Data Structures and Algorithms Analysis in C

他趴下是个脆弱的少年，站起来将成为一名矢志不移的坚强斗士

《卡拉马佐夫兄弟》

数据结构和算法的重要地位，怎么提都不为过。当然了，我们除了了解各个数据结构和算法的实现，更为重要的是，需要了解这些数据结构和算法有哪些重要的应用。分别适用于哪些场景。

这轮螺旋提升的主要目的也很明确：复习之前学习过的数据结构和算法，然后分别找到应用的场景。最终能够应用到实践中去，指导我们未来的架构设计。

为了节约时间，我们将进行以下几轮迭代：

1. 把书中有伪代码的，都实现一遍；

这个过程要求比较简单：编译通过，能够简单使用。

1. 书中没有伪代码，但是有实现方案的，都是实现一遍；

这个过程要求就有点高了。

1. 书中没有实现方案，但是提到应用场景的，实现一遍；
2. 书中的有意义的例题实现一遍；

实现例题还是很有意义的。

1. 横向比较各个算法的优劣/性能/占用内存大小；
2. 各个主题

比如红黑树在数据库中的应用/

1. Reference扩展
2. 将典型的算法应用到ACM训练题中去；
3. 参考开源软件中对应的算法；

## Chap1 Introduction

## Chap2 Algorithm Analysis

## Chap3 Lists ,Stacks ,and Queue

这个章节主要介绍了3个基本数据结构：List/Stack/Queue.

详细内容如下：

### 3.1 List

List有很多种形式，包括最简单的Linked List，以及Double Linked List/Doubly Linked Lists/Circularly Linked Lists等实现。

List的应用例子，有两个：

1. The Polynomial ADT
2. Multilists

我们在List中实现的方案是Pointer。在这章的最后，展示了一种替代方案：Cursor Implementation。通过array的方式，替代Pointer的方式来实现List。

这章需要实现的代码有：

|  |
| --- |
| List ADT  List.h  List.c  已经实现 |

|  |
| --- |
| 多项式ADT(Array版)  Polynomial.h  已经实现 |

|  |
| --- |
| 多项式ADT(List版)  Polynomial\_list.h (暂时还未完成) |

|  |
| --- |
| Mutilist(多维列表)  没有源码，暂未实现。 |

|  |
| --- |
| Cursor implementation of Linked Lists  所谓的cursor implement of linked list  就是在没有指针的情况下，用数组来模拟List ADT。  要实现两个目标：   1. 实现类似List ADT中的Node结构：每个Node保存数据和指向next node的“指针”； 2. 实现malloc()和free()；   用什么来代替指针呢？那就是array index。  原来每个Node中next是指针，指向下一个节点；现在Next是**数组索引**。  这个例子的用意也非常明显，拿array和pointer进行对比，可以看到指针的本质就是数组。 |

### 3.2 Stack

Stack包括：

Stack(Implemented in Pointer)的实现

Stack(Implemented in Array)的实现

Stack Applications

Stack应用还是比较多的。

这章需要实现的代码有：

|  |
| --- |
| Stack(List实现) **已经实现**  原理很直接，维护一个List：Stack每次push一个元素，都将这个元素插入到这个List的头部，pop()方法也一样，从List头部这个元素取出来。 |

|  |
| --- |
| Stack(Array实现) **已经实现**  原理很直接，将Stack内容维护在一个数组里。数组最后一个元素是Stack的top元素。由TopOfStack来标志这个Top元素的位置(TopOfStack is array index)。 |

1. Queue

Queue的内容包括Queue的实现/Queue的应用。Queue的操作：

1. Enqueue(从尾部Rear插入数据X)：Size++ Rear++ Q[Rear]=X
2. Dequeue(从头部Front取出数据X)：X=Q[Front] Size— Front++

初始化：

Size= 0；

Rear= 0；

Front= 1；

需要实现的代码有：

|  |
| --- |
| queue.h(array实现) **已经实现** |

## Chap4 Trees

这章主要介绍了树相关的数据结构：Binary Tree/Binary Search Tree/AVL Tree/Splay Tree/B Tree。

### 4.2 Binary Trees

这节需要实现的代码有：

|  |
| --- |
| binary\_tree.h **已经实现** |

实现的代码非常简单，每个node下最多能挂2个child node.

### 4.3 The Search Tree

Binary search tree的规则也非常直接：left child node要小于current node，right child node要大于current node。基于这个规则上会有一些操作。

有了这些规则，目标也非常明确，能够利用这个树的特点，进行排序。因为这类树非常容易实现findMin()/findMax。

|  |
| --- |
| binary\_search\_tree.h**已经实现** |

### 4.4 AVL Trees

AVL树定义如下：AVL Tree作为binary search tree的一种，必须满足以下要求：AVL Tree中的每个节点，其left subtree和right subtree的高度差不能大于1。

为啥要搞AVL Tree？AVL Tree和为了解决树太深的问题。因为如果树太深的话，findMin()/findMax()就比较耗时。

如何实现？实现的方案很直接：旋转。

|  |
| --- |
| avl\_tree.h**已经实现**  因为avl tree 是binary search tree的一种，所以绝大多数方法都和binary search tree一致，不同的方法是：  Height() // 可以直接返回当前节点的Height  Insert() // 非常麻烦的实现  Delete() // 比insert()更加麻烦的实现 |

