图论

二分图

最大匹配HK

```
//前向星  n=1e5的简单图也可以跑  左右分开的 不需要i+n
const int N=2e5 + 7, inf = 0x3f3f3f3f;
int tot,head[N];
struct node {int nxt, to;}e[N<<1];
void addedge(int u, int v) {
    e[++tot].to=v,e[tot].nxt=head[u];
    head[u]=tot;
}
int Mx[N],My[N],dx[N],dy[N],dis,uN;  //左边的点集  右边无所谓图建好就行
bool used[N];
bool bfs() {
    queue<int>q;
    dis=inf;
    memset(dx,-1,sizeof(dx));
    memset(dy,-1,sizeof(dy));
    for(int i=1;i<=uN;++i) if(Mx[i]==-1) q.push(i),dx[i]=0;
    while(!q.empty()) {
        int u=q.front(); q.pop();
        if(dx[u]>dis) break;
        for(int i=head[u];i;i=e[i].nxt) {
            int v=e[i].to;
            if (dy[v]==-1) {
                dy[v]=dx[u]+1;
                if(My[v]==-1) dis=dy[v];
                else {
                    dx[My[v]]=dy[v]+1;
                    q.push(My[v]);
                }
            }
        }
    }
    return dis!=inf;
}
bool dfs(int u) {
    for(int i=head[u];i;i=e[i].nxt) {
        int v=e[i].to;
        if(!used[v] && dy[v]==dx[u]+1) {
            used[v]=1;
            if(My[v]!=-1 && dy[v]==dis) continue;
            if(My[v]==-1 || dfs(My[v])) {
                My[v]=u; Mx[u]=v;
```

```
                    return 1;
                }
            }
        }
        return 0;
}
int hk() {
    int res=0;
    memset(Mx,-1,sizeof(Mx)); memset(My,-1,sizeof(My));
    while(bfs()) {
        memset(used, 0, sizeof(used));
        for(int i=1;i<=uN;++i) if(Mx[i]==-1 && dfs(i)) ++res;
    }
    return res;
}
void init() {
    memset(head,0,sizeof(head));
    tot=0;
}
```

匈牙利

```
const int maxn = 505; // O(n^3)
bool vis[maxn];
vector<int> g[maxn];
int link[maxn];
bool dfs(int u) {
    for (int i = 0; i < g[u].size(); ++i) {
        int v = g[u][i];
        if (!vis[v]) {
            vis[v] = true;
            if (!link[v] || dfs(link[v])) {
                link[v] = u;
                return true;
            }
        }
    }
    return false;
}
int hungary() {
    memset(link, 0, sizeof(link));
    int ans = 0;
    for (int i = 1; i <= n; i++) {
        memset(vis, false, sizeof(vis));
        if (dfs(i)) ans++;
    }
    return ans;
}
```

最大权完备KM匹配

- 2019南京J match是右边连向左边

```
const int maxn = 505, inf = 1000;  //注意inf的设置 一个不可能达到的权值即可
int n, m, match[maxn], pre[maxn];  // n为完备匹配的点数 左右分开的 不需要i+n
int mp[maxn][maxn];
int slack[maxn], ex[maxn], ey[maxn];
bool vis[maxn];
void bfs(int u) {
    int x, y = 0, yy = 0;
    int delta;
    for (int i = 1; i <= n; ++i) pre[i] = 0, vis[i] = 0, slack[i] = inf;
    match[y] = u;
    while (1) {
        x = match[y];
        delta = inf;
        vis[y] = 1;
        for (int i = 1; i <= n; i++) {
            if (vis[i]) continue;
            if (slack[i] > ex[x] + ey[i] - mp[x][i]) {
                slack[i] = ex[x] + ey[i] - mp[x][i];
                pre[i] = y;
            }
            if (slack[i] < delta) {
                delta = slack[i];
                yy = i;
            }
        }
        for (int i = 0; i <= n; i++) {
            if (vis[i])
                ex[match[i]] -= delta, ey[i] += delta;
            else slack[i] -= delta;
        }
        y = yy;
        if (match[y] == -1) break;
    }
    while (y) {
        match[y] = match[pre[y]];
        y = pre[y];
    }
}
int KM(){  //注意最大权是否开ll
    for (int i = 1; i <= n; ++i) ex[i] = ey[i] = 0, match[i] = -1;
    for (int i = 1; i <= n; i++) bfs(i);
    int res = 0;
    for (int i = 1; i <= n; i++)
        if (match[i] != -1) res += mp[match[i]][i];
    return res;
```

```
}
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++) mp[i][j] = -inf;
    for (int i = 1; i <= m; i++) {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        mp[u][v] = w;
    }
    printf("%lld\n", KM());
    for (int i = 1; i <= n; i++) printf("%lld ", matched[i]);
    printf("\n");
}
```

