ACM ICPC - Code Notebook

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

May 3, 2017

Contents

Chapter 1

Data Structure

1.1 Segment Tree

1.1.1 Segment Tree & Lazy Propagation

```
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

```
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

```
/*
call build(1, 0, n-1, 0) & build(1, 0, n-1, 1)
to build base graph on tree

update(1, 0, n-1, l, r, x, 0):
add vertex [x,x]->[l, r]

update(1, 0, n-1, l, r, x, 1):
add vertex [l,r]->[x,x]
*/
class segtree
{
    const static int N=1e+5+35;
public:
    vector< pair<int,int> >data[8*N]; //graph
    int idx[4*N][2], id;
    segtree(){};
    void set(int n)
    {
        id=n;
        for(int i=0; i<8*n; i++)
            data[i].clear();
    }
    inline void addEdge(int u, int v, int w)
    {
        data[u].pb({v, w});
    }
    inline void build(int no, int l, int r, int t)
    {
        idx[no][t]=id++;
        if(l==r)
        {
            if(!t)
                addEdge(idx[no][t], l, 0);
            else
                addEdge(l, idx[no][t], 0);
            return;
        }
        int nxt=no<<1;
        int mid=(l+r)>>1;
        build(nxt, l, mid, t);
        build(nxt+1, mid+1, r, t);
        if(!t)
        {
            addEdge(idx[no][t], idx[nxt][t], 0);
            addEdge(idx[no][t], idx[nxt+1][t], 0);
        }
        else
        {
            addEdge(idx[nxt][t], idx[no][t], 0);
            addEdge(idx[nxt+1][t], idx[no][t], 0);
        }
    }
    inline void update(int no, int l, int r, int i, int j, int u, int w, int t)
    {
        if(l>j || r<i)
            return;
        if(l>=i && r<=j)
        {
            if(!t)
                addEdge(u, idx[no][t], w);
            else
                addEdge(idx[no][t], u, w);
            return;
        }
        int nxt=no<<1;
        int mid=(l+r)>>1;
        update(nxt, l, mid, i, j, u, w, t);
        update(nxt+1, mid+1, r, i, j, u, w, t);
    }
};
```

1.1.4 Quadtree

```
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);
    }
};
```

1.1.5 Mergesort Segtree

```
class mergesort_segtree
{
    const static int N=100000;
    vector<int>tr[4*N];
public:
    mergesort_segtree(){};
    void build(int no, int l, int r, vector<int>&data)
    {
        if(l==r)
        {
            tr[no].push_back(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].resize(tr[nxt].size()+tr[nxt+1].size());
        merge(tr[nxt].begin(), tr[nxt].end(), tr[nxt+1].begin(), tr[nxt+1].end(), tr[no].begin());
    }
    //how many numbers in (i, j) are greater or equal than k
    int query(int no, int l, int r, int i, int j, int k)
    {
        if(r<i || l>j)
            return 0;
        if(l>=i && r<=j)
            return (int)(tr[no].end()-upper_bound(tr[no].begin(), tr[no].end(), k));
        int nxt=no*2;
        int mid=(l+r)/2;
        int ql=query(nxt, l, mid, i, j, k);
        int qr=query(nxt+1, mid+1, r, i, j, k);
        return ql+qr;
    }
};
```

1.1.6 Persistent Segtree

```
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);
    }
};
```

1.2 Fenwick Tree

1.2.1 Fenwick Tree 1D

```
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

```
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

```
//srand(time(NULL))
int vrand()
{
    return abs(rand()<<(rand()%31));
}

struct node
{
    //x=key, y=priority key, c=tree count
    int x, y, c;
    node *L, *R;
    node(){};
    node(int _x)
    {
        x=_x, y=vrand(), c=0;
        L=R=NULL;
    }
};

int cnt(node *root)
{
    return root?root->c:0;
}

void upd_cnt(node *root)
{
    if(root)
        root->c=1+cnt(root->L)+cnt(root->R);
}

void split(node *root, int x, node *&L, node *&R)
{
    if(!root)
        L=R=NULL;
    else if(x < root->x)
        split(root->L, x, L, root->L), R=root;
    else
        split(root->R, x, root->R, R), L=root;
    upd_cnt(root);
}

void insert(node *&root, node *it)
{
    if(!root)
        root=it;
    else if(it->y > root->y)
        split(root, it->x, it->L, it->R), root=it;
    else
        insert(it->x < root->x? root->L:root->R, it);
    upd_cnt(root);
}

void merge(node *&root, node *L, node *R)
{
    if(!L || !R)
        root=L?L:R;
    else if(L->y > R->y)
        merge(L->R, L->R, R), root=L;
    else
        merge(R->L, L, R->L), root=R;
    upd_cnt(root);
}

void erase(node *&root, int x)
{
    if(root->x==x)
        merge(root, root->L, root->R);
    else
        erase(x < root->x? root->L:root->R, x);
    upd_cnt(root);
}
```

```
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

```
//srand(time(NULL))
int vrand()
{
    return abs(rand()<<(rand()%31));
}

struct node
{
    //basic treap: x=key, y=priority key, c=tree count;
    int x, y, c;
    //treap operations: v=max(x), lazy=lazy value of propagation, rev=reversed
    int v, lazy, rev;

    node *L, *R;
    node(){};
    node(int _x)
    {
        x=_x, y=vrand();
        L=R=NULL;
        v=x;
        lazy=0;
        rev=0;
    }
};

//updating functions
inline int get_cnt(node *root)
{
    return root?root->c:0;
}

inline void upd_cnt(node *root)
{
    if(root)
        root->c=1+get_cnt(root->L)+get_cnt(root->R);
}

inline void push(node *&root)
{
    if(root && root->rev)
    {
        root->rev=0;
        swap(root->L, root->R);
        if(root->L)
            root->L->rev^=1;
        if(root->R)
            root->R->rev^=1;
    }
}

inline void propagate(node *&root)
{
    if(root)
    {
        if(!root->lazy)
            return;
        int lazy=root->lazy;
        root->x+=lazy;

        if(root->L)
            root->L->lazy=lazy;
        if(root->R)
            root->R->lazy=lazy;
        root->lazy=0;
    }
}

inline int get_max(node *root)
{
    return root?root->v:-INF;
}

inline void upd_max(node *root)
{
```

```
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

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

//srand(time(NULL))
int vrand()
{
    return abs(rand()<<(rand()%31));
}
struct node
{
    int x, y, c;
    int lazy, rev;
    hnode v;
    node *L, *R;
    node(){};
    node(int _x)
    {
        x=_x, y=vrand();
        L=R=NULL;
        v=hnode((ull)_x, 1);
        lazy=0;
        rev=0;
    }
};

