寻找质因数

```
#include <iostream>
#include <algorithm>

using namespace std;

bool is_prime(int x)
{
    if (x < 2) return false;
    for (int i = 2; i <= x / i; i ++ )//不要用开方函数或者i*i小于x。开方函数慢，i*i可能越界
        if (x % i == 0)
            return false;
    return true;
}

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

    while (n -- )
    {
        int x;
        cin >> x;
        if (is_prime(x)) cout << "Yes" << endl;
        else cout << "No" << endl;;
    }

    return 0;
}
```

分解质因数

```
#include <iostream>
#include <algorithm>

using namespace std;

void divide(int x)
{
    for (int i = 2; i <= x / i; i ++ )//i <= x / i:防止越界，速度大于 i < sqrt(x)
        if (x % i == 0)//i为底数
        {
            int s = 0;//s为指数
            while (x % i == 0) x /= i, s ++ ;
            cout << i << ' ' << s << endl;//输出
```

```
        }
    if (x > 1) cout << x << ' ' << 1 << endl;//如果x还有剩余，单独处理
    cout << endl;
}

int main()
{
    int n;
    cin >> n;
    while (n -- )
    {
        int x;
        cin >> x;
        divide(x);
    }

    return 0;
}
```

AcWing 869. 试除法求约数

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main()
{
    int T;
    cin >> T;
    while(T--)
    {
        int n;
        cin >> n;
        vector<int> res;
        //因为约数成对出现，所以只需要循环到根号x
        // 不要是用 i *i <= n，因为可能溢出
        for(int i = 1; i <= n /i; i++)
        {
            if(n % i == 0)
            {
                res.push_back(i);
                //如果i * i = x,添加i即可，不用添加 x / i
                if(n / i != i)
                    res.push_back(n / i);
            }
        }
        sort(res.begin(), res.end());
        for(auto x : res) cout << x << " ";
```

```
        cout << endl;

    }
}
```

AcWing 869. 试除法求约数

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main()
{
    int T;
    cin >> T;
    while(T--)
    {
        int n;
        cin >> n;
        vector<int> res;
        //因为约数成对出现，所以只需要循环到根号x
        // 不要是用 i *i <= n，因为可能溢出
        for(int i = 1; i <= n /i; i++)
        {
            if(n % i == 0)
            {
                res.push_back(i);
                //如果i * i = x,添加i即可，不用添加 x / i
                if(n / i != i)
                    res.push_back(n / i);
            }
        }
        sort(res.begin(), res.end());
        for(auto x : res) cout << x << " ";
        cout << endl;

    }
}
```

约数个数

```
#include <iostream>
#include <algorithm>
#include <unordered_map>
using namespace std;
const int mod = 1e9+7 ;
int main()
```

```
{


 int n; cin>>n;
 unordered_map<int, int> h;

 while(n--){
     int k; cin>>k;
     for(int i = 2; i <= k / i; i++){
         if(k % i == 0){
            while(k % i == 0){
                h[i]++;
                k /= i;
            }
         }
     }

     if(k > 1) h[k]++;
 }

 long long res = 1;
 for(auto m : h) res = res * (m.second + 1) % mod;

 cout<<res;

 return 0;
}
```

AcWing 871. 约数之和

基本思想：

如果 $N = p1^{c1} * p2^{c2} * \ldots * pk^{ck}$

约数个数：$(c1 + 1) * (c2 + 1) * \ldots * (ck + 1)$

约数之和：$(p1^0 + p1^1 + \ldots + p1^{c1}) * \ldots * (pk^0 + pk^1 + \ldots + pk^{ck})$

```
#include <iostream>
#include <algorithm>
#include <unordered_map>
#include <vector>

using namespace std;

typedef long long LL;

const int N = 110, 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 p : primes)
    {
        LL a = p.first, b = p.second;
        LL t = 1;//这里必须是long long
        while (b -- ) t = (t * a + 1) % mod;
        res = res * t % mod;
    }

    cout << res << endl;

    return 0;
}
```

