**JavaScript二分搜索**

1. 二分搜索概述

二分搜索（Binary search），也叫折半搜索(half-interval search)，是一种在**有序数组**中把数组**不断折半**去查找某一项的搜索算法。这种算法每一次比较都使搜索范围缩小一半，相对于逐个遍历查找效率会更好。算法复杂度为：O(log N)。

步骤是：

1. 找到数组中间项，如果中间项正好是要查找的元素，则直接返回；
2. 如果要查找的元素大于或小于中间项，则在大于或小于中间项的那一半中继续查找，重复第一步找到中间项；
3. 折半搜索完整个数组(搜索区间长度等于1时)，如果还找不到，则表示不存在该查找项。
4. 二分搜索执行过程分析：

**有序数列：**

2, 3, 8, 10, 13

设查找项：**10**，中间项：**8**

2, 3, 8, 10, 13

折半查找，数组右侧：

中间项：**10**

2, 3, 8, 10, 13

返回下标：**3**

1. 二分搜索代码实现
   1. 非递归，传递左右区间指针。

function binarySearch(arr, target) {

var l = 0, r = arr.length - 1

var midIndex, mid

// l是左侧，r是最右侧。搜索区间长度小于1时停止

while (l <= r) {

midIndex = Math.floor((l + r) / 2)

mid = arr[midIndex]

// 中间项等于目标项则返回下标

if (mid === target) {

return midIndex

} else if (target > mid) {

// 大于中间项折半查找右侧

l = midIndex + 1

} else {

// 小于中间项折半查找左侧

r = midIndex - 1

}

}

return -1

}

var arr = [2, 3, 8, 10, 13]

binarySearch(arr, 10) // 3

* 1. 递归实现，移动左右区间指针。

function binarySearch(arr, target, l, r) {

l = l === undefined ? 0 : l

r = r === undefined ? arr.length - 1 : r

var midIndex = Math.floor((l + r) / 2)

var mid = arr[midIndex]

// 中间值等于查找项说明找到了，则返回中间项下标

if (target === mid) {

return midIndex

}

// 如果查找区间长度大于1需要继续查找

if (l < r) {

if (target > mid) {

// 折半右侧部分开始递归查找

return binarySearch(arr, target, midIndex + 1, r)

} else {

// 折半左侧部分开始递归查找

return binarySearch(arr, target, l, midIndex - 1)

}

}

return -1

}

var arr = [2, 3, 8, 10, 13]

binarySearch(arr, 10) // 3

* 1. 存在重复项，传递左右区间指针。向左查找（向右查找类似）。

function binarySearchLeft(arr, target) {

var l = 0, r = arr.length - 1

var midIndex, mid

// 搜索区间小于等于1停止搜索

while (l <= r) {

// 找到中间值

midIndex = Math.floor((l + r) / 2)

mid = arr[midIndex]

if (target > mid) {

// 大于中间项折半查找右侧

l = midIndex + 1

} else {

// 小于等于中间项则折半查找左侧，同时缩小右侧范围

r = midIndex - 1

}

}

// 此时l是最左侧目标项

if (arr[l] === target) {

return l

}

return -1

}

var arr = [2, 10, 10, 10, 13]

binarySearchLeft(arr, 10) // 1

源码：

● 标准递归版本，一个数组，传递左右区间指针

function binarySearch(arr, target, l, r) {

l = l === undefined ? 0 : l

r = r === undefined ? arr.length - 1 : r

var midIndex = Math.floor((l + r) / 2)

var mid = arr[midIndex]

console.log('start:', arr, target, l, r, midIndex)

// 中间值等于查找项说明找到了，则返回中间项下标

if (target === mid) {

return midIndex

}

// 如果查找区间长度大于1

if (l < r) {

if (target > mid) {

// 折半右侧部分开始递归查找

console.log('right:', arr, target, l, r, midIndex)

return binarySearch(arr, target, midIndex + 1, r)

} else {

// 折半左侧部分开始递归查找

console.log('left:', arr, target, l, r, midIndex)

return binarySearch(arr, target, l, midIndex - 1)

}

}

return -1

}

var arr = [2, 3, 8, 10, 13]

binarySearch(arr, 10)

● 标准非递归版本，一个数组，移动左右区间指针

function binarySearch(arr, target) {

var l = 0, r = arr.length - 1

var midIndex, mid

// 搜索区间小于等于1停止搜索

while (l <= r) {

// 找到中间值

midIndex = Math.floor((l + r) / 2)

mid = arr[midIndex]

console.log('start:', arr, target, l, r, midIndex)

// 中间项等于目标项则返回下标

if (mid === target) {

return midIndex

} else if (target > mid) {

// 大于中间项折半查找右侧

console.log('right:', arr, target, l, r, midIndex)

l = midIndex + 1

} else {

// 小于中间项折半查找左侧

console.log('left:', arr, target, l, r, midIndex)

r = midIndex - 1

}

}

return -1

}

var arr = [2, 3, 8, 10, 13]

binarySearch(arr, 10)

● // 二分搜索递归查找，记录下右侧位置

function binarySearch(arr, target, resultIndex) {

if (arr.length <= 1 && target !== arr[0]) return -1

var midIndex = Math.floor((arr.length - 1) / 2)

var mid = arr[midIndex]

resultIndex = resultIndex === undefined ? 0 : resultIndex

console.log('start:', arr, midIndex, mid, target, resultIndex)

if (target === mid) {

return resultIndex + midIndex

} else if (target > mid) {

// 如果在右侧，则需要将中间index加上

console.log('right:', arr, midIndex, mid, target, resultIndex)

return binarySearch(arr.splice(midIndex + 1), target, resultIndex + midIndex + 1)

} else {

console.log('left:', arr, midIndex, mid, target)

return binarySearch(arr.splice(0, midIndex), target, resultIndex)

}

return -1

}

var arr = [2, 3, 8, 10, 13]

binarySearch(arr, 10)

● 标准非递归版本，自左查找最符合项。存在多个相同项。

function binarySearchLeft(arr, target) {

var l = 0, r = arr.length - 1

var midIndex, mid

// 搜索区间小于等于1停止搜索

while (l <= r) {

// 找到中间值

midIndex = Math.floor((l + r) / 2)

mid = arr[midIndex]

console.log('start:', arr, target, l, r, midIndex)

if (target > mid) {

// 大于中间项折半查找右侧

console.log('right:', arr, target, l, r, midIndex)

l = midIndex + 1

} else {

// 小于中间项折半查找左侧

console.log('left:', arr, target, l, r, midIndex)

r = midIndex - 1

}

}

// 此时l是最左侧目标项

if (arr[l] === target) {

return l

}

return -1

}

var arr = [2, 10, 10, 10, 13]

binarySearchLeft(arr, 10)

● 标准非递归版本，自右查找符合项。存在多个相同项。

function binarySearchRight(arr, target) {

var l = 0, r = arr.length - 1

var midIndex, mid

// 搜索区间小于等于1停止搜索

while (l <= r) {

// 找到中间值

midIndex = Math.floor((l + r) / 2)

mid = arr[midIndex]

console.log('start:', arr, target, l, r, midIndex)

if (target >= mid) {

// 大于中间项折半查找右侧

console.log('right:', arr, target, l, r, midIndex)

l = midIndex + 1

} else {

// 小于中间项折半查找左侧

console.log('left:', arr, target, l, r, midIndex)

r = midIndex - 1

}

}

// 此时l是最左侧目标项

if (arr[r] === target) {

return r

}

return -1

}

var arr = [2, 10, 10, 10, 13]

binarySearchRight(arr, 10)