//updating functions
inline int get_cnt(node *root)
{
    return root?root->c:0;
}

inline void upd_cnt(node *root)
{
    if(root)
    {
        root->c=1+get_cnt(root->L)+get_cnt(root->R);
    }
}

inline void push(node *&root)
{
    if(root && root->rev)
    {
        root->rev=0;
        swap(root->L, root->R);
        if(root->L)
            root->L->rev^=1;
        if(root->R)
            root->R->rev^=1;
    }
}

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

```
#define INF 0x3F3F3F3F
int mergesort(vector<int>&data)
{
    if(data.size()==1)
        return 0;
    vector<int>L, R;
    int t=data.size();
    for(int i=0; i<t/2; i++)
        L.push_back(data[i]);
    for(int i=t/2; i<t; i++)
        R.push_back(data[i]);
    int ret=mergesort(L)+mergesort(R);
    for(int i=0, j=0, k=0; j<L.size() || k<R.size(); i++)
    {
        int x=j<L.size()?L[j]:INF;
        int y=k<R.size()?R[k]:INF;
        if(x<y)
        {
            data[i]=x;
            j++;
        }
        else
        {
            data[i]=y;
            k++;
            ret+=(L.size()-j);
        }
    }
    return ret;
}
```

1.4.2 Merge Sort & Array

```
#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

```
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

```
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

```
#define pb push_back
#define ALL(x) x.begin(),x.end()

const int N=1e+5+35;
const int M=20;
const int SN=sqrt(2*N)+1;

class mo
{
public:
    int l, r, i, lc;
    mo(){};
    mo(int _l, int _r, int _lc, int _i)
    {
        l=_l, r=_r, lc=_lc, 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 n, q;
int h[N], lca[N][M];
vector<int>g[N];
int dl[N], dr[N], di[2*N], cur;

void dfs(int u, int p)
{
    dl[u]=++cur;
    di[cur]=u;
    lca[u][0]=p;
    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];
        if(v==p)
            continue;
        h[v]=h[u]+1;
        dfs(v, u);
    }
    dr[u]=++cur;
    di[cur]=u;
}

inline int getLca(int u, int v)
{
    if(h[u]>h[v])
        swap(u, v);
    for(int i=M-1; i>=0; i--)
        if(h[v]-(1<<i)>=h[u])
            v=lca[v][i];
    if(u==v)
        return u;
    for(int i=M-1; i>=0; i--)
    {
        if(lca[u][i]!=lca[v][i])
        {
            u=lca[u][i];
            v=lca[v][i];
        }
    }
    return lca[u][0];
}

map<string,int>remap;
int data[N], ans[N], vis[N], freq[N], cnt;
inline void update(int u)
{
```

```
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 Articulations, Bridges & Cycles

2.1.2 Strongly Connected Components

Tarjan

```
class graph
{
    const static int MN=1e+5;
public:
    vector<int>data[MN], aux;
    bool vis[MN];
    int grp[MN];
    int dfs_num[MN], dfs_low[MN];
    int dfs_cnt, numSCC;

    graph(){};
    void clear()
    {
        for(int i=0; i<MN; i++)
        {
            data[i].clear();
            dfs_num[i]=-1;
            dfs_low[i]=0;
            vis[i]=false;
        }
        aux.clear();
        dfs_cnt=numSCC=0;
    }
    void add_edge(int u, int v)
    {
        data[u].push_back(v);
    }
    void tarjanSCC(int u)
    {
        dfs_num[u]=dfs_low[u]=dfs_cnt++;
        aux.push_back(u);
        vis[u]=true;

        for(int i=0; i<data[u].size(); i++)
        {
            int v=data[u][i];
            if(dfs_num[v]==-1)
                tarjanSCC(v);
            if(vis[v])
                dfs_low[v]=min(dfs_low[v], dfs_low[u]);
        }

        if(dfs_num[u]==dfs_low[u])
        {
            while(1)
            {
                int v=aux.back();
                aux.pop_back();
                vis[v]=false;
                grp[v]=numSCC;
                if(u==v)
                    break;
            }
            numSCC++;
        }
    }
};
```

2.1.3 Semi-Strongly Connected Components

2.2 Single Source Shortest Path

2.2.1 Dijkstra

2.2.2 Bellmanford

```
class node
{
public:
    int x, y, d;
    node(){};
    node(int _x, int _y, int _d)
    {
        x=_x, y=_y, d=_d;
    }
};

int n, v;
vector<node>graph;
int dist[1035];
bool bellmanford(int s)
{
    memset(dist, INF, sizeof(dist));
    dist[s]=0;
    for(int i=0; i<n-1; i++)
    {
        for(int j=0; j<graph.size(); j++)
        {
            int x=graph[j].x;
            int y=graph[j].y;
            int d=graph[j].d;
            if(dist[y]>dist[x]+d)
                dist[y]=dist[x]+d;
        }
    }

    for(int i=0; i<graph.size(); i++)
    {
        int x=graph[i].x;
        int y=graph[i].y;
        int d=graph[i].d;
        if(dist[x]<INF && dist[y]>dist[x]+d)
            return true;
    }
    return false;
}
```

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

```
const int MN=1e+3;
vector<int>g[MN];
int match[MN], rmatch[MN], vis[MN];
int findmatch(int u)
{
    if(vis[u])
        return 0;
    vis[u]=true;
    for(int v:g[u])
    {
        if(match[v]==-1 || findmatch(match[v]))
        {
            match[v]=u;
            rmatch[u]=v;
            return 1;
        }
    }
    return 0;
}

int maxMatch(int n)
{
    int ret=0;
    memset(match, -1, sizeof(match));
    for(int i=0; i<n; i++)
    {
        memset(vis, false, sizeof(vis));
        ret+=findmatch(i);
    }
    return ret;
}
```

2.5.2 Maximum Flow

Dinic

```
class graph
{
    const static int N=100000;
public:
    vector< pair<int,int> >edge;
    vector<int>adj[N];
    int ptr[N];
    int dist[N];

    graph(){};
    void clear()
    {
        for(int i=0; i<N; i++)
            adj[i].clear();
        edge.clear();
    }
    void add_edge(int u, int v, int c)
    {
        adj[u].push_back(edge.size());
        edge.push_back(mp(v, c));
        adj[v].push_back(edge.size());
        edge.push_back(mp(u, 0)); //(u, c) if is non-directed
    }
    bool dinic_bfs(int s, int t)
    {
        memset(dist, -1, sizeof(dist));
        dist[s]=0;

        queue<int>bfs;
        bfs.push(s);
        while(!bfs.empty() && dist[t]==-1)
        {
            int u=bfs.front();
            bfs.pop();
            for(int i=0; i<adj[u].size(); i++)
            {
                int idx=adj[u][i];
                int v=edge[idx].F;

                if(dist[v]==-1 && edge[idx].S>0)
                {
                    dist[v]=dist[u]+1;
                    bfs.push(v);
                }
            }
        }
        return dist[t]!=-1;
    }
    int dinic_dfs(int u, int t, int flow)
    {
        if(u==t)
            return flow;
        for(int &i=ptr[u]; i<adj[u].size(); i++)
        {
            int idx=adj[u][i];
            int v=edge[idx].F;
            if(dist[v]==dist[u]+1 && edge[idx].S>0)
            {
                int cf=dinic_dfs(v, t, min(flow, edge[idx].S));
                if(cf>0)
                {
                    edge[idx].S-=cf;
                    edge[idx^1].S+=cf;
                    return cf;
                }
            }
        }
        return 0;
    }
    int maxflow(int s, int t)
    {
        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:
$u \to uu(flow, 0)$
$uu \to vv(flow, cost)$
$vv \to v(flow, 0)$
$v \to uu(flow, 0)$
$vv \to u(flow, 0)$

Dijkstra