生成树

Kruskal、prim

完全图最小异或生成树

```
const int N = 4000005;
int n, rt = 0;
int bin[30], sz[N], ls[N], rs[N], totnode = 0;
ll ans;
void insert(int &x, int v, int d) { //从高到低 前缀字典树 1右 0左
    if (!x) x = ++totnode;
    if (d == -1) {
        sz[x] = 1;
        return;
    }
    if (bin[d] & v) insert(rs[x], v, d - 1);
    else insert(ls[x], v, d - 1);
    sz[x] = sz[ls[x]] + sz[rs[x]];
}
int query(int x, int v, int d) {//左边到底了 对右边统计
    if (d == -1) return 0;
    if (v & bin[d]) { //此前统计这层为1
        if (rs[x]) return query(rs[x], v, d - 1);  //如果右侧同样为1 可以无贡献加入
        return query(ls[x], v, d - 1) + bin[d];
    } else { //同0
        if (ls[x]) return query(ls[x], v, d - 1);
        return query(rs[x], v, d - 1) + bin[d];
    }
}
int merge(int x, int y, int d, int val, int sd) { //固定右子树y  对左子树0递归
    if(d == -1) return query(y, val, sd);  //到底了 对开始统计右子树
    int ret = 1 << 30;
    if(ls[x]) ret = min(ret, merge(ls[x], y, d - 1, val, sd));  // 0和0 合并
```

```
    if(rs[x]) ret = min(ret, merge(rs[x], y, d - 1, val + bin[d], sd));  // 0和1 合并
    return ret;
}
void dfs(int x, int d) {
    if (d == -1) return;
    if (ls[x]) dfs(ls[x], d - 1);
    if (rs[x]) dfs(rs[x], d - 1);
    if (!ls[x] || !rs[x]) return;  //有一个是空的 可以内部消化
    int l = ls[x], r = rs[x];
    // if(sz[l]>sz[r]) swap(l,r);
    ans += merge(l, r, d - 1, 0, d - 1) + bin[d];  //左右有 0 1  对左边的0递归
}
vector<pair<int, int> > g[maxn];
int val[maxn];
void dfs2(int u, int fa) {
    for (auto &it : g[u]) {
        int v = it.first, w = it.second;
        if (v == fa) continue;
        val[v] = val[u] ^ w;
        dfs2(v, u);
    }
}
int main() {
    bin[0] = 1;
    for (int i = 1; i <= 29; ++i) bin[i] = bin[i - 1] << 1;
    scanf("%d", &n);
    for (int i = 0, u, v, w; i < n - 1; ++i) {
        scanf("%d%d%d", &u, &v, &w);
        g[u].push_back({v, w});
        g[v].push_back({u, w});
    }
    val[0] = 0;
    dfs2(0, -1);
    for (int i = 0; i < n; ++i) insert(rt, val[i], 29);
    dfs(rt, 29);
    printf("%lld\n", ans);
}
```

网络流

MCMF,SPFA

```
const int maxn = 50 + 7, inf = 0x3f3f3f3f;
struct Edge {
    int from, to, cap, flow;
    ll cost;
};
struct MCMF {
    int n, m, s, t;
```

```
    vector<Edge> edges;
    vector<int> G[maxn];
    int inq[maxn];
    ll d[maxn];   //最短路数组
    int p[maxn];  //记录路径
    int a[maxn];  //记录流量
    void init(int n) {
        this->n = n;
        for (int i = 0; i < n; i++) G[i].clear();
        edges.clear();
    }
    void addedge(int from, int to, int cap, int cost) {
        edges.push_back(Edge{from, to, cap, 0, cost});
        edges.push_back(Edge{to, from, 0, 0, -cost});
        m = edges.size();
        G[from].push_back(m - 2);
        G[to].push_back(m - 1);
    }
    bool spfa(int s, int t, int &flow, ll &cost) {
        for (int i = 0; i < n; i++) d[i] = inf;
        memset(inq, 0, sizeof(inq));
        d[s] = 0;
        inq[s] = 1;
        p[s] = 0;
        a[s] = inf;
        queue<int> q;
        q.push(s);
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            inq[u] = 0;
            for (int i = 0; i < G[u].size(); i++) {
                Edge &e = edges[G[u][i]];
                if (e.cap > e.flow && d[e.to] > d[u] + e.cost) {
                    d[e.to] = d[u] + e.cost;              //松弛
                    p[e.to] = G[u][i];                    //记录上一个点
                    a[e.to] = min(a[u], e.cap - e.flow);  //流量控制
                    if (!inq[e.to]) {
                        q.push(e.to);
                        inq[e.to] = 1;
                    }
                }
            }
        }
        if (d[t] == inf) return false;  //不存在最短路
        flow += a[t];
        cost += d[t] * a[t];
        int u = t;
        while (u != s) {
```

```
            edges[p[u]].flow += a[t];
            edges[p[u] ^ 1].flow -= a[t];
            u = edges[p[u]].from;
        }
        return true;
    }
    ll Mincost(int s, int t) {
        int flow = 0;
        ll cost = 0;
        while (spfa(s, t, flow, cost))
            ;
        if (flow == 0) cost = -1;
        return cost;
    }
} F;
```

