Caiyiwen's ACM/ICPC Template

Tire Tree

```
#include<cstdio>
#include<cstring>
using namespace std;
const int maxn=(int)1<<20;
const int alphabet_size=26;
int tot,root;
struct trie_node
{
    int count,children[alphabet_size];
    bool exsit;
}tree[maxn];
int create_trie_node()
{
    tree[tot].count=0;
    tree[tot].exsit=false;
    memset(tree[tot].children,-1,sizeof(tree[tot].children));
    tot++;
    return tot-1;
}
void init()
{
    tot=0;
    root=create_trie_node();
}
void trie_insert(int root,char key[])
{
    int pos=0;
    while(key[pos]!='\0')
    {
        if(tree[root].children[key[pos]-'a']==-1)
        {
            tree[root].children[key[pos]-'a']=create_trie_node();
        }
        tree[root].count++;
        root=tree[root].children[key[pos]-'a'];
        pos++;
    }
    tree[root].exsit=true;
    tree[root].count++;
}
int trie_search(int root,char key[])
{
    int pos=0;
    while(key[pos]!='\0'&&root!=-1)
    {
        root=tree[root].children[key[pos]-'a'];
        pos++;
    }
    if(root==-1)
```

```
    {
        return 0;
    }
    else
    {
        return tree[root].count;
    }
}
int main()
{

    return 0;
}
```

Java Fast I/O

```
//快读模板1：更快，但没有next()用于读字符串
import java.io.*;
import java.util.*;

public class standard1 {

    public static void main(String[] args) throws IOException {
        Reader in=new Reader();
        PrintWriter out=new PrintWriter(System.out);



        out.close();
    }


    static class Reader
    {
        final private int BUFFER_SIZE = 1 << 16;
        private DataInputStream din;
        private byte[] buffer;
        private int bufferPointer, bytesRead;

        public Reader()
        {
            din = new DataInputStream(System.in);
            buffer = new byte[BUFFER_SIZE];
            bufferPointer = bytesRead = 0;
        }

        public Reader(String file_name) throws IOException
        {
            din = new DataInputStream(new FileInputStream(file_name));
            buffer = new byte[BUFFER_SIZE];
            bufferPointer = bytesRead = 0;
        }

        public String readLine() throws IOException
        {
            byte[] buf = new byte[64]; // line length
```

```
            int cnt = 0, c;
            while ((c = read()) != -1)
            {
                if (c == '\n')
                    break;
                buf[cnt++] = (byte) c;
            }
            return new String(buf, 0, cnt);
        }

        public int nextInt() throws IOException
        {
            int ret = 0;
            byte c = read();
            while (c <= ' ')
                c = read();
            boolean neg = (c == '-');
            if (neg)
                c = read();
            do
            {
                ret = ret * 10 + c - '0';
            }  while ((c = read()) >= '0' && c <= '9');

            if (neg)
                return -ret;
            return ret;
        }

        public long nextLong() throws IOException
        {
            long ret = 0;
            byte c = read();
            while (c <= ' ')
                c = read();
            boolean neg = (c == '-');
            if (neg)
                c = read();
            do {
                ret = ret * 10 + c - '0';
            }
            while ((c = read()) >= '0' && c <= '9');
            if (neg)
                return -ret;
            return ret;
        }

        public double nextDouble() throws IOException
        {
            double ret = 0, div = 1;
            byte c = read();
            while (c <= ' ')
                c = read();
            boolean neg = (c == '-');
            if (neg)
                c = read();

            do {
```

```
                ret = ret * 10 + c - '0';
            }
            while ((c = read()) >= '0' && c <= '9');

            if (c == '.')
