1. 缺失的最小正整数

有n（不超过200）个绝对值比10000小的整数。请你设计C/C++程序，找出其中最小的没有出现的正整数。

输入格式（2行）

第一行，一个正整数n（代表整数的个数）

第二行，n个整数

输出格式

一行，一个正整数（代表缺失的最小正整数）

思路一：排序

用写过的my_sort()函数对数组a进行从小到大排序，然后遍历有序数组，同步计数和对比。

```
int res = 1;
for (int i = 0; i < n; i++){
    if (a[i] == res) ++res;
    else if (a[i] > res) break;
}
cout << res << endl;
return 0;
```

思路二：哈希数组

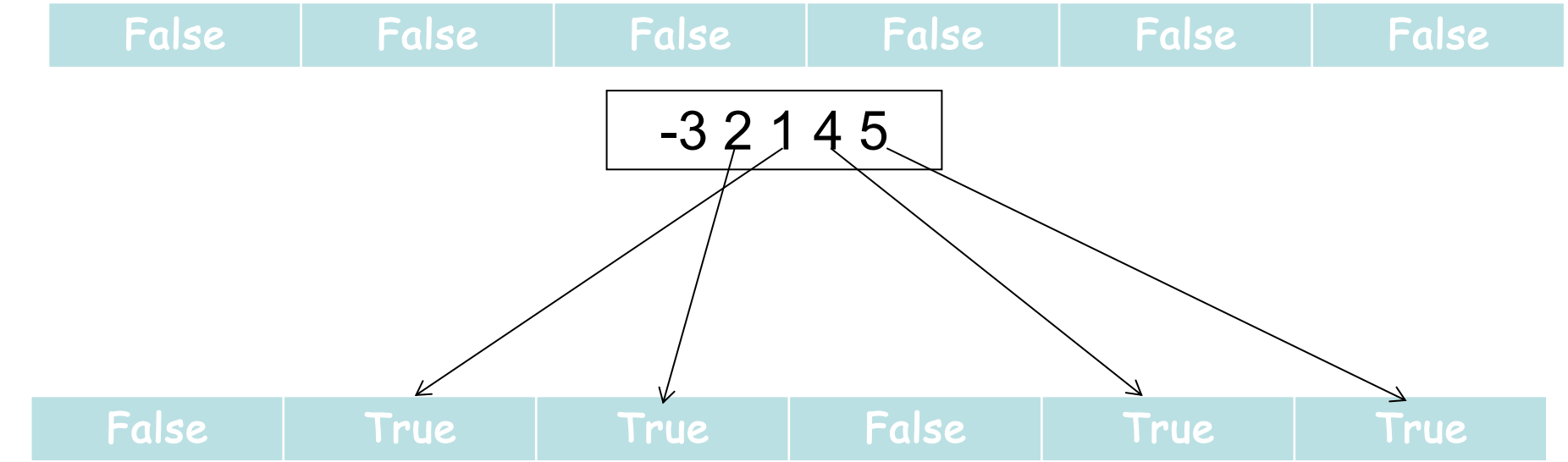
用一个哈希数组记录[0,10000)以内出现的整数，

然后遍历哈希数组找到最小没有出现的整数。

输入：
`5`
`-3` `2` `1` `4` `5`

预期输出：
`3`

哈希数组：



思路二：哈希数组

```
#define N 200
#define MAX 10000

int a[N];
int cnt[MAX];

int main()
{
    int n, i;
    scanf("%d", &n);
    for(i = 0; i < n; ++i)
        scanf("%d", &a[i]);
    ……
    return 0;
} // find first lack pos integer
```

```
for (int j = 0; j < i; j++)
{
    if (a[j] > 0)
        cnt[a[j]] = 1;
}
for (i = 1; i <= MAX; i++)
{
    if (cnt[i] == 0)
    {
        printf("%d\n", i);
        break;
    }
}
```

2. 抽屉里的金币

亚南极海域的一艘船上有一排装了机关的透明抽屉，每个抽屉都存有金币。一只小蓝鲸跳到船上，它发现一旦拿了某个抽屉的金币，相邻的两个抽屉就会自动锁死，不能从锁死的抽屉拿金币。请你设计C/C++程序，计算小蓝鲸最多能拿到多少金币。

输入格式（2行）

第一行，一个整数 n（1≤n≤100，代表抽屉数）

第二行，n 个正整数（依次代表第1个~第n个抽屉的金币数，用空格分隔）

输出格式（1行）

一个正整数（代表小蓝鲸最多能拿到的金币数）

思路一：递归+查表

分析：
- ✓ 该游戏是一个决策性问题
- ✓ 每次决策都可以选择 拿 或 不拿 当前抽屉的金币
- ✓ 每次决策是否最大化获益只和 前两次 的结果以及 当前抽屉的金币数 有关
- ✓ 从前往后顺序地处理

