生成树

Spanning Tree

Kruskal

Idea：贪心。每次找最小边权的边，若其两端点都不在当前的树中，则将该边加入生成树中。用并查集来维护端点是否在树中。

Complexity：$O(E\lg E+E\alpha(V))$ 【路径压缩+启发式合并】

Code：

```
struct Edge{
    int u, v, dis;
    bool operator < (const Edge &a) const{
        return dis < a.dis;
    }
}edge[M<<1];

int n, m, x, y, z, fa[N], ans;

int findfa(int x){
    if(x != fa[x])  fa[x] = findfa(fa[x]);
    return fa[x];
}
void unionn(int x, int y){
    fa[findfa(y)] = findfa(x);
}

int main(){
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i++)
        fa[i] = i;
    for(int i = 1; i <= m; i++)
        scanf("%d%d%d", &edge[i].u, &edge[i].v, &edge[i].dis);
    sort(edge + 1, edge + m + 1);
    int cnt = 0;
    for(int i = 1; i <= m; i++){
        if(findfa(edge[i].u) != findfa(edge[i].v)){
            unionn(edge[i].u, edge[i].v);
            cnt++;
            ans += edge[i].dis;
            if(cnt == n - 1)
                break;
        }
    }
    if(cnt != n - 1)
        return puts("orz"), 0;
    printf("%d\n", ans);
    return 0;
}
```

Prim

Idea：维护集合 S，凡在集合 S 中的点都已经纳入已有生成树中。每次选取距离**集合** S 的最近的不在 S 中的点加入 S，并更新与之相连的所有点的距离。（类比 $\textbf{Dijkstra}$）

Complexity：

- $O(V^2+E)$ 【朴素实现】
- $O((V+E)\lg V)$ 【小根堆实现（随时删除旧节点）】
- $O((V+E)\lg E)$ 【优先队列实现】
- $O(E+V\lg V)$ 【斐波那契堆实现】

Code：

```
struct Node{
    LL dis;
    int num;
    bool operator < (const Node &a) const{
        return a.dis < dis;
    }
};

LL dis[N];
bool inS[N];
void prim(){
    for(int i = 1; i <= n; i++){
        dis[i] = INF;
        inS[i] = 0;
    }
    priority_queue<Node> q;
    dis[1] = 0;
    q.push( (Node){0, 1} );
    while(!q.empty()){
        Node cur = q.top();
        q.pop();
        if(inS[cur.num])    continue;
        inS[cur.num] = 1;
        ans += cur.dis;
        for(int i = head[cur.num]; i; i = edge[i].nxt){
            if(dis[edge[i].to] > edge[i].len){
                dis[edge[i].to] = edge[i].len;
                q.push( (Node){dis[edge[i].to], edge[i].to} );
            }
        }
    }
}
```

Boruvka

Idea：假设当前最小生成森林的边集为 E，形成了一些**连通块**，定义一个连通块的**最小边**是它连向其他连通块的边中权值最小的那条。每一轮我们把所有最小边加入 E，然后更新新的连通块的最小边。当所有连通块都没有最小边时找到了最小生成树（森林）。

Complexity：$O((V+E)\lg V)$

Code：

```
int fa[N];
int findfa(int x){ return x == fa[x] ? x : fa[x] = findfa(fa[x]); }
void unionn(int x, int y){ fa[findfa(y)] = findfa(x); }

int mndis[N], mark[N];
vector<int> ans; // ans stores edges in MST
void Boruvka(){
    for(int i = 1; i <= n; i++) fa[i] = i;
    while(1){
        vector<bool> vis(m+5);
        for(int i = 1; i <= n; i++) mndis[i] = INF, mark[i] = 0;
        for(int i = 1; i <= m; i++){
```

```
            if(findfa(edge[i].u) == findfa(edge[i].v))  continue;
            if(mndis[findfa(edge[i].u)] > edge[i].dis){
                mndis[findfa(edge[i].u)] = edge[i].dis;
                mark[findfa(edge[i].u)] = i;
            }
            if(mndis[findfa(edge[i].v)] > edge[i].dis){
                mndis[findfa(edge[i].v)] = edge[i].dis;
                mark[findfa(edge[i].v)] = i;
            }
        }
        bool ok = true;
        for(int i = 1; i <= n; i++){
            if(findfa(i) != i)  continue;
            if(mark[i] && !vis[mark[i]]){
                ok = false;
                ans.pb(mark[i]);
                unionn(edge[mark[i]].u, edge[mark[i]].v);
                vis[mark[i]] = true;
            }
        }
        if(ok)  break;
    }
}
```

