

Code Library



Himemiya Nanao @ Perfect Freeze

July 29, 2013

Contents

1 data structure	1	5 graph	25
1.1 atlantis	1	5.1 2-sat	25
1.2 Binary Indexed tree	2	5.2 Articulation	25
1.3 COT	2	5.3 Augmenting Path Algorithm for Maximum Cardinality Bi- partite Matching	25
1.4 GSS7	2	5.4 Biconnected Component - Edge	25
1.5 Leftist tree	3	5.5 Biconnected Component	26
1.6 Network	4	5.6 Blossom algorithm	26
1.7 OTOCI	5	5.7 Bridge	27
1.8 picture	6	5.8 chu-liu algorithm	27
1.9 Size Blanced Tree	6	5.9 Covering problems	28
1.10 Sparse Table - rectangle	8	5.10 Difference constraints	28
1.11 Sparse Table - square	8	5.11 Dinitz's algorithm	28
1.12 Sparse Table	8	5.12 Feasible flow problem	28
1.13 Trea	8	5.13 Flow network	29
2 dynamic programming	9	5.14 Hamiltonian circuit	30
2.1 knapsack problem	9	5.15 Hopcroft-Karp algorithm	30
2.2 LCIS	9	5.16 Improved Shortest Augment- ing Path Algorithm	30
2.3 LCS	9	5.17 k Shortest Path	31
3 geometry	9	5.18 Kariv-Hakimi Algorithm	32
3.1 3D	9	5.19 Kuhn-Munkres algorithm	32
3.2 3DCH	10	5.20 LCA - DA	33
3.3 circle ploy's intersection area	11	5.21 LCA - tarjan - minmax	33
3.4 circle's area	11	5.22 Minimum Ratio Spanning Tree	33
3.5 circle	12	5.23 Minimum-cost flow problem	34
3.6 closest point pair	13	5.24 Stable Marriage	34
3.7 ellipse	14	5.25 Stoer-Wagner Algorithm	35
3.8 Graham's scan	14	5.26 Strongly Connected Component	35
3.9 half-plane intersection	15	5.27 ZKW's Minimum-cost flow	35
3.10 kdtree	15	5.28 ZKW's SAP	36
3.11 Manhattan minimum span- ning tree	16	6 math	36
3.12 others	17	6.1 cantor	36
3.13 Pick's theorem	17	6.2 Continued fraction	36
3.14 PointInPoly	17	6.3 Discrete logarithms - BSGS	37
3.15 rotating caliper	17	6.4 Divisor function	37
3.16 shit	18	6.5 Extended Euclidean Algorithm	37
3.17 sort - polar angle	18	6.6 Gaussian elimination	37
3.18 triangle	18	6.7 inverse element	38
4 geometry/tmp	19	6.8 Linear programming	38
4.1 circle	19	6.9 Lucas' theorem(2)	39
4.2 circles	20	6.10 Lucas' theorem	39
4.3 halfplane	20	6.11 Matrix	39
4.4 line	21	6.12 Miller-Rabin Algorithm	40
4.5 line3d	21	6.13 Multiset	40
4.6 plane	22	6.14 Pell's equation	40
4.7 point	22	6.15 Pollard's rho algorithm	40
4.8 point3d	23	6.16 Prime	41
4.9 polygon	23	6.17 Reduced Residue System	41
4.10 polygons	24	6.18 System of linear congruences	41
		7 others	42
		7.1 .vimrc	42
		7.2 bigint	42
		7.3 Binary Search	43
		7.4 java	43
		7.5 others	44

8	search	44
8.1	dlx	44
8.2	dlx - exact cover	44
8.3	dlx - repeat cover	45
8.4	fibonacci knapsack	45
9	string	46
9.1	Aho-Corasick Algorithm	46
9.2	Gusfield's Z Algorithm	46
9.3	Manacher's Algorithm	46
9.4	Morris-Pratt Algorithm	47
9.5	smallest representation	47
9.6	Suffix Array - DC3 Algorithm	47
9.7	Suffix Array - Prefix-doubling Algorithm	48

1 data structure

1.1 atlantis

```

1  #include<cstdio>
2  #include<algorithm>
3  #include<map>
4
5  #define MAXX 111
6  #define inf 333
7  #define MAX inf*5
8
9  int mid[MAX],cnt[MAX];
10 double len[MAX];
11
12 int n,i,cas;
13 double x1,x2,y1,y2;
14 double ans;
15 std::map<double,int>map;
16 std::map<double,int>::iterator it;
17 double rmap[inf];
18
19 void make(int id,int l,int r)
20 {
21     mid[id]=(l+r)>>1;
22     if(l!=r)
23     {
24         make(id<<1,l,mid[id]);
25         make(id<<1|1,mid[id]+1,r);
26     }
27 }
28
29 void update(int id,int ll,int rr,int l,int r,int val)
30 {
31     if(ll==l && rr==r)
32     {
33         cnt[id]+=val;
34         if(cnt[id])
35             len[id]=rmap[r]-rmap[l-1];
36         else
37             if(l!=r)
38                 len[id]=len[id<<1]+len[id<<1|1];
39             else
40                 len[id]=0;
41         return;
42     }
43     if(mid[id]>=r)
44         update(id<<1,ll,mid[id],l,r,val);
45     else
46         if(mid[id]<l)
47             update(id<<1|1,mid[id]+1,rr,mid[id]+1,r,val);
48         else
49         {
50             update(id<<1,ll,mid[id],l,mid[id],val);
51             update(id<<1|1,mid[id]+1,rr,mid[id]+1,r,val);
52         }
53     if(!cnt[id])
54         len[id]=len[id<<1]+len[id<<1|1];
55 }
56
57 struct node
58 {
59     double l,r,h;
60     char f;
61     inline bool operator<(const node &a)const
62     {
63         return h<a.h;
64     }
65     inline void print()
66     {
67         printf("%lf %lf %lf %d\n",l,r,h,f);
68     }
69 }ln[inf];
70
71 int main()
72 {
73     make(1,1,inf);
74     while(scanf("%d",&n),n)
75     {
76         n<=1;
77         map.clear();
78         for(i=0;i<n;++i)
79         {
80             scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);
81             if(x1>x2)
82                 std::swap(x1,x2);
83             if(y1>y2)
84                 std::swap(y1,y2);
85             ln[i].l=x1;
86             ln[i].r=x2;
87             ln[i].h=y1;
88             ln[i].f=1;
89             ln[++i].l=x1;
90             ln[i].r=x2;
91             ln[i].h=y2;
92             ln[i].f=-1;
93             map[x1]=1;
94             map[x2]=1;
95         }
96         i=1;
97         for(it=map.begin();it!=map.end();++it,++i)
98         {
99             it->second=i;
100             rmap[i]=it->first;
101         }
102         std::sort(ln,ln+n);
103         ans=0;
104         update(1,1,inf,map[ln[0].l]+1,map[ln[0].r],ln[0].f);
105         for(i=1;i<n;++i)
106         {
107             ans+=len[i]*(ln[i].h-ln[i-1].h);
108             update(1,1,inf,map[ln[i].l]+1,map[ln[i].r],ln[i].f);
109         }
110         printf("Test case #%d\nTotal explored area: %.2lf\n\n",++cas,ans);
111     }

```

```

112     return 0;
113 }

```

1.2 Binary Indexed tree

```

1  int tree[MAXX];
2
3  inline int lowbit(const int &a)
4  {
5      return a&-a;
6  }
7
8  inline void update(int pos,const int &val)
9  {
10     while(pos<MAXX)
11     {
12         tree[pos]+=val;
13         pos+=lowbit(pos);
14     }
15 }
16
17 inline int read(int pos)
18 {
19     int re(0);
20     while(pos>0)
21     {
22         re+=tree[pos];
23         pos-=lowbit(pos);
24     }
25     return re;
26 }
27
28 int find_Kth(int k)
29 {
30     int now=0;
31     for(char i=20;i>=0;--i)
32     {
33         now|=(1<<i);
34         if(now>MAXX || tree[now]>=k)
35             now^=(1<<i);
36         else k-=tree[now];
37     }
38     return now+1;
39 }

```

1.3 COT

```

1  #include<cstdio>
2  #include<algorithm>
3
4  #define MAXX 100111
5  #define MAX (MAXX*23)
6  #define N 18
7
8  int sz[MAX],lson[MAX],rson[MAX],cnt;
9  int head[MAXX];
10 int pre[MAXX][N];
11 int map[MAXX],m;
12
13 int edge[MAXX],nxt[MAXX<<1],to[MAXX<<1];
14 int n,l,j,k,q,l,r,mid;
15 int num[MAXX],dg[MAXX];
16
17 int make(int l,int r)
18 {
19     if(l==r)
20         return ++cnt;
21     int id(++cnt),mid((l+r)>>1);
22     lson[id]=make(l,mid);
23     rson[id]=make(mid+1,r);
24     return id;
25 }
26
27 inline int update(int id,int pos)
28 {
29     int re(++cnt);
30     l=1;
31     r=m;
32     int nid(re);
33     sz[nid]=sz[id]+1;
34     while(l<r)
35     {
36         mid=(l+r)>>1;
37         if(pos<=mid)
38         {
39             lson[nid]=++cnt;
40             rson[nid]=rson[id];
41             nid=lson[nid];
42             id=lson[id];
43             r=mid;
44         }
45         else
46         {
47             lson[nid]=lson[id];
48             rson[nid]=++cnt;
49             nid=rson[nid];
50             id=rson[id];
51             l=mid+1;
52         }
53         sz[nid]=sz[id]+1;
54     }
55     return re;
56 }
57
58 void rr(int now,int fa)
59 {
60     dg[now]=dg[fa]+1;
61     head[now]=update(head[fa],num[now]);

```

```

62     for(int i=edge[now];i;i=nxt[i])
63         if(to[i]!=fa)
64         {
65             j=1;
66             for(pre[to[i]][0]=now;j<N;++j)
67                 pre[to[i]][j]=pre[pre[to[i]][j-1]][j-1];
68             rr(to[i],now);
69         }
70 }
71
72 inline int query(int a,int b,int n,int k)
73 {
74     static int tmp,t;
75     l=1;
76     r=m;
77     a=head[a];
78     b=head[b];
79     t=num[n];
80     n=head[n];
81     while(l<r)
82     {
83         mid=(l+r)>>1;
84         tmp=sz[lson[a]]+sz[lson[b]]-2*sz[lson[n]]+(l<=t && t<=mid);
85         if(tmp>=k)
86         {
87             a=lson[a];
88             b=lson[b];
89             n=lson[n];
90             r=mid;
91         }
92         else
93         {
94             k-=tmp;
95             a=rson[a];
96             b=rson[b];
97             n=rson[n];
98             l=mid+1;
99         }
100     }
101     return l;
102 }
103
104 inline int lca(int a,int b)
105 {
106     static int i,j;
107     j=0;
108     if(dg[a]<dg[b])
109         std::swap(a,b);
110     for(i=dg[a]-dg[b];i>=1,++j)
111         if(i&1)
112             a=pre[a][j];
113     if(a==b)
114         return a;
115     for(i=N-1;i>=0;--i)
116         if(pre[a][i]!=pre[b][i])
117         {
118             a=pre[a][i];
119             b=pre[b][i];
120         }
121     return pre[a][0];
122 }
123
124 int main()
125 {
126     scanf("%d %d",&n,&q);
127     for(i=1;i<=n;++i)
128     {
129         scanf("%d",&num[i]);
130         map[i]=num[i];
131     }
132     std::sort(map+1,map+n+1);
133     m=std::unique(map+1,map+n+1)-map-1;
134     for(i=1;i<=n;++i)
135         num[i]=std::lower_bound(map+1,map+m+1,num[i])-map;
136     for(i=1;i<=n;++i)
137     {
138         scanf("%d %d",&j,&k);
139         nxt[++cnt]=edge[j];
140         edge[j]=cnt;
141         to[cnt]=k;
142     }
143     nxt[++cnt]=edge[k];
144     edge[k]=cnt;
145     to[cnt]=j;
146 }
147 cnt=0;
148 head[0]=make(1,m);
149 rr(1,0);
150 while(q--)
151 {
152     scanf("%d %d %d",&i,&j,&k);
153     printf("%d\n",map[query(i,j,lca(i,j),k)]);
154 }
155 return 0;
156 }

```

1.4 GSS7

```

1  #include<cstdio>
2  #include<algorithm>
3  #include<queue>
4
5  #define MAXX 100111
6  #define MAX (MAXX<<1)
7
8  struct node
9  {
10     bool set,rev;
11     node *pre,*nxt[2],*fa;
12     int lmax,max,rmax,sum,val,sz;
13     node();
14     node(int a);
15 }*tree[MAXX],*nil,*a,*b;
16

```

```

17 node::node()
18 {
19     rev=set=false;
20     fa=pre=nil;
21     nxt[0]=nxt[1]=nil;
22     sz=lmax=max=rmax=sum=val=0;
23 }
24
25 node::node(int a)
26 {
27     set=rev=false;
28     sum=val=a;
29     sz=1;
30     lmax=max=rmax=std::max(0,a);
31     fa=pre=nxt[0]=nxt[1]=nil;
32 }
33
34 inline void add(node &x,const node &l,const node &r)
35 {
36     x.max=std::max(l.rmax+r.lmax, std::max(l.lmax,r.max));
37     x.lmax=std::max(l.lmax,l.sum+r.lmax);
38     x.rmax=std::max(r.rmax,r.sum+l.rmax);
39     x.sum=l.sum+r.sum;
40 }
41
42 inline void up(node *id)
43 {
44     id->sz=id->nxt[0]->sz+id->nxt[1]->sz+1;
45     id->sum=id->val+id->nxt[0]->sum+id->nxt[1]->sum;
46     id->lmax=std::max(id->nxt[0]->lmax, id->nxt[0]->sum+id->val+id->
        nxt[1]->lmax);
47     id->rmax=std::max(id->nxt[1]->rmax, id->nxt[1]->sum+id->val+id->
        nxt[0]->rmax);
48     id->max=std::max(id->nxt[0]->rmax+id->val+id->nxt[1]->lmax, std
        ::max(id->nxt[0]->max, id->nxt[1]->max));
49 }
50
51 inline void set(node *id,int val)
52 {
53     if(id==nil)
54         return;
55     id->set=true;
56     id->val=val;
57     id->sum=val*id->sz;
58     id->max=id->lmax=id->rmax=std::max(0,id->sum);
59 }
60
61 inline void down(node *id)
62 {
63     if(id==nil)
64         return;
65     if(id->rev)
66     {
67         id->rev=false;
68         for(int i(0);i<2;++i)
69             if(id->nxt[i]!=nil)
70             {
71                 id->nxt[i]->rev=true;
72                 std::swap(id->nxt[i]->nxt[0], id->nxt[i]->nxt[1]);
73                 std::swap(id->nxt[i]->lmax, id->nxt[i]->rmax);
74             }
75     }
76     if(id->set)
77     {
78         for(int i(0);i<2;++i)
79             if(id->nxt[i]!=nil)
80                 set(id->nxt[i], id->val);
81         id->set=false;
82     }
83 }
84
85 inline void rot(node *id,int tp)
86 {
87     node *k(id->pre);
88     k->nxt[tp^1]=id->nxt[tp];
89     if(id->nxt[tp]!=nil)
90         id->nxt[tp]->pre=k;
91     if(k->pre!=nil)
92         k->pre->nxt[k=k->pre->nxt[1]]=id;
93     id->pre=k->pre;
94     id->nxt[tp]=k;
95     k->pre=id;
96     up(k);
97     up(id);
98 }
99
100 node *fresh(node* id)
101 {
102     node *re(id);
103     if(id->pre!=nil)
104         re=fresh(id->pre);
105     down(id);
106     return re;
107 }
108
109 inline void splay(node *id)
110 {
111     node *rt(fresh(id));
112     if(id!=rt)
113         for(std::swap(rt->fa, id->fa); id->pre!=nil; rot(id, id==id->pre
            ->nxt[0]));
114 }
115
116 inline void access(node *id)
117 {
118     for(node *to(nil); id!=nil; id=id->fa)
119     {
120         splay(id);
121         id->nxt[1]->pre=nil;
122         if(id->nxt[1]!=nil)
123             id->nxt[1]->fa=id;
124         id->nxt[1]=to;
125         if(to!=nil)
126             to->pre=id;
127         to->fa=nil;
128         up(to=id);
129     }
130 }
131
132 inline void lca(node *to,node *id)
133 {
134     access(to);
135     splay(id);
136     for(to=nil; id->fa!=nil; splay(id=id->fa))
137     {
138         id->nxt[1]->pre=nil;
139         if(id->nxt[1]!=nil)
140             id->nxt[1]->fa=id;
141         id->nxt[1]=to;
142         if(to!=nil)
143             to->pre=id;
144         to->fa=nil;
145         up(to=id);
146     }
147 }
148
149 int n,i,j,k;
150 int nxt[MAX],to[MAX],edge[MAXX],cnt;
151 std::queue<int>q;
152
153 inline void add(int a,int b)
154 {
155     nxt[++cnt]=edge[a];
156     edge[a]=cnt;
157     to[cnt]=b;
158 }
159
160 void rr(int now,int fa)
161 {
162     for(int i(edge[now]); i;i=nxt[i])
163         if(to[i]!=fa)
164         {
165             tree[to[i]]->fa=tree[now];
166             rr(to[i],now);
167         }
168 }
169
170 /*
171 void print(node *id)
172 {
173     if(id!=nil)
174     {
175         print(id->nxt[0]);
176         printf("%2d %2d %2d %2d %2d %2d %c %2d\n", id->val, id->sum, id
            ->sz, id->lmax, id->max, id->rmax, id->rev?'r':'n', id->
            pre->val);
177         print(id->nxt[1]);
178     }
179 }
180 */
181
182 int main()
183 {
184     nil=new node();
185     scanf("%d",&n);
186     for(i=1;i<=n;++i)
187     {
188         scanf("%d",&j);
189         tree[i]=new node(j);
190     }
191     for(i=1;i<=n;++i)
192     {
193         scanf("%d %d",&j,&k);
194         add(j,k);
195         add(k,j);
196     }
197     tree[0]=nil;
198     rr(1,0);
199     scanf("%d",&n);
200     while(n--)
201     {
202         scanf("%d %d %d",&k,&i,&j);
203         a=tree[i];
204         b=tree[j];
205         access(a);
206         splay(a);
207         a->rev=true;
208         std::swap(a->nxt[0], a->nxt[1]);
209         std::swap(a->lmax, a->rmax);
210         access(b);
211         splay(b);
212         /*
213         print(b);
214         puts("");
215         printf("%d %d %d %d\n", b->sum, b->nxt[0]->sum, b->val, b->nxt
            [1]->sum);
216         */
217         if(k==1)
218             printf("%d\n", b->max);
219         else
220         {
221             scanf("%d",&k);
222             set(b,k);
223         }
224     }
225     return 0;
226 }

```

1.5 Leftist tree

```

1 #include<stdio>
2 #include<algorithm>
3
4 #define MAXX 100111
5
6 int val[MAXX],l[MAXX],r[MAXX],d[MAXX];
7
8 int set[MAXX];
9
10 int merge(int a,int b)
11 {
12     if(!a)
13         return b;
14     if(!b)

```

```

15     return a;
16     if(val[a]<val[b]) // max-heap
17         std::swap(a,b);
18     r[a]=merge(r[a],b);
19     if(d[l[a]]<d[r[a]])
20         std::swap(l[a],r[a]);
21     d[a]=d[r[a]]+1;
22     set[l[a]]=set[r[a]]=a; // set a as father of its sons
23     return a;
24 }
25
26 inline int find(int &a)
27 {
28     while(set[a]) //brute-force to get the index of root
29         a=set[a];
30     return a;
31 }
32
33 inline void reset(int i)
34 {
35     l[i]=r[i]=d[i]=set[i]=0;
36 }
37
38 int n,i,j,k;
39
40 int main()
41 {
42     while(scanf("%d",&n)!=EOF)
43     {
44         for(i=1;i<=n;++i)
45         {
46             scanf("%d",&val[i]);
47             reset(i);
48         }
49         scanf("%d",&n);
50         while(n--)
51         {
52             scanf("%d %d",&i,&j);
53             if(find(i)==find(j))
54                 puts("-1");
55             else
56             {
57                 k=merge(l[i],r[i]);
58                 val[i]>>=1;
59                 reset(i);
60                 set[i]=merge(i,k);
61
62                 k=merge(l[j],r[j]);
63                 val[j]>>=1;
64                 reset(j);
65                 set[j]=merge(j,k);
66
67                 set[k=merge(i,j)]=0;
68                 printf("%d\n",val[k]);
69             }
70         }
71     }
72     return 0;
73 }

```

1.6 Network

```

1 //HLD_( : 3J2) _
2 #include<cstdio>
3 #include<algorithm>
4 #include<cstdlib>
5
6 #define MAXX 80111
7 #define MAXE (MAXX<<1)
8 #define N 18
9
10 int edge[MAXX],nxt[MAXE],to[MAXE],cnt;
11 int fa[MAXX][N],dg[MAXX];
12
13 inline int lca(int a,int b)
14 {
15     static int i,j;
16     j=0;
17     if(dg[a]<dg[b])
18         std::swap(a,b);
19     for(i=dg[a]-dg[b];i>=1,++j)
20         if(i&1)
21             a=fa[a][j];
22     if(a==b)
23         return a;
24     for(i=N-1;i>=0,--i)
25         if(fa[a][i]!=fa[b][i])
26             {
27                 a=fa[a][i];
28                 b=fa[b][i];
29             }
30     return fa[a][0];
31 }
32
33 inline void add(int a,int b)
34 {
35     nxt[++cnt]=edge[a];
36     edge[a]=cnt;
37     to[cnt]=b;
38 }
39
40 int sz[MAXX],pre[MAXX],next[MAXX];
41
42 void rr(int now)
43 {
44     sz[now]=1;
45     int max,id;
46     max=0;
47     for(int i=edge[now];i;i=nxt[i])
48         if(to[i]!=fa[now][0])
49             {
50                 fa[to[i]][0]=now;
51                 dg[to[i]]=dg[now]+1;
52                 rr(to[i]);

```

```

53                 sz[now]+=sz[to[i]];
54                 if(sz[to[i]]>max)
55                 {
56                     max=sz[to[i]];
57                     id=to[i];
58                 }
59             }
60     if(max)
61     {
62         next[now]=id;
63         pre[id]=now;
64     }
65 }
66
67 #define MAXT (MAXX*N*5)
68
69 namespace Treap
70 {
71     int cnt;
72     int son[MAXT][2],key[MAXT],val[MAXT],sz[MAXT];
73
74     inline void init()
75     {
76         key[0]=RAND_MAX;
77         val[0]=0xc0c0c0c0;
78         cnt=0;
79     }
80
81     inline void up(int id)
82     {
83         sz[id]=sz[son[id][0]]+sz[son[id][1]]+1;
84     }
85     inline void rot(int &id,int tp)
86     {
87         static int k;
88         k=son[id][tp];
89         son[id][tp]=son[k][tp^1];
90         son[k][tp^1]=id;
91         up(id);
92         up(k);
93         id=k;
94     }
95     void insert(int &id,int v)
96     {
97         if(id)
98         {
99             int k(v>val[id]);
100             insert(son[id][k],v);
101             if(key[son[id][k]]<key[id])
102                 rot(id,k);
103             else
104                 up(id);
105             return;
106         }
107         id=++cnt;
108         key[id]=rand()-1;
109         val[id]=v;
110         sz[id]=1;
111         son[id][0]=son[id][1]=0;
112     }
113     void del(int &id,int v)
114     {
115         if(!id)
116             return;
117         if(val[id]==v)
118         {
119             int k(key[son[id][1]]<key[son[id][0]]);
120             if(!son[id][k])
121             {
122                 id=0;
123                 return;
124             }
125             rot(id,k);
126             del(son[id][k^1],v);
127         }
128         else
129             del(son[id][v>val[id]],v);
130         up(id);
131     }
132     int rank(int id,int v)
133     {
134         if(!id)
135             return 0;
136         if(val[id]<=v)
137             return sz[son[id][0]]+1+rank(son[id][1],v);
138         return rank(son[id][0],v);
139     }
140     void print(int id)
141     {
142         if(!id)
143             return;
144         print(son[id][0]);
145         printf("%d ",val[id]);
146         print(son[id][1]);
147     }
148 }
149
150 int head[MAXX],root[MAXX],len[MAXX],pos[MAXX];
151
152 #define MAX (MAXX*6)
153 #define mid (l+r>>1)
154 #define lc lson[id],l,mid
155 #define rc rson[id],mid+1,r
156
157 int lson[MAX],rson[MAX];
158 int treap[MAX];
159
160 void make(int &id,int l,int r,int r,the)
161 {
162     id=++cnt;
163     static int k;
164     for(k=l;k<=r;++k)
165         Treap::insert(treap[id],the[k]);
166     if(l!=r)
167     {
168         make(lc,the);
169         make(rc,the);
170     }
171 }
172

```

```

173 int query(int id,int l,int r,int a,int b,int q)
174 {
175     if(a<=l && r<=b)
176         return Treap::rank(treap[id],q);
177     int re(0);
178     if(a<=mid)
179         re=query(lc,a,b,q);
180     if(b>mid)
181         re+=query(rc,a,b,q);
182     return re;
183 }
184
185 inline int query(int a,int b,int v)
186 {
187     static int re;
188     for(re=0;root[a]!=root[b];a=fa[root[a]][0])
189         re+=query(head[root[a]],1,len[root[a]],1,pos[a],v);
190     re+=query(head[root[b]],1,len[root[b]],pos[b],pos[a],v);
191     return re;
192 }
193
194 inline void update(int id,int l,int r,int pos,int val,int n)
195 {
196     while(l<=r)
197     {
198         Treap::del(treap[id],val);
199         Treap::insert(treap[id],n);
200         if(l==r)
201             return;
202         if(pos<=mid)
203         {
204             id=lson[id];
205             r=mid;
206         }
207         else
208         {
209             id=rson[id];
210             l=mid+1;
211         }
212     }
213 }
214
215 int n,q,i,j,k;
216 int val[MAXX];
217
218 int main()
219 {
220     srand(1e9+7);
221     scanf("%d %d",&n,&q);
222     for(i=1;i<=n;++i)
223         scanf("%d",&val[i]);
224     for(k=1;k<=n;++k)
225     {
226         scanf("%d %d",&i,&j);
227         add(i,j);
228         add(j,i);
229     }
230     rr(rand() % n+1);
231     for(j=1;j<=n;++j)
232         for(i=1;i<=n;++i)
233             fa[i][j]=fa[i][j-1][j-1];
234
235     Treap::init();
236     ont=0;
237     for(i=1;i<=n;++i)
238         if(!pre[i])
239         {
240             static int tmp[MAXX];
241             for(k=1,j=i;j; j=next[j],++k)
242             {
243                 pos[j]=k;
244                 root[j]=i;
245                 tmp[k]=val[j];
246             }
247             --k;
248             len[i]=k;
249             make(head[i],1,k,tmp);
250         }
251     while(q--)
252     {
253         scanf("%d",&k);
254         if(k)
255         {
256             static int a,b,c,d,l,r,ans,m;
257             scanf("%d %d",&a,&b);
258             c=lca(a,b);
259             if(dg[a]+dg[b]-2*dg[c]+1<k)
260             {
261                 puts("invalid request!");
262                 continue;
263             }
264             k=dg[a]+dg[b]-2*dg[c]+1-k+1;
265             if(dg[a]<dg[b])
266                 std::swap(a,b);
267             l=1e9;
268             r=1e9;
269             if(b!=c)
270             {
271                 d=a;
272                 for(i=0,j=dg[a]-dg[c]-1;j;j>>=1,++i)
273                     if(j&1)
274                         d=fa[d][i];
275                 while(l<=r)
276                 {
277                     m=l+r>>1;
278                     if(query(a,d,m)+query(b,c,m)>=k)
279                     {
280                         ans=m;
281                         r=m-1;
282                     }
283                     else
284                         l=m+1;
285                 }
286             }
287             else
288             {
289                 while(l<=r)
290                 {
291                     m=l+r>>1;
292                     if(query(a,c,m)>=k)

```

```

293             {
294                 ans=m;
295                 r=m-1;
296             }
297             else
298                 l=m+1;
299         }
300     }
301     printf("%d\n",ans);
302 }
303
304 else
305 {
306     scanf("%d %d",&i,&j);
307     update(head[root[i]],1,len[root[i]],pos[i],val[i],j);
308     val[i]=j;
309 }
310 return 0;
311 }

```

1.7 OTOCI

```

1 //down
2 //debugup/down/select
3 #include<cstdio>
4 #include<algorithm>
5
6 #define MAXX 30111
7
8 int nxt[MAXX][2],fa[MAXX],pre[MAXX],val[MAXX],sum[MAXX];
9 bool rev[MAXX];
10
11 inline void up(int id)
12 {
13     static int i;
14     sum[id]=val[id];
15     for(i=0;i<2;++i)
16         if(nxt[id][i])
17             sum[id]+=sum[nxt[id][i]];
18 }
19
20 inline void rot(int id,int tp)
21 {
22     static int k;
23     k=pre[id];
24     nxt[k][tp^1]=nxt[id][tp];
25     if(nxt[id][tp])
26         pre[nxt[id][tp]]=k;
27     if(pre[k])
28         nxt[pre[k]][k==nxt[pre[k]][1]]+=id;
29     pre[id]=pre[k];
30     nxt[id][tp]=k;
31     pre[k]=id;
32     up(k);
33     up(id);
34 }
35
36 inline void down(int id) //down
37 {
38     static int i;
39     if(rev[id])
40     {
41         rev[id]=false;
42         std::swap(nxt[id][0],nxt[id][1]);
43         for(i=0;i<2;++i)
44             if(nxt[id][i])
45                 rev[nxt[id][i]]^=true;
46     }
47 }
48
49 int freshen(int id)
50 {
51     int re(id);
52     if(pre[id])
53         re=freshen(pre[id]);
54     down(id);
55     return re;
56 }
57
58 inline void splay(int id)//down
59 {
60     static int rt;
61     if(id!=(rt=freshen(id)))
62         for(std::swap(fa[id],fa[rt]);pre[id];rot(id,id==nxt[pre[id]
63             ][0]));
64     /* another faster method:
65     if(id!=rt)
66     {
67         std::swap(fa[id],fa[rt]);
68         do
69         {
70             rt=pre[id];
71             if(pre[rt])
72             {
73                 k=(nxt[pre[rt]][0]==rt);
74                 if(nxt[rt][k]==id)
75                     rot(id,k^1);
76                 else
77                     rot(rt,k);
78                 rot(id,k);
79             }
80             else
81                 rot(id,id==nxt[rt][0]);
82         } while(pre[id]);
83     }
84     */
85 }
86
87 inline void access(int id)
88 {
89     static int to;
90     for(to=0;id;id=fa[id])
91     {

```

```

92     splay(id);
93     if (nxt[id][1])
94     {
95         pre[nxt[id][1]]=0;
96         fa[nxt[id][1]]=id;
97     }
98     nxt[id][1]=to;
99     if (to)
100     {
101         pre[to]=id;
102         fa[to]=0;
103     }
104     up(to=id);
105 }
106
107 inline int getrt(int id)
108 {
109     access(id);
110     splay(id);
111     while (nxt[id][0])
112     {
113         id=nxt[id][0];
114         down(id);
115     }
116     return id;
117 }
118
119 inline void makert(int id)
120 {
121     access(id);
122     splay(id);
123     if (nxt[id][0])
124         rev[id]=!true;
125 }
126
127 int n, i, j, k, q;
128 char buf[11];
129
130 int main()
131 {
132     scanf("%d", &n);
133     for (i=1; i<=n; ++i)
134         scanf("%d", val+i);
135     scanf("%d", &q);
136     while (q--)
137     {
138         scanf("%s %d %d", buf, &i, &j);
139         switch (buf[0])
140         {
141             case 'b':
142                 if (getrt(i)==getrt(j))
143                     puts("no");
144                 else
145                 {
146                     puts("yes");
147                     makert(i);
148                     fa[i]=j;
149                 }
150             case 'p':
151                 break;
152             case 'a':
153                 access(i);
154                 splay(i);
155                 val[i]=j;
156                 up(i);
157                 break;
158             case 's':
159                 if (getrt(i)!=getrt(j))
160                     puts("impossible");
161                 else
162                 {
163                     makert(i);
164                     access(j);
165                     splay(j);
166                     printf("%d\n", sum[j]);
167                 }
168                 break;
169         }
170     }
171     return 0;
172 }

```

```

31     if (l==1 && rr==r)
32     {
33         cnt[id]+=val;
34         if (cnt[id])
35         {
36             rt[id]=lf[id]=true;
37             len[id]=rmap[r]-rmap[l-1];
38             seg[id]=1;
39         }
40         else
41             if (l!=r)
42             {
43                 len[id]=len[id<<1]+len[id<<1|1];
44                 seg[id]=seg[id<<1]+seg[id<<1|1];
45                 if (rt[id<<1] && lf[id<<1|1])
46                     --seg[id];
47                 rt[id]=rt[id<<1|1];
48                 lf[id]=lf[id<<1];
49             }
50         else
51         {
52             len[id]=0;
53             rt[id]=lf[id]=false;
54             seg[id]=0;
55         }
56         return;
57     }
58     if (mid[id]>=r)
59         update(id<<1, 1, mid[id], l, r, val);
60     else
61         if (mid[id]<l)
62             update(id<<1|1, mid[id]+1, rr, l, r, val);
63         else
64         {
65             update(id<<1, 1, mid[id], l, mid[id], val);
66             update(id<<1|1, mid[id]+1, rr, mid[id]+1, r, val);
67         }
68     if (!cnt[id])
69     {
70         len[id]=len[id<<1]+len[id<<1|1];
71         seg[id]=seg[id<<1]+seg[id<<1|1];
72         if (rt[id<<1] && lf[id<<1|1])
73             --seg[id];
74         rt[id]=rt[id<<1|1];
75         lf[id]=lf[id<<1];
76     }
77 }
78
79 struct node
80 {
81     int l, r, h;
82     char val;
83     inline bool operator<(const node &a) const
84     {
85         return h==a.h?val<a.val:h<a.h; // trick watch out. val<a.val
86             ? val>a.val?
87     }
88     inline void print()
89     {
90         printf("%d %d %d %d\n", l, r, h, val);
91     }
92 } ln[inf];
93
94 int main()
95 {
96     make(1, 1, inf);
97     scanf("%d", &n);
98     n<<=1;
99     map.clear();
100     for (i=0; i<n; ++i)
101     {
102         scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
103         ln[i].l=x1;
104         ln[i].r=x2;
105         ln[i].h=y1;
106         ln[i].val=1;
107         ln[++i].l=x1;
108         ln[i].r=x2;
109         ln[i].h=y2;
110         ln[i].val=-1;
111         map[x1]=1;
112         map[x2]=1;
113     }
114     i=1;
115     for (it=map.begin(); it!=map.end(); ++it, ++i)
116     {
117         it->second=i;
118         rmap[i]=it->first;
119     }
120     i=0;
121     std::sort(ln, ln+n);
122     update(1, 1, inf, map[ln[0].l]+1, map[ln[0].r], ln[0].val);
123     sum+=len[1];
124     last=len[1];
125     for (i=1; i<n; ++i)
126     {
127         sum+=2*seg[i]*(ln[i].h-ln[i-1].h);
128         update(1, 1, inf, map[ln[i].l]+1, map[ln[i].r], ln[i].val);
129         sum+=abs(len[i]-last);
130         last=len[i];
131     }
132     printf("%lld\n", sum);
133     return 0;

```

1.8 picture

```

1 #include<cstdio>
2 #include<algorithm>
3 #include<map>
4
5 #define MAXX 5555
6 #define MAX MAXX<<3
7 #define inf 10011
8
9 int n, i;
10 int mid[MAX], cnt[MAX], len[MAX], seg[MAX];
11 bool rt[MAX], lf[MAX];
12
13 std::map<int, int> map;
14 std::map<int, int>::iterator it;
15 int rmap[inf];
16 long long sum;
17 int x1, x2, y1, y2, last;
18
19 void make(int id, int l, int r)
20 {
21     mid[id]=(l+r)>>1;
22     if (l!=r)
23     {
24         make(id<<1, l, mid[id]);
25         make(id<<1|1, mid[id]+1, r);
26     }
27 }
28
29 void update(int id, int ll, int rr, int l, int r, int val)
30 {

```

1.9 Size Blanced Tree

```

1 template<class Tp> class sbt
2 {
3     public:
4         inline void init()
5         {
6             rt=cnt=l[0]=r[0]=sz[0]=0;
7         }

