

ACM模板

by 鱼竿钓鱼干

E-mail:851892190@qq.com

参考: acwing板子, 洛谷题解, 各类博客

版本2020/12

ACM模板

- 自查纠错/数据测试/复杂度反推

 - 自查纠错

 - 数据测试

 - 复杂度反推

- 数学

 - 质数

 - 试除法

 - 埃氏筛

 - 线性筛

 - 质因数

 - 分解质因数

 - 试除法求约数

 - 约数个数(多个数相乘的)

 - 约数个数和

 - 约数和

 - 欧拉函数

 - 筛法求欧拉函数 (1~n, 欧拉函数之和)

- GCD/LCM

 - GCD

 - LCM

- 组合数

 - 递推

 - 递归

- 快速幂

 - 裸的

 - 快速幂+取模

- 阶乘

 - 阶乘位数

- 回文

 - 判断

- 斐波那契数列

递归

递推

数学公式杂烩

公式猜测($rp+++++$)

待定系数法

几何

平面分割

任意多边形面积

数论

gcd/lcm

互质

质数/质因数/唯一分解定理

常识

进制与位运算

进制转换

10进制转K进制

K进制转10进制

Tip

二进制枚举

枚举子集 $O(2^{|S|})$, $|S|$ 表示集合中元素个数

枚举子集的子集 $O(3^n)$

位运算

求二进制中1的个数

暴力 $O(\log N)$

位运算 $O(1)$

$\log_2(x)$ 取整

求 $\log_2(x)$ (暴力) $O(\log x)$

求 $\log_2(x)$ (位运算二分展开) $O(\log \log x)$

求 $\log_2(x)$ (二分) $O(\log \log x)$

预处理 $\log_2(x)$ 1~n中所有数 $O(n)$

龟速加($a*b\%p$)

STL:bitset

高精

A+B

A-B

A*b

A/b

大数阶乘 (vector太慢了, 用数组)

其他处理

去前导0

多组输入初始化

排序

快排

归并

排序

求逆序对

STL

sort+结构体+cmp

输入输出

快读

C++关闭同步

数据结构

STL

栈

后缀表达式

单调栈

队列

普通队列

循环队列

单调队列(滑动窗口)

Trie树

存储查找字符串集合

并查集

普通并查集

维护距离(向量本质，有向图)

维护大小

扩展域

链表

普通链表

双链表

哈希表

堆

查找

二分查找

整数二分

浮点数二分

STL

搜索与图论

图

图的存储

邻接表

邻接矩阵

建图小技巧

DFS

有多少个连通块(洪泛)

DFS遍历图(树的重心)

列出所有解

全排列

STL

组合输出

BFS

最短步数（边的权值均为1，STL写法）

BFS+路径保存

BFS遍历图(边权1最短路，手动模拟队列写法)

拓扑序列（有向无环图AOV）

连通块里多少块

前缀和/差分

一维前缀和

二维前缀和

一维差分

二维差分

注意事项

字符串

KMP

滑动窗口

DP

DP思考方式

状态表示

集合

属性

状态计算

划分

计算过程

背包

01背包

二维

一维

完全背包

二维

二维优化

一维优化

多重背包

暴力朴素

二进制优化

分组背包

背包方案数

二维

一维

线性DP

数字三角

最长上升子序列 (LIS)

最长公共子序列 (LCS)

区间DP

记忆化搜索

自查纠错/数据测试/复杂度反推

自查纠错

- ☐ 循环
 - ☐ 等号和赋值符号区分
 - ☐ for循环三个分号不是同一变量
 - ☐ 循环内部操作和循环变量不对应
 - ☐ break放错循环层
 - ☐ while没有考虑直接为0的情况
 - ☐ 质数相关的背板子不要忘记是</<=
- ☐ 数组
 - ☐ 数组过大/过小（特别是用板子的时候）
 - ☐ 访问开辟内存（特别是数组里面有运算的时候）
- ☐ 输入
 - ☐ scanf没&
 - ☐ 换行符问题
 - ☐ 忘记多组输入
 - ☐ 忘记输入结束符
 - ☐ 尽量不要以换行符结束输入（可能运行超时）
- ☐ 输出
 - ☐ 浮点数用整型输出
 - ☐ 最后一个数据末尾多空格（查不到的时候直接for循环遍历，检测空格和换行输出yes）
 - ☐ 区分隔一行和换行
- ☐ 运算过程
 - ☐ 注意隐式转换
 - ☐ 取模运算不能%0
 - ☐ 注意数据溢出（要对式子进行优化）
 - ☐ int类型向加相乘
 - ☐ 递推递归dp
 - ☐ 把所有int改成long long
 - ☐ 变量初始化
- ☐ STL

☐ map查找之前先判key是否存在

数据测试

☐ 样例数据

☐ 小范围手动模拟+朴素暴力算法对拍

☐ 极限数据

☐ 极限大

☐ 极限小

☐ 随机数据

☐ 重复

☐ 乱序

☐ 边界分类

☐ 多情况分类组合

☐ 特殊数据

☐ 根据题意的

☐ 零（除法，取模）

☐ 全0

☐ 部分0

☐ 字符串

☐ 开头空格

☐ 结尾空格

☐ 中间空格

☐ 输入输出的前导0

☐ 是否区分大小写

☐ 几何

☐ 凹多边形

☐ 最后一次数据

☐ 存储

☐ 计算

☐ 多组输入

☐ 数据保留

☐ 数据清空

☐ 所有数组

☐ 所有变量

☐ 所有stl容器

- ☐ 特殊变量
- ☐ flag
- ☐ sum
- ☐ max/min
- ☐ 累乘

复杂度反推

1. $n \leq 30$, 指数级别
dfs+剪枝, 状压DP
 2. $n \leq 100$ $O(n^3)$
floyd, dp, 高斯消元
 3. $n \leq 1000$ $O(n^2)$, $O(n^2 \log n)$
dp, 二分, 朴素Dijkstra, 朴素Prim, Bellman-Ford
 4. $n \leq 10000$ $O(n \cdot \sqrt{n})$
块状链表, 分块, 莫队
 5. $n \leq 1e5$ $O(n \log n)$
sort, 线段树, 树状数组, set/map, heap, 拓扑排序, dijkstra+heap, prim+heap, spfa, 求凸包、求半平面交、二分、CDQ分治、整体二分
 6. $n \leq 1e6$ $O(n)$ 常数小的 $O(n \log n)$
输入输出**100w**的时候必须**scanf**
hash, 双指针, 并查集, kmp, AC自动机
常数小的 $O(n \log n)$ sort, 树状数组, heap, dijkstra, spfa
 7. $n \leq 1e7$ $O(\sqrt{n})$
判断质数
 8. $n \leq 1e18$ $O(\log n)$
gcd, 快速幂
 9. $n \leq 1e1000$ $O((\log n)^2)$
高精加减乘除
 10. $n \leq 1e100000$ $O(\log k \cdot \log \log k)$, k 表示位数
高精度加减, FFT/NTT
-

数学

质数

试除法

```
bool is_prime(long long x)
{
    if (x < 2) return false;
    for (int i = 2; i <= x / i; i++) //i*i<=x可能会溢出
        if (x % i == 0)
            return false;
    return true;
}
```

埃氏筛

```
int primes[N], cnt; // primes[] 存储所有素数
bool st[N]; // st[x] 存储x是否被筛掉
//筛掉每个数的倍数，如果p没有被筛掉，那么说明p不是2~p-1任何一个数倍数即，2~p-1都不是p约数
//优化：只要筛1~n所有质数的倍数就行了，唯一分解定理
void get_primes(int n)
{
    for (int i = 2; i <= n; i++)
    {
        if (st[i]) continue;
        primes[cnt++] = i;
        for (int j = i + i; j <= n; j += i) //筛倍数
            st[j] = true;
    }
}
```

线性筛

```
int primes[N], cnt; // primes[] 存储所有素数
bool st[N]; // st[x] 存储x是否被筛掉
//n只会被最小质因子筛掉
void get_primes(int n)
{
    for (int i = 2; i <= n; i++) //不要忘记等号
    {
        if (!st[i]) primes[cnt++] = i;
        for (int j = 0; primes[j] <= n / i; j++) //不要忘记等号
        {
            st[primes[j] * i] = true; //合数一定有最小质因数，用最小质因数的倍数
            //筛去合数
            if (i % primes[j] == 0) break; //prime[j]一定是i最小质因子，也一定是prime[j]*i的最小质因子
        }
    }
}
```


