

National Chiao Tung University

Team Reference Document

Contents

| | | |
|----------|--------------------------------------|-----------|
| 1 | Building Environment | 2 |
| 1.1 | Default | 2 |
| 1.2 | Print File | 2 |
| 1.3 | Vimrc | 2 |
| 2 | To Be Classify | 2 |
| 2.1 | AC Trie | 2 |
| 2.2 | BCC | 3 |
| 2.3 | BigInteger | 3 |
| 2.4 | Bipartite Matching | 4 |
| 2.5 | BK | 4 |
| 2.6 | Black Magic | 5 |
| 2.7 | Blossom | 5 |
| 2.8 | Dice | 6 |
| 2.9 | Dinic | 7 |
| 2.10 | Dinic Flow | 7 |
| 2.11 | Extgcd | 8 |
| 2.12 | General Weighted Matching | 8 |
| 2.13 | Geometry | 12 |
| 2.14 | Heavy Light Decomposition | 14 |
| 2.15 | Huafen | 15 |
| 2.16 | KDtree Insert | 16 |
| 2.17 | KM | 18 |
| 2.18 | KM N3 | 19 |
| 2.19 | KM No Big Int | 19 |
| 2.20 | Maximum Density | 20 |
| 2.21 | Min Cost Flow | 21 |
| 2.22 | Mincostflow | 21 |
| 2.23 | Monotone | 22 |
| 2.24 | MST Directed | 22 |
| 2.25 | NTT | 23 |
| 2.26 | SCC | 24 |
| 2.27 | Splay Tree | 25 |
| 2.28 | Stable Marriage | 26 |
| 2.29 | Suffix Array | 27 |
| 2.30 | Suffix Automaton | 27 |
| 2.31 | Tonelli Shanks | 28 |
| 2.32 | Treap | 29 |
| 2.33 | Z Algorithm | 29 |
| 3 | Lucas’s theorem | 29 |
| 4 | 無權邊的生成樹個數 Kirchhoff’s Theorem | 29 |
| 5 | monge | 29 |
| 6 | 四心 | 29 |
| 7 | Runge-Kutta | 30 |
| 8 | Householder Matrix | 30 |

1 Building Environment

1.1 Default

```

1 #define F(n) Fi(i,n)
2 #define Fi(i,n) Fl(i,0,n)
3 #define Fl(i,l,n) for(int i=l;i<n;++i)
4 #include <bits/stdc++.h>
5 #include <ext/pb_ds/assoc_container.hpp>
6 #include <ext/pb_ds/priority_queue.hpp>
7 using namespace std;
8 using namespace __gnu_pbds;
9 const double PI = acos(-1);
10 main() {
11     ios_base::sync_with_stdio(false);
12     cin.tie(NULL);
13     cout << fixed << setprecision(7) << PI << endl;
14 }

```

1.2 Print File

```

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

```

1.3 Vimrc

```

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

```

2 To Be Classify

2.1 AC Trie

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 const int MAXS = 1000100, MAXN = 10010, MAXP = 51;
4 char str[MAXS], pattern[MAXN][MAXP];
5 struct actrie
6 {
7     actrie *flink, *nxt[26]; //failure link, trie structure
8     int pcnt;
9     actrie()
10    {
11        flink = NULL, pcnt = 0;
12        memset(nxt, 0, sizeof(nxt));
13    }
14 };
15 actrie *root, *que[MAXN*MAXP];
16 void addPattern(char *P)
17 {
18     actrie *now = root;
19     for(int i = 0; P[i]; i++)
20     {
21         if(now->nxt[ P[i] - 'a' ] == NULL) now->nxt[ P[i] - 'a' ] = new
22         actrie();
23         now = now->nxt[ P[i] - 'a' ];
24     }
25     ++now->pcnt;
26 }
27 void build()
28 {
29     int front = 0, rear = 1;
30     que[0] = root;
31     while(front < rear)
32     {
33         actrie *now = que[front], *fnode;
34         for(int i = 0; i < 26; i++)
35             if(now->nxt[i])
36             {
37                 fnode = now->flink;
38                 while(fnode && fnode->nxt[i] == NULL) fnode = fnode->flink;
39
40                 if(fnode) now->nxt[i]->flink = fnode->nxt[i];
41                 else now->nxt[i]->flink = root;
42                 que[rear++] = now->nxt[i];
43             }
44         ++front;
45     }
46 }
47 int match(char * S)
48 {
49     int ret = 0;
50     actrie *now = root;
51     for(int i = 0; S[i]; i++)
52     {
53         while(now && now->nxt[ S[i] - 'a' ] == NULL) now = now->flink;
54         if(now)
55         {

```

```

56     now = now->nxt[ S[i]-'a' ];
57     actrie *temp = now;
58     while(temp && temp->pcnt != -1)
59     {
60         ret += temp->pcnt;
61         temp->pcnt = -1;
62         temp = temp->flink;
63     }
64 }
65     else now = root;
66 }
67 return ret;
68 }
69 int main(){
70     int T;
71     scanf("%d",&T);
72     while(T--){
73         int n;
74         root = new actrie();
75         scanf("%d",&n);
76         for(int i=0;i<n;i++){
77             scanf("%s",pattern[i]);
78             addPattern(pattern[i]);
79         }
80         build();
81         scanf("%s",str);
82         printf("%d\n",match(str));
83     }
84     return 0;
85 }

```

2.2 BCC

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  const int MAXN = 10000;
4  vector <int> adja[MAXN];
5  int gcnt, top, timeStamp, dfn[MAXN], low[MAXN], depth[MAXN];
6  pair<int, int> stk[MAXN], ans[MAXN];
7  set <int> group[MAXN];
8  bool cut[MAXN];
9  void BCC(int now, int nextv){
10     int sf, st;
11     group[gcnt].clear();
12     do{
13         sf = stk[top-1].first, st = stk[top-1].second;
14         group[gcnt].insert(sf);
15         group[gcnt].insert(st);
16         --top;
17     }while(sf != now || st != nextv);
18     ++gcnt;
19 }
20 void tarjan(int now, int parent, int d){
21     int child = 0;

```

```

22     dfn[now] = low[now] = ++timeStamp, depth[now] = d;
23     for(int i = 0; i < adja[now].size(); i++){
24         int nextv = adja[now][i];
25         if(nextv == parent) continue;
26         if(dfn[nextv] == 0){
27             stk[top++] = make_pair(now, nextv);
28             tarjan(nextv, now, d+1);
29             low[now] = min(low[now], low[nextv]);
30             ++child;
31             if( (parent != -1 && low[nextv] >= dfn[now]) || (parent == -1 &&
child >= 2)){
32                 cut[now] = true;
33                 if(parent != -1) BCC(now, nextv);
34             }
35             if(parent == -1) BCC(now, nextv);
36         }
37         else if(depth[nextv] < depth[now]-1){
38             stk[top++] = make_pair(now, nextv);
39             low[now] = min(low[now], dfn[nextv]);
40         }
41     }
42 }
43 int main(){
44     int n,m,x,y,cnt=0;
45     while(~scanf("%d",&n)){
46         cnt=timeStamp=top=gcnt=0;
47         memset(cut, 0, sizeof(cut));
48         memset(dfn, 0, sizeof(dfn));
49         for(int i=0;i<n;i++)adja[i].clear();
50         for(int i=0;i<n;i++){
51             scanf("%d ",&x);
52             scanf("(%d",&m);
53             while(m--){
54                 scanf("%d",&y);
55                 adja[x].push_back(y);
56             }
57         }
58         for(int i=0;i<n;i++)
59             if(dfn[i]==0)tarjan(i, -1, 1);
60         for(int i=0;i<gcnt;i++){
61             if(group[i].size()==2){
62                 //critical links
63             }
64         }
65     }
66 }

```

2.3 BigInteger

```

1  import java.math.*;
2  import java.io.*;
3  import java.util.*;
4  public class Main{
5      public static void main(String []argv){

```

```

6      c[0][0]=BigInteger.ONE;
7      for(int i=1;i<3001;i++){
8          c[i][0]=BigInteger.ONE;
9          c[i][i]=BigInteger.ONE;
10         for(int j=1;j<i;j++)c[i][j]=c[i-1][j].add(c[i-1][j-1]);
11     }
12     Scanner scanner = new Scanner(System.in);
13     int T = scanner.nextInt();
14     BigInteger x;
15     BigInteger ans;
16     while(T-- > 0){
17         ans = BigInteger.ZERO;
18         int n = scanner.nextInt();
19         for(int i=0;i<n;i++){
20             x = new BigInteger(scanner.next());
21             if(i%2 == 1)ans=ans.subtract(c[n-1][i].multiply(x));
22             else ans=ans.add(c[n-1][i].multiply(x));
23         }
24         if(n%2 == 0)ans=BigInteger.ZERO.subtract(ans);
25         System.out.println(ans);
26     }
27 }
28 }

```

2.4 Bipartite Matching

```

1 #include<bits/stdc++.h>
2 #define V 20100
3 #define inf 0x3f3f3f3f
4 int mx[V],my[V],dis[V],que[V];
5 bool vis[V];
6 vector<int> g[V];
7 bool DFS(int u){
8     vis[u]=true;
9     for(int i=0;i<g[u].size();i++){
10         int v=my[g[u][i]];
11         if(v==-1||!vis[v]&&dis[v]==dis[u]+1&&DFS(v)){
12             mx[u]=g[u][i];
13             my[g[u][i]]=u;
14             return true;
15         }
16     }
17     return false;
18 }
19 // n is the size of left hand side
20 int Hopcroft_Karp(int n){
21     int matching=0,qt,qf,sp,i,u,v;
22     bool flag=true;
23     memset(mx,-1,sizeof(mx));
24     memset(my,-1,sizeof(my));
25     while(flag){
26         flag=false;
27         qt=qf=0;
28         sp=inf;

```

```

29     for(i=0;i<n;i++){
30         if(mx[i]==-1){
31             dis[i]=0;
32             que[qt++]=i;
33         }
34         else dis[i]=inf;
35     }
36     while(qf<qt){
37         u=que[qf++];
38         if(dis[u]>=sp) continue;
39         for(i=0;i<g[u].size();i++){
40             v=my[g[u][i]];
41             if(v==-1){
42                 if(dis[u]+1<sp){
43                     sp=dis[u]+1;
44                     flag=true;
45                 }
46             }
47             else if(dis[u]+1<dis[v]){
48                 dis[v]=dis[u]+1;
49                 que[qt++]=v;
50             }
51         }
52     }
53     if(flag){
54         memset(vis,0,sizeof(vis));
55         for(i=0;i<n;i++){
56             if(dis[i]==0&&DFS(i)) matching++;
57         }
58     }
59 }
60 return matching;
61 }

```

2.5 BK

```

1 //vertex ordering: Keep removing the vertex with minimum degree(not added
  here)
2 typedef long long ll;
3 int n;
4 vector<ll> v, ne;
5
6 // ne[u] is the neighbours of u
7 // v is the result
8 void BronKerbosch(ll R, ll P, ll X){
9     if ((P == 0LL) && (X == 0LL)) {v.push_back(R);return;}
10    int u = 0;
11    for (; u < n; u++) if ( (P[X] & (1LL << u) ) break;
12    for (int i = 0; i < n; i++)
13        if ( (P&ne[u]) & (1LL << i) ){
14            BronKerbosch(R | (1LL << i), P & ne[i], X & ne[i]);
15            P -= (1LL << i); X |= (1LL << i);
16        }
17 }

```

2.6 Black Magic

```

1 #include<ext/rope>
2 using namespace std;
3 using namespace __gnu_cxx;
4 const int MAXN = 50000 + 10;
5 rope ro,l[MAXN],tmp;
6 char str[200+10];
7 main(){
8     int T,op,p,c,d=0,cnt=1,v;
9     scanf("%d",&T);
10    while(T--){
11        scanf("%d",&op);
12        if(op==1){
13            scanf("%d%s",&p,str);
14            p-=d;
15            ro.insert(p,str);
16            l[cnt++]=ro;
17        }
18        else if(op==2){
19            scanf("%d%d",&p,&c);
20            p-=d,c-=d;
21            ro.erase(p-1,c);
22            l[cnt++]=ro;
23        }
24        else{
25            scanf("%d%d%d",&v,&p,&c);
26            p-=d,v-=d,c-=d;
27            tmp=l[v].substr(p-1,c);
28            d+=count(tmp.begin(),tmp.end(),'c');
29            cout<<tmp<<endl;
30        }
31    }
32 }
33 #include<bits/extc++.h>
34 using namespace std;
35 using namespace __gnu_pbds;
36 __gnu_pbds::priority_queue<int> h1,h2;
37 typedef tree<int,null_type,less<int>,rb_tree_tag,
38     tree_order_statistics_node_update> set_t;
39 int main(){
40     printf("heap:\n");
41     for(int i=1;i<=10;i+=2)h1.push(i);
42     for(int i=2;i<=10;i+=2)h2.push(i);
43
44     printf("%d\n",h1.top());
45     printf("%d\n",h2.top());
46     h1.join(h2);
47     printf("%d\n",h1.size());
48     printf("%d\n",h2.size());
49     printf("%d\n",h1.top());

