# 一 .排序算法

## 1.1 四种排序算法

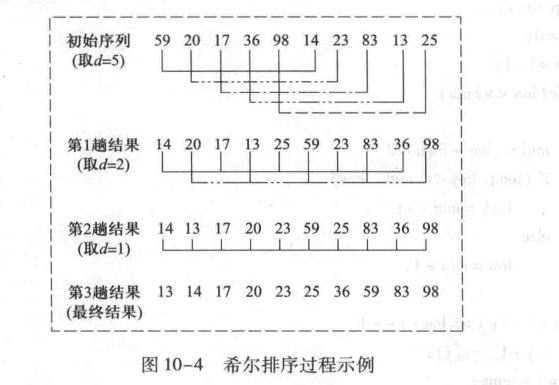
### 1.1.1 希尔排序

**·基本思想**

希尔排序(Shell Sort)是插入排序的一种。也称缩小增量排序，是直接插入排序算法的一种更高效的改进版本。希尔排序是非稳定排序算法。该方法因DL．Shell于1959年提出而得名。希尔排序是记录按下标的一定增量分组，对每组使用直接插入排序算法排序；随着增量逐渐减少，每组包含的关键词越来越多，当增量减至1时，整个文件恰被分成一组，算法便终止。

我们分割待排序记录的目的是减少待排序记录的个数，并使整个序列向基本有序发展。而如上面这样分完组后，就各自排序的方法达不到我们的要求。因此，我们需要采取跳跃分割的策略：将相距某个“增量”的记录组成一个子序列，这样才能保证在子序列内分别进行直接插入排序后得到的结果是基本有序而不是局部有序。

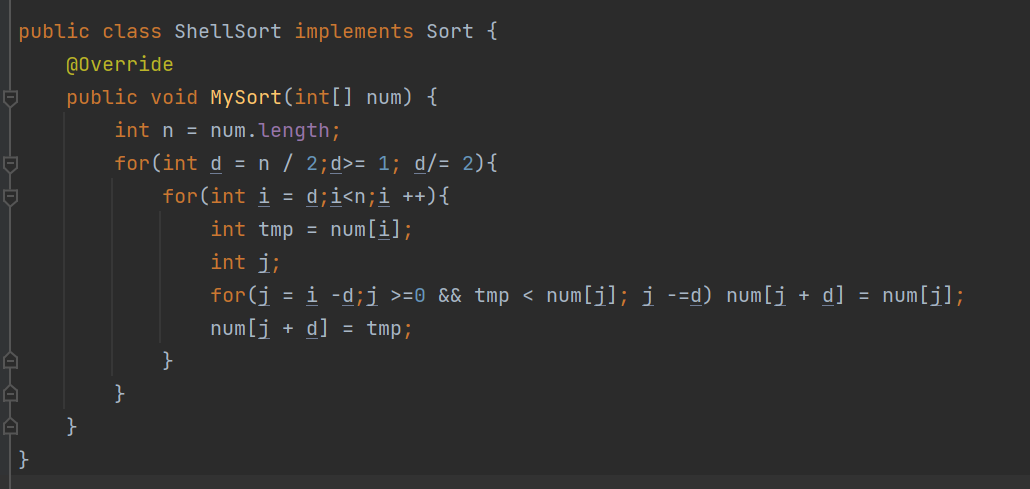
**·操作方法**

1.首先确定增量序列，即将序列分割成若干个子序列的间隔。常用的增量序列为：希尔增量序列，即d[n] = n/2, d[n-1] = d[n]/2, ..., d[1] = 1。

2.对于每个增量，对子序列进行插入排序。假设当前的增量为d[i]，则将序列分割成d[i]个子序列，分别对这些子序列进行插入排序。

3.逐步缩小增量，重复第二步操作，直到增量为1时，对整个序列进行插入排序，排序完成

·**代码实现和运行结果**

****

### 1.1.2 快速排序

**·基本思想**

快速排序的基本思想是：通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据比另一部分的所有数据要小，再按这种方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，使整个数据变成有序序列。

