**ACM**

**图论&数据结构**

Haoxin Gong

目录

[图论 9](#_Toc58268252)

[最短路 9](#_Toc58268253)

[模板 9](#_Toc58268254)

[1.Dijkstra 9](#_Toc58268255)

[2.K短路 10](#_Toc58268256)

[3.分层图最短路 11](#_Toc58268257)

[4.Spfa正负环 13](#_Toc58268258)

[5.Spfa最短路 13](#_Toc58268259)

[6.差分约束 14](#_Toc58268260)

[7.6.Floyd输出最小环 14](#_Toc58268261)

[8.可重复次短路 16](#_Toc58268262)

[9.Bitset优化Floyd 17](#_Toc58268263)

[题集 17](#_Toc58268264)

[1.无向图K个关键点最远距离 17](#_Toc58268265)

[2.最短路径树边权和最小 19](#_Toc58268266)

[3.删关键边保持最短路不变 20](#_Toc58268267)

[4.abc不可走->ab&bc不可走 22](#_Toc58268268)

[5.距离s恰好为d的点数可以在边上 24](#_Toc58268269)

[6.花费x最多跑y 26](#_Toc58268270)

[7.lowbit建32个虚点 27](#_Toc58268271)

[8.更新部分边权使最短路为L 29](#_Toc58268272)

[9.地铁线路按边算最短路 31](#_Toc58268273)

[10.多源最短路加油站询问两点是否可达 32](#_Toc58268274)

[11.K种颜色询问最短路 36](#_Toc58268275)

[12.最多100个环查询最短路 38](#_Toc58268276)

[13.多米诺骨牌最短路 40](#_Toc58268277)

[14.K条绳子烧完时间（记到边最短路） 43](#_Toc58268278)

[15.有节点保护最短路 44](#_Toc58268279)

[16.平面切割无法从S到T 46](#_Toc58268280)

[17.平面&直线&圆离散最短路 48](#_Toc58268281)

[18.离散化附加边枚举中间点 50](#_Toc58268282)

[19.S->M->T枚举中间点最长 51](#_Toc58268283)

[20.必经过K个点的最短路对K离散化 53](#_Toc58268284)

[21.单调性限制前K个点不能切换模式 54](#_Toc58268285)

[22.AGCT找2，3，4环 56](#_Toc58268286)

[23.a->b->c分配边权枚举汇点 57](#_Toc58268287)

[24.一条边反向是否改变dis（正反跑） 59](#_Toc58268288)

[25.差分约束裸题 61](#_Toc58268289)

[26.道路与航线 62](#_Toc58268290)

[27.逆行一次S能经过最多点 64](#_Toc58268291)

[28.小雨坐地铁分层图最短路 67](#_Toc58268292)

[图的连通性 69](#_Toc58268293)

[模板 69](#_Toc58268294)

[1.无向图边双 69](#_Toc58268295)

[2.无向图点双 69](#_Toc58268296)

[3.无向图割点 70](#_Toc58268297)

[4.无向图关键点 71](#_Toc58268298)

[5.无向图可重割边 73](#_Toc58268299)

[6.无向图输出最小割 74](#_Toc58268300)

[7.无向图全局最小割 76](#_Toc58268301)

[8.无向图边双带重边 78](#_Toc58268302)

[9.有向图缩点 79](#_Toc58268303)

[10.仙人掌图判断 79](#_Toc58268304)

[11.有向图判环 81](#_Toc58268305)

[12.DAG图可达数 82](#_Toc58268306)

[13.DAG图必经点 82](#_Toc58268307)

[14.DAG图最小表示法 83](#_Toc58268308)

[15.DAG图S到T路径数 84](#_Toc58268309)

[16.动态图的连通性 84](#_Toc58268310)

[题集: 87](#_Toc58268311)

[1.矿场搭建 87](#_Toc58268312)

[2.无向图变强连通 89](#_Toc58268313)

[3.构造方案使图强连通 91](#_Toc58268314)

[4.删一点求每点不可达数 92](#_Toc58268315)

[5.固定边方向使图无环方案数 93](#_Toc58268316)

[6.删除多余边仍然强连通 95](#_Toc58268317)

[7.定义边方向使得S到达点最多&最少 96](#_Toc58268318)

[8.定义边方向变DAG图 98](#_Toc58268319)

[二分图&匹配 99](#_Toc58268320)

[模板 99](#_Toc58268321)

[1.二分图的性质 99](#_Toc58268322)

[2.匈牙利算法 100](#_Toc58268323)

[3.二分图最大带权匹配 100](#_Toc58268324)

[4.无向图匹配带花树算法 101](#_Toc58268325)

[5.二分图可行边和必须边 103](#_Toc58268326)

[题集： 104](#_Toc58268327)

[1.陆地覆盖后最大独立集 104](#_Toc58268328)

[2.三分图染色 106](#_Toc58268329)

[3.DAG图最小路径覆盖 107](#_Toc58268330)

[4.注意可以横纵坐标当成二分图 107](#_Toc58268331)

[网络流 107](#_Toc58268332)

[1.Dinic弧优化 107](#_Toc58268333)

[2.Dij费用流 109](#_Toc58268334)

[3.Spfa费用流 113](#_Toc58268335)

[4.最小割可行边&必须边 120](#_Toc58268336)

[5.网格图对偶图建图 123](#_Toc58268337)

[6.输出最小路径覆盖 124](#_Toc58268338)

[7.最大权闭合子图输出方案 127](#_Toc58268339)

[欧拉路 130](#_Toc58268340)

[1.欧拉图判断 130](#_Toc58268341)

[2.欧拉路径输出 130](#_Toc58268342)

[3.多条欧拉路覆盖无向图方案 132](#_Toc58268343)

[拓扑排序 134](#_Toc58268344)

[模板: 134](#_Toc58268345)

[1.拓扑排序 134](#_Toc58268346)

[题集： 135](#_Toc58268347)

[1.相对位置建边字典序最小 135](#_Toc58268348)

[2.两条路径最大路径交 136](#_Toc58268349)

[生成树 139](#_Toc58268350)

[模板： 139](#_Toc58268351)

[1.Kruscal 139](#_Toc58268352)

[2.Prime生成树 140](#_Toc58268353)

[3.异或MST 141](#_Toc58268354)

[4.斯坦纳树 142](#_Toc58268355)

[5.有根最小树形图 143](#_Toc58268356)

[6.无根最小树形图 145](#_Toc58268357)

[7.MST计数 147](#_Toc58268358)

[题集： 148](#_Toc58268359)

[1.MST必须边 148](#_Toc58268360)

[2.完全图最多MST构造 153](#_Toc58268361)

[3.生成树使得点1度数为d 153](#_Toc58268362)

[2-sat 156](#_Toc58268363)

[1.2-sat 156](#_Toc58268364)

[2.2-sat字典序输出方案 158](#_Toc58268365)

[婚姻稳定 159](#_Toc58268366)

[数据结构 161](#_Toc58268367)

[并查集 161](#_Toc58268368)

[模板 161](#_Toc58268369)

[1.路径压缩 161](#_Toc58268370)

[2.启发式合并 162](#_Toc58268371)

[3.可撤销并查集 162](#_Toc58268372)

[题集 163](#_Toc58268373)

[1.启发式合并差分 163](#_Toc58268374)

[2.补图连通块 164](#_Toc58268375)

[3.可撤销并查集删除边求连通块最大点权 166](#_Toc58268376)

[最近共同祖先 168](#_Toc58268377)

[1.LCA倍增 168](#_Toc58268378)

[2.LCA离线 169](#_Toc58268379)

[3.LCA按顺序打印路径 170](#_Toc58268380)

[4.树上K条路径交 170](#_Toc58268381)

[5.树上ST表 171](#_Toc58268382)

[6.树上差分 172](#_Toc58268383)

[灭绝树 172](#_Toc58268384)

[1.每个点支配数 172](#_Toc58268385)

[2.删多少点使得x，y其中一个无法到达终点 174](#_Toc58268386)

[ST表 176](#_Toc58268387)

[1.一维ST表 176](#_Toc58268388)

[2.二维ST表 177](#_Toc58268389)

[树的直径 179](#_Toc58268390)

[模板 179](#_Toc58268391)

[1.树形dp 179](#_Toc58268392)

[2.两次DFS 179](#_Toc58268393)

[题集 180](#_Toc58268394)

[树的重心 182](#_Toc58268395)

[1.树形dp求树重心 182](#_Toc58268396)

[2.一些性质 182](#_Toc58268397)

[点分治 183](#_Toc58268398)

[1.询问有无长度为k的路径 183](#_Toc58268399)

[2.模3为0的点对数 185](#_Toc58268400)

[3.长度为K的路径边数最小 187](#_Toc58268401)

[4.距离<=k的路径数 189](#_Toc58268402)

[5.模3为1，2，3的路径和 192](#_Toc58268403)

[树链剖分 194](#_Toc58268404)

[模板 194](#_Toc58268405)

[题集: 195](#_Toc58268406)

[1.子树和路径修改 195](#_Toc58268407)

[2.区间覆盖区间最大值 198](#_Toc58268408)

[平衡树 203](#_Toc58268409)

[1.FHQ平衡树 203](#_Toc58268410)

[虚树 205](#_Toc58268411)

[模板 205](#_Toc58268412)

[单调栈 207](#_Toc58268413)

[模板 207](#_Toc58268414)

[1.右边第一个比他大的数下标 207](#_Toc58268415)

[题集 208](#_Toc58268416)

[1.最大全1子矩阵 208](#_Toc58268417)

[2.字典序最小的排列 209](#_Toc58268418)

[3.黑格子不可走子矩阵个数 209](#_Toc58268419)

[单调队列 210](#_Toc58268420)

[1.区间最大&最小值 210](#_Toc58268421)

[2.最大最小差值为k的区间数 211](#_Toc58268422)

[主席树 212](#_Toc58268423)

[模板 212](#_Toc58268424)

[1.区间第K大 212](#_Toc58268425)

[题集 214](#_Toc58268426)

[1.区间不同数个数 214](#_Toc58268427)

[2.区间与和的个数 215](#_Toc58268428)

[线段树&树状数组 217](#_Toc58268429)

[模板 217](#_Toc58268430)

[1.树状数组单点修改区间查询 217](#_Toc58268431)

[2.树状数组区间修改区间查询 217](#_Toc58268432)

[3.线段树区间加修改区间查询 218](#_Toc58268433)

[4.线段树区间加&乘 219](#_Toc58268434)

[5.区间01反转查询最长0，1子段和 221](#_Toc58268435)

[6.区间异或线段树 223](#_Toc58268436)

[7.三次幂加乘覆盖 225](#_Toc58268437)

[8.维护树直径 229](#_Toc58268438)

[9.单点修改最大子段和 232](#_Toc58268439)

[10.区间1e9的动态开点 234](#_Toc58268440)

[莫队 236](#_Toc58268441)

[模板 236](#_Toc58268442)

[1.区间不同数 236](#_Toc58268443)

[题集 237](#_Toc58268444)

[1.查询区间数次数的平方 237](#_Toc58268445)

[树上启发式合并 239](#_Toc58268446)

[模板 239](#_Toc58268447)

[1.颜色种类 239](#_Toc58268448)

[题集： 240](#_Toc58268449)

[1.差分答案，树状数组后缀和 240](#_Toc58268450)

[字符串 243](#_Toc58268451)

[哈希 243](#_Toc58268452)

[1.SRC哈希 243](#_Toc58268453)

[2.康托展开 245](#_Toc58268454)

[3.矩阵哈希 245](#_Toc58268455)

[4.大数减法字符串哈希 246](#_Toc58268456)

[5.手写双double哈希 246](#_Toc58268457)

[01字典树 247](#_Toc58268458)

[模板 247](#_Toc58268459)

[AC自动机 248](#_Toc58268460)

[模板 248](#_Toc58268461)

[KMP 249](#_Toc58268462)

[模板 249](#_Toc58268463)

[马拉车 250](#_Toc58268464)

[Manacher最长回文串 250](#_Toc58268465)

[最小表示法 251](#_Toc58268466)

[模板 251](#_Toc58268467)

[数符转换 252](#_Toc58268468)

[模板 252](#_Toc58268469)

[大整数string加法 253](#_Toc58268470)

[模板 253](#_Toc58268471)

[大数减法 254](#_Toc58268472)

[模板 254](#_Toc58268473)

[大数取模 254](#_Toc58268474)

[模板 254](#_Toc58268475)

[杂项 255](#_Toc58268476)

[1.逆波兰数 255](#_Toc58268477)

[2.曼哈顿矩阵表示 258](#_Toc58268478)

[3.等比矩阵求前N项和 258](#_Toc58268479)

[4.已知一条边构造RT三角形 260](#_Toc58268480)

[5.矩阵已知三点求第四点 261](#_Toc58268481)

[6.简易对拍器 261](#_Toc58268482)

[r.cpp 261](#_Toc58268483)

[cmp.bat 264](#_Toc58268484)

[7.圆上2\*N个点不相交概率 264](#_Toc58268485)

[8.贪心二分 264](#_Toc58268486)

[9.除法向上取整 265](#_Toc58268487)

[10.三个矩形能否构成正方形 265](#_Toc58268488)

[11.二维离散化 266](#_Toc58268489)

[12.三元环计数 268](#_Toc58268490)

[13.01BFS 269](#_Toc58268491)

[14.树上最小支配集 270](#_Toc58268492)

[15.无向图入侵颜色 272](#_Toc58268493)

[16.O3优化 273](#_Toc58268494)

[STL 273](#_Toc58268495)

[1.Vector 273](#_Toc58268496)

[2.List 274](#_Toc58268497)

[3.Deque 275](#_Toc58268498)

[4.Map 275](#_Toc58268499)

[5.Multimap 276](#_Toc58268500)

[6.Unordered\_map 277](#_Toc58268501)

[7.Set 277](#_Toc58268502)

[8.Multiset 277](#_Toc58268503)

[9.Unordered\_set 277](#_Toc58268504)

[10.String 277](#_Toc58268505)

[11.Bitset 278](#_Toc58268506)

# 图论

## 最短路

### 模板

#### 1.Dijkstra

//M \* log(M)

vector <pir> G[N];

int dis[N], n, m;

bool vis[N];

void Dijkstra(int s) { // 根据需要改pir的long long

priority\_queue <pir, vector <pir>, greater <pir>> q;

MEM(dis, INF); MEM(vis, 0);

dis[s] = 0;

q.push({ 0, s });

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (vis[u])

continue;

vis[u] = true;

for (auto it : G[u]) {

int v = it.second, w = it.first;

if (dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

q.push({ dis[v], v });

}

}

}

}

#### 2.K短路

struct Edge {

int v, nxt;

ll w;

}G[N \* 100], Gr[N \* 100];

struct Node {

int v;

ll g, h;

bool operator < (const Node &oth) const {

if (g + h != oth.g + oth.h)

return g + h > oth.g + oth.h;

return g > oth.g;

}

};

int h[N], rh[N], idx;

int n, m, sp, tp, k, mxt; // 起点，终点，k

ll dis[N];

bool vis[N];

void init() {

for (int i = 1; i <= n; i++)

h[i] = rh[i] = 0;

idx = 0;

}

void Add(int u, int v, ll w) {

G[++idx] = { v, h[u], w };

h[u] = idx;

Gr[idx] = { u, rh[v], w };

rh[v] = idx;

}

void Dijkstra() {

for (int i = 1; i <= n; i++)

dis[i] = LINF, vis[i] = false;

dis[tp] = 0;

priority\_queue <Node> q;

q.push({ tp, 0, 0 });

while (!q.empty()) {

Node now = q.top();

int u = now.v;

ll d = now.h;

q.pop();

vis[u] = true;

for (int i = rh[u]; i; i = Gr[i].nxt) {

int v = Gr[i].v;

ll w = Gr[i].w;

if (!vis[v] && dis[v] > d + w) {

dis[v] = d + w;

q.push({ v, 0, dis[v] });

}

}

}

}

ll Astar() {

if (sp == tp)

k++;

Dijkstra();

if (dis[sp] == LINF)

return -1;

priority\_queue <Node> q;

q.push({ sp, 0, 0 });

while (!q.empty()) {

Node now = q.top();

int u = now.v;

ll d = now.g;

q.pop();

if (u == tp && --k == 0)

return d;

for (int i = h[u]; i; i = G[i].nxt) {

int v = G[i].v;

ll w = G[i].w;

q.push({ v, d + w, dis[v] });

}

}

return -1;

}

#### 3.分层图最短路

// M \* log(M)

vector <pir> G[N];

int n, m, k;

ll dis[N];

bool vis[N];

void add(int u, int v, ll w) {

G[u].push\_back({ w, v });

}

void dijkstra(int s) { // 根据需要改pir的long long

priority\_queue <pir, vector <pir>, greater <pir>> q;

MEM(dis, LINF); MEM(vis, 0);

dis[s] = 0;

q.push({ 0, s });

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (vis[u])

continue;

vis[u] = true;

for (auto it : G[u]) {

int v = it.second; ll w = it.first;

if (dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

q.push({ dis[v], v });

}

}

}

}

int main()

{

cin >> n >> m >> k;

for (int i = 0; i < m; i++) {

int ui, vi; ll wi;

sc("%d %d %lld", &ui, &vi, &wi);

add(ui, vi, wi); add(vi, ui, wi);

for (int j = 0; j < k; j++) {

add(ui + (j + 1) \* n, vi + (j + 1) \* n, wi);

add(vi + (j + 1) \* n, ui + (j + 1) \* n, wi);

add(ui + j \* n, vi + (j + 1) \* n, 0ll);

add(vi + j \* n, ui + (j + 1) \* n, 0ll);

}

}

dijkstra(1);

printf("%lld\n", dis[n + k \* n]);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 4.Spfa正负环

vector <pir> G[N];

int dis[N], n, m, k;

int num[N];

bool vis[N];

bool spfa(int x) {

queue <int> q;

vis[x] = true;

q.push(x); dis[x] = 0;

num[x]++;

while (!q.empty()) {

int ans = q.front();

q.pop();

vis[ans] = false;

for (int i = 0; i < SZ(G[ans]); i++) {

int vi = G[ans][i].second, wi = G[ans][i].first;

if (dis[vi] > dis[ans] + wi) {

dis[vi] = dis[ans] + wi;

if (vis[vi])

continue;

vis[vi] = true;

num[vi]++;

if (num[vi] > n)

return true;

q.push(vi);

}

}

}

return false;

}

#### 5.Spfa最短路

vector <pir> G[N];

int dis[N], n, m;

bool vis[N];

void spfa(int x) {

queue <int> q;

MEM(dis, INF); vis[x] = true;

q.push(x); dis[x] = 0;

while (!q.empty()) {

int ans = q.front();

q.pop();

vis[ans] = false;

for (auto it : G[ans]) {

int vi = it.second, wi = it.first;

if (dis[vi] > dis[ans] + wi) {

dis[vi] = dis[ans] + wi;

if (vis[vi])

continue;

vis[vi] = true;

q.push(vi);

}

}

}

}

#### 6.差分约束

把式子变成 dis[v] <= dis[u] + w的形式

u往v连w的边权，用Spfa跑，无负环则有解

最短路是最大解，最长路是最小解

#### 7.6.Floyd输出最小环

// N \* N \* N

int a[N][N], n, m, ans = INF;

int d[N][N], mid[N][N];

vector <int> v;

void init() {

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= n; j++) {

if (i == j)

a[i][j] = 0;

else

a[i][j] = INF;

}

}

}

void dfs(int x, int y) {

if (!mid[x][y])

return;

dfs(x, mid[x][y]);

v.push\_back(mid[x][y]);

dfs(mid[x][y], y);

}

void floyd() {

for (int k = 1; k <= n; k++) {

for (int i = 1; i < k; i++) {

for (int j = i + 1; j < k; j++) {

if ((ll)d[i][j] + a[j][k] + a[k][i] < ans) {

ans = d[i][j] + a[j][k] + a[k][i];

v.clear();

v.push\_back(i);

dfs(i, j);

v.push\_back(j);

v.push\_back(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];

mid[i][j] = k;

}

}

}

}

}

int main()

{

cin >> n >> m;

init();

for (int i = 0; i < m; i++) {

int ui, vi, wi;

sc("%d %d %d", &ui, &vi, &wi);

Min(a[ui][vi], wi), Min(a[vi][ui], wi);

}

MPY(d, a);

floyd();

if (ans == INF)

printf("No solution.\n");

else {

for (auto it : v)

printf("%d ", it);

printf("\n");

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 8.可重复次短路

vector <pir> G[N];

int d1[N], d2[N], n, m;

void Dijkstra(int u) {

priority\_queue <pir, vector <pir>, greater <pir> > q;

MEM(d1, INF), MEM(d2, INF);

d1[u] = 0, q.push({ 0, u });

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (d2[u] < now.first)

continue; //次短路要大于等于最短路

for (int i = 0; i < SZ(G[u]); i++) {

int v = G[u][i].second;

int w = now.first + G[u][i].first;

if (d1[v] > w) {

swap(d1[v], w);

q.push({ d1[v], v });

}

if (d1[v] <= w && d2[v] >= w) { //次短路在最短路与其他路径之间

d2[v] = w;

q.push({ d2[v], v });

}

}

}

}

#### 9.Bitset优化Floyd

// N \* N / 32

bitset <N> bt[N];

int n;

void Floyd() {

for (int k = 1; k <= n; k++) {

for (int i = 1; i <= n; i++) {

if (bt[i][k])

bt[i] |= bt[k]; //每个点都异或一下

}

}

}

### 题集

#### 1.无向图K个关键点最远距离

vector <pir> G[N];

ll dis[N], d[N];

int n, m, p;

int a[N], prv[N];

bool vis[N];

void Dijkstra() {

priority\_queue <pir, vector <pir>, greater <pir>> q;

MEM(dis, LINF), MEM(d, LINF);

for (int i = 1; i <= p; i++) {

dis[a[i]] = 0;

prv[a[i]] = a[i];

q.push({ 0, a[i] });

}

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (vis[u])

continue;

vis[u] = true;

for (auto it : G[u]) {

int v = it.second; ll w = it.first;

if (dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

prv[v] = prv[u];

q.push({ dis[v], v });

}

else if (prv[u] != prv[v]) {

ll cost = dis[u] + dis[v] + w;

Min(d[prv[u]], cost);

Min(d[prv[v]], cost);

}

}

}

}

int main()

{

cin >> n >> m >> p;

for (int i = 1; i <= p; i++)

sc("%d", &a[i]);

for (int i = 0; i < m; i++) {

int u, v; ll w;

sc("%d %d %lld", &u, &v, &w);

G[u].push\_back({ w, v });

G[v].push\_back({ w, u });

}

Dijkstra();

for (int i = 1; i <= p; i++)

printf("%lld ", d[a[i]]);

puts("");

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 2.最短路径树边权和最小

vector <pair <int, int>> G[N];

ll dis[N], val[N], d[N];

int pre[N], n, m, x;

bool vis[N];

ll Dijkstra(int s) {

priority\_queue <pir, vector <pir>, greater <pir>> q;

MEM(dis, LINF), MEM(d, LINF);

q.push({ 0, s});

dis[s] = 0, d[s] = 0;

ll ans = 0;

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (vis[u])

continue;

vis[u] = true;

ans += d[u];

for (auto it : G[u]) {

int id = it.first, v = it.second;

ll w = val[id];

if (dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

q.push({ dis[v], v });

pre[v] = id;

d[v] = w;

}

else if (dis[v] == dis[u] + w && w < d[v])

d[v] = w, pre[v] = id;

}

}

return ans;

}

int main()

{

cin >> n >> m;

for (int i = 1; i <= m; i++) {

int u, v;;

sc("%d %d %lld", &u, &v, &val[i]);

G[u].push\_back({ i, v });

G[v].push\_back({ i, u });

}

cin >> x;

printf("%lld\n", Dijkstra(x));

vector <int> ver;

for (int i = 1; i <= n; i++)

if (pre[i])

ver.push\_back(pre[i]);

sort(ALL(ver));

for (auto it : ver)

printf("%d ", it);

puts("");

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 3.删关键边保持最短路不变

// 先去重，再求最短路条数，如果关键边长度>mindis或者多条最短路则可以删

vector <pir> G[N];

int n, m, k;

ll dis[N], dp[N];

bool vis[N], ok[N];

void Dijkstra(int s) { // 根据需要改pir的long long

priority\_queue <pir, vector <pir>, greater <pir>> q;

MEM(dis, LINF);

dis[s] = 0, dp[s] = 1;

q.push({ 0, s });

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (vis[u])

continue;

vis[u] = true;

if (dp[u] > 1)

ok[u] = true;

for (auto it : G[u]) {

int v = it.second, w = it.first;

if (dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

dp[v] = dp[u];

q.push({ dis[v], v });

}

else if (dis[v] == dis[u] + w)

dp[v] = (dp[v] + dp[u]) % Mod;

}

}

}

int main()

{

cin >> n >> m >> k;

for (int i = 0; i < m; i++) {

int u, v; ll w;

sc("%d %d %lld", &u, &v, &w);

G[u].push\_back({ w, v });

G[v].push\_back({ w, u });

}

int ans = 0;

unordered\_map <int, ll> mp;

for (int i = 1; i <= k; i++) {

int v; ll w;

sc("%d %lld", &v, &w);

if (mp.count(v) == NULL)

mp[v] = w;

else

ans++, Min(mp[v], w);

}

for (auto it : mp) {

ll w = it.second;

int u = 1, v = it.first;

G[u].push\_back({ w, v });

G[v].push\_back({ w, u });

}

Dijkstra(1);

for (auto it : mp) {

int u = it.first;

ll w = it.second;

if (dis[u] < w || ok[u])

ans++;

}

printf("%d\n", ans);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 4.abc不可走->ab&bc不可走

struct node

{

int w;

int u, prv;

};

vector <pir> G[M];

map <pir, int> ev, vis;

bool eg[N][N];

pir edge[M \* 2];

int pre[M \* 2];

int n, m, k;

int idx, ans = INF;

void BFS() {

queue <node> q;

for (auto it : G[1]) {

int v = it.first, id = it.second;

if (v == n) {

printf("1\n1 %d\n", n);

exit(0);

}

q.push({ 1, v, id });

pre[id] = -1;

eg[1][v] = true;

}

while (!q.empty()) {

node now = q.front();

q.pop();

int u = now.u;

if (u == n && now.w < ans)

ans = now.w, idx = now.prv;

for (auto it : G[u]) {

int v = it.first, id = it.second;

if (eg[u][v] || vis.count({ now.prv, id }))

continue;

eg[u][v] = true;

q.push({ now.w + 1, v, id });

pre[id] = now.prv;

}

}

}

int main()

{

cin >> n >> m >> k;

for (int i = 1; i <= 2 \* m; i += 2) {

int u, v;

sc("%d %d", &u, &v);

G[u].push\_back({ v, i });

G[v].push\_back({ u, i + 1 });

ev[{u, v}] = i;

ev[{v, u}] = i + 1;

edge[i] = { u, v };

edge[i + 1] = { v, u };

}

while (k--) {

int a, b, c;

sc("%d %d %d", &a, &b, &c);

vis[{ev[{a, b}], ev[{b, c}]}] = true;

}

BFS();

if (ans == INF)

puts("-1"), exit(0);

printf("%d\n", ans);

vector <int> vec;

for (int i = idx; i != -1; i = pre[i]) {

if (pre[i] == -1)

vec.push\_back(edge[i].second), vec.push\_back(edge[i].first);

else

vec.push\_back(edge[i].second);

}

reverse(ALL(vec));

for (auto it : vec)

cout << it << ' ';

putchar('\n');

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 5.距离s恰好为d的点数可以在边上

vector <pir> G[N];

int dis[N], cost[N];

int n, m, s, l, ans;

bool vis[N];

pir eg[N];

void Dijkstra(int s) {

priority\_queue <pir, vector <pir>, greater <pir>> q;

MEM(dis, INF);

dis[s] = 0, q.push({ 0, s });

int ans = 0;

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (vis[u])

continue;

vis[u] = true;

for (auto it : G[u]) {

int v = it.first, id = it.second;

if (dis[v] > dis[u] + cost[id]) {

dis[v] = dis[u] + cost[id];

q.push({ dis[v], v });

}

}

}

}

int main()

{

cin >> n >> m >> s;

for (int i = 1; i <= m; i++) {

int u, v, w;

sc("%d %d %d", &u, &v, &w);

G[u].push\_back({ v, i });

G[v].push\_back({ u, i });

cost[i] = w;

eg[i] = { u, v };

}

cin >> l;

Dijkstra(s);

for (int i = 1; i <= n; i++)

if (dis[i] == l)

ans++;

for (int i = 1; i <= m; i++) {

int u = eg[i].first, v = eg[i].second;

int d1 = dis[u], d2 = dis[v];

if (d1 > d2)

swap(d1, d2);

if (d1 + cost[i] == d2) { // 是最短路

if (l > d1 && l < d2)

ans++;

}

else {

if (d1 >= l)

continue;

int w = cost[i] - (d2 - d1); // 剩下的长度

if (d2 >= l)

ans++;

else {

if (d2 + w / 2 == l) {

ans++;

if (w & 1)

ans++;

}

else if (d2 + w / 2 > l)

ans += 2;

}

}

}

printf("%d\n", ans);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 6.花费x最多跑y

struct Edge

{

int u, prv; // 当前点，出发点

ll w, t; // 距离，花费

bool operator < (const Edge &oth) const {

if (t == oth.t)

return w > oth.w;

else

return t > oth.t;

}

};

vector <pir> G[N];

ll d[N], c[N];

int n, m, sp, tp;

bool vis[N][N];

ll Dijkstra() {

if (sp == tp)

return 0;

priority\_queue <Edge> q;

q.push({ sp, sp, 0, c[sp] });

while (!q.empty()) {

Edge now = q.top();

q.pop();

int u = now.u;

int prv = now.prv;

if (u == tp)

return now.t;

if (vis[u][prv])

continue;

vis[u][prv] = true;

q.push({ u, u, 0, now.t + c[u] });

for (auto it : G[u]) {

int v = it.second; ll w = it.first;

if (now.w + w <= d[prv])

q.push({ v, prv, now.w + w, now.t });

}

}

return -1;

}

int main()

{

cin >> n >> m >> sp >> tp;

for (int i = 0; i < m; i++) {

int u, v; ll w;

sc("%d %d %lld", &u, &v, &w);

G[u].push\_back({ w, v });

G[v].push\_back({ w, u });

}

for (int i = 1; i <= n; i++)

sc("%lld %lld", &d[i], &c[i]);

printf("%lld\n", Dijkstra());

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 7.lowbit建32个虚点

vector <pir> G[N];

ll dis[N];

bool vis[N];

int n;

void Init() {

for (int i = 1; i <= n + 32; i++)

G[i].clear(), dis[i] = LINF, vis[i] = false;

}

void Add(int u, int v, ll w) {

G[u].push\_back({ w, v });

G[v].push\_back({ w, u });

}

void Dijkstra(int s) { // 根据需要改pir的long long

priority\_queue <pir, vector <pir>, greater <pir>> q;

dis[s] = 0;

q.push({ 0ll, s });

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (vis[u])

continue;

vis[u] = true;

for (auto it : G[u]) {

int v = it.second; ll w = it.first;

if (dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

q.push({ dis[v], v });

}

}

}

}

int main()

{

int T; cin >> T;

while (T--) {

sc("%d", &n);

Init();

for (int i = 1; i <= n; i++) {

ll t;

int cnt0 = 1;

sc("%lld", &t);

while (t) {

if (t & 1)

Add(i, n + cnt0, fpow(2ll, cnt0 - 1));

t >>= 1;

cnt0++;

}

}

Dijkstra(1);

if (dis[n] == LINF)

puts("Impossible");

else

printf("%lld\n", dis[n] / 2);

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 8.更新部分边权使最短路为L

// 正反求最短路，看中间边

vector <pir> G[N];

ll d1[N], d2[N], val[N \* 10], L;

int sp, tp;

int n, m;

pair <int, int> edge[N \* 10];

bool vis[N];

void Dijkstra(int s, ll \*dis, int op) {

priority\_queue <pir, vector <pir>, greater <pir>> q;

MEM(vis, 0);

dis[s] = 0;

q.push({ 0, s });

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (vis[u])

continue;

vis[u] = true;

if (op == 1) {

for (auto it : G[u]) {

int v = it.first, id = it.second;

ll w = val[id];

if (!w)

continue;

if (dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

q.push({ dis[v], v });

}

}

}

else {

for (auto it : G[u]) {

int v = it.first, id = it.second;

if (!val[id])

val[id] = max(1ll, L - d2[v] - d1[u]);

ll w = val[id];

if (dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

q.push({ dis[v], v });

}

}

}

}

}

int main()

{

MEM(d1, LINF), MEM(d2, LINF);

cin >> n >> m >> L >> sp >> tp;

for (int i = 1; i <= m; i++) {

int u, v; ll w;

sc("%d %d %lld", &u, &v, &w);

G[u].push\_back({ v, i });

G[v].push\_back({ u, i });

edge[i] = { u, v };

val[i] = w;

}

Dijkstra(tp, d2, 1);

if (d2[sp] < L)

puts("NO"), exit(0);

Dijkstra(sp, d1, 2);

if (d1[tp] != L)

puts("NO"), exit(0);

printf("YES\n");

for (int i = 1; i <= m; i++)

printf("%d %d %lld\n", edge[i].first, edge[i].second, val[i]);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 9.地铁线路按边算最短路

// 通过求边间接得到点

struct Edge

{

ll w;

int v, c, id;

bool operator < (const Edge &oth) const {

return w > oth.w;

}

};

vector <Edge> G[N];

ll dis[N], n, m; // dis为到每条边的最短路

bool vis[N];

ll Dijkstra(int x) {

priority\_queue <Edge> q;

MEM(dis, LINF);

for (auto it : G[x]) {

q.push({ it.w, it.v, it.c, it.id });

dis[it.id] = it.w;

}

ll ans = LINF;

while (!q.empty()) {

Edge now = q.top();

q.pop();

int u = now.v;

int id = now.id;

if (vis[id])

continue;

vis[id] = true;

if (u == n)

Min(ans, dis[id]);

for (auto it : G[u]) {

ll cost = it.w + abs(now.c - it.c);

int v = it.v, nxt = it.id;

if (dis[nxt] > dis[id] + cost) {

dis[nxt] = dis[id] + cost;

q.push({ dis[nxt], v, it.c, nxt });

}

}

}

return ans;

}

int main()

{

cin >> n >> m;

for (int i = 1; i <= m; i++) {

int u, v, c; ll w;

sc("%d %d %d %lld", &u, &v, &c, &w);

G[u].push\_back({ w, v, c, i });

G[v].push\_back({ w, u, c, i });

}

printf("%lld\n", Dijkstra(1));

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 10.多源最短路加油站询问两点是否可达

//通过多源最短路然后合并累积在加油站之间建边

struct node

{

int u, v, w;

bool operator < (const node &oth) const {

return w < oth.w;

}

}edge[N << 1];

int cnt;

vector <pir> G[N << 1], e[N << 1];

int dis[N], fz[N], n, m;

int a[N], near[N], k;

bool vis[N];

int find\_(int x) {

if (x != fz[x])

x = fz[x] = find\_(fz[x]);

return x;

}

void Dijkstra() { // 根据需要改pir的long long

priority\_queue <pir, vector <pir>, greater <pir>> q;

MEM(dis, INF);

for (int i = 1; i <= k; i++) {

q.push({ 0, a[i] });

near[a[i]] = a[i];

dis[a[i]] = 0;

}

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (vis[u])

continue;

vis[u] = true;

for (auto it : G[u]) {

int v = it.second, w = it.first;

if (dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

q.push({ dis[v], v });

near[v] = near[u];

}

}

}

}

void Kruscal() {

sort(edge + 1, edge + cnt + 1);

for (int i = 1; i <= cnt; i++) {

int u = edge[i].u, v = edge[i].v;

int w = edge[i].w;

int uu = find\_(edge[i].u);

int vv = find\_(edge[i].v);

if (uu == vv)

continue;

fz[uu] = vv;

e[u].push\_back({ w, v });

e[v].push\_back({ w, u });