Dinic

```
const int maxn = 2e3 + 7, inf = 0x3f3f3f3f;
struct Edge {
    int from, to, cap, flow;
};
struct Dinic {
    int n, m, s, t;
    vector<Edge> edges;   //边的信息
    vector<int> G[maxn];  //正反边的标号
    bool vis[maxn];
    int d[maxn];
    int cur[maxn];
    void init(int n) {
        this->n = n;
        for (int i = 0; i < n; i++) G[i].clear();
        edges.clear();
    }
    void addedge(int from, int to, int cap, int c = 0) ///注意加的是单向 如果双向 需要cap
==c
        edges.push_back(Edge{from, to, cap, 0});
        edges.push_back(Edge{to, from, c, 0});
        m = edges.size();
        G[from].push_back(m - 2);
        G[to].push_back(m - 1);
    }
    bool BFS() {
        memset(vis, 0, sizeof(vis));
        queue<int> q;
        q.push(s);
        d[s] = 0;
        vis[s] = 1;
        while (!q.empty()) {
```

```
            int u = q.front();
            q.pop();
            for (int i = 0; i < G[u].size(); i++) {
                Edge& e = edges[G[u][i]];
                if (!vis[e.to] && e.cap > e.flow) {
                    vis[e.to] = 1;
                    d[e.to] = d[u] + 1;
                    q.push(e.to);
                }
            }
        }
        return vis[t];
    }
    int DFS(int u, int dist) {
        if (u == t || dist == 0) return dist;
        int flow = 0, f;
        for (int& i = cur[u]; i < G[u].size(); i++) {
            Edge& e = edges[G[u][i]];
            if (d[u] + 1 == d[e.to] &&
                (f = DFS(e.to, min(dist, e.cap - e.flow))) > 0) {
                e.flow += f;
                edges[G[u][i] ^ 1].flow -= f;
                flow += f;
                dist -= f;
                if (!dist) break;
            }
        }
        return flow;
    }
    int Maxflow(int s, int t) {
        this->s = s;
        this->t = t;
        int flow = 0;
        while (BFS()) {
            memset(cur, 0, sizeof(cur));
            flow += DFS(s, inf);
        }
        return flow;
    }
} net;
```

MCMF,dj,势函数？可跑负权,O(n^2logn)

```
struct edge {
    int to, cap, cost, rev;
    edge() {}
    edge(int to, int _cap, int _cost, int _rev)
        : to(to), cap(_cap), cost(_cost), rev(_rev) {}
};
```

```
int V, H[maxn], dis[maxn], PreV[maxn], PreE[maxn];
vector<edge> G[maxn];
void init(int n) {
    V = n;
    for (int i = 0; i <= V; ++i) G[i].clear();
}
void AddEdge(int from, int to, int cap, int cost) {
    G[from].push_back(edge(to, cap, cost, G[to].size()));
    G[to].push_back(edge(from, 0, -cost, G[from].size() - 1));
}
int Min_cost_max_flow(int s, int t, int f, int& flow) {
    int res = 0;
    fill(H, H + 1 + V, 0);
    while (f) {
        priority_queue<pair<int, int>, vector<pair<int, int>>,
                       greater<pair<int, int>>>
            q;
        fill(dis, dis + 1 + V, inf);
        dis[s] = 0;
        q.push(pair<int, int>(0, s));
        while (!q.empty()) {
            pair<int, int> now = q.top();
            q.pop();
            int v = now.second;
            if (dis[v] < now.first) continue;
            for (int i = 0; i < G[v].size(); ++i) {
                edge& e = G[v][i];
                if (e.cap > 0 && dis[e.to] > dis[v] + e.cost + H[v] - H[e.to]) {
                    dis[e.to] = dis[v] + e.cost + H[v] - H[e.to];
                    PreV[e.to] = v;
                    PreE[e.to] = i;
                    q.push(pair<int, int>(dis[e.to], e.to));
                }
            }
        }
        if (dis[t] == inf) break;
        for (int i = 0; i <= V; ++i) H[i] += dis[i];
        int d = f;
        for (int v = t; v != s; v = PreV[v])
            d = min(d, G[PreV[v]][PreE[v]].cap);
        f -= d;
        flow += d;
        res += d * H[t];
        for (int v = t; v != s; v = PreV[v]) {
            edge& e = G[PreV[v]][PreE[v]];
            e.cap -= d;
            G[v][e.rev].cap += d;
        }
```

```
    }
    return res;
}
```

最短路

dijkstra 统计到某点最短路方案数，计算两点延最短路不相遇方案数

```
    dijkstra(st, dis[0], vis[0], ans[0]);
    dijkstra(ed, dis[1], vis[1], ans[1]);
    ll path_num = ((ans[0][ed] % MOD) * (ans[1][st] % MOD)) % MOD;
    ll D = dis[0][ed];
    for (int i = 1; i <= n; i++){
        if (dis[0][i] == dis[1][i] && D == dis[1][i] * 2){
            path_num = path_num - ((ans[0][i] * ans[0][i] % MOD) * (ans[1][i] * ans[1]
[i] % MOD)) % MOD;
            path_num = (path_num + MOD) % MOD;
        }
        for (int u = head[i]; ~u; u = e[u].next){
            int w = e[u].w;
            int v = e[u].v;
            ll d1 = dis[0][i], d2 = dis[1][v];
            if (d1 + w + d2 == D && d1 * 2 < D && d2 * 2 < D){
                path_num = path_num - ((ans[0][i] * ans[0][i] % MOD) * (ans[1][v] *
ans[1][v] % MOD)) % MOD;
                path_num = (path_num + MOD) % MOD;
            }
        }
    }
    cout << path_num << endl;
```