```
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();
    }
```

```
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

```
int stoer_wagner(int n)
{
    int ret=INF;
    for(int i=0; i<n; i++)
        v[i]=i;

    while(n>1)
    {
        a[ v[0] ]=true;
        for(int i=1; i<n; i++)
        {
            a[ v[i] ]=false;
            na[i-1]=i;
            w[i]=graph[ v[0] ][ v[i] ];
        }

        int prev=v[0];
        for(int i=1; i<n; i++)
        {
            int zj=-1;
            for(int j=1; j<n; j++)
            {
                if(!a[ v[j] ] && (zj<0 || w[j]>w[zj]))
                    zj=j;
            }

            a[ v[zj] ]=true;

            if(i==n-1)
            {
                ret=min(ret, w[zj]);

                for(int j=0; j<n; j++)
                    graph[ v[j] ][prev]=graph[prev][ v[j] ]+=graph[ v[zj] ][ v[j] ];
                v[zj]=v[--n];
                break;
            }
            prev=v[zj];

            for(int j=1; j<n; j++)
                if(!a[ v[j] ])
                    w[j]+=graph[ v[zj] ][ v[j] ];
        }
    }
    return ret;
}
```

2.6 Tree

2.6.1 Lowest Common Ancestor

```
const int MN=1e+5+35;
const int LMN=1+log2(MN);
vector<int>graph[MN];
int LVL[MN];
int T[MN];
int dp[MN][LMN];
bool vis[MN];

void dfs(int u, int f, int d)
{
    vis[u]=true;
    LVL[x]=d;
    dp[x][0]=f;
    for(int i=1; i<LMN; i++)
        dp[x][i]=dp[ dp[x][i-1] ][i-1];

    vis[x]=true;
    for(int i=0; i<graph[x].size(); i++)
    {
        int v=graph[x][u];
        if(!vis[v])
            dfs(v, x, d+1);
    }
}

inline int lca(int u, int v)
{
    if(LVL[u]>LVL[v])
        swap(u, v);
    for(int i=LMN-1; i>=0; i--)
        if(LVL[v]-(1<<i)>=LVL[u])
            v=dp[v][i];
    if(u==v)
        return u;
    for(int i=LMN-1; i>=0; i--)
    {
        if(dp[u][i]!=dp[v][i])
        {
            u=dp[u][i];
            v=dp[v][i];
        }
    }
    return dp[u][0];
}
```

2.6.2 Centroid Decomposition

```
class graph
{
    const static int N=1e+5;
    const static int LN=log2(N)+1;
public:
    vector<int>g[N];
    int h[N], lca[N][LN];

    int sz[N];
    int cg[N], gsz, dlt[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++)
            lca[u][i]=lca[ lca[u][i-1] ][i-1];
        for(int v:g[u])
        {
            if(v==f)
                continue;
            h[v]=h[u]+1;
            buildLca(v, u);
        }
    }
    inline int getLca(int u, int v)
    {
        if(h[u]>h[v])
            swap(u, v);
        for(int i=LN-1; i>=0; i--)
            if(h[v]-(1<<i)>=h[u])
                v=lca[v][i];
        if(u==v)
            return u;
        for(int i=LN-1; i>=0; i--)
        {
            if(lca[u][i]!=lca[v][i])
            {
                u=lca[u][i];
                v=lca[v][i];
            }
        }
        return lca[u][0];
    }
    inline int getDist(int u, int v)
    {
        return h[u]+h[v]-2*h[getLca(u, v)];
    }
    void buildSz(int u, int f)
    {
        gsz++;
        sz[u]=1;
        for(int v:g[u])
        {
            if(v==f || dlt[v])
                continue;
            buildSz(v, u);
            sz[u]+=sz[v];
        }
    }
    int findCentroid(int u, int f)
    {
        for(int v:g[u])
        {
            if(v==f || dlt[v])
                continue;
            if(sz[v]*2>=gsz)
                return findCentroid(v, u);
        }
        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

```
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 All-Pairs Distance & Centroid Decomposition

2.6.6 All-Pairs Distance & FFT

2.7 MISC

2.7.1 2-SAT

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]

```
typedef long long ll;
const int MN=1e+4+35;
const int MN2=535;
int p, a;
ll data[MN];
inline ll getValue(int l, int r)
{
    return (r-l+1)*(data[r]-data[l-1]);
}
ll dp[MN2][MN];
inline void solve(int k, int l, int r, int L, int R)
{
    if(l>r)
        return;
    int m=(l+r)/2;
    int s=L;
    dp[k][m]=LINF;
    for(int i=max(m, L); i<=R; i++)
    {
        if(dp[k][m]>dp[k-1][i+1]+getValue(m+1, i+1))
        {
            dp[k][m]=dp[k-1][i+1]+getValue(m+1, i+1);
            s=i;
        }
    }
    solve(k, l, m-1, L, s);
    solve(k, m+1, r, s, R);
}
int main()
{
    scanf("%d %d", &p, &a);
    for(int i=1; i<=p; i++)
    {
        ll x;
        scanf("%lld", &x);
        data[i]=data[i-1]+x;
    }
    for(int i=0; i<=p; i++)
        dp[0][i]=LINF;
    for(int i=0; i<=a; i++)
        dp[i][p]=0;
    for(int i=1; i<=a; i++)
        solve(i, 0, p-1, 0, p-1);
    printf("%lld\n", dp[a][0]);
}
```

