**2013 年 10 月(非完整版)**

# [（Graph）图论模板]

[Yucept]

# 网络流

## 算法模板

**Dinic**

```
struct Edge {
    int from, to, cap, flow;
    Edge(){}
    Edge(int _from, int _to, int _cap, int _flow)
    : from(_from), to(_to), cap(_cap), flow(_flow){}
};
struct Dinic {
    int n, m, s, t;
    Edge edges[maxm * 2];
    int head[maxn];
    int next[maxm * 2];
    bool inq[maxn];
    int d[maxn];
    int cur[maxn];
    void init(int n){
        this->n = n;
        m = 0;
        memset(head, -1, sizeof(head[0]) * (n + 1));
    }
    void AddEdge(int from, int to, int cap) {
        next[m] = head[from];
        edges[m] = Edge(from, to, cap, 0);
        head[from] = m ++; //
        next[m] = head[to];
        edges[m] = Edge(to, from, 0, 0);
        head[to] = m ++;
    }
    bool bfs() {
        memset(inq, false, sizeof(inq[0]) * (n + 1));
        queue<int> q;
        q.push(s);
        d[s] = 0; inq[s] = true;
        while(!q.empty()) {
            int u = q.front(); q.pop();
            for(int i = head[u]; i != -1; i = next[i]) {
                Edge& e = edges[i];
                int v = e.to;
                if(!inq[v] && e.cap > e.flow) {
                    inq[v] = true;
                    d[v] = d[u] + 1;
                    q.push(v);
                    if(v == t) return true;
                }
            }
        }
        return false;
    }
```

```
    int dfs(int u, int a) {
        if(u == t || a == 0) return a;
        int flow = 0, f;
        for(int& i = cur[u]; i != -1; i = next[i]) {
            Edge& e = edges[i];
            int v = e.to;
            if(d[u] + 1 == d[v] && (f = dfs(v, min(a, e.cap - e.flow))) > 0) {
                e.flow += f;
                edges[i ^ 1].flow -= f;
                flow += f;
                a -= f;
                if(a == 0) break;
            }
        }
        return flow;
    }
    int Maxflow(int s, int t) {
        this->s = s; this->t = t;
        int flow = 0;
        while(bfs()) {
            memcpy(cur, head, sizeof(head[0]) * (n + 1));
            flow += dfs(s, inf);
        }
        return flow;
    }
};
```

**ISAP**

```
struct ISAP {
    int n, m, s, t;
    Edge edges[maxm * 2];
    int head[maxn];
    int next[maxm * 2];
    bool inq[maxn];
    int d[maxn];
    int cur[maxn];
    int p[maxn];
    int num[maxn];
    void init(int n){
        this->n = n;
        m = 0;
        memset(head, -1, sizeof(head[0]) * (n + 1));
    }
    void AddEdge(int from, int to, int cap) {
        next[m] = head[from];
        edges[m] = Edge(from, to, cap, 0);
        head[from] = m ++;
        next[m] = head[to];
        edges[m] = Edge(to, from, 0, 0);
        head[to] = m ++;
    }
    bool bfs() {
        memset(inq, false, sizeof(inq[0]) * (n + 1));
```

```
    queue<int> q;
    q.push(t);
    d[t] = 0; inq[t] = true;
    while(!q.empty()) {
        int u = q.front(); q.pop();
        for(int i = head[u]; i != -1; i = next[i]) {
            Edge& e = edges[i];
            int v = e.to;
            if(!inq[v] && e.cap >= e.flow) {
                //printf("u = %d, v = %d\n", u, v);
                inq[v] = true;
                d[v] = d[u] + 1;
                q.push(v);
            }
        }
    }
    return inq[s];
}
int Augment() {
    int x = t, a = inf;
    while(x != s) {
        Edge& e = edges[p[x]];
        a = min(a, e.cap - e.flow);
        x = edges[p[x]].from;
    }
    x = t;
    while(x != s) {
        edges[p[x]].flow += a;
        edges[p[x] ^ 1].flow -= a;
        x = edges[p[x]].from;
    }
    return a;
}
int Maxflow(int s, int t) {
    this->s = s; this->t = t;
    int flow = 0;
    bfs();
    memset(num, 0, sizeof(num[0]) * (n + 1));
    for(int i = 0; i < n; i ++) num[d[i]] ++;
    int x = s;
    memcpy(cur, head, sizeof(head[0]) * (n + 1));
    while(d[x] < n) {
        if(x == t) {
            flow += Augment();
            x = s;
        }
        bool ok = false;
        for(int i = cur[x]; i != -1; i = next[i]) {
            Edge& e = edges[i];
            if(e.cap > e.flow && d[x] == d[e.to] + 1) {
                ok = true;
                p[e.to] = i;
```

```
                cur[x] = i;
                x = e.to;
                break;
            }
        }
        if(!ok) {
            int mt = n - 1;
            for(int i = head[x]; i != -1; i = next[i]) {
                Edge& e = edges[i];
                if(e.cap > e.flow) mt = min(mt, d[e.to]);
            }
            if(-- num[d[x]] == 0) break; //gap 优化
            num[d[x] = mt + 1] ++;
            cur[x] = head[x];
            if(x != s) x = edges[p[x]].from;
        }
    }
    return flow;
  }
};
```

**普通费用流**

```
struct MCMF {
  int n, m, s, t;
  Edge edges[maxm * 2];
  int head[maxn];
  int next[maxm * 2];
  bool inq[maxn];
  int d[maxn];
  int p[maxn];
  int a[maxn];
  void init(int n) {
    this->n = n;
    m = 0;
    memset(head, -1, sizeof(head[0]) * (n + 1));
  }
  void AddEdge(int from, int to, int cap, int cost) {
    next[m] = head[from];
    edges[m] = Edge(from, to, cap, 0, cost);
    head[from] = m ++;
    next[m] = head[to];
    edges[m] = Edge(to, from, 0, 0, -cost);
    head[to] = m ++;
  }
  bool BellmanFord(int s, int t, int& flow, int& cost) {
    for(int i = 0; i < n; i ++) d[i] = inf;
    memset(inq, false, sizeof(inq[0]) * (n + 1));
    d[s] = 0; inq[s] = true; p[s] = 0; a[s] = inf;
    queue<int> q;
    q.push(s);
    while(!q.empty()) {
      int u = q.front(); q.pop();
      inq[u] = false;
```

```
        for(int i = head[u]; i != -1; i = next[i]) {
            Edge& e = edges[i]; int v = e.to;
            if(e.cap > e.flow && d[v] > d[u] + e.cost) {
                d[v] = d[u] + e.cost;
                p[v] = i;
                a[v] = min(a[u], e.cap - e.flow);
                if(!inq[v]) {q.push(v); inq[v] = true;}
            }
        }
    }
    if(d[t] == inf) return false;
    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;
}
int Mincost(int s, int t) {
    int flow = 0, cost = 0;
    while(BellmanFord(s, t, flow, cost));
    return cost;
}
```

**ZKW（高效）费用流**

```
};
struct ZKW_flow{
    int st, ed, n, m;
    int head[maxn];
    int cap[maxm], cost[maxm], to[maxm], next[maxm];
    void init(int n){
        this->n = n;
        memset(head, -1, sizeof(head[0])* (n + 1));
        m = 0;
    }
    void AddEdge(int u, int v, int c, int w) {
        cap[m] = c; cost[m] = w; to[m] = v;
        next[m] = head[u]; head[u] = m ++;
        cap[m] = 0; cost[m] = -w; to[m] = u;
        next[m] = head[v]; head[v] = m++;
    }
    int dis[maxn];
    void Dijkstra() {
        for(int i = 0; i <= n; ++i) dis[i] = inf;
        priority_queue<pii, vector<pii>, greater<pii> > Q;
        dis[st] = 0;
        Q.push(make_pair(0, st));
        while(!Q.empty()){
            int u = Q.top().second, d = Q.top().first;
            Q.pop();
```

```
        if(dis[u] != d) continue;
        for(int p = head[u]; p != -1; p = next[p]){
            int &v = to[p];
            if(cap[p] && dis[v] > d + cost[p]){
                dis[v] = d + cost[p];
                Q.push(make_pair(dis[v], v));
            }
        }
    }
    for(int i = 0; i <= n; ++ i) dis[i] = dis[ed] - dis[i];
}
int MinCost, MaxFlow;
bool used[maxm];
int Add_Flow(int u, int flow){
    if(u == ed){
        MaxFlow += flow;
        MinCost += dis[st] * flow;
        return flow;
    }
    used[u] = true;
    int now = flow;
    for(int p = head[u]; p != -1; p = next[p]){
        int &v = to[p];
        if(cap[p] && !used[v] && dis[u] == dis[v] + cost[p]){
            int tmp = Add_Flow(v, min(now, cap[p]));
            cap[p] -= tmp;
            cap[p^1] += tmp;
            now -= tmp;
            if(!now) break;
        }
    }
    return flow - now;
}
bool modify_label(){
    int d = inf;
    for(int u = 0; u <= n; ++ u) if(used[u])
        for(int p = head[u]; p != -1; p = next[p]){
            int &v = to[p];
            if(cap[p] && !used[v]) d = min(d, dis[v] + cost[p] - dis[u]);
        }
    if(d == inf) return false;
    for(int i = 0; i <= n; ++ i) if(used[i]) dis[i] += d;
    return true;
}
int MCF(int s, int t){
    st = s, ed = t;
    MinCost = MaxFlow = 0;
    Dijkstra();
    while(true){
        while(true){
            for(int i = 0; i <= n; ++ i) used[i] = false;
            if(!Add_Flow(st, inf)) break;
```

```
        }
        if(!modify_label()) break;
    }
    FLOW = MaxFlow; //定义的 FLOW 为最终求得的流量
    return MinCost;
  }
};
```

# 建模

**矩阵取值问题（费用流）**

1.矩阵的每个格子里有一个数字，从左上角走到右下角，走 k 次，求最大的数字之和。（格子里的数字第一次走过后变成 0）。

```
MCMF mc;
int N, K;
int src, sink;
int mat[55][55];
void Prepare() {
  int Nt = N * N;
  mc.init(2 * Nt + 2);
  src = 0; sink = 2 * Nt + 1;
  for(int i = 1; i <= N; i ++) {
    for(int j = 1; j <= N; j ++) {
      scanf("%d", &mat[i][j]); //每个格子的数值
    }
  }
  for(int i = 1; i <= N; i ++) {
    for(int j = 1; j <= N; j ++) {
      int u = (i - 1) * N + j, v;
      mc.AddEdge(u, u + Nt, 1, -mat[i][j]); //拆点，每个点只有第一次有值
      mc.AddEdge(u, u + Nt, K - 1, 0); //剩下 K-1 次的值为 0
      if(j < N) {
        v = u + 1;
        mc.AddEdge(u + Nt, v, K, 0);
      }
      if(i < N) {
        v = u + N;
        mc.AddEdge(u + Nt, v, K, 0);
      }
    }
  }
  mc.AddEdge(src, 1, K, 0); //超级源点到左上角 1 号点，容量为 K，限制 k 次
  mc.AddEdge(2 * Nt, sink, K, 0);
}
int main()
{
  while(scanf("%d%d", &N, &K) != EOF) {
    Prepare();
    int ans = mc.Mincost(src, sink);
    printf("%d\n", -ans); //这是一个求最大的问题，所以要将格子的值取相反数之后求最小费用再将最后的结果求相反数。
  }
```

```
    return 0;
}
```

**最大流+二分**

```
/*
二分一般是用来限制源点流出的流量。
对于点有流量限制（比如只能走一次）可以拆点变成边来限制
拆点要准确，注意入点和出点。
*/
ISAP sap;
int Tot;
int N, M, K, src, sink;
bool like[maxn/4][maxn/4];
bool check(int cap) {
    sap.init(4 * N + 2);
    for(int i = 1; i <= N; i ++) {
        for(int j = 1; j <= N; j ++) {
            if(like[i][j]) sap.AddEdge(i, j + 3 * N, 1);
            else sap.AddEdge(i + N, j + 2 * N, 1);
        }
    }
    for(int i = 1; i <= N; i ++) {
        sap.AddEdge(i, i + N, K);
        sap.AddEdge(i + 2 * N, i + 3 * N, K); //点的限制为 K
        sap.AddEdge(src, i, cap); //源点流出的限制
        sap.AddEdge(i + 3 * N, sink, cap);
    }
    int tflow = sap.Maxflow(src, sink);
    //printf("cap = %d, flow = %d\n", cap, tflow);
    if(tflow == N * cap) return true; //判断满流
    return false;
}
void Prepare() {
    scanf("%d%d%d", &N, &M, &K);
    memset(like, false, sizeof(like));
    int u, v;
    while(M --) {
        scanf("%d%d", &u, &v);
        like[u][v] = true;
    }
    src = 0, sink = 4 * N + 1;
}
int main()
{
    //freopen("input.txt", "r", stdin);
    scanf("%d", &Tot);
    for(int ac = 1; ac <= Tot; ac ++) {
        Prepare();
        int l = 0, r = N, ans;
        while(l <= r) {
            ans = (l + r) >> 1;
            if(check(ans)) l = ans + 1;
            else r = ans - 1;
```

```
        }
        printf("%d\n", l - 1);
    }
    return 0;
}
```

