Hello World ! ---poj

//博弈，计算几何另外搞, LCA模板有没有

多重背包模板题 二进制优化

题意：有分别价值为1,2,3,4,5,6的6种物品，输入6个数字，表示相应价值的物品的数量，问一下能不能将物品分成两份，是两份的总价值相等，其中>一个物品不能切开

思路：dfs或多重背包+二进制优化

void Multiple\_Pack()

{

for(int i = 1; i <= 6; i++) {

int j;

for(j = 0; num[i] >= (1 << (j+1)) - 1; j++) //拆成1,2,2^2,...,2^(k-1),M-2^k+1，且k是满足M-2^k+1 > 0的最大整数

One\_Pack(i\*(1 << j));

One\_Pack(i\*(num[i] - (1<<j) + 1));

}

}

最大矩阵和 经典DP

题意：最大子序列和的扩展。

思路：枚举每一行为起始行，枚举终点行，把这个区域压成一维，运用最大子段和求该区域的最大矩阵和。

int calc(int \* tmp)

{

int sum = 0, ans;

memset(&ans, 128, sizeof(int));

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

sum += tmp[i];

ans = max(ans, sum);

if(sum < 0) sum = 0;

else sum = max(sum, tmp[i]);

}

return ans;

}

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

for(int j = 1; j <= n; j++) tmp[j] = 0;

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

for(int k = 1; k <= n; k++) tmp[k] += a[j][k];

int sum = calc(tmp);

ans = max(ans, sum);

}

}

/\*

差分约束 好题 隐含条件

题意：给出n个闭区间[ai,bi]，每个区间对应一个ci，表示集合Z在区间[ai,bi]内ci个相同元素，问集合Z至少有几个元素

思路：就说说隐含条件吧，0 <= ai - ai-1 <= 1 ，我只想到 要小于等于1，没有想到要大于等于0，就WA了，这个点得记住，不然没办法当场思考出>来.

由输入建边简单，难的是要发觉隐含条件，就是这是个`数列区间`，有0 <= s[i+1] - s[i] <= 1(s[i]表[0,i]有多少个Z元素)这个隐含条件...

an - a0 >= 0

an >= a0 + 0

av - au-1 >= k

av >= au-1 + k

ai - ai-1 <= 1

ai-1 >= ai - 1

ai - ai-1 <= 0

ai-1 >= ai + 0

\*/

/\*

数论 本原勾股数组

题意：求解N以内有多少个本原勾股数组

本原勾股数组 : 满足a^2 + b^2 = c^2且a,b,c的最大公约数为1

思路：根据下面的叙述实现，枚举s,t,判断一下

看下面的证明有两点疑惑：

(1)证明a、b奇偶性不同；

(2)证c-b与c+b互质？

摘:

本章主要讨论的是勾股数组，也就是关于满足a^2+b^2=c^2的三元组（a,b,c）的问题。

其实，对于勾股数组的个数进行讨论并没有多大意义，因为已知a，b，c为勾股数组，那么显然有da，db，dc(d>0)也为勾股数组，因为(da)^2+(db)^2=d^2(a^+b^2)=d^2c^2=(dc)^2。

因此着重研究的是关于本原勾股数组的问题，本原勾股数组即为a，b，c为勾股数组且满足a，b，c的最大公约数为1。

对于本原勾股数组，显然a和b奇偶性不同只需要将a=2x+1,b=2y+1,c=2z代入a^2+b^2=c^2即可推出有公约数2。由于a和b奇偶性不同，那么显然c为奇>数。

那么最关心的是如何求出所有的本原勾股数组。

如果将公式转化一下，得到a^2=c^2-b^2=(c+b)(c-b)，那么显然有，c+b和c-b没有公因子，用反证法证明：

如果d|(c+b)且d|(c-b)，那么显然有d|((c+b)+(c-b))和d|((c+b)-(c-b))即d|2b且d|2c，因为在定义本原勾股数组的时候已经有了b和c的最大公约数

是1的约定(虽然定义是a，b，c最大公约数是1，但是如果gcd(b,c)=d>1，显然有d|c不满足定义)。所以d要么为1，要么为2。但是如果d=2时，那么显然a，b，c均为偶数，不满足定义，那么d只能为1，证明了c+b和c-b没有公因子。

因此，如果将a^2进行质因数分解，那么会有a=p1^t1\*p2^t2\*p3^t3...pn^tn，其中指数t1,t2,t3...tn为偶数（因为这样才能保证a^2开根号后为整数

），又因为c+b和c-b没有公因数，所以c+b和c-b各取a分解后的其中某一些pk^tk，因此，c+b和c-b均为平方数。那么假设c+b=s^2,c-b=t^2，则有c=(s^2+t^2)/2,b=(s^2-t^2)/2，a=st，因此形如(st,(s^2-t^2)/2,(s^2+t^2)/2)的三元组为本原勾股数组。(其中s和t都是奇数，因为如果s和t中有且只有一个为奇>数，那么显然(s^2+t^2)/2不会是整数，而如果两个数都是偶数，那么显然该三元组有公因子2，与本原勾股数组定义矛盾。)

\*/

bool check(int a, int b, int c)

{

return a\*a+b\*b==c\*c && gcd(c, gcd(a, b)) == 1;

}

int main()

{

while(scanf("%d", &n) != EOF) {

int p = 0, ans = 0, a, b, c;

memset(vis, false, sizeof(bool)\*(n+5));

for(int oo = 2\*(int)(sqrt(n+0.5)), s = 1; s <= oo; s += 2)

for(int t = 1; t < s; t += 2) {

int a = s\*t, b = (s\*s-t\*t)/2, c = (s\*s+t\*t)/2;

if(c > n) break;

// if(gcd(s, t) != 1) continue;

if(check(a,b,c)) {

ans++; int k = 1;

while(c\*k <= n) {

vis[a\*k] = vis[b\*k] = vis[c\*k] = true;

k++;

}

}

}

for(int i = 1; i <= n; i++) p += !vis[i];

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

}

return 0;

/\*

线段树 扩点 扩线 区间覆盖

题意：给出n条平行于Y轴的线段(y', y'', x)，然后3条一组，问有多少组可见线段组。“可见”的定义为，两条线段能由一条水平先连接但不交于其它>的线段。“可见线段组”的定义为该组内的3条线段两两可见。

心得：之前看poj\_3225，死活看不懂，跳过了。做这道题，它们的离散化方法是一样的：线段 = 一个闭区间 + 两个端点。而线段树是只存点的，为了

解决这个问题我们可以把这个闭区间也化成一个点，这样就可以存入线段树里了。而线段 \* 2这一方法正好可以实现这一点。如下图，

区间[0,2]离散化后就变成这么些点:

真实的线段: [0], (0,1), [1], (1,2), [2] ...

化为线段树里对应的点: 0 1 2 3 4 ...

当然也可以用poj\_2528这样的方法来做，不过就是要处理的东西多些，麻烦些。poj\_2528的解法是应对区间过大时防止MLE/TLE的解法，跟这道题>不一样，这道题的重点是同化区间和点，poj\_2528的重点是缩短区间的长度。

\*\*\* 偶数点代表点，奇数代表线段，遇到有线段类的题目（用线段树做）经常要考虑乘以2，表示浮点的线段 \*\*\*

思路：

下面摘抄自大牛们的blog:

"能明显的感觉到是区间覆盖问题了。但是有一个细节问题，就是中间的水平线不一定经过整点，所以即使这个区间的所有点都被覆盖，也不能说其就>不能穿过一条线，于是，可以将所有线段的长度扩大至2倍，这样就解决了这个问题。"

"从左到右，一次对每条线段，先进行查询，看左边能看见多少条线段，然后进行覆盖，因为很明显，如果一条线段能看见另一条线段，那么这个关系>必然是相互的，所以对每条线段，只需要往左看就行了"

"注意：如 样例 中 ，0 2 2,3 4 2这两条线段，可以看到 2 3之间是没有被覆盖的，但是在线段树中我们看不到这条线段，因为 变成 浮点数了，不>能处理，那么我们可以将 坐标 x2，这样就变成 4 6，中间就多出一个点 5 了，就可以判断了。。

偶数点代表点，奇数代表线段，遇到有线段类的题目（用线段树做）经常要考虑乘以2，表示浮点的线段。。。poj 3225这题类似"

"注意:由于线段包括端点,如果这条直线y=c刚好穿过某条线段的端点,则情况会变得有些麻烦.可以采用这种方法来做:将上下纵坐标乘2,横坐标不变,改

变原来[y,y+1)的节点存储方式,变为[y,y]式,这样,就可以简单地处理端点问题,并且它对于所有情况都有很好的效果.自己画个图就明白了."