}

}

// 树上ST表

int dep[N], f[N][25], st[N][25];

void DFS(int x, int fa) {

dep[x] = dep[fa] + 1;

f[x][0] = fa;

for (int i = 1; (1 << i) <= dep[x]; i++) {

f[x][i] = f[f[x][i - 1]][i - 1];

st[x][i] = max(st[x][i - 1], st[f[x][i - 1]][i - 1]);

}

for (auto it : e[x]) {

int v = it.second;

int w = it.first;

if (v != fa)

st[v][0] = w, DFS(v, x);

}

}

int LCA(int x, int y) {

int mx = -INF;

if (dep[x] > dep[y])

swap(x, y);

for (int i = 20; i >= 0; i--) {

if (dep[y] - (1 << i) >= dep[x])

Max(mx, st[y][i]), y = f[y][i];

}

if (x == y)

return mx;

for (int i = 20; i >= 0; i--) {

if (f[x][i] == f[y][i]) continue;

Max(mx, max(st[x][i], st[y][i]));

x = f[x][i], y = f[y][i];

}

return max(mx, max(st[x][0], st[y][0])); // 最后要取max

}

int main()

{

cin >> n >> k >> m;

for (int i = 1; i <= k; i++)

sc("%d", &a[i]);

for (int i = 1; i <= m; i++) {

int u, v, w;

sc("%d %d %d", &u, &v, &w);

G[u].push\_back({ w, v });

G[v].push\_back({ w, u });

}

// 找到距离每个点最近的加油站

Dijkstra();

// 加油站之间的边

for (int i = 1; i <= n; i++) {

fz[i] = i;

for (auto it : G[i]) {

int u = i, v = it.second;

int w = it.first;

if (near[u] != near[v])

edge[++cnt] = { near[u], near[v], dis[u] + dis[v] + w };

}

}

Kruscal();

for (int i = 1; i <= k; i++) {

if (!dep[a[i]])

DFS(a[i], 0);

}

int q;

cin >> q;

while (q--) {

int u, v, w;

sc("%d %d %d", &u, &v, &w);

if (find\_(u) != find\_(v))

puts("NIE");

else {

int dist = LCA(u, v);

if (dist <= w)

puts("TAK");

else

puts("NIE");

}

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 11.K种颜色询问最短路

// 预处理k种颜色到每个点最短路dis[k][x][y]，遇到新颜色把该颜色所有点全部加进去，最后查询枚举中间点

struct node

{

int x, y, step;

};

vector <pir> col[45];

int dir[4][2] = { 1, 0, -1, 0, 0, 1, 0, -1 };

int a[N][N], n, m, k;

int d[45][N][N];

bool vis[N][N], ID[45];

void init() {

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= m; j++) {

vis[i][j] = false;

}

}

for (int i = 1; i <= k; i++)

ID[i] = false;

}

void BFS(int id) {

init();

queue <node> q;

for (auto it : col[id]) {

int nx = it.first, ny = it.second;

q.push({ nx, ny, 0 });

vis[nx][ny] = true;

d[id][nx][ny] = 0;

}

ID[id] = true;

while (!q.empty()) {

node now = q.front();

q.pop();

int x = now.x, y = now.y;

int w = a[x][y];

// 同色

if (!ID[w]) {

ID[w] = true;

for (auto it : col[w]) {

int mx = it.first, my = it.second;

if (vis[mx][my])

continue;

d[id][mx][my] = now.step + 1;

vis[mx][my] = true;

q.push({ mx, my, now.step + 1 });

}

}

// 相邻

for (int i = 0; i < 4; i++) {

int xx = x + dir[i][0];

int yy = y + dir[i][1];

if (xx < 1 || xx > n || yy < 1 || yy > m || vis[xx][yy] || ID[a[xx][yy]])

continue;

q.push({ xx, yy, now.step + 1 });

d[id][xx][yy] = now.step + 1;

vis[xx][yy] = true;

}

}

}

int main()

{

cin >> n >> m >> k;

MEM(d, INF);

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= m; j++) {

sc("%d", &a[i][j]);

col[a[i][j]].push\_back({ i, j });

}

}

for (int i = 1; i <= k; i++)

BFS(i);

int q;

cin >> q;

while (q--) {

int x1, y1, x2, y2;

sc("%d %d %d %d", &x1, &y1, &x2, &y2);;

int ans = abs(x1 - x2) + abs(y1 - y2);

for (int i = 1; i <= k; i++)

Min(ans, d[i][x1][y1] + d[i][x2][y2] + 1);

printf("%d\n", ans);

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 12.最多100个环查询最短路

// 先求树，剩下的边暴力

// 可以推广成少的边暴力大的边数据结构维护

// 也可以先跑附加边 \* M，询问的时候枚举即可

vector <pir> G[N];

bool vis[N], path[N << 1]; // 走过的点，边

int ans[N << 1], n, m, Q;

pir edge[N << 1], que[N << 1];

// LCA倍增

int dep[N], dis[N], f[N][25];

void dfs(int x, int fa, int fid) {

dep[x] = dep[fa] + 1;

f[x][0] = fa;

vis[x] = true;

path[fid] = true;

for (int i = 1; (1 << i) <= dep[x]; i++)

f[x][i] = f[f[x][i - 1]][i - 1];

for (auto it : G[x]) {

int v = it.first, id = it.second;

if (vis[v])

continue;

dis[v] = dis[x] + 1;

dfs(v, x, id);

}

}

int LCA(int x, int y) {

if (dep[x] > dep[y])

swap(x, y);

for (int i = 20; i >= 0; i--) {

if (dep[y] - (1 << i) >= dep[x])

y = f[y][i];

}

if (x == y)

return x;

for (int i = 20; i >= 0; i--) {

if (f[x][i] == f[y][i])

continue;

x = f[x][i], y = f[y][i];

}

return f[x][0];

}

// 暴力枚举

int d[N];

void BFS(int x, int y) {

queue <int> q;

for (int i = 1; i <= n; i++)

d[i] = INF;

q.push(x), q.push(y);

d[x] = d[y] = 0;

while (!q.empty()) {

int u = q.front();

q.pop();

for (auto it : G[u]) {

int v = it.first;

if (d[v] > d[u] + 1) {

d[v] = d[u] + 1;

q.push(v);

}

}

}

for (int i = 1; i <= Q; i++) {

int u = que[i].first;

int v = que[i].second;

Min(ans[i], d[u] + d[v] + 1);

}

}

int main()

{

cin >> n >> m;

for (int i = 1; i <= m; i++) {

int u, v;

sc("%d %d", &u, &v);

G[u].push\_back({ v, i });

G[v].push\_back({ u, i });

edge[i] = { u, v };

}

dfs(1, 0, 0);

cin >> Q;

for (int i = 1; i <= Q; i++) {

int u, v;

sc("%d %d", &u, &v);

que[i] = { u, v };

ans[i] = dis[u] + dis[v] - 2 \* dis[LCA(u, v)];

}

for (int i = 1; i <= m; i++) {

if (path[i])

continue;

BFS(edge[i].first, edge[i].second);

}

for (int i = 1; i <= Q; i++)

printf("%d\n", ans[i]);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 13.多米诺骨牌最短路

struct Edge

{

int w, id;

bool operator < (const Edge &oth) const {

return w > oth.w;

}

};

vector <pir> G[N << 1];

vector <int> edge[N];

int dis[N], wt[N \* N], n, m;

pir node[N \* N];

int ID[3];

bool vis[N];

void init() {

for (int i = 1; i <= n; i++)

G[i].clear();

for (int i = 1; i <= m; i++)

edge[i].clear();

}

void dijkstra(int x) {

priority\_queue <Edge> q;

MEM(dis, INF), MEM(vis, 0);

q.push({ 0, 1 });

dis[1] = 0;

while (!q.empty()) {

Edge now = q.top();

q.pop();

int u = now.id;

if (vis[u])

continue;

vis[u] = true;

for (int i = 0; i < SZ(G[u]); i++) {

int v = G[u][i].first;

int id = G[u][i].second;

edge[id].push\_back(dis[u]);

if (dis[v] > dis[u] + wt[id]) {

dis[v] = dis[u] + wt[id];

q.push({ dis[v], v });

}

}

}

}

int main()

{

int Case = 0;

while (sc("%d %d", &n, &m) && n + m) {

init();

for (int i = 1; i <= m; i++) {

int u, v, w;

sc("%d %d %d", &u, &v, &w);

G[u].push\_back({ v, i });

G[v].push\_back({ u, i });

wt[i] = w;

if (u > v)

swap(u, v);

node[i] = { u, v };

}

dijkstra(1);

double mx = 0.0;

for (int i = 1; i <= m; i++) {

sort(ALL(edge[i]));

int l = edge[i][0];

int r = edge[i][1];

int w = wt[i];

if (l + w <= r) {

if (mx < (double)l + w) {

mx = (double)l + w;

ID[2] = 0;

if (dis[node[i].first] == r)

ID[1] = node[i].first;

else

ID[1] = node[i].second;

}

}

else {

if (mx < (double)r + (l + w - r) / 2.0) {

mx = (double)r + (l + w - r) / 2.0;

ID[1] = node[i].first;

ID[2] = node[i].second;

}

}

}

printf("System #%d\n", ++Case);

printf("The last domino falls after %.1f seconds, ", mx);

if (n == 1)

printf("at key domino 1.\n");

else if (ID[2])

printf("between key dominoes %d and %d.\n", ID[1], ID[2]);

else

printf("at key domino %d.\n", ID[1]);

printf("\n");

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 14.K条绳子烧完时间（记到边最短路）

struct node

{

int id; ll w;

bool operator < (const node &oth) const {

return w > oth.w;

}

};

vector <pir> G[N << 1];

vector <ll> vec[N];

ll dis[N], val[N];

bool vis[N];

int n, k;

void dijkstra() {

priority\_queue <node> q;

MEM(dis, LINF);

for (int i = 0; i < k; i++) {

int t;

sc("%d", &t);

dis[t] = 0;

q.push({ t, 0ll }); // 加入队列

}

while (!q.empty()) {

node now = q.top();

q.pop();

int u = now.id;

if (vis[u])

continue;

vis[u] = true;

for (auto it : G[u]) {

int v = it.second, id = it.first;

vec[id].push\_back(dis[u]);

if (dis[v] > dis[u] + val[id]) {

dis[v] = dis[u] + val[id];

q.push({ v, dis[v] });

}

}

}

}

int main()

{

cin >> n >> k;

for (int i = 1; i < n; i++) {

int u, v;

sc("%d %d %lld", &u, &v, &val[i]);

G[u].push\_back({ i, v });

G[v].push\_back({ i, u });

}

dijkstra();

ll ans = 0;

for (int i = 1; i < n; i++) {

ll l = vec[i][0], r = vec[i][1];

ll w = val[i];

if (l > r)

swap(l, r);

if (l + w <= r)

Max(ans, 2ll \* (l + w));

else

Max(ans, 2 \* r + (l + w - r));

}

printf("%lld\n", ans);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 15.有节点保护最短路

vector <pir> G[N];

vector <int> lim[N];

int low[N];

ll dis[N];

bool vis[N];

void dijkstra(int x) {

priority\_queue <pir, vector <pir>, greater <pir>> q;

MEM(dis, LINF);

dis[x] = 0;

q.push({ 0ll, x });

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (vis[u] || low[u])

continue;

vis[u] = true;

for (auto v : lim[u]) {

low[v]--;

Max(dis[v], dis[u]);

if (!low[v])

q.push({ dis[v], v });

}

for (auto it : G[u]) {

int v = it.second; ll w = it.first;

if (dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

q.push({ dis[v], v });

}

}

}

}

int main()

{

int n, m;

cin >> n >> m;

for (int i = 0; i < m; i++) {

int u, v; ll w;

sc("%d %d %lld", &u, &v, &w);

G[u].push\_back({ w, v });

}

for (int i = 1; i <= n; i++) {

int t, v;

sc("%d", &t);

low[i] = t;

while (t--) {

sc("%d", &v);

lim[v].push\_back(i);

}

}

dijkstra(1);

printf("%lld\n", dis[n]);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 16.平面切割无法从S到T

//离散化左上角和右下角即可

struct node

{

int x, y; ll w;

bool operator < (const node &oth) const {

return w > oth.w;

}

};

int dir[4][2] = { 1, 0, -1, 0, 0, 1, 0, -1 };

ll a[N][N], dis[N][N];

bool vis[N][N];

int n, m, q;

void init() {

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= m; j++) {

dis[i][j] = LINF;

vis[i][j] = false;

}

}

}

ll bfs() {

priority\_queue <node> q;

init(); // 初始化

for (int i = 2; i < m; i++) { // 上面

if (a[1][i] == -1)

q.push({ 1, i, 0 }), dis[1][i] = 0ll;

else if (a[1][i] > 0)

q.push({ 1, i, a[1][i] }), dis[1][i] = a[1][i];

}

for (int i = 1; i < n; i++) { // 右边

if (a[i][m] == -1)

q.push({ i, m, 0 }), dis[i][m] = 0ll;

else if (a[i][m] > 0)

q.push({ i, m, a[i][m] }), dis[i][m] = a[i][m];

}

while (!q.empty()) {

node now = q.top();

q.pop();

int x = now.x, y = now.y;

if (vis[x][y])

continue;

vis[x][y] = true;

if (x == n || y == 1) //切割成功

return dis[x][y];

for (int i = 0; i < 4; i++) {

int xx = now.x + dir[i][0];

int yy = now.y + dir[i][1];

if (xx < 1 || xx > n || yy < 1 || yy > m || !a[xx][yy]) // 越界或不可走

continue;

if (a[xx][yy] == -1) {

dis[xx][yy] = now.w;

q.push({ xx, yy, dis[xx][yy] });

}

else if (dis[xx][yy] > dis[x][y] + a[xx][yy]) {

dis[xx][yy] = dis[x][y] + a[xx][yy];

q.push({ xx, yy, dis[xx][yy] });

}

}

}

return -1;

}

int main()

{

cin >> q >> n >> m;

while (q--) {

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= m; j++) {

sc("%lld", &a[i][j]);

}

}

printf("%lld\n", bfs());

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 17.平面&直线&圆离散最短路

struct Circle

{

double x, y, r;

} e[N];

vector <pir> G[N << 1];

int n, a, b, c1, c2;

double dis[N];

bool vis[N];

void init() {

for (int i = 1; i <= n + 2; i++)

dis[i] = 1000000000.0;

}

double NodeLine(double x, double y, double a, double b, double c) {

return abs(a \* x + b \* y + c) / sqrt(a \* a + b \* b);

}

double Node(double x1, double y1, double x2, double y2) {

return sqrt(P2(x1 - x2) + P2(y1 - y2));

}

void add(int u, int v, double w) {

G[u].push\_back({ w, v });

}

void dijkstra(int s) { // 根据需要改pir的long long

priority\_queue <pir, vector <pir>, greater <pir>> q;

dis[s] = 0.0;

q.push({ 0.0, s });

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (vis[u])

continue;

vis[u] = true;

for (auto it : G[u]) {

int v = it.second; double w = it.first;

if (dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

q.push({ dis[v], v });

}

}

}

}

int main()

{

cin >> n >> a >> b >> c1 >> c2;

init();

for (int i = 1; i <= n; i++)

sc("%lf %lf %lf", &e[i].x, &e[i].y, &e[i].r);

double dist = abs(c1 - c2) / sqrt(a \* a + b \* b);

add(n + 1, n + 2, dist); // 直线之间的距离

add(n + 2, n + 1, dist);

for (int i = 1; i <= n; i++) {

double x1 = e[i].x, y1 = e[i].y;

double r1 = e[i].r;

double spi = NodeLine(x1, y1, a, b, c1);

spi -= r1;

Max(spi, 0.0);

add(i, n + 1, spi), add(n + 1, i, spi); // 圆和起点线

double tpi = NodeLine(x1, y1, a, b, c2);

tpi -= r1;

Max(tpi, 0.0);

add(i, n + 2, tpi), add(n + 2, i, tpi); // 圆和终点线

for (int j = 1; j <= n; j++) {

if (i == j)

continue;

double x2 = e[j].x, y2 = e[j].y;

double r2 = e[j].r;

double dist = Node(x1, y1, x2, y2);

dist -= (r1 + r2);

Max(dist, 0.0);

add(i, j, dist); // 圆与圆之间距离

}

}

dijkstra(n + 1);

printf("%.10f\n", dis[n + 2]);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 18.离散化附加边枚举中间点

vector <pir> G[N];

ll dis[45][N], pre[N];

int mp[N], ev[N];

int n, m;

bool vis[N];

void dijkstra(int s) { // 根据需要改pir的long long

priority\_queue <pir, vector <pir>, greater <pir>> q;

MEM(vis, 0);

dis[s][ev[s]] = 0;

q.push({ 0, ev[s] });

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (vis[u])

continue;

vis[u] = true;

for (auto it : G[u]) {

int v = it.second; ll w = it.first;

if (dis[s][v] > dis[s][u] + w) {

dis[s][v] = dis[s][u] + w;

q.push({ dis[s][v], v });

}

}

}

}

ll dist(int x, int y) {

if (x > y)

swap(x, y);

ll tot = pre[y - 1] - pre[x - 1];

return min(tot, pre[x - 1] + pre[n] - pre[y - 1]);

}

int main()

{

cin >> n >> m;

MEM(dis, LINF);

for (int i = 1; i <= n; i++) {

sc("%lld", &pre[i]);

int nxt = (i % n) + 1;

G[i].push\_back({ pre[i], nxt });

G[nxt].push\_back({ pre[i], i });

pre[i] += pre[i - 1];

}

int tot = 0;

for (int i = 0; i < m; i++) {

int u, v; ll w;

sc("%d %d %lld", &u, &v, &w);

if (!mp[u])

mp[u] = ++tot, ev[tot] = u;

if (!mp[v])

mp[v] = ++tot, ev[tot] = v;

G[u].push\_back({ w, v });

G[v].push\_back({ w, u });

}

for (int i = 1; i <= tot; i++) // 离散最短路

dijkstra(i);

int q;

cin >> q;

while (q--) {

int u, v;

sc("%d %d", &u, &v);

ll ans = dist(u, v);

for (int i = 1; i <= tot; i++)

Min(ans, dis[i][u] + dis[i][v]);

printf("%lld\n", ans);

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 19.S->M->T枚举中间点最长

vector <pir> G[N];

int dis[N], n, m, ans;

bool vis[N];

void init() {

for (int i = 1; i <= n; i++)

G[i].clear();

ans = -1;

}

void dijkstra(int s) { // 根据需要改pir的long long

priority\_queue <pir, vector <pir>, greater <pir>> q;

MEM(dis, INF); MEM(vis, 0);

dis[s] = 0;

q.push({ 0, s });

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (vis[u])

continue;

vis[u] = true;

for (auto it : G[u]) {

int v = it.second, w = it.first;

if (dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

q.push({ dis[v], v });

}

}

}

sort(dis + 1, dis + n + 1, greater <int>());

int j = 1;

while (j <= n && dis[j] == INF)

j++;

if (j < n - 1)

Max(ans, dis[j] + dis[j + 1]);

}

int main()

{

int T; cin >> T;

while (T--) {

sc("%d %d", &n, &m);

init();

for (int i = 0; i < m; i++) {

int u, v, w;

sc("%d %d %d", &u, &v, &w);

G[u].push\_back({ w, v });

G[v].push\_back({ w, u });

}

for (int i = 1; i <= n; i++)

dijkstra(i);

printf("%d\n", ans);

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 20.必经过K个点的最短路对K离散化

vector <pir> G[N];

int dis[N], a[12], b[12];

int d[12][N];

bool vis[N];

void dijkstra(int s) { // 根据需要改pir的long long

priority\_queue <pir, vector <pir>, greater <pir>> q;

MEM(dis, INF); MEM(vis, 0);

dis[s] = 0;

q.push({ 0, s });

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (vis[u])

continue;

vis[u] = true;

for (auto it : G[u]) {

int v = it.second, w = it.first;

if (dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

q.push({ dis[v], v });

}

}

}

}

int main()

{

int T; cin >> T;

while (T--) {

int n, m;

sc("%d %d", &n, &m);

for (int i = 0; i < n + 5; i++)

G[i].clear();

for (int i = 0; i < m; i++) {

int u, v, w;

sc("%d %d %d", &u, &v, &w);

G[u].push\_back({ w, v });

G[v].push\_back({ w, u });

}

int q;

sc("%d", &q);

for (int i = 1; i <= q; i++) {

b[i] = i;

sc("%d", &a[i]);

dijkstra(a[i]);

for (int j = 0; j < n; j++)

d[i][j] = dis[j];

}

int ans = INF;

do {

int tot = 0;

tot += d[b[1]][0];

for (int i = 1; i < q; i++)

tot += d[b[i]][a[b[i + 1]]];

tot += d[b[q]][0];

Min(ans, tot);

} while (next\_permutation(b + 1, b + q + 1));

printf("%d\n", ans);

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 21.单调性限制前K个点不能切换模式

vector <pir> Ga[N], Gb[N];

ll dsa[N], dsb[N], dta[N], dtb[N];

ll ans[N];

bool vis[N];

void init() {

MEM(dsa, LINF), MEM(dsb, LINF), MEM(dta, LINF), MEM(dtb, LINF);

}

void dijkstra(int x, int op, ll \*dis) {

priority\_queue <pir, vector <pir>, greater <pir>> q;

MEM(vis, 0);

dis[x] = 0;

q.push({ 0, x });

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (vis[u])

continue;

vis[u] = true;

if (op == 1) {

for (auto it : Ga[u]) {

int v = it.second, w = it.first;

if (dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

q.push({ dis[v], v });

}

}

}

else {

for (auto it : Gb[u]) {

int v = it.second, w = it.first;

if (dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

q.push({ dis[v], v });

}

}

}

}

}

int main()

{

int n, m;

cin >> n >> m;

init();

for (int i = 0; i < m; i++) {

int u, v, a, b;

sc("%d %d %d %d", &u, &v, &a, &b);

Ga[u].push\_back({ a, v });

Ga[v].push\_back({ a, u });

Gb[u].push\_back({ b, v });

Gb[v].push\_back({ b, u });

}

int s, t;

cin >> s >> t;

dijkstra(s, 1, dsa), dijkstra(t, 1, dta);

dijkstra(s, 2, dsb), dijkstra(t, 2, dtb);

ll me = LINF;

for (int i = n; i >= 1; i--) {

ll dist = dsa[i] + dtb[i];

if (dist < me)

me = dist;

ans[i] = me;

}

for (int i = 1; i <= n; i++)

printf("%lld\n", ans[i]);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 22.AGCT找2，3，4环

int a[30][30], d[30][30], ans;

char s1[N], s2[N];

void floyd2() {

for (int i = 1; i <= 26; i++) {

for (int j = 1; j <= 26; j++) {

if (a[i][j] && a[j][i]) {

int min\_ = min(a[i][j], a[j][i]);

ans += min\_;

a[i][j] -= min\_, a[j][i] -= min\_;

}

}

}

}

void floyd3() {

for (int i = 1; i <= 26; i++) {

for (int j = 1; j <= 26; j++){

for (int k = 1; k <= 26; k++) {

if (a[i][j] && a[j][k] && a[k][i]) {

int min\_ = min({ a[i][j], a[j][k], a[k][i] });

ans += min\_ \* 2;

a[i][j] -= min\_, a[j][k] -= min\_, a[k][i] -= min\_;

}

}

}

}

}

void floyd4() {

for (int i = 1; i <= 26; i++) {

for (int j = 1; j <= 26; j++){

for (int k = 1; k <= 26; k++) {

for (int l = 1; l <= 26; l++) {

if (a[i][j] && a[j][k] && a[k][l] && a[l][i]) {

int min\_ = min({ a[i][j], a[j][k], a[k][l], a[l][i] });

ans += min\_ \* 3;

a[i][j] -= min\_, a[j][k] -= min\_, a[k][l] -= min\_, a[l][i] -= min\_;

}

}

}

}

}

}

int main()

{

sc("%s %s", s1 + 1, s2 + 1);

int sz = strlen(s1 + 1);

for (int i = 1; i <= sz; i++) {

if (s1[i] != s2[i])

a[s1[i] - 'A' + 1][s2[i] - 'A' + 1]++;

}

floyd2();

floyd3();

floyd4();

cout << ans << endl;

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 23.a->b->c分配边权枚举汇点

vector <int> G[N];

int n, m, a, b, c;

int d1[N], d2[N], d3[N];

ll e[N];

void init() {

for (int i = 1; i <= n; i++)

G[i].clear(), d1[i] = d2[i] = d3[i] = INF;

}

void bfs(int s, int \*dis) {

queue <pir> q;

q.push({ s, 0 });

dis[s] = 0;

while (!q.empty()) {

pir now = q.front();

q.pop();

int u = now.first;

for (auto v : G[u]) {

if (dis[v] > dis[u] + 1) {

dis[v] = dis[u] + 1;

q.push({ v, dis[v] });

}

}

}

}

int main()

{

int T; cin >> T;

while (T--) {

sc("%d %d %d %d %d", &n, &m, &a, &b, &c);

init();

for (int i = 1; i <= m; i++)

sc("%lld", &e[i]);

sort(e + 1, e + m + 1);

for (int i = 1; i <= m; i++)

e[i] += e[i - 1];

for (int i = 0; i < m; i++) {

int ui, vi;

sc("%d %d", &ui, &vi);

G[ui].push\_back(vi);

G[vi].push\_back(ui);

}

bfs(a, d1); bfs(b, d2); bfs(c, d3);

ll ans = LINF;

for (int i = 1; i <= n; i++) {

int bx = d2[i];

int ac = d1[i] + d3[i];

if (bx + ac > m)

continue;

Min(ans, e[bx] + e[bx + ac]);

}

printf("%lld\n", ans);

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 24.一条边反向是否改变dis（正反跑）

struct node

{

int u, v; ll w;

}p[N << 1];

vector <pir> G[N << 1], e[N << 1];

ll dis1[N], dis2[N];

bool vis1[N], vis2[N];

int n, m, q;

void dijkstra1(int x) {

MEM(dis1, INF);

priority\_queue <pir, vector <pir>, greater <pir>> q;

q.push({ 0, x }); dis1[x] = 0;

while (!q.empty()) {

pir now = q.top();

q.pop();

int ans = now.second;

if (vis1[ans])

continue;

vis1[ans] = true;

for (auto it : G[ans]) {

int vi = it.second; ll wi = it.first;

if (dis1[vi] > dis1[ans] + wi) {

dis1[vi] = dis1[ans] + wi;

q.push({ dis1[vi], vi });

}

}

}

}

void dijkstra2(int x) {

MEM(dis2, INF);

priority\_queue <pir, vector <pir>, greater <pir>> q;

q.push({ 0, x }); dis2[x] = 0;

while (!q.empty()) {

pir now = q.top();

q.pop();

int ans = now.second;

if (vis2[ans])

continue;

vis2[ans] = true;

for (auto it : e[ans]) {

int vi = it.second; ll wi = it.first;

if (dis2[vi] > dis2[ans] + wi) {

dis2[vi] = dis2[ans] + wi;

q.push({ dis2[vi], vi });

}

}

}

}

int main()

{

cin >> n >> m;

for (int i = 1; i <= m; i++) {

int ui, vi; ll wi;

sc("%d %d %lld", &ui, &vi, &wi);

G[ui].push\_back({ wi, vi });

e[vi].push\_back({ wi, ui });

p[i] = { ui, vi, wi };

}

dijkstra1(1); dijkstra2(n);

cin >> q;

while (q--) {

int id;

sc("%d", &id);

int ui = p[id].u, vi = p[id].v;

ll wi = p[id].w;

if (dis1[vi] + dis2[ui] + wi < dis1[n])

printf("YES\n");

else

printf("NO\n");

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 25.差分约束裸题

struct node

{

int w;

int v, nxt;

}G[N << 1];

int h[N], idx;

int dis[N], cur[N], n, m;

bool vis[N];

void Add(int u, int v, int w) {

G[++idx] = { w, v, h[u] };

h[u] = idx;

}

bool Spfa(int x) {

queue <int> q;

for (int i = 1; i <= n; i++)

dis[i] = INF, vis[i] = false;

q.push(x), vis[x] = true;

cur[x++];

while (!q.empty()) {

int u = q.front();

q.pop();

vis[u] = false;

for (int i = h[u]; i; i = G[i].nxt) {

int v = G[i].v, w = G[i].w;

if (dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

if (vis[v])

continue;

cur[v]++;

vis[v] = true, q.push(v);

if (cur[v] >= n)

return false;

}

}

}

return true;

}

int main()

{

cin >> n >> m;

for (int i = 0; i < m; i++) {

int op, u, v, w;

sc("%d %d %d", &op, &u, &v);

if (op == 1) { // dis[v] <= dis[u] - w

sc("%d", &w);

Add(u, v, -w);

}

else if (op == 2) { // dis[u] <= dis[v] + w

sc("%d", &w);

Add(v, u, w);

}

else // dis[u] <= dis[v] + 0

Add(v, u, 0);

}

for (int i = 1; i <= n; i++)

Add(0, i, 0);

if (!Spfa(0))

puts("No");

else

puts("Yes");

return 0; // 改数组大小!!!用pair改宏定义!!!

}

#### 26.道路与航线

vector <pir> G[N];

int dfn[N], m1, m2, tot;

int in[N], n, s;

ll dis[N];

bool vis[N];

priority\_queue <pir, vector <pir>, greater <pir>> q;

queue <int> dq;

void dfs(int x) {

dfn[x] = tot;

for (auto it : G[x]) {

if (!dfn[it.second])

dfs(it.second);

}

}

void dijkstra(int x) {

for (int i = 1; i <= n; i++) {

if (dfn[i] == x)

q.push({ dis[i], i });

}

while (!q.empty()) {

pir now = q.top();

q.pop();

int ans = now.second;

if (vis[ans])

continue;

vis[ans] = true;

for (auto it : G[ans]) {

int vi = it.second, wi = it.first;

if (dis[vi] > dis[ans] + wi) {

dis[vi] = dis[ans] + wi;

if (dfn[vi] == x)

q.push({ dis[vi], vi });

}

if (dfn[vi] != x && !--in[dfn[vi]])

dq.push(dfn[vi]);

}

}

}

void topsort() {

dis[s] = 0;

for (int i = 1; i <= tot; i++) {

if (!in[i])

dq.push(i);

}

while (!dq.empty()) {

int ans = dq.front();

dq.pop();

dijkstra(ans);

}

}

int main()

{

cin >> n >> m1 >> m2 >> s;

MEM(dis, INF);

for (int i = 0; i < m1; i++) {

int ui, vi; ll wi;

sc("%d %d %lld", &ui, &vi, &wi);

G[ui].push\_back({ wi, vi });

G[vi].push\_back({ wi, ui });

}

for (int i = 1; i <= n; i++) {

if (!dfn[i])

tot++, dfs(i);

}

for (int i = 0; i < m2; i++) {

int ui, vi; ll wi;

sc("%d %d %lld", &ui, &vi, &wi);

G[ui].push\_back({ wi, vi });

in[dfn[vi]]++;

}

topsort();

for (int i = 1; i <= n; i++) {

if (dis[i] > 1e9)

printf("NO PATH\n");

else

printf("%d\n", dis[i]);

}

return 0; // 改数组大小!!!

}

#### 27.逆行一次S能经过最多点

// 正反跑最长路，枚举汇点

vector <pir> g[N];

vector <int> G[N];

int dfn[N], low[N], idx, scnt;

int stk[N], n, m, s;

int cnt[N], suo[N];

int d1[N], d2[N];

bool vis[N];

int pos;

void Add(int u, int v) { // 偶数正，奇数反

g[u].push\_back({ v, pos++ });

g[v].push\_back({ u, pos++ });

}

void Tarjan(int x) {

dfn[x] = low[x] = ++idx;

stk[++s] = x;

vis[x] = true;

for (auto v : G[x]) {

if (!dfn[v]) {

Tarjan(v);

Min(low[x], low[v]);

}

else if (vis[v])

Min(low[x], low[v]);

}

if (dfn[x] == low[x]) {

scnt++;

int k;

do {

k = stk[s--];

suo[k] = scnt;

cnt[scnt]++;

vis[k] = false;

} while (k != x);

}

}

void Spfa(int x, int k, int \*dis) {

for (int i = 1; i <= n; i++)

vis[i] = false;

queue <int> q;

dis[x] = cnt[x];

q.push(x), vis[x] = true;

while (!q.empty()) {

int u = q.front();

q.pop();

vis[u] = false;

for (auto it : g[u]) {

int v = it.first, id = it.second;

if (id % 2 != k)

continue;

if (dis[v] < dis[u] + cnt[v]) {

dis[v] = dis[u] + cnt[v];

if (!vis[v])

q.push(v), vis[v] = true;

}

}

}

}

int main()

{

cin >> n >> m;

for (int i = 0; i < m; i++) {

int u, v;

sc("%d %d", &u, &v);

G[u].push\_back(v);

}

for (int i = 1; i <= n; i++) {

if (!dfn[i])

Tarjan(i);

}

for (int i = 1; i <= n; i++) {

for (auto it : G[i]) {

int u = suo[i], v = suo[it];

if (u != v)

Add(u, v);

}

}

Spfa(suo[1], 0, d1), Spfa(suo[1], 1, d2);

int ans = cnt[suo[1]];

for (int i = 1; i <= scnt; i++) {

if (!d1[i])

continue;

for (auto it : g[i]) {

int v = it.first, id = it.second;

if (!d2[v])

continue;

if (id & 1)

Max(ans, d1[i] + d2[v] - cnt[suo[1]]);

}

}

printf("%d\n", ans);

return 0; // 改数组大小!!!用pair改宏定义!!!

}

#### 28.小雨坐地铁分层图最短路

// N个车站M个地铁线，进线有花费，下一站有花费，相同站不同线路可以互相走

// 分层，每次把不同层相同站入队

vector <pir> G[N];

set <int> st[N];

int n, m, s, t;

ll a[N], dis[N];

bool vis[N];

void Add(int u, int v, ll w) {

G[u].push\_back({ w, v });

G[v].push\_back({ w, u });

}

void Dijkstra() { // 根据需要改pir的long long

priority\_queue <pir, vector <pir>, greater <pir>> q;

MEM(dis, LINF);

for (int i = 1; i <= m; i++) {

q.push({ a[i], (i - 1) \* n + s });

dis[(i - 1) \* n + s] = a[i];

}

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (vis[u])

continue;

vis[u] = true;

int cu = u % n;

if (!cu)

cu = n;

for (auto it : st[cu]) {

int v = (it - 1) \* n + cu;

if (dis[v] > dis[u] + a[it]) {

dis[v] = dis[u] + a[it];

q.push({ dis[v], v });

}

}

for (auto it : G[u]) {

int v = it.second; ll w = it.first;

if (dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

q.push({ dis[v], v });

}

}

}

}

int main()

{

cin >> n >> m >> s >> t;

for (int i = 1; i <= m; i++) {

ll b; int c;

sc("%lld %lld %d", &a[i], &b, &c);

int u = 0, v;

while (c--) {

sc("%d", &v);

st[v].insert(i);

if (u)

Add((i - 1) \* n + u, (i - 1) \* n + v, b);

u = v;

}

}

if (s == t)

puts("0"), exit(0);

Dijkstra();

ll ans = LINF;

for (int i = 1; i <= m; i++)

Min(ans, dis[(i - 1) \* n + t]);

if (ans == LINF)

puts("-1");

else

printf("%lld\n", ans);

return 0; // 改数组大小!!!用pair改宏定义!!!