**最大权闭合子图**

```
/**
N 个点需要一定费用来建设，M 条边可以获得一定收益
连边：N 个点为花费点，M 条边为获利点
src 向获利点连边，容量为 profit
获利点向与其相关的花费点连边，容量为 inf
花费点向 sink 连边，容量为 cost
*/
void Prepare() {
    src = 0; sink = N + M + 1;
    Di.init(N + M + 2);
    for(int i = 1; i <= N; i ++) {
        scanf("%d", &cost);
        Di.AddEdge(i, sink, cost);
    }
    int u, v, pf;
    ans = 0;
    for(int i = 1; i <= M; i ++) {
        scanf("%d%d%d", &u, &v, &pf);
        ans += pf;
        Di.AddEdge(i + N, u, inf);
        Di.AddEdge(i + N, v, inf);
        Di.AddEdge(src, i + N, pf);
    }
}
int main()
{
    while(scanf("%d%d", &N, &M)!= EOF) {
        Prepare();
        printf("%d\n", ans - Di.Maxflow(src, sink));
    }
    return 0;
}
```

**动态流+分层拆点**

```
/*
分层拆点要注意连边不出现问题
出点向入点连边
*/
Dinic Di;
int Tot;
int N, M, src, sink;
vector<int> G[60];
int D, W, C[60];
struct Pt{
    int x, y;
};
Pt pt[60];
```

```
int sqr(int x) {
    return x * x;
}
bool Near(Pt a, Pt b) {
    return sqr(a.x - b.x) + sqr(a.y - b.y) <= D * D;
}
void Prepare() {
    for(int i = 1; i <= N; i ++) {
        scanf("%d%d%d", &pt[i].x, &pt[i].y, &C[i]);
    }
    for(int i = 1; i <= N; i ++) {
        for(int j = i + 1; j <= N; j ++) {
            if(Near(pt[i], pt[j])) {
                G[i].push_back(j); G[j].push_back(i);
            }
        }
    }
    src = 0; sink = 2*(N + M) * N + 1;
    Di.init(sink + 1);
}
void conect(int cur) {
    int u, v;
    for(int i = 1; i <= N; i ++) {
        u = 2 * cur * N + i, v = u + N;
        Di.AddEdge(u, v, C[i]); //对每层的点拆点，表示点的容量限制
        if(pt[i].y <= D) { //超级源能到的点连边
            Di.AddEdge(src, u, M);
        }
        if(pt[i].y + D >= W) { //能到超级汇的连边
            Di.AddEdge(v, sink, M);
        }
    }
    if(cur) {
        for(int i = 1; i <= N; i ++) {
            u = 2 * cur * N + i;
            int sz = G[i].size();
            for(int j = 0; j < sz; j ++) {
                v = (2 * cur - 1) * N + G[i][j];
                Di.AddEdge(v, u, inf); //上一层的出点到下一层的入点连边
            }
        }
    }}
int main()
{
    //freopen("input.txt", "r", stdin);
    while(scanf("%d%d%d%d", &N, &M, &D, &W) != EOF) {
        Prepare();
        if(D >= W) {printf("1\n"); continue;}
        if(N == 0) {printf("IMPOSSIBLE\n"); continue;}
        int cur, ans = 0;
        for(cur = 0; cur < N + M; cur ++) {
            conect(cur);
```

```
            ans += Di.Maxflow(src, sink);
            if(ans >= M) break;
        }
        if(cur == N + M) printf("IMPOSSIBLE\n");
        else printf("%d\n", cur + 2);
    }
    return 0;
}
```

**混合图欧拉回路**

```
ISAP sap;
int Tot;
int N, M, src, sink;
int din[maxn], dout[maxn];
void Prepare() {
    scanf("%d%d", &N, &M);
    memset(din, 0, sizeof(din));
    memset(dout, 0, sizeof(dout));
    int u, v, kind;
    sap.init(N + 2);
    src = 0, sink = N + 1;
    while(M --) {
        scanf("%d%d%d", &u, &v, &kind);
        din[v] ++, dout[u] ++;
        if(!kind) sap.AddEdge(u, v, 1);
    }
}
bool check() {
    int tflow = 0;
    for(int i = 1; i <= N; i ++) {
        if((din[i] + dout[i]) & 1) {
            return false;
        }
    }
    for(int i = 1; i <= N; i ++) {
        int x = abs(din[i] - dout[i]) / 2;
        if(din[i] > dout[i]){
            sap.AddEdge(i, sink, x);
            tflow += x;
        }
        if(dout[i] > din[i])
            sap.AddEdge(src, i, x);
    }
    if(tflow == sap.Maxflow(src, sink)) return true;
    return false;
}
int main()
{
    //freopen("input.txt", "r", stdin);
    scanf("%d", &Tot);
    for(int ac = 1; ac <= Tot; ac ++) {
        Prepare();
        bool ok = check();
```

```
        if(!ok) printf("impossible\n");
        else printf("possible\n");
    }
    return 0;
}
```

# 最短路

## 算法模板

**Dijkstra**

```
struct HeapNode {
    int d, u;
    HeapNode(){}
    HeapNode(int _d, int _u)
    : d(_d), u(_u) {}
    bool operator < (const HeapNode& rhs) const {
        return d > rhs.d;
    }
};
struct Dijkstra {
    int n, m;
    Edge edges[maxm];
    int next[maxm];
    int head[maxn];
    int d[maxn];
    int layer[maxn];
    bool done[maxn];
    void init(int n) {
        this->n = n;
        m = 0;
        memset(head, -1, sizeof(head[0]) * (n + 1));
    }
    void AddEdge(int from, int to, int dist) {
        next[m] = head[from];
        edges[m] = Edge(from, to, dist);
        head[from] = m ++;
    }
    void dijkstra(int src, int des) {
        priority_queue<HeapNode> q;
        for(int i = 1; i <= n; i ++) d[i] = inf;
        d[src] = 0;
        memset(done, false, sizeof(done[0]) * (n + 1));
        q.push(HeapNode(0, src));
        while(!q.empty()) {
            HeapNode x = q.top(); q.pop();
            int u = x.u;
            //printf("%d %d\n", x.u, x.d);
            if(done[u]) continue;
            done[u] = true;
            for(int i = head[u]; i != -1; i = next[i]) {
                Edge& e = edges[i];
                int v = e.to;
```

```
            if(d[v] > d[u] + e.dist) {
                d[v] = d[u] + e.dist;
              // printf("u = %d, v = %d, d[v] = %d\n", u, v, d[v]);
                q.push(HeapNode(d[v], v));
            }
        }
    }
  }
};
```

**Floyd**

```
typedef long long LL;
const LL inf = -1;
LL d[maxn][maxn];
void floyd(int n) {
    for(int k = 1; k <= n; k ++) {
        for(int i = 1; i <= n; i ++) {
            for(int j = 1; j <= n; j ++) {
                if(d[i][k] == inf || d[k][j] == inf) continue;
                if(d[i][j] > d[i][k] + d[k][j] || d[i][j] == inf)
                    d[i][j] = d[i][k] + d[k][j];
            }
        }
    }
}
```

**SPFA**

```
/*
有向边，求单源点到所有点的最短路，直接 SPFA
求多源点到单终点的最短路，反向建图，就变成
单源点到多终点的最短路问题
*/
struct Edge{
  int from, to, dist;
  Edge(){}
  Edge(int _from, int _to, int _dist)
  : from(_from), to(_to), dist(_dist){}
};
struct SPFA{
  int n, m;
  Edge edges[maxm];
  int next[maxm];
  int height[maxm]; //某些题目的限制条件
  int head[maxn];
  int inq[maxn];
  int d[maxn];
  int cnt[maxn];
  void init(int n) {
    this->n = n;
    m = 0;
    memset(head, -1, sizeof(head[0]) * (n + 1));
  }
  void AddEdge(int from, int to, int dist) {
    next[m] = head[from];
```

```
    edges[m] = Edge(from, to, dist);
    head[from] = m ++;
}
int spfa(int src, int des, int limit) {
    queue<int> q;
    for(int i = 0; i <= n; i ++){
        d[i] = inf; inq[i] = false;
    }
    inq[src] = true; d[src] = 0;
    q.push(src);
    while(!q.empty()){
        int u = q.front(); q.pop();
        inq[u] = false;
        for(int i = head[u]; i != -1; i = next[i]){
            Edge& e = edges[i];
            int v = e.to;
            if(height[i] < limit) continue;
            if(d[v] > d[u] + e.dist){
                d[v] = d[u] + e.dist;
                if(!inq[v]){
                    inq[v] = true; q.push(v);
                }
            }
        }
    }
    if(d[des] == inf) return -1;
    return d[des];
  }
};
```

# 一些题型

## 求最短路次短路条数

```
/***
 求 s 到 t 的最短路与次短路（这里要求只比最短路多 1）的条数之和
 联想到最小，次小的一种更新关系：
 if(x<最小)更新最小，次小
 else if(==最小)更新方法数
 else if(x<次小)更新次小
 else if(x==次小)更新方法数
 同时记录 s 到 u 最短，次短路及方法数
 用一个堆每次取最小的，更新完后再入堆
 还是那个原理，第一次遇到的就是最优的，然后 done 标记为真
 方法数注意是加法原理，不是乘法
***/
struct HeapNode{
    int d, u;
    int mark; //代表种类，次短为 1，最短为 0
    HeapNode(){}
    HeapNode(int _d, int _u, int _mark)
    : d(_d), u(_u), mark(_mark){}
    bool operator < (const HeapNode& rhs) const{
        return rhs.d < d;
```

```
    }
};
struct Edge{
    int from, to, dist;
    Edge(){}
    Edge(int _from, int _to, int _dist)
    : from(_from), to(_to), dist(_dist){}
};
int n, m, ans, s, t;
vector<int> G[maxn];
vector<Edge> edges;
bool done[maxn][2];
//d[t][0]是最短路的值，d[t][1]是次短路的值
int d[maxn][2], c[maxn][2];
//c[t][0] 最短路条数， c[t][1] 次短路条数
void init(){
    edges.clear();
    for(int i = 0; i <= n; i ++){
        G[i].clear();
    }
}
void AddEdge(int from, int to, int dist){
    edges.push_back(Edge(from, to, dist));
    int m = edges.size();
    G[from].push_back(m - 1);
}
```

```
void Dijkstra(int s, int t){
    priority_queue<HeapNode> q;
    for(int j = 0; j < 2; j ++)
        for(int i = 0; i <= n; i ++){
            d[i][j] = inf; done[i][j] = false; c[i][j] = 0;
        }
    d[s][0] = 0; c[s][0] = 1;
    q.push(HeapNode(d[s][0], s, 0));
    while(!q.empty()){
        HeapNode x = q.top(); q.pop();
        int u = x.u, mark = x.mark;
        if(done[u][mark]) continue;
        done[u][mark] = true;
        int sz = G[u].size();
        for(int i = 0; i < sz; i ++){
            Edge& e = edges[G[u][i]];
            int v = e.to;
            int dis = d[u][mark] + e.dist;
            if(d[v][0] > dis){
                if(d[v][0] != inf){
                    d[v][1] = d[v][0];
                    c[v][1] = c[v][0];
                    q.push(HeapNode(d[v][1], v, 1));
                }
                d[v][0] = dis;
                c[v][0] = c[u][mark];
```

```
            q.push(HeapNode(d[v][0], v, 0));
        }else if(d[v][0] == dis){
            c[v][0] += c[u][mark];
        }else if(d[v][1] > dis){
            d[v][1] = dis;
            c[v][1] = c[u][mark];
            q.push(HeapNode(d[v][1], v, 1));
        }else if(d[v][1] == dis){
            c[v][1] += c[u][mark];
        }
    }
  }
}
int main()
{
  int Tot;
  scanf("%d", &Tot);
  while(Tot --){
    scanf("%d%d", &n, &m);
    init();
    int u, v, w;
    while(m --){
      scanf("%d%d%d", &u, &v, &w);
      AddEdge(u, v, w);
    }
    scanf("%d%d", &s, &t);
    Dijkstra(s, t);
    if(d[t][0] + 1 == d[t][1])
      ans = c[t][0] + c[t][1];
    else ans = c[t][0];
    printf("%d\n", ans);
  }
  return 0;
}
```