\*/

/\*

树的同构 树的最小表示法

题意：判断两棵树是否同构。输入给出的是dfs的方向，0表下去，1表上来。

思路：比较树的最小表示即可。

树的最小表示跟串的最小表示在做法上其实没什么关系。。。

树的最小表示：将每棵子树用0/1串表示，然后按字典序把他们串起来，最终根的连接串如果一样，则同构。

byvoid有向树与树的括号序列最小表示法：https://www.byvoid.com/blog/directed-tree-bracket-sequence

\*/

string min\_Presentation(string s)

{

vector<string> v;

int sum = 0, len = s.length(), st = 0;

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

if(s[i] == '0') sum++;

else sum--;

if(sum == 0) {

if(i-1 == st+1) v.push\_back("01"); //叶子节点

else v.push\_back("0" + min\_Presentation(s.substr(st+1, i-st-1)) + "1"); //记住把首尾提取出来

st = i + 1;

}

}

sort(v.begin(), v.end());

string res = "";

for(int Size = v.size(), i = 0; i < Size; i++) res += v[i]; //按字典序串起来

return res;

}

int main()

{

while(cases--) {

cin >> s1 >> s2;

cout << (min\_Presentation(s1) == min\_Presentation(s2) ? "same" : "different") << endl;

}

return 0;

}

/\*

求树的重心

树的重心定义：删掉重心g后，原来的树分成了若干棵子树，要使得这些子树的节点数最大的最小，即Min( Max{ sum\_of\_nodes(subtree) }

思路：树形DP

对于点g，删除g后，剩余有son[k]可子树和一棵向上子树，节点数分别为nodes\_of\_son[i] 和 n - sum{nodes\_of\_son}，取最大值

\*/

void dfs(int u)

{

visit[u] = true;

son[u] = 1;

Max[u] = 0;

for(int k = head[u]; k != -1; k = edge[k].next) {

int v = edge[k].v;

if(visit[v]) continue;

dfs(v);

son[u] += son[v];

Max[u] = max(Max[u], son[v]);

}

Max[u] = max(Max[u], n - son[u]);

}

/\*

判断一个图的最小生成树是否唯一

题意：判断一个图的最小生成树是否唯一

思路：判断一个图的最小生成树是否唯一，可以求其次小生成树。如果它的次小生成树权值之和等于最小生成树权值之和，则最小生成树不唯一，否则最小生成树唯一。

求次小生成树我的方法是O(N^2 + M)的。首先求出最小生成树，记录权值之和为MinST。然后枚举添加边(u,v)，加上以后一定形成一个环，找到环上非(u,v)边的权值最大的边，把它删掉，计算当前生成树的权值之和，取所有枚举加边后生成树权值之和的最小值，就是次小生成树。

实际上具体更简单的方法为从每个顶点i，遍历整个最小生成树，定义F[j]为从i到j的路径上最大边的权值，用O(N)的方法求出每个F[j]。然后枚举顶点i的邻域，遍历每条不再最小生成树中的边(i,j)，加上这条边，并删除环上最大边(非(i,j))，新的生成树权值之和为MinST + w(i,j) - F[j]，记录其最小值即可，时间复杂度为O(N^2 + M)。求最小生成树可以用最简单的Prim，时间复杂度为O(N^2)，用更好的算法是没有意义的。

这种方法比起那种求出最小生成树后，枚举删除最小生成树上每条边，然后再求最小生成树的方法应该要快些，因为那种方法的时间复杂度为O(N ( N logN + M) )(Prim + Heap)。

本来想写得优美一点的，谁知道越写越臃肿 =\_= 看上面摘抄的分析就可以了，别看代码=\_=

\*/

void dfs(int u, int pre, int d)

{

dis[u] = d;

for(vector<Pii> :: iterator it = adj[u].begin(); it != adj[u].end(); it++)

if(it->first != pre) dfs(it->first, u, max(d, it->second));

}

void prim(int s)

{

memset(used, false, sizeof(used));

memset(dis, 127, sizeof(dis));

memset(vis, false, sizeof(vis));

for(int i = 1; i <= n; i++) adj[i].clear();

while( !Q.empty() ) Q.pop();

Q.push( pair<Pii, int>(Pii(0, -1), s) );

dis[s] = 0;

while( !Q.empty() ) {

int u = Q.top().second; int tk = Q.top().first.second; Q.pop();

if(vis[u]) continue; else vis[u] = true;

if(tk != -1) {

adj[u].push\_back( Pii(edge[tk].u, edge[tk].w) );

adj[edge[tk].u].push\_back( Pii(u, edge[tk].w) );

used[tk] = used[tk^1] = true;

}

for(int k = head[u]; k != -1; k = edge[k].next) {

int v = edge[k].v, w = edge[k].w;

if(!vis[v] && dis[v] > w) dis[v] = w, Q.push( pair<Pii,int>(Pii(w, k), v) );

}

}

mst = 0;

for(int i = 1; i <= n; i++) mst += dis[i];

}

bool solve()

{

prim(1);

for(int i = 1; i <= n; i++) if(dis[i] == INF) { mst = 0; return true; }

int ans = INF;

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

dfs(i, i, 1);

for(int k = head[i]; k != -1; k = edge[k].next) if(!used[k]) {

int v = edge[k].v, w = edge[k].w;

ans = min(ans, mst + w - dis[v]);

}

}

return ans != mst;

}

/\*

连通分量 好题 启发题 二分匹配运用

题意：N个男人N个女人，给出男人喜欢的女人编号，只要男人喜欢女人，他们就可以结婚。每个人都有唯一的结婚对象。

下面给出一个可行的匹配方案，即一个女人对应的男人一定是喜欢她的。现问所有合法方案。合法方案就是，每个人有唯一的结婚对象，且该女

人是男人喜欢的。

思路：这题一个突破口是那个合法的方案！

先写出算法，再说思路：对于给出的可行方案(boy,girl)，将这两人合成一个大点。然后对于大点u(boy,girl)和大点v(boy',girl')，如果boy>喜欢girl'，则连条边(u --> v)。最后求强连通分量，同一个连通分量里的所有男人都可以娶到连通分量里的所有的女人。

如果题目没有给出第一个可行方案怎么办？先二分匹配搞出来一个。

http://www.gonglin91.com/poj-1904-kings-quest/

\*/

/\*

摘：

题意：有n个男人和n个女人，下面n行依次给出每个男人喜欢那些女人（男人和女人都分别从1到n编号），这些男人最后只能跟他喜欢的女人结婚。另>外一行内给出n个数字，表示一种匹配方案，第i个数字m[i]，表示m[i]这个女人要嫁给i这个男人。注意，这个方案是一个合法的方案，即一个女人一定对>应一个男人，而且这个女人一定是上面给出的，这个男人喜欢的女人之一。问题是，除了这个合法的方案，还没有没其他合法的方案，合法的方案就是，每

个男人都可以娶到女人，每个女人只会嫁一个男人

这题一个突破口是那个合法的方案。如果男人u喜欢女人v，就添加一个有向边u->v，在合法方案中，将女人a嫁给了男人b，那么添加一条有向边a->b

我们来考虑一下这个图，这个图至少存在n个小环，因为可行方案产生了n对合法的夫妻，一对夫妻之间形成一个小环。

暂且把这n个小环看做N个点。看看点和点之间有没有边，如果有边，一定是一条男人指向女人的边，因为我们在添加有向边的时候，女人指向男人的边

只有n条，就是那个合法方案产生的n条，这n条边都在小环里面了，所以其他边一定是男人指向女人的边。在这N个点中，可不可能形成环内，是可能的。如

果这N个点不形成环会怎样？从A这个点出发，最终回不到A这个点！从A出发，指向B，意思就是A里面的那个男人，指向了B里面的女人（大点之间的边一定>是男人指向女人的边），那么A里面的女人就没人匹配了，接着如果B指向C，说明B里面的男人指向C里面的女人，那么B里面的女人又没人匹配了………………

说明就很明显了，对于一个大点X，如果X可以指向别的点Y，并且选择了这条边的话，那么说明X里面的男人选择了Y里面的女人，那么X里面的女人必定

没人匹配了，并且Y里面的男人也是没人匹配的，那么Y男人一定要到别的点找女人，也必定导致那个点的男人无法匹配，以此类推…..一个点的男人夺走了>其他点的女人那么那个点的男人一定要去别处找女人，唯一完满的结局就是某个男人找到了第1个女人，别忘了那个女人还没人匹配。这样才能结束这个死>循环。

所以如果大点能形成环，那么环内的大点里面的男人就有了不止一种选择，并且做出不同的选择都会合法。再说得直白一点就是，对于一个连通分量里

面的男人，它可能选择这个连通分量里面的任意一个女人（但是这个女人必须是他喜欢的，有的女人他不喜欢但也可以到达），并且保证合法

其实这题给出了一组可行解，就是一个最大的突破口，试想就算没给出这组可行解一样可以问同样的问题，但是恐怕就没那么好想了。如果真的没有给

出可行解，可以运行一次最大匹配，那么就找到了一组可行解，就变为了原题

注意最后答案是要排序再输出的

\*/

/\*

二维DP

题意：n(n<=40)根棍子，长为ai(ai<=40)，连成一个三角形，使得面积最大（要所有棍子都参与）。

思路：一开始像判定性DP那样预处理出boo[i]是否能到达，然后枚举两条边长搞，这样是错的。如数据： 3 8 23 90 就会WA。因为dp[n]=true,dp[m]=true(n>m)，不代表dp[n-m]也等于true！

看数据范围很小。令dp[i][j]表示第一条边长为i，第二条边长为j是否可达。

则dp[i][j] = dp[i][j] || (j>=w&&dp[i-w][j]) || (j>=w&&dp[i][j-w])

二维DP，二维DP都长怎样呢？

\*/

memset(f, false, sizeof(f));

f[0][0] = true;

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

for(int j = half; j >= 0; j--)

for(int k = j; k >= 0; k--)

f[j][k] = f[j][k] || (j>=a[i] && f[j-a[i]][k]) || (k>=a[i] && f[j][k-a[i]]);

