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 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);
        }
};
```

1.1.3 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;
        }
};
```

1.1.4 Persistent Segtree

```
class persistent_segtree
{
        const static int N=100000;
        int n;
        int tr[N];
        int root[N], L[N], R[N];
        int cnt, id;
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);
        }
        //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)
{
```

```
        if(root)
                root->v=max(root->x, max(get_max(root->L), get_max(root->R)));
}

inline void update(node *root)
{
        propagate(root);
        upd_cnt(root);
        upd_max(root);
}

void merge(node *&root, node *L, node *R)
{
        push(L);
        push(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;
        update(root);
}

void split(node *root, node *&L, node *&R, int x, int add=0)
{
        if(!root)
                return void(L=R=NULL);
        push(root);
        int ix=add+get_cnt(root->L); //implicit key
        if(x<=ix)
                split(root->L, L, root->L, x, add), R=root;
        else
                split(root->R, root->R, R, x, add+1+get_cnt(root->L)), L=root;
        update(root);
}

//insert function
void insert(node *&root, int pos, int x)//(insert x at position pos)
{
        node *R1, *R2;
        split(root, R1, R2, pos);
        merge(R1, R1, new node(x));
        merge(root, R1, R2);
}

//erase value x
void erase_x(node *&root, int x)
{
        if(!root)
                return;
        if(root->x==x)
                merge(root, root->L, root->R);
        else
                erase_x(x < root->x? root->L:root->R, x);
        update(root);
}

//erase kth value
void erase_kth(node *&root, int x)
{
        if(!root)
                return;
        int Lc=get_cnt(root->L);
        if(x-Lc-1==0)
                merge(root, root->L, root->R);
        else if(x>Lc)
                erase_kth(root->R, x-Lc-1);
        else
                erase_kth(root->L, x);
        update(root);
}

//add x to [l,r]
inline void paint(node *&root, int l, int r, int x)
{
        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     }
```

1.4 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

```
#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

2.1.3 Semi-Strongly Connected Components

2.2 Single Source Shortest Path

2.2.1 Dijkstra

2.2.2 Bellmanford

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

```
                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

Dijkstra

```
/*
undirected graph:
u->uu(flow, 0)
uu->vv(flow, cost)
vv->v(flow, 0)
v->xx(flow, 0)
vv->u(flow, 0)
*/
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();
        }
        inline void add_edge(int u, int v, FTYPE c, FTYPE cst)
        {
```

```
73                  adj[u].push_back(edge.size());
74                  edge.push_back(Edge(v, c, cst));
75                  adj[v].push_back(edge.size());
76                  edge.push_back(Edge(u, 0, -cst));
77          }
78
79          pfc dijkstra(int s, int t)
80          {
81                  for(register int i=0; i<n; i++)
82                          dist[i]=CINF;
83                  dist[s]=0;
84                  flow[s]=FINF;
85                  priority_queue<node>heap;
86                  heap.push(node(s, 0));
87                  while(!heap.empty())
88                  {
89                          int u=heap.top().u;
90                          CTYPE d=heap.top().d;
91                          heap.pop();
92                          if(d>dist[u])
93                                  continue;
94                          for(int i=0; i<adj[u].size(); i++)
95                          {
96                                  int idx=adj[u][i];
97                                  int v=edge[idx].to;
98                                  CTYPE w=edge[idx].cost;
99                                  if(!edge[idx].cap || dist[v]<=d+w+pot[u]-pot[v])
100                                         continue;
101                                 if(d+w<dist[v])
102                                 {
103                                         dist[v]=d+w;
104                                         prev[v]=u;
105                                         eidx[v]=idx;
106                                         flow[v]=min(flow[u], edge[idx].cap);
107                                         heap.push(node(v, d+w));
108                                 }
109                         }
110                 }
111                 if(dist[t]==CINF)
112                         return mp(FINF, CINF);
113                 pfc ret=mp(flow[t], 0);
114                 for(int u=t; u!=s; u=prev[u])
115                 {
116                         int idx=eidx[u];
117                         edge[idx].cap-=flow[t];
118                         edge[idx^1].cap+=flow[t];
119                         ret.second+=flow[t]*edge[idx].cost;
120                 }
121                 return ret;
122         }
123
124         inline pfc mfmc(int s, int t)
125         {
126                 pfc ret=mp(0, 0);
127                 pfc got;
128                 while((got=dijkstra(s, t)).first!=FINF)
129                         ret+=got;
130                 return ret;
131         }
132 };
```

