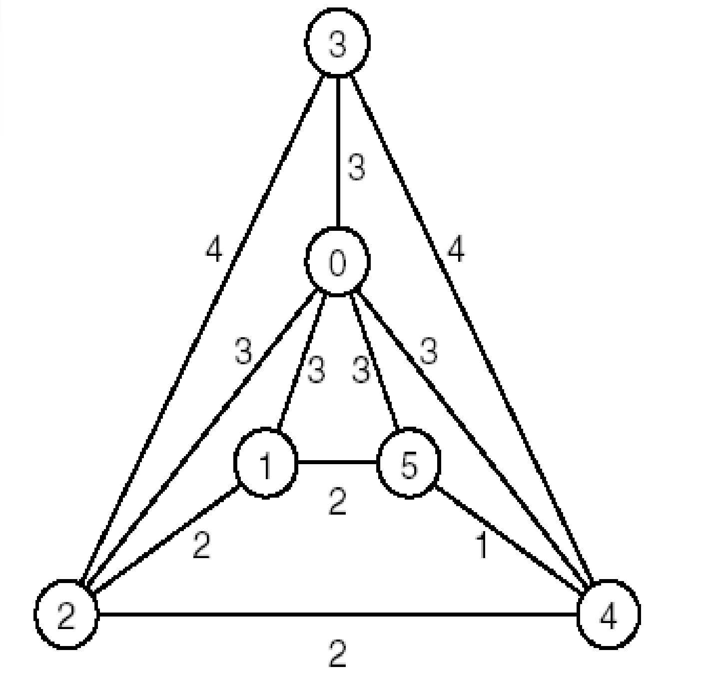
Minimal Spanning Trees
最小生成树

最小生成树

❖ The Problem

Shortest paths from source 0 to all vertices in a network(网):

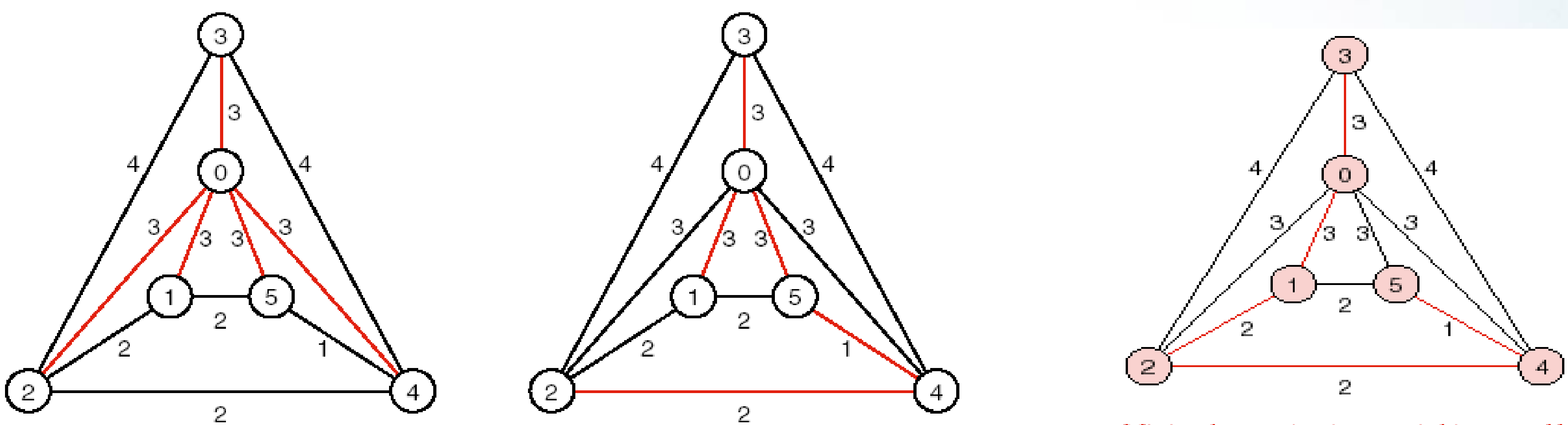


最小生成树

- The Problem
  - If the original network is based on a connected graph *G*, then the shortest paths from a particular source vertex to all other vertices in *G* form a tree that links up all the vertices of *G*.
  - A (connected) tree that is build up out of all the vertices and some of the edges of *G* is called a *spanning tree* of *G*.

最小生成树

- DEFINITION A *minimal spanning tree* of a connected network is a spanning tree such that the sum of the weights of its edges is as small as possible.