其基本原理如下：

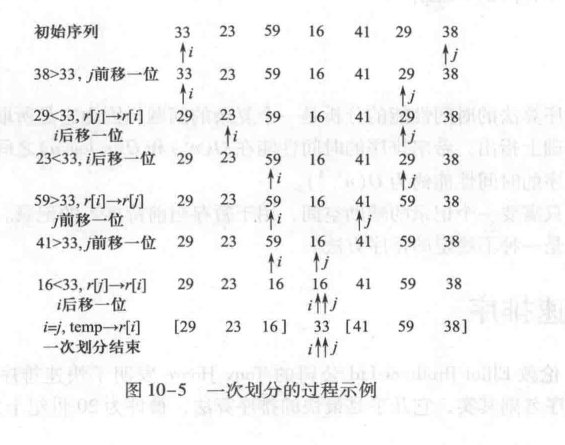
1.选取一个基准元素（pivot），可以选择数组的第一个元素、最后一个元素、中间元素等。

2.将数组中小于等于基准元素的数放在其左边，大于等于基准元素的数放在其右边，可以使用双指针法实现。

3.对左右两个子序列重复以上步骤，直到子序列长度为1或0。

4.整个数组排序完成。

**·操作方法**

快速排序的划分函数的实现如下：

选取数组的第一个元素r[i]为基准元素temp。

设置两个指针i和j，分别指向数组的第一个和最后一个元素。

循环进行以下操作：

1.从右往左找到第一个小于基准元素temp的元素r[j]。

2.将r[j]移到左边，即r[i]的位置，并将i加1。

3.从左往右找到第一个大于基准元素temp的元素r[i]。

4.将r[i]移到右边，即r[j]的位置，并将j减1。

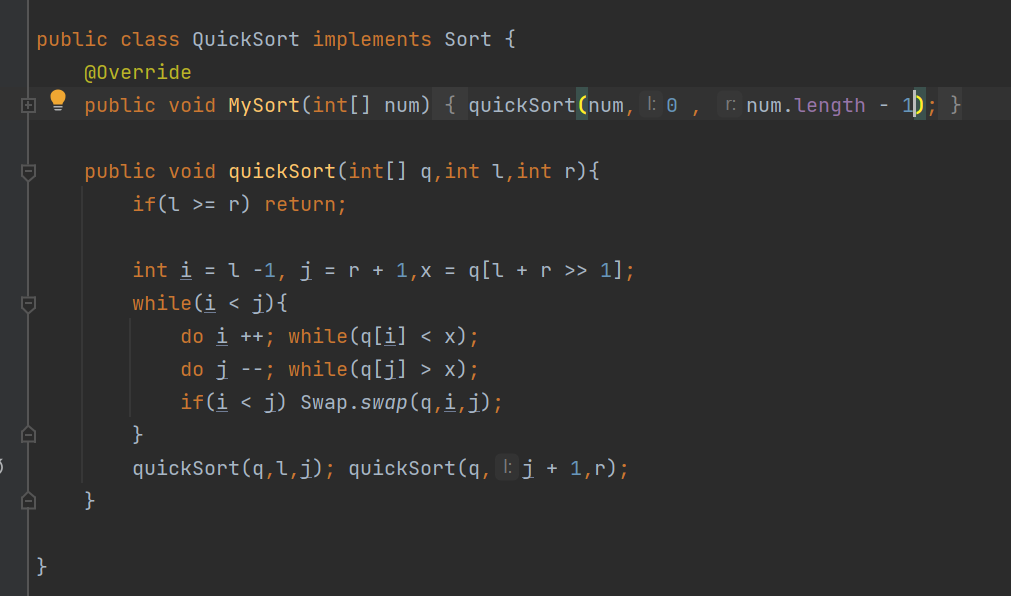
5.重复上述操作，直到i>=j。

将基准元素temp放在最终的位置i处，并返回i。

该函数的时间复杂度为O(n)，其中n为子序列的长度。

使用划分函数执行原理中的1,2操作，即可得到快速排序的结果。该快速排序的时间复杂度为O(nlogn)，空间复杂度为O(logn)，其中n为数组的长度。由于使用了递归，因此需要考虑栈空间的使用。

·**代码实现**

****

**·改进**

由于上述代码所选取的基准元素为第一个数，所以在数组有序时快速排序会退化成冒泡排序。为了避免这个问题，我们可以选取中间值为基准元素，即temp=r[i+j>>1].

