目录

[DP 5](#_Toc31240)

[例题 5](#_Toc12685)

[1.n个数k个乘号 5](#_Toc2446)

[2.三维DP 6](#_Toc30601)

[3.有限制的背包 6](#_Toc13967)

[3.凑数问题 6](#_Toc24012)

[背包 7](#_Toc32155)

[树形DP 7](#_Toc22089)

[1.入门题:没有上司的舞会 7](#_Toc31193)

[2.树形背包(洛谷p2014) 8](#_Toc17549)

[3.树形背包(边权)(洛谷p2015) 9](#_Toc18872)

[4.牛客370F 10](#_Toc14863)

[数位DP 10](#_Toc6771)

[1.HDU 2089 不含62和4 10](#_Toc4668)

[2.统计0~9在1~n各出现了多少次 11](#_Toc6263)

[概率DP 12](#_Toc21426)

[最长公共子序列(元素不同)O(nlogn) 13](#_Toc13351)

[最长公共子序列+输出序列O(n^2) 13](#_Toc10377)

[最长公共子串+输出子串O(n^2) 15](#_Toc26004)

[最大连续子列和O(n) 15](#_Toc3476)

[最长递增子序列O(n^2) 16](#_Toc12065)

[最长递增子序列O(nlogn) 16](#_Toc6403)

[状压DP 17](#_Toc24114)

[1.棋盘铺1\*2和2\*1的木块。 17](#_Toc31443)

[2.玉米田(洛谷p1879) 18](#_Toc16976)

[四维DP 19](#_Toc11752)

[洛谷P1004/P1006 19](#_Toc5735)

[插头DP 20](#_Toc18116)

[1.(洛谷p5056)求棋盘中的哈密顿回路个数，障碍不能走 20](#_Toc29967)

[2. (洛谷p3272)用L型的地板铺满整个客厅有多少种不同的方案？ 21](#_Toc30011)

[3.(洛谷p3190)求棋盘中权值最大的回路。 23](#_Toc14331)

[数学 25](#_Toc14493)

[一、基本定理 25](#_Toc25605)

[1、鸽巢原理 25](#_Toc536)

[2、计算n^2的因子数 25](#_Toc3693)

[3、 矩阵快速幂 26](#_Toc8049)

[4、十进制快速幂 27](#_Toc19785)

[5、欧拉函数 27](#_Toc22597)

[6、母函数 28](#_Toc5175)

[7、阶乘尾0 29](#_Toc14946)

[8、扩展埃式筛法 29](#_Toc10186)

[9、计算两个数的所有公约数 29](#_Toc13833)

[10、线性基 30](#_Toc25939)

[求一个数能不能由一群数异或得到 30](#_Toc28946)

[求由一群数异或得到的所有数第k小的数 32](#_Toc18927)

[一群数任选使异或值最大 33](#_Toc12353)

[一群值任选使异或值最小 34](#_Toc15102)

[二维问题 34](#_Toc28762)

[在图中找一条1到n的路径，使边权异或和最大。 35](#_Toc25466)

[求1到n异或最大和 36](#_Toc21680)

[11、 基础数论模板（逆元、lucas、快速乘、中国剩余定理） 37](#_Toc21692)

[12、 递推公式模板 38](#_Toc32575)

[13、 博弈论 40](#_Toc20146)

[基础博弈 40](#_Toc18071)

[1. 巴士博弈 40](#_Toc29340)

[2. 威佐夫博弈 40](#_Toc8225)

[3. 斐波那契博弈 40](#_Toc31937)

[4. nim博弈 40](#_Toc2437)

[高阶博弈 40](#_Toc17597)

[1. 抛硬币问题 40](#_Toc347)

[2. K倍动态减法游戏 41](#_Toc28273)

[3. 阶梯博弈 42](#_Toc29713)

[4. \*NIM积 43](#_Toc1847)

[5. 树上删边游戏 44](#_Toc416)

[14、常用结论和方法 44](#_Toc22278)

[二、 刷题总结 45](#_Toc10337)

[1、 期望与概率 45](#_Toc8537)

[走迷宫 45](#_Toc2628)

[摇色子，走出通道 46](#_Toc17214)

[Race to 1 Again 47](#_Toc12178)

[2、欧拉函数 48](#_Toc3455)

[3、快速幂 48](#_Toc4333)

[4、 唯一分解定理 49](#_Toc23858)

[5、 欧几里德和同余方程 51](#_Toc10790)

[高精度 51](#_Toc19886)

[一、 经典题型 51](#_Toc9436)

[1、 java版线性筛 51](#_Toc30513)

[2、 大数任意进制转换 52](#_Toc24838)

[3、 高精度开根号 53](#_Toc1985)

[4、 除法保留小数 54](#_Toc4816)

[5、高精度小数加法(非科学计数法+去尾0+java) 54](#_Toc12303)

[数据结构 55](#_Toc10366)

[一、 线段树 55](#_Toc27742)

[1、 线段树模板递归版 55](#_Toc12735)

[2、 线段树求最小逆序数 57](#_Toc29833)

[3、 模板改编题 59](#_Toc2600)

[4、 线段树——离散化+区间覆盖 61](#_Toc16711)

[5、 线段树——区间合并(维护连续零) 63](#_Toc15908)

[6、线段树——区间染色+区间统计 64](#_Toc591)

[7、线段树——维护DFS序 66](#_Toc24886)

[8、 主席树之静态区间第k大 71](#_Toc28476)

[9、 划分树 72](#_Toc32293)

[二、 LCA 73](#_Toc15084)

[1、LCA-ST表O(nlogn) 73](#_Toc19141)

[2、LCA-tarjan离线O(n+q) 74](#_Toc162)

[3、LCA-倍增O((n+q)logn) 75](#_Toc18176)

[5、 LCA应用 76](#_Toc13171)

[三、 单调队列 77](#_Toc2345)

[四、 单调栈 78](#_Toc27078)

[五、 树 80](#_Toc18912)

[1、树的直径 80](#_Toc28210)

[2、二分+LIS+DFS序 81](#_Toc5130)

[3、字典树应用 82](#_Toc2056)

[查找字符串是否出现 82](#_Toc25009)

[查找前缀出现次数 83](#_Toc16696)

[4、 求树的直径 87](#_Toc29900)

[六、 图论 88](#_Toc23684)

[1、图论知识点 88](#_Toc29491)

[2、并查集 88](#_Toc1557)

[并查集递归版 88](#_Toc18255)

[Poj2236 89](#_Toc23025)

[3、二分图 90](#_Toc4941)

[二分图染色（判断二分图） 90](#_Toc30433)

[4、连通图 92](#_Toc13017)

[Tarjan 92](#_Toc18888)

[5、 最短路 94](#_Toc26533)

[dijkstra变形 94](#_Toc23680)

[Dijkstra链式前向星+堆优化(效率最高) 96](#_Toc28496)

[Dijkstra邻接表+堆优化 97](#_Toc3241)

[第K短路 99](#_Toc30061)

[最短路计数+维护点权和最大O(nlogn) 101](#_Toc29219)

[6、 最小生成树 102](#_Toc10925)

[kruskal+邻接表O(ElogE)//HDU 1233 102](#_Toc14844)

[次小生成树 104](#_Toc21120)

[生成树计数 106](#_Toc8722)

[最小生成树计数（矩阵树） 108](#_Toc25261)

[最小生成树邻接矩阵prim算法 110](#_Toc25300)

[7、 拓扑排序 111](#_Toc27821)

[DAG最长路 111](#_Toc32682)

[七、 字符串 113](#_Toc31768)

[1、 基本知识点 113](#_Toc23378)

[2、 KMP 113](#_Toc8865)

[判断pattern是否是text的子串，并返回匹配的第一个位置 113](#_Toc30313)

[kmp中的next数组求最小循环节的应用 114](#_Toc13284)

[KMP求周期性前缀 输出：i 前缀 115](#_Toc28461)

[KMP之剪花布条 116](#_Toc18322)

[Manacher之最长回文第一种写法 117](#_Toc12117)

[Manacher之最长回文第二种写法 118](#_Toc13869)

[Manacher改编---》求字符串环的最大回文长度 119](#_Toc29808)

[找子串问题 119](#_Toc22965)

[3、 AC自动机 121](#_Toc9380)

[1.求目标串出现了多少个模式串 121](#_Toc23315)

[2.有N个由小写字母组成的模式串以及一个文本串T。每个模式串可能会在文本串中出现多次。你需要找出哪些模式串在文本串T中出现的次数最多。 122](#_Toc18925)

[八、 RMQ 124](#_Toc21656)

[1、 RMQ板子 124](#_Toc17885)

[2、 一维RMQ 125](#_Toc14702)

[3、 二维RMQ 126](#_Toc22281)

[九、 技巧题 128](#_Toc27137)

[1、 二分 128](#_Toc12209)

[（1） 、二分模板 128](#_Toc23815)

[（2） 、二分应用 128](#_Toc11075)

[2、 异或前缀和 129](#_Toc2433)

[3、 贪心题 130](#_Toc28897)

[4、 任意进制数判断回文 131](#_Toc23977)

[5、 Dfs 132](#_Toc29453)

[数拆成幂次和//洛谷p1010 132](#_Toc32251)

[求n个字符串环的最长公共子串 133](#_Toc26294)

[小字辈 134](#_Toc20750)

[背包 136](#_Toc28194)

[6、 尺取法 137](#_Toc25986)

[绝对半径 137](#_Toc10941)

[北京网赛--环游城市 138](#_Toc21351)

[7、 bfs+dp 139](#_Toc16646)

[8、 算法优化 141](#_Toc3913)

[1、 后缀数组比大小 141](#_Toc13647)

[2、 Dijkstra+堆优化 142](#_Toc20026)

[9、 树的遍历 144](#_Toc27183)

[1、给定前序和中序，求层序 144](#_Toc2848)

[10、 Bfs 145](#_Toc3855)

[求矩阵中块的个数 145](#_Toc16237)

[走迷宫的最小步数 146](#_Toc26752)

# DP

## 例题

1.n个数k个乘号  
/\*洛谷p1018

题意：长度为n的数字串插入k个乘号使乘积最大

思路：设dp[i][j]表示前i个数插入j个乘号的乘积最大值。

预处理出用t枚举0到i-1后面插乘号，dp[i][j]=max(dp[t][j-1]\*a[t+1][i],dp[i][j])\*/

**import** java.math.\*;

**import** java.util.\*;

**public** **class** Main {

**static** BigInteger *a*[][] = **new** BigInteger[45][45];

**static** BigInteger *dp*[][] = **new** BigInteger[45][45];

**public** **static** BigInteger max(BigInteger a, BigInteger b) {

**return** a.compareTo(b) < 0 ? b : a;

}

**public** **static** **void** main(String[] args) {

**int** n, k;

Scanner cin = **new** Scanner(System.*in*);

n = cin.nextInt();

k = cin.nextInt();

cin.nextLine();// 注意！！！

String s = cin.nextLine();

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

Arrays.*fill*(*dp*[i], BigInteger.*ZERO*);

Arrays.*fill*(*a*[i], BigInteger.*ZERO*);

}

**int** l = s.length();

**for** (**int** i = 0; i < l; i++) {

**for** (**int** j = i; j < l; j++) {

BigInteger sum = BigInteger.*ZERO*;

**for** (**int** m = i; m <= j; m++) {

sum = sum.multiply(BigInteger.*valueOf*(10)).add(

BigInteger.*valueOf*(s.charAt(m) - '0'));

}

*a*[i][j] = sum;

}

}

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

*dp*[i][0] = *a*[0][i];

}

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

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

**if** (j - 1 >= i)

**break**;

**for** (**int** t = 0; t < i; t++) {

*dp*[i][j] = *max*(*dp*[t][j - 1].multiply(*a*[t + 1][i]), *dp*[i][j]);

}

}

}

System.*out*.println(*dp*[n - 1][k]);

}

}

### 2.三维DP

题意：给你一个n\*m的矩阵和1-n\*m个数，问有多少种情况满足纳什均衡的点只有一个。纳什均衡点是指这个元素在所在行和所在列都是最大的。

思路：设dp[i][j][k]表示用i个数覆盖j行k列的方案数。dp[1][1][1]=n\*m，多出一行：dp[i+1][j+1][k]=(n-j)\*k\*dp[i][j][k],多出一列：dp[i+1][j][k+1]=(m-k)\*j\*dp[i][j][k],行列都不多出：dp[i][j][k]=(j\*k-i)\*dp[i][j][k]。

### 3.有限制的背包

题意:一共有a3 a2 a1…g3 g2 g1段位且递增，每个人只能邀请段位差距不超过5的人，5个人满了就可以开一局，问最多同时能开几局。

const int maxn=200005;

int get(string s)

{

int t=3-(s[1]-'0')+1;

char c=s[0];

t+=(s[0]-'a')\*3;

return t;

}

int a[maxn],dp[maxn];

int main()

{

std::ios::sync\_with\_stdio(false);

int n,t;

cin>>t;

while(t--)

{

cin>>n;

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

memset(a,0,sizeof(a));

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

{

string s;

cin>>s;

a[i]=get(s);

}

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

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

{

dp[i]=dp[i-1];

int flag=0;

for(int j=1;j<=4;j++)

{

if(a[i-j+1]-a[i-j]>5){flag=1;break;}

}

if(!flag) dp[i]=max(dp[i],dp[i-5]+1);

}

cout<<dp[n]<<endl;

}

return 0;

}

### 3.凑数问题

链接：https://ac.nowcoder.com/acm/contest/1/D

小M想知道某件物品的重量，但是摆在他面前的只有一个天平（没有游标）和一堆石子，石子可以放左边也可以放右边。他现在知道每个石子的重量。问能不能根据上述条件，能不能测出所问的重量。

思路：此题数据小（100）,设dp[i][j]表示前i个石子能否凑出质量j，初始化dp[i][0]=1，if(dp[i-1][j]){dp[i][j]=1;dp[i][j+a[i]]=1;dp[i][abs(a[i]-j)]=1;}

## 背包

//v[]是价值,w[]是重量,m是容量

//01背包+滚动数组O(n^2)

#define inf 0x3f3f3f3f

#define ll long long

const int maxn=1e3+10;

int w[maxn],v[maxn],dp[maxn];

int main()

{

std::ios::sync\_with\_stdio(false);

int m,n;

while(cin>>m>>n)

{

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

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

{

cin>>w[i];

}

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

{

cin>>v[i];

}

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

{

for(int j=m; j>=w[i]; j--)

{

dp[j]=max(dp[j],dp[j-w[i]]+v[i]);

}

}

cout<<dp[m]<<endl;

}

return 0;

}

//完全背包+滚动数组O(n^2)

/\*for(int i=1; i<=n; i++)

{

for(int j=w[i]; j<=m; j++)

{

dp[j]=max(dp[j],dp[j-w[i]]+v[i]);

}

}\*/

## 树形DP

### 1.入门题:没有上司的舞会

某大学有N个职员，编号为1~N。他们之间有从属关系，也就是说他们的关系就像一棵以校长为根的树，父结点就是子结点的直接上司。现在有个周年庆宴会，宴会每邀请来一个职员都会增加一定的快乐指数Ri，但是呢，如果某个职员的上司来参加舞会了，那么这个职员就无论如何也不肯来参加舞会了。所以，请你编程计算，邀请哪些职员可以使快乐指数最大，求最大的快乐指数。

思路：dp[x][0]、dp[x][1]表示以x为根的子树且选和不选x的最大值。dp[x][0]=max(dp[v][1],dp[v][0](v为x的所有孩子)，dp[x][1]+=∑dp[v][0]。

const int maxn=1e4;

int r[maxn],in[maxn],dp[maxn][2];

vector<int> g[maxn];

void dfs(int now)

{

dp[now][0]=0;

dp[now][1]=r[now];

for(int i=0;i<g[now].size();i++)

{

int v=g[now][i];

dfs(v);

dp[now][0]+=max(dp[v][1],dp[v][0]);

dp[now][1]+=dp[v][0];

}

}

int main()

{

std::ios::sync\_with\_stdio(false);

int n;

cin>>n;

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

cin>>r[i];

int u,v;

while(cin>>u>>v&&u+v)

{

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

in[u]++;

}

int root;

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

{

if(!in[i])

{

root=i;

break;

}

}

dfs(root);

cout<<max(dp[root][0],dp[root][1])<<endl;

return 0;

}

### 2.树形背包(洛谷p2014)

现在有N门功课，每门课有个学分，每门课有一门或没有直接先修课（若课程a是课程b的先修课即只有学完了课程a，才能学习课程b）。一个学生要从这些课程里选择M门课程学习，问他能获得的最大学分是多少？

#define inf 0x3f3f3f3f

#define ll long long

const int maxn=1e3;

struct node

{

int to,val;

};

vector<node> g[maxn];

int n,m,s[maxn],dp[maxn][maxn],num[maxn];

void dfs(int u,int fa)

{

num[u]=1;

dp[u][1]=s[u];

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

{

int v=g[u][i].to;

if(v==fa) continue;

dfs(v,u);

for(int j=num[u];j;j--)

{

for(int k=1;k<=num[v];k++)

{

dp[u][j+k]=max(dp[u][j+k],dp[u][j]+dp[v][k]);

}

}

num[u]+=num[v];

}

}

int main()

{

std::ios::sync\_with\_stdio(false);

while(cin>>n>>m)

{

memset(num,0,sizeof(num));

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

memset(s,0,sizeof(s));

m++;

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

{

int k;

cin>>k>>s[i];

g[k].push\_back({i,0});

}

dfs(0,-1);

cout<<dp[0][m]<<endl;

}

return 0;

}

### 3.树形背包(边权)(洛谷p2015)

题意：有一棵苹果树，如果树枝有分叉，一定是分2叉（就是说没有只有1个儿子的结点）。这棵树共有N个结点（叶子点或者树枝分叉点），编号为1-N,树根编号一定是1。现在这颗树枝条太多了，需要剪枝。但是一些树枝上长有苹果。给定需要保留的树枝数量，求出最多能留住多少苹果。

思路：将要保留的边的数量++，就转化成删除等量的点了。方法和上题类似，注意边权转化成点权是给to这个点。

#define inf 0x3f3f3f3f

#define ll long long

const int maxn=1e3;

struct node

{

int to,val;

};

vector<node> g[maxn];

int n,m,s[maxn],dp[maxn][maxn],num[maxn];

void dfs(int u,int fa)

{

num[u]=1;

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

{

int v=g[u][i].to;

if(v==fa) continue;

dp[v][1]=g[u][i].val;

dfs(v,u);

for(int j=num[u];j;j--)

{

for(int k=1;k<=num[v];k++)

{

dp[u][j+k]=max(dp[u][j+k],dp[u][j]+dp[v][k]);

}

}

num[u]+=num[v];

}

}

int main()

{

std::ios::sync\_with\_stdio(false);

while(cin>>n>>m)

{

memset(num,0,sizeof(num));

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

m++;

for(int i=0;i<n-1;i++)

{

int u,v,w;

cin>>u>>v>>w;

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

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

}

dfs(1,0);

cout<<dp[1][m]<<endl;

}

return 0;

}

### 4.牛客370F

题意:在一棵树中选一个点S，删除一些边使得所有叶子结点到达不了S。

思路:设dp[i]表示以i为根的子树的所有叶子到达不了i的最小代价。dp[u]+=min(dp[v],w),v为u的孩子，w为u-v的边权。

## 数位DP

### 1.HDU 2089 不含62和4

#define inf 0x3f3f3f3f

#define ll long long

const int maxn=200005;

const int mod=1e9+7;

const double eps=1e-8;

const double PI = acos(-1.0);

#define lowbit(x) (x&(-x))

int a[65],dp[65][2];

int dfs(int pos,int if6,int limit)//pos:枚举到第几位 if6:前一位是否为6 limit:前一位是否达到上限

{

if(pos==0) return 1; //枚举到个位

if(!limit&&dp[pos][if6]) return dp[pos][if6]; //如果上一位没有到上限且dp记忆过，直接返回

int up=limit?a[pos]:9; //上一位有限制就只枚举到这一位上的数，否则0~9都要枚举

int ans=0; //记录数量

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

{

if(i==4) continue;//剪掉含4的数

if(if6&&i==2) continue;//剪掉上一位是6且这一位是2的数

ans+=dfs(pos-1,i==6,limit&&i==up);//往低位枚举，这一位是不是6，上一位和这一位是否达到上限

}

if(!limit) //如果没有达到上限，那么可以记忆

dp[pos][if6]=ans;

return ans;

}

int solve(int x)

{

int cnt=0;

while(x)

{

a[++cnt]=x%10;

x/=10;

}

return dfs(cnt,0,1);//从最高位枚举

}

int main()

{

std::ios::sync\_with\_stdio(false);

int n,m;

while(cin>>n>>m&&n+m)

{

cout<<solve(m)-solve(n-1)<<endl;

}

return 0;

}

### 2.统计0~9在1~n各出现了多少次

typedef long long ll;

const int N = 15;

ll f[N][2][N][2];

int num[N]; //num来存这个数每个位子上的数码

/\*

记忆化搜索。

len是当前为从高到低第几位。issmall表示当前位是否和num[len]相等，0是相等，1是不相等。

sum表示当前数字出现的次数。zero表示之前是否是前导0。d是当前在算的数码。

\*/

ll dfs(int len, bool issmall, int sum, bool zero, int d)

{

ll ret = 0;

if (len == 0) return sum; //边界条件

if (f[len][issmall][sum][zero] != -1) return f[len][issmall][sum][zero]; //记忆化

for (int i = 0; i < 10; i ++){

if (!issmall && i > num[len]) break;

/\*

由于我们是从高位到低位枚举的，所以如果之前一位的数码和最大数的数码相同，这一位就只能枚举到num[len]；

否则如果之前一位比最大数的数码小，那这一位就可以从0~9枚举了。

\*/

ret += dfs(len-1, issmall || (i<num[len]), sum+((!zero || i) && (i==d)), zero && (i == 0), d);

/\*

继续搜索，数位减一，issmall的更新要看之前有没有相等，且这一位有没有相等；

sum的更新要看之前是否为前导0或者这一位不是0；

zero的更新就看之前是否为前导0且这一位继续为0；

d继续传进去。

\*/

}

f[len][issmall][sum][zero] = ret;

//记忆化，把搜到的都记下来

return ret;

}

ll solve(ll x, int d)

{

int len = 0;

while (x){

num[++ len] = x%10;

x /= 10;

} //数字转数位

memset(f, -1, sizeof f); //初始化

return dfs(len, 0, 0, 1, d); //开始在第len位上，最高位只能枚举到num[len]所以issmall是0，sum=0，有前导0。

}

int main()

{

ll k; //注意都要开long long

while(~scanf("%lld", &k))

{

for (int i = 0; i < 10; i ++)

printf("%lld\n", solve(k, i));

}

return 0;

}

## 概率DP

1. 链接：<https://ac.nowcoder.com/acm/contest/368/C>

**题意**:在一共有n天，第i天如果有流星雨的话，会有wi颗流星雨。第i天有流星雨的概率是pi。如果第一天有流星雨了，那么第二天有流星雨的可能性是p2+P，否则是p2。

求n天后，流星雨颗数的期望。

第一行三个整数，n,a,b，其中n为天数,P=a/b,第二行n个整数wi。接下来n行，每行两个整数，x,y，第i+2行表示第i天有流星雨的概率pi=x/y。

**思路**:设L[i]为第i天真正有流星雨的概率,那么L[i]=(L[i-1]\*(p[i]+P)+(1-L[i-1])\*p[i],注意除法的地方用逆元,最后每一天的概率乘以w[i]求和即为答案。复杂度O(nlogn)。

#define inf 0x3f3f3f3f

#define ll long long

const int maxn=200005;

const int mod=1e9+7;

ll qpow(ll a,ll b){ll res=1;while(b){if(b&1) res=res\*a%mod;a=a\*a%mod;b>>=1;}return res;}

ll inv(ll a){return qpow(a,mod-2);}

ll w[maxn],L[maxn],p[maxn],P;

int main()

{

std::ios::sync\_with\_stdio(false);

ll n,a,b;

cin>>n>>a>>b;

P=a\*inv(b)%mod;

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

{

cin>>w[i];

}

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

{

ll x,y;

cin>>x>>y;

p[i]=x\*inv(y)%mod;

if(i==1) L[i]=p[i];

else L[i]=((p[i]+P)%mod\*L[i-1]%mod+(1-L[i-1]+mod)%mod\*p[i]%mod)%mod;

}

ll ans=0;

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

{

ans+=L[i]\*w[i]%mod;

ans%=mod;

}

cout<<ans<<endl;

return 0;

}

## 最长公共子序列(元素不同)O(nlogn)

int a[100001],a1[100001],c[100001],n,b[100001],d[100001],l,cnt,len,idx,ans;

int find(int l,int r,int x)

{

while(l<r)

{

int mid=(l+r)/2;

if(d[mid]>=x)

r=mid;

else

l=mid+1;

}

return l;

}

int main()

{

scanf("%d",&n);

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

{

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

a1[b[i]]=i;

}

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

{

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

a[i]=a1[c[i]];

}

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

{

idx=find(1,len+1,a[i]);

if(idx>len)

++len;

d[idx]=a[i];

}

printf("%d",len);

}

## 最长公共子序列+输出序列O(n^2)

//字符串版

const int maxn=1e3+10;

int dp[maxn][maxn];

int main()

{

std::ios::sync\_with\_stdio(false);

string s1,s2;

while(cin>>s1>>s2)

{

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

int l1=s1.length(),l2=s2.length();

for(int i=1;i<=l1;i++)

{

for(int j=1;j<=l2;j++)

{

if(s1[i-1]==s2[j-1])

{

dp[i][j]=dp[i-1][j-1]+1;

}

else

{

dp[i][j]=max(dp[i-1][j],dp[i][j-1]);

}

}

}

cout<<dp[l1][l2]<<endl;

}

return 0;

}

//输出子序列

#define inf 0x3f3f3f3f

#define ll long long

const int maxn=1005;

const int mod=1e9+7;

const double eps=1e-8;

const double PI = acos(-1.0);

#define lowbit(x) (x&(-x))

int dp[maxn][maxn];

int main()

{

std::ios::sync\_with\_stdio(false);

string s1,s2;

while(cin>>s1>>s2)

{

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

int l1=s1.length(),l2=s2.length();

for(int i=1; i<=l1; i++)

{

for(int j=1; j<=l2; j++)

{

if(s1[i-1]==s2[j-1])

{

dp[i][j]=dp[i-1][j-1]+1;

}

else

{

dp[i][j]=max(dp[i-1][j],dp[i][j-1]);

}

}

}

int i=l1,j=l2;

stack<char> s;

while(i&&j)

{

if(s1[i-1]==s2[j-1])

{

i--;

j--;

s.push(s1[i]);

}

else if(dp[i-1][j]>dp[i][j-1])

{

i--;

}

else if(dp[i-1][j]<=dp[i][j-1])

{

j--;

}

}

while(!s.empty())

{

cout<<s.top();

s.pop();

}

cout<<endl;

}

return 0;

}

## 最长公共子串+输出子串O(n^2)

int dp[1005][1005];

int main()

{

std::ios::sync\_with\_stdio(false);

string s1,s2;

while(cin>>s1>>s2)

{

int l1=s1.length(),l2=s2.length(),ans=0,last=0;

for(int i=1; i<=l1; i++)

{

for(int j=1; j<=l2; j++)

{

if(s1[i-1]==s2[j-1])

{

dp[i][j]=dp[i-1][j-1]+1;

if(ans<dp[i][j])

{

ans=dp[i][j];

last=i;

}

}

else

{

dp[i][j]=0;

}

}

}

if(ans)

{

for(int i=last-ans; i<last; i++)

cout<<s1[i];

cout<<endl;

}

else

cout<<"NULL"<<endl;

}

return 0;

}

## 最大连续子列和O(n)

#define inf 0x3f3f3f3f

#define ll long long

const int maxn=100005;

ll dp[maxn],a[maxn];

int main()

{

std::ios::sync\_with\_stdio(false);

int k;

cin>>k;

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

{

cin>>a[i];

}

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

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

{

if(a[i]+dp[i-1]>0)

{

dp[i]=dp[i-1]+a[i];

}

else if(a[i]+dp[i-1]<=0)

dp[i]=0;

}

ll ans=0;

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

{

ans=max(ans,dp[i]);

}

cout<<ans<<endl;

return 0;

}

## 最长递增子序列O(n^2)

dp[i]表示以第i个数结尾的LIS，第i个数可以选也可以不选，如果第i个数大于前i个数，则选，否则不选，所以外层循环i枚举以第i个数结尾，内层循环j枚举0~i的数，如果a[i]>a[j]，就更新dp[i]=max(dp[j]+1,dp[i])

#define inf 0x3f3f3f3f

#define ll long long

const int maxn=200005;

const double eps=1e-8;

int main()

{

std::ios::sync\_with\_stdio(false);

int n;

cin>>n;

int a[1005];

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

{

cin>>a[i];

}

int dp[1005];

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

{

dp[i]=1;

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

{

if(a[i]>a[j])

{

dp[i]=max(dp[j]+1,dp[i]);

}

}

}

int ans=0;

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

{

ans=max(ans,dp[i]);

}

cout<<ans<<endl;

return 0;

}

## 最长递增子序列O(nlogn)

/\*每次取栈顶元素和读到的元素做比较,如果大于，则将它入栈；如果小于，则二分查找栈中的比它大的第1个数，并替换它。最长序列长度即为最后模拟的大小。\*/

#define ll long long

const int maxn=100005;

ll a[maxn],cnt,s[maxn],n;//a是原序列,s是栈

ll lis()

{

cnt=0;

s[++cnt]=a[1];

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

{

if(a[i]>s[cnt]) s[++cnt]=a[i];

else s[lower\_bound(s+1,s+cnt+1,a[i])-s]=a[i];

}

return cnt;

}

int main()

{

std::ios::sync\_with\_stdio(false);

while(cin>>n)

{

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

{

cin>>a[i];

}

cout<<lis()<<endl;

}

return 0;

}

## 状压DP

### 1.棋盘铺1\*2和2\*1的木块。

题意:有一个N\*M(N<=5,M<=1000)的棋盘，现在有1\*2及2\*1的小木块无数个，要盖满整个棋盘，有多少种方式？答案只需要mod1,000,000,007即可。

