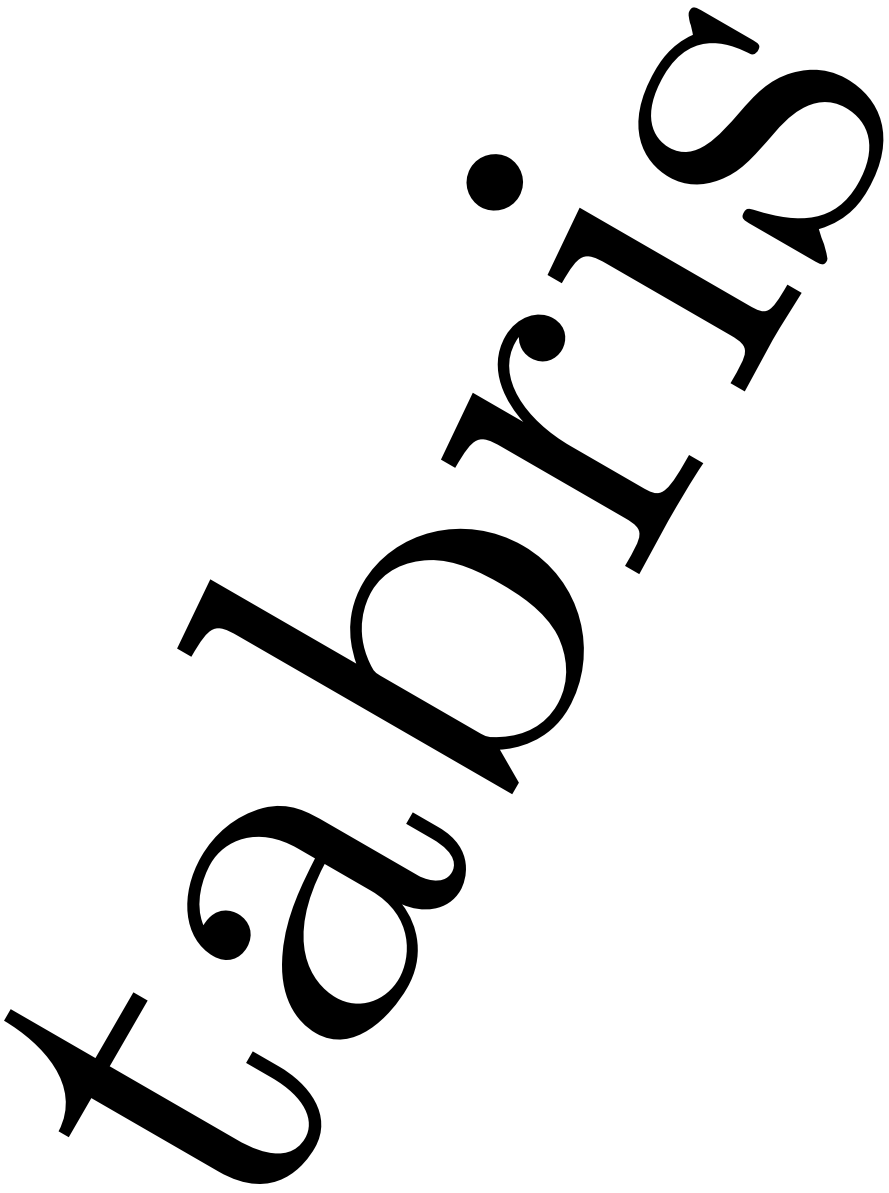
Algorithm Library

tabris

January 22, 2018



Contents

Part I
Graph Theory

1 Basement

1.1 Representation

```
//邻接表 空间复杂度 O(e+v)
vectorG[N];  //G[i][j] 表示有一条 i->G[i][j] 的有向边
void add(int u,int v){
    G[u].push_back(v);
}
void dfs(int u,int f){
    int gz=G[u].size();
    for(int i=0,to;i<gz;i++){
        to = G[u][i];
        if(to == f) continue;
        dfs(to,u);
    }
}

//前向星  空间复杂度 O(e)
struct edge{
    int to,next,w;
}G[N<<1];
int head[N],tot;
void add(int u,int v,int w){
    G[tot].w=w,G[tot].to=v,G[tot].next=head[u],head[u]=tot++;
}
void dfs(int u,int f){
    for(int i=head[u],to;i!=-1;i=G[i].next){
        to = G[i].to;
        if(to == f) continue;
        dfs(to,u);
    }
}
```

2 Shortest Path

2.1 SPFA

```
LL d[N];
int inq[N],n,m,q;
LL spfa(int s,int t){
    for(int i=1;i<=n;i++) d[i]=INF,inq[i]=0;
    queue<int >q;
    q.push(s);d[s]=0,inq[s]=1;
    while(!q.empty()){
        int u =q.front();q.pop();
        inq[u]=0;
        for(int i=head[u],v;i!=-1;i=G[i].next){
            v=G[i].to;
            if(d[v]>d[u]+G[i].w){
                d[v]=d[u]+G[i].w;
                if(inq[v]==1) continue;
                inq[v]=1;
                q.push(v);
```

```
            }
        }
    }
    return d[t];
}
```

2.2 Dijkstra

```
vector<pair<int ,int > >G[N];
LL d[N];
LL dijkstra(int s,int t){
    for(int i=1;i<=n;i++)d[i]=INF;
    d[s]=0;
    priority_queue<pair<int ,int> >q;
    q.push(make_pair(-d[s],s));
    while(!q.empty()){
        int u =q.top().second;q.pop();
        if(u==t) return d[t];
        for(int i=0,v;i<G[u].size();i++){
            v=G[u][i].first;
            if(d[v]>d[u]+G[u][i].second){
                d[v]=d[u]+G[u][i].second;
                q.push(make_pair(-d[v],v));
            }
        }
    }
    return d[t];
}
```

3 Minimum Spanning Tree

3.1 Kruskal

```
/*****************************************************
kruskal 适用于稀疏图，边数接近 T(k*n) 时采用 （k 为常数
*****************************************************/
struct node {
    int u,v,w;
}e[N*4];
bool cmp(node a,node b){
    return a.w>b.w;
}
int pre[N];
int findi(int x){
    int r=x;
    while(r!=pre[r])r=pre[r];
    for(int i=x,j;i!=r;i=j){
        j=pre[i];
        pre[i]=r;
    }
    return r;
}
void join(int x,int y){
    x=findi(x),y=findi(y);
    pre[x]=y;
}
void MST(){
    sort(e+1,e+m+1,cmp);
```

```
    for(int i=1,sum=0;i<=m&&sum<n-1;i++){
        if(findi(e[i].u)==findi(e[i].v)) continue;
        join(e[i].u,e[i].v);
        add(e[i].u,e[i].v,e[i].w);
        sum++;
    }
}
```

3.2 Prim

```
/*****************************************************
Prim 适用于稠密图，边数达到/接近 T(n^2) 时建议采用
*****************************************************/

#define MAX 100
#define MAXCOST 0x7fffffff

int graph[MAX][MAX];
int prim(int graph[][MAX], int n) {
    int lowcost[MAX];
    int mst[MAX];
    int i, j, min, minid, sum = 0;
    for (i = 2; i <= n; i++) {
        lowcost[i] = graph[1][i];
        mst[i] = 1;
    }
    mst[1] = 0;
    for (i = 2; i <= n; i++) {
        min = MAXCOST;
        minid = 0;
        for (j = 2; j <= n; j++) {
            if (lowcost[j] < min && lowcost[j] != 0) {
                min = lowcost[j];
                minid = j;
            }
        }
        cout << "V" << mst[minid] << "-V" << minid << "=" << min << endl;
        sum += min;
        lowcost[minid] = 0;
        for (j = 2; j <= n; j++) {
            if (graph[minid][j] < lowcost[j]) {
                lowcost[j] = graph[minid][j];
                mst[j] = minid;
            }
        }
    }
    return sum;
}

/****
带输出路径的 prim
****/
double map[800][800];
double weight[1000];
int flag[1000];
void prim()  {
    memset(weight,INF,sizeof(weight));
    int pos=1;
```

```
    double min_weight;
    int p;
//--------------表示遍历了第一个点-------------------//
    for(int i=1;i<=n;i++){
        weight[i]=map[pos][i];
        flag[i]=pos;//记录它的父亲节点
    }
    flag[pos]=-1;
//----------------寻找最短的一截路---------------//
    for(int i=1;i<n;i++){
        min_weight=INF;
        p=-1;

        for(int j=1;j<=n;j++){
            if(flag[j]!=-1&&min_weight>weight[j]){
                min_weight=weight[j];
                p=j;
            }
        }
//-------------判断是否有短路到达---------------//
        if(p!=-1){
            if(min_weight==0)
                flag[p]=-1;
            else{
                printf("%d %d\n",flag[p],p);
                flag[p]=-1;
            }
        }
//----------------更新一截路---------------------//
        for(int j=1;j<=n;j++)
        if(flag[j]!=-1&&weight[j]>map[p][j]){
            weight[j]=map[p][j];
            flag[j]=p;
        }
    }
}
```

4 Directed Graph

4.1 Topological Sorting

```
/*************************************************************
拓扑排序 (Topological Sorting) 是一个有向无环图 (DAG, Directed Acyclic Graph) 的所有顶点的线性
↪ 序列。
该序列必须满足下面两个条件:
1. 每个顶点出现且只出现一次。
2. 若存在一条从顶点 A 到顶点 B 的路径，那么在序列中顶点 A 出现在顶点 B 的前面。

是'特殊'的图中的一种，可以在排序过程中进行计算一些值.
*************************************************************/

void topo(){
    queue<int>q;
    for(int i=1;i<=n;i++)if(0==deg[i]) q.push(i);

    for(int u;!q.empty();){
        u=q.front();
        for(int i=head[u],to;i!=-1;i=G[i].next){
            to = G[i].to;
            deg[to]--;
            if(0==deg[to]) q.push(to);
        }
    }
}
```

5 Bipartite Graph

5.1 Hungary

```
/*****************************************************
二分图匹配

算法介绍
最大匹配数    : 最大匹配的匹配边的数目
最小点覆盖数  : 选取最少的点，使任意一条边至少有一个端点被选择
最大独立数    : 选取最多的点，使任意所选两点均不相连
最小路径覆盖数: 对于一个 DAG (有向无环图)，选取最少条路径，使得每个顶点属于且仅属于一条路径。路径
↪  长可以为 0 (即单个点)。

定理 1: 最小点覆盖数   = 最大匹配数 (这是 Konig 定理)
定理 2: 最大独立数     = 顶点数 - 最大匹配数
定理 3: 最小路径覆盖数 = 顶点数 - 最大匹配数

独立集: 任意两点都不相连的顶点的集合
独立数: 独立集中顶点的个数
完全子图: 任意两点都相连的顶点的集合
最大完全数: 最大完全子图中顶点的个数
最大完全数 = 原图的补图的最大独立数
最大独立数 = 顶点数-最大匹配数

匈牙利算法求解二分图匹配问题
*****************************************************/
int k,n,m;
int match[N];
int vis[N];
vector<int >G[N];

int tot,a;
int col[N];
//二分图染色  用来判断是不是二分图
bool color(int s){
    queue<int>q;
    q.push(s);col[s]=1;
    while(!q.empty()){
        int u = q.front();q.pop();
        tot++;
        if(col[u]==1) a++;
        int gz=G[u].size();
        for(int i=0,to;i<gz;i++){
            to=G[u][i];
            if(col[to]==0){
                col[to]=3-col[u];
                q.push(to);
            }
            else if(col[to]==col[u])
                return false;
        }
    }
    return true;
}

bool findi(int u){
    for(int i=0;i<G[u].size();i++){
        int v=G[u][i];
```

```
        if(vis[v]==0){
            vis[v]=1;
            if(match[v]==-1||findi(match[v])){
                match[v]=u;
                return true;
            }
        }
    }
    return false;
}

void init(){
    memset(match,-1,sizeof(match));
    for(int i=1;i<=n;i++) G[i].clear(),col[i]=vis[i]=0;
}

void add(int u,int v){
    G[u].push_back(v);
    G[v].push_back(u); //无向图才需要
}

int maxMATCH(){
    int ans = 0;  //ans 是最大匹配数,
    for(int i=1;i<=n;i++){
        memset(vis,0,sizeof(vis));
        if(findi(i)) ans++;
    }
    ans/=2;   //对于无向图的话 需要 ans/=2;
    return ans ;
}

bool perfectMATCH(){
    if(maxMATCH()==n) return true;
    else              return false;
}
```

5.2 Two Set

```
vector<int> G[N];
vector<int> G2[N];

vector<int> S;
int vis[N] ;  //标记数组
int sccno[N]; //记录数组 sccno[i] 记录  i 属于哪个联通分支
int scc_cnt;
int n;
void dfs1(int u){
    if(vis[u]) return ;
    vis[u] = true;
    for(int i = 0; i < G[u].size(); i++){
        dfs1(G[u][i]);
    }
    S.push_back(u);
}
void dfs2(int u){
    if(sccno[u]) return;
    sccno[u] = scc_cnt;
    for(int i = 0; i < G2[u].size(); i++){
```

```
        dfs2(G2[u][i]);
    }
}
void find_scc(int n){
    scc_cnt = 0;
    S.clear();
    memset(vis , 0 , sizeof(vis));
    memset(sccno , 0 , sizeof(sccno));
    for(int i = 0; i < n; i++) dfs1(i);
    for(int i = n - 1; i >= 0; i--) if(!sccno[S[i]]){
        scc_cnt++;
        dfs2(S[i]);
    }
}
void AddEdge(int u , int v) {
    G[u].push_back(v);
    G2[v].push_back(u);
}
int solve(){
    /***
    输入数据
    */
    /***
    加边  边为两者不能匹配的
    example :  {u,v}  当选择 u 的时候一定不能选择 v , 需要选择 v';
    */
    find_scc(2 * n);  //注意 x2

    for(int i = 0; i < n; i++){
        if(sccno[i] == sccno[i + n]){
            //如果有 x 与 x' 同时被取或者未取, 则匹配失败
            puts("NO");
            return 0;
        }
    }
    puts("YES");
    for(int i = 0; i < n; i++){
        if(sccno[i] > sccno[i + n]){
            //相关 i;
        }
        else {
            //相关 i';
        }
    }
    return 0;
}
```

6 Strongly Connected

6.1 tarjon

```
vector<int>G[N];
vector<pair<int,int> >G2[N];

int low[N],dfn[N],w[N],d[N];
int vis[N],val[N];
int color[N],cnt,tot;
int mystack[N],len;

void dfs(int u){vis[u]=1;
    dfn[u]=low[u]=++cnt;
    mystack[++len]=u;
    int gz=G[u].size();
    for(int to,i=0;i<gz;i++){
        to=G[u][i];
        if(vis[to]==0) dfs(to);
        if(vis[to]==1) low[u]=min(low[u],low[to]);
    }

    if(dfn[u]==low[u]){
        ++tot;
        do{
            val[tot]+=w[mystack[len]];
            color[mystack[len]]=tot;
            vis[mystack[len]]=2;
        }while(mystack[len--]!=u);
    }
}

int n,m;
int main(){
    scanf("%d%d",&n,&m);
    cnt=tot=len=0;
    for(int i=1;i<=n;i++)    scanf("%d",&w[i]);

    for(int i=1,u,v;i<=m;i++){
        scanf("%d%d",&u,&v);
        G[u].push_back(v); //看是有向图还是无向图
    }

    //缩点
    for(int i=1;i<=n;i++)if(vis[i]==0) dfs(i);

    //建新图 G2

    for(int i=1,gz;i<=n;i++){vis[i]=0,d[i]=0;
        gz=G[i].size();
        for(int j=0,to;j<gz;j++){
            to=G[i][j];
            if(color[i]!=color[to])
                G2[color[i]].push_back(make_pair(val[color[to]],color[to]));
        }
    }

    return 0;
}
```

7 Tree

7.1 centre of gravity of tree

```
/*****************************************************
树的重心

树的重心：找到一个点，其所有的子树中最大的子树节点数最少，那么这个点就是这棵树的重心，删去重心后，
↪ 生成的多棵树尽可能平衡。
树的重心可以通过简单的两次搜索求出，第一遍搜索求出每个结点的子结点数量 son[u]，第二遍搜索找出使
↪ max{son[u],n-son[u]-1} 最小的结点。
实际上这两步操作可以在一次遍历中解决。对结点 u 的每一个儿子 v，递归的处理 v，求出 son[v]，然后判
↪ 断是否是结点数最多的子树，处理完所有子结点后，判断 u 是否为重心。
例题 poj 1655
*****************************************************/
int n;

vector<int >G[N];
void add(int u,int v){G[u].push_back(v);}

int sz[N],siz,zx;

void dfs(int u,int f=0){
    sz[u]=1;int mx=0;
    int gz=G[u].size();
    for(int i=0,to;i<gz;i++){
        to = G[u][i];
        if(to == f) continue;
        dfs(to,u);
        sz[u]+=sz[to];
        mx=max(mx,sz[to]);
    }
    mx=max(mx,n-sz[u]);
    if(mx==siz&&u<zx) zx=u;
    if(mx<siz) zx=u,siz=mx;
}

int main(){
    int _;
    for(scanf("%d",&_);_--;){
        scanf("%d",&n);
        for(int i=1,u,v;i<n;i++){
            scanf("%d%d",&u,&v);
            add(u,v);add(v,u);
        }
        siz = n*10; dfs(1);
        printf("%d %d\n",zx,siz);
        for(int i=1;i<=n;i++) G[i].clear();
    }
    return 0;
}
```

7.2 Least Common Ancestors

```
// 一. 树链剖分
// 复杂度 O(n)-O(\log{n})

// 代码略

// 二. 倍增求 lca
// 复杂度 O(n\log{n})-O(\log{n})

int fa[N][20],dep[N],mi[N][20];//(顺带求路上边长最小值)
void dfs(int u,int f,int d){
    fa[u][0]=f;dep[u]=d;
    for(int i=1;i<20;i++)
        fa[u][i]=fa[fa[u][i-1]][i-1],mi[u][i]=min(mi[u][i-1],mi[fa[u][i-1]][i-1]);
    int gz=G[u].size();
    for(int i=0,to;i<gz;i++){
        to=G[u][i].first;
        if(to==f)continue;
        mi[to][0]=G[u][i].second;
        dfs(to,u,d+1);
    }
}
void solve(int u,int v){
    if(dep[u]<dep[v]) swap(u,v);
    int lca=-1,tu=u,tv=v;

    for(int i=19;i>=0;i--)
        if(dep[fa[u][i]]>=dep[v])
            u=fa[u][i];

    if(u==v) lca=v;// return v;
    else{
        for(int i=19;i>=0;i--){
            if(fa[u][i]==fa[v][i]) continue;
            u=fa[u][i];
            v=fa[v][i];
        }
        lca = fa[v][0];
    }
    int miu=INF,miv=INF;
    for(int i=19;i>=0;i--){
        if(dep[fa[tu][i]]>=dep[lca])
            miu=min(miu,mi[tu][i]),tu=fa[tu][i];
        if(dep[fa[tv][i]]>=dep[lca])
            miv=min(miv,mi[tv][i]),tv=fa[tv][i];
    }
    printf("%d\n",min(miu,miv));
}

// 三. lca 转 rmq
// 复杂度 O(n\log{n})-O(\log{n})

vector<int> g[N], sp;
int dep[N], dfn[N], lg[2 * N];
pair<int, int> dp[21][2 * N];

void dfs(int u, int fa) {
    dep[u] = dep[fa] + 1;
```

```
    dfn[u] = sp.size();
    sp.push_back(u);
    for (auto v : g[u]) {
        if (v != fa) {
            dfs(v, u);
            sp.push_back(u);
        }
    }
}

void lca_init() {
    int n=sp.size();
    for (int i=1,j=0;i<=n;i++) {
        if (i==(1<<(j+1)))j++;
        lg[i]=j;
    }
    for(int i=0;i<n;i++){
        dp[0][i]={dfn[sp[i]],sp[i]};
    }
    for(int j=1; (1<<j)<=n;j++) {
        for (int i=0;i+(1<<j)-1<n;i++) {
        dp[j][i]=min(dp[j-1][i],dp[j-1][i+(1<<(j-1))]);
        }
    }
}

int lca(int u, int v) {
    int l=dfn[u],r=dfn[v];
    if (l>r) swap(l,r);
    int j=lg[r-l+1];
    return min(dp[j][l],dp[j][r-(1<<j)+1]).second;
}
```

8 Network Flows

8.1 Dinic

8.2 ISAP

```
#include <stdio.h>
#include <string.h>
#include <algorithm>
#define clear(A, X) memset (A, X, sizeof A)
#define copy(A, B) memcpy (A, B, sizeof A)
using namespace std;

const int maxE = 1000000;
const int maxN = 100000;
const int maxQ = 1000000;
const int oo = 0x3f3f3f3f;

struct Edge {
   int v;//弧尾
   int c;//容量
   int n;//指向下一条从同一个弧头出发的弧
} edge[maxE];//边组
```

```
int adj[maxN], cntE;//前向星的表头
int Q[maxQ], head, tail;//队列
int d[maxN], cur[maxN], pre[maxN], num[maxN];
int sourse, sink, nv;//sourse: 源点, sink: 汇点, nv: 编号修改的上限
int n, m;

void addedge (int u, int v, int c) {//添加边
    //正向边
    edge[cntE].v = v;
    edge[cntE].c = c;//正向弧的容量为 c
    edge[cntE].n = adj[u];
    adj[u] = cntE++;

    //反向边
    edge[cntE].v = u;
    edge[cntE].c = 0;//反向弧的容量为 0
    edge[cntE].n = adj[v];
    adj[v] = cntE++;
}

void rev_bfs () {//反向 BFS 标号
    clear (num, 0);
    clear (d, -1);//没标过号则为-1
    d[sink] = 0;//汇点默认为标过号
    num[0] = 1;
    head = tail = 0;
    Q[tail++] = sink;

    while (head != tail) {
        int u = Q[head++];
        for (int i = adj[u]; ~i; i = edge[i].n) {
            int v = edge[i].v;
            if (~d[v]) continue;//已经标过号
            d[v] = d[u] + 1;//标号
            Q[tail++] = v;
            num[d[v]]++;
        }
    }
}

int ISAP() {
    copy (cur, adj);//复制, 当前弧优化
    rev_bfs ();//只用标号一次就够了, 重标号在 ISAP 主函数中进行就行了
    int flow = 0, u = pre[sourse] = sourse, i;

    while (d[sink] < nv) {//最长也就是一条链, 其中最大的标号只会是 nv - 1, 如果大于等于 nv 了说
    ↪  明中间已经断层了。
        if (u == sink) {//如果已经找到了一条增广路, 则沿着增广路修改流量
            int f = oo, neck;
            for (i = sourse; i != sink; i = edge[cur[i]].v) {
                if (f > edge[cur[i]].c){
                    f = edge[cur[i]].c;//不断更新需要减少的流量
                    neck = i;//记录回退点, 目的是为了不用再回到起点重新找
                }
            }
            for (i = sourse; i != sink; i = edge[cur[i]].v) {//修改流量
```

```
                edge[cur[i]].c -= f;
                edge[cur[i] ^ 1].c += f;
            }
            flow += f;//更新
            u = neck;//回退
        }
        for (i = cur[u]; ~i; i = edge[i].n) if (d[edge[i].v] + 1 == d[u] && edge[i].c)
        ↪ break;
        if (~i) {//如果存在可行增广路，更新
            cur[u] = i;//修改当前弧
            pre[edge[i].v] = u;
            u = edge[i].v;
        }
        else {//否则回退，重新找增广路
            if (0 == (--num[d[u]])) break;//GAP 间隙优化，如果出现断层，可以知道一定不会再有增
            ↪ 广路了
            int mind = nv;
            for (i = adj[u]; ~i; i = edge[i].n) {
                if (edge[i].c && mind > d[edge[i].v]) {//寻找可以增广的最小标号
                    cur[u] = i;//修改当前弧
                    mind = d[edge[i].v];
                }
            }
            d[u] = mind + 1;
            num[d[u]]++;
            u = pre[u];//回退
        }
    }

    return flow;
}

void init () {//初始化
    clear (adj, -1);
    cntE = 0;
}

void work () {
    int u, v, c;
    init ();
    for (int i = 0; i < m; ++ i) scanf ("%d%d%d", &u, &v, &c), addedge (u, v, c);
    sourse = 1; sink = n; nv = sink + 1;
    printf ("%d\n", ISAP ());
}
int main() {
    while (~scanf("%d%d", &m, &n)) work ();
    return 0;
}
```

Part II
String

9 String Match

9.1 KMP

```
// KMP

char s1[N],s2[N];
int Next[N];//Next[i] 表示从 [0~i] 中最长公共前缀的长.
void get_next(char *s,int len){
    for(int i=0,j=-1;i<=len;++i,++j){
        Next[i]=j;
        while(j>=0&&s[i]!=s[j]) j = Next[j];
    }
//    for(int i=0;i<=len;i++) printf("%d%c",Next[i],(i==len)?'\n':' ');
}
//在串 s 上找 sz
int KMP (char *s,int len,char *sz,int l){
    int i=0,j=0,cnt=0;
    while(i<len/*&&j<l*/){
        if(s[i]==sz[j])  i++,j++;
        else{
            if(0==j)  i++;
            else      j=Next[j];
        }
        if(j==l) cnt++;
    }

//   return (j==l)?(i-l+1):-1; //找第一次出现的位置
    return cnt; //找出现的个数
}
/*
            for(int j=0,i=0;i<=len;i++,j++){
                ans ^= 1LL*(j)*(j)*(len-k-j)*(i-j) ;
                while(j>=0&&a[i]!=p[j]) j=Next[j];
            }
*/
```

9.2 Trie

```
/*****************************
字典树 trie
指针实现
双数组
可持久化
*****************************/

// point
const int N = 1010101;
const int Max = 26;

typedef struct node{
    struct node *next[Max];
    int num;
}Node;
```

```
//创建一个新节点
Node *createNew(){
    Node *p=new Node;
    for(int i=0;i<Max;i++) p->next[i]=NULL;
    p->num=0;
    return p;
}
Node *head;

//插入一个字符串
void Insert_str(char str[]){
    int len=strlen(str);
  //  printf("len = %d--  ",len);
    Node *t,*p=head;
    for(int i=0;i<len;i++){
        int c=str[i]-'a';
        if( p->next[c] == NULL ){
            //lalal
            t=createNew();
            p->next[c]=t;
        }
        p=p->next[c];
        p->num++;
       // cout<<p->num<<"-"<<str[i]<<" ";
    }
}

int Search_str(char str[]){
    Node *p=head;
    int len=strlen(str);

    int counts=0;
    for(int i=0;i<len;i++){
        int c=str[i]-'a';
        if(p->next[c]==NULL){
        //  cout<<" 不存在字符串"<<endl;
            counts=0;
            return 0;
        }
        else{
            p=p->next[c];
            counts=p->num;
            //printf("%d ",p->num);
        }
    }
    return counts;
}

//Double Arrays Trie-DAT
struct TRIE{
    int ch[N][30],cnt;

    int newnode(){
        ++cnt;
        memset(ch[cnt],0,sizeof(ch[cnt]));
        return cnt;
    }
```

```
    void init(){
        cnt = 0;
        memset(ch[cnt],0,sizeof(ch[cnt]));
    }

    void insert(char *s){
        int now = 0;
        for(int i=0;s[i];i++){
            if(!ch[now][s[i]-'a'])
                ch[now][s[i]-'a']=newnode();
            now=ch[now][s[i]-'a'];
        }
    }

    int match(char *s){
        int now = 0,ans = 0;
        for(int i=0;s[i];i++){
            if(!ch[now][s[i]-'a']) return i;
            now=ch[now][s[i]-'a'];
            ans++;
        }
        return ans;
    }
}trie;


//可持久化 01 字典树

int trie[N*20][2],sz[N*20],rt[N],cnt;
//将 x 插入到 trie ,i 为位数 (下同)
void update(int pre,int i,int x){
    int now = ++ cnt;
    if(0==i){
        trie [now][0]=trie [now][1]=0;
        sz[now]=sz[pre]+1;
        return ;
    }
    int bt = (x>>(i-1))&1;
    trie[now][1-bt]=trie[pre][1-bt];
    trie[now][bt]  =update(trie[pre][bt],i-1,x);
    sz[now]=sz[trie[now][0]]+sz[trie[now][1]];
    return now;
}
//查询 [l,r] 区间与 x 异或结果最大的值, ss=rt[l-1],tt=rt[r];
int query(int ss,int tt,int i,int x){
    if(0==i) return 0;
    int bt = (x>>(i-1))&1;
    if(sz[trie[tt][1-bt]]-sz[trie[ss][1-bt]])
        return (1<<(i-1))+query(trie[ss][1-bt],trie[tt][1-bt],i-1,x);
    else
        return query(trie[ss][bt],trie[tt][bt],i-1,x);
}
int main(){
    //初始化
    memset(sz,0,sizeof(sz));
    trie[0][0]=trie[0][1]=tr[0]=0;
    rt[0]=update(rt[0],25,0);
```

```
    for(int i=1,x;i<=n;i++){
        scanf("%d",&x);
        rt[i]=update(rt[i-1],25,x);
    }
}
```

9.3 Aho Corasick Auto Mation

```
struct  Aho_Corasick_automaton{
    int ch[N][130],val[N],fail[N],cnt,root;
    int newnode(){
        memset(ch[cnt],-1,sizeof(ch[cnt]));
        fail[cnt]=val[cnt++]=-1;
        return cnt-1;
    }
    void init(){
        cnt = 0;
        root = newnode();
    }
    void insert(char *s,int id){
        int now = root;
        for(int i=0;s[i];i++){
            if( ch[now][s[i]] == -1)
                ch[now][s[i]] = newnode();
            now=ch[now][s[i]];
        }
        val[now]=id;
    }
    void build(){
        fail[root]=root;
        queue<int>q;
        for(int i=0;i<130;i++){
            if(-1==ch[root][i])ch[root][i]=root;
            else {
                fail[ch[root][i]]=root;
                q.push(ch[root][i]);
            }
        }
        for(int now;!q.empty();){
            now=q.front();q.pop();
            for(int i=0;i<130;i++){
                if(-1==ch[now][i]) ch[now][i]=ch[fail[now]][i];
                else {
                fail[ch[now][i]]=ch[fail[now]][i];
                q.push(ch[now][i]);
                }
            }
        }
    }
    int vis[1000];
    int Get(int x){
        int flag = 0;
        for(;x!=root;x=fail[x])
            if(val[x]!=-1){
                vis[val[x]]=1;
                flag = 1;
```

```
            }
        return flag;
    }
    int match(char *s){
        int now = root,flag = 0;
        memset(vis,0,sizeof(vis));
        for(int i=0;s[i];i++){
            now = ch[now][s[i]];
            flag |= Get(now);
        }
        return flag;
    }
}ac;
int main(){
    ac.init();            // 初始化自动机
    for(;;)ac.insert(s); // 插入子串 s
    ac.build();           // 构造 fail
    ac.match(s);          // 对串 s 进行匹配
}
```

9.4 Suffix Arrays

```
/*********************************************
后缀数组
SA[i] = j  : 表示为按照从小到大排名为 i 的后缀 是以 j(下标) 开头的后缀
rank[i] = j: 表示为按照从小到大排名 以 i 为下标开始的后缀 排名为 j
Heigth[i]  : 表示 Suffix[SA[i]] 和 Suffix[SA[i - 1]] 的最长公共前缀, 也就是排名相邻的两个后缀
↪  的最长公共前缀
RANK 表示你排第几 SA 表示排第几的是谁 (记住这个就行)
*********************************************/
const int MAXN=400010;
//以下为倍增算法求后缀数组
int wa[MAXN],wb[MAXN],wv[MAXN],Ws[MAXN];
int cmp(int *r,int a,int b,int l) {
    return r[a]==r[b]&&r[a+l]==r[b+l];
}
/**< 传入参数: str,sa,len+1,ASCII_MAX+1 */
void da(const int r[],int sa[],int n,int m) {
    int i,j,p,*x=wa,*y=wb,*t;
    for(i=0; i<m; i++) Ws[i]=0;
    for(i=0; i<n; i++) Ws[x[i]=r[i]]++;//以字符的 ascii 码为下标
    for(i=1; i<m; i++) Ws[i]+=Ws[i-1];
    for(i=n-1; i>=0; i--) sa[--Ws[x[i]]]=i;
    /*cout<<"SA"<<endl;;
    for(int i=0;i<n+1;i++)cout<<sa[i]<<' ';*/
    for(j=1,p=1; p<n; j*=2,m=p) {
        for(p=0,i=n-j; i<n; i++) y[p++]=i;
        for(i=0; i<n; i++) if(sa[i]>=j) y[p++]=sa[i]-j;
        for(i=0; i<n; i++) wv[i]=x[y[i]];
        for(i=0; i<m; i++) Ws[i]=0;
        for(i=0; i<n; i++) Ws[wv[i]]++;
        for(i=1; i<m; i++) Ws[i]+=Ws[i-1];
        for(i=n-1; i>=0; i--) sa[--Ws[wv[i]]]=y[i];
        for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1; i<n; i++)
            x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;
    }
    return;
}
```

```
int sa[MAXN],Rank[MAXN],height[MAXN];
//求 height 数组
/**< str,sa,len */
void calheight(const char *r,int *sa,int n) {
    int i,j,k=0;
    for(i=1; i<=n; i++) Rank[sa[i]]=i;
    for(i=0; i<n; height[Rank[i++]]=k)
        for(k?k--:0,j=sa[Rank[i]-1]; r[i+k]==r[j+k]; k++);
    // Unified  不要乱用，出来检查为了方便的时候 否则容易 RE, WA
//    for(int i=n; i>=1; --i) ++sa[i],Rank[i]=Rank[i-1];

}

//求 lcp(suffixal(i),suffixal(j))

int mm[MAXN],dp[MAXN][20];
void initrmq(int n,int b[]) {
    mm[0]=-1;
    for(int i=1; i<=n; i++) {
        mm[i]=((i&(i-1))==0)?mm[i-1]+1:mm[i-1];
        dp[i][0]=b[i];
    }
    for(int j=1; j<=mm[n]; j++)
        for(int i=1; i+(1<<j)-1<=n; i++)
            dp[i][j]=min(dp[i][j-1],dp[i+(1<<(j-1))][j-1]);
}

int lcp(int x,int y) {
    x=Rank[x],y=Rank[y];
    if(x>y) swap(x,y); x++;
    int k=mm[y-x+1];
    return min(dp[x][k],dp[y-(1<<k)+1][k]);
}

char s[N];
int a[N];

int main(){
    scanf("%s",s);
    int ls = strlen(s);
    for(int i=0;i<ls;i++) a[i]=s[i]-'a'+1; a[ls]=0;

    da(a,sa,ls+1,30);
    calheight(a,sa,ls);
    initrmq(ls+1,height);
    return 0;
}
```

9.5 Suffix Automaton

```
/***
 * 1 call init()
 * 2 call add(x) to add every character in order
 *
 * Args:
 * Return:
 *   an automaton
 *   link: link path pointer
 *    len: maximum length
 */
struct node{
  node* chd[26], *link;
  int len;
}a[3*N], *head, *last;
int top;
void init(){
    memset(a, 0, sizeof(a));
    top = 0;
    head = last = &a[0];
}
void add(int x){
        node *p = &a[++top], *mid;
        p->len = last->len + 1;
        mid = last, last = p;
        for (; mid && !mid->chd[x]; mid = mid->link)  mid->chd[x] = p;
        if (!mid)  p->link = head;
        else{
            if (mid->len + 1 == mid->chd[x]->len) {
            p->link = mid->chd[x];
        } else {
            node *q = mid->chd[x], *r = &a[++top];
            *r = *q, q->link = p->link = r;
            r->len = mid->len + 1;
            for (; mid && mid->chd[x] == q; mid = mid->link)  mid->chd[x] = r;
        }
    }
}
```

10 Manachar

```
/***
Manacher 是一个能在 O(n) 的复杂度内解决字符串中最长回文子串的问题

str  原字符串
a    进行 Manacher 算法的为减轻编码难度改进的字符串
p[i] 以 a[i] 为中心的回文串的半径 (含 a[i])
id   最长回文子串的中心位置
mx   id+p[id] 最长回文子串的左外边界。

算法核心的部分
if(mx > i) p[i] = (p[(id<<1)-i]<(mx-i))?p[(id<<1)-1]:(mx-i);
else p[i]=1;

首先我们知道对于一个字符串进行操作的时候，是从左到右依次进行的
那么由于回文串的对称性，
那么可以确定的是对于 a[i] 为中心的回文串的长度至少是在以这个最大的回文串中对称的那个位置为中心的
↪  回文串的长度，但只能确定的是在最大的回文串的那一部分中的长度，至于之外的就要一个个的匹配了
*/
/***
id  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
a:  $ # 1 # 2 # 2 # 1 # 2 # 3 # 2 # 1 #
p_i 1 1 2 1 2 5 2
*/
char a[N],str[N];
int p[N];
void Manacher(){
    int len  = 0;
    a[len] = '$', a[++len] = '#';
    int slen = strlen(str)-1;
    Rep(i,0,slen) a[++len] = str[i],a[++len] = '#';

    memset(p,0,sizeof(p));
    int id = -1,mx = -1,mxp = -1;
    Rep(i,0,len){
        if(mx > i) p[i] = (p[(id<<1)-i]<(mx-i))?p[(id<<1)-i]:(mx-i);
        else p[i]=1;
        while(i-p[i]>=0&&i+p[i]<=len&&a[i-p[i]]==a[i+p[i]]) p[i]++;
        if(p[i]+i>mx) mx=p[i]+i,id=i;
        if(p[i]>mxp) mxp = p[i];
    }
    printf("%d\n",mxp-1);
    return ;
}
```

Part III
Data structure

11 UFS

```
//一般并查集
int pre[N];
int findi(int x){
    return pre[x]=(x==pre[x])?x:findi(pre[x]);
}
void join(int x,int y){
    int fx = findi(x),fy = findi(y);
    pre[fx]=fy;
    return ;
}

//带权并查集

int pre[N];
int findi(int x){
    if(pre[x]==x) return x;
    int r = findi(pre[x]);
    /**
    省略了权值间关系转化，具体视情况而定
    */
    pre[x]=r;
    return r;
}


//并查集（可拆点）

int pre[N],h[N],hh;
int findi(int x){
    if(pre[x]==x) return x;
    int r = findi(pre[x]);
    pre[x]=r;
    return r;
}
void join(int x,int y,int X,int Y){
    int fx = findi(x),fy = findi(y);
    pre[fx]=fy;
    return ;
}
void creat(int now){
    h[now]=++hh;
    pre[hh]=hh;
}

//可持久化
int n , m ;
struct node {
    int ls , rs ;
    int l , r ;
    int fa ;
    int dep ;
}a[200000 * 50];
```

```
int cnt ;
int root[200050];

int build(int l,int r) {
    ++cnt ;
    a[cnt].l = l ;a[cnt].r = r ;
    a[cnt].fa = l ;a[cnt].dep = 0 ;
    if(l == r)   return cnt;
    int mid = (l + r) >>1 ;
    int z = cnt ;
    a[z].ls = build(l , mid) ;
    a[z].rs = build(mid + 1 , r) ;
    return z ;
}

void upda(int x , int &y , int pos , int val) {
    y = ++cnt ;
    if(a[x].l == a[x].r) {
        a[y] = a[x] ;
        a[y].fa = val ;
        return ;
    }
    a[y] = a[x] ;
    int mid = (a[y].l + a[y].r) >>1 ;
    if(pos <= mid) upda(a[x].ls , a[y].ls , pos , val) ;
    else           upda(a[x].rs , a[y].rs , pos , val) ;
    return ;
}

int query(int x , int pos) {
    if(a[x].l == a[x].r) return x ;
    int mid = (a[x].l + a[x].r) >>1 ;
    if(pos <= mid) return query(a[x].ls , pos) ;
    else return query(a[x].rs , pos) ;
}

int find_(int x , int pos) {
    int p = query(x , pos) ;
    if(a[p].fa != pos)  return find_(x , a[p].fa) ;
    else return p ;
}

void add(int x , int pos) {
    if(a[x].l == a[x].r) {
        a[x].dep ++ ;
        return ;
    }
    int mid = (a[x].l + a[x].r) >>1 ;
    if(pos <= mid) add(a[x].ls , pos) ;
    else add(a[x].rs , pos) ;
}

int main() {
    while(scanf("%d%d",&n,&m) != EOF) {
        cnt=0;
        root[0]=build(1,n);
        int last=0;
```

```
        for(int i=1,op,q,w,k;i<=m;i++){
            scanf("%d",&op) ;
            if(op == 1) {
                scanf("%d%d",&q,&w) ;
                q ^= last ;
                w ^= last ;
                root[i] = root[i-1] ;
                int pp = find_(root[i] , q) ;
                int qq = find_(root[i] , w) ;
                if(a[pp].dep > a[qq].dep) swap(pp , qq) ;

                if(a[pp].fa == a[qq].fa) continue ;

                upda(root[i-1] , root[i] , a[pp].fa , a[qq].fa) ;

                if(a[pp].dep == a[qq].dep)
                    add(root[i],a[qq].fa);
            }
            else if(op == 2) {
                scanf("%d",&k) ;
                root[i] = root[k^last] ;
            }
            else {
                scanf("%d%d",&q,&w);
                q^=last;w^=last;
                root[i]=root[i-1];
                int fq=find_(root[i-1],q);
                int fw=find_(root[i-1],w);
                if(fq!=fw) last=0;
                else last=1;
                printf("%d\n",last) ;
            }
        }
    }
}
```

12 BIT

```
//一维树状数组
const int N = 50000 + 5;                //数列的大小
#define lowbit(x)  (x&(-x))             //lowbit 操作
int sum[N],cnt;
void update(int index,int val){     //单点更新   (+val)
    for(int i=index;i<N;i+=lowbit(i)){//i<=N   不能 <=cnt<--错了
        sum[i]+=val;
    }
}
int getSum(int index) {                 //求解 1~index 的和
  int ans = 0;
  for (int i = index; i; i -= lowbit(i))
    ans += sum[i];
  return ans;
}
void update(int l,int r,int val){
    update(l,val),update(r+1,-val);
}
int query(int l,int r){
    return getSum(r)-getSum(l-1);
```

```
}
/*
一维区间更新 (a,b)
update(a,1);
update(b+1,-1);
% 注意这种情况下 不能区间查询
*/

//二维区间更新
const int N = 1000+5;
#define lowbit(x) (x&-x)
LL sum[N][N];
void update(int xi,int yi,int val){
    for(int i=xi;i<=N;i+=lowbit(i))
        for(int j=yi;j<=N;j+=lowbit(j))
            sum[i][j]+=val;
    return;
}
int getSum(int xi,int yi){//就是 (x,y) 这个位置的值
    int ans = 0;
    for(int i=xi;i>0;i-=lowbit(i))
        for(int j=yi;j>0;j-=lowbit(j))
            ans+=sum[i][j];
    return ans ;
}
void update(int x,int y,int X,int Y,int val){
    update(x,y,val);
    update(x,Y+1,-val);
    update(X+1,y,-val);
    update(X+1,Y+1,val);
}
/*
二维区间更新 {(a,b)|a [x,X],b [y,Y]}
1.update(x,y,val);
2.update(x,Y+1,-val);
3.update(X+1,y,-val);
4.update(X+1,Y+1,val);
% 注意这种情况下 不能区间查询

*/
```