### 1.1.3 堆排序

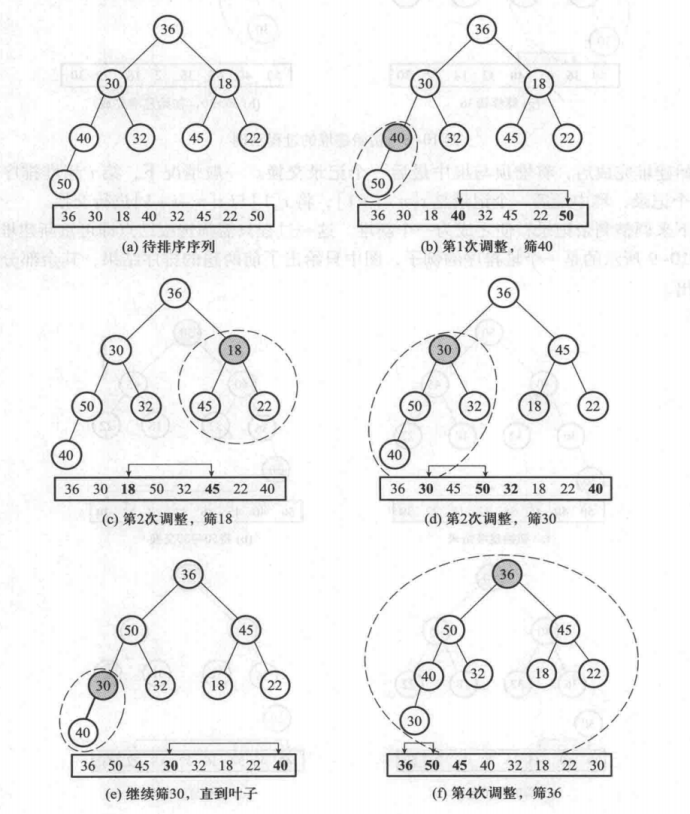
**·基本思想**

首先将待排序的元素构建成一个大根堆（或小根堆），然后将堆顶元素（最大元素或最小元素）和堆底元素交换位置，再对剩余的元素重新构建堆。如此往复执行，直到排序完成。

堆排序的具体实现可以分为两个步骤：

1.建立堆：将待排序的元素构建成一个堆。具体实现可以采用从后往前遍历数组，对每个元素进行下沉操作，从而将数组变成一个符合堆的定义的完全二叉树。

2.排序：依次取出堆顶元素，与堆底元素交换位置，并重新构建堆。如此往复执行，直到堆中所有元素都被取出，即可完成排序。

**·操作方法**

堆排序的建堆操作的实现如下：

1.接收一个记录数组r和两个索引k和m，其中k是二叉堆的父节点的索引，m是堆的最大索引。

2.while循环在左孩子节点（2k）小于或等于堆的最大索引（m）时进行迭代。

3.在while循环内部，代码首先检查父节点的右孩子节点（2k + 1）是否小于堆的最大索引（m），如果是，则比较左右孩子节点的值以确定哪个孩子节点的值更高。将更高值的孩子节点的索引存储在变量j中。

4.然后，代码检查父节点r[i]的值是否大于子节点r[j]的值。如果是，则循环被中断，因为维护了堆属性。如果不是，则代码交换父节点和子节点的值，更新索引i为子节点j的索引，并将索引j设置为新父节点的左孩子节点的索引（2i）。

