

# ACM模板

---

by 鱼竿钓鱼干

E-mail:[851892190@qq.com](mailto:851892190@qq.com)

参考：acwing板子（以这个为主+个人做题经验补充），洛谷题解，各类博客，各平台比赛，算法竞赛进阶指南

版本：2021/6/13

更新内容：

1. 计算几何基础
2. 数位DP板子
3. 删除一些不必要的东西
4. stirling数
5. 快速幂防爆
6. 单调队列滑动窗口长度有范围的版本
7. DP参数表
8. Pollard-rho大数分解法 $O(N^{1/4})$

计划更新内容（暑假）：

1. 数组转vector
2. 线段树
3. 计算几何

---

## ACM模板

复杂度反推

数学

质数

质因数

欧拉函数

费马小定理

整除分块

组合数

Stirling数

错排

阶乘

斐波那契数列

扩展欧几里得

博弈论

进制与位运算

进制转换

二进制枚举

位运算

连续自然数异或前缀和 $O(1)$

等差数列异或和

求二进制中1的个数

$\log_2(x)$ 取整

龟速加( $a*b\%p$ ，主要防爆)

快速幂

高精

- 排序
- 输入输出
- 数据结构
  - STL
  - 栈
    - 表达式
    - 单调栈
  - 队列
    - 单调队列(滑动窗口)
  - Trie树
    - 存储查找字符串集合
  - 并查集
  - 哈希表
  - ST (区间查询)
  - Multiset实现带修改的最值查询**
  - 线段树
- 查找
  - 二分
- 搜索与图论
  - 图
    - DFS
    - BFS
    - 最短路
    - 最小生成树
    - 二分图
  - 前缀和/差分
    - 前缀和
    - 差分
- 字符串
  - KMP
- DP
  - 参数表
  - 经典模型
    - 背包
      - 01背包
      - 完全背包
      - 多重背包
      - 分组背包
      - 背包方案数
  - 数位DP
- 计算几何
  - 求解凸包
  - 平面最近点对
  - 半平面交

---

## 复杂度反推

- $n \leq 30$ , 指数级别
  - dfs+剪枝, 状压DP, 指数型枚举 (二进制)
- $n \leq 100$   $O(n^3)$ 
  - floyd, dp, 高斯消元
- $n \leq 1000$   $O(n^2), O(n^2 \log n)$ 
  - 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, map(超过 $4 \cdot 1e5$ 就别用了)
  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，后面可以直接用来判是否为素数
                        // n只会被最小质因子筛掉

void get_primes(int n)
{
    memset(st, 0, sizeof st);
    st[0] = st[1] = 1;
    for (int i = 2; i <= n; i++) // 不要忘记等号
    {
        if (!st[i]) primes[cnt++] = i;
        for (int j = 1; 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;
            // primes[j] 一定是i的最小质因子，也一定是primes[j]*i的最小质因子
        }
    }
}
```

## 大范围区间筛素数

## 线性筛加速质因数分解

```
// Author fishingrod
// CSDN: https://blog.csdn.net/qq\_39354847
#pragma GCC optimize(2)
#pragma GCC optimize(3)
#pragma GCC optimize("Ofast")
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define int long long

ll T, Q, n, m, k, p, ans, cnt, sum, tmp, num, last;
map<int, int> prime;
int pri[N], st[N]; // st标记同时预处理除i对应最小质因子

inline void get_primes(int n)
{
    memset(st, 0, sizeof st);
    st[0] = st[1] = 1;
    for (int i = 2; i <= n; i++)
    {
        if (!st[i])
        {
            pri[cnt++] = i;
            st[i] = i;
        }
        for (int j = 1; pri[j] <= n / i; j++)
        {
            st[pri[j] * i] = pri[j];
        }
    }
}
```

```

        if (i % pri[j] == 0) break;
    }
}
}
inline void divide(int x)
{
    prime.clear();
    while(x>1)
    {
        prime[st[x]]++;
        x/=st[x];
    }
}

signed main()
{
    n=scanf("%d",&n);
    get_primes(n);
    for(int i=2; i<=n; i++)divide(i);
    return 0;
}

```

### 线性筛预处理质因子个数

```

#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
ll T;
const int N=2e7+10;
ll pr[N],m[N];
bool st[N];
ll cnt;
void get_prime()
{
    for(ll i=2;i<=N;i++)
    {
        if(st[i]==0)
        {
            pr[++cnt]=i;
            m[i]=1;
        }
        for(ll j=1;pr[j]<=N/i;j++)
        {
            st[i*pr[j]]=1;
            m[i*pr[j]]=m[i]+1;
            if(i%pr[j]==0)//i是pr[j]的倍数,所以就不增加了
            {
                m[i*pr[j]]=m[i];
                break;
            }
        }
    }
}

```

# 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;
*/
```

## Pollard-rho大数分解法 $O(N^{1/4})$

```
#include <iostream>
#include <time.h>
#include <algorithm>
#include <stdio.h>

typedef long long LL;

using namespace std;

const int times = 20;
LL fac[1001];
int cnt;

LL mul(LL a,LL b,LL mod){
    LL ans = 0;
    while (b){
        if (b & 1){
            ans = (ans + a) % mod;
        }
        a = (a<<1) % mod;
        b >>= 1;
    }
    return ans;
}

LL pow(LL a,LL b,LL mod){
    LL ans = 1;
    while (b){

```

```

        if (b & 1){
            ans = mul(ans,a,mod);
        }
        b >>= 1;
        a = mul(a,a,mod);
    }
    return ans;
}

bool witness(LL a,LL n){
    LL temp = n - 1;
    int j = 0;
    while (temp % 2 == 0){ // 其实就是得到 m
        j++;
        temp /= 2;
    }
    LL x = pow(a,temp,n);
    if (x == 1 || x == n-1){ // 判断 $a^m$ 
        return true;
    }
    while (j--){
        x = mul(x,x,n); // 进一步判断  $a^{(2m)}$   $a^{(4m)}$  ...
        if (x == n-1)
            return true;
    }
    return false;
}

bool miller_rabin(LL n){
    if (n == 2){ // 如果是2肯定是素数
        return true;
    }
    if (n<2 || n % 2 == 0){ //如果小于2或者是大于2的偶数肯定不是素数
        return false;
    }
    for (int i=0;i<times;i++){ //随机化检验
        LL a = rand() % (n-1) + 1;
        if (!witness(a,n))
            return false;
    }
    return true;
}

LL gcd(LL a,LL b){ // 这里的gcd和一般的gcd不一样
    if (a == 0){ // pollard_rho的需要
        return 1;
    }
    if (a < 0){ // 可能有负数
        return gcd(-a,b);
    }
    while (b){
        LL t = a % b;
        a = b;
        b = t;
    }
    return a;
}

```

```

LL pollard_rho(LL n, LL c){ // 找因子
    LL i = 1, k = 2; // 用来判断是否成环
    LL xx = rand() % n, y = xx;
    while (1){
        i++;
        xx = (mul(xx, xx, n) + c) % n;
        LL d = gcd(y-xx, n);
        if (1 < d && d < n){ // 找到一个因数
            return d;
        }
        if (y == xx){ // 出现循环，那么查找失败
            return n;
        }
        if (i == k){ // 相当一个优化？
            y = xx;
            k <<= 1;
        }
    }
}

void find(LL n){ // 通过因数来找质因子
    if (miller_rabin(n)){
        fac[cnt++] = n; // 记录质因子
        return ;
    }
    LL p = n;
    while (p >= n)
        p = pollard_rho(p, rand() % (n-1) + 1); // 如果转了一圈还是p那么继续
    find(p);
    find(n/p);
}

int main(){
    srand(time(NULL));
    int t;
    scanf("%d", &t);
    while (t--){
        LL x;
        scanf("%lld", &x);
        if (miller_rabin(x)){
            printf("Prime\n");
            continue;
        }
        cnt = 0;
        find(x);
        for(int i=0; i<cnt; i++) printf("%lld\n", fac[i]);
    }
    return 0;
}