AcWing 872. 最大公约数

```
#include <iostream>
#include <algorithm>
using namespace std;
int main()
{
    int T;
    cin >> T;
    while(T--)
    {
```

```
        int a, b;
        cin >> a >> b;
        //a % b == 0,  则b就是最大公约数，可以提前结束循环。
        while(a % b)
        {
            int c = a % b;
            a = b;
            b = c;
        }
        cout << b << endl;
    }
}
```

快速幂

```
#include<iostream>
using namespace std;

//a这个参数必然是long long
long long qmi(long long a,int b,int p)
{
    long long res=1;
    while(b)//对b进行二进制化,从低位到高位
    {
        //如果b的二进制表示的第0位为1,则乘上当前的a
        if(b&1) res = res *a %p;
        //b右移一位
        b>>=1;
        //更新a,a依次为a^{2^0},a^{2^1},a^{2^2},....,a^{2^logb}
        a=a*a%p;
    }
    return res;
}
int main()
{
    int n;
    cin>>n;
    while(n--)
    {
        int a,b,p;
        long long res=1;
        cin>>a>>b>>p;
        res = qmi(a,b,p);
        cout<<res<<endl;
    }
    return 0;
}
```

拓展欧几里得

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

void exgcd(int a, int b, int & x, int & y){

    if(b == 0){
        x = 1; y = 0;
        return;
    }
    exgcd(b, a % b, x, y);
    int x1 = x, y1 = y;
    x = y1, y = x1 - a / b * y1;
}


int main(){

    int n;cin>>n;

    while(n--){

        int a, b, x, y; cin>>a>>b;
        exgcd(a, b, x, y);
        cout<<x<<" "<<y<<endl;
    }
    return 0;
}
```

求组合数

```
#include<iostream>
using namespace std;
const int mod = 1e9+7;
long long f[2010][2010];
int main()
{
    //预处理
    for(int i=0;i<=2000;i++)
    {
        for(int j=0;j<=i;j++)
        {
            if(!j) f[i][j]=1;
            else f[i][j]=(f[i-1][j-1]+f[i-1][j])%mod;
        }
    }
```

```
    int n;
    cin>>n;
    while(n--)
    {
        int a,b;
        cin>>a>>b;
        printf("%ld\n",f[a][b]);
    }
}
```

快速求逆元

```
#include <iostream>
using namespace std;
typedef long long LL;

LL qmi(LL a, int m, int p){

    LL res = 1;
    while(m){

        if(m & 1) res = res * a % p;
        a = (a * a) % p;
        m >>= 1;

    }
    return res;
}


int main(){


    int n; cin>>n;

    while(n--){

        int a, p; cin>>a>>p;
        if( a % p == 0) puts("impossible");
        else cout<<qmi(a, p - 2, p)<<endl;
    }
    return 0;
}
```

求组合数 1e5

```
#include<iostream>
#include<algorithm>

using namespace std;

typedef long long LL;
const int N=100010,mod=1e9+7;

LL fact[N],infact[N];//fact表示阶乘,infact表示阶乘的逆元

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

int main(){

    fact[0] = 1;
    infact[0] = 1;
    for(int i = 1; i <= 1e5; i++){
        fact[i] = (LL)fact[i - 1] * i % mod;
        infact[i] = (LL)infact[i - 1] * qmi(i, mod - 2, mod) % mod;
    }

    int n; cin>>n;
    while(n--){
        int a, b; cin>>a>>b;
        cout<<fact[a] * infact[b] % mod * infact[a - b] % mod<<endl;
    }

    return 0;
}
```

求组合数

```
#include<iostream>
#include<algorithm>
#include<vector>

using namespace std;

const int N=5010;

int primes[N],cnt;
int sum[N];
bool st[N];

void get_primes(int n)
{
    for(int i=2;i<=n;i++)
    {
        if(!st[i])primes[cnt++]=i;
        for(int j=0;primes[j]*i<=n;j++)
        {
            st[primes[j]*i]=true;
            if(i%primes[j]==0)break;//==0每次漏
        }
    }
}
// 对p的各个<=a的次数算整除下取整倍数
int get(int n,int p)
{
    int res =0;
    while(n)
    {
        res+=n/p;
        n/=p;
    }
    return res;
}
//高精度乘
vector<int> mul(vector<int> a, int b)
{
    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;
    }
    // while(C.size()>1 && C.back()==0) C.pop_back();//考虑b==0时才有pop多余的0 b!=0不需要这行
    return c;
}

int main()
{
    int a,b;
    cin >> a >> b;
    get_primes(a);

    for(int i=0;i<cnt;i++)
    {
        int p = primes[i];
        sum[i] = get(a,p)-get(a-b,p)-get(b,p);//是a-b不是b-a
    }

