ACM template

BJTU

Tshu WanG

2018

Tshu's ACM­­­ template

Table of Contents

[Basic Algorithm 1](#_Toc525287161)

[Set 5](#_Toc525287164)

[Data Structure 6](#_Toc525287166)

[Graph 13](#_Toc525287167)

[Computational Geometry 23](#_Toc525287168)

[Math Problem 29](#_Toc525287169)

[String 43](#_Toc525287170)

[Others 47](#_Toc525287171)

## 

## Basic Algorithm

### Sort

// Bubble Sort  
for (int i = 0; i < N; i++)  
 for (int j = 0; j < N - i - 1; j++)  
 if (A[j] > A[j + 1]) swap(A[j], A[j + 1]);

// Insertion Sort  
for (int i = 1; i < N; i++) {  
 int tmp = A[i], j;  
 for (j = i - 1; j >= 0 && A[j] > tmp; j--)  
 A[j + 1] = A[j];  
 A[++j] = tmp;  
}

// Selection Sort  
for (int i = 0; i < N; i++)  
 for (int j = i + 1; j < N; j++)  
 if (A[i] > A[j]) swap(A[i], A[j])

// Merge Sort  
void merge\_sort(int A[], int l, int r) {  
 if (l + 1 >= r) return ;  
 int mid = (l + r) / 2;  
 merge\_sort(A, l, mid);  
 merge\_sort(A, mid, r);  
 int i, j, k;  
 i = l; j = mid; k = l;  
 while (i < mid && j < r) {  
 if (A[i] <= A[j]) B[k++] = A[i++];  
 else B[k++] = A[j++];  
 }  
 while (i < mid) {  
 B[k++] = A[i++];  
 }  
 while (j < r) {  
 B[k++] = A[j++];  
 }  
 for (int i = l; i < r; i++)  
 A[i] = B[i];  
 return ;  
}

// Quick Sort  
void quicksort(int A[], int l, int r) {  
 int i = l, j = r, mid = A[(r - l) / 2 + l];  
 while (i <= j) {  
 while (A[i] < mid) i++;  
 while (A[j] > mid) j--;  
 if (i <= j) {  
 swap(A[i], A[j]);  
 ++i; --j;  
 }  
 }  
 if (i < r) quicksort(A, i, r);  
 if (l < j) quicksort(A, l, j);  
 return ;  
}

* Heap Sort（见堆的内容）

### DP

#### LIS

int A[maxn];  
long lis(int n) {  
 int dp[maxn];  
 fill(dp, dp + n, INF);  
 for (int i = 0; i < n; ++i)  
 \*lower\_bound(dp, dp + n, A[i]) = A[i];// lds: -A[i]; ln: upper\_bound  
 return lower\_bound(dp, dp + n, INF) - dp;  
}

#### Knapsack Problem

* 0/1 背包

for (int i = 0; i < N; ++i)  
 for (int j = W; j >= w[i]; --j)  
 f[j] = max(f[j - w[i]] + c[i], f[j]);

* 完全背包

for (int i = 0; i < N; ++i)  
 for (int j = w[i]; j <= W; ++j)  
 f[j] = max(f[j - w[i]] + c[i], f[v]);

注意循环顺序的不同背后思路。

* 一个简单的优化：若两件物品 i、j 满足 且 ，则将物品 j 去掉，不用考虑。
* 转化为 01 背包问题求解：
  + 第 i 种物品转化为 件费用于价值均不变的物品。
  + 第 i 种物品拆成费用为 ，价值为 的若干件物品其中 k 满足
* 多重背包
* 优化：转化为 01 背包问题
  + 将第 i 件物品分成若干件物品，每件物品的系数分别为：
  + 根据 w，v 范围改变 DP 对象，可以考虑针对不同价值计算最小的重量。（ ，其中 j 代表价值总和）

for (int i = 0; i < N; ++i) {  
 int num = m[i];  
 for (int k = 1; num > 0; k <<= 1) {  
 int mul = min(k, num);  
 for (int j = W; j >= w[i] \* mul; --j) {  
 f[j] = max(f[j - w[i] \* mul] + v[i] \* mul, f[j]);  
 }  
 num -= mul;  
 }  
}

* 混合三种背包

弄清楚上面三种背包后分情况就好

* 超大背包
  + ，,

int n;  
ll w[maxn], v[maxn], W;  
Pll ps[1 << (maxn / 2)]; // (w, v);  
  
void solve() {  
 int n2 = n / 2;  
 for (int i = 0; i < 1 << n2; ++i) {  
 ll sw = 0, sv = 0;  
 for (int j = 0; j < n2; ++j)  
 if (i >> j & 1) {  
 sw += w[j];  
 sv += v[j];  
 }  
 ps[i] = Pll(sw, sv);  
 }  
 sort(ps, ps + (1 << n2));  
 int m = 1;  
 for (int i = 1; i < 1 << n2; ++i)  
 if (ps[m - 1].second < ps[i].second)  
 ps[m++] = ps[i];  
  
 ll res = 0;  
 for (int i = 0; i < 1 << (n - n2); ++i) {  
 ll sw = 0, sv = 0;  
 for (int j = 0; j < n - n2; ++j)  
 if (i >> j & 1) {  
 sw += w[n2 + j];  
 sv += v[n2 + j];  
 }  
 if (sw <= W) {  
 ll tv = (lower\_bound(ps, ps + m, make\_pair(W - sw, INF)) - 1)->second;  
 res = max(res, sv + tv);  
 }  
 }  
 printf("%lld\n", res);  
}

* 二维费用背包

二维费用可由最多取 m 件等方式隐蔽给出。

* 分组背包

for (int k = 0; k < K; ++k)  
 for (j = W; j >= 0; --j)  
 for (int i = 0; i <= m[k]; ++i)  
 f[j] = max(f[j - w[i]] + v[i], f[j]);

显然可以对每组中物品应用完全背包中“一个简单有效的优化”

* 有依赖背包

由 NOIP2006 金明的预算方案引申，对每个附件先做一个 01 背包，再与组件得到一个 个物品组。 更一般问题，依赖关系由「森林」形式给出，涉及到树形 DP 以及泛化物品，这里不表。

* 背包问题方案总数

更多内容详见「背包九讲」

#### Maximum Subarray Sum

int max\_subarray\_sum(int A[], int n) {  
 int res, cur;  
 if (!A || n <= 0) return 0;  
 res = cur = a[0];  
 for (int i = 0; i < n; ++i) {  
 if (cur < 0) cur = a[i];  
 else cur += a[i];  
 res = max(cur, res);  
 }  
 return res;  
}

## Set

// 子集枚举  
int sub = sup;  
do {  
 sub = (sub - 1) & sup;  
} while (sub != sup); // -1 & sup = sup;  
  
// 势为 k 的集合枚举  
int comb = (1 << k) - 1;  
while (comb < 1 << n) {  
 int x = comb & -comb, y = comb + x;  
 comb = ((comb & ~y) / x >> 1) | y;  
}  
  
// 排列组合  
do {  
  
} while (next\_permutation(A, A + N)); // prev\_permutation  
  
// 高维前缀和(子集/超集和)  
// 子集和  
for (int i = 0; i < k; i++)  
 for (int s = 0; s < 1 << k; s++)  
 if (s >> i & 1) cnt[s] += cnt[s ^ (1 << i)];  
// 超集和  
for (int i = 0; i < k; i++)  
 for (int s = 0; s < 1 << k; s++)  
 if (!(s >> i & 1)) cnt[s] += cnt[s | (1 << i)];

### Bit operation

int \_\_builtin\_ffs (unsigned int x)  
//返回x的最后一位1的是从后向前第几位，比如7368（1110011001000）返回4。  
int \_\_builtin\_clz (unsigned int x)  
// 返回前导的0的个数。  
int \_\_builtin\_ctz (unsigned int x)  
// 返回后面的0个个数，和\_\_builtiin\_clz相对。  
int \_\_builtin\_popcount (unsigned int x)  
// 返回二进制表示中1的个数。  
int \_\_builtin\_parity (unsigned int x)  
// 返回x的奇偶校验位，也就是x的1的个数模2的结果。

## Data Structure

// Heap  
int heap[maxn], sz = 0;  
void push(int x) {  
 int i = sz++;  
  
 while (i > 0) {  
 int p = (i - 1) / 2;  
 if (heap[p] <= x) break;  
 heap[i] = heap[p];  
 i = p;  
 }  
 heap[i] = x;  
}  
int pop() {  
 int ret = heap[0];  
 int x = heap[--sz];  
 int i = 0;  
 while (i \* 2 + 1 < sz) {  
 int a = i \* 2 + 1, b = i \* 2 + 2;  
 if (b < sz && heap[b] < heap[a]) a = b;  
 if (heap[a] >= x) break;  
 heap[i] = heap[a];  
 i = a;  
 }  
 heap[i] = x;  
 return ret;  
}

// Binary Search Tree  
struct node {  
 int val;  
 node \*lch, rch;  
};  
  
node \*insert(node \*p, int x) {  
 if (p == NULL) {  
 node \*q = new node;  
 q->val = x;  
 q->lch = q->rch = NULL;  
 return q;  
 } else {  
 if (x < p->val) p->lch = insert(p->lch, x);  
 else p->rch = insert(p->rch, x);  
 return p;  
 }  
}  
bool find(node \*p, int x) {  
 if (p == NULL) return false;  
 else if (x == p->val) return true;  
 else if (x < p->val) return find(p->lch, x);  
 else return find(p->rch, x);  
}  
node \*remove(node \*p, int x) {  
 if (p == NULL) return NULL;  
 else if (x < p->val) p->lch = remove(p->lch, x);  
 else if (x > p->val) p->rch = remove(p->rch, x);  
 else if (p->lch == NULL) {  
 node \*q = p->rch;  
 delete p;  
 return q;  
 } else if (p->lch->rch == NULL) {  
 node \*q = p->lch;  
 q->rch = p->rch;  
 delete p;  
 return q;  
 } else {  
 // 把左儿子子孙中最大的节点提到需要删除的节点上  
 node \*q;  
 for (q = p->lch; q->rch->rch != NULL; q = q->rch);  
 node \*r = q->rch;  
 q->rch = r->lch;  
 r->lch = p->lch;  
 r->rch = p->rch;  
 delete p;  
 return r;  
 }  
 return p;  
}

// Union-find Set  
int par[maxn];  
int rnk[maxn];  
void init(int n) {  
 for (int i = 0; i < n; ++i) {  
 par[i] = i;  
 rnk[i] = 0;  
 }  
}  
int find(int x) {  
 return par[x] == x? x : par[x] = find(par[x]);  
}  
bool same(int x, int y) {  
 return find(x) == find(y);  
}  
void unite(int x, int y) {  
 x = find(x);  
 y = find(y);  
 if (x == y) return;  
 if (rnk[x] < rnk[y]) {  
 par[x] = y;  
 } else {  
 par[y] = x;  
 if (rnk[x] == rnk[y]) rnk[x]++;  
 }  
}

**当然，更快捷简单的做法，是使用 C++ 的 container。**

// Segment Tree  
const int maxn = 1 << 17;  
int n, dat[2 \* maxn - 1];  
void init(int \_n) {  
 n = 1;  
 while (n < \_n) n <<= 1;  
 for (int i = 0; i < 2 \* n - 1; ++i)  
 dat[i] = INF;  
}  
void update(int k, int a) {  
 k += n - 1;  
 dat[k] = a;  
 while (k > 0) {  
 k = (k - 1) / 2;  
 dat[k] = min(dat[2 \* k + 1], dat[2 \* k + 2]);  
 }  
}  
// query [a, b), index k in [l, r)  
// query(a, b, 0, 0, n)  
int query(int a, int b, int k, int l, int r) {  
 if (r <= a || b <= l) return INF;  
 if (a <= l && r <= b) return dat[k];  
 else {  
 int v1 = query(a, b, k \* 2 + 1, l, (l + r) / 2);  
 int v2 = query(a, b, k \* 2 + 2, (l + r) / 2, r);  
 return min(v1, v2);  
 }  
}

// RMQ  
int n, dat[2 \* maxn - 1];  
void init(int \_n) {  
 n = 1;  
 while (n < \_n) n <<= 1;  
 for (int i = 0; i < 2 \* n - 1; ++i)  
 dat[i] = INF;  
}  
void update(int k, int a) {  
 k += n - 1;  
 dat[k] = a;  
 while (k > 0) {  
 k = (k - 1) / 2;  
 dat[k] = min(dat[2 \* k + 1], dat[2 \* k + 2]);  
 }  
}  
// query [a, b), index k in [l, r)  
// query(a, b, 0, 0, n)  
int query(int a, int b, int k, int l, int r) {  
 if (r <= a || b <= l) return INF;  
 if (a <= l && r <= b) return dat[k];  
 else {  
 int v1 = query(a, b, k \* 2 + 1, l, (l + r) / 2);  
 int v2 = query(a, b, k \* 2 + 2, (l + r) / 2, r);  
 return min(v1, v2);  
 }  
}

// IntervalTree2D  
// UVa11297 Census：带build的版本  
// Rujia Liu  
#include<algorithm>  
using namespace std;  
  
const int INF = 1<<30;  
const int maxn = 2000 + 10;  
  
int A[maxn][maxn];  
  
struct IntervalTree2D {  
 int Max[maxn][maxn], Min[maxn][maxn], n, m;  
 int xo, xleaf, row, x1, y1, x2, y2, x, y, v, vmax, vmin; // 参数、查询结果和中间变量  
  
 void query1D(int o, int L, int R) {  
 if(y1 <= L && R <= y2) {  
 vmax = max(Max[xo][o], vmax); vmin = min(Min[xo][o], vmin);  
 } else {  
 int M = L + (R-L)/2;  
 if(y1 <= M) query1D(o\*2, L, M);  
 if(M < y2) query1D(o\*2+1, M+1, R);  
 }  
 }  
  
 void query2D(int o, int L, int R) {  
 if(x1 <= L && R <= x2) { xo = o; query1D(1, 1, m); }  
 else {  
 int M = L + (R-L)/2;  
 if(x1 <= M) query2D(o\*2, L, M);  
 if(M < x2) query2D(o\*2+1, M+1, R);  
 }  
 }  
  
 void modify1D(int o, int L, int R) {  
 if(L == R) {  
 if(xleaf) { Max[xo][o] = Min[xo][o] = v; return; }  
 Max[xo][o] = max(Max[xo\*2][o], Max[xo\*2+1][o]);  
 Min[xo][o] = min(Min[xo\*2][o], Min[xo\*2+1][o]);  
 } else {  
 int M = L + (R-L)/2;  
 if(y <= M) modify1D(o\*2, L, M);  
 else modify1D(o\*2+1, M+1, R);  
 Max[xo][o] = max(Max[xo][o\*2], Max[xo][o\*2+1]);  
 Min[xo][o] = min(Min[xo][o\*2], Min[xo][o\*2+1]);  
 }  
 }  
  
 void modify2D(int o, int L, int R) {  
 if(L == R) { xo = o; xleaf = 1; modify1D(1, 1, m); }  
 else {  
 int M = L + (R-L)/2;  
 if(x <= M) modify2D(o\*2, L, M);  
 else modify2D(o\*2+1, M+1, R);  
 xo = o; xleaf = 0; modify1D(1, 1, m);  
 }  
 }  
  