Bellmanford

2.5.4 Minimum Cut

2.6 Tree

2.6.1 Lowest Common Ancestor

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

2.6.2 Centroid Decomposition

```
const int N=1e+5;
const int M=log2(N)+1;

set<int>g[N]; //graph
int h[N]; //heigh of nodes
int trSz[N], sz; //tree subsize, size of current tree
int lca[N][M]; //lca sparse table
int cg[N]; //centroid graph

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

inline int getDist(int u, int v)
{
        return h[u]+h[v]-2*h[getLca(u, v)];
}

void centDfs(int u, int l)
{
        trSz[u]=1;
        sz++;
        for(auto v:g[u])
        {
                if(v==l)
                        continue;
                centDfs(v, u);
                trSz[u]+=trSz[v];
        }
}

int findCentroid(int u, int l)
{
        for(auto v:g[u])
        {
                if(v==l)
                        continue;
                if(trSz[v]*2>=sz)
                        return findCentroid(v, u);
        }
        return u;
}

inline void buildCentroid(int u, int l)
```

```
75  {
76          sz=0;
77          centDfs(u, u);
78          int c=findCentroid(u, u); //actual centroid
79          cg[c]=(u==l?c:l);
80          for(auto v:g[c])
81          {
82                  g[v].erase(g[v].find(c));
83                  buildCentroid(v, c);
84          }
85          g[c].clear();
86  }
```

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 Sum

2.7 MISC

2.7.1 2-SAT

Chapter 3

Dynamic Programming

3.1 Optimizations

3.1.1 Divide and Counquer

```
1  /// David Mateus Batista <david.batista3010@gmail.com>
2  /// Computer Science - Federal University of Itajuba - Brazil
3  /// Uri Online Judge - 2475
4  #include <bits/stdc++.h>
5
6  using namespace std;
7
8  typedef long long ll;
9  typedef unsigned long long ull;
10 typedef long double ld;
11 typedef pair<int,int> pii;
12 typedef pair<ll,ll> pll;
13
14 #define INF 0x3F3F3F3F
15 #define LINF 0x3F3F3F3F3F3F3F3FLL
16 #define DINF (double)1e+30
17 #define EPS (double)1e-9
18 #define PI (double)acos(-1.0)
19 #define RAD(x) (double)(x*PI)/180.0
20 #define PCT(x,y) (double)x*100.0/y
21 #define pb push_back
22 #define mp make_pair
23 #define pq priority_queue
24 #define F first
25 #define S second
26 #define D(x) x&(-x)
27 #define ALL(x) x.begin(),x.end()
28 #define SET(a,b) memset(a, b, sizeof(a))
29 #define DEBUG(x,y) cout << x << y << endl
30 #define gcd(x,y) __gcd(x, y)
31 #define lcm(x,y) (x/gcd(x,y))*y
32 #define bitcnt(x) __builtin_popcountll(x)
33 #define lbit(x) 63-__builtin_clzll(x)
34 #define zerosbitll(x) __builtin_ctzll(x)
35 #define zerosbit(x) __builtin_ctz(x)
36
37 enum {North, East, South, West};
38 //{0, 1, 2, 3}
39 //{Up, Right, Down, Left}
40
41 int mi[] = {-1, 0, 1, 0, -1, 1, 1, -1};
42 int mj[] = {0, 1, 0, -1, 1, 1, -1, -1};
43
44 const int MN=1e+4+35;
45 const int MN2=535;
46 int p, a;
47 ll data[MN];
48
49 inline ll getValue(int l, int r)
50 {
```

```
51          return (r-l+1)*(data[r]-data[l-1]);
52  }
53
54  ll dp[MN2][MN];
55  inline void solve(int k, int l, int r, int L, int R)
56  {
57          if(l>r)
58                  return;
59          int m=(l+r)/2;
60          int s=L;
61          dp[k][m]=LINF;
62          for(int i=max(m, L); i<=R; i++)
63          {
64                  if(dp[k][m]>dp[k-1][i+1]+getValue(m+1, i+1))
65                  {
66                          dp[k][m]=dp[k-1][i+1]+getValue(m+1, i+1);
67                          s=i;
68                  }
69          }
70          solve(k, l, m-1, L, s);
71          solve(k, m+1, r, s, R);
72  }
73
74  int main()
75  {
76          scanf("%d %d", &p, &a);
77          for(int i=1; i<=p; i++)
78          {
79                  ll x;
80                  scanf("%lld", &x);
81                  data[i]=data[i-1]+x;
82          }
83
84          for(int i=0; i<=p; i++)
85                  dp[0][i]=LINF;
86          for(int i=0; i<=a; i++)
87                  dp[i][p]=0;
88          for(int i=1; i<=a; i++)
89                  solve(i, 0, p-1, 0, p-1);
90          printf("%lld\n", dp[a][0]);
91          return 0;
92  }
```

