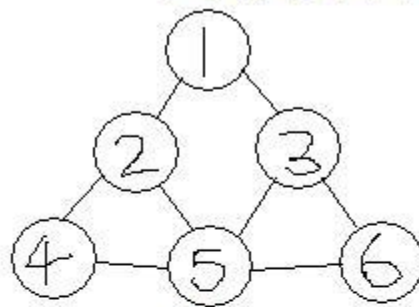




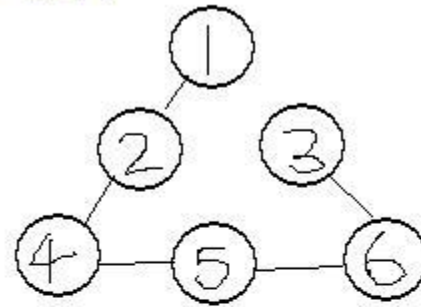
最小生成树

生成树是一个连通图 G 的一个极小连通子图。包含 G 的所有 n 个顶点，但只有 $n-1$ 条边，并且是连通的。

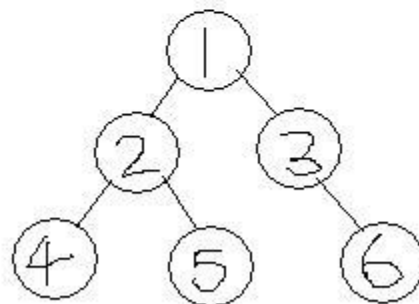
无向图及其不同生成树



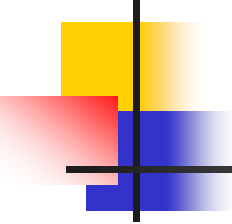
无向图



生成树1



生成树2

- 
-
- 当生成树中所包含的边的权值和最小，我们称之为最小生成树。

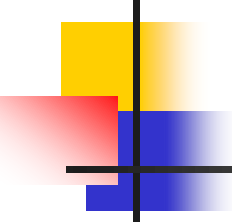


最小生成树性质

最小生成树的边数必然是顶点数减一， $|E| = |V| - 1$ 。

最小生成树不可以有循环。

最小生成树不必是唯一的。

- 
-
- 本节所介绍的2种最小生成树算法，都是基于贪心策略。
 - I.Kruskal算法
 - II.Prim算法

应用举例——最小生成树

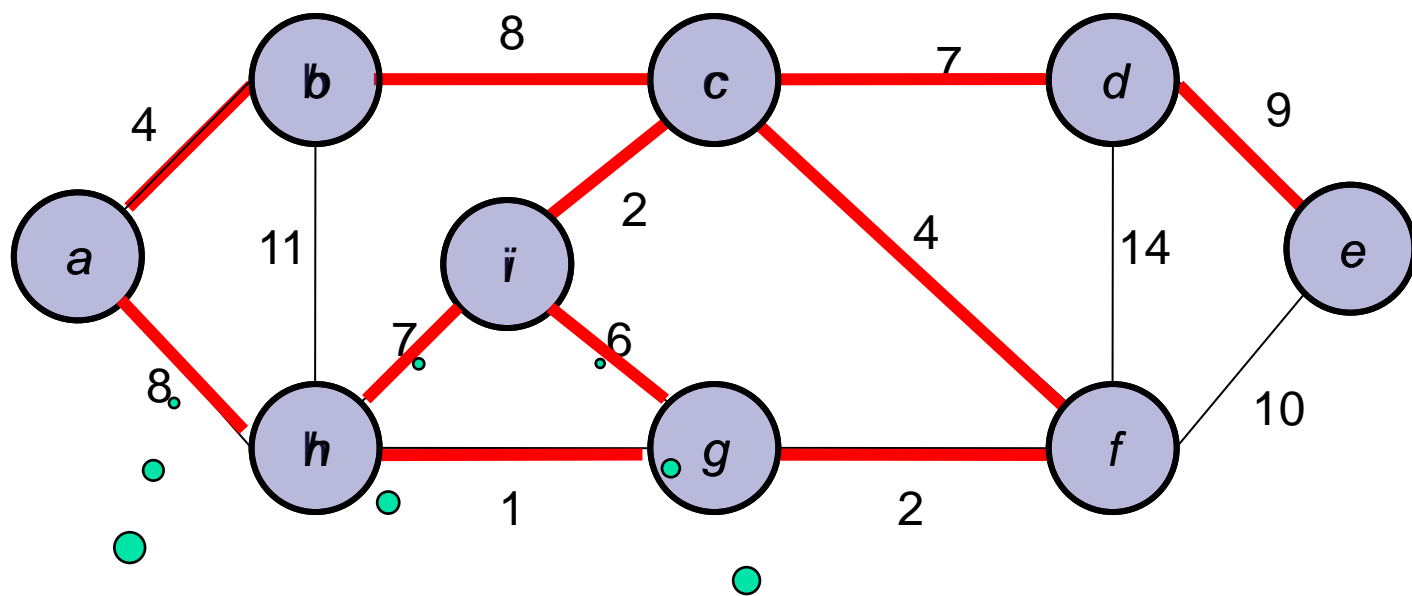


克鲁斯卡尔 (Kruskal) 算法

基本思想：

假设 $WN=(V, E)$ 是一个含有 n 个顶点的连通网，则按照克鲁斯卡尔算法构造最小生成树的过程为：先构造一个只含 n 个顶点，而边集为空的子图，若将该子图中各个顶点看成是各棵树上的根结点，则它是一个含有 n 棵树的一个森林。之后，从网的边集 E 中选取一条权值最小的边，若该条边的两个顶点分属不同的树，则将其加入子图，也就是说，将这两个顶点分别所在的两棵树合成一棵树；反之，若该条边的两个顶点已落在同一棵树上，则不可取，而应该取下一条权值最小的边再试之。依次类推，直至森林中只有一棵树，也即子图中含有 $n-1$ 条边为止。

Kruskal 算法过程



两点属于
同一棵树

两点属于
同一棵树

两点属于同
一棵树

当森林只剩下一
棵树时，算法结
束

```
#include<iostream>
#include<algorithm>
using namespace std;