思路:假设第一列已经填满，则第二列的摆设方式，只与第一列对第二列的影响有关。同理，第三列的摆设方式也只与第二列对它的影响有关。那么，使用一个长度为N的二进制数state来表示这个影响。dp[i][state]表示该填充第i列，第i-1列对它的影响是state的时候的方法数。i<=M,0<=state<2N

对于每一列，情况数也有很多，但由于N很小，所以可以采取搜索的办法去处理。对于每一列，搜索所有可能的放木块的情况，并记录它对下一列的影响，之后更新状态。状态转移方程如下：

dp[i][state]=∑dp[i-1][pre]每一个pre可以通过填放成为state。

//第i列，枚举到了第j行，当前状态是state，对下一列的影响是nex

#define ll long long

int n, m;

ll dp[1005][34];

void dfs(int i,int j,int state,int nex)

{

if (j==n)

{

dp[i+1][nex]+=dp[i][state];

return;

}

//如果这个位置已经被上一列所占用,直接跳过

if (((1<<j)&state)>0)

dfs(i,j+1,state,nex);

//如果这个位置是空的，尝试放一个1\*2的

if (((1<<j)&state)==0)

dfs(i,j+1,state,nex|(1<<j));

//如果这个位置以及下一个位置都是空的，尝试放一个2\*1的

if (j+1<n && ((1<<j)&state)==0 && ((1<<(j+1))&state)==0)

dfs(i,j+2,state,nex);

return;

}

int main()

{

while (cin>>n>>m)

{

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

if (n==0 && m==0) break;

dp[1][0]=1;

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

{

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

if (dp[i][j])

{

dfs(i,0,j,0);

}

}

cout<<dp[m+1][0]<<endl;

}

}

### 2.玉米田(洛谷p1879)

农场主John新买了一块长方形的新牧场，这块牧场被划分成M行N列(1 ≤ M ≤ 12; 1 ≤ N ≤ 12)，每一格都是一块正方形的土地。John打算在牧场上的某几格里种上美味的草，供他的奶牛们享用。遗憾的是，有些土地相当贫瘠，不能用来种草。并且，奶牛们喜欢独占一块草地的感觉，于是John不会选择两块相邻的土地，也就是说，没有哪两块草地有公共边。John想知道，如果不考虑草地的总块数，那么，共有多少种种植方案可供他选择？（当然，把新牧场完全荒废也是一种方案）

const int mod = 100000000;

int n, m;

int a[13][13];

int F[13];

//第[i]行的土地状态

int f[13][1 << 12 + 5];

//f[i][j]前[i]行的状态为j时的合法方案数 注意是前i行不是第i行

bool g[1 << 12 + 5];

//g[i]记录i这个状态是否合法

int main()

{

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

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

{

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

{

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

F[i] = (F[i] << 1) + a[i][j];

}

}

for (int i = 0; i < (1 << n); i++)

g[i] = (!(i & (i << 1))) && (!(i & (i >> 1)));

f[0][0] = 1;

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

{

//枚举每行

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

{

//枚举这行每个可能的状态

if (g[j] && ((j & F[i]) == j))

{

//这行的状态没有并排的1

//且状态j与F[i]相同

//使j与F[i]相同保证是在肥沃土地上种草

for (int k = 0; k < (1 << n); k++)

{

//枚举上一行的状态

if ((k & j) == 0)

{

//与该行状态取&为0说明上一行与这一行不存在任意一块草地有公共边

f[i][j] = (f[i][j] + f[i - 1][k]) % mod;

}

}

}

}

}

int ans = 0;

for (int i = 0; i < (1 << n); i++)

{

ans = (ans + f[m][i]) % mod;

//最后将前m行所有满足条件的方案数累加

}

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

return 0;

}

## 四维DP

### 洛谷P1004/P1006

题意：某人从图的左上角的A点出发，可以向下行走，也可以向右走，直到到达右下角的B点。在走过的路上，他可以取走方格中的数（取走后的方格中将变为数字0）。此人从A点到B点共走两次，试找出2条这样的路径，使得取得的数之和为最大。

输入格式：

输入的第一行为一个整数N（表示N×N的方格图），接下来的每行有三个整数，前两个表示位置，第三个数为该位置上所放的数。一行单独的0表示输入结束。

输出格式：

只需输出一个整数，表示2条路径上取得的最大的和。

思路：设dp[i][j][k][l]表示第一个人走到(i,j)的最大值和第二个人走到(k,l)的最大值。

int g[10][10],dp[10][10][10][10];

int main()

{

std::ios::sync\_with\_stdio(false);

int n;

cin>>n;

int a,b,c;

while(cin>>a>>b>>c&&a+b+c)

{

g[a][b]=c;

}

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

{

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

{

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

{

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

{

dp[i][j][k][l]=max(dp[i-1][j][k-1][l],max(max(dp[i-1][j][k][l-1],dp[i][j-1][k-1][l]),dp[i][j-1][k][l-1]))+g[i][j]+g[k][l];

if(i==k&&j==l)

dp[i][j][k][l]-=g[i][j];

}

}

}

}

cout<<dp[n][n][n][n]<<endl;

return 0;

}

## 插头DP

### 1.(洛谷p5056)求棋盘中的哈密顿回路个数，障碍不能走

第1行，n,m(2<=n,m<=12)

从第2行到第n+1行，每行一段字符串(m个字符)，"\*"表不能铺线，"."表必须铺

输出一个整数，表示总方案数

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

#define ll long long

#define hash ddf

using namespace std;

int n,m,x[15][15],cur,pre,ex,ey;

int st[2][300010];ll ans[2][300010],re;

int tot[2],bit[20],state[300010],st\_tot,hash=300000;

struct edge{

int to,next;

}a[300010];

void insert(int sta,ll val){

// cout<<"insert "<<sta<<ends<<val<<endl;

int p=sta%hash,i;

for(i=state[p];i;i=a[i].next){

if(st[cur][a[i].to]==sta){

ans[cur][a[i].to]+=val;return;

}

}

tot[cur]++;

a[++st\_tot].to=tot[cur];

a[st\_tot].next=state[p];

state[p]=st\_tot;st[cur][tot[cur]]=sta;ans[cur][tot[cur]]=val;

}

int main(){

int i,j,k,l,now,down,right;ll val;char s[20];

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

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

scanf("%s",s);

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

if(s[j]=='.')

x[i][j+1]=1,ex=i,ey=j+1;

}

for(i=1;i<15;i++) bit[i]=i<<1;

cur=0;tot[cur]=1;ans[cur][1]=1;st[cur][1]=0;

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

for(j=1;j<=tot[cur];j++) st[cur][j]<<=2;

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

// cout<<"begin "<<i<<ends<<j<<endl;

st\_tot=0;memset(state,0,sizeof(state));

pre=cur;cur^=1;tot[cur]=0;

for(k=1;k<=tot[pre];k++){

now=st[pre][k];val=ans[pre][k];

down=(now>>bit[j-1])%4;right=(now>>bit[j])%4;

// cout<<" from "<<now<<ends<<val<<ends<<down<<ends<<right<<endl;

if(!x[i][j]){

if(!down&&!right){

insert(now,val);continue;

}

}

else if(!down&&!right){

if(x[i][j+1]&&x[i+1][j])

insert(now+(1<<bit[j-1])+((1<<bit[j])<<1),val);

}

else if(!down&&right){

if(x[i][j+1]) insert(now,val);

if(x[i+1][j])

insert(now-right\*(1<<bit[j])+right\*(1<<bit[j-1]),val);

}

else if(down&&!right){

if(x[i+1][j]) insert(now,val);

if(x[i][j+1])

insert(now+down\*(1<<bit[j])-down\*(1<<bit[j-1]),val);

}

else if(down==1&&right==1){

int cnt=1;

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

if((now>>bit[l])%4==1) cnt++;

if((now>>bit[l])%4==2) cnt--;

if(!cnt){

insert(now-(1<<bit[l])-(1<<bit[j])-(1<<bit[j-1]),val);

break;

}

}

}

else if(down==2&&right==2){

int cnt=1;

for(l=j-2;l>=0;l--){

if((now>>bit[l])%4==2) cnt++;

if((now>>bit[l])%4==1) cnt--;

if(!cnt){

insert(now+(1<<bit[l])-((1<<bit[j])<<1)-((1<<bit[j-1])<<1),val);

break;

}

}

}

else if(down==2&&right==1){

insert(now-((1<<bit[j-1])<<1)-(1<<bit[j]),val);

}

else if(down==1&&right==2){

if(i==ex&&j==ey) re+=val;

}

}

}

}

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

}

### (洛谷p3272)用L型的地板铺满整个客厅有多少种不同的方案？

需要注意的是，L型地板的两端长度可以任意变化，但不能长度为0。输入的第一行包含两个整数，R和C，表示客厅的大小。接着是R行，每行C个字符。'\_'表示对应的位置是空的，必须铺地板；'\*'表示对应的位置有柱子，不能铺地板。

输出方案数%20110520。

#define hash deep\_dark\_fantasy

#define ll long long

#define MOD 20110520

using namespace std;

inline int read(){

int re=0,flag=1;char ch=getchar();

while(ch>'9'||ch<'0'){

if(ch=='-') flag=-1;

ch=getchar();

}

while(ch>='0'&&ch<='9') re=(re<<1)+(re<<3)+ch-'0',ch=getchar();

return re\*flag;

}

int n,m,x[150][150],cur,pre,ex,ey;

int st[2][300010];ll ans[2][300010],re;

int tot[2],bit[20],state[300010],st\_tot,hash=300000;

struct edge{

int to,next;

}a[300010];

void insert(int now,ll val){

int p=now%hash;

for(int i=state[p];i;i=a[i].next){

if(st[cur][a[i].to]==now){

ans[cur][a[i].to]+=val;

ans[cur][a[i].to]%=MOD;return;

}

}

tot[cur]++;

a[++st\_tot].to=tot[cur];

a[st\_tot].next=state[p];

state[p]=st\_tot;st[cur][tot[cur]]=now;ans[cur][tot[cur]]=val%MOD;

}

void dp(){

int i,j,k,down,right,now;ll val;

cur=0;tot[cur]=1;ans[cur][1]=1;st[cur][1]=0;

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

for(j=1;j<=tot[cur];j++) st[cur][j]<<=2;

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

memset(state,0,sizeof(state));st\_tot=0;

pre=cur;cur^=1;tot[cur]=0;

for(k=1;k<=tot[pre];k++){

now=st[pre][k];val=ans[pre][k];

right=(now>>bit[j-1])%4;down=(now>>bit[j])%4;

if(!x[i][j]){//障碍格子

if(!down&&!right){

insert(now,val);continue;

}

}

if(!right&&!down){//第一种情况

if(x[i+1][j]&&x[i][j+1])

insert(now+((1<<bit[j-1])<<1)+((1<<bit[j])<<1),val);

if(x[i+1][j]) insert(now+(1<<bit[j-1]),val);

if(x[i][j+1]) insert(now+(1<<bit[j]),val);

}

if(right==1&&!down){//第三种情况

if(x[i][j+1]) insert(now-(1<<bit[j-1])+(1<<bit[j]),val);

if(x[i+1][j]) insert(now+(1<<bit[j-1]),val);

}

if(down==1&&!right){//第二种情况

if(x[i+1][j]) insert(now-(1<<bit[j])+(1<<bit[j-1]),val);

if(x[i][j+1]) insert(now+(1<<bit[j]),val);

}

if(right==2&&!down){//第五种情况

if(i==ex&&j==ey) re+=val,re%=MOD;

if(x[i][j+1]) insert(now-((1<<bit[j-1])<<1)+((1<<bit[j])<<1),val);

insert(now-((1<<bit[j-1])<<1),val);

}

if(down==2&&!right){//第四种情况

if(i==ex&&j==ey) re+=val,re%=MOD;

if(x[i+1][j]) insert(now-((1<<bit[j])<<1)+((1<<bit[j-1])<<1),val);

insert(now-((1<<bit[j])<<1),val);

}

if(down==1&&right==1){//第六种情况

if(i==ex&&j==ey) re+=val,re%=MOD;

insert(now-(1<<bit[j-1])-(1<<bit[j]),val);

}

}

}

}

}

int main(){

int i,j;char ch;

n=read();m=read();

for(i=1;i<=20;i++) bit[i]=i<<1;

if(n>m){

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

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

ch=getchar();

while(ch!='\*'&&ch!='\_') ch=getchar();

x[i][j]=(ch=='\_');

if(x[i][j]) ex=i,ey=j;

}

}

}

else{

swap(n,m);

for(i=m;i>0;i--){

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

ch=getchar();

while(ch!='\*'&&ch!='\_') ch=getchar();

x[j][i]=(ch=='\_');

if(x[j][i]&&((j>ex)||(j==ex&&i>ey))) ex=j,ey=i;

}

}

}

dp();

printf("%lld",re);

}

### 3.(洛谷p3190)求棋盘中权值最大的回路。

#define ll long long

#define hash deep\_dark\_fantasy

#define inf 1e9

using namespace std;

inline int read(){

int re=0,flag=1;char ch=getchar();

while(ch>'9'||ch<'0'){

if(ch=='-') flag=-1;

ch=getchar();

}

while(ch>='0'&&ch<='9') re=(re<<1)+(re<<3)+ch-'0',ch=getchar();

return re\*flag;

}

int n,m,x[150][150],cur,pre,ex,ey;

int st[2][300010];ll ans[2][300010],re;

int tot[2],bit[20],state[300010],st\_tot,hash=300000;

struct edge{

int to,next;

}a[300010];

void insert(int sta,ll val){

int p=sta%hash,i;

for(i=state[p];i;i=a[i].next){

if(st[cur][a[i].to]==sta){

ans[cur][a[i].to]=max(ans[cur][a[i].to],val);return;

}

}

tot[cur]++;

a[++st\_tot].to=tot[cur];

a[st\_tot].next=state[p];

state[p]=st\_tot;st[cur][tot[cur]]=sta;ans[cur][tot[cur]]=val;

}

void dp(){

int i,j,k,l,now,down,right;ll val;re=-inf;

cur=0;tot[cur]=1;ans[cur][1]=0;st[cur][1]=0;

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

for(j=1;j<=tot[cur];j++) st[cur][j]<<=2;

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

pre=cur;cur^=1;tot[cur]=0;st\_tot=0;memset(state,0,sizeof(state));

for(k=1;k<=tot[pre];k++){

now=st[pre][k];val=ans[pre][k];

right=(now>>bit[j-1])%4;down=(now>>bit[j])%4;

if(!down&&!right){

insert(now,val);

if(j!=m)

insert(now+(1<<bit[j-1])+((1<<bit[j])<<1),val+x[i][j]);

}

if(down&&!right){

insert(now-down\*(1<<bit[j])+down\*(1<<bit[j-1]),val+x[i][j]);

if(j!=m)insert(now,val+x[i][j]);

}

if(right&&!down){

insert(now,val+x[i][j]);

if(j!=m)

insert(now+right\*(1<<bit[j])-right\*(1<<bit[j-1]),val+x[i][j]);

}

if(right==1&&down==1){

int cnt=1;

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

if((now>>bit[l])%4==1) cnt++;

if((now>>bit[l])%4==2) cnt--;

if(!cnt){

insert(now-(1<<bit[l])-(1<<bit[j])-(1<<bit[j-1]),val+x[i][j]);

break;

}

}

}

if(right==2&&down==2){

int cnt=1;

for(l=j-2;l>=0;l--){

if((now>>bit[l])%4==1) cnt--;

if((now>>bit[l])%4==2) cnt++;

if(!cnt){

insert(now+(1<<bit[l])-((1<<bit[j])<<1)-((1<<bit[j-1])<<1),val+x[i][j]);

break;

}

}

}

if(right==2&&down==1){

insert(now-((1<<bit[j-1])<<1)-(1<<bit[j]),val+x[i][j]);

}

if(right==1&&down==2){

if((now==(1<<bit[j-1])+((1<<bit[j])<<1))&&(val+x[i][j]>re)){

re=val+x[i][j];

}

}

}

}

}

}

int main(){

int i,j;

n=read();m=read();

for(i=1;i<=10;i++) bit[i]=(i<<1);

for(i=1;i<=n;i++) for(j=1;j<=m;j++) x[i][j]=read();

dp();

printf("%lld",re);

}

# 数学

## 一、基本定理

### 1、鸽巢原理

第一抽屉原理

原理1

把多于n+1个物体放到n个抽屉里，则至少有一个抽屉里的东西不少于两件。

原理2

把多于m\*n+1(n不为0)个物体放到n个抽屉里面，则至少有一个抽屉里面不少于(m+1)的物体。

第二抽屉原理

把(m\*n -1 )个物体放入n个抽屉中，其中必须有一个抽屉不多于(m-1)个物体。

如将3\*5-1 = 14个物体放入5个抽屉中，则必定有一个抽屉中的物体数目少于3-1=2.

例题：

1.HDU1205：不能把同种类的糖果放在一起吃，只能先吃一种，下次吃另一种。问是否存在一种吃糖果的顺序使他能把所有糖果吃完。

思路：找到最大数量的糖果种类，设最大数量为ma,然后在ma-1个空里插其余糖果即可。因为其他糖果数量都比最大的少，所以不会出现两个同类的糖果插在一个空。即判断ma>=除最大数量的糖果外其他糖果数之和-1。

### 2、计算n^2的因子数

举个栗子，比如n=12，拆成质数幂之积(2^2)\*3，由唯一分解定理，12的因子数为(2+1)\*(1+1)=6，那么12的平方144可以写成（2^2\*3）^2=2^(2\*2)\*3^(1\*2)，所以12^2的因子数为(2\*2+1)\*(2\*1+1)=15，从这里就可以看出只需要对n求根号a以内的因子即可求得n^2的因子数。当然，如果最后得到的是个质数，那么有可能找不到根号n以内的因子，特判一下即可，这个数相当于质数的一次方，所以乘上(2\*1+1)=3即可。

int sn=sqrt(n);

ll ans=1;

for(int i=2; i<=sn; i++)

{

int cnt=0;

while(n%i==0&&n>1)

{

cnt++;

n/=i;

}

ans\*=(2\*cnt+1);

}

if(n!=1)

ans\*=3;

### 矩阵快速幂

#define mod 7

#define ll long long

int n=2; //方阵的行列数

struct Matrix

{

ll m[105][105]; //矩阵结构体

};

Matrix operator\*(Matrix a,Matrix b) //模拟矩阵乘法

{

Matrix c;

memset(c.m,0,sizeof(c.m));

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

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

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

c.m[i][j]=(c.m[i][j]+a.m[i][k]\*b.m[k][j]+mod)%mod;

return c;

}

Matrix operator^(Matrix a,int b) //快速幂 矩阵a的b次方

{

Matrix c;

memset(c.m,0,sizeof(c.m));

for(int i=1; i<=n; i++) //初始化为单位矩阵

c.m[i][i]=1;

while(b)

{

if(b&1)

c=c\*a; //快速幂

a=a\*a;

b>>=1;

}

return c;

}

int main()

{

int a,b,c;

std::ios::sync\_with\_stdio(false);

while(cin>>a>>b>>c)

{

if(!a) break;

if(c<3)

cout<<1<<endl;

else

{

Matrix M;

M.m[1][1]=a,M.m[1][2]=b,M.m[2][1]=1,M.m[2][2]=0;

Matrix ans=M^(c-2);

cout<<(ans.m[1][1]+ans.m[1][2])%mod<<endl;

}

}

return 0;

}

### 4、十进制快速幂

比如求2^N次方，N是特别大的数，用string存

#include<bits/stdc++.h>

using namespace std;

#define ll long long

const int maxn=1e7+10;

ll a, mod;

char ss[maxn];

ll qpow ( ll a, ll b, ll mod )

{

ll rt;

for (rt = 1; b; b >>= 1, a = a \* a % mod)

if (b & 1)

rt = rt \* a % mod;

return rt;

}

int main ()

{

scanf ("%lld%lld%s", &a, &mod,ss);

int len = strlen(ss);

reverse(ss,ss+len);

ll base = a;

ll cur = 1;

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

{

cur = cur \* qpow (base, ss[i] - '0', mod) % mod;

base = qpow (base, 10, mod) % mod;

}

printf ("%lld", cur);

}

### 5、欧拉函数

欧拉函数的定义：对正整数n，欧拉函数是小于或等于n的数中与n互质的数的数目。例如euler(8)=4，因为1,3,5,7均和8互质。

Euler函数表达通式：euler(x)=x(1-1/p1)(1-1/p2)(1-1/p3)(1-1/p4)…(1-1/pn),其中p1,p2……pn为x的所有素因数，x是不为0的整数。euler(1)=1（唯一和1互质的数就是1本身）。?

欧拉公式的延伸：一个数的所有质因子之和是euler(n)\*n/2。

直接求：

long long phi(long long x)

{

int res = x,a = x;

for(int i=2;i\*i<=a;i++)

{

if(a%i==0)

{

res = res/i\*(i-1);//res -= res/i;

while(a%i==0)a/=i;

}

}

if(a>1)res =res/a\*(a-1);//res -= res/a;

return res;

}

打表：

#include<bits/stdc++.h>

using namespace std;

#define Max 1000001

int euler[Max];

int main(){

euler[1]=1;

for(int i=2;i<Max;i++)

euler[i]=i;

for(int i=2;i<Max;i++)

if(euler[i]==i)

for(int j=i;j<Max;j+=i)

euler[j]=euler[j]/i\*(i-1);//先进行除法是为了防止中间数据的溢出

}

### 6、母函数

//找凑不出来的第一个钱。

#include<bits/stdc++.h>

using namespace std;

const int maxn=1e5;

int a[maxn],b[maxn],n[maxn],v[3]={1,2,5};//a储存多项式系数,b是中间变量

int main()

{

while(cin>>n[0]>>n[1]>>n[2]&&n[0]+n[1]+n[2])

{

memset(a,0,sizeof(a));

int mx=n[0]\*1+n[1]\*2+n[2]\*5;

for(int i=0;i<=n[0];i++) a[i]=1; //初始化第一个多项式1+x+x^2+…+x^n

for (int i=1; i<=2; i++) //1~i多项式相乘

{

memset(b,0,sizeof(b));

for(int j=0;j<=mx;j++) //对于之前的项

{

if(a[j])

for(int k=0;k+j<=mx&&k<=v[i]\*n[i];k+=v[i]) //1+x^2i+x^3i+…x^n

{

b[j+k]+=a[j];

}

}

memcpy(a,b,sizeof(b));//b赋值给a

}

int i;

for(i=0;i<=mx;++i)

{

if(!a[i]) break;

}

cout<<i<<endl;

}

}

//hdu1028 求n被划分成整数之和的方案数

#include<bits/stdc++.h>

using namespace std;

const int maxn=1e5;

int a[maxn],b[maxn],n;//a储存多项式系数,b是中间变量

int main()

{

while(cin>>n)

{

for(int i=0;i<=n;i++) a[i]=1; //初始化第一个多项式1+x+x^2+…+x^n

for (int i=2; i<=n; i++) //1~i多项式相乘

{

memset(b,0,sizeof(b));

for(int j=0;j<=n;j++) //对于之前的项

{

for(int k=0;k+j<=n;k+=i) //1+x^2i+x^3i+…x^n

{

b[j+k]+=a[j];

}

}

memcpy(a,b,sizeof(b));//b赋值给a

}

cout<<a[n]<<endl;

}

}

### 7、阶乘尾0

#include<bits/stdc++.h>

using namespace std;

int main()

{

std::ios::sync\_with\_stdio(false);

int m;

cin>>m;

while(m--)

{

int n;

cin>>n;

int t=n,ans=0;

while(n)

{

ans+=n/5;

n/=5;

}

cout<<ans<<endl;

}

return 0;

}

### 8、扩展埃式筛法

比如因子数量是3的最小数是4

筛出每个因子数量对应的数的最小值。如果因子数量是n,则这个数是b[n]。

void init() //扩展埃式筛法

{

for (int i = 1; i <= 1e6; i++)

{

for (int j = i; j <= 1e6; j+=i)

{

a[j]++;

}

if(!b[a[i]]) //保证是最小的

{

b[a[i]]=i;//下标保存因子数，值对应的最小值

}

}

}

### 9、计算两个数的所有公约数

//https://ac.nowcoder.com/acm/contest/280/C

计算两数的所有公约数，复杂度是nlogn

#include<bits/stdc++.h>

using namespace std;

#define inf 0x3f3f3f3f

#define ll long long

const int maxn=200005;

const int mod=1e9+7;

const double eps=1e-8;

const double PI = acos(-1.0);

#define lowbit(x) (x&(-x))

ll gcd(ll a,ll b){return b==0?a:gcd(b,a%b);}

ll g[maxn];

int main()

{

std::ios::sync\_with\_stdio(false);

ll a,b,cnt=0;

cin>>a>>b;

ll gab=gcd(a,b);

for(int i=1;i<=sqrt(gab);i++)

{

if(a%i==0&&b%i==0)

{

if(i\*i==gab)

g[cnt++]=i;

else

g[cnt++]=i,g[cnt++]=gab/i;

}

}

sort(g,g+cnt);

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

{

cout<<g[i]<<" ";

}

return 0;

}

### 10、线性基

线性基概念：

1、若干数的线性基是一组数a1,a2,...an，其中ax的最高位的1在第x位。通过线性基中元素xor出的数的值域与原来的数xor出数的值域相同。

2、只要有关异或运算和求最值，就可以用到线性基

3、同时线性基的任何一个非空子集都不会使得其xor和为0，

### 求一个数能不能由一群数异或得到

#include<bits/stdc++.h>

using namespace std;

#define ll long long

const int maxn=200005;

const int maxbit=63;

ll p[maxbit],a[maxn];//p储存第i位为1的数

int main()

{

std::ios::sync\_with\_stdio(false);

int n;

cin>>n;

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

cin>>a[i];

for(int i=1;i<=n;i++)//求线性基

{

for(int j=maxbit-1;j>=0;j--)

{

if((a[i]>>j)&1)//如果一个数的第j位为1

{

if(p[j]==0)//如果第j位还未赋值

{

p[j]=a[i];break;

}

else

{

a[i]^=p[j];//否则就异或这个已存在的p[j]使得a的第j位变成0

}

}

}

}

int num=0;//线性基的大小

for(int i=maxbit-1;i>=0;i--)

{

if(p[i])

{

num++;

}

}

int k;

while(cin>>k)

{

if(k==0)

{

if(n==num)//每一位为1的只有一个，无法异或使得每一位都为0

{

cout<<"false"<<endl;

}

else

{

cout<<"true"<<endl;

}

}

else

{

bool flag=0;

for(int j=maxbit-1;j>=0;j--)

{

if((k>>j)&1)//尝试把k异或成0

{

k^=p[j];

}

if(k==0)

{

flag=1;break;

}

}

if(flag)

{

cout<<"true"<<endl;

}

else

{

cout<<"false"<<endl;

}

}

}

return 0;

}

### 求由一群数异或得到的所有数第k小的数

#include<bits/stdc++.h>

using namespace std;

#define ll long long

const int maxn=200005;

const int maxbit=63;

ll a[maxn],p[maxbit],q[maxbit];

int main()

{

std::ios::sync\_with\_stdio(false);

int t,cas=0;

cin>>t;

while(t--)

{

memset(p,0,sizeof(p));

memset(q,0,sizeof(q));

cout<<"Case #"<<++cas<<":"<<endl;

int n;

cin>>n;

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

{

cin>>a[i];

}

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

{

for(int j=maxbit-1; j>=0; j--)

{

if((a[i]>>j)&1)//如果第j位为1

{

if(p[j]==0)

{

p[j]=a[i];

break;

}

else

{

a[i]^=p[j];

}

}

}

}

int num=0;

for(int i=0; i<maxbit; i++) //把高位的第i位也是1的那一位异或成0

{

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

{

if((p[j]>>i)&1)

{

p[j]^=p[i];

}

}

}

for(int i=0; i<maxbit; i++) //从低到高放到新的数组q中

{

if(p[i])

{

q[num++]=p[i];

}

}

int Q;

cin>>Q;

while(Q--)

{

ll k;

cin>>k;

if(num!=n)//可以异或得0

{

k--;

}

if(k>=(1LL<<num))//超出能表示的个数，不包含0能表示2^num-1个数

{

cout<<-1<<endl;

}

else

{

ll ans=0;

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

{

if((k>>i)&1)

{

ans^=q[i];

}

}

cout<<ans<<endl;

}

}

}

return 0;

}

### 一群数任选使异或值最大

//3.一群数任选使异或值最大：从高位到低位扫描。如果当前ans^p[i]能使得答案变大，那么就异或。最后得到的ans就是线性基异或集合中的最大值。

#include<bits/stdc++.h>

using namespace std;

#define ll long long

const int maxn=200005;

const int maxbit=63;

ll p[maxbit],a[maxn];//p储存第i位为1的数

int main()

{

std::ios::sync\_with\_stdio(false);

int n;

cin>>n;

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

cin>>a[i];

for(int i=1;i<=n;i++)//求线性基

{

for(int j=maxbit-1;j>=0;j--)

{

if((a[i]>>j)&1)//如果一个数的第j位为1

{

if(p[j]==0)//如果第j位还未赋值

{

p[j]=a[i];break;

}

else

{

a[i]^=p[j];//否则就异或这个已存在的p[j]使得a的第j位变成0

}

}

}

}

ll ans=0;

for(int i=maxbit-1;i>=0;i--)

{

if((ans^p[i])>ans) //注意前面要加上括号，^的优先级比>低

ans=ans^p[i];

}

cout<<ans<<endl;

return 0;

}

### 一群值任选使异或值最小

//4.一群数任选使异或值最小：最小值就是线性基中最低位的数。

### 二维问题

//5.在a[i].a异或不为0的情况下，a[i].b的和的最大值:按b从大到小排序，按这个顺序遍历一下a[i].a，如果能插入线性基则ans加上a[i].b。

#include<bits/stdc++.h>

using namespace std;

#define ll long long

const int maxn=200005;

const int maxbit=63;

ll p[maxbit];//p储存第i位为1的数

struct node

{

ll a,v;

friend bool operator<(node a,node b)