 // 只构建xo为叶子（即x1=x2）的y树  
 void build1D(int o, int L, int R) {  
 if(L == R) Max[xo][o] = Min[xo][o] = A[row][L];  
 else {  
 int M = L + (R-L)/2;  
 build1D(o\*2, L, M);  
 build1D(o\*2+1, M+1, R);  
 Max[xo][o] = max(Max[xo][o\*2], Max[xo][o\*2+1]);  
 Min[xo][o] = min(Min[xo][o\*2], Min[xo][o\*2+1]);  
 }  
 }  
  
 void build2D(int o, int L, int R) {  
 if(L == R) { xo = o; row = L; build1D(1, 1, m); }  
 else {  
 int M = L + (R-L)/2;  
 build2D(o\*2, L, M);  
 build2D(o\*2+1, M+1, R);  
 for(int i = 1; i <= m\*4; i++) {  
 Max[o][i] = max(Max[o\*2][i], Max[o\*2+1][i]);  
 Min[o][i] = min(Min[o\*2][i], Min[o\*2+1][i]);  
 }  
 }  
 }  
  
 void query() {  
 vmax = -INF; vmin = INF;  
 query2D(1, 1, n);  
 }  
  
 void modify() {  
 modify2D(1, 1, n);  
 }  
  
 void build() {  
 build2D(1, 1, n);  
 }  
};  
  
IntervalTree2D t;  
  
#include<cstdio>  
  
int main() {  
 int n, m, Q, x1, y1, x2, y2, x, y, v;  
 char op[10];  
 scanf("%d%d", &n, &m);  
 t.n = n; t.m = m;  
 for(int i = 1; i <= n; i++)  
 for(int j = 1; j <= m; j++)  
 scanf("%d", &A[i][j]);  
 t.build();  
  
 scanf("%d", &Q);  
 while(Q--) {  
 scanf("%s", op);  
 if(op[0] == 'q') {  
 scanf("%d%d%d%d", &t.x1, &t.y1, &t.x2, &t.y2);  
 t.query();  
 printf("%d %d\n", t.vmax, t.vmin);  
 } else {  
 scanf("%d%d%d", &t.x, &t.y, &t.v);  
 t.modify();  
 }  
 }  
 return 0;  
}

//Sparse Table  
const int maxn = 1e5 + 10;  
const int MAX\_K = 31 - \_\_builtin\_clz(maxn);  
  
int n, ST[maxn][MAX\_K + 1], A[maxn];  
void build(int N) {  
 for (int i = 0; i < N; ++i)  
 ST[i][0] = A[i];  
 int k = 31 - \_\_builtin\_clz(N);  
 for (int j = 1; j <= k; ++j)  
 for (int i = 0; i <= N - (1 << j); ++i)  
 ST[i][j] = min(ST[i][j - 1], ST[i + (1 << (j - 1))][j - 1]);  
}  
int query(int l, int r) {  
 if (l >= r) return 0;  
 int ans = INF, k = 31 - \_\_builtin\_clz(r - l);  
 for (int j = k; j >= 0; --j)  
 if (l + (1 << j) - 1 <= r) {  
 ans = min(ans, ST[l][j]);  
 l += 1 << j;  
 }  
 return ans;  
}  
int RMQ(int l, int r) {  
 if (l >= r) return 0;  
 int k = 31 - \_\_builtin\_clz(r - l);  
 return min(ST[l][k], ST[r - (1 << k)][k]);  
}

// lowbit  
int lowbit(int i) {  
 return ~i & i + 1;  
}

// 单点修改/查询  
int bit[maxn];  
int sum(int i) {  
 int s = 0;  
 while (i > 0) {  
 s += bit[i];  
 i -= i & -i;  
 }  
 return s;  
}  
void add(int i, int x) {  
 while (i <= n) {  
 bit[i] += x;  
 i += i & -i;  
 }  
}

// 区间修改/查询  
struct bit {  
 int bit[maxn];  
 int sum(int i) {  
 int s = 0;  
 while (i > 0) {  
 s += bit[i];  
 i -= i & -i;  
 }  
 return s;  
 }  
 void add(int i, int x) {  
 while (i <= n) {  
 bit[i] += x;  
 i += i & -i;  
 }  
 }  
}a, b;  
inline void add(int l, int r, int t) {  
 a.add(l,t); a.add(r+1,-t);  
 b.add(l,-t\*(l-1)); b.add(r+1,t\*r);  
}  
inline int get(int i) {  
 return a.sum(i)\*i+b.sum(i);  
}  
inline int get(int l, int r) {  
 return get(r)-get(l - 1);  
}

// 二维单点修改/查询  
int bit[maxn][maxn];  
int sum(int x, int y) {  
 int res = 0;  
 for (int i = x; i > 0; i -= i & -i)  
 for (int j = y; j > 0; j -= j & -j)  
 res += bit[i][j];  
 return res;  
}  
void add(int x, int y, int k) {  
 for (int i = x; i <= n; i += i & -i)  
 for (int j = y; j <= n; j += j & -j)  
 bit[i][j] += k;  
}

// 二维区间修改/查询  
struct bit {  
 int a[maxn][maxn];  
 inline int lowbit(int x) {  
 return x&(-x);  
 }  
 inline void add(int x,int y,int t) {  
 int i,j;  
 for(i=x;i<maxn;i+=lowbit(i)) {  
 for(j=y;j<maxn;j+=lowbit(j))a[i][j]+=t;  
 }  
 }  
 inline int get(int x,int y) {  
 int ans=0;  
 int i,j;  
 for(i=x;i>0;i-=lowbit(i)) {  
 for(j=y;j>0;j-=lowbit(j))ans+=a[i][j];  
 }  
 return ans;  
 }  
}a,b,c,d;  
inline void add(int x1,int y1,int x2,int y2,int t) {  
 a.add(x1,y1,t),a.add(x1,y2+1,-t);  
 a.add(x2+1,y1,-t),a.add(x2+1,y2+1,t);  
  
 b.add(x1,y1,t\*x1); b.add(x2+1,y1,-t\*(x2+1));  
 b.add(x1,y2+1,-t\*x1); b.add(x2+1,y2+1,t\*(x2+1));  
  
 c.add(x1,y1,t\*y1); c.add(x2+1,y1,-t\*y1);  
 c.add(x1,y2+1,-t\*(y2+1)); c.add(x2+1,y2+1,t\*(y2+1));  
  
 d.add(x1,y1,t\*x1\*y1); d.add(x2+1,y1,-t\*(x2+1)\*y1);  
 d.add(x1,y2+1,-t\*x1\*(y2+1)); d.add(x2+1,y2+1,t\*(x2+1)\*(y2+1));  
}  
inline int get(int x,int y) {  
 return a.get(x,y)\*(x+1)\*(y+1)-b.get(x,y)\*(y+1)-(x+1)\*c.get(x,y)+d.get(x,y);  
}  
inline int get(int x1,int y1,int x2,int y2) {  
 return get(x2,y2)-get(x2,y1-1)-get(x1-1,y2)+get(x1-1,y1-1);  
}

## Graph

struct edge {  
 int from;  
 int to, dis;  
};  
vector<edge> G[MAX\_V];  
vector<edge> es;  
bool vis[MAX\_V];  
int V, E, pre[MAX\_V], dist[MAX\_V];  
// int cost[MAX\_V][MAX\_V];

// Shortest Way  
void dijkstra(int s) {  
 priority\_queue<Pii, vector<Pii>, greater<Pii> > que;// fisrt 是最短距离，second 是顶点编号  
 fill(dist, dist + V, INF);  
 dist[s] = 0; que.push(Pii(0, s));  
 while (!que.empty()) {  
 Pii p = que.top(); que.pop();  
 int v = p.second;  
 if (dist[v] < p.first) continue;  
 for (int i = 0; i < G[v].size(); i++) {  
 edge e = G[v][i];  
 if (dist[e.to] > dist[v] + e.dis) {  
 dist[e.to] = dist[v] + e.dis;  
 que.push(Pii(dist[e.to], e.to));  
 }  
 }  
 }  
}  
void bellman\_ford(int s) {  
 fill(dist, dist + V, INF);  
 dist[s] = 0;  
 while (true) {  
 bool update = false;  
 for (int i = 0; i < E; ++i) {  
 edge e = es[i];  
 if (dist[e.from] != INF && dist[e.from] + e.dis < dist[e.to]) {  
 update = true;  
 dist[e.to] = dist[e.from] + e.dis;  
 }  
 }  
 if (!update) break;  
 }  
}  
bool find\_negative\_loop() {  
 memset(dist, 0, sizeof dist);  
 for (int i = 0; i < V; ++i)  
 for (int j = 0; j < E; ++j) {  
 edge e = es[j];  
 if (d[e.to] > d[e.from] + e.dis) {  
 d[e.to] = d[e.from] + e.dis;  
 if (i == V - 1) return true;  
 }  
 }  
 return false;  
}  
void spfa(int s) {  
 queue<int> que;  
 fill(dist, dist + V, INF);  
 fill(vis, vis + V, false);  
 dist[s] = 0; que.push(s); vis[s] = true;  
 while (!que.empty()) {  
 int v = que.front(); que.pop();  
 vis[v] = false;  
 for (int i = 0; i < G[v].size(); ++i) {  
 int u = G[v][i].to;  
 if (dist[u] > dist[v] + G[v][i].dis) {  
 dist[u] = dist[v] + G[v][i].dis;  
 if (!vis[u]) {  
 que.push(u);  
 vis[u] = true;  
 }  
 }  
 }  
 }  
}

// Spanning Tree  
int prime() {  
 /\*  
 fill(dist, dist + V, INF);  
 fill(vis, vis + V, false);  
 dist[0] = 0;  
 int res = 0;  
 while (true) {  
 int v = -1;  
 for (int u = 0; u < V; ++u) {  
 if(!vis[u] && (v == -1 || dist[u] < dist[v])) v = u;  
 }  
 if (v == -1) break;  
 vis[v] = true;  
 res += dist[v];  
 for (int u = 0; u < V; u++)  
 dist[u] = min(dist[u], cost[v][u]);  
 }  
 //\*/  
 priority\_queue<Pii, vector<Pii>, greater<Pii> > que;  
 int res = 0;  
 fill(dist, dist + V, INF);  
 fill(vis, vis + V, false);  
 dist[0] = 0;  
 que.push(Pii(0, 0));  
 while (!que.empty()) {  
 Pii p = que.top(); que.pop();  
 int v = p.second;  
 if (vis[v] || dist[v] < p.first) continue;  
 res += dist[v]; vis[v] = true;  
 for (int i = 0; i < G[v].size(); ++i) {  
 edge e = G[v][i];  
 if (dist[e.to] > e.dis) {  
 dist[e.to] = e.dis;  
 que.push(Pii(dist[e.to], e.to));  
 }  
 }  
 }  
 return res;  
}  
  
bool cmp(const edge e1, const edge e2) {  
 return e1.dis < e2.dis;  
}  
int kruskal() {  
 sort(es.begin(), es.end(), cmp);  
 init(V);  
 int res = 0;  
 for (int i = 0; i < E; ++i) {  
 edge e = es[i];  
 if (!same(e.from, e.to)) {  
 unite(e.from, e.to);  
 res += e.dis;  
 }  
 }  
 return res;  
}

// SCC  
int V, cmp[MAX\_V];  
vector<int> G[MAX\_V], rG[MAX\_V], vs;  
bool used[MAX\_V];  
  
void add\_edge(int from, int to) {  
 G[from].push\_back(to); rG[to].push\_back(from);  
}  
void dfs(int v) {  
 used[v] = true;  
 for (int i = 0; i < G[v].size(); ++i)  
 if (!used[G[v][i]]) dfs(G[v][i]);  
 vs.push\_back(v);  
}  
void rdfs(int v, int k) {  
 used[v] = true;  
 cmp[v] = k;  
 for (int i = 0; i < rG[v].size(); ++i)  
 if (!used[rG[v][i]]) rdfs(rG[v][i], k);  
}  
int scc() {  
 memset(used, 0, sizeof used);  
 vs.clear();  
 for (int v = 0; v < V; ++v)  
 if (!used[v]) dfs(v);  
 memset(used, 0, sizeof used);  
 int k = 0;  
 for (int i = vs.size() - 1; i >= 0; --i)  
 if (!used[vs[i]]) rdfs(vs[i], k++);  
 return k;  
}

// Bipartite Matching  
void add\_edge(int u, int v) {  
 G[u].push\_back(v); G[v].push\_back(u);  
}  
bool dfs(int v) {  
 used[v] = true;  
 for (int i = 0; i < (int)G[v].size(); i++) {  
 int u = G[v][i], w = match[u];  
 if (w < 0 || (!used[w] && dfs(w))) {  
 match[v] = u; match[u] = v;  
 return true;  
 }  
 }  
 return false;  
}  
int bipartite\_matching() {  
 int res = 0;  
 memset(match, -1, sizeof match);  
 for (int v = 0; v < V; v++)  
 if (match[v] < 0) {  
 memset(used, false, sizeof used);  
 if (dfs(v)) ++res;  
 }  
 return res;  
}