其他

强连通、点双、边双

```
int dfn[N],low[N],clk,stk[N],instk[N],tp,scc[N],sc,sz[N];
void tarjan(int u) {
    low[u]=dfn[u]=++clk,stk[++tp]=u,instk[u]=1;
    for(int i=head[u];i;i=e[i].nxt) {
        int v=e[i].to;
        if(!dfn[v]) {//树边
            tarjan(v);
            low[u]=min(low[u],low[v]);
        } else if(instk[v]) {//回边
            low[u]=min(low[u],dfn[v]);
        }//前向边和横叉边所在scc已处理完毕
    }
    if(dfn[u]==low[u]) {
```

```
        ++sc;
        int cur;
        do {
            cur=stk[tp--];
            scc[cur]=sc,sz[sc]++,instk[cur]=0;
        }while(cur!=u);
    }
}
typedef pair<int,int> pii;
const int N=1e5+7,M=1e6+7;
int n,m;
namespace G {
    struct edge {
        int nxt,to;
    }e[M];
  int head[N],tot;
  void init() {
    tot=0;
    memset(head,0,sizeof(head));
  }
  void add(int u,int v) {
        e[++tot].to=v,e[tot].nxt=head[u],head[u]=tot;
        e[++tot].to=u,e[tot].nxt=head[v],head[v]=tot;
  }
}
namespace VBCC {///vertex  cutnode
  int dfn[N],clk,low[N],bcc_cnt,iscut[N];
  vector<int>bcc[N];
  set<int>bccno[N];
  stack<pii>stk;
  int bcc_edge[N];
  void dfs(int u,int fa) {
    using namespace G;
    low[u]=dfn[u]=++clk;
    int child=0;
    for(int i=head[u];i;i=e[i].nxt) {
      int v=e[i].to;
      if(v==fa) continue;
      if(!dfn[v]) {
        stk.push(pii(u,v));
        dfs(v,u);
        ++child;
        low[u]=min(low[u],low[v]);
        if(low[v]>=dfn[u]) {
          iscut[u]=true;
          ++bcc_cnt;
          bcc[bcc_cnt].clear();
          while(1) {
            pii e=stk.top(); stk.pop();
```

```
            ++bcc_edge[bcc_cnt];
            if(bccno[e.first].count(bcc_cnt)==0) {
              bccno[e.first].insert(bcc_cnt);
              bcc[bcc_cnt].push_back(e.first);
            }
            if(bccno[e.second].count(bcc_cnt)==0) {
              bccno[e.second].insert(bcc_cnt);
              bcc[bcc_cnt].push_back(e.second);
            }
            if(e.first==u && e.second==v) break;
          }
        }
      } else if(dfn[v]<dfn[u]) {
        stk.push(pii(u,v));
        low[u]=min(low[u],dfn[v]);
      }
    }
    if(fa==-1 && child>1) iscut[u]=true;
  }
  void find_bcc() {
    memset(dfn,0,sizeof(dfn));
        memset(iscut,0,sizeof(iscut));
        memset(bcc_edge,0,sizeof(bcc_edge));
    clk=bcc_cnt=0;
    for(int i=0;i<N;++i) bccno[i].clear();
    for(int i=1;i<=n;++i) if(!dfn[i]) dfs(i,-1);
  }
}
namespace EBCC {///edge  cut_edge=bcc_cnt-1 割边就是端点所属不一样的边 注意重边的处理
  int dfn[N],low[N],clk,bcc_cnt,bccno[N],stk[N],tp;
  void dfs(int u,int fa) {
    using namespace G;
    dfn[u]=low[u]=++clk;
    stk[++tp]=u;
    for(int i=head[u];i;i=e[i].nxt) {
      int v=e[i].to;
      if(v==fa) continue;
      if(!dfn[v]) {
        dfs(v,u);
        low[u]=min(low[u],low[v]);
      } else low[u]=min(low[u],dfn[v]);
    }
    if(dfn[u]==low[u]) {
      ++bcc_cnt;
            int cur;
            do {
                cur=stk[tp--];
                bccno[cur]=bcc_cnt;
            }while(cur!=u);
```

```
        }
    }
    void find_bcc() {
      memset(dfn,0,sizeof(dfn));
          memset(low,0,sizeof(low));
      memset(bccno,0,sizeof(bccno));
      clk=bcc_cnt=0;
      for(int i=1;i<=n;++i) if(!dfn[i]) dfs(i,-1);
    }
}
```

圆方树

```
/*
    圆方树：割点将点双连接起来
    任意两点圆方树路径，等于原图两点简单路径集合
    圆方树可以给每个点赋值，方便计算
*/
typedef long long ll;
const int maxn = 3e5 + 10;
vector<int> G[maxn];  //原图
vector<int> T[maxn];  //圆方树图
int dfn[maxn], low[maxn], vis[maxn];
int n, m, tot, scc;      //记得scc从n+1开始
int Stack[maxn+100], pos;
ll val[maxn],sz[maxn],ans = 0;
ll num=0;  //当前树点数
void tarjan(int x) {
    dfn[x] = low[x] = ++tot;
    Stack[++pos] = x; //不用vis
    ++num;
    for (auto to : G[x]) {
        if (!dfn[to]) {
            tarjan(to);
            low[x] = min(low[x], low[to]);
            if (dfn[x] == low[to]) {  //以x为根的点双
                scc++;
                //val[scc] = 0;
                int y;
                while (y = Stack[pos--]) {
                    T[scc].push_back(y);
                    T[y].push_back(scc);
                    //val[scc]++;
                    //注意圆方树判断  y和to  不是x！！
                    if (y == to) { break; }
                }
                T[scc].push_back(x);
                T[x].push_back(scc);
```

```
                //val[scc]++;

            }
        } else low[x] = min(low[x], dfn[to]);
    }
}
    for (int i = 1; i <= n ; i++) val[i] = ;
    for (int i = 1; i <= n; i++) {
        if (!dfn[i]) { //对于这棵树
            num = 0;
            pos=0;//记得清空栈
            tarjan(i);
        }
    }
```