```

### 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++)//从1开始，约数啊
        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-->0)
    {
        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

### 欧拉函数

```

11 phi(11 x)
{
    11 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*primes[j]]=1;
            if(i%primes[j]==0)
            {
                phi[primes[j]*i]=phi[i]*primes[j];
                break;
            }
            phi(primes[j]*i)=phi[i]*(primes[j]-1)
        }
    }
    11 res=0;
    for(int i=1;i<=n;i++)res+=phi[i];
    return res;
}

int main()
{
    11 n;
    cin>>n;
}

```

```

        cout<<get_eulers(n)<<endl;

        return 0;
    }

```

## 费马小定理

```

template<typename T>
const T MOD = 1e9 + 7;
T Q_Power(T a , T b ){
    T res = 1;
    while(b){
        if(b & 1) res = (res * a) % MOD;
        b >>= 1;
        a = (a * a) % MOD;
    }
}

T x_1(T a){ return Q_Power(a,MOD - 2) ; } //逆元

```

费马小定理主要用于求解在取模为素数的情况下，对于式子  $a \times x \equiv 1 \pmod{MOD}$  求解  $a$  的逆元  $x$ ，与上述扩展欧几里得算法拥有相当的时间复杂度，但却更好写些。

## 整除分块

以下为例题的解法

求:  $\sum_{i=1}^n \lfloor \frac{n}{i} \rfloor$   
(待更新)

```

for(int l = 1 , r ; l <= n ; l = r + 1){
    r = n / (n / l);
    ans += (r - l + 1) * (n / l);
}

```

## 组合数

### 递推

```

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;

```

### 预处理逆元求组合数

首先预处理出所有阶乘取模的余数`fact[N]`，以及所有阶乘取模的逆元`infact[N]`  
如果取模的数是质数，可以用费马小定理求逆元

```

int qmi(int a, int k, int p)    // 快速幂模板
{
    int res = 1;
    while (k)
    {
        if (k & 1) res = (LL)res * a % p;

```

```

        a = (LL)a * a % p;
        k >>= 1;
    }
    return res;
}

// 预处理阶乘的余数和阶乘逆元的余数
fact[0] = infact[0] = 1;
for (int i = 1; i < N; i ++ )
{
    fact[i] = (LL)fact[i - 1] * i % mod;
    infact[i] = (LL)infact[i - 1] * qmi(i, mod - 2, mod) % mod;
}

```

## Lucas定理

若 $p$ 是质数，则对于任意整数  $1 \leq m \leq n$ ，有：

$$C(n, m) = C(n \% p, m \% p) * C(n / p, m / p) \pmod{p}$$

```

int qmi(int a, int k, int p) // 快速幂模板
{
    int res = 1 % p;
    while (k)
    {
        if (k & 1) res = (LL)res * a % p;
        a = (LL)a * a % p;
        k >>= 1;
    }
    return res;
}

int C(int a, int b, int p) // 通过定理求组合数C(a, b)
{
    if (a < b) return 0;

    LL x = 1, y = 1; // x是分子, y是分母
    for (int i = a, j = 1; j <= b; i --, j ++ )
    {
        x = (LL)x * i % p;
        y = (LL)y * j % p;
    }

    return x * (LL)qmi(y, p - 2, p) % p;
}

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;
}

```

## 分解质因数法求组合数

当我们要求出组合数的真实值，而非对某个数的余数时，分解质因数的方式比较好用：

1. 筛法求出范围内的所有质数

2. 通过  $C(a, b) = a! / b! / (a - b)!$  这个公式求出每个质因子的次数。  $n!$  中  $p$  的次数是  $n / p + n / p^2 + n / p^3 + \dots$

3. 用高精度乘法将所有质因子相乘

```
int primes[N], cnt;      // 存储所有质数
int sum[N];             // 存储每个质数的次数
bool st[N];             // 存储每个数是否已被筛掉

void get_primes(int n)    // 线性筛法求素数
{
    for (int i = 2; i <= n; i++)
    {
        if (!st[i]) primes[cnt++] = i;
        for (int j = 0; primes[j] <= n / i; j++)
        {
            st[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}

int get(int n, int p)    // 求n! 中的次数
{
    int res = 0;
    while (n)
    {
        res += n / p;
        n /= p;
    }
    return res;
}

vector<int> mul(vector<int> a, int b)    // 高精度乘低精度模板
{
    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;
}

get_primes(a); // 预处理范围内的所有质数

for (int i = 0; i < cnt; i++) // 求每个质因数的次数
```

```

{
    int p = primes[i];
    sum[i] = get(a, p) - get(b, p) - get(a - b, p);
}

vector<int> res;
res.push_back(1);

for (int i = 0; i < cnt; i ++ )    // 用高精度乘法将所有质因子相乘
    for (int j = 0; j < sum[i]; j ++ )
        res = mul(res, primes[i]);

```

## Stirling数

### 第一类

### 第二类

用于求解n个不同小球分为m相同堆，且不存在空的方案数

```

for(int i=1;i<=n;i++){
    stir[i][i]=1;
    stir[i][0]=0;
}
for(int i=2;i<=n;i++)
    for(int j=1;j<=m;j++)
        stir[i][j]=stir[i-1][j]*j+stir[i-1][j-1];

```

### Bell数

Bell (n) 为基数为n的集合划分方案数(非空)，他是第二类stirling数之和

$$Bell(n) = \sum_{k=1}^n Stirling(n, k)$$

### 错排

```

LL D[N]={1,0};//0是1, 1是0别写错了
for(int i=2;i<=N;i++)D[i]=(i-1)*(D[i-1]+D[i-2])%mod;

```

## 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;
}

```

## 斐波那契数列

矩阵快速幂

```

11 Fib_Q_M_Power(11 n){
    if(n <= 1) return n;
    if(n == 2) return 1;
    11 ans[2][2] = {1,0,0,1}, tmp[2][2] , a[2][2] = {1,1,1,0};
    while(n){
        if(n & 1){
            memset(tmp,0,sizeof(tmp));
            for(int i = 0 ; i < 2 ; i ++ ){
                for(int j = 0 ; j < 2 ; j ++ ){
                    for(int k = 0 ; k < 2 ; k ++ ){
                        tmp[i][j] = (tmp[i][j] + ans[i][k] * a[k][j] % MOD) %
MOD;
                    }
                }
            }

            for(int i = 0 ; i < 2 ; i ++ ){
                for(int j = 0 ; j < 2 ; j ++ ) ans[i][j] = tmp[i][j];
            }

        }

        n >>= 1;
        memset(tmp,0,sizeof(tmp));
        for(int i = 0 ; i < 2 ; i ++ ){
            for(int j = 0 ; j < 2 ; j ++ ){
                for(int k = 0 ; k < 2 ; k ++ ){
                    tmp[i][j] = (tmp[i][j] + a[i][k] * a[k][j] % MOD) % MOD;
                }
            }
        }

        for(int i = 0 ; i < 2 ; i ++ ){
            for(int j = 0 ; j < 2 ; j ++ ){
                a[i][j] = tmp[i][j];
            }
        }
    }
}

```