```



```

8     inline void ins(const Tp &a)
9     {
10         ins(rt,a);
11     }
12     inline void del(const Tp &a)
13     {
14         del(rt,a);
15     }
16     inline bool find(const Tp &a)
17     {
18         return find(rt,a);
19     }
20     inline Tp pred(const Tp &a)
21     {
22         return pred(rt,a);
23     }
24     inline Tp succ(const Tp &a)
25     {
26         return succ(rt,a);
27     }
28     inline bool empty()
29     {
30         return !sz[rt];
31     }
32     inline Tp min()
33     {
34         return min(rt);
35     }
36     inline Tp max()
37     {
38         return max(rt);
39     }
40     inline void delsmall(const Tp &a)
41     {
42         dels(rt,a);
43     }
44     inline int rank(const Tp &a)
45     {
46         return rank(rt,a);
47     }
48     inline Tp sel(const int &a)
49     {
50         return sel(rt,a);
51     }
52     inline Tp dsel(int a)
53     {
54         return dsel(rt,a);
55     }
56 private:
57     int cnt,rt,l[MAXX],r[MAXX],sz[MAXX];
58     Tp val[MAXX];
59     inline void rro(int &pos)
60     {
61         int k(l[pos]);
62         l[pos]=r[k];
63         r[k]=pos;
64         sz[k]=sz[pos];
65         sz[pos]=sz[l[pos]]+sz[r[pos]]+1;
66         pos=k;
67     }
68     inline void lro(int &pos)
69     {
70         int k(r[pos]);
71         r[pos]=l[k];
72         l[k]=pos;
73         sz[k]=sz[pos];
74         sz[pos]=sz[l[pos]]+sz[r[pos]]+1;
75         pos=k;
76     }
77     inline void mt(int &pos,bool flag)
78     {
79         if(!pos)
80             return;
81         if(flag)
82             if(sz[r[r[pos]]]>sz[l[pos]])
83                 lro(pos);
84             else
85                 if(sz[l[r[pos]]]>sz[l[pos]])
86                 {
87                     rro(r[pos]);
88                     lro(pos);
89                 }
90             else
91                 return;
92         else
93             if(sz[l[l[pos]]]>sz[r[pos]])
94                 rro(pos);
95             else
96                 if(sz[r[l[pos]]]>sz[r[pos]])
97                 {
98                     lro(l[pos]);
99                     rro(pos);
100                 }
101             else
102                 return;
103         mt(l[pos],false);
104         mt(r[pos],true);
105         mt(pos,false);
106         mt(pos,true);
107     }
108     void ins(int &pos,const Tp &a)
109     {
110         if(pos)
111         {
112             ++sz[pos];
113             if(a<val[pos])
114                 ins(l[pos],a);
115             else
116                 ins(r[pos],a);
117             mt(pos,a>val[pos]);
118             return;
119         }
120         pos=++cnt;
121         l[pos]=r[pos]=0;
122         val[pos]=a;
123         sz[pos]=1;
124     }
125     Tp del(int &pos,const Tp &a)
126     {
127         --sz[pos];
128
129         if(val[pos]==a || (a<val[pos] && !l[pos]) || (a>val[pos]
130             && !r[pos]))
131         {
132             Tp ret(val[pos]);
133             if(!l[pos] || !r[pos])
134                 pos=l[pos]+r[pos];
135             else
136                 val[pos]=del(l[pos],val[pos]+1);
137             return ret;
138         }
139         else
140             if(a<val[pos])
141                 return del(l[pos],a);
142             else
143                 return del(r[pos],a);
144     }
145     bool find(int &pos,const Tp &a)
146     {
147         if(!pos)
148             return false;
149         if(a<val[pos])
150             return find(l[pos],a);
151         else
152             return (val[pos]==a || find(r[pos],a));
153     }
154     Tp pred(int &pos,const Tp &a)
155     {
156         if(!pos)
157             return a;
158         if(a>val[pos])
159         {
160             Tp ret(pred(r[pos],a));
161             if(ret==a)
162                 return val[pos];
163             else
164                 return ret;
165         }
166         return pred(l[pos],a);
167     }
168     Tp succ(int &pos,const Tp &a)
169     {
170         if(!pos)
171             return a;
172         if(a<val[pos])
173         {
174             Tp ret(succ(l[pos],a));
175             if(ret==a)
176                 return val[pos];
177             else
178                 return ret;
179         }
180         return succ(r[pos],a);
181     }
182     Tp min(int &pos)
183     {
184         if(l[pos])
185             return min(l[pos]);
186         else
187             return val[pos];
188     }
189     Tp max(int &pos)
190     {
191         if(r[pos])
192             return max(r[pos]);
193         else
194             return val[pos];
195     }
196     void dels(int &pos,const Tp &v)
197     {
198         if(!pos)
199             return;
200         if(val[pos]<v)
201         {
202             pos=r[pos];
203             dels(pos,v);
204             return;
205         }
206         dels(l[pos],v);
207         sz[pos]=1+sz[l[pos]]+sz[r[pos]];
208     }
209     int rank(const int &pos,const Tp &v)
210     {
211         if(val[pos]==v)
212             return sz[l[pos]]+1;
213         if(v<val[pos])
214             return rank(l[pos],v);
215         return rank(r[pos],v)+sz[l[pos]]+1;
216     }
217     Tp sel(const int &pos,const int &v)
218     {
219         if(sz[l[pos]]+1==v)
220             return val[pos];
221         if(v>sz[l[pos]])
222             return sel(r[pos],v-sz[l[pos]]-1);
223         return sel(l[pos],v);
224     }
225     Tp dsel(int &pos,int k)
226     {
227         --sz[pos];
228         if(sz[l[pos]]+1==k)
229         {
230             Tp re(val[pos]);
231             if(!l[pos] || !r[pos])
232                 pos=l[pos]+r[pos];
233             else
234                 val[pos]=del(l[pos],val[pos]+1);
235             return re;
236         }
237         if(k>sz[l[pos]])
238             return dsel(r[pos],k-1-sz[l[pos]]);
239         return dsel(l[pos],k);
240     }
241     };

```

1.10 Sparse Table - rectangle

```

1  #include<iostream>
2  #include<cstdio>
3  #include<algorithm>
4
5  #define MAXX 310
6
7  int mat[MAXX][MAXX];
8  int table[9][9][MAXX][MAXX];
9  int n;
10 short lg[MAXX];
11
12 int main()
13 {
14     for(int i(2);i<MAXX;++i)
15         lg[i]=lg[i>>1]+1;
16     int T;
17     std::cin >> T;
18     while (T--)
19     {
20         std::cin >> n;
21         for (int i = 0; i < n; ++i)
22             for (int j = 0; j < n; ++j)
23             {
24                 std::cin >> mat[i][j];
25                 table[0][0][i][j] = mat[i][j];
26             }
27
28         //
29         for(int i=0;i<lg[n];++i) // width
30         {
31             for(int j=0;j<lg[n];++j) //height
32             {
33                 if(i==0 && j==0)
34                     continue;
35                 for(int ii=0;ii+(1<<j)<=n;++ii)
36                     for(int jj=0;jj+(1<<i)<=n;++jj)
37                         if(i==0)
38                             table[i][j][ii][jj]=std::min(table[i][j-1][ii][jj],table[i][j-1][ii+(1<<(j-1))][jj]);
39
40                 else
41                     table[i][j][ii][jj]=std::min(table[i-1][j][ii][jj],table[i-1][j][ii+(1<<(i-1))][jj]);
42             }
43         }
44         long long N;
45         std::cin >> N;
46         int r1, c1, r2, c2;
47         for (int i = 0; i < N; ++i)
48         {
49             scanf("%d%d%d%d",&r1,&c1,&r2,&c2);
50             --r1;
51             --c1;
52             --r2;
53             --c2;
54             int w=lg[c2-c1+1];
55             int h=lg[r2-r1+1];
56             printf("%d\n",std::min(table[w][h][r1][c1],std::min(table[w][h][r1][c2-(1<<w)+1],std::min(table[w][h][r2-(1<<h)+1][c1],table[w][h][r2-(1<<h)+1][c2-(1<<w)+1]))));
57         }
58     }
59     return 0;

```

1.11 Sparse Table - square

```

1  int num[MAXX][MAXX],max[MAXX][MAXX][10];
2  short lg[MAXX];
3
4  int main()
5  {
6      for(i=2;i<MAXX;++i)
7          lg[i]=lg[i>>1]+1;
8      scanf("%hd %d",&n,&q);
9      for(i=0;i<n;++i)
10         for(j=0;j<n;++j)
11         {
12             scanf("%d",num[i][j]);
13             max[i][j][0]=num[i][j];
14         }
15         for(k=1;k<lg[n];++k)
16         {
17             l=n+1-(1<<k);
18             for(i=0;i<l;++i)
19                 for(j=0;j<l;++j)
20                     max[i][j][k]=std::max(std::max(max[i][j][k-1],max[i+(1<<(k-1))][j][k-1]),std::max(max[i][j+(1<<(k-1))][k-1],max[i+(1<<(k-1))][j+(1<<(k-1))][k-1]));
21         }
22         printf("Case %hd:\n",t);
23         while(q--)
24         {
25             scanf("%hd %hd %hd",&i,&j,&l);
26             --i;
27             --j;
28             k=lg[l];
29             printf("%d\n",std::max(std::max(max[i][j][k],max[i][j+1-(1<<k)][k]),std::max(max[i+1-(1<<k)][j][k],max[i+1-(1<<k)][j+1-(1<<k)][k])));
30         }
31     }

```

1.12 Sparse Table

```

1  int num[MAXX],min[MAXX][20];
2  int lg[MAXX];
3
4
5  int main()
6  {
7      for(i=2;i<MAXX;++i)
8          lg[i]=lg[i>>1]+1;
9      scanf("%d %d",&n,&q);
10     for(i=1;i<=n;++i)
11     {
12         scanf("%d",num+i);
13         min[i][0]=num[i];
14     }
15     for(j=1;j<=lg[n];++j)
16     {
17         l=n+1-(1<<j);
18         j_-=1;
19         j_=(1<<j_);
20         for(i=1;i<=l;++i)
21             min[i][j]=std::min(min[i][j_],min[i+j_][j_]);
22     }
23     printf("Case %hd:\n",t);
24     while(q--)
25     {
26         scanf("%d %d",&i,&j);
27         k=lg[j-i+1];
28         printf("%d\n",std::min(min[i][k],min[j-(1<<k)+1][k]));
29     }
30 }

```

1.13 Trea

```

1  #include<cstdlib>
2  #include<ctime>
3  #include<cstring>
4
5  struct node
6  {
7      char *ch[2];
8      int sz,val,key;
9      node() {memset(this,0,sizeof(node));}
10     node(int a);
11 }*null;
12
13 node::~node(int a):sz(1),val(a),key(rand()-1){ch[0]=ch[1]=null;}
14
15 class Trepap
16 {
17     inline void up(node *pos)
18     {
19         pos->sz=pos->ch[0]->sz+pos->ch[1]->sz+1;
20     }
21     inline void rot(node *pos,int tp)
22     {
23         node *k(pos->ch[tp]);
24         pos->ch[tp]=k->ch[tp^1];
25         k->ch[tp^1]=pos;
26         up(pos);
27         up(k);
28         pos=k;
29     }
30
31     void insert(node *pos,int val)
32     {
33         if(pos!=null)
34         {
35             int t(val>pos->val);
36             insert(pos->ch[t],val);
37             if(pos->ch[t]->key<pos->key)
38                 rot(pos,t);
39             else
40                 up(pos);
41             return;
42         }
43         pos=new node(val);
44     }
45     void rec(node *pos)
46     {
47         if(pos!=null)
48         {
49             rec(pos->ch[0]);
50             rec(pos->ch[1]);
51             delete pos;
52         }
53     }
54     inline int sel(node *pos,int k)
55     {
56         while(pos->ch[0]->sz+1!=k)
57             if(pos->ch[0]->sz>=k)
58                 pos=pos->ch[0];
59             else
60             {
61                 k=pos->ch[0]->sz+1;
62                 pos=pos->ch[1];
63             }
64         return pos->val;
65     }
66     void del(node *pos,int val)
67     {
68         if(pos!=null)
69         {
70             if(pos->val==val)
71             {
72                 int t(pos->ch[1]->key<pos->ch[0]->key);
73                 if(pos->ch[t]==null)
74                 {
75                     delete pos;
76                     pos=null;
77                     return;

```

```

78         }
79         rot(pos,t);
80         del(pos->ch[t^1],val);
81     }
82     else
83         del(pos->ch[val>pos->val],val);
84     up(pos);
85 }
86 }
87 public:
88     node *rt;
89
90     Treap():rt(NULL){}
91     inline void insert(int val)
92     {
93         insert(rt,val);
94     }
95     inline void reset()
96     {
97         rec(rt);
98         rt=NULL;
99     }
100     inline int sel(int k)
101     {
102         if(k<1 || k>rt->sz)
103             return 0;
104         return sel(rt->sz+1-k);
105     }
106     inline void del(int val)
107     {
108         del(rt,val);
109     }
110     inline int size()
111     {
112         return rt->sz;
113     }
114 }treap[MAXX];
115
116 int:
117 {
118     srand(time(0));
119     null=new node();
120     null->val=0xc0c0c0c0;
121     null->sz=0;
122     null->key=RAND_MAX;
123     null->ch[0]=null->ch[1]=null;
124     for(i=0;i<MAXX;++i)
125         treap[i].rt=null;
126 }

```

2 dynamic programming

2.1 knapsack problem

```

1 multiple-choice knapsack problem:
2
3 for k
4     for v=V..0
5         for ik
6             f[v]=max(f[v],f[v-c[i]]+w[i])

```

2.2 LCIS

```

1 #include<cstdio>
2 #include<cstring>
3 #include<vector>
4
5 #define MAXX 1111
6
7 int T;
8 int n,m,p,i,j,k;
9 std::vector<int>the[2];
10 int dp[MAXX],path[MAXX];
11 int ans[MAXX];
12
13 int main()
14 {
15     the[0].reserve(MAXX);
16     the[1].reserve(MAXX);
17     {
18         scanf("%d",&n);
19         the[0].resize(n);
20         for(i=0;i<n;++i)
21             scanf("%d",&the[0][i]);
22         scanf("%d",&m);
23         the[1].resize(m);
24         for(i=0;i<m;++i)
25             scanf("%d",&the[1][i]);
26         memset(dp,0,sizeof dp);
27         for(i=0;i<the[0].size();++i)
28         {
29             n=0;
30             p=-1;
31             for(j=0;j<the[1].size();++j)
32             {
33                 if(the[0][i]==the[1][j] && n+1>dp[j])
34                 {
35                     dp[j]=n+1;
36                     path[j]=i;
37                 }
38                 if(the[1][j]<the[0][i] && n<dp[j])
39                 {
40                     n=dp[j];
41                     p=j;
42                 }

```

```

43     }
44 }
45 n=0;
46 p=-1;
47 for(i=0;i<the[1].size();++i)
48     if(dp[i]>n)
49         n=dp[p=i];
50 printf("%d\n",n);
51 for(i=n-1;i>=0;--i)
52 {
53     ans[i]=the[1][p];
54     p=path[p];
55 }
56 for(i=0;i<n;++i)
57     printf("%d ",ans[i]);
58 puts("");
59 }
60 return 0;
61 }

```

2.3 LCS

```

1 #include<cstdio>
2 #include<algorithm>
3 #include<vector>
4
5 #define MAXX 111
6 #define N 128
7
8 std::vector<char>the[2];
9 std::vector<int>dp(MAXX),p[N];
10
11 int i,j,k;
12 char buf[MAXX];
13 int t;
14
15 int main()
16 {
17     the[0].reserve(MAXX);
18     the[1].reserve(MAXX);
19     while(gets(buf),buf[0]!='#')
20     {
21         the[0].resize(0);
22         for(i=0;buf[i];++i)
23             the[0].push_back(buf[i]);
24         the[1].resize(0);
25         gets(buf);
26         for(i=0;buf[i];++i)
27             the[1].push_back(buf[i]);
28         for(i=0;i<N;++i)
29             p[i].resize(0);
30         for(i=0;i<the[1].size();++i)
31             p[the[1][i]].push_back(i);
32         dp.resize(1);
33         dp[0]=-1;
34         for(i=0;i<the[0].size();++i)
35             for(j=p[the[0][i]].size()-1;j>=0;--j)
36             {
37                 k=p[the[0][i]][j];
38                 if(k>dp.back())
39                     dp.push_back(k);
40                 else
41                     *std::lower_bound(dp.begin(),dp.end(),k)=k;
42             }
43         printf("Case %d: you can visit at most %d cities.\n",++t,
44             dp.size()-1);
45     }
46     return 0;
47 }

```

3 geometry

3.1 3D

```

1 struct pv
2 {
3     double x,y,z;
4     pv() {}
5     pv(double xx,double yy,double zz):x(xx),y(yy),z(zz) {}
6     pv operator -(const pv& b)const
7     {
8         return pv(x-b.x,y-b.y,z-b.z);
9     }
10     pv operator *(const pv& b)const
11     {
12         return pv(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);
13     }
14     double operator &(const pv& b)const
15     {
16         return x*b.x+y*b.y+z*b.z;
17     }
18 };
19
20 //
21 double Norm(pv p)
22 {
23     return sqrt(p&p);
24 }
25
26 // \theta
27 pv Trans(pv pa,pv V,double theta)
28 {
29     double s = sin(theta);
30     double c = cos(theta);
31     double x,y,z;
32     x = V.x;

```

```

33     y = V.y;
34     z = V.z;
35     pv pp =
36         pv (
37             (x*x*(1-c)+c)*pa.x+(x*y*(1-c)-z*s)*pa.y+(x*z*(1-c)+y*s
38                 )*pa.z,
39             (y*x*(1-c)+z*s)*pa.x+(y*y*(1-c)+c)*pa.y+(y*z*(1-c)-x*s
40                 )*pa.z,
41             (x*z*(1-c)-y*s)*pa.x+(y*z*(1-c)+x*s)*pa.y+(z*z*(1-c)+c
42                 )*pa.z
43         );
44     //
45     x=r*sin()*cos();
46     y=r*sin()*sin();
47     z=r*cos();
48     r=sqrt(x*2+y*2+z*2);
49     r=sqrt(x^2+y^2+z^2);
50     r=atan(y/x);
51     r=acos(z/r);
52     r[0,]
53     [0,2]
54     [0,]
55     lat1[-/2,/2]
56     lng1[-,]
57     pv getpv(double lat,double lng,double r)
58     {
59         lat += pi/2;
60         lng += pi;
61         return
62             pv(r*sin(lat)*cos(lng),r*sin(lat)*sin(lng),r*cos(lat));
63     }
64     //
65     #include<stdio>
66     #include<cmath>
67     #define MAXX 1111
68     char buf[MAXX];
69     const double r=6875.0/2,pi=acos(-1.0);
70     double a,b,c,x1,x2,y2,ans;
71     int main()
72     {
73         double y1;
74         while(gets(buf)!=NULL)
75         {
76             gets(buf);
77             gets(buf);
78             scanf("%lf%lf%lf" %s\n", &a, &b, &c, buf);
79             x1=a+b/60+c/3600;
80             x1=x1*pi/180;
81             if(buf[0]!='S')
82                 x1=-x1;
83             scanf("%s",buf);
84             scanf("%lf%lf%lf" %s\n", &a, &b, &c, buf);
85             y1=a+b/60+c/3600;
86             y1=y1*pi/180;
87             if(buf[0]!='W')
88                 y1=-y1;
89             gets(buf);
90             scanf("%lf%lf%lf" %s\n", &a, &b, &c, buf);
91             x2=a+b/60+c/3600;
92             x2=x2*pi/180;
93             if(buf[0]!='S')
94                 x2=-x2;
95             scanf("%s",buf);
96             scanf("%lf%lf%lf" %s\n", &a, &b, &c, buf);
97             y2=a+b/60+c/3600;
98             y2=y2*pi/180;
99             if(buf[0]!='W')
100                 y2=-y2;
101             ans=acos(cos(x1)*cos(x2)*cos(y1-y2)+sin(x1)*sin(x2))*r;
102             printf("The distance to the iceberg: %.21f miles.\n",ans);
103             if(ans>0.005<100)
104                 puts("DANGER!");
105             gets(buf);
106             return 0;
107         }
108     }
109     inline bool ZERO(const double &a)
110     {
111         return fabs(a)<eps;
112     }
113     //
114     inline bool ZERO(pv p)
115     {
116         return (ZERO(p.x) && ZERO(p.y) && ZERO(p.z));
117     }
118     //
119     bool LineIntersect(Line3D L1, Line3D L2)
120     {
121         pv s = L1.s-L1.e;
122         pv e = L2.s-L2.e;
123         pv p = s*e;
124         if (ZERO(p))
125             return false;
126         p = (L2.s-L1.e)*(L1.s-L1.e);
127         return ZERO(p&L2.e);
128     }

```

```

150     //
151     bool inter(pv a,pv b,pv c,pv d)
152     {
153         pv ret = (a-b)*(c-d);
154         pv t1 = (b-a)*(c-a);
155         pv t2 = (b-a)*(d-a);
156         pv t3 = (d-c)*(a-c);
157         pv t4 = (d-c)*(b-c);
158         return sgn(t1&ret)*sgn(t2&ret) < 0 && sgn(t3&ret)*sgn(t4&ret) <
159             0;
160     }
161     //
162     bool OnLine(pv p, Line3D L)
163     {
164         return ZERO((p-L.s)*(L.e-L.s));
165     }
166     //
167     bool OnSeg(pv p, Line3D L)
168     {
169         return (ZERO((L.s-p)*(L.e-p)) && EQ(Norm(p-L.s)+Norm(p-L.e),
170             Norm(L.e-L.s)));
171     }
172     //
173     double Distance(pv p, Line3D L)
174     {
175         return (Norm((p-L.s)*(L.e-L.s))/Norm(L.e-L.s));
176     }
177     //
178     double Inclination(Line3D L1, Line3D L2)
179     {
180         pv u = L1.e - L1.s;
181         pv v = L2.e - L2.s;
182         return acos( (u & v) / (Norm(u)*Norm(v)) );
183     }

```

3.2 3DCH

```

1     #include<stdio>
2     #include<cmath>
3     #include<vector>
4     #include<algorithm>
5
6     #define MAXX 1111
7     #define eps 1e-8
8     #define inf 1e20
9
10    struct pv
11    {
12        double x,y,z;
13        pv(){}
14        pv(const double &xx,const double &yy,const double &zz):x(xx),y(
15            yy),z(zz){}
16        inline pv operator-(const pv &i)const
17        {
18            return pv(x-i.x,y-i.y,z-i.z);
19        }
20        inline pv operator+(const pv &i)const //
21        {
22            return pv(y+i.z-z+i.y,z+i.x-x+i.z,x+i.y-y+i.x);
23        }
24        inline double operator^(const pv &i)const //
25        {
26            return x*i.x+y*i.y+z*i.z;
27        }
28        inline double len()
29        {
30            return sqrt(x*x+y*y+z*z);
31        }
32    };
33    struct pla
34    {
35        short a,b,c;
36        bool ok;
37        pla(){}
38        pla(const short &aa,const short &bb,const short &cc):a(aa),b(bb
39            ),c(cc),ok(true){}
40        inline void set();
41        inline void print()
42        {
43            printf("%hd %hd %hd\n",a,b,c);
44        }
45    };
46    pv pnt [MAXX];
47    std::vector<pla>fac;
48    short to [MAXX] [MAXX];
49
50    inline void pla::set()
51    {
52        to[a][b]=to[b][c]=to[c][a]=fac.size();
53    }
54
55    inline double ptof(const pv &p,const pla &f) //?
56    {
57        return (pnt[f.b]-pnt[f.a])*(pnt[f.c]-pnt[f.a])^(p-pnt[f.a]);
58    }
59
60    inline double vol(const pv &a,const pv &b,const pv &c,const pv &d)
61        //6
62    {
63        return (b-a)*(c-a)^(d-a);
64    }
65    inline double ptof(const pv &p,const short &f) //pF
66    {
67        return fabs(vol(pnt[fac[f].a],pnt[fac[f].b],pnt[fac[f].c],p)/((
68            pnt[fac[f].b]-pnt[fac[f].a])*(pnt[fac[f].c]-pnt[fac[f].a
69            ])).len());

```

```

68 }
69
70 void dfs(const short&, const short&);
71
72 void deal(const short &p, const short &a, const short &b)
73 {
74     if(fac[to[a][b]].ok)
75         if(ptof(pnt[p], fac[to[a][b]]) > eps)
76             dfs(p, to[a][b]);
77     else
78     {
79         pla add(b, a, p);
80         add.set();
81         fac.push_back(add);
82     }
83 }
84
85 void dfs(const short &p, const short &now)
86 {
87     fac[now].ok = false;
88     deal(p, fac[now].b, fac[now].a);
89     deal(p, fac[now].c, fac[now].b);
90     deal(p, fac[now].a, fac[now].c);
91 }
92
93 inline void make()
94 {
95     fac.resize(0);
96     if(n < 4)
97         return;
98
99     for(i = 1; i < n; ++i)
100         if((pnt[0] - pnt[i]).len() > eps)
101         {
102             std::swap(pnt[i], pnt[1]);
103             break;
104         }
105     if(i == n)
106         return;
107
108     for(i = 2; i < n; ++i)
109         if(((pnt[0] - pnt[1]) * (pnt[1] - pnt[i])).len() > eps)
110         {
111             std::swap(pnt[i], pnt[2]);
112             break;
113         }
114     if(i == n)
115         return;
116
117     for(i = 3; i < n; ++i)
118         if(fabs((pnt[0] - pnt[1]) * (pnt[1] - pnt[2]) ^ (pnt[2] - pnt[i])) > eps)
119         {
120             std::swap(pnt[3], pnt[i]);
121             break;
122         }
123     if(i == n)
124         return;
125
126     for(i = 0; i < 4; ++i)
127     {
128         pla add((i+1)%4, (i+2)%4, (i+3)%4);
129         if(ptof(pnt[i], add) > 0)
130             std::swap(add.c, add.b);
131         add.set();
132         fac.push_back(add);
133     }
134     for(; i < n; ++i)
135         for(j = 0; j < fac.size(); ++j)
136             if(fac[j].ok && ptof(pnt[i], fac[j]) > eps)
137             {
138                 dfs(i, j);
139                 break;
140             }
141
142     short tmp(fac.size());
143     fac.resize(0);
144     for(i = 0; i < tmp; ++i)
145         if(fac[i].ok)
146             fac.push_back(fac[i]);
147 }
148
149 inline pv gc() //
150 {
151     pv re(0, 0, 0), o(0, 0, 0);
152     double all(0), v;
153     for(i = 0; i < fac.size(); ++i)
154     {
155         v = vol(o, pnt[fac[i].a], pnt[fac[i].b], pnt[fac[i].c]);
156         re += (pnt[fac[i].a] + pnt[fac[i].b] + pnt[fac[i].c]) * 0.25 * v;
157         all += v;
158     }
159     return re * (1/all);
160 }
161
162 inline bool same(const short &s, const short &t) //
163 {
164     pv &a = pnt[fac[s].a], &b = pnt[fac[s].b], &c = pnt[fac[s].c];
165     return fabs(vol(a, b, c, pnt[fac[t].a])) < eps && fabs(vol(a, b, c, pnt[fac[t].b])) < eps && fabs(vol(a, b, c, pnt[fac[t].c])) < eps;
166 }
167
168 //
169 inline short facetcnt()
170 {
171     short ans = 0;
172     for(short i = 0; i < fac.size(); ++i)
173     {
174         for(j = 0; j < i; ++j)
175             if(same(i, j))
176                 break;
177         if(j == i)
178             ++ans;
179     }
180     return ans;
181 }
182
183 //
184 inline short trianglecnt()
185 {

```

```

186     return fac.size();
187 }
188
189 // *2
190 inline double area(const pv &a, const pv &b, const pv &c)
191 {
192     return (b-a) * (c-a).len();
193 }
194
195 //
196 inline double area()
197 {
198     double ret(0);
199     for(i = 0; i < fac.size(); ++i)
200         ret += area(pnt[fac[i].a], pnt[fac[i].b], pnt[fac[i].c]);
201     return ret/2;
202 }
203
204 //
205 inline double volume()
206 {
207     pv o(0, 0, 0);
208     double ret(0);
209     for(short i = 0; i < fac.size(); ++i)
210         ret += vol(o, pnt[fac[i].a], pnt[fac[i].b], pnt[fac[i].c]);
211     return fabs(ret/6);
212 }

```

3.3 circle ploy's intersection area

```

1 bool InCircle(Point a, double r)
2 {
3     return cmp(a.x*a.x+a.y*a.y, r*r) <= 0;
4     // 'EPS'
5 }
6
7 double CalcArea(Point a, Point b, double r)
8 {
9     Point p[4];
10    int tot = 0;
11    p[tot++] = a;
12
13    Point tv = Point(a, b);
14    Line tmp = Line(Point(0, 0), Point(tv.y, -tv.x));
15    Point near = LineToLine(Line(a, b), tmp);
16    if (cmp(near.x*near.x+near.y*near.y, r*r) <= 0)
17    {
18        double A, B, C;
19        A = near.x*near.x+near.y*near.y;
20        C = r;
21        B = C*C-A;
22        double tv1 = tv.x*tv.x+tv.y*tv.y;
23        double tmp = sqrt(B/tv1); //
24        p[tot] = Point(near.x+tmp*tv.x, near.y+tmp*tv.y);
25        if (OnSeg(Line(a, b), p[tot]) == true) tot++;
26        p[tot] = Point(near.x-tmp*tv.x, near.y-tmp*tv.y);
27        if (OnSeg(Line(a, b), p[tot]) == true) tot++;
28    }
29    if (tot == 3)
30    {
31        if (cmp(Point(p[0], p[1]).Length(), Point(p[0], p[2]).Length()) > 0)
32            swap(p[1], p[2]);
33    }
34    p[tot++] = b;
35
36    double res = 0.0, theta, a0, al, sgn;
37    for (int i = 0; i < tot-1; i++)
38    {
39        if (InCircle(p[i], r) == true && InCircle(p[i+1], r) == true)
40        {
41            res += 0.5 * xmult(p[i], p[i+1]);
42        }
43        else
44        {
45            a0 = atan2(p[i+1].y, p[i+1].x);
46            al = atan2(p[i].y, p[i].x);
47            if (a0 < al) a0 += 2*pi;
48            theta = a0 - al;
49            if (cmp(theta, pi) >= 0) theta = 2*pi - theta;
50            sgn = xmult(p[i], p[i+1]) / 2.0;
51            if (cmp(sgn, 0) < 0) theta = -theta;
52            res += 0.5 * r * r * theta;
53        }
54    }
55    return res;
56 }
57
58 //
59
60 area2 = 0.0;
61 for (int i = 0; i < resn; i++) //
62     area2 += CalcArea(p[i], p[(i+1)%resn], r);

```

3.4 circle's area

```

1 //
2 {
3     for (int i = 0; i < n; i++)
4     {
5         scanf("%lf%lf%lf", &c[i].c.x, &c[i].c.y, &c[i].r);
6         del[i] = false;
7     }
8     for (int i = 0; i < n; i++)
9         if (del[i] == false)
10         {
11             if (c[i].r == 0.0)
12                 del[i] = true;

```

```

13     for (int j = 0; j < n; j++)
14         if (i != j)
15             if (del[j] == false)
16                 if (cmp(Point(c[i].c, c[j].c).Len() + c[i].r, c[j].r
17                     ) <= 0)
18                     del[i] = true;
19     }
20     tn = n;
21     n = 0;
22     for (int i = 0; i < tn; i++)
23         if (del[i] == false)
24             c[n++] = c[i];
25 }
26 //ans[i]
27 const double pi = acos(-1.0);
28 const double eps = 1e-8;
29 struct Point
30 {
31     double x, y;
32     Point() {}
33     Point(double _x, double _y)
34     {
35         x = _x;
36         y = _y;
37     }
38     double Length()
39     {
40         return sqrt(x*x+y*y);
41     }
42 };
43 struct Circle
44 {
45     Point c;
46     double r;
47 };
48 struct Event
49 {
50     double tim;
51     int typ;
52     Event() {}
53     Event(double _tim, int _typ)
54     {
55         tim = _tim;
56         typ = _typ;
57     }
58 };
59
60 int cmp(const double& a, const double& b)
61 {
62     if (fabs(a-b) < eps) return 0;
63     if (a < b) return -1;
64     return 1;
65 }
66
67 bool Eventcmp(const Event& a, const Event& b)
68 {
69     return cmp(a.tim, b.tim) < 0;
70 }
71
72 double Area(double theta, double r)
73 {
74     return 0.5*r*r*(theta-sin(theta));
75 }
76
77 double xmult(Point a, Point b)
78 {
79     return a.x*b.y - a.y*b.x;
80 }
81
82 int n, cur, tote;
83 Circle c[1000];
84 double ans[1001], pre[1001], AB, AC, BC, theta, fai, a0, a1;
85 Event e[4000];
86 Point lab;
87
88 int main()
89 {
90     while (scanf("%d", &n) != EOF)
91     {
92         for (int i = 0; i < n; i++)
93             scanf("%lf%lf%lf", &c[i].c.x, &c[i].c.y, &c[i].r);
94         for (int i = 1; i <= n; i++)
95             ans[i] = 0.0;
96         for (int i = 0; i < n; i++)
97         {
98             tote = 0;
99             e[tote++] = Event(-pi, 1);
100             e[tote++] = Event(pi, -1);
101             for (int j = 0; j < n; j++)
102                 if (j != i)
103                 {
104                     lab = Point(c[j].c.x - c[i].c.x, c[j].c.y - c[i].c.y);
105                     AB = lab.Length();
106                     AC = c[i].r;
107                     BC = c[j].r;
108                     if (cmp(AB+AC, BC) <= 0)
109                     {
110                         e[tote++] = Event(-pi, 1);
111                         e[tote++] = Event(pi, -1);
112                         continue;
113                     }
114                     if (cmp(AB+BC, AC) <= 0) continue;
115                     if (cmp(AB, AC+BC) > 0) continue;
116                     theta = atan2(lab.y, lab.x);
117                     fai = acos((AC*AC+AB*AB-BC*BC)/(2.0*AC*AB));
118                     a0 = theta-fai;
119                     if (cmp(a0, -pi) < 0) a0 += 2*pi;
120                     a1 = theta+fai;
121                     if (cmp(a1, pi) > 0) a1 -= 2*pi;
122                     if (cmp(a0, a1) > 0)
123                     {
124                         e[tote++] = Event(a0, 1);
125                         e[tote++] = Event(pi, -1);
126                         e[tote++] = Event(-pi, 1);
127                         e[tote++] = Event(a1, -1);
128                     }
129                     else
130                     {
131                         e[tote++] = Event(a0, 1);

```

```

132             e[tote++] = Event(a1, -1);
133         }
134     }
135     sort(e, e+tote, Eventcmp);
136     cur = 0;
137     for (int j = 0; j < tote; j++)
138     {
139         if (cur != 0 && cmp(e[j].tim, pre[cur]) != 0)
140         {
141             ans[cur] += Area(e[j].tim - pre[cur], c[i].r);
142             ans[cur] += xmult(Point(c[i].c.x + c[i].r*cos(pre[cur]
143                 ), c[i].c.y + c[i].r*sin(pre[cur])),
144                 Point(c[i].c.x + c[i].r*cos(e[j].tim), c[i].c.y +
145                     c[i].r*sin(e[j].tim)))/2.0;
146             cur += e[j].typ;
147             pre[cur] = e[j].tim;
148         }
149     }
150     for (int i = 1; i < n; i++)
151         ans[i] -= ans[i+1];
152     for (int i = 1; i <= n; i++)
153         printf("%d] = %.3f\n", i, ans[i]);
154     return 0;
155 }

```

3.5 circle

```

1 //
2 #include<cstdio>
3 #include<cmath>
4 #include<vector>
5 #include<algorithm>
6
7 #define MAXX 333
8 #define eps 1e-8
9
10 struct pv
11 {
12     double x, y;
13     pv() {}
14     pv(const double &xx, const double &yy):x(xx), y(yy) {}
15     inline pv operator-(const pv &i) const
16     {
17         return pv(x-i.x, y-i.y);
18     }
19     inline double cross(const pv &i) const
20     {
21         return x*i.y - y*i.x;
22     }
23     inline void print()
24     {
25         printf("%lf %lf\n", x, y);
26     }
27     inline double len()
28     {
29         return sqrt(x*x+y*y);
30     }
31 } pnt[MAXX];
32
33 struct node
34 {
35     double k;
36     bool flag;
37     node() {}
38     node(const double &kk, const bool &ff):k(kk), flag(ff) {}
39     inline bool operator<(const node &i) const
40     {
41         return k < i.k;
42     }
43 };
44
45 std::vector<node> alpha;
46
47 short n, i, j, k, l;
48 short ans, sum;
49 double R=2;
50 double theta, phi, d;
51 const double pi(acos(-1.0));
52
53 int main()
54 {
55     alpha.reserve(MAXX<<1);
56     while (scanf("%hd", &n), n)
57     {
58         for (i=0; i<n; ++i)
59             scanf("%lf %lf", &pnt[i].x, &pnt[i].y);
60         ans=0;
61         for (i=0; i<n; ++i)
62         {
63             alpha.resize(0);
64             for (j=0; j<n; ++j)
65                 if (i!=j)
66                 {
67                     if ((d=(pnt[i]-pnt[j]).len())>R)
68                         continue;
69                     if ((theta=atan2(pnt[j].y-pnt[i].y, pnt[j].x-pnt[i].x)
70                         )<0)
71                         theta+=2*pi;
72                     phi=acos(d/R);
73                     alpha.push_back(node(theta-phi, true));
74                     alpha.push_back(node(theta+phi, false));
75                 }
76             std::sort(alpha.begin(), alpha.end());
77             for (j=0; j<alpha.size(); ++j)
78             {
79                 if (alpha[j].flag)
80                     ++sum;
81                 else
82                     --sum;
83                 ans+=std::max(ans, sum);
84             }

