

最小生成树

引言

图论的基础内容，例如稀疏图用邻接链表，稠密图用邻接矩阵、DFS、BFS和Dijkstra都已经写过了，一个重要工具并查集也写过了。那么接下来，本文是图论的核心内容：最小生成树。

最小生成树的定义等等，都已经十分清楚了，无需多讲。还需要用到一个工具叫割集（反圈），列在第一部分。“割集”是离散数学的名词，与任韩图论中“反圈”是同一个概念。割集也称为反圈，是因为在最小生成树体系下割集与圈是对偶的两个概念，很多性质可以照搬。

接下来会简要阐述算法部分、Prim算法（又称反圈法或割集法），可以使用优先队列优化，适用于稠密图，因此常与邻接矩阵搭配、Kruskal算法（避圈法），必须采用并查集优化，适用于稀疏图，因此常与邻接链表搭配。

其他不常见的算法例如Rosenstiehl算法（破圈法），因为难以优化，不如上两种算法好，已经被弃用了，因此不讲。

割集（反圈）

割集（反圈）：将图的顶点集划分为两部分，连接两部分的全体边构成割集。将图 G 的顶点集合 V 分成非空两部分 S 和 $V-S$ 、图 G 中连接 S 和 $V-S$ 两部分的边的全体，称为 G 的一个反圈。

简单地来讲，反圈刻画了图 G 两部分顶点间的连通性。如果这两部分顶点之间连通性强，那么对应的反圈的边数多。同样地，如果这两部分顶点之间连通性弱，那么对应的反圈的边数少。

因此有一个简单结论：

对于图 G 的支撑子图 H 、如果 H 连通，那么 H 与 G 的任意一个反圈有公共边。如果 H 不连通，那么存在 G 的反圈与 H 无公共边。

这个模型称为“反圈”的原因，与下面的结论有关。

设 T 是图 $G=(V,E)$ 的一个支撑树，则 T 满足以下特征：

T 中没有圈、 $G-E(T)$ 中没有 G 的反圈。

T 添加任何一条边后，图中的边包含且仅包含一个圈，称为 G 关于 T 的一个基本圈。

对于 T 中任意一条边 $e \in G-E(T)$ 、 e 中包含且仅包含 G 的一个反圈，称为 G 关于 T 的一个基本反圈。

只需要解释基本反圈的唯一性。事实上这个命题等价于：

从树中任意去掉一条边，恰好包含两个分支。

因此仅当反圈选择的两部分顶点恰好为这两个分支的时候，才能与去掉一条边的树恰好没有公共边。

基本割集（反圈）：生成树的每条树枝对应一个基本割集。

割集与生成树至少有一条公共边。

圈与割集有偶数条公共边。

圈在生成树外至少有一条边。

基本圈中的弦，位于圈中树枝的每一个基本割集，不在其他基本割集中。

最小支撑树T满足以下特征：

对于T中任意一条边 $e \in E$ 是由 e 决定的基本反圈中的最小权边。

对于不在T中的任意一条边 $e \in E$ 是由 e 决定的基本圈中的最大权边。

Prim算法（反圈法）

第一步：任取一个节点作为初始点。

第二步：从已选点和未选点构成的反圈中，选择权最小的边。如果有多条边权最小，则任选其中一条。

第三步：若在某一步，反圈为空集，则图中没有支撑树。若在某一步，已经选择了图的所有顶点，则所有被选择的边构成最小树，算法终止。

可以看一个程序示例：

```
//weights为权重数组 n为顶点个数 src为最小树的第一个顶点 edge为最小生成树边

void Prim(int weights[][512], int n, int src, int edges[])
{
    int minweight[512],min;
    int i,j,k;
    for(i=0;i<n;i++)//初始化相关数组
    {
        minweight[i]=weights[src][i]; //将src顶点与之有边的权值存入数组
        edges[i]=src; //初始化第一个顶点为src
    }
    minweight[src]=0; //将第一个顶点src顶点加入生成树
    for(i=1;i<n;i++)
    {
        min=32767;
        for(j=0,k=0;j<n;j++)
        {
            if(minweight[j]!=0&&minweight[j]<min)//在数组中找最小值，其下标为k
            {
                min=minweight[j];
                k=j;
            }
        }
        minweight[k]=0; //找到最小树的一个顶点
        for(j=0;j<n;j++)
        {
            if(minweight[j]!=0&&weights[k][j]<minweight[j] )
```

```

        {
            minweight[j]=weights[k][j];    //将小于当前权值的边(k,j) 权值加
入数组中
            edges[j]=k;    //将边(j,k) 信息存入边数组中
        }
    }
}

```

上面这个版本没有被优化。若想看优化，可以参照下面两个版本。

```

#include<algorithm>
#include<iostream>
#include<cstring>
#include<stdio.h>
#include<math.h>
#include<string>
#include<stdio.h>
#include<queue>
#include<stack>
#include<map>
#include<deque>
using namespace std;
struct edge//保存边的情况[]to为通往的边，不需要保存from
{
    int to;
    int v;
    friend bool operator<(const edge& x,const edge& y)//优先队列即最小堆
    {
        return x.v>y.v;
    }
};
priority_queue<edge>q;
int vis[105];//判断是否标记数组
int p[105][105];//存图
int n;
int main()
{
    int i,j,x,y,d2,d1,s,key;
    edge now;
    while(scanf("%d",&n)!=EOF)
    {
        for(i=0;i<n;i++)
        {
            vis[i]=0;//初始化一下
            for(j=0;j<n;j++)
            {
                scanf("%d",&p[i][j]);
            }
        }
        s=0;
    }
}

