最小斯坦纳树

算法简介

一种用于计算只含图中部分关键节点的最小生成树的算法。

算法实现

考虑状压 dp第一维表示当前包含节点的点集，第二维表示树根节点。考虑两步转移

$$ \text{dp}(s,u)\gets \min_{k\subset s}(\text{dp}(k,u)+\text{dp}(s\oplus k,u)) $$

$$ \text{dp}(s,u)\gets \min(\text{dp}(s,u),\text{dp}(s,v)+w) $$

时间复杂度据说是 $O(2^kn^3+3^kn)$

算法模板

spfa 版本

```
namespace SteinerTree{
    int n,k,dp[1<<MAXK][MAXN];
    bool inque[MAXN];
    void spfa(int S){
        queue<int>q;
        _rep(i,1,n){
            if(dp[S][i]!=Inf){
                q.push(i);
                inque[i]=true;
            }
        }
        while(!q.empty()){
            int u=q.front();q.pop();
            inque[u]=false;
            for(int i=head[u];i;i=edge[i].next){
                int v=edge[i].to;
                if(dp[S][u]+edge[i].w<dp[S][v]){
                    dp[S][v]=dp[S][u]+edge[i].w;
                    if(!inque[v]){
                        q.push(v);
                        inque[v]=true;
                    }
                }
            }
        }
```

```
    }
    int build(int Node_cnt,int Keynode_cnt,int *key_node){
        n=Node_cnt,k=Keynode_cnt;
        _for(i,1,1<<k)_rep(j,1,n)
        dp[i][j]=Inf;
        _for(i,0,k)
        dp[1<<i][key_node[i]]=0;
        _for(i,0,1<<k){
            _rep(j,1,n){
                for(int k=i&(i-1);k;k=(k-1)&i)
                dp[i][j]=min(dp[i][j],dp[k][j]+dp[i^k][j]);
            }
            spfa(i);
        }
        int ans=Inf;
        _rep(i,1,n)
        ans=min(ans,dp[(1<<k)-1][i]);
        return ans;
    }
}
```

dijkstra 版本

```
namespace SteinerTree{
    int n,k,dp[1<<MAXK][MAXN];
    bool vis[MAXN];
    void dijkstra(int S){
        priority_queue<pair<int,int>,vector<pair<int,int>
>,greater<pair<int,int> > >q;
        _rep(i,1,n){
            vis[i]=false;
            if(dp[S][i]!=Inf)
            q.push(make_pair(dp[S][i],i));
        }
        while(!q.empty()){
            int u=q.top().second;q.pop();
            if(vis[u])continue;
            vis[u]=true;
            for(int i=head[u];i;i=edge[i].next){
                int v=edge[i].to;
                if(dp[S][u]+edge[i].w<dp[S][v]){
                    dp[S][v]=dp[S][u]+edge[i].w;
                    q.push(make_pair(dp[S][v],v));
                }
            }
        }
    }
    int build(int Node_cnt,int Keynode_cnt,int *key_node){
        n=Node_cnt,k=Keynode_cnt;
```

```
        _for(i,1,1<<k)_rep(j,1,n)
        dp[i][j]=Inf;
        _for(i,0,k)
        dp[1<<i][key_node[i]]=0;
        _for(i,0,1<<k){
            _rep(j,1,n){
                for(int k=i&(i-1);k;k=(k-1)&i)
                dp[i][j]=min(dp[i][j],dp[k][j]+dp[i^k][j]);
            }
            dijkstra(i);
        }
        int ans=Inf;
        _rep(i,1,n)
        ans=min(ans,dp[(1<<k)-1][i]);
        return ans;
    }
}
```

算法习题

习题一

洛谷p6192

题意

给定一个图，图中有 p 个关键点，每个关键点有一个类型 c_i求一个边权最小的子图使得要求所有 c_i 相同的关键点连通。

题解

先对所有关键点跑一遍最小斯坦纳树，接下来设 $g(s)$ 表示类型集合二进制表示为 s 的所有关键点连通的最小费用。

设 s' 表示所有 $1\le i\le p$ 且 $c_i\in s$ 构成的集合的二进制表示，则 $g(s)$ 的初始值为 $\min_{1\le u\le n}dp(s',u)$

接下来跑一遍 $g(s)\gets \min_{k\subset s}(g(k)+g(s\oplus k))$ 即可，时间复杂度等于最小斯坦纳树时间复杂度。

```
const int MAXN=1005,MAXM=3005,MAXK=10,Inf=1e9;
int head[MAXN],edge_cnt,key_node[MAXK];
struct Edge{
    int to,next,w;
}edge[MAXM<<1];
void Insert(int u,int v,int w){
```

```
    edge[++edge_cnt]=Edge{v,head[u],w};
    head[u]=edge_cnt;
}
namespace SteinerTree{
    int n,k,dp[1<<MAXK][MAXN];
    bool inque[MAXN];
    void spfa(int S){
        queue<int>q;
        _rep(i,1,n){
            if(dp[S][i]!=Inf){
                q.push(i);
                inque[i]=true;
            }
        }
        while(!q.empty()){
            int u=q.front();q.pop();
            inque[u]=false;
            for(int i=head[u];i;i=edge[i].next){
                int v=edge[i].to;
                if(dp[S][u]+edge[i].w<dp[S][v]){
                    dp[S][v]=dp[S][u]+edge[i].w;
                    if(!inque[v]){
                        q.push(v);
                        inque[v]=true;
                    }
                }
            }
        }
    }
    int build(int Node_cnt,int Keynode_cnt,int *key_node){
        n=Node_cnt,k=Keynode_cnt;
        _for(i,1,1<<MAXK)_rep(j,1,n)
        dp[i][j]=Inf;
        _for(i,0,k)
        dp[1<<i][key_node[i]]=0;
        _for(i,0,1<<k){
            _rep(j,1,n){
                for(int k=i&(i-1);k;k=(k-1)&i)
                dp[i][j]=min(dp[i][j],dp[k][j]+dp[i^k][j]);
            }
            spfa(i);
        }
        int ans=Inf;
        _rep(i,1,n)
        ans=min(ans,dp[(1<<k)-1][i]);
        return ans;
    }
}
vector<int> Node[MAXK];
int g[1<<MAXK];
```

```
int main()
{
    int n=read_int(),m=read_int(),p=read_int();
    while(m--){
        int u=read_int(),v=read_int(),w=read_int();
        Insert(u,v,w);Insert(v,u,w);
    }
    _for(i,0,p){
        int c=read_int(),d=read_int();
        Node[c-1].push_back(i);
        key_node[i]=d;
    }
    SteinerTree::build(n,p,key_node);
    _for(i,1,1<<MAXK){
        int s=0;
        _for(j,0,MAXK){
            if((i>>j)&1) _for(k,0,Node[j].size())
            s|=(1<<Node[j][k]);
        }
        g[i]=Inf;
        _rep(j,1,n)
        g[i]=min(g[i],SteinerTree::dp[s][j]);
    }
    _for(i,1,1<<MAXK){
        for(int j=i&(i-1);j;j=(j-1)&i)
        g[i]=min(g[i],g[j]+g[i^j]);
    }
    enter(g[(1<<MAXK)-1]);
    return 0;
}
```