struct Edge
{
    int a,b;//边的两个顶点的编号
    int d;//边的权值
}edge[11111];
int n,m,s;//n为无向图的顶点个数,m为边的条数,s用来存放最小生成树的总权值
int root[111];//存储父节点
bool cmp(Edge a,Edge b)
{
    return a.d<b.d;
}
int find(int a)//寻找父节点
{
    if(root[a]==a) return a;
    return root[a]=find(root[a]);
}
```



```
bool Union(int a,int b,int d)//合并
```

```
{
```

```
    if(a==b) return 0;//a==b说明边的两个顶点一属于同一颗树，  
    所以不需要合并，直接返回
```

```
    root[a]=b;//将a的父节点更新为b，从而将树a，b合并成一棵树
```

```
    s+=d;//将边的权值加到s当中
```

```
    return 1;
```

```
}
```

```
void kruskal()
```

```
{
```

```
    for(int i=0;i<n;i++)//初始化，将各顶点的父节点标记为本身
```

```
        root[i]=i;
```

```
    sort(edge,edge+m,cmp);//将所有边 根据边的权值 从小到大排列
```

```
    s=0;
```

```
    for(i=0;i<m;i++)//遍历所有的边
```

```
{
```

```
        if(Union(find(edge[i].a),find(edge[i].b),edge[i].d))
```

```
            n--;//当合并成功，森林的树就少一棵
```

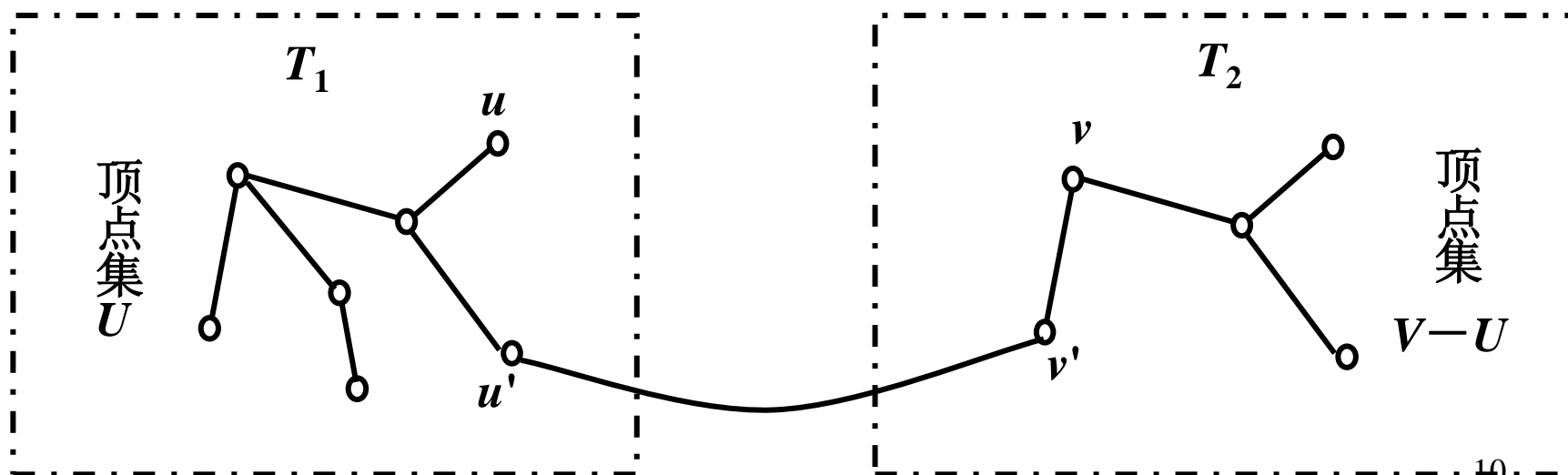
```
    }//当遍历完所有的边时,如果n!=1,说明该无向图不是连通图,不存在最小生成树
```

```
}
```