}

## 图的连通性

### 模板

#### 1.无向图边双

vector <int> G[N];

int dfn[N], low[N], cnt, scnt;

int stk[N], s;

void Tarjan(int x, int fa) {

dfn[x] = low[x] = ++cnt;

stk[++s] = x;

for (auto v : G[x]) {

if (!dfn[v]) {

Tarjan(v, x);

Min(low[x], low[v]);

}

else if (v != fa)

Min(low[x], low[v]);

}

if (dfn[x] == low[x]) {

scnt++;

int k;

do {

k = stk[s--];

} while (k != x);

}

}

for (int i = 1; i <= n; i++) {

if (!dfn[i])

Tarjan(i, i);

}

#### 2.无向图点双

vector <int> G[N];

int dfn[N], low[N], idx, n;

int st[N], s;

void Tarjan(int x) {

dfn[x] = low[x] = ++idx;

st[++s] = x;

for (auto v : G[x]) {

if (!dfn[v]) {

Tarjan(v);

Min(low[x], low[v]);

if (low[v] >= dfn[x]) {

int k;

do {

k = st[s--];

} while (k != v);

}

}

else

Min(low[x], dfn[v]);

}

}

#### 3.无向图割点

vector <int> G[N];

int dfn[N], low[N], cnt, scnt;

bool cut[N];

void Tarjan(int x, int fa) {

dfn[x] = low[x] = ++cnt;

int son = 0;

for (auto v : G[x]) {

if (!dfn[v]) {

Tarjan(v, x);

Min(low[x], low[v]);

if (x != fa && low[v] >= dfn[x])

cut[x] = 1;

if (x == fa)

son++;

}

Min(low[x], dfn[v]);

}

if (x == fa && son >= 2)

cut[x] = 1;

}

for (int i = 1; i <= n; i++) {

if (!dfn[i])

Tarjan(i, i);

}

#### 4.无向图关键点

struct node

{

int u, v; ll w;

} a[N];

vector <pir> G[N << 1];

vector <int> e[N << 1];

int dfn[N], low[N], n, m, cnt, scnt;

ll d1[N], d2[N];

bool vis[N];

int cut[N];

void init() {

for (int i = 1; i <= n; i++)

G[i].clear(), e[i].clear();

MEM(d1, LINF), MEM(d2, LINF);

MEM(dfn, 0), MEM(low, 0);

MEM(cut, 0);

cnt = scnt = 0;

}

void dijkstra(int s, ll \*dis) { // 根据需要改pir的long long

priority\_queue <pir, vector <pir>, greater <pir>> q;

dis[s] = 0;

MEM(vis, 0);

q.push({ 0, s });

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (vis[u])

continue;

vis[u] = true;

for (auto it : G[u]) {

int v = it.second; ll w = it.first;

if (dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

q.push({ dis[v], v });

}

}

}

}

void tarjan(int x, int fa) {

dfn[x] = low[x] = ++cnt;

int son = 0;

for (auto v : e[x]) {

if (!dfn[v]) {

tarjan(v, x);

Min(low[x], low[v]);

if (x != fa && low[v] >= dfn[x])

cut[x] = 1;

if (x == fa)

son++;

}

Min(low[x], dfn[v]);

}

if (x == fa && son >= 2)

cut[x] = 1;

}

int main()

{

int T; cin >> T;

while (T--) {

sc("%d %d", &n, &m);

init();

for (int i = 1; i <= m; i++) {

int u, v; ll w;

sc("%d %d %lld", &u, &v, &w);

G[u].push\_back({ w, v });

G[v].push\_back({ w, u });

a[i] = { u, v, w };

}

dijkstra(1, d1), dijkstra(n, d2);

for (int i = 1; i <= m; i++) {

int u = a[i].u, v = a[i].v;

ll w = a[i].w;

if (d1[u] > d1[v])

swap(u, v);

if (d1[u] + d2[v] + w == d1[n]) {

e[u].push\_back(v), e[v].push\_back(u);

cut[u] = cut[v] = 2;

}

}

for (int i = 1; i <= n; i++) {

if (cut[i] && !dfn[i])

tarjan(i, i);

}

cut[1] = cut[n] = 1;

for (int i = 1; i <= n; i++)

printf("%d ", cut[i]);

printf("\n");

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 5.无向图可重割边

vector <pir> G[N];

int dfn[N], low[N], cnt;

void Tarjan(int x, int fa) { // fa是编号

dfn[x] = low[x] = ++cnt;

for (auto it : G[x]) {

int v = it.first, id = it.second;

if (id == fa)

continue;

if (!dfn[v]) {

Tarjan(v, id);

Min(low[x], low[v]);

if (low[v] > dfn[x])

vis[id] = true;

}

else

Min(low[x], dfn[v]);

}

}

#### 6.无向图输出最小割

// 思路：跑完最大流后从源点开始遍历，如果流量不为0，

继续DFS，最后遍历整个图，若点u标记了点v没标记，就输出

struct Edge

{

ll f;

int v, nxt;

}G[M << 1];

int n, m, sp, tp;

int h[N], cnt, L, R;

int dep[N], cur[N];

bool vis[N];

map <pir, bool> mp;

void init() {

mp.clear();

MEM(h, -1), MEM(vis, 0);

L = 1, R = n, cnt = 0;

}

void Add(int u, int v, ll f) {

G[cnt].v = v, G[cnt].f = f;

G[cnt].nxt = h[u], h[u] = cnt++;

G[cnt].v = u, G[cnt].f = f;

G[cnt].nxt = h[v], h[v] = cnt++;

}

bool BFS() {

queue <int> q;

for (int i = L; i <= R; i++)

dep[i] = 0;

q.push(sp), dep[sp] = 1;

while (!q.empty()) {

int u = q.front();

q.pop();

for (int i = h[u]; i != -1; i = G[i].nxt) {

int v = G[i].v;

if (!dep[v] && G[i].f) {

dep[v] = dep[u] + 1;

q.push(v);

}

}

}

return dep[tp];

}

ll DFS(int x, ll flow) {

if (x == tp || !flow)

return flow;

ll k, res = 0;

for (int &i = cur[x]; i != -1; i = G[i].nxt) {

int v = G[i].v;

if (dep[v] == dep[x] + 1 && G[i].f) {

k = DFS(v, min(flow - res, G[i].f));

if (k) {

G[i].f -= k;

G[i ^ 1].f += k;

res += k;

if (res == flow)

return flow;

}

}

}

if (!res)

dep[x] = 0;

return res;

}

ll Dinic() {

ll flow = 0;

while (BFS()) {

for (int i = L; i <= R; i++)

cur[i] = h[i];

flow += DFS(sp, LINF);

}

return flow;

}

void Path(int x) {

vis[x] = true;

for (int i = h[x]; i != -1; i = G[i].nxt) {

int v = G[i].v;

int w = G[i].f;

if (!vis[v] && w)

Path(v);

}

}

int main()

{

while (sc("%d %d", &n, &m) && n + m) {

sp = 1, tp = 2;

init();

for (int i = 0; i < m; i++) {

int u, v, w;

sc("%d %d %d", &u, &v, &w);

Add(u, v, w);

}

int ans = Dinic();

Path(sp);

for (int i = 1; i <= n; i++) {

for (int j = h[i]; j != -1; j = G[j].nxt) {

int v = G[j].v;

int w = G[j].f;

if (mp.count({ i, v }))

continue;

if (vis[i] && !vis[v] || !vis[i] && vis[v]) {

printf("%d %d\n", i, v);

mp[{i, v}] = mp[{v, i}] = true;

}

}

}

puts("");

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 7.无向图全局最小割

//使图不连通

int p[N][N], n, m;

int dis[N], v[N];

bool vis[N];

int SW() {

int ans = INF;

for (int i = 0; i < n; i++) v[i] = i;

while (n > 1) {

MEM(dis, 0); MEM(vis, 0);

int pre = 0, k; vis[v[pre]] = true;

for (int i = 1; i < n; i++) {

k = -1;

for (int j = 1; j < n; j++) {

if (!vis[v[j]]) {

dis[v[j]] += p[v[pre]][v[j]];

if (k == -1 || dis[v[j]] > dis[v[k]]) k = j;

}

}

vis[v[k]] = true;

if (i == n - 1) {

Min(ans, dis[v[k]]);

for (int j = 0; j < n; j++) {

p[v[pre]][v[j]] += p[v[j]][v[k]];

p[v[j]][v[pre]] += p[v[j]][v[k]];

}

v[k] = v[--n];

}

pre = k;

}

}

return ans;

}

int main()

{

while (~sc("%d %d", &n, &m)) {

MEM(p, 0);

for (int i = 0; i < m; i++) {

int ui, vi, wi;

sc("%d %d %d", &ui, &vi, &wi);

p[ui][vi] += wi;

p[vi][ui] += wi;

}

cout << SW() << endl;

}

return 0;

}

#### 8.无向图边双带重边

struct node

{

int v, f, nxt;

}G[N << 1];

int h[N], cnt, scnt;

int dfn[N], low[N];

int stk[N], n, s;

int idx;

void Init() {

for (int i = 1; i <= n; i++)

h[i] = -1;

idx = 0;

}

void Add(int u, int v) {

G[idx] = { v, 0, h[u] };

h[u] = idx++;

}

void Tarjan(int x) {

dfn[x] = low[x] = ++cnt;

stk[++s] = x;

for (int i = h[x]; ~i; i = G[i].nxt) {

if (G[i].f)

continue;

int v = G[i].v;

G[i].f = G[i ^ 1].f = true;

if (!dfn[v]) {

Tarjan(v);

Min(low[x], low[v]);

}

else

Min(low[x], dfn[v]);

}

if (dfn[x] == low[x]) {

scnt++;

int k;

do {

k = stk[s--];

} while (k != x);

}

}

#### 9.有向图缩点

vector <int> G[N];

int dfn[N], low[N], idx, scnt;

int stk[N], n, m, s;

bool vis[N];

void Tarjan(int x) {

dfn[x] = low[x] = ++idx;

stk[++s] = x;

vis[x] = true;

for (auto v : G[x]) {

if (!dfn[v]) {

Tarjan(v);

Min(low[x], low[v]);

}

else if (vis[v])

Min(low[x], low[v]);

}

if (dfn[x] == low[x]) {

scnt++;

int k;

do {

k = stk[s--];

vis[k] = false;

} while (k != x);

}

}

for (int i = 1; i <= n; i++) {

if (!dfn[i])

Tarjan(i);

}

#### 10.仙人掌图判断

vector <int> G[N];

int dfn[N], low[N], idx, n;

int st[N], s, scnt;

bool vis[N], anc[N];

bool ok;

void Init() {

for (int i = 1; i <= n; i++) {

G[i].clear();

dfn[i] = 0;

}

ok = true;

s = idx = scnt = 0;

}

void Tarjan(int x) {

dfn[x] = low[x] = ++idx;

vis[x] = anc[x] = true;

st[++s] = x;

int cnt = 0;

for (auto v : G[x]) {

if (!dfn[v]) {

Tarjan(v);

Min(low[x], low[v]);

}

else {

if (vis[v])

Min(low[x], dfn[v]);

if (dfn[x] < dfn[v]) // 前向边

ok = false;

else if (!anc[v]) // 横叉边

ok = false;

}

cnt += low[v] < dfn[x];

}

anc[x] = false;

if (cnt > 1)

ok = false;

if (dfn[x] == low[x]) {

scnt++;

int k;

do {

k = st[s--];

vis[k] = false;

} while (k != x);

}

}

int main()

{

int T; cin >> T;

while (T--) {

sc("%d", &n);

Init();

int u, v;

while (sc("%d %d", &u, &v) && u + v)

G[u + 1].push\_back(v + 1);

for (int i = 1; i <= n; i++) {

if (!dfn[i])

Tarjan(i);

}

if (scnt != 1)

ok = false;

puts(ok ? "YES" : "NO");

}

return 0; // 改数组大小!!!用pair改宏定义!!!

}

#### 11.有向图判环

bool dfs(int x) {

if (dfn[x] == 1)

return true; // 当前路径又走一次出环

if (dfn[x] == -1) // 已访问过的无环路径

return false;

dfn[x] = 1; // 当前路径

for (auto v : G[x])

if (dfs(v))

return true;

dfn[x] = -1; // 无环访问标记

return false;

}

#### 12.DAG图可达数

vector <int> G[N << 1];

bitset <N> bt[N];

bool vis[N];

void dfs(int x) {

vis[x] = true;

bt[x][x] = true;

for (auto v : G[x]) {

if (!vis[v])

dfs(v);

bt[x] |= bt[v];

}

}

#### 13.DAG图必经点

vector <int> G[N], ans;

int in[N], dp[N], n;

int dep[N];

bool vis[N];

void topsort() {

queue <int> q;

int now = 0;

for (int i = 1; i <= n; i++) {

dp[i] = 1;

if (!in[i])

q.push(i), now++, vis[i] = true;

}

while (!q.empty()) {

int u = q.front();

q.pop();

if (dp[u] == now && !vis[now])

ans.push\_back(now);

for (auto v : G[u]) {

in[v]--;

if (!in[v])

q.push(v), now++;

dp[v] += dp[u];

}

}

}

#### 14.DAG图最小表示法

vector <int> G[N];

bitset <N> bt[N];

int in[N], n, m;

int cnt[N];

pir s[N];

bool cmp(int a, int b) {

return cnt[a] < cnt[b];

}

void topsort() {

queue <int> q;

for (int i = 1; i <= n; i++) {

if (!in[i])

q.push(i);

}

int tot = 0;

while (!q.empty()) {

int u = q.front();

q.pop();

cnt[u] = ++tot;

s[u] = { tot, u };

for (auto v : G[u]) {

in[v]--;

if (!in[v])

q.push(v);

}

}

}

int main()

{

cin >> n >> m;

for (int i = 0; i < m; i++) {

int ui, vi;

sc("%d %d", &ui, &vi);

G[ui].push\_back(vi);

in[vi]++;

}

topsort();

sort(s + 1, s + n + 1, greater <pir>());

for (int i = 1; i <= n; i++)

sort(ALL(G[i]), cmp);

int ans = 0;

for (int i = 1; i <= n; i++) {

int u = s[i].second;

for (auto v : G[u]) {

if (!bt[u][v])

bt[u][v] = true, bt[u] |= bt[v];

else

ans++;

}

}

cout << ans << endl;

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 15.DAG图S到T路径数

ll dfs(int x) {

if (x == n)

return 1;

if (dp[x])

return dp[x];

ll tot = 0;

for (auto v : G[x])

tot = (tot + dfs(v)) % Mod;

dp[x] = tot;

return dp[x];

}

#### 16.动态图的连通性

// 时间分治线段树

struct node

{

int u, v, id;

};

vector <pir> ver[M \* 4];

vector <node> ask[M];

int fz[N], sz[N], ans[M], n, q, dfn;

map <pir, int> vis;

stack <pir> st;

void Init() {

for (int i = 1; i <= n; i++)

fz[i] = i, sz[i] = 1;

}

int Find(int x) {

while (x != fz[x])

x = fz[x];

return x;

}

void Merge(int x, int y) {

x = Find(x), y = Find(y);

if (x == y) {

st.push({ 0, 0 });

return;

}

if (sz[x] > sz[y])

swap(x, y);

fz[x] = y, sz[y] += sz[x];

st.push({ x, y });

}

void Cancle() {

int x = st.top().first, y = st.top().second;

st.pop();

fz[x] = x, sz[y] -= sz[x];

}

#define ls (o << 1)

#define rs (ls | 1)

void Update(int o, int L, int R, int l, int r, pir val) {

if (L >= l && R <= r)

ver[o].push\_back(val);

else {

int mid = (L + R) >> 1;

if (mid >= l)

Update(ls, L, mid, l, r, val);

if (mid < r)

Update(rs, mid + 1, R, l, r, val);

}

}

void Ask(int o, int L, int R) {

for (auto it : ver[o])

Merge(it.first, it.second);

if (L == R) {

for (auto it : ask[L]) { // 叶子结点询问

int u = it.u, v = it.v;

if (Find(u) == Find(v))

ans[it.id] = true;

}

}

else {

int mid = (L + R) >> 1;

Ask(ls, L, mid), Ask(rs, mid + 1, R);

}

int sz = SZ(ver[o]);

while (sz--)

Cancle();

}

int main()

{

#ifdef OlaMins

freopen("D:/input.txt", "r", stdin);

//freopen("D:/output.txt", "w", stdout);

#endif

cin >> n >> q;

Init();

for (int i = 1; i <= q; i++) {

int op, u, v;

sc("%d %d %d", &op, &u, &v);

if (u > v)

swap(u, v);

if (op == 0)

vis[{u, v}] = i;

else if (op == 1) {

int lst = vis[{u, v}];

vis.erase({ u, v }); // 删除上次边

Update(1, 1, q, lst, i - 1, { u, v });

}

else

ask[i].push\_back({ u, v, ++dfn });

}

for (auto it : vis) // 剩余没删的保持到q

Update(1, 1, q, it.second, q, { it.first.first, it.first.second });

Ask(1, 1, q);

for (int i = 1; i <= dfn; i++)

printf("%c\n", ans[i] ? 'Y' : 'N');

return 0; // 改数组大小!!!用pair改宏定义!!!

}

### 题集:

#### 1.矿场搭建

/\*用Tarjan跑出割点，然后DFS搜索所有的联通块,计算每一个联通快中的割点数目

如果没有割点,至少需要建立两个出口,从任意非割点的地方选择两个点建立

如果这个分组只有一个割点,只需要在分组内设立一个出口

可以设立在任意一个非割点的地方

如果有两个及以上个割点，则无需建立，可以直接到达其他联通块\*/

vector <int> G[N << 1];

int dfn[N], low[N], cnt, scnt;

bool cut[N], vis[N], vist[N];

int m, w;

set <int> st;

void init() {

for (int i = 1; i <= 1000; i++)

G[i].clear();

MEM(dfn, 0), MEM(low, 0);

MEM(cut, 0), MEM(vis, 0);

MEM(vist, 0);

cnt = scnt = 0;

}

void tarjan1(int x, int fa) {

dfn[x] = low[x] = ++cnt;

int son = 0;

for (auto v : G[x]) {

if (!dfn[v]) {

tarjan1(v, x);

Min(low[x], low[v]);

if (x != fa && low[v] >= dfn[x])

cut[x] = true;

if (x == fa)

son++;

}

Min(low[x], dfn[v]);

}

if (x == fa && son >= 2)

cut[x] = true;

}

void dfs(int x) {

w++;

vis[x] = true;

for (auto v : G[x]) {

if (cut[v]) {

st.insert(v);

continue;

}

if (!vis[v])

dfs(v);

}

}

int main()

{

int Case = 0;

while (sc("%d", &m) && m) {

init();

for (int i = 0; i < m; i++) {

int u, v;

sc("%d %d", &u, &v);

G[u].push\_back(v);

G[v].push\_back(u);

vist[u] = vist[v] = true;

}

for (int i = 1; i <= 1000; i++) {

if (!dfn[i])

tarjan1(i, i); // 割点

}

ll ans = 1;

int sum = 0;

for (int i = 1; i <= 1000; i++) {

if (cut[i] || vis[i] || !vist[i])

continue;

w = 0;

st.clear();

dfs(i);

int sz = SZ(st);

if (!sz)

sum += 2, ans \*= (ll)w \* (ll)(w - 1) / 2;

else if (sz == 1)

sum++, ans \*= (ll)w;

}

printf("Case %d: %d %lld\n", ++Case, sum, ans);

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 2.无向图变强连通

struct Edge {

int v, id, pre;

};

vector <Edge> G[N];

int dfn[N], low[N], cnt, scnt;

int stk[N], s;

bool ans[N];

void tarjan(int x, int fa) {

dfn[x] = low[x] = ++cnt;

stk[++s] = x;

for (auto it : G[x]) {

int v = it.v;

if (!dfn[v]) {

tarjan(v, x);

Min(low[x], low[v]);

}

else if (v != fa)

Min(low[x], low[v]);

}

if (dfn[x] == low[x]) {

scnt++;

int k;

do {

k = stk[s--];

} while (k != x);

}

}

void dfs(int x, int fa) {

dfn[x] = low[x] = ++cnt;

for (auto it : G[x]) {

int v = it.v, id = it.id;

int pre = it.pre;

if (!dfn[v]) {

ans[id] = pre;

dfs(v, x);

Min(low[x], low[v]);

}

else if (v != fa) {

if (low[v] < low[x])

ans[id] = pre, low[x] = low[v];

}

}

}

int main()

{

int n, m;

cin >> n >> m;

for (int i = 1; i <= m; i++) {

int u, v;

sc("%d %d", &u, &v);

G[u].push\_back({ v, i, 1 });

G[v].push\_back({ u, i, 0 });

}

for (int i = 1; i <= n; i++) {

if (!dfn[i])

tarjan(i, i);

}

if (scnt != 1)

printf("impossible\n"), exit(0);

MEM(dfn, 0); MEM(low, 0);

cnt = 0;

dfs(1, 1);

for (int i = 1; i <= m; i++)

printf("%d", ans[i]);

printf("\n");

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 3.构造方案使图强连通

vector <int> G[N];

int dfn[N], low[N], cnt, scnt;

int stk[N], s;

bool vis[N], e[N][N];

int n, m;

void init() {

for (int i = 1; i <= n; i++)

G[i].clear(), dfn[i] = low[i] = vis[i] = 0;

MEM(e, 0);

s = cnt = scnt = 0;

}

void dfs(int x, int fa) {

vis[x] = true;

for (int i = 0; i < SZ(G[x]); i++) {

int v = G[x][i];

if (e[x][v])

continue;

printf("%d %d\n", x, v);

e[x][v] = e[v][x] = true;

if (!vis[v])

dfs(v, x);

}

}

void tarjan(int x, int fa) {

dfn[x] = low[x] = ++cnt;

stk[++s] = x;

for (int i = 0; i < SZ(G[x]); i++) {

int v = G[x][i];

if (!dfn[v]) {

tarjan(v, x);

Min(low[x], low[v]);

}

else if (v != fa)

Min(low[x], low[v]);

}

if (dfn[x] == low[x]) {

scnt++;

int k;

do {

k = stk[s--];

} while (k != x);

}

}

int main()

{

int T; cin >> T;

while (T--) {

sc("%d %d", &n, &m);

init();

for (int i = 0; i < m; i++) {

int u, v;

sc("%d %d", &u, &v);

G[u].push\_back(v);

G[v].push\_back(u);

}

for (int i = 1; i <= n; i++) {

if (!dfn[i])

tarjan(i, i);

}

if (scnt != 1)

printf("NO\n");

else

printf("YES\n"), dfs(1, 0);

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 4.删一点求每点不可达数

void Tarjan(int x, int fa) {

sz[x] = 1;

dfn[x] = low[x] = ++idx;

ll tot = 0;

for (auto v : G[x]) {

if (!dfn[v]) {

Tarjan(v, x);

sz[x] += sz[v];

Min(low[x], low[v]);

if (low[v] >= dfn[x]) // 回不来包括一个边双连通分量

ans[x] += 1ll \* sz[v] \* (n - sz[v]), tot += sz[v];

}

else if (v != fa)

Min(low[x], dfn[v]);

}

ans[x] += n - 1 + 1ll \* (n - tot - 1) \* (tot + 1); // 剩余子树贡献

}

for (int i = 1; i <= n; i++) {

if (!dfn[i])

Tarjan(i, i);

}

#### 5.固定边方向使图无环方案数

struct node

{

int v, f, nxt;

}G[N << 1];

int h[N];

int dfn[N], low[N], cnt, scnt;

int stk[N], n, s;

ll cir, edge;

int idx;

void Init() {

for (int i = 1; i <= n; i++)

h[i] = -1;

idx = 0;

}

void Add(int u, int v) {

G[idx] = { v, 0, h[u] };

h[u] = idx++;

}

void Tarjan(int x) {

dfn[x] = low[x] = ++cnt;

stk[++s] = x;

for (int i = h[x]; ~i; i = G[i].nxt) {

if (G[i].f)

continue;

edge++;

int v = G[i].v;

G[i].f = G[i ^ 1].f = true;

if (!dfn[v]) {

Tarjan(v);

Min(low[x], low[v]);

}

else

Min(low[x], dfn[v]);

}

if (dfn[x] == low[x]) {

scnt++;

int k, num = 0;

do {

num++;

k = stk[s--];

} while (k != x);

Max(cir, (ll)num);

}

}

int main()

{

cin >> n;

Init();

for (int i = 1; i <= n; i++) {

int v;

sc("%d", &v);

Add(i, v), Add(v, i);

}

ll ans = 1;

for (int i = 1; i <= n; i++)

if (!dfn[i]) {

cir = edge = 0;

Tarjan(i);

ll tot = (dpow(2ll, edge) - dpow(2ll, edge - cir + 1) + Mod) % Mod;

ans = (ans \* tot) % Mod;

}

printf("%lld\n", ans);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 6.删除多余边仍然强连通

vector <pir> G[N];

pir eg[N];

int vis[N], dfn;

int n, m;

map <pir, int> mp;

void Init() {

for (int i = 1; i <= n; i++)

G[i].clear();

mp.clear();

}

void DFS(int x, int k) {

vis[x] = dfn;

for (auto it : G[x]) {

int v = it.first, id = it.second;

if (id % 2 != k || vis[v] == dfn)

continue;

if (!k)

mp[{x, v}]++;

else

mp[{v, x}]++;

DFS(v, k);

}

}

int main()

{

int T; cin >> T;

while (T--) {

sc("%d %d", &n, &m);

Init();

for (int i = 0; i < m; i++) {

int u, v;

sc("%d %d", &u, &v);

G[u].push\_back({ v, 2 \* i });

G[v].push\_back({ u, 2 \* i + 1 });

eg[i] = { u, v };

}

dfn++, DFS(1, 0);

dfn++, DFS(1, 1);

vector <pir> ver;

for (int i = 0; i < m; i++) {

int u = eg[i].first, v = eg[i].second;

if (!mp.count({ u, v }))

ver.push\_back(eg[i]);

}

for (int i = 0; i < m - 2 \* n; i++)

printf("%d %d\n", ver[i].first, ver[i].second);

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 7.定义边方向使得S到达点最多&最少

vector <pir> G[N];

int n, m, m1, s;

bool vis[N], eg[N \* 2];

int ans[2];

char ch[2][N \* 2];

bool cur[N \* 2];

void Init() {

MEM(vis, 0), MEM(eg, 0);

}

void DFS(int x, int op) {

vis[x] = true, ans[op]++;

for (auto it : G[x]) {

int v = it.first, id = it.second;

if (!cur[id]) { // 有向边

if (!vis[v])

DFS(v, op);

}

else {

if (eg[id])

continue;

eg[id] = eg[id ^ 1] = true;

if (id & 1)

ch[op][id / 2] = !op ? '-' : '+';

else

ch[op][id / 2] = !op ? '+' : '-';

if (!vis[v] && !op)

DFS(v, op);

}

}

}

int main()

{

cin >> n >> m >> s;

for (int i = 0; i < m; i++) {

int op, u, v;

sc("%d %d %d", &op, &u, &v);

G[u].push\_back({ v, i \* 2 });

if (op == 2) {

cur[i \* 2] = cur[i \* 2 + 1] = true;

G[v].push\_back({ u, i \* 2 + 1 });

ch[0][i] = ch[1][i] = '+';

}

}

DFS(s, 0), Init();

DFS(s, 1);

for (int i = 0; i < 2; i++) {

printf("%d\n", ans[i]);

for (int j = 0; j <= 2 \* m + 1; j++) {

if (ch[i][j])

printf("%c", ch[i][j]);

}

putchar('\n');

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 8.定义边方向变DAG图

vector <int> G[N];

int top[N];

int in[N], n, m;

pir eg[N];

void Init() {

for (int i = 1; i <= n; i++)

G[i].clear(), in[i] = 0;

}

bool Topsort() {

queue <int> q;

for (int i = 1; i <= n; i++)

if (!in[i])

q.push(i);

int cnt = 0;

while (!q.empty()) {

int u = q.front();

q.pop();

top[u] = ++cnt;

for (auto v : G[u])

if (!--in[v])

q.push(v);

}

return cnt == n;

}

int main()

{

int T; cin >> T;

while (T--) {

sc("%d %d", &n, &m);

Init();

for (int i = 1; i <= m; i++) {

int op, u, v;

sc("%d %d %d", &op, &u, &v);

if (op)

G[u].push\_back(v), in[v]++;

eg[i] = { u, v };

}

if (!Topsort())

puts("NO");

else {

puts("YES");

for (int i = 1; i <= m; i++) {

int u = eg[i].first;

int v = eg[i].second;

if (top[u] > top[v])

swap(u, v);

printf("%d %d\n", u, v);

}

}

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

## 二分图&匹配

### 模板

#### 1.二分图的性质

最小点覆盖：选取最少的点，使任意一条边至少有一个点被覆盖

最小点覆盖 = 最大匹配

最小边覆盖：选取最少的边，使任意一点被至少一条边覆盖

最小边覆盖 = 有效顶点数 - 最大匹配

最大独立集：选取最多的点，使任意所选两点均不相连

最大独立集 = 有效顶点数 - 最大匹配 （无向图的最大匹配要 / 2）

最大团：选取最多的点，使任意所选两点均有边直接连接

最大团 = 补图的最大独立集

最小路径覆盖：对于一个有向无环图，选取最少的路径，使得每个顶点属于且仅属于一条路径，路径长度可以为0即单个点，将原图节点拆分为两个点，一个接出边一个接入边。

最小路径覆盖 = 原图顶点数 - 最大匹配

可交最小路径覆盖，先进行传递闭包，再按照不可交路径覆盖建图跑最大匹配。其意义为，边u->x->v传递闭包后u->x->v还有u->v，在第一条路径经过x时第二条路径可以直接跳过，避免重复覆盖点x。

Hall定理： // 检测是否完备匹配

二部图G中的两部分顶点组成的集合分别为X,Y, X={X1,X2,X3,X4,.........,Xm},Y=\{y1, y2, y3, y4 ,.........,yn\},G中有一组无公共点的边，一端恰好为组成X的点的充分必要条件是：

X中的任意k个点至少与Y中的k个点相邻。（1≤k≤m)

二分图用网络流：sqrt(N) \* M

#### 2.匈牙利算法

// N \* M

vector <int> G[N];

int pre[N];

bool vis[N];

bool Find(int x) {

for (auto v : G[x]) {

if (!vis[v]) {

vis[v] = true;

if (!pre[v] || Find(pre[v])) {

pre[v] = x;

return true;

}

}

}

return false;

}

#### 3.二分图最大带权匹配

// N \* N \* N

ll lx[N], ly[N], visy[N], w[N][N];

ll slk[N], pre[N], link[N];

int n;

void bfs(int k) {

int x, y = 0, yy = 0, delta;

MEM(pre, 0);

for (int i = 1; i <= n; i++)

slk[i] = LINF;

link[y] = k;

while (true) {

x = link[y], delta = LINF, visy[y] = 1;

for (int i = 1; i <= n; i++) {

if (!visy[i]) {

if (slk[i] > lx[x] + ly[i] - w[x][i])

slk[i] = lx[x] + ly[i] - w[x][i], pre[i] = y;

if (slk[i] < delta)

delta = slk[i], yy = i;

}

}

for (int i = 0; i <= n; i++) {

if (visy[i])

lx[link[i]] -= delta, ly[i] += delta;

else

slk[i] -= delta;

}

y = yy;

if (link[y] == -1)

break;

}

while (y)

link[y] = link[pre[y]], y = pre[y];

}

int KM() {

MEM(link, -1);

for (int i = 1; i <= n; i++)

MEM(visy, 0), bfs(i);

int res = 0;

for (int i = 1; i <= n; i++)

if (link[i])

res += w[link[i]][i];

return res;

}

#### 4.无向图匹配带花树算法

// 无向图最大匹配 M \* N ^ 0.5

vector <int> G[N];

int pre[N], match[N], n, m;

int f[N], col[N], ID;

int id[N];

queue <int> q;

void init() {

ID = 0;

for (int i = 1; i <= n; i++)

id[i] = match[i] = 0;

}

int find\_(int x) {

if (x != f[x])

x = f[x] = find\_(f[x]);

return x;

}

int LCA(int x, int y) {

for (++ID;; swap(x, y)) {

if (x) {

x = find\_(x);

if (id[x] == ID)

return x;

id[x] = ID;

x = pre[match[x]];

}

}

}

void Calc(int x, int y, int t) {

while (find\_(x) != t) {

pre[x] = y;

y = match[x];

if (col[y] == 2)

col[y] = 1, q.push(y);

if (find\_(x) == x)

f[x] = t;

if (find\_(y) == y)

f[y] = t;

x = pre[y];

}

}

bool BFS(int x) {

for (int i = 1; i <= n; i++)

f[i] = i, pre[i] = col[i] = 0;

while (!q.empty())

q.pop();

q.push(x), col[x] = 1;

while (!q.empty()) {

int u = q.front();

q.pop();

for (auto v : G[u]) {

if (find\_(u) == find\_(v) || col[v] == 2)

continue;

if (!col[v]) {

col[v] = 2;

pre[v] = u;

if (!match[v]) {

for (int s = v, lst; s; s = lst) {

lst = match[pre[s]];

match[s] = pre[s];

match[pre[s]] = s;

}

return true;

}

col[match[v]] = 1;

q.push(match[v]);

}

else {

int lca = LCA(u, v);

Calc(u, v, lca);

Calc(v, u, lca);

}

}

}

return false;

}

for (int i = 1; i <= n; i++) {

if (!match[i] && BFS(i))

ans++;

}

#### 5.二分图可行边和必须边

必须边：

跑一遍最大匹配，遍历整张图，如果是匹配的边，反向，否则正向，tarjan缩点，再遍历一次，如果两个点不在一个强连通分量中并且该边是匹配边，那么该边是必须边

可行边：

如果两个点在一个强连通分量中且不是匹配边，那么该边是可行边

### 题集：

#### 1.陆地覆盖后最大独立集

int dir[4][2] = { 1, 0, -1, 0, 0, 1, 0, -1 };

vector <int> G[M << 1];

int pre[M], n, m;

char str[N][N];

bool vis[N][N], vist[M];

void add(int u, int v) {

G[u].push\_back(v);

G[v].push\_back(u);

}

int Calc(int x, int y) {

return (x - 1) \* m + y;

}

void dfs(int x, int y) {

vis[x][y] = true;

for (int i = 0; i < 4; i++) {

int xx = x + dir[i][0];

int yy = y + dir[i][1];

if (xx >= 1 && xx <= n && yy >= 1 && yy <= m && !vis[xx][yy]) {

if (str[xx][yy] == 'C')

str[xx][yy] = 'W';

else if (str[xx][yy] == 'L')

dfs(xx, yy);

}

}

}

bool find\_(int x) {

for (auto v : G[x]) {

if (!vist[v]) {

vist[v] = true;

if (!pre[v] || find\_(pre[v])) {

pre[v] = x;

return true;

}

}

}

return false;

}

int main()

{

cin >> n >> m;

for (int i = 1; i <= n; i++)

sc("%s", str[i] + 1);

int ans = 0;

// 将陆地周围的云变成海洋

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= m; j++) {

if (str[i][j] == 'L' && !vis[i][j])

ans++, dfs(i, j);

}

}

// 云层间建图

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= m; j++) {

if (str[i][j] != 'C')

continue;

int u = Calc(i, j);

if (j < m && str[i][j + 1] == 'C')

add(u, Calc(i, j + 1));

if (i < n && str[i + 1][j] == 'C')

add(u, Calc(i + 1, j));

}

}

int tot = 0, sum = 0; // 匹配数，总点数

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= m; j++) {

if (str[i][j] != 'C')

continue;

sum++;

MEM(vist, 0);

if (find\_(Calc(i, j)))

tot++;

}

}

printf("%d\n", ans + sum - tot / 2);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 2.三分图染色

vector <int> G[N];

int c[N];

int main()

{

int n, m;

cin >> n >> m;

for (int i = 1; i <= n; i++)

c[i] = 1;

for (int i = 0; i < m; i++) {

int u, v;

sc("%d %d", &u, &v);

if (u > v)

swap(u, v);

G[u].push\_back(v);

}

int mx = 1;

for (int i = 1; i <= n; i++) {

for (auto v : G[i]) {

if (c[i] == c[v])

c[v]++, Max(mx, c[v]);

}

}

int c1 = 0, c2 = 0, c3 = 0;

for (int i = 1; i <= n; i++) {

if (c[i] == 1)

c1++;

else if (c[i] == 2)

c2++;

else

c3++;

