

NCTU_Yggdarsill

Contents

1 Building Environment	1
1.1 C++	1
1.2 Default	1
1.3 Preferences	2
1.4 Print File	2
1.5 Vimrc	2
2 Convolution	2
2.1 FFT	2
2.2 SunMoon FFT	3
3 Geometry	4
3.1 Geometry	4
3.2 MinimumCoveringCircle	6
4 GNU Black Magic	6
4.1 GNU Bitwise Operation	6
5 Graph	7
5.1 Articulation Point	7
5.2 BCC	7
5.3 MST Directed	8
5.4 SCC	8
6 Java	9
6.1 Big Integer	9
6.2 Prime	10
7 Matching	10
7.1 Bipartite Matching	10
7.2 Blossom	10
7.3 Dinic	12
7.4 General Weighted Matching	12
7.5 KM	16
7.6 Min Cost Flow	17
7.7 Stable Marriage	17
8 Mathematics	18
8.1 Extended GCD	18
8.2 Lucas's Theorem	18

8.3 Miller-Rabin	19
9 String	19
9.1 AC Automaton	19
9.2 BWT	20
9.3 Suffix Array	21
9.4 Suffix Automaton	21
9.5 Z Algorithm	22
10 Struct	22
10.1 Treap	22
11 Tree	23
11.1 Heavy Light Decomposition	23

1 Building Environment

1.1 C++

```
1 {
2     "shell_cmd": "g++ -Wall \"${file}\" -o \"${file_path}/${file_base_name}\"
3     ,
4     "file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)??:?(.*)$",
5     "working_dir": "${file_path}",
6     "selector": "source.c, source.c++",
7
8     "variants":
9     [
10         {
11             "name": "Run",
12             "shell_cmd": "g++ -Wall \"${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 \"\"\"
13         }
14     ]
15 }
```

1.2 Default

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

```

8 using namespace std;
9 using namespace __gnu_pbds;
10 const double PI = acos(-1);
11 main() {
12     ios_base::sync_with_stdio(false);
13     cin.tie(NULL);
14     cout << fixed << setprecision(7) << PI << endl;
15 }

```

1.3 Preferences

```

1 {
2     "color_scheme": "Packages/Color Scheme - Default/Monokai Bright.tmTheme",
3     // "font_face": "Courier New", // Uncomment if defaults is proportional.
4     "font_size": 18
5 }

```

1.4 Print File

```

1 import sublime, sublime_plugin
2 import os
3
4 class print_file(sublime_plugin.TextCommand):
5     def run(self, edit):
6         os.system('cat -n "%s" > tmp.print; lpr tmp.print' % self.view.file_name
7             ())
8         self.view.show_popup("JIZZ!!")

```

1.5 Vimrc

```

1 set tabstop=4
2 set autoindent
3
4 map <F9> :w<LF>:!g++ -O2 -std=c++11 -o %.out % && echo "----Start----" &&
5     ./%.out<LF>
6
7 imap <F9> <ESC><F9>

```

2 Convolution

2.1 FFT

```

1 #include <cstdio>
2 #include <cstring>
3 #include <cmath>
4 const double PI=acos(-1.0);
5 typedef struct {
6     double real;
7     double im;
8 } COMPLEX;
9 COMPLEX X[66000],Y[66000],A[66000];
10 COMPLEX EE(COMPLEX a,COMPLEX b)
11 {
12     COMPLEX c;
13     c.real=a.real*b.real-a.im*b.im;
14     c.im=a.real*b.im+a.im*b.real;
15     return c;
16 }
17 /* 1 FFT , -1 IFFT */
18 void fft(COMPLEX x[],int nfft,int isign)
19 {
20     int i,j=0,k;
21     COMPLEX t;
22     for(i=1, j = nfft / 2;i<nfft-1;i++)
23     {
24         if(i<j)
25         {
26             t=x[j];
27             x[j]=x[i];
28             x[i]=t;
29         }
30         k=nfft/2;
31         while(k<=j)
32         {
33             j-=k;
34             k/=2;
35         }
36         if (j < k)
37             j+=k;
38     }
39     int le,lei,ip;
40     COMPLEX u,w, v;
41     for(le=2;le<=nfft;le *= 2)
42     {
43         lei=le/2;
44         w.real=cos(2.0*PI*isign/le);
45         w.im=sin(2.0*PI*isign/le);
46         for(i=0;i<nfft;i+=le)
47         {
48             u.real=1.0;
49             u.im=0.0;
50             for(j = i ; j < i + lei ; ++j)
51             {
52                 ip=j+lei;
53                 v = x[j];
54                 t=EE(u, x[ip]);
55                 x[j].real=v.real+t.real;
56                 x[j].im=v.im+t.im;

```

```

57         x[ip].real=v.real-t.real;
58         x[ip].im=v.im-t.im;
59         u=EE(u,w);
60     }
61 }
62 }
63 }
64 void FFT(COMPLEX x[], int nfft)
65 {
66     fft(x,nfft,1);
67 }
68 void IFFT(COMPLEX x[],int nfft)
69 {
70     int i;
71     fft(x,nfft,-1);
72
73     for(i=0;i<nfft;i++)
74     {
75         x[i].real /= nfft;
76         x[i].im /= nfft;
77     }
78 }
79 int main(void) {
80     int t_num;
81     int i,ii,iii;
82     int p_num;
83     int Nx;
84     int NFFT;
85     int temp;
86     scanf("%d",&t_num);
87     for(i=0;i<t_num;i++){
88         scanf("%d",&p_num);
89         Nx=p_num*2-1;
90         NFFT = 2 << (int)log2(Nx);
91         for(ii=0;ii<p_num;++ii){
92             scanf("%d",&temp);
93             X[ii].real=(double)temp;
94             X[ii].im=0.0;
95         }
96         for(iii=0;iii<p_num;++iii){
97
98             scanf("%d",&temp);
99             Y[iii].real=(double)temp;
100             Y[iii].im=0.0;
101         }
102         for(ii=p_num;ii<NFFT;ii++)
103         {
104             X[ii].real=0.0;
105             X[ii].im=0.0;
106             Y[ii].real=0.0;
107             Y[ii].im=0.0;
108         }
109         FFT(X,NFFT);
110         FFT(Y,NFFT);
111         for(ii=0;ii<NFFT;ii++){
112             A[ii] = EE(X[ii], Y[ii]);

```

```

113     }
114     IFFT(A,NFFT);
115     for(ii=0;ii<Nx;ii++){
116         printf("%d ", (int)round(A[ii].real));
117     }
118     printf("\n");
119 }
120 return 0;
121 }

```

2.2 SunMoon FFT

```

1 #ifndef SUNMOON_FFT
2 #define SUNMOON_FFT
3 #include<vector>
4 #include<complex>
5 #include<algorithm>
6 template<typename T,typename VT=std::vector<std::complex<T> > >
7 struct FFT{
8     const T pi;
9     FFT(const T pi=acos((T)-1)):pi(pi){}
10    inline unsigned int bit_reverse(unsigned int a,int len){
11        a=((a&0x55555555U)<<1)|((a&0xAAAAAAAAU)>>1);
12        a=((a&0x33333333U)<<2)|((a&0xCCCCCCCCU)>>2);
13        a=((a&0x0F0F0F0FU)<<4)|((a&0xF0F0F0F0U)>>4);
14        a=((a&0x00FF00FFU)<<8)|((a&0xFF00FF00U)>>8);
15        a=((a&0x0000FFFFU)<<16)|((a&0xFFFF0000U)>>16);
16        return a>>(32-len);
17    }
18    inline void fft(bool is_inv,VT &in,VT &out,int N){
19        int bitlen=std::__lg(N),num=is_inv?-1:1;
20        for(int i=0;i<N;++i)out[bit_reverse(i,bitlen)]=in[i];
21        for(int step=2;step<=N;step<<=1){
22            const int mh=step>>1;
23            for(int i=0;i<mh;++i){
24                std::complex<T> wi=exp(std::complex<T>(0,i*num*pi/mh));
25                for(int j=i;j<N;j+=step){
26                    int k=j+mh;
27                    std::complex<T> u=out[j],t=wi*out[k];
28                    out[j]=u+t;
29                    out[k]=u-t;
30                }
31            }
32        }
33        if(is_inv)for(int i=0;i<N;++i)out[i]/=N;
34    }
35 };
36 #endif

```

3 Geometry

3.1 Geometry

```

1 const double eps = 1e-10;
2 const double INF = 1.0/0.0;
3 const double SIDE = 10000;
4 const double PI = acos(-1.0);
5 const int MAXN = 500000 + 10;
6 struct PT{
7     double x,y;
8     PT(){}
9     PT(double x,double y):x(x),y(y){}
10    PT operator + (const PT& p)const{
11        return PT(x+p.x,y+p.y);
12    }
13    PT operator - (const PT& p)const{
14        return PT(x-p.x,y-p.y);
15    }
16    PT operator * (double c)const{
17        return PT(x*c,y*c);
18    }
19    PT operator / (double c)const{
20        return PT(x/c,y/c);
21    }
22    PT rot(double a)const{return PT(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a));}
23    double operator *(const PT& p)const{
24        return x*p.x+y*p.y;
25    }
26    double operator ^(const PT& p)const{
27        return x*p.y-y*p.x;
28    }
29    bool operator ==(const PT& p)const{
30        return fabs(x-p.x)<eps&&fabs(y-p.y)<eps;
31    }
32    double len2()const{return x*x+y*y;}
33    double len()const{return sqrt(len2());}
34 }poi[MAXN],stk[MAXN];
35 struct LINE{
36     PT a,b;
37     double angle;
38     LINE(){}
39     LINE(PT _a,PT _b):a(_a),b(_b),angle(atan2(_b.y-_a.y,_b.x-_a.x)){}
40 }line[MAXN],deq[MAXN];
41 int top;
42 inline int ori(const PT& p1,const PT& p2,const PT& p3){
43     double a=(p2-p1)^(p3-p1);
44     if(a>-eps&&a<eps)return 0;
45     return a>0 ? 1:-1;
46 }
47 inline bool btw(const PT& p1,const PT& p2,const PT& p3){
48     return (p2-p1)*(p3-p1)<eps;
49 }
50 //segment intersection
51 inline bool intersection(const PT& p1,const PT& p2,const PT& p3,const PT& p4)