            {
                while ((c = read()) >= '0' && c <= '9')
                {
                    ret += (c - '0') / (div *= 10);
                }
            }

            if (neg)
                return -ret;
            return ret;
        }

        private void fillBuffer() throws IOException
        {
            bytesRead = din.read(buffer, bufferPointer = 0, BUFFER_SIZE);
            if (bytesRead == -1)
                buffer[0] = -1;
        }

        private byte read() throws IOException
        {
            if (bufferPointer == bytesRead)
                fillBuffer();
            return buffer[bufferPointer++];
        }

        public void close() throws IOException
        {
            if (din == null)
                return;
            din.close();
        }
    }

}
```

```
//快读模板2：较慢，但有next()
import java.io.*;
import java.math.*;
import java.util.*;
public class standard2 {
    public static void main(String[] args) {
        InputStream inputStream = System.in;// new
FileInputStream("C:\\Users\\wavator\\Downloads\\test.in");
        OutputStream outputStream = System.out;
        InputReader in = new InputReader(inputStream);
        PrintWriter out = new PrintWriter(outputStream);
        Task solver = new Task();
        solver.solve(in, out);
        out.close();
    }
    static class Task {
```

```
        public void solve(InputReader in, PrintWriter out) {
        }
    }
    static class InputReader {
        public BufferedReader reader;
        public StringTokenizer tokenizer;

        public InputReader(InputStream stream) {
            reader = new BufferedReader(new InputStreamReader(stream), 32768);
            tokenizer = null;
        }
        public String next() {
            while (tokenizer == null || !tokenizer.hasMoreTokens()) {
                try {
                    tokenizer = new StringTokenizer(reader.readLine());
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
            return tokenizer.nextToken();
        }
        public int nextInt() {
            return Integer.parseInt(next());
        }
        public long nextLong() {
            return Long.parseLong(next());
        }
        public double nextDouble() {
            return Double.parseDouble(next());
        }
        public char[] nextCharArray() {
            return next().toCharArray();
        }
//         public boolean hasNext() {
//             try {
//                 return reader.ready();
//             } catch(IOException e) {
//                 throw new RuntimeException(e);
//             }
//         }
        public boolean hasNext() {
            try {
                String string = reader.readLine();
                if (string == null) {
                    return false;
                }
                tokenizer = new StringTokenizer(string);
                return tokenizer.hasMoreTokens();
            } catch (IOException e) {
                return false;
            }
        }
        public BigInteger nextBigInteger() {
            return new BigInteger(next());
        }
        public BigDecimal nextBigDecinal() {
            return new BigDecimal(next());
        }
```

```
    }
}
```

Chinese Remainder Theorem

```
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.Scanner;
public class Main
{
    public static final BigInteger zero=new BigInteger("0");