}

if (mx != 3 || c1 \* c2 + c2 \* c3 + c1 \* c3 != m)

printf("-1\n");

else {

for (int i = 1; i <= n; i++)

printf("%d ", c[i]);

printf("\n");

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 3.DAG图最小路径覆盖

定义 ：DAG图中， 找出最少的路径经过所有点

不相交 ：每一个点经过的顶点各不相同

N – 最大匹配数

相交 ： 每一个点经过的顶点可以相同

先用floyed求出传递闭包，然后转化成不相交最小路径覆盖

#### 4.注意可以横纵坐标当成二分图

## 网络流

### 1.Dinic弧优化

// N \* N \* M

// 二分图为M \* sqrt(N)

struct Edge

{

ll f;

int v, nxt;

}G[M];

int n, m, sp, tp;

int h[N], cnt, L, R;

int dep[N], cur[N];

void Init() {

for (int i = 1; i <= n; i++)

h[i] = -1;

L = 1, R = n, cnt = 0;

}

void Add(int u, int v, ll f) {

G[cnt] = { f, v, h[u] };

h[u] = cnt++;

G[cnt] = { 0, u, h[v] };

h[v] = cnt++;

}

bool BFS() {

queue <int> q;

for (int i = L; i <= R; i++)

dep[i] = 0;

q.push(sp), dep[sp] = 1;

while (!q.empty()) {

int u = q.front();

q.pop();

for (int i = h[u]; i != -1; i = G[i].nxt) {

int v = G[i].v;

if (!dep[v] && G[i].f) {

dep[v] = dep[u] + 1;

if (v == tp)

return true;

q.push(v);

}

}

}

return false;

}

ll DFS(int x, ll flow) {

if (x == tp || !flow)

return flow;

ll k, res = 0;

for (int &i = cur[x]; i != -1; i = G[i].nxt) {

int v = G[i].v;

if (dep[v] == dep[x] + 1 && G[i].f) {

k = DFS(v, min(flow - res, G[i].f));

if (k) {

G[i].f -= k;

G[i ^ 1].f += k;

res += k;

if (res == flow)

return flow;

}

}

}

if (!res)

dep[x] = 0;

return res;

}

ll Dinic() {

ll flow = 0;

while (BFS()) {

for (int i = L; i <= R; i++)

cur[i] = h[i];

flow += DFS(sp, LINF);

}

return flow;

}

### 2.Dij费用流

// vector版

struct Edge

{

ll f, w;

int v, r;

};

vector <Edge> G[N];

int n, m, sp, tp;

int pre[N], prv[N]; // 边，点

ll dis[N], h[N];

ll mincost, mxflow;

void Init() {

for (int i = sp; i <= tp; i++)

G[i].clear();

mincost = mxflow = 0;

}

void Add(int u, int v, ll f, ll w) {

G[u].push\_back({ f, w, v, SZ(G[v]) });

G[v].push\_back({ 0, -w, u, SZ(G[u]) - 1 });

}

bool Dijkstra() { //pir记得改long long

for (int i = sp; i <= tp; i++)

dis[i] = LINF;

dis[sp] = 0;

pre[tp] = prv[tp] = -1;

priority\_queue <pir, vector <pir>, greater <pir>> q;

q.push({ 0, sp });

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (dis[u] < now.first)

continue;

for (int i = 0; i < SZ(G[u]); i++) {

int v = G[u][i].v;

ll f = G[u][i].f, w = G[u][i].w;

if (f && dis[v] > dis[u] + w + h[u] - h[v]) {

dis[v] = dis[u] + w + h[u] - h[v];

q.push({ dis[v], v });

pre[v] = i;

prv[v] = u;

}

}

}

return pre[tp] != -1;

}

void MCFC() {

for (int i = sp; i <= tp; i++)

h[i] = 0;

while (Dijkstra()) {

for (int i = 1; i <= n; i++)

h[i] += dis[i];

ll mi = LINF;

int v = tp;

while (v != sp) {

int id = pre[v], u = prv[v];

Min(mi, G[u][id].f);

v = prv[v];

}

v = tp;

while (v != sp) {

int id = pre[v], u = prv[v];

int r = G[u][id].r;

G[u][id].f -= mi;

G[v][r].f += mi;

v = prv[v];

}

mxflow += mi;

mincost += mi \* h[tp];

}

}

// 链式前向星版

struct Edge

{

ll f, w;

int v, nxt;

}G[N];

int head[N], cnt;

int n, m, sp, tp;

int pre[N], prv[N]; // 边，点

ll dis[N], h[N];

ll mincost, mxflow;

void Init() {

for (int i = sp; i <= tp; i++)

head[i] = -1;

mincost = mxflow = 0;

}

void Add(int u, int v, ll f, ll w) {

G[cnt] = { f, w, v, head[u] };

head[u] = cnt++;

G[cnt] = { 0, -w, u, head[v] };

head[v] = cnt++;

}

bool Dijkstra() { //pir记得改long long

for (int i = sp; i <= tp; i++)

dis[i] = LINF;

dis[sp] = 0;

pre[tp] = prv[tp] = -1;

priority\_queue <pir, vector <pir>, greater <pir>> q;

q.push({ 0, sp });

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (dis[u] < now.first)

continue;

for (int i = head[u]; i != -1; i = G[i].nxt) {

int v = G[i].v;

ll f = G[i].f, w = G[i].w;

if (f && dis[v] > dis[u] + w + h[u] - h[v]) {

dis[v] = dis[u] + w + h[u] - h[v];

q.push({ dis[v], v });

pre[v] = i;

prv[v] = u;

}

}

}

return pre[tp] != -1;

}

void MCFC() {

for (int i = sp; i <= tp; i++)

h[i] = 0;

while (Dijkstra()) {

for (int i = sp; i <= tp; i++)

h[i] += dis[i];

ll mi = LINF;

int v = tp;

while (v != sp) {

int id = pre[v], u = prv[v];

Min(mi, G[id].f);

v = prv[v];

}

v = tp;

while (v != sp) {

int id = pre[v], u = prv[v];

G[id].f -= mi;

G[id ^ 1].f += mi;

v = prv[v];

}

mxflow += mi;

mincost += mi \* h[tp];

}

}

### 3.Spfa费用流

// vector版 N \* M \* f

struct Edge

{

ll f, w;

int v, r;

};

vector <Edge> G[N];

ll dis[N];

ll mxflow, mincost;

int n, m, sp, tp;

int prv[N], pre[N];

bool vis[N];

void Init() {

for (int i = sp; i <= tp; i++)

G[i].clear();

mxflow = mincost = 0;

}

void Add(int u, int v, ll f, ll w) {

G[u].push\_back({ f, w, v, SZ(G[v]) });

G[v].push\_back({ 0, -w, u, SZ(G[u]) - 1 });

}

bool Spfa() { // 找到一条最短路的增广

queue <int> q;

for (int i = sp; i <= tp; i++) {

dis[i] = LINF;

vis[i] = false;

}

q.push(sp);

vis[sp] = true;

dis[sp] = 0, prv[tp] = -1;

while (!q.empty()) {

int u = q.front();

q.pop();

vis[u] = false;

for (int i = 0; i < SZ(G[u]); i++) {

ll f = G[u][i].f, w = G[u][i].w;

int v = G[u][i].v;

if (f && dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

prv[v] = u;

pre[v] = i;

if (!vis[v]) {

vis[v] = true;

q.push(v);

}

}

}

}

return prv[tp] != -1;

}

void MCFC() { // 对于当前最短路更新流量

while (Spfa()) {

ll mi = LINF;

int v = tp;

while (v != sp) {

int id = pre[v], u = prv[v];

Min(mi, G[u][id].f);

v = prv[v];

}

v = tp;

while (v != sp) {

int id = pre[v], u = prv[v];

int r = G[u][id].r;

G[u][id].f -= mi;

G[v][r].f += mi;

v = prv[v];

}

mxflow += mi;

mincost += mi \* dis[tp];

}

}

// 链式前向星

struct Edge

{

ll f, w;

int v, nxt;

}G[N];

ll dis[N];

ll mxflow, mincost;

int n, m, sp, tp;

int h[N], cnt;

int prv[N], pre[N];

bool vis[N];

void Init() {

for (int i = sp; i <= tp; i++)

h[i] = -1;

cnt = 0;

mxflow = mincost = 0;

}

void Add(int u, int v, ll f, ll w) {

G[cnt] = { f, w, v, h[u] };

h[u] = cnt++;

G[cnt] = { 0, -w, u, h[v] };

h[v] = cnt++;

}

bool Spfa() { // 找到一条最短路的增广

queue <int> q;

for (int i = sp; i <= tp; i++) {

dis[i] = LINF;

vis[i] = false;

}

q.push(sp);

vis[sp] = true;

dis[sp] = 0, prv[tp] = -1;

while (!q.empty()) {

int u = q.front();

q.pop();

vis[u] = false;

for (int i = h[u]; i != -1; i = G[i].nxt) {

ll f = G[i].f, w = G[i].w;

int v = G[i].v;

if (f && dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

prv[v] = u;

pre[v] = i;

if (!vis[v]) {

vis[v] = true;

q.push(v);

}

}

}

}

return prv[tp] != -1;

}

void MCFC() { // 对于当前最短路更新流量

while (Spfa()) {

ll mi = LINF;

int v = tp;

while (v != sp) {

int id = pre[v], u = prv[v];

Min(mi, G[id].f);

v = prv[v];

}

v = tp;

while (v != sp) {

int id = pre[v], u = prv[v];

G[id].f -= mi;

G[id ^ 1].f += mi;

v = prv[v];

}

mxflow += mi;

mincost += mi \* dis[tp];

}

}

最小割可行边&必须边  
// 先跑一遍最大流，如果正向流量为0，该边一定是可形边

// 如果一条边在可行路径上是唯一的边，那就是必须边

// 实现方法：对残余网络跑Tarjan，遍历所有正向边，若流量为0

// 并且两个端点不在一个集合，为可行边，若两个端点分别与起点，终点

// 处于一个集合，那就是必须边

struct Edge

{

ll f;

int id, v, nxt;

}G[M];

int n, m, sp, tp;

int h[N], cnt, L, R;

int dep[N], cur[N];

int dfn[N], low[N], idx;

int st[N], suo[N], s, scnt;

bool vis[N], c1[M], c2[M];

void init() {

for (int i = 1; i <= n; i++)

h[i] = -1;

L = 1, R = n, cnt = 0;

}

void Add(int u, int v, int id, ll f) {

G[cnt] = { f, id, v, h[u] };

h[u] = cnt++;

G[cnt] = { 0, id, u, h[v] };

h[v] = cnt++;

}

bool BFS() {

queue <int> q;

for (int i = L; i <= R; i++)

dep[i] = 0;

q.push(sp), dep[sp] = 1;

while (!q.empty()) {

int u = q.front();

q.pop();

for (int i = h[u]; i != -1; i = G[i].nxt) {

int v = G[i].v;

if (!dep[v] && G[i].f) {

dep[v] = dep[u] + 1;

if (v == tp)

return true;

q.push(v);

}

}

}

return false;

}

ll DFS(int x, ll flow) {

if (x == tp || !flow)

return flow;

ll k, res = 0;

for (int &i = cur[x]; i != -1; i = G[i].nxt) {

int v = G[i].v;

if (dep[v] == dep[x] + 1 && G[i].f) {

k = DFS(v, min(flow - res, G[i].f));

if (k) {

G[i].f -= k;

G[i ^ 1].f += k;

res += k;

if (res == flow)

return flow;

}

}

}

if (!res)

dep[x] = 0;

return res;

}

ll Dinic() {

ll flow = 0;

while (BFS()) {

for (int i = L; i <= R; i++)

cur[i] = h[i];

flow += DFS(sp, LINF);

}

return flow;

}

void Tarjan(int x) {

dfn[x] = low[x] = ++idx;

st[++s] = x;

vis[x] = true;

for (int i = h[x]; i != -1; i = G[i].nxt) {

int v = G[i].v;

int f = G[i].f;

if (!f)

continue;

if (!dfn[v]) {

Tarjan(v);

Min(low[x], low[v]);

}

else if (vis[v])

Min(low[x], low[v]);

}

if (dfn[x] == low[x]) {

scnt++;

int k;

do {

k = st[s--];

vis[k] = false;

suo[k] = scnt;

} while (k != x);

}

}

int main()

{

cin >> n >> m >> sp >> tp;

init();

for (int i = 1; i <= m; i++) {

int u, v, w;

sc("%d %d %d", &u, &v, &w);

Add(u, v, i, w);

}

int ans = Dinic();

for (int i = 1; i <= n; i++) {

if (!dfn[i])

Tarjan(i);

}

for (int i = 1; i <= n; i++) {

for (int j = h[i]; j != -1; j = G[j].nxt) {

int v = G[j].v;

int id = G[j].id;

if (G[j].f || (j & 1))

continue;

if (suo[i] != suo[v])

c1[id] = true;

if (suo[i] == suo[sp] && suo[v] == suo[tp])

c2[id] = true;

}

}

for (int i = 1; i <= m; i++)

printf("%d %d\n", c1[i], c2[i]);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

### 4.最小割可行边&必须边

// 先跑一遍最大流，如果正向流量为0，该边一定是可形边

// 如果一条边在可行路径上是唯一的边，那就是必须边

// 实现方法：对残余网络跑Tarjan，遍历所有正向边，若流量为0

// 并且两个端点不在一个集合，为可行边，若两个端点分别与起点，终点

// 处于一个集合，那就是必须边

struct Edge

{

ll f;

int id, v, nxt;

}G[M];

int n, m, sp, tp;

int h[N], cnt, L, R;

int dep[N], cur[N];

int dfn[N], low[N], idx;

int st[N], suo[N], s, scnt;

bool vis[N], c1[M], c2[M];

void init() {

for (int i = 1; i <= n; i++)

h[i] = -1;

L = 1, R = n, cnt = 0;

}

void Add(int u, int v, int id, ll f) {

G[cnt] = { f, id, v, h[u] };

h[u] = cnt++;

G[cnt] = { 0, id, u, h[v] };

h[v] = cnt++;

}

bool BFS() {

queue <int> q;

for (int i = L; i <= R; i++)

dep[i] = 0;

q.push(sp), dep[sp] = 1;

while (!q.empty()) {

int u = q.front();

q.pop();

for (int i = h[u]; i != -1; i = G[i].nxt) {

int v = G[i].v;

if (!dep[v] && G[i].f) {

dep[v] = dep[u] + 1;

if (v == tp)

return true;

q.push(v);

}

}

}

return false;

}

ll DFS(int x, ll flow) {

if (x == tp || !flow)

return flow;

ll k, res = 0;

for (int &i = cur[x]; i != -1; i = G[i].nxt) {

int v = G[i].v;

if (dep[v] == dep[x] + 1 && G[i].f) {

k = DFS(v, min(flow - res, G[i].f));

if (k) {

G[i].f -= k;

G[i ^ 1].f += k;

res += k;

if (res == flow)

return flow;

}

}

}

if (!res)

dep[x] = 0;

return res;

}

ll Dinic() {

ll flow = 0;

while (BFS()) {

for (int i = L; i <= R; i++)

cur[i] = h[i];

flow += DFS(sp, LINF);

}

return flow;

}

void Tarjan(int x) {

dfn[x] = low[x] = ++idx;

st[++s] = x;

vis[x] = true;

for (int i = h[x]; i != -1; i = G[i].nxt) {

int v = G[i].v;

int f = G[i].f;

if (!f)

continue;

if (!dfn[v]) {

Tarjan(v);

Min(low[x], low[v]);

}

else if (vis[v])

Min(low[x], low[v]);

}

if (dfn[x] == low[x]) {

scnt++;

int k;

do {

k = st[s--];

vis[k] = false;

suo[k] = scnt;

} while (k != x);

}

}

int main()

{

cin >> n >> m >> sp >> tp;

init();

for (int i = 1; i <= m; i++) {

int u, v, w;

sc("%d %d %d", &u, &v, &w);

Add(u, v, i, w);

}

int ans = Dinic();

for (int i = 1; i <= n; i++) {

if (!dfn[i])

Tarjan(i);

}

for (int i = 1; i <= n; i++) {

for (int j = h[i]; j != -1; j = G[j].nxt) {

int v = G[j].v;

int id = G[j].id;

if (G[j].f || (j & 1))

continue;

if (suo[i] != suo[v])

c1[id] = true;

if (suo[i] == suo[sp] && suo[v] == suo[tp])

c2[id] = true;

}

}

for (int i = 1; i <= m; i++)

printf("%d %d\n", c1[i], c2[i]);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

### 5.网格图对偶图建图

void Add(int u, int v, ll w) {

G[u].push\_back({ w, v });

G[v].push\_back({ w, u });

}

int ID(int x, int y) {

return (x - 2) \* (n - 1) + y - 1;

}

void Add\_Row(int x, int y, ll w) {

if (x == n)

return;

if (y == n)

Add(ID(x + 1, y), T, w);

else if (y == 1)

Add(S, ID(x + 1, y + 1), w);

else

Add(ID(x + 1, y), ID(x + 1, y + 1), w);

}

void Add\_Col(int x, int y, ll w) {

if (y == 1)

return;

if (x == 1)

Add(S, ID(x + 1, y), w);

else if (x == n)

Add(ID(x, y), T, w);

else

Add(ID(x, y), ID(x + 1, y), w);

}

### 6.输出最小路径覆盖

struct Edge

{

ll f;

int v, nxt;

}G[M];

int n, m, sp, tp;

int h[N], cnt, L, R;

int dep[N], cur[N];

int fz[N];

int find\_(int x) {

if (x != fz[x])

x = fz[x] = find\_(fz[x]);

return x;

}

void init() {

sp = 0, tp = 2 \* n + 1;

for (int i = sp; i <= tp; i++)

h[i] = -1;

L = sp, R = tp, cnt = 0;

}

void Add(int u, int v, ll f) {

G[cnt] = { f, v, h[u] };

h[u] = cnt++;

G[cnt] = { 0, u, h[v] };

h[v] = cnt++;

}

bool BFS() {

queue <int> q;

for (int i = L; i <= R; i++)

dep[i] = 0;

q.push(sp), dep[sp] = 1;

while (!q.empty()) {

int u = q.front();

q.pop();

for (int i = h[u]; i != -1; i = G[i].nxt) {

int v = G[i].v;

if (!dep[v] && G[i].f) {

dep[v] = dep[u] + 1;

if (v == tp)

return true;

q.push(v);

}

}

}

return false;

}

ll DFS(int x, ll flow) {

if (x == tp || !flow)

return flow;

ll k, res = 0;

for (int &i = cur[x]; i != -1; i = G[i].nxt) {

int v = G[i].v;

if (dep[v] == dep[x] + 1 && G[i].f) {

k = DFS(v, min(flow - res, G[i].f));

if (k) {

G[i].f -= k;

G[i ^ 1].f += k;

res += k;

if (res == flow)

return flow;

}

}

}

if (!res)

dep[x] = 0;

return res;

}

void Output(int x) {

printf("%d ", x);

for (int i = h[x]; i != -1; i = G[i].nxt) {

int v = G[i].v;

int w = G[i].f;

if (!w && v > n)

Output(v - n);

}

}

ll Dinic() {

ll flow = 0;

while (BFS()) {

for (int i = L; i <= R; i++)

cur[i] = h[i];

flow += DFS(sp, LINF);

}

for (int i = 1; i <= n; i++)

fz[i] = i;

for (int i = 1; i <= n; i++) {

for (int j = h[i]; j != -1; j = G[j].nxt) {

int v = G[j].v, w = G[j].f;

if (!w && v > n && v < tp)

fz[find\_(v - n)] = find\_(i);

}

}

for (int i = 1; i <= n; i++) {

if (find\_(i) == i)

Output(i), puts("");

}

return flow;

}

int main()

{

cin >> n >> m;

init();

// 源点->入，出->汇点

for (int i = 1; i <= n; i++)

Add(sp, i, 1), Add(n + i, tp, 1);

for (int i = 0; i < m; i++) {

int u, v;

sc("%d %d", &u, &v);

Add(u, n + v, 1);

}

printf("%d\n", n - Dinic());

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

### 7.最大权闭合子图输出方案

// 最大权闭合子图：就是一幅图中每个点，以及每个点的出边的点都在这幅图中，也就是这幅图中的所有点的出边都是指向子图内部的。

// 求法：S->正权点，负权点->汇点，其余都是INF，答案就是正权和 – 最小割

struct Edge

{

ll f;

int v, nxt;

}G[M];

int n, m, sp, tp;

int h[N], cnt, L, R;

int dep[N], cur[N];

ll a[N], b[N];

void Init() {

sp = 0, tp = n + m + 1;

for (int i = sp; i <= tp; i++)

h[i] = -1;

L = sp, R = tp, cnt = 0;

}

void Add(int u, int v, ll f) {

G[cnt] = { f, v, h[u] };

h[u] = cnt++;

G[cnt] = { 0, u, h[v] };

h[v] = cnt++;

}

bool BFS() {

queue <int> q;

for (int i = L; i <= R; i++)

dep[i] = 0;

q.push(sp), dep[sp] = 1;

while (!q.empty()) {

int u = q.front();

q.pop();

for (int i = h[u]; i != -1; i = G[i].nxt) {

int v = G[i].v;

if (!dep[v] && G[i].f) {

dep[v] = dep[u] + 1;

if (v == tp)

return true;

q.push(v);

}

}

}

return false;

}

ll DFS(int x, ll flow) {

if (x == tp || !flow)

return flow;

ll k, res = 0;

for (int &i = cur[x]; i != -1; i = G[i].nxt) {

int v = G[i].v;

if (dep[v] == dep[x] + 1 && G[i].f) {

k = DFS(v, min(flow - res, G[i].f));

if (k) {

G[i].f -= k;

G[i ^ 1].f += k;

res += k;

if (res == flow)

return flow;

}

}

}

if (!res)

dep[x] = 0;

return res;

}

ll Dinic() {

ll flow = 0;

while (BFS()) {

for (int i = L; i <= R; i++)

cur[i] = h[i];

flow += DFS(sp, LINF);

}

return flow;

}

int main()

{

cin >> n >> m;

Init();

ll ans = 0;

for (int i = 1; i <= n; i++) {

sc("%lld", &a[i]);

ans += a[i];

Add(sp, i, a[i]);

char tools[10000];

memset(tools, 0, sizeof tools);

cin.getline(tools, 10000);

int ulen = 0, tool;

while (sscanf(tools + ulen, "%d", &tool) == 1)//之前已经用scanf读完了赞助商同意支付该实验的费用

{//tool是该实验所需仪器的其中一个

//这一行，你可以将读进来的编号进行储存、处理，如连边。

Add(i, n + tool, LINF);

if (tool == 0)

ulen++;

else {

while (tool) {

tool /= 10;

ulen++;

}

}

ulen++;

}

}

for (int i = 1; i <= m; i++) {

sc("%lld", &b[i + n]);

Add(n + i, tp, b[i + n]);

}

ans -= Dinic();

for (int i = 1; i <= n; i++) {

if (dep[i])

printf("%d ", i);

}

puts("");

for (int i = n + 1; i <= n + m; i++) {

if (dep[i])

printf("%d ", i - n);

}

printf("\n%lld\n", ans);

return 0; // 改数组大小!!!用pair改宏定义!!!

}

## 欧拉路

### 1.欧拉图判断

无向图

欧拉通路：联通图且奇度点为0或2

欧拉回路：联通图且所有点都为偶数度

有向图：

欧拉通路：基图联通且所有顶点出、入度相等或者有两个点入度出度差为1

欧拉回路：基图联通且所有点入度=出度

### 2.欧拉路径输出

vector <int> G[N];

vector <int> vec;

int dep[N], ev[N];

pir eg[N];

bool vis[N];

void DFS(int x) {

vis[x] = true;

printf("%d ", ev[x]);

for (auto v : G[x]) {

if (!vis[v])

DFS(v);

}

}

int main()

{

int n;

cin >> n;

for (int i = 1; i <= n; i++) {

int u, v;

sc("%d %d", &u, &v);

vec.push\_back(u), vec.push\_back(v);

eg[i] = { u, v };

}

sort(ALL(vec));

vec.erase(unique(ALL(vec)), vec.end());

for (int i = 1; i <= n; i++) {

int u = eg[i].first;

int v = eg[i].second;

int uu = lower\_bound(ALL(vec), u) - vec.begin() + 1;

int vv = lower\_bound(ALL(vec), v) - vec.begin() + 1;

ev[uu] = u, ev[vv] = v;

G[uu].push\_back(vv);

G[vv].push\_back(uu);

dep[uu]++, dep[vv]++;

}

int sz = SZ(vec), rt;

for (int i = 1; i <= sz; i++) {

if (dep[i] & 1) {

rt = i;

break;

}

}

DFS(rt);

putchar('\n');

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

### 3.多条欧拉路覆盖无向图方案

struct Edge

{

int v, nxt;

}G[M];

vector <int> vec[N], ans[N];

int h[N], cnt, idx;

int dep[N], n, m;

bool ver[N], edge[M];

void Init() {

for (int i = 0; i <= n; i++)

vec[i].clear(), ans[i].clear();

MEM(h, -1), MEM(dep, 0);

MEM(ver, 0), MEM(edge, 0);

cnt = idx = 0;

}

void Add(int u, int v) {

G[cnt] = { v, h[u] };

h[u] = cnt++;

}

void DFS(int x) {

ver[x] = true;

for (int i = h[x]; i != -1; i = G[i].nxt) {

int v = G[i].v;

if (edge[i])

continue;

edge[i] = edge[i ^ 1] = true;

DFS(v);

if (i >= 2 \* m)

idx++;

else if (i != -1)

vec[idx].push\_back(i);

}

}

int main()

{

while (~sc("%d %d", &n, &m)) {

Init();

for (int i = 0; i < m; i++) {

int u, v;

sc("%d %d", &u, &v);

Add(u, v), Add(v, u);

dep[u]++, dep[v]++;

}

int lst = -1;

for (int i = 1; i <= n; i++) {

if (!(dep[i] & 1))

continue;

if (lst == -1)

lst = i;

else {

Add(lst, i);

Add(i, lst);

lst = -1;

}

}

for (int i = 1; i <= n; i++) {

if (!ver[i] && (dep[i] & 1))

idx++, DFS(i);

}

for (int i = 1; i <= n; i++) {

if (!ver[i])

idx++, DFS(i);

}

int scnt = 0;

for (int i = 1; i <= idx; i++) {

if (vec[i].empty())

continue;

++scnt;

for (int j = SZ(vec[i]) - 1; j >= 0; j--)

ans[scnt].push\_back(vec[i][j]);

}

printf("%d\n", scnt);

for (int i = 1; i <= scnt; i++) {

printf("%d ", SZ(ans[i]));

for (auto it : ans[i]) {

if (it & 1)

it ^= 1, printf("-%d ", it / 2 + 1);

else

printf("%d ", it / 2 + 1);

}

puts("");

}

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

## 拓扑排序

### 模板:

#### 1.拓扑排序

vector <int> G[N];

int in[N], n;

int Topsort() {

queue <int> q;

for (int i = 1; i <= n; i++) {

if (!in[i])

q.push(i);

}

while (!q.empty()) {

int u = q.front();

q.pop();

for (auto v : G[u]) {

in[v]--;

if (!in[v])

q.push(v);

}

}

}

### 题集：

#### 1.相对位置建边字典序最小

unordered\_map <string, int> mp;

unordered\_map <int, string> ev;

int pre[N], s, l, n;

int in[N], idx[N];

vector <int> G[N];

bool e[M][M];

string str[M];

struct node

{

int id;

bool operator < (const node &oth) const {

return idx[id] > idx[oth.id];

}

};

void topsort() {

priority\_queue <node> q;

for (int i = 1; i <= n; i++) {

if (!in[i])

q.push({ i });

}

while (!q.empty()) {

node u = q.top();

q.pop();

cout << ev[u.id] << " ";

for (auto v : G[u.id]) {

in[v]--;

if (!in[v])

q.push({ v });

}

}

cout << endl;

}

int main()

{

cin >> s >> l >> n;

for (int i = 1; i <= s; i++)

cin >> str[i];

sort(str + 1, str + s + 1);

for (int i = 1; i <= s; i++)

mp[str[i]] = i;

for (int i = 0; i < l; i++) {

string u, v;

cin >> u >> v;

int uu = mp[u], vv = mp[v];

e[uu][vv] = e[vv][uu] = true;

}

for (int i = 1; i <= n; i++) {

string ch;

cin >> ch;

int u = mp[ch];

idx[i] = u;

ev[i] = ch;

for (int j = 1; j <= s; j++) {

if (e[j][u] || !pre[j])

continue;

G[pre[j]].push\_back(i);

in[i]++;

}

pre[u] = i;

}

topsort();

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 2.两条路径最大路径交

struct node

{

int u, v, w;

}a[N];

vector <pir> G[N << 1], e[N << 1];

int n, m, s1, t1, s2, t2;

int d1[N], d2[N], d3[N], d4[N];

int dp[N], ans;

int in[N];

bool vis[N];

void init() {

MEM(d1, INF); MEM(d2, INF); MEM(d3, INF); MEM(d4, INF);

}

void dijkstra(int x, int \*dis) {

priority\_queue <pir, vector <pir>, greater <pir>> q;

MEM(vis, 0);

dis[x] = 0, q.push({ 0, x });

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (vis[u])

continue;

vis[u] = true;

for (auto it : G[u]) {

int v = it.second, w = it.first;

if (dis[v] > dis[u] + w) {

dis[v] = dis[u] + w;

q.push({ dis[v], v });

}

}

}

}

void topsort() {

queue <int> q;

for (int i = 1; i <= n; i++) {

if (!in[i])

q.push(i);

}

while (!q.empty()) {

int u = q.front();

q.pop();

Max(ans, dp[u]);

for (auto it : e[u]) {

int v = it.second, w = it.first;

in[v]--;

Max(dp[v], dp[u] + w);

if (!in[v])

q.push(v);

}

}

}

int main()

{

cin >> n >> m;

cin >> s1 >> t1 >> s2 >> t2;

init();

for (int i = 1; i <= m; i++) {

int ui, vi, wi;

sc("%d %d %d", &ui, &vi, &wi);

G[ui].push\_back({ wi, vi });

G[vi].push\_back({ wi, ui });

a[i] = { ui, vi, wi };

}

dijkstra(s1, d1), dijkstra(t1, d2);

dijkstra(s2, d3), dijkstra(t2, d4);

for (int i = 1; i <= m; i++) {

int u = a[i].u, v = a[i].v, w = a[i].w;

if (d1[v] < d1[u])

swap(u, v);

if (d1[u] + d2[v] + w != d1[t1])

continue;

if (d3[v] < d3[u])

swap(u, v);

if (d3[u] + d4[v] + w == d3[t2])

e[u].push\_back({ w, v }), in[v]++;

}

topsort();

printf("%d\n", ans);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

## 生成树

### 模板：

#### 1.Kruscal

// M \* log(M)

struct node

{

int u, v, w;

bool operator < (const node &oth) const {

return w < oth.w;

}

}p[M];

int f[N], n, m, edge;

int ans;

void init() {

for (int i = 1; i <= n; i++)

f[i] = i;

}

int find\_(int x) {

if (x != f[x])

x = f[x] = find\_(f[x]);

return x;

}

void unite(int x, int y) {

x = find\_(x);

y = find\_(y);

if (x != y)

f[x] = y;

}

int main()

{

cin >> n >> m;

init();

for (int i = 1; i <= m; i++)

sc("%d %d %d", &p[i].u, &p[i].v, &p[i].w);

sort(p + 1, p + m + 1);

for (int i = 1; i <= m; i++) {

int u = p[i].u, v = p[i].v, w = p[i].w;

if (edge == n - 1)

break;

if (find\_(u) != find\_(v)) {

unite(u, v);

ans += w;

edge++;

}

}

cout << ans << endl;

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 2.Prime生成树

// N \* N

int n, m;

int dis[N], d[N][N];

bool vis[N];

ll Prim() {

MEM(dis, INF);

for (int i = 1; i <= n; i++)

dis[i] = d[1][i];

vis[1] = true;

int tot = 0;

int cnt = 0;

for (int i = 1; i < n; i++) {

cnt++;

int mi = INF;

int u = -1;

for (int j = 1; j <= n; j++)

if (!vis[j] && dis[j] < mi)

mi = dis[j], u = j;

vis[u] = true;

tot += mi;

if (u != -1) {

for (int v = 1; v <= n; v++)

if (!vis[v] && dis[v] > d[u][v])

dis[v] = d[u][v];

}

else

break;

}

if (cnt != n - 1)

return -1;

return tot;

}

#### 3.异或MST

// 已知点权

int a[N], n;

int Find(int p, int \*A, int m, int \*B, int k) {

if (p == 0 || m == 0)

return 1 << k;

if (k == 0)

return 0;

int x = lower\_bound(A, A + p, ((A[0] >> k - 1) | 1) << k - 1) - A;

int y = lower\_bound(B, B + m, ((B[0] >> k - 1) | 1) << k - 1) - B;

if (x == 0 && y == m || x == p && y == 0)

return Find(p, A, m, B, k - 1) + (1 << (k - 1));

return min(Find(x, A, y, B, k - 1), Find(p - x, A + x, m - y, B + y, k - 1));

}

ll XOR\_MST(int p, int \*A, int k) {

if (p == 1)

return 0;

int x = lower\_bound(A, A + p, ((A[0] >> (k - 1)) | 1) << (k - 1)) - A;

if (x == 0 || x == p)

return XOR\_MST(p, A, k - 1);

return (1ll << (k - 1)) + Find(x, A, p - x, A + x, k - 1) + XOR\_MST(x, A, k - 1) + XOR\_MST(p - x, A + x, k - 1);

}

sort(a, a + n);

n = unique(a, a + n) - a;

printf("%lld\n", XOR\_MST(n, a, 30));

#### 4.斯坦纳树

//最小斯坦纳树，就是要花费最小的代价，连通给定的 kk 个关键点，这是一个组合优化问题。

//O(n \* (3 ^ k) + m \* log(m) \* (2 ^ k))

priority\_queue <pir> q;

vector <pir> G[N];

int dp[N][2010], a[N]; // dp2为2^k

int n, m, k;

bool vis[N];

void Init() {

MEM(dp, INF);

}

void Dijkstra(int s) {

MEM(vis, 0);

while (!q.empty()) {

pir now = q.top();

q.pop();

int u = now.second;

if (vis[u])

continue;

vis[u] = true;

for (auto it : G[u]) {

int v = it.second, w = it.first;

if (dp[v][s] > dp[u][s] + w) {

dp[v][s] = dp[u][s] + w;

q.push({ -dp[v][s], v });

}

}

}

}

int main()

{

cin >> n >> m >> k;

Init();

for (int i = 0; i < m; i++) {

int u, v, w;

sc("%d %d %d", &u, &v, &w);

G[u].push\_back({ w, v });

G[v].push\_back({ w, u });

}

for (int i = 1; i <= k; i++) {

sc("%d", &a[i]);

dp[a[i]][1 << (i - 1)] = 0;

}

for (int s = 1; s < (1 << k); s++) {

for (int i = 1; i <= n; i++) {

for (int j = s & (s - 1); j; j = s & (j - 1))

Min(dp[i][s], dp[i][j] + dp[i][s ^ j]);

if (dp[i][s] != INF)

q.push({ -dp[i][s], i });

}

Dijkstra(s);

}

printf("%d\n", dp[a[1]][(1 << k) - 1]);

return 0; // 改数组大小!!!用pair改宏定义!!!