{

return a.v>b.v;

}

};

node a[maxn];

int main()

{

std::ios::sync\_with\_stdio(false);

int n;

cin>>n;

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

cin>>a[i].a>>a[i].v;

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

ll ans=0;

for(int i=1;i<=n;i++)//求线性基

{

for(int j=maxbit-1;j>=0;j--)

{

if((a[i].a>>j)&1)//如果一个数的第j位为1

{

if(p[j]==0)//如果第j位还未赋值

{

p[j]=a[i].a;break;

}

else

{

a[i].a^=p[j];//否则就异或这个已存在的p[j]使得a的第j位变0

}

}

}

if(a[i].a!=0)

ans+=a[i].v;

}

cout<<ans<<endl;

return 0;

}

### 在图中找一条1到n的路径，使边权异或和最大。

#include<bits/stdc++.h>

using namespace std;

#define ll long long

const int maxn=5e5+10;

const int maxbit=64;

struct node

{

ll to,val;

};

vector<node> g[maxn];

ll dis[maxn],ans,cir[maxn],sz=0,p[maxbit];

int vis[maxn];

void dfs(ll now,ll fa,ll x)

{

dis[now]=x;

vis[now]=1;

for(int i=0;i<g[now].size();i++)

{

int to=g[now][i].to;

if(to!=fa&&vis[to])

{

cir[++sz]=dis[now]^dis[to]^g[now][i].val;

}

else if(!vis[to])

{

dfs(to,now,x^g[now][i].val);

}

}

}

int main()

{

std::ios::sync\_with\_stdio(false);

int n,m;

cin>>n>>m;

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

{

ll u,v,w;

cin>>u>>v>>w;

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

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

}

dfs(1,0,0);

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

{

for(int j=maxbit-1;j>=0;j--)

{

if((cir[i]>>j)&1)

{

if(p[j]==0)

{

p[j]=cir[i];break;

}

else

cir[i]^=p[j];

}

}

}

ans=dis[n];

for(int i=maxbit-1;i>=0;i--)

{

ans=max(ans,ans^p[i]);

}

cout<<ans<<endl;

return 0;

}

### 求1到n异或最大和

//7.求1到n异或最大和，i能到j当且仅当a[i]>a[j]，花费为^a[j]，初始为a[1]。https://ac.nowcoder.com/acm/problem/22144

#include<bits/stdc++.h>

using namespace std;

#define ll long long

const int maxn=3005;

const int maxbit=64;

int ans,sz=0,p[maxbit];

int a[maxn],b[maxn];

int main()

{

std::ios::sync\_with\_stdio(false);

int n;

cin>>n;

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

{

cin>>a[i];

}

if(a[1]<=a[n])

{

cout<<-1<<endl;

return 0;

}

ans=a[1]^a[n];

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

{

if(a[i]<a[1]&&a[i]>a[n])

{

b[++sz]=a[i];

}

}

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

{

for(int j=maxbit-1; j>=0; j--)

{

if((b[i]>>j)&1)

{

if(p[j]==0)

{

p[j]=b[i];

break;

}

else

b[i]^=p[j];

}

}

}

for(int i=maxbit-1; i>=0; i--)

{

ans=max(ans,ans^p[i]);

}

cout<<ans<<endl;

return 0;

}

### 基础数论模板（逆元、lucas、快速乘、中国剩余定理）

HDU 5446 求C(n,m)%(p1\*p2\*…pk)，其中pi为素数

逆元+快速乘+lucas+中国剩余定理

1.中国剩余定理（CRT）:

设m1,…，mk是两两互质的正整数，那么对于a1,…,ak来说，一次同余方程组x≡ai(modmi)，必有解唯一解为 x=∑(1~k) Mi\*ai\*Mi^(-1)(mod m)m=m1\*m2\*…mk，Mi=m/mi，Mi^(-1)为Mi对于模mi的逆.

typedef long long ll;

const int maxn =1e5+10;

ll n,m,p[15],M,k;//M=p1\*p2\*…pk,p是小素数

ll mul(ll a,ll b,ll p) //快速乘取模，计算a\*b%p

{

ll res=0;

while(b)

{

if(b&1) res =(res+a)%p;

a=(a+a)%p;

b>>=1;

}

return res;

}

ll qpow(ll a,ll b,ll p)//快速幂 a^b%p

{

ll t=a%p,ans=1;

while(b)

{

if(b&1) ans=ans\*t%p;

t=t\*t%p;

b>>=1;

}

return ans;

}

ll inv(ll a,ll p)//逆元

{

return qpow(a,p-2,p);

}

ll C(ll n,ll m,ll p) //求组合数C(n,m)%p

{

if(m>n||n<0||m<0) return 0;

if(m==0||n==m) return 1;

if(m>n-m) m=n-m;

ll ca=1,cb=1;

for(ll i = 0; i<m; i++)

{

ca=ca\*(n-i)%p;

cb=cb\*(m-i)%p;

}

return ca\*inv(cb,p)%p;

}

ll Lucas(ll n,ll m,ll p)//大组合数C(n,m)%p

{

if(m ==0) return 1;

else return (C(n%p,m%p,p)\*Lucas(n/p,m/p,p))%p;

}

ll china()//中国剩余定理

{

ll ans=0;

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

{

ans=(ans+mul(mul(Lucas(n,m,p[i]),M/p[i],M),inv(M/p[i],p[i]),M))%M;

}

return ans;

}

int main()

{

ios::sync\_with\_stdio(false);

int t;

scanf("%d",&t);

while(t--)

{

scanf("%lld%lld%lld",&n,&m,&k);

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

{

scanf("%lld",&p[i]);

}

M=1;

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

{

M\*=p[i];

}

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

}

return 0;

}

### 递推公式模板

把前几项丢进去，可以算出第N项的结果

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

#include <vector>

#include <string>

#include <map>

#include <set>

#include <cassert>[[1]](#endnote-0)

#include<bits/stdc++.h>

using namespace std;

#define rep(i,a,n) for (int i=a;i<n;i++)

#define per(i,a,n) for (int i=n-1;i>=a;i--)

#define pb push\_back

#define mp make\_pair

#define all(x) (x).begin(),(x).end()

#define fi first

#define se second

#define SZ(x) ((int)(x).size())

typedef vector<int> VI;

typedef long long ll;

typedef pair<int,int> PII;

const ll mod=1000000007;

ll powmod(ll a,ll b) {ll res=1;a%=mod; assert(b>=0); for(;b;b>>=1){if(b&1)res=res\*a%mod;a=a\*a%mod;}return res;}

// head

int \_,n;

namespace linear\_seq {

const int N=10010;

ll res[N],base[N],\_c[N],\_md[N];

vector<int> Md;

void mul(ll \*a,ll \*b,int k) {

rep(i,0,k+k) \_c[i]=0;

rep(i,0,k) if (a[i]) rep(j,0,k) \_c[i+j]=(\_c[i+j]+a[i]\*b[j])%mod;

for (int i=k+k-1;i>=k;i--) if (\_c[i])

rep(j,0,SZ(Md)) \_c[i-k+Md[j]]=(\_c[i-k+Md[j]]-\_c[i]\*\_md[Md[j]])%mod;

rep(i,0,k) a[i]=\_c[i];

}

int solve(ll n,VI a,VI b) { // a 系数 b 初值 b[n+1]=a[0]\*b[n]+...

// printf("%d\n",SZ(b));

ll ans=0,pnt=0;

int k=SZ(a);

assert(SZ(a)==SZ(b));

rep(i,0,k) \_md[k-1-i]=-a[i];\_md[k]=1;

Md.clear();

rep(i,0,k) if (\_md[i]!=0) Md.push\_back(i);

rep(i,0,k) res[i]=base[i]=0;

res[0]=1;

while ((1ll<<pnt)<=n) pnt++;

for (int p=pnt;p>=0;p--) {

mul(res,res,k);

if ((n>>p)&1) {

for (int i=k-1;i>=0;i--) res[i+1]=res[i];res[0]=0;

rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]]-res[k]\*\_md[Md[j]])%mod;

}

}

rep(i,0,k) ans=(ans+res[i]\*b[i])%mod;

if (ans<0) ans+=mod;

return ans;

}

VI BM(VI s) {

VI C(1,1),B(1,1);

int L=0,m=1,b=1;

rep(n,0,SZ(s)) {

ll d=0;

rep(i,0,L+1) d=(d+(ll)C[i]\*s[n-i])%mod;

if (d==0) ++m;

else if (2\*L<=n) {

VI T=C;

ll c=mod-d\*powmod(b,mod-2)%mod;

while (SZ(C)<SZ(B)+m) C.pb(0);

rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c\*B[i])%mod;

L=n+1-L; B=T; b=d; m=1;

} else {

ll c=mod-d\*powmod(b,mod-2)%mod;

while (SZ(C)<SZ(B)+m) C.pb(0);

rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c\*B[i])%mod;

++m;

}

}

return C;

}

int gao(VI a,ll n) {

VI c=BM(a);

c.erase(c.begin());

rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;

return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));

}

};

int main() {

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

vector<int>v;

v.push\_back(1);

v.push\_back(2);

v.push\_back(4);

v.push\_back(7);

v.push\_back(13);

v.push\_back(24);

//VI{1,2,4,7,13,24}

printf("%d\n",linear\_seq::gao(v,n-1));

}

}

### 博弈论

### 基础博弈

### 1. 巴士博弈

1堆石头 一次可取1-m个 结论:n%(m+1)=0时先手必败

### 2. 威佐夫博弈

有两堆若干个物品，两个人轮流从某一堆或同时从两堆取同样多的物品，规定每次最少取1个，最后取光者胜。

时先手必败。

### 3. 斐波那契博弈

1堆石子有n个，两个轮流取。先取者第一次可以取任意多个，但不能全部取完。以后每次取的石子数不能超过上次取子数的2倍，取完者胜。

结论:当n为斐波那契数时，先手必败。

### 4. nim博弈

有n堆若干个物品，两个轮流从某一堆取任意多的物品，规定每次至少取一个，多者不限，最后取光者得胜。异或和为0时先手必败。

### 高阶博弈

### 1. 抛硬币问题

一般的翻硬币的游戏规则是这样的:

N枚硬币排成一排，有的正面朝上，有的反面朝上。我们从左开始对硬币按1到N编号。

第一，游戏者根据某些约束翻硬币，但他所翻动的硬币中，最右边那个硬币必须是从正面翻到方面。例如只能翻3个硬币的情况，那么第三个硬币必须从正面翻到反面。如果局面是正正反，那么就不能翻硬币了，因为第三个是反的。

第二，谁不能翻谁输。

有这样的结论:局面的SG值为局面中每个正面朝上的妻子单一存在时的SG值异或和。即一个有k个硬币朝上，朝上硬币位置分布在的翻硬币游戏中，SG值等于k个独立的开始时只有一个硬币朝上的翻硬币游戏的SG值异或和。比如THHTTH这个游戏中，2号，3号，6号位是朝上的，它等价于TH,TTH,TTTTTH三个游戏和，即SG[THHTTH]=SG[TH]^SG[TTH]^SG[TTTTTH]。

**约束条件一:每次只能翻一个硬币。**

显然对于任意一个正面的硬币,SG=1所以奇数个正面硬币先手必胜，偶数个正面硬币，先手必败。

**约束条件二:每次能翻转一个或两个硬币(不用连续)**

每个硬币的SG值为它的编号，初始编号为1，与NIM游戏一样。

**约束条件三:每次必须连续反转k个硬币**

设k=3，枚举一下sg值可以发现sg值的规律为001001001….那么推广一下就是k-1个0加1个1作为一个循环节。

**约束条件4:每次翻动一个硬币后，必须翻动其左侧最近三个硬币中的一个，即翻动第x个硬币后，必须选择x-1,x-2,x-3中的其中一个硬币进行翻动，除非x是小于等于3的。(Subtraction Games)**

当N==1时，硬币为:正，先手必赢，所以sg[1]=1.

当N==2时，硬币为反正，因为先手可以翻成反反或正反，可能性为2，所以sg[2]=2

当N==3时，硬币为反反正，先手操作后可以为反反反，正反反，反正反。所以sg[3]=3

位置x:0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Sg[x]:0 1 2 3 0 1 2 3 0 1 2 3 0 1 2

**约束条件5:每次必须翻动两个硬币，而且这两个硬币的距离在可行集S={1,2,3}中，硬币序号从0开始。(Twins游戏)**

当N==1时，硬币为:正 先手必输 所以sg[0]=0

当N==2时，硬币为反正 先手必赢，所以sg[1]=1

当N==3时，硬币为反反正，先手必应，sg[2]=2

当N==4时，sg[3]=3

当N==5时，sg[4]=0

位置x:0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Sg[x]:0 1 2 3 0 1 2 3 0 1 2 3 0 1 2

**约束条件6:每次可以翻动一个，二个或三个硬币(Mock Turtles游戏)**

**初始编号从0开始。**

当N==1时硬币为正，先手必胜，所以sg[0]=1

当N==2时硬币为反正，先手操作后可能为反反或正反，方案数为2，所以sg[1]=2.

当N==3时硬币为反反正，先手必胜，操作后可能为，反反反，反正反，正反正，正正反，方案数为4所以sg[2]=4

位置x:0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Sg[x]:1 2 4 7 8 11 13 14 16 19 21 22 25 26 28

看上去sg为2x或2x+1,我们称一个非负整数为odious,当且仅当该数的二进制形式的1出现次数是奇数，否则成为evil。所以1,2,4,7是odious因为他们的二进制为1,10,100,111而0,3,5,6是evil，因为他们的二进制形式是0,11,101,110.而上面那个表中，貌似sg值都是odious数，所以当2x为odious时，sg值为2x，否则为2x+1.

**约束条件7:每次可以连续翻动任意个硬币，至少翻一个。(Ruler游戏)**

初始编号从1开始。

那么这个游戏的SG函数是g(n)=mex{0,g(n-1,g(n-1)^g(n-2)^g(n-3),…,g(n-1)^…^g(1))}

根据SG函数可以得到SG值表如下。

位置x: 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Sg[x]: 1 2 1 4 1 2 1 8 1 2 1 4 1 2

所以sg值为x的因数当中2的能达到的最大次幂。比如14=2\*7，最大1次幂，即2；16=2\*2\*2\*2，最大4次幂，即16.

这个游戏成为尺子游戏是因为SG函数很像尺子上的刻度。

### 2. K倍动态减法游戏

//hdu2486

//规则和斐波那契博弈类似，但不同的是2倍变成了k倍

/\*

这就是K倍动态减法游戏，可以参考曹钦翔从“k倍动态减法游戏”出发探究一类组合游戏问题的论文。

首先k=1的时候，必败态是2^i,因为我们把数二进制分解后，拿掉最后一个1，那么会导致对方永远也取不完，我们可以拿到最后一个1.

k=2的时候，必败态是斐波那契数列，因为任何一个整数n都可以写成两项斐波那契数的和，所以我们拿掉1，对方永远取不完高两位的数。

k的时候我们必须构造数列，将n写成数列中一些项的和，使得这些被取到的项的相邻两个倍数差距>k 那么每次去掉最后一个1 还是符合上面的条件。设这个数列已经被构造了i 项，第 i 项为a[ i ]，前 i 项可以完美对1..b[ i ] 编码使得每个编码的任意两项倍数>K 那么有

a[ i+1 ] = b[ i ] + 1;这是显然的 因为b[ i ] + 1没法构造出来，只能新建一项表示

然后计算b[ i+1] 既然要使用 a[ i+1 ] 那么下一项最多只能是某个 a[ t ] 使得 a[ t ] \* K < a[ i+1 ] 于是

b[ i ] = b[ t ] + a[ i+1 ]

然后判断n是否在这个数列里面

如果在，那么先手必败。否则不停的减掉数列a中的项构造出n的分解，最后一位就是了

\*/

#include<bits/stdc++.h>

using namespace std;

const int maxn=2e6+5;

long long A[maxn],B[maxn];

int main()

{

int T;

scanf("%d",&T);

int cas=1;

while(T--)

{

long long n,k;

scanf("%lld %lld",&n,&k);

int i=0,j=0;

A[0]=1,B[0]=1;

while(A[i]<n)

{

i++;

A[i]=B[i-1]+1;

while(A[j+1]\*k<A[i])j++;

if(A[j]\*k<A[i])B[i]=A[i]+B[j];

else B[i]=A[i];

}

printf("Case %d: ",cas++);

if(A[i]==n)puts("lose");

else

{

int ans=-1;

for(int j=i;j>=0&&n;j--)

{

if(n>=A[j])

{

ans=j;

n-=A[j];

}

}

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

}

}

}

### 3. 阶梯博弈

//poj-1704

/\*从左到右有一排石子，给出石子所在的位置。规定每个石子只能向左移动，且不能跨过前面的石子。最左边的石子最多只能移动到1位置。每次选择一个石子按规则向左移动，问先手是否能赢。

我们把棋子按位置升序排列后，从后往前把他们两两绑定成一对。如果总个数是奇数，就把最前面一个和边界（位置为0）绑定。

在同一对棋子中，如果对手移动前一个，你总能对后一个移动相同的步数，所以一对棋子的前一个和前一对棋子的后一个之间有多少个空位置对最终的结果是没有影响的。

于是我们只需要考虑同一对的两个棋子之间有多少空位。

这样一来就成了N堆取石子游戏了.

\*/

#include<iostream>

#include<algorithm>

#include<cstring>

#include<stdio.h>

using namespace std;

const int maxn=1005;

int a[maxn];

int main()

{

int T;

scanf("%d",&T);

while(T--)

{

int n;

scanf("%d",&n);

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

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

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

int begin=2;

if(n%2)begin=1;

int ans=0;

a[0]=0;

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

{

ans^=(a[i]-a[i-1]-1);

}

if(ans==0)puts("Bob will win");

else puts("Georgia will win");

}

return 0;

}

### 4. \*NIM积

/\*

一个二维矩阵上，有若干个亮着的灯泡

每次选择一个矩阵（右上角的灯泡必须是亮的），改变四个角灯泡的状态,不能操作的人输

Hdu3404

\*/

#include<iostream>

#include<cstdio>

#include<cstring>

#define N 2000000

using namespace std;

int m[2][2]={0,0,0,1};

int Nim\_Mult\_Power(int x,int y){

if(x<2)

return m[x][y];

int a=0;

for(;;a++)

if(x>=(1<<(1<<a))&&x<(1<<(1<<(a+1))))

break;

int m=1<<(1<<a);

int p=x/m,s=y/m,t=y%m;

int d1=Nim\_Mult\_Power(p,s);

int d2=Nim\_Mult\_Power(p,t);

return (m\*(d1^d2))^Nim\_Mult\_Power(m/2,d1);

}

int Nim\_Mult(int x,int y){

if(x<y)

return Nim\_Mult(y,x);

if(x<2)

return m[x][y];

int a=0;

for(;;a++)

if(x>=(1<<(1<<a))&&x<(1<<(1<<(a+1))))

break;

int m=1<<(1<<a);

int p=x/m,q=x%m,s=y/m,t=y%m;

int c1=Nim\_Mult(p,s),c2=Nim\_Mult(p,t)^Nim\_Mult(q,s),c3=Nim\_Mult(q,t);

return (m\*(c1^c2))^c3^Nim\_Mult\_Power(m/2,c1);

}

int main(){

int t,n,x,y;

scanf("%d",&t);

while(t--){

scanf("%d",&n);

int ret=0;

while(n--){

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

ret^=Nim\_Mult(x,y);

}

if(ret)

puts("Have a try, lxhgww.");

else

puts("Don't waste your time.");

}

return 0;

}

### 5. 树上删边游戏

给定一颗n个点的有根树，每次可以shandiao 一个子树，则叶子节点的SG为0，非叶子节点的SG为他所有孩子节点(SG值+1)的异或和。

### 14、常用结论和方法

1.HDU 1297 Children’s Queue

题意：长度为N的序列 F要么不出现 要么不单独出现 即MFM FMM FMF 的情况。

思路：设f[i]为长度为i的满足条件的方案数，如果最后一个人是M，那么f[i]=f[i-1]。如果最后一个人F，那么倒数第二个也应该是F，这时去掉倒数第二个开始后面的F，如果第n-4和第n-3个分别是MF，那么f[i-2]是不满足条件的，还要加上f[i-4],如果第n-3个是M，那么f[i-2]就是满足条件的了。所以f[i]=f[i-1]+f[i-2]+f[i-4]。

1.

[HDU-6418] 两个人打牌，有剪刀、石头和布三种牌，赢一次得一分，输一次扣一分。给出两人手上剪刀石头布的数量a,b,c,a',b',c'，问在一个人随机出牌的情况下，另一个人的期望得分是多少。

思路:概率论中有一个经典的问题，类似商场里面的抽奖，不论是第几个抽奖，拿到奖项的概率都是一样的。这题中，因为每一张牌都要出，所以前面的得分会使得后面的牌带来一定的劣势，扣分同理，可以推导出同样的牌，先后出得分的期望概率是一样的。所以计算初始情况下的每种牌的期望得分，再乘以这种牌的数量，就是最后的得分期望值。例如出剪刀的期望得分就是a'\*(c-b)/(a+b+c)，所以结果就是a'\*(c-b)/(a+b+c)+b'\*(a-c)/(a+b+c)+c'\*(b-a)/(a+b+c)

2.

[CodeForces-1099B]在一个无限大的网格上画正方形，只能画单位1长度的线段，且端点只能在整数格点上。你如果想画(x,y)到(x,y+1)这样一条线段，如果存在(x′,y)到(x′,y+1)的另外一条线段，则可以直接画，否则就需要用尺子。对于(x,y) 到(x+1,y)这样的线段也是类似的。求最少用几次尺子能画出n个正方形。

思路:当n=k^2,ans=2\*k,否则找构成大正方形的最长的边,即k=ceil(sqrt(n))，然后计算剩余有多少个位置放小正方形使得最长边还是k-1，即t=n-(k-1)\*(k-1)，如果t<k,也就是剩余的小正方形可以让最长边仍为k，则答案为2\*k-1,否则答案为2\*k。

3.

[NYOJ-417]在1~n个数中，随机取m个数,问在这m个数中是否一定存在一个数是另一个数的倍数，是则回答“YES",否则”NO"。

思路:1到n所有数的倍数分组数为n/2向上取整，如果m>ceil(n\*1.0/2)则输出YES。

4.

[洛谷p1007]每个士兵都有一个初始面对的方向，他们会以匀速朝着这个方向行走，中途不会自己改变方向。但是，如果两个士兵面对面相遇，他们无法彼此通过对方，于是就分别转身，继续行走。转身不需要任何的时间。求最少和最多需要多少时间就可能全部撤离独木桥。

思路:两个人相遇转身，相当于"交换灵魂"后继续走。mx=max(mx,max(p,l-p+1));mi=max(mi,min(p,l-p+1));

5.

小B准备出模拟赛。

她把题目按难度分为四等，分值分别为6,7,8,9。

已知小B共出了m道题，共n分。

求小B最少出了多少道6分题。

思路:有解的充要条件是6m<=n<=9m,设6分有x个,则7(m-x)<=n-6x<=9(m-x),解得7m-n<=x<=(9m-n)/3,所以答案是max(0,7m-n)。

6.

约瑟夫O（n）:

n个人(编号1~n)，报数为m的被淘汰，求最后剩下的人。

#include<stdio.h>

int main()

{

int n, m,i,s=0;

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

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

s=(s+m)%i;

printf("%d", s+1);

return 0;

}

7.

一张地图上有有N个城市，他们可以通过双向道路互相连接，但是每两座城市只能有一条双向道路互相连接。

现在我们想要满足条件“地图中不能有任意三个城市可以互相直达”，请问满足这个条件的最大道路数是多少？

思路：把n个点分成2组，组内不连，组间连，所以有n/2(n-n/2)种。

1.有n个学生打羽毛球，其中有a个人没拍没球，b个人只有拍，c个人只有球，d个人有拍有球。现在要求选出几个人组织一场比赛，一共有2^n种情况，组织成功的条件是至少有2个拍1个球。求有多少种情况无法组织成功。

思路:分类考虑。第一类，abcd都不选，即0000，有一种情况。第二类，选1种，abc任选多少都可以(不能一个都不选)，d只能选一个,所以有C(a,1)+…C(a,a)+C(b,1)+…C(b,b)+C(c,1)+…C(c,c)+C(d,1)=2^a-1+2^b-1+2^c-1+d。第三类，选2种，ab、ac可以任取，ad中d只能取1，bc中b只能取1，bd不存在，cd中d只能取1，所以有(2^a-1)\*(2^b-1)+(2^a-1)\*(2^c-1)+(2^a-1)\*d+b\*(2^c-1)+(2^c-1)\*d。第四类，选3种,abc中b只能取1，acd中d只能取1，bcd不存在，有(2^a-1)\*b\*(2^c-1)+(2^a-1)\*(2^c-1)\*d。显然，四种全选也是不可能的。所以最终结果是1+2^a-1+2^b-1+2^c-1+d+(2^a-1)\*(2^b-1)+(2^a-1)\*(2^c-1)+(2^a-1)\*d+b\*(2^c-1)+(2^a-1)\*b\*(2^c-1)+(2^a-1)\*(2^c-1)\*d。

## 刷题总结

### 期望与概率

### **走迷宫**

思路：设花费时间 出迷宫的期望为E。

每一个选择仅仅有两种情况——设当前门花费时间的绝对值为 T

一：选择的门能够直接把你传送出去。期望为1 / N \* T。

二：选择的门把你传送到原来的位置，期望为1 / N \* T。又回到初始状态，则出去的期望为1 / N \* (T + E)。

设全部能够将你传送出去的门的时间值 总和为sum1，全部能够将你传送回去的门的时间值 总和为sum2。

设全部能够将你传送出去的门的数目为door1，全部能够将你传送回去的门的数目为door2。

得到等式

E = 1 / N \* (sum1)  + 1 / N \* (sum2 + door2 \* E)。

化简得 E = (sum1 + sum2) / (N-door2); 当然若door2等于N。说明不可能出迷宫。

#include <iostream>

#include<bits/stdc++.h>

using namespace std;

#define ll long long

ll gcd(ll a,ll b)

{

return b==0?a:gcd(b,a%b);

}

int a[105];

int main()

{

ll n,door2,sum;

scanf("%d",&n);

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

{

door2=0;

sum=0;

memset(a,0,sizeof(a));

//cout<<endl;

int m;

scanf("%d",&m);

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

{

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

if(a[i]<0)

{

door2++;

sum=sum-a[i];

}

else

sum+=a[i];

}

if(door2==m)

cout<<"Case "<<i+1<<": inf"<<endl;

else

{

int temp=m-door2;

int ans=gcd(sum,temp);

cout<<"Case "<<i+1<<": "<<sum/ans<<"/"<<temp/ans<<endl;

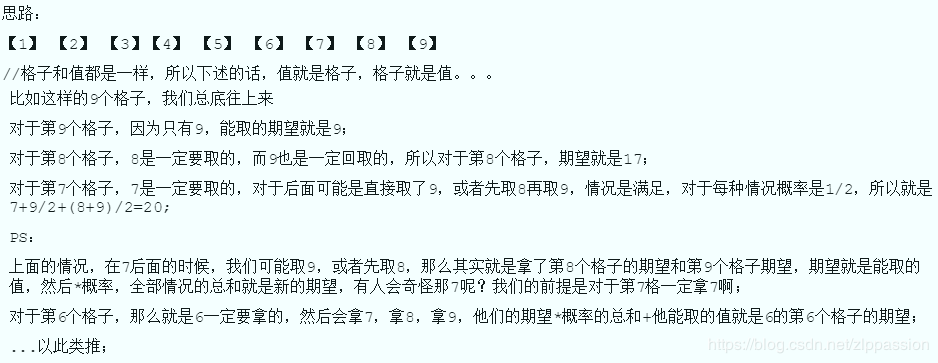
}

}

return 0;

}

### 摇色子，走出通道



#include <iostream>

#include<bits/stdc++.h>

using namespace std;

double dp[105];

int main()

{

int T;

scanf("%d",&T);

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

{

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

int m;

scanf("%d",&m);

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

{

scanf("%lf",&dp[j]);

}

for(int j=m-2; j>=0; j--)

{

int t=6;

int k=m-1-j;

if(k<t)

t=k;

for(int z=1; z<=t; z++)

{

dp[j]+=dp[j+z]\*1.0/t;

}

}

printf("Case %d: %lf\n",i,dp[0]);

}

return 0;

}

### **Race to 1 Again**

题意：给你一个n，n可以变一次变成他的某一个约数，然后在变成他的某个约数，直到变成1，求n变为1的期望次数？

const int N=1e5+5;

double p[N];

void init(){

memset(p,0,sizeof(p));

for(int i=2;i<=N;i++){

double sum=0;

int cnt=0;

p[1]=0;

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

if(i%j==0){

cnt++;

sum+=(p[j]+1); //+1表示除以因子算走一步了

if(j!=i/j){ //与另一个因子不同

cnt++;

sum+=(p[i/j]+1);

}

}

}

p[i]=sum/(cnt-1);

}

}

int main(){

init();

int T,t=1;scanf("%d",&T);

while(T--){

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

printf("Case %d: %lf\n",t++,p[n]);

}

return 0;

}

### 2、欧拉函数

计算1-N区间里有多少数和N的GCD是大于M的。

先看两个数  N = a\*b，X= a\*d。因为gcd ( N , X ) = a  所以b,d这两个数互质。又因为d可以是任何一个小于b的数。所以d值数量的的多少就是b的欧拉函数值。

所以，我们可以枚举a，然后去求b，然后再求b的欧拉函数值。

#include<bits/stdc++.h>

using namespace std;

int euler(int n)

{

int res=n;

for(int i=2; i\*i<=n; i++)

{

if(n%i==0)

{

res=res/i\*(i-1);

while(n%i==0)

n/=i;

}

}

if(n>1)

res-=res/n;

return res;

}

int main()

{

int t;

scanf("%d",&t);

while(t--)

{

int n,m;

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

int ans=0;

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

{

if(n%i==0)