```
    return ans[1][0];  
}
```

## 扩展欧几里得

扩欧求解同余方程 $ax+by=1$ （要求解最小正整数）

```
#include<bits/stdc++.h>  
using namespace std;  
  
typedef long long LL;  
  
int exgcd(int a,int b ,int &x,int &y)  
{  
    if(!b)  
    {  
        x=1,y=0;  
        return a;  
    }  
  
    int d=exgcd(b,a%b,y,x);  
    y-=a/b*x;  
    return d;  
}  
  
int get_mod(int a,int b)  
{  
    return (a%b+b)%b;  
}  
  
int main()  
{  
    int a,b;  
    cin>>a>>b;  
    int x,y;  
    exgcd(a,b,x,y);  
    cout<<get_mod(x,b)<<endl;  
    return 0;  
}
```

求解线性同余方程 $ax+my=b$ （保证解在int范围内）

```
#include<bits/stdc++.h>  
using namespace std;  
  
typedef long long LL;  
int exgcd(int a,int b,int &x,int &y)  
{  
    if(!b)  
    {  
        x=1,y=0;  
        return a;  
    }  
  
    int d=exgcd(b,a%b,y,x);  
    y-=a/b*x;  
    return d;  
}
```

```

int main()
{
    int n;
    cin>>n;
    while(n-->0)
    {
        int a,b,m,x,y;
        cin>>a>>b>>m;
        //即求解ax+my=b
        int d=exgcd(a,m,x,y);
        if(b%d)puts("impossible");//gcd(a,b)|b看有没有解
        else cout<<(LL)x*b/d%m<<endl;
    }
    return 0;
}

```

## 小结

$x_1, y_1$  为  $ax + by = c$  的特解

那么他的最小正整数解为

$d = \gcd(a, b), p = \text{abs}(\frac{b}{d})$  防止出现负数

$x = ((x + p) \% p) \% p$ , 如果  $x = 0$  那么  $x = p$

输出int范围内的解

$x = x_1 \% b$

## 博弈论

**SG函数** (以NIM集合博弈为例题，每次可以拿走集合里的数)

```

#include<bits/stdc++.h>
using namespace std;
const int N=110,M=10010;
int n,m;
int s[N],f[M];
int sg(int x){//这是一棵搜索树
    if(f[x]!=-1)return f[x];//记忆化搜索 保证时间复杂度不是指数级别 每个状态只算一次 记忆化搜索是对搜索的一种优化，也是动态规划的一种实现方式
    unordered_set<int>S;//所有可以到的局面，函数内部声明，后面递归出来的和本次的S是不一样的
    for(int i=0;i<m;++i){
        int sum=s[i]; //当前的数的个数是s[i]
        if(x>=sum) S.insert(sg(x-sum)); //当前的数的个数大于sum 才能把它加进来 从x中取走s[i]个石子，要递归下去啊！
    }
    for(int i=0;;++i){//不属于当前集合的最小自然数 MEX
        if(!S.count(i)) return f[x]=i;
    }
}

int main(){
    cin>>m;
    for(int i=0;i<m;++i)cin>>s[i];
    cin>>n;
    memset(f,-1,sizeof(f));//初始化
    int res=0;
    for(int i=0;i<n;++i){

```

```

int x;
cin>>x; //对每一堆求sg然后异或起来
res^=sg(x);
}
if(res)puts("Yes");
else puts("No");
}

```

## 常见博弈结论

### 1. 尼姆游戏 Nim Game

给定  $N$  堆物品，第  $i$  堆物品有  $A_i$  个。两名玩家轮流行动，每次可以任选一堆，取走任意多个物品，可把一堆取光，但不能不取。**取走最后一件物品者获胜**。两人都采取最优策略，问先手是否必胜。

结论:异或和为0，先手必败，反之先手必胜

### 2. 巴什博弈 Bash Game

有 1 堆石子，总个数是  $n$ ，两名玩家轮流在石子堆中拿石子，每次至少取 1 个，至多取  $m$  个。**取走最后一个石子的玩家为胜者**。判定先手和后手谁胜。

结论：若  $(m+1) \mid n$ （整除）则先手必败，否则先手必胜

### 3. 威佐夫博弈 Wythoff Game

有两堆石子，石子数可以不同。两人轮流取石子，每次可以在一堆中取，或者从两堆中取走相同个数的石子，数量不限，**取走最后一个石头的人获胜**。判定先手是否必胜。

结论：假设两堆石子为  $(a, b)$  其中  $(a < b)$ ，当且仅当  $(b - a) * \frac{\sqrt{5}+1}{2} = a$ ，反之必胜

### 4. 斐波那契博弈 Fibonacci Game

有一堆个数为  $n$  ( $n \geq 2$ ) 的石子，游戏双方轮流取石子，规则如下：

先手不能在第一次把所有的石子取完，至少取 1 颗；

之后每次可以取的石子数至少为 1，至多为对手刚取的石子数的 2 倍。

约定**取走最后一个石子的人为赢家**，求必败态。

结论:先手必败，当且仅当石子数为斐波那契数

（有一个定理，任何正整数可以表示为若干个不连续斐波那契数之和）

## 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;
}
```

## 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（二进制每位数字对应一种状态），表示一个集合，i 枚举 s 的所有子集
//s=9=1001,有子集9=1001,8=1000,1=0001
for (int i = s; i; i = s & i - 1)
//[0,2^n-1], n个元素的子集有2^n个
for(int i = 0;i<(1<<n);i++)
```

## 枚举方案

```
void binary_enum(int n)
{
    for(int i=0;i<(1<<n);i++)
        for(int j=0;j<n;j++)
        {
            if(i&(1<<j))
            {
                array[j]//别写成array[i]
            }
        }
}
```

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  
 $a \wedge b$  按位异或//可以用来排除出现偶数次的数  
 $\sim a$  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:  $x \mid= 1 \ll i$   
 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:  $(x \wedge x \gg 31) - (x \gg 31)$  (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:  $x \neq -1, x \wedge -1, x + 1, \sim x$

异或, 加法恒等式

$a+b=a \mid b+2(a \& b)$  若  $a+b=a \mid b \rightarrow a \& b=0$

若  $a-b=a \mid b \rightarrow a-b$  每一位不能发生借位, 即若 a 已知, 对于 a 的每一位, 若为 1, 则可以 0 1 若为 0, 则指南 0

## 连续自然数异或前缀和O(1)

```
LL f(LL x)
{
    if(x%4==1) return 1;
    if(x%4==2) return x+1;
    if(x%4==3) return 0;
    if(x%4==0) return x;
}
```

## 等差数列异或和

```
#include<cstdio>
#include<algorithm>
#include<iostream>
#include<cstring>
using namespace std;
typedef long long LL;
LL x,y,z;//x是首项 y是不能超过y z是公差
LL f(LL a,LL b,LL c,LL n)//首项 公差 c 项数
{
    if (n==0) return 0;
    LL ans=(b/c)*n*(n-1)/2+(a/c)*n;
    ans=ans+f((b*n+a)%c,c,b%c,((b%c)*n+(a%c))/c);
    return ans;
}
int main()
{
    while (scanf("%I64d%I64d%I64d",&x,&y,&z)!=EOF)
    {
        LL ans=0;
        for (LL u=0;u<32;u++)
            ans=ans|((f(x,z,(1LL<<u),(y-x)/z+1)&1)<<u);
        printf("%I64d\n",ans);
    }
    return 0;
}
```

## 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 $\log_2(x)$  (二分)  $O(\log \log x)$

```
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 $\log_2(x)$  1~n中所有数  $O(n)$

```
int log_2[N]; // 存  $\log_2(i)$  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( $a*b\%p$ , 主要防爆)