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

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

    for (int i = res.size() - 1; i >= 0; i -- ) printf("%d", res[i]);
    puts("");

    return 0;
}
```

能被整除的数

```
#include<iostream>
using namespace std;
typedef long long LL;

const int N = 20;
int p[N], n, m;

int main() {
    cin >> n >> m;
    for(int i = 0; i < m; i++) cin >> p[i];
```

```
    int res = 0;
    //枚举从1 到 1111...(m个1)的每一个集合状态, (至少选中一个集合)
    for(int i = 1; i < 1 << m; i++) {
        int t = 1;              //选中集合对应质数的乘积
        int s = 0;              //选中的集合数量

        //枚举当前状态的每一位
        for(int j = 0; j < m; j++){
            //选中一个集合
            if(i >> j & 1){
                //乘积大于n, 则n/t = 0, 跳出这轮循环,这一步的判断必须要有, 目前猜测的原因是, 如果不判断, t的范围会超越, 会出现一些小于n的数字, 导致错误
                if((LL)t * p[j] > n){
                    t = -1;
                    break;
                }
                s++;                    //有一个1, 集合数量+1
                t *= p[j];
            }
        }

        if(t == -1) continue;

        if(s & 1) res += n / t;             //选中奇数个集合, 则系数应该是1, n/t为当前这种状态的集合数量
        else res -= n / t;                      //反之则为 -1
    }

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

滑雪

```
#include <iostream>
#include <algorithm>
#include <cstring>
using namespace std;
const int N = 310;
int n,m; //网格滑雪场的行和列
int f[N][N]; //状态转移式
int h[N][N]; //网格滑雪场
int dx[4] = {-1,0,1,0};
int dy[4] = {0,1,0,-1};
int vist[N][N];
```

```
int dp(int x, int y){

    if(vist[x][y] == 1){
        return f[x][y];
    }

    vist[x][y] = 1;
    f[x][y] = 1;

    for(int i = 0; i < 4; i++){
        int a = x + dx[i], b = y + dy[i];
        if(a >= 1 && a <= n && b >= 1 && b <= m && h[x][y] > h[a][b]){
             f[x][y] = max(f[x][y], dp(a, b) + 1);
        }
    }

    return f[x][y];


}


int main(){

    cin>>n>>m;
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= m; j++){
            cin>>h[i][j];
        }
    }

    memset(f, -1, sizeof f);

    int res = 0;
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= m; j++){
            res = max(res, dp(i, j));
        }
    }

    cout<<res;
    return 0;

}
```

\92. 递归实现指数型枚举

```
#include<iostream>
using namespace std;

int n;
int path[15], state[15];


int main(){
    cin>>n;

    for(int i = 0; i < 1 << n; i++){
        for(int j = 0; j <= n; j++){
            if(i >> j & 1) cout<< j + 1<<" ";
        }
        cout<<endl;
    }

    return 0;
}
```

递归实现指数型枚举

```
#include <iostream>
using namespace std;

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

// 当前枚举到第pos个坑， 上一个坑填的是start-1，这次只能从start开始找数填， 一共要填tar个坑
void dfs(int pos, int start, int tar) {
    if (pos == tar + 1) {
        for (int i = 1; i <= tar; i ++ ) cout << a[i] << " ";
        cout << endl;
        return ;
    }

    // 选数填坑，选择的数范围是start～n
    for (int i = start; i <= n; i ++) {
        if (!vis[i]) {
            vis[i] = true; a[pos] = i;
                dfs (pos + 1, i + 1, tar);
            vis[i] = false;
        }
    }
```

```
}

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

递归实现指数型枚举