```

```

50
51     printf("\ntree:\n");
52     set_t s;
53     for(int i=0;i<5;i++)s.insert(10*i);
54     printf("%d\n",*s.find_by_order(0));
55     printf("%d\n",*s.find_by_order(3));
56     printf("%d\n",s.find_by_order(5)==s.end());
57
58     printf("%d\n",s.order_of_key(0));
59     printf("%d\n",s.order_of_key(30));
60     printf("%d\n",s.order_of_key(35));
61     printf("%d\n",s.order_of_key(100));
62     return 0;
63 }

```

2.7 Blossom

```

1 int V;
2 bool adj[MAXN][MAXN];
3 int w[MAXN][MAXN];
4 int p[MAXN];
5 int m[MAXN];
6 int d[MAXN];
7 int c1[MAXN], c2[MAXN];
8 int q[MAXN], *qf, *qb;
9 int pp[MAXN];
10 int f(int x){return x==pp[x]?x:(pp[x]=f(pp[x]));}
11 void u(int x, int y){pp[x]=y;}
12 int v[MAXN];
13 void path(int r, int x){
14     if(r==x) return;
15     if(d[x]==0){
16         path(r, p[p[x]]);
17         int i=p[x], j=p[p[x]];
18         m[i]=j; m[j]=i;
19     }
20     else if(d[x]==1){
21         path(m[x], c1[x]);
22         path(r, c2[x]);
23         int i=c1[x], j=c2[x];
24         m[i]=j; m[j]=i;
25     }
26 }
27 int lca(int x, int y, int r){
28     int i=f(x), j=f(y);
29     while(i!=j && v[i]!=2 && v[j]!=1){
30         v[i]=1; v[j]=2;
31         if(i!=r) i=f(p[i]);
32         if(j!=r) j=f(p[j]);
33     }
34     int b=i, z=j; if(v[j]==1) swap(b, z);
35     for(i=b; i!=z; i=f(p[i])) v[i]=-1;
36     v[z]=-1;
37     return b;

```

```

38 }
39 void contract_one_side(int x, int y, int b){
40     for (int i = f(x); i != b; i = f(p[i])){
41         u(i, b);
42         if (d[i] == 1) c1[i] = x, c2[i] = y, *qb++ = i;
43     }
44 }
45 bool BFS(int r){
46     for (int i=0; i<V; ++i) pp[i] = i;
47     memset(v, -1, sizeof(v));
48     memset(d, -1, sizeof(d));
49     d[r] = 0;
50     qf = qb = q;
51     *qb++ = r;
52     while (qf < qb)
53         for (int x=*qf++, y=0; y<V; ++y)
54             if (adj[x][y] && m[y] != y && f(x) != f(y))
55                 if (d[y] == -1)
56                     if (m[y] == -1){
57                         path(r, x);
58                         m[x] = y; m[y] = x;
59                         return true;
60                     }
61                     else{
62                         p[y] = x; p[m[y]] = y;
63                         d[y] = 1; d[m[y]] = 0;
64                         *qb++ = m[y];
65                     }
66                 else
67                     if (d[f(y)] == 0) {
68                         int b = lca(x, y, r);
69                         contract_one_side(x, y, b);
70                         contract_one_side(y, x, b);
71                     }
72     return false;
73 }
74 int match_result(){
75     int res=0;
76     memset(m, -1, sizeof(m));
77     for(int i=0; i<V; i++){
78         if(m[i]==-1){
79             if(BFS(i)) res++;
80             else m[i]=i;
81         }
82     }
83     return res;
84 }
85 int num[10000 + 10], top;
86 int main(){
87     int T, Case=0, n;
88     scanf("%d", &T);
89     while(T--){
90         scanf("%d", &n);
91         V=(1<n);
92         top=0;
93         for(int i=0; i<V; i++){

```

```

94             for(int j=i+1; j<V; j++){
95                 scanf("%d", &w[i][j]);
96                 num[top++] = w[i][j];
97             }
98         }
99         sort(num, num+top);
100         top = (unique(num, num+top) - num);
101         int l=0, r=top-1, mid;
102         while(r>l){
103             mid=(l+r+1)/2;
104             memset(adj, false, sizeof(adj));
105             for(int i=0; i<V; i++){
106                 for(int j=i+1; j<V; j++){
107                     if(w[i][j]>=num[mid]) adj[i][j]=adj[j][i]=true;
108                 }
109             }
110             int res=match_result();
111             if(res==V/2) l=mid;
112             else r=mid-1;
113         }
114         printf("Case %d: %d\n", ++Case, num[l]);
115     }
116 }

```

2.8 Dice

```

1 //source: chikOkU - Osaka University
2 enum DR{L,R,U,D,NONE};
3 int R_table[6][4]={
4     {2,3,5,4},
5     {3,1,4,6},
6     {2,6,5,1},
7     {1,5,6,2},
8     {1,3,6,4},
9     {4,5,3,2}
10 };
11 struct dice{
12     int t,f;
13     int getR(){
14         int id=find(R_table[t-1], R_table[t-1]+4, f)-R_table[t-1];
15         id=(id+1)%4;
16         return R_table[t-1][id];
17     }
18     DR getDir(int x){
19         if(x==t) return NONE;
20         else if(t+x==7) return NONE;
21         else if(f==x) return U;
22         else if(f+x==7) return D;
23         int r = getR();
24         if(x==r) return R;
25         else return L;
26     }
27     void rot(DR dr){
28         if(dr==L) t=getR();

```

```

29     else if(dr==R) t=7-getR();
30     else if(dr==U){
31         int nt=7-f;
32         f=t;
33         t=nt;
34     }
35     else{
36         int nf=7-t;
37         t=f;
38         f=nf;
39     }
40 }
41 }

```

2.9 Dinic

```

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

```

```

39                 e.cap-=d;
40                 g[e.to][e.rev].cap+=d;
41                 return d;
42             }
43         }
44     }
45     return 0;
46 }
47 int max_flow(int s,int t){
48     int flow=0;
49     for(;;){
50         bfs(s);
51         if(level[t]<0) return flow;
52         memset(iter,0,sizeof(iter));
53         int f;
54         while((f=dfs(s,t,1e9))>0)
55             flow+=f;
56     }
57 }

```

2.10 Dinic Flow

```

1 #define maxnode (200+10)
2 #define maxedge (400+10)
3 #define INF 1023456789
4 #include<bits/stdc++.h>
5 using namespace std;
6 int node, src, dest, nedge;
7 int head[maxnode], point[maxedge], nxt[maxedge], flow[maxedge], capa[maxedge];
8 int dist[maxnode], Q[maxnode], work[maxnode];
9 //set number of node, source, and destination (one base)
10 void init(int _node, int _src, int _dest) {
11     node = _node;
12     src = _src;
13     dest = _dest;
14     nedge = 0;
15     memset(point, -1, sizeof(point));
16     for (int i = 1; i <= node; i++) head[i] = -1;
17     nedge = 0;
18 }
19 void add_edge(int u, int v, int c1, int c2) {
20     point[nedge] = v, capa[nedge] = c1, flow[nedge] = 0, nxt[nedge] = head[u], head[u] = (nedge++);
21     point[nedge] = u, capa[nedge] = c2, flow[nedge] = 0, nxt[nedge] = head[v], head[v] = (nedge++);
22 }
23 bool dinic_bfs() {
24     memset(dist, 255, sizeof (dist));
25     dist[src] = 0;
26     int sizeQ = 0;
27     Q[sizeQ++] = src;
28     for (int cl = 0; cl < sizeQ; cl++)
29         for (int k = Q[cl], i = head[k]; i >= 0; i = nxt[i])

```

```

30         if (flow[i] < capa[i] && dist[point[i]] < 0) {
31             dist[point[i]] = dist[k] + 1;
32             Q[sizeQ++] = point[i];
33         }
34         return dist[dest] >= 0;
35     }
36     int dinic_dfs(int x, int exp) {
37         if (x == dest) return exp;
38         for (int &i = work[x]; i >= 0; i = nxt[i]) {
39             int v = point[i], tmp;
40             if (flow[i] < capa[i] && dist[v] == dist[x] + 1 && (tmp = dinic_dfs(v
, min(exp, capa[i] - flow[i]))) > 0) {
41                 flow[i] += tmp;
42                 flow[i^1] -= tmp;
43                 return tmp;
44             }
45         }
46         return 0;
47     }
48     int dinic_flow() {
49         int result = 0;
50         while (dinic_bfs()) {
51             for (int i = 0; i < node; i++) work[i] = head[i];
52             while (1) {
53                 int delta = dinic_dfs(src, INF);
54                 if (delta == 0) break;
55                 result += delta;
56             }
57         }
58         return result;
59     }
60     int main() {
61         int n,m,x,y,z;
62         while (scanf("%d%d", &n, &m) == 2) {
63             init(m, 1, m);
64             for (int i=0; i<n; i++) {
65                 scanf("%d%d%d", &x, &y, &z);
66                 add_edge(x, y, z, 0);
67             }
68             printf("%d\n", dinic_flow());
69         }
70         return 0;
71     }

```

2.11 Extgcd

```

1 long long extgcd(long long a, long long b, long long &x, long long &y) {
2     long long d=a;
3     if (b!=0) {
4         d=extgcd(b, a%b, y, x);
5         y--=(a/b)*x;
6     }
7     else x=1, y=0;
8     return d;

```

```

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

```

2.12 General Weighted Matching

```

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

```



```

37  int res = c - '0';
38  while (c = getchar(), '0' <= c && c <= '9')
39      res = res * 10 + c - '0';
40  return res;
41 }
42
43 const int MaxNX = MaxN + MaxN;
44
45 struct edge
46 {
47     int v, u, w;
48
49     edge(){}
50     edge(const int &_v, const int &_u, const int &_w)
51         : v(_v), u(_u), w(_w){}
52 };
53
54 int n, m;
55 edge mat[MaxNX + 1][MaxNX + 1];
56
57 int n_matches;
58 s64 tot_weight;
59 int mate[MaxNX + 1];
60 int lab[MaxNX + 1];
61
62 int q_n, q[MaxN];
63 int fa[MaxNX + 1], col[MaxNX + 1];
64 int slackv[MaxNX + 1];
65
66 int n_x;
67 int bel[MaxNX + 1], blofrom[MaxNX + 1][MaxN + 1];
68 vector<int> bloch[MaxNX + 1];
69
70 inline int e_delta(const edge &e) // does not work inside blossoms
71 {
72     return lab[e.v] + lab[e.u] - mat[e.v][e.u].w * 2;
73 }
74 inline void update_slackv(int v, int x)
75 {
76     if (!slackv[x] || e_delta(mat[v][x]) < e_delta(mat[slackv[x]][x]))
77         slackv[x] = v;
78 }
79 inline void calc_slackv(int x)
80 {
81     slackv[x] = 0;
82     for (int v = 1; v <= n; v++)
83         if (mat[v][x].w > 0 && bel[v] != x && col[bel[v]] == 0)
84             update_slackv(v, x);
85 }
86
87 inline void q_push(int x)
88 {
89     if (x <= n)
90         q[q_n++] = x;
91     else
92 {