```

//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;
}

```

## 快速幂

```

//如果mod 1e18级别, 改成加法防爆
int quick(int a,int b)
{
    int res=1;
    a=a%mod;
    while(b)
    {
        if(b&1) res=(res*a)%mod;//改成+
        a=(a*a)%mod;//改成+
        b>>=1;
    }
    return res;
}

```

## 高精

vector开O2优化才能和数组媲美

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

**归并排序+求逆序对**（交换重排，根据奇偶性判局面可达（可能是字符串，二维平面，数组序列））

```

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

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

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

//i不会等于1只有1
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;//i是1别打成l
    while(i<=mid&&j<=r)
    {
        if(q[i]<=q[j])tmp[k++]=q[i++];
        else
        {
            tmp[k++]=q[j++];
            res+=mid-i+1;//q[i]>q[j]，左区间剩下的所有数与右区间当前数成为逆序对
        }
    }
    while(i<=mid)tmp[k++]=q[i++];    //扫尾
}

```

```

while(j<=r)tmp[k++]=q[j++];
for(int i=1,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);
}

```

## 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);
    cin.tie(0);
}

```

### \_\_int128 (GCC)

```

inline __int128 read(){
    __int128 x = 0, f = 1;
    char ch = getchar();
    while(ch < '0' || ch > '9'){
        if(ch == '-')
            f = -1;
        ch = getchar();
    }
    while(ch >= '0' && ch <= '9'){
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x * f;
}

inline void print(__int128 x){
    if(x < 0){

```

```

        putchar('-');
        x = -x;
    }
    if(x > 9)
        print(x / 10);
    putchar(x % 10 + '0');
}

```

## 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

### 表达式

#### 后缀表达式

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

const int N=1e5+10;
stack<int> num;
stack<char> op;

void eval()
{
    auto b=num.top();num.pop();
    auto a=num.top();num.pop();
    auto c=op.top();op.pop();
    int x;
    if(c=='+')x=a+b;
    else if(c=='-')x=a-b;
    else if(c=='*')x=a*b;
    else x=a/b;
    num.push(x);
}

int main()
{
    unordered_map<char,int>pr{{'+',1},{ '-',1},{ '*',2},{ '/',2}}; //运算符优先级
    string s;
    cin>>s;
    for(int i=0;i<s.size();i++)
    {
        auto c=s[i];
        if(isdigit(c))//转数字
        {
            int x=0,j=i;
            while(j<s.size()&&isdigit(s[j]))
                x=x*10+s[j++]-'0';
            i=j-1;
            num.push(x);
        }
        else if(c=='(')op.push(c);
        else if(c==')')//括号直接算
        {
```

```

        while(op.top()!='(')eval();
        op.pop();
    }
    else
    {
        while(op.size() && pr[op.top()] >= pr[c])eval(); //如果当前优先级比前面的低就
算
        op.push(c); //操作符入栈
    }
}
while(op.size())eval(); //剩余计算
printf("%d", num.top());
return 0;
}

```

## 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

### 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;
}

/*
求区间长度S~T的区间最小值
int l=1,r=0;
for(int i=s;i<=n;i++){
    while(l<=r&&sum[q[r]]>=sum[i-s])r--;//发现降序，弹出队尾
    q[++r]=i-s;
    while(l<=r&&q[l]<i-t)l++;//超出长度了弹出队首
    if(l<=r&&sum[i]-sum[q[l]]>=0)return 1;//这行根据题目写
}

*/

```

## 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){ return f[x] == x ? x : find(f[x]) ; }

void merge_set(int x,int y){
    int fx = find(x) , fy = find(y);
    if(fx!=fy) siz[fy]+=siz[fx] , f[fx]=fy; // 联通的大小。
}

void init(){ for(int i = 1 ; i <= n ; i ++ ) f[i] = i , siz[i] = 1 ; }

```

### 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)];//要在合并之前
            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",&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;
}

```

## 哈希表

### 字符串hash

```

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

typedef unsigned long long ULL;
const int P=131,N=1e5+10;

char str[N];
ULL p[N],has[N];
int n,m;

ULL get_hash(int l,int r)
{
    if(l>r||l<1||r>n)return 0;
    return has[r]-has[l-1]*p[r-l+1];
}

int main()
{
    scanf("%d%d",&n,&m);
    scanf("%s",str+1);
    p[0]=1;
    for(int i=1;i<=n;i++)
    {
        has[i]=has[i-1]*P+str[i];
        p[i]=p[i-1]*P;
    }
}

```

```

int l1,r1,l2,r2;

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

```

## ST (区间查询)

```

#include<bits/stdc++.h>
using namespace std;
const int N=2e5+10;
int a[N];
int f[N][50];
int n,m;

void ST_prework()
{
    for(int i=1;i<=n;i++)f[i][0]=a[i];//递推边界
    int t=log(n)/log(2)+1;
    for(int j=1;j<t;j++)//先区间长度
        for(int i=1;i<=n-(1<<j)+1;i++)//然后遍历起点
            f[i][j]=max(f[i][j-1],f[i+(1<<(j-1))][j-1]);
}

int ST_query(int l,int r)
{
    int k=log(r-l+1)/log(2);
    return max(f[l][k],f[r-(1<<k)+1][k]);
}

int main()
{
    scanf("%d",&n);
    for(int i=1;i<=n;i++)scanf("%d",&a[i]);
    ST_prework();
    scanf("%d",&m);
    while(m--)
    {
        int l,r;
        scanf("%d%d",&l,&r);
        printf("%d\n",ST_query(l,r));
    }

    return 0;
}

```

## Multiset实现带修改的最值查询

# 线段树

## 查找

### 二分

#### 二分的应用

1. 数据范围大，数据量小，把数据存到普通数组，排序，二分查找下标，可以得到区间数据数量
2. 最大求最小，最小求最大
3. 区间gcd具有单调性，最大平均数，中位数，分数规划

#### 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，即右半段
{
    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，即左半段
{
    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 1;
    }

    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;
}

```

```

#include<bits/stdc++.h>
using namespace std;
const double eps=1e-7;
double y;
double f(double x)
{
    return 0.0001*x*x*x*x*x+0.003*x*x*x+0.5*x-3;
}
int main()
{
    while(scanf("%lf",&y)!=EOF)
    {
        double mid;
        double l=-20.0,r=20.0;
        while(l<=r)
        {
            mid=(l+r)/2.0;

```



```

        if(fabs(f(mid)-y)<1e-5)break;//如果直接数值型的可以这样处理保证精度
        if(f(mid)<y)l=mid;
        else r=mid;
    }
    printf("%.4lf\n",mid);
}
return 0;
}

```

## 整数三分（凸函数）

```

LL check(LL mid)//带入函数
{
    return ;
}
void solve()
{
    cin>>x>>y;
    LL l=0,r=min(x/2,y/3);
    while(l<r)
    {
        LL lmid=l+(r-l)/3;
        LL rmid=r-(r-l)/3;
        if(check(lmid)>check(rmid))r=rmid-1;
        else l=lmid+1;
    }
    cout<<check(l)<<endl;
}

```

## 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，双向图开双倍2*N!
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

用一个二维数组表示所建的图。通常查询修改的复杂度为  $O(1)$  十分方便快捷，但由于其空间复杂度为  $O(n^2)$ ，二维数组过大的时候会采用邻接表。

一般无向图相关于主对角线对称，主对角线为 0

有向带权图，使用邻接矩阵会好很多。

## DFS

### 有多少个连通块(Flood Fill算法)

```
#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;
}
```

求联通块大小的题目若涉及到查询次数较多的题目，用并查集为更优选项。

### 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为根子树大小一部分
        }
    }
    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<=l&&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, 这个不要忘记了
}