```
#include <iostream>
using namespace std;

int main() {
    int n;
    cin >> n;
    // state 是每一个状态
    for (int state = 0; state < 1 << n; state ++ ) {
        // 用指针j遍历二进制数state中的每一位
        for (int j = 0; j < n; j ++ ) {
            if (state >> j & 1) cout << j + 1 << " ";
        }
        cout << endl;
    }
    return 0;
}
```

递归实现指数型枚举

```
#include <iostream>
using namespace std;

int n;

// u是当前枚举到的数，state是二进制数记录哪些数被选
void dfs(int u, int state) {
    if (u == n) {
        for (int i = 0; i < n; i ++)
            if (state >> i & 1)
                cout << i + 1 << " ";
            cout << endl;
        return ;
    }

    dfs (u + 1, state);  // 不用u这个数
```

```
    dfs (u + 1, state | (1 << u)); // 用u这个数
}

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

AcWing 94. 递归实现排列型枚举

```
#include<iostream>
using namespace std;
const int N = 10;
int path[N];//保存序列
int state[N];//数字是否被用过
int n;
void dfs(int u)
{
    if(u > n)//数字填完了，输出
    {
        for(int i = 1; i <= n; i++)//输出方案
            cout << path[i] << " ";
        cout << endl;
    }

    for(int i = 1; i <= n; i++)//空位上可以选择的数字为:1 ~ n
    {
        if(!state[i])//如果数字 i 没有被用过
        {
            path[u] = i;//放入空位
            state[i] = 1;//数字被用，修改状态
            dfs(u + 1);//填下一个位
            state[i] = 0;//回溯，取出 i
        }
    }
}

int main()
{

    cin >> n;
    dfs(1);
    return 0;
}
```

\717. 简单斐波那契

```
#include<iostream>
using namespace std;
const int N = 50;

int n;
int dp[N];


int main(){

    cin>>n;
    dp[1] = 0; dp[2] = 1;

    for(int i = 3; i <= n; i++){
        dp[i] = dp[i - 1] + dp[i - 2];
    }

    for(int i = 1; i <= n; i++){
        cout<<dp[i]<<" ";
    }


    return 0;
}
```

95. 费解的开关

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>

using namespace std;

const int N = 6;
int dx[N] = {-1, 0, 1, 0, 0}, dy[N] = {0, 1, 0, -1, 0};
char g[N][N], backup[N][N];


// 这个操作是把(x, y)以及上下左右的灯都变成相反的颜色
void turn (int x, int y)
{
    for (int i = 0; i < 5; i ++ )
    {
        int a = x + dx[i], b = y + dy[i];
```

```
        //如果在边界外边，直接忽略即可
        if (a < 0 || a >= 5 || b < 0 || b >= 5) continue;

        g[a][b] ^= 1;   //异或，不同的时候就变成相反的数
    }

}


int main()
{
    int n;
    scanf("%d", &n);
    while(n -- )
    {
        // 按行输入，把每一行当成一个字符串
        for (int i = 0; i < 5; i ++ ) cin >> g[i];

        int res = 10;

        // 这里我们枚举了第一行的32种按法，不用管是亮是灭，把第一行所有情况都按一遍
        // 按每种情况的第一行，去遍历接下来的行
        // 枚举32种第一行的按法只是可能会减少步数，如果直接从第二行开始答案一定是固定的了，找不到最优解或者可能没有解

        for (int op = 0; op < 32; op ++ )
        {
            // 我在对这种情况操作的时候，得先备用一下
            // 把原始数组备份一下，然后操作g，操作完了还原，然后再操作
            memcpy(backup, g, sizeof g);

            int step = 0;

            // 第一行的按法(在这里 1 表示按了，0 表示不按)，这里只是为了输出第一行按完之后的状态
            for (int i = 0; i < 5; i ++ )
                if (op >> i & 1)  // 数字2 对应了 00010 表示第2个位置的按一下
                                  // 00010 >> 1 & 1  是1 所以turn(0, 1) 就是第一行第二个位置
                {                 // 数字3 对应了00011 表示第1 和第2个位置的按一下
                    step ++ ;
                    turn (0, i);;
                }

