

# 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  $\text{dp}$  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

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

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

时间复杂度据说是  $O(2^{kn^3} + 3^{kn})$

## 算法模板

### $\text{spfa}$ 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;  
    }  
}
```

## $\text{\text{dijkstra}}$ 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;
}
}
```

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

Permanent link:  
[https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal\\_string:jxm2001:%E6%9C%80%E5%B0%8F%E6%96%AF%E5%9D%A6%E7%BA%B3%E6%A0%91&rev=1610963299](https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:%E6%9C%80%E5%B0%8F%E6%96%AF%E5%9D%A6%E7%BA%B3%E6%A0%91&rev=1610963299)

Last update: 2021/01/18 17:48