应用举例——最小生成树

MST性质

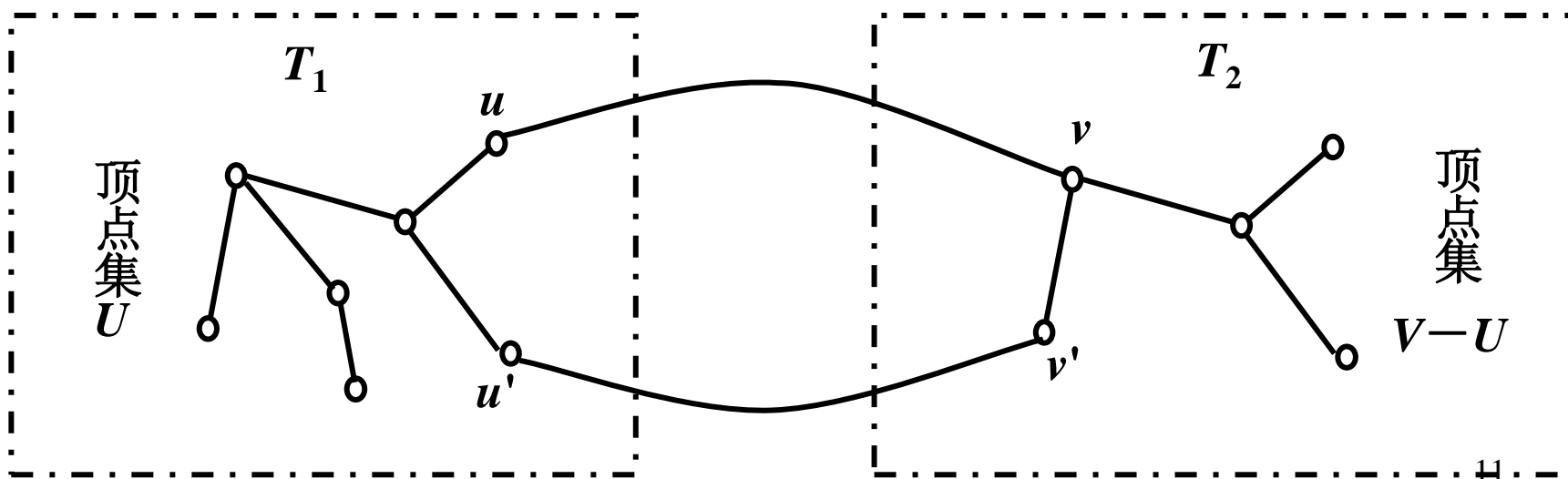
假设 $G=(V, E)$ 是一个无向连通网， U 是顶点集 V 的一个非空子集。若 (u, v) 是一条具有最小权值的边，其中 $u \in U$ ， $v \in V-U$ ，则必存在一棵包含边 (u, v) 的最小生成树。



应用举例——最小生成树

MST性质

假设 $G=(V, E)$ 是一个无向连通网， U 是顶点集 V 的一个非空子集。若 (u, v) 是一条具有最小权值的边，其中 $u \in U$ ， $v \in V-U$ ，则必存在一棵包含边 (u, v) 的最小生成树。



应用举例——最小生成树

普里姆 (Prim) 算法

基本思想： 设 $G=(V, E)$ 是具有 n 个顶点的连通网， $T=(U, TE)$ 是 G 的最小生成树， T 的**初始状态**为 $U=\{u_0\}$ ($u_0 \in V$)， $TE=\{ \}$ ，重复执行下述操作：在所有 $u \in U$ ， $v \in V-U$ 的边中找一条代价最小的边 (u, v) 并入集合 TE ，同时 v 并入 U ，直至 $U=V$ 。