```

```

{
52     int a123=ori(p1,p2,p3);
53     int a124=ori(p1,p2,p4);
54     int a341=ori(p3,p4,p1);
55     int a342=ori(p3,p4,p2);
56     if(a123==0&&a124==0)return btw(p1,p3,p4)||btw(p2,p3,p4)||btw(p3,p1,p2)||
        btw(p4,p1,p2);
57     return a123*a124 <= 0 && a341*a342 <= 0;
58 }
59 inline PT intersectionPoint(const PT& p1,const PT& p2,const PT& p3,const PT&
    p4){
60     double a123=(p2-p1)^(p3-p1);
61     double a124=(p2-p1)^(p4-p1);
62     return (p4*a123-p3*a124)/(a123-a124);
63 }
64 //line intersection
65 inline PT intersectionPoint(const LINE& l1,const LINE& l2){
66     PT p1=l1.a,p2=l1.b,p3=l2.a,p4=l2.b;
67     double a123=(p2-p1)^(p3-p1);
68     double a124=(p2-p1)^(p4-p1);
69     return (p4*a123-p3*a124)/(a123-a124);
70 }
71 PT foot(const LINE& l,const PT& p){
72     PT m(l.b.y-l.a.y,l.a.x-l.b.x);
73     return p+m*(l.a-p ^ l.b-p)/((l.b-l.a).len2());
74 }
75 PT mirror(const LINE& l,const PT& p){
76     PT m(l.b.y-l.a.y,l.a.x-l.b.x);
77     return p+m*(l.a-p ^ l.b-p)/((l.b-l.a).len2())*2;
78 }
79 //segment-point distance
80 inline double sp_dis(PT a,PT l1,PT l2){
81     if((a-l1)*(l2-l1)<0) return (l1-a).len();
82     else if((a-l2)*(l1-l2)<0) return (l2-a).len();
83     return fabs(l1-a^l2-a)/((l2-l1).len());
84 }
85
86 struct cir{
87     point c;
88     double r;
89 }o[10];
90 double out_ang(cir a,cir b){ //a.c+(b.c-a.c).unit().rot(ang)*b.r
91     return acos((a.r-b.r)/(a.c-b.c).len());
92 }
93 double in_ang(cir a,cir b){
94     return acos((a.r+b.r)/(a.c-b.c).len());
95 }
96 int main(){
97     double tmp,sum;
98     if(fabs(o[i].r-o[j].r)<(o[j].c-o[i].c).len()){
99         tmp = out_ang(o[i],o[j]);
100        sum = ang_add(cl,tmp);
101        pi=o[i].c+point(o[i].r*cos(sum),o[i].r*sin(sum));
102        pj=o[j].c+point(o[j].r*cos(sum),o[j].r*sin(sum));
103        sum = ang_add(cl,-tmp);
104        pi=o[i].c+point(o[i].r*cos(sum),o[i].r*sin(sum));

```

```

105     pj=o[j].c+point(o[j].r*cos(sum),o[j].r*sin(sum));
106 }
107 if(o[i].r+o[j].r<(o[j].c-o[i].c).len()){
108     tmp = in_ang(o[i],o[j]);
109     sum = ang_add(cl,tmp);
110     pi=o[i].c+point(o[i].r*cos(sum),o[i].r*sin(sum));
111     pj=o[j].c-point(o[j].r*cos(sum),o[j].r*sin(sum));
112     sum = ang_add(cl,-tmp);
113     pi=o[i].c+point(o[i].r*cos(sum),o[i].r*sin(sum));
114     pj=o[j].c-point(o[j].r*cos(sum),o[j].r*sin(sum));
115 }
116 }
117
118 inline double dist(const PT& p1,const PT& p2){
119     return sqrt((p2-p1)*(p2-p1));
120 }
121 inline double tri(const PT& p1,const PT& p2,const PT& p3){
122     return fabs((p2-p1)^(p3-p1));
123 }
124 inline double getPerimeter(){
125     double res=0.0;
126     poi[top++]=poi[0];
127     for(int i=0;i<top-1;i++)res+=dist(poi[i],poi[i+1]);
128     return res;
129 }
130 inline double getarea(){
131     double res=0.0;
132     for(int i=1;i<top-1;i++)res+=tri(poi[0],poi[i],poi[i+1]);
133     return 0.5*res;
134 }
135
136 //convex hull
137 inline bool cmp_convex(const PT &a,const PT &b){
138     if(a.x!=b.x)return a.x<b.x;
139     return a.y<b.y;
140 }
141 inline void convex_hull(PT a[],int &n){
142     top=0;
143     sort(a,a+n,cmp_convex);
144     for(int i=0;i<n;i++){
145         while(top>=2&&ori(stk[top-2],stk[top-1],a[i])>=0)top--;
146         stk[top++]=a[i];
147     }
148     for(int i=n-2,t=top+1;i>=0; i--){
149         while(top>=t&&ori(stk[top-2],stk[top-1],a[i])>=0)top--;
150         stk[top++]=a[i];
151     }
152     top--;
153     for(int i=0;i<top;i++)poi[i]=stk[i];
154 }
155 //half plane intersection
156 inline bool cmp_half_plane(const LINE &a,const LINE &b){
157     if(fabs(a.angle-b.angle)<eps)return ori(a.a,a.b,b.a)<0;
158     return a.angle > b.angle;
159 }
160 inline void half_plane_intersection(LINE a[],int &n){

```

```

161     int m=1,front=0,rear=1;
162     sort(a,a+n,cmp_half_plane);
163     for(int i=1;i<n;i++){
164         if(fabs(a[i].angle-a[m-1].angle)>eps)a[m++]=a[i];
165     }
166     deq[0]=a[0],deq[1]=a[1];
167     for(int i=2;i<m;i++){
168         while(front<rear&&ori(a[i].a,a[i].b,intersectionPoint(deq[rear],deq[
169             rear-1]))<0)rear--;
170         while(front<rear&&ori(a[i].a,a[i].b,intersectionPoint(deq[front],deq[
171             front+1]))<0)front++;
172         deq[++rear]=a[i];
173     }
174     while(front<rear&&ori(deq[front].a,deq[front].b,intersectionPoint(deq[rear
175         ],deq[rear-1]))<0)rear--;
176     while(front<rear&&ori(deq[rear].a,deq[rear].b,intersectionPoint(deq[front
177         ],deq[front+1]))<0)front++;
178     if(front==rear)return;
179 }
180
181 top=0;
182 for(int i=front;i<rear;i++)poi[top++]=intersectionPoint(deq[i],deq[i+1]);
183 if(rear>front+1)poi[top++]=intersectionPoint(deq[front],deq[rear]);
184
185 //smallest cover rectangle
186 double ans1,ans2;
187 void rotating_calipers(){
188     ans1=ans2=INF;
189     int j=1,k=1,l=1;
190     poi[top]=poi[0];
191     for(int i=0;i<top;i++){
192         while(tri(poi[i],poi[i+1],poi[j])<tri(poi[i],poi[i+1],poi[j+1])) j=(j
193             +1)%top;
194         while(((poi[i+1]-poi[i])*(poi[k+1]-poi[k]))>eps)k=(k+1)%top;
195         if(i==0)l=(k+1)%top;
196         while(((poi[i+1]-poi[i])*(poi[l+1]-poi[l]))<=-eps)l=(l+1)%top;
197         double tmp1 = tri(poi[i],poi[i+1],poi[j])/dist(poi[i],poi[i+1]);
198         double tmp2 = (((poi[k]-poi[i])*(poi[i+1]-poi[i]))-((poi[l]-poi[i])*(
199             poi[i+1]-poi[i]))) / dist(poi[i],poi[i+1]);
200         if((tmp1+tmp2)*2.0<ans1)ans1=(tmp1+tmp2)*2.0;
201         if(tmp1*tmp2<ans2)ans2=tmp1*tmp2;
202     }
203 }
204 int main(){
205     int n,m;
206     while(~scanf("%d",&n)&&n){
207         for(int i=0;i<n;i++)scanf("%lf%lf",&poi[i].x,&poi[i].y);
208         convex_hull(poi,n);
209         rotating_calipers();
210         printf("%.2f %.2f\n",ans2,ans1);
211     }
212 }

```

```

211 inline bool online(const LINE &L, const PT &p) {
212     return ori(p, L.a, L.b) == 0 && btw(p, L.a, L.b);
213 }
214 inline bool on_convex(const PT& p) {
215     for(int i=0; i<top; i++)
216         if(p==poi[i]) return 1;
217     poi[top]=poi[0];
218     for(int i=0; i<top; i++) {
219         line[i].a=poi[i];
220         line[i].b=poi[i+1];
221     }
222     for(int i=0; i<top; i++)
223         if(online(line[i], p)) return 1;
224     return 0;
225 }
226 //originally in long long, should be modified
227 bool in_simple_polygon(PT b[], int k) {
228     bool flag=false;
229     for(int j=0; j<k; j++) {
230         if(((p-b[j])^(p-b[(j+1)%k]))==0 && (p-b[j])*(p-b[(j+1)%k])<=0) {
231             flag=true;
232             break;
233         }
234         if((b[j].y<p.y)^(b[(j+1)%k].y<p.y)) {
235             long long xss=(b[j]-p)^(b[(j+1)%k]-p);
236             if((xss<0)^(b[j].y<b[(j+1)%k].y)) {
237                 flag^=1;
238             }
239         }
240     }
241     return flag;
242 }

```

3.2 MinimumCoveringCircle

```

1 #define F(n) Fi(i,n)
2 #define Fi(i,n) Fl(i,0,n)
3 #define Fl(i,l,n) for(int i=(l); i<(int)(n); ++i)
4 #include <bits/stdc++.h>
5 using namespace std;
6 const double eps = 1e-6;
7 #define x first
8 #define y second
9 typedef pair<double, double> point;
10 inline double dq(const point& p1, const point& p2) {
11     return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
12 }
13 inline point oc(const point& pa, const point& pb, const point& pc) {
14     double a, b, c, d, e, f, delta, dx, dy;
15     // ax + by = c
16     // dx + ey = f
17     a = pa.x - pb.x;
18     b = pa.y - pb.y;
19     c = a*(pa.x+pb.x)/2 + b*(pa.y+pb.y)/2;

```

```

20     d = pa.x - pc.x;
21     e = pa.y - pc.y;
22     f = d*(pa.x+pc.x)/2 + e*(pa.y+pc.y)/2;
23     delta = a*e-b*d;
24     dx = c*e-f*b;
25     dy = a*f-d*c;
26     return point(dx/delta, dy/delta);
27 }
28 inline point enc(const vector<point>& tmp) {
29     point O = tmp[0];
30     double r = 0;
31     Fl(i, 1, tmp.size()) if (dq(O, tmp[i]) - r > eps) {
32         O = tmp[i], r = 0;
33         Fi(j, i) if (dq(O, tmp[j]) - r > eps) {
34             O = point((tmp[i].x+tmp[j].x)/2, (tmp[i].y+tmp[j].y)/2);
35             r = dq(O, tmp[j]);
36             Fi(k, j) if (dq(O, tmp[k]) - r > eps)
37                 O = oc(tmp[i], tmp[j], tmp[k]), r = dq(O, tmp[k]);
38         }
39     }
40     return O;
41 }
42 int n;
43 vector<point> v;
44 int main() {
45     ios_base::sync_with_stdio(false);
46     cin.tie(NULL);
47     while (cin >> n) {
48         if (!n) break;
49         v.clear();
50         F(n) {
51             point tp;
52             cin >> tp.x >> tp.y;
53             v.push_back(tp);
54         }
55         point ct = enc(v);
56         cout << setprecision(2) << fixed << ct.x << ' ' << ct.y << ' ' << dq(ct, v[0]) << '\n';
57     }
58 }