3.1.2 Divide and Counquer - Example 2

```
typedef long long ll;
const int MN=6005;
ll v[MN];
ll dp[MN][MN];
ll c[MN][MN];

ll sum[MN];
ll multisum[MN];

void solve(int k, int l, int r, int L, int R)
{
    if(l>r)
        return;
    int m=(l+r)/2;
    int s=-1;
    dp[k][m]=LINF;
    for(int i=max(m, L); i<=R; i++)
    {
        if(dp[k][m]>dp[k-1][i+1]+c[m][i])
        {
            dp[k][m]=dp[k-1][i+1]+c[m][i];
            s=i;
        }
    }
    solve(k, l, m-1, L, s);
    solve(k, m+1, r, s, R);
}

ll dist(int l, int r, int mid)
{
    ll s1=sum[mid]-sum[l];
    ll p1=multisum[mid]-multisum[l];
    ll s2=sum[r+1]-sum[mid+1];
    ll p2=multisum[r+1]-multisum[mid+1];
    return (s1*mid-p1)+(p2-s2*mid);
}

int main()
{
    int n;
    ll b, k;
    scanf("%d %lld %lld", &n, &b, &k);
    for(int i=0; i<n; i++)
    {
        scanf("%lld", &v[i]);
        sum[i+1]=sum[i]+v[i];
        multisum[i+1]=multisum[i]+v[i]*1LL*i;
    }

    for(int i=0; i<n; i++)
    {
        int mid=i;
        ll tot=0;
        ll smid=v[i];
        for(int j=i; j<n; j++)
        {
            tot+=v[j];
            while(smid+smid<tot)
                smid+=v[++mid];
            c[i][j]=k*dist(i, j, mid);
        }
    }
    for(int i=0; i<=n; i++)
        dp[0][i]=LINF;
    for(int i=0; i<=k; i++)
        dp[i][n]=0;

    for(int i=1; i<=n; i++)
    {
        solve(i, 0, n-1, 0, n-1);
        i>1?printf(" "):NULL;
        printf("%lld", i*b+dp[i][0]);
    }
    printf("\n");
```

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

3.1.3 Convex Hull I

Original recurrence:

$dp[i] = min(dp[j] + b[j] * a[i])$ 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

```
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

```
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

3.2 Matrix Exponentiation

```
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)
        return mp(val[k-1], 0);
    matrix base=build();
    matrix fib=mat_pow(base, k-n);

    ll ret=0;
    reverse(val, val+n);
    for(int i=0; i<n; i++)
        ret=modsum(ret, modmul(fib[0][i], val[i]));

    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

```
char str[100];
int dp[100][300][2];
bool memo[100][300][2];
int n, k;

//numeros de 0 a x, tal que a soma dos digitos eh igual a k
int solve(int i, int s, int t)
{
    if(i==n)
    {
        if(!t && s==k)
            return 1;
        return 0;
    }
    if(s>k)
        return 0;
    if(memo[i][s][t])
        return dp[i][s][t];
    int &ret=dp[i][s][t]=0;
    if(t)
    {
        for(int j=0; j<=str[i]-'0'; j++)
        {
            if(j==str[i]-'0')
                ret+=solve(i+1, s+j, 1);
            else
                ret+=solve(i+1, s+j, 0);
        }
    }
    else
    {
        for(int j=0; j<10; j++)
        {
            ret+=solve(i+1, s+j, 0);
        }
    }
    memo[i][s][t]=true;
    return ret;
}

//quantos bits ativos existem entre 0 e x
string str2;
int n2;
int dp2[100][300][2];
bool memo2[100][300][2];
int solve2(int i, int s, int t)
{
    if(i==n2)
        return s;
    if(memo2[i][s][t])
        return dp2[i][s][t];
    int &ret=dp2[i][s][t]=0;
    if(t)
    {
        for(int j=0; j<=str2[i]-'0'; j++)
        {
            if(j==str2[i]-'0')
                ret+=solve2(i+1, s+(j==1), 1);
            else
                ret+=solve2(i+1, s+(j==1), 0);
        }
    }
    else
    {
        for(int j=0; j<2; j++)
        {
            ret+=solve2(i+1, s+(j==1), 0);
        }
    }
    memo2[i][s][t]=true;
    return ret;
}

//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.

```
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;
}
```

Chapter 4

String

4.1 Hash

```
typedef unsigned long long ull;
class hashc
{
public:
    vector<ull>prefix;
    vector<ull>power;
    int k=37;
    int t;
    hashc(){};
    hashc(vector<int>&data)
    {
        t=data.size();
        prefix.resize(t+1, 0);
        power.resize(t+1, 0);
        prefix[0]=0;
        power[0]=1;
        for(int i=0; i<t; i++)
        {
            prefix[i+1]=prefix[i]*k+data[i];
            power[i+1]=power[i]*k;
        }
    }

    hashc build(string &str)
    {
        vector<int>data(str.size());
        for(int i=0; i<str.size(); i++)
            data[i]=(str[i]-'a'+1);
        return hashc(data);
    }

    ull get()
    {
        return prefix[t];
    }
    ull calc(int l, int r)
    {
        return prefix[r]-(prefix[l-1]*power[r-l+1]);
    }
    bool same(int xl, int xr, int yl, int yr)
    {
        return this->calc(xl, xr)==this->calc(yl, yr);
    }
    int find(hashc &pattern)
    {
        int pt=pattern.t;
        ull val=pattern.calc(1, pt);
        for(int i=1; i<=t-pt+1; i++)
        {
            if(this->calc(i, i+pt-1)==val)
                return i-1;
        }
        return -1;
```

```
54          }
55      };
```

4.2 KMP

```
int lps[1000000];
void lps_calc(string &str)
{
    lps[0]=0;
    for(int i=1, j=0, f=0; i<str.size(); i+=f, f=0)
    {
        if(str[i]==str[j])
        {
            lps[i]=j;
            j++;
            f=1;
        }
        else
        {
            if(j>0)
            {
                j=lps[j-1];
            }
            else
            {
                lps[i]=0;
                f=1;
            }
        }
    }
}

//finding str in pat
void kmp(string &str, string &pat)
{
    lps_calc(pat);
    int i=0, j=0;
    while(i<str.size())
    {
        if(str[i]==pat[j])
        {
            i++;
            j++;
        }
        if(j==pat.size())
        {
            printf("Padrao encontrado em: [%d, %d]", i-j, (i-j)+pat.size()-1);
            j=lps[j-1];
        }
        else if(i<str.size() && str[i]!=pat[j])
        {
            if(j!=0)
                j=lps[j-1];
            else
                i++;
        }
    }
}
```

4.3 Aho Corasick

```
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

4.5 Z-Algorithm

4.6 Suffix Array & LCP

```
const int MN=1e+6+35;
int data[MN], sa[MN], lcp[MN], lcp_rank[MN];

// lexicographic order for pairs
inline bool leq(int a1, int a2, int b1, int b2)
{
    return(a1 < b1 || a1 == b1 && a2 <= b2);
}

// and triples
inline bool leq(int a1, int a2, int a3, int b1, int b2, int b3)
{
    return(a1 < b1 || a1 == b1 && leq(a2,a3, b2,b3));
} // and triples