**A_star 求 k 短路**

```
/**
起点和终点一样时要将 K + 1
*/
struct HeapNode{
  int d, u;
  HeapNode(){}
  HeapNode(int _d, int _u)
  : d(_d), u(_u){}
  bool operator < (const HeapNode& rhs) const{
    return rhs.d < d;
  }
};
struct Edge{
  int from, to, dist;
  Edge(){}
  Edge(int _from, int _to, int _dist)
  : from(_from), to(_to), dist(_dist){}
```

```
};
struct KSP {
    int n, m;
    int Ghead[maxn];
    int Fhead[maxn];
    int next[maxm * 2]; //边的数组可以尽量开大些
    Edge edges[maxm * 2];
    int cnt[maxn], d[maxn];
    void init(int n){
        this->n = n;
        m = 0;
        memset(Ghead, -1, sizeof(Ghead[0]) * (n + 1));
        memset(Fhead, -1, sizeof(Fhead[0]) * (n + 1));
    }
    void AddEdge(int from, int to, int dist){
        next[m] = Ghead[from];
        edges[m] = Edge(from, to, dist);
        Ghead[from] = m ++;
        next[m] = Fhead[from];
        edges[m] = Edge(to, from, dist);
        Fhead[to] = m ++;
    }
    void Dijkstra(int src){
        priority_queue<HeapNode> q;
        for(int i = 0; i <= n; i ++){
            d[i] = inf; cnt[i] = 0;
        }
        d[src] = 0;
        q.push(HeapNode(d[src], src));
        while(!q.empty()){
            HeapNode x = q.top(); q.pop();
            int u = x.u;
            if(cnt[u]) continue;
            cnt[u] = 1;
            for(int i = Fhead[u]; i != -1; i = next[i]){
                Edge& e = edges[i];
                int v = e.to;
                if(d[v] > d[u] + e.dist){
                    d[v] = d[u] + e.dist;
                    q.push(HeapNode(d[v], v));
                }
            }
        }
    }
int Kth_Astar(int src, int des, int k){
    priority_queue<HeapNode> q;
    if(d[src] == inf) return -1;
    q.push(HeapNode(d[src], src));
    memset(cnt, 0, sizeof(cnt[0]) * (n + 1));
    while(!q.empty()){
        HeapNode x = q.top(); q.pop();
        int u = x.u;
```

```
            cnt[u] ++;
            if(cnt[u] > k) continue;
            if(cnt[des] == k) return x.d;
            for(int i = Ghead[u]; i != -1; i = next[i]){
                Edge& e = edges[i];
                int v = e.to;
                q.push(HeapNode((x.d - d[u] + e.dist + d[v]), v));
            }
        }
        return -1;
    }
};
int N, M, K, S, T;
KSP ks;
int main()
{
    while(scanf("%d%d", &N, &M) != EOF){
        ks.init(N);
        int u, v, w;
        while(M --){
            scanf("%d%d%d", &u, &v, &w);
            ks.AddEdge(u, v, w);
        }
        scanf("%d%d%d", &S, &T, &K);
        if(S == T) K ++;
        ks.Dijkstra(T);
        int ans = ks.Kth_Astar(S, T, K);
        printf("%d\n", ans);
    }
    return 0;
}
```

**二维最短路**

```
/**
用 D[U][FUEL]  表示走到 U 点剩 FUEL 油量时的最小费用
*/
struct HeapNode{
    int d, u, fuel;
    bool operator < (const HeapNode& rhs) const{
        return rhs.d < d;
    }
};
struct Edge{
    int from, to, dist;
};
int n, m;
vector<int> G[maxn];
vector<Edge> edges;
bool done[maxn];
int d[maxn][maxc], p[maxn];
int f[maxn];
void init(){
    edges.clear();
```

```
    for(int i = 0; i <= n; i ++){
        G[i].clear();
    }
}
void addedge(int from, int to, int dist){
    edges.push_back((Edge){from, to, dist});
    edges.push_back((Edge){to, from, dist});
    int m = edges.size();
    G[from].push_back(m - 2);
    G[to].push_back(m - 1);
}
int Dijkstra(int s, int t, int cap){
    priority_queue<HeapNode> q;
    for(int i = 0; i <= n; i ++){
        for(int j = 0; j <= cap; j ++){
            d[i][j] = inf;
        }
    }
    d[s][0] = 0;
    q.push((HeapNode){d[s][0], s, 0});
    while(!q.empty()){
        HeapNode x = q.top(); q.pop();
        int u = x.u;
        if(u == t) return x.d;
        //每次加一单位的油扩展结点
        int fuel = x.fuel + 1;
        int cost = x.d + p[u];
        if(fuel <= cap && d[u][fuel] > cost){
            q.push((HeapNode){cost, u, fuel});
            d[u][fuel] = cost;
            //printf("u = %d, fuel = %d\n", u, fuel);
        }
        int SIZE = G[u].size();
        for(int i = 0; i < SIZE; i ++){
            Edge& e = edges[G[u][i]];
            int v = e.to;
            if(x.fuel >= e.dist && d[v][x.fuel - e.dist] > x.d){
                d[v][x.fuel - e.dist] = x.d;
                q.push((HeapNode){d[v][x.fuel - e.dist], v, x.fuel - e.dist});
                //printf("v = %d, fuel = %d\n", v, x.fuel - e.dist);
            }
        }
    }
    return -1;
}

int main()
{
    while(scanf("%d%d", &n, &m) != EOF){
        for(int i = 0; i < n; i ++) scanf("%d", &p[i]);
        while(m --){
            int u, v, d;
```

```
            scanf("%d%d%d", &u, &v, &d);
            addedge(u, v, d);
        }
        int q;
        scanf("%d", &q);
        while(q --){
            int c, s, e;
            scanf("%d%d%d", &c, &s, &e);
            int ans = Dijkstra(s, e, c);
            if(ans != -1) printf("%d\n", ans);
            else printf("impossible\n");
        }
    }
    return 0;
}
```

**删除最短路上的一条边再求最短路**

```
//要删除一条最短路上的边使从 1 到 n 的时间最长甚至不可达。
//枚举每条边，然后找最坏情况。

struct Edge{
    int from, to, weight;
};
int n, m;
int inq[maxn], d[maxn];
int pre[maxn], pe[maxn]; //pre[]是当前点在最短路中的前置点，pe[]存的是边的编号
vector<int> G[maxn];
vector<Edge> edges;
void init(){
    edges.clear();
    for(int i = 0; i <= n; i ++){
        G[i].clear(); pre[i] = -1;
    }
}
void addedge(int from, int to, int weight){
    edges.push_back((Edge){from, to, weight});
    edges.push_back((Edge){to, from, weight});
    int m = edges.size();
    G[from].push_back(m - 2);
    G[to].push_back(m - 1);
}
int spfa(int s, int t, int pt, int num){ //pt 为 2 代表删边后求最短路
    queue<int> q;
    memset(inq, false, sizeof inq);
    for(int i = 0; i <= n; i ++) d[i] = inf;
    inq[s] = true; d[s] = 0;
    q.push(s);
    while(!q.empty()){
        int u = q.front(); q.pop();
        inq[u] = false;
        for(int i = 0; i < G[u].size(); i ++){
            Edge& e = edges[G[u][i]];
```

```
                int v = e.to;
                if(num == G[u][i]) continue;
                if(d[u] + e.weight < d[v]){
                    d[v] = d[u] + e.weight;
                    if(pt == 1) {
                        pre[v] = u;
                        pe[v] = G[u][i];
                    }
                    if(!inq[v]){
                        q.push(v);
                        inq[v] = true;
                    }
                }
            }
        }
        if(d[t] == inf) return -1;
        return d[t];
}
int main()
{
        int T;
        scanf("%d", &T);
        while(T --){
            scanf("%d%d", &n, &m);
            init();
            while(m --){
                int u, v, w;
                scanf("%d%d%d", &u, &v, &w);
                addedge(u, v, w);
            }
            int s = 1, t = n;
            int ans = spfa(s, t, 1, 2 * m + 1);
            if(ans == -1) {
                printf("-1\n");
                continue;
            }
            int u = t;
            while(u != s){
                int res = spfa(s, t, 2, pe[u]);
                if(res == -1){ //如果删除某条边不可达，则 ans = -1
                    ans = -1;
                    break;
                }
                ans = max(ans, res);
                u = pre[u];
            }
            printf("%d\n", ans);
        }
        return 0;
}
```

**二分枚举+最短路**

```
SPFA sp;
```

```
int N, M;
int Tot;
int maxh, ans;
int main()
{
    int Tot;
    scanf("%d", &Tot);
    for(int ac = 1; ac <= Tot; ac ++){
        scanf("%d%d", &N, &M);
        sp.init(N);
        if(ac > 1) printf("\n");
        maxh = 0;
        int u, v, h, d;
        while(M --){
            scanf("%d%d%d", &u, &v, &h);
            maxh = max(h, maxh);
            d = 1;
            sp.height[sp.m] = h;
            sp.AddEdge(u, v, d);
            sp.height[sp.m] = h;
            sp.AddEdge(v, u, d);
        }
        int L = 0, R = maxh, mid;
        while(L <= R){
            mid = (L + R) / 2;
            ans = sp.spfa(1, N, mid);
            if(ans == -1) R = mid - 1;
            else L = mid + 1;
        }
        printf("Scenario #%d:\n", ac);
        printf("%d\n", R);
    }
    return 0;
}
```

**分层最短路**

```
/*
N 个点，然后有 N 层，要假如 2*N 个点。
总共是 3*N 个点。
点 1~N 就是对应的实际的点 1~N.   要求的就是 1 到 N 的最短路。
然后点 N+1 ~ 3*N  是 N 层拆出出来的点。
然后用优先队列优化的 Dijkstra 就可以搞出最短路了
*/
Dijkstra sp;
void Prepare() {
    scanf("%d%d%d", &N, &M, &C);
    for(int i = 1; i <= N; i ++) {G[i].clear();}
    for(int i = 1; i <= N; i ++) {
        scanf("%d", &sp.layer[i]);
        G[sp.layer[i]].push_back(i);
    }
    sp.init(3*N);
    //printf("cnt = %d\n", cnt);
```

```
        for(int i = 1; i <= N; i ++) {
            int sz = G[i].size();
            for(int j = 0; j < sz; j ++) {
                int v = G[i][j];
                //printf("i = %d, v = %d\n", i, v);
                sp.AddEdge(2*N + i, v, 0); //v 属于第 i 层
                sp.AddEdge(v, i + N, 0);
            }
        }
        /***注意上面层的入点是下面层的出点********/
        for(int i = 2; i <= N; i ++) {
            sp.AddEdge(N + i, 2*N + (i-1), C); //第 i 层到第 i-1 层
            sp.AddEdge(N + (i-1), 2*N + i, C); //第 i-1 层到第 i 层
        }
    int u, v, d;
    while(M --){
        scanf("%d%d%d", &u, &v, &d);
        sp.AddEdge(u, v, d);
        sp.AddEdge(v, u, d);
    }
}
int main()
{
    //freopen("input.txt", "r", stdin);
    int Tot;
    scanf("%d", &Tot);
    for(int ac = 1; ac <= Tot; ac ++){
        Prepare();
        printf("Case #%d: ", ac);
        sp.dijkstra(1, N);
        ans = sp.d[N];
        if(ans < inf) printf("%d\n", ans);
            else printf("-1\n");
    }
    return 0;
}
```

# 生成树&并查集

**种类并查集（POJ 1182）**

```
int pa[maxn], ra[maxn];
const int same = 0; //同类
const int enemy = 1; //被父节点吃
const int food = 2; //吃父节点
int n, m;
int findset(int x){
    if(pa[x] == x) return x;
    int tmp = pa[x];
    pa[x] = findset(pa[x]);
    ra[x] = (ra[tmp] + ra[x]) % 3;
    return pa[x];
}
void init()
```

```
{
    for(int i = 0; i <= n; i ++){
        pa[i] = i; ra[i] = same;
    }
}
void merge(int x, int y, int u, int v, int d){
    pa[v] = u;
    ra[v] = (3 - ra[y] + (d - 1) + ra[x]) % 3;
    // 3 - ra[y]是 v 和 y 的关系  d - 1 是  y 和 x 的关系
    // ra[x]是 x 和 u 的关系，得出 v 和 u 的关系
}
int main()
{
    int d, x, y;
    int ans;
    scanf("%d%d", &n, &m);
        init();
        ans = 0;
        while(m --){
            scanf("%d%d%d", &d, &x, &y);
            if(x > n || y > n){
                ans ++; continue;
            }
            if(d == 2 && x == y) {
                ans ++; continue;
            }
            int u = findset(x), v = findset(y);

            if(u != v){
                merge(x, y, u, v, d);
            }
            else {
                if(d == 1){
                    if(ra[x] != ra[y]) ans ++;
                }
                if(d == 2){
                    if((ra[y] + 3 - ra[x]) % 3 != 1) ans ++;
                }
            }
        }
        printf("%d\n", ans);
    return 0;
}
```