            // 然后通过第一行按完之后的状态，按234行
            for (int i =0; i < 4; i ++ )
                for (int j = 0; j < 5;j ++ )
                    if (g[i][j] == '0')
                    {
```

```
                        step ++;
                        turn (i + 1, j);  // 如果这个位置是灭的，就按下一行对应的位置
                    }

            bool dark = false;
            for (int j = 0; j < 5; j ++ )
                if (g[4][j] == '0')
                {
                    dark = true;
                    break;
                }


            // 对于32种情况的这一种，如果所有的全亮就记录下步数(事实上只记录了最后一行是否dark)
            if (!dark) res = min(res, step);
            memcpy (g, backup, sizeof g);
        }

        if(res > 6) res = -1;
        cout << res << endl;

    }
    return 0;
}
```

带分数

```
#include <iostream>

using namespace std;

const int N = 10;

int target;  // 题目给出的目标数
int num[N];  // 保存全排列的结果
bool used[N];  // 生成全排列过程中标记是否使用过
int cnt;  // 计数，最后输出的结果

// 计算num数组中一段的数是多少
int calc(int l, int r) {
  int res = 0;
  for (int i = l; i <= r; i++) {
    res = res * 10 + num[i];
  }
  return res;
}

// 生成全排列
```

```
// 当全排列生成后进行分段
void dfs(int u) {
  // 用两层循环分成三段
  if (u == 9) {
    for (int i = 0; i < 7; i++) {
      for (int j = i + 1; j < 8; j++) {
        int a = calc(0, i);
        int b = calc(i + 1, j);
        int c = calc(j + 1, 8);
        // 注意判断条件，因为C++中除法是整除，所以要转化为加减乘来计算
        if (a * c + b == c * target) {
          cnt++;
        }
      }
    }
    return;
  }
  // 搜索模板
  for (int i = 1; i <= 9; i++) {
    if (!used[i]) {
      used[i] = true; // 标记使用
      num[u] = i;
      dfs(u + 1);
      used[i] = false; // 还原现场
    }
  }
}

int main() {
  scanf("%d", &target);
  dfs(0);
  printf("%d\n", cnt);
  return 0;
}
```

带分数

```
#include <bits/stdc++.h>

using namespace std;

const int N = 10;

int target;
int num[N];

int calc(int l, int r) {
  int res = 0;
```

```
  for (int i = l; i <= r; i++) {
    res = res * 10 + num[i];
  }
  return res;
}

int main() {
  cin >> target;
  for (int i = 0; i < 9; i++) {
    num[i] = i + 1;
  }
  int res = 0;
  do {
    for (int i = 0; i < 9; i++) {
      for (int j = i + 1; j < 9; j++) {
        int a = calc(0, i);
        int b = calc(i + 1, j);
        int c = calc(j + 1, 8);
        if (a == 0 || b == 0 || c == 0) {
          continue;
        }
        if (a * c + b == c * target) {
          ++res;
        }
      }
    }
    // 调用函数生成全排列
  } while (next_permutation(num, num + 9));
  cout << res << '\n';
  return 0;
}
```

飞行员兄弟

```
#include<iostream>
#include<cstring>
#include<cstdio>
#include<vector>
#include<algorithm>

//这个宏定义其实也就最后输出的时候应用了（如果我没猜错的话），但是y总的习惯就是好习惯!
#define x first
#define y second

using namespace std;

typedef pair<int,int> PII;
```

```
const int N=5;

char g[N][N],backup[N][N];


//映射函数
int get(int x,int y)
{
    return x*4+y;//返回第x行第y列上的数是多少
}

void turn_one(int x,int y)
{
    if(g[x][y]=='+') g[x][y]='-';
    else g[x][y]='+';
}

void turn_all(int x,int y)
{
    for(int i=0;i<4;i++)
    {
        turn_one(x,i);
        turn_one(i,y);
    }
    turn_one(x,y);