```

4 GNU Black Magic

4.1 GNU Bitwise Operation

```

1 int __builtin_ffs (unsigned int x)
2 int __builtin_ffsl (unsigned long)
3 int __builtin_ffsll (unsigned long long)
4 // 返回右起第一個1的位置
5 // Returns one plus the index of the least significant 1-bit of x, or if x is
   zero, returns zero.
6

```

```

7 int __builtin_clz (unsigned int x)
8 int __builtin_clzl (unsigned long)
9 int __builtin_clzll (unsigned long long)
10 // 返回左起第一個1之前0的個數
11 // Returns the number of leading 0-bits in x, starting at the most
    significant bit position. If x is 0, the result is undefined.
12
13 int __builtin_ctz (unsigned int x)
14 int __builtin_ctzl (unsigned long)
15 int __builtin_ctzll (unsigned long long)
16 // 返回右起第一個1之後的0的個數
17 // Returns the number of trailing 0-bits in x, starting at the least
    significant bit position. If x is 0, the result is undefined.
18
19 int __builtin_popcount (unsigned int x)
20 int __builtin_popcountl (unsigned long)
21 int __builtin_popcountll (unsigned long long)
22 // 返回1的個數
23 // Returns the number of 1-bits in x.
24
25 int __builtin_parity (unsigned int x)
26 int __builtin_parityl (unsigned long)
27 int __builtin_parityll (unsigned long long)
28 // 返回1的個數的奇偶性(1的個數 mod 2的值)
29 // Returns the parity of x, i.e. the number of 1-bits in x modulo 2.

```

5 Graph

5.1 Articulation Point

```

1 void tarjan(int u, int p)
2 { // u 為當前點, p 為當前點之母節點
3   // cnt 為 DFS 次序
4   low[u] = dfn[u] = ++cnt;
5   int i, v;
6   for (i = 0 ; i < G[u].size() ; ++i) {
7     v = G[u][i];
8     if (u == rt && !dfn[v]) ++c;
9     if (!dfn[v]){
10      // (u, v) 為 Tree Edge
11      tarjan(v, u);
12      low[u] = min(low[u], low[v]);
13      // To check if u is AP or not.
14      if (dfn[u] <= low[v] && u != rt) ge[u] = 1;
15    }
16    // 注意不可以同一條邊走兩次，且根節點特判
17    if (v != p && p != -1) low[u] = min(low[u], dfn[v]);
18  }
19 }

```

5.2 BCC

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int MAXN = 10000;
4 vector <int> adja[MAXN];
5 int gcnt, top, timeStamp, dfn[MAXN], low[MAXN], depth[MAXN];
6 pair<int, int> stk[MAXN], ans[MAXN];
7 set <int> group[MAXN];
8 bool cut[MAXN];
9 void BCC(int now, int nextv){
10   int sf, st;
11   group[gcnt].clear();
12   do{
13     sf = stk[top-1].first, st = stk[top-1].second;
14     group[gcnt].insert(sf);
15     group[gcnt].insert(st);
16     --top;
17   }while(sf != now || st != nextv);
18   ++gcnt;
19 }
20 void tarjan(int now, int parent, int d){
21   int child = 0;
22   dfn[now] = low[now] = ++timeStamp, depth[now] = d;
23   for(int i = 0; i < adja[now].size(); i++){
24     int nextv = adja[now][i];
25     if(nextv == parent) continue;
26     if(dfn[nextv] == 0){
27       stk[top++] = make_pair(now, nextv);
28       tarjan(nextv, now, d+1);
29       low[now] = min(low[now], low[nextv]);
30       ++child;
31       if( (parent != -1 && low[nextv] >= dfn[now]) || (parent == -1 &&
    child >= 2)){
32         cut[now] = true;
33         if(parent != -1) BCC(now, nextv);
34       }
35       if(parent == -1) BCC(now, nextv);
36     }
37     else if(depth[nextv] < depth[now]-1){
38       stk[top++] = make_pair(now, nextv);
39       low[now] = min(low[now], dfn[nextv]);
40     }
41   }
42 }
43 int main(){
44   int n,m,x,y,cnt=0;
45   while(~scanf("%d",&n)){
46     cnt=timeStamp=top=gcnt=0;
47     memset(cut, 0, sizeof(cut));
48     memset(dfn, 0, sizeof(dfn));
49     for(int i=0;i<n;i++)adja[i].clear();
50     for(int i=0;i<n;i++){
51       scanf("%d ",&x);
52       scanf("(%d",&m);
53       while(m--){

```

```

54         scanf("%d",&y);
55         adj[a[x].push_back(y);
56     }
57 }
58 for(int i=0;i<n;i++)
59     if(dfn[i]==0)tarjan(i, -1, 1);
60 for(int i=0;i<gcnt;i++){
61     if(group[i].size()==2){
62         //critical links
63     }
64 }
65 }
66 }

```

5.3 MST Directed

```

1 #include<cstdio>
2 #include<vector>
3 #include<algorithm>
4 #define N 100100
5 using namespace std;
6 struct edge{
7     edge(){}
8     edge(int _f,int _d):f(_f),d(_d){}
9     int f;
10    int d;
11    bool operator<(const edge &rhs)const{return d<rhs.d;}
12 };
13 struct node{
14     int sz,v,now;
15     node *l,*r;
16     void pull() {sz=1+(l?l->sz:0)+(r?r->sz:0);}
17 }pq[N];
18 int pa[N],sub[N],stk[N],top;
19 bool vis[N],instk[N];
20 vector<edge> rg[N];
21 void init(int n){
22     for(int i=0;i<n;i++){
23         pa[i]=i;
24         sub[i]=0;
25         pq[i].l=pq[i].r=NULL;
26         pq[i].sz=1;
27         pq[i].v=i;
28         pq[i].now=0;
29     }
30 }
31 int find(int x){
32     if(pa[x]==x) return x;
33     int y=find(pa[x]);
34     if(pa[x]!=y) sub[x]+=sub[pa[x]],pa[x]=y;
35     return pa[x];
36 }
37 inline int get_sub(int x){
38     if(x==find(x)) return sub[x];

```

```

39     else return sub[x]+sub[pa[x]];
40 }
41 inline int get_cost(const node& a){
42     return rg[a.v][a.now].d-get_sub(a.v);
43 }
44 bool cmp(const node& a,const node& b){
45     return get_cost(a)<get_cost(b);
46 }
47 node* merge(node *a,node *b){
48     if(!a||!b) return a?a:b;
49     if(cmp(*b,*a)) swap(a,b);
50     a->r=merge(a->r,b);
51     if((a->l?a->l->sz:0)<(a->r?a->r->sz:0)) swap(a->l,a->r);
52     a.pull();
53     return a;
54 }
55 int min_cost_arborescence(int r,int n){
56     vis[r]=true;
57     int res=0;
58     for(int i=0;i<n;i++){
59         if(!vis[i]){
60             top=0;
61             int u=i;
62             while(!vis[u]){
63                 // ...
64             }
65         }
66     }
67 }
68 int main(){
69     int n,m,r,x,y,w;
70     scanf("%d%d%d",&n,&m,&r);
71     for(int i=0;i<m;i++){
72         scanf("%d%d%d",&x,&y,&w);
73         rg[y].push_back(edge(x,w));
74     }
75     sort(rg[r].begin(),rg[r].end(),cmp);
76 }

```

5.4 SCC

```

1 #include <cstdlib>
2 #include <iostream>
3 #include <vector>
4 #include <queue>
5 #define N 300002
6 using namespace std;
7 vector<int>go[N],back[N],tree[N];
8 int hu[N],ST[N],st=0,scc[N],scCo[N],scmx[N];
9 bool wed[N];
10 int DFS_go(int now){
11     //cout<<now<<" DFS ";
12     wed[now]=true;
13     for(int i=0;i<go[now].size();i++){

```



```

14     if(!wed[go[now][i]])
15         DFS_go(go[now][i]);
16     }
17     ST[st++]=now;
18     return 0;
19 }
20 int DFS_back(int now,int id){
21     wed[now]=true;
22     scc[now]=id;
23     int sum=1;
24     if(now==0) sum=0;
25     for(int i=0;i<back[now].size();i++){
26         if(!wed[back[now][i]])
27             sum+=DFS_back(back[now][i],id);
28     }
29     return sum;
30 }
31 int DFS_tree(int now)
32 {
33     if(scmx[now]!=0) return scmx[now];
34     int mx=0,tmp;
35     for(int i=0;i<tree[now].size();i++){
36         tmp=DFS_tree(tree[now][i]);
37         mx=(mx>tmp)? mx:tmp;
38     }
39     scmx[now]=mx+scCo[now];
40     return mx+scCo[now];
41 }
42 int main(int argc,char *argv[])
43 {
44     ios_base::sync_with_stdio(false);
45     int n,k;
46     char c;
47     cin>>n>>k>>hu[1];
48     go[0].push_back(1);
49     back[1].push_back(0);
50     for(int i=2;i<=n;i++){
51         cin>>hu[i];
52         if(hu[i]>=hu[i-1]){
53             go[i].push_back(i-1);
54             back[i-1].push_back(i);
55         }
56         if(hu[i-1]>=hu[i]){
57             go[i-1].push_back(i);
58             back[i].push_back(i-1);
59         }
60         go[0].push_back(i);
61         back[i].push_back(0);
62     }
63     for(int i=1;i<=n;i++){
64         cin>>c;
65         if(c=='T'){
66             go[i].push_back(0);
67             back[0].push_back(i);
68         }
69     }

```

