

# Algorithm

Shuwen He

1201220707@pku.edu.cn

Written in Peking University Library

2020 年 1 月 9 日

# 目录

<b>第一章</b>	<b>Algorithm</b>	<b>3</b>
1.1	Book . . . . .	3
1.2	Overview . . . . .	3
<b>第二章</b>	<b>Logarithm</b>	<b>4</b>
2.1	lgn . . . . .	4
<b>第三章</b>	<b>Stack</b>	<b>5</b>
<b>第四章</b>	<b>Queue</b>	<b>6</b>
4.1	array . . . . .	6
<b>第五章</b>	<b>list</b>	<b>7</b>
5.1	list . . . . .	7
<b>第六章</b>	<b>hash table</b>	<b>12</b>
<b>第七章</b>	<b>divide and conquer</b>	<b>13</b>
<b>第八章</b>	<b>Search</b>	<b>14</b>
8.1	Binary Search . . . . .	14
8.2	Binary Search Tree . . . . .	14
<b>第九章</b>	<b>Sort</b>	<b>15</b>
9.1	comparison sort . . . . .	15
9.1.1	quick sort . . . . .	15
9.1.2	merge sort . . . . .	16
9.1.3	heap sort . . . . .	16
9.2	bubble sort . . . . .	16
<b>第十章</b>	<b>Dynamic programming</b>	<b>17</b>
10.1	调度问题 . . . . .	17
10.2	矩阵链乘法 . . . . .	17
10.3	公共子序列 . . . . .	17
10.4	Binary search tree . . . . .	17

目录	2
<b>第十一章 greedy algorithm</b>	<b>18</b>
11.1 Minimum spanning tree . . . . .	18
11.2 Huffman coding . . . . .	18
<b>第十二章 RBTREE</b>	<b>19</b>
12.1 nature . . . . .	19
12.2 rotate . . . . .	19

# Chapter

# Algorithm

## Book

discrete mathematics

Abstract algebra

Ordinary differential equation

Mathematical analysis

Probability statistics

Computer software and theory

English

## 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

**lg**

$\lg n = \log_2 n$  (Base 2 logarithm)

$\ln n = \log_e n$  (Natural logarithm)

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

$\lg \lg n = \lg(\lg n)$  (complex)

# Chapter

# Stack

stack : push + pop

# Chapter

# Queue

queue : enqueue + dequeue

**array**

# Chapter

## list

double linked list: value + next and prev

### list

```
1      package main
2
3      import (
4          "fmt"
5      )
6
7      type ListNode struct{
8          Val int
9          Next *ListNode
10     }
11
12     func main1() {
13         headNode := &ListNode{}
14         listData := headNode
15
16         InsertTail(1, listData, headNode)
17         PrintList(listData)
18
19         InsertTail(2, listData, headNode)
20         PrintList(listData)
21
22         InsertTail(3, listData, headNode)
23         PrintList(listData)
24     }
25
26     func InsertTail(value int, list, position *ListNode) {
27         tempCell := new(ListNode)
28
```



```

29         if tempCell == nil{
30             fmt.Println("out of space")
31         }
32
33         tempCell.Val = value
34         tempCell.Next = position.Next
35         position.Next = tempCell
36     }
37
38     func PrintList(list *ListNode) {
39         if list.Next != nil{
40             fmt.Print(list.Val, "->")
41             PrintList(list.Next)
42         } else {
43             fmt.Println(list.Val)
44         }
45     }
46
47     // 给出两个非空的链表用来表示两个非负的整数。其中，它们
48     // 各自的位数是按照逆序的方式存储的，
49     // 并且它们的每个节点只能存储一位数字。
50     // 如果，我们将这两个数相加起来，则会返回一个新的链表来
51     // 表示它们的和。
52     // 您可以假设除了数字0之外，这两个数都不会以0开头。
53     // 示例：
54     // 输入：(2 -> 4 -> 3) + (5 -> 6 -> 4)
55     // 输出：7 -> 0 -> 8
56     // 原因：342 + 465 = 807
57
58     // type ListNode2 struct {
59     //     Val int
60     //     Next *ListNode2
61     // }
62
63     type List struct {
64         headNode2 *ListNode // head node

