ACM ICPC - Code Notebook

David Batista - david.batista3010@gmail.com
Federal University of Itajuba - Brazil

September 8, 2017

Contents

Chapter 1

Data Structure

1.1 Segment Tree

1.1.1 Segment Tree & Lazy Propagation

```cpp
class segtree
{
    const static int N=100000;
    int tr[4*N], lazy[4*N];
public:
    segtree(){};
    void clear()
    {
        memset(tr, 0, sizeof(tr));
        memset(lazy, 0, sizeof(lazy));
    }
    void build(int no, int l, int r, vector<int>&data)
    {
        if(l==r)
        {
            tr[no]=data[l];
            return;
        }
        int nxt=no*2;
        int mid=(l+r)/2;
        build(nxt, l, mid, data);
        build(nxt+1, mid+1, r, data);
        tr[no]=tr[nxt]+tr[nxt+1];
    }
    void propagate(int no, int l, int r)
    {
        if(!lazy[no])
            return;

        tr[no]+=(r-l+1)*lazy[no];
        if(l!=r)
        {
            int nxt=no*2;
            lazy[nxt]+=lazy[no];
            lazy[nxt+1]+=lazy[no];
        }
        lazy[no]=0;
    }
    void update(int no, int l, int r, int i, int j, int x)
    {
        propagate(no, l, r);
        if(l>j || r<i)
            return;
        if(l>=i && r<=j)
        {
            lazy[no]=x;
            propagate(no, l, r);
            return;
        }
        int nxt=no*2;
```

```
51            int mid=(l+r)/2;
52            update(nxt, l, mid, i, j, x);
53            update(nxt+1, mid+1, r, i, j, x);
54            tr[no]=tr[nxt]+tr[nxt+1];
55        }
56        int query(int no, int l, int r, int i, int j)
57        {
58            propagate(no, l, r);
59            if(l>j || r<i)
60                return 0;
61            if(l>=i && r<=j)
62                return tr[no];
63            int nxt=no*2;
64            int mid=(l+r)/2;
65            int ql=query(nxt, l, mid, i, j);
66            int qr=query(nxt+1, mid+1, r, i, j);
67            return (ql+qr);
68        }
69    };
```

1.1.2 Segment Tree & Hash

```cpp
const int NC=1e+5;
ull aux[NC];
void precalc(ull k)//prime k
{
    aux[0]=1LL;
    for(int i=1; i<NC; i++)
        aux[i]=aux[i-1]*k;
}

class node
{
public:
    ull v;
    int s;
    node(){};
    node(ull _v, int _s)
    {
        v=_v; s=_s;
    }//
    node operator +(const node &foo) const
    {
        return node(v+(foo.v*aux[s]), s+foo.s);
    }
};

class segtree
{
    const static int N=1e+5+35;
    node tr[4*N];
public:
    segtree(){};
    void update(int no, int l, int r, int i, int j, node x)
    {
        if(l>j || r<i)
            return;
        if(l>=i && r<=j)
        {
            tr[no]=x;
            return;
        }
        int mid=(l+r)>>1;
        int nxt=no<<1;
        update(nxt, l, mid, i, j, x);
        update(nxt+1, mid+1, r, i, j, x);
        tr[no]=tr[nxt]+tr[nxt+1];
    }
    node query(int no, int l, int r, int i, int j)
    {
        if(l>j || r<i)
            return node(0LL, 0);
        if(l>=i && r<=j)
            return tr[no];
        int mid=(l+r)>>1;
        int nxt=no<<1;
        node ql=query(nxt, l, mid, i, j);
        node qr=query(nxt+1, mid+1, r, i, j);
        return ql+qr;
    }
};
```

1.1.3 Segment Tree & Range Graph

```
1   /*
2   call build(1, 0, n-1, 0) & build(1, 0, n-1, 1)
3   to build base graph on tree
4
5   update(1, 0, n-1, l, r, x, 0):
6   add vertex [x,x]->[l, r]
7
8   update(1, 0, n-1, l, r, x, 1):
9   add vertex [l,r]->[x,x]
10  */
11  class segtree
12  {
13      const static int N=1e+5+35;
14  public:
15      vector< pair<int,int> >data[8*N]; //graph
16      int idx[4*N][2], id;
17      segtree(){};
18      void set(int n)
19      {
20          id=n;
21          for(int i=0; i<8*n; i++)
22              data[i].clear();
23      }
24      inline void addEdge(int u, int v, int w)
25      {
26          data[u].pb({v, w});
27      }
28      inline void build(int no, int l, int r, int t)
29      {
30          idx[no][t]=id++;
31          if(l==r)
32          {
33              if(!t)
34                  addEdge(idx[no][t], l, 0);
35              else
36                  addEdge(l, idx[no][t], 0);
37              return;
38          }
39          int nxt=no<<1;
40          int mid=(l+r)>>1;
41          build(nxt, l, mid, t);
42          build(nxt+1, mid+1, r, t);
43          if(!t)
44          {
45              addEdge(idx[no][t], idx[nxt][t], 0);
46              addEdge(idx[no][t], idx[nxt+1][t], 0);
47          }
48          else
49          {
50              addEdge(idx[nxt][t], idx[no][t], 0);
51              addEdge(idx[nxt+1][t], idx[no][t], 0);
52          }
53      }
54      inline void update(int no, int l, int r, int i, int j, int u, int w, int t)
55      {
56          if(l>j || r<i)
57              return;
58          if(l>=i && r<=j)
59          {
60              if(!t)
61                  addEdge(u, idx[no][t], w);
62              else
63                  addEdge(idx[no][t], u, w);
64              return;
65          }
66          int nxt=no<<1;
67          int mid=(l+r)>>1;
68          update(nxt, l, mid, i, j, u, w, t);
69          update(nxt+1, mid+1, r, i, j, u, w, t);
70      }
71  };
```

1.1.4 Quadtree

```cpp
class quadtree
{
    //needs to be NxN
    const static int N=100000;
    int tr[16*N];
public:
    quadtree(){};
    void build(int node, int l1, int r1, int l2, int r2, vector< vector<int> >data)
    {
        if(l1==l2 && r1==r2)
        {
            tr[node]=data[l1][r1];
            return;
        }
        int nxt=node*4;
        int midl=(l1+l2)/2;
        int midr=(r1+r2)/2;

        build(nxt-2, l1, r1, midl, midr, data);
        build(nxt-1, midl+1, r1, l2, midr, data);
        build(nxt, l1, midr+1, midl, r2, data);
        build(nxt+1, midl+1, midr+1, l2, r2, data);

        tr[node]=tr[nxt-2]+tr[nxt-1]+tr[nxt]+tr[nxt+1];
    }
    void update(int node, int l1, int r1, int l2, int r2, int i, int j, int x)
    {
        if(l1>l2 || r1>r2)
            return;
        if(i>l2 || j>r2 || i<l1 || j<r1)
            return;
        if(i==l1 && i==l2 && j==r1 && j==r2)
        {
            tr[node]=x;
            return;
        }
        int nxt=node*4;
        int midl=(l1+l2)/2;
        int midr=(r1+r2)/2;

        update(nxt-2, l1, r1, midl, midr, i, j, x);
        update(nxt-1, midl+1, r1, l2, midr, i, j, x);
        update(nxt, l1, midr+1, midl, r2, i, j, x);
        update(nxt+1, midl+1, midr+1, l2, r2, i, j, x);

        tr[node]=tr[nxt-2]+tr[nxt-1]+tr[nxt]+tr[nxt+1];
    }
    int query(int node, int l1, int r1, int l2, int r2, int i1, int j1, int i2, int j2)
    {
        if(i1>l2 || j1>r2 || i2<l1 || j2<r1 || i1>i2 || j1>j2)
            return 0;
        if(i1<=l1 && j1<=r1 && l2<=i2 && r2<=j2)
            return tr[node];
        int nxt=node*4;
        int midl=(l1+l2)/2;
        int midr=(r1+r2)/2;

        int q1=query(nxt-2, l1, r1, midl, midr, i1, j1, i2, j2);
        int q2=query(nxt-1, midl+1, r1, l2, midr, i1, j1, i2, j2);
        int q3=query(nxt, l1, midr+1, midl, r2, i1, j1, i2, j2);
        int q4=query(nxt+1, midl+1, midr+1, l2, r2, i1, j2, i2, j2);

        return q1+q2+q3+q4;
    }
};
```

1.1.5 Mergesort Segtree

```
1   class mergesort_segtree
2   {
3       const static int N=100000;
4       vector<int>tr[4*N];
5   public:
6       mergesort_segtree(){};
7       void build(int no, int l, int r, vector<int>&data)
8       {
9           if(l==r)
10          {
11              tr[no].push_back(data[l]);
12              return;
13          }
14          int nxt=no*2;
15          int mid=(l+r)/2;
16          build(nxt, l, mid, data);
17          build(nxt+1, mid+1, r, data);
18          tr[no].resize(tr[nxt].size()+tr[nxt+1].size());
19          merge(tr[nxt].begin(), tr[nxt].end(), tr[nxt+1].begin(), tr[nxt+1].end(), tr[no].begin());
20      }
21      //how many numbers in (i, j) are greater or equal than k
22      int query(int no, int l, int r, int i, int j, int k)
23      {
24          if(r<i || l>j)
25              return 0;
26          if(l>=i && r<=j)
27              return (int)(tr[no].end()-upper_bound(tr[no].begin(), tr[no].end(), k));
28          int nxt=no*2;
29          int mid=(l+r)/2;
30          int ql=query(nxt, l, mid, i, j, k);
31          int qr=query(nxt+1, mid+1, r, i, j, k);
32          return ql+qr;
33      }
34  };
```

1.1.6 Persistent Segtree

```cpp
class persistent_segtree
{
    const static int N=100000;
    int n, cnt, id;
    int tr[N];
    int root[N], L[N], R[N];
public:
    persistent_segtree(){};
    void set(int _n)
    {
        memset(tr, 0, sizeof(tr));
        memset(root, 0, sizeof(root));
        memset(L, 0, sizeof(L));
        memset(R, 0, sizeof(R));
        id=0;
        cnt=1;
        n=_n;
    }
    void build(int no, int l, int r, vector<int>&data)
    {
        if(l==r)
        {
            tr[no]=data[l];
            return;
        }
        int mid=(l+r)/2;
        L[no]=cnt++;
        R[no]=cnt++;
        build(L[no], l, mid, data);
        build(R[no], mid+1, r, data);
        tr[no]=tr[ L[no] ]+tr[ R[no] ];
    }
    int update(int no, int l, int r, int i, int x)
    {
        int newno=cnt++;
        tr[newno]=tr[no];
        L[newno]=L[no];
        R[newno]=R[no];
        if(l==r)
        {
            tr[newno]=x;
            return newno;
        }
        int mid=(l+r)/2;
        if(i<=mid)
            L[newno]=update(L[newno], l, mid, i, x);
        else
            R[newno]=update(R[newno], mid+1, r, i, x);
        tr[newno]=tr[ L[newno] ]+tr[ R[newno] ];
        return newno;
    }
    int query(int no, int l, int r, int i, int j)
    {
        if(r<i || l>j)
            return 0;
        if(l>=i && r<=j)
            return tr[no];
        int mid=(l+r)/2;
        int ql=query(L[no], l, mid, i, j);
        int qr=query(R[no], mid+1, r, i, j);
        return ql+qr;
    }
    //update the i-th value to x.
    void update(int i, int x)
    {
        root[id+1]=update(root[id], 0, n-1, i, x);
        id++;
    }
    //returns sum(l, r) after the k-th update.
    int query(int l, int r, int k)
    {
        return query(root[k], 0, n-1, l, r);
    }
};
```

10

1.2 Fenwick Tree

1.2.1 Fenwick Tree 1D

```cpp
class fenwicktree
{
    #define D(x) x&(-x)
    const static int N=100000;
    int tr[N], n;
public:
    fenwicktree(){};
    void build(int _n)
    {
        n=_n;
        memset(tr, 0, sizeof(tr));
    }
    void update(int i, int x)
    {
        for(i++; i<=n; i+=D(i))
            tr[i]+=x;
    }
    int query(int i)
    {
        int ret=0;
        for(i++; i>0; i-=D(i))
            ret+=tr[i];
        return ret;
    }
    int rquery(int l, int r)
    {
        return query(r)-query(l-1);
    }
    void set(int i, int x)
    {
        update(i, -rquery(i, i)+x);
    }
    void rset(int l, int r, int x)
    {
        update(l, x);
        update(r+1, -x);
    }
};
```

1.2.2 Fenwick Tree 2D

```cpp
class fenwicktree
{
    #define D(x) x&(-x)
    const static int N=1000;
    int tr[N][N], n, m;
public:
    fenwicktree(){};
    void build(int _n, int _m)
    {
        n=_n, m=_m;
        memset(tr, 0, sizeof(tr));
    }
    void update(int r, int c, int x)
    {
        for(int i=r+1; i<=n; i+=D(i))
            for(int j=c+1; j<=m; j+=D(j))
                tr[i][j]+=x;
    }
    int query(int r, int c)
    {
        int ret=0;
        for(int i=r+1; i>0; i-=D(i))
            for(int j=c+1; j>0; j-=D(j))
                ret+=tr[i][j];
        return ret;
    }
    int rquery(int r1, int c1, int r2, int c2)
    {
        if((r1>r2 && c1>c2) || (r1==r2 && c1>c2) || (r1>r2 && c1==c2))
        {
            swap(r1, r2);
            swap(c1, c2);
        }
        else if(r1<r2 && c1>c2)
        {
            swap(c1,c2);
        }
        else if(r1>r2 && c1<c2)
        {
            swap(r1,r2);
        }
        return query(r2, c2)-query(r1-1, c2)-query(r2, c1-1)+query(r1-1, c1-1);
    }
    void set(int r, int c, int x)
    {
        update(r, c, -rquery(r, c, r, c)+x);
    }
};
```

1.3 Cartesian Tree

1.3.1 Cartesian Tree

```
1   //srand(time(NULL))
2   int vrand()
3   {
4       return abs(rand()<<(rand()%31));
5   }
6
7   struct node
8   {
9       //x=key, y=priority key, c=tree count
10      int x, y, c;
11      node *L, *R;
12      node(){};
13      node(int _x)
14      {
15          x=_x, y=vrand(), c=0;
16          L=R=NULL;
17      }
18  };
19
20  int cnt(node *root)
21  {
22      return root?root->c:0;
23  }
24
25  void upd_cnt(node *root)
26  {
27      if(root)
28          root->c=1+cnt(root->L)+cnt(root->R);
29  }
30
31  void split(node *root, int x, node *&L, node *&R)
32  {
33      if(!root)
34          L=R=NULL;
35      else if(x < root->x)
36          split(root->L, x, L, root->L), R=root;
37      else
38          split(root->R, x, root->R, R), L=root;
39      upd_cnt(root);
40  }
41
42  void insert(node *&root, node *it)
43  {
44      if(!root)
45          root=it;
46      else if(it->y > root->y)
47          split(root, it->x, it->L, it->R), root=it;
48      else
49          insert(it->x < root->x? root->L:root->R, it);
50      upd_cnt(root);
51  }
52
53  void merge(node *&root, node *L, node *R)
54  {
55      if(!L || !R)
56          root=L?L:R;
57      else if(L->y > R->y)
58          merge(L->R, L->R, R), root=L;
59      else
60          merge(R->L, L, R->L), root=R;
61      upd_cnt(root);
62  }
63
64  void erase(node *&root, int x)
65  {
66      if(root->x==x)
67          merge(root, root->L, root->R);
68      else
69          erase(x < root->x? root->L:root->R, x);
70      upd_cnt(root);
71  }
```

```
72
73   node *unite(node *L, node *R)
74   {
75       if(!L || !R)
76           return L?L:R;
77       if(L->y < R->y)
78           swap(L, R);
79       node *Lt, *Rt;
80       split(R, L->x, Lt, Rt);
81       L->L=unite(L->L, Lt);
82       L->R=unite(L->R, Rt);
83       return L;
84   }
85
86   int find(node *root, int x)
87   {
88       if(!root)
89           return 0;
90       if(root->x==x)
91           return 1;
92       if(x > root->x)
93           return find(root->R, x);
94       else
95           return find(root->L, x);
96   }
97
98   int findkth(node *root, int x)
99   {
100      if(!root)
101          return -1;
102      int Lc=cnt(root->L);
103      if(x-Lc-1==0)
104          return root->x;
105      if(x>Lc)
106          return findkth(root->R, x-Lc-1);
107      else
108          return findkth(root->L, x);
109  }
```

1.3.2 Implicit Cartesian Tree

```
1   //srand(time(NULL))
2   int vrand()
3   {
4       return abs(rand()<<(rand()%31));
5   }
6
7   struct node
8   {
9       //basic treap: x=key, y=priority key, c=tree count;
10      int x, y, c;
11      //treap operations: v=max(x), lazy=lazy value of propagation, rev=reversed
12      int v, lazy, rev;
13
14      node *L, *R;
15      node(){};
16      node(int _x)
17      {
18          x=_x, y=vrand();
19          L=R=NULL;
20          v=x;
21          lazy=0;
22          rev=0;
23      }
24  };
25
26  //updating functions
27  inline int get_cnt(node *root)
28  {
29      return root?root->c:0;
30  }
31
32  inline void upd_cnt(node *root)
33  {
34      if(root)
35          root->c=1+get_cnt(root->L)+get_cnt(root->R);
36  }
37
38  inline void push(node *&root)
39  {
40      if(root && root->rev)
41      {
42          root->rev=0;
43          swap(root->L, root->R);
44          if(root->L)
45              root->L->rev^=1;
46          if(root->R)
47              root->R->rev^=1;
48      }
49  }
50
51  inline void propagate(node *&root)
52  {
53      if(root)
54      {
55          if(!root->lazy)
56              return;
57          int lazy=root->lazy;
58          root->x+=lazy;
59
60          if(root->L)
61              root->L->lazy=lazy;
62          if(root->R)
63              root->R->lazy=lazy;
64          root->lazy=0;
65      }
66  }
67
68  inline int get_max(node *root)
69  {
70      return root?root->v:-INF;
71  }
72
73  inline void upd_max(node *root)
74  {
```

```
75      if(root)
76          root->v=max(root->x, max(get_max(root->L), get_max(root->R)));
77  }
78
79  inline void update(node *root)
80  {
81      propagate(root);
82      upd_cnt(root);
83      upd_max(root);
84  }
85
86  void merge(node *&root, node *L, node *R)
87  {
88      push(L);
89      push(R);
90      if(!L || !R)
91          root=L?L:R;
92      else if(L->y > R->y)
93          merge(L->R, L->R, R), root=L;
94      else
95          merge(R->L, L, R->L), root=R;
96      update(root);
97  }
98
99  void split(node *root, node *&L, node *&R, int x, int add=0)
100 {
101     if(!root)
102         return void(L=R=NULL);
103     push(root);
104     int ix=add+get_cnt(root->L); //implicit key
105     if(x<=ix)
106         split(root->L, L, root->L, x, add), R=root;
107     else
108         split(root->R, root->R, R, x, add+1+get_cnt(root->L)), L=root;
109     update(root);
110 }
111
112 //insert function
113 void insert(node *&root, int pos, int x)//(insert x at position pos)
114 {
115     node *R1, *R2;
116     split(root, R1, R2, pos);
117     merge(R1, R1, new node(x));
118     merge(root, R1, R2);
119 }
120
121 //erase value x
122 void erase_x(node *&root, int x)
123 {
124     if(!root)
125         return;
126     if(root->x==x)
127         merge(root, root->L, root->R);
128     else
129         erase_x(x < root->x? root->L:root->R, x);
130     update(root);
131 }
132
133 //erase kth value
134 void erase_kth(node *&root, int x)
135 {
136     if(!root)
137         return;
138     int Lc=get_cnt(root->L);
139     if(x-Lc-1==0)
140         merge(root, root->L, root->R);
141     else if(x>Lc)
142         erase_kth(root->R, x-Lc-1);
143     else
144         erase_kth(root->L, x);
145     update(root);
146 }
147
148 //add x to [l,r]
149 inline void paint(node *&root, int l, int r, int x)
150 {
151     node *R1, *R2, *R3;
```

```
152        split(root, R1, R2, l);
153        split(R2, R2, R3, r−l+1);
154        R2−>lazy=x;
155        propagate(R2);
156
157        merge(root, R1, R2);
158        merge(root, root, R3);
159    }
160
161    //max range query [l,r]
162    inline int rquery(node *&root, int l, int r)
163    {
164        node *R1, *R2, *R3;
165        split(root, R1, R2, l);
166        split(R2, R2, R3, r−l+1);
167        int ret=R2−>v;
168        merge(root, R1, R2);
169        merge(root, root, R3);
170        return ret;
171    }
172
173    inline void reverse(node *&root, int l, int r)//reverse elements [l, r]
174    {
175        node *R1, *R2, *R3;
176        split(root, R1, R2, l);
177        split(R2, R2, R3, r−l+1);
178        R2−>rev^=1;
179        merge(root, R1, R2);
180        merge(root, root, R3);
181    }
182
183    //output functions
184    int poscnt=0;
185    void output_all(node *root)
186    {
187        if(!root)
188            return;
189        update(root);
190        push(root);
191        output_all(root−>L);
192        printf("[%d]_%d\n", poscnt++, root−>x);
193        output_all(root−>R);
194    }
195
196    int output_kth(node *root, int x)
197    {
198        if(!root)
199            return −1;
200        update(root);
201        push(root);
202        int Lc=get_cnt(root−>L);
203        if(x−Lc−1==0)
204            return root−>x;
205        if(x>Lc)
206            return output_kth(root−>R, x−Lc−1);
207        else
208            return output_kth(root−>L, x);
209    }
```