```

```

93     for (int i = 0; i < size(bloch[x]); i++)
94         q_push(bloch[x][i]);
95     }
96 }
97 inline void set_mate(int xv, int xu)
98 {
99     mate[xv] = mat[xv][xu].u;
100    if (xv > n)
101    {
102        edge e = mat[xv][xu];
103        int xr = blofrom[xv][e.v];
104        int pr = find(bloch[xv].begin(), bloch[xv].end(), xr) - bloch[xv].begin();
105        if (pr % 2 == 1)
106        {
107            reverse(bloch[xv].begin() + 1, bloch[xv].end());
108            pr = size(bloch[xv]) - pr;
109        }
110
111        for (int i = 0; i < pr; i++)
112            set_mate(bloch[xv][i], bloch[xv][i ^ 1]);
113        set_mate(xr, xu);
114
115        rotate(bloch[xv].begin(), bloch[xv].begin() + pr, bloch[xv].end());
116    }
117 }
118 inline void set_bel(int x, int b)
119 {
120     bel[x] = b;
121     if (x > n)
122     {
123         for (int i = 0; i < size(bloch[x]); i++)
124             set_bel(bloch[x][i], b);
125     }
126 }
127
128 inline void augment(int xv, int xu)
129 {
130     while (true)
131     {
132         int xnu = bel[mate[xv]];
133         set_mate(xv, xu);
134         if (!xnu)
135             return;
136         set_mate(xnu, bel[fa[xnu]]);
137         xv = bel[fa[xnu]], xu = xnu;
138     }
139 }
140 inline int get_lca(int xv, int xu)
141 {
142     static bool book[MaxNX + 1];
143     for (int x = 1; x <= n_x; x++)
144         book[x] = false;
145     while (xv || xu)
146     {
147         if (xv)

```

```

148     {
149         if (book[xv])
150             return xv;
151         book[xv] = true;
152         xv = bel[mate[xv]];
153         if (xv)
154             xv = bel[fa[xv]];
155     }
156     swap(xv, xu);
157 }
158 return 0;
159 }
160
161 inline void add_blossom(int xv, int xa, int xu)
162 {
163     int b = n + 1;
164     while (b <= n_x && bel[b])
165         b++;
166     if (b > n_x)
167         n_x++;
168
169     lab[b] = 0;
170     col[b] = 0;
171
172     mate[b] = mate[xa];
173
174     bloch[b].clear();
175     bloch[b].push_back(xa);
176     for (int x = xv; x != xa; x = bel[fa[bel[mate[x]]]])
177         bloch[b].push_back(x), bloch[b].push_back(bel[mate[x]]), q_push(bel[mate[
178             x]]);
179     reverse(bloch[b].begin() + 1, bloch[b].end());
180     for (int x = xu; x != xa; x = bel[fa[bel[mate[x]]]])
181         bloch[b].push_back(x), bloch[b].push_back(bel[mate[x]]), q_push(bel[mate[
182             x]]);
183
184     set_bel(b, b);
185
186     for (int x = 1; x <= n_x; x++)
187     {
188         mat[b][x].w = mat[x][b].w = 0;
189         blofrom[b][x] = 0;
190     }
191     for (int i = 0; i < size(bloch[b]); i++)
192     {
193         int xs = bloch[b][i];
194         for (int x = 1; x <= n_x; x++)
195             if (mat[b][x].w == 0 || e_delta(mat[xs][x]) < e_delta(mat[b][x]))
196                 mat[b][x] = mat[xs][x], mat[x][b] = mat[x][xs];
197         for (int x = 1; x <= n_x; x++)
198             if (blofrom[xs][x])
199                 blofrom[b][x] = xs;
200     }
201     calc_slackv(b);
202 }
203 inline void expand_blossom1(int b) // lab[b] == 1
204 {
205     for (int i = 0; i < size(bloch[b]); i++)
206         set_bel(bloch[b][i], bloch[b][i]);
207
208     int xr = blofrom[b][mat[b][fa[b]].v];
209     int pr = find(bloch[b].begin(), bloch[b].end(), xr) - bloch[b].begin();
210     if (pr % 2 == 1)
211     {
212         reverse(bloch[b].begin() + 1, bloch[b].end());
213         pr = size(bloch[b]) - pr;
214     }
215     for (int i = 0; i < pr; i += 2)
216     {
217         int xs = bloch[b][i], xns = bloch[b][i + 1];
218         fa[xs] = mat[xns][xs].v;
219         col[xs] = 1, col[xns] = 0;
220         slackv[xs] = 0, calc_slackv(xns);
221         q_push(xns);
222     }
223     col[xr] = 1;
224     fa[xr] = fa[b];
225     for (int i = pr + 1; i < size(bloch[b]); i++)
226     {
227         int xs = bloch[b][i];
228         col[xs] = -1;
229         calc_slackv(xs);
230     }
231     bel[b] = 0;
232 }
233 inline void expand_blossom_final(int b) // at the final stage
234 {
235     for (int i = 0; i < size(bloch[b]); i++)
236     {
237         if (bloch[b][i] > n && lab[bloch[b][i]] == 0)
238             expand_blossom_final(bloch[b][i]);
239         else
240             set_bel(bloch[b][i], bloch[b][i]);
241     }
242     bel[b] = 0;
243 }
244
245 inline bool on_found_edge(const edge &e)
246 {
247     int xv = bel[e.v], xu = bel[e.u];
248     if (col[xu] == -1)
249     {
250         int nv = bel[mate[xu]];
251         fa[xu] = e.v;
252         col[xu] = 1, col[nv] = 0;
253         slackv[xu] = slackv[nv] = 0;
254         q_push(nv);
255     }
256     else if (col[xu] == 0)
257     {

```

```

258     int xa = get_lca(xv, xu);
259     if (!xa)
260     {
261         augment(xv, xu), augment(xu, xv);
262         for (int b = n + 1; b <= n_x; b++)
263             if (bel[b] == b && lab[b] == 0)
264                 expand_blossom_final(b);
265         return true;
266     }
267     else
268         add_blossom(xv, xa, xu);
269 }
270 return false;
271 }
272
273 bool match()
274 {
275     for (int x = 1; x <= n_x; x++)
276         col[x] = -1, slackv[x] = 0;
277
278     q_n = 0;
279     for (int x = 1; x <= n_x; x++)
280         if (bel[x] == x && !mate[x])
281             fa[x] = 0, col[x] = 0, slackv[x] = 0, q_push(x);
282     if (q_n == 0)
283         return false;
284
285     while (true)
286     {
287         for (int i = 0; i < q_n; i++)
288         {
289             int v = q[i];
290             for (int u = 1; u <= n; u++)
291                 if (mat[v][u].w > 0 && bel[v] != bel[u])
292                 {
293                     int d = e_delta(mat[v][u]);
294                     if (d == 0)
295                     {
296                         if (on_found_edge(mat[v][u]))
297                             return true;
298                     }
299                     else if (col[bel[u]] == -1 || col[bel[u]] == 0)
300                         update_slackv(v, bel[u]);
301                 }
302             }
303
304             int d = INF;
305             for (int v = 1; v <= n; v++)
306                 if (col[bel[v]] == 0)
307                     tension(d, lab[v]);
308             for (int b = n + 1; b <= n_x; b++)
309                 if (bel[b] == b && col[b] == 1)
310                     tension(d, lab[b] / 2);
311             for (int x = 1; x <= n_x; x++)
312                 if (bel[x] == x && slackv[x])
313                     {

```

```

314                     if (col[x] == -1)
315                         tension(d, e_delta(mat[slackv[x]][x]));
316                     else if (col[x] == 0)
317                         tension(d, e_delta(mat[slackv[x]][x]) / 2);
318                 }
319
320             for (int v = 1; v <= n; v++)
321             {
322                 if (col[bel[v]] == 0)
323                     lab[v] -= d;
324                 else if (col[bel[v]] == 1)
325                     lab[v] += d;
326             }
327             for (int b = n + 1; b <= n_x; b++)
328                 if (bel[b] == b)
329                 {
330                     if (col[bel[b]] == 0)
331                         lab[b] += d * 2;
332                     else if (col[bel[b]] == 1)
333                         lab[b] -= d * 2;
334                 }
335
336             q_n = 0;
337             for (int v = 1; v <= n; v++)
338                 if (lab[v] == 0) // all unmatched vertices' labels are zero! cheers!
339                     return false;
340             for (int x = 1; x <= n_x; x++)
341                 if (bel[x] == x && slackv[x] && bel[slackv[x]] != x && e_delta(mat[
342                     slackv[x]][x]) == 0)
343                 {
344                     if (on_found_edge(mat[slackv[x]][x]))
345                         return true;
346                 }
347             for (int b = n + 1; b <= n_x; b++)
348                 if (bel[b] == b && col[b] == 1 && lab[b] == 0)
349                     expand_blossom1(b);
350             }
351         return false;
352     }
353
354 void calc_max_weight_match()
355 {
356     for (int v = 1; v <= n; v++)
357         mate[v] = 0;
358
359     n_x = n;
360     n_matches = 0;
361     tot_weight = 0;
362
363     bel[0] = 0;
364     for (int v = 1; v <= n; v++)
365         bel[v] = v, bloch[v].clear();
366     for (int v = 1; v <= n; v++)
367         for (int u = 1; u <= n; u++)
368             blofrom[v][u] = v == u ? v : 0;

```

```

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

```

2.13 Geometry

```

1 const double eps = 1e-10;
2 const double INF = 1.0/0.0;
3 const double SIDE = 10000;
4 const double PI = acos(-1.0);
5 const int MAXN = 500000 + 10;
6 struct PT{
7     double x,y;
8     PT(){}
9     PT(double x,double y):x(x),y(y){}
10    PT operator + (const PT& p)const{
11        return PT(x+p.x,y+p.y);
12    }
13    PT operator - (const PT& p)const{

```

```

14        return PT(x-p.x,y-p.y);
15    }
16    PT operator * (double c)const{
17        return PT(x*c,y*c);
18    }
19    PT operator / (double c)const{
20        return PT(x/c,y/c);
21    }
22    PT rot(double a)const{return PT(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a));}
23    double operator *(const PT& p)const{
24        return x*p.x+y*p.y;
25    }
26    double operator ^(const PT& p)const{
27        return x*p.y-y*p.x;
28    }
29    bool operator ==(const PT& p)const{
30        return fabs(x-p.x)<eps&&fabs(y-p.y)<eps;
31    }
32    double len2()const{return x*x+y*y;}
33    double len()const{return sqrt(len2());}
34 }poi[MAXN],stk[MAXN];
35 struct LINE{
36     PT a,b;
37     double angle;
38     LINE(){}
39     LINE(PT _a,PT _b):a(_a),b(_b),angle(atan2(_b.y-_a.y,_b.x-_a.x)){}
40 }line[MAXN],deq[MAXN];
41 int top;
42 inline int ori(const PT& p1,const PT& p2,const PT& p3){
43     double a=(p2-p1)^(p3-p1);
44     if(a>-eps&&a<eps)return 0;
45     return a>0 ? 1:-1;
46 }
47 inline bool btw(const PT& p1,const PT& p2,const PT& p3){
48     return (p2-p1)*(p3-p1)<eps;
49 }
50 //segment intersection
51 inline bool intersection(const PT& p1,const PT& p2,const PT& p3,const PT& p4)
52 {
53     int a123=ori(p1,p2,p3);
54     int a124=ori(p1,p2,p4);
55     int a341=ori(p3,p4,p1);
56     int a342=ori(p3,p4,p2);
57     if(a123==0&&a124==0)return btw(p1,p3,p4)||btw(p2,p3,p4)||btw(p3,p1,p2)||
58         btw(p4,p1,p2);
59     return a123*a124 <= 0 && a341*a342 <= 0;
60 }
61 inline PT intersectionPoint(const PT& p1,const PT& p2,const PT& p3,const PT&
62     p4){
63     double a123=(p2-p1)^(p3-p1);
64     double a124=(p2-p1)^(p4-p1);
65     return (p4*a123-p3*a124)/(a123-a124);
66 }
67 //line intersection
68 inline PT intersectionPoint(const LINE& l1,const LINE& l2){
69     PT p1=l1.a,p2=l1.b,p3=l2.a,p4=l2.b;