**并查集求集合个数（倒过来用）**

```
int pa[maxn], ans[maxn], e[maxn];
bool mark[maxn];
struct Edge{
    int u, v;
}edges[maxn];

int findset(int x) {
    return pa[x] == x ? x : (pa[x] = findset(pa[x]));
```

```
}
int N, M, Q;
int main()
{
    while(scanf("%d%d", &N, &M) != EOF) {
        for(int i = 1; i <= N; i ++) pa[i] = i;
        for(int i = 1; i <= M; i ++) {
            scanf("%d%d", &edges[i].u, &edges[i].v);
            mark[i] = false;
        }
        scanf("%d", &Q);
        for(int i = 1; i <= Q; i ++) {
            int id;
            scanf("%d", &id);
            mark[id] = true;
            e[i] = id;
        }
        int cnt = N;
        for(int i = 1; i <= M; i ++) {
            if(!mark[i]) {
                int x = findset(edges[i].u), y = findset(edges[i].v);
                if(x != y) {
                    pa[x] = y;
                    cnt --;
                }
            }
        }
        for(int i = Q; i >= 1; i --) {
            ans[i] = cnt;
            int x = findset(edges[e[i]].u), y = findset(edges[e[i]].v);
            if(x != y) {
                pa[x] = y;
                cnt --;
            }
        }
        for(int i = 1; i < Q; i ++)
            printf("%d ", ans[i]);
        printf("%d\n", ans[Q]);
    }
    return 0;
}
```

**种类并查集+计数（POJ 1988）**

```
int pa[maxn], cnt[maxn], pl[maxn];
int n, p;
int findset(int x){
    if(pa[x] == x) return x;
    int tmp = pa[x];
    pa[x] = findset(pa[x]);
    pl[x] += pl[tmp];
    return pa[x];

}
```

```
void init(){
    for(int i = 0; i < maxn; i ++){
        pa[i] = i; cnt[i] = 1; pl[i] = 0;
    }
}
void merge(int u, int v){
    pa[v] = u;
    pl[v] += cnt[u];
    cnt[u] += cnt[v];
}
int main()
{
    while(scanf("%d", &p) != EOF){
        init();
        while(p --){
            char op[5];
            scanf("%s", op);
            int x, y;
            if(op[0] == 'M'){
                scanf("%d%d", &x, &y);
                int u = findset(x), v = findset(y);
                if(u != v){
                    merge(u, v);
                }
            }
            else {
                scanf("%d", &x);
                int u = findset(x);
                printf("%d\n", cnt[u] - pl[x] - 1);
            }
        }
    }
    return 0;
}
```

## Kruskal

**位运算枚举树中的点重新构图**

```
int pa[maxn];
int findset(int x) {
  return pa[x] != x ? (pa[x] = findset(pa[x])) : x;
}
struct Edge {
  int from, to, dist;
  Edge(){}
  Edge(int _u, int _v, int _c)
  :from(_u), to(_v), dist(_c){}
  bool operator < (const Edge& rhs) const {
    return dist < rhs.dist;
  }
};
Edge e[maxm];
int MST(int n, int m) {
```

```
    sort(e, e + m);
    for(int i = 0; i < n; i ++) pa[i] = i;
    int cnt = 0, mst = 0;
    for(int i = 0; i < m; i ++) {
        int u = e[i].from, v = e[i].to;
        int x = findset(u), y = findset(v);
        int dist = e[i].dist;
        if(x != y) {
            pa[y] = x;
            mst += dist;
            if(++ cnt == n - 1) return mst;
        }
    }
    return INF;
}
int n, m, K;
Edge ed[maxm];
int pl[maxn];
int id[maxn];
int main() {
    int Tot;
    scanf("%d", &Tot);
    for(int ac = 1; ac <= Tot; ac ++) {
        scanf("%d%d%d", &N, &M, &K);
        for(int i = 0; i < N; i ++) scanf("%d", &pl[i]);
        for(int i = 0; i < M; i ++) {
            int u, v, c;
            scanf("%d%d%d", &u, &v, &c);
            ed[i] = Edge(u-1, v-1, c);
        }
        int mst = pl[0];
        int bestn = 1, ans = pl[0];
        for(int S = 0; S < (1<<(N-1)); S ++) {
            n = 1; m = 0;
            memset(id, -1, sizeof(id));
            id[0] = 0;
            for(int i = 0; i < N-1; i ++)
                if((S>>i) & 1) id[i+1] = n ++;
            for(int i = 0; i < M; i++) {
                int u = id[ed[i].from];
                int v = id[ed[i].to];
                if(u >= 0 && v >= 0)
                    e[m ++] = Edge(u, v, ed[i].dist);
            }
            if(MST(n, m) <= K) {
                int tot = pl[0];
                for(int i = 0; i < N-1; i ++)
                    if((S>>i) & 1) tot += pl[i+1];
                if(tot > ans) {
                    ans = tot; bestn = n;
                }
            }
```

```
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

**K 度最小生成树**

```
const int maxn = 30;
const int inf = 0x3f3f3f3f;
struct HeapNode{
        int u, d;
        HeapNode(){}
        HeapNode(int _u,int _d):u(_u), d(_d){}
        bool operator < (const HeapNode& rhs) const{
                return rhs.d < d;
        }
};
map<string, int> mp;
int g[maxn][maxn]; //邻接矩阵存图
int lowc[maxn];
int col[maxn]; //记录每个点所属生成树
int pre[maxn]; //记录生成树中的父亲节点
int fst[maxn]; //记录每个生成树到根的最小距离的点
int max_side[maxn];
int n, m, k;
int Prim(int src, int id) {
    priority_queue<HeapNode> Q;
    while(!Q.empty()) Q.pop();
    lowc[src] = 0;
    Q.push(HeapNode(src, 0));
    int ans = 0;
    while(!Q.empty()) {
        HeapNode x = Q.top(); Q.pop();
        int u = x.u;
        if(!col[u]) {
            col[u] = id;
            ans += lowc[u];
            for(int v = 1; v < n; v ++) {
                if(!col[v] && g[u][v] != 0 && lowc[v] > g[u][v]) {
                    pre[v] = u;
                    lowc[v] = g[u][v];
                    Q.push(HeapNode(v, lowc[v]));
                }
            }
        }
    }
    return ans;
}
void update(int u, int fa, int maxside) {
    max_side[u] = max(maxside, g[u][fa]);
    for(int v = 1; v < n; v ++) {
        if(v != fa && g[u][v] != 0 && (pre[u] == v || pre[v] == u)) {
            update(v, u, max_side[u]);
```

```
        }
    }
}
void solve() {
    int v, ans, cnt;
    for(v = 0; v < n; v ++) {
        lowc[v] = inf;
        col[v] = pre[v] = fst[v] = 0;
    }
    ans = 0, cnt = 1;
    for(v = 1; v < n; v ++) {
        if(!col[v]) {
            ans += Prim(v, cnt ++);
        }
    }
    for(v = 1; v < n; v ++) {
        int id = col[v];
        if(g[0][v] != 0 && (!fst[id] || g[0][v] < g[0][fst[id]])) {
            fst[id] = v;
        }
    }
    for(int i = 1; i < cnt; i ++) {
        ans += g[0][fst[i]];
        g[0][fst[i]] = g[fst[i]][0] = 0;
        update(fst[i], 0, 0);
    }
    k = k - cnt + 1;
    while(k --) {
        int tmp = 0;
        for(v = 1; v < n; v ++) {
            if(g[0][v] != 0 && (tmp == 0 || max_side[tmp] - g[0][tmp]
                                        < max_side[v] - g[0][v]))
                tmp = v;
        }
        if(max_side[tmp] <= g[0][tmp]) break;
        ans = ans - max_side[tmp] + g[0][tmp];
        g[0][tmp] = g[tmp][0] = 0;
        int p = 0;
        for(v = tmp; pre[v] != 0; v = pre[v]) {
            if(p == 0 || g[p][pre[p]] < g[v][pre[v]])
                p = v;
        }
        pre[p] = 0;
        update(tmp, 0, 0);
    }
    printf("Total miles driven: %d\n", ans);
}
int main(){
    //freopen("input.txt","r",stdin);
    char a[20], b[20];
    int c;
    while(scanf("%d", &m) != EOF) {
```

```
        mp["Park"] = 0;
        n = 1;
        memset(g, 0, sizeof(g));
        while(m --) {
            scanf("%s %s %d",a ,b, &c);
            if(!mp.count(a)) mp[a] = n++;
            if(!mp.count(b)) mp[b] = n++;
            int u = mp[a], v = mp[b];
            if(!g[u][v] || g[u][v] > c)
                g[u][v] = g[v][u] = c;
        }
        scanf("%d", &k);
        solve();
    }
    return 0;
}
```

**斯坦纳树（HDU 4085）**

```
/*          斯坦纳树
给你 n,m,k ，分别表示有 n 个点，m 条边，每条边有一个权值，
表示修复这条边需要的代价，从前 k 个点中任取一个使其和
后 k 个点中的某一个点，通过边连接，并且必须是一一对应，
问最小的代价是多少。
*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <queue>
using namespace std;
const int inf = 0x3f3f3f3f;
const int maxn = 60;
struct Edge{
    int from, to, dist;
    Edge(){}
    Edge(int _u, int _v, int _d)
    :from(_u), to(_v), dist(_d){}
};
Edge ed[maxn*maxn];
int head[maxn], nxt[maxn*maxn], m;
int N, M, K, mask;
int st[maxn], vis[maxn][1<<11];
int d[maxn][1<<11], dp[1<<11];
void AddEdge(int from, int to, int dist) {
    nxt[m] = head[from]; ed[m] = Edge(from, to, dist);
    head[from] = m ++;
}
bool check(int S) {
    int res = 0;
    for(int i = 0; S; i++, S >>= 1){
        res += (S&1)*(i<K ? 1 : -1);
    }
    return res == 0;
}
```

```
void init(int n){
    m = 0; mask = (1<<(2*K))-1;
    memset(st, 0, sizeof(st[0])*(n+1));
    memset(vis, 0, sizeof(vis));
    memset(head, -1, sizeof(head[0])*(n+1));
    for(int i = 1; i <= n; i ++) {
        for(int j = 0; j <= mask; j ++) d[i][j] = inf;
    }
}
queue<int> Q;
void SPFA() {
    while(!Q.empty()) {
        int u = Q.front()/10000, S = Q.front()%10000;
        vis[u][S] = 0;
        Q.pop();
        for(int i = head[u]; i != -1; i = nxt[i]) {
            int v = ed[i].to, w = ed[i].dist;
            if(d[v][S|st[v]] > d[u][S] + w) {
                d[v][S|st[v]] = d[u][S] + w;
                if((S|st[v]) == S && !vis[v][S]) {
                    Q.push(v*10000 + S);
                    vis[v][S] = 1;
                }
            }
        }
    }
}
int main()
{
    int Tot;
    scanf("%d", &Tot);
    while(Tot --) {
        scanf("%d%d%d", &N, &M, &K);
        init(N);
        int u, v, w;
        while(M --) {
            scanf("%d%d%d", &u, &v, &w);
            AddEdge(u, v, w);
            AddEdge(v, u, w);
        }
        for(int i = 1; i <= K; i ++) {
            st[i] = 1<<(i-1);
            d[i][st[i]] = 0;
            st[N-i+1] = (1<<(K + (i-1)));
            d[N-i+1][st[N-i+1]] = 0;
        }
        for(int S = 0; S <= mask; S ++){
            for(int i = 1; i <= N; i ++) {
                for(int p = (S-1)&S; p; p = (p-1)&S)
                    d[i][S] = min(d[i][S], d[i][p|st[i]] + d[i][(S-p)|st[i]]);
                if(d[i][S]<inf && !vis[i][S])
                    Q.push(i*10000 + S), vis[i][S] = 1;
```

```
            }
            SPFA();
        }
        for(int S = 0; S <= mask; S ++) {
            dp[S] = inf;
            for(int i = 1; i <= N; i ++)
                dp[S] = min(dp[S], d[i][S]);
        }
        for(int S = 1; S <= mask; S ++) {
            if(check(S)) {
                for(int p = (S-1)&S; p; p = (p-1)&S) {
                    if(check(p))
                        dp[S] = min(dp[S], dp[p] + dp[S - p]);
                }
            }
        }
        if(dp[mask] >= inf) puts("No solution");
        else printf("%d\n", dp[mask]);
    }
    return 0;
}
```

**有向树形图（UVA 11865）**