13 Segment tree

```
/*********************************
线段树是重中之重，最考验思维的东西了
但线段树的编码很容易，这里给出维护 [区间加 + 区间乘 + 区间和 MOD] 的代码 +[二维线段树]
*********************************/

// 维护 [区间加 + 区间乘 + 区间和 MOD] ----------------------------------------
int n,p;
int a[N];

LL sum[N<<2],mul[N<<2],add[N<<2];

void pushup(int rt){
    sum[rt]=(sum[rt<<1]+sum[rt<<1|1])%p;
}
```

```
void pushdown(int rt,int l,int r){
    add[rt<<1]  =(add[rt<<1]  *mul[rt]+add[rt])%p;
    add[rt<<1|1]=(add[rt<<1|1]*mul[rt]+add[rt])%p;
    mul[rt<<1]  =(mul[rt<<1]  *mul[rt])%p;
    mul[rt<<1|1]=(mul[rt<<1|1]*mul[rt])%p;
    int m = r+l >> 1;
    sum[rt<<1]  =(sum[rt<<1]  *mul[rt]+add[rt]*(m-l+1))%p;
    sum[rt<<1|1]=(sum[rt<<1|1]*mul[rt]+add[rt]*(r-m))%p;
    add[rt]=0,mul[rt]=1;
}

void build(int rt,int l,int r){
    mul[rt]=1,add[rt]=0;
    if(l==r){sum[rt]=a[l]%p; return;}
    int m = r+l >> 1;
    build(rt<<1  ,l  ,m);
    build(rt<<1|1,m+1,r);
    pushup(rt);
}

void update(int rt,int l,int r,int L,int R,int tadd,int tmul){
    if(L<=l&&r<=R){
        if(tadd!=-1){
            add[rt]=(add[rt]+tadd)%p;
            sum[rt]=(sum[rt]+(LL)tadd*(r-l+1))%p;
        }
        if(tmul!=-1){
            add[rt]=(add[rt]*tmul)%p;
            mul[rt]=(mul[rt]*tmul)%p;
            sum[rt]=(sum[rt]*tmul)%p;
        }
        return ;
    }
    pushdown(rt,l,r);
    int m = r+l >> 1;
    if(L<=m) update(rt<<1  ,l  ,m,L,R,tadd,tmul);
    if(R> m) update(rt<<1|1,m+1,r,L,R,tadd,tmul);
    pushup(rt);
    return ;
}

LL query(int rt,int l,int r,int L,int R){
    if(L<=l&&r<=R)    return sum[rt]%p;
    pushdown(rt,l,r);
    int m = r+l >> 1;LL ans = 0;
    if(L<=m) ans=(ans+query(rt<<1  ,l  ,m,L,R))%p;
    if(R> m) ans=(ans+query(rt<<1|1,m+1,r,L,R))%p;
    pushup(rt);
    return ans%p;
}

int main(){
    while(~scanf("%d%d",&n,&p)){
        for(int i=1;i<=n;i++) scanf("%d",&a[i]);
        build(1,1,n);
        int m ;scanf("%d",&m);
        int op,l,r,c;
        while(m--){
```

```
            scanf("%d",&op);
            if(op==1){//区间加
                scanf("%d%d%d",&l,&r,&c);
                update(1,1,n,l,r,-1,c);
            }
            else if(op==2){//区间乘
                scanf("%d%d%d",&l,&r,&c);
                update(1,1,n,l,r,c,-1);
            }
            else {
                scanf("%d%d",&l,&r);
                printf("%lld\n",query(1,1,n,l,r));
            }
        }
    }
    return 0;
}

// 维护 [二维最大值，最小值]----------------------------------------
//实现二 : 树套树实现

#define y0 fucky0
#define y1 fucky1
int mmin,mmax,x0,y0,x1,y1; //待查询的矩形 + 查询出来的最大最小值

int mx[N<<2][N<<2];
int mn[N<<2][N<<2];

void pushup(int rt,int xrt){
    mx[xrt][rt]=max(mx[xrt][rt<<1],mx[xrt][rt<<1|1]);
    mn[xrt][rt]=min(mn[xrt][rt<<1],mn[xrt][rt<<1|1]);
}
void buildY(int rt,int l,int r,int xrt,int x){
    if(l==r){
        if(x<0){
            mx[xrt][rt]=max(mx[xrt<<1][rt],mx[xrt<<1|1][rt]);
            mn[xrt][rt]=min(mn[xrt<<1][rt],mn[xrt<<1|1][rt]);
        }
        else {
            mx[xrt][rt]=mn[xrt][rt]=a[x][l];
        }
        return ;
    }
    int m = r+l >> 1;
    buildY(rt<<1  ,l  ,m,xrt,x);
    buildY(rt<<1|1,m+1,r,xrt,x);
    pushup(rt,xrt);
}
void buildX(int rt,int l,int r){
    if(l==r){
        buildY(1,1,n,rt,l);
        return ;
    }
    int m = r+l >> 1;
    buildX(rt<<1  ,l  ,m);
    buildX(rt<<1|1,m+1,r);
    buildY(1,1,n,rt,-1);
}
```

```
void updateY(int rt,int l,int r,int xrt,int x,int y,int v){
    if(l==r){
        if(x<0){
            mx[xrt][rt]=max(mx[xrt<<1][rt],mx[xrt<<1|1][rt]);
            mn[xrt][rt]=min(mn[xrt<<1][rt],mn[xrt<<1|1][rt]);
        }
        else {
            mx[xrt][rt]=mn[xrt][rt]=v;
        }
        return ;
    }
    int m = r+l >> 1;
    if(y<=m) updateY(rt<<1  ,l  ,m,xrt,x,y,v);
    else     updateY(rt<<1|1,m+1,r,xrt,x,y,v);
    pushup(rt,xrt);
}
void updateX(int rt,int l,int r,int x,int y,int v){
    if(l==r){
        updateY(1,1,n,rt,l,y,v);
        return ;
    }
    int m = r+l >> 1;
    if(x<=m) updateX(rt<<1  ,l  ,m,x,y,v);
    else     updateX(rt<<1|1,m+1,r,x,y,v);
    updateY(1,1,n,rt,-1,y,v);
}
void queryY(int rt,int l,int r,int xrt){
    if(y0<=l&&r<=y1){
        mmin=min(mmin,mn[xrt][rt]);
        mmax=max(mmax,mx[xrt][rt]);
        return ;
    }
    int m = r+l >> 1;
    if(y0<=m) queryY(rt<<1  ,l  ,m,xrt);
    if(y1> m) queryY(rt<<1|1,m+1,r,xrt);
}
void queryX(int rt,int l,int r){
    if(x0<=l&&r<=x1){queryY(1,1,n,rt);return;}
    int m = r+l >> 1;
    if(x0<=m) queryX(rt<<1  ,l  ,m);
    if(x1> m) queryX(rt<<1|1,m+1,r);
}


int main(){
    int _ = 1,kcase = 0;
    for(scanf("%d",&_);_--;){
        scanf("%d",&n);
        for(int i=1;i<=n;i++)for(int j=1;j<=n;j++){
            scanf("%d",&a[i][j]);
        }
        buildX(1,1,n);
        int x,y,l;
        printf("Case #%d:\n",++kcase);
        for(scanf("%d",&m);m--;){
            scanf("%d%d%d",&x,&y,&l);

            x0=max(1,x-l/2);
```

```
            y0=max(1,y-l/2);
            x1=min(n,x+l/2);
            y1=min(n,y+l/2);
            mmin=INF,mmax=-INF;

            queryX(1,1,n);
            printf("%d\n",(mmin+mmax)/2);
            updateX(1,1,n,x,y,(mmin+mmax)/2);
        }
    }
    return 0;
}

// [矩形面积并] ----------------------------------------
int n;
struct Seg {
    double l, r, h; int d;
    Seg() {}
    Seg(double l, double r, double h, int d): l(l), r(r), h(h), d(d) {}
    bool operator< (const Seg& rhs) const {return h < rhs.h;}
} a[N];

int cnt[N << 2]; //根节点维护的是 [l, r+1] 的区间
double sum[N << 2], all[N];

#define lson l, m, rt << 1
#define rson m + 1, r, rt << 1 | 1

void push_up(int l, int r, int rt) {
    if(cnt[rt]) sum[rt] = all[r + 1] - all[l];
    else if(l == r) sum[rt] = 0; //leaves have no sons
    else sum[rt] = sum[rt << 1] + sum[rt << 1 | 1];
}

void update(int L, int R, int v, int l, int r, int rt) {
    if(L <= l && r <= R) {
        cnt[rt] += v;
        push_up(l, r, rt);
        return;
    }
    int m = l + r >> 1;
    if(L <= m) update(L, R, v, lson);
    if(R > m) update(L, R, v, rson);
    push_up(l, r, rt);
}

int main() {
    int kase = 0;
    while(scanf("%d", &n) == 1 && n) {
        for(int i = 1; i <= n; ++i) {
            double x1, y1, x2, y2;
            scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
            a[i] = Seg(x1, x2, y1, 1);
            a[i + n] = Seg(x1, x2, y2, -1);
            all[i] = x1; all[i + n] = x2;
        }
        n <<= 1;
        sort(a + 1, a + 1 + n);
```

```
        sort(all + 1, all + 1 + n);
        int m = unique(all + 1, all + 1 + n) - all - 1;

        memset(cnt, 0, sizeof cnt);
        memset(sum, 0, sizeof sum);

        double ans = 0;
        for(int i = 1; i < n; ++i) {
            int l = lower_bound(all + 1, all + 1 + m, a[i].l) - all;
            int r = lower_bound(all + 1, all + 1 + m, a[i].r) - all;
            if(l < r) update(l, r - 1, a[i].d, 1, m, 1);
            ans += sum[1] * (a[i + 1].h - a[i].h);
        }
        printf("Test case #%d\nTotal explored area: %.2f\n\n", ++kase, ans);
    }
    return 0;
}

// [矩形面积交] ----------------------------------------
int n;
struct Seg {
    double l, r, h; int d;
    Seg() {}
    Seg(double l, double r, double h, double d): l(l), r(r), h(h), d(d) {}
    bool operator< (const Seg& rhs) const {
        return h < rhs.h;
    }
} a[N];

int cnt[N << 2];
double one[N << 2], two[N << 2], all[N];

#define lson l, m, rt << 1
#define rson m + 1, r, rt << 1 | 1

void push_up(int l, int r, int rt) {
    if(cnt[rt] >= 2) two[rt] = one[rt] = all[r + 1] - all[l];
    else if(cnt[rt] == 1) {
        one[rt] = all[r + 1] - all[l];
        if(l == r) two[rt] = 0;
        else two[rt] = one[rt << 1] + one[rt << 1 | 1];
    } else {
        if(l == r) one[rt] = two[rt] = 0;
        else {
            one[rt] = one[rt << 1] + one[rt << 1 | 1];
            two[rt] = two[rt << 1] + two[rt << 1 | 1];
        }
    }
}

void update(int L, int R, int v, int l, int r, int rt) {
    if(L <= l && r <= R) {
        cnt[rt] += v;
        push_up(l, r, rt);
        return;
    }
    int m = l + r >> 1;
    if(L <= m) update(L, R, v, lson);
```

```
    if(R > m) update(L, R, v, rson);
    push_up(l, r, rt);
}

int main() {
    int t; scanf("%d", &t);
    while(t--) {
        scanf("%d", &n);
        for(int i = 1; i <= n; ++i) {
            double x1, y1, x2, y2;
            scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
            a[i] = Seg(x1, x2, y1, 1);
            a[i + n] = Seg(x1, x2, y2, -1);
            all[i] = x1; all[i + n] = x2;
        }
        n <<= 1;
        sort(a + 1, a + 1 + n);
        sort(all + 1, all + 1 + n);
        int m = unique(all + 1, all + 1 + n) - all - 1;

        memset(cnt, 0, sizeof cnt);
        memset(one, 0, sizeof one);
        memset(two, 0, sizeof two);

        double ans = 0;
        for(int i = 1; i < n; ++i) {
            int l = lower_bound(all + 1, all + 1 + m, a[i].l) - all;
            int r = lower_bound(all + 1, all + 1 + m, a[i].r) - all;
            if(l < r) update(l, r - 1, a[i].d, 1, m, 1);
            ans += two[1] * (a[i + 1].h - a[i].h);
        }
        printf("%.2f\n", ans);
    }
    return 0;
}

// [矩形周长并] ----------------------------------------
int n, m[2];
int sum[N << 2], cnt[N << 2], all[2][N];
struct Seg {
    int l, r, h, d;
    Seg() {}
    Seg(int l, int r, int h, int d): l(l), r(r), h(h), d(d) {}
    bool operator< (const Seg& rhs) const {return h < rhs.h;}
} a[2][N];

#define lson l, m, rt << 1
#define rson m + 1, r, rt << 1 | 1

void push_up(int p, int l, int r, int rt) {
    if(cnt[rt]) sum[rt] = all[p][r + 1] - all[p][l];
    else if(l == r) sum[rt] = 0;
    else sum[rt] = sum[rt << 1] + sum[rt << 1 | 1];
}

void update(int p, int L, int R, int v, int l, int r, int rt) {
    if(L <= l && r <= R) {
        cnt[rt] += v;
```

```
        push_up(p, l, r, rt);
        return;
    }
    int m = l + r >> 1;
    if(L <= m) update(p, L, R, v, lson);
    if(R > m) update(p, L, R, v, rson);
    push_up(p, l, r, rt);
}

int main() {
    while(scanf("%d", &n) == 1) {
        for(int i = 1; i <= n; ++i) {
            int x1, y1, x2, y2;
            scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
            all[0][i] = x1, all[0][i + n] = x2;
            all[1][i] = y1, all[1][i + n] = y2;
            a[0][i] = Seg(x1, x2, y1, 1);
            a[0][i + n] = Seg(x1, x2, y2, -1);
            a[1][i] = Seg(y1, y2, x1, 1);
            a[1][i + n] = Seg(y1, y2, x2, -1);
        }
        n <<= 1;
        sort(all[0] + 1, all[0] + 1 + n);
        m[0] = unique(all[0] + 1, all[0] + 1 + n) - all[0] - 1;
        sort(all[1] + 1, all[1] + 1 + n);
        m[1] = unique(all[1] + 1, all[1] + 1 + n) - all[1] - 1;
        sort(a[0] + 1, a[0] + 1 + n);
        sort(a[1] + 1, a[1] + 1 + n);

//      for(int i = 0; i < 2; ++i){
//          for(int j = 1; j <= m[i]; ++j) cout << all[i][j] <<' '; cout << endl;
//      } cout << endl;

        int ans = 0;
        for(int i = 0; i < 2; ++i) {
            int t = 0, last = 0;
            memset(cnt, 0, sizeof cnt);
            memset(sum, 0, sizeof sum);
            for(int j = 1; j <= n; ++j) {
                int l = lower_bound(all[i] + 1, all[i] + 1 + m[i], a[i][j].l) - all[i];
                int r = lower_bound(all[i] + 1, all[i] + 1 + m[i], a[i][j].r) - all[i];
                if(l < r) update(i, l, r - 1, a[i][j].d, 1, m[i], 1);
                t += abs(sum[1] - last);
                last = sum[1];
            }
            ans += t;
        }
        printf("%d\n", ans);
    }
    return 0;
}

// [线段树合并] ----------------------------------------

/*****************************
是对多颗同样的线段树进行合并的操作，
合并过程对左右子树的关系进行计算.
对每个节点建立一颗线段树，进而在回溯的过程中进行合并
```

```
*****************************/

LL sum[N*10],ans,cnt0,cnt1;
int l[N],r[N],v[N],cnt ;
int ls[N*10],rs[N*10],root[N],tot;

void rd(int x){
    v[x]=read();
    if(!v[x]){
        l[x]=++cnt;rd(l[x]);
        r[x]=++cnt;rd(r[x]);
    }
}

void pushup(int rt){
    sum[rt]=sum[ls[rt]]+sum[rs[rt]];
}

void build(int &rt,int l,int r,int pos){
    if(0==rt)rt=++tot,ls[rt]=rs[rt]=0;
    if(l==r){sum[rt]=1;return;}
    int m = r+l >>1;
    if(pos<=m) build(ls[rt],l  ,m,pos);
    else       build(rs[rt],m+1,r,pos);
    pushup(rt);
}

int marge(int x,int y){
    if(!x)return y;if(!y)return x;
    cnt0+=sum[rs[x]]*sum[ls[y]]; // no change
    cnt1+=sum[ls[x]]*sum[rs[y]]; // change
    ls[x]=marge(ls[x],ls[y]);
    rs[x]=marge(rs[x],rs[y]);
    pushup(x);
    return x;
}

void solve(int x){
    if(!x) return ;
    solve(l[x]),solve(r[x]);
    if(!v[x]){
        cnt0=cnt1=0;
        root[x]=marge(root[l[x]],root[r[x]]);
        ans+=min(cnt0,cnt1);
    }
}

int n;
int main(){
    tot = cnt = 0;
    n=read();
    rd(++cnt);
    for(int i=1;i<=cnt;i++)if(v[i])
        build(root[i],1,n,v[i]);
    solve(1);
    printf("%lld",ans);
    return 0;
}
```

14 Chair tree

```
/*****************************************
主席树（函数式线段树，可持久化线段树）

主席树（函数式线段树，可持久化线段树）其实就是维护多颗线段树，
每更新一个元素，那么就根据它的上一状态新建一颗线段树，然后就是线段树的操作了，
一般来维护（区间第 K 大，区间不同元素个数（在线做法））
每次新建一颗线段树，都只是开 O(log(n)) 的节点，
然后指向前一状态的其他不需要更新的节点，这样的话大大降低了总空间复杂度

主席树的具体维护要看不同情况而定，需要怎么维护就怎么维护即可
主席树一般可以看做维护树与树的前缀和，
*****************************************/

int rt[N*20];   //表示更新当前元素所形成的不同线段树的树根，
int ls[N*20];   //当前节点的左儿子
int rs[N*20];   //当前节点的右儿子
int sum[N*20];  //主席树节点维护的值
int tot;        //节点的标号

void build(int &rt,int l,int r){ //建树 一般是先建一颗空树（即没有元素更新在其上）让之后的更新
↪  依他开始，
    rt=++tot;
    sum[rt]=0;
    if(l==r) return ;
    int m = (r+l)>>1;
    build(ls[rt],l,m);
    build(rs[rt],m+1,r);
}

void update(int &rt,int l,int r,int last,int pos){
    rt = ++tot;
    ls[rt]=ls[last];
    rs[rt]=rs[last];
    sum[rt]=sum[last]+1;
    if(l==r) return ;
    int m = (r+l)>>1;
    if(pos<=m) update(ls[rt],l,m,ls[last],pos);
    else       update(rs[rt],m+1,r,rs[last],pos);
}

int query(int ss,int tt,int l,int r,int k){
    if(l==r)return l;
    int m = (l+r)>>1;
    int cnt=sum[ls[tt]]-sum[ls[ss]];
    if(k<=cnt) return query(ls[ss],ls[tt],l,m,k);
    else       return query(rs[ss],rs[tt],m+1,r,k-cnt);
}

/*****************************************
可修改的主席树

利用树状数组来维护修改信息

*****************************************/
#include <bits/stdc++.h>
using namespace std;
```

```
typedef long long LL;
#define lson l, m
#define rson m+1, r
const int N=60005;
int a[N], Hash[N];
int T[N], L[N<<5], R[N<<5], sum[N<<5];
int S[N];
int n, m, tot;
struct node{
    int l, r, k;
    bool Q;
}op[10005];

int build(int l, int r){
    int rt=(++tot);
    sum[rt]=0;
    if(l>=r) return rt;
    int m=(l+r)>>1;
    L[rt]=build(lson);
    R[rt]=build(rson);
    return rt;
}

int update(int pre, int l, int r, int x, int val){
    int rt=(++tot);
    L[rt]=L[pre], R[rt]=R[pre], sum[rt]=sum[pre]+val;
    if(l>=r) return rt;
    int m=(l+r)>>1;
    if(x<=m)  L[rt]=update(L[pre], lson, x, val);
    else      R[rt]=update(R[pre], rson, x, val);
    return rt;
}

#define lowbit(x) (x&-x)
int use[N];
void add(int x, int pos, int val){
    for(;x<=n;x+=lowbit(x))
        S[x]=update(S[x], 1, m, pos, val);
}
int Sum(int x){
    int ret=0;
    for(;x>0;x-=lowbit(x)) ret+=sum[L[use[x]]];
    return ret;
}

int query(int u, int v, int lr, int rr, int l, int r, int k){
    if(l>=r)     return l;
    int m=(l+r)>>1;
    int tmp=Sum(v)-Sum(u)+sum[L[rr]]-sum[L[lr]];
    if(tmp>=k){
        for(int i=u;i;i-=lowbit(i)) use[i]=L[use[i]];
        for(int i=v;i;i-=lowbit(i)) use[i]=L[use[i]];
        return query(u, v, L[lr], L[rr], lson, k);
    }
    else{
        for(int i=u;i;i-=lowbit(i)) use[i]=R[use[i]];
        for(int i=v;i;i-=lowbit(i)) use[i]=R[use[i]];
        return query(u, v, R[lr], R[rr], rson, k-tmp);
```

```
    }
}

void modify(int x, int p, int d){
    for(;x<=n;x+=lowbit(x))
        S[x]=update(S[x], 1, m, p, d);
}

int main(){
    int _;
    for(scanf("%d", &_);_--;){
        int q;
        scanf("%d%d", &n, &q);
        tot=0; m=0;
        for(int i=1;i<=n;i++) {
            scanf("%d", &a[i]);
            Hash[++m]=a[i];
        }
        for(int i=0;i<q;i++) {
            char s[10];
            scanf("%s", s);
            if(s[0]=='Q') {
                scanf("%d%d%d", &op[i].l, &op[i].r, &op[i].k);
                op[i].Q=1;
            }
            else {
                scanf("%d%d", &op[i].l, &op[i].r);
                op[i].Q=0;
                Hash[++m]=op[i].r;
            }
        }
        sort(Hash+1, Hash+1+m);
        int mm=unique(Hash+1, Hash+1+m)-(Hash+1);
        m=mm;
        T[0]=build(1, m);
        for(int i=1;i<=n;i++) T[i]=update(T[i-1], 1, m, lower_bound(Hash+1, Hash+1+m,
        ↪  a[i])-Hash, 1);
        for(int i=1;i<=n;i++) S[i]=T[0];
        for(int i=0;i<q;i++)  {
            if(op[i].Q) {
                for(int j=op[i].l-1;j;j-=lowbit(j)) use[j]=S[j];
                for(int j=op[i].r  ;j;j-=lowbit(j)) use[j]=S[j];
                printf("%d\n", Hash[query(op[i].l-1, op[i].r, T[op[i].l-1], T[op[i].r], 1,
                ↪  m, op[i].k)]);
            }
            else {
                modify(op[i].l, lower_bound(Hash+1, Hash+1+m, a[op[i].l])-Hash, -1);
                modify(op[i].l, lower_bound(Hash+1, Hash+1+m,   op[i].r )-Hash,  1);
                a[op[i].l]=op[i].r;
            }
        }
    }
    return 0;
}
```

15 Splay

```
/*************SPLAY-tree**********
Splay 为伸展树，也是一颗平衡树

第一种是 利用二叉排序树性质维护一堆数的
另一种是 利用伸展树性质维护一个序列的
***/

int n,m;

int ch[N][2];  //ch[][0] lson ch[][1] rson
int f[N];      //father
int sz[N];     //size
int val[N];    //value of node_i
int cnt[N];    //counts of the node_i
int root;      //root of splay-tree
int tot;       //tot,total,is the number of node of tree

void pushup(int x){
    if(x)sz[x]=sz[ch[x][0]]+sz[ch[x][1]]+cnt[x];
}

void rotate(int x,int k){   // k = 0 左旋，  k = 1 右旋
    int y=f[x];int z=f[y];
    ch[y][!k]=ch[x][k];if(ch[x][k])f[ch[x][k]]=y;
    f[x]=z;if(z)ch[z][ch[z][1]==y]=x;
    f[y]=x;ch[x][k]=y;
    pushup(y),pushup(x);
}
// 单旋
// void splay(int x,int goal){
//     for(int y=f[x];f[x]!=goal;y=f[x])
//         rotate(x,(ch[y][0]==x));
//     if(goal==0) root=x;
// }
// 双旋
void splay(int x,int goal){//将 x 旋转到 goal 的下面
    while(f[x] != goal){
        if(f[f[x]] == goal) rotate(x , ch[f[x]][0] == x);
        else   {
            int y=f[x],z=f[y];
            int K = (ch[z][0]==y);
            if(ch[y][K] == x) rotate(x,!K),rotate(x,K);
            else rotate(y,K),rotate(x,K);
        }
    }
    pushup(x);
    if(goal==0) root=x;
}
void newnode(int rt,int v,int fa){
//    printf("newnode : rt =  %d\n",rt);
    f[rt]=fa;val[rt]=v,sz[rt]=cnt[rt]=1;
    ch[rt][0]=ch[rt][1]=0;
}

void delnode(int &rt){ //其实是为内存回收做准备的  回头再完善
    f[rt]=val[rt]=sz[rt]=cnt[rt]=0;
```

```
    ch[rt][0]=ch[rt][1]=rt=0;
}

/************************** 以下是 DEBUG**************************/

void Traversal(int rt){
    if(!rt) return;
    Traversal(ch[rt][0]);
    printf("%d f[]=%d sz[]=%d lson=%d rson=%d
    ↪  val[]=%d\n",rt,f[rt],sz[rt],ch[rt][0],ch[rt][1],val[rt]);
    Traversal(ch[rt][1]);
}
void debug(){
    printf("ROOT = %d <---\n",root);
    Traversal(root);
}

/************************** 以下是前置操作 **************************/

//以 x 为根的子树 的极值点  0 极小 1 极大
int extreme(int x,int k){
    while(ch[x][k])x=ch[x][k];splay(x,0);
    return x;
}
//以 x 为根的子树 第 k 个数的位置
int kth(int x,int k){
    if(sz[ch[x][0]]+1<=k&&k<=sz[ch[x][0]]+cnt[x]) return x;
    else if(sz[ch[x][0]]>=k) return kth(ch[x][0],k);
    else return kth(ch[x][1],k-sz[ch[x][0]]-cnt[x]);
}
int search(int rt,int x){
        if(ch[rt][0]&&val[rt]>x) return search(ch[rt][0],x);
    else if(ch[rt][1]&&val[rt]<x)return search(ch[rt][1],x);
    else return rt;
}
/************************** 以下是正经操作 **************************/
//前驱
int prec(int x){
    int k=search(root,x);
    splay(k,0);//debug();
    if(val[k]<x) return k;
    return extreme(ch[k][0],1);
}
//后继
int sufc(int x){
    int k=search(root,x);
    splay(k,0);//debug();
    if(val[k]>x) return k;
    return extreme(ch[k][1],0);
}
//返回数值 x 的排名
int rk(int x){
    int k=search(root,x);
    splay(k,0);
    return sz[ch[root][0]]+1;
}
//按照二叉排序树性质插入 x
void _insert(int x){
```

```
    int y=search(root,x),k=-1;
    if(val[y]==x){
        cnt[y]++;
        sz[y]++;
        for(int yy=y;yy;yy=f[yy]) pushup(yy);
    }
    else {
        int p=prec(x),s=sufc(x);
        splay(p,0);splay(s,p);
        newnode(++tot,x,ch[root][1]);
        ch[ch[root][1]][0]=tot;
        for(int z=ch[root][1];z;z=f[z])pushup(z);
    }
    if(k==-1) splay(y,0);else splay(tot,0);
}

void _delete(int x){
    int y=search(root,x);
    if(val[y]!=x) return;
    if(cnt[y]>1){
        cnt[y]--;
        sz[y]--;
        for(int yy=y;yy;yy=f[yy]) pushup(yy);
    }
    else if(ch[y][0]==0||ch[y][1]==0){
        int z=f[y];
        ch[z][ch[z][1]==y]=ch[y][ch[y][0]==0];
        f[ch[y][ch[y][0]==0]]=z;delnode(y);
        for(int yy=z;yy;yy=f[yy]) pushup(yy);
    }
    else {
        int p=prec(x),s=sufc(x);
        splay(p,0);splay(s,p);
        ch[ch[root][1]][0]=0;
        delnode(ch[ch[root][1]][0]);
        for(int yy=s;yy;yy=f[yy]) pushup(yy);
    }
}

int main(){
    scanf("%d",&n);
    tot=0,root=1;
    newnode(++tot,-INF,0),newnode(++tot,INF,root);
    ch[root][1]=tot;
    for(int op,x;n--;){
        op=read(),x=read();
             if(op==1) _insert(x);
        else if(op==2) _delete(x);
        else if(op==3) printf("%d\n",rk(x)-1);
        else if(op==4) printf("%d\n",val[kth(root,x+1)]);
        else if(op==5) printf("%d\n",val[prec(x)]);
        else           printf("%d\n",val[sufc(x)]);
    }
    return 0;
}
```

//

```
int n,m;

int ch[N][2];  //ch[][0] lson ch[][1] rson
int f[N];      //father
int sz[N];     //size
int val[N];    //value of node_i
int lazy[N];   //lazy-tag
int mi[N];     //min of son-tree : root of i
int rev[N];    //tag of revear
int root;      //root of splay-tree
int tot;       //tot,total,is the number of node of tree

void myswap(int &x,int &y){
    x^=y,y^=x,x^=y;
}
int min(int x,int y){
    return (x<y)?x:y;
}
void update_rev(int x){
    if(!x) return ;
    myswap(ch[x][0],ch[x][1]);
    rev[x]^=1;
}

void update_add(int x,int v){
    if(x) lazy[x]+=v,val[x]+=v,mi[x]+=v;
}

void pushdown(int x){
    if(!x) return ;
    if(lazy[x]){
        update_add(ch[x][0],lazy[x]);
        update_add(ch[x][1],lazy[x]);
        lazy[x]=0;
    }
    if(rev[x]){
        update_rev(ch[x][0]);
        update_rev(ch[x][1]);
        rev[x]=0;
    }
}

void pushup(int x){
    if(!x)return ;
    sz[x]=1,mi[x]=val[x];
    if(ch[x][0])sz[x]+=sz[ch[x][0]],mi[x]=min(mi[x],mi[ch[x][0]]);
    if(ch[x][1])sz[x]+=sz[ch[x][1]],mi[x]=min(mi[x],mi[ch[x][1]]);
}

void rotate(int x,int k){   // k = 0 左旋,  k = 1 右旋
    int y=f[x];int z=f[y];
    pushdown(y),pushdown(x);
    ch[y][!k]=ch[x][k];if(ch[x][k])f[ch[x][k]]=y;
    f[x]=z;if(z)ch[z][ch[z][1]==y]=x;
    f[y]=x;ch[x][k]=y;
    pushup(y),pushup(x);
}
// 单旋
```

```
// void splay(int x,int goal){
//     for(int y=f[x];f[x]!=goal;y=f[x])
//         rotate(x,(ch[y][0]==x));
//     if(goal==0) root=x;
// }
// 双旋
void splay(int x,int goal){//将 x 旋转到 goal 的下面
    while(f[x] != goal){
        if(f[f[x]] == goal) rotate(x , ch[f[x]][0] == x);
        else   {
            int y=f[x],z=f[y];
            int K = (ch[z][0]==y);
            if(ch[y][K] == x) rotate(x,!K),rotate(x,K);
            else rotate(y,K),rotate(x,K);
        }
    }
    pushup(x);
    if(goal==0) root=x;
}
void newnode(int rt,int v,int fa){
    f[rt]=fa;
    mi[rt]=val[rt]=v;sz[rt]=1;
    ch[rt][0]=ch[rt][1]=rev[rt]=lazy[rt]=0;
}
void delnode(int rt){
    f[rt]=mi[rt]=val[rt]=sz[rt]=0;
    ch[rt][0]=ch[rt][1]=rev[rt]=lazy[rt]=0;
}
void build(int &rt,int l,int r,int fa){
    if(l>r) return ;
    int m = r+l >> 1;
    rt=m; newnode(rt,val[rt],fa);
    build(ch[rt][0],l,m-1,rt);
    build(ch[rt][1],m+1,r,rt);
    pushup(rt);
}

void init(int n){
    root=0;
    f[0]=sz[0]=ch[0][0]=ch[0][1]=rev[0]=0;
    build(root,1,n,0);
    pushup(root);
}

/************************** 以下是 DEBUG**************************/
void Traversal(int rt){
    if(!rt) return;
    pushdown(ch[rt][0]);Traversal(ch[rt][0]);
    printf("%d f[]=%d sz[]=%d lson=%d rson=%d val[]=%d mi[]=%d
    ↪  \n",rt,f[rt],sz[rt],ch[rt][0],ch[rt][1],val[rt],mi[rt]);
    pushdown(ch[rt][1]);Traversal(ch[rt][1]);
    pushup(rt);
}
void debug(){
    printf("ROOT = %d <---\n",root);
    pushdown(root);
    Traversal(root);
}
```

```
/************************ 以下是前置操作 ************************/

//以 x 为根的子树 的最左节点
int x_left(int x){
    for(pushdown(x);ch[x][0];pushdown(x)) x=ch[x][0];
    return x;
}
//以 x 为根的子树 的最右节点
int x_right(int x){
    for(pushdown(x);ch[x][1];pushdown(x)) x=ch[x][1];
    return x;
}
//以 x 为根的子树 第 k 个数的位置
int kth(int x,int k){
    pushdown(x);
    if(sz[ch[x][0]]+1 == k) return x;
    else if(sz[ch[x][0]]>=k) return kth(ch[x][0],k);
    else return kth(ch[x][1],k-sz[ch[x][0]]-1);
}

/************************** 以下是正经操作 **************************/
/***
    如果有区间为 [1,n] 情况不好处理,
    所以我们可以 多添加一个 head, 一个 tail
    这样的话区间 [1,n] 就是 tail 的左儿子了,
*/
//区间交换
void exchange(int l1,int r1,int l2,int r2){
    int x=kth(root,l2-1),y=kth(root,r2+1);
    splay(x,0),splay(y,x);
    int tmp_right = ch[y][0]; ch[y][0]=0;
    x=kth(root,l1-1),y=kth(root,l1);
    splay(x,0),splay(y,x);
    ch[y][0] = tmp_right;
    f[tmp_right]=y;
}

//区间翻转
void reversal(int l,int r){
    int x=kth(root,l-1),y=kth(root,r+1);
    splay(x,0);splay(y,x);
    update_rev(ch[y][0]);
}

//区间加
void add(int l,int r,int v){
    int x=kth(root,l-1),y=kth(root,r+1);
    splay(x,0);splay(y,x);
    update_add(ch[y][0],v);
}

//在第 k 个数后插入值为 x 的节点
void _insert(int k,int x){
    int r=kth(root,k),rr=kth(root,k+1);
    splay(r,0),splay(rr,r);
    newnode(++tot,x,rr);ch[rr][0]=tot;
    for(r=rr;r;r=f[r])pushdown(r),pushup(r);
```

```
    splay(rr,0);
}

//删除第 k 个数
void _delete(int k){
    splay(kth(root,k-1),0);
    splay(kth(root,k+1),root);
    delnode(ch[ch[root][1]][0]);
    ch[ch[root][1]][0]=0;
    pushup(ch[root][1]);
    pushup(root);
}

//int get_max(int l,int r){
//    int x=kth(root,l-1),y=kth(root,r+1);
//    splay(x,0);splay(y,x);
//    return mx[ch[y][0]];
//}

int get_min(int l,int r){
    int x=kth(root,l-1),y=kth(root,r+1);
    splay(x,0);splay(y,x);
    return mi[ch[y][0]];
}

/*****************************************************/

char s[12];

int main(){
    scanf("%d",&n);
    val[1]=val[n+2]=1000000000;
    for(int i=1+1;i<=n+1;i++) val[i]=read();
    tot=n+2;init(n+2);
    scanf("%d",&m);
    for(int i=1,d,l,r,v;i<=m;i++){
        scanf("%s",s);
        if(s[0]=='A'){ //ADD
            scanf("%d%d%d",&l,&r,&d);
            add(l+1,r+1,d);
        }
        else if(s[0]=='I'){ //INSERT
            scanf("%d%d",&l,&d);
            _insert(l+1,d);
        }
        else if(s[0]=='M'){ //MIN
            scanf("%d%d",&l,&r);
            printf("%d\n",get_min(l+1,r+1));
        }
        else if(s[0]=='D'){ //DELETE
            scanf("%d",&l);
            _delete(l+1);
        }
        else if(s[3]=='E'){ //REVERSE
            scanf("%d%d",&l,&r);
            reversal(l+1,r+1);
        }
        else { //REVOLVE
```

```
            scanf("%d%d%d",&l,&r,&d);
            d=(d%(r-l+1)+r-l+1)%(r-l+1);
            if(d) exchange(l +1,r-d +1,r-d+1 +1,r +1);
        }
    }
    return 0;
}
```

16 Link Cut Tree

```
//动态维护一组森林，要求支持一下操作：
//link(a,b) ：如果 a,b 不在同一颗子树中，则通过在 a,b 之间连边的方式，连接这两颗子树
//cut(a,b)  ：如果 a,b 在同一颗子树中，且 a!=b，则将 a 视为这颗子树的根以后，切断 b 与其父亲结点的连接
//ADD(a,b,w)：如果 a,b 在同一颗子树中，则将 a,b 之间路径上所有点的点权增加 w
//query(a,b)：如果 a,b 在同一颗子树中，返回 a,b 之间路径上点权的最大值
const int MAXN = 300010;
int ch[MAXN][2],Fa[MAXN],key[MAXN];
int add[MAXN],rev[MAXN],Max[MAXN];
bool rt[MAXN];

void Update_Add(int r,int d){
    if(!r)return;
    key[r] += d;
    add[r] += d;
    Max[r] += d;
}
void Update_Rev(int r){
    if(!r)return;
    swap(ch[r][0],ch[r][1]);
    rev[r] ^= 1;
}
void push_down(int r){
    if(add[r]){
        Update_Add(ch[r][0],add[r]);
        Update_Add(ch[r][1],add[r]);
        add[r] = 0;
    }
    if(rev[r]){
        Update_Rev(ch[r][0]);
        Update_Rev(ch[r][1]);
        rev[r] = 0;
    }
}
void push_up(int r){
    Max[r] = max(max(Max[ch[r][0]],Max[ch[r][1]]),key[r]);
}
void Rotate(int x){
    int y = Fa[x], kind = ch[y][1]==x;
    ch[y][kind] = ch[x][!kind];
    Fa[ch[y][kind]] = y;
    Fa[x] = Fa[y];
    Fa[y] = x;
    ch[x][!kind] = y;
    if(rt[y]) rt[y] = false, rt[x] = true;
    else ch[Fa[x]][ch[Fa[x]][1]==y] = x;
    push_up(y);
}
```

```
//P 函数先将根结点到 r 的路径上所有的结点的标记逐级下放
void P(int r){
    if(!rt[r])P(Fa[r]);
    push_down(r);
}
void Splay(int r){
    P(r);
    while( !rt[r] ){
        int f = Fa[r], ff = Fa[f];
        if(!rt[f]){
            if( (ch[ff][1]==f)==(ch[f][1]==r) ) Rotate(f);
            else Rotate(r);
        }
        Rotate(r);
    }
    push_up(r);
}
int Access(int x){
    int y = 0;
    for(;x;x=Fa[y=x]){
        Splay(x);
        rt[ch[x][1]] = true, rt[ch[x][1]=y] = false;
        push_up(x);
    }
    return y;
}
//判断是否是同根 (真实的树, 非 splay)
bool judge(int u,int v){
    while(Fa[u]) u = Fa[u];
    while(Fa[v]) v = Fa[v];
    return u == v;
}
//使 r 成为它所在的树的根
void mroot(int r){
    Access(r);
    Splay(r);
    Update_Rev(r);
}
//调用后 u 是原来 u 和 v 的 lca,v 和 ch[u][1] 分别存着 lca 的 2 个儿子
//(原来 u 和 v 所在的 2 颗子树)
void lca(int &u,int &v){
    Access(v), v = 0;
    while(u){
        Splay(u);
        if(!Fa[u])return;
        rt[ch[u][1]] = true;
        rt[ch[u][1]=v] = false;
        push_up(u);
        u = Fa[v = u];
    }
}
void link(int u,int v){
    if(judge(u,v)){
        puts("-1");
        return;
    }
    mroot(u);
    Fa[u] = v;
```

```
}
//使 u 成为 u 所在树的根，并且 v 和它父亲的边断开
void cut(int u,int v){
    if(u == v || !judge(u,v)){
        puts("-1");
        return;
    }
    mroot(u);
    Splay(v);
    Fa[ch[v][0]] = Fa[v];
    Fa[v] = 0;
    rt[ch[v][0]] = true;
    ch[v][0] = 0;
    push_up(v);
}
void ADD(int u,int v,int w){
    if(!judge(u,v)){
        puts("-1");
        return;
    }
    lca(u,v);
    Update_Add(ch[u][1],w);
    Update_Add(v,w);
    key[u] += w;
    push_up(u);
}
void query(int u,int v){
    if(!judge(u,v)){
        puts("-1");
        return;
    }
    lca(u,v);
    printf("%d\n",max(max(Max[v],Max[ch[u][1]]),key[u]));
}

struct Edge{
    int to,next;
}edge[MAXN*2];
int head[MAXN],tot;
void addedge(int u,int v){
    edge[tot].to = v;edge[tot].next = head[u];head[u] = tot++;
}
void dfs(int u){
    for(int i = head[u];i != -1; i = edge[i].next){
        int v = edge[i].to;
        if(Fa[v] != 0)continue;
        Fa[v] = u;
        dfs(v);
    }
}

int main(){
    int n,q,u,v;
    while(scanf("%d",&n) == 1){
        tot = 0;
        for(int i = 0;i <= n;i++){
            head[i] = -1;
            Fa[i] = 0;
```

```
            ch[i][0] = ch[i][1] = 0;
            rev[i] = 0;
            add[i] = 0;
            rt[i] = true;
        }
        Max[0] = -2000000000;
        for(int i = 1;i < n;i++){
            scanf("%d%d",&u,&v);
            addedge(u,v);
            addedge(v,u);
        }
        for(int i = 1;i <= n;i++){
            scanf("%d",&key[i]); Max[i] = key[i];
        }
        Fa[1]=-1;
        dfs(1);
        Fa[1]=0;
        scanf("%d",&q);
        for(int x,y,w,op;q--;){
            scanf("%d",&op);
            if(op == 1){
                scanf("%d%d",&x,&y);
                link(x,y);
            }
            else if(op == 2){
                scanf("%d%d",&x,&y);
                cut(x,y);
            }
            else if(op == 3) {
                scanf("%d%d%d",&w,&x,&y);
                ADD(x,y,w);
            } else {
                scanf("%d%d",&x,&y);
                query(x,y);
            }
        }
        puts("");
    }
    return 0;
}
```