```

```

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DAG）

```

#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;
}

```

## 最短路

### dijkstra朴素稠密图O (n^2)

```

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

const int N=510;
int n,m;
int g[N][N]; //邻接矩阵处理稠密图
int dist[N];
bool st[N];
int dijkstra()
{
    memset(dist,0x3f,sizeof(dist)); //距离初始化为正无穷
    memset(st,0,sizeof st);
    dist[1]=0; //一号点初始化为0

    for(int i=0;i<n;i++) //迭代n次
    {
        int t=-1; //t开始为-1表示还没确定最短路
        for(int j=1;j<=n;j++)
            if(!st[j]&&(t==-1||dist[t]>dist[j])) //所有st[j]=0的点中找到距离最小的点
                t=j;
        st[t]=1;
    }
}

```

```

        for(int j=1;j<=n;j++)//用t更新其他点到1的距离，遍历边有效更新m次
            dist[j]=min(dist[j],dist[t]+g[t][j]);
    }

    if(dist[n]==0x3f3f3f3f)return -1;
    return dist[n];
}
int main()
{
    scanf("%d%d",&n,&m);

    memset(g,0x3f,sizeof(g));//初始化点位无穷

    while(m--)
    {
        int a,b,c;
        scanf("%d%d%d",&a,&b,&c);
        g[a][b]=min(g[a][b],c);//处理重边保留距离最短的即可
    }

    int t=dijkstra();

    printf("%d\n",t);

    return 0;
}

```

### dijkstra堆优化稀疏图 $O(m\log n)$

```

#include<bits/stdc++.h>
using namespace std;
typedef pair<int,int> PII;
int n,m,s;
const int N=5e5+10;//邻接表N看边数啊啊啊
int h[N],e[N],ne[N],w[N],idx;
int dist[N];
bool vis[N];
void add(int a,int b,int c)
{
    e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
}
void dijkstra(int st)
{
    memset(dist,0x3f,sizeof dist);//你开0x3f就要对于int的数组，不要用long long数组开
    memset 0x3f
    memset(vis,0,sizeof vis);
    dist[st]=0;
    priority_queue<PII,vector<PII>,greater<PII>>heap;//小根堆，顺序不能换，因为pair按
    first排序
    heap.push({0,st});//first距离，second编号
    while(heap.size())
    {
        //维护当前未被st标记且离源点最近的点
        auto t=heap.top();
    }
}

```

```

        heap.pop();
        int ver=t.second,distance=t.first;
        if(vis[ver])continue;
        vis[ver]=1;
        for(int i=h[ver];i!=-1;i=ne[i])//用t更新其他点
        {
            int j=e[i];
            if(dist[j]>distance+w[i])
            {
                dist[j]=distance+w[i];//松弛
                heap.push({dist[j],j});
            }
        }
    }
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n>>m>>s;
    memset(h,-1,sizeof h);
    while(m--)
    {
        int a,b,c;
        cin>>a>>b>>c;
        add(a,b,c);
    }
    dijkstra(s);
    for(int i=1;i<=n;i++)
    {
        if(dist[i]!=0x3f3f3f3f)cout<<dist[i]<<" ";
        else cout<<"2147483647 ";
    }
    return 0;
}

```

### dijkstra反向建图求多个点到起点的最短路

```

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

const int N=1e4+10,M=2e5+10;
int h[N],e[M],w[M],ne[M],idx;
int n,m;
int dist[N];
bool vis[N];
typedef pair<int,int> PII;
void add(int a,int b,int c)
{
    e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
}
void dijkstra(int st)
{
    memset(dist,0x3f,sizeof dist);
    memset(vis,0,sizeof vis);
    dist[st]=0;
    priority_queue<PII,vector<PII>,greater<PII>>heap;
}

```

```

heap.push({0,st});
while(heap.size())
{
    auto t=heap.top();
    int ver=t.second,distance=t.first;
    heap.pop();
    if(vis[ver])continue;
    vis[ver]=1;
    for(int i=h[ver];i!=-1;i=ne[i])
    {
        int j=e[i];
        if(dist[j]>distance+w[i])
        {
            dist[j]=distance+w[i];
            heap.push({dist[j],j});
        }
    }
}

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin>>n>>m;
    memset(h,-1,sizeof h);
    while(m--)
    {
        int a,b,c;
        cin>>a>>b>>c;
        add(a,b,c);
        add(b+n,a+n,c);
    }
    dijkstra(1);
    int res=0;
    for(int i=2;i<=n;i++)res+=dist[i];
    dijkstra(n+1);
    for(int i=n+2;i<=2*n;i++)res+=dist[i];
    cout<<res;
}

```

### bellman\_ford()(处理边数限制)

```

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

const int N=510,M=10010;

int n,m,k;
int dist[N],backup[N];

struct Edge

```

```

{
    int a,b,w;
}edges[M];

int bellman_ford()
{
    memset(dist,0x3f,sizeof dist);
    dist[1]=0;//初始化
    for(int i=0;i<k;i++)
    {
        memcpy(backup,dist,sizeof dist);//防止串联更新
        for(int j=0;j<m;j++)
        {
            int a=edges[j].a,b=edges[j].b,w=edges[j].w;
            dist[b]=min(dist[b],backup[a]+w);//用备份更新
        }
    }
    if(dist[n]>0x3f3f3f3f/2)return -1;//
    return dist[n];
}

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n>>m>>k;

    for(int i=0;i<m;i++)
    {
        int a,b,w;
        cin>>a>>b>>w;
        edges[i]={a,b,w};
    }
    int t=bellman_ford();
    if(t==-1)puts("impossible");
    else cout<<t<<endl;
    return 0;
}

```

## SPFA

```

#include<bits/stdc++.h>
using namespace std;
const int N=1e5+10;
int h[N],e[N],w[N],ne[N],idx;
bool st[N];
int dist[N];
int n,m;
void add(int a,int b,int c)
{
    e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
}

int spfa()
{
    memset(dist,0x3f,sizeof dist);

```

```

dist[1]=0;
queue<int>q;
q.push(1);
st[1]=1;
while(q.size())
{
    int t=q.front();
    q.pop();
    st[t]=0;
    for(int i=h[t];i!=-1;i=ne[i])
    {
        int j=e[i];
        if(dist[j]>dist[t]+w[i])
        {
            dist[j]=dist[t]+w[i];
            if(!st[j])
            {
                q.push(j);
                st[j]=1;
            }
        }
    }
}
if(dist[n]>0x3f3f3f/2)return -1;
return dist[n];
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n>>m;
    memset(h,-1,sizeof h);
    while(m--)
    {
        int a,b,c;
        cin>>a>>b>>c;
        add(a,b,c);
    }
    int t=spfa();
    if(t==-1)puts("impossible");
    else cout<<t<<endl;
    return 0;
}

```

### SPFA (判有无负权环)

```

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

const int N=2010,M=10010;
int h[N],e[M],ne[M],w[M],idx;
int dist[N],cnt[N];
bool st[N];
int n,m;
void add(int a,int b,int c)
{

```

```

        e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
    }
    bool spfa()
    {
        queue<int>q;
        for(int i=1;i<=n;i++)q.push(i),st[i]=1;
        while(q.size())
        {
            int t=q.front();
            q.pop();
            st[t]=0;
            for(int i=h[t];i!=-1;i=ne[i])
            {
                int j=e[i];
                if(dist[j]>dist[t]+w[i])
                {
                    dist[j]=dist[t]+w[i];
                    cnt[j]=cnt[t]+1;//不要写++cnt[j], 重边会影响的
                    if(cnt[j]>=n)return 1;
                    if(!st[j])
                    {
                        q.push(j);
                        st[j]=1;
                    }
                }
            }
        }
        return 0;
    }
    int main()
    {
        ios::sync_with_stdio(0);
        cin.tie(0);
        cin>>n>>m;
        memset(h,-1,sizeof h);
        while(m--)
        {
            int a,b,c;
            cin>>a>>b>>c;
            add(a,b,c);
        }
        if(spfa())cout<<"Yes"<<endl;
        else cout<<"No"<<endl;
        return 0;
    }
}

```

## Floyd(多源汇最短路)

