

原 算法导论--最小生成树 (Kruskal和Prim算法)

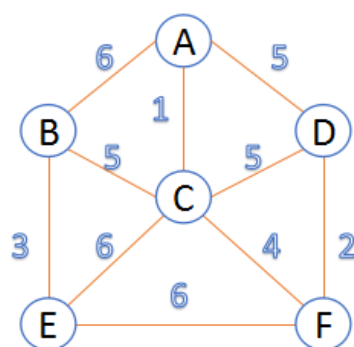
2016年07月14日 16:58:59 勿在浮砂筑高台 阅读数: 66522 标签: [kruskal](#) [prim](#) [最小生成树](#) [更多](#)

版权声明: 转载请注明出处! PS:欢迎大家提问或指正文章的错误! <https://blog.csdn.net/luoshixian099/article/details/51908175>

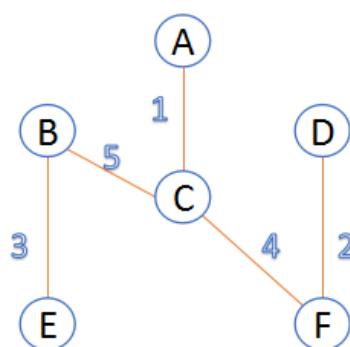
转载请注明出处: 勿在浮砂筑高台<http://blog.csdn.net/luoshixian099/article/details/51908175>

关于图的几个概念定义:

- **连通图**: 在无向图中, 若任意两个顶点 v_i 与 v_j 都有路径相通, 则称该无向图为连通图。
- **强连通图**: 在有向图中, 若任意两个顶点 v_i 与 v_j 都有路径相通, 则称该有向图为强连通图。
- **连通网**: 在连通图中, 若图的边具有一定的意义, 每一条边都对应着一个数, 称为权; 权代表着连接连个顶点的代价, 称这种连通图叫做连通网。
- **生成树**: 一个连通图的生成树是指一个连通子图, 它含有图中全部 n 个顶点, 但只有足以构成一棵树的 $n-1$ 条边。一颗有 n 个顶点的生成树有且仅有 $n-1$ 条边, 如果生成树中再添加一条边, 则必定成环。
- **最小生成树**: 在连通网的所有生成树中, 所有边的代价和最小的生成树, 称为最小生成树。



