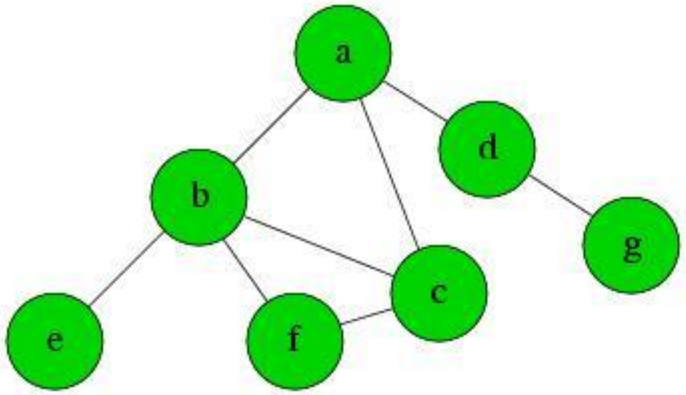


Figure 8: breadth-first search 8

Breadth-First-Search

- BFS Animation

Minimizing costs
有权图的最小生成树

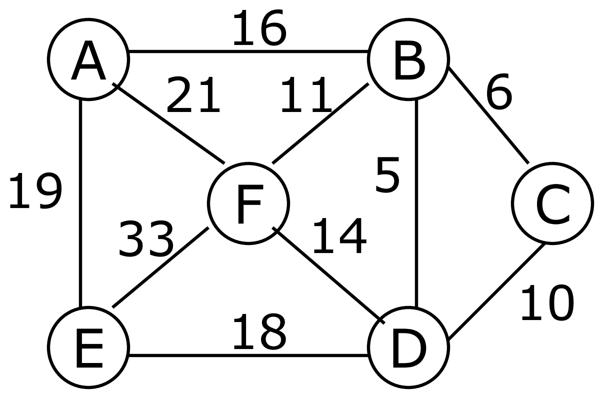
- 把一些城市或者地点都连起来，使得任意两个地方都是有路可以通行的，为了达到这个最低标准，如何设计路线，使得成本最低？

- To keep costs down, you could connect cities (or viewpoints) with a spanning tree

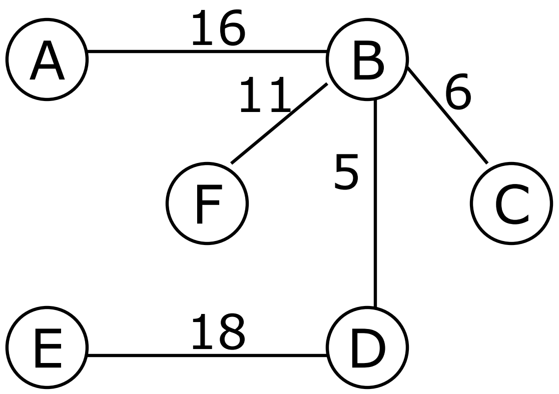
- *minimum-cost* spanning tree

Minimum-cost spanning trees
有权图的最小生成树

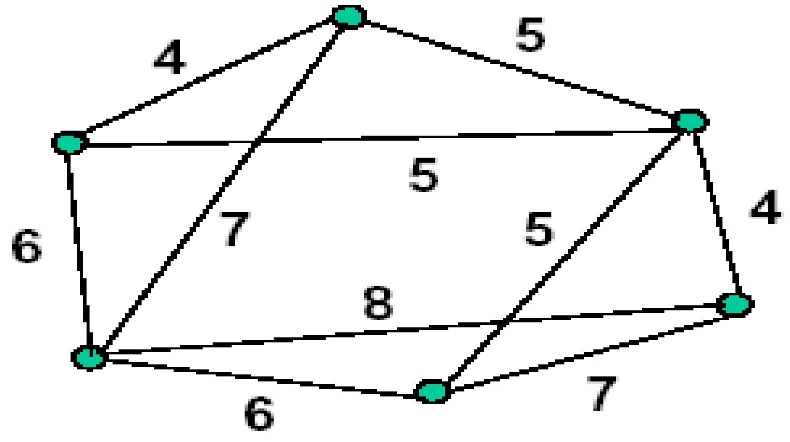
- Suppose you have a connected undirected graph with a weight (or cost) associated with each edge
- **The cost of a spanning tree：** would be the sum of the costs of its edges 所有边的权的和
- A minimum-cost spanning tree is that spanning tree that has the lowest cost