2-sat + tarjan缩点SCC

```
///每个变量只有两种赋值,给出一系列限制条件,求出一组解
///x->y表示 选了x必须选y  每个点要么0要么1,就两种
bool two_sat() {///注意从哪里开始  0~2*n-1  2i 2i+1
    for(int i=0;i<2*n;++i) if(!dfn[i]) tarjan(i);
    for(int i=0;i<n;++i) if(scc[i<<1]==scc[i<<1|1]) return 0;
  for(int i=0;i<n;++i) if(scc[i<<1]<scc[i<<1|1]) ans[i]=0; else ans[i]=1;
    return 1;
}
    int n1=(a1<<1)+c1;//n1 conflict with n2,=> select n1,must !n2
    int n2=(a2<<1)+c2;
    add(n1,n2^1),add(n2,n1^1);

bool dfs(int u) {//O(n^2)
    if(ans[u^1]) return false;
    if(ans[u]) return true;
    ans[u]=1; stk[tp++]=u;
    for(auto &v:g[u]) if(!dfs(v)) return false;
    return true;
}
bool min_lex() {
    for(int i=0;i<2*n;i+=2) {
        if(!ans[i] && !ans[i+1]) {
            tp=0;
            if(!dfs(i)) {
                while(tp) ans[stk[--tp]]=0;
                if(!dfs(i+1)) return false;
            }
        }
    }
    return true;
}
```

一般图最大匹配带花树

```
///求解一般图博弈问题，每个点是否必胜
vector<int>G[maxn];
namespace Blossom { //注意初始化 Blossom::n 0~n-1
    const int N=500+5;
    bool ban[N];
    int mate[N],n,ret;
    int nxt[N],dsu[N],mark[N],vis[N];
    queue<int> Q;
    int get(int x) {return (x==dsu[x]) ? x: (dsu[x]=get(dsu[x]));}
    void merge(int a,int b) {dsu[get(a)]=get(b);}
    int lca(int x,int y) {
        static int t=0;
        ++t;
        for(;;swap(x,y))
            if(x!=-1) {
                if(vis[x=get(x)]==t) return x;
                vis[x]=t;
                x=(mate[x]!=-1)?nxt[mate[x]]:-1;
            }
    }
    void group(int a,int p) {
        for(int b,c;a!=p;merge(a,b),merge(b,c),a=c) {
            b=mate[a],c=nxt[b];
            if(get(c)!=p)nxt[c]=b;
            if(mark[b]==2)mark[b]=1,Q.push(b);
            if(mark[c]==2)mark[c]=1,Q.push(c);
        }
    }
    void aug(int s,const vector<int> G[]) {///start from s ,do augment
        for(int i=0;i<n;++i) nxt[i]=vis[i]=-1,dsu[i]=i,mark[i]=0;
        while(!Q.empty()) Q.pop();
        Q.push(s);
        mark[s]=1;
        while(mate[s]==-1 && !Q.empty()) {
            int x=Q.front(); Q.pop();
            for(auto &y:G[x])
            if(!ban[y] && y!=mate[x]&& get(x)!=get(y) && mark[y]!=2) {
                if(mark[y]==1) {
                    int p=lca(x,y);
                    if(get(x)!=p) nxt[x]=y;
                    if(get(y)!=p) nxt[y]=x;
                    group(x,p),group(y,p);
                }
                else if(mate[y]==-1) {
                    nxt[y]=x;
                    for(int j=y,k,l;j!=-1;j=l) k=nxt[j],l=mate[k],mate[j]=k,mate[k]=j;
                    break;
```

```
                }
                else nxt[y]=x,Q.push(mate[y]),mark[mate[y]]=1,mark[y]=2;
            }
        }
    }
    void solve(int n,const vector<int> G[]) {
        for(int i=0;i<n;++i) mate[i]=-1;
        for(int i=0;i<n;++i) if(mate[i]==-1) aug(i,G);
        for(int i=0;i<n;++i) {
            int j=mate[i];
            if(j==-1) {
                cout<<"1";
                continue;
            }
            mate[i]=mate[j]=-1;
            ban[i]=1;
            aug(j,G);
            ban[i]=0;
            if(mate[j]==-1) {
                cout<<"0";
                aug(i,G);
            }
            else cout<<"1";
        }
        cout<<'\n';
    }
};
```

最小割树

```
int node[210],vis[210];
void cut(int u){
    vis[u]=1;
    for(int i=h[u];~i;i=a[i].next){
        int v=a[i].to;
        if(a[i].w&&!vis[v]) cut(v);
    }
}
void solve(int l,int r){
    if(l==r) return;
    for(int i=0;i<=cnt;i++) a[i].w=a[i].W;
    int tmp,t[2][210]={0};
    memset(vis,0,sizeof(vis));
    tmp=dinic(node[l],node[r]);
    cut(node[l]);
    for(int i=1;i<=n;i++){
        if(vis[i]){
```

```
            for(int j=1;j<=n;j++){
                if(!vis[j]){
                    ans[i][j]=ans[j][i]=min(ans[i][j],tmp);
                }
            }
        }
    }
    for(int i=l;i<=r;i++) t[vis[node[i]]][++t[vis[node[i]]][0]]=node[i];
    for(int i=l,j=1;j<=t[0][0];++j,++i) node[i]=t[0][j];
    for(int i=l+t[0][0],j=1;j<=t[1][0];j++,++i) node[i]=t[1][j];
    solve(l,l+t[0][0]-1);
    solve(l+t[0][0],r);
}
```