/\*

思路题 序列DP

题意：一个数列，n个数，找三个 m个连续数的子数列，使其和最大。

思路：找3个连续子数列。那其实我们可以一个一个找，在找了第一个的基础上找第二个，在找了第二个基础上找第三个。

dp[1][i] 表在1-i中找第一个m子序列的最优值

dp[2][i] 表在1-i中找第二个子序列的最优值

dp[3][i] 表在1-i中找第三个子序列的最优值

dp[2][i] = dp[1][j] + sum{a[i-m+1],a[i-m+2]...a[i]} , j <= i-m

改方程需要枚举j，复杂度很大，但正如之前所说，因为我们找第2个m序列是建立在找了第1个m序列的基础上的。我们可以用O(n)的时间预处理>初个pre[i]记录1-i中最大的dp[1][j](1 <= j <= i)，这样，我们就不用枚举j了，直接就dp[2][i] = dp[1][pre[j-m]] + sum{a[i-m+1],a[i-m+2]...a[i]}

\*/

/\*

树状数组应用 两个树状数组 好题

题意：n个牛站一排，已知它们的声调v[i]和座标x[i]。两只牛i,j想沟通的话，它们必须用同一个声调max(v[i], v[j])，沟通起来耗费的能量为max(v[i], v[j]) \* dis(x[i], x[j]). 问这n\*(n-1)/2对的牛都交流，总共要耗费多少能量。注意，同一只牛和不同牛交流所用的声调不一定都相同。

思路：按声调排序，从小到大一个一个插入树状数组，插入牛i之前，先计算之前插入的牛相对于当前牛i的总距离dis，因为当前声调最高的是牛i，所

以总ans += dis \* v[i].

然后就自己卡在如何处理相对距离上了，不知道怎么处理。

看了Discuss后才知道可以用两个树状数组来解决这个问题。再开另一个依然以座标为下标，但元素值表示“出现的次数”的树状数组...表述不太

好，就是说Query(x)返回的是当前座标区间[1,x]中的牛的数目，这样就解决了之前卡住的相对距离的问题了。

\*/

int main()

{

long long tot = 0;

scanf("%d", &n);

for(int i = 0; i < n; i++) scanf("%d%d", &cow[i].first, &cow[i].second);

sort(cow, cow+n);

long long ans = 0;

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

long long tmp = Query(cow[i].second, c); //[1,x] 的总距离

int m = Query(cow[i].second, num); //[1,x] 出现的牛的数目

ans += (cow[i].second \* m - tmp) \* cow[i].first + ((tot - tmp) - (cow[i].second \* (i - m))) \* cow[i].first;//根据相对距离算结果

Update(cow[i].second, cow[i].second, c);

Update(cow[i].second, 1, num);

tot += cow[i].second;

}

printf("%I64d\n", ans);

return 0;

}

/\*

状压 记忆化搜索 好题

题意：经典邮递员问题。题意：给出n个顶点m条边的无向图。从其中某个顶点出发将每条边遍历至少一次回到出发点。求最短路。

思路：欧拉回路的每个点的度必为偶数。所以只要将所有度数为奇数的点找出，将其两两之间的最短路求出，状态压缩记忆化搜索即可。度数为奇数的

点必为偶数个。因为每条边连两个顶点，所以所有点的总度数为偶数，假设奇度的点有x个，则偶度得点n-x个，可以x必为偶数。所以只要在奇度点之间每>两个连一条边即可。

简单证明原图种度数为奇数的点必为偶数个：原图总度数tot = 边数|E| \* 2，是偶数，所以...

\*/

void Update\_min(long long &x, long long y) { if(x == -1) x = y; else if(x > y) x = y; }

long long DP(int S)

{

if(S == 0) return 0;

if(f[S] != -1) return f[S];

for(int i = 0; i < n; i++) if(S & (1 << i)) {

for(int j = i+1; j < n; j++) if(S & (1 << j)) {

int y = S ^ (1 << i) ^ (1 << j);

Update\_min(f[S], b[i][j] + DP(y));

}

break; //TODO 跟lightoj\_1018一样，这里可以用break剪枝的，原因相同，在lightoj\_1018.cpp有写

}

return f[S];

}

int main()

{

int cases, Cas = 0, m, x, y, z;

scanf("%d", &cases);

while(cases--) {

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

init();

long long sum = 0;

while(m--) {

scanf("%d%d%d", &x, &y, &z);

a[y][x] = a[x][y] = min(a[x][y], z);

deg[x] ++ , deg[y] ++ ;

sum += z;

}

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

for(int i = 1; i <= n; i++) if(i != k)

for(int j = 1; j <= n; j++) if(j != i && j != k)

a[i][j] = min(a[i][j], a[i][k] + a[k][j]);

int Index = 0;

for(int i = 1; i <= n; i++) if(deg[i] & 1) {

odd[Index++] = i;

}

for(int i = 0; i < Index; i++)

for(int j = i+1; j < Index; j++)

b[i][j] = b[j][i] = a[odd[i]][odd[j]];

n = Index;

printf("Case %d: %lld\n", ++Cas, DP((1 << n) - 1) + sum);

}

return 0;

}

/\*

期望DP 概率DP

题意：一个bug仅属于一个系统和一个分类，现有n个系统和s个分类，一天可以找出一个bug，问找到的bug覆盖所有系统和分类所期望的天数。

思路：可以把该题建成一个有向图：状态(i,j)表已找出的bug覆盖i个系统j个分类，e(i,j)在图里作为一个节点，在数学上表示(i,j)状态到(n,s)的期

望，所以e(n,s) = 0, 目标是求e(0,0)；概率关系看作有向边，易知这是个有自环的图。

因为想用递推直接搞，而递推的条件是图无环，我们可以通过一些数学上的方法把自环消掉。直接上公式了。

E(T) = c + Σ(p\*E(S)) 无环图,c表示一步的代价，p表示S->T的概率

(1-p')E(T) = c + Σ(p\*E(S)) 有自环图，p'表示自己到自己的概率

我是直接上递归的，当然DP的比较快。

e(i,j) = p(i,j)\*e(i,j) + p(i+1,j)\*e(i+1,j) + p(i,j+1)\*e(i,j+1) + p(i+1,j+1)\*e(i+1,j+1)

\*/

#include <stdio.h>

#define MAXN 1005

#define ZERO 1e-7

int n, s;

double e[MAXN][MAXN];

double calc(int i, int j)

{

if(e[i][j] + 1 > ZERO) return e[i][j];

e[i][j] = 1;

if(i + 1 <= n && j + 1 <= s) e[i][j] += calc(i+1, j+1) \* ((1-i\*1.0/n)\*(1-j\*1.0/s));

if(i + 1 <= n) e[i][j] += calc(i+1, j) \* ((1-i\*1.0/n)\*(j\*1.0/s));

if(j + 1 <= s) e[i][j] += calc(i, j+1) \* ((i\*1.0/n)\*(1-j\*1.0/s));

e[i][j] /= (1 - 1.0\*i/n\*j\*1.0/s);

return e[i][j];

}

int main()

{

scanf("%d%d", &n, &s);

for(int i = 0; i <= n; i++) for(int j = 0; j <= s; j++) e[i][j] = -1;

e[n][s] = 0;

printf("%.4f\n", calc(0, 0));

return 0;

/\*

中国剩余定理

题意：给A,B,C,k，求最小非负数t，使得(A + C\*t) = B mod 2^k

思路: (A + C\*t)= B mod 2^k

C\*t ≡ B - A(mod 2^k)

C\*x + (2^k)\*y = B-A; (只面熟ax+by=n形式=\_=)

欧几里得求ax+by=d的x0

ax+by=n的任一解x1 = x0\*(n/d)

任意解x = x1 + b/d\*i

最小解x\_ = (x1 % (b/d) + b/d) % (b/d)

\*/

#include <stdio.h>

typedef long long ll;

ll EGCD(ll a, ll b, ll &x, ll &y)

{

if(b == 0) {

x = 1, y = 0;

return a;

}

ll d = EGCD(b, a%b, x, y);

ll t = x; x = y, y = t - a/b\*y;

return d;

}

int main()

{

ll A, B, C, m, n, d, x, y, t;

while(scanf("%I64d %I64d %I64d %I64d", &A, &B, &C, &m), A||B||C||m) {

m = (1LL << m);

n = B - A;

d = EGCD(C, m, x, y);

if(n % d != 0) { puts("FOREVER"); continue; }

t = m/d;

x = x\*(n/d); //x0是C\*x + m\*y = d的解, x才是C\*x+(2^k)\*y = B-A的解

x = (x % t + t) % t;

printf("%I64d\n", x);

}

return 0;

}

/\*

最小费用最大流 基础题

题意：无向带权图G(V, E)，要从点1走到点n再走回来1，期间不能重复走过同一个点，求最短路径。

思路：建图：只能经过一次，所以边的容量为1

边的长度为费用

因为其实是求两条最小增广路，为了控制要“两条”，可以设一个超级源点和1相连，容量为2，或者超级汇点和n相连

估计题目保证了至少有两条不相交路径；无负权回路(因为我的SPFA都没有写回路的判断就AC了)

启发：求一条最短路用最短路算法

求两条可相交的最短路有次短路

求多条不可相交的路径就用最大流了...因为最大流中，容量可以控制能经过多少次.

\*\*\*\*\*\*\*\*\*\*\*最小费用最大流的一些记录 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

最小费用最大流MCMF的思想就是Ford-Forkson，跟EK求最大流相似，不断寻找增广路.

