堆

2022年1月7日　　21:19

堆是一个完全二叉树
小根堆：从上到下递增
大根堆：从上到下递减

838堆排序

```
//如何手写一个堆？完全二叉树 5个操作
//1. 插入一个数         heap[ ++ size] = x; up(size);
//2. 求集合中的最小值   heap[1]
//3. 删除最小值         heap[1] = heap[size]; size -- ;down(1);
//4. 删除任意一个元素   heap[k] = heap[size]; size -- ;up(k); down(k);
//5. 修改任意一个元素   heap[k] = x; up(k); down(k);
#include<bits/stdc++.h>
using namespace std;
const int N=1e6+10;
int n,m;
int h[N],idx;
//h[i]表示第i个结点存储的值 i从1开始 2*i是i的左子结点 2*i+1是i的右子结点
//idx表示堆存储元素的个数 也是最后一个结点下标
void down(int u)
{
    int t=u;//先存储当前结点下标
    //更新t为这爷仨中最小值的结点的下标
    if(u*2<=idx && h[u*2]<h[t]) t=u*2;
    if(u*2+1<=idx && h[u*2+1]<h[t]) t=u*2+1;
    if(t!=u)//不满足小根堆
    {
        swap(h[u],h[t]);//交换数值
        down(t);//往下试探 保证原来那个节点down到最下
    }
}
int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++) scanf("%d",&h[i]);
    idx=n;
    for(int i=n/2;i;i--) down(i);//构造小根堆

    while(m--)
    {
        //本题思路：先输出小根堆顶 再删除堆顶 重复m此 就是前m小
        printf("%d ",h[1]);
        h[1]=h[idx];
        idx--;
```

```
        h[1]=h[idx];
        idx--;
        down(1);
    }
    return 0;
}
```

839模拟堆

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+10;
int heap[N],idx;
int ph[N];//存储第k个插入的数在堆中的下标
int hp[N];//存储堆中各个下标是第k个插入的
            //ph[k]==j hp[j]==k;
void heap_swap(int a,int b)
{
    //a和b是结点在堆中的下标
    //一一映射关系 不用考虑交换顺序
    swap(ph[hp[a]],ph[hp[b]]);
    swap(hp[a], hp[b]);
    swap(heap[a], heap[b]);
}
void down(int u)
{
    int t=u;
    if(u*2<=idx && heap[u*2]<heap[t]) t=u*2;
    if(u*2+1<=idx && heap[u*2+1]<heap[t]) t=u*2+1;
    if(t!=u)
    {
        heap_swap(t,u);
        down(t);
    }
}

void up(int u)
{
    while(u>=2 && heap[u/2]>heap[u])
    {
        heap_swap(u/2,u);
        u/=2;
    }
}
```

```
    }
}

int main()
{
    int n,m=0; //m表示第m个插入的数
    scanf("%d",&n);
    while(n--)
    {
        string op;
        cin>>op;
        if(op=="I")
        {
            int x;
            scanf("%d",&x);
            idx++;
            m++;
            ph[m]=idx,hp[idx]=m;
            heap[idx]=x;
            up(idx);
        }
        else if(op=="PM")
        {
            printf("%d\n",heap[1]);
        }
        else if(op=="DM")
        {
            heap_swap(1,idx);
            idx--;//一定要先自减idx再up()或down()
            down(1);
        }
        else if(op=="D")
        {
            int k;
            scanf("%d",&k);
            k=ph[k];
            heap_swap(k,idx);
            idx--;
            up(k);
            down(k);
        }
        else
```

```
        }
        else
        {
            int k,x;
            scanf("%d%d",&k,&x);
            k=ph[k];
            heap[k]=x;
            down(k);
            up(k);
        }
    }
    return 0;
}
```

哈希表

2022年1月8日 16:34

散列表（Hash table，也叫哈希表），是根据关键码值(Key value)而直接进行访问的数据结构。
也就是说，它通过把关键码值映射到表中一个位置来访问记录，以加快查找的速度。
这个映射函数叫做散列函数，存放记录的数组叫做散列表。

拉链法

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+3;// 取大于1e5的第一个质数，取质数冲突的概率最小
int hasp[N],e[N],ne[N],idx;//开槽和单链表

void insert(int x)
{
    //求key 把x映射到 0-1e5 之间的数
    //c++中如果是负数 那他取模也是负的 所以 加N 再 %N 就一定是一个正数
    int k=(x % N + N) % N;
    //head后插入数值法
    //hasp[i]相当于head
    e[idx]=x;
    ne[idx]=hasp[k];
    hasp[k]=idx;
    idx++;
}

bool query(int x)
{
    int k=(x % N + N) % N;
    for(int i=hasp[k];i!=-1;i=ne[i])
        if(e[i]==x)
            return true;
    return false;
}

int main()
{
    int n;
    scanf("%d",&n);
    //整个hasp初始化为-1
    memset(hasp,-1,sizeof hasp);
```

```
    //整个hasp初始化为-1
    memset(hasp,-1,sizeof hasp);

    while(n--)
    {
        char op[2];
        int x;
        scanf("%s%d",op,&x);
        if(op[0]=='I') insert(x);
        else
        {
            if(query(x)) printf("Yes\n");
            else printf("No\n");
        }
    }
    return 0;
}
```