```

```

67     double a123=(p2-p1)^(p3-p1);
68     double a124=(p2-p1)^(p4-p1);
69     return (p4*a123-p3*a124)/(a123-a124);
70 }
71 PT foot(const LINE& l,const PT& p){
72     PT m(l.b.y-l.a.y,l.a.x-l.b.x);
73     return p+m*(l.a-p ^ l.b-p)/((l.b-l.a).len2());
74 }
75 PT mirror(const LINE& l,const PT& p){
76     PT m(l.b.y-l.a.y,l.a.x-l.b.x);
77     return p+m*(l.a-p ^ l.b-p)/((l.b-l.a).len2())*2;
78 }
79 //segment-point distance
80 inline double sp_dis(PT a,PT l1,PT l2){
81     if((a-l1)*(l2-l1)<0) return (l1-a).len();
82     else if((a-l2)*(l1-l2)<0) return (l2-a).len();
83     return fabs(l1-a^l2-a)/((l2-l1).len());
84 }
85
86 struct cir{
87     point c;
88     double r;
89 }o[10];
90 double out_ang(cir a,cir b){ //a.c+(b.c-a.c).unit().rot(ang)*b.r
91     return acos((a.r-b.r)/(a.c-b.c).len());
92 }
93 double in_ang(cir a,cir b){
94     return acos((a.r+b.r)/(a.c-b.c).len());
95 }
96 int main(){
97     double tmp,sum;
98     if(fabs(o[i].r-o[j].r)<(o[j].c-o[i].c).len()){
99         tmp = out_ang(o[i],o[j]);
100        sum = ang_add(c1,tmp);
101        pi=o[i].c+point(o[i].r*cos(sum),o[i].r*sin(sum));
102        pj=o[j].c+point(o[j].r*cos(sum),o[j].r*sin(sum));
103        sum = ang_add(c1,-tmp);
104        pi=o[i].c+point(o[i].r*cos(sum),o[i].r*sin(sum));
105        pj=o[j].c+point(o[j].r*cos(sum),o[j].r*sin(sum));
106    }
107    if(o[i].r+o[j].r<(o[j].c-o[i].c).len()){
108        tmp = in_ang(o[i],o[j]);
109        sum = ang_add(c1,tmp);
110        pi=o[i].c+point(o[i].r*cos(sum),o[i].r*sin(sum));
111        pj=o[j].c-point(o[j].r*cos(sum),o[j].r*sin(sum));
112        sum = ang_add(c1,-tmp);
113        pi=o[i].c+point(o[i].r*cos(sum),o[i].r*sin(sum));
114        pj=o[j].c-point(o[j].r*cos(sum),o[j].r*sin(sum));
115    }
116 }
117
118 inline double dist(const PT& p1,const PT& p2){
119     return sqrt((p2-p1)*(p2-p1));
120 }
121 inline double tri(const PT& p1,const PT& p2,const PT& p3){
122     return fabs((p2-p1)^(p3-p1));

```

```

123 }
124 inline double getPerimeter(){
125     double res=0.0;
126     poi[top++]=poi[0];
127     for(int i=0;i<top-1;i++)res+=dist(poi[i],poi[i+1]);
128     return res;
129 }
130 inline double getarea(){
131     double res=0.0;
132     for(int i=1;i<top-1;i++)res+=tri(poi[0],poi[i],poi[i+1]);
133     return 0.5*res;
134 }
135
136 //convex hull
137 inline bool cmp_convex(const PT &a,const PT &b){
138     if(a.x!=b.x)return a.x<b.x;
139     return a.y<b.y;
140 }
141 inline void convex_hull(PT a[],int &n){
142     top=0;
143     sort(a,a+n,cmp_convex);
144     for(int i=0;i<n;i++){
145         while(top>=2&&ori(stk[top-2],stk[top-1],a[i])>=0)top--;
146         stk[top++]=a[i];
147     }
148     for(int i=n-2,t=top+1;i>=0; i--){
149         while(top>=t&&ori(stk[top-2],stk[top-1],a[i])>=0)top--;
150         stk[top++]=a[i];
151     }
152     top--;
153     for(int i=0;i<top;i++)poi[i]=stk[i];
154 }
155 //half plane intersection
156 inline bool cmp_half_plane(const LINE &a,const LINE &b){
157     if(fabs(a.angle-b.angle)<eps)return ori(a.a,a.b,b.a)<0;
158     return a.angle > b.angle;
159 }
160 inline void half_plane_intersection(LINE a[],int &n){
161     int m=1,front=0,rear=1;
162     sort(a,a+n,cmp_half_plane);
163     for(int i=1;i<n;i++){
164         if(fabs(a[i].angle-a[m-1].angle)>eps)a[m++]=a[i];
165     }
166     deq[0]=a[0],deq[1]=a[1];
167     for(int i=2;i<m;i++){
168         while(front<rear&&ori(a[i].a,a[i].b,intersectionPoint(deq[rear],deq[rear-1]))<0)rear--;
169         while(front<rear&&ori(a[i].a,a[i].b,intersectionPoint(deq[front],deq[front+1]))<0)front++;
170         deq[++rear]=a[i];
171     }
172     while(front<rear&&ori(deq[front].a,deq[front].b,intersectionPoint(deq[rear],deq[rear-1]))<0)rear--;
173     while(front<rear&&ori(deq[rear].a,deq[rear].b,intersectionPoint(deq[front],deq[front+1]))<0)front++;
174     if(front==rear)return;

```

```

175
176     top=0;
177     for(int i=front;i<rear;i++) poi[top++]=intersectionPoint(deq[i],deq[i+1]);
178     if(rear>front+1) poi[top++]=intersectionPoint(deq[front],deq[rear]);
179 }
180
181
182
183 //smallest cover rectangle
184 double ans1,ans2;
185 void rotating_calipers() {
186     ans1=ans2=INF;
187     int j=1,k=1,l=1;
188     poi[top]=poi[0];
189     for(int i=0;i<top;i++) {
190         while(tri(poi[i],poi[i+1],poi[j])<tri(poi[i],poi[i+1],poi[j+1])) j=(j
+1)%top;
191         while((poi[i+1]-poi[i])*(poi[k+1]-poi[k]))>eps) k=(k+1)%top;
192         if(i==0) l=(k+1)%top;
193         while((poi[i+1]-poi[i])*(poi[l+1]-poi[l]))<-eps) l=(l+1)%top;
194         double tmp1 = tri(poi[i],poi[i+1],poi[j])/dist(poi[i],poi[i+1]);
195         double tmp2 = (((poi[k]-poi[i])*(poi[i+1]-poi[i]))-((poi[l]-poi[i])*(
poi[i+1]-poi[i]))) / dist(poi[i],poi[i+1]);
196         if((tmp1+tmp2)*2.0<ans1) ans1=(tmp1+tmp2)*2.0;
197         if(tmp1*tmp2<ans2) ans2=tmp1*tmp2;
198     }
199 }
200 }
201 int main() {
202     int n,m;
203     while(~scanf("%d",&n)&&n) {
204         for(int i=0;i<n;i++) scanf("%lf%lf",&poi[i].x,&poi[i].y);
205         convex_hull(poi,n);
206         rotating_calipers();
207         printf("%.2f %.2f\n",ans2,ans1);
208     }
209 }
210
211 inline bool online(const LINE &L,const PT &p){
212     return ori(p,L.a,L.b)==0&&btw(p,L.a,L.b);
213 }
214 inline bool on_convex(const PT& p){
215     for(int i=0;i<top;i++)
216         if(p==poi[i]) return 1;
217     poi[top]=poi[0];
218     for(int i=0;i<top;i++) {
219         line[i].a=poi[i];
220         line[i].b=poi[i+1];
221     }
222     for(int i=0;i<top;i++)
223         if(online(line[i],p)) return 1;
224     return 0;
225 }
226 //originally in long long, should be modified
227 bool in_simple_polygon(PT b[],int k){
228     bool flag=false;

```

```

229     for(int j=0;j<k;j++) {
230         if(((p-b[j])^(p-b[(j+1)%k]))==0&&(p-b[j])*(p-b[(j+1)%k])<=0) {
231             flag=true;
232             break;
233         }
234         if((b[j].y<p.y)^(b[(j+1)%k].y<p.y)) {
235             long long xss=(b[j]-p)^(b[(j+1)%k]-p);
236             if((xss<0)^(b[j].y<b[(j+1)%k].y)) {
237                 flag^=1;
238             }
239         }
240     }
241     return flag;
242 }

```

2.14 Heavy Light Decomposition

```

1 //with set value && query sum, 1-based with n points
2 //remove vis in DFS, add it back if something weird happen(I don't think it
   's required)
3 using namespace std;
4 int sz[N],top[N],up[N],dep[N];
5 int lightval[N]; //value on light edge
6 struct node{
7     node(){}
8     node(int _l,int _r):val(1),l(_l),r(_r),lc(NULL),rc(NULL){}
9     int l,r;
10    node *lc,*rc;
11    int sum;
12    int val;
13    int qsum() {return val>=0?val*(r-l):sum;}
14    void push() {
15        if(val>=0) {
16            sum=val*(r-l);
17            lc->val=rc->val=val;
18            val=-1;
19        }
20    }
21    void pull() {
22        sum=lc->qsum()+rc->qsum();
23    }
24 };
25 node* tr[N];
26 node* build(int l,int r){
27     node *now=new node(l,r);
28     if(r-l>1) {
29         now->lc=build(l,(l+r)/2);
30         now->rc=build((l+r)/2,r);
31     }
32     return now;
33 }
34 //partial
35 int qry(node* now,int l,int r){
36     if(l>=r) return 0;

```

```

37  if(l==now->l&&r==now->r){
38      return now->qsum();
39  }
40  int m=(now->l+now->r)/2;
41  now->push();
42  if(l>=m){
43      return qry(now->rc,l,r);
44  }
45  else if(r<=m){
46      return qry(now->lc,l,r);
47  }
48  else return qry(now->lc,l,m)+qry(now->rc,m,r);
49 }
50 void set0(node *now,int l,int r){
51     if(l>=r) return;
52     if(l==now->l&&r==now->r){
53         now->val=0;
54         return;
55     }
56     int m=(now->l+now->r)/2;
57     now->push();
58     if(l>=m){
59         set0(now->rc,l,r);
60     }
61     else if(r<=m){
62         set0(now->lc,l,r);
63     }
64     else{
65         set0(now->lc,l,m);
66         set0(now->rc,m,r);
67     }
68     now->pull();
69 }
70 vector<int> g[N];
71 void DFS(int u,int p,int d){
72     dep[u]=d;
73     sz[u]=1;
74     for(int i=0;i<g[u].size();i++){
75         int v=g[u][i];
76         if(v==p) continue;
77         DFS(v,u,d+1);
78         sz[u]+=sz[v];
79     }
80 }
81 void decomp(int u,int p,bool istop){
82     bool ed=true;
83     if(istop) top[u]=u,up[u]=p,lightval[u]=1;
84     else top[u]=top[p],up[u]=up[p];
85     for(int i=0;i<g[u].size();i++){
86         int v=g[u][i];
87         if(v==p) continue;
88         if(sz[v]>=sz[u]-sz[v]){
89             decomp(v,u,false);
90             ed=false;
91         }
92         else decomp(v,u,true);

```

```

93     }
94     if(ed){
95         tr[top[u]]=build(dep[top[u]],dep[u]);
96     }
97 }
98 //global
99 int qry(int u,int v){
100     int res=0;
101     while(top[u]!=top[v]){
102         if(dep[top[u]]>dep[top[v]]) swap(u,v);
103         res+=qry(tr[top[v]],dep[top[v]],dep[v]);
104         res+=lightval[top[v]];
105         v=up[top[v]];
106     }
107     if(dep[u]>dep[v]) swap(u,v);
108     res+=qry(tr[top[v]],dep[u],dep[v]);
109     return res;
110 }
111 void set0(int u,int v){
112     while(top[u]!=top[v]){
113         if(dep[top[u]]>dep[top[v]]) swap(u,v);
114         set0(tr[top[v]],dep[top[v]],dep[v]);
115         lightval[top[v]]=0;
116         v=up[top[v]];
117     }
118     if(dep[u]>dep[v]) swap(u,v);
119     set0(tr[top[v]],dep[u],dep[v]);
120 }
121 int main(){
122     DFS(1,0,0);
123     decomp(1,0,true);
124 }

```

2.15 Huafen

```

1  const int MAXN = 100000 + 10;
2  int tree[30][MAXN]={},sorted[MAXN]={},toleft[30][MAXN]={};
3  void build(int l,int r,int dep){
4      if(l==r) return;
5      int mid=(l+r)>>1;
6      int same=mid-l+1;
7      for(int i=l;i<=r;i++) if(tree[dep][i]<sorted[mid]) same--;
8      int lpos=l,rpos=mid+1;
9      for(int i=l;i<=r;i++){
10         if(tree[dep][i]<sorted[mid]) tree[dep+1][lpos++]=tree[dep][i];
11         else if(tree[dep][i]==sorted[mid]&&same>0) tree[dep+1][lpos++]=tree[
12             dep][i],same--;
13         else tree[dep+1][rpos++]=tree[dep][i];
14         toleft[dep][i]=toleft[dep][l-l+1]+lpos-1;
15     }
16     build(l,mid,dep+1);
17     build(mid+1,r,dep+1);
18 }
19 int query(int L,int R,int l,int r,int dep,int k){