// stably sort a[0..n-1] to b[0..n-1] with keys in 0..K from r
static void radixPass(int* a, int* b, int* r, int n, int K)
{// count occurrences
    int* c = new int[K + 1]; // counter array
    for (int i = 0; i <= K; i++)
        c[i] = 0; // reset counters
    for (int i = 0; i < n; i++)
        c[r[a[i]]]++; // count occurrences
    for (int i = 0, sum = 0; i <= K; i++) // exclusive prefix sums
    {
        int t = c[i];
        c[i] = sum;
        sum += t;
    }
    for (int i = 0;  i < n; i++)
        b[c[r[a[i]]]++] = a[i]; // sort
}

// find the suffix array SA of s[0..n-1] in {1..K}ËĘn
// require s[n]=s[n+1]=s[n+2]=0, n>=2
void suffixArray(int* s, int* SA, int n, int K)
{
    int n0 = (n+2)/3, n1 = (n+1)/3, n2 = n/3, n02 = n0+n2;
    int* s12 = new int[n02+3]; s12[n02] = s12[n02+1] = s12[n02+2] = 0;
    int* SA12 = new int[n02+3]; SA12[n02] = SA12[n02+1] = SA12[n02+2] = 0;
    int* s0 = new int[n0];
    int* SA0 = new int[n0];
    // generate positions of mod 1 and mod 2 suffixes
    // the "+(n0-n1)" adds a dummy mod 1 suffix if n%3 == 1
    for (int i=0, j=0; i < n + (n0-n1); i++)
        if (i%3 != 0) s12[j++] = i;
    // lsb radix sort the mod 1 and mod 2 triples
    radixPass(s12 , SA12, s+2, n02, K);
    radixPass(SA12, s12 , s+1, n02, K);
    radixPass(s12 , SA12, s  , n02, K);
    // find lexicographic names of triples
    int name = 0, c0 = -1, c1 = -1, c2 = -1;
    for (int i = 0; i < n02; i++)
    {
        if (s[SA12[i]] != c0 || s[SA12[i]+1] != c1 || s[SA12[i]+2] != c2)
        {
            name++;
            c0 = s[SA12[i]];
            c1 = s[SA12[i]+1];
            c2 = s[SA12[i]+2];
        }
        if (SA12[i]%3 == 1) s12[SA12[i]/3] = name; // left half
        else s12[SA12[i]/3 + n0] = name; // right half
    }
    // recurse if names are not yet unique
    if (name < n02)
    {
        suffixArray(s12, SA12, n02, name);
        // store unique names in s12 using the suffix array
        for(int i=0; i<n02; i++)
            s12[SA12[i]] = i + 1;
    }
    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 }
```

4.7 Suffix Tree

Chapter 5

Mathematic

5.1 Prime Numbers

5.1.1 Erastotenes Sieve

5.1.2 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 }
```

5.1.3 Miller Rabin

```
//millerRabin(n) returns if n is prime
//not accurate for all n
#define gcd(x, y) __gcd(x, y)
ll powmod(ll a, ll b, ll m)
{
    ll ret=1;
    while(b)
    {
        if(b&1)
            ret=(ret*a)%m, --b;
        else
            a=(a*a)%m, b>>=1;
    }
    return ret;
}

bool millerRabin(ll n)
{
    ll b=2;
    for(ll g; (g=gcd(n, b))!=1; b++)
        if(n>g)
            return false;
    ll p=0, q=n-1;
    while((q&1)==0)
        p++, q>>=1;
    ll rem=powmod(b, q, n);
    if(rem==1 || rem==n-1)
        return true;
    for(ll i=1; i<p; i++)
    {
        rem=(rem*rem)%n;
        if(rem==n-1)
            return true;
    }
    return false;
}
```

5.1.4 BPSW

```
//bpsw(n) returns if n is prime
#define gcd(x, y) __gcd(x, y)
ll jacobi(ll a, ll b)
{
    if(a==0 || a==1)
        return a;
    if(a<0)
    {
        if((b&2)==0)
            return jacobi(-a, b);
        return -jacobi(-a, b);
    }
    ll a1=a, e=0;
    while((a1&1)==0)
        a1>>=1, e++;
    ll s;
    if((e&1)==0 || (b&7)==1 || (b&7)==7)
        s=1;
    else
        s=-1;
    if((b&3)==3 && (a1&3)==3)
        s=-s;
    if(a1==1)
        return s;
    return s*jacobi(b%a1, a1);
}

bool bpsw(ll n)
{
    if((ll)sqrt(n+0.0)*(ll)sqrt(n+0.0)==n)
        return false;
    ll dd=5;
    while(1)
    {
        ll g=gcd(n, abs(dd));
        if(1<g && g<n)
            return false;
        if(jacobi(dd, n)==-1)
            break;
        dd=dd<0?-dd+2:-dd-2;
    }
    ll p=1, q=(p*p-dd)/4;
    ll d=n+1, s=0;
    while((d&1)==0)
        s++, d>>=1;
    ll u=1, v=p, u2m=1, v2m=p, qm=q, qm2=q*2, qkd=q;
    for(ll mask=2; mask<=d; mask<<=1)
    {
        u2m=(u2m*v2m)%n;
        v2m=(v2m*v2m)%n;
        while(v2m<qm2)
            v2m+=n;
        v2m-=qm2;
        qm=(qm*qm)%n;
        qm2=qm*2;
        if(d&mask)
        {
            ll t1=(u2m*v)%n, t2=(v2m*u)%n;
            ll t3=(v2m*v)%n, t4=(((u2m*u)%n)*dd)%n;
            u=t1+t2;
            if(u&1)
                u+=n;
            u=(u>>1)%n;
            v=(t3+t4);
            if(v&1)
                v+=n;
            v=(v>>1)%n;
            qkd=(qkd*qm)%n;
        }
    }
    if(u==0 || v==0)
        return true;
    ll qkd2=qkd*2;
    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  }
```

5.1.5 Primality Test

```
//call sieve() before isPrime(x)
//define k=50 as trivial limit
bool isPrime(ll x)
{
    if(x==1)
        return false;
    if(x==2)
        return true;
    if(x%2==0)
        return false;
    for(int i=0; i<k && x>pr[i]; i++)
        if(x%pr[i]==0)
            return false;
    if(pr[k-1]*pr[k-1]>=x)
        return true;
    //return only millerRabin(x) for fast process
    //not accurate for all x
    return millerRabin(x)?bpsw(x):false;
}
```

5.1.6 Java Pollard Rho Decomposition

```
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 Chinese Remainder Theorem

5.3 Fast Fourier Transformation