    public static final BigInteger minus_one=new BigInteger("-1");
    public static final BigInteger one=new BigInteger("1");
    public static BigInteger x,y;
    public static BigInteger extend_gcd(BigInteger a,BigInteger b,boolean
reversed)
    {
        if(a.compareTo(zero)==0&&b.compareTo(zero)==0)
        {
            return minus_one;
        }
        if(b.compareTo(zero)==0)
        {
            if(!reversed)
            {
                x=one;
                y=zero;
            }
            else
            {
                y=one;
                x=zero;
            }
            return a;
        }
        BigInteger d=extend_gcd(b,a.mod(b),!reversed);
        if(!reversed)
        {
            y=y.subtract(a.divide(b).multiply(x));
        }
        else
        {
            x=x.subtract(a.divide(b).multiply(y));
        }
        return d;
    }
    public static BigInteger m[]=new BigInteger[105],a[]=new BigInteger[105];
    public static BigInteger m0,a0;
    public static boolean solve(BigInteger m,BigInteger a)
    {
        BigInteger g=extend_gcd(m0,m,false);
        if(a.subtract(a0).abs().mod(g).compareTo(zero)!=0)
        {
            return false;
        }
        x=x.multiply(a.subtract(a0).divide(g));
```

```
        x=x.mod(m.divide(g));
        a0=a0.add(x.multiply(m0));
        m0=m0.multiply(m.divide(g));
        a0=a0.mod(m0);
        if(a0.compareTo(zero)<0)
        {
            a0=a0.add(m0);
        }
        return true;
    }
    public static boolean MLES(int n)
    {
        boolean flag=true;
        m0=one;
        a0=zero;
        for(int i=0;i<n;i++)
        {
            if(!solve(m[i],a[i]))
            {
                flag=false;
                break;
            }
        }
        return flag;
    }
    public static long Fib[]=new long[(int)1e5+10];
    public static void main(String[]args)
    {
        Fib[0]=1;
        Fib[1]=2;
        int cnt=0;
        for(int i=2;i<=(int)1e6;i++)
        {
            Fib[i]=Fib[i-1]+Fib[i-2];
            if(Fib[i]>(long)1e15)
            {
                cnt=i;
                break;
            }
        }
        Scanner input=new Scanner(System.in);
        int n=input.nextInt();
        for(int i=0;i<n;i++)
        {
            m[i]=new BigInteger(input.next());
            a[i]=new BigInteger(input.next());
        }
        boolean ok=MLES(n);
        if(!ok)
        {
            System.out.println("Tankernb!");
        }
        else
        {
            boolean flag=false;
            for(int i=0;i<cnt;i++)
            {
                if(a0.equals(new BigInteger(Long.toString(Fib[i]))))
```

```
                {
                    flag=true;
                    break;
                }
            }
            if(flag)
            {
                System.out.println("Lbnb!");
            }
            else
            {
                System.out.println("Zgxnb!");
            }
        }
    }
}
```

KMP

```
#include<cstdio>
#include<cstring>
using namespace std;
const int maxn=300;
const int maxN=300;
char t[maxN],p[maxn];
int next[maxn];
void CalcNext()
{
    int pLen=strlen(p);
    next[0]=-1;
    int k=-1,j=0;
    while(j<pLen)
    {
        if(k==-1||p[k]==p[j])
        {
            j++;
            k++;
            if(k!=-1&&p[j]==p[k])
            {
                next[j]=next[k];
            }
            else
            {