次小生成树

Idea：用 $\mathbf{Kruskal}$ 或 $\mathbf{Prim}$ 算法得到最小生成树后，枚举未出现在最小生成树中的边，添加这条边后，树上会形成一个环，把该环中最大的边删去，即得到**非严格**次小生成树；倘若求**严格**次小生成树，则需要记录环中最大边和严格小于最大边的**次大边**，当最大边与枚举的非树边相等时，删去次大边。

实现方法采用倍增法求 LCA，每次求解非树边两端点的 LCA，同时维护最大边权与次大边权。

ATT：我的倍增法求 LCA 最后一步并没有走到 LCA 处（`fa[x][0]` 和 `fa[y][0]` 才是 LCA），所以维护边权信息的时候不要忘了最后还有的这两条边。

Code（严格次小生成树）：

```
#include<cstdio>
#include<algorithm>

using namespace std;

typedef long long LL;

const int N = 100005;
const int M = 300005;

int n, m, rt;
LL sum, ans = 1e16;

struct LCA{
    struct Edge{
        int nxt, to, dis;
    }edge[M<<1];
    int head[N], edgeNum;
    void addEdge(int from, int to, int dis){
        edge[++edgeNum] = (Edge){head[from], to, dis};
        head[from] = edgeNum;
    }

    int fa[N][25], dep[N];
    LL mx1[N][25], mx2[N][25];
    void dfs(int x, int f, int depth){
```

```
        dep[x] = depth, fa[x][0] = f;
        for(int i = head[x]; i; i = edge[i].nxt){
            if(edge[i].to == f) continue;
            dfs(edge[i].to, x, depth+1);
            mx1[edge[i].to][0] = edge[i].dis;
            mx2[edge[i].to][0] = 0;
        }
    }
    void init(){
        for(int j = 1; (1 << j) <= n; j++){
            for(int i = 1; i <= n; i++){
                if(fa[i][j-1]){
                    fa[i][j] = fa[fa[i][j-1]][j-1];
                    mx1[i][j] = max(mx1[i][j-1], mx1[fa[i][j-1]][j-1]);
                    mx2[i][j] = max(mx2[i][j-1], mx2[fa[i][j-1]][j-1]);
                    if(mx1[i][j-1] != mx1[fa[i][j-1]][j-1])
                        mx2[i][j] = max(mx2[i][j], min(mx1[i][j-1], mx1[fa[i][j-1]][j-1]));
                }
            }
        }
    }
    inline void update(LL mx1, LL mx2, LL &res1, LL &res2){
        if(res1 < mx1)  res2 = max(res1, mx2), res1 = mx1;
        else if(res1 == mx1)    res2 = max(res2, mx2);
        else    res2 = max(res2, mx1);
    }
    int lca(int x, int y, LL &max1, LL &max2){
        if(dep[x] < dep[y]) swap(x, y);
        for(int i = 20; i >= 0; i--){
            if(dep[x] - (1 << i) >= dep[y]){
                update(mx1[x][i], mx2[x][i], max1, max2);
                x = fa[x][i];
            }
        }
        if(x == y)  return x;
        for(int i = 20; i >= 0; i--){
            if(fa[x][i] && fa[x][i] != fa[y][i]){
                update(mx1[x][i], mx2[x][i], max1, max2);
                update(mx1[y][i], mx2[y][i], max1, max2);
                x = fa[x][i], y = fa[y][i];
            }
        }
        update(mx1[x][0], mx2[x][0], max1, max2);
        update(mx1[y][0], mx2[y][0], max1, max2);
        return fa[x][0];
    }
}lca;

struct Edge{
    int u, v, dis;
    bool inMST;
    bool operator < (const Edge &A) const{ return dis < A.dis; }
}edge[M];

int fa[N];
int findfa(int x){ return x == fa[x] ? x : fa[x] = findfa(fa[x]); }
inline void unionn(int x, int y){ fa[findfa(y)] = findfa(x); }

void Kruskal(){
    for(int i = 1; i <= n; i++) fa[i] = i;
    sort(edge+1, edge+m+1);
    int cnt = 0;
    for(int i = 1; i <= m; i++){
        if(findfa(edge[i].u) == findfa(edge[i].v))  continue;
        unionn(edge[i].u, edge[i].v);

        lca.addEdge(edge[i].u, edge[i].v, edge[i].dis);
```

```
            lca.addEdge(edge[i].v, edge[i].u, edge[i].dis);
            edge[i].inMST = true;
            if(!rt) rt = edge[i].u;

            sum += edge[i].dis;
            cnt++;
            if(cnt == n - 1)    break;
        }
    }

int main(){
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= m; i++)
        scanf("%d%d%d", &edge[i].u, &edge[i].v, &edge[i].dis);
    Kruskal();

    lca.dfs(rt, 0, 1);
    lca.init();

    for(int i = 1; i <= m; i++){
        if(edge[i].inMST)   continue;
        LL mx1 = 0, mx2 = 0;
        lca.lca(edge[i].u, edge[i].v, mx1, mx2);
        if(edge[i].dis > mx1)   ans = min(ans, sum - mx1 + edge[i].dis);
        else    ans = min(ans, sum - mx2 + edge[i].dis);
    }
    printf("%lld\n", ans);
    return 0;
}
```