```
#define PI (double)acos(-1.0)
typedef complex<double> base;
void fft(vector<base>&data, bool invert)
{
    int n=data.size();
    for(int i=1, j=0; i<n; i++)
    {
        int bit=n>>1;
        for(; j>=bit; bit>>=1)
            j-=bit;
        j+=bit;
        if(i<j)
            swap(data[i], data[j]);
    }

    for(int len=2; len<=n; len<<=1)
    {
        double ang=2*PI/len*(invert?-1:1);
        base wlen(cos(ang), sin(ang));
        for(int i=0; i<n; i+=len)
        {
            base w(1);
            for(int j=0; j<len/2; j++)
            {
                base u=data[i+j], v=data[i+j+len/2]*w;
                data[i+j]=u+v;
                data[i+j+len/2]=u-v;
                w*=wlen;
            }
        }
    }
    if(invert)
        for(int i=0; i<n; i++)
            data[i]/=n;
}

vector<int>fft_multiply(vector<int>&a, vector<int>&b)
{
    vector<base>fa(a.begin(), a.end());
    vector<base>fb(b.begin(), b.end());
    int n=1;
    while(n<max(a.size(), b.size()))
        n<<=1;
    n<<=1;
    fa.resize(n);
    fb.resize(n);
    fft(fa, false);
    fft(fb, false);
    for(int i=0; i<n; i++)
        fa[i]*=fb[i];
    fft(fa, true);

    vector<int>ret(n);
    for(int i=0; i<n; i++)
        ret[i]=(int)(fa[i].real()+0.5);

    int carry=0;
    for(int i=0; i<n; i++)
    {
        ret[i]+=carry;
        carry=ret[i]/10;
        ret[i]%=10;
    }
    return ret;
}

int main()
{
    int n, m;
    scanf("%d %d", &n, &m);
    vector<int>a,b;

    for(int i=0; i<n; i++)
    {
```

```
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 }
```

5.4 Modular Math

5.4.1 Multiplicative Inverse

```
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;
}
```

5.4.2 Linear All Multiplicative Inverse

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

5.5 Gaussian Elimination

```
const int MAXN = 110;

typedef double Number;
const Number EPS = 1e-9;

Number mat[MAXN][MAXN];
int idx[MAXN];   // row index
int pivot[MAXN]; // pivot of row i

// Solves Ax = B, where A is a neq x nvar matrix and B is mat[*][nvar]
// Returns a vector of free variables (empty if system is defined,
// or {-1} if no solution exists)
// Reduces matrix to reduced echelon form
vector<int> solve(int nvar, int neq)
{
    for(int i = 0; i < neq; i++) idx[i] = i;
    int currow = 0;
    vector<int> freeVars;
    for(int col = 0; col < nvar; col++)
    {
        int pivotrow = -1;
        Number val = 0;
        for(int row = currow; row < neq; row++)
        {
            if(fabs(mat[idx[row]][col]) > val + EPS)
            {
                val = fabs(mat[idx[row]][col]);
                pivotrow = row;
            }
        }
        if(pivotrow == -1) { freeVars.push_back(col); continue; }
        swap(idx[currow], idx[pivotrow]);
        pivot[currow] = col;
        for(int c = 0; c <= nvar; c++)
        {
            if(c == col) continue;
            mat[idx[currow]][c] = mat[idx[currow]][c] / mat[idx[currow]][col];
        }
        mat[idx[currow]][col] = 1;
        for(int row = 0; row < neq; row++)
        {
            if(row == currow) continue;
            Number k = mat[idx[row]][col] / mat[idx[currow]][col];
            for(int c = 0; c <= nvar; c++)
                mat[idx[row]][c] -= k * mat[idx[currow]][c];
        }
        currow++;
    }
    for(int row = currow; row < neq; row++)
        if(mat[idx[row]][nvar] != 0) return vector<int>(1, -1);
    return freeVars;
}
```

5.6 Combinatorics

Chapter 6

Geometry

6.1 2d

6.1.1 Point Template

```
inline int cmp(double x, double y = 0, double tol = eps)
{
    return (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
}

struct point
{
    double x, y;
    point(double x = 0, double y = 0): x(x), y(y) {}
    point operator +(point q) { return point(x + q.x, y + q.y); }
    point operator -(point q) { return point(x - q.x, y - q.y); }
    point operator *(double t) { return point(x * t, y * t); }
    point operator /(double t) { return point(x / t, y / t); }
    double operator *(point q) {return x * q.x + y * q.y;}//a*b = |a||b|cos(ang)
    double operator %(point q) {return x * q.y - y * q.x;}//a%b = |a||b|sin(ang)
    double polar() { return ((y > -eps) ? atan2(y,x) : 2*Pi + atan2(y,x)); }
    double mod() { return sqrt(x * x + y * y); }
    double mod2() { return (x * x + y * y); }
    point rotate(double t) {return point(x*cos(t)-y*sin(t), x*sin(t)+y*cos(t));}
    int cmp(point q) const
    {
        if (int t = ::cmp(x, q.x)) return t;
        return ::cmp(y, q.y);
    }
    bool operator ==(point q) const { return cmp(q) == 0; }
    bool operator !=(point q) const { return cmp(q) != 0; }
    bool operator < (point q) const { return cmp(q) < 0; }
    static point pivot;
};
point point::pivot;
typedef vector<point> polygon;
```

6.1.2 Functions

```
double abs(point p) { return hypot(p.x, p.y); }
double arg(point p) { return atan2(p.y, p.x); }

inline int ccw(point p, point q, point r)
{
    return cmp((p - r) % (q - r));
}

//Projeta o vetor v sobre o vetor u (cuidado precisao)
point proj(point v, point u)
{
    return u*((u*v) / (u*u));
}

//\angle(p,q,r)| e o menor angulo entre os vetores u(p-q) e v(r-q)
// p->q->r virar pra esquerda => angle(p,q,r) < 0
inline double angle(point p, point q, point r)
{
    point u = p - q, v = r - q;
    return atan2(u % v, u * v);
}

//Decide se q esta sobre o segmento fechado pr.
bool between(point p, point q, point r)
{
    return ccw(p, q, r) == 0 && cmp((p - q) * (r - q)) <= 0;
}

//Decide se os segmentos fechados pq e rs tem pontos em comum
bool seg_intersect(point p, point q, point r, point s)
{
    point A = q - p, B = s - r, C = r - p, D = s - q;
    int a = cmp(A % C) + 2 * cmp(A % D);
    int b = cmp(B % C) + 2 * cmp(B % D);
    if (a == 3 || a == -3 || b == 3 || b == -3) return false;
    if (a || b || p == r || p == s || q == r || q == s) return true;
    int t = (p < r) + (p < s) + (q < r) + (q < s);
    return t != 0 && t != 4;
}

// Calcula a distancia do ponto r ao segmento pq.
double seg_distance(point p, point q, point r)
{
    point A = r - q, B = r - p, C = q - p;
    double a = A * A, b = B * B, c = C * C;
    if (cmp(b, a + c) >= 0) return sqrt(a);
    else if (cmp(a, b + c) >= 0) return sqrt(b);
    else return fabs(A % B) / sqrt(c);
}

// Classifica o ponto p em relacao ao poligono T.
// Retorna 0, -1 ou 1 dependendo se p esta no exterior, na fronteira
// ou no interior de T, respectivamente.
int in_poly(point p, polygon& T) {
    double a = 0; int N = T.size();
    for (int i = 0; i < N; i++) {
        if (between(T[i], p, T[(i+1) % N])) return -1;
        a += angle(T[i], p, T[(i+1) % N]);
    }
    return cmp(a) != 0;
}