```

```

85     printf("%hd\n",ans+1);
86 }
87 return 0;
88 }
89 //
90 //
91 #include<stdio>
92 #include<cmath>
93
94 #define MAXX 511
95 #define eps 1e-8
96
97 struct pv
98 {
99     double x,y;
100     pv() {}
101     pv(const double &xx,const double &yy):x(xx),y(yy) {}
102     inline pv operator-(const pv &i) const
103     {
104         return pv(x-i.x,y-i.y);
105     }
106     inline pv operator+(const pv &i) const
107     {
108         return pv(x+i.x,y+i.y);
109     }
110     inline double cross(const pv &i) const
111     {
112         return x*i.y-y*i.x;
113     }
114     inline double len()
115     {
116         return sqrt(x*x+y*y);
117     }
118     inline pv operator/(const double &a) const
119     {
120         return pv(x/a,y/a);
121     }
122     inline pv operator*(const double &a) const
123     {
124         return pv(x*a,y*a);
125     }
126 }
127 pnt[MAXX],o,tl,lt,aa,bb,cc,dd;
128
129 short n,i,j,k,l;
130 double r,u;
131
132 inline pv ins(const pv &a1,const pv &a2,const pv &b1,const pv &b2)
133 {
134     tl=a2-a1;
135     lt=b2-b1;
136     u=(b1-a1).cross(lt)/(tl).cross(lt);
137     return a1+tl*u;
138 }
139
140 inline pv get(const pv &a,const pv &b,const pv &c)
141 {
142     aa=(a+b)/2;
143     bb.x=aa.x-a.y+b.y;
144     bb.y=aa.y+a.x-b.x;
145     cc=(a+c)/2;
146     dd.x=cc.x-a.y+c.y;
147     dd.y=cc.y+a.x-c.x;
148     return ins(aa,bb,cc,dd);
149 }
150
151 int main()
152 {
153     while(scanf("%hd",&n),n)
154     {
155         for(i=0;i<n;++i)
156             scanf("%lf %lf",&pnt[i].x,&pnt[i].y);
157         o=pnt[0];
158         r=0;
159         for(i=1;i<n;++i)
160             if((pnt[i]-o).len()>r+eps)
161             {
162                 o=pnt[i];
163                 r=0;
164                 for(j=0;j<i;++j)
165                     if((pnt[j]-o).len()>r+eps)
166                     {
167                         o=(pnt[i]+pnt[j])/2;
168                         r=(o-pnt[j]).len();
169                         for(k=0;k<j;++k)
170                             if((o-pnt[k]).len()>r+eps)
171                             {
172                                 o=get(pnt[i],pnt[j],pnt[k]);
173                                 r=(o-pnt[i]).len();
174                             }
175                     }
176             }
177         printf("%.2lf %.2lf %.2lf\n",o.x,o.y,r);
178     }
179     return 0;
180 }
181
182 //
183 double dis(int x,int y)
184 {
185     return sqrt((double)(x*x+y*y));
186 }
187
188 double area(int x1,int y1,int x2,int y2,double r1,double r2)
189 {
190     double s=dis(x2-x1,y2-y1);
191     if(r1+r2<s) return 0;
192     else if(r2-r1>s) return PI*r1*r1;
193     else if(r1-r2>s) return PI*r2*r2;
194     double q1=acos((r1*r1+s*s-r2*r2)/(2*r1*s));
195     double q2=acos((r2*r2+s*s-r1*r1)/(2*r2*s));
196     return (r1*r1*q1+r2*r2*q2-r1*s*sin(q1));
197 }
198
199 //
200 {
201     for(int i = 0; i < 3; i++)
202         scanf("%lf%lf",&p[i].x,&p[i].y);
203     tp = pv(p[0].x+p[1].x)/2,(p[0].y+p[1].y)/2;
204     l[0] = Line(tp,pv(tp.x-(p[1].y-p[0].y),tp.y+(p[1].x-p[0].x)));

```

```

205     tp = pv((p[0].x+p[2].x)/2,(p[0].y+p[2].y)/2);
206     l[1] = Line(tp,pv(tp.x-(p[2].y-p[0].y),tp.y+(p[2].x-p[0].x)));
207     tp = LineToLine(l[0],l[1]);
208     r = pv(tp,p[0]).Length();
209     printf("%.6f,%.6f,%.6f\n",tp.x,tp.y,r);
210 }
211 //
212 {
213     for(int i = 0; i < 3; i++)
214         scanf("%lf%lf",&p[i].x,&p[i].y);
215     if(xmult(pv(p[0],p[1]),pv(p[0],p[2])) < 0)
216         swap(p[1],p[2]);
217     for(int i = 0; i < 3; i++)
218         len[i] = pv(p[i],p[(i+1)%3]).Length();
219     tr = (len[0]+len[1]+len[2])/2;
220     r = sqrt((tr-len[0])*(tr-len[1])*(tr-len[2])/tr);
221     for(int i = 0; i < 2; i++)
222     {
223         v = pv(p[i],p[i+1]);
224         tv = pv(-v.y,v.x);
225         tr = tv.Length();
226         tv = pv(tv.x*tr/tv.y*r/tr);
227         tp = pv(p[i].x+tv.x,p[i].y+tv.y);
228         l[i].s = tp;
229         tp = pv(p[i+1].x+tv.x,p[i+1].y+tv.y);
230         l[i].e = tp;
231     }
232     tp = LineToLine(l[0],l[1]);
233     printf("%.6f,%.6f,%.6f\n",tp.x,tp.y,r);
234 }
235

```

3.6 closest point pair

```

1 //
2
3 struct Point {double x, y;} p[10], t[10];
4 bool cmpx(const Point& i, const Point& j) {return i.x < j.x;}
5 bool cmpy(const Point& i, const Point& j) {return i.y < j.y;}
6
7 double DnC(int L, int R)
8 {
9     if (L >= R) return le9; //
10
11     /* Divide */
12
13     int M = (L + R) / 2;
14
15     /* Conquer */
16
17     double d = min(DnC(L,M), DnC(M+1,R));
18     // if (d == 0.0) return d; //
19
20     /* MergeYO(NlogN) */
21
22     int N = 0; //
23     for (int i=M; i>=L && p[M].x - p[i].x < d; --i) t[N++] = p[i];
24     for (int i=M+1; i<=R && p[i].x - p[M].x < d; ++i) t[N++] = p[i];
25     sort(t, t+N, cmpy); // Quicksort O(NlogN)
26
27     /* MergeO(N) */
28
29     for (int i=0; i<N-1; ++i)
30         for (int j=1; j<=2 && i+j<N; ++j)
31             d = min(d, distance(t[i], t[i+j]));
32
33     return d;
34 }
35
36 double closest_pair()
37 {
38     sort(p, p+10, cmpx);
39     return DnC(0, N-1);
40 }
41
42 //2
43
44 struct Point {double x, y;} p[10], t[10];
45 bool cmpx(const Point& i, const Point& j) {return i.x < j.x;}
46 bool cmpy(const Point& i, const Point& j) {return i.y < j.y;}
47
48 double DnC(int L, int R)
49 {
50     if (L >= R) return le9; //
51
52     /* Divide */
53
54     int M = (L + R) / 2;
55
56     // X
57     double x = p[M].x;
58
59     /* Conquer */
60
61     // Y
62     double d = min(DnC(L,M), DnC(M+1,R));
63     // if (d == 0.0) return d; //
64
65     /* MergeYO(N) */
66
67     // Y
68     int N = 0; //
69     for (int i=0; i<M; ++i)
70         if (x - p[i].x < d)
71             t[N++] = p[i];
72
73     // Y
74     int P = N; // P
75     for (int i=M+1; i<=R; ++i)
76         if (p[i].x - x < d)
77             t[N++] = p[i];
78
79

```

```

80 // YMerge Sort
81 inplace_merge(t, t+p, t+N, cmpy);
82
83 /* MergeO(N) */
84
85 for (int i=0; i<N; ++i)
86     for (int j=1; j<=2 && i+j<N; ++j)
87         d = min(d, distance(t[i], t[i+j]));
88
89 /* MergeYO(N) */
90
91 // Merge Sort
92 inplace_merge(p+L, p+M+1, p+R+1, cmpy);
93
94 return d;
95 }
96
97 double closest_pair()
98 {
99     sort(p, p+10, cmpx);
100     return DnC(0, N-1);
101 }
102
103 //mzry
104 //
105 double calc_dis(Point &a, Point &b) {
106     return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
107 }
108 //
109 bool operator<(const Point &a, const Point &b) {
110     if(a.y != b.y) return a.x < b.x;
111     return a.x < b.x;
112 }
113 double Gao(int l, int r, Point pnts[]) {
114     double ret = inf;
115     if(l == r) return ret;
116     if(l+1 == r) {
117         ret = min(calc_dis(pnts[l], pnts[l+1]), ret);
118         return ret;
119     }
120     if(l+2 == r) {
121         ret = min(calc_dis(pnts[l], pnts[l+1]), ret);
122         ret = min(calc_dis(pnts[l], pnts[l+2]), ret);
123         ret = min(calc_dis(pnts[l+1], pnts[l+2]), ret);
124         return ret;
125     }
126
127     int mid = l+r>>1;
128     ret = min(ret, Gao(l, mid, pnts));
129     ret = min(ret, Gao(mid+1, r, pnts));
130
131     for(int c = l; c<=r; c++)
132         for(int d = c+1; d <=c+7 && d<=r; d++) {
133             ret = min(ret, calc_dis(pnts[c], pnts[d]));
134         }
135     return ret;
136 }
137
138 //
139 #include <iostream>
140 #include <cstdio>
141 #include <cstring>
142 #include <map>
143 #include <vector>
144 #include <cmath>
145 #include <algorithm>
146 #define Point pair<double, double>
147 using namespace std;
148
149 const int step[9][2] =
150     {{-1,-1},{-1,0},{-1,1},{0,-1},{0,0},{0,1},{1,-1},{1,0},{1,1}};
151
152 int n,x,y,nx,ny;
153 map<pair<int,int>,vector<Point>>> g;
154 vector<Point> tmp;
155 Point p[20000];
156 double tx,ty,ans,nowans;
157 vector<Point>::iterator it,op,ed;
158 pair<int,int> gird;
159 bool flag;
160
161 double Dis(Point p0,Point p1)
162 {
163     return sqrt((p0.first-p1.first)*(p0.first-p1.first)+
164         (p0.second-p1.second)*(p0.second-p1.second));
165 }
166
167 double CalcDis(Point p0,Point p1,Point p2)
168 {
169     return Dis(p0,p1)+Dis(p0,p2)+Dis(p1,p2);
170 }
171
172 void build(int n,double w)
173 {
174     g.clear();
175     for (int i = 0; i < n; i++)
176         g[make_pair((int)floor(p[i].first/w), (int)floor(p[i].second/w))]
177             .push_back(p[i]);
178 }
179
180 int main()
181 {
182     int t;
183     scanf("%d",&t);
184     for (int ft = 1; ft <= t; ft++)
185     {
186         scanf("%d",&n);
187         for (int i = 0; i < n; i++)
188         {
189             scanf("%lf%lf",&tx,&ty);
190             p[i] = make_pair(tx,ty);
191         }
192         random_shuffle(p,p+n);
193         ans = CalcDis(p[0],p[1],p[2]);
194         build(3,ans/2.0);
195         for (int i = 3; i < n; i++)
196         {
197             x = (int)floor(2.0*p[i].first/ans);
198             y = (int)floor(2.0*p[i].second/ans);
199             tmp.clear();

```

```

197     for (int k = 0; k < 9; k++)
198     {
199         nx = x+step[k][0];
200         ny = y+step[k][1];
201         gird = make_pair(nx,ny);
202         if (g.find(gird) != g.end())
203         {
204             op = g[gird].begin();
205             ed = g[gird].end();
206             for (it = op; it != ed; it++)
207                 tmp.push_back(*it);
208         }
209     }
210     flag = false;
211     for (int j = 0; j < tmp.size(); j++)
212         for (int k = j+1; k < tmp.size(); k++)
213         {
214             nowans = CalcDis(p[i],tmp[j],tmp[k]);
215             if (nowans < ans)
216             {
217                 ans = nowans;
218                 flag = true;
219             }
220         }
221     if (flag == true)
222         build(i+1,ans/2.0);
223     else
224         g[make_pair((int)floor(2.0*p[i].first/ans), (int)floor(2.0*p[i].second/ans))]
225             .push_back(p[i]);
226     printf("%.3f\n",ans);
227 }
228 }

```

3.7 ellipse

```

1 sq(x-h)/sq(q) + sq(y-k)/sq(b) = 1
2
3 x=h+a*cos(t);
4 y=k+b*sin(t);
5
6 area: pi*a*b;
7 distance from center to focus: f=sqrt(sq(a)-sq(b));
8 eccentricity: e=sqrt(a-sq(b/a))=f/a;
9 focal parameter: sq(b)/sqrt(sq(a)-sq(b))=sq(b)/f;
10
11 double circumference(double a,double b) // accuracy: pow(0.5,53);
12 {
13     double x=a;
14     double y=b;
15     if(x<y)
16         std::swap(x,y);
17     double digits=53,tol=sqrt(pow(0.5,digits));
18     if(digits*y<tol*x)
19         return 4*x;
20     double s=0,m=1;
21     while(x>(tol+1)*y)
22     {
23         double tx=x;
24         double ty=y;
25         x=0.5f*(tx+ty);
26         y=sqrt(tx*ty);
27         m*=2;
28         s+=m*pow(x-y,2);
29     }
30     return pi*(pow(a+b,2)-s)/(x+y);
31 }

```

3.8 Graham's scan

```

1 pv pnt[MAXX];
2
3 inline bool com(const pv &a,const pv &b)
4 {
5     if(fabs(t=(a-pnt[0]).cross(b-pnt[0]))>eps)
6         return t>0;
7     return (a-pnt[0]).len()<(b-pnt[0]).len();
8 }
9
10 inline void graham(std::vector<pv> &ch,const int n)
11 {
12     std::nth_element(pnt,pnt,pnt+n);
13     std::sort(pnt+1,pnt+n,com);
14     ch.resize(0);
15     ch.push_back(pnt[0]);
16     ch.push_back(pnt[1]);
17     static int i;
18     for(i=2;i<n;++i)
19         if(fabs((pnt[i]-ch[0]).cross(ch[1]-ch[0]))>eps)
20         {
21             ch.push_back(pnt[i]);
22             break;
23         }
24     else
25         ch.back()=pnt[i];
26     for(;i<n;++i)
27     {
28         while((ch.back()-ch[ch.size()-2]).cross(pnt[i]-ch[ch.size()-2])<eps)
29             ch.pop_back();
30         ch.push_back(pnt[i]);
31     }
32 }

```


3.9 half-plane intersection

```

1 //abc
2 inline pv ins(const pv &p1,const pv &p2)
3 {
4     u=fabs(a*p1.x+b*p1.y+c);
5     v=fabs(a*p2.x+b*p2.y+c);
6     return pv((p1.x+v*p2.x*u)/(u+v), (p1.y+v*p2.y*u)/(u+v));
7 }
8
9 inline void get(const pv& p1,const pv& p2,double &a,double &b,
10 double &c)
11 {
12     a=p2.y-p1.y;
13     b=p1.x-p2.x;
14     c=p2.x*p1.y-p2.y*p1.x;
15 }
16 inline pv ins(const pv &x,const pv &y)
17 {
18     get(x,y,d,e,f);
19     return pv((b*f-c*e)/(a*e-b*d), (a*f-c*d)/(b*d-a*e));
20 }
21
22 std::vector<pv> p[2];
23 inline bool go()
24 {
25     k=0;
26     p[k].resize(0);
27     p[k].push_back(pv(-inf,inf));
28     p[k].push_back(pv(-inf,-inf));
29     p[k].push_back(pv(inf,-inf));
30     p[k].push_back(pv(inf,inf));
31     for(i=0;i<n;++i)
32     {
33         get(pnt[i],pnt[(i+1)%n],a,b,c);
34         c+=the*sqrt(a*a+b*b);
35         p[k].resize(0);
36         for(l=0;l<p[k].size();++l)
37             if(a*p[k][l].x+b*p[k][l].y+c<eps)
38                 p[k].push_back(p[k][l]);
39         else
40         {
41             m=(l+p[k].size()-1)*p[k].size();
42             if(a*p[k][m].x+b*p[k][m].y+c<-eps)
43                 p[k].push_back(ins(p[k][m],p[k][l]));
44             m=(l+1)*p[k].size();
45             if(a*p[k][m].x+b*p[k][m].y+c<-eps)
46                 p[k].push_back(ins(p[k][m],p[k][l]));
47         }
48         k=!k;
49         if(p[k].empty())
50             break;
51     }
52     //p[k]
53     return p[k].empty();
54 }
55 //
56 //
57 //
58 inline pv ins(const pv &a,const pv &b)
59 {
60     u=fabs(ln.cross(a-pnt[i]));
61     v=fabs(ln.cross(b-pnt[i]));u;
62     t1=b-a;
63     return pv(u*t1.x/v+a.x,u*t1.y/v+a.y);
64 }
65 }
66
67 int main()
68 {
69     j=0;
70     for(i=0;i<n;++i)
71     {
72         ln=pnt[(i+1)%n]-pnt[i];
73         p[!j].resize(0);
74         for(k=0;k<p[j].size();++k)
75             if(ln.cross(p[j][k]-pnt[i])<=0)
76                 p[!j].push_back(p[j][k]);
77         else
78         {
79             l=(k-l+p[j].size())*p[j].size();
80             if(ln.cross(p[j][l]-pnt[i])<0)
81                 p[!j].push_back(ins(p[j][k],p[j][l]));
82             l=(k+1)*p[j].size();
83             if(ln.cross(p[j][l]-pnt[i])<0)
84                 p[!j].push_back(ins(p[j][k],p[j][l]));
85         }
86         j=!j;
87     }
88     //p[j]
89 }
90 //mrzy
91
92 bool HPIcmp(Line a, Line b)
93 {
94     if (fabs(a.k - b.k) > eps)
95         return a.k < b.k;
96     return ((a.s - b.s) * (b.e - b.s)) < 0;
97 }
98
99 Line Q[100];
100
101 void HPI(Line line[], int n, Point res[], int &resn)
102 {
103     int tot = n;
104     std::sort(line, line + n, HPIcmp);
105     tot = 1;
106     for (int i = 1; i < n; i++)
107         if (fabs(line[i].k - line[i - 1].k) > eps)
108             line[tot++] = line[i];
109     int head = 0, tail = 1;
110     Q[0] = line[0];
111     Q[1] = line[1];
112     resn = 0;
113     for (int i = 2; i < tot; i++)
114     {

```

```

116         if (fabs(Q[tail].e-Q[tail].s)*Q[tail - 1].e-Q[tail - 1].s)
117             < eps || fabs((Q[head].e-Q[head].s)*Q[head + 1].e-
118             Q[head + 1].s) < eps)
119             return;
120         while (head < tail && (((Q[tail]&Q[tail - 1]) - line[i].s) *
121             (line[i].e-line[i].s) > eps)
122             --tail;
123         while (head < tail && (((Q[head]&Q[head + 1]) - line[i].s) *
124             (line[i].e-line[i].s) > eps)
125             ++head;
126         Q[++tail] = line[i];
127     }
128     while (head < tail && (((Q[tail]&Q[tail - 1]) - Q[head].s) * (Q
129         [head].e-Q[head].s) > eps)
130         tail--;
131     while (head < tail && (((Q[head]&Q[head + 1]) - Q[tail].s) * (Q
132         [tail].e-Q[tail].s) > eps)
133         head++;
134     if (tail <= head + 1)
135         return;
136     for (int i = head; i < tail; i++)
137         res[resn++] = Q[i] & Q[i + 1];
138     if (head < tail + 1)
139         res[resn++] = Q[head] & Q[tail];
140 }

```

3.10 kdtree

```

1 #include <iostream>
2 #include <cstdio>
3 #include <cstdlib>
4 #include <algorithm>
5 #include <stack>
6 #include <algorithm>
7 using namespace std;
8 #define MAXN 100010
9 typedef long long ll;
10 struct Point{
11     ll x,y;
12     void operator =(const Point &p){
13         x=p.x; y=p.y;
14     }
15     ll dis(const Point &a){
16         return (x-a.x)*(x-a.x)+(y-a.y)*(y-a.y);
17     }
18 }point[MAXN],pp[MAXN];
19
20 struct Node{
21     int split;//{0,1} 0xly
22     Point p;//
23     tree[MAXN*4];
24 }
25 bool cmpx(const Point &a,const Point &b)
26 {
27     return a.x<b.x;
28 }
29 bool cmpy(const Point &a,const Point &b)
30 {
31     return a.y<b.y;
32 }
33
34 void initTree(int x,int y,int split,int pos)
35 {
36     if(y<x) return ;
37     int mid=(x+y)>>1;
38     random_shuffle(point+x,point+y);
39     if(split==0) nth_element(point+x,point+mid,point+y+1,cmpx);
40     else nth_element(point+x,point+mid,point+y+1,cmpy);
41     tree[pos].split=split;
42     tree[pos].p=point[mid];
43     initTree(x,mid-1,(split^1),2*pos);
44     initTree(mid+1,y,(split^1),2*pos+1);
45 }
46
47 ll ans;
48 void insert(int x,int y,Point &p,int pos)
49 {
50     if(y<x) return ;
51     int mid=(x+y)>>1;
52     ll temp=p.dis(tree[pos].p);
53     if(temp!=0) ans=min(ans,temp);
54     if(tree[pos].split==0){
55         if(p.x<tree[pos].p.x){
56             insert(x,mid-1,p,2*pos);
57             if(ans>=(p.x-tree[pos].p.x)*(p.x-tree[pos].p.x))
58                 insert(mid+1,y,p,2*pos+1);
59         }
60         else{
61             insert(mid+1,y,p,2*pos+1);
62             if(ans>=(p.x-tree[pos].p.x)*(p.x-tree[pos].p.x))
63                 insert(x,mid-1,p,2*pos);
64         }
65     }
66     else{
67         if(p.y<tree[pos].p.y){
68             insert(x,mid-1,p,2*pos);
69             if(ans>=(p.y-tree[pos].p.y)*(p.y-tree[pos].p.y))
70                 insert(mid+1,y,p,2*pos+1);
71         }
72         else{
73             insert(mid+1,y,p,2*pos+1);
74             if(ans>=(p.y-tree[pos].p.y)*(p.y-tree[pos].p.y))
75                 insert(x,mid-1,p,2*pos);
76         }
77     }
78 }
79
80 int main()
81 {
82     int cases,n;
83     scanf("%d",&cases);
84     while(cases--){

```

```

87     {
88         scanf("%d",&n);
89         for(int i=1;i<=n;i++){
90             scanf("%I64d%I64d",&pp[i].x,&pp[i].y);
91             point[i]=pp[i];
92         }
93         initTree(1,n,0,1);
94         for(int i=1;i<=n;i++){
95             ans=1LL<<62;
96             insert(1,n,pp[i],1);
97             printf("%I64d\n",ans);
98         }
99     }
100     return 0;
101 }

```

3.11 Manhattan minimum spanning tree

```

1  #include <cstdio>
2  #include <algorithm>
3  #include <cstring>
4  #include <iostream>
5
6  using namespace std;
7
8  const int maxn = 60000;
9
10 struct node {int x, y, k[2];} b[maxn];
11 struct bian {int a, b, c;} g[maxn * 8];
12 struct point {int k[2];} d[maxn * 8];
13 long long s[maxn], ans;
14 int i, n, m, a[maxn], lim, h, mid, bh[maxn * 2], f[maxn], num, e[
15     maxn * 2], next[maxn * 2], first[maxn], tot;
16 int comx(int p, int q) {return b[p].x < b[q].x;}
17 int comy(int p, int q) {return b[p].y < b[q].y;}
18 int comc(const bian &p, const bian &q) {return p.c < q.c;}
19 int dist(int p, int q) {return abs(b[p].x - b[q].x) + abs(b[p].y -
20     b[q].y);}
21 int maxbh(int p, int q, int k) {return b[p].k[k] > b[q].k[k] ? p :
22     q;}
23 int minbh(int p, int q, int k) {return b[p].k[k] < b[q].k[k] ? p :
24     q;}
25 int getfa(int x) {if (f[x] != x) f[x] = getfa(f[x]); return f[x];}
26 long long gcd(long long p, long long q) {return (!p || !q) ? p + q
27     : gcd(q, p % q);}
28 void link(int u, int v)
29 {
30     e[++num] = v, next[num] = first[u], first[u] = num;
31     e[++num] = u, next[num] = first[v], first[v] = num;
32 }
33 void add(int x, int k)
34 {
35     int y = h + b[x].k[1]; d[y].k[0] = minbh(d[y].k[0], x, 0);
36     for (y >= 1; y; y >= 1) d[y].k[0] = minbh(d[y] < 1].k[0], d[y
37         < 1 ^ 1].k[0], 0);
38     y = h + b[x].k[0];
39     d[y].k[1] = k ? maxbh(x, d[y].k[1], 1) : minbh(d[y].k[1], x, 1)
40         ;
41     for (y >= 1; y; y >= 1)
42         d[y].k[1] = k ? maxbh(x, d[y] < 1].k[1], 1) : minbh(d[y] < 1
43             ^ 1].k[1], x, 1);
44 }
45 int ask(int l, int r, int k, int boss)
46 {
47     for (mid = 0, l += h - 1, r += h + 1; (l ^ r) != 1; l >= 1, r
48         >= 1)
49     {
50         if (!(l & 1)) mid = boss ? maxbh(mid, d[l + 1].k[k], k) :
51             minbh(mid, d[r - 1].k[k], k);
52         if (r & 1) mid = boss ? maxbh(mid, d[r - 1].k[k], k) : minbh
53             (mid, d[r - 1].k[k], k);
54     } return mid;
55 }
56 void manhattan()
57 {
58     sort(bh + 1, bh + m + 1, comx);
59     b[0].k[0] = maxn * 3, b[0].k[1] = -1;
60     for (add(bh[m], 1), i = m - 1; i; add(bh[i], 1), --i)
61     {
62         g[++tot].a = bh[i], g[tot].b = ask(b[bh[i]].k[1], lim, 0, 0)
63             ;
64         g[tot].c = dist(g[tot].a, g[tot].b);
65         if (g[tot].b == 0) --tot;
66         g[++tot].a = bh[i], g[tot].b = ask(1, b[bh[i]].k[0], 1, 1);
67         g[tot].c = dist(g[tot].a, g[tot].b);
68         if (g[tot].b == 0) --tot;
69     }
70     b[0].k[1] = b[0].k[0];
71     memset(d, 0, sizeof(d));
72     sort(bh + 1, bh + m + 1, comy);
73     for (add(bh[m], 0), i = m - 1; i; add(bh[i], 0), --i)
74     {
75         g[++tot].a = bh[i], g[tot].b = ask(1, b[bh[i]].k[1], 0, 0);
76         g[tot].c = dist(g[tot].a, g[tot].b);
77         if (g[tot].b == 0) --tot;
78         g[++tot].a = bh[i], g[tot].b = ask(1, b[bh[i]].k[0], 1, 0);
79         g[tot].c = dist(g[tot].a, g[tot].b);
80         if (g[tot].b == 0) --tot;
81     }
82 }
83 void kruskal()
84 {
85     sort(g + 1, g + tot + 1, comc);
86     for (i = 1; i <= tot; ++i)
87     {
88         int f1 = getfa(g[i].a), f2 = getfa(g[i].b);
89         if (f1 != f2) link(g[i].a, g[i].b), f[f1] = f2;
90     } tot = 0; memset(f, 0, sizeof(f));
91 }
92 void dfs(int x, int fa)
93 {
94     bh[++tot] = x;

```

```

83     for (int p = first[x]; p; p = next[p])
84         if (e[p] != fa) dfs(e[p], x), bh[++tot] = x;
85 }
86 void del(int l, int r)
87 {
88     if (l > r) return ;
89     for (int j = l; j <= r; ++j)
90         ans -= 1LL * f[a[j]] * f[a[j]], ans += 1LL * (--f[a[j]]) * f
91             [a[j]];
92 }
93 void ins(int l, int r)
94 {
95     if (l > r) return ;
96     for (int j = l; j <= r; ++j)
97         ans -= 1LL * f[a[j]] * f[a[j]], ans += 1LL * (++f[a[j]]) * f
98             [a[j]];
99 }
100 int main()
101 {
102     freopen("hose.in", "r", stdin);
103     freopen("hose.out", "w", stdout);
104     scanf("%d%d", &n, &m);
105     for (i = 1; i <= n; ++i)
106         scanf("%d", &a[i]);
107     for (i = 1; i <= m; ++i)
108     {
109         scanf("%d%d", &b[bh[i]] = f[i] = i].x, &b[i].y);
110         b[i].k[0] = b[i].x + b[i].y;
111         b[i].k[1] = b[i].y - b[i].x + maxn;
112         lim = max(lim, max(b[i].k[0], b[i].k[1]));
113     }
114     for (h = 1; h <= lim; h <= 1);
115     manhattan();
116     kruskal();
117     dfs(1, 0);
118     ins(b[bh[1]].x, b[bh[1]].y);
119     for (s[1] = ans, i = 2; i <= tot; s[bh[i]] = ans, ++i)
120     {
121         ins(b[bh[i]].x, b[bh[i - 1]].x - 1);
122         ins(b[bh[i - 1]].y + 1, b[bh[i]].y);
123         del(b[bh[i - 1]].x, b[bh[i]].x - 1);
124         del(b[bh[i]].y + 1, b[bh[i - 1]].y);
125     }
126     for (i = 1; i <= m; ++i)
127     {
128         long long fz = s[i] - b[i].k[1] - 1 + maxn, fm = 1LL * (b[i
129             ].k[1] + 1 - maxn) * (b[i].k[1] - maxn);
130         long long gys = gcd(fz, fm);
131         printf("%lld/%lld\n", fz/gys, fm/gys);
132     }
133     return 0;
134 }
135
136 #include<iostream>
137 #include<cstdio>
138 #include<algorithm>
139 #include<cmath>
140 #include<cstring>
141 #define maxn 55000
142 #define inf 2147483647
143 using namespace std;
144 struct query
145 {
146     int l,r,s,w;
147 }a[maxn];
148 int c[maxn];
149 long long col[maxn],size[maxn],ans[maxn];
150 int n,m,cnt,len;
151
152 long long gcd(long long x,long long y)
153 {
154     return (!x)?y:gcd(y%x,x);
155 }
156
157 bool cmp(query a,query b)
158 {
159     return (a.w==b.w)?a.r<b.r:a.w<b.w;
160 }
161
162 int main()
163 {
164     //freopen("hose.in","r",stdin);
165     scanf("%d%d",&n,&m);
166     for (int i=1;i<=n;i++) scanf("%d",&c[i]);
167     len=(int)sqrt(m);
168     cnt=(len+len==m)?len:len+1;
169     for (int i=1;i<=m;i++)
170     {
171         scanf("%d%d",&a[i].l,&a[i].r);
172         if (a[i].l>a[i].r) swap(a[i].l,a[i].r);
173         size[i]=a[i].r-a[i].l+1;
174         a[i].w=a[i].l/len+1;
175         a[i].s=i;
176     }
177     sort(a+1,a+m+1,cmp);
178     int i=1;
179     while (i<=m)
180     {
181         int now=a[i].w;
182         memset(col,0,sizeof(col));
183         for (int j=a[i].l;j<=a[i].r;j++) ans[a[i].s]+=2*(col[c[j]
184             ]++));
185         i++;
186         for (;a[i].w==now;i++)
187         {
188             ans[a[i].s]=ans[a[i-1].s];
189             for (int j=a[i-1].r+1;j<=a[i].r;j++)
190                 ans[a[i].s]+=2*(col[c[j]]++);
191             if (a[i-1].l<a[i].l)
192                 for (int j=a[i-1].l;j<a[i].l;j++)
193                     ans[a[i].s]-=2*(--col[c[j]]);
194             else
195                 for (int j=a[i].l;j<a[i-1].l;j++)
196                     ans[a[i].s]+=2*(col[c[j]]++);
197         }
198     }

```

```

199     }
200   }
201   long long all,num;
202   for (int i=1;i<=m;i++)
203   {
204       if (size[i]==1) all=1; else all=size[i]*(size[i]-1);
205       num=gcd(ans[i],all);
206       printf("%lld/%lld\n",ans[i]/num,all/num);
207   }
208   return 0;
209 }

```

3.12 others

```

1  eps
2
3  sqrt(a), asin(a), acos(a) aa0-1e-12sqrt(a)0aal,asin(a)acos(a)a
4
5  case0:005,0:010:0050000000001()0:004999999999()printf("%.21f", a)
6  aa + eps, a - eps
7
8  -0.000
9
10 double
11
12 a==b fabs(a-b)<eps
13 a!=b fabs(a-b)>eps
14 a<b a+eps<b
15 a<=b a<b+eps
16 a>b a>b+eps
17 a>=b a+eps>b
18
19
20
21 cos/sin/tan
22 acos [-1,+1][0,]
23 asin [-1,+1][-/2,+/2]
24 atan [-/2,+/2]
25 atan2 (y,x)(),tan(y/x),[-,+]xy
26
27 other
28
29 log (ln)
30 log10
31 ceil
32 floor
33
34 round
35
36 cpp:
37 java: add 0.5,then floor
38 cpp:
39 4
40 6
41 5050
42 55050
43
44 rotate mat:
45 [ cos(theta) -sin(theta) ]
46 [ sin(theta) cos(theta) ]

```

3.13 Pick's theorem

```

1
2
3 A:
4 i:
5 b:
6 A = i + b/2 - 1
7
8
9 A = 2i + b - 2

```

3.14 PointInPoly

```

1 /*
2 ,
3 poly3
4
5 0 -- poly
6 1 -- poly
7 2 -- poly
8 */
9
10 int inPoly(pv p,pv poly[], int n)
11 {
12     int i, count;
13     Line ray, side;
14
15     count = 0;
16     ray.s = p;
17     ray.e.y = p.y;
18     ray.e.x = -1; //-INF
19
20     for (i = 0; i < n; i++)
21     {
22         side.s = poly[i];
23         side.e = poly[(i+1)%n];
24
25         if(OnSeg(p, side))
26             return 1;
27
28         // side x

```

```

29     if (side.s.y == side.e.y)
30         continue;
31
32     if (OnSeg(side.s, ray))
33     {
34         if (side.s.y > side.e.y)
35             count++;
36     }
37     else
38     if (OnSeg(side.e, ray))
39     {
40         if (side.e.y > side.s.y)
41             count++;
42     }
43     else
44     if (inter(ray, side))
45         count++;
46
47     return ((count % 2 == 1) ? 0 : 2);
48 }

```

3.15 rotating caliper

```

1 //
2
3 inline double go()
4 {
5     l=ans=0;
6     for(i=0;i<n;++i)
7     {
8         t1=pnt[(i+1)%n]-pnt[i];
9         while(abs(t1.cross(pnt[(l+1)%n]-pnt[i]))>abs(t1.cross(pnt[l]-pnt[i])))
10             l=(l+1)%n;
11         ans=std::max(ans, std::max(dist(pnt[l],pnt[i]),dist(pnt[l],pnt[(i+1)%n])));
12     }
13     return ans;
14 }
15
16 //
17 double go()
18 {
19     sq=sp=0;
20     for(i=1;i<ch[1].size();++i)
21         if(ch[1][sq]<ch[1][i])
22             sq=i;
23     tpv=sp;
24     tq=sq;
25     ans=(ch[0][sp]-ch[1][sq]).len();
26     do
27     {
28         a1=ch[0][sp];
29         a2=ch[0][(sp+1)%ch[0].size()];
30         b1=ch[1][sq];
31         b2=ch[1][(sq+1)%ch[1].size()];
32         tpv=b1-(b2-a1);
33         tpv.x = b1.x - (b2.x - a1.x);
34         tpv.y = b1.y - (b2.y - a1.y);
35         len=(tpv-a1).cross(a2-a1);
36         if(fabs(len)<eps)
37         {
38             ans=std::min(ans,p2l(a1,b1,b2));
39             ans=std::min(ans,p2l(a2,b1,b2));
40             ans=std::min(ans,p2l(b1,a1,a2));
41             ans=std::min(ans,p2l(b2,a1,a2));
42             sp=(sp+1)%ch[0].size();
43             sq=(sq+1)%ch[1].size();
44         }
45     }
46     else
47     if(len<-eps)
48     {
49         ans=std::min(ans,p2l(b1,a1,a2));
50         sp=(sp+1)%ch[0].size();
51     }
52     else
53     {
54         ans=std::min(ans,p2l(a1,b1,b2));
55         sq=(sq+1)%ch[1].size();
56     }
57 }while(tp!=sp || tq!=sq);
58 return ans;
59 }
60
61 // by mzry
62 inline void solve()
63 {
64     resa = resb = 1e100;
65     double dis1,dis2;
66     Point xp[4];
67     Line l[4];
68     int a,b,c,d;
69     int sa,sb,sc,sd;
70     a = b = c = d = 0;
71     sa = sb = sc = sd = 0;
72     Point va,vb,vc,vd;
73     for (a = 0; a < n; a++)
74     {
75         va = Point(p[a],p[(a+1)%n]);
76         vb = Point(-va.x,-va.y);
77         vb = Point(-va.y,va.x);
78         vd = Point(-vb.x,-vb.y);
79         if (sb < sa)
80         {
81             b = a;
82             sb = sa;
83         }
84         while (xmult(vb,Point(p[b],p[(b+1)%n])) < 0)
85             b = (b+1)%n;
86         sb++;
87     }
88     if (sc < sb)
89     {

```

```

90         c = b;
91         sc = sb;
92     }
93     while (xmult(vc,Point(p[c],p[(c+1)%n])) < 0)
94     {
95         c = (c+1)%n;
96         sc++;
97     }
98     if (sd < sc)
99     {
100         d = c;
101         sd = sc;
102     }
103     while (xmult(vd,Point(p[d],p[(d+1)%n])) < 0)
104     {
105         d = (d+1)%n;
106         sd++;
107     }
108     //`p[a],p[b],p[c],p[d]`
109     sa++;
110 }
111 }
112 }
113
114 //
115 P = { p(1) , ... , p(m) } Q = { q(1) , ... , q(n) } (p(i), q(j))
      P Q
116
117 (p(i), q(j))
118 p(i-1), p(i+1), q(j-1), q(j+1) (p(i), q(j))
119
120
121
122 1 P Q y x
123 2 x
124 3 (p(i), q(j))
125 4 (p(i), q(j)) p(i-1), p(i+1), q(j-1), q(j+1) (p(i), q(j))
126 534
127 6
128
129 156 O(N) N
130
131
132
133 //
134 1 P y yminP Q y ymaxQ
135 2 yminP ymaxQ LP LQ LP LQ yminP ymaxQ
136 3 p(i)= yminP q(j)= ymaxQ (p(i), q(j)) p(i-1),p(i+1) (p(i), q(j))
      q(j-1),q(j+1) (p(i), q(j)) CS
137
138 5
139 645 (yminP,ymaxQ)
140 7CS
141
142 ////
143 1 xminP xmaxP yminP ymaxP
144 2 P
145 3
146 4
147 5/
148 645 90
149 7