```
typedef int type;
const int maxn = 100 + 10;
const int maxm = 10000 + 10;
const int inf = INT_MAX;
struct Edge{
    int from, to, b;
    type dist;
    Edge(){}
    Edge(int _from, int _to, int _b, type _dist)
    :from(_from), to(_to), b(_b), dist(_dist){}
    bool operator < (const Edge& rhs) const {
        return b > rhs.b;
    }
}E[maxm];
struct DMST {
    int pre[maxn], ID[maxn], vis[maxn];
    type In[maxn];
    Edge edges[maxm];
    type Directed_MST(int root, int n, int m) {
        type ans = 0;
        while(true) {
            //找最小入边
            for(int i = 0; i < n; i ++) In[i] = inf;
            for(int i = 0; i < m; i ++) {
                int u = edges[i].from, v = edges[i].to;
                if(edges[i].dist < In[v] && u != v) {
                    In[v] = edges[i].dist; pre[v] = u;
                }
            }
            for(int i = 0; i < n; i ++) {
```

```
    if(i == root) continue;
    if(In[i] == inf) {return -1;}
}
//找环
int cntnode = 0;
memset(ID, -1, sizeof(ID));
memset(vis, -1, sizeof(vis));
In[root] = 0;
for(int i = 0; i < n; i ++) {
    ans += In[i];
    int v = i;
    while(vis[v] != i && ID[v] == -1 && v != root) {
        vis[v] = i; v = pre[v];
    }
    if(v != root && ID[v] == -1) {
        for(int u = pre[v]; u != v; u = pre[u]) {
            ID[u] = cntnode;
        }
        ID[v] = cntnode ++;
    }
}
if(cntnode == 0) break;
for(int i = 0; i < n; i ++) {
    if(ID[i] == -1) {
        ID[i] = cntnode ++;
    }
```

```
        }
        //缩点，重新标记
        for(int i = 0; i < m; i ++) {
            int v = edges[i].to;
            edges[i].from = ID[edges[i].from];
            edges[i].to = ID[edges[i].to];
            if(edges[i].from != edges[i].to) {
                edges[i].dist -= In[v];
            }
        }
        n = cntnode;
        root = ID[root];
    }
    return ans;
  }
};
DMST dem;
int N, M, C;
int Tot;
bool check(int cnt) {
    //printf("cnt = %d\n", cnt);
    for(int i = 0; i < cnt; i ++) {
        dem.edges[i] = E[i];
    }
    int ans = dem.Directed_MST(0, N, cnt);
    //printf("b = %d, ans = %d\n", E[cnt].b, ans);
```

```
    if(ans == -1 || ans > C) return false;
    return true;
}
int main()
{
    //freopen("input.txt", "r", stdin);
    scanf("%d", &Tot);
    for(int ac = 1; ac <= Tot; ac ++) {
        scanf("%d%d%d", &N, &M, &C);
        int u, v, b, c;
        for(int i = 0; i < M; i ++) {
            scanf("%d%d%d%d", &u, &v, &b, &c);
            E[i] = Edge(u, v, b, c);
        }
        sort(E, E + M);
        int l = 1, r = M, mid;
        int ans = -1;
        while(l <= r) {
            mid = (l + r) >> 1;
            if(check(mid)) {
                r = mid - 1; ans = E[mid - 1].b;
            }
            else l = mid + 1;
        }
        if(ans >= 0) printf("%d kbps\n", ans);
        else printf("streaming not possible.\n");
    }
    return 0;
}
```

**最小直径生成树（URAL 1569）**

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <algorithm>
#include <vector>
#include <queue>
using namespace std;

typedef pair<int, int> pii;
const int inf = 0x3f3f3f3f;
const int maxn = 200 + 10;
int w[maxn][maxn];
int d[maxn][maxn];
int pre[maxn];
bool inq[maxn];
int dp[maxn], ans;
int cu, cv;
void init(int n) {
    for(int i = 1; i <= n; i ++) {
        for(int j = 1; j <= n; j ++) {
            w[i][j] = inf;
        }
```

```
            w[i][i] = 0;
        }
}
void Floyd(int n) {
        for(int k = 1; k <= n; k ++)
        for(int i = 1; i <= n; i ++)
        for(int j = 1; j <= n; j ++) {
            if(d[i][j] > d[i][k] + d[k][j])
                d[i][j] = d[i][k] + d[k][j];
        }
}
void update(int u, int v, int n) {
        vector<pii> vt, tk;
        for(int i = 1; i <= n; i ++) {
            vt.push_back(make_pair(d[u][i], d[v][i]));
        }
        sort(vt.begin(), vt.end());
        /*按照 d[u][i]  从小到大排，考虑 d[u][i] < d[u][j]
        && d[v][i] > d[u][j] (i < j)*/
        int sz = vt.size();
        for(int i = 0;i < sz; i ++) {
            while(!tk.empty() && tk.back().second <= vt[i].second) {
                tk.pop_back();
            }
            tk.push_back(vt[i]);
        }
```

```
int D = inf;
int tmp;
if(tk.size() == 1) {
        if(tk[0].first < tk[0].second) {
            tmp = 0;
            D = tk[0].first << 2;
        }else {
            tmp = w[u][v] << 1;
            D = tk[0].second << 2;
        }
} else {
        sz = tk.size();
        for(int i = 1; i < sz; i ++) {
            if(D > ((w[u][v] + tk[i - 1].first + tk[i].second) << 1)){
                tmp = w[u][v] + tk[i].second - tk[i - 1].first;
                D = ((w[u][v] + tk[i - 1].first + tk[i].second) << 1);
            }
        }
}
if(D < ans) {
        cu = u;
        cv = v;
        ans = D;
        dp[u] = tmp;
        dp[v] = (w[u][v] << 1) - tmp;
}
```

```
}
vector<pii> edges;
void Spfa(int n) {
	queue<int> q;
	memset(pre, -1, sizeof(pre));
	memset(inq, false, sizeof(inq));
	for(int i = 1; i <= n; i ++) {
		if(i != cu && i != cv) {
			dp[i] = inf;
		}
	}
	inq[cu] = inq[cv] = true;
	q.push(cu); q.push(cv);
	while(!q.empty()) {
		int u = q.front(); q.pop();
		inq[u] = false;
		for(int v = 1; v <= n; v ++) {
			if(w[u][v] != inf) {
				if(dp[v] > dp[u] + (w[u][v] << 1)) {
					dp[v] = dp[u] + (w[u][v] << 1);
					pre[v] = u;
					if(!inq[v]) {
						q.push(v);
						inq[v] = true;
					}
				}
			}
		}
		edges.clear();
		for(int u = 1; u <= n; u ++) {
			if(pre[u] != -1) {
				edges.push_back(make_pair(min(u, pre[u]), max(u, pre[u])));
			}
		}
		if(cv != cu) {
			edges.push_back(make_pair(min(cv, cu), max(cv, cu)));
		}
	}
}
int N, M;
int main()
{
	//freopen("input.txt", "r", stdin);
	while(scanf("%d%d", &N, &M) != EOF) {
		init(N);
		int a, b;
		while(M --) {
			scanf("%d%d", &a, &b);
			w[a][b] = w[b][a] = min(1, w[a][b]);
		}
		memcpy(d, w, sizeof(d));
```

```
        Floyd(N);
        ans = inf;
        for(int i = 1; i <= N; i ++) {
            for(int j = 1; j <= N; j ++) {
                if(w[i][j] != inf) {
                    update(i, j, N);
                }
            }
        }
        //printf("%d\n", ans / 2);
        Spfa(N);
        sort(edges.begin(), edges.end());
        for(vector<pii> ::iterator it = edges.begin(); it != edges.end(); it ++) {
            printf("%d %d\n", it->first, it->second);
        }
    }
    return 0;
}
```

**次小生成树（UVA 10462）**

```
struct Edge{
  int from, to, dist;
  Edge(){}
  Edge(int _from, int _to, int _dist)
  :from(_from), to(_to), dist(_dist){}
  bool operator < (const Edge& rhs) const{
    return dist < rhs.dist; //在用到优先队列时要换成 >
  }
};
const int inf = 0x3f3f3f3f;
const int maxn = 100 + 10;
const int logmaxn = 32;
const int maxm = 200 + 10;
int n, m;
int pa[maxn];
int maxcost[maxn][logmaxn], fa[maxn];
int anc[maxn][logmaxn];
int L[maxn], cost[maxn];
bool vis[maxm];
Edge e[maxm];
vector<int> G[maxn], C[maxn];
int findset(int x){
  return pa[x] == x ? x : (pa[x] = findset(pa[x]));
}
void dfs(int u, int pa, int level) {
  L[u] = level;
  int sz = G[u].size();
  for(int i = 0; i < sz; i ++)
  {
    int v = G[u][i];
    if(v != pa) {
      fa[v] = u, cost[v] = C[u][i];
      dfs(v, u, level + 1);
```

```
        }
    }
}
void preprocess(){
    for(int i = 0; i < n; i ++){
        anc[i][0] = fa[i]; maxcost[i][0] = cost[i];
    for(int j = 1; (1 << j) < n; j ++)
        anc[i][j] = -1;
    }
    for(int j = 1; (1 << j) < n; j ++)
        for(int i = 0; i < n; i ++)
            if(anc[i][j - 1] != -1) {
                int a = anc[i][j - 1];
                anc[i][j] = anc[a][j - 1];
                maxcost[i][j] = max(maxcost[i][j - 1], maxcost[a][j - 1]);
            }
}
int query(int p, int q){
    int log;
    if(L[p] < L[q]) swap(p, q);
    for(log = 1; (1 << log) <= L[p]; log ++); log --;
    int ans = -inf;
    for(int i = log; i >= 0; i --) {
        if(L[p] - (1 << i) >= L[q]) {
            ans = max(ans, maxcost[p][i]); p = anc[p][i];
        }
    }
    if(p == q) return ans;
    for(int i = log; i >= 0; i --) {
        if(anc[p][i] != -1 && anc[p][i] != anc[q][i]) {
            ans = max(ans, maxcost[p][i]); p = anc[p][i];
            ans = max(ans, maxcost[q][i]); q = anc[q][i];
        }
    }
    ans = max(ans, cost[p]);
    ans = max(ans, cost[q]);
    return ans;
}
int Tot;
int main()
{
    //freopen("input.txt", "r", stdin);
    scanf("%d", &Tot);
    for(int ac = 1; ac <= Tot; ac ++) {
        scanf("%d%d", &n, &m);
        for(int i = 0; i < n; i ++){
            pa[i] = i;
            G[i].clear(); C[i].clear();
        }
        for(int i = 0; i < m; i ++){
            int a, b, c;
            scanf("%d%d%d", &a, &b, &c);
```

```
    a --; b --;
    e[i] = Edge(a, b, c);
}
sort(e, e + m);
int ans = 0, cnt = 1;
memset(vis, false, sizeof vis);
for(int i = 0; i < m; i ++){
    int x = e[i].from, y = e[i].to, d = e[i].dist;
    int u = findset(x), v = findset(y);
    if(u != v){
        vis[i] = true; pa[v] = u; ans += d;
        G[x].push_back(y); C[x].push_back(d);
        G[y].push_back(x); C[y].push_back(d);
        cnt ++;
    }
}
printf("Case #%d : ", ac);
if(cnt < n) {
    printf("No way\n");
    continue;
}
dfs(0, -1, 0);
preprocess();
int smst = inf;
for(int i = 0; i < m; i ++) {
    if(!vis[i]) {
            int u = e[i].from, v = e[i].to;
            smst = min(ans + e[i].dist - query(u, v), smst);
        }
    }
    if(smst == inf) printf("No second way\n");
    else printf("%d\n", smst);
  }
  return 0;
}
```

**最小生成树最大边最大**

```
typedef pair<int, int> pii;
const double inf = 0x3f3f3f3f;
const int maxn = 2000 + 10;
const int maxm = 1000000 + 100;
struct Edge{
    int from, to, dist;
    Edge(){}
    Edge(int _from, int _to, int _dist)
    :from(_from), to(_to), dist(_dist){}
    bool operator < (const Edge& rhs) const {
        return dist < rhs.dist;
    }
};
int N, M;
int mst, trz;
bool intree[maxm];
```

```
Edge e[maxm];
int dp[maxn][maxn];
int ca[maxn], cb[maxn];
int cacnt, cbcnt;
int res;
int main()
{
    //freopen("input.txt", "r", stdin);
    int k, i, j;
    int u, v, w;
    while(scanf("%d%d", &N, &M) != EOF) {
        for(i = 0; i < M; i ++) {
            scanf("%d%d%d", &e[i].from, &e[i].to, &e[i].dist);
            intree[i] = 0;
        }
        sort(e, e + M);
        for(i = 1; i <= N; i ++) {
            for(j = 1; j <= N; j ++) {
                dp[i][j] = -1;
            }
            dp[i][i] = 0;
        }
        mst = trz = 0;
        for(k = 0; k < M; k ++) {
            u = e[k].from; v = e[k].to; w = e[k].dist;
            if(dp[u][v] != -1) continue; //已经在一个块中
            intree[k] = 1;
            cacnt = cbcnt = 0;
            for(i = 1; i <= N; i ++) {
                if(dp[i][u] != -1) { //找成环的边
                    ca[cacnt ++] = i;
                }
                if(dp[i][v] != -1) {
                    cb[cbcnt ++] = i;
                }
            }
            int ut, vt;
            for(i = 0; i < cacnt; i ++) {
                for(j = 0; j < cbcnt; j ++) {
                    ut = ca[i]; vt = cb[j];
                    if(ut != vt) { //构成环的最大权值
                        dp[ut][vt] = dp[vt][ut] = max(w, max(dp[ut][u], dp[vt][v]));
                    }
                }
            }
            mst += w;
            trz ++;
            if(trz == N - 1) break;
        }
        if(trz < N - 1) {
            puts("disconnected"); continue;
        }
```

```
        res = mst;
        for(i = 0; i < M; i ++) {
            if(!intree[i]) {
                w = mst - e[i].dist - dp[e[i].from][e[i].to];
                res = min(res, w);
            }
        }
        printf("%d\n", res);
    }
    return 0;
}
```