要说和EK的不同点，就是费用流用的是最短路算法来增流，而EK因为不用考虑费用，所以只要BFS就可以增流。EK出来的增广路是“无序”的，MCMF出来>的增广路是按费用排序了的.

有个细节，EK的BFS里只要到达了sink就可以return，因为到达了sink就算找到了一条增广路；但是MCMF得结束了SPFA才可以.

pre[]数组用来记录增广路径，指向的是边。那能不能像之前那样，用pre指向点来记录呢？

Answer: 不能，因为这里不是用邻接矩阵来记录流的，是用邻接表，即直接在边上面记录，所以用pre[]无法逆向回src. 这跟是不是费用流无关，是看

你用什么数据结构来存而已。

struct Edge { int u, v, value, f, next };

对于每条边<u, v, c, f>我们都要建立其逆向边<v, u, -c, 0>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\*

概率DP

题意：在ACM比赛种出题，共M道题，T个队，要求冠军出的题数不少于N，问每队至少过一题且冠军至少N题的概率。

思路：易知每个队至少出一题的概率是相互独立的。p[i][j]表示队i出j题的概率，g[i][j][k]表示队i对于前j题而言作出k题的概率。

g[i][j][k] = g[i][j-1][k-1] \* p[i][j] + g[i][j-1][k] \* (1 - p[i][j])

则对于队i，至少作出1题的概率为 1 - g[i][M][0].

则每个队至少作出1题的概率为 ans = ∏(1-g[i][M][0])

接着考虑至少有一队过N题的概率 = ans - 所有队都没过N题的概率

= ans - ∏( Σ g[i][M][j] ) , j = 1,2,...,N-1

\*/

/\*

欧拉回路 经典题

题意：每条边经过刚好两次，最后回到起点，输出路线

思路：dfs

\*/

void dfs(int u)

{

while(head[u] != -1) {

int k = head[u];

int v = edge[k].v;

head[u] = edge[k].next;

dfs(v);

}

printf("%d\n", u); //TODO 不解，放在后面就对，放在前面就WA。。

}

int main()

{

int m, u, v;

while(scanf("%d%d", &n, &m) != EOF) {

init();

while(m--) {

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

add\_edge(u, v), add\_edge(v, u);

}

dfs(1);

}

return 0;

}

[**第 K 最短路 问题**](http://www.cnblogs.com/acSzz/archive/2012/07/31/2616268.html)

在这求第  k 短路用的是，A\*+dij 所谓的A\* 是一种启发式搜索，他给搜索选定一定的方向，避免了，无谓的搜索，如何来确定搜索的顺序？..也就是用一个值来表示这个值为f[x]..每次搜索取f[x]最小的拓展...那么这个f[x]=h[x]+g[x]其中这个h[x]就是当前搜索时的代价..如求K段路这个就是前一个点的h[x']+边的长度...而g[x]是一个估价函数..估价函数要小于是对当前点到目标的代价的估计..这个估计必须小于等于实际值~~否则会出错...A\*的关键也就是构造g[x]..，我们的dij 算法。最短路是，就是一种 A\* 搜索，其中 g[x]=0;

而这里要说的求K短路一种方法..就是用BFS+A\*来搜索的过程...g[x]的设定为到这个点到目标点的最短路（这个可以用dij 一次求出） 径...显然其实小于等于实际值的...h[x]就是搜索到这个点的代价（及走过的路程）..用一个优先队列来做..每次取出h[x]+g[x]最小的点来拓展...拓展 也就是通过这点来更新其能直接经过一条边到达的点..这里做好一个新点就丢进优先队列里去..反正总会从对首弹出h[x]+g[x]最小的点..

首先，我们在放优先队列的是这样的节点，他包括，从原点 到达本节点 的路径长度  len ，然后我们在优先列里，按照 len +dis[i] （dis 到达终点的最最短路）的最小，优先级 排队，

那么当我们第一次搜索到 E 点时，这时肯定是 最短路径，第二次取出的，就是第二短路，以此类推，

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | struct cnode  {  **int** u;  **int** len;      cnode (**int** uu,**int** ww):u(uu),len(ww){}      friend **bool** operator < (cnode a,cnode b)      {          return a.len+dis[a.u]>b.len+dis[b.u];      }  }; |

　　我们从一个点，如何扩展的下一个点呢，就是，将与相连的点 ，入队列，（ 这样 就会走重复的边 ）

       队列里面的节点就是，原点 经过 len 的路经 所到达的状态

1 int A\_star(int s)

2 {

3 int i;

4 if(dis[s]==inf)return -1;//这个一定要有，若s到t不连通的话，下面会 死循环

5 priority\_queue<cnode>que;

6

7 CL(tol,0);

8 que.push(cnode(s,0));

9 while(!que.empty())

10 {

11 cnode a=que.top();que.pop();

12 int u=a.u;

13 int len=a.len;

14 tol[u]++;

15 if(tol[t]==k)return len;

16

17 for(i=0;i<g[u].size();i++)

18 {

19 node b=g[u][i];

20 int v=b.u;

21 que.push(cnode(v,len+b.w));

22 }

23

24 }

25 return -1;

26 }

那要是本身就不存在K短路呢？？那就是e拓展不到K但是其他点很有可能一直打圈圈无限下去...这里就要用个条件来判断一 下...首先在找某个点作为优先队列头出现了几次就用了一个计数器times[]..所求的点times[e]==k就代表得到了解..如果当前想拓展的 点times[]>k就没必要拓展了..因为这个点已经是求到k+1短路了..从这个点继续往下搜肯定得到的是大于等于k+1短路的路径...就像 1->2有3条路..2->3有2条路..那1->3有6条路的概念差不多..没必要对其进行拓展了..

注意 ：

  若是 有向边时，我们求 到终点的最短距离时，要将边 反向

vector<int> v[MAXN], rv[MAXN], e[MAXN], re[MAXN];

class KNode {

public :

int u, len;

KNode(int u, int len) : u(u), len(len) { }

friend bool operator < (const KNode &a, const KNode &b) {

return a.len + dis[a.u] > b.len + dis[b.u]; //

}

};

int main()

{

int x, y, z, m, start, end;

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

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

scanf("%d%d%d", &x, &y, &z);

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

e[x].push\_back(z);

rv[y].push\_back(x);

re[y].push\_back(z);

}

scanf("%d%d%d", &start, &end, &K);

if(start == end) K++;

dijkstra(end, start);

if(dis[start] == INF) {

printf("-1\n");

return 0;

}

printf("%d\n", A\_star(start, end));

return 0;

}

int A\_star(int start, int end)

{

memset(times, 0, sizeof(times));

priority\_queue<KNode> Q;

while(!Q.empty()) Q.pop();

Q.push(KNode(start, 0));

while(!Q.empty()) {

KNode now = Q.top();

Q.pop();

int u = now.u;

int len = now.len;

times[u]++;

if(times[end] == K) return len;

if(times[u] > K) continue; //

for(int Size = v[u].size(), i = 0; i < Size; i++) {

int tmp = v[u][i];

Q.push(KNode(tmp, len+e[u][i]));

}

}

return -1;

}

void dijkstra(int s, int t)

{

deque<int> Q;

Q.clear();

for(int i = 0; i <= n; i++) dis[i] = INF;

memset(visit, false, sizeof(visit));

dis[s] = 0;

visit[s] = true;

Q.push\_back(s);

while(!Q.empty()) {

int now = Q.front();

Q.pop\_front();

visit[now] = false;

for(int Size = rv[now].size(), i = 0; i < Size; i++) {

int tmp = rv[now][i];

if(dis[now] + re[now][i] < dis[tmp]) {

dis[tmp] = dis[now] + re[now][i];

if(!visit[tmp]) {

visit[tmp] = true;

Q.push\_back(tmp);

}

}

}

}

}

/\*

2-SAT 二分求最优

题意：给出 n 对钥匙，每对只能挑一把使用，每把只能用一次，当一对钥匙中的一把被使用后，另一把也就不能再用了；然后给出 m 道门，每个门都

有两把钥匙可以打开，问最多能开几道门（按给出的顺序开）

思路： 对于一扇门所需钥匙(x,y)，它们的矛盾关系边应该是什么呢？因为“每扇门有两把钥匙可以打开”，并且“一定要其中一把钥匙”，所以，矛>盾边应该是x'--y'，即表示钥匙x和钥匙y都没选取到(x'表没有选x)

二分答案，2-sat验证

http://blog.csdn.net/l04205613/article/details/6670249

\*/

/\*

连通性 经典题

题意：判断一个有向图是否弱连通，即对于任意两点(u,v)存在一条路径使得u能到达v或者v能到达u

思路：缩点+判单链

缩点后如果有两个或两个以上的点出度为0或者入度为0，则false

缩点后的图是不是单链。如果不是单链，即有分叉，则根据缩点后的图的性质可知，两个分叉上的点不能到达，false

判单链可以暴力dfs，可以拓扑排序，可以DP最长链。我写了前面两种。

\*/

/\*

线段树 好题 好巧妙

题意：人们按顺序来排队或插队，按顺序给出每个人要插入的位置p，即当前者要在队列里第p个人后面；还有每个人的编号。输出最后的队列的编号序

列。

思路：记得以前做过一遍了，可是现在再自己做一遍还是不会。太弱了...