```

```
vis[0]=1;//标记起始点
key=0;//随便找起始点
while(!q.empty())q.pop();
for(i=0;i<n-1;i++)//n-1次
{
    for(j=0;j<n;j++)//记入新加入点的情况
    {
        if(!vis[j])//没标记过的点就加入全家桶套餐
        {
            now.to=j;
            now.v=p[key][j];
            q.push(now);
        }
    }
    while(!q.empty()&&vis[q.top().to])//最小边但是标记过就放弃
    {
        q.pop();
    }
    if(q.empty())
        break;
    now=q.top();
    key=now.to;
    s+=now.v;//累加最小边的和
    vis[key]=1;
    q.pop();
}
printf("%d\n",s);
}
return 0;
}
```

```
#include <iostream>
#include <queue>
#include <map>
#include <cstring>
using namespace std;
#define maxint 0x3f3f3f3f
#define maxnum 1051

int link[maxnum][maxnum];
int c[maxnum][maxnum];
int sum,n;//sum为最小权之和n为顶点个数

struct node
{
    int s;//起点
    int e;//终点
    int w;//权
};
```

```

bool operator < (const node &a, const node &b)
{
    return a.w > b.w;
}

void prim(int s)
{
    int i, j, k, m, t, u, total;
    int vis[maxnum]; // 标记访问
    memset(vis, 0, sizeof(vis)); // 初始化vis均为0, 即未被访问
    priority_queue <node> qq; // 声明一个存储node结构体的优先队列

    struct node nn;

    total = 1;
    vis[s] = 1;
    sum = 0;
    while(total < n) // 遍历所有的顶点
    {
        for(i=1; i<link[s][0]; i++) // 遍历所有和s点相连的边, s点为源点
        {
            if(!vis[link[s][i]]) // 若这个边没被访问, 就将其加入优先队列
            {
                nn.s = s;
                nn.e = link[s][i];
                nn.w = c[s][nn.e];
                qq.push(nn);
            }
        }

        // 这里就是简单处理一下特殊情况
        while(!qq.empty() && vis[qq.top().e]) // 遇到顶点和集合外的顶点没有相连的
            qq.pop(); // 恰巧这个点作为终点是最短的, 因为这个顶点没背标记过, 所以会错误的
        计入在内

        // 将优先队列的队顶元素输出
        nn = qq.top();
        s = nn.e;
        sum += nn.w; // 队顶的边就是最适合的边, 因为优先队列的作用就是对权值进行排序, 队顶
        总是

                // 最大或最小的权值的边, 又因为没被访问过, 所有一定是最适合的
        // cout<<nn.s<<" "<<nn.e<<" "<<nn.w<<endl;
        vis[s] = 1; // 标记为集合内的元素
        qq.pop();
        total++; // 访问的点数加一
    }
}

int main()
{

```

```
int i,j,k;
int line,len;
int t,s,d,p,q;

cin>>n>>line;

for(i=1;i<=n;i++)
{
    link[i][0] = 1;
}

for(i=1;i<=line;i++)
{
    cin>>p>>q>>len;
    c[p][q] = c[q][p] = len;
    link[p][link[p][0]++] = q;
    link[q][link[q][0]++] = p;
}

cin>>s;//输入起始点
prim(s);

cout<<sum<<endl;

return 0;
}
```

Kruskal算法（避圈法）

第一步：选取一个无圈支撑子图。初始时选择图的全体节点。

第二步：若图连通，则它是最小支撑树。

第三步：若图不连通，则选择边 e 两个端点属于不同分支，并且权最小。重复上述过程。

有趣结论

在本文的最后，介绍一个与生成树（支撑树）有趣的结论：如何编程判断一个图是不是二部图。

如果存在图 G 的一个支撑树 T 使得所有的基本圈长都是偶数，那么 G 的所有圈长都是偶数，即 G 是二部图。

当然，如果是二部图，所有圈长都会是偶数，因此随便一个生成树都可以判断，不一定最小。

证明这个结论的关键，在于编程中常用的“异或”运算。

异或运算的实质是，将并集中出现偶数次的元素去掉。为了叙述方便，简记每个节点度数都是偶数的子图为“偶度子图”。有以下显然结论：

非空偶度子图一定有圈。

偶度子图的异或还是偶度子图。

不含奇长圈的图（二部图），异或也不含奇长圈。

于是要证结论等价于：

图G中任意一个圈可以写成基本圈的异或。

这近乎于显然。图G中的圈C必然有边不在支撑树T中，每一条这样的边都对应一个基本圈。这些基本圈的异或全体可以得到图C0只要证明C0和C完全相同。

由条件C0是偶度子图。

对于树T之外，圈C中的边，由基本圈的定义，这些边在C0中存在并且仅存在一次。

对于树T之外，不在圈C中的边，显然也不会选取的基本圈中，所以不在C0中。

因此考虑C0和C的异或，得到的结果必然包含在树T之内。但是它们的异或是偶度子图，而树T当中没有圈，由于非空偶度子图一定有圈，只能说明C0和C的异或是空集。因此C0和C完全相同。

From:

<https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link:

<https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:namespace:%E6%9C%80%E5%B0%8F%E7%94%9F%E6%88%90%E6%A0%91&rev=1594373531>

Last update: 2020/07/10 17:32