1.3.3 Implicit Cartesian Tree & Hash

```
1   const int NC=1e+5;
2   ull aux[NC];
3   void precalc(ull k)//prime k
4   {
5       aux[0]=1LL;
6       for(int i=1; i<NC; i++)
7           aux[i]=aux[i-1]*k;
8   }
9   class hnode
10  {
11  public:
12      ull v;
13      int s;
14      hnode(){};
15      hnode(ull _v, int _s)
16      {
17          v=_v; s=_s;
18      }
19      hnode operator +(const hnode &foo) const
20      {
21          return hnode(v+(foo.v*aux[s]), s+foo.s);
22      }
23  };
24
25  //srand(time(NULL))
26  int vrand()
27  {
28      return abs(rand()<<(rand()%31));
29  }
30  struct node
31  {
32      int x, y, c;
33      int lazy, rev;
34      hnode v;
35      node *L, *R;
36      node(){};
37      node(int _x)
38      {
39          x=_x, y=vrand();
40          L=R=NULL;
41          v=hnode((ull)_x, 1);
42          lazy=0;
43          rev=0;
44      }
45  };
46
47  //updating functions
48  inline int get_cnt(node *root)
49  {
50      return root?root->c:0;
51  }
52
53  inline void upd_cnt(node *root)
54  {
55      if(root)
56      {
57          root->c=1+get_cnt(root->L)+get_cnt(root->R);
58      }
59  }
60
61  inline void push(node *&root)
62  {
63      if(root && root->rev)
64      {
65          root->rev=0;
66          swap(root->L, root->R);
67          if(root->L)
68              root->L->rev^=1;
69          if(root->R)
70              root->R->rev^=1;
71      }
72  }
73
74  inline void propagate(node *&root)
```

```
75    {
76        if(root)
77        {
78            if(!root->lazy)
79                return;
80            int lazy=root->lazy;
81            root->x=lazy;
82            root->v=hnode(lazy, root->v.s);
83            if(root->L)
84                root->L->lazy=lazy;
85            if(root->R)
86                root->R->lazy=lazy;
87            root->lazy=0;
88        }
89    }
90
91    inline hnode getHash(node *root)
92    {
93        if(root)
94        {
95            propagate(root);
96            return root->v;
97        }
98        return hnode(0, 0);
99    }
100
101   inline void updHash(node *root)
102   {
103       if(root)
104           root->v=(hnode(root->x, 1)+getHash(root->L))+getHash(root->R);
105   }
106
107   inline void update(node *root)
108   {
109       propagate(root);
110       upd_cnt(root);
111       updHash(root);
112   }
113
114   void merge(node *&root, node *L, node *R)
115   {
116       push(L);
117       push(R);
118       if(!L || !R)
119           root=L?L:R;
120       else if(L->y > R->y)
121           merge(L->R, L->R, R), root=L;
122       else
123           merge(R->L, L, R->L), root=R;
124       update(root);
125   }
126
127   void split(node *root, node *&L, node *&R, int x, int add=0)
128   {
129       if(!root)
130           return void(L=R=NULL);
131       push(root);
132       int ix=add+get_cnt(root->L); //implicit key
133       if(x<=ix)
134           split(root->L, L, root->L, x, add), R=root;
135       else
136           split(root->R, root->R, R, x, add+1+get_cnt(root->L)), L=root;
137       update(root);
138   }
139
140   //insert function
141   void insert(node *&root, int pos, int x)//(insert x at position pos)
142   {
143       node *R1, *R2;
144       split(root, R1, R2, pos);
145       merge(R1, R1, new node(x));
146       merge(root, R1, R2);
147   }
148
149   //erase value x
150   void erase_x(node *&root, int x)
151   {
```

```
152        if(!root)
153            return;
154        if(root->x==x)
155            merge(root, root->L, root->R);
156        else
157            erase_x(x < root->x? root->L:root->R, x);
158        update(root);
159    }
160
161    //1-indexed: erase kth value
162    void erase_kth(node *&root, int x)
163    {
164        if(!root)
165            return;
166        int Lc=get_cnt(root->L);
167        if(x-Lc-1==0)
168            merge(root, root->L, root->R);
169        else if(x>Lc)
170            erase_kth(root->R, x-Lc-1);
171        else
172            erase_kth(root->L, x);
173        update(root);
174    }
175
176    //change [l, r] to x: l==r only
177    inline void paint(node *&root, int l, int r, int x)
178    {
179        node *R1, *R2, *R3;
180        split(root, R1, R2, l);
181        split(R2, R2, R3, r-l+1);
182        R2->lazy=x;
183        propagate(R2);
184
185        merge(root, R1, R2);
186        merge(root, root, R3);
187    }
188
189    //hash from [l, r]
190    inline hnode rquery(node *&root, int l, int r)
191    {
192        node *R1, *R2, *R3;
193        split(root, R1, R2, l);
194        split(R2, R2, R3, r-l+1);
195        hnode ret=R2->v;
196        merge(root, R1, R2);
197        merge(root, root, R3);
198        return ret;
199    }
200
201    //reverse elements [l, r]
202    inline void reverse(node *&root, int l, int r)
203    {
204        node *R1, *R2, *R3;
205        split(root, R1, R2, l);
206        split(R2, R2, R3, r-l+1);
207        R2->rev^=1;
208        merge(root, R1, R2);
209        merge(root, root, R3);
210    }
211
212    //output functions
213    int poscnt=0;
214    void output_all(node *root)
215    {
216        if(!root)
217            return;
218        update(root);
219        push(root);
220        output_all(root->L);
221        printf("[%d]_%d\n", poscnt++, root->x);
222        output_all(root->R);
223    }
224
225    //1-indexed
226    int output_kth(node *root, int x)
227    {
228        if(!root)
```

```
229            return −1;
230        update ( root ) ;
231        push ( root ) ;
232        int Lc=get_cnt ( root−>L ) ;
233        if ( x−Lc−1==0)
234            return root−>x ;
235        if ( x>Lc )
236            return output_kth ( root−>R, x−Lc−1) ;
237        else
238            return output_kth ( root−>L, x ) ;
239    }
```

1.4 Merge Sort & Swap Count

1.4.1 Merge Sort & Vector

```
1   #define INF 0x3F3F3F3F
2   int mergesort(vector<int>&data)
3   {
4       if(data.size()==1)
5           return 0;
6       vector<int>L, R;
7       int t=data.size();
8       for(int i=0; i<t/2; i++)
9           L.push_back(data[i]);
10      for(int i=t/2; i<t; i++)
11          R.push_back(data[i]);
12      int ret=mergesort(L)+mergesort(R);
13      for(int i=0, j=0, k=0; j<L.size() || k<R.size(); i++)
14      {
15          int x=j<L.size()?L[j]:INF;
16          int y=k<R.size()?R[k]:INF;
17          if(x<y)
18          {
19              data[i]=x;
20              j++;
21          }
22          else
23          {
24              data[i]=y;
25              k++;
26              ret+=(L.size()-j);
27          }
28      }
29      return ret;
30  }
```

1.4.2 Merge Sort & Array

```c
#define INF 0x3F3F3F3F
int temp[100000];
int mergesort(int data[], int l, int r)
{
    if(abs(l-r)<=1)
        return 0;
    int mid=(l+r)/2;
    int ret=mergesort(data, l, mid)+mergesort(data, mid, r);
    for(int i=l; i<r; i++)
        temp[i]=data[i];
    for(int i=l, j=l, k=mid; j<mid || k<r; i++)
    {
        int x=j<mid?temp[j]:INF;
        int y=k<r?temp[k]:INF;
        if(x<y)//x<=y
        {
            data[i]=x;
            j++;
        }
        else
        {
            data[i]=y;
            k++;
            ret+=(mid-j);
        }
    }
    return ret;
}
```

1.5 Sparse Table

```cpp
class sparsetable
{
    #define lbit(x) 63-__builtin_clzll(x);
    const static int N=100000, LN=20;
    int data[N][LN], n, ln;
public:
    sparsetable(){};
    void clear()
    {
        memset(data, 0, sizeof(data));
    }
    void build(vector<int>&foo)
    {
        n=foo.size();
        ln=lbit(n);
        for(int i=0; i<n; i++)
            data[i][0]=foo[i];
        for(int j=1; j<=ln; j++)
            for(int i=0; i<n-(1<<j)+1; i++)
                data[i][j]=max(data[i][j-1], data[i+(1<<(j-1))][j-1]);
    }
    int query(int l, int r)
    {
        int i=abs(l-r)+1;
        int j=lbit(i);
        return max(data[l][j], data[l-(1<<j)+1][j]);
    }
};
```

1.6 SQRT Decomposition

1.6.1 Array

```cpp
const int N=100000;
int SN=sqrt(N);

class mo
{
public:
    int l, r, i;
    mo(){};
    mo(int _l, int _r, int _i)
    {
        l=_l, r=_r, i=_i;
    }
    bool operator <(const mo &foo) const
    {
        if((r/SN)!=(foo.r/SN))
            return (r/SN)<(foo.r/SN);
        if(l!=foo.l)
            return l<foo.l;
        return i<foo.i;
    }
};

int data[N], freq[N], ans[N];
int cnt=0;
void update(int p, int s)
{
    int x=data[p];
    if(s==1)
    {
        if(freq[x]==0)
            cnt++;
    }
    else
    {
        if(freq[x]==1)
            cnt--;
    }
    freq[x]+=s;
}

int main()
{
    int n;
    scanf("%d", &n);
    for(int i=1; i<=n; i++)
        scanf("%d", &data[i]);

    int q;
    scanf("%d", &q);
    vector<mo>querys;
    for(int i=0; i<q; i++)
    {
        int l, r;
        scanf("%d_%d", &l, &r);
        querys.push_back(mo(l, r, i));
    }
    sort(querys.begin(), querys.end());

    int l=1, r=1;
    cnt=0;
    memset(freq, 0, sizeof(freq));
    update(l, 1);
    for(int i=0; i<q; i++)
    {
        int li=querys[i].l;
        int ri=querys[i].r;
        int ii=querys[i].i;
        while(l>li)
            update(--l, 1);
        while(r<ri)
            update(++r, 1);
```

```
72          while(l<li)
73              update(l++, -1);
74          while(r>ri)
75              update(r--, -1);
76          ans[ii]=cnt;
77      }
78      for(int i=0; i<querys.size(); i++)
79          printf("%d\n", ans[i]);
80      return 0;
81  }
```

1.6.2 Tree

```
1   #define pb push_back
2   #define ALL(x) x.begin(),x.end()
3
4   const int N=1e+5+35;
5   const int M=20;
6   const int SN=sqrt(2*N)+1;
7
8   class mo
9   {
10  public:
11      int l, r, i, lc;
12      mo(){};
13      mo(int _l, int _r, int _lc, int _i)
14      {
15          l=_l, r=_r, lc=_lc, i=_i;
16      }
17      bool operator <(const mo &foo) const
18      {
19          if((r/SN)!=(foo.r/SN))
20              return (r/SN)<(foo.r/SN);
21          if(l!=foo.l)
22              return l<foo.l;
23          return i<foo.i;
24      }
25  };
26
27  int n, q;
28  int h[N], lca[N][M];
29  vector<int>g[N];
30  int dl[N], dr[N], di[2*N], cur;
31
32  void dfs(int u, int p)
33  {
34      dl[u]=++cur;
35      di[cur]=u;
36      lca[u][0]=p;
37      for(int i=1; i<M; i++)
38          lca[u][i]=lca[ lca[u][i-1] ][i-1];
39      for(int i=0; i<g[u].size(); i++)
40      {
41          int v=g[u][i];
42          if(v==p)
43              continue;
44          h[v]=h[u]+1;
45          dfs(v, u);
46      }
47      dr[u]=++cur;
48      di[cur]=u;
49  }
50
51  inline int getLca(int u, int v)
52  {
53      if(h[u]>h[v])
54          swap(u, v);
55      for(int i=M-1; i>=0; i--)
56          if(h[v]-(1<<i)>=h[u])
57              v=lca[v][i];
58      if(u==v)
59          return u;
60      for(int i=M-1; i>=0; i--)
61      {
62          if(lca[u][i]!=lca[v][i])
63          {
64              u=lca[u][i];
65              v=lca[v][i];
66          }
67      }
68      return lca[u][0];
69  }
70
71  map<string,int>remap;
72  int data[N], ans[N], vis[N], freq[N], cnt;
73  inline void update(int u)
74  {
```

```
75          int x=data[u];
76          if(vis[u] && (−−freq[ data[u] ]==0))
77              cnt−−;
78          else if(!vis[u] && (freq[ data[u] ]++==0))
79              cnt++;
80          vis[u]^=1;
81      }
82
83      int main()
84      {
85          scanf("%d_%d", &n, &q);
86          for(int i=1; i<=n; i++)
87          {
88              char temp[25];
89              scanf("%s", temp);
90              string temp2=string(temp);
91              if(!remap.count(temp2))
92                  remap[temp2]=remap.size();
93              data[i]=remap[temp2];
94          }
95          for(int i=1; i<n; i++)
96          {
97              int u, v;
98              scanf("%d_%d", &u, &v);
99              g[u].pb(v);
100             g[v].pb(u);
101         }
102         dfs(1, 0);
103
104         vector<mo>query;
105         for(int i=0; i<q; i++)
106         {
107             int u, v;
108             scanf("%d_%d", &u, &v);
109             int lc=getLca(u, v);
110             if(dl[u]>dl[v])
111                 swap(u, v);
112             query.pb(mo(u==lc?dl[u]:dr[u], dl[v], lc, i));
113         }
114         sort(ALL(query));
115
116         int l=query[0].l, r=query[0].l−1;
117         cnt=0;
118         for(int i=0; i<q; i++)
119         {
120             int li=query[i].l;
121             int ri=query[i].r;
122             int lc=query[i].lc;
123             int ii=query[i].i;
124             while(l>li)
125                 update(di[−−l]);
126             while(r<ri)
127                 update(di[++r]);
128             while(l<li)
129                 update(di[l++]);
130             while(r>ri)
131                 update(di[r−−]);
132
133             int u=di[l], v=di[r];
134             if(lc!=u && lc!=v)
135                 update(lc);
136             ans[ii]=cnt;
137             if(lc!=u && lc!=v)
138                 update(lc);
139         }
140         for(int i=0; i<q; i++)
141             printf("%d\n", ans[i]);
142         return 0;
143     }
```

Chapter 2

Graph

2.1 Components

2.1.1 Bridges

```
1   const int MAXN = ...;
2   vector<int> g[MAXN];
3   bool used[MAXN];
4   int timer, tin[MAXN], fup[MAXN];
5
6   void dfs (int v, int p = -1) {
7       used[v] = true;
8       tin[v] = fup[v] = timer++;
9       for (size_t i=0; i<g[v].size(); ++i) {
10          int to = g[v][i];
11          if (to == p)  continue;
12          if (used[to])
13              fup[v] = min (fup[v], tin[to]);
14          else {
15              dfs (to, v);
16              fup[v] = min (fup[v], fup[to]);
17              if (fup[to] > tin[v])
18                  IS_BRIDGE(v,to);
19          }
20      }
21  }
22
23  void find_bridges() {
24      timer = 0;
25      for (int i=0; i<n; ++i)
26          used[i] = false;
27      for (int i=0; i<n; ++i)
28          if (!used[i])
29              dfs (i);
30  }
```

2.1.2 Cut Points

```cpp
vector<int> g[MAXN];
bool used[MAXN];
int timer, tin[MAXN], fup[MAXN];

void dfs (int v, int p = -1) {
    used[v] = true;
    tin[v] = fup[v] = timer++;
    int children = 0;
    for (size_t i=0; i<g[v].size(); ++i) {
        int to = g[v][i];
        if (to == p)  continue;
        if (used[to])
            fup[v] = min (fup[v], tin[to]);
        else {
            dfs (to, v);
            fup[v] = min (fup[v], fup[to]);
            if (fup[to] >= tin[v] && p != -1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if (p == -1 && children > 1)
        IS_CUTPOINT(v);
}

int main() {
    int n;
    ... ÑĞÑĆĐţĐiĐÿĐţ n Đÿ g ...

    timer = 0;
    for (int i=0; i<n; ++i)
        used[i] = false;
    dfs (0);
}
```

2.1.3 Strongly Connected Components

Tarjan

```
1   class graph
2   {
3       const static int MN=1e+5;
4   public:
5       vector<int>data[MN], aux;
6       bool vis[MN];
7       int grp[MN];
8       int dfs_num[MN], dfs_low[MN];
9       int dfs_cnt, numSCC;
10
11      graph(){};
12      void clear()
13      {
14          for(int i=0; i<MN; i++)
15          {
16              data[i].clear();
17              dfs_num[i]=-1;
18              dfs_low[i]=0;
19              vis[i]=false;
20          }
21          aux.clear();
22          dfs_cnt=numSCC=0;
23      }
24      void add_edge(int u, int v)
25      {
26          data[u].push_back(v);
27      }
28      void tarjanSCC(int u)
29      {
30          dfs_num[u]=dfs_low[u]=dfs_cnt++;
31          aux.push_back(u);
32          vis[u]=true;
33
34          for(int i=0; i<data[u].size(); i++)
35          {
36              int v=data[u][i];
37              if(dfs_num[v]==-1)
38                  tarjanSCC(v);
39              if(vis[v])
40                  dfs_low[v]=min(dfs_low[v], dfs_low[u]);
41          }
42
43          if(dfs_num[u]==dfs_low[u])
44          {
45              while(1)
46              {
47                  int v=aux.back();
48                  aux.pop_back();
49                  vis[v]=false;
50                  grp[v]=numSCC;
51                  if(u==v)
52                      break;
53              }
54              numSCC++;
55          }
56      }
57  };
```