### 4.5 Splay Trees

暂无实现。

### 4.6 B-Trees

暂无实现。

## Chap5 Priority Queues(Heaps)

这章介绍了一个新的数据结构：Priority queue。从名字也能看出来，这个queue是优先级的，优先级高的node被优先处理。

包括的内容有：Binary Heap/Priority queue的应用/d-heaps/Leftist Heaps/Skew Heaps/Binomial Queues/

### 5.3 Binary Heap

|  |
| --- |
| binary\_heap.h **已经实现** |

Binary heap的规则很直接：child node要比current node大。这样便于实现Heap的目标：总是能很方便地从Priority Queue中取到最小的数据：

DeleteMin(H) Insert(H)

这次，我们用数据来保存数据。那么在数组中怎么表示节点上下级的关系呢：

|  |
| --- |
| First node:  H->Elements[1]  Current node:  H->Elements[i]  Parant node :  H→Elements[ i / 2]  left child node:  H→Elements[ i \* 2]  right child node:  H→Elements[ i \* 2 + 1] |

接下来，我们解读一下代码。最能体现binary heap特点的方法有两个：

|  |
| --- |
| void Insert(ElementType X, PriorityQueue H)  ElementType DeleteMin(PriorityQueue H) |

分别是插入元素和删除元素。

对于插入元素的动作，简单来说，就是 find the proper place for the inserted element , from bottom to the top。

对于deleteMin的操作，简单来说就是first delete the top element(min node),然后补充top这个空缺 ,from top to the bottom。

具体实现请参考代码binary\_heap.h。

**5.4 Applications of Priority Queues**

### 5.5 Leftist Heaps

leftist heaps的定义就不在这里说啦，可以参考书本。主要的意思就是树的左边部分会比较茂盛。

那为啥要用leftist heaps呢？主要是为了解决priority tree的merge问题。leftist tree的merge算法，能够较好地平衡树。

|  |
| --- |
| leftist\_heap.h done 未调试成功 |

### 5.7 Binomial queues

有了之前这么多树，为啥还要搞出一个binomial queue呢？因为我们要不断改进priority queue的性能。Binomial queue有什么优势呢？Insertion特别快。

Binomial queue有一个特色，就是他不是一棵树，而是一个森林+\_+。这个森林中每棵树都是heap-ordered。

典型的操作有：Merge/Insert/DeleteMin。实现细节参考书中的Figure。

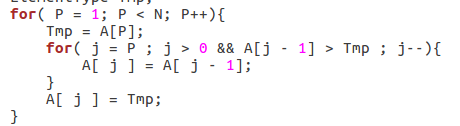
|  |
| --- |
| binomial\_queue.h |

## Chap6 Sorting

这章介绍了Sorting的几个相关算法。Sorting算法包括：Insertion Sort/Shell Sort/Heapsort/Mergsort/Quicksort/Sorting Large Structures/Bucket Sort and Radix Sort/External Sorting

### 6.2 Insertion Sort

Insertion sort的思路非常直接。两个for循环，代码如下：



### 6.4 Shell Sort

Shell sort is a little tricky。3个for循环。

如果要搞清楚shell sort的详细运行机制，最形象的方法，还是把每次运行的结果以Excel的形式记录下来，参考《ShellSort.ods》。

具体实现代码如下：

### 

和Insertion sort相比，Shell sort的效果相当高，给10000个数据排序的结果如下：

shell sort in 0.104999 seconds

insertion sort in 55.406030 seconds

### 6.5 Heap Sort

### Heap Sort的实现方式要参考第五章的binary heap，binary heap通过数组的方式表示了binary tree，关键是the root of binary heap is the min(or max) element。我们可以通过这个特性来检索。

### 检索的步骤如下：

### 1.build a binary heap ,using the input elements。

### 2.delete the root of the binary heap(deleteMax) ,one by one。

### 3.the elements we fetch from step 2 is in sorted order；

### 代码如下：

|  |
| --- |
|  |

### 6.6 Merge Sort

for循环。

### 6.7 Quick Sort

## Chap7 Hashing

这张介绍了Hashing的相关算法，Hashing内容包括

## Chap8 The Disjoint Set ADT

## Chap9 Graph Algorithms

9.1 Definitions

介绍图的基本概念

9.2 Topological Sort

9.3 Shortes-Path Algorithms

求解最短路径的各种算法介绍。图中的线段是有权重的，计算权重最低的最短路径。

9.4 Network Flow Problems

模拟计算机网络的场景，算出网络的最大流量。

9.5 Minimum Spanning Tree

9.6 Application of Depth-First Search

深度优先检索

9.7 Introducton to NP-Completeness

## Chap10 Algorithm Design Techniques

10.1 Greedy Algorithms

10.2 Divide and Conquer

10.3 Dynamic Programming

10.4 Randomized Algorithms

10.5 Backtracking Algorithms

## Chap11 Amotized Analysis

11.1 An Unrelated Puzzle

11.2 Binomial Queues

11.3 Skew Heaps

11.4 Fibonacci Heaps

11.5 Splay Trees

## Chap12 Advanced Data Structures and Implementation

12.1 Top-Down Splay Trees

12.2 Red Black Trees

12.3 Deterministic Skip Lists

12.4 AA-Trees

12.5 Treaps

12.6 k-d Trees

12.7 Pairing Heaps