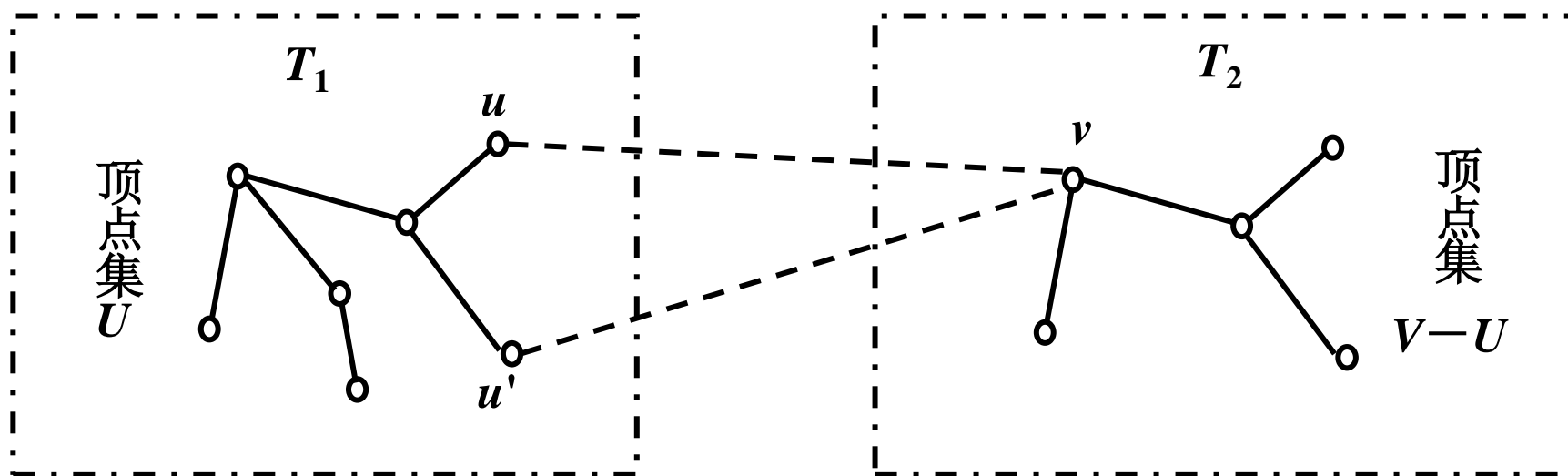
1. 初始化： $U=\{u_0\}$ ($u_0 \in V$)， $TE=\{ \}$;
2. 循环直到 $U=V$ 为止
 - 2.1 在所有 $u \in U$ ， $v \in V-U$ 的边中找一条代价最小的边 (u, v) ;
 - 2.2 $TE=TE+\{(u, v)\}$;
 - 2.3 $U=U+\{v\}$;