```

3.16 shit

```

1 struct pv
2 {
3     double x,y;
4     pv() :x(0),y(0) {}
5     pv(double xx,double yy):x(xx),y(yy) {}
6     inline pv operator+(const pv &l)const
7     {
8         return pv(x+i.x,y+i.y);
9     }
10    inline pv operator-(const pv &l)const
11    {
12        return pv(x-i.x,y-i.y);
13    }
14    inline bool operator ==(const pv &l)const
15    {
16        return fabs(x-i.x)<eps && fabs(y-i.y)<eps;
17    }
18    inline bool operator<(const pv &l)const
19    {
20        return y==i.y?x<i.x:y<i.y;
21    }
22    inline double cross(const pv &l)const
23    {
24        return x*i.y-y*i.x;
25    }
26    inline double dot(const pv &l)const
27    {
28        return x*i.x+y*i.y;
29    }
30    inline double len()
31    {
32        return sqrt(x*x+y*y);
33    }
34    inline pv rotate(pv p,double theta)
35    {
36        static pv v;
37        v=-this-p;
38        static double c,s;
39        c=cos(theta);
40        s=sin(theta);
41        return pv(p.x+v.x*c-v.y*s,p.y+v.x*s+v.y*c);
42    }
43 };
44
45 inline int dblcmp(double d)
46 {
47     if(fabs(d)<eps)
48         return 0;
49     return d>eps?-1;

```

```

50 }
51
52 inline int cross(pv *a,pv *b) // 0 1 2
53 {
54     int d1=dblcmp((a[1]-a[0]).cross(b[0]-a[0]));
55     int d2=dblcmp((a[1]-a[0]).cross(b[1]-a[0]));
56     int d3=dblcmp((b[1]-b[0]).cross(a[0]-b[0]));
57     int d4=dblcmp((b[1]-b[0]).cross(a[1]-b[0]));
58     if((d1*d2)==-2 && (d3*d4)==-2)
59         return 2;
60     return ((d1==0 && dblcmp((b[0]-a[0]).dot(b[0]-a[1]))<=0 )||
61            (d2==0 && dblcmp((b[1]-a[0]).dot(b[1]-a[1]))<=0 )||
62            (d3==0 && dblcmp((a[0]-b[0]).dot(a[0]-b[1]))<=0 )||
63            (d4==0 && dblcmp((a[1]-b[0]).dot(a[1]-b[1]))<=0));
64 }
65
66 inline bool pntonseg(const pv &p,const pv *a)
67 {
68     return fabs((p-a[0]).cross(p-a[1]))<eps && (p-a[0]).dot(p-a[1])
69         <eps;
70 }
71 pv rotate(pv v,pv p,double theta,double sc=1) // rotate vector v,
      theta [0,2]
72 {
73     static pv re;
74     re=p;
75     v=v-p;
76     p.x=sc*cos(theta);
77     p.y=sc*sin(theta);
78     re.x+=v.x*p.x-v.y*p.y;
79     re.y+=v.x*p.y+v.y*p.x;
80     return re;
81 }
82
83 struct line
84 {
85     pv pnt[2];
86     line(double a,double b,double c) // a*x + b*y + c = 0
87     {
88         #define maxl 1e2 //preciseness should not be too high ( compare
            with eps )
89         if(fabs(b)>eps)
90         {
91             pnt[0]=pv(maxl,(c+a*maxl)/(-b));
92             pnt[1]=pv(-maxl,(c-a*maxl)/(-b));
93         }
94         else
95         {
96             pnt[0]=pv(-c/a,maxl);
97             pnt[1]=pv(-c/a,-maxl);
98         }
99     }
100     #undef maxl
101     pv cross(const line &v)const
102     {
103         double a=(v.pnt[1]-v.pnt[0]).cross(pnt[0]-v.pnt[0]);
104         double b=(v.pnt[1]-v.pnt[0]).cross(pnt[1]-v.pnt[0]);
105         return pv((pnt[0].x*b-pnt[1].x*a)/(b-a),(pnt[0].y*b-pnt[1].y
            *a)/(b-a));
106     }
107 };
108
109 inline std::pair<pv,double> getcircle(const pv &a,const pv &b,
      const pv &c)
110 {
111     static pv ct;
112     ct=line(2*(b.x-a.x),2*(b.y-a.y),a.len()-b.len()).cross(line(2*(
        c.x-b.x),2*(c.y-b.y),b.len()-c.len()));
113     return std::make_pair(ct,sqrt((ct-a).len()));
114 }

```

3.17 sort - polar angle

```

1 inline bool cmp(const Point& a,const Point& b)
2 {
3     if (a.y*b.y <= 0)
4     {
5         if (a.y > 0 || b.y > 0)
6             return a.y < b.y;
7         if (a.y == 0 && b.y == 0)
8             return a.x < b.x;
9     }
10    return a.cross(b) > 0;
11 }

```

3.18 triangle

```

1 Area:
2 p=(a+b+c)/2
3 area=sqrt(p*(p-a)*(p-b)*(p-c));
4 area=a*b*sin(C)/2;
5 area=sq(a)*sin(B)*sin(C)/2/sin(B+C);
6 area=sq(a)/2/(cot(B)+cot(C));
7
8 centroid:
9 center of mass
10 intersection of triangle's three triangle medians
11
12 Trigonometric conditions:
13 tan(A/2)+tan(B/2)+tan(B/2)*tan(C/2)+tan(A/2)*tan(C/2)==1
14 sq(sin(A/2))+sq(sin(B/2))+sq(sin(C/2))+2*sin(A/2)*sin(B/2)*sin(C
    /2)==1
15
16 Circumscribed circle:
17 diameter=a*b*c/(2*area);
18 diameter=sqrt(2*area/sin(A)/sin(B)/sin(C));
19 diameter=a/sin(A)=b/sin(B)=c/sin(C);

```

```

20
21 Incircle:
22 inradius=2*area/(a+b+c);
23 coordinates(x,y)=a*(xa,ya)/(a+b+c)+b*(xb,yb)/(a+b+c)+c*(xc,yc)/(a+
    b+c);
24
25 Excircles:
26 radius[a]=2*area/(b+c-a);
27 radius[b]=2*area/(a+c-b);
28 radius[c]=2*area/(a+b-c);
29
30 Steiner circumellipse (least area circumscribed ellipse)
31 area= area * 4*pi/3/sqrt(3);
32 center is the triangle's centroid.
33
34 Steiner inellipse (maximum area inellipse)
35 area= area * pi/3/sqrt(3);
36 center is the triangle's centroid.
37
38 Fermat Point:
39 120
40
41 120
42
43 ABC'BCA'CAB'
44 CC'BB'AA'

```

4 geometry/tmp

4.1 circle

```

1 struct circle
2 {
3     point p;
4     double r;
5     circle() {}
6     circle(point _p,double _r):
7     p(_p),r(_r){};
8     circle(double x,double y,double _r):
9     p(point(x,y)),r(_r){};
10    circle(point a,point b,point c){//
11    {
12        p=line(a.add(b).div(2),a.add(b).div(2).add(b.sub(a).rotleft(1)
            )),crosspoint(line(c.add(b).div(2),c.add(b).div(2).add(
                b.sub(c).rotleft(1))));
13        r=p.distance(a);
14    }
15    circle(point a,point b,point c,bool t){//
16    {
17        line u,v;
18        double m=atan2(b.y-a.y,b.x-a.x),n=atan2(c.y-a.y,c.x-a.x);
19        u.a=a;
20        u.b=u.a.add(point(cos((n+m)/2),sin((n+m)/2)));
21        v.a=b;
22        m=atan2(a.y-b.y,a.x-b.x),n=atan2(c.y-b.y,c.x-b.x);
23        v.b=v.a.add(point(cos((n+m)/2),sin((n+m)/2)));
24        p=u.crosspoint(v);
25        r=line(a,b).dispointtoseg(p);
26    }
27    void input()
28    {
29        p.input();
30        scanf("%lf",&r);
31    }
32    void output()
33    {
34        printf("%.2lf %.2lf %.2lf\n",p.x,p.y,r);
35    }
36    bool operator==(circle v)
37    {
38        return ((p==v.p)&&dblcmp(r-v.r)==0);
39    }
40    bool operator<(circle v)const
41    {
42        return ((p<v.p)|| (p==v.p)&&dblcmp(r-v.r)<0);
43    }
44    double area()
45    {
46        return pi*r*r;
47    }
48    double circumference()
49    {
50        return 2*pi*r;
51    }
52    //0
53    //1
54    //2
55    int relation(point b)
56    {
57        double dst=b.distance(p);
58        if (dblcmp(dst-r)<0)return 2;
59        if (dblcmp(dst-r)==0)return 1;
60        return 0;
61    }
62    int relationseg(line v)
63    {
64        double dst=v.dispointtoseg(p);
65        if (dblcmp(dst-r)<0)return 2;
66        if (dblcmp(dst-r)==0)return 1;
67        return 0;
68    }
69    int relationline(line v)
70    {
71        double dst=v.dispointtoline(p);
72        if (dblcmp(dst-r)<0)return 2;
73        if (dblcmp(dst-r)==0)return 1;
74        return 0;
75    }
76    //a b r
77    int getcircle(point a,point b,double r,circle&c1,circle&c2)
78    {

```

```

79    circle x(a,r),y(b,r);
80    int t=x.pointcrosscircle(y,c1.p,c2.p);
81    if (!t)return 0;
82    c1.r=c2.r=r;
83    return t;
84    }
85    //u q r1
86    int getcircle(line u,point q,double r1,circle &c1,circle &c2)
87    {
88        double dis=u.dispointtoline(q);
89        if (dblcmp(dis-r1*2)>0)return 0;
90        if (dblcmp(dis)==0)
91        {
92            c1.p=q.add(u.b.sub(u.a).rotleft(1).trunc(r1));
93            c2.p=q.add(u.b.sub(u.a).rotright(1).trunc(r1));
94            c1.r=c2.r=r1;
95            return 2;
96        }
97        line u1=line(u.a.add(u.b.sub(u.a).rotleft(1).trunc(r1)),u.b.
            add(u.b.sub(u.a).rotleft(1).trunc(r1)));
98        line u2=line(u.a.add(u.b.sub(u.a).rotright(1).trunc(r1)),u.b.
            add(u.b.sub(u.a).rotright(1).trunc(r1)));
99        circle cc=circle(q,r1);
100        point p1,p2;
101        if (!cc.pointcrossline(u1,p1,p2))cc.pointcrossline(u2,p1,p2);
102        c1=circle(p1,r1);
103        if (p1==p2)
104        {
105            c2=c1;return 1;
106        }
107        c2=circle(p2,r1);
108        return 2;
109    }
110    //u,v r1
111    int getcircle(line u,line v,double r1,circle &c1,circle &c2,
        circle &c3,circle &c4)
112    {
113        if (u.parallel(v))return 0;
114        line u1=line(u.a.add(u.b.sub(u.a).rotleft(1).trunc(r1)),u.b.
            add(u.b.sub(u.a).rotleft(1).trunc(r1)));
115        line u2=line(u.a.add(u.b.sub(u.a).rotright(1).trunc(r1)),u.b.
            add(u.b.sub(u.a).rotright(1).trunc(r1)));
116        line v1=line(v.a.add(v.b.sub(v.a).rotleft(1).trunc(r1)),v.b.
            add(v.b.sub(v.a).rotleft(1).trunc(r1)));
117        line v2=line(v.a.add(v.b.sub(v.a).rotright(1).trunc(r1)),v.b.
            add(v.b.sub(v.a).rotright(1).trunc(r1)));
118        c1.r=c2.r=c3.r=c4.r=r1;
119        c1.p=u1.crosspoint(v1);
120        c2.p=u1.crosspoint(v2);
121        c3.p=u2.crosspoint(v1);
122        c4.p=u2.crosspoint(v2);
123        return 4;
124    }
125    //cx,cy r1
126    int getcircle(circle cx,circle cy,double r1,circle&c1,circle&c2)
127    {
128        circle x(cx.p,r1+cx.r),y(cy.p,r1+cy.r);
129        int t=x.pointcrosscircle(y,c1.p,c2.p);
130        if (!t)return 0;
131        c1.r=c2.r=r1;
132        return t;
133    }
134    int pointcrossline(line v,point &p1,point &p2)//relationseg
135    {
136        if (!(*this).relationline(v))return 0;
137        point a=v.lineprog(p);
138        double d=v.dispointtoline(p);
139        d=sqrt(r*r-d*d);
140        if (dblcmp(d)==0)
141        {
142            p1=a;
143            p2=a;
144            return 1;
145        }
146        p1=a.sub(v.b.sub(v.a).trunc(d));
147        p2=a.add(v.b.sub(v.a).trunc(d));
148        return 2;
149    }
150    //5
151    //4
152    //3
153    //2
154    //1
155    int relationcircle(circle v)
156    {
157        double d=p.distance(v.p);
158        if (dblcmp(d-r-v.r)>0)return 5;
159        if (dblcmp(d-r-v.r)==0)return 4;
160        double l=fabs(r-v.r);
161        if (dblcmp(d-r-v.r)<0&&dblcmp(d-l)>0)return 3;
162        if (dblcmp(d-l)==0)return 2;
163        if (dblcmp(d-l)<0)return 1;
164    }
165    int pointcrosscircle(circle v,point &p1,point &p2)
166    {
167        int rel=relationcircle(v);
168        if (rel==1||rel==5)return 0;
169        double d=p.distance(v.p);
170        double l=(d+(sqrt(r)-sqrt(v.r))/d)/2;
171        double h=sqrt((sqrt(r)-sqrt(l)));
172        p1=p.add(v.p.sub(p).trunc(l).add(v.p.sub(p).rotleft(1).trunc(
            h)));
173        p2=p.add(v.p.sub(p).trunc(l).add(v.p.sub(p).rotright(1).trunc(
            h)));
174        if (rel==2||rel==4)
175        {
176            return 1;
177        }
178        return 2;
179    }
180    // ()
181    int tangentline(point q,line &u,line &v)
182    {
183        int x=relation(q);
184        if (x==2)return 0;
185        if (x==1)
186        {
187            u=line(q,q.add(q.sub(p).rotleft(1)));
188            v=u;
189            return 1;

```

```

190     }
191     double d=p.distance(q);
192     double l=sqr(r)/d;
193     double h=sqrt(sqr(r)-sqr(l));
194     u=line(q,p.add(q.sub(p).trunc(l).add(q.sub(p).rotleft().trunc(h))));
195     v=line(q,p.add(q.sub(p).trunc(l).add(q.sub(p).rotright().trunc(h))));
196     return 2;
197 }
198 double areacircle(circle v)
199 {
200     int rel=relationcircle(v);
201     if (rel>=4) return 0.0;
202     if (rel<=2) return min(area(),v.area());
203     double d=p.distance(v.p);
204     double hf=(r+v.r+d)/2.0;
205     double ss=2*sqrt(hf*(hf-r)*(hf-v.r)*(hf-d));
206     double al=acos((r+r+d-d-v.r*v.r)/(2.0*r*d));
207     al=al+r;
208     double a2=acos((v.r*v.r+d-d-r*r)/(2.0*v.r*d));
209     a2=a2+v.r*v.r;
210     return al+a2-ss;
211 }
212 double areatriangle(point a,point b)
213 {
214     if (dblcmp(p.sub(a).det(p.sub(b))==0)) return 0.0;
215     point q[5];
216     int len=0;
217     q[len++]=a;
218     line l(a,b);
219     point p1,p2;
220     if (pointcrossline(l,q[1],q[2])==2)
221     {
222         if (dblcmp(a.sub(q[1]).dot(b.sub(q[1])))<0) q[len++]=q[1];
223         if (dblcmp(a.sub(q[2]).dot(b.sub(q[2])))<0) q[len++]=q[2];
224     }
225     q[len++]=b;
226     if (len==4&&(dblcmp(q[0].sub(q[1]).dot(q[2].sub(q[1]))>0))
227         swap(q[1],q[2]));
228     double res=0;
229     int i;
230     for (i=0;i<len-1;i++)
231     {
232         if (relation(q[i])==0||relation(q[i+1])==0)
233         {
234             double arg=p.rad(q[i],q[i+1]);
235             res+=r*r*arg/2.0;
236         }
237         else
238             res+=fabs(q[i].sub(p).det(q[i+1].sub(p))/2.0);
239     }
240     return res;
241 }
242 }
243 };

```

```

55     memset(ans,0,sizeof(ans));
56     vector<pair<double,int> >v;
57     for (i=0;i<n;i++)
58     {
59         v.clear();
60         v.push_back(make_pair(-pi,1));
61         v.push_back(make_pair(pi,-1));
62         for (j=0;j<n;j++) if (i!=j)
63         {
64             point q=c[j].p.sub(c[i].p);
65             double ab=q.len(),ac=c[i].r,bc=c[j].r;
66             if (dblcmp(ab+ac-bc)<=0)
67             {
68                 v.push_back(make_pair(-pi,1));
69                 v.push_back(make_pair(pi,-1));
70                 continue;
71             }
72             if (dblcmp(ab+bc-ac)<=0) continue;
73             if (dblcmp(ab-ac-bc)>0) continue;
74             double th=atan2(q.y,q.x),fa=acos((ac*ac+ab*ab-bc*bc)/(2.0*ac*ab));
75             double a0=th-fai;
76             if (dblcmp(a0+pi)<0) a0+=2*pi;
77             double al=th+fai;
78             if (dblcmp(al-pi)>0) al-=2*pi;
79             if (dblcmp(a0-al)>0)
80             {
81                 v.push_back(make_pair(a0,1));
82                 v.push_back(make_pair(pi,-1));
83                 v.push_back(make_pair(-pi,1));
84                 v.push_back(make_pair(al,-1));
85             }
86             else
87             {
88                 v.push_back(make_pair(a0,1));
89                 v.push_back(make_pair(al,-1));
90             }
91         }
92         sort(v.begin(),v.end());
93         int cur=0;
94         for (j=0;j<v.size();j++)
95         {
96             if (cur&&dblcmp(v[j].first-pre[cur]))
97             {
98                 ans[cur]+=areaarc(v[j].first-pre[cur],c[i].r);
99                 ans[cur]+=0.5*point(c[i].p.x+c[i].r*cos(pre[cur]),c[i].p.y
100                     +c[i].r*sin(pre[cur])).det(point(c[i].p.x+c[i].r*cos(v[j].first),c[i].p.y+c[i].r*sin(v[j].first)));
101             }
102             cur+=v[j].second;
103             pre[cur]=v[j].first;
104         }
105         for (i=1;i<n;i++)
106         {
107             ans[i]-=ans[i+1];
108         }
109     }
110 };

```

4.2 circles

```

1  const int maxn=500;
2  struct circles
3  {
4      circle c[maxn];
5      double ans[maxn]; //ans[i] i
6      double pre[maxn];
7      int n;
8      circles() {}
9      void add(circle cc)
10     {
11         c[n++]=cc;
12     }
13     bool inner(circle x,circle y)
14     {
15         if (x.relationcircle(y)!=1) return 0;
16         return dblcmp(x.r-y.r)<=0?1:0;
17     }
18     void init_or() //
19     {
20         int i,j,k=0;
21         bool mark[maxn]={0};
22         for (i=0;i<n;i++)
23         {
24             for (j=0;j<n;j++) if (i!=j&&!mark[j])
25             {
26                 if ((c[i]==c[j])||inner(c[i],c[j])) break;
27             }
28             if (j<n) mark[i]=1;
29         }
30         for (i=0;i<n;i++) if (!mark[i]) c[k++]=c[i];
31         n=k;
32     }
33     void init_and() //
34     {
35         int i,j,k=0;
36         bool mark[maxn]={0};
37         for (i=0;i<n;i++)
38         {
39             for (j=0;j<n;j++) if (i!=j&&!mark[j])
40             {
41                 if ((c[i]==c[j])||inner(c[j],c[i])) break;
42             }
43             if (j<n) mark[i]=1;
44         }
45         for (i=0;i<n;i++) if (!mark[i]) c[k++]=c[i];
46         n=k;
47     }
48     double areaarc(double th,double r)
49     {
50         return 0.5*sqr(r)*(th-sin(th));
51     }
52     void getarea()
53     {
54         int i,j,k;

```

4.3 halfplane

```

1  struct halfplane:public line
2  {
3      double angle;
4      halfplane() {}
5      // a->b()
6      halfplane(point _a,point _b)
7      {
8          a=_a;
9          b=_b;
10     }
11     halfplane(line v)
12     {
13         a=v.a;
14         b=v.b;
15     }
16     void calcangle()
17     {
18         angle=atan2(b.y-a.y,b.x-a.x);
19     }
20     bool operator<(const halfplane &b) const
21     {
22         return angle<b.angle;
23     }
24 };
25 struct halfplanes
26 {
27     int n;
28     halfplane hp[maxn];
29     point p[maxn];
30     int que[maxn];
31     int st,ed;
32     void push(halfplane tmp)
33     {
34         hp[n++]=tmp;
35     }
36     void unique()
37     {
38         int m=1,i;
39         for (i=1;i<n;i++)
40         {
41             if (dblcmp(hp[i].angle-hp[i-1].angle)) hp[m++]=hp[i];
42             else if (dblcmp(hp[m-1].b.sub(hp[m-1].a).det(hp[i].a.sub(hp[m-1].a))>0) hp[m-1]=hp[i];
43         }
44         n=m;
45     }
46     bool halfplaneinsert()
47     {
48         int i;
49         for (i=0;i<n;i++) hp[i].calcangle();
50         sort(hp,hp+n);
51         unique();

```

```

52     que[st=0]=0;
53     que[ed=1]=1;
54     p[1]=hp[0].crosspoint(hp[1]);
55     for (i=2; i<n; i++)
56     {
57         while (st<ed&&dblcmp((hp[i].b.sub(hp[i].a).det(p[ed]).sub(hp[i]
58             ].a)))<0)ed--;
59         while (st<ed&&dblcmp((hp[i].b.sub(hp[i].a).det(p[st+1]).sub(hp
60             [i].a)))<0)st++;
61         que[++ed]=i;
62         if (hp[i].parallel(hp[que[ed-1]]))return false;
63         p[ed]=hp[i].crosspoint(hp[que[ed-1]]);
64         while (st<ed&&dblcmp(hp[que[st]].b.sub(hp[que[st]].a).det(p[ed]
65             ].sub(hp[que[st]].a)))<0)ed--;
66         while (st<ed&&dblcmp(hp[que[ed]].b.sub(hp[que[ed]].a).det(p[st
67             +1].sub(hp[que[ed]].a)))<0)st++;
68         if (st+1>ed)return false;
69         return true;
70     }
71     void getconvex(polygons &con)
72     {
73         p[st]=hp[que[st]].crosspoint(hp[que[ed]]);
74         con.n=ed-st+1;
75         int j=st, i=0;
76         for (; j<=ed; j++, i++)
77         {
78             con.p[i]=p[j];
79         }
80     };

```

4.4 line

```

1  struct line
2  {
3      point a,b;
4      line() {}
5      line(point _a, point _b)
6      {
7          a=_a;
8          b=_b;
9      }
10     bool operator==(line v)
11     {
12         return (a==v.a && b==v.b);
13     }
14     //angle
15     line(point p, double angle)
16     {
17         a=p;
18         if (dblcmp(angle-pi/2)==0)
19         {
20             b=a.add(point(0,1));
21         }
22         else
23         {
24             b=a.add(point(1,tan(angle)));
25         }
26     }
27     //ax+by+c=0
28     line(double _a, double _b, double _c)
29     {
30         if (dblcmp(_a)==0)
31         {
32             a=point(0,-_c/_b);
33             b=point(1,-_c/_b);
34         }
35         else if (dblcmp(_b)==0)
36         {
37             a=point(-_c/_a,0);
38             b=point(-_c/_a,1);
39         }
40         else
41         {
42             a=point(0,-_c/_b);
43             b=point(1,(-_c-_a)/_b);
44         }
45     }
46     void input()
47     {
48         a.input();
49         b.input();
50     }
51     void adjust()
52     {
53         if (b<a) swap(a,b);
54     }
55     double length()
56     {
57         return a.distance(b);
58     }
59     double angle() // 0<=angle<180
60     {
61         double k=atan2(b.y-a.y, b.x-a.x);
62         if (dblcmp(k)<0) k+=pi;
63         if (dblcmp(k-pi)==0) k-=pi;
64         return k;
65     }
66     //
67     //1
68     //2
69     //3
70     int relation(point p)
71     {
72         int c=dblcmp(p.sub(a).det(b.sub(a)));
73         if (c<0) return 1;
74         if (c>0) return 2;
75         return 3;
76     }
77     bool pointonseg(point p)
78     {
79         return dblcmp(p.sub(a).det(b.sub(a)))==0&&dblcmp(p.sub(a).

```

```

80     }
81     bool parallel(line v)
82     {
83         return dblcmp(b.sub(a).det(v.b.sub(v.a)))==0;
84     }
85     //2
86     //1
87     //0
88     int segcrossseg(line v)
89     {
90         int d1=dblcmp(b.sub(a).det(v.a.sub(a)));
91         int d2=dblcmp(b.sub(a).det(v.b.sub(a)));
92         int d3=dblcmp(v.b.sub(v.a).det(a.sub(v.a)));
93         int d4=dblcmp(v.b.sub(v.a).det(b.sub(v.a)));
94         if ((d1*d2)==-2&&(d3*d4)==-2) return 2;
95         return (d1==0&&dblcmp(v.a.sub(a).dot(v.a.sub(b)))<=0 ||
96             d2==0&&dblcmp(v.b.sub(a).dot(v.b.sub(b)))<=0 ||
97             d3==0&&dblcmp(a.sub(v.a).dot(a.sub(v.b)))<=0 ||
98             d4==0&&dblcmp(b.sub(v.a).dot(b.sub(v.b)))<=0);
99     }
100     int linecrossseg(line v) // *this seg v line
101     {
102         int d1=dblcmp(b.sub(a).det(v.a.sub(a)));
103         int d2=dblcmp(b.sub(a).det(v.b.sub(a)));
104         if ((d1*d2)==-2) return 2;
105         return (d1==0 || d2==0);
106     }
107     //0
108     //1
109     //2
110     int linecrossline(line v)
111     {
112         if ((~this).parallel(v))
113         {
114             return v.relation(a)==3;
115         }
116         return 2;
117     }
118     point crosspoint(line v)
119     {
120         double a1=v.b.sub(v.a).det(a.sub(v.a));
121         double a2=v.b.sub(v.a).det(b.sub(v.a));
122         return point((a.x*a2-b.x*a1)/(a2-a1), (a.y*a2-b.y*a1)/(a2-a1));
123     }
124     double dispointtoline(point p)
125     {
126         return fabs(p.sub(a).det(b.sub(a)))/length();
127     }
128     double dispointtoseg(point p)
129     {
130         if (dblcmp(p.sub(b).dot(a.sub(b)))<0 || dblcmp(p.sub(a).dot(b.
131             sub(a)))<0)
132         {
133             return min(p.distance(a), p.distance(b));
134         }
135         return dispointtoline(p);
136     }
137     point lineprog(point p)
138     {
139         return a.add(b.sub(a).mul(b.sub(a).dot(p.sub(a))/b.sub(a).
140             len2());
141     }
142     point symmetrpoint(point p)
143     {
144         point q=lineprog(p);
145         return point(2*q.x-p.x, 2*q.y-p.y);
146     }

```

4.5 line3d

```

1  struct line3
2  {
3      point3 a,b;
4      line3() {}
5      line3(point3 _a, point3 _b)
6      {
7          a=_a;
8          b=_b;
9      }
10     bool operator==(line3 v)
11     {
12         return (a==v.a && b==v.b);
13     }
14     void input()
15     {
16         a.input();
17         b.input();
18     }
19     double length()
20     {
21         return a.distance(b);
22     }
23     bool pointonseg(point3 p)
24     {
25         return dblcmp(p.sub(a).det(p.sub(b)).len())==0&&dblcmp(a.sub(
26             p).dot(b.sub(p)))<0;
27     }
28     double dispointtoline(point3 p)
29     {
30         return b.sub(a).det(p.sub(a)).len()/a.distance(b);
31     }
32     double dispointtoseg(point3 p)
33     {
34         if (dblcmp(p.sub(b).dot(a.sub(b)))<0 || dblcmp(p.sub(a).dot(b.
35             sub(a)))<0)
36         {
37             return min(p.distance(a), p.distance(b));
38         }
39         return dispointtoline(p);
40     }
41     point3 lineprog(point3 p)
42     {
43         return a.add(b.sub(a).mul(b.sub(a).dot(p.sub(a))/b.sub(a).
44             len2());
45     }

```

```

41     return a.add(b.sub(a).trunc(b.sub(a).dot(p.sub(a))/b.distance
42         (a)));
43 point3 rotate(point3 p,double ang)//parg
44 {
45     if (dblcmp((p.sub(a).det(p.sub(b)).len()))==0)return p;
46     point3 f1=b.sub(a).det(p.sub(a));
47     point3 f2=b.sub(a).det(f1);
48     double len=fabs(a.sub(p).det(b.sub(p)).len()/a.distance(b));
49     f1=f1.trunc(len);f2=f2.trunc(len);
50     point3 h=p.add(f2);
51     point3 pp=h.add(f1);
52     return h.add((p.sub(h)).mul(cos(ang*1.0)).add((pp.sub(h)).mul(
53         sin(ang*1.0)));
54 };

```

4.6 plane

```

1 struct plane
2 {
3     point3 a,b,c,o;
4     plane(){}
5     plane(point3 _a,point3 _b,point3 _c)
6     {
7         a=_a;
8         b=_b;
9         c=_c;
10        o=pvec();
11    }
12    plane(double _a,double _b,double _c,double _d)
13    {
14        //ax+by+cz+d=0
15        o=point3(_a,_b,_c);
16        if (dblcmp(_a)!=0)
17        {
18            a=point3((-_d-_c-_b)/_a,1,1);
19        }
20        else if (dblcmp(_b)!=0)
21        {
22            a=point3(1,(-_d-_c-_a)/_b,1);
23        }
24        else if (dblcmp(_c)!=0)
25        {
26            a=point3(1,1,(-_d-_a-_b)/_c);
27        }
28    }
29    void input()
30    {
31        a.input();
32        b.input();
33        c.input();
34        o=pvec();
35    }
36    point3 pvec()
37    {
38        return b.sub(a).det(c.sub(a));
39    }
40    bool pointonplane(point3 p)//
41    {
42        return dblcmp(p.sub(a).dot(o))==0;
43    }
44    //0
45    //1
46    //2
47    int pointontriangle(point3 p)//abc
48    {
49        if (!pointonplane(p))return 0;
50        double s=a.sub(b).det(c.sub(b)).len();
51        double s1=p.sub(a).det(p.sub(b)).len();
52        double s2=p.sub(a).det(p.sub(c)).len();
53        double s3=p.sub(b).det(p.sub(c)).len();
54        if (dblcmp(s-s1-s2-s3))return 0;
55        if (dblcmp(s1)&dblcmp(s2)&dblcmp(s3))return 2;
56        return 1;
57    }
58    //
59    //0
60    //1
61    //2
62    bool relationplane(plane f)
63    {
64        if (dblcmp(o.det(f.o).len()))return 0;
65        if (pointonplane(f.a))return 2;
66        return 1;
67    }
68    double angleplane(plane f)//
69    {
70        return acos(o.dot(f.o)/(o.len()*f.o.len()));
71    }
72    double dispoint(point3 p)//
73    {
74        return fabs(p.sub(a).dot(o)/o.len());
75    }
76    point3 pttoplane(point3 p)//
77    {
78        line3 u=line3(p,p.add(o));
79        crossline(u,p);
80        return p;
81    }
82    int crossline(line3 u,point3 sp)//
83    {
84        double x=o.dot(u.b.sub(a));
85        double y=o.dot(u.a.sub(a));
86        double d=x-y;
87        if (dblcmp(fabs(d))==0)return 0;
88        p=u.a.mul(x).sub(u.b.mul(y)).div(d);
89        return 1;
90    }
91    int crossplane(plane f,line3 su)//
92    {
93        point3 oo=o.det(f.o);
94        point3 vo=o.det(oo);
95        double d=fabs(f.o.dot(v));

```

```

96     if (dblcmp(d)==0)return 0;
97     point3 q=a.add(v.mul(f.o.dot(f.a.sub(a))/d));
98     u=line3(q,q.add(oo));
99     return 1;
100 }
101 };

```

4.7 point

```

1 using namespace std;
2
3 #define mp make_pair
4 #define pb push_back
5
6 const double eps=1e-8;
7 const double pi=acos(-1.0);
8 const double inf=1e20;
9 const int maxp=8;
10
11 int dblcmp(double d)
12 {
13     if (fabs(d)<eps)return 0;
14     return d>eps?1:-1;
15 }
16
17 inline double sqr(double x)
18 {
19     return x*x;
20 }
21
22 struct point
23 {
24     double x,y;
25     point(){}
26     point(double _x,double _y):
27         x(_x),y(_y){};
28     void input()
29     {
30         scanf("%lf%lf",&x,&y);
31     }
32     void output()
33     {
34         printf("%.2f %.2f\n",x,y);
35     }
36     bool operator==(point a)const
37     {
38         return dblcmp(a.x-x)==0&&dblcmp(a.y-y)==0;
39     }
40     bool operator<(point a)const
41     {
42         return dblcmp(a.x-x)==0?dblcmp(y-a.y)<0:x<a.x;
43     }
44     double len()
45     {
46         return hypot(x,y);
47     }
48     double len2()
49     {
50         return x*x+y*y;
51     }
52     double distance(point p)
53     {
54         return hypot(x-p.x,y-p.y);
55     }
56     point add(point p)
57     {
58         return point(x+p.x,y+p.y);
59     }
60     point sub(point p)
61     {
62         return point(x-p.x,y-p.y);
63     }
64     point mul(double b)
65     {
66         return point(x*b,y*b);
67     }
68     point div(double b)
69     {
70         return point(x/b,y/b);
71     }
72     double dot(point p)
73     {
74         return x*p.x+y*p.y;
75     }
76     double det(point p)
77     {
78         return x*p.y-y*p.x;
79     }
80     double rad(point a,point b)
81     {
82         point p=*this;
83         return fabs(atan2(fabs(a.sub(p).det(b.sub(p))),a.sub(p).dot(b
84             .sub(p))));
85     }
86     point trunc(double r)
87     {
88         double l=len();
89         if (!dblcmp(l))return *this;
90         r/=l;
91         return point(x*r,y*r);
92     }
93     point rotleft()
94     {
95         return point(-y,x);
96     }
97     point rotright()
98     {
99         return point(y,-x);
100    }
101    point rotate(point p,double angle)//pangle
102    {
103        point v=*this->sub(p);
104        double c=cos(angle),s=sin(angle);
105        return point(p.x+v.x*c-v.y*s,p.y+v.x*s+v.y*c);

```



```
105     }
106 };
```

4.8 point3d

```
1 struct point3
2 {
3     double x,y,z;
4     point3(){}
5     point3(double _x,double _y,double _z):
6     x(_x),y(_y),z(_z){};
7     void input()
8     {
9         scanf("%lf%lf%lf",&x,&y,&z);
10    }
11    void output()
12    {
13        printf("%.2lf %.2lf %.2lf\n",x,y,z);
14    }
15    bool operator==(point3 a)
16    {
17        return dblcmp(a.x-x)==0&&dblcmp(a.y-y)==0&&dblcmp(a.z-z)==0;
18    }
19    bool operator<(point3 a)const
20    {
21        return dblcmp(a.x-x)==0?dblcmp(y-a.y)==0?dblcmp(z-a.z)<0:y<a
22            .y:x<a.x;
23    }
24    double len()
25    {
26        return sqrt(len2());
27    }
28    double len2()
29    {
30        return x*x+y*y+z*z;
31    }
32    double distance(point3 p)
33    {
34        return sqrt((p.x-x)*(p.x-x)+(p.y-y)*(p.y-y)+(p.z-z)*(p.z-z));
35    }
36    point3 add(point3 p)
37    {
38        return point3(x+p.x,y+p.y,z+p.z);
39    }
40    point3 sub(point3 p)
41    {
42        return point3(x-p.x,y-p.y,z-p.z);
43    }
44    point3 mul(double d)
45    {
46        return point3(x*d,y*d,z*d);
47    }
48    point3 div(double d)
49    {
50        return point3(x/d,y/d,z/d);
51    }
52    double dot(point3 p)
53    {
54        return x*p.x+y*p.y+z*p.z;
55    }
56    point3 det(point3 p)
57    {
58        return point3(y*p.z-p.y*z,p.x*z-x*p.z,x*p.y-p.x*y);
59    }
60    double rad(point3 a,point3 b)
61    {
62        point3 p=(+this);
63        return acos(a.sub(p).dot(b.sub(p))/(a.distance(p)*b.distance(
64            p)));
65    }
66    point3 trunc(double r)
67    {
68        r/=len();
69        return point3(x*r,y*r,z*r);
70    }
71    point3 rotate(point3 o,double r) // building?
72    {
73    }
74 };
```