看上去，递归可以解决！

```
int m(int a[], int n) {
    if (n == 1) {
        return a[0];
    }
    if (n == 2) {
        if (a[0] > a[1])
            return a[0];
        return a[1];
    }
    if (a[n - 1] + m(a, n - 2) > m(a, n - 1))
        return a[n - 1] + m(a, n - 2);
    return m(a, n - 1);
}
```

思路一：递归+查表

```
int m(int a[], int n) {
    if (n == 1) {
        return a[0];
    }
    if (n == 2) {
        if (a[0] > a[1])
            return a[0];
        return a[1];
    }
    if (a[n - 1] + m(a, n - 2) > m(a, n - 1))
        return a[n - 1] + m(a, n - 2);
    return m(a, n - 1);
}
```

朴素递归存在大量重复计算

```
n=1, rest=8
n=1, rest=8
n=2, rest=10
n=1, rest=8
n=3, rest=17
n=1, rest=8
n=4, rest=17
n=1, rest=8
n=2, rest=10
n=1, rest=8
n=3, rest=17
n=1, rest=8
n=1, rest=8
n=2, rest=10
n=1, rest=8
n=3, rest=17
n=1, rest=8
n=5, rest=18
n=1, rest=8
```

思路一：递归+查表

```
int rest[105] = {0};

int m(int a[], int n) {
    if (rest[n-1] > 0)
        return rest[n-1];
    if (n == 1) {
        rest[0] = a[0];
        return a[0];
    }
    if (n == 2) {
        if (a[0] > a[1]) {
            rest[0] = a[0];
            return a[0];
        }
        rest[1] = a[1];
        return a[1];
    }
    if (a[n - 1] + m(a, n - 2) > m(a, n - 1)) {
        rest[n-1] = a[n - 1] + m(a, n - 2);
        return rest[n-1];
    }
    rest[n-1] = m(a, n - 1);
    return rest[n-1];
}
```

引入一个数组来存储计算过的函数值！

思路二：动态规划

1　　2　　3　　1　　3+1

2　　7　　4　　6　　9　　9+7

2　　5　　4　　6　　6+5

尽量先找最大值?

2　　5　　4　　5 ?

分析：

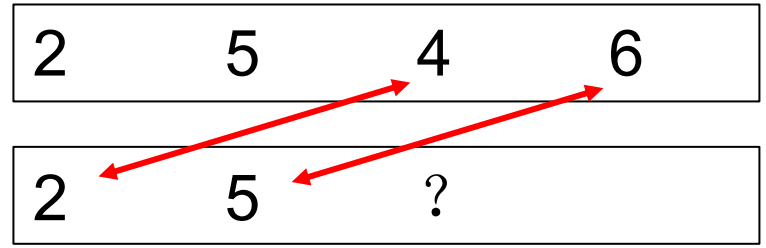
- ✓ 该游戏是一个决策性问题
- ✓ 每次决策都可以选择 拿 或 不拿 当前抽屉的金币
- ✓ 每次决策是否最大化获益只和 前两次 的结果以及 当前抽屉的金币数 有关
- ✓ 从前往后顺序地处理

思路二：动态规划

```
#define N 20
int main()
{
      int drawer[N];
      int n;
      scanf("%d", &n);
      for(int i=0; i < n; ++i)
            scanf("%d", &drawer[i]);

      printf("%d \n", CoinsDP(drawer, n));

      return 0;
 }
```

思路二：动态规划

```
int dp[N] = {0};

int CoinsDP(int drawer[], int n)
{
      dp[0] = drawer[0];
      dp[1] = myMax(dp[0], drawer[1]);
      int temp;
      for(int i=2; i < n; ++i)
      {
            temp = dp[i-2] + drawer[i];
            dp[i] = myMax(temp, dp[i-1]);
      }
      return dp[n-1];
}
```



```
printf("%d \n", Coins(drawer, n));
```

```
int Coins(int drawer[], int n)
{
      int pre = drawer[0];
      int cur = myMax(pre, drawer[1]);
      int next;
      int temp;
      for(int i=2; i < n; ++i)
      {     temp = pre + drawer[i];
            next = myMax(temp, cur);
            pre = cur;
            cur = next;
      }
      return next;
}
```

2 5 4 6

2

5

6

11

注意逻辑和次序

```
inline int myMax(int x, int y)
{
      if(x >= y) return x;
      else return y;
}
```

3. 机器猫和老鼠

在一列火车上，有一只来回巡视的机器猫和一只机灵的老鼠。火车由n(2⩽n⩽50)节车厢组成，用编号1~n表示从火车头**到**火车尾**的每一节车厢。**

机器猫的运动方式如下：

- 在每1分钟里，机器猫都只会笨笨地沿着一个给定的方向从一节车厢移动到相邻的另一节车厢；
- 当机器猫到达火车头时，它会**改变它的运动方向**，即它下1分钟会移动到2号车厢并且朝着火车尾方向移动；
- 同理，当机器猫到达火车尾，它会**改变运动方向** ，即它下1分钟会移动到n-1号车厢并且朝着火车头的方向移动；如此反复。