应用举例——最小生成树

普里姆 (Prim) 算法

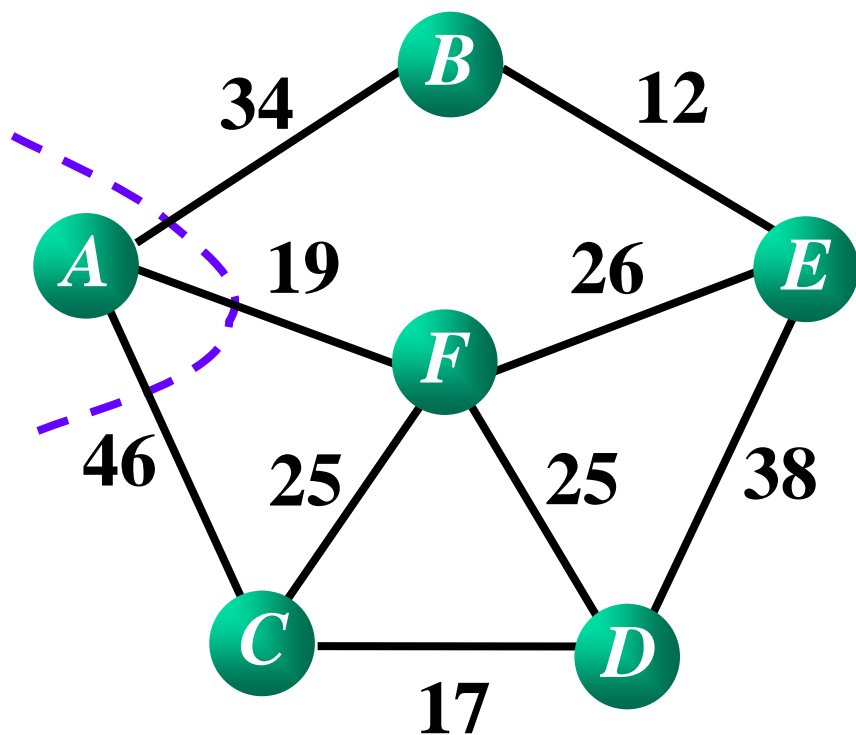
关键:是如何找到连接 U 和 $V-U$ 的最短边。

利用MST性质, 可以用下述方法构造候选最短边集:
对 $V-U$ 中的每个顶点, 分别求其到 U 的最短边。



应用举例——最小生成树

Prim算法



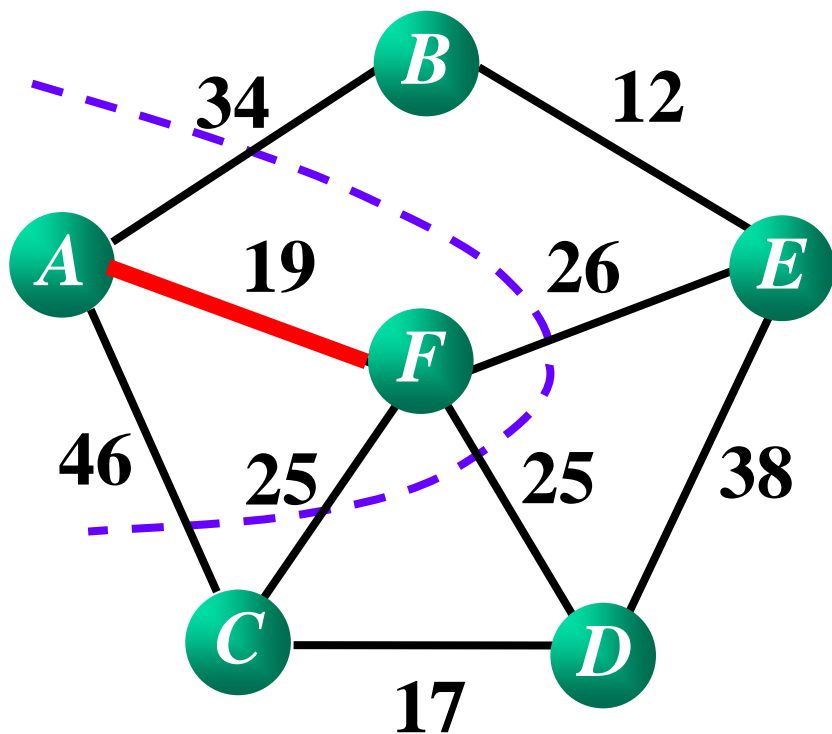
$U=\{A\}$

$V-U=\{B, C, D, E, F\}$

$\text{cost}=\{(A, B)34,$
 $(A, C)46,$
 $(A, D)\infty,$
 $(A, E)\infty,$
 $\underline{(A, F)19}\}$