// Network Flow  
struct edge{  
 int to, cap, rev;  
};  
vector<edge> G[MAX\_V];  
int level[MAX\_V], iter[MAX\_V];  
void add\_edge(int from, int to, int cap) {  
 G[from].push\_back((edge){to, cap, static\_cast<int>(G[to].size())});  
 G[to].push\_back((edge){from, 0, static\_cast<int>(G[from].size() - 1)});  
}  
// Ford-Fulkerson  
int dfs(int v, int t, int f) {  
 if (v == t) return f;  
 flag[v] = true;  
 for (int i = 0; i < (int)G[v].size(); i++) {  
 edge &e = G[v][i];  
 if (!flag[e.to] && e.cap > 0) {  
 int d = dfs(e.to, t, min(f, e.cap));  
 if (d > 0) {  
 e.cap -= d;  
 G[e.to][e.rev].cap += d;  
 return d;  
 }  
 }  
 }  
 return 0;  
}  
int max\_flow(int s, int t) {  
 int flow = 0;  
 for(;;) {  
 memset(flag, false, sizeof flag);  
 int f = dfs(s, t, INF);  
 if (!f) return flow;  
 flow += f;  
 }  
}  
// Dinic  
void bfs(int s) {  
 memset(level, -1, sizeof(level));  
 queue<int> que;  
 level[s] = 0; que.push(s);  
 while (!que.empty()) {  
 int v = que.front(); que.pop();  
 for (int i = 0; i < G[v].size(); ++i) {  
 edge &e = G[v][i];  
 if (e.cap > 0 && level[e.to] < 0) {  
 level[e.to] = level[v] + 1;  
 que.push(e.to);  
 }  
 }  
 }  
}  
int dfs(int v, int t, int f) {  
 if (v == t) return f;  
 for (int &i = iter[v]; i < G[v].size(); ++i) {  
 edge &e = G[v][i];  
 if (e.cap > 0 && level[v] < level[e.to]) {  
 int d = dfs(e.to, t, min(f, e.cap));  
 if (d > 0) {  
 e.cap -= d;  
 G[e.to][e.rev].cap += d;  
 return d;  
 }  
 }  
 }  
 return 0;  
}  
int max\_flow(int s, int t) {  
 int flow = 0;  
 for (;;) {  
 bfs(s);  
 if (level[t] < 0) return flow;  
 memset(iter, 0, sizeof iter);  
 int f;  
 while ((f = dfs(s, t, INF)) > 0) {  
 flow += f;  
 }  
 }  
}

ISAP  
// UVa11248 Frequency Hopping：使用ISAP算法，加优化  
// Rujia Liu  
#include<cstdio>  
#include<cstring>  
#include<queue>  
#include<vector>  
#include<algorithm>  
using namespace std;  
  
const int maxn = 100 + 10;  
const int INF = 1000000000;  
  
struct Edge {  
 int from, to, cap, flow;  
};  
  
bool operator < (const Edge& a, const Edge& b) {  
 return a.from < b.from || (a.from == b.from && a.to < b.to);  
}  
  
struct ISAP {  
 int n, m, s, t;  
 vector<Edge> edges;  
 vector<int> G[maxn]; // 邻接表，G[i][j]表示结点i的第j条边在e数组中的序号  
 bool vis[maxn]; // BFS使用  
 int d[maxn]; // 从起点到i的距离  
 int cur[maxn]; // 当前弧指针  
 int p[maxn]; // 可增广路上的上一条弧  
 int num[maxn]; // 距离标号计数  
  
 void AddEdge(int from, int to, int cap) {  
 edges.push\_back((Edge){from, to, cap, 0});  
 edges.push\_back((Edge){to, from, 0, 0});  
 m = edges.size();  
 G[from].push\_back(m-2);  
 G[to].push\_back(m-1);  
 }  
  
 bool BFS() {  
 memset(vis, 0, sizeof(vis));  
 queue<int> Q;  
 Q.push(t);  
 vis[t] = 1;  
 d[t] = 0;  
 while(!Q.empty()) {  
 int x = Q.front(); Q.pop();  
 for(int i = 0; i < G[x].size(); i++) {  
 Edge& e = edges[G[x][i]^1];  
 if(!vis[e.from] && e.cap > e.flow) {  
 vis[e.from] = 1;  
 d[e.from] = d[x] + 1;  
 Q.push(e.from);  
 }  
 }  
 }  
 return vis[s];  
 }  
  
 void ClearAll(int n) {  
 this->n = n;  
 for(int i = 0; i < n; i++) G[i].clear();  
 edges.clear();  
 }  
  
 void ClearFlow() {  
 for(int i = 0; i < edges.size(); i++) edges[i].flow = 0;  
 }  
  
 int Augment() {  
 int x = t, a = INF;  
 while(x != s) {  
 Edge& e = edges[p[x]];  
 a = min(a, e.cap-e.flow);  
 x = edges[p[x]].from;  
 }  
 x = t;  
 while(x != s) {  
 edges[p[x]].flow += a;  
 edges[p[x]^1].flow -= a;  
 x = edges[p[x]].from;  
 }  
 return a;  
 }  
  
 int Maxflow(int s, int t, int need) {  
 this->s = s; this->t = t;  
 int flow = 0;  
 BFS();  
 memset(num, 0, sizeof(num));  
 for(int i = 0; i < n; i++) num[d[i]]++;  
 int x = s;  
 memset(cur, 0, sizeof(cur));  
 while(d[s] < n) {  
 if(x == t) {  
 flow += Augment();  
 if(flow >= need) return flow;  
 x = s;  
 }  
 int ok = 0;  
 for(int i = cur[x]; i < G[x].size(); i++) {  
 Edge& e = edges[G[x][i]];  
 if(e.cap > e.flow && d[x] == d[e.to] + 1) { // Advance  
 ok = 1;  
 p[e.to] = G[x][i];  
 cur[x] = i; // 注意  
 x = e.to;  
 break;  
 }  
 }  
 if(!ok) { // Retreat  
 int m = n-1; // 初值注意  
 for(int i = 0; i < G[x].size(); i++) {  
 Edge& e = edges[G[x][i]];  
 if(e.cap > e.flow) m = min(m, d[e.to]);  
 }  
 if(--num[d[x]] == 0) break;  
 num[d[x] = m+1]++;  
 cur[x] = 0; // 注意  
 if(x != s) x = edges[p[x]].from;  
 }  
 }  
 return flow;  
 }  
  
 vector<int> Mincut() { // call this after maxflow  
 BFS();  
 vector<int> ans;  
 for(int i = 0; i < edges.size(); i++) {  
 Edge& e = edges[i];  
 if(!vis[e.from] && vis[e.to] && e.cap > 0) ans.push\_back(i);  
 }  
 return ans;  
 }  
  
 void Reduce() {  
 for(int i = 0; i < edges.size(); i++) edges[i].cap -= edges[i].flow;  
 }  
  
 void print() {  
 printf("Graph:\n");  
 for(int i = 0; i < edges.size(); i++)  
 printf("%d->%d, %d, %d\n", edges[i].from, edges[i].to , edges[i].cap, edges[i].flow);  
 }  
};  
  
  
ISAP g;  
  
int main() {  
 int n, e, c, kase = 0;  
 while(scanf("%d%d%d", &n, &e, &c) == 3 && n) {  
 g.ClearAll(n);  
 while(e--) {  
 int b1, b2, fp;  
 scanf("%d%d%d", &b1, &b2, &fp);  
 g.AddEdge(b1-1, b2-1, fp);  
 }  
 int flow = g.Maxflow(0, n-1, INF);  
 printf("Case %d: ", ++kase);  
 if(flow >= c) printf("possible\n");  
 else {  
 vector<int> cut = g.Mincut();  
 g.Reduce();  
 vector<Edge> ans;  
 for(int i = 0; i < cut.size(); i++) {  
 Edge& e = g.edges[cut[i]];  
 e.cap = c;  
 g.ClearFlow();  
 if(flow + g.Maxflow(0, n-1, c-flow) >= c) ans.push\_back(e);  
 e.cap = 0;  
 }  
 if(ans.empty()) printf("not possible\n");  
 else {  
 sort(ans.begin(), ans.end());  
 printf("possible option:(%d,%d)", ans[0].from+1, ans[0].to+1);  
 for(int i = 1; i < ans.size(); i++)  
 printf(",(%d,%d)", ans[i].from+1, ans[i].to+1);  
 printf("\n");  
 }  
 }  
 }  
 return 0;  
}

// min\_cost\_flow  
void add\_edge(int from, int to, int cap, int cost) {  
 G[from].push\_back((edge){to, cap, cost, (int)G[to].size()});  
 G[to].push\_back((edge){from, 0, -cost, (int)G[from].size() - 1});  
}  
int min\_cost\_flow(int s, int t, int f) {  
 int res = 0;  
 fill(h, h + V, 0);  
 while (f > 0) {  
 priority\_queue<Pii, vector<Pii>, greater<Pii> > que;  
 fill(dist, dist + V, INF);  
 dist[s] = 0; que.push(Pii(0, s));  
 while (!que.empty()) {  
 Pii p = que.top(); que.pop();  
 int v = p.second;  
 if (dist[v] < p.first) continue;  
 for (int i = 0; i < (int)G[v].size(); i++) {  
 edge &e = G[v][i];  
 if (e.cap > 0 && dist[e.to] > dist[v] + e.cost + h[v] - h[e.to]) {  
 dist[e.to] = dist[v] + e.cost + h[v] - h[e.to];  
 prevv[e.to] = v;  
 preve[e.to] = i;  
 que.push(Pii(dist[e.to], e.to));  
 }  
 }  
 }  
 if (dist[t] == INF) return -1;  
 for (int v = 0; v < V; v++) h[v] += dist[v];  
 int d = f;  
 for (int v = t; v != s; v = prevv[v])  
 d = min(d, G[prevv[v]][preve[v]].cap);  
 f -= d;  
 res += d \* h[t];  
 for (int v = t; v != s; v = prevv[v]) {  
 edge &e = G[prevv[v]][preve[v]];  
 e.cap -= d;  
 G[v][e.rev].cap += d;  
 }  
 }  
 return res;  
}

// stoer\_wagner 全局最小割  
void search() {  
 memset(vis, false, sizeof vis);  
 memset(wet, 0, sizeof wet);  
 S = T = -1;  
 int imax, tmp;  
 for (int i = 0; i < V; i++) {  
 imax = -INF;  
 for (int j = 0; j < V; j++)  
 if (!cmb[j] && !vis[j] && wet[j] > imax) {  
 imax = wet[j];  
 tmp = j;  
 }  
 if (T == tmp) return;  
 S = T; T = tmp;  
 mc = imax;  
 vis[tmp] = true;  
 for (int j = 0; j < V; j++)  
 if (!cmb[j] && !vis[j])  
 wet[j] += G[tmp][j];  
 }  
}  
int stoer\_wagner() {  
 memset(cmb, false, sizeof cmb);  
 int ans = INF;  
 for (int i = 0; i < V - 1; i++) {  
 search();  
 ans = min(ans, mc);  
 if (ans == 0) return 0;  
 cmb[T] = true;  
 for (int j = 0; j < V; j++)  
 if (!cmb[j]) {  
 G[S][j] += G[T][j];  
 G[j][S] += G[j][T];  
 }  
 }  
 return ans;  
}

// LCA--Doubling  
const int MAX\_LOG\_V = 32 - \_\_builtin\_clz(MAX\_V);  
  
vector<int> G[MAX\_V];  
int root, parent[MAX\_LOG\_V][MAX\_V], depth[MAX\_V];  
  
void dfs(int v, int p, int d) {  
 parent[0][v] = p;  
 depth[v] = d;  
 for (int i = 0; i < G[v].size(); i++)  
 if (G[v][i] != p) dfs(G[v][i], v, d + 1);  
}  
void init(int V) {  
 dfs(root, -1, 0);  
 for (int k = 0; k + 1 < MAX\_LOG\_V; k++)  
 for (int v = 0; v < V; v++)  
 if (parent[k][v] < 0) parent[k + 1][v] = -1;  
 else parent[k + 1][v] = parent[k][parent[k][v]];  
}  
int lca(int u, int v) {  
 if (depth[u] > depth[v]) swap(u, v);  
 for (int k = 0; k < MAX\_LOG\_V; k++)  
 if ((depth[v] - depth[u]) >> k & 1)  
 v = parent[k][v];  
 if (u == v) return u;  
 for (int k = MAX\_LOG\_V - 1; k >= 0; k--)  
 if (parent[k][u] != parent[k][v])  
 u = parent[k][u], v = parent[k][v];  
 return parent[0][u];  
}  
// LCA--RMQ  
vector<int> G[MAX\_V];  
int root, vs[MAX\_V \* 2 - 1], depth[MAX\_V \* 2 - 1], id[MAX\_V];  
  
int ST[2 \* MAX\_V][MAX\_K];  
void rmq\_init(int\* A, int N) {  
 for (int i = 0; i < N; i++)  
 ST[i][0] = i;  
 int k = 31 - \_\_builtin\_clz(N);  
 for (int j = 1; j <= k; j++)  
 for (int i = 0; i <= N - (1 << j); ++i)  
 if (A[ST[i][j - 1]] <= A[ST[i + (1 << (j - 1))][j - 1]])  
 ST[i][j] = ST[i][j - 1];  
 else ST[i][j] = ST[i + (1 << (j - 1))][j - 1];  
}  
int query(int l, int r) {  
 if (l >= r) return -1;  
 int k = 31 - \_\_builtin\_clz(r - l);  
 return (depth[ST[l][k]] <= depth[ST[r - (1 << k)][k]]) ? ST[l][k] : ST[r - (1 << k)][k];  
}  
void dfs(int v, int p, int d, int &k) {  
 id[v] = k;  
 vs[k] = v;  
 depth[k++] = d;  
 for (int i = 0; i < G[v].size(); i++) {  
 if (G[v][i] != p) {  
 dfs(G[v][i], v, d + 1, k);  
 vs[k] = v;  
 depth[k++] = d;  
 }  
 }  
}  
void init(int V) {  
 int k = 0;  
 dfs(root, -1, 0, k);  
 rmq\_init(depth, 2 \* V - 1);  
}  
int lca(int u, int v) {  
 return vs[query(min(id[u], id[v]), max(id[u], id[v]) + 1)];  
}