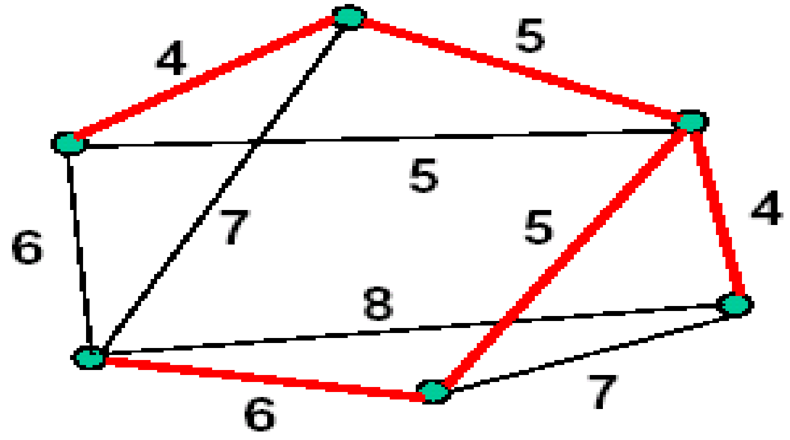
A connected, undirected graph



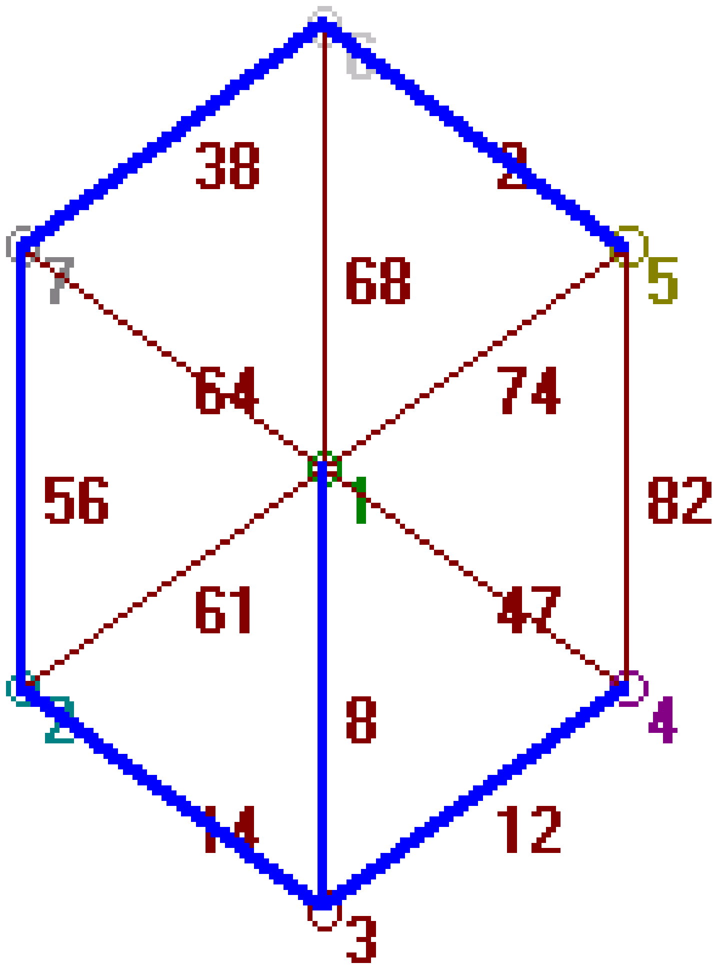
A minimum-cost spanning tree



a weighted graph G



a minimum weight spanning tree for G



A weighted graph with the minimal spanning tree in blue.

Minimum Spanning Tree (MST)

- Tree (usually undirected):
  - Connected graph with no cycles
  - |E| = |V| - 1
- Spanning tree
  - Connected subgraph that covers all vertices
  - If the original graph not tree,
    graph has several spanning trees
- Minimum spanning tree
  - Spanning tree with minimum sum of edge weights (among all spanning trees)
  - Example: build a railway system to connect N cities, with the smallest total length of the railroad

Minimum Spanning Tree Algorithms

- Basic idea:
  - Start from a vertex (or edge), and expand the tree, avoiding loops
  - Pick the minimum weight edge at each step
- Known algorithms
  - Prim: start from a vertex, expand the connected tree
  - Kruskal: start with the min weight edge,
    add min weight edges while avoiding cycles
    (build a forest of small trees, merge them)

Prim’s algorith 普林算法

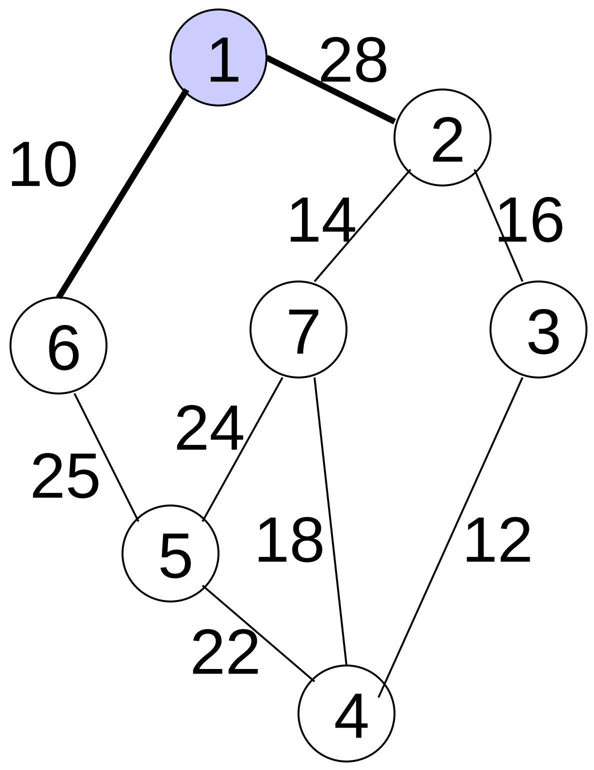
Given by Robert Prim in 1957

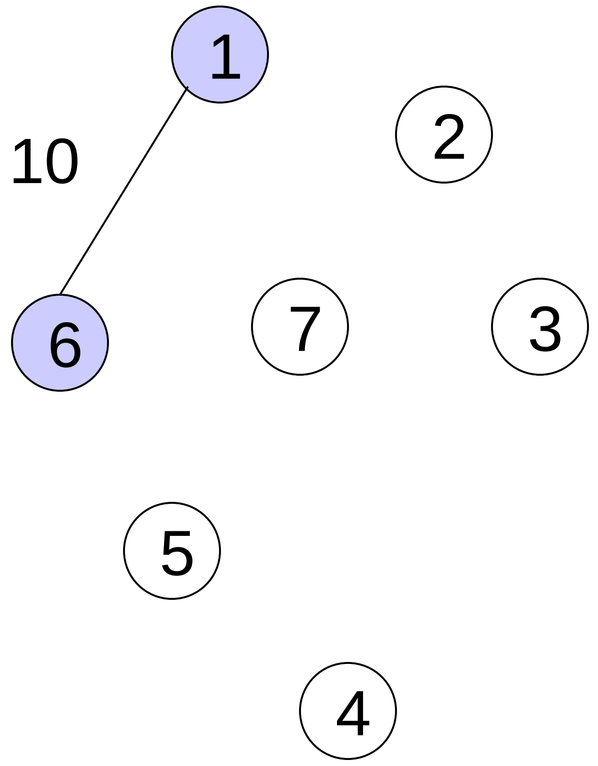
- 1. Begin by choosing any edge with smallest weight, putting it into the spanning tree 选择任意一条权最小的边，以其两个端点和这条边作为生成树的起始边和点，添加到生成树中；

- 2. Successively add to the tree edges of minimum weight that are incident to the vertex already in the tree and not forming a simple circuit with those edges already in the tree. 继续加入与已经在树中的结点关联的最小权的，不形成简单回路的边。