```

```

19     if(l==r) return tree[dep][l];
20     int mid=(L+R)>>1;
21     int cnt=toleft[dep][r]-toleft[dep][l-1];
22     if(cnt>=k){
23         int newl=L+toleft[dep][l-1]-toleft[dep][L-1];
24         int newr=newl+cnt-1;
25         return query(L,mid,newl,newr,dep+1,k);
26     }
27     else{
28         int newr=r+toleft[dep][R]-toleft[dep][r];
29         int newl=newr-(r-l-cnt);
30         return query(mid+1,R,newl,newr,dep+1,k-cnt);
31     }
32 }
33 int main(){
34     int n,m,a,b,c;
35     while(~scanf("%d%d", &n, &m)){
36         for(int i=1;i<=n;i++){
37             scanf("%d",&tree[0][i]);
38             sorted[i]=tree[0][i];
39         }
40         sort(sorted+1,sorted+n+1);
41         build(1,n,0);
42         while(m--){
43             scanf("%d%d%d",&a,&b,&c);
44             printf("%d\n",query(1,n,a,b,0,c));
45         }
46     }
47     return 0;
48 }

```

2.16 KDtree Insert

```

1 #include<algorithm>
2 #include<cmath>
3 #include<cstdio>
4 #include<queue>
5 #include<cstdlib>
6 #include<vector>
7 #define MAXN 50100
8 using namespace std;
9 inline long long sq(long long x){return x*x;}
10 const double alpha=0.75;
11 int W,H,rx[MAXN],ry[MAXN];
12 namespace KDTree{
13     struct Point {
14         int x,y;
15         int index;
16         long long distance(const Point &b)const{
17             return sq(x-b.x) + sq(y-b.y);
18         }
19         bool operator==(const Point& rhs){return index==rhs.index;}
20     };
21     struct qnode{

```

```

22     Point p;
23     long long dis;
24     qnode(){}
25     qnode(Point _p,long long _dis){
26         p = _p;
27         dis = _dis;
28     }
29     bool operator <(const qnode &b)const{
30         if(dis != b.dis) return dis < b.dis;
31         else return p.index < b.p.index;
32     }
33 };
34 priority_queue<qnode>q;
35 inline bool cmpX(const Point &a,const Point &b){
36     return a.x < b.x || (a.x == b.x && a.y < b.y) || (a.x == b.x && a.y == b.y && a.index < b.index);
37 }
38 inline bool cmpY(const Point &a,const Point &b){
39     return a.y < b.y || (a.y == b.y && a.x < b.x) || (a.y == b.y && a.x == b.x && a.index < b.index);
40 }
41 bool cmp(const Point &a,const Point &b,bool div){
42     return div?cmpY(a,b):cmpX(a,b);
43 }
44 struct Node{
45     Point e;
46     Node *lc,*rc;
47     int size;
48     bool div;
49     inline void pull(){
50         size = 1 + lc->size + rc->size;
51     }
52     inline bool isBad(){
53         return lc->size > alpha*size || rc->size > alpha*size;
54     }
55 }pool[MAXN],*tail,*root,*recycle[MAXN],*null;
56 int rc_cnt;
57 void init(){
58     tail = pool;
59     null = tail++;
60     null->lc = null->rc = null;
61     null->size = 0;
62     rc_cnt = 0;
63     root = null;
64 }
65 Node *newNode(Point e){
66     Node *p;
67     if(rc_cnt)p = recycle[--rc_cnt];
68     else p = tail++;
69     p->e = e;
70     p->lc = p->rc = null;
71     p->size = 1;
72     return p;
73 }
74 Node *build(Point *a,int l,int r,bool div){
75     if(l >= r) return null;

```



```

76     int mid = (l+r)/2;
77     nth_element(a+l,a+mid,a+r,div?cmpY:cmpX);
78     Node *p = newNode(a[mid]);
79     p->div = div;
80     p->lc = build(a,l,mid,!div);
81     p->rc = build(a,mid+1,r,!div);
82     p->pull();
83     return p;
84 }
85 void getTree(Node *p,vector<Point>& v){
86     if(p==null) return;
87     getTree(p->lc,v);
88     v.push_back(p->e);
89     recycle[rc_cnt++]=p;
90     getTree(p->rc,v);
91 }
92 Node *rebuild(vector<Point>& v,int l,int r,bool div){
93     if(l>=r) return null;
94     int mid = (l+r)/2;
95     nth_element(v.begin()+l,v.begin()+mid,v.begin()+r,div?cmpY:cmpX);
96     Node *p = newNode(v[mid]);
97     p->div = div;
98     p->lc = rebuild(v,l,mid,!div);
99     p->rc = rebuild(v,mid+1,r,!div);
100    p->pull();
101    return p;
102 }
103 void rebuild(Node *p){
104     vector<Point> v;
105     getTree(p,v);
106     p = rebuild(v,0,v.size(),p->div);
107 }
108 Node **insert(Node *p,Point a,bool div){
109     if(p==null){
110         p = newNode(a);
111         p->div = div;
112         return &null;
113     }
114     else{
115         Node **res;
116         if(cmp(a,p->e,div)) res=insert(p->lc,a,!div);
117         else res=insert(p->rc,a,!div);
118         p->pull();
119         if(p->isBad()) res=&p;
120         return res;
121     }
122 }
123 void insert(Point e){
124     Node **p = insert(root,e,0);
125     if(*p!=null) rebuild(*p);
126 }
127 Node **get_min(Node *p,bool div){
128     if(p->div==div){
129         if(p->lc!=null) return get_min(p->lc,div);
130         else return &p;
131     }

```

```

132     else{
133         Node **res=&p,**tmp;
134         if(p->lc!=null){
135             tmp = get_min(p->lc,div);
136             if(cmp((*tmp)->e,(*res)->e,div)) res=tmp;
137         }
138         if(p->rc!=null){
139             tmp = get_min(p->rc,div);
140             if(cmp((*tmp)->e,(*res)->e,div)) res=tmp;
141         }
142         return res;
143     }
144 }
145 void del(Node *p){
146     Node **nxt;
147     if(p->rc!=null){
148         nxt = get_min(p->rc,p->div);
149         p->e = (*nxt)->e;
150         del(*nxt);
151     }
152     else if(p->lc!=null){
153         nxt = get_min(p->lc,p->div);
154         p->e = (*nxt)->e;
155         del(*nxt);
156         p->rc = p->lc;
157         p->lc = null;
158     }
159     else{
160         recycle[rc_cnt++]=p;
161         p=null;
162     }
163 }
164 void del(Node *p,Point d){
165     if(p->e==d){
166         del(p);
167     }
168     else if(cmp(d,p->e,p->div)) del(p->lc,d);
169     else del(p->rc,d);
170 }
171 void search(Point p,Node *t,bool div,int m){
172     if(!t) return;
173     if(cmp(p,t->e,div)){
174         search(p,t->lc,!div,m);
175         if(q.size() < m){
176             q.push(qnode(t->e,p.distance(t->e)));
177             search(p,t->rc,!div,m);
178         }
179     }
180     else {
181         if(p.distance(t->e) <= q.top().dis){
182             q.push(qnode(t->e,p.distance(t->e)));
183             q.pop();
184         }
185         if(!div){
186             if(sq(t->e.x-p.x) <= q.top().dis)
187                 search(p,t->rc,!div,m);

```

```

188     else {
189         if(sq(t->e.y-p.y) <= q.top().dis)
190             search(p,t->rc,!div,m);
191     }
192 }
193 }
194 else {
195     search(p,t->rc,!div,m);
196     if(q.size() < m){
197         q.push(qnode(t->e,p,distance(t->e)));
198         search(p,t->lc,!div,m);
199     }
200     else {
201         if(p.distance(t->e) <= q.top().dis){
202             q.push(qnode(t->e,p,distance(t->e)));
203             q.pop();
204         }
205         if(!div){
206             if(sq(t->e.x-p.x) <= q.top().dis)
207                 search(p,t->lc,!div,m);
208         }
209         else {
210             if(sq(t->e.y-p.y) <= q.top().dis)
211                 search(p,t->rc,!div,m);
212         }
213     }
214 }
215 }
216 void search(Point p,int m){
217     while(!q.empty())q.pop();
218     search(p,root,0,m);
219 }
220 void getRange(Node *p,vector<Point>& v,int x1,int x2,int y1,int y2){
221     if(p==null) return;
222     if(x1<=p->e.x && p->e.x<=x2 && y1<=p->e.y && p->e.y<=y2) v.push_back(p->e);
223     if(p->div ? y1<=p->e.y : x1<=p->e.x) getRange(p->lc,v,x1,x2,y1,y2);
224     if(p->div ? y2>=p->e.y : x2>=p->e.x) getRange(p->rc,v,x1,x2,y1,y2);
225 }
226 void solve(Point p){
227     del(root,p);
228     insert(p);
229 }
230 };
231 KDTree::Point p[MAXN];
232 int main(){
233     KDTree::init();
234     KDTree::root = KDTree::build(p,0,n,0);
235     while(q--){
236         KDTree::Point tmp,p1,p2;
237         scanf("%d%d",&tmp.x,&tmp.y);
238         search(tmp,2);
239         p1=KDTree::q.top().p;
240         KDTree::q.pop();
241         p2=KDTree::q.top().p;
242         KDTree::q.pop();

```

```

243     }
244     return 0;
245 }

```

2.17 KM

```

1  const int MAXN = 210, inf = 200000000; //KM Algorithm
2  int cost[MAXN][MAXN];
3  int lx[MAXN], ly[MAXN], mat[MAXN], slack[MAXN];
4  bool sx[MAXN], sy[MAXN];
5  int N;
6
7  bool extend(int now)
8  {
9      sx[now] = true;
10     int temp;
11
12     for(int i = 0; i < N; i++)
13         if(!sy[i]) {
14             temp = -(cost[now][i]-lx[now]-ly[i]);
15             if(temp==0) {
16                 sy[i] = true;
17                 if(mat[i]==-1 || extend(mat[i])) {
18                     mat[i] = now;
19                     return true;
20                 }
21             }
22             else if(temp < slack[i])
23                 slack[i] = temp;
24         }
25     return false;
26 }
27
28 int KM() //finding the maximum value of perfect matching
29 {
30     int ret = 0;
31     memset(lx, 0, sizeof(lx));
32     memset(ly, 0, sizeof(ly));
33     memset(mat, -1, sizeof(mat));
34     //matching precalculation
35     for(int i = 0; i < N; i++)
36     {
37         lx[i] = -inf;
38         for(int j = 0; j < N; j++)
39             lx[i] = max(lx[i], cost[i][j]);
40     }
41     //KM
42     for(int i = 0; i < N; i++)
43     {
44         for(int j = 0; j < N; j++)
45             slack[j] = inf;
46
47         while(true)
48         {

```

```

49     memset(sx, false, sizeof(sx));
50     memset(sy, false, sizeof(sy));
51
52     if(extend(i)) break;
53     int themin = inf+1;
54
55     for(int j = 0; j < N; j++)
56         if(!sy[j] && slack[j] < themin)
57             themin = slack[j];
58
59     for(int j = 0; j < N; j++)
60     {
61         if(sx[j]) lx[j] -= themin;
62         if(sy[j]) ly[j] += themin;
63         else slack[j] -= themin;
64     }
65 }
66
67
68 for(int i = 0; i < N; i++)
69     ret += cost[mat[i]][i];
70 return ret;
71 }

```

2.18 KM N3

```

1 int X,Y ;
2 int adj[510][510],lx[510], ly[510],mx[510], my[510];
3 bool vx[510], vy[510];
4 bool DFS(int x){
5     vx[x] = true;
6     for (int y=0; y<Y; ++y)
7         if (!vy[y])
8             if (lx[x] + ly[y] == adj[x][y]){
9                 vy[y] = true;
10                if (my[y] == -1 || DFS(my[y])){
11                    mx[x] = y; my[y] = x;
12                    return true;
13                }
14            }
15     return false;
16 }
17 int Hungarian(){
18     memset(ly, 0, sizeof(ly));
19     for (int x=0; x<X; ++x)
20         for (int y=0; y<Y; ++y)
21             if (adj[x][y] != 1e9)
22                 lx[x] = max(lx[x], adj[x][y]);
23     memset(mx, -1, sizeof(mx));
24     memset(my, -1, sizeof(my));
25     for (int x=0; x<X; ++x)
26         while (true){
27             memset(vx, false, sizeof(vx));
28             memset(vy, false, sizeof(vy));

