题目：

给定一个长度为n的整数序列，请找出最长的不包含重复的数的连续区间，输出它的长度。

输入格式:

第一行包含整数n。

第二行包含n个整数（均在0~100000范围内），表示整数序列。

输出格式:

共一行，包含一个整数，表示最长的不包含重复的数的连续区间的长度。数据范围:

1≤n≤1000001≤n≤100000

输入样例:：

5

1 2 2 3 5

输出样例：

3

思路：

暴力法：

当然可以用暴力法：对每个 i 和 j 都遍历一遍，对每个 i 和 j 都check一下中间的数据是否满足给定的条件。这样的时间复杂度是O(n^2)；数据稍微大点就会超时。

代码如下：

for (int i = 0; i < n; i++)

for (int j = 0; j <= i; j++)

if (check(v1,j, i) == 0)//检查 i 和 j 之间是否有重复的数字

res = max(res, i - j + 1);

//check函数

int check(vector<int>& v1, int l, int r)

{

for (int i = l+1; i <=r ; i++)

for (int j = l; j < i; j++)

if (v1[i] == v1[j])

return 1;

return 0;

}

双指针法一：

仔细考虑暴力法就会发现，暴力法在解题时有很多地方是重复计算了 ( i 指针在 j 指针的后面，i是遍历的整个数组的，j 是遍历 0 到 i 的)：

比如 j = 0，i = 5，此时发现 i，j 是满足题解条件的；那么后面的 j = 1到5，i = 5 就不用计算了，肯定是满足条件的。

所以引出了双指针法：还是上面的例子，双指针法就是说，既然发现 j = 0，i = 5满足题解条件，那就不用计算 j = 1到5，i = 5了，直接计算 j = 1，i = 6，如果不满足条件，那就计算 j = 2，i = 6，然后接着计算。

这样就是 i 和 j 指针都是从前移到后，也就是计算2n次。时间复杂度是O(2n)。

核心代码如下：（但是还会超时）。

for (int i = 0, j = 0; i < n; i++)

{

while (j <= i)

{

if (check(v1, j, i) == 0)

{

res = max(res, i - j + 1);

break;

}

else

j++;

}

}

//找得到重复的数返回1

int check(vector<int>& v1, int l, int r)

{

for (int i = l + 1; i <= r; i++)

for (int j = l; j < i; j++)

{

if (v1[i] == v1[j])

return 1;

}

return 0;

}

双指针法二（最终版）：

但是上面代码还是超时，为什么呢？因为check函数写的不好，循环太多，直接是暴力计算找重复数字的，显然不好。

所以引出一个新的check方法：对于寻找是否有重复数字，一般用hash，没人用暴力。所以用hash就可以计算。

但是这道题还有一种计算方法：

用一个辅助数组S保存原数组V1每个元素存在的次数，和hash类似。

比如说 V1 = {1,2, 2, 3, 5 }。那 S就是 {0,1,2,1,0,1 }。S[V1[i]]表示的是V1[i]的个数。

此处我们用S数组只保存 j 和 i 指针之间的数的个数。

算法思路： 如果j = 0，i = 5，此时检查S数组元素都是 <=1的。那下一步的情况就是 i 。i之后将S数组更新，只需要检查S[v1[i]]元素是不是比1大即可，因为随着 i 的递增，S数组中变化的只有S[v1[i]]元素。

如果检查S[v1[i]]元素发现该元素比 1 大。那说明 j 指针和 i 指针之间有某个元素出现了两次。所以 i 指针保持不动， j 指针往后移动( j 指针不可能往前移动的，上次j指针往后移动就是因为 j 和 i之间有重复元素，这一往前移动肯定有重复元素)。j 指针往后移动之前需要先更新S数组，即进行 S[v1[j]]– 操作。然后 j 指针再往后移动。移动之后只需要检查 i 指针对应的S[v1[i]]元素是否大于1即可。（因为 j 指针移动之后只有两种情况，1.重复元素刚好没了，则S[v1[i]]肯定==1；2.重复元素还在，那S[v1[i]]==2，需要 j 继续往后移动 )。等S[v1[i]]==1 时，说明 j 和 i 之间已经没有重复元素了，可以更新res值，然后 i++。

核心代码：

for (int i = 0,j = 0; i < n; i++)

{

S[v1[i]]++;

while ( S[v1[i]] > 1) --S[v1[j++]];

res = max(res, i - j + 1);

}

代码实现：

#include<iostream>

#include <vector>

#include <algorithm>

#define N 100010

using namespace std;

int main()

{

int n;

cin >> n;

vector<int> v1(n,0);

for (int i = 0; i < n; i++)

cin >> v1[i];

vector<int> S(N,0);

int res = 0;

for (int i = 0,j = 0; i < n; i++)

{

S[v1[i]]++;

while ( S[v1[i]] > 1) --S[v1[j++]];

res = max(res, i - j + 1);

}

cout << res;

return 0;

}

作者：bbk

链接：https://www.acwing.com/solution/content/23474/

来源：AcWing

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

const int N = 100010;

int n;

int main()

{

cin >> n;

vector<int> v(n,0); //优化空间为On

for(int i = 0;i < n; i ++) cin >> v[i]; //输入

vector<int> tmp(N,0); //辅助数组,注意这里上限取为值的上限

int res = 0;

for(int i = 0,j = 0;i < n;i ++){

tmp[v[i]] ++ ; // 频率标记数组，标记当前序列出现的数字的出现次数

while(tmp[v[i]] > 1) --tmp[v[j++]]; // 如果频率超1，就要把重复元素前的全清掉--

res = max(res,i - j + 1);

}

cout << res;

return 0;

}