```

#include<bits/stdc++.h>
using namespace std;
int n,m,q;
const int N=210,INF=1e9;
int d[N][N];

void floyd()
{
    for(int k=1;k<=n;k++)//先循环k

```

```

        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                d[i][j]=min(d[i][j],d[i][k]+d[k][j]);
    }
    int main()
    {
        ios::sync_with_stdio(0);
        cin.tie(0);
        cin>>n>>m>>q;
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                if(i==j)d[i][j]=0;//处理自环
                else d[i][j]=INF;

        while(m--)
        {
            int a,b,w;
            cin>>a>>b>>w;
            d[a][b]=min(d[a][b],w);
        }
        floyd();
        while(q--)
        {
            int a,b;
            cin>>a>>b;
            if(d[a][b]>INF/2)cout<<"impossible"<<endl;
            else cout<<d[a][b]<<endl;
        }
        return 0;
    }

```

### Floyd求最短环

```

#include<bits/stdc++.h>
using namespace std;
const int N=200;
typedef long long ll;
const int INF=0x7f7f7f7f;
ll g[N][N],dist[N][N];
int n,m;
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin>>n>>m;
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            if(i!=j)dist[i][j]=g[i][j]=INF;
    while(m--)
    {
        ll a,b,c;
        cin>>a>>b>>c;
        g[a][b]=g[b][a]=min(g[a][b],c);
        dist[a][b]=dist[b][a]=min(g[a][b],c);
    }
    ll ans=INF;

```



```

for(int k=1;k<=n;k++)
{
    for(int i=1;i<k;i++)
        for(int j=i+1;j<k;j++)
            ans=min(ans,dist[i][j]+g[i][k]+g[k][j]);
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
        {
            dist[i][j]=min(dist[i][j],dist[i][k]+dist[k][j]);
            dist[j][i]=dist[i][j];
        }
}
if(ans==INF)cout<<"No solution.";
else cout<<ans;
return 0;
}

```

## 最小生成树

### Kruskal (稀疏图) ( $O(m\log m)$ )

```

#include<bits/stdc++.h>
using namespace std;
const int N=2e6+10;
int n,m;
int f[N];
struct Edge
{
    int a,b,w;
}edges[N];
bool cmp(Edge a,Edge b)
{
    return a.w<b.w;//如果写成if的最后别忘记加return 0
}

int find(int x)
{
    if(f[x]!=x)f[x]=find(f[x]);
    return f[x];
}

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n>>m;
    for(int i=0;i<m;i++)
    {
        int a,b,c;
        cin>>a>>b>>c;
        edges[i]={a,b,c};
    }
    sort(edges,edges+m,cmp);//对边排序
    int res=0,cnt=0;//res最小生成树边权重之和, cnt记录最小生成树中的边数
    //并查集看有没有在集合里
    for(int i=0;i<=m;i++)f[i]=i;
    for(int i=0;i<m;i++)

```

```

{
    int a=edges[i].a,b=edges[i].b,w=edges[i].w;
    a=find(a),b=find(b);
    if(a!=b)
    {
        f[a]=b;
        res+=w;
        cnt++;
    }
}
if(cnt<n-1)cout<<"impossible"<<endl;
else cout<<res<<endl;
return 0;
}

```

### Prim(稠密图)( $O(n^2)$ )

```

#include<bits/stdc++.h>
using namespace std;
const int N=1e3,INF=0x3f3f3f3f;
int dist[N],g[N][N];//dist是顶点到任意一个树顶点的最短距离
bool st[N];
int n,m;
int prim()
{
    memset(dist,0x3f,sizeof dist);
    int res=0;//存最小生成树所有边长度之和
    for(int i=0;i<n;i++)
    {
        int t=-1;
        for(int j=1;j<=n;j++)//找集合外所有点中到集合距离最小的点
            if(!st[j]&&(t==-1||dist[t]>dist[j]))
                t=j;
        if(i&&dist[t]==INF)return INF;//不是第一个点而且最短距离都为INF，就不存在最小生成树
        if(i)res+=dist[t];//只要不是第一个点
        st[t]=1;
        //扫描顶点t的所有边，在以t为中心更新其他点到树的距离（这时候t已经在生成树里了，其他点到t距离就是到生成树距离）
        for(int j=1;j<=n;j++)dist[j]=min(dist[j],g[t][j]);
    }
    return res;
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n>>m;
    memset(g,0x3f,sizeof g);
    while(m--)
    {
        int a,b,c;
        cin>>a>>b>>c;
        g[a][b]=g[b][a]=min(g[a][b],c);
    }
    int t=prim();
    if(t==INF)cout<<"impossible";
}

```

```

    else cout<<t<<endl;
    return 0;
}

```

## 二分图

### 染色法判二分图( $O(m+n)$ )

```

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

const int N=1e6+10,M=2e6+10;

int n,m;
int h[N],e[M],ne[M],idx;
int color[N];

bool dfs(int u,int c)//u为点编号, c为染色
{
    color[u]=c;
    for(int i=h[u];i!=-1;i=ne[i])//遍历和点连接的点
    {
        int j=e[i];
        if(!color[j])//没染色,那就染(3-c实现1染2, 2染1)
        {
            if(!dfs(j,3-c))return 0;
        }
        else if(color[j]==c)return 0;//已经染色
    }
    return 1;
}

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

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n>>m;
    memset(h,-1,sizeof h);
    while(m--)
    {
        int a,b;
        cin>>a>>b;
        add(a,b);
        add(b,a);
    }
    bool flag=1;//染色是否有矛盾发生
    for(int i=1;i<=n;i++)
        if(!color[i])
        {
            if(!dfs(i,1))//dfs false有矛盾发生
            {
                flag=0;
                break;
            }
        }
}

```

```

    }

    if(flag)puts("Yes");
    else puts("No");
}

```

## 匈牙利算法二分图最大匹配图

```

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

const int N=510,M=100010;

int n1,n2,m;
int h[N],e[M],ne[M],idx;
int match[N];
bool st[N];

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

bool find(int x)
{
    for(int i=h[x];i!=-1;i=ne[i])//x是男的，j是妹子，遍历男的看上的所有妹子
    {
        int j=e[i];
        if(!st[j])
        {
            st[j]=1;
            if(match[j]==0||find(match[j]))//妹子没有匹配或者妹子原本匹配的男的有备胎
            {
                match[j]=x;
                return 1;
            }
        }
    }
    return 0;
}

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin>>n1>>n2>>m;
    memset(h,-1,sizeof h);
    while(m--)
    {
        int a,b;
        cin>>a>>b;
        add(a,b);
    }
    int res=0;
    for(int i=1;i<=n1;i++)
    {
        memset(st,0,sizeof st);
        if(find(i))res++;
    }
}

```

```

    }
    cout<<res<<endl;
    return 0;
}

```

## 前缀和/差分

### 前缀和

#### 一维前缀和

$$s[i] = a[1] + a[2] + \dots + a[i]$$

$$a[1] + \dots + a[r] = s[r] - s[1 - 1]$$

#### 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( $x1, y1$ )为左上角, ( $x2, y2$ )为右下角的子矩阵的和为:  
 $s[x2, y2] - s[x1 - 1, y2] - s[x2, y1 - 1] + s[x1 - 1, y1 - 1]$