## Computational Geometry

const double eps = 1e-10;  
int sgn(double x) { return x < -eps ? -1 : x > eps ? 1 : 0;}  
inline double add(double a, double b) {  
 if (abs(a + b) < eps \* (abs(a) + abs(b))) return 0;  
 return a + b;  
};  
struct Point {  
 double x, y;  
 Point(double x = 0, double y = 0) : x(x), y(y) {}  
 Point operator + (Point p) { return Point(x + p.x, y + p.y); }  
 Point operator - (Point p) { return Point(x - p.x, y - p.y); }  
 Point operator \* (double d) { return Point(x \* d, y \* d); }  
 bool operator < (Point p) const { return x != p.x? x < p.x : y < p.y; }  
 double dot(Point p) { return add(x \* p.x, y \* p.y); }// 内积  
 double det(Point p) { return add(x \* p.y, -y \* p.x); }// 外积  
 Point ver() { return Point(-y, x); }  
};  
bool on\_seg(Point p1, Point p2, Point q) {  
 return sgn((p1 - q).det(p2 - q)) == 0 && sgn((p1 - q).dot(p2 - q)) <= 0;  
}  
Point intersection(Point p1, Point p2, Point q1, Point q2) {  
 // 判断是否相交  
 return p1 + (p2 - p1) \* ((q2 - q1).det(q1 - p1) / (q2 - q1).det(p2 - p1));  
}  
// 凸包  
int convex\_hull(Point \*ps, int n, Point \*ch) {  
 sort(ps, ps + n);  
 int k = 0;  
 for (int i = 0; i < n; ++i) {  
 while (k > 1 && (ch[k - 1] - ch[k - 2]).det(ps[i] - ch[k - 1]) <= 0) k--;  
 ch[k++] = ps[i];  
 }  
 for (int i = n - 2, t = k; i >= 0; --i) {  
 while (k > t && (ch[k - 1] - ch[k - 2]).det(ps[i] - ch[k - 1]) <= 0) k--;  
 ch[k++] = ps[i];  
 }  
 return k - 1;  
}

// UVa11275 3D Triangles  
// Rujia Liu  
#include<cstdio>  
#include<cmath>  
using namespace std;  
  
struct Point3 {  
 double x, y, z;  
 Point3(double x=0, double y=0, double z=0):x(x),y(y),z(z) { }  
};  
  
typedef Point3 Vector3;  
  
Vector3 operator + (const Vector3& A, const Vector3& B) { return Vector3(A.x+B.x, A.y+B.y, A.z+B.z); }  
Vector3 operator - (const Point3& A, const Point3& B) { return Vector3(A.x-B.x, A.y-B.y, A.z-B.z); }  
Vector3 operator \* (const Vector3& A, double p) { return Vector3(A.x\*p, A.y\*p, A.z\*p); }  
Vector3 operator / (const Vector3& A, double p) { return Vector3(A.x/p, A.y/p, A.z/p); }  
  
const double eps = 1e-8;  
int dcmp(double x) {  
 if(fabs(x) < eps) return 0; else return x < 0 ? -1 : 1;  
}  
  
double Dot(const Vector3& A, const Vector3& B) { return A.x\*B.x + A.y\*B.y + A.z\*B.z; }  
double Length(const Vector3& A) { return sqrt(Dot(A, A)); }  
double Angle(const Vector3& A, const Vector3& B) { return acos(Dot(A, B) / Length(A) / Length(B)); }  
Vector3 Cross(const Vector3& A, const Vector3& B) { return Vector3(A.y\*B.z - A.z\*B.y, A.z\*B.x - A.x\*B.z, A.x\*B.y - A.y\*B.x); }  
double Area2(const Point3& A, const Point3& B, const Point3& C) { return Length(Cross(B-A, C-A)); }  
  
Point3 read\_point3() {  
 Point3 p;  
 scanf("%lf%lf%lf", &p.x, &p.y, &p.z);  
 return p;  
}  
  
// p1和p2是否在线段a-b的同侧  
bool SameSide(const Point3& p1, const Point3& p2, const Point3& a, const Point3& b) {  
 return dcmp(Dot(Cross(b-a, p1-a), Cross(b-a, p2-a))) >= 0;  
}  
  
// 点在三角形P0, P1, P2中  
bool PointInTri(const Point3& P, const Point3& P0, const Point3& P1, const Point3& P2) {  
 return SameSide(P, P0, P1, P2) && SameSide(P, P1, P0, P2) && SameSide(P, P2, P0, P1);  
}  
  
// 三角形P0P1P2是否和线段AB相交  
bool TriSegIntersection(const Point3& P0, const Point3& P1, const Point3& P2, const Point3& A, const Point3& B, Point3& P) {  
 Vector3 n = Cross(P1-P0, P2-P0);  
 if(dcmp(Dot(n, B-A)) == 0) return false; // 线段A-B和平面P0P1P2平行或共面  
 else { // 平面A和直线P1-P2有惟一交点  
 double t = Dot(n, P0-A) / Dot(n, B-A);  
 if(dcmp(t) < 0 || dcmp(t-1) > 0) return false; // 不在线段AB上  
 P = A + (B-A)\*t; // 交点  
 return PointInTri(P, P0, P1, P2);  
 }  
}  
  
bool TriTriIntersection(Point3\* T1, Point3\* T2) {  
 Point3 P;  
 for(int i = 0; i < 3; i++) {  
 if(TriSegIntersection(T1[0], T1[1], T1[2], T2[i], T2[(i+1)%3], P)) return true;  
 if(TriSegIntersection(T2[0], T2[1], T2[2], T1[i], T1[(i+1)%3], P)) return true;  
 }  
 return false;  
}  
  
int main() {  
 int T;  
 scanf("%d", &T);  
 while(T--) {  
 Point3 T1[3], T2[3];  
 for(int i = 0; i < 3; i++) T1[i] = read\_point3();  
 for(int i = 0; i < 3; i++) T2[i] = read\_point3();  
 printf("%d\n", TriTriIntersection(T1, T2) ? 1 : 0);  
 }  
 return 0;  
}

// LA3218/UVa1340 Find the Border  
// Rujia Liu  
// 注意：本题可以直接使用“卷包裹”法求出外轮廓。本程序只是为了演示PSLG的实现  
#include<cstdio>  
#include<vector>  
#include<cmath>  
#include<algorithm>  
#include<cstring>  
#include<cassert>  
using namespace std;  
  
const double eps = 1e-8;  
double dcmp(double x) {  
 if(fabs(x) < eps) return 0; else return x < 0 ? -1 : 1;  
}  
  
struct Point {  
 double x, y;  
 Point(double x=0, double y=0):x(x),y(y) { }  
};  
  
typedef Point Vector;  
  
Vector operator + (Vector A, Vector B) {  
 return Vector(A.x+B.x, A.y+B.y);  
}  
  
Vector operator - (Point A, Point B) {  
 return Vector(A.x-B.x, A.y-B.y);  
}  
  
Vector operator \* (Vector A, double p) {  
 return Vector(A.x\*p, A.y\*p);  
}  
  
// 理论上这个“小于”运算符是错的，因为可能有三个点a, b, c, a和b很接近（即a<b好b<a都不成立），b和c很接近，但a和c不接近  
// 所以使用这种“小于”运算符的前提是能排除上述情况  
bool operator < (const Point& a, const Point& b) {  
 return dcmp(a.x - b.x) < 0 || (dcmp(a.x - b.x) == 0 && dcmp(a.y - b.y) < 0);  
}  
  
bool operator == (const Point& a, const Point &b) {  
 return dcmp(a.x-b.x) == 0 && dcmp(a.y-b.y) == 0;  
}  
  
double Dot(Vector A, Vector B) { return A.x\*B.x + A.y\*B.y; }  
double Cross(Vector A, Vector B) { return A.x\*B.y - A.y\*B.x; }  
double Length(Vector A) { return sqrt(Dot(A, A)); }  
  
typedef vector<Point> Polygon;  
  
Point GetLineIntersection(const Point& P, const Vector& v, const Point& Q, const Vector& w) {  
 Vector u = P-Q;  
 double t = Cross(w, u) / Cross(v, w);  
 return P+v\*t;  
}  
  
bool SegmentProperIntersection(const Point& a1, const Point& a2, const Point& b1, const Point& b2) {  
 double c1 = Cross(a2-a1,b1-a1), c2 = Cross(a2-a1,b2-a1),  
 c3 = Cross(b2-b1,a1-b1), c4=Cross(b2-b1,a2-b1);  
 return dcmp(c1)\*dcmp(c2)<0 && dcmp(c3)\*dcmp(c4)<0;  
}  
  
bool OnSegment(Point p, Point a1, Point a2) {  
 return dcmp(Cross(a1-p, a2-p)) == 0 && dcmp(Dot(a1-p, a2-p)) < 0;  
}  
  
// 多边形的有向面积  
double PolygonArea(Polygon poly) {  
 double area = 0;  
 int n = poly.size();  
 for(int i = 1; i < n-1; i++)  
 area += Cross(poly[i]-poly[0], poly[(i+1)%n]-poly[0]);  
 return area/2;  
}  
  
struct Edge {  
 int from, to; // 起点，终点，左边的面编号  
 double ang;  
};  
  
const int maxn = 10000 + 10; // 最大边数  
  
// 平面直线图（PSGL）实现  
struct PSLG {  
 int n, m, face\_cnt;  
 double x[maxn], y[maxn];  
 vector<Edge> edges;  
 vector<int> G[maxn];  
 int vis[maxn\*2]; // 每条边是否已经访问过  
 int left[maxn\*2]; // 左面的编号  
 int prev[maxn\*2]; // 相同起点的上一条边（即顺时针旋转碰到的下一条边）的编号  
  
 vector<Polygon> faces;  
 double area[maxn]; // 每个polygon的面积  
  
 void init(int n) {  
 this->n = n;  
 for(int i = 0; i < n; i++) G[i].clear();  
 edges.clear();  
 faces.clear();  
 }  
  
 // 有向线段from->to的极角  
 double getAngle(int from, int to) {  
 return atan2(y[to]-y[from], x[to]-x[from]);  
 }  
  
 void AddEdge(int from, int to) {  
 edges.push\_back((Edge){from, to, getAngle(from, to)});  
 edges.push\_back((Edge){to, from, getAngle(to, from)});  
 m = edges.size();  
 G[from].push\_back(m-2);  
 G[to].push\_back(m-1);  
 }  
  
 // 找出faces并计算面积  
 void Build() {  
 for(int u = 0; u < n; u++) {  
 // 给从u出发的各条边按极角排序  
 int d = G[u].size();  
 for(int i = 0; i < d; i++)  
 for(int j = i+1; j < d; j++) // 这里偷个懒，假设从每个点出发的线段不会太多  
 if(edges[G[u][i]].ang > edges[G[u][j]].ang) swap(G[u][i], G[u][j]);  
 for(int i = 0; i < d; i++)  
 prev[G[u][(i+1)%d]] = G[u][i];  
 }  
  
 memset(vis, 0, sizeof(vis));  
 face\_cnt = 0;  
 for(int u = 0; u < n; u++)  
 for(int i = 0; i < G[u].size(); i++) {  
 int e = G[u][i];  
 if(!vis[e]) { // 逆时针找圈  
 face\_cnt++;  
 Polygon poly;  
 for(;;) {  
 vis[e] = 1; left[e] = face\_cnt;  
 int from = edges[e].from;  
 poly.push\_back(Point(x[from], y[from]));  
 e = prev[e^1];  
 if(e == G[u][i]) break;  
 assert(vis[e] == 0);  
 }  
 faces.push\_back(poly);  
 }  
 }  
  
 for(int i = 0; i < faces.size(); i++) {  
 area[i] = PolygonArea(faces[i]);  
 }  
 }  
};  
  
PSLG g;  
  
const int maxp = 100 + 5;  
int n, c;  
Point P[maxp];  
  
Point V[maxp\*(maxp-1)/2+maxp];  
  
// 在V数组里找到点p  
int ID(Point p) {  
 return lower\_bound(V, V+c, p) - V;  
}  
  
// 假定poly没有相邻点重合的情况，只需要删除三点共线的情况  
Polygon simplify(const Polygon& poly) {  
 Polygon ans;  
 int n = poly.size();  
 for(int i = 0; i < n; i++) {  
 Point a = poly[i];  
 Point b = poly[(i+1)%n];  
 Point c = poly[(i+2)%n];  
 if(dcmp(Cross(a-b, c-b)) != 0) ans.push\_back(b);  
 }  
 return ans;  
}  
  
void build\_graph() {  
 c = n;  
 for(int i = 0; i < n; i++)  
 V[i] = P[i];  
  
 vector<double> dist[maxp]; // dist[i][j]是第i条线段上的第j个点离起点（P[i]）的距离  
 for(int i = 0; i < n; i++)  
 for(int j = i+1; j < n; j++)  
 if(SegmentProperIntersection(P[i], P[(i+1)%n], P[j], P[(j+1)%n])) {  
 Point p = GetLineIntersection(P[i], P[(i+1)%n]-P[i], P[j], P[(j+1)%n]-P[j]);  
 V[c++] = p;  
 dist[i].push\_back(Length(p - P[i]));  
 dist[j].push\_back(Length(p - P[j]));  
 }  
  
 // 为了保证“很接近的点”被看作同一个，这里使用了sort+unique的方法  
 // 必须使用前面提到的“理论上是错误”的小于运算符，否则不能保证“很接近的点”在排序后连续排列  
 // 另一个常见的处理方式是把坐标扩大很多倍（比如100000倍），然后四舍五入变成整点（计算完毕后再还原），用少许的精度损失换来鲁棒性和速度。  
 sort(V, V+c);  
 c = unique(V, V+c) - V;  
  
 g.init(c); // c是平面图的点数  
 for(int i = 0; i < c; i++) {  
 g.x[i] = V[i].x;  
 g.y[i] = V[i].y;  
 }  
 for(int i = 0; i < n; i++) {  
 Vector v = P[(i+1)%n] - P[i];  
 double len = Length(v);  
 dist[i].push\_back(0);  
 dist[i].push\_back(len);  
 sort(dist[i].begin(), dist[i].end());  
 int sz = dist[i].size();  
 for(int j = 1; j < sz; j++) {  
 Point a = P[i] + v \* (dist[i][j-1] / len);  
 Point b = P[i] + v \* (dist[i][j] / len);  
 if(a == b) continue;  
 g.AddEdge(ID(a), ID(b));  
 }  
 }  
  
 g.Build();  
  
 Polygon poly;  
 for(int i = 0; i < g.faces.size(); i++)  
 if(g.area[i] < 0) { // 对于连通图，惟一一个面积小于零的面是无限面  
 poly = g.faces[i];  
 reverse(poly.begin(), poly.end()); // 对于内部区域来说，无限面多边形的各个顶点是顺时针的  
 poly = simplify(poly); // 无限面多边形上可能会有相邻共线点  
 break;  
 }  
  
 int m = poly.size();  
 printf("%d\n", m);  
  
