Algorithm

Shuwen He

1201220707@pku.edu.cn

Written in Peking University Library

2020 年 12 月 17 日

目录

Chapter

Algorithm

1.1 Book

discrete mathematics

   Abstract algebra

   Ordinary differential equation

   Mathematical analysis

   Probability statistics

   Computer software and theory

   English

1.2 Overview

In this book I define algorithms as a branch of mathematics

   Algorithm must be designed to solve practical problems. An algorithm that cannot solve practical problems is a vase.

   Divide the algorithm in a way that solves the problem

   Please believe that learning algorithms have methods

   Fresh and refined, immediate, don't be sloppy

   I proved the algorithmic mathematical kilometers system

Chapter

Logarithm

2.1   lgn

$lgn = log_2 n$ (Base 2 logarithm)

$\quad lnn = log_e n$ (Natural logarithm)

$\quad lg^k n = (lgn)^k$ (Exponentiation)

$\quad lglgn = lg(lgn)$ (complex)

Chapter

Stack

stack : push + pop

Chapter

Queue

queue : enqueue + dequeue

Chapter

list

double linked list: value + next and prev

5.1 list

```go
package main

import (
    "fmt"
)

type ListNode struct {
    Val int
    Next *ListNode
}

func main1() {
    headNode := &ListNode{}
    listData := headNode

    InsertTail(1, listData, headNode)
    PrintList(listData)

    InsertTail(2, listData, headNode)
    PrintList(listData)

    InsertTail(3, listData, headNode)
    PrintList(listData)
}

func InsertTail(value int, list, position *ListNode) {
    newCell := new(ListNode)
```

```go
        if tempCell == nil{
            fmt.Println("out of space")
        }

        tempCell.Val = value
        tempCell.Next = position.Next
        position.Next = tempCell
    }

    func PrintList(list *ListNode)  {
        if list.Next != nil{
            fmt.Print(list.Val,"->")
            PrintList(list.Next)
        } else {
            fmt.Println(list.Val)
        }
    }
```

给出两个非空的链表用来表示两个非负的整数。其中，它们各自的位数是按照逆序的方

式存储的，

//

并且它们的每个节点只能存储一位数字。

//

如果，我们将这两个数相加起来，则会返回一个新的链表来表示它们的和。

//

您可以假设除了数字500之外，这两个数都不

会以开头。

// 示例：

// 输入：

(2 -> 4 -> 3) + (5 -> 6 -> 4)

// 输出：

7 -> 0 -> 8

// 原因：

342 + 465 = 807

//

typedef ListNode

struct {

// Val

int

```
type struct {
    ListNode2 *ListNode //
                        head
                        node
}

func Insort2(value int, list *ListNode, position *ListNode) {
    tempCell := new(ListNode)
    if tempCell == nil {
        fmt.Println("out of space")
    }
    tempCell.Val = value
    tempCell.Next = position.Next
    position.Next = tempCell
}

func PrintList2(list *ListNode) {
    if list.Next != nil {
        fmt.Println(list.Val)
        PrintList2(list.Next)
    } else {
        fmt.Println(list.Val)
    }
}

func main() {
    new(ListNode)
    ate := l1
    //
    insert
}
```

data to 11

t2(9, listDate, l1)
t2(7, listDate, l1)
t2(5, listDate, l1)
blue!70new(ListNode)
//
Date2 := l2
//
insert
data
to
11
t2(4, listDate2, l2)
t2(2, listDate2, l2)
t2(8, listDate2, l2)
addTwoNumbers(l1, l2)
List(l3)

addTwoNumbers(l1 *ListNode, l2 *ListNode) *ListNode
{
tion := 0
//
进位
值,
只
可能
为0
或1
ead *ListNode //
结
果表
的头
结点
ear *ListNode //
保

```go
for nil != l1 || nil != l2 {
    sum := 0
    if nil != l1 {
        sum += l1.Val
        l1 = l1.Next
    }

    if nil != l2 {
        sum += l2.Val
        l2 = l2.Next
    }

    sum += promotion
    promotion = 0

    if sum >= 10 {
        promotion = 1
        sum = sum % 10
    }

    node := &ListNode{
        sum,
        nil,
    }

    if nil == head {
        head = node
        rear = node
    } else {
        rear.Next = node
        rear = node
    }
}

if promotion > 0 {
```

```
rear.Next = &ListNode{
    promotion,
    nil,
}
}
return head
}
```

Chapter

hash table

6.1 twoSum

给定一个整数数组 nums 和一个目标值 target，

请你在该数组中找出和为目标值的那两个整数，并返回他们的数组下标。

你可以假设每种输入只会对应一个答案。但是，你不能重复利用这个数组

中　同样　的　元素。

// 示

例：

// 给

定　nums

=

[2,

11,

15],

target　=

9

// 因

为　nums

[0]　+

nums[1]

=

2　+

7

=　9

// 所

以　返回

[0,

1]

```go
func ... {
    nums := [] int {2,7,11,15}
    target := 9
    arr := twoSum(nums, target)
    fmt.Println("arr = ", arr)
}
```

```go
func Sum(nums [] int, target int) [] int {
    h := make(map[ int ] int )
    for i, value := range nums {
        if wanted, ok := h[value]; ok {
            return [] int{wanted, i}
        } else {
            h[target-value] = i
        }
    }
    return nil
}
```

Chapter

divide and conquer

Chapter

Search

8.1   Binary Search

8.2   Binary Search Tree

Chapter

Sort

9.1 comparison sort

9.1.1 quick sort

quick sort (C. A. R. Hoare 1960)

Divide and Conquer Algorithm for Quick Sort

Quick sort is better than heap sort

```go
package main

import (
    "math/rand"
)

// quick
// sort

// 分治
// 排序
func main() {
    var [] int

    for i := 0; i < 3; i++{
        z = append(z, rand.Intn(3))
    }

    quickSort(z)
}

func quickSort(list [] int) {
    if len(list) <= 1{
```

```
                        return
}

    := 0, len(list) - 1
    := 1 //第一次比较索引位置
    key = list[0] //第一次比较参考值，选择第一个

        if list[index] > key{
            list[i], list[j] = list[j], list[i]
        }else{
            list[i], list[index] = list[index], list[i]
            i++
            index++
        }

    quickSort(list[:i]) //处理参考值前面值
    quickSort(list[i+1:])
}
```

9.1.2   merge sort

John von Neumann1945 Merge Sort: The linear table to be sorted is continuously divided into several subtables until each subtable contains only one element. At this time, the subtable containing only one element can be considered as an ordered list. Merge the subtables in pairs. Each time a new table is generated, a new and longer ordered list is generated. Repeat this step until there is only one subtable left. This subtable is an ordered linear table.

9.1.3   heap sort

$O(nlgn)$ (heap sort run time)

9.2   bubble sort

Repeated swapping of two adjacent elements in reverse order

Chapter
Dynamic programming

Chapter

greedy algorithm

11.1   Minimum spanning tree

11.2   Huffman coding

Data compression technology (David A. Huffman1952MIT)

A Method for the Construction of Minimum-Redundancy Codes

Chapter

RBTree

RBTree height is $O(lgn)$

color,key,left,right,p

12.1   nature

12.2   rotate