质因数

分解质因数

```
void divide(int n)
{
    for(int i=2;i<n/i;i++)//不要忘记等号
        if(n%i==0)//i一定是质数
        {
            int s=0;
            while(n%i==0)
            {
                n/=i;
                s++;
            }
            printf("%d %d\n",i,s);
        }
    if(n>1)printf("%d %d\n",n,1);//处理唯一一个>sqrt(n)的
    puts(" ");
}

/*
给定两个数n, m, 其中m是一个素数。
将n ( $0<=n<=2^{31}$ ) 的阶乘分解质因数, 求其中有多少个m。
while(n/m) ans+=n/m,n/=m;
*/
```

试除法求约数

```
#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);//约数通常成对出现, 特判完全平方
        }
    sort(res.begin(),res.end());
    return res;
}

int main()
{
    int n;
    cin>>n;
    while(n-->0)
    {
```

```

        int x;
        cin>>x;
        auto res=get_divisors(x);
        for(auto t:res)cout<<t<<' ';
        cout<<endl;
    }
}

```

约数个数(多个数相乘的)

```

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

typedef long long LL;

const int mod=1e9+7;

int main()
{
    int n;
    cin>>n;
    unordered_map<int,int>primes;
    while(n--)
    {
        int x;
        cin>>x;
        for(int i=2;i<=x/i;i++)
            while(x%i==0)
            {
                x/=i;
                primes[i]++;
            }
        if(x>1)primes[x]++;
    }
    LL res=1;
    for(auto prime:primes)res=res*(prime.second+1)%mod;
    cout<<res<<endl;
    return 0;
}

```

约数个数和

```

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

int main()
{
    int res=0,n;
    cin>>n;
    for(int i=1;i<=n;i++) res+=n/i;
    cout<<res;
    return 0;
}

```

约数和

```

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

const int mod=1e9+7;
typedef long long LL;
int main()
{
    int n,x;
    unordered_map<int,int>primes;
    cin>>n;
    while(n-->0)
    {
        cin>>x;
        for(int i=2;i<=x/i;i++)
            while(x%i==0)
            {
                x/=i;
                primes[i]++;
            }
        if(x>1)primes[x]++;
    }
    LL res=1;
    for(auto prime:primes)
    {
        int p=prime.first,a=prime.second;
        LL t=1;
        while(a-->0)t=(t*p+1)%mod;
        res=res*t%mod;
    }
    cout<<res<<endl;
    return 0;
}

```

欧拉函数

```
ll phi(ll x)
{
    ll 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;
}
```

筛法求欧拉函数 (1~n,欧拉函数之和)

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

typedef long long ll;
const int N=1e6+10;

ll n,primes[N],phi[N],cnt;
bool st[N];

ll get_eulers(ll n)
{
    phi[1]=1;
    for(int i=2;i<=n/i;i++)
    {
        if(!st[i])
        {
            prime[cnt++]=i;
            phi[i]=i-1;
        }
        for(int j=0;prime[j]<=n/i;j++)
        {
            st[i*prime[j]]=1;
            if(i%prime[j]==0)
            {
                phi[prime[j]*i]=phi[i]*prime[j];
                break;
            }
            phi(prime[j]*i)=phi[i]*(prime[j]-1)
        }
    }
    ll res=0;
```

```

        for(int i=1;i<=n;i++)res+=phi[i];
        return res;
    }

    int main()
    {
        ll n;
        cin>>n;

        cout<<get_eulers(n)<<endl;

        return 0;
    }

```

GCD/LCM

GCD

```

int gcd(int a, int b)
{
    return b ? gcd(b, a % b) : a;
}

```

LCM

```

int lcm(int a, int b)
{
    return a*b/gcd(a,b);
}

```

组合数

递推

```

#include<bits/stdc++.h>
using namespace std;
long long c[40][40];
int main()
{
    for(int i=0;i<=40;i++)
        for(int j=0;j<=i;j++)
        {
            if(j==0||i==0)c[i][j]=1;
            else c[i][j]=c[i-1][j-1]+c[i-1][j];
        }
    for(int i=0;i<=40;i++)
    {
        for(int j=0;j<=i;j++)

```

```

        {
            printf("%d ",c[i][j]);
        }
        printf("\n");
    }

    return 0;
}

```

递归

```

#include<stdio.h>

int c(int i,int j)
{
    if(i<0||j<0||i<j) return 0;
    if(i==j) return 1;
    if(j==1) return i;//这个是i不是1啊
    else return c(i-1,j-1)+c(i-1,j);
}

int main()
{
    int i,j,zuhe;
    scanf("%d%d",&i,&j);
    zuhe=c(i,j);
    printf("C[%d][%d]=%d",i,j,zuhe);
    return 0;
}

```

快速幂

裸的

```

typedef long long ll;
ll fastpower(ll base,ll power)
{
    ll result=1;
    while(power>0)//不会包含power=0的情况所以这种要特判
    {
        if(power&1)//等价于if(power%2==1)
        {
            result=result*base;
        }
        power>>=1;//等价于power=power/2
        base=base*base;
    }
    return result;
}

```

快速幂+取模

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
int k;
ll fastpower(ll base,ll power)
{
    ll result=1;
    result%=k;//处理1 0 1
    while(power>0)//不会包含power=0的情况所以这种要特判
    {
        if(power&1)//等价于if(power%2==1)
        {
            result=result*base%k;
        }

        power>>=1;//等价于power=power/2
        base=(base*base)%k;
    }
    return result;
}

int main()
{
    int b,p;
    cin>>b>>p>>k;
    cout<<b<<"^"<<p<<" mod "<<k<<"="<<fastpower(b,p);
    return 0;
}
```

阶乘

阶乘位数

```

#include<bits/stdc++.h>
using namespace std;
int main()
{
    double n;
    while(cin>>n)
    {
        double s=0;
        for(int i=1;i<=n;i++) s+=log10(i); //n!取对数
        cout<<int(s)+1<<endl;
    }
    return 0;
}

```

回文

判断

```

bool hw(int x)
{
    int numa=x, numb=0; //下面其实就一个数字反转
    while (numa)
    {
        numb=numb*10+numa%10;
        numa/=10;
    }
    if (x==numb) return true;
    else return false;
}

```

```

bool hw(string a)
{
    string b;
    b=a;
    reverse(b.begin(), b.end()); //不要拼错了，这玩意也不可以直接赋值
    if(a==b) return true;
    else return false;
}

```

斐波那契数列

递归

```
式子f[n]=f[n-1]+f[n-2]
终止条件f[1]=1    f[2]=1
long long f(int n)
{
    if(n==1||n==2) return 1;
    return f(n-1)+f(n-2);
}
```

递推

```
#include<bits/stdc++.h>
using namespace std;
long long f[50];
int main()
{
    int n;
    cin>>n;

    f[0]=0;f[1]=1;
    cout<<f[1]<<" ";
    for(int i=2;i<=n;i++)
    {
        f[i]=f[i-1]+f[i-2];
        cout<<f[i]<<" ";
    }
    return 0;
}
```

数学公式杂烩

公式猜测(rp+++++)

待定系数法

技巧

1. 对于分类讨论，建议取靠后的数据带入，前面数据单独输出（至于从第几项开始，你感觉可以递推或者递归了那就从哪里开始代）
2. 一般是两项，有时候可能三项，有时候可能还有常数项
3. 分类讨论，分奇偶找找规律

几何

平面分割

直线分割

折线分割

封闭曲线分割

平面分割空间

任意多边形面积

数论

gcd/lcm

互质

质数/质因数/唯一分解定理

常识

闰年：

- 1.普通情况求闰年只需除以4可除尽即可
- 2.如果是100的倍数但不是400的倍数,那就不是闰年了

月份

31天：1, 3, 5, 7, 8, 10, 12

2月闰年29，平年28

30天：4, 6, 9, 11

进制与位运算

进制转换

10进制转K进制

```
string itoA(LL n,int radix)    //n是待转数字，radix是指定的进制
{
    string ans;
    do{
        int t=n%radix;
        if(t>=0&&t<=9)    ans+=t+'0';
        else ans+=t-10+'A';//注意大小写
        n/=radix;
    }while(n!=0);    //使用do{}while（）以防止输入为0的情况
    reverse(ans.begin(),ans.end()); //逆序翻转
    return ans;
}
```