开放寻址法

```
#include<bits/stdc++.h>
using namespace std;
//开放寻址法 一般开 数据范围的2-3倍 大概率没有冲突
const int N=2e5+3;    //大于数据范围的第一个指数
const int null=0x3f3f3f3f; //规定空指针为 null
int hasp[N],n;

int find(int x)
{
    int k=(x%N+N)%N;
    //hasp[k]非空 并且值不是x k就向后走 直到hasp[k] 空或x 为止
    while(hasp[k] != null && hasp[k]!=x)
    {
        k++;
        if(k==N) k=0; //向后走 走到头了 也没找到坑返回k=0
    }
    return k;//如果这个位置是空的, 则返回的k是他应该存储的位置
}
int main()
{
    scanf("%d",&n);
    memset(hasp,0x3f,sizeof hasp);
    while(n--)
    {
        char op[2];
        int x;
```

```
    {
        char op[2];
        int x;
        scanf("%s%d",op,&x);
        if(op[0]=='i') hasp[find(x)]=x;
        else
        {
            if(hasp[find(x)]==null) printf("No\n");
            else printf("Yes\n");
        }
    }
    return 0;
}
```

字符串哈希 （字符串前缀哈希法）
应用：快速判断两个字符串是否相等

str="ABCDEF"
hasp[0]=0;
hasp[1]="A"的哈希值
hasp[2]="AB"的哈希值

hasp[i]=hasp[i-1] * P + str[i];

hasp[r]-hasp[l-1] * P ^ (r - l + 1);

(字符串哈希) O(n)+O(m)O(n)+O(m)
全称字符串前缀哈希法，把字符串变成一个p进制数字（哈希值），实现不同的字符串映射到不同的数字。
对形如 X1X2X3⋯Xn−1XnX1X2X3⋯Xn−1Xn 的字符串,采用字符的ascii 码乘上 P 的次方来计算哈希值。

映射公式 $(X_1 \times P^{n-1} + X_2 \times P^{n-2} + \cdots + X_{n-1} \times P^1 + X_n \times P^0) \bmod Q$

注意点：
1. 任意字符不可以映射成0，否则会出现不同的字符串都映射成0的情况，比如A,AA,AAA皆为0
2. 冲突问题：通过巧妙设置P (131 或 13331) , Q (2^64)的值，一般可以理解为不产生冲突。

问题是比较不同区间的子串是否相同，就转化为对应的哈希值是否相同。
求一个字符串的哈希值就相当于求前缀和，求一个字符串的子串哈希值就相当于求部分和。

前缀和公式 $h[i+1] = h[i] \times P + s[i]$ $i \in [0, n-1]$ h为前缀和数组，s为字符串数组
区间和公式 $h[l,r] = h[r] - h[l-1] \times P^{r-l+1}$

区间和公式的理解: ABCDE 与 ABC 的前三个字符值是一样，只差两位

区间和公式 $h[l,r] = h[r] - h[l-1] \times P^{r-l+1}$

区间和公式的理解: ABCDE 与 ABC 的前三个字符值是一样，只差两位，
乘上 P^2 把 ABC 变为 ABC00，再用 ABCDE - ABC00 得到 DE 的哈希值。

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+10,P=131;
typedef unsigned long long ULL;
ULL hasp[N],p[N];
// h[i]: 前i个字符的哈希值（把字符按P进制转化成十进制的key）
// p[i]: p的i次方
// P=131 Q=2^64 在99%情况下不会冲突
ULL get(int l,int r)
{
    return hasp[r]-hasp[l-1]*p[r-l+1];
    //先左移(r-l+1)个单位 再相减得区间key
}
int main()
{
    int n,m;
    char str[N];
    scanf("%d%d%s",&n,&m,str+1);

    p[0]=1;

    for(int i=1;i<=n;i++)
    {
        p[i]=p[i-1]*P; //求p的i次方
        hasp[i]=hasp[i-1]*P+str[i];//P进制的字符转化成十进制的key
    }

    while(m--)
    {
        int l1,r1,l2,r2;
        scanf("%d%d%d%d",&l1,&r1,&l2,&r2);
        //验证
        if(get(l1,r1)==get(l2,r2)) puts("Yes");
        else puts("No");
    }
    return 0;
}
```

memset

2022年1月9日 18:49

memset(hasp,-1,sizeof hasp) 相当于 for(int i=0;i<hasp.size();i++) hasp[i]=-1;

---数论---

2022年1月26日 12:56

要求：弄清楚原理

质数

2022年1月26日　12:56

质数：从2开始（包含）的整数，只包含1和本身这两个约数，就叫质数或素数。

合数：与质数相对。

0和1既不是质数也不是合数

（1）质数的判定——试除法

根据一个数的两个约数是成对出现的优化算法

比如12的约数对有：1与12、2与6、3与4。

就可以对朴素的试除法做出如下优化

时间复杂度：O(n)→O(sqrt(n))

```
for(int i=2; i < n ;i++)
↓
for(int i=2; i <= n/i ;i++)

//能避免在极限情况下的溢出等错误问题
```

```
bool is_prime(int n)
{
    if(n<2) return false;
    for(int i=2;i<=n/i;i++)
        if(n%i==0)
            return false;
    return true;
}
```

```
bool is_prime(int n)
{
    if(n<2) return false;
    for(int i=2;i<=n/i;i++)
        if(n%i==0)
            return false;
    return true;
}
```

（2）分解质因数——试除法

结论：

大于1的整数都能分解成质数的乘积

n中最多只包含一个大于sqrt(n)的质因子

代码：

```
void divide(int x)
{
    for(int i=2;i<=x/i;i++)
    {

        if(x%i==0)
        {
            int s=0;//指数
            while(x%i==0) x/=i,s++;
            cout<<i<<' '<<s<<endl;
        }
    }
    if(x>1) cout<<x<<' '<<1<<endl;
    cout<<endl;
}
```

时间复杂度：O(logn-sqrt(n))

（3）质数筛

1、朴素质数筛 O(nlogn)

```
const int N=1e6+10;
int prime[N],cnt;
bool up[N];//true为非质数

void get_primes(int n)
{
    for(int i=2;i<=n;i++)
    {
        //如果这个数是质数 用prime储存它 cnt计数
        if(up[i]==false) prime[cnt++]=i;
        //他的倍数都不是质数
        for(int j=i+i;j<=n;j+=i) up[j]=true;
    }
}
```

2、埃氏质数筛 O(n*loglogn)