# 2-sat

**建图**

用 2*u 表示选择 u，2*u+1 表示不选择 u

对于有限制（如 u，v 不能同时出现）

如果选 u，不选 v，那么 2*u-->2*v+1 连一条边，选 v 不选 u 同理

更广泛一点，对于 a 来表示 u 的一种情况，b 表示 v 的一种情况

则 a-->b^1 连一条边，由对称性 b-->a^1 连一条边

**Dfs 版**

```
struct TwoSAT {
    int n, m;
    int head[maxn * 2];
    int to[maxm * 2];
    int next[maxm * 2];
    bool mark[maxn * 2];
    int S[maxn * 2], c;
    void init(int n) {
        this->n = n;
        m = 0;
        memset(head, -1, sizeof(head[0]) * (2*n + 1));
        memset(mark, false, sizeof(mark[0]) * (2*n + 1));
    }
    bool dfs(int x) {
        if(mark[x ^ 1]) return false;
        if(mark[x]) return true;
        mark[x] = true;
        S[c ++] = x;
        for(int i = head[x]; i != -1; i = next[i]) {
            if(!dfs(to[i])) return false;
        }
        return true;
    }
    /*连边直接上条件*/
    void AddEdge(int x, int y) {
        next[m] = head[x]; to[m] = y; head[x] = m ++;
    }
    bool tsat() {
        for(int i = 0; i < n * 2; i += 2) {
            if(!mark[i] && !mark[i + 1]) {
                c = 0;
                if(!dfs(i)) {
```

```
                while(c > 0) mark[S[-- c]] = false;
                if(!dfs(i + 1)) return false;
            }
        }
    }
    return true;
  }
};
```

**Tarjan 版**

```
struct Edge{
  int to, next;
  Edge(){}
  Edge(int _to, int _next) : to(_to), next(_next){}
};
struct Tarjan {
  int m;
  int head[maxn];
  Edge edges[maxm];
  int pre[maxn], sccno[maxn], lowlink[maxn], scc_cnt, dfs_clock;
  int S[maxn], top;
  int cnt[maxn]; //记录每个强连通分量中点的个数
  void init() {
    m = 0;
    memset(head, -1, sizeof(head));
    memset(cnt, 0, sizeof(cnt));
  }
void AddEdge(int from, int to) {
  int next = head[from];
  head[from] = m;
  edges[m ++] = Edge(to, next);
}
void dfs(int u) {
  pre[u] = lowlink[u] = ++ dfs_clock;
  S[top ++] = u;
  for(int i = head[u]; i != -1; i = edges[i].next) {
    int v = edges[i].to;
    if(!pre[v]) {
      dfs(v);
      lowlink[u] = min(lowlink[u], lowlink[v]);
    }else if(!sccno[v]) {
      lowlink[u] = min(lowlink[u], pre[v]);
    }
  }
  if(pre[u] == lowlink[u]) {
    scc_cnt ++;
    while(true) {
      cnt[scc_cnt] ++;
      int x = S[-- top];
      sccno[x] = scc_cnt;
      if(x == u) break;
    }
  }
```

```
    }
    void find_scc(int n) {
        dfs_clock = top = scc_cnt = 0;
        memset(pre, 0, sizeof(pre));
        memset(sccno, 0, sizeof(sccno));
        for(int i = 0; i < n; i ++) {
            if(!pre[i]) dfs(i);
        }
    }
};
Tarjan sat;
int N, M;

void Prepare() {
    sat.init();
    int u, v;
    while(M --) {
        scanf("%d%d", &u, &v);
        if(u > 0 && v > 0) {
            sat.AddEdge(2*(u - 1) + 1, 2*(v - 1));
            sat.AddEdge(2*(v - 1) + 1, 2*(u - 1));
        }
        if(u < 0 && v < 0) {
            sat.AddEdge(2*(-u - 1), 2*(-v - 1) + 1);
            sat.AddEdge(2*(-v - 1), 2*(-u - 1) + 1);
        }
        if(u > 0 && v < 0) {
            sat.AddEdge(2*(u - 1) + 1, 2*(-v - 1) + 1);
            sat.AddEdge(2*(-v - 1), 2*(u - 1));
        }
        if(u < 0 && v > 0) {
            sat.AddEdge(2*(-u - 1), 2*(v - 1));
            sat.AddEdge(2*(v - 1) + 1, 2*(-u - 1) + 1);
        }
    }
}
int main()
{
    while(scanf("%d%d", &N, &M) != EOF) {
        Prepare();
        sat.find_scc(N * 2);
        bool ok = true;
        for(int i = 0; i < N * 2; i += 2) {
            if(sat.sccno[i] == sat.sccno[i + 1])
                ok = false;
        }
        if(ok) puts("1");
        else puts("0");
    }
    return 0;
}
```

**2-sat 输出解**

```
TwoSAT sat;
int N;
bool ok;
int S[maxn], T[maxn], D[maxn];
bool check(int s1, int t1, int s2, int t2) {
    if((s1 < s2 && s2 < t1) ||
            (s1 < t2 && t2 < t1) ||
            (s2 <= s1 && t1 <= t2)
    ) return true;
    return false;
}

void Prepare() {
    int u, v;
    sat.init(N);
    ok = true;
    int hh, mm;
    for(int i = 0; i < N; i ++) {
        scanf("%d:%d", &hh, &mm);
        S[i] = hh * 60 + mm;
        scanf("%d:%d %d", &hh, &mm, &D[i]);
        T[i] = hh * 60 + mm;
        if(T[i] < S[i] + D[i]) ok = false;
        for(int j = 0; j < i; j ++) {
            u = 2 * i; v = 2 * j;
            int s1 = S[i], t1 = S[i] + D[i];
            int s2 = T[i] - D[i], t2 = T[i];
            int s3 = S[j], t3 = S[j] + D[j];
            int s4 = T[j] - D[j], t4 = T[j];
            if(check(s1, t1, s3, t3)) {
                sat.AddEdge(u, v ^ 1);
            }
            if(check(s1, t1, s4, t4)) {
                sat.AddEdge(u, v);
            }
            if(check(s2, t2, s3, t3)){
                sat.AddEdge(u ^ 1, v ^ 1);
            }
            if(check(s2, t2, s4, t4)){
                sat.AddEdge(u ^ 1, v);
            }
        }
    }
}
int main()
{
    while(scanf("%d", &N) != EOF) {
        Prepare();
        if(!ok) {
            puts("NO");
            continue;
        }
```

```
        ok = sat.tsat();
        if(!ok) puts("NO");
        else{
            puts("YES");
            for(int i = 0; i < 2 * N; i += 2) {
                if(sat.mark[i]) {
                    printf("%02d:%02d %02d:%02d\n", S[i/2]/60, S[i/2]%60,
                            (S[i/2] + D[i/2])/60, (S[i/2] + D[i/2])%60);
                }
                else printf("%02d:%02d %02d:%02d\n", (T[i/2] - D[i/2])/60,
(T[i/2]-D[i/2])%60,
                            T[i/2]/60, T[i/2]%60);
            }
        }
    }
    return 0;
}
```

**与边相关的 2-sat**

```
TwoSAT sat;
int N, M;
struct Edge{
    int u, v;
};
Edge e[maxn];
bool check(int i, int j) { //判断是否可以共存
    if(e[i].u < e[j].u && e[j].u < e[i].v && e[j].v > e[i].v) return true;
    if(e[j].u < e[i].u && e[i].u < e[j].v && e[j].v < e[i].v) return true;
    return false;
}
void Prepare() {
    int u, v;
    sat.init(M);
    for(int i = 0; i < M; i ++) {
        scanf("%d%d", &u, &v);
        if(u > v) swap(u, v);
        e[i].u = u, e[i].v = v;
        for(int j = 0; j < i; j ++) {
            if(check(i, j)) {
                int a = 2 * i, b = 2 * j;
                sat.AddEdge(a, b ^ 1);
                sat.AddEdge(a ^ 1, b);
                sat.AddEdge(b, a ^ 1);
                sat.AddEdge(b ^ 1, a);
            }
        }
    }
}
int main()
{
    while(scanf("%d%d", &N, &M) != EOF) {
        Prepare();
        bool ok = sat.tsat();
```

```
        if(!ok) puts("the evil panda is lying again");
        else puts("panda is telling the truth...");
    }
    return 0;
}
```

**位枚举+2-sat**

```
TwoSAT sat;
int N;
int mat[maxn][maxn];
bool ok;
void Prepare() {
    for(int i = 0; i < N; i ++) {
        for(int j = 0; j < N; j ++) {
            scanf("%d", &mat[i][j]);
        }
    }
    ok = true;
    for(int i = 0; i < N; i ++) {
        if(mat[i][i] != 0) {ok = false; break;}
        for(int j = 0; j < N; j ++) {
            if(mat[i][j] != mat[j][i]) {
                ok = false; break;
            }
        }
    }
}
```

```
void conect(int x, int y, int c) {
    if(x % 2 == 0 && y % 2 == 0) { //与
        if(c == 1) {
            sat.AddEdge(2*x, 2*y);
            sat.AddEdge(2*y, 2*x);
        }
        else {
            sat.AddEdge(2*x, 2*y + 1);
            sat.AddEdge(2*y, 2*x + 1);
        }
    }
    else if(x % 2 == 1 && y % 2 == 1) { //或
        if(c == 1) {
            sat.AddEdge(2*x + 1, 2*y);
            sat.AddEdge(2*y + 1, 2*x);
        }
        else {
            sat.AddEdge(2*x + 1, 2*y + 1);
            sat.AddEdge(2*y + 1, 2*x + 1);
        }
    }
    else { //异或
        if(c == 1) {
            sat.AddEdge(2*x + 1, 2*y);
            sat.AddEdge(2*y + 1, 2*x);
            sat.AddEdge(2*x, 2*y + 1);
```

```
            sat.AddEdge(2*y, 2*x + 1);
        }
        else {
            sat.AddEdge(2*x + 1, 2*y + 1);
            sat.AddEdge(2*x, 2*y);
            sat.AddEdge(2*y, 2*x);
            sat.AddEdge(2*y + 1, 2*x + 1);
        }
    }
}
int main()
{
    while(scanf("%d", &N) != EOF) {
        Prepare();
        if(!ok) {puts("NO"); continue;}
        for(int k = 0; k < 32; k ++) {
            sat.init(N);
            for(int i = 0; i < N; i ++) {
                for(int j = 0; j < N; j ++) {
                    if(i == j) continue;
                    int c = ((mat[i][j] >> k) & 1);
                    conect(i, j, c);
                }
            }
            ok = sat.tsat(); if(!ok) break;
        }
        if(ok) puts("YES");
        else puts("NO");
    }
    return 0;
}
```

# 连通分量

## 强连通

**Tarjan**

```
struct Edge{
    int to, next;
    Edge(){}
    Edge(int _to, int _next) : to(_to), next(_next){}
};
struct Tarjan {
    int m;
    int head[maxn];
    Edge edges[maxm];
    int pre[maxn], sccno[maxn], lowlink[maxn], scc_cnt, dfs_clock;
    int S[maxn], top;
    int cnt[maxn]; //记录每个强连通分量中点的个数
    void init() {
        m = 0;
        memset(head, -1, sizeof(head));
        memset(cnt, 0, sizeof(cnt));
```

```
}
void AddEdge(int from, int to) {
    int next = head[from];
    head[from] = m;
    edges[m ++] = Edge(to, next);
}
void dfs(int u) {
    pre[u] = lowlink[u] = ++ dfs_clock;
    S[top ++] = u;
    for(int i = head[u]; i != -1; i = edges[i].next) {
        int v = edges[i].to;
        if(!pre[v]) {
            dfs(v);
            lowlink[u] = min(lowlink[u], lowlink[v]);
        }else if(!sccno[v]) {
            lowlink[u] = min(lowlink[u], pre[v]);
        }
    }
    if(pre[u] == lowlink[u]) {
        scc_cnt ++;
        while(true) {
            cnt[scc_cnt] ++;
            int x = S[-- top];
            sccno[x] = scc_cnt;
            if(x == u) break;
        }
    }
    void find_scc(int n) {
        dfs_clock = top = scc_cnt = 0;
        memset(pre, 0, sizeof(pre));
        memset(sccno, 0, sizeof(sccno));
        for(int i = 0; i < n; i ++) {
            if(!pre[i]) dfs(i);
        }
    }
};
```