2.1.4 Semi-Strongly Connected Components

2.2 Single Source Shortest Path

2.2.1 Dijkstra

2.2.2 Bellmanford

```
1   class node
2   {
3   public:
4       int x, y, d;
5       node(){};
6       node(int _x, int _y, int _d)
7       {
8           x=_x, y=_y, d=_d;
9       }
10  };
11
12  int n, v;
13  vector<node>graph;
14  int dist[1035];
15  bool bellmanford(int s)
16  {
17      memset(dist, INF, sizeof(dist));
18      dist[s]=0;
19      for(int i=0; i<n-1; i++)
20      {
21          for(int j=0; j<graph.size(); j++)
22          {
23              int x=graph[j].x;
24              int y=graph[j].y;
25              int d=graph[j].d;
26              if(dist[y]>dist[x]+d)
27                  dist[y]=dist[x]+d;
28          }
29      }
30
31      for(int i=0; i<graph.size(); i++)
32      {
33          int x=graph[i].x;
34          int y=graph[i].y;
35          int d=graph[i].d;
36          if(dist[x]<INF && dist[y]>dist[x]+d)
37              return true;
38      }
39      return false;
40  }
```

2.3 All Pairs Shortest Path

2.3.1 Floyd Warshall

2.4 Minimum Spannig Tree

2.4.1 Kruskal

2.4.2 Prim

2.5 Flow

2.5.1 Maximum Bipartite Matching

```
 1  const int MN=1e+3;
 2  vector<int>g[MN];
 3  int match[MN], rmatch[MN], vis[MN];
 4  int findmatch(int u)
 5  {
 6      if(vis[u])
 7          return 0;
 8      vis[u]=true;
 9      for(int v:g[u])
10      {
11          if(match[v]==-1 || findmatch(match[v]))
12          {
13              match[v]=u;
14              rmatch[u]=v;
15              return 1;
16          }
17      }
18      return 0;
19  }
20
21  int maxMatch(int n)
22  {
23      int ret=0;
24      memset(match, -1, sizeof(match));
25      for(int i=0; i<n; i++)
26      {
27          memset(vis, false, sizeof(vis));
28          ret+=findmatch(i);
29      }
30      return ret;
31  }
```

2.5.2 Maximum Flow

Dinic

```
1   class graph
2   {
3       const static int N=100000;
4   public:
5       vector< pair<int,int> >edge;
6       vector<int>adj[N];
7       int ptr[N];
8       int dist[N];
9
10      graph(){};
11      void clear()
12      {
13          for(int i=0; i<N; i++)
14              adj[i].clear();
15          edge.clear();
16      }
17      void add_edge(int u, int v, int c)
18      {
19          adj[u].push_back(edge.size());
20          edge.push_back(mp(v, c));
21          adj[v].push_back(edge.size());
22          edge.push_back(mp(u, 0)); //(u, c) if is non-directed
23      }
24      bool dinic_bfs(int s, int t)
25      {
26          memset(dist, -1, sizeof(dist));
27          dist[s]=0;
28
29          queue<int>bfs;
30          bfs.push(s);
31          while(!bfs.empty() && dist[t]==-1)
32          {
33              int u=bfs.front();
34              bfs.pop();
35              for(int i=0; i<adj[u].size(); i++)
36              {
37                  int idx=adj[u][i];
38                  int v=edge[idx].F;
39
40                  if(dist[v]==-1 && edge[idx].S>0)
41                  {
42                      dist[v]=dist[u]+1;
43                      bfs.push(v);
44                  }
45              }
46          }
47          return dist[t]!=-1;
48      }
49      int dinic_dfs(int u, int t, int flow)
50      {
51          if(u==t)
52              return flow;
53          for(int &i=ptr[u]; i<adj[u].size(); i++)
54          {
55              int idx=adj[u][i];
56              int v=edge[idx].F;
57              if(dist[v]==dist[u]+1 && edge[idx].S>0)
58              {
59                  int cf=dinic_dfs(v, t, min(flow, edge[idx].S));
60                  if(cf>0)
61                  {
62                      edge[idx].S-=cf;
63                      edge[idx^1].S+=cf;
64                      return cf;
65                  }
66              }
67          }
68          return 0;
69      }
70      int maxflow(int s, int t)
71      {
72          int ret=0;
```

```
73          while(dinic_bfs(s, t))
74          {
75              memset(ptr, 0, sizeof(ptr));
76              int cf=dinic_dfs(s, t, INF);
77              if(cf==0)
78                  break;
79              ret+=cf;
80          }
81          return ret;
82      }
83  };
```

2.5.3   Minimum Cost Maximum Flow

Undirected graph:
$u \rightarrow uu(flow, 0)$
$uu \rightarrow vv(flow, cost)$
$vv \rightarrow v(flow, 0)$
$v \rightarrow uu(flow, 0)$
$vv \rightarrow u(flow, 0)$

Dijkstra

```cpp
typedef int FTYPE; //type of flow
typedef int CTYPE; //type of cost
typedef pair<FTYPE,CTYPE>pfc;
const CTYPE CINF=INF;
const FTYPE FINF=INF;

void operator+=(pfc &p1, pfc &p2)
{
    p1.F+=p2.F;
    p1.S+=p2.S;
}

class graph
{
    const static int MN=1e+4;
public:
    int n;
    FTYPE flow[MN];
    CTYPE dist[MN], pot[MN];
    int prev[MN], eidx[MN];

    struct Edge
    {
        int to;
        FTYPE cap;
        CTYPE cost;
        Edge(){};
        Edge(int _to, FTYPE _cap, CTYPE _cost)
        {
            to=_to;
            cap=_cap;
            cost=_cost;
        }//
    };
    struct node
    {
        int u;
        CTYPE d;
        node(){};
        node(int _u, CTYPE _d)
        {
            u=_u;
            d=_d;
        }
        bool operator <(const node &foo) const
        {
            return d>foo.d;
        }
    };
    graph(){};
    vector<int>adj[MN];
    vector<Edge>edge;
    inline void set(int _n)
    {
        n=_n;
    }
    inline void reset()
    {
        for(int i=0; i<MN; i++)
            adj[i].clear();
        edge.clear();
    }
```

```cpp
63        inline void add_edge(int u, int v, FTYPE c, FTYPE cst)
64        {
65            adj[u].push_back(edge.size());
66            edge.push_back(Edge(v, c, cst));
67            adj[v].push_back(edge.size());
68            edge.push_back(Edge(u, 0, -cst));
69        }
70
71        pfc dijkstra(int s, int t)
72        {
73            for(register int i=0; i<n; i++)
74                dist[i]=CINF;
75            dist[s]=0;
76            flow[s]=FINF;
77            priority_queue<node>heap;
78            heap.push(node(s, 0));
79            while(!heap.empty())
80            {
81                int u=heap.top().u;
82                CTYPE d=heap.top().d;
83                heap.pop();
84                if(d>dist[u])
85                    continue;
86                for(int i=0; i<adj[u].size(); i++)
87                {
88                    int idx=adj[u][i];
89                    int v=edge[idx].to;
90                    CTYPE w=edge[idx].cost;
91                    if(!edge[idx].cap || dist[v]<=d+w+pot[u]-pot[v])
92                        continue;
93                    if(d+w<dist[v])
94                    {
95                        dist[v]=d+w;
96                        prev[v]=u;
97                        eidx[v]=idx;
98                        flow[v]=min(flow[u], edge[idx].cap);
99                        heap.push(node(v, d+w));
100                   }
101               }
102           }
103           if(dist[t]==CINF)
104               return mp(FINF, CINF);
105           pfc ret=mp(flow[t], 0);
106           for(int u=t; u!=s; u=prev[u])
107           {
108               int idx=eidx[u];
109               edge[idx].cap-=flow[t];
110               edge[idx^1].cap+=flow[t];
111               ret.second+=flow[t]*edge[idx].cost;
112           }
113           return ret;
114       }
115
116       inline pfc mfmc(int s, int t)
117       {
118           pfc ret=mp(0, 0);
119           pfc got;
120           while((got=dijkstra(s, t)).first!=FINF)
121               ret+=got;
122           return ret;
123       }
124   };
```

Bellmanford

2.5.4 Minimum Cut

Stoer Wagner

```
1    int stoer_wagner(int n)
2    {
3        int ret=INF;
4        for(int i=0; i<n; i++)
5            v[i]=i;
6
7        while(n>1)
8        {
9            a[ v[0] ]=true;
10           for(int i=1; i<n; i++)
11           {
12               a[ v[i] ]=false;
13               na[i-1]=i;
14               w[i]=graph[ v[0] ][ v[i] ];
15           }
16
17           int prev=v[0];
18           for(int i=1; i<n; i++)
19           {
20               int zj=-1;
21               for(int j=1; j<n; j++)
22               {
23                   if(!a[ v[j] ] && (zj<0 || w[j]>w[zj]))
24                       zj=j;
25               }
26
27               a[ v[zj] ]=true;
28
29               if(i==n-1)
30               {
31                   ret=min(ret, w[zj]);
32
33                   for(int j=0; j<n; j++)
34                       graph[ v[j] ][prev]=graph[prev][ v[j] ]+=graph[ v[zj] ][ v[j] ];
35                   v[zj]=v[--n];
36                   break;
37               }
38               prev=v[zj];
39
40               for(int j=1; j<n; j++)
41                   if(!a[ v[j] ])
42                       w[j]+=graph[ v[zj] ][ v[j] ];
43           }
44       }
45       return ret;
46   }
```

2.6 Tree

2.6.1 Lowest Common Ancestor

```
1    const int MN=1e+5+35;
2    const int LMN=1+log2(MN);
3    vector<int>graph[MN];
4    int LVL[MN];
5    int T[MN];
6    int dp[MN][LMN];
7    bool vis[MN];
8
9    void dfs(int u, int f, int d)
10   {
11       vis[u]=true;
12       LVL[x]=d;
13       dp[x][0]=f;
14       for(int i=1; i<LMN; i++)
15           dp[x][i]=dp[ dp[x][i−1] ][i−1];
16
17       vis[x]=true;
18       for(int i=0; i<graph[x].size(); i++)
19       {
20           int v=graph[x][u];
21           if(!vis[v])
22               dfs(v, x, d+1);
23       }
24   }
25
26   inline int lca(int u, int v)
27   {
28       if(LVL[u]>LVL[v])
29           swap(u, v);
30       for(int i=LMN−1; i>=0; i−−)
31           if(LVL[v]−(1<<i)>=LVL[u])
32               v=dp[v][i];
33       if(u==v)
34           return u;
35       for(int i=LMN−1; i>=0; i−−)
36       {
37           if(dp[u][i]!=dp[v][i])
38           {
39               u=dp[u][i];
40               v=dp[v][i];
41           }
42       }
43       return dp[u][0];
44   }
```

2.6.2 Centroid Decomposition

```
1   class graph
2   {
3       const static int N=1e+5;
4       const static int LN=log2(N)+1;
5   public:
6       vector<int>g[N];
7       int h[N], lca[N][LN];
8
9       int sz[N];
10      int cg[N], gsz, dlt[N];
11      graph(){};
12      inline void addEdge(int u, int v)
13      {
14          g[u].pb(v);
15          g[v].pb(u);
16      }
17      void buildLca(int u, int f)
18      {
19          lca[u][0]=f;
20          for(int i=1; i<LN; i++)
21              lca[u][i]=lca[ lca[u][i-1] ][i-1];
22          for(int v:g[u])
23          {
24              if(v==f)
25                  continue;
26              h[v]=h[u]+1;
27              buildLca(v, u);
28          }
29      }
30      inline int getLca(int u, int v)
31      {
32          if(h[u]>h[v])
33              swap(u, v);
34          for(int i=LN-1; i>=0; i--)
35              if(h[v]-(1<<i)>=h[u])
36                  v=lca[v][i];
37          if(u==v)
38              return u;
39          for(int i=LN-1; i>=0; i--)
40          {
41              if(lca[u][i]!=lca[v][i])
42              {
43                  u=lca[u][i];
44                  v=lca[v][i];
45              }
46          }
47          return lca[u][0];
48      }
49      inline int getDist(int u, int v)
50      {
51          return h[u]+h[v]-2*h[getLca(u, v)];
52      }
53      void buildSz(int u, int f)
54      {
55          gsz++;
56          sz[u]=1;
57          for(int v:g[u])
58          {
59              if(v==f || dlt[v])
60                  continue;
61              buildSz(v, u);
62              sz[u]+=sz[v];
63          }
64      }
65      int findCentroid(int u, int f)
66      {
67          for(int v:g[u])
68          {
69              if(v==f || dlt[v])
70                  continue;
71              if(sz[v]*2>=gsz)
72                  return findCentroid(v, u);
73          }
74          return u;
```

```
75          }
76      inline void buildCentroid(int u, int f)
77      {
78          gsz=0;
79          buildSz(u, u);
80          int c=findCentroid(u, u);
81          cg[c]=(u==f)?c:f;
82          dlt[c]=1;
83          for(int v:g[c])
84          {
85              if(v==c || dlt[v])
86                  continue;
87              buildCentroid(v, c);
88          }
89      }
90  };
```

2.6.3 Heavy Light Decomposition on Edges

```cpp
class segtree
{
    const static int N=1e+5;
public:
    int tr[4*N];
    segtree(){};
    void reset()
    {
        memset(tr, 0, sizeof(tr));
    }
    void update(int no, int l, int r, int i, int val)
    {
        if(r<i || l>i)
            return;
        if(l>=i && r<=i)
        {
            tr[no]=val;
            return;
        }
        int nxt=(no<<1);
        int mid=(l+r)>>1;
        update(nxt, l, mid, i, val);
        update(nxt+1, mid+1, r, i, val);
        tr[no]=tr[nxt]+tr[nxt+1];
    }
    int query(int no, int l, int r, int i, int j)
    {
        if(r<i || l>j)
            return 0;
        if(l>=i && r<=j)
            return tr[no];
        int nxt=(no<<1);
        int mid=(l+r)>>1;
        return query(nxt, l, mid, i, j)+query(nxt+1, mid+1, r, i, j);
    }
};

const int N=1e+5;
const int M=log2(N)+1;
int n;
segtree tr;
vector< pair<int,int> >g[N];
int lca[N][M];
int h[N], trSz[N];

//in - use X[], Y[] in case
//of edge weights
int X[N], Y[N], W[N];

//hld
int chainInd[N], chainSize[N], chainHead[N], chainPos[N], chainNo, posInBase[N];
int ptr;

void dfs(int u, int l)
{
    trSz[u]=1;
    lca[u][0]=l;
    for(int i=1; i<M; i++)
        lca[u][i]=lca[ lca[u][i-1] ][i-1];
    for(int i=0; i<g[u].size(); i++)
    {
        int v=g[u][i].first;
        if(v==l)
            continue;
        h[v]=h[u]+1;
        dfs(v, u);
        trSz[u]+=trSz[v];
    }
}

inline int getLca(int u, int v)
{
    if(h[u]>h[v])
        swap(u, v);
```

```
75        for(int i=M-1; i>=0; i--)
76              if(h[v]-(1<<i)>=h[u])
77                    v=lca[v][i];
78        if(u==v)
79              return u;
80        for(int i=M-1; i>=0; i--)
81        {
82              if(lca[u][i]!=lca[v][i])
83              {
84                    u=lca[u][i];
85                    v=lca[v][i];
86              }
87        }
88        return lca[u][0];
89    }
90
91    //dont use 'c' if the weight is on the vertex
92    //instead of the edge
93    inline void hld(int u, int l, int c)
94    {
95        if(chainHead[chainNo]==-1)
96              chainHead[chainNo]=u;
97        chainInd[u]=chainNo;
98        chainPos[u]=chainSize[chainNo]++;
99        tr.update(1, 0, n, ptr, c);
100       posInBase[u]=ptr++;
101
102       int msf, idx;
103       msf=idx=-1;
104       for(int i=0; i<g[u].size(); i++)
105       {
106             int v=g[u][i].first;
107             if(v==l)
108                   continue;
109             if(trSz[v]>msf)
110             {
111                   msf=trSz[v];
112                   idx=i;
113             }
114       }
115       if(idx>=0)
116             hld(g[u][idx].first, u, g[u][idx].second);
117       for(int i=0; i<g[u].size(); i++)
118       {
119             if(i==idx)
120                   continue;
121             int v=g[u][i].first;
122             int w=g[u][i].second;
123             if(v==l)
124                   continue;
125             chainNo++;
126             hld(v, u, w);
127       }
128   }
129
130   inline int query_up(int u, int v)
131   {
132       int uchain=chainInd[u];
133       int vchain=chainInd[v];
134       int ret=0;
135       while(true)
136       {
137             uchain=chainInd[u];
138             if(uchain==vchain)
139             {
140                   ret+=tr.query(1, 0, n, posInBase[v]+1, posInBase[u]);
141                   break;
142             }
143             int head=chainHead[uchain];
144             ret+=tr.query(1, 0, n, posInBase[head],posInBase[u]);
145             u=head;
146             u=lca[u][0];
147       }
148       return ret;
149   }
150
151   //returns sum of all edges weights
```

```
152    //from 'u' to 'v'
153    inline int query(int u, int v)
154    {
155        if(u==v)
156            return 0;
157        int l=getLca(u, v);
158        return query_up(u, l)+query_up(v, l);
159    }
160
161    //set and edge to value 'val'
162    inline void update(int u, int val)
163    {
164        int x=X[u], y=Y[u];
165        if(lca[x][0]==y)
166            tr.update(1, 0, n, posInBase[x], val);
167        else
168            tr.update(1, 0, n, posInBase[y], val);
169    }
170
171    void clearHld()
172    {
173        //tr.reset();
174        for(int i=0; i<=n; i++)
175        {
176            g[i].clear();
177            chainHead[i]=-1;
178            chainSize[i]=0;
179        }
180        ptr=1;
181        chainNo=0;
182    }
183
184    int main()
185    {
186        scanf("%d", &n);
187        clearHld();
188        for(int i=1; i<n; i++)
189        {
190            scanf("%d %d %d", &X[i], &Y[i], &W[i]);
191            g[ X[i] ].push_back({Y[i], W[i]});
192            g[ Y[i] ].push_back({X[i], W[i]});
193        }
194        dfs(1, 0);
195        hld(1, 0, 0);
196        int q;
197        scanf("%d", &q);
198        while(q--)
199        {
200            int o, x, y;
201            scanf("%d %d %d", &o, &x, &y);
202            if(o==1)
203                printf("%d\n", query(x, y));
204            else
205                update(x, y);
206        }
207        return 0;
208    }
```