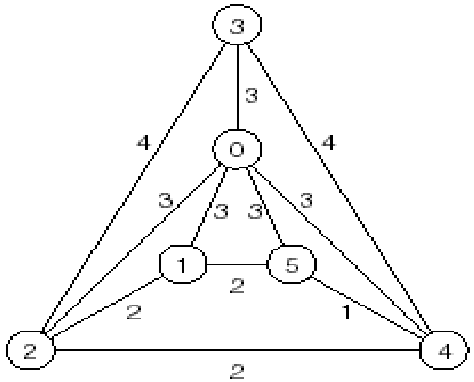
Weight sum of tree = 15
(a)

Weight sum of tree = 12
(b)

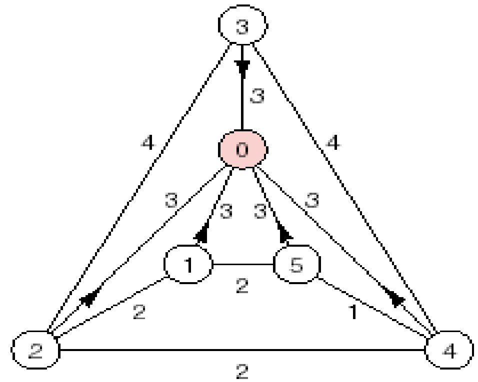
Minimal spanning tree, weight sum = 11
(g)

最小生成树

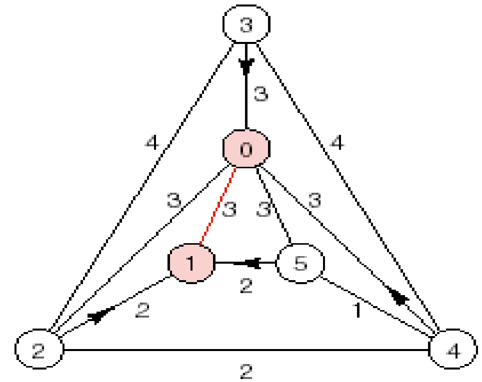
- 最小生成树的普里姆算法：
  - 算法思想：
    - 设连通网N=(V,{E})，生成树T=(U,{TE})
    - （1）初始化：U={V1},{TE}=Ø，即选取初始结点；
    - （2）建立过程：选取权值最小的边（Vi,Vj）并入{TE}，该边必须满足的条件是：$\boldsymbol{Vi} \in \boldsymbol{U}$且$\boldsymbol{Vj} \in (\boldsymbol{V}-\boldsymbol{U})$，再将$\boldsymbol{Vj}$加入U中；
    - （3）重复上面两步，直到V==U为止。



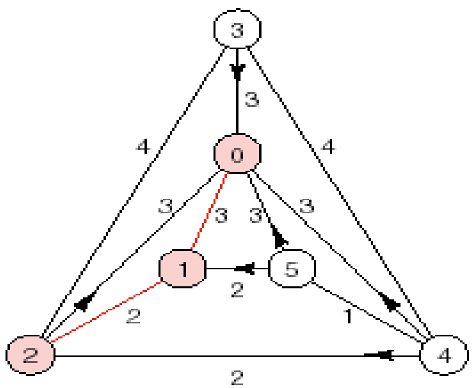
(a)



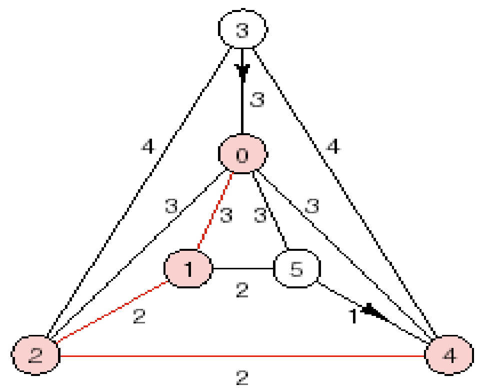
(b)



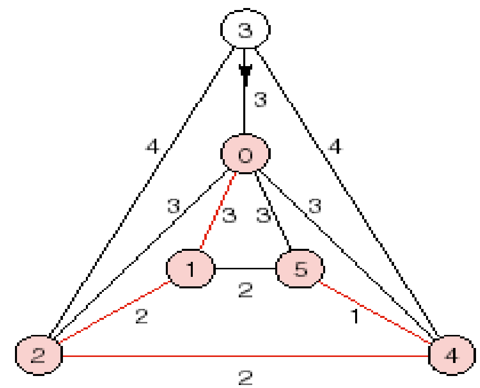
(c)