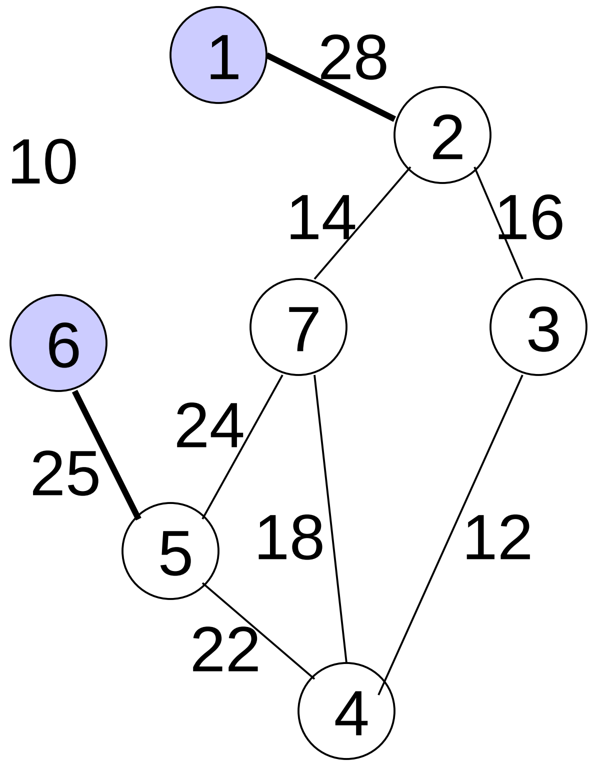
- 3. Stop when n-1 edges have been added当边数为n-1时，停止.（为什么？）