4.9 polygon

```
1 struct polygon
2 {
3     int n;
4     point p[maxp];
5     line l[maxp];
6     void input()
7     {
8         n=4;
9         p[0].input();
10        p[2].input();
11        double dis=p[0].distance(p[2]);
12        p[1]=p[2].rotate(p[0],pi/4);
13        p[1]=p[0].add((p[1].sub(p[0])).trunc(dis/sqrt(2.0)));
14        p[3]=p[2].rotate(p[0],2*pi-pi/4);
15        p[3]=p[0].add((p[3].sub(p[0])).trunc(dis/sqrt(2.0)));
16    }
17    void add(point q)
18    {
19        p[n++]=q;
20    }
21    void getline()
22    {
23        for(int i=0;i<n;i++)
24        {
25            l[i]=line(p[i],p[(i+1)%n]);
```

```
26    }
27    }
28    struct cmp
29    {
30        point p;
31        cmp(const point &p0){p=p0;}
32        bool operator() (const point &aa,const point &bb)
33        {
34            point a=aa,b=bb;
35            int d=dblcmp(a.sub(p).det(b.sub(p)));
36            if (d==0)
37            {
38                return dblcmp(a.distance(p)-b.distance(p))<0;
39            }
40            return d>0;
41        }
42    };
43    void norm()
44    {
45        point mi=p[0];
46        for (int i=1;i<n;i++)mi=min(mi,p[i]);
47        sort(p,p+n,cmp(mi));
48    }
49    void getconvex(polygon &convex)
50    {
51        int i,j,k;
52        sort(p,p+n);
53        convex.n=n;
54        for (i=0;i<min(n,2);i++)
55        {
56            convex.p[i]=p[i];
57        }
58        if (n<=2)return;
59        int &top=convex.n;
60        top=1;
61        for (i=2;i<n;i++)
62        {
63            while (top&&convex.p[top].sub(p[i]).det(convex.p[top-1].
64                sub(p[i]))<=0)
65                top--;
66            convex.p[++top]=p[i];
67        }
68        int temp=top;
69        convex.p[++top]=p[n-2];
70        for (i=n-3;i>=0;i--)
71        {
72            while (top!=temp&&convex.p[top].sub(p[i]).det(convex.p[
73                top-1].sub(p[i]))<=0)
74                top--;
75            convex.p[++top]=p[i];
76        }
77        bool isconvex()
78        {
79            bool s[3];
80            memset(s,0,sizeof(s));
81            int i,j,k;
82            for (i=0;i<n;i++)
83            {
84                j=(i+1)%n;
85                k=(j+1)%n;
86                s[dblcmp(p[j].sub(p[i]).det(p[k].sub(p[i])))+1]=1;
87                if (s[0]&&s[2])return 0;
88            }
89            return 1;
90        }
91        //3
92        //2
93        //1
94        //0
95        int relationpoint(point q)
96        {
97            int i,j;
98            for (i=0;i<n;i++)
99            {
100                if (p[i]==q)return 3;
101            }
102            getline();
103            for (i=0;i<n;i++)
104            {
105                if (l[i].pointonseg(q))return 2;
106            }
107            int cnt=0;
108            for (i=0;i<n;i++)
109            {
110                j=(i+1)%n;
111                int k=dblcmp(q.sub(p[j]).det(p[i].sub(p[j])));
112                int u=dblcmp(p[i].y-q.y);
113                int v=dblcmp(p[j].y-q.y);
114                if (k>0&&u<0&&v>=0)cnt++;
115                if (k<0&&v<0&&u>=0)cnt--;
116            }
117            return cnt!=0;
118        }
119        //1
120        //2
121        //0
122        int relationline(line u)
123        {
124            int i,j,k=0;
125            getline();
126            for (i=0;i<n;i++)
127            {
128                if (l[i].segcrossseg(u)==2)return 1;
129                if (l[i].segcrossseg(u)==1)k=1;
130            }
131            if (!k)return 0;
132            vector<point>vp;
133            for (i=0;i<n;i++)
134            {
135                if (l[i].segcrossseg(u))
136                {
137                    if (l[i].parallel(u))
138                    {
139                        vp.pb(u.a);
140                        vp.pb(u.b);
141                        vp.pb(l[i].a);
142                        vp.pb(l[i].b);
143                        continue;
144                    }
145                }
146            }
147        }
148    }
149 };
```

```

144         vp.pb(l[i].crosspoint(u));
145     }
146 }
147 sort(vp.begin(), vp.end());
148 int sz=vp.size();
149 for (i=0; i<sz-1; i++)
150 {
151     point mid=vp[i].add(vp[i+1]).div(2);
152     if (relationpoint(mid)==1) return 1;
153 }
154 return 2;
155 }
156 //u
157 //
158 void convexcut(line u, polygon &po)
159 {
160     int i, j, k;
161     int &top=po.n;
162     top=0;
163     for (i=0; i<n; i++)
164     {
165         int d1=dblcmp(p[i].sub(u.a).det(u.b.sub(u.a)));
166         int d2=dblcmp(p[(i+1)%n].sub(u.a).det(u.b.sub(u.a)));
167         if (d1>0) po.p[top++]=p[i];
168         if (d1*d2<0) po.p[top++]=u.crosspoint(line(p[i], p[(i+1)%n]));
169     }
170 }
171 double getcircumference()
172 {
173     double sum=0;
174     int i;
175     for (i=0; i<n; i++)
176     {
177         sum+=p[i].distance(p[(i+1)%n]);
178     }
179     return sum;
180 }
181 double getarea()
182 {
183     double sum=0;
184     int i;
185     for (i=0; i<n; i++)
186     {
187         sum+=p[i].det(p[(i+1)%n]);
188     }
189     return fabs(sum)/2;
190 }
191 bool getdir() //1 0
192 {
193     double sum=0;
194     int i;
195     for (i=0; i<n; i++)
196     {
197         sum+=p[i].det(p[(i+1)%n]);
198     }
199     if (dblcmp(sum)>0) return 1;
200     return 0;
201 }
202 point getbarycentre() // centroid
203 {
204     point ret(0,0);
205     double area=0;
206     int i;
207     for (i=1; i<n-1; i++)
208     {
209         double tmp=p[i].sub(p[0]).det(p[i+1].sub(p[0]));
210         if (dblcmp(tmp)==0) continue;
211         area+=tmp;
212         ret.x+=(p[0].x+p[i].x+p[i+1].x)/3*tmp;
213         ret.y+=(p[0].y+p[i].y+p[i+1].y)/3*tmp;
214     }
215     if (dblcmp(area) ret=ret.div(area);
216     return ret;
217 }
218 double areaintersection(polygon po) // refer: HPI
219 {
220 }
221 double areaunion(polygon po)
222 {
223     return getarea()+po.getarea()-areaintersection(po);
224 }
225 double areacircle(circle c)
226 {
227     int i, j, k, l, m;
228     double ans=0;
229     for (i=0; i<n; i++)
230     {
231         int j=(i+1)%n;
232         if (dblcmp(p[j].sub(c.p).det(p[i].sub(c.p)))>=0)
233         {
234             ans+=c.areastriangle(p[i], p[j]);
235         }
236         else
237         {
238             ans-=c.areastriangle(p[i], p[j]);
239         }
240     }
241     return fabs(ans);
242 }
243 //
244 //0
245 //1
246 //2
247 int relationcircle(circle c)
248 {
249     getline();
250     int i, x=2;
251     if (relationpoint(c.p)!=1) return 0;
252     for (i=0; i<n; i++)
253     {
254         if (c.relationseg(l[i])==2) return 0;
255         if (c.relationseg(l[i])==1) x=1;
256     }
257     return x;
258 }
259 void find(int st, point tri[], circle &c)
260 {
261     if (!st)
262     {
263         c=circle(point(0,0), -2);
264     }
265     if (st==1)
266     {
267         c=circle(tri[0], 0);
268     }
269     if (st==2)
270     {
271         c=circle(tri[0].add(tri[1]).div(2), tri[0].distance(tri[1])/2.0);
272     }
273     if (st==3)
274     {
275         c=circle(tri[0], tri[1], tri[2]);
276     }
277 }
278 void solve(int cur, int st, point tri[], circle &c)
279 {
280     find(st, tri, c);
281     if (st==3) return;
282     int i;
283     for (i=0; i<cur; i++)
284     {
285         if (dblcmp(p[i].distance(c.p)-c.r)>0)
286         {
287             tri[st]=p[i];
288             solve(i, st+1, tri, c);
289         }
290     }
291 }
292 circle mincircle() //
293 {
294     random_shuffle(p, p+n);
295     point tri[4];
296     circle c;
297     solve(n, 0, tri, c);
298     return c;
299 }
300 int circlecover(double r) //
301 {
302     int ans=0, i, j;
303     vector<pair<double, int>> >v;
304     for (i=0; i<n; i++)
305     {
306         v.clear();
307         for (j=0; j<n; j++) if (i!=j)
308         {
309             point q=p[i].sub(p[j]);
310             double d=q.len();
311             if (dblcmp(d-2*r)<=0)
312             {
313                 double arg=atan2(q.y, q.x);
314                 if (dblcmp(arg)<0) arg+=2*pi;
315                 double t=acos(d/(2*r));
316                 v.push_back(make_pair(arg-t+2*pi, -1));
317                 v.push_back(make_pair(arg+t+2*pi, 1));
318             }
319         }
320         sort(v.begin(), v.end());
321         int cur=0;
322         for (j=0; j<v.size(); j++)
323         {
324             if (v[j].second==-1) ++cur;
325             else --cur;
326             ans=max(ans, cur);
327         }
328     }
329     return ans+1;
330 }
331 int pointinpolygon(point q) //
332 {
333     if (getdir()) reverse(p, p+n);
334     if (dblcmp(q.sub(p[0]).det(p[n-1].sub(p[0]))==0)
335     {
336         if (line(p[n-1], p[0]).pointonseg(q)) return n-1;
337         return -1;
338     }
339     int low=1, high=n-2, mid;
340     while (low<=high)
341     {
342         mid=(low+high)>>1;
343         if (dblcmp(q.sub(p[0]).det(p[mid].sub(p[0]))>=0&&dblcmp(q.sub(p[0]).det(p[mid+1].sub(p[0]))<0)
344         {
345             polygon c;
346             c.p[0]=p[mid];
347             c.p[1]=p[mid+1];
348             c.p[2]=p[0];
349             c.n=3;
350             if (c.relationpoint(q)) return mid;
351             return -1;
352         }
353         if (dblcmp(q.sub(p[0]).det(p[mid].sub(p[0]))>0)
354         {
355             low=mid+1;
356         }
357         else
358         {
359             high=mid-1;
360         }
361     }
362     return -1;
363 }
364 };

```

4.10 polygons

```

1 struct polygons
2 {
3     vector<polygon> p;
4     polygons()
5     {
6         p.clear();
7     }

```

```

8 void clear()
9 {
10     p.clear();
11 }
12 void push(polygon q)
13 {
14     if (dblcmp(q.getarea())) p.pb(q);
15 }
16 vector<pair<double,int>> e;
17 void ins(point s,point t,point X,int i)
18 {
19     double r=fabs(t.x-s.x)>eps?(X.x-s.x)/(t.x-s.x):(X.y-s.y)/(t.y-s
        .y);
20     r=min(r,1.0);r=max(r,0.0);
21     e.pb(mp(r,i));
22 }
23 double polyareaunion()
24 {
25     double ans=0.0;
26     int c0,c1,c2,i,j,k,w;
27     for (i=0;i<p.size();i++)
28     {
29         if (p[i].getdir()==0) reverse(p[i].p,p[i].p+p[i].n);
30     }
31     for (i=0;i<p.size();i++)
32     {
33         for (k=0;k<p[i].n;k++)
34         {
35             point &s=p[i].p[k],&t=p[i].p[(k+1)%p[i].n];
36             if (!dblcmp(s.det(t))) continue;
37             e.clear();
38             e.pb(mp(0.0,1));
39             e.pb(mp(1.0,-1));
40             for (j=0;j<p.size();j++) if (i!=j)
41             {
42                 for (w=0;w<p[j].n;w++)
43                 {
44                     point a=p[j].p[w],b=p[j].p[(w+1)%p[j].n],c=p[j].p[(w-1+p[
                        j].n)%p[j].n];
45                     c0=dblcmp(t.sub(s).det(c.sub(s)));
46                     c1=dblcmp(t.sub(s).det(a.sub(s)));
47                     c2=dblcmp(t.sub(s).det(b.sub(s)));
48                     if (c1*c2<0) ins(s,t,line(s,t).crosspoint(line(a,b)),-c2);
49                     else if (!c1&&c0*c2<0) ins(s,t,a,-c2);
50                     else if (!c1&&!c2)
51                     {
52                         int c3=dblcmp(t.sub(s).det(p[j].p[(w+2)%p[j].n].sub(s)
                            );
53                         int dp=dblcmp(t.sub(s).dot(b.sub(a)));
54                         if (dp&&c0) ins(s,t,a,dp>0?c0*((j>1)^ (c0<0)):- (c0<0));
55                         if (dp&&c3) ins(s,t,b,dp>0?-c3*((j>1)^ (c3<0)):c3<0);
56                     }
57                 }
58             }
59             sort(e.begin(),e.end());
60             int ct=0;
61             double tot=0.0,last;
62             for (j=0;j<e.size();j++)
63             {
64                 if (ct==p.size()) tot+=e[j].first-last;
65                 ct+=e[j].second;
66                 last=e[j].first;
67             }
68             ans+=s.det(t)*tot;
69         }
70     }
71     return fabs(ans)*0.5;
72 }
73 };
```

5 graph

5.1 2-sat

```

1 #define maxn 2008
2 struct Twosat
3 {
4     int n;
5     std::vector<int> G[maxn*2];
6     bool mark[maxn*2];
7     int s[maxn*2],c;
8
9     bool dfs(int x)
10    {
11        if(mark[x^1]) return false;
12        if(mark[x]) return true;
13        mark[x]=true;
14        s[c++]=x;
15        for(int i=0;i<G[x].size();i++)
16        {
17            if(!dfs(G[x][i])) return false;
18            return true;
19        }
20
21        void init(int n)
22        {
23            this->n=n;
24            for(int i=0;i<n*2;i++)
25                G[i].clear();
26            memset(mark,0,sizeof(mark));
27        }
28        void add_clause(int x,int xval,int y,int yval)//
29        {
30            x=x*2+xval;
31            y=y*2+yval;
32            G[x^1].push_back(y);
33            G[y^1].push_back(x);
34        }
35        bool solve()
36        {
37            for(int i=0;i<n*2;i+=2)
```

```

38         if(!mark[i]&&!mark[i+1])
39         {
40             c=0;
41             if(!dfs(i))
42             {
43                 while(c>0)
44                     mark[s[--c]]=false;
45                 if(!dfs(i+1))
46                     return false;
47             }
48             return true;
49         }
50     }
51 };
```

5.2 Articulation

```

1 void dfs(int now,int fa) // now1
2 {
3     int p(0);
4     dfn[now]=low[now]=cnt++;
5     for(std::list<int>::const_iterator it(edge[now].begin());it!=
        edge[now].end();++it)
6     {
7         if(dfn[*it]==-1)
8         {
9             dfs(*it,now);
10            ++p;
11            low[now]=std::min(low[now],low[*it]);
12            if((now==1 && p>1) || (now!=1 && low[*it]>=dfn[now])) //
13                ans.insert(now);
14            }
15            else
16            if(*it!=fa)
17                low[now]=std::min(low[now],dfn[*it]);
18 }
```

5.3 Augmenting Path Algorithm for Maximum Cardinality Bipartite Matching

```

1 #include<cstdio>
2 #include<cstring>
3
4 #define MAXX 111
5
6 bool Map[MAXX][MAXX],visit[MAXX];
7 int link[MAXX],n,m;
8 bool dfs(int t)
9 {
10     for (int i=0; i<m; i++)
11     {
12         if (!visit[i] && Map[t][i]){
13             visit[i] = true;
14             if (link[i]==-1 || dfs(link[i])){
15                 link[i] = t;
16                 return true;
17             }
18         }
19     }
20     return false;
21 }
22 int main()
23 {
24     int k,a,b,c;
25     while (scanf("%d",&n),n){
26         memset(Map,false,sizeof(Map));
27         scanf("%d",&m,&k);
28         while (k--){
29             scanf("%d%d",&a,&b,&c);
30             if (b && c)
31                 Map[b][c] = true;
32         }
33         memset(link,-1,sizeof(link));
34         int ans = 0;
35         for (int i=0; i<n; i++){
36             memset(visit,false,sizeof(visit));
37             if (dfs(i))
38                 ans++;
39         }
40         printf("%d\n",ans);
41 }
```

5.4 Biconnected Component - Edge

```

1 // hdu 4612
2 #include<stdio>
3 #include<algorithm>
4 #include<set>
5 #include<string>
6 #include<stack>
7 #include<queue>
8
9 #define MAXX 20011
10 #define MAXE (100011*2)
11 #pragma comment(linker, "/STACK:16777216")
12
13 int edge[MAXX],to[MAXE],nxt[MAXE],cnt;
14 #define v to[i]
15 inline void add(int a,int b)
16 {
17     nxt[++cnt]=edge[a];
```

```

18     edge[a]=cnt;
19     to[cnt]=b;
20 }
21
22 int dfn[MAXX], low[MAXX], col[MAXX], belong[MAXX];
23 int idx, bcnt;
24 std::stack<int> st;
25
26 void tarjan(int now, int last)
27 {
28     col[now]=1;
29     st.push(now);
30     dfn[now]=low[now]=++idx;
31     bool flag(false);
32     for(int i=edge[now]; i; i=nxt[i])
33     {
34         if(v==last && !flag)
35         {
36             flag=true;
37             continue;
38         }
39         if(!col[v])
40         {
41             tarjan(v, now);
42             low[now]=std::min(low[now], low[v]);
43             /*
44             if (low[v]>dfn[now])
45             then this is a bridge
46             */
47         }
48         else
49             if(col[v]==1)
50                 low[now]=std::min(low[now], dfn[v]);
51     }
52     col[now]=2;
53     if(dfn[now]==low[now])
54     {
55         ++bcnt;
56         static int x;
57         do
58         {
59             x=st.top();
60             st.pop();
61             belong[x]=bcnt;
62         } while(x!=now);
63     }
64 }
65
66 std::set<int> set[MAXX];
67
68 int dist[MAXX];
69 std::queue<int> q;
70 int n, m, i, j, k;
71
72 inline int go(int s)
73 {
74     static std::set<int>::const_iterator it;
75     memset(dist, 0x3f, sizeof dist);
76     dist[s]=0;
77     q.push(s);
78     while(!q.empty())
79     {
80         s=q.front();
81         q.pop();
82         for(it=set[s].begin(); it!=set[s].end(); ++it)
83             if(dist[*it]>dist[s]+1)
84             {
85                 dist[*it]=dist[s]+1;
86                 q.push(*it);
87             }
88     }
89     return std::max_element(dist+1, dist+1+bcnt)-dist;
90 }
91
92 int main()
93 {
94     while(scanf("%d %d", &n, &m), (n||m))
95     {
96         cnt=0;
97         memset(edge, 0, sizeof edge);
98         while(m--)
99         {
100             scanf("%d %d", &i, &j);
101             add(i, j);
102             add(j, i);
103         }
104
105         memset(dfn, 0, sizeof dfn);
106         memset(belong, 0, sizeof belong);
107         memset(low, 0, sizeof low);
108         memset(col, 0, sizeof col);
109         bcnt=idx=0;
110         while(!st.empty())
111             st.pop();
112
113         tarjan(1, -1);
114         for(i=1; i<=bcnt; ++i)
115             set[i].clear();
116         for(i=1; i<=n; ++i)
117             for(j=edge[i]; j; j=nxt[j])
118                 set[belong[i]].insert(belong[to[j]]);
119         for(i=1; i<=bcnt; ++i)
120             set[i].erase(i);
121         /*
122         printf("%d\n", dist[go(1)]);
123         for(i=1; i<=bcnt; ++i)
124             printf("%d\n", dist[i]);
125         puts("");
126         */
127         printf("%d\n", bcnt-1-dist[go(1)]);
128     }
129     return 0;
130 }

```

5.5 Biconnected Component

```

1  #include<cstdio>
2  #include<cstring>
3  #include<stack>
4  #include<queue>
5  #include<algorithm>
6
7  const int MAXN=100000*2;
8  const int MAXM=200000;
9
10 //0-based
11
12 struct edges
13 {
14     int to, next;
15     bool cut, visit;
16 } edge[MAXN<1];
17
18 int head[MAXN], low[MAXN], dpt[MAXN], L;
19 bool visit[MAXN], cut[MAXN];
20 int idx;
21 std::stack<int> st;
22 int bcc[MAXN];
23
24 void init(int n)
25 {
26     L=0;
27     memset(head, -1, 4*n);
28     memset(visit, 0, n);
29 }
30
31 void add_edge(int u, int v)
32 {
33     edge[L].cut=edge[L].visit=false;
34     edge[L].to=v;
35     edge[L].next=head[u];
36     head[u]=L++;
37 }
38
39 void dfs(int u, int fu, int deg)
40 {
41     cut[u]=false;
42     visit[u]=true;
43     low[u]=dpt[u]=deg;
44     int tot=0;
45     for(int i=head[u]; i!=-1; i=edge[i].next)
46     {
47         int v=edge[i].to;
48         if(edge[i].visit)
49             continue;
50         st.push(i/2);
51         edge[i].visit=edge[i^1].visit=true;
52         if(visit[v])
53         {
54             low[u]=dpt[v]>low[u]?low[u]:dpt[v];
55             continue;
56         }
57         dfs(v, u, deg+1);
58         edge[i].cut=edge[i^1].cut=(low[v]>dpt[u] || edge[i].cut);
59         if(u!=fu) cut[u]=low[v]>=dpt[u]?1:cut[u];
60         if(low[v]>=dpt[u] || u==fu)
61         {
62             while(st.top()!=i/2)
63             {
64                 int x=st.top()*2, y=st.top()*2+1;
65                 bcc[st.top()]=idx;
66                 st.pop();
67             }
68             bcc[i/2]=idx++;
69             st.pop();
70         }
71         low[u]=low[v]>low[u]?low[u]:low[v];
72         tot++;
73     }
74     if(u==fu && tot>1)
75         cut[u]=true;
76 }
77
78 int main()
79 {
80     int n, m;
81     while(scanf("%d%d", &n, &m) != EOF)
82     {
83         init(n);
84         for(int i=0; i<m; i++)
85         {
86             int u, v;
87             scanf("%d%d", &u, &v);
88             add_edge(u, v);
89             add_edge(v, u);
90         }
91         idx=0;
92         for(int i=0; i<n; i++)
93             if(!visit[i])
94                 dfs(i, i, 0);
95     }
96     return 0;
97 }

```

5.6 Blossom algorithm

```

1  #include<cstdio>
2  #include<vector>
3  #include<cstring>
4  #include<algorithm>
5
6  #define MAXX 233
7
8  bool map[MAXX][MAXX];
9  std::vector<int> p[MAXX];
10 int m[MAXX];

```

```

11 int vis[MAXX];
12 int q[MAXX],*qf,*qb;
13
14 int n;
15
16 inline void label(int x,int y,int b)
17 {
18     static int i,z;
19     for(i=b+1;i<p[x].size();++i)
20         if(vis[z=p[x][i]]==1)
21         {
22             p[z]=p[y];
23             p[z].insert(p[z].end(),p[x].rbegin(),p[x].rend()-i);
24             vis[z]=0;
25             *qb++=z;
26         }
27 }
28
29 inline bool bfs(int now)
30 {
31     static int i,x,y,z,b;
32     for(i=0;i<n;++i)
33         p[i].resize(0);
34     p[now].push_back(now);
35     memset(vis,-1,sizeof vis);
36     vis[now]=0;
37     qf=qb=q;
38     *qb++=now;
39
40     while(qf<qb)
41         for(x=*qf++;y=0;y<n;++y)
42             if(map[x][y] && m[y]!=y && vis[y]!=1)
43             {
44                 if(vis[y]==-1)
45                     if(m[y]==-1)
46                     {
47                         for(i=0;i+1<p[x].size();i+=2)
48                         {
49                             m[p[x][i]]=p[x][i+1];
50                             m[p[x][i+1]]=p[x][i];
51                         }
52                         m[x]=y;
53                         m[y]=x;
54                         return true;
55                     }
56                 else
57                 {
58                     p[z=m[y]]=p[x];
59                     p[z].push_back(y);
60                     p[z].push_back(z);
61                     vis[y]=1;
62                     vis[z]=0;
63                     *qb++=z;
64                 }
65             }
66     else
67     {
68         for(b=0;b<p[x].size() && b<p[y].size() && p[x][b]==
69             p[y][b];++b);
70         label(x,y,b);
71         label(y,x,b);
72     }
73     return false;
74 }
75
76 int i,j,k;
77 int ans;
78
79 int main()
80 {
81     scanf("%d",&n);
82     for(i=0;i<n;++i)
83         p[i].reserve(n);
84     while(scanf("%d %d",&i,&j)!=EOF)
85     {
86         --i;
87         --j;
88         map[i][j]=map[j][i]=true;
89     }
90     memset(m,-1,sizeof m);
91     for(i=0;i<n;++i)
92         if(m[i]==-1)
93         {
94             if(bfs(i))
95                 ++ans;
96             else
97                 m[i]=i;
98         }
99     printf("%d\n",ans<<1);
100     for(i=0;i<n;++i)
101         if(i<m[i])
102             printf("%d %d\n",i+1,m[i]+1);
103     return 0;
104 }

```

5.7 Bridge

```

1 void dfs(const short &now,const short &fa)
2 {
3     dfn[now]=low[now]=cnt++;
4     for(int i(0);i<edge[now].size();++i)
5         if(dfn[edge[now][i]]==--1)
6         {
7             dfs(edge[now][i],now);
8             low[now]=std::min(low[now],low[edge[now][i]]);
9             if(low[edge[now][i]]>dfn[now]) //,
10             {
11                 if(edge[now][i]<now)
12                 {
13                     j=edge[now][i];
14                     k=now;
15                 }
16                 else

```

```

17         {
18             j=now;
19             k=edge[now][i];
20         }
21         ans.push_back(node(j,k));
22     }
23 }
24 else
25     if(edge[now][i]!=fa)
26         low[now]=std::min(low[now],low[edge[now][i]]);
27 }

```

5.8 chu-liu algorithm

```

1 #include<cstdio>
2 #include<cstring>
3 #include<algorithm>
4
5 const int inf = 0x5fffffff;
6
7 int n,m,u,v,cost,dis[1001][1001],L;
8 int pre[1001],id[1001],visit[1001],in[1001];
9
10 void init(int n)
11 {
12     L = 0;
13     for(int i = 0; i < n; i++)
14         for(int j = 0; j < n; j++)
15             dis[i][j] = inf;
16 }
17
18 struct Edge
19 {
20     int u,v,cost;
21 };
22
23 Edge e[1001*1001];
24
25 int zhuliu(int root,int n,int m,Edge e[])
26 {
27     int res = 0,u,v;
28     while (true)
29     {
30         for(int i = 0; i < n; i++)
31             in[i] = inf;
32         for(int i = 0; i < m; i++)
33             if (e[i].u != e[i].v && e[i].cost < in[e[i].v])
34             {
35                 pre[e[i].v] = e[i].u;
36                 in[e[i].v] = e[i].cost;
37             }
38         for(int i = 0; i < n; i++)
39             if (i != root)
40                 if (in[i] == inf)
41                     return -1;
42         int tn = 0;
43         memset(id,-1,sizeof(id));
44         memset(visit,-1,sizeof(visit));
45         in[root] = 0;
46         for(int i = 0; i < n; i++)
47         {
48             res += in[i];
49             v = i;
50             while (visit[v] != i && id[v] == -1 && v != root)
51             {
52                 visit[v] = i;
53                 v = pre[v];
54             }
55             if(v != root && id[v] == -1)
56             {
57                 for(int u = pre[v] ; u != v ; u = pre[u])
58                     id[u] = tn;
59                 id[v] = tn++;
60             }
61         }
62         if(tn == 0) break;
63         for(int i = 0; i < n; i++)
64             if (id[i] == -1)
65                 id[i] = tn++;
66         for(int i = 0; i < m; i++)
67         {
68             int v = e[i].v;
69             e[i].u = id[e[i].u];
70             e[i].v = id[e[i].v];
71             if (e[i].u != e[i].v)
72                 e[i++].cost -= in[v];
73             else
74                 std::swap(e[i],e[--m]);
75         }
76         n = tn;
77         root = id[root];
78     }
79     return res;
80 }
81
82 int main()
83 {
84     while (scanf("%d%d",&n,&m) != EOF)
85     {
86         init(n);
87         for(int i = 0; i < m; i++)
88         {
89             scanf("%d%d%d",&u,&v,&cost);
90             if (u == v) continue;
91             dis[u][v] = std::min(dis[u][v],cost);
92         }
93         L = 0;
94         for(int i = 0; i < n; i++)
95             for(int j = 0; j < n; j++)
96                 if (dis[i][j] != inf)
97                 {
98                     e[L].u = i;
99                     e[L].v = j;
100

```

```

101         e[L++].cost = dis[i][j];
102     }
103     printf("%d\n", zhuliu(0, n, L, e));
104 }
105     return 0;
106 }

```

5.9 Covering problems

```

1
2
3
4    , , .
5
6    SGus, ssssG
7
8    ()
9
10   GDDuvu,vu,vG=G = (V;E)UVu,vU < u; v >EUGGUGGGGUVu; vU< u; v >
      EUGGUGGGG
11
12
13   += V
14   =
15   =
16
17   minimum cover:
18   vertex cover vertex bipartite graph = maximum cardinality
      bipartite matching
19
20   X
21   Y
22
23   Graph Traversal
24
25   Graph Traversal
26
27   vertex cover edge
28
29   edge cover vertex
30   Maximum Matching Minimum Edge Cover
31
32   edge cover edge
33
34   path cover vertex
35   general graph: NP-H
36   tree: DP
37   DAG: ,ans=-
38
39   path cover edge
40   minimize the count of euler path ( greedy is ok? )
41
42   cycle cover vertex
43   general: NP-H
44   weighted: do like path cover vertex, with KM algorithm
45
46   cycle cover edge
47   NP-H

```

5.10 Difference constraints

```

1   for a - b <= c
2       add(b,a,c);
3
4
5
6   //(?)
7
8   0

```

5.11 Dinitz's algorithm

```

1   #include<stdio>
2   #include<algorithm>
3   #include<string>
4
5   #define MAXX 111
6   #define MAXM (MAXX*MAXX*4)
7   #define inf 0x3f3f3f3f
8
9   int n;
10  int w[MAXX], h[MAXX], q[MAXX];
11  int edge[MAXX], to[MAXM], cap[MAXM], nxt[MAXM], cnt;
12  int source, sink;
13
14  inline void add(int a, int b, int c)
15  {
16      nxt[cnt]=edge[a];
17      edge[a]=cnt;
18      to[cnt]=b;
19      cap[cnt]=c;
20      ++cnt;
21  }
22
23  inline bool bfs()
24  {
25      static int *qf, *qb;
26      static int i;
27      memset(h, -1, sizeof h);
28      qf=qb=q;
29      h[*qb++]=source]=0;
30      for(; qf!=qb; ++qf)

```

```

31         for(i=edge[*qf]; i!=-1; i=nxt[i])
32             if(cap[i] && h[to[i]]==-1)
33                 h[*qb++=to[i]]=h[*qf]+1;
34         return h[sink]!=-1;
35     }
36
37     int dfs(int now, int maxcap)
38     {
39         if(now==sink)
40             return maxcap;
41         int flow(maxcap), d;
42         for(int &i(w[now]); i!=-1; i=nxt[i])
43             if(cap[i] && h[to[i]]==h[now]+1) // && (flow=dfs(to[i], std::
                min(maxcap, cap[i])))
44             {
45                 d=dfs(to[i], std::min(flow, cap[i]));
46                 cap[i]-=d;
47                 cap[i^1]+=d;
48                 flow-=d;
49                 if(!flow)
50                     return maxcap;
51             }
52         return maxcap-flow;
53     }
54
55     int nc, np, m, i, j, k;
56     int ans;
57
58     int main()
59     {
60         while(scanf("%d %d %d %d", &n, &np, &nc, &m) != EOF)
61         {
62             cnt=0;
63             memset(edge, -1, sizeof edge);
64             while(m--)
65             {
66                 while(getchar() != ' ');
67                 scanf("%d", &i);
68                 while(getchar() != ' ');
69                 scanf("%d", &j);
70                 while(getchar() != ' ');
71                 scanf("%d", &k);
72                 if(i!=j)
73                 {
74                     ++i;
75                     ++j;
76                     add(i, j, k);
77                     add(j, i, 0);
78                 }
79             }
80             source=++n;
81             while(np--)
82             {
83                 while(getchar() != ' ');
84                 scanf("%d", &i);
85                 while(getchar() != ' ');
86                 scanf("%d", &j);
87                 ++i;
88                 add(source, i, j);
89                 add(i, source, 0);
90             }
91             sink=++n;
92             while(nc--)
93             {
94                 while(getchar() != ' ');
95                 scanf("%d", &i);
96                 while(getchar() != ' ');
97                 scanf("%d", &j);
98                 ++i;
99                 add(i, sink, j);
100                add(sink, i, 0);
101            }
102            ans=0;
103            while(bfs())
104            {
105                memcpy(w, edge, sizeof edge);
106                ans+=dfs(source, inf);
107                /*
108                while((k=dfs(source, inf))
109                    ans+=k;
110                */
111            }
112            printf("%d\n", ans);
113        }
114        return 0;
115    }

```

5.12 Feasible flow problem

```

1   #include<stdio>
2   #include<string>
3   #include<algorithm>
4
5   #define MAXX (255)
6   #define inf 0x3f3f3f3f
7
8   int cap[MAXX][MAXX];
9   int h[MAXX];
10  int last[MAXX];
11  int source, sink;
12
13  int mat[MAXX][MAXX][2];
14  bool bg, flag;
15
16  int n;
17
18  inline bool bfs()
19  {
20      static int q[MAXX], *qf, *qb, i;
21      memset(h, -1, sizeof h);
22      qf=qb=q;
23      for(h[*qb++=source]=0; qf!=qb; ++qf)
24          for(i=1; i<=n; ++i)
25              if(cap[*qf][i] && h[i]==-1)

```

```

26     {
27         h[*qb++i]=h[*qf]+1;
28         if(i==sink)
29             return true;
30     }
31     return false;
32 }
33
34 int dfs(int now,int maxcap)
35 {
36     if(now==sink)
37         return maxcap;
38     for(int i(last[now]),f;i<=n;++i)
39         if(cap[now][i] && h[i]==h[now]+1 && (f=dfs(i,std::min(maxcap
40             ,cap[now][i]))))
41         {
42             cap[now][i]-=f;
43             cap[i][now]+=f;
44             return f;
45         }
46     return 0;
47 }
48
49 int T;
50 int m,i,j,k,c;
51 int s,t,a,b;
52 int sr[MAXX],sc[MAXX];
53 char buf[11];
54
55 inline void gao(int x,int y)
56 {
57     switch(buf[0])
58     {
59         case '>':
60             mat[x][y][0]=std::max(mat[x][y][0],k+1);
61             if(mat[x][y][0]>mat[x][y][1])
62                 flag=true;
63                 break;
64         case '=':
65             if(k<mat[x][y][0] || k>mat[x][y][1])
66                 flag=true;
67                 mat[x][y][0]=mat[x][y][1]=k;
68                 break;
69         case '<':
70             mat[x][y][1]=std::min(mat[x][y][1],k-1);
71             if(mat[x][y][0]>mat[x][y][1])
72                 flag=true;
73                 break;
74     }
75 }
76
77 int main()
78 {
79     bg=true;
80     scanf("%d",&T);
81     while(T--)
82     {
83         if(!bg)
84             puts("");
85         memset(mat,0,sizeof mat);
86         scanf("%d %d",&n,&m);
87         for(i=1;i<=n;++i)
88             scanf("%d",&sr[i]);
89         for(i=1;i<=n;++i)
90             scanf("%d",&sc[i]);
91         s=n+m+1;
92         t=s+1;
93         source=t+1;
94         sink=source+1;
95         for(i=1;i<=n;++i)
96             for(j=1;j<=m;++j)
97             {
98                 mat[i][j+n][0]=0;
99                 mat[i][j+n][1]=inf;
100             }
101         bg=flag=false;
102         scanf("%d",&c);
103         while(c--)
104         {
105             scanf("%d %d %s %d",&i,&j,buf,&k);
106             if(i)
107                 if(j)
108                     gao(i,j+n);
109                 else
110                     for(j=1;j<=m;++j)
111                         gao(i,j+n);
112             else
113                 if(j)
114                     for(i=1;i<=n;++i)
115                         gao(i,j+n);
116                 else
117                     for(i=1;i<=n;++i)
118                         for(j=1;j<=m;++j)
119                             gao(i,j+n);
120         }
121         if(flag)
122         {
123             puts("IMPOSSIBLE");
124             continue;
125         }
126         memset(cap,0,sizeof cap);
127         for(i=1;i<=n;++i)
128             mat[s][i][0]=mat[s][i][1]=sr[i];
129         for(i=1;i<=m;++i)
130             mat[i+n][t][0]=mat[i+n][t][1]=sc[i];
131
132         a=0;
133         for(i=1;i<=t;++i)
134         {
135             b=0;
136             for(j=1;j<=t;++j)
137             {
138                 b+=mat[j][i][0]-mat[i][j][0];
139                 cap[i][j]=mat[i][j][1]-mat[i][j][0];
140             }
141             if(b>0)
142                 a+=(cap[source][i]=b);
143             else
144                 cap[i][sink]=-b;
145         }