                next[j]=k;//No if and be this!!!
            }
        }
        else
        {
            k=next[k];
        }
    }
}
int KMP()
{
    int tLen=strlen(t),pLen=strlen(p);
    int ans=-1;
```

```
    int i=0,j=0;
    while(i<tLen)
    {
        if(j==-1||t[i]==p[j])
        {
            i++;
            j++;
        }
        else
        {
            j=next[j];
        }
        if(j==pLen)
        {
            ans=i-pLen;
            break;
        }
    }
    return ans;
}
int main()
{
    scanf("%s%s",t,p);
    CalcNext();
    printf("%d\n",KMP());
    return 0;
}
```

Segment Tree

```
#include<cstdio>
using namespace std;
const int MAXN=50010;
int a[MAXN],ans[MAXN<<2],lazy[MAXN<<2];
void PushUp(int rt)
{
    ans[rt]=ans[rt<<1]+ans[rt<<1|1];
}
void PushDown(int rt,int ln,int rn)//ln表示左子树元素结点个数，rn表示右子树结点个数
{
    if (lazy[rt])
    {
        lazy[rt<<1]+=lazy[rt];
        lazy[rt<<1|1]+=lazy[rt];
        ans[rt<<1]+=lazy[rt]*ln;
        ans[rt<<1|1]+=lazy[rt]*rn;
        lazy[rt]=0;
    }
}
void Build(int l,int r,int rt)
{
    if(l==r)
    {
        ans[rt]=a[l];
        return;
    }
```

```
    int mid=(l+r)>>1;
    Build(l,mid,rt<<1);
    Build(mid+1,r,rt<<1|1);
    PushUp(rt);
}
void Add(int L,int C,int l,int r,int rt)
{
    if(l==r)
    {
        ans[rt]+=C;
        return;
    }
    int mid=(l+r)>>1;
    //PushDown(rt,mid-l+1,r-mid); 若既有点更新又有区间更新，需要这句话
    if(L<=mid)
    {
        Add(L,C,l,mid,rt<<1);
    }
    else
    {
        Add(L,C,mid+1,r,rt<<1|1);
    }
    PushUp(rt);
}
void Update(int L,int R,int C,int l,int r,int rt)
{
    if(L<=l&&r<=R)
    {
        ans[rt]+=C*(r-l+1);
        lazy[rt]+=C;
        return;
    }
    int mid=(l+r)>>1;
    PushDown(rt,mid-l+1,r-mid);
    if(L<=mid)
    {
        Update(L,R,C,l,mid,rt<<1);
    }
    if(R>mid)
    {
        Update(L,R,C,mid+1,r,rt<<1|1);
    }
    PushUp(rt);
}
long long Query(int L,int R,int l,int r,int rt)
{
    if(L<=l&&r<=R)
    {
        return ans[rt];
    }
    int mid=(l+r)>>1;
    PushDown(rt,mid-l+1,r-mid);//若更新只有点更新，不需要这句
    long long ANS=0;
    if(L<=mid)
    {
        ANS+=Query(L,R,l,mid,rt<<1);
    }
    if(R>mid)
```

```
    {
        ANS+=Query(L,R,mid+1,r,rt<<1|1);
    }
    return ANS;
}
int main()
{

    return 0;
}
```

leftist Tree

```
#include<cstdio>
#include<queue>
#include<algorithm>
using namespace std;
const int maxn=10001,inf=0x3f3f3f3f;
int tot,n;
struct node
{
    int val,lc,rc,dis,fa;
}tree[maxn];
int merge(int x,int y)
{
    if(x==0)
    {
        return y;
    }
    if(y==0)
    {
        return x;
    }
    if(tree[x].val>tree[y].val||(tree[x].val==tree[y].val&&x>y))//这里改成<就莫名不对
    {
        swap(x,y);
    }
    tree[x].rc=merge(tree[x].rc,y);
    tree[tree[x].rc].fa=x;
    if(tree[tree[x].rc].dis>tree[tree[x].lc].dis)
    {
        swap(tree[x].rc,tree[x].lc);
    }
    tree[x].dis=tree[x].rc==0?0:tree[tree[x].rc].dis+1;
    return x;
}
int add(int val,int x)
{
    tree[tot].lc=tree[tot].rc=tree[tot].dis=0;
    tree[tot++].val=val;
    return merge(tot-1,x);
}
int del(int x)
{
    int l=tree[x].lc,r=tree[x].rc;
```

```
    tree[x].fa=tree[x].lc=tree[x].rc=tree[x].dis=0;
    tree[x].val=-inf;
    tree[l].fa=l,tree[r].fa=r;
    return merge(l,r);
}
int find(int i)
{
    while(tree[i].fa&&i!=tree[i].fa)//如果上面没改，第二个条件可以去掉
    {
        i=tree[i].fa;
    }
    return i;
}
int build()
{
    queue<int>q;