就不分析了。建一棵空树，每个节点保存该区间的空位。从后往前插入，p[i]的意义就变成“第i个人站的位置前面要有p[i]个空位”

\*/

void Update(int u, int x)

{

f[u].rest--;

if(f[u].l == f[u].r) {

f[u].num = y;

return ;

}

if(f[L(u)].rest > x) Update(L(u), x);

else Update(R(u), x - f[L(u)].rest);

}

/\*

线段树 约瑟夫环问题 好题 反素数

题意：N个孩子顺时针坐成一个圆圈且从1到N编号，每个孩子手中有一张标有非零整数的卡片。第K个孩子先出圈，如果他手中卡片上的数字A大于零，>下一个出圈的是他左手边第A个孩子。否则，下一个出圈的是他右手边第(-A)个孩子。第p个出圈的孩子会得到F(p)个糖果，F(p)为p的因子数。求得到糖果>数最多的是哪个孩子及得到多少糖果。

思路：至今仍不知道下面这个公式是怎么推出来的...

if(a[index] > 0) pos = (pos - 1 + a[index] - 1) % n + 1;

else pos = ((pos - 1 + a[index]) % n + n) % n + 1;

线段树方面跟上题一样。

http://blog.csdn.net/ahfywff/article/details/7222193

\*/

int Query(int root, int num)

{

f[root].sum--;

if(f[root].l == f[root].r) {

return f[root].l;

}

if(f[L(root)].sum >= num) Query(L(root), num);

else Query(R(root), num - f[L(root)].sum);

}

int main()

{

int K;

while(scanf("%d%d", &n, &K) != EOF) {

for(int i = 1; i <= n; i++) scanf("%s%d", name[i], a+i);

int p = fansushu(n);

build(1, 1, n);

int pos = K; /\* 当前环中第pos个孩子 \*/

int index ; /\* idx为当前出去的孩子的原始位置 \*/

while(p--) {

index = Query(1, pos);

n--; //不先抹消k的位置吗？ ---在Query里面做了

if(n == 0) break;

if(a[index] > 0) pos = (pos - 1 + a[index] - 1) % n + 1;

else pos = ((pos - 1 + a[index]) % n + n) % n + 1;

}

printf("%s %d\n", name[index], ans);

}

return 0;

}

/\*

中国剩余定理CRT 扩展欧几里得求最小正整数解

题意：中国剩余定理模板题，求解一般同余方程组(C≡ai mod mi不一定两两互质)，输入顺序为ai, mi

中国剩余定理：

假设C ≡ a1 (mod m1)，C ≡ a2 (mod m2)

令C = m1\*x1 + a1，则m1\*x1 ≡ a2-a1(mod m2)，d = gcd(m1, m2)

由斐蜀定理知，若d不能整除(a2-a1)，则方程组无解

用扩展欧几里得可以求出m1\*y1 + m2\*y2 = d的任一解y1， 对应到m1\*x1 + m2\*x2 = a2-a1的解x1

就可以表示为x1 = y1\*(a2-a1)/d，C1 = m1\*x1 + a1

接下来我们要求最小正整数解C0。

我们知道，原方程m1\*x1 ≡ a2-a1(mod m2)的任意解C有C ≡ C1 mod m2/d //不会证

C = C1 + (m2/d)\*x，所以，最小正整数解C0 = [C1 % (m2/d) + m2/d] % (m2/d)

接下来是不断合并上面两个方程C ≡ a1 (mod m1)，C ≡ a2 (mod m2)并继续迭代下去，一直到最后

合并：原二方程可合并为C ≡ C0 (mod LCM(m1,m2))

//这里的C0是前面方程的最小公共解，我们还要继续迭代到只剩一个方程，找到所有方程的公共最

小解

下面摘一段“为什么上面两条方程就可以被C’ ≡ C (mod LCM(m1,m2))代替”的证明，目前看不懂~~

"

http://scturtle.is-programmer.com/posts/19363.html

题意有:C=A1+X1\*B1=A2+X2\*B2

因为B是lcm(B1,B2)所以有:

B=Y1\*B1=Y2\*B2

又C`=C+Z\*B,把C带入得:

C`=A1+(X1+Z\*Y1)\*B1=A2+(X2+Z\*Y2)\*B2

所以C`就符合最初的两个式子了吧,不清楚为何是最小的

"

\*/

void CRT(ll &m1, ll &a1, ll m2, ll a2, ll x, ll d) //引用！ x是m1\*x + m2\*y = d的任意解，在外面由扩展欧几里得求出来，d是gcd(m1,m2)

{

ll C, m, t;

t = m2 / d; //

x = (a2-a1)/d \* x; //x1 = x0\*(a1-a2)/d，x0是x\*m1 + y\*m2 = a1-a2的任意解

x = (x % t + t) % t; //这样就能保证x1是最小正整数解 ????????????????

C = x\*m1 + a1; //合并C ≡ a1 (mod m1), C ≡ a2 (mod m2)成 C' ≡ C (mod LCM(m1,m2))

m = LCM(m1, m2, d); //因为x是前两式最小解，所以C = x\*m1 + a1

//??????????? 为啥能合并???

a1 = C, m1 = m;

}

int main()

{

int n;

while(scanf("%d", &n) != EOF) {

ll a1, a2, m1, m2, x, y; bool flag = true;

scanf("%I64d %I64d", &m1, &a1);

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

scanf("%I64d %I64d", &m2, &a2); if(!flag) continue;

ll d = EGCD(m1, m2, x, y);

if((a2 - a1) % d != 0) { flag = false; continue; }

CRT(m1, a1, m2, a2, x, y, d);

}

printf("%I64d\n", flag ? a1 : -1);

}

return 0;

}

/\*

状压DP 背包问题 好题

题意：n(n<=10)件物品，每件物品重ai。两辆容量分别为C1,C2的车。要把这n件物品搬到另外一个地方（两辆车都是一起的）问最少搬几趟？

思路：初看根本没思路，就算看数据范围知道是状压也完全没思路，想记忆化搜，但也不知道怎么搜。

预处理出每一种状态是否能一趟运走。这个判断用0/1背包来搞就可以了。

接下来是核心。还是0/1背包，dp[T]表状态T最少运几趟，则dp[T] = min{ dp[S] + 1 } | T能由S再加一趟达到。

涨见识了。背包问题改为求几趟竟然要这么麻烦的状压...

\*/

bool check(int x)

{

memset(f, false, sizeof(f));

f[0] = true;

int sum = 0;

for(int i = 0; i < n; i++) if((1 << i) & x) sum += a[i];

if(sum > C1 + C2) return false;

for(int i = 0; i < n; i++) if((1 << i) & x) {

for(int j = C1; j >= a[i]; j--)

if(f[j-a[i]]) f[j] = true;

}

for(int j = C1; j >= 0; j--) if(f[j]) return sum-j <= C2;

return sum <= C2;

}

int main()

{

while(cases--) {

scanf("%d%d%d", &n, &C1, &C2);

for(int i = 0; i < n; i++) scanf("%d", &a[i]);

s[0] = true;

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

s[i] = check(i);

memset(dp, 127, sizeof(dp));

dp[0] = 0;

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

if(s[i]) {

dp[i] = 1;

continue;

}

for(int j = 0; j < i; j++) if((i | j) == i) {

int k = i - j;

if(s[k]) dp[i] = min(dp[i], dp[j] + 1);

}

}

printf("Scenario #%d:\n%d\n\n", ++Cas, dp[(1 << n) - 1]);

}

return 0;

}

/\*

双连通 点双连通 边双连通 奇圈 经典题 好题

TLE + WA 了 N次，终于AC了...

题意：

亚瑟王要在圆桌上召开骑士会议，为了不引发骑士之间的冲突，并且能够让会议的议题有令人满意的结果，每次开会前都必须对出席会议的骑士有如下要求：

1、 相互憎恨的两个骑士不能坐在直接相邻的2个位置；

2、 出席会议的骑士数必须是奇数，这是为了让投票表决议题时都能有结果。

如果出现有某些骑士无法出席所有会议（例如这个骑士憎恨所有的其他骑士），则亚瑟王为了世界和平会强制把他剔除出骑士团。

现在给定准备去开会的骑士数n，再给出m对憎恨对（表示某2个骑士之间使互相憎恨的），问有几个骑士永远无法开会。

注意：1、所给出的憎恨关系一定是双向的，不存在单向憎恨关系。

2、由于是圆桌会议，则每个出席的骑士身边必定刚好有2个骑士。即每个骑士的座位两边都必定各有一个骑士。

3、一个骑士无法开会，就是说至少有3个骑士才可能开会。

解题：

1.读清楚题目要求，是“有几个骑士不能和任何人形成任何的圆圈”！！这里先称他们为“悲剧骑士”，其他能开会的人称“欢乐骑士”

2.能相邻坐即意味着两者没有仇恨，那很容易联想到建仇恨关系的补图~G

3.在~G里，悲剧骑士要么不能与别人形成环，要么只能和别人形成偶数环。换个角度，欢乐骑士是不仅能和别人形成环，且这些环里至少有一个(n >= 3 && n % 2 != 0)，就是说欢乐英雄至少存在在一个奇圈里。

4.双连通分量和奇圈有如下关系：

（1） 如果一个双连通分量内的某些顶点在一个奇圈中（即双连通分量含有奇圈），那么这个双连通分量的其他顶点也在某个奇圈中；

（2） 如果一个双连通分量含有奇圈，则他必定不是一个二分图。反过来也成立，这是一个充要条件。

第一个定理在这里特别重要，借助该定理我们可以把“骑士i是否至少存在于一个奇圈里”转换成“骑士i所在的双连通分量是否存在奇圈”

5.Tarjan可以求出双连通分量，每符合dfn[u] <= low[v]就出来一个双连通分量，所以我们要在每出来一个双连通分量的时候都判断该双连通分量是否存在奇圈，若存在，则该双连通分量里的所有骑士都有资格开会。

6.判断奇圈用染色法，网上很多人直接用二分图染色法，我觉得那是错的，但是该题的数据弱，可以过=\_= 得好好理解定理1才能比较好地理解奇圈染色法.