瓶颈生成树

Definition：所有生成树中，最大边权最小的生成树称为瓶颈生成树。

Theorem：最小生成树是瓶颈生成树的充分不必要条件，即最小生成树一定是瓶颈生成树，而瓶颈生成树不一定是最小生成树。

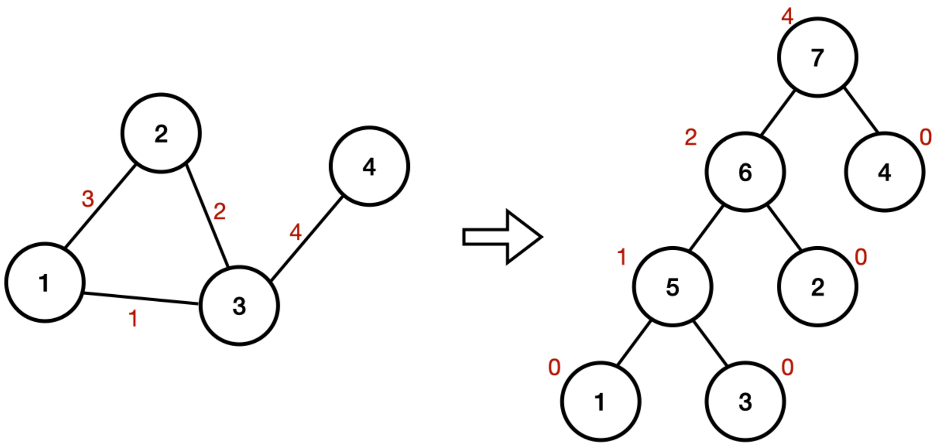
最小瓶颈路

Definition：从 x 到 y 的最小瓶颈路是所有从 x 到 y 的简单路径中最大边权最小的路径。

Theorem：最小生成树上从 x 到 y 的路径一定是一条最小瓶颈路（但是最小瓶颈路上的路径不一定在任何一棵最小生成树上）

Kruskal 重构树

Definition：在执行 $\textbf{Kruskal}$ 算法的过程中，我们会从小到大依次加边。首先新建 n 个集合，每个集合恰有一个节点，点权为 0。每一次加边会合并两个集合，我们可以新建一个点，点权为加入边的边权，同时将两个集合的根节点分别设为新建点的左儿子和右儿子。然后我们将两个集合和新建点合并成一个集合。将新建点设为根。如此，在进行 $n-1$ 轮之后我们得到了一棵恰有 n 个叶子的二叉树，同时每个非叶子节点恰好有两个儿子。这棵树就叫 $\textbf{Kruskal}$ 重构树。（摘自 oi-wiki）



Properties:

- $\textbf{Kruskal}$ 重构树上任一节点权值 $\geqslant$ 其子节点权值，类似大根堆。
- 最小生成树上 x 到 y 路径中的最大值等于 $\textbf{Kruskal}$ 重构树上 x 和 y 的 LCA 的点权。

  换句话说，到 x 的简单路径的最大边权 $\leqslant val$ 的所有点 y 都在 $\textbf{Kruskal}$ 重构树的某一棵子树内，这个子树的根节点是 x 到根的路径上最浅的权值 $\leqslant val$ 的点。

ATT：我的代码中，$\textbf{Kruskal}$ 重构树的大小为 $rt = 2n - 1$，且以 rt 为根节点。注意在之后的操作中把 n 换成 rt。

