我对称吗

```cpp
#include <iostream>

using namespace std;
const int N = 110;
int n, arr[N];

bool check() {
  for (int i = 0; i < n / 2; i++)
    if (arr[i] != arr[n - 1 - i]) return false;
  return true;
}

int main() {
  scanf("%d", &n);
  for (int i = 0; i < n; i++)
    scanf("%d", &arr[i]);
  puts(check() ? "YES" : "NO");
    return 0;
}
```

跳格子

假设从格子的最左端跳到最右端，并且题意是可以上下左右跳，也可以斜着跳。

因为必须要从最左跳到最右，而且跳过的格子不能重复跳，那么上下两个格子至少必须跳一个。

为了使分数最大，根据数据范围分为以下的情况。

如果上下两个格子都是正数，那为了使答案最大，两个格子都必须跳。

如果两个格子中有一个是正数，另一个不是正数，那肯定就是跳是正数的格子。

如果两个格子都不是正数，即为负数，但根据上面的分析你上下两个格子至少跳一个，为了使分数最大，所以就跳格子上的值大的那个。

```cpp
#include <bits/stdc++.h>
using namespace std;
int a[105];
int main(){
```

```
  int n;cin >> n;
  for(int i = 1; i <= n; i ++)cin >> a[i];
  int x, ans = 0;
  for(int i = 1; i <= n; i ++){
    cin >> x;
    if(x > 0 && a[i] > 0){
      ans += x + a[i];
    }else if(x < 0 && a[i] < 0){
      ans += max(x, a[i]);
    }else{
      ans += x > a[i] ? x : a[i];
    }
  }
  cout << ans << endl;
}
```

计数

简单模拟即可

```
int T; cin >> T;
while (T--) {
    mst(cot);//清空数组
    int n; cin >> n;
    string s; cin >> s;
    for (char c : s)cot[c]++;
    int mx = *max_element(cot, cot + 128);
    for (int i = 'a'; i <= 'z'; i++) {
        if (cot[i] == mx) {
            cout << (char)i << endl;
            break;
        }
    }
}
```

复杂的求和

观察式子可以发现，其实 就是

$a_1*(b_1+b_2+b_3....+b_n)+a_2*(b_1+b_2+b_3....+b_n)+...+a_n*(b_1+b_2+b_3....+b_n)$

即两个数组的和相乘

```cpp
#include<bits/stdc++.h>
#define int long long
using namespace std;
const int N=1e5+100;
const int mod=1e9+7;
int a[N];
int b[N];
typedef pair<int,int> P;
signed main(){
  int n;
  cin>>n;
  int sum1=0;
  int sum2=0;
  for(int i=1;i<=n;i++)
  {
    cin>>a[i];
    sum1+=a[i];
  }
  for(int i=1;i<=n;i++)
  {
    cin>>b[i];
    sum2+=b[i];
  }
  cout<<sum1*sum2<<endl;
}
```

计数

这个题数据范围很小 有很多做法

~~我直接破防 甚至把数据改到了1e6 当然 又改回来了~~

输入之后逐个判断：

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

```cpp
int res[5];
bool isp(int x){
  if(x<2) return false;
  for(int i=2;i<=sqrt(x);i++){
    if(x%i==0)return false;
  }
  return true;
}
bool is2(int x){
  int t=sqrt(x);
  return t*t==x;
}
bool is3(int x){
  if(x>0){
    for(int i=0;i*i*i<=x;i++){
      if(i*i*i==x)return true;
    }
  }else{
    for(int i=0;i*i*i>=x;i--){
      if(i*i*i==x)return true;
    }
  }
  return false;
}
int main(){
  int n;cin>>n;
  for(int i=0;i<n;i++){
    int x;cin>>x;
    if(x&1)res[0]++;
    else res[1]++;
    if(isp(x))res[2]++;
    if(is2(x))res[3]++;
    if(is3(x))res[4]++;
  }
  for(int i=0;i<5;i++)cout<<res[i]<<' ';
  #define _ 0
  return ~~(0^_^0);
}

```

打表：

```cpp
#include <iostream>
#include <set>
using namespace std;
set<int> isp=
{2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97
};
set<int> is2={0,1,4,9,16,25,36,49,64,81,100};
set<int> is3={0,-1,1,-8,8,-27,27,-64,64};
int res[5];
int main(){
  int t;cin>>t;
  while(t--){
    int x;cin>>x;
    if(x&1)res[0]++;
    else res[1]++;
    if(isp.count(x))res[2]++;
    if(is2.count(x))res[3]++;
    if(is3.count(x))res[4]++;
  }
  for(int i=0;i<5;i++)cout<<res[i]<<' ';
  #define _ 0
  return ~~(0^_^0);
}
```