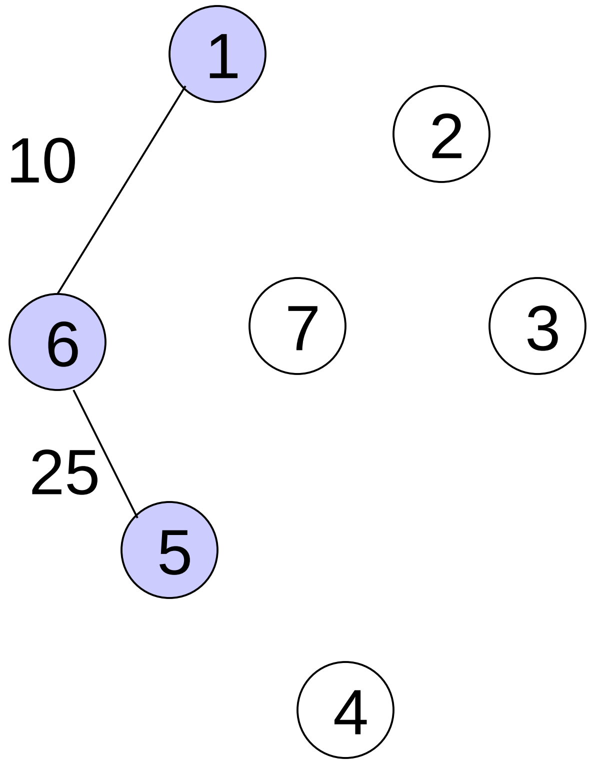
Prim’s algorithm



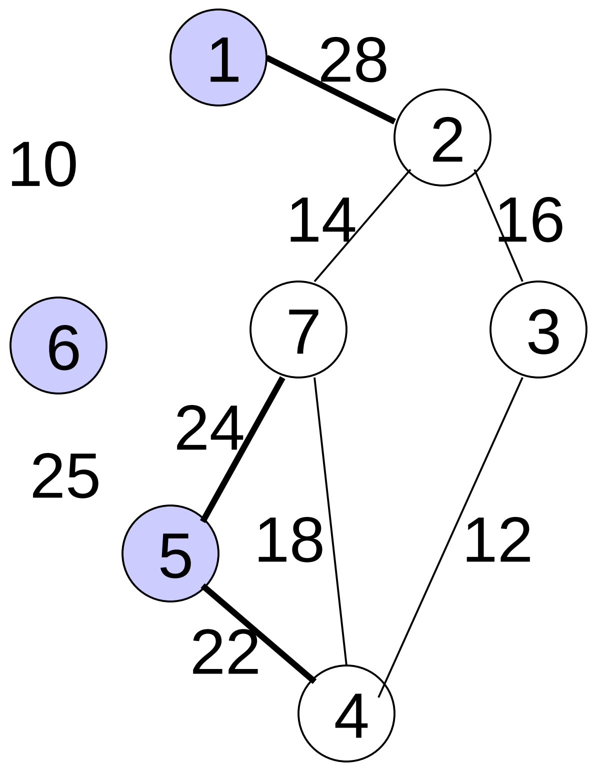


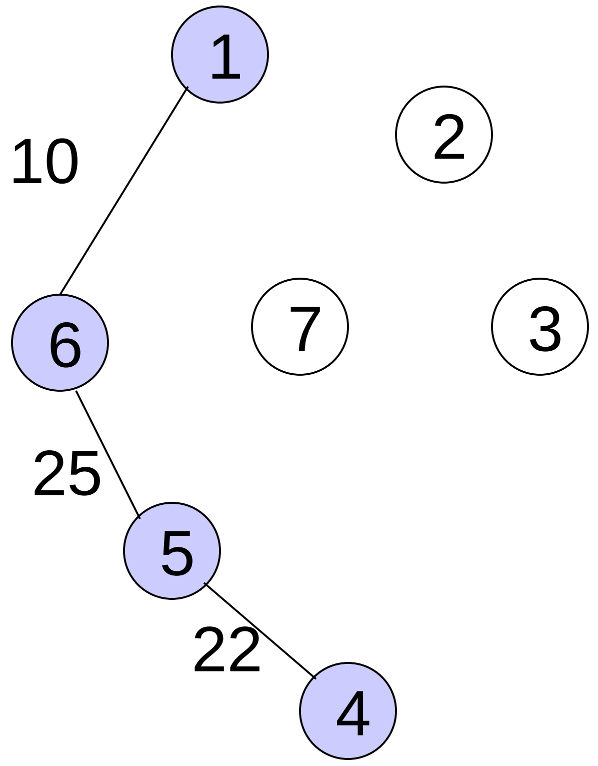
Prim's algorithm--example



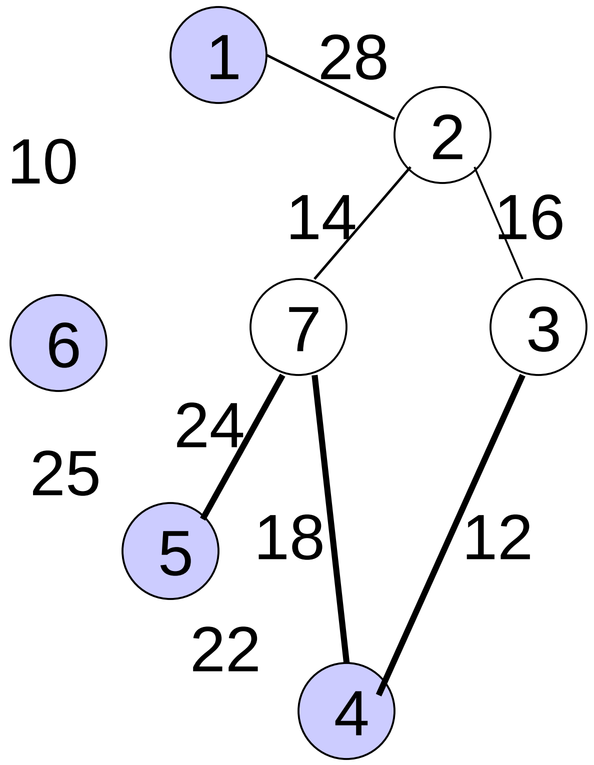


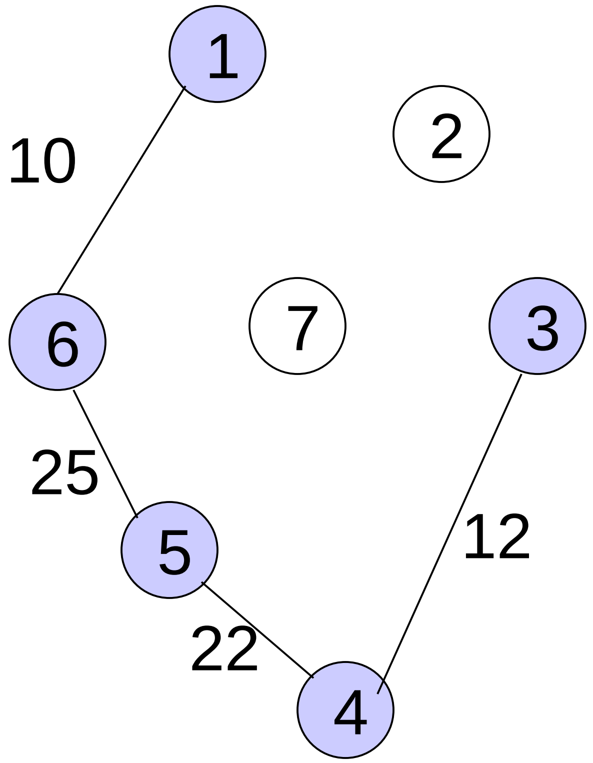
Prim's algorithm--example



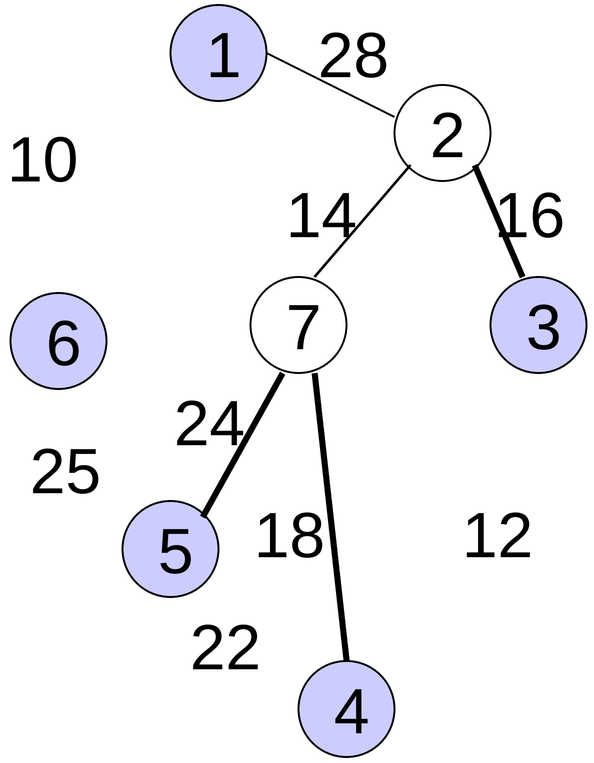


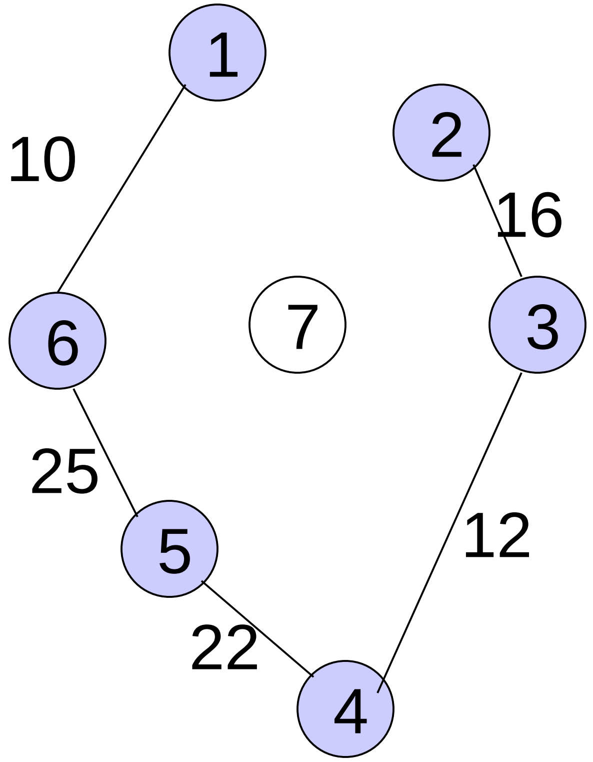
Prim's algorithm--example



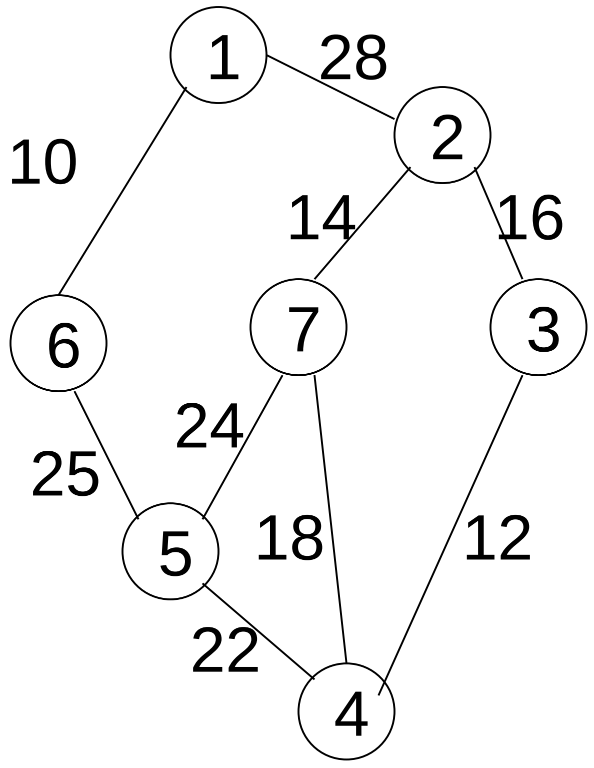


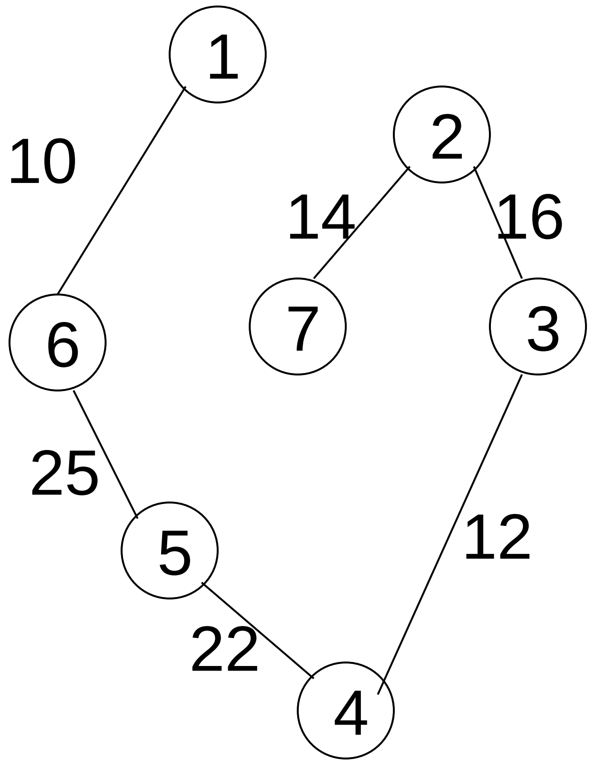
