NCTU_Yggdarsill

Contents

1 Building Environment

1.1 C++11

```
{
    "shell_cmd": "g++ -std=c++11 -Wall \"${file}\" -o \"${file_path}/${
    file_base_name}\"",
    "file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)?:? (.*)$",
    "working_dir": "${file_path}",
    "selector": "source.c, source.c++",

    "variants":
    [
        {
            "name": "Run",
            "shell_cmd": "g++ -std=c++11 -Wall \"${file}\" -o \"${file_path}/
    ${file_base_name}\" && gnome-terminal -e 'bash -c \"${file_path}/${
    file_base_name}; echo Press ENTER to continue; read line; exit; exec bash
    \"'"
        }
    ]
}
```

1.2 Default

```
#define F(n) Fi(i,n)
#define Fi(i,n) Fl(i,0,n)
#define Fl(i,l,n) for(int i=(l);i<(int)(n);++i)
#include <bits/stdc++.h>
#include <bits/extc++.h>
// #include <ext/pb_ds/assoc_container.hpp>
// #include <ext/pb_ds/priority_queue.hpp>
```

```
using namespace std;
using namespace __gnu_pbds;
const double PI = acos(-1);
main(){
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cout << fixed << setprecision(7) << PI << endl;
}
```

1.3 Preferences

```
{
  "color_scheme": "Packages/Color Scheme - Default/Monokai Bright.tmTheme",
  // "font_face": "Courier New",  // Uncomment if defaults is proportional.
  "font_size": 18
}
```

1.4 Print File

```
import sublime, sublime_plugin
import os

class print_file(sublime_plugin.TextCommand):
  def run(self, edit):
    os.system('cat -n "%s" > tmp.print; lpr tmp.print' % self.view.file_name())
    self.view.show_popup("JIZZ!!")
```

1.5 Vimrc

```
set tabstop=4
set autoindent

map <F9> :w<LF>:!g++ -O2 -std=c++11 -o %.out % && echo "----Start----" && ./%.out<LF>
imap <F9> <ESC><F9>
```

2 Convolution

2.1 FFT

```
#include <cstdio>
#include <cstring>
#include <cmath>
const double PI=acos(-1.0);
typedef  struct {
    double  real;
    double  im;
} COMPLEX;
COMPLEX X[66000],Y[66000],A[66000];
COMPLEX EE(COMPLEX a,COMPLEX b)
{
    COMPLEX c;
    c.real=a.real*b.real-a.im*b.im;
    c.im=a.real*b.im+a.im*b.real;
    return c;
}
/* 1 FFT , -1 IFFT */
void fft(COMPLEX x[],int nfft,int isign)
{
    int i,j=0,k;
    COMPLEX t;
    for(i=1, j = nfft / 2;i<nfft-1;i++)
    {
        if(i<j)
        {
            t=x[j];
            x[j]=x[i];
            x[i]=t;
        }
        k=nfft/2;
        while(k<=j)
        {
            j-=k;
            k/=2;
        }
        if (j < k)
            j+=k;
    }
    int le,lei,ip;
    COMPLEX u,w, v;
    for(le=2;le<=nfft;le *= 2)
    {
        lei=le/2;
        w.real=cos(2.0*PI*isign/le);
        w.im=sin(2.0*PI*isign/le);
        for(i=0;i<nfft;i+=le)
        {
            u.real=1.0;
            u.im=0.0;
            for(j = i ; j < i + lei ; ++j)
            {
                ip=j+lei;
                v = x[j];
                t=EE(u, x[ip]);
                x[j].real=v.real+t.real;
                x[j].im=v.im+t.im;
                x[ip].real=v.real-t.real;
                x[ip].im=v.im-t.im;
                u=EE(u,w);
            }
        }
    }
}
void FFT(COMPLEX x[], int nfft)
{
    fft(x,nfft,1);
}
void IFFT(COMPLEX x[],int nfft)
{
    int i;
    fft(x,nfft,-1);

    for(i=0;i<nfft;i++)
    {
        x[i].real /= nfft;
        x[i].im /= nfft;
    }
}
int main(void) {
    int t_num;
    int i,ii,iii;
    int p_num;
    int Nx;
    int NFFT;
    int temp;
    scanf("%d",&t_num);
    for(i=0;i<t_num;i++){
        scanf("%d",&p_num);
        Nx=p_num*2-1;
        NFFT = 2 << (int)log2(Nx);
        for(ii=0;ii<p_num;++ii){
            scanf("%d",&temp);
            X[ii].real=(double)temp;
            X[ii].im=0.0;
        }
        for(iii=0;iii<p_num;++iii){

            scanf("%d",&temp);
            Y[iii].real=(double)temp;
            Y[iii].im=0.0;
        }
        for(ii=p_num;ii<NFFT;ii++)
        {
            X[ii].real=0.0;
            X[ii].im=0.0;
            Y[ii].real=0.0;
            Y[ii].im=0.0;
        }
        FFT(X,NFFT);
        FFT(Y,NFFT);
        for(ii=0;ii<NFFT;ii++){
            A[ii] = EE(X[ii], Y[ii]);
        }
        IFFT(A,NFFT);
        for(ii=0;ii<Nx;ii++){
            printf("%d ", (int)round(A[ii].real));
        }
        printf("\n");
    }
    return 0;
}
```

3 Geometry

3.1 Geometry

```
const double eps = 1e-10;
const double INF = 1.0/0.0;
const double SIDE = 10000;
const double PI = acos(-1.0);
const int MAXN = 500000 + 10;
struct PT{
    double x,y;
    PT(){}
    PT(double x,double y):x(x),y(y){}
    PT operator + (const PT& p)const{
        return PT(x+p.x,y+p.y);
    }
    PT operator - (const PT& p)const{
        return PT(x-p.x,y-p.y);
    }
    PT operator * (double c)const{
        return PT(x*c,y*c);
    }
    PT operator / (double c)const{
        return PT(x/c,y/c);
    }
  PT rot(double a)const{return PT(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a));}
    double operator *(const PT& p)const{
        return x*p.x+y*p.y;
    }
    double operator ^(const PT& p)const{
        return x*p.y-y*p.x;
    }
    bool operator ==(const PT& p)const{
        return fabs(x-p.x)<eps&&fabs(y-p.y)<eps;
    }
  double len2()const{return x*x+y*y;}
  double len()const{return sqrt(len2());}
}poi[MAXN],stk[MAXN];
struct LINE{
    PT a,b;
    double angle;
    LINE(){}
```

```cpp
    LINE(PT _a,PT _b):a(_a),b(_b),angle(atan2(_b.y-_a.y,_b.x-_a.x)){}
}line[MAXN],deq[MAXN];
int top;
inline int ori(const PT& p1,const PT& p2,const PT& p3){
    double a=(p2-p1)^(p3-p1);
    if(a>-eps&&a<eps)return 0;
    return a>0 ? 1:-1;
}
inline bool btw(const PT& p1,const PT& p2,const PT& p3){
    return (p2-p1)*(p3-p1)<eps;
}
//segment intersection
inline bool intersection(const PT& p1,const PT& p2,const PT& p3,const PT& p4){
    int a123=ori(p1,p2,p3);
    int a124=ori(p1,p2,p4);
    int a341=ori(p3,p4,p1);
    int a342=ori(p3,p4,p2);
    if(a123==0&&a124==0)return btw(p1,p3,p4)||btw(p2,p3,p4)||btw(p3,p1,p2)||btw(p4,p1,p2);
    return a123*a124 <= 0 && a341*a342 <= 0;
}
inline PT intersectionPoint(const PT& p1,const PT& p2,const PT& p3,const PT& p4){
    double a123=(p2-p1)^(p3-p1);
    double a124=(p2-p1)^(p4-p1);
    return (p4*a123-p3*a124)/(a123-a124);
}
//line intersection
inline PT intersectionPoint(const LINE& l1,const LINE& l2){
    PT p1=l1.a,p2=l1.b,p3=l2.a,p4=l2.b;
    double a123=(p2-p1)^(p3-p1);
    double a124=(p2-p1)^(p4-p1);
    return (p4*a123-p3*a124)/(a123-a124);
}
PT foot(const LINE& l,const PT& p){
  PT m(l.b.y-l.a.y,l.a.x-l.b.x);
  return p+m*(l.a-p ^ l.b-p)/((l.b-l.a).len2());
}
PT mirror(const LINE& l,const PT& p){
  PT m(l.b.y-l.a.y,l.a.x-l.b.x);
  return p+m*(l.a-p ^ l.b-p)/((l.b-l.a).len2())*2;
}
//segment-point distance
inline double sp_dis(PT a,PT l1,PT l2){
    if((a-l1)*(l2-l1)<0) return (l1-a).len();
  else if((a-l2)*(l1-l2)<0) return (l2-a).len();
    return fabs(l1-a^l2-a)/((l2-l1).len());
}

struct cir{
    point c;
    double r;
}o[10];
double out_ang(cir a,cir b){    //a.c+(b.c-a.c).unit().rot(ang)*b.r
    return acos((a.r-b.r)/(a.c-b.c).len());
}
double in_ang(cir a,cir b){
    return acos((a.r+b.r)/(a.c-b.c).len());
}
int main(){
  double tmp,sum;
  if(fabs(o[i].r-o[j].r)<(o[j].c-o[i].c).len()){
    tmp = out_ang(o[i],o[j]);
    sum = ang_add(cl,tmp);
    pi=o[i].c+point(o[i].r*cos(sum),o[i].r*sin(sum));
    pj=o[j].c+point(o[j].r*cos(sum),o[j].r*sin(sum));
    sum = ang_add(cl,-tmp);
    pi=o[i].c+point(o[i].r*cos(sum),o[i].r*sin(sum));
    pj=o[j].c+point(o[j].r*cos(sum),o[j].r*sin(sum));
  }
  if(o[i].r+o[j].r<(o[j].c-o[i].c).len()){
    tmp = in_ang(o[i],o[j]);
    sum = ang_add(cl,tmp);
    pi=o[i].c+point(o[i].r*cos(sum),o[i].r*sin(sum));
    pj=o[j].c-point(o[j].r*cos(sum),o[j].r*sin(sum));
    sum = ang_add(cl,-tmp);
    pi=o[i].c+point(o[i].r*cos(sum),o[i].r*sin(sum));
    pj=o[j].c-point(o[j].r*cos(sum),o[j].r*sin(sum));
  }
}

inline double dist(const PT& p1,const PT& p2){
    return sqrt((p2-p1)*(p2-p1));
}
inline double tri(const PT& p1,const PT& p2,const PT& p3){
    return fabs((p2-p1)^(p3-p1));
}
inline double getPerimeter(){
    double res=0.0;
    poi[top++]=poi[0];
    for(int i=0;i<top-1;i++)res+=dist(poi[i],poi[i+1]);
    return res;
}
inline double getarea(){
    double res=0.0;
    for(int i=1;i<top-1;i++)res+=tri(poi[0],poi[i],poi[i+1]);
    return 0.5*res;
}

//convex hull
inline bool cmp_convex(const PT &a,const PT &b){
    if(a.x!=b.x)return a.x<b.x;
    return a.y<b.y;
}
inline void convex_hull(PT a[],int &n){
    top=0;
    sort(a,a+n,cmp_convex);
    for(int i=0;i<n;i++){
        while(top>=2&&ori(stk[top-2],stk[top-1],a[i])>=0)top--;
        stk[top++]=a[i];
    }
    for(int i=n-2,t=top+1;i>=0; i--){
        while(top>=t&&ori(stk[top-2],stk[top-1],a[i])>=0)top--;
        stk[top++]=a[i];
    }
    top--;
    for(int i=0;i<top;i++)poi[i]=stk[i];
}
//half plane intersection
inline bool cmp_half_plane(const LINE &a,const LINE &b){
    if(fabs(a.angle-b.angle)<eps)return ori(a.a,a.b,b.a)<0;
    return a.angle > b.angle;
}
inline void half_plane_intersection(LINE a[],int &n){
    int m=1,front=0,rear=1;
    sort(a,a+n,cmp_half_plane);
    for(int i=1;i<n;i++){
        if(fabs(a[i].angle-a[m-1].angle)>eps)a[m++]=a[i];
    }
    deq[0]=a[0],deq[1]=a[1];
    for(int i=2;i<m;i++){
        while(front<rear&&ori(a[i].a,a[i].b,intersectionPoint(deq[rear],deq[rear-1]))<0)rear--;
        while(front<rear&&ori(a[i].a,a[i].b,intersectionPoint(deq[front],deq[front+1]))<0)front++;
        deq[++rear]=a[i];
    }
  while(front<rear&&ori(deq[front].a,deq[front].b,intersectionPoint(deq[rear],deq[rear-1]))<0)rear--;
    while(front<rear&&ori(deq[rear].a,deq[rear].b,intersectionPoint(deq[front],deq[front+1]))<0)front++;
    if(front==rear)return;

    top=0;
    for(int i=front;i<rear;i++)poi[top++]=intersectionPoint(deq[i],deq[i+1]);
    if(rear>front+1)poi[top++]=intersectionPoint(deq[front],deq[rear]);
}




//smallest cover rectangle
double ans1,ans2;
void rotating_calipers(){
    ans1=ans2=INF;
    int j=1,k=1,l=1;
    poi[top]=poi[0];
    for(int i=0;i<top;i++){
        while(tri(poi[i],poi[i+1],poi[j])<tri(poi[i],poi[i+1],poi[j+1])) j=(j+1)%top;
        while(((poi[i+1]-poi[i])*(poi[k+1]-poi[k]))>eps)k=(k+1)%top;
        if(i==0)l=(k+1)%top;
        while(((poi[i+1]-poi[i])*(poi[l+1]-poi[l]))<-eps)l=(l+1)%top;
        double tmp1 = tri(poi[i],poi[i+1],poi[j])/dist(poi[i],poi[i+1]);
        double tmp2 = (((poi[k]-poi[i])*(poi[i+1]-poi[i]))-((poi[l]-poi[i])*(poi[i+1]-poi[i])))/dist(poi[i],poi[i+1]);
        if((tmp1+tmp2)*2.0<ans1)ans1=(tmp1+tmp2)*2.0;
        if(tmp1*tmp2<ans2)ans2=tmp1*tmp2;
    }
}
int main(){
    int n,m;
    while(~scanf("%d",&n)&&n){
        for(int i=0;i<n;i++)scanf("%lf%lf",&poi[i].x,&poi[i].y);
        convex_hull(poi,n);
        rotating_calipers();
        printf("%.2f %.2f\n",ans2,ans1);
    }
}

inline bool online(const LINE &L,const PT &p){
    return ori(p,L.a,L.b)==0&&btw(p,L.a,L.b);
}
inline bool on_convex(const PT& p){
    for(int i=0;i<top;i++)
        if(p==poi[i])return 1;
    poi[top]=poi[0];
    for(int i=0;i<top;i++){
        line[i].a=poi[i];
        line[i].b=poi[i+1];
    }
    for(int i=0;i<top;i++)
        if(online(line[i],p))return 1;
    return 0;
}
//originally in long long, should be modified
bool in_simple_polygon(PT b[],int k){
  bool flag=false;
  for(int j=0;j<k;j++){
    if(((p-b[j])^(p-b[(j+1)%k]))==0&&(p-b[j])*(p-b[(j+1)%k])<=0){
      flag=true;
      break;
    }
    if((b[j].y<p.y)^(b[(j+1)%k].y<p.y)){
      long long xss=(b[j]-p)^(b[(j+1)%k]-p);
      if((xss<0)^(b[j].y<b[(j+1)%k].y)){
        flag^=1;
      }
    }
  }
  return flag;
}
```