连通网G



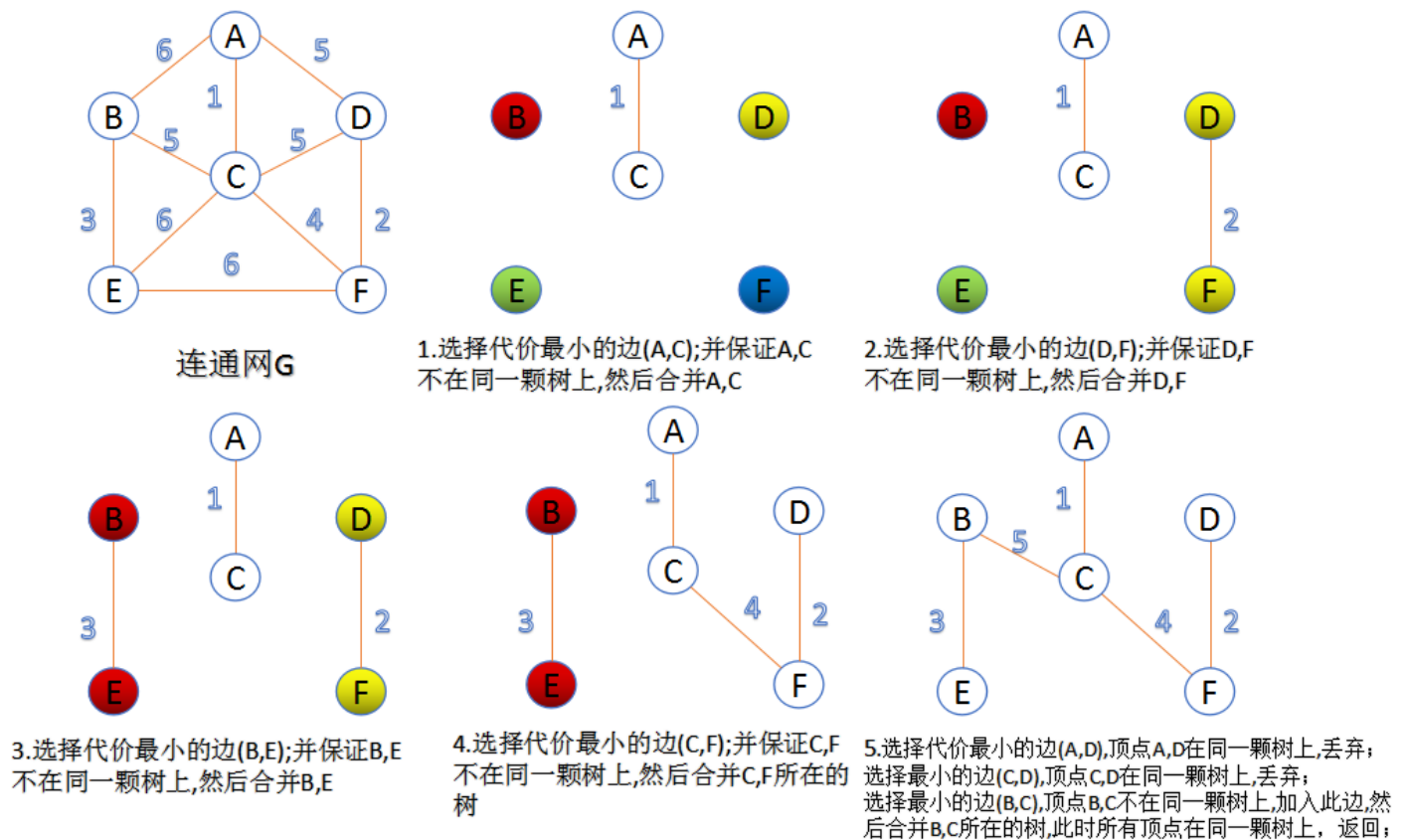
最小生成树

下面介绍两种求最小生成树算法

1.Kruskal算法

此算法可以称为“加边法”, 初始最小生成树边数为0, 每迭代一次就选择一条满足条件的最小代价边, 加入到最小生成树的边集合里。

1. 把图中的所有边按代价从小到大排序;
2. 把图中的 n 个顶点看成独立的 n 棵树组成的森林;
3. 按权值从小到大选择边, 所选的边连接的两个顶点 u_i, v_i 应属于两颗不同的树, 则成为最小生成树的一条边, 并将这两颗树合并作为一颗树。
4. 重复(3),直到所有顶点都在一颗树内或者有 $n-1$ 条边为止。



2.Prim算法

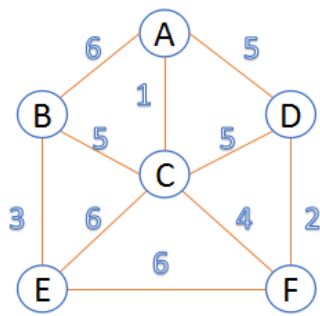
此算法可以称为“加点法”，每次迭代选择代价最小的边对应的点，加入到最小生成树中。算法从某一个顶点 s 开始，逐渐长大覆盖整个连通网的所有顶点。

1. 图的所有顶点集合为 V ；初始令集合 $u = \{s\}$, $v = V - u$ ；
2. 在两个集合 u, v 能够组成的边中，选择一条代价最小的边 (u_0, v_0) ，加入到最小生成树中，并把 v_0 并入到集合 u 中。
3. 重复上述步骤，直到最小生成树有 $n-1$ 条边或者 n 个顶点为止。

