目录

[大数模板 2](#_Toc420440111)

[快速幂 4](#_Toc420440112)

[矩阵快速幂 4](#_Toc420440113)

[背包问题 4](#_Toc420440114)

[KMP 4](#_Toc420440115)

[Manacher 5](#_Toc420440116)

[求素数表 5](#_Toc420440117)

[组合数计算 5](#_Toc420440118)

[数组存图边 7](#_Toc420440119)

[后缀数组 7](#_Toc420440120)

[LIS 8](#_Toc420440121)

## 大数模板

#include<iostream>

#include<string>

#include<cstring>

#include<iomanip>

#include<algorithm>

using namespace std;

#define MAXLEN 500//大数的位数

#define MAXN 9999//每一位int大小

#define DLEN 4//每一位int长度

#define MAXSIZE 10

class BigNum

{

private:

int a[MAXLEN]; //可以控制大数的位数

int len; //大数长度

public:

BigNum(){ len = 1;memset(a,0,sizeof(a)); } //构造函数

BigNum(const int); //将一个int类型的变量转化为大数

BigNum(const char\*); //将一个字符串类型的变量转化为大数

BigNum(const BigNum &); //拷贝构造函数

BigNum &operator=(const BigNum &); //重载赋值运算符，大数之间进行赋值运算

friend istream& operator>>(istream&, BigNum&); //重载输入运算符

friend ostream& operator<<(ostream&, BigNum&); //重载输出运算符

BigNum operator+(const BigNum &) const; //重载加法运算符，两个大数之间的相加运算

BigNum operator-(const BigNum &) const; //重载减法运算符，两个大数之间的相减运算

BigNum operator\*(const BigNum &) const; //重载乘法运算符，两个大数之间的相乘运算

BigNum operator/(const int &) const; //重载除法运算符，大数对一个整数进行相除运算

BigNum operator^(const int &) const; //大数的n次方运算

int operator%(const int &) const; //大数对一个int类型的变量进行取模运算

bool operator>(const BigNum & T)const; //大数和另一个大数的大小比较

bool operator>(const int & t)const; //大数和一个int类型的变量的大小比较

void print(); //输出大数

};

BigNum::BigNum(const int b) //将一个int类型的变量转化为大数

{

int c,d = b;

len = 0;

memset(a,0,sizeof(a));

while(d > MAXN)

{

c = d - (d / (MAXN + 1)) \* (MAXN + 1);

d = d / (MAXN + 1);

a[len++] = c;

}

a[len++] = d;

}

BigNum::BigNum(const char\*s) //将一个字符串类型的变量转化为大数

{

int t,k,index,l,i;

memset(a,0,sizeof(a));

l=strlen(s);

len=l/DLEN;

if(l%DLEN)

len++;

index=0;

for(i=l-1;i>=0;i-=DLEN)

{

t=0;

k=i-DLEN+1;

if(k<0)

k=0;

for(int j=k;j<=i;j++)

t=t\*10+s[j]-'0';

a[index++]=t;

}

}

BigNum::BigNum(const BigNum & T) : len(T.len) //拷贝构造函数

{

int i;

memset(a,0,sizeof(a));

for(i = 0 ; i < len ; i++)

a[i] = T.a[i];

}

BigNum & BigNum::operator=(const BigNum & n) //重载赋值运算符，大数之间进行赋值运算

{

int i;

len = n.len;

memset(a,0,sizeof(a));

for(i = 0 ; i < len ; i++)

a[i] = n.a[i];

return \*this;

}

istream& operator>>(istream & in, BigNum & b) //重载输入运算符

{

char ch[MAXSIZE\*4];

int i = -1;

in>>ch;

int l=strlen(ch);

int count=0,sum=0;

for(i=l-1;i>=0;)

{

sum = 0;

int t=1;

for(int j=0;j<4&&i>=0;j++,i--,t\*=10)

{

sum+=(ch[i]-'0')\*t;

}

b.a[count]=sum;

count++;