17 KD-tree

```
struct KDtree{
    int ans,cnt,root,X,Y;
    struct tree{
        int d[2],mn[2],mx[2];
        int l,r;
        int newnode(int x,int y){
            d[0]=x,d[1]=y;
            mx[0]=mn[0]=x;
            mx[1]=mn[1]=y;
        }
    }T[N+M];

    static bool cmp0(const KDtree::tree &a,const KDtree::tree &b){
        return a.d[0]<b.d[0]||a.d[0]==b.d[0]&&a.d[1]<b.d[1];
    }
    static bool cmp1(const KDtree::tree &a,const KDtree::tree &b){
        return a.d[1]<b.d[1]||a.d[1]==b.d[1]&&a.d[0]<b.d[0];
    }

    void init(){cnt = 0;}

    int newnode(int x,int y){T[++cnt].newnode(x,y);}

    void up(int p,int q){
        if(0==q) return ;
        T[p].mn[0]=min(T[p].mn[0],T[q].mn[0]);
        T[p].mn[1]=min(T[p].mn[1],T[q].mn[1]);
        T[p].mx[0]=max(T[p].mx[0],T[q].mx[0]);
        T[p].mx[1]=max(T[p].mx[1],T[q].mx[1]);
    }

    int bd(int l,int r,int D){
        int m = r+l >> 1;
        if(D)nth_element(T+l+1,T+m+1,T+r+1,cmp1);
        else nth_element(T+l+1,T+m+1,T+r+1,cmp0);
        T[m].l = (l==m)?0:bd(l,m-1,1-D);
        T[m].r = (r==m)?0:bd(m+1,r,1-D);
        up(m,T[m].l); up(m,T[m].r);
        return m;
    }

    int build(){root = bd(1,cnt,0);}

    void insert(int k){
        int p = root,D = 0;
        for(;1;D=1-D){
            up(p,k);
            if(T[k].d[D]<=T[p].d[D]){
                if(0==T[p].l){T[p].l=k;return;}
                p=T[p].l;
            }
            else {
                if(0==T[p].r){T[p].r=k;return;}
                p=T[p].r;
            }
        }
```

```
    }

    int dis(int p,int x,int y){// 点 (x,y) 与 p 所在子树的范围的距离
        int res = 0;
        if(x > T[p].mx[0]) res += x - T[p].mx[0];
        if(x < T[p].mn[0]) res += T[p].mn[0] - x;
        if(y > T[p].mx[1]) res += y - T[p].mx[1];
        if(y < T[p].mn[1]) res += T[p].mn[1] - y;
        return res;
    }

    void ask(int p){
        int d0 = abs(X-T[p].d[0]) + abs(Y-T[p].d[1]);
        if(d0<ans) ans = d0;
        int dl=(T[p].l)?dis(T[p].l,X,Y):INF;
        int dr=(T[p].r)?dis(T[p].r,X,Y):INF;
        if(dl<dr){
            if(dl<ans) ask(T[p].l);
            if(dr<ans) ask(T[p].r);
        }
        else {
            if(dr<ans) ask(T[p].r);
            if(dl<ans) ask(T[p].l);
        }
    }

    int query(int x,int y){
        ans = INF;
        X = x,Y = y;
        ask(root);
        return ans;
    }
}kd;
```

18 tranform : tree -> arrary

18.1 dfs order

```
/*************************************************************
dfs 序

其实就是从根节点进行搜索，
然后向下 dfs 遍历树，依次进行编号，
同时能保证子树的编号一定大于父节点的编号，

同时借用两个数组，L[_],R[_]
分别表示这个节点 u 的子树的节点编号在 (L[u],R[u])，开区间内。

这样在进行对子树进行的操作的时候可以借助数据结构对区间进行查找，
*************************************************************/
vector<int >G[N];
int cnt = 0;
void dfs(int u,int f){
    L[u]=cnt++;
    for(int i=0;i<G[u].size();i++)if(G[u][i]!=f)
        dfs(G[u][i]);
    R[u]=cnt;
}
```

18.2 Heavy light Decomposition

```
/*********************************************************
树链剖分

树链剖分是一种将树形结构转化为线性结构的算法
通过两次树的遍历，将树剖分成一个个的 [重链],
且对每个节点进行编号，确保一条链上的节点编号连续
这样一来，我们就能通过一个维护区间关系的数据结构来维护树上，属同一个链上的元素

在维护两个节点 (u,v) 的时候即：维护两个节点 (u,v) 间的元素，
我么从深度大的不断向上维护，最后遍历的位置，两个节点一定在一条链上（且深度小的就是 LCA(u,v)）
*********************************************************/
int dep[N];   //每个节点的深度
int fa[N];    //每个节点的父节点
int sz[N];    //每个节点所有的子节点个数（包括自身）
int son[N];   //每个节点的重儿子
void dfs1(int u,int ff,int deep){
    son[u]=0;fa[u]=ff;sz[u]=1;dep[u]=deep;
    for(int i=head[u];i!=-1;i=G[i].next){
        int v=G[i].to;
        if(v==ff) continue;
        dfs1(v,u,deep+1);
        sz[u]+=sz[v];
        if(sz[v]>sz[son[u]]) son[u]=v; //重儿子子节点个数大
    }
}

int top[N];   //节点所在链上的【根】
int tree[N];  //节点对应在线段树/树状数组的位置
int pre[N];   //在线段树/树状数组的位置对应的节点的标号  (树状数组时一般不需要)
int cnt;      //对链上节点编号
void dfs2(int u,int ff){
    tree[u]=++cnt;pre[tree[u]]=u;top[u]=ff;
    if(son[u]) dfs2(son[u],ff); //先遍历重链
    else return ;
    for(int i=head[u];i!=-1;i=G[i].next){
        int v=G[i].to;
        if(v!=fa[u]&&v!=son[u]) dfs2(v,v);
    }
}

int findi(int x,int y){
    int fx=top[x],fy=top[y];
    int ans = 0;
    while(fx!=fy){
        if(dep[fx]<dep[fy]) myswap(x,y),myswap(fx,fy);
        ans+=getSum(tree[x])-getSum(tree[fx]-1);  //不断向上维护区间
        x=fa[fx],fx=top[x];
    }
    if(dep[x]>dep[y]) myswap(x,y);
    if(x!=y) ans+=getSum(tree[y])-getSum(tree[x]);
    return ans ;
}
```

19 SparseTable

```
// 一维 ST 表
int dp[N][20],mm[N];
void initrmq(int n,int b[]){
    mm[0]=-1;
    for(int i=1;i<=n;i++){
        //mm[i]=((i&(i-1))==0)?mm[i-1]+1:mm[i-1];
        mm[i]=mm[i>>1]+1;
        dp[i][0]=b[i];
    }
    for(int j=1;j<=mm[n];j++)
        for(int i=1;i+(1<<j)-1<=n;i++)
        dp[i][j]=min(dp[i][j-1],dp[i+(1<<(j-1))][j-1]);
}
int rmq(int x,int y){
    int k=mm[y-x+1];
    return min(dp[x][k],dp[y-(1<<k)+1][k]);
}

// 二维 ST 表
int mm[255];
int dpmin[255][255][8][8];//最小值
int dpmax[255][255][8][8];//最大值
void initRMQ(int n,int m){
    mm[0] = -1;
    for(int i = 1;i <= 500;i++)
        //mm[i] = ((i&(i-1))==0)?mm[i-1]+1:mm[i-1];
        mm[i]=mm[i>>1]+1;
    for(int i = 1;i <= n;i++)
        for(int j = 1;j <= m;j++)
            dpmin[i][j][0][0] = dpmax[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){
                        dpmin[i][j][ii][jj] =
                        ↪ min(dpmin[i][j][ii-1][jj],dpmin[i+(1<<(ii-1))][j][ii-1][jj]);
                        dpmax[i][j][ii][jj] =
                        ↪ max(dpmax[i][j][ii-1][jj],dpmax[i+(1<<(ii-1))][j][ii-1][jj]);
                    } else {
                        dpmin[i][j][ii][jj] =
                        ↪ min(dpmin[i][j][ii][jj-1],dpmin[i][j+(1<<(jj-1))][ii][jj-1]);
                        dpmax[i][j][ii][jj] =
                        ↪ max(dpmax[i][j][ii][jj-1],dpmax[i][j+(1<<(jj-1))][ii][jj-1]);
                    }
                }
}
//查询矩形的最大值
int rmq1(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]));
}
```

```
//查询矩形的最小值
int rmq2(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]));
}
```

20 Leftist Tree

```
struct LT{
    const static int maxn=N;
    int tot=0,v[maxn],l[maxn],r[maxn],d[maxn],fa[maxn];
    int merge(int x,int y){ //合并两个堆 然后返回写的堆顶
        if(!x) return fa[x]=y;
        if(!y) return fa[y]=x;
        if(v[x]<v[y])swap(x,y);

        r[x]=merge(r[x],y);
        fa[r[x]]=x;

        if(d[l[x]]<d[r[x]]) swap(l[x],r[x]);

        d[x]=d[r[x]]+1;
        return x;
    }
    int init(int x){//初始化一个权值为 x 的点
        tot++;
        v[tot]=x;
        l[tot]=r[tot]=d[tot]=0;
        fa[tot]=-1;
        return tot;
    }
    int insert(int x,int y){//往第 x 个左偏树插入一个权值为 y 的点
        return merge(x,init(y));
    }
    int pop(int x){ //将堆顶 pop 出去 返回写的根
        int root=merge(l[x],r[x]);
        fa[root]=-1;
        l[x]=r[x]=0;
        return root;
    }
    int root(int x){
        while(fa[x]!=-1) x=fa[x];
        return x;
    }
    int top(int x){return v[root(x)];}
}lt;
```

Part IV
Math

21 Number Theory

21.1 Basement

费马小定理: $a^{p-1} \equiv 1 \pmod{p}$
欧拉定理: $a^{\varphi(m)} \equiv 1 \pmod{m}$ $\varphi()$ 为欧拉函数
殴拉降幂: $A^x \% C = A^{x\%\varphi(C)+\varphi(C)} \% C, (x \geq \varphi(C))$
威尔逊定理: $(p-1)! \equiv -1 \pmod{p}$
素数距离: 任意两个相邻素数距离不超过 246 (截止 2014 年 2 月)
唯一分解定理: 对于任意一个 N 我们可以写成 $N = P_1^{a_1} * P_2^{a_2} * P_3^{a_3} * ... * P_n^{a_n}$

21.1.1 inverse

```
1.exgcd
void exgcd(int a,int b,int &d,int &x,int &y){
    if(b==0) {
        x = 1,y = 0,d=a;
        return ;
    }
    exgcd(b,a%b,d,x,y);
    int t = x;
    x = y;
    y = t - ( a / b ) * y;
    return ;
}
int inv(int a){
    int x,y,d;
    exgcd(a,MOD,d,x,y);
    if(d==1)  return (x%MOD+MOD)%MOD;   //返回最小正整数解
    return -1;  //不存在逆元
}
2.Femat
int qmod(int a,int b){
    int res = 1;
    for(;b;b>>=1,a=a*a%MOD)
        if(b&1) res=res*a%MOD;
    return res;
}
int inv(int a){
    return qmod(a,MOD-2);
}
```

21.1.2 Guass

```
// 高斯消元（初等行列变换）

const int MAXN=50;
int a[MAXN][MAXN];//增广矩阵
int x[MAXN];//解集
bool free_x[MAXN];//标记是否是不确定的变元

void Debug(void){
    for (int i = 0,j; i < equ; i++){
        for (j = 0; j < var + 1; j++)
            cout << a[i][j] << " ";
```

```
        cout << endl;
    }cout << endl;
}

inline int lcm(int a,int b){
    return a/__gcd(a,b)*b;//先除后乘防溢出
}

// 高斯消元法解方程组 (Gauss-Jordan elimination).(-2 表示有浮点数解，但无整数解，
//-1 表示无解，0 表示唯一解，大于 0 表示无穷解，并返回自由变元的个数)
//有 equ 个方程，var 个变元。增广矩阵行数为 equ, 分别为 0 到 equ-1, 列数为 var+1, 分别为 0 到
↪ var.
int Gauss(int equ,int var){
    int i,j,k;
    int max_r;// 当前这列绝对值最大的行.
    int col;//当前处理的列
    int ta,tb;
    int LCM;
    int temp;
    int free_x_num;
    int free_index;

    for(int i=0;i<=var;i++){
        x[i]=0;
        free_x[i]=true;
    }

    //转换为阶梯阵.
    col=0; // 当前处理的列
    for(k = 0;k < equ && col < var;k++,col++)
    {// 枚举当前处理的行.
// 找到该 col 列元素绝对值最大的那行与第 k 行交换.(为了在除法时减小误差)
        max_r=k;
        for(i=k+1;i<equ;i++){
            if(abs(a[i][col])>abs(a[max_r][col])) max_r=i;
        }
        if(max_r!=k){// 与第 k 行交换.
            for(j=k;j<var+1;j++) swap(a[k][j],a[max_r][j]);
        }
        if(a[k][col]==0){// 说明该 col 列第 k 行以下全是 0 了，则处理当前行的下一列.
            k--; continue;
        }
        for(i=k+1;i<equ;i++){// 枚举要删去的行.
            if(a[i][col]!=0){
                LCM = lcm(abs(a[i][col]),abs(a[k][col]));
                ta = LCM/abs(a[i][col]);
                tb = LCM/abs(a[k][col]);
                if(a[i][col]*a[k][col]<0)tb=-tb;//异号的情况是相加
                for(j=col;j<var+1;j++){
                    a[i][j] = a[i][j]*ta-a[k][j]*tb;
                }
            }
        }
    }

  // Debug();

    // 1. 无解的情况: 化简的增广阵中存在 (0, 0, ..., a) 这样的行 (a != 0).
```

```
    for (i = k; i < equ; i++)
    { // 对于无穷解来说，如果要判断哪些是自由变元，那么初等行变换中的交换就会影响，则要记录交换.
        if (a[i][col] != 0) return -1;
    }
    // 2. 无穷解的情况: 在 var * (var + 1) 的增广阵中出现 (0, 0, ..., 0) 这样的行，即说明没有
    ↪  形成严格的上三角阵.
    // 且出现的行数即为自由变元的个数.
    if (k < var){
        // 首先，自由变元有 var - k 个，即不确定的变元至少有 var - k 个.
        for (i = k - 1; i >= 0; i--){
            // 第 i 行一定不会是 (0, 0, ..., 0) 的情况，因为这样的行是在第 k 行到第 equ 行.
            // 同样，第 i 行一定不会是 (0, 0, ..., a), a != 0 的情况，这样的无解的.
            free_x_num = 0; // 用于判断该行中的不确定的变元的个数，如果超过 1 个，则无法求解，
            ↪  它们仍然为不确定的变元.
            for (j = 0; j < var; j++){
                if (a[i][j] != 0 && free_x[j]) free_x_num++, free_index = j;
            }
            if (free_x_num > 1) continue; // 无法求解出确定的变元.
            // 说明就只有一个不确定的变元 free_index，那么可以求解出该变元，且该变元是确定的.
            temp = a[i][var];
            for (j = 0; j < var; j++){
                if (a[i][j] != 0 && j != free_index) temp -= a[i][j] * x[j];
            }
            x[free_index] = temp / a[i][free_index]; // 求出该变元.
            free_x[free_index] = 0; // 该变元是确定的.
        }
        return var - k; // 自由变元有 var - k 个.
    }
    // 3. 唯一解的情况: 在 var * (var + 1) 的增广阵中形成严格的上三角阵.
    // 计算出 Xn-1, Xn-2 ... X0.
    for (i = var - 1; i >= 0; i--){
        temp = a[i][var];
        for (j = i + 1; j < var; j++){
            if (a[i][j] != 0) temp -= a[i][j] * x[j];
        }
        if (temp % a[i][i] != 0) return -2; // 说明有浮点数解，但无整数解.
        x[i] = temp / a[i][i];
    }
    return 0;
}
int main(void){
    int i, j;
    int equ,var;
    while (scanf("%d %d", &equ, &var) != EOF){
        memset(a, 0, sizeof(a));
        for (i = 0; i < equ; i++)
            for (j = 0; j < var + 1; j++)
                scanf("%d", &a[i][j]);
//        Debug();
        int free_num = Gauss(equ,var);
        if (free_num == -1) printf(" 无解!\n");
        else if (free_num == -2) printf(" 有浮点数解，无整数解!\n");
        else if (free_num > 0){
            printf(" 无穷多解! 自由变元个数为%d\n", free_num);
            for (i = 0; i < var; i++){
                if (free_x[i]) printf("x%d 是不确定的\n", i + 1);
                else printf("x%d: %d\n", i + 1, x[i]);
            }
```

```
        } else {
            for (i = 0; i < var; i++)
                printf("x%d: %d\n", i + 1, x[i]);
        }puts("");
    }
    return 0;
}


// 异或消元

// g[][] 为 01 矩阵
// 求解自由元个数

LL  g[333][333];
int Gauss(int n){
    int i, j, r, c, cnt;
    for (c = cnt = 0; c < n; c++){
        for (r = cnt; r < weishu; r++)if (g[r][c])break;
        if (r < weishu){
            if (r != cnt){
                for (i = 0; i < n; i++)  swap(g[r][i],g[cnt][i]);
            }
            for (i = cnt + 1; i < weishu; i++)if(g[i][c]){
                for(j = 0; j < n; j++)
                    g[i][j] ^= g[cnt][j];
            }
            cnt++;
        }
    }
    return n - cnt;
}
```

21.1.3 linear basis

```
/**************************************
线性基

基: 在线性代数中，基（也称为基底）是描述、刻画向量空间
的基本工具。向量空间的基是它的一个特殊的子集，基的元素
称为基向量。向量空间中任意一个元素，都可以唯一地表示成
基向量的线性组合。如果基中元素个数有限，就称向量空间为
有限维向量空间，将元素的个数称作向量空间的维数。
同样的，线性基是一种特殊的基，它通常会在异或运算中出现，
它的意义是：通过原集合 S 的某一个最小子集 S1 使得 S1 内元素相
互异或得到的值域与原集合 S 相互异或得到的值域相同。

性质
1. 线性基能相互异或得到原集合的所有相互异或得到的值。
2. 线性基是满足性质 1 的最小的集合
3. 线性基没有异或和为 0 的子集。
**************************************/
void Guass(){
    for (int i=1;i<=sz;i++)
        for (int j=63;j>=0;j--)
            if ((A[i]>>j)&1){
                if (!P[j]) {P[j]=A[i]; break;}
                else A[i]^=P[j];
            }
```

```
    for (int j=0;j<=63;j++) if (P[j]) r++;
}
```

r 为极大无关组的大小

21.2 Congruence problem

余数定理:

计算 $\left(\frac{a}{b}\right) \pmod c$, 其中 b 能整除 a
如果 b 与 c 互素, 则 $(\frac{a}{b})\%c = a * b^{phi(c)-1}\%c$
如果 b 与 c 不互素, 则 $(\frac{a}{b})\%c = \frac{a\%(b*c)}{b}$
对于 b 与 c 互素和不互素都有 $(\frac{a}{b})\%c = \frac{a\%(b*c)}{b}$ 成立

21.2.1 gcd

```
LL exgcd_euclid(LL a,LL b,LL &x,LL &y){
    if(b==0){
        x=1,y=0;
        return a;
    }
    LL r=exgcd_euclid(b,a%b,x,y);
    LL t=x;x=y;y=t-a/b*y;
    return r;
}

LL exgcd(LL m,LL &x,LL n,LL &y){
    LL x1,x0,y1,y0;
    x0=1,y0=0;
    x1=0,y1=1;
    LL r=(m%n+n)%n;
    LL q=(m-r)/n;
    x=0,y=1;
    while(r){
        x=x0-q*x1,y=y0-q*y1,x0=x1,y0=y1;
        x1=x,y1=y;
        m=n,n=r,r=m%n;
        q=(m-r)/n;
    }
    return n;
}
```

21.2.2 China remainder theory

```
LL CRT(LL a[],LL m[],LL len){ //x%m[i]=a[i],m[] 必须满足两两互质
    LL i,x,y,M,n=1,ret=0;
    for(i=0; i<len; ++i) n*=m[i];
    for(i=0; i<len; ++i){
        M=n/m[i];
        exgcd(M,m[i],x,y);
        ret=(ret+qmod(qmod(x,M,n),a[i],n))%n;
    }
    return (ret+n)%n;
}
```

21.2.3 Function Of Congruence

```
// X mod a[i] = b[i]
// 求 (0,m] 之间满足的 X 的个数

LL exgcd(LL a,LL b,LL &x,LL &y){
    if(!b){ x=1,y=0; return a; }
    else{
        int r = exgcd(b,a%b,x,y);
        int t = x;x = y;
        y = t - (a/b)*y;
        return r;
    }
}
LL a[11],b[11];

LL lcm(int a,int b){
    return a/__gcd(a,b)*b;
}

int main(){
    int _;
    for(scanf("%d",&_);_--;){
        int n,m;
        scanf("%d%d",&m,&n);
        for(int i=0;i<n;i++) scanf("%I64d",&a[i]);
        for(int i=0;i<n;i++) scanf("%I64d",&b[i]);

        bool flag = 1;
        LL x,y,r,t,m0=1;
        int a1,b1,c1;
        for(int i=0;i<n;i++) m0=lcm(m0,a[i]);

        for(int i=1;i<n&&flag;i++){
            a1 = a[0],b1 = a[i],c1=b[i]-b[0];
            r = exgcd(a1,b1,x,y);

            if(c1%r!=0) flag = 0; //当前方程 没有解

            t = b1/r;
            x=(x*(c1/r)%t+t)%t;
            b[0]=a[0]*x+b[0];
            a[0]=a[0]*(a[i]/r);
        }
        // 最后的 b[0] 是最终的解

        int sum = 0;
        b[0]%=m0 ;

        if(b[0]<=m)         sum = 1+(m-b[0])/m0;//sum 求的是 (0,m] 之间解的个数
        if(sum&&b[0]==0)    sum--;  //要求正整数时 不能有 0

        if(!flag)     puts("0");
        else printf("%d\n",sum);

    }
    return 0;
}
```

21.3 Prime

21.3.1 Sieve

```
// 筛法求素数

// 埃拉托斯特尼筛法
void Prime(){
    memset(Or,0,sizeof(Or));
    for(int i=2;i<N;i++){
        if(Or[i]==0){
            p[++p[0]]=i;
            for(int j=i+i;j<N;j+=i)
                Or[j]=1;
        }
    }
}

// 线性筛
int prime[20000],kp=0;
int Is_or[65536];
void Prime(){
    int n =65536; //2~n 之间的素数
    kp=0;
    memset(Is_or,1,sizeof(Is_or));
    Is_or[0]=Is_or[1]=0;
    for(int i=2;i<n;i++){
        if(Is_or[i])    prime[kp++]=i;
        for(int j=0;j<kp&&i*prime[j]<n;j++){
            Is_or[i*prime[j]]=0;
            if(i%prime[j]==0) break;
        }
    }
    return ;
}

// 区间筛 hdu 6069
// 给你 1<=l<=r<=1e12,r-l<=1e6 , 问 [l,r] 区间有多少个素数
void Prime(){
    kp = 0;
    memset(Is,true,sizeof(Is));
    for(int i=2;i<N;i++){
        if(Is[i]){
            prime[kp++]=i;
            for(int j=i+i;j<N;j+=i) Is[j]=0;
        }
    }
//    printf("kp = %d\n",kp);
}
void Prime2(LL l,LL r){
    memset(Is,true,sizeof(Is));
    for(LL i=0,b;i<kp;i++){
        b=l/prime[i];
        while(b*prime[i]<l||b<=1) b++;
        for(LL j=b*prime[i];j<=r;j+=prime[i])if(j>=1){
            Is[j-l]=false
        }
    }
    if(l==1) Is[0]=false;
```

```
    return ;
}
```

21.3.2 Fundamental Theorem of Arithmetic

```
// 复杂度有时候会很坏 分解一个数组的时候要慎重
int prime[N],kp;
int Is_or[N][2];
void Prime(){
    kp = 0;
    memset(Is_or,true,sizeof(Is_or));
    Is_or[0][0]=Is_or[1][0]=0;
    for(int i=2;i<=100000;i++){
        if(Is_or[i][0]) Is_or[i][1]=kp,prime[kp++]=i;//记录其为第几个素数
        for(int j=0;j<kp&&prime[j]*i<=100000;j++){
            Is_or[prime[j]*i][0]=0;
            if(0==i%prime[j]) break;
        }
    }
    return ;
}
int main(){
    int tem;
    cin>>tem;
    for(int j=0;j<kp&&tem>=prime[j];j++){
        if(Is_or[tem][0]) {a[Is_or[tem][1]]++;break;}
        //if(0==tem%prime[j]) ;
        while(0==tem%prime[j]) a[j]++,tem/=prime[j];
    }
}
```

21.3.3 Miller Rabbin

```
//************************************************************
// Miller_Rabin 算法进行素数测试
//速度快，而且可以判断 <2^63 的数
//复杂度 O(slog^3n)
//************************************************************
const int S=20;//随机算法判定次数，S 越大，判错概率越小
//计算 (a*b)%c.   a,b 都是 long long 的数，直接相乘可能溢出的
//  a,b,c <2^63
long long mult_mod(long long a,long long b,long long c){
    a%=c;b%=c;
    long long ret=0;
    while(b){
        if(b&1){ret+=a;ret%=c;}
        a<<=1;
        if(a>=c)a%=c;
        b>>=1;
    }
    return ret;
}
//计算  x^n %c
long long pow_mod(long long x,long long n,long long mod)//x^n%c{
    if(n==1)return x%mod;
    x%=mod;
    long long tmp=x;
    long long ret=1;
    while(n){
```

```
        if(n&1) ret=mult_mod(ret,tmp,mod);
        tmp=mult_mod(tmp,tmp,mod);
        n>>=1;
    }
    return ret;
}

//以 a 为基,n-1=x*2^t      a^(n-1)=1(mod n)  验证 n 是不是合数
//一定是合数返回 true, 不一定返回 false
bool check(long long a,long long n,long long x,long long t){
    long long ret=pow_mod(a,x,n);
    long long last=ret;
    for(int i=1;i<=t;i++){
        ret=mult_mod(ret,ret,n);
        if(ret==1&&last!=1&&last!=n-1) return true;//合数
        last=ret;
    }
    if(ret!=1) return true;
    return false;
}

// Miller_Rabin() 算法素数判定
//是素数返回 true.(可能是伪素数, 但概率极小)
//合数返回 false;

bool Miller_Rabin(long long n){
    if(n<2)return false;
    if(n==2)return true;
    if((n&1)==0) return false;//偶数
    long long x=n-1;
    long long t=0;
    while((x&1)==0){x>>=1;t++;}
    for(int i=0;i<S;i++){
        long long a=rand()%(n-1)+1;//rand() 需要 stdlib.h 头文件
        if(check(a,n,x,t))
            return false;//合数
    }
    return true;
}
```

21.3.4 Pollard Rho

```
//*************************************************
//pollard_rho 算法进行质因数分解
//复杂度 O(n^{1/4})
//*************************************************
long long factor[100];//质因数分解结果 (刚返回时是无序的)
int tol;//质因数的个数。数组小标从 0 开始
long long gcd(long long a,long long b){
    if(a==0)return 1;//???????
    if(a<0) return gcd(-a,b);
    while(b){
        long long t=a%b;
        a=b;
        b=t;
    }
    return a;
}
long long Pollard_rho(long long x,long long c){
```

```
    long long i=1,k=2;
    long long x0=rand()%x;
    long long y=x0;
    while(1){
        i++;
        x0=(mult_mod(x0,x0,x)+c)%x;
        long long d=gcd(y-x0,x);
        if(d!=1&&d!=x) return d;
        if(y==x0) return x;
        if(i==k){y=x0;k+=k;}
    }
}
//对 n 进行素因子分解
void findfac(long long n){
    if(Miller_Rabin(n)){ //素数
        factor[tol++]=n; //值得注意的是 这里的 factor 并不是有序的!!!!!
        return;
    }
    long long p=n;
    while(p>=n)p=Pollard_rho(p,rand()%(n-1)+1);
    findfac(p);
    findfac(n/p);
}
```

21.3.5 count primes

```
// 代码一:
// 复杂度大概 O(n^(3/4))

#include <bits/stdc++.h>
#define ll long long
using namespace std;
ll f[340000],g[340000],n;
void init(){
    ll i,j,m;
    for(m=1;m*m<=n;++m)f[m]=n/m-1;
    for(i=1;i<=m;++i)g[i]=i-1;
    for(i=2;i<=m;++i){
        if(g[i]==g[i-1])continue;
        for(j=1;j<=min(m-1,n/i/i);++j){
            if(i*j<m)f[j]-=f[i*j]-g[i-1];
            else f[j]-=g[n/i/j]-g[i-1];
        }
        for(j=m;j>=i*i;--j)g[j]-=g[j/i]-g[i-1];
    }
}
int main(){
    while(scanf("%I64d",&n)!=EOF){
        init();
        cout<<f[1]<<endl;
    }
    return 0;
}

// 代码二:
// 复杂度大概 O(n^(2/3))

#include<cstdio>
#include<cmath>
```

```cpp
using namespace std;
#define LL long long
const int N = 5e6 + 2;
bool np[N];
int prime[N], pi[N];
int getprime()  {
    int cnt = 0;
    np[0] = np[1] = true;
    pi[0] = pi[1] = 0;
    for(int i = 2; i < N; ++i)  {
        if(!np[i]) prime[++cnt] = i;
        pi[i] = cnt;
        for(int j = 1; j <= cnt && i * prime[j] < N; ++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 < N)  {
        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 < N)   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 < N)   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;
}

//照素数统计只多了这么个部分..
// 4 因子个数统计
LL getans(LL x){ // x < 1e11
    LL ans = pi[sqrt3(x)];
    for(int i=1,ed=pi[sqrt2(x-1)];i<=ed;++i){
        ans += lehmer_pi(x/p[i])-i;
    }
    return ans;
}

int main(){
    init();
    for(LL n;~scanf("%lld",&n);)  {
        printf("%lld\n",lehmer_pi(n));
    }
    return 0;
}
```

21.4 Multiplicative Function

21.4.1 Euler Function

欧拉函数 $\varphi(n)$

$\varphi(n)$ 积性函数, 对于一个质数 p 和正整数 k, 有
$\varphi(p^k) = p^k - p^{k-1} = (p-1)p^{k-1} = p^k(1-\frac{1}{p})$

$n = \sum_{d|n} \varphi(d)$
当 $n > 1$ 时,1...n 中与 n 互质的整数和为 $n\frac{\varphi(n)}{2}$

```
void phi_table(){ //欧拉函数。。。
    int i,j;
    for(i=2; i<=5e6; i++)phi[i]=0;
    phi[1]=1;
    for(i=2; i<=5e6; i++)if(!phi[i])
        for(j=i; j<=5e6; j+=i){
            if(!phi[j])phi[j]=j;
            phi[j]=phi[j]/i*(i-1);
        }
    phi[0]=0;
```

```
    return 0;
}

void phi_table(int maxn){
    for(int i=1;i<=maxn;i++)phi[i]=i;
    for(int i=2;i<=maxn;i+=2)phi[i]/=2;
    for(int i=3;i<=maxn;i+=2)
    if(phi[i]==i)for(int j=i;j<=maxn;j+=i)
        phi[j]=phi[j]/i*(i-1);
    return ;
}
//  以上是打表的形式  这是求单个的
LL Phi(LL n)
{
    LL rea=n;
    for(int i=0; prime[i]*prime[i]<=n&&i<kp; i++)
    {
        if(n%prime[i]==0)
        {
            rea=rea-rea/prime[i];
            while(n%prime[i]==0)
                n/=prime[i];
        }
    }
    if(n>1)    rea=rea-rea/n;
    return rea;
}
/***********************/
// O(n) 求素数 + 欧拉函数
// 用最小的素因子筛掉每个数
int prime[N],phi[N],cnt;// prime: 记录质数, phi 记录欧拉函数
int Min_factor[N];// i 的最小素因子
bool vis[N];
void Init(){
    cnt=0;
    phi[1]=1;
    int x;
    for(int i=2;i<N;i++){
        if(!vis[i]){
            prime[++cnt]=i;
            phi[i]=i-1;
            Min_factor[i]=i;
        }
        for(int k=1;k<=cnt&&prime[k]*i<N;k++){
            x=prime[k]*i;
            vis[x]=true;
            Min_factor[x]=prime[k];
            if(i%prime[k]==0){
                phi[x]=phi[i]*prime[k];
                break;
            }
            else phi[x]=phi[i]*(prime[k]-1);
        }
    }
}
```

21.5 Fast Calculation

21.5.1 Fast Exponentiation Mod

```
inline LL qmod(LL a, LL b,LL P) {
    LL ans=1;
    for(; b; b>>=1, a=a*a%P)
        if(b&1) ans=ans*a%P;
    return ans;
}
```

21.5.2 Fast Multi

```
// O(1) 解决快速乘取膜
LLu qmodx(LLu a,LLu b,LLu c){
    a%=c,b%=c;
    if(c<=1000000000) return a*b%c;
    return (a*b-(LLu)(a/(long double)c*b+1e-8)*c+c)%c;
}
```

21.5.3 Fast Sqrt

```
int sqrt(float x) {
    if(x == 0) return 0;
    float result = x;
    float xhalf = 0.5f*result;
    int i = *(int*)&result;
    i = 0x5f375a86- (i>>1); // what the fuck?
    result = *(float*)&i;
    result = result*(1.5f-xhalf*result*result); // Newton step, repeating increases accuracy
    result = result*(1.5f-xhalf*result*result);
    return 1.0f/result;
}
```

21.5.4 Matrix

```
const int M = 4; // 方阵的大小为 M*M
struct Matrix{
    LL m[M][M];
    void clear0(){
        for(int i=0;i<M;i++)for(int j=0;j<M;j++)
                m[i][j]=0;
    }
    void clearE(){
        for(int i=0;i<M;i++)for(int j=0;j<M;j++)
                m[i][j]=(i==j);
    }
};
Matrix operator * (Matrix &a,Matrix &b){
    Matrix c;c.clear0();
    for(int k=0;k<M;k++)for(int i=0;i<M;i++)for(int j=0;j<M;j++)
                c.m[i][j]=(c.m[i][j]+a.m[i][k]*b.m[k][j]+MOD)%MOD;
    return c;
}
Matrix operator ^ (Matrix &a,LL b){
    Matrix c;c.clearE();
    for(;b;b>>=1,a=a*a)
        if(b&1) c=c*a;
    return c;
}
```

21.5.5 Fast Fourier Transform

```
/*********************************************
用于快速求卷积 c=a b
卷积可以类比两个多项式相乘
正常暴力求卷积的复杂度是 O(n^2) ,
通过 FFT 加速 求卷积的复杂度能降到 O(nlog_{2}n)
注意 FFT 的长度必须为 2^k
*********************************************/
struct Complex{
    double real, image;
    Complex(double _real, double _image){
        real = _real;
        image = _image;
    }
    Complex(){}

    Complex operator + (const Complex &tmp){return Complex(real + tmp.real, image +
    ↪  tmp.image);}
    Complex operator - (const Complex &tmp){return Complex(real - tmp.real, image -
    ↪  tmp.image);}
    Complex operator * (const Complex &tmp){return Complex(real*tmp.real - image*tmp.image,
    ↪  real*tmp.image + image*tmp.real);}
};
int rev(int id, int len){
    int ret = 0;
    for(int i = 0; (1 << i) < len; i++){
        ret <<= 1;
        if(id & (1 << i)) ret |= 1;
    }
    return ret;
}
Complex A[N];
void FFT(Complex *a, int len, int DFT){
    for(int i = 0; i < len; i++)
        A[rev(i, len)] = a[i];
    for(int s = 1; (1 << s) <= len; s++){
        int m = (1 << s);
        Complex wm = Complex(cos(DFT*2*PI/m), sin(DFT*2*PI/m));
        for(int k = 0; k < len; k += m){
            Complex w = Complex(1, 0);
            for(int j = 0; j < (m >> 1); j++){
                Complex t = w*A[k + j + (m >> 1)];
                Complex u = A[k + j];
                A[k + j] = u + t;
                A[k + j + (m >> 1)] = u - t;
                w = w*wm;
            }
        }
    }
    if(DFT == -1) for(int i = 0; i < len; i++) A[i].real /= len, A[i].image /= len;
    for(int i = 0; i < len; i++) a[i] = A[i];
    return;
}
int main()
{
    /**
    求卷积 c=a b
    la 为 a 的长度
```

```
    lb 为 b 的长度
    len 为最后结果的长度.
    **/
    int sa,sb;
    sa=sb=0;
    while((1<<sa)<la) sa++;
    while((1<<sb)<lb) sb++;
    int len = (1<<(max(sa,sb)+1));
    A = FFT(A,len,1);
    B = FFT(B,len,1);
    for(int i=0;i<len;i++)  A[i]=A[i]*B[i],ans[i]=0;
    A = FFT(A,len,-1);
    /**
    这是最后的卷积的结果.
    **/
}
```

21.5.6 Fast Number Theoretic Transform

```
/*****************************************
类似 FFT, 也是求一类卷积问题,
在模意义下的卷积问题, 对模数有要求. 多数情况下为 998244353
注意 NTT 的长度必须为 2^k
*****************************************/
const int Maxn=50000;
LL A[Maxn<<2],B[Maxn<<2];
int ans[Maxn<<2];
inline LL qmod(LL a, LL b,LL P) {
    LL ans=1;
    for(; b; b>>=1, a=a*a%P)
        if(b&1) ans=ans*a%P;
    return ans;
}
struct NTT {
    int pos[Maxn<<2],k,G,Mod;
    inline void init(int len) {
        Mod = 998244353,G = 3;
        for(k=1; k<=len; k<<=1);
        for(int i=1; i<k; i++)
            pos[i]=(i&1)?((pos[i>>1]>>1)^(k>>1)):(pos[i>>1]>>1);
    }
    inline void dft(LL *a) {
        for(int i=1; i<k; i++)if(pos[i]>i)swap(a[pos[i]],a[i]);
        for(int m1=1; m1<k; m1<<=1) {
            int m2=m1<<1;
            LL wn=qmod(G,(Mod-1)/m2,Mod)%Mod;
            for(int i=0; i<k; i+=m2) {
                LL w=1;
                for(int j=0; j<m1; j++) {
                    LL &A=a[i+j],&B=a[i+j+m1],t=B*w%Mod;
                    B=(A-t+Mod)%Mod;
                    A=(A+t)%Mod;
                    w=w*wn%Mod;
                }
            }
        }
    }
    inline void mui(LL *A,LL *B,int m) {
        init(m);
```

```
        dft(A);dft(B);
        for(int i=0; i<k; i++)A[i]=A[i]*B[i]%Mod;
        dft(A);
        reverse(A+1,A+k);
        int inv=qmod(k,Mod-2,Mod)%Mod;
        for(int i=0; i<k; i++)A[i]=inv*A[i]%Mod;
    }
} ntt;
```

21.5.7 Fast Walsh Transform

```
/*****************************************
解决一类卷积问题
c[x]=_{x=i j}a[i]×b[j] ， 表示位运算
注意 FWT 的长度必须为 2^k
*****************************************/
void FWT(int a[],int n){
    for(int d=1;d<n;d<<=1)
        for(int m=d<<1,i=0;i<n;i+=m)
            for(int j=0;j<d;j++){
                int x=a[i+j],y=a[i+j+d];
                a[i+j]=(x+y)%MOD,a[i+j+d]=(x-y+MOD)%MOD;
                //xor:a[i+j]=x+y,a[i+j+d]=(x-y+MOD)%MOD;
                //and:a[i+j]=x+y;
                //or :a[i+j+d]=x+y;
            }
}
void UFWT(int a[],int n){
    const int rev = (MOD+1)>>1;
    for(int d=1;d<n;d<<=1)
        for(int m=d<<1,i=0;i<n;i+=m)
            for(int j=0;j<d;j++){
                int x=a[i+j],y=a[i+j+d];
                a[i+j]=1LL*(x+y)*rev%MOD,a[i+j+d]=(1LL*(x-y)*rev%MOD+MOD)%MOD;
                //xor:a[i+j]=(x+y)/2,a[i+j+d]=(x-y)/2;  inv
                //and:a[i+j]=x-y;
                //or :a[i+j+d]=y-x;
            }
}
void solve(int a[],int b[],int n){
    FWT(a,n);
    FWT(b,n);
    for(int i=0;i<n;i++) a[i]=1LL*a[i]*b[i]%MOD;
    UFWT(a,n);
}
```

21.6 Primitive Root

原根

定义: 设 $m>1$,$\gcd(a,m)=1$, 使得 $a^r\equiv \pmod m$ 成立的最小的 r, 称为 a 对模 m 的阶, 记为.$\delta_m(a)$

定理: 如果模 m 有原根, 那么它一共有个 $\varphi(\varphi(m))$ 原根.

定理: 若 $m>1$,$\gcd(a,m)=1$,$a^n\equiv \pmod m$ 则 $\delta_m(a)|n$.

定理: 如果 p 为素数, 那么素数 p 一定存在原根, 并且模 p 的原根的个数为 $\varphi(p-1)$.

定理: 设 m 是正整数,a 是整数, 若 a 模 m 的阶等于 $\varphi(m)$, 则称 a 为模 m 的一个原根.

假设一个数 g 对于模 m 来说是原根, 那么 $g^i \pmod p$ 的结果两两不同, 且有 $1 < g < p, 0 \le i < p$ 那么 g 可以称为是模 p 的一个原根, 归根到底就是 $g^{p-1} \equiv \pmod p$ 当且仅当指数为 $p-1$ 的时候成立. (这里是素数)

模 m 有原根的充要条件: $m = 2, 4, p^a, 2p^a$, 其中 p 是奇素数.