7.该题用点双或边双的思维来做都可以，我是用点双，网上较多的是边双，差别不大，点双要处理一点小细节.

例如从栈里取出当前双连通块的点时不能把割点也取出栈，因为割点可以属于多个双连通块。

刚TLE的时候黄叔说“边双的题你用点来做当然不行啦”，嘻嘻~~

8.到后面感觉，染色是关键。。。很多细节，如要理解定理1才能写好，理解不好就会写出个二分图染色；如要初始化color, yes,

\*/

bool path[MAXN][MAXN], label[MAXN],visit[MAXM \* 2], yes[MAXN], in\_stack[MAXN];

int n, m, edge\_num, Index;

stack<int> S;

//result[]记录当前双连通块的骑士 color[]染色用

//label[i]记录i是否有资格开会 yes[i]表示当前双连通块是否包含骑士i visit[k]由于是无向图，用来判断边是否重用

bool dye(int u, int c, int f) //这里忘记判断父节点了

{

bool flag = true;

if(color[u]) return color[u] != c;

color[u] = c;

for(int k = head[u]; k != -1; k = edge[k].next) {

int v = edge[k].v;

if(!yes[v] || v == f) continue;

if(dye(v, 3-c, u)) return true; //一开始是直接返回true，没有后面的else，函数结尾直接return false，但是当是叶子节点的时候会出错

else flag = false;

}

//color[u] = 0; //如果都不成功就抹掉标记，因为是dfs /// ??? //好像不用耶。。得好好理解定理1

return flag;

}

void judge(int u, int v)

{

int sum = 0, x;

memset(yes, false, sizeof(yes));

do {

x = S.top();

S.pop();

result[sum++] = x;

yes[x] = true;

in\_stack[x] = false;

} while(x != v); //最后的u不要出栈，因为割点可能属于多个双连通块

result[sum++] = u;

yes[u] = true; //这是最后改的点。。。。终于AC了。。。

for(int i = 0; i < sum; i++) color[result[i]] = 0; //抹掉标记 //之前放在后面，当然会WA啦！！

if(sum < 3 || !dye(result[0], 1, -1)) return ;

for(int i = 0; i < sum; i++) label[result[i]] = true; //标记这些人有开会资格

}

}

void Tarjan(int u)

{

dfn[u] = low[u] = Index++;

S.push(u);

in\_stack[u] = true;

for(int k = head[u]; k != -1; k = edge[k].next) if(!visit[k]) {

visit[k] = visit[k ^ 1] = true;

int v = edge[k].v;

if(!dfn[v]) {

Tarjan(v);

low[u] = min(low[u], low[v]);

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

judge(u, v); //遇到割点，判断当前双连通分量有没有奇圈。

}

} else if(in\_stack[v]) low[u] = min(low[u], dfn[v]); //为什么要判断是否in\_stack?

}

}

int work()

{

int x, y;

init();

while(m--) {

scanf("%d%d", &x, &y);

if(path[x][y]) continue;

path[x][y] = path[y][x] = true;

}

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

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

if(path[i][j] == false) { //这里没处理好，TLE了好多次。一定要(i,j),(j,i)连着加到存边数组里面，不然Tarjan里面

//visit[k] = visit[k^1] = true 就错了，之前没注意到

add\_edge(i, j);

add\_edge(j, i);

}

for(int i = 1; i <= n; i++) if(!dfn[i]) Tarjan(i);

int sum = 0;

for(int i = 1; i <= n; i++) sum += (label[i] == false);

return sum;

}

/\*

二分图最大匹配

题意：N\*N网格，其中一些格可以放Asteroids，每个Asteroids在同一行或同一列的话会打架，问最多能放多少Asteroids？

经典二分图最大匹配 匈牙利算法模板

本来觉得是模板题，后来看了hdu 1045(Fire Net)后更想明白了这道题的构图：把每一行看成一个点作为二分图的X部，同理列为Y部，若有Asteroids>格在(r,c)，则说明r\_X与c\_Y有交集，有交集就连一条边。

\*/

/\*

概率DP

题目：有2^n个队，相邻的两两打淘汰赛，求最后哪个队夺冠的概率最大。假设a1,a2,a3,..ak(i<k, ai<ak)为前一轮胜利的参赛者序列，则当前轮由a1,a2对手，a3,a4对手...

思路：dp[i][j]表示第i轮时队j胜利，那么dp[i][j]前提就是i-1轮的时候j胜利，而且在第i轮赢了对手。

那么接下来就是枚举j在第i轮时可能的对手k了，j要在i轮胜，就要在i轮把这些可能的对手k都打败。

通过二进制发现规律，j与k的高位都相等，到了第i-1位刚好相反，就用这个性质判断是不是j在i轮的对手。

dp[i][j] = Σ(dp[i-1][j] \* dp[i-1][k] \* p[j][k]), (k >> (i-1) ^ 1) == (j >> (i-1))

收获：原来这个规则下有这么个规律呀，j,k能在第i轮相遇的条件是(k >> (i-1) ^ 1) == (j >> (i-1)) ! “高位相同，i-1位相反”

我看的题解：http://blog.csdn.net/acm\_cxlove/article/details/7921608 几乎一模一样了=\_= 数学白来搞概率伤不起呀

\*/

for(int j = 0; j < top; j++) f[0][j] = 1;

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

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

f[i][j] = 0;

for(int k = 0; k < top; k++) if(((k >> (i-1)) ^ 1) == (j >> (i-1)) ) { //TODO

f[i][j] += f[i-1][j] \* f[i-1][k] \* p[j][k];

}

}

/\*

0/1背包统计方案 好题

题意：有n件物品，每件都有价值，用m的钱买这些物品，。尽可能多的买东西，（直到钱不够买其他的东西为止）。问有多少种买东西的方法。。

思路：摘抄：http://blog.csdn.net/scorpiocj/article/details/6607580

考虑什么情况下一个都放不进去，就是剩下物品中体积最小的那个（体积为v)都放不进去，即背包装载容量应为[V-v+1,V]

那么我们可以将物品按容量排序，从小到大枚举不放入背包的体积最小的，那么此时比它小的物品都应该放入背包中，我们可以对剩下的i+1>到n的物品做01背包，然后将可行的方案统计出来，这样，第i件物品考虑了i-1次，总复杂度为O(n^2\*V)

n较大的话会超时，我们可以反过来做，每个物品只放入背包中一次，即枚举的时候从最大的开始，先统计此时的方案数，然后将物品放入背>包，这样一直下去，总复杂度O(n\*V)

具体可以参见《浅谈几类背包问题》

看不会优化的部分，所以只写了n^2 \* V的。

\*/

int main()

{

int cases, Cas = 0;

scanf("%d", &cases);

while(cases--) {

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

for(int i = 1; i <= n; i++) scanf("%d", &a[i]);

sort(a+1, a+n+1);

if(a[1] > m) {

printf("%d 0\n", ++Cas);

continue;

}

for(int i = 1; i <= n; i++) sum[i] = sum[i-1] + a[i];

long long ans = 0;

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

memset(dp, 0, sizeof(dp));

dp[0] = 1;

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

for(int j = m - sum[u-1]; j >= a[i]; j--)

dp[j] += dp[j-a[i]];

for(int j = max(0, m - sum[u-1] - a[u] + 1); j <= m - sum[u-1]; j++)

ans += dp[j];

}

printf("%d %I64d\n", ++Cas,ans);

}

return 0;

}

/\*

2-SAT 判定问题

题意：给出一个圆上的若干点对，要求用线把这些点对连接起来，线可以在完全在圆里面或完全在圆外面，问有没有可能使得所有线段不交叉，相交与

点不算交叉

思路：要是告诉是2-SAT就好做了...-\_-||

构图是以线段为元素来构图，若i,j交叉，即矛盾，则

x <-> y'

x'<-> x

就是双向图啦。

跟之前的几道题不同的就是它建的是双向边。

\*/

/\*

数位DP 有前导0处理

题意：[l,r]中的数转为二进制后0的个数不小于1的个数的数。

思路：常规数位DP，就是要注意前导 0

dp状态：dp[pos][one][zero]

注意点：

LL dfs(int pos, int one, int zero, bool full, bool first) {

res += dfs(pos-1, one+i, zero+(i==0 && !first), full && i == end, first && i == 0);

//zero表当前0的个数，那如果现在first为true的话zero即不应该增加！TODO

}

\*/

LL dfs(int pos, int one, int zero, bool full, bool first)

{

if(pos == -1) {

if(first) return 1;

return zero >= one;

}

if(!first && !full && dp[pos][one][zero] != -1) return dp[pos][one][zero];

LL res = 0;

int end = full ? digit[pos] : 1;

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

res += dfs(pos-1, one+i, zero+(i==0 && !first), full && i == end, first && i == 0); //TODO

}

if(!full && !first) dp[pos][one][zero] = res;

return res;