3.1.2 Convex Hull I

Linear

```
//Original recurrence:
//        dp[i]=min(dp[j]+b[j]*a[i]) for j<i
//Condition:
//        b[j]>=b[j+1]
//        a[i]<=a[i+1]
// Solution:
// Hull cht=Hull();
// 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]

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

```
//Original recurrence:
//        dp[i]=min(dp[j]+b[j]*a[i]) for j<i
//Condition:
//        b[j]>=b[j+1]
//        a[i]<=a[i+1]
// Solution:
// HullDynamic 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]

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});
```

```
75                  return ret.m*x+ret.b;
76          }
77  };
```

3.1.3 Convex Hull II

3.1.4 Knuth Optimization

3.2 Digits

```
1   /// David Mateus Batista <david.batista3010@gmail.com>
2   /// Computer Science - Federal University of Itajuba - Brazil
3
4   #include <bits/stdc++.h>
5
6   using namespace std;
7
8   typedef long long ll;
9   typedef unsigned long long ull;
10
11  #define INF 0x3F3F3F3F
12  #define LINF 0x3F3F3F3F3F3F3F3FLL
13  #define DINF (double)1e+30
14  #define EPS (double)1e-9
15  #define PI (double)acos(-1.0)
16  #define RAD(x) (double)(x*PI)/180.0
17  #define PCT(x,y) (double)x*100.0/y
18  #define pb push_back
19  #define mp make_pair
20  #define pq priority_queue
21  #define F first
22  #define S second
23  #define D(x) x&(-x)
24  #define RESET(a,b) memset(a, b, sizeof(a))
25  #define DEBUG(x,y) cout << x << y << endl
26  #define gcd(x,y) __gcd(x, y)
27  #define lcm(x,y) (x*y)/gcd(x,y)
28  #define bitcount(x) __builtin_popcount(x)
29  #define llbitcount(x) __builtin_popcountll(x)
30
31  enum {North, East, South, West};
32  //{0, 1, 2, 3}
33  //{Up, Right, Down, Left}
34
35  int mi[] = {-1, 0, 1, 0, -1, 1, 1, -1};
36  int mj[] = {0, 1, 0, -1, 1, 1, -1, -1};
37
38  char str[100];
39  int dp[100][300][2];
40  bool memo[100][300][2];
41  int n, k;
42
43  //numeros de 0 a x, tal que a soma dos digitos eh igual a k
44  int solve(int i, int s, int t)
45  {
46          if(i==n)
47          {
48                  if(!t && s==k)
49                          return 1;
50                  return 0;
51          }
52          if(s>k)
53                  return 0;
54          if(memo[i][s][t])
55                  return dp[i][s][t];
56          int &ret=dp[i][s][t]=0;
57          if(t)
58          {
59                  for(int j=0; j<=str[i]-'0'; j++)
60                  {
61                          if(j==str[i]-'0')
62                                  ret+=solve(i+1, s+j, 1);
63                          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
        char str3[100];
        int n3;
        int dp3[100][300][2];
        bool memo3[100][300][2];
        int solve3(int i, int s, int t)
        {
                if(i==n3)
                        return !s;
                if(memo3[i][s][t])
                        return dp3[i][s][t];
                int &ret=dp3[i][s][t]=0;
                if(t)
                {
                        for(int j=0; j<=str3[i]-'0'; j++)
                        {
                                if(j==str3[i]-'0')
                                        ret+=solve3(i+1, (s+j)%k, 1);
                                else
                                        ret+=solve3(i+1, (s+j)%k, 0);
                        }
                }
                else
                {
                        for(int j=0; j<10; j++)
                        {
                                ret+=solve3(i+1, (s+j)%k, 0);
                        }
                }
                memo3[i][s][t]=true;
```