}

#### 5.有根最小树形图

// N \* N

struct Edge

{

int u, v, w;

}e[M];

int pre[N], id[N], vist[N];

int in[N];

int dmst(int rt, int n, int m) { // 根,点,边, 点下标从0开始

int u, v;

int res = 0;

while (true) {

for (int i = 0; i < n; i++)

in[i] = INF;

for (int i = 0; i < m; i++){

int ui = e[i].u, vi = e[i].v, wi = e[i].w;

if (wi < in[vi] && ui != vi) {

pre[vi] = ui;

in[vi] = wi;

}

}

for (int i = 0; i < n; i++) {

if (i == rt)

continue;

if (in[i] == INF)

return -1;

}

in[rt] = 0;

int cnt = 0;

MEM(id, -1), MEM(vist, -1);

for (int i = 0; i < n; i++) {

res += in[i];

v = i;

while (vist[v] != i && id[v] == -1 && v != rt)

vist[v] = i, v = pre[v];

if (v != rt && id[v] == -1) {

for (u = pre[v]; u != v; u = pre[u])

id[u] = cnt;

id[v] = cnt++;

}

}

if (!cnt)

break;

for (int i = 0; i < n; i++) {

if (id[i] == -1)

id[i] = cnt++;

}

for (int i = 0; i < m; i++) {

int ui = e[i].u, vi = e[i].v, wi = e[i].w;

v = vi;

e[i].u = id[ui];

e[i].v = id[vi];

if (ui != vi)

e[i].w -= in[v];

}

n = cnt;

rt = id[rt];

}

return res;

}

#### 6.无根最小树形图

struct Edge

{

int u, v, w;

}e[M];

int pre[N], id[N], vist[N];

int in[N], n, m, root;

int dmst(int rt, int n, int m) {

int u, v;

int res = 0;

while (true) {

for (int i = 0; i < n; i++)

in[i] = INF;

for (int i = 0; i < m; i++){

int ui = e[i].u, vi = e[i].v, wi = e[i].w;

if (wi < in[vi] && ui != vi) {

pre[vi] = ui;

if (ui == rt)

root = i;

in[vi] = wi;

}

}

for (int i = 0; i < n; i++) {

if (i == rt)

continue;

if (in[i] == INF)

return -1;

}

in[rt] = 0;

int cnt = 0;

MEM(id, -1), MEM(vist, -1);

for (int i = 0; i < n; i++) {

res += in[i];

v = i;

while (vist[v] != i && id[v] == -1 && v != rt)

vist[v] = i, v = pre[v];

if (v != rt && id[v] == -1) {

for (u = pre[v]; u != v; u = pre[u])

id[u] = cnt;

id[v] = cnt++;

}

}

if (!cnt)

break;

for (int i = 0; i < n; i++) {

if (id[i] == -1)

id[i] = cnt++;

}

for (int i = 0; i < m; i++) {

int ui = e[i].u, vi = e[i].v, wi = e[i].w;

v = vi;

e[i].u = id[ui];

e[i].v = id[vi];

if (ui != vi)

e[i].w -= in[v];

}

n = cnt;

rt = id[rt];

}

return res;

}

int main()

{

while (~sc("%d %d", &n, &m)) {

int r = 0, m1 = m;

for (int i = 0; i < m; i++) {

int ui, vi, wi;

sc("%d %d %d", &ui, &vi, &wi);

e[i] = { ui, vi, wi };

r += wi;

}

r++;

for (int i = 0; i < n; i++)

e[m++] = { n, i, r };

int ans = dmst(n, n + 1, m);

root -= m1;

if (ans == -1 || ans >= 2 \* r)

printf("impossible\n");

else

printf("%d %d\n", ans - r, root);

printf("\n");

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 7.MST计数

// N \* N \* N

struct node

{

ll w;

int u, v, pos;

bool operator < (const node &oth) const {

return w < oth.w;

}

}e[N];

int f[N], n, m, cnt, sum, T;

int st[N], num[N], tot;

int ans = 1;

int Find(int x) {

return f[x] == x ? x : Find(f[x]);

}

void DFS(int x, int y, int c) {

if (y == st[x + 1]) {

if (c == num[x]) sum++;

return;

}

int l = Find(e[y].u), r = Find(e[y].v);

if (l != r)

f[l] = r, DFS(x, y + 1, c + 1), f[l] = l, f[r] = r;

DFS(x, y + 1, c);

}

int main()

{

cin >> n >> m;

for (int i = 1; i <= m; i++)

sc("%d %d %lld", &e[i].u, &e[i].v, &e[i].w);

sort(e + 1, e + m + 1);

for (int i = 1; i <= m; i++) {

if (e[i].w != e[i - 1].w)

st[++cnt] = i;

e[i].pos = cnt;

}

st[cnt + 1] = m + 1;

for (int i = 1; i <= n; i++)

f[i] = i;

int K = 0;

for (int i = 1; i <= m; i++) {

int u = e[i].u, v = e[i].v;

int uu = Find(u), vv = Find(v);

if (uu != vv) {

f[uu] = vv;

num[e[i].pos]++;

K++;

}

}

if (K != n - 1)

puts("0"), exit(0);

for (int i = 1; i <= n; i++)

f[i] = i;

for (int i = 1; i <= cnt; i++) {

if (num[i]) {

sum = 0, DFS(i, st[i], 0);

for (int j = st[i]; j < st[i + 1]; j++)

f[Find(e[j].u)] = Find(e[j].v);

ans = ((ans % Mod) \* (sum % Mod)) % Mod;

}

}

printf("%d\n", ans);

return 0;

}

### 题集：

#### 1.MST必须边

// Tarjan求割边法

struct node

{

int u, v, w;

bool operator < (const node &oth) const {

return w < oth.w;

}

}e[N];

vector <pir> G[N << 1];

int f[N], n, m, p;

int tot;

void init() {

for (int i = 1; i <= n; i++)

f[i] = i;

}

int find\_(int x) {

if (x != f[x])

x = f[x] = find\_(f[x]);

return x;

}

int dfn[N], low[N], cnt;

void tarjan(int x, int fa) { // fa是编号

dfn[x] = low[x] = ++cnt;

for (auto it : G[x]) {

int v = it.first, id = it.second;

if (id == fa)

continue;

if (!dfn[v]) {

tarjan(v, id);

Min(low[x], low[v]);

if (low[v] > dfn[x])

tot++;

}

else

Min(low[x], dfn[v]);

}

}

int main()

{

cin >> n >> m >> p;

init();

for (int i = 1; i <= m; i++)

sc("%d %d %d", &e[i].u, &e[i].v, &e[i].w);

sort(e + 1, e + m + 1);

int j = 1;

for (int i = 1; i <= m; i = j + 1) {

int u = e[i].u, v = e[i].v, w = e[i].w;

j = i + 1;

while (j <= m && e[j].w == w)

j++;

j--;

for (int k = i; k <= j; k++) {

int uu = find\_(e[k].u), vv = find\_(e[k].v);

if (uu == vv)

continue;

G[uu].push\_back({ vv, k });

G[vv].push\_back({ uu, k });

}

for (int k = i; k <= j; k++) {

int uu = find\_(e[k].u), vv = find\_(e[k].v);

if (uu == vv || dfn[uu])

continue;

tarjan(uu, 0);

}

for (int k = i; k <= j; k++) {

int uu = find\_(e[k].u), vv = find\_(e[k].v);

if (uu == vv)

continue;

G[uu].clear(), G[vv].clear();

dfn[uu] = low[uu] = 0;

dfn[vv] = low[vv] = 0;

f[uu] = vv;

}

}

cout << tot << endl;

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

// 启发式合并

struct node

{

int u, v, w;

bool operator < (const node &oth) const {

return w < oth.w;

}

}e[N];

vector <pir> G[N << 1];

int fz[N];

bool vis[N];

int find\_(int x) {

if (x != fz[x])

x = fz[x] = find\_(fz[x]);

return x;

}

int dep[N], f[N][25];

void dfs(int x, int fa) {

dep[x] = dep[fa] + 1;

f[x][0] = fa;

for (int i = 1; (1 << i) <= dep[x]; i++)

f[x][i] = f[f[x][i - 1]][i - 1];

for (auto it : G[x]) {

int v = it.first;

if (v != fa)

dfs(v, x);

}

}

int LCA(int x, int y) {

if (dep[x] > dep[y])

swap(x, y);

for (int i = 20; i >= 0; i--) {

if (dep[y] - (1 << i) >= dep[x])

y = f[y][i];

}

if (x == y)

return x;

for (int i = 20; i >= 0; i--) {

if (f[x][i] == f[y][i])

continue;

x = f[x][i], y = f[y][i];

}

return f[x][0];

}

unordered\_map <int, int> mp[N];

int ans;

void DFS(int x, int fa) {

for (auto it : G[x]) {

int v = it.first, w = it.second;

if (v == fa)

continue;

DFS(v, x);

if (mp[v].count(w) == NULL)

ans++;

if (SZ(mp[v]) > SZ(mp[x]))

swap(mp[x], mp[v]);

for (auto it : mp[v]) {

int wi = it.first;

mp[x][wi] += it.second;

if (!mp[x][wi])

mp[x].erase(it.first);

}

}

}

int main()

{

int n, m, p;

cin >> n >> m >> p;

for (int i = 1; i <= n; i++)

fz[i] = i;

for (int i = 1; i <= m; i++)

sc("%d %d %d", &e[i].u, &e[i].v, &e[i].w);

sort(e + 1, e + m + 1);

for (int i = 1; i <= m; i++) {

int u = e[i].u, v = e[i].v, w = e[i].w;

int uu = find\_(u), vv = find\_(v);

if (uu != vv) {

fz[uu] = vv;

G[u].push\_back({ v, w });

G[v].push\_back({ u, w });

vis[i] = true;

}

}

dfs(1, 0);

for (int i = 1; i <= m; i++) {

if (vis[i])

continue;

int u = e[i].u, v = e[i].v, w = e[i].w;

int lca = LCA(u, v);

mp[u][w]++, mp[v][w]++, mp[lca][w] -= 2;

}

DFS(1, 0);

cout << ans << endl;

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 2.完全图最多MST构造

int main()

{

int T, Case = 0;

cin >> T;

while (T--) {

int n;

sc("%d", &n);

printf("Case #%d: %d\n", ++Case, n / 2);

for (int i = 1; i <= n / 2; i++) {

int u = i, v = i + 1;

for (int j = 1; j <= n / 2; j++) {

printf("%d %d\n", u, v);

u = (u - 2 + n) % n + 1;

if (u != v)

printf("%d %d\n", u, v);

v++;

}

}

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 3.生成树使得点1度数为d

struct node

{

int u, v, w;

bool operator < (const node &oth) const {

return w < oth.w;

}

}a[N];

vector <int> G[N];

int f[N], n, m, k;

int cnt1;

pir eg[N];

bool vis[N], mst[N];

int Find(int x) {

if (x != f[x])

x = f[x] = Find(f[x]);

return x;

}

void Merge(int x, int y) {

x = Find(x);

y = Find(y);

if (x != y)

f[x] = y;

}

void DFS(int x) {

vis[x] = true;

for (auto v : G[x]) {

if (Find(v) == Find(x) || v == 1)

continue;

Merge(x, v);

printf("%d %d\n", x, v);

DFS(v);

}

}

int main()

{

cin >> n >> m >> k;

for (int i = 1; i <= n; i++)

f[i] = i;

for (int i = 1; i <= m; i++) {

int u, v, w = 0;

sc("%d %d", &u, &v);

if (u == 1 || v == 1)

w = 1, cnt1++;

a[i] = { u, v, w };

G[u].push\_back(v);

G[v].push\_back(u);

}

if (cnt1 < k)

puts("NO"), exit(0); // 度数不够

sort(a + 1, a + m + 1);

// 先看不和1连的边

for (int i = 1; i <= m; i++) {

int u = a[i].u, v = a[i].v;

int uu = Find(u), vv = Find(v);

if (a[i].w == 0) { // 连起来

if (uu == vv)

continue;

f[uu] = vv;

}

else

break;

}

set <int> st;

for (int i = 2; i <= n; i++)

st.insert(Find(i));

if (SZ(st) > k) // 太多必须边要连

puts("NO"), exit(0);

int cnt = 0; // 先连必须边

for (int i = 1; i <= m; i++) {

if (!k)

break;

if (!a[i].w)

continue;

int u = a[i].u, v = a[i].v;

int uu = Find(u), vv = Find(v);

if (uu != vv) {

k--;

mst[i] = true;

f[uu] = vv;

eg[++cnt] = { u, v };

}

}

for (int i = 1; i <= m; i++) { // 其他边随意

if (!k)

break;

if (!a[i].w || mst[i])

continue;

int u = a[i].u, v = a[i].v;

k--;

eg[++cnt] = { u, v };

}

for (int i = 1; i <= n; i++)

f[i] = i;

puts("YES");

for (int i = 1; i <= cnt; i++) {

int u = eg[i].first;

int v = eg[i].second;

printf("%d %d\n", u, v);

Merge(u, v);

}

for (int i = 2; i <= n; i++)

if (!vis[i])

DFS(i);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

## 2-sat

### 1.2-sat

//N个人，每组两个人必选一个，M对矛盾关系，能否选N个人

vector <int> G[N];

int dfn[N], low[N], cnt, scnt;

int stk[N], s;

int suo[N], n, m;

bool vis[N];

void init() {

for (int i = 0; i <= 2 \* n; i++)

G[i].clear(), dfn[i] = vis[i] = suo[i] = 0;

s = cnt = scnt = 0;

}

void Tarjan(int x) {

dfn[x] = low[x] = ++cnt;

stk[++s] = x;

vis[x] = true;

for (auto v : G[x]) {

if (!dfn[v]) {

Tarjan(v);

Min(low[x], low[v]);

}

else if (vis[v])

Min(low[x], low[v]);

}

if (dfn[x] == low[x]) {

scnt++;

int k;

do {

k = stk[s--];

vis[k] = false;

suo[k] = scnt;

} while (k != x);

}

}

int main()

{

while (~sc("%d %d", &n, &m)) {

init();

//0-n妻子，n-2\*n-1丈夫

for (int i = 0; i < m; i++) {

int a1, a2, c1, c2;

sc("%d %d %d %d", &a1, &a2, &c1, &c2);

int u = a1, v = a2;

if (c1) // 丈夫

u += n;

if (c2)

v += n;

if (c2)

G[u].push\_back(v - n);

else

G[u].push\_back(v + n);

if (c1)

G[v].push\_back(u - n);

else

G[v].push\_back(u + n);

}

for (int i = 0; i < 2 \* n; i++) {

if (!dfn[i])

Tarjan(i);

}

bool ok = true;

for (int i = 0; i < n; i++) {

if (suo[i] == suo[i + n]) {

ok = false;

break;

}

}

if (ok)

printf("YES\n");

else

printf("NO\n");

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

### 2.2-sat字典序输出方案

vector <int> G[N];

int n, m;

stack <int> st;

bool vis[N];

void init() {

for (int i = 0; i < 2 \* n; i++)

G[i].clear(), vis[i] = false;

}

bool DFS(int x) {

if (vis[x ^ 1]) return false; // 选到相邻的点， 直接退出

if (vis[x]) return true;

vis[x] = true, st.push(x);

for (auto v : G[x]) {

if (!DFS(v))

return false;

}

return true;

}

bool Check() {

for (int i = 0; i < 2 \* n; i += 2) {

if (!vis[i] && !vis[i ^ 1]) {

while (!st.empty())

st.pop();

if (!DFS(i)) {

while (!st.empty()) { // 一个点不可选清空

int k = st.top(); st.pop();

vis[k] = false;

}

if (!DFS(i ^ 1))

return false; // 两个点都不可选

}

}

}

return true;

}

int main()

{

while (~sc("%d %d", &n, &m)) {

init();

for (int i = 0; i < m; i++) {

int u, v;

sc("%d %d", &u, &v);

u--, v--;

G[u].push\_back(v ^ 1);

G[v].push\_back(u ^ 1);

}

if (Check()) {

for (int i = 0; i < 2 \* n; i += 2) {

if (vis[i])

printf("%d\n", i + 1); // vis表示选择

else

printf("%d\n", i + 2);

}

}

else printf("NIE\n");

}

return 0;

}

## 婚姻稳定

/\*有N男N女，每个人都按照他对异性的喜欢程度排名。现在需要写出一个算法安排这N个男的、N个女的结婚，要求两个人的婚姻应该是稳定的。

何为稳定？

有两对夫妻M1 F2，M2 F1。M1心目中更喜欢F1，但是他和F2结婚了，M2心目中更喜欢F2，但是命运却让他和F1结婚了，显然这样的婚姻是不稳定的，随时都可能发生M1和F1私奔或者M2和F2私奔的情况。所以在做出匹配选择的时候（也就是结婚的时候），我们需要做出稳定的选择，以防这种情况的发生。\*/

// N \* N

//原文链接：https://blog.csdn.net/cscmaker/article/details/8291131

int man\_que[N][N], wom\_que[N][N];

int wif[N], hus[N], nxt[N], n, T;

map <char, int> id\_man, id\_wom;

map <int, char> name\_man, name\_wom;

string line;

queue <int> q;

void init() {

id\_man.clear(), id\_wom.clear(), name\_man.clear(), name\_wom.clear();

MEM(nxt, 0); MEM(man\_que, 0); MEM(wom\_que, 0);

MEM(wif, 0); MEM(hus, 0);

}

void Engage(int x, int y) {

int m = hus[y];

if (m) {

wif[m] = 0;

q.push(m);

}

wif[x] = y, hus[y] = x;

}

int main()

{

cin >> T;

while (T--) {

sc("%d", &n); init();

for (int i = 1; i <= n; i++) {

cin >> line;

id\_man[line[0]] = i, name\_man[i] = line[0];

}

for (int i = 1; i <= n; i++) {

cin >> line;

id\_wom[line[0]] = i, name\_wom[i] = line[0];

}

for (int i = 1; i <= n; i++) {

cin >> line; int tot = 0;

int k = id\_man[line[0]];

for (int j = 2; j < SZ(line); j++)

man\_que[k][++tot] = id\_wom[line[j]];

wif[i] = 0; nxt[i] = 1; q.push(i);

}

for (int i = 1; i <= n; i++) {

cin >> line; int tot = 0;

int k = id\_wom[line[0]];

for (int j = 2; j < SZ(line); j++)

wom\_que[k][id\_man[line[j]]] = ++tot;

hus[i] = 0;

}

while (!q.empty()) {

int m = q.front(); q.pop();

int w = man\_que[m][nxt[m]++];

if (w == 0) continue;

if (!hus[w]) Engage(m, w);

else if (wom\_que[w][m] < wom\_que[w][hus[w]])

Engage(m, w);

else q.push(m);

}

while (!q.empty()) q.pop();

for (map <char, int> ::iterator it = id\_man.begin(); it != id\_man.end(); it++)

cout << it->first << " " << name\_wom[wif[it->second]] << endl;

cout << endl;

}

return 0;

}

# 数据结构

## 并查集

### 模板

#### 1.路径压缩

int f[N], n, m;

void init() {

for (int i = 1; i <= n; i++)

f[i] = i;

}

int Find(int x) {

if (x != f[x])

x = f[x] = Find(f[x]);

return x;

}

void Merge(int x, int y) {

x = Find(x);

y = Find(y);

if (x != y)

f[x] = y;

}

#### 2.启发式合并

int f[N], sz[N], n, m;

void init() {

for (int i = 1; i <= n; i++)

f[i] = i, sz[i] = 1;

}

int Find(int x) {

while (x != f[x])

x = f[x];

return x;

}

void Merge(int x, int y) {

x = Find(x);

y = Find(y);

if (x == y)

return;

if (sz[x] > sz[y])

swap(x, y); // 小的连到大的

f[x] = y, sz[y] += sz[x];

}

#### 3.可撤销并查集

int fa[N], sz[N], n, m;

stack <pir> st;

void init() {

for (int i = 1; i <= n; i++)

fa[i] = i, sz[i] = 1;

}

int find\_(int x) {

while (x != fa[x])

x = fa[x];

return x;

}

void unite(int x, int y) {

x = find\_(x);

y = find\_(y);

if (x == y)

return;

if (sz[x] > sz[y])

swap(x, y);

fa[x] = y;

st.push({ x, y });

sz[y] += sz[x];

}

void Cancle() {

int x = st.top().first, y = st.top().second;

st.pop();

fa[x] = x;

sz[y] -= sz[x];

}

### 题集

#### 1.启发式合并差分

// 合并点，连通块加某个值

vector <int> G[N];

int f[N], n, q;

int ans[N], c[N];

void Init() {

for (int i = 1; i <= n; i++)

f[i] = i, G[i].push\_back(i);

}

int Find(int x) {

if (x != f[x])

x = f[x] = Find(f[x]);

return x;

}

int main()

{

cin >> n >> q;

Init();

while (q--) {

int op, x, y;

sc("%d %d %d", &op, &x, &y);

if (op == 1) {

int u = Find(x), v = Find(y);

if (u == v)

continue;

if (G[u].size() > G[v].size())

swap(u, v);

for (auto it : G[u]) {

G[v].push\_back(it);

ans[it] -= c[v] - c[u];

}

f[u] = v;

}

else

c[Find(x)] += y;

}

for (int i = 1; i <= n; i++) {

ans[i] += c[Find(i)];

printf("%d", ans[i]);

if (i == n)

printf("\n");

else

printf(" ");

}

return 0;

}

#### 2.补图连通块

// 注意复杂度的计算

set <int> G[N], s;

int f[N], n, m;

bool vis[N];

void Init() {

for (int i = 1; i <= n; i++)

f[i] = i, s.insert(i);

}

int Find(int x) {

if (x != f[x])

x = f[x] = Find(f[x]);

return x;

}

void Merge(int x, int y) {

x = Find(x), y = Find(y);

if (x != y)

f[x] = y;

}

void BFS(int x) {

queue <int> q;

q.push(x);

vis[x] = true;

while (!q.empty()) {

int u = q.front();

q.pop();

for (auto it = s.begin(); it != s.end();) {

int v = \*it++;

if (G[u].find(v) == G[u].end()) { // 总共N次

q.push(v);

vis[v] = true;

s.erase(v);

Merge(u, v);

}

// 无边一共2 \* M次

}

}

}

int main()

{

cin >> n >> m;

Init();

for (int i = 0; i < m; i++) {

int u, v;

sc("%d %d", &u, &v);

G[u].insert(v);

G[v].insert(u);

}

for (int i = 1; i <= n; i++) {

if (!vis[i])

BFS(i);

}

set <int> st;

for (int i = 1; i <= n; i++)

st.insert(Find(i));

printf("%d\n", SZ(st) - 1);

return 0; // 改数组大小!!!用pair改宏定义!!!

}

#### 3.可撤销并查集删除边求连通块最大点权

struct node

{

int id, w;

bool operator < (const node &oth) const {

return w > oth.w;

}

};

struct Ask

{

int op, x;

}qi[N];

pir eg[N];

set <node> s[N];

int fa[N], sz[N], n, m, q;

int a[N];

bool vis[N];

stack <pir> st;

void Init() {

for (int i = 1; i <= n; i++)

fa[i] = i, sz[i] = 1;

}

int Find(int x) {

while (x != fa[x])

x = fa[x];

return x;

}

void Merge(int x, int y) {

x = Find(x);

y = Find(y);

if (x == y) {

st.push({ 0, 0 });

return;

}

if (sz[x] > sz[y])

swap(x, y);

fa[x] = y;

st.push({ x, y });

sz[y] += sz[x];

for (auto it : s[x])

s[y].insert(it);

}

void Cancle() {

int x = st.top().first, y = st.top().second;

st.pop();

if (!x && !y)

return;

fa[x] = x;

sz[y] -= sz[x];

for (auto it : s[x]) {

if (s[y].find(it) != s[y].end())

s[y].erase(it);

}

}

int main()

{

cin >> n >> m >> q;

Init();

for (int i = 1; i <= n; i++)

sc("%d", &a[i]), s[i].insert({ i, a[i] });

for (int i = 1; i <= m; i++)

sc("%d %d", &eg[i].first, &eg[i].second);

for (int i = 1; i <= q; i++) {

sc("%d %d", &qi[i].op, &qi[i].x);

if (qi[i].op == 2)

vis[qi[i].x] = true;

}

for (int i = 1; i <= m; i++)

if (!vis[i])

Merge(eg[i].first, eg[i].second);

for (int i = q; i >= 1; i--) {

if (qi[i].op == 1)

continue;

int u = eg[qi[i].x].first;

int v = eg[qi[i].x].second;

Merge(u, v);

}

for (int i = 1; i <= q; i++) {

int op = qi[i].op, x = qi[i].x;

if (op == 1) {

int u = Find(x);

while (!s[u].empty() && !a[s[u].begin()->id])

s[u].erase(s[u].begin());

if (s[u].empty())

puts("0");

else {

printf("%d\n", s[u].begin()->w);

a[s[u].begin()->id] = 0;

}

}

else

Cancle();

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

## 最近共同祖先

#### 1.LCA倍增

vector <int> G[N];

int dep[N], f[N][25];

void DFS(int x, int fa) {

dep[x] = dep[fa] + 1;

f[x][0] = fa;

for (int i = 1; (1 << i) <= dep[x]; i++)

f[x][i] = f[f[x][i - 1]][i - 1];

for (auto v : G[x]) {

if (v != fa)

DFS(v, x);

}

}

int LCA(int x, int y) {

if (dep[x] > dep[y])

swap(x, y);

for (int i = 20; i >= 0; i--) {

if (dep[y] - (1 << i) >= dep[x])

y = f[y][i];

}

if (x == y)

return x;

for (int i = 20; i >= 0; i--) {

if (f[x][i] == f[y][i])

continue;

x = f[x][i], y = f[y][i];

}

return f[x][0];

}

#### 2.LCA离线

vector <int> G[N];

vector <pir> Ask[N];

int f[N], ans[N], n;

bool vis[N];

void init() {

for (int i = 1; i <= n; i++)

f[i] = i;

}

int find\_(int x) {

if (x != f[x])

x = f[x] = find\_(f[x]);

return x;

}

void unite(int x, int y) {

x = find\_(x);

y = find\_(y);

if (x != y)

f[x] = y;

}

void dfs(int x, int fa) {

vis[x] = true;

for (auto v : G[x]) {

if (v == fa)

continue;

dfs(v, x);

f[v] = x;

}

for (auto it : Ask[x]) {

int u = it.first, idx = it.second;

if (vis[u] && !ans[idx])

ans[idx] = find\_(u);

}

}

#### 3.LCA按顺序打印路径

void lca(int x, int y)

{

if (x == y)

printf(x), return;

if (dep[x] >= dep[y])

printf(x), lca(fz[x], y);

else

lca(x, fz[y]), printf(y);

}

#### 4.树上K条路径交

while (q--) {

int m, u, v, uu, vv, flag = 1;

scanf("%d %d %d", &m, &u, &v);

for (int i = 1; i < m; ++i) {

scanf("%d%d", &uu, &vv);

if (flag) {

int a[] = { LCA(u, uu), LCA(u, vv), LCA(v, uu), LCA(v, vv) };

sort(a, a + 4, [](int x, int y){return dep[x] > dep[y]; }); //只保留深度最大两个

if (dep[a[1]] < dep[LCA(u, v)] || dep[a[1]] < dep[LCA(uu, vv)]) //如果深度较小点 超出两路径lca则无交

flag = 0;

u = a[0], v = a[1];

}

}

printf("%d\n", flag ? abs(dep[u] + dep[v] - dep[LCA(u, v)] \* 2 + 1) : 0); //最终两点间距离即为路径交

}

#### 5.树上ST表

vector <pir> G[N];

int dep[N], f[N][25], st[N][25];

void dfs(int x, int fa) {

dep[x] = dep[fa] + 1;

f[x][0] = fa;

for (int i = 1; (1 << i) <= dep[x]; i++) {

f[x][i] = f[f[x][i - 1]][i - 1];

st[x][i] = max(st[x][i - 1], st[f[x][i - 1]][i - 1]);

}

for (auto it : G[x]) {

int v = it.v, w = it.w;

if (v != fa)

st[v][0] = w, dfs(v, x);

}

}

int LCA(int x, int y) {

int mx = -INF;

if (dep[x] > dep[y])

swap(x, y);

for (int i = 20; i >= 0; i--) {

if (dep[y] - (1 << i) >= dep[x])

Max(mx, st[y][i]), y = f[y][i];

}

if (x == y)

return mx;

for (int i = 20; i >= 0; i--) {

if (f[x][i] == f[y][i]) continue;

Max(mx, max(st[x][i], st[y][i]));

x = f[x][i], y = f[y][i];

}

return max(mx, max(st[x][0], st[y][0])); // 最后要取max

}

#### 6.树上差分

边：

val[u]++, val[v]++, val[lca] – 2

点

val[u]++, val[v]++, val[lca]--, val[fz[lca]]--

## 灭绝树

### 1.每个点支配数

vector <int> G[N], r[N];

int f[N][25], dep[N];

int in[N], top[N], n, cnt;

int ans[N];

int LCA(int x, int y) {

if (dep[x] > dep[y])

swap(x, y);

for (int i = 20; i >= 0; i--) {

if (dep[y] - (1 << i) >= dep[x])

y = f[y][i];

}

if (x == y)

return x;

for (int i = 20; i >= 0; i--) {

if (f[x][i] == f[y][i])

continue;

x = f[x][i], y = f[y][i];

}

return f[x][0];

}

void Topsort() {

queue <int> q;

for (int i = 1; i <= n; i++) {

if (!in[i])

q.push(i);

}

while (!q.empty()) {

int u = q.front();

q.pop();

top[++cnt] = u;

if (r[u].empty())

f[u][0] = 0;

else {

int lca = r[u][0];

for (auto v : r[u])

lca = LCA(lca, v);

f[u][0] = lca;

}

dep[u] = dep[f[u][0]] + 1;

for (int i = 1; (1 << i) <= dep[u]; i++)

f[u][i] = f[f[u][i - 1]][i - 1];

for (auto v : G[u]) {

in[v]--;

if (!in[v])

q.push(v);

}

}

}

int main()

{

cin >> n;

for (int i = 1; i <= n; i++) {

int u;

while (sc("%d", &u) && u) {

G[u].push\_back(i);

r[i].push\_back(u);

in[i]++;

}

}

Topsort();

for (int i = n; i >= 1; i--) {

int u = top[i];

int fa = f[u][0];

ans[u]++;

ans[fa] += ans[u];

}

for (int i = 1; i <= n; i++)

printf("%d\n", ans[i] - 1);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

### 2.删多少点使得x，y其中一个无法到达终点

vector <int> G[N], r[N];

int f[N][25], dep[N];

int dis[N], in[N], n, m;

void Init() {

MEM(f, 0);

for (int i = 1; i <= n; i++) {

G[i].clear(), r[i].clear();

dep[i] = in[i] = 0;

}

}

int LCA(int x, int y) {

if (dep[x] > dep[y])

swap(x, y);

for (int i = 20; i >= 0; i--) {

if (dep[y] - (1 << i) >= dep[x])

y = f[y][i];

}

if (x == y)

return x;

for (int i = 20; i >= 0; i--) {

if (f[x][i] == f[y][i])

continue;

x = f[x][i], y = f[y][i];

}

return f[x][0];

}

void Topsort() {

queue <int> q;

for (int i = 1; i <= n; i++) {

if (!in[i])

q.push(i);

}

while (!q.empty()) {

int u = q.front();

q.pop();

if (r[u].empty())

f[u][0] = 0;

else {

int lca = r[u][0];

for (auto v : r[u])

lca = LCA(lca, v);

f[u][0] = lca;

}

dep[u] = dep[f[u][0]] + 1;

for (int i = 1; (1 << i) <= dep[u]; i++)

f[u][i] = f[f[u][i - 1]][i - 1];

for (auto v : G[u]) {

in[v]--;

if (!in[v])

q.push(v);

}

G[u].clear();

G[f[u][0]].push\_back(u);

}

}

int main()

{

int T; cin >> T;

while (T--) {

sc("%d %d", &n, &m);

Init();

for (int i = 0; i < m; i++) {

int u, v;

sc("%d %d", &u, &v);

r[u].push\_back(v);

G[v].push\_back(u);

in[u]++;

}

Topsort();

int q;

sc("%d", &q);

while (q--) {

int u, v;

sc("%d %d", &u, &v);

int lca = LCA(u, v);

printf("%d\n", dep[u] + dep[v] - dep[lca]);

}

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

## ST表

### 1.一维ST表

int st[N][25], n, m;

int Log[N];

void init() {

Log[0] = -1;

for (int i = 1; i <= n; i++)

sc("%d", &st[i][0]), Log[i] = Log[i >> 1] + 1;

for (int p = 1; p <= Log[n]; p++) {

for (int i = 1; i + (1 << p) - 1 <= n; i++)

st[i][p] = max(st[i][p - 1], st[i + (1 << (p - 1))][p - 1]);

}

}

int Ask(int l, int r) {

int d = Log[r - l + 1];

return max(st[l][d], st[r - (1 << d) + 1][d]);

}

### 2.二维ST表

// 滚动数组优化

int tb[2][N][N];

int lg[N], n, m, s;

void ST() {

lg[0] = -1;

for (int i = 1; i <= s; i++)

lg[i] = lg[i >> 1] + 1;

for (int i = 1; i <= n; i++)

for (int j = 1; j <= m; j++)

tb[0][i][j] = i \* j / gcd(i, j);

for (int k = 1; k <= lg[s]; k++) {

int kk = k & 1;

for (int i = 1; i + (1 << k) - 1 <= n; i++) {

for (int j = 1; j + (1 << k) - 1 <= m; j++) {

tb[kk][i][j] = max({ tb[!kk][i][j], tb[!kk][i][j + (1 << k - 1)],

tb[!kk][i + (1 << k - 1)][j], tb[!kk][i + (1 << k - 1)][j + (1 << k - 1)] });

}

}

}

}

int Ask(int x1, int y1, int k) {

int x2 = x1 + k - (1 << lg[k]), y2 = y1 + k - (1 << lg[k]); k = lg[k] & 1;

return max({ tb[k][x1][y1], tb[k][x1][y2], tb[k][x2][y1], tb[k][x2][y2] });

}

// 普通版本

int a[N][N];

int mx[N][N][15], mi[N][N][15];

int Log[N], n, m, s;

void Init() {

for (int i = 2; i < N; i++)

Log[i] = Log[i >> 1] + 1;

for (int i = 1; i <= n; i++)

for (int j = 1; j <= m; j++)

mx[i][j][0] = mi[i][j][0] = a[i][j];

for (int k = 1; k <= 12; k++) {

for (int i = 1; i + (1 << k) - 1 <= n; i++) {

for (int j = 1; j + (1 << k) - 1 <= m; j++) {

int g1 = mx[i][j][k - 1];

int g2 = mx[i + (1 << (k - 1))][j][k - 1];

int g3 = mx[i][j + (1 << (k - 1))][k - 1];

int g4 = mx[i + (1 << k - 1)][j + (1 << k - 1)][k - 1];

mx[i][j][k] = max({ g1, g2, g3, g4 });

g1 = mi[i][j][k - 1];

g2 = mi[i + (1 << (k - 1))][j][k - 1];

g3 = mi[i][j + (1 << (k - 1))][k - 1];

g4 = mi[i + (1 << k - 1)][j + (1 << k - 1)][k - 1];

mi[i][j][k] = min({ g1, g2, g3, g4 });

}

}

}

}

int Ask(int x, int y, int sz) {

int k = Log[sz];

int g1 = mx[x][y][k];

int g2 = mx[x + sz - (1 << k)][y][k];

int g3 = mx[x][y + sz - (1 << k)][k];

int g4 = mx[x + sz - (1 << k)][y + sz - (1 << k)][k];

int max\_ = max({ g1, g2, g3, g4 });

g1 = mi[x][y][k];

g2 = mi[x + sz - (1 << k)][y][k];

g3 = mi[x][y + sz - (1 << k)][k];

g4 = mi[x + sz - (1 << k)][y + sz - (1 << k)][k];

int min\_ = min({ g1, g2, g3, g4 });

return max\_ - min\_;

}

## 树的直径

### 模板

#### 1.树形dp

vector <pir> G[N];

int dp[N], ans;

void DFS(int x, int fa) {

for (auto it : G[x]) {

int v = it.second, w = it.first;

if (v == fa)

continue;

DFS(v, x);

Max(ans, dp[x] + dp[v] + w);

Max(dp[x], dp[v] + w);

}

}

#### 2.两次DFS

vector <pir> G[N];

int mx, n;

void DFS(int x, int fa, ll dist, int &rt) {

if (dist > mx)

mx = dist, rt = x;

for (auto it : G[x]) {

int v = it.second; ll w = it.first;

if (v != fa)