}

/\*

次最短路（边可允许重复走） 好题

题意：无向图，求次最短路，边可重复走

摘：https://www.byvoid.com/blog/pku-3255/

这道题不同于一般的次短路径问题，因为允许边重走。看似更为复杂了，其实是更简单了一些。方

法为先用Heap+Dijkstra求出1和N的单源最短路径，把无向边看成两个有向边，然后枚举每单向条边

(u,v)，计算Dist=dis(1,u) + dis(N,v) + w，看看此时Dist的值是否大于dis(1,N)，如果是的话用

它更新次短路径，保留一个最小的值。

我画了一下图，如果边e(u,v,w)在次最短路，那么情况无非3种：

(1) e不在最短路上，且u,v也不在最短路上，则Dist=dis(1,u) + dis(N,v) + w很显然使用（可以

看成最短与次短就是两条有岔口的路径

(2) e不在最短路上，u,v有一点在最短路上，则Dist=dis(1,u) + dis(N,v) + w就等于最短路长度

shortest + 2\*w

(3) e在最短路上，则Dist=dis(1,u) + dis(N,v) + w大于shortest的话，其实就是s->u->v->u->v->t

由于AC后发现自己的dijstra不是堆优化的=\_=弄混了，应该是pair<dis, node>的，写成pair<node,pair>

了=\_=，所以干脆删掉了

\*/

int solve()

{

dijstra(1, n, sp[0]), dijstra(n, 1, sp[1]);

int short\_path = sp[0][n], ans = INF;

for(int i = 0; i < edge\_num; i++) {

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

int tmp = sp[0][u] + sp[1][v] + w;

if(tmp > short\_path && ans > tmp) ans = tmp;

}

return ans;

}

/\*

RMQ 线段树

题意：给出非递减序列{A}，若干询问区间[l,r]里出现最多的数。

思路：要是线段树的话那就是水题。这里用RMQ的Sparse Table写。思路还是看别人的。感觉还是挺不错的思路 。

把原数列转化一下。。。

if (num2[i] == num2[i-1]) num[i] = num[i-1]+1;

else num[i] = 1;

然后对查询区间[l,r]分两部分，前面被切割部分和后面完整部分，前面直接计数，后面完整部分RMQ求最大值

\*/

int Query(int l, int r)

{

if(l > r) return 0;

int k = (int)(log((r - l + 1) \* 1.0)/log(2.0));

return max(st[l][k], st[r-(1<<k)+1][k]);

}

int main()

{

int query, l , r;

while(scanf("%d%d", &n, &query) , n) {

init();

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

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

if(a[i] == a[i-1]) num[i] = num[i-1] + 1;

else num[i] = 1;

}

for(int i = 1; i <= n; i++) st[i][0] = num[i];

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

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

st[i][j] = max(st[i][j-1], st[i + (1 << (j-1))][j-1]);

while(query--) {

scanf("%d%d", &l, &r);

int tmp = l;

while(a[l] == a[l-1] && l <= r) l++;

printf("%d\n", max(Query(l, r), l - tmp));

}

}

return 0;

}

/\*

树形DP LCA 统计 好题 神题

题意：在一个无向树上给出若干条sample paths，最后输出每条边被多少条sample paths覆盖。

思路：暴力法是LCA直接暴搞，TLE。

然后想到像一维线段树那样标记法add[]搞，于是对于每条sample path(u,v)，标记add[u]++, add[v]++, add[ ancester(u,v) ] -= 2

但是后面就不知道怎么像一维线段树那样统计了。想了很久都没办法。然后看题解。

哇靠，太巧妙了！dfs回溯统计add，太妙了！！！赞！

//最后对整个图搜一遍，回溯的时候每个边加一下儿子几点add[]的值就可以了。

int dfs(int u, int fa)

{

for(int k = head[u]; k != -1; k = edge[k].next) {

int v = edge[k].v;

if(v == fa) continue;

times[ k/2+1 ] = dfs(v, u); //k/2+1是对应的边

add[u] += add[v];

}

return add[u];

}

\*/

/\*

最小费用最大流 好题

题意：给定一个n\*n的方格，每个方格里有一个数，从左上角出发，只能向下走或向右走，取过的数不计算权值，问走k次能得到的最大权值和

思路： 睡觉去了。。。好题，要补上题解

\*/

摘抄：

题意：有个方阵，每个格子里都有一个非负数，从左上角走到右下角，每次走一步，只能往右或往下走，经过的数字拿走

每次都找可以拿到数字和最大的路径走，走k次，求最大和

这是最大费用最大流问题,每次spfa都找的是一条和最大的路径s--到左上角的边流量是K限制增广次数

求最大费用最大流只需要把费用换成相反数，用最小费用最大流求解即可

构图过程：如上图

每个点拆分成两个aa'   之间建两条边(当然还要建退边)，分别是   (费用为该点相反数,流量为1) (费用为0,流量为k-1)

路过这点时，可以通过前边那条边拿到数字，

以后再从这儿过，就没有数字可拿了，走的就是第二条边

然后是 没点向 右和下 建一条边  费用0，流量k

然后s、t连接左上角和右下角即可

求最小费用最大流的方法和EK求最大流的方法类似，

都是找一条增广路径，然后把流加进去

只不过EK找的是路径最短的增广路(广搜即可得)

而最小费用最大流 找的是费用最小的路径(对费用求最短路)

void build()

{

int num;

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

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

scanf("%d", &num);

int u = i\*n + j, v = n\*n + u;

add\_edge(u, v, num, 1);

add\_edge(u, v, 0, K-1);

if(j < n-1) {

add\_edge(v, u+1, 0, K);

}

if(i < n-1) {

add\_edge(v, u+n, 0, K);

}

}

src = n\*n\*2 , sink = src + 1;

add\_edge(src, 0, 0, K);

add\_edge(src-1, sink, 0, K);

}

/\*

最优比率环

摘抄：

思路：之前做过最小比率生成树，也是属于0/1整数划分问题，这次碰到这道最优比率环，很是熟悉，可惜精度没控制好，要不就是wa,要不就是tle，郁闷>啊！实在是懒得码字，直接copy吧：

题目的意思是：求一个环的{点权和}除以{边权和}，使得那个环在所有环中{点权和}除以{边权和}最大。

令在一个环里，点权为v[i]，对应的边权为e[i]，

即要求：∑（i=1，n）v[i]/∑（i=1，n)e[i]最大的环（n为环的点数），

设题目答案为ans，

即对于所有的环都有 ∑（i=1，n）（v[i]）/∑（i=1，n)（e[i]）<=ans

变形得ans\* ∑（i=1，n）（e[i]）>=∑（i=1，n）（v[i]）

再得 ∑（i=1，n）（ans\*e[i]-v[i]） >=0

稍分析一下，就有：

当k<ans时，就存在至少一个环∑（i=1，n）（k\*e[i]-v[i]）<0，即有负权回路(边权为k\*e[i]-v[i])；

当k>=ans时，就对于所有的环∑（i=1，n）（k\*e[i]-v[i]）>=0，即没有负权回路。

然后我们就可以使新的边权为k\*e[i]-v[i]，用spfa来判断付权回路,二分ans。

\*/

/\*

二分图带权匹配 拆点 好题

题意：m家工厂，n个玩具订单，An\*m矩阵描述j号工厂生产i号订单的话所需时间，同一时间内一工厂只能生产一个订单的玩具，问加工完所有玩具所话

费的时间的最小平均值（话费之间包括等待时间---如果有两个订单在同一家工厂里生产的话，第二个等第一个订单做完所花费的时间）

思路：做专题的时候看到拆点，于是自然想到把m个工厂拆成m\*n个点，但是因为题目要算等待时间，所以就不知道怎么建图。

看题解知道，若有t1,t2,t3号玩具订单要依次在f号工厂做，则所花费的时间 = t1 + (t1+t2) + (t1+t2+t3)，而巧妙就巧妙在这里，\*\*\*这个式

子可以再变换成 3\*t1 + 2\*t2 + t3 \*\*\*

到这里换个思维建图，把订单执行的顺序反过来思考，这里用exec\_time[f]来表示f上总花费时间：若t3订单最后一个执行，则exec\_time[fac]>里面只包括一个t3。若t2是倒数第二个执行，则exec\_time[f]包括2\*t2 ... 扩展开，订单t在工厂f作为倒数第k个执行，则花费时间k\*t[i][f]

这是一个很重要也很妙的一点，抓住题目只要求“平均值”，也就是总和，对于工厂f，t1 + (t1+t2) + (t1+t2+t3)和3\*t1 + 2\*t2 + t3是相等的

，就是要求我们打破常规思维，\*\*\*为什么我们一定要把某订单的执行时间看成一个整体呢？\*\*\* 题目只要求总时间和，常规思维的话要考虑“该订单前面有

谁”，而现在把时间总和的计算公式变成了第二种形式后，我们就只要考虑“改订单是第几个执行”，有种DP里通过变换获得了无后效性。

太棒了，简单的变换反映了思维角度的变换，使得我们要考虑的问题变得简单了，只要考虑“我在第几”而非“前面是谁”。

建图：还是把工厂拆成n\*m个点, X部某点i, Y部某个点j，边权i-->j没有实际意义，所以我无法描述他们的意义，只能还是要上面的式子来说明

，edge[i-->j] = k \* t[i][f], 其中k表i为倒数第k个在f工厂执行。

由于拆点后对于同一订单，k\*t[i][f] > (k-1)\*t[i][f] > ... > 1\*t[i][f]，所以能保证最优匹配一定是最小总和。

不由得再赞叹一下，有时为了实现我们的目标，没必要硬要每个模型都有实际意义，可能只要它有数学意义就足够了，而这个数学意义平常比较难想，

得画到纸上才Aha, insight.