应用举例——最小生成树

Prim算法



$U = \{A, \mathbf{F}\}$

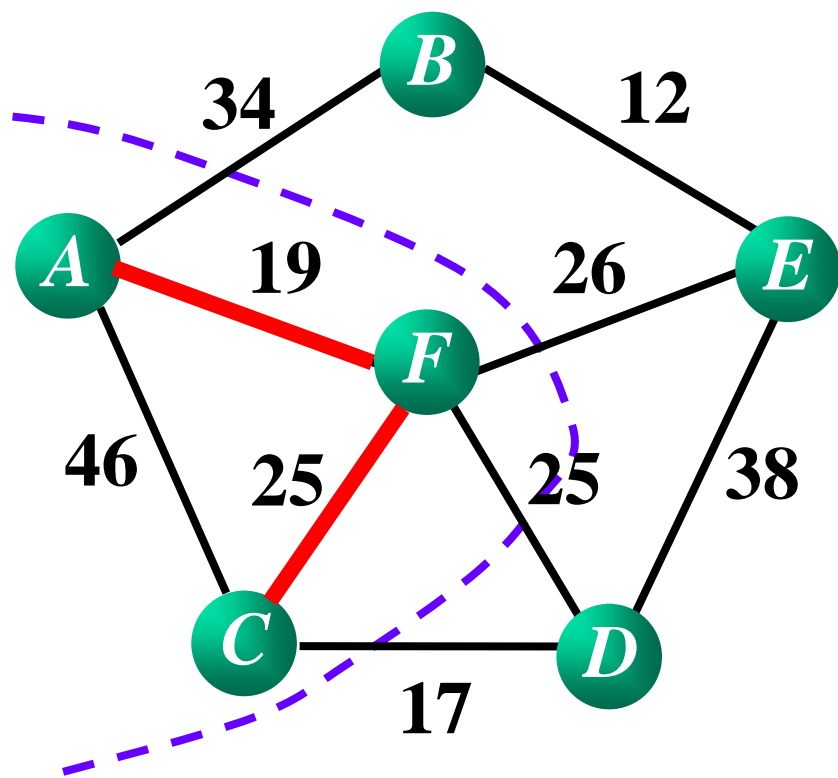
$V - U = \{B, C, D, E\}$

$\text{cost} = \{(A, B)34,$
 $(F, C)25,$
 $(F, D)25,$
 $(F, E)26\}$

$\text{cost}' = \{(A, B)34, (A, C)46,$
 $(A, D)\infty, (A, E)\infty, \mathbf{(A, F)19}\}$