```
const int N=1e6+10;
int prime[N],cnt;
bool up[N];//true为非质数

void get_primes(int n)
{
    for(int i=2;i<=n;i++)
    {
        //如果这个数是质数 用prime储存它 cnt计数
        if(up[i]==false)
        {
            prime[cnt++]=i;
            //筛质数的倍数
            for(int j=i+i;j<=n;j+=i) up[j]=true;
        }
    }
}
```

3、线性质数筛 O(n)

核心：合数n被最小质因子筛掉

```
const int N=1e6+10;
int prime[N],cnt;
bool up[N];//true为非质数

void get_primes(int n)
{
    for(int i=2;i<=n;i++)
    {
        if(up[i]==false) prime[cnt++]=i;
        for(int j=0;prime[j]<=n/i;j++)
        {
            up[prime[j]*i]=true;
            //prime[j]一定是i的最小质因子
            //prime[j]一定是prime[j]*i的最小质因子
            if(i%prime[j]==0) break;
        }
    }
}
```

约数

2022年1月26日 12:56

背景知识：

a除b：b/a

a除以b：a/b

a能整除b：b/a是整数

a能被b整除：a/b是整数 b为约数

定义：

若整数n除以d的余数为0，即d能整除n，则称d是n的倍数，n是d的倍数，记作d ｜ n。

PS：

int范围内的某个数的最多约数为1500个左右

题目：

1、试除法求约数 O(n*sqrt(a))

```cpp
#include<bits/stdc++.h>
using namespace std;
vector<int> get_divisors(int n)
{
    vector<int> res;
    for(int i=1;i<=n/i;i++)
        if(n%i==0)
        {
            res.push_back(i);
            if(i!=n/i) res.push_back(n/i);
        }
    sort(res.begin(),res.end());
    return res;
}

int main()
{
    int n;
    cin>>n;
    while(n--)
    {
        int x;
        cin>>x;
        vector<int> res=get_divisors(x);
        for(int i=0;i<res.size();i++) cout<<res[i]<<' ';
        cout<<endl;
    }
    return 0;
}
```

2、约数个数

```cpp
#include<bits/stdc++.h>
using namespace std;
const int mod=1e9+7;
unordered_map<int,int> primes;
void divide(int x)
{
    for(int i=2;i<=x/i;i++)
        while(x%i==0) x/=i,primes[i]++;//约数i的数量++;
    if(x>1) primes[x]++;
}
int main()
{
    int n;
    cin>>n;
    while(n--)
    {
        int x;
        cin>>x;
        divide(x);
    }
    long long res=1;
    for (auto p : primes) res = res * (p.second + 1) % mod;
    cout<<res;
    return 0;
}
```

3、约数之和

```cpp
#include<bits/stdc++.h>
using namespace std;
const int mod=1e9+7;
unordered_map<int,int> primes;
void divide(int x)
{
    for(int i=2;i<=x/i;i++)
        while(x%i==0) x/=i,primes[i]++;
    if(x>1) primes[x]++;
}
int main()
{
    int n;
    cin>>n;
    while(n--)
    {
        int x;
        cin>>x;
        divide(x);
    }
    long long res=1;
    for (auto prime : primes)
    {
        int p=prime.first,c=prime.second;
        long long t=1;
        while(c--) t=(p*t+1)%mod;//求p的0次幂到p的c次幂之和
        res=res*t%mod;
    }
    cout<<res;
    return 0;
}
```

```
如果 N = p1^c1 * p2^c2 * ... *pk^ck
约数个数: (c1 + 1) * (c2 + 1) * ... * (ck + 1)
约数之和: (p1^0 + p1^1 + ... + p1^c1) * ... * (pk^0 + pk^1 + ... + pk^ck)
```

4、最大公约数——欧几里得算法——辗转相除法

```cpp
#include<bits/stdc++.h>
using namespace std;
int gcd(int a,int b)
{
	return b?gcd(b,a%b):a;
}
int main()
{
	int n;
	cin>>n;
	while(n--)
	{
		int a,b;
		cin>>a>>b;
		cout<<gcd(a,b)<<endl;
	}
	return 0;
}
```

核心：(a,b)的最大公约数==(b,a%b)的最大公约数

欧拉函数

2022年1月26日　　12:56

定义：1-n中与n互为质数的数的个数

互质数：公因数只有1的非零自然数

互质数的性质：

①如果a与n互为质数，那么（a^phi[n]）%n　同于1

②极性函数：phi[a*b]=phi[a]*phi[b]

> 1~N中与N互质的数的个数被称为欧拉函数，记为$\phi(N)$。
> 若在算数基本定理中，$N = p_1^{\alpha 1} p_2^{\alpha 2} \cdots p_k^{\alpha k}$，则：
> $\phi(N) = N * \frac{p_1-1}{p_1} * \frac{p_2-1}{p_2} * \ldots * \frac{p_k-1}{p_k}$

1、求n的欧拉函数