}

int main()
{
    for(int i=0;i<4;i++)
        for(int j=0;j<4;j++)
            cin>>g[i][j];

    vector<PII> res;//这是记录方案所需要的结构

    //枚举所有的方案
    for(int op=0;op<1<<16;op++)
    {
        vector<PII> temp;//temp里面存的是方案
        //先备份一下，为什么？因为这又不是最终方案，我们要把所有方案都试一遍，求最少的
        memcpy(backup,g,sizeof g);

        //枚举16个位置,进行操作
        for(int i=0;i<4;i++)
            for(int j=0;j<4;j++)
                if(op>>get(i,j)&1) //如果当前位置是1的话--get的作用就是返回二进制数中那一位是第几位，从而判断是否为1
                {
```

```
                    temp.push_back({i,j});
                    //按一下开关
                    turn_all(i,j);
                }


        //判断所有灯泡是否全亮
        bool has_closed=false;
        for(int i=0;i<4;i++)
            for(int j=0;j<4;j++)
                if(g[i][j]=='+') has_closed=true;

        if(has_closed==false)
        {
            //如果方案为空或者他的操作数大于我们刚存好的新的方案，那么就修改它
            if(res.empty()||res.size()>temp.size()) res=temp;
        }
        //还原回来，供下一个方案操作
        memcpy(g,backup,sizeof g);
    }
    //因为没说无解，所以可以猜想一下一定有解
    cout<<res.size()<<endl;
    //这里的迭代函数就是一种简便写法，不要误解
    //另外原题下标从1开始，所以下面加1了
    for(auto op:res) cout<<op.x+1<<" "<<op.y+1<<endl;

    return 0;
}
```

翻硬币

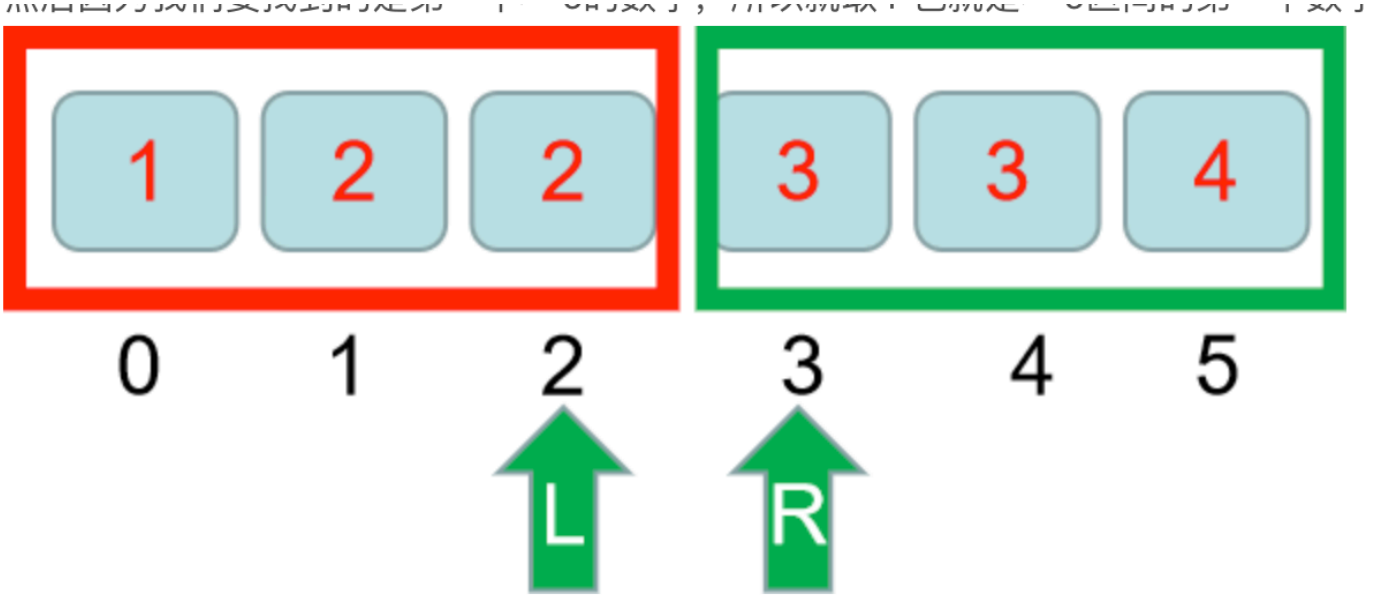
```
//cpp
#include <iostream>
#include <string>
using namespace std;
const int N = 110;//用不到，看范围就直接写了