由于不断向集合 u 中加点，所以最小代价边必须同步更新；需要建立一个辅助数组closedge,用来维护集合 v 中每个顶点与集合 u 中最小代价边信息，：

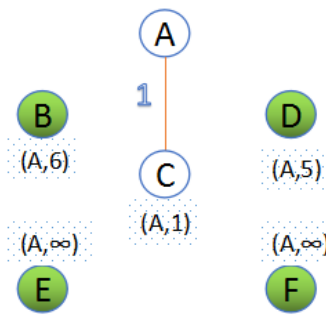
```

1 struct
2 {
3     char vertexData    //表示u中顶点信息
4     UINT lowestcost    //最小代价
5 }closedge[vexCounts]
```

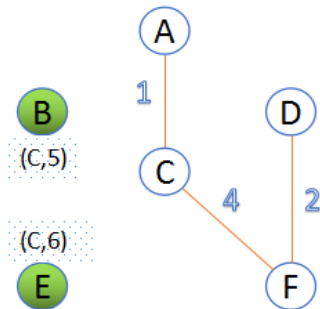
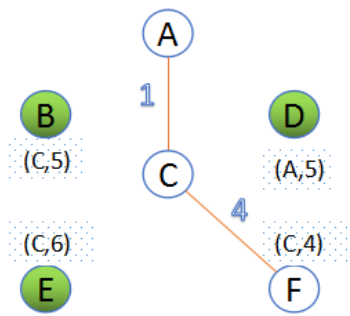


连通网G

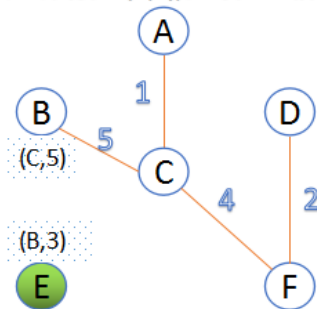
1. 初始 $u=\{A\}$, $v=\{B,C,D,E,F\}$; 顶点B下方(A,6), 表示与集合u中A的代价为6作为最小代价边。选择最小的代价边(A,C), 把C并入到集合u中。



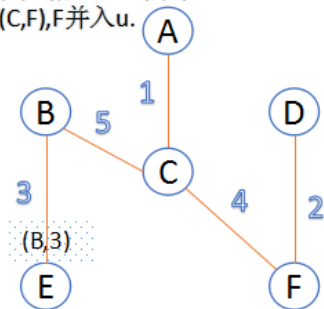
2. $u=\{A,C\}$, $v=\{B,D,E,F\}$; 更新v中顶点与集合u的最小的代价边; 例如: 顶点E之前为(A, ∞), 更新为(C, 6); 选择最小代价边(C,F), F并入u。



3. $u=\{A,C,F\}$, $v=\{B,D,E\}$; 更新v中顶点与集合u的最小的代价边; 选择最小代价边(F,D), D并入u。



4. $u=\{A,C,F,D\}$, $v=\{B,E\}$; 更新v中顶点与集合u的最小的代价边; 选择最小代价边(C,B), B并入u。



5. $u=\{A,C,F,D,B\}$, $v=\{E\}$; 更新v中顶点与集合u的最小的代价边; 选择最小代价边(B,E), E并入u。

3.完整代码

```

1  /*****
2  CSDN 勿在浮沙筑高台 http://blog.csdn.net/luoshixian099算法导论--最小生成树 (Prim、Kruskal) 2016年7月14日
3  *****/
4  #include <iostream>
5  #include <vector>
6  #include <queue>
7  #include <algorithm>
8  using namespace std;
9  #define INFINITE 0xFFFFFFFF
10 #define VertexData unsigned int // 顶点数据
11 #define UINT unsigned int
12 #define vexCounts 6 // 顶点数量
13 char vextex[] = { 'A', 'B', 'C', 'D', 'E', 'F' };
14 struct node
15 {
16     VertexData data;
17     unsigned int lowestcost;
18 }closededge[vexCounts]; // Prim算法中的辅助信息
19 typedef struct
20 {
21     VertexData u;
22     VertexData v;
23     unsigned int cost; // 边的代价
24 }Arc; // 原始图的边信息