}

b.len =count++;

return in;

}

ostream& operator<<(ostream& out, BigNum& b) //重载输出运算符

{

int i;

cout << b.a[b.len - 1];

for(i = b.len - 2 ; i >= 0 ; i--)

{

cout.width(DLEN);

cout.fill('0');

cout << b.a[i];

}

return out;

}

BigNum BigNum::operator+(const BigNum & T) const //两个大数之间的相加运算

{

BigNum t(\*this);

int i,big; //位数

big = T.len > len ? T.len : len;

for(i = 0 ; i < big ; i++)

{

t.a[i] +=T.a[i];

if(t.a[i] > MAXN)

{

t.a[i + 1]++;

t.a[i] -=MAXN+1;

}

}

if(t.a[big] != 0)

t.len = big + 1;

else

t.len = big;

return t;

}

BigNum BigNum::operator-(const BigNum & T) const //两个大数之间的相减运算

{

int i,j,big;

bool flag;

BigNum t1,t2;

if(\*this>T)

{

t1=\*this;

t2=T;

flag=0;

}

else

{

t1=T;

t2=\*this;

flag=1;

}

big=t1.len;

for(i = 0 ; i < big ; i++)

{

if(t1.a[i] < t2.a[i])

{

j = i + 1;

while(t1.a[j] == 0)

j++;

t1.a[j--]--;

while(j > i)

t1.a[j--] += MAXN;

t1.a[i] += MAXN + 1 - t2.a[i];

}

else

t1.a[i] -= t2.a[i];

}

t1.len = big;

while(t1.a[len - 1] == 0 && t1.len > 1)

{

t1.len--;

big--;

}

if(flag)

t1.a[big-1]=0-t1.a[big-1];

return t1;

}

BigNum BigNum::operator\*(const BigNum & T) const //两个大数之间的相乘运算

{

BigNum ret;

int i,j,up;

int temp,temp1;

for(i = 0 ; i < len ; i++)

{

up = 0;

for(j = 0 ; j < T.len ; j++)

{

temp = a[i] \* T.a[j] + ret.a[i + j] + up;

if(temp > MAXN)

{

temp1 = temp - temp / (MAXN + 1) \* (MAXN + 1);

up = temp / (MAXN + 1);

ret.a[i + j] = temp1;

}

else

{

up = 0;

ret.a[i + j] = temp;

}

}

if(up != 0)

ret.a[i + j] = up;

}

ret.len = i + j;

while(ret.a[ret.len - 1] == 0 && ret.len > 1)

ret.len--;

return ret;

}

BigNum BigNum::operator/(const int & b) const //大数对一个整数进行相除运算

{

BigNum ret;

int i,down = 0;

for(i = len - 1 ; i >= 0 ; i--)

{

ret.a[i] = (a[i] + down \* (MAXN + 1)) / b;

down = a[i] + down \* (MAXN + 1) - ret.a[i] \* b;

}

ret.len = len;

while(ret.a[ret.len - 1] == 0 && ret.len > 1)

ret.len--;

return ret;

}

int BigNum::operator %(const int & b) const //大数对一个int类型的变量进行取模运算

{

int i,d=0;

for (i = len-1; i>=0; i--)

{

d = ((d \* (MAXN+1))% b + a[i])% b;

}

return d;

}

BigNum BigNum::operator^(const int & n) const //大数的n次方运算

{

BigNum t,ret(1);

int i;

if(n<0)

exit(-1);

if(n==0)

return 1;

if(n==1)

return \*this;

int m=n;

while(m>1)

{

t=\*this;

for( i=1;i<<1<=m;i<<=1)

{

t=t\*t;

}

m-=i;

ret=ret\*t;

if(m==1)

ret=ret\*(\*this);

}

return ret;

}

bool BigNum::operator>(const BigNum & T) const //大数和另一个大数的大小比较

{

int ln;

if(len > T.len)

return true;

else if(len == T.len)