求模素数 P 原根的方法: 对 $p-1$ 素因子分解, 即 $p-1 = p_1^{a_1} p_2^{a_2} ... p_k^{a_k}$ 是的标准分解式, 若恒有 $g^{\frac{p-1}{p_i}} \neq 1 \pmod p$ 成立, 则 g 就是 p 的原根. (对于合数求原根, 只需把 $p-1$ 换成 $\varphi(p)$ 即可)

```
// 如果 mod 为素数
int get(int mod) {
    for(int i = 2; ; i++) {
        set<int> s;
        for(int j = 1, x = 1; j < mod; j++) {
            x = (x*i)%mod;
            s.insert(x);
        }
        if(s.size() == mod-1) return i;
    }
    return -1; //没有原根 , 但是不可能 素数一定有原根
}

-----------------------------------------------------------------
//注意爆 int, 所以用 LL
LL qmod(LL a,LL b,LL c){
    LL res = 1;a%=c;
    for(;b;b>>=1,a=a*a%c)
        if(b&1) res=res*a%c;
    return res;
}

int prime[N];
int Is_or[N][2];

void Prime(){
    int n = 100000;
    prime[0]=0;
    memset(Is_or,1,sizeof(Is_or));
    for(int i=2;i<=n;i++){
        if(Is_or[i][0]) prime[++prime[0]]=i,Is_or[i][1]=prime[0];
        for(int j=1;j<=prime[0]&&i*prime[j]<=n;j++){
            Is_or[i*prime[j]][0]=0;
            if(0==i%prime[j]) break;
        }
    }
    return ;
}

int Phi(int x){
    //因为本题中数据都是质数, 所以欧拉函数值就都是 x-1 了。
    return x-1;
}

int a[10000],cnt;
void divide(int n){
    cnt = 0;
    for(int i=1;prime[i]*prime[i]<=n;i++){
        if(n<=prime[prime[0]]&&Is_or[n][0]){a[++cnt]=n;n=1;break;}
        if(n%prime[i]==0){a[++cnt]=prime[i];n/=prime[i];}
```

```
        while(n%prime[i]==0){n/=prime[i];}
    }
    if(n>1)a[++cnt]=n;
    return;
}

void work(int n){
    int phi = Phi(n);
    bool flag ;
    for(int i=2;i<n;i++){ //一个数的原根是很小的 所以暴力枚举就行，但其实是有优化方法的，
        flag = true;
        for(int j=1;j<=cnt;j++){
            int tmp = phi/a[j];
            if(qmod(i,tmp,n)==1){ flag = false; break; }
        }
        if(flag){ printf("%d\n",i);return ; }
    }
        puts(" 没有原根");
}

int main(){
    Prime();
    for(int n;~scanf("%d",&n);){
        divide(Phi(n));work(n);
    }
    return 0;
}
```

22 Combinatorial mathematics

22.1 combinatorial number

22.1.1 M 个盘子取 N 个球

1: 球同, 盒同, 盒不可以为空 $P_m(N)$-这符号表示部分数为 m 的 N-分拆的个数.
2: 球同, 盒同, 盒可以为空 $P_m(N+M)$ 为什么要加 M, 与 4 为什么要在 3 的基础上加 M 是一样的, 就是为了保证不为空
3: 球同, 盒不同, 盒不可以为空 $C(N-1, M-1)$
4: 球同, 盒不同, 盒可以为空 $C(N+M-1, M-1)$
5: 球不同, 盒同, 盒不可以为空 $S(N, M)$ –第二类斯特林数
6: 球不同, 盒同, 盒可以为空 $S(N,1)+S(N,2)+S(N,3)+...+S(N,M)$
7: 球不同, 盒不同, 盒不可以为空 $M!*S(N,M)$
8: 球不同, 盒不同, 盒可以为空 M^N –表示 M 的 N 次方

22.1.2 Pascal's Triangle

```
/**************************************
k*C(n,k)=n*C(n-1,k-1);
C(n,0)+C(n,2)+…=C(n,1)+C(n,3)+…
1*C(n,1)+2*C(n,2)+…+n*C(n,n)=n*2^(n-1)
**************************************/
LL f[2222][2222];
void init(){
    int n = 2000;
    for(int i=0; i<=n-1; i++)for(int j=0; j<=n-1; j++)
        f[i][0]=f[i][i]=1;
    for(int i=2; i<=n-1; i++)for(int j=1; j<=i-1; j++)
            f[i][j]=f[i-1][j-1]+f[i-1][j];
```

```
}
LL C(int n,int m){
    return f[n][m];
}
```

22.1.3 factorial/inverse

```
LL Fac[N],Inv[N];
LL qmod(LL a,LL b){
    LL res = 1ll;
    for(;b;b>>=1,a=a*a%MOD)
        if(b&1) res=res*a%MOD;
    return res;
}
void init(){
    //方法一  费马小定理
    fac[0]=1;
    for(LL i=1;i<N;i++)fac[i]=fac[i-1]*i%MOD;
    for(LL i=1;i<N;i++)inv[i]=qmod(fac[i],MOD-2);

    //方法二  inv{(n-i)!} = inv(n!)*n    //阶乘逆元
    Fac[0] = 1;
    for (LL i = 1; i < N; i++) Fac[i] = (Fac[i-1] * i) % MOD;
    Inv[N-1] = pow_mod(Fac[N-1], MOD-2);//Fac[N]^{MOD-2}
    for (LL i = N - 2; i >= 0; i--) Inv[i] = Inv[i+1] * (i + 1) % MOD;
}
LL C(int n,int m,int MOD){
        if(m>n) return 0;
        return Fac[n]*Inv[m]%MOD*Inv[n-m]%MOD;
}
LL A(int n,int m){
    if(m>n) return 0;
    return Fac[n]*Inv[n-m]%MOD;
}
```

22.1.4 Lucas

```
LL Lucas(LL n,LL m,LL p) {
    if (m==0) return 1;
    return (C(n%p,m%p,p)*Lucas(n/p,m/p,p))%p;
}
```

22.2 Catalan number

```
/**********************************
Catalan 数的定义
令 h(1)=1, Catalan 数满足递归式: h(n) = h(1)*h(n-1)+h(2)*h(n-2) + ... + h(n-1)h(1), n>=2
该递推关系的解为: h(n) = C(2n-2,n-1)/n, n=1,2,3,...() 其中 C(2n-2,n-1) 表示 2n-2 个中取 n-1 个
↪  的组合数)
1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796,
**********************************/
```

22.3 Stirling number[one]

```
/*************************************************
第一类 Stirling 数
定理: 第一类 Stirling 数 s(p,k) 计数的是把 p 个对象排成 k 个非空循环排列的方法数。
递推公式为:
s(p,p)=1 (p>=0)  有 p 个人和 p 个圆圈, 每个圆圈就只有一个人
```

```
s(p,0)=0 (p>=1)  如果至少有 1 个人，那么任何的安排都至少包含一个圆圈
s(p,k)=(p-1)*s(p-1,k)+s(p-1,k-1)
**************************************************/
long long s[maxn][maxn];//存放要求的第一类 Stirling 数
const long long mod=1e9+7;//取模

void init()//预处理
{
    memset(s,0,sizeof(s));
    s[1][1]=1;
    for(int i=2;i<=maxn-1;i++)
        for(int j=1;j<=i;j++)
    {
        s[i][j]=s[i-1][j-1]+(i-1)*s[i-1][j];
        if(s[i][j]>=mod)
            s[i][j]%=mod;
    }
}
```

22.4 Stirling number[two]

```
/*********************************************************
第二类 Stirling 数
定理：第二类 Stirling 数 S(p,k) 计数的是把 p 元素集合划分到 k 个不可区分的盒子里且没有空盒子的
↪ 划分个数。
递推公式有：
S(p,p)=1 (p>=0)
S(p,0)=0 (p>=1)
S(p,k)=k*S(p-1,k)+S(p-1,k-1)  (1<=k<=p-1)
*********************************************************/
long long s[maxn][maxn];//存放要求的 Stirling 数
const long long mod = 1e9 + 7; //取模

void init() { //预处理
    memset(s,0,sizeof(s));
    s[1][1]=1;
    for(int i=2;i<=maxn-1;i++)
        for(int j=1;j<=i;j++) {
            s[i][j]=s[i-1][j-1]+j*s[i-1][j];
            if (s[i][j]>=mod)
                s[i][j]%=mod;
        }
}
```

22.5 Bell number

```
/*************************************
Bell 数
定理：Bell 数 B(p) 是将 p 元素集合分到非空且不可区分盒子的划分个数（没有说分到几个盒子里面）。
B(p)=S(p,0)+S(p,1)+.....+S(p,k)  (S(,) 为第二类 Stirling)
*************************************/
```

22.6 Principle of inclusion-exclusion

```
// - dfs
void dfs(int id,bool flag,int cnt){
    /**
    按要求计算值 cnt
    **/
```

```
    if(flag ) ans += cnt;
    else      ans -= cnt;
    for(int i=id+1;i<m;i++)dfs(i,!flag,cnt);
}
int main(){
    ans = 0;
    for(int i=0;i<m;i++) dfs(i,true,1);
    printf("%d\n",ans);
}

// - 二进制枚举
int ans = 0,num,sum;
for(int i=1;i<(1<<m);i++){
    num = 0, sum = 1;
    for(int j=0;j<m;j++){
        if(i&(1<<j)){
            /**
            按要求计算值 sum
            **/
            num++;
        }
    }
    if(num&1) ans+=sum;
    else      ans-=sum;
}
```

22.7 Möbius inversion formula

设 f 为算术函数,f 的和函数 F 为 $F(n)=\sum_{d|n} f(d)$, 它是依据 f 的值决定的. 是否存在一种用 F 求 f 的简单方法? 这就是**莫比乌斯反演公式**

莫比乌斯函数

$$\mu(n)=\begin{cases} 1 & ,n=1 \\ (-1)^r & ,n=p_1*p_2*...*p_r \\ 0 & ,other \end{cases}$$

莫比乌斯函数是一个**乘性函数**

莫比乌斯函数的和函数 F,$F(n)=\sum_{d|n}\mu(d)$ 满足 $F(n)=\sum_{d|n}\mu(d)=\begin{cases} 1 & ,n=1 \\ 0 & ,n>1 \end{cases}$

莫比乌斯反演公式

对于 f 与其和函数 $F(n)=\sum_{d|n} f(d)$

形式一: $f(n)=\sum_{d|n}\mu(d)F(\frac{n}{d})$

形式二: $f(n)=\sum_{n|d}\mu(\frac{d}{n})F(d)$

注意有这样的两种形式,

```
// Sqrt() 计算
int mobius(int n){
        int m = 1;
        for(int i=2;i*i<=n;i++){
                if(n%i==0){
                        m*=-1;
                        int k = 0;
                        n/=i;
                        if(n%i==0) {m=0;break;}//某个素因子的幂大于 1
                }
        }
        if(n>1) m *= -1;
```

```
        return m;
}
// 普通筛法 求莫比乌斯函数
void mubius(){
    mu[1]=1;
    for(int i=1;i<=n;i++){
        for(int j=i+i;j<=n;i++)
            mu[j]-=mu[i];
    }
}

// 线性筛法预处理 mobius 函数
int prime[N],kp;
int Is_or[N],mu[N];

void Prime(){
    int x;
    mu[1]=1;
    memset(Is_or,true,sizeof(Is_or));
    for(int i=2;i<=n;i++){
        if(Is_or[i]) prime[kp++]=i,mu[i]=-1;
        for(int j=0;j<kp&&i*prime[j]<=n;j++){
            x = i*prime[j];
            Is_or[x]=false;
            if(0==i%prime[j]) break;
            mu[x] = -mu[i];
        }
    }
    return ;
}
```

23 Game Theory

23.1 Wythoff Game

```
// 威佐夫博弈 (Wythoff Game): 有两堆各若干个物品,
// 两个人轮流从某一堆或同时从两堆中取同样多的物品,
// 规定每次至少取一个, 多者不限, 最后取光者得胜。

void (int n,int m){
    if(n<m)n=n^m,m=m^n,n=n^m;
    int k=n-m;
    n=(int)(k*(1+sqrt(5))/2.0);
    if(n==m) first lose;
    else first win;
}
```

23.1.1 Sprague Grundy

```
// 预处理
int SG[MAX+10],s[N+10],has[N+10];
void getSG(int n){ //n 代表最大的状态既石子个数
    SG[0]=0;
    for(int i=1; i<=n; i++) {
        memset(has,0,sizeof(has));  //has 数组的大小可以优化 否则可能会 TLE
        for(int j=0; j<t; j++)      //t 代表所有可以取的石子的数量的种类数
            if(i-s[j]>=0/* 这里可以加入 has 数组的上界以防数组越界 */)
                has[SG[i-s[j]]]=1;
        for(int j=0;; j++)if(!has[j]){
            SG[i]=j;break;
        }
    }
}
// 记忆化搜索
//注意要先对 SG memset(SG,-1,sizeof(SG));
int SG[MAX+10],s[N+10];
bool has[min(MAX,N)+10];
int dfsSG(int n){
    if(SG[n]!=-1)return SG[n];
    memset(has,0,sizeof(has));
    for(int i=0; i<t; i++)
        if(x-s[i]>=0) vis[dfs(x-s[i])]=1;
    for(int i=0;; i++)if(0==has[i]){
        SG[x]=i;break;
    }
    return SG[n];
}
```

Part V
Computational Geometry

24 向量旋转

我们想将向量 $\overrightarrow{x,y}$ 以 x 为轴点逆时针旋转, 且旋转角为 α

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix}$$

25 Various Code

25.1 2-D

```
/*********************************************
尽量不要用除法，三角函数，强制类型转换（尤其是 double 转 int）等操作，否则精度损失比较大 。
const double PI = acos(-1.0);
const double EPS = 1e-8;
对小数点后 1 位 取整 的方法为 ±1 （+1 位进位 -1 为退位）
eg: 8(double)=7.9999999999…;
8(int)=8;
int(8(double))=7; !!!!!!
这样精度损失就大了 ~~~~
*********************************************/
#include <bits/stdc++.h>
typedef long long int LL;
using namespace std;

/*****************************************************/
const double eps = 1e-7;
const double Pi  = acos(-1.0);

double torad(double deg) { return deg / 180 * PI; }
inline int 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) { }
    inline void read(){scanf("%lf%lf", &x, &y);}
};
/*********************************
叉乘
(P1->P0)*(P2->P0) 的叉积
若结果为正，则 <P0,P1> 在 <P0,P2> 的顺时针方向;
若为 0 则 <P0,P1><P0,P2> 共线;
若为负则 <P0,P1> 在 <P0,P2> 的在逆时针方向;
*********************************/
double multi(Point p1,Point p2,Point p0){
    return ((p1.x-p0.x)*(p2.y-p0.y)-(p1.y-p0.y)*(p2.x-p0.x));
}
/*********************************
点乘
(P0->P1)·(P0->P2) 的点积
若结果为正，方向基本相同，夹角在 0° 到 90° 之间;
若为 0     ，正交，相互垂直;
```

```
若为负    , 方向基本相反, 夹角在 90° 到 180° 之间;
*********************************/
double dot(Point p1,Point p2,Point p0){
    return ((p1.x-p0.x)*(p2.x-p0.x)+(p1.y-p0.y)*(p2.y-p0.y));
}

//两点距离
double dis(Point p1,Point p2){
    return(sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y)));
}

//极角排序
bool cmp(Point a,Point b){ //极角排序
    return a.y*b.x < a.x*b.y;
}

/***************
线段
***************/
//线段相交 s1e1 和 s2e2
bool IsIntersected(Point s1,Point e1,Point s2,Point e2) { //两个线段相交
    return(max(s1.x,e1.x)>=min(s2.x,e2.x))&&
          (max(s2.x,e2.x)>=min(s1.x,e1.x))&&
          (max(s1.y,e1.y)>=min(s2.y,e2.y))&&
          (max(s2.y,e2.y)>=min(s1.y,e1.y))&&
          (multi(s1,s2,e1)*multi(s1,e2,e1)<=0)&&
          (multi(s2,s1,e2)*multi(s2,e1,e2)<=0);
}
//两线段交点 /*AB 与 CD 交点 */
point intersection(point &A,point &B,point &C,point &D){
    point p;
    double a1=A.y-B.y;
    double b1=B.x-A.x;
    double c1=A.x*B.y-B.x*A.y;

    double a2=C.y-D.y;
    double b2=D.x-C.x;
    double c2=C.x*D.y-D.x*C.y;

    p.x=(b1*c2-b2*c1)/(a1*b2-a2*b1);
    p.y=(a2*c1-a1*c2)/(a1*b2-a2*b1);
    return p;
}

//相交圆面积
struct Round {
    double x, y;
    double r;
}
double solve(Round a, Round b){
    double d = dis(a, b);
    if (d >= a.r + b.r) return 0;
    if (d <= fabs(a.r - b.r)) {
        double r = a.r < b.r ? a.r : b.r;
        return PI * r * r;
    }
    double ang1 = acos((a.r * a.r + d * d - b.r * b.r) / 2. / a.r / d);
    double ang2 = acos((b.r * b.r + d * d - a.r * a.r) / 2. / b.r / d);
```

```
    double ret = ang1 * a.r * a.r + ang2 * b.r * b.r - d * a.r * sin(ang1);
    return ret;
}

//点到直线距离

/******************
点 [p3] 在向量 [p1->p2] 上的投影
设 p0 为 p3 在 [p1->p2] 的投影点
k = |P0-P1|/|P2-P1|
******************/
double k(Point p1,Point p2,point p3){
    return (P3-P1)*(P2-P1)/(|P2-P1|*|P2-P1|);
}

//三角形面积
// * 已知三条边和 [外] 接圆半径, 公式 S = a*b*c/4/R
double trangle_Area(double a,double b,double c,double R){
    return a*b*c/4/R;
}
// * 已知三条边和 [内] 接圆半径, 公式 S = a*b*c/4/R
double trangle_Area(double a,double b,double c,double R){
    return r*(a+b+c)/2.0;
}
// * 已知三边求面积
double trangle_Area(double a,double b,double c){
    double p = (a+b+c)/2.0;
    return sqrt(p*(p-a)*(p-b)*(p-c));
}
// * 已知三点求面积
double trangle_Area(Point p1,Point p2,Point p3){
    double t = p1.y*(p3.x-p2.x)+p2.y*(p1.x-p3.x)+p3.y*(p2.x-p1.x);
    if(t<0) t=-t;
    return t/2.0;
}

//卷包法求凸包 (值得注意的是凸包大小 <3 的时候)
bool check(Point sp, Point ep, Point op){
    return (sp.x - op.x) * (ep.y - op.y) >= (ep.x - op.x) * (sp.y - op.y);
}
bool cmp_graham(Point a,Point b){
  return (a.y<b.y||a.y==b.y&&a.x<b.x);
}
// p[0~(n-1)] 为点集 n 为点集大小 res[0~(n-1)]/[1~n] 为凸包集合
int graham(Point *p, int n, Point *res){
    int i,len,top=1;
    sort(p,p+n,cmp_graham); //排序
    if(n==0) return 0; res[0]=p[0];
    if(n==1) return 1; res[1]=p[1];
    if(n==2) return 2; res[2]=p[2];
    for (i=2;i<n;i++){
        while(top&&check(p[i],res[top],res[top-1])) top--;
        res[++top]=p[i];
    }
    len = top;
    res[++top]=p[n-2];
    for (i=n-3;i>=0;i--){
        while(top!=len&&check(p[i],res[top],res[top-1]))top--;
```

```
        res[++top]=p[i];
    }
    return top; //  返回凸包中点的个数
}

//旋转卡壳
double rotating_calipers(point *poi,int n) {
    int q=1;//第一个的对踵点是第二个（初始化）
    double ans=0;//答案清零
    poi[n]=poi[0];//最后一个点是它本身 （防 +1 溢出）
    for(int p=0;p<n;p++){//找所有点
        while(abs(multi(poi[p+1],poi[q+1],poi[p]))>abs(multi(poi[p+1],poi[q],poi[p])))//找三
        ↪  角形面积最大的：枚举法
            q=(q+1)%n;//不忘模 n
        ans=max(ans,max(dis(poi[p],poi[q]),dis(poi[p+1],poi[q+1])));//三角形两边找一个最大的
        ↪  （还要处理 p+1 和 q+1 是防止平行）
        ans=max(ans,max(dis(poi[p],poi[q+1]),dis(poi[p+1],poi[q])));
    }
    return ans;
}

//最小矩形覆盖凸包 *p 凸包点集 n 凸包个数
double Cross(Point a,Point b,Point c){
    return (c.x-a.x)*(b.y-a.y) - (b.x-a.x)*(c.y-a.y);
}
double dot(Point p0,Point p1,Point p2){
    return ((p1.x-p0.x)*(p2.x-p0.x)+(p1.y-p0.y)*(p2.y-p0.y));
}
double rotating_calipers(Point *p,int n){
    int R=1,U=1,L;
    double ans = 10000000000000;
    p[n]=p[0];
    for(int i=0;i<n;i++){
        while(dcmp(Cross(p[i],p[i+1],p[U+1])-Cross(p[i],p[i+1],p[U]))<=0) U=(U+1)%n;//最上面
        ↪  一点
        while(dcmp(dot(p[i],p[i+1],p[R+1])-dot(p[i],p[i+1],p[R]))>0) R=(R+1)%n;//最右一点
        if(i==0) L=R;
        while(dcmp(dot(p[i],p[i+1],p[L+1])-dot(p[i],p[i+1],p[L]))<=0) L=(L+1)%n;//最左一点
        double d=dis(p[i],p[i+1])*dis(p[i],p[i+1]);
        double area=fabs(Cross(p[i],p[i+1],p[U]))* //求面积
            fabs(dot(p[i],p[i+1],p[R])-dot(p[i],p[i+1],p[L]))/d;
        if(area<ans) ans = area;
    }
    return ans;
}

//半平面交
Point P[maxn];
struct Line {
    Point a, b;
    double angle;
    void getAngle() {angle = atan2(b.y-a.y, b.x-a.x);}
} L[maxn], deq[maxn];

int dcmp(double x) {
    return x < -eps ? -1 : x > eps;
}
double xmult(Point a, Point b, Point c) {
```

```
    return (a.x-c.x)*(b.y-c.y)-(a.y-c.y)*(b.x-c.x);
}
bool cmp(Line u, Line v) {
    int d = dcmp(u.angle-v.angle);
    if(d) return d > 0;
    return dcmp(xmult(u.a, v.a, v.b)) > 0;
    ///Clockwise: 大于 0 取向量左半部分为半平面, 小于 0, 取右半部分
}
Point intersection(Line u, Line v)  {
    Point ret = u.a;
    double t = ((u.a.x-v.a.x)*(v.a.y-v.b.y)
              -(u.a.y-v.a.y)*(v.a.x-v.b.x))
              /((u.a.x-u.b.x)*(v.a.y-v.b.y)
              -(u.a.y-u.b.y)*(v.a.x-v.b.x));
    ret.x += (u.b.x-u.a.x)*t, ret.y += (u.b.y-u.a.y)*t;
    return ret;
}
bool judge(Line l1, Line l2, Line l3) {
    Point p = intersection(l2, l3);
    return dcmp(xmult(p, l1.a, l1.b)) < 0;
    ///Clockwise: 大于小于符号与上面 cmp() 中注释处相反
}
void HPI(Line L[], int n){
    sort(L, L+n, cmp);
    int tmp = 1;
    for(int i = 1; i < n; ++i) {
        if(dcmp(L[i].angle-L[tmp-1].angle) != 0) {
            L[tmp++] = L[i];
        }
    }
    n = tmp;
    deq[0] = L[0], deq[1] = L[1];
    head = 0, tail = 1;
    for(int i = 2; i < n; ++i) {
        while(head < tail && judge(L[i], deq[tail-1], deq[tail]))
            tail--;
        while(head < tail && judge(L[i], deq[head+1], deq[head]))
            head++;
        deq[++tail] = L[i];
    }
    while(head < tail && judge(deq[head], deq[tail-1], deq[tail]))
        tail--;
    while(head < tail && judge(deq[tail], deq[head+1], deq[head]))
        head++;
    if(head == tail) return ;
    it = 0;
    for(int i = head; i < tail; ++i) {
        P[it++] = intersection(deq[i], deq[i+1]);
    }
    if(tail > head+1) {
        P[it++] = intersection(deq[head], deq[tail]);
    }
}
double getArea(Point p[], int n) {
    double area = 0;
    for(int i = 1; i < n-1; ++i) {
        area += xmult(P[0], P[i], P[i+1]);
    }
```

```
    return fabs(area)/2.0;
}

int main()  {
    int n;
    while(~scanf("%d", &n)) {
        n += 4;  //加四个点 表示四周
        L[0]=(Line){(Point){0, 10000}, (Point){0, 0}};
        L[1]=(Line){(Point){10000, 10000}, (Point){0, 10000}};
        L[2]=(Line){(Point){10000, 0}, (Point){10000, 10000}};
        L[3]=(Line){(Point){0, 0}, (Point){10000, 0}};
        L[0].getAngle(), L[1].getAngle(), L[2].getAngle(), L[3].getAngle();
        for(int i = 4; i < n; ++i) {
            scanf("%lf%lf%lf%lf", &L[i].a.x, &L[i].a.y, &L[i].b.x, &L[i].b.y);
            L[i].getAngle();
        }
        HPI(L, n);
        printf("%.1f\n", getArea(P, it));
    }
    return 0;
}

//*********************************************************
#include<math.h>
#define MAXN 1000
#define offset 10000
#define eps 1e-8
#define PI acos(-1.0)//3.14159265358979323846
//判断一个数是否为 0，是则返回 true，否则返回 false
#define zero(x)(((x)>0?(x):-(x))<eps)
//返回一个数的符号，正数返回 1，负数返回 2，否则返回 0
#define _sign(x)((x)>eps?1:((x)<-eps?2:0))
struct point {
    double x,y;
};
struct line{
    point a,b;
};//直线通过的两个点，而不是一般式的三个系数
//求矢量 [p0,p1],[p0,p2] 的叉积
//p0 是顶点
//若结果等于 0，则这三点共线
//若结果大于 0，则 p0p2 在 p0p1 的逆时针方向
//若结果小于 0，则 p0p2 在 p0p1 的顺时针方向
double xmult(point p1,point p2,point p0){
    return(p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
//计算 dotproduct(P1-P0).(P2-P0)
double dmult(point p1,point p2,point p0){
    return(p1.x-p0.x)*(p2.x-p0.x)+(p1.y-p0.y)*(p2.y-p0.y);
}
//两点距离
double distance(point p1,point p2){
    return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
//判三点共线
int dots_inline(point p1,point p2,point p3){
    return zero(xmult(p1,p2,p3));
}
```

```
//判点是否在线段上，包括端点
int dot_online_in(point p,line l){
    return zero(xmult(p,l.a,l.b))&&(l.a.x-p.x)*(l.b.x-p.x)<eps&&(l.a.y-p.y)*(l.b.y-p.y)<eps;
}
//判点是否在线段上，不包括端点
int dot_online_ex(point p,line l){
    return dot_online_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-l.a.y))&&
           (!zero(p.x-l.b.x)||!zero(p.y-l.b.y));
}
//判两点在线段同侧，点在线段上返回 0
int same_side(point p1,point p2,line l){
    return xmult(l.a,p1,l.b)*xmult(l.a,p2,l.b)>eps;
}
//判两点在线段异侧，点在线段上返回 0
int opposite_side(point p1,point p2,line l){
    return xmult(l.a,p1,l.b)*xmult(l.a,p2,l.b)<-eps;
}
//判两直线平行
int parallel(line u,line v){
    return zero((u.a.x-u.b.x)*(v.a.y-v.b.y)-(v.a.x-v.b.x)*(u.a.y-u.b.y));
}
//判两直线垂直
int perpendicular(line u,line v){
    return zero((u.a.x-u.b.x)*(v.a.x-v.b.x)+(u.a.y-u.b.y)*(v.a.y-v.b.y));
}
//判两线段相交，包括端点和部分重合
int intersect_in(line u,line v){
    if(!dots_inline(u.a,u.b,v.a)||!dots_inline(u.a,u.b,v.b))
        return!same_side(u.a,u.b,v)&&!same_side(v.a,v.b,u);
    return
     ↪  dot_online_in(u.a,v)||dot_online_in(u.b,v)||dot_online_in(v.a,u)||dot_online_in(v.b,u);
}
//判两线段相交，不包括端点和部分重合
int intersect_ex(line u,line v){
    return opposite_side(u.a,u.b,v)&&opposite_side(v.a,v.b,u);
}
//计算两直线交点，注意事先判断直线是否平行!
//线段交点请另外判线段相交 (同时还是要判断是否平行!)
point intersection(line u,line v){
    point ret=u.a;
    double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))/
             ((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
    ret.x+=(u.b.x-u.a.x)*t;
    ret.y+=(u.b.y-u.a.y)*t;
    return ret;
}
//点到直线上的最近点
point ptoline(point p,line l){
    point t=p;
    t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
    return intersection(p,t,l.a,l.b);
}
//点到直线距离
double disptoline(point p,line l){
    return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);
}
//点到线段上的最近点
point ptoseg(point p,line l){
```

```
    point t=p;
    t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
    if(xmult(l.a,t,p)*xmult(l.b,t,p)>eps)
        return distance(p,l.a)<distance(p,l.b)?l.a:l.b;
    return intersection(p,t,l.a,l.b);
}
//点到线段距离
double disptoseg(point p,line l){
    point t=p;
    t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
    if(xmult(l.a,t,p)*xmult(l.b,t,p)>eps)
        return distance(p,l.a)<distance(p,l.b)?distance(p,l.a):distance(p,l.b);
    return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);
}
struct TPoint{
    double x,y;
    TPoint operator-(TPoint&a){
        TPoint p1;
        p1.x=x-a.x;
        p1.y=y-a.y;
        return p1;
    }
};

struct TLine{
    double a,b,c;
};

//求 p1 关于 p2 的对称点
TPoint symmetricalPoint(TPoint p1,TPoint p2){
    TPoint p3;
    p3.x=2*p2.x-p1.x;
    p3.y=2*p2.y-p1.y;
    return p3;
}
//p 点关于直线 L 的对称点
TPoint symmetricalPointofLine(TPoint p,TLine L){
    TPoint p2;
    double d;
    d=L.a*L.a+L.b*L.b;
    p2.x=(L.b*L.b*p.x-L.a*L.a*p.x-2*L.a*L.b*p.y-2*L.a*L.c)/d;
    p2.y=(L.a*L.a*p.y-L.b*L.b*p.y-2*L.a*L.b*p.x-2*L.b*L.c)/d;
    return p2;
}
//求线段所在直线，返回直线方程的三个系数
//两点式化为一般式
TLine lineFromSegment(TPoint p1,TPoint p2){
    TLine tmp;
    tmp.a=p2.y-p1.y;
    tmp.b=p1.x-p2.x;
    tmp.c=p2.x*p1.y-p1.x*p2.y;
    return tmp;
}
//求直线的交点
//求直线的交点，注意平行的情况无解，避免 RE
TPoint LineInter(TLine l1,TLine l2){
    //求两直线得交点坐标
    TPoint tmp;
```

```
    double a1=l1.a;
    double b1=l1.b;
    double c1=l1.c;
    double a2=l2.a;
    double b2=l2.b;
    double c2=l2.c;
    //注意这里 b1=0
    if(fabs(b1)<eps){
        tmp.x=-c1/a1;
        tmp.y=(-c2-a2*tmp.x)/b2;
    }
    else{
        tmp.x=(c1*b2-b1*c2)/(b1*a2-b2*a1);
        tmp.y=(-c1-a1*tmp.x)/b1;
    }
    //cout<<" 交点坐标"<<endl;
    //cout<<a1*tmp.x+b1*tmp.y+c1<<endl;
    //cout<<a2*tmp.x+b2*tmp.y+c2<<endl;
    return tmp;
}
//矢量（点）V 以 P 为顶点逆时针旋转 angle(弧度) 并放大 scale 倍
point rotate(point v,point p,double angle,double scale){
    point ret=p;
    v.x-=p.x,v.y-=p.y;
    p.x=scale*cos(angle);
    p.y=scale*sin(angle);
    ret.x+=v.x*p.x-v.y*p.y;
    ret.y+=v.x*p.y+v.y*p.x;
    return ret;
}
//矢量（点）V 以 P 为顶点逆时针旋转 angle(弧度)
point rotate(point v,point p,double angle){
    double cs=cos(angle),sn=sin(angle);
    v.x-=p.x,v.y-=p.y;
    p.x+=v.x*cs-v.y*sn;
    p.y+=v.x*sn+v.y*cs;
    return p;
}
```

25.2 3-D

```
/*****************************************
三维空间
*****************************************/
//点，线，面
struct point{
    double x,y,z;
};
struct line{
    point a,b;
};
struct plane{
    point a,b,c;
};
//计算 cross product U x V
point xmult(point u,point v){
    point ret;
    ret.x=u.y*v.z-v.y*u.z;
```

```
    ret.y=u.z*v.x-u.x*v.z;
    ret.z=u.x*v.y-u.y*v.x;
    return ret;
}
//计算 dot product  U . V
double dmult(point u,point v){
    return u.x*v.x+u.y*v.y+u.z*v.z;
}
//矢量差 U - V
point subt(point u,point v){
    point ret;
    ret.x=u.x-v.x;
    ret.y=u.y-v.y;
    ret.z=u.z-v.z;
    return ret;
}
//取平面法向量
point pvec(plane s){
    return xmult(subt(s.a,s.b),subt(s.b,s.c));
}
point pvec(point s1,point s2,point s3){
    return xmult(subt(s1,s2),subt(s2,s3));
}
//两点距离，单参数取向量大小
double dis(point p1,point p2){
    return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y)+(p1.z-p2.z)*(p1.z-p2.z));
}
// 向量大小
double vlen(point p){
    return sqrt(p.x*p.x+p.y*p.y+p.z*p.z);
}
//判断四点是不是共面
bool judge(point a,point b,point c,point d){
    double tmp = dmult(prec(a,b,c),smult(d,a));
    return  ( abs(tmp) < eps );
}
//点到平面距离
double ptoplane(point p,point s1,point s2,point s3){
    return fabs(dmult(pvec(s1,s2,s3),subt(p,s1)))/vlen(pvec(s1,s2,s3));
}
//三角形面积
double Area_triangle(point a,point b,point c){
    double ab=dis(a,b),bc=dis(b,c),ac=dis(a,c);
    double p=(ab+bc+ac)/2;
    return sqrt(p* (p-ab) * (p-bc)*(p-ac));
}
double Area_triangle(point b,point p1,point p2){
    point a=xmult(smult(b,p1),smult(b,p2));
    return sqrt(a.x*a.x+a.y*a.y+a.z*a.z)/2.0;
}
```

Part VI
STL

26 Set

```
// set 模版类的定义在头文件 <set> 中.

// 定义 set 对象的示例代码如下:

set<int> s;
set<double> ss;

// set 的基本操作:

s.begin()       //  返回指向第一个元素的迭代器
s.clear()       //  清除所有元素
s.count()       //  返回某个值元素的个数
s.empty()       //  如果集合为空, 返回 true(真)
s.end()         //  返回指向最后一个元素之后的迭代器, 不是最后一个元素
s.equal_range() //  返回集合中与给定值相等的上下限的两个迭代器
s.erase()       //  删除集合中的元素
s.find()        //  返回一个指向被查找到元素的迭代器
s.get_allocator()   //  返回集合的分配器
s.insert()      //  在集合中插入元素
s.lower_bound() //  返回指向大于 (或等于) 某值的第一个元素的迭代器
s.key_comp()    //  返回一个用于元素间值比较的函数
s.max_size()    //  返回集合能容纳的元素的最大限值
s.rbegin()      //  返回指向集合中最后一个元素的反向迭代器
s.rend()        //  返回指向集合中第一个元素的反向迭代器
s.size()        //  集合中元素的数目
s.swap()        //  交换两个集合变量
s.upper_bound() //  返回大于某个值元素的迭代器
s.value_comp()  //  返回一个用于比较元素间的值的函数

// multiset 和 set 的基本操作相似, 需要注意的是, 集合的 count() 能返回 0(无) 或者 1(有), 而多
↪  重集合是有多少个返回多少个.
```

27 Multiset

```
multiset

// 在 <set> 头文件中, 还定义了另一个非常实用的模版类 multiset(多重集合)。
// 多重集合与集合的区别在于集合中不能存在相同元素, 而多重集合中可以存在。

// 定义 multiset 对象的示例代码如下:

multiset<int> s;
multiset<double> ss;

// multiset 和 set 的基本操作相似, 需要注意的是, 集合的 count() 能返回 0(无) 或者 1(有),
// 而多重集合是有多少个返回多少个。
```

28 bitset

```
bitset
```

```
// 可以当作一个 bool 型数组考虑,bitset<N> bs;  可以考虑成一个数组 bool bs[N]。

// 相关操作:

bs.set();    //全部置 1,bs.reset() 全部置 0;
bs.set(pos); //等价于 bs[pos]=1,bs.reset(pos) 等价于 bs[pos]=0;


// 最重点的来了,bitset<N> a, b;

!a                  //按位取反
a^b                 //按位异或
a|b                 //按位或
a&b                 //按位与
a=b<<3              //整体移位
a.count();          //a 中 1 的个数

//bitset 优化有什么用呢

//如果有一个 bool 数组 a[N] 和 b[N] 把每一个位异或的话,一定是
for (int i = 0; i < N; ++i) c[i] = a[i] ^ b[i];

//但是如果用 bitset 直接 a^b 的话,只需要 O(N/机器字节数)
//这样可以实现常数优化。
```

29 Map

map

```
// 在 STL 的头文件中 <map> 中定义了模版类 map 和 multimap,用有序二叉树表存储类型为 pair<const
↪  Key, T> 的元素对序列。
// 序列中的元素以 const Key 部分作为标识,map 中所有元素的 Key 值必须是唯一的,multimap 则允许有
↪  重复的 Key 值。

// 可以将 map 看作是由 Key 标识元素的元素集合,这类容器也被称为“关联容器”,可以通过一个 Key
↪  值来快速决定一个元素,因此非常适合于需要按照 Key 值查找元素的容器。
// map 模版类需要四个模版参数,第一个是键值类型,第二个是元素类型,第三个是比较算子,第四个是分
↪  配器类型。
// 其中键值类型和元素类型是必要的。

// 定义 map 对象的代码示例:

map<string, int> m;
// map 的基本操作:

/*  向 map 中插入元素  */
m[key] = value; //  [key] 操作是 map 很有特色的操作,如果在 map 中存在键值为 key 的元素对,则
↪  返回该元素对的值域部分,否则将会创建一个键值为 key 的元素对,值域为默认值。所以可以用该操作
↪  向 map 中插入元素对或修改已经存在的元素对的值域部分。
m.insert(make_pair(key, value));    //  也可以直接调用 insert 方法插入元素对,insert 操作会返
↪  回一个 pair,当 map 中没有与 key 相匹配的键值时,其 first 是指向插入元素对的迭代器,其
↪  second 为 true;若 map 中已经存在与 key 相等的键值时,其 first 是指向该元素对的迭代
↪  器,second 为 false。

/*  查找元素  */
int i = m[key]; //  要注意的是,当与该键值相匹配的元素对不存在时,会创建键值为 key(当另一个元素
↪  是整形时,m[key]=0) 的元素对。
```

```
map<string, int>::iterator it = m.find(key);    // 如果 map 中存在与 key 相匹配的键值时,find
↪ 操作将返回指向该元素对的迭代器，否则，返回的迭代器等于 map 的 end()(参见 vector 中提到的
↪ begin() 和 end() 操作)。

/* 删除元素 */
m.erase(key);   // 删除与指定 key 键值相匹配的元素对，并返回被删除的元素的个数。
m.erase(it);    // 删除由迭代器 it 所指定的元素对，并返回指向下一个元素对的迭代器。

/* 其他操作 */
m.size();       // 返回元素个数
m.empty();      // 判断是否为空
m.clear();      // 清空所有元素
```

30 HashMap

Hashmap

```
// 线性处理冲突

struct hashmap{
    unsigned long long int a[maxn];
    int head[hashh];
    int next[maxn];
    int b[maxn];
    int size,cnt;
    void init(){ //注意初始化
        memset(head,-1,sizeof(head));
        size=cnt=0;
    }
    bool find(unsigned long long int val){
        int tmp=(val%hashh+hashh)%hashh;
        for(int i=head[tmp];i!=-1;i=next[i])if(val==a[i])
            return true;//b[i]; //可以映射为一个值
        return false;//0;
    }
    void add(unsigned long long int val){
        int tmp=(val%hashh+hashh)%hashh;
        if(find(val))return ;
        a[size]=val;
        b[size]=++cnt;
        next[size]=head[tmp];//令 next 指向-1、
        head[tmp]=size++;
    }
}H;
```

31 Queue

queue

```
// queue 模版类的定义在 <queue> 头文件中.
// queue 与 stack 相似,queue 模版类也需要两个模版参数，一个元素类型，一个容器类型，元素类型时必
↪ 须的，容器类型时可选的，默认为 deque 类型.

// 定义 queue 对象的示例代码必须如下:
queue<int> q;
queue<double> qq;
```

```
// queue 的基本操作:

q.push(x);  //  入队列
q.pop();    //  出队列
q.front();  //  访问队首元素
q.back();   //  访问队尾元素 s
q.empty();  //  判断队列是否为空
q.size();   //  访问队列中的元素个数

priority_queue

// 在 <queue> 头文件中，还定义了另一个非常有用的模版类 priority_queue(优先队列). 优先队列与队列
↪  的差别在于
// 优先队列不是按照入队的顺序出队，而是按照队列中元素的优先权出队列 (默认为大者优先，也可以通过
↪  指定算子
// 来指定自己的优先顺序).

// priority_queue 模版类有三个模版参数，第一个是元素类型，第二个是容器类型，第三个是比较算子.
// 其中后两者都可以忽略，默认容器为 vector，默认算子为 less，即小的往前排，大的往后排 (出队列时
↪  列尾元素先出队).
// 定义 priority_queue 对象的代码示例:

priority_queue<int> q;
priority_queue<pair<int, int> > qq;                    //  注意在两个尖括号之间一定要留空格，防
↪  止误判
priority_queue<int, vector<int>, greater<int> > qqq;//  定义小的先出队列

// priority_queue 的基本操作与 queue 的略微不同.

// priority_queue 的基本操作:

q.empty()     //  如果队列为空，则返回 true，否则返回 false
q.size()      //  返回队列中元素的个数
q.pop()       //  删除队首元素，但不返回其值
q.top()       //  返回具有最高优先级的元素值，但不删除该元素
q.push(item)  //  在基于优先级的适当位置插入新元素

// 初学者在使用 priority_queue 时，最困难的可能就是如何定义比较算子了.
// 如果是基本数据类型，或已定义了比较运算符的类，
// 可以直接使用 STL 的 less 算子和 greater 算子——默认为使用 less 算子.
// 如果要定义自己的比较算子，方法有多种，这里介绍其中一种: 重载比较运算符.
// 优先队列试图这两个元素 x 和 y 代入比较运算符 (对于 less 算子，调用 x < y，对于 greater 算
↪  子，调用 x > y),
// 若结果为真，则 x 排在 y 前面,y 将先出队列，反之，则 y 排在 x 前面,x 将先出队列.
// 如下算子示例:

class T{
    int x, y, z;
};

// 如果想要按照 z 的顺序从小到大出队列，只需要改动比较运算符重载为:

bool operator < (const T &tOne, const T &tTwo){
    return tOne.z > tTwo.z;  //  按照 z 的顺序来决定 tOne 和 tTwo 的顺序
}
// 则会得到和第二个例子一样的结果，所以，决定算子的是比较运算符重载函数内部的返回值.
```

Part VII
Other

32 Divide

32.1 Normal Divide

For Example :marge sort

```
void margesort(int a[],int f[],int p,int r){
    if(p==r) {f[p]=a[r];return ;}
    int mid = (p+r)>>1;
    margesort(a,f,p,mid);
    margesort(a,f,mid+1,r);

    int l1=p,l2=mid+1;
    for(int i=p;i<=r;i++){
        if(l1<=mid&&l2<=r){
            if(a[l1]<a[l2]) f[i]=a[l1++];
            else f[i]=a[l2++];
        }
        else if(l1<=mid)   f[i]=a[l1++];
        else if(l2<=r)     f[i]=a[l2++];
    }
    for(int i=p;i<=r;i++) a[i]=f[i];
    return ;
}
```

For Example :平面最近点对

```
struct P{
    int x, y;
    bool operator <(const P& B)const { return x < B.x; }
}p[100050];
int dis(P &A, P &B) { return (A.x-B.x)*(A.x-B.x) + (A.y-B.y)*(A.y-B.y); }
P Q[100050];
int Divide(int l, int r){
    if(l == r) return 1e7;
    int mid = (l+r)>>1, d, tx = p[mid].x, tot = 0;
    d = min(Divide(l, mid), Divide(mid+1, r));
    for(int i = l, j = mid+1; (i <= mid || j <= r); i++){
        while(j <= r && (p[i].y > p[j].y || i > mid)) Q[tot++] = p[j], j++; //归并按 y 排序
        if(abs(p[i].x - tx) < d && i <= mid) { //选择中间符合要求的点
            for(int k = j-1; k > mid && j-k < 3; k--) d = min(d, dis(p[i], p[k]));
            for(int k = j; k <= r && k-j < 2; k++) d = min(d, dis(p[i], p[k]));
        }
        if(i <= mid) Q[tot++] = p[i];
    }
    for(int i = l, j = 0; i <= r; i++, j++) p[i] = Q[j];
    return d;
}
int main(){
    int n;cin>>n;
    for(int i = 1; i <= n; i++) cin>>p[i].x>>p[i].y;
    sort(p+1, p+1+n); cout<<Divide(1, n)<<endl;
}
```

32.2 Tree Divide

32.2.1 Point Divide

For Example :