DFS(v, x, dist + w, rt);

}

}

int rt1, rt2;

mx = 0, DFS(1, 0, 0, rt1);

mx = 0, DFS(rt1, 0, 0, rt2);

### 题集

#### 1.树直径必须边

vector <pir> G[N];

int f[N], r1, r2;

ll dis[N], mx;

bool vis[N];

void dfs(int x, int fa, int &now) {

if (dis[x] > mx)

mx = dis[x], now = x;

f[x] = fa;

for (auto it : G[x]) {

int v = it.second, w = it.first;

if (v == fa)

continue;

dis[v] = dis[x] + w;

dfs(v, x, now);

}

}

void DFS(int x, int fa, ll dist) {

Max(mx, dist);

for (auto it : G[x]) {

int v = it.second, w = it.first;

if (v == fa || vis[v])

continue;

DFS(v, x, dist + w);

}

}

int main()

{

int n;

cin >> n;

for (int i = 1; i < n; i++) {

int ui, vi, wi;

sc("%d %d %d", &ui, &vi, &wi);

G[ui].push\_back({ wi, vi });

G[vi].push\_back({ wi, ui });

}

mx = -LINF;

dfs(1, 0, r1);

MEM(dis, 0);

mx = -LINF, dfs(r1, 0, r2);

printf("%lld\n", mx);

for (int i = r2; i; i = f[i])

vis[i] = true;

bool ok = false;

int l = r1, r = r2;

for (int i = f[r2]; i != r1; i = f[i]) {

mx = 0;

ll dl = dis[i], dr = dis[r2] - dis[i];

DFS(i, 0, 0);

if (mx == dr)

r = i;

if (mx == dl && !ok)

l = i, ok = true;

}

int ans = 0;

while (r != l)

r = f[r], ans++;

printf("%d\n", ans);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 2.树直径及其数量

vector <pir> G[N];

int cnt[N], n;

ll d[N], k, ans;

void DFS(int x, int fa) {

d[x] = 0, cnt[x] = 1;

for (auto it : G[x]) {

int v = it.second; ll w = it.first;

if (v == fa)

continue;

DFS(v, x);

if (d[x] + d[v] + w > ans) {

ans = d[x] + d[v] + w;

k = cnt[x] \* cnt[v];

}

else if (d[x] + d[v] + w == ans)

k += cnt[x] \* cnt[v];

if (d[x] < d[v] + w) {

d[x] = d[v] + w;

cnt[x] = cnt[v];

}

else if (d[x] == d[v] + w)

cnt[x] += cnt[v];

}

}

## 树的重心

### 1.树形dp求树重心

vector <int> G[N];

int t[N], mxt[N], n, mi;

void DFS(int x, int fa, int &u, int sum) {

t[x] = 1;

for (auto v : G[x]) {

if (v == fa)

continue;

DFS(v, x, u, sum);

t[x] += t[v];

Max(mxt[x], t[v]);

}

Max(mxt[x], sum - t[x]);

if (mxt[x] < mi)

mi = mxt[x], u = x;

}

### 2.一些性质

1.当一个点是整颗树的重心时，以它为根的所有的子树尺寸都不超过整颗树一半。

2.所有点到这个点的距离和是最小的(默认边权为1)。

3.如果有多个重心，那么所有点到这些点的距离都是一样的。

4.两颗树连接起来形成新树，那么重心在两颗树重心的路径上。

5.把一颗树删除或者添加一个节点，那重心最多偏移距离1。

## 点分治

### 1.询问有无长度为k的路径

// N \* log(N)

vector <pir> G[N];

int mxt[N], sz[N], n, q;

int ask[N], rt, snum, mx;

bool vis[N], ans[110];

void DFS\_Mid(int x, int fa) { // 子树重心

mxt[x] = 0, sz[x] = 1;

for (auto it : G[x]) {

int v = it.first;

if (vis[v] || v == fa)

continue;

DFS\_Mid(v, x);

sz[x] += sz[v];

Max(mxt[x], sz[v]);

}

Max(mxt[x], snum - sz[x]);

if (mxt[x] < mx)

mx = mxt[x], rt = x;

}

int d[N], cnt[M], res[M]; // 距离，距离数，all距离类型

int now[M]; // 当前路径

void DFS(int x, int fa) {

now[++now[0]] = d[x];

for (auto it : G[x]) {

int v = it.first, w = it.second;

if (vis[v] || v == fa || d[x] + w > 1e7)

continue;

d[v] = d[x] + w;

DFS(v, x);

}

}

void Calc(int x) {

res[0] = 0, cnt[0] = 1;

for (auto it : G[x]) {

int v = it.first, w = it.second;

if (vis[v])

continue;

d[v] = w, now[0] = 0;

DFS(v, x);

for (int i = 1; i <= q; i++) { // 询问

if (ans[i])

continue;

for (int j = 1; j <= now[0]; j++) { // 当前路径

if (ask[i] >= now[j] && cnt[ask[i] - now[j]]) {

ans[i] = true;

break;

}

}

}

for (int j = 1; j <= now[0]; j++)

res[++res[0]] = now[j], cnt[now[j]]++;

}

for (int i = 1; i <= res[0]; i++)

cnt[res[i]] = 0;

}

void Solve(int x) {

vis[x] = true;

Calc(x);

for (auto it : G[x]) {

int v = it.first;

if (vis[v])

continue;

snum = sz[v], mx = INF;

DFS\_Mid(v, 0);

Solve(rt);

}

}

int main()

{

cin >> n >> q;

for (int i = 1; i < n; i++) {

int u, v, w;

sc("%d %d %d", &u, &v, &w);

G[u].push\_back({v, w});

G[v].push\_back({u, w});

}

for (int i = 1; i <= q; i++)

sc("%d", &ask[i]);

snum = n, mx = INF;

DFS\_Mid(1, 0);

Solve(rt);

for (int i = 1; i <= q; i++)

if (ans[i])

puts("AYE");

else

puts("NAY");

return 0; // 改数组大小!!!用pair改宏定义!!!

}

### 2.模3为0的点对数

vector <pir> G[N];

int n, mxt[N], t[N], rt;

int mx, snum;

ll ans;

bool vis[N];

void DFS\_Mid(int x, int fa) {

mxt[x] = 0, t[x] = 1;

for (auto it : G[x]) {

int v = it.first;

if (vis[v] || v == fa)

continue;

DFS\_Mid(v, x);

t[x] += t[v];

Max(mxt[x], t[v]);

}

Max(mxt[x], snum - t[x]);

if (mxt[x] < mx)

mx = mxt[x], rt = x;

}

// 点分治

int dp[5], d[N], num[3];

void DFS(int x, int fa) {

dp[d[x]]++;

for (auto it : G[x]) {

int v = it.first, w = it.second;

if (vis[v] || v == fa)

continue;

d[v] = (d[x] + w) % 3;

DFS(v, x);

}

}

void Calc(int x) {

num[0] = 1;

for (auto it : G[x]) {

int v = it.first, w = it.second;

if (vis[v])

continue;

for (int i = 0; i < 3; i++)

dp[i] = 0;

d[v] = w;

DFS(v, x);

for (int i = 0; i < 3; i++) {

if (num[(3 - i) % 3])

ans += dp[i] \* num[(3 - i) % 3];

}

for (int i = 0; i < 3; i++)

num[i] += dp[i];

}

for (int i = 0; i < 3; i++)

num[i] = 0;

}

void Solve(int x) { // 找重心

vis[x] = true;

Calc(x);

for (auto it : G[x]) {

int v = it.first;

if (vis[v])

continue;

snum = t[v], mx = INF;

DFS\_Mid(v, 0);

Solve(rt);

}

}

ll gcd(ll a, ll b) {

return !b ? a : gcd(b, a % b);

}

int main()

{

cin >> n;

for (int i = 1; i < n; i++) {

int u, v, w;

sc("%d %d %d", &u, &v, &w);

G[u].push\_back({v, w % 3});

G[v].push\_back({u, w % 3});

}

snum = n, mx = INF;

DFS\_Mid(1, 0);

Solve(rt);

ans <<= 1, ans += n;

ll gc = gcd(ans, 1ll \* n \* n);

printf("%lld/%lld\n", ans / gc, 1ll \* n \* n / gc);

return 0; // 改数组大小!!!用pair改宏定义!!!

}

### 3.长度为K的路径边数最小

vector <pir> G[N];

int mxt[N], sz[N], n, k;

int rt, mx, scnt, ans = INF;

bool vis[N];

void DFS\_Mid(int x, int fa) {

mxt[x] = 0, sz[x] = 1;

for (auto it : G[x]) {

int v = it.first;

if (vis[v] || v == fa)

continue;

DFS\_Mid(v, x);

sz[x] += sz[v];

Max(mxt[x], sz[v]);

}

Max(mxt[x], scnt - sz[x]);

if (mxt[x] < mx)

mx = mxt[x], rt = x;

}

int d[N], cnt[M], mi[M];

int now[M], num[M], res[M];

void DFS(int x, int fa, int dist) {

now[++now[0]] = d[x]; // 距离

num[++num[0]] = dist; // 长度

for (auto it : G[x]) {

int v = it.first, w = it.second;

if (v == fa || vis[v] || d[x] + w > k)

continue;

d[v] = d[x] + w;

DFS(v, x, dist + 1);

}

}

void Calc(int x) {

cnt[0] = 1, res[0] = mi[0] = 0;

for (auto it : G[x]) {

int v = it.first, w = it.second;

if (vis[v] || w > k)

continue;

d[v] = w, num[0] = now[0] = 0;

DFS(v, x, 1);

for (int i = 1; i <= now[0]; i++) {

if (cnt[k - now[i]])

Min(ans, mi[k - now[i]] + num[i]);

}

for (int i = 1; i <= now[0]; i++) {

if (!mi[now[i]])

mi[now[i]] = num[i];

else

Min(mi[now[i]], num[i]);

res[++res[0]] = now[i];

cnt[now[i]]++;

}

}

for (int i = 1; i <= res[0]; i++)

cnt[res[i]] = mi[res[i]] = 0;

}

void Solve(int x) {

vis[x] = true;

Calc(x);

for (auto it : G[x]) {

int v = it.first;

if (vis[v])

continue;

scnt = sz[v], mx = INF;

DFS\_Mid(v, 0);

Solve(rt);

}

}

int main()

{

cin >> n >> k;

for (int i = 1; i < n; i++) {

int u, v, w;

sc("%d %d %d", &u, &v, &w);

G[u + 1].push\_back({v + 1, w});

G[v + 1].push\_back({u + 1, w});

}

scnt = n, mx = INF;

DFS\_Mid(1, 0);

Solve(rt);

if (ans == INF)

puts("-1");

else

printf("%d\n", ans);

return 0; // 改数组大小!!!用pair改宏定义!!!

}

### 4.距离<=k的路径数

int s[N];

void Add(int x, int v) {

x++;

while (x < N) {

s[x] += v;

x = (x | (x - 1)) + 1;

}

}

int Ask(int x) {

x++;

int tot = 0;

while (x > 0) {

tot += s[x];

x &= x - 1;

}

return tot;

}

int Rge\_Ask(int l, int r) {

return Ask(r) - Ask(l - 1);

}

vector <pir> G[N];

int mxt[N], sz[N], n, q;

int rt, snum, mx, k;

bool vis[N];

ll ans;

void DFS\_Mid(int x, int fa) { // 子树重心

mxt[x] = 0, sz[x] = 1;

for (auto it : G[x]) {

int v = it.first;

if (vis[v] || v == fa)

continue;

DFS\_Mid(v, x);

sz[x] += sz[v];

Max(mxt[x], sz[v]);

}

Max(mxt[x], snum - sz[x]);

if (mxt[x] < mx)

mx = mxt[x], rt = x;

}

int now[N], res[N];

void DFS(int x, int fa, int dist) {

if (dist > k)

return;

ans++;

now[++now[0]] = dist;

for (auto it : G[x]) {

int v = it.first, w = it.second;

if (v == fa || vis[v])

continue;

DFS(v, x, dist + w);

}

}

void Calc(int x) {

res[0] = 0;

for (auto it : G[x]) {

int v = it.first, w = it.second;

if (vis[v])

continue;

now[0] = 0;

DFS(v, 0, w);

for (int i = 1; i <= now[0]; i++)

ans += Rge\_Ask(0, k - now[i]);

for (int i = 1; i <= now[0]; i++)

Add(now[i], 1), res[++res[0]] = now[i];

}

for (int i = 1; i <= res[0]; i++)

Add(res[i], -1);

}

void Solve(int x) {

vis[x] = true;

Calc(x);

for (auto it : G[x]) {

int v = it.first;

if (vis[v])

continue;

snum = sz[v], mx = INF;

DFS\_Mid(v, 0);

Solve(rt);

}

}

int main()

{

cin >> n;

for (int i = 1; i < n; i++) {

int u, v, w;

sc("%d %d %d", &u, &v, &w);

G[u].push\_back({ v, w });

G[v].push\_back({ u, w });

}

cin >> k;

snum = n, mx = INF;

DFS\_Mid(1, 0);

Solve(rt);

printf("%lld\n", ans);

return 0; // 改数组大小!!!用pair改宏定义!!!

}

### 5.模3为1，2，3的路径和

vector <pir> G[N];

int mxt[N], sz[N], n;

int rt, scnt, mx;

bool vis[N];

ll ans[3];

void Init() {

for (int i = 1; i <= n; i++) {

G[i].clear();

vis[i] = false;

}

for (int i = 0; i < 3; i++)

ans[i] = 0;

}

void DFS\_Mid(int x, int fa) {

mxt[x] = 0, sz[x] = 1;

for (auto it : G[x]) {

int v = it.first;

if (vis[v] || v == fa)

continue;

DFS\_Mid(v, x);

sz[x] += sz[v];

Max(mxt[x], sz[v]);

}

Max(mxt[x], scnt - sz[x]);

if (mxt[x] < mx)

mx = mxt[x], rt = x;

}

ll now[N];

ll cnt[3], val[3];

void DFS(int x, int fa, ll dist) {

now[++now[0]] = dist;

for (auto it : G[x]) {

int v = it.first; ll w = it.second;

if (v != fa && !vis[v])

DFS(v, x, dist + w);

}

}

void Calc(int x) {

cnt[0] = 1;

for (auto it : G[x]) {

int v = it.first; ll w = it.second;

if (vis[v])

continue;

now[0] = 0;

DFS(v, 0, w);

for (int i = 0; i < 3; i++) {

if (!cnt[i])

continue;

for (int j = 1; j <= now[0]; j++) {

ll d = now[j] % 3;

ll nxt = (d + i) % 3; // 合并两条链

ans[nxt] += (val[i] + cnt[i] \* now[j] % Mod) % Mod;

ans[nxt] %= Mod;

}

}

for (int i = 1; i <= now[0]; i++) {

cnt[now[i] % 3]++;

val[now[i] % 3] = (val[now[i] % 3] + now[i]) % Mod;

}

}

for (int i = 0; i < 3; i++)

cnt[i] = val[i] = 0;

}

void Solve(int x) {

vis[x] = true;

Calc(x);

for (auto it : G[x]) {

int v = it.first;

if (vis[v])

continue;

scnt = sz[v], mx = INF;

DFS\_Mid(v, 0);

Solve(rt);

}

}

int main()

{

while (~sc("%d", &n)) {

Init();

for (int i = 1; i < n; i++) {

int u, v; ll w;

sc("%d %d %lld", &u, &v, &w);

u++, v++;

G[u].push\_back({ v, w });

G[v].push\_back({ u, w });

}

scnt = n, mx = INF;

DFS\_Mid(1, 0);

Solve(rt);

for (int i = 0; i < 3; i++)

printf("%lld%c", 2ll \* ans[i] % Mod, i == 2 ? '\n' : ' ');

}

return 0; // 改数组大小!!!用pair改宏定义!!!

}

## 树链剖分

### 模板

// 存图 N \* log(N) \* log(N)

vector <int> G[N];

int fz[N], son[N], sz[N];

int idx[N], dfn;

int dep[N], top[N];

int n, q;

void DFS1(int x, int fa, int dist) {

fz[x] = fa; // 父亲

dep[x] = dist; //深度

sz[x] = 1; // 子树大小

int mx = 0; // 重儿子个数

for (auto v : G[x]) {

if (v == fa)

continue;

DFS1(v, x, dist + 1);

sz[x] += sz[v];

if (sz[v] > mx)

mx = sz[v], son[x] = v;

}

}

void DFS2(int x, int topfz) {

idx[x] = ++dfn; // 重新编号

top[x] = topfz;

if (!son[x])

return;

DFS2(son[x], topfz);

for (auto v : G[x]) {

if (v != fz[x] && v != son[x])

DFS2(v, v);

}

}

// 线段树

void Range\_Update(int x, int y, ll w) {

while (top[x] != top[y]) {

if (dep[top[x]] < dep[top[y]])

swap(x, y);

Update(1, 1, n, idx[top[x]], idx[x], w);

x = fz[top[x]];

}

if (dep[x] > dep[y])

swap(x, y);

Update(1, 1, n, idx[x], idx[y], w);

}

### 题集:

#### 1.子树和路径修改

// 存图

vector <int> G[N];

int fz[N], son[N], sz[N];

int idx[N], ev[N], dfn;

int dep[N], top[N];

int n, q;

void dfs1(int x, int fa, int dist) {

fz[x] = fa; // 父亲

dep[x] = dist; //深度

sz[x] = 1; // 子树大小

int mx = 0; // 重儿子个数

for (auto v : G[x]) {

if (v == fa)

continue;

dfs1(v, x, dist + 1);

sz[x] += sz[v];

if (sz[v] > mx)

mx = sz[v], son[x] = v;

}

}

void dfs2(int x, int topfz) {

idx[x] = ++dfn; // 重新编号

top[x] = topfz;

if (!son[x])

return;

dfs2(son[x], topfz);

for (auto v : G[x]) {

if (v != fz[x] && v != son[x])

dfs2(v, v);

}

}

// 线段树

int lzy[N \* 4], wi[N \* 4]; // 标记，权值

void Build(int rt, int l, int r) {

lzy[rt] = -1;

if (l == r)

return;

int mid = (l + r) >> 1;

Build(ls, l, mid);

Build(rs, mid + 1, r);

}

void Pushdown(int rt, int len) {

lzy[ls] = lzy[rt];

lzy[rs] = lzy[rt];

wi[ls] = lzy[rt] \* (len - (len >> 1));

wi[rs] = lzy[rt] \* (len >> 1);

lzy[rt] = -1;

}

void Update(int rt, int L, int R, int l, int r, int w) {

if (L >= l && R <= r) {

wi[rt] = (R - L + 1) \* w;

lzy[rt] = w;

return;

}

if (lzy[rt] != -1)

Pushdown(rt, R - L + 1);

int mid = (L + R) >> 1;

if (mid >= l)

Update(ls, L, mid, l, r, w);

if (mid < r)

Update(rs, mid + 1, R, l, r, w);

wi[rt] = wi[ls] + wi[rs];

}

void Range\_Update(int x, int y, int w) {

while (top[x] != top[y]) {

if (dep[top[x]] < dep[top[y]])

swap(x, y);

Update(1, 1, n, idx[top[x]], idx[x], w);

x = fz[top[x]];

}

if (dep[x] > dep[y])

swap(x, y);

Update(1, 1, n, idx[x], idx[y], w);

}

int Query(int rt, int L, int R, int x) {

if (L == x && R == x)

return wi[rt];

if (lzy[rt] != -1)

Pushdown(rt, R - L + 1);

int mid = (L + R) >> 1;

if (mid >= x)

return Query(ls, L, mid, x);

else

return Query(rs, mid + 1, R, x);

}

int main()

{

cin >> n;

for (int i = 1; i < n; i++) {

int u, v;

sc("%d %d", &u, &v);

G[u].push\_back(v);

G[v].push\_back(u);

}

dfs1(1, 0, 1);

dfs2(1, 1);

Build(1, 1, n);

cin >> q;

while (q--) {

int op, x;

sc("%d %d", &op, &x);

if (op == 1)

Update(1, 1, n, idx[x], idx[x] + sz[x] - 1, 1);

else if (op == 2)

Range\_Update(1, x, 0);

else

printf("%d\n", Query(1, 1, n, idx[x]));

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 2.区间覆盖区间最大值

// 存图

vector <pir> G[N << 1];

int fz[N], son[N], sz[N];

int idx[N], dfn;

int dep[N], top[N];

int wt[N], ev[N], n, q;

pir a[N];

char op[10];

void dfs1(int x, int fa, int dist) {

fz[x] = fa; // 父亲

dep[x] = dist; //深度

sz[x] = 1; // 子树大小

int mx = 0; // 重儿子个数

for (auto it : G[x]) {

int v = it.second;

int w = it.first;

if (v == fa)

continue;

wt[v] = w;

dfs1(v, x, dist + 1);

sz[x] += sz[v];

if (sz[v] > mx)

mx = sz[v], son[x] = v;

}

}

void dfs2(int x, int topfz) {

idx[x] = ++dfn; // 重新编号

top[x] = topfz;

ev[dfn] = wt[x];

if (!son[x])

return;

dfs2(son[x], topfz);

for (auto it : G[x]) {

int v = it.second;

if (v != fz[x] && v != son[x])

dfs2(v, v);

}

}

// 线段树

int mx[N \* 4];

int lzy[N \* 4], add[N \* 4]; // 覆盖，加

void Build(int rt, int L, int R) {

lzy[rt] = -1;

if (L == R) {

mx[rt] = ev[L];

return;

}

int mid = (L + R) >> 1;

Build(ls, L, mid);

Build(rs, mid + 1, R);

mx[rt] = max(mx[ls], mx[rs]);

}

void Pushdown(int rt) {

if (lzy[rt] != -1) {

add[ls] = add[rs] = 0;

mx[ls] = mx[rs] = lzy[rt];

lzy[ls] = lzy[rs] = lzy[rt];

lzy[rt] = -1;

}

if (add[rt]) {

mx[ls] += add[rt], mx[rs] += add[rt];

add[ls] += add[rt], add[rs] += add[rt];

add[rt] = 0;

}

}

void Update(int rt, int L, int R, int l, int r, int w) {

if (L >= l && R <= r) {

mx[rt] = w;

lzy[rt] = w, add[rt] = 0;

return;

}

Pushdown(rt);

int mid = (L + R) >> 1;

if (mid >= l)

Update(ls, L, mid, l, r, w);

if (mid < r)

Update(rs, mid + 1, R, l, r, w);

mx[rt] = max(mx[ls], mx[rs]);

}

void Add(int rt, int L, int R, int l, int r, int w) {

if (L >= l && R <= r) {

mx[rt] += w;

add[rt] += w;

return;

}

Pushdown(rt);

int mid = (L + R) >> 1;

if (mid >= l)

Add(ls, L, mid, l, r, w);

if (mid < r)

Add(rs, mid + 1, R, l, r, w);

mx[rt] = max(mx[ls], mx[rs]);

}

int Query(int rt, int L, int R, int l, int r) {

if (L >= l && R <= r)

return mx[rt];

Pushdown(rt);

int mid = (L + R) >> 1;

int tot = 0;

if (mid >= l)

Max(tot, Query(ls, L, mid, l, r));

if (mid < r)

Max(tot, Query(rs, mid + 1, R, l, r));

return tot;

}

void Range\_Update(int x, int y, int w) {

while (top[x] != top[y]) {

if (dep[top[x]] < dep[top[y]])

swap(x, y);

Update(1, 1, n, idx[top[x]], idx[x], w);

x = fz[top[x]];

}

if (dep[x] > dep[y])

swap(x, y);

Update(1, 1, n, idx[x] + 1, idx[y], w);

}

void Range\_Add(int x, int y, int w) {

while (top[x] != top[y]) {

if (dep[top[x]] < dep[top[y]])

swap(x, y);

Add(1, 1, n, idx[top[x]], idx[x], w);

x = fz[top[x]];

}

if (dep[x] > dep[y])

swap(x, y);

Add(1, 1, n, idx[x] + 1, idx[y], w);

}

int Range\_Query(int x, int y) {

int tot = 0;

while (top[x] != top[y]) {

if (dep[top[x]] < dep[top[y]])

swap(x, y);

Max(tot, Query(1, 1, n, idx[top[x]], idx[x]));

x = fz[top[x]];

}

if (dep[x] > dep[y])

swap(x, y);

Max(tot, Query(1, 1, n, idx[x] + 1, idx[y]));

return tot;

}

int main()

{

cin >> n;

for (int i = 1; i < n; i++) {

int u, v, w;

sc("%d %d %d", &u, &v, &w);

G[u].push\_back({ w, v });

G[v].push\_back({ w, u });

a[i] = { u, v };

}

dfs1(1, 0, 1);

dfs2(1, 1);

Build(1, 1, n);

while (sc("%s", op) && op[0] != 'S') {

int x, y, w;

if (op[1] == 'h') {

sc("%d %d", &x, &w);

int u = a[x].first;

int v = a[x].second;

if (dep[u] < dep[v])

swap(u, v);

Update(1, 1, n, idx[u], idx[u], w);

}

else if (op[1] == 'o') {

sc("%d %d %d", &x, &y, &w);

Range\_Update(x, y, w);

}

else if (op[0] == 'A') {

sc("%d %d %d", &x, &y, &w);

Range\_Add(x, y, w);

}

else {

sc("%d %d", &x, &y);

printf("%d\n", Range\_Query(x, y));

}

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

## 平衡树

### 1.FHQ平衡树

mt19937 mt(time(NULL)); // mt()

typedef unsigned int uint;

struct Treap

{

int ls, rs;

uint key;

int val, sz;

}t[N];

int rt, idx;

void Init() {

rt = idx = 0;

}

int Create(int x) {

idx++;

t[idx] = { 0, 0, mt(), x, 1 };

return idx;

}

void Pushup(int u) {

t[u].sz = t[t[u].ls].sz + t[t[u].rs].sz + 1;

}

void Split(int u, int val, int &l, int &r) {

if (!u)

l = r = 0;

else {

if (t[u].val <= val) {

l = u;

Split(t[u].rs, val, t[u].rs, r);

}

else {

r = u;

Split(t[u].ls, val, l, t[u].ls);

}

Pushup(u);

}

}

int Merge(int l, int r) {

if (!l || !r)

return l | r;

if (t[l].key > t[r].key) {

t[l].rs = Merge(t[l].rs, r);

Pushup(l); return l;

}

else {

t[r].ls = Merge(l, t[r].ls);

Pushup(r); return r;

}

}

int L, R, mid;

void Insert(int val) {

Split(rt, val, L, R);

rt = Merge(Merge(L, Create(val)), R);

}

void Delete(int val) {

Split(rt, val, L, mid);

Split(L, val - 1, L, R);

R = Merge(t[R].ls, t[R].rs);

rt = Merge(Merge(L, R), mid);

}

int Rank(int val) {

Split(rt, val - 1, L, R);

int k = t[L].sz + 1;

rt = Merge(L, R);

return k;

}

int Kth(int k) {

int x = rt;

while (x && t[t[x].ls].sz + 1 != k)

if (t[t[x].ls].sz >= k)

x = t[x].ls;

else

k -= t[t[x].ls].sz + 1, x = t[x].rs;

return t[x].val;

}

int Pre(int val) {

Split(rt, val - 1, L, R);

int x = L;

while (t[x].rs)

x = t[x].rs;

rt = Merge(L, R);

return t[x].val;

}

int Nxt(int val) {

Split(rt, val, L, R);

int x = R;

while (t[x].ls)

x = t[x].ls;

rt = Merge(L, R);

return t[x].val;

}

## 虚树

### 模板

// 查询一些关键点，问断开边最小值使得所有点无法到达根

vector <pir> G[N];

int dep[N], f[N][25];

int din[N], dou[N], cnt;

int tr[N], st[N], idx;

int mi[N];

ll sum[N];

bool vis[N];

void DFS(int x, int fa) {

dep[x] = dep[fa] + 1;

f[x][0] = fa;

din[x] = ++cnt;

for (int i = 1; (1 << i) <= dep[x]; i++)

f[x][i] = f[f[x][i - 1]][i - 1];

for (auto it : G[x]) {

int v = it.second;

int w = it.first;

if (v != fa) {

mi[v] = min(w, mi[x]);

DFS(v, x);

}

}

dou[x] = ++cnt;

}

int LCA(int x, int y) {

if (dep[x] > dep[y])

swap(x, y);

for (int i = 20; i >= 0; i--) {

if (dep[y] - (1 << i) >= dep[x])

y = f[y][i];

}

if (x == y)

return x;

for (int i = 20; i >= 0; i--) {

if (f[x][i] == f[y][i])

continue;

x = f[x][i], y = f[y][i];

}

return f[x][0];

}

bool cmp(int a, int b) { // 欧拉序排序

int k1 = a > 0 ? din[a] : dou[-a];

int k2 = b > 0 ? din[b] : dou[-b];

return k1 < k2;

}

void Calc(int k) {

sort(tr + 1, tr + k + 1, cmp); // 关键点排序

// 把LCA加入

for (int i = 1; i < k; i++) {

int lca = LCA(tr[i], tr[i + 1]);

if (!vis[lca])

vis[lca] = true, tr[++k] = lca;

}

int nc = k;

for (int i = 1; i <= nc; i++)

tr[++k] = -tr[i];

if (!vis[1])

tr[++k] = 1, tr[++k] = -1;

sort(tr + 1, tr + k + 1, cmp); // 加入1号点

for (int i = 1; i <= k; i++) {

if (tr[i] > 0)

st[++idx] = tr[i];

else {

int u = st[idx--];

if (u == 1)

printf("%lld\n", sum[1]);

else

sum[st[idx]] += min(sum[u], (ll)mi[u]);

vis[u] = false;

sum[u] = 0;

}

}

}

mi[1] = INF;

DFS(1, 0); // 预处理LCA和DFS序

## 单调栈

### 模板

#### 1.右边第一个比他大的数下标

int a[N], ans[N];

int main()

{

int n;

cin >> n;

stack <int> st;

for (int i = 1; i <= n; i++)

sc("%d", &a[i]);

for (int i = 1; i <= n; i++) {

while (!st.empty() && a[i] > a[st.top()]) {

ans[st.top()] = i;

st.pop();

}

st.push(i);

}

for (int i = 1; i <= n; i++)

cout << ans[i] << ' ';

cout << endl;

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

### 题集

#### 1.最大全1子矩阵

int main()

{

while (~sc("%d %d", &n, &m)) {

Init();

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= m; j++) {

sc("%d", &a[i][j]);

if (a[i][j])

col[i][j] += col[i - 1][j] + 1;

}

col[i][m + 1] = -INF;

}

int ans = 0;

for (int i = 1; i <= n; i++) {

s = 0;

MEM(L, 0), st[++s] = 1, L[1] = 1;

for (int j = 1; j <= m + 1; j++) {

if (col[i][j] >= col[i][st[s]])

L[j] = j, st[++s] = j;

else {

int l;

while (s && col[i][j] < col[i][st[s]]) {

int tot = (j - L[st[s]]) \* col[i][st[s]];

Max(ans, tot);

l = L[st[s--]];

}

L[j] = l, st[++s] = j;

}

}

}

printf("%d\n", ans);

}

return 0; // 改数组大小!!!用pair改宏定义!!!