K进制转10进制

```
LL Atoi(string s,int radix)    //s是给定的radix进制字符串
{
    LL ans=0;
    for(int i=0;i<s.size();i++)
    {
        char t=s[i];
        if(t>='0'&&t<='9') ans=ans*radix+t-'0';
        else ans=ans*radix+t-'A'+10;
    }
    return ans;
}
```

Tip

1. 有时候不用真的转换，可以直接按格式打印

二进制枚举

枚举子集 $O(2^{|S|})$, $|S|$ 表示集合中元素个数

```
// s 是一个二进制数，表示一个集合，i 枚举 s 的所有子集
for (int i = S; i; i = S & i - 1)
    // blablabla
```

枚举子集的子集 $O(3^n)$

```
for (int S = 0; S < 1 << n; S ++ ) // 枚举集合 {0, ..., n - 1} 的所有子集
    for (int i = S; i; i = S & i - 1) // 枚举子集 S 的子集
        // blablabla
```

位运算

a&b按位与
a|b按位或
a^b按位异或
~a按位取反，用于表示负数 $-x=\sim x+1$
 $a<<b=a*2^b$
 $a>>b=a/2^b$ （去小数）
!a非

将 x 第 i 位取反: $x \oplus 1 \ll i$
将 x 第 i 位制成 1: $x |= 1 \ll i$
将 x 第 i 位制成 0: $x \&= -1 \wedge 1 \ll i$ 或 $x \&= \sim(1 \ll i)$
取 x 对 2 取模的结果: $x \& 1$
取 x 的第 i 位是否为 1: $x \& 1 \ll i$ 或 $x \gg i \& 1$
取 x 的最后一位: $x \& -x$
取 x 的绝对值: $(x \oplus x \gg 31) - (x \gg 31)$ (int 型)
判断 x 是否不为 2 的整次方幂: $x \& x - 1$

判断 a 是否不等于 b: $a \neq b, a - b, a \wedge b$
判断 x 是否不等于-1: $x \neq -1, x \wedge -1, x + 1, \sim x$

求二进制中1的个数

暴力 $O(\log N)$

```
int count(int x)
{
    int res = 0;
    while (x) res += x & 1, x >>= 1;
    return res;
}
```

位运算 $O(1)$

```
int count(int x)
{
    x = (x >> 1 & 0x55555555) + (x & 0x55555555);
    x = (x >> 2 & 0x33333333) + (x & 0x33333333);
    x = (x >> 4 & 0x0f0f0f0f) + (x & 0x0f0f0f0f);
    return x % 255;
}
```

$\log_2(x)$ 取整

求 $\log_2(x)$ (暴力) $O(\log x)$

```
int log_2(int x)
{
    int res = 0;
    while (x >> 1) res ++, x >>= 1;
    return res;
}
```

求 $\log_2(x)$ (位运算二分展开) $O(\log \log x)$

```
int log_2(int x)
{
    int res = 0;
    if (x & 0xffff0000) res += 16, x >>= 16;
    if (x & 0xff00) res += 8, x >>= 8;
    if (x & 0xf0) res += 4, x >>= 4;
    if (x & 0xc) res += 2, x >>= 2;
    if (x & 2) res ++;
    return res;
}
```

求log₂(x) (二分) O(loglogx)

```
int log_2(int x)
{
    int l = 0, r = 31, mid;
    while (l < r)
    {
        mid = l + r + 1 >> 1;
        if (x >> mid) l = mid;
        else r = mid - 1;
    }
}
```

预处理log₂(x) 1~n中所有数 O (n)

```
int log_2[N];    // 存 log2(i) 的取整结果
void init(int n)
{
    for (int i = 0; 1 << i <= n; i++)
        log_2[1 << i] = i;
    for (int i = 1; i <= n; i++)
        if (!log_2[i])
            log_2[i] = log_2[i - 1];
}
```

龟速加(a*b%p)

```
//把b转为2进制, b=11= (1011) 2=2^3+2^1+2^0
//a*b%p=8a%p+2a%p+a%p
//开long long
int add(int a, int b)
{
    while (b)
    {
        int x = a ^ b;
        b = (a & b) << 1;
        a = x;
    }
    return a;
}
```

STL:bitset

高精

A+B

```
#include<bits/stdc++.h>
using namespace std;
//如果k进制，那么10都改成k就行了，传进去的时候注意改A~10,F~15;
vector<int>add(vector<int> &A,vector<int> &B)
{
    if(A.size()<B.size())return add(B,A);
    vector<int>c;
    int t=0;//一定要初始化为0
    for(int i=0;i<A.size();i++)//A+B+t
    {
        t+=A[i];
        if(i<B.size())t+=B[i];
        c.push_back(t%10);
        t/=10;
    }
    if(t)c.push_back(t);//处理最高位
    return c;
}
int main()
{
    string a,b;
    vector<int>A,B;
    cin>>a>>b;
    for(int i=a.size()-1;i>=0;i--)A.push_back(a[i]-'0');//逆序输入,方便进位
    for(int i=b.size()-1;i>=0;i--)B.push_back(b[i]-'0');
    auto c=add(A,B);
    for(int i=c.size()-1;i>=0;i--)cout<<c[i];//逆序输出
    return 0;
}
```

A-B

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

void trimzero(vector<int> &A)//处理输入前的0和输出时的0
{
    while(A.size()>1&&A.back()==0)A.pop_back();
}
bool cmp(vector<int> &A,vector<int> &B)
{
    if(A.size()!=B.size())return A.size()>B.size();
    for(int i=A.size()-1;i>=0;i--)
        if(A[i]!=B[i])return A[i]>B[i];
}
```

```

        return 1;
    }

vector<int>sub(vector<int> &A,vector<int> &B)
{
    vector<int>c;
    for(int i=0,t=0;i<A.size();i++)
    {
        t=A[i]-t;
        if(i<B.size())t-=B[i];
        c.push_back((t+10)%10);
        if(t<0)t=1;
        else t=0;
    }
    trimzero(c);
    return c;
}

int main()
{
    string a,b;
    cin>>a>>b;
    vector<int>A,B;
    for(int i=a.size()-1;i>=0;i--)A.push_back(a[i]-'0');
    for(int i=b.size()-1;i>=0;i--)B.push_back(b[i]-'0');
    trimzero(A);
    trimzero(B);
    if(cmp(A,B))
    {
        auto c=sub(A,B);
        for(int i=c.size()-1;i>=0;i--)cout<<c[i];
    }
    else
    {
        auto c=sub(B,A);
        cout<<"-";
        for(int i=c.size()-1;i>=0;i--)cout<<c[i];
    }
    return 0;
}

```

A*b

```

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

```

```

void trimzero(vector<int> &A)
{
    while(A.size()>1&&A.back()==0)A.pop_back();
}
vector<int> mul(vector<int> &A,int b)
{
    vector<int>c;
    for(int i=0,t=0;i<A.size()||t;i++)
    {
        if(i<A.size())t+=A[i]*b;
        c.push_back(t%10);
        t/=10;
    }
    trimzero(c);
    return c;
}
int main()
{
    string a;
    int b;
    cin>>a>>b;
    vector<int>A;
    for(int i=a.size()-1;i>=0;i--)A.push_back(a[i]-'0');
    trimzero(A);
    auto c=mul(A,b);
    for(int i=c.size()-1;i>=0;i--)cout<<c[i];
    return 0;
}

```

A/b

```

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

void trimzero(vector<int> &A)
{
    while(A.size()>0&&A.back()==0)A.pop_back();
}
vector<int>div(vector<int> &A,int b,int &r)
{
    vector<int>c;
    r=0;
    for(int i=A.size()-1;i>=0;i--)//出发比较特别从高位开始搞
    {
        r=r*10+A[i];
        c.push_back(r/b);
        r%=b;
    }
}

```



```

        reverse(c.begin(), c.end());
        trimzero(c);
        return c;
    }
int main()
{
    string a;
    int b;
    vector<int>A;
    cin>>a>>b;
    for(int i=a.size()-1;i>=0;i--)A.push_back(a[i]-'0');
    int r;
    auto c=div(A,b,r);
    for(int i=c.size()-1;i>=0;i--)cout<<c[i];
    cout<<endl<<r<<endl;
    return 0;
}

```

大数阶乘 (vector太慢了, 用数组)