{

if(i>=m)

ans+=euler(n/i); //计算sqrt(n)左边的

if(n/i>=m&&i\*i!=n)

ans+=euler(i);//计算sqrt(n)右边的i\*i==n时，在上个语句已经执行

}

}

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

}

return 0;

}

### **3、快速幂**

m/n,求小数点后K1到K2位数字，若不足则用 0 补齐

#include<bits/stdc++.h>

using namespace std;

#define ll long long

const int inf=0x3f3f3f3f;

const ll mod=1e9+7;

ll m,n,k1,k2;

ll q\_pow(ll a,ll n,ll mod){

ll ans=m; ll t=a;

while(n){

if(n&1) ans=(ans\*t)%mod;

t=t\*t%mod;

n>>=1;

}

return ans;

}

int main(){

int t;

scanf("%d",&t);

while(t--){

scanf("%lld%lld%lld%lld",&m,&n,&k1,&k2);

ll f=q\_pow(10,k1-1,n);

cout<<f<<endl;

ll i=k1;

while(1){

f\*=10;

printf("%lld",f/n);

f%=n;

i++;

if(i==k2+1)

break;

}

printf("\n");

}

}

### 唯一分解定理

口算训练：每个问题给出三个正整数*l*,*r*,*d*，小Q需要通过口算快速判断*al*×*al*+1×...×*ar*−1×*ar*是不是*d*的倍数。

这题正解是要对每一个数分解质因数，由唯一分解定理可知，任何一个数可拆成唯一的素数的幂之积。举个例子，24=2^3\*3，6=2\*3，因为24的2和3的个数都能凑出6=2\*3来，所以24是6的倍数。预处理用二维向量，一维表示因子，另一维表示能拆成此因子的数的下标。查询的时候看d的每个因子所在的区间数量是否能凑够，upper\_bound找第一个大于r的位置，lower\_bound找第一个大于等于l的位置，相减即为此区间d的这个因子的个数

#include<bits/stdc++.h>

using namespace std;

#define inf 0x3f3f3f3f

#define ll long long

const int N=1e5+5;

const int mod=1e9+7;

const double eps=1e-8;

const double PI = acos(-1.0);

#define lowbit(x) (x&(-x))

vector<int> g[N];

int main()

{

int t;

scanf("%d",&t);

while(t--)

{

int n,m;

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

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

g[i].clear();

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

{

int x;

scanf("%d",&x);

int sx=sqrt(x);

for(int j=2;j<=sx; j++)

{

while(x!=1&&x%j==0)

{

g[j].push\_back(i);//i是下标，j是因子

x/=j;

}

if(x==1)

break;

}

if(x>1)//eg:x=7,sx=2,

{

g[x].push\_back(i);//切记i是下标

}

}

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

{

int l,r,d;

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

int flag=0;

for(int j=2;j\*j<=d;j++)

{

int cnt=0;

while(d!=1&&d%j==0)

{

cnt++;

d/=j;

}

if(cnt>upper\_bound(g[j].begin(),g[j].end(),r)-lower\_bound(g[j].begin(),g[j].end(),l))

//这样相减就是l到r之间j因子的个数，只有大于了cnt，才会满足要求，否则就不满足。

{

flag=1;

break;

}

if(d==1)

break;

}

if(d>1&&!flag&&!(upper\_bound(g[d].begin(),g[d].end(),r)-lower\_bound(g[d].begin(),g[d].end(),l)))

{

flag=1;

}

if(!flag)

puts("Yes");

else

puts("No");

}

}

return 0;

}

### 欧几里德和同余方程

公青蛙一开始在x位置，母青蛙在y位置。公青蛙每次跳m米，母青蛙每次跳n米，并且都是向右跳的。地球经线长度是L，然后地球是圆的，也就是说，跳到L、L+1、L+2……其实就是跳到0、1、2。 公青蛙想追母青蛙，问多少次后它们能跳到一起。如果它们永远不能相遇，就输出Impossible（好可怜啊！）

很明显嘛，就是求一个k，使x + k\*m ≡ y + k\*n (mod L)---》(n-m) \* k ≡ x-y (mod L)-----》(n-m)\*k + L\*s = x-y。这就是ax + by = c求整数x的模型。

#include<cstdio>

#include<iostream>

#include<cstring>

#include<algorithm>

using namespace std;

int exgcd(long long a,long long b,long long& x,long long& y)

{

int d=a;

if(b!=0) {

d=exgcd(b,a%b,y,x);

y-=(a/b)\*x;

}

else{

x=1; y=0;

}

return d;

}

int main()

{

long long x,y,n,m,l;

while(~scanf("%lld%lld%lld%lld%lld",&x,&y,&m,&n,&l)){

long long a=m-n,b=l,c=y-x;

if(a<0) {

a=-a; c=-c;

}

long long X,Y;

long long d=exgcd(a,b,X,Y);

if(c%d!=0) {

printf("Impossible\n");

}

else{

long long s=l/d;

printf("%lld\n",((X\*c/d)%s+s)%s);

}

}

}

# 高精度

## 经典题型

### java版线性筛

/\*

题意：求一个大数k（保证是两个素数乘积）的较小素数是否小于L。

POJ2635

\*/

import java.math.\*;

import java.util.\*;

public class Main {

final int maxn=(int)1e6+10;

static int tot=0;

int [] prime=new int[maxn],vis=new int [maxn];

public void shai() {

tot=0;

Arrays.fill(vis, 0);

for(int i=2;i<maxn;i++)

{

if(vis[i]==0)

{

prime[tot++]=i;

}

for(int j=0;j<tot&&prime[j]\*i<maxn;j++)

{

vis[prime[j]\*i]=1;

if(i%prime[j]==0)

{

break;

}

}

}

}

public static void main(String[] args) {

BigInteger k,L;

Main M=new Main();

M.shai();

Scanner cin=new Scanner(System.in);

while(cin.hasNext())

{

k=cin.nextBigInteger();

L=cin.nextBigInteger();

BigInteger zero=BigInteger.ZERO;

int i,flag=0;

if(k.compareTo(zero)==0&&L.compareTo(zero)==0)

{

break;

}

// System.out.println(M.tot);

for(i=0;i<M.tot;i++)

{

if(k.mod(BigInteger.valueOf(M.prime[i])).compareTo(zero)==0)

{

if(BigInteger.valueOf(M.prime[i]).compareTo(L)<0)

{

System.out.println("BAD "+M.prime[i]);

flag=1;

break;

}

}

}

if(flag==0)

{

System.out.println("GOOD");

}

}

}

}

### 大数任意进制转换

#include<bits/stdc++.h>

using namespace std;

const int MAXN = 1000;

int t[MAXN], A[MAXN];

char OldData[MAXN], NewData[MAXN]; // 转换前、后的数据

int olds, news; // 转换前、后的进制

// 调用方式：输入olds、news、OldData，然后调用trans(),输出NewData

void trans()

{

int i, len, k;

len = strlen(OldData);

for(i=len; i>=0; --i)

t[len-1-i] = OldData[i] - (OldData[i]<58 ? 48 : OldData[i]<97 ? 55 : 61);

for(k=0; len;)

{

for(i=len; i>=1; --i)

{

t[i-1] += t[i]%news\*olds;

t[i] /= news;

}

A[k++] = t[0] % news;

t[0] /= news;

while(len>0 && !t[len-1]) --len;

}

NewData[k] = NULL;

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

NewData[k-1-i] = A[i] + (A[i]<10 ? 48 : A[i]<36 ? 55 : 61);

}

int main()

{

cin>>OldData>>olds>>news;

trans();

cout<<NewData<<endl;

}

### 高精度开根号

import java.math.BigInteger;

import java.util.Scanner;

public class Main{

public static void main(String args[]) {

Scanner cin=new Scanner(System.in);

while(cin.hasNext())

{

String s=cin.next();

System.out.println(bigSqrt(s));

}

}

public static BigInteger bigSqrt(String s) {

BigInteger remain = BigInteger.ZERO;

BigInteger odd = BigInteger.ZERO;

BigInteger ans = BigInteger.ZERO;

int group = 0, k = 0;

if (s.length() % 2 == 1) {

group = s.charAt(0) - '0';

k = -1;

} else {

group = (s.charAt(0) - '0') \* 10 + s.charAt(1) - '0';

k = 0;

}

for (int j = 0; j < (s.length() + 1) / 2; j++) {

if (j != 0)

group = (s.charAt(j \* 2 + k) - '0') \* 10 + s.charAt(j \* 2 + k + 1) - '0';

odd = BigInteger.valueOf(20).multiply(ans).add(BigInteger.ONE);

remain = BigInteger.valueOf(100).multiply(remain).add(BigInteger.valueOf(group));

int count = 0;

while (remain.compareTo(odd) >= 0) {

count++;

remain = remain.subtract(odd);

odd = odd.add(BigInteger.valueOf(2));

}

ans = ans.multiply(BigInteger.TEN).add(BigInteger.valueOf(count));

}

return ans;

}

}

### 除法保留小数

1.除法保留小数

例如2.35

1.BigDecimal.ROUND\_DOWN 2.3

2.BigDecimal.ROUND\_UP 2.4

3.BigDecimal.ROUND\_HALF\_UP 2.4

4.BigDecimal.ROUND\_HALF\_DOWN 2.3 (3和4都是四舍五入,只有0.5时与ROUND\_HALF\_UP不同）

import java.math.\*;

import java.util.\*;

public class Main {

public static void main(String[] args) {

int t;

Scanner cin=new Scanner (System.in);

while(cin.hasNext())

{

BigDecimal a,b;

int c;

String s;

a=cin.nextBigDecimal();

b=cin.nextBigDecimal();

c=cin.nextInt();

s=cin.next();

if(s.compareTo("Xiang")==0)

{

System.out.println(a.divide(b,c,BigDecimal.ROUND\_FLOOR));//c是保留小数几位

}

else

System.out.println(a.divide(b,c,BigDecimal.ROUND\_HALF\_UP));

}

}

}

### 5、高精度小数加法(非科学计数法+去尾0+java)

package bisai;

//toPlainString:不使用任何指数,toString:有必要时使用科学计数法,stripTrailingZeros():移除尾0

import java.math.BigDecimal;

import java.util.Scanner;

public class Main{

public static void main(String args[]) {

Scanner cin=new Scanner(System.in);

while(cin.hasNext())

{

BigDecimal a=cin.nextBigDecimal(),b=cin.nextBigDecimal();

//System.out.println(a.add(b).stripTrailingZeros().toPlainString());

String s = a.add(b).stripTrailingZeros().toPlainString();

if (s.startsWith("0."))//去除小数点后面的0；

System.out.println(s.substring(1));

}

}

}

/\*

去除小数点前面的0

String s = ans.stripTrailingZeros().toPlainString();

if (s.startsWith("0."))

System.out.println(s.substring(1));

\*/

# 数据结构

## 线段树

### 线段树模板递归版

//HDU 1166模板题

//线段树详解：https://blog.csdn.net/zearot/article/details/48299459

#include<bits/stdc++.h>

using namespace std;

#define inf 0x3f3f3f3f

#define ll long long

const int maxn=50005;

const double eps=1e-8;

const double PI = acos(-1.0);

#define lowbit(x) (x&(-x))

int Sum[maxn<<2],Add[maxn<<2];//Sum求和，Add为懒惰标记

int A[maxn],n;//存原数组数据下标[1,n]

//PushUp函数更新节点信息 ，这里是求和

void PushUp(int rt)

{

Sum[rt]=Sum[rt<<1]+Sum[rt<<1|1];

}

//Build函数建树

void Build(int l,int r,int rt) //l,r表示当前节点区间，rt表示当前节点编号

{

if(l==r) //若到达叶节点

{

Sum[rt]=A[l];//储存数组值

return;

}

int m=(l+r)>>1;

//左右递归

Build(l,m,rt<<1);

Build(m+1,r,rt<<1|1);

//更新信息

PushUp(rt);

}

//点修改 假设A[L]+=C:

void Update(int L,int C,int l,int r,int rt) //l,r表示当前节点区间，rt表示当前节点编号

{

if(l==r) //到叶节点，修改

{

Sum[rt]+=C;

return;

}

int m=(l+r)>>1;

//根据条件判断往左子树调用还是往右

if(L <= m) Update(L,C,l,m,rt<<1);

else Update(L,C,m+1,r,rt<<1|1);

PushUp(rt);//子节点更新了，所以本节点也需要更新信息

}

//下推标记

void PushDown(int rt,int ln,int rn)

{

//ln,rn为左子树，右子树的数字数量。

if(Add[rt])

{

//下推标记

Add[rt<<1]+=Add[rt];

Add[rt<<1|1]+=Add[rt];

//修改子节点的Sum使之与对应的Add相对应

Sum[rt<<1]+=Add[rt]\*ln;

Sum[rt<<1|1]+=Add[rt]\*rn;

//清除本节点标记

Add[rt]=0;

}

}

//区间修改 假设A[L,R]+=C

void Update(int L,int R,int C,int l,int r,int rt) //L,R表示操作区间，l,r表示当前节点区间，rt表示当前节点编号

{

if(L <= l && r <= R) //如果本区间完全在操作区间[L,R]以内

{

Sum[rt]+=C\*(r-l+1);//更新数字和，向上保持正确

Add[rt]+=C;//增加Add标记，表示本区间的Sum正确，子区间的Sum仍需要根据Add的值来调整

return ;

}

int m=(l+r)>>1;

PushDown(rt,m-l+1,r-m);//下推标记

//这里判断左右子树跟[L,R]有无交集，有交集才递归

if(L <= m) Update(L,R,C,l,m,rt<<1);

if(R > m) Update(L,R,C,m+1,r,rt<<1|1);

PushUp(rt);//更新本节点信息

}

//区间查询

int Query(int L,int R,int l,int r,int rt) //L,R表示操作区间，l,r表示当前节点区间，rt表示当前节点编号

{

if(L <= l && r <= R)

{

//在区间内，直接返回

return Sum[rt];

}

int m=(l+r)>>1;

//下推标记，否则Sum可能不正确

PushDown(rt,m-l+1,r-m);

//累计答案

int ANS=0;

if(L <= m) ANS+=Query(L,R,l,m,rt<<1);

if(R > m) ANS+=Query(L,R,m+1,r,rt<<1|1);

return ANS;

}

/\*

//建树

Build(1,n,1);

//点修改

Update(L,C,1,n,1);

//区间修改

Update(L,R,C,1,n,1);

//区间查询

int ANS=Query(L,R,1,n,1);

\*/

int main()

{

std::ios::sync\_with\_stdio(false);

std::cin.tie(0);

std::cout.tie(0);

int t,k=0;

cin>>t;

while(t--)

{

memset(Add,0,sizeof(Add));

int n;

cin>>n;

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

{

cin>>A[i];

}

Build(1,n,1);

string s;

int a,b;

cout<<"Case "<<++k<<":"<<endl;

while(cin>>s&&s[0]!='E')

{

cin>>a>>b;

if(s[0]=='Q')

{

cout<<Query(a,b,1,n,1)<<endl;

}

else if(s[0]=='A')

{

Update(a,b,1,n,1);

}

else if(s[0]=='S')

{

Update(a,-b,1,n,1);

}

}

}

return 0;

}

### 线段树求最小逆序数

/\*1.hdu1394:给一个0-n-1的排列，这个排列中的逆序数为数对 (ai, aj) 满足 i < j and ai > aj的个数。依次把第一个数放到排列的末尾会得到另外n-1个排列，求这n个排列中的最小的逆序数。

思路：每次把首位移到末尾，逆序数变化是大于a[0]的个数（(n-2)-(a[0]-1)）-小于a[0]的个数（a[0]），即sum=sum+(n-2)-(a[0]-1)-a[0]。创建线段树的时候就是按下标顺序插入的，所以a[i]产生的逆序数=在a[i]之前插入并且大于a[i]的数，查询a[i]~n-1的Sum即可。然后更新a[i]所在的区间表示这个数已经出现过。

\*/

/\*the next line contains a permutation of the n integers from 0 to n-1. \*/

//该序列最大值为n-1,最小值为0

#include<bits/stdc++.h>

using namespace std;

#define ll long long

const int maxn=50005;

int Sum[maxn<<2],Add[maxn<<2];//Sum求和，Add为懒惰标记

int A[maxn],n;//存原数组数据下标[1,n]

void PushUp(int rt)

{

Sum[rt]=Sum[rt<<1]+Sum[rt<<1|1];

}

void Build(int l,int r,int rt) //l,r表示当前节点区间，rt表示当前节点编号

{

if(l==r) //若到达叶节点

{

Sum[rt]=0;//储存数组值

return;

}

int m=(l+r)>>1;

//左右递归

Build(l,m,rt<<1);

Build(m+1,r,rt<<1|1);

//更新信息

PushUp(rt);

}

void Update(int L,int l,int r,int rt) //l,r表示当前节点区间，rt表示当前节点编号

{

if(l==r) //到叶节点，修改

{

Sum[rt]=1;//代表已经插入

return;

}

int m=(l+r)>>1;

//根据条件判断往左子树调用还是往右

if(L <= m) Update(L,l,m,rt<<1);

else Update(L,m+1,r,rt<<1|1);

PushUp(rt);//子节点更新了，所以本节点也需要更新信息

}

void PushDown(int rt,int ln,int rn)

{

//ln,rn为左子树，右子树的数字数量。

if(Add[rt])

{

//下推标记

Add[rt<<1]+=Add[rt];

Add[rt<<1|1]+=Add[rt];

//修改子节点的Sum使之与对应的Add相对应

Sum[rt<<1]+=Add[rt]\*ln;

Sum[rt<<1|1]+=Add[rt]\*rn;

//清除本节点标记

Add[rt]=0;

}

}

int Query(int L,int R,int l,int r,int rt) //L,R表示操作区间，l,r表示当前节点区间，rt表示当前节点编号

{

if(L <= l && r <= R)

{

//在区间内，直接返回

return Sum[rt];

}

int m=(l+r)>>1;

//下推标记，否则Sum可能不正确

PushDown(rt,m-l+1,r-m);

//累计答案

int ANS=0;

if(L <= m) ANS+=Query(L,R,l,m,rt<<1);

if(R > m) ANS+=Query(L,R,m+1,r,rt<<1|1);

return ANS;

}

int main()

{

int t,k=0;

int n;

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

{

memset(Add,0,sizeof(Add));

memset(Sum,0,sizeof(Sum));

Build(0,n-1,1);

int sum=0;

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

{

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

//第一个n-1参数代表最大值，A[i]当前数，因为逆序数是看在当前数位置前的数比它大的有多少个

//后面三个参数是不动的

sum+=Query(A[i],n-1,0,n-1,1);

Update(A[i],0,n-1,1);//把当前节点赋为1，代表已经插入

}

int mi=sum;

for(int i=0;i<n-1;i++)

{

sum=sum+n-A[i]\*2-1;

mi=min(mi,sum);

}

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

}

return 0;

}

### 模板改编题

线段树功能:update:单点替换 query:区间最值

在使用%c时一定要注意，当出问题时，可以通过在%c前加空格，或加getchar(0,或在之后加空格等。都试试就好，每个题不一样。虽然题虽然也类似模板题，但也要自己改一下函数，这应该是比赛常考的吧

对于每一次询问操作，在一行里面输出最高成绩。

Sample Input

5 6

1 2 3 4 5

Q 1 5

U 3 6

Q 3 4

Q 4 5

U 2 9

Q 1 5

Sample Output

5

6

5

9\*/

#include<bits/stdc++.h>

using namespace std;

const int maxn=2e5+5; //元素总个数

#define ls l,m,rt<<1

#define rs m+1,r,rt<<1|1

//int Sum[maxn<<2],Add[maxn<<2];//Sum求和，Add为懒惰标记

int MAX[maxn<<2];

int A[maxn],n;//存原数组数据下标[1,n]

//(1)建树：

//PushUp函数更新节点信息 ，这里是求和

void PushUp(int rt)

{

//Sum[rt]=Sum[rt<<1]+Sum[rt<<1|1];

//由于这道题不是求和，是求最大值，所以pushup也需要更改

MAX[rt]=max(MAX[rt<<1],MAX[rt<<1|1]);

}

//Build函数建树

void Build(int l,int r,int rt) //l,r表示当前节点区间，rt表示当前节点编号

{

if(l==r) //若到达叶节点

{

MAX[rt]=A[l];//储存数组值

// scanf("%d",&MAX[rt]);//这一步是在干啥啊，。，

return;

}

int m=(l+r)>>1;

//左右递归

Build(l,m,rt<<1);

Build(m+1,r,rt<<1|1);

//更新信息

PushUp(rt);

}

//(2)点修改：

void Update(int L,int C,int l,int r,int rt) //l,r表示当前节点区间，rt表示当前节点编号

{

if(l==r) //到叶节点，修改

{

MAX[rt]=C;

return;

}

int m=(l+r)>>1;

//根据条件判断往左子树调用还是往右

if(L <= m)

Update(L,C,l,m,rt<<1);

else

Update(L,C,m+1,r,rt<<1|1);

PushUp(rt);//子节点更新了，所以本节点也需要更新信息

}

int query(int L,int R,int l,int r,int rt)//L,R表示操作区间，l,r表示当前节点区间，rt表示当前节点编号

{

if(L<=l&&r<=R)

{

return MAX[rt];

}

int m=(l+r)>>1;

int ret=0;

if(L<=m)

ret=max(ret,query(L,R,l,m,rt<<1));

if(R>m)

ret=max(ret,query(L,R,m+1,r,rt<<1|1));

return ret;

}

int main()

{

int N,M;

while(scanf("%d%d",&N,&M)!=EOF)

{

memset(A,0,sizeof(A));

memset(MAX,0,sizeof(MAX));

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

{

scanf("%d",&A[p]);

}

Build(1,N,1);

char c;

int a,b;

for(int t=1;t<=M;t++)

{

scanf(" %c%d%d",&c,&a,&b);//当用%c的时候需要注意一下，上面有没有换行符等

// cin>>c>>a>>b;用这个也对

if(c=='Q')

{

long long ANS=query(a,b,1,N,1);

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

}

else if(c=='U')

{

Update(a,b,1,N,1);

}

}

}

return 0;

}

### 线段树——离散化+区间覆盖

/\*

题意：n（n<=10000)?个人依次贴海报,给出每张海报所贴的范围li，ri（1<=li<=ri<=10000000)?。求出最后还能看见多少张海报。

思路：由于数据范围比较大，用数组存不下，所以可以离散化成n的范围，而这题的离散化也比较特殊，因为查询的时候是一个一个点向右查找的，如果遇到1 10,1 4,6 10这种数据，那么会被离散化成1 4,1 2,3 4，这样计算的时候就只有两种颜色了，实际上有三种颜色，所以可以在离散化后每两个距离大于1的点之间再加一个点，比如小的+1。初始化直接memset(sum,-1,sizeof(sum))而不用再build，因为涂色是从0开始。update时找到左右区间直接赋值即可。因为要求海报的种数，所以用一个vis数组来存某个颜色是否计算过，没有被计算过而且有颜色答案就++（注意这里不用查到l==r，因为只要有区间有这种颜色就行了）。l==r的时候要return，不然递归会卡死。

\*/

#include<algorithm>

#include<cstdio>

#include<cstring>

using namespace std;

const int maxn=10005;

int vis[maxn<<3],sum[maxn<<4];

int li[maxn<<1],ri[maxn<<1],lsh[maxn<<2];

void pushDown(int rt)

{

if(sum[rt]!=-1)

{

sum[rt<<1]=sum[rt<<1|1]=sum[rt];

sum[rt]=-1;

}

}

void update(int L,int R,int C,int l,int r,int rt)

{

if(L<=l&&r<=R)

{

sum[rt]=C;

return ;

}

pushDown(rt);

int m=(l+r)>>1;

if(m>=L)

{

update(L,R,C,l,m,rt<<1);

}

if(m<R)

{

update(L,R,C,m+1,r,rt<<1|1);

}

}

int ans;

void query(int l,int r,int rt)

{

if(!vis[sum[rt]]&&sum[rt]!=-1)

{

ans++;

vis[sum[rt]]=1;

return ;

}

if(l==r)

return;

pushDown(rt);

int m=(l+r)>>1;

query(l,m,rt<<1);

query(m+1,r,rt<<1|1);

}

int main()

{

int t,n;

scanf("%d",&t);

while(t--)

{

scanf("%d",&n);

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

memset(vis,0,sizeof(vis));

int tot=0;

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

{

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

lsh[tot++]=li[i];

lsh[tot++]=ri[i];

}

sort(lsh,lsh+tot);

int mm=unique(lsh,lsh+tot)-lsh;

int tt=mm;

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

{

if(lsh[i]-lsh[i-1]>1)

{

lsh[mm++]=lsh[i-1]+1;

}

}

sort(lsh,lsh+mm);

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

{

int x=lower\_bound(lsh,lsh+mm,li[i])-lsh+1;

int y=lower\_bound(lsh,lsh+mm,ri[i])-lsh+1;

update(x,y,i,1,mm,1);

}

ans=0;

query(1,mm,1);

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

}

}

### 线段树——区间合并(维护连续零)

#include<bits/stdc++.h>

using namespace std;

#define inf 0x3f3f3f3f

#define ll long long

const int maxn=50005;

const int mod=1e9+7;

const double eps=1e-8;

const double PI = acos(-1.0);

#define lowbit(x) (x&(-x))

int len[maxn<<2],llen[maxn<<2],rlen[maxn<<2];

void pushUp(int rt,int length)

{

llen[rt]=llen[rt<<1]; //当前结点的最长连续前缀=左子结点的最长连续前缀

rlen[rt]=rlen[rt<<1|1];

if(llen[rt<<1]==(length-(length>>1))) llen[rt]+=llen[rt<<1|1];

//如果左子结点的最长连续前缀为整个左子区间，那么本结点的前缀还要加上右子区间的最长前缀

if(rlen[rt<<1|1]==(length>>1)) rlen[rt]+=rlen[rt<<1];

len[rt]=max(max(len[rt<<1],len[rt<<1|1]),rlen[rt<<1]+llen[rt<<1|1]);

}

void build(int l,int r,int rt)

{

if(l==r)

{

len[rt]=llen[rt]=rlen[rt]=1;

return ;

}

int m=(l+r)>>1;

build(l,m,rt<<1);

build(m+1,r,rt<<1|1);

pushUp(rt,r-l+1);

}

void update(int loc,int C,int l,int r,int rt)

{

if(l==r)

{

len[rt]=llen[rt]=rlen[rt]=C;

return ;

}

int m=(l+r)>>1;

if(loc<=m) update(loc,C,l,m,rt<<1);

if(loc>m) update(loc,C,m+1,r,rt<<1|1);

pushUp(rt,r-l+1);

}

int query(int loc,int l,int r,int rt)

{

if(l==r) return len[rt];

int m=(l+r)>>1;

if(loc<=m)

{

if(loc+rlen[rt<<1]>m) return rlen[rt<<1]+llen[rt<<1|1];

//如果loc在左子区间的最长后缀和右子区间的最长前缀中，直接输出这两个前后缀之和即可

else return query(loc,l,m,rt<<1);

//否则的话，继续向左子节点查询

}

else

{

if(m+llen[rt<<1|1]>=loc) return rlen[rt<<1]+llen[rt<<1|1];

//因为右子区间是(mid,r]左开区间，所以这里判断loc是否在右子区间的前缀的范围内用的是 ">="

else return query(loc,m+1,r,rt<<1|1);

}

}

int main()

{

std::ios::sync\_with\_stdio(false);

int n,m;

while(cin>>n>>m)

{

build(1,n,1);

int tot=0,stk[maxn];

while(m--)

{

char c;

cin>>c;

if(c=='D')

{

int loc;

cin>>loc;

stk[++tot]=loc;

update(loc,0,1,n,1);

}

else if(c=='R')

{

int loc=stk[tot--];

update(loc,1,1,n,1);

}

else

{

int loc;

cin>>loc;

cout<<query(loc,1,n,1)<<endl;

}

}

}

return 0;

}

### 6、线段树——区间染色+区间统计

//ZOJ 1610

/\*

在一条长度为8000的线段上染色，每次把区间[a,b]染成c颜色。显然，后面染上去的颜色会覆盖掉之前的颜色。

求染完之后，每个颜色在线段上有多少个间断的区间。

\*/

/\*

此题有个坑点，就是染色是染区间，而不是染点，什么意思呢？举个栗子，比如1 2 1，3 4 1（[1,2]和[3,4]被染成1），最后查找的时候是按左到右叶子查找的，所以会把这两个区间当成连续的，实质上是间隔的（[2,3]没有被染色），解决方法就是update的时候让x+1或y-1（y-1的时候要保证update的l参数是0而不是1，比如0 1?1，如果不从0开始找那么会忽略这个被染色的区间），这样就能保证区间是不连续的（左开右闭或左闭右开），最后统计区间得个数时用一个last变量保存上一个区间点的颜色，如果不一样就更新。还有就是col初始化为-1是因为有0这种颜色。

\*/

#include<bits/stdc++.h>

using namespace std;

#define inf 0x3f3f3f3f

#define ll long long

const int maxn=8005;

const int mod=1e9+7;

const double eps=1e-8;

const double PI = acos(-1.0);

#define lowbit(x) (x&(-x))

ll gcd(ll a,ll b){return b==0?a:gcd(b,a%b);}

ll qpow(ll a,ll b){ll t=1;while(b){if(b%2){t=(t\*a)%mod;b--;}a=(a\*a)%mod;b/=2;}return t;}

ll inv(ll a,ll p){return qpow(a,p-2);}

int col[maxn<<2],last=-1,ans[maxn];

void pushDown(int rt)

{

if(col[rt]!=-1)

{

col[rt<<1]=col[rt<<1|1]=col[rt];

col[rt]=-1;

}

}

void update(int L,int R,int C,int l,int r,int rt)

{

if(L<=l&&r<=R)

{

col[rt]=C;

return ;

}

if(col[rt]==C) return ;

pushDown(rt);

int m=(l+r)>>1;

if(L<=m) update(L,R,C,l,m,rt<<1);

if(m<R) update(L,R,C,m+1,r,rt<<1|1);

}

void query(int l,int r,int rt)

{

if(l==r)

{

if(col[rt]!=-1&&col[rt]!=last)

{

ans[col[rt]]++;

}

last=col[rt];

return ;

}

pushDown(rt);

int m=(l+r)>>1;

query(l,m,rt<<1);

query(m+1,r,rt<<1|1);

}

int main()

{

std::ios::sync\_with\_stdio(false);

int n;

while(cin>>n&&n)

{

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

memset(ans,0,sizeof(ans));

int x,y,c;

last=-1;

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

{

cin>>x>>y>>c;

update(x+1,y,c,1,maxn,1);

}

query(1,maxn,1);

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

{

if(ans[i])

{

cout<<i<<" "<<ans[i]<<endl;

}

}

cout<<endl;

}

return 0;

}

### 7、线段树——维护DFS序

/\*

树的dfs序就是用来维护一系列树上的问题的，这类问题主要是解决一棵树上的所有后代结点信息的更改和祖先结点有关，主要先通过dfs来记录一个树的每一个顶点的出入时间戳，来控制它子树上的所有结点的状态的更新。

用in数组记录每个结点入栈的时间，out数组记录出栈时间（如果一个结点出栈之前没有结点出栈，那么出栈时间为上一个结点（可能是它自己）的入栈时间；否则出栈时间就等于上一个点的出栈时间）

\*/

//第一种——更改某个结点只影响其父节点 POJ 3321

/\*

题意：

给一棵n个节点的树，每个节点开始有一个苹果，m次操作?