    for(int i=1;i<=n;i++)
    {
        q.push(i);
    }
    while(!q.empty())
    {
        if(q.size()==1)
        {
            break;
        }
        else
        {
            int x=q.front();
            q.pop();
            int y=q.front();
            q.pop();
            q.push(merge(x,y));
        }
    }
    int finally=q.front();
    q.pop();
    return finally;
}
int main()
{
    tot=0;
    scanf("%d",&n);
    return 0;
}
```

Chairman Tree

```
#include<cstdio>
using namespace std;
const int N=200010;
struct Node
{
    int num,lch,rch,origin;
    long long sum;
    Node():num(0),sum(0),lch(-1),rch(-1),origin(-1){}
```

```
}tree[N*40];
int tot,a[N],rt[N],curver,qnum;
long long pre_sum[N],qsum;
void init()
{
    tot=0;
    curver=0;
}
int createIndentity(int p)
{
    int cp=tot++;
    tree[cp]=Node();
    tree[cp].origin=p;
    tree[cp].num=tree[p].num;
    tree[cp].sum=tree[p].sum;
    return cp;
}
void pushup(int p)
{
    tree[p].sum=tree[tree[p].lch].sum+tree[tree[p].rch].sum;
    tree[p].num=tree[tree[p].lch].num+tree[tree[p].rch].num;
}
int build(int l,int r)
{
    int p=tot++;
    tree[p]=Node();
    if(l==r)
    {
        tree[p].sum=tree[p].num=0;
        return p;
    }
    int mid=(l+r)>>1;
    tree[p].lch=build(l,mid);
    tree[p].rch=build(mid+1,r);
    pushup(p);
    return p;
}
int add(int p,int l,int r,int x,int y,int z)
{
    int cp=tot++;
    tree[cp]=Node();
    tree[cp].origin=p;
    if(x<=l&&r<=y)
    {
        tree[cp].sum=tree[p].sum+z*(r-l+1);
        tree[cp].num=tree[p].num+r-l+1;
        return cp;
    }
    int mid=(l+r)>>1;
    if(x<=mid)
    {
        tree[cp].lch=add(tree[p].lch,l,mid,x,y,z);
    }
    else
    {
        tree[cp].lch=tree[p].lch;
    }
    if(mid<y)
```

```
    {
        tree[cp].rch=add(tree[p].rch,mid+1,r,x,y,z);
    }
    else
    {
        tree[cp].rch=tree[p].rch;
    }
    pushup(cp);
    return cp;
}
void query(int x,int y,int l,int r,int k)
{
    if(l>k)
    {
        return;
    }
    if(r<=k)
    {
        qsum+=tree[y].sum-tree[x].sum;
        qnum+=tree[y].num-tree[x].num;
        return;
    }
    int mid=(l+r)>>1;
    query(tree[x].lch,tree[y].lch,l,mid,k);
    query(tree[x].rch,tree[y].rch,mid+1,r,k);
}
int main()
{
    init();
    int n,q;
    scanf("%d%d",&n,&q);
    rt[0]=build(1,N);
    for(int i=1;i<=n;i++)
    {
        scanf("%d",&a[i]);
        pre_sum[i]=pre_sum[i-1]+a[i];
        rt[i]=add(rt[i-1],1,N,a[i],a[i],a[i]);
    }
    while(q--)
    {
        int l,r,x,y;
        scanf("%d%d%d%d",&l,&r,&x,&y);
        double l1=0.0,r1=100000.0,eps=1e-10;
        while(r1-l1>eps)
        {
            double mid=(l1+r1)/2;
            qsum=0,qnum=0;
            query(rt[l-1],rt[r],1,N,(int)mid);
            int num=(r-l+1)-qnum;
            double last=qsum+mid*num;
            double now=((double)pre_sum[r]-pre_sum[l-1])/y*(y-x);
            if(last<now)
            {
                l1=mid;
            }
            else
            {
                r1=mid;
```

```
            }
        }
        printf("%.15lf\n",l1);
    }
    return 0;
}
```

Persistant Segment Tree

```
#include<cstdio>
using namespace std;
// 可持久化线段树
const int N=100010;
struct Node
{
    int sum,lch,rch,lazy,origin;
    Node():sum(0),lch(-1),rch(-1),lazy(0),origin(-1){}
}tree[(N<<2)*4];
int tot,a[N],rt[N],curver;
void init()
{
    tot=0;
    curver=0;
}
int createIndentity(int p)
{    // 创建影子节点
