项目文档

谈瑞

项目十：几种排序算法比较

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 专业班级：软件工程4班  电话：18936361545 | 学号：1452775  电子邮件：tanrui106@163.com | 课目：数据结构课程设计  个人网站：http://guitoubing.top |
|  |  |  |

目录

[项目简介 3](#_Toc496957709)

[项目概要 3](#_Toc496957710)

[项目功能及要求 3](#_Toc496957711)

[项目结构 4](#_Toc496957712)

[项目类的实现 5](#_Toc496957713)

[Maze类 5](#_Toc496957714)

[Node类 6](#_Toc496957715)

[主要代码分析 7](#_Toc496957716)

[Maze.cpp 7](#_Toc496957717)

[Maze类默认构造函数 7](#_Toc496957718)

[Maze类默认析构函数 9](#_Toc496957719)

[findRoad函数 10](#_Toc496957720)

[运行测试 12](#_Toc496957721)

[几点不足 15](#_Toc496957722)

[代码不够精简 15](#_Toc496957723)

# 项目简介

## 项目概要

随机函数产生10000个随机数，用快速排序，直接插入排序，冒泡排序，选择排序，希尔排序，堆排序，归并排序，基数排序的排序方法排序，并统计每种排序所花费的排序时间和交换次数。

## 项目功能及要求

实现对八种排序算法的花费时间、查找次数的比较，并分析出各种排序算法的优劣

# 项目结构

本项目对八种排序方法进行了实现，输出了对10000个随机数进行排列时的花费时间及交换次数，同时对每种算法的实现方法进行了思考，分析了各类算法的优劣，而本项目在真正实现时也并非是对原标准算法照搬照抄，而是先理解其原理而后以自己的想法组织代码，因此在性能、代码形式等方面可能会与标准算法有或多或少的差异。

下面用表格的形式对八种算法进行一下比较：

## 性能分析

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 时间复杂度 | 花费时间 | 查找次数 | 交换次数 | 所需额外空间 |
| 冒泡排序 |  |  |  |  |  |
| 选择排序 |  |  |  |  |  |
| 直接插入排序 |  |  |  |  |  |
| 希尔排序 |  |  |  |  |  |
| 快速排序 |  |  |  |  |  |
| 堆排序 |  |  |  |  |  |
| 归并排序 |  |  |  |  |  |
| 基数排序 |  |  |  |  |  |

## 特点比较

|  |  |  |  |
| --- | --- | --- | --- |
|  | 核心思想 | 优点 | 缺点 |
| 冒泡排序 | 冒泡排序可以说是想法最简单的排序了，它重复的遍历序列，每次遍历将对每两个元素进行比较，较大的将后移，一次遍历下来，最大的元素浮到序列顶端，时间复杂度为O(n2) | 稳定的排序算法 | 慢 |
| 选择排序 | 选择排序基本思想是将序号从i到n的元素序列中的具有最小排序码的元素上调，调至子序列首位，重复操作直到i=n-1，排序即结束，时间复杂度为O(n2) | 移动的次数是固定的，对于n个元素就只需要n-1次移动 | 不稳定的排序算法  ，需要很多次比较 |
| 直接插入排序 | 插入排序，个人认为与选择排序的算法思想相似，均是每次将子序列长度加1后，进行调整。时间复杂度也为O(n2)，基本思想是第i次插入元素时，后面i-1个元素已经是排好序的了（这里采用了倒序插入，即从大到小），此时将其插入到其应该在的位置，如此反复n次序列就已经排好序了 | 稳定的排序算法，速度较快 | 需要进行大量的数据移动工作，这对于数组来说不友好，但是可以用链表解决这一问题 |
| 希尔排序 |  |  |  |
| 快速排序 | 快速排序的基本思想是对序列进行分层，以序列第一个元素为基准，将排序码大的元素后移，排序码小的前移，此时该基准元素的位置已经是固定下来了，再对左右两个子序列递归前面的操作，最终即可得到排好序的序列了 |  |  |
| 堆排序 |  |  |  |
| 归并排序 |  |  |  |
| 基数排序 |  |  |  |

# 项目类的实现

## BSortTree类

采用二叉搜索树构建主体结构

private成员

public成员

|  |  |
| --- | --- |
| 类成员 | 作用 |
| Josepth (); | Josepth类的默认构造函数，此处用户输入约瑟夫环的限制条件 |
| ~Josepth (); | Josepth类的默认析构函数，在程序结束后释放链表中所有在游戏中存活下的结点 |
| void setNumbers(int, const int); | 设置限制条件，同时判断输入是否合理，若不合理则重新输入 |
| Passenger\* setJosepthNode(int); | 递归设置约瑟夫环结点，最后构建约瑟夫环 |
| Passenger\* setBeginNode(int); | 根据开始位置设置开始结点 |
| Passenger\* deleteNode(Passenger\*); | 删除结点，即杀人 |
| void startKilling(Passenger\*); | 游戏开始入口 |
| Passenger \*present, \*head, \*tail; | 头尾结点、当前结点 |
| int amount, firstIndex,  deathNumber, amountLeft; | 约瑟夫环的限制条件 |
|  |  |

# Passenger类

public成员

Passenger类存储了迷宫的结点

private成员

|  |  |
| --- | --- |
| 类成员 | 作用 |
| Passenger(); | Passenger类的默认构造函数 |
| Passenger(int, Passenger\*); | Passenger类的重载构造函数，通过参数number及next指针构造实例 |
| ~Passenger(); | Passenger类的默认析构函数 |
| int number; | 旅客的位置编号 |
| Passenger\* next; | 指向下一个旅客的指针 |
|  |  |

# 代码分析