//Encontra o ponto de intersecao das retas pq e rs.
point line_intersect(point p, point q, point r, point s)
{
    point a = q - p, b = s - r, c = point(p % q, r % s);
    return point(point(a.x, b.x) % c, point(a.y, b.y) % c) / (a % b);
}

// Calcula a area orientada do poligono T.
// Se o poligono P estiver em setido anti-horario, poly_area(P) > 0,
// e <0 caso contrario
double poly_area(polygon& T)
{
```

```
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

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

using namespace std;

#define EPS 1e-9
#define PI acos(-1.0)

double DEG_to_RAD(double d) { return d * PI / 180.0; }

double RAD_to_DEG(double r) { return r * 180.0 / PI; }

struct point { double x, y;   // only used if more precision is needed
  point() { x = y = 0.0; }                      // default constructor
  point(double _x, double _y) : x(_x), y(_y) {}        // user-defined
  bool operator == (point other) const {
   return (fabs(x - other.x) < EPS && (fabs(y - other.y) < EPS)); } };

struct vec { double x, y;  // name: 'vec' is different from STL vector
  vec(double _x, double _y) : x(_x), y(_y) {} };

vec toVec(point a, point b) {       // convert 2 points to vector a->b
  return vec(b.x - a.x, b.y - a.y); }

double dist(point p1, point p2) {                // Euclidean distance
  return hypot(p1.x - p2.x, p1.y - p2.y); }           // return double

// returns the perimeter, which is the sum of Euclidian distances
// of consecutive line segments (polygon edges)
double perimeter(const vector<point> &P) {
  double result = 0.0;
  for (int i = 0; i < (int)P.size()-1; i++)  // remember that P[0] = P[n-1]
    result += dist(P[i], P[i+1]);
  return result; }

// returns the area, which is half the determinant
double area(const vector<point> &P) {
  double result = 0.0, x1, y1, x2, y2;
  for (int i = 0; i < (int)P.size()-1; i++) {
    x1 = P[i].x; x2 = P[i+1].x;
    y1 = P[i].y; y2 = P[i+1].y;
    result += (x1 * y2 - x2 * y1);
  }
  return fabs(result) / 2.0; }

double dot(vec a, vec b) { return (a.x * b.x + a.y * b.y); }

double norm_sq(vec v) { return v.x * v.x + v.y * v.y; }

double angle(point a, point o, point b) {  // returns angle aob in rad
  vec oa = toVec(o, a), ob = toVec(o, b);
  return acos(dot(oa, ob) / sqrt(norm_sq(oa) * norm_sq(ob))); }

double cross(vec a, vec b) { return a.x * b.y - a.y * b.x; }

// note: to accept collinear points, we have to change the '> 0'
// returns true if point r is on the left side of line pq
bool ccw(point p, point q, point r) {
  return cross(toVec(p, q), toVec(p, r)) > 0; }

// returns true if point r is on the same line as the line pq
bool collinear(point p, point q, point r) {
  return fabs(cross(toVec(p, q), toVec(p, r))) < EPS; }

// returns true if we always make the same turn while examining
// all the edges of the polygon one by one
bool isConvex(const vector<point> &P) {
  int sz = (int)P.size();
  if (sz <= 3) return false;   // a point/sz=2 or a line/sz=3 is not convex
  bool isLeft = ccw(P[0], P[1], P[2]);              // remember one result
  for (int i = 1; i < sz-1; i++)            // then compare with the others
    if (ccw(P[i], P[i+1], P[(i+2) == sz ? 1 : i+2]) != isLeft)
      return false;           // different sign -> this polygon is concave
  return true; }                                  // this polygon is convex

```

```
// returns true if point p is in either convex/concave polygon P
bool inPolygon(point pt, const vector<point> &P) {
  if ((int)P.size() == 0) return false;
  double sum = 0;    // assume the first vertex is equal to the last vertex
  for (int i = 0; i < (int)P.size()-1; i++) {
    if (ccw(pt, P[i], P[i+1]))
      sum += angle(P[i], pt, P[i+1]);                    // left turn/ccw
    else sum -= angle(P[i], pt, P[i+1]); }               // right turn/cw
  return fabs(fabs(sum) - 2*PI) < EPS; }

// line segment p-q intersect with line A-B.
point lineIntersectSeg(point p, point q, point A, point B) {
  double a = B.y - A.y;
  double b = A.x - B.x;
  double c = B.x * A.y - A.x * B.y;
  double u = fabs(a * p.x + b * p.y + c);
  double v = fabs(a * q.x + b * q.y + c);
  return point((p.x * v + q.x * u) / (u+v), (p.y * v + q.y * u) / (u+v)); }

// cuts polygon Q along the line formed by point a -> point b
// (note: the last point must be the same as the first point)
vector<point> cutPolygon(point a, point b, const vector<point> &Q) {
  vector<point> P;
  for (int i = 0; i < (int)Q.size(); i++) {
    double left1 = cross(toVec(a, b), toVec(a, Q[i])), left2 = 0;
    if (i != (int)Q.size()-1) left2 = cross(toVec(a, b), toVec(a, Q[i+1]));
    if (left1 > -EPS) P.push_back(Q[i]);       // Q[i] is on the left of ab
    if (left1 * left2 < -EPS)        // edge (Q[i], Q[i+1]) crosses line ab
      P.push_back(lineIntersectSeg(Q[i], Q[i+1], a, b));
  }
  if (!P.empty() && !(P.back() == P.front()))
    P.push_back(P.front());        // make P's first point = P's last point
  return P; }

point pivot;
bool angleCmp(point a, point b) {                  // angle-sorting function
  if (collinear(pivot, a, b))                                // special case
    return dist(pivot, a) < dist(pivot, b);    // check which one is closer
  double d1x = a.x - pivot.x, d1y = a.y - pivot.y;
  double d2x = b.x - pivot.x, d2y = b.y - pivot.y;
  return (atan2(d1y, d1x) - atan2(d2y, d2x)) < 0; }   // compare two angles

vector<point> CH(vector<point> P) {   // the content of P may be reshuffled
  int i, j, n = (int)P.size();
  if (n <= 3) {
    if (!(P[0] == P[n-1])) P.push_back(P[0]); // safeguard from corner case
    return P;                             // special case, the CH is P itself
  }

  // first, find P0 = point with lowest Y and if tie: rightmost X
  int P0 = 0;
  for (i = 1; i < n; i++)
    if (P[i].y < P[P0].y || (P[i].y == P[P0].y && P[i].x > P[P0].x))
      P0 = i;

  point temp = P[0]; P[0] = P[P0]; P[P0] = temp;    // swap P[P0] with P[0]