1.将某个结点的苹果数异或 1?

2.查询一棵子树内的苹果数

思路：因为更改某个结点不影响其子结点，而影响其父结点，所以只需pushUp操作即可，update的时候也是到l==r（找到要更改的结点）对其更改，query的时候就是找in[a]到out[a]这段区间的和了。

\*/

#include<cstdio>

#include<cstring>

#include<algorithm>

#include<iostream>

#include<string>

#include<vector>

#include<stack>

#include<bitset>

#include<cstdlib>

#include<cmath>

#include<set>

#include<list>

#include<deque>

#include<map>

#include<queue>

#include<iomanip>

using namespace std;

#define inf 0x3f3f3f3f

#define ll long long

const int maxn=200005;

const double eps=1e-8;

const double PI = acos(-1.0);

struct node

{

int to,next;

}eg[maxn];

int head[maxn],tot=0,sum[maxn<<2];

void init()

{

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

tot=0;

}

void addedge(int u,int v)

{

eg[tot].to=v;

eg[tot].next=head[u];

head[u]=tot++;

}

int in[maxn],out[maxn],id=0,q[maxn];

void dfs(int x,int fa)

{

q[++id]=x;

in[x]=id;

for(int i=head[x];~i;i=eg[i].next)

{

if(eg[i].to!=fa)

{

dfs(eg[i].to,x);

}

}

out[x]=id;

}

void pushUp(int rt)

{

sum[rt]=sum[rt<<1]+sum[rt<<1|1];

}

void build(int l,int r,int rt)

{

if(l==r)

{

sum[rt]=1;

return;

}

int m=(l+r)>>1;

build(l,m,rt<<1);

build(m+1,r,rt<<1|1);

pushUp(rt);

}

void update(int L,int l,int r,int rt)

{

if(l==r)

{

sum[rt]^=1;

return ;

}

int m=(l+r)>>1;

if(m>=L) update(L,l,m,rt<<1);

else update(L,m+1,r,rt<<1|1);

pushUp(rt);

}

int query(int L,int R,int l,int r,int rt)

{

if(l>=L&&r<=R)

{

return sum[rt];

}

int m=(l+r)>>1;

int ans=0;

if(m>=L)

ans+=query(L,R,l,m,rt<<1);

if(m<R)

ans+=query(L,R,m+1,r,rt<<1|1);

return ans;

}

int main()

{

std::ios::sync\_with\_stdio(false);

std::cin.tie(0);

std::cout.tie(0);

int n;

while(cin>>n)

{

id=0;

memset(in,0,sizeof(in));

memset(out,0,sizeof(out));

init();

for(int i=0;i<n-1;i++)

{

int a,b;

cin>>a>>b;

addedge(a,b);

addedge(b,a);

}

dfs(1,0);

int m;

cin>>m;

build(1,n,1);

while(m--)

{

char c;

int a;

cin>>c>>a;

if(c=='C')

{

update(in[a],1,n,1);

}

else

{

cout<<query(in[a],out[a],1,n,1)<<endl;

}

}

}

return 0;

}

//第二种——更改某个结点只影响其子结点（HDU - 3974）

/\*

题意是给定一棵树，然后一种操作是指定一个点，这个点及这个点的的子树被染色，另一种操作是指定一个点，问这个点的颜色。

\*/

/\*

思路：因为更改某个点不影响其父节点，所以只用pushDown操作，把根节点信息下推，所以update里要传两个参数L和R，分别是in[x],out[x]，定位到这个区间，然后更改，再下推更新其子结点。query的时候因为是要找某个点的任务是什么，所以直接找in[x]就行了，即l==r的时候返回sum[rt]。

\*/

#include<bits/stdc++.h>

using namespace std;

#define inf 0x3f3f3f3f

#define ll long long

const int maxn=50005;

const int mod=1e9+7;

const double eps=1e-8;

const double PI = acos(-1.0);

#define lowbit(x) (x&(-x))

int sum[maxn<<2],id;

int in[maxn],out[maxn],vis[maxn],n;

vector<int> g[maxn];

void init()

{

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

g[i].clear();

memset(vis,0,sizeof(vis));

// memset(sum,-1,sizeof(sum));

id=0;

}

void addedge(int u,int v)

{

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

}

void dfs(ll x)

{

in[x]=++id;

for(int i=0;i<g[x].size();i++)

{

dfs(g[x][i]);

}

out[x]=id;

}

void pushDown(int rt)

{

if(sum[rt]!=-1)

{

sum[rt<<1]=sum[rt];

sum[rt<<1|1]=sum[rt];

sum[rt]=-1;

}

}

void build(int l,int r,int rt)

{

sum[rt]=-1;//开始所有节点赋为-1

if(l==r)

{

return ;

}

int m=(l+r)>>1;

build(l,m,rt<<1);

build(m+1,r,rt<<1|1);

}

void update(int L,int R,int C,int l,int r,int rt)

{

if(L<=l&&r<=R)

{

sum[rt]=C;

return ;

}

pushDown(rt);

int m=(l+r)>>1;

if(m>=L)

{

update(L,R,C,l,m,rt<<1);

}

if(R>m)

update(L,R,C,m+1,r,rt<<1|1);

}

int query(int L,int l,int r,int rt)

{

if(l==r)

{

return sum[rt];

}

pushDown(rt);

int m=(l+r)>>1;

if(m>=L)

query(L,l,m,rt<<1);

else

query(L,m+1,r,rt<<1|1);

}

int main()

{

std::ios::sync\_with\_stdio(false);

int t,cas=0;

cin>>t;

while(t--)

{

init();

cin>>n;

for(int i=0;i<n-1;i++)

{

int u,v;

cin>>u>>v;

addedge(v,u);

vis[u]=1;

}

cout<<"Case #"<<++cas<<":"<<endl;

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

{

if(!vis[i])

dfs(i);

}

/\* for(int i=1;i<=n;i++)

{

cout<<in[i]<<" "<<out[i]<<endl;

}\*/

build(1,n,1);

int m;

cin>>m;

while(m--)

{

char c;

int x,y;

cin>>c>>x;

if(c=='T')

{

cin>>y;

// cout<<in[x]<<" "<<out[x]<<endl;

update(in[x],out[x],y,1,n,1);

}

else

{

cout<<query(in[x],1,n,1)<<endl;

}

}

}

return 0;

}

### 主席树之静态区间第k大

#include<cstdio>

#include<cstring>

#include<algorithm>

#define mid (l+r)/2

using namespace std;

const int N = 200010;

int n, q, m, cnt = 0;

int a[N], b[N], T[N];

int sum[N<<5], L[N<<5], R[N<<5];

inline int build(int l, int r)

{

int rt = ++ cnt;

sum[rt] = 0;

if (l < r)

{

L[rt] = build(l, mid);

R[rt] = build(mid+1, r);

}

return rt;

}

inline int update(int pre, int l, int r, int x)

{

int rt = ++ cnt;

L[rt] = L[pre];

R[rt] = R[pre];

sum[rt] = sum[pre]+1;

if (l < r)

{

if (x <= mid) L[rt] = update(L[pre], l, mid, x);

else R[rt] = update(R[pre], mid+1, r, x);

}

return rt;

}

inline int query(int u, int v, int l, int r, int k)

{

if (l >= r) return l;

int x = sum[L[v]] - sum[L[u]];

if (x >= k) return query(L[u], L[v], l, mid, k);

else return query(R[u], R[v], mid+1, r, k-x);

}

int main()

{

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

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

{

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

b[i] = a[i];

}

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

m = unique(b+1, b+1+n)-b-1;

T[0] = build(1, m);

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

{

int t = lower\_bound(b+1, b+1+m, a[i])-b;

T[i] = update(T[i-1], 1, m, t);

}

while (q --)

{

int x, y, z;

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

int t = query(T[x-1], T[y], 1, m, z);

printf("%d\n", b[t]);

}

return 0;

}

### 划分树

/\*

HDU 2665 划分树模板题 (查询区间第K大)

\*/

const int MAXN=100010;

int tree[30][MAXN];//表示每层每个位置的值

int sorted[MAXN];//已经排序的数

int toleft[30][MAXN];//toleft[p][i]表示第i层从1到i有多少个数分入左边

void build(int l,int r,int dep)

{

if(l==r)return;

int mid=(l+r)>>1;

int same=mid-l+1;//表示等于中间值而且被分入左边的个数

for(int i=l;i<=r;i++)

if(tree[dep][i]<sorted[mid])

same--;

int lpos=l;

int rpos=mid+1;

for(int i=l;i<=r;i++)

{

if(tree[dep][i]<sorted[mid])//比中间的数小，分入左边

tree[dep+1][lpos++]=tree[dep][i];

else if(tree[dep][i]==sorted[mid]&&same>0)

{

tree[dep+1][lpos++]=tree[dep][i];

same--;

}

else //比中间值大分入右边

tree[dep+1][rpos++]=tree[dep][i];

toleft[dep][i]=toleft[dep][l-1]+lpos-l;//从1到i放左边的个数

}

build(l,mid,dep+1);

build(mid+1,r,dep+1);

}

//查询区间第k大的数，[L,R]是大区间，[l,r]是要查询的小区间

int query(int L,int R,int l,int r,int dep,int k)

{

if(l==r)return tree[dep][l];

int mid=(L+R)>>1;

int cnt=toleft[dep][r]-toleft[dep][l-1];//[l,r]中位于左边的个数

if(cnt>=k)

{

//L+要查询的区间前被放在左边的个数

int newl=L+toleft[dep][l-1]-toleft[dep][L-1];

//左端点加上查询区间会被放在左边的个数

int newr=newl+cnt-1;

return query(L,mid,newl,newr,dep+1,k);

}

else

{

int newr=r+toleft[dep][R]-toleft[dep][r];

int newl=newr-(r-l-cnt);

return query(mid+1,R,newl,newr,dep+1,k-cnt);

}

}

int main(){

int T;

int n,m;

int s,t,k;

scanf("%d",&T);

while(T--)

{

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

memset(tree,0,sizeof(tree));

for(int i=1;i<=n;i++)//从1开始

{

scanf("%d",&tree[0][i]);

sorted[i]=tree[0][i];

}

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

build(1,n,0);

while(m--)

{

scanf("%d%d%d",&s,&t,&k);

printf("%d\n",query(1,n,s,t,0,k));

}

}

return 0;

}

## LCA

### 1、LCA-ST表O(nlogn)

#include<cmath>

#include<cstdio>

#include<cstring>

#include<iostream>

#include<algorithm>

const int N=1000100;

using namespace std;

int n,m,s,tot=0,cnt=0;

int head[N],nxt[N],to[N];

int fir[N],order[N],depth[N];

int f[20][N],rec[20][N];

void addedge(int x,int y)

{

cnt++;

nxt[cnt]=head[x];

head[x]=cnt;

to[cnt]=y;

}

void dfs(int u,int dep)//dfs处理出三个数组

{

fir[u]=++tot,order[tot]=u,depth[tot]=dep;

for(int i=head[u];i!=-1;i=nxt[i])

{

int v=to[i];

if(!fir[v])

{

dfs(v,dep+1);

order[++tot]=u,depth[tot]=dep;

}

}

}

int lca(int l,int r)

{

l=fir[l],r=fir[r];

if(l>r) swap(l,r);

int k=0;

while((1<<k)<=r-l+1) k++;

k--;

if(f[k][l]<f[k][r-(1<<k)+1]) return rec[k][l];

else return rec[k][r-(1<<k)+1];

}

int main()

{

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

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

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

{

int x,y;

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

addedge(x,y);

addedge(y,x);

}

dfs(s,1);

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

f[0][i]=depth[i],rec[0][i]=order[i];

for(int i=1;i<=log(tot)/log(2);i++)

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

{

if(f[i-1][j]<f[i-1][j+(1<<(i-1))])

f[i][j]=f[i-1][j],rec[i][j]=rec[i-1][j];

else f[i][j]=f[i-1][j+(1<<(i-1))],rec[i][j]=rec[i-1][j+(1<<(i-1))];

}

//rec记录的是区间内深度最小值的编号

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

{

int l,r;

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

printf("%d\n",lca(l,r));

}

return 0;

}

### 2、LCA-tarjan离线O(n+q)

#include<cstdio>

#define il inline

const int N=500010;

struct edge

{

int to,nxt;

} g[2\*N];

struct Edge

{

int to,nxt,lca;

} G[2\*N];

int head[N],cnt,fa[N],Cnt,s,Head[N],dis[N],m,n;

bool vis[N];

il void add(int u,int v)

{

g[++cnt]=(edge){v,head[u]};

head[u]=cnt;

}

il void Add(int u,int v)

{

G[++Cnt]=(Edge){v,Head[u]};

Head[u]=Cnt;

}

il int find(int x)

{

if(x!=fa[x])fa[x]=find(fa[x]);

return fa[x];

}

il void Tarjan(int x)

{

vis[x]=true;

fa[x]=x;

for(int v,i=head[x]; i; i=g[i].nxt)

{

v=g[i].to;

if(!vis[v])Tarjan(v),fa[v]=x;

}

for(int v,i=Head[x]; i; i=G[i].nxt)

if(vis[G[i].to])

{

int S=-1;

if(i%2)S=1;

G[i].lca=find(G[i].to);

G[i+S].lca=G[i].lca;

}

}

int main ( )

{

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

for(int u,v,i=1; i<n; i++)scanf("%d%d",&u,&v),add(u,v),add(v,u);

for(int u,v,i=1; i<=m; i++)scanf("%d%d",&u,&v),Add(u,v),Add(v,u);

Tarjan(s);

for(int i=1; i<=m; i++)printf("%d\n",G[i\*2].lca);

return 0;

}

### 3、LCA-倍增O((n+q)logn)

//洛谷p3379模板题

#include<cmath>

#include<cstdio>

#include<cstring>

#include<iostream>

#include<algorithm>

using namespace std;

const int N=1000100;

int n,m,s,tot=0,cnt=0;

int head[N],nxt[N],to[N];

int d[N],f[30][N];

void addedge(int x,int y)

{

cnt++;

nxt[cnt]=head[x];

head[x]=cnt;

to[cnt]=y;

}

void dfs(int u,int dep)//处理出各个点的深度

{

d[u]=dep;

for(int i=head[u];i!=-1;i=nxt[i])

{

int v=to[i];

if(!d[v]) dfs(v,dep+1),f[0][v]=u;

}

}

int LCA(int x,int y)

{

int l=0;

while((1<<l)<=n) l++;

l--;//l表示的是最大的i为多少，当然，不用求l也可以，只要是一个够大的数像20即可

if(d[x]<d[y]) swap(x, y);//让x为深度较大的

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

if(d[y]<=d[x]-(1<<i)) x=f[i][x];//不断爬树，使深度相同

if(x==y) return x;

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

{

if(f[i][x]!=f[i][y])//不同就一起爬树

{

x=f[i][x];

y=f[i][y];

}

}

return f[0][x];

}

int main()

{

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

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

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

{

int x,y;

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

addedge(x,y);

addedge(y,x);

}

f[0][s]=s;

dfs(s,1);

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

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

f[i][j]=f[i-1][f[i-1][j]];//f[i][j] 表示 j的2^i 倍祖先,所以f[0][v]就是v的父亲节点u

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

{

int l,r;

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

printf("%d\n",LCA(l,r));

}

return 0;

}

### LCA应用

（1）.树上两点最短距离(poj1986)

思路:dfs处理出每个点i到1的距离dis[i],答案就是dis[u]+dis[v]-2\*dis[lca(u,v)]。

（2）.森林中两点最短距离(hdu2874)

思路:建立一个虚根0，和所有树的跟连一条边权为0的边，用并查集判断两点是否在同一颗树中，后面思路同（1）。

## 单调队列

//洛谷p1886

//n个数字,窗口大小为k，求每次滑动后窗口的最大值和最小值 https://www.luogu.org/problemnew/solution/P1886

#include<cstdio>

#include<cstring>

using namespace std;

struct Monotone\_queue

{

static const int maxn=1000001;

int n,k,a[maxn];

int q[maxn],head,tail,p[maxn];//同题目叙述一样，q是单调队列，p是对应编号。

void read()

{

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

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

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

}//读入不必说了

void monotone\_max()//单调最大值

{

head=1;

tail=0;

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

{

while(head<=tail&&q[tail]<=a[i])

tail--;

q[++tail]=a[i];

p[tail]=i;

while(p[head]<=i-k)

head++;

if(i>=k)printf("%d ",q[head]);

}

printf("\n");

}

void monotone\_min()

{

head=1;

tail=0;//为啥要这样呢?因为head要严格对应首元素，tail要严格对应尾元素，所以当tail>=head时，说明有元素。而一开始队列为空，说一要这样赋值。其实这跟普通队列一样。

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

{

//a[i]表示当前要处理的值

while(head<=tail&&q[tail]>=a[i])

tail--;//只要队列里有元素，并且尾元素比待处理值大，即表示尾元素已经不可能出场，所以出队。直到尾元素小于待处理值，满足"单调"。

q[++tail]=a[i];//待处理值入队。

p[tail]=i;//同时存下其编号

while(p[head]<=i-k)

head++;//如果队首元素已经"过时"，出队。

if(i>=k)printf("%d ",q[head]);//输出最值，即队首元素。i>=k表示该输出，至于why就自己看题目。

}

printf("\n");

}

} worker;

int main()

{

worker.read();

worker.monotone\_min();

worker.monotone\_max();

return 0;

}

/\*

变形：

1.求出每一项前的k个数到它这个区间内的最小值。

思路：去掉if(i>=k)的判断即可。

\*/

## 单调栈

1.奶牛头朝右，问所有奶牛能看的的奶牛数的和。（poj 3250）

#include<iostream>

#include<stack>

using namespace std;

#define inf 0x3f3f3f3f

#define ll long long

const int maxn=200005;

const int mod=1e9+7;

const double eps=1e-8;

ll a[maxn];

int main()

{

std::ios::sync\_with\_stdio(false);

int n;

while(cin>>n)

{

stack<int> s;

ll ans=0;

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

{

if(i<=n)

cin>>a[i];

else

a[i]=inf;

if(s.empty()||a[i]<a[s.top()])

{

s.push(i);

}

else

{

while(!s.empty()&&a[i]>=a[s.top()])

{

ans+=i-s.top()-1;

s.pop();

}

s.push(i);

}

}

cout<<ans<<endl;

}

return 0;

}

2.求仅由0，1组成的矩阵中，全部由1组成的子矩阵的最大面积。(poj3494)

思路:用h[j]表示第j列的1的连续情况，即如果这一行是1并且上一行也是，那么h[j]++，用a[j]暂存h[j]的值，维护一个单调递减栈（栈顶最大），如果a[j]>=a[栈顶]，入栈；否则出栈直到找到第一个比a[j]大的，并且同时更新答案，即(j-top)\*a[j]。最后把最后出栈的入栈，更新a[top]为a[j]，表示top到j都存在一个高度为a[j]的矩阵。

#include<cstdio>

#include<stack>

#include<cstring>

using namespace std;

const int maxn=2005;

int h[maxn],a[maxn];

int main()

{

int m,n;

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

{

memset(h,0,sizeof(h)),memset(a,0,sizeof(a));

int ans=0;

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

{

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

{

int c;

scanf("%d",&c);

if(c) h[j]++;

else h[j]=0;

a[j]=h[j];

}

a[n+1]=-1;

stack<int> s;

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

{

if(s.empty()||a[j]>=a[s.top()]) s.push(j);

else

{

int top;

while(!s.empty()&&a[j]<a[s.top()])

{

top=s.top();

s.pop();

ans=max(ans,a[top]\*(j-top));

}

s.push(top);

a[top]=a[j];

}

}

}

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

}

return 0;

}

3.给出一组数字，求一区间，使得区间元素和乘以区间最小值最大，结果要求给出这个最大值和区间的左右端点。(poj2796)

思路:和上题类似，用a[i]暂存每个点的值，s储存前缀和，维护一个单调递减栈(栈顶最大)，大于栈顶就入栈，否则往左边延伸，找到比它大的所有值，更新能保证a[i]最小的区间的答案，最后一个出栈的要入栈，并更新a[top]=a[i]，这里是贪心+并查集的思想，因为a[i]>=0，所以保证a[i]最小的前提下区间越大越好。

#include<cstdio>

#include<stack>

#include<cstring>

using namespace std;

#define ll long long

const int maxn=100005;

ll a[maxn],s[maxn];

int main()

{

int n;

while(~scanf("%d",&n))

{

memset(s,0,sizeof(s));

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

{

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

s[i]=s[i-1]+a[i];

}

a[n+1]=-1;

stack<int> st;

ll ans=0,l,r;

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

{

if(st.empty()||a[i]>=a[st.top()])

st.push(i);

else

{

int top;

while(!st.empty()&&a[i]<a[st.top()])

{

top=st.top();

st.pop();

if((s[i-1]-s[top-1])\*a[top]>=ans)

{

ans=(s[i-1]-s[top-1])\*a[top];

l=top;

r=i-1;

}

}

st.push(top);

a[top]=a[i];

}

}

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

}

return 0;

}

## 树

### 1、树的直径

题意：有一颗n个结点的带权的无向树, 在s结点放两个机器人, 这两个机器人会把树的每条边都走一遍, 但是最后机器人不要求回到出发点. 问你两个机器人走的路总长之和的最小值是多少?

思路：考虑从一个结点遍历整个树再回到原点需要把每个边计算两遍，这里机器人不用回到出发点，所以两个机器人到达的点越远越好。让两个机器人在初始位置在直径上背道而驰，这样最优解就是所有边\*2-直径，因为直径只走了一次，而其他边必走两遍。

#include<queue>

#include<vector>

#include<cstdio>

#include<cstring>

#include<iostream>

#include<algorithm>

#define ll long long

using namespace std;

struct node

{

ll to,val;

};

const int N=100010;

vector<node> g[N];

ll vis[N],dis[N],ans;

ll bfs(ll x)

{

memset(dis,0,sizeof(dis));

memset(vis,0,sizeof(vis));

queue<ll> q;

q.push(x);

vis[x]=1;

ll point=0;

while(!q.empty())

{

ll f=q.front();

q.pop();

if(dis[f]>ans)

{

ans=dis[f];

point=f;

}

int sz=g[f].size();

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

{

node temp=g[f][i];

if(vis[temp.to]==0)

{

vis[temp.to]=1;

dis[temp.to]=dis[f]+temp.val;

q.push(temp.to);

}

}

}

return point;

}

int main()

{

ll n,m;

while(scanf("%lld %lld",&n,&m)==2)

{

ll sum=0;

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

{

ll u,v,w;

scanf("%lld %lld %lld",&u,&v,&w);

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

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

sum+=w;

}

ans=0;

ll point=bfs(m);

ans=0;

bfs(point);

printf("%d\n",(ll)(sum\*2-ans));

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

}

}

### 2、二分+LIS+DFS序

链接：https://ac.nowcoder.com/acm/contest/368/B

题意:有一棵n个节点的二叉树，1为根节点，每个节点有一个值wi。现在要选出尽量多的点。

对于任意一棵子树，都要满足：

如果选了根节点的话，在这棵子树内选的其他的点都要比根节点的值大；

如果在左子树选了一个点，在右子树中选的其他点要比它小。

第一行一个整数n。

第二行n个整数wi，表示每个点的权值。

接下来n行，每行两个整数a,b。第i+2行表示第i个节点的左右儿子节点。没有为0。

思路:因为根节点的权值最小，其次是右子树的点，最后是左子树的点，所以按照先根，再右子树，再左子树的顺序dfs整棵树，求出dfs序，在dfs序上求最长上升子序列。

#include<bits/stdc++.h>

using namespace std;

#define ll long long

const int maxn=100005;

ll w[maxn],lc[maxn],rc[maxn],q[maxn],tot,a[maxn],cnt;

void dfs(int x)

{

if(!x) return ;

q[++tot]=x;

dfs(rc[x]);

dfs(lc[x]);

}

int main()

{

std::ios::sync\_with\_stdio(false);

int n;

cin>>n;

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

{

cin>>w[i];

}

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

{

cin>>lc[i]>>rc[i];

}

dfs(1);

a[++cnt]=w[q[1]];

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

{

if(w[q[i]]>a[cnt]) a[++cnt]=w[q[i]];

else a[lower\_bound(a+1,a+cnt+1,w[q[i]])-a]=w[q[i]];

}

cout<<cnt<<endl;

return 0;

}

### 3、字典树应用

洛谷p2580:查找某个字符串出现一次还是两次还是不出现。

思路：用前缀的板子，search函数+vis[root]判断即可。

### 查找字符串是否出现

/\*

trie tree的储存方式：将字母储存在边上，边的节点连接与它相连的字母

trie[rt][x]=tot:rt是上个节点编号，x是字母，tot是下个节点编号

\*/

#include<cstdio>

#include<iostream>

#include<algorithm>

#include<cstring>

#define maxn 2000010

using namespace std;

int tot=1,n;

int trie[maxn][26];

bool isw[maxn];//查询整个单词用

void insert(char \*s,int rt)

{

for(int i=0;s[i];i++)

{

int x=s[i]-'a';

if(trie[rt][x]==0)//现在插入的字母在之前同一节点处未出现过

{

trie[rt][x]=++tot;//字母插入一个新的位置，否则不做处理

}

rt=trie[rt][x];//为下个字母的插入做准备

}

isw[rt]=true;//标志该单词末位字母的尾结点，在查询整个单词时用到

}

bool find(char \*s,int rt)

{

for(int i=0;s[i];i++)

{

int x=s[i]-'a';

if(trie[rt][x]==0)return false;//以rt为头结点的x字母不存在，返回0

rt=trie[rt][x];//为查询下个字母做准备

}

return true;

//查询整个单词时，应该return isw[rt]

}

char s[22];

int main()

{

tot=0;

int rt=1;

scanf("%d",&n);

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

{

cin>>s;

insert(s,rt);

}

scanf("%d",&n);

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

{

cin>>s;

if(find(s,rt))printf("YES\n");

else printf("NO\n");

}

return 0;

}

### 查找前缀出现次数

#include<iostream>

#include<cstring>

#include<cstdio>

#include<algorithm>

using namespace std;

int trie[400001][26],len,root,tot,sum[400001];

bool p;

int n,m;

char s[11];

void insert()

{

len=strlen(s);

root=0;

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

{

int id=s[i]-'a';

if(!trie[root][id]) trie[root][id]=++tot;

sum[trie[root][id]]++;//前缀后移一个位置保存

root=trie[root][id];

}

}

int search()

{

root=0;

len=strlen(s);

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

{

int id=s[i]-'a';

if(!trie[root][id]) return 0;

root=trie[root][id];

}//root经过此循环后变成前缀最后一个字母所在位置的后一个位置

return sum[root];//因为前缀后移了一个保存，所以此时的sum[root]就是要求的前缀出现的次数

}

int main()

{

scanf("%d",&n);

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

{

cin>>s;

insert();

}

scanf("%d",&m);

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

{

cin>>s;

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

}

}

字典树之求前缀单词的数量

Input

输入数据的第一部分是一张单词表,每行一个单词,单词的长度不超过10,它们代表的是老师交给Ignatius统计的单词,一个空行代表单词表的结束.第二部分是一连串的提问,每行一个提问,每个提问都是一个字符串.

注意:本题只有一组测试数据,处理到文件结束.

Output

对于每个提问,给出以该字符串为前缀的单词的数量.