3.2 MinimumCoveringCircle

```
#define F(n) Fi(i,n)
#define Fi(i,n) Fl(i,0,n)
#define Fl(i,l,n) for(int i=(l);i<(int)(n);++i)
#include <bits/stdc++.h>
using namespace std;
const double eps = 1e-6;
```

```cpp
#define x first
#define y second
typedef pair<double, double> point;
inline double dq(const point& p1, const point& p2) {
  return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
inline point oc(const point& pa, const point& pb, const point& pc) {
  double a, b, c, d, e, f, delta, dx, dy;
  // ax + by = c
  // dx + ey = f
  a = pa.x - pb.x;
  b = pa.y - pb.y;
  c = a*(pa.x+pb.x)/2 + b*(pa.y+pb.y)/2;
  d = pa.x - pc.x;
  e = pa.y - pc.y;
  f = d*(pa.x+pc.x)/2 + e*(pa.y+pc.y)/2;
  delta = a*e-b*d;
  dx = c*e-f*b;
  dy = a*f-d*c;
  return point(dx/delta, dy/delta);
}
inline point enc(const vector<point>& tmp) {
  point O = tmp[0];
  double r = 0;
  Fl(i, 1, tmp.size()) if (dq(O, tmp[i]) - r > eps) {
    O = tmp[i], r = 0;
    Fi(j, i) if (dq(O, tmp[j]) - r > eps) {
      O = point((tmp[i].x+tmp[j].x)/2, (tmp[i].y+tmp[j].y)/2);
      r = dq(O, tmp[j]);
      Fi(k, j) if (dq(O, tmp[k]) - r > eps)
        O = oc(tmp[i], tmp[j], tmp[k]), r = dq(O, tmp[k]);
    }
  }
  return O;
}
int n;
vector<point> v;
int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  while (cin >> n) {
    if (!n) break;
    v.clear();
    F(n) {
      point tp;
      cin >> tp.x >> tp.y;
      v.push_back(tp);
    }
    point ct = enc(v);
    cout << setprecision(2) << fixed << ct.x << ' ' << ct.y << ' ' << dq(ct, v[0]) << '\n';
  }
}
```

4 GNU Black Magic

4.1 Black Magic

```cpp
#include<ext/rope>
using namespace std;
using namespace __gnu_cxx;
const int MAXN = 50000 + 10;
crope ro,l[MAXN],tmp;
char str[200+10];
main(){
    int T,op,p,c,d=0,cnt=1,v;
    scanf("%d",&T);
    while(T--){
        scanf("%d",&op);
        if(op==1){
            scanf("%d%s",&p,str);
            p-=d;
            ro.insert(p,str);
            l[cnt++]=ro;
        }
        else if(op==2){
            scanf("%d%d",&p,&c);
            p-=d,c-=d;
            ro.erase(p-1,c);
            l[cnt++]=ro;
        }
        else{
            scanf("%d%d%d",&v,&p,&c);
            p-=d,v-=d,c-=d;
            tmp=l[v].substr(p-1,c);
            d+=count(tmp.begin(),tmp.end(),'c');
            cout<<tmp<<endl;
        }
    }
}
#include<bits/extc++.h>
using namespace std;
using namespace __gnu_pbds;
__gnu_pbds::priority_queue<int> h1,h2;
typedef tree<int,null_type,less<int>,rb_tree_tag, tree_order_statistics_node_update> set_t;

int main(){
    printf("heap:\n");
    for(int i=1;i<=10;i+=2)h1.push(i);
    for(int i=2;i<=10;i+=2)h2.push(i);

    printf("%d\n",h1.top());
    printf("%d\n",h2.top());
    h1.join(h2);
    printf("%d\n",h1.size());
    printf("%d\n",h2.size());
    printf("%d\n",h1.top());

    printf("\ntree:\n");
    set_t s;
    for(int i=0;i<5;i++)s.insert(10*i);
    printf("%d\n",*s.find_by_order(0));
    printf("%d\n",*s.find_by_order(3));
    printf("%d\n",s.find_by_order(5)==s.end());

    printf("%d\n",s.order_of_key(0));
    printf("%d\n",s.order_of_key(30));
    printf("%d\n",s.order_of_key(35));
    printf("%d\n",s.order_of_key(100));
    return 0;
}
```

4.2 GNU Bitwise Operation

```
int __builtin_ffs (unsigned int x)
int __builtin_ffsl (unsigned long)
int __builtin_ffsll (unsigned long long)
// 返回右起第一個1的位置
// Returns one plus the index of the least significant 1-bit of x, or if x is zero, returns zero.

int __builtin_clz (unsigned int x)
int __builtin_clzl (unsigned long)
int __builtin_clzll (unsigned long long)
// 返回左起第一個1之前0的個數
// Returns the number of leading 0-bits in x, starting at the most significant bit position. If x is 0, the result is undefined.

int __builtin_ctz (unsigned int x)
int __builtin_ctzl (unsigned long)
int __builtin_ctzll (unsigned long long)
// 返回右起第一個1之後的0的個數
// Returns the number of trailing 0-bits in x, starting at the least significant bit position. If x is 0, the result is undefined.

int __builtin_popcount (unsigned int x)
int __builtin_popcountl (unsigned long)
int __builtin_popcountll (unsigned long long)
// 返回1的個數
// Returns the number of 1-bits in x.

int __builtin_parity (unsigned int x)
int __builtin_parityl (unsigned long)
int __builtin_parityll (unsigned long long)
// 返回1的個數的奇偶性(1的個數 mod 2的值)
// Returns the parity of x, i.e. the number of 1-bits in x modulo 2.
```