差分约束

SLF优化：采用deque<>,设从u扩展出了,v,队列中队首元素为k,若dis[v]<dis[k],则将v插入队首，否则插入队尾(队列为空时直接插入队尾)

```
/* 差分约束，找到所有约束条件（包括隐含
   最大解：  最短路 a-b<=c add(b,a,c)建边 出现负环则无解 否则存dis
   最小解：  最长路 b-a>=-c add(a,b,-c)建边 不可能有正环 有解存在dis
   (也可以构造乘积 a/b>=c的形式)
   dis初始值分别为 0x3f, -0x3f
   可以由区间关系等得到路径,下为最短路 */
const int maxn = 3e4 + 10, maxm = 5e4 + 10;
int e[maxm], ne[maxm], w[maxm];
int h[maxn],dis[maxn],num[maxn]; //入队次数
bool vis[maxn];
int idx;//边数  记得 memset(h, -1, sizeof(h));
void add(int u, int v, int val){
    e[idx] = v, w[idx] = val;
    ne[idx] = h[u], h[u] = idx++;
}
bool spfa(int x)
{
    memset(dis, 0x3f, sizeof(dis));
    memset(num, 0, sizeof(num));
    memset(vis, 0, sizeof(vis));
    queue<int> q;
    q.push(x);
    dis[x] = 0;
    vis[x] = true;
    num[x]++;
    while (!q.empty()){
        int u = q.front();
        q.pop();
        vis[u] = false;
        for (int i = h[u]; ~ i; i = ne[i]){
```

```
            if (dis[e[i]] > dis[u] + w[i]){
                dis[e[i]] = dis[u] + w[i];
                if (!vis[e[i]]){
                    vis[e[i]] = true;
                    q.push(e[i]);
                    //这里比 num[e[i]]++快很多
                    num[e[i]] = num[u] + 1;
                    if (num[e[i]] > n + 10){
                        return false;
                    }
                }
            }
        }
    }
    return true;
}
```

最小树形图

```
/*  指定点为根的有向生成树，要求权值和最小
    扩展：无根树的树形图
    建立０号节点，向每个点连sum(all)+1边
    因为权值很大，结果肯定只包含一条
    if(ans>sum+sum+1)无解 , else 最终答案＝ans-sum-1
    若输出最小根节点，０号点最后选择的出边，一定是所求
    遍历时if(u==root) pos=i , 答案  pos-m-1
*/
typedef long long ll;
typedef pair<int, int> pii;
const int maxn = 1e2 + 50, maxm = 1e4 + 50;
const int inf = 0x3f3f3f3f;
int n, m, root;
ll ans;
struct edge { int u, v, w; } e[maxm];
int cnt, fa[maxn], id[maxn], top[maxn], mi[maxn];
// cnt当前图环的数量
// id[u]代表u节点在第id[u]个环中
// top[u]代表u所在链的代表元素 类似并查集
// mi[u]为当前连到u点的最短边的边权
// fa[v]当前连到v点的最短边的u
int getans() {
    while (1) {
        for (int i = 1; i <= n; ++i) {
            id[i] = top[i] = 0, mi[i] = inf;
        }
        for (int i = 1; i <= m; ++i) {
            int u = e[i].u, v = e[i].v;
            if (u != v && e[i].w < mi[v]) {
```

```
                //不是自环 更新入边
                fa[v] = u, mi[v] = e[i].w;
                //if(u==root) pos=i; //无根树找最小根
            }
        }
        for (int i = 1; i <= n; i++) {
            if (i != root && mi[i] == inf) return -1;
        }
        for (int i = 1; i <= n; ++i) {
            if (i == root) continue;
            ans += mi[i];
            int v = i;
            while (top[v] != i && !id[v] && v != root) {
                top[v] = i;
                v = fa[v];
            }
            if (!id[v] && v != root) {  // top[v]==i 成环
                id[v] = ++cnt;
                for (int u = fa[v]; u != v; u = fa[u]) {
                    id[u] = cnt;
                }
            }
        }
        if (!cnt) return 1;             //没环  有解
        for (int i = 1; i <= n; ++i) {  //自己成环
            if (!id[i]) id[i] = ++cnt;
        }
        for (int i = 1; i <= m; ++i) {
            int u = e[i].u, v = e[i].v;
            e[i].u = id[u], e[i].v = id[v];
            if (id[u] != id[v]) e[i].w -= mi[v];
            //当前边的两个端点不在同一个环内
        }
        root = id[root];
        n = cnt, cnt = 0;
        //缩完点后 当前点数就为环数 根节点就是根节点所在的环
    }
}
if (getans())  cout << ans << endl;
else  cout << "-1" << endl;
```