{

ln = len - 1;

while(a[ln] == T.a[ln] && ln >= 0)

ln--;

if(ln >= 0 && a[ln] > T.a[ln])

return true;

else

return false;

}

else

return false;

}

bool BigNum::operator >(const int & t) const //大数和一个int类型的变量的大小比较

{

BigNum b(t);

return \*this>b;

}

void BigNum::print() //输出大数

{

int i;

cout << a[len - 1];

for(i = len - 2 ; i >= 0 ; i--)

{

cout.width(DLEN);

cout.fill('0');

cout << a[i];

}

cout << endl;

}

int main(void)

{

int i,n;

BigNum x[101]; //定义大数的对象数组

x[0]=1;

for(i=1;i<101;i++)

x[i]=x[i-1]\*(4\*i-2)/(i+1);

while(scanf("%d",&n)==1 && n!=-1)

{

x[n].print();

}

}

## 快速幂

long long mod;

long long myPow(long long a,long long b)

{

long long r=1, base=a%mod;

while ( b )

{

if ( b&1 ){

r \*= base;

r %= mod;

}

base \*= base;

base %= mod;

b>>=1;

}

return r;

}

## 矩阵快速幂

const int MAXN=100;

struct Mat {

int mat[MAXN][MAXN];

};

int N;//矩阵大小

long long MOD;//幂次

Mat operator \* (Mat a, Mat &b) {

Mat c;

memset (c.mat, 0, sizeof(c.mat));

for (int k = 0; k < N; k++ ) {//n为矩阵大小

for(int i = 0; i < N; i++ ) {

for(int j = 0; j < N; j++ ) {

c.mat[i][j] += a.mat[i][k] \* b.mat[k][j];

//c.mat[i][j] %= MOD;

}

}

}

return c;

}

Mat operator ^ (Mat a, long long k) {

Mat c;

for(int i = 0; i < N; i++){

for(int j = 0; j < N; j++){

c.mat[i][j] = (i == j); //初始化为单位矩阵

}

}

while( k )

{

if(k&1){

c = c\*a;

}

k >>= 1;

a = a\*a;

}

return c;

}

## 背包问题

01背包

void ZeroToOne(int weight, int value)

{

for (int i=maxV; i>=weight; i--){

bag[i] = max(bag[i], bag[i-weight]+value);

}

}

完全背包

void CompleteBag(int weight, int value)

{

for (int i=weight; i<=maxV; i++){

bag[i] = max(bag[i], bag[i-weight]+value);

}

}

多重背包

void MultipleBag(int num, int weight, int value)

{

if ( num\*weight>=maxV ){

CompleteBag(weight, value);

return;

}

int k = 1;

while ( k<num )

{

ZeroToOne(weight\*k, value\*k);

num -= k;

k \*= 2;

}

ZeroToOne(weight\*num, value\*num);

}

分组背包

凡是物品间有相互关系的都可以当做分组背包，多变

每组最多取一个：把同一组的物品视为并列的物品，在同一层上进行背包（hdu1712）

每组至少取一个：增加一条选择路径：上一组完成的背包+当前物品（hdu3033）

01背包最优解个数（hdu2126）

用bag数组存最优解，同时num数组存最优解个数

01背包第K优解（hdu2639）

用bag[v][n]更新花费v以内的前n个解

01背包花费为负数（poj2184）

花费为负时，v-cost>v（每个v由v-cost与自身比较），所以当花费为负从小到大遍历

花费为正时，同普通01背包，从大到小遍历

01背包花费为double（poj2184）

将花费与价值对调后进行背包，逆向思维

状压+01背包（poj2923）

将所有可能的状态进行状压存储，以这些状态为花费进行背包，花费判断为：v&sig==0（没有同一件物品被重复）

## KMP

参数：原串和模式串

返回值：在原串中模式串的个数

可选：单个字符是否可以被重复使用于多个匹配中, 记录每一位0~i的前缀与模式串前缀的匹配长度

———————————————————————————

const int MAXN = 100000 ;

int nextLink[MAXN+100];

int numKMP[MAXN+100];