Description: 给你一棵树, 问你两点间距离不超过 k 的点对个数
Solve : 点分治, 每次找重心, 求经过重心的满足的解的个数, 最后累加.

```
#include <bits/stdc++.h>
typedef long long int LL;
using namespace std;
const int N   = 20000+7;

int n,k,ans;

bool vis[N];
int d[N],f[N];

struct edge{
    int to,next;
    int w;
}G[N<<1];
int head[N],tot;
void add(int u,int v,int w=0){
    G[tot].w=w,G[tot].to=v,G[tot].next=head[u],head[u]=tot++;
}

/******** 重心 begin********/
int sz[N],dn[N],siz,zx;

void getzx(int u,int fa=0){
    sz[u]=1;dn[u]=0;
    for(int i=head[u],to;i!=-1;i=G[i].next){
        to = G[i].to;
        if(to == fa||vis[to]) continue;
        getzx(to,u);
        sz[u]+=sz[to];
        dn[u]=max(dn[u],sz[to]);
    }
    dn[u]=max(dn[u],siz-sz[u]);
    if(dn[u]<dn[zx]) zx=u;
}

/********* 重心 end********/
/**
d[] 为当前节点到当前树根的距离
f[] 就是记录当前处理的子树的每个节点的 d[]
*/
void getd(int u,int fa=0){
    f[++f[0]] = d[u];
    for(int i=head[u],to;i!=-1;i=G[i].next){
        to=G[i].to;
        if(to==fa||vis[to]) continue;
        d[to]=d[u]+G[i].w;
        getd(to,u);
    }
}

int cal(int u,int w){
```

```
    d[u]=w; f[0]=0;int sum = 0;
    getd(u); sort(f+1,f+f[0]+1);
    for(int l=1,r=f[0];l<r;){
        if(f[l]+f[r]<=k){sum+=r-l;l++;}
        else r--;
    }
    return sum;
}
void solve(int u){
    ans+=cal(u,0);
    vis[u]=1;
    for(int i=head[u],to;i!=-1;i=G[i].next){
        to=G[i].to;
        if(vis[to]) continue;
        ans-=cal(to,G[i].w);
        siz=sz[to],zx=0;getzx(to);
        solve(zx);
    }
}
int main(){
    while(~scanf("%d%d",&n,&k)&&(n||k)){
        memset(head,-1,sizeof(head));
        memset(vis,0,sizeof(vis));
        dn[0]=n*10;zx=0;tot=0;

        for(int i=1,u,v,w;i<n;i++){
            scanf("%d%d%d",&u,&v,&w);
            add(u,v,w);add(v,u,w);
        }

        siz = n; getzx(1);
        ans = 0;solve(zx);

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

32.2.2 Edge Divide

For Example :

Description:
给你一棵树, 同样是两个操作:
(1) 单点颜色修改
(2) 询问整棵树中, 最远的白色两点的距离 <==> $maxdist(a,b), color[a] = color[b] = white$.
Solve : 边分治, 每次找中心边, 求经过中心边的满足的最远距离, 最后维护.
注意面对菊花树的时候边分治会退化, 所以要加虚点变成二叉树
具体也不大会, 看命吧

```
#include <bits/stdc++.h>
using namespace std;

#define MAXN 210000
#define MAXM 4000000
#define WHITE 1
#define BLACK 0

int V[MAXM],E[MAXM],NEXT[MAXM],FIR[MAXN]; int N;
int v[MAXN*2],e[MAXN*2],next[MAXN*2],pre[MAXN*2],fir[MAXN],ed[MAXN]; int n;
int col[MAXN],size[MAXN];
int p,mi,midedge,tot;

char buf[8000000],*pt = buf,*o = buf;
int getint(){
    int f = 1,x = 0;
    while((*pt != '-') && (*pt < '0' || *pt > '9'))    pt ++;
    if(*pt == '-')    f = -1,pt ++;    else    x = *pt++ - 48;
    while(*pt >= '0' && *pt <= '9')    x = x * 10 + *pt ++ - 48;
    return x * f;
}
char getch(){
    char ch;
    while(*pt < 'A' || *pt > 'Z')    pt ++;
    ch=*pt;pt++;
    return ch;
}

void ADD1(int x,int y,int z){
    V[tot]=y;E[tot]=z;NEXT[tot]=FIR[x];FIR[x]=tot;tot++;
}

void ADD(int x,int y,int z){
    V[tot]=y;E[tot]=z;NEXT[tot]=FIR[x];FIR[x]=tot;tot++;
    V[tot]=x;E[tot]=z;NEXT[tot]=FIR[y];FIR[y]=tot;tot++;
}

void add(int x,int y,int z){
    v[tot]=y;e[tot]=z;next[tot]=fir[x];fir[x]=tot;tot++;
    v[tot]=x;e[tot]=z;next[tot]=fir[y];fir[y]=tot;tot++;
}

void getpre(){
    memset(ed,255,sizeof(ed));
    for(int i=1;i<=n;i++)
        for(int j=fir[i];~j;j=next[j]){
```

```
            pre[j]=ed[i];
            ed[i]=j;
        }
}

void _delete(int x,int i){
    if(fir[x]==i) fir[x]=next[i]; else next[pre[i]]=next[i];
    if(ed[x]==i) ed[x]=pre[i]; else pre[next[i]]=pre[i];
}

void init(){
    memset(FIR,255,sizeof(FIR));tot=0;
    N=getint();
    for(int i=1;i<N;i++){
        int x=getint(),y=getint(),z=getint();
        ADD(x,y,z);
    }
}

void check(int u,int fa){
    int father=0;
    for(int i=FIR[u];~i;i=NEXT[i])
    if(V[i]!=fa)
        if(father==0){
            add(u,V[i],E[i]);
            father=u;
            check(V[i],u);
        }else{
            ++n;col[n]=BLACK;
            add(father,n,0);add(n,V[i],E[i]);
            father=n;
            check(V[i],u);
        }
}

void rebuild(){
    memset(fir,255,sizeof(fir));tot=0;
    n=N;
    for(int i=1;i<=n;i++) col[i]=WHITE;
    check(1,0);
    getpre();
    memset(FIR,255,sizeof(FIR));tot=0;
}

struct point{
    int dist,id;
    bool operator<(const point&b)const{
        return dist<b.dist;
    }
};

struct node{
    int rt,midlen,ans;
    int lc,rc;
    priority_queue<point>Q;
}T[MAXN*4];

int cnt=0;
```

```
void dfs_size(int u,int fa,int dist){
    ADD1(u,p,dist);if(col[u])T[p].Q.push((point){dist,u});
    size[u]=1;
    for(int i=fir[u];~i;i=next[i])
        if(v[i]!=fa){
            dfs_size(v[i],u,dist+e[i]);
            size[u]+=size[v[i]];
        }
}

void dfs_midedge(int u,int code){
    if(max(size[u],size[T[p].rt]-size[u])<mi){
        mi=max(size[u],size[T[p].rt]-size[u]);
        midedge=code;
    }
    for(int i=fir[u];~i;i=next[i])
        if(i!=(code^1))
            dfs_midedge(v[i],i);
}

void push_up(int p){
    T[p].ans=-1;
    while(!T[p].Q.empty()&&(col[T[p].Q.top().id]==0)) T[p].Q.pop();
    int lc=T[p].lc,rc=T[p].rc;
    if(lc==0&&rc==0){
         if(col[T[p].rt]) T[p].ans=0;
    }else{
        if(T[lc].ans>T[p].ans) T[p].ans=T[lc].ans;
        if(T[rc].ans>T[p].ans) T[p].ans=T[rc].ans;
        if(!T[lc].Q.empty()&&!T[rc].Q.empty())
            T[p].ans=max(T[lc].Q.top().dist+T[rc].Q.top().dist+T[p].midlen,T[p].ans);
    }
}

void dfs(int pt,int u){
    T[pt].rt=u;
    p=pt;dfs_size(u,0,0);
    midedge=-1;mi=n;dfs_midedge(u,-1);
    if(~midedge){
        int p1=v[midedge],p2=v[midedge^1];
        T[pt].midlen=e[midedge];
        _delete(p1,midedge^1);
        _delete(p2,midedge);
        dfs(T[pt].lc=++cnt,p1);
        dfs(T[pt].rc=++cnt,p2);
    }
    push_up(pt);
}

void change(int x){
    col[x]^=1;
    if(col[x]==BLACK)
        for(int i=FIR[x];~i;i=NEXT[i])
            push_up(V[i]);
    else
        for(int i=FIR[x];~i;i=NEXT[i]){
            T[V[i]].Q.push((point){E[i],x});
```

```
            push_up(V[i]);
        }
}


void solve(){
    int x;char op;
    int q=getint();
    while(q--){
        op=getch();
        if(op=='A')
            if(~T[1].ans)
                printf("%d\n",T[1].ans);
            else
                puts("They have disappeared.");
        else{
            x=getint();
            change(x);
        }
    }
}

int main(){
    init();
    rebuild();
    dfs(cnt=1,1);
    solve();
    return 0;
}
```

QTREE4

32.3 CDQ Divide

```
/***************
CDQ 分治解决 [三维偏序] 问题
一维排序 二维 CDQ 三维 BIT
对一个区间 [l,r] 它的答案是 [l,m]+[m+1,r]+[l,m] 对 [m+1,r] 影响的贡献
***************/
int n,m;
int f[N],ans[N];

typedef pair<pair<int,int>,pair<int,int> > node ;

node a[N],tl[N],tr[N];
bool cmp(node a,node b){return a.y<b.y;}

int sum[N];
#define lowbit(x) (x&-x)
int update(int i,int v){for(;i&&i<N;i+=lowbit(i))sum[i]+=v;}
int getSum(int i){int ans=0;for(;i;i-=lowbit(i))ans+=sum[i];return ans;}

void cal(int l,int r){
    int m = r+l >> 1;
    int ll=0,lr=0,pl=1,pr=1;
    for(int i=l  ;i<=m;i++) tl[++ll]=a[i];sort(tl+1,tl+ll+1,cmp);
    for(int i=m+1;i<=r;i++) tr[++lr]=a[i];sort(tr+1,tr+lr+1,cmp);

    vector<int> v;
    for(;pr<=lr;++pr){
        for(;pl<=ll&&tl[pl].y<=tr[pr].y;++pl){
            update(tl[pl].z,1);
            v.push_back(tl[pl].z);
        }
        ans[tr[pr].id]+=getSum(tr[pr].z);
    }
    for(int i=0;i<v.size();++i) update(v[i],-1);
}

void solve(int l,int r){
    if(l == r) return ;
    int m = r+l >> 1;
    solve(l  ,m);
    cal(l,r);
    solve(m+1,r);
}

int main(){
    scanf("%d%d",&n,&m);
    rep(i,1,n) scanf("%d%d%d",&a[i].x,&a[i].y,&a[i].z);
    sort(a+1,a+n+1);
    rep(i,1,n) a[i].id = i;
    solve(1,n);

    for(int i=n-1;i;i--)if(a[i].first==a[i+1].first && a[i].z == a[i+1].z){
        ans[i] = max(ans[i],ans[i+1]);
    }
    for(int i=1;i<=n;i++) ++f[ans[i]];
    for(int i=0;i<n;i++) printf("%d\n",f[i]);
    return 0;
}
```

33 手动开栈

C++
```
#pragma comment(linker, "/STACK:102400000,102400000")
```
G++
```
  int size = 256 << 20; // 256MB
    char *p = (char*)malloc(size) + size;
    __asm__("movl %0, %%esp\n" :: "r"(p));
```

34 fastIO

```
inline int read(){
    int x=0,f=1;char ch=getchar();
    while(ch<'0'||ch>'9'){if(ch=='-')f=-1;ch=getchar();}
    while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}
    return x*f;
}
inline void write(int x){
     if(x<0) putchar('-'),x=-x;
     if(x>9) write(x/10);
     putchar(x%10+'0');
}
```

35 Hash

35.1 Cantor expension

```
/*****************************************************
康托展开是对全排列的一个 hash
康托展开的公式是 X=an*(n-1)!+an-1*(n-2)!+…+ai*(i-1)!+…+a2*1!+a1*0!
其中，ai 为当前未出现的元素中是排在第几个（从 0 开始）。
*****************************************************/
int jiecheng[10] = {1,1,2,6,24,120,720,5040,40320,362880};
int a[10],b[10];
int Cantor_expansion(int a[],int len){
    int x=0;
    for(int i=0; i<len; i++)
        for(int j=i+1; j<len; j++)if(a[j]<a[i])
             x+=jiecheng[len-1-i];
    return x ;
}
bool h[len+5];//len 就是序列长度
void Cantor_inexpansion(int x,int len){
    memset(h,0,sizeof(h));
    int ind ,tem;
    for(int i=0,j; i<len; i++){
        tem = x/jiecheng[len-1-i];
        x  %=   jiecheng[len-1-i];
        for(j=1,ind=0;; j<=len; j++){
            if(h[j]) continue;
            if(ind==tem){a[i]=j;break;}
            ind++;
        }
        h[a[i]]=1;
    }
    return ;
}
```

35.2 String Hash

```
unsigned long long int h[5555],x[5555];
void Hash(char s[],int len){
    h[0]=0;x[0]=1;
    for(int i=1;i<=len;i++){
        x[i]=x[i-1]*29;
        h[i]=h[i-1]+x[i]*(s[i]-'a');
    }
}
bool judge(int l,int r,int l2,int r2){
    if(r-l != r2-l2) return false;
    if( (h[r]-h[l-1])*(x[l2-1]) == (h[r2]-h[l2-1]) )
        return true;
    return false;
}
```

36 BigNumber

36.1 C++ bignumber

鉴于现场赛的关系, 省略.

36.2 Java Bignumber

```
BigInteger or BigDecimal

import java.io.*;
import java.util.*;
import java.math.BigInteger;
public class Main{
    public static void main(String args[]) throws Exception {
        Scanner cin=new Scanner(System.in); //输入
        int t,n;
        n = cin.nextInt(); //整数输入
        System.out.println("Case #"+cas+": "+m); //Java 输出

        BigInteger a; //大数声明 一
        BigInteger b; //大数声明 一

        BigInteger.valueOf(mod) //大数声明 二  直接计算 , 类似 C++ 强转 支持整型和 string 类型

        cin.BigInteger();//读入一个 BigInteger;

        //大数运算
        a.add(b)    //a+b;
        a.subtract(); //相减
        a.multiply(); //相乘
        a.divide();   //相除取整
        a.remainder();//取余
        a.pow();      //a.pow(b)=a^b
        a.gcd();      //最大公约数
        a.abs();      //绝对值 |a|
        a.negate();   //取反数 -a
        a.mod();      //a.mod(b)=a%b=a.remainder(b);
        max();min();  //
        if( a.compareTo(b) == 0 ) System.out.println("a == b"); //大整数 a==b
    else if( a.compareTo(b) > 0 ) System.out.println("a > b"); //大整数 a>b
    else if( a.compareTo(b) < 0 ) System.out.println("a < b"); //大整数 a<b
        boolean equals(); //是否相等
        //
        a.toString()  // 转换为字符串
    }
}

BigDecimal //并且保留小数点后 2 位小数
DecimalFormat df = new DecimalFormat("0.00"); // 保留几位小数

stripTrailingZeros() //去掉小数末尾多余的 0
toPlainString()      //不会出现科学计数法的数
```

37 polynomial gcd

```
//多项式 gcd
#define vi vector<int>
int n;

int inv(int x){return qmod(x,n-2,n);}
vi vimod(vi f,vi g){
    int fz = f.size(),gz = g.size();
    for(int i=0;i<fz;i++){
        if(fz-i-gz < 0) break;
        int a=f[i]*inv(g[0])%n;
        for(int j=0;j<gz;j++){
            int now=i+j;
            f[now]=((f[now]-a*g[j]%n)%n+n)%n;
        }
    }
    vi ans;
    int p=-1;
    for(int i=0;i<fz;i++)if(f[i]!=0){p=i;break;}
    if(p>=0) for(int i=p;i<fz;i++)ans.pb(f[i]);
    return ans;
}

vi gcd(vi f,vi g){
    if(g.size()==0) return f;
    return  gcd(g,vimod(f,g));
}
vi f,g;
int main(){
    int kcase = 0;
    while(~scanf("%d",&n)&&n){
        f.clear(),g.clear();
        int d,x;
        scanf("%d",&d);
        for(int i=0;i<=d;i++){
            scanf("%d",&x);
            f.pb(x);
        }
        scanf("%d",&d);
        for(int i=0;i<=d;i++){
            scanf("%d",&x);
            g.pb(x);
        }
        vi ans = gcd(f,g);
        int tmp = inv(ans[0]);
        printf("Case %d: %d",++kcase,ans.size()-1);
        for(int i=0;i<ans.size();i++){
            printf(" %d",ans[i]*tmp%n);
        }
        puts("");
    }

    return 0;
}
```

KTH Royal Institute of Technology

Omogen Heap

Simon Lindholm, Johan Sannemo, Mårten Wiman

ACM-ICPC World Finals 2017

May 24, 2017

Contest (1)

template.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;

#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define trav(a, x) for(auto& a : x)
#define all(x) x.begin(), x.end()
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

int main() {
  cin.sync_with_stdio(0); cin.tie(0);
  cin.exceptions(cin.failbit);
}
```

.bashrc

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++14 \
  -fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps = <>
```

.vimrc

```
set cin aw ai is ts=4 sw=4 tm=50 nu noeb bg=dark ru cul
sy on   |   im jk <esc>   |   im kj <esc>   |   no ; :
```

troubleshoot.txt

```
Pre-submit:
Write a few simple test cases, if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.

Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all datastructures between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Any uninitialized variables?
Any overflows?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a team mate.
Ask the team mate to look at your code.
Go for a small walk, e.g. to the toilet.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a team mate do it.

Runtime error:
Have you tested all corner cases locally?
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
Any possible division by 0? (mod 0 for example)
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?
Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded:
Do you have any possible infinite loops?
What is the complexity of your algorithm?
Are you copying a lot of unnecessary data? (References)
How big is the input and output? (consider scanf)
Avoid vector, map. (use arrays/unordered_map)
What do your team mates think about your algorithm?

Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all datastructures between test cases?
```

Mathematics (2)

2.1 Equations

$$ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The extremum is given by $x = -b/2a$.

$$\begin{aligned} ax + by &= e \\ cx + dy &= f \end{aligned} \Rightarrow \begin{aligned} x &= \frac{ed - bf}{ad - bc} \\ y &= \frac{af - ec}{ad - bc} \end{aligned}$$

In general, given an equation $Ax = b$, the solution to a variable x_i is given by

$$x_i = \frac{\det A_i'}{\det A}$$

where A_i' is A with the i'th column replaced by b.

2.2 Recurrences

If $a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}$, and $r_1, \ldots, r_k$ are distinct roots of $x^k + c_1 x^{k-1} + \cdots + c_k$, there are $d_1, \ldots, d_k$ s.t.

$$a_n = d_1 r_1^n + \cdots + d_k r_k^n.$$

Non-distinct roots r become polynomial factors, e.g. $a_n = (d_1 n + d_2) r^n$.

2.3 Trigonometry

$$\sin(v + w) = \sin v \cos w + \cos v \sin w$$
$$cos(v + w) = \cos v \cos w - \sin v \sin w$$
$$\tan(v + w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$$
$$\sin v + \sin w = 2 \sin \frac{v + w}{2} \cos \frac{v - w}{2}$$
$$\cos v + \cos w = 2 \cos \frac{v + w}{2} \cos \frac{v - w}{2}$$
$$(V + W) \tan(v - w)/2 = (V - W) \tan(v + w)/2$$

where V, W are lengths of sides opposite angles v, w.

$$a \cos x + b \sin x = r \cos(x - \phi)$$
$$a \sin x + b \cos x = r \sin(x + \phi)$$

where $r = \sqrt{a^2 + b^2}, \phi = \text{atan2}(b, a)$.

2.4 Geometry

2.4.1 Triangles

Side lengths: a, b, c

Semiperimeter: $p = \frac{a + b + c}{2}$

Area: $A = \sqrt{p(p - a)(p - b)(p - c)}$

Circumradius: $R = \frac{abc}{4A}$

Inradius: $r = \frac{A}{p}$

Length of median (divides triangle into two equal-area triangles): $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$

Length of bisector (divides angles in two):

$$s_a = \sqrt{bc\left[1 - \left(\frac{a}{b + c}\right)^2\right]}$$

Law of sines: $\frac{\sin \alpha}{a} = \frac{\sin \beta}{b} = \frac{\sin \gamma}{c} = \frac{1}{2R}$

Law of cosines: $a^2 = b^2 + c^2 - 2bc \cos \alpha$

Law of tangents: $\frac{a + b}{a - b} = \frac{\tan \frac{\alpha + \beta}{2}}{\tan \frac{\alpha - \beta}{2}}$

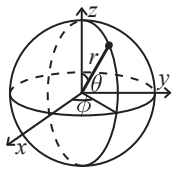
2.4.2 Quadrilaterals

With side lengths a, b, c, d, diagonals e, f, diagonals angle θ, area A and magic flux $F = b^2 + d^2 - a^2 - c^2$:

$$4A = 2ef \cdot \sin \theta = F \tan \theta = \sqrt{4e^2 f^2 - F^2}$$

For cyclic quadrilaterals the sum of opposite angles is $180°$, $ef = ac + bd$, and $A = \sqrt{(p - a)(p - b)(p - c)(p - d)}$.

2.4.3 Spherical coordinates



$$\begin{array}{cc} x = r\sin\theta\cos\phi & r = \sqrt{x^2+y^2+z^2} \\ y = r\sin\theta\sin\phi & \theta = \operatorname{acos}(z/\sqrt{x^2+y^2+z^2}) \\ z = r\cos\theta & \phi = \operatorname{atan2}(y,x) \end{array}$$

2.5 Derivatives/Integrals

$$\begin{array}{cc} \frac{d}{dx}\arcsin x = \frac{1}{\sqrt{1-x^2}} & \frac{d}{dx}\arccos x = -\frac{1}{\sqrt{1-x^2}} \\ \frac{d}{dx}\tan x = 1+\tan^2 x & \frac{d}{dx}\arctan x = \frac{1}{1+x^2} \\ \int \tan ax = -\frac{\ln|\cos ax|}{a} & \int x\sin ax = \frac{\sin ax - ax\cos ax}{a^2} \\ \int e^{-x^2} = \frac{\sqrt{\pi}}{2}\operatorname{erf}(x) & \int xe^{ax}dx = \frac{e^{ax}}{a^2}(ax-1) \end{array}$$

Integration by parts:

$$\int_a^b f(x)g(x)dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)dx$$

2.6 Sums

$$c^a + c^{a+1} + \cdots + c^b = \frac{c^{b+1}-c^a}{c-1}, c \neq 1$$

$$\begin{aligned} 1+2+3+\cdots+n &= \frac{n(n+1)}{2} \\ 1^2+2^2+3^2+\cdots+n^2 &= \frac{n(2n+1)(n+1)}{6} \\ 1^3+2^3+3^3+\cdots+n^3 &= \frac{n^2(n+1)^2}{4} \\ 1^4+2^4+3^4+\cdots+n^4 &= \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} \end{aligned}$$

2.7 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots, (-\infty < x < \infty)$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots, (-1 < x \leq 1)$$

$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \ldots, (-1 \leq x \leq 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, (-\infty < x < \infty)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots, (-\infty < x < \infty)$$

2.8 Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x. It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x xp_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$ where σ is the standard deviation. If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

Expectation is linear:

$$\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

For independent X and Y,

$$V(aX + bY) = a^2V(X) + b^2V(Y).$$

2.8.1 Discrete distributions

Binomial distribution

The number of successes in n independent yes/no experiments, each which yields success with probability p is $\operatorname{Bin}(n,p)$, $n = 1, 2, \ldots$, $0 \leq p \leq 1$.

$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$\mu = np,\ \sigma^2 = np(1-p)$$

$\operatorname{Bin}(n,p)$ is approximately $\operatorname{Po}(np)$ for small p.

First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each wich yields success with probability p is $\operatorname{Fs}(p)$, $0 \leq p \leq 1$.

$$p(k) = p(1-p)^{k-1},\ k = 1, 2, \ldots$$

$$\mu = \frac{1}{p},\ \sigma^2 = \frac{1-p}{p^2}$$

Poisson distribution

The number of events occurring in a fixed period of time t if these events occur with a known average rate κ and independently of the time since the last event is $\operatorname{Po}(\lambda)$, $\lambda = t\kappa$.

$$p(k) = e^{-\lambda}\frac{\lambda^k}{k!}, k = 0, 1, 2, \ldots$$

$$\mu = \lambda,\ \sigma^2 = \lambda$$

2.8.2 Continuous distributions

Uniform distribution

If the probability density function is constant between a and b and 0 elsewhere it is $\operatorname{U}(a,b)$, $a < b$.

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

$$\mu = \frac{a+b}{2},\ \sigma^2 = \frac{(b-a)^2}{12}$$

Exponential distribution

The time between events in a Poisson process is $\operatorname{Exp}(\lambda)$, $\lambda > 0$.

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$$\mu = \frac{1}{\lambda},\ \sigma^2 = \frac{1}{\lambda^2}$$

Normal distribution

Most real random values with mean μ and variance σ^2 are well described by $\mathcal{N}(\mu, \sigma^2)$, $\sigma > 0$.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

If $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$ then

$$aX_1 + bX_2 + c \sim \mathcal{N}(\mu_1 + \mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2)$$

2.9 Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let $X_1, X_2, \ldots$ be a sequence of random variables generated by the Markov process. Then there is a transition matrix $\mathbf{P} = (p_{ij})$, with $p_{ij} = \Pr(X_n = i | X_{n-1} = j)$, and $\mathbf{p}^{(n)} = \mathbf{P}^n \mathbf{p}^{(0)}$ is the probability distribution for X_n (i.e., $p_i^{(n)} = \Pr(X_n = i)$), where $\mathbf{p}^{(0)}$ is the initial distribution.

π is a stationary distribution if $\pi = \pi\mathbf{P}$. If the Markov chain is *irreducible* (it is possible to get to any state from any state), then $\pi_i = \frac{1}{\mathbb{E}(T_i)}$ where $\mathbb{E}(T_i)$ is the expected time between two visits in state i. π_j/π_i is the expected number of visits in state j between two visits in state i.

For a connected, undirected and non-bipartite graph, where the transition probability is uniform among all neighbors, π_i is proportional to node i's degree.

A Markov chain is *ergodic* if the asymptotic distribution is independent of the initial distribution. A finite Markov chain is ergodic iff it is irreducible and *aperiodic* (i.e., the gcd of cycle lengths is 1). $\lim_{k\to\infty} \mathbf{P}^k = \mathbf{1}\pi$.

A Markov chain is an A-chain if the states can be partitioned into two sets $\mathbf{A}$ and $\mathbf{G}$, such that all states in $\mathbf{A}$ are absorbing ($p_{ii} = 1$), and all states in $\mathbf{G}$ leads to an absorbing state in $\mathbf{A}$. The probability for absorption in state $i \in \mathbf{A}$, when the initial state is j, is $a_{ij} = p_{ij} + \sum_{k\in\mathbf{G}} a_{ik} p_{kj}$. The expected time until absorption, when the initial state is i, is $t_i = 1 + \sum_{k\in\mathbf{G}} p_{ki} t_k$.

Data structures (3)

OrderStatisticTree.h

Description: A set (not multiset!) with support for finding the n'th element, and finding the index of an element.
Time: $\mathcal{O}(\log N)$

16 lines

```
#include <bits/extc++.h>
using namespace __gnu_pbds;

template <class T>
using Tree = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

void example() {
  Tree<int> t, t2; t.insert(8);
  auto it = t.insert(10).first;
  assert(it == t.lower_bound(9));
  assert(t.order_of_key(10) == 1);
  assert(t.order_of_key(11) == 2);
  assert(*t.find_by_order(0) == 8);
  t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
}
```

SegmentTree.h

Description: Zero-indexed max-tree. Bounds are inclusive to the left and exclusive to the right. Can be changed by modifying T, LOW and f.
Time: $\mathcal{O}(\log N)$.

31 lines

```
struct Tree {
  typedef int T;
  const T LOW = -1234567890;
  T f(T a, T b) { return max(a, b); }

  int n;
  vi s;
  Tree() {}
  Tree(int m, T def=0) { init(m, def); }
  void init(int m, T def) {
    n = 1; while (n < m) n *= 2;
    s.assign(n + m, def);
    s.resize(2 * n, LOW);
    for (int i = n; i --> 1; )
      s[i] = f(s[i * 2], s[i*2 + 1]);
  }
  void update(int pos, T val) {
    pos += n;
    s[pos] = val;
    for (pos /= 2; pos >= 1; pos /= 2)
      s[pos] = f(s[pos * 2], s[pos * 2 + 1]);
  }
  T query(int l, int r) { return que(1, l, r, 0, n); }
  T que(int pos, int l, int r, int lo, int hi) {
    if (r <= lo || hi <= l) return LOW;
    if (l <= lo && hi <= r) return s[pos];
    int m = (lo + hi) / 2;
    return f(que(2 * pos, l, r, lo, m),
        que(2 * pos + 1, l, r, m, hi));
  }
};
```

LazySegmentTree.h

Description: Segment tree with ability to add or set values of large intervals, and compute max of intervals. Can be changed to other things. Use with a bump allocator for better performance, and SmallPtr or implicit indices to save memory.
Usage: `Node* tr = new Node(v, 0, sz(v));`
Time: $\mathcal{O}(\log N)$.

"../various/BumpAllocator.h" 50 lines

```
const int inf = 1e9;

struct Node {
  Node *l = 0, *r = 0;
  int lo, hi, mset = inf, madd = 0, val = -inf;
  Node(int lo,int hi):lo(lo),hi(hi){} // Large interval of -inf
  Node(vi& v, int lo, int hi) : lo(lo), hi(hi) {
    if (lo + 1 < hi) {
      int mid = lo + (hi - lo)/2;
      l = new Node(v, lo, mid); r = new Node(v, mid, hi);
      val = max(l->val, r->val);
    }
    else val = v[lo];
  }
  int query(int L, int R) {
    if (R <= lo || hi <= L) return -inf;
    if (L <= lo && hi <= R) return val;
    push();
    return max(l->query(L, R), r->query(L, R));
  }
  void set(int L, int R, int x) {
    if (R <= lo || hi <= L) return;
    if (L <= lo && hi <= R) mset = val = x, madd = 0;
    else {
      push(), l->set(L, R, x), r->set(L, R, x);
      val = max(l->val, r->val);
    }
  }
  void add(int L, int R, int x) {
    if (R <= lo || hi <= L) return;
    if (L <= lo && hi <= R) {
      if (mset != inf) mset += x;
      else madd += x;
      val += x;
    }
    else {
      push(), l->add(L, R, x), r->add(L, R, x);
      val = max(l->val, r->val);
    }
  }
  void push() {
    if (!l) {
      int mid = lo + (hi - lo)/2;
      l = new Node(lo, mid); r = new Node(mid, hi);
    }
    if (mset != inf)
      l->set(lo,hi,mset), r->set(lo,hi,mset), mset = inf;
    else if (madd)
      l->add(lo,hi,madd), r->add(lo,hi,madd), madd = 0;
  }
};
```

UnionFind.h

Description: Disjoint-set data structure.
Time: $\mathcal{O}(\alpha(N))$

13 lines

```
struct UF {
  vi e;
  UF(int n) : e(n, -1) {}
  bool same_set(int a, int b) { return find(a) == find(b); }
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x : e[x] = find(e[x]); }
  void join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return;
    if (e[a] > e[b]) swap(a, b);
    e[a] += e[b]; e[b] = a;
  }
};
```

SubMatrix.h

Description: Calculate submatrix sums quickly, given upper-left and lower-right corners (half-open).
Usage: `SubMatrix<int> m(matrix);`
`m.sum(0, 0, 2, 2); // top left 4 elements`
Time: $\mathcal{O}\left(N^2 + Q\right)$

13 lines

```
template <class T>
struct SubMatrix {
  vector<vector<T>> p;
  SubMatrix(vector<vector<T>>& v) {
    int R = sz(v), C = sz(v[0]);
    p.assign(R+1, vector<T>(C+1));
    rep(r,0,R) rep(c,0,C)
      p[r+1][c+1] = v[r][c] + p[r][c+1] + p[r+1][c] - p[r][c];
  }
  T sum(int u, int l, int d, int r) {
    return p[d][r] - p[d][l] - p[u][r] + p[u][l];
  }
};
```

Matrix.h

Description: Basic operations on square matrices.
Usage: `Matrix<int, 3> A;`
`A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}};`
`vector<int> vec = {1,2,3};`
`vec = (A^N) * vec;`

26 lines

```
template <class T, int N> struct Matrix {
  typedef Matrix M;
  array<array<T, N>, N> d{};
  M operator*(const M& m) const {
    M a;
    rep(i,0,N) rep(j,0,N)
      rep(k,0,N) a.d[i][j] += d[i][k]*m.d[k][j];
    return a;
  }
  vector<T> operator*(const vector<T>& vec) const {
    vector<T> ret(N);
    rep(i,0,N) rep(j,0,N) ret[i] += d[i][j] * vec[j];
    return ret;
  }
  M operator^(ll p) const {
    assert(p >= 0);
    M a, b(*this);
    rep(i,0,N) a.d[i][i] = 1;
    while (p) {
      if (p&1) a = a*b;
      b = b*b;
      p >>= 1;
    }
    return a;
  }
};
```

LineContainer.h

Description: Container where you can add lines of the form kx+m, and query maximum values at points x. Useful for dynamic programming.
Time: $\mathcal{O}\left(\log N\right)$

32 lines

```
bool Q;
struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const {
    return Q ? p < o.p : k < o.k;
  }
};

struct LineContainer : multiset<Line> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) { x->p = inf; return false; }
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    Q = 1; auto l = *lower_bound({0,0,x}); Q = 0;
    return l.k * x + l.m;
  }
};
```

Treap.h

Description: A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.
Time: $\mathcal{O}\left(\log N\right)$

55 lines

```
struct Node {
  Node *l = 0, *r = 0;
  int val, y, c = 1;
  Node(int val) : val(val), y(rand()) {}
  void recalc();
};

int cnt(Node* n) { return n ? n->c : 0; }
void Node::recalc() { c = cnt(l) + cnt(r) + 1; }

template<class F> void each(Node* n, F f) {
  if (n) { each(n->l, f); f(n->val); each(n->r, f); }
}

pair<Node*, Node*> split(Node* n, int k) {
  if (!n) return {};
  if (cnt(n->l) >= k) { // "n->val >= v" for lower_bound(v)
    auto pa = split(n->l, k);
    n->l = pa.second;
    n->recalc();
    return {pa.first, n};
  } else {
    auto pa = split(n->r, k - cnt(n->l) - 1);
    n->r = pa.first;
    n->recalc();
    return {n, pa.second};
  }
}

Node* merge(Node* l, Node* r) {
  if (!l) return r;
  if (!r) return l;
  if (l->y > r->y) {
    l->r = merge(l->r, r);
    l->recalc();
    return l;
  } else {
    r->l = merge(l, r->l);
    r->recalc();
    return r;
  }
}

Node* ins(Node* t, Node* n, int pos) {
  auto pa = split(t, pos);
  return merge(merge(pa.first, n), pa.second);
}

// Example application: move the range [l, r) to index k
void move(Node*& t, int l, int r, int k) {
  Node *a, *b, *c;
  tie(a,b) = split(t, l); tie(b,c) = split(b, r - l);
  if (k <= l) t = merge(ins(a, b, k), c);
  else t = merge(a, ins(c, b, k - r));
}
```

FenwickTree.h

Description: Computes partial sums a[0] + a[1] + ... + a[pos - 1], and updates single elements a[i], taking the difference between the old and new value.
Time: Both operations are $\mathcal{O}\left(\log N\right)$.

22 lines

```
struct FT {
  vector<ll> s;
  FT(int n) : s(n) {}
  void update(int pos, ll dif) { // a[pos] += dif
    for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
  }
  ll query(int pos) { // sum of values in [0, pos)
    ll res = 0;
    for (; pos > 0; pos &= pos - 1) res += s[pos-1];
    return res;
  }
  int lower_bound(ll sum) {// min pos st sum of [0, pos] >= sum
    // Returns n if no sum is >= sum, or -1 if empty sum is.
    if (sum <= 0) return -1;
    int pos = 0;
    for (int pw = 1 << 25; pw; pw >>= 1) {
      if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
        pos += pw, sum -= s[pos-1];
    }
    return pos;
  }
};
```

FenwickTree2d.h

Description: Computes sums a[i,j] for all i<I, j<J, and increases single elements a[i,j]. Requires that the elements to be updated are known in advance (call fakeUpdate() before init()).
Time: $\mathcal{O}\left(\log^2 N\right)$. (Use persistent segment trees for $\mathcal{O}\left(\log N\right)$.)

"FenwickTree.h" 22 lines

```
struct FT2 {
  vector<vi> ys; vector<FT> ft;
  FT2(int limx) : ys(limx) {}
  void fakeUpdate(int x, int y) {
    for (; x < sz(ys); x |= x + 1) ys[x].push_back(y);
  }
  void init() {
    trav(v, ys) sort(all(v)), ft.emplace_back(sz(v));
  }
  int ind(int x, int y) {
    return (int)(lower_bound(all(ys[x]), y) - ys[x].begin()); }
  void update(int x, int y, ll dif) {
    for (; x < sz(ys); x |= x + 1)
      ft[x].update(ind(x, y), dif);
  }
  ll query(int x, int y) {
    ll sum = 0;
```

```
    for (; x; x &= x - 1)
      sum += ft[x-1].query(ind(x-1, y));
    return sum;
  }
};
```

RMQ.h

Description: Range Minimum Queries on an array. Returns min(V[a], V[a + 1], ... V[b - 1]) in constant time. Set inf to something reasonable before use.
Usage: `RMQ rmq(values);`
`rmq.query(inclusive, exclusive);`
Time: $\mathcal{O}(|V| \log |V| + Q)$

21 lines

```
const int inf = numeric_limits<int>::max();

template <class T>
struct RMQ {
  vector<vector<T>> jmp;

  RMQ(const vector<T>& V) {
    int N = sz(V), on = 1, depth = 1;
    while (on < sz(V)) on *= 2, depth++;
    jmp.assign(depth, V);
    rep(i,0,depth-1) rep(j,0,N)
      jmp[i+1][j] = min(jmp[i][j],
      jmp[i][min(N - 1, j + (1 << i))]);
  }

  T query(int a, int b) {
    if (b <= a) return inf;
    int dep = 31 - __builtin_clz(b - a);
    return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
  }
};
```

Numerical (4)

Polynomial.h

18 lines

```
struct Polynomial {
  int n; vector<double> a;
  Polynomial(int n): n(n), a(n+1) {}
  double operator()(double x) const {
    double val = 0;
    for(int i = n; i >= 0; --i) (val *= x) += a[i];
    return val;
  }
  void diff() {
    rep(i,1,n+1) a[i-1] = i*a[i];
    a.pop_back(); --n;
  }
  void divroot(double x0) {
    double b = a.back(), c; a.back() = 0;
    for(int i=n--; i--;) c = a[i], a[i] = a[i+1]*x0+b, b=c;
    a.pop_back();
  }
};
```

BinarySearch.h

Description: Finds a zero point of f on the interval [a b]. f(a) must be less than 0 and f(b) greater than 0. Useful for solving equations like kx=sin(x) as in the example below.
Usage: `double func(double x) { return .23*x-sin(x); }`
`double x0 = bs(0,4,func);`
Time: $\mathcal{O}(\log((b-a)/\epsilon))$

9 lines

```
double bs(double a, double b, double (*f)(double)) {
  //for(int i = 0; i < 60; ++i){
  while (b-a > 1e-6) {
    double m = (a+b)/2;
    if (f(m) > 0) b = m;
    else a = m;
  }
  return a;
}
```

GoldenSectionSearch.h

Description: Finds the argument minimizing the function f in the interval $[a,b]$ assuming f is unimodal on the interval, i.e. has only one local minimum. The maximum error in the result is *eps*. Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.
Usage: `double func(double x) { return 4+x+.3*x*x; }`
`double xmin = gss(-1000,1000,func);`
Time: $\mathcal{O}(\log((b-a)/\epsilon))$

14 lines

```
double gss(double a, double b, double (*f)(double)) {
  double r = (sqrt(5)-1)/2, eps = 1e-7;
  double x1 = b - r*(b-a), x2 = a + r*(b-a);
  double f1 = f(x1), f2 = f(x2);
  while (b-a > eps)
    if (f1 < f2) { //change to > to find maximum
      b = x2; x2 = x1; f2 = f1;
      x1 = b - r*(b-a); f1 = f(x1);
    } else {
      a = x1; x1 = x2; f1 = f2;
      x2 = a + r*(b-a); f2 = f(x2);
    }
  return a;
}
```

PolyRoots.h

Description: Finds the real roots to a polynomial.
Usage: `vector<double> roots; Polynomial p(2);`
`p.a[0] = 2; p.a[1] = -3; p.a[2] = 1;`
`poly_roots(p,-1e10,1e10,roots); // x^2-3x+2=0`

"Polynomial.h" 25 lines

```
void poly_roots(const Polynomial& p, double xmin, double xmax,
     vector<double>& roots) {
  if (p.n == 1) { roots.push_back(-p.a.front()/p.a.back()); }
  else {
    Polynomial d = p;
    d.diff();
    vector<double> dr;
    poly_roots(d, xmin, xmax, dr);
    dr.push_back(xmin-1);
    dr.push_back(xmax+1);
    sort(all(dr));
    for (auto i = dr.begin(), j = i++; i != dr.end(); j = i++){
      double l = *j, h = *i, m, f;
      bool sign = p(l) > 0;
      if (sign ^ (p(h) > 0)) {
        //for(int i = 0; i < 60; ++i){
        while(h - l > 1e-8) {
          m = (l + h) / 2, f = p(m);
          if ((f <= 0) ^ sign) l = m;
          else h = m;
        }
        roots.push_back((l + h) / 2);
      }
    }
  }
}
```

PolyInterpolate.h

Description: Given n points (x[i], y[i]), computes an n-1-degree polynomial p that passes through them: $p(x) = a[0] * x^0 + ... + a[n-1] * x^{n-1}$. For numerical precision, pick $x[k] = c * \cos(k/(n-1) * \pi), k = 0 \ldots n-1$.
Time: $\mathcal{O}(n^2)$

13 lines

```
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
  vd res(n), temp(n);
  rep(k,0,n-1) rep(i,k+1,n)
    y[i] = (y[i] - y[k]) / (x[i] - x[k]);
  double last = 0; temp[0] = 1;
  rep(k,0,n) rep(i,0,n) {
    res[i] += y[k] * temp[i];
    swap(last, temp[i]);
    temp[i] -= last * x[k];
  }
  return res;
}
```

HillClimbing.h

Description: Poor man's optimization for unimodal functions.

16 lines

```
typedef array<double, 2> P;

double func(P p);

pair<double, P> hillClimb(P start) {
  pair<double, P> cur(func(start), start);
  for (double jmp = 1e9; jmp > 1e-20; jmp /= 2) {
    rep(j,0,100) rep(dx,-1,2) rep(dy,-1,2) {
      P p = cur.second;
      p[0] += dx*jmp;
      p[1] += dy*jmp;
      cur = min(cur, make_pair(func(p), p));
    }
  }
  return cur;
}
```

Integrate.h

Description: Simple integration of a function over an interval using Simpson's rule. The error should be proportional to h^4, although in practice you will want to verify that the result is stable to desired precision when epsilon changes.

8 lines

```
double quad(double (*f)(double), double a, double b) {
  const int n = 1000;
  double h = (b - a) / 2 / n;
  double v = f(a) + f(b);
  rep(i,1,n*2)
    v += f(a + i*h) * (i&1 ? 4 : 2);
  return v * h / 3;
}
```

IntegrateAdaptive.h

Description: Fast integration using an adaptive Simpson's rule.
Usage: `double z, y;`
`double h(double x) { return x*x + y*y + z*z <= 1; }`
`double g(double y) { ::y = y; return quad(h, -1, 1); }`
`double f(double z) { ::z = z; return quad(g, -1, 1); }`
`double sphereVol = quad(f, -1, 1), pi = sphereVol*3/4;`

16 lines