 // 挑选坐标最小的点作为输出的起点  
 int start = 0;  
 for(int i = 0; i < m; i++)  
 if(poly[i] < poly[start]) start = i;  
 for(int i = start; i < m; i++)  
 printf("%.4lf %.4lf\n", poly[i].x, poly[i].y);  
 for(int i = 0; i < start; i++)  
 printf("%.4lf %.4lf\n", poly[i].x, poly[i].y);  
}  
  
int main() {  
 while(scanf("%d", &n) == 1 && n) {  
 for(int i = 0; i < n; i++) {  
 int x, y;  
 scanf("%d%d", &x, &y);  
 P[i] = Point(x, y);  
 }  
 build\_graph();  
 }  
 return 0;  
}

## Math Problem

// returning count of nk in range [l, r], from Infinity  
template<typename T> T mps(T l, T r, T k) {  
 return ((r - (r % k + k) % k) - (l + (k - l % k) % k)) / k + 1;  
}  
template<typename T> T gcd(T a, T b) {  
 //return (b)? gcd(b, a % b) : a;  
 while (b) { T t = a % b; a = b; b = t; } return a;  
}  
template<typename T> T lcm(T a, T b) {  
 return a / gcd(a, b) \* b;  
}  
// find (x, y) s.t. a x + b y = gcd(a, b) = d  
template<typename T> T exgcd(T a, T b, T &x, T &y) {  
 T d = a;  
 if (b) {  
 d = exgcd(b, a % b, y, x);  
 y -= a / b \* x;  
 } else {  
 x = 1; y = 0;  
 }  
 return d;  
}  
template<typename T> T modular\_linear(T a, T b, T n) {  
 T d, e, x, y;  
 d = exgcd(a, n, x, y);  
 if (b % d)  
 return -1;  
 e = x \* (b / d) % n + n;  
 return e % (n / d);  
}  
template<typename T> T mod\_mult(T a, T b, T mod) {  
 T res = 0;  
 while (b) {  
 if (b & 1) {  
 res = (res + a) % mod;  
 // res += a;  
 // if (res >= mod) res -= mod;  
 }  
 a = (a + a) % mod;  
 // a <<= 1;  
 // if (a >= mod) a -= mod;  
 b >>= 1;  
 }  
 return res;  
}  
template<typename T> T mod\_pow(T x, T n, T mod) {  
 T res = 1;  
 while (n) {  
 if (n & 1) res = mod\_mult(res, x, mod);  
 x = mod\_mult(x, x, mod);  
 n >>= 1;  
 }  
 return res;  
 // return b ? mod\_pow(a \* a % mod, b >> 1, mod) \* (b & 1 ? a : 1) % mod : 1;  
}  
template<typename T> T mod\_inverse(T a, T m) {  
 T x, y;  
 exgcd(a, m, x, y);  
 return (m + x % m) % m;  
}  
template<typename T> T mod\_inv(T x, T mod) {  
 return x == 1 ? 1 : (mod - (mod / x) \* inv(mod % x) % mod) % mod;  
}  
void init\_inverse() {  
 inv[1] = 1;  
 for (int i = 2; i < maxn; i++) inv[i] = (MOD - (MOD / i) \* inv[MOD % i] % MOD) % MOD;  
}  
//A[i] \* x % M[i] = B[i];  
std::pair<int, int> linear\_congruence(const std::vector<int> &A, const std::vector<int> &B, const std::vector<int> &M) {  
 // wa 了把中间量开大？\* 溢出  
 int x = 0, m = 1;  
 for(int i = 0; i < A.size(); i++) {  
 int a = A[i] \* m, b = B[i] - A[i] \* x, d = gcd(M[i], a);  
 if(b % d != 0) return std::make\_pair(0, -1); // no solutioin  
 int t = b / d \* mod\_inverse(a / d, M[i] / d) % (M[i] / d);  
 x = x + m \* t;  
 m \*= M[i] / d;  
 }  
 while (x < m) x += m;  
 return std::make\_pair(x % m, m);  
}  
ll CRT(vector<ll> &a, vector<ll> &m) {  
 ll M = 1LL, res = 0;  
 for (int i = 0; i < m.size(); ++i)  
 M \*= m[i];  
 for (int i = 0; i < m.size(); ++i) {  
 ll Mi, Ti;  
 Mi = M / m[i]; Ti = mod\_inverse(Mi, mi);  
 res = (res + a[i] \* (Mi \* Ti % M) % M) % M;  
 }  
 return res;  
}  
ll fact[maxn + 10], iact[maxn + 10];  
void init() {  
 fact[0] = 1;  
 for (int i = 1; i < maxn; ++i)  
 fact[i] = fact[i - 1] \* i % MOD;  
 iact[maxn - 1] = mod\_pow(fact[maxn - 1], mod - 2, mod);  
 for (int i = maxn - 2; i >= 0; --i)  
 iact[i] = iact[i + 1] \* (i + 1) % mod;  
}  
int mod\_fact(int n, int p, int &e) {  
 e = 0;  
 if (n == 0) return 1;  
 int res = mod\_fact(n / p, p, e);  
 e += n / p;  
 if (n / p % 2 != 0) return res \* (p - fact[n % p]) % p;  
 return res \* fact[n % p] % p;  
}  
int mod\_comb(int n, int k, int p) {  
 if (n < 0 || k < 0 || n < k) return 0;  
 if (n == 0) return 1;  
 int e1, e2, e3;  
 int a1 = mod\_fact(n, p, e1), a2 = mod\_fact(k, p, e2), a3 = mod\_fact(n - k, p, e3);  
 if (e1 > e2 + e3) return 0;  
 return a1 \* mod\_inverse(a2 \* a3 % p, p) % p;  
}  
ll lucas(ll n, ll k, const ll &p) {  
 if (n < 0 || k < 0 || n < k) return 0;  
 if (n == 0) return 1;  
 return lucas(n / p, k / p, p) \* mod\_comb(n % p, k % p, p) % p;  
}

// 矩阵快速幂  
typedef vector<int> vec;  
typedef vector<vec> mat;  
mat G(maxn);  
  
mat mat\_mul(mat &A, mat &B) {  
 mat C(A.size(), vec(B[0].size()));  
 for (int i = 0; i < A.size(); ++i)  
 for (int k = 0; k < B.size(); ++k)  
 for (int j = 0; j < B[0].size(); ++j)  
 C[i][j] = (C[i][j] + A[i][k] % MOD \* B[k][j] % MOD + MOD) % MOD;  
 return C;  
}  
mat mat\_pow(mat A, ll n) {  
 mat B(A.size(), vec(A.size()));  
 for (int i = 0; i < A.size(); ++i)  
 B[i][i] = 1;  
 while (n > 0) {  
 if (n & 1) B = mat\_mul(B, A);  
 A = mat\_mul(A, A);  
 n >>= 1;  
 }  
 return B;  
}

// prime number  
bool is\_prime(int n) {  
 for (int i = 2; i \* i <= n; ++i)  
 if (n % i == 0) return false;  
 return n != 1;  
}  
vector<int> divisor(int n) {  
 vector<int> res;  
 for (int i = 1; i \* i <= n; ++i) {  
 if (n % i == 0) {  
 res.push\_back(i);  
 if (i != n / i) res.push\_back(n / i);  
 }  
 }  
 return res;  
}  
map<int, int> prime\_factor(int n) {  
 map<int, int> res;  
 for (int i = 2; i \* i <= n; ++i) {  
 while (n % i == 0) {  
 ++res[i];  
 n /= i;  
 }  
 }  
 if (n != 1) res[n] = 1;  
 return res;  
}  
int prime[maxn];  
bool isPrime[maxn + 1];  
int seive(int n) {  
 int p = 0;  
 fill(isPrime, isPrime + n + 1, true);  
 isPrime[0] = isPrime[1] = false;  
 for (int i = 2; i <= n; ++i)  
 if (isPrime[i]) {  
 prime[p++] = i;  
 for (int j = 2 \* i; j <= n; j += i) isPrime[j] = false;  
 }  
 return p;  
}  
// the number of prime in [L, r)  
// 对区间 [l, r）内的整数执行筛法，prime[i - l] = true <=> i 是素数  
bool segPrimeSmall[MAX\_L];  
bool segPrime[MAX\_SQRT\_R];  
void segment\_sieve(ll l, ll r) {  
 for (int i = 0; (ll)i \* i < r; ++i) segPrimeSmall[i] = true;  
 for (int i = 0; i < r - l; ++i) segPrime[i] = true;  
 for (int i = 2; (ll)i \* i < r; ++i) {  
 if (segPrimeSmall[i]) {  
 for (int j = 2 \* i; (ll)j \* j <= r; j += i) segPrimeSmall[j] = false;  
 for (ll j = max(2ll, (l + i - 1) / i) \* i; j < r; j += i) segPrime[j - l] = false;  
 }  
 }  
}  
// Miller\_Rabin  
bool check(ll a, ll n, ll x, ll t) {  
 ll res = mod\_pow(a, x, n);  
 ll last = res;  
 for (int i = 1; i <= t; ++i) {  
 res = mod\_mult(res, res, n);  
 if (res == 1 && last != 1 && last != n - 1) return true;  
 last = res;  
 }  
 if (res != 1) return true;  
 return false;  
}  
bool Miller\_Rabin(ll n) {  
 if (n < maxn) return isPrime[n]; // small number may get wrong answer?!  
 if (n < 2) return false;  
 if (n == 2) return true;  
 if ((n & 1) == 0) return false;  
 ll x = n - 1, t = 0;  
 while ((x & 1) == 0) {  
 x >>= 1;  
 ++t;  
 }  
 for (int i = 0; i < S; ++i) {  
 ll a = rand() % (n - 1) + 1;  
 if (check(a, n, x, t))  
 return false;  
 }  
 return true;  
}  
// find factors  
vector<ll> factor;  
ll Pollard\_rho(ll x, ll c) {  
 ll i = 1, k = 2;  
 ll x0 = rand() % x;  
 ll y = x0;  
 while (true) {  
 ++i;  
 x0 = (mod\_mult(x0, x0, x) + c) % x;  
 ll d;  
 if (y == x0) d = 1;  
 else  
 if (y > x0)  
 d = gcd(y - x0, x);  
 else d = gcd(x0 - y, x);  
 if (d != 1 && d != x) return d;  
 if (y == x0) return x;  
 if (i == k) {  
 y = x0;  
 k += k;  
 }  
 }  
}  
void find\_factor(ll n) {  
 if (n == 1) return ;  
 if (Miller\_Rabin(n)) {  
 factor.push\_back(n);  
 return ;  
 }  
 ll p = n;  
 while (p >= n) p = Pollard\_rho(p, rand() % (n - 1) + 1);  
 find\_factor(p);  
 find\_factor(n / p);  
}

#include<bits/stdc++>  
//Meisell-Lehmer  
const int maxn = 5e6 + 2;  
bool np[maxn];  
int prime[maxn], pi[maxn];  
int getprime()  
{  
 int cnt = 0;  
 np[0] = np[1] = true;  
 pi[0] = pi[1] = 0;  
 for(int i = 2; i < maxn; ++i)  
 {  
 if(!np[i]) prime[++cnt] = i;  
 pi[i] = cnt;  
 for(int j = 1; j <= cnt && i \* prime[j] < maxn; ++j)  
 {  
 np[i \* prime[j]] = true;  
 if(i % prime[j] == 0) break;  
 }  
 }  
 return cnt;  
}  
const int M = 7;  
const int PM = 2 \* 3 \* 5 \* 7 \* 11 \* 13 \* 17;  
int phi[PM + 1][M + 1], sz[M + 1];  
void init() {  
 getprime();  
 sz[0] = 1;  
 for(int i = 0; i <= PM; ++i) phi[i][0] = i;  
 for(int i = 1; i <= M; ++i) {  
 sz[i] = prime[i] \* sz[i - 1];  
 for(int j = 1; j <= PM; ++j) phi[j][i] = phi[j][i - 1] - phi[j / prime[i]][i - 1];  
 }  
}  
int sqrt2(ll x) {  
 ll r = (ll)sqrt(x - 0.1);  
 while(r \* r <= x) ++r;  
 return int(r - 1);  
}  
int sqrt3(ll x) {  
 ll r = (ll)cbrt(x - 0.1);  
 while(r \* r \* r <= x) ++r;  
 return int(r - 1);  
}  
ll getphi(ll x, int s)  
{  
 if(s == 0) return x;  
 if(s <= M) return phi[x % sz[s]][s] + (x / sz[s]) \* phi[sz[s]][s];  
 if(x <= prime[s]\*prime[s]) return pi[x] - s + 1;  
 if(x <= prime[s]\*prime[s]\*prime[s] && x < maxn) {  
 int s2x = pi[sqrt2(x)];  
 ll ans = pi[x] - (s2x + s - 2) \* (s2x - s + 1) / 2;  
 for(int i = s + 1; i <= s2x; ++i) ans += pi[x / prime[i]];  
 return ans;  
 }  
 return getphi(x, s - 1) - getphi(x / prime[s], s - 1);  
}  
ll getpi(ll x) {  
 if(x < maxn) return pi[x];  
 ll ans = getphi(x, pi[sqrt3(x)]) + pi[sqrt3(x)] - 1;  
 for(int i = pi[sqrt3(x)] + 1, ed = pi[sqrt2(x)]; i <= ed; ++i) ans -= getpi(x / prime[i]) - i + 1;  
 return ans;  
}  
ll lehmer\_pi(ll x) {  
 if(x < maxn) return pi[x];  
 int a = (int)lehmer\_pi(sqrt2(sqrt2(x)));  
 int b = (int)lehmer\_pi(sqrt2(x));  
 int c = (int)lehmer\_pi(sqrt3(x));  
 ll sum = getphi(x, a) +(ll)(b + a - 2) \* (b - a + 1) / 2;  
 for (int i = a + 1; i <= b; i++) {  
 ll w = x / prime[i];  
 sum -= lehmer\_pi(w);  
 if (i > c) continue;  
 ll lim = lehmer\_pi(sqrt2(w));  
 for (int j = i; j <= lim; j++) sum -= lehmer\_pi(w / prime[j]) - (j - 1);  
 }  
 return sum;  
}  
int main() {  
 init();  
 ll n;  
 while(~scanf("%lld",&n))  
 {  
 printf("%lld\n",lehmer\_pi(n));  
 }  
 return 0;  
}