//KMP，返回匹配到模式串的个数

int xKMP(char \* mOri, char \* mPar)

{

int mP, mQ;

int mOriLen=strlen(mOri), mParLen=strlen(mPar);

//构造next数组

nextLink[0] = -1;

mP = 0, mQ = -1;

while ( mP<mParLen )

{

if ( mQ==-1 || mPar[mQ]==mPar[mP] ){

nextLink[++mP] = ++mQ;

numKMP[mP-1] = mQ;//记录每一位0~i的前缀与模式串前缀的匹配长度

}

else {

mQ = nextLink[mQ];

}

}

//kmp

mP = mQ = 0;

int mNum = 0;

while ( mP<mOriLen )

{

if ( mQ==-1 || mOri[mP]==mPar[mQ] ){

mQ++, mP++;

}

else{

mQ = nextLink[mQ];

}

if ( mQ==mParLen ){

mNum++;

mQ = 0;//不可重复使用字符

}

}

return mNum;

}

## Manacher

参数：原串

返回值：最长回文串长度

附：同时记录每一位的最长回文串长度

————————————————————————————————

const int MAXN = 110000 ;

char strOri[MAXN+100];

int maxPalindrome[MAXN\*2+100];//最长回文

//manacher，返回最长的回文，同时记录下每位的最长回文

int xManacher(char \* mOri)

{

char mStr[MAXN\*2+100];

//构造新串

int mLen = strlen(mOri);

mStr[0]='#';

for (int i = 0; i < mLen ; i++)

{

mStr[i\*2+1] = mOri[i];

mStr[i\*2+2] = '#';

}

//manacher

int k = 1, maxAns = 1;

maxPalindrome[0] = 1, maxPalindrome[1] = 3;

mLen = mLen\*2+1;

for (int i = 2; i < mLen ; i++)

{

maxPalindrome[i] = min( maxPalindrome[2\*k-i], maxPalindrome[k]-2\*(i-k) );

int a = maxPalindrome[i]/2+1;

while ( i-a>=0 && mStr[i-a]==mStr[i+a] ) a++;

maxPalindrome[i] = a\*2-1;

k = ( (maxPalindrome[k]/2+k)>(maxPalindrome[i]/2+i) )? k:i;

maxAns = max(maxAns, maxPalindrome[i]/2);

}

return maxAns;

}

## 求素数表

参数：素数最大范围

返回值：素数个数

————————————————————————————————

// 求素数表

int prime[1000000+100] = {0};

bool isPrime[1000000+100];

int getPrime(int maxn)

{

int n = 0;

int k = (int) sqrt(maxn + 0.5);

memset(isPrime, true, sizeof(isPrime));

for (int i = 2; i <= maxn; i++)

{

if ( isPrime[i] )

{

prime[n++] = i;

if ( i <= k ){

for (int j = i \* i; j <= maxn; j += i) isPrime[j] = false;

}

}

}

return n;

}

## 组合数计算

**1一般组合数计算**

mod的要求：无

有效范围：1<=n,m<=60

————————————————————————————————

//一般方法求C（n，m）最后取模。C(62,28)溢出。有效范围1<n,m<=60

ll CNormal(int n, int m)

{

if ( m>n ) return 0;

ll ans = 1;

for (int i=1; i<=m; i++)

{

ans \*= (n – m + i);

ans /= i;

}

return ans % mod;

}

**2杨辉三角计算组合数**

mod的要求：无

有效范围：1<=n,m<=1000

————————————————————————————————

//杨辉三角求C（n，m）

ll matrix[110][110]={0};

ll CMatrix(int n, int m)

{

if ( m>n ) return 0;

ll ans = 0;

matrix[1][0] = matrix[1][1] = 1;

for (int i=2; i<=n; i++)

{

int k = min(i, m);

for(int j=0; j<=k; j++)