5 Graph

5.1 BCC

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 10000;
vector <int> adja[MAXN];
int gcnt, top, timeStamp, dfn[MAXN], low[MAXN], depth[MAXN];
pair<int, int> stk[MAXN],ans[MAXN];
set <int> group[MAXN];
bool cut[MAXN];
void BCC(int now, int nextv){
    int sf, st;
    group[gcnt].clear();
    do{
        sf = stk[top-1].first, st = stk[top-1].second;
        group[gcnt].insert(sf);
        group[gcnt].insert(st);
        --top;
    }while(sf != now || st != nextv);
    ++gcnt;
}
void tarjan(int now, int parent, int d){
    int child = 0;
    dfn[now] = low[now] = ++timeStamp, depth[now] = d;
    for(int i = 0; i < adja[now].size(); i++){
        int nextv = adja[now][i];
        if(nextv == parent) continue;
        if(dfn[nextv] == 0){
            stk[top++] = make_pair(now, nextv);
            tarjan(nextv, now, d+1);
            low[now] = min(low[now], low[nextv]);
            ++child;
            if( (parent != -1 && low[nextv] >= dfn[now]) || (parent == -1 && child >= 2)){
                cut[now] = true;
                if(parent != -1) BCC(now, nextv);
            }
            if(parent == -1) BCC(now, nextv);
        }
        else if(depth[nextv] < depth[now]-1){
            stk[top++] = make_pair(now, nextv);
            low[now] = min(low[now], dfn[nextv]);
        }
    }
}
int main(){
    int n,m,x,y,cnt=0;
    while(~scanf("%d",&n)){
        cnt=timeStamp=top=gcnt=0;
        memset(cut, 0, sizeof(cut));
        memset(dfn, 0, sizeof(dfn));
        for(int i=0;i<n;i++)adja[i].clear();
        for(int i=0;i<n;i++){
            scanf("%d ",&x);
            scanf("(%d)",&m);
            while(m--){
                scanf("%d",&y);
                adja[x].push_back(y);
            }
        }
        for(int i=0;i<n;i++)
            if(dfn[i]==0)tarjan(i, -1, 1);
        for(int i=0;i<gcnt;i++){
            if(group[i].size()==2){
                //critical links
            }
        }
    }
}
```

5.2 MST Directed

```
#include<cstdio>
#include<vector>
#include<algorithm>
#define N 100100
using namespace std;
struct edge{
  edge(){}
  edge(int _f,int _d):f(_f),d(_d){}
  int f;
  int d;
  bool operator<(const edge &rhs)const{return d<rhs.d;}
};
struct node{
  int sz,v,now;
  node *l,*r;
  void pull(){sz=1+(l?l->sz:0)+(r?r->sz:0);}
}pq[N];
int pa[N],sub[N],stk[N],top;
bool vis[N],instk[N];
vector<edge> rg[N];
void init(int n){
  for(int i=0;i<n;i++){
    pa[i]=i;
    sub[i]=0;
    pq[i].l=pq[i].r=NULL;
    pq[i].sz=1;
    pq[i].v=i;
    pq[i].now=0;
  }
}
int find(int x){
  if(pa[x]==x) return x;
  int y=find(pa[x]);
  if(pa[x]!=y) sub[x]+=sub[pa[x]],pa[x]=y;
  return pa[x];
}
inline int get_sub(int x){
  if(x==find(x)) return sub[x];
  else return sub[x]+sub[pa[x]];
}
inline int get_cost(const node& a){
  return rg[a.v][a.now].d-get_sub(a.v);
}
bool cmp(const node& a,const node& b){
  return get_cost(a)<get_cost(b);
}
node* merge(node *a,node *b){
  if(!a||!b) return a?a:b;
  if(cmp(*b,*a)) swap(a,b);
  a->r=merge(a->r,b);
  if((a->l?a->l->sz:0)<(a->r?a->r->sz:0)) swap(a->l,a->r);
  a.pull();
  return a;
}
int min_cost_arborescence(int r,int n){
  vis[r]=true;
  int res=0;
  for(int i=0;i<n;i++){
    if(!vis[i]){
      top=0;
      int u=i;
      while(!vis[u]){

      }
    }
  }
}
int main(){
  int n,m,r,x,y,w;
  scanf("%d%d%d",&n,&m,&r);
  for(int i=0;i<m;i++){
    scanf("%d%d%d",&x,&y,&w);
    rg[y].push_back(edge(x,w));
    sort()
  }
}
```

5.3 SCC

```
#include <cstdlib>
#include <iostream>
#include <vector>
#include <queue>
#define N 300002
using namespace std;
vector<int>go[N],back[N],tree[N];
int hu[N],ST[N],st=0,scc[N],scCo[N],scmx[N];
bool wed[N];
int DFS_go(int now){
    //cout<<now<<" DFS ";
    wed[now]=true;
    for(int i=0;i<go[now].size();i++){
        if(!wed[go[now][i]])
            DFS_go(go[now][i]);
    }
    ST[st++]=now;
    return 0;
}
int DFS_back(int now,int id){
    wed[now]=true;
    scc[now]=id;
    int sum=1;
    if(now==0)sum=0;
    for(int i=0;i<back[now].size();i++){
        if(!wed[back[now][i]])
            sum+=DFS_back(back[now][i],id);
    }
    return sum;
}
int DFS_tree(int now)
{
    if(scmx[now]!=0)return scmx[now];
    int mx=0,tmp;
    for(int i=0;i<tree[now].size();i++){
        tmp=DFS_tree(tree[now][i]);
        mx=(mx>tmp)? mx:tmp;
    }
    scmx[now]=mx+scCo[now];
    return mx+scCo[now];
}
int main(int argc,char *argv[])
{
    ios_base::sync_with_stdio(false);
    int n,k;
    char c;
    cin>>n>>k>>hu[1];
    go[0].push_back(1);
    back[1].push_back(0);
    for(int i=2;i<=n;i++){
        cin>>hu[i];
        if(hu[i]>=hu[i-1]){
            go[i].push_back(i-1);
            back[i-1].push_back(i);
        }
        if(hu[i-1]>=hu[i]){
            go[i-1].push_back(i);
            back[i].push_back(i-1);
        }
        go[0].push_back(i);
        back[i].push_back(0);
    }
    for(int i=1;i<=n;i++){
        cin>>c;
        if(c=='T'){
            go[i].push_back(0);
            back[0].push_back(i);
        }
    }
    for(int i=0;i<=n;i++)
        if(!wed[i])DFS_go(i);
    //cout<<endl;
    fill((bool*)wed,(bool*)wed+N,false);
    int tsc=0;
    // for(int i=0;i<st;i++)cout<<ST[i]<<" HH ";
    // cout<<endl;
    while(st!=0)
        if(!wed[ST[--st]]){
            scCo[tsc]=DFS_back(ST[st],tsc);
            tsc++;
        }
    // for(int i=0;i<N;i++)
    //     while(!back[i].empty())back[i].pop_back();
    for(int i=0;i<=n;i++)
        for(int j=0;j<go[i].size();j++){
            if(scc[i]!=scc[go[i][j]]){
                tree[scc[i]].push_back(scc[go[i][j]]);
            }
        }
    // for(int i=0;i<=n;i++)cout<<scc[i]<<" BB ";
    // cout<<endl;
    // for(int i=0;i<tsc;i++)cout<<scCo[i]<<" GG ";
    cout<<DFS_tree(scc[k])<<endl;
    //system("pause");
    return 0;
}
```

6 Java

6.1 Big Integer

```
import java.math.*;
import java.io.*;
import java.util.*;
public class Main{
    public static void main(String []argv){
        c[0][0]=BigInteger.ONE;
        for(int i=1;i<3001;i++){
            c[i][0]=BigInteger.ONE;
            c[i][i]=BigInteger.ONE;
            for(int j=1;j<i;j++)c[i][j]=c[i-1][j].add(c[i-1][j-1]);
        }
        Scanner scanner = new Scanner(System.in);
        int T = scanner.nextInt();
        BigInteger x;
        BigInteger ans;
        while(T-- > 0){
            ans = BigInteger.ZERO;
            int n = scanner.nextInt();
            for(int i=0;i<n;i++){
                x = new BigInteger(scanner.next());
                if(i%2 == 1)ans=ans.subtract(c[n-1][i].multiply(x));
                else ans=ans.add(c[n-1][i].multiply(x));
            }
            if(n%2 == 0)ans=BigInteger.ZERO.subtract(ans);
            System.out.println(ans);
        }
    }
}
```

6.2 Prime

```
import java.math.*;
import java.io.*;
import java.util.*;
public class Main{
    public static void main(String []argv){
        Scanner scanner = new Scanner(System.in);
        int T = scanner.nextInt();
        for (int cs = 0 ; cs < T ; cs++){
            if (cs != 0) {
                System.out.println("");
            }
            int a = scanner.nextInt();
            int b = scanner.nextInt();
            for (int i = a ; i <= b ; i++) {
                BigInteger x = BigInteger.valueOf(i);
                if (x.isProbablePrime(5) == true) {
                    System.out.println(x);
                }
            }

        }
    }
}
```

7 Matching

7.1 Bipartite Matching

```
#include<bits/stdc++.h>
#define V 20100
#define inf 0x3f3f3f3f
int mx[V],my[V],dis[V],que[V];
bool vis[V];
vector<int> g[V];
bool DFS(int u){
  vis[u]=true;
  for(int i=0;i<g[u].size();i++){
    int v=my[g[u][i]];
    if(v==-1||!vis[v]&&dis[v]==dis[u]+1&&DFS(v)){
      mx[u]=g[u][i];
      my[g[u][i]]=u;
      return true;
    }
  }
  return false;
}
// n is the size of left hand side
int Hopcroft_Karp(int n){
  int matching=0,qt,qf,sp,i,u,v;
  bool flag=true;
  memset(mx,-1,sizeof(mx));
  memset(my,-1,sizeof(my));
  while(flag){
    flag=false;
    qt=qf=0;
    sp=inf;
    for(i=0;i<n;i++){
      if(mx[i]==-1){
        dis[i]=0;
        que[qt++]=i;
      }
      else dis[i]=inf;
    }
    while(qf<qt){
      u=que[qf++];
      if(dis[u]>=sp) continue;
      for(i=0;i<g[u].size();i++){
        v=my[g[u][i]];
        if(v==-1){
          if(dis[u]+1<sp){
            sp=dis[u]+1;
            flag=true;
          }
        }
        else if(dis[u]+1<dis[v]){
          dis[v]=dis[u]+1;
          que[qt++]=v;
        }
      }
    }
    if(flag){
      memset(vis,0,sizeof(vis));
      for(i=0;i<n;i++){
        if(dis[i]==0&&DFS(i)) matching++;
      }
    }
  }
  return matching;
}
```

7.2 Blossom