Sample Input

banana

band

bee

absolute

acm

ba

b

band

abc

Sample Output

2

3

1

0

#include <iostream>

#include<string>

using namespace std;

struct trienode

{

int countt;//统计单词前缀出现的次数

trienode\* nextt[26];//指向各子树的指针

bool exist;//标记该结点处是否构成单词

trienode():countt(0),exist(false)

{

for(int i=0;i<26;i++)

{

nextt[i]=NULL;

}

}

};

void trieinsert(trienode\* root,string &word)

{

trienode \*node=root;

int id;

int len=word.size();

int i=0;

while(i<len)

{

id=word[i]-'a';

if(node->nextt[id]==NULL)

node->nextt[id]=new trienode();

node=node->nextt[id];

node->countt+=1;

i++;

}

node->exist=true;//单词结束，可以构成一个单词

}

int triesearch(trienode\* root,string &word)

{

trienode \*node=root;

int len=word.size();

int i=0;

while(i<len)

{

int id=word[i]-'a';

if(node->nextt[id]!=NULL)

{

node=node->nextt[id];

i++;

}

else

return 0;

}

return node->countt;

}

int main()

{

trienode \*root=new trienode();

// int flag=false;

string word;

while(getline(cin,word)&&word.compare("")!=0)

{

trieinsert(root,word);

}

while(cin>>word)

cout<<triesearch(root,word)<<endl;

return 0;

}

例题：

给你一堆英文单词（可能有4000000个。用普通查询铁定让你TLE）。找出出现次数最多的，输出这个单词，并输出出现的次数。

#include<iostream>

#include<algorithm>

using namespace std;

struct Dictree

{

int count;//单词出现的次数

struct Dictree \*tire[26];//26个子节点

}\*a;

void init()

{

a = new Dictree;

for(int i = 0;i < 26;i++)

a->tire[i] = NULL;//子节点指针置空

}

int insert(char str[])

{

int len ,res;

Dictree \*head = a;//head是头指针，每次变化。但a不变，每次从头指针开始

len = strlen(str);

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

{

res = (int) (str[i] - 97);//下标对应字母

if(head->tire[res] == NULL)//没有此字母，开始新节点并初始化

{

head->tire[res] = new Dictree;

head = head->tire[res];//和下一个顺序不能反，先指向开辟节点，再赋值为0

head->count = 0;

for(int j = 0;j < 26;j++)

{

head->tire[j] = NULL;

}

}

else head = head->tire[res];

}

head->count++;//记录单词出现次数

return head->count;//返回单词出现的次数

}

int main()

{

int num, tmp,maxlen = 0;

char ani[11],ans[11];

init();

cin>>num;

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

{

cin>>ani;

tmp = insert(ani);

if(tmp > maxlen)

{

maxlen = tmp;

strcpy(ans,ani);

}

}

cout<<ans<<" "<<maxlen<<endl;

return 0;

}

### 求树的直径

#include<queue>

#include<vector>

#include<cstdio>

#include<cstring>

#include<iostream>

#include<algorithm>

using namespace std;

struct Node

{

int to,cap;

};

const int N=100010;

vector<Node> v[N];

int vis[N],dis[N],ans;

int bfs(int x)

{

memset(dis,0,sizeof(dis));

memset(vis,0,sizeof(vis));

queue<int> q;

q.push(x);

vis[x]=1;

int point=0;

while(!q.empty())

{

int f=q.front();

q.pop();

if(dis[f]>ans)

{

ans=dis[f];

point=f;

}

for(int i=0;i<v[f].size();i++)

{

Node temp=v[f][i];

if(vis[temp.to]==0)

{

vis[temp.to]=1;

dis[temp.to]=dis[f]+temp.cap;

q.push(temp.to);

}

}

}

return point;

}

int main()

{

int n;

while(~scanf("%d",&n))

{

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

{

int x,y;

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

v[x].push\_back((Node){y,1});

v[y].push\_back((Node){x,1});

}

ans=0;

int point=bfs(1);

ans=0;

bfs(point);

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

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

}

}

## 图论

### 1、图论知识点

1.Purfer Sequence :一个含有 n 个节点的 Purfer Sequence 有 n-2 个数，Purfer Sequence 中的每个数是 1~n 中的一个数,一个 Purfer Sequence 和一棵树一一对应，一个点的度数减一表示它在 Purfer Sequence 中出现了几次。

例BZOJ1005:给出标号为1到N的点,以及某些点最终的度数,允许在任意两点间连线,可产生多少棵度数满足要求的树?

思路:记cnt为度数有要求的点的个数，sum=d[1]-1+…d[cnt]-1,n为点的个数,排列组合得出C(n-2,sum)\*(sum!/((d[1]-1)!\*…d[cnt]-1)!)\*(n-cnt)^(n-2-sum）。

2.判断有向树:用并查集判时还要注意只有一个根节点，if(fu!=fv&&fv==v){pre[fu]=fv;}else{flag=0;}

3.

点覆盖、最小点覆盖：点覆盖集即一个点集，使得所有边至少有一个端点在集合里。或者说是“点” 覆盖了所有“边”。。

最小点覆盖(minimum vertex covering)：点最少的点覆盖。

点覆盖数(vertex covering number)：最小点覆盖的点数。

独立集：独立集即一个点集，集合中任两个结点不相邻，则称V为独立集。或者说是导出的子图是零图（没有边）的点集。

最大独立集(maximum independent set)：点最多的独立集。

独立数(independent number)：最大独立集的点。

若把上面最小点覆盖和最大独立集中的端点数改成点的权值，分别就是最小点权覆盖和最大点权独立集的定义。

最大点权独立集=总权值-最小点权覆盖集。

最小点权覆盖集=图的最小割值=最大流。

4.欧拉路：

无向图：有两个奇度点的叫欧拉路径，无奇度点的叫欧拉回路

有向图：起点出度比入度大1终点入度比出度大1的叫欧拉路径，所有点入度=出度的叫欧拉回路。

### 2、并查集

### 并查集递归版

//个人比较喜欢的写法

int pre[maxn];

void init()

{

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

{

pre[i]=i;

}

}

int find(int x)

{

if(x==pre[x])

return x;

else

{

int root=find(pre[x]);

return pre[x]=root;

}

}

### Poj2236

每台电脑只能与距离它不超过 d 米的其它电脑直接通信。如果电脑 A 和电脑 B 可以直接通信，或存在一台电脑 C 既可与 A 也可与 B 通信，那么电脑 A 和电脑 B 之间就能够通信。 1. "O p" (1 <= p <= N)，表示维护电脑 p 。   
2. "S p q" (1 <= p, q <= N)，表示测试电脑 p 和 q 是否能够通信。

#include <iostream>

#include<cmath>

#include<vector>

using namespace std;

const int N=1e3+5;

int n;

double d;

struct node

{

double x,y;

int sta;

};

double dis(node a,node b)

{

return sqrt((a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y));

}

node g[N];

int pre[N];

int find(int x)

{

if(x==pre[x])

return x;

return pre[x]=find(pre[x]);

}

vector<int> v[N];

int main()

{

ios::sync\_with\_stdio(false);

while(cin>>n>>d)

{

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

{

cin>>g[i].x>>g[i].y;

g[i].sta=0;

}

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

{

pre[i]=i;

}

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

{

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

{

if(dis(g[i],g[j])<=d)

{

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

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

}

}

}

char c;

int a,b;

while(cin>>c>>a)

{

if(c=='S')

{

cin>>b;

int ta=find(a),tb=find(b);

if(ta!=tb||!g[a].sta||!g[b].sta)

cout<<"FAIL"<<endl;

else

cout<<"SUCCESS"<<endl;

}

else

{

g[a].sta=1;

int sz=v[a].size(),fi=find(a);

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

{

int fj=find(v[a][i]);

if(fi!=fj&&g[v[a][i]].sta)

pre[fj]=fi;//注意此处如果写成pre[fi]=fj；就错了，另一个题就不一定，两种都试一下

}

}

}

}

return 0;

}

### 3、二分图

### 二分图染色（判断二分图）

/\*

无向图G为二分图的充分必要条件是：G至少有两个顶点,且当存在回路时，其所有回路的长度均为偶数。回路就是环路，也就是判断是否存在奇数环。判断二分图方法：用染色法，把图中的点染成黑色和白色。首先取一个点染成白色，然后将其相邻的点染成黑色，如果发现有相邻且同色的点，那么就退出，可知这个图并非二分图。

\*/

//dfs法

#include<cstdio>

#include<cstring>

#include<vector>

using namespace std;

const int maxn=1e5+10;

vector<int> g[maxn];

int n,m,col[maxn],flag;//col:0,1,-1分别为未染色，染黑，染白

void dfs(int x,int color)

{

col[x]=color;

for(int i=0;i<g[x].size();i++)

{

int v=g[x][i];

if(col[v]==col[x])

{

flag=1;

return ;

}

if(!col[v])

dfs(v,-color);

}

}

int main()

{

while(~scanf("%d%d",&n,&m))

{

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

{

g[i].clear();

}

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

{

int u,v;

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

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

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

}

flag=0;

memset(col,0,sizeof(col));

for(int i=1;i<=n&&!flag;i++)

{

if(!col[i])

{

dfs(i,1);

}

}

if(flag)

{

puts("No");

}

else

{

puts("Yes");

}

}

return 0;

}

//bfs法

#include<cstdio>

#include<cstring>

#include<vector>

#include<queue>

using namespace std;

const int maxn=1e5+10;

vector<int> g[maxn];

int n,m,col[maxn],flag;//col:0,1,-1分别为未染色，染黑，染白

void bfs(int s)

{

queue<int> q;

q.push(s);

col[s]=1;

while(!q.empty())

{

int f=q.front();

q.pop();

for(int i=0; i<g[f].size(); i++)

{

int v=g[f][i];

if(!col[v])

{

q.push(v);

col[v]=-col[f];

}

if(col[f]==col[v])

{

flag=1;

return ;

}

}

}

}

int main()

{

while(~scanf("%d%d",&n,&m))

{

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

{

g[i].clear();

}

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

{

int u,v;

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

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

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

}

flag=0;

memset(col,0,sizeof(col));

for(int i=1; i<=n&&!flag; i++)

{

if(!col[i])

{

bfs(i);

}

}

if(flag)

{

puts("No");

}

else

{

puts("Yes");

}

}

return 0;

}

### 4、连通图

### Tarjan

/\* POJ 1236

题意：有n台电脑，现在有些电脑可以传数据到另外的电脑上，即有k条边。问题1：现在问至少需要给多少台电脑传送数据才能到达全部电脑。问题2：问至少添加多少条边才能使得往任意一台电脑传送数据便可以到达所有电脑。

思路：问题1可以转化为求入度为0的缩点个数，问题2可以转换为求max(入度为0的点个数，出度为0的点个数），因为tarjan后图变成一个DAG，要将该DAG变为一个强连通，那么最少连边就考虑吧那些出度零点连一条边到入度零点。所以答案就是在入度零点和出度零点中找最大值。先用tarjan求出强连通和缩点，如果scc为1，则输出1 0；否则照上述思路写。

\*/

/\*

\* Tarjan 算法

\* 复杂度 O(N+M)

\*/

#include<iostream>

#include<cstring>

using namespace std;

const int maxn = 20010;//点数

const int maxm = 50010;//边数

struct Edge

{

int to,next;

} edge[maxm];

int head[maxn],tot;

int low[maxn],dfn[maxn],Stack[maxn],belong[maxn];//belong 数组的值是1~scc

int index,top;

int scc;//强连通分量的个数

bool inStack[maxn];

int num[maxn];//各个强连通分量包含点的个数，数组编号 1 ~ scc

//num 数组不一定需要，结合实际情况

void addedge(int u,int v)

{

edge[tot].to = v;

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

head[u] = tot++;

}

void tajan(int u)

{

int v;

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

Stack[top++] = u;

inStack[u] = true;

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

{

v = edge[i].to;

if( !dfn[v] )

{

tajan(v);

if( low[u] > low[v] )low[u] = low[v];

}

else if(inStack[v] && low[u] > dfn[v])

low[u] = dfn[v];

}

if(low[u] == dfn[u])

{

scc++;

do

{

v = Stack[ -- top];

inStack[v] = false;

belong[v] = scc;

num[scc]++;

}

while( v != u);

}

}

void solve(int N)

{

memset(dfn,0,sizeof(dfn));

memset(inStack,false,sizeof(inStack));

memset(num,0,sizeof(num));

index = scc = top = 0;

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

if(!dfn[i])

tajan(i);

}

void init()

{

tot = 0;

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

}

int main()

{

int n;

while(cin>>n)

{

int in[maxn],out[maxn];

memset(in,0,sizeof(in));

memset(out,0,sizeof(out));

init();

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

{

int a;

while(cin>>a&&a)

addedge(i,a);

}

solve(n);

if(scc==1)

{

cout<<1<<endl<<0<<endl;

continue;

}

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

{

for(int j=head[i];~j;j=edge[j].next)

{

if(belong[i]!=belong[edge[j].to])

{

in[belong[edge[j].to]]++;

out[belong[i]]++;

}

}

}

int ans1=0,ans2=0;

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

{

if(in[i]==0)

ans1++;

}

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

{

if(out[i]==0)

ans2++;

}

ans2=max(ans1,ans2);

cout<<ans1<<endl<<ans2<<endl;

}

}

### 最短路

### dijkstra变形

变形1：(poj2253) 求两点之间所有路径最小的最大长度边，松弛操作改为

if(max(f.w,t.w)<dis[t.p])

{

dis[t.p]=max(f.w,t.w);

pq.push(node(t.p,dis[t.p]));

}

变形2：(poj1797) 求两点之间所有路径最大的最小长度边，改动较大，如下：

void Dijkstra(int now)

{

for(int i=0; i<=n; i++) dis[i]=-1;

dis[now]=INF;

priority\_queue <node> pq;

pq.push(node(now,dis[now]));

while(!pq.empty())

{

node f=pq.top();

pq.pop();

for(int i=0; i<eg[f.p].size(); i++)

{

node t=eg[f.p][i];

if(dis[t.p]<min(t.w,f.w))

{

dis[t.p]=min(t.w,f.w);

pq.push(node(t.p,dis[t.p]));

}

}

}

}

变形3：乘积最短路 取log2取值然后存入边中，其余过程不变，最后输出最短路qpow(2,dis[i])。

例题链接：https://ac.nowcoder.com/acm/contest/283/H

变形4：某些点限制遍历次数求1到n最短路(牛客370B)

只能穿过不超过 K 次被戒严的城镇。被戒严的城镇用1表示，其他用0表示。

#include<bits/stdc++.h>

using namespace std;

const int maxn=500010;

const int INF=0x3f3f3f3f;

struct node

{

int p,w,c;

node(int a,int b,int d=0)

{

p=a;

w=b;

c=d;

}

friend bool operator<(node a,node b) //权值小的先出队

{

if(a.w!=b.w) return a.w>b.w;

return a.p>b.p;

}

};

vector <node> eg[maxn];

int dis[maxn][11],n,f[maxn];

int u,v,m,k,w;

void addedge(int u,int v,int w)

{

eg[u].push\_back(node(v,w));

}

void Dijkstra(int now)

{

memset(dis,INF,sizeof(dis));

dis[now][f[now]]=0;

priority\_queue <node> pq;

pq.push(node(now,0,f[now]));

while(!pq.empty())

{

node fr=pq.top();

pq.pop();

for(int i=0; i<eg[fr.p].size(); i++)

{

node t=eg[fr.p][i];

int tmp=f[t.p]+fr.c;

if(tmp>k)

continue;

if(dis[t.p][tmp]>t.w+fr.w)

{

dis[t.p][tmp]=t.w+fr.w;

pq.push(node(t.p,dis[t.p][tmp],tmp));

}

}

}

}

int main()

{

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

{

for(int i=1; i<=n; i++) eg[i].clear(),cin>>f[i];

while(m--)

{

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

addedge(u,v,w);

addedge(v,u,w);

}

f[n]=0;

Dijkstra(1);

int ans=INF;

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

{

ans=min(ans,dis[n][i]);

}

if(ans==INF)

puts("-1");

else

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

}

return 0;

}

### Dijkstra链式前向星+堆优化(效率最高)

#include<bits/stdc++.h>

using namespace std;

const int inf=0x3f3f3f3f;

const int N=2e5+5;

struct qnode

{

int v;

int c;

qnode(int \_v=0,int \_c=0):v(\_v),c(\_c){}

bool operator <(const qnode &r)const

{

return c>r.c;

}

};

struct edge

{

int v,cost;

int next;

};

edge eg[200000];

int head[N],dist[N],tot;

bool vis[N];

void Dijkstra(int n,int sta)//点的编号从1开始

{

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

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

priority\_queue<qnode> pq;

dist[sta]=0;

pq.push(qnode(sta,0));

qnode tmp;

while(!pq.empty())

{

tmp=pq.top();

pq.pop();

int u=tmp.v;

if(vis[u])continue;

vis[u]=true;

for(int i=head[u];~i;i=eg[i].next)

{

int v=eg[i].v;

int cost=eg[i].cost;

if(!vis[v]&&dist[v]>dist[u]+cost)

{

dist[v]=dist[u]+cost;

pq.push(qnode(v,dist[v]));

}

}

}

}

void init()

{

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

tot=0;

}

void addedge(int u,int v,int w)

{

eg[tot].v=v;

eg[tot].cost=w;

eg[tot].next=head[u];

head[u]=tot++;

}

int main()

{

int n,m;

init();

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

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

{

int u,v,w;

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

addedge(u,v,w);

}

Dijkstra(n,1);

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

return 0;

}

### Dijkstra邻接表+堆优化

//<邻接表+堆优化+防重边 O((E+V)logV)> POJ2387模板题

#include <cstdio>

#include <iostream>

#include <cstring>

#include <cctype>

#include <cstdlib>

#include <algorithm>

#include <vector>

#include <queue>

#include <stack>

#include <set>

#include <map>

#include <list>

using namespace std;

const int maxn=500010;

const int INF=0x3f3f3f3f;

struct node

{

int p,w;

node(int a,int b)

{

p=a;

w=b;

}

friend bool operator<(node a,node b) //权值小的先出队

{

if(a.w!=b.w) return a.w>b.w;

return a.p>b.p;

}

};

vector <node> eg[maxn];

int dis[maxn],n;

void addedge(int u,int v,int w)

{

eg[u].push\_back(node(v,w));

}

void Dijkstra(int now)

{

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

dis[now]=0;

priority\_queue <node> pq;

pq.push(node(now,dis[now]));

while(!pq.empty())

{

node f=pq.top();

pq.pop();

for(int i=0; i<eg[f.p].size(); i++)

{

node t=eg[f.p][i];

if(dis[t.p]>t.w+f.w)

{

dis[t.p]=t.w+f.w;

pq.push(node(t.p,dis[t.p]));

}

}

}

}

int main()

{

int u,v,m,w;

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

{

for(int i=0; i<=n; i++) eg[i].clear();

while(m--)

{

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

addedge(u,v,w);

}

Dijkstra(1);

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

}

return 0;

}

### 第K短路

#include <map>

#include <set>

#include <cmath>

#include <ctime>

#include <stack>

#include <queue>

#include <cstdio>

#include <cctype>

#include <bitset>

#include <string>

#include <vector>

#include <cstring>

#include <iostream>

#include <algorithm>

#include <functional>

using namespace std;

typedef long long LL;

const int maxn = 1000 + 5;

const int INF = 0x3f3f3f3f;

int s, t, k;

bool vis[maxn];

int dist[maxn];

struct Node

{

int v, c;

Node (int \_v = 0, int \_c = 0) : v(\_v), c(\_c) {}

bool operator < (const Node &rhs) const

{

return c + dist[v] > rhs.c + dist[rhs.v];

}

};

struct Edge

{

int v, cost;

Edge (int \_v = 0, int \_cost = 0) : v(\_v), cost(\_cost) {}

};

vector<Edge>E[maxn], revE[maxn];

void Dijkstra(int n, int s)

{

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

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

priority\_queue<Node>que;

dist[s] = 0;

que.push(Node(s, 0));

while (!que.empty())

{

Node tep = que.top();

que.pop();

int u = tep.v;

if (vis[u]) continue;

vis[u] = true;

for (int i = 0; i < (int)E[u].size(); i++)

{

int v = E[u][i].v;

int cost = E[u][i].cost;

if (!vis[v] && dist[v] > dist[u] + cost)

{

dist[v] = dist[u] + cost;

que.push(Node(v, dist[v]));

}

}

}

}

int astar(int s)

{

priority\_queue<Node> que;

que.push(Node(s, 0));

k--;

while (!que.empty())

{

Node pre = que.top();

que.pop();

int u = pre.v;

if (u == t)

{

if (k) k--;

else return pre.c;

}

for (int i = 0; i < (int)revE[u].size(); i++)

{

int v = revE[u][i].v;

int c = revE[u][i].cost;

que.push(Node(v, pre.c + c));

}

}

return -1;

}

void addedge(int u, int v, int w)

{

revE[u].push\_back(Edge(v, w));

E[v].push\_back(Edge(u, w));

}

int main()

{

int n, m, u, v, w;

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

{

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

{

E[i].clear();

revE[i].clear();

}

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

{

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

addedge(u, v, w);

}

scanf("%d%d%d", &s, &t, &k);

Dijkstra(n, t);

if (dist[s] == INF)

{

puts("-1");

continue;

}

if (s == t) k++;

printf("%d\n", astar(s));

}

return 0;

}

### 最短路计数+维护点权和最大O(nlogn)

//num数组:s到d的最短路的条数,weight数组:点权,w数组:s到d的最大权值和，pre数组记录路径

#include <iostream>

#include <string.h>

#include <stdio.h>

#include <algorithm>

#include <queue>

#include <vector>

using namespace std;

const int INF=0x3f3f3f3f;

const int MAXN=30010;

struct qnode

{

int v;

int c;

qnode(int \_v=0,int \_c=0):v(\_v),c(\_c) {}

bool operator <(const qnode &r)const

{

return c>r.c;

}

};

struct edge

{

int v,cost;

int next;

};

edge eg[200000];

int head[MAXN],dist[MAXN],tot,num[MAXN],w[MAXN],weight[MAXN];

bool vis[MAXN];

void Dijkstra(int n,int sta)//点的编号从1开始

{

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

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

priority\_queue<qnode> pq;

dist[sta]=0;

pq.push(qnode(sta,0));

qnode tmp;

num[sta]=1;

w[sta]=weight[sta];

while(!pq.empty())

{

tmp=pq.top();

pq.pop();

int u=tmp.v;

if(vis[u])continue;

vis[u]=true;

for(int i=head[u]; ~i; i=eg[i].next)

{

int v=eg[i].v;

int cost=eg[i].cost;

if(!vis[v]&&dist[v]>dist[u]+cost)

{

dist[v]=dist[u]+cost;

num[v]=num[u];

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

pq.push(qnode(v,dist[v]));

}

else if(!vis[v]&&dist[v]==dist[u]+cost)

{

if(w[u]+weight[v]>w[v])

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

num[v]+=num[u];

}

}

}

}

void init()

{

memset(weight,0,sizeof(weight));

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

memset(num,0,sizeof(num));

tot=0;

}

void addedge(int u,int v,int w)

{

eg[tot].v=v;

eg[tot].cost=w;

eg[tot].next=head[u];

head[u]=tot++;

}

int main()

{

int n,m,s,e;

init();

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

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

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

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

{

int u,v,w;

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

addedge(u+1,v+1,w);//pat1003下标是从0开始

addedge(v+1,u+1,w);

}

Dijkstra(n,s+1);

printf("%d %d\n",num[e+1],w[e+1]);

return 0;

}

### 最小生成树

### kruskal+邻接表O(ElogE)//HDU 1233

#include<bits/stdc++.h>

using namespace std;

#define inf 0x3f3f3f3f

#define ll long long

const int maxn=100005;//点数

const int maxm=100005;//边数

const int mod=1e9+7;

const double eps=1e-8;

const double PI = acos(-1.0);

#define lowbit(x) (x&(-x))

struct edge

{

ll u,v,w;

} eg[maxm];

ll tot=0,pre[maxn],n,m;

void addedge(ll u,ll v,ll w)

{

eg[tot].u=u;

eg[tot].v=v;

eg[tot++].w=w;

}

bool cmp(edge a,edge b)

{

return a.w<b.w;

}

ll find(ll x)

{

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

else return pre[x]=find(pre[x]);

}

ll kruskal(ll n)

{

sort(eg,eg+tot,cmp);

ll cnt=0,ans=0;

for(ll i=0; i<tot; i++)

{

ll u=eg[i].u,v=eg[i].v,w=eg[i].w;

ll fu=find(u),fv=find(v);

if(fu!=fv)

{

ans+=w;

pre[fu]=fv;

cnt++;

}

if(cnt==n-1) break;

}

if(cnt<n-1) return -1;

else return ans;

}

void init()

{

tot=0;

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

pre[i]=i;

}

int main()

{

std::ios::sync\_with\_stdio(false);

while(cin>>n&&n)

{

init();

for(int i=0; i<n\*(n-1)/2; i++)

{

ll a,b,c;

cin>>a>>b>>c;

addedge(a,b,c);

}

cout<<kruskal(n)<<endl;

}

return 0;

}

### 次小生成树

//POJ 1679

#include<cstdio>

#include<cstring>

#include<algorithm>

#include<iostream>

#include<string>

#include<vector>

#include<stack>

#include<bitset>

#include<cstdlib>

#include<cmath>

#include<set>

#include<list>

#include<deque>

#include<map>

#include<queue>

#include<iomanip>

using namespace std;

#define inf 0x3f3f3f3f

#define ll long long

const int maxn=105;

const double eps=1e-8;

const double PI = acos(-1.0);

ll gcd(ll a,ll b)

{

return b==0?a:gcd(b,a%b);

}

bool vis[maxn];

int lowc[maxn],pre[maxn],MAX[maxn][maxn],used[maxn][maxn],g[maxn][maxn],n,m;

int prim(int cost[][maxn],int n)

{

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

memset(pre,0,sizeof(pre));

memset(MAX,0,sizeof(MAX));

memset(used,0,sizeof(used));

vis[0]=true;

pre[0]=-1;

int ans=0;

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

{

lowc[i]=cost[0][i];

}

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

{

int minc=inf;

int p=-1;

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

{

if(!vis[j]&&minc>lowc[j])

{

minc=lowc[j];

p=j;

}

}

if(minc==inf) return -1;

vis[p]=true;

ans+=minc;

used[p][pre[p]]=used[pre[p]][p]=1;

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

{

if(vis[j])

{

MAX[j][p]=MAX[p][j]=max(MAX[j][pre[p]],lowc[p]);

}

if(!vis[j]&&lowc[j]>cost[p][j])

{

lowc[j]=cost[p][j];

pre[j]=p;

}

}

}

return ans;

}

int secondPrim(int sum)

{

int ans=inf;///记录次小生成树的总值

for(int i=0;i<n;i++)///遍历所有的边

{

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

{

if(!used[i][j]&&g[i][j]!=inf)///如果该边不是构成最小生成树的边而且可以走的通

{

ans=min(sum+g[i][j]-MAX[i][j],ans);///更新次小生成树，加上该边减去环中最大的边

}

}

}

if(ans==inf) return -1;

return ans;

}

int main()

{

std::ios::sync\_with\_stdio(false);

int t;

cin>>t;

while(t--)

{

cin>>n>>m;

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

{

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

{

if(i==j) g[i][j]=0;

else

g[i][j]=g[j][i]=inf;

}

}

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

{

int a,b,w;

cin>>a>>b>>w;

g[a-1][b-1]=g[b-1][a-1]=w;

}

int a1=prim(g,n),a2=secondPrim(a1);

if(a1==a2)

cout<<"Not Unique!"<<endl;

else

cout<<a1<<endl;

}

return 0;

}

### 生成树计数

//HDU 4305 根据两点的距离不大于R，而且中间没有点建立一个图。之后就是求生成树计数了。

/\*

Matrix-Tree定理(Kirchhoff矩阵-树定理)。Matrix-Tree定理是解决生成树计数问题最有力的武器之一。它首先于1847年被Kirchhoff证明。在介绍定理之前，我们首先明确几个概念：

1、G的度数矩阵D[G]是一个n\*n的矩阵，并且满足：当i≠j时,dij=0；当i=j时，dij等于vi的度数。

2、G的邻接矩阵A[G]也是一个n\*n的矩阵， 并且满足：如果vi、vj之间有边直接相连，则aij=1，否则为0。

我们定义G的Kirchhoff矩阵(也称为拉普拉斯算子)C[G]为C[G]=D[G]-A[G]，则Matrix-Tree定理可以描述为：G的所有不同的生成树的个数等于其Kirchhoff矩阵C[G]任何一个n-1阶主子式的行列式的绝对值。所谓n-1阶主子式，就是对于r(1≤r≤n)，将C[G]的第r行、第r列同时去掉后得到的新矩阵，用Cr[G]表示。

\*/

#include <stdio.h>

#include <algorithm>

#include <iostream>

#include <string.h>

#include <vector>

#include <queue>

#include <map>

#include <set>

#include <list>

#include <string>

#include <math.h>

using namespace std;

struct Point

{

int x,y;

Point(int \_x = 0,int \_y = 0)

{

x = \_x,y = \_y;

}

Point operator - (const Point &b)const

{

return Point(x-b.x,y-b.y);

}

int operator ^(const Point &b)const

{

return x\*b.y - y\*b.x;

}

void input()

{

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

}

};

struct Line

{

Point s,e;

Line() {}

Line(Point \_s,Point \_e)

{

s = \_s;

e = \_e;

}

};

bool onSeg(Point P,Line L)

{

return

((L.s-P)^(L.e-P)) == 0 &&

(P.x-L.s.x)\*(P.x-L.e.x) <= 0 &&

(P.y-L.s.y)\*(P.y-L.e.y) <= 0;

}

int sqdis(Point a,Point b)

{

return (a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y);

}

const int MOD = 10007;

int INV[MOD];

//求ax = 1( mod m) 的x值，就是逆元(0<a<m)

long long inv(long long a,long long m)

{

if(a == 1)return 1;

return inv(m%a,m)\*(m-m/a)%m;

}

struct Matrix

{

int mat[330][330];

void init()

{

memset(mat,0,sizeof(mat));

}

int det(int n)//求行列式的值模上MOD，需要使用逆元

{

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

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

mat[i][j] = (mat[i][j]%MOD+MOD)%MOD;

int res = 1;

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

{

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

if(mat[j][i]!=0)

{

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

swap(mat[i][k],mat[j][k]);

if(i != j)

res = (-res+MOD)%MOD;

break;

}

if(mat[i][i] == 0)

{

res = -1;//不存在(也就是行列式值为0)

break;

}

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

{

//int mut = (mat[j][i]\*INV[mat[i][i]])%MOD;//打表逆元

int mut = (mat[j][i]\*inv(mat[i][i],MOD))%MOD;

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

mat[j][k] = (mat[j][k]-(mat[i][k]\*mut)%MOD+MOD)%MOD;

}

res = (res \* mat[i][i])%MOD;

}

return res;

}

};

Point p[330];

int n,R;

bool check(int k1,int k2)//判断两点的距离小于等于R，而且中间没有点阻隔

{

if(sqdis(p[k1],p[k2]) > R\*R)return false;

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

if(i!=k1 && i!=k2)

if(onSeg(p[i],Line(p[k1],p[k2])))

return false;

return true;

}

int g[330][330];

int main()

{

//预处理逆元

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

INV[i] = inv(i,MOD);

int T;

scanf("%d",&T);

while(T--)

{

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

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

p[i].input();

memset(g,0,sizeof(g));

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

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

if(check(i,j))

g[i][j] = g[j][i] = 1;

Matrix ret;

ret.init();