{

if ( j==0 || j==i ) matrix[i][j] = 1;

else matrix[i][j] = (matrix[i-1][j] + matrix[i-1][j-1]) % mod;

}

}

return matrix[n][m];

}

**3利用乘法逆元求组合数**

mod的要求：mod为素数，且m<mod或GCD(mod, m)=1

有效范围：1<=n,m<=10^6

————————————————————————————————

//获得a在mod下的乘法逆元

ll GetInverse(ll a, ll mod)

{

ll ans = 1, n = mod - 2;

a %= mod;

while ( n )

{

if ( n&1 ){

ans = ans \* a % mod;

}

n >>= 1;

a = a \* a % mod;

}

return ans;

}

//逆元求C（n，m）%mod

ll C(ll n, ll m)

{

if ( m>n ) return 0;

ll ans = 1;

for (int i=1; i<=m; i++)

{

ll a = (n + i - m) % mod;

ll b = i % mod;

ans = ans \* (a \* GetInverse(b, mod) % mod) % mod;

}

return ans;

}

**4分解因子求组合数**

mod的要求：无（可为合数）

有效范围：1<=n,m,p<=10^6

————————————————————————————————

// 求素数表

int prime[1000000] = {0};

bool isPrime[1000000];

void getPrime(int maxn)

{

int n = 0;

memset(isPrime, true, sizeof(isPrime));

for (int i=2; i<=maxn; i++)

{

if ( isPrime[i] )

{

prime[n++] = i;

for (int j=i\*2; j<=maxn; j+=i) isPrime[j] = false;

}

}

}

//快速幂

long long myPow(long long a,long long b)

{

long long r=1, base=a % mod;

while ( b )

{

if ( b&1 ){

r \*= base;

r %= mod;

}

base \*= base;

base %= mod;

b >>= 1;

}

return r;

}

//获得n！中因子p的个数

ll getPNum(ll n, ll p)

{

ll ans = 0;

while ( n )

{

ans += n / p;

n /= p;

}

return ans;

}

//利用因子求C(n,m)

ll CFactor(ll n, ll m)

{

if ( m>n ) return 0;

ll ans = 1;

for (int i=0; prime[i]!=0 && prime[i]<=n; i++)

{

ll a = getPNum(n, prime[i]);

ll b = getPNum(m, prime[i]);

ll c = getPNum(n - m, prime[i]);

a -= (b + c);

ans \*= myPow(prime[i], a);

ans %= mod;

}

return ans;

}

**5 Lucas定理**

mod的要求：素数

有效范围：1<=n,m,p<=10^9

————————————————————————————————

//获得a在mod下的乘法逆元

ll GetInverse(ll a, ll mod)

{

ll ans = 1, n = mod - 2;

a %= mod;

while ( n )

{

if ( n&1 ){

ans = ans \* a % mod;

}

n >>= 1;

a = a \* a % mod;

}

return ans;

}

//逆元求C（n，m）%mod

ll C(ll n, ll m)

{

if ( m>n ) return 0;

ll ans = 1;

for (int i=1; i<=m; i++)

{

ll a = (n + i - m) % mod;

ll b = i % mod;

ans = ans \* (a \* GetInverse(b, mod) % mod) % mod;

}

return ans;

}

//利用lucas定理求C（n，m）

ll CLucas(ll n, ll m)

{

//if ( m==0 ) return 1;

//return C(n % mod, m % mod) \* CLucas(n / mod, m / mod) % mod;

ll ans = 1;

while ( m )

{

ans = ans \* C(n % mod, m % mod) % mod;

n /= mod;

m /= mod;

}

return ans;

}

**6半预处理型**

mod的要求：p<10^6，且为素数

有效范围：1<=n,m<=10^9

————————————————————————————————

//半预处理

const ll MAXN = 100000;

ll fac[MAXN+100];

void init(int mod)

{

fac[0] = 1;

for (int i=1; i<mod; i++){

fac[i] = fac[i-1] \* i % mod;