```

```

145     cap[t][s]=inf;
146     c=n;
147     n=sink;
148     for(i=1;i<=n;++i)
149         last[i]=1;
150     for(b=0;bfs();b+=dfs(source,inf));
151
152     // printf("%d %d\n",a,b);
153     if(a!=b)
154         puts("IMPOSSIBLE");
155     else
156     {
157         n=c;
158         for(i=1;i<=n;++i)
159         {
160             for(j=1;j<=m;++j)
161                 printf("%d ",cap[j+n][i]+mat[i][j+n][0]);
162             puts("");
163         }
164     }
165     return 0;
166 }
167

```

5.13 Flow network

```

1 Maximum weighted closure of a graph:
2
3 closure
4
5 inf
6
7
8
9 sum()-{}
10
11
12
13 Eulerian circuit:
14
15
16
17
18 :
19 1 //
20 1
21 abs(/2)
22 abs(/2)
23
24 trick
25
26 pathcircuit
27
28
29
30 Feasible flow problem:
31 refer Feasible flow problem.cpp
32
33 0inf
34
35 <a->b cap(u,d)><ss->b cap(u)><a->st cap(u)><a->b cap(d-u)>
36
37 Maximum flow: //
38
39 Minimum flow: //
40 t->s
41 ->
42 tips:
43
44
45
46
47 Minimum cost feasible flow problem:
48 TODO
49
50
51
52
53 Minimum weighted vertex cover edge for bipartite graph:
54 for all vertex in X:
55 edge < s->x cap(weight(x)) >
56 for all vertex in Y:
57 edge < y->t cap(weight(y)) >
58 for original edges
59 edge < x->y cap(inf) >
60
61 ans=(maximum flow)={minimum cut}
62 ( ( && ) || ( && ) )
63
64
65
66 Maximum weighted vertex independent set for bipartite graph:
67 ans=Sum()-value(Minimum weighted vertex cover edge)
68
69
70
71
72 :
73
74
75
76 /inf
77
78 ans=sum()-{}
79
80
81
82 :
83
84
85 cap[i`1]cap[i]
86
87
88

```

```

89 void rr(int now)
90 {
91     done[now]=true;
92     ++cnt;
93     for(int i=edge[now];i!=-1;i=nxt[i])
94         if(cap[i] && !done[v])
95             rr(v);
96 }
97
98 void dfs(int now)
99 {
100     done[now]=true;
101     ++cnt;
102     for(int i=edge[now];i!=-1;i=nxt[i])
103         if(cap[i^1] && !done[v])
104             dfs(v);
105 }
106
107 memset(done,0,sizeof done);
108 cnt=0;
109 rr(source);
110 dfs(sink);
111 puts(cnt==n?"UNIQUE":"AMBIGUOUS");
112
113
114
115 Tips:
116 ;
117 inf;
118 ;
119 inf;

```

```

81     }
82     next[b]=a;
83     for(i=a;i!=b;i=next[i])
84         if(find(i))
85             {
86                 a=next[b=i];
87                 break;
88             }
89     }
90     while(a!=b)
91     {
92         printf("%d ",a);
93         a=next[a];
94     }
95     printf("%d\n",b);
96 }
97     return 0;
98 }

```

5.15 Hopcroft-Karp algorithm

```

1  #include<cstdio>
2  #include<cstring>
3
4  #define MAXX 50111
5  #define MAX 150111
6
7  int nx,p;
8  int i,j,k;
9  int x,y;
10 int ans;
11 bool flag;
12
13 int edge[MAXX],nxt[MAX],to[MAX],cnt;
14
15 int cx[MAXX],cy[MAXX];
16 int px[MAXX],py[MAXX];
17
18 int q[MAXX],*qf,*qb;
19
20 bool ag(int i)
21 {
22     int j,k;
23     for(k=edge[i];k;nxt[k])
24         if(py[j=to[k]]==px[i]+1)
25             {
26                 py[j]=0;
27                 if(cy[j]==-1 || ag(cy[j]))
28                     {
29                         cx[i]=j;
30                         cy[j]=i;
31                         return true;
32                     }
33             }
34     return false;
35 }
36
37 int main()
38 {
39     scanf("%d %d %d",&nx,&p);
40     while(p--)
41     {
42         scanf("%d %d",&i,&j);
43         nxt[++cnt]=edge[i];
44         edge[i]=cnt;
45         to[cnt]=j;
46     }
47     memset(cx,-1,sizeof cx);
48     memset(cy,-1,sizeof cy);
49     while(true)
50     {
51         memset(px,0,sizeof px);
52         memset(py,0,sizeof py);
53         qf=qb=q;
54         flag=false;
55
56         for(i=1;i<=nx;++i)
57             if(cx[i]==-1)
58                 *qb+=i;
59         while(qf!=qb)
60             for(k=edge[i=*qf++];k;nxt[k])
61                 if(!py[j=to[k]])
62                     {
63                         py[j]=px[i]+1;
64                         if(cy[j]==-1)
65                             flag=true;
66                         else
67                             {
68                                 px[cy[j]]=py[j]+1;
69                                 *qb+=cy[j];
70                             }
71                     }
72         if(!flag)
73             break;
74         for(i=1;i<=nx;++i)
75             if(cx[i]==-1 && ag(i))
76                 ++ans;
77     }
78     printf("%d\n",ans);
79     return 0;
80 }

```

5.14 Hamiltonian circuit

```

1  //if every point connect with not less than [(N+1)/2] points
2  #include<cstdio>
3  #include<algorithm>
4  #include<cstring>
5
6  #define MAXX 177
7  #define MAX (MAXX*MAXX)
8
9  int edge[MAXX],nxt[MAX],to[MAX],cnt;
10
11 inline void add(int a,int b)
12 {
13     nxt[++cnt]=edge[a];
14     edge[a]=cnt;
15     to[cnt]=b;
16 }
17
18 bool done[MAXX];
19 int n,m,i,j,k;
20
21 inline int find(int a)
22 {
23     static int i;
24     for(i=edge[a];i;nxt[i])
25         if(!done[to[i]])
26             {
27                 edge[a]=nxt[i];
28                 return to[i];
29             }
30     return 0;
31 }
32
33 int a,b;
34 int next[MAXX],pre[MAXX];
35 bool mat[MAXX][MAXX];
36
37 int main()
38 {
39     while(scanf("%d %d",&n,&m)!=EOF)
40     {
41         for(i=1;i<=n;++i)
42             next[i]=done[i]=edge[i]=0;
43         memset(mat,0,sizeof mat);
44         cnt=0;
45         while(m--)
46             {
47                 scanf("%d %d",&i,&j);
48                 add(i,j);
49                 add(j,i);
50                 mat[i][j]=mat[j][i]=true;
51             }
52         a=1;
53         b=to[edge[a]];
54         cnt=2;
55         done[a]=done[b]=true;
56         next[a]=b;
57         while(cnt<n)
58             {
59                 while(i=find(a))
60                 {
61                     next[i]=a;
62                     done[a=i]=true;
63                     ++cnt;
64                 }
65                 while(i=find(b))
66                 {
67                     next[b]=i;
68                     done[b=i]=true;
69                     ++cnt;
70                 }
71                 if(!mat[a][b])
72                     for(i=next[a];next[i]!=b;i=next[i])
73                         if(mat[a][next[i]] && mat[i][b])
74                             {
75                                 for(j=next[i];j!=b;j=next[j])
76                                     pre[next[j]]=j;
77                                 for(j=b;j!=next[i];j=pre[j])
78                                     next[j]=pre[j];
79                                 std::swap(next[i],b);
80                                 break;

```

5.16 Improved Shortest Augmenting Path Algorithm

```

1  #include<cstdio>
2  #include<cstring>

```



```

3  #include<algorithm>
4
5  #define MAXX 5111
6  #define MAXM (30111*4)
7  #define inf 0x3f3f3f3f3f3f3f11
8
9  int edge[MAXX],to[MAXM],nxt[MAXM],cnt;
10 #define v to[i]
11 long long cap[MAXM];
12
13 int n;
14 int h[MAXX],gap[MAXX],pre[MAXX],w[MAXX];
15
16 inline void add(int a,int b,long long c)
17 {
18     nxt[++cnt]=edge[a];
19     edge[a]=cnt;
20     to[cnt]=b;
21     cap[cnt]=c;
22 }
23
24 int source,sink;
25
26 inline long long go()
27 {
28     static int now,N,i;
29     static long long min,mf;
30     memset(gap,0,sizeof gap);
31     memset(h,0,sizeof h);
32     memcpy(w,edge,sizeof w);
33     gap[0]=N=sink; // caution
34     mf=0;
35
36     pre[now=source]=-1;
37     while(h[source]<N)
38     {
39         if(now==sink)
40         {
41             min=inf;
42             for(i=pre[now];i!=-1;i=pre[to[i^1]])
43                 min=std::min(min,cap[i]);
44             for(i=pre[now];i!=-1;i=pre[to[i^1]])
45             {
46                 cap[i]=min;
47                 cap[i^1]+=min;
48             }
49             now=source;
50             mf+=min;
51         }
52         for(i=w[now];i!=-1;i=nxt[i])
53             if(cap[i] && h[v]+i==h[now])
54             {
55                 w[now]=pre[v]=i;
56                 now=v;
57                 break;
58             }
59         if(i!=-1)
60             continue;
61         if(!--gap[h[now]])
62             return mf;
63         min=N;
64         for(i=w[now]=edge[now];i!=-1;i=nxt[i])
65             if(cap[i])
66                 min=std::min(min,(long long)h[v]);
67         ++gap[h[now]=min+1];
68         if(now!=source)
69             now=to[pre[now]^1];
70     }
71     return mf;
72 }
73
74 int m,i,j,k;
75 long long ans;
76
77 int main()
78 {
79     scanf("%d %d",&n,&m);
80     source=1;
81     sink=n;
82     cnt=-1;
83     memset(edge,-1,sizeof edge);
84     while(m--)
85     {
86         scanf("%d %d %lld",&i,&j,&ans);
87         add(i,j,ans);
88         add(j,i,ans);
89     }
90     printf("%lld\n",go());
91     return 0;
92 }

```

5.17 k Shortest Path

```

1  #include<cstdio>
2  #include<cstring>
3  #include<queue>
4  #include<vector>
5
6  int K;
7
8  class states
9  {
10 public:
11     int cost,id;
12 };
13
14 int dist[1000];
15
16 class cmp
17 {
18 public:
19     bool operator () (const states &i,const states &j)
20     {
21         return i.cost>j.cost;
22     }
23 }

```

```

22     }
23 };
24
25 class cmp2
26 {
27 public:
28     bool operator () (const states &i,const states &j)
29     {
30         return i.cost+dist[i.id]>j.cost+dist[j.id];
31     }
32 };
33
34 struct edges
35 {
36     int to,next,cost;
37 } edger[100000],edge[100000];
38
39 int headr[1000],head[1000],Lr,L;
40
41 void dijkstra(int s)
42 {
43     states u;
44     u.id=s;
45     u.cost=0;
46     dist[s]=0;
47     std::priority_queue<states,std::vector<states>,cmp> q;
48     q.push(u);
49     while (!q.empty())
50     {
51         u=q.top();
52         q.pop();
53         if (u.cost!=dist[u.id])
54             continue;
55         for (int i=headr[u.id]; i!=-1; i=edger[i].next)
56         {
57             states v=u;
58             v.id=edger[i].to;
59             if (dist[v.id]>dist[u.id]+edger[i].cost)
60             {
61                 v.cost=dist[v.id]=dist[u.id]+edger[i].cost;
62                 q.push(v);
63             }
64         }
65     }
66 }
67
68 int num[1000];
69
70 inline void init(int n)
71 {
72     Lr=L=0;
73     memset(head,-1,4*n);
74     memset(headr,-1,4*n);
75     memset(dist,63,4*n);
76     memset(num,0,4*n);
77 }
78
79 void add_edge(int u,int v,int x)
80 {
81     edge[L].to=v;
82     edge[L].cost=x;
83     edge[L].next=head[u];
84     head[u]=L++;
85     edger[Lr].to=u;
86     edger[Lr].cost=x;
87     edger[Lr].next=headr[v];
88     headr[v]=Lr++;
89 }
90
91 inline int a_star(int s,int t)
92 {
93     if (dist[s]==0x3f3f3f3f)
94         return -1;
95     std::priority_queue<states,std::vector<states>,cmp2> q;
96     states tmp;
97     tmp.id=s;
98     tmp.cost=0;
99     q.push(tmp);
100     while (!q.empty())
101     {
102         states u=q.top();
103         q.pop();
104         num[u.id]++;
105         if (num[t]==K)
106             return u.cost;
107         for (int i=head[u.id]; i!=-1; i=edge[i].next)
108         {
109             int v=edge[i].to;
110             tmp.id=v;
111             tmp.cost=u.cost+edge[i].cost;
112             q.push(tmp);
113         }
114     }
115     return -1;
116 }
117
118 int main()
119 {
120     int n,m;
121     scanf("%d %d",&n,&m);
122     init(n);
123     for (int i=0; i<m; i++)
124     {
125         int u,v,x;
126         scanf("%d %d %d",&u,&v,&x);
127         add_edge(u-1,v-1,x);
128     }
129     int s,t;
130     scanf("%d %d %d",&s,&t,&K);
131     if (s==t)
132         ++K;
133     dijkstra(t-1);
134     printf("%d\n",a_star(s-1,t-1));
135 }

```

5.18 Kariv-Hakimi Algorithm

```

1  #include<cstdio>
2  #include<algorithm>
3  #include<vector>
4  #include<cstring>
5  #include<set>
6
7  #define MAXX 211
8  #define inf 0x3f3f3f3f
9
10 int e[MAXX][MAXX],dist[MAXX][MAXX];
11 double dp[MAXX],ta;
12 int ans,d;
13 int n,m,a,b;
14 int i,j,k;
15 typedef std::pair<int,int> pii;
16 std::vector<pii>vt[2];
17 bool done[MAXX];
18 typedef std::pair<double,int> pdi;
19 std::multiset<pdi>q;
20 int pre[MAXX];
21
22 int main()
23 {
24     vt[0].reserve(MAXX);
25     vt[1].reserve(MAXX);
26     scanf("%d %d",&n,&m);
27     memset(e,0x3f,sizeof(e));
28     while(m-->0)
29     {
30         scanf("%d %d %d",&i,&j,&k);
31         e[i][j]=e[j][i]=std::min(e[i][j],k);
32     }
33     for(i=1;i<=n;++i)
34         e[i][i]=0;
35     memcpy(dist,e,sizeof(dist));
36     for(k=1;k<=n;++k)
37         for(i=1;i<=n;++i)
38             for(j=1;j<=n;++j)
39                 dist[i][j]=std::min(dist[i][j],dist[i][k]+dist[k][j]);
40     ans=inf;
41     for(i=1;i<=n;++i)
42         for(j=1;j<=n;++j)
43             if(e[i][j]!=inf)
44             {
45                 vt[0].resize(0);
46                 vt[1].resize(0);
47                 static int i;
48                 for(i=1;i<=n;++i)
49                     vt[0].push_back(pii(dist[i][i],dist[j][i]));
50                 std::sort(vt[0].begin(),vt[0].end());
51                 for(i=0;i<vt[0].size();++i)
52                 {
53                     while(!vt[1].empty() && vt[1].back().second<=vt[0][i].second)
54                         vt[1].pop_back();
55                     vt[1].push_back(vt[0][i]);
56                 }
57                 d=inf;
58                 if(vt[1].size()==1)
59                     if(vt[1][0].first<vt[1][0].second)
60                     {
61                         ta=0;
62                         d=(vt[1][0].first<<1);
63                     }
64                 else
65                 {
66                     ta=e[i][j];
67                     d=(vt[1][0].second<<1);
68                 }
69                 else
70                     for(i=1;i<vt[1].size();++i)
71                         if(d>e[i][j]+vt[1][i-1].first+vt[1][i].second)
72                         {
73                             ta=(e[i][j]+vt[1][i].second-vt[1][i-1].first)/(double)2.0f;
74                             d=e[i][j]+vt[1][i-1].first+vt[1][i].second;
75                         }
76                 if(d<ans)
77                 {
78                     ans=d;
79                     a=i;
80                     b=j;
81                     dp[i]=ta;
82                     dp[j]=e[i][j]-ta;
83                 }
84             }
85     printf("%d\n",ans);
86     for(i=1;i<=n;++i)
87         if(i!=a && i!=b)
88             dp[i]=1e20;
89     q.insert(pdi(dp[a],a));
90     if(a!=b)
91         q.insert(pdi(dp[b],b));
92     if(a!=b)
93         pre[b]=a;
94     while(!q.empty())
95     {
96         k=q.begin()->second;
97         q.erase(q.begin());
98         if(done[k])
99             continue;
100         done[k]=true;
101         for(i=1;i<=n;++i)
102             if(e[k][i]!=inf && dp[k]+e[k][i]<dp[i])
103             {
104                 dp[i]=dp[k]+e[k][i];
105                 q.insert(pdi(dp[i],i));
106                 pre[i]=k;
107             }
108     }
109     vt[0].resize(0);
110     for(i=1;i<=n;++i)
111         if(pre[i])
112             if(i<pre[i])
113                 printf("%d %d\n",i,pre[i]);
114     else

```

```

115         printf("%d %d\n",pre[i],i);
116     return 0;
117 }

```

5.19 Kuhn-Munkres algorithm

```

1  bool match(int u)//
2  {
3      vx[u]=true;
4      for(int i=1;i<=n;++i)
5          if(lx[u]+ly[i]==g[u][i]&&!vy[i])
6          {
7              vy[i]=true;
8              if(!d[i]||match(d[i]))
9              {
10                 d[i]=u;
11                 return true;
12             }
13         }
14     return false;
15 }
16 inline void update()//
17 {
18     int i,j;
19     int a=1<<30;
20     for(i=1;i<=n;++i)if(vx[i])
21         for(j=1;j<=n;++j)if(!vy[j])
22             a=min(a,lx[i]+ly[j]-g[i][j]);
23     for(i=1;i<=n;++i)
24     {
25         if(vx[i])lx[i]-=a;
26         if(vy[i])ly[i]+=a;
27     }
28 }
29 void km()
30 {
31     int i,j;
32     for(i=1;i<=n;++i)
33     {
34         lx[i]=ly[i]=d[i]=0;
35         for(j=1;j<=n;++j)
36             lx[i]=max(lx[i],g[i][j]);
37     }
38     for(i=1;i<=n;++i)
39     {
40         while(true)
41         {
42             memset(vx,0,sizeof(vx));
43             memset(vy,0,sizeof(vy));
44             if(match(i))
45                 break;
46             update();
47         }
48     }
49     int ans=0;
50     for(i=1;i<=n;++i)
51         if(d[i]!=0)
52             ans+=g[d[i]][i];
53     printf("%d\n",ans);
54 }
55 int main()
56 {
57     while(scanf("%d\n",&n)!=EOF)
58     {
59         for(int i=1;i<=n;++i)gets(s[i]);
60         memset(g,0,sizeof(g));
61         for(int i=1;i<=n;++i)
62             for(int j=1;j<=n;++j)
63                 if(i!=j) g[i][j]=cal(s[i],s[j]);
64         km();
65     }
66     return 0;
67 }
68
69 //bupt
70 //km n^3
71 int dfs(int u)//
72 {
73     int v;
74     sx[u]=1;
75     for(v=1;v<=n;v++)
76         if(!sy[v] && lx[u]+ly[v]==map[u][v])
77         {
78             sy[v]=1;
79             if(match[v]==-1||dfs(match[v]))
80             {
81                 match[v]=u;
82                 return 1;
83             }
84         }
85     return 0;
86 }
87
88 int bestmatch(void)//km
89 {
90     int i,j,u;
91     for(i=1;i<=n;i++)//
92     {
93         lx[i]=-1;
94         ly[i]=0;
95         for(j=1;j<=n;j++)
96             if(lx[i]<map[i][j])
97                 lx[i]=map[i][j];
98     }
99     memset(match,-1,sizeof(match));
100     for(u=1;u<=n;u++)
101     {
102         while(true)
103         {
104             memset(sx,0,sizeof(sx));
105             memset(sy,0,sizeof(sy));
106             if(dfs(u))

```

```

109     break;
110     int dx=Inf; //--
111     for (i=1; i<=n; i++)
112     {
113         if (sx[i])
114             for (j=1; j<=n; j++)
115                 if (!sy[j] && dx>lx[i]+ly[j]-map[i][j])
116                     dx=lx[i]+ly[j]-map[i][j];
117     }
118     for (i=1; i<=n; i++)
119     {
120         if (sx[i])
121             lx[i]-=dx;
122         if (sy[i])
123             ly[i]+=dx;
124     }
125 }
126 }
127 int sum=0;
128 for (i=1; i<=n; i++)
129     sum+=map[match[i]][i];
130 return sum;
131 }

```

5.20 LCA - DA

```

1  int edge[MAXX],nxt[MAXX<<1],to[MAXX<<1],cnt;
2  int pre[MAXX][N],dg[MAXX];
3
4  inline void add(int j,int k)
5  {
6      nxt[++cnt]=edge[j];
7      edge[j]=cnt;
8      to[cnt]=k;
9  }
10
11 void rr(int now,int fa)
12 {
13     dg[now]=dg[fa]+1;
14     for(int i=edge[now];i;i=nxt[i])
15         if(to[i]!=fa)
16         {
17             static int j;
18             j=1;
19             for(pre[to[i]][0]=now;j<N;++j)
20                 pre[to[i]][j]=pre[pre[to[i]][j-1]][j-1];
21             rr(to[i],now);
22         }
23 }
24
25 inline int lca(int a,int b)
26 {
27     static int i,j;
28     j=0;
29     if(dg[a]<dg[b])
30         std::swap(a,b);
31     for(i=dg[a]-dg[b];i;i>=1,++)
32         if(i&1)
33             a=pre[a][j];
34     if(a==b)
35         return a;
36     for(i=N-1;i>=0;--i)
37         if(pre[a][i]!=pre[b][i])
38         {
39             a=pre[a][i];
40             b=pre[b][i];
41         }
42     return pre[a][0];
43
44 // looks like above is a wrong version
45
46 static int i,log;
47 for(log=0;(1<<(log+1))<=dg[a];++log);
48 for(i=log;i>=0;--i)
49     if(dg[a]-(1<<i)>=dg[b])
50         a=pre[a][i];
51 if(a==b)
52     return a;
53 for(i=log;i>=0;--i)
54     if(pre[a][i]!=-1 && pre[a][i]!=pre[b][i])
55         a=pre[a][i],b=pre[b][i];
56 return pre[a][0];
57 }

```

5.21 LCA - tarjan - minmax

```

1  #include<cstdio>
2  #include<list>
3  #include<algorithm>
4  #include<cstring>
5
6  #define MAXX 10011
7  #define inf 0x5fffffff
8
9  short T,t;
10 int set[MAXX],min[MAXX],max[MAXX],ans[2][MAXX];
11 bool done[MAXX];
12 std::list<std::pair<int,int>> edge[MAXX];
13 std::list<std::pair<int,int>> q[MAXX];
14 int n,i,j,k,l,m;
15
16 struct node
17 {
18     int a,b,id;
19     node() {}
20     node(const int &aa,const int &bb,const int &idd): a(aa),b(bb),
21         id(idd) {}
22 };

```

```

22
23 std::list<node>to[MAXX];
24
25 int find(const int &a)
26 {
27     if(set[a]==a)
28         return a;
29     int b=set[a];
30     set[a]=find(set[a]);
31     max[a]=std::max(max[a],max[b]);
32     min[a]=std::min(min[a],min[b]);
33     return set[a];
34 }
35
36 void tarjan(const int &now)
37 {
38     done[now]=true;
39     for(std::list<std::pair<int,int>> >::const_iterator it(q[now].
40         begin());it!=q[now].end();++it)
41         if(done[it->first])
42             if(it->second>0)
43                 to[find(it->first)].push_back(node(now,it->first,it->
44                     second));
45         else
46             to[find(it->first)].push_back(node(it->first,now,-it->
47                 second));
48     for(std::list<std::pair<int,int>> >::const_iterator it(edge[now]
49         ).begin();it!=edge[now].end();++it)
50         if(!done[it->first])
51         {
52             tarjan(it->first);
53             set[it->first]=now;
54             min[it->first]=it->second;
55             max[it->first]=it->second;
56         }
57     for(std::list<node>::const_iterator it(to[now].begin());it!=to[
58         now].end();++it)
59     {
60         find(it->a);
61         find(it->b);
62         ans[0][it->id]=std::min(min[it->b],min[it->a]);
63         ans[1][it->id]=std::max(max[it->a],max[it->b]);
64     }
65 }
66
67 int main()
68 {
69     scanf("%hd",&T);
70     for(t=1;t<=T;++t)
71     {
72         scanf("%d",&n);
73         for(i=1;i<=n;++i)
74         {
75             edge[i].clear();
76             q[i].clear();
77             to[i].clear();
78             done[i]=false;
79             set[i]=i;
80             min[i]=inf;
81             max[i]=0;
82         }
83         for(i=1;i<=n;++i)
84         {
85             scanf("%d%d%d",&j,&k,&l);
86             edge[j].push_back(std::make_pair(k,l));
87             edge[k].push_back(std::make_pair(j,l));
88         }
89         scanf("%d",&m);
90         for(i=0;i<=m;++i)
91         {
92             scanf("%d %d",&j,&k);
93             q[j].push_back(std::make_pair(k,i));
94             q[k].push_back(std::make_pair(j,-i));
95         }
96         tarjan(1);
97         printf("Case %hd:\n",t);
98         for(i=0;i<=m;++i)
99             printf("%d %d\n",ans[0][i],ans[1][i]);
100     }
101     return 0;
102 }

```

5.22 Minimum Ratio Spanning Tree

```

1  #include<cstdio>
2  #include<cstring>
3  #include<cmath>
4
5  #define MAXX 1111
6
7  struct
8  {
9      int x,y;
10     double z;
11 } node[MAXX];
12
13 struct
14 {
15     double l,c;
16 } map[MAXX][MAXX];
17
18 int n,l,f[MAXX],pge[MAXX];
19 double dis[MAXX];
20
21 double mst(double x)
22 {
23     int i,j,tmp;
24     double min,s=0,t=0;
25     memset(f,0,sizeof(f));
26     f[1]=1;
27     for (i=2; i<=n; i++)
28     {

```

```

29     dis[i]=map[l][i].c-map[l][i].l*x;
30     pre[i]=l;
31 }
32 for (i=1; i<n; i++)
33 {
34     min=le10;
35     for (j=1; j<=n; j++)
36         if (!f[j] && min>dis[j])
37         {
38             min=dis[j];
39             tmp=j;
40         }
41     f[tmp]=1;
42     t+=map[pre[tmp]][tmp].l;
43     s+=map[pre[tmp]][tmp].c;
44     for (j=1; j<=n; j++)
45         if (!f[j] && map[tmp][j].c-map[tmp][j].l*x<dis[j])
46         {
47             dis[j]=map[tmp][j].c-map[tmp][j].l*x;
48             pre[j]=tmp;
49         }
50     }
51     return s/t;
52 }
53
54 int main()
55 {
56     int i,j;
57     double a,b;
58     while (scanf("%d",&n),n);
59     {
60         for (i=1; i<=n; i++)
61             scanf("%d%d%lf",&node[i].x,&node[i].y,&node[i].z);
62         for (i=1; i<=n; i++)
63             for (j=i+1; j<=n; j++)
64             {
65                 map[j][i].l=map[i][j].l=sqrt(1.0*(node[i].x-node[j].x)
66                     *(node[i].x-node[j].x)+(node[i].y-node[j].y)*(
67                     node[i].y-node[j].y));
68                 map[j][i].c=map[i][j].c=fabs(node[i].z-node[j].z);
69             }
70         a=0,b=mst(a);
71         while (fabs(b-a)>1e-8)
72         {
73             a=b;
74             b=mst(a);
75         }
76         printf("%.3lf\n",b);
77     }
78     return 0;

```

5.23 Minimum-cost flow problem

```

1 // like Edmonds-Karp Algorithm
2 #include<cstdio>
3 #include<cstring>
4 #include<algorithm>
5 #include<queue>
6
7 #define MAXX 5011
8 #define MAXE (MAXX*10*2)
9 #define inf 0x3f3f3f3f
10
11 int edge[MAXX],nxt[MAXE],to[MAXE],cap[MAXE],cst[MAXE],cnt;
12 #define v to[i]
13 inline void adde(int a,int b,int c,int d)
14 {
15     nxt[++cnt]=edge[a];
16     edge[a]=cnt;
17     to[cnt]=b;
18     cap[cnt]=c;
19     cst[cnt]=d;
20 }
21 inline void add(int a,int b,int c,int d)
22 { adde(a,b,c,d); adde(b,a,0,-d); }
23
24 int dist[MAXX],pre[MAXX];
25 int source,sink;
26 std::queue<int>q;
27 bool in[MAXX];
28
29 inline bool go()
30 {
31     static int now,i;
32     memset(dist,0x3f,sizeof dist);
33     dist[source]=0;
34     pre[source]=-1;
35     q.push(source);
36     in[source]=true;
37     while(!q.empty())
38     {
39         in[now=q.front()]=false;
40         q.pop();
41         for(i=edge[now];i!=-1;i=nxt[i])
42             if(cap[i] && dist[v]>dist[now]+cst[i])
43             {
44                 dist[v]=dist[now]+cst[i];
45                 pre[v]=i;
46                 if(!in[v])
47                 {
48                     q.push(v);
49                     in[v]=true;
50                 }
51             }
52     }
53     return dist[sink]!=inf;
54 }
55
56 inline int mcmf(int &flow)
57 {
58     static int ans,i;
59     flow=ans=0;

```

```

60     while(go())
61     {
62         static int min;
63         min=inf;
64         for(i=pre[sink];i!=-1;i=pre[to[i^1]])
65             min=std::min(min,cap[i]);
66         flow+=min;
67         ans+=min*dist[sink];
68         for(i=pre[sink];i!=-1;i=pre[to[i^1]])
69         {
70             cap[i]-=min;
71             cap[i^1]+=min;
72         }
73     }
74     return ans;
75 }
76
77 // TQ's version
78 struct mcmf
79 {
80     struct Edge
81     {
82         int from,to,cap,flow,cost;
83     };
84     int n,m,s,t;
85     std::vector<Edge>edges;
86     std::vector<int>G[maxn];
87     int inq[maxn],d[maxn],p[maxn],a[maxn];
88
89     void init(int n)
90     {
91         this->n=n;
92         for(int i=0;i<n;++i)
93             G[i].clear();
94         edges.clear();
95     }
96
97     void addedge(int from,int to,int cap,int cost)
98     {
99         Edge x={from,to,cap,0,cost};
100         edges.push_back(x);
101         Edge y={to,from,0,0,-cost};
102         edges.push_back(y);
103         m=edges.size();
104         G[from].push_back(m-2);
105         G[to].push_back(m-1);
106     }
107
108     int mincost(int s,int t)
109     {
110         int flow=0,cost=0;
111         while(BellmanFord(s,t,flow,cost));
112         if(flow!=(n-1)/2) return -1;
113         return cost;
114     }
115 private:
116     bool BellmanFord(int s,int t,int& flow,int& cost)
117     {
118         for(int i=0;i<n;++i)
119             d[i]=INF;
120         memset(inq,0,sizeof(inq));
121         d[s]=0; inq[s]=1; p[s]=0; a[s]=INF;
122         std::queue<int>q;
123         Q.push(s);
124         while(!Q.empty())
125         {
126             int u=Q.front();
127             Q.pop();
128             inq[u]=0;
129             for(int i=0;i<G[u].size();++i)
130             {
131                 Edge& e=edges[G[u][i]];
132                 if(e.cap>e.flow && d[e.to]>d[u]+e.cost)
133                 {
134                     d[e.to]=d[u]+e.cost;
135                     p[e.to]=G[u][i];
136                     a[e.to]=min(a[u],e.cap-e.flow);
137                     if(!inq[e.to])
138                     {
139                         Q.push(e.to);
140                         inq[e.to]=1;
141                     }
142                 }
143             }
144         }
145         if(d[t]==INF)
146             return false;
147         flow+=a[t];
148         cost+=d[t]*a[t];
149         int u=t;
150         while(u!=s)
151         {
152             edges[p[u]].flow+=a[t];
153             edges[p[u]^1].flow-=a[t];
154             u=edges[p[u]].from;
155         }
156         return true;
157     }
158 }G;

```

5.24 Stable Marriage

```

1 //
2
3 while(!g.empty()) //
4 {
5     if(dfn[edge[g.front()]].front()<=-1)
6         dfn[edge[g.front()]].front()=g.front(); //
7     else
8     {
9         for(it=edge[edge[g.front()]].front().begin();it!=edge[edge[g
10             .front()]].front().end();++it)
11             if(*it==dfn[edge[g.front()]].front() || *it==g.front())
12                 //

```

```

11         break;
12         if(*it==g.front()) //
13         {
14             g.push_back(dfn[edge[g.front()].front()]);
15             dfn[edge[g.front()].front()]=g.front();
16         }
17         else
18             g.push_back(g.front()); //
19     }
20     edge[g.front()].pop_front(); //
21     g.pop_front();
22 }

```

5.25 Stoer-Wagner Algorithm

```

1 #include <iostream>
2 using namespace std;
3 const int maxn=510;
4 int map[maxn][maxn];
5 int n;
6 void contract(int x,int y)//
7 {
8     int i,j;
9     for (i=0; i<n; i++)
10         if (i!=x) map[x][i]+=map[y][i],map[i][x]+=map[i][y];
11     for (i=y+1; i<n; i++) for (j=0; j<n; j++)
12     {
13         map[i-1][j]=map[i][j];
14         map[j][i-1]=map[j][i];
15     }
16     n--;
17 }
18 int w[maxn],c[maxn];
19 int sx,tx;
20 int mincut()
21 //
22 {
23     int i,j,k,t;
24     memset(c,0,sizeof(c));
25     c[0]=1;
26     for (i=0; i<n; i++) w[i]=map[0][i];
27     for (i=1; i<n; i++)
28     {
29         t=k=-1;
30         for (j=0; j<n; j++) if (c[j]==0&&w[j]>k)
31             k=w[t=j];
32         c[sx=t]=1;
33         for (j=0; j<n; j++) w[j]+=map[t][j];
34     }
35     for (i=0; i<n; i++) if (c[i]==0) return w[tx=i];
36 }
37 int main()
38 {
39     int i,j,k,m;
40     while (scanf("%d%d",&n,&m)!=EOF)
41     {
42         memset(map,0,sizeof(map));
43         while (m--)
44         {
45             scanf("%d%d%d",&i,&j,&k);
46             map[i][j]+=k;
47             map[j][i]+=k;
48         }
49         int mint=999999999;
50         while (n>1)
51         {
52             k=mincut();
53             if (k<mint) mint=k;
54             contract(sx,tx);
55         }
56         printf("%d\n",mint);
57     }
58     return 0;
59 }

```

5.26 Strongly Connected Component

```

1 //
2 void dfs(const short &now)
3 {
4     dfn[now]=low[now]=cnt++;
5     st.push(now);
6     for(std::list<short>::const_iterator it(edge[now].begin()); it!=
7         edge[now].end(); ++it)
8         if(dfn[*it]==-1)
9         {
10             dfs(*it);
11             low[now]=std::min(low[now], low[*it]);
12         }
13         else
14             if(sc[*it]==-1)
15                 low[now]=std::min(low[now], dfn[*it]);
16     if(dfn[now]==low[now])
17     {
18         while(sc[now]==-1)
19         {
20             sc[st.top()]=p;
21             st.pop();
22         }
23         ++p;
24     }
25 }

```

5.27 ZKW's Minimum-cost flow

```

1 #include<cstdio>
2 #include<algorithm>
3 #include<cstring>
4 #include<vector>
5 #include<deque>
6
7 #define MAXX 111
8 #define MAXN 211
9 #define MAXE (MAXN*MAXN*3)
10 #define inf 0x3f3f3f3f
11
12 char buf[MAXX];
13
14 int edge[MAXN],nxt[MAXE],to[MAXE],cap[MAXE],cst[MAXE],cnt;
15
16 inline void adde(int a,int b,int c,int k)
17 {
18     nxt[cnt]=edge[a];
19     edge[a]=cnt;
20     to[cnt]=b;
21     cap[cnt]=c;
22     cst[cnt]=k;
23     ++cnt;
24 }
25
26 inline void add(int a,int b,int c,int k)
27 {
28     adde(a,b,c,k);
29     adde(b,a,0,-k);
30 }
31
32 int n,mf,cost,pil;
33 int source,sink;
34 bool done[MAXN];
35
36 int aug(int now,int maxcap)
37 {
38     if(now==sink)
39     {
40         mf+=maxcap;
41         cost+=maxcap*pil;
42         return maxcap;
43     }
44     done[now]=true;
45     int l=maxcap;
46     for(int i=edge[now];i!=-1;i=nxt[i])
47         if(cap[i]&&!cst[i]&&!done[to[i]])
48         {
49             int d(aug(to[i],std::min(l,cap[i])));
50             cap[i]-=d;
51             cap[i^1]+=d;
52             l-=d;
53             if(!l)
54                 return maxcap;
55         }
56     return maxcap-l;
57 }
58
59 inline bool label()
60 {
61     static int d,i,j;
62     d=inf;
63     for(i=1;i<n;++i)
64         if(done[i])
65             for(j=edge[i];j!=-1;j=nxt[j])
66                 if(cap[j]&&!done[to[j]]&&cst[j]<d)
67                     d=cst[j];
68     if(d==inf)
69         return false;
70     for(i=1;i<n;++i)
71         if(done[i])
72             for(j=edge[i];j!=-1;j=nxt[j])
73             {
74                 cst[j]-=d;
75                 cst[j^1]+=d;
76             }
77     pil+=d;
78     return true;
79 }
80 /* primal-dual approach
81 static int d[MAXN],i,j;
82 static std::deque<int>g;
83 memset(d,0x3f,sizeof d);
84 d[sink]=0;
85 q.push_back(sink);
86 while(!q.empty())
87 {
88     static int dt,now;
89     now=q.front();
90     q.pop_front();
91     for(i=edge[now];i!=-1;i=nxt[i])
92         if((d[to[i]]>dt=(d[now]-cst[i])<d[to[i]])
93             if((d[to[i]]>dt)<d[q.empty()?0:q.front()])
94                 q.push_front(to[i]);
95         else
96             q.push_back(to[i]);
97 }
98 for(i=1;i<n;++i)
99     for(j=edge[i];j!=-1;j=nxt[j])
100         cst[j]+d[to[j]]-d[i];
101 pil+=d[source];
102 return d[source]!=inf;
103 */
104
105 int m,i,j,k;
106 typedef std::pair<int,int> pii;
107 std::vector<pii>M(MAXN),H(MAXN);
108
109 int main()
110 {
111     while(scanf("%d %d",&n,&m),(n|m))
112     {
113         M.resize(0);
114         H.resize(0);
115         for(i=0;i<n;++i)
116             {

