□分查找算法

在有序数组中查找值

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

□分查找泛型情况

头尾指针重合位置就是所要查找的位置 alt text

- □找第一个1
-
  ```
  while(head < tail)
  {
      mid = (head + tail) / 2;
      if(arr[mid] < target) head = mid + 1;
      else tail = mid;
  }
  return head;
  ```

alt text

- □找最后一个1
-
  ```
  while(head < tail)
  {
      mid = (head + tail + 1) / 2; //□加1则会陷入死循环
      if(arr[mid] < target) head = mid;
      else tail = mid - 1;
  }
  return head;
  ```

跳跃表

每一个节点都有不同的高度，处于同一个高度的节点被串成一个链表 alt text

- □找：待□找元素为x，从左上角的节点开始□找，若当前节点的下一个节点的值比x小，则往后走一位，若比x大，则向下走一位
- 插入：找到待插入节点的前一个节点，按照□找的规则移动，当不能再向下移动时，说明已经□到插入的节点的前一个节点

哈希表与布隆过滤器

哈希表

- 时间复杂度为O(1)
- 底层是一个数组，由哈希函数计算出来的哈希值来分配存储位置，数组中的每一个元素是一个特定的数据结构
- 哈希函数与数组中元素是可以自定义的
- 若不同元素计算出来的哈希值相同，则会造成存储冲突，有以下冲突处理方式
  - 开放地址：依次往后遍历，看哪个位置没有存储元素

  - 再哈希法：多造几个哈希函数
  - 建立公共溢出区：本质上是一个□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

alt text

- 用unordered_map模拟哈希表
- 用当前遍历到的数据去哈希表内□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

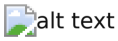
```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map <int,int> h;
        vector<int> ret(2);
        for(int i = 0;i < nums.size();i++)
        {
            if(h.find(target - nums[i]) != h.end())
            {
                ret[0] = h[target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

- 二分□找(在有序数组中□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□找，□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



- 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

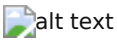
alt text

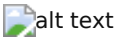
- 个人做法：
- 先对下标数组排序，然后从前往后遍历数组，看是否出现过相同数字

```
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，□找到则返回true，未□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

alt text

- 将某个数组插入哈希表
- 利用set特性可以剔除相同的数字
- □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

alt text

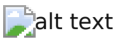
哈希表+滑动窗口

- 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;s[i];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+□分算法

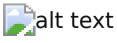
- 二分算法的泛型情况，□找最后一个1 

- 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;s[i]; 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;
    }
};
```

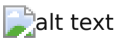
寻找两个正序数组的中位数

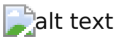


- 将问题化成求两个正序数组中第k大的值
- 使用二分算法的思维，各取两数组中k / 2个元素，通过比较去除k / 2个元素
- 将问题化成求解剩余数组中第k / 2大的值
- 使用递归求解

```
#include <cinttypes>
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



如图，显而易见，□分算法的泛型情况 

- 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□□□□□□
{
    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□□□□□□□□□□

```
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;
}
```