

二分查找算法

在有序数组中查找值

数组和函数之间没有本质差别 数组消耗空间，函数消耗时间 因此在底层数组和函数没有区别，也就意味着二分查找算法可以应用于函数 不仅可以应用在有序数组上，也可以作用在单调函数上

二分查找泛型情况

头尾指针重合位置就是所要查找的位置

0	0	0	0	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---

- 查找第一个1

```
while(head < tail)
{
    mid = (head + tail) / 2;
    if(arr[mid] < target) head = mid + 1;
    else tail = mid;
}
return head;
```



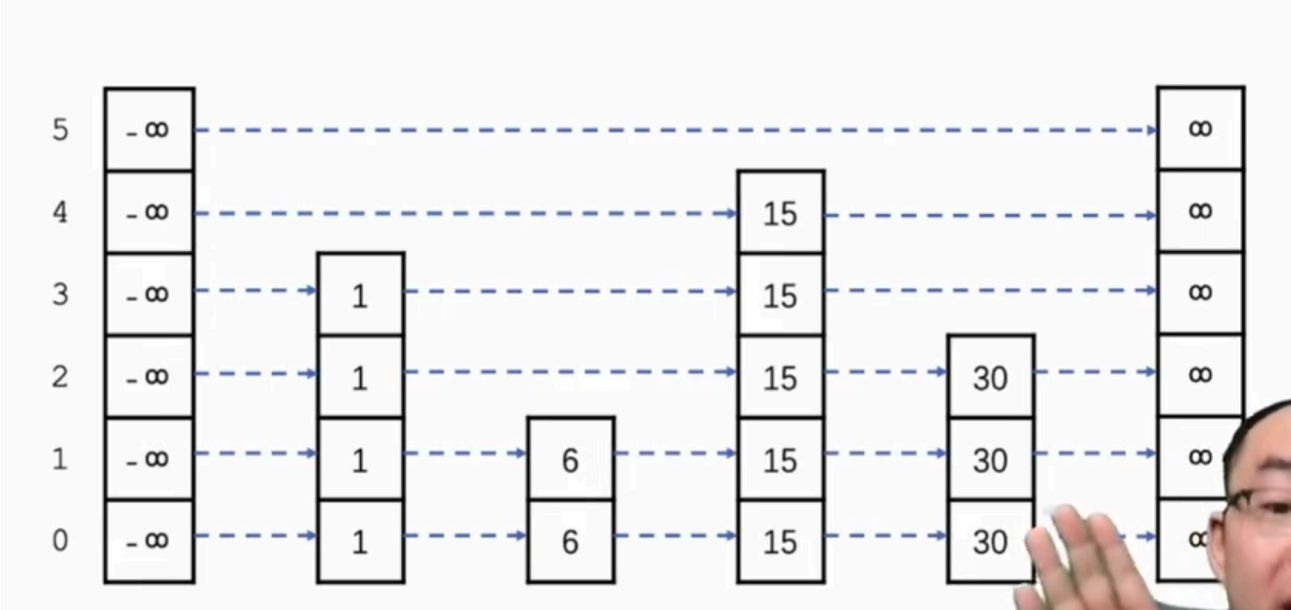
1	1	1	1	1	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---

- 查找最后一个1

```
while(head < tail)
{
    mid = (head + tail + 1) / 2; //不加1则会陷入死循环
    if(arr[mid] < target) head = mid;
    else tail = mid - 1;
}
```

```
    }  
    return head;
```

跳跃表 (Skiplist)



- 建立公共溢出区：本质上是一个查找数据结构
- 链式地址法：将数组中的元素变成链表
- 所需要的存储空间非常巨大


布隆过滤器

- 底层是数组
- 使用过的存储空间标记为1，未使用过标记为0
- 使用多个哈希函数计算哈希值
- 对于元素出现过的判断是概率性判断，没法做精确的判断
- 可用于爬虫爬取网页的场景中

哈希表，存储空间与元素数量强相关，布隆过滤器，存储空间与元素数量弱相关


两数之和


1. 两数之和

已解答 

简单

 相关标签

 相关企业

 提示

Ax

给定一个整数数组 `nums` 和一个整数目标值 `target`，请你在该数组中找出 **和为目标值** `target` 的那 **两个** 整数，并返回它们的数组下标。

你可以假设每种输入只会对应一个答案。但是，数组中同一个元素在答案里不能重复出现。

你可以按任意顺序返回答案。

示例 1:

输入: `nums = [2,7,11,15]`, `target = 9`

输出: `[0,1]`

解释: 因为 `nums[0] + nums[1] == 9`，返回 `[0, 1]`。

示例 2:

