基础图论

——鲍佳

主要内容

（一），最短路
（二），最小生成树
（三），割点及割边
（四），LCA（最近公共祖先）
（五），无向图双连通分量
（六），有向图强连通分量

（一），最短路算法

1，Dijkstra 算法

Dijkstra 算法用于处理一类单源最短路问题，该算法适用于有向图和无向图

算法思想：

清除所有点的标记
设 d[s] = 0， 其他 d[i] = inf
循环 n 次
{
    在所有未标记的点中，选出 d 值最小的点 u
    给 u 标记
    对于从 u 出发的点 v，更新 d[v] = min{d[v] ， d[u]+cost(u，v)}
}

时间复杂度为 O(n^2)。
优化算法，可以用优先队列快速查询 d 值最小的 u，从而使算法达到 O(nlogm)

2，Bellman-Ford 算法

如果一个图存在负环，那么最短路更本不存在，甚至在求最短路的时候会出现死循环若一个图不含有负环，那么最短路最多只经过 n 个点，故可以通过 n 轮松弛操作求最短路，这样就避免了出现死循环的问题，**而且如果我们做了这 n 次松弛后，图还能松弛，那么表示这个图有负环**

算法思想：

d[s]=0，d[i] = inf
循环 n 次
{
对于每条边（u，v）更新 d[v] = min{d[v]，d[u]}
}
可以看出时间复杂度为 O(nm)

3，spfa 算法

Spfa 算法是 Bellman-Ford 算法的队列优化
代码

```
void spfa(int k){
    for (int i=0;i<=n;i++) dist2[i]=(i==k? 0 : 99999999);
    memset(vis,0,sizeof(vis));
    q1.push(k);
    while(!q1.empty())
    {
        int x=q.front();
        q.pop();
        vis[x]=false;
        for (int i=0;i<map[x].size();i++)
        {
            if(dist[map[x][i].id]>dist[x]+map[x][i].cost)
            {
                dist[map[x][i].id]=ist[x]+map[x][i].cost;
                if(!vis[map[x][i].id])
                {
                    vis[map[x][i].id]=true;
                    q.push(map[x][i].id);
                }
            }
        }
    }
}
```

最坏情况下时间复杂度还是 O(nm)，但是实际中远小于这个复杂度

4，Floyd 算法

这个算法基于 dp 的思想对于两条边(u，k)，(k，v)，dp[u][v] = min(dp[u][v]，dp[u][k]+dp[k][v])

代码

```
void Floyd()
{
    for (int k=0;k<=n;k++)
    for (int i=0;i<=n;i++)
    for (int j=0;j<=n;j++)
    dist1[i][j]<?=dist1[i][k]+dist1[k][j];
}
```

（二），最小生成树

1，prim 算法

算法思想：

从任意一起点 u 出发，将其加入最小生成树中，每次选出没有在生成树中的点 v，且 v 到这个生成树的距离最小，将 v 加入最小生成树中，用 v 去更新其他还没有加入生成树的点的距离。

时间复杂度为 O(n^2)

2，kruskal 算法

算法思想

将所有的边按边的权值排序，依次查看每条边，若当前边的两个端点 u，v 不在同一个集合，则将这条边加入最小生成树中，并合并这两个集合

时间复杂度不知，主要是不知道怎么快速判断两个点是否在同一集合及怎么合并两个集合，有一个很好的方法就是用并查集，这样就能将这个算法优化到 O(mlogm)，对于稀疏图，这个算法是相当优秀的

代码

```
struct node
{
    int left,right,cost;
}road[MAX];
```

```
bool cmp(node x,node y){return x.cost<y.cost;}
int p[MAX],m,n;
int find(int x){return x==p[x]? x:p[x]=find(p[x]);}
int kruskal()
{
    int ans=0;
    for (int i=0;i<=n;i++) p[i]=i;
    for (int i=0;i<m;i++)
    {
        int x=find(road[i].left);
        int y=find(road[i].right);
        if(x!=y)
        {
            ans+=road[i].cost;
            p[x]=y;
        }
    }
    return ans;
}
```