int main()
{
    string s1, s2;//s1:初始状态，s2:目标状态
    int cnt = 0;//记录翻转次数
    cin >> s1 >> s2;
    for (int i = 0; i < s1.size() - 1; i++)
    {
        if (s1[i] != s2[i])//第 i 个位置上状态不同，就翻转该位置和后一个位置硬币
        {
            cnt++;//翻转一次硬币
            if (s1[i] == '*') s1[i] = 'o';//翻转 i 位置上的硬币
```

```
            else s1[i] = '*';
            if (s1[i+1] == 'o') s1[i+1] = '*';//翻转 i + 1 位置上的硬币
            else s1[i+1] = 'o';
        }
    }
    cout << cnt;//输出翻转次数
}
```

二分模板

```
int L=-1,R=n;
while(L+1!=R)
{
  int mid=L+R>>1;
  if(check()) L=mid;
  else R=mid;
  //最后根据你所分左右两边区间的结果
  //选取L或者R作为结果
}
```



数的范围

```
#include<iostream>
using namespace std;
const int N=1e5+5;
int n, m, q[N];
int main()
{

    cin>>n>>m;
```

```
    for(int i = 0; i < n; i++) cin>>q[i];

    while(m--){

        int k; cin>>k;
        int l = -1, r = n;

        while(l + 1 != r){
            int mid = l + r >> 1;
            if(q[mid] >= k) r = mid;//右半部分大于k
            else l = mid;
        }

        if(q[r] != k) cout<<"-1 -1"<<endl;
        else{
            cout<<r<<" ";

            int l = -1, r = n;

            while(l + 1 != r){
                int mid = l + r >> 1;
                if(q[mid] <= k) l = mid;//左半部分小于k
                else r = mid;
            }

            cout<<l<<endl;

        }
    }
    return 0;
}
```

四次方，拉格朗日

```
//四平方和
//模拟哈希表
#include<bits/stdc++.h>

using namespace std;

typedef pair<int,int> PII;

const int N = 5e6 + 10;

int n;
int r[N * 2];//小技巧，避免pair，r[c^2+d^2]=c;可以推导出d

int main()
```

```
{
    cin >> n;

    memset(r, -1, sizeof r);

    for(int c = 0; c * c <= n; c++)
    {
        for(int d = c; c * c + d * d <= n; d++)
        {
            int t = c * c + d * d;
            if(r[t] == -1)
            {
                r[t] = c;
            }
        }
    }

    for(int a = 0; a * a <= n; a ++)
    {
        for(int b = a; a * a + b * b <= n; b++)
        {
            int t = n - a * a - b * b;
            int c = r[t];
            if(r[t] == -1) continue;
            int d = sqrt(t - c * c);
            printf("%d %d %d %d\n", a, b, c, d);
            return 0;
        }
    }
    return 0;
}
```

730. 机器人跳跃问题

```
#include<bits/stdc++.h>
using namespace std;
const int N=100010;
int num[N];
int n;
int main(){
    cin>>n;
    for(int i=0;i<n;i++)cin>>num[i];
    int res = 0;
        for(int i = n - 1; i >= 0; i--){
            if((num[i] + res) % 2 == 0)
                res = (num[i] + res) / 2;
            else
                res = (num[i] + res) / 2 + 1;
```

```
        }
    cout<<res<<endl;
}
```

```
#include<iostream>
using namespace std;
const int N = 1e5 + 10;

int n;
int q[N];

bool check(int e){

    for(int i = 0; i < n; i++){
        e = 2 * e - q[i];
        if(e > 1e5) return true;
        if(e < 0){
            return false;
        }

    }
    return true;

}

int main(){

    cin>>n;
    for(int i = 0; i < n; i++) scanf("%d", &q[i]);

    int l = -1, r = 1e5;
    while(l + 1 != r){

        int mid = (l + r) >> 1;
        if(check(mid)) r = mid;
        else l = mid;