3句话，算出你的生日

思路

蓝桥杯C/C++组真的很喜欢考日期类的题目，大家如果做过往年的题目就能知道了。所以别骂出题人了

这题我们需要计算两个日期做差得到的天数，同时也需要计算某一个日期加上一个天数所得到的日期。如果每次直接计算两个日期的相差天数很麻烦，不妨直接计算某一个日期到某个 *固定日期* （如1年1月1日 / 1970年1月1日）所相差的天数。这样的话，某个日期加上天数也可以转化为计算某个固定日期加上天数所得到的日期。

下面观察数据范围有：

由于每年可能是365天也可能是366天，所以在计算天数时，我们需要当前处理的年份是否为闰年，如果我们在每个测试样例中每次都判断的话，每次大致会执行1e5次闰年函数，对于$T = 50000$的样例显然可能超时，考虑可以使用一下两种方法优化：

1. 由于闰年规则，每400年一定会有97个闰年，所以如果日期相差过大，可以直接先提前处理400年。
2. 在测试样例前提前计算好每一年到 *固定日期* 的天数。

首先定义一个结构体 `struct Day{int y, m, d;}`
因此我们定义两个函数分别为`int dayFrom1970(Day)`代表某个日期到1970年1月1日的天数； `Day dayTo1970(int)` 代表从1970年开始经过天数所得到的日期。

最后由于天数最大为 `100000 * 365 * x < 3e8`，可以不用开`long long`。

ps:

> 另外，有个和日期有关的蔡勒公式，可以计算出一个日子是周几，可能会有用。
>
> 计算数据范围/时间复杂度真的很重要，不然很容易爆int或者爆时间
>
> 个人感觉还有更优的写法

一个较暴力的做法

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

using namespace std;
void solve();

signed main() {
#ifdef ZZY_DEBUG_LEVEL
    freopen("abc.in", "r", stdin);
//    freopen("abc.out", "w", stdout);
#endif
    int T = 1;
    cin >> T;
    while (T--) {
        solve();
    }
    return 0;
}

// ------------------------------

bool isPrime(int y) {
    return y % 400 == 0 || y % 100 && y % 4 == 0;
}

int MONTH[13] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
const int DAY400 = 97 * 366 + 303 * 365;
```

```
27
28  struct day {
29      int y, m, d;
30
31      day *operator++() {
32          bool flg = isPrime(y);
33          d++;
34          MONTH[2] += flg;
35          if (d > MONTH[m]) {
36              d = 1;
37              m++;
38              if (m == 13) {
39                  m = 1;
40                  y++;
41              }
42          }
43          MONTH[2] -= flg;
44          return this;
45      }
46
47      day *operator--() {
48          bool flg = isPrime(y);
49          d--;
50          MONTH[2] += flg;
51          if (d == 0) {
52              m--;
53              if (m == 0) {
54                  m = 12;
55                  y--;
56              }
57              d = MONTH[m];
58          }
59          MONTH[2] -= flg;
60          return this;
61      }
62
63      bool operator==(const day &rhs) const {
64          return y == rhs.y &&
65                 m == rhs.m &&
66                 d == rhs.d;
67      }
68
69      bool operator!=(const day &rhs) const {
```

```
        return !(rhs == *this);
    }
};

void solve() {
    day d1, d2;
    int rate;
    cin >> d1.y >> d1.m >> d1.d;
    cin >> d2.y >> d2.m >> d2.d;
    cin >> rate;

    int gap = 0;
    while (d2 != d1) {
        ++d1;
        gap++;
    }
    gap *= rate;
    for (; gap; gap--) {
        --d2;
    }

   printf("%05d %02ld %02ld\n", d2.y, d2.m, d2.d);
}
```

AC代码

```
// 也可以提前打表 把这100000年 每年1月1号距离1970年1月1号的长度
#include <bits/stdc++.h>

using namespace std;
#define int ll

typedef long long ll;

void solve();

signed main() {
#ifdef ZZY_DEBUG_LEVEL
    freopen("abc.in", "r", stdin);
    freopen("abc.out2", "w", stdout);
#endif
    int T = 1;
```

```
    cin >> T;
    while (T--) {
        solve();
    }
    return 0;
}

// --------------------------------

int const N = 1e5 + 10;

bool isPrime(int y) {
    return y % 400 == 0 || y % 100 && y % 4 == 0;
}

int MONTH[13] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
const int DAY400 = 97 * 366 + 303 * 365;

int dayTo1970(int y, int m, int d) {
    // 每400年一定有97个闰年
    int day = (y - 1970) / 400 * DAY400;
    y -= (y - 1970) / 400 * 400;

    if (m > 2 && isPrime(y)) day++;

    day += d;
    m--;

    for (; m; m--) {
        day += MONTH[m];
    }

    while (y > 1970) {
        y--;
        day += isPrime(y) + 365;
    }

    return day - 1;
}

// 因为这份代码没有定义Day这个结构体，就用引用传值的方法了
void dayFrom1970(int day, int &y, int &m, int &d) {
    m = 1;
```