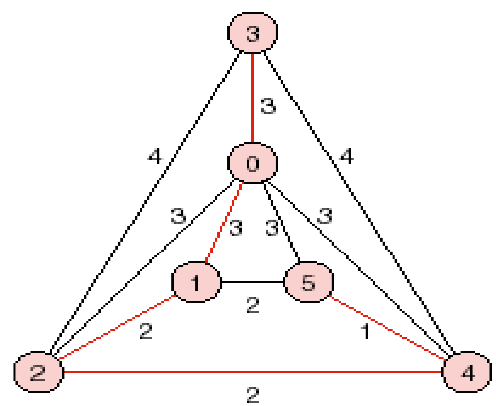


(d)



(e)

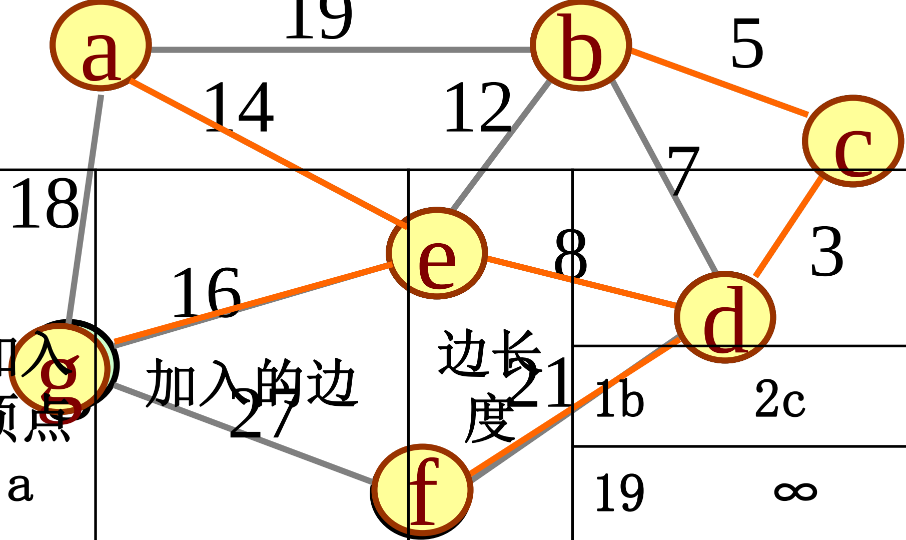




Minimal spanning tree, weight sum = 11
(g)

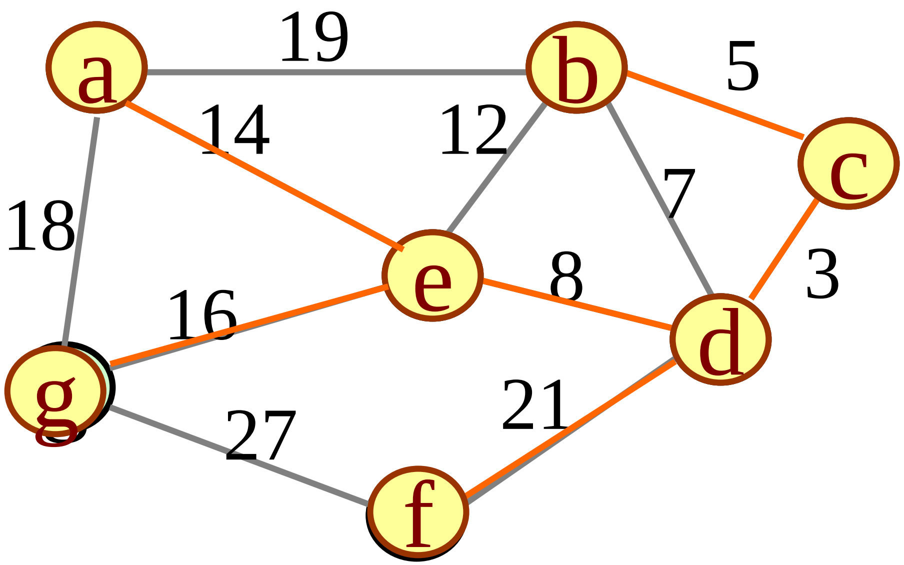
最小生成树

- ❖ 最小生成树的普里姆算法：
  - ➢ 算法核心：求权值最小的边$(\boldsymbol{Vi}, \boldsymbol{Vj})$,其中$\boldsymbol{Vi} \in \boldsymbol{U}$, $\boldsymbol{Vj} \in (\boldsymbol{V} - \boldsymbol{U})$
    - 当U={V0}，即只含初始结点时，最小权值为：与Vi相邻的边中权值最小的边——由邻接矩阵可以直接得到
    - 当U={V0,V1……，Vi}时，即含有结点数超过1个时，最小权值为：与V0,V1,……，Vi所有结点相邻的所有边中最小值—即与V0,V1……Vi-1的相邻边中最小值与Vi相邻的边中最小值中较小的，然后递推得到。