```

70     for(int i=0;i<=n;i++){
71         if(!wed[i])DFS_go(i);
72         //cout<<endl;
73         fill((bool*)wed,(bool*)wed+N,false);
74         int tsc=0;
75         // for(int i=0;i<st;i++)cout<<ST[i]<<" HH ";
76         // cout<<endl;
77         while(st!=0)
78             if(!wed[ST[--st]]){
79                 scCo[tsc]=DFS_back(ST[st],tsc);
80                 tsc++;
81             }
82         // for(int i=0;i<N;i++)
83         //     while(!back[i].empty())back[i].pop_back();
84         for(int i=0;i<=n;i++){
85             for(int j=0;j<go[i].size();j++){
86                 if(scc[i]!=scc[go[i][j]]){
87                     tree[scc[i]].push_back(scc[go[i][j]]);
88                 }
89             }
90             // for(int i=0;i<=n;i++)cout<<scc[i]<<" BB ";
91             // cout<<endl;
92             // for(int i=0;i<tsc;i++)cout<<scCo[i]<<" GG ";
93             cout<<DFS_tree(scc[k])<<endl;
94             //system("pause");
95             return 0;
96     }

```

6 Java

6.1 Big Integer

```

1 import java.math.*;
2 import java.io.*;
3 import java.util.*;
4 public class Main{
5     public static void main(String []argv){
6         c[0][0]=BigInteger.ONE;
7         for(int i=1;i<3001;i++){
8             c[i][0]=BigInteger.ONE;
9             c[i][i]=BigInteger.ONE;
10            for(int j=1;j<i;j++)c[i][j]=c[i-1][j].add(c[i-1][j-1]);
11        }
12        Scanner scanner = new Scanner(System.in);
13        int T = scanner.nextInt();
14        BigInteger x;
15        BigInteger ans;
16        while(T-- > 0){
17            ans = BigInteger.ZERO;
18            int n = scanner.nextInt();
19            for(int i=0;i<n;i++){
20                x = new BigInteger(scanner.next());

```

```

21         if(i%2 == 1) ans=ans.subtract(c[n-1][i].multiply(x));
22         else ans=ans.add(c[n-1][i].multiply(x));
23     }
24     if(n%2 == 0) ans=BigInteger.ZERO.subtract(ans);
25     System.out.println(ans);
26 }
27 }
28 }

```

6.2 Prime

```

1 import java.math.*;
2 import java.io.*;
3 import java.util.*;
4 public class Main{
5     public static void main(String []argv){
6         Scanner scanner = new Scanner(System.in);
7         int T = scanner.nextInt();
8         for (int cs = 0 ; cs < T ; cs++){
9             if (cs != 0) {
10                 System.out.println("");
11             }
12             int a = scanner.nextInt();
13             int b = scanner.nextInt();
14             for (int i = a ; i <= b ; i++) {
15                 BigInteger x = BigInteger.valueOf(i);
16                 if (x.isProbablePrime(5) == true) {
17                     System.out.println(x);
18                 }
19             }
20         }
21     }
22 }
23 }

```

7 Matching

7.1 Bipartite Matching

```

1 #include<bits/stdc++.h>
2 #define V 20100
3 #define inf 0x3f3f3f3f
4 int mx[V],my[V],dis[V],que[V];
5 bool vis[V];
6 vector<int> g[V];
7 bool DFS(int u){
8     vis[u]=true;
9     for(int i=0;i<g[u].size();i++){
10         int v=my[g[u][i]];

```

```

11         if (v!=-1 || !vis[v] && dis[v]==dis[u]+1 && DFS(v)) {
12             mx[u]=g[u][i];
13             my[g[u][i]]=u;
14             return true;
15         }
16     }
17     return false;
18 }
19 // n is the size of left hand side
20 int Hopcroft_Karp(int n){
21     int matching=0,qt,qf,sp,i,u,v;
22     bool flag=true;
23     memset(mx,-1,sizeof(mx));
24     memset(my,-1,sizeof(my));
25     while(flag){
26         flag=false;
27         qt=qf=0;
28         sp=inf;
29         for(i=0;i<n;i++){
30             if(mx[i]==-1){
31                 dis[i]=0;
32                 que[qt++]=i;
33             }
34             else dis[i]=inf;
35         }
36         while(qf<qt){
37             u=que[qf++];
38             if(dis[u]>=sp) continue;
39             for(i=0;i<g[u].size();i++){
40                 v=my[g[u][i]];
41                 if(v!=-1){
42                     if(dis[u]+1<sp){
43                         sp=dis[u]+1;
44                         flag=true;
45                     }
46                 }
47                 else if(dis[u]+1<dis[v]){
48                     dis[v]=dis[u]+1;
49                     que[qt++]=v;
50                 }
51             }
52         }
53         if(flag){
54             memset(vis,0,sizeof(vis));
55             for(i=0;i<n;i++){
56                 if(dis[i]==0 && DFS(i)) matching++;
57             }
58         }
59     }
60     return matching;
61 }

```

7.2 Blossom

```

1 int V;
2 bool adj[MAXN][MAXN];
3 int w[MAXN][MAXN];
4 int p[MAXN];
5 int m[MAXN];
6 int d[MAXN];
7 int c1[MAXN], c2[MAXN];
8 int q[MAXN], *qf, *qb;
9 int pp[MAXN];
10 int f(int x) {return x == pp[x] ? x : (pp[x] = f(pp[x]));}
11 void u(int x, int y) {pp[x] = y;}
12 int v[MAXN];
13 void path(int r, int x){
14     if (r == x) return;
15     if (d[x] == 0){
16         path(r, p[p[x]]);
17         int i = p[x], j = p[p[x]];
18         m[i] = j; m[j] = i;
19     }
20     else if (d[x] == 1){
21         path(m[x], c1[x]);
22         path(r, c2[x]);
23         int i = c1[x], j = c2[x];
24         m[i] = j; m[j] = i;
25     }
26 }
27 int lca(int x, int y, int r){
28     int i = f(x), j = f(y);
29     while (i != j && v[i] != 2 && v[j] != 1){
30         v[i] = 1; v[j] = 2;
31         if (i != r) i = f(p[i]);
32         if (j != r) j = f(p[j]);
33     }
34     int b = i, z = j; if (v[j] == 1) swap(b, z);
35     for (i = b; i != z; i = f(p[i])) v[i] = -1;
36     v[z] = -1;
37     return b;
38 }
39 void contract_one_side(int x, int y, int b){
40     for (int i = f(x); i != b; i = f(p[i])){
41         u(i, b);
42         if (d[i] == 1) c1[i] = x, c2[i] = y, *qb++ = i;
43     }
44 }
45 bool BFS(int r){
46     for (int i=0; i<V; ++i) pp[i] = i;
47     memset(v, -1, sizeof(v));
48     memset(d, -1, sizeof(d));
49     d[r] = 0;
50     qf = qb = q;
51     *qb++ = r;
52     while (qf < qb)
53         for (int x=*qf++, y=0; y<V; ++y)
54             if (adj[x][y] && m[y] != y && f(x) != f(y))
55                 if (d[y] == -1)
56                     if (m[y] == -1){

```

```

57                     path(r, x);
58                     m[x] = y; m[y] = x;
59                     return true;
60                 }
61             else{
62                 p[y] = x; p[m[y]] = y;
63                 d[y] = 1; d[m[y]] = 0;
64                 *qb++ = m[y];
65             }
66         else
67             if (d[f(y)] == 0) {
68                 int b = lca(x, y, r);
69                 contract_one_side(x, y, b);
70                 contract_one_side(y, x, b);
71             }
72     return false;
73 }
74 int match_result(){
75     int res=0;
76     memset(m, -1, sizeof(m));
77     for(int i=0; i<V; i++){
78         if(m[i]==-1){
79             if(BFS(i)) res++;
80             else m[i]=i;
81         }
82     }
83     return res;
84 }
85 int num[10000 + 10], top;
86 int main(){
87     int T, Case=0, n;
88     scanf("%d", &T);
89     while(T--){
90         scanf("%d", &n);
91         V = (1<<n);
92         top=0;
93         for(int i=0; i<V; i++){
94             for(int j=i+1; j<V; j++){
95                 scanf("%d", &w[i][j]);
96                 num[top++] = w[i][j];
97             }
98         }
99         sort(num, num+top);
100         top = (unique(num, num+top) - num);
101         int l=0, r=top-1, mid;
102         while(r>l){
103             mid = (l+r+1)/2;
104             memset(adj, false, sizeof(adj));
105             for(int i=0; i<V; i++){
106                 for(int j=i+1; j<V; j++){
107                     if(w[i][j] >= num[mid]) adj[i][j] = adj[j][i] = true;
108                 }
109             }
110             int res = match_result();
111             if(res == V/2) l = mid;
112             else r = mid-1;

```

```

113     }
114     printf("Case %d: %d\n", ++Case, num[l]);
115 }
116 }

```

7.3 Dinic

```

1 //Dinic
2 #define V 1000
3 struct edge{
4     edge() {}
5     edge(int a, int b, int c): to(a), cap(b), rev(c) {}
6     int to, cap, rev;
7 };
8 vector<edge> g[V];
9 int level[V];
10 int iter[V];
11 void add_edge(int from, int to, int cap){
12     g[from].push_back(edge(to, cap, g[to].size()));
13     g[to].push_back(edge(from, 0, g[from].size()-1));
14 }
15 void bfs(int s){
16     memset(level, -1, sizeof(level));
17     queue<int> que;
18     level[s]=0;
19     que.push(s);
20     while(!que.empty()){
21         int v=que.front();
22         que.pop();
23         for(int q=0; q<g[v].size(); q++){
24             edge &e=g[v][q];
25             if(e.cap>0&&level[e.to]<0){
26                 level[e.to]=level[v]+1;
27                 que.push(e.to);
28             }
29         }
30     }
31 }
32 int dfs(int v, int t, int f){
33     if(v==t) return f;
34     for(int &q=iter[v]; q<g[v].size(); ++q){
35         edge &e=g[v][q];
36         if(e.cap>0&&level[v]<level[e.to]){
37             int d=dfs(e.to, t, min(f, e.cap));
38             if(d>0){
39                 e.cap-=d;
40                 g[e.to][e.rev].cap+=d;
41                 return d;
42             }
43         }
44     }
45     return 0;
46 }
47 int max_flow(int s, int t){

```

```

48     int flow=0;
49     for(;;){
50         bfs(s);
51         if(level[t]<0) return flow;
52         memset(iter, 0, sizeof(iter));
53         int f;
54         while((f=dfs(s, t, 1e9))>0)
55             flow+=f;
56     }
57 }

```

7.4 General Weighted Matching

```

1 #include <iostream>
2 #include <cstdio>
3 #include <algorithm>
4 #include <vector>
5 using namespace std;
6
7 typedef long long s64;
8
9 const int INF = 2147483647;
10
11 const int MaxN = 400;
12 const int MaxM = 79800;
13
14 template <class T>
15 inline void tension(T &a, const T &b)
16 {
17     if (b < a)
18         a = b;
19 }
20 template <class T>
21 inline void relax(T &a, const T &b)
22 {
23     if (b > a)
24         a = b;
25 }
26 template <class T>
27 inline int size(const T &a)
28 {
29     return (int)a.size();
30 }
31
32 inline int getint()
33 {
34     char c;
35     while (c = getchar(), '0' > c || c > '9');
36
37     int res = c - '0';
38     while (c = getchar(), '0' <= c && c <= '9')
39         res = res * 10 + c - '0';
40     return res;
41 }

```