}

#### 2.字典序最小的排列

int a[N], st[N], s;

int cnt[N];

bool vis[N];

int main()

{

int n, k;

cin >> n >> k;

for (int i = 1; i <= n; i++)

sc("%d", &a[i]), cnt[a[i]]++;

st[++s] = a[1], cnt[a[1]]--;

vis[a[1]] = true;

for (int i = 2; i <= n; i++) {

cnt[a[i]]--;

if (vis[a[i]])

continue;

while (s && a[i] < st[s] && cnt[st[s]])

vis[st[s--]] = false;

st[++s] = a[i];

vis[a[i]] = true;

}

for (int i = 1; i < k; i++)

printf("%d ", st[i]);

printf("%d\n", st[k]);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 3.黑格子不可走子矩阵个数

int a[N][N], S[N], top;

int h[N];

ll dp[N];

int main()

{

int n, m;

sc("%d %d", &n, &m);

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= m; j++) {

sc("%d", &a[i][j]);

}

}

ll ans = 0;

for (int i = 1; i <= n; i++) {

top = 0; // 清空栈

for (int j = 1; j <= m; j++) {

if (!a[i][j])

h[j] = i; // 当前列最高的

while (top && h[S[top]] < h[j])

top--;

S[++top] = j; // 矩阵面积

dp[top] = dp[top - 1] + (ll)(i - h[S[top]]) \* (ll)(j - S[top - 1]);

ans += dp[top];

}

}

printf("%lld\n", ans);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

## 单调队列

### 1.区间最大&最小值

int a[N], q1[N], q2[N];

int n, k;

void Min\_que() {

int l = 1, r = 0;

for (int i = 1; i <= n; i++) {

while (l <= r && q1[l] + k <= i)

l++;

while (l <= r && a[i] < a[q1[r]])

r--;

q1[++r] = i;

if (i >= k)

printf("%d ", a[q1[l]]);

}

putchar('\n');

}

void Max\_que() {

int l = 1, r = 0;

for (int i = 1; i <= n; i++) {

while (l <= r && q2[l] + k <= i)

l++;

while (l <= r && a[i] > a[q2[r]])

r--;

q2[++r] = i;

if (i >= k)

printf("%d ", a[q2[l]]);

}

putchar('\n');

}

int main()

{

cin >> n >> k;

for (int i = 1; i <= n; i++)

sc("%d", &a[i]);

Min\_que();

Max\_que();

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

### 2.最大最小差值为k的区间数

int q1[N], q2[N];

ll a[N];

int n, k, b, c;

int main()

{

cin >> n >> k;

cin >> a[0] >> b >> c;

for (int i = 1; i <= n; i++)

a[i] = (a[i - 1] \* (ll)b + c) % Mod;

ll ans = 0;

int L = 1;

int l1 = 1, l2 = 1;

int r1 = 0, r2 = 0;

for (int i = 1; i <= n; i++) {

while (l1 <= r1 && a[i] <= a[q1[r1]])

r1--;

while (l2 <= r2 && a[i] >= a[q2[r2]])

r2--;

q1[++r1] = i, q2[++r2] = i;

while (a[q2[l2]] - a[q1[l1]] > k) {

ans += n - i + 1;

L++;

if (q1[l1] < L)

l1++;

if (q2[l2] < L)

l2++;

}

}

printf("%lld\n", ans);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

## 主席树

### 模板

#### 1.区间第K大

vector <int> v;

int ans[N \* 32], ls[N \* 32], rs[N \* 32];

int a[N], n, q;

int root[N], cnt;

void Update(int L, int R, int &rt, int lst, int w) {

ans[++cnt] = ans[lst] + 1;

ls[cnt] = ls[lst], rs[cnt] = rs[lst];

rt = cnt;

if (L == R)

return;

int mid = (L + R) >> 1;

if (mid >= w)

Update(L, mid, ls[rt], ls[lst], w);

else

Update(mid + 1, R, rs[rt], rs[lst], w);

}

int Ask(int L, int R, int l, int r, int k) {

if (L == R)

return L;

int sum = ans[ls[r]] - ans[ls[l]];

int mid = (L + R) >> 1;

if (sum >= k)

return Ask(L, mid, ls[l], ls[r], k);

else

return Ask(mid + 1, R, rs[l], rs[r], k - sum);

}

int main()

{

cin >> n >> q;

for (int i = 1; i <= n; i++)

sc("%d", &a[i]), v.push\_back(a[i]);

sort(ALL(v));

v.erase(unique(ALL(v)), v.end());

for (int i = 1; i <= n; i++) {

int ev = lower\_bound(ALL(v), a[i]) - v.begin() + 1;

Update(1, n, root[i], root[i - 1], ev);

}

while (q--) {

int l, r, k;

sc("%d %d %d", &l, &r, &k);

printf("%d\n", v[Ask(1, n, root[l - 1], root[r], k) - 1]);

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

### 题集

#### 1.区间不同数个数

struct Tree

{

ll ans;

int l, r;

}t[N \* 25];

vector <int> v;

int n, q;

int root[N], cnt;

int vis[N], pre[N];

void Update(int L, int R, int &rt, int lst, int w) {

t[++cnt] = t[lst];

t[cnt].ans++;

rt = cnt;

if (L == R)

return;

int mid = (L + R) >> 1;

if (mid >= w)

Update(L, mid, t[rt].l, t[lst].l, w);

else

Update(mid + 1, R, t[rt].r, t[lst].r, w);

}

int Ask(int L, int R, int l, int r, int al, int ar) {

if (L >= al && R <= ar)

return t[r].ans - t[l].ans;

int mid = (L + R) >> 1, tot = 0;

if (mid >= al)

tot = Ask(L, mid, t[l].l, t[r].l, al, ar);

if (mid < ar)

tot += Ask(mid + 1, R, t[l].r, t[r].r, al, ar);

return tot;

}

int main()

{

cin >> n;

for (int i = 1; i <= n; i++) {

int x;

sc("%d", &x);

if (vis[x])

pre[i] = vis[x];

vis[x] = i;

}

for (int i = 1; i <= n; i++)

Update(0, n, root[i], root[i - 1], pre[i]);

cin >> q;

while (q--) {

int l, r;

sc("%d %d", &l, &r);

printf("%d\n", Ask(0, n, root[l - 1], root[r], 0, l - 1));

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 2.区间与和的个数

map <int, int> mp, tmp, las;

int ans[N \* 680], ls[N \* 680], rs[N \* 680];

int root[N], cnt;

int a[N], n, q;

void Update(int L, int R, int &rt, int lst, int w, int c) {

rt = ++cnt;

ans[rt] = ans[lst] + c;

ls[rt] = ls[lst], rs[rt] = rs[lst];

if (L == R)

return;

int mid = (L + R) >> 1;

if (mid >= w)

Update(L, mid, ls[rt], ls[lst], w, c);

else

Update(mid + 1, R, rs[rt], rs[lst], w, c);

}

int Ask(int L, int R, int l, int r, int al, int ar) {

if (L >= al && R <= ar)

return ans[r] - ans[l];

int mid = (L + R) >> 1, tot = 0;

if (mid >= al)

tot = Ask(L, mid, ls[l], ls[r], al, ar);

if (mid < ar)

tot += Ask(mid + 1, R, rs[l], rs[r], al, ar);

return tot;

}

int main()

{

cin >> n;

for (int i = 1; i <= n; i++) {

int x;

sc("%d", &x);

mp[x] = i, tmp.clear();

for (auto it : mp)

Max(tmp[it.first & x], it.second);

root[i] = root[i - 1];

for (auto it : tmp) {

int v = it.first, pos = it.second;

if (las[v])

Update(1, n, root[i], root[i], las[v], -1);

Update(1, n, root[i], root[i], pos, 1);

las[v] = pos;

}

swap(mp, tmp);

}

int lans = 0;

cin >> q;

while (q--) {

int l, r;

sc("%d %d", &l, &r);

l = (l ^ lans) % n + 1;

r = (r ^ lans) % n + 1;

if (l > r)

swap(l, r);

lans = Ask(1, n, root[l - 1], root[r], l, r);

printf("%d\n", lans);

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

## 线段树&树状数组

### 模板

#### 1.树状数组单点修改区间查询

// 下标从0开始Add和Ask函数进去x+1即可

int s[N], n, q;

void Add(int x, int v) {

while (x <= n) {

s[x] += v;

x = (x | (x - 1)) + 1;

}

}

int Ask(int x) {

int tot = 0;

while (x > 0) {

tot += s[x];

x &= x - 1;

}

return tot;

}

int Rge\_Ask(int l, int r) {

return Ask(r) - Ask(l - 1);

}

#### 2.树状数组区间修改区间查询

// 树状数组

ll d[N], di[N];

void Add(int x, ll y) {

for (int i = x; i <= n; i += lowbit(i))

d[i] += y, di[i] += y \* x;

}

void Range\_Add(int l, int r, ll w) {

Add(l, w);

Add(r + 1, -w);

}

ll Ask(int x) {

ll tot = 0;

for (int i = x; i; i -= lowbit(i))

tot += (x + 1) \* d[i] - di[i];

return tot;

}

ll Range\_Ask(int l, int r) {

return Ask(r) - Ask(l - 1);

}

#### 3.线段树区间加修改区间查询

#define ls rt << 1

#define rs ls | 1

ll ans[N \* 4], lzy[N \* 4];

int n, m;

void Pushup(int rt) {

ans[rt] = ans[ls] + ans[rs];

}

void Build(int rt, int L, int R) {

if (L == R) {

int w;

sc("%d", &w);

ans[rt] = w;

}

else {

int mid = (L + R) >> 1;

Build(ls, L, mid);

Build(rs, mid + 1, R);

Pushup(rt);

}

}

void Pushdown(int rt, int len) {

int Llen = len - (len >> 1);

int Rlen = len >> 1;

lzy[ls] += lzy[rt], lzy[rs] += lzy[rt];

ans[ls] += lzy[rt] \* Llen, ans[rs] += lzy[rt] \* Rlen;

lzy[rt] = 0;

}

void Update(int rt, int L, int R, int l, int r, ll k) {

if (L >= l && R <= r) {

ans[rt] += 1ll \* (R - L + 1) \* k;

lzy[rt] += k;

}

else {

if (lzy[rt])

Pushdown(rt, R - L + 1);

int mid = (L + R) >> 1;

if (mid >= l)

Update(ls, L, mid, l, r, k);

if (mid < r)

Update(rs, mid + 1, R, l, r, k);

Pushup(rt);

}

}

ll Ask(int rt, int L, int R, int l, int r) {

if (L >= l && R <= r)

return ans[rt];

if (lzy[rt])

Pushdown(rt, R - L + 1);

int mid = (L + R) >> 1;

ll tot = 0;

if (mid >= l)

tot += Ask(ls, L, mid, l, r);

if (mid < r)

tot += Ask(rs, mid + 1, R, l, r);

return tot;

}

#### 4.线段树区间加&乘

#define ls rt << 1

#define rs ls | 1

ll sum[N \* 4], add[N \* 4], mul[N \* 4];

int n, m, P;

void Pushup(int rt) {

sum[rt] = (sum[ls] + sum[rs]) % P;

}

void Build(int rt, int L, int R) {

add[rt] = 0, mul[rt] = 1;

if (L == R) {

ll w;

sc("%lld", &w);

sum[rt] = w % P;

}

else {

int mid = (L + R) >> 1;

Build(ls, L, mid);

Build(rs, mid + 1, R);

Pushup(rt);

}

}

void Pushdown(int rt, int len) {

int Llen = len - (len >> 1);

int Rlen = len >> 1;

sum[ls] = ((sum[ls] \* mul[rt]) % P + (add[rt] \* Llen) % P) % P;

sum[rs] = ((sum[rs] \* mul[rt]) % P + (add[rt] \* Rlen) % P) % P;

add[ls] = ((add[ls] \* mul[rt]) % P + add[rt]) % P;

add[rs] = ((add[rs] \* mul[rt]) % P + add[rt]) % P;

mul[ls] = (mul[ls] \* mul[rt]) % P;

mul[rs] = (mul[rs] \* mul[rt]) % P;

mul[rt] = 1, add[rt] = 0;

}

void Mul(int rt, int L, int R, int l, int r, ll k) {

if (L >= l && R <= r) {

sum[rt] = (sum[rt] \* k) % P;

mul[rt] = (mul[rt] \* k) % P;

add[rt] = (add[rt] \* k) % P;

}

else {

Pushdown(rt, R - L + 1);

int mid = (L + R) >> 1;

if (mid >= l)

Mul(ls, L, mid, l, r, k);

if (mid < r)

Mul(rs, mid + 1, R, l, r, k);

Pushup(rt);

}

}

void Add(int rt, int L, int R, int l, int r, ll k) {

int len = (R - L) + 1;

if (L >= l && R <= r) {

sum[rt] = (sum[rt] + len \* k % P) % P;

add[rt] = (add[rt] + k) % P;

}

else {

Pushdown(rt, len);

int mid = (L + R) >> 1;

if (mid >= l)

Add(ls, L, mid, l, r, k);

if (mid < r)

Add(rs, mid + 1, R, l, r, k);

Pushup(rt);

}

}

ll Ask(int rt, int L, int R, int l, int r) {

if (L >= l && R <= r)

return sum[rt];

Pushdown(rt, R - L + 1);

int mid = (L + R) >> 1;

ll tot = 0;

if (mid >= l)

tot = Ask(ls, L, mid, l, r);

if (mid < r)

tot = (tot + Ask(rs, mid + 1, R, l, r)) % P;

return tot;

}

#### 5.区间01反转查询最长0，1子段和

#define ls rt << 1

#define rs ls | 1

struct Tree

{

int l, r, len, lzy;

int lmx, rmx, mx;

int lmx1, rmx1, mx1;

}t[N \* 4];

int a[N];

char str[N];

void Pushup(int rt) {

t[rt].lmx1 = t[ls].lmx1, t[rt].rmx1 = t[rs].rmx1;

if (t[ls].mx1 == t[ls].len)

Max(t[rt].lmx1, t[ls].mx1 + t[rs].lmx1);

if (t[rs].mx1 == t[rs].len)

Max(t[rt].rmx1, t[rs].mx1 + t[ls].rmx1);

t[rt].mx1 = max(t[rt].lmx1, t[rt].rmx1);

Max(t[rt].mx1, t[ls].rmx1 + t[rs].lmx1);

Max(t[rt].mx1, max(t[ls].mx1, t[rs].mx1));

t[rt].lmx = t[ls].lmx, t[rt].rmx = t[rs].rmx;

if (t[ls].mx == t[ls].len)

Max(t[rt].lmx, t[ls].mx + t[rs].lmx);

if (t[rs].mx == t[rs].len)

Max(t[rt].rmx, t[rs].mx + t[ls].rmx);

t[rt].mx = max(t[rt].lmx, t[rt].rmx);

Max(t[rt].mx, t[ls].rmx + t[rs].lmx);

Max(t[rt].mx, max(t[ls].mx, t[rs].mx));

}

void Build(int rt, int l, int r) {

t[rt].l = l, t[rt].r = r;

t[rt].lzy = 0;

t[rt].len = r - l + 1;

t[rt].lmx = t[rt].rmx = t[rt].mx = 0;

t[rt].lmx1 = t[rt].rmx1 = t[rt].mx1 = 0;

if (l == r) {

if (a[l] == 1)

t[rt].lmx1 = t[rt].rmx1 = t[rt].mx1 = 1;

else

t[rt].lmx = t[rt].rmx = t[rt].mx = 1;

return;

}

int mid = (l + r) >> 1;

Build(ls, l, mid);

Build(rs, mid + 1, r);

Pushup(rt);

}

void Calc(int rt) {

swap(t[rt].lmx, t[rt].lmx1);

swap(t[rt].rmx, t[rt].rmx1);

swap(t[rt].mx, t[rt].mx1);

}

void Pushdown(int rt) {

t[ls].lzy ^= 1;

swap(t[ls].lmx, t[ls].lmx1);

swap(t[ls].rmx, t[ls].rmx1);

swap(t[ls].mx, t[ls].mx1);

t[rs].lzy ^= 1;

swap(t[rs].lmx, t[rs].lmx1);

swap(t[rs].rmx, t[rs].rmx1);

swap(t[rs].mx, t[rs].mx1);

t[rt].lzy ^= 1;

}

void Update(int rt, int l, int r) {

if (t[rt].l >= l && t[rt].r <= r) {

Calc(rt); t[rt].lzy ^= 1;

return;

}

int mid = (t[rt].l + t[rt].r) >> 1;

if (t[rt].lzy)

Pushdown(rt);

if (mid >= l)

Update(ls, l, r);

if (mid < r)

Update(rs, l, r);

Pushup(rt);

}

#### 6.区间异或线段树

// 按位取反即可

int wt[21][N \* 4], a[N];

bool lzy[21][N \* 4];

void Pushup(int rt) {

for (int i = 0; i <= 20; i++)

wt[i][rt] = wt[i][ls] + wt[i][rs];

}

void Build(int rt, int L, int R) {

if (L == R) {

for (int i = 0; i <= 20; i++) {

if ((a[L] >> i) & 1)

wt[i][rt] = 1;

}

return;

}

int mid = (L + R) >> 1;

Build(ls, L, mid);

Build(rs, mid + 1, R);

Pushup(rt);

}

void Pushdown(int rt, int Len) {

int Llen = Len - (Len >> 1);

int Rlen = Len >> 1;

for (int i = 0; i <= 20; i++) {

if (lzy[i][rt]) {

wt[i][ls] = Llen - wt[i][ls];

wt[i][rs] = Rlen - wt[i][rs];

lzy[i][ls] = !lzy[i][ls];

lzy[i][rs] = !lzy[i][rs];

lzy[i][rt] = false;

}

}

}

void Update(int rt, int L, int R, int l, int r, int w) {

int Len = R - L + 1;

if (L >= l && R <= r) {

for (int i = 0; i <= 20; i++) {

if ((w >> i) & 1) {

wt[i][rt] = Len - wt[i][rt];

lzy[i][rt] = !lzy[i][rt];

}

}

return;

}

Pushdown(rt, Len);

int mid = (L + R) >> 1;

if (mid >= l)

Update(ls, L, mid, l, r, w);

if (mid < r)

Update(rs, mid + 1, R, l, r, w);

Pushup(rt);

}

ll Ask(int rt, int L, int R, int l, int r) {

if (L >= l && R <= r) {

ll sum = 0;

for (int i = 0; i <= 20; i++)

sum += fpow(2ll, (ll)i) \* wt[i][rt];

return sum;

}

Pushdown(rt, R - L + 1);

int mid = (L + R) >> 1;

ll tot = 0;

if (mid >= l)

tot = Ask(ls, L, mid, l, r);

if (mid < r)

tot += Ask(rs, mid + 1, R, l, r);

return tot;

}

int main()

{

int n;

cin >> n;

for (int i = 1; i <= n; i++)

sc("%d", &a[i]);

Build(1, 1, n);

int q;

cin >> q;

while (q--) {

int op, x, y, w;

sc("%d", &op);

if (op == 1) {

sc("%d %d", &x, &y);

printf("%lld\n", Ask(1, 1, n, x, y));

}

else {

sc("%d %d %d", &x, &y, &w);

Update(1, 1, n, x, y, w);

}

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 7.三次幂加乘覆盖

// 线段树

ll lzy[N \* 4][3]; // 加，乘，覆盖

ll wt[N \* 4][3]; // p次幂

void Build(int rt, int L, int R) {

for (int i = 0; i < 3; i++)

wt[rt][i] = 0;

if (L == R)

return;

lzy[rt][0] = 0;

lzy[rt][1] = 1;

lzy[rt][2] = 0;

int mid = (L + R) >> 1;

Build(ls, L, mid);

Build(rs, mid + 1, R);

}

void Pushup(int rt) {

wt[rt][0] = (wt[ls][0] + wt[rs][0]) % Mod;

wt[rt][1] = (wt[ls][1] + wt[rs][1]) % Mod;

wt[rt][2] = (wt[ls][2] + wt[rs][2]) % Mod;

}

void Pushdown(int rt, int len) {

int Llen = len - (len >> 1);

int Rlen = len >> 1;

if (lzy[rt][2]) { // 覆盖

wt[ls][2] = dpow(lzy[rt][2], 3) \* Llen % Mod;

wt[rs][2] = dpow(lzy[rt][2], 3) \* Rlen % Mod;

wt[ls][1] = dpow(lzy[rt][2], 2) \* Llen % Mod;

wt[rs][1] = dpow(lzy[rt][2], 2) \* Rlen % Mod;

wt[ls][0] = lzy[rt][2] \* Llen % Mod;

wt[rs][0] = lzy[rt][2] \* Rlen % Mod;

lzy[ls][0] = lzy[rs][0] = 0;

lzy[ls][1] = lzy[rs][1] = 1;

lzy[ls][2] = lzy[rs][2] = lzy[rt][2];

lzy[rt][2] = 0;

}

if (lzy[rt][1] != 1) { // 乘法

wt[ls][2] = wt[ls][2] \* dpow(lzy[rt][1], 3) % Mod;

wt[rs][2] = wt[rs][2] \* dpow(lzy[rt][1], 3) % Mod;

wt[ls][1] = wt[ls][1] \* dpow(lzy[rt][1], 2) % Mod;

wt[rs][1] = wt[rs][1] \* dpow(lzy[rt][1], 2) % Mod;

wt[ls][0] = wt[ls][0] \* lzy[rt][1] % Mod;

wt[rs][0] = wt[rs][0] \* lzy[rt][1] % Mod;

lzy[ls][0] = lzy[ls][0] \* lzy[rt][1] % Mod;

lzy[rs][0] = lzy[rs][0] \* lzy[rt][1] % Mod;

lzy[ls][1] = lzy[ls][1] \* lzy[rt][1] % Mod;

lzy[rs][1] = lzy[rs][1] \* lzy[rt][1] % Mod;

lzy[rt][1] = 1;

}

if (lzy[rt][0]) { // 加法

wt[ls][2] = (wt[ls][2] + (wt[ls][1] + wt[ls][0] \* lzy[rt][0] % Mod) \* (3 \* lzy[rt][0] % Mod) % Mod

+ (Llen \* dpow(lzy[rt][0], 3)) % Mod) % Mod;

wt[rs][2] = (wt[rs][2] + (wt[rs][1] + wt[rs][0] \* lzy[rt][0] % Mod) \* (3 \* lzy[rt][0] % Mod) % Mod

+ (Rlen \* dpow(lzy[rt][0], 3)) % Mod) % Mod;

wt[ls][1] = (wt[ls][1] + (wt[ls][0] \* 2 \* lzy[rt][0] % Mod + Llen \* dpow(lzy[rt][0], 2) % Mod) % Mod) % Mod;

wt[rs][1] = (wt[rs][1] + (wt[rs][0] \* 2 \* lzy[rt][0] % Mod + Rlen \* dpow(lzy[rt][0], 2) % Mod) % Mod) % Mod;

wt[ls][0] = (wt[ls][0] + Llen \* lzy[rt][0] % Mod) % Mod;

wt[rs][0] = (wt[rs][0] + Rlen \* lzy[rt][0] % Mod) % Mod;

lzy[ls][0] = (lzy[ls][0] + lzy[rt][0]) % Mod;

lzy[rs][0] = (lzy[rs][0] + lzy[rt][0]) % Mod;

lzy[rt][0] = 0;

}

}

void Update(int rt, int L, int R, int l, int r, int w, int op) {

if (L >= l && R <= r) {

// 加法

int len = R - L + 1;

if (op == 1) {

wt[rt][2] = (wt[rt][2] + ((wt[rt][1] + wt[rt][0] \* w % Mod) % Mod) \* (3 \* w % Mod) % Mod + len \* dpow(w, 3) % Mod) % Mod;

wt[rt][1] = (wt[rt][1] + wt[rt][0] \* 2 \* w % Mod + len \* dpow(w, 2) % Mod) % Mod;

wt[rt][0] = (wt[rt][0] + len \* w % Mod) % Mod;

lzy[rt][0] = (lzy[rt][0] + w) % Mod;

}

else if (op == 2) {

wt[rt][2] = wt[rt][2] \* dpow(w, 3) % Mod;

wt[rt][1] = wt[rt][1] \* dpow(w, 2) % Mod;

wt[rt][0] = wt[rt][0] \* w % Mod;

lzy[rt][1] = (lzy[rt][1] \* w) % Mod;

lzy[rt][0] = lzy[rt][0] \* w % Mod;

}

else {

wt[rt][2] = dpow(w, 3) \* len % Mod;

wt[rt][1] = dpow(w, 2) \* len % Mod;

wt[rt][0] = w \* len % Mod;

lzy[rt][0] = 0, lzy[rt][1] = 1;

lzy[rt][2] = w;

}

return;

}

Pushdown(rt, R - L + 1);

int mid = (L + R) >> 1;

if (mid >= l)

Update(ls, L, mid, l, r, w, op);

if (mid < r)

Update(rs, mid + 1, R, l, r, w, op);

Pushup(rt);

}

ll Query(int rt, int L, int R, int l, int r, int k) {

if (L >= l && R <= r)

return wt[rt][k];

Pushdown(rt, R - L + 1);

int mid = (L + R) >> 1;

ll ans = 0;

if (mid >= l)

ans = Query(ls, L, mid, l, r, k);

if (mid < r)

ans += Query(rs, mid + 1, R, l, r, k);

return ans % Mod;

}

int main()

{

int n, q;

while (sc("%d %d", &n, &q) && n + q) {

Build(1, 1, n);

while (q--) {

int op, x, y, w;

sc("%d %d %d %d", &op, &x, &y, &w);

if (op <= 3)

Update(1, 1, n, x, y, w, op);

else

printf("%lld\n", Query(1, 1, n, x, y, w - 1));

}

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 8.维护树直径

vector <int> G[N];

int dep[N], f[N][25];

int l[N], ev[N], cnt;

char op[5];

void Init(int x, int fa) {

l[x] = ++cnt, ev[cnt] = x;

for (auto v : G[x])

if (v != fa)

Init(v, x);

}

void DFS(int x, int fa) {

dep[l[x]] = dep[l[fa]] + 1;

f[l[x]][0] = l[fa];

for (int i = 1; (1 << i) <= dep[l[x]]; i++)

f[l[x]][i] = f[f[l[x]][i - 1]][i - 1];

for (auto v : G[x]) {

if (v != fa)

DFS(v, x);

}

}

int LCA(int x, int y) {

if (dep[x] > dep[y])

swap(x, y);

for (int i = 20; i >= 0; i--) {

if (dep[y] - (1 << i) >= dep[x])

y = f[y][i];

}

if (x == y)

return x;

for (int i = 20; i >= 0; i--) {

if (f[x][i] == f[y][i])

continue;

x = f[x][i], y = f[y][i];

}

return f[x][0];

}

int Dis(int x, int y) {

x = l[x], y = l[y];

return dep[x] + dep[y] - 2 \* dep[LCA(x, y)];

}

#define ls rt << 1

#define rs ls | 1

struct Tree

{

int l, r, w;

Tree operator + (const Tree &oth) const {

Tree v;

v.w = -1;

if (v.w < w)

v = \*this;

if (v.w < oth.w)

v = oth;

if (w != -1 && oth.w != -1) {

int val;

val = Dis(l, oth.l);

if (val > v.w)

v.l = l, v.r = oth.l, v.w = val;

val = Dis(l, oth.r);

if (val > v.w)

v.l = l, v.r = oth.r, v.w = val;

val = Dis(r, oth.l);

if (val > v.w)

v.l = r, v.r = oth.l, v.w = val;

val = Dis(r, oth.r);

if (val > v.w)

v.l = r, v.r = oth.r, v.w = val;

}

return v;

}

}t[N \* 4];

void Build(int rt, int L, int R) {

if (L == R) {

t[rt] = { ev[L], ev[L], 0 };

return;

}

int mid = (L + R) >> 1;

Build(ls, L, mid);

Build(rs, mid + 1, R);

t[rt] = t[ls] + t[rs];

}

void Update(int rt, int L, int R, int x) {

if (L == R) {

if (t[rt].w == -1)

t[rt].w = 0;

else

t[rt].w = -1;

return;

}

int mid = (L + R) >> 1;

if (mid >= x)

Update(ls, L, mid, x);

else

Update(rs, mid + 1, R, x);

t[rt] = t[ls] + t[rs];

}

int main()

{

int n, q;

cin >> n;

for (int i = 1; i < n; i++) {

int u, v;

sc("%d %d", &u, &v);

G[u].push\_back(v);

G[v].push\_back(u);

}

Init(1, 0);

DFS(1, 0);

Build(1, 1, n);

cin >> q;

while (q--) {

int x;

sc("%s", op);

if (op[0] == 'G')

printf("%d\n", t[1].w);

else {

sc("%d", &x);

Update(1, 1, n, l[x]);

}

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

#### 9.单点修改最大子段和

struct Tree

{

int l, r;

int lmx, rmx, mx, sum;

}t[N \* 4];

int n, m;

void Pushup(int rt) {

t[rt].lmx = t[ls].lmx, t[rt].rmx = t[rs].rmx;

Max(t[rt].lmx, t[ls].sum + t[rs].lmx);

Max(t[rt].rmx, t[rs].sum + t[ls].rmx);

t[rt].mx = max(t[ls].mx, t[rs].mx);

Max(t[rt].mx, t[ls].rmx + t[rs].lmx);

t[rt].sum = t[ls].sum + t[rs].sum;

}

void Build(int rt, int l, int r) {

t[rt].l = l, t[rt].r = r;

if (l == r) {

sc("%d", &t[rt].mx);

t[rt].lmx = t[rt].rmx = t[rt].sum = t[rt].mx;

return;

}

int mid = (l + r) >> 1;

Build(ls, l, mid);

Build(rs, mid + 1, r);

Pushup(rt);

}

void Update(int rt, int x, int y) {

if (t[rt].l == x && t[rt].r == x) {

t[rt].lmx = t[rt].rmx = t[rt].sum = t[rt].mx = y;

return;

}

int mid = (t[rt].l + t[rt].r) >> 1;

if (mid < x)

Update(rs, x, y);

else

Update(ls, x, y);

Pushup(rt);

}

Tree Query(int rt, int l, int r) {

if (t[rt].l >= l && t[rt].r <= r)

return t[rt];

int mid = (t[rt].l + t[rt].r) >> 1;

Tree L, R, K;

L = R = { t[rt].l, t[rt].r, -INF, -INF, -INF };

K.sum = 0;

if (mid >= l)

L = Query(ls, l, r), K.sum += L.sum;

if (mid < r)

R = Query(rs, l, r), K.sum += R.sum;

K.mx = max(L.mx, R.mx);

Max(K.mx, L.rmx + R.lmx);

K.lmx = max(L.lmx, L.sum + R.lmx);

if (mid < l)

K.lmx = R.lmx;

K.rmx = max(R.rmx, R.sum + L.rmx);

if (mid >= r)

K.rmx = L.rmx;

return K;

}

int main()

{

cin >> n >> m;

Build(1, 1, n);

for (int i = 0; i < m; i++) {

int op, xi, yi;

sc("%d %d %d", &op, &xi, &yi);

if (op == 1) {

if (xi > yi)

swap(xi, yi);

printf("%d\n", Query(1, xi, yi).mx);

}

else

Update(1, xi, yi);

}

return 0; // 改数组大小!!!

}

#### 10.区间1e9的动态开点

int bt[N \* 50];

int ls[N \* 50], rs[N \* 50], lzy[N \* 50];

unordered\_map <int, int> rt;

int cnt;

void Pushdown(int o, int len) {

int Llen = len - (len >> 1);

int Rlen = len >> 1;

if (!ls[o])

ls[o] = ++cnt;

if (!rs[o])

rs[o] = ++cnt;

lzy[ls[o]] = lzy[rs[o]] = lzy[o];

bt[ls[o]] = Llen \* lzy[o];

bt[rs[o]] = Rlen \* lzy[o];

lzy[o] = -1;

}

void Update(int &o, int L, int R, int l, int r, int k) {

int len = R - L + 1;

if (!o)

o = ++cnt, lzy[o] = -1;

if (L >= l && R <= r) {

bt[o] = len \* k;

lzy[o] = k;

return;

}

if (lzy[o] != -1)

Pushdown(o, len);

int mid = (L + R) >> 1;

if (mid >= l)

Update(ls[o], L, mid, l, r, k);

if (mid < r)

Update(rs[o], mid + 1, R, l, r, k);

bt[o] = bt[ls[o]] + bt[rs[o]];

}

int Ask(int o, int L, int R, int l, int r) {

int len = R - L + 1;

if (L >= l && R <= r)

return bt[o];

if (lzy[o] != -1)

Pushdown(o, len);

int mid = (L + R) >> 1, tot = 0;

if (mid >= l)

tot = Ask(ls[o], L, mid, l, r);

if (mid < r)

tot += Ask(rs[o], mid + 1, R, l, r);

return tot;

}

int main()

{

int n, q;

cin >> n >> q;

while (q--) {

int l, r, op;

sc("%d %d %d", &l, &r, &op);

if (op == 2)

op = 0;

Update(rt[1], 1, n, l, r, op);

printf("%d\n", n - Ask(1, 1, n, 1, n));

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

## 莫队

### 模板

#### 1.区间不同数

// N \* sqrt(N)

struct node

{

int l, r, id;

}s[N];

int a[N], bel[N], n, m, k;

int cnt[N], L, R, tot;

int sz, len;

int ans[N];

vector <int> v;

bool cmp(node a, node b) {

return (bel[a.l] ^ bel[b.l]) ? bel[a.l] < bel[b.l] : ((bel[a.l] & 1) ? a.r < b.r : a.r > b.r);

}

void Init() {

L = 1, R = tot = 0;

sz = sqrt(n), len = ((double)n / sz);

for (int i = 1; i <= sz; i++) {

for (int j = (i - 1) \* len; j <= i \* len; j++)

bel[j] = i;

}

}

void Add(int x) {

tot += !cnt[a[x]];

cnt[a[x]]++;

}

void Del(int x) {

tot -= cnt[a[x]] == 1;

cnt[a[x]]--;

}

int main()

{

cin >> n;

Init();

for (int i = 1; i <= n; i++)

sc("%d", &a[i]), v.push\_back(a[i]);

sort(ALL(v));

v.erase(unique(ALL(v)), v.end());

for (int i = 1; i <= n; i++)

a[i] = lower\_bound(ALL(v), a[i]) - v.begin() + 1;

cin >> m;

for (int i = 1; i <= m; i++)

sc("%d %d", &s[i].l, &s[i].r), s[i].id = i;

sort(s + 1, s + m + 1, cmp);

for (int i = 1; i <= m; i++) {

int l = s[i].l, r = s[i].r, id = s[i].id;

while (L < l)

Del(L++);

while (L > l)

Add(--L);

while (R > r)

Del(R--);

while (R < r)

Add(++R);

ans[id] = tot;

}

for (int i = 1; i <= m; i++)

printf("%d\n", ans[i]);

return 0; // 改数组大小!!!用pair改宏定义!!!

}

### 题集

#### 1.查询区间数次数的平方

struct node

{

int l, r, id;

}s[N];

int a[N], bel[N], n, m, k;

int cnt[N], L, R, tot;

int sz, len;

ll ans[N];

bool cmp(node a, node b) {

return (bel[a.l] ^ bel[b.l]) ? bel[a.l] < bel[b.l] : ((bel[a.l] & 1) ? a.r < b.r : a.r > b.r);

}

void Init() {

L = 1, R = tot = 0;

sz = sqrt(n), len = ((double)n / sz);

for (int i = 1; i <= sz; i++) {

for (int j = (i - 1) \* len; j <= i \* len; j++)

bel[j] = i;

}

}

void Add(int x) {

tot -= P2(cnt[a[x]]);

cnt[a[x]]++, tot += P2(cnt[a[x]]);

}

void Del(int x) {

tot -= P2(cnt[a[x]]);

cnt[a[x]]--, tot += P2(cnt[a[x]]);

}

int main()

{

cin >> n >> m >> k;

Init();

for (int i = 1; i <= n; i++)

sc("%d", &a[i]);

for (int i = 1; i <= m; i++)

sc("%d %d", &s[i].l, &s[i].r), s[i].id = i;

sort(s + 1, s + m + 1, cmp);

for (int i = 1; i <= m; i++) {

int l = s[i].l, r = s[i].r, id = s[i].id;

while (L < l)

Del(L++);

while (L > l)

Add(--L);

while (R > r)

Del(R--);

while (R < r)

Add(++R);

ans[id] = tot;

}

for (int i = 1; i <= m; i++)

printf("%lld\n", ans[i]);

return 0; // 改数组大小!!!用pair改宏定义!!!

}

## 树上启发式合并

### 模板

#### 1.颜色种类

vector <int> G[N];

set <int> st[N];

int a[N], ans[N];

void dfs(int x, int fa) {

st[x].insert(a[x]);

for (auto v : G[x]) {

if (v == fa)

continue;

dfs(v, x);

if (SZ(st[v]) > SZ(st[x]))

swap(st[x], st[v]);

for (auto it : st[v])

st[x].insert(it);

}

ans[x] = SZ(st[x]);

}

### 题集：

#### 1.差分答案，树状数组后缀和

vector <int> G[N];

int dp[N][3];

void DFS(int x, int fa) {

dp[x][0] = 1;

int tot = INF;

bool ok = false;

for (int i = 0; i < SZ(G[x]); i++) {

int v = G[x][i];

if (v == fa)

continue;

DFS(v, x);

dp[x][0] += min(dp[v][0], min(dp[v][1], dp[v][2]));

dp[x][2] += min(dp[v][0], dp[v][1]);

if (dp[v][0] <= dp[v][1]) {

ok = true;

dp[x][1] += dp[v][0];

}

else {

Min(tot, dp[v][0] - dp[v][1]);

dp[x][1] += dp[v][1];

}unordered\_map <int, int> mp[N]; // 下标,颜色,次数

vector <int> G[N << 1];

vector <pir> q[N]; // 答案下标，查询的k

int a[N], ans[N];

int d[N], M; // 颜色数量出现次数,后缀和

void add(int x, int y) {

if (!x)

return;

x = N - x;

while (x <= N) {

d[x] += y;

x += lowbit(x);

}

}

int Ask(int x) {

int tot = 0;

x = N - x;

while (x) {

tot += d[x];

x -= lowbit(x);

}

return tot;

}

void dfs(int x, int fa) {

vector <int> vec;

for (int i = 0; i < SZ(q[x]); i++)

vec.push\_back(Ask(q[x][i].second)); // 递归前答案

if (mp[x].count(a[x]))

add(mp[x][a[x]], -1);

mp[x][a[x]]++, add(mp[x][a[x]], 1); // 加上本身

for (auto v : G[x]) {

if (v == fa)

continue;

dfs(v, x);

if (SZ(mp[v]) > SZ(mp[x]))

swap(mp[x], mp[v]);

for (auto it : mp[v]) {

int num = it.second; // 失去该次数

add(num, -1);

if (mp[x].count(it.first))

add(mp[x][it.first], -1); // 失去该次数

mp[x][it.first] += num; // 合并后的次数 + 1

add(mp[x][it.first], 1);

}

}

for (int i = 0; i < SZ(q[x]); i++) // 减去更新前的值

ans[q[x][i].first] = Ask(q[x][i].second) - vec[i];

}

int main()

{

int n, m;

cin >> n >> m;

for (int i = 1; i <= n; i++)

sc("%d", &a[i]), Max(M, a[i]);

for (int i = 1; i < n; i++) {

int u, v;

sc("%d %d", &u, &v);

G[u].push\_back(v);

G[v].push\_back(u);

}

for (int i = 1; i <= m; i++) {

int u, k;

sc("%d %d", &u, &k);

q[u].push\_back({ i, k });

}

dfs(1, 0);

for (int i = 1; i <= m; i++)

printf("%d\n", ans[i]);

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

if (!ok)

dp[x][1] += tot;

}

int main()

{

int n;

cin >> n;

for (int i = 1; i < n; i++) {

int u, v;

sc("%d %d", &u, &v);

G[u].push\_back(v);

G[v].push\_back(u);

}

DFS(1, 0);

printf("%d\n", min(dp[1][0], dp[1][1]));

return 0; // 改数组大小!!!用pair改宏定义!!!