应用举例——最小生成树

Prim算法



$U = \{A, F, \text{C}\}$

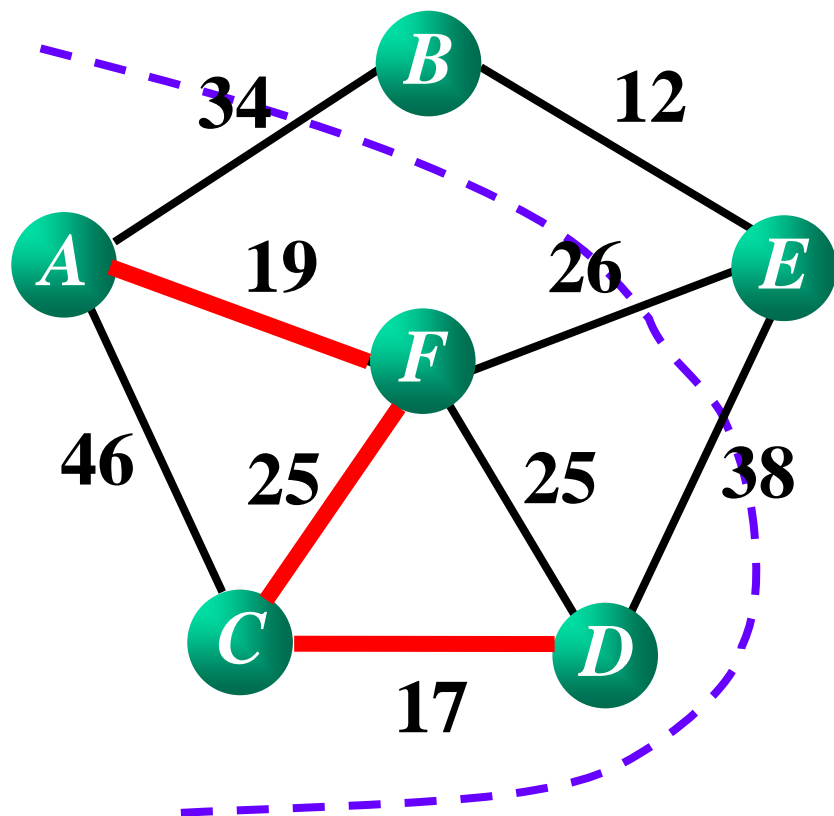
$V - U = \{B, D, E\}$

$\text{cost} = \{(A, B)34, \underline{(C, D)17}, (F, E)26\}$

$\text{cost}' = \{(A, B)34, \cancel{(F, C)25}, \cancel{(F, D)25}, (F, E)26\}$

应用举例——最小生成树

Prim算法



$U = \{A, F, C, \mathbf{D}\}$

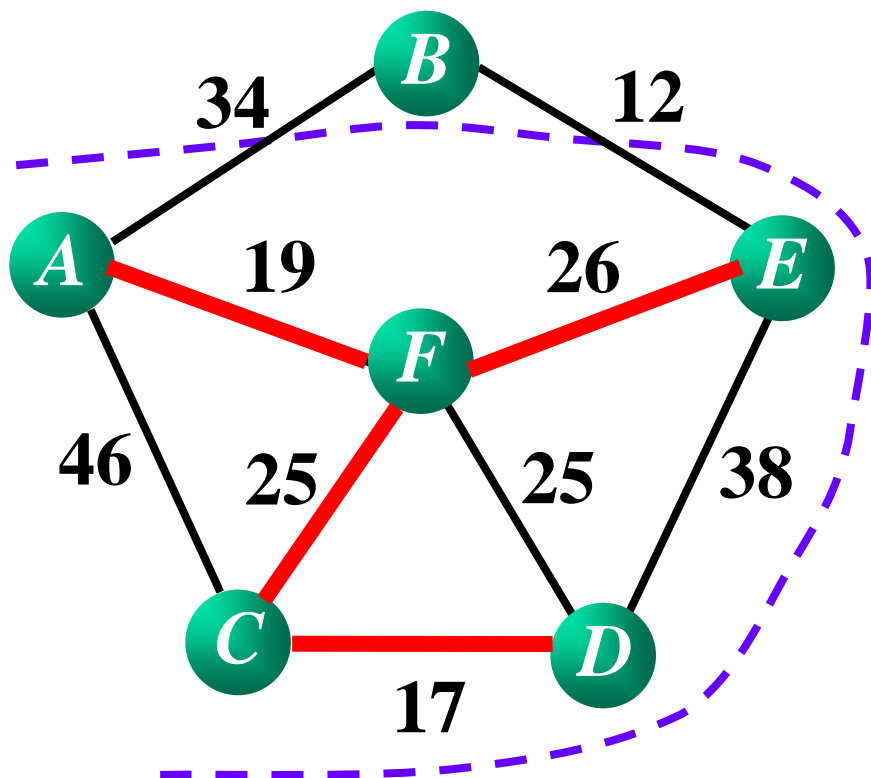
$V - U = \{B, E\}$

$\text{cost} = \{(A, B)34, \mathbf{(F, E)26}\}$

$\text{cost}' = \{(A, B)34, \mathbf{(C, D)17}, (F, E)26\}$

应用举例——最小生成树

Prim算法



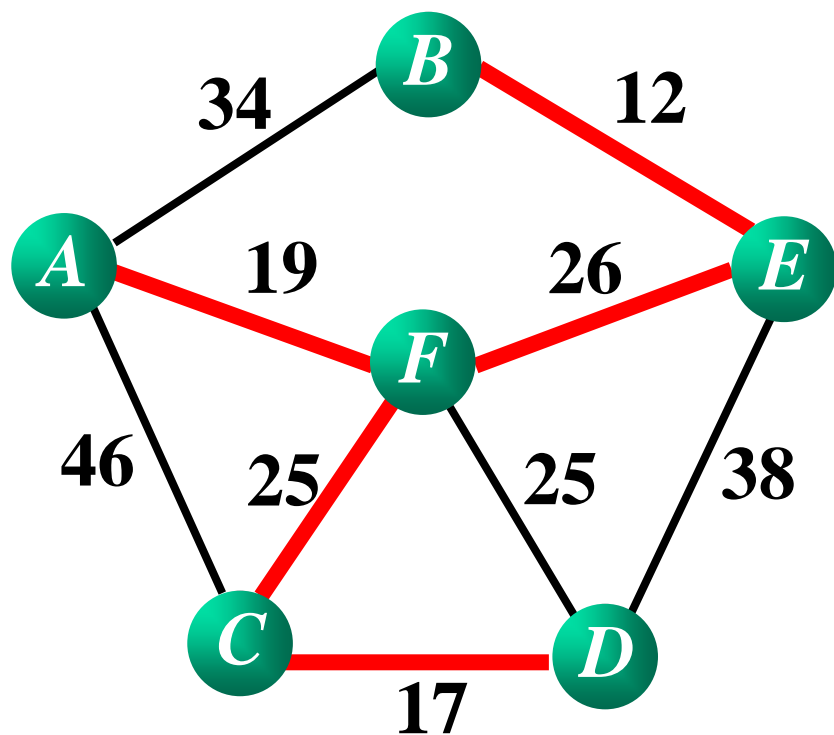
$U = \{A, F, C, D, \mathbf{E}\}$

$V - U = \{B\}$

$\text{cost} = \{(\mathbf{E}, \mathbf{B})\mathbf{12}\}$

应用举例——最小生成树

Prim算法



$U = \{A, F, C, D, E, \mathbf{B}\}$

$V - U = \{ \}$

```
#include <iostream>
#include <vector>
using namespace std;
```

