数据结构与算法 (七)

张铭 主讲

采用教材：张铭，王腾蛟，赵海燕 编写
高等教育出版社，2008. 6 （“十一五”国家级规划教材)

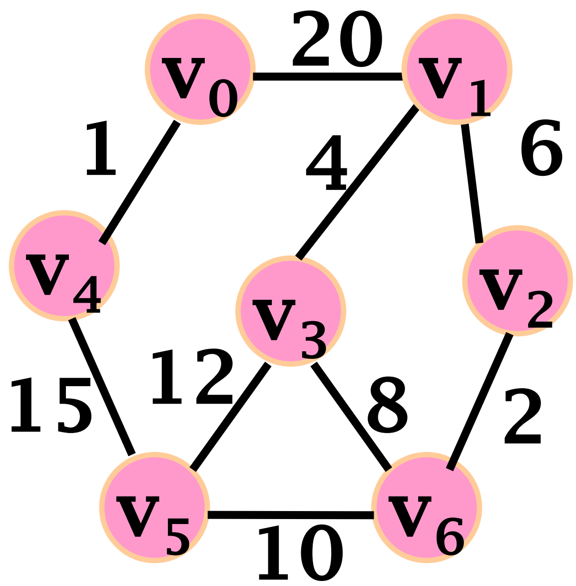
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg

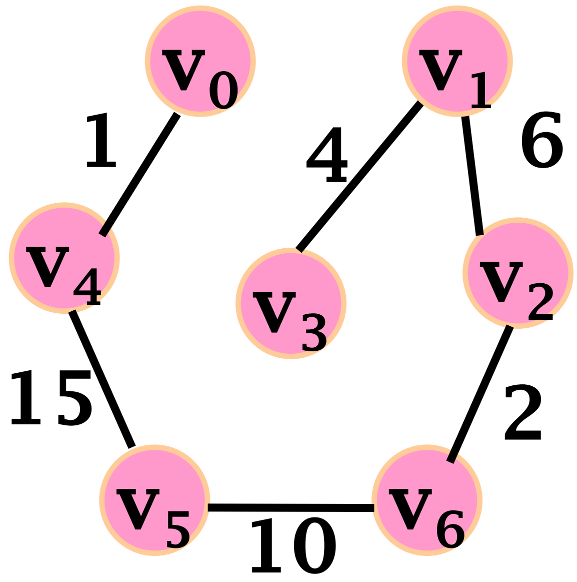
第7章 图

- 7.1 图的定义和术语
- 7.2 图的抽象数据类型
- 7.3 图的存储结构
- 7.4 图的遍历
- 7.5 最短路径
- 7.6 最小生成树

7.6 最小生成树

- 图 G 的生成树是一棵包含 G 的所有顶点的树，树上所有权值总和表示代价，那么在 G 的所有的生成树中
- 代价最小的生成树称为图 G 的 **最小生成树** (**minimum-cost spanning tree**，简称 **MST**)