```
int V;
bool adj[MAXN][MAXN];
int w[MAXN][MAXN];
int p[MAXN];
int m[MAXN];
int d[MAXN];
int c1[MAXN], c2[MAXN];
int q[MAXN], *qf, *qb;
int pp[MAXN];
int f(int x) {return x == pp[x] ? x : (pp[x] = f(pp[x]));}
void u(int x, int y) {pp[x] = y;}
int v[MAXN];
void path(int r, int x){
    if (r == x) return;
    if (d[x] == 0){
        path(r, p[p[x]]);
        int i = p[x], j = p[p[x]];
        m[i] = j; m[j] = i;
    }
    else if (d[x] == 1){
        path(m[x], c1[x]);
        path(r, c2[x]);
        int i = c1[x], j = c2[x];
        m[i] = j; m[j] = i;
    }
}
int lca(int x, int y, int r){
    int i = f(x), j = f(y);
    while (i != j && v[i] != 2 && v[j] != 1){
        v[i] = 1; v[j] = 2;
        if (i != r) i = f(p[i]);
        if (j != r) j = f(p[j]);
    }
    int b = i, z = j; if (v[j] == 1) swap(b, z);
    for (i = b; i != z; i = f(p[i])) v[i] = -1;
    v[z] = -1;
    return b;
}
void contract_one_side(int x, int y, int b){
    for (int i = f(x); i != b; i = f(p[i])){
        u(i, b);
        if (d[i] == 1) c1[i] = x, c2[i] = y, *qb++ = i;
    }
}
bool BFS(int r){
    for (int i=0; i<V; ++i) pp[i] = i;
    memset(v, -1, sizeof(v));
    memset(d, -1, sizeof(d));
    d[r] = 0;
    qf = qb = q;
    *qb++ = r;
    while (qf < qb)
        for (int x=*qf++, y=0; y<V; ++y)
            if (adj[x][y] && m[y] != y && f(x) != f(y))
                if (d[y] == -1)
                    if (m[y] == -1){
                        path(r, x);
                        m[x] = y; m[y] = x;
                        return true;
                    }
                    else{
                        p[y] = x; p[m[y]] = y;
                        d[y] = 1; d[m[y]] = 0;
                        *qb++ = m[y];
                    }
                else
                    if (d[f(y)] == 0) {
                        int b = lca(x, y, r);
                        contract_one_side(x, y, b);
                        contract_one_side(y, x, b);
                    }
    return false;
}
int match_result(){
    int res=0;
    memset(m,-1,sizeof(m));
    for(int i=0;i<V;i++){
        if(m[i]==-1){
            if(BFS(i))res++;
            else m[i]=i;
        }
    }
    return res;
}
int num[10000 + 10],top;
int main(){
    int T,Case=0,n;
    scanf("%d",&T);
    while(T--){
        scanf("%d",&n);
        V=(1<<n);
        top=0;
        for(int i=0;i<V;i++){
            for(int j=i+1;j<V;j++){
                scanf("%d",&w[i][j]);
                num[top++]=w[i][j];
            }
        }
        sort(num,num+top);
        top = (unique(num,num+top)-num);
        int l=0,r=top-1,mid;
        while(r>l){
            mid=(l+r+1)/2;
            memset(adj,false,sizeof(adj));
            for(int i=0;i<V;i++){
                for(int j=i+1;j<V;j++){
                    if(w[i][j]>=num[mid])adj[i][j]=adj[j][i]=true;
                }
            }
            int res=match_result();
            if(res==V/2)l=mid;
            else r=mid-1;
        }
        printf("Case %d: %d\n",++Case,num[l]);
    }
}
```

7.3 Dinic

```
//Dinic
#define V 1000
struct edge{
    edge(){}
    edge(int a,int b,int c):to(a),cap(b),rev(c){}
    int to,cap,rev;
};
vector<edge> g[V];
int level[V];
int iter[V];
void add_edge(int from,int to,int cap){
    g[from].push_back(edge(to,cap,g[to].size()));
    g[to].push_back(edge(from,0,g[from].size()-1));
}
void bfs(int s){
    memset(level,-1,sizeof(level));
    queue<int>que;
    level[s]=0;
    que.push(s);
    while(!que.empty()){
        int v=que.front();
        que.pop();
        for(int q=0;q<g[v].size();q++){
            edge &e=g[v][q];
            if(e.cap>0&&level[e.to]<0){
                level[e.to]=level[v]+1;
                que.push(e.to);
            }
        }
    }
}
int dfs(int v,int t,int f){
    if(v==t)return f;
    for(int &q=iter[v];q<g[v].size();++q){
        edge &e=g[v][q];
        if(e.cap>0&&level[v]<level[e.to]){
            int d=dfs(e.to,t,min(f,e.cap));
            if(d>0){
                e.cap-=d;
                g[e.to][e.rev].cap+=d;
                return d;
            }
        }
    }
    return 0;
}
int max_flow(int s,int t){
    int flow=0;
    for(;;){
        bfs(s);
        if(level[t]<0)return flow;
        memset(iter,0,sizeof(iter));
        int f;
        while((f=dfs(s,t,1e9))>0)
              flow+=f;
    }
}
```

7.4 General Weighted Matching

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <vector>
using namespace std;

typedef long long s64;

const int INF = 2147483647;

const int MaxN = 400;
const int MaxM = 79800;

template <class T>
inline void tension(T &a, const T &b)
{
  if (b < a)
    a = b;
}
template <class T>
inline void relax(T &a, const T &b)
{
  if (b > a)
    a = b;
}
template <class T>
inline int size(const T &a)
{
  return (int)a.size();
}

inline int getint()
{
  char c;
  while (c = getchar(), '0' > c || c > '9');

  int res = c - '0';
  while (c = getchar(), '0' <= c && c <= '9')
    res = res * 10 + c - '0';
  return res;
}

const int MaxNX = MaxN + MaxN;

struct edge
{
  int v, u, w;

  edge(){}
  edge(const int &_v, const int &_u, const int &_w)
    : v(_v), u(_u), w(_w){}
};

int n, m;
edge mat[MaxNX + 1][MaxNX + 1];

int n_matches;
s64 tot_weight;
int mate[MaxNX + 1];
int lab[MaxNX + 1];

int q_n, q[MaxN];
int fa[MaxNX + 1], col[MaxNX + 1];
int slackv[MaxNX + 1];

int n_x;
int bel[MaxNX + 1], blofrom[MaxNX + 1][MaxN + 1];
vector<int> bloch[MaxNX + 1];

inline int e_delta(const edge &e) // does not work inside blossoms
{
  return lab[e.v] + lab[e.u] - mat[e.v][e.u].w * 2;
}
inline void update_slackv(int v, int x)
{
  if (!slackv[x] || e_delta(mat[v][x]) < e_delta(mat[slackv[x]][x]))
    slackv[x] = v;
}
inline void calc_slackv(int x)
{
  slackv[x] = 0;
  for (int v = 1; v <= n; v++)
    if (mat[v][x].w > 0 && bel[v] != x && col[bel[v]] == 0)
      update_slackv(v, x);
}

inline void q_push(int x)
{
  if (x <= n)
    q[q_n++] = x;
  else
  {
    for (int i = 0; i < size(bloch[x]); i++)
      q_push(bloch[x][i]);
  }
}
inline void set_mate(int xv, int xu)
{
  mate[xv] = mat[xv][xu].u;
  if (xv > n)
  {
    edge e = mat[xv][xu];
    int xr = blofrom[xv][e.v];
    int pr = find(bloch[xv].begin(), bloch[xv].end(), xr) - bloch[xv].begin();
    if (pr % 2 == 1)
    {
      reverse(bloch[xv].begin() + 1, bloch[xv].end());
      pr = size(bloch[xv]) - pr;
    }

    for (int i = 0; i < pr; i++)
      set_mate(bloch[xv][i], bloch[xv][i ^ 1]);
    set_mate(xr, xu);

    rotate(bloch[xv].begin(), bloch[xv].begin() + pr, bloch[xv].end());
  }
}
inline void set_bel(int x, int b)
{
  bel[x] = b;
  if (x > n)
  {
    for (int i = 0; i < size(bloch[x]); i++)
      set_bel(bloch[x][i], b);
  }
}

inline void augment(int xv, int xu)
{
  while (true)
  {
    int xnu = bel[mate[xv]];
    set_mate(xv, xu);
    if (!xnu)
      return;
    set_mate(xnu, bel[fa[xnu]]);
    xv = bel[fa[xnu]], xu = xnu;
  }
}
inline int get_lca(int xv, int xu)
{
  static bool book[MaxNX + 1];
  for (int x = 1; x <= n_x; x++)
    book[x] = false;
  while (xv || xu)
  {
    if (xv)
    {
      if (book[xv])
        return xv;
```

```cpp
      book[xv] = true;
      xv = bel[mate[xv]];
      if (xv)
        xv = bel[fa[xv]];
    }
    swap(xv, xu);
  }
  return 0;
}

inline void add_blossom(int xv, int xa, int xu)
{
  int b = n + 1;
  while (b <= n_x && bel[b])
    b++;
  if (b > n_x)
    n_x++;

  lab[b] = 0;
  col[b] = 0;

  mate[b] = mate[xa];

  bloch[b].clear();
  bloch[b].push_back(xa);
  for (int x = xv; x != xa; x = bel[fa[bel[mate[x]]]])
    bloch[b].push_back(x), bloch[b].push_back(bel[mate[x]]), q_push(bel[mate[x]]);
  reverse(bloch[b].begin() + 1, bloch[b].end());
  for (int x = xu; x != xa; x = bel[fa[bel[mate[x]]]])
    bloch[b].push_back(x), bloch[b].push_back(bel[mate[x]]), q_push(bel[mate[x]]);

  set_bel(b, b);

  for (int x = 1; x <= n_x; x++)
  {
    mat[b][x].w = mat[x][b].w = 0;
    blofrom[b][x] = 0;
  }
  for (int i = 0; i < size(bloch[b]); i++)
  {
    int xs = bloch[b][i];
    for (int x = 1; x <= n_x; x++)
      if (mat[b][x].w == 0 || e_delta(mat[xs][x]) < e_delta(mat[b][x]))
        mat[b][x] = mat[xs][x], mat[x][b] = mat[x][xs];
    for (int x = 1; x <= n_x; x++)
      if (blofrom[xs][x])
        blofrom[b][x] = xs;
  }
  calc_slackv(b);
}
inline void expand_blossom1(int b) // lab[b] == 1
{
  for (int i = 0; i < size(bloch[b]); i++)
    set_bel(bloch[b][i], bloch[b][i]);

  int xr = blofrom[b][mat[b][fa[b]].v];
  int pr = find(bloch[b].begin(), bloch[b].end(), xr) - bloch[b].begin();
  if (pr % 2 == 1)
  {
    reverse(bloch[b].begin() + 1, bloch[b].end());
    pr = size(bloch[b]) - pr;
  }

  for (int i = 0; i < pr; i += 2)
  {
    int xs = bloch[b][i], xns = bloch[b][i + 1];
    fa[xs] = mat[xns][xs].v;
    col[xs] = 1, col[xns] = 0;
    slackv[xs] = 0, calc_slackv(xns);
    q_push(xns);
  }
  col[xr] = 1;
  fa[xr] = fa[b];
  for (int i = pr + 1; i < size(bloch[b]); i++)
  {
    int xs = bloch[b][i];
    col[xs] = -1;
    calc_slackv(xs);
  }

  bel[b] = 0;
}
inline void expand_blossom_final(int b) // at the final stage
{
  for (int i = 0; i < size(bloch[b]); i++)
  {
    if (bloch[b][i] > n && lab[bloch[b][i]] == 0)
      expand_blossom_final(bloch[b][i]);
    else
      set_bel(bloch[b][i], bloch[b][i]);
  }
  bel[b] = 0;
}

inline bool on_found_edge(const edge &e)
{
  int xv = bel[e.v], xu = bel[e.u];
  if (col[xu] == -1)
  {
    int nv = bel[mate[xu]];
    fa[xu] = e.v;
    col[xu] = 1, col[nv] = 0;
    slackv[xu] = slackv[nv] = 0;
    q_push(nv);
  }
  else if (col[xu] == 0)
  {
    int xa = get_lca(xv, xu);
    if (!xa)
    {
      augment(xv, xu), augment(xu, xv);
      for (int b = n + 1; b <= n_x; b++)
        if (bel[b] == b && lab[b] == 0)
          expand_blossom_final(b);
      return true;
    }
    else
      add_blossom(xv, xa, xu);
  }
  return false;
}

bool match()
{
  for (int x = 1; x <= n_x; x++)
    col[x] = -1, slackv[x] = 0;

  q_n = 0;
  for (int x = 1; x <= n_x; x++)
    if (bel[x] == x && !mate[x])
      fa[x] = 0, col[x] = 0, slackv[x] = 0, q_push(x);
  if (q_n == 0)
    return false;

  while (true)
  {
    for (int i = 0; i < q_n; i++)
    {
      int v = q[i];
      for (int u = 1; u <= n; u++)
        if (mat[v][u].w > 0 && bel[v] != bel[u])
        {
          int d = e_delta(mat[v][u]);
          if (d == 0)
          {
            if (on_found_edge(mat[v][u]))
              return true;
          }
          else if (col[bel[u]] == -1 || col[bel[u]] == 0)
            update_slackv(v, bel[u]);
        }
    }

    int d = INF;
    for (int v = 1; v <= n; v++)
      if (col[bel[v]] == 0)
        tension(d, lab[v]);
    for (int b = n + 1; b <= n_x; b++)
      if (bel[b] == b && col[b] == 1)
        tension(d, lab[b] / 2);
    for (int x = 1; x <= n_x; x++)
      if (bel[x] == x && slackv[x])
      {
        if (col[x] == -1)
          tension(d, e_delta(mat[slackv[x]][x]));
        else if (col[x] == 0)
          tension(d, e_delta(mat[slackv[x]][x]) / 2);
      }

    for (int v = 1; v <= n; v++)
    {
      if (col[bel[v]] == 0)
        lab[v] -= d;
      else if (col[bel[v]] == 1)
        lab[v] += d;
    }
    for (int b = n + 1; b <= n_x; b++)
      if (bel[b] == b)
      {
        if (col[bel[b]] == 0)
          lab[b] += d * 2;
        else if (col[bel[b]] == 1)
          lab[b] -= d * 2;
      }

    q_n = 0;
    for (int v = 1; v <= n; v++)
      if (lab[v] == 0) // all unmatched vertices' labels are zero! cheers!
        return false;
    for (int x = 1; x <= n_x; x++)
      if (bel[x] == x && slackv[x] && bel[slackv[x]] != x && e_delta(mat[
    slackv[x]][x]) == 0)
      {
        if (on_found_edge(mat[slackv[x]][x]))
          return true;
      }
    for (int b = n + 1; b <= n_x; b++)
      if (bel[b] == b && col[b] == 1 && lab[b] == 0)
        expand_blossom1(b);
  }
  return false;
}

void calc_max_weight_match()
{
  for (int v = 1; v <= n; v++)
    mate[v] = 0;

  n_x = n;
  n_matches = 0;
  tot_weight = 0;

  bel[0] = 0;
  for (int v = 1; v <= n; v++)
    bel[v] = v, bloch[v].clear();
  for (int v = 1; v <= n; v++)
    for (int u = 1; u <= n; u++)
      blofrom[v][u] = v == u ? v : 0;

  int w_max = 0;
  for (int v = 1; v <= n; v++)
    for (int u = 1; u <= n; u++)
```

```
      relax(w_max, mat[v][u].w);
  for (int v = 1; v <= n; v++)
    lab[v] = w_max;

  while (match())
    n_matches++;

  for (int v = 1; v <= n; v++)
    if (mate[v] && mate[v] < v)
      tot_weight += mat[v][mate[v]].w;
}

int main()
{
  n = getint(), m = getint();

  for (int v = 1; v <= n; v++)
    for (int u = 1; u <= n; u++)
      mat[v][u] = edge(v, u, 0);

  for (int i = 0; i < m; i++)
  {
    int v = getint(), u = getint(), w = getint();
    mat[v][u].w = mat[u][v].w = w;
  }

  calc_max_weight_match();

  printf("%lld\n", tot_weight);
  for (int v = 1; v <= n; v++)
    printf("%d ", mate[v]);
  printf("\n");

  return 0;
}
```