**最少加几条边让原图强连通**

```
Tarjan tar;
int N, M;
bool out0[maxn], in0[maxn]; //
int Map[505][505];
void Pre() {
    tar.init();
    for(int i = 1; i <= N; i ++) {
        for(int j = 1; j <= M; j ++) {
            scanf("%d", &Map[i][j]);
        }
    }
    for(int i = 1; i <= N; i ++) {
        for(int j = 1; j <= M; j ++) {
            if(j < M) {
```

```
            int u = (i - 1) * M + j - 1;
            int v = u + 1;
            if(Map[i][j + 1] > Map[i][j]) {
                tar.AddEdge(v, u);
            }
            else if(Map[i][j + 1] == Map[i][j]) {
                tar.AddEdge(u, v); tar.AddEdge(v, u);
            }
            else tar.AddEdge(u, v);
            //printf("u = %d, v = %d\n", u, v);
        }
        if(i < N) {
            int u = (i - 1) * M + j - 1;
            int v = u + M;
            if(Map[i][j] > Map[i + 1][j]) {
                tar.AddEdge(u, v);
            }
            else if(Map[i][j] == Map[i + 1][j]) {
                tar.AddEdge(u, v);
                tar.AddEdge(v, u);
            }
            else tar.AddEdge(v, u);
            //printf("u = %d, v = %d\n", u, v);
        }
    }
}
}
int main()
{
    while(scanf("%d%d", &M, &N) != EOF) {
        Pre();
        tar.find_scc(N * M);

        if(tar.scc_cnt == 1) {
            printf("0\n"); continue;
        }
        for(int i = 1; i <= tar.scc_cnt; i ++) {
            in0[i] = out0[i] = true; //true 表示入度或出度为 0
        }
        for(int u = 0; u < N * M; u ++) {
            for(int i = tar.head[u]; i != -1; i = tar.edges[i].next) {
                int v = tar.edges[i].to;
                if(tar.sccno[v] != tar.sccno[u]) {
                    in0[tar.sccno[v]] = out0[tar.sccno[u]] = false;
                }
            }
        }
        int a, b;
        a = b = 0;
        for(int i = 1; i <= tar.scc_cnt; i ++) {
            if(in0[i]) a ++;     //入度为零的强连通分量个数
            if(out0[i]) b ++;    //出度为零的强连通分量个数
```

```
        }
        int ans = max(a, b);
        printf("%d\n", ans);
    }
    return 0;
}
```

**添加一些边使原图仍是简单图且非强连通图（要使添边数目最大）**

将这个图变成两个强连通，而且是一个连通分量只能指向另一个连通分量，可以先缩点

```
Tarjan tar;
int N, M, Tot;
int in0[maxn], out0[maxn];
int mint;
void Prepare() {
    scanf("%d%d", &N, &M);
    tar.init();
    int u, v;
    for(int i = 0; i < M; i ++) {
        scanf("%d%d", &u, &v);
        u --; v --;
        tar.AddEdge(u, v);
    }
    memset(in0, 0, sizeof(in0[0]) * (N + 1));
    memset(out0, 0, sizeof(out0[0]) * (N + 1));
}
int main()
{
    //freopen("input.txt", "r", stdin);
    scanf("%d", &Tot);
    for(int ac = 1; ac <= Tot; ac ++) {
        Prepare();
        tar.find_scc(N);
        printf("Case %d: ", ac);
        if(tar.scc_cnt == 1) {printf("-1\n"); continue;}
        for(int u = 0; u < N; u ++) {
            for(int i = tar.head[u]; i != -1; i = tar.edges[i].next) {
                int v = tar.edges[i].to;
                if(tar.sccno[u] != tar.sccno[v]) {
                    in0[tar.sccno[v]] ++; out0[tar.sccno[u]] ++;
                }
            }
        }
        mint = maxn;
        int cur;
        for(int i = 1; i <= tar.scc_cnt; i ++) {
            if(in0[i] == 0 || out0[i] == 0) {
                if(mint > tar.cnt[i]) {
                    mint = tar.cnt[i]; cur = i;
                }
            }
        }
        LL ans = (N - mint) * (N - mint - 1) + mint * (mint - 1);
```

```
        ans += (mint * (N - mint));
        ans -= M;
        printf("%I64d\n", ans);
    }
    return 0;
}
```

# 双连通

### 双连通找桥（有重边）

```
struct Edge{
    int from, to, id, mark;
    Edge(){}
    Edge(int _from, int _to, int _id, int _mark)
    :from(_from), to(_to), id(_id), mark(_mark){}
};
struct BCC {
    int n, m;
    int pre[maxn], dfs_clock;
    int head[maxn];
    int low[maxn];
    int next[maxm];
    Edge edges[maxm];
    vector<int> Bridges;
    void init(int n) {
        this->n = n;
        memset(head, -1, sizeof(head[0])*(n + 1));
        memset(pre, 0, sizeof(pre[0])*(n + 1));
        memset(low, 0, sizeof(low[0])*(n + 1));
        dfs_clock = 0;
        Bridges.clear();
        m = 0;
    }
    void AddEdge(int from, int to, int id){
        next[m] = head[from];
        edges[m] = Edge(from, to, id, 0);
        head[from] = m ++;
    }
    void dfs(int u, int fa) {
        low[u] = pre[u] = ++ dfs_clock;
        for(int i = head[u]; i != -1; i = next[i]) {
            Edge& e = edges[i];
            int v = e.to;
            if(edges[i].mark) continue;
            edges[i].mark = edges[i ^ 1].mark = 1;
            if(!pre[v]) {
                dfs(v, u);
                low[u] = min(low[u], low[v]);
                if(low[v] > pre[u]) {
                    Bridges.push_back(e.id);
                    //printf("u = %d, v = %d\n", u, v);
                }
            }else {
```

```
                low[u] = min(low[u], pre[v]);
            }
        }
    }
};
BCC bcc;
int N, M;
int weight[maxm];
void Pre() {
    bcc.init(N);
    int u, v;
    for(int i = 0; i < M; i ++) {
        scanf("%d%d%d", &u, &v, &weight[i]);
        //printf("u = %d, v = %d\n", u, v);
        bcc.AddEdge(u, v, i);
        bcc.AddEdge(v, u, i);
    }
}
int main()
{
    //freopen("input.txt", "r", stdin);
    while(scanf("%d%d", &N, &M), N + M) {
        Pre();
        bcc.dfs(1, -1);
        bool flag = false;
        for(int i = 1; i <= N; i ++) {
            if(!bcc.pre[i]) {flag = true; break;}
        }
        if(flag) {
            printf("0\n"); continue;
        }
        int sz = bcc.Bridges.size();
        if(sz == 0) {
            printf("-1\n"); continue;
        }
        //printf("sz = %d\n", sz);
        int ans = maxm;
        for(int i = 0; i < sz; i ++) {
            //printf("%d\n", bcc.edges[bcc.Bridges[i]].dist);
            ans = min(ans, weight[bcc.Bridges[i]]);
        }
        if(ans == 0) ans ++;
        printf("%d\n", ans);
    }
    return 0;
}
```

# 二分图

**最少删除几条边使含奇圈的图变成二分图**

```
int e[maxn][2];
int N, M, Tot;
int main()
```

```
{
    scanf("%d", &Tot);
    while(Tot --) {
        scanf("%d%d", &N, &M);
        for(int i = 0; i < M; i ++) {
            scanf("%d%d", &e[i][0], &e[i][1]);
        }
        int ans = M + 10, cnt;
        for(int i = 0; i < (1 << N); i ++) { //枚举所有染色的情况
            cnt = 0;
            for(int j = 0; j < M; j ++) {
                if(((i >> e[j][0]) & 1) == ((i >> e[j][1]) & 1)) {
                    cnt ++;
                }
            }
            if(ans > cnt) ans = cnt;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

**三正则图匹配**

```
/**
给一 n 个点的三正则图，求最大匹配。
根据握手定理，n 一定是偶数。
由于三正则图，而且题目提示是 2 边连通，所以图中不存在桥，
也就是一定可以找到一条回路经过每个顶点至少一次
（强连通的定义：强连通图一定存在一条回路记过每个顶点至
少一次）由于是三则图，每个顶点的度是 3，如果这条回路经过
某个顶点 2 次，那么这个顶点的度就是 4，这个和条件矛盾。
这条经过每个顶点一次的交错路就可以作出 n/2 匹配。
*/
#include <stdio.h>
int K, N;
int main(){
    scanf("%d", &K);
    int u, v;
    for(int ac = 1; ac <= K; ac ++) {
        scanf("%d", &N);
        for(int i = 0; i < N*3 / 2; i ++) scanf("%d%d", &u, &v);
        printf("%d\n", N / 2);
    }
    return 0;
}
```

## 二分图最大匹配

**最小点覆盖&最大独立集**

最小点覆盖 = 最大匹配数

最大独立集建模：

将有联系的 X 点和 Y 点连边，求最大匹配

然后点的总数 - 最大匹配数 = 独立点个数

最小路径覆盖=最大独立集

**匈牙利**

```
struct BPM {
    int n, m;
    int head[maxn];
    int next[12 * maxn];
    int to[12 * maxn];
    int left[maxn];    // left[i]为右边第 i 个点的匹配点编号，-1 表示不存在
    bool T[maxn];    // T[i]为右边第 i 个点是否已标记
    int right[maxn]; //  求最小覆盖用
    bool S[maxn];      //  求最小覆盖用
    void init(int n) {
        this->n = n;
        m = 0;
        memset(head, -1, sizeof(head[0])*(n + 1));
    }
    void AddEdge(int u, int v) {
        next[m] = head[u]; to[m] = v; head[u] = m ++;
    }
    bool dfs(int u) {
      S[u] = true;
      for(int i = head[u]; i != -1; i = next[i]) {
            int v = to[i];
            if(!T[v]) {
                T[v] = true;
                if(left[v] == -1 || dfs(left[v])) {
                    left[v] = u; right[u] = v;
                    return true;
                }
            }
        }
        return false;
    }
    int MaxMatch() {
        int ans = 0;
        memset(left, -1, sizeof(left[0])*(n + 1));
        memset(right, -1, sizeof(right[0])*(n + 1));
        for(int u = 0; u < n; u ++) {
            memset(T, false, sizeof(T[0])*(n + 1));
            memset(S, false, sizeof(S[0])*(n + 1));
            if(dfs(u)) ans ++;
        }
            return ans;
        }
};
```

**HK 算法**

```
struct HK {
    int left[maxn], right[maxn];
    int dx[maxn], dy[maxn];
    int n, m;
    int head[maxn];
    int to[maxm];
    int next[maxm];
```

```
void init(int n) {
    this->n = n;
    m = 0;
    memset(head, -1, sizeof(head[0]) * (n + 1));
}
void AddEdge(int u, int v) {
    next[m] = head[u];
    to[m] = v;
    head[u] = m ++;
}
bool bfs() {
    bool flag = false;
    memset(dx, 0, sizeof(dx));
    memset(dy, 0, sizeof(dy));
    queue<int> Q;
    for(int i = 1; i <= n; i ++) {
        if(right[i] == -1) Q.push(i);
    }
    while(!Q.empty()) {
        int u = Q.front();
        Q.pop();
        for(int i = head[u]; i != -1; i = next[i]) {
            int v = to[i];
            if(dy[v] == 0) {
                dy[v] = dx[u] + 1;
                if(left[v] == -1) flag = true;
                else {
                    dx[left[v]] = dy[v] + 1;
                    Q.push(left[v]);
                }
            }
        }
    }
    return flag;
}
int dfs(int u) {
    for(int i = head[u]; i != -1; i = next[i]) {
        int v = to[i];
        if(dy[v] == dx[u] + 1) {
            dy[v] = 0;
            if(left[v] == -1 || dfs(left[v])) {
                left[v] = u;
                right[u] = v;
                return true;
            }
        }
    }
    return false;
}
int MaxMatch() {
    int res = 0;
    memset(left, -1, sizeof(left));
```

```
        memset(right, -1, sizeof(right));
        while(bfs()) {
            for(int i = 1; i <= n; i ++) {
                if(right[i] == -1) res += dfs(i);
            }
        }
        return res;
    }
};
```

**二分图多重最大匹配（可加限制条件）**