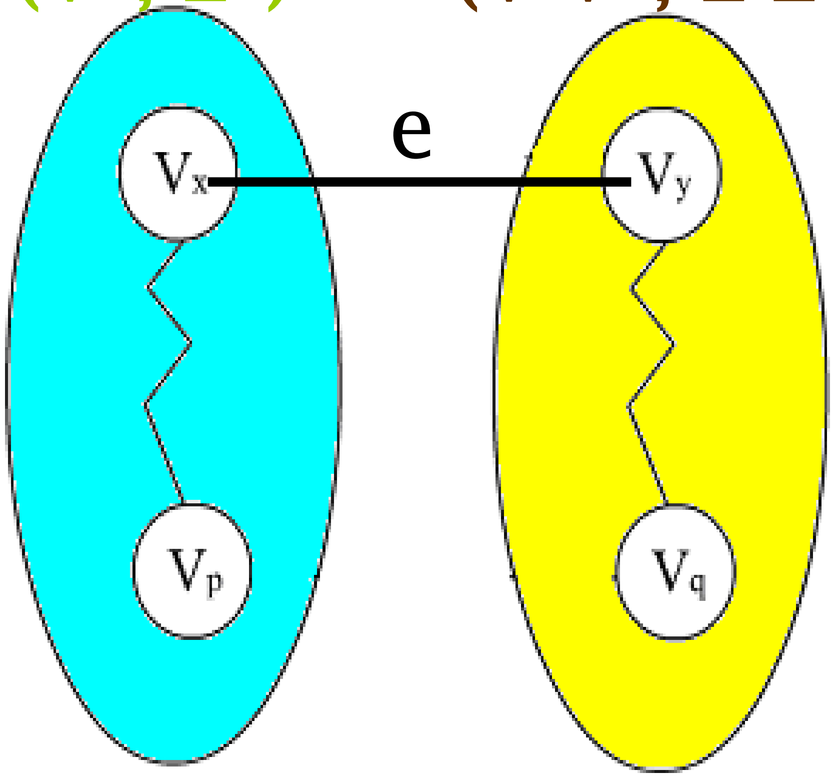


7.6.1 Prim 算法

- 与 Dijkstra 算法类似——也是贪心法
  - 从图中任意一个顶点开始 (例如 v_0)，首先把这个顶点包括在 MST，U = (V*, E*) 里
    - 初始 V*= {v_0}，E* = {}
  - 然后在那些其一个端点已在 MST 里，另一个端点**还不是 MST里的边**，找权最小的一条边 (v_p, v_q) ，并把此v_q 包括进 MST 里……
  - 如此进行下去，每次往 MST 里加一个顶点和一条权最小的边，直到把所有的顶点都包括进 MST 里
- 算法结束时 V* = V，E* 包含了 G 中的 n-1 条边