}

}

//半预处理逆元求C（n，m）%mod

ll C(ll n, ll m)

{

if ( m>n ) return 0;

return fac[n] \* (GetInverse(fac[m]\*fac[n-m], mod)) % mod;

}

//利用lucas定理求C（n，m）

ll CLucas(ll n, ll m)

{

//if ( m==0 ) return 1;

//return C(n % mod, m % mod) \* CLucas(n / mod, m / mod) % mod;

ll ans = 1;

while ( m )

{

ans = ans \* C(n % mod, m % mod) % mod;

n /= mod;

m /= mod;

}

return ans;

}

**7全预处理型**

mod的要求：p<10^4，且为素数

有效范围：1<=n,m<=10^9

————————————————————————————————

//阶乘预处理

int prime[10000+10] = {0};

bool isPrime[10000+10];

int inverse[10000][1250] = {0};

int fac[10000][1250] = {0};

void init(int maxn)

{

int n = 0;

memset(isPrime, true, sizeof(isPrime));

for (int i=2; i<maxn; i++)

{

if ( isPrime[i] )

{

prime[i] = n;

for (int j=i\*2; j<maxn; j+=i) isPrime[j] = false;

//计算逆元

inverse[0][n] = inverse[1][n] = 1;

for (int j=2; j<i; j++)

{

inverse[j][n] = (i - i/j) \* inverse[i%j][n] % i;

}

//预处理阶乘

fac[0][n] = fac[1][n] = 1;

for (int j=2; j<i; j++)

{

fac[j][n] = j \* fac[j-1][n] % i;//普通阶乘

inverse[j][n] = inverse[j][n] \* inverse[j-1][n] % i;//逆元阶乘

}

n++;

}

}

}

//全预处理求C（n，m）%mod

ll C(int n, int m)

{

if ( m>n ) return 0;

int num = prime[i];

return (ll)fac[n][num] \* inverse[m][num] % i \* inverse[n-m][num] % i;

}

//利用lucas定理求C（n，m）

ll CLucas(ll n, ll m)

{

//if ( m==0 ) return 1;

//return C(n % mod, m % mod) \* CLucas(n / mod, m / mod) % mod;

ll ans = 1;

while ( m )

{

ans = ans \* C(n % mod, m % mod) % mod;

n /= mod;

m /= mod;

}

return ans;

}

## 数组存图边

参数：有向边a->b（若存无向边即存双向）

数组：headE[i]以i为起点的最后一条边的编号

nextE[i]第i条边的上一条边的编号

toP[i]第i条边所指向的点的编号

int headE[MAXN\*2+10], nextE[MAXN\*2+10], toV[MAXN\*2+10];

int cn;

void addEdge(int a, int b)

{

toV[++cn] = b;

nextE[cn] = headE[a];

headE[a] = cn;

}

## 后缀数组

参数：构造的字符串

数组：rank[i]以i为起点的后缀的排名(0 – len-1)

sa[i]第i名的后缀的起点(1 - len)

height[i]排名为i和i-1的后缀的最长公共前缀(1 - len)

RMQArray[][]存放RMQ的询问数组

const int MAXN = 400005;

char ori[MAXN];

int rk[MAXN],sa[MAXN],height[MAXN],w[MAXN],wa[MAXN],res[MAXN];

int RMQArray[MAXN][20];

void getSa (int len,int up) {

int \*k = rk,\*id = height,\*r = res, \*cnt = wa;

for(int i = 0;i < up; i++) cnt[i] = 0;

for(int i = 0;i < len; i++) cnt[k[i] = w[i]]++;

for(int i = 0;i < up; i++) cnt[i+1] += cnt[i];

for(int i = len - 1; i >= 0; i--) {

sa[--cnt[k[i]]] = i;