```
int phi(int x)
{
    int res=x;
    for(int i=2;i<=x/i;i++)
    {
        if(x%i==0)
        {
            res=res/i*(i-1);
            while(x%i==0) x/=i;
        }
    }
    if(x>1) res=res/x*(x-1);
    return res;
}
```

2、线性筛求1-n欧拉函数之和

筛法求欧拉函数（适用于1~n求欧拉的情况）

① 先写出线性筛算法的模板。

② 考虑特殊情况：若该数是质数p的话，那么该数的欧拉函数就是$p-1$。

③ 每个数的欧拉函数与质因子的次数无关。例如：$N=2^{100}\times 3^{100}$，但是N的欧拉函数还是$N\times(1-\frac{1}{2})\times(1-\frac{1}{3})$

④ 若$i \bmod primes[j] == 0$，由于$primes[j]$是i的一个质因子，并且在计算i的欧拉函数的时候已经计算了$primes[j]$出现的情况$(1-\frac{1}{primes[j]})$，所以$\varphi(primes[j]\times i)=primes[j]\times\varphi(i)$。

⑤ 若$i \bmod primes[j] != 0$，$primes[j]$就一定是$primes[j]\times i$的最小质因子，而且$primes[j]$不包含在i的质因子当中。由于$\varphi(i)=i\times(1-\frac{1}{p_1})\times(1-\frac{1}{p_2})\times\ldots\times(1-\frac{1}{p_k})$，所以$\varphi(primes[j]\times i)=primes[j]\times i\times(1-\frac{1}{p_1})\times(1-\frac{1}{p_2})\times\ldots\times(1-\frac{1}{p_k})\times(1-\frac{1}{primes[j]})$。最终就可以得到$\varphi(primes[j]\times i)=primes[j]\times\varphi(i)\times(1-\frac{1}{primes[j]})=\varphi(i)\times(primes[j]-1)$。

⑥ 代码如下所示：

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=1e6+10;
int primes[N],cnt;
int phi[N];// phi[i]==i的欧拉函数值
bool st[N];
void get_eulers(int n)
{
    phi[1]=1;
    for(int i=2;i<=n;i++)
    {
        //如果i是质数的话 i的欧拉函数=i-1
        if(!st[i])
        {
            primes[cnt++]=i;
            phi[i]=i-1;
        }
        for(int j=0;primes[j]<=n/i;j++)
        {
            //合数用效率极高的线性筛筛掉
            st[primes[j]*i]=true;
            //因为p[j]是i的质因子
            //进而i*prime[j]的欧拉函数==phi[i]*primes[j]
            if(i%primes[j]==0)
            {
                phi[primes[j]*i]=phi[i]*primes[j];
                break;
            }
            phi[primes[j]*i]=phi[i]*(primes[j]-1);
        }
    }
}
int main()
{
    int n;
    cin>>n;
    get_eulers(n);
    long long res=0;
    for(int i=1;i<=n;i++) res+=phi[i];
    cout<<res;
    return 0;
}
```

快速幂

2022年1月26日 12:56

快速幂

定义：快速地求出底数的n次幂

时间复杂度：O(log2N)

求 a 的 b 次方对 p 取模的值，其中 $1 \le a,b,p \le 10^9$。

相关题目：POJ1995 Raising Modulo Numbers

根据数学常识，每一个正整数可以唯一表示为若干指数不重复的 2 的次幂的和。也就是说，如果 b 在二进制表示下有 k 位，其中第 $i(0 \le i < k)$ 位的数字是 c_i，那么：

$$b = c_{k-1}2^{k-1} + c_{k-2}2^{k-2} + \cdots + c_0 2^0$$

于是：

$$a^b = a^{c_{k-1}*2^{k-1}} * a^{c_{k-2}*2^{k-2}} * \cdots * a^{c_0*2^0}$$

因为 $k = \lceil \log_2(b+1) \rceil$（其中⌈⌉表示上取整），所以上式乘积项的数量不多于 $\lceil \log_2(b+1) \rceil$ 个。又因为：

$$a^{2^i} = \left(a^{2^{i-1}}\right)^2$$

所以我们很容易通过 k 次递推求出每个乘积项，当 $c_i = 1$ 时，把该乘积项累积到答案中。$b \,\&\, 1$ 运算可以取出 b 在二进制表示下的最低位，而 $b >> 1$ 运算可以舍去最低位，在递推的过程中将二者结合，就可以遍历 b 在二进制表示下的所有数位 c_i。整个算法的时间复杂度[①]为 $O(\log_2 b)$。

```
int power(int a, int b, int p) { // calculate (a ^ b) mod p
    int ans = 1 % p;
    for (; b; b >>= 1) {
        if (b & 1) ans = (long long)ans * a % p;
        a = (long long)a * a % p;
    }
    return ans;
}
```

在上面的代码片段中，我们通过“右移（>>）”“与（&）”运算的结合，遍历了 b 的二进制表示下的每一位。在循环到第 i 次时（从 0 开始计数），变量 a 中存储的是 a^{2^i}，若 b 该位为 1，则把此时的变量 b 累积到答案 ans 中。

值得提醒的是，在 C++语言中，两个数值执行算术运算时，以**参与运算**的最高数值类型为基准，与保存结果的变量类型无关。换言之，虽然两个 32 位整数的乘积可能超过 int 类型的表示范围，但是 CPU 只会用 1 个 32 位寄存器保存结果，造成我们常说的越界现象。因此，我们必须把**其中一个数强制转换成 64 位整数类型** long long 参与运算，从而得到正确的结果。最终对 p 取模以后，执行赋值操作时，该结果会被隐式转换成 int 存回 ans 中。

于是一个问题就出现了。因为 C++ 内置的最高整数类型是 64 位，若运算 $a * b \bmod p$ 中的三个变量 a, b, p 都在 10^{18} 级别，则不存在一个可供强制转换的 128 位整数类型，我们需要一些特殊的处理办法。

$4^5 \bmod 10$

$4^{2^0} \equiv 4 \pmod{10}$

平方

$4^{2^1} \equiv 6 \pmod{10}$

平方

$4^{2^2} \equiv 6 \pmod{10}$

$4^5 = 4^{(101)_2}$

$= 4^{2^0+2^2}$

$= 4^{2^0} \cdot 4^{2^2}$

$= 4 \times 6$

$\equiv 4 \pmod{10}$

```cpp
#include<bits/stdc++.h>
using namespace std;

int quick_power(int a,int b,int p)
{
    int res=1;
    while(b)
    {
        //如果二进制的个位为1
        if(b&1) res=(long long)res*a%p;
        b>>=1;//抹掉最后一位
        a=(long long)a*a%p;
    }
    return res;
}
int main()
{
    int n;
    cin>>n;
    while(n--)
    {
        int a,b,p;
        cin>>a>>b>>p;
        cout<<quick_power(a,b,p)<<endl;
    }
    return 0;
}
```