2.6.4   Heavy Light Decomposition on Vertex

2.6.5 Centroid Decomposition & All-Pairs Distance

```cpp
/// David Mateus Batista <david.batista3010@gmail.com>
/// Computer Science - Federal University of Itajuba - Brazil

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef unsigned long long ull;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<ll,ll> pll;

#define INF 0x3F3F3F3F
#define LINF 0x3F3F3F3F3F3F3F3FLL
#define DINF (double)1e+30
#define EPS (double)1e-9
#define PI (double)acos(-1.0)
#define RAD(x) (double)(x*PI)/180.0
#define PCT(x,y) (double)x*100.0/y

#define pb push_back
#define mp make_pair
#define pq priority_queue
#define F first
#define S second

#define D(x) x&(-x)
#define SZ(x) (int)x.size()
#define ALL(x) x.begin(),x.end()
#define SET(a,b) memset(a, b, sizeof(a))

#define gcd(x,y) __gcd(x, y)
#define lcm(x,y) (x/gcd(x,y))*y

#define bitcnt(x) __builtin_popcountll(x)
#define lbit(x) 63-__builtin_clzll(x)
#define zerosbitll(x) __builtin_ctzll(x)
#define zerosbit(x) __builtin_ctz(x)

enum {North, East, South, West};
//{0, 1, 2, 3}
//{Up, Right, Down, Left}

int mi[] = {-1, 0, 1, 0, -1, 1, 1, -1};
int mj[] = {0, 1, 0, -1, 1, 1, -1, -1};

class graph
{
    const static int N=1e+5+35;
    const static int LN=log2(N)+1;
public:
    //tree
    vector<int>g[N];
    int h[N], lca[N][LN];
    //centroid
    vector<int>cgt[N];
    int sz[N];
    int cg[N], gsz, dlt[N];
    //updates & querys
    ll sum[N], cnt[N];
    int vis[N], idx[N];
    vector<ll>psum[N], pcnt[N];

    graph(){};
    inline void addEdge(int u, int v)
    {
        g[u].pb(v);
        g[v].pb(u);
    }
    void buildLca(int u, int f)
    {
        lca[u][0]=f;
        for(int i=1; i<LN; i++)
```

```
 75                 lca[u][i]=lca[ lca[u][i-1] ][i-1];
 76             for(int v:g[u])
 77             {
 78                 if(v==f)
 79                     continue;
 80                 h[v]=h[u]+1;
 81                 buildLca(v, u);
 82             }
 83         }
 84         inline int getLca(int u, int v)
 85         {
 86             if(h[u]>h[v])
 87                 swap(u, v);
 88             for(int i=LN-1; i>=0; i--)
 89                 if(h[v]-(1<<i)>=h[u])
 90                     v=lca[v][i];
 91             if(u==v)
 92                 return u;
 93             for(int i=LN-1; i>=0; i--)
 94             {
 95                 if(lca[u][i]!=lca[v][i])
 96                 {
 97                     u=lca[u][i];
 98                     v=lca[v][i];
 99                 }
100             }
101             return lca[u][0];
102         }
103         inline int getDist(int u, int v)
104         {
105             return h[u]+h[v]-2*h[getLca(u, v)];
106         }
107         void buildSz(int u, int f)
108         {
109             gsz++;
110             sz[u]=1;
111             for(int v:g[u])
112             {
113                 if(v==f || dlt[v])
114                     continue;
115                 buildSz(v, u);
116                 sz[u]+=sz[v];
117             }
118         }
119         int findCentroid(int u, int f)
120         {
121             for(int v:g[u])
122             {
123                 if(v==f || dlt[v])
124                     continue;
125                 if(sz[v]*2>=gsz)
126                     return findCentroid(v, u);
127             }
128             return u;
129         }
130         inline void buildCentroid(int u, int f)
131         {
132             gsz=0;
133             buildSz(u, u);
134             int c=findCentroid(u, u);
135             cg[c]=(u==f)?c:f;
136
137             if(c!=cg[c])
138             {
139                 idx[c]=cgt[f].size();
140                 cgt[f].pb(c);
141                 psum[f].pb(0);
142                 pcnt[f].pb(0);
143             }
144
145             dlt[c]=1;
146             for(int v:g[c])
147             {
148                 if(v==c || dlt[v])
149                     continue;
150                 buildCentroid(v, c);
151             }
```

```
152            }
153
154        void update(int u)
155        {
156            int v=u, l=-1;
157            int k=(vis[u]==0)?1:-1;
158            while(1)
159            {
160                ll d=k*getDist(u, v);
161                sum[v]+=d;
162                if(l!=-1)
163                {
164                    psum[v][idx[l]]+=d;
165                    pcnt[v][idx[l]]+=k;
166                }
167                cnt[v]+=k;
168                if(v==cg[v])
169                    break;
170                l=v;
171                v=cg[v];
172            }
173            vis[u]^=1;
174        }
175        ll query(int u)
176        {
177            ll ret=0;
178            int v=u, l=-1;
179            while(1)
180            {
181                ll d=getDist(u, v);
182                ret+=sum[v]+cnt[v]*d;
183                if(l!=-1)
184                {
185                    ret-=psum[v][idx[l]];
186                    ret-=pcnt[v][idx[l]]*d;
187                }
188                if(v==cg[v])
189                    break;
190                l=v;
191                v=cg[v];
192            }
193            return ret;
194        }
195    };
196
197    int n;
198    int haha[10000];
199    graph data;
200
201    void update(int x)
202    {
203        data.update(x);
204        haha[x]^=1;
205    }
206
207    ll naive(int x)
208    {
209        ll ret=0;
210        for(int i=1; i<=n; i++)
211        {
212            ret+=haha[i]*data.getDist(x, i);
213        }
214        return ret;
215    }
216
217    int main()
218    {
219        scanf("%d", &n);
220        for(int i=1; i<n; i++)
221        {
222            int x, y;
223            scanf("%d_%d", &x, &y);
224            x++, y++;
225            data.addEdge(x, y);
226        }
227        data.buildLca(1, 1);
228        data.buildCentroid(1, 1);
```

47

```
229
230        int q;
231        scanf("%d", &q);
232        while(q--)
233        {
234            int x, y;
235            scanf("%d_%d", &x, &y);
236            y++;
237            if(x==0)
238                update(y);
239            else
240                printf("%lld\n", data.query(y));
241        }
242        return 0;
243    }
```

2.6.6   All-Pairs Distance & FFT

2.7 MISC

2.7.1 2-SAT

```
1   const int MN=2*1e+5+35;
2   int n, m;
3   vector<int>g[MN], rg[MN];
4   bool vis[MN];
5   int grp[MN];
6   stack<int>kos;
7   void dfsOne(int u)
8   {
9       vis[u]=true;
10      for(int i=0; i<g[u].size(); i++)
11      {
12          int v=g[u][i];
13          if(!vis[v])
14              dfsOne(v);
15      }
16      kos.push(u);
17  }
18
19  void dfsTwo(int u, int k)
20  {
21      vis[u]=true;
22      for(int i=0; i<rg[u].size(); i++)
23      {
24          int v=rg[u][i];
25          if(!vis[v])
26              dfsTwo(v, k);
27      }
28      grp[u]=k;
29  }
30
31  void kosaraju()
32  {
33      for(int i=0; i<2*n; i++)
34          if(!vis[i])
35              dfsOne(i);
36      int k=1;
37      SET(vis, false);
38      while(!kos.empty())
39      {
40          int u=kos.top();
41          kos.pop();
42          if(!vis[u])
43              dfsTwo(u, k++);
44      }
45  }
46
47  bool _2sat()
48  {
49      kosaraju();
50      for(int i=0; i<n; i++)
51          if(grp[i]==grp[i+n])
52              return false;
53      return true;
54  }
```

2.7.2 Games on Random Graphs

```
1   struct state {
2       char p, t;
3       bool pstep;
4   };};
5
6   vector <state> g [100] [100] [2];
7   // 1 = policeman coords; 2 = thief coords; 3 = 1 if policeman's step or 0 if thief's.
8   bool win [100] [100] [2];
9   bool loose [100] [100] [2];
10  bool used [100] [100] [2];
11  int degree [100] [100] [2];
12
13  void dfs (char p, char t, bool pstep) {
14      used [p] [t] [pstep] = true;
15      for (vector <state> :: iterator i = g [p] [t] [pstep] .begin (); i! = g [p] [t] [pstep] .end ();
            ++ i)
16          if (! used [i-> p] [i-> t] [i-> pstep]) {
17              if (loose [p] [t] [pstep])
18                  win [i-> p] [i-> t] [i-> pstep] = true;
19              else if (--degree [i-> p] [i-> t] [i-> pstep] == 0)
20                  loose [i-> p] [i-> t] [i-> pstep] = true;
21              else
22                  continue;
23              dfs (i-> p, i-> t, i-> pstep);
24          }
25  }
26
27
28  int main () {
29
30      int n, m;
31      cin >> n >> m;
32      vector <string> a (n);
33      for (int i = 0; i <n; ++ i)
34          cin >> a [i];
35
36      for (int p = 0; p <n * m; ++ p)
37          for (int t = 0; t <n * m; ++ t)
38              for (char pstep = 0; pstep <= 1; ++ pstep) {
39                  int px = p / m, py = p% m, tx = t / m, ty = t% m;
40                  if (a [px] [py] == '*' || a [tx] [ty] == '*') continue;
41
42                  bool & wwin = win [p] [t] [pstep];
43                  bool & lloose = loose [p] [t] [pstep];
44                  if (pstep)
45                      wwin = px == tx && py == ty, lloose =! wwin && a [tx] [ty] == 'E';
46                  else
47                      wwin = a [tx] [ty] == 'E', lloose =! wwin && px == tx && py == ty;
48                  if (wwin || lloose) continue;
49
50                  state st = {p, t,! pstep};
51                  g [p] [t] [pstep] .push_back (st);
52                  st.pstep = pstep! = 0;
53                  degree [p] [t] [pstep] = 1;
54
55                  const int dx [] = {-1, 0, 1, 0, -1, -1, 1, 1};
56                  const int dy [] = {0, 1, 0, -1, -1, 1, -1, 1};
57                  for (int d = 0; d <(pstep? 8: 4); ++ d) {
58                      int ppx = px, ppy = py, ttx = tx, tty = ty;
59                      if (pstep)
60                          ppx + = dx [d], ppy + = dy [d];
61                      else
62                          ttx + = dx [d], tty + = dy [d];
63                      if (ppx> = 0 && ppx <n && ppy> = 0 && ppy <m && a [ppx] [ppy]! = '*' &&
64                          ttx> = 0 && ttx <n && tty> = 0 && tty <m && a [ttx] [tty]! = '*')
65                      {
66                          g [ppx * m + ppy] [ttx * m + tty] [! pstep] .push_back (st);
67                          ++ degree [p] [t] [pstep];
68                      }
69                  }
70              }
71
72      for (int p = 0; p <n * m; ++ p)
73          for (int t = 0; t <n * m; ++ t)
```

```
74              for (char pstep = 0; pstep <= 1; ++ pstep)
75                  if ((win [p] [t] [pstep] || loose [p] [t] [pstep]) &&! used [p] [t] [pstep])
76                      dfs (p, t, pstep! = 0);
77
78      int p_st, t_st;
79      for (int i = 0; i <n; ++ i)
80          for (int j = 0; j <m; ++ j)
81              if (a [i] [j] == 'C')
82                  p_st = i * m + j;
83              else if (a [i] [j] == 'T')
84                  t_st = i * m + j;
85
86      cout << (win [p_st] [t_st] [true]? "WIN": loose [p_st] [t_st] [true]? "LOSS": "DRAW");
87
88  }
```

Chapter 3

Dynamic Programming

3.1 Optimizations

3.1.1 Divide and Counquer - Example 1

Memory can be optimized by using dp[2][N] & dp[k%2][m] + dp[(k-1)%2][m]

```
1    typedef long long ll;
2    const int MN=1e+4+35;
3    const int MN2=535;
4    int p, a;
5    ll data[MN];
6    inline ll getValue(int l, int r)
7    {
8        return (r-l+1)*(data[r]-data[l-1]);
9    }
10   ll dp[MN2][MN];
11   inline void solve(int k, int l, int r, int L, int R)
12   {
13       if(l>r)
14           return;
15       int m=(l+r)/2;
16       int s=L;
17       dp[k][m]=LINF;
18       for(int i=max(m, L); i<=R; i++)
19       {
20           if(dp[k][m]>dp[k-1][i+1]+getValue(m+1, i+1))
21           {
22               dp[k][m]=dp[k-1][i+1]+getValue(m+1, i+1);
23               s=i;
24           }
25       }
26       solve(k, l, m-1, L, s);
27       solve(k, m+1, r, s, R);
28   }
29   int main()
30   {
31       scanf("%d_%d", &p, &a);
32       for(int i=1; i<=p; i++)
33       {
34           ll x;
35           scanf("%lld", &x);
36           data[i]=data[i-1]+x;
37       }
38       for(int i=0; i<=p; i++)
39           dp[0][i]=LINF;
40       for(int i=0; i<=a; i++)
41           dp[i][p]=0;
42       for(int i=1; i<=a; i++)
43           solve(i, 0, p-1, 0, p-1);
44       printf("%lld\n", dp[a][0]);
45   }
```

3.1.2 Divide and Counquer - Example 2

```
1    typedef long long ll;
2    const int MN=6005;
3    ll v[MN];
4    ll dp[MN][MN];
5    ll c[MN][MN];
6
7    ll sum[MN];
8    ll multisum[MN];
9
10   void solve(int k, int l, int r, int L, int R)
11   {
12       if(l>r)
13           return;
14       int m=(l+r)/2;
15       int s=-1;
16       dp[k][m]=LINF;
17       for(int i=max(m, L); i<=R; i++)
18       {
19           if(dp[k][m]>dp[k-1][i+1]+c[m][i])
20           {
21               dp[k][m]=dp[k-1][i+1]+c[m][i];
22               s=i;
23           }
24       }
25       solve(k, l, m-1, L, s);
26       solve(k, m+1, r, s, R);
27   }
28
29   ll dist(int l, int r, int mid)
30   {
31       ll s1=sum[mid]-sum[l];
32       ll p1=multisum[mid]-multisum[l];
33       ll s2=sum[r+1]-sum[mid+1];
34       ll p2=multisum[r+1]-multisum[mid+1];
35       return (s1*mid-p1)+(p2-s2*mid);
36   }
37
38   int main()
39   {
40       int n;
41       ll b, k;
42       scanf("%d_%lld_%lld", &n, &b, &k);
43       for(int i=0; i<n; i++)
44       {
45           scanf("%lld", &v[i]);
46           sum[i+1]=sum[i]+v[i];
47           multisum[i+1]=multisum[i]+v[i]*1LL*i;
48       }
49
50       for(int i=0; i<n; i++)
51       {
52           int mid=i;
53           ll tot=0;
54           ll smid=v[i];
55           for(int j=i; j<n; j++)
56           {
57               tot+=v[j];
58               while(smid+smid<tot)
59                   smid+=v[++mid];
60               c[i][j]=k*dist(i, j, mid);
61           }
62       }
63       for(int i=0; i<=n; i++)
64           dp[0][i]=LINF;
65       for(int i=0; i<=k; i++)
66           dp[i][n]=0;
67
68       for(int i=1; i<=n; i++)
69       {
70           solve(i, 0, n-1, 0, n-1);
71           i>1?printf("_"):NULL;
72           printf("%lld", i*b+dp[i][0]);
73       }
74       printf("\n");
```

```
75
76      return  0;
77  }
```

3.1.3 Convex Hull I

Original recurrence:
$$dp[i] = min(dp[j] + b[j] * a[i]) \text{ for j<i}$$
Conditions:
$$b[j] >= b[j+1]$$
$$a[i] <= a[i+1]$$
Solution:
Hull cht=Hull() or DynamicHull cht;
cht.insertLine(b[0], dp[0])
for(int i=1; i<n; i++)
{
    dp[i]=cht.query(a[i]);
    cht.insertLine(b[i], dp[i]);
}
answer is dp[n-1];

Linear

```cpp
class Hull
{
    const static int CN=1e+5+35;
public:
    long long a[CN], b[CN];
    double x[CN];
    int head, tail;
    Hull():head(1), tail(0){};

    long long query(long long xx)
    {
        if(head>tail)
            return 0;
        while(head<tail && x[head+1]<=xx)
            head++;
        x[head]=xx;
        return a[head]*xx+b[head];
    }

    void insertLine(long long aa, long long bb)
    {
        double xx=-1e18;
        while(head<=tail)
        {
            if(aa==a[tail])
                return;
            xx=1.0*(b[tail]-bb)/(aa-a[tail]);
            if(head==tail || xx>=x[tail])
                break;
            tail--;
        }
        a[++tail]=aa;
        b[tail]=bb;
        x[tail]=xx;
    }
};
```

Dynamic

```cpp
const long long is_query=-(1LL<<62);
class Line
{
public:
    long long m, b;
    mutable function<const Line*()>succ;
    bool operator < (const Line &rhs) const
    {
        if(rhs.b!=is_query)
            return m<rhs.m;
        const Line *s=succ();
        if(!s)
            return 0;
        long long x=rhs.m;
        return (b-s->b)<((s->m-m)*x);
    }
};

class HullDynamic: public multiset<Line>
{
public:
    void clear()
    {
        clear();
    }
    bool bad(iterator y)
    {
        auto z=next(y);
        if(y==begin())
        {
            if(z==end())
                return 0;
            return (y->m==z->m && y->b<=z->b);
        }
        auto x=prev(y);
        if(z==end())
            return (y->m == x->m && y->b<=x->b);
        return ((x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m - x->m));
    }
    void insertLine(ll m, ll b)
    {
        auto y=insert({m, b});
        y->succ=[=]
        {
            return next(y)==end()?0:&*next(y);
        };
        if(bad(y))
        {
            erase(y);
            return;
        }
        while(next(y)!=end() && bad(next(y)))
            erase(next(y));
        while(y!=begin() && bad(prev(y)))
            erase(prev(y));
    }
    long long query(long long x)
    {
        auto ret=*lower_bound((Line){x, is_query});
        return ret.m*x+ret.b;
    }
};
```

3.1.4 Convex Hull II

3.1.5 Knuth Optimization

```
// http://codeforces.com/blog/entry/8219
// Original Recurrence:
//   dp[i][j] = min(dp[i][k] + dp[k][j]) + C[i][j]    for k = i+1..j-1
// Necessary & Sufficient Conditions:
//   A[i][j-1] <= A[i][j] <= A[i+1][j]
//   with A[i][j] = smallest k that gives optimal answer
// Also applicable if the following conditions are met:
//   1. C[a][c] + C[b][d] <= C[a][d] + C[b][c] (quadrangle inequality)
//   2. C[b][c] <= C[a][d]                     (monotonicity)
//   for all a <= b <= c <= d
// To use:
//   Calculate dp[i][i] and A[i][i]
//
//   FOR(len = 1..n-1)
//     FOR(i = 1..n-len) {
//       j = i + len
//       FOR(k = A[i][j-1]..A[i+1][j])
//         update(dp[i][j])
//     }

// OPTCUT
const int MN = 2011;
int a[MN], dp[MN][MN], C[MN][MN], A[MN][MN];
int n;

void solve() {
    cin >> n; FOR(i,1,n) { cin >> a[i]; a[i] += a[i-1]; }
    FOR(i,1,n) FOR(j,i,n) C[i][j] = a[j] - a[i-1];

    FOR(i,1,n) dp[i][i] = 0, A[i][i] = i;

    FOR(len,1,n-1)
        FOR(i,1,n-len) {
            int j = i + len;
            dp[i][j] = 2000111000;
            FOR(k,A[i][j-1],A[i+1][j]) {
                int cur = dp[i][k-1] + dp[k][j] + C[i][j];
                if (cur < dp[i][j]) {
                    dp[i][j] = cur;
                    A[i][j] = k;
                }
            }
        }
    cout << dp[1][n] << endl;
}
```