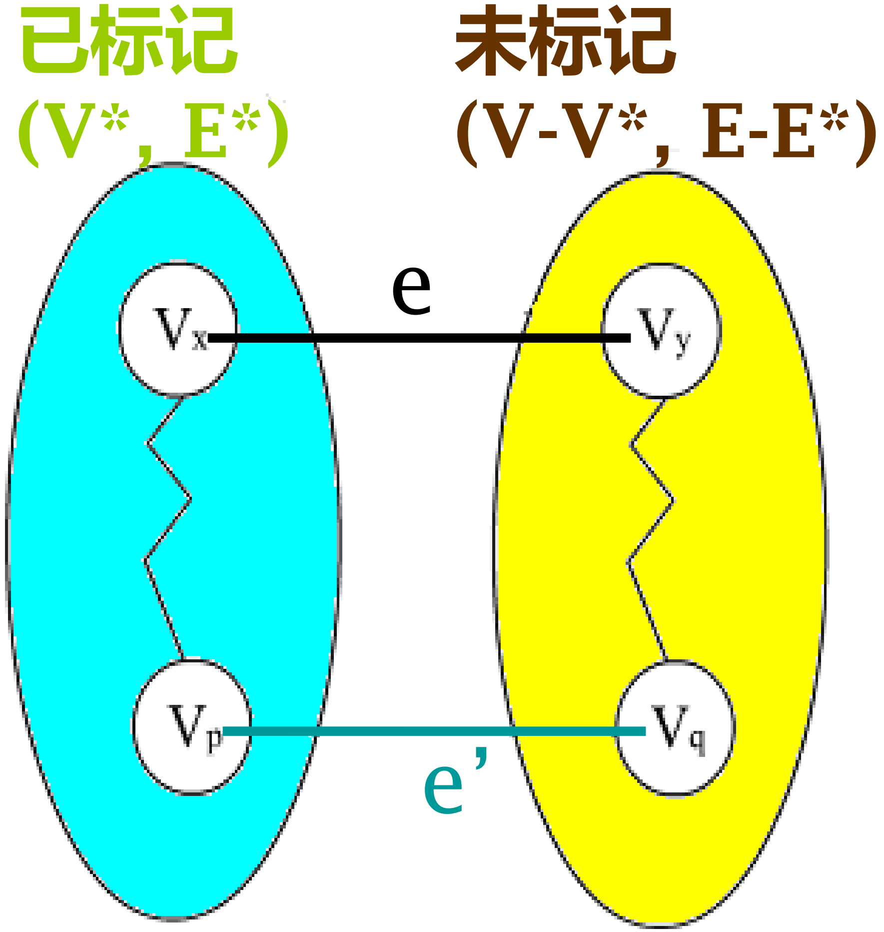
Prim 算法的 MST 性质

- 设 T(V*，E*) 是一棵正在构造的生成树
- E 中有边 e= (v_x，v_y)，其中v_x∈v*，v_y不属于V*
  - e 的权 w(e) 是所有一个端点在V* 里，另一端不在 V* 里的边的权中最小者
- 则一定存在 G 的一棵包括 T 的 MST 包括边 e= (v_x，v_y)



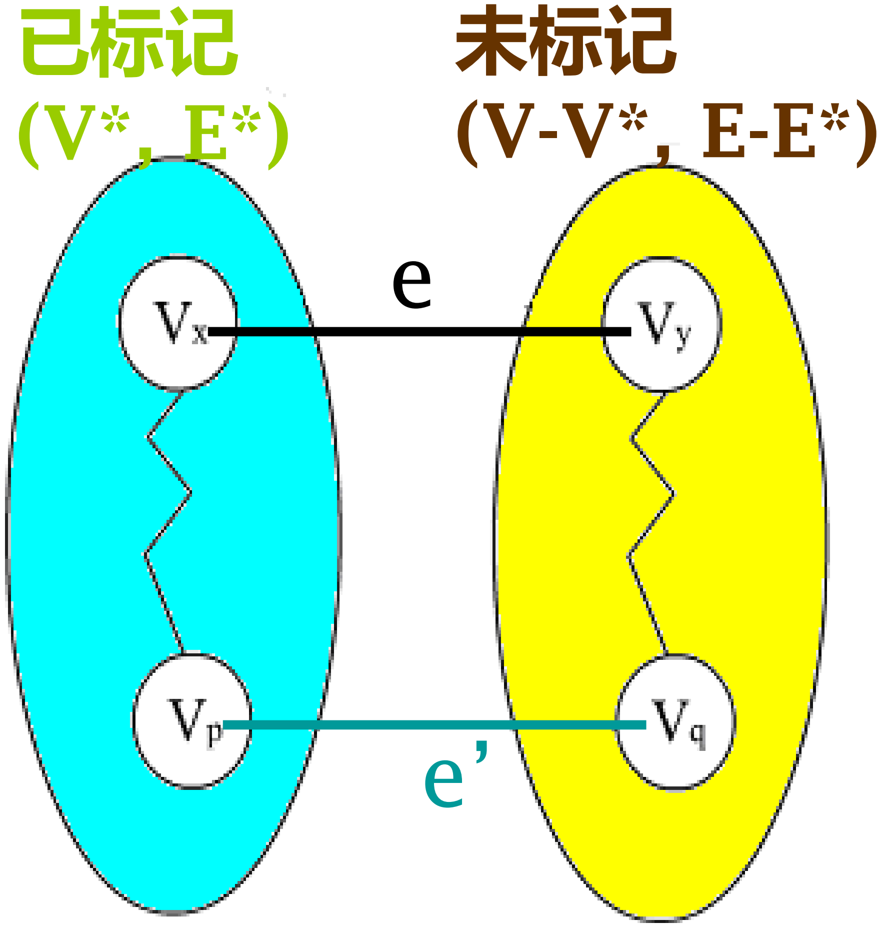
反证法证明 MST 性质

- 用反证法
  - 设G的任何一棵包括 T 的 MST 都不包括 e= (v_x , v_y) ，且设 T’ 是一棵这样的 MST
  - 由于 T’ 是连通的，因此有从 v_x 到 v_y 的路径 v_x , … , v_y