Prim’s algorithm--example





Prim's algorithm--example



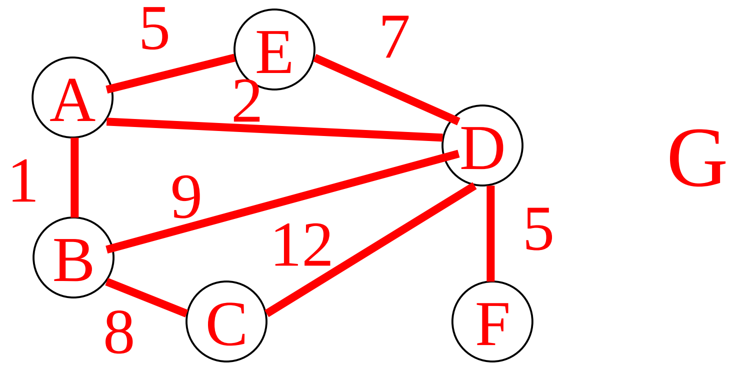


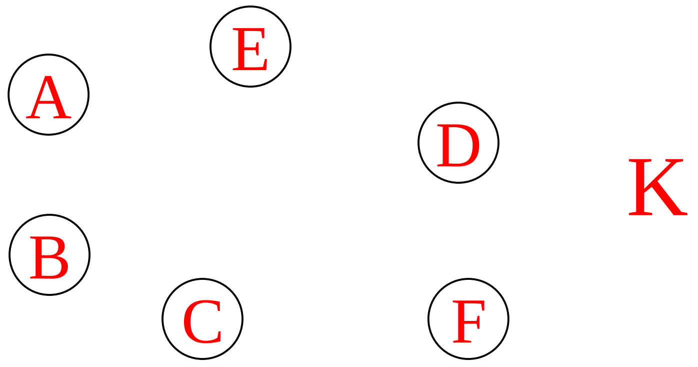
Cost = 99

Kruskal’s algorithm 算法

- 1. Pick the cheapest edge that does not create a simple cycle in the graph。首先选择图中权最小的边加入到树中；
- 2. 从尚未被选的边集中选择权最小的且不形成简单回路的边
- 3. Add the edge to the solution and remove it from the graph. 将选择好的边移至目标树中
- 4. Continue until all nodes are part of the tree. 重复上述过程，直到n-1条边为止