```

#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;
    }
}

```

## Tip:前缀和和一些注意点(激光炸弹为例)

1. 卡内存，直接累加读入，开const in N的时候别太浪

```
cin>>s[i][j];
s[i][j]+=s[i-1][j]+s[i][j-1]-s[i-1][j-1];
```

2. 卡边界，为了方便处理前缀和，最好把前面的s[0]留出来，所以预处理

```
s[++x][++y]+=w;
//s[x++][y++]+=w; 错误
```

3. 覆盖范围处理(r不出界)

```
r=min(5001,r);
//5001为最大可能边界
```

```
#include<bits/stdc++.h>
using namespace std;
const int N=5e3+10;
typedef long long ll;
int s[N][N];
int n,r,x,y,w;
int main()
{
    cin>>n>>r;
    r=min(5001,r); //预处理半径
    while(n-->0)
    {
        cin>>x>>y>>w;
        s[++x][++y]+=w;
    }
    for(int i=1;i<=5001;i++)
        for(int j=1;j<=5001;j++)
            s[i][j]+=s[i-1][j]+s[i][j-1]-s[i-1][j-1]; //直接累加节省内存

    int res=0;
    //枚举右下角
    for(int i=r;i<=5001;i++) //直接开地图最大
        for(int j=r;j<=5001;j++)
            res=max(res,s[i][j]-s[i][j-r]-s[i-r][j]+s[i-r][j-r]);

    cout<<res;
    return 0;
}
```

## 差分

### 一维差分

```
void init()
{
    memset(b,0,sizeof b);
}
```

```

        for(int i=1;i<=n;i++)b[i]=a[i]-a[i-1];
    }
    void edit(int l,int r,int c)
    {
        //区间预处理非常规情况
        if(l>r)swap(l,r);
        if(l<st)l=st;
        if(r<st)r=st;
        if(l>ed)l=ed;
        if(r>ed)r=ed;
        b[l]+=c;b[r+1]-=c;
    }
    void build()
    {
        for(int i=1;i<=n;i++)a[i]=a[i-1]+b[i];
    }
}

```

```

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

//给区间[l, r]中的每个数加上c: B[l] += c, B[r + 1] -= c
//要确保修改的时候r+1>=l,l和r在区间范围内
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:

$S[x1, y1] += c$ ,  $S[x2 + 1, y1] -= c$ ,  $S[x1, y2 + 1] -= c$ ,  $S[x2 + 1, y2 + 1] += c$

## 例题

```

#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[j]直到开
        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;
}
```

---

## DP

### 参数表

参数	适用题型	状态转移	注意事项
第i个			
时间			
余数			
差值			
最大值	某个属性不能为0		
方向			
坐标	走地图		不一定所有方向都能转移 分类，注意出界问题
剩余使用 次数			
轮廓形状			
路径状态			
区间	合并，染色，两端操作，匹配		
组数	分组问题	$f[i][j] = \max(f[i-1][j], f[u-1][j] + i - u + 1)$	利用while等找到新增一组的合法左边界u
子集和	SOSDP		
已匹配数	老鼠进洞模型		
以某个字符为结尾	dp[i][j],到i个字符为止，以字符j为结尾的满足条件数量		

## 经典模型

### 背包

#### 01背包

有时候可能二进制枚举/DFS更好

### 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
            //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;
}

```

## 数位DP

```
void init() // 根据题意做预处理,一般是组合数/dp。
{
    for(int i = 0; i <= 9; i++) // 对第一位初始化
        f[1][i] = 1;

    // DP过程
}

int dp(int n)
{
    if(!n) return 1;
    vector<int> num;
    // 取出每一位数字,可以根据进制转化问题替换 10
    while(n) num.push_back(n % 10), n /= 10;
    n = num.size();

    LL ans = 0;
    int last = 0;
    for(int i = n - 1; i >= 0; i--)
    {
        int x = num[i];

        // 分类讨论,关注分支(<x,=x)是否存在,操作是否存在
    }

    return ans;
}
```

## 计算几何

### 求解凸包

#### Graham Scan

```
const int maxn = 1e3 + 5;
struct Point {
    double x, y;
    Point(double x = 0, double y = 0):x(x),y(y){}
};
typedef Point Vector;
Point lst[maxn];
int stk[maxn], top;
Vector operator - (Point A, Point B){
    return Vector(A.x-B.x, A.y-B.y);
}
int sgn(double x){
    if(fabs(x) < eps)
        return 0;
    if(x < 0)
        return -1;
    return 1;
}
double Cross(Vector v0, Vector v1) {
```



```

    return v0.x*v1.y - v1.x*v0.y;
}
double Dis(Point p1, Point p2) { //计算 p1p2的 距离
    return sqrt((p2.x-p1.x)*(p2.x-p1.x)+(p2.y-p1.y)*(p2.y-p1.y));
}
bool cmp(Point p1, Point p2) { //极角排序函数 , 角度相同则距离小的在前面
    int tmp = sgn(Cross(p1 - lst[0], p2 - lst[0]));
    if(tmp > 0)
        return true;
    if(tmp == 0 && Dis(lst[0], p1) < Dis(lst[0], p2))
        return true;
    return false;
}
//点的编号0 ~ n - 1
//返回凸包结果stk[0 ~ top - 1]为凸包的编号
void Graham(int n) {
    int k = 0;
    Point p0;
    p0.x = lst[0].x;
    p0.y = lst[0].y;
    for(int i = 1; i < n; ++i) {
        if( (p0.y > lst[i].y) || ((p0.y == lst[i].y) && (p0.x > lst[i].x)) ) {
            p0.x = lst[i].x;
            p0.y = lst[i].y;
            k = i;
        }
    }
    lst[k] = lst[0];
    lst[0] = p0;
    sort(lst + 1, lst + n, cmp);
    if(n == 1) {
        top = 1;
        stk[0] = 0;
        return ;
    }
    if(n == 2) {
        top = 2;
        stk[0] = 0;
        stk[1] = 1;
        return ;
    }
    stk[0] = 0;
    stk[1] = 1;
    top = 2;
    for(int i = 2; i < n; ++i) {
        while(top > 1 && Cross(lst[stk[top - 1]] - lst[stk[top - 2]], lst[i] -
lst[stk[top - 2]]) <= 0)
            --top;
        stk[top] = i;
        ++top;
    }
    return ;
}

```

```

struct Point {
    double x, y;
    Point(double x = 0, double y = 0):x(x),y(y){}
};
typedef Point Vector;
Vector operator - (Point A, Point B){
    return Vector(A.x-B.x, A.y-B.y);
}
bool operator < (const Point& a, const Point& b){
    if(a.x == b.x)
        return a.y < b.y;
    return a.x < b.x;
}
double Cross(Vector v0, Vector v1) {
    return v0.x*v1.y - v1.x*v0.y;
}
//计算凸包，输入点数组为 p，个数为 n， 输出点数组为 ch。函数返回凸包顶点数
//如果不希望凸包的边上有输入点，则把两个 <= 改为 <
//在精度要求高时建议用dcmp比较
//输入不能有重复点，函数执行完后输入点的顺序被破坏
int ConvexHull(Point* p, int n, Point* ch) {
    sort(p, p+n);
    int m = 0;
    for(int i = 0; i < n; ++i) {
        while(m > 1 && Cross(ch[m-1] - ch[m-2], p[i] - ch[m-2]) < 0) {
            m--;
        }
        ch[m++] = p[i];
    }
    int k = m;
    for(int i = n-2; i >= 0; --i) {
        while(m > k && Cross(ch[m-1] - ch[m-2], p[i] - ch[m-2]) < 0) {
            m--;
        }
        ch[m++] = p[i];
    }
    if(n > 1)
        --m;
    return m;
}

```

### 判断点是不是在凸包内