Code：

```cpp
int n, m, rt; // rt is the root of 'Kruskal Tree'

int val[N]; // points' value of 'Kruskal Tree'
struct Edge{
    int nxt, to;
}edge[N<<1];
int head[N], edgeNum;
void addEdge(int from, int to){
    edge[++edgeNum] = (Edge){head[from], to};
    head[from] = edgeNum;
}

namespace MST{
    int fa[N];
    int findfa(int x){ return x == fa[x] ? x : fa[x] = findfa(fa[x]); }
    inline void unionn(int x, int y){ fa[findfa(y)] = findfa(x); }

    struct Edge{
        int u, v, dis;
        bool operator < (const Edge &A) const{ return dis < A.dis; }
    }edge[M];
    void Kruskal(){
        rt = n;
        for(int i = 1; i <= n * 2; i++) fa[i] = i; // pay attention to *2
        sort(edge+1, edge+m+1);
        int cnt = 0;
        for(int i = 1; i <= m; i++){
            if(findfa(edge[i].u) == findfa(edge[i].v))  continue;

            // build the tree:
            ++rt;
            addEdge(rt, findfa(edge[i].u)), addEdge(findfa(edge[i].u), rt);
            addEdge(rt, findfa(edge[i].v)), addEdge(findfa(edge[i].v), rt);
            unionn(rt, edge[i].u), unionn(rt, edge[i].v);
            val[rt] = edge[i].dis;

            cnt++;
```

```
            if(cnt == n - 1)    break;
        }
    }
}

int main(){
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= m; i++)
        scanf("%d%d%d", &MST::edge[i].u, &MST::edge[i].v, &MST::edge[i].dis);
    MST::Kruskal();

    // now the Kruskal tree is rooted at rt (rt == 2n-1)
    // do something...

    return 0;
}
```

矩阵树定理 Matrix-Tree Theorem

无向图情形

定义**度数矩阵**：

$$D_{ij}=\begin{cases}\deg(i)&i=j\\0&i\neq j\end{cases}$$

设 $\#e(i,j)$ 表示 i 和 j 之间的边数，定义**邻接矩阵**：

$$A_{ij}=A_{ji}=\begin{cases}0&i=j\\\#e(i,j)&i\neq j\end{cases}$$

定义 **Laplace 矩阵**（亦称 **Kirchhoff 矩阵**）：

$$L_{ij}=D_{ij}-A_{ij}$$

矩阵树定理（无向图行列式形式）：对于任意 i，都有：

$$\text{生成树个数}=\det L\begin{pmatrix}1,2,\cdots,i-1,i+1,\cdots,n\\1,2,\cdots,i-1,i+1,\cdots,n\end{pmatrix}$$

其中，$L\begin{pmatrix}1,2,\cdots,i-1,i+1,\cdots,n\\1,2,\cdots,i-1,i+1,\cdots,n\end{pmatrix}$ 表示除去 L 的第 i 行和第 i 列后构成的子矩阵。

也就是说，**Laplace** 矩阵的任意 $n-1$ 阶主子式都相等，且等于生成树个数。

矩阵树定理（无向图特征值形式）：设 $\lambda_1,\lambda_2,\cdots,\lambda_{n-1}$ 为 L 的 $n-1$ 个非零特征值，那么有：

$$\text{生成树个数}=\frac{1}{n}\lambda_1\lambda_2\cdots\lambda_{n-1}$$

有向图情形

定义**出度矩阵**和**入度矩阵**：

$$D^{\text{out}}_{ij}=\begin{cases}\deg^{\text{out}}(i)&i=j\\0&i\neq j\end{cases}\quad,\quad D^{\text{in}}_{ij}=\begin{cases}\deg^{\text{in}}(i)&i=j\\0&i\neq j\end{cases}$$

设 $\#e(i,j)$ 表示 i 和 j 之间的边数，定义**邻接矩阵**：

$$A_{ij}=\begin{cases}0&i=j\\\#e(i,j)&i\neq j\end{cases}$$

定义出度和入度的 **Laplace 矩阵**：

$$L^{\text{out}}_{ij}=D^{\text{out}}_{ij}-A_{ij}\quad,\quad L^{\text{in}}_{ij}=D^{\text{in}}_{ij}-A_{ij}$$

矩阵树定理（有向图根向形式）：

$$\text{以 } k \text{ 为根的根向树形图个数} = \det L^{\text{out}} \begin{pmatrix} 1, 2, \cdots, k-1, k+1, \cdots, n \\ 1, 2, \cdots, k-1, k+1, \cdots, n \end{pmatrix}$$

矩阵树定理（有向图叶向形式）：

$$\text{以 } k \text{ 为根的叶向树形图个数} = \det L^{\text{in}} \begin{pmatrix} 1, 2, \cdots, k-1, k+1, \cdots, n \\ 1, 2, \cdots, k-1, k+1, \cdots, n \end{pmatrix}$$