快速幂求逆元

$a / n \equiv a * x \pmod p$

两边同乘b可得 $a \equiv a * n * x \pmod p$

即 $1 \equiv n * x \pmod p$

同 $n * x \equiv 1 \pmod p$①

由费马小定理可知，当p为质数时

$n ^ {(p - 1)} \equiv 1 \pmod p$

拆一个b出来可得 $n * n ^ {(p - 2)} \equiv 1 \pmod p$②

故当m为质数时，n % p的乘法逆元 $x = n ^ {(p - 2)}$

扩展欧几里得算法

2022年1月30日　15:28

bezout定理（裴蜀定理）：

定义:

对于任意整数a,b，存在一对整数x,y，满足ax+by=gcd(a,b)。

证明:

若b=0时，显然有一对整数 x=1,y=0，使得a*1+0*0=gcd(a,0);

若b>0时，则gcd(a,b)=gcd(b,a%b)。

假设存在一对整数x,y,满足b*x+a%b*y=gcd(a,a%b),

因为b*x+(a%b)*y=b*x+(a-a/b*b)*y=a*y+b*(x-(a/b)*y)

x'=y,y'=x-(a/b)*y;　//可以翻转传入 减少不必要的swap

代码:

```
int exgcd(int a,int b,int &x,int &y)
{
    if(b==0)
    {
        x=1,y=0;
        return a;
    }
    int d=exgcd(b,a%b,y,x);
    y-=(a/b)*x;
    return d;
}
```

线性同余方程

定义:

给定整数a,b,m，求一个整数x满足 a*x同于b(mod m),或者给出无解。

因为未知数的指数为1，所以我们称之为一次同余方程，也称线性同余方程。

过程:

a*x同于b(mod m) ⟷ a*x-b是m的倍数，不妨设为-y倍。

于是，该方程可以改写为：a*x+m*y=b;①

根据bezout定理及其证明过程，只要①式扩大b/d倍，就可以变回原式。

线性同余方程有解当且仅当(d)gcd(a,m)|b。

求解：

先用扩展欧几里得算法求出一组整数x1,y1，满足a*x1+m*y1=gcd(a,m)=d。

然后 x = x1*(b/d)就是线性同余方程的一个解。

通解：所有模m/d 与 x同余的整数。

代码：

```
int main()
{
    int n;
    cin>>n;
    while(n--)
    {
        int a,b,m,x,y;
        cin>>a>>b>>m;
        int d=exgcd(a,m,x,y);
        if(b%d) cout<<"impossible"<<endl;
        else cout<<(long long) x*(b/d)<<endl;
    }
    return 0;
}
```

中国剩余定理

2022年1月30日 17:40

定义：

给定2n个整数a1,a2,...,an和m1,m2,m3, ... ,mn，求一个最小非负整数x，满足任意的i∈[1,n]，x=mi（mod ai）。

高斯消元

2022年2月2日　18:04

定义：求解1个包含n个方程n个未知数的多元线性方程组

时间复杂度：O(n^3)

解的情况：

①无解-出现0=!0的等式

②无穷多解-出现多个0=0的方程等式

③唯一解-完美阶梯型

初等行列变换（操作、核心、解不变）：

①把一个等式两边同乘一个非零的数

②交换某两个等式

③把某个等式的若干倍加到另一个等式上去

过程：

1、枚举每一列（依次固定行）

①找到绝对值最大的一行

②把这一行换到最顶层

③将该行第一个系数变为1（等式两边同除第一个系数）

④将当前列除了第一行的系数都消成0

（通过初等行列变换使方程组变梯形矩阵的形式）

a11	a12	a13	a14	...	a1n	=	b1
0	a22	a23	a24	...	a2n	=	b2
0	0	a33	a34	...	a3n	=	b3
0	0	0	a44	...	a4n	=	b4
0	0	0	0	...	...	...	...
0	0	0	0	0	ann	=	bn

2、求解未知数

高斯消元解线性方程组：

```
#include<bits/stdc++.h>
using namespace std;
const int N=110;
```

```
using namespace std;
const int N=110;
const double eps=1e-6;
int n;
double a[N][N];//存放系数
int gauss()
{
    int c,r;//c为列 r为行
    //枚举每一列
    for(c=0,r=0;c<n;c++)
    {
        //1、t存储c列绝对值最大的数 所在的行
        int t=r;
        for(int i=r;i<n;i++)
            if(fabs(a[i][c])>fabs(a[t][c]))
                t=i;
        //如果这一列上所有的数都是0的话 跳过看下一列
        if(fabs(a[t][c])<eps) continue;
        //2、让第t行与未固定的行（第r行）交换 （每走完一列固定一行）
        for(int i=c;i<=n;i++) swap(a[t][i],a[r][i]);
        //3、把未固定行的每个系数除第一个系数
        for(int i=n;i>=c;i--) a[r][i]/=a[r][c];
        //4、把本次固定行下面所有行的c列的系数化为0
        for(int i=r+1;i<n;i++)
            if(fabs(a[i][c])>eps)
                for(int j=n;j>=c;j--)
                    a[i][j]-=a[r][j]*a[i][c];
        r++;//固定下一行
    }

    if(r<n)
    {
        for(int i=r;i<n;i++)
        {
            if(fabs(a[i][n])>eps) return 2;//无解
        }
        return 1;//无穷多解
    }
    //原方程有n个解 从下到上求解
    for(int i=n-1;i>=0;i--)
    {
```

```
    for(int i=n-1;i>=0;i--)
    {
        for(int j=i+1;j<n;j++)
        {
            a[i][n]-=a[j][n]*a[i][j];
        }
    }
    return 0;//完美解
}
int main()
{
    cin>>n;

    for(int i=0;i<n;i++)
        for(int j=0;j<n+1;j++)
            cin>>a[i][j];

    int t=gauss();

    if(t==1){
        cout<<"Infinite group solutions"<<endl;
    }else if(t==2){
        cout<<"No solution"<<endl;
    }else{
        for(int i=0;i<n;i++)
        {
            //去掉-0.00情况
            if(fabs(a[i][n])<eps) a[i][n]=0;
            printf("%.2f/n",a[i][n]);
        }
    }
    return 0;
}
```