    int cp=tot++;
    tree[cp]=Node();
    tree[cp].origin=p;
    tree[cp].sum=tree[p].sum;
    tree[cp].lazy=tree[p].lazy;
    return cp;
}
void pushup(int p)
{
    tree[p].sum=tree[tree[p].lch].sum+tree[tree[p].rch].sum;
}
void pushdown(int p,int l,int r,int m)
{
    int lch=tree[p].lch,rch=tree[p].rch;
    if(lch==-1||rch==-1)
    {
        int o=tree[p].origin;
        lch=tree[p].lch=createIndentity(tree[o].lch);
        rch=tree[p].rch=createIndentity(tree[o].rch);
    }
    tree[lch].lazy+=tree[p].lazy;
    tree[rch].lazy+=tree[p].lazy;
    tree[lch].sum+=tree[p].lazy*(m-l+1);
    tree[rch].sum+=tree[p].lazy*(r-m);
    tree[p].lazy=0;
}
int build(int l,int r)
{
    int p=tot++;
    tree[p]=Node();
```

```
    if(l==r)
    {
        tree[p].sum=a[l];
        return p;
    }
    int mid=(l+r)>>1;
    tree[p].lch=build(l,mid);
    tree[p].rch=build(mid+1,r);
    pushup(p);
    return p;
}
int add(int p,int l,int r,int x,int y,int z)
{
    int cp=tot++;       // create shadow node
    tree[cp]=Node();
    tree[cp].origin=p;  // origin node number, prepared for pushdown operation
    if(x<=l&&r<=y)
    {
        tree[cp].lazy=tree[p].lazy+z;
        tree[cp].sum=tree[p].sum+z*(r-l+1);
        return cp;
    }
    int mid=(l+r)>>1;
    if(tree[p].lazy)
    {
        pushdown(p,l,r,mid);
    }
    if(x<=mid)
    {
        tree[cp].lch=add(tree[p].lch,l,mid,x,y,z);
    }
    else
    {
        tree[cp].lch=tree[p].lch;
    }
    if(mid<y)
    {
        tree[cp].rch=add(tree[p].rch,mid+1,r,x,y,z);
    }
    else
    {
        tree[cp].rch=tree[p].rch;
    }
    pushup(cp);
    return cp;
}
int query(int p,int l,int r,int x,int y)
{
    if(x<=l&&r<=y)
    {
        return tree[p].sum;
    }
    int mid=l+r>>1,ret=0;
    if(tree[p].lazy)
    {
        pushdown(p,l,r,mid);
    }
    if(x<=mid)
```

```
    {
        ret+=query(tree[p].lch,l,mid,x,y);
    }
    if(mid<y)
    {
        ret+=query(tree[p].rch,mid+1,r,x,y);
    }
    return ret;
}
int main() {
    int n;
    scanf("%d", &n);
    for (int i=1;i<=n;++i) scanf("%d", a+i);
    //
    init();
    rt[curver]=build(1,n);
    for (;;){
        int u,v,w;
        char q[3];
        printf("q/m:");
        scanf("%s", q);
        if (q[0]=='m') {
            printf("Please input u, v, w and we will add w to [u,v]: ");
            scanf("%d%d%d", &u, &v, &w);
            rt[curver+1]=add(rt[curver],1,n,u,v,w);
            ++curver;
            for (int i=1;i<=n;++i) {
                printf("%d ", query(rt[curver],1,n,i,i));
            }
            puts("");
        }else {
            printf("Please input ver: ");
            scanf("%d", &w);
            for (int i=1;i<=n;++i) {
                printf("%d ", query(rt[w],1,n,i,i));
            }
            puts("");
        }
    }
    return 0;
}
```

Binary Exponatiation

```
long long power(long long a,long long b)
{
    long long ans=1,base=a;
    while(b)
    {
        if(b&1)
        {
            ans*=base;
        }
        base*=base;
        b>>=1;
    }
```

```
    return ans;
}
```

Matrix Binary Exponatiation

```
#include<cstdio>
#include<cstring>
using namespace std;
const int N=10,mod=1;
int tmp[N][N];
void multi(int a[][N],int b[][N],int n)
{
    memset(tmp,0,sizeof(tmp));
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            for(int k=0;k<n;k++)
            {
                tmp[i][j]=(tmp[i][j]+a[i][k]*b[k][j]%mod)%mod;