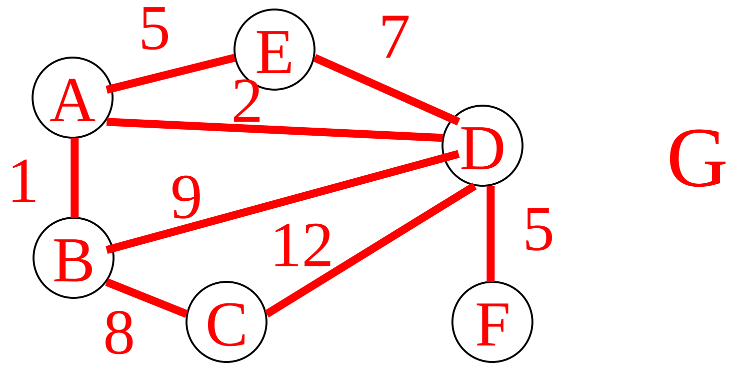
Kruskal’s Algorithm--example

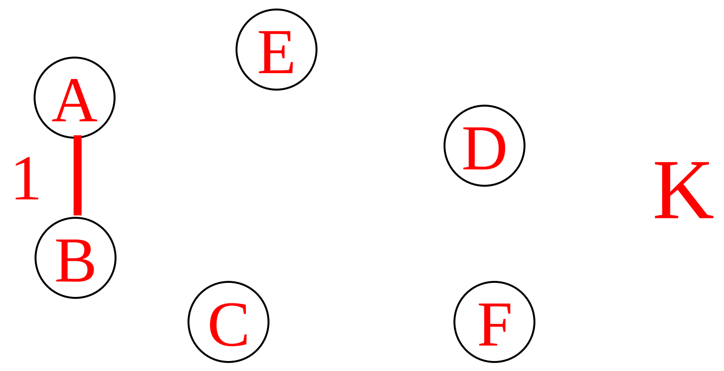




{A,B}	1
{A,D}	2
{A,E}	5
{D,F}	5
{E,D}	7
{B,C}	8
{B,D}	9
{C,D}	12

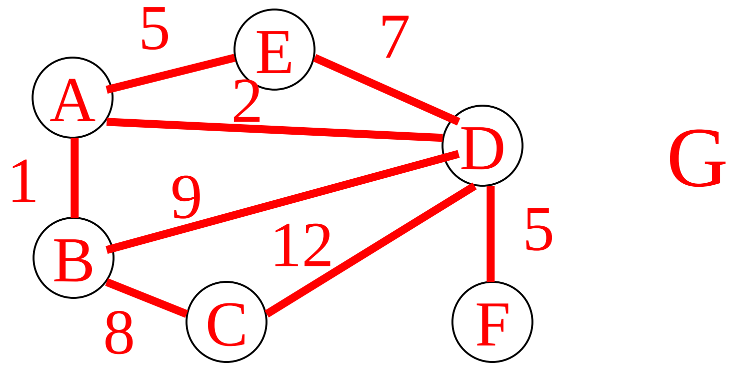
Kruskal's Algorithm--example

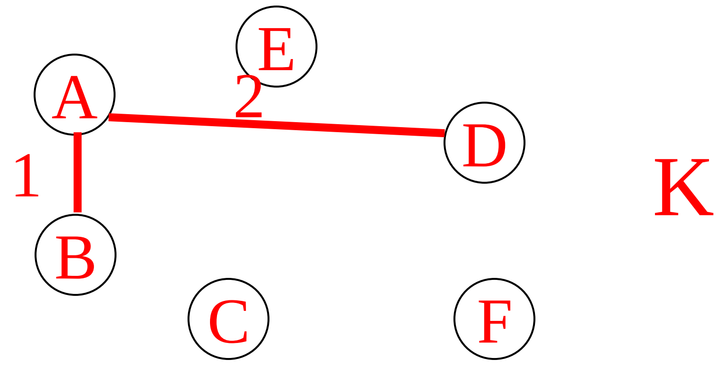




{A,B}	1
{A,D}	2
{A,E}	5
{D,F}	5
{E,D}	7
{B,C}	8
{B,D}	9
{C,D}	12

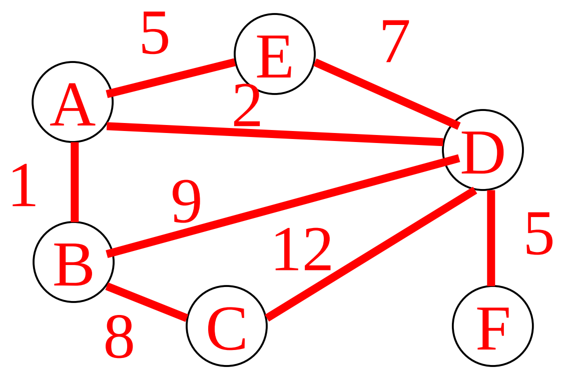
Kruskal’s Algorithm--example



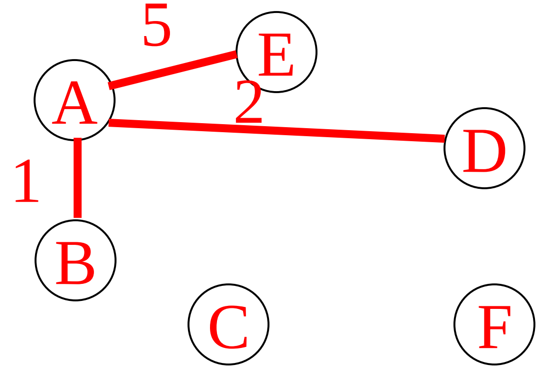


{A,B}	1
{A,D}	2
{A,E}	5
{D,F}	5
{E,D}	7
{B,C}	8
{B,D}	9
{C,D}	12

Kruskal's Algorithm--example



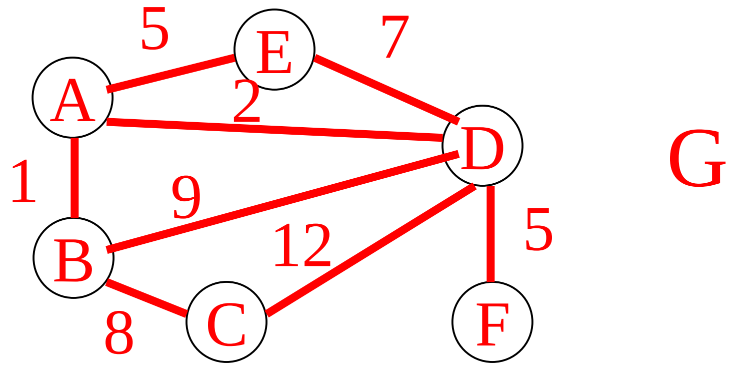
G

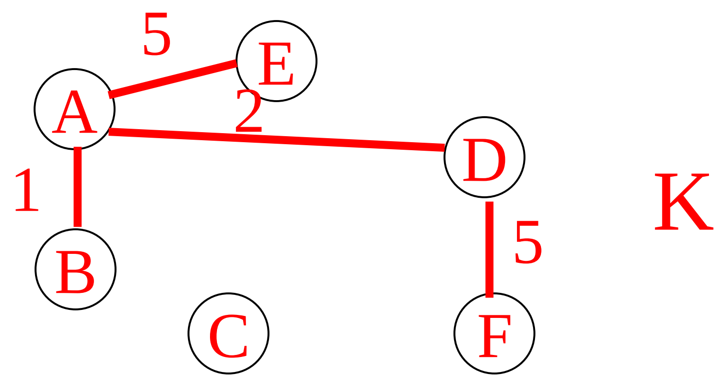


K

{A,B}	1
{A,D}	2
{A,E}	5
{D,F}	5
{E,D}	7
{B,C}	8
{B,D}	9
{C,D}	12

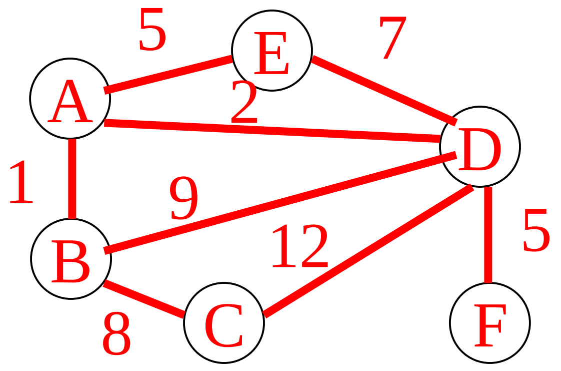
Kruskal's Algorithm--example



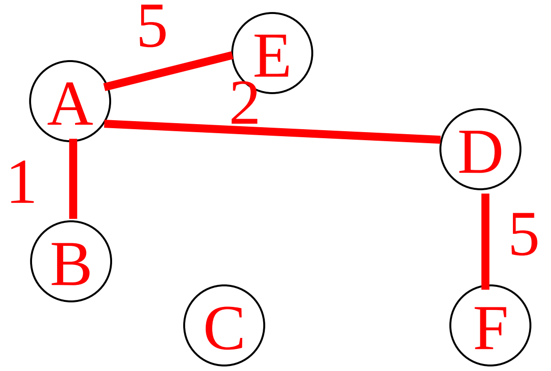


{A,B}	1
{A,D}	2
{A,E}	5
{D,F}	5
{E,D}	7
{B,C}	8
{B,D}	9
{C,D}	12

Kruskal's Algorithm--example



G

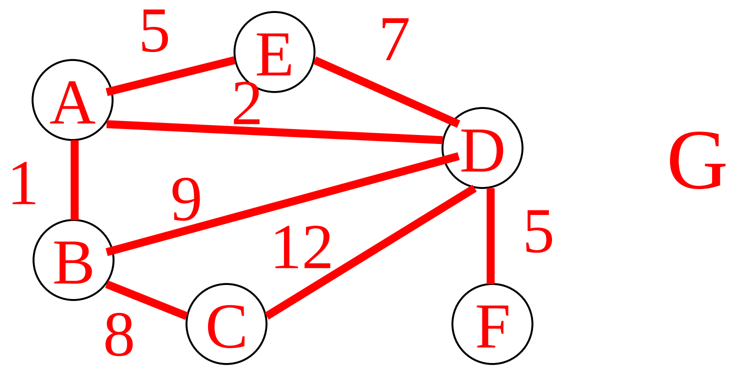


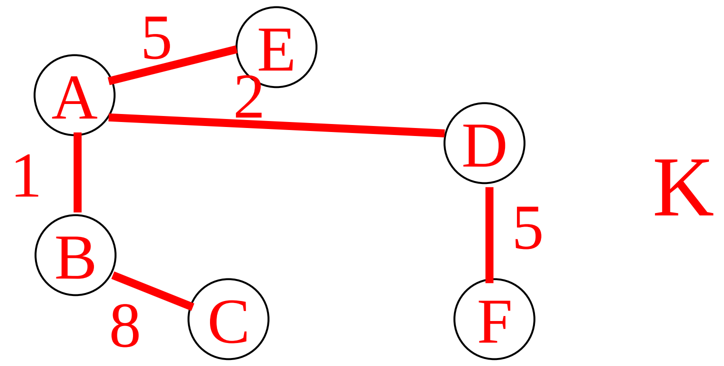
K

{E,D} already connected

{A,B}	1
{A,D}	2
{A,E}	5
{D,F}	5