高斯消元解异或线性方程组:

求组合数

2022年2月4日 23:12

求组合数 I

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=2010,mod=1e9+7;
int c[N][N];
//从i里面选j个 化为 从i里拿走一个球
//j里包含那个球：从剩下的里面挑j-1个 c[i-1][j-1]
//j里不包含那个球：从剩下的里面挑j个 c[i-1][j]
//通过递推求出所有C值 再进行查表 优化时间复杂度
void init()
{
    for(int i=0;i<N;i++)
        for(int j=0;j<=i;j++)
            if(!j) c[i][j]=1;
            else c[i][j]=(c[i-1][j]+c[i-1][j-1])%mod;
}
int main()
{
    init();
    int n;
    cin>>n;
    while(n--)
    {
        int a,b;
        cin>>a>>b;
        cout<<c[a][b]<<endl;
    }
    return 0;
}
```

求组合数II

```cpp
//C[a][b] == a! ÷ b!+(a-b)!
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
const int N=100010,mod=1e9+7;
int fact[N],infact[N];
//快速幂求逆元
```

```
//C[a][b] == a! ÷ b!+(a-b)!
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
const int N=100010,mod=1e9+7;
int fact[N],infact[N];
//快速幂求逆元
int quick_power(int a,int b,int p)
{
    int res=1;
    while(b)
    {
        if(b&1) res=(LL)res*a%p;
        a=(LL)a*a%p;
        b>>=1;
    }
    return res;
}

int main()
{
    //0的阶乘是1
    fact[0]=infact[0]=1;
    //预处理阶乘
    for(int i=1;i<N;i++)
    {
        fact[i]=(LL) fact[i-1] * i % mod;
        infact[i]=(LL) infact[i-1] * quick_power(i,mod-2,mod) % mod;
    }
    int n;
    cin>>n;
    while(n--)
    {
        int a,b;
        cin>>a>>b;
        cout<<(LL)fact[a]*infact[b]%mod*infact[a-b]%mod<<endl;
    }
    return 0;
```

求组合数Ⅲ

卢卡斯定理：C(n,m)%p=C(n/p,m/p)*C(n%p,m%p)%p

```
#include<bits/stdc++.h>
using namespace std;
```

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
int p;
int quick_power(int a,int b,int p)
{
    int res=1;
    while(b)
    {
        if(b&1) res=(LL)res*a%p;
        a=(LL) a*a%p;
        b>>=1;
    }
    return res;
}
int C(int a,int b,int p)
{
    if(a<b) return 0;
    int res=1;
    for(int i=1,j=a;i<=b;i++,j--)
    {
        res=(LL) res*j%p;//分子
        res=(LL) res*quick_power(i,p-2,p)%p;//逆元分母
    }
    return res;
}
int lucas(LL a,LL b,int p)
{
    if(a<p && b<p) return C(a,b,p);
    return (LL) C(a%p,b%p,p)*lucas(a/p,b/p,p)%p;
}
int main()
{
    int n;
    cin>>n;
    while(n--)
    {
        LL a,b;
        cin>>a>>b>>p;
        cout<<lucas(a,b,p)<<endl;
    }
    return 0;
}
```