7.5 KM

```
#define MAXN 100
#define INF INT_MAX
int g[MAXN][MAXN],lx[MAXN],ly[MAXN],slack_y[MAXN];
int px[MAXN],py[MAXN],match_y[MAXN],par[MAXN];
int n;
void adjust(int y){//把增廣路上所有邊反轉
  match_y[y]=py[y];
  if(px[match_y[y]]!=-2)
    adjust(px[match_y[y]]);
}
bool dfs(int x){//DFS找增廣路
  for(int y=0;y<n;++y){
    if(py[y]!=-1)continue;
    int t=lx[x]+ly[y]-g[x][y];
    if(t==0){
      py[y]=x;
      if(match_y[y]==-1){
        adjust(y);
        return 1;
      }
      if(px[match_y[y]]!=-1)continue;
      px[match_y[y]]=y;
      if(dfs(match_y[y]))return 1;
    }else if(slack_y[y]>t){
      slack_y[y]=t;
      par[y]=x;
    }
  }
  return 0;
}
inline int km(){
  memset(ly,0,sizeof(int)*n);
  memset(match_y,-1,sizeof(int)*n);
  for(int x=0;x<n;++x){
    lx[x]=-INF;
    for(int y=0;y<n;++y){
      lx[x]=max(lx[x],g[x][y]);
    }
  }
  for(int x=0;x<n;++x){
    for(int y=0;y<n;++y)slack_y[y]=INF;
    memset(px,-1,sizeof(int)*n);
    memset(py,-1,sizeof(int)*n);
    px[x]=-2;
    if(dfs(x))continue;
    bool flag=1;
    while(flag){
      int cut=INF;
      for(int y=0;y<n;++y)
        if(py[y]==-1&&cut>slack_y[y])cut=slack_y[y];
      for(int j=0;j<n;++j){
        if(px[j]!=-1)lx[j]-=cut;
        if(py[j]!=-1)ly[j]+=cut;
        else slack_y[j]-=cut;
      }
      for(int y=0;y<n;++y){
        if(py[y]==-1&&slack_y[y]==0){
          py[y]=par[y];
          if(match_y[y]==-1){
            adjust(y);
            flag=0;
            break;
          }
          px[match_y[y]]=y;
          if(dfs(match_y[y])){
            flag=0;
            break;
          }
        }
      }
    }
  }
  int ans=0;
  for(int y=0;y<n;++y)if(g[match_y[y]][y]!=-INF)ans+=g[match_y[y]][y];
  return ans;
}
```

7.6 Min Cost Flow

```
#define maxnode (1000+10)
#define maxedge (40000+10)
#define INF 1023456789
#include<bits/stdc++.h>
using namespace std;
int node, src, dest, nedge;
int head[maxnode], point[maxedge], nxt[maxedge], flow[maxedge], capa[maxedge], wt[maxedge];
int dist[maxnode], in[maxnode], from[maxnode], mf[maxnode];
//set number of node, source, and destination (one base)
void init(int _node, int _src, int _dest) {
    node = _node;
    src = _src;
    dest = _dest;
    nedge = 0;
    memset(point, -1, sizeof(point));
    for (int i = 1; i <= node; i++) head[i] = -1;
    nedge = 0;
}
void add_edge(int u, int v, int c1, int w) {
    point[nedge] = v, capa[nedge] = c1, flow[nedge] = 0, nxt[nedge] = head[u], wt[nedge]=w, head[u] = (nedge++);
    point[nedge] = u, capa[nedge] = 0, flow[nedge] = 0, nxt[nedge] = head[v], wt[nedge]=-w, head[v] = (nedge++);
}
int sp(int &left){
  for(int i=1;i<=node;i++) dist[i]=INF;
  queue<int> que;
  que.push(src);
  in[src]=1;
  mf[src]=left;
  dist[src]=0;
  while(!que.empty()){
    int u=que.front();
    que.pop();
    in[u]=0;
    if(dist[u]>=dist[dest]) continue;
    for(int v=head[u];v!=-1;v=nxt[v]){
      if(flow[v]==capa[v]) continue;
      if(dist[u]+wt[v]<dist[point[v]]){
        dist[point[v]]=dist[u]+wt[v];
        from[point[v]]=v;
        mf[point[v]]=min(mf[u],capa[v]-flow[v]);
        if(!in[point[v]]){
          in[point[v]]=1;
          que.push(point[v]);
        }
      }
    }
  }
  left-=mf[dest];
  if(dist[dest]<INF){
    for(int u=dest;u!=src;u=point[from[u]^1]){
      flow[from[u]]+=mf[dest];
      flow[from[u]^1]-=mf[dest];
    }
  }
  return dist[dest];
}
int min_cost_flow(){
  int res=0,tmp,maxflow=2;
  while(maxflow&&(tmp=sp(maxflow))<INF) res+=tmp;
  return res;
}
int main(){
  int n,m,x,y,z;
  while(scanf("%d%d",&n,&m)==2){
    init(n,1,n);
    for(int i=0;i<m;i++){
      scanf("%d%d%d",&x,&y,&z);
      add_edge(x,y,1,z);
      add_edge(y,x,1,z); //undirected
    }
    printf("%d\n",min_cost_flow());
  }
  return 0;
}
```