（三），割点及割边

在谈割点之前，先看看什么叫一棵树的 dfs 系列。对于 dfs，可以说就是一个栈的应用，每次都是将搜索对象入栈，处理完后将其出栈，这样我们在 dfs 时就可以记录下每个点的入栈和出栈的时间

```
void dfs(int v)
{
    In[v] = tot++;
    操作
    Out[v]= tot++
}
```

这样就发现 dfs 系列有很大用途，可以知道一个点 u 的所有子孙的 dfs 值都在(in[u],out[u])之间，而且对于 dfs 系列的一段，它是两个端点的一条路径

割点的定义：

在一个无向图中，如果删除一个点 u，连通分量数目增加，那么这个点 u 就是割点
同理可以定义割边

定理

在无向连通图 G 的 dfs 树中，非根结点 u 是 G 的割点当且仅当 u 存在一个子结点 v，使得 v 及其所有后代都没有反向边连回 u 的祖先

对于一条边（u，v），若 v 及其子孙的反向边不能到达 v 的祖先，那么这条边就是割边，

算法思想

基于上面的定理及 dfs 系列，可以得到割点及割边的算法，

设 low[u]表示 u 及其后代所能连回的最早祖先的 in 值，则定理中的条件为 low[u]>=in[u]，若 low[v]>in[u]，则（u，v）便是割边

代码

```
struct CUT_V
{
    static const int maxn=10000+10;
    int dfs_clock,low[maxn],n,m,pre[maxn],sumcut;
    vector<int>group[maxn];
    bool iscut[maxn];

    void init()
    {
        for (int i=0;i<=n;i++) group[i].clear();
        memset(iscut,0,sizeof(iscut));
        memset(pre,0,sizeof(pre));
        sumcut=0;    dfs_clock=0;
    }

    void addedge(int u,int v)
    {
        group[u].push_back(v);
        group[v].push_back(u);
    }

    int dfs(int u,int fa)
    {
        int lowu=pre[u]=++dfs_clock;
        int child=0;
        for (int i=0;i<group[u].size();i++)
        {
            int v=group[u][i];
            if(!pre[v])
            {
```

```
                child++;
                int lowv=dfs(v,u);
                lowu=min(lowu,lowv);
                if (lowv>=pre[u]) iscut[u]=true;
            }
            else if(pre[v]<pre[u] && v!=fa) lowu=min(lowu,pre[v]);
        }
        if (fa==-1 && child==1) iscut[u]=false;
        low[u]=lowu;
        return lowu;
    }

    int get_sum()
    {
        int ans=dfs(1,-1);
        for (int i=1;i<=n;i++) if (iscut[i]) sumcut++;
        return sumcut;
    }
};
```

对于割边，只需稍加修改便可得到

（四），LCA（最近公共祖先）

LCA 是基于树来说的。

基于 RMQ 思想的 LCA：

这个算法只适用于静态的树（不改变树结构）

设 anc[u][k] 表示 u 的第 2^k 个祖先，可以知道 d[u][0] 表示的就是 u 的父亲，我们只需要 dfs 一边，求出每个结点 u 的父亲 fa[u]，便可在线性时间内求出所有的 anc 值

代码如下

```
void RMQ(int n){
    memset(anc,0,sizeof(anc));
    for (int i=1;i<=n;i++) anc[i][0]=fa[i];
    for (int k=1;k<=20;k++)
        for (int v=1;v<=n;v++)
            anc[v][k]=anc[anc[v][k-1]][k-1];
}
```

若要查询 u 和 v 的 LCA，首先看看 u 和 v 的深度是否相同，不相同的话把深度大的移到和深度小的同一深度

代码如下

```
int swim(int x,int H)
{
    for (int i=0;H>0;i++) {if (H&1) x=anc[x][i]; H/=2; }
    return x;
}
```

再利用 anc 来求出 LCA，具体过程在下面代码中

代码如下

```
int LCA(int u,int v)
{
    int k;
    if (dep[u]>dep[v]) swap(u,v);
    v=swim(v,dep[v]-dep[u]);
    if (u==v) return v;
    while(1)
    {
        for (k=0;anc[u][k]!=anc[v][k];k++);
        if (k==0) return anc[u][0];
        u=anc[u][k-1];
        v=anc[v][k-1];
    }
}
```