3.2 Matrix Exponentiation

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 & ... & 0 \\ 0 & 0 & 1 & 0 & ... & 0 \\ 0 & 0 & 0 & 1 & ... & 0 \\ ... & ... & ... & ... & ... & ... \\ c_k & c_{k-1} & c_{k-2} & c_{k-3} & ... & c_1 \end{bmatrix}$$

$$F_N = T^{N-1}F_1$$

```cpp
typedef long long ll;
typedef vector<vector<ll> > matrix;
const ll MOD=303700049;
int n, t;
ll k;
ll val[101];

ll modmul(ll a, ll b)
{
    return ((a%MOD)*(b%MOD))%MOD;
}

ll modsum(ll a, ll b)
{
    return ((a%MOD)+(b%MOD))%MOD;
}

matrix basem;
matrix mat_mul(matrix A, matrix B)
{
    int t=A.size();
    matrix ret=basem;
    for(int i=0; i<t; i++)
    {
        for(int j=0; j<t; j++)
        {
            for(int k=0; k<t; k++)
            {
                ret[i][j]=(ret[i][j]+A[i][k]*B[k][j]);
            }
            ret[i][j]%=MOD;
        }
    }
    return ret;
}

matrix mat_pow(matrix &A, ll k)
{
    if(k==1)
        return A;
    if(k&1)
        return mat_mul(A, mat_pow(A,k-1));
    matrix ret=mat_pow(A, k>>1);
    return mat_mul(ret, ret);
}

//o build pode variar, sendo ele a base do fibonacci
matrix build()
{
    matrix ret(t, vector<ll>(t));
    for(int i=0; i<n; i++)
        ret[0][i]=i+1;
    for(int i=1; i<n; i++)
        for(int j=0; j<n; j++)
            ret[i][j]=(j+1==i);
    for(int i=0; i<n; i++)
        ret[t-1][i]=i+1;
    ret[t-1][t-1]=1;
    return ret;
}

pair<ll,ll>calc(ll k)
{
    if(n>=k)
```

```
65            return mp(val[k-1], 0);
66        matrix base=build();
67        matrix fib=mat_pow(base, k-n);
68
69        ll ret=0;
70        reverse(val, val+n);
71        for(int i=0; i<n; i++)
72            ret=modsum(ret, modmul(fib[0][i], val[i]));
73
74        ll sum=0;
75        for(int i=0; i<n; i++)
76            sum=modsum(sum, modmul(fib[n][i], val[i]));
77        return mp(ret, sum);
78    }
79
80    void solve()
81    {
82        //First = f(n-x), Second = somatoaria de f(0) ate f(n-x)
83        pair<ll,ll>ans=calc(k);
84        if(k>n)
85        {
86            for(int i=0; i<n; i++)
87            {
88                ans.S=ans.S+val[i];
89                if(ans.S>MOD)
90                    ans.S%=MOD;
91            }
92        }
93        else
94        {
95            for(int i=0; i<k; i++)
96            {
97                ans.S=ans.S+val[i];
98                if(ans.S>MOD)
99                    ans.S%=MOD;
100           }
101       }
102       printf("%lld %lld\n", ans.F, ans.S);
103   }
104
105   int main()
106   {
107       while(scanf("%d %lld", &n, &k)!=EOF)
108       {
109           t=n+1;
110           basem.clear();
111           basem.resize(t, vector<ll>(t));
112           //val[i] = valores iniciais conhecidos da recorrencia
113           for(int i=0; i<n; i++)
114           {
115               scanf("%lld", &val[i]);
116           }
117           solve();
118       }
119       return 0;
120   }
```

3.3 Digits

```
1   char str[100];
2   int dp[100][300][2];
3   bool memo[100][300][2];
4   int n, k;
5
6   //numeros de 0 a x, tal que a soma dos digitos eh igual a k
7   int solve(int i, int s, int t)
8   {
9       if(i==n)
10      {
11          if(!t && s==k)
12              return 1;
13          return 0;
14      }
15      if(s>k)
16          return 0;
17      if(memo[i][s][t])
18          return dp[i][s][t];
19      int &ret=dp[i][s][t]=0;
20      if(t)
21      {
22          for(int j=0; j<=str[i]-'0'; j++)
23          {
24              if(j==str[i]-'0')
25                  ret+=solve(i+1, s+j, 1);
26              else
27                  ret+=solve(i+1, s+j, 0);
28          }
29      }
30      else
31      {
32          for(int j=0; j<10; j++)
33          {
34              ret+=solve(i+1, s+j, 0);
35          }
36      }
37      memo[i][s][t]=true;
38      return ret;
39  }
40
41  //quantos bits ativos existem entre 0 e x
42  string str2;
43  int n2;
44  int dp2[100][300][2];
45  bool memo2[100][300][2];
46  int solve2(int i, int s, int t)
47  {
48      if(i==n2)
49          return s;
50      if(memo2[i][s][t])
51          return dp2[i][s][t];
52      int &ret=dp2[i][s][t]=0;
53      if(t)
54      {
55          for(int j=0; j<=str2[i]-'0'; j++)
56          {
57              if(j==str2[i]-'0')
58                  ret+=solve2(i+1, s+(j==1), 1);
59              else
60                  ret+=solve2(i+1, s+(j==1), 0);
61          }
62      }
63      else
64      {
65          for(int j=0; j<2; j++)
66          {
67              ret+=solve2(i+1, s+(j==1), 0);
68          }
69      }
70      memo2[i][s][t]=true;
71      return ret;
72  }
73
74  //numeros de 1 a x, tal que a soma dos digitos eh multiplo de k
```

```
75    char str3[100];
76    int n3;
77    int dp3[100][300][2];
78    bool memo3[100][300][2];
79    int solve3(int i, int s, int t)
80    {
81        if(i==n3)
82            return !s;
83        if(memo3[i][s][t])
84            return dp3[i][s][t];
85        int &ret=dp3[i][s][t]=0;
86        if(t)
87        {
88            for(int j=0; j<=str3[i]-'0'; j++)
89            {
90                if(j==str3[i]-'0')
91                    ret+=solve3(i+1, (s+j)%k, 1);
92                else
93                    ret+=solve3(i+1, (s+j)%k, 0);
94            }
95        }
96        else
97        {
98            for(int j=0; j<10; j++)
99            {
100                ret+=solve3(i+1, (s+j)%k, 0);
101            }
102        }
103        memo3[i][s][t]=true;
104        return ret;
105    }
106
107    //numeros de 1 a x, tal que o xor dos digitos eh igual a k
108    char str4[100];
109    int n4;
110    int dp4[100][300][2];
111    bool memo4[100][300][2];
112    int solve4(int i, int s, int t)
113    {
114        if(i==n4)
115            return s==k;
116        if(memo4[i][s][t])
117            return dp4[i][s][t];
118        int &ret=dp4[i][s][t]=0;
119        if(t)
120        {
121            for(int j=0; j<=str4[i]-'0'; j++)
122            {
123                if(j==str4[i]-'0')
124                    ret+=solve4(i+1, (s^j), 1);
125                else
126                    ret+=solve4(i+1, (s^j), 0);
127            }
128        }
129        else
130        {
131            for(int j=0; j<10; j++)
132            {
133                ret+=solve4(i+1, (s^j), 0);
134            }
135        }
136        memo4[i][s][t]=true;
137        return ret;
138    }
```

3.4   Grundy Numbers

Positions have the following properties:

- All terminal positions are losing.

- If a player is able to move to a losing position then he is in a winning position.

- If a player is able to move only to the winning positions then he is in a losing position.

```cpp
const int MN=1e+5;
bool memo[MN];
int dp[MN];
int grundy(int x)
{
    if(x==0)
        return 0;
    if(memo[x])
        return dp[x];
    set<int>mex;
    for(;;)//moves
        mex.insert(grundy(x-(moves)));
    int &ret=dp[x]=0;
    while(mex.count(ret))
        ret++;
    memo[x]=true;
    return ret;
}
```

3.5   All Bitmasks subpatterns

```
1  for(int m=0; m<(1<<n); ++m)
2      for(int s=m; s; s=(s-1)&m)
```

Chapter 4

String

4.1 Hash

```
1   typedef unsigned long long ull;
2   class hashc
3   {
4   public:
5       vector<ull>prefix;
6       vector<ull>power;
7       int k=37;
8       int t;
9       hashc(){};
10      hashc(vector<int>&data)
11      {
12          t=data.size();
13          prefix.resize(t+1, 0);
14          power.resize(t+1, 0);
15          prefix[0]=0;
16          power[0]=1;
17          for(int i=0; i<t; i++)
18          {
19              prefix[i+1]=prefix[i]*k+data[i];
20              power[i+1]=power[i]*k;
21          }
22      }
23
24      hashc build(string &str)
25      {
26          vector<int>data(str.size());
27          for(int i=0; i<str.size(); i++)
28              data[i]=(str[i]-'a'+1);
29          return hashc(data);
30      }
31
32      ull get()
33      {
34          return prefix[t];
35      }
36      ull calc(int l, int r)
37      {
38          return prefix[r]-(prefix[l-1]*power[r-l+1]);
39      }
40      bool same(int xl, int xr, int yl, int yr)
41      {
42          return this->calc(xl, xr)==this->calc(yl, yr);
43      }
44      int find(hashc &pattern)
45      {
46          int pt=pattern.t;
47          ull val=pattern.calc(1, pt);
48          for(int i=1; i<=t-pt+1; i++)
49          {
50              if(this->calc(i, i+pt-1)==val)
51                  return i-1;
52          }
53          return -1;
```

```
54        }
55    };
```

4.2 KMP

```
1   int lps[1000000];
2   void lps_calc(string &str)
3   {
4       lps[0]=0;
5       for(int i=1, j=0, f=0; i<str.size(); i+=f, f=0)
6       {
7           if(str[i]==str[j])
8           {
9               lps[i]=j;
10              j++;
11              f=1;
12          }
13          else
14          {
15              if(j>0)
16              {
17                  j=lps[j-1];
18              }
19              else
20              {
21                  lps[i]=0;
22                  f=1;
23              }
24          }
25      }
26  }
27
28  //finding str in pat
29  void kmp(string &str, string &pat)
30  {
31      lps_calc(pat);
32      int i=0, j=0;
33      while(i<str.size())
34      {
35          if(str[i]==pat[j])
36          {
37              i++;
38              j++;
39          }
40          if(j==pat.size())
41          {
42              printf("Padrao encontrado em: [%d, %d]", i-j, (i-j)+pat.size()-1);
43              j=lps[j-1];
44          }
45          else if(i<str.size() && str[i]!=pat[j])
46          {
47              if(j!=0)
48                  j=lps[j-1];
49              else
50                  i++;
51          }
52      }
53  }
```

4.3 Aho Corasick

```cpp
class aho_corasick
{
private:
    static const int MNT=1e+6;
    static const int MNC=26;
public:
    int trie[MNT][MNC];
    int term[MNT];
    int link[MNT];
    int sum[MNT];
    int cnt=1;
    aho_corasick(){};
    void clear()
    {
        RESET(trie, 0);
        RESET(term, 0);
        RESET(link, 0);
        RESET(sum, 0);
        cnt=1;
    }
    int node(int x, int j)
    {
        return trie[x][j];
    }
    int end(int x, int j)
    {
        return term[ node(x,j) ];
    }
    void insert(char *str)
    {
        int sz=strlen(str);
        int no=0;
        for(int i=0; i<sz; i++)
        {
            int x=str[i]-'a';
            if(!trie[no][x])
                trie[no][x]=cnt++;
            sum[ trie[no][x] ]++;
            no=trie[no][x];
        }
        term[no]++;
    }
    bool find(char *str)
    {
        int sz=strlen(str);
        int no=0;
        for(int i=0; i<sz; i++)
        {
            int x=str[i]-'a';
            if(!sum[ trie[no][x] ])
                return false;
            no=trie[no][x];
        }
        return true;
    }
    void erase(char *str)
    {
        int sz=strlen(str);
        int no=0;
        for(int i=0; i<sz; i++)
        {
            int x=str[i]-'a';
            sum[ trie[no][x] ]--;
            no=trie[no][x];
        }
        term[no]--;
    }
    void update_link()
    {
        queue<int>aho;
        aho.push(0);
        while(!aho.empty())
        {
            int x=aho.front();
```

```
75              aho.pop();
76              term[x]|=term[ link[x]  ];
77              for(int i=0; i<MNC; i++)
78              {
79                  if(trie[x][i])
80                  {
81                      int y=trie[x][i];
82                      aho.push(y);
83                      link[y]=x?trie[ link[x]  ][i]:0;
84                  }
85                  else
86                  {
87                      trie[x][i]=trie[ link[x]  ][i];
88                  }
89              }
90          }
91      }
92  };
```

4.4 Manacher

```
1    char s[200000];
2    int n;
3    // Encontrar palindromos − inicializa d1 e d2 com zeros, e eles guadram
4    // o numero de palindromos centrados na posicao i (d1[i] e d2[i])
5    // impar
6    int d1[200000], d2[200000];
7    void imp(){
8        int l=0, r=−1;
9        for (int i=0; i<n; ++i) {
10           int k = (i>r ? 0 : min (d1[l+r−i], r−i)) + 1;
11           while (i+k < n && i−k >= 0 && s[i+k] == s[i−k])  ++k;
12           d1[i] = −−k;
13           if (i+k > r)
14               l = i−k,  r = i+k;
15       }
16   }
17   // par
18   void par(){
19       int l=0, r=−1;
20       for (int i=0; i<n; ++i) {
21           int k = (i>r ? 0 : min (d2[l+r−i+1], r−i+1)) + 1;
22           while (i+k−1 < n && i−k >= 0 && s[i+k−1] == s[i−k])  ++k;
23           d2[i] = −−k;
24           if (i+k−1 > r)
25               l = i−k,  r = i+k−1;
26       }
27   }
```

4.5 Z-Algorithm

```cpp
// Z-algorithm , O(N)
// Builds array z such that z[i] = size of longest prefix substring
// starting at index i
vector<int> Z(string s) {
    vector<int> z(1,s.size());
    int l=0,r=0;
    for(int a=1;a<(int)s.size();++a) {
        if(r < a) {
            l = r = a;
            while(r<(int)s.size() && s[r] == s[r-l]) ++r;
            z.push_back(r-l);
            r--;
        }
        else if(z[a-l] < r-a+1) z.push_back(min<int>(z[a-l],s.size()-a));
        else {
            l = a;
            while(r<(int)s.size() && s[r] == s[r-l]) ++r;
            z.push_back(r-l);
            r--;
        }
    }
    return z;
}
```

4.6 Suffix Array & LCP

```cpp
1   const int MN=1e+6+35;
2   int data[MN], sa[MN], lcp[MN], lcp_rank[MN];
3
4   // lexicographic order for pairs
5   inline bool leq(int a1, int a2, int b1, int b2)
6   {
7       return(a1 < b1 || a1 == b1 && a2 <= b2);
8   }
9
10  // and triples
11  inline bool leq(int a1, int a2, int a3, int b1, int b2, int b3)
12  {
13      return(a1 < b1 || a1 == b1 && leq(a2,a3, b2,b3));
14  } // and triples
15
16  // stably sort a[0..n-1] to b[0..n-1] with keys in 0..K from r
17  static void radixPass(int* a, int* b, int* r, int n, int K)
18  {// count occurrences
19      int* c = new int[K + 1]; // counter array
20      for (int i = 0; i <= K; i++)
21          c[i] = 0; // reset counters
22      for (int i = 0; i < n; i++)
23          c[r[a[i]]]++; // count occurrences
24      for (int i = 0, sum = 0; i <= K; i++) // exclusive prefix sums
25      {
26          int t = c[i];
27          c[i] = sum;
28          sum += t;
29      }
30      for (int i = 0;  i < n; i++)
31          b[c[r[a[i]]]++] = a[i]; // sort
32  }
33
34  // find the suffix array SA of s[0..n-1] in {1..K}ËĘn
35  // require s[n]=s[n+1]=s[n+2]=0, n>=2
36  void suffixArray(int* s, int* SA, int n, int K)
37  {
38      int n0 = (n+2)/3, n1 = (n+1)/3, n2 = n/3, n02 = n0+n2;
39      int* s12 = new int[n02+3]; s12[n02] = s12[n02+1] = s12[n02+2] = 0;
40      int* SA12 = new int[n02+3]; SA12[n02] = SA12[n02+1] = SA12[n02+2] = 0;
41      int* s0 = new int[n0];
42      int* SA0 = new int[n0];
43      // generate positions of mod 1 and mod 2 suffixes
44      // the "+(n0-n1)" adds a dummy mod 1 suffix if n%3 == 1
45      for (int i=0, j=0; i < n + (n0-n1); i++)
46          if (i%3 != 0) s12[j++] = i;
47      // lsb radix sort the mod 1 and mod 2 triples
48      radixPass(s12 , SA12, s+2, n02, K);
49      radixPass(SA12, s12 , s+1, n02, K);
50      radixPass(s12 , SA12, s  , n02, K);
51      // find lexicographic names of triples
52      int name = 0, c0 = -1, c1 = -1, c2 = -1;
53      for (int i = 0; i < n02; i++)
54      {
55          if (s[SA12[i]] != c0 || s[SA12[i]+1] != c1 || s[SA12[i]+2] != c2)
56          {
57              name++;
58              c0 = s[SA12[i]];
59              c1 = s[SA12[i]+1];
60              c2 = s[SA12[i]+2];
61          }
62          if (SA12[i]%3 == 1) s12[SA12[i]/3] = name; // left half
63          else s12[SA12[i]/3 + n0] = name; // right half
64      }
65      // recurse if names are not yet unique
66      if (name < n02)
67      {
68          suffixArray(s12, SA12, n02, name);
69          // store unique names in s12 using the suffix array
70          for(int i=0; i<n02; i++)
71              s12[SA12[i]] = i + 1;
72      }
73      else // generate the suffix array of s12 directly
```

```
74          {
75              for(int i = 0;  i < n02; i++)
76                  SA12[s12[i] − 1] = i;
77          }
78      // stably sort the mod 0 suffixes from SA12 by their first character
79      for (int i=0, j=0; i<n02; i++)
80          if (SA12[i] < n0) s0[j++] = 3∗SA12[i];
81      radixPass(s0, SA0, s, n0, K);
82      // merge sorted SA0 suffixes and sorted SA12 suffixes
83      for (int p = 0, t = n0−n1, k = 0; k < n; k++)
84      {
85          #define GetI() (SA12[t] < n0 ? SA12[t] ∗ 3 + 1 : (SA12[t] − n0) ∗ 3 + 2)
86          int i = GetI(); // pos of current offset 12 suffix
87          int j = SA0[p]; // pos of current offset 0 suffix
88          if (SA12[t] < n0 ? // different compares for mod 1 and mod 2 suffixes
89              leq(s[i], s12[SA12[t] + n0], s[j], s12[j/3]) :
90              leq(s[i],s[i+1],s12[SA12[t]−n0+1], s[j],s[j+1],s12[j/3+n0]))
91          {// suffix from SA12 is smaller
92              SA[k] = i; t++;
93              if (t == n02) // done —— only SA0 suffixes left
94              for (k++; p < n0; p++, k++) SA[k] = SA0[p];
95          }
96          else
97          {// suffix from SA0 is smaller
98              SA[k] = j; p++;
99              if (p == n0) // done —— only SA12 suffixes left
100             for (k++; t < n02; t++, k++) SA[k] = GetI();
101         }
102     }
103 }
104
105 void buildlcp(int n)
106 {
107     int k=0;
108     for(int i=0; i<n; i++)
109         lcp_rank[ sa[i] ]=i;
110     for(int i=0; i<n; i++, k?k−−:0)
111     {
112         if(lcp_rank[i]==n−1)
113         {
114             k=0;
115             continue;
116         }
117         int j=sa[ lcp_rank[i]+1 ];
118         while(i+k<n && j+k<n && data[i+k]==data[k+j])
119             k++;
120         lcp[ lcp_rank[i] ]=k;
121     }
122 }
123
124 int main()
125 {
126     int n;
127     scanf("%d", &n);
128     for(int i=0; i<n; i++)
129     {
130         char x;
131         scanf(" %c", &x);
132         data[i]=(int)x;
133     }
134     //data[i]>=1
135     data[n]=data[n+1]=data[n+2]=data[n+3]=0;
136     n++;
137     //suffixArray(string, ponteiro para suffix array, numero de elementos da string, number of
                elementos do alfabeto);
138     suffixArray(data, sa, n, 256);
139     for(int i=0; i<n; i++)
140         printf("%d ", sa[i]);
141     printf("\n\n");
142
143     //buildlcp(numero de elementos da string)
144     buildlcp(n);
145     for(int i=0; i<n; i++)
146         printf("%d\n", lcp[i]);
147     return 0;
148 }
```