}

int d = 1,p = 0;

while(p < len){

for(int i = len - d; i < len; i++) id[p++] = i;

for(int i = 0;i < len; i++) if(sa[i] >= d) id[p++] = sa[i] - d;

for(int i = 0;i < len; i++) r[i] = k[id[i]];

for(int i = 0;i < up; i++) cnt[i] = 0;

for(int i = 0;i < len; i++) cnt[r[i]]++;

for(int i = 0;i < up; i++) cnt[i+1] += cnt[i];

for(int i = len - 1; i >= 0; i--) {

sa[--cnt[r[i]]] = id[i];

}

swap(k,r);

p = 0;

k[sa[0]] = p++;

for(int i = 0;i < len - 1; i++) {

if(sa[i]+d < len && sa[i+1]+d <len &&r[sa[i]] == r[sa[i+1]]&& r[sa[i]+d] == r[sa[i+1]+d])

k[sa[i+1]] = p - 1;

else k[sa[i+1]] = p++;

}

if(p >= len) return ;

d \*= 2,up = p, p = 0;

}

}

void getHeight(int len) {

for(int i = 0;i < len; i++) rk[sa[i]] = i;

height[0] = 0;

for(int i = 0,p = 0; i < len - 1; i++) {

int j = sa[rk[i]-1];

while(i+p < len&& j+p < len&& w[i+p] == w[j+p]) {

p++;

}

height[rk[i]] = p;

p = max(0, p - 1);

}

}

void getRMQ(int len)

{

int k = (int)(log(len - 1 + 0.5) / log(2.0)) + 1;

for (int i = 1; i < len; i++) RMQArray[i][0] = height[i];

for (int i = 1; i < k; i++){

for (int j = 1; j + (1<<i) <= len; j++){

RMQArray[j][i] = min( RMQArray[j][i-1], RMQArray[j+(1<<(i-1))][i-1] );

}

}

}

//参数：ori中的位置

int qRMQ(int l, int r)

{

int L = min(rk[l], rk[r]) + 1;

int R = max(rk[l], rk[r]);

int k = (int)(log(R - L + 0.5 + 1) / log(2.0));

return min( RMQArray[L][k], RMQArray[R-(1<<k)+1][k]);

}

int getSuffix(char s[]) {

int len = strlen(s), up = 0;

for(int i = 0; i < len; i++) {

w[i] = s[i];

up = max(up, w[i]);

}

w[len++] = 0;

getSa(len,up+1);

getHeight(len);

getRMQ(len);

// for (int i = 1; i < len; i++) printf("%d(sa=%d)=%s\n", i, sa[i], ori + sa[i]);

// for(int i = 1; i < len; i++) printf("%d=%d ", i, height[i]);

// printf("\n");

return len;

}

## LIS

参数：数列长度n

返回值：最长上升子序列长度

const int MAXN = 1000000 + 10;

int num[MAXN];

vector<int> ans;

int getLIS(int n)

{

ans.clear();

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

vector<int>::iterator it = lower\_bound(ans.begin(), ans.end(), num[i]);

if ( it == ans.end() ) ans.push\_back(num[i]);

else \*it = num[i];

}

return ans.size();

}

## 数位dp

//#define DEBUG

typedef long long ll;

int num[15];

ll f[15][10][15][2];

ll dfs(int n, int per, int m, bool flag, int flag2)

{

if ( !n ) return (flag2 && !m);

if ( !flag && ~f[n][per][m][flag2] ) return f[n][per][m][flag2];

ll ans = 0;

int sig = (flag)? num[n]:9;

for (int i = 0; i <= sig; i++){

int cPer = i;

int cM = (m \* 10 + i) % 13;

ans += dfs(n - 1, cPer, cM, flag && (i == sig), flag2 || (per == 1 && cPer == 3));

}

if ( !flag ) f[n][per][m][flag2] = ans;

return ans;

}

ll getAns(ll x)

{

#ifdef DEBUG

cout<<"num="<<x<<endl;

#endif

int cn = 0;

while ( x ) num[++cn] = x % 10, x /= 10;

return dfs(cn, 0, 0, 1, 0);

}