7.7 Stable Marriage

```
#define F(n) Fi(i, n)
#define Fi(i, n) Fl(i, 0, n)
#define Fl(i, l, n) for(int i = l ; i < n ; ++i)
#include <bits/stdc++.h>
using namespace std;
int D, quota[205], weight[205][5];
int S, scoretodep[12005][205], score[5];
int P, prefer[12005][85], iter[12005];
int ans[12005];
typedef pair<int, int> PII;
map<int, int> samescore[205];
typedef priority_queue<PII, vector<PII>, greater<PII>> QQQ;
QQQ pri[205];
void check(int d) {
  PII t = pri[d].top();
  int v;
  if (pri[d].size() - samescore[d][t.first] + 1 <= quota[d]) return;
  while (pri[d].top().first == t.first) {
    v = pri[d].top().second;
    ans[v] = -1;
    --samescore[d][t.first];
    pri[d].pop();
  }
}
void push(int s, int d) {
  if (pri[d].size() < quota[d]) {
    pri[d].push(PII(scoretodep[s][d], s));
    ans[s] = d;
    ++samescore[s][scoretodep[s][d]];
  } else if (scoretodep[s][d] >= pri[d].top().first) {
    pri[d].push(PII(scoretodep[s][d], s));
    ans[s] = d;
    ++samescore[s][scoretodep[s][d]];
    check(d);
  }
}
void f() {
  int over;
  while (true) {
    over = 1;
    Fi (q, S) {
      if (ans[q] != -1 || iter[q] >= P) continue;
      push(q, prefer[q][iter[q]++]);
      over = 0;
    }
    if (over) break;
  }
}
main() {
  ios::sync_with_stdio(false);
  cin.tie(NULL);
  int sadmit, stof, dexceed, dfew;
  while (cin >> D, D) { // Beware of the input format or judge may troll us.
    sadmit = stof = dexceed = dfew = 0;
    memset(iter, 0, sizeof(iter));
    memset(ans, 0, sizeof(ans));
    Fi (q, 205) {
      pri[q] = QQQ();
      samescore[q].clear();
    }
    cin >> S >> P;
    Fi (q, D) {
      cin >> quota[q];
      Fi (w, 5) cin >> weight[q][w];
    }
    Fi (q, S) {
      Fi (w, 5) cin >> score[w];
      Fi (w, D) {
        scoretodep[q][w] = 0;
        F (5) scoretodep[q][w] += weight[w][i] * score[i];
      }
    }
    Fi (q, S) Fi (w, P) {
      cin >> prefer[q][w];
      --prefer[q][w];
    }
    f();
    Fi (q, D) sadmit += pri[q].size();
    Fi (q, S) if (ans[q] == prefer[q][0]) ++stof;
    Fi (q, D) if (pri[q].size() > quota[q]) ++dexceed;
    Fi (q, D) if (pri[q].size() < quota[q]) ++dfew;
    cout << sadmit << ' ' << stof << ' ' << dexceed << ' ' << dfew << '\n';
  }
}
```

8 Mathematics

8.1 Extgcd

```
long long extgcd(long long a,long long b,long long &x,long long &y){
    long long d=a;
    if(b!=0){
        d=extgcd(b,a%b,y,x);
        y-=(a/b)*x;
    }
    else x=1,y=0;
    return d;
}
int main(){
    int T;
    long long a,b,m,GCD,x,y;
    while(~scanf("%d",&T))
        while(T--){
            scanf("%lld%lld%lld",&m,&a,&b);
            GCD=extgcd(a,m,x,y);
            if(GCD!=1)printf("No inverse, gcd(a,m)=%lld\n",GCD);
            else{
                b=((-b*x)%m+m)%m;
                printf("%lld %lld\n",(x%m+m)%m,b);
            }
        }
}
```

8.2 Miller-Rabin

```
inline long long mod_mul(long long a,long long b,long long m){
  a%=m,b%=m;
  long long y=(long long)((double)a*b/m+0.5);/* fast for m < 2^58 */
  long long r=(a*b-y*m)%m;
  return r<0?r+m:r;
}
template<typename T>
inline T pow(T a,T b,T mod){//a^b%mod
  T ans=1;
  for(;b;a=mod_mul(a,a,mod),b>>=1)
    if(b&1)ans=mod_mul(ans,a,mod);
  return ans;
}
int sprp[3]={2,7,61};//int範圍可解
int llsprp[7]={2,325,9375,28178,450775,9780504,1795265022};//至少unsigned long long範圍
template<typename T>
inline bool isprime(T n,int *sprp,int num){
  if(n==2)return 1;
  if(n<2||n%2==0)return 0;
  int t=0;
  T u=n-1;
  for(;u%2==0;++t)u>>=1;
  for(int i=0;i<num;++i){
    T a=sprp[i]%n;
    if(a==0||a==1||a==n-1)continue;
    T x=pow(a,u,n);
    if(x==1||x==n-1)continue;
    for(int j=0;j<t;++j){
      x=mod_mul(x,x,n);
      if(x==1)return 0;
      if(x==n-1)break;
    }
    if(x==n-1)continue;
    return 0;
  }
  return 1;
}
```

9 String

9.1 AC Automaton

```
#ifndef SUNMOON_AHO_CORASICK_AUTOMATON
#define SUNMOON_AHO_CORASICK_AUTOMATON
#include<queue>
#include<vector>
template<char L='a',char R='z'>
class ac_automaton{
  private:
    struct joe{
      int next[R-L+1],fail,efl,ed,cnt_dp,vis;
      joe():ed(0),cnt_dp(0),vis(0){
        for(int i=0;i<=R-L;++i)next[i]=0;
      }
    };
  public:
    std::vector<joe> S;
    std::vector<int> q;
    int qs,qe,vt;
    ac_automaton():S(1),qs(0),qe(0),vt(0){}
    inline void clear(){
      q.clear();
      S.resize(1);
      for(int i=0;i<=R-L;++i)S[0].next[i]=0;
      S[0].cnt_dp=S[0].vis=qs=qe=vt=0;
    }
    inline void insert(const char *s){
      int o=0;
      for(int i=0,id;s[i];++i){
        id=s[i]-L;
        if(!S[o].next[id]){
          S.push_back(joe());
          S[o].next[id]=S.size()-1;
        }
        o=S[o].next[id];
      }
      ++S[o].ed;
    }
    inline void build_fail(){
      S[0].fail=S[0].efl=-1;
      q.clear();
      q.push_back(0);
      ++qe;
      while(qs!=qe){
        int pa=q[qs++],id,t;
        for(int i=0;i<=R-L;++i){
          t=S[pa].next[i];
          if(!t)continue;
          id=S[pa].fail;
          while(~id&&!S[id].next[i])id=S[id].fail;
          S[t].fail=~id?S[id].next[i]:0;
          S[t].efl=S[S[t].fail].ed?S[t].fail:S[S[t].fail].efl;
          q.push_back(t);
          ++qe;
        }
      }
    }
    /*DP出每個前綴在字串s出現的次數並傳回所有字串被s匹配成功的次數O(N+M)*/
    inline int match_0(const char *s){
      int ans=0,id,p=0,i;
      for(i=0;s[i];++i){
        id=s[i]-L;
        while(!S[p].next[id]&&p)p=S[p].fail;
        if(!S[p].next[id])continue;
        p=S[p].next[id];
        ++S[p].cnt_dp;/*匹配成功則它所有後綴都可以被匹配(DP計算)*/
      }
      for(i=qe-1;i>=0;--i){
        ans+=S[q[i]].cnt_dp*S[q[i]].ed;
        if(~S[q[i]].fail)S[S[q[i]].fail].cnt_dp+=S[q[i]].cnt_dp;
      }
      return ans;
    }
    /*多串匹配走efl邊並傳回所有字串被s匹配成功的次數O(N*M^1.5)*/
    inline int match_1(const char *s)const{
      int ans=0,id,p=0,t;
      for(int i=0;s[i];++i){
        id=s[i]-L;
        while(!S[p].next[id]&&p)p=S[p].fail;
        if(!S[p].next[id])continue;
        p=S[p].next[id];
        if(S[p].ed)ans+=S[p].ed;
        for(t=S[p].efl;~t;t=S[t].efl){
          ans+=S[t].ed;/*因為都走efl邊所以保證匹配成功*/
        }
      }
      return ans;
    }
    /*枚舉(s的子字串∩A)的所有相異字串各恰一次並傳回次數O(N*M^(1/3))*/
    inline int match_2(const char *s){
      int ans=0,id,p=0,t;
      ++vt;
      /*把戳記vt+=1，只要vt沒溢位，所有S[p].vis==vt就會變成false
      這種利用vt的方法可以O(1)歸零vis陣列*/
      for(int i=0;s[i];++i){
        id=s[i]-L;
        while(!S[p].next[id]&&p)p=S[p].fail;
        if(!S[p].next[id])continue;
        p=S[p].next[id];
        if(S[p].ed&&S[p].vis!=vt){
          S[p].vis=vt;
          ans+=S[p].ed;
        }
        for(t=S[p].efl;~t&&S[t].vis!=vt;t=S[t].efl){
          S[t].vis=vt;
          ans+=S[t].ed;/*因為都走efl邊所以保證匹配成功*/
        }
      }
      return ans;
    }
    /*把AC自動機變成真的自動機*/
    inline void evolution(){
      for(qs=1;qs!=qe;){
        int p=q[qs++];
        for(int i=0;i<=R-L;++i)
          if(S[p].next[i]==0)S[p].next[i]=S[S[p].fail].next[i];
      }
    }
};
#endif
```

9.2 BWT

```
// BWT
const int N = 8;            // 字串長度
int s[N+N+1] = "suffixes";  // 字串，後面預留一倍空間。
int sa[N];                  // 後綴陣列
int pivot;

int cmp(const void* i, const void* j)
{
    return strncmp(s+*(int*)i, s+*(int*)j, N);
}

// 此處便宜行事，採用 O(N²logN) 的後綴陣列演算法。
void BWT()
{
    strncpy(s + N, s, N);
    for (int i=0; i<N; ++i) sa[i] = i;
    qsort(sa, N, sizeof(int), cmp);
    // 當輸入字串的所有字元都相同，必須當作特例處理。
    // 或者改用stable sort。

    for (int i=0; i<N; ++i)
        cout << s[(sa[i] + N-1) % N];

    for (int i=0; i<N; ++i)
        if (sa[i] == 0)
        {
            pivot = i;
            break;
        }
}

// Inverse BWT
const int N = 8;            // 字串長度
char t[N+1] = "xuffessi";   // 字串
int pivot;
int next[N];

void IBWT()
{
    vector<int> index[256];
    for (int i=0; i<N; ++i)
        index[t[i]].push_back(i);

    for (int i=0, n=0; i<256; ++i)
        for (int j=0; j<index[i].size(); ++j)
            next[n++] = index[i][j];

    int p = pivot;
    for (int i=0; i<N; ++i)
        cout << t[p = next[p]];
}
```