求组合数IV

过程：

1、筛出2-a中的所有质数

2、分解质因数

把C[a][b]=a!÷b!*(a-b)!转化为质数次幂相乘的形式

求出每个质数在式子中出现的次数

3、利用高精度乘把质数幂乘出来

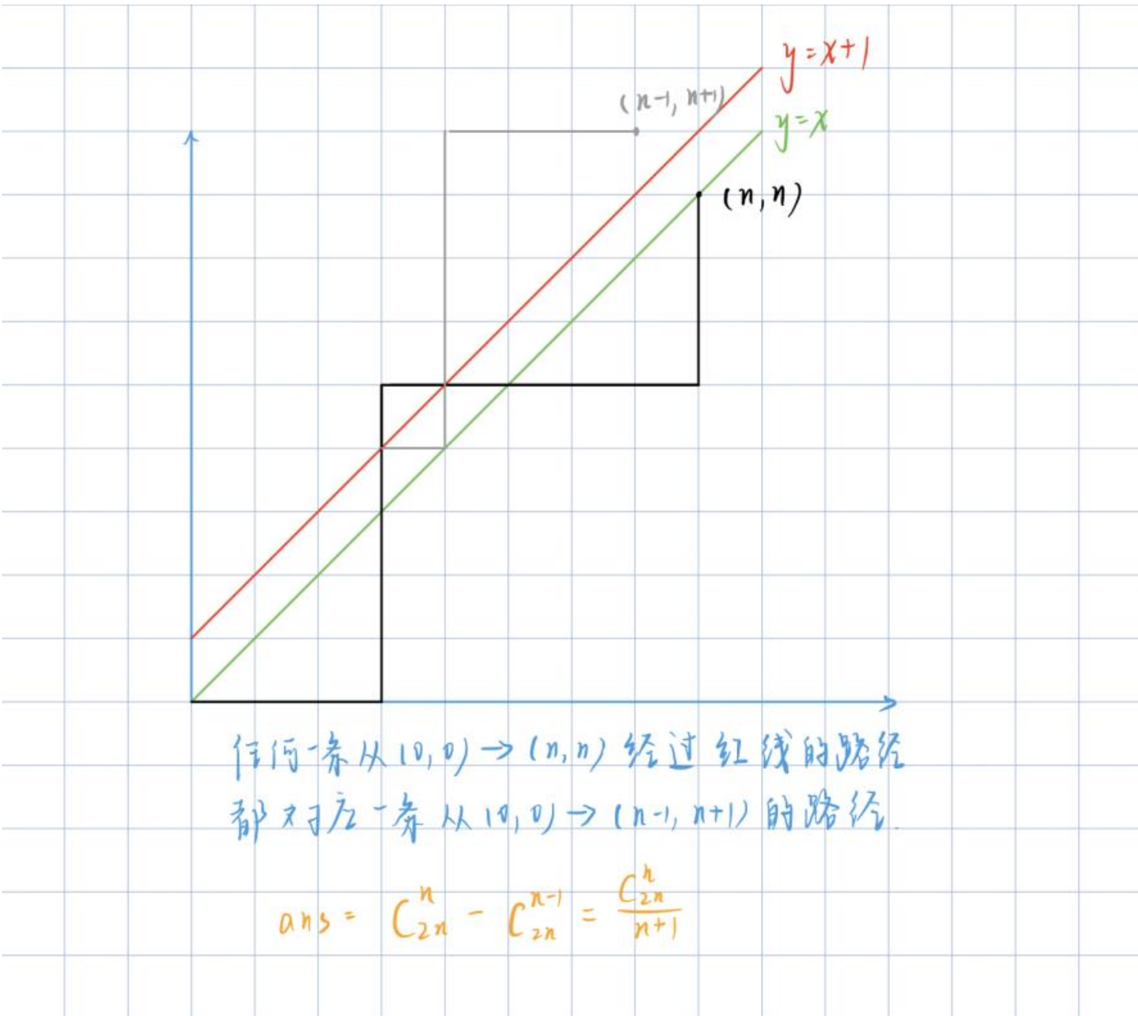
代码：

```
#include<bits/stdc++.h>
using namespace std;
const int N=5010;
int primes[N],cnt,sum[N];
bool up[N];
void get_primes(int n)
{
    for(int i=2;i<=n;i++)
    {
        if(!up[i]) primes[cnt++]=i;
        for(int j=0;primes[j]<=n/i;j++)
        {
            up[primes[j]*i]=true;
            if(i%primes[j]==0) break;
        }
    }
}
int get(int n,int p)
{
    int res=0;
    while(n)
    {
        res+=n/p;
        n/=p;
    }
    return res;
}
vector<int> mul(vector<int> &A,int b)
{
    vector<int> C;
    int t=0;
```

```
    vector<int> C;
    int t=0;
    for(int i=0;i<A.size();i++)
    {
        t+=A[i]*b;
        C.push_back(t%10);
        t/=10;
    }
    while(t)
    {
        C.push_back(t%10);
        t/=10;
    }
    return C;
}
int main()
{
    int a,b;
    cin>>a>>b;
    //筛出2-a中所有的质数
    get_primes(a);
    //求出每个质数在C[a][b]中出现的次数
    //因为C[a][b]已转化为质数幂相乘的形式
    for(int i=0;i<cnt;i++)
    {
        int p=primes[i];
        sum[i]=get(a,p)-get(b,p)-get(a-b,p);
    }
    //高精乘把质数幂乘出来
    vector<int> res;
    res.push_back(1);

    for(int i=0;i<cnt;i++)
        for(int j=0;j<sum[i];j++)
            res=mul(res,primes[i]);

    for(int i=res.size()-1;i>=0;i--) cout<<res[i];

    return 0;
}
```

卡特兰数

2022年2月7日 15:58



解法：

将 01 序列置于坐标系中，起点定于原点。若 0 表示向右走，1 表示向上走，那么任何前缀中 0 的个数不少于 1 的个数就转化为，路径上的任意一点，横坐标大于等于纵坐标。题目所求即为这样的合法路径数量。

由图可知，任何一条不合法的路径（如黑色路径），都对应一条从 (0,0)(0,0) 走到 (n−1,n+1)(n−1,n+1) 的一条路径（如灰色路径）。而任何一条 (0,0)(0,0) 走到 (n−1,n+1)(n−1,n+1) 的路径，也对应了一条从 (0,0)(0,0) 走到 (n,n)(n,n) 的不合法路径。

公式：

res=C[2n][n] - C[2n][n-1] = C[2n][n] /(n+1) = (n~2n)! / n! /(n+1)

代码：

```cpp
#include<bits/stdc++.h>
using namespace std;
const int mod=1e9+7;
//因为mod是质数 所以我们可以根据费马小定理求逆元
int quick_power(int a,int b,int p)
{
    int res=1;
    while(b)
    {
        if(1&b) res=(long long)res*a%p;
        b>>=1;
        a=(long long)a*a%p;
    }
    return res;
}
int main()
{
    int n;
    cin>>n;
    int a=2*n,b=n;
    int res=1;
    for(int i=a;i>a-b;i--) res=(long long)res*i%mod;
    for(int i=1;i<=b;i++) res=(long long) res*quick_power(i,mod-2,mod)%mod;
    res=(long long) res*quick_power(n+1,mod-2,mod)%mod;
    cout<<res;
    return 0;
}
```