```

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

int main()
{
    int n,ws;
    while(scanf("%d",&n)!=EOF)
    {
        double s=0;
        for(int i=1;i<=n;i++)s+=log10(i);
        ws=int(s)+1;//求位数
        int f[ws];
        memset(f,0,sizeof(f));
        int ans,jw,j;
        f[0]=1;
        for(int i=2;i<=n;i++)
        {
            int jw=0;
            for(j=0;j<ws;j++)
            {
                int ans=f[j]*i+jw;
                f[j]=ans%10;
                jw=ans/10;
            }
        }
        for(j=ws-1;j>=0;j--)printf("%d",f[j]);
        printf("\n");
    }
}

```

```
    return 0;
}
```

其他处理

去前导0

```
void trimzero(vector<int> &A) //处理输入前的0和输出时的0
{
    while(A.size()>1&&A.back()==0)A.pop_back();
}
```

多组输入初始化

多次输入或者累计运算
记得清空vector
vector<int>a;
a.clear();

排序

快排

```
void quick_sort(int q[], int l, int r)
{
    if (l >= r) return; //没有数

    int i = l - 1, j = r + 1, x = q[l + r >> 1]; //l-1和r+1, 因为每次都要往中间移动一个, 这样可以保证开头的和以后的都一样不用特别处理, 移动一次后找到真正边界
    while (i < j)
    {
        do i++; while (q[i] < x);
        do j--; while (q[j] > x);
        if (i < j) swap(q[i], q[j]);
    }
    quick_sort(q, l, j), quick_sort(q, j + 1, r);
}
```

归并

排序

```
int q[N], tmp[N];
void merge_sort(int q[], int l, int r) //这里只有<=/>=没有</>
{
    if (l >= r) return;
```

```

int mid=l+r>>1;
merge_sort(q,l,mid),merge_sort(q,mid+1,r);
int k=0,i=l,j=mid+1;
while(i<=mid&&j<=r)
{
    if(q[i]<=q[j])tmp[k++]=q[i++];
    else tmp[k++]=q[j++];
}
while(i<=mid)tmp[k++]=q[i++];
while(j<=r)tmp[k++]=q[j++];

for(int i=l,j=0;i<=r;i++,j++)q[i]=tmp[j];//不要写成i=1
}

```

求逆序对

```

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

typedef long long ll;
const int N=1e6+10;

int n;
int q[N],tmp[N];

ll merge_sort(int l,int r)
{
    if(l>=r) return 0;
    int mid=l+r>>1;
    ll res=merge_sort(l,mid)+merge_sort(mid+1,r);
    int k=0,i=l,j=mid+1;
    while(i<=mid&&j<=r)
    {
        if(q[i]<=q[j])tmp[k++]=q[i++];
        else
        {
            tmp[k++]=q[j++];
            res+=mid-i+1;
        }
    }
    while(i<=mid)tmp[k++]=q[i++]; //扫尾
    while(j<=r)tmp[k++]=q[j++];
    for(int i=l,j=0;i<=r;i++,j++)q[i]=tmp[j];//不要写成i=1
    return res;
}

```

```

int main()
{
    int n;
    cin>>n;
    for(int i=0;i<n;i++)cin>>q[i];
    cout<<merge_sort(0,n-1);
}

```

STL

sort+结构体+cmp

```

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

struct student
{
    string xm;
    int y;
    int m;
    int d;
    int xh;
};student a[105];

bool cmp(student a,student b)
{
    if(a.y<b.y)
        return true; //想按什么情况来排序，就是这种情况下返回的值是true，但是这里面似乎
    不能用>=或者<=这样的符号
    else if(a.y==b.y&&a.m<b.m)
        return true;
    else if(a.y==b.y&&a.m==b.m&&a.d<b.d)
        return true;
    else if(a.y==b.y&&a.m==b.m&&a.d==b.d&&a.xh>b.xh)
        return true;
    else
        return false;
}

int main()
{
    int n;
    cin>>n;
    for(int i=1;i<=n;i++)
    {
        cin>>a[i].xm>>a[i].y>>a[i].m>>a[i].d;
    }
}

```

```

        a[i].xh=i;
    }

    sort(a+1,a+n+1,cmp);//[a+1,a+n]范围按照cmp排序

    for(int i=1;i<=n;i++)
    {
        cout<<a[i].xm<<endl;
    }

    return 0;
}

```

输入输出

快读

```

inline int read()
{
    int x=0,y=1;char c=getchar();//y代表正负(1.-1)，最后乘上x就可以了。
    while (c<'0' || c>'9') {if (c=='-') y=-1;c=getchar();} //如果c是负号就把y
    赋为-1
    while (c>='0' && c<='9') x=x*10+c-'0',c=getchar();
    return x*y; //乘起来输出
}

```

c++关闭同步

```

int main()
{
    ios::sync_with_stdio(false);
}

```

数据结构

STL

vector, 变长数组, 倍增的思想

- size() 返回元素个数
- empty() 返回是否为空
- clear() 清空
- front() / back()
- push_back() / pop_back()

`begin() / end()`

`[]`

支持比较运算，按字典序

`pair<int, int>`

`first`, 第一个元素

`second`, 第二个元素

支持比较运算，以`first`为第一关键字，以`second`为第二关键字（字典序）

`string`, 字符串

`size() / length()` 返回字符串长度

`empty()`

`clear()`

`substr(起始下标, (子串长度))` 返回子串

`c_str()` 返回字符串所在字符数组的起始地址

`queue`, 队列

`size()`

`empty()`

`push()` 向队尾插入一个元素

`front()` 返回队头元素

`back()` 返回队尾元素

`pop()` 弹出队头元素

`priority_queue`, 优先队列，默认是大根堆

`size()`

`empty()`

`push()` 插入一个元素

`top()` 返回堆顶元素

`pop()` 弹出堆顶元素

定义成小根堆的方式: `priority_queue<int, vector<int>, greater<int>> q;`

`stack`, 栈

`size()`

`empty()`

`push()` 向栈顶插入一个元素

`top()` 返回栈顶元素

`pop()` 弹出栈顶元素

`deque`, 双端队列

`size()`

`empty()`

`clear()`

```
front() / back()
push_back() / pop_back()
push_front() / pop_front()
begin() / end()
[]
```

set, map, multiset, multimap, 基于平衡二叉树（红黑树），动态维护有序序列

```
size()
empty()
clear()
begin() / end()
++, -- 返回前驱和后继，时间复杂度  $O(\log n)$ 

set / multiset
    insert() 插入一个数
    find() 查找一个数
    count() 返回某一个数的个数 (由于set不重复原则所以只返回01)
    erase()
        (1) 输入是一个数x, 删除所有x  $O(k + \log n)$ 
        (2) 输入一个迭代器, 删除这个迭代器
    lower_bound() / upper_bound()
        lower_bound(x) 返回大于等于x的最小的数的迭代器
        upper_bound(x) 返回大于x的最小的数的迭代器
map / multimap
    insert() 插入的数是一个pair
    erase() 输入的参数是pair或者迭代器
    find()
    [] 注意multimap不支持此操作。 时间复杂度是  $O(\log n)$ 
    lower_bound() / upper_bound()
```

unordered_set, unordered_map, unordered_multiset, unordered_multimap, 哈希表

和上面类似，增删改查的时间复杂度是 $O(1)$

不支持 lower_bound() / upper_bound(), 迭代器的++, --

bitset, 压位

```
bitset<10000> s;
~, &, |, ^
>>, <<
==, !=
[]
```

count() 返回有多少个1

`any()` 判断是否至少有一个1
`none()` 判断是否全为0

`set()` 把所有位置成1
`set(k, v)` 将第k位变成v
`reset()` 把所有位变成0
`flip()` 等价于~
`flip(k)` 把第k位取反

栈

后缀表达式

单调栈

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

const int N=1e6+5;

int n;
int stk[N],tt;

int main()
{
    cin>>n;
    for(int i=0;i<n;i++)
    {
        int x;
        cin>>x;
        while(tt&&stk[tt]>=x) tt--; //1~i-1单调递增的栈,出栈了就不会再回来了
        if(tt) cout<<stk[tt]<<" ";
        else cout<<-1<<" ";
        stk[++tt]=x;
    }
}
```