```

42
43 const int MaxNX = MaxN + MaxN;
44
45 struct edge
46 {
47     int v, u, w;
48
49     edge() {}
50     edge(const int &_v, const int &_u, const int &_w)
51         : v(_v), u(_u), w(_w) {}
52 };
53
54 int n, m;
55 edge mat[MaxNX + 1][MaxNX + 1];
56
57 int n_matches;
58 s64 tot_weight;
59 int mate[MaxNX + 1];
60 int lab[MaxNX + 1];
61
62 int q_n, q[MaxN];
63 int fa[MaxNX + 1], col[MaxNX + 1];
64 int slackv[MaxNX + 1];
65
66 int n_x;
67 int bel[MaxNX + 1], blofrom[MaxNX + 1][MaxN + 1];
68 vector<int> bloch[MaxNX + 1];
69
70 inline int e_delta(const edge &e) // does not work inside blossoms
71 {
72     return lab[e.v] + lab[e.u] - mat[e.v][e.u].w * 2;
73 }
74 inline void update_slackv(int v, int x)
75 {
76     if (!slackv[x] || e_delta(mat[v][x]) < e_delta(mat[slackv[x]][x]))
77         slackv[x] = v;
78 }
79 inline void calc_slackv(int x)
80 {
81     slackv[x] = 0;
82     for (int v = 1; v <= n; v++)
83         if (mat[v][x].w > 0 && bel[v] != x && col[bel[v]] == 0)
84             update_slackv(v, x);
85 }
86
87 inline void q_push(int x)
88 {
89     if (x <= n)
90         q[q_n++] = x;
91     else
92     {
93         for (int i = 0; i < size(bloch[x]); i++)
94             q_push(bloch[x][i]);
95     }
96 }
97 inline void set_mate(int xv, int xu)

```

```

98 {
99     mate[xv] = mat[xv][xu].u;
100     if (xv > n)
101     {
102         edge e = mat[xv][xu];
103         int xr = blofrom[xv][e.v];
104         int pr = find(bloch[xv].begin(), bloch[xv].end(), xr) - bloch[xv].begin();
105         if (pr % 2 == 1)
106         {
107             reverse(bloch[xv].begin() + 1, bloch[xv].end());
108             pr = size(bloch[xv]) - pr;
109         }
110
111         for (int i = 0; i < pr; i++)
112             set_mate(bloch[xv][i], bloch[xv][i ^ 1]);
113         set_mate(xr, xu);
114
115         rotate(bloch[xv].begin(), bloch[xv].begin() + pr, bloch[xv].end());
116     }
117 }
118 inline void set_bel(int x, int b)
119 {
120     bel[x] = b;
121     if (x > n)
122     {
123         for (int i = 0; i < size(bloch[x]); i++)
124             set_bel(bloch[x][i], b);
125     }
126 }
127
128 inline void augment(int xv, int xu)
129 {
130     while (true)
131     {
132         int xnu = bel[mate[xv]];
133         set_mate(xv, xu);
134         if (!xnu)
135             return;
136         set_mate(xnu, bel[fa[xnu]]);
137         xv = bel[fa[xnu]], xu = xnu;
138     }
139 }
140 inline int get_lca(int xv, int xu)
141 {
142     static bool book[MaxNX + 1];
143     for (int x = 1; x <= n_x; x++)
144         book[x] = false;
145     while (xv || xu)
146     {
147         if (xv)
148         {
149             if (book[xv])
150                 return xv;
151             book[xv] = true;
152             xv = bel[mate[xv]];

```

```

153     if (xv)
154         xv = bel[fa[xv]];
155     }
156     swap(xv, xu);
157 }
158 return 0;
159 }
160
161 inline void add_blossom(int xv, int xa, int xu)
162 {
163     int b = n + 1;
164     while (b <= n_x && bel[b])
165         b++;
166     if (b > n_x)
167         n_x++;
168
169     lab[b] = 0;
170     col[b] = 0;
171
172     mate[b] = mate[xa];
173
174     bloch[b].clear();
175     bloch[b].push_back(xa);
176     for (int x = xv; x != xa; x = bel[fa[bel[mate[x]]]])
177         bloch[b].push_back(x), bloch[b].push_back(bel[mate[x]]), q_push(bel[mate[
178             x]]);
179     reverse(bloch[b].begin() + 1, bloch[b].end());
180     for (int x = xu; x != xa; x = bel[fa[bel[mate[x]]]])
181         bloch[b].push_back(x), bloch[b].push_back(bel[mate[x]]), q_push(bel[mate[
182             x]]);
183
184     set_bel(b, b);
185
186     for (int x = 1; x <= n_x; x++)
187     {
188         mat[b][x].w = mat[x][b].w = 0;
189         blofrom[b][x] = 0;
190     }
191     for (int i = 0; i < size(bloch[b]); i++)
192     {
193         int xs = bloch[b][i];
194         for (int x = 1; x <= n_x; x++)
195             if (mat[b][x].w == 0 || e_delta(mat[xs][x]) < e_delta(mat[b][x]))
196                 mat[b][x] = mat[xs][x], mat[x][b] = mat[x][xs];
197         for (int x = 1; x <= n_x; x++)
198             if (blofrom[xs][x])
199                 blofrom[b][x] = xs;
200     }
201     calc_slackv(b);
202 }
203 inline void expand_blossom1(int b) // lab[b] == 1
204 {
205     for (int i = 0; i < size(bloch[b]); i++)
206         set_bel(bloch[b][i], bloch[b][i]);
207
208     int xr = blofrom[b][mat[b][fa[b]].v];

```

```

207     int pr = find(bloch[b].begin(), bloch[b].end(), xr) - bloch[b].begin();
208     if (pr % 2 == 1)
209     {
210         reverse(bloch[b].begin() + 1, bloch[b].end());
211         pr = size(bloch[b]) - pr;
212     }
213
214     for (int i = 0; i < pr; i += 2)
215     {
216         int xs = bloch[b][i], xns = bloch[b][i + 1];
217         fa[xs] = mat[xns][xs].v;
218         col[xs] = 1, col[xns] = 0;
219         slackv[xs] = 0, calc_slackv(xns);
220         q_push(xns);
221     }
222     col[xr] = 1;
223     fa[xr] = fa[b];
224     for (int i = pr + 1; i < size(bloch[b]); i++)
225     {
226         int xs = bloch[b][i];
227         col[xs] = -1;
228         calc_slackv(xs);
229     }
230
231     bel[b] = 0;
232 }
233 inline void expand_blossom_final(int b) // at the final stage
234 {
235     for (int i = 0; i < size(bloch[b]); i++)
236     {
237         if (bloch[b][i] > n && lab[bloch[b][i]] == 0)
238             expand_blossom_final(bloch[b][i]);
239         else
240             set_bel(bloch[b][i], bloch[b][i]);
241     }
242     bel[b] = 0;
243 }
244
245 inline bool on_found_edge(const edge &e)
246 {
247     int xv = bel[e.v], xu = bel[e.u];
248     if (col[xu] == -1)
249     {
250         int nv = bel[mate[xu]];
251         fa[xu] = e.v;
252         col[xu] = 1, col[nv] = 0;
253         slackv[xu] = slackv[nv] = 0;
254         q_push(nv);
255     }
256     else if (col[xu] == 0)
257     {
258         int xa = get_lca(xv, xu);
259         if (!xa)
260         {
261             augment(xv, xu), augment(xu, xv);
262             for (int b = n + 1; b <= n_x; b++)

```

```

263     if (bel[b] == b && lab[b] == 0)
264         expand_blossom_final(b);
265     return true;
266 }
267 else
268     add_blossom(xv, xa, xu);
269 }
270 return false;
271 }
272
273 bool match()
274 {
275     for (int x = 1; x <= n_x; x++)
276         col[x] = -1, slackv[x] = 0;
277
278     q_n = 0;
279     for (int x = 1; x <= n_x; x++)
280         if (bel[x] == x && !mate[x])
281             fa[x] = 0, col[x] = 0, slackv[x] = 0, q_push(x);
282     if (q_n == 0)
283         return false;
284
285     while (true)
286     {
287         for (int i = 0; i < q_n; i++)
288         {
289             int v = q[i];
290             for (int u = 1; u <= n; u++)
291                 if (mat[v][u].w > 0 && bel[v] != bel[u])
292                 {
293                     int d = e_delta(mat[v][u]);
294                     if (d == 0)
295                     {
296                         if (on_found_edge(mat[v][u]))
297                             return true;
298                     }
299                     else if (col[bel[u]] == -1 || col[bel[u]] == 0)
300                         update_slackv(v, bel[u]);
301                 }
302             }
303
304             int d = INF;
305             for (int v = 1; v <= n; v++)
306                 if (col[bel[v]] == 0)
307                     tension(d, lab[v]);
308             for (int b = n + 1; b <= n_x; b++)
309                 if (bel[b] == b && col[b] == 1)
310                     tension(d, lab[b] / 2);
311             for (int x = 1; x <= n_x; x++)
312                 if (bel[x] == x && slackv[x])
313                 {
314                     if (col[x] == -1)
315                         tension(d, e_delta(mat[slackv[x]][x]));
316                     else if (col[x] == 0)
317                         tension(d, e_delta(mat[slackv[x]][x]) / 2);
318                 }

```

```

319
320     for (int v = 1; v <= n; v++)
321     {
322         if (col[bel[v]] == 0)
323             lab[v] -= d;
324         else if (col[bel[v]] == 1)
325             lab[v] += d;
326     }
327     for (int b = n + 1; b <= n_x; b++)
328         if (bel[b] == b)
329         {
330             if (col[bel[b]] == 0)
331                 lab[b] += d * 2;
332             else if (col[bel[b]] == 1)
333                 lab[b] -= d * 2;
334         }
335
336     q_n = 0;
337     for (int v = 1; v <= n; v++)
338         if (lab[v] == 0) // all unmatched vertices' labels are zero! cheers!
339             return false;
340     for (int x = 1; x <= n_x; x++)
341         if (bel[x] == x && slackv[x] && bel[slackv[x]] != x && e_delta(mat[
342             slackv[x]][x]) == 0)
343         {
344             if (on_found_edge(mat[slackv[x]][x]))
345                 return true;
346         }
347     for (int b = n + 1; b <= n_x; b++)
348         if (bel[b] == b && col[b] == 1 && lab[b] == 0)
349             expand_blossom1(b);
350     return false;
351 }
352
353 void calc_max_weight_match()
354 {
355     for (int v = 1; v <= n; v++)
356         mate[v] = 0;
357
358     n_x = n;
359     n_matches = 0;
360     tot_weight = 0;
361
362     bel[0] = 0;
363     for (int v = 1; v <= n; v++)
364         bel[v] = v, bloch[v].clear();
365     for (int v = 1; v <= n; v++)
366         for (int u = 1; u <= n; u++)
367             blofrom[v][u] = v == u ? v : 0;
368
369     int w_max = 0;
370     for (int v = 1; v <= n; v++)
371         for (int u = 1; u <= n; u++)
372             relax(w_max, mat[v][u].w);
373     for (int v = 1; v <= n; v++)

```