```

```

29         if (DFS(x)) break;
30         int d = 1e9;
31         for (int xx=0; xx<X; ++xx) if (vx[xx])
32             for (int y=0; y<Y; ++y) if (!vy[y])
33                 if (adj[xx][y] != 1e9)
34                     d = min(d, lx[xx] + ly[y] - adj[xx][y]);
35         if (d == 1e9) return -1e9;
36         for (int xx=0; xx<X; ++xx)
37             if (vx[xx])
38                 lx[xx] -= d;
39         for (int y=0; y<Y; ++y)
40             if (vy[y])
41                 ly[y] += d;
42     }
43     int weight = 0;
44     for (int x=0; x<X; ++x)
45         weight += adj[x][mx[x]];
46     return weight;
47 }
48 int main()
49 {
50     int ans;
51     while(~scanf("%d",&X)){
52         Y=X;
53         for(int q=0;q<X;++q)
54             for(int w=0;w<X;++w)
55                 scanf("%d",&adj[q][w]);
56         ans=Hungarian();
57         printf("%d",lx[0]);
58         for(int q=1;q<X;++q)
59             printf(" %d",lx[q]);
60         printf("\n%d",ly[0]);
61         for(int q=1;q<X;++q)
62             printf(" %d",ly[q]);
63         printf("\n%d\n",ans);
64     }
65     return 0;
66 }

```

2.19 KM No Big Int

```

1 #include<cstdio>
2 #include<utility>
3 #include<cstring>
4 #include<algorithm>
5 using namespace std;
6 const int MAXN=1010;
7 long long inf=1LL<<60;
8 long long cost[MAXN][MAXN];
9 long long lx[MAXN], ly[MAXN], slack[MAXN];
10 int mat[MAXN];
11 bool sx[MAXN], sy[MAXN];
12 bool cant[MAXN][MAXN];
13 int N;

```

```

14 bool extend(int now)
15 {
16     sx[now] = true;
17     long long temp;
18     for(int i = 0; i < N; i++){
19         if(!sy[i]){
20             temp = -(cost[now][i]-lx[now]-ly[i]);
21             if(temp==0){
22                 sy[i] = true;
23                 if(mat[i]==-1 || extend(mat[i])) {
24                     mat[i] = now;
25                     return true;
26                 }
27             }
28             else if(temp < slack[i])
29                 slack[i] = temp;
30         }
31     }
32     return false;
33 }
34
35 pair<long long,bool> KM() //finding the maximum value of perfect matching
36 {
37     long long ret = 0;
38     memset(mat, -1, sizeof(mat));
39     //matching precalculation
40     for(int i = 0; i < N; i++)
41     {
42         lx[i] = -inf;
43         for(int j = 0; j < N; j++)
44             lx[i] = max(lx[i], cost[i][j]);
45         ly[i] = 0;
46     }
47     //KM
48     for(int i = 0; i < N; i++)
49     {
50         for(int j = 0; j < N; j++)
51             slack[j] = inf;
52
53         while(true)
54         {
55             memset(sx, false, sizeof(sx));
56             memset(sy, false, sizeof(sy));
57
58             if(extend(i)) break;
59             long long themin = inf+1;
60
61             for(int j = 0; j < N; j++)
62                 if(!sy[j] && slack[j] < themin)
63                     themin = slack[j];
64
65             for(int j = 0; j < N; j++)
66             {
67                 if(sx[j]) lx[j] = lx[j] - themin;
68                 if(sy[j]) ly[j] = ly[j] + themin;
69                 else slack[j] = slack[j] - themin;

```

```

70         }
71     }
72 }
73
74 for(int i = 0; i < N; i++){
75     if(cant[mat[i]][i]) return make_pair(0LL, false);
76     ret = ret + cost[mat[i]][i];
77 }
78 return make_pair(ret, true);
79 }
80 int main(){
81     int T;
82     scanf("%d", &T);
83     while(T--){
84         memset(cant, 0, sizeof(cant));
85         int k, x, y;
86         long long L, U;
87         long long xv[MAXN], yv[MAXN];
88         scanf("%d%lld%lld%d", &N, &L, &U, &k);
89         U-=L;
90         while(k--){
91             scanf("%d%d", &x, &y);
92             cant[x-1][y-1]=true;
93         }
94         for(int i=0;i<N;i++) scanf("%lld",&xv[i]);
95         for(int i=0;i<N;i++) scanf("%lld",&yv[i]);
96         for(int i=0;i<N;i++){
97             for(int j=0;j<N;j++){
98                 if(cant[i][j]) cost[i][j]=-inf;
99                 else cost[i][j]=-min(max(0LL,xv[i]+yv[j]-L),U);
100             }
101         }
102         pair<long long,bool> ans=KM();
103         if(ans.second) printf("%lld\n",-ans.first);
104         else puts("no");
105     }
106     return 0;
107 }

```

2.20 Maximum Density

```

1 /*
2 solve a problem that find a continuous cells with Maximum Density
3 whose length is at least F
4 */
5 const int maxN = 100001;
6 long long sum[maxN];
7 int main()
8 {
9     int N, F, ans;
10    scanf("%d%d", &N, &F);
11
12    for(int i = 1; i <= N; i++)
13    {

```

```

14     int temp;
15     scanf("%d", &temp);
16     sum[i] = sum[i-1]+temp*1000;
17 }
18
19 int front = 1, rear = F;
20 ans = sum[F]/F;
21
22 while( rear < N )
23 {
24     rear++;
25     int density = (sum[rear]-sum[front-1])/(rear-front+1),
26     f = rear-F+1 , nd = (sum[rear]-sum[f-1])/(rear-f+1);
27
28     if(nd >= density) front = f, density = nd;;
29     ans = ans > density ? ans : density;
30 }
31 printf("%d\n", ans);
32 return 0;
33 }

```

2.21 Min Cost Flow

```

1 #define maxnode (1000+10)
2 #define maxedge (40000+10)
3 #define INF 1023456789
4 #include<bits/stdc++.h>
5 using namespace std;
6 int node, src, dest, nedge;
7 int head[maxnode], point[maxedge], nxt[maxedge], flow[maxedge], capa[maxedge],
  wt[maxedge];
8 int dist[maxnode], in[maxnode], from[maxnode], mf[maxnode];
9 //set number of node, source, and destination (one base)
10 void init(int _node, int _src, int _dest) {
11     node = _node;
12     src = _src;
13     dest = _dest;
14     nedge = 0;
15     memset(point, -1, sizeof(point));
16     for (int i = 1; i <= node; i++) head[i] = -1;
17     nedge = 0;
18 }
19 void add_edge(int u, int v, int c1, int w) {
20     point[nedge] = v, capa[nedge] = c1, flow[nedge] = 0, nxt[nedge] = head[u],
  wt[nedge]=w, head[u] = (nedge++);
21     point[nedge] = u, capa[nedge] = 0, flow[nedge] = 0, nxt[nedge] = head[v],
  wt[nedge]=-w, head[v] = (nedge++);
22 }
23 int sp(int &left){
24     for(int i=1;i<=node;i++) dist[i]=INF;
25     queue<int> que;
26     que.push(src);
27     in[src]=1;
28     mf[src]=left;

```

```

29     dist[src]=0;
30     while(!que.empty()){
31         int u=que.front();
32         que.pop();
33         in[u]=0;
34         if(dist[u]>=dist[dest]) continue;
35         for(int v=head[u];v!=-1;v=nxt[v]){
36             if(flow[v]==capa[v]) continue;
37             if(dist[u]+wt[v]<dist[point[v]]){
38                 dist[point[v]]=dist[u]+wt[v];
39                 from[point[v]]=v;
40                 mf[point[v]]=min(mf[u],capa[v]-flow[v]);
41                 if(!in[point[v]]){
42                     in[point[v]]=1;
43                     que.push(point[v]);
44                 }
45             }
46         }
47     }
48     left-=mf[dest];
49     if(dist[dest]<INF){
50         for(int u=dest;u!=src;u=point[from[u]^1]){
51             flow[from[u]]+=mf[dest];
52             flow[from[u]^1]-=mf[dest];
53         }
54     }
55     return dist[dest];
56 }
57 int min_cost_flow(){
58     int res=0,tmp,maxflow=2;
59     while(maxflow&&(tmp=sp(maxflow))<INF) res+=tmp;
60     return res;
61 }
62 int main(){
63     int n,m,x,y,z;
64     while(scanf("%d%d", &n, &m)==2){
65         init(n,1,n);
66         for(int i=0;i<m;i++){
67             scanf("%d%d%d", &x, &y, &z);
68             add_edge(x,y,1,z);
69             add_edge(y,x,1,z); //undirected
70         }
71         printf("%d\n",min_cost_flow());
72     }
73     return 0;
74 }

```

2.22 Mincostflow

```

1 typedef pair<int,int> P;
2 struct edge{
3     edge(){}
4     edge(int a,int b,int c,int d):to(a),cap(b),cost(c),rev(d){}
5     int to,cap,cost,rev;

```

```

6 };
7 #define V 1000
8 vector<edge> g[V];
9 int h[V], dist[V], prev_v[V], prev_e[V];
10 void add_edge(int from, int to, int cap, int cost) {
11     g[from].push_back(edge(to, cap, cost, g[to].size()));
12     g[to].push_back(edge(from, 0, -cost, g[from].size()-1));
13 }
14 int min_costflow(int s, int t, int f) {
15     int res=0;
16     memset(h, 0, sizeof(h));
17     while(f>0) {
18         priority_queue<P, vector<P>, greater<P> > que;
19         fill(dist, dist+V, 1e9);
20         dist[s]=0;
21         que.push(P(dist[s], s));
22         while(!que.empty()) {
23             P p=que.top();
24             que.pop();
25             int v=p.second;
26             if(dist[v]<p.first) continue;
27             for(int i=0; i<g[v].size(); ++i) {
28                 edge &e=g[v][i];
29                 if(e.cap>0&&dist[e.to]>dist[v]+e.cost+h[v]-h[e.to]) {
30                     dist[e.to]=dist[v]+e.cost+h[v]-h[e.to];
31                     prev_v[e.to]=v;
32                     prev_e[e.to]=i;
33                     que.push(P(dist[e.to], e.to));
34                 }
35             }
36         }
37         if(dist[t]==1e9) return -1;
38         for(int v=0; v<V; ++v) h[v]+=dist[v];
39         int d=f;
40         for(int v=t; v!=s; v=prev_v[v]) d=min(d, g[prev_v[v]][prev_e[v]].cap);
41         f-=d;
42         res+=d*h[t];
43         for(int v=t; v!=s; v=prev_v[v]) {
44             edge &e=g[prev_v[v]][prev_e[v]];
45             e.cap-=d;
46             g[v][e.rev].cap+=d;
47         }
48     }
49     return res;
50 }

```

2.23 Monotone

```

1 #include<cstdio>
2 #include<vector>
3 #include<algorithm>
4 #define N 50010
5 using namespace std;
6 long long dp[N], c[N], sum[N];

```

```

7 int len;
8 inline long long sq(long long x) {
9     return x*x;
10 }
11 inline long long cost(int a, int b) {
12     return sq(sum[b]-sum[a]-len);
13 }
14 int main() {
15     int n, i, j, l, r, m, s;
16     vector<int> k, p;
17     scanf("%d%d", &n, &len);
18     len++;
19     for(i=1; i<=n; i++) {
20         scanf("%lld", &c[i]);
21         c[i]++;
22         sum[i]=sum[i-1]+c[i];
23     }
24     p.push_back(1);
25     k.push_back(0);
26     for(i=1; i<=n; i++) {
27         j=upper_bound(p.begin(), p.end(), i)-1-p.begin();
28         dp[i]=dp[k[j]]+cost(k[j], i);
29         r=n+1;
30         while(!p.empty() && p.back()>i) {
31             if(dp[i]+cost(i, p.back())<=dp[k.back()]+cost(k.back(), p.back())) {
32                 r=p.back();
33                 p.pop_back();
34                 k.pop_back();
35             }
36             else break;
37         }
38         l=max(p.back()-1, i);
39         s=1;
40         while(l+s<r) s<=<=1;
41         while(s) {
42             while(l+s>=r) s>>=1;
43             if(!s) break;
44             if(dp[k.back()]+cost(k.back(), l+s)<dp[i]+cost(i, l+s)) l+=s;
45             else s>>=1;
46         }
47         if(l+1<=n) {
48             k.push_back(i);
49             p.push_back(l+1);
50         }
51     }
52 }
53 printf("%lld\n", dp[n]);
54 return 0;
55 }