// 欧拉函数  
int euler\_phi(int n) {  
 int res = n;  
 for (int i = 2; i \* i <= n; ++i) {  
 if (n % i == 0) {  
 res = res / i \* (i - 1);  
 for (; n % i == 0; n /= i);  
 }  
 }  
 if (n != 1) res = res / n \* (n - 1);  
 return res;  
}  
int euler[maxn];  
void euler\_phi\_sieve() {  
 for (int i = 0; i < maxn; ++i) euler[i] = i;  
 for (int i = 2; i < maxn; ++i)  
 if (euler[i] == i)  
 for (int j = i; j < maxn; j += i) euler[j] = euler[j] / i \* (i - 1);  
}

* Moebius 如果 ，则

对于 函数，有如下性质：

int mu[maxn];  
void moebius() {  
 int cnt = 0; mu[1] = 1;  
 memset(vis, 0, sizeof vis);  
 for (int i = 2; i < maxn; ++i) {  
 if (!vis[i]) {  
 prime[cnt++] = i;  
 mu[i] = -1;  
 }  
 for (int j = 0; j < cnt && i \* prime[j] < maxn; ++j) {  
 vis[i \* prime[j]] = true;  
 if (i % prime[j])  
 mu[i \* prime[j]] = -mu[i];  
 else  
 mu[i \* prime[j]] = 0, break;  
 }  
 }  
}  
  
map<int, int> moebius(int n) {  
 map<int, int> res;  
 vector<int> primes;  
 for (int i = 2; i \* i <= n; ++i) {  
 if (n % i == 0) {  
 primes.push\_back(i);  
 while (n % i == 0) n /= i;  
 }  
 }  
 if (n != 1) primes.push\_back(n);  
  
 int m = primes.size();  
 for (int i = 0; i < (1 << m); ++i) {  
 int mu = 1, d = 1;  
 for (int j = 0; j < m; ++j) {  
 if (i >> j & 1) {  
 mu \*= -1;  
 d \*= primes[j];  
 }  
 }  
 res[d] = mu;  
 }  
 return res;  
}

// Guass\_jordan  
const double eps = 1e-8;  
typedef vector<double> vec;  
typedef vector<vec> mat;  
  
vec gauss\_joedan(const mat &A, const vec& b) {  
 int n = A.size();  
 mat B(n, vec(n + 1));  
 for (int i = 0; i < n; ++i)  
 for (int j = 0; j < n; ++j) B[i][j] = A[i][j];  
 for (int i = 0; i < n; ++i) B[i][n] = b[i];  
  
 for (int i = 0; i < n; ++i) {  
 int pivot = i;  
 for (int j = i; j < n; ++j)  
 if (abs(B[j][i]) > abs(B[pivot][i])) pivot = j;  
 if (i != pivot) swap(B[i], B[pivot]);  
  
 if (abs(B[i][i]) < eps) return vec();  
  
 for (int j = i + 1; j <= n; ++j) B[i][j] /= B[i][i];  
 for (int j = 0; j < n; ++j) if (i != j)  
 for (int k = i + 1; k <= n; ++k) B[j][k] -= B[j][i] \* B[i][k];  
 }  
  
 vec x(n);  
 for (int i = 0; i < n; ++i) x[i] = B[i][n];  
 return x;  
}  
  
vec gauss\_joedan\_xor(const mat& A, const vec& b) {  
 int n = A.size();  
 mat B(n, vec(n + 1));  
 for (int i = 0; i < n; ++i)  
 for (int j = 0; j < n; ++j) B[i][j] = A[i][j];  
 for (int i = 0; i < n; ++i) B[i][n] = b[i];  
  
 for (int i = 0; i < n; ++i) {  
 int pivot = i;  
 for (int j = i; j < n; ++j)  
 if (B[j][i]) {  
 pivot = j;  
 break;  
 }  
 if (pivot != i) swap(B[i], B[pivot]);  
  
 for (int j = 0; j < n; ++j) if (i != j && B[j][i])  
 for (int k = i + 1; k <= n; ++k) B[j][k] ^= B[i][k];  
 }  
 }  
  
 vec x(n);  
 for (int i = 0; i < n; ++i) x[i] = B[i][n];  
 return x;  
}

Simpson 公式——二次函数近似原函数积分:

// LA3485 Bridge: 自适应辛普森版  
// Rujia Liu  
#include<cstdio>  
#include<cmath>  
  
// 这里为了方便，把a声明成全局的。  
// 这不是一个好的编程习惯，但在本题中却可以提高代码的可读性  
double a;  
  
// simpson公式用到的函数  
double F(double x) {  
 return sqrt(1 + 4\*a\*a\*x\*x);  
}  
  
// 三点simpson法。这里要求F是一个全局函数  
double simpson(double a, double b) {  
 double c = a + (b-a)/2;  
 return (F(a)+4\*F(c)+F(b))\*(b-a)/6;  
}  
  
// 自适应Simpson公式（递归过程）。已知整个区间[a,b]上的三点simpson值A  
double asr(double a, double b, double eps, double A) {  
 double c = a + (b-a)/2;  
 double L = simpson(a, c), R = simpson(c, b);  
 if(fabs(L+R-A) <= 15\*eps) return L+R+(L+R-A)/15.0;  
 return asr(a, c, eps/2, L) + asr(c, b, eps/2, R);  
}  
  
// 自适应Simpson公式（主过程）  
double asr(double a, double b, double eps) {  
 return asr(a, b, eps, simpson(a, b));  
}  
  
// 用自适应Simpson公式计算宽度为w，高度为h的抛物线长  
double parabola\_arc\_length(double w, double h) {  
 a = 4.0\*h/(w\*w); // 修改全局变量a，从而改变全局函数F的行为  
 return asr(0, w/2, 1e-5)\*2;  
}  
  
int main() {  
 int T;  
 scanf("%d", &T);  
 for(int kase = 1; kase <= T; kase++) {  
 int D, H, B, L;  
 scanf("%d%d%d%d", &D, &H, &B, &L);  
 int n = (B+D-1)/D; // 间隔数  
 double D1 = (double)B / n;  
 double L1 = (double)L / n;  
 double x = 0, y = H;  
 while(y-x > 1e-5) { // 二分法求解高度  
 double m = x + (y-x)/2;  
 if(parabola\_arc\_length(D1, m) < L1) x = m; else y = m;  
 }  
 if(kase > 1) printf("\n");  
 printf("Case %d:\n%.2lf\n", kase, H-x);  
 }  
 return 0;  
}

// Multiplying Polynomials  
// UVa12298 Super Poker II  
// Rujia Liu  
#include <complex>  
#include <cmath>  
#include <vector>  
using namespace std;  
  
const long double PI = acos(0.0) \* 2.0;  
  
typedef complex<double> CD;  
  
// Cooley-Tukey的FFT算法，迭代实现。inverse = false时计算逆FFT  
inline void FFT(vector<CD> &a, bool inverse) {  
 int n = a.size();  
 // 原地快速bit reversal  
 for(int i = 0, j = 0; i < n; i++) {  
 if(j > i) swap(a[i], a[j]);  
 int k = n;  
 while(j & (k >>= 1)) j &= ~k;  
 j |= k;  
 }  
  
 double pi = inverse ? -PI : PI;  
 for(int step = 1; step < n; step <<= 1) {  
 // 把每相邻两个“step点DFT”通过一系列蝴蝶操作合并为一个“2\*step点DFT”  
 double alpha = pi / step;  
 // 为求高效，我们并不是依次执行各个完整的DFT合并，而是枚举下标k  
 // 对于一个下标k，执行所有DFT合并中该下标对应的蝴蝶操作，即通过E[k]和O[k]计算X[k]  
 // 蝴蝶操作参考：http://en.wikipedia.org/wiki/Butterfly\_diagram  
 for(int k = 0; k < step; k++) {  
 // 计算omega^k. 这个方法效率低，但如果用每次乘omega的方法递推会有精度问题。  
 // 有更快更精确的递推方法，为了清晰起见这里略去  
 CD omegak = exp(CD(0, alpha\*k));  
 for(int Ek = k; Ek < n; Ek += step << 1) { // Ek是某次DFT合并中E[k]在原始序列中的下标  
 int Ok = Ek + step; // Ok是该DFT合并中O[k]在原始序列中的下标  
 CD t = omegak \* a[Ok]; // 蝴蝶操作：x1 \* omega^k  
 a[Ok] = a[Ek] - t; // 蝴蝶操作：y1 = x0 - t  
 a[Ek] += t; // 蝴蝶操作：y0 = x0 + t  
 }  
 }  
 }  
  
 if(inverse)  
 for(int i = 0; i < n; i++) a[i] /= n;  
}  
  
// 用FFT实现的快速多项式乘法  
inline vector<double> operator \* (const vector<double>& v1, const vector<double>& v2) {  
 int s1 = v1.size(), s2 = v2.size(), S = 2;  
 while(S < s1 + s2) S <<= 1;  
 vector<CD> a(S,0), b(S,0); // 把FFT的输入长度补成2的幂，不小于v1和v2的长度之和  
 for(int i = 0; i < s1; i++) a[i] = v1[i];  
 FFT(a, false);  
 for(int i = 0; i < s2; i++) b[i] = v2[i];  
 FFT(b, false);  
 for(int i = 0; i < S; i++) a[i] \*= b[i];  
 FFT(a, true);  
 vector<double> res(s1 + s2 - 1);  
 for(int i = 0; i < s1 + s2 - 1; i++) res[i] = a[i].real(); // 虚部均为0  
 return res;  
}  
  
/////////// 题目相关  
#include<cstdio>  
#include<cstring>  
const int maxn = 50000 + 10;  
  
int composite[maxn];  
void sieve(int n) {  
 int m = (int)sqrt(n+0.5);  
 memset(composite, 0, sizeof(composite));  
 for(int i = 2; i <= m; i++) if(!composite[i])  
 for(int j = i\*i; j <= n; j+=i) composite[j] = 1;  
}  
  
const char\* suites = "SHCD";  
int idx(char suit) {  
 return strchr(suites, suit) - suites;  
}  
  
int lost[4][maxn];  
int main(int argc, char \*argv[]) {  
 sieve(50000);  
 int a, b, c;  
 while(scanf("%d%d%d", &a, &b, &c) == 3 && a) {  
 memset(lost, 0, sizeof(lost));  
 for(int i = 0; i < c; i++) {  
 int d; char s;  
 scanf("%d%c", &d, &s);  
 lost[idx(s)][d] = 1;  
 }  
 vector<double> ans(1,1), poly;  
 for(int s = 0; s < 4; s++) {  
 poly.clear();  
 poly.resize(b+1, 0);  
 for(int i = 4; i <= b; i++)  
 if(composite[i] && !lost[s][i]) poly[i] = 1.0;  
 ans = ans \* poly;  
 ans.resize(b+1);  
 }  
 for(int i = a; i <= b; i++)  
 printf("%.0lf\n", fabs(ans[i]));  
 printf("\n");  
 }  
 return 0;  
}

// LA4746 Decrypt Messages  
// Rujia Liu  
#include <cstdio>  
#include <cstdlib>  
#include <cstring>  
#include <cmath>  
#include <vector>  
#include <map>  
#include <algorithm>  
#include <iostream>  
using namespace std;  
  
typedef long long LL;  
  
//// 日期时间部分  
  
const int SECONDS\_PER\_DAY = 24 \* 60 \* 60;  
  
const int num\_days[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};  
  
bool is\_leap(int year) {  
 if (year % 400 == 0) return true;  
 if (year % 4 == 0) return year % 100 != 0;  
 return false;  
}  
  
int leap\_second(int year, int month) {  
 return ((year % 10 == 5 || year % 10 == 8) && month == 12) ? 1 : 0;  
}  
  
void print(int year, int month, int day, int hh, int mm, int ss) {  
 printf("%d.%02d.%02d %02d:%02d:%02d\n", year, month, day, hh, mm, ss);  
}  
  
void print\_time(LL t) {  
 int year = 2000;  
 while(1) {  
 int days = is\_leap(year) ? 366 : 365;  
 LL sec = (LL)days \* SECONDS\_PER\_DAY + leap\_second(year, 12);  
 if(t < sec) break;  
 t -= sec;  
 year++;  
 }  
  
 int month = 1;  
 while(1) {  
 int days = num\_days[month-1];  
 if(is\_leap(year) && month == 2) days++;  
 LL sec = (LL)days \* SECONDS\_PER\_DAY + leap\_second(year, month);  
 if(t < sec) break;  
 t -= sec;  
 month++;  
 }  
  
 if(leap\_second(year, month) && t == 31 \* SECONDS\_PER\_DAY)  
 print(year, 12, 31, 23, 59, 60);  
 else {  
 int day = t / SECONDS\_PER\_DAY + 1;  
 t %= SECONDS\_PER\_DAY;  
 int hh = t / (60\*60);  
 t %= 60\*60;  
 int mm = t / 60;  
 t %= 60;  
 int ss = t;  
 print(year, month, day, hh, mm, ss);  
 }  
}  
  
//// 数论部分  
  
LL gcd(LL a, LL b) {  
 return b ? gcd(b, a%b) : a;  
}  
  
// 求d = gcd(a, b)，以及满足ax+by=d的(x,y)（注意，x和y可能为负数）  
// 扩展euclid算法。  
void gcd(LL a, LL b, LL& d, LL& x, LL& y) {  
 if(!b){ d = a; x = 1; y = 0; }  
 else{ gcd(b, a%b, d, y, x); y -= x\*(a/b); }  
}  
  
// 注意，返回值可能是负的  
int pow\_mod(LL a, LL p, int MOD) {  
 if(p == 0) return 1;  
 LL ans = pow\_mod(a, p/2, MOD);  
 ans = ans \* ans % MOD;  
 if(p%2) ans = ans \* a % MOD;  
 return ans;  
}  
  
// 注意，返回值可能是负的  
int mul\_mod(LL a, LL b, int MOD) {  
 return a \* b % MOD;  
}  
  
// 求ax = 1 (mod MOD) 的解，其中a和MOD互素。  
// 注意，由于MOD不一定为素数，因此不能直接用pow\_mod(a, MOD-2, MOD)求解  
// 解法：先求ax + MODy = 1的解(x,y)，则x为所求  
int inv(LL a, int MOD) {  
 LL d, x, y;  
 gcd(a, MOD, d, x, y);  
 return (x + MOD) % MOD; // 这里的x可能是负数，因此要调整  
}  
  