```

374     lab[v] = w_max;
375
376     while (match())
377         n_matches++;
378
379     for (int v = 1; v <= n; v++)
380         if (mate[v] && mate[v] < v)
381             tot_weight += mat[v][mate[v]].w;
382 }
383
384 int main()
385 {
386     n = getint(), m = getint();
387
388     for (int v = 1; v <= n; v++)
389         for (int u = 1; u <= n; u++)
390             mat[v][u] = edge(v, u, 0);
391
392     for (int i = 0; i < m; i++)
393     {
394         int v = getint(), u = getint(), w = getint();
395         mat[v][u].w = mat[u][v].w = w;
396     }
397
398     calc_max_weight_match();
399
400     printf("%lld\n", tot_weight);
401     for (int v = 1; v <= n; v++)
402         printf("%d ", mate[v]);
403     printf("\n");
404
405     return 0;
406 }

```

7.5 KM

```

1 #define MAXN 100
2 #define INF INT_MAX
3 int g[MAXN][MAXN], lx[MAXN], ly[MAXN], slack_y[MAXN];
4 int px[MAXN], py[MAXN], match_y[MAXN], par[MAXN];
5 int n;
6 void adjust(int y) { //把增廣路上所有邊反轉
7     match_y[y] = py[y];
8     if (px[match_y[y]] != -2)
9         adjust(px[match_y[y]]);
10 }
11 bool dfs(int x) { //DFS找增廣路
12     for (int y = 0; y < n; ++y) {
13         if (py[y] != -1) continue;
14         int t = lx[x] + ly[y] - g[x][y];
15         if (t == 0) {
16             py[y] = x;
17             if (match_y[y] == -1) {
18                 adjust(y);

```

```

19         return 1;
20     }
21     if (px[match_y[y]] != -1) continue;
22     px[match_y[y]] = y;
23     if (dfs(match_y[y])) return 1;
24 } else if (slack_y[y] > t) {
25     slack_y[y] = t;
26     par[y] = x;
27 }
28 }
29 return 0;
30 }
31 inline int km() {
32     memset(ly, 0, sizeof(int) * n);
33     memset(match_y, -1, sizeof(int) * n);
34     for (int x = 0; x < n; ++x) {
35         lx[x] = -INF;
36         for (int y = 0; y < n; ++y) {
37             lx[x] = max(lx[x], g[x][y]);
38         }
39     }
40     for (int x = 0; x < n; ++x) {
41         for (int y = 0; y < n; ++y) slack_y[y] = INF;
42         memset(px, -1, sizeof(int) * n);
43         memset(py, -1, sizeof(int) * n);
44         px[x] = -2;
45         if (dfs(x)) continue;
46         bool flag = 1;
47         while (flag) {
48             int cut = INF;
49             for (int y = 0; y < n; ++y)
50                 if (py[y] == -1 && cut > slack_y[y]) cut = slack_y[y];
51             for (int j = 0; j < n; ++j) {
52                 if (px[j] != -1) lx[j] -= cut;
53                 if (py[j] != -1) ly[j] += cut;
54                 else slack_y[j] -= cut;
55             }
56             for (int y = 0; y < n; ++y) {
57                 if (py[y] == -1 && slack_y[y] == 0) {
58                     py[y] = par[y];
59                     if (match_y[y] == -1) {
60                         adjust(y);
61                         flag = 0;
62                         break;
63                     }
64                     px[match_y[y]] = y;
65                     if (dfs(match_y[y])) {
66                         flag = 0;
67                         break;
68                     }
69                 }
70             }
71         }
72     }
73     int ans = 0;
74     for (int y = 0; y < n; ++y) if (g[match_y[y]][y] != -INF) ans += g[match_y[y]][y];

```



```

75     return ans;
76 }

```

7.6 Min Cost Flow

```

1 #define maxnode (1000+10)
2 #define maxedge (40000+10)
3 #define INF 1023456789
4 #include <bits/stdc++.h>
5 using namespace std;
6 int node, src, dest, nedge;
7 int head[maxnode], point[maxedge], nxt[maxedge], flow[maxedge], capa[maxedge], wt[maxedge];
8 int dist[maxnode], in[maxnode], from[maxnode], mf[maxnode];
9 //set number of node, source, and destination (one base)
10 void init(int _node, int _src, int _dest) {
11     node = _node;
12     src = _src;
13     dest = _dest;
14     nedge = 0;
15     memset(point, -1, sizeof(point));
16     for (int i = 1; i <= node; i++) head[i] = -1;
17     nedge = 0;
18 }
19 void add_edge(int u, int v, int c1, int w) {
20     point[nedge] = v, capa[nedge] = c1, flow[nedge] = 0, nxt[nedge] = head[u], wt[nedge]=w, head[u] = (nedge++);
21     point[nedge] = u, capa[nedge] = 0, flow[nedge] = 0, nxt[nedge] = head[v], wt[nedge]=-w, head[v] = (nedge++);
22 }
23 int sp(int &left) {
24     for(int i=1;i<=node;i++) dist[i]=INF;
25     queue<int> que;
26     que.push(src);
27     in[src]=1;
28     mf[src]=left;
29     dist[src]=0;
30     while(!que.empty()) {
31         int u=que.front();
32         que.pop();
33         in[u]=0;
34         if(dist[u]>=dist[dest]) continue;
35         for(int v=head[u];v!=-1;v=nxt[v]){
36             if(flow[v]==capa[v]) continue;
37             if(dist[u]+wt[v]<dist[point[v]]){
38                 dist[point[v]]=dist[u]+wt[v];
39                 from[point[v]]=v;
40                 mf[point[v]]=min(mf[u],capa[v]-flow[v]);
41                 if(!in[point[v]]){
42                     in[point[v]]=1;
43                     que.push(point[v]);
44                 }
45             }
46         }
47     }
48     left-=mf[dest];
49     if(dist[dest]<INF){
50         for(int u=dest;u!=src;u=point[from[u]^1]){
51             flow[from[u]]+=mf[dest];
52             flow[from[u]^1]-=mf[dest];
53         }
54     }
55     return dist[dest];
56 }
57 int min_cost_flow() {
58     int res=0,tmp,maxflow=2;
59     while(maxflow&&(tmp=sp(maxflow))<INF) res+=tmp;
60     return res;
61 }
62 int main() {
63     int n,m,x,y,z;
64     while(scanf("%d%d",&n,&m)==2) {
65         init(n,1,n);
66         for(int i=0;i<m;i++){
67             scanf("%d%d%d",&x,&y,&z);
68             add_edge(x,y,1,z);
69             add_edge(y,x,1,z); //undirected
70         }
71         printf("%d\n",min_cost_flow());
72     }
73     return 0;
74 }

```

```

47 }
48 left-=mf[dest];
49 if(dist[dest]<INF){
50     for(int u=dest;u!=src;u=point[from[u]^1]){
51         flow[from[u]]+=mf[dest];
52         flow[from[u]^1]-=mf[dest];
53     }
54 }
55 return dist[dest];
56 }
57 int min_cost_flow() {
58     int res=0,tmp,maxflow=2;
59     while(maxflow&&(tmp=sp(maxflow))<INF) res+=tmp;
60     return res;
61 }
62 int main() {
63     int n,m,x,y,z;
64     while(scanf("%d%d",&n,&m)==2) {
65         init(n,1,n);
66         for(int i=0;i<m;i++){
67             scanf("%d%d%d",&x,&y,&z);
68             add_edge(x,y,1,z);
69             add_edge(y,x,1,z); //undirected
70         }
71         printf("%d\n",min_cost_flow());
72     }
73     return 0;
74 }

```

7.7 Stable Marriage

```

1 #define F(n) Fi(i, n)
2 #define Fi(i, n) Fl(i, 0, n)
3 #define Fl(i, l, n) for(int i = l ; i < n ; ++i)
4 #include <bits/stdc++.h>
5 using namespace std;
6 int D, quota[205], weight[205][5];
7 int S, scoretodep[12005][205], score[5];
8 int P, prefer[12005][85], iter[12005];
9 int ans[12005];
10 typedef pair<int, int> PII;
11 map<int, int> samescore[205];
12 typedef priority_queue<PII, vector<PII>, greater<PII>> QQQ;
13 QQQ pri[205];
14 void check(int d) {
15     PII t = pri[d].top();
16     int v;
17     if (pri[d].size() - samescore[d][t.first] + 1 <= quota[d]) return;
18     while (pri[d].top().first == t.first) {
19         v = pri[d].top().second;
20         ans[v] = -1;
21         --samescore[d][t.first];
22         pri[d].pop();
23     }

```

```

24 }
25 void push(int s, int d) {
26     if (pri[d].size() < quota[d]) {
27         pri[d].push(PII(scoretodep[s][d], s));
28         ans[s] = d;
29         ++samescore[s][scoretodep[s][d]];
30     } else if (scoretodep[s][d] >= pri[d].top().first) {
31         pri[d].push(PII(scoretodep[s][d], s));
32         ans[s] = d;
33         ++samescore[s][scoretodep[s][d]];
34         check(d);
35     }
36 }
37 void f() {
38     int over;
39     while (true) {
40         over = 1;
41         Fi (q, S) {
42             if (ans[q] != -1 || iter[q] >= P) continue;
43             push(q, prefer[q][iter[q]++]);
44             over = 0;
45         }
46         if (over) break;
47     }
48 }
49 main() {
50     ios::sync_with_stdio(false);
51     cin.tie(NULL);
52     int sadmit, stof, dexceed, dfew;
53     while (cin >> D, D) { // Beware of the input format or judge may troll us.
54         sadmit = stof = dexceed = dfew = 0;
55         memset(iter, 0, sizeof(iter));
56         memset(ans, 0, sizeof(ans));
57         Fi (q, 205) {
58             pri[q] = QQQ();
59             samescore[q].clear();
60         }
61         cin >> S >> P;
62         Fi (q, D) {
63             cin >> quota[q];
64             Fi (w, 5) cin >> weight[q][w];
65         }
66         Fi (q, S) {
67             Fi (w, 5) cin >> score[w];
68             Fi (w, D) {
69                 scoretodep[q][w] = 0;
70                 F (5) scoretodep[q][w] += weight[w][i] * score[i];
71             }
72         }
73         Fi (q, S) Fi (w, P) {
74             cin >> prefer[q][w];
75             --prefer[q][w];
76         }
77         f();
78         Fi (q, D) sadmit += pri[q].size();
79         Fi (q, S) if (ans[q] == prefer[q][0]) ++stof;

```