{E,D}	7
{B,C}	8
{B,D}	9
{C,D}	12

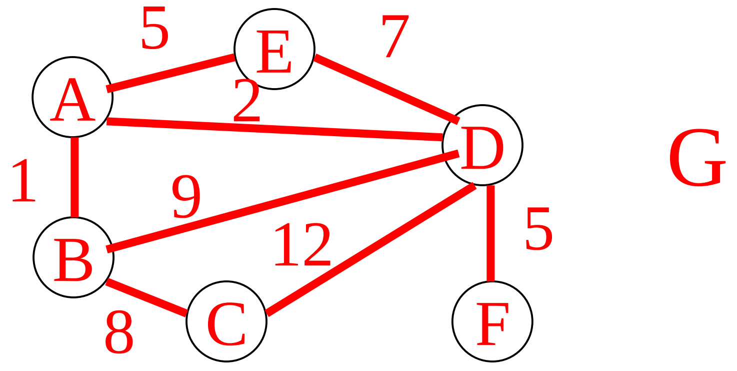
Kruskal’s Algorithm--example

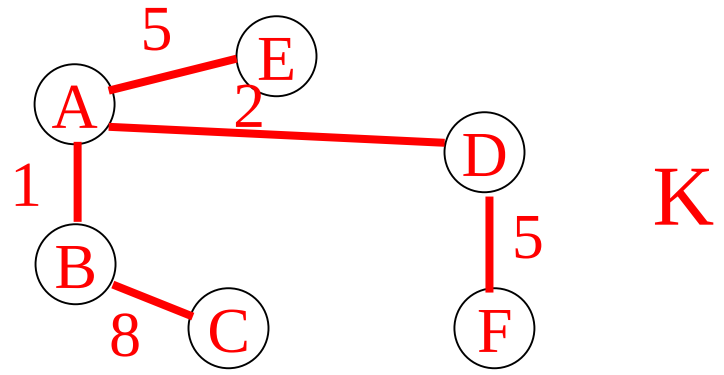




{A,B}	1
{A,D}	2
{A,E}	5
{D,F}	5
{E,D}	7
{B,C}	8
{B,D}	9
{C,D}	12

Kruskal’s Algorithm--example

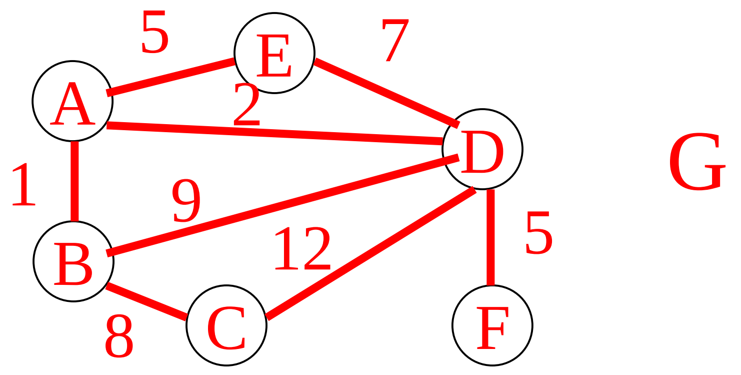


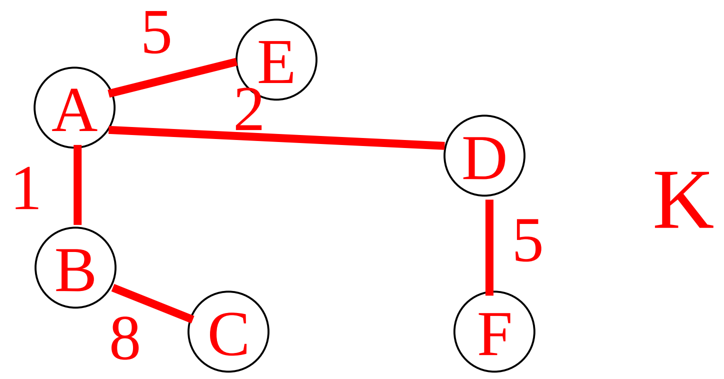


{A,B}	1
{A,D}	2
{A,E}	5
{D,F}	5
{E,D}	7
{B,C}	8
{B,D}	9
{C,D}	12

{B,D} already connected

Kruskal’s Algorithm--example





{C,D} already connected

{A,B}	1
{A,D}	2
{A,E}	5
{D,F}	5
{E,D}	7
{B,C}	8
{B,D}	9
{C,D}	12

最小生成树算法证明思路

- 讲解Kruskal算法的证明思路：
- 1. 可以确定该算法形成的就是生成树
- 2. 只需要证明该算法产生的生成树的总权与任意一棵最小生成树总权相同
- 3. 记原图为G,且有n个结点，该算法生成的树为K， 再假设任一棵最小生成树为T。设法证明K的总权等于T的总权。

因为：

假设由Kruskal 算法得到的生成树K的边集合及顺序为：{e_1, e_2, …, e_k, …e_{n-1}}，比较K与T的边。 如果K的所有边都在T中， 说明K=T则已。

否则， 这n-1条边{e_1, e_2, …, e_k, …e_{n-1}}一定有边不在树T中。

Kruskal算法证明思路续

- 从左至右，假设K的第1条不在T中的边e_k，(e_1, e_2, …, e_{k-1} 即在K中同时又在T中)， 将e_k加入到T中，必然形成一条简单回路α；而这条简单回路α也必然有边s_k在T中但不在K中。将T中的这条边s_k用e_k不来替代，得到另一个棵生成树T'， 其总权不会比T的总权小，否则T就不是最小生成树；
- 说明K中这条不在T中的边e_k的权不会比不在K中、但在T中的边s_k的权小；也即$e_k >= s_k$。