```
int N, D[maxn], W;
int an[8], Dsum;
struct BPM {
    int n, m;
    bool T[maxd];
    int left[maxn][maxd];
    int vcnt[maxd];
    bool G[maxd][maxn];
    void init(int n, int m) {
        this->n = n; this->m = m;
        memset(G, false, sizeof G);
    }
    bool match(int u/*,int limit*/ ) { //可以加一个限制条件 limit
        for(int v = 0; v < m; v ++) {
            if(!T[v] && G[u][v]) {
                T[v] = true;
                int limit = D[v];
                if(vcnt[v] < limit) {
                    left[v][vcnt[v] ++] = u;
                    return true;
                }
                for(int i = 0; i < vcnt[v]; i ++) {
                    if(match(left[v][i])) {
                        left[v][i] = u;
                        return true;
                    }
                }
            }
        }
        return false;
    }
    int MaxMatch() { //求多重最大匹配
        int ans = 0;
        memset(vcnt, 0, sizeof(vcnt));
        memset(left, -1, sizeof(left));
        for(int u = 0; u < n; u ++) {
            memset(T, false, sizeof(T));
            if(match(u)) ans ++;
        }
        return ans;
    }
    bool Perfect(){ //求多重完美匹配
```

```
        memset(vcnt, 0, sizeof(vcnt));
        memset(left, -1, sizeof(left));
        for(int u = 0; u < n; u ++) {
            memset(T, false, sizeof(T));
            if(!match(u)) return false;
        }
        return true;
    }
};
BPM Mt;
char str[105];
void Pre() {
    scanf("%d", &N);
    Mt.init(7 * 50 + 1, N);
    Dsum = 0;
    for(int i = 0; i < N; i ++) {
        for(int j = 0; j < 7; j ++){
            scanf("%d", &an[j]);
        }
        scanf("%d%d", &D[i], &W);
        Dsum += D[i];
        for(int j = 0; j < W; j ++) {
            for(int k = 0; k < 7; k ++){
                if(an[k] == 1) {
                    Mt.G[j * 7 + k][i] = true;
                }
            }
        }
    }
}
int main()
{
    int Tot;
    scanf("%d", &Tot);
    while(Tot --){
        Pre();
        int ans = Mt.MaxMatch();
        if(ans == Dsum) printf("Yes\n");
        else printf("No\n");
    }
    return 0;
}
```

**奇偶匹配&位运算**

```
const int maxn = 60 * 60 + 10;
const int dy[] = {-2, -1, 1, 2, 2, 1, -1, -2, 0, 1, 0, -1};
const int dx[] = {-1, -2, -2, -1, 1, 2, 2, 1, -1, 0, 1, 0};
BPM mt;
int N, M;
int mat[60][60];
bool check(int x, int y) {
    if(x < 0 || y < 0 || y >= M || x >= N) return false;
    return true;
```

```
}
void conect(int x, int y, int val) {
    int u = x * M + y;
    for(int k = 0; k < 12; k ++) {
        if(!((val >> k) & 1)) continue;
        int nx = x + dx[k], ny = y + dy[k];
        if(check(nx, ny) && mat[nx][ny] != -1) {
            int v = nx * M + ny;
            if((x + y) & 1) //x+y 为奇数的向偶数连边
                mt.AddEdge(u, v);
            if((nx + ny) & 1)
                mt.AddEdge(v, u);
        }
    }
}
void Prepare() {
    mt.init(N*M);
    for(int i = 0; i < N; i ++) {
        for(int j = 0; j < M; j ++) {
            scanf("%d", &mat[i][j]);
        }
    }
    for(int i = 0; i < N; i ++) {
        for(int j = 0; j < M; j ++) {
            if(mat[i][j] != -1) {
                conect(i, j, mat[i][j]);
            }
        }
    }
}
int main(){
    int ac = 1;
    while(scanf("%d%d", &N, &M), N + M) {
        Prepare();
        int ans = mt.MaxMatch();
        printf("%d. %d\n", ac ++, ans);
    }
    return 0;
}
```

**输入 N*N 矩形的数据，通过行交换或者列交换来使正对角线的数值都为 1.**

- 题解：按照正常逻辑也可以进行判断。
- 转换成二分图匹配问题更简单一些 （行号和列号相等）
- 左边为矩形的行数，右边为矩形中‘1’所在的列。

```
struct BPM {
    int n, m;
    int left[maxn];
    bool T[maxn];
    int G[maxn][maxn];
    void init(int n, int m) {
        this->n = n;
        this->m = m;
```

```
    }
    bool dfs(int u) {
        for(int v = 1; v <= m; v ++) {
            if(!T[v] && G[u][v]) {
                T[v] = true;
                if(left[v] == -1 || dfs(left[v])) {
                    left[v] = u;
                    return true;
                }
            }
        }
        return false;
    }
    bool Perfect() {
        memset(left, -1, sizeof(left));
        for(int u = 1; u <= n; u ++) {
            memset(T, false, sizeof(T));
            if(!dfs(u)) return false;
        }
        return true;
    }
};
BPM mt;
int N;
int x[maxn], y[maxn];
int cnt;
void Prepare() {
    mt.init(N, N);
    for(int i = 1; i <= N; i ++) {
        for(int j = 1; j <= N; j ++) {
            scanf("%d", &mt.G[i][j]);
        }
    }
}
int main()
{
    while(scanf("%d", &N) != EOF) {
        Prepare();
        bool ok = mt.Perfect();
        if(!ok) {printf("-1\n"); continue;}
        cnt = 0;
        for(int i = 1; i <= N; i ++) {
            int cur = i, j;
            for(j = i; j <= N; j ++) {
                if(mt.left[cur] >= mt.left[j]) cur = j;
            }
            if(cur != i) {
                x[cnt] = cur, y[cnt ++] = i;
                swap(mt.left[cur], mt.left[i]);
            }
        }
        printf("%d\n", cnt);
```

```
    for(int i = 0; i < cnt; i ++) {
      printf("C %d %d\n", x[i], y[i]);
    }
  }
  return 0;
}
```

**一行变多行，一列变多列**

X[i][j] 和 Y[i][j]分别表示第 i 行 j 列的新的 X,Y 坐标值

```
memset(X, -1, sizeof X);
  int x = -1;
  for(int i = 1; i <= R; i ++) {
    for(int j = 1; j <= C; j ++) {
      if(g[i][j] == '.');
      else {
        if(g[i][j - 1] != '*') X[i][j] = ++ x;
        else X[i][j] = x;
      }
    }
  }
  memset(Y, -1, sizeof Y);
  int y = -1;
  for(int j = 1; j <= C; j ++) {
    for(int i = 1; i <= R; i ++) {
      if(g[i][j] == '.');
      else {
        if(g[i - 1][j] != '*') Y[i][j] = ++ y;
        else Y[i][j] = y;
      }
    }
  }
```

**求最小覆盖。X 和 Y 为最小覆盖中的点集**

```
int mincover(vector<int>& X, vector<int>& Y) {
    int ans = MaxMatch();
    memset(S, false, sizeof(S));
    memset(T, false, sizeof(T));
    for(int u = 0; u < n; u ++)
      if(right[u] == -1) dfs(u); //  从所有 X 未盖点出发增广
    for(int u = 0; u < n; u ++)
      if(!S[u]) X.push_back(u); // X 中的未标记点
    for(int v = 0; v < m; v ++)
      if(T[v]) Y.push_back(v);    // Y 中的已标记点
  return ans;
}
```

## 二分图最佳匹配（最大权）

**KM 模版**

```
int W[maxn][maxn];
int lx[maxn], ly[maxn];
int left[maxn];
int slack[maxn]; //
bool S[maxn], T[maxn];
int ny, nx; //x 结点和 y 结点的个数
```

```
int N, M, Tot;
int Z;
bool dfs(int u) {
    S[u] = true;
    for(int v = 1; v <= ny; v ++) {
        if(T[v]) continue;
        int tmp = lx[u] + ly[v] - W[u][v];
        if(tmp == 0) {
            T[v] = true;
            if(left[v] == -1 || dfs(left[v])) {
                left[v] = u;
                return true;
            }
        } else if(slack[v] > tmp) {
            slack[v] = tmp;
        }
    }
    return false;
}
void update() {
    int a = inf;
    for(int j = 1; j <= ny; j ++) {
        if(!T[j]) {
            a = min(a, slack[j]);
        }
    }
    for(int i = 1; i <= nx; i ++) {
        if(S[i]) lx[i] -= a;
    }
    for(int i = 1; i <= ny; i ++){
        if(T[i]) ly[i] += a;
        else{
            slack[i] -= a;
        }
    }
}
void KM() {
    memset(left, -1, sizeof left);
    memset(ly, 0, sizeof ly);
    for(int i = 1; i <= nx; i ++) {
        lx[i] = -inf; //=0 || -inf  视具体情况
        for(int j = 1; j <= ny; j ++) {
            lx[i] = max(lx[i], W[i][j]);
        }
    }
    for(int i = 1; i <= nx; i ++) {
        for(int j = 1; j <= ny; j ++)
            slack[j] = inf;
        while(true) {
            memset(T, false, sizeof(T));
            memset(S, false, sizeof(S));
            if(dfs(i)) break;
```

```
            else update();
        }
    }
}
void Prepare(){
    scanf("%d%d", &N, &M);
    ny = N; nx = N;
    for(int i = 1; i <= N; i ++) {
        for(int j = 1; j <= N; j ++) {
            W[i][j] = -inf;
        }
    }
    int u, v, w;
    while(M --) {
        scanf("%d%d%d", &u, &v, &w);
        W[u][v] = max(-w, W[u][v]);
    }
}
int main()
{
    scanf("%d", &Tot);
    while(Tot --) {
        Prepare();
        KM();
        int ans = 0;
        for(int i = 1; i <= ny; i ++){
            if(left[i] == -1) continue;
            ans += W[left[i]][i];
        }
        printf("%d\n", -ans);
    }
    return 0;
}
```

**回溯求所有解**

```
void Pre(){
    scanf("%d", &N);
    ny = N;
    nx = N;
    memset(W, 0, sizeof W);
    for(int i = 1; i <= N; i ++) {
        for(int j = 0; j < N; j ++) {
            scanf("%d", &Z);
            W[Z][i] -= j;
        }
    }
    for(int i = 1; i <= N; i ++) {
        for(int j = 0; j < N; j ++) {
            scanf("%d", &Z);
            W[i][Z] -= j;
        }
    }
}
```

```
void find(int dep, int sum) {
    if(sum > ans) return;
    if(dep > nx) {
        if(sum != ans) return;
        printf("Best Pairing %d\n", cnt ++);
        for(int i = 1; i <= ny; i ++) {
            printf("Supervisor %d with Employee %d\n", i, left[i]);
        }
    }
    else {
        for(int i = 1; i <= ny; i ++) {
            if(!vis[i]) {
                vis[i] = true;
                left[dep] = i;
                find(dep + 1, sum - W[dep][i]);
                vis[i] = false;
            }
        }
    }
}
int main()
{
    scanf("%d", &Tot);
    for(int ac = 1; ac <= Tot; ac ++) {
        Pre();
        KM();
        ans = 0;
        for(int i = 1; i <= ny; i ++){
            if(left[i] == -1) continue;
            ans -= W[left[i]][i];
        }
        printf("Data Set %d, Best average difference: ", ac);
        printf("%.6f\n", (double)ans / 2 / nx);
        cnt = 1;
        memset(vis, false, sizeof(vis));
        find(1, 0);
        printf("\n");
    }
    return 0;
}
```

**偏好度**

```
struct Brute {
    int H, A;
};
Brute St[maxn], Xt[maxn];
int V[maxn];
int check(Brute St, Brute Xt) {
    return (St.H - 1) / Xt.A >= (Xt.H - 1) / St.A ? 1 : -1;
}
void Prepare() {
    nx = ny = N;
    for(int i = 1; i <= N; i ++) scanf("%d", &V[i]);
```

```
    for(int i = 1; i <= N; i ++) scanf("%d", &St[i].H);
    for(int i = 1; i <= N; i ++) scanf("%d", &Xt[i].H);
    for(int i = 1; i <= N; i ++) scanf("%d", &St[i].A);
    for(int i = 1; i <= N; i ++) scanf("%d", &Xt[i].A);
    for(int i = 1; i <= N; i ++) {
        for(int j = 1; j <= N; j ++) {
            W[i][j] = 10 * check(St[i], Xt[j]) * V[i];
            if(i == j) W[i][j] ++;
        }
    }
}
int main()
{
    while(scanf("%d", &N), N) {
        Prepare();
        KM();
        int cnt = 0, ans = 0;
        for(int i = 1; i <= N; i ++) {
            if(W[left[i]][i] % 10 != 0) {cnt ++; W[left[i]][i] --;}
            ans += W[left[i]][i];
        }
        ans /= 10;
        if(ans < 0) printf("Oh, I lose my dear seaco!\n");
        else printf("%d %.3f%%\n", ans, cnt * 100.0 / N);
    }
    return 0;
}
```