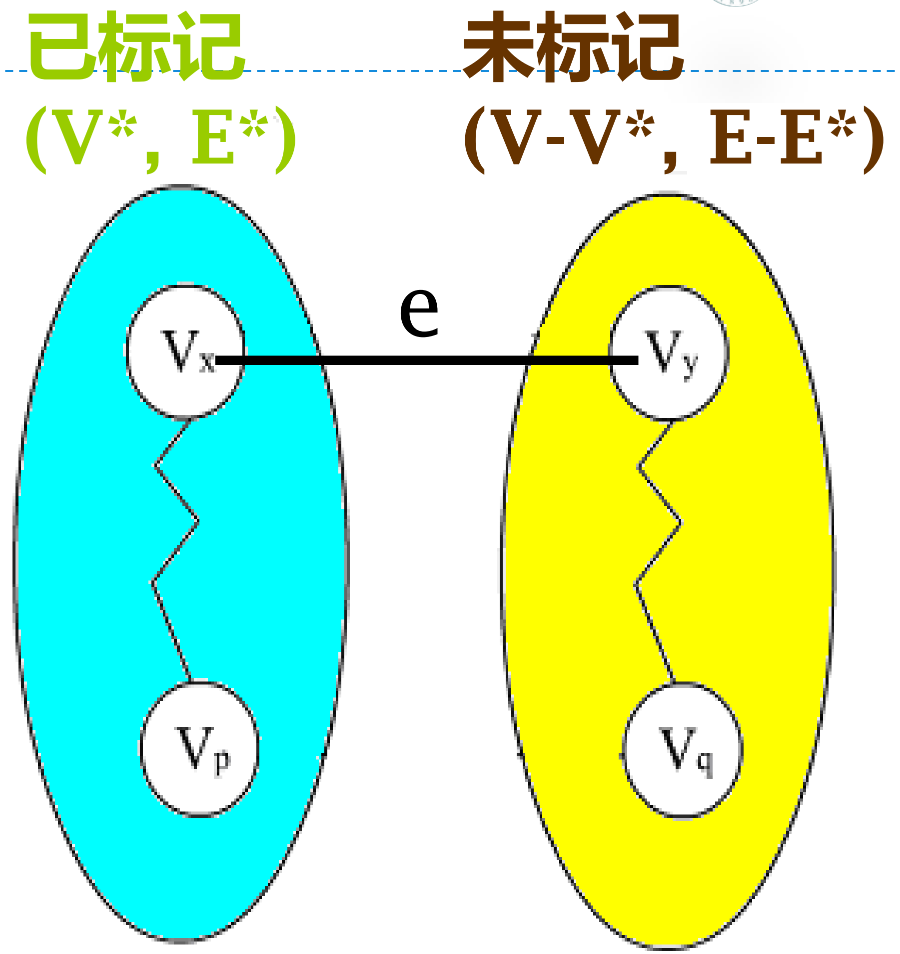
反证法证明 MST 性质 (续)

- 把边 $e=(V_x, V_y)$ 加进树 T'，得到一个回路 $v_x, \dots, v_y, v_x$
- 上述路径 $v_x, \dots, v_y$ 中必有边 $e'=(v_p, v_q)$，其中 $v_p \in V^*$，v_q 不属于 V^*，由条件知边的权 $w(e') \geq w(e)$，从回路中去掉边 e'
- 回路打开，成为另一棵生成树 T''，T'' 包括边 $e=(v_x, v_y)$，且各边权的总和不大于 T' 各边权的总和



反证法证明 MST 性质 (续)

- 因此 T” 是一棵包括边 e 的 MST，与假设矛盾，即证明了我们的结论
- 也证明了 Prim 算法构造 MST 的方法是正确的
  - 因为我们是从 T 包括任意一个顶点和0条边开始，每一步加进去的都是 MST 中应包括的边，直至最后得到 MST