            }
        }
    }
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            a[i][j]=tmp[i][j];
        }
    }
}
int res[N][N];
void Pow(int a[][N],int n)
{
    memset(res,0,sizeof(res));//n是幂，N是矩阵大小
    for(int i=0;i<N;i++)
    {
        res[i][i]=1;
    }
    while(n)
    {
        if(n&1)
        {
            multi(res,a,N);//res=res*a;复制直接在multi里面实现了；
        }
        multi(a,a,N);//a=a*a
        n>>=1;
    }
}
int main()
{

    return 0;
}
```

Union-Find Set

```
#include<cstdio>
using namespace std;
const int maxn=1001;
int par[maxn],Rank[maxn];
void makeSet(int size)
{
    for(int i=1;i<=size;i++)
    {
        par[i]=i;
        Rank[i]=0;
    }
}
int find(int x)
{
    int k,j,r;
    r=x;
    while(r!=par[r])
    {
        r=par[r];
    }
    k=x;
    while(k!=r)
    {
        j=par[k];
        par[k]=r;
        k=j;
    }
    return r;
}
void unite(int x,int y)
{
    x=find(x);
    y=find(y);
    if(x==y)
    {
        return;
    }
    if(Rank[x]<Rank[y])
    {
        par[x]=y;
    }
    else
    {
        par[y]=x;
        if(Rank[x]==Rank[y])
        {
            Rank[x]++;
        }
    }
}
int main()
{
    return 0;
}
```

AC Automation

```
#include<cstdio>
#include<cstring>
#include<queue>
using namespace std;
const int maxn=(int)1<<20;
const int alphabet_size=26;
int tot,root;
struct node
{
    int count,children[alphabet_size],fail;
    bool exsit;
}tree[maxn];
bool vis[maxn];
int create_trie_node()
{
    tree[tot].count=0;
    tree[tot].exsit=false;
    tree[tot].fail=-1;
    memset(tree[tot].children,-1,sizeof(tree[tot].children));
    tot++;
    return tot-1;
}
void init()
{
    memset(vis,false,sizeof(vis));
    tot=0;
    root=create_trie_node();
}
void trie_insert(int rot,char key[])
{
    int pos=0;
    while(key[pos]!='\0')
    {
        if(tree[rot].children[key[pos]-'a']==-1)
        {
            tree[rot].children[key[pos]-'a']=create_trie_node();
        }
        rot=tree[rot].children[key[pos]-'a'];
        pos++;
    }
    tree[rot].exsit=true;
    tree[rot].count++;
}
void get_fail()
{
    queue<int>q;
    tree[root].fail=root;
    for(int i=0;i<alphabet_size;i++)
    {
        if(tree[root].children[i]!=-1)
        {
            tree[tree[root].children[i]].fail=root;
            q.push(tree[root].children[i]);
        }
```

```
        else
        {
            tree[root].children[i]=root;
        }
    }
    while(!q.empty())
    {
        int cr=q.front();
        q.pop();
        for(int i=0;i<alphabet_size;i++)
        {
            if(tree[cr].children[i]!=-1)
            {
                tree[tree[cr].children[i]].fail=tree[tree[cr].fail].children[i];
                q.push(tree[cr].children[i]);
            }
            else
            {
                tree[cr].children[i]=tree[tree[cr].fail].children[i];
            }
        }
        int tmp=tree[cr].fail;
        while(tmp!=root&&!tree[tmp].exsit)
        {
            tmp=tree[tmp].fail;
        }
        if(tree[tmp].exsit)
        {
            tree[cr].exsit=true;
        }
    }
}
int Match(char s[])
{
    int now=root;
    int ans=0;
    int len=strlen(s);
    for(int i=0;i<len;i++)
    {
        int k=s[i]-'a';
        if(tree[now].children[k]!=-1)
        {
            now=tree[now].children[k];
        }
        else
        {
            int p=tree[now].fail;