```
    d = 1;
    y = 1970 + (day / DAY400) * 400;
    day -= (day / DAY400) * DAY400;

    for (; day >= 365 + isPrime(y); y++) {
        day -= 365 + isPrime(y);
    }

    MONTH[2] += isPrime(y);

    for (; day >= MONTH[m]; m++) {
        day -= MONTH[m];
    }
    d += day;

    MONTH[2] -= isPrime(y);
}

void solve() {
    int y, m, d;
    cin >> y >> m >> d;
    int day1 = dayTo1970(y, m, d);
    cin >> y >> m >> d;
    int day2 = dayTo1970(y, m, d);
    int rate;
    cin >> rate;

    int day = day2 - (day2 - day1) * rate;

    dayFrom1970(day, y, m, d);
    printf("%05lld %02lld %02lld\n", y, m, d);
}
```

古代的炼金术真是奇妙

首先模拟下，就会发现就是个减法

合成次数	剩余量
1	n-m+1
2	n-m+1-m+1
3	n-m+1-m+1-m+1

减法一次一次减很慢，就可以转换成除法

其实有很多种方法转换，这里就讲下我的做法

当$n < m$时，显然直接输出n即可

当 $n\%(m-1) == 0$时，答案应该是$m-1$，因为此时已经不够m个了

当 $n\%(m-1)! = 0$时，答案就是 n%(m-1)

```cpp
#include<bits/stdc++.h>
using namespace std;
#define int long long//注意数据范围，需要开long long
signed main() {
    int n, m;cin >> n >> m;
    if (n < m)cout << n;
    else if (n % (m - 1) == 0) {
        cout << m - 1;
    }
    else
        cout << n % (m - 1);
    cout << '\n';
}
```

yesterday_once_more

先简化一下题目， 如果将函数的返回值改成每次都是返回 1，那么调用函数得到的答案就是 $C_{end-start+1}^{count}$

即从 $end-start+1$ 个数中选择 $count$ 个数的答案， 为什么呢?

```cpp
long long yesterday_once_more(int start, int count, int end) {
    if (count == 0) {
        return 1;
        // return end - (start-1);
    }
    long long sum = 0;
    for (int i = start; i <= end - count + 1; i++) {
        sum += yesterday_once_more(i + 1, count - 1, end);
    }
    return sum;
}
```

观察函数发现可以将递归函数改为for循环形式，如下：

```cpp
long long sum = 0;
for(int i = start; i <= end - count + 1; i++){
    for(int j = i+1; j <= end - (count - 1) + 1; j++){
        ..... 省略很多for循环, 总共 count 个 for 循环
        for(int k = j+1; k <= end; k++){
            sum += 1
        }
    }
}
```

现在知道为什么是 $C_{end-start+1}^{count}$ 了吗 ？

不明白的话我们再模拟一下

```
假设 start = 1, count = 2, end = 3

依次有 1 2 3 三个数字

第一次完整的for循环选择的数字依次为 i=1 j=2
第二次完整的for循环选择的数字依次为 i=1 j=3
第三次完整的for循环选择的数字依次为 i=2 j=3

实际意义就和从 3 个数字中选择 2 个数字一样
```

再将函数的返回值改回 $return\ \ end-(start-1);$

返回 1 和返回 end - (start-1); 有什么差别呢 ？

同样改成 for 循环形式：

```
long long sum = 0;
for(int i = start; i <= end - count + 1; i++){
    for(int j = i+1; j <= end - (count - 1) + 1; j++){
        ..... 省略很多for循环，总共 count 个 for 循环
        for(int k = j+1; k <= end; k++){
            sum += end - k;
        }
    }
}
```

我们忽略最后一层for循环，取而代之我们去遍历最后一层 for 循环的值，那么对于一个 $end-k$，有多少种情况可以得到它呢， 这取决于最后一层前面的循环， 而最后一层前面的循环也是有实际意义的，因为最后一层已经被选了一个数字 k，所以前面的循环要从 $(k-1)-start+1$ 个数字中选出 $count-1$ 个数字， 即 $C_{k-start}^{count-1}$

那么答案就是

```
int total = end - start + 1;
for(int i = 1; i <= total; i++){ // 遍历最后一层的值，实际上 从 i = count 后才有意义， 因为
                // 最后一层循环的 k 初值最小为 count
  res += C[i-1][count-1] * (total-(i+1)+1);
}
cout << res;
```

代码：