```

80     Fi (q, D) if (pri[q].size() > quota[q]) ++dexceed;
81     Fi (q, D) if (pri[q].size() < quota[q]) ++dfew;
82     cout << sadmit << ' ' << stof << ' ' << dexceed << ' ' << dfew << '\n';
83 }
84 }

```

8 Mathematics

8.1 Extended GCD

```

1 long long extgcd(long long a, long long b, long long &x, long long &y) {
2     long long d=a;
3     if (b!=0) {
4         d=extgcd(b, a%b, y, x);
5         y-= (a/b)*x;
6     }
7     else x=1, y=0;
8     return d;
9 }
10 int main() {
11     int T;
12     long long a, b, m, GCD, x, y;
13     while (~scanf("%d", &T))
14         while (T--) {
15             scanf("%lld%lld%lld", &m, &a, &b);
16             GCD=extgcd(a, m, x, y);
17             if (GCD!=1) printf("No inverse, gcd(a,m)=%lld\n", GCD);
18             else {
19                 b=((-b*x)%m+m)%m;
20                 printf("%lld %lld\n", (x%m+m)%m, b);
21             }
22         }
23 }

```

8.2 Lucas's Theorem

```

1 bigM = int(1e9+7)
2 fac = [1]*10001
3 for i in range(1, 10001):
4     fac[i] = fac[i-1]*i
5 ifac = [pow(fac[i], bigM-2, bigM) for i in range(10001)]
6 def f(a, b, M):
7     if b == 0 or b == a:
8         return 1
9     elif a < b:
10         return 0
11     elif a < M:
12         return fac[a]*ifac[b]*ifac[a-b]%bigM
13     else:

```

```

14     return f(a//M, b//M, M) * f(a%M, b%M, M) % bigM
15 t = int(input())
16 for cases in range(t):
17     a, b, M = [int(x) for x in input().split()]
18     print(f(a, b, M))

```

8.3 Miller-Rabin

```

1 inline long long mod_mul(long long a, long long b, long long m) {
2     a%=m, b%=m;
3     long long y=(long long)((double)a*b/m+0.5); /* fast for m < 2^58 */
4     long long r=(a*b-y*m)%m;
5     return r<0?r+m:r;
6 }
7 template<typename T>
8 inline T pow(T a, T b, T mod) { // a^b%mod
9     T ans=1;
10    for(; b; a=mod_mul(a, a, mod), b>>=1)
11        if(b&1) ans=mod_mul(ans, a, mod);
12    return ans;
13 }
14 int sprp[3]={2, 7, 61}; // int範圍可解
15 int llsprp[7]={2, 325, 9375, 28178, 450775, 9780504, 1795265022}; // 至少unsigned
16                                     long long範圍
17 template<typename T>
18 inline bool isprime(T n, int *sprp, int num) {
19     if(n==2) return 1;
20     if(n<2 || n%2==0) return 0;
21     int t=0;
22     T u=n-1;
23     for(; u%2==0; ++t) u>>=1;
24     for(int i=0; i<num; ++i) {
25         T a=sprp[i]%n;
26         if(a==0 || a==1 || a==n-1) continue;
27         T x=pow(a, u, n);
28         if(x==1 || x==n-1) continue;
29         for(int j=0; j<t; ++j) {
30             x=mod_mul(x, x, n);
31             if(x==1) return 0;
32             if(x==n-1) break;
33         }
34         if(x==n-1) continue;
35         return 0;
36     }
37     return 1;
38 }

```

9 String

9.1 AC Automaton

```

1 #ifndef SUNMOON_AHO_CORASICK_AUTOMATON
2 #define SUNMOON_AHO_CORASICK_AUTOMATON
3 #include<queue>
4 #include<vector>
5 template<char L='a', char R='z'>
6 class ac_automaton {
7     private:
8         struct joe {
9             int next[R-L+1], fail, efl, ed, cnt_dp, vis;
10            joe(): ed(0), cnt_dp(0), vis(0) {}
11            for(int i=0; i<=R-L; ++i) next[i]=0;
12        };
13    public:
14        std::vector<joe> S;
15        std::vector<int> q;
16        int qs, qe, vt;
17        ac_automaton(): S(1), qs(0), qe(0), vt(0) {}
18        inline void clear() {
19            q.clear();
20            S.resize(1);
21            for(int i=0; i<=R-L; ++i) S[0].next[i]=0;
22            S[0].cnt_dp=S[0].vis=qs=qe=vt=0;
23        }
24        inline void insert(const char *s) {
25            int o=0;
26            for(int i=0; id=s[i]; ++i) {
27                id=s[i]-L;
28                if(!S[o].next[id]) {
29                    S.push_back(joe());
30                    S[o].next[id]=S.size()-1;
31                }
32                o=S[o].next[id];
33            }
34            ++S[o].ed;
35        }
36        inline void build_fail() {
37            S[0].fail=S[0].efl=-1;
38            q.clear();
39            q.push_back(0);
40            ++qe;
41            while(qs!=qe) {
42                int pa=q[qs++], id, t;
43                for(int i=0; i<=R-L; ++i) {
44                    t=S[pa].next[i];
45                    if(!t) continue;
46                    id=S[pa].fail;
47                    while(~id&&!S[id].next[i]) id=S[id].fail;
48                    S[t].fail=~id?S[id].next[i]:0;
49                    S[t].efl=S[S[t].fail].ed?S[t].fail:S[S[t].fail].efl;
50                    q.push_back(t);
51                }
52            }
53        }
54    };
55 }

```

```

52     ++qe;
53 }
54 }
55 }
56 /*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)*/
57 inline int match_0(const char *s){
58     int ans=0,id,p=0,i;
59     for(i=0;s[i];++i){
60         id=s[i]-L;
61         while(!S[p].next[id]&&p)p=S[p].fail;
62         if(!S[p].next[id])continue;
63         p=S[p].next[id];
64         ++S[p].cnt_dp; /*匹配成功則它所有後綴都可以被匹配(DP計算)*/
65     }
66     for(i=qe-1;i>=0;--i){
67         ans+=S[q[i]].cnt_dp*S[q[i]].ed;
68         if(~S[q[i]].fail)S[S[q[i]].fail].cnt_dp+=S[q[i]].cnt_dp;
69     }
70     return ans;
71 }
72 /*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)*/
73 inline int match_1(const char *s) const{
74     int ans=0,id,p=0,t;
75     for(int i=0;s[i];++i){
76         id=s[i]-L;
77         while(!S[p].next[id]&&p)p=S[p].fail;
78         if(!S[p].next[id])continue;
79         p=S[p].next[id];
80         if(S[p].ed)ans+=S[p].ed;
81         for(t=S[p].efl;~t;t=S[t].efl){
82             ans+=S[t].ed; /*因為都走efl邊所以保證匹配成功*/
83         }
84     }
85     return ans;
86 }
87 /*枚舉(s的子字串sA)的所有相異字串各恰一次並傳回次數O(N*M^(1/3))*/
88 inline int match_2(const char *s){
89     int ans=0,id,p=0,t;
90     ++vt;
91     /*把戳記vt+=1，只要vt沒溢位，所有S[p].vis==vt就會變成false
92     這種利用vt的方法可以O(1)歸零vis陣列*/
93     for(int i=0;s[i];++i){
94         id=s[i]-L;
95         while(!S[p].next[id]&&p)p=S[p].fail;
96         if(!S[p].next[id])continue;
97         p=S[p].next[id];
98         if(S[p].ed&&S[p].vis!=vt){
99             S[p].vis=vt;
100             ans+=S[p].ed;
101         }
102         for(t=S[p].efl;~t&&S[t].vis!=vt;t=S[t].efl){
103             S[t].vis=vt;
104             ans+=S[t].ed; /*因為都走efl邊所以保證匹配成功*/
105         }
106     }
107     return ans;

```

```

108 }
109 /*把AC自動機變成真的自動機*/
110 inline void evolution(){
111     for(qs=1;qs!=qe;){
112         int p=q[qs++];
113         for(int i=0;i<=R-L;++i)
114             if(S[p].next[i]==0)S[p].next[i]=S[S[p].fail].next[i];
115     }
116 }
117 };
118 #endif

```

9.2 BWT

```

1 // BWT
2 const int N = 8; // 字串長度
3 int s[N+N+1] = "suffixes"; // 字串，後面預留一倍空間。
4 int sa[N]; // 後綴陣列
5 int pivot;
6
7 int cmp(const void* i, const void* j)
8 {
9     return strcmp(s+(int*)i, s+(int*)j, N);
10 }
11
12 // 此處便宜行事，採用 O(N^2 log N) 的後綴陣列演算法。
13 void BWT()
14 {
15     strncpy(s + N, s, N);
16     for (int i=0; i<N; ++i) sa[i] = i;
17     qsort(sa, N, sizeof(int), cmp);
18     // 當輸入字串的所有字元都相同，必須當作特例處理。
19     // 或者改用stable sort。
20
21     for (int i=0; i<N; ++i)
22         cout << s[(sa[i] + N-1) % N];
23
24     for (int i=0; i<N; ++i)
25         if (sa[i] == 0)
26         {
27             pivot = i;
28             break;
29         }
30 }
31
32 // Inverse BWT
33 const int N = 8; // 字串長度
34 char t[N+1] = "xuffessi"; // 字串
35 int pivot;
36 int next[N];
37
38 void IBWT()
39 {
40     vector<int> index[256];

```

```

41     for (int i=0; i<N; ++i)
42         index[t[i]].push_back(i);
43
44     for (int i=0, n=0; i<256; ++i)
45         for (int j=0; j<index[i].size(); ++j)
46             next[n++] = index[i][j];
47
48     int p = pivot;
49     for (int i=0; i<N; ++i)
50         cout << t[p = next[p]];
51 }

```

9.3 Suffix Array