Prim 算法

```
void Prim(Graph& G, int s, Edge* &MST)  {     // s是始点，MST存边
    int MSTtag = 0;                           // 最小生成树的边计数
    MST = new Edge[G.VerticesNum()-1];        // 为数组MST申请空间
    Dist *D;
    D = new Dist[G. VerticesNum()];           // 为数组D申请空间
    for (int i = 0; i < G.VerticesNum(); i++) {  // 初始化Mark和D数组
        G.Mark[i] = UNVISITED;
        D[i].index = i;
        D[i].length = INFINITE;
        D[i].pre = s;                         // D[i].pre = -1 呢？
    }
    D[s].length = 0;
    G.Mark[s]= VISITED;                       // 开始顶点标记为VISITED
    int v = s;
```

7.6 最小生成树

```
for (i = 0; i < G.VerticesNum()-1; i++) {
    // 因为v的加入，需要刷新与v相邻接的顶点的D值
    for (Edge e = G.FirstEdge(v); G.IsEdge(e); e = G.NextEdge(e))
        if (G.Mark[G.ToVertex(e)] != VISITED &&
        (D[G.ToVertex(e)].length > e.weight)) {
            D[G.ToVertex(e)].length = e.weight;
            D[G.ToVertex(e)].pre = v;
        }
    v = minVertex(G, D);                     // 在D数组中找最小值记为v
    if (v == -1) return;                     // 非连通，有不可达顶点
    G.Mark[v] = VISITED;                     // 标记访问过
    Edge edge(D[v].pre, D[v].index, D[v].length); // 保存边
    AddEdgetoMST(edge, MST, MSTtag++);       // 将边加入MST
  }
}
```

在 Dist 数组中找最小值

```
int minVertex(Graph& G, Dist* & D)  {
    int i, v = -1;
    int MinDist = INFINITY;
    for (i = 0; i < G.VerticesNum(); i++)
        if ((G.Mark[i] == UNVISITED) && (D[i] < MinDist)){
            v = i;          // 保存当前发现的最小距离顶点
            MinDist = D[i];
        }
    return v;
}
```