```



63



14



```

25 void AdjMatrix(unsigned int adjMat[][vexCounts]) //邻接矩阵表示法
26 {
27     for (int i = 0; i < vexCounts; i++) //初始化邻接矩阵
28         for (int j = 0; j < vexCounts; j++)
29             {
30                 adjMat[i][j] = INFINITE;
31             }
32     adjMat[0][1] = 6; adjMat[0][2] = 1; adjMat[0][3] = 5;
33     adjMat[1][0] = 6; adjMat[1][2] = 5; adjMat[1][4] = 3;
34     adjMat[2][0] = 1; adjMat[2][1] = 5; adjMat[2][3] = 5; adjMat[2][4] = 6; adjMat[2][5] = 4;
35     adjMat[3][0] = 5; adjMat[3][2] = 5; adjMat[3][5] = 2;
36     adjMat[4][1] = 3; adjMat[4][2] = 6; adjMat[4][5] = 6;
37     adjMat[5][2] = 4; adjMat[5][3] = 2; adjMat[5][4] = 6;
38 }
39 int Minmum(struct node * closededge) //返回最小代价边
40 {
41     unsigned int min = INFINITE;
42     int index = -1;
43     for (int i = 0; i < vexCounts; i++)
44     {
45         if (closededge[i].lowestcost < min && closededge[i].lowestcost != 0)
46         {
47             min = closededge[i].lowestcost;
48             index = i;
49         }
50     }
51     return index;
52 }
53 void MiniSpanTree_Prim(unsigned int adjMat[][vexCounts], VertexData s)
54 {
55     for (int i = 0; i < vexCounts; i++)
56     {
57         closededge[i].lowestcost = INFINITE;
58     }
59     closededge[s].data = s; //从顶点s开始
60     closededge[s].lowestcost = 0;
61     for (int i = 0; i < vexCounts; i++) //初始化辅助数组
62     {
63         if (i != s)
64         {
65             closededge[i].data = s;
66             closededge[i].lowestcost = adjMat[s][i];
67         }
68     }
69     for (int e = 1; e <= vexCounts - 1; e++) //n-1条边时退出
70     {
71         int k = Minmum(closededge); //选择最小代价边
72         cout << vextex[closededge[k].data] << "--" << vextex[k] << endl; //加入到最小生成树
73         closededge[k].lowestcost = 0; //代价置为0
74         for (int i = 0; i < vexCounts; i++) //更新v中顶点最小代价边信息
75         {
76             if (adjMat[k][i] < closededge[i].lowestcost)
77             {
78                 closededge[i].data = k;
79                 closededge[i].lowestcost = adjMat[k][i];
80             }
81         }
82     }

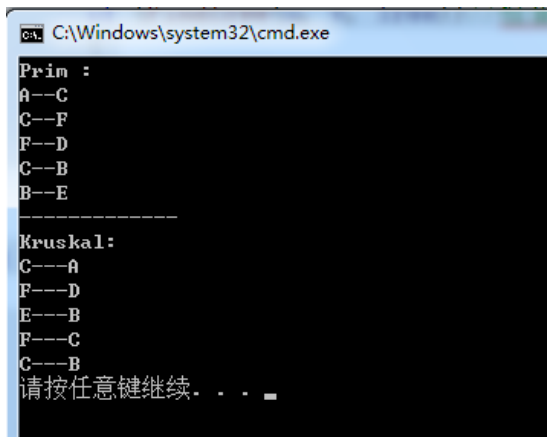
```

```

83 }
84 void ReadArc(unsigned int adjMat[][vexCounts],vector<Arc> &vertexArc) //保存图的边代价信息
85 {
86     Arc * temp = NULL;
87     for (unsigned int i = 0; i < vexCounts;i++)
88     {
89         for (unsigned int j = 0; j < i; j++)
90         {
91             if (adjMat[i][j]!=INFINITE)
92             {
93                 temp = new Arc;
94                 temp->u = i;
95                 temp->v = j;
96                 temp->cost = adjMat[i][j];
97                 vertexArc.push_back(*temp);
98             }
99         }
100     }
101 }
102 bool compare(Arc A, Arc B)
103 {
104     return A.cost < B.cost ? true : false;
105 }
106 bool FindTree(VertexData u, VertexData v,vector<vector<VertexData> > &Tree)
107 {
108     unsigned int index_u = INFINITE;
109     unsigned int index_v = INFINITE;
110     for (unsigned int i = 0; i < Tree.size();i++) //检查u,v分别属于哪颗树
111     {
112         if (find(Tree[i].begin(), Tree[i].end(), u) != Tree[i].end())
113             index_u = i;
114         if (find(Tree[i].begin(), Tree[i].end(), v) != Tree[i].end())
115             index_v = i;
116     }
117
118     if (index_u != index_v) //u,v不在一颗树上, 合并两颗树
119     {
120         for (unsigned int i = 0; i < Tree[index_v].size();i++)
121         {
122             Tree[index_u].push_back(Tree[index_v][i]);
123         }
124         Tree[index_v].clear();
125         return true;
126     }
127     return false;
128 }
129 void MiniSpanTree_Kruskal(unsigned int adjMat[][vexCounts])
130 {
131     vector<Arc> vertexArc;
132     ReadArc(adjMat, vertexArc);//读取边信息
133     sort(vertexArc.begin(), vertexArc.end(), compare);//边按从小到大排序
134     vector<vector<VertexData> > Tree(vexCounts); //6棵独立树
135     for (unsigned int i = 0; i < vexCounts; i++)
136     {
137         Tree[i].push_back(i); //初始化6棵独立树的信息
138     }
139     for (unsigned int i = 0; i < vertexArc.size(); i++)//依次从小到大取最小代价边
140     {