火车在每1分钟内都会有一个状态：或者是处在行驶状态用0表示，或者是处在停止状态，用1表示。(即到站停靠)

老鼠的运动方式如下：

- 每1分钟里，如果当前火车是行驶状态，那么老鼠只能移动到它相邻的车厢，或者待在原来的车厢不动。
- 每1分钟里，如果当前火车是停止状态，那么老鼠会在先下车，然后选择1~n中的任意一节车厢上车。

运动顺序：每一分钟内，如果火车是行驶状态：老鼠先进行移动，然后机器猫再移动。如果火车是停止状态，老鼠先下车，然后机器猫移动，最后老鼠再上车。同时，机智的老鼠利用它的耗子尾汁可以知道每一分钟内，机器猫的位置和运动方向。

如果在某一时刻机器猫和老鼠在同一节车厢，那么老鼠会被机器猫捉住。

3. 机器猫和老鼠

问：在给定火车在t（1≤t≤100）分钟内的运动状态序列，老鼠和机器猫的初始位置，以及机器猫的初始方向的情况下，假设老鼠每一分钟都会选择最好移动方式**（最好的移动方式是唯一的，可以手动模拟一下试试）来**避免被捉**或者**尽可能地拖延被捉**，老鼠在t分钟内能否被机器猫捉到；如果能，最后在哪节车厢被捉?**

提示：需要动动脑筋，使用学过的知识可以解决，但是需要细心，思考其中的细节。

输入格式：

第一行：三个正整数（分别代表车厢的节数 n、老鼠一开始所在的车厢号 m 和机器猫一开始所在的车厢号 k，用空格分隔，2≤n≤50，1≤m,k≤n，m≠k)

第二行：一个整数（只有 -1 和 1 两种取值，-1 代表机器猫起初是向火车头方向运动，1 代表向火车尾方向运动）

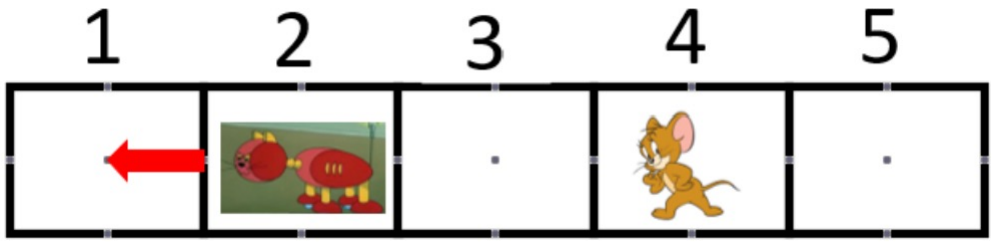
第三行：一个正整数（代表火车运动状态序列的分钟数 t，1≤t≤100）

第四行：t 个整数（代表 t 分钟内每一分钟火车的状态，只有 0 和 1 两种取值，0 表示行驶状态，1 表示停止状态，用空格分隔）

输出格式：

如果机器猫不能捉住老鼠，输出-1；如果能捉住老鼠，输出老鼠被捉住时所在的车厢号

思路：模拟（猫鼠尽可能远离）



- 输入：
  - 5 4 2
  - -1
  - 4
  - 0 0 0 1
- 输出：-1

- 输入：
  - 5 3 2
  - -1
  - 7
  - 0 0 0 1 0 0 1
- 输出：
  - -1

- 输入：
  - 3 2 1
  - 1
  - 4
  - 0 0 0 1
- 输出：
  - 3

```
int s[101] = { -1 };
……
Move(n, m, k, d, t, s);
```

```
void Move(int n, int m, int k, int d, int t, int s[])
{
        for (int i = 0; i < t; ++i) {
                if (m == k) {
                        cout << m << endl;
                        return;
                } //抓个正着
                if (k == n && d == 1 ||  k == 1 && d == -1) {
                        d = -d;
                }
                if (s[i] == 0) {
                        if (m > 1 && m < n) {
                                m = (m < k ? m - 1 : m + 1); //老鼠是否在前
                        } //老鼠要远离猫
                        k = k + d;
                } //行驶中
                else {
                        ……
                } //车停了
        }
        cout << -1 << endl;
        return;
}
```

```
void Move(int n, int m, int k, int d, int t, int s[])
{
        for (int i = 0; i < t; ++i) {
                ……
                if (k == n && d == 1 ||  k == 1 && d == -1) {
                        d = -d;
                }
                if (s[i] == 0) {
                        ……
                } //行驶中
                else {
                        k = k + d;
                        if (k == 1 || k == n) {//猫在头部 老鼠就到尾部去
                                m = n + 1 - k;     //猫在尾部 老鼠就到头部去
                        }
                        else {
                                m = (d < 0 ? n : 1);//猫向前 老鼠就到尾部去
                        }                                   //猫向后 老鼠就到头部去
                } //车停了
        }
        cout << -1 << endl;
        return;
}
```