}

# 字符串

## 哈希

### 1.SRC哈希

// pair <区间和，区间异或和> 是一个单独的对

// 判断区间是否拥有相同的排列

// 以下为牛客多校查询K包的题

unordered\_map <int, int> vis;

int a[N], n, k;

int pre[N], suf[N];

ll sum[N], Xor[N];

void Init() {

vis.clear();

pre[0] = 0;

for (int i = 1; i <= n; i++) {

pre[i] = pre[i - 1];

if (!vis.count(a[i]))

pre[i]++;

vis[a[i]]++;

}

suf[n + 1] = 0, vis.clear();

for (int i = n; i >= 1; i--) {

suf[i] = suf[i + 1];

if (!vis.count(a[i]))

suf[i]++;

vis[a[i]]++;

}

}

bool Calc(ll tot, ll pxor, int x) {

for (int i = x; i <= n; i += k) {

int r = i + k - 1;

if (r > n)

return suf[i] == n - i + 1;

if (sum[r] - sum[i - 1] != tot || (Xor[r] ^ Xor[i - 1]) != pxor)

return false;

}

return true;

}

bool Check() {

if (n <= k) {

for (int i = 2; i <= n; i++) {

if (pre[i - 1] == i - 1 && suf[i] == n - i + 1)

return true;

}

}

else {

ll tot =0, pxor = 0;

for (int i = 1; i <= k; i++)

tot += i, pxor ^= i;

for (int i = 1; i <= n; i++) {

if (pre[i - 1] != i - 1)

break;

if (Calc(tot, pxor, i))

return true;

}

}

return false;

}

int main()

{

int T; cin >> T;

while (T--) {

sc("%d %d", &n, &k);

bool ok = true;

for (int i = 1; i <= n; i++) {

sc("%d", &a[i]);

if (a[i] > k)

ok = false;

sum[i] = sum[i - 1] + a[i];

Xor[i] = Xor[i - 1] ^ a[i];

}

if (!ok) {

puts("NO");

continue;

}

Init(); // 前后缀数字个数

if (Check())

puts("YES");

else

puts("NO");

}

return 0; // 改数组大小!!!用pair改宏定义!!!

}

### 2.康托展开

fac[0] = 1;

for (int i = 1; i < 100; ++i)

fac[i] = fac[i - 1] \* i;

ll Contor(int \*a)

{

ll res = 0;

for (int i = 1; i <= M; ++i) {

ll cnt = 0;

for (int j = i + 1; j <= M; ++j)

if (a[i] > a[j])

++cnt;

res += cnt \* fac[M - i];

}

return res;

}

//可以将1到n的排列无碰撞的压缩到0~n!-1范围内

### 3.矩阵哈希

ll has(int s[][8]) { // 矩阵的哈希值

ll tot = 0;

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= n; j++) {

tot = (tot \* B + s[i][j]) % MOD;

}

}

return tot;

}

### 4.大数减法字符串哈希

ll w[N], hs[N];

ll Get(int l, int r) {

return ((hs[r] - hs[l - 1] \* w[r - l + 1]) % Mod + Mod) % Mod;

}

w[0] = 1;

for (int i = 1; i < N; i++)

w[i] = w[i - 1] \* 10 % Mod;

for (int i = 1; i <= n; i++) {

hs[i] = (hs[i - 1] \* 10 + a[i]) % Mod; // a为字符串代表的数字

}

### 5.手写双double哈希

struct HashMap

{

static const int MXSZ = 1e7 + 100;

static const int MOD = 1e7 + 19;

struct node

{

ll x, y;

int val, nxt;

}elem[MXSZ];

int head[MOD], tot;

void Init()

{

tot = 0;

memset(head, -1, sizeof(head));

}

int& Find(double xx, double yy)

{

ll x = xx \* 100000000 + 0.1;

ll y = yy \* 100000000 + 0.1;

int k = ((x % MOD \* (ll)(1e18) % MOD + y) % MOD + MOD) % MOD;

for (int i = head[k]; ~i; i = elem[i].nxt)

if (elem[i].x == x && elem[i].y == y)

return elem[i].val;

elem[tot].x = x, elem[tot].y = y, elem[tot].nxt = head[k], head[k] = tot;

return elem[tot++].val = 0;

}

}mp;

## 01字典树

#### 模板

int trie[N \* 32][3], pos;

int vis[N \* 32];

void Init() {

pos = 0;

MEM(vis, 0), MEM(trie, 0);

}

void Add(int x) {

int c = 0, u;

for (int i = 31; i >= 0; i--) {

u = x >> i & 1;

if (trie[c][u] == 0)

trie[c][u] = ++pos;

c = trie[c][u];

}

vis[c] = x;

}

int Ask(int x) {

int c = 0, u;

for (int i = 31; i >= 0; i--) {

u = x >> i & 1;

if (trie[c][u ^ 1])

c = trie[c][u ^ 1];

else

c = trie[c][u];

}

return vis[c];

}

## AC自动机

### 模板

int trie[N][30], fail[N], sum[N];

int n, tot;

char s[N];

void Init() {

MEM(sum, 0), MEM(fail, -1), MEM(trie, 0);

tot = 0;

}

void Insert(char \*s) {

int rt = 0, sz = strlen(s);

for (int i = 0; i < sz; i++) {

int pos = s[i] - 'a';

if (!trie[rt][pos])

trie[rt][pos] = ++tot;

rt = trie[rt][pos];

}

sum[rt]++;

}

void Getfail() {

queue <int> q;

for (int i = 0; i < 26; i++) {

if (trie[0][i]) {

fail[trie[0][i]] = 0;

q.push(trie[0][i]);

}

}

while (!q.empty()) {

int u = q.front();

q.pop();

for (int i = 0; i < 26; i++) {

if (trie[u][i]) {

fail[trie[u][i]] = trie[fail[u]][i];

q.push(trie[u][i]);

}

else

trie[u][i] = trie[fail[u]][i];

}

}

}

int Ask(char \*s) {

int rt = 0, ans = 0;

int sz = strlen(s);

for (int i = 0; i < sz; i++) {

int id = s[i] - 'a';

rt = trie[rt][id];

for (int j = rt; j != -1 && sum[j] != -1; j = fail[j]) {

ans += sum[j];

sum[j] = -1;

}

}

return ans;

}

int main()

{

sc("%d", &n);

Init();

for (int i = 0; i < n; i++) {

sc("%s", s);

Insert(s);

}

Getfail();

sc("%s", s);

printf("%d\n", Ask(s));

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

## KMP

### 模板

int a[M], b[N], nxt[N];

int n, m;

void Get\_Next() {

MEM(nxt, -1);

nxt[1] = 0;

int i = 1, j = 0;

while (i <= m) {

if (!j || b[i] == b[j])

nxt[++i] = ++j;

else

j = nxt[j];

}

}

int KMP() {

Get\_Next();

int i = 1, j = 1, ans = 0;

while (i <= n) {

if (!j || a[i] == b[j])

i++, j++;

else

j = nxt[j];

if (j == m + 1)

return i - m; // 匹配上

}

return -1;

}

## 马拉车

### Manacher最长回文串

char s[N], str[N];

int Len[N], sz;

void Init() {

sz = strlen(s), MEM(str, 0);

int k = 0;

str[k++] = '%';

for (int i = 0; i < sz; i++) {

str[k++] = '#';

str[k++] = s[i];

}

str[k++] = '#';

sz = strlen(str);

}

int Manacher() {

int idx = 0, mx = 0, ans = 0;

Len[0] = 0;

for (int i = 1; i < sz; i++) {

if (mx > i)

Len[i] = min(mx - i, Len[2 \* idx - 1]);

else

Len[i] = 1;

while (str[i + Len[i]] == str[i - Len[i]])

Len[i]++;

if (mx < Len[i] + i) {

mx = Len[i] + i;

idx = i;

Max(ans, Len[i]);

}

}

return ans - 1;

}

## 最小表示法

### 模板

string getMin(string s)

{

int len = s.size();

s += s;

int i = 0, j = 1, k = 0;

while (i < len && j < len && k < len) {

int tmp = s[(i + k) % len] - s[(j + k) % len];

if (tmp == 0) k++;

else {

if (tmp > 0) i+= k + 1;

else j += k + 1;

if (i == j) j++;

k = 0;

}

}

string t = s.substr(min(i, j), len);

return t;

}

## 数符转换

### 模板

v[i].find(str) != string::npos

//能找到

字符串转换成数字 ：

浮点型 ：

string s;

cin >> s;

double tmp = stod(s);

int型 ：

string s;

cin >> s;

int ans = stoi(s);

long long型 ：

string s;

cin >> s;

ll ans = stoll(s);

数字转换成字符串 ：

任意型 ：

double/int/long long ans ;

cin >> ans;

string s = to\_string(ans);

查找某个字符串是否是另一个字符串的子串 ：

string line, s;

cin >> line >> s;

if (line.find(s) != string :: npos ) 表示s是line的子串

字符串大小写转换 ：

transform(a.begin(), a.end(), a.begin(), :: tolower); 变小写

transform(a.begin(), a.end(), a.begin(), :: toupper);

## 大整数string加法

### 模板

string add(string a, string b)

{

a = a.substr(a.find\_first\_not\_of('0'));

b = b.substr(b.find\_first\_not\_of('0'));

long long len1 = a.length();

long long len2 = b.length();

long long len = max(len1, len2) + 10;

string ans(len, '0');

reverse(a.begin(), a.end());

reverse(b.begin(), b.end());

for (int i = 0; i<len1; i++)

ans[i] = a[i];

int tmp = 0;

for (int i = 0; i<len; i++)

{

if (len1 >= len2)//考虑a>=b

{

if (i<len2)

{

tmp += (ans[i] - '0') + (b[i] - '0');

}

else

{

tmp += (ans[i] - '0');

}

ans[i] = tmp % 10 + '0';

tmp /= 10;

}

else if (len1<len2)//a<b

{

if (i<len1)

{

tmp += (ans[i] - '0') + (b[i] - '0');

}

else if (i<len2)

{

tmp += (b[i] - '0');

}

ans[i] = tmp % 10 + '0';

tmp /= 10;

}

}

reverse(ans.begin(), ans.end());

return ans.substr(ans.find\_first\_not\_of('0'));

}

## 大数减法

### 模板

void Minus(int \*a, int \*b, int sz) {

for (int i = sz - 1; i >= 0; i--)

a[i] -= b[i];

for (int i = sz - 1; i >= 0; i--) {

if (a[i] < 0) {

a[i - 1]--;

a[i] += 10;

}

}

}

## 大数取模

### 模板

int mod(string str, int c) {

int number[110];

for (int i = 0; i < str.size(); i++)

number[i] = str[i] - '0';

int sum = 0;

for (int i = 0; i < str.size(); i++)

sum = ((ll)sum \* 10 + number[i]) % c;

return sum;

}

# 杂项

## 1.逆波兰数

const int N = 1e3 + 10;

char s[N];

struct node

{

double v;

char c; //操作符 为0则为值

node &operator + (const node &oth) //+-\*/根据需求改变

{

v += oth.v;

return \*this;

}

node &operator - (const node &oth)

{

v -= oth.v;

return \*this;

}

node &operator \* (const node &oth)

{

v \*= oth.v;

return \*this;

}

node &operator / (const node &oth)

{

v /= oth.v;

return \*this;

}

};

int Lv(char c) //符号优先级 特殊函数保留逗号

{

if (c == '+')

return 1;

if (c == '-')

return 1;

if (c == '\*')

return 2;

if (c == '/')

return 2;

return 0;

}

queue<node> q;

stack<node> stk;

void ToRPN(char \*s) //输入字符串s逆波兰处理后存入q中

{

double t = 0, p = 0; //数字 小数权值

for (int i = 0; s[i]; i++)

{

if (isdigit(s[i])) //数字 直接进队

{

if (!i || !isdigit(s[i - 1]) && s[i - 1] != '.') //数字第一位

t = s[i] - '0';

else if (p == 0) //没到小数点

t = t \* 10 + s[i] - '0';

else //小数位

t = t + (s[i] - '0') / p, p \*= 10;

if (!isdigit(s[i + 1]) && s[i + 1] != '.') //数字结束 入队并清空小数标记

q.push({ t, 0 }), p = 0;

}

else if (s[i] == '.') //标记小数开始

p = 10;

else if (s[i] == '(') //左括号直接进栈

stk.push({ 0.0, s[i] });

else if (s[i] == ')') //右括号 将上一个左括号后的操作全部进队

{

while (stk.top().c != '(')

q.push(stk.top()), stk.pop();

stk.pop();

}

else //其它操作符 将比自身优先级高或者相等的全部进队自身再进栈

{

while (!stk.empty() && Lv(s[i]) <= Lv(stk.top().c))

q.push(stk.top()), stk.pop();

stk.push({ 0, s[i] });

}

}

while (!stk.empty()) //剩余全部进队

q.push(stk.top()), stk.pop();

}

double CalRPN() //计算q中的逆波兰并返回答案

{

while (!q.empty())

{

char c = q.front().c;

if (!c) //数字 直接进栈

stk.push(q.front()), q.pop();

else //操作符 计算

{

q.pop();

node lst = stk.top();

stk.pop();

node fst = stk.top();

stk.pop();

if (c == '+')

fst = fst + lst;

if (c == '-')

fst = fst - lst;

if (c == '\*')

fst = fst \* lst;

if (c == '/')

fst = fst / lst;

stk.push(fst);

}

}

double res = stk.top().v; //返回答案

stk.pop();

return res;

}

int main()

{

int T;

cin >> T;

while (T--)

{

scanf("%s", s);

for (int i = 0;; i++) //消除等号

if (s[i] == '=')

{

s[i] = 0;

break;

}

ToRPN(s);

printf("%.2f\n", CalRPN());

}

return 0;

}

## 2.曼哈顿矩阵表示

//d为距离当前点的曼哈顿距离

for (int x = -d; x <= d; x++)

for (int y = abs(x) - d; y <= d - abs(x); y++)

表示坐标:i + x, j + y

## 3.等比矩阵求前N项和

ll n, k, I[N][N];

ll a[N][N], ret[N][N];

void mul(ll a[N][N], ll b[N][N], ll c[N][N])

{

ll tmp[N][N];

for (ll i = 1; i <= n; i++)

for (ll j = 1; j <= n; j++)tmp[i][j] = 0;

for (ll i = 1; i <= n; i++)

for (ll j = 1; j <= n; j++)

for (ll k = 1; k <= n; k++)

tmp[i][j] += a[i][k] \* b[k][j] % MOD, tmp[i][j] %= MOD;

for (ll i = 1; i <= n; i++)

for (ll j = 1; j <= n; j++)

c[i][j] = (tmp[i][j] + MOD) % MOD;

}

void work(ll a[N][N], ll num, ll c[N][N])

{

ll s[N][N];

for (ll i = 1; i <= n; i++)

for (ll j = 1; j <= n; j++)s[i][j] = a[i][j];

for (ll i = 1; i <= n; i++)for (ll j = 1; j <= n; j++)c[i][j] = 0;

for (ll i = 1; i <= n; i++)c[i][i] = 1;

while (num)

{

if (num & 1)mul(c, s, c);

mul(s, s, s); num >>= 1;

}

}

void cal(ll a[N][N], ll num, ll c[N][N])

{

ll t[N][N], now[N][N], ret[N][N];

if (num == 1)

{

for (ll i = 1; i <= n; i++)

for (ll j = 1; j <= n; j++)

c[i][j] = a[i][j];

return;

}

else if (num == 0)

{

for (ll i = 1; i <= n; i++)

for (ll j = 1; j <= n; j++)

c[i][j] = a[i][j];

return;

}

if (num & 1)

{

work(a, num / 2 + 1, now);

cal(a, num / 2, ret);

work(a, num / 2 + 1, t);

for (ll i = 1; i <= n; i++)

for (ll j = 1; j <= n; j++)

t[i][j] += I[i][j], t[i][j] %= MOD;

mul(ret, t, ret);

for (ll i = 1; i <= n; i++)

for (ll j = 1; j <= n; j++)

ret[i][j] += now[i][j], ret[i][j] %= MOD;

for (ll i = 1; i <= n; i++)

for (ll j = 1; j <= n; j++)

c[i][j] = (ret[i][j] + MOD) % MOD;

}

else

{

cal(a, num / 2, ret);

work(a, num / 2, now);

for (ll i = 1; i <= n; i++)

for (ll j = 1; j <= n; j++)

now[i][j] += I[i][j], now[i][j] %= MOD;

mul(now, ret, ret);

for (ll i = 1; i <= n; i++)

for (ll j = 1; j <= n; j++)

c[i][j] = (ret[i][j] + MOD) % MOD;

}

}

void init()

{

for (ll i = 1; i <= n; i++)

I[i][i] = 1;

}

int main()

{

cin >> n >> k;

for (ll i = 1; i <= n; i++)

for (ll j = 1; j <= n; j++)

sc("%lld", &a[i][j]);

init(); cal(a, k, ret);

for (ll i = 1; i <= n; i++)

{

for (ll j = 1; j <= n; j++)

printf("%lld ", ret[i][j]);

printf("\n");

}

}

## 4.已知一条边构造RT三角形

ll n, k;

int main()

{

cin >> n;

if (!(n & 1)) {

k = n / 2;

if (k < 2) cout << -1 << endl;

else cout << k \* k - 1 << " " << k \* k + 1 << endl;

}

else {

k = n / 2;

if (k <= 0) cout << -1 << endl;

else cout << 2 \* k \* k + 2 \* k << " " << 2 \* k \* k + 2 \* k + 1 << endl;

}

return 0;

}

## 5.矩阵已知三点求第四点

pir arr(int x1, int y1, int x2, int y2, int x3, int y3) {

return{ x1 + x3 - x2, y1 + y3 - y2 };

}

pir Getarr(int x1, int y1, int x2, int y2, int x3, int y3) {

pir now;

if ((x1 - x2)\*(x2 - x3) + (y1 - y2)\*(y2 - y3) == 0)

return arr(x1, y1, x2, y2, x3, y3);

if ((x1 - x3)\*(x2 - x3) + (y1 - y3)\*(y2 - y3) == 0)

return arr(x1, y1, x3, y3, x2, y2);

if ((x1 - x3)\*(x2 - x1) + (y1 - y3)\*(y2 - y1) == 0)

return arr(x3, y3, x1, y1, x2, y2);

}

## 6.简易对拍器

### r.cpp

#include <bits/stdc++.h>

#include <random>

using namespace std;

typedef long long ll;

#define pir pair <int, int>

mt19937\_64 mt;

stringstream ss, ss2;

void out() { ss << endl; } template <typename S, typename... T> void out(S x, T ... y){ ss << x << " "; out(y...); }

ll dr(ll l, ll r) {

return mt() % (r - l + 1) + l;

}

char cr(const string &s) {

return s[mt() % s.size()];

}

map<int, map<int, int>> g;

void Tre(int n)

{

g.clear();

for (int i = 2; i <= n; ++i)

{

int u = dr(1, i - 1), v = i;

out(u, v);

g[u][v] = g[v][u] = 1;

}

}

void Tree(int n, ll l = 0, ll r = -1) //生成n个点的树 边权范围[l,r]

{

g.clear();

for (int i = 2; i <= n; ++i)

{

int u = dr(1, i - 1), v = i;

if (l <= r)

out(u, v, dr(l, r));

else

out(u, v);

g[u][v] = g[v][u] = 1;

}

}

//生成n个点m条边的无向图 conn是否必连通 uniq是否无重边 边权范围[l,r] 返回除生成树附加的边

vector<pir> Graph(int n, int m, bool conn = 1, bool uniq = 1, ll l = 0, ll r = -1)

{

vector<pir> vec;

g.clear();

if (uniq)

assert(m <= n \* (n - 1) / 2); //无重边自环

if (conn) //必连通

{

assert(m >= n - 1), m -= n - 1;

Tree(n, l, r);

}

while (m--)

{

int u = dr(1, n), v = dr(1, n);

while (u == v || (uniq && g[u][v]))

u = dr(1, n), v = dr(1, n);

if (uniq)

g[u][v] = g[v][u] = 1;

if (l <= r)

out(u, v, dr(l, r));

else

out(u, v);

vec.push\_back({ u, v });

}

return vec;

}

//生成n个点m条边的DAG uniq是否无重边 边权范围[l, r]

void DAG(int n, int m, bool uniq = 1, ll l = 0, ll r = -1, int id = 1)

{

g.clear();

assert(n != 1 || m == 0);

if (uniq) //无重边

assert(m <= n \* (n - 1) / 2);

while (m--)

{

int u = dr(1, n - 1), v = dr(u + 1, n);

while (uniq && g[u][v])

u = dr(1, n - 1), v = dr(u + 1, n);

g[u][v] = 1;

if (l <= r)

out(u - 1 + id, v - 1 + id, dr(l, r));

else

out(u - 1 + id, v - 1 + id);

}

}

int main(int argc, char \*argv[])

{

int n = dr(10, 100), m = dr(10, 100);

out(n, m);

Graph(n, m, 1, 1);

return 0;

}

### cmp.bat

set path=%path%;D:\CodeBlocksV1.6\CodeBlocks\TDM\bin

g++ a.cpp -o a.exe -std=c++11

g++ b.cpp -o b.exe -std=c++11

g++ r.cpp -o r.exe -std=c++11

@echo off

:loop

r.exe %random% > in.txt

a.exe < in.txt > aout.txt

b.exe < in.txt > bout.txt

fc /A aout.txt bout.txt

if not errorlevel 1 goto loop

pause

## 7.圆上2\*N个点不相交概率

方案数为：

(2 \* N)! / (2 ^ n) / (n!)

合法方案数为：

(2 \* N)! / (n! \* n!) / (n + 1) 即卡特兰数

二者相除即为答案

## 8.贪心二分

int Lis(int sz) {

vector <int> v;

int cnt = 0;

arr[++cnt] = b[1];

for (int i = 2; i <= sz; i++) {

if (b[i] > arr[cnt])

arr[++cnt] = b[i];

else

arr[lower\_bound(arr + 1, arr + cnt + 1, b[i]) - arr] = b[i];

}

return cnt;

}

## 9.除法向上取整

非负(x + y - 1) / y

## 10.三个矩形能否构成正方形

struct node

{

int x, y; // 宽，长

bool operator < (const node &oth) const {

return y > oth.y;

}

} a[5];

int main()

{

for (int i = 1; i <= 3; i++) {

sc("%d %d", &a[i].x, &a[i].y);

if (a[i].x > a[i].y)

swap(a[i].x, a[i].y);

}

sort(a + 1, a + 4);

// 三根长相同合并

if (a[2].y == a[3].y && a[1].y == a[2].y) {

if (a[1].x + a[2].x + a[3].x == a[1].y)

cout << "YES" << endl;

else

cout << "NO" << endl;

exit(0);

}

// 一根带两根

int x = a[1].y - a[1].x, y = a[1].y;

if (a[2].x == a[3].x && a[2].x == x && a[2].y + a[3].y == y)

cout << "YES" << endl, exit(0);

if (a[2].x == a[3].y && a[2].x == x && a[2].y + a[3].x == y)

cout << "YES" << endl, exit(0);

if (a[2].y == a[3].y && a[2].y == x && a[2].x + a[3].x == y)

cout << "YES" << endl, exit(0);

if (a[2].y == a[3].x && a[2].y == x && a[2].x + a[3].y == y)

cout << "YES" << endl, exit(0);

cout << "NO" << endl;

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

## 11.二维离散化

int dir[4][2] = { 1, 0, -1, 0, 0, 1, 0, -1 };

int xi[N], yi[N];

int x[N], y[N];

int n, m, nx, ny, k;

ll sx[N], sy[N], ans;

bool vis[N][N];

unordered\_map <int, int> evx, evy;

void init() {

MEM(vis, 0);

evx.clear(), evy.clear();

}

void dfs(int x, int y) {

vis[x][y] = true;

ans += sx[x] \* sy[y];

for (int i = 0; i < 4; i++) {

int xx = x + dir[i][0];

int yy = y + dir[i][1];

if (xx >= 1 && xx <= nx && yy >= 1 && yy <= ny && !vis[xx][yy])

dfs(xx, yy);

}

}

int main()

{

int T; cin >> T;

int Case = 0;

while (T--) {

sc("%d %d %d", &n, &m, &k);

init();

// 边界

int cx = 0, cy = 0;

x[++cx] = 1, x[++cx] = n;

y[++cy] = 1, y[++cy] = m;

for (int i = 1; i <= k; i++) {

sc("%d %d", &xi[i], &yi[i]);

x[++cx] = xi[i];

y[++cy] = yi[i];

}

// 离散x轴

sort(x + 1, x + cx + 1);

int szx = unique(x + 1, x + cx + 1) - (x + 1);

nx = 0;

for (int i = 1; i <= szx; i++) {

// 中间隔着一个

if (x[i] != x[i - 1] + 1)

sx[++nx] = x[i] - x[i - 1] - 1;

sx[++nx] = 1;

evx[x[i]] = nx;

}

// 离散y轴

sort(y + 1, y + cy + 1);

int szy = unique(y + 1, y + cy + 1) - (y + 1);

ny = 0;

for (int i = 1; i <= szy; i++) {

if (y[i] != y[i - 1] + 1)

sy[++ny] = y[i] - y[i - 1] - 1;

sy[++ny] = 1;

evy[y[i]] = ny;

}

for (int i = 1; i <= k; i++)

vis[evx[xi[i]]][evy[yi[i]]] = true;

vector <ll> vec;

for (int i = 1; i <= nx; i++) {

for (int j = 1; j <= ny; j++) {

if (vis[i][j])

continue;

ans = 0;

dfs(i, j);

vec.push\_back(ans);

}

}

printf("Case #%d:\n", ++Case);

printf("%d\n", SZ(vec));

sort(ALL(vec));

for (int i = 0; i < SZ(vec) - 1; i++)

printf("%lld ", vec[i]);

printf("%lld\n", vec[SZ(vec) - 1]);

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

## 12.三元环计数

// 度数不相同时，按编号连保证无环，相同时小度数连大度数

vector <int> G[N];

int dep[N], n, m;

int dfn[N];

pir eg[N];

int main()

{

cin >> n >> m;

for (int i = 0; i < m; i++) {

int u, v;

sc("%d %d", &u, &v);

dep[u]++, dep[v]++;

eg[i] = { u, v };

}

for (int i = 0; i < m; i++) {

int u = eg[i].first, v = eg[i].second;

if (dep[u] == dep[v]) {

if (u > v)

swap(u, v);

}

else {

if (dep[u] > dep[v])

swap(u, v);

}

G[u].push\_back(v);

}

int ans = 0;

for (int i = 1; i <= n; i++) {

for (auto v : G[i])

dfn[v] = i;

for (auto v : G[i]) {

for (auto it : G[v]) {

if (dfn[it] == i)

ans++;

}

}

}

printf("%d\n", ans);

return 0; // 改数组大小!!!用pair改宏定义!!!

}

## 13.01BFS

// 问起点到终点转几次弯

// 如果不转弯，距离不变加到队首，否则距离更改加到队尾

struct node

{

int x, y, dir, step;

};

int dir[4][2] = { 1, 0, -1, 0, 0, 1, 0, -1 };

char str[N][N];

int n, sx, sy;

bool vis[N][N][4];

int bfs() {

deque <node> q;

for (int i = 0; i < 4; i++)

q.push\_back({ sx, sy, i, 0 });

while (!q.empty()) {

node now = q.front();

q.pop\_front();

int x = now.x, y = now.y;

int d = now.dir, w = now.step;

vis[x][y][d] = true;

if (str[x][y] == 'B')

return now.step;

for (int i = 0; i < 4; i++) {

int xx = x + dir[i][0];

int yy = y + dir[i][1];

if (xx < 1 || xx > n || yy < 1 || yy > n || str[xx][yy] == 'x' || vis[xx][yy][i])

continue;

if (i == d)

q.push\_front({ xx, yy, i, now.step });

else

q.push\_back({ xx, yy, i, now.step + 1 });

}

}

return -1;

}

int main()

{

cin >> n;

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= n; j++) {

cin >> str[i][j];

if (str[i][j] == 'A')

sx = i, sy = j;

}

}

printf("%d\n", bfs());

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

## 14.树上最小支配集

/\*对于图G = (V, E) 来说,最小支配集指的是从 V 中取尽量少的点组成一个集合,

使得 V 中剩余的点都与取出来的点有边相连。\*/

vector <int> G[N];

int dp[N][3];

void DFS(int x, int fa) {

dp[x][0] = 1;

int tot = INF;

bool ok = false;

for (int i = 0; i < SZ(G[x]); i++) {

int v = G[x][i];

if (v == fa)

continue;

DFS(v, x);

dp[x][0] += min(dp[v][0], min(dp[v][1], dp[v][2]));

dp[x][2] += min(dp[v][0], dp[v][1]);

if (dp[v][0] <= dp[v][1]) {

ok = true;

dp[x][1] += dp[v][0];

}

else {

Min(tot, dp[v][0] - dp[v][1]);

dp[x][1] += dp[v][1];

}

}

if (!ok)

dp[x][1] += tot;

}

int main()

{

int n;

cin >> n;

for (int i = 1; i < n; i++) {

int u, v;

sc("%d %d", &u, &v);

G[u].push\_back(v);

G[v].push\_back(u);

}

DFS(1, 0);

printf("%d\n", min(dp[1][0], dp[1][1]));

return 0; // 改数组大小!!!用pair改宏定义!!!

}

## 15.无向图入侵颜色

vector <int> G[N];

int f[N], n, m;

bool vis[N];

void init() {

for (int i = 1; i <= n; i++)

G[i].clear(), f[i] = i, vis[i] = false;

}

int find\_(int x) {

if (x != f[x])

x = f[x] = find\_(f[x]);

return x;

}

void Calc(int x) {

vector <int> link = G[x];

G[x].clear();

for (auto it : link) {

int v = find\_(it);

if (v == x)

continue;

f[v] = x;

vis[v] = true;

if (SZ(G[v]) > SZ(G[x]))

swap(G[x], G[v]);

G[x].insert(G[x].end(), ALL(G[v]));

}

}

int main()

{

int T; cin >> T;

while (T--) {

sc("%d %d", &n, &m);

init();

for (int i = 0; i < m; i++) {

int u, v;

sc("%d %d", &u, &v);

u++, v++;

G[u].push\_back(v);

G[v].push\_back(u);

}

int q;

sc("%d", &q);

while (q--) {

int x;

sc("%d", &x);

x++;

if (!vis[x])

Calc(x);

}

for (int i = 1; i <= n; i++)

printf("%d ", find\_(i) - 1);

puts("");

}

return 0; // 改数组大小!!!用pair记得改宏定义!!!

}

## 16.二维差分

int c[N][N], n, q;

int s[N][N];

void Add(int x1, int y1, int x2, int y2, int v) {

c[x1][y1] += v, c[x2 + 1][y2 + 1] += v;

c[x2 + 1][y1] -= v, c[x1][y2 + 1] -= v;

}

int Ask(int x1, int y1, int x2, int y2) {

return s[x2][y2] + s[x1 - 1][y1 - 1] - s[x1 - 1][y2] - s[x2][y1 - 1];

}

int main()

{

cin >> n >> q;

for (int i = 1; i <= q; i++) {

int x1, y1, x2, y2;

sc("%d %d %d %d", &x1, &y1, &x2, &y2);

Add(x1, y1, x2, y2, 1);

}

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= n; j++) {

c[i][j] += c[i - 1][j] + c[i][j - 1] - c[i - 1][j - 1];

}

}

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= n; j++) {

s[i][j] = c[i][j] + s[i - 1][j] + s[i][j - 1] - s[i - 1][j - 1];

}

}

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= n; j++) {

printf("%d%c", Ask(i, j, i, j), " \n"[j == n]);

}

}

return 0; // 改数组大小!!!用pair改宏定义!!!

}

## 17.O3优化

#pragma GCC optimize(3)

# STL

## 1.Vector

//将vec2拷贝进vec

vec = vec2;

//返回容器的最后一个元素

vec.back();

//返回容器的最后元素逆迭代器

vec.rbegin();

//返回容器的起始位置的逆迭代器

vec.rend();

//在插入位置插入若干元素 数量可选参数

vec.insert(插入位置迭代器, [数量], 元素);

//在vec的插入位置插入[vec2首迭代器, vec2尾迭代器)范围内元素

vec.insert(vec插入位置迭代器, vec2首迭代器, vec2尾迭代器);

//删除[首迭代器, 尾迭代器)范围内元素

vec.earse(首迭代器, [尾迭代器]);

//常数交换两vector元素

vec.swap(vec2);

//判断vec和vec2是否相等 不等!= 比较字典序< > <= >=

vec == vec2;

## 2.List

list<int>ls, ls2;

//将ls2拷贝进ls

ls = ls2;

//返回容器第一个元素

ls.front();

//返回容器最后一个元素

ls.back();

//判断容器是否为空 常数复杂度

ls.empty();

//返回容器元素个数 线性复杂度

ls.size();

//将链表ls2合并进ls 要求两链表升序 合并后ls2清空

ls.merge(ls2);

//常数时间将ls2连接至ls指定位置

ls.splice(位置迭代器, ls2);

//移除所有指定的元素 remove\_if(判别函数)移除判别成功的元素

ls.remove(元素);

//将链表反转 线性复杂度

ls.reverse();

//将链表除重 要求有序链表

ls.unique();

//对链表快排

ls.sort();

//将迭代器向后移动若干次 可以为负 无随机访问容器线性复杂度

advance(迭代器, 次数);

//返回两个迭代器之间的距离 无随机访问容器线性复杂度

distance(迭代器, 迭代器2);

## 3.Deque

//访问指定位置的元素 常数复杂度

dq[位置];

//返回容器元素数量 常数复杂度

dq.size();

//将容器末尾的元素删除 常数复杂度

dq.pop\_back();

//在容器头部插入一个元素 常数复杂度

dq.push\_front();

//将容器头部的元素删除 常数复杂度

dq.pop\_front();

## 4.Map

//返回对应键值的元素引用 无元素则创建

mp[键值];

//返回容器的元素数量 常数复杂度

mp.size();

//插入一个新元素 不使用C++11则make\_pair(键值, 元素);

mp.insert({ 键值, 元素 });

//删除[首迭代器, 尾迭代器)范围内元素

mp.earse(首迭代器, [尾迭代器]);

//删除等于改值的元素

mp.earse(值);

//查找对应键值是否出现 对数复杂度

mp.count(键值);

//返回对应键值元素的迭代器 没有精确匹配返回结束位置

mp.find(键值);

//返回第一个大于等于键值的元素迭代器

mp.lower\_bound(键值);

//返回第一个大于键值的元素迭代器

mp.upper\_bound(键值);

## 5.Multimap

//插入一个新元素

mp.insert({ 键值, 元素 });

//返回对应键值元素出现的个数 线性复杂度

mp.count(键值);

//删除所有等于改值的元素

mp.earse(值);

## 6.Unordered\_map

//需要引用头文件#include <unordered\_map> 无lower\_bound和upper\_bound其它同map

## 7.Set

//插入一个新键值

st.insert(键值);

//set其它操作同map 操作单键值

## 8.Multiset

//删除所有等于改值的元素

st.earse(值);

## 9.Unordered\_set

//需要引用头文件#include <unordered\_set> 无lower\_bound和upper\_bound其它同set

## 10.String

//返回s是否为空串 size复杂度不定

s.empty();

//在s尾部添加若干元素

s.append(数量, 元素);

//在s尾部增加s2的串

s += s2;

//将s从位置指定长度的部分替换为 s2也可以换为首迭代器和尾迭代器

s.replace(位置, 长度, s2);

//返回一个从指定位置开始指定长度的字符串副本 不指定长度到结尾 复杂度与长度成线性

s.substr(位置, [长度]);

//从指定位置开始查找s2串的出现位置 未找到返回-1

s.find(s2, [位置]);

//反向查找字符串s到指定位置返回出现位置 位置为正向位置

s.rfind(s2, [位置]);

//从输入流读取一行字符到字符串

getline(输入流, s);

//字符串转换为int形 长整型stoll 浮点型stod

stoi(s, [0], [进制]);

//整形/浮点型转字符串

s = to\_string([数值])

## 11.Bitset

bit.size() 返回大小（位数）

bit.count() 返回1的个数

bit.any() 返回是否有1

bit.none() 返回是否没有1

bit.set() 全都变成1

bit.set(p) 将第p + 1位变成1（bitset是从第0位开始的！）

bit.set(p, x) 将第p + 1位变成x

bit.reset() 全都变成0

bit.reset(p) 将第p + 1位变成0

bit.flip() 全都取反

bit.flip(p) 将第p + 1位取反

bit.to\_ulong() 返回它转换为unsigned long的结果，如果超出范围则报错

bit.to\_ullong() 返回它转换为unsigned long long的结果，如果超出范围则报错

bit.to\_string() 返回它转换为string的结果