```

2.24 MST Directed

```

1 #include<cstdio>
2 #include<vector>

```

```

3 #include<algorithm>
4 #define N 100100
5 using namespace std;
6 struct edge{
7     edge(){}
8     edge(int _f,int _d):f(_f),d(_d){}
9     int f;
10    int d;
11    bool operator<(const edge &rhs)const{return d<rhs.d;}
12 };
13 struct node{
14     int sz,v,now;
15     node *l,*r;
16     void pull() {sz=1+(l?l->sz:0)+(r?r->sz:0);}
17 }pq[N];
18 int pa[N],sub[N],stk[N],top;
19 bool vis[N],instk[N];
20 vector<edge> rg[N];
21 void init(int n){
22     for(int i=0;i<n;i++){
23         pa[i]=i;
24         sub[i]=0;
25         pq[i].l=pq[i].r=NULL;
26         pq[i].sz=1;
27         pq[i].v=i;
28         pq[i].now=0;
29     }
30 }
31 int find(int x){
32     if(pa[x]==x) return x;
33     int y=find(pa[x]);
34     if(pa[x]!=y) sub[x]+=sub[pa[x]],pa[x]=y;
35     return pa[x];
36 }
37 inline int get_sub(int x){
38     if(x==find(x)) return sub[x];
39     else return sub[x]+sub[pa[x]];
40 }
41 inline int get_cost(const node& a){
42     return rg[a.v][a.now].d-get_sub(a.v);
43 }
44 bool cmp(const node& a,const node& b){
45     return get_cost(a)<get_cost(b);
46 }
47 node* merge(node *a,node *b){
48     if(!a||!b) return a?a:b;
49     if(cmp(*b,*a)) swap(a,b);
50     a->r=merge(a->r,b);
51     if((a->l?a->l->sz:0)<(a->r?a->r->sz:0)) swap(a->l,a->r);
52     a.pull();
53     return a;
54 }
55 int min_cost_arborescence(int r,int n){
56     vis[r]=true;
57     int res=0;
58     for(int i=0;i<n;i++){

```

```

59         if(!vis[i]){
60             top=0;
61             int u=i;
62             while(!vis[u]){
63                 }
64             }
65         }
66     }
67 }
68 int main(){
69     int n,m,r,x,y,w;
70     scanf("%d%d%d",&n,&m,&r);
71     for(int i=0;i<m;i++){
72         scanf("%d%d%d",&x,&y,&w);
73         rg[y].push_back(edge(x,w));
74         sort()
75     }
76 }

```

2.25 NTT

```

1 //prime for 1<20 : 998244353, 1051721729, 1053818881
2 long long pow_mod(long long a,long long p,long long q){
3     int r=1;
4     while(p){
5         if(p&1) r=r*a%q;
6         p>>=1;
7         a=a*a%q;
8     }
9     return r;
10 }
11 bool prime_test(long long p){
12     long long q=p-1,s=0;
13     while(!(q&1)){
14         q>>=1;
15         s++;
16     }
17     for(int i=0;i<20;i++){
18         long long a=rand()%(p-1)+1,x=pow_mod(a,q,p);
19         if(x==1) continue;
20         bool flag=false;
21         for(int j=0;j<s;j++){
22             if(x==p-1){
23                 flag=true;
24                 break;
25             }
26             x=x*x%p;
27         }
28         if(!flag) return false;
29     }
30     return true;
31 }
32 void build(){
33     int num=0;

```

```

34 for(long long i=1000;num<2;i++){
35     long long p=i<<20|1;
36     if(prime_test(p)){
37         prm[num]=p;
38         bool flag=true;
39         for(long long g=2;flag;g++){
40             flag=false;
41             long long tmp=pow_mod(g,i,p);
42             for(int j=0;j<20;j++){
43                 rt[num][20-j]=tmp;
44                 if(tmp==1){
45                     flag=true;
46                     break;
47                 }
48                 tmp=tmp*tmp%p;
49             }
50         }
51         num++;
52     }
53 }
54 }
55 void FFT(long long x[], bool pos, int u){
56     for (int i=1, j=0; i<N; ++i){
57         for (int k=N>>1; !((j^=k)&k); k>>=1) ;
58         if (i>j) swap(x[i], x[j]);
59     }
60     for (int k=2; k<=N; k<=1){
61         long long om = pos?rt[u][__lg(k)]:pow_mod(rt[u][__lg(k)],prm[u]-2,prm[u]);
62         for (int j=0; j<N; j+=k){
63             long long mul = 1;
64             for (int i=j; i<j+k/2; i++){
65                 long long a = x[i], b = x[i+k/2]*mul%prm[u];
66                 x[i] = (a + b)%prm[u];
67                 x[i+k/2] = (a - b)%prm[u];
68                 mul = mul*om%prm[u];
69             }
70         }
71     }
72 }
73 //double
74 const double pi = 2.0 * acos(0);
75 const int N = 8;
76 complex<double> x[N];
77 void FFT(){
78     // reverse bit and replace
79     for (int i=1, j=0; i<N; ++i){
80         for (int k=N>>1; !((j^=k)&k); k>>=1) ;
81         if (i>j) swap(x[i], x[j]);
82     }
83     for (int k=2; k<=N; k<=1){
84         double w = -2.0 * pi / k;
85         complex<double> dth(cos(w), sin(w));
86         // 每k個做一次FFT
87         for (int j=0; j<N; j+=k){
88             complex<double> theta(1, 0);

```

```

89         for (int i=j; i<j+k/2; i++){
90             complex<double> a = x[i];
91             complex<double> b = x[i + k/2] * theta;
92             x[i] = a + b;
93             x[i + k/2] = a - b;
94             theta *= dtheta;
95         }
96     }
97 }
98 }

```

2.26 SCC

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define MAX 10010
4 vector<int> edge[MAX], group[MAX];
5 bool instk[MAX];
6 int stk[MAX], groupID[MAX], nGroup, dfn[MAX], low[MAX], top, nowDfn;
7 void tarjan(int start){
8     dfn[start]=low[start]=++nowDfn;
9     instk[start]=1;
10    stk[top++]=start;
11    for(int i=0;i<edge[start].size();i++){
12        int next=edge[start][i];
13        if(!dfn[next]){
14            tarjan(next);
15            if(low[start]>low[next])
16                low[start]=low[next];
17        }
18        if(instk[next])
19            if(low[start]>dfn[next])
20                low[start]=dfn[next];
21    }
22    if(dfn[start]==low[start]){
23        do{
24            --top;
25            instk[stk[top]]=0;
26            groupID[stk[top]]=nGroup;
27            group[nGroup].push_back(stk[top]);
28        }while(stk[top]!=start);
29        ++nGroup;
30    }
31 }
32 void init(int n){
33     for(int i=0;i<n;i++){
34         instk[i]=dfn[i]=0, edge[i].clear(), group[i].clear();
35     }
36     nowDfn=nGroup=top=0;
37 }
38 int main(){
39     int T,n,m,i,j,k,x,y;
40     while(scanf("%d%d",&n,&m),n||m){
41         init(n);

```



```

42     scanf("%d%d",&x,&y);
43     edge[x-1].push_back(y-1);
44 }
45 for(i=0;i<n;i++)
46     if(dfn[i]==0) tarjan(i);
47 if(nGroup==1) puts("Yes");
48 else puts("No");
49 }
50 return 0;
51 }

```

2.27 Splay Tree

```

1 #include<cstdio>
2 #include<string>
3 using namespace std;
4 struct node{
5     node *ch[2],*par;
6     long long sum;
7     int val,sz,add;
8     node(){}
9     node(int x):par(NULL),val(x),sum(x),add(0),sz(1){ch[0]=ch[1]=NULL;}
10    bool dir(){return !par||par->ch[1]==this;}
11    void pull();
12    void push();
13 }pool[100100];
14 inline long long qsum(node *x){
15     return x?1LL*x->add*x->sz+x->sum:0;
16 }
17 inline int qsz(node *x){return x?x->sz:0;}
18 void node::pull(){
19     sum=val+qsum(ch[0])+qsum(ch[1]);
20     sz=1+qsz(ch[0])+qsz(ch[1]);
21 }
22 void node::push(){
23     if(add){
24         val+=add;
25         sum+=add*sz;
26         if(ch[0]) ch[0]->add+=add;
27         if(ch[1]) ch[1]->add+=add;
28         add=0;
29     }
30 }
31 inline void con(node *p,node *c,bool d){
32     p->ch[d]=c;
33     if(c) c->par=p;
34 }
35 void splay(node *x){
36     x->push();
37     while(x->par){
38         node *p=x->par,*g=p->par;
39         bool d=x->dir(),pd=p->dir();
40         con(p,x->ch[d^1],d);
41         con(x,p,d^1);

```

```

42     if(g){
43         if(g->par) con(g->par,x,g->dir());
44         else x->par=NULL;
45         if(d^pd){
46             con(g,x->ch[d],pd);
47             con(x,g,pd^1);
48         }
49         else{
50             con(g,p->ch[pd^1],pd);
51             con(p,g,pd^1);
52         }
53         g->pull();
54     }
55     else x->par=NULL;
56     p->pull();
57     x->pull();
58 }
59 }
60 void check_tree(node *t,int d){
61     if(!t) return;
62     check_tree(t->ch[0],d+1);
63     for(int i=0;i<d;i++) printf("\t");
64     printf("%d\n",t->val);
65     check_tree(t->ch[1],d+1);
66 }
67 void split(node *t,int k,node *&a,node *&b){
68     if(!k){
69         a=NULL; b=t; return;
70     }
71     int rod;
72     while(k!=(rod=qsz(t->ch[0])+1)){
73         t->push();
74         if(k>rod) k-=rod,t=t->ch[1];
75         else t=t->ch[0];
76     }
77     splay(t);
78     a=t;
79     a->push();
80     b=a->ch[1];
81     a->ch[1]=NULL;
82     a->pull();
83     if(b) b->par=NULL;
84 }
85 node* merge(node *a,node *b){
86     if(!a) return b;
87     while(a->ch[1]){
88         a->push();
89         a=a->ch[1];
90     }
91     splay(a);
92     con(a,b,1);
93     a->pull();
94     return a;
95 }
96 int main(){
97     int n,q,x;

```

```

98  node *root=NULL,*a,*b,*c;
99  scanf("%d%d",&n,&q);
100  for(int i=0;i<n;i++){
101      scanf("%d",&x);
102      node *tmp=new (pool+i) node(x);
103      root=merge(root,tmp);
104  }
105  for(int i=0;i<q;i++){
106      char tp;
107      int x,y,z;
108      scanf(" %c%d%d",&tp,&x,&y);
109      split(root,x-1,a,b);
110      split(b,y-x+1,b,c);
111      if(tp=='C'){
112          scanf("%d",&z);
113          b->add+=z;
114      }
115      else printf("%lld\n",qsum(b));
116      root=merge(a,merge(b,c));
117  }
118  return 0;
119 }

```

2.28 Stable Marriage

```

1  #define F(n) Fi(i, n)
2  #define Fi(i, n) Fl(i, 0, n)
3  #define Fl(i, l, n) for(int i = l ; i < n ; ++i)
4  #include <bits/stdc++.h>
5  using namespace std;
6  int D, quota[205], weight[205][5];
7  int S, scoretodep[12005][205], score[5];
8  int P, prefer[12005][85], iter[12005];
9  int ans[12005];
10 typedef pair<int, int> PII;
11 map<int, int> samescore[205];
12 typedef priority_queue<PII, vector<PII>, greater<PII>> QQQ;
13 QQQ pri[205];
14 void check(int d) {
15     PII t = pri[d].top();
16     int v;
17     if (pri[d].size() - samescore[d][t.first] + 1 <= quota[d]) return;
18     while (pri[d].top().first == t.first) {
19         v = pri[d].top().second;
20         ans[v] = -1;
21         --samescore[d][t.first];
22         pri[d].pop();
23     }
24 }
25 void push(int s, int d) {
26     if (pri[d].size() < quota[d]) {
27         pri[d].push(PII(scoretodep[s][d], s));
28         ans[s] = d;
29         ++samescore[s][scoretodep[s][d]];

```