队列

普通队列

```
// hh 表示队头, tt表示队尾, 队列从0开始
int q[N], hh = 0, tt = -1;
//如果第一个要插入队尾的元素已经知道了, 那tt开局用0即可
int hh=0, tt=-1
q[0]=1;
// 向队尾插入一个数
q[ ++ tt] = x;

// 从队头弹出一个数
hh ++ ;

// 队头的值
q[hh];
int t=q[hh++]; //t变为队头同时弹出原来的队头常用于bfs

// 判断队列是否为空
if (hh <= tt)
{

}

//循环
q[++tt]=q[hh];
hh++;
```

循环队列

```
// hh 表示队头, tt表示队尾的后一个位置
int q[N], hh = 0, tt = 0;

// 向队尾插入一个数
q[tt ++ ] = x;
if (tt == N) tt = 0;

// 从队头弹出一个数
hh ++ ;
if (hh == N) hh = 0;

// 队头的值
q[hh];

// 判断队列是否为空
if (hh != tt)
{

}
```

单调队列(滑动窗口)

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

const int N=1e6+5;

int a[N],q[N],n,k;//q存下标

int main()
{
    scanf("%d%d",&n,&k);
    for(int i=0;i<n;i++)scanf("%d",&a[i]);
    int hh=0,tt=-1;
    for(int i=0;i<n;i++)
    {
        //判断队头是已经滑出窗口
        if(hh<=tt&& i-k+1>q[hh])hh++;
        while(hh<=tt&&a[q[tt]]>=a[i])tt--;
        q[++tt]=i;
        if(i>=k-1)printf("%d ",a[q[hh]]);
    }
    puts("");
    hh=0,tt=-1;
    memset(q,0,sizeof(q));
    for(int i=0;i<n;i++)
    {
        if(hh<=tt&& i-k+1>q[hh])hh++;
        while(hh<=tt&&a[q[tt]]<=a[i])tt--;
        q[++tt]=i;
        if(i>=k-1)printf("%d ",a[q[hh]]);
    }
    return 0;
}
```

Trie树

存储查找字符串集合

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

const int N=1e6+10;
```

int son[N][26], cnt[N], idx; // 下标是0的点既是根节点又是空节点, cnt是对应每个停止符的数量。

char str[N];

void insert(char *str)

```
{
    int p=0;
    for(int i=0; str[i]; i++)
    {
        int u=str[i]-'a';
        if(!son[p][u]) son[p][u]=++idx; // p是根u是儿子, 如果没有儿子, idx只是查询有没有
        p=son[p][u];
    }
    cnt[p]++;
}
```

int query(char *str)

```
{
    int p=0;
    for(int i=0; str[i]; i++)
    {
        int u=str[i]-'a';
        if(!son[p][u]) return 0;
        p=son[p][u];
    }
    return cnt[p];
}
```

int main()

```
{
    int n;
    char op[2];
    scanf("%d", &n);
    while(n--)
    {
        scanf("%s%s", op, str);
        if(op[0]=='I') insert(str);
        else printf("%d\n", query(str));
    }
    return 0;
}
```

并查集

普通并查集

```
int find(int x)
{
    if (f[x] != x) f[x] = find(f[x]);
    return f[x];
}
void merge_set(int x, int y)
{
    int fx = find(x), fy = find(y);
    if (fx == fy) return; // 不要忘记
    else f[fx] = fy;
}
```

维护距离(向量本质, 有向图)

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

const int N = 1e6 + 10;
int n, k, cnt;
int f[N], d[N];
int find(int x)
{
    if (f[x] != x)
    {
        int t = find(f[x]);
        d[x] += d[f[x]];
        f[x] = t;
    }
    return f[x];
}
int main()
{
    cin >> n >> k;
    for (int i = 1; i <= n; i++) f[i] = i;
    while (k--)
    {
        int op, x, y;
        cin >> op >> x >> y;
        if (x > n || y > n) cnt++;
        else
        {
            int fx = find(x), fy = find(y);
            if (op == 1) // 同类
```

```

        {
            if (fx==fy&&(d[x]-d[y])%3) cnt++;
            else if (fx!=fy)
            {
                f[fx]=fy;
                d[fx]=d[y]-d[x];
            }
        }
    else
    {
        if (fx==fy&&(d[x]-d[y]-1)%3) cnt++;
        else if (fx!=fy)
        {
            f[fx]=fy;
            d[fx]=d[y]+1-d[x];
        }
    }
}

cout<<cnt;
return 0;
}

```

维护大小

```

#include<bits/stdc++.h>
using namespace std;
const int N=1e6+10;

int n,m;
int f[N],siz[N];
int find(int x)
{
    if (f[x]!=x) f[x]=find(f[x]);
    return f[x];
}

int main()
{
    scanf("%d%d",&n,&m);
    for (int i=1;i<=n;i++) //看清题目可能从0开始
    {
        f[i]=i;
        siz[i]=1;
    }
    while (m--)
    {

```

```

int a,b;
char op[2];
scanf("%s",op);
if(op[0]=='C')
{
    scanf("%d%d",&a,&b);
    if(find(a)==find(b)) continue;//判断一下有没有在同一集合里了
    siz[find(b)]+=siz[find(a)];
    f[find(a)]=find(b);
}
else if(op[1]=='1')
{
    scanf("%d%d",&a,&b);
    if(find(a)==find(b)) puts("Yes");
    else puts("No");
}
else
{
    scanf("%d",&a);
    printf("%d\n",siz[find(a)]);
}
}
return 0;
}

```

扩展域

```

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

const int N=1e6+10;
int f[N],enem[N];//enem存p的敌人
int find(int x)
{
    if(f[x]!=x) f[x]=find(f[x]);
    return f[x];
}
void merge_set(int x,int y)
{
    int fx=find(x),fy=find(y);
    if(fx==fy) return;//不要忘记
    else f[fx]=fy;
}
int main()
{
    int n,m,cnt;
    char op[2];

```

```

scanf ("%d%d", &n, &m);
cnt=0;
for(int i=1;i<=2*n;i++) f[i]=i;
for(int i=1;i<=m;i++)
{
    int p,q;
    scanf ("%s%d%d", op, &p, &q);
    int fp=find(p),fq=find(q);
    if (op[0]=='F') merge_set (p, q);
    else
    {
        if (!enem[p]) enem[p]=q;
        else merge_set (q, enem[p]);
        if (!enem[q]) enem[q]=p;
        else merge_set (p, enem[q]);
    }
}
for(int i=1;i<=n;i++)
{
    if (f[i]==i) cnt++;
}
printf ("%d", cnt);
return 0;
}

```

链表

普通链表

```

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

const int N=1e6+5;
int head,e[N],ne[N],idx;//把e和ne想成节点idx的两个属性
//初始化
void init()
{
    head=-1;
    idx=0;//0开头
}
//链表头插入x
void add_to_head(int x)
{
    e[idx]=x,ne[idx]=head,head=idx++;//最后一个重新把head指向了idx，因为开局head为-1嘛
}
//在k后面插入一个点

```

```

void add(int k,int x)
{
    e[idx]=x,ne[idx]=ne[k],ne[k]=idx++;//先用idx在+1
}
//移除k后面的点
void remove(int k)
{
    ne[k]=ne[ne[k]];
}
//移除头节点，要保证头节点存在
void remove_head()
{
    head=ne[head];
}

int main()
{
    int m,k,x;
    cin>>m;
    char op;
    init();//不要忘记初始化
    while(m--)
    {
        cin>>op;
        if(op=='H')
        {
            cin>>x;
            add_to_head(x);
        }
        else if(op=='D')
        {
            cin>>k;
            if(!k)remove_head();
            else remove(k-1); //因为0开头的，所以第k个下标是k-1
        }
        else if(op=='I')
        {
            cin>>k>>x;
            add(k-1,x);
        }
    }
    for(int i=head;i!=-1;i=ne[i])cout<<e[i]<<" ";//链表输出方式,记住是ne[i]
    和i!=-1

    return 0;
}

```



```
}
```

双链表

哈希表

堆

查找

二分查找

整数二分

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

const int N=1e5+5;
int a[N],x;

bool check_1(int mid) //判定条件if中的式子要考虑会不会溢出或者因为整型除法失效
{
    if(a[mid]>=x) return 1; //不要写成a[x]=x
    else return 0;
}

bool check_2(int mid)
{
    if(a[mid]<=x) return 1;
    else return 0;
}

int bsearch_1(int l,int r) //第一个满足条件的值
{
    while(l<r)
    {
        int mid=l+r>>1;
        if(check_1(mid)) r=mid; //方便记忆，右边第一个
        else l=mid+1; //别忘记else
    }
    if(a[l]!=x) return -1;
    else return l; //不要写成return 1;
}

int bsearch_2(int l,int r) //最后一个满足条件的值
```

```

{
    while(l<r)
    {
        int mid=l+r+1>>1;
        if(check_2(mid))l=mid;//方便记忆，左边最后一个
        else r=mid-1;//别忘记else
    }
    if(a[l]!=x)return -1;
    else return l;
}

int main()
{
    int n,q;
    scanf("%d%d",&n,&q);
    for(int i=0;i<n;i++)scanf("%d",&a[i]);
    while(q--)
    {
        scanf("%d",&x);
        printf("%d %d\n",bsearch_1(0,n-1),bsearch_2(0,n-1));
    }
}