9.3 Suffix Array

```
//should initialize s and n first
#define N 301000
using namespace std;
char s[N]; //string=s,suffix array=sar,longest common prefix=lcp
int rk[2][N],id[2][N];
int n,p;
int cnt[N];
int len[N],od[N],sar[N];
inline int sr(int i,int t){ //rank of shifted position
```

```
  return i+t<n?rk[p][i+t]:-1;
}
inline bool check_same(int i,int j,int t){
  return rk[p][i]==rk[p][j]&&sr(i,t)==sr(j,t);
}
bool cmp(int i,int j){
  return s[i]<s[j];
}
void sa(){ //length of array s
  int i,t,now,pre;
  memset(cnt,0,sizeof(cnt));
  for(i=0;i<n;i++){
    id[p][i]=i;
    rk[p][i]=s[i];
    cnt[s[i]]++;
  }
  for(i=1;i<128;i++) cnt[i]+=cnt[i-1];
  sort(id[p],id[p]+n,cmp);
  for(t=1;t<n;t<<=1){
        //least significant bit is already sorted
    for(i=n-1;i>=0;i--){
            now=id[p][i]-t;
      if(now>=0) id[p^1][--cnt[rk[p][now]]]=now;
    }
    for(i=n-t;i<n;i++){
            id[p^1][--cnt[rk[p][i]]]=i;
    }
    memset(cnt,0,sizeof(cnt));
    now=id[p^1][0];
    rk[p^1][now]=0;
    cnt[0]++;
    for(i=1;i<n;i++){
      pre=now;
      now=id[p^1][i];
      if(check_same(pre,now,t)){
        rk[p^1][now]=rk[p^1][pre];
      }
      else{
        rk[p^1][now]=rk[p^1][pre]+1;
      }
      cnt[rk[p^1][now]]++;
    }
    p^=1;
    if(rk[p][now]==n-1) break;
    for(i=1;i<n;i++) cnt[i]+=cnt[i-1];
  }
  memcpy(sar,id[p],sizeof(sar));
}
void lcp(){
    int i,l,pre;
    for(i=0;i<n;i++) od[sar[i]]=i;
    for(i=0;i<n;i++){
        if(i) l=len[od[i-1]]?len[od[i-1]]-1:0;
        else l=0;
        if(od[i]){
            pre=sar[od[i]-1];
            while(pre+l<n&&i+l<n&&s[pre+l]==s[i+l]) l++;
            len[od[i]]=l;
        }
        else len[0]=0;
    }
}
```

9.4 Suffix Automaton

```
#include<bits/stdc++.h>
#define C 96
#define N 200100
using namespace std;
struct SAM{
  struct node{
    node *nxt[C],*pre;
    int len;
    vector<int> pos;
  };
  node mem[N*2],*root,*ed;
  int top;
  SAM(){
    top = 0;
    root = new_node(0);
    ed = root;
  }
  node *new_node(int l){
    for(int i=0;i<C;i++) mem[top].nxt[i]=NULL;
    mem[top].pre=NULL;
    mem[top].len=l;
    mem[top].pos.clear();
    return mem+(top++);
  }
  node *split_node(int l,node *p){
    for(int i=0;i<C;i++) mem[top].nxt[i]=p->nxt[i];
    mem[top].pre = p->pre;
    mem[top].len = l;
    mem[top].pos.assign()
    p->pre = mem+top;
    return mem+(top++);
  }
  void push(char c){
    node *nw = new_node(ed->len+1),*ptr=ed->pre;
    ed->nxt[c] = nw;
    nw->pos.push_back(ed->len);
    for(;ptr;ptr=ptr->pre){
      if(ptr->nxt[c]){
        if(ptr->nxt[c]->len==ptr->len+1){
          nw->pre = ptr->nxt[c];
        }
        else{
          node *tmp=ptr->nxt[c];
          nw->pre = split_node(ptr->len+1,tmp);
          while(ptr && ptr->nxt[c]==tmp){
            ptr->nxt[c] = nw->pre;
            ptr = ptr->pre;
          }
        }
        break;
      }
      else{
        ptr->nxt[c] = nw;
      }
    }
    if(!nw->pre) nw->pre = root;
    ed = ed->nxt[c];
  }
  void init(){
    while(top){
      mem[--top].pos.clear();
    }
    root = new_node(0);
    ed = root;
  }
  void push(char *s){
    for(int i=0;s[i];i++) push(s[i]-32);
  }
  long long count(){
    long long ans=0;
    for(int i=1;i<top;i++){
      ans+=mem[i].len-mem[i].pre->len;
    }
    return ans;
  }
}sam;
char S[N];
int main(){
  int T;
  scanf("%d",&T);
  while(T--){
    scanf("%s",S);
    sam.build(S);
    printf("%lld\n",sam.count());
  }
  return 0;
}
```

9.5 Z Algorithm

```
void Zalg(char *s, int *z, int n) {
  z[0]=n;
  for(int L=0, R=0, i=1; i<n; i++) {
    if(i<=R && z[i-L]<=R-i) z[i]=z[i-L];
    else {
      L=i;
      if(i>R) R=i;
      while(R<n && s[R-L]==s[R]) R++;
      z[i]=(R--)-L;
    }
  }
}
```

10 Struct

10.1 Splay Tree

```
#include<cstdio>
#include<string>
using namespace std;
struct node{
  node *ch[2],*par;
  long long sum;
  int val,sz,add;
  node(){}
  node(int x):par(NULL),val(x),sum(x),add(0),sz(1){ch[0]=ch[1]=NULL;}
  bool dir(){return !par||par->ch[1]==this;}
  void pull();
  void push();
}pool[100100];
inline long long qsum(node *x){
  return x?1LL*x->add*x->sz+x->sum:0;
}
inline int qsz(node *x){return x?x->sz:0;}
void node::pull(){
  sum=val+qsum(ch[0])+qsum(ch[1]);
  sz=1+qsz(ch[0])+qsz(ch[1]);
}
void node::push(){
  if(add){
    val+=add;
    sum+=add*sz;
    if(ch[0]) ch[0]->add+=add;
    if(ch[1]) ch[1]->add+=add;
    add=0;
  }
}
inline void con(node *p,node *c,bool d){
  p->ch[d]=c;
  if(c) c->par=p;
}
void splay(node *x){
  x->push();
  while(x->par){
    node *p=x->par,*g=p->par;
    bool d=x->dir(),pd=p->dir();;
    con(p,x->ch[d^1],d);
    con(x,p,d^1);
    if(g){
      if(g->par) con(g->par,x,g->dir());
      else x->par=NULL;
      if(d^pd){
        con(g,x->ch[d],pd);
        con(x,g,pd^1);
      }
      else{
        con(g,p->ch[pd^1],pd);
        con(p,g,pd^1);
      }
      g->pull();
    }
    else x->par=NULL;
    p->pull();
    x->pull();
  }
}
void check_tree(node *t,int d){
  if(!t) return;
  check_tree(t->ch[0],d+1);
  for(int i=0;i<d;i++) printf("\t");
  printf("%d\n",t->val);
  check_tree(t->ch[1],d+1);
}
void split(node *t,int k,node *&a,node *&b){
  if(!k){
    a=NULL; b=t; return;
  }
  int rod;
  while( k != (rod=qsz(t->ch[0])+1) ){
    t->push();
    if(k>rod) k-=rod,t=t->ch[1];
    else t=t->ch[0];
  }
  splay(t);
  a=t;
  a->push();
  b=a->ch[1];
  a->ch[1]=NULL;
  a->pull();
  if(b) b->par=NULL;
}
node* merge(node *a,node *b){
  if(!a) return b;
  while(a->ch[1]){
    a->push();
    a=a->ch[1];
  }
  splay(a);
  con(a,b,1);
  a->pull();
  return a;
}
int main(){
  int n,q,x;
  node *root=NULL,*a,*b,*c;
  scanf("%d%d",&n,&q);
  for(int i=0;i<n;i++){
    scanf("%d",&x);
    node *tmp=new (pool+i) node(x);
    root=merge(root,tmp);
  }
  for(int i=0;i<q;i++){
    char tp;
    int x,y,z;
    scanf(" %c%d%d",&tp,&x,&y);
    split(root,x-1,a,b);
    split(b,y-x+1,b,c);
    if(tp=='C'){
      scanf("%d",&z);
      b->add+=z;
    }
    else printf("%lld\n",qsum(b));
    root=merge(a,merge(b,c));
  }
  return 0;
}
```

10.2 Treap

```
struct Treap{
  Treap *l,*r;
  int pri,sz,val,add;
  Treap(int _val):pri(rand()),sz(1),val(_val),add(0),l(NULL),r(NULL){}
};

int size(Treap *t){
  return t?t->sz:0;
}
void pull(Treap *t){
  t->sz=size(t->l)+size(t->r)+1;
}
void push(Treap *t){
  t->val+=t->add;
  if(t->l) t->l->add+=t->add;
  if(t->r) t->r->add+=t->add;
  t->add=0;
}
Treap* merge(Treap *a,Treap *b){
  if(!a||!b) return a?a:b;
  if(a->pri > b->pri){
    push(a);
    a->r = merge(a->r,b);
    pull(a);
    return a;
  }
  else{
    push(b);
    b->l = merge(a,b->l);
    pull(b);
    return b;
  }
}
void split(Treap *t,int k,Treap *&a,Treap *&b){
  if(!t) a=b=NULL;
  else{
    push(t);
    if(size(t->l) < k){
      a=t;
      split(t->r,k-size(t->l)-1,a->r,b);
      pull(a);
    }
    else{
      b=t;
      split(t->l,k,a,b->l);
      pull(b);
    }
  }
}
```