// 解模方程（即离散对数）a^x = b。要求MOD为素数  
// 解法：Shank的大步小步算法  
int log\_mod(int a, int b, int MOD) {  
 int m, v, e = 1, i;  
 m = (int)sqrt(MOD);  
 v = inv(pow\_mod(a, m, MOD), MOD);  
 map<int,int> x;  
 x[1] = 0;  
 for(i = 1; i < m; i++){ e = mul\_mod(e, a, MOD); if (!x.count(e)) x[e] = i; }  
 for(i = 0; i < m; i++){  
 if(x.count(b)) return i\*m + x[b];  
 b = mul\_mod(b, v, MOD);  
 }  
 return -1;  
}  
  
// 返回MOD（不一定是素数）的某一个原根，phi为MOD的欧拉函数值（若MOD为素数则phi=MOD-1）  
// 解法：考虑phi(MOD)的所有素因子p，如果所有m^(phi/p) mod MOD都不等于1，则m是MOD的原根  
int get\_primitive\_root(int MOD, int phi) {  
 // 计算phi的所有素因子  
 vector<int> factors;  
 int n = phi;  
 for(int i = 2; i\*i <= n; i++) {  
 if(n % i != 0) continue;  
 factors.push\_back(i);  
 while(n % i == 0) n /= i;  
 }  
 if(n > 1) factors.push\_back(n);  
  
 while(1) {  
 int m = rand() % (MOD-2) + 2; // m = 2~MOD-1  
 bool ok = true;  
 for(int i = 0; i < factors.size(); i++)  
 if(pow\_mod(m, phi/factors[i], MOD) == 1) { ok = false; break; }  
 if(ok) return m;  
 }  
}  
  
// 解线性模方程 ax = b (mod n)，返回所有解（模n剩余系）  
// 解法：令d = gcd(a, n)，两边同时除以d后得a'x = b' (mod n')，由于此时gcd(a',n')=1，两边同时左乘a'在模n'中的逆即可，最后把模n'剩余系中的解转化为模n剩余系  
vector<LL> solve\_linear\_modular\_equation(int a, int b, int n) {  
 vector<LL> ans;  
 int d = gcd(a, n);  
 if(b % d != 0) return ans;  
 a /= d; b /= d;  
 int n2 = n / d;  
 int p = mul\_mod(inv(a, n2), b, n2);  
 for(int i = 0; i < d; i++)  
 ans.push\_back(((LL)i \* n2 + p) % n);  
 return ans;  
}  
  
// 解高次模方程 x^q = a (mod p)，返回所有解（模n剩余系）  
// 解法：设m为p的一个原根，且x = m^y, a = m^z，则m^qy = m^z(mod p)，因此qy = z(mod p-1)，解线性模方程即可  
vector<LL> mod\_root(int a, int q, int p) {  
 vector<LL> ans;  
 if(a == 0) {  
 ans.push\_back(0);  
 return ans;  
 }  
 int m = get\_primitive\_root(p, p-1); // p是素数，因此phi(p)=p-1  
 int z = log\_mod(m, a, p);  
 ans = solve\_linear\_modular\_equation(q, z, p-1);  
 for(int i = 0; i < ans.size(); i++)  
 ans[i] = pow\_mod(m, ans[i], p);  
 sort(ans.begin(), ans.end());  
 return ans;  
}  
  
int main() {  
 int T, P, Q, A;  
 cin >> T;  
 for(int kase = 1; kase <= T; kase++) {  
 cin >> P >> Q >> A;  
 vector<LL> ans = mod\_root(A, Q, P);  
 cout << "Case #" << kase << ":" << endl;  
 if (ans.empty()) {  
 cout << "Transmission error" << endl;  
 } else {  
 for(int i = 0; i < ans.size(); i++) print\_time(ans[i]);  
 }  
 }  
 return 0;  
}

## String

Hash，KMP，Extend KMP，trie树，Manacher 算法，AC自动机，后缀数组，后缀树，后缀自动机，回文自动机

// 最小最大表示法：  
int getMinString(const string &s) {  
 int len = (int)s.length();  
 int i = 0, j = 1, k = 0;  
 while(i < len && j < len && k < len) {  
 int t = s[(i + k) % len] - s[(j + k) % len];  
 if(t == 0) k++;  
 else {  
 if(t > 0) i += k + 1;//getMaxString: t < 0  
 else j += k + 1;  
 if(i == j) j++;  
 k = 0;  
 }  
 }  
 return min(i, j);  
}

// KMP  
int nxt[maxn];  
void getNext(const string &str) {  
 int len = str.length();  
 int j = 0, k;  
 k = nxt[0] = -1;  
 while (j < len) {  
 if (k == -1 || str[j] == str[k])  
 nxt[++j] = ++k;  
 else k = nxt[k];  
 }  
}  
int kmp(const string &tar, const string &pat) {  
 getNext(pat);  
 int num, j, k;  
 int lenT = tar.length(), lenP = pat.length();  
 num = j = k = 0;  
 while (j < lenT) {  
 if(k == -1 || tar[j] == pat[k])  
 j++, k++;  
 else k = nxt[k];  
 if(k == lenP) {  
 // res = max(res, j - lenP);  
 k = nxt[k];  
 ++num;  
 }  
 }  
 return num;//lenP - res - 1;  
}

主串 s[0...n] 模式串 t[0..m] bitset D 中 D[j] = 1 表示模式串前缀 是主串 的后缀。 D = (D << 1 | 1) & B[s[i + 1]]

bitset<maxm> D, S[256];  
void shiftAnd(int n, int m) {  
 D.reset();  
 for (int i = 0; i < n; i++) {  
 D <<= 1; D.set(0);  
 D &= B[s[i]];  
 if (D[m - 1]) {  
 char tmp = s[i + 1];  
 s[i + 1] = '\0';  
 puts(s + (i - n + 1));  
 s[i + 1] = tmp;  
 }  
 }  
}

// Suffix Array & LCP Array  
int n, k;  
int lcp[maxn], sa[maxn];  
int rnk[maxn], tmp[maxn];  
  
bool compare\_sa(int i, int j) {  
 if (rnk[i] != rnk[j]) return rnk[i] < rnk[j];  
 else {  
 int ri = i + k <= n? rnk[i + k] : -1;  
 int rj = j + k <= n? rnk[j + k] : -1;  
 return ri < rj;  
 }  
}  
void construct\_sa(string &S, int \*sa) {  
 n = S.length();  
 for (int i = 0; i <= n; i++) {  
 sa[i] = i;  
 rnk[i] = i < n? S[i] : -1;  
 }  
 for (k = 1; k <= n; k \*= 2) {  
 sort(sa, sa + n + 1, compare\_sa);  
 tmp[sa[0]] = 0;  
 for (int i = 1; i <= n; i++)  
 tmp[sa[i]] = tmp[sa[i - 1]] + (compare\_sa(sa[i - 1], sa[i]) ? 1 : 0);  
 memcpy(rnk, tmp, sizeof(int) \* (n + 1));  
 }  
}  
void construct\_lcp(string &S, int \*sa, int \*lcp) {  
 n = S.length();  
 for (int i = 0; i <= n; i++) rnk[sa[i]] = i;  
 int h = 0;  
 lcp[0] = 0;  
 for (int i = 0; i < n; i++) {  
 int j = sa[rnk[i] - 1];  
 if (h > 0) h--;  
 for (; j + h < n && i + h < n; h++)  
 if (S[j + h] != S[i + h]) break;  
 lcp[rnk[i] - 1] = h;  
 }  
}

// AC 自动机  
int ans[maxn], d[maxn];  
  
struct Trie {  
 int nxt[maxn][26], fail[maxn], end[maxn];  
 int root, L;  
 int newnode() {  
 for(int i = 0; i < 26; i++)  
 nxt[L][i] = -1;  
 end[L++] = 0;  
 return L-1;  
 }  
 void init() {  
 L = 0;  
 root = newnode();  
 }  
 void insert(char buf[]) {  
 int len = strlen(buf);  
 int now = root;  
 for(int i = 0; i < len; i++) {  
 if(nxt[now][buf[i]-'a'] == -1)  
 nxt[now][buf[i]-'a'] = newnode();  
 now = nxt[now][buf[i]-'a'];  
 }  
 end[now] = 1;  
 d[now] = len;  
 }  
 void build() {  
 queue<int> Q;  
 fail[root] = root;  
 for(int i = 0; i < 26; i++)  
 if(nxt[root][i] == -1)  
 nxt[root][i] = root;  
 else {  
 fail[nxt[root][i]] = root;  
 Q.push(nxt[root][i]);  
 }  
 while( !Q.empty() ) {  
 int now = Q.front(); Q.pop();  
 for(int i = 0; i < 26; i++)  
 if(nxt[now][i] == -1)  
 nxt[now][i] = nxt[fail[now]][i];  
 else {  
 fail[nxt[now][i]] = nxt[fail[now]][i];  
 Q.push(nxt[now][i]);  
 }  
 }  
 }  
 void solve(char buf[]) {  
 int cur = root;  
 int len = strlen(buf);  
 int index;  
 for(int i = 0; i < len; ++i) {  
 if(buf[i] >= 'A' && buf[i] <= 'Z')  
 index = buf[i] - 'A';  
 else if(buf[i] >= 'a' && buf[i] <= 'z')  
 index = buf[i] - 'a';  
 else continue;  
 cur = nxt[cur][index];  
 int x = cur;  
 while(x != root) {  
 if(end[x]) {  
 ans[i + 1] -= 1;  
 ans[i - d[x] + 1] += 1;  
 break;  
 }  
 x = fail[x];  
 }  
 }  
 }  
};  
  
Trie ac;

## Others

### Divide-and-Conquer Tree

//uva 12161  
struct edge {  
 int to, damage, length, next;  
};  
int G[maxn], En, N, M, T;  
edge E[maxn \* 2];  
  
void add\_edge(int from, int to, int damage, int length) {  
 edge e = {to, damage, length, G[from]};  
 E[En] = e;  
 G[from] = En++;  
}  
  
int ans, subtree\_size[maxn];  
bool flag[maxn];  
  
int s, t;  
Pii ds[maxn];  
  
int compute\_subtree\_size(int v, int p) {  
 int c = 1;  
 for (int j = G[v]; ~j; j = E[j].next) {  
 int w = E[j].to;  
 if (w == p || flag[w]) continue;  
 c += compute\_subtree\_size(w, v);  
 }  
 return subtree\_size[v] = c;  
}  
  
Pii search\_centroid(int v, int p, int t) {  
 Pii res = Pii(INT\_MAX, -1);  
 int s = 1, m = 0;  
 for (int j = G[v]; ~j; j = E[j].next) {  
 int w = E[j].to;  
 if (w == p || flag[w]) continue;  
 res = min(res, search\_centroid(w, v, t));  
 m = max(subtree\_size[w], m);  
 s += subtree\_size[w];  
 }  
 m = max(m, t - s);  
 res = min(res, Pii(m, v));  
 return res;  
}  
  
void enumrate\_path(int v, int p, int damage, int length) {  
 ds[t++] = Pii(damage, length);  
 for (int j = G[v]; ~j; j = E[j].next) {  
 int w = E[j].to;  
 if (w == p || flag[w]) continue;  
 if (damage + E[j].damage <= M) {  
 enumrate\_path(w, v, damage + E[j].damage, length + E[j].length);  
 }  
 }  
}  
  
void remove\_useless(int s, int &t) {  
 if (s == t) return;  
 int tt;  
 for (int i = tt = s + 1; i < t; i++) {  
 if (ds[i].first == ds[tt - 1].first) continue;  
 if (ds[i].second <= ds[tt - 1].second) continue;  
 ds[tt++] = ds[i];  
 }  
 t = tt;  
}  
  
void solve\_sub\_problem(int v) {  
 compute\_subtree\_size(v, -1);  
 int c = search\_centroid(v, -1, subtree\_size[v]).second;  
 flag[c] = true;  
 for (int j = G[c]; ~j; j = E[j].next) {  
 if (flag[E[j].to]) continue;  
 solve\_sub\_problem(E[j].to);  
 }  
  
 s = t = 0;  
 for (int j = G[c]; ~j; j = E[j].next) {  
 int w = E[j].to;  
 if (flag[w]) continue;  
 if (E[j].damage <= M)  
 enumrate\_path(w, v, E[j].damage, E[j].length);  
 if (s > 0) {  
 sort(ds + s, ds + t);  
 remove\_useless(s, t);  
 for (int l = 0, r = t - 1; l < s && r >= s; l++) {  
 while (r >= s && ds[l].first + ds[r].first > M) r--;  
 if (r >= s)  
 ans = max(ans, ds[l].second + ds[r].second);  
 }  
 }  
 sort(ds, ds + t);  
 remove\_useless(0, t);  
 s = t;  
 }  
  
 flag[c] = false;  
}

### Simplex Algorithm

// UVa10498 Happiness!  
// Rujia Liu  
#include<cstdio>  
#include<cstring>  
#include<algorithm>  
#include<cassert>  
using namespace std;  
  
// 改进单纯性法的实现  
// 参考：http://en.wikipedia.org/wiki/Simplex\_algorithm  
// 输入矩阵a描述线性规划的标准形式。a为m+1行n+1列，其中行0~m-1为不等式，行m为目标函数（最大化）。列0~n-1为变量0~n-1的系数，列n为常数项  
// 第i个约束为a[i][0]\*x[0] + a[i][1]\*x[1] + ... <= a[i][n]  
// 目标为max(a[m][0]\*x[0] + a[m][1]\*x[1] + ... + a[m][n-1]\*x[n-1] - a[m][n])  
// 注意：变量均有非负约束x[i] >= 0  
const int maxm = 500; // 约束数目上限  
const int maxn = 500; // 变量数目上限  
const double INF = 1e100;  
const double eps = 1e-10;  
  
struct Simplex {  
 int n; // 变量个数  
 int m; // 约束个数  
 double a[maxm][maxn]; // 输入矩阵  
 int B[maxm], N[maxn]; // 算法辅助变量  
  
 void pivot(int r, int c) {  
 swap(N[c], B[r]);  
 a[r][c] = 1 / a[r][c];  
 for(int j = 0; j <= n; j++) if(j != c) a[r][j] \*= a[r][c];  
 for(int i = 0; i <= m; i++) if(i != r) {  
 for(int j = 0; j <= n; j++) if(j != c) a[i][j] -= a[i][c] \* a[r][j];  
 a[i][c] = -a[i][c] \* a[r][c];  
 }  
 }  
  
 bool feasible() {  
 for(;;) {  
 int r, c;  
 double p = INF;  
 for(int i = 0; i < m; i++) if(a[i][n] < p) p = a[r = i][n];  
 if(p > -eps) return true;  
 p = 0;  
 for(int i = 0; i < n; i++) if(a[r][i] < p) p = a[r][c = i];  
 if(p > -eps) return false;  
 p = a[r][n] / a[r][c];  
 for(int i = r+1; i < m; i++) if(a[i][c] > eps) {  
 double v = a[i][n] / a[i][c];  
 if(v < p) { r = i; p = v; }  
 }  
 pivot(r, c);  
 }  
 }  
  