```

浮点数二分

```

bool check(double x) { /* ... */ } // 检查x是否满足某种性质

double bsearch_3(double l, double r) //输入l和r的时候保证l<r不要输入一个负数就
反过来了不能写(-n,n)
{
    const double eps = 1e-6;    // eps 表示精度，取决于题目对精度的要求，一般比
要求的两位有效数字
    while (r - l > eps)
    {
        double mid = (l + r) / 2;
        if (check(mid)) r = mid;
        else l = mid;
    }
    return l;
}

```

搜索与图论

图

图的存储

邻接表

```
// 对于每个点k, 开一个单链表, 存储k所有可以走到的点。h[k]存储这个单链表的头结点
int h[N], e[N], ne[N], idx;

// 添加一条边a->b
void add(int a, int b)
{
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++ ;
}

// 初始化
idx = 0;
memset(h, -1, sizeof h);
```

邻接矩阵

建图小技巧

1. 反向建图日神仙
- 2.

DFS

有多少个连通块 (洪泛)

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

int n,m,cnt;
char mp[505][505];
int xx[]={0,0,1,-1};
int yy[]={1,-1,0,0};

void dfs(int x,int y)
{
    for(int i=0;i<4;i++)
    {
        int dx=x+xx[i];
        int dy=y+yy[i];
        if(dx>=0&&dx<=n+1&&dy>=0&&dy<=m+1&&mp[dx][dy]!='*')
        {
            mp[dx][dy]='*';//直接标记
        }
    }
}
```

```

        dfs(dx, dy);
    }
}

int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            char c;
            cin>>c;
            mp[i][j]=c;
        }
    }
    dfs(0,0); //方式开局就是这*
    for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++)
            if(mp[i][j]=='0')
                cnt++;

    cout<<cnt;
    return 0;
}

```

DFS遍历图(树的重心)

```

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

const int N=1e6+10;
const int M=N*2; //无向图两条边

int h[N], e[M], ne[M], idx;
bool st[N];
int ans=N;
int n;
void add(int a, int b) //a指向b
{
    e[idx]=b, ne[idx]=h[a], h[a]=idx++;
}
//以u为根的子树大小（点数量）
int dfs(int u)
{
    st[u]=1; //标记一下

```

```

int sum=1,res=0;//sum
for(int i=h[u];i!=-1;i=ne[i])//遍历与u连通的点
{
    int j=e[i];
    if(!st[j])
    {
        int s=dfs(j);//当前子树大小
        res=max(res,s);
        sum+=s;//s是u为根子数大小一部分
    }
}
res=max(res,n-sum);//n-sum为，子树上面一坨
ans=min(ans,res);
return sum;//以u为根子节点大小
}
int main()
{
    cin>>n;
    memset(h,-1,sizeof(h));//初始化
    for(int i=1;i<n;i++)
    {
        int a,b;
        cin>>a>>b;
        add(a,b),add(b,a);//无向图两条边
    }
    dfs(1);//图当中的编号开始搜，随便从哪个点开始都可以
    cout<<ans<<endl;
    return 0;
}

```

列出所有解

全排列

```

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

int n;
bool vis[30];
int a[20];

void pr()
{
    for(int i=1;i<=n;i++)
    {
        cout<<setw(5)<<a[i];
    }
}

```

```

    }
    cout<<endl;
}
void dfs(int x) //x是层数
{
    if(x>n)
    {
        pr(); //超出了n就结束了
    }
    for(int i=1; i<=n; i++)
    {
        if(!vis[i])
        {
            a[x]=i; //第x层是i;
            vis[i]=1;
            dfs(x+1);
            vis[i]=0; //释放回到上一个节点，消去访问记录，其实a[x]也要消去只不过新的值会覆盖
        }
    }
}

int main()
{
    cin>>n;
    dfs(1);
    return 0;
}

```

STL

```

//prev_permutation函数可以制造前一个排列，如果已经为第一个，则返回false。
#include<bits/stdc++.h>
using namespace std;
int n, a[10000];
int main()
{
    cin>>n;
    for(int i=0; i<n; i++) //读入数据
        cin>>a[i];
    if(prev_permutation(a, a+n)) //如果为真就输出数组
        for(int i=0; i<n; i++)
            cout<<a[i]<<" ";
    else cout<<"ERROR"; //否则输出ERROR
    cout<<endl;
    return 0;
}

```

```
//next_permutation同理
int main()
{
    string str = "abcde";
    int num = 1;
    while(next_permutation(str.begin(),str.end()))
    {
        num++;
        cout<<str<<endl;
        if (num==5)
            break;
    }
    return 0;
}
```

组合输出

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

int n,r;
int a[50];
bool vis[50];

void pr()
{
    for(int i=1;i<=r;i++)
        cout<<setw(3)<<a[i];
    cout<<endl;
}

void dfs(int x)
{
    if(x>r)
    {
        pr();
        return;
    }
    for(int i=1;i<=n;i++)
    {
        if(!vis[i]&&(i>a[x-1]||x==1))
        {
            a[x]=i;
            vis[i]=1;
            dfs(x+1);
            vis[i]=0;
        }
    }
}
```

```

    }
}
}
int main()
{
    cin>>n>>r;
    dfs(1);
    return 0;
}

```

BFS

最短步数（边的权值均为1，STL写法）

```

#include<bits/stdc++.h>
using namespace std;
int l,r,c;
int xx[]={1,-1,0,0,0,0};
int yy[]={0,0,1,-1,0,0};
int zz[]={0,0,0,0,1,-1};
int sx,sy,sz,ex,ey,ez;
char mp[40][40][40];
bool vis[40][40][40];
bool flag;
struct node
{
    int x,y,z,s;//s存步数
};

void bfs(int z,int x,int y)
{
    queue<node>q;
    q.push((node){x,y,z,0}); //创建结构体队列
    vis[z][x][y]=1; //不要忘记
    while(!q.empty())
    {
        if(q.front().x==ex&&q.front().y==ey&&q.front().z==ez)
        {
            flag=1;
            printf("Escaped in %d minute(s).",q.front().s);
            break;
        }
        for(int i=0;i<6;i++)
        {
            int dx=q.front().x+xx[i]; //是队头的xyz不是x+xx[i]
            int dy=q.front().y+yy[i];
            int dz=q.front().z+zz[i];

```


//看清地图范围0~n-1还是1~n, 看清是n*n还是n*m, 哪个是行哪个是列也要看

清楚

```
        if(dx>=1&&dy>=1&&dz>=1&&dz<=1&&dx<=r&&dy<=c&&!vis[dz][dx]
[dy]&&mp[dz][dx][dy]!='#')
        {
            q.push((node){dx,dy,dz,q.front().s+1});
            vis[dz][dx][dy]=1;
        }
    }
    q.pop();
}
int main()
{
    cin>>l>>r>>c;
    for(int i=1;i<=l;i++)
    {
        for(int j=1;j<=r;j++)
        {
            for(int k=1;k<=c;k++)
            {
                cin>>mp[i][j][k];
                if(mp[i][j][k]=='S')
                {
                    sz=i;sx=j;sy=k;
                }
                if(mp[i][j][k]=='E')
                {
                    ez=i;ex=j;ey=k;
                }
            }
        }
    }
    bfs(sz,sx,sy);
    if(!flag)printf("Trapped!");
    return 0;
}
```