Chapter 5

Mathematic

5.1   Prime Numbers

5.1.1 Linear Sieve

```
1   //prime(x):(lp[x]==x)
2   const int MN=1e+6;
3   long long lp[MN+1];
4   vector<long long>pr;
5
6   void sieve()
7   {
8       for(long long i=2; i<=MN; i++)
9       {
10          if(lp[i]==0)
11          {
12              lp[i]=i;
13              pr.push_back(i);
14          }
15          for(long long j=0; j<pr.size() && pr[j]<=lp[i] && i*pr[j]<=MN; j++)
16              lp[i*pr[j]]=pr[j];
17      }
18  }
```

5.1.2 Miller Rabin

```
1   //millerRabin(n) returns if n is prime
2   //not accurate for all n
3   #define gcd(x, y) __gcd(x, y)
4   ll powmod(ll a, ll b, ll m)
5   {
6       ll ret=1;
7       while(b)
8       {
9           if(b&1)
10              ret=(ret*a)%m, --b;
11          else
12              a=(a*a)%m, b>>=1;
13      }
14      return ret;
15  }
16
17  bool millerRabin(ll n)
18  {
19      ll b=2;
20      for(ll g; (g=gcd(n, b))!=1; b++)
21          if(n>g)
22              return false;
23      ll p=0, q=n-1;
24      while((q&1)==0)
25          p++, q>>=1;
26      ll rem=powmod(b, q, n);
27      if(rem==1 || rem==n-1)
28          return true;
29      for(ll i=1; i<p; i++)
30      {
31          rem=(rem*rem)%n;
32          if(rem==n-1)
33              return true;
34      }
35      return false;
36  }
```

5.1.3 BPSW

```
1   //bpsw(n) returns if n is prime
2   #define gcd(x, y) __gcd(x, y)
3   ll jacobi(ll a, ll b)
4   {
5       if(a==0 || a==1)
6           return a;
7       if(a<0)
8       {
9           if((b&2)==0)
10              return jacobi(-a, b);
11          return -jacobi(-a, b);
12      }
13      ll a1=a, e=0;
14      while((a1&1)==0)
15          a1>>=1, e++;
16      ll s;
17      if((e&1)==0 || (b&7)==1 || (b&7)==7)
18          s=1;
19      else
20          s=-1;
21      if((b&3)==3 && (a1&3)==3)
22          s=-s;
23      if(a1==1)
24          return s;
25      return s*jacobi(b%a1, a1);
26  }
27
28  bool bpsw(ll n)
29  {
30      if((ll)sqrt(n+0.0)*(ll)sqrt(n+0.0)==n)
31          return false;
32      ll dd=5;
33      while(1)
34      {
35          ll g=gcd(n, abs(dd));
36          if(1<g && g<n)
37              return false;
38          if(jacobi(dd, n)==-1)
39              break;
40          dd=dd<0?-dd+2:-dd-2;
41      }
42      ll p=1, q=(p*p-dd)/4;
43      ll d=n+1, s=0;
44      while((d&1)==0)
45          s++, d>>=1;
46      ll u=1, v=p, u2m=1, v2m=p, qm=q, qm2=q*2, qkd=q;
47      for(ll mask=2; mask<=d; mask<<=1)
48      {
49          u2m=(u2m*v2m)%n;
50          v2m=(v2m*v2m)%n;
51          while(v2m<qm2)
52              v2m+=n;
53          v2m-=qm2;
54          qm=(qm*qm)%n;
55          qm2=qm*2;
56          if(d&mask)
57          {
58              ll t1=(u2m*v)%n, t2=(v2m*u)%n;
59              ll t3=(v2m*v)%n, t4=(((u2m*u)%n)*dd)%n;
60              u=t1+t2;
61              if(u&1)
62                  u+=n;
63              u=(u>>1)%n;
64              v=(t3+t4);
65              if(v&1)
66                  v+=n;
67              v=(v>>1)%n;
68              qkd=(qkd*qm)%n;
69          }
70      }
71      if(u==0 || v==0)
72          return true;
73      ll qkd2=qkd*2;
74      for(ll r=1; r<s; r++)
```

```
75        {
76            v=(v*v)%n−qkd2;
77            v+=v<0?n:0;
78            v+=v<0?n:0;
79            v−=v>=n?n:0;
80            v−=v>=n?n:0;
81            if(v==0)
82                return true;
83            if(r<s−1)
84            {
85                qkd=(qkd*1LL*qkd)%n;
86                qkd2=qkd*2;
87            }
88        }
89        return false;
90  }
```

5.1.4 Primality Test

```
1    //call sieve() before isPrime(x)
2    //define k=50 as trivial limit
3    bool isPrime(ll x)
4    {
5        if(x==1)
6            return false;
7        if(x==2)
8            return true;
9        if(x%2==0)
10           return false;
11       for(int i=0; i<k && x>pr[i]; i++)
12           if(x%pr[i]==0)
13               return false;
14       if(pr[k-1]*pr[k-1]>=x)
15           return true;
16       //return only millerRabin(x) for fast process
17       //not accurate for all x
18       return millerRabin(x)?bpsw(x):false;
19   }
```

5.1.5 Java Pollard Rho Decomposition

```java
public static Random rand = new Random();
public static long v, ans=1, fact;
public static long gcd(long x, long y)
{
    if(y==0)
        return x;
    return gcd(y, x%y);
}
public static long rho(long x)
{
    long a, b, cnt=2;
    a=b=rand.nextLong()%x;
    for(long i=1; ;i++)
    {
        BigInteger Ba=BigInteger.valueOf(a);
        BigInteger Bx=BigInteger.valueOf(x);
        BigInteger aux=Ba.multiply(Ba).add(BigInteger.valueOf(2)).mod(Bx);
        a=aux.longValue();
        if(a==b)
            return 0;
        long g=gcd(Math.abs(a-b),x);
        if(g!=1)
            return g;
        if(i==cnt)
        {
            b=a;
            cnt*=2;
        }
    }
}
public static void solve(long x)
{
    BigInteger Bx=BigInteger.valueOf(x);
    if(Bx.isProbablePrime(20))
    {
        long cnt=0;
        while(v%x==0)
        {
            v/=x;
            cnt++;
        }
        ans*=(cnt+1);
        if(v!=1)
            solve(v);
    }
    else
    {
        for(fact=rho(x); fact==0; fact=rho(x)){}
        solve(fact);
    }
}
public static void main(String[] args) throws Exception
{
    ans=1;
    v=sc.nextLong();
    if(v!=1)
        solve(v);
    System.out.println(ans);
}
```

5.2 Fast Fourier Transformation

```
1   #define PI (double)acos(-1.0)
2   typedef complex<double> base;
3   void fft(vector<base>&data, bool invert)
4   {
5       int n=data.size();
6       for(int i=1, j=0; i<n; i++)
7       {
8           int bit=n>>1;
9           for(; j>=bit; bit>>=1)
10              j-=bit;
11          j+=bit;
12          if(i<j)
13              swap(data[i], data[j]);
14      }
15
16      for(int len=2; len<=n; len<<=1)
17      {
18          double ang=2*PI/len*(invert?-1:1);
19          base wlen(cos(ang), sin(ang));
20          for(int i=0; i<n; i+=len)
21          {
22              base w(1);
23              for(int j=0; j<len/2; j++)
24              {
25                  base u=data[i+j], v=data[i+j+len/2]*w;
26                  data[i+j]=u+v;
27                  data[i+j+len/2]=u-v;
28                  w*=wlen;
29              }
30          }
31      }
32      if(invert)
33          for(int i=0; i<n; i++)
34              data[i]/=n;
35  }
36
37  vector<int>fft_multiply(vector<int>&a, vector<int>&b)
38  {
39      vector<base>fa(a.begin(), a.end());
40      vector<base>fb(b.begin(), b.end());
41      int n=1;
42      while(n<max(a.size(), b.size()))
43          n<<=1;
44      n<<=1;
45      fa.resize(n);
46      fb.resize(n);
47      fft(fa, false);
48      fft(fb, false);
49      for(int i=0; i<n; i++)
50          fa[i]*=fb[i];
51      fft(fa, true);
52
53      vector<int>ret(n);
54      for(int i=0; i<n; i++)
55          ret[i]=(int)(fa[i].real()+0.5);
56
57      int carry=0;
58      for(int i=0; i<n; i++)
59      {
60          ret[i]+=carry;
61          carry=ret[i]/10;
62          ret[i]%=10;
63      }
64      return ret;
65  }
66
67  int main()
68  {
69      int n, m;
70      scanf("%d %d", &n, &m);
71      vector<int>a,b;
72
73      for(int i=0; i<n; i++)
74      {
```

```
75          int x;
76          scanf("%d", &x);
77          a.pb(x);
78      }
79
80      for(int i=0; i<m; i++)
81      {
82          int x;
83          scanf("%d", &x);
84          b.pb(x);
85      }
86      reverse(a.begin(), a.end());
87      reverse(b.begin(), b.end());
88
89      vector<int>ans=fft_multiply(a, b);
90      reverse(ans.begin(), ans.end());
91      bool flag=false;
92      for(int i=0; i<ans.size(); i++)
93      {
94          if(ans[i])
95              flag=true;
96          if(flag)
97              printf("%d", ans[i]);
98      }
99      printf("\n");
100     return 0;
101 }
```

5.3 Modular Math

5.3.1 Multiplicative Inverse

```cpp
template<typename T>T extGcd(T a, T b, T &x, T &y)
{
    if(b==0)
    {
        x=1;
        y=0;
        return a;
    }
    else
    {
        T g=extGcd(b, a%b, y, x);
        y-=a/b*x;
        return g;
    }
}

template<typename T>T invMod(T a, T m)
{
    T x, y;
    extGcd(a, m, x, y);
    return (x%m+m)%m;
}
```

5.3.2 Linear All Multiplicative Inverse

```cpp
r[1]=1
for(int i=2; i<m; i++)
    r[i]=((m-(m/i) * r[m%i])%m)%m;
```

5.3.3 Factorial

$(N!)\%P$ in $O(P * log_n)$

```
1   int factmod (int n, int p) {
2       int res = 1;
3       while (n > 1) {
4           res = (res * ((n/p) % 2 ? p-1 : 1)) % p;
5           for (int i=2; i<=n%p; ++i)
6               res = (res * i) % p;
7           n /= p;
8       }
9       return res % p;
10  }
```

5.4 Gaussian Elimination

```
1    const int MAXN = 110;
2
3    typedef double Number;
4    const Number EPS = 1e-9;
5
6    Number mat[MAXN][MAXN];
7    int idx[MAXN];     // row index
8    int pivot[MAXN]; // pivot of row i
9
10   // Solves Ax = B, where A is a neq x nvar matrix and B is mat[*][nvar]
11   // Returns a vector of free variables (empty if system is defined,
12   // or {-1} if no solution exists)
13   // Reduces matrix to reduced echelon form
14   vector<int> solve(int nvar, int neq)
15   {
16       for(int i = 0; i < neq; i++) idx[i] = i;
17       int currow = 0;
18       vector<int> freeVars;
19       for(int col = 0; col < nvar; col++)
20       {
21           int pivotrow = -1;
22           Number val = 0;
23           for(int row = currow; row < neq; row++)
24           {
25               if(fabs(mat[idx[row]][col]) > val + EPS)
26               {
27                   val = fabs(mat[idx[row]][col]);
28                   pivotrow = row;
29               }
30           }
31           if(pivotrow == -1) { freeVars.push_back(col); continue; }
32           swap(idx[currow], idx[pivotrow]);
33           pivot[currow] = col;
34           for(int c = 0; c <= nvar; c++)
35           {
36               if(c == col) continue;
37               mat[idx[currow]][c] = mat[idx[currow]][c] / mat[idx[currow]][col];
38           }
39           mat[idx[currow]][col] = 1;
40           for(int row = 0; row < neq; row++)
41           {
42               if(row == currow) continue;
43               Number k = mat[idx[row]][col] / mat[idx[currow]][col];
44               for(int c = 0; c <= nvar; c++)
45                   mat[idx[row]][c] -= k * mat[idx[currow]][c];
46           }
47           currow++;
48       }
49       for(int row = currow; row < neq; row++)
50           if(mat[idx[row]][nvar] != 0) return vector<int>(1, -1);
51       return freeVars;
52   }
```

5.5 Combinatorics

$$C_n^k = \frac{n!}{k!(n-k)!}$$

```
1   int C (int n, int k)
2   {
3       int res = 1;
4       for (int i=n-k+1; i<=n; ++i)
5           res *= i;
6       for (int i=2; i<=k; ++i)
7           res /= i;
8       return res;
9   }
```

Improved implementation:

```
1   int C (int n, int k)
2   {
3       double res = 1;
4       for (int i=1; i<=k; ++i)
5           res = res * (n-k+i) / i;
6       return (int) (res + 0.01);
7   }
```

5.6 Integral

$\int_a^b f(x)dx$

```
1   double a, b;
2   const int N = 1000*1000;
3   double s = 0;
4   double h = (b - a) / N;
5   for (int i=0; i<=N; ++i) {
6       double x = a + h * i;
7       s += f(x) * ((i==0 || i==N) ? 1 : ((i&1)==0) ? 2 : 4);
8   }
9   s *= h / 3;
```

Chapter 6

Geometry

6.1 2d

6.1.1 Point Template

```
1   inline int cmp(double x, double y = 0, double tol = eps)
2   {
3       return (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
4   }
5
6   struct point
7   {
8       double x, y;
9       point(double x = 0, double y = 0): x(x), y(y) {}
10      point operator +(point q) { return point(x + q.x, y + q.y); }
11      point operator -(point q) { return point(x - q.x, y - q.y); }
12      point operator *(double t) { return point(x * t, y * t); }
13      point operator /(double t) { return point(x / t, y / t); }
14      double operator *(point q) {return x * q.x + y * q.y;}//a*b = |a||b|cos(ang)
15      double operator %(point q) {return x * q.y - y * q.x;}//a%b = |a||b|sin(ang)
16      double polar() { return ((y > -eps) ? atan2(y,x) : 2*Pi + atan2(y,x)); }
17      double mod() { return sqrt(x * x + y * y); }
18      double mod2() { return (x * x + y * y); }
19      point rotate(double t) {return point(x*cos(t)-y*sin(t), x*sin(t)+y*cos(t));}
20      int cmp(point q) const
21      {
22          if (int t = ::cmp(x, q.x)) return t;
23          return ::cmp(y, q.y);
24      }
25      bool operator ==(point q) const { return cmp(q) == 0; }
26      bool operator !=(point q) const { return cmp(q) != 0; }
27      bool operator < (point q) const { return cmp(q) < 0; }
28      static point pivot;
29  };
30  point point::pivot;
31  typedef vector<point> polygon;
```

6.1.2 Functions

```cpp
1    double abs(point p) { return hypot(p.x, p.y); }
2    double arg(point p) { return atan2(p.y, p.x); }
3
4    inline int ccw(point p, point q, point r)
5    {
6        return cmp((p - r) % (q - r));
7    }
8
9    //Projeta o vetor v sobre o vetor u (cuidado precisao)
10   point proj(point v, point u)
11   {
12       return u*((u*v) / (u*u));
13   }
14
15   //\angle(p,q,r)| e o menor angulo entre os vetores u(p-q) e v(r-q)
16   // p->q->r virar pra esquerda => angle(p,q,r) < 0
17   inline double angle(point p, point q, point r)
18   {
19       point u = p - q, v = r - q;
20       return atan2(u % v, u * v);
21   }
22
23   //Decide se q esta sobre o segmento fechado pr.
24   bool between(point p, point q, point r)
25   {
26       return ccw(p, q, r) == 0 && cmp((p - q) * (r - q)) <= 0;
27   }
28
29   //Decide se os segmentos fechados pq e rs tem pontos em comum
30   bool seg_intersect(point p, point q, point r, point s)
31   {
32       point A = q - p, B = s - r, C = r - p, D = s - q;
33       int a = cmp(A % C) + 2 * cmp(A % D);
34       int b = cmp(B % C) + 2 * cmp(B % D);
35       if (a == 3 || a == -3 || b == 3 || b == -3) return false;
36       if (a || b || p == r || p == s || q == r || q == s) return true;
37       int t = (p < r) + (p < s) + (q < r) + (q < s);
38       return t != 0 && t != 4;
39   }
40
41   // Calcula a distancia do ponto r ao segmento pq.
42   double seg_distance(point p, point q, point r)
43   {
44       point A = r - q, B = r - p, C = q - p;
45       double a = A * A, b = B * B, c = C * C;
46       if (cmp(b, a + c) >= 0) return sqrt(a);
47       else if (cmp(a, b + c) >= 0) return sqrt(b);
48       else return fabs(A % B) / sqrt(c);
49   }
50
51   // Classifica o ponto p em relacao ao poligono T.
52   // Retorna 0, -1 ou 1 dependendo se p esta no exterior, na fronteira
53   // ou no interior de T, respectivamente.
54   int in_poly(point p, polygon& T) {
55       double a = 0; int N = T.size();
56       for (int i = 0; i < N; i++) {
57           if (between(T[i], p, T[(i+1) % N])) return -1;
58           a += angle(T[i], p, T[(i+1) % N]);
59       }
60       return cmp(a) != 0;
61   }
62
63   //Encontra o ponto de intersecao das retas pq e rs.
64   point line_intersect(point p, point q, point r, point s)
65   {
66       point a = q - p, b = s - r, c = point(p % q, r % s);
67       return point(point(a.x, b.x) % c, point(a.y, b.y) % c) / (a % b);
68   }
69
70   // Calcula a area orientada do poligono T.
71   // Se o poligono P estiver em setido anti-horario, poly_area(P) > 0,
72   // e <0 caso contrario
73   double poly_area(polygon& T)
74   {
```

```
75        double s = 0; int n = T.size();
76        for (int i = 0; i < n; i++)
77            s += T[i] % T[(i+1) % n];
78        return s / 2;
79    }
80
81    //Calcula o incentro de um triangulo
82    point incenter(point p, point q, point r)
83    {
84        double a = (p-q).mod(), b = (p-r).mod(), c = (q-r).mod();
85        return (r*a + q*b + p*c) / (a + b + c);
86    }
87
88    //Centro de massa de um poligono
89    point centro_massa(polygon p)
90    {
91        double x=0., y=0., area = poly_area(p);
92        p.push_back(p[0]);
93        for (int i = 0; i < p.size()-1; i++) {
94            x += (p[i].x + p[i+1].x) * (p[i] % p[i+1]);
95            y += (p[i].y + p[i+1].y) * (p[i] % p[i+1]);
96        }
97        return point(x/(6*area), y/(6*area));
98    }
```