- 用 `unordered_map` 模拟哈希表
- 用当前遍历到的数据去哈希表内查找是否有数字与其匹配，有则插入数组，没有则将当前数字插入哈希表

```

class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map <int,int> h;
        vector<int> ret(2);
        for(int i = 0;i < nums.size();i++)
        {
            if(h.find(target - nums[i]) != h.end())
            {
                ret[0] = h[target - nums[i]];
                ret[1] = i;
                break;
            }
            h[nums[i]] = i;
        }
        return ret;
    }
};

```

- 二分查找(在有序数组中查找)
- 对下标数组进行排序
- 从前到后遍历数组，对后半部分采用二分查找，查找是否有数字与遍历到的数字匹配

```

class Solution {
public:
    int find(vector<int> &nums,vector<int> &ind,int target,int i)
    {
        int head = i,tail = nums.size() - 1,mid;
        while(head <= tail)
        {
            mid = (tail + head) / 2;
            if(nums[ind[mid]] == target)
                return mid;
            if(nums[ind[mid]] < target)
                head = mid + 1;
            else
                tail = mid - 1;
        }
        return 0;
    }
    vector<int> twoSum(vector<int>& nums, int target) {
        vector<int> ret(2);
        vector<int> ind(nums.size());
        for(int i = 0;i < nums.size();i++) ind[i] = i;
        sort(ind.begin(),ind.end(),[&](int i,int j)->bool

```

```

        {
            return nums[i] < nums[j];
        });
        for(int i = 0; i < nums.size();i++)
        {
            if(find(nums,ind,target - nums[ind[i]],i + 1))
            {
                ret[0] = ind[i];
                ret[1] = ind[find(nums,ind,target - nums[ind[i]],i + 1)];
                break;
            }
        }
        return ret;
    }
};

```

搜索插入位置

35. 搜索插入位置

已解答 

简单

 相关标签

 相关企业

Ax

给定一个排序数组和一个目标值，在数组中找到目标值，并返回其索引。如果目标值不存在于数组中，返回它将会被按顺序插入的位置。

请务必使用时间复杂度为 $O(\log n)$ 的算法。

示例 1:

输入: `nums = [1,3,5,6], target = 5`

输出: `2`

示例 2:

输入: `nums = [1,3,5,6], target = 2`

输出: `1`

- 也可以用二分算法的泛型情况来写

```

class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
        int head = 0, tail = nums.size() - 1, mid;
        while(head <= tail)
        {

```

```

        mid = (head + tail) / 2;
        if(nums[mid] == target) return mid;
        if(nums[mid] < target) head = mid + 1;
        else tail = mid - 1;
    }
    return head ; //到这说明没有找到target, 返回插入位置
};


```


存在重复元素

217. 存在重复元素

已解答 

简单

 相关标签

 相关企业

Ax

给你一个整数数组 `nums` 。如果任一值在数组中出现 **至少两次**，返回 `true` ；如果数组中每个元素互不相同，返回 `false` 。

- 个人做法：
- 先对下标数组排序，然后从前往后遍历数组，看是否出现过相同数字

```

class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        int head ,tail ,mid;
        vector<int> ind(nums.size());
        for(int i = 0;i < nums.size();i++) ind[i] = i;
        sort(ind.begin(),ind.end(),[&](int i ,int j)->bool{
            return nums[i] < nums[j];
        });
        for(int i = 0;i < nums.size();i++)
        {
            head = i + 1,tail = nums.size() - 1;
            while(head <= tail)
            {
                mid = (head + tail) / 2;
                if(nums[ind[mid]] == nums[ind[i]]) return true;
                if(nums[ind[mid]] < nums[ind[i]]) head = mid + 1;
                else tail = mid - 1;
            }
        }
        return false;
    }
};

```

```
    }  
};
```

- 船长做法：
- 使用哈希表来辅助，查找到则返回true，未查到则插入

```
class Solution {  
public:  
    bool containsDuplicate(vector<int>& nums) {  
        unordered_set<int> s;  
        for(auto x:nums)  
        {  
            if(s.find(x) != s.end()) return true;  
            s.insert(x);  
        }  
        return false;  
    }  
};
```

两个数组的交集

349. 两个数组的交集

已解答 (

简单

🔖 相关标签

🔒 相关企业

Ax

给定两个数组 `nums1` 和 `nums2`，返回 它们的 [交集](#)。输出结果中的每个元素一定是 **唯一** 的。我们可以 **不考虑输出结果的顺序**。