BFS+路径保存

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

struct node
{
    int x,y,s;
};

int n,m;
```

```

int xx[]={0,0,1,-1};
int yy[]={1,-1,0,0};
const int N=105;
node pre[N][N]; //保存路径
int g[N][N]; //保存图
bool vis[N][N];
void bfs(int sx,int sy)
{
    queue<node>q;
    q.push((node){1,1,0});
    vis[sx][sy]=1; //不要忘记
    pre[sx][sy]=(node){1,1,0};
    while(!q.empty())
    {
        if(q.front().x==n&&q.front().y==m)
        {
            cout<<q.front().s<<endl;
            cout<<n<<" "<<m<<endl;
            while(n!=sx||m!=sy) //输出路径
            {
                cout<<pre[n][m].x<<" "<<pre[n][m].y<<endl;
                n=pre[n][m].x;
                m=pre[n][m].y;
            }
            break;
        }
        for(int i=0;i<4;i++)
        {
            int dx=q.front().x+xx[i];
            int dy=q.front().y+yy[i];
            if(dx>=1&&dy>=1&&dx<=n&&dy<=m&&!g[dx][dy]&&!vis[dx][dy])
            {
                pre[dx][dy]=(node)
{q.front().x,q.front().y,q.front().s}; //保留从哪里转移过来的就行
                q.push((node){dx,dy,q.front().s+1});
                vis[dx][dy]=1;
            }
        }
        q.pop();
    }
}

int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++)
            cin>>g[i][j];
    bfs(1,1);
}

```

```
}
```

BFS遍历图(边权1最短路, 手动模拟队列写法)

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

const int N=1e6+10;
int h[N],e[N],ne[N],idx;
int n,m;
int d[N],q[N]; //d距离, q队列

void add(int a,int b)
{
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}

int bfs()
{
    int hh=0,tt=0;
    q[0]=1; //0号节点是编号为1的点, q[0]=v可以做v开始搜的广搜
    memset(d,-1,sizeof(d));
    d[1]=0; //存储每个节点离起点的距离, 这个不要忘了
    while(hh<=tt)
    {
        int t=q[hh++]; //t=q[hh] 队头同时hh+1弹出队头
        for(int i=h[t];i!=-1;i=ne[i])
        {
            int j=e[i];
            //如果j没被扩展过
            if(d[j]==-1)
            {
                d[j]=d[t]+1; //d[j] 存储j离起点距离, 并标记访问过
                q[++tt]=j; //压入j
            }
        }
    }
    return d[n];
}

int main()
{
    cin>>n>>m;
    memset(h,-1,sizeof(h));
    for(int i=1;i<=m;i++)
    {
        int a,b;
```

```

        cin>>a>>b;
        add(a,b);
    }
    cout<<bfs()<<endl;
    return 0;
}

```

拓扑序列 (有向无环图AOV)

```

#include<bits/stdc++.h>
using namespace std;
const int N=1e6+10;
int n,m;
int h[N],e[N],ne[N],idx;
int q[N],d[N]; //q队列存储层次遍历序列，d存储i号节点入度

void add(int a,int b)
{
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}
//返回布尔序列是否存在，若存在，则存储在q数组中
bool topsort()
{
    int hh=0,tt=-1;
    //遍历每个节点，入队为0则入队
    for(int i=1;i<=n;i++)
        if(!d[i])
            q[++tt]=i;

    while(hh<=tt)
    {
        //队列不为空则取出头节点
        int t=q[hh++]; //出队的顺序就是拓扑序
        //遍历头节点每个出边
        for(int i=h[t];i!=-1;i=ne[i])
        {
            int j=e[i];
            //出边能到的节点入度减1
            d[j]--;
            if(d[j]==0) q[++tt]=j; //如果节点j，入度0则入队
        }
    }

    return tt==n-1; //不要打成=，所有点都入队了说明存在拓扑序列
}

int main()

```

```

{
    cin>>n>>m;
    memset(h,-1,sizeof(h));
    for(int i=0;i<m;i++)
    {
        int a,b;
        cin>>a>>b;
        add(a,b);
        d[b]++; //b节点入度增加1
    }
    if(topsort())
    {
        for(int i=0;i<n;i++)printf("%d ",q[i]);
        puts("");
    }
    else puts("-1");
    return 0;
}

```

连通块里多少块

```

#include<bits/stdc++.h>
using namespace std;
int xx[]={0,0,1,-1};
int yy[]={1,-1,0,0};
int vis[1005][1005];
bool mp[1005][1005];
int ans[1000005];
int color,cnt,n,m;

void bfs(int x, int y)
{
    queue<int>h;
    queue<int>l;
    h.push(x);l.push(y);
    vis[x][y]=color;
    while(!h.empty())
    {
        for(int i=0;i<4;i++)
        {
            int dx=h.front()+xx[i];
            int dy=l.front()+yy[i];
            if(dx>=1&&dx<=n&&dy>=1&&dy<=n&&mp[dx][dy]!=mp[h.front()][l.front()]&&!vis[dx][dy])
            {
                h.push(dx);l.push(dy);
                vis[dx][dy]=color; //颜色标记区分不同的连通块
            }
        }
    }
}

```

```

        }
    }
    h.pop(); l.pop(); //弹出多少次就有多少格子
    cnt++;
}
return;
}
int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
        {
            char ch;
            cin>>ch;
            if(ch=='1')mp[i][j]=1;
            else mp[i][j]=0;
        }
    }
    //如果是确定的起点那下面的直接bfs(sx,sy)即可
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
        {
            if(!vis[i][j])//排除已经搜索过的连通块
            {
                color++;
                bfs(i,j);
                ans[color]=cnt;
                cnt=0;//初始化cnt
            }
        }
    }
    for(int i=1;i<=m;i++)
    {
        int x,y;
        cin>>x>>y;
        cout<<ans[vis[x][y]]<<endl;
    }
    return 0;
}

```

前缀和/差分

一维前缀和

```
S[i] = a[1] + a[2] + ... a[i]
a[l] + ... + a[r] = S[r] - S[l - 1]
```

二维前缀和

$S[i, j]$ = 第*i*行*j*列格子左上部分所有元素的和
以(x_1, y_1)为左上角, (x_2, y_2)为右下角的子矩阵的和为:
 $S[x_2, y_2] - S[x_1 - 1, y_2] - S[x_2, y_1 - 1] + S[x_1 - 1, y_1 - 1]$

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

const int N=1e3+5;
int mp[N][N], dp[N][N];
int main()
{
    int n,m,q,x1,x2,y1,y2;
    cin>>n>>m>>q;
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            cin>>mp[i][j];
        }
    }
    memset(dp,0,sizeof(dp));

    //预处理二位前缀和数组dp
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            dp[i][j]=dp[i-1][j]+dp[i][j-1]-dp[i-1][j-1]+mp[i][j];
        }
    }

    while(q--)
    {
        cin>>x1>>y1>>x2>>y2;
        cout<<dp[x2][y2]-dp[x2][y1-1]-dp[x1-1][y2]+dp[x1-1][y1-1]<<endl;
    }
}
```

一维差分

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

//给区间[1, r]中的每个数加上c: B[1] += c, B[r + 1] -= c
const int N=1e6+5;
int a[N],b[N];
int main()
{
    int n,l,r,c;
    cin>>n;
    for(int i=1;i<=n;i++)
    {
        cin>>a[i];
        b[i]=a[i]-a[i-1]; //构造差分数组
    }
    while(cin>>l>>r>>c)
    {
        b[l]+=c;
        b[r+1]-=c;
        for(int i=1;i<=n;i++) a[i]=a[i-1]+b[i];
        for(int i=1;i<=n;i++) cout<<a[i]<<" "; //如果是连续差分求最终值，把这条
        放到while外面就可以了，一般差分数据量比较大建议快速读入
        cout<<endl;
    }
    return 0;
}
```

二维差分

给以 $(x1, y1)$ 为左上角， $(x2, y2)$ 为右下角的子矩阵中的所有元素加上 c ：
 $S[x1, y1] += c$, $S[x2 + 1, y1] -= c$, $S[x1, y2 + 1] -= c$, $S[x2 + 1, y2 + 1] += c$

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

const int N=1e4+5;
int a[N][N],b[N][N];

inline int read()
{
    int x=0,y=1;char c=getchar(); //y代表正负(1.-1)，最后乘上x就可以了。
}
```