```
141         return ret;
142 }
143
144 //numeros de 1 a x, tal que o xor dos digitos eh igual a k
145 char str4[100];
146 int n4;
147 int dp4[100][300][2];
148 bool memo4[100][300][2];
149 int solve4(int i, int s, int t)
150 {
151         if(i==n4)
152                 return s==k;
153         if(memo4[i][s][t])
154                 return dp4[i][s][t];
155         int &ret=dp4[i][s][t]=0;
156         if(t)
157         {
158                 for(int j=0; j<=str4[i]-'0'; j++)
159                 {
160                         if(j==str4[i]-'0')
161                                 ret+=solve4(i+1, (s^j), 1);
162                         else
163                                 ret+=solve4(i+1, (s^j), 0);
164                 }
165         }
166         else
167         {
168                 for(int j=0; j<10; j++)
169                 {
170                         ret+=solve4(i+1, (s^j), 0);
171                 }
172         }
173         memo4[i][s][t]=true;
174         return ret;
175 }
176
177 void init()
178 {
179         freopen("in.txt","r",stdin);
180         freopen("out.txt","w",stdout);
181         cout << "[FREOPEN]" << endl;
182         return;
183 }
184
185 int main()
186 {
187         init();
188         int x;
189         scanf("%d %d", &x, &k);
190         sprintf(str, "%d", x);
191
192         //bruteforce A
193         int cnt=0;
194         for(int i=0; i<=x; i++)
195         {
196                 int aux=0;
197                 int j=i;
198                 while(j)
199                 {
200                         aux+=j%10;
201                         j/=10;
202                 }
203                 cnt+=aux==k;
204         }
205         //pd x bruteforce
206         n=strlen(str);
207         printf("%d [%d]\n", solve(0, 0, 1), cnt);
208
209         //bruteforce B
210         int cnt2=0;
211         for(int i=0; i<=x; i++)
212         {
213                 int j=i;
214                 while(j)
215                 {
216                         cnt2+=j%2;
217                         j/=2;
```

```
218                     }
219             }
220             for(int i=0, y=x; y>0; y/=2)
221                     str2.pb((y%2)+'0');
222             reverse(str2.begin(), str2.end());
223             n2=str2.size();
224
225             printf("%d [%d]\n", solve2(0, 0, 1), cnt2);
226
227             //Bruteforce C
228             int cnt3=0;
229             for(int i=0; i<=x; i++)
230             {
231                     int y=i;
232                     int aux=0;
233                     while(y)
234                     {
235                             aux+=y%10;
236                             y/=10;
237                     }
238                     cnt3+=(aux%k==0);
239             }
240             sprintf(str3, "%d", x);
241             n3=strlen(str3);
242             printf("%d [%d]\n", solve3(0, 0, 1), cnt3);
243
244
245             //Bruteforce D
246             int cnt4=0;
247             for(int i=0; i<=x; i++)
248             {
249                     int y=i;
250                     int aux=0;
251                     while(y)
252                     {
253                             aux^=y%10;
254                             y/=10;
255                     }
256                     //printf("%d\n", aux);
257                     cnt4+=(aux==k);
258             }
259             sprintf(str4, "%d", x);
260             n4=strlen(str4);
261             printf("%d [%d]\n", solve4(0, 0, 1), cnt4);
262             return 0;
263     }
```

3.3 Grundy Numbers

Chapter 4

String

4.1 Hash

4.2 KMP

4.3 Aho Corasick

