```
/*
给定一个整数数组和一个整数 k，判断数组中是否存在两个不同的索引 i 和 j，使得 nums [i] = nums [j]，并且 i 和 j 的差的绝对值最大为 k。

示例 1:

输入: nums = [1,2,3,1], k = 3
输出: true

示例 2:

输入: nums = [1,0,1,1], k = 1
输出: true

示例 3:

输入: nums = [1,2,3,1,2,3], k = 2
输出: false

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/contains-duplicate-ii
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
*/
```

分析:根据题意,只要满足两个相等的元素之间的索引差值$\leq k$即可.

- 方法一:滑动窗口法,时间复杂度$O(n*k)$,空间复杂度$O(n)$,其中n是数组长度,但执行时超时了.
- 方法二:map数据结构保存法,时间复杂度$O(n)$(未包含map操作的时间复杂度),空间复杂度$O(n)$.
- 方法二(优化):实际上不需要用vector来保存每次出现的位置,只要保存最近一次出现的次数即可.
- todo:其实我不会计算map相关的时间复杂度和空间复杂度.

方法一:C++_滑动窗口法

```
class Solution
{

    private:
        /* (left,right]*/
        bool haveSame(vector<int>& nums,int left , int right)
        {
            int     i       = 0     ;
            for(i = left+1;i<=right;i++)
            {
                if(nums[left]==nums[i])
                {
                    return true;
                }
            }
            return false;
        }
```

```
18
19      public:
20          bool containsNearbyDuplicate(vector<int>& nums, int k)
21          {
22              int right = 0           ;
23              int i     = 0           ;
24              int len   = nums.size() ;
25              for(i = 0;i<len;i++)
26              {
27                  right = (i+k < len)? (i+k) : (len-1);
28                  if(haveSame(nums,i,right))
29                  {
30                      return true;
31                  }
32              }
33
34              return false;
35
36          }
37  };
38
39  /*
40  22 / 23 个通过测试用例
41      状态：超出时间限制
42
43  */
```

方法二:C++_map数据结构保存法

```
1   class Solution
2   {
3
4       public:
5           bool containsNearbyDuplicate(vector<int>& nums, int k)
6           {
7               if(nums.size()< 2)
8               {
9                   return false;
10              }
11              else
12              {
13                  map<int,vector<int>>  mivi;
14                  for(int i = 0 ; i < nums.size();i++)
15                  {
16                      if(mivi.count(nums[i]))
17                      {
18                          mivi[nums[i]].push_back(i);
19                          if(mivi[nums[i]][mivi[nums[i]].size()-1]-
    mivi[nums[i]][mivi[nums[i]].size()-2] <=k)
20                          {
21                              return true;
22                          }
23                      }
24                      else
25                      {
26                          vector<int> vi;
```

```
                        vi.push_back(i);
                        mivi[nums[i]] = vi;
                    }
                }
            }
            return false;

        }
};

/*
执行结果:
通过
显示详情
执行用时 :64 ms, 在所有 cpp 提交中击败了25.25% 的用户
内存消耗 :22.4 MB, 在所有 cpp 提交中击败了5.03%的用户
*/
```

方法三:C++_map数据结构保存法(优化存储)

```
class Solution
{

    public:
        bool containsNearbyDuplicate(vector<int>& nums, int k)
        {
            if(nums.size()< 2)
            {
                return false;
            }
            else
            {
                map<int,int>  mii;
                for(int i = 0 ; i < nums.size();i++)
                {
                    if(mii.count(nums[i]))
                    {
                        if(i - mii[nums[i]] <=k)
                        {
                            return true;
                        }
                        else
                        {
                            mii[nums[i]] = i;
                        }
                    }
                    else
                    {
                        mii[nums[i]] = i;
                    }
                }
            }
            return false;

        }
};
```

```
37
38  /*
39  执行结果:
40  通过
41  显示详情
42  执行用时 :56 ms, 在所有 cpp 提交中击败了39.80% 的用户
43  内存消耗 :15.3 MB, 在所有 cpp 提交中击败了28.08%的用户
44  */
```

AlimyBreak
2019.11.05