```

    while (c<'0' || c>'9') {if (c=='-') y=-1;c=getchar();} //如果c是负号就把y
    赋为-1
    while (c>='0' && c<='9') x=x*10+c-'0', c=getchar();
    return x*y; //乘起来输出
}

int main()
{
    int n,m,q,x1,x2,y1,y2,c;
    n=read(),m=read(),q=read();
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            a[i][j]=read();
            b[i][j]=a[i][j]-a[i-1][j]-a[i][j-1]+a[i-1][j-1]; //预处理差分
        }
    }
    while(q--)
    {
        x1=read(),y1=read(),x2=read(),y2=read(),c=read();
        b[x1][y1]+=c;
        b[x1][y2+1]-=c;
        b[x2+1][y1]-=c;
        b[x2+1][y2+1]+=c;
    }
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            a[i][j]=a[i-1][j]+a[i][j-1]-a[i-1][j-1]+b[i][j];
            printf("%d ",a[i][j]);
        }
        printf("\n");
    }
    return 0;
}

```

注意事项

1. 前缀和，差分尽量使用快读
2. 涉及最大最小到时候min，max初值设为0，以免遗漏开头的
3. 前缀和左上角，差分右下角，两者互为逆运算，容斥定理可推公式
4. 前缀和区间快速求和，差分区间增减修改

字符串

KMP

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

const int N=1e5+10,M=1e6+10;
int n,m;
char p[N],s[M];
int ne[N]; //最长公共前缀后缀

int main()
{
    cin>>n>>p+1>>m>>s+1;
    //预处理 ne数组
    for(int i=2,j=0;i<=n;i++) //从第二个开始处理，第一个肯定0啊
    {
        while(j&&p[i]!=p[j+1])j=ne[j]; //j+1和i试探一下看一不一样，不一样就
j=ne[i]直到开头
        if(p[i]==p[j+1])j++;
        ne[i]=j;
    }
    //kmp 匹配
    for(int i=1,j=0;i<=m;i++)
    {
        while(j&&s[i]!=p[j+1])j=ne[j];
        if(s[i]==p[j+1])j++;
        if(j==n)
        {
            printf("%d ",i-n);
            j=ne[j];
        }
    }
    return 0;
}
```

滑动窗口

DP

DP思考方式

状态表示

集合

1. 维度的确定是最少要用几个维度来唯一确定如：背包有容量和价值表状态，最原始两维度。最长子序列以*i*结尾，就只要一个。最长公共子序列两个序列，就要两个维度
2. 所有满足+（题目条件+状态表示的条件）+的集合

属性

1. max(开long long)
2. min（开long long）
3. 方案数（直接开unsigned long long防爆）
4. 具体方案（记录状态转移）

状态计算

划分

原则：补充不漏

1. 以*i*-1为倒数第二个（LIS）
2. 转移来源(数字三角)
3. 选*i*与不选*i*/选几个（背包）
4. *a*[*i*],*b*[*j*]是否包含在子序列当中

计算过程

要保证前面的已经计算好了，状态转移不能成环

背包

01背包

二维

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

const int N=1e4+10;

int n,m;
int v[N],w[N];
int f[N][N];

int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++) cin>>v[i]>>w[i];
```

```

//所有状态f[0~n][0~m]
//f[0][0~m]=0所以i就不从0开始了
for(int i=1;i<=n;i++)
{
    for(int j=0;j<=m;j++)
    {
        f[i][j]=f[i-1][j]; //左边不含i，最大值就是f[i-1][j]
        if(j>=v[i]) f[i][j]=max(f[i][j], f[i-1][j-v[i]]+w[i]); //装得下
        //v[i]才有这种情况，第一个就是左边最大值，第二个就是右边最大值，>=不要打成>
    }
}
cout<<f[n][m]<<endl;
return 0;
}

```

一维

```

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

const int N=1e4+10;
int n,m;
int v[N],w[N];
int f[N]; //有时候要开long long 不然会爆

int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++) cin>>v[i]>>w[i];

    for(int i=1;i<=n;i++)
        for(int j=m;j>=v[i];j--)
            f[j]=max(f[j], f[j-v[i]]+w[i]);
    cout<<f[m]<<endl;
    return 0;
}

```

完全背包

二维

```

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

const int N=1e4+10;

```

```

int n,m;
int v[N],w[N];
int f[N][N];

int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>v[i]>>w[i];

    for(int i=1;i<=n;i++)
        for(int j=0;j<=m;j++)
            for(int k=0;k*v[i]<=j;k++)
                f[i][j]=max(f[i][j],f[i-1][j-v[i]*k]+w[i]*k);
    cout<<f[n][m]<<endl;
    return 0;
}

```

二维优化

```

#include<bits/stdc++.h>
using namespace std;
const int N = 1010;
int f[N][N];
int v[N],w[N];
int main()
{
    int n,m;
    cin>>n>>m;
    for(int i = 1 ; i <= n ; i ++ )cin>>v[i]>>w[i];

    for(int i = 1 ; i<=n ; i++)
        for(int j = 0 ; j<=m ; j++)
        {
            f[i][j] = f[i-1][j];
            if(j>=v[i])f[i][j]=max(f[i][j],f[i][j-v[i]]+w[i]);
        }
    cout<<f[n][m]<<endl;
}

```

一维优化

```

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

const int N=1e4+10;

```

```

int n,m;
int v[N],w[N];//有时候要开long long不然会爆
int f[N];

int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>v[i]>>w[i];

    for(int i=1;i<=n;i++)
        for(int j=v[i];j<=m;j++)
            f[j]=max(f[j],f[j-v[i]]+w[i]);
    cout<<f[m]<<endl;
    return 0;
}

```

多重背包

暴力朴素

```

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

const int N=1e4+10;

int n,m;
int v[N],w[N],s[i];
int f[N][N];

int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>v[i]>>w[i]>>s[i];

    for(int i=1;i<=n;i++)
        for(int j=0;j<=m;j++)
            for(int k=0;k<=s[i]&& k*v[i]<=j;k++)
                f[i][j]=max(f[i][j],f[i][j-v[i]*k]+w[i]*k)
    cout<<f[n][m]<<endl;
    return 0;
}

```

二进制优化

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

const int N=25000,M=2010;//N要拆出来所以1000*log2000

int n,m;
int v[N],w[N];
int f[N];
int main()
{
    cin>>n>>m;
    int cnt=0;
    for(int i=1;i<=n;i++)
    {
        int a,b,s;//体积, 价值, 个数
        cin>>a>>b>>s;
        int k=1;
        while(k<=s)
        {
            cnt++;
            v[cnt]=a*k;//k个物品打包
            w[cnt]=b*k;//k个物品打包
            s-=k;
            k*=2;
        }
        if(s>0)//补上c
        {
            cnt++;
            v[cnt]=a*s;
            w[cnt]=b*s;
        }
    }
    //01背包
    n=cnt;
    for(int i=1;i<=n;i++)
        for(int j=m;j>=v[i];j--)
            f[j]=max(f[j],f[j-v[i]]+w[i]);

    cout<<f[m]<<endl;
    return 0;
}
```

分组背包

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

const int N=110;

int n,m;
int v[N][N],w[N][N],s[N]; //s表示第i组物品种类
int f[N];

int main()
{
    cin>>n>>m; //n组物品, m容量
    for(int i=1;i<=n;i++)
    {
        cin>>s[i];
        for(int j=0;j<s[i];j++)
            cin>>v[i][j]>>w[i][j];
    }

    for(int i=1;i<=n;i++)
        for(int j=m;j>=0;j--) //i-1推i逆序
            for(int k=0;k<s[i];k++) //有点像完全背包, k就是下标, 注意自己是0开始
                //还是1开始的。选第i组的第k件物品
                if(v[i][k]<=j)
                    f[j]=max(f[j],f[j-v[i][k]]+w[i][k]);

    cout<<f[m]<<endl;
    return 0;
}
```

背包方案数

二维

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

const int N=1e4+10;
int w[N],f[N][N];
int main(){
    int n,m;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++) scanf("%d",&w[i]);
    for(int i=0;i<=n;i++) f[i][0]=1; //从0开始

    for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++)
        {
```



```

        f[i][j]+=f[i-1][j];
        if(j>=w[i])f[i][j]+=f[i-1][j-w[i]];
    }

    printf("%d",f[n][m]);
    return 0;
}

```

一维

```

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

const int N=1e4+10;
int w[N],f[N];
int main()
{
    int n,m;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)scanf("%d",&w[i]);
    f[0]=1;
    for(int i=1;i<=n;i++)
        for(int j=m;j>=w[i];j--)
            f[j]+=f[j-w[i]];

    printf("%d",f[m]);
    return 0;
}

```

线性DP

数字三角

最长上升子序列 (LIS)

最长公共子序列 (LCS)

区间DP

记忆化搜索