```

1 //should initialize s and n first
2 #define N 301000
3 using namespace std;
4 char s[N]; //string=s,suffix array=sar,longest common prefix=lcp
5 int rk[2][N],id[2][N];
6 int n,p;
7 int cnt[N];
8 int len[N],od[N],sar[N];
9 inline int sr(int i,int t){ //rank of shifted position
10     return i+t<n?rk[p][i+t]:-1;
11 }
12 inline bool check_same(int i,int j,int t){
13     return rk[p][i]==rk[p][j]&&sr(i,t)==sr(j,t);
14 }
15 bool cmp(int i,int j){
16     return s[i]<s[j];
17 }
18 void sa(){ //length of array s
19     int i,t,now,pre;
20     memset(cnt,0,sizeof(cnt));
21     for(i=0;i<n;i++){
22         id[p][i]=i;
23         rk[p][i]=s[i];
24         cnt[s[i]]++;
25     }
26     for(i=1;i<128;i++) cnt[i]+=cnt[i-1];
27     sort(id[p],id[p]+n,cmp);
28     for(t=1;t<n;t<=1){
29         //least significant bit is already sorted
30         for(i=n-1;i>=0;i--){
31             now=id[p][i]-t;
32             if(now>=0) id[p^1][--cnt[rk[p][now]]]=now;
33         }
34         for(i=n-t;i<n;i++){
35             id[p^1][--cnt[rk[p][i]]]=i;
36         }
37         memset(cnt,0,sizeof(cnt));
38         now=id[p^1][0];
39         rk[p^1][now]=0;
40         cnt[0]++;

```

```

41     for(i=1;i<n;i++){
42         pre=now;
43         now=id[p^1][i];
44         if(check_same(pre,now,t)){
45             rk[p^1][now]=rk[p^1][pre];
46         }
47         else{
48             rk[p^1][now]=rk[p^1][pre]+1;
49         }
50         cnt[rk[p^1][now]]++;
51     }
52     p^=1;
53     if(rk[p][now]==n-1) break;
54     for(i=1;i<n;i++) cnt[i]+=cnt[i-1];
55 }
56 memcpy(sar,id[p],sizeof(sar));
57 }
58 void lcp(){
59     int i,l,pre;
60     for(i=0;i<n;i++) od[sar[i]]=i;
61     for(i=0;i<n;i++){
62         if(i) l=len[od[i-1]]?len[od[i-1]]-1:0;
63         else l=0;
64         if(od[i]){
65             pre=sar[od[i]-1];
66             while(pre+l<n&&i+l<n&&s[pre+l]==s[i+l]) l++;
67             len[od[i]]=l;
68         }
69         else len[0]=0;
70     }
71 }

```

9.4 Suffix Automaton

```

1 #include<bits/stdc++.h>
2 #define C 96
3 #define N 200100
4 using namespace std;
5 struct SAM{
6     struct node{
7         node *nxt[C],*pre;
8         int len;
9         vector<int> pos;
10     };
11     node mem[N*2],*root,*ed;
12     int top;
13     SAM(){
14         top = 0;
15         root = new_node(0);
16         ed = root;
17     }
18     node *new_node(int l){
19         for(int i=0;i<C;i++) mem[top].nxt[i]=NULL;
20         mem[top].pre=NULL;

```

```

21     mem[top].len=1;
22     mem[top].pos.clear();
23     return mem+(top++);
24 }
25 node *split_node(int l,node *p){
26     for(int i=0;i<C;i++) mem[top].nxt[i]=p->nxt[i];
27     mem[top].pre = p->pre;
28     mem[top].len = 1;
29     mem[top].pos.assign()
30     p->pre = mem+top;
31     return mem+(top++);
32 }
33 void push(char c){
34     node *nw = new_node(ed->len+1),*ptr=ed->pre;
35     ed->nxt[c] = nw;
36     nw->pos.push_back(ed->len);
37     for(;ptr;ptr=ptr->pre){
38         if(ptr->nxt[c]){
39             if(ptr->nxt[c]->len==ptr->len+1){
40                 nw->pre = ptr->nxt[c];
41             }
42             else{
43                 node *tmp=ptr->nxt[c];
44                 nw->pre = split_node(ptr->len+1,tmp);
45                 while(ptr && ptr->nxt[c]==tmp){
46                     ptr->nxt[c] = nw->pre;
47                     ptr = ptr->pre;
48                 }
49             }
50             break;
51         }
52         else{
53             ptr->nxt[c] = nw;
54         }
55     }
56     if(!nw->pre) nw->pre = root;
57     ed = ed->nxt[c];
58 }
59 void init(){
60     while(top){
61         mem[--top].pos.clear();
62     }
63     root = new_node(0);
64     ed = root;
65 }
66 void push(char *s){
67     for(int i=0;s[i];i++) push(s[i]-32);
68 }
69 long long count(){
70     long long ans=0;
71     for(int i=1;i<top;i++){
72         ans+=mem[i].len-mem[i].pre->len;
73     }
74     return ans;
75 }
76 }sam;

```

```

77 char S[N];
78 int main(){
79     int T;
80     scanf("%d",&T);
81     while(T--){
82         scanf("%s",S);
83         sam.build(S);
84         printf("%lld\n",sam.count());
85     }
86     return 0;
87 }

```

9.5 Z Algorithm

```

1 void Zalg(char *s, int *z, int n) {
2     z[0]=n;
3     for(int L=0, R=0, i=1; i<n; i++) {
4         if(i<=R && z[i-L]<=R-i) z[i]=z[i-L];
5         else {
6             L=i;
7             if(i>R) R=i;
8             while(R<n && s[R-L]==s[R]) R++;
9             z[i]=(R--)-L;
10        }
11    }
12 }

```

10 Struct

10.1 Treap

```

1 struct Treap{
2     Treap *l,*r;
3     int pri,sz,val,add;
4     Treap(int _val):pri(rand()),sz(1),val(_val),add(0),l(NULL),r(NULL){}
5 };
6
7 int size(Treap *t){
8     return t?t->sz:0;
9 }
10 void pull(Treap *t){
11     t->sz=size(t->l)+size(t->r)+1;
12 }
13 void push(Treap *t){
14     t->val+=t->add;
15     if(t->l) t->l->add+=t->add;
16     if(t->r) t->r->add+=t->add;
17     t->add=0;
18 }

```

```

19 Treap* merge(Treap *a,Treap *b){
20     if(!a||!b) return a?a:b;
21     if(a->pri > b->pri){
22         push(a);
23         a->r = merge(a->r,b);
24         pull(a);
25         return a;
26     }
27     else{
28         push(b);
29         b->l = merge(a,b->l);
30         pull(b);
31         return b;
32     }
33 }
34 void split(Treap *t,int k,Treap *&a,Treap *&b){
35     if(!t) a=b=NULL;
36     else{
37         push(t);
38         if(size(t->l) < k){
39             a=t;
40             split(t->r,k-size(t->l)-1,a->r,b);
41             pull(a);
42         }
43         else{
44             b=t;
45             split(t->l,k,a,b->l);
46             pull(b);
47         }
48     }
49 }

```

```

16         sum=val*(r-l);
17         lc->val=rc->val=val;
18         val=-1;
19     }
20 }
21 void pull(){
22     sum=lc->qsum()+rc->qsum();
23 }
24 };
25 node* tr[N];
26 node* build(int l,int r){
27     node *now=new node(l,r);
28     if(r-l>1){
29         now->lc=build(l,(l+r)/2);
30         now->rc=build((l+r)/2,r);
31     }
32     return now;
33 }
34 //partial
35 int qry(node* now,int l,int r){
36     if(l>=r) return 0;
37     if(l==now->l&&r==now->r){
38         return now->qsum();
39     }
40     int m=(now->l+now->r)/2;
41     now->push();
42     if(l>=m){
43         return qry(now->rc,l,r);
44     }
45     else if(r<=m){
46         return qry(now->lc,l,r);
47     }
48     else return qry(now->lc,l,m)+qry(now->rc,m,r);
49 }
50 void set0(node *now,int l,int r){
51     if(l>=r) return;
52     if(l==now->l&&r==now->r){
53         now->val=0;
54         return;
55     }
56     int m=(now->l+now->r)/2;
57     now->push();
58     if(l>=m){
59         set0(now->rc,l,r);
60     }
61     else if(r<=m){
62         set0(now->lc,l,r);
63     }
64     else{
65         set0(now->lc,l,m);
66         set0(now->rc,m,r);
67     }
68     now->pull();
69 }
70 vector<int> g[N];
71 void DFS(int u,int p,int d){

```

11 Tree

11.1 Heavy Light Decomposition

```

1 //with set value && query sum, 1-based with n points
2 //remove vis in DFS, add it back if something weird happen(I don't think it
   s required)
3 using namespace std;
4 int sz[N],top[N],up[N],dep[N];
5 int lightval[N]; //value on light edge
6 struct node{
7     node(){}
8     node(int _l,int _r):val(1),l(_l),r(_r),lc(NULL),rc(NULL){}
9     int l,r;
10    node *lc,*rc;
11    int sum;
12    int val;
13    int qsum(){return val>=0?val*(r-l):sum;}
14    void push(){
15        if(val>=0){

```

```
72  dep[u]=d;
73  sz[u]=1;
74  for(int i=0;i<g[u].size();i++){
75      int v=g[u][i];
76      if(v==p) continue;
77      DFS(v,u,d+1);
78      sz[u]+=sz[v];
79  }
80  }
81  void decomp(int u,int p,bool istop){
82      bool ed=true;
83      if(istop) top[u]=u,up[u]=p,lightval[u]=1;
84      else top[u]=top[p],up[u]=up[p];
85      for(int i=0;i<g[u].size();i++){
86          int v=g[u][i];
87          if(v==p) continue;
88          if(sz[v]>=sz[u]-sz[v]){
89              decomp(v,u,false);
90              ed=false;
91          }
92          else decomp(v,u,true);
93      }
94      if(ed){
95          tr[top[u]]=build(dep[top[u]],dep[u]);
96      }
97  }
98  //global
99  int qry(int u,int v){
100     int res=0;
101     while(top[u]!=top[v]){
102         if(dep[top[u]]>dep[top[v]]) swap(u,v);
103         res+=qry(tr[top[v]],dep[top[v]],dep[v]);
104         res+=lightval[top[v]];
105         v=up[top[v]];
106     }
107     if(dep[u]>dep[v]) swap(u,v);
108     res+=qry(tr[top[v]],dep[u],dep[v]);
109     return res;
110 }
111 void set0(int u,int v){
112     while(top[u]!=top[v]){
113         if(dep[top[u]]>dep[top[v]]) swap(u,v);
114         set0(tr[top[v]],dep[top[v]],dep[v]);
115         lightval[top[v]]=0;
116         v=up[top[v]];
117     }
118     if(dep[u]>dep[v]) swap(u,v);
119     set0(tr[top[v]],dep[u],dep[v]);
120 }
121 int main(){
122     DFS(1,0,0);
123     decomp(1,0,true);
124 }
```