```

ll in(Node a)
{
    if(a*A[1]>0||A[tot]*a>0) return 0;
    ll ps=lower_bound(A+1,A+tot+1,a,cmp2)-A-1;
    return (a-A[ps])*(A[ps%tot+1]-A[ps])<=0;
}

```

### 求多边形面积

```

//!求凸多边形的有向面积
//叉积的几何意义就是三角形有向面积的二倍，所以这里要除以二
double convex_polygon_area(Point* p, int n)

```

```

{
    double area = 0;
    for(int i = 1; i <= n - 2; ++ i)
        area += Cross(p[i] - p[0], p[i + 1] - p[0]);
    return area / 2;
    //return fabs(area / 2); //不加的话求的是有向面积，逆时针为负，顺时针为正
}
//!求非凸多边形的有向面积
//我们又积求得的三角形面积是有向的，在外面的面积可以正负抵消掉，
//因此非凸多边形也适用，可以从任意点出发划分
//可以取原点为起点，减少叉乘次数
double poly_on_area(Point* p, int n)
{
    double area = 0;
    for(int i = 1; i <= n - 2; ++ i)
        area += Cross(p[i] - p[0], p[i + 1] - p[0]);
    return area / 2;
}

```

## 平面最近点对

### 分治法优化

```

#include<cstdio>
#include<algorithm>
#include<cmath>
#define R register
using namespace std;
const int N = 200007;

typedef long long ll;

struct Point {
    ll x, y;
    bool operator < (const Point& t) const {
        return y < t.y;
    }
};

Point v[N], tmp[N];
int n;

ll get_dist(Point A, Point B)
{
    return (ll)(A.x - B.x) * (A.x - B.x) + (A.y - B.y) * (A.y - B.y);
}

ll solve(const int l, const int r){//求平面最近点对的分治算法
    if(l == r) return 0x3f3f3f3f3f;
    if(l + 1 == r) return get_dist(v[l], v[r]); //分治到了一个点对，直接返回答案
    int mid = (l + r) >> 1;
    ll d1 = solve(l, mid), d2 = solve(mid + 1, r);
    ll d = min(d1, d2);
    int tot = 0;
    //先筛选x
    for(int i = l; i <= r; ++ i){

```

```

        if((v[mid].x - v[i].x) * (v[mid].x - v[i].x) <= d)tmp[ ++ tot] = v[i];
    }
    sort(tmp + 1, tmp + tot + 1); //按y排序
    //再筛选y
    for(int i = 1; i <= tot; ++ i){
        for(int j = i + 1; j <= tot; ++ j){
            if((tmp[i].y - tmp[j].y) * (tmp[i].y - tmp[j].y) > d)break; //剪枝优化，不加过不了本题
            d = min(d, get_dist(tmp[i], tmp[j]));
        }
    }
    return d;
}

int main()
{
    scanf("%d", &n);
    for(int i = 1; i <= n; ++ i){
        int x;
        scanf("%d", &x);
        v[i].y = v[i - 1].y + x; //y就是前缀和，我们通过公式推出来的
        v[i].x = i;
    }
    ll minv = solve(1, n);
    printf("%lld\n", minv);
    return 0;
}

```

## 随机优化

@3A17K巨佬的神仙操作！

我们充分发扬人类智慧：

将所有点全部绕原点旋转同一个角度，然后按  $x$  坐标排序

根据数学直觉，在随机旋转后，答案中的两个点在数组中肯定不会离得太远

所以我们只取每个点向后的5个点来计算答案

这样速度快得飞起，在  $n=1000000$  时都可以在1s内卡过

注意旋转 $\theta$ 角时坐标变换

$x' = x \cos \theta - y \sin \theta$

$y' = x \sin \theta + y \cos \theta$

代码如下：

```

#include<cstdio>
#include<cmath>
#include<iostream>
#include<algorithm>
using namespace std;
const int N=2e5+50;
#define D double
struct spot{
    D a[4];
}p[N];
D x,y,x_,y_,z,w,ans;
int n;
bool mmp(const spot &u,const spot &v){
    return u.a[0]<v.a[0];
}

```

```

int main(){
    scanf("%d",&n);
    z=sin(1),w=cos(1);    //旋转1弧度≈57°
    for(int i=1;i<=n;i++){
        scanf("%lf%lf",&x,&y);
        x_=x*w-y*z;
        y_=x*z+y*w;    //计算旋转后的坐标
        p[i].a[0]=x_;
        p[i].a[1]=y_;
        p[i].a[2]=x;
        p[i].a[3]=y;    //存下来
    }
    sort(p+1,p+n+1,mmp);    //排序
    for(int i=n+1;i<=n+10;i++)
    p[i].a[0]=p[i].a[1]=-N-0.01;    //边界处理
    ans=2e9+0.01;    //初始化答案
    for(int i=1;i<=n;i++)
    for(int j=1;j<=5;j++){    //枚举
        x=p[i].a[2];y=p[i].a[3];
        x_=p[i+j].a[2];y_=p[i+j].a[3];
        z=sqrt((x-x_)*(x-x_)+(y-y_)*(y-y_));    //计算距离
        if(ans>z)ans=z;    //更新答案
    }
    printf("%.4lf\n",ans);    //输出
}

```

## 半平面交

//半平面交一般是一个凸多边形，但是有时候会是一个无界多边形

//甚至会是一个直线、线段、点，但是结果一定是凸的

//点p在有向直线L的左边（线上的不算）（叉积大于0a在b的左侧，小于0在右侧[sin夹角]）

```
bool on_left(Line L, Point P){return Cross(L.v, P - L.P) > 0;}
```

//两个有向直线的交点/假定交点唯一存在

```
Point get_intersection(Line a, Line b)
```

```
{
    Vector u = a.P - b.P;
    double t = Cross(b.v, u) / Cross(a.v, b.v);
    return a.P + a.v * t;
}
```

//半平面交的主过程

//L数组存所有有向直线，n为有向直线个数，半平面交的交点存在poly数组中

```
int half_plane_intersection(Line* L, int n, Point* poly)
```

```
{
    sort(L, L + n); //按照极角排序

    int first, last; //双端队列
    Point *p = new Point[n]; //p[i]为q[i]和q[i + 1]的交点
    Line *q = new Line[n]; //手写的Line类型的双端队列（数组）
    q[first = last = 0] = L[0]; //双端队列初始化的时候只有一个半平面L[0]
    for(int i = 1; i < n; ++ i){
        while(first < last && !on_left(L[i], p[last - 1])) last -- ;
        while(first < last && !on_left(L[i], p[first])) first ++ ;
        q[++ last] = L[i]; //新的点是一定要放进去的
        if(fabs(Cross(q[last].v, q[last - 1].v)) < eps){
            //相邻的两个向量平行且同向，取内侧的那一个

```

```

        last -- ; //如果新的向量上的一个点在老的向量的左侧就取新的
        if(on_left(q[last], L[i].P))q[last] = L[i];
    }
    if(first < last)p[last - 1] = get_intersection(q[last - 1], q[last]);
}
while(first < last && !on_left(q[first], p[last - 1]))last -- ;
//删除无用的平面

if(last - first <= 1)return 0; //空集
p[last] = get_intersection(q[last], q[first]); //计算首尾两个半平面交（环状）
//从手写deque中复制答案到输出数组中
int m = 0;
for(int i = first ; i <= last; ++ i)poly[m ++ ] = p[i];
return m;
}

```