这样就得到了一个求 LCA 的一个在线算法，当然有利用 dfs 思想的 Tarjan 算法和基于动态树的求法

（五），无向图双连通分量

定义

点双连通分量；

对于一个连通图，如果任意两点存在两条**点不重复**的路径，则说这个图是一个双连通的，点双连通的极大连通子图称为双连通分量

同理可定义边双连通分量

可以知道，每条边恰好属于一个双连通分量，不同的双连通分量可能会有公共点，公共点一定是割点。

算法思想

点双连通分量：

在求割点的时候就可以顺便求出点双连通分量，用一个栈来储存 dfs 时的边，在搜索的时候，没碰到一条边时，就将这条边加入栈中，若出现 low[v]>=in[u]时，说明 u 是一个割点，这个时候，依次把栈顶中的边取出，直到碰到边（u，v），取出的这些边及相关的点就构成一个双连通分量。具体看下面代码

```
int dfs(int u,int fa)
{
    int lowu=pre[u]=++dfs_clock;
    int child=0;
    for (int i=0;i<group[u].size();i++)
    {
        int v=group[u][i];
        Edge e=(Edge){u,v};
        if (!pre[v])
        {
            S.push(e);
            child++;
            int lowv=dfs(v,u);
            lowu=min(lowu,lowv);
            if (lowv>=pre[u])
            {
                iscut[u]=true;
                bcc_cnt++;
                bcc[bcc_cnt].clear();
                while (1)
                {
                    Edge x=S.top(); S.pop();
                    if (bccno[x.from]!=bcc_cnt)
                    {
                        bcc[bcc_cnt].push_back(x.from);
                        bccno[x.from]=bcc_cnt;
                    }
                    if (bccno[x.to]!=bcc_cnt)
                    {
                        bcc[bcc_cnt].push_back(x.to);
                        bccno[x.to]=bcc_cnt;
                    }
```

```
                    if (x.from==u && x.to==v) break;
                }
            }
        }
        else if (pre[v]<pre[u] && v!=fa)
        {
            S.push(e);
            lowu=min(pre[v],lowu);
        }
    }
    if(fa<0 && child==1) iscut[u]=false;
    return lowu;
}
```

边双连通分量

求出所有割边，把割边去掉，剩下的就是一个个边双连通分量

构造双连通图

给你一个无向图，问至少加多少条边，可以使它成为一个双连通图

做法：

求出所有割边，去掉割边，则剩下的都是双连通的，把每一个双连通缩成一个点，把割边加回来，这样就得到一颗树，对于一颗树，要把它变成双连通的，只需要看他的叶子个数 leaf，则最少需要加的边数为（leaf+1）/2 条

（六），有向图强连通分量

定义

在一个有向图中，若没两个点都能相互到达，则称这个图为强连通的，对于一个有向图，每一个极大强连通图就叫这个图的一个强连通分量

算法思想

同样是利用 dfs 系列，用一个栈来维护强连通分量里的点，当得到 low[u]==in[u]时，重栈中依次取出每个点，直到碰到 u，则这些点就构成了一个强连通分量，具体看代码

代码

```
void dfs(int u)
{
	pre[u]=lowlink[u]=++dfs_clock;
	S.push(u);
	for (int i=0;i<group[u].size();i++)
	{
		int v=group[u][i];
		if (!pre[v])
		{
			dfs(v);
			lowlink[u]=min(lowlink[u],lowlink[v]);
		}
		else if (!sccno[v])
		{
			lowlink[u]=min(lowlink[u],pre[v]);
		}
	}
	if (lowlink[u]==pre[u])
	{
		scc_cnt++;
		scc[scc_cnt].clear();
		while (1)
		{
			int x=S.top();
			S.pop();
			scc[scc_cnt].push_back(x);
			sccno[x]=scc_cnt;
			if (x==u) break;
		}
	}
}
```