迭代次数	加入顶点	加入的边	边长度	Distance[i]					
				1b	2c	3d	4e	5f	6g
	a			19 *a*	∞ *a*	∞ *a*	*14* *a*	∞ *a*	18 *a*
1	**e**	**(a,e)**	**14**	12 *e*	∞ *a*	8 *e*	*14* *a*	∞ *a*	16 *e*
2	**d**	**(e,d)**	**8**	7 *d*	3 *d*	8 *e*	*14* *a*	21 *d*	16 *e*
3	**c**	**(d,c)**	3	5 *c*	3 *d*	8 *e*	*14* *a*	21 *d*	16 *e*
4	**b**	**(c,b)**	5	5 *c*	3 *d*	8 *e*	*14* *a*	21 *d*	16 *e*
5	**g**	**(e,g)**	16	5 *c*	3 *d*	8 *e*	*14* *a*	21 *d*	16 *e*
6	**f**	**(d,f)**	21						

例如:



	0a	1b	2c	3d	4e	5f	6g
neighbour		c	d	e	a	d	e
distance		5	3	8	14	21	16

最小生成树

- implementation of Prim‘s Algorithm

```
template <class Weight, int graph_size>
class Network: public Digraph<Weight, graph_size> {
public:
Network( );
void read( ); //overridden method to enter a Network
void make_empty(int size = 0);
void add_edge(Vertex v, Vertex w, Weight x);
void minimal_spanning(Vertex source,
Network<Weight, graph_size> &tree) const;
};
```

最小生成树

- implementation of Prim Algorithm
  - read is overridden to make sure that the weight of any edge (*v,w*) matches that of the edge *(w, v):* In this way, we preserve our data structure from the potential corruption of undirected edges.
  - make_empty(**int** size) creates a Network with size vertices and no edges.
  - add_edge adds an edge with a specified weight to a Network.
  - As for the shortest-path algorithm, we assume that the **class** Weight has comparison operators.
  - We expect clients to declare a largest possible Weight value called infinity.

最小生成树

- implementation of Prim‘s Algorithm

```
template <class Weight, int graph_size>
void Network < Weight, graph_size > :: minimal_spanning(Vertex
        source,Network<Weight, graph_size> &tree) const
/* Post: The Network tree contains a minimal spanning tree for the
        connected component(连通分量)of the original Network that
        contains vertex source . */
{ tree.make_empty(count);
bool component[graph_size]; // Vertices in set X
Weight distance[graph_size]; // Distances of vertices adjacent to X
```

养天地正气 法古今完人

最小生成树

- implementation of Prim's Algorithm

```
Vertex neighbor[graph_size]; // Nearest neighbor in set X
Vertex w;
for (w = 0; w < count; w++) {
        component[w] = false;
        distance[w] = adjacency[source][w];
        neighbor[w] = source;
 }
component[source] = true; // source alone is in the set X.
for (int i = 1; i < count; i++) {
Vertex v; //Add one vertex v to X on each pass.
Weight min = infinity;
```

养天地正气 法古今完人

最小生成树

```
for (w = 0; w < count; w++)
if (!component[w] && distance[w] < min) {
        v = w;
        min = distance[w];
}
if (min < infinity) {
        component[v] = true;
        tree . add_edge (v, neighbor[v], distance[v]);
        for (w = 0; w < count; w++)
                if (!component[w] && adjacency[v][w] < distance[w]) {
                        distance[w] = adjacency[v][w];
                        neighbor[w] = v;
        }
}else break; // finished a component in disconnected graph
} }
```

最小生成树

- 最小生成树的克鲁斯卡尔算法：
  - 基本思想：
    - ❿ 首先将边去除，形成n个结点构成的n个连通分量
    - ❿ 选择vi,vj，满足vi和vj属于不同的连通分量，且连接vi,vj的边/弧权值最小；
    - ❿ 即，将连通分量的个数降低一个
    - ❿ 如此循环直到整个构成一个连通图为止（即循环n-1次，生成n-1条边）

最小生成树

◆ 最小生成树的克鲁斯卡尔算法：

➢ 示例：