11 Tree

11.1 Heavy Light Decomposition

```
//with set value && query sum, 1-based with n points
//remove vis in DFS, add it back if something weird happen(I don’t think it’s required)
using namespace std;
int sz[N],top[N],up[N],dep[N];
int lightval[N];   //value on light edge
struct node{
  node(){}
  node(int _l,int _r):val(1),l(_l),r(_r),lc(NULL),rc(NULL){}
  int l,r;
  node *lc,*rc;
  int sum;
  int val;
  int qsum(){return val>=0?val*(r-l):sum;}
  void push(){
    if(val>=0){
      sum=val*(r-l);
      lc->val=rc->val=val;
      val=-1;
    }
  }
  void pull(){
    sum=lc->qsum()+rc->qsum();
  }
};
node* tr[N];
node* build(int l,int r){
  node *now=new node(l,r);
  if(r-l>1){
    now->lc=build(l,(l+r)/2);
    now->rc=build((l+r)/2,r);
  }
  return now;
}
//partial
int qry(node* now,int l,int r){
  if(l>=r) return 0;
  if(l==now->l&&r==now->r){
    return now->qsum();
  }
  int m=(now->l+now->r)/2;
  now->push();
  if(l>=m){
    return qry(now->rc,l,r);
  }
  else if(r<=m){
    return qry(now->lc,l,r);
  }
  else return qry(now->lc,l,m)+qry(now->rc,m,r);
}
void set0(node *now,int l,int r){
  if(l>=r) return;
  if(l==now->l&&r==now->r){
    now->val=0;
    return;
  }
  int m=(now->l+now->r)/2;
  now->push();
  if(l>=m){
    set0(now->rc,l,r);
  }
  else if(r<=m){
    set0(now->lc,l,r);
  }
  else{
    set0(now->lc,l,m);
    set0(now->rc,m,r);
  }
  now->pull();
}
vector<int> g[N];
void DFS(int u,int p,int d){
  dep[u]=d;
  sz[u]=1;
  for(int i=0;i<g[u].size();i++){
    int v=g[u][i];
    if(v==p) continue;
    DFS(v,u,d+1);
    sz[u]+=sz[v];
}
}
void decom(int u,int p,bool istop){
  bool ed=true;
  if(istop) top[u]=u,up[u]=p,lightval[u]=1;
  else top[u]=top[p],up[u]=up[p];
  for(int i=0;i<g[u].size();i++){
    int v=g[u][i];
    if(v==p) continue;
    if(sz[v]>=sz[u]-sz[v]){
      decom(v,u,false);
      ed=false;
    }
    else decom(v,u,true);
  }
  if(ed){
    tr[top[u]]=build(dep[top[u]],dep[u]);
  }
}
//global
int qry(int u,int v){
  int res=0;
  while(top[u]!=top[v]){
    if(dep[top[u]]>dep[top[v]]) swap(u,v);
    res+=qry(tr[top[v]],dep[top[v]],dep[v]);
    res+=lightval[top[v]];
    v=up[top[v]];
  }
  if(dep[u]>dep[v]) swap(u,v);
  res+=qry(tr[top[v]],dep[u],dep[v]);
  return res;
}
void set0(int u,int v){
  while(top[u]!=top[v]){
    if(dep[top[u]]>dep[top[v]]) swap(u,v);
    set0(tr[top[v]],dep[top[v]],dep[v]);
    lightval[top[v]]=0;
    v=up[top[v]];
  }
  if(dep[u]>dep[v]) swap(u,v);
  set0(tr[top[v]],dep[u],dep[v]);
}
int main(){
  DFS(1,0,0);
  decom(1,0,true);
}
```

11.2 KDtree Insert

```
#include<algorithm>
#include<cmath>
#include<cstdio>
#include<queue>
#include<cstdlib>
#include<vector>
#define MAXN 50100
using namespace std;
inline long long sq(long long x){return x*x;}
const double alpha=0.75;
int W,H,rx[MAXN],ry[MAXN];
namespace KDTree{
struct Point {
  int x,y;
  int index;
  long long distance(const Point &b)const{
    return sq(x-b.x) + sq(y-b.y);
  }
  bool operator==(const Point& rhs){return index==rhs.index;}
};
struct qnode{
  Point p;
  long long dis;
  qnode(){}
  qnode(Point _p,long long _dis){
    p = _p;
    dis = _dis;
  }
  bool operator <(const qnode &b)const{
    if(dis != b.dis)return dis < b.dis;
    else return p.index < b.p.index;
  }
};
priority_queue<qnode>q;
inline bool cmpX(const Point &a,const Point &b){
  return a.x < b.x || (a.x == b.x && a.y < b.y) || (a.x == b.x && a.y == b.y && a.index < b.index);
}
inline bool cmpY(const Point &a,const Point &b){
  return a.y < b.y || (a.y == b.y && a.x < b.x) || (a.y == b.y && a.x == b.x && a.index < b.index);
}
bool cmp(const Point &a,const Point &b,bool div){
  return div?cmpY(a,b):cmpX(a,b);
}
struct Node{
  Point e;
  Node *lc,*rc;
  int size;
  bool div;
  inline void pull(){
    size = 1 + lc->size + rc->size;
  }
  inline bool isBad(){
    return lc->size > alpha*size || rc->size > alpha*size;
  }
}pool[MAXN],*tail,*root,*recycle[MAXN],*null;
int rc_cnt;
void init(){
  tail = pool;
  null = tail++;
  null->lc = null->rc = null;
  null->size = 0;
  rc_cnt = 0;
  root = null;
}
Node *newNode(Point e){
  Node *p;
```

```
    if(rc_cnt)p = recycle[--rc_cnt];
    else p = tail++;
    p->e = e;
    p->lc = p->rc = null;
    p->size = 1;
    return p;
  }
  Node *build(Point *a,int l,int r,bool div){
    if(l >= r)return null;
    int mid = (l+r)/2;
    nth_element(a+l,a+mid,a+r,div?cmpY:cmpX);
    Node *p = newNode(a[mid]);
    p->div = div;
    p->lc = build(a,l,mid,!div);
    p->rc = build(a,mid+1,r,!div);
    p->pull();
    return p;
  }
  void getTree(Node *p,vector<Point>& v){
    if(p==null) return;
    getTree(p->lc,v);
    v.push_back(p->e);
    recycle[rc_cnt++]=p;
    getTree(p->rc,v);
  }
  Node *rebuild(vector<Point>& v,int l,int r,bool div){
    if(l>=r) return null;
    int mid = (l+r)/2;
    nth_element(v.begin()+l,v.begin()+mid,v.begin()+r,div?cmpY:cmpX);
    Node *p = newNode(v[mid]);
    p->div = div;
    p->lc = rebuild(v,l,mid,!div);
    p->rc = rebuild(v,mid+1,r,!div);
    p->pull();
    return p;
  }
  void rebuild(Node *&p){
    vector<Point> v;
    getTree(p,v);
    p = rebuild(v,0,v.size(),p->div);
  }
  Node **insert(Node *&p,Point a,bool div){
    if(p==null){
      p = newNode(a);
      p->div = div;
      return &null;
    }
    else{
      Node **res;
      if(cmp(a,p->e,div)) res=insert(p->lc,a,!div);
      else res=insert(p->rc,a,!div);
      p->pull();
      if(p->isBad()) res=&p;
      return res;
    }
  }
  void insert(Point e){
    Node **p = insert(root,e,0);
    if(*p!=null) rebuild(*p);
  }
  Node **get_min(Node *&p,bool div){
    if(p->div==div){
      if(p->lc!=null) return get_min(p->lc,div);
      else return &p;
    }
    else{
      Node **res=&p,**tmp;
      if(p->lc!=null){
        tmp = get_min(p->lc,div);
        if(cmp((*tmp)->e,(*res)->e,div)) res=tmp;
      }
      if(p->rc!=null){
        tmp = get_min(p->rc,div);
        if(cmp((*tmp)->e,(*res)->e,div)) res=tmp;
      }
      return res;
    }
  }
  void del(Node *&p){
    Node **nxt;
    if(p->rc!=null){
      nxt = get_min(p->rc,p->div);
      p->e = (*nxt)->e;
      del(*nxt);
    }
    else if(p->lc!=null){
      nxt = get_min(p->lc,p->div);
      p->e = (*nxt)->e;
      del(*nxt);
      p->rc = p->lc;
      p->lc = null;
    }
    else{
      recycle[rc_cnt++]=p;
      p=null;
    }
  }
  void del(Node *&p,Point d){
    if(p->e==d){
      del(p);
    }
    else if(cmp(d,p->e,p->div)) del(p->lc,d);
    else del(p->rc,d);
  }
  void search(Point p,Node *t,bool div,int m){
    if(!t)return;
    if(cmp(p,t->e,div)){
      search(p,t->lc,!div,m);
      if(q.size() < m){
        q.push(qnode(t->e,p.distance(t->e)));
        search(p,t->rc,!div,m);
      }
      else {
        if(p.distance(t->e) <= q.top().dis){
          q.push(qnode(t->e,p.distance(t->e)));
          q.pop();
        }
        if(!div){
          if(sq(t->e.x-p.x) <= q.top().dis)
            search(p,t->rc,!div,m);
        }
        else {
          if(sq(t->e.y-p.y) <= q.top().dis)
            search(p,t->rc,!div,m);
        }
      }
    }
    else {
      search(p,t->rc,!div,m);
      if(q.size() < m){
        q.push(qnode(t->e,p.distance(t->e)));
        search(p,t->lc,!div,m);
      }
      else {
        if(p.distance(t->e) <= q.top().dis){
          q.push(qnode(t->e,p.distance(t->e)));
          q.pop();
        }
        if(!div){
          if(sq(t->e.x-p.x) <= q.top().dis)
            search(p,t->lc,!div,m);
        }
        else {
          if(sq(t->e.y-p.y) <= q.top().dis)
            search(p,t->lc,!div,m);
        }
      }
    }
  }
  void search(Point p,int m){
    while(!q.empty())q.pop();
    search(p,root,0,m);
  }
  void getRange(Node *p,vector<Point>& v,int x1,int x2,int y1,int y2){
    if(p==null) return;
    if(x1<=p->e.x && p->e.x<=x2 && y1<=p->e.y && p->e.y<=y2) v.push_back(p->e
    );
    if(p->div ? y1<=p->e.y : x1<=p->e.x) getRange(p->lc,v,x1,x2,y1,y2);
    if(p->div ? y2>=p->e.y : x2>=p->e.x) getRange(p->rc,v,x1,x2,y1,y2);
  }
  void solve(Point p){
    del(root,p);
    insert(p);
  }
};
KDTree::Point p[MAXN];
int main(){
  KDTree::init();
  KDTree::root = KDTree::build(p,0,n,0);
  while(q--){
    KDTree::Point tmp,p1,p2;
    scanf("%d%d",&tmp.x,&tmp.y);
    search(tmp,2);
    p1=KDTree::q.top().p;
    KDTree::q.pop();
    p2=KDTree::q.top().p;
    KDTree::q.pop();
  }
  return 0;
}
```