示例 1:

输入: `nums1 = [1,2,2,1]`, `nums2 = [2,2]`

输出: `[2]`

示例 2:

输入: `nums1 = [4,9,5]`, `nums2 = [9,4,9,8,4]`

输出: `[9,4]`

解释: `[4,9]` 也是可通过的

- 将某个数组插入哈希表
- 利用set特性可以剔除相同的数字
- 查找到就插入结果数组

```
class Solution {
public:
    vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {
        vector<int> ret;
        unordered_set<int> h;
        for(auto x: nums1) h.insert(x);
        for(auto x: nums2)
        {
            if(h.find(x) == h.end()) continue;
            ret.push_back(x);
            h.erase(x);
        }
        return ret;
    }
};
```

无重复字符的最大子串

3. 无重复字符的最长子串

已解答 ✓

中等

🏷 相关标签

🏢 相关企业

Ax

给定一个字符串 `s`，请你找出其中不含有重复字符的 **最长子串** 的长度。

示例 1:

输入: `s = "abcabcbb"`

输出: 3

解释: 因为无重复字符的最长子串是 `"abc"`，所以其长度为 3。

示例 2:

输入: `s = "bbbbbb"`

输出: 1

解释: 因为无重复字符的最长子串是 `"b"`，所以其长度为 1。

示例 3:

输入: `s = "pwwkew"`

输出: 3

解释: 因为无重复字符的最长子串是 `"wke"`，所以其长度为 3。
请注意，你的答案必须是 **子串** 的长度，`"pwke"` 是一个子序列，不是子串。

哈希表 + 滑动窗口

- 用ans记录子串长度
- 在哈希表中找到相同字符就更新ans，并移动i，当j到字符串末尾就返回(说明是最大值)
- 遇到相同字符还要更新map中字符的位置，否则i移动会发生错误
- 要确保头指针一直向后移动

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        unordered_map<char,int> h;
        int ans = 0;
        for(int i = 0; i < s.size(); i++)
        {
            for(int j = i; j < s.size(); j++)
            {
```

```

        if(j == s.size() || h.find(s[j]) != h.end())
        {
            ans = max(ans, j - i);
            if(j == s.size()) return ans;
            i = max(i, h[s[j]] + 1);
        }
        h[s[j]] = j;
    }
}
return ans;
}
};

```

滑动窗口+二分算法

- 二分算法的泛型情况，查找最后一个1

给定一个字符串 s ，请你找出其中不含有重复字符的 **最长子串** 的长度。

长度: 1 2 3 ... 2 $l+1$ $l+2$ $l+3$... n

例 1: 能否: 1 1 1 ... 1 0 0 0 ... 0

- 使用ans来记录字符的个数，使用k来计算不同字符的个数

```

class Solution {
public:
    bool check(string &s, int l)
    {
        int ans[256] = {0}, k = 0;
        for(int i = 0; i < s.size(); i++)
        {
            ans[s[i]] += 1;
            if(ans[s[i]] == 1) k += 1;
            if(i >= l)
            {
                ans[s[i-l]] -= 1;
                if(ans[s[i-l]] == 0) k -= 1;
            }
            if(k == l) return true;
        }
        return false;
    }

    int lengthOfLongestSubstring(string s) {
        int head = 0, tail = s.size(), mid;
        while(head < tail)
    }
}

```

```
        {
            mid = (head + tail + 1) / 2;
            if(check(s,mid)) head = mid;
            else tail = mid - 1;
        }
        return head;
    }
};
```

寻找两个正序数组的中位数



4. 寻找两个正序数组的中位数

已解答

困难

相关标签

相关企业

Ax

给定两个大小分别为 m 和 n 的正序（从小到大）数组 `nums1` 和 `nums2`。请你找出并返回这两个正序数组的 **中位数**。

算法的时间复杂度应该为 $O(\log(m+n))$ 。

示例 1:

输入: `nums1 = [1,3]`, `nums2 = [2]`

输出: `2.00000`

解释: 合并数组 = `[1,2,3]` , 中位数 `2`

示例 2:

输入: `nums1 = [1,2]`, `nums2 = [3,4]`

输出: `2.50000`

解释: 合并数组 = `[1,2,3,4]` , 中位数 $(2 + 3) / 2 = 2.5$

- 将问题化成求两个正序数组中第 k 大的值
- 使用二分算法的思维，各取两数组中 $k/2$ 个元素，通过比较去除 $k/2$ 个元素
- 将问题化成求解剩余数组中第 $k/2$ 大的值
- 使用递归求解