 // 解有界返回1，无解返回0，无界返回-1。b[i]为x[i]的值，ret为目标函数的值  
 int simplex(int n, int m, double x[maxn], double& ret) {  
 this->n = n;  
 this->m = m;  
 for(int i = 0; i < n; i++) N[i] = i;  
 for(int i = 0; i < m; i++) B[i] = n+i;  
 if(!feasible()) return 0;  
 for(;;) {  
 int r, c;  
 double p = 0;  
 for(int i = 0; i < n; i++) if(a[m][i] > p) p = a[m][c = i];  
 if(p < eps) {  
 for(int i = 0; i < n; i++) if(N[i] < n) x[N[i]] = 0;  
 for(int i = 0; i < m; i++) if(B[i] < n) x[B[i]] = a[i][n];  
 ret = -a[m][n];  
 return 1;  
 }  
 p = INF;  
 for(int i = 0; i < m; i++) if(a[i][c] > eps) {  
 double v = a[i][n] / a[i][c];  
 if(v < p) { r = i; p = v; }  
 }  
 if(p == INF) return -1;  
 pivot(r, c);  
 }  
 }  
};  
  
//////////////// 题目相关  
#include<cmath>  
Simplex solver;  
  
int main() {  
 int n, m;  
 while(scanf("%d%d", &n, &m) == 2) {  
 for(int i = 0; i < n; i++) scanf("%lf", &solver.a[m][i]); // 目标函数  
 solver.a[m][n] = 0; // 目标函数常数项  
 for(int i = 0; i < m; i++)  
 for(int j = 0; j < n+1; j++)  
 scanf("%lf", &solver.a[i][j]);  
 double ans, x[maxn];  
 assert(solver.simplex(n, m, x, ans) == 1);  
 ans \*= m;  
 printf("Nasa can spend %d taka.\n", (int)floor(ans + 1 - eps));  
 }  
 return 0;  
}

### DLX

// LA2659 Sudoku  
// Rujia Liu  
#include<cstdio>  
#include<cstring>  
#include<vector>  
  
using namespace std;  
  
const int maxr = 5000;  
const int maxn = 2000;  
const int maxnode = 20000;  
  
// 行编号从1开始，列编号为1~n，结点0是表头结点; 结点1~n是各列顶部的虚拟结点  
struct DLX {  
 int n, sz; // 列数，结点总数  
 int S[maxn]; // 各列结点数  
  
 int row[maxnode], col[maxnode]; // 各结点行列编号  
 int L[maxnode], R[maxnode], U[maxnode], D[maxnode]; // 十字链表  
  
 int ansd, ans[maxr]; // 解  
  
 void init(int n) { // n是列数  
 this->n = n;  
  
 // 虚拟结点  
 for(int i = 0 ; i <= n; i++) {  
 U[i] = i; D[i] = i; L[i] = i-1, R[i] = i+1;  
 }  
 R[n] = 0; L[0] = n;  
  
 sz = n + 1;  
 memset(S, 0, sizeof(S));  
 }  
  
 void addRow(int r, vector<int> columns) {  
 int first = sz;  
 for(int i = 0; i < columns.size(); i++) {  
 int c = columns[i];  
 L[sz] = sz - 1; R[sz] = sz + 1; D[sz] = c; U[sz] = U[c];  
 D[U[c]] = sz; U[c] = sz;  
 row[sz] = r; col[sz] = c;  
 S[c]++; sz++;  
 }  
 R[sz - 1] = first; L[first] = sz - 1;  
 }  
  
 // 顺着链表A，遍历除s外的其他元素  
 #define FOR(i,A,s) for(int i = A[s]; i != s; i = A[i])  
  
 void remove(int c) {  
 L[R[c]] = L[c];  
 R[L[c]] = R[c];  
 FOR(i,D,c)  
 FOR(j,R,i) { U[D[j]] = U[j]; D[U[j]] = D[j]; --S[col[j]]; }  
 }  
  
 void restore(int c) {  
 FOR(i,U,c)  
 FOR(j,L,i) { ++S[col[j]]; U[D[j]] = j; D[U[j]] = j; }  
 L[R[c]] = c;  
 R[L[c]] = c;  
 }  
  
 // d为递归深度  
 bool dfs(int d) {  
 if (R[0] == 0) { // 找到解  
 ansd = d; // 记录解的长度  
 return true;  
 }  
  
 // 找S最小的列c  
 int c = R[0]; // 第一个未删除的列  
 FOR(i,R,0) if(S[i] < S[c]) c = i;  
  
 remove(c); // 删除第c列  
 FOR(i,D,c) { // 用结点i所在行覆盖第c列  
 ans[d] = row[i];  
 FOR(j,R,i) remove(col[j]); // 删除结点i所在行能覆盖的所有其他列  
 if(dfs(d+1)) return true;  
 FOR(j,L,i) restore(col[j]); // 恢复结点i所在行能覆盖的所有其他列  
 }  
 restore(c); // 恢复第c列  
  
 return false;  
 }  
  
 bool solve(vector<int>& v) {  
 v.clear();  
 if(!dfs(0)) return false;  
 for(int i = 0; i < ansd; i++) v.push\_back(ans[i]);  
 return true;  
 }  
  
};  
  
////////////// 题目相关  
#include<cassert>  
  
DLX solver;  
  
const int SLOT = 0;  
const int ROW = 1;  
const int COL = 2;  
const int SUB = 3;  
  
// 行/列的统一编解码函数。从1开始编号  
int encode(int a, int b, int c) {  
 return a\*256+b\*16+c+1;  
}  
  
void decode(int code, int& a, int& b, int& c) {  
 code--;  
 c = code%16; code /= 16;  
 b = code%16; code /= 16;  
 a = code;  
}  
  
char puzzle[16][20];  
  
bool read() {  
 for(int i = 0; i < 16; i++)  
 if(scanf("%s", puzzle[i]) != 1) return false;  
 return true;  
}  
  
int main() {  
 int kase = 0;  
 while(read()) {  
 if(++kase != 1) printf("\n");  
 solver.init(1024);  
 for(int r = 0; r < 16; r++)  
 for(int c = 0; c < 16; c++)  
 for(int v = 0; v < 16; v++)  
 if(puzzle[r][c] == '-' || puzzle[r][c] == 'A'+v) {  
 vector<int> columns;  
 columns.push\_back(encode(SLOT, r, c));  
 columns.push\_back(encode(ROW, r, v));  
 columns.push\_back(encode(COL, c, v));  
 columns.push\_back(encode(SUB, (r/4)\*4+c/4, v));  
 solver.addRow(encode(r, c, v), columns);  
 }  
  
 vector<int> ans;  
 assert(solver.solve(ans));  
  
 for(int i = 0; i < ans.size(); i++) {  
 int r, c, v;  
 decode(ans[i], r, c, v);  
 puzzle[r][c] = 'A'+v;  
 }  
 for(int i = 0; i < 16; i++)  
 printf("%s\n", puzzle[i]);  
 }  
 return 0;  
}

### Pruning

// LA4451 House of Cards  
// Rujia Liu  
#include<cstdio>  
#include<cstring>  
#include<algorithm>  
#include<vector>  
using namespace std;  
  
const int UP = 0;  
const int FLOOR = 1;  
const int DOWN = 2;  
  
const int maxn = 20;  
int n, deck[maxn\*2];  
  
struct State {  
 int card[8], type[8]; // 两张相同的FLOOR牌代表一张真实的FLOOR牌  
 int hold[2];  
 int pos;  
 int score; // MAX游戏者(即Axel)的得分  
  
 State child() const {  
 State s;  
 memcpy(&s, this, sizeof(s));  
 s.pos = pos + 1;  
 return s;  
 }  
  
 State() {  
 for(int i = 0; i < 8; i++) {  
 card[i] = deck[i];  
 type[i] = i % 2 == 0 ? UP : DOWN;  
 }  
 hold[0] = hold[1] = score = 0;  
 pos = 8;  
 }  
  
 bool isFinal() {  
 if(pos == 2\*n) {  
 score += hold[0] + hold[1];  
 hold[0] = hold[1] = 0;  
 return true;  
 }  
 return false;  
 }  
  
 int getScore(int c1, int c2, int c3) const {  
 int S = abs(c1) + abs(c2) + abs(c3);  
 int cnt = 0;  
 if(c1 > 0) cnt++; if(c2 > 0) cnt++; if(c3 > 0) cnt++;  
 return cnt >= 2 ? S : -S;  
 }  
  
 void expand(int player, vector<State>& ret) const {  
 int cur = deck[pos];  
  
 // 决策1：拿在手里  
 if(hold[player] == 0) {  
 State s = child();  
 s.hold[player] = cur;  
 ret.push\_back(s);  
 }  
  
 // 决策2：摆楼面牌  
 for(int i = 0; i < 7; i++) if(type[i] == DOWN && type[i+1] == UP) {  
 // 用当前的牌  
 State s = child();  
 s.score += getScore(card[i], card[i+1], cur);  
 s.type[i] = s.type[i+1] = FLOOR;  
 s.card[i] = s.card[i+1] = cur;  
 ret.push\_back(s);  
  
 if(hold[player] != 0) {  
 // 用手里的牌  
 State s = child();  
 s.score += getScore(card[i], card[i+1], hold[player]);  
 s.type[i] = s.type[i+1] = FLOOR;  
 s.card[i] = s.card[i+1] = hold[player];  
 s.hold[player] = cur;  
 ret.push\_back(s);  
 }  
 }  
  
 // 决策3：新的山峰  
 if(hold[player] != 0)  
 for(int i = 0; i < 7; i++) if(type[i] == FLOOR && type[i+1] == FLOOR && card[i] == card[i+1]) {  
 State s = child();  
 s.score += getScore(card[i], hold[player], cur);  
 s.type[i] = UP; s.type[i+1] = DOWN;  
 s.card[i] = cur; s.card[i+1] = hold[player]; s.hold[player] = 0;  
 ret.push\_back(s);  
  
 swap(s.card[i], s.card[i+1]);  
 ret.push\_back(s);  
 }  
 }  
};  
  
// 带alpha-beta剪枝的对抗搜索  
int alphabeta(State& s, int player, int alpha, int beta) {  
 if(s.isFinal()) return s.score; // 终态  
  
 vector<State> children;  
 s.expand(player, children); // 扩展子结点  
  
 int n = children.size();  
 for(int i = 0; i < n; i++) {  
 int v = alphabeta(children[i], player^1, alpha, beta);  
 if(!player) alpha = max(alpha, v); else beta = min(beta, v);  
 if(beta <= alpha) break; // alpha-beta剪枝  
 }  
 return !player ? alpha : beta;  
}  
  
const int INF = 1000000000;  
  
int main() {  
 int kase = 0;  
 char P[10];  
 while(scanf("%s", P) == 1 && P[0] != 'E') {  
 scanf("%d", &n);  
 for(int i = 0; i < n\*2; i++) {  
 char ch;  
 scanf("%d%c", &deck[i], &ch);  
 if(ch == 'B') deck[i] = -deck[i];  
 }  
 State initial;  
 int first\_player = deck[0] > 0 ? 0 : 1;  
 int score = alphabeta(initial, first\_player, -INF, INF);  
  
 if(P[0] == 'B') score = -score;  
 printf("Case %d: ", ++kase);  
 if(score == 0) printf("Axel and Birgit tie\n");  
 else if(score > 0) printf("%s wins %d\n", P, score);  
 else printf("%s loses %d\n", P, -score);  
 }  
 return 0;  
}

### cpp-fastIO

#### 关同步

#define IOS std::ios::sync\_with\_stdio(false); std::cin.tie(nullptr); std::cout.tie(nullptr);  
#define endl "\n"

关同步后 C IO（scanf, printf, getchar, putchar, fgets, puts, etc.） 与 C++ IO（cin, cout, etc.） 不可同时使用。

#### 读入挂

##### getchar 版

inline void read(int &x) { // 可根据情况去掉负数  
 int t = 1;  
 char ch = getchar();  
 while (ch < '0' || ch > '9') { if (ch == '-') t = -1; ch = getchar();}  
 x = 0;  
 while (ch >= '0' && ch <= '9') { x = x \* 10 + ch -'0'; ch = getchar();}  
 x \*= t;  
}  
void print(int i){  
 if(i < 10) {  
 putchar('0' + i);  
 return ;  
 }  
 print(i / 10);  
 putchar('0' + i % 10);  
}

##### freed 版

namespace fastIO {  
#define BUF\_SIZE 100000 // 本地小数据测试改为1  
 //fread -> read  
 bool IOerror = 0;  
 inline char nc() {  
 static char buf[BUF\_SIZE], \*p1 = buf + BUF\_SIZE, \*pend = buf + BUF\_SIZE;  
 if(p1 == pend) {  
 p1 = buf;  
 pend = buf + fread(buf, 1, BUF\_SIZE, stdin);  
 if(pend == p1) {  
 IOerror = 1;  
 return -1;  
 }  
 }  
 return \*p1++;  
 }  
 inline bool blank(char ch) {  
 return ch == ' ' || ch == '\n' || ch == '\r' || ch == '\t';  
 }  
 inline void read(int &x) {  
 char ch;  
 while(blank(ch = nc()));  
 if(IOerror)  
 return;  
 for(x = ch - '0'; (ch = nc()) >= '0' && ch <= '9'; x = x \* 10 + ch - '0');  
 }  
#undef BUF\_SIZE  
};  
using namespace fastIO;  
// while (read(n), !fastIO::IOerror) {}