```
typedef double d;
d simpson(d (*f)(d), d a, d b) {
  d c = (a+b) / 2;
  return (f(a) + 4*f(c) + f(b)) * (b-a) / 6;
}
d rec(d (*f)(d), d a, d b, d eps, d S) {
```

```
  d c = (a+b) / 2;
  d S1 = simpson(f, a, c);
  d S2 = simpson(f, c, b), T = S1 + S2;
  if (abs (T - S) <= 15*eps || b-a < 1e-10)
    return T + (T - S) / 15;
  return rec(f, a, c, eps/2, S1) + rec(f, c, b, eps/2, S2);
}
d quad(d (*f)(d), d a, d b, d eps = 1e-8) {
  return rec(f, a, b, eps, simpson(f, a, b));
}
```

Determinant.h

Description: Calculates determinant of a matrix. Destroys the matrix.
Time: $\mathcal{O}(N^3)$

15 lines

```
double det(vector<vector<double>>& a) {
  int n = sz(a); double res = 1;
  rep(i,0,n) {
    int b = i;
    rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
    if (i != b) swap(a[i], a[b]), res *= -1;
    res *= a[i][i];
    if (res == 0) return 0;
    rep(j,i+1,n) {
      double v = a[j][i] / a[i][i];
      if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
    }
  }
  return res;
}
```

IntDeterminant.h

Description: Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.
Time: $\mathcal{O}(N^3)$

18 lines

```
const ll mod = 12345;
ll det(vector<vector<ll>>& a) {
  int n = sz(a); ll ans = 1;
  rep(i,0,n) {
    rep(j,i+1,n) {
      while (a[j][i] != 0) { // gcd step
        ll t = a[i][i] / a[j][i];
        if (t) rep(k,i,n)
          a[i][k] = (a[i][k] - a[j][k] * t) % mod;
        swap(a[i], a[j]);
        ans *= -1;
      }
    }
    ans = ans * a[i][i] % mod;
    if (!ans) return 0;
  }
  return (ans + mod) % mod;
}
```

Simplex.h

Description: Solves a general linear maximization problem: maximize $c^T x$ subject to $Ax \leq b$, $x \geq 0$. Returns -inf if there is no solution, inf if there are arbitrarily good solutions, or the maximum value of $c^T x$ otherwise. The input vector is set to an optimal x (or in the unbounded case, an arbitrary solution fulfilling the constraints). Numerical stability is not guaranteed. For better performance, define variables such that $x = 0$ is viable.
Usage: `vvd A = {{1,-1}, {-1,1}, {-1,-2}};`
`vd b = {1,1,-4}, c = {-1,-1}, x;`
`T val = LPSolver(A, b, c).solve(x);`
Time: $\mathcal{O}(NM * \#pivots)$, where a pivot may be e.g. an edge relaxation. $\mathcal{O}(2^n)$ in the general case.

68 lines

```
typedef double T; // long double, Rational, double + mod<P>...
typedef vector<T> vd;
typedef vector<vd> vvd;

const T eps = 1e-8, inf = 1/.0;
#define MP make_pair
#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])) s=j

struct LPSolver {
  int m, n;
  vi N, B;
  vvd D;

  LPSolver(const vvd& A, const vd& b, const vd& c) :
    m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
      rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
      rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i];}
      rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
      N[n] = -1; D[m+1][n] = 1;
    }

  void pivot(int r, int s) {
    T *a = D[r].data(), inv = 1 / a[s];
    rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
      T *b = D[i].data(), inv2 = b[s] * inv;
      rep(j,0,n+2) b[j] -= a[j] * inv2;
      b[s] = a[s] * inv2;
    }
    rep(j,0,n+2) if (j != s) D[r][j] *= inv;
    rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }

  bool simplex(int phase) {
    int x = m + phase - 1;
    for (;;) {
      int s = -1;
      rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
      if (D[x][s] >= -eps) return true;
      int r = -1;
      rep(i,0,m) {
        if (D[i][s] <= eps) continue;
        if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
                     < MP(D[r][n+1] / D[r][s], B[r])) r = i;
      }
      if (r == -1) return false;
      pivot(r, s);
    }
  }

  T solve(vd &x) {
    int r = 0;
    rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] < -eps) {
      pivot(r, n);
      if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
      rep(i,0,m) if (B[i] == -1) {
        int s = 0;
        rep(j,1,n+1) ltj(D[i]);
        pivot(i, s);
      }
    }
    bool ok = simplex(1); x = vd(n);
    rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return ok ? D[m][n+1] : inf;
  }
};
```

SolveLinear.h

Description: Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost.
Time: $\mathcal{O}(n^2 m)$

38 lines

```
typedef vector<double> vd;
const double eps = 1e-12;

int solveLinear(vector<vd>& A, vd& b, vd& x) {
  int n = sz(A), m = sz(x), rank = 0, br, bc;
  if (n) assert(sz(A[0]) == m);
  vi col(m); iota(all(col), 0);

  rep(i,0,n) {
    double v, bv = 0;
    rep(r,i,n) rep(c,i,m)
      if ((v = fabs(A[r][c])) > bv)
        br = r, bc = c, bv = v;
    if (bv <= eps) {
      rep(j,i,n) if (fabs(b[j]) > eps) return -1;
      break;
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) swap(A[j][i], A[j][bc]);
    bv = 1/A[i][i];
    rep(j,i+1,n) {
      double fac = A[j][i] * bv;
      b[j] -= fac * b[i];
      rep(k,i+1,m) A[j][k] -= fac*A[i][k];
    }
    rank++;
  }

  x.assign(m, 0);
  for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    rep(j,0,i) b[j] -= A[j][i] * b[i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

SolveLinear2.h

Description: To get all uniquely determined values of x back from SolveLinear, make the following changes:

"SolveLinear.h" 7 lines

```
rep(j,0,n) if (j != i) // instead of rep(j,i+1,n)
// ... then at the end:
x.assign(m, undefined);
rep(i,0,rank) {
  rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
  x[col[i]] = b[i] / A[i][i];
fail:; }
```

SolveLinearBinary.h

Description: Solves $Ax = b$ over $\mathbb{F}_2$. If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys A and b.
Time: $\mathcal{O}(n^2 m)$

34 lines

```
typedef bitset<1000> bs;

int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
  int n = sz(A), rank = 0, br;
  assert(m <= sz(x));
  vi col(m); iota(all(col), 0);
  rep(i,0,n) {
```

```
    for (br=i; br<n; ++br) if (A[br].any()) break;
    if (br == n) {
      rep(j,i,n) if(b[j]) return -1;
      break;
    }
    int bc = (int)A[br]._Find_next(i-1);
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) if (A[j][i] != A[j][bc]) {
      A[j].flip(i); A[j].flip(bc);
    }
    rep(j,i+1,n) if (A[j][i]) {
      b[j] ^= b[i];
      A[j] ^= A[i];
    }
    rank++;
  }

  x = bs();
  for (int i = rank; i--;) {
    if (!b[i]) continue;
    x[col[i]] = 1;
    rep(j,0,i) b[j] ^= A[j][i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

MatrixInverse.h

Description: Invert matrix A. Returns rank; result is stored in A unless singular (rank < n). Can easily be extended to prime moduli; for prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1})$ (mod p^k) where A^{-1} starts as the inverse of A mod p, and k is doubled in each step.
Time: $\mathcal{O}(n^3)$

36 lines

```
int matInv(vector<vector<double>>& A) {
  int n = sz(A); vi col(n);
  vector<vector<double>> tmp(n, vector<double>(n));
  rep(i,0,n) tmp[i][i] = 1, col[i] = i;

  rep(i,0,n) {
    int r = i, c = i;
    rep(j,i,n) rep(k,i,n)
      if (fabs(A[j][k]) > fabs(A[r][c]))
        r = j, c = k;
    if (fabs(A[r][c]) < 1e-12) return i;
    A[i].swap(A[r]); tmp[i].swap(tmp[r]);
    rep(j,0,n)
      swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
    swap(col[i], col[c]);
    double v = A[i][i];
    rep(j,i+1,n) {
      double f = A[j][i] / v;
      A[j][i] = 0;
      rep(k,i+1,n) A[j][k] -= f*A[i][k];
      rep(k,0,n) tmp[j][k] -= f*tmp[i][k];
    }
    rep(j,i+1,n) A[i][j] /= v;
    rep(j,0,n) tmp[i][j] /= v;
    A[i][i] = 1;
  }

  for (int i = n-1; i > 0; --i) rep(j,0,i) {
    double v = A[j][i];
    rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
  }

  rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
  return n;
}
```

Tridiagonal.h

Description: Solves a linear equation system with a tridiagonal matrix with diagonal diag, subdiagonal sub and superdiagonal super, i.e., $x = \text{tridiagonal}(d, p, q, b)$ solves the equation system

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \end{pmatrix} = \begin{pmatrix} d_0 & p_0 & 0 & 0 & \cdots & 0 \\ q_0 & d_1 & p_1 & 0 & \cdots & 0 \\ 0 & q_1 & d_2 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & q_{n-3} & d_{n-2} & p_{n-2} \\ 0 & 0 & \cdots & 0 & q_{n-2} & d_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \end{pmatrix}.$$

The size of diag and b should be the same and super and sub should be one element shorter. T is intended to be double.
This is useful for solving problems on the type

$$a_i = b_i a_{i-1} + c_i a_{i+1} + d_i,\ 1 \le i \le n,$$

where a_0, a_{n+1}, b_i, c_i and d_i are known. a can then be obtained from

$$\{a_i\} = \text{tridiagonal}(\{1,-1,-1,...,-1,1\}, \{0,c_1,c_2,\ldots,c_n\}, \{b_1,b_2,\ldots,b_n,0\}, \{a_0,d_1,d_2,\ldots,d_n,a_{n+1}\}).$$

Usage:
```
int n = 1000000;
vector<double> diag(n,-1), sup(n-1,.5), sub(n-1,.5), b(n,1);
vector<double> x = tridiagonal(diag, super, sub, b);
```
Time: $\mathcal{O}(N)$

14 lines

```
template <class T>
vector<T> tridiagonal(vector<T> diag, const vector<T>& super,
    const vector<T>& sub, vector<T> b) {
  rep(i,0,sz(b)-1) {
    diag[i+1] -= super[i]*sub[i]/diag[i];
    b[i+1] -= b[i]*sub[i]/diag[i];
  }
  for (int i = sz(b); --i > 0;) {
    b[i] /= diag[i];
    b[i-1] -= b[i]*super[i-1];
  }
  b[0] /= diag[0];
  return b;
}
```

FFT.h

Description: Fast Fourier transform. Also includes a function for convolution: `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. a and b should be of roughly equal size. For convolutions of integers, rounding the results of conv works if $(|a| + |b|)\max(a,b) < \sim 10^9$ (in theory maybe 10^6); you may want to use an NTT from the Number Theory chapter instead.
Time: $\mathcal{O}(N \log N)$

<valarray> 29 lines

```
typedef valarray<complex<double> > carray;
void fft(carray& x, carray& roots) {
  int N = sz(x);
  if (N <= 1) return;
  carray even = x[slice(0, N/2, 2)];
  carray odd = x[slice(1, N/2, 2)];
  carray rs = roots[slice(0, N/2, 2)];
  fft(even, rs);
  fft(odd, rs);
  rep(k,0,N/2) {
    auto t = roots[k] * odd[k];
    x[k    ] = even[k] + t;
    x[k+N/2] = even[k] - t;
  }
}

typedef vector<double> vd;
vd conv(const vd& a, const vd& b) {
  int s = sz(a) + sz(b) - 1, L = 32-__builtin_clz(s), n = 1<<L;
  if (s <= 0) return {};
  carray av(n), bv(n), roots(n);
  rep(i,0,n) roots[i] = polar(1.0, -2 * M_PI * i / n);
  copy(all(a), begin(av)); fft(av, roots);
  copy(all(b), begin(bv)); fft(bv, roots);
  roots = roots.apply(conj);
  carray cv = av * bv; fft(cv, roots);
  vd c(s); rep(i,0,s) c[i] = cv[i].real() / n;
  return c;
}
```

Number theory (5)

5.1 Modular arithmetic

ModularArithmetic.h

Description: Operators for modular arithmetic. You need to set `mod` to some number first and then you can use the structure.

"euclid.h" 18 lines

```
const ll mod = 17; // change to something else
struct Mod {
  ll x;
  Mod(ll xx) : x(xx) {}
  Mod operator+(Mod b) { return Mod((x + b.x) % mod); }
  Mod operator-(Mod b) { return Mod((x - b.x + mod) % mod); }
  Mod operator*(Mod b) { return Mod((x * b.x) % mod); }
  Mod operator/(Mod b) { return *this * invert(b); }
  Mod invert(Mod a) {
    ll x, y, g = euclid(a.x, mod, x, y);
    assert(g == 1); return Mod((x + mod) % mod);
  }
  Mod operator^(ll e) {
    if (!e) return Mod(1);
    Mod r = *this ^ (e / 2); r = r * r;
    return e&1 ? *this * r : r;
  }
};
```

ModInverse.h

Description: Pre-computation of modular inverses. Assumes LIM $\le$ mod and that mod is a prime.

3 lines

```
const ll mod = 1000000007, LIM = 200000;
ll* inv = new ll[LIM] - 1; inv[1] = 1;
rep(i,2,LIM) inv[i] = mod - (mod / i) * inv[mod % i] % mod;
```

ModPow.h

6 lines

```
const ll mod = 1000000007; // faster if const
ll modpow(ll a, ll e) {
  if (e == 0) return 1;
  ll x = modpow(a * a % mod, e >> 1);
  return e & 1 ? x * a % mod : x;
}
```

ModSum.h

Description: Sums of mod'ed arithmetic progressions.
`modsum(to, c, k, m)` $= \sum_{i=0}^{to-1} (ki+c)\%m$. `divsum` is similar but for floored division.
Time: $\log(m)$, with a large constant.

21 lines

```
typedef unsigned long long ull;
ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }

ull divsum(ull to, ull c, ull k, ull m) {
  ull res = k / m * sumsq(to) + c / m * to;
  k %= m; c %= m;
  if (k) {
    ull to2 = (to * k + c) / m;
    res += to * to2;
    res -= divsum(to2, m-1 - c, m, k) + to2;
  }
  return res;
}

ll modsum(ull to, ll c, ll k, ll m) {
  c %= m;
  k %= m;
  if (c < 0) c += m;
  if (k < 0) k += m;
  return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
}
```

ModMulLL.h

Description: Calculate $a \cdot b \bmod c$ (or $a^b \bmod c$) for large c.
Time: $\mathcal{O}(64/bits \cdot \log b)$, where $bits = 64 - k$, if we want to deal with k-bit numbers.

19 lines

```
typedef unsigned long long ull;
const int bits = 10;
// if all numbers are less than 2^k, set bits = 64-k
const ull po = 1 << bits;
ull mod_mul(ull a, ull b, ull &c) {
  ull x = a * (b & (po - 1)) % c;
  while ((b >>= bits) > 0) {
    a = (a << bits) % c;
    x += (a * (b & (po - 1))) % c;
  }
  return x % c;
}
ull mod_pow(ull a, ull b, ull mod) {
  if (b == 0) return 1;
  ull res = mod_pow(a, b / 2, mod);
  res = mod_mul(res, res, mod);
  if (b & 1) return mod_mul(res, a, mod);
  return res;
}
```

ModSqrt.h

Description: Tonelli-Shanks algorithm for modular square roots.
Time: $\mathcal{O}\left(\log^2 p\right)$ worst case, often $\mathcal{O}(\log p)$

"ModPow.h" 30 lines

```
ll sqrt(ll a, ll p) {
  a %= p; if (a < 0) a += p;
  if (a == 0) return 0;
  assert(modpow(a, (p-1)/2, p) == 1);
  if (p % 4 == 3) return modpow(a, (p+1)/4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
  ll s = p - 1;
  int r = 0;
  while (s % 2 == 0)
    ++r, s /= 2;
  ll n = 2; // find a non-square mod p
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p);
  ll g = modpow(n, s, p);
  for (;;) {
    ll t = b;
    int m = 0;
    for (; m < r; ++m) {
      if (t == 1) break;
      t = t * t % p;
    }
    if (m == 0) return x;
    ll gs = modpow(g, 1 << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
    r = m;
  }
}
```

5.2 Number theoretic transform

NTT.h

Description: Number theoretic transform. Can be used for convolutions modulo specific nice primes of the form $2^a b+1$, where the convolution result has size at most 2^a. For other primes/integers, use two different primes and combine with CRT. May return negative values.
Time: $\mathcal{O}(N \log N)$

"ModPow.h" 38 lines

```
const ll mod = (119 << 23) + 1, root = 3; // = 998244353
// For p < 2^30 there is also e.g. (5 << 25, 3), (7 << 26, 3),
// (479 << 21, 3) and (483 << 21, 5). The last two are > 10^9.

typedef vector<ll> vl;
void ntt(ll* x, ll* temp, ll* roots, int N, int skip) {
  if (N == 1) return;
  int n2 = N/2;
  ntt(x     , temp, roots, n2, skip*2);
  ntt(x+skip, temp, roots, n2, skip*2);
  rep(i,0,N) temp[i] = x[i*skip];
  rep(i,0,n2) {
    ll s = temp[2*i], t = temp[2*i+1] * roots[skip*i];
    x[skip*i] = (s + t) % mod; x[skip*(i+n2)] = (s - t) % mod;
  }
}
void ntt(vl& x, bool inv = false) {
  ll e = modpow(root, (mod-1) / sz(x));
  if (inv) e = modpow(e, mod-2);
  vl roots(sz(x), 1), temp = roots;
  rep(i,1,sz(x)) roots[i] = roots[i-1] * e % mod;
  ntt(&x[0], &temp[0], &roots[0], sz(x), 1);
}
vl conv(vl a, vl b) {
  int s = sz(a) + sz(b) - 1; if (s <= 0) return {};
  int L = s > 1 ? 32 - __builtin_clz(s - 1) : 0, n = 1 << L;
  if (s <= 200) { // (factor 10 optimization for |a|,|b| = 10)
    vl c(s);
    rep(i,0,sz(a)) rep(j,0,sz(b))
      c[i + j] = (c[i + j] + a[i] * b[j]) % mod;
    return c;
  }
  a.resize(n); ntt(a);
  b.resize(n); ntt(b);
  vl c(n); ll d = modpow(n, mod-2);
  rep(i,0,n) c[i] = a[i] * b[i] % mod * d % mod;
  ntt(c, true); c.resize(s); return c;
}
```

5.3 Primality

eratosthenes.h

Description: Prime sieve for generating all primes up to a certain limit. isprime[i] is true iff i is a prime.
Time: lim=100'000'000 $\approx$ 0.8 s. Runs 30% faster if only odd indices are stored.

11 lines

```
const int MAX_PR = 5000000;
bitset<MAX_PR> isprime;
vi eratosthenes_sieve(int lim) {
  isprime.set(); isprime[0] = isprime[1] = 0;
  for (int i = 4; i < lim; i += 2) isprime[i] = 0;
  for (int i = 3; i*i < lim; i += 2) if (isprime[i])
    for (int j = i*i; j < lim; j += i*2) isprime[j] = 0;
  vi pr;
  rep(i,2,lim) if (isprime[i]) pr.push_back(i);
  return pr;
}
```

MillerRabin.h

Description: Miller-Rabin primality probabilistic test. Probability of failing one iteration is at most 1/4. 15 iterations should be enough for 50-bit numbers.
Time: 15 times the complexity of $a^b \mod c$.

"ModMulLL.h" 16 lines

```
bool prime(ull p) {
  if (p == 2) return true;
  if (p == 1 || p % 2 == 0) return false;
  ull s = p - 1;
  while (s % 2 == 0) s /= 2;
  rep(i,0,15) {
    ull a = rand() % (p - 1) + 1, tmp = s;
    ull mod = mod_pow(a, tmp, p);
    while (tmp != p - 1 && mod != 1 && mod != p - 1) {
      mod = mod_mul(mod, mod, p);
      tmp *= 2;
    }
    if (mod != p - 1 && tmp % 2 == 0) return false;
  }
  return true;
}
```

factor.h

Description: Pollard's rho algorithm. It is a probabilistic factorisation algorithm, whose expected time complexity is good. Before you start using it, run `init(bits)`, where bits is the length of the numbers you use.
Time: Expected running time should be good enough for 50-bit numbers.

"MillerRabin.h", "eratosthenes.h", "euclid.h" 37 lines

```
vector<ull> pr;
ull f(ull a, ull n, ull &has) {
  return (mod_mul(a, a, n) + has) % n;
}
vector<ull> factor(ull d) {
  vector<ull> res;
  for (size_t i = 0; i < pr.size() && pr[i]*pr[i] <= d; i++)
    if (d % pr[i] == 0) {
      while (d % pr[i] == 0) d /= pr[i];
      res.push_back(pr[i]);
    }
  //d is now a product of at most 2 primes.
  if (d > 1) {
    if (prime(d))
      res.push_back(d);
    else while (true) {
      ull has = rand() % 2321 + 47;
      ull x = 2, y = 2, c = 1;
```

```
      for (; c==1; c = gcd((y > x ? y - x : x - y), d)) {
        x = f(x, d, has);
        y = f(f(y, d, has), d, has);
      }
      if (c != d) {
        res.push_back(c); d /= c;
        if (d != c) res.push_back(d);
        break;
      }
    }
  }
  return res;
}
void init(int bits) {//how many bits do we use?
  vi p = eratosthenes_sieve(1 << ((bits + 2) / 3));
  vector<ull> pr(p.size());
  for (size_t i=0; i<pr.size(); i++)
    pr[i] = p[i];
}
```

5.4 Divisibility

euclid.h

Description: Finds the Greatest Common Divisor to the integers a and b. Euclid also finds two integers x and y, such that $ax + by = \gcd(a, b)$. If a and b are coprime, then x is the inverse of $a \pmod b$. 7 lines

```
ll gcd(ll a, ll b) { return __gcd(a, b); }

ll euclid(ll a, ll b, ll &x, ll &y) {
  if (b) { ll d = euclid(b, a % b, y, x);
    return y -= a/b * x, d; }
  return x = 1, y = 0, a;
}
```

Euclid.java

Description: Finds {x, y, d} s.t. ax + by = d = gcd(a, b). 11 lines

```
static BigInteger[] euclid(BigInteger a, BigInteger b) {
  BigInteger x = BigInteger.ONE, yy = x;
  BigInteger y = BigInteger.ZERO, xx = y;
  while (b.signum() != 0) {
    BigInteger q = a.divide(b), t = b;
    b = a.mod(b); a = t;
    t = xx; xx = x.subtract(q.multiply(xx)); x = t;
    t = yy; yy = y.subtract(q.multiply(yy)); y = t;
  }
  return new BigInteger[]{x, y, a};
}
```

5.4.1 Bézout's identity

For $a \neq, b \neq 0$, then $d = gcd(a, b)$ is the smallest positive integer for which there are integer solutions to

$$ax + by = d$$

If (x, y) is one solution, then all solutions are given by

$$\left(x + \frac{kb}{\gcd(a,b)}, y - \frac{ka}{\gcd(a,b)}\right), \quad k \in \mathbb{Z}$$

phiFunction.h

Description: *Euler's totient* or *Euler's phi* function is defined as $\phi(n) :=$ # of positive integers $\leq n$ that are coprime with n. The *cototient* is $n - \phi(n)$. $\phi(1) = 1$, p prime $\Rightarrow \phi(p^k) = (p-1)p^{k-1}$, m, n coprime $\Rightarrow \phi(mn) = \phi(m)\phi(n)$. If $n = p_1^{k_1}p_2^{k_2}...p_r^{k_r}$ then $\phi(n) = (p_1-1)p_1^{k_1-1}...(p_r-1)p_r^{k_r-1}$. $\phi(n) = n \cdot \prod_{p|n}(1 - 1/p)$.
$\sum_{d|n} \phi(d) = n$, $\sum_{1 \leq k \leq n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$
Euler's thm: a, n coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod n$.
Fermat's little thm: p prime $\Rightarrow a^{p-1} \equiv 1 \pmod p$ $\forall a$. 10 lines

```
const int LIM = 5000000;
int phi[LIM];

void calculatePhi() {
  rep(i,0,LIM) phi[i] = i&1 ? i : i/2;
  for(int i = 3; i < LIM; i += 2)
    if(phi[i] == i)
      for(int j = i; j < LIM; j += i)
        (phi[j] /= i) *= i-1;
}
```

5.5 Chinese remainder theorem

chinese.h

Description: Chinese Remainder Theorem.
chinese(a, m, b, n) returns a number x, such that $x \equiv a \pmod m$ and $x \equiv b \pmod n$. For not coprime n, m, use chinese_common. Note that all numbers must be less than 2^{31} if you have Z = unsigned long long.
Time: $\log(m + n)$
"euclid.h" 13 lines

```
template <class Z> Z chinese(Z a, Z m, Z b, Z n) {
  Z x, y; euclid(m, n, x, y);
  Z ret = a * (y + m) % m * n + b * (x + n) % n * m;
  if (ret >= m * n) ret -= m * n;
  return ret;
}

template <class Z> Z chinese_common(Z a, Z m, Z b, Z n) {
  Z d = gcd(m, n);
  if (((b -= a) %= n) < 0) b += n;
  if (b % d) return -1; // No solution
  return d * chinese(Z(0), m/d, b/d, n/d) + a;
}
```

5.6 Pythagorean Triples

The Pythagorean triples are uniquely generated by

$$a = k \cdot (m^2 - n^2), \;\; b = k \cdot (2mn), \;\; c = k \cdot (m^2 + n^2),$$

with $m > n > 0$, $k > 0$, $m \perp n$, and either m or n even.

5.7 Primes

$p = 962592769$ is such that $2^{21} \mid p - 1$, which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). There are 78498 primes less than 1 000 000.

Primitive roots exist modulo any prime power p^a, except for $p = 2, a > 2$, and there are $\phi(\phi(p^a))$ many. For $p = 2, a > 2$, the group $\mathbb{Z}_{2^a}^\times$ is instead isomorphic to $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

5.8 Estimates

$\sum_{d|n} d = O(n \log \log n)$.

The number of divisors of n is at most around 100 for $n < 5e4$, 500 for $n < 1e7$, 2000 for $n < 1e10$, 200 000 for $n < 1e19$.

Combinatorial (6)

6.1 The Twelvefold Way

Counts the # of functions $f : N \to K$, $|N| = n$, $|K| = k$. The elements in N and K can be distinguishable or indistinguishable, while f can be injective (one-to-one) of surjective (onto).

N	K	none	injective	surjective
dist	dist	k^n	$\frac{k!}{(k-n)!}$	$k!S(n,k)$
indist	dist	$\binom{n+k-1}{n}$	$\binom{k}{n}$	$\binom{n-1}{n-k}$
dist	indist	$\sum_{t=0}^{k} S(n,t)$	$[n \leq k]$	$S(n,k)$
indist	indist	$\sum_{t=1}^{k} p(n,t)$	$[n \leq k]$	$p(n,k)$

Here, $S(n, k)$ is the Stirling number of the second kind, and $p(n, k)$ is the partition number.

6.2 Permutations

6.2.1 Factorial

n	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800
n	11	12	13	14	15	16	17			
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14			
n	20	25	30	40	50	100	150	171		
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX		

intperm.h

Description: Permutations to/from integers. The bijection is order preserving.
Time: $\mathcal{O}(n^2)$ 20 lines

```
int factorial[] = {1, 1, 2, 6, 24, 120, 720, 5040}; // etc.
template <class Z, class It>
void perm_to_int(Z& val, It begin, It end) {
  int x = 0, n = 0;
  for (It i = begin; i != end; ++i, ++n)
    if (*i < *begin) ++x;
  if (n > 2) perm_to_int<Z>(val, ++begin, end);
  else val = 0;
```

```
  val += factorial[n-1]*x;
}
/* range [begin, end) does not have to be sorted. */
template <class Z, class It>
void int_to_perm(Z val, It begin, It end) {
  Z fac = factorial[end - begin - 1];
  // Note that the division result will fit in an integer!
  int x = val / fac;
  nth_element(begin, begin + x, end);
  swap(*begin, *(begin + x));
  if (end - begin > 2) int_to_perm(val % fac, ++begin, end);
}
```

6.2.2 Cycles

Let the number of n-permutations whose cycle lengths all belong to the set S be denoted by $g_S(n)$. Then

$$\sum_{n=0}^{\infty} g_S(n)\frac{x^n}{n!} = \exp\left(\sum_{n\in S}\frac{x^n}{n}\right)$$

6.2.3 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1)+D(n-2)) = nD(n-1)+(-1)^n = \left\lfloor \frac{n!}{e} \right\rceil$$

derangements.h

Description: Generates the i:th derangement of S_n (in lexicographical order). 38 lines

```
template <class T, int N>
struct derangements {
  T dgen[N][N], choose[N][N], fac[N];
  derangements() {
    fac[0] = choose[0][0] = 1;
    memset(dgen, 0, sizeof(dgen));
    rep(m,1,N) {
      fac[m] = fac[m-1] * m;
      choose[m][0] = choose[m][m] = 1;
      rep(k,1,m)
        choose[m][k] = choose[m-1][k-1] + choose[m-1][k];
    }
  }
  T DGen(int n, int k) {
    T ans = 0;
    if (dgen[n][k]) return dgen[n][k];
    rep(i,0,k+1)
      ans += (i&1?-1:1) * choose[k][i] * fac[n-i];
    return dgen[n][k] = ans;
  }
  void generate(int n, T idx, int *res) {
    int vals[N];
    rep(i,0,n) vals[i] = i;
    rep(i,0,n) {
      int j, k = 0, m = n - i;
      rep(j,0,m) if (vals[j] > i) ++k;
      rep(j,0,m) {
        T p = 0;
        if (vals[j] > i) p = DGen(m-1, k-1);
        else if (vals[j] < i) p = DGen(m-1, k);
        if (idx <= p) break;
        idx -= p;
      }
      res[i] = vals[j];
      memmove(vals + j, vals + j + 1, sizeof(int)*(m-j-1));
    }
  }
};
```

6.2.4 Involutions

An involution is a permutation with maximum cycle length 2, and it is its own inverse.

$$a(n) = a(n-1) + (n-1)a(n-2)$$

$$a(0) = a(1) = 1$$

1, 1, 2, 4, 10, 26, 76, 232, 764, 2620, 9496, 35696, 140152

6.2.5 Stirling numbers of the first kind

$$s(n,k) = (-1)^{n-k}c(n,k)$$

$c(n,k)$ is the unsigned Stirling numbers of the first kind, and they count the number of permutations on n items with k cycles.

$$s(n,k) = s(n-1,k-1) - (n-1)s(n-1,k)$$

$$s(0,0) = 1, s(n,0) = s(0,n) = 0$$

$$c(n,k) = c(n-1,k-1) + (n-1)c(n-1,k)$$

$$c(0,0) = 1, c(n,0) = c(0,n) = 0$$

6.2.6 Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \geq j$, k j:s s.t. $\pi(j) > j$.

$$E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$$

$$E(n,0) = E(n,n-1) = 1$$

$$E(n,k) = \sum_{j=0}^{k}(-1)^j\binom{n+1}{j}(k+1-j)^n$$

6.2.7 Burnside's lemma

Given a group G of symmetries and a set X, the number of elements of X *up to symmetry* equals

$$\frac{1}{|G|}\sum_{g\in G}|X^g|,$$

where X^g are the elements fixed by g ($g.x = x$).

If $f(n)$ counts "configurations" (of some sort) of length n, we can ignore rotational symmetry using $G = \mathbb{Z}_n$ to get

$$g(n) = \frac{1}{n}\sum_{k=0}^{n-1} f(\gcd(n,k)) = \frac{1}{n}\sum_{k|n} f(k)\phi(n/k).$$

6.3 Partitions and subsets

6.3.1 Partition function

Partitions of n with exactly k parts, $p(n,k)$, i.e., writing n as a sum of k positive integers, disregarding the order of the summands.

$$p(n,k) = p(n-1,k-1) + p(n-k,k)$$

$$p(0,0) = p(1,n) = p(n,n) = p(n,n-1) = 1$$

For partitions with any number of parts, $p(n)$ obeys

$$p(0) = 1,\ p(n) = \sum_{k\in\mathbb{Z}\setminus\{0\}}(-1)^{k+1}p(n-k(3k-1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

n	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	∼2e5	∼2e8

6.3.2 Binomials

binomial.h
Description: The number of k-element subsets of an n-element set, $\binom{n}{k} = \frac{n!}{k!(n-k)!}$
Time: $\mathcal{O}(\min(k, n-k))$
6 lines

```
ll choose(int n, int k) {
  ll c = 1, to = min(k, n-k);
  if (to < 0) return 0;
  rep(i,0,to) c = c * (n - i) / (i + 1);
  return c;
}
```

binomialModPrime.h
Description: Lucas' thm: Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + ... + n_1 p + n_0$ and $m = m_k p^k + ... + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod{p}$. fact and invfact must hold pre-computed factorials / inverse factorials, e.g. from ModInverse.h.
Time: $\mathcal{O}(\log_p n)$
10 lines

```
ll chooseModP(ll n, ll m, int p, vi& fact, vi& invfact) {
  ll c = 1;
  while (n || m) {
    ll a = n % p, b = m % p;
    if (a < b) return 0;
    c = c * fact[a] % p * invfact[b] % p * invfact[a - b] % p;
    n /= p; m /= p;
  }
  return c;
}
```

RollingBinomial.h
Description: $\binom{n}{k} \pmod{m}$ in time proportional to the difference between (n, k) and the previous (n, k).
14 lines

```
const ll mod = 1000000007;
vector<ll> invs; // precomputed up to max n, inclusively
struct Bin {
  int N = 0, K = 0;  ll r = 1;
  void m(ll a, ll b) { r = r * a % mod * invs[b] % mod; }
  ll choose(int n, int k) {
    if (k > n || k < 0) return 0;
    while (N < n) ++N, m(N, N-K);
    while (K < k) ++K, m(N-K+1, K);
    while (K > k) m(K, N-K+1), --K;
    while (N > n) m(N-K, N), --N;
    return r;
  }
};
```

multinomial.h
Description: $\binom{\sum k_i}{k_1, k_2, \ldots, k_n} = \frac{(\sum k_i)!}{k_1! k_2! \ldots k_n!}$
Time: $\mathcal{O}((\sum k_i) - k_1)$
6 lines

```
ll multinomial(vi& v) {
  ll c = 1, m = v.empty() ? 1 : v[0];
  rep(i,1,sz(v)) rep(j,0,v[i])
    c = c * ++m / (j+1);
  return c;
}
```

6.3.3 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

$$S(n,k) = S(n-1,k-1) + kS(n-1,k)$$

$$S(n,1) = S(n,n) = 1$$

$$S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

6.3.4 Bell numbers

Total number of partitions of n distinct elements.

$$B(n) = \sum_{k=1}^{n}\binom{n-1}{k-1}B(n-k) = \sum_{k=1}^{n}S(n,k)$$

$$B(0) = B(1) = 1$$

The first are 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597. For a prime p

$$B(p^m + n) \equiv mB(n) + B(n+1) \pmod{p}$$

6.3.5 Triangles

Given rods of length $1, \ldots, n$,

$$T(n) = \frac{1}{24}\begin{cases} n(n-2)(2n-5) & n \text{ even} \\ (n-1)(n-3)(2n-1) & n \text{ odd} \end{cases}$$

is the number of distinct triangles (positive are) that can be constructed, i.e., the # of 3-subsets of $[n]$ s.t. $x \le y \le z$ and $z \ne x + y$.

6.4 General purpose numbers

6.4.1 Catalan numbers

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_{n+1} = \frac{2(2n+1)}{n+2}C_n$$

$$C_0 = 1, C_{n+1} = \sum C_i C_{n-i}$$

First few are 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900.

- # of monotonic lattice paths of a $n \times n$-grid which do not pass above the diagonal.
- # of expressions containing n pairs of parenthesis which are correctly matched.
- # of full binary trees with with $n+1$ leaves (0 or 2 children).
- # of non-isomorphic ordered trees with $n+1$ vertices.
- # of ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- # of permutations of $[n]$ with no three-term increasing subsequence.

6.4.2 Super Catalan numbers

The number of monotonic lattice paths of a $n \times n$-grid that do not touch the diagonal.

$$S(n) = \frac{3(2n-3)S(n-1) - (n-3)S(n-2)}{n}$$

$$S(1) = S(2) = 1$$

1, 1, 3, 11, 45, 197, 903, 4279, 20793, 103049, 518859

6.4.3 Motzkin numbers

Number of ways of drawing any number of nonintersecting chords among n points on a circle. Number of lattice paths from $(0,0)$ to $(n,0)$ never going below the x-axis, using only steps NE, E, SE.

$$M(n) = \frac{3(n-1)M(n-2) + (2n+1)M(n-1)}{n+2}$$

$$M(0) = M(1) = 1$$

1, 1, 2, 4, 9, 21, 51, 127, 323, 835, 2188, 5798, 15511, 41835, 113634

6.4.4 Narayana numbers

Number of lattice paths from $(0,0)$ to $(2n,0)$ never going below the x-axis, using only steps NE and SE, and with k peaks.

$$N(n,k) = \frac{1}{n}\binom{n}{k}\binom{n}{k-1}$$

$$N(n,1) = N(n,n) = 1$$

$$\sum_{k=1}^{n} N(n,k) = C_n$$

1, 1, 1, 1, 3, 1, 1, 6, 6, 1, 1, 10, 20, 10, 1, 1, 15, 50

6.4.5 Schröder numbers

Number of lattice paths from $(0,0)$ to (n,n) using only steps N,NE,E, never going above the diagonal. Number of lattice paths from $(0,0)$ to $(2n,0)$ using only steps NE, SE and double east EE, never going below the x-axis. Twice the Super Catalan number, except for the first term. 1, 2, 6, 22, 90, 394, 1806, 8558, 41586, 206098

Graph (7)

7.1 Fundamentals

bellmanFord.h

Description: Calculates shortest path in a graph that might have negative edge distances. Propagates negative infinity distances (sets dist = -inf), and returns true if there is some negative cycle. Unreachable nodes get dist = inf.

Time: $\mathcal{O}(EV)$

27 lines

```
typedef ll T; // or whatever
struct Edge { int src, dest; T weight; };
struct Node { T dist; int prev; };
struct Graph { vector<Node> nodes; vector<Edge> edges; };

const T inf = numeric_limits<T>::max();
bool bellmanFord2(Graph& g, int start_node) {
  trav(n, g.nodes) { n.dist = inf; n.prev = -1; }
  g.nodes[start_node].dist = 0;

  rep(i,0,sz(g.nodes)) trav(e, g.edges) {
    Node& cur = g.nodes[e.src];
    Node& dest = g.nodes[e.dest];
    if (cur.dist == inf) continue;
    T ndist = cur.dist + (cur.dist == -inf ? 0 : e.weight);
    if (ndist < dest.dist) {
      dest.prev = e.src;
      dest.dist = (i >= sz(g.nodes)-1 ? -inf : ndist);
    }
  }
  bool ret = 0;
  rep(i,0,sz(g.nodes)) trav(e, g.edges) {
    if (g.nodes[e.src].dist == -inf)
      g.nodes[e.dest].dist = -inf, ret = 1;
  }
  return ret;
}
```

FloydWarshall.h

Description: Calculates all-pairs shortest path in a directed graph that might have negative edge distances. Input is an distance matrix m, where $m[i][j] =$ `inf` if i and j are not adjacent. As output, $m[i][j]$ is set to the shortest distance between i and j, `inf` if no path, or `-inf` if the path goes through a negative-weight cycle.

Time: $\mathcal{O}(N^3)$

12 lines

```
const ll inf = 1LL << 62;
void floydWarshall(vector<vector<ll>>& m) {
  int n = sz(m);
  rep(i,0,n) m[i][i] = min(m[i][i], {});
  rep(k,0,n) rep(i,0,n) rep(j,0,n)
    if (m[i][k] != inf && m[k][j] != inf) {
      auto newDist = max(m[i][k] + m[k][j], -inf);
      m[i][j] = min(m[i][j], newDist);
    }
  rep(k,0,n) if (m[k][k] < 0) rep(i,0,n) rep(j,0,n)
    if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf;
}
```

TopoSort.h

Description: Topological sorting. Given is an oriented graph. Output is an ordering of vertices (array idx), such that there are edges only from left to right. The function returns false if there is a cycle in the graph.

Time: $\mathcal{O}(|V| + |E|)$

18 lines

```
template <class E, class I>
bool topo_sort(const E &edges, I &idx, int n) {
  vi indeg(n);
  rep(i,0,n)
    trav(e, edges[i])
      indeg[e]++;
  queue<int> q; // use priority queue for lexic. smallest ans.
  rep(i,0,n) if (indeg[i] == 0) q.push(-i);
  int nr = 0;
  while (q.size() > 0) {
    int i = -q.front(); // top() for priority queue
    idx[i] = ++nr;
    q.pop();
    trav(e, edges[i])
      if (--indeg[e] == 0) q.push(-e);
  }
  return nr == n;
}
```

7.2 Euler walk

EulerWalk.h

Description: Eulerian undirected/directed path/cycle algorithm. Returns a list of nodes in the Eulerian path/cycle with src at both start and end, or empty list if no cycle/path exists. To get edge indices back, also put it->second in s (and then ret).

Time: $\mathcal{O}(E)$ where E is the number of edges.

27 lines

```
struct V {
  vector<pii> outs; // (dest, edge index)
  int nins = 0;
};

vi euler_walk(vector<V>& nodes, int nedges, int src=0) {
  int c = 0;
  trav(n, nodes) c += abs(n.nins - sz(n.outs));
  if (c > 2) return {};
  vector<vector<pii>::iterator> its;
  trav(n, nodes)
    its.push_back(n.outs.begin());
  vector<bool> eu(nedges);
  vi ret, s = {src};
  while(!s.empty()) {
    int x = s.back();
    auto& it = its[x], end = nodes[x].outs.end();
    while(it != end && eu[it->second]) ++it;
    if(it == end) { ret.push_back(x); s.pop_back(); }
    else { s.push_back(it->first); eu[it->second] = true; }
  }
  if(sz(ret) != nedges+1)
    ret.clear(); // No Eulerian cycles/paths.
  // else, non-cycle if ret.front() != ret.back()
  reverse(all(ret));
  return ret;
}
```

7.3 Network flow

PushRelabel.h

Description: Push-relabel using the highest label selection rule and the gap heuristic. Quite fast in practice. To obtain the actual flow, look at positive values only.

Time: $\mathcal{O}\left(V^2\sqrt{E}\right)$

51 lines