6.1.3 Polygons

```cpp
1   #include<bits/stdc++.h>
2
3   using namespace std;
4
5   #define EPS 1e-9
6   #define PI acos(-1.0)
7
8   double DEG_to_RAD(double d) { return d * PI / 180.0; }
9
10  double RAD_to_DEG(double r) { return r * 180.0 / PI; }
11
12  struct point { double x, y;    // only used if more precision is needed
13    point() { x = y = 0.0; }                          // default constructor
14    point(double _x, double _y) : x(_x), y(_y) {}         // user-defined
15    bool operator == (point other) const {
16     return (fabs(x - other.x) < EPS && (fabs(y - other.y) < EPS)); } };
17
18  struct vec { double x, y;  // name: 'vec' is different from STL vector
19    vec(double _x, double _y) : x(_x), y(_y) {} };
20
21  vec toVec(point a, point b) {       // convert 2 points to vector a->b
22    return vec(b.x - a.x, b.y - a.y); }
23
24  double dist(point p1, point p2) {              // Euclidean distance
25    return hypot(p1.x - p2.x, p1.y - p2.y); }          // return double
26
27  // returns the perimeter, which is the sum of Euclidian distances
28  // of consecutive line segments (polygon edges)
29  double perimeter(const vector<point> &P) {
30    double result = 0.0;
31    for (int i = 0; i < (int)P.size()-1; i++)  // remember that P[0] = P[n-1]
32      result += dist(P[i], P[i+1]);
33    return result; }
34
35  // returns the area, which is half the determinant
36  double area(const vector<point> &P) {
37    double result = 0.0, x1, y1, x2, y2;
38    for (int i = 0; i < (int)P.size()-1; i++) {
39      x1 = P[i].x; x2 = P[i+1].x;
40      y1 = P[i].y; y2 = P[i+1].y;
41      result += (x1 * y2 - x2 * y1);
42    }
43    return fabs(result) / 2.0; }
44
45  double dot(vec a, vec b) { return (a.x * b.x + a.y * b.y); }
46
47  double norm_sq(vec v) { return v.x * v.x + v.y * v.y; }
48
49  double angle(point a, point o, point b) {  // returns angle aob in rad
50    vec oa = toVec(o, a), ob = toVec(o, b);
51    return acos(dot(oa, ob) / sqrt(norm_sq(oa) * norm_sq(ob))); }
52
53  double cross(vec a, vec b) { return a.x * b.y - a.y * b.x; }
54
55  // note: to accept collinear points, we have to change the '> 0'
56  // returns true if point r is on the left side of line pq
57  bool ccw(point p, point q, point r) {
58    return cross(toVec(p, q), toVec(p, r)) > 0; }
59
60  // returns true if point r is on the same line as the line pq
61  bool collinear(point p, point q, point r) {
62    return fabs(cross(toVec(p, q), toVec(p, r))) < EPS; }
63
64  // returns true if we always make the same turn while examining
65  // all the edges of the polygon one by one
66  bool isConvex(const vector<point> &P) {
67    int sz = (int)P.size();
68    if (sz <= 3) return false;   // a point/sz=2 or a line/sz=3 is not convex
69    bool isLeft = ccw(P[0], P[1], P[2]);               // remember one result
70    for (int i = 1; i < sz-1; i++)             // then compare with the others
71      if (ccw(P[i], P[i+1], P[(i+2) == sz ? 1 : i+2]) != isLeft)
72        return false;               // different sign -> this polygon is concave
73    return true; }                              // this polygon is convex
74
```

```
75    // returns true if point p is in either convex/concave polygon P
76    bool inPolygon(point pt, const vector<point> &P) {
77      if ((int)P.size() == 0) return false;
78      double sum = 0;      // assume the first vertex is equal to the last vertex
79      for (int i = 0; i < (int)P.size()-1; i++) {
80        if (ccw(pt, P[i], P[i+1]))
81            sum += angle(P[i], pt, P[i+1]);                    // left turn/ccw
82        else sum -= angle(P[i], pt, P[i+1]); }                 // right turn/cw
83      return fabs(fabs(sum) - 2*PI) < EPS; }
84
85    // line segment p-q intersect with line A-B.
86    point lineIntersectSeg(point p, point q, point A, point B) {
87      double a = B.y - A.y;
88      double b = A.x - B.x;
89      double c = B.x * A.y - A.x * B.y;
90      double u = fabs(a * p.x + b * p.y + c);
91      double v = fabs(a * q.x + b * q.y + c);
92      return point((p.x * v + q.x * u) / (u+v), (p.y * v + q.y * u) / (u+v)); }
93
94    // cuts polygon Q along the line formed by point a -> point b
95    // (note: the last point must be the same as the first point)
96    vector<point> cutPolygon(point a, point b, const vector<point> &Q) {
97      vector<point> P;
98      for (int i = 0; i < (int)Q.size(); i++) {
99        double left1 = cross(toVec(a, b), toVec(a, Q[i])), left2 = 0;
100       if (i != (int)Q.size()-1) left2 = cross(toVec(a, b), toVec(a, Q[i+1]));
101       if (left1 > -EPS) P.push_back(Q[i]);         // Q[i] is on the left of ab
102       if (left1 * left2 < -EPS)         // edge (Q[i], Q[i+1]) crosses line ab
103         P.push_back(lineIntersectSeg(Q[i], Q[i+1], a, b));
104     }
105     if (!P.empty() && !(P.back() == P.front()))
106       P.push_back(P.front());         // make P's first point = P's last point
107     return P; }
108
109   point pivot;
110   bool angleCmp(point a, point b) {                    // angle-sorting function
111     if (collinear(pivot, a, b))                              // special case
112       return dist(pivot, a) < dist(pivot, b);      // check which one is closer
113     double d1x = a.x - pivot.x, d1y = a.y - pivot.y;
114     double d2x = b.x - pivot.x, d2y = b.y - pivot.y;
115     return (atan2(d1y, d1x) - atan2(d2y, d2x)) < 0; }    // compare two angles
116
117   vector<point> CH(vector<point> P) {    // the content of P may be reshuffled
118     int i, j, n = (int)P.size();
119     if (n <= 3) {
120       if (!(P[0] == P[n-1])) P.push_back(P[0]); // safeguard from corner case
121       return P;                             // special case, the CH is P itself
122     }
123
124     // first, find P0 = point with lowest Y and if tie: rightmost X
125     int P0 = 0;
126     for (i = 1; i < n; i++)
127       if (P[i].y < P[P0].y || (P[i].y == P[P0].y && P[i].x > P[P0].x))
128         P0 = i;
129
130     point temp = P[0]; P[0] = P[P0]; P[P0] = temp;     // swap P[P0] with P[0]
131
132     // second, sort points by angle w.r.t. pivot P0
133     pivot = P[0];                         // use this global variable as reference
134     sort(++P.begin(), P.end(), angleCmp);             // we do not sort P[0]
135
136     // third, the ccw tests
137     vector<point> S;
138     S.push_back(P[n-1]); S.push_back(P[0]); S.push_back(P[1]);   // initial S
139     i = 2;                                    // then, we check the rest
140     while (i < n) {                // note: N must be >= 3 for this method to work
141       j = (int)S.size()-1;
142       if (ccw(S[j-1], S[j], P[i])) S.push_back(P[i++]);  // left turn, accept
143       else S.pop_back(); }    // or pop the top of S until we have a left turn
144     return S; }
145                                           // return the result
146   void init()
147   {
148     freopen("in.txt","r",stdin);
149     freopen("out.txt","w",stdout);
150     cout << "[FREOPEN]" << endl;
151     return;
```

```cpp
152    }
153
154    int main()
155    {
156      init();
157      // 6 points, entered in counter clockwise order, 0-based indexing
158      vector<point> P;
159      P.push_back(point(1, 1));
160      P.push_back(point(3, 3));
161      P.push_back(point(9, 1));
162      P.push_back(point(12, 4));
163      P.push_back(point(9, 7));
164      P.push_back(point(1, 7));
165      P.push_back(P[0]); // loop back
166
167      printf("Perimeter_of_polygon_=_%.2f\n", perimeter(P)); // 31.64
168      printf("Area_of_polygon_=_%.2f\n", area(P)); // 49.00
169      printf("Is_convex_=_%d\n", isConvex(P)); // false (P1 is the culprit)
170
171      //// the positions of P6 and P7 w.r.t the polygon
172      //7 P5----------------P4
173      //6 |                   \
174      //5 |                    \
175      //4 |    P7              P3
176      //3 |    P1___            /
177      //2 | / P6     \ ___     /
178      //1 P0               P2
179      //0 1 2 3 4 5 6 7 8 9 101112
180      point P6(3, 2); // outside this (concave) polygon
181      printf("Point_P6_is_inside_this_polygon_=_%d\n", inPolygon(P6, P)); // false
182      point P7(3, 4); // inside this (concave) polygon
183      printf("Point_P7_is_inside_this_polygon_=_%d\n", inPolygon(P7, P)); // true
184      // cutting the original polygon based on line P[2] -> P[4] (get the left side)
185      //7 P5----------------P4
186      //6 |                |   \
187      //5 |                |     \
188      //4 |                |      P3
189      //3 |    P1___       |     /
190      //2 | /        \ ___ |   /
191      //1 P0               P2
192      //0 1 2 3 4 5 6 7 8 9 101112
193      // new polygon (notice the index are different now):
194      //7 P4----------------P3
195      //6 |                |
196      //5 |                |
197      //4 |                |
198      //3 |    P1___       |
199      //2 | /        \ ___ |
200      //1 P0               P2
201      //0 1 2 3 4 5 6 7 8 9
202      P = cutPolygon(P[2], P[4], P);
203      printf("Perimeter_of_polygon_=_%.2f\n", perimeter(P)); // smaller now 29.15
204      printf("Area_of_polygon_=_%.2f\n", area(P)); // 40.00
205      // running convex hull of the resulting polygon (index changes again)
206      //7 P3----------------P2
207      //6 |                |
208      //5 |                |
209      //4 |    P7          |
210      //3 |                |
211      //2 |                |
212      //1 P0----------------P1
213      //0 1 2 3 4 5 6 7 8 9
214      P = CH(P); // now this is a rectangle
215      for(int i=0; i<P.size(); i++)
216        printf("%.0f_%.0f\n", P[i].x, P[i].y);
217      printf("Perimeter_of_polygon_=_%.2f\n", perimeter(P)); // precisely 28.00
218      printf("Area_of_polygon_=_%.2f\n", area(P)); // precisely 48.00
219      printf("Is_convex_=_%d\n", isConvex(P)); // true
220      printf("Point_P6_is_inside_this_polygon_=_%d\n", inPolygon(P6, P)); // true
221      printf("Point_P7_is_inside_this_polygon_=_%d\n", inPolygon(P7, P)); // true
222
223      return 0;
224    }
```

6.1.4 Point Inside Polygon

```cpp
#include <bits/stdc++.h>

using namespace std;

int nn, mm;

struct pt {
    int x, y;
};

struct ang {
    int a, b;
};

bool operator < (const ang & p, const ang & q) {
    if (p.b == 0 && q.b == 0)
        return p.a < q.a;
    return p.a * 1ll * q.b < p.b * 1ll * q.a;
}

long long sq (pt & a, pt & b, pt & c) {
    return a.x*1ll*(b.y-c.y) + b.x*1ll*(c.y-a.y) + c.x*1ll*(a.y-b.y);
}

bool solve(vector<pt> poly, vector<pt> points, int n, int m)
{
    vector<pt> p(n);
    int zero_id = 0;
    for (int i=0; i<n; ++i) {
        p[i].x = poly[i].x;
        p[i].y = poly[i].y;
        if (p[i].x < p[zero_id].x || p[i].x == p[zero_id].x && p[i].y < p[zero_id].y)
            zero_id = i;
    }
    pt zero = p[zero_id];
    rotate (p.begin(), p.begin()+zero_id, p.end());
    p.erase (p.begin());
    --n;

    vector<ang> a (n);
    for (int i=0; i<n; ++i) {
        a[i].a = p[i].y - zero.y;
        a[i].b = p[i].x - zero.x;
        if (a[i].a == 0)
            a[i].b = a[i].b < 0 ? -1 : 1;
    }
    for (int i = 0; i < points.size(); ++i) {
        pt q;
        q.x = points[i].x;
        q.y = points[i].y;
        bool in = false;
        if (q.x >= zero.x)
            if (q.x == zero.x && q.y == zero.y)
                in = true;
            else {
                ang my = { q.y-zero.y, q.x-zero.x };
                if (my.a == 0)
                    my.b = my.b < 0 ? -1 : 1;
                vector<ang>::iterator it = upper_bound (a.begin(), a.end(), my);
                if (it == a.end() && my.a == a[n-1].a && my.b == a[n-1].b)
                    it = a.end()-1;
                if (it != a.end() && it != a.begin()) {
                    int p1 = int (it - a.begin());
                    if (sq (p[p1], p[p1-1], q) < 0)
                        in = true;
                }
            }
        if(in == false)
            return false;
    }
    return true;
}

int main() {
```

93

```
75
76
77        cin >> nn >> mm;
78        vector<pt> firstpoints(nn), secondpoints(mm);
79        for(int i = 0; i < nn; ++i)
80            cin >> firstpoints[i].x >> firstpoints[i].y;
81        for(int i = 0; i < mm; ++i)
82            cin >> secondpoints[i].x >> secondpoints[i].y;
83
84        reverse(firstpoints.begin(), firstpoints.end());
85        reverse(secondpoints.begin(), secondpoints.end());
86        bool a1 = solve(firstpoints, secondpoints, nn, mm);
87        bool a2 = solve(secondpoints, firstpoints, mm, nn);
88        if(a1 || a2)
89            printf("YES\n");
90        else
91            printf("NO\n");
92    }
```

6.1.5 Circle Line Intersection

```
1   double r, a, b, c; // ŔšŃ̃ŔĐ¿ĐřĐiŃŃĐţ ĐřĐřĐiĐiŃŃĐţ
2
3   double x0 = −a∗c/(a∗a+b∗b),   y0 = −b∗c/(a∗a+b∗b);
4   if (c∗c > r∗r∗(a∗a+b∗b)+EPS)
5       puts ("no␣points");
6   else if (abs (c∗c − r∗r∗(a∗a+b∗b)) < EPS) {
7       puts ("1␣point");
8       cout << x0 << '␣' << y0 << '\n';
9   }
10  else {
11      double d = r∗r − c∗c/(a∗a+b∗b);
12      double mult = sqrt (d / (a∗a+b∗b));
13      double ax,ay,bx,by;
14      ax = x0 + b ∗ mult;
15      bx = x0 − b ∗ mult;
16      ay = y0 − a ∗ mult;
17      by = y0 + a ∗ mult;
18      puts ("2␣points");
19      cout << ax << '␣' << ay << '\n' << bx << '␣' << by << '\n';
20  }
```

6.1.6 Larges Circle Inside Convex Polygon

```cpp
const double EPS = 1E-9;
const double PI = ...;

struct pt {
    double x, y;
    pt() { }
    pt (double x, double y) : x(x), y(y) { }
    pt operator- (const pt & p) const {
        return pt (x-p.x, y-p.y);
    }
};

double dist (const pt & a, const pt & b) {
    return sqrt ((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}

double get_ang (const pt & a, const pt & b) {
    double ang = abs (atan2 (a.y, a.x) - atan2 (b.y, b.x));
    return min (ang, 2*PI-ang);
}

struct line {
    double a, b, c;
    line (const pt & p, const pt & q) {
        a = p.y - q.y;
        b = q.x - p.x;
        c = - a * p.x - b * p.y;
        double z = sqrt (a*a + b*b);
        a/=z, b/=z, c/=z;
    }
};

double det (double a, double b, double c, double d) {
    return a * d - b * c;
}

pt intersect (const line & n, const line & m) {
    double zn = det (n.a, n.b, m.a, m.b);
    return pt (
        - det (n.c, n.b, m.c, m.b) / zn,
        - det (n.a, n.c, m.a, m.c) / zn
    );
}

bool parallel (const line & n, const line & m) {
    return abs (det (n.a, n.b, m.a, m.b)) < EPS;
}

double get_h (const pt & p1, const pt & p2,
    const pt & l1, const pt & l2, const pt & r1, const pt & r2)
{
    pt q1 = intersect (line (p1, p2), line (l1, l2));
    pt q2 = intersect (line (p1, p2), line (r1, r2));
    double l = dist (q1, q2);
    double alpha = get_ang (l2 - l1, p2 - p1) / 2;
    double beta = get_ang (r2 - r1, p1 - p2) / 2;
    return l * sin(alpha) * sin(beta) / sin(alpha+beta);
}

struct cmp {
    bool operator() (const pair<double,int> & a, const pair<double,int> & b) const {
        if (abs (a.first - b.first) > EPS)
            return a.first < b.first;
        return a.second < b.second;
    }
};

int main() {
    int n;
    vector<pt> p;
    ... ŃĞŃĆĐţĐiĐÿĐţ n Đÿ p ...

    vector<int> next (n), prev (n);
    for (int i=0; i<n; ++i) {
```

```
75          next[i] = (i + 1) % n;
76          prev[i] = (i - 1 + n) % n;
77      }
78
79      set < pair<double,int>, cmp > q;
80      vector<double> h (n);
81      for (int i=0; i<n; ++i) {
82          h[i] = get_h (
83              p[i], p[next[i]],
84              p[i], p[prev[i]],
85              p[next[i]], p[next[next[i]]]
86          );
87          q.insert (make_pair (h[i], i));
88      }
89
90      double last_time;
91      while (q.size () > 2) {
92          last_time = q.begin()->first;
93          int i = q.begin()->second;
94          q.erase (q.begin());
95
96          next[prev[i]] = next[i];
97          prev[next[i]] = prev[i];
98          int nxt = next[i],    nxt1 = (nxt+1)%n,
99              prv = prev[i],    prv1 = (prv+1)%n;
100         if (parallel (line (p[nxt], p[nxt1]), line (p[prv], p[prv1])))
101             break;
102
103         q.erase (make_pair (h[nxt], nxt));
104         q.erase (make_pair (h[prv], prv));
105
106         h[nxt] = get_h (
107             p[nxt], p[nxt1],
108             p[prv1], p[prv],
109             p[next[nxt]], p[(next[nxt]+1)%n]
110         );
111         h[prv] = get_h (
112             p[prv], p[prv1],
113             p[(prev[prv]+1)%n], p[prev[prv]],
114             p[nxt], p[nxt1]
115         );
116
117         q.insert (make_pair (h[nxt], nxt));
118         q.insert (make_pair (h[prv], prv));
119     }
120
121     cout << last_time << endl;
122 }
```

6.1.7 All nearest pair of points

Call:
$sort(a, a + n, \&cmp\_x);$
$mindist = 1E20;$
$rec(0, n - 1);$

```cpp
1    struct pt {
2        int x, y, id;
3    };
4
5    inline bool cmp_x (const pt & a, const pt & b) {
6        return a.x < b.x || a.x == b.x && a.y < b.y;
7    }
8
9    inline bool cmp_y (const pt & a, const pt & b) {
10       return a.y < b.y;
11   }
12
13   pt a[MAXN];
14
15   double mindist;
16   int ansa, ansb;
17
18   inline void upd_ans (const pt & a, const pt & b) {
19       double dist = sqrt ((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y) + .0);
20       if (dist < mindist)
21           mindist = dist,  ansa = a.id,  ansb = b.id;
22   }
23
24   void rec (int l, int r) {
25       if (r - l <= 3) {
26           for (int i=l; i<=r; ++i)
27               for (int j=i+1; j<=r; ++j)
28                   upd_ans (a[i], a[j]);
29           sort (a+l, a+r+1, &cmp_y);
30           return;
31       }
32
33       int m = (l + r) >> 1;
34       int midx = a[m].x;
35       rec (l, m),  rec (m+1, r);
36       static pt t[MAXN];
37       merge (a+l, a+m+1, a+m+1, a+r+1, t, &cmp_y);
38       copy (t, t+r-l+1, a+l);
39
40       int tsz = 0;
41       for (int i=l; i<=r; ++i)
42           if (abs (a[i].x - midx) < mindist) {
43               for (int j=tsz-1; j>=0 && a[i].y - t[j].y < mindist; --j)
44                   upd_ans (a[i], t[j]);
45               t[tsz++] = a[i];
46           }
47   }
```