容斥原理

2022年2月6日 0:22

从n个数当中选任意多（0-n）个数的方案数

左边：从n中选0、1、2...n个数

右边：队中每个数都有选或不选的情况 相乘 得2^n

c[n][0]+c[n][1]+c[n][2]+...+c[n][n]==2^n

时间复杂度：2^n-1

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
const int N=20;
int n,m;
int p[N];//存放质数
int main()
{
    cin>>n>>m;
    for(int i=0;i<m;i++) cin>>p[i];
    //根据容斥原理，此题只需划分出m个集合
    //每个集合里面是对应质数及其小于等于n的倍数(求大小)
    //所有满足题意的数的个数 = 各个集合的并集
    int res=0;
    //枚举从m个集合选(1~m)集合的状态
    //利用二进制位运算 01表示状态
    for(int i=1;i<1<<m;i++)
    {
        int t=1;//选中集合对应质数的乘积
        //由于p皆为质数 质数的乘积就是满足题意的数的最小公倍数
        int s=0;//选中集合的数量
        //枚举当前状态的每一个集合(质数)
        for(int j=0;j<m;j++)
        {
            //某个集合(质数)被选上
            if(i>>j & 1)
            {
                //过大 跳出
                if((LL)t*p[j] > n)
```

```
            {
                //过大 跳出
                if((LL)t*p[j] > n)
                {
                    t=-1;
                    break;
                }
                s++;//被选上++
                t*=p[j];//质数乘积
            }
        }
        if(t==-1) continue;
        //选中的集合的质数 奇数个系数为1 偶数-1
        //n/t为当前状态集合（交集）的大小
        if(s&1) res+=n/t;
        else res-=n/t;
    }
    cout<<res;
    return 0;
}
```

简单博弈论

2022年2月7日　　17:09

NIM游戏

若一个游戏满足：

1、由两名玩家交替行动

2、在游戏进行的任意时刻，可以执行的合法行动与轮到哪位玩家无关

3、不能行动的玩家判负

则称该游戏为一个公平组合游戏。

尼姆游戏（NIM）属于公平组合游戏，但常见的棋类游戏，比如围棋就不是公平组合游戏，因为围棋交战双方分别只能落黑子和白子，胜负判定也比较负责，不满足条件2和3。

题目描述

给定n堆石子，两位玩家轮流操作，每次操作可以从任意一堆石子中拿走任意数量的石子（可以拿完，但不能不拿），最后无法进行操作的人视为失败。

问如果两人都采用最优策略，先手是否必胜。

例如：有两堆石子，第一堆有2个，第二堆有3个，先手必胜。

操作步骤：

1. 先手从第二堆拿走1个，此时第一堆和第二堆数目相同
2. 无论后手怎么拿，先手都在另外一堆石子中取走相同数量的石子即可。

必胜状态和必败状态

在解决这个问题之前，先来了解两个名词：

1、必胜状态，先手进行某一个操作，留给后手是一个必败状态时，对于先手来说是一个必胜状态。即先手可以走到某一个必败状态。

2、必败状态，先手无论如何操作，留给后手都是一个必胜状态时，对于先手来说是一个必败状态。即先手走不到任何一个必败状态。

结论

假设nn堆石子，石子数目分别是a1,a2,…,ana1,a2,…,an，

如果a1⊕a2⊕…⊕an≠0a1⊕a2⊕…⊕an≠0，先手必胜；

否则先手必败。

证明

操作到最后时，每堆石子数都是0，0⊕0⊕…0=0

在操作过程中，如果 a1⊕a2⊕…⊕an=x≠0a1⊕a2⊕…⊕an=x≠0。那么玩家必然可以通过拿走某一堆若干个石子将异或结果变为0。

证明：不妨设x的二进制表示中最高一位1在第k位，

那么在a1,a2,…,ana1,a2,…,an中，必然有一个数ai，它的第k为时1，
且ai⊕x<ai，那么从第i堆石子中拿走(ai−ai⊕x)个石子，第i堆石子还剩ai⊕x，
此时a1⊕a2⊕…⊕ai⊕x⊕…⊕an=x⊕x=0

在操作过程中，如果 a1⊕a2⊕…⊕an=0，那么无论玩家怎么拿，必然会导致最终异或结果不为0。

反证法：假设玩家从第i堆石子拿走若干个，结果仍是0。
不妨设还剩下a′个，因为不能不拿，所以0≤a′<ai，且a1⊕a2⊕…⊕a′⊕…⊕an=0。
那么(a1⊕a2⊕…⊕ai⊕…an)⊕(a1⊕a2⊕…⊕a′⊕…⊕an)=ai⊕a′=0，则 ai=a′，与假设0≤a′<ai矛盾。

基于上述三个证明:

1. 如果先手面对的局面是a1⊕a2⊕…⊕an≠0，那么先手总可以通过拿走某一堆若干个石子，将局面变成a1⊕a2⊕…⊕an=0。如此重复，最后一定是后手面临最终没有石子可拿的状态。先手必胜。
2. 如果先手面对的局面是a1⊕a2⊕…⊕an=0，那么无论先手怎么拿，都会将局面变成a1⊕a2⊕…⊕an≠0，那么后手总可以通过拿走某一堆若干个石子，将局面变成a1⊕a2⊕…⊕an=0。如此重复，最后一定是先手面临最终没有石子可拿的状态。先手必败。

代码:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cin>>n;
    int res=0;
    while(n--)
    {
        int x;
        cin>>x;
        res^=x;
    }
    if(res) cout<<"Yes";
    else cout<<"No";
    return 0;
}
```

Mex运算

设S是一个非负整数集合，求出不属于集合S的最小非负整数的运算。

SG函数

点x的SG值 等于 x能到达的点(相邻的点)的SG值组成的集合的Mex值
SG(x) == Mex{SG(y1),SG(y2) ... SG(yk)}

```
SG(终点) == 0;
```