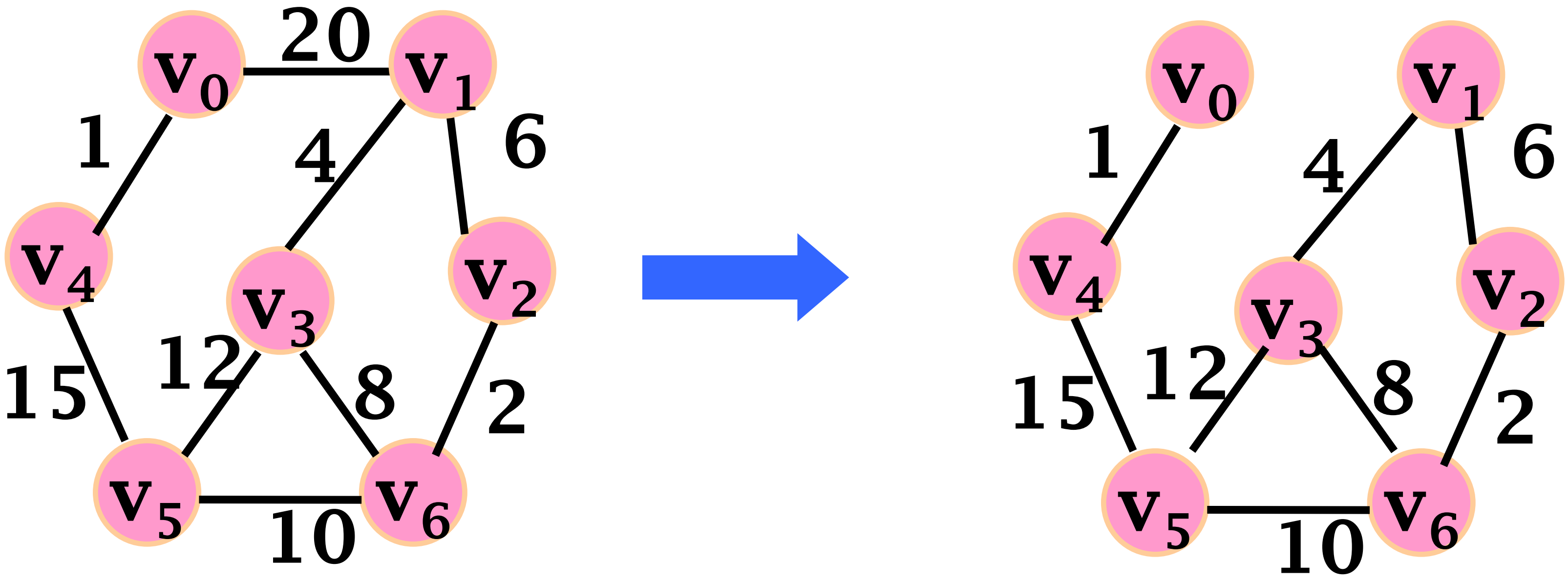
Prim 算法时间复杂度

- Prim 算法非常类似于 Dijkstra 算法
  - Prim 算法框架与 Dijkstra 算法相同
  - Prim 算法中的距离值不需要累积，直接用最小边
- 本算法通过直接比较 D 数组元素，确定代价最小的边就需要总时间O(n^2)；取出权最小的顶点后，修改 D 数组共需要时间O(e)，因此共需要花费O(n^2) 的时间
  - 算法适合于稠密图
  - 对于稀疏图，可以像 Dijkstra 算法那样用堆来保存距离值

7.6.2 Kruskal 算法

- 首先将 G 中的 n 个顶点看成是独立的 n 个连通分量，这时的状态是有 n 个顶点而无边的森林，可以记为 T = <V , {}>
- 然后在 E 中选择代价最小的边，如果该边依附于两个不同的连通分支，那么将这条边加入到 T 中，否则舍去这条边而选择下一条代价最小的边
- 依此类推，直到 T 中所有顶点都在同一个连通分量中为止，此时就得到图 G 的一棵最小生成树

最小生成树 Kruskal 算法



Kruskal 最小生成树算法

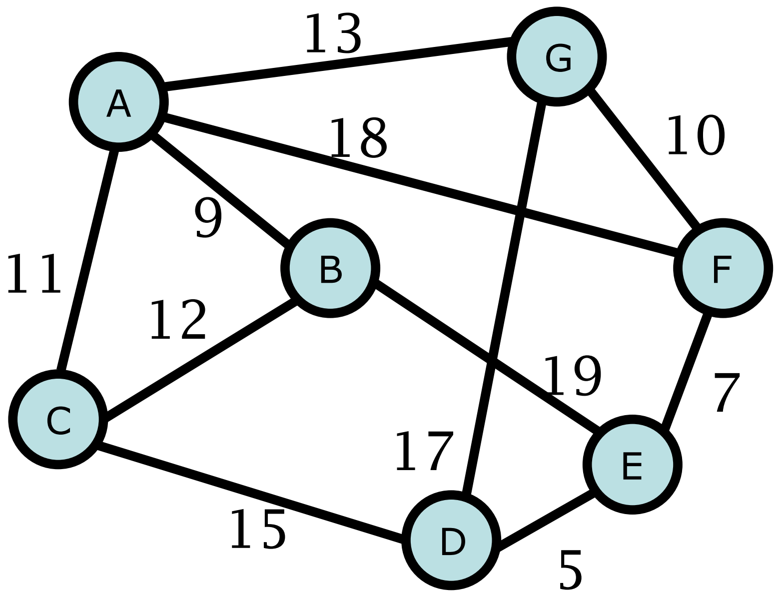
```
void Kruskal(Graph& G, Edge* &MST)  {     // MST存最小生成树的边
    ParTree<int> A(G.VerticesNum());       // 等价类
    MinHeap<Edge> H(G.EdgesNum());         // 最小堆
    MST = new Edge[G.VerticesNum()-1];     // 为数组MST申请空间
    int MSTtag = 0;                        // 最小生成树的边计数
    for (int i = 0; i < G.VerticesNum(); i++)  // 将所有边插入最小堆H中
        for (Edge e = G. FirstEdge(i); G.IsEdge(e); e = G. NextEdge(e))
            if (G.FromVertex(e) < G.ToVertex(e))// 防重复边
                H.Insert(e);
    int EquNum = G.VerticesNum();          // 开始有n个独立顶点等价类
```