```
typedef ll Flow;
struct Edge {
  int dest, back;
  Flow f, c;
};

struct PushRelabel {
  vector<vector<Edge>> g;
  vector<Flow> ec;
  vector<Edge*> cur;
  vector<vi> hs; vi H;
  PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}

  void add_edge(int s, int t, Flow cap, Flow rcap=0) {
    if (s == t) return;
    Edge a = {t, sz(g[t]), 0, cap};
    Edge b = {s, sz(g[s]), 0, rcap};
    g[s].push_back(a);
    g[t].push_back(b);
  }

  void add_flow(Edge& e, Flow f) {
    Edge &back = g[e.dest][e.back];
    if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
    e.f += f; e.c -= f; ec[e.dest] += f;
    back.f -= f; back.c += f; ec[back.dest] -= f;
  }
  Flow maxflow(int s, int t) {
    int v = sz(g); H[s] = v; ec[t] = 1;
    vi co(2*v); co[0] = v-1;
    rep(i,0,v) cur[i] = g[i].data();
    trav(e, g[s]) add_flow(e, e.c);

    for (int hi = 0;;) {
      while (hs[hi].empty()) if (!hi--) return -ec[s];
      int u = hs[hi].back(); hs[hi].pop_back();
      while (ec[u] > 0)  // discharge u
        if (cur[u] == g[u].data() + sz(g[u])) {
          H[u] = 1e9;
          trav(e, g[u]) if (e.c && H[u] > H[e.dest]+1)
            H[u] = H[e.dest]+1, cur[u] = &e;
          if (++co[H[u]], !--co[hi] && hi < v)
            rep(i,0,v) if (hi < H[i] && H[i] < v)
              --co[H[i]], H[i] = v + 1;
          hi = H[u];
        } else if (cur[u]->c && H[u] == H[cur[u]->dest]+1)
          add_flow(*cur[u], min(ec[u], cur[u]->c));
        else ++cur[u];
    }
  }
};
```

MinCostMaxFlow.h

Description: Min-cost max-flow. cap[i][j] != cap[j][i] is allowed; double edges are not. If costs can be negative, call setpi before maxflow, but note that negative cost cycles are not allowed (that's NP-hard). To obtain the actual flow, look at positive values only.
Time: Approximately $\mathcal{O}\left(E^2\right)$

81 lines

```
#include <bits/extc++.h>

const ll INF = numeric_limits<ll>::max() / 4;
typedef vector<ll> VL;

struct MCMF {
  int N;
  vector<vi> ed, red;
  vector<VL> cap, flow, cost;
  vi seen;
  VL dist, pi;
  vector<pii> par;

  MCMF(int N) :
    N(N), ed(N), red(N), cap(N, VL(N)), flow(cap), cost(cap),
    seen(N), dist(N), pi(N), par(N) {}

  void addEdge(int from, int to, ll cap, ll cost) {
    this->cap[from][to] = cap;
    this->cost[from][to] = cost;
    ed[from].push_back(to);
    red[to].push_back(from);
  }

  void path(int s) {
    fill(all(seen), 0);
    fill(all(dist), INF);
    dist[s] = 0; ll di;

    __gnu_pbds::priority_queue<pair<ll, int>> q;
    vector<decltype(q)::point_iterator> its(N);
    q.push({0, s});

    auto relax = [&](int i, ll cap, ll cost, int dir) {
      ll val = di - pi[i] + cost;
      if (cap && val < dist[i]) {
        dist[i] = val;
        par[i] = {s, dir};
        if (its[i] == q.end()) its[i] = q.push({-dist[i], i});
        else q.modify(its[i], {-dist[i], i});
      }
    };

    while (!q.empty()) {
      s = q.top().second; q.pop();
      seen[s] = 1; di = dist[s] + pi[s];
      trav(i, ed[s]) if (!seen[i])
        relax(i, cap[s][i] - flow[s][i], cost[s][i], 1);
      trav(i, red[s]) if (!seen[i])
        relax(i, flow[i][s], -cost[i][s], 0);
    }
    rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
  }

  pair<ll, ll> maxflow(int s, int t) {
    ll totflow = 0, totcost = 0;
    while (path(s), seen[t]) {
      ll fl = INF;
      for (int p,r,x = t; tie(p,r) = par[x], x != s; x = p)
        fl = min(fl, r ? cap[p][x] - flow[p][x] : flow[x][p]);
      totflow += fl;
      for (int p,r,x = t; tie(p,r) = par[x], x != s; x = p)
        if (r) flow[p][x] += fl;
        else flow[x][p] -= fl;
    }
    rep(i,0,N) rep(j,0,N) totcost += cost[i][j] * flow[i][j];
    return {totflow, totcost};
  }

  // If some costs can be negative, call this before maxflow:
  void setpi(int s) { // (otherwise, leave this out)
    fill(all(pi), INF); pi[s] = 0;
    int it = N, ch = 1; ll v;
    while (ch-- && it--)
      rep(i,0,N) if (pi[i] != INF)
        trav(to, ed[i]) if (cap[i][to])
          if ((v = pi[i] + cost[i][to]) < pi[to])
            pi[to] = v, ch = 1;
    assert(it >= 0); // negative cost cycle
  }
};
```

EdmondsKarp.h

Description: Flow algorithm with guaranteed complexity $O(VE^2)$. To get edge flow values, compare capacities before and after, and take the positive values only.

35 lines

```
template<class T> T edmondsKarp(vector<unordered_map<int, T>>&
     graph, int source, int sink) {
  assert(source != sink);
  T flow = 0;
  vi par(sz(graph)), q = par;

  for (;;) {
    fill(all(par), -1);
    par[source] = 0;
    int ptr = 1;
    q[0] = source;

    rep(i,0,ptr) {
      int x = q[i];
      trav(e, graph[x]) {
        if (par[e.first] == -1 && e.second > 0) {
          par[e.first] = x;
          q[ptr++] = e.first;
          if (e.first == sink) goto out;
        }
      }
    }
    return flow;
out:
    T inc = numeric_limits<T>::max();
    for (int y = sink; y != source; y = par[y])
      inc = min(inc, graph[par[y]][y]);

    flow += inc;
    for (int y = sink; y != source; y = par[y]) {
      int p = par[y];
      if ((graph[p][y] -= inc) <= 0) graph[p].erase(y);
      graph[y][p] += inc;
    }
  }
}
```

MinCut.h

Description: After running max-flow, the left side of a min-cut from s to t is given by all vertices reachable from s, only traversing edges with positive residual capacity.

1 lines

GlobalMinCut.h

Description: Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.
Time: $\mathcal{O}\left(V^3\right)$

31 lines

```
pair<int, vi> GetMinCut(vector<vi>& weights) {
  int N = sz(weights);
  vi used(N), cut, best_cut;
  int best_weight = -1;

  for (int phase = N-1; phase >= 0; phase--) {
    vi w = weights[0], added = used;
    int prev, k = 0;
    rep(i,0,phase){
      prev = k;
      k = -1;
      rep(j,1,N)
        if (!added[j] && (k == -1 || w[j] > w[k])) k = j;
      if (i == phase-1) {
        rep(j,0,N) weights[prev][j] += weights[k][j];
        rep(j,0,N) weights[j][prev] = weights[prev][j];
        used[k] = true;
        cut.push_back(k);
        if (best_weight == -1 || w[k] < best_weight) {
          best_cut = cut;
          best_weight = w[k];
        }
      } else {
        rep(j,0,N)
          w[j] += weights[k][j];
        added[k] = true;
      }
    }
  }
  return {best_weight, best_cut};
}
```

7.4 Matching

hopcroftKarp.h

Description: Find a maximum matching in a bipartite graph.
Usage: `vi ba(m, -1); hopcroftKarp(g, ba);`
Time: $\mathcal{O}\left(\sqrt{V}E\right)$

48 lines

```
bool dfs(int a, int layer, const vector<vi>& g, vi& btoa,
      vi& A, vi& B) {
  if (A[a] != layer) return 0;
  A[a] = -1;
  trav(b, g[a]) if (B[b] == layer + 1) {
    B[b] = -1;
    if (btoa[b] == -1 || dfs(btoa[b], layer+2, g, btoa, A, B))
      return btoa[b] = a, 1;
  }
  return 0;
}

int hopcroftKarp(const vector<vi>& g, vi& btoa) {
  int res = 0;
  vi A(g.size()), B(btoa.size()), cur, next;
  for (;;) {
    fill(all(A), 0);
    fill(all(B), -1);

    cur.clear();
    trav(a, btoa) if(a != -1) A[a] = -1;
    rep(a,0,sz(g)) if(A[a] == 0) cur.push_back(a);

    for (int lay = 1;; lay += 2) {
```

```
      bool islast = 0;
      next.clear();
      trav(a, cur) trav(b, g[a]) {
        if (btoa[b] == -1) {
          B[b] = lay;
          islast = 1;
        }
        else if (btoa[b] != a && B[b] == -1) {
          B[b] = lay;
          next.push_back(btoa[b]);
        }
      }
      if (islast) break;
      if (next.empty()) return res;
      trav(a, next) A[a] = lay+1;
      cur.swap(next);
    }

    rep(a,0,sz(g)) {
      if(dfs(a, 0, g, btoa, A, B))
        ++res;
    }
  }
}
```

DFSMatching.h

Description: This is a simple matching algorithm but should be just fine in most cases. Graph g should be a list of neighbours of the left partition. n is the size of the left partition and m is the size of the right partition. If you want to get the matched pairs, $match[i]$ contains match for vertex i on the right side or -1 if it's not matched.
Time: $\mathcal{O}(EV)$ where E is the number of edges and V is the number of vertices.

24 lines

```
vi match;
vector<bool> seen;
bool find(int j, const vector<vi>& g) {
  if (match[j] == -1) return 1;
  seen[j] = 1; int di = match[j];
  trav(e, g[di])
    if (!seen[e] && find(e, g)) {
      match[e] = di;
      return 1;
    }
  return 0;
}
int dfs_matching(const vector<vi>& g, int n, int m) {
  match.assign(m, -1);
  rep(i,0,n) {
    seen.assign(m, 0);
    trav(j,g[i])
      if (find(j, g)) {
        match[j] = i;
        break;
      }
  }
  return m - (int)count(all(match), -1);
}
```

WeightedMatching.h

Description: Min cost bipartite matching. Negate costs for max cost.
Time: $\mathcal{O}(N^3)$

79 lines

```
typedef vector<double> vd;
bool zero(double x) { return fabs(x) < 1e-10; }
double MinCostMatching(const vector<vd>& cost, vi& L, vi& R) {
  int n = sz(cost), mated = 0;
  vd dist(n), u(n), v(n);
  vi dad(n), seen(n);

  rep(i,0,n) {
    u[i] = cost[i][0];
    rep(j,1,n) u[i] = min(u[i], cost[i][j]);
  }
  rep(j,0,n) {
    v[j] = cost[0][j] - u[0];
    rep(i,1,n) v[j] = min(v[j], cost[i][j] - u[i]);
  }

  L = R = vi(n, -1);
  rep(i,0,n) rep(j,0,n) {
    if (R[j] != -1) continue;
    if (zero(cost[i][j] - u[i] - v[j])) {
      L[i] = j;
      R[j] = i;
      mated++;
      break;
    }
  }

  for (; mated < n; mated++) { // until solution is feasible
    int s = 0;
    while (L[s] != -1) s++;
    fill(all(dad), -1);
    fill(all(seen), 0);
    rep(k,0,n)
      dist[k] = cost[s][k] - u[s] - v[k];

    int j = 0;
    for (;;) {
      j = -1;
      rep(k,0,n){
        if (seen[k]) continue;
        if (j == -1 || dist[k] < dist[j]) j = k;
      }
      seen[j] = 1;
      int i = R[j];
      if (i == -1) break;
      rep(k,0,n) {
        if (seen[k]) continue;
        auto new_dist = dist[j] + cost[i][k] - u[i] - v[k];
        if (dist[k] > new_dist) {
          dist[k] = new_dist;
          dad[k] = j;
        }
      }
    }

    rep(k,0,n) {
      if (k == j || !seen[k]) continue;
      auto w = dist[k] - dist[j];
      v[k] += w, u[R[k]] -= w;
    }
    u[s] += dist[j];

    while (dad[j] >= 0) {
      int d = dad[j];
      R[j] = R[d];
      L[R[j]] = j;
      j = d;
    }
    R[j] = s;
    L[s] = j;
  }
  auto value = vd(1)[0];
  rep(i,0,n) value += cost[i][L[i]];
  return value;
}
```

GeneralMatching.h

Description: Matching for general graphs. Fails with probability N/mod.
Time: $\mathcal{O}(N^3)$

"../numerical/MatrixInverse-mod.h" 40 lines

```
vector<pii> generalMatching(int N, vector<pii>& ed) {
  vector<vector<ll>> mat(N, vector<ll>(N)), A;
  trav(pa, ed) {
    int a = pa.first, b = pa.second, r = rand() % mod;
    mat[a][b] = r, mat[b][a] = (mod - r) % mod;
  }

  int r = matInv(A = mat), M = 2*N - r, fi, fj;
  assert(r % 2 == 0);

  if (M != N) do {
    mat.resize(M, vector<ll>(M));
    rep(i,0,N) {
      mat[i].resize(M);
      rep(j,N,M) {
        int r = rand() % mod;
        mat[i][j] = r, mat[j][i] = (mod - r) % mod;
      }
    }
  } while (matInv(A = mat) != M);

  vi has(M, 1); vector<pii> ret;
  rep(it,0,M/2) {
    rep(i,0,M) if (has[i])
      rep(j,i+1,M) if (A[i][j] && mat[i][j]) {
        fi = i; fj = j; goto done;
    } assert(0); done:
    if (fj < N) ret.emplace_back(fi, fj);
    has[fi] = has[fj] = 0;
    rep(sw,0,2) {
      ll a = modpow(A[fi][fj], mod-2);
      rep(i,0,M) if (has[i] && A[i][fj]) {
        ll b = A[i][fj] * a % mod;
        rep(j,0,M) A[i][j] = (A[i][j] - A[fi][j] * b) % mod;
      }
      swap(fi,fj);
    }
  }
  return ret;
}
```

7.5 Minimum vertex cover

MinimumVertexCover.h

Description: Finds a minimum vertex cover in a bipartite graph. The size is the same as the size of a maximum matching, and the complement is an independent set.

"DFSMatching.h" 20 lines

```
vi cover(vector<vi>& g, int n, int m) {
  int res = dfs_matching(g, n, m);
  seen.assign(m, false);
  vector<bool> lfound(n, true);
  trav(it, match) if (it != -1) lfound[it] = false;
  vi q, cover;
  rep(i,0,n) if (lfound[i]) q.push_back(i);
  while (!q.empty()) {
    int i = q.back(); q.pop_back();
    lfound[i] = 1;
    trav(e, g[i]) if (!seen[e] && match[e] != -1) {
      seen[e] = true;
      q.push_back(match[e]);
    }
  }
```

```
  rep(i,0,n) if (!lfound[i]) cover.push_back(i);
  rep(i,0,m) if (seen[i]) cover.push_back(n+i);
  assert(sz(cover) == res);
  return cover;
}
```

7.6 DFS algorithms

SCC.h

Description: Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.
Usage: `scc(graph, [&](vi& v) { ... })` visits all components in reverse topological order. `comp[i]` holds the component index of a node (a component only has edges to components with lower index). `ncomps` will contain the number of components.
Time: $\mathcal{O}(E+V)$

24 lines

```
vi val, comp, z, cont;
int Time, ncomps;
template<class G, class F> int dfs(int j, G& g, F f) {
  int low = val[j] = ++Time, x; z.push_back(j);
  trav(e,g[j]) if (comp[e] < 0)
    low = min(low, val[e] ?: dfs(e,g,f));

  if (low == val[j]) {
    do {
      x = z.back(); z.pop_back();
      comp[x] = ncomps;
      cont.push_back(x);
    } while (x != j);
    f(cont); cont.clear();
    ncomps++;
  }
  return val[j] = low;
}
template<class G, class F> void scc(G& g, F f) {
  int n = sz(g);
  val.assign(n, 0); comp.assign(n, -1);
  Time = ncomps = 0;
  rep(i,0,n) if (comp[i] < 0) dfs(i, g, f);
}
```

BiconnectedComponents.h

Description: Finds all biconnected components in an undirected graph, and runs a callback for the edges in each. In a biconnected component there are at least two distinct paths between any two nodes. Note that a node can be in several components. An edge which is not in a component is a bridge, i.e., not part of any cycle.
Usage: `int eid = 0; ed.resize(N);`
`for each edge (a,b) {`
`ed[a].emplace_back(b, eid);`
`ed[b].emplace_back(a, eid++); }`
`bicomps([&](const vi& edgelist) {...});`
Time: $\mathcal{O}(E+V)$

34 lines

```
vi num, st;
vector<vector<pii>> ed;
int Time;
template<class F>
int dfs(int at, int par, F f) {
  int me = num[at] = ++Time, e, y, top = me;
  trav(pa, ed[at]) if (pa.second != par) {
    tie(y, e) = pa;
    if (num[y]) {
      top = min(top, num[y]);
      if (num[y] < me)
        st.push_back(e);
    } else {
      int si = sz(st);
      int up = dfs(y, e, f);
      top = min(top, up);
      if (up == me) {
        st.push_back(e);
        f(vi(st.begin() + si, st.end()));
        st.resize(si);
      }
      else if (up < me)
        st.push_back(e);
      // else e is a bridge
    }
  }
  return top;
}

template<class F>
void bicomps(F f) {
  num.assign(sz(ed), 0);
  rep(i,0,sz(ed)) if (!num[i]) dfs(i, -1, f);
}
```

2sat.h

Description: Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type $(a||b)\&\&(!a||c)\&\&(d||!b)\&\&\ldots$ becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions (~x).
Usage: `TwoSat ts(number of boolean variables);`
`ts.either(0, ~3); // Var 0 is true or var 3 is false`
`ts.set_value(2); // Var 2 is true`
`ts.at_most_one({0,~1,2}); // <= 1 of vars 0, ~1 and 2 are true`
`ts.solve(); // Returns true iff it is solvable`
`ts.values[0..N-1] holds the assigned values to the vars`
Time: $\mathcal{O}(N+E)$, where N is the number of boolean variables, and E is the number of clauses.

57 lines

```
struct TwoSat {
  int N;
  vector<vi> gr;
  vi values; // 0 = false, 1 = true

  TwoSat(int n = 0) : N(n), gr(2*n) {}

  int add_var() { // (optional)
    gr.emplace_back();
    gr.emplace_back();
    return N++;
  }

  void either(int f, int j) {
    f = (f >= 0 ? 2*f : -1-2*f);
    j = (j >= 0 ? 2*j : -1-2*j);
    gr[f^1].push_back(j);
    gr[j^1].push_back(f);
  }
  void set_value(int x) { either(x, x); }

  void at_most_one(const vi& li) { // (optional)
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i,2,sz(li)) {
      int next = add_var();
      either(cur, ~li[i]);
      either(cur, next);
      either(~li[i], next);
      cur = ~next;
    }
    either(cur, ~li[1]);
  }

  vi val, comp, z; int time = 0;
  int dfs(int i) {
    int low = val[i] = ++time, x; z.push_back(i);
    trav(e, gr[i]) if (!comp[e])
      low = min(low, val[e] ?: dfs(e));
    ++time;
    if (low == val[i]) do {
      x = z.back(); z.pop_back();
      comp[x] = time;
      if (values[x>>1] == -1)
        values[x>>1] = !(x&1);
    } while (x != i);
    return val[i] = low;
  }

  bool solve() {
    values.assign(N, -1);
    val.assign(2*N, 0); comp = val;
    rep(i,0,2*N) if (!comp[i]) dfs(i);
    rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
    return 1;
  }
};
```

7.7 Trees

TreePower.h

Description: Calculate power of two jumps in a tree. Assumes the root node points to itself.
Time: $\mathcal{O}(|V|\log|V|)$

14 lines

```
vector<vi> treeJump(vi& P){
  int on = 1, d = 1;
  while(on < sz(P)) on *= 2, d++;
  vector<vi> jmp(d, P);
  rep(i,1,d) rep(j,0,sz(P))
    jmp[i][j] = jmp[i-1][jmp[i-1][j]];
  return jmp;
}

int jmp(vector<vi>& tbl, int nod, int steps){
  rep(i,0,sz(tbl))
    if(steps&(1<<i)) nod = tbl[i][nod];
  return nod;
}
```

LCA.h

Description: Lowest common ancestor. Finds the lowest common ancestor in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected. Can also find the distance between two nodes.
Usage: `LCA lca(undirGraph);`
`lca.query(firstNode, secondNode);`
`lca.distance(firstNode, secondNode);`
Time: $\mathcal{O}(|V|\log|V|+Q)$

"../data-structures/RMQ.h" 38 lines

```
typedef vector<pii> vpi;
typedef vector<vpi> graph;
const pii inf(1 << 29, -1);

struct LCA {
  vi time;
  vector<ll> dist;
  RMQ<pii> rmq;

  LCA(graph& C) : time(sz(C), -99), dist(sz(C)), rmq(dfs(C)) {}
```

```
  vpi dfs(graph& C) {
    vector<tuple<int, int, int, ll> > q(1);
    vpi ret;
    int T = 0, v, p, d; ll di;
    while (!q.empty()) {
      tie(v, p, d, di) = q.back();
      q.pop_back();
      if (d) ret.emplace_back(d, p);
      time[v] = T++;
      dist[v] = di;
      trav(e, C[v]) if (e.first != p)
        q.emplace_back(e.first, v, d+1, di + e.second);
    }
    return ret;
  }

  int query(int a, int b) {
    if (a == b) return a;
    a = time[a], b = time[b];
    return rmq.query(min(a, b), max(a, b)).second;
  }
  ll distance(int a, int b) {
    int lca = query(a, b);
    return dist[a] + dist[b] - 2 * dist[lca];
  }
};
```

CompressTree.h

Description: Given a rooted tree and a subset S of nodes, compute the minimal subtree that contains all the nodes by adding all (at most $|S| - 1$) pairwise LCA's and compressing edges. Returns a list of (par, orig_index) representing a tree rooted at 0. The root points to itself.
Time: $\mathcal{O}(|S| \log |S|)$

"LCA.h" 20 lines

```
vpi compressTree(LCA& lca, const vi& subset) {
  static vi rev; rev.resize(sz(lca.dist));
  vi li = subset, &T = lca.time;
  auto cmp = [&](int a, int b) { return T[a] < T[b]; };
  sort(all(li), cmp);
  int m = sz(li)-1;
  rep(i,0,m) {
    int a = li[i], b = li[i+1];
    li.push_back(lca.query(a, b));
  }
  sort(all(li), cmp);
  li.erase(unique(all(li)), li.end());
  rep(i,0,sz(li)) rev[li[i]] = i;
  vpi ret = {pii(0, li[0])};
  rep(i,0,sz(li)-1) {
    int a = li[i], b = li[i+1];
    ret.emplace_back(rev[lca.query(a, b)], b);
  }
  return ret;
}
```

HLD.h

Description: Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most log(n) light edges. The function of the HLD can be changed by modifying T, LOW and f. f is assumed to be associative and commutative.
Usage: `HLD hld(G);`

```
hld.update(index, value);
tie(value, lca) = hld.query(n1, n2);
```

"../data-structures/SegmentTree.h" 93 lines

```
typedef vector<pii> vpi;

struct Node {
  int d, par, val, chain = -1, pos = -1;
};

struct Chain {
  int par, val;
  vector<int> nodes;
  Tree tree;
};

struct HLD {
  typedef int T;
  const T LOW = -(1<<29);
  void f(T& a, T b) { a = max(a, b); }

  vector<Node> V;
  vector<Chain> C;

  HLD(vector<vpi>& g) : V(sz(g)) {
    dfs(0, -1, g, 0);
    trav(c, C){
      c.tree.init(sz(c.nodes), 0);
      for (int ni : c.nodes)
        c.tree.update(V[ni].pos, V[ni].val);
    }
  }

  void update(int node, T val) {
    Node& n = V[node]; n.val = val;
    if (n.chain != -1) C[n.chain].tree.update(n.pos, val);
  }

  int pard(Node& nod) {
    if (nod.par == -1) return -1;
    return V[nod.chain == -1 ? nod.par : C[nod.chain].par].d;
  }

  // query all *edges* between n1, n2
  pair<T, int> query(int i1, int i2) {
    T ans = LOW;
    while(i1 != i2) {
      Node n1 = V[i1], n2 = V[i2];
      if (n1.chain != -1 && n1.chain == n2.chain) {
        int lo = n1.pos, hi = n2.pos;
        if (lo > hi) swap(lo, hi);
        f(ans, C[n1.chain].tree.query(lo, hi));
        i1 = i2 = C[n1.chain].nodes[hi];
      } else {
        if (pard(n1) < pard(n2))
          n1 = n2, swap(i1, i2);
        if (n1.chain == -1)
          f(ans, n1.val), i1 = n1.par;
        else {
          Chain& c = C[n1.chain];
          f(ans, n1.pos ? c.tree.query(n1.pos, sz(c.nodes))
                        : c.tree.s[1]);
          i1 = c.par;
        }
      }
    }
    return make_pair(ans, i1);
  }

  // query all *nodes* between n1, n2
  pair<T, int> query2(int i1, int i2) {
    pair<T, int> ans = query(i1, i2);
    f(ans.first, V[ans.second].val);
    return ans;
  }

  pii dfs(int at, int par, vector<vpi>& g, int d) {
    V[at].d = d; V[at].par = par;
    int sum = 1, ch, nod, sz;
    tuple<int,int,int> mx(-1,-1,-1);
    trav(e, g[at]){
      if (e.first == par) continue;
      tie(sz, ch) = dfs(e.first, at, g, d+1);
      V[e.first].val = e.second;
      sum += sz;
      mx = max(mx, make_tuple(sz, e.first, ch));
    }
    tie(sz, nod, ch) = mx;
    if (2*sz < sum) return pii(sum, -1);
    if (ch == -1) { ch = sz(C); C.emplace_back(); }
    V[nod].pos = sz(C[ch].nodes);
    V[nod].chain = ch;
    C[ch].par = at;
    C[ch].nodes.push_back(nod);
    return pii(sum, ch);
  }
};
```

LinkCutTree.h

Description: Represents a forest of unrooted trees. You can add and remove edges (as long as the result is still a forest), and check whether two nodes are in the same tree.
Time: All operations take amortized $\mathcal{O}(\log N)$.

96 lines

```
struct Node { // Splay tree. Root's pp contains tree's parent.
  Node *p = 0, *pp = 0, *c[2];
  bool flip = 0;
  Node() { c[0] = c[1] = 0; fix(); }
  void fix() {
    if (c[0]) c[0]->p = this;
    if (c[1]) c[1]->p = this;
    // (+ update sum of subtree elements etc. if wanted)
  }
  void push_flip() {
    if (!flip) return;
    flip = 0; swap(c[0], c[1]);
    if (c[0]) c[0]->flip ^= 1;
    if (c[1]) c[1]->flip ^= 1;
  }
  int up() { return p ? p->c[1] == this : -1; }
  void rot(int i, int b) {
    int h = i ^ b;
    Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
    if ((y->p = p)) p->c[up()] = y;
    c[i] = z->c[i ^ 1];
    if (b < 2) {
      x->c[h] = y->c[h ^ 1];
      z->c[h ^ 1] = b ? x : this;
    }
    y->c[i ^ 1] = b ? this : x;
    fix(); x->fix(); y->fix();
    if (p) p->fix();
    swap(pp, y->pp);
  }
  void splay() {
    for (push_flip(); p; ) {
      if (p->p) p->p->push_flip();
      p->push_flip(); push_flip();
      int c1 = up(), c2 = p->up();
      if (c2 == -1) p->rot(c1, 2);
      else p->p->rot(c2, c1 != c2);
    }
  }
  Node* first() {
```

```
    push_flip();
    return c[0] ? c[0]->first() : (splay(), this);
  }
};

struct LinkCut {
  vector<Node> node;
  LinkCut(int N) : node(N) {}

  void link(int u, int v) { // add an edge (u, v)
    assert(!connected(u, v));
    make_root(&node[u]);
    node[u].pp = &node[v];
  }

  void cut(int u, int v) { // remove an edge (u, v)
    Node *x = &node[u], *top = &node[v];
    make_root(top); x->splay();
    assert(top == (x->pp ?: x->c[0]));
    if (x->pp) x->pp = 0;
    else {
      x->c[0] = top->p = 0;
      x->fix();
    }
  }

  bool connected(int u, int v) { // are u, v in the same tree?
    Node* nu = access(&node[u])->first();
    return nu == access(&node[v])->first();
  }

  void make_root(Node* u) {
    access(u);
    u->splay();
    if(u->c[0]) {
      u->c[0]->p = 0;
      u->c[0]->flip ^= 1;
      u->c[0]->pp = u;
      u->c[0] = 0;
      u->fix();
    }
  }

  Node* access(Node* u) {
    u->splay();
    while (Node* pp = u->pp) {
      pp->splay(); u->pp = 0;
      if (pp->c[1]) {
        pp->c[1]->p = 0; pp->c[1]->pp = pp; }
      pp->c[1] = u; pp->fix(); u = pp;
    }
    return u;
  }
};
```

7.8 Matrix tree theorem

MatrixTree.h

Description: To count the number of spanning trees in an undirected graph G: create an $N \times N$ matrix `mat`, and for each edge $(a, b) \in G$, do `mat[a][a]++, mat[b][b]++, mat[a][b]--, mat[b][a]--`. Remove the last row and column, and take the determinant.

1 lines

Geometry (8)

8.1 Geometric primitives

Point.h

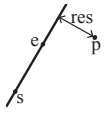
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

25 lines

```
template <class T>
struct Point {
  typedef Point P;
  T x, y;
  explicit Point(T x=0, T y=0) : x(x), y(y) {}
  bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
  bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
  P operator+(P p) const { return P(x+p.x, y+p.y); }
  P operator-(P p) const { return P(x-p.x, y-p.y); }
  P operator*(T d) const { return P(x*d, y*d); }
  P operator/(T d) const { return P(x/d, y/d); }
  T dot(P p) const { return x*p.x + y*p.y; }
  T cross(P p) const { return x*p.y - y*p.x; }
  T cross(P a, P b) const { return (a-*this).cross(b-*this); }
  T dist2() const { return x*x + y*y; }
  double dist() const { return sqrt((double)dist2()); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x); }
  P unit() const { return *this/dist(); } // makes dist()=1
  P perp() const { return P(-y, x); } // rotates +90 degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around the origin
  P rotate(double a) const {
    return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
};
```

lineDistance.h

Description:
Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. a==b gives nan. P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance.

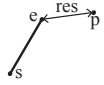


"Point.h" 4 lines

```
template <class P>
double lineDist(const P& a, const P& b, const P& p) {
  return (double)(b-a).cross(p-a)/(b-a).dist();
}
```

SegmentDistance.h

Description:
Returns the shortest distance between point p and the line segment from point s to e.



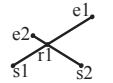
Usage:
```
Point<double> a, b(2,2), p(1,1);
bool onSegment = segDist(a,b,p) < 1e-10;
```

"Point.h" 6 lines

```
typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
  if (s==e) return (p-s).dist();
  auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)));
  return ((p-s)*d-(e-s)*t).dist()/d;
}
```

SegmentIntersection.h

Description:
If a unique intersetion point between the line segments going from s1 to e1 and from s2 to e2 exists r1 is set to this point and 1 is returned. If no intersection point exists 0 is returned and if infinitely many exists 2 is returned and r1 and r2 are set to the two ends of the common line. The wrong position will be returned if P is Point<int> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long. Use segmentIntersectionQ to get just a true/false answer.



Usage:
```
Point<double> intersection, dummy;
if (segmentIntersection(s1,e1,s2,e2,intersection,dummy)==1)
cout << "segments intersect at " << intersection << endl;
```

"Point.h" 27 lines

```
template <class P>
int segmentIntersection(const P& s1, const P& e1,
    const P& s2, const P& e2, P& r1, P& r2) {
  if (e1==s1) {
    if (e2==s2) {
      if (e1==e2) { r1 = e1; return 1; } //all equal
      else return 0; //different point segments
    } else return segmentIntersection(s2,e2,s1,e1,r1,r2);//swap
  }
  //segment directions and separation
  P v1 = e1-s1, v2 = e2-s2, d = s2-s1;
  auto a = v1.cross(v2), a1 = v1.cross(d), a2 = v2.cross(d);
  if (a == 0) { //if parallel
    auto b1=s1.dot(v1), c1=e1.dot(v1),
         b2=s2.dot(v1), c2=e2.dot(v1);
    if (a1 || a2 || max(b1,min(b2,c2))>min(c1,max(b2,c2)))
      return 0;
    r1 = min(b2,c2)<b1 ? s1 : (b2<c2 ? s2 : e2);
    r2 = max(b2,c2)>c1 ? e1 : (b2>c2 ? s2 : e2);
    return 2-(r1==r2);
  }
  if (a < 0) { a = -a; a1 = -a1; a2 = -a2; }
  if (0<a1 || a<-a1 || 0<a2 || a<-a2)
    return 0;
  r1 = s1-v1*a2/a;
  return 1;
}
```

SegmentIntersectionQ.h

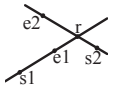
Description: Like segmentIntersection, but only returns true/false. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.

"Point.h" 16 lines

```
template <class P>
bool segmentIntersectionQ(P s1, P e1, P s2, P e2) {
  if (e1 == s1) {
    if (e2 == s2) return e1 == e2;
    swap(s1,s2); swap(e1,e2);
  }
  P v1 = e1-s1, v2 = e2-s2, d = s2-s1;
  auto a = v1.cross(v2), a1 = d.cross(v1), a2 = d.cross(v2);
  if (a == 0) { // parallel
    auto b1 = s1.dot(v1), c1 = e1.dot(v1),
         b2 = s2.dot(v1), c2 = e2.dot(v1);
    return !a1 && max(b1,min(b2,c2)) <= min(c1,max(b2,c2));
  }
  if (a < 0) { a = -a; a1 = -a1; a2 = -a2; }
  return (0 <= a1 && a1 <= a && 0 <= a2 && a2 <= a);
}
```

lineIntersection.h

Description:
If a unique intersetion point of the lines going through s1,e1 and s2,e2 exists r is set to this point and 1 is returned. If no intersection point exists 0 is returned and if infinitely many exists -1 is returned. If s1==e1 or s2==e2 -1 is returned. The wrong position will be returned if P is Point<int> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.



Usage:
```
point<double> intersection;
if (1 == LineIntersection(s1,e1,s2,e2,intersection))
cout << "intersection point at " << intersection << endl;
```

"Point.h" — 9 lines

```
template <class P>
int lineIntersection(const P& s1, const P& e1, const P& s2,
    const P& e2, P& r) {
  if ((e1-s1).cross(e2-s2)) { //if not parallell
    r = s2-(e2-s2)*(e1-s1).cross(s2-s1)/(e1-s1).cross(e2-s2);
    return 1;
  } else
    return -((e1-s1).cross(s2-s1)==0 || s2==e2);
}
```

sideOf.h

Description: Returns where p is as seen from s towards e. 1/0/-1 $\Leftrightarrow$ left/on line/right. If the optional argument *eps* is given 0 is returned if p is within distance *eps* from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.

Usage:
```
bool left = sideOf(p1,p2,q)==1;
```

"Point.h" — 11 lines

```
template <class P>
int sideOf(const P& s, const P& e, const P& p) {
  auto a = (e-s).cross(p-s);
  return (a > 0) - (a < 0);
}
template <class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
  auto a = (e-s).cross(p-s);
  double l = (e-s).dist()*eps;
  return (a > l) - (a < -l);
}
```

onSegment.h

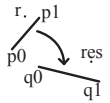
Description: Returns true iff p lies on the line segment from s to e. Intended for use with e.g. Point<long long> where overflow is an issue. Use (segDist(s,e,p)<=epsilon) instead when using Point<double>.

"Point.h" — 5 lines

```
template <class P>
bool onSegment(const P& s, const P& e, const P& p) {
  P ds = p-s, de = p-e;
  return ds.cross(de) == 0 && ds.dot(de) <= 0;
}
```

linearTransformation.h

Description:
Apply the linear transformation (translation, rotation and scaling) which takes line p0-p1 to line q0-q1 to point r.



"Point.h" — 6 lines

```
typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1,
    const P& q0, const P& q1, const P& r) {
  P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
  return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2();
}
```

Angle.h

Description: A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping.

Usage:
```
vector<Angle> v = {w[0], w[0].t360() ...}; // sorted
int j = 0; rep(i,0,n) {
while (v[j] < v[i].t180()) ++j;
} // sweeps j such that (j-i) represents the number of
positively oriented triangles with vertices at 0 and i
```

37 lines

```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
  Angle operator-(Angle a) const { return {x-a.x, y-a.y, t}; }
  int quad() const {
    assert(x || y);
    if (y < 0) return (x >= 0) + 2;
    if (y > 0) return (x <= 0);
    return (x <= 0) * 2;
  }
  Angle t90() const { return {-y, x, t + (quad() == 3)}; }
  Angle t180() const { return {-x, -y, t + (quad() >= 2)}; }
  Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
  // add a.dist2() and b.dist2() to also compare distances
  return make_tuple(a.t, a.quad(), a.y * (ll)b.x) <
         make_tuple(b.t, b.quad(), a.x * (ll)b.y);
}
bool operator>=(Angle a, Angle b) { return !(a < b); }
bool operator>(Angle a, Angle b) { return b < a; }
bool operator<=(Angle a, Angle b) { return !(b < a); }

// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
  if (b < a) swap(a, b);
  return (b < a.t180() ?
          make_pair(a, b) : make_pair(b, a.t360()));
}

Angle operator+(Angle a, Angle b) { // where b is a vector
  Angle r(a.x + b.x, a.y + b.y, a.t);
  if (r > a.t180()) r.t--;
  return r.t180() < a ? r.t360() : r;
}
```

8.2 Circles

CircleIntersection.h

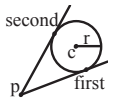
Description: Computes a pair of points at which two circles intersect. Returns false in case of no intersection.

"Point.h" — 14 lines

```
typedef Point<double> P;
bool circleIntersection(P a, P b, double r1, double r2,
    pair<P, P>* out) {
  P delta = b - a;
  assert(delta.x || delta.y || r1 != r2);
  if (!delta.x && !delta.y) return false;
  double r = r1 + r2, d2 = delta.dist2();
  double p = (d2 + r1*r1 - r2*r2) / (2.0 * d2);
  double h2 = r1*r1 - p*p*d2;
  if (d2 > r*r || h2 < 0) return false;
  P mid = a + delta*p, per = delta.perp() * sqrt(h2 / d2);
  *out = {mid + per, mid - per};
  return true;
}
```

circleTangents.h

Description:
Returns a pair of the two points on the circle with radius r centered around c whos tangent lines intersect p. If p lies within the circle NaN-points are returned. P is intended to be Point<double>. The first point is the one to the right as seen from the p towards c.



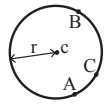
Usage:
```
typedef Point<double> P;
pair<P,P> p = circleTangents(P(100,2),P(0,0),2);
```

"Point.h" — 6 lines

```
template <class P>
pair<P,P> circleTangents(const P &p, const P &c, double r) {
  P a = p-c;
  double x = r*r/a.dist2(), y = sqrt(x-x*x);
  return make_pair(c+a*x+a.perp()*y, c+a*x-a.perp()*y);
}
```

circumcircle.h

Description:
The circumcirle of a triangle is the circle intersecting all three vertices. ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.



"Point.h" — 9 lines

```
typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
  return (B-A).dist()*(C-B).dist()*(A-C).dist()/
      abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
  P b = C-A, c = B-A;
  return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

MinimumEnclosingCircle.h

Description: Computes the minimum circle that encloses a set of points.

Time: expected $\mathcal{O}(n)$

"circumcircle.h" — 28 lines

```
pair<double, P> mec2(vector<P>& S, P a, P b, int n) {
  double hi = INFINITY, lo = -hi;
  rep(i,0,n) {
    auto si = (b-a).cross(S[i]-a);
    if (si == 0) continue;
    P m = ccCenter(a, b, S[i]);
    auto cr = (b-a).cross(m-a);
    if (si < 0) hi = min(hi, cr);
    else lo = max(lo, cr);
  }
  double v = (0 < lo ? lo : hi < 0 ? hi : 0);
  P c = (a + b) / 2 + (b - a).perp() * v / (b - a).dist2();
  return {(a - c).dist2(), c};
}
pair<double, P> mec(vector<P>& S, P a, int n) {
  random_shuffle(S.begin(), S.begin() + n);
  P b = S[0], c = (a + b) / 2;
  double r = (a - c).dist2();
  rep(i,1,n) if ((S[i] - c).dist2() > r * (1 + 1e-8)) {
    tie(r,c) = (n == sz(S) ?
      mec(S, S[i], i) : mec2(S, a, S[i], i));
  }
  return {r, c};
}
pair<double, P> enclosingCircle(vector<P> S) {
  assert(!S.empty()); auto r = mec(S, S[0], sz(S));
  return {sqrt(r.first), r.second};
}
```

8.3 Polygons

insidePolygon.h

Description: Returns true if p lies within the polygon described by the points between iterators begin and end. If strict false is returned when p is on the edge of the polygon. Answer is calculated by counting the number of intersections between the polygon and a line going from p to infinity in the positive x-direction. The algorithm uses products in intermediate steps so watch out for overflow. If points within epsilon from an edge should be considered as on the edge replace the line "if (onSegment..." with the comment bellow it (this will cause overflow for int and long long).

Usage:
```
typedef Point<int> pi;
vector<pi> v; v.push_back(pi(4,4));
v.push_back(pi(1,2)); v.push_back(pi(2,1));
bool in = insidePolygon(v.begin(),v.end(), pi(3,4), false);
```

Time: $\mathcal{O}(n)$

"Point.h", "onSegment.h", "SegmentDistance.h" — 14 lines

```
template <class It, class P>
bool insidePolygon(It begin, It end, const P& p,
    bool strict = true) {
  int n = 0; //number of isects with line from p to (inf,p.y)
  for (It i = begin, j = end-1; i != end; j = i++) {
    //if p is on edge of polygon
    if (onSegment(*i, *j, p)) return !strict;
    //or: if (segDist(*i, *j, p) <= epsilon) return !strict;
    //increment n if segment intersects line from p
    n += (max(i->y,j->y) > p.y && min(i->y,j->y) <= p.y &&
        ((*j-*i).cross(p-*i) > 0) == (i->y <= p.y));
  }
  return n&1; //inside if odd number of intersections
}
```

PolygonArea.h

Description: Returns twice the signed area of a polygon. Clockwise enumeration gives negative area. Watch out for overflow if using int as T!

"Point.h" — 6 lines

```
template <class T>
T polygonArea2(vector<Point<T>>& v) {
  T a = v.back().cross(v[0]);
  rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
  return a;
}
```

PolygonCenter.h

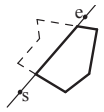
Description: Returns the center of mass for a polygon.

"Point.h" — 10 lines

```
typedef Point<double> P;
Point<double> polygonCenter(vector<P>& v) {
  auto i = v.begin(), end = v.end(), j = end-1;
  Point<double> res{0,0}; double A = 0;
  for (; i != end; j=i++) {
    res = res + (*i + *j) * j->cross(*i);
    A += j->cross(*i);
  }
  return res / A / 3;
}
```

PolygonCut.h

Description: Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.



Usage:
```
vector<P> p = ...;
p = polygonCut(p, P(0,0), P(1,0));
```

"Point.h", "lineIntersection.h" — 15 lines

```
typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
  vector<P> res;
  rep(i,0,sz(poly)) {
    P cur = poly[i], prev = i ? poly[i-1] : poly.back();
    bool side = s.cross(e, cur) < 0;
    if (side != (s.cross(e, prev) < 0)) {
      res.emplace_back();
      lineIntersection(s, e, cur, prev, res.back());
    }
    if (side)
      res.push_back(cur);
  }
  return res;
}
```

ConvexHull.h

Description: Returns a vector of indices of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.

Usage:
```
vector<P> ps, hull;
trav(i, convexHull(ps)) hull.push_back(ps[i]);
```

Time: $\mathcal{O}(n \log n)$

"Point.h" — 20 lines

```
typedef Point<ll> P;
pair<vi, vi> ulHull(const vector<P>& S) {
  vi Q(sz(S)), U, L;
  iota(all(Q), 0);
  sort(all(Q), [&S](int a, int b){ return S[a] < S[b]; });
  trav(it, Q) {
#define ADDP(C, cmp) while (sz(C) > 1 && S[C[sz(C)-2]].cross(\
  S[it], S[C.back()]) cmp 0) C.pop_back(); C.push_back(it);
    ADDP(U, <=); ADDP(L, >=);
  }
  return {U, L};
}

vi convexHull(const vector<P>& S) {
  vi u, l; tie(u, l) = ulHull(S);
  if (sz(S) <= 1) return u;
  if (S[u[0]] == S[u[1]]) return {0};
  l.insert(l.end(), u.rbegin()+1, u.rend()-1);
  return l;
}
```

PolygonDiameter.h

Description: Calculates the max squared distance of a set of points.