二分图多重匹配类型题把匈牙利算法扩展到多对一

这道题把KM算法扩展到能加上`顺序`元素

\*/

int main()

{

int cases;

scanf("%d", &cases);

while(cases--) {

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

init();

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

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

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

}

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

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

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

//add\_edge(i, j\*n+k, -a[i][j]\*(1+k));

add\_edge(i, k\*m+j, -a[i][j]\*(k+1)); //拆点

M = n\*m; //Y部点

KM();

long long ans = 0;

for(int i = 0; i < M; i++) if(link[i] != -1) {

ans += a[link[i]][i%m] \* (i/m+1);

//ans += a[link[i]][i/n] \* (i%n+1);

}

printf("%lld\n", ans); //ps: 输出double/float最好都用%f而不是%lf, %f才是标准，%lf只是编译器提供的便利

}

return 0;

}

/\*

双连通 缩点 桥 LCA

题意：给一个无向连通图，问每次新加进一条边后，图中桥的数目。 重边算一条。

思路：求双连通分量，利用并查集缩点，形成一棵树，树边肯定都是桥，然后每对点x,y,找原图中x,y点对应的新图中的点，如果不是一个点，则向上>找它们的LCA,因为它们之间连了一条边，所以这些点到它们的LCA之间的边都不是割边了，找LCA时，先将两点上升到同一层次，然后一起再向上找父亲节点

，其间遇到桥就把桥的标记删除，并且答案减1

思路还是挺简单的，不过我做了很久，把调试过程记录下来吧。准确来说的对拍调试过程。

1. ans是当前桥的数目。两种方法，一种是在Tarjan里面if (dfn[u] <= low[v]) ans++；第二种是直接ans = Order - 1; Order表示连通块的个数，>因为缩点之后是一棵树，那树的没一条边都是桥，所以桥数 = 点数 - 1.

2. 题目没有说当有重边的时候要怎么办，实验表明，当有重边的时候只算一条。那Tarjan的时候有两个办法防止无尽循环，一是在读边的时候就注意>有没有重边，则在Tarjan里面可以用visit[]来记录某边是否已经用过；而是改边判为点判，即记录点的父节点，然后递归之前判断一下。

感觉还是第二种比较省事。这里记一下代码:

for(int k = head[u]; k != -1; k = edge[k].next) {

int v = edge[k].v;

if(fa[u] == v) continue; //!!

if(!dfn[v]) {

fa[v] = u;...

....

3. 建缩点之后的树还是挺简单的，可以当作模板了。

4. 朴素lca的各个语句的位置弄混了...

5. 下面这句调试语句棒=帮了很多忙

for(int i = 1; i <= n; i++) printf("# %d\t%d\t%d\n", i, belong[i], deep[belong[i]]); //查了这里才知道

\*/

/\*

LCA 并查集 Tarjan离线 好题 难题 神题！！！

题意：

收获：并查集在压缩路径的时候可以更新一些数据，这点是我之前没见过的。 （详见Find()）

Tarjan离线搞最小公共祖先以递归回到祖先lca的时候才能有信息回答query(u,v)，之前都是遍历到v点的时候就能回答query(u,v)（假设u点已经遍历过了），这道题是要递归回到祖先z的时候才能回答。

思路：对于每个节点x，当前祖先为z

max(x)/min(x) x到z路径上的最大/小值

up(x) x->z路径上的dp值

down(x) z->x路径上的dp值

则ans( query(u,v) ) = max{ up(x), down(y), max(y)-min(x) }

方程不是重点，重点是如何实现.

代码解释：

//================================================================================

int Find(int x)

//把更新过程放在Find里面，边向上并查集找祖先，边更新x-lca的信息，把并查集用得太好了！！

//先把当前x的父亲y记录下来，然后father[x] = Find(Father[x]) 向上找y的祖先并更新节点y的属性。

//之后利用y点的信息更新x点信息。想象一下，超美妙。假设当前的祖先为z，则Find(father[x])过程中更新了区间(y, z)的信息，存在节点y里面，那接下来就用区间(y, z)的信息（存在节点y），和(x, y)的信息（存在节点x）来更新区间(x, z)的信息，并存到x节点。

{

if(x == father[x]) return x;

int y = father[x];

father[x] = Find(father[x]);

a[x].up = max(a[y].max - a[x].min, max(a[x].up, a[y].up));

a[x].down = max(a[x].max - a[y].min, max(a[x].down, a[y].down));

a[x].max = max(a[x].max, a[y].max);

a[x].min = min(a[x].min, a[y].min);

return father[x];

}

//==================================================================================

for(int Size = Quetion[u].size(), i = 0; i < Size; i++)

//以前的LCA都是对于query(u,v)，若vis[v] == true，则根据Find(v)就可以对query进行回答，但这道题不行。因为我们接下来要递归下去，要保持当前子树的祖先都是x，所以我们必须得在递归回来之后才能Union(x, fa)。那不Union，根据Find(v)找到当前v的祖先（当前树的根节点）又没有办法回答query(u,v)，那就不能像一般LCA那样在这里就回答咯~~

//怎么办呢？首先我们应该考虑，什么时候我们才能回答query(u,v)？设当前lca(u,v) = z，分析一下知道，要回答query(u,v)，就要知道(u->z)和(z->v)的信息。那自然我们就知道---当递归回到z的时候我们能回答query(u,v).

//所以这里我们把每个query(u,v)存到Q[z]里面，在LCA递归回到z的时候再输出。

{

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

if(!vis[v]) continue;

int fv = Find(v);

if(id >= 0) Q[fv].push\_back(u), Q[fv].push\_back(v), Q[fv].push\_back(id);

else Q[fv].push\_back(v), Q[fv].push\_back(u), Q[fv].push\_back(-id);

}

for(int k = head[u]; k != -1; k = edge[k].next) {

int v = edge[k].v;

if(!vis[v]) {

LCA(v);

Union(u, v); //Union的位置很重要！递归回来才能Union

}

}

for(int Size = Q[u].size(), i = 0; i < Size; i += 3) {

//这里就是对于每个z的回答. 这里u表示lca(x,y) = u

int x = Q[u][i], y = Q[u][i+1], id = Q[u][i+2];

Find(x), Find(y);

ans[ id ] = max( a[x].up, max(a[y].down, a[y].max - a[x].min) );

}

\*/

//poj 3728

#include <algorithm>

#include <vector>

#include <string.h>

#include <stdio.h>

using namespace std;

#define MAXN 50005

struct Tree\_Node {

int min, max, up, down;

} a[MAXN];

struct Edge {

int next, v;

} edge[MAXN \* 2];

typedef pair<int,int> PII;

vector< PII > Quetion[MAXN];

vector<int> Q[MAXN];

int head[MAXN], father[MAXN], ans[MAXN];

bool vis[MAXN];

int n, edge\_num;

void init()

{

edge\_num = 0;

memset(head, -1, sizeof(head));

memset(vis, false, sizeof(vis));

for(int i = 1; i <= n; i++) Quetion[i].clear(), Q[i].clear();

for(int i = 1; i <= n; i++) father[i] = i;

}

void add\_edge(int u, int v)

{

edge[edge\_num].v = v;

edge[edge\_num].next = head[u];

head[u] = edge\_num++;

}

int Find(int x)

{

if(x == father[x]) return x;

int y = father[x];

father[x] = Find(father[x]);

a[x].up = max(a[y].max - a[x].min, max(a[x].up, a[y].up));

a[x].down = max(a[x].max - a[y].min, max(a[x].down, a[y].down));

a[x].max = max(a[x].max, a[y].max);

a[x].min = min(a[x].min, a[y].min);

return father[x];

}

void Union(int u, int v)

{

father[v] = u;

}

void LCA(int u)

{

vis[u] = true;

for(int Size = Quetion[u].size(), i = 0; i < Size; i++) {

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

if(!vis[v]) continue;

int fv = Find(v);

if(id >= 0) Q[fv].push\_back(u), Q[fv].push\_back(v), Q[fv].push\_back(id);

else Q[fv].push\_back(v), Q[fv].push\_back(u), Q[fv].push\_back(-id);

}

for(int k = head[u]; k != -1; k = edge[k].next) {

int v = edge[k].v;

if(!vis[v]) {

LCA(v);

Union(u, v); //Union的位置很重要！

}

}

for(int Size = Q[u].size(), i = 0; i < Size; i += 3) {

int x = Q[u][i], y = Q[u][i+1], id = Q[u][i+2];

Find(x), Find(y);

ans[ id ] = max( a[x].up, max(a[y].down, a[y].max - a[x].min) );

}

}

int main()

{

int u, v, query;

while(scanf("%d", &n) != EOF) {

init();

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

scanf("%d", &a[i].min);

a[i].max = a[i].min;

a[i].up = a[i].down = 0;

}

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

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

add\_edge(u, v), add\_edge(v, u);

}

scanf("%d", &query);

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

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

Quetion[u].push\_back( PII(v, i) ), Quetion[v].push\_back( PII(u, -i) );

}

LCA(1);

for(int i = 1; i <= query; i++) printf("%d\n", ans[i]);

}

return 0;

}