```
long long C[70][70];

void Base() {
    for (int i = 0; i <= 60; i++) {
        for (int j = 0; j <= i; j++) {
            if (j == 0) {
                C[i][j] = 1;
            } else {
                C[i][j] = C[i - 1][j] + C[i - 1][j - 1];
            }
        }
```

```
    }
    int start, count, end;
    cin >> start >> count >> end;

    long long res = 0;
    int total = end - start + 1;
    for (int i = 1; i <= total; i++) {
        res += C[i - 1][count - 1] * (total - (i + 1) + 1);
    }
    cout << res;
}
```

出题人的解答就到这里了，但是在验题和比赛过程中看到有的同学提交的代码，发现式子可以进一步简化为

$C_{end-start+1}^{count+1}$

甚至有的同学用记忆化搜索做出来的，

甚至有找规律用前缀和做出来的，

可能大多数同学都是找规律和记忆化搜索做出来的， 但是以上同学都 ：

相当优秀

特殊序列（前缀和，二分，双指针）

这道题确实比较恶心，它有两个坑，一个是k存在0的情况，这种情况下是另外的一个公式，并且它可能爆int，即超过int的范围，所以你需要先考虑k是否等于0，并且考虑什么时候会爆int，

1.双指针：

考虑到恰好有k个，那么我们可以直接用一个l和一个r来标记位置，只需要满足$[l,r]$中有k个1即可，然后每次记录完成后两个指针都向后分别找到第一个1的位置，然后继续记录，最后知道r到达边界，无法继续走的时候停下，总时间复杂度为O(2n)，注意本题需要用longlong来存储，因为最大的时候是当k等于0的时候，这个时候会发现是可以用等差数列求和公式计算的，它的数量级n^2，也就是说会大过int的范围，所以需要用longlong来存储

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

#define _ 0
```

```cpp
#define pll pair<int, int>
#define ll long long
#define int long long
#define endl '\n'
const int N = 1e6 + 10;
const int inf = 1e9;
const int mod = 1e9 + 7;
const double eps = 1e-8;
const double pi = 3.1415926;
//const int M = 710;

vector<pll> str;

void solve() {
    int n; cin >> n;
    string s; cin >> s;
    int l = 0;
    pll p = { -1, -1 };
    if (n == 0) {
        int sum = 0, tmp = 0;
        for (int i = 0; i < s.size(); i++) {
            if (s[i] == '0') tmp++;
            else {
                if (tmp) sum += (1 + tmp) * tmp / 2;
                tmp = 0;
            }
        }
        if (tmp) sum += (1 + tmp) * tmp / 2;
        cout << sum << endl;
        return;
    }
    for (int i = 0; i < s.size(); i++) {
        if (s[i] == '0') l++;
        else {
            if (p.first == -1) p = { l, -1 };
            else p.second = l, str.push_back(p), p = { l, -1 };
            l = 0;
        }
    }
    // cout << str.size() << endl;
    // for (auto i : str) cout << i.first << ' ' << i.second << endl;
    int ans = 0;
    p.second = l, str.push_back(p);
```

```
    if (n > str.size()) cout << 0 << endl;
    else {
        int l = 0;
        for (int i = n - 1; i < str.size(); i++) {
            ans += (str[l].first + 1) * (str[i].second + 1);
            l++;
        }
        cout << ans << endl;
    }

}


signed main() {
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    int T = 1; //cin >> T;
    while (T--)solve();
    return ~~(0 ^ _ ^ 0);
}
```

2.二分

当对整个数列进行前缀和之后，我们会发现整个数列是不严格递增的，对于这种具有单调性的数列，我们可以采用二分的方式来对于每一个a_i找到$a_i + k$的位置，这里推荐学习一下

```
lower_bound 和upper_bound
```

是c++内置的二分函数，其中lower是大于等于这个数的第一个数，upper是严格大于这个数的第一个数

具体实现和上一种方式差不多（感谢验题人yxc的倾情贡献）

```
const int MAXN = 1e6 + 5;
int pre[MAXN], k, n;//开了 #define int long long
string tmp, s;
void slove() {
    cin >> k;
    cin >> tmp; s = "?" + tmp;//我喜欢下标1开始的
    n = tmp.length();
    for (int i = 1; i <= n; i++)pre[i] = pre[i - 1] + (s[i] == '1');
    int ans = 0;
    for (int i = 1; i <= n; i++) {
        ans += upper_bound(pre + i, pre + 1 + n, k + pre[i - 1]) -
            lower_bound(pre + i, pre + 1 + n, k + pre[i - 1]);
```

```
    }
    cout << ans << endl;
}
```