```
1  class aho_corasick
2  {
3  private:
4          static const int MNT=1e+6;
5          static const int MNC=26;
6  public:
7          int trie[MNT][MNC];
8          int term[MNT];
9          int link[MNT];
10         int sum[MNT];
11         int cnt=1;
12         aho_corasick(){};
13         void clear()
14         {
15                 RESET(trie, 0);
16                 RESET(term, 0);
17                 RESET(link, 0);
18                 RESET(sum, 0);
19                 cnt=1;
20         }
21         int node(int x, int j)
22         {
23                 return trie[x][j];
24         }
25         int end(int x, int j)
26         {
27                 return term[ node(x,j) ];
28         }
29         void insert(char *str)
30         {
31                 int sz=strlen(str);
32                 int no=0;
33                 for(int i=0; i<sz; i++)
34                 {
35                         int x=str[i]-'a';
36                         if(!trie[no][x])
37                                 trie[no][x]=cnt++;
38                         sum[ trie[no][x] ]++;
39                         no=trie[no][x];
40                 }
41                 term[no]++;
42         }
43         bool find(char *str)
44         {
45                 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();
                        aho.pop();
                        term[x]|=term[ link[x] ];
                        for(int i=0; i<MNC; i++)
                        {
                                if(trie[x][i])
                                {
                                        int y=trie[x][i];
                                        aho.push(y);
                                        link[y]=x?trie[ link[x] ][i]:0;
                                }
                                else
                                {
                                        trie[x][i]=trie[ link[x] ][i];
                                }
                        }
                }
        }
};
```

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)
```

```
18  {// count occurrences
19      int* c = new int[K + 1]; // counter array
20      for (int i = 0; i <= K; i++)
21          c[i] = 0; // reset counters
22      for (int i = 0; i < n; i++)
23          c[r[a[i]]]++; // count occurrences
24      for (int i = 0, sum = 0; i <= K; i++) // exclusive prefix sums
25      {
26          int t = c[i];
27          c[i] = sum;
28          sum += t;
29      }
30      for (int i = 0;  i < n; i++)
31          b[c[r[a[i]]]++] = a[i]; // sort
32  }
33
34  // find the suffix array SA of s[0..n-1] in {1..K}ËËn
35  // require s[n]=s[n+1]=s[n+2]=0, n>=2
36  void suffixArray(int* s, int* SA, int n, int K)
37  {
38      int n0 = (n+2)/3, n1 = (n+1)/3, n2 = n/3, n02 = n0+n2;
39      int* s12 = new int[n02+3]; s12[n02] = s12[n02+1] = s12[n02+2] = 0;
40      int* SA12 = new int[n02+3]; SA12[n02] = SA12[n02+1] = SA12[n02+2] = 0;
41      int* s0 = new int[n0];
42      int* SA0 = new int[n0];
43      // generate positions of mod 1 and mod 2 suffixes
44      // the "+(n0-n1)" adds a dummy mod 1 suffix if n%3 == 1
45      for (int i=0, j=0; i < n + (n0-n1); i++)
46          if (i%3 != 0) s12[j++] = i;
47      // lsb radix sort the mod 1 and mod 2 triples
48      radixPass(s12 , SA12, s+2, n02, K);
49      radixPass(SA12, s12 , s+1, n02, K);
50      radixPass(s12 , SA12, s  , n02, K);
51      // find lexicographic names of triples
52      int name = 0, c0 = -1, c1 = -1, c2 = -1;
53      for (int i = 0; i < n02; i++)
54      {
55          if (s[SA12[i]] != c0 || s[SA12[i]+1] != c1 || s[SA12[i]+2] != c2)
56          {
57              name++;
58              c0 = s[SA12[i]];
59              c1 = s[SA12[i]+1];
60              c2 = s[SA12[i]+2];
61          }
62          if (SA12[i]%3 == 1) s12[SA12[i]/3] = name; // left half
63          else s12[SA12[i]/3 + n0] = name; // right half
64      }
65      // recurse if names are not yet unique
66      if (name < n02)
67      {
68          suffixArray(s12, SA12, n02, name);
69          // store unique names in s12 using the suffix array
70          for(int i=0; i<n02; i++)
71                  s12[SA12[i]] = i + 1;
72      }
73      else // generate the suffix array of s12 directly
74      {
75          for(int i = 0;  i < n02; i++)
76                  SA12[s12[i] - 1] = i;
77      }
78      // stably sort the mod 0 suffixes from SA12 by their first character
79      for (int i=0, j=0; i<n02; i++)
80          if (SA12[i] < n0) s0[j++] = 3*SA12[i];
81      radixPass(s0, SA0, s, n0, K);
82      // merge sorted SA0 suffixes and sorted SA12 suffixes
83      for (int p = 0, t = n0-n1, k = 0; k < n; k++)
84      {
85          #define GetI() (SA12[t] < n0 ? SA12[t] * 3 + 1 : (SA12[t] - n0) * 3 + 2)
86          int i = GetI(); // pos of current offset 12 suffix
87          int j = SA0[p]; // pos of current offset 0 suffix
88          if (SA12[t] < n0 ? // different compares for mod 1 and mod 2 suffixes
89              leq(s[i], s12[SA12[t] + n0], s[j], s12[j/3]) :
90              leq(s[i],s[i+1],s12[SA12[t]-n0+1], s[j],s[j+1],s12[j/3+n0]))
91          {// suffix from SA12 is smaller
92              SA[k] = i; t++;
93              if (t == n02) // done --- only SA0 suffixes left
94              for (k++; p < n0; p++, k++) SA[k] = SA0[p];
```

```
95          }
96          else
97          {// suffix from SA0 is smaller
98              SA[k] = j; p++;
99              if (p == n0) // done --- only SA12 suffixes left
100             for (k++; t < n02; t++, k++) SA[k] = GetI();
101         }
102     }
103 }
104
105 void buildlcp(int n)
106 {
107         int k=0;
108         for(int i=0; i<n; i++)
109                 lcp_rank[ sa[i] ]=i;
110         for(int i=0; i<n; i++, k?k--:0)
111         {
112                 if(lcp_rank[i]==n-1)
113                 {
114                         k=0;
115                         continue;
116                 }
117                 int j=sa[ lcp_rank[i]+1 ];
118                 while(i+k<n && j+k<n && data[i+k]==data[k+j])
119                         k++;
120                 lcp[ lcp_rank[i] ]=k;
121         }
122 }
123
124 int main()
125 {
126         int n;
127         scanf("%d", &n);
128         for(int i=0; i<n; i++)
129         {
130                 char x;
131                 scanf("␣%c", &x);
132                 data[i]=(int)x;
133         }
134     //data[i]>=1
135         data[n]=data[n+1]=data[n+2]=data[n+3]=0;
136         n++;
137     //suffixArray(string, ponteiro para suffix array, numero de elementos da string, number of
        elementos do alfabeto);
138         suffixArray(data, sa, n, 256);
139         for(int i=0; i<n; i++)
140                 printf("%d␣", sa[i]);
141         printf("\n\n");
142
143     //buildlcp(numero de elementos da string)
144         buildlcp(n);
145         for(int i=0; i<n; i++)
146                 printf("%d\n", lcp[i]);
147         return 0;
148 }
```