  // second, sort points by angle w.r.t. pivot P0
  pivot = P[0];                    // use this global variable as reference
  sort(++P.begin(), P.end(), angleCmp);             // we do not sort P[0]

  // third, the ccw tests
  vector<point> S;
  S.push_back(P[n-1]); S.push_back(P[0]); S.push_back(P[1]);   // initial S
  i = 2;                                          // then, we check the rest
  while (i < n) {          // note: N must be >= 3 for this method to work
    j = (int)S.size()-1;
    if (ccw(S[j-1], S[j], P[i])) S.push_back(P[i++]);  // left turn, accept
    else S.pop_back(); }   // or pop the top of S until we have a left turn
  return S; }
                                  // return the result
void init()
{
  freopen("in.txt","r",stdin);
  freopen("out.txt","w",stdout);
  cout << "[FREOPEN]" << endl;
  return;
```

```
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   }
```

6.2 3d

6.2.1 Point Template

```
#define vetor point

// FORMULAS.
//  vetores a,b; a*b = a.mod()*b.mod()*cos( angulo entre a e b) =>
// a*b = |a|*|b|*cos(t)
//  vetores a,b; (a^b).mod() = a.mod()*b.mod()*sin( angulo entre a e b)

inline int cmp(ld x, ld y = 0, ld tol = eps)
{
    return (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
}
struct point
{
    ld x, y, z;
    point(ld x = 0, ld y = 0, ld z = 0): x(x), y(y), z(z) {}
    point operator +(point q) { return point(x + q.x, y + q.y, z + q.z); }
    point operator -(point q) { return point(x - q.x, y - q.y, z - q.z); }
    point operator *(ld t) { return point(x * t, y * t, z * t); }
    point operator /(ld t) { return point(x / t, y / t, z / t); }
    point operator ^(point q) {
        return point(y*q.z - z*q.y, z*q.x - x*q.z, x*q.y - y*q.x); }
    ld operator *(point q) { return x * q.x + y * q.y + z * q.z; }
    ld mod() { return sqrt(x * x + y * y + z * z); }
    ld mod2() { return x * x + y * y + z * z; }
    point projecao(vetor u) { return (*this) * ((*this)*u) / ((*this)*(*this)); }

    int cmp(point q) const
    {
        if (int t = ::cmp(x, q.x)) return t;
        if (int t = ::cmp(y, q.y)) return t;
        return ::cmp(z, q.z);
    }
    bool operator ==(point q) const { return cmp(q) == 0; }
    bool operator !=(point q) const { return cmp(q) != 0; }
    bool operator < (point q) const { return cmp(q) < 0; }
};

// RETAS, SEMIRETAS, SEGMENTOS E TRIANGULOS
struct reta
{
    point a, b;//  <--a----b---->
    reta(point A=point(0,0,0), point B=point(0,0,0)): a(A), b(B) { }

    //verifica se o ponto p esta na reta ab
    bool belongs(point p)
    {
        return cmp(((a-p)^(b-p)).mod()) == 0;
    }
};
struct semireta
{
    point a, b; //  |a----b---->
    semireta(point A=point(0,0,0), point B=point(0,0,0)): a(A), b(B) { }
};
struct segmento
{
    point a, b; //  |a----b|
    segmento(point A=point(0,0,0), point B=point(0,0,0)): a(A), b(B) { }
    bool between(point p) {
        return cmp(((a-p)^(b-p)).mod()) == 0 && cmp((a-p) * (b-p)) <= 0;
    }
};
struct triangulo
{
    point a, b, c;
    triangulo(point A, point B, point C): a(A), b(B), c(C) { }
    ld area() { return 0.5*((b-a)^(c-a)).mod(); }

    //retorna o ponto que eh a projecao de p no plano abc
    point projecao(point p)
    {
```

```
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)
```

```
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

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.

```
double minimumWidth(polygon &ch)
{
    double ret=DINF;
    int sz=ch.size();
    int j=1;
    for(int i=0; i<sz; i++)
    {
        int nxt=(j+1)%sz;
        while(distPointLine(ch[i], ch[(i+1)%sz], ch[j])<distPointLine(ch[i], ch[(i+1)%sz], ch[nxt]))
        {
            j=(j+1)%sz;
            nxt=(j+1)%sz;
        }
        ret=min(ret, distPointLine(ch[i], ch[(i+1)%sz], ch[j]));
    }
    return ret;
}
```

6.5 KD Tree

```
struct point
{
    ll x, y, z;
    point(ll x=0, ll y=0, ll z=0): x(x), y(y), z(z) {}
    point operator-(point q) { return point(x-q.x, y-q.y, z-q.z); }
    ll operator*(point q) { return x*q.x + y*q.y + z*q.z; }
};
typedef vector<point> polygon;

struct KDTreeNode
{
    point p;
    int level;
    KDTreeNode *below, *above;

    KDTreeNode (const point& q, int levl)
    {
        p = q;
        level = levl;
        below = above = 0;
    }
    ~KDTreeNode() { delete below, above; }

    int diff (const point& pt)
    {
        switch (level)
        {
        case 0: return pt.x - p.x;
        case 1: return pt.y - p.y;
        case 2: return pt.z - p.z;
        }
        return 0;
    }
    ll distSq (point& q) { return (p-q)*(p-q); }

    int rangeCount (point& pt, ll K)
    {
        int count = (distSq(pt) < K*K) ? 1 : 0;
        int d = diff(pt);
        if (-d <= K && above != 0)
            count += above->rangeCount(pt, K);
        if (d <= K && below != 0)
            count += below->rangeCount(pt, K);
        return count;
    }
};

class KDTree
{
public:
    polygon P;
    KDTreeNode *root;
    int dimention;
    KDTree() {}
    KDTree(polygon &poly, int D)
    {
        P = poly;
        dimention = D;
        root = 0;
        build();
    }
    ~KDTree() { delete root; }

    //count the number of pairs that has a distance less than K
    ll countPairs(ll K)
    {
        ll count = 0;
        f(i, 0, P.size())
            count += root->rangeCount(P[i], K) - 1;
        return count;
    }

protected:
    void build()
```

```
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 }
```

6.6 Range Tree

6.7 Circle Sweep

Chapter 7

Templates

7.1 C++

```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};

inline void solve()
{

}

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  }
```

7.2 Java

7.3 Time Check

```
using namespace std::chrono;
class timecheck
{
public:
    high_resolution_clock::time_point t1, t2;
    void start()
    {
        t1 = high_resolution_clock::now();
    }
    void end()
    {
        t2= high_resolution_clock::now();
        duration<double> time_span = duration_cast<duration<double>>(t2 - t1);
        cout << "Time: " << time_span.count() << "s" << endl;
    }
};
```

Chapter 8

Formulas

8.1 Areas

8.2 Volumes

8.3 Series

8.4 Combinatorics

8.5 Integral