树

trick

点分治

重心分治，处理树上距离，路径统计等

例题：距离小于k点对，需要bit

以下为树上所有点对距离放在ans桶里，需要fft。大致思想可以从此得知(现学。

```
vector<pii> E[N];vector<int> aux;int sz[N], h[N];bool vis[N];
void prepare(int u, int fa) {
    sz[u] = 1;
    h[u] = 0;
    for(auto &e : E[u]) {
        int v, w;
        tie(v, w) = e;
        if(v == fa || vis[v]) continue;
        prepare(v, u);
        sz[u] += sz[v];
        h[u] = max(h[v] + w, h[u]);
    }
}
void getroot(int u, int fa, int &m, int &root) {
    if(sz[u] * 2 < m) return;
    if(sz[u] < sz[root]) root = u;
    for(auto &e : E[u]) {
        int v, w;
        tie(v, w) = e;
        if(v == fa || vis[v]) continue;
        getroot(v, u, m, root);
    }
}
void getinfo(int u, int fa, int curd) {
    if(aux.size() <= curd) {
        while(aux.size() < curd) aux.push_back(0);
        aux.push_back(1);
    } else {
        aux[curd]++;
    }
    for(auto &e : E[u]) {
        int v, w;
        tie(v, w) = e;
        if(v == fa || vis[v]) continue;
        getinfo(v, u, curd + w);
    }
}
const int M = 2e5 + 10;
int ans[M];
void conquer(int u) {
    prepare(u, 0);
    sort(all(E[u]), [&](pii x, pii y) { return h[x.fi] + x.se < h[y.fi] + y.se;});
```

```
    vector<int> z = {1};
    for(auto &e : E[u]) {
        int v, w;
        tie(v, w) = e;
        if(vis[v]) continue;
        aux.clear();
        getinfo(v, u, w);
        auto c = multiply(z, aux);
        for(int i = 1; i < (int) c.size(); i++) {
            ans[i] += c[i];
        }
        if(z.size() < aux.size()) {
            z.resize(aux.size());
        }
        for(int i = 0; i < aux.size(); i++) {
            z[i] += aux[i];
        }
    }
}
void divide(int u) {
    prepare(u, 0);
    int m = sz[u], root = u;
    getroot(u, 0, m, root);
    conquer(root);
    vis[root] = 1;
    for(auto &e : E[root]) {
        int v, w;
        tie(v, w) = e;
        if(vis[v]) continue;
        divide(v);
    }
}
```

dsu

只需考虑如何继承重儿子，新增和删除结点的影响。

不好维护的，可以结合bit、状压、并查集其他工具来做。

```
int sz[maxn],hson[maxn],a[maxn],in[maxn],out[maxn],dfn,c[maxn];
vector<int>g[maxn];
ll ans;
void dfs(int u,int f)
{
    in[u]=++dfn,c[dfn]=u,sz[u]=1,hson[u]=0;
    int mxsz=0;
    for(auto &v:g[u])
    {
        if(v==f) continue;
```

```
        dfs(v,u);
        sz[u]+=sz[v];
        if(sz[v]>mxsz) hson[u]=v,mxsz=sz[v];
    }
    out[u]=dfn;
}
void add(int u,int x) ///add  del
void solve(int u,int f,bool keep)
{
    for(auto &v:g[u])
    {
        if(v==f || v==hson[u]) continue;
        solve(v,u,0);
    }
    if(hson[u]) solve(hson[u],u,1);
    add(u,1);
    for(auto &v:g[u])///solve light-son-tree
    {
        if(v==hson[u] || v==f) continue;
        for(int i=in[v];i<=out[v];++i) ///calc the answer
        for(int i=in[v];i<=out[v];++i)
        {
            int cur=c[i];
            add(cur,1);
        }
    }
    if(!keep)
    {
        for(int i=in[u];i<=out[u];++i)
        {
            int cur=c[i];
            add(cur,-1);
        }
    }
}
```

树上计数

prefer序列(见笔记)

lca + 树剖

树剖或者倍增预处理。原理重儿子是最大的，从任何点到根轻边不超log_2n条

```
const int maxn=3e5+7,inf=0x3f3f3f3f,mod=1e9+7;
int dep[maxn],fa[maxn][22];//类似可维护k级祖先距离和最大边点权
vector<int>g[maxn];
void dfs(int u,int f)
{
```

```
    dep[u]=dep[f]+1;
    fa[u][0]=f;
    for(int j=1;j<=20;++j) fa[u][j]=fa[fa[u][j-1]][j-1];
    for(auto &v:g[u])
    {
        if(v==f) continue;
        dfs(v,u);
    }
}
int lca(int u,int v)
{
    if(dep[u]<dep[v]) swap(u,v);
    for(int j=20;j>=0;--j) if(dep[fa[u][j]]>=dep[v]) u=fa[u][j];
    if(u==v) return u;
    for(int j=20;j>=0;--j) if(fa[u][j]!=fa[v][j]) u=fa[u][j],v=fa[v][j];
    return fa[u][0];
}
```

```
void dfs1(int u, int fa) {
    pa[u] = fa;
    sz[u] = 1;
    dep[u] = dep[fa] + 1;
    hson[u] = 0;
    for (int i = head[u], v; i; i = e[i].nxt) {
        v = e[i].to;
        if (v == fa) continue;
        dfs1(v, u);
        sz[u] += sz[v];
        if (sz[v] > sz[hson[u]]) hson[u] = v;
    }
}
void dfs2(int u, int t) {
    top[u] = t;
    dfn[u] = ++cnt;
    if (hson[u] != 0) dfs2(hson[u], t);
    for (int i = head[u], v; i; i = e[i].nxt)
        if ((v = e[i].to) != hson[u] && v != pa[u]) dfs2(v, v);
}
int lca(int x, int y) {
    while (top[x] != top[y])  //判断是否在一条重链上
    {
        if (dep[top[x]] >= dep[top[y]])
            x = pa[top[x]];  //链顶深度大的上跳
        else
            y = pa[top[y]];
    }
    return dep[x] < dep[y] ? x : y;
}
```