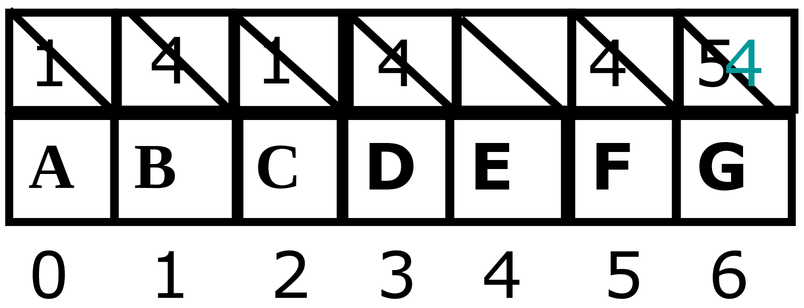
7.6 最小生成树

```
    while (EquNum > 1) {                    // 当等价类的个数大于1时合并等价类
        if (H.isEmpty()) {
            cout << "不存在最小生成树." <<endl;
            delete [] MST;
            MST = NULL;                     // 释放空间
            return;
        }
        Edge e = H.RemoveMin();             // 取权最小的边
        int from = G.FromVertex(e);         // 记录该条边的信息
        int to = G.ToVertex(e);
        if (A.Different(from,to)) {         // 边e的两个顶点不在一个等价类
            A.Union(from,to);               // 合并边的两个顶点所在的等价类
            AddEdgetoMST(e,MST,MSTtag++); // 将边e加到MST
            EquNum--;                       // 等价类的个数减1
        }
    }
}
```

7.6 最小生成树

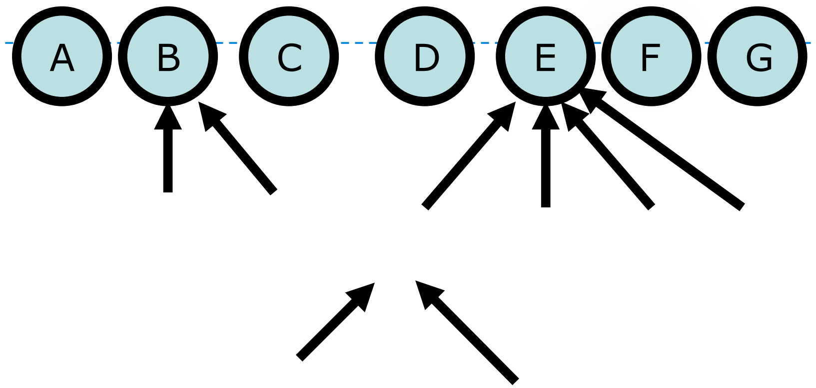
算法演示





- (D,E,5)
- (F,E,7)
- (A,B,9)
- (F,G,10)
- (A,C,11)
- (B,C,12)
- (A,G,13)
- (C,D,15)

…………



Kruskal 算法代价

- 使用了路径压缩，Different() 和 Union() 函数几乎是常数
- 假设可能对几乎所有边都判断过了
  - 则最坏情况下算法时间代价为 Θ (elog e)，即堆排序的时间
- 通常情况下只找了略多于 n 次，MST 就已经生成
  - 时间代价接近于 Θ (nlog e)

讨论

- 最小生成树是否唯一？
  - 不一定
  - 试设计算法生成所有的最小生成树
- 如果边的权都不相等
  - 一定是唯一的

数据结构与算法

谢谢聆听

国家精品课“数据结构与算法”
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

张铭，王腾蛟，赵海燕
高等教育出版社，2008. 6。“十一五”国家级规划教材