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

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

if(i != j && g[i][j])

{

ret.mat[i][j] = -1;

ret.mat[i][i]++;

}

printf("%d\n",ret.det(n-1));

}

return 0;

}

### 最小生成树计数（矩阵树）

//HDU 4408

/\*题意是给定n个点，m条边的无向图，求最小生成树的个数对p取模。

用kruscal计算最小生成树时，每次取连接了两个不同连通块的最小的边。也就是先处理d1条c1长度的边，再处理d2条c2长度的边。长度相同的边无论怎么选，最大连通情况都是固定的。 分别对ci长度的边产生的几个连通块计算生成树数量再乘起来，然后把这些连通块缩点，再计算ci+1长度的边。

生成树计数用Matrix-Tree定理，这题的缩点后是会产生重边的，Matrix-tree也适用：

Kirchhoff矩阵任意n-1阶子矩阵的行列式的绝对值就是无向图的生成树的数量。

Kirchhoff矩阵的定义是度数矩阵-邻接矩阵。

1、G的度数矩阵D[G]：n\*n的矩阵，Dii等于Vi的度数，其余为0。

2、G的邻接矩阵A[G]：n\*n的矩阵， Vi、Vj之间有边直接相连，则 Aij=ij之间的边数，否则为0。

并查集fa[i]是当前长度之前，节点所属的联通块，ka[i]是当前长度的边连接后它在的连通块。

\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <vector>

using namespace std;

typedef long long ll;

const int N=101;

const int M=1001;

ll n,m,p,ans;

vector<int>gra[N];

struct edge{

int u,v,w;

}e[M];

int cmp(edge a,edge b){

return a.w<b.w;

}

ll mat[N][N],g[N][N];

ll fa[N],ka[N],vis[N];

ll det(ll c[][N],ll n){

ll i,j,k,t,ret=1;

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

for(j=0;j<n;j++) c[i][j]%=p;

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

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

while(c[j][i]){

t=c[i][i]/c[j][i];

for(k=i; k<n; k++)

c[i][k]=(c[i][k]-c[j][k]\*t)%p;

swap(c[i],c[j]);

ret=-ret;

}

if(c[i][i]==0)

return 0L;

ret=ret\*c[i][i]%p;

}

return (ret+p)%p;

}

ll find(ll a,ll f[]){

return f[a]==a?a:find(f[a],f);

}

void matrix\_tree(){//对当前长度的边连接的每个联通块计算生成树个数

for(int i=0;i<n;i++)if(vis[i]){//当前长度的边连接了i节点

gra[find(i,ka)].push\_back(i);//将i节点压入所属的联通块

vis[i]=0;//一边清空vis数组

}

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

if(gra[i].size()>1){//联通块的点数为1时生成树数量是1

memset(mat,0,sizeof mat);//清空矩阵

int len=gra[i].size();

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

for(int k=j+1;k<len;k++){//构造这个联通块的矩阵（有重边）

int u=gra[i][j],v=gra[i][k];

if(g[u][v]){

mat[k][j]=(mat[j][k]-=g[u][v]);

mat[k][k]+=g[u][v];mat[j][j]+=g[u][v];

}

}

ans=ans\*det(mat,gra[i].size()-1)%p;

for(int j=0;j<len;j++)fa[gra[i][j]]=i;//缩点

}

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

{

gra[i].clear();

ka[i]=fa[i]=find(i,fa);

}

}

int main(){

while(scanf("%lld%lld%lld",&n,&m,&p),n){

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

int u,v,w;

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

u--;v--;

e[i]=(edge){u,v,w};

}

sort(e,e+m,cmp);

memset(g,0,sizeof g);

ans=1;

for(ll i=0;i<n;i++)ka[i]=fa[i]=i;

for(ll i=0;i<=m;i++){//边从小到大加入

if(i&&e[i].w!=e[i-1].w||i==m)//处理完长度为e[i-1].w的所有边

matrix\_tree();//计算生成树

ll u=find(e[i].u,fa),v=find(e[i].v,fa);//连的两个缩点后的点

if(u!=v)//如果不是一个

{

vis[v]=vis[u]=1;

ka[find(u,ka)]=find(v,ka);//两个分量在一个联通块里。

g[u][v]++,g[v][u]++;//邻接矩阵

}

}

int flag=1;

for(int i=1;i<n;i++)if(fa[i]!=fa[i-1])flag=0;

printf("%lld\n",flag?ans%p:0);//注意p可能为1，这样m=0时如果ans不%p就会输出1

}

}

### 最小生成树邻接矩阵prim算法

bool vis[maxn];

int lowc[maxn];

int prim(int cost[][maxn],int n)

{

int ans=0;

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

vis[0]=true;

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

{

lowc[i]=cost[0][i];

}

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

{

int minc=inf;

int p=-1;

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

{

if(!vis[j]&&minc>lowc[j])

{

minc=lowc[j];

p=j;

}

}

if(minc==inf) return -1;

ans+=minc;

vis[p]=true;

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

{

if(!vis[j]&&lowc[j]>cost[p][j])

lowc[j]=cost[p][j];

}

}

return ans;

}

#### 最小生成树总结

变形1：有的边的权值已确定。

直接把确定的边的权值赋值为0，其余不变。

变形2：有的边权不满足条件。

直接把这些边权赋值为inf，其余不变。

变形3：有 N 个点和 M 条边，每条边都有一个权值 V。选取 N-1 条边。将这 N 个点连接起来(即形成一个树)。要求这些边的权值的最大值尽可能的小，在此前提下这些边的权值和尽可能的大。

思路：用kruskal先求一遍最小生成树，记录最后一个加进去的边权(即最小的最大值)，再对小于等于这个边权的边求最大生成树即可。

### 拓扑排序

vector<int> G[maxn];

int in[maxn]; //入度

int topo[maxn], t;

bool topoSort()

{

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

t = 0; //优先队列保证最小序列

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

{

if(in[i] == 0) q.push(i);

}

int cnt = 0;

while(!q.empty())

{

int u = q.top();

q.pop();

topo[t++] = u;

++cnt;

for(int v = 0; v < G[u].size(); ++v)

{

if(--in[G[u][v]] == 0) q.push(G[u][v]); //找入度为0的边入队

}

}

if(cnt != n) return 0; //有环

return 1;

}

### DAG最长路

#include<bits/stdc++.h>

#define ll long long

using namespace std;

#define inf 0x3f3f3f3f

const int N=1005;

struct node

{

int to,val;

};

int dp[N];

vector<node> g[N\*N/2];

int dfs(ll i){

if(dp[i]!=0) return dp[i];

int sz=g[i].size();

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

dp[i]=max(dp[i],dfs(g[i][j].to)+g[i][j].val);

}

return dp[i];

}

int main()

{

int n,t,m;

scanf("%d",&t);

while(t--)

{

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

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

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

{

int x,y,z;

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

g[x].push\_back({y,z});

}

int ma=0;

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

{

ma=max(ma,dfs(i));

}

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

for(int i=0;i<=n;i++) g[i].clear();

}

}

/\*

//打印路径：

int DP(int i){

if(dp[i]>0) return dp[i];

for(int j=0;j<n;j++){ //遍历i的所有可达出边

if(map[i][j]!=Inf){

int temp=DP(j)+map[i][j];//单独计算dp

if(dp[i]<temp){//可以获得更长的路径

dp[i]=temp;

next[i]=j; //保存i的后继顶点j

}

}

}

return dp[i];

}

//调用前需先获得最大的dp[i],然后将i作为路径的起点传入

void printPath(int i){

cout<<i;

while(next[i]!=-1){//next数组初始化为-1

i=next[i];

cout<<"->"<<i;

}

}

\*/

# 字符串

## 基本知识点

ASCII: A~Z:65~90,a~z:97~122

1、strcmp(str1,str2):

如果返回值 < 0，则表示 str1 小于 str2。

如果返回值 > 0，则表示 str2 小于 str1。

如果返回值 = 0，则表示 str1 等于 str2。

2、string 类大写转小写：transform(strA.begin(), strA.end(), strA.begin(), ::tolower);

string类小写转大写：transform(strA.begin(), strA.end(), strA.begin(), ::toupper);

## KMP

### 判断pattern是否是text的子串，并返回匹配的第一个位置

//All integers are in the range of [-1000000, 1000000].

#include <iostream>

#include<cstdio>

#include<cstring>

using namespace std;

const int maxn=1000005;

int next1[maxn];

int text[maxn],pattern[maxn];

int ans;

void getnext(int s[],int len)

{

int j=-1;

next1[0]=-1;

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

{

while(j!=-1&&s[i]!=s[j+1])

j=next1[j];//不断执行此循环，知道j回退到-1，或是s[i]==s[j+1]

if(s[i]==s[j+1])

j++;

next1[i]=j;

}

}

//kmp算法，判断pattern是否是text的子串

bool kmp(int text1[],int pattern1[],int n,int m)

{

// int n=strlen(text),m=strlen(pattern);

getnext(pattern1,m);//计算pattern的子串

int j=-1;//初始化为-1，表示当前还没有一位被匹配

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

{

while(j!=-1&&text1[i]!=pattern1[j+1])

{

j=next1[j];

}

if(text1[i]==pattern1[j+1])//当第一次执行该循环？不是这样的，后面可能不匹配

{

ans=i;

j++;

}

if(j==m-1)

return true;

}

return false;

}

int main()

{

int T;

scanf("%d",&T);

int m,n;

while(T--)

{

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

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

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

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

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

if(kmp(text,pattern,n,m))

cout<<ans-m+2<<endl;//ans-m+2代表成功匹配的初始位置

else

cout<<"-1"<<endl;

ans=0;

}

return 0;

}

kmp中的next数组求最小循环节的应用

//例如abcabcabc 3; aaaa 4; abs 1

#include<iostream>

#include<cstdio>

#include<cstring>

using namespace std;

const int maxn=1000005;

int next1[maxn];

void getnext(char \*s)

{

int mp=strlen(s);

int j=0,k=-1;

next1[0]=-1;

while(j<mp)

{

if(k==-1||s[j]==s[k])

next1[++j]=++k;

else

k=next1[k];

}

}

char s[maxn];

int main()

{

while(gets(s))

{

if(strcmp(s,".")==0)

break;

else

{

getnext(s);

int mp=strlen(s);

int ans=1;

if(mp%(mp-next1[mp])==0)

ans=mp/(mp-next1[mp]);

cout<<ans<<endl;

}

}

}

KMP求周期性前缀 输出：i 前缀

#include <iostream>

#include <stdio.h>

using namespace std;

char a[1000010];

int next1[1000010];

int n;

void GetNext()

{

int j=0,k=-1;

next1[0]=-1;

while(j<n)

{

if(k==-1||a[j]==a[k])

next1[++j]=++k;

else

k=next1[k];

}

}

void DisRes(int num)

{

int j;

printf("Test case #%d\n",num);

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

{

if(next1[i]==-1 || next1[i]==0) //next[i]是-1或0的忽略，说明之前没有周期性前缀

continue;

j = i - next1[i];

if(i%j==0) //能整除，说明存在周期性前缀

printf("%d %d\n",i,i/j); //输出这个前缀的长度和周期数

}

printf("\n");

}

int main()

{

int num = 0;

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

{

if(n==0) break;

scanf("%s",a);

GetNext(); //获得next[]数组

DisRes(++num); //输出结果

}

return 0;

}

### KMP之剪花布条

/\*HDU2087

花纹条和小饰条不会超过1000个字符长。如果遇见#字符，则不再进行工作。

Sample Input

abcde a3

aaaaaa aa

#

Sample Output

0

3\*/

#include <iostream>

#include<cstring>

#include<cstdio>

using namespace std;

const int maxn=1005;

int next1[maxn];

char text1[maxn],pattern[maxn];

void getnext(char s[])

{

next1[0]=-1;

int j=0,k=-1;

int len=strlen(s);

while(j<len-1)

{

if(k==-1||s[j]==s[k])

next1[++j]=++k;

else

k=next1[k];

}

}

int main()

{

while(scanf("%s",text1)&&text1[0]!='#')

{

scanf("%s",pattern);

getnext(pattern);

int len1=strlen(pattern),len2=strlen(text1);

int i=0,j=0;

int k=0;

while(i<len2)

{

if(j==-1||text1[i]==pattern[j])

{

i++;

j++;

}

else

j=next1[j];

if(j==len1)

{

k++;

j=0;

}

}

cout<<k<<endl;

}

return 0;

}

### Manacher之最长回文第一种写法

/\*

\* 求最长回文子串

\*/

#include<iostream>

#include<string>

#include<cstdio>

using namespace std;

const int MAXN=1e6+10;

char Ma[MAXN\*2];

int Mp[MAXN\*2];

void Manacher(string s,int len)

{

int l=0;

Ma[l++]='$';

Ma[l++]='#';

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

{

Ma[l++]=s[i];

Ma[l++]='#';

}

Ma[l]=0;

int mx=0,id=0;

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

{

Mp[i]=mx>i?min(Mp[2\*id-i],mx-i):1;

while(Ma[i+Mp[i]]==Ma[i-Mp[i]])Mp[i]++;

if(i+Mp[i]>mx)

{

mx=i+Mp[i];

id=i;

}

}

}

/\*

\* abaaba

\* i: 0 1 2 3 4 5 6 7 8 9 10 11 12 13

\* Ma[i]: $ # a # b # a # a # b # a #

\* Mp[i]: 1 1 2 1 4 1 2 7 2 1 4 1 2 1

\*/

int main()

{

int cas=0;

string s;

while(getline(cin,s))

{

if(s=="END")

break;

int len=s.length();

Manacher(s,len);

int ans=0;

for(int i=0; i<2\*len+2; i++)

ans=max(ans,Mp[i]-1);

printf("Case %d: %d\n",++cas,ans);

}

return 0;

}

### Manacher之最长回文第二种写法

#include <stdio.h>

#include <stdlib.h>

#include<math.h>

#include<string.h>

#define min(a,b) a<b?a:b

char str[220005],s[220005];

int len,p[220005];

void solve()

{

int i,mx ,id;

mx=0;

for( i=1;s[i]!='\0';i++)

{

p[i]=mx>i?min(p[2\*id-i],mx-i):1;

while(s[i+p[i]]==s[i-p[i]])

p[i]++;

if(i+p[i]>mx)

{

mx=i+p[i];

id=i;

}

}

}

void init()

{

s[0]='$';

s[1]='#';

int i;

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

{

s[i\*2+2]=str[i];

s[i\*2+3]='#';

}

len=len\*2+2;

s[len]='#';

}

int main()

{

int i;

while(scanf("%s",str)!=EOF)

{

len=strlen(str);

init();

solve();

int ans=0;

for( i=1;i<len;i++)

{

if(ans<p[i])

{

ans=p[i];

}

}

printf("%d\n",ans-1);

}

return 0;

}

### Manacher改编---》求字符串环的最大回文长度

可以把这个字符串最前面的某一段连续的字符(不改变顺序)移动到原先字符串的末尾。那么请问小A通过这样的操作之后(也可以选择不移动)能够得到最大回文子串的长度是多少。

#include<bits/stdc++.h>

**using** **namespace** std;

**const** **int** MAXN=1e4+5;

**char** Ma[MAXN\*2];

**int** Mp[MAXN\*2];

**void** Manacher(**char**\* s,**int** len)

{

**int** l=0;

    Ma[l++]='$';

    Ma[l++]='#';

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

    {

        Ma[l++]=s[i];

        Ma[l++]='#';

    }

    Ma[l]=0;

**int** mx=0,id=0;

**for**(**int** i=0; i<l; i++)

    {

        Mp[i]=mx>i?min(Mp[2\*id-i],mx-i):1;

**while**(Ma[i+Mp[i]]==Ma[i-Mp[i]])Mp[i]++;

**if**(i+Mp[i]>mx)

        {

            mx=i+Mp[i];

            id=i;

        }

    }

}

**int** main()

{

**int** cas=0;

**char** s[MAXN];

**while**(cin>>s)

    {

**int** len=**strlen**(s),ans=0;

        Manacher(s,len);

**for**(**int** i=0; i<2\*len+2; i++)

            ans=max(ans,Mp[i]-1);

**for**(**int** j=0; j<len; j++)

        {

            s[len+j]=s[j];

            Manacher(s+j+1,len);

**for**(**int** i=0; i<2\*len+2; i++)

                ans=max(ans,Mp[i]-1);

        }

        cout<<ans<<endl;

    }

**return** 0;

}

### 找子串问题

这个找字串的问题，题目大概意思就是找出所有字符串中共同拥有的一个子串，

该子串（正、逆字符）是任何一个母串的子串，求该子串的最长长度。

方法：首先找到最短的那个字符串，然后对它进行反转，再从大到小的长度对其余输入的字符串进行查找，for循环写的很巧妙

/\*Sample Input

2

3

ABCD

BCDFF

BRCD

2

rose

orchid

Sample Output

2

2\*/

#include <iostream>

#include<string>

#include<cstdio>

#include<algorithm>

using namespace std;

const int maxn=105;

int main()

{

int T;

cin>>T;

while(T--)

{

int n;

scanf("%d",&n);

string s[maxn];

int len=105;

int pos=0;

int ma=0;

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

{

cin>>s[i];

if(len>s[i].length())

{

len=s[i].length();

pos=i;

}

}

for(int i=s[pos].length(); i>=0; i--)

{

for(int j=0; j<s[pos].length()-i+1; j++)

{

string s1,s2;

s1=s[pos].substr(j,i);

s2=s1;

reverse(s2.begin(),s2.end());

int t;

for( t=0; t<n; t++)

{

if(s[t].find(s1,0)==-1&&s[t].find(s2,0)==-1)

break;

}

if(t==n&&ma<s1.length())

ma=s1.length();

}

}

cout<<ma<<endl;

}

return 0;

}

## AC自动机

### 1.求目标串出现了多少个模式串

#include<bits/stdc++.h>

using namespace std;

struct Tree//字典树

{

int fail;//失配指针

int vis[26];//子节点的位置

int end;//标记有几个单词以这个节点结尾

}AC[1000000];//Trie树

int cnt=0;//Trie的指针

inline void Build(string s)

{

int l=s.length();

int now=0;//字典树的当前指针

for(int i=0;i<l;++i)//构造Trie树

{

if(AC[now].vis[s[i]-'a']==0)//Trie树没有这个子节点

AC[now].vis[s[i]-'a']=++cnt;//构造出来

now=AC[now].vis[s[i]-'a'];//向下构造

}

AC[now].end+=1;//标记单词结尾

}

void Get\_fail()//构造fail指针

{

queue<int> Q;//队列

for(int i=0;i<26;++i)//第二层的fail指针提前处理一下

{

if(AC[0].vis[i]!=0)

{

AC[AC[0].vis[i]].fail=0;//指向根节点

Q.push(AC[0].vis[i]);//压入队列

}

}

while(!Q.empty())//BFS求fail指针

{

int u=Q.front();

Q.pop();

for(int i=0;i<26;++i)//枚举所有子节点

{

if(AC[u].vis[i]!=0)//存在这个子节点

{

AC[AC[u].vis[i]].fail=AC[AC[u].fail].vis[i];

//子节点的fail指针指向当前节点的

//fail指针所指向的节点的相同子节点

Q.push(AC[u].vis[i]);//压入队列

}

else//不存在这个子节点

AC[u].vis[i]=AC[AC[u].fail].vis[i];

//当前节点的这个子节点指向当

//前节点fail指针的这个子节点

}

}

}

int AC\_Query(string s)//AC自动机匹配

{

int l=s.length();

int now=0,ans=0;

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

{

now=AC[now].vis[s[i]-'a'];//向下一层

for(int t=now;t&&AC[t].end!=-1;t=AC[t].fail)//循环求解

{

ans+=AC[t].end;

AC[t].end=-1;

}

}

return ans;

}

int main()

{

int n;

string s;

cin>>n;

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

{

cin>>s;

Build(s);

}

AC[0].fail=0;//结束标志

Get\_fail();//求出失配指针

cin>>s;//文本串

cout<<AC\_Query(s)<<endl;

return 0;

}

### 2.有N个由小写字母组成的模式串以及一个文本串T。每个模式串可能会在文本串中出现多次。你需要找出哪些模式串在文本串T中出现的次数最多。

#include<iostream>

#include<cstdio>

#include<cstdlib>

#include<cstring>

#include<cmath>

#include<queue>

#include<algorithm>

using namespace std;

struct Tree//字典树

{

int fail;//失配指针

int vis[26];//子节点的位置

int end;//标记以这个节点结尾的单词编号

}AC[100000];//Trie树

int cnt=0;//Trie的指针

struct Result

{

int num;

int pos;

}Ans[100000];//所有单词的出现次数

bool operator <(Result a,Result b)

{

if(a.num!=b.num)

return a.num>b.num;

else

return a.pos<b.pos;

}

string s[100000];

inline void Clean(int x)

{

memset(AC[x].vis,0,sizeof(AC[x].vis));

AC[x].fail=0;

AC[x].end=0;

}

inline void Build(string s,int Num)

{

int l=s.length();

int now=0;//字典树的当前指针

for(int i=0;i<l;++i)//构造Trie树

{

if(AC[now].vis[s[i]-'a']==0)//Trie树没有这个子节点

{

AC[now].vis[s[i]-'a']=++cnt;//构造出来

Clean(cnt);

}

now=AC[now].vis[s[i]-'a'];//向下构造

}

AC[now].end=Num;//标记单词结尾

}

void Get\_fail()//构造fail指针

{

queue<int> Q;//队列

for(int i=0;i<26;++i)//第二层的fail指针提前处理一下

{

if(AC[0].vis[i]!=0)

{

AC[AC[0].vis[i]].fail=0;//指向根节点

Q.push(AC[0].vis[i]);//压入队列

}

}

while(!Q.empty())//BFS求fail指针

{

int u=Q.front();

Q.pop();

for(int i=0;i<26;++i)//枚举所有子节点

{

if(AC[u].vis[i]!=0)//存在这个子节点

{

AC[AC[u].vis[i]].fail=AC[AC[u].fail].vis[i];

//子节点的fail指针指向当前节点的

//fail指针所指向的节点的相同子节点

Q.push(AC[u].vis[i]);//压入队列

}

else//不存在这个子节点

AC[u].vis[i]=AC[AC[u].fail].vis[i];

//当前节点的这个子节点指向当

//前节点fail指针的这个子节点

}

}

}

int AC\_Query(string s)//AC自动机匹配

{

int l=s.length();

int now=0,ans=0;

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

{

now=AC[now].vis[s[i]-'a'];//向下一层

for(int t=now;t;t=AC[t].fail)//循环求解

Ans[AC[t].end].num++;

}

return ans;

}

int main()

{

int n;

while(233)

{

cin>>n;

if(n==0)break;

cnt=0;

Clean(0);

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

{

cin>>s[i];

Ans[i].num=0;

Ans[i].pos=i;

Build(s[i],i);

}

AC[0].fail=0;//结束标志

Get\_fail();//求出失配指针

cin>>s[0];//文本串

AC\_Query(s[0]);

sort(&Ans[1],&Ans[n+1]);

cout<<Ans[1].num<<endl;

cout<<s[Ans[1].pos]<<endl;

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

{

if(Ans[i].num==Ans[i-1].num)

cout<<s[Ans[i].pos]<<endl;

else

break;

}

}

return 0;

}

# RMQ

## RMQ板子

输入中第一行有两个数m,n表示有m(m<=100000)笔账,n表示有n个问题，n<=100000。

后面n行分别是n个问题,每行有2个数字说明开始结束的账目编号。

问在a到b号账中最少的一笔是多少？

输入样例#1：

10 3

1 2 3 4 5 6 7 8 9 10

2 7

3 9

1 10

输出样例#1：

2 3 1

#include <cstdio>

#include <iostream>

#include <cstring>

#include <cmath>

#include <algorithm>

using namespace std;

const int size =1e5+10;

int a[size];

int f[size][40];

void init\_rmq(int m) {

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

f[i][0]=a[i];

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

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

if(i+(1<<j)-1 <= m) {

f[i][j]=min(f[i][j-1],f[i+(1<<(j-1))][j-1]);

}

}

}

}

int query\_rmq(int l,int r) {

int i=0;

for(;(1<<i)+l-1<=r;i++){};

i--;

return min(f[l][i],f[r-(1<<i)+1][i]);

}

int main() {

int m,n;

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

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

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

init\_rmq(m);

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

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

printf("%d ",query\_rmq(l,r));

}

return 0;

}

## 一维RMQ

/\*

O(nlogn)预处理 O(1)查询

POJ 3264 查询区间最大值与最小值的差

\*/

#include<cstdio>

#include<algorithm>

using namespace std;

const int maxn=50010;

int mm[maxn],dpMax[maxn][20],dpMin[maxn][20];

//初始化Rmq,b数组下标从1开始，从0开始简单修改

void initRmq(int n,int b[]) //O(nlogn)预处理

{

mm[0]=-1;

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

{

mm[i]=((i&(i-1))==0)?mm[i-1]+1:mm[i-1];

dpMax[i][0]=b[i];

dpMin[i][0]=b[i];

}

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

{

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

{

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

dpMin[i][j]=min(dpMin[i][j-1],dpMin[i+(1<<(j-1))][j-1]);

}

}

}

//查询最大值

int rmqMax(int x,int y) //O(1)查询

{

int k=mm[y-x+1];

return max(dpMax[x][k],dpMax[y-(1<<k)+1][k]);

}

int rmqMin(int x,int y)

{

int k=mm[y-x+1];

return min(dpMin[x][k],dpMin[y-(1<<k)+1][k]);

}

int main()

{

int n,q;

while(~scanf("%d%d",&n,&q))

{

int b[maxn];

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

{

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

}

initRmq(n,b);

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

{

int a,b;

scanf("%d%d",&a,&b);

printf("%d\n",rmqMax(a,b)-rmqMin(a,b));

}

}

return 0;

}

## 二维RMQ

/\*

HDU 2888

题意：给定一个n \* m的矩阵，再给定q个询问，每次询问(r1，c1)为左上角，(r2,c2)为右下角的子矩形的最大值，并且判断该最大值是否出现在了这个子矩阵的4个顶角上？

二维 RMQ，预处理复杂度 n\*m\*log\*(n)\*log(m)

数组下标从 1 开始

\*/

#include <stdio.h>

#include <algorithm>

#include <iostream>

#include <string.h>

using namespace std;

int val[310][310];

int dpMax[310][310][9][9];//最大值

//int dpMin[310][310][9][9];//最小值

int mm[310];//二进制位数减一，使用前初始化

void initRmq(int n,int m)

{

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

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

{

dpMax[i][j][0][0] = val[i][j];

// dpMin[i][j][0][0] = val[i][j];

}

for(int ii = 0; ii <= mm[n]; ii++)

for(int jj = 0; jj <= mm[m]; jj++)

if(ii+jj)

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

for(int j = 1; j + (1<<jj) - 1 <= m; j++)

{

if(ii)

{

dpMax[i][j][ii][jj] = max(dpMax[i][j][ii-1][jj],dpMax[i+(1<<(ii-1))][j][ii-1][jj]);

// dpMin[i][j][ii][jj] = min(dpMin[i][j][ii-1][jj],dpMin[i+(1<<(ii-1))][j][ii-1][jj]);

}

else

{

dpMax[i][j][ii][jj] = max(dpMax[i][j][ii][jj-1],dpMax[i][j+(1<<(jj-1))][ii][jj-1]);

// dpMin[i][j][ii][jj] = min(dpMin[i][j][ii][jj-1],dpMin[i][j+(1<<(jj-1))][ii][jj-1]);

}

}

}

//查询矩形内的最大值(x1<=x2,y1<=y2)

int rmqMax(int x1,int y1,int x2,int y2)

{

int k1 = mm[x2-x1+1];

int k2 = mm[y2-y1+1];

x2 = x2 - (1<<k1) + 1;

y2 = y2 - (1<<k2) + 1;

return max(max(dpMax[x1][y1][k1][k2],dpMax[x1][y2][k1][k2]),max(dpMax[x2][y1][k1][k2],dpMax[x2][y2][k1][k2]));

}

/\*int rmqMin(int x1,int y1,int x2,int y2)

{

int k1 = mm[x2-x1+1];

int k2 = mm[y2-y1+1];

x2 = x2 - (1<<k1) + 1;

y2 = y2 - (1<<k2) + 1;

return min(min(dpMin[x1][y1][k1][k2],dpMin[x1][y2][k1][k2]),min(dpMin[x2][y1][k1][k2],dpMin[x2][y2][k1][k2]));

}\*/

int main()

{

//在外面对mm数组进行初始化

mm[0] = -1;

for(int i = 1; i <= 305; i++)

mm[i] = ((i&(i-1))==0)?mm[i-1]+1:mm[i-1];

int n,m;//行和列

int Q;//查询次数

int r1,c1,r2,c2;//左上角，右下角

while(scanf("%d%d",&n,&m) == 2)

{

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

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

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

initRmq(n,m);

scanf("%d",&Q);

while(Q--)

{

scanf("%d%d%d%d",&r1,&c1,&r2,&c2);

if(r1 > r2)swap(r1,r2);

if(c1 > c2)swap(c1,c2);

int tmp = rmqMax(r1,c1,r2,c2);

printf("%d ",tmp);

if(tmp == val[r1][c1] || tmp == val[r1][c2] || tmp == val[r2][c1] || tmp == val[r2][c2])

printf("yes\n");

else printf("no\n");

}

}

return 0;

}

# 技巧题

## 二分

### 、二分模板

while (l<=r)

{

int mid=(l+r)/2;

if (check(mid))

{

ans=mid;

r=mid-1;

}

else l=mid+1;

}

printf("%d",ans);

### 、二分应用

/\*小Q会告诉你他的算法的时间复杂度为O(n^a(logn)^b)，且蕴含在这个大O记号下的常数为1。同时，小Q还会告诉你评测机在规定时限内可以执行k条指令。小Q认为只要n^a(⌈log2n⌉)^b不超过k，那么就是合理的数据范围。其中，⌈x⌉表示最小的不小于x的正整数，即x上取整。