最小斯坦纳树

```
/*
最小代价，连通给定的ｋ个点,状压ＤＰ，答案子图一定是树
DP[i][S],以ｉ为根，Ｓ为点集的最小代价
１）ｉ度数为１,相邻点转移  DP[j][S]+w(j,i)->DP[i][S]
2) i度数>1 划分子树考虑   DP[i][T]+DP[i][S-T]->DP[i][S] (T包含于Ｓ)
类似背包,2)枚举子集，1)最短路三角不等式 O(n*3^k+mlogm*2^k)
*/
#define fi first
#define se second
typedef pair<int, int> pii;
const int maxn = 510;
int n, m, k;
vector<pii> G[maxn];
int p[maxn], vis[maxn], dp[maxn][4200];
priority_queue<pii> q;
void dijkstra(int s) {
    memset(vis, 0, sizeof(vis));
    while (!q.empty()) {
        pii u = q.top();
        q.pop();
        if (vis[u.se]) continue;
        vis[u.se] = 1;
        for (auto it : G[u.se]) {
            if (dp[it.se][s] > dp[u.se][s] + it.fi) {
                dp[it.se][s] = dp[u.se][s] + it.fi;
                q.push(make_pair(-dp[it.se][s], it.se));
            }
        }
    }
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    memset(dp, 0x3f, sizeof(dp));
    cin >> n >> m >> k;
    for (int i = 1; i <= m; i++) {
        int x, y, z;
        cin >> x >> y >> z;
        G[x].push_back({z, y});
        G[y].push_back({z, x});
    }
    //目标点数
    for (int i = 1; i <= k; i++) {
        cin >> p[i];
        dp[p[i]][1 << (i - 1)] = 0;
    }
```

```
    for (int s = 1; s < (1 << k); s++) {
        for (int i = 1; i <= n; i++) {
            for (int subs = s & (s - 1); subs; subs = s & (subs - 1)) {
                dp[i][s] = min(dp[i][s], dp[i][subs] + dp[i][s ^ subs]);
            }
            if (dp[i][s] != 0x3f3f3f3f) {
                q.push(make_pair(-dp[i][s], i));
            }
        }
        dijkstra(s);
    }
    printf("%d\n", dp[p[1]][(1 << k) - 1]);
}
```

无向图全局最小割

```
int solve() {
  int res = INF;
  for (int i = 1; i <= n; i++) mask[i] = i;
  while (n > 1) {
    int k, pre = 1;  // 默认1号点是集合的第一个点
    for (int i = 1; i <= n; i++) vis[mask[i]] = 0, d[mask[i]] = 0;
    vis[mask[pre]] = true;
    for (int i = 2; i <= n; i++) {
      k = -1;
      for (int j = 1; j <= n; j++) {  // 寻找距离最远的点加入集合
        if (!vis[mask[j]]) {
          d[mask[j]] += G[mask[pre]][mask[j]];
          if (k == -1 || d[mask[k]] < d[mask[j]]) k = j;
        }
      }
      vis[mask[k]] = true;  // 加入集合
      if (i == n) {         // 只剩一个点
        res = min(res, d[mask[k]]);
        for (int j = 1; j <= n; j++) {  // 修改边权
          G[mask[pre]][mask[j]] += G[mask[j]][mask[k]];
          G[mask[j]][mask[pre]] += G[mask[j]][mask[k]];
        }
        mask[k] = mask[n--];  // 去掉最后加入的点
      }
      pre = k;
    }
  }
  return res;
}
```

欧拉回路

模型：每个点被两种覆盖，差值不超过1，转化到欧拉回路

t=1代表无向图，t=2代表有向图。输出方案，负数代表走反向边

```
constexpr int N(1e5 + 5), M(4e5 + 5);
int head[N], next[M], to[M], tot;
void init(int n) {
  memset(head, -1, (n + 1) * sizeof(int));
  tot = 0;
}
void add(int x, int y) {
  to[tot] = y, next[tot] = head[x], head[x] = tot++;
}
bool vis[M];
int stack[M], top;
void dfs(int x) {
  for (int i; ~(i = head[x]); ) {
    head[x] = next[i];
    if (!vis[i / 2]) {
      vis[i / 2] = true;
      dfs(to[i]);
      stack[++top] = i;
    }
  }
}
int d[N];
int main() {
  int t, n, m;
  std::cin >> t >> n >> m;
  init(n);
  for (int i = 0; i < m; i++) {
    int x, y;
    std::cin >> x >> y;
    add(x, y);
    if (t == 1) {
      add(y, x);
      d[x]++, d[y]++;
    } else {
      tot++;
      d[x]++, d[y]--;
    }
  }
  if (t == 1) {
    for (int i = 1; i <= n; i++) if (d[i] & 1) return puts("NO"), 0;
  } else {
    for (int i = 1; i <= n; i++) if (d[i]) return puts("NO"), 0;
  }
  dfs(to[0]);
  if (top < m) return puts("NO"), 0;
```

```
  std::cout << "YES\n";
  if (t == 1) {
    while (top) {
      int i = stack[top--];
      std::cout << (i & 1 ? -i / 2 - 1 : i / 2 + 1) << " ";
    }
  } else {
    while (top) {
      int i = stack[top--];
      std::cout << i / 2 + 1 << " ";
    }
  }
  return 0;
}
```