6.2 3d

6.2.1 Point Template

```
1   #define vetor point
2
3   // FORMULAS.
4   //   vetores a,b; a*b = a.mod()*b.mod()*cos( angulo entre a e b) =>
5   // a*b = |a|*|b|*cos(t)
6   //   vetores a,b; (a^b).mod() = a.mod()*b.mod()*sin( angulo entre a e b)
7
8   inline int cmp(ld x, ld y = 0, ld tol = eps)
9   {
10      return (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
11  }
12  struct point
13  {
14      ld x, y, z;
15      point(ld x = 0, ld y = 0, ld z = 0): x(x), y(y), z(z) {}
16      point operator +(point q) { return point(x + q.x, y + q.y, z + q.z); }
17      point operator -(point q) { return point(x - q.x, y - q.y, z - q.z); }
18      point operator *(ld t) { return point(x * t, y * t, z * t); }
19      point operator /(ld t) { return point(x / t, y / t, z / t); }
20      point operator ^(point q) {
21          return point(y*q.z - z*q.y, z*q.x - x*q.z, x*q.y - y*q.x); }
22      ld operator *(point q) { return x * q.x + y * q.y + z * q.z; }
23      ld mod() { return sqrt(x * x + y * y + z * z); }
24      ld mod2() { return x * x + y * y + z * z; }
25      point projecao(vetor u) { return (*this) * ((*this)*u) / ((*this)*(*this)); }
26
27      int cmp(point q) const
28      {
29          if (int t = ::cmp(x, q.x)) return t;
30          if (int t = ::cmp(y, q.y)) return t;
31          return ::cmp(z, q.z);
32      }
33      bool operator ==(point q) const { return cmp(q) == 0; }
34      bool operator !=(point q) const { return cmp(q) != 0; }
35      bool operator < (point q) const { return cmp(q) < 0; }
36  };
37
38  // RETAS, SEMIRETAS, SEGMENTOS E TRIANGULOS
39  struct reta
40  {
41      point a, b;// <---a----b--->
42      reta(point A=point(0,0,0), point B=point(0,0,0)): a(A), b(B) { }
43
44      //verifica se o ponto p esta na reta ab
45      bool belongs(point p)
46      {
47          return cmp(((a-p)^(b-p)).mod()) == 0;
48      }
49  };
50  struct semireta
51  {
52      point a, b; // |a----b--->
53      semireta(point A=point(0,0,0), point B=point(0,0,0)): a(A), b(B) { }
54  };
55  struct segmento
56  {
57      point a, b; // |a----b|
58      segmento(point A=point(0,0,0), point B=point(0,0,0)): a(A), b(B) { }
59      bool between(point p) {
60          return cmp(((a-p)^(b-p)).mod()) == 0 && cmp((a-p) * (b-p)) <= 0;
61      }
62  };
63  struct triangulo
64  {
65      point a, b, c;
66      triangulo(point A, point B, point C): a(A), b(B), c(C) { }
67      ld area() { return 0.5*((b-a)^(c-a)).mod(); }
68
69      //retorna o ponto que eh a projecao de p no plano abc
70      point projecao(point p)
71      {
```

```cpp
72              vetor w = (b-a)^(c-a);
73              return p - w.projecao(p-a);
74          }
75          //verifica se p esta dentro de abc
76          // se retornar true, entao a,b,c,p sao coplanares
77          bool inside(point p)
78          {
79              return cmp(((p-a)^(b-a)).mod() +
80                          ((p-b)^(c-b)).mod() +
81                          ((p-c)^(a-c)).mod() -
82                          ((b-a)^(c-a)).mod()) == 0;
83          }
84  };
85
86  //Produto misto
87  ld produto_misto(point p, point q, point r)
88  {
89      return (p^q)*r;
90  }
91  //Volume do tetraedro pqrs
92  ld volume(point p, point q, point r, point s)
93  {
94      return fabs(produto_misto(q-p, r-p, s-p)) / 6.0;
95  }
96
97  // DISTANCIA ENTRE OBJETOS GEOMETRICOS
98  ld distancia(point p, reta r)
99  {
100      vetor v = r.b-r.a, w = p-r.a;
101      return (v^w).mod() / v.mod();
102  }
103  ld distancia(point p, semireta s)
104  {
105      vetor v = s.b-s.a, w = p-s.a;
106      if (cmp(v*w) <= 0) return (p-s.a).mod();
107      return (v^w).mod() / v.mod();
108  }
109  ld distancia(point p, segmento s)
110  {
111      point proj = s.a + (s.b-s.a).projecao(p-s.a);
112      if (segmento(s.a,s.b).between(proj))
113          return (p-proj).mod();
114      return min((p-s.a).mod(), (p-s.b).mod());
115  }
116  ld distancia(point p, triangulo T)
117  {
118      point proj = T.projecao(p);
119      if (T.inside(proj)) return (p-proj).mod();
120      return min( distancia(p, segmento(T.a, T.b)),
121                  min(distancia(p, segmento(T.b, T.c)),
122                      distancia(p, segmento(T.c, T.a))));
123  }
124  ld distancia(reta r, reta s)
125  {
126      vetor u = r.b-r.a, v = s.b-s.a, w = s.a-r.a;
127      ld a = u*u, b = u*v, c = v*v, d = u*w, e = v*w;
128      ld D = a*c - b*b, sc, tc;
129      if (D < eps)
130      {
131          sc = 0;
132          tc = (b > c) ? d/b : e/c;
133      }
134      else
135      {
136          sc = (b*e - c*d) / D;
137          tc = (a*e - b*d) / D;
138      }
139      vetor dP = w + (u * sc) - (v * tc);
140      return dP.mod();
141  }
142  ld distancia(segmento X, segmento Y)
143  {
144      point p = X.a, q = X.b;
145      point r = Y.a, s = Y.b;
146      if (p == q) return distancia(p, Y);
147      if (r == s) return distancia(r, X);
148      if (cmp(((p-q)^(s-r)).mod()) == 0)
```

```cpp
149            return min( min(distancia(p,Y),distancia(q,Y)),
150                        min(distancia(p,Y),distancia(q,Y)));
151        vetor v = q-p, u = s-r, t = (r-p);
152        ld b = ((t*v)*(v*u) - (t*u)*(v*v)) / ((u*u)*(v*v) - (u*v)*(v*u));
153        ld a = (b*(u*v) + t*v) / (v*v);
154        if (cmp(a) >= 0 && cmp(a,1.0) <= 0 && cmp(b) >= 0 && cmp(b,1.0) <= 0)
155            return ((p+v*a) - (r+u*b)).mod();
156        point ini = ((cmp(a) < 0)?p:q);
157        point fim = ((cmp(b) < 0)?r:s);
158        return (ini-fim).mod();
159    }
160
161    //Calcula o centro da esfera circunscrita de uma piramide triangular
162    point circumsphere(point p, point q, point r, point s)
163    {
164        point a = q-p, b = r-p, c = s-p;
165        return p + ((a^b)*c.mod2() + (c^a)*b.mod2() + (b^c)*a.mod2()) / (a*(b^c)*2);
166    }
167
168    //Calcula o circuncentro de um triangulo no espaco
169    point circumcenter(point p, point q, point r)
170    {
171        point a = (q-p)^((q-p)^(r-p)), b = (r-p)^((q-p)^(r-p)); ld t;
172        if (fabs(a.x) < eps) t = (r.x-q.x)/2/b.x;
173        else if (fabs(a.y) < eps) t = (r.y-q.y)/2/b.y;
174        else if (fabs(a.z) < eps) t = (r.z-q.z)/2/b.z;
175        else
176        {
177            t = a.x*(r.y-q.y) - a.y*(r.x-q.x);
178            t = t / (2*a.y*b.x - 2*a.x*b.y);
179        }
180        return (p+q)/2 + a*t;
181    }
182
183    //Verifica se T[a], T[b], T[c] eh face do convex hull
184    //OBS.: Cuidade com mais de 3 pontos coplanares
185    bool ishullface(vector <point> &T, int a, int b, int c)
186    {//TODO testar
187        int n = (int)T.size(), pos = 0, neg = 0;
188        for (int i = 0; i < n; i++)
189        {
190            ld pm = produto_misto(T[b]-T[a], T[c]-T[a], T[i]-T[a]);
191            if (cmp(pm) < 0) neg++;
192            if (cmp(pm) > 0) pos++;
193        }
194        return (neg*pos == 0);
195    }
```

6.3 Convex Hull

6.3.1 Graham Scan

```
1   struct pt {
2       double x, y;
3   };
4
5   bool cmp (pt a, pt b) {
6       return a.x < b.x || a.x == b.x && a.y < b.y;
7   }
8
9   bool cw (pt a, pt b, pt c) {
10      return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y) < 0;
11  }
12
13  bool ccw (pt a, pt b, pt c) {
14      return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y) > 0;
15  }
16
17  void convex_hull (vector<pt> & a) {
18      if (a.size() == 1) return;
19      sort (a.begin(), a.end(), &cmp);
20      pt p1 = a[0], p2 = a.back();
21      vector<pt> up, down;
22      up.push_back (p1);
23      down.push_back (p1);
24      for (size_t i=1; i<a.size(); ++i) {
25          if (i==a.size()-1 || cw (p1, a[i], p2)) {
26              while (up.size()>=2 && !cw (up[up.size()-2], up[up.size()-1], a[i]))
27                  up.pop_back();
28              up.push_back (a[i]);
29          }
30          if (i==a.size()-1 || ccw (p1, a[i], p2)) {
31              while (down.size()>=2 && !ccw (down[down.size()-2], down[down.size()-1], a[i]))
32                  down.pop_back();
33              down.push_back (a[i]);
34          }
35      }
36      a.clear();
37      for (size_t i=0; i<up.size(); ++i)
38          a.push_back (up[i]);
39      for (size_t i=down.size()-2; i>0; --i)
40          a.push_back (down[i]);
41  }
```

6.3.2 Monotone Chain

Use 2d point template

```
polygon convexHull(polygon T)
{
    int n=T.size(), k=0;
    polygon H(2*n);

    sort(T.begin(), T.end());
    //lower_hull
    for(int i=0; i<n; i++)
    {
        while(k>=2 && ccw(H[k-1], T[i], H[k-2])<=0)
            k--;
        H[k++]=T[i];
    }
    //upper_hull
    for(int i=n-2, t=k+1; i>=0; i--)
    {
        while(k>=t && ccw(H[k-1], T[i], H[k-2])<=0)
            k--;
        H[k++]=T[i];
    }
    H.resize(k);
    return H;
}
```

6.4 Rotating Calipers

Only work on clockwise(or anticlockwise) ordered polygons.

```
 1  double minimumWidth(polygon &ch)
 2  {
 3      double ret=DINF;
 4      int sz=ch.size();
 5      int j=1;
 6      for(int i=0; i<sz; i++)
 7      {
 8          int nxt=(j+1)%sz;
 9          while(distPointLine(ch[i], ch[(i+1)%sz], ch[j])<distPointLine(ch[i], ch[(i+1)%sz], ch[nxt]))
10          {
11              j=(j+1)%sz;
12              nxt=(j+1)%sz;
13          }
14          ret=min(ret, distPointLine(ch[i], ch[(i+1)%sz], ch[j]));
15      }
16      return ret;
17  }
```

6.5 KD Tree

```cpp
1   struct point
2   {
3       ll x, y, z;
4       point(ll x=0, ll y=0, ll z=0): x(x), y(y), z(z) {}
5       point operator-(point q) { return point(x-q.x, y-q.y, z-q.z); }
6       ll operator*(point q) { return x*q.x + y*q.y + z*q.z; }
7   };
8   typedef vector<point> polygon;
9
10  struct KDTreeNode
11  {
12      point p;
13      int level;
14      KDTreeNode *below, *above;
15
16      KDTreeNode (const point& q, int levl)
17      {
18          p = q;
19          level = levl;
20          below = above = 0;
21      }
22      ~KDTreeNode() { delete below, above; }
23
24      int diff (const point& pt)
25      {
26          switch (level)
27          {
28          case 0: return pt.x - p.x;
29          case 1: return pt.y - p.y;
30          case 2: return pt.z - p.z;
31          }
32          return 0;
33      }
34      ll distSq (point& q) { return (p-q)*(p-q); }
35
36      int rangeCount (point& pt, ll K)
37      {
38          int count = (distSq(pt) < K*K) ? 1 : 0;
39          int d = diff(pt);
40          if (-d <= K && above != 0)
41              count += above->rangeCount(pt, K);
42          if (d <= K && below != 0)
43              count += below->rangeCount(pt, K);
44          return count;
45      }
46  };
47
48  class KDTree
49  {
50  public:
51      polygon P;
52      KDTreeNode *root;
53      int dimention;
54      KDTree() {}
55      KDTree(polygon &poly, int D)
56      {
57          P = poly;
58          dimention = D;
59          root = 0;
60          build();
61      }
62      ~KDTree() { delete root; }
63
64      //count the number of pairs that has a distance less than K
65      ll countPairs(ll K)
66      {
67          ll count = 0;
68          f(i, 0, P.size())
69              count += root->rangeCount(P[i], K) - 1;
70          return count;
71      }
72
73  protected:
74      void build()
```

```cpp
75              {
76                  random_shuffle(all(P));
77                  f(i, 0, P.size()) {
78                      root = insert(root, P[i], -1);
79                  }
80              }
81          KDTreeNode *insert(KDTreeNode* t, const point& pt, int parentLevel)
82              {
83                  if (t == 0)
84                  {
85                      t = new KDTreeNode (pt, (parentLevel+1) % dimention);
86                      return t;
87                  }
88                  else
89                  {
90                      int d = t->diff(pt);
91                      if (d <= 0) t->below = insert (t->below, pt, t->level);
92                      else t->above = insert (t->above, pt, t->level);
93                  }
94                  return t;
95              }
96      };
97
98      int main()
99      {
100         int n, k;
101         point e;
102         polygon p;
103         while (cin >> n >> k && n+k)
104         {
105             p.clear();
106             f(i, 0, n)
107             {
108                 cin >> e.x >> e.y >> e.z;
109                 p.pb(e);
110             }
111             KDTree tree(p, 3);
112             cout << tree.countPairs(k) / 2LL << endl;
113         }
114         return 0;
115     }
```

Chapter 7

Misc

7.1 Josephus

```
//O(n)
int joseph (int n, int k) {
    int res = 0;
    for (int i=1; i<=n; ++i)
        res = (res + k) % i;
    return res + 1;
}
//O(klogn)
int joseph (int n, int k) {
    if (n == 1)  return 0;
    if (k == 1)  return n-1;
    if (k > n)  return (joseph (n-1, k) + k) % n;
    int cnt = n / k;
    int res = joseph (n - cnt, k);
    res -= n % k;
    if (res < 0)  res += n;
    else  res += res / (k - 1);
    return res;
}
```

Chapter 8

Templates

8.1 C++

```
1   /// David Mateus Batista <david.batista3010@gmail.com>
2   /// Computer Science − Federal University of Itajuba − Brazil
3
4   #include <bits/stdc++.h>
5
6   using namespace std;
7
8   typedef long long ll;
9   typedef unsigned long long ull;
10  typedef long double ld;
11  typedef pair<int,int> pii;
12  typedef pair<ll,ll> pll;
13
14  #define INF 0x3F3F3F3F
15  #define LINF 0x3F3F3F3F3F3F3F3FLL
16  #define DINF (double)1e+30
17  #define EPS (double)1e−9
18  #define PI (double)acos(−1.0)
19  #define RAD(x) (double)(x∗PI)/180.0
20  #define PCT(x,y) (double)x∗100.0/y
21
22  #define pb push_back
23  #define mp make_pair
24  #define pq priority_queue
25  #define F first
26  #define S second
27
28  #define D(x) x&(−x)
29  #define SZ(x) (int)x.size()
30  #define ALL(x) x.begin(),x.end()
31  #define SET(a,b) memset(a, b, sizeof(a))
32
33  #define gcd(x,y) __gcd(x, y)
34  #define lcm(x,y) (x/gcd(x,y))∗y
35
36  #define bitcnt(x) __builtin_popcountll(x)
37  #define lbit(x) 63−__builtin_clzll(x)
38  #define zerosbitll(x) __builtin_ctzll(x)
39  #define zerosbit(x) __builtin_ctz(x)
40
41  enum {North, East, South, West};
42  //{0, 1, 2, 3}
43  //{Up, Right, Down, Left}
44
45  int mi[] = {−1, 0, 1, 0, −1, 1, 1, −1};
46  int mj[] = {0, 1, 0, −1, 1, 1, −1, −1};
47
48  inline void solve()
49  {
50
51  }
52
53  template<class num>inline void rd(num &x)
```

```
54    {
55        char c;
56        while(isspace(c = getchar()));
57        bool neg = false;
58        if(!isdigit(c))
59            neg=(c=='-'), x=0;
60        else
61            x=c-'0';
62        while(isdigit(c=getchar()))
63            x=(x<<3)+(x<<1)+c-'0';
64        if(neg)
65            x=-x;
66    }
67
68    int main()
69    {
70        #ifdef LOCAL_PROJECT
71        freopen("in.txt","r",stdin);
72        freopen("out.txt","w",stdout);
73        #else
74        #endif
75
76        solve();
77        return 0;
78    }
```

8.2 Time Check

```
1   using namespace std::chrono;
2   class timecheck
3   {
4   public:
5       high_resolution_clock::time_point t1, t2;
6       void start()
7       {
8           t1 = high_resolution_clock::now();
9       }
10      void end()
11      {
12          t2= high_resolution_clock::now();
13          duration<double> time_span = duration_cast<duration<double>>(t2 - t1);
14          cout << "Time: " << time_span.count() << "s" << endl;
15      }
16  };
```

8.3 Rand

```
1   inline ll bigRand(ll mod)//[0, mod]
2   {
3       mod--;
4       ll ret=rand()*rand();
5       return (abs(ret<<(rand()%32))%mod)+(rand()%2);
6   }
7
8   inline int smallRand(int mod)//[0, mod]
9   {
10      mod--;
11      int ret=rand()*rand();
12      return (abs(ret<<(rand()%20))%mod)+(rand()%2);
13  }
```

8.4 Java

```java
/// David Mateus Batista <david.batista3010@gmail.com>
/// Computer Science − Federal University of Itajuba − Brazil

import java.io.*;
import java.util.*;
import java.math.*;

public class Main
{
    static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    static Scanner sc = new Scanner(System.in);

    public static void main(String[] args) throws Exception
    {

    }
}
```

Chapter 9

Formulas

9.1   Combinatorics

- Symmetry rule: $C_n^k = C_{n-k}^k$
- Entry-Excerpt: $C_n^k = \dfrac{n}{k} * C_{n-1}^{k-1}$
- Summation by k: $\sum_{k=0}^{n} C_n^k = 2^n$
- Summation by n: $\sum_{m=0}^{n} C_m^k = C_{n+1}^{k+1}$
- Summation over n and k: $\sum_{k=0}^{m} C_{n+k}^k = C_{n+m+1}^m$
- Summation of squares: $(C_n^0)^2 + (C_n^1)^2 + ... + (C_n^n)^2 = C_{2n}^n$
- Weighted summation: $1C_n^1 + 2C_n^2 + ... + nC_n^n = n2^{n-1}$
- Contact with Fibonacci numbers: $C_n^0 + C_{n-1}^1 + ... + C_{n-k}^k + ... + C_0^n = Fib_{n+1}$

9.2 Catalan Numbers

- $C_{n,m} = \binom{n+m}{n}$
- $C_n = \binom{2n}{n}$