自然，小Q希望题目的数据范围n越大越好，他希望你写一个程序帮助他设置最大的数据范围。

Input

第一行包含一个正整数T(1≤T≤1000)，表示测试数据的组数。

每组数据包含一行三个正整数a,b,k(1≤a,b≤10,106≤k≤1018)，分别描述时间复杂度以及允许的指令数。

Output

对于每组数据，输出一行一个正整数n，即最大可能的n。

Sample Input

3

1 1 100000000

2 1 100000000

1 3 200000000

Sample Output

4347826

2886

48828

\*/

#include<bits/stdc++.h>

typedef long long ll;

using namespace std;

ll a,b,k;

ll s[100]={1};

int check(ll n)

{

int j=0;

for(j=0;j<=62;j++)

if(n<=s[j])break;

long double t=k;

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

{

t=t/(long double)j;

if(t<1.0)return 0;

}

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

{

t=t/(long double)n;

if(t<1.0)return 0;

}

return 1;

}

int main()

{

int T;

scanf("%d",&T);

for(int i=1;i<=62;i++)

{

s[i]=ll(1ll<<i);//1ll是把1转化为longlong,因为1是int型

// cout<<s[i]<<endl;//s[i]存的即是2的i次方

}

while(T--)

{

scanf("%lld%lld%lld",&a,&b,&k);

ll l=0,r=1e18,mid;

ll ans=0;

while(l<=r)

{

mid=(l+r)/2;

if(check(mid))

{

ans=mid;

l=mid+1;

}

else r=mid-1;

}

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

/\*if(check(r))printf("%lld\n",r);

else printf("%lld\n",l);\*/

}

return 0;

}

## 异或前缀和

//Codeforces Round #539C (Div. 2)

题意：给你n个数，求有多少长度为偶数的区间满足a[l]^…a[mid]=a[mid+1]^a[r]。

思路：例如1 2 3 4 5，预处理异或前缀和得到1 3 0 4 1。可以发现[2,5]是满足条件的区间，异或到a[5]时又出现了1说明a[2]…a[5]=0，即a[2]^a[3]=a[4]^a[5]。保证区间长度为偶数只需要保证每次出现的值的位次的奇偶性相同即可，后面异或到的数只要出现过，ans就加上这个数前面出现的次数(此位置可以和前面每个出现此数的位置构成一个偶区间)。

#include<bits/stdc++.h>

using namespace std;

#define ll long long

const int maxn=3e5+5;

ll a[maxn],b[(1<<20)+5][2];

int main()

{

std::ios::sync\_with\_stdio(false);

int n;

cin>>n;

ll ans=0;

b[0][0]=1;

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

{

cin>>a[i];

a[i]^=a[i-1];

ans+=b[a[i]][i&1];

b[a[i]][i&1]++;

}

cout<<ans<<endl;

return 0;

}

## 贪心题

1.洛谷p1016

题意：一个旅行家想驾驶汽车以最少的费用从一个城市到另一个城市（假设出发时油箱是空的）。给定两个城市之间的距离D1、汽车油箱的容量C（以升为单位）、每升汽油能行驶的距离D2、出发点每升汽油价格P和沿途油站数N（N可以为零），油站ii离出发点的距离Di、每升汽油价格Pi（i=1,2,…,Ni=1,2,…,N）。计算结果四舍五入至小数点后两位。如果无法到达目的地，则输出“No Solution”。

思路：找距离当前可达的点的最便宜的点，如果这个点比当前的点价格还高，那就在这一站加满油;如果这个点比当前价格低，那就只加够到那个点的油。注意这两种情况都要考虑是不是可以直接到目的地了。

#include<bits/stdc++.h>

using namespace std;

#define inf 0x3f3f3f3f

#define ll long long

const int maxn=10;

struct node

{

double price,dis;

bool operator<(node b)

{

return dis<b.dis;

}

};

int main()

{

std::ios::sync\_with\_stdio(false);

double d1,c,d2,p;

int n;

cin>>d1>>c>>d2>>p>>n;

node g[maxn];

g[0].price=p,g[0].dis=d2;

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

{

cin>>g[i].dis>>g[i].price;

}

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

double ans=0;

double nowd=0,nowc=0,nowp=p;

int gg=1;

while(1)

{

int flag=0;

double mi=inf,md=0;

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

{

if(g[i].dis>nowd&&g[i].dis-nowd<=c\*d2&&g[i].price<mi)

{

mi=g[i].price;

md=g[i].dis;

flag=1;

}

}

if(d1-nowd>c\*d2&&!flag)

{

cout<<"No Solution"<<endl;

gg=0;

break;

}

else if(flag)

{

if(mi>nowp)

{

if(c\*d2>=d1-nowd)

{

if(nowc<(d1-nowd)/d2)

ans+=((d1-nowd)/d2-nowc)\*nowp;

break;

}

ans+=(nowp\*(c-nowc));

nowc=c-(md-nowd)/d2;

nowd=md;

nowp=mi;

}

else

{

ans+=(nowp\*(md-nowd)/d2);

nowc=0;

nowd=md;

nowp=mi;

}

}

else if(nowd-d1<=nowc\*d2)

{

if(nowc<(d1-nowd)/d2)

ans+=((d1-nowd)/d2-nowc)\*nowp;

break;

}

}

if(gg)

cout<<fixed<<setprecision(2)<<ans<<endl;

return 0;

}

## 任意进制数判断回文

//洛谷p1015

例如：给定一个十进制数56，将56加65（即把56从右向左读），得到121是一个回文数。

又如：对于十进制数87：

STEP1：87+78=165

STEP2：165+561=726

STEP3：726+627=1353

STEP4：1353+3531=4884

在这里的一步是指进行了一次N进制的加法，上例最少用了4步得到回文数4884。

写一个程序，给定一个N进制数M，求最少经过几步可以得到回文数。如果在30步以内（包含30步）不可能得到回文数，则输出Impossible!

#include<bits/stdc++.h>

using namespace std;

#define ll long long

ll now;

ll n;

string m;

ll d(char c)

{

if(c>='0'&&c<='9')

{

return c-'0';

}

else

return c-'A'+10;

}

bool solve(ll a)

{

ll t=0;

for(ll i=a;i>0;i/=n)

{

t=t\*n+i%n;

}

now=t+a;

return a==t;

}

int main()

{

std::ios::sync\_with\_stdio(false);

while(cin>>n>>m)

{

now=0;

int l=m.length()-1;

for(int i=l;i>=0;i--)

{

now=now\*n+d(m[i]);

}

int flag=0;

for(int i=0;i<=30;i++)

{

if(solve(now))

{

flag=1;

cout<<"STEP="<<i<<endl;

break;

}

}

if(!flag)

cout<<"Impossible!"<<endl;

}

return 0;

}

## Dfs

### 数拆成幂次和//洛谷p1010

任何一个正整数都可以用2的幂次方表示，同时约定方次用括号来表示，即a^b可表示为a(b)。如137=2(2(2)+2+2(0))+2(2+2(0))+2(0)

#include<bits/stdc++.h>

using namespace std;

void dfs(int n)

{

if(n)

{

int i=0;

while(pow(2,i)<=n) i++;

i--;

n-=pow(2,i);

cout<<2;

if(i!=1)

cout<<"(";

if(i==0||i==2)

cout<<i<<")";

else if(i!=1)

{

dfs(i);

cout<<")";

}

if(n)

{

cout<<"+";

dfs(n);

}

}

}

int main()

{

std::ios::sync\_with\_stdio(false);

int n;

cin>>n;

dfs(n);

return 0;

}

### 求n个字符串环的最长公共子串

题意：给n个首位连接的字符串，找这n个字符串环中的最长公共子串。

思路：因为数据量和字符串长度很小，可以用dfs枚举出每个字符串的所有子串，用set保存。至于字符串环的处理，考虑每次枚举出的字符串可以从0到sz（此字符串的长度）枚举分成两半，再把后面的部分加到前面即可构造出所有的环形字符串。

int sz=t.length();

        for(int j=0;j<sz;j++)

        {

            st[x].insert(t.substr(j,sz-j)+t.substr(0,j));

        }

此地方是这个代码的精髓所在我认为。太巧妙了，这样就可以充分考虑到是个环，也能充分插入以每一个开头的字符串了。

#include <iostream>

#include<bits/stdc++.h>

using namespace std;

int n,len;

set<string> st[12];

string s,t;

void dfs(int x, int l)

{

if(l==len)

{

if(t=="")

return;

int sz=t.length();

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

{

st[x].insert(t.substr(j,sz-j)+t.substr(0,j));

}

return ;

}

t+=s[l];

dfs(x,l+1);//要选l开头的

t.erase(t.end()-1);

dfs(x,l+1);//不选l开头的

}

int main()

{

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

{

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

{

st[i].clear();

}

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

{

cin>>s;

len=s.length();

t="";

dfs(i,0);

}

string ans;

set<string>::iterator it=st[0].begin();

for(;it!=st[0].end();it++)

{

int flag=0;

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

{

// if(!st[j].count(\*it))

if(st[j].find(\*it)==st[j].end())

{

flag=1;

break;

}

}

if(!flag)

{

if((\*it).length()>ans.length()) ans=\*it;

else if((\*it).length()==ans.length()&&(\*it)<ans) ans=\*it;

}

}

if(ans=="")

cout<<0<<endl;

else

cout<<ans<<endl;

}

return 0;

}

### 小字辈

\*L2-2 小字辈 （25 分）

本题给定一个庞大家族的家谱，要请你给出最小一辈的名单。

输入格式：

输入在第一行给出家族人口总数 N（不超过 100 000 的正整数） —— 简单起见，我们把家族成员从 1 到 N 编号。

随后第二行给出 N 个编号，其中第 i 个编号对应第 i 位成员的父/母。家谱中辈分最高的老祖宗对应的父/母编号为 -1。

一行中的数字间以空格分隔。

输出格式：

首先输出最小的辈分（老祖宗的辈分为 1，以下逐级递增）。然后在第二行按递增顺序输出辈分最小的成员的编号。

编号间以一个空格分隔，行首尾不得有多余空格。

输入样例：

9

2 6 5 5 -1 5 6 4 7

输出样例：

4

1 9\*/

#include <bits/stdc++.h>

using namespace std;

const int N=1e5+5;

int root;

vector<int> g[N];

int dep[N];

void dfs(int u,int fa)

{

if(fa!=-1)

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

int sz=g[u].size();

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

{

dfs(g[u][i],u);

}

}

vector<int> ans;

int main()

{

int n;

cin>>n;

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

{

int c;

cin>>c;

if(c==-1)

{

root=i;

}

else

{

g[c].push\_back(i);//父亲的孩子push

}

}

dep[root]=1;

dfs(root,-1);

int ma=0;

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

{

ma=max(ma,dep[i]);

}

cout<<ma<<endl;

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

{

if(ma==dep[i])

{

ans.push\_back(i);

}

}

int sz=ans.size();

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

{

cout<<ans[i];

if(i!=sz-1)

cout<<" ";

}

return 0;

}

### 背包

/\*问题：有n件物品，每件物品的重量为w[i]，价值为c[i],现在需要选出若干件物品放入一个容器为V的背包中，

使得在选入背包的物品重量和不超过容量V的前提下，让背包中物品的价值之和最大，求最大价值（1<=n<=20）\*/

/\*如果选择不放入index号物品，那么sumM,sumC就不变，接下来处理index+1好物品，即前往DFS(index+1,sumM,sumC)这条分支，

如果选择放入index号物品，那么sumM+w[index],sumC+c[index],即前往DFS(index+1,sumM++w[index],sumC+c[index])\*/

#include <iostream>

#include<cstdio>

using namespace std;

const int maxn=30;

int n,v,maxvalue=0;//物品件数，背包容量v，最大价值maxvalue

int w[maxn],c[maxn];//w[i]为每件物品的重量，c[i]为每件物品的价值

//DFS,index为当前处理的物品编号、

//sumW,和sumC分别为当前总重量和当前总价值

void DFS(int index,int sumM,int sumC)

{

if(index==n)

{

return;//已经完成对n件物品的选择

}

DFS(index+1,sumM,sumC);//不选第index件物品

if(sumM+w[index]<=v)//只有加入第index件物品后未超过容量V，才能继续

{

if(sumC+c[index]>maxvalue)

{

maxvalue=sumC+c[index];

}

DFS(index+1,sumM+w[index],sumC+c[index]);//选第index件物品

}

}

int main()

{

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

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

{

scanf("%d",&w[i]);//每件物品的重量

}

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

{

scanf("%d",&c[i]);//每件物品的价值

}

DFS(0,0,0);//初始时为第0件物品，当前总重量和总价值为0

cout<<maxvalue<<endl;

return 0；

}

k个数之和为x,且平方和最大

\*问题：给定N个整数（可能有负数），从中选取k个数，使得这k个数之和恰好等于一个给定的整数x，如果有多种方案，

选择它们中元素平方和最大的一个。数据保证这样的方案唯一。例如，从4个整数{2，3，3，4}中选择两个数，使它们的和为6，

显然有两种方案{2，4}与{3，3}，其中平方和最大的方案为{2，4}\*/

#include <iostream>

#include<cstdio>

#include<vector>

using namespace std;

const int maxn=20;

int n,k,x,maxsumsqu=-1,A[maxn];//序列A中n个数选取k个数使得和为x,最大平方和为maxsumsqu

vector<int> temp,ans;//temp存放临时方案，ans存放平方和最大的方案

//当前处理index号整数，当前已选整数个数为nowK

//当前已选整数之和为sum,当前已选整数平方和为sumsqu;

void DFS(int index,int nowK,int sum,int sumsqu)

{

if(nowK==k&&sum==x)//找到k个数的和为x

{

if(sumsqu>maxsumsqu)

{

maxsumsqu=sumsqu;

ans=temp;//更新最优方案

}

return ;

}

//已经处理完n个数，或者超过k个数，或者和超过x,返回

if(index==n||nowK>k||sum>x)

return ;

/\*当试图进入“选index号数”这条分支时，就把A[index]加入temp中，当这条分支结束时，就把它从temp中去除，使它不会影响“不选index号数”这个分支\*/

//选index号数

temp.push\_back(A[index]);

DFS(index+1,nowK+1,sum+A[index],sumsqu+A[index]\*A[index]);

temp.pop\_back();

//不选index号数

DFS(index+1,nowK,sum,sumsqu);

}

int main()

{

scanf("%d%d%d",&n,&k,&x);

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

{

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

}

DFS(0,0,0,0);

while(!ans.empty())

{

cout<<ans.front()<<" ";

ans.erase(ans.begin());//这里打印的是头，所以删除的也应该是头

}

return 0;

}

## 尺取法

### 绝对半径

输入n,k,其中n是序列的长度，k是当事人可以扔掉序列中元素的最多个数，求扔掉一些子弹后，最长的连续相同序列的长度

#include<bits/stdc++.h>

using namespace std;

const int maxn=100005;

int a[maxn];

map<int,int> mp;//

int main()

{

int n,k;

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

{

//q.clear();

mp.clear();

memset(a,0,sizeof(a));

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

{

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

}

int l=1,r=1,ans=0;

while(r<=n)

{

while(r<=n&&r-l-mp[a[l]]<=k)

{

mp[a[r]]++;

r++;

}

ans=max(ans,mp[a[l]]);

// mp[a[l++]]--;

mp[a[l]]--;

l++;

while(l<=n&&a[l]==a[l-1])

{

mp[a[l]]--;

l++;

}

}

cout<<ans<<endl;

}

return 0;

}

### 北京网赛--环游城市

解题思路：/\*有n个城市围成一圈，编号为1-n，到达第i个城市需要花费ai金币 同时获得bi金币，起初有c金币，可以任意选择一个城市作为起点，

问能否从一个城市出发环游一圈？

用d[i]表示a[i]-b[i]，如果小于0，说明不能从i这个点出发。因为要求编号最小，所以用l和r从1开始，r枚举l这个点能走到哪，

l从小到大枚举出发点，围成一圈可以通过开两倍数组，把前面复制到后面来解决。\*/

一般面对这种围成一圈的，都可以开两倍数组，d[i+n]=d[i];在ans小于0时，应该是用while语句而不是if语句，直到某一个以l为开头的为正才符合要求。还有当l>n时，就说明每个点都枚举了一遍，没有符合要求的了，即-1.

#include <iostream>

#include<cstdio>

#include<cstring>

using namespace std;

//map<int,int>a,b,d;

const int maxn=1e6+5;

int a[maxn],b[maxn],d[maxn];

int main()

{

int T,n,c;

cin>>T;

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

{

memset(a,0,sizeof(a));

memset(b,0,sizeof(b));

memset(d,0,sizeof(d));

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

int temp;

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

{

scanf("%d",&temp);

a[i]=temp;

}

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

{

scanf("%d",&temp);

b[i]=temp;

d[i+n]=d[i]=a[i]-b[i];

}

int l=1,r=1;

long long ans=c;

while(l<=r&&r-l+1<=n)

{

ans+=d[r];

r++;

while(ans<0)

{

ans-=d[l];

l++;

}

}

if(l>n)

cout<<"-1"<<endl;

else

cout<<l<<endl;

}

return 0;

}

## bfs+dp

题意：

给一个100×100的迷宫，’.’表示路面，’S’表示起点，’T’表示终点；’#’表示毒气区，进入毒气区必须要消耗一个氧气；’B’表示氧气区，每次进入自动获得一个氧气，可反复进入从而获得多个，但最多携带5个；’P’表示加速药，获得原理和氧气一样，使用后使下一次移动不耗时，可以无限携带。一次移动可以移动到相邻的四个格子，花费一个单位时间，如果移动到了毒气区，将在毒气区额外停留一个单位时间。求从S到T的最短时间，如果不能到达，输出-1。

思路：

用dp[x][y][t]表示到达(x,y)且氧气瓶为t个时的最短时间，bfs根据每个点的特征处理每个点的最短时间（这里不用vis保存状态），最后在dp[ex][ey][0~5]里取最小值。

#include <iostream>

#include<bits/stdc++.h>

using namespace std;

const int inf=0x3f3f3f3f;

char a[105][105];

int sx,sy,ex,ey;

struct node

{

int x,y,num;

node(int i,int j,int k):x(i),y(j),num(k) {}

};

int dp[105][105][6];

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

int n,m;

int bfs()

{

queue<node> q;

node aa=node(sx,sy,0);

memset(dp,inf,sizeof(dp));

dp[sx][sy][0]=0;

q.push(aa);

while(!q.empty())

{

node t=q.front();

q.pop();

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

{

int newx=t.x+dir[i][0];

int newy=t.y+dir[i][1];

if(newx>=0&&newx<n&&newy>=0&&newy<m)

{

if((a[newx][newy]=='S'||a[newx][newy]=='.')&&dp[newx][newy][t.num]>dp[t.x][t.y][t.num]+1)

{

dp[newx][newy][t.num]=dp[t.x][t.y][t.num]+1;

q.push(node(newx,newy,t.num));

}

else if((a[newx][newy]=='#')&&t.num>0&&dp[newx][newy][t.num-1]>dp[t.x][t.y][t.num]+2)

{

dp[newx][newy][t.num-1]=dp[t.x][t.y][t.num]+2;

q.push(node(newx,newy,t.num-1));

}

else if(a[newx][newy]=='P'&&dp[newx][newy][t.num]>dp[t.x][t.y][t.num])

{

dp[newx][newy][t.num]=dp[t.x][t.y][t.num];

q.push(node(newx,newy,t.num));

}

else if (a[newx][newy]=='B'&&t.num<5&&dp[newx][newy][t.num+1]>dp[t.x][t.y][t.num]+1)

{

dp[newx][newy][t.num+1]=dp[t.x][t.y][t.num]+1;

q.push(node(newx,newy,t.num+1));

}

else if (a[newx][newy]=='B'&&t.num==5&&dp[newx][newy][t.num]>dp[t.x][t.y][t.num]+1)

{

dp[newx][newy][t.num]=dp[t.x][t.y][t.num]+1;

q.push(node(newx,newy,t.num));

}

else if(a[newx][newy]=='T'&&dp[newx][newy][t.num]>dp[t.x][t.y][t.num]+1)

{

dp[newx][newy][t.num]=dp[t.x][t.y][t.num]+1;

q.push(node(newx,newy,t.num));

}

}

}

}

int ans=inf;

for(int i=0;i<6;i++)

{

ans=min(ans,dp[ex][ey][i]);

}

if(ans!=inf)

return ans;

return -1;

}

int main()

{

//int n,m;

while(~scanf("%d%d",&n,&m)&&n&&m)

{

memset(a,0,sizeof(a));

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

{

scanf("%s",a[i]);

}

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

{

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

{

if(a[i][j]=='S')

{

sx=i;

sy=j;

}

else if(a[i][j]=='T')

{

ex=i;

ey=j;

}

}

}

cout<<bfs()<<endl;

}

return 0;

}

## 算法优化

### 后缀数组比大小

题意：设sufi表示以i为开始的后缀，即S[i..n]。请对每个i(1≤i<n)，判断sufi和sufi+1的字典序大小关系。

解答：我开始想的是每个索引的字母相比即可得到答案。然而当这个索引的字母和下个索引的字母相同时，不能直接看出。于是我想到了每当有这个情况发生时，就写一个for循环，直到有不相等的情况，输出答案。然而，由于我是每个左寅每个索引的算，这样就会在算下一个索引时，执行与刚才同样的步骤，这样就会超时啊。所以以后超时的时候，要考虑自己的代码复杂度，是否是有很多情况重复计算了。还有就是用string比char[]慢很多。

#include<bits/stdc++.h>

using namespace std;

char S[int(1e6+5)] ;

int main()

{

int T;

scanf("%d",&T);

while(T--)

{

int n;

scanf("%d",&n);

scanf("%s",S);

for(int i=0;i<n-1;i++)

{

int pos=i;

while(S[pos]==S[pos+1]&&pos<n-2)

pos++;

if(S[pos]<S[pos+1])

{

for(int j=0;j<pos-i+1;j++)

{

printf("<");

}

}

else if(S[pos]>S[pos+1])

{

for(int j=0;j<pos-i+1;j++)

{

printf(">");

}

}

else if(pos==n-2)

{

for(int j=0;j<pos-i+1;j++)

printf(">");

}

i=pos;

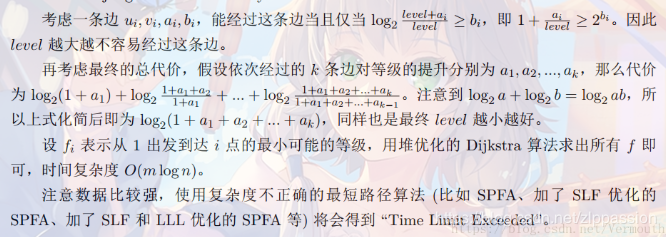
}

printf("\n");

}

}

### **Dijkstra+堆优化**



/\*高玩小Q不仅喜欢玩寻宝游戏，还喜欢一款升级养成类游戏。在这个游戏的世界地图中一共有n个城镇，编号依次为1到n。

这些城镇之间有m条单向道路，第i 条单项道路包含四个参数ui,vi,ai,bi，表示一条从ui号城镇出发，在vi号城镇结束的单向道路，

因为是单向道路，这不意味着小Q可以从vi沿着该道路走到ui。小Q的初始等级level为1，每当试图经过一条道路时，需要支付

cost=log2level+ailevel点积分，并且经过该道路后，小Q的等级会提升ai级，到达level+ai级。但是每条道路都会在一定意义上歧视低消费玩家

，准确地说，如果该次所需积分cost<bi，那么小Q不能经过该次道路，也不能提升相应的等级。

小Q位于1号城镇，等级为1，现在为了做任务要到n号城镇去。这将会是一次奢侈的旅行，求小Q找到需要支付的总积分最少的

#include<bits/stdc++.h>

using namespace std;

const int maxn = 1000050;

const long long INF = 1ll<<61; //这里一开始用了0x3f3f3f3f...sb了

struct qnode

{

int v;

long long c;//c就是代表等级

qnode(int v=0,long long c=0):v(v),c(c){}

bool operator <(const qnode &r)const

{

return c>r.c;

}

};

struct Edge

{

int to;//to即是v

int w,MIN;//w即是ai,MIN即是bi

Edge(int to,int w,int MIN):to(to),w(w),MIN(MIN){}

};

vector<Edge>E[maxn];

bool vis[maxn];

long long dis[maxn];

void Dijstra(int n,int start)

{

memset(vis,false,sizeof(vis)); //nima,一开始赋值成-1了....

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

dis[i] = INF;//dis[i]相当于level

priority\_queue<qnode> que;//

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

dis[start] = 1;//初始等级为1

que.push(qnode(start,1));

qnode tmp;

while(!que.empty())

{

tmp = que.top();//当前节点

que.pop();

int u = tmp.v;//当前节点的编号

if(vis[u])

continue;

vis[u] = 1;

for(int i=0;i<E[u].size();i++)

{

int v = E[u][i].to;//当前节点的“对象”，可到达的那一边

int a = E[u][i].w;//到达那边可增加的等级

//if()

if(log2((dis[u]+a\*1.0) / dis[u]\*1.0)<E[u][i].MIN)

//if((a/dis[u])<E[u][i].MIN) //这里不乘1.0也可以

continue;

if(!vis[v]&&dis[v]>dis[u]+a) //这里的！VIS重要

{

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

que.push(qnode(v,dis[v]));

}

}

}

if(dis[n]==INF)

printf("-1\n");

else

printf("%d\n",(int)log2(dis[n]\*1.0));

}

int main()

{

int t,m,u,v,n;

int a,b;

scanf("%d",&t);

while(t--)

{

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

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

E[i].clear();

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

{

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

//b=(1ll<<b)-1;

E[u].push\_back(Edge(v,a,b));

}

Dijstra(n,1);

}

return 0;

}

## 树的遍历

### 1、给定前序和中序，求层序

#include <iostream>

#include<map>

#include<queue>

#include<cstdio>

using namespace std;

int n,post[35],in[35];

map<int,int> lch,rch;

int build(int L1,int R1,int L2,int R2)

{

if(L1>R1)

return -1;

int root=post[R1];

int p=L2;

while(in[p]!=root)

p++;

int cnt=p-L2;

lch[root]=build(L1,L1+cnt-1,L2,p-1);

rch[root]=build(L1+cnt,R1-1,p+1,R2);

return root;

}

int ans=0;

void layer(int a)

{

queue<int> q;

q.push(a);

while(!q.empty())

{

int t=q.front();

q.pop();

cout<<t;

ans++;

if(ans<n)

cout<<" ";

if(lch[t]!=-1)

q.push(lch[t]);

if(rch[t]!=-1)

q.push(rch[t]);

}

}

int main()

{

cin>>n;

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

{

cin>>post[i];

}

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

{

cin>>in[i];

}

build(0,n-1,0,n-1);

layer(post[n-1]);

return 0;

}

## Bfs

### 求矩阵中块的个数

/\*问题：给出一个m\*n的矩阵，矩阵中的元素为0或1，称位置（x,y）与其上下左右的位置为相邻。如果矩阵中有若干个1是相邻的，

那么称这些1构成了一个“块”，求给定的矩阵中“块”的个数\*/

#include <iostream>

#include<queue>

#include<cstdio>

using namespace std;

const int maxn=100;

bool inq[maxn][maxn]={false};

int juzhen[maxn][maxn];

int X[]={0,0,-1,1};

int Y[]={1,-1,0,0};

int n,m;//n\*m矩阵

int ans=0;

struct node

{

int x;

int y;

}Node;//Node只是一个结点

bool check(int x,int y)

{

if(juzhen[x][y]==0||inq[x][y]==true)

{

return false;

}

if(x>=n||x<0||y>=m||y<0)//这是二维数组，自行理解

{

return false;

}

return true;

}

void BFS(int x,int y)

{

queue<node> Q;

Node.x=x;

Node.y=y;

int newX,newY;

Q.push(Node);

inq[x][y]=true;//这步我忘

while(!Q.empty())

{

node s=Q.front();//取出队首元素,这步我不会定义

Q.pop();//队首元素出队

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

{

newX=s.x+X[i];

newY=s.y+Y[i];

while(check(newX,newY))

{

Node.x=newX;

Node.y=newY;

Q.push(Node);

inq[newX][newY]=true;

}

}

}

}

int main()

{

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

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

{

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

{

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

}

}

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

{

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

{

if(juzhen[i][j]==1&&inq[i][j]==false)

{

ans++;

BFS(i,j);

}

}

}

cout << ans << endl;

return 0;

}

### 走迷宫的最小步数

/\*问题: 给定一个n\*m大小的迷宫，其中\*代表不可通过的墙壁，而“.”代表平地，S表示起点，T代表终点，移动过程中，

如果当前位置为(x,y)（下标从0开始），且每次只能前往上下左右（x,y+1）,（x,y-1）,（x-1,y）,（x+1,y）四个位置的平地，求从起点S到达终点T的最小步数\*/

#include <iostream>

#include<cstdio>

#include<queue>

using namespace std;

const int maxn=100;

int X[]={0,0,-1,1};

int Y[]={1,-1,0,0};

int n,m;

int step=0;

struct node

{

int x;

int y;

int step;

}S,T,Node;//S为起点，T为终点，Node为临时结点

char mig[maxn][maxn];

bool inq[maxn][maxn]={false};

bool check(int x,int y)

{

if(x>=n||x<0||y>=m||y<0)

{

return false;

}

if(mig[x][y]=='\*'||inq[x][y]==true)

{

return false;

}

return true;

}

int BFS()

{

queue<node> Q;

Q.push(S);

while(!Q.empty())

{

node top=Q.front();

Q.pop();

if(top.x==T.x&&top.y==T.y)

{

return top.step;

}

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

{

int newX=top.x+X[i];

int newY=top.y+Y[i];

if(check(newX,newY))

{

Node.x=newX;

Node.y=newY;

Node.step=top.step+1;

Q.push(Node);

inq[newX][newY]=true;

}

}

}

return -1;//无法到达终点

}

int main()

{

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

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

{

getchar();//过滤到每行后的换行符

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

{

mig[i][j]=getchar();

}

mig[i][m+1]='\0';

}

scanf("%d%d%d%d",&S.x,&S.y,&T.x,&T.y);

S.step=0;//初始化起点的层数为0，即S到S的最小步数为0

cout <<BFS()<< endl;

return 0;

}

1. [↑](#endnote-ref-0)