    }

    cout<<r;
    return 0;
}
```

分巧克力

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 100010;

int n, k;  // 有n块巧克力，k个小朋友
int wid[N], len[N];  // 分别保存巧克力的长和宽

// 计算当正方形的边长为u时，总共能分得多少块巧克力
int get_sum(int u) {
  int sum = 0;
  for (int i = 0; i < n; i++) {
    // 要的就是整除
    sum += ((wid[i] / u) * (len[i] / u));
  }
  return sum;
}

int main() {
  scanf("%d%d", &n, &k);
  for (int i = 0; i < n; i++) {
    scanf("%d%d", &wid[i], &len[i]);
  }
  // 二分模板
  // 按照y总所分析的，我这里所取的性质是“当边长取mid时，能够分得k块巧克力”
  // 显然在一个数轴上，目标值的左侧（比目标值小的数）都是满足这个性质的，右边则不满足
  // 所以我们要找的值是满足这个性质的最右端
  // 如果当前取的mid满足这个性质，那么l有可能就是我们要找的目标值，所以用l = mid来更新
  // 综上所述，用以下模板
  int l = 0, r = N;
  while (l < r) {
    int mid = (l + r + 1) >> 1;
    if (get_sum(mid) >= k) {
      l = mid;
    } else {
      r = mid - 1;
    }
  }
  printf("%d\n", l);
  return 0;
}
```

炸弹

```cpp
#include <iostream>

using namespace std;

#define N 5002
int f[N][N];

int main() {

    int n, r;
    int max_x, max_y;
    int x = 0, y = 0, w = 0;
    cin >> n >> r;
    r = min(r, 5001); // 爆炸范围超过5001时覆盖了所有目标范围，因此没必要考虑超出范围。
    max_x = max_y = r;// 最小的爆炸计算范围是在一个最小爆炸范围之内计算
    for (int i = 0; i < n; ++i) {
        cin >> x >> y >> w;
        f[++x][++y] += w; // 将目标范围统一偏移1，便于边界计算。
        max_x = max(x, max_x); // 如果有目标落在一个最小爆炸范围之外，则扩大计算范围。
        max_y = max(y, max_y);
    }
    // 下面两个循环的计算公式都需要自己用手画一下，才能获得利用二维前缀和计算总价值的结果
    for (int i = 1; i <= max_x; ++i) { // 循环从偏移的1开始
        for (int j = 1; j <= max_y; ++j) {
            f[i][j] = f[i - 1][j] + f[i][j - 1] - f[i - 1][j - 1] + f[i][j];
        }
    }

    int ans = 0;
    for (int i = r; i <= max_x; ++i) {
        for (int j = r; j <= max_y; ++j) {
            ans = max(ans, f[i][j] - f[i - r][j] - f[i][j - r] + f[i - r][j - r]);
        }
    }

    cout << ans << endl;

    return 0;
}
```

最大不能凑成的数

```
#include <iostream>
#include <algorithm>
using namespace std;
int n, m, minn, maxx, ans;
bool dp[1000000];
int main() {
    cin >> n >> m;
    dp[0] = true;
    minn = min(n, m);
    maxx = max(n, m);
    for (int i = minn; i < n * m; i++) {
        if (dp[i - minn]) {
            dp[i] = true;
        } else if (i >= maxx && dp[i - maxx]) {
            dp[i] = true;
        } else {
            ans = i;
        }
    }

    if(n == 2 && m == 3) cout<<1<<endl;
    else cout << ans;
    return 0;
}
```

```
#include <iostream>

using namespace std;

//给定一个m，是否能用p和q凑出来
bool dfs(int m,int p,int q)
{
    if(m == 0) return true;

    if(m >= p && dfs(m - p,p,q)) return true;
    if(m >= q && dfs(m - q,p,q)) return true;

    return false;
}

int main()
{
    int p,q;
    cin >> p >> q;
    int res = 0;
    for(int i = 1; i <= 1000;i ++)
```

```
    {
        if(!dfs(i,p,q)) res = i;
    }

    cout << res << endl;

    return 0;
}
```