```

```
141     VertexData u = vertexArc[i].u;
142     VertexData v = vertexArc[i].v;
143     if (FindTree(u, v, Tree))//检查此边的两个顶点是否在一颗树内
144     {
145         cout << vextex[u] << "----" << vextex[v] << endl;//把此边加入到最小生成树中
146     }
147 }
148 }
149
150 int main()
151 {
152     unsigned int adjMat[vexCounts][vexCounts] = { 0 };
153     AdjMatrix(adjMat); //邻接矩阵
154     cout << "Prim :" << endl;
155     MiniSpanTree_Prim(adjMat,0); //Prim算法,从顶点0开始.
156     cout << "-----" << endl << "Kruskal:" << endl;
157     MiniSpanTree_Kruskal(adjMat);//Kruskal算法
158     return 0;
159 }
```



```
C:\Windows\system32\cmd.exe
Prim :
A--C
C--F
F--D
C--B
B--E
-----
Kruskal:
C---A
F---D
E---B
F---C
C---B
请按任意键继续. . .
```

Reference:

数据结构-耿国华

算法导论-第三版



静静静静静rj: 大神大神,你为什么只更到2017年就没有更新啦。。。是以后都不更了吗吗😭😭😭 (2个月前 #13楼)



whm1234566777: 学习了 (3个月前 #12楼)



stone-jack: 内存泄漏!! (3个月前 #11楼)



ljp946: 万分感谢!!! (3个月前 #10楼)



ljp946: 感谢大犇的解释,很清楚! (3个月前 #9楼)



AUTUMN_QQ: 谢谢您!我终于理解了 (4个月前 #8楼)