```

```
65     func Insert2(value int, list *ListNode, position *
        ListNode) {
66         tempCell := new(ListNode)
67         if tempCell == nil {
68             fmt.Println("out of space")
69         }
70         tempCell.Val = value
71         tempCell.Next = position.Next
72         position.Next = tempCell
73     }
74
75     func PrintList2(list *ListNode) {
76         if list.Next != nil {
77             fmt.Println(list.Val)
78             PrintList2(list.Next)
79         } else {
80             fmt.Println(list.Val)
81         }
82     }
83
84     func main() {
85         l1 := new(ListNode)
86         listDate := l1
87         // insert data to l1
88         Insert2(9, listDate, l1)
89         Insert2(7, listDate, l1)
90         Insert2(5, listDate, l1)
91         l2 := new(ListNode)
92         //
93         listDate2 := l2
94         // insert data to l1
95         Insert2(4, listDate2, l2)
96         Insert2(2, listDate2, l2)
97         Insert2(8, listDate2, l2)
98         l3 := addTwoNumbers(l1, l2)
99         PrintList(l3)
100     }
101
```

```
102     func addTwoNumbers(l1 *ListNode, l2 *ListNode) *ListNode
103     {
104         promotion := 0      // 进位值，只可能为0或1
105         var head *ListNode // 结果表的头结点
106         var rear *ListNode // 保存结果表的尾结点
107         for nil != l1 || nil != l2 {
108             sum := 0
109             if nil != l1 {
110                 sum += l1.Val
111                 l1 = l1.Next
112             }
113             if nil != l2 {
114                 sum += l2.Val
115                 l2 = l2.Next
116             }
117             sum += promotion
118             promotion = 0
119
120             if sum >= 10 {
121                 promotion = 1
122                 sum = sum % 10
123             }
124
125             node := &ListNode{
126                 sum,
127                 nil,
128             }
129
130             if nil == head {
131                 head = node
132                 rear = node
133             } else {
134                 rear.Next = node
135                 rear = node
136             }
137         }
138     }
```

```
139         if promotion > 0 {
140             rear.Next = &ListNode{
141                 promotion ,
142                 nil ,
143             }
144         }
145         return head
146     }
```

# Chapter

## hash table

# Chapter

## divide and conquer

# Chapter

# Search

Binary Search

Binary Search Tree

# Chapter

# Sort

## comparison sort

### quick sort

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

Divide and Conquer Algorithm for Quick Sort

Quick sort is better than heap sort

```
1      package main
2
3      import (
4          "math/rand"
5      )
6
7      // quick sort
8      // 分治排序
9      func main() {
10         var z []int
11
12         for i := 0; i < 3; i++{
13             z = append(z, rand.Intn(3))
14         }
15
16         quickSort(z)
17     }
18
19     func quickSort(list []int) {
20         if len(list) <= 1{
21             return
22         }
23
24         i, j := 0, len(list) - 1
```



```

25         index := 1 // 第一次比较索引位置
26         key := list[0] // 第一次比较参考值, 选择第一个
27
28         if list[index] > key{
29             list[i], list[j] = list[j], list[i]
30             j—
31         } else {
32             list[i], list[index] = list[index], list[i]
33             i++
34             index++
35         }
36
37         quickSort(list[:i]) // 处理参考值前面值
38         quickSort(list[i+1:])
39     }

```

### merge sort

John von Neumann 1945 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.

### heap sort

$O(n \lg n)$  (heap sort run time)

### bubble sort

Repeated swapping of two adjacent elements in reverse order

# Chapter

## Dynamic programming

调度问题

矩阵链乘法

公共子序列

Binary search tree

# Chapter

## greedy algorithm

**Minimum spanning tree**

**Huffman coding**

Data compression technology (David A. Huffman1952MIT)

A Method for the Construction of Minimum-Redundancy Codes

# Chapter

# RBTree

RBTree height is  $O(\lg n)$

color, key, left, right, p

**nature**

**rotate**