"ConvexHull.h" — 19 lines

```
vector<pii> antipodal(const vector<P>& S, vi& U, vi& L) {
  vector<pii> ret;
  int i = 0, j = sz(L) - 1;
  while (i < sz(U) - 1 || j > 0) {
    ret.emplace_back(U[i], L[j]);
    if (j == 0 || (i != sz(U)-1 && (S[L[j]] - S[L[j-1]])
          .cross(S[U[i+1]] - S[U[i]]) > 0)) ++i;
    else --j;
  }
  return ret;
}

pii polygonDiameter(const vector<P>& S) {
  vi U, L; tie(U, L) = ulHull(S);
  pair<ll, pii> ans;
  trav(x, antipodal(S, U, L))
    ans = max(ans, {(S[x.first] - S[x.second]).dist2(), x});
  return ans.second;
}
```

PointInsideHull.h

Description: Determine whether a point t lies inside a given polygon (counter-clockwise order). The polygon must be such that every point on the circumference is visible from the first point in the vector. It returns 0 for points outside, 1 for points on the circumference, and 2 for points inside.

Time: $\mathcal{O}(\log N)$

"Point.h", "sideOf.h", "onSegment.h" — 22 lines

```
typedef Point<ll> P;
int insideHull2(const vector<P>& H, int L, int R, const P& p) {
  int len = R - L;
  if (len == 2) {
    int sa = sideOf(H[0], H[L], p);
    int sb = sideOf(H[L], H[L+1], p);
    int sc = sideOf(H[L+1], H[0], p);
    if (sa < 0 || sb < 0 || sc < 0) return 0;
    if (sb==0 || (sa==0 && L == 1) || (sc == 0 && R == sz(H)))
      return 1;
    return 2;
  }
  int mid = L + len / 2;
  if (sideOf(H[0], H[mid], p) >= 0)
    return insideHull2(H, mid, R, p);
  return insideHull2(H, L, mid+1, p);
}

int insideHull(const vector<P>& hull, const P& p) {
  if (sz(hull) < 3) return onSegment(hull[0], hull.back(), p);
  else return insideHull2(hull, 1, sz(hull), p);
}
```

LineHullIntersection.h

Description: Line-convex polygon intersection. The polygon must be ccw and have no colinear points. isct(a, b) returns a pair describing the intersection of a line with the polygon: • $(-1,-1)$ if no collision, • $(i,-1)$ if touching the corner i, • (i,i) if along side $(i,i+1)$, • (i,j) if crossing sides $(i,i+1)$ and $(j,j+1)$. In the last case, if a corner i is crossed, this is treated as happening on side $(i,i+1)$. The points are returned in the same order as the line hits the polygon.

Time: $\mathcal{O}(N + Q \log n)$

"Point.h" — 63 lines

```
ll sgn(ll a) { return (a > 0) - (a < 0); }
typedef Point<ll> P;
struct HullIntersection {
  int N;
  vector<P> p;
  vector<pair<P, int>> a;

  HullIntersection(const vector<P>& ps) : N(sz(ps)), p(ps) {
    p.insert(p.end(), all(ps));
    int b = 0;
    rep(i,1,N) if (P{p[i].y,p[i].x} < P{p[b].y, p[b].x}) b = i;
    rep(i,0,N) {
      int f = (i + b) % N;
      a.emplace_back(p[f+1] - p[f], f);
    }
  }

  int qd(P p) {
    return (p.y < 0) ? (p.x >= 0) + 2
         : (p.x <= 0) * (1 + (p.y <= 0));
  }

  int bs(P dir) {
    int lo = -1, hi = N;
    while (hi - lo > 1) {
      int mid = (lo + hi) / 2;
      if (make_pair(qd(dir), dir.y * a[mid].first.x) <
        make_pair(qd(a[mid].first), dir.x * a[mid].first.y))
```

```
      hi = mid;
    else lo = mid;
  }
  return a[hi%N].second;
}

bool isign(P a, P b, int x, int y, int s) {
  return sgn(a.cross(p[x], b)) * sgn(a.cross(p[y], b)) == s;
}

int bs2(int lo, int hi, P a, P b) {
  int L = lo;
  if (hi < lo) hi += N;
  while (hi - lo > 1) {
    int mid = (lo + hi) / 2;
    if (isign(a, b, mid, L, -1)) hi = mid;
    else lo = mid;
  }
  return lo;
}

pii isct(P a, P b) {
  int f = bs(a - b), j = bs(b - a);
  if (isign(a, b, f, j, 1)) return {-1, -1};
  int x = bs2(f, j, a, b)%N,
      y = bs2(j, f, a, b)%N;
  if (a.cross(p[x], b) == 0 &&
      a.cross(p[x+1], b) == 0) return {x, x};
  if (a.cross(p[y], b) == 0 &&
      a.cross(p[y+1], b) == 0) return {y, y};
  if (a.cross(p[f], b) == 0) return {f, -1};
  if (a.cross(p[j], b) == 0) return {j, -1};
  return {x, y};
}
};
```

8.4 Misc. Point Set Problems

closestPair.h

Description: $i1$, $i2$ are the indices to the closest pair of points in the point vector p after the call. The distance is returned.
Time: $\mathcal{O}(n \log n)$

"Point.h" 58 lines

```
template <class It>
bool it_less(const It& i, const It& j) { return *i < *j; }
template <class It>
bool y_it_less(const It& i,const It& j) {return i->y < j->y;}

template<class It, class IIt> /* IIt = vector<It>::iterator */
double cp_sub(IIt ya, IIt yaend, IIt xa, It &i1, It &i2) {
  typedef typename iterator_traits<It>::value_type P;
  int n = yaend-ya, split = n/2;
  if(n <= 3) { // base case
    double a = (*xa[1]-*xa[0]).dist(), b = 1e50, c = 1e50;
    if(n==3) b=(*xa[2]-*xa[0]).dist(), c=(*xa[2]-*xa[1]).dist()
        ;
    if(a <= b) { i1 = xa[1];
      if(a <= c) return i2 = xa[0], a;
      else return i2 = xa[2], c;
    } else { i1 = xa[2];
      if(b <= c) return i2 = xa[0], b;
      else return i2 = xa[1], c;
  } }
  vector<It> ly, ry, stripy;
  P splitp = *xa[split];
  double splitx = splitp.x;
  for(IIt i = ya; i != yaend; ++i) { // Divide
    if(*i != xa[split] && (**i-splitp).dist2() < 1e-12)
      return i1 = *i, i2 = xa[split], 0;// nasty special case!
    if (**i < splitp) ly.push_back(*i);
    else ry.push_back(*i);
  } // assert((signed)lefty.size() == split)
  It j1, j2; // Conquer
  double a = cp_sub(ly.begin(), ly.end(), xa, i1, i2);
  double b = cp_sub(ry.begin(), ry.end(), xa+split, j1, j2);
  if(b < a) a = b, i1 = j1, i2 = j2;
  double a2 = a*a;
  for(IIt i = ya; i != yaend; ++i) { // Create strip (y-sorted)
    double x = (*i)->x;
    if(x >= splitx-a && x <= splitx+a) stripy.push_back(*i);
  }
  for(IIt i = stripy.begin(); i != stripy.end(); ++i) {
    const P &p1 = **i;
    for(IIt j = i+1; j != stripy.end(); ++j) {
      const P &p2 = **j;
      if(p2.y-p1.y > a) break;
      double d2 = (p2-p1).dist2();
      if(d2 < a2) i1 = *i, i2 = *j, a2 = d2;
  } }
  return sqrt(a2);
}

template<class It> // It is random access iterators of point<T>
double closestpair(It begin, It end, It &i1, It &i2 ) {
  vector<It> xa, ya;
  assert(end-begin >= 2);
  for (It i = begin; i != end; ++i)
    xa.push_back(i), ya.push_back(i);
  sort(xa.begin(), xa.end(), it_less<It>);
  sort(ya.begin(), ya.end(), y_it_less<It>);
  return cp_sub(ya.begin(), ya.end(), xa.begin(), i1, i2);
}
```

kdTree.h

Description: KD-tree (2d, can be extended to 3d)

"Point.h" 63 lines

```
typedef long long T;
typedef Point<T> P;
const T INF = numeric_limits<T>::max();

bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; }

struct Node {
  P pt; // if this is a leaf, the single point in it
  T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
  Node *first = 0, *second = 0;

  T distance(const P& p) { // min squared distance to a point
    T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
    T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
    return (P(x,y) - p).dist2();
  }

  Node(vector<P>&& vp) : pt(vp[0]) {
    for (P p : vp) {
      x0 = min(x0, p.x); x1 = max(x1, p.x);
      y0 = min(y0, p.y); y1 = max(y1, p.y);
    }
    if (vp.size() > 1) {
      // split on x if the box is wider than high (not best
          heuristic...)
      sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
      // divide by taking half the array for each child (not
      // best performance with many duplicates in the middle)
      int half = sz(vp)/2;
      first = new Node({vp.begin(), vp.begin() + half});
      second = new Node({vp.begin() + half, vp.end()});
    }
  }
};

struct KDTree {
  Node* root;
  KDTree(const vector<P>& vp) : root(new Node({all(vp)})) {}

  pair<T, P> search(Node *node, const P& p) {
    if (!node->first) {
      // uncomment if we should not find the point itself:
      // if (p == node->pt) return {INF, P()};
      return make_pair((p - node->pt).dist2(), node->pt);
    }

    Node *f = node->first, *s = node->second;
    T bfirst = f->distance(p), bsec = s->distance(p);
    if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);

    // search closest side first, other side if needed
    auto best = search(f, p);
    if (bsec < best.first)
      best = min(best, search(s, p));
    return best;
  }

  // find nearest point to a point, and its squared distance
  // (requires an arbitrary operator< for Point)
  pair<T, P> nearest(const P& p) {
    return search(root, p);
  }
};
```

DelaunayTriangulation.h

Description: Computes the Delaunay triangulation of a set of points. Each circumcircle contains none of the input points. If any three points are colinear or any four are on the same circle, behavior is undefined.
Time: $\mathcal{O}(n^2)$

"Point.h", "3dHull.h" 10 lines

```
template<class P, class F>
void delaunay(vector<P>& ps, F trifun) {
  if (sz(ps) == 3) { int d = (ps[0].cross(ps[1], ps[2]) < 0);
    trifun(0,1+d,2-d); }
  vector<P3> p3;
  trav(p, ps) p3.emplace_back(p.x, p.y, p.dist2());
  if (sz(ps) > 3) trav(t, hull3d(p3)) if ((p3[t.b]-p3[t.a]).
      cross(p3[t.c]-p3[t.a]).dot(P3(0,0,1)) < 0)
    trifun(t.a, t.c, t.b);
}
```

8.5 3D

PolyhedronVolume.h

Description: Magic formula for the volume of a polyhedron. Faces should point outwards.

6 lines

```
template <class V, class L>
double signed_poly_volume(const V& p, const L& trilist) {
  double v = 0;
  trav(i, trilist) v += p[i.a].cross(p[i.b]).dot(p[i.c]);
  return v / 6;
}
```

Point3D.h

Description: Class to handle points in 3D space. T can be e.g. double or long long.

32 lines

```
template <class T> struct Point3D {
  typedef Point3D P;
  typedef const P& R;
  T x, y, z;
  explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
  bool operator<(R p) const {
    return tie(x, y, z) < tie(p.x, p.y, p.z); }
  bool operator==(R p) const {
    return tie(x, y, z) == tie(p.x, p.y, p.z); }
  P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
  P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
  P operator*(T d) const { return P(x*d, y*d, z*d); }
  P operator/(T d) const { return P(x/d, y/d, z/d); }
  T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
  P cross(R p) const {
    return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
  }
  T dist2() const { return x*x + y*y + z*z; }
  double dist() const { return sqrt((double)dist2()); }
  //Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
  double phi() const { return atan2(y, x); }
  //Zenith angle (latitude) to the z-axis in interval [0, pi]
  double theta() const { return atan2(sqrt(x*x+y*y),z); }
  P unit() const { return *this/(T)dist(); } //makes dist()=1
  //returns unit vector normal to *this and p
  P normal(P p) const { return cross(p).unit(); }
  //returns point rotated 'angle' radians ccw around axis
  P rotate(double angle, P axis) const {
    double s = sin(angle), c = cos(angle); P u = axis.unit();
    return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
  }
};
```

3dHull.h

Description: Computes all faces of the 3-dimension hull of a point set. *No four points must be coplanar*, or else random results will be returned. All faces will point outwards.

Time: $\mathcal{O}\left(n^2\right)$

"Point3D.h"

49 lines

```
typedef Point3D<double> P3;

struct PR {
  void ins(int x) { (a == -1 ? a : b) = x; }
  void rem(int x) { (a == x ? a : b) = -1; }
  int cnt() { return (a != -1) + (b != -1); }
  int a, b;
};

struct F { P3 q; int a, b, c; };

vector<F> hull3d(const vector<P3>& A) {
  assert(sz(A) >= 4);
  vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
  vector<F> FS;
  auto mf = [&](int i, int j, int k, int l) {
    P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
    if (q.dot(A[l]) > q.dot(A[i]))
      q = q * -1;
    F f{q, i, j, k};
    E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
    FS.push_back(f);
  };
  rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
    mf(i, j, k, 6 - i - j - k);
  rep(i,4,sz(A)) {
    rep(j,0,sz(FS)) {
      F f = FS[j];
      if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
        E(a,b).rem(f.c);
        E(a,c).rem(f.b);
        E(b,c).rem(f.a);
        swap(FS[j--], FS.back());
        FS.pop_back();
      }
    }
    int nw = sz(FS);
    rep(j,0,nw) {
      F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f.c);
      C(a, b, c); C(a, c, b); C(b, c, a);
    }
  }
  trav(it, FS) if ((A[it.b] - A[it.a]).cross(
    A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
  return FS;
};
```

sphericalDistance.h

Description: Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 (ϕ_1) and f2 (ϕ_2) from x axis and zenith angles (latitude) t1 (θ_1) and t2 (θ_2) from z axis. All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so if that is what you have you can use only the two last rows. dx*radius is then the difference between the two points in the x direction and d*radius is the total distance between the points.

8 lines

```
double sphericalDistance(double f1, double t1,
    double f2, double t2, double radius) {
  double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
  double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
  double dz = cos(t2) - cos(t1);
  double d = sqrt(dx*dx + dy*dy + dz*dz);
  return radius*2*asin(d/2);
}
```

Strings (9)

KMP.h

Description: pi[x] computes the length of the longest prefix of s that ends at x, other than s[0..x] itself This is used by find to find all occurances of a string.

Usage: `vi p = pi(pattern); vi occ = find(word, p);`

Time: $\mathcal{O}\left(pattern\right)$ for pi, $\mathcal{O}\left(word + pattern\right)$ for find

16 lines

```
vi pi(const string& s) {
  vi p(sz(s));
  rep(i,1,sz(s)) {
    int g = p[i-1];
    while (g && s[i] != s[g]) g = p[g-1];
    p[i] = g + (s[i] == s[g]);
  }
  return p;
}

vi match(const string& s, const string& pat) {
  vi p = pi(pat + '\0' + s), res;
  rep(i,sz(p)-sz(s),sz(p))
    if (p[i] == sz(pat)) res.push_back(i - 2 * sz(pat));
  return res;
}
```

Manacher.h

Description: For each position in a string, computes p[0][i] = half length of longest even palindrome around pos i, p[1][i] = longest odd (half rounded down).

Time: $\mathcal{O}\left(N\right)$

11 lines

```
void manacher(const string& s) {
  int n = sz(s);
  vi p[2] = {vi(n+1), vi(n)};
  rep(z,0,2) for (int i=0,l=0,r=0; i < n; i++) {
    int t = r-i+!z;
    if (i<r) p[z][i] = min(t, p[z][l+t]);
    int L = i-p[z][i], R = i+p[z][i]-!z;
    while (L>=1 && R+1<n && s[L-1] == s[R+1])
      p[z][i]++, L--, R++;
    if (R>r) l=L, r=R;
}}
```

MinRotation.h

Description: Finds the lexicographically smallest rotation of a string.

Usage: `rotate(v.begin(), v.begin()+min_rotation(v), v.end());`

Time: $\mathcal{O}\left(N\right)$

8 lines

```
int min_rotation(string s) {
  int a=0, N=sz(s); s += s;
  rep(b,0,N) rep(i,0,N) {
    if (a+i == b || s[a+i] < s[b+i]) {b += max(0, i-1); break;}
    if (s[a+i] > s[b+i]) { a = b; break; }
  }
  return a;
}
```

SuffixArray.h

Description: Builds suffix array for a string. $a[i]$ is the starting index of the suffix which is i-th in the sorted suffix array. The returned vector is of size $n+1$, and $a[0] = n$. The `lcp` function calculates longest common prefixes for neighbouring strings in suffix array. The returned vector is of size $n+1$, and $ret[0] = 0$.

Memory: $\mathcal{O}\left(N\right)$

Time: $\mathcal{O}\left(N \log^2 N\right)$ where N is the length of the string for creation of the SA. $\mathcal{O}\left(N\right)$ for longest common prefixes.

61 lines

```
typedef pair<ll, int> pli;
void count_sort(vector<pli> &b, int bits) { // (optional)
  //this is just 3 times faster than stl sort for N=10^6
  int mask = (1 << bits) - 1;
  rep(it,0,2) {
    int move = it * bits;
    vi q(1 << bits), w(sz(q) + 1);
    rep(i,0,sz(b))
      q[(b[i].first >> move) & mask]++;
    partial_sum(q.begin(), q.end(), w.begin() + 1);
    vector<pli> res(b.size());
    rep(i,0,sz(b))
      res[w[(b[i].first >> move) & mask]++] = b[i];
    swap(b, res);
  }
}
struct SuffixArray {
  vi a;
  string s;
  SuffixArray(const string& _s) : s(_s + '\0') {
    int N = sz(s);
    vector<pli> b(N);
    a.resize(N);
    rep(i,0,N) {
      b[i].first = s[i];
      b[i].second = i;
    }
```

```
    int q = 8;
    while ((1 << q) < N) q++;
    for (int moc = 0;; moc++) {
      count_sort(b, q); // sort(all(b)) can be used as well
      a[b[0].second] = 0;
      rep(i,1,N)
        a[b[i].second] = a[b[i - 1].second] +
          (b[i - 1].first != b[i].first);

      if ((1 << moc) >= N) break;
      rep(i,0,N) {
        b[i].first = (ll)a[i] << q;
        if (i + (1 << moc) < N)
          b[i].first += a[i + (1 << moc)];
        b[i].second = i;
      }
    }
    rep(i,0,sz(a)) a[i] = b[i].second;
  }
  vi lcp() {
    // longest common prefixes: res[i] = lcp(a[i], a[i-1])
    int n = sz(a), h = 0;
    vi inv(n), res(n);
    rep(i,0,n) inv[a[i]] = i;
    rep(i,0,n) if (inv[i] > 0) {
      int p0 = a[inv[i] - 1];
      while (s[i + h] == s[p0 + h]) h++;
      res[inv[i]] = h;
      if(h > 0) h--;
    }
    return res;
  }
};
```

SuffixTree.h

Description: Ukkonen's algorithm for online suffix tree construction. Each node contains indices [l, r) into the string, and a list of child nodes. Suffixes are given by traversals of this tree, joining [l, r) substrings. The root is 0 (has l = -1, r = 0), non-existent children are -1. To get a complete tree, append a dummy symbol – otherwise it may contain an incomplete path (still useful for substring matching, though).
Time: $\mathcal{O}(26N)$ 50 lines

```
struct SuffixTree {
  enum { N = 200010, ALPHA = 26 }; // N ~ 2*maxlen+10
  int toi(char c) { return c - 'a'; }
  string a; // v = cur node, q = cur position
  int t[N][ALPHA],l[N],r[N],p[N],s[N],v=0,q=0,m=2;

  void ukkadd(int i, int c) { suff:
    if (r[v]<=q) {
      if (t[v][c]==-1) { t[v][c]=m;  l[m]=i;
        p[m++]=v; v=s[v]; q=r[v];  goto suff; }
      v=t[v][c]; q=l[v];
    }
    if (q==-1 || c==toi(a[q])) q++; else {
      l[m+1]=i;  p[m+1]=m;  l[m]=l[v];  r[m]=q;
      p[m]=p[v];  t[m][c]=m+1;  t[m][toi(a[q])]=v;
      l[v]=q;  p[v]=m;  t[p[m]][toi(a[l[m]])]=m;
      v=s[p[m]];  q=l[m];
      while (q<r[m]) { v=t[v][toi(a[q])];  q+=r[v]-l[v]; }
      if (q==r[m])  s[m]=v;  else s[m]=m+2;
      q=r[v]-(q-r[m]);  m+=2;  goto suff;
    }
  }

  SuffixTree(string a) : a(a) {
    fill(r,r+N,sz(a));
    memset(s, 0, sizeof s);
    memset(t, -1, sizeof t);
    fill(t[1],t[1]+ALPHA,0);
    s[0] = 1; l[0] = l[1] = -1; r[0] = r[1] = p[0] = p[1] = 0;
    rep(i,0,sz(a)) ukkadd(i, toi(a[i]));
  }

  // example: find longest common substring (uses ALPHA = 28)
  pii best;
  int lcs(int node, int i1, int i2, int olen) {
    if (l[node] <= i1 && i1 < r[node]) return 1;
    if (l[node] <= i2 && i2 < r[node]) return 2;
    int mask = 0, len = node ? olen + (r[node] - l[node]) : 0;
    rep(c,0,ALPHA) if (t[node][c] != -1)
      mask |= lcs(t[node][c], i1, i2, len);
    if (mask == 3)
      best = max(best, {len, r[node] - len});
    return mask;
  }
  static pii LCS(string s, string t) {
    SuffixTree st(s + (char)('z' + 1) + t + (char)('z' + 2));
    st.lcs(0, sz(s), sz(s) + 1 + sz(t), 0);
    return st.best;
  }
};
```

Hashing.h

Description: Various self-explanatory methods for string hashing. 45 lines

```
typedef unsigned long long H;
static const H C = 123891739; // arbitrary

// Arithmetic mod 2^64-1. 5x slower than mod 2^64 and more
// code, but works on evil test data (e.g. Thue-Morse).
// "typedef H K;" instead if you think test data is random.
struct K {
  typedef __uint128_t H2;
  H x; K(H x=0) : x(x) {}
  K operator+(K o){ return x + o.x + H(((H2)x + o.x)>>64); }
  K operator*(K o){ return K(x*o.x)+ H(((H2)x * o.x)>>64); }
  H operator-(K o) { K a = *this + ~o.x; return a.x + !~a.x; }
};

struct HashInterval {
  vector<K> ha, pw;
  HashInterval(string& str) : ha(sz(str)+1), pw(ha) {
    pw[0] = 1;
    rep(i,0,sz(str))
      ha[i+1] = ha[i] * C + str[i],
      pw[i+1] = pw[i] * C;
  }
  H hashInterval(int a, int b) { // hash [a, b)
    return ha[b] - ha[a] * pw[b - a];
  }
};

vector<H> getHashes(string& str, int length) {
  if (sz(str) < length) return {};
  K h = 0, pw = 1;
  rep(i,0,length)
    h = h * C + str[i], pw = pw * C;
  vector<H> ret = {h - 0};
  rep(i,length,sz(str)) {
    ret.push_back(h * C + str[i] - pw * str[i-length]);
    h = ret.back();
  }
  return ret;
}

H hashString(string& s) {
  K h = 0;
  trav(c, s) h = h * C + c;
  return h - 0;
}
```

AhoCorasick.h

Description: Aho-Corasick tree is used for multiple pattern matching. Initialize the tree with create(patterns). find(word) returns for each position the index of the longest word that ends there, or -1 if none. findAll(_, word) finds all words (up to $N\sqrt{N}$ many if no duplicate patterns) that start at each position (shortest first). Duplicate patterns are allowed; empty patterns are not. To find the longest words that start at each position, reverse all input.
Time: Function create is $\mathcal{O}(26N)$ where N is the sum of length of patterns. find is $\mathcal{O}(M)$ where M is the length of the word. findAll is $\mathcal{O}(NM)$. 67 lines

```
struct AhoCorasick {
  enum {alpha = 26, first = 'A'};
  struct Node {
    // (nmatches is optional)
    int back, next[alpha], start = -1, end = -1, nmatches = 0;
    Node(int v) { memset(next, v, sizeof(next)); }
  };
  vector<Node> N;
  vector<int> backp;
  void insert(string& s, int j) {
    assert(!s.empty());
    int n = 0;
    trav(c, s) {
      int& m = N[n].next[c - first];
      if (m == -1) { n = m = sz(N); N.emplace_back(-1); }
      else n = m;
    }
    if (N[n].end == -1) N[n].start = j;
    backp.push_back(N[n].end);
    N[n].end = j;
    N[n].nmatches++;
  }
  AhoCorasick(vector<string>& pat) {
    N.emplace_back(-1);
    rep(i,0,sz(pat)) insert(pat[i], i);
    N[0].back = sz(N);
    N.emplace_back(0);

    queue<int> q;
    for (q.push(0); !q.empty(); q.pop()) {
      int n = q.front(), prev = N[n].back;
      rep(i,0,alpha) {
        int &ed = N[n].next[i], y = N[prev].next[i];
        if (ed == -1) ed = y;
        else {
          N[ed].back = y;
          (N[ed].end == -1 ? N[ed].end : backp[N[ed].start])
            = N[y].end;
          N[ed].nmatches += N[y].nmatches;
          q.push(ed);
        }
      }
    }
  }
  vi find(string word) {
    int n = 0;
    vi res; // ll count = 0;
    trav(c, word) {
      n = N[n].next[c - first];
      res.push_back(N[n].end);
      // count += N[n].nmatches;
    }
    return res;
```

```
  }
  vector<vi> findAll(vector<string>& pat, string word) {
    vi r = find(word);
    vector<vi> res(sz(word));
    rep(i,0,sz(word)) {
      int ind = r[i];
      while (ind != -1) {
        res[i - sz(pat[ind]) + 1].push_back(ind);
        ind = backp[ind];
      }
    }
    return res;
  }
};
```

Various (10)

10.1 Intervals

IntervalContainer.h

Description: Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).
Time: $\mathcal{O}(\log N)$

25 lines

```
template <class T>
auto addInterval(set<pair<T, T>>& is, T L, T R) {
  if (L == R) return is.end();
  auto it = is.lower_bound({L, R}), before = it;
  while (it != is.end() && it->first <= R) {
    R = max(R, it->second);
    before = it = is.erase(it);
  }
  if (it != is.begin() && (--it)->second >= L) {
    L = min(L, it->first);
    R = max(R, it->second);
    is.erase(it);
  }
  return is.insert(before, {L,R});
};

template <class T>
void removeInterval(set<pair<T, T>>& is, T L, T R) {
  if (L == R) return;
  auto it = addInterval(is, L, R);
  T r2 = it->second;
  if (it->first == L) is.erase(it);
  else (T&)it->second = L;
  if (R != r2) is.emplace(R, r2);
};
```

IntervalCover.h

Description: Compute indices of smallest set of intervals covering another interval. Intervals should be [inclusive, exclusive). To support [inclusive, inclusive], change (A) to add `|| R.empty()`. Returns empty set on failure (or if G is empty).
Time: $\mathcal{O}(N \log N)$

19 lines

```
template<class T>
vi cover(pair<T, T> G, vector<pair<T, T>> I) {
  vi S(sz(I)), R;
  iota(all(S), 0);
  sort(all(S), [&](int a, int b) { return I[a] < I[b]; });
  T cur = G.first;
  int at = 0;
  while (cur < G.second) { // (A)
    pair<T, int> mx = make_pair(cur, -1);
    while (at < sz(I) && I[S[at]].first <= cur) {
      mx = max(mx, make_pair(I[S[at]].second, S[at]));
      at++;
    }
    if (mx.second == -1) return {};
    cur = mx.first;
    R.push_back(mx.second);
  }
  return R;
}
```

ConstantIntervals.h

Description: Split a monotone function on [from, to) into a minimal set of half-open intervals on which it has the same value. Runs a callback g for each such interval.
Usage: `constantIntervals(0, sz(v), [&](int x){return v[x];}, [&](int lo, int hi, T val){...});`
Time: $\mathcal{O}\left(k \log \frac{n}{k}\right)$

19 lines

```
template<class F, class G, class T>
void rec(int from, int to, F f, G g, int& i, T& p, T q) {
  if (p == q) return;
  if (from == to) {
    g(i, to, p);
    i = to; p = q;
  } else {
    int mid = (from + to) >> 1;
    rec(from, mid, f, g, i, p, f(mid));
    rec(mid+1, to, f, g, i, p, q);
  }
}
template<class F, class G>
void constantIntervals(int from, int to, F f, G g) {
  if (to <= from) return;
  int i = from; auto p = f(i), q = f(to-1);
  rec(from, to-1, f, g, i, p, q);
  g(i, to, q);
}
```

10.2 Misc. algorithms

TernarySearch.h

Description: Find the smallest i in $[a, b]$ that maximizes $f(i)$, assuming that $f(a) < \ldots < f(i) \geq \cdots \geq f(b)$. To reverse which of the sides allows non-strict inequalities, change the $<$ marked with (A) to $<=$, and reverse the loop at (B). To minimize f, change it to $>$, also at (B).
Usage: `int ind = ternSearch(0,n-1,[&](int i){return a[i];});`
Time: $\mathcal{O}(\log(b-a))$

13 lines

```
template<class F>
int ternSearch(int a, int b, F f) {
  assert(a <= b);
  while (b - a >= 5) {
    int mid = (a + b) / 2;
    if (f(mid) < f(mid+1)) // (A)
      a = mid;
    else
      b = mid+1;
  }
  rep(i,a+1,b+1) if (f(a) < f(i)) a = i; // (B)
  return a;
}
```

Karatsuba.h

Description: Faster-than-naive convolution of two sequences: $c[x] = \sum a[i]b[x-i]$. Uses the identity $(aX+b)(cX+d) = acX^2 + bd + ((a+c)(b+d) - ac - bd)X$. Doesn't handle sequences of very different length well. See also FFT, under the Numerical chapter.
Time: $\mathcal{O}\left(N^{1.6}\right)$

1 lines

LIS.h

Description: Compute indices for the longest increasing subsequence.
Time: $\mathcal{O}(N \log N)$

17 lines

```
template<class I> vi lis(vector<I> S) {
  vi prev(sz(S));
  typedef pair<I, int> p;
  vector<p> res;
  rep(i,0,sz(S)) {
    p el { S[i], i };
    //S[i]+1 for non-decreasing
    auto it = lower_bound(all(res), p { S[i], 0 });
    if (it == res.end()) res.push_back(el), it = --res.end();
    *it = el;
    prev[i] = it==res.begin() ?0:(it-1)->second;
  }
  int L = sz(res), cur = res.back().second;
  vi ans(L);
  while (L--) ans[L] = cur, cur = prev[cur];
  return ans;
}
```

LCS.h

Description: Finds the longest common subsequence.
Memory: $\mathcal{O}(nm)$.
Time: $\mathcal{O}(nm)$ where n and m are the lengths of the sequences.

14 lines

```
template <class T> T lcs(const T &X, const T &Y) {
  int a = sz(X), b = sz(Y);
  vector<vi> dp(a+1, vi(b+1));
  rep(i,1,a+1) rep(j,1,b+1)
    dp[i][j] = X[i-1]==Y[j-1] ? dp[i-1][j-1]+1 :
      max(dp[i][j-1],dp[i-1][j]);
  int len = dp[a][b];
  T ans(len,0);
  while(a && b)
    if(X[a-1]==Y[b-1]) ans[--len] = X[--a], --b;
    else if(dp[a][b-1]>dp[a-1][b]) --b;
    else --a;
  return ans;
}
```

10.3 Dynamic programming

DivideAndConquerDP.h

Description: Given $a[i] = \min_{lo(i) \le k < hi(i)}(f(i,k))$ where the (minimal) optimal k increases with i, computes $a[i]$ for $i = L..R-1$.
Time: $\mathcal{O}\left((N + (hi - lo))\log N\right)$

18 lines

```
struct DP { // Modify at will:
  int lo(int ind) { return 0; }
  int hi(int ind) { return ind; }
  ll f(int ind, int k) { return dp[ind][k]; }
  void store(int ind, int k, ll v) { res[ind] = pii(k, v); }

  void rec(int L, int R, int LO, int HI) {
    if (L >= R) return;
    int mid = (L + R) >> 1;
    pair<ll, int> best(LLONG_MAX, LO);
    rep(k, max(LO,lo(mid)), min(HI,hi(mid)))
      best = min(best, make_pair(f(mid, k), k));
    store(mid, best.second, best.first);
    rec(L, mid, LO, best.second+1);
    rec(mid+1, R, best.second, HI);
  }
  void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
};
```

KnuthDP.h

Description: When doing DP on intervals: $a[i][j] = \min_{i<k<j}(a[i][k] + a[k][j]) + f(i,j)$, where the (minimal) optimal k increases with both i and j, one can solve intervals in increasing order of length, and search $k = p[i][j]$ for $a[i][j]$ only between $p[i][j-1]$ and $p[i+1][j]$. This is known as Knuth DP. Sufficient criteria for this are if $f(b,c) \le f(a,d)$ and $f(a,c) + f(b,d) \le f(a,d) + f(b,c)$ for all $a \le b \le c \le d$. Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.
Time: $\mathcal{O}\left(N^2\right)$

1 lines

10.4 Debugging tricks

- `signal(SIGSEGV, [](int) { _Exit(0); });` converts segfaults into Wrong Answers. Similarly one can catch SIGABRT (assertion failures) and SIGFPE (zero divisions). `_GLIBCXX_DEBUG` violations generate SIGABRT (or SIGSEGV on gcc 5.4.0 apparently).
- `feenableexcept(29);` kills the program on NaNs (1), 0-divs (4), infinities (8) and denormals (16).

10.5 Optimization tricks

10.5.1 Bit hacks

- `x & -x` is the least bit in `x`.
- `for (int x = m; x; ) { --x &= m; ... }` loops over all subset masks of `m` (except `m` itself).
- `c = x&-x, r = x+c; (((r^x) >> 2)/c) | r` is the next number after `x` with the same number of bits set.
- `rep(b,0,K) rep(i,0,(1 << K)) if (i & 1 << b) D[i] += D[i^(1 << b)];` computes all sums of subsets.

10.5.2 Pragmas

- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize for loops and optimizes floating points better (assumes associativity and turns off denormals).
- `#pragma GCC target ("avx,avx2")` can double performance of vectorized code, but causes crashes on old machines.
- `#pragma GCC optimize ("trapv")` kills the program on integer overflows (but is really slow).

BumpAllocator.h

Description: When you need to dynamically allocate many objects and don't care about freeing them. "new X" otherwise has an overhead of something like 0.05us + 16 bytes per allocation.

8 lines

```
// Either globally or in a single class:
static char buf[450 << 20];
void* operator new(size_t s) {
  static size_t i = sizeof buf;
  assert(s < i);
  return (void*)&buf[i -= s];
}
void operator delete(void*) {}
```

SmallPtr.h

Description: A 32-bit pointer that points into BumpAllocator memory.

"BumpAllocator.h" 10 lines

```
template<class T> struct ptr {
  unsigned ind;
  ptr(T* p = 0) : ind(p ? unsigned((char*)p - buf) : 0) {
    assert(ind < sizeof buf);
  }
  T& operator*() const { return *(T*)(buf + ind); }
  T* operator->() const { return &**this; }
  T& operator[](int a) const { return (&**this)[a]; }
  explicit operator bool() const { return ind; }
};
```

BumpAllocatorSTL.h

Description: BumpAllocator for STL containers.
Usage: `vector<vector<int, small<int>>> ed(N);`

14 lines

```
char buf[450 << 20] alignas(16);
size_t buf_ind = sizeof buf;

template <class T> struct small {
  typedef T value_type;
  small() {}
  template <class U> small(const U&) {}
  T* allocate(size_t n) {
    buf_ind -= n * sizeof(T);
    buf_ind &= 0 - alignof(T);
    return (T*)(buf + buf_ind);
  }
  void deallocate(T*, size_t) {}
};
```

Unrolling.h

5 lines

```
#define F {...; ++i;}
int i = from;
while (i&3 && i < to) F // for alignment, if needed
while (i + 4 <= to) { F F F F }
while (i < to) F
```

SIMD.h

Description: Cheat sheet of SSE/AVX intrinsics, for doing arithmetic on several numbers at once. Can provide a constant factor improvement of about 4, orthogonal to loop unrolling. Operations follow the pattern "_mm(256)?_name_(si(128|256)|epi(8|16|32|64)|pd|ps)". Not all are described here; grep for _mm_ in /usr/lib/gcc/*/4.9/include/ for more. If AVX is unsupported, try 128-bit operations, "emmintrin.h" and #define __SSE__ and __MMX__ before including it. For aligned memory use _mm_malloc(size, 32) or int buf[N] alignas(32), but prefer loadu/storeu.

43 lines

```
#pragma GCC target ("avx2") // or sse4.1
#include "immintrin.h"

typedef __m256i mi;
#define L(x) _mm256_loadu_si256((mi*)&(x))

// High-level/specific methods:
// load(u)?_si256, store(u)?_si256, setzero_si256, _mm_malloc
// blendv_(epi8|ps|pd) (z?y:x), movemask_epi8 (hibits of bytes)
// i32gather_epi32(addr, x, 4): map addr[] over 32-b parts of x
// sad_epu8: sum of absolute differences of u8, outputs 4xi64
// maddubs_epi16: dot product of unsigned i7's, outputs 16xi15
// madd_epi16: dot product of signed i16's, outputs 8xi32
// extractf128_si256(, i) (256->128), cvtsi128_si32 (128->lo32)
// permute2f128_si256(x,x,1) swaps 128-bit lanes
// shuffle_epi32(x, 3*64+2*16+1*4+0) == x for each lane
// shuffle_epi8(x, y) takes a vector instead of an imm

// Methods that work with most data types (append e.g. _epi32):
// set1, blend (i8?x:y), add, adds (sat.), mullo, sub, and/or,
// andnot, abs, min, max, sign(1,x), cmp(gt|eq), unpack(lo|hi)

int sumi32(mi m) { union {int v[8]; mi m;} u; u.m = m;
  int ret = 0; rep(i,0,8) ret += u.v[i]; return ret; }
mi zero() { return _mm256_setzero_si256(); }
mi one() { return _mm256_set1_epi32(-1); }
bool all_zero(mi m) { return _mm256_testz_si256(m, m); }
bool all_one(mi m) { return _mm256_testc_si256(m, one()); }

ll example_filteredDotProduct(int n, short* a, short* b) {
  int i = 0; ll r = 0;
  mi zero = _mm256_setzero_si256(), acc = zero;
  while (i + 16 <= n) {
    mi va = L(a[i]), vb = L(b[i]); i += 16;
    va = _mm256_and_si256(_mm256_cmpgt_epi16(vb, va), va);
    mi vp = _mm256_madd_epi16(va, vb);
    acc = _mm256_add_epi64(_mm256_unpacklo_epi32(vp, zero),
      _mm256_add_epi64(acc, _mm256_unpackhi_epi32(vp, zero)));
  }
  union {ll v[4]; mi m;} u; u.m = acc; rep(i,0,4) r += u.v[i];
  for (;i<n;++i) if (a[i] < b[i]) r += a[i]*b[i]; // <- equiv
  return r;
}
```

Techniques (A)

techniques.txt 159 lines

```
Recursion
Divide and conquer
  Finding interesting points in N log N
Algorithm analysis
  Master theorem
  Amortized time complexity
Greedy algorithm
  Scheduling
  Max contigous subvector sum
  Invariants
  Huffman encoding
Graph teory
  Dynamic graphs (extra book-keeping)
  Breadth first search
  Depth first search
  * Normal trees / DFS trees
  Dijkstra's algoritm
  MST: Prim's algoritm
  Bellman-Ford
  Konig's theorem and vertex cover
  Min-cost max flow
  Lovasz toggle
  Matrix tree theorem
  Maximal matching, general graphs
  Hopcroft-Karp
  Hall's marriage theorem
  Graphical sequences
  Floyd-Warshall
  Eulercykler
  Flow networks
  * Augumenting paths
  * Edmonds-Karp
  Bipartite matching
  Min. path cover
  Topological sorting
  Strongly connected components
  2-SAT
  Cutvertices, cutedges och biconnected components
  Edge coloring
  * Trees
  Vertex coloring
  * Bipartite graphs (=> trees)
  * 3^n (special case of set cover)
  Diameter and centroid
  K'th shortest path
  Shortest cycle
Dynamic programmering
  Knapsack
  Coin change
  Longest common subsequence
  Longest increasing subsequence
  Number of paths in a dag
  Shortest path in a dag
  Dynprog over intervals
  Dynprog over subsets
  Dynprog over probabilities
  Dynprog over trees
  3^n set cover
  Divide and conquer
  Knuth optimization
  Convex hull optimizations
  RMQ (sparse table a.k.a 2^k-jumps)
  Bitonic cycle
  Log partitioning (loop over most restricted)
Combinatorics
  Computation of binomial coefficients
  Pigeon-hole principle
  Inclusion/exclusion
  Catalan number
  Pick's theorem
Number theory
  Integer parts
  Divisibility
  Euklidean algorithm
  Modular arithmetic
  * Modular multiplication
  * Modular inverses
  * Modular exponentiation by squaring
  Chinese remainder theorem
  Fermat's small theorem
  Euler's theorem
  Phi function
  Frobenius number
  Quadratic reciprocity
  Pollard-Rho
  Miller-Rabin
  Hensel lifting
  Vieta root jumping
Game theory
  Combinatorial games
  Game trees
  Mini-max
  Nim
  Games on graphs
  Games on graphs with loops
  Grundy numbers
  Bipartite games without repetition
  General games without repetition
  Alpha-beta pruning
Probability theory
Optimization
  Binary search
  Ternary search
  Unimodality and convex functions
  Binary search on derivative
Numerical methods
  Numeric integration
  Newton's method
  Root-finding with binary/ternary search
  Golden section search
Matrices
  Gaussian elimination
  Exponentiation by squaring
Sorting
  Radix sort
Geometry
  Coordinates and vectors
  * Cross product
  * Scalar product
  Convex hull
  Polygon cut
  Closest pair
  Coordinate-compression
  Quadtrees
  KD-trees
  All segment-segment intersection
Sweeping
  Discretization (convert to events and sweep)
  Angle sweeping
  Line sweeping
  Discrete second derivatives
Strings
  Longest common substring
  Palindrome subsequences
  Knuth-Morris-Pratt
  Tries
  Rolling polynom hashes
  Suffix array
  Suffix tree
  Aho-Corasick
  Manacher's algorithm
  Letter position lists
Combinatorial search
  Meet in the middle
  Brute-force with pruning
  Best-first (A*)
  Bidirectional search
  Iterative deepening DFS / A*
Data structures
  LCA (2^k-jumps in trees in general)
  Pull/push-technique on trees
  Heavy-light decomposition
  Centroid decomposition
  Lazy propagation
  Self-balancing trees
  Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
  Monotone queues / monotone stacks / sliding queues
  Sliding queue using 2 stacks
  Persistent segment tree
```