4.7 Suffix Tree

Chapter 5

Mathematic

5.1 Prime Numbers

5.1.1 Erastotenes Sieve

5.1.2 Linear Sieve

```
//prime(x):(lp[x]==x)
const int MN=1e+6;
long long lp[MN+1];
vector<long long>pr;

void sieve()
{
        for(long long i=2; i<=MN; i++)
        {
                if(lp[i]==0)
                {
                        lp[i]=i;
                        pr.push_back(i);
                }
                for(long long j=0; j<pr.size() && pr[j]<=lp[i] && i*pr[j]<=MN; j++)
                        lp[i*pr[j]]=pr[j];
        }
}
```

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++)
        {
                v=(v*v)%n-qkd2;
                v+=v<0?n:0;
                v+=v<0?n:0;
                v-=v>=n?n:0;
                v-=v>=n?n:0;
                if(v==0)
                        return true;
                if(r<s-1)
                {
                        qkd=(qkd*1LL*qkd)%n;
                        qkd2=qkd*2;
                }
        }
        return false;
}
```

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;
}
```

5.2 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

5.4.2 Linear All Multiplicative Inverse

5.5 Gaussian Elimination

5.6 Combinatorics

Chapter 6

Geometry

6.1 2d Template

6.2 3d Template

6.3 Polygon Template

6.4 Convex Hull

6.4.1 Graham Scan

6.4.2 Monotone Chain

6.5 Rotating Calipers

6.6 KD Tree

6.7 Range Tree

6.8 Circle Sweep