```
#include <inttypes>
class Solution {
public:
```

```

int findk(vector<int>& n1,int ind1,vector<int>& n2,int ind2,int k)
{
    int n = n1.size(),m = n2.size();
    if(k == 1)
    {
        int a = (ind1 == n)?INT32_MAX:n1[ind1];
        int b = (ind2 == m)?INT32_MAX:n2[ind2];
        return min(a,b);
    }
    if(n == ind1) return n2[k - 1];
    if(m == ind2) return n1[k - 1];
    int cnt1 = min(k / 2,n - ind1);
    int cnt2 = min(k - cnt1,m - ind2);
    cnt1 = k - cnt2;
    if(n1[cnt1 + ind1 - 1] <= n2[cnt2 + ind2 - 1])
        return findk(n1,ind1 + cnt1,n2,ind2,k - cnt1);
    else
        return findk(n1,ind1,n2,ind2 + cnt2,k - cnt2);
}

double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2)
{
    int n = nums1.size(),m = nums2.size();
    if((n + m) % 2 == 1) return findk(nums1,0,nums2,0,(n + m) / 2 + 1);
    double a = findk(nums1,0,nums2,0,(n + m) / 2);
    double b = findk(nums1,0,nums2,0,(n + m) / 2 + 1);
    return (a + b) / 2.0;
}
};

```

奶牛围栏

时间限制:1 s 空间限制:256 MB



#244. 奶牛围栏

描述

提交

自定义测试

题解视频

上一题

下一题

统计

题目描述

约翰打算建一个围栏来圈养他的奶牛。作为最挑剔的兽类，奶牛们要求这个围栏必须是正方形的，而且围栏里至少要有 C ($1 \leq C \leq 500$) 个草场，来供应她们的午餐。

约翰的土地上共有 N ($C \leq N \leq 500$) 个草场，每个草场在一块1x1的方格内，而且这个方格的坐标不会超过 10000。有时候，会有多个草场在同一个方格内，那他们的坐标就会相同。

现求围栏的最小边长为多少。

输入

第一行输入两个数 C, N 。

接下来 N 行每行两个数，表示每个草场的坐标 X_i, Y_i 。

输出

输出围栏的最小边长。

如图, 显而易见, 二分算法的泛型情况



正方形边长 L	0	1	2	...	$L-1$	L	...	N
是否 $\geq C$	0	0	0	...	0	1	...	1

- 对于二维空间, 先对x坐标进行扫描, 再对y坐标进行扫描, 判断边长为 l 的正方形内草场数量是否大于等于 C

```
#include <bits/stdc++.h> //hzoj244 奶牛围栏
using namespace std;
struct point
{
    int x,y;
}arr[505];

int temp[505];

bool cmp(const point &a,const point &b)
{
    if(a.x != b.x) return a.x < b.x;
    return a.y < b.y;
}

int check_y(point *arr,int n,int c,int b,int e,int l)
{
    int cnt = 0;
    for(int i = b;i <= e;i++) temp[cnt++] = arr[i].y;
    sort(temp,temp + cnt);
    for(int i = c - 1;i < cnt;i++)
    {
        if(temp[i] - temp[i - c + 1] < l) return 1;
    }
    return 0;
}

int check(point *arr,int n,int c,int l)//l为正方形边长
{
    int j = 0;
    for(int i = 0;i < n;i++)
    {
        while(arr[i].x - arr[j].x >= l) j += 1;
        if(i - j + 1 < c) continue;
        if(check_y(arr,n,c,j,i,l))
            return 1;
    }
}
```

```

    }
    return 0;
}

int bs(int l,int r,point *arr,int n,int c)
{
    int mid = 0;
    while(l < r)
    {
        mid = (l + r) / 2;
        if(check(arr,n,c,mid)) r = mid;
        else l = mid + 1;
    }
    return l;
}

int main()
{
    int c,n;
    cin >> c >> n;
    for(int i = 0; i < n;i++)
        cin >> arr[i].x >> arr[i].y;
    sort(arr,arr + n,cmp);
    cout << bs(0,10000,arr,n,c) << endl;
    return 0;
}

```

check_y也可以写成如下形式

```

int check_y(point *arr,int n,int c,int b,int e,int l)
{
    int j = 0,cnt = 0;
    for(int i = b;i <= e;i++) temp[cnt++] = arr[i].y;
    sort(temp,temp + e - b + 1);
    for(int i = 0;i <= e - b;i++)
    {
        while(temp[i] - temp[j] >= l) j += 1;
        if(i - j + 1 >= c) return 1;
    }
    return 0;
}

```