```

```

117     scanf ("%s", buf);
118     for (j=0; j<m; ++j)
119         if (buf[j] == 'm')
120             M.push_back(pii(i, j));
121     else
122         if (buf[j] == 'H')
123             H.push_back(pii(i, j));
124 }
125 n=M.size()+H.size();
126 source=++n;
127 sink=++n;
128 memset(edge, -1, sizeof edge);
129 cnt=0;
130 for (i=0; i<M.size(); ++i)
131     for (j=0; j<H.size(); ++j)
132         add(i+1, j+1+M.size(), 1, abs(M[i].first-H[j].first)+abs(
133             M[i].second-H[j].second));
134 for (i=0; i<M.size(); ++i)
135     add(source, i+1, 1, 0);
136 for (i=0; i<H.size(); ++i)
137     add(i+1+M.size(), sink, 1, 0);
138 mf=cost=pl=0;
139 do
140     do
141         memset(done, 0, sizeof done);
142         while (aug(source, inf));
143     while (label());
144     /* primal-dual approach
145     while (label())
146         do
147             memset(done, 0, sizeof done);
148             while (aug(source, inf));
149     */
150     printf ("%d\n", cost);
151 }
152 return 0;
153 }

```

5.28 ZKW's SAP

```

1 // wrong answer at poj 1149
2 // wrong answer at uestc 1195
3 #include<cstdio>
4 #include<algorithm>
5 #include<cstring>
6
7 #define MAXX 5111
8 #define MAXM (30111*4)
9 #define inf 0x3f3f3f3f3f3f3f3f
10
11 int edge[MAXX], to[MAXM], nxt[MAXM], cnt;
12 int w[MAXX];
13 long long cap[MAXM];
14
15 int n;
16 int h[MAXX], vh[MAXX];
17
18 inline void add(int a, int b, long long c)
19 {
20     nxt[cnt]=edge[a];
21     edge[a]=cnt;
22     to[cnt]=b;
23     cap[cnt]=c;
24     ++cnt;
25 }
26
27 int source, sink;
28
29 long long aug(int now, long long flow)
30 {
31     if (now==sink)
32         return flow;
33     long long l(flow);
34     for (int i(edge[now]); i!=-1; i=nxt[i])
35         if (cap[i] && h[to[i]]+l==h[now])
36             {
37                 long long d(aug(to[i], std::min(l, cap[i])));
38                 cap[i]-=d;
39                 cap[i^1]+=d;
40                 l-=d;
41                 if (h[source]==n || !l)
42                     return flow-l;
43             }
44     int minh(n);
45     for (int i(edge[now]=w[now]); i!=-1; i=nxt[i])
46         if (cap[i] && h[to[i]]+1<minh)
47             minh=h[to[i]]+1;
48     if (!--vh[h[now]])
49         h[source]=n;
50     else
51         ++vh[h[now]=minh];
52     return flow-l;
53 }
54
55 int m, i, j, k;
56 long long ans;
57
58 int main()
59 {
60     scanf ("%d %d", &n, &m);
61     source=1;
62     sink=n;
63     memset(edge, -1, sizeof edge);
64     while (m--)
65     {
66         scanf ("%d %d %lld", &i, &j, &ans);
67         add(i, j, ans);
68         add(j, i, 0);
69     }
70     add(j, i, ans);
71     add(i, j, 0);
72 }
73 memcpy(w, edge, sizeof edge);
74 memset(h, 0, sizeof h);

```

```

75     memset(vh, 0, sizeof vh);
76     vh[0]=n;
77     ans=0;
78     while (h[source]<n)
79         ans+=aug(source, inf);
80     printf ("%lld\n", ans);
81     return 0;
82 }

```

6 math

6.1 cantor

```

1 const int PermSize = 12;
2 int fac[PermSize] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320,
3     362880, 3628800, 39916800};
4
5 inline int Cantor(int a[])
6 {
7     int i, j, cnt;
8     int res = 0;
9     for (i = 0; i < PermSize; ++i)
10     {
11         cnt = 0;
12         for (j = i + 1; j < PermSize; ++j)
13             if (a[i] > a[j])
14                 ++cnt;
15         res = res + cnt * fac[PermSize - i - 1];
16     }
17     return res;
18 }
19
20 bool h[13];
21
22 inline void UnCantor(int x, int res[])
23 {
24     int i, j, l, t;
25     for (i = 1; i <= 12; i++)
26         h[i] = false;
27     for (i = 1; i <= 12; i++)
28     {
29         t = x / fac[12 - i];
30         x -= t * fac[12 - i];
31         for (j = 1, l = 0; l <= t; j++)
32             if (!h[j])
33                 l++;
34         j--;
35         h[j] = true;
36         res[i - 1] = j;
37     }
38 }

```

6.2 Continued fraction

```

1 // not tested yet
2 #include<cstdio>
3 #include <iostream>
4 #include <cmath>
5 #include <cstring>
6
7 #define min(a,b) (a>b?b:a)
8
9 long long d[10000], num[10000], dnm[10000];
10 long long l, p;
11 long long ll, 12;
12
13 void rr(double num)
14 {
15     int sub = floor(num);
16     d[i++] = sub;
17     if (sub == num)
18         return;
19     if (i > 2000)
20         return;
21     rr(1 / (num - sub));
22 }
23
24 long long numerator(int n)
25 {
26     if (num[n] != 0)
27         return num[n];
28     long long i = -1;
29     if (n == 0)
30         i = d[0];
31     else
32         if (n == 1)
33             i = d[0] * d[1] + 1;
34         else
35             i = numerator(n - 1) * d[n] + numerator(n - 2);
36     num[n] = i;
37     return i;
38 }
39
40 if (i > p)
41 {
42     ll = n - 1;
43     i = 0;
44     num[n] = 0;
45 }
46 else
47     num[n] = i;
48 return i;
49 }
50
51 long long denominator(int n)
52 {
53     if (dnm[n] != 0)

```

```

53     return dnm[n];
54 long long i = -1;
55 if (n == 0)
56     i = 1;
57 else
58     if (n == 1)
59         i = d[1];
60     else
61         i = denominator(n - 1) * d[n] + denominator(n - 2);
62 dnm[n] = i;
63 return i;
64
65 if (i > p)
66 {
67     l2 = n - 1;
68     i = 0;
69     dnm[n] = 0;
70 }
71 else
72     dnm[n] = i;
73 return i;
74 }
75
76 int main()
77 {
78     int n;
79     while (scanf("%d", &n) != EOF)
80     {
81         if (n == 0)
82             return 0;
83
84         memset(num, 0, sizeof num);
85         memset(dnm, 0, sizeof dnm);
86
87         i = 0;
88         rr(sqrt((double)n));
89
90         numerator(25);
91         denominator(25);
92
93         int f;
94         for (f = 0; f < 25; ++f)
95             printf("%lld/%lld\n", num[f], dnm[f]);
96     }
97     return 0;
98 }

```

6.3 Discrete logarithms - BSGS

```

1 //The running time of BSGS and the space complexity is O(\sqrt{n})
2 //Pollard's rho algorithm for logarithms' running time is
   approximately O(\sqrt{p}) where p is n's largest prime
   factor.
3 #include<cstdio>
4 #include<cmath>
5 #include<cstring>
6
7 struct Hash // std::map is bad. clear()
8 {
9     static const int mod=100003; // prime is good
10    static const int MAXX=47111; // bigger than sqrt(c)
11    int hd[mod],nxt[MAXX],cnt;
12    long long v[MAXX],k[MAXX]; // a*k v (mod c)
13    inline void init ()
14    {
15        memset(hd,0,sizeof hd);
16        cnt=0;
17    }
18    inline long long find(long long v)
19    {
20        static int now;
21        for(now=hd[v%mod];now;nxt[now])
22            if(this->v[now]==v)
23                return k[now];
24        return -1ll;
25    }
26    inline void insert(long long k,long long v)
27    {
28        if(find(v)!=-1ll)
29            return;
30        nxt[++cnt]=hd[v%mod];
31        hd[v%mod]=cnt;
32        this->v[cnt]=v;
33        this->k[cnt]=k;
34    }
35 }hash;
36
37 long long god(long long a,long long b)
38 {
39     return b?god(b,a%b):a;
40 }
41
42 long long exgcd(long long a,long long b,long long &x,long long &y)
43 {
44     if(b)
45     {
46         long long re(exgcd(b,a%b,x,y)),tmp(x);
47         x=y;
48         y=tmp-(a/b)*y;
49         return re;
50     }
51     x=1ll;
52     y=0ll;
53     return a;
54 }
55
56 inline long long bsgs(long long a,long long b,long long c) // a^x
   b (mod c)
57 {
58     static long long x,y,d,g,m,am,k;
59     static int i,cnt;
60     a%=c;
61     b%=c;
62     x=1ll*c; // if c==1....

```

```

63     for(i=0;i<100;++i)
64     {
65         if(x==b)
66             return i;
67         x=(x*a)%c;
68     }
69     d=1ll*c;
70     cnt=0;
71     while((g=gcd(a,c))!=1ll)
72     {
73         if(b%g)
74             return -1ll;
75         ++cnt;
76         c/=g;
77         b/=g;
78         d=a/g*d%c;
79     }
80     hash.init();
81     m=sqrt((double)c); // maybe need a ceil
82     am=1ll*c;
83     hash.insert(0,am);
84     for(i=1;i<=m;++i)
85     {
86         am=am*a%c;
87         hash.insert(i,am);
88     }
89     for(i=0;i<=m;++i)
90     {
91         g=exgcd(d,c,x,y);
92         x=(x*b/g%c+c)%c;
93         k=hash.find(x);
94         if(k!=-1ll)
95             return i*m+k+cnt;
96         d=d*a%c;
97     }
98     return -1ll;
99 }
100
101 long long k,p,n;
102
103 int main()
104 {
105     while(scanf("%lld %lld %lld",&k,&p,&n)!=EOF)
106     {
107         if(n>p || (k=bsgs(k,n,p))!=-1ll)
108             puts("Orz,I cant find D!");
109         else
110             printf("%lld\n",k);
111     }
112     return 0;
113 }

```

6.4 Divisor function

```

1 sum of positive divisors function
2 (n)=(pow(p[0],a[0]+1)-1)/(p[0]-1)* (pow(p[1],a[1]+1)-1)/(p[1]-1)*
   ... (pow(p[n-1],a[n-1]+1)-1);

```

6.5 Extended Euclidean Algorithm

```

1 //ax+by=gcd(a,b)
2 long long ex_gcd(long long a,long long b,long long &x,long long &y)
   )
3 {
4     if (b)
5     {
6         long long ret = ex_gcd(b,a%b,x,y),tmp = x;
7         x = y;
8         y = tmp-(a/b)*y;
9         return ret;
10    }
11    else
12    {
13        x = 1;
14        y = 0;
15        return a;
16    }
17 }

```

6.6 Gaussian elimination

```

1 #define N
2
3 inline int ge(int a[N][N],int n) //
4 {
5     static int i,j,k,l;
6     for(j=i=0;j<n;++j) //i,j
7     {
8         for(k=i;k<n;++k)
9             if(a[k][j])
10                break;
11        if(k==n)
12            continue;
13        for(l=0;l<n;++l)
14            std::swap(a[i][l],a[k][l]);
15        for(l=0;l<n;++l)
16            if(l!=i && a[l][j])
17                for(k=0;k<n;++k)
18                    a[l][k]^=a[i][k];
19        ++i;
20    }
21    for(j=i;j<n;++j)
22        if(a[j][n])

```

```

23     return -1; //
24     return i;
25 }
26 /*
27 */
28
29 void dfs(int v)
30 {
31     if (v==n)
32     {
33         static int x[MAXX], ta[MAXX] [MAXX];
34         static int tmp;
35         memcpy(x, ans, sizeof(x));
36         memcpy(ta, a, sizeof(ta));
37         for (i=1; i>=0; --i)
38         {
39             for (j=i+1; j<n; ++j)
40                 ta[i][j] = (x[j] && ta[i][j]); //
41             x[i] = ta[i][i];
42         }
43         for (tmp=i=0; i<n; ++i)
44             if (x[i])
45                 ++tmp;
46         cnt = std::min(cnt, tmp);
47         return;
48     }
49     ans[v]=0;
50     dfs(v+1);
51     ans[v]=1;
52     dfs(v+1);
53 }
54
55 inline int ge(int a[N] [N], int n)
56 {
57     static int i, j, k, l;
58     for (i=j=0; j<n; ++j)
59     {
60         for (k=i; k<n; ++k)
61             if (a[k][i])
62                 break;
63         if (k<n)
64         {
65             for (l=0; l<=n; ++l)
66                 std::swap(a[i][l], a[k][l]);
67             for (k=0; k<n; ++k)
68                 if (k!=i && a[k][i])
69                     for (l=0; l<=n; ++l)
70                         a[k][l] ^= a[i][l];
71             ++i;
72         }
73         else //
74         {
75             l=n-1-j+1;
76             for (k=0; k<n; ++k)
77                 std::swap(a[k][l], a[k][i]);
78         }
79     }
80     if (i==n)
81     {
82         for (i=cnt=0; i<n; ++i)
83             if (a[i][n])
84                 ++cnt;
85         printf("%d\n", cnt);
86         continue;
87     }
88     for (j=i; j<n; ++j)
89         if (a[j][n])
90             break;
91     if (j<n)
92         puts("impossible");
93     else
94     {
95         memset(ans, 0, sizeof(ans));
96         cnt=111;
97         dfs(1);
98         printf("%d\n", cnt);
99     }
100 }
101 /*
102 */
103
104
105 inline void ge(int a[N] [N], int m, int n) // m*n
106 {
107     static int i, j, k, l, b, c;
108     for (i=j=0; i<m && j<n; ++j)
109     {
110         for (k=i; k<m; ++k)
111             if (a[k][j])
112                 break;
113         if (k==m)
114             continue;
115         for (l=0; l<=n; ++l)
116             std::swap(a[i][l], a[k][l]);
117         for (k=0; k<m; ++k)
118             if (k!=i && a[k][j])
119             {
120                 b=a[k][j];
121                 c=a[i][j];
122                 for (l=0; l<=n; ++l)
123                     a[k][l] = (a[k][l] * c - a[i][l] * b) % 7 + 7;
124             }
125         ++i;
126     }
127     for (j=i; j<m; ++j)
128         if (a[j][n])
129             break;
130     if (j<m)
131     {
132         puts("Inconsistent data.");
133         return;
134     }
135     if (i<n)
136         puts("Multiple solutions.");
137     else
138     {
139         memset(ans, 0, sizeof(ans));
140         for (i=n-1; i>=0; --i)
141         {
142             k=a[i][n];

```

```

143         for (j=i+1; j<n; ++j)
144             k = ((k - a[i][j]) * ans[j]) % 7 + 7;
145         while (k % a[i][i])
146             k += 7;
147         ans[i] = (k / a[i][i]) % 7;
148     }
149     for (i=0; i<n; ++i)
150         printf("%d%c", ans[i], i+1==n ? '\n' : ' ');
151 }
152 }

```

6.7 inverse element

```

1 inline void getInv2(int x, int mod)
2 {
3     inv[1]=1;
4     for (int i=2; i<=x; i++)
5         inv[i] = (mod - (mod/i) * inv[mod%i] % mod) % mod;
6 }
7
8 long long power(long long x, long long y, int mod)
9 {
10     long long ret=1;
11     for (long long a=x%mod; y; y>>=1, a=a*a%mod)
12         if (y&1)
13             ret=ret*a%mod;
14     return ret;
15 }
16
17 inline int getInv(int x, int mod) //mod
18 {
19     return power(x, mod-2);
20 }

```

6.8 Linear programming

```

1 #include<cstdio>
2 #include<cstring>
3 #include<cmath>
4 #include<algorithm>
5
6 #define MAXN 33
7 #define MAXM 33
8 #define eps 1e-8
9
10 double a[MAXN] [MAXM], b[MAXN], c[MAXN];
11 double x[MAXM], d[MAXN] [MAXM];
12 int ix[MAXN+MAXM];
13 double ans;
14 int n, m;
15 int i, j, k, r, s;
16 double D;
17
18 inline bool simplex()
19 {
20     r=n;
21     s=m++;
22     for (i=0; i<n+m; ++i)
23         ix[i]=i;
24     memset(d, 0, sizeof d);
25     for (i=0; i<n; ++i)
26     {
27         for (j=0; j+1<m; ++j)
28             d[i][j] = -a[i][j];
29         d[i][m-1]=1;
30         d[i][m]=b[i];
31         if (d[r][m]>d[i][m])
32             r=i;
33     }
34     for (j=0; j+1<m; ++j)
35         d[n][j]=c[j];
36         d[n+1][m-1]=-1;
37         while (true)
38         {
39             if (r<n)
40             {
41                 std::swap(ix[s], ix[r+m]);
42                 d[r][s]=1./d[r][s];
43                 for (j=0; j<m; ++j)
44                     if (j!=s)
45                         d[r][j] += -d[r][s];
46                 for (i=0; i<n+1; ++i)
47                     if (i!=r)
48                     {
49                         for (j=0; j<=m; ++j)
50                             if (j!=s)
51                                 d[i][j] += d[r][j] * d[i][s];
52                                 d[i][s] *= d[r][s];
53                     }
54             }
55             r=-1;
56             s=-1;
57             for (j=0; j<m; ++j)
58                 if ((s<0 || ix[s]>ix[j]) && (d[n+1][j]>eps || (d[n+1][j]>-eps && d[n][j]>eps)))
59                     s=j;
60             if (s<0)
61                 break;
62             for (i=0; i<n; ++i)
63                 if (d[i][s]<-eps && (r<0 || (D=(d[r][m]/d[r][s]-d[i][m]/d[i][s]))<-eps || (D<eps && ix[r+m]>ix[i+m])))
64                     r=i;
65             if (r<0)
66                 return false;
67         }
68         if (d[n+1][m]<-eps)
69             return false;
70         for (i=m; i<n+m; ++i)

```



```

71     if(ix[i]+1<m)
72         x[ix[i]]=d[i-m][m]; // answer
73     ans=d[n][m]; // maxium value
74     return true;
75 }
76
77 int main()
78 {
79     while (scanf("%d %d",&m,&n)!=EOF)
80     {
81         for(i=0;i<m;++i)
82             scanf("%lf",c+i); // max{ sum{c[i]*x[i]} }
83         for(i=0;i<n;++i)
84         {
85             for(j=0;j<m;++j)
86                 scanf("%lf",a[i+j]); // sum{ a[i]*x[i] } <= b
87             scanf("%lf",b+i);
88             b[i]=n;
89         }
90         simplex();
91         printf("Nasa can spend %.0lf taka.\n",ceil(ans));
92     }
93     return 0;
94 }

```

6.9 Lucas' theorem(2)

```

1  #include<cstdio>
2  #include<cstring>
3  #include<iostream>
4
5  int mod;
6  long long num[100000];
7  int ni[100],mi[100];
8  int len;
9
10 void init(int p)
11 {
12     mod=p;
13     num[0]=1;
14     for (int i=1; i<p; i++)
15         num[i]=i*num[i-1]%p;
16 }
17
18 void get (int n,int ni[],int p)
19 {
20     for (int i = 0; i < 100; i++)
21         ni[i] = 0;
22     int tlen = 0;
23     while (n != 0)
24     {
25         ni[tlen++] = n%p;
26         n /= p;
27     }
28     len = tlen;
29 }
30
31 long long power(long long x,long long y)
32 {
33     long long ret=1;
34     for (long long a=x%mod; y>=1,a=a*a%mod)
35         if (y&1)
36             ret=ret*a%mod;
37     return ret;
38 }
39
40 long long getInv(long long x)//'mod'
41 {
42     return power(x,mod-2);
43 }
44
45 long long calc(int n,int m,int p)//C(n,m)%p
46 {
47     init(p);
48     long long ans=1;
49     for (; n && m && ans; n/=p,m/=p)
50     {
51         if (n%p>=m%p)
52             ans = ans*num[n%p]%p *getInv(num[m%p]%p)%p *getInv(num[n%
53                 p-m%p])%p;
54         else
55             ans=0;
56     }
57     return ans;
58 }
59
60 int main()
61 {
62     int t;
63     scanf("%d",&t);
64     while (t--)
65     {
66         int n,m,p;
67         scanf("%d%d%d",&n,&m,&p);
68         printf("%lld\n",calc(n+m,m,p));
69     }
70     return 0;
71 }

```

6.10 Lucas' theorem

```

1  #include <stdio>
2  /*
3   Lucas C(n,m)%p
4   */
5  void gcd(int n,int k,int &x,int &y)
6  {
7      if (k)
8      {

```

```

9          gcd(k,n%k,x,y);
10         int t=x;
11         x=y;
12         y=t-(n/k)*y;
13         return;
14     }
15     x=1;
16     y=0;
17 }
18
19 int CmodP(int n,int k,int p)
20 {
21     if(k>n)
22         return 0;
23     int a,b,flag=0,x,y;
24     a=b=1;
25     for(int i=1;i<=k;i++)
26     {
27         x=n-i+1;
28         y=i;
29         while(x%p==0)
30         {
31             x/=p;
32             ++flag;
33         }
34         while(y%p==0)
35         {
36             y/=p;
37             --flag;
38         }
39         x%=p;
40         y%=p;
41
42         a*=x;
43         b*=y;
44
45         b%=p;
46         a%=p;
47     }
48     if(flag)
49         return 0;
50     gcd(b,p,x,y);
51     if(x<0)
52         x+=p;
53     a*=x;
54     a%=p;
55     return a;
56 }
57
58 //Lucas C(n,m) % p ,p
59 long long Lucas(long long n, long long m, long long p)
60 {
61     long long ans=1;
62     while(m && n && ans)
63     {
64         ans*=CmodP(n%p,m%p,p);
65         ans=ans%p;
66         n=n/p;
67         m=m/p;
68     }
69     return ans;
70 }
71
72 int main()
73 {
74     long long n,k,p,ans;
75     int cas=0;
76     while (scanf("%I64d%I64d%I64d",&n,&k,&p)!=EOF)
77     {
78         if(k>n-k)
79             k=n-k;
80         ans=Lucas(n+1,k,p)+n-k;
81         printf("Case #%d: %I64d\n",++cas,ans%p);
82     }
83     return 0;
84 }

```

6.11 Matrix

```

1  struct Matrix
2  {
3      const int N(52);
4      int a[N][N];
5      inline Matrix operator*(const Matrix &b)const
6      {
7          static Matrixres;
8          static int i,j,k;
9          for(i=0;i<N;++i)
10             for(j=0;j<N;++j)
11             {
12                 res.a[i][j]=0;
13                 for(k=0;k<N;++k)
14                     res.a[i][j]+=a[i][k]*b.a[k][j];
15             }
16             return res;
17     }
18     inline Matrix operator^(int y)const
19     {
20         static Matrix res,x;
21         static int i,j;
22         for(i=0;i<N;++i)
23         {
24             for(j=0;j<N;++j)
25             {
26                 res.a[i][j]=0;
27                 x.a[i][j]=a[i][j];
28             }
29             res.a[i][i]=1;
30         }
31         for(;y;y>=1,x=x*x)
32             if(y&1)
33                 res=res*x;
34         return res;
35     }
36 };

```

```

37
38 Fibonacci Matrix
39 [1 1]
40 [1 0]

```

6.12 Miller-Rabin Algorithm

```

1 inline unsigned long long multi_mod(const unsigned long long &a,
  unsigned long long b,const unsigned long long &n)
2 {
3     unsigned long long exp(a%n),tmp(0);
4     while(b)
5     {
6         if(b&1)
7         {
8             tmp+=exp;
9             if(tmp>n)
10                 tmp-=n;
11         }
12         exp<<=1;
13         if(exp>n)
14             exp-=n;
15         b>>=1;
16     }
17     return tmp;
18 }
19
20 inline unsigned long long exp_mod(unsigned long long a,unsigned
  long long b,const unsigned long long &c)
21 {
22     unsigned long long tmp(1);
23     while(b)
24     {
25         if(b&1)
26             tmp=multi_mod(tmp,a,c);
27         a=multi_mod(a,a,c);
28         b>>=1;
29     }
30     return tmp;
31 }
32
33 inline bool miller_rabbin(const unsigned long long &n,short T)
34 {
35     if(n==2)
36         return true;
37     if(n<2 || !(n&1))
38         return false;
39     unsigned long long a,u(n-1),x,y;
40     short t(0),i;
41     while(!(u&1))
42     {
43         ++t;
44         u>>=1;
45     }
46     while(T-->0)
47     {
48         a=rand()%(n-1)+1;
49         x=exp_mod(a,u,n);
50         for(i=0;i<t;++i)
51         {
52             y=multi_mod(x,x,n);
53             if(y==1 && x!=1 && x!=n-1)
54                 return false;
55             x=y;
56         }
57         if(y!=1)
58             return false;
59     }
60     return true;
61 }

```

6.13 Multiset

```

1 Permutation:
2 MultiSet S={1 m,4 s,4 i,2 p}
3 P(S)=(1+4+4+2)!/1!/4!/4!/2!
4
5 Combination:
6 MultiSet S={ a1, a2,... ak}
7 C(S,r)=(r+k-1)!/r!/(k-1)! =C(r,r+k-1)
8
9 if(r>min{count(element[i])})
10     you have to resolve this problem with inclusion-exclusion
    principle.
11
12 MS T={3 a,4 b,5 c}
13 MS T*={ a, b, c}
14 A1={C(T*,10)|count(a)>3} // C(6,8)
15 A2={C(T*,10)|count(b)>4} // C(5,7)
16 A3={C(T*,10)|count(c)>5} // C(4,6)
17
18 C(T,10)=C(T*,10)-(|A1|+|A2|+|A3|)+(|A1 A2|+|A1 A3|+|A2 A3|)-|A1 A2
    A3|
19 C(10,12) C(1,3) C(0,2) 0 0
20 ans=6

```

6.14 Pell's equation

```

1 find the (x,y)pair that x^2-n*y^2=1
2 these is not solution if and only if n is a square number.
3
4 solution:

```

```

5 simply brute-force search the integer y, get (x1,y1). ( O(sqrt(n))
  )
6 or we can enumerate the continued fraction of sqrt(n), as (x/y),
  it will be much more faster
7
8 other solution pairs' matrix:
9 [ x1 n*y1 ]
10 [ y1 x1 ]
11 k-th solution is pow({matrix},k)

```

6.15 Pollard's rho algorithm

```

1 #include<stdio>
2 #include<stdlib>
3 #include<list>
4
5 short T;
6 unsigned long long a;
7 std::list<unsigned long long>fac;
8
9 inline unsigned long long multi_mod(const unsigned long long &a,
  unsigned long long b,const unsigned long long &n)
10 {
11     unsigned long long exp(a%n),tmp(0);
12     while(b)
13     {
14         if(b&1)
15         {
16             tmp+=exp;
17             if(tmp>n)
18                 tmp-=n;
19         }
20         exp<<=1;
21         if(exp>n)
22             exp-=n;
23         b>>=1;
24     }
25     return tmp;
26 }
27
28 inline unsigned long long exp_mod(unsigned long long a,unsigned
  long long b,const unsigned long long &c)
29 {
30     unsigned long long tmp(1);
31     while(b)
32     {
33         if(b&1)
34             tmp=multi_mod(tmp,a,c);
35         a=multi_mod(a,a,c);
36         b>>=1;
37     }
38     return tmp;
39 }
40
41 inline bool miller_rabbin(const unsigned long long &n,short T)
42 {
43     if(n==2)
44         return true;
45     if(n<2 || !(n&1))
46         return false;
47     unsigned long long a,u(n-1),x,y;
48     short t(0),i;
49     while(!(u&1))
50     {
51         ++t;
52         u>>=1;
53     }
54     while(T-->0)
55     {
56         a=rand()%(n-1)+1;
57         x=exp_mod(a,u,n);
58         for(i=0;i<t;++i)
59         {
60             y=multi_mod(x,x,n);
61             if(y==1 && x!=1 && x!=n-1)
62                 return false;
63             x=y;
64         }
65         if(y!=1)
66             return false;
67     }
68     return true;
69 }
70
71 unsigned long long gcd(const unsigned long long &a,const unsigned
  long long &b)
72 {
73     return b?gcd(b,a%b):a;
74 }
75
76 inline unsigned long long pollar_rho(const unsigned long long n,
  const unsigned long long &c)
77 {
78     unsigned long long x(rand()%(n-1)+1),y,d,i(1),k(2);
79     y=x;
80     while(true)
81     {
82         ++i;
83         x=(multi_mod(x,x,n)+c)%n;
84         d=gcd((x-y+n)%n,n);
85         if(d>1 && d<n)
86             return d;
87         if(x==y)
88             return n;
89         if(i==k)
90         {
91             k<<=1;
92             y=x;
93         }
94     }
95 }
96
97 void find(const unsigned long long &n,short c)
98 {

```

```

99     if(n==1)
100         return;
101     if(miller_rabbin(n,6))
102     {
103         fac.push_back(n);
104         return;
105     }
106     unsigned long long p(n);
107     short k(c);
108     while(p>=n)
109         p=pollar_rho(p,c--);
110     find(p,k);
111     find(n/p,k);
112 }
113
114 int main()
115 {
116     scanf("%hd",&T);
117     while(T-->0)
118     {
119         scanf("%llu",&a);
120         fac.clear();
121         find(a,120);
122         if(fac.size()==1)
123             puts("Prime");
124         else
125         {
126             fac.sort();
127             printf("%llu\n",fac.front());
128         }
129     }
130     return 0;
131 }

```

6.16 Prime

```

1  #include<vector>
2
3  std::vector<int>prm;
4  bool flag[MAXX];
5
6  int main()
7  {
8      prm.reserve(MAXX); // pi(x)=x/ln(x);
9      for(i=2;i<MAXX;++i)
10      {
11          if(!flag[i])
12              prm.push_back(i);
13          for(j=0;j<prm.size() && i*prm[j]<MAXX;++j)
14          {
15              flag[i*prm[j]]=true;
16              if(i%prm[j]==0)
17                  break;
18          }
19      }
20      return 0;
21 }

```

6.17 Reduced Residue System

```

1  Euler's totient function:
2
3  nnnn
4  (1)=111
5  m,n(mn)=(m)(n)
6
7  inline long long phi(int n)
8  {
9      static int i;
10     static int re;
11     re=n;
12     for(i=0;prm[i]*prm[i]<=n;++i)
13         if(n%prm[i]==0)
14         {
15             re-=re/prm[i];
16             do
17                 n/=prm[i];
18             while(n%prm[i]==0);
19         }
20     if(n!=1)
21         re-=re/n;
22     return re;
23 }
24
25 inline void Euler()
26 {
27     static int i,j;
28     phi[1]=1;
29     for(i=2;i<MAXX;++i)
30         if(!phi[i])
31             for(j=i;j<MAXX;j+=i)
32             {
33                 if(!phi[j])
34                     phi[j]=j;
35                 phi[j]=phi[j]/i*(i-1);
36             }
37 }
38
39 Multiplicative order:
40
41 the multiplicative order of a modulo n is the smallest positive
42 integer k with
43 a^k ≡ 1 (mod n).
44
45 mx,ord(x) (m) (aka. Euler's totient theorem)
46
47 :
48 method l(m) (m) d pow(x,d,m)==1;

```

```

48 method 2
49 inline long long ord(long long x,long long m)
50 {
51     static long long ans;
52     static int i,j;
53     ans=phi(m);
54     for(i=0;i<fac.size();++i)
55         for(j=0;j<fac[i].second && pow(x,ans/fac[i].first,m)!=1ll;++j)
56             ans/=fac[i].first;
57     return ans;
58 }
59
60 Primitive root:
61
62 ord(x)=(m)xm
63 pow(x,d) {d(m)} pow(x,d)%m==1 dd(m)xm
64
65 m= 1,2,4,pow(p,n),2*pow(p,n) {p,n} m //
66
67 m((m))
68
69 iip[j],pow(i,(m)/p[j])%mlimord(i)==(m)
70
71 Carmichael function:
72
73 (n) is defined as the smallest positive integer m such that
74 pow(a,m)%n==1 { for a!=1 && gcd(a,n)=1 }
75 (1)x lcm(ord(x))
76
77 if n=pow(p[0],a[0])*pow(p[1],a[1])*...*pow(p[m-1],a[m-1])
78 then (n)=lcm((pow(p[0],a[0])),(pow(p[1],a[1])),...,(pow(p[m-1],a[m-1])));
79
80 if n=pow(2,a)*pow(p[0],a[0])*pow(p[1],a[1])*...*pow(p[m-1],a[m-1])
81 then (n)=lcm(pow(2,c),(pow(p[0],a[0])),(pow(p[1],a[1])),...,(pow(p[m-1],a[m-1])));
82
83 if n=pow(2,a)*pow(p[0],a[0])*pow(p[1],a[1])*...*pow(p[m-1],a[m-1])
84 then (n)=lcm(pow(2,c),(pow(p[0],a[0])),(pow(p[1],a[1])),...,(pow(p[m-1],a[m-1])));
85 { c=0 if a<2; c=1 if a==2; c=a-2 if a>3; }
86
87 Carmichael's theorem:
88 if gcd(a,n)=1
89 then pow(a,(n))%n==1
90

```

6.18 System of linear congruences

```

1  // minimal val that for all (m,a) , val%m == a
2  #include<cstdio>
3
4  #define MAXX 11
5
6  int T,t;
7  int m[MAXX],a[MAXX];
8  int n,i,j,k;
9  int x,y,c,d;
10 int lcm;
11
12 int exgcd(int a,int b,int &x,int &y)
13 {
14     if(b)
15     {
16         int re(exgcd(b,a%b,x,y)),tmp(x);
17         x=y;
18         y=tmp-(a/b)*y;
19         return re;
20     }
21     x=1;
22     y=0;
23     return a;
24 }
25
26 int main()
27 {
28     scanf("%d",&T);
29     for(t=1;t<=T;++t)
30     {
31         scanf("%d",&n);
32         lcm=1;
33         for(i=0;i<n;++i)
34         {
35             scanf("%d",&m[i]);
36             lcm*=m[i]/exgcd(lcm,m[i],x,y);
37         }
38         for(i=0;i<n;++i)
39             scanf("%d",&a[i]);
40         for(i=1;i<n;++i)
41         {
42             c=a[i]-a[0];
43             d=exgcd(m[0],m[i],x,y);
44             if(c%d)
45                 break;
46             y=m[i]/d;
47             c/=d;
48             x=(x*c%y+y)%y;
49             a[0]+=m[0]*x;
50             m[0]*=y;
51         }
52         printf("Case %d: %d\n",t,i<n?-1:(a[0]?a[0]:lcm));
53     }
54     return 0;
55 }

```

7 others

7.1 .vimrc

```

1 set number
2 set history=1000000
3 set autoindent
4 set smartindent
5 set tabstop=4
6 set shiftwidth=4
7 set expandtab
8 set showmatch
9
10 set nocomp
11 filetype plugin indent on
12
13 filetype on
14 syntax on

```

7.2 bigint

```

1 // header files
2 #include <cstdio>
3 #include <string>
4 #include <algorithm>
5 #include <iostream>
6
7 struct Bigint
8 {
9     // representations and structures
10     std::string a; // to store the digits
11     int sign; // sign = -1 for negative numbers, sign = 1 otherwise
12     // constructors
13     Bigint() {} // default constructor
14     Bigint( std::string b ) { (*this) = b; } // constructor for std
15         ::string
16     // some helpful methods
17     int size() // returns number of digits
18     {
19         return a.size();
20     }
21     Bigint inverseSign() // changes the sign
22     {
23         sign *= -1;
24         return (*this);
25     }
26     Bigint normalize( int newSign ) // removes leading 0, fixes
27         sign
28     {
29         for( int i = a.size() - 1; i > 0 && a[i] == '0'; i-- )
30             a.erase(a.begin() + i);
31         sign = ( a.size() == 1 && a[0] == '0' ) ? 1 : newSign;
32         return (*this);
33     }
34     // assignment operator
35     void operator = ( std::string b ) // assigns a std::string to
36         Bigint
37     {
38         a = b[0] == '-' ? b.substr(1) : b;
39         reverse( a.begin(), a.end() );
40         this->normalize( b[0] == '-' ? -1 : 1 );
41     }
42     // conditional operators
43     bool operator < ( const Bigint &b ) const // less than operator
44     {
45         if( sign != b.sign )
46             return sign < b.sign;
47         if( a.size() != b.a.size() )
48             return sign == 1 ? a.size() < b.a.size() : a.size() > b.a
49                 .size();
50         for( int i = a.size() - 1; i >= 0; i-- )
51             if( a[i] != b.a[i] )
52                 return sign == 1 ? a[i] < b.a[i] : a[i] > b.a[i];
53         return false;
54     }
55     bool operator == ( const Bigint &b ) const // operator for
56         equality
57     {
58         return a == b.a && sign == b.sign;
59     }
60     // mathematical operators
61     Bigint operator + ( Bigint b ) // addition operator overloading
62     {
63         if( sign != b.sign )
64             return (*this) - b.inverseSign();
65         Bigint c;
66         for(int i = 0, carry = 0; i<a.size() || i<b.size() || carry;
67             i++)
68         {
69             carry+=(i<a.size() ? a[i]-48 : 0)+(i<b.size() ? b.a[i]
70                 -48 : 0);
71             c.a += (carry % 10 + 48);
72             carry /= 10;
73         }
74         return c.normalize(sign);
75     }
76     Bigint operator - ( Bigint b ) // subtraction operator
77         overloading
78     {
79         if( sign != b.sign )
80             return (*this) + b.inverseSign();
81         int s = sign; sign = b.sign = 1;
82         if( (*this) < b )
83             return ((b - (*this)).inverseSign()).normalize(-s);
84         Bigint c;
85         for( int i = 0, borrow = 0; i < a.size(); i++ )
86         {
87             borrow = a[i] - borrow - (i < b.size() ? b.a[i] : 48);