- 由于e_1, e_2, …, e_{k-1}, s_k都是T中的边， 不形成简单回路；考虑到K添加边的方法，如果$e_k > s_k$, 则与Kruskal选边算法矛盾。所以$e_k = s_k$;在T中用e_k替换s_k得到的生成树T'也是最小生成树。
- 这种替换不改变树的总权。如此重复这样的替换，最后可以将树T边都逐步转换成K的边, 而保持树的总权不变。
- 这说明K的总权与T的总权是相等的。也即K也是最小生成树，总权等于T的权

- 请同学们比较两个算法： Prim 和 Kruskal 算法

Question 思考问题

- 1. Why the subgraph got from Prim algorithm and Kruskal’s Algorithm is a spanning tree? 为什么两个算法得到的树都是是生成树？
- (这里只说是生成树，是“最小的”不作要求，自己去看）
- 2. Can you design a algorithm using removing edges from connected graph to get minimum spanning tree? 能否设计一个算法，移除边的算法，设法得到最小生成树？

- *注意事项：在移除边时该注意什么？*
- 算法的实现将在数据结构里学习

Exercises

- 7.5节 T2 ， T6， T8

例题习题选讲

- 例题1： 任意一棵结点数大于1的树都是偶图

- 例题2：G是简单连通图，G是树当且仅当G的每一条边都是割边。

- 例题3：若G是一棵树，而且其所有结点中最大度数大于等于k, 则G至少有K片树叶 (无向树中度为1的结点称为叶结点；有向树中出度为0的结点称为叶结点）

- 例题4：任何一棵树是否是平面图？为什么？
- 例题5：如果在一棵树（结点数大于2）添加一条边，使其成为一个简单图。那么这个图是否是平面图？

- 例题6： G是结点数大于2的连通图，则G至少存在两个结点，把这两个结点删除后，仍然连通。
- 例题7：若图为欧拉图，则图必然没有割边。

例题习题选讲

- 例题8：证明: 在多于两个人的人群中, 至少有两个人的朋友数是相同的. (北京中科院计算所某年的考研题目)

- 证明思路: 将每个人看成一个结点, 一个人与另外一人有朋友关系, 则两个结点是邻接的; 在这样一个图中, 考察边数, 总度数, 及完全图的度数等之间的关系, 再用反正法.

例题习题选讲

- 例题9： n（n>3)支球队进行比赛，比赛进行了n+1场，则必然存在一支球队它至少参加了3场比赛。