//Prim算法实现

```
void prim_test()
```

```
{
```

```
    int n;
```

```
    cin >> n;
```

```
    vector<vector<int> > A(n, vector<int>(n));
```

```
    for(int i = 0; i < n ; ++i) {
```

```
        for(int j = 0; j < n; ++j) {
```

```
            cin >> A[i][j];
```

```
        }
```

```
    }
```

```
    int pos, minimum;
```

```
    int min_tree = 0;
```

```
    //lowcost数组记录每2个点间最小权值，visited数组标记某点是否已访问
```

```
    vector<int> visited, lowcost;
```

```
    for (int i = 0; i < n; ++i) {
```

```
        visited.push_back(0);    //初始化为0，表示都没加入
```

```
    }
```

```

visited[0] = 1; //最小生成树从第一个顶点开始
for (int i = 0; i < n; ++i) {
    lowcost.push_back(A[0][i]); //权值初始化为与第一个点的距离
}
for (int i = 0; i < n; ++i) { //枚举n个顶点
    minimum = max_int;
    for (int j = 0; j < n; ++j) { //找到最小权边对应顶点
        if(!visited[j] && minimum > lowcost[j]) {
            minimum = lowcost[j];
            pos = j;
        }
    }
    if (minimum == max_int) //如果min = max_int表示已经不再有点可以加入最小生成树中
        break;
    min_tree += minimum;
    visited[pos] = 1; //加入最小生成树中
    for (int j = 0; j < n; ++j) {
        if(!visited[j] && lowcost[j] > A[pos][j]) lowcost[j] = A[pos][j]; //更新可更新边的权值
    }
}
cout << min_tree << endl;
}

```

用priority_queue实现 Prim + 堆

```
#define INFINITE 9000000000
```

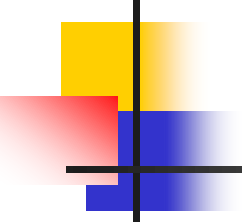
```
struct XEdge
{
    int v;
    int w;
    XEdge(int v_ = 0, int w_ = INFINITE):v(v_),w(w_) { }
};
```

```
vector<vector<XEdge> > G(30); //图的邻接表
bool operator <(const XEdge & e1, const XEdge & e2)
{
    return e1.w > e2.w;
}
```



```
int HeapPrim(const vector<vector<XEdge> > & G, int n)  
//G是邻接表,n是顶点数目, 返回值是最小生成树权值和
```

```
{  
    int i,j,k;  
    XEdge xDist(0,0);  
    priority_queue<XEdge> pq;  
    vector<int> vDist(n); //各顶点到已经建好的那部分树的距离(可以不要)  
    vector<int> vUsed(n);  
    int nDoneNum = 0;  
    for( i = 0;i < n;i ++ ) {  
        vUsed[i] = 0;  
        vDist[i] = INFINITE;  
    }  
    nDoneNum = 0;  
    int nTotalW = 0;  
    pq.push(XEdge(0,0));
```



```

while( nDoneNum < n && !pq.empty() ) {
    do {
        xDist = pq.top();          pq.pop();
    } while( vUsed[xDist.v] == 1 && ! pq.empty());
    if( vUsed[xDist.v] == 0 ) {
        nTotalW += xDist.w;
        vUsed[xDist.v] = 1; nDoneNum ++;
        for( i = 0; i < G[xDist.v].size(); i ++ ) {
            int k = G[xDist.v][i].v;
            if( vUsed[k] == 0 ) {
                if( vDist[k] > G[xDist.v][i].w ) {
                    vDist[k] = G[xDist.v][i].w;
                    pq.push(XEdge(k, G[xDist.v][i].w));
                }
            }
        }
    }
}

if( nDoneNum < n )
    return -1; //图不连通
return nTotalW;
}

```


问题描述 ccf-csp201412-4 最优灌溉

雷雷承包了很多片麦田，为了灌溉这些麦田，雷雷在第一个麦田挖了一口很深的水井，所有的麦田都从这口井来引水灌溉。

为了灌溉，雷雷需要建立一些水渠，以连接水井和麦田，雷雷也可以利用部分麦田作为“中转站”，利用水渠连接不同的麦田，这样只要一片麦田能被灌溉，则与其连接的麦田也能被灌溉。

现在雷雷知道哪些麦田之间可以建设水渠和建设每个水渠所需要的费用（注意不是所有麦田之间都可以建立水渠）。请问灌溉所有麦田最少需要多少费用来修建水渠。

输入格式

输入的第一行包含两个正整数 n, m ，分别表示麦田的片数和雷雷可以建立的水渠的数量。麦田使用 $1, 2, 3, \dots$ 依次标号。

接下来 m 行，每行包含三个整数 a_i, b_i, c_i ，表示第 a_i 片麦田与第 b_i 片麦田之间可以建立一条水渠，所需要的费用为 c_i 。

输出格式

输出一行，包含一个整数，表示灌溉所有麦田所需要的最小费用。

样例输入

4 4
1 2 1
2 3 4
2 4 2
3 4 3

样例输出

6