            while(p!=-1&&tree[p].children[k]==-1)
            {
                p=tree[p].fail;
            }
            if(p==-1)
            {
                now=root;
            }
            else
            {
                now=tree[p].children[k];
```

```
            }
        }
        if(tree[now].exsit)
        {
            int tn=now;
            while(tn!=-1&&tn!=root&&!vis[tn])
            {
                if(tree[tn].exsit)
                {
                    ans+=tree[tn].count;
                }
                //tree[tn].count=0;
                vis[tn]=true;
                tn=tree[tn].fail;
            }
        }
    }
    return ans;
}
int main()
{

    return 0;
}
```

Big Integer

```
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
int ll_to_bi(long long num,char*bi)
{
    int pos=0;
    while(num)
    {
        bi[pos++]=num%10;
        num/=10;
    }
    return pos;
}
int string_to_bi(char*num)
{
    int len=strlen(num);
    for(int i=0;i<len;i++)
    {
        num[i]-='0';
    }
    for(int i=0;i<len>>1;i++)
    {
        swap(num[i],num[len-i-1]);
    }
    return len;
}
void print_bi(char*bi,int len)
{
```

```
    for(int i=len-1;i>=0;i--)
    {
        putchar(bi[i]+'0');
    }
}
int proceed(char*num,int len)
{
    bool proceeded=false;
    for(int i=0;i<len-1;i++)
    {
        num[i+1]+=num[i]/10;
        num[i]%=10;
    }
    if(num[len-1]>9)
    {
        num[len]=num[len-1]/10;
        num[len-1]%=10;
        proceeded=true;
    }
    return proceeded?len+1:len;
}
int add(char*a1,int len1,char*a2,int len2,char*res)
{
    int min_len=min(len1,len2);
    int max_len=max(len1,len2);
    char*remain;
    if(min_len==len1)
    {
        remain=a2;
    }
    else
    {
        remain=a1;
    }
    for(int i=0;i<min_len;i++)
    {
        res[i]=a1[i]+a2[i];
    }
    for(int i=min_len;i<max_len;i++)
    {
        res[i]=remain[i];
    }
    return proceed(res,max_len);
}
int multiply(char*a1,int len1,char*a2,int len2,char*res)
{
    for(int i=0;i<len1+len2;i++)
    {
        res[i]=0;
    }
    int len=0;
    for(int i=0;i<len2;i++)
    {
        for(int j=0;j<len1;j++)
        {
            res[i+j]+=a1[j]*a2[i];
        }
        len=proceed(res,max(len,i+len1));
```

```
    }
    return len;
}
int power(char*num,int ori_len,long long times,char*res)
{
    res[0]=1;
    int len_res=1;
    char*base=new char[ori_len];
    int len_base=ori_len;
    for(int i=0;i<len_base;i++)
    {
        base[i]=num[i];
    }
    while(times)
    {
        if(times&1)
        {
            char*res_tmp=new char[len_res];
            for(int i=0;i<len_res;i++)
            {
                res_tmp[i]=res[i];
            }
            len_res=multiply(res_tmp,len_res,base,len_base,res);
            delete[]res_tmp;
        }
        char*base_tmp=new char[len_base];
        char*base_res=new char[len_base<<1];
        for(int i=0;i<len_base;i++)
        {
            base_tmp[i]=base[i];
        }
        len_base=multiply(base_tmp,len_base,base_tmp,len_base,base_res);
        delete[]base_tmp;
        delete[]base;
        base=base_res;
        times>>=1;
    }
    delete[]base;
    return len_res;
}
int main()
{
    char a1[1000],a2[1000],res[1000];
    long long times;
    scanf("%s%lld",a1,&times);
    int len1=string_to_bi(a1);
    int len=power(a1,len1,times,res);
    print_bi(res,len);
    printf("\n");
    return 0;
}
```