```

30     } else if (scoretodep[s][d] >= pri[d].top().first) {
31         pri[d].push(PII(scoretodep[s][d], s));
32         ans[s] = d;
33         ++samescore[s][scoretodep[s][d]];
34         check(d);
35     }
36 }
37 void f() {
38     int over;
39     while (true) {
40         over = 1;
41         Fi (q, S) {
42             if (ans[q] != -1 || iter[q] >= P) continue;
43             push(q, prefer[q][iter[q]++]);
44             over = 0;
45         }
46         if (over) break;
47     }
48 }
49 main() {
50     ios::sync_with_stdio(false);
51     cin.tie(NULL);
52     int sadmit, stof, dexceed, dfew;
53     while (cin >> D, D) { // Beware of the input format or judge may troll us.
54         sadmit = stof = dexceed = dfew = 0;
55         memset(iter, 0, sizeof(iter));
56         memset(ans, 0, sizeof(ans));
57         Fi (q, 205) {
58             pri[q] = QQQ();
59             samescore[q].clear();
60         }
61         cin >> S >> P;
62         Fi (q, D) {
63             cin >> quota[q];
64             Fi (w, 5) cin >> weight[q][w];
65         }
66         Fi (q, S) {
67             Fi (w, 5) cin >> score[w];
68             Fi (w, D) {
69                 scoretodep[q][w] = 0;
70                 F (5) scoretodep[q][w] += weight[w][i] * score[i];
71             }
72         }
73         Fi (q, S) Fi (w, P) {
74             cin >> prefer[q][w];
75             --prefer[q][w];
76         }
77         f();
78         Fi (q, D) sadmit += pri[q].size();
79         Fi (q, S) if (ans[q] == prefer[q][0]) ++stof;
80         Fi (q, D) if (pri[q].size() > quota[q]) ++dexceed;
81         Fi (q, D) if (pri[q].size() < quota[q]) ++dfew;
82         cout << sadmit << ' ' << stof << ' ' << dexceed << ' ' << dfew << '\n';
83     }
84 }

```

2.29 Suffix Array

```

1 //should initialize s and n first
2 #define N 301000
3 using namespace std;
4 char s[N]; //string=s,suffix array=sar,longest common prefix=lcp
5 int rk[2][N],id[2][N];
6 int n,p;
7 int cnt[N];
8 int len[N],od[N],sar[N];
9 inline int sr(int i,int t){ //rank of shifted position
10     return i+t<n?rk[p][i+t]:-1;
11 }
12 inline bool check_same(int i,int j,int t){
13     return rk[p][i]==rk[p][j]&&sr(i,t)==sr(j,t);
14 }
15 bool cmp(int i,int j){
16     return s[i]<s[j];
17 }
18 void sa(){ //length of array s
19     int i,t,now,pre;
20     memset(cnt,0,sizeof(cnt));
21     for(i=0;i<n;i++){
22         id[p][i]=i;
23         rk[p][i]=s[i];
24         cnt[s[i]]++;
25     }
26     for(i=1;i<128;i++) cnt[i]+=cnt[i-1];
27     sort(id[p],id[p]+n,cmp);
28     for(t=1;t<n;t<=1){
29         //least significant bit is already sorted
30         for(i=n-1;i>=0;i--){
31             now=id[p][i]-t;
32             if(now>=0) id[p^1][--cnt[rk[p][now]]]=now;
33         }
34         for(i=n-t;i<n;i++){
35             id[p^1][--cnt[rk[p][i]]]=i;
36         }
37         memset(cnt,0,sizeof(cnt));
38         now=id[p^1][0];
39         rk[p^1][now]=0;
40         cnt[0]++;
41         for(i=1;i<n;i++){
42             pre=now;
43             now=id[p^1][i];
44             if(check_same(pre,now,t)){
45                 rk[p^1][now]=rk[p^1][pre];
46             }
47             else{
48                 rk[p^1][now]=rk[p^1][pre]+1;
49             }
50             cnt[rk[p^1][now]]++;
51         }
52         p^=1;
53         if(rk[p][now]==n-1) break;
54         for(i=1;i<n;i++) cnt[i]+=cnt[i-1];

```

```

55     }
56     memcpy(sar,id[p],sizeof(sar));
57 }
58 void lcp(){
59     int i,l,pre;
60     for(i=0;i<n;i++) od[sar[i]]=i;
61     for(i=0;i<n;i++){
62         if(i) l=len[od[i-1]]?len[od[i-1]]-1:0;
63         else l=0;
64         if(od[i]){
65             pre=sar[od[i]-1];
66             while(pre+l<n&&i+l<n&&s[pre+l]==s[i+l]) l++;
67             len[od[i]]=l;
68         }
69         else len[0]=0;
70     }
71 }

```

2.30 Suffix Automaton

```

1 #include<bits/stdc++.h>
2 #define C 96
3 #define N 200100
4 using namespace std;
5 struct SAM{
6     struct node{
7         node *nxt[C],*pre;
8         int len;
9         vector<int> pos;
10     };
11     node mem[N*2],*root,*ed;
12     int top;
13     SAM(){
14         top = 0;
15         root = new_node(0);
16         ed = root;
17     }
18     node *new_node(int l){
19         for(int i=0;i<C;i++) mem[top].nxt[i]=NULL;
20         mem[top].pre=NULL;
21         mem[top].len=l;
22         mem[top].pos.clear();
23         return mem+(top++);
24     }
25     node *split_node(int l,node *p){
26         for(int i=0;i<C;i++) mem[top].nxt[i]=p->nxt[i];
27         mem[top].pre = p->pre;
28         mem[top].len = l;
29         mem[top].pos.assign()
30         p->pre = mem+top;
31         return mem+(top++);
32     }
33     void push(char c){
34         node *nw = new_node(ed->len+1),*ptr=ed->pre;

```

```

35     ed->nxt[c] = nw;
36     nw->pos.push_back(ed->len);
37     for(ptr; ptr=ptr->pre){
38         if(ptr->nxt[c]){
39             if(ptr->nxt[c]->len==ptr->len+1){
40                 nw->pre = ptr->nxt[c];
41             }
42             else{
43                 node *tmp=ptr->nxt[c];
44                 nw->pre = split_node(ptr->len+1,tmp);
45                 while(ptr && ptr->nxt[c]==tmp){
46                     ptr->nxt[c] = nw->pre;
47                     ptr = ptr->pre;
48                 }
49             }
50             break;
51         }
52         else{
53             ptr->nxt[c] = nw;
54         }
55     }
56     if(!nw->pre) nw->pre = root;
57     ed = ed->nxt[c];
58 }
59 void init(){
60     while(top){
61         mem[--top].pos.clear();
62     }
63     root = new_node(0);
64     ed = root;
65 }
66 void push(char *s){
67     for(int i=0; s[i]; i++) push(s[i]-32);
68 }
69 long long count(){
70     long long ans=0;
71     for(int i=1; i<top; i++){
72         ans+=mem[i].len-mem[i].pre->len;
73     }
74     return ans;
75 }
76 }sam;
77 char S[N];
78 int main(){
79     int T;
80     scanf("%d", &T);
81     while(T--){
82         scanf("%s", S);
83         sam.build(S);
84         printf("%lld\n", sam.count());
85     }
86     return 0;
87 }

```

2.31 Tonelli Shanks

```

1  #include<cstdio>
2  #include<cassert>
3  #include<cstdlib>
4  using namespace std;
5  int pow_mod(int a, int p, int q){ //a^p mod q
6      int r=1;
7      while(p){
8          if(p&1) r=1LL*r*a%q;
9          a=1LL*a*a%q;
10         p>>=1;
11     }
12     return r;
13 }
14 int Jacobi(int q, int p){ //q/p
15     if(p==1) return 1;
16     q%=p;
17     int c2=0, m2;
18     while(!(q&1)){
19         q>>=1;
20         c2^=1;
21     }
22     if((p&7)==7|| (p&7)==1||!c2) m2=1;
23     else m2=-1;
24     if((p&2)&&(q&2)) m2*=-1;
25     return m2*Jacobi(p, q);
26 }
27 int Tonelli_Shanks(int a, int p){ //p is prime, gcd(a,p)=1
28     if(p==2) return 1;
29     if(Jacobi(a, p)==-1) return -1;
30     int s=0, q=p-1, z=2;
31     while(!(q&1)) q>>=1, s++;
32     while(Jacobi(z, p)==1) z++;
33     z = pow_mod(z, q, p);
34     int zp[30]={z};
35     for(int i=1; i<s; i++) zp[i]=1LL*zp[i-1]*zp[i-1]%p;
36     int r = pow_mod(a, (q+1)>>1, p), t = pow_mod(a, q, p);
37     while(t!=1){
38         int m=0;
39         for(int i=t; i!=1; i=1LL*i*i%p) m++;
40         r=1LL*r*zp[s-m-1]%p;
41         t=1LL*t*zp[s-m]%p;
42     }
43     return r;
44 }
45 int main(){
46     for(int i=0; i<37; i++){
47
48     }
49     return 0;
50 }

```

2.32 Treap

```

1 struct Treap{
2     Treap *l,*r;
3     int pri,sz,val,add;
4     Treap(int _val):pri(rand()),sz(1),val(_val),add(0),l(NULL),r(NULL){}
5 };
6
7 int size(Treap *t){
8     return t?t->sz:0;
9 }
10 void pull(Treap *t){
11     t->sz=size(t->l)+size(t->r)+1;
12 }
13 void push(Treap *t){
14     t->val+=t->add;
15     if(t->l) t->l->add+=t->add;
16     if(t->r) t->r->add+=t->add;
17     t->add=0;
18 }
19 Treap* merge(Treap *a,Treap *b){
20     if(!a||!b) return a?a:b;
21     if(a->pri > b->pri){
22         push(a);
23         a->r = merge(a->r,b);
24         pull(a);
25         return a;
26     }
27     else{
28         push(b);
29         b->l = merge(a,b->l);
30         pull(b);
31         return b;
32     }
33 }
34 void split(Treap *t,int k,Treap *&a,Treap *&b){
35     if(!t) a=b=NULL;
36     else{
37         push(t);
38         if(size(t->l) < k){
39             a=t;
40             split(t->r,k-size(t->l)-1,a->r,b);
41             pull(a);
42         }
43         else{
44             b=t;
45             split(t->l,k,a,b->l);
46             pull(b);
47         }
48     }
49 }

```

2.33 Z Algorithm

```

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

```

3 Lucas's theorem

$$\binom{m}{n} \equiv \prod_{i=0}^k \binom{m_i}{n_i} \pmod{p}$$

where $m = m_k p^k + m_{k-1} p^{k-1} + \dots + m_1 p + m_0$ and $n = n_k p^k + n_{k-1} p^{k-1} + \dots + n_1 p + n_0$.

4 無權邊的生成樹個數 Kirchhoff's Theorem

- 定義 $n \times m$ 矩陣 $E = (a_{i,j})$, n 為點數, m 為邊數, 若 i 點在 j 邊上, i 為小點 $a_{i,j} = 1$, i 為大點 $a_{i,j} = -1$, 否則 $a_{i,j} = 0$ 。
(證明省略)
 - 令 $E(E^T) = Q$, 他是一種有負號的 kirchhoff 的矩陣, 取 Q 的子矩陣即為 $F(F^T)$
- 結論: 做 Q 取子矩陣算 \det 即為所求。(除去第一行第一列 by mz)

5 monge

$$i \leq i' < j \leq j'$$

$$m(i, j) + m(i', j') \leq m(i', j) + m(i, j')$$

$$k(i, j - 1) \leq k(i, j) \leq k(i + 1, j)$$

6 四心

$$\frac{sa* A + sb*B + sc*C}{sa + sb + sc}$$

外心 $\sin 2A : \sin 2B : \sin 2C$

內心 $\sin A : \sin B : \sin C$
垂心 $\tan A : \tan B : \tan C$
重心 $1 : 1 : 1$

7 Runge-Kutta

$$\begin{aligned}y_{n+1} &= y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\k_1 &= f(t_n, y_n) \\k_2 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \\k_3 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2) \\k_4 &= f(t_n + h, y_n + hk_3)\end{aligned}$$

8 Householder Matrix

$$I - 2\frac{vv^T}{v^Tv}$$