```

```

82             c.a += borrow >= 0 ? borrow + 48 : borrow + 58;
83             borrow = borrow >= 0 ? 0 : 1;
84         }
85         return c.normalize(s);
86     }
87     Bigint operator * ( Bigint b ) // multiplication operator
88         overloading
89     {
90         Bigint c("0");
91         for( int i = 0, k = a[i] - 48; i < a.size(); i++, k = a[i] -
92             48 )
93         {
94             while(k-->0)
95                 c = c + b; // ith digit is k, so, we add k times
96             b.a.insert(b.a.begin(), '0'); // multiplied by 10
97         }
98         return c.normalize(sign * b.sign);
99     }
100     Bigint operator / ( Bigint b ) // division operator overloading
101     {
102         if( b.size() == 1 && b.a[0] == '0' )
103             b.a[0] /= ( b.a[0] - 48 );
104         Bigint c("0"), d;
105         for( int j = 0; j < a.size(); j++ )
106             d.a += "0";
107         int dSign = sign * b.sign;
108         b.sign = 1;
109         for( int i = a.size() - 1; i >= 0; i-- )
110         {
111             c.a.insert( c.a.begin(), '0' );
112             c = c + a.substr( i, 1 );
113             while( !( c < b ) )
114             {
115                 c = c - b;
116                 d.a[i]++;
117             }
118             return d.normalize(dSign);
119         }
120     }
121     Bigint operator % ( Bigint b ) // modulo operator overloading
122     {
123         if( b.size() == 1 && b.a[0] == '0' )
124             b.a[0] /= ( b.a[0] - 48 );
125         Bigint c("0"), d;
126         b.sign = 1;
127         for( int i = a.size() - 1; i >= 0; i-- )
128         {
129             c.a.insert( c.a.begin(), '0' );
130             c = c + a.substr( i, 1 );
131             while( !( c < b ) )
132                 c = c - b;
133         }
134         return c.normalize(sign);
135     }
136     // output method
137     void print()
138     {
139         if( sign == -1 )
140             putchar('-');
141         for( int i = a.size() - 1; i >= 0; i-- )
142             putchar(a[i]);
143     }
144 };
145
146 int main()
147 {
148     Bigint a, b, c; // declared some Bigint variables
149     // taking Bigint input //
150     // taking Bigint input //
151     // taking Bigint input //
152     // taking Bigint input //
153     std::string input; // std::string to take input
154     std::cin >> input; // take the Big integer as std::string
155     a = input; // assign the std::string to Bigint a
156     std::cin >> input; // take the Big integer as std::string
157     b = input; // assign the std::string to Bigint b
158     // Using mathematical operators //
159     // Using mathematical operators //
160     // Using mathematical operators //
161     // Using mathematical operators //
162     c = a + b; // adding a and b
163     c.print(); // printing the Bigint
164     puts(""); // newline
165     c = a - b; // subtracting b from a
166     c.print(); // printing the Bigint
167     puts(""); // newline
168     c = a * b; // multiplying a and b
169     c.print(); // printing the Bigint
170     puts(""); // newline
171     c = a / b; // dividing a by b
172     c.print(); // printing the Bigint
173     puts(""); // newline
174     c = a % b; // a modulo b
175     c.print(); // printing the Bigint
176     puts(""); // newline
177     // Using conditional operators //
178     // Using conditional operators //
179     // Using conditional operators //
180     // Using conditional operators //
181     if( a == b )
182         puts("equal"); // checking equality
183     else
184         puts("not equal");
185     if( a < b )
186         puts("a is smaller than b"); // checking less than operator
187     return 0;
188 }

```

7.3 Binary Search

```

1  // [0,n)
2  inline int go(int A[],int n,int x) // return the least i that make
   A[i]==x;
3  {
4      static int l,r,mid,re;
5      l=0;
6      r=n-1;
7      re=-1;
8      while(l<=r)
9      {
10         mid=l+r>>1;
11         if(A[mid]<x)
12             l=mid+1;
13         else
14         {
15             r=mid-1;
16             if(A[mid]==x)
17                 re=mid;
18         }
19     }
20     return re;
21 }
22
23 inline int go(int A[],int n,int x) // return the largest i that
   make A[i]==x;
24 {
25     static int l,r,mid,re;
26     l=0;
27     r=n-1;
28     re=-1;
29     while(l<=r)
30     {
31         mid=l+r>>1;
32         if(A[mid]<=x)
33         {
34             l=mid+1;
35             if(A[mid]==x)
36                 re=mid;
37         }
38         else
39             r=mid-1;
40     }
41     return re;
42 }
43
44 inline int go(int A[],int n,int x) // retrun the largest i that
   make A[i]<x;
45 {
46     static int l,r,mid,re;
47     l=0;
48     r=n-1;
49     re=-1;
50     while(l<=r)
51     {
52         mid=l+r>>1;
53         if(A[mid]<x)
54         {
55             l=mid+1;
56             re=mid;
57         }
58         else
59             r=mid-1;
60     }
61     return re;
62 }
63
64 inline int go(int A[],int n,int x) // return the largest i that
   make A[i]<=x;
65 {
66     static int l,r,mid,re;
67     l=0;
68     r=n-1;
69     re=-1;
70     while(l<=r)
71     {
72         mid=l+r>>1;
73         if(A[mid]<=x)
74         {
75             l=mid+1;
76             re=mid;
77         }
78         else
79             r=mid-1;
80     }
81     return re;
82 }
83
84 inline int go(int A[],int n,int x) // return the least i that make
   A[i]>x;
85 {
86     static int l,r,mid,re;
87     l=0;
88     r=n-1;
89     re=-1;
90     while(l<=r)
91     {
92         mid=l+r>>1;
93         if(A[mid]<=x)
94             l=mid+1;
95         else
96         {
97             r=mid-1;
98             re=mid;
99         }
100    }
101    return re;
102 }
103
104 inline int go(int A[],int n,int x) // upper_bound();
105 {
106     static int l,r,mid;
107     l=0;
108     r=n-1;
109     while(l<r)
110     {
111         mid=l+r>>1;

```

```

112         if(A[mid]<=x)
113             l=mid+1;
114         else
115             r=mid;
116     }
117     return r;
118 }
119
120 inline int go(int A[],int n,int x) // lower_bound();
121 {
122     static int l,r,mid;
123     l=0;
124     r=n-1;
125     while(l<r)
126     {
127         mid=l+r>>1;
128         if(A[mid]<x)
129             l=mid+1;
130         else
131             r=mid;
132     }
133     return r;
134 }

```

7.4 java

```

1  //Scanner
2
3  Scanner in=new Scanner(new FileReader("asdf"));
4  PrintWriter pw=new PrintWriter(new FileWriter("out"));
5  boolean in.hasNext();
6  String in.next();
7  BigDecimal in.nextBigDecimal();
8  BigInteger in.nextBigInteger();
9  BigInteger in.nextBigInteger(int radix);
10 double in.nextDouble();
11 int in.nextInt();
12 int in.nextInt(int radix);
13 String in.nextLine();
14 long in.nextLong();
15 long in.nextLong(int radix);
16 short in.nextShort();
17 short in.nextShort(int radix);
18 int in.radix(); //Returns this scanner's default radix.
19 Scanner in.useRadix(int radix); // Sets this scanner's default
   radix to the specified radix.
20 void in.close(); //Closes this scanner.
21
22 //String
23
24 char str.charAt(int index);
25 int str.compareTo(String anotherString); // <0 if less. ==0 if
   equal. >0 if greater.
26 int str.compareToIgnoreCase(String str);
27 String str.concat(String str);
28 boolean str.contains(CharSequence s);
29 boolean str.endsWith(String suffix);
30 boolean str.startsWith(String prefix);
31 boolean str.startsWith(String prefix,int toffset);
32 int str.hashCode();
33 int str.indexOf(int ch);
34 int str.indexOf(int ch,int fromIndex);
35 int str.indexOf(String str);
36 int str.indexOf(String str,int fromIndex);
37 int str.lastIndexOf(int ch);
38 int str.lastIndexOf(int ch,int fromIndex);
39 //ry
40 int str.length();
41 String str.substring(int beginIndex);
42 String str.substring(int beginIndex,int endIndex);
43 String str.toLowerCase();
44 String str.toUpperCase();
45 String str.trim(); // Returns a copy of the string, with leading
   and trailing whitespace omitted.
46
47 //StringBuilder
48 StringBuilder str.insert(int offset,...);
49 StringBuilder str.reverse();
50 void str.setCharAt(int index,int ch);
51
52 //BigInteger
53 compareTo(); equals(); doubleValue(); longValue(); hashCode();
   toString(); toString(int radix); max(); min(); mod();
   modPow(BigInteger exp, BigInteger m); nextProbablePrime();
   pow();
54 andNot(); and(); xor(); not(); or(); getLowestSetBit(); bitCount();
   bitLength(); setBit(int n); shiftLeft(int n); shiftRight(int
   n);
55 add(); divide(); divideAndRemainder(); remainder(); multiply();
   subtract(); gcd(); abs(); signum(); negate();
56
57 //BigDecimal
58 movePointLeft(); movePointRight(); precision(); stripTrailingZeros();
   toBigInteger(); toPlainString();
59
60
61 //sort
62 class pii implements Comparable
63 {
64     public int a,b;
65     public int compareTo(Object i)
66     {
67         pii c=(pii)i;
68         return a==c.a?c.b-b:c.a-a;
69     }
70 }
71
72 class Main
73 {
74     public static void main(String[] args)
75     {
76         pii[] the=new pii[2];
77         the[0]=new pii();
78         the[1]=new pii();

```

```

79     the[0].a=1;
80     the[0].b=1;
81     the[1].a=1;
82     the[1].b=2;
83     Arrays.sort(the);
84     for(int i=0;i<2;++i)
85         System.out.printf("%d %d\n",the[i].a,the[i].b);
86     }
87 }

```

7.5 others

```

1  god damn it windows:
2  #pragma comment(linker, "/STACK:16777216")
3  #pragma comment(linker, "/STACK:102400000,102400000")
4
5
6  chmod +x [filename]
7
8  while true; do
9      ./gen > input
10     ./sol < input > output.sol
11     ./bf < input > output.bf
12
13     diff output.sol output.bf
14     if[ $? -ne 0];then break fi
15 done
16
17
18 1
19 2calm_down();calm_down();calm_down();
20 3
21 4
22 5/
23 6//hash//
24 6.1 or
25 6.2

```

8 search

8.1 dlx

```

1
2 011
3
4
5 011

```

8.2 dlx - exact cover

```

1  #include<cstdio>
2  #include<cstring>
3  #include<algorithm>
4  #include<vector>
5
6  #define N 256
7  #define MAXN N*22
8  #define MAXM N*5
9  #define inf 0x3f3f3f3f
10 const int MAXX (MAXN+MAXM);
11
12 bool mat[MAXN][MAXM];
13
14 int u[MAXX],d[MAXX],l[MAXX],r[MAXX],ch[MAXX],rh[MAXX];
15 int sz[MAXM];
16 std::vector<int>ans(MAXX);
17 int hd,cnt;
18
19 inline int node(int up,int down,int left,int right)
20 {
21     u[cnt]=up;
22     d[cnt]=down;
23     l[cnt]=left;
24     r[cnt]=right;
25     u[down]=d[up]=l[right]=r[left]=cnt;
26     return cnt++;
27 }
28
29 inline void init(int n,int m)
30 {
31     cnt=0;
32     hd=node(0,0,0,0);
33     static int i,j,k,r;
34     for(j=1;j<=m;++j)
35     {
36         ch[j]=node(cnt,cnt,l[hd],hd);
37         sz[j]=0;
38     }
39     for(i=1;i<=n;++i)
40     {
41         r=-1;
42         for(j=1;j<=m;++j)
43             if(mat[i][j])
44             {
45                 if(r==-1)
46                 {
47                     r=node(u[ch[j]],ch[j],cnt,cnt);
48                     rh[r]=i;
49                     ch[r]=ch[j];
50                 }

```

```

51         else
52         {
53             k=node(u[ch[j]],ch[j],l[r],r);
54             rh[k]=i;
55             ch[k]=ch[j];
56         }
57         ++sz[j];
58     }
59 }
60
61 inline void rm(int c)
62 {
63     l[r[c]]=l[c];
64     r[l[c]]=r[c];
65     static int i,j;
66     for(i=d[c];i!=c;i=d[i])
67         for(j=r[i];j!=i;j=r[j])
68         {
69             u[d[j]]=u[j];
70             d[u[j]]=d[j];
71             --sz[ch[j]];
72         }
73 }
74
75 inline void add(int c)
76 {
77     static int i,j;
78     for(i=u[c];i!=c;i=u[i])
79         for(j=l[i];j!=i;j=l[j])
80         {
81             ++sz[ch[j]];
82             u[d[j]]=d[u[j]]=j;
83         }
84     l[r[c]]=r[l[c]]=c;
85 }
86
87 bool dlx(int k)
88 {
89     if(hd==r[hd])
90     {
91         ans.resize(k);
92         return true;
93     }
94     int s=inf,c;
95     int i,j;
96     for(i=r[hd];i!=hd;i=r[i])
97         if(sz[i]<s)
98         {
99             s=sz[i];
100             c=i;
101         }
102     rm(c);
103     for(i=d[c];i!=c;i=d[i])
104     {
105         ans[k]=rh[i];
106         for(j=r[i];j!=i;j=r[j])
107             rm(ch[j]);
108         if(dlx(k+1))
109             return true;
110         for(j=l[i];j!=i;j=l[j])
111             add(ch[j]);
112     }
113     add(c);
114     return false;
115 }
116
117 #include <stdio>
118 #include <string>
119
120 #define N 1024
121 #define M 1024*110
122 using namespace std;
123
124 int l[M], r[M], d[M], u[M], col[M], row[M], h[M], res[N], cntcol[N];
125
126 int dcnt = 0;
127 //
128 inline void addnode(int &x)
129 {
130     ++x;
131     r[x] = l[x] = u[x] = d[x] = x;
132 }
133 //xrowx
134 inline void insert_row(int rowx, int x)
135 {
136     r[l[rowx]] = x;
137     l[x] = l[rowx];
138     r[x] = rowx;
139     l[rowx] = x;
140 }
141 //xcolx
142 inline void insert_col(int colx, int x)
143 {
144     d[u[colx]] = x;
145     u[x] = u[colx];
146     d[x] = colx;
147     u[colx] = x;
148 }
149 //
150 inline void dlx_init(int cols)
151 {
152     memset(h, -1, sizeof(h));
153     memset(cntcol, 0, sizeof(cntcol));
154     dcnt = -1;
155     addnode(dcnt);
156     for (int i = 1; i <= cols; ++i)
157     {
158         addnode(dcnt);
159         insert_row(0, dcnt);
160     }
161 }
162 //
163 inline void remove(int c)
164 {
165     l[r[c]] = l[c];
166     r[l[c]] = r[c];
167     for (int i = d[c]; i != c; i = d[i])
168         for (int j = r[i]; j != i; j = r[j])
169

```

```

170         u[d[j]] = u[j];
171         d[u[j]] = d[j];
172         cntcol[col[j]--];
173     }
174 }
175 //
176 inline void resume(int c)
177 {
178     for (int i = u[c]; i != c; i = u[i])
179         for (int j = l[i]; j != i; j = l[j])
180         {
181             u[d[j]] = j;
182             d[u[j]] = j;
183             cntcol[col[j]]++;
184         }
185     l[r[c]] = c;
186     r[l[c]] = c;
187 }
188 //
189 bool DLX(int deep)
190 {
191     if (r[0] == 0)
192     {
193         //Do anything you want to do here
194         printf("%d", deep);
195         for (int i = 0; i < deep; ++i) printf(" %d", res[i]);
196         puts("");
197         return true;
198     }
199     int min = INT_MAX, tempc;
200     for (int i = r[0]; i != 0; i = r[i])
201         if (cntcol[i] < min)
202         {
203             min = cntcol[i];
204             tempc = i;
205         }
206     remove(tempc);
207     for (int i = d[tempc]; i != tempc; i = d[i])
208     {
209         res[deep] = row[i];
210         for (int j = r[i]; j != i; j = r[j]) remove(col[j]);
211         if (DLX(deep + 1)) return true;
212         for (int j = l[i]; j != i; j = l[j]) resume(col[j]);
213     }
214     resume(tempc);
215     return false;
216 }
217 //"}"
218 inline void insert_node(int x, int y)
219 {
220     cntcol[y]++;
221     addnode(dcnt);
222     row[dcnt] = x;
223     col[dcnt] = y;
224     insert_col(y, dcnt);
225     if (h[x] == -1) h[x] = dcnt;
226     else insert_row(h[x], dcnt);
227 }
228 int main()
229 {
230     int n, m;
231     while ("scanf("%d%d", &n, &m))
232     {
233         dlx_init(m);
234         for (int i = 1; i <= n; ++i)
235         {
236             int k, x;
237             scanf("%d", &k);
238             while (k--)
239             {
240                 scanf("%d", &x);
241                 insert_node(i, x);
242             }
243         }
244         if (!DLX(0))
245             puts("NO");
246     }
247     return 0;
248 }

```

8.3 dlx - repeat cover

```

1  #include<cstdio>
2  #include<cstring>
3  #include<algorithm>
4
5  #define MAXN 110
6  #define MAXM 1000000
7  #define INF 0x7FFFFFFF
8
9  using namespace std;
10
11 int G[MAXN][MAXN];
12 int L[MAXN], R[MAXN], U[MAXN], D[MAXN];
13 int size, ans, S[MAXN], H[MAXN], C[MAXN];
14 bool vis[MAXN * 100];
15 void Link(int r, int c)
16 {
17     U[size] = c;
18     D[size] = D[c];
19     U[D[c]] = size;
20     D[c] = size;
21     if (H[r] < 0)
22         H[r] = L[size] = R[size] = size;
23     else
24     {
25         L[size] = H[r];
26         R[size] = R[H[r]];
27         L[R[H[r]]] = size;
28         R[H[r]] = size;
29     }
30     S[c]++;
31     C[size++] = c;
32 }

```

```

33 void Remove(int c)
34 {
35     int i;
36     for (i = D[c]; i != c; i = D[i])
37     {
38         L[R[i]] = L[i];
39         R[L[i]] = R[i];
40     }
41 }
42 void Resume(int c)
43 {
44     int i;
45     for (i = D[c]; i != c; i = D[i])
46         L[R[i]] = R[L[i]] = i;
47 }
48 int A()
49 {
50     int i, j, k, res;
51     memset(vis, false, sizeof(vis));
52     for (res = 0, i = R[0]; i = R[i])
53     {
54         if (!vis[i])
55         {
56             res++;
57             for (j = D[i]; j != i; j = D[j])
58             {
59                 for (k = R[j]; k != j; k = R[k])
60                     vis[C[k]] = true;
61             }
62         }
63     }
64     return res;
65 }
66 void Dance(int now)
67 {
68     if (R[0] == 0)
69         ans = min(ans, now);
70     else if (now + A() < ans)
71     {
72         int i, j, temp, c;
73         for (temp = INF, i = R[0]; i = R[i])
74         {
75             if (temp > S[i])
76             {
77                 temp = S[i];
78                 c = i;
79             }
80         }
81         for (i = D[c]; i != c; i = D[i])
82         {
83             Remove(i);
84             for (j = R[i]; j != i; j = R[j])
85                 Remove(j);
86             Dance(now + 1);
87             for (j = L[i]; j != i; j = L[j])
88                 Resume(j);
89             Resume(i);
90         }
91     }
92 }
93 void Init(int m)
94 {
95     int i;
96     for (i = 0; i <= m; i++)
97     {
98         R[i] = i + 1;
99         L[i + 1] = i;
100         U[i] = D[i] = i;
101         S[i] = 0;
102     }
103     R[m] = 0;
104     size = m + 1;
105 }

```

8.4 fibonacci knapsack

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<algorithm>
4
5  #define MAXX 71
6
7  struct mono
8  {
9      long long weig, cost;
10 } goods[MAXX];
11
12 short n, T, t, i;
13 long long carry, sumw, sumc;
14 long long ans, las[MAXX];
15
16 int com(const void *n, const void *m)
17 {
18     struct mono *a = (struct mono *)n, *b = (struct mono *)m;
19     if (a->weig != b->weig)
20         return a->weig > b->weig;
21     else
22         return b->cost < a->cost;
23 }
24
25 bool comp(const struct mono a, const struct mono b)
26 {
27     if (a.weig != b.weig)
28         return a.weig < b.weig;
29     else
30         return b.cost < a.cost;
31 }
32
33 void dfs(short i, long long cost_n, long long carry_n, short last)
34 {
35     if (ans < cost_n)
36         ans = cost_n;
37     if (i == n || goods[i].weig > carry_n || cost_n + las[i] <= ans)
38         return;

```

```

39     if(last || (goods[i].weig!=goods[i-1].weig && goods[i].cost>
40         goods[i-1].cost))
41         dfs(i+1,cost_n+goods[i].cost,carry_n-goods[i].weig,1);
42     dfs(i+1,cost_n,carry_n,0);
43 }
44 int main()
45 {
46     // freopen("asdf","r",stdin);
47     scanf("%hd",&T);
48     for(t=1;t<=T;++t)
49     {
50         scanf("%hd%lld",&n,&carry);
51         sumw=0;
52         sumc=0;
53         ans=0;
54         for(i=0;i<n;++i)
55         {
56             scanf("%lld%lld",&goods[i].weig,&goods[i].cost);
57             sumw+=goods[i].weig;
58             sumc+=goods[i].cost;
59         }
60         if(sumw<=carry)
61         {
62             printf("Case %hd: %lld\n",t,sumc);
63             continue;
64         }
65         // qsort(goods,n,sizeof(struct mono),com);
66         std::sort(goods,goods+n,comp);
67         for(i=0;i<n;++i)
68         {
69             // printf("%lld %lld\n",goods[i].weig,goods[i].cost);
70             las[i]=sumc;
71             sumc-=goods[i].cost;
72         }
73         dfs(0,0,carry,1);
74         printf("Case %hd: %lld\n",t,ans);
75     }
76     return 0;
77 }

```

9 string

9.1 Aho-Corasick Algorithm

```

1 //trie graph
2 #include<cstring>
3 #include<queue>
4
5 #define MAX 1000111
6 #define N 26
7
8 int nxt[MAX][N],fal[MAX],cnt;
9 bool ed[MAX];
10 char buf[MAX];
11
12 inline void init(int a)
13 {
14     memset(nxt[a],0,sizeof(nxt[0]));
15     fal[a]=0;
16     ed[a]=false;
17 }
18
19 inline void insert()
20 {
21     static int i,p;
22     for(i=p=0;buf[i];++i)
23     {
24         if(!nxt[p][map[buf[i]]])
25             init(nxt[p][map[buf[i]]]=++cnt);
26         p=nxt[p][map[buf[i]]];
27     }
28     ed[p]=true;
29 }
30
31 inline void make()
32 {
33     static std::queue<int>q;
34     int i,now,p;
35     q.push(0);
36     while(!q.empty())
37     {
38         now=q.front();
39         q.pop();
40         for(i=0;i<N;++i)
41             if(nxt[now][i])
42             {
43                 q.push(p=nxt[now][i]);
44                 if(now)
45                     fal[p]=nxt[fal[now]][i];
46                 ed[p]=ed[fal[p]];
47             }
48         else
49             nxt[now][i]=nxt[fal[now]][i]; // trienxt
50     }
51 }
52
53 // normal version
54 #define N 128
55
56 char buf[MAXX];
57 int cnt[1111];
58
59 struct node
60 {
61     node *fal,*nxt[N];
62     int idx;
63     node() { memset(this,0,sizeof node); }
64 }*rt;
65
66 std::queue<node*>Q;

```

```

67
68 void free(node *p)
69 {
70     for(int i(0);i<N;++i)
71         if(p->nxt[i])
72             free(p->nxt[i]);
73     delete p;
74 }
75
76 inline void add(char *s,int idx)
77 {
78     static node *p;
79     for(p=rt;*s;++s)
80     {
81         if(!p->nxt[*s])
82             p->nxt[*s]=new node();
83         p=p->nxt[*s];
84     }
85     p->idx=idx;
86 }
87
88 inline void make()
89 {
90     Q.push(rt);
91     static node *p,*q;
92     static int i;
93     while(!Q.empty())
94     {
95         p=Q.front();
96         Q.pop();
97         for(i=0;i<N;++i)
98             if(p->nxt[i])
99             {
100                 q=p->fal;
101                 while(q)
102                 {
103                     if(q->nxt[i])
104                     {
105                         p->nxt[i]->fal=q->nxt[i];
106                         break;
107                     }
108                     q=q->fal;
109                 }
110                 if(!q)
111                     p->nxt[i]->fal=rt;
112                 Q.push(p->nxt[i]);
113             }
114     }
115 }
116
117 inline void match(const char *s)
118 {
119     static node *p,*q;
120     for(p=rt;*s;++s)
121     {
122         while(p!=rt && !p->nxt[*s])
123             p=p->fal;
124         p=p->nxt[*s];
125         if(!p)
126             p=rt;
127         for(q=p;q!=rt && q->idx;q=q->fal) // why q->idx ? looks like
128             // not necessary at all, I delete it in an other
129             // solution
130             ++cnt[q->idx];
131     }
132     //dfsfal
133     //BIT

```

9.2 Gusfield's Z Algorithm

```

1 inline void make(int *z,char *buf)
2 {
3     int i,j,l,r;
4     l=0;
5     r=1;
6     z[0]=strlen(buf);
7     for(i=1;i<z[0];++i)
8         if(r<=i || z[i-1]>=r-i)
9         {
10             j=std::max(i,r);
11             while(j<z[0] && buf[j]==buf[j-i])
12                 ++j;
13             z[i]=j-i;
14             if(i<j)
15             {
16                 l=i;
17                 r=j;
18             }
19         }
20         else
21             z[i]=z[i-1];
22 }
23
24 for(i=1;i<len && i+z[i]<len;++i); //i=

```

9.3 Manacher's Algorithm

```

1 #include<stdio>
2 #include<vector>
3
4 #define MAXX 1111
5
6 std::vector<char>str;
7 char buf[MAXX];
8 int z[MAXX<<1];
9 int i,j,l,r;

```



```

10 int ii,n,c;
11
12 inline int match(const int &a,const int &b)
13 {
14     int i(0);
15     while(a-i>0 && b+i<str.size() && str[a-i]==str[b+i])//i1
16         ++i;
17     return i;
18 }
19
20 int main()
21 {
22     gets(buf);
23     str.reserve(MAXX<<1);
24     for(i=0;buf[i];++i)
25     {
26         str.push_back('$');
27         str.push_back(buf[i]);
28     }
29     str.push_back('$');
30
31     z[0]=1;
32     c=l=r=0;
33     for(i=1;i<str.size();++i)
34     {
35         ii=(l<<1)-i;
36         n=r+1-i;
37
38         if(i>r)
39         {
40             z[i]=match(i,i);
41             l=i;
42             r=i+z[i]-1;
43         }
44         else
45             if(z[ii]==n)
46             {
47                 z[i]=n+match(i-n,i+n);
48                 l=i;
49                 r=i+z[i]-1;
50             }
51         else
52             z[i]=std::min(z[ii],n);
53         if(z[i]>z[c])
54             c=i;
55     }
56
57     for(i=c-z[c]+2,n=c+z[c];i<n;i+=2)
58         putchar(str[i]);
59     puts("");
60     return 0;
61 }
62
63 //package:
64
65 inline int match(const int a,const int b,const std::vector<int> &
66     str)
67 {
68     static int i;
69     i=0;
70     while(a-i>0 && b+i<str.size() && str[a-i]==str[b+i])
71         ++i;
72     return i;
73 }
74
75 inline void go(int *z,const std::vector<int> &str)
76 {
77     static int c,l,r,i,ii,n;
78     z[0]=1;
79     c=l=r=0;
80     for(i=1;i<str.size();++i)
81     {
82         ii=(l<<1)-i;
83         n=r+1-i;
84
85         if(i>r)
86         {
87             z[i]=match(i,i,str);
88             l=i;
89             r=i+z[i]-1;
90         }
91         else
92             if(z[ii]==n)
93             {
94                 z[i]=n+match(i-n,i+n,str);
95                 l=i;
96                 r=i+z[i]-1;
97             }
98         else
99             z[i]=std::min(z[ii],n);
100         if(z[i]>z[c])
101             c=i;
102     }
103
104 inline bool check(int *z,int a,int b) //[a,b]
105 {
106     a=a*2-1;
107     b=b*2-1;
108     int m=(a+b)/2;
109     return z[m]>=b-m+1;
110 }

```

9.4 Morris-Pratt Algorithm

```

1 inline void make(char *buf,int *fal)
2 {
3     static int i,j;
4     fal[0]=-1;
5     for(i=1,j=-1;buf[i];++i)
6     {
7         while(j>=0 && buf[j+1]!=buf[i])
8             j=fal[j];
9         if(buf[j+1]==buf[i])

```

```

10         ++j;
11         fal[i]=j;
12     }
13 }
14
15 inline int match(char *p,char *t,int* fal)
16 {
17     static int i,j,re;
18     re=0;
19     for(i=0,j=-1;t[i];++i)
20     {
21         while(j>=0 && p[j+1]!=t[i])
22             j=fal[j];
23         if(p[j+1]==t[i])
24             ++j;
25         if(!p[j+1])
26         {
27             ++re;
28             j=fal[j];
29         }
30     }
31     return re;
32 }
33

```

9.5 smallest representation

```

1 int min(char a[],int len)
2 {
3     int i = 0,j = 1,k = 0;
4     while (i < len && j < len && k < len)
5     {
6         int cmp = a[(j+k)%len]-a[(i+k)%len];
7         if (cmp == 0)
8             k++;
9         else
10         {
11             if (cmp > 0)
12                 j += k+1;
13             else
14                 i += k+1;
15             if (i == j) j++;
16             k = 0;
17         }
18     }
19     return std::min(i,j);
20 }

```

9.6 Suffix Array - DC3 Algorithm

```

1 #include<cstdio>
2 #include<cstring>
3 #include<algorithm>
4
5 #define MAXX 1111
6 #define F(x) ((x)/3+((x)%3==1?0:tb))
7 #define G(x) ((x)<tb?(x)*3+1:((x)-tb)*3+2)
8
9 int wa[MAXX],wb[MAXX],wv[MAXX],ws[MAXX];
10
11 inline bool c0(const int *str,const int &a,const int &b)
12 {
13     return str[a]==str[b] && str[a+1]==str[b+1] && str[a+2]==str[b+2];
14 }
15
16 inline bool c12(const int *str,const int &k,const int &a,const int &b)
17 {
18     if (k==2)
19         return str[a]<str[b] || str[a]==str[b] && c12(str,l,a+1,b+1);
20     else
21         return str[a]<str[b] || str[a]==str[b] && wv[a+1]<wv[b+1];
22 }
23
24 inline void sort(int *str,int *a,int *b,const int &n,const int &m)
25 {
26     memset(ws,0,sizeof(ws));
27     int i;
28     for(i=0;i<n;++i)
29         ++ws[wv[i]=str[a[i]]];
30     for(i=1;i<m;++i)
31         ws[i]+=ws[i-1];
32     for(i=n-1;i>=0;--i)
33         b[--ws[wv[i]]]=a[i];
34 }
35
36 inline void dc3(int *str,int *sa,const int &n,const int &m)
37 {
38     int *strn(str+n);
39     int *san(sa+n),tb((n+1)/3),ta(0),tbc(0),i,j,k;
40     str[n]=str[n+1]=0;
41     for(i=0;i<n;++i)
42         if(i%3)
43             wa[tbc++]=i;
44     sort(str+2,wa,wb,tbc,m);
45     sort(str+1,wb,wa,tbc,m);
46     sort(str,wa,wb,tbc,m);
47     for(i=j=1,strn[F(wb[0])]=0;i<tbc;++i)
48         strn[F(wb[i])]=c0(str,wb[i-1],wb[i])?j-1:j++;
49     if(j<tbc)
50         dc3(strn,san,tbc,j);
51     else
52         for(i=0;i<tbc;++i)
53             san[strn[i]]=i;
54     for(i=0;i<tbc;++i)
55         if(san[i]<tb)

```

```

56     wb[ta++]=san[i]*3;
57     if(n%3==1)
58         wb[ta+]=n-1;
59     sort(str,wb,wa,ta,m);
60     for(i=0;i<tb;+i)
61         wv[wb[i]]=G(san[i])=1;
62     for(i=j=k=0;i<ta && j<tb;+i)
63         sa[k++]=c12(str,wb[j]%3,wa[i],wb[j])?wa[i++]:wb[j++];
64     while(i<ta)
65         sa[k++]=wa[i++];
66     while(j<tb)
67         sa[k++]=wb[j++];
68 }
69
70 int rk[MAXX],lcpa[MAXX],sa[MAXX*3];
71 int str[MAXX*3]; //int
72
73 int main()
74 {
75     scanf("%d %d",&n,&j);
76     for(i=0;i<n;+i)
77     {
78         scanf("%d",&k);
79         num[i]=k-j+100;
80         j=k;
81     }
82     num[n]=0;
83
84     dc3(num,sa,n+1,191); //191: str
85
86     for(i=1;i<=n;+i) // rank
87         rk[sa[i]]=i;
88     for(i=k=0;i<n;+i) // lcp
89         if(!rk[i])
90             lcpa[0]=0;
91     else
92     {
93         j=sa[rk[i]-1];
94         if(k>0)
95             --k;
96         while(num[i+k]==num[j+k])
97             ++k;
98         lcpa[rk[i]]=k;
99     }
100
101     for(i=1;i<=n;+i)
102         sptb[0][i]=1;
103     for(i=1;i<=lg[n];+i) //sparse table RMQ
104     {
105         k=n+1-(1<<i);
106         for(j=1;j<=k;+j)
107         {
108             a=sptb[i-1][j];
109             b=sptb[i-1][j+(1<<(i-1))];
110             sptb[i][j]=lcpa[a]<lcpa[b]?a:b;
111         }
112     }
113 }
114
115
116 inline int ask(int l,int r)
117 {
118     a=lg[r-l+1];
119     r--(1<<a)-1;
120     l=sptb[a][l];
121     r=sptb[a][r];
122     return lcpa[l]<lcpa[r]?l:r;
123 }
124
125 inline int lcp(int l,int r) // [l,r]rmq
126 {
127     l=rk[l];
128     r=rk[r];
129     if(l>r)
130         std::swap(l,r);
131     return lcpa[ask(l+1,r)];
132 }

```

```

33     for(i=1; i<m; i++)
34         wss[i]+=wss[i-1];
35     for(i=n-1; i>=0; i--)
36         sa[--wss[wv[i]]]=y[i];
37     for(t=x,x=y,y=t,p=1,i=1,x[sa[0]]=0; i<n; i++)
38         x[sa[i]]=cmp(y,n,sa[i-1],sa[i],j)?p-1:p++;
39 }
40 for(int i=0; i<n; i++)
41     rank[sa[i]]=i;
42 for(int i=0,j=0,k=0; i<n; height[rank[i++]]=k)
43     if(rank[i]>0)
44         for(k?k--:0,j=sa[rank[i]-1]; i+k < n && j+k < n && str[i+k]==str[j+k]; ++k);
45 }

```

9.7 Suffix Array - Prefix-doubling Algorithm

```

1  int wx[maxn],wy[maxn],*x,*y,wss[maxn],wv[maxn];
2
3  bool cmp(int *x,int n,int a,int b,int l)
4  {
5      return a+l<n && b+l<n && r[a]==r[b]&&r[a+l]==r[b+l];
6  }
7  void da(int str[],int sa[],int rank[],int height[],int n,int m)
8  {
9      int *s = str;
10     int *x=wx,*y=wy,*t,p;
11     int i,j;
12     for(i=0; i<m; i++)
13         wss[i]=0;
14     for(i=0; i<n; i++)
15         wss[x[i]]=s[i]++;
16     for(i=1; i<m; i++)
17         wss[i]+=wss[i-1];
18     for(i=n-1; i>=0; i--)
19         sa[--wss[x[i]]]=i;
20     for(j=1,p=1; p<n && j<n; j*=2,m=p)
21     {
22         for(i=n-j,p=0; i<n; i++)
23             y[p++]=i;
24         for(i=0; i<n; i++)
25             if(sa[i]-j>0)
26                 y[p++]=sa[i]-j;
27         for(i=0; i<n; i++)
28             wv[i]=x[y[i]];
29         for(i=0; i<m; i++)
30             wss[i]=0;
31         for(i=0; i<n; i++)
32             wss[wv[i]]++;

```