这个过程重复进行，直到堆属性被恢复并且父节点的值大于或等于其两个子节点的值。

在此基础上，就可以得到堆排序的操作方法：

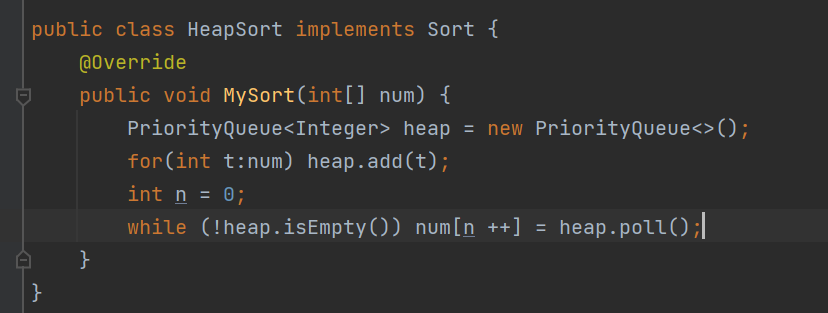
1.调用Sift函数建立一个大根堆，从n/2的位置开始向前调整，直到整个数组都符合堆的定义。

2.从数组的最后一个元素开始，依次将堆顶元素（即最大元素）与当前元素交换位置，并对剩余元素重新调整堆。这个过程可以通过Sift函数来实现，每次调整的范围是从堆顶到当前位置的前一个位置。

3.重复步骤2，直到整个数组都排好序。

总的时间复杂度为O(nlogn)，其中建立堆的时间复杂度为O(n)，每次调整堆的时间复杂度为O(logn)，需要调整n-1次，因此总的时间复杂度为O(nlogn)。由于只需要一个额外的变量来辅助交换元素，所以空间复杂度为O(1)。

·**代码实现**



### 1.1.4 归并排序

**·基本思想**

归并排序是一种基于分治思想的排序算法，它将待排序的序列分为若干个子序列，每个子序列都是有序的，然后再将子序列合并成一个有序的序列。

归并排序可以使用递归来写，也可以不使用递归来写，递归版的归并排序的基本思想可以概括为以下几个步骤：

1.将待排序序列分成两个子序列，分别对每个子序列递归地进行归并排序，直到子序列长度为1。

2.将两个有序子序列合并成一个有序序列。具体实现可以通过比较两个子序列的首元素，将较小的元素插入到新的序列中，然后将已经插入的元素从原子序列中删除，并重复这个过程，直到所有元素都插入到新的序列中。

重复步骤2，直到所有子序列都被合并成一个有序序列。

**·操作方法**

归并排序合并函数作用：将分割得到的子序列两两合并，得到更大的有序子序列，直到最后只剩下一个有序序列为止。具体实现方法如下：

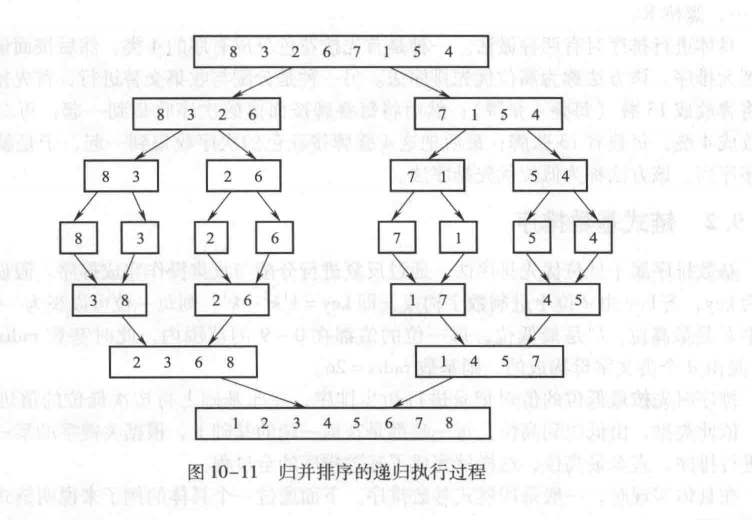
参数r代表原始序列，r1代表辅助序列，s、m、t分别表示需要合并的两个子序列的起始、中间和终止位置。

在函数中，使用三个变量i、j和k来分别表示两个子序列和辅助序列的当前位置。从i和j开始，依次将较小的元素放入r1数组中，直到其中一个子序列的元素全部被放入r1数组为止。然后将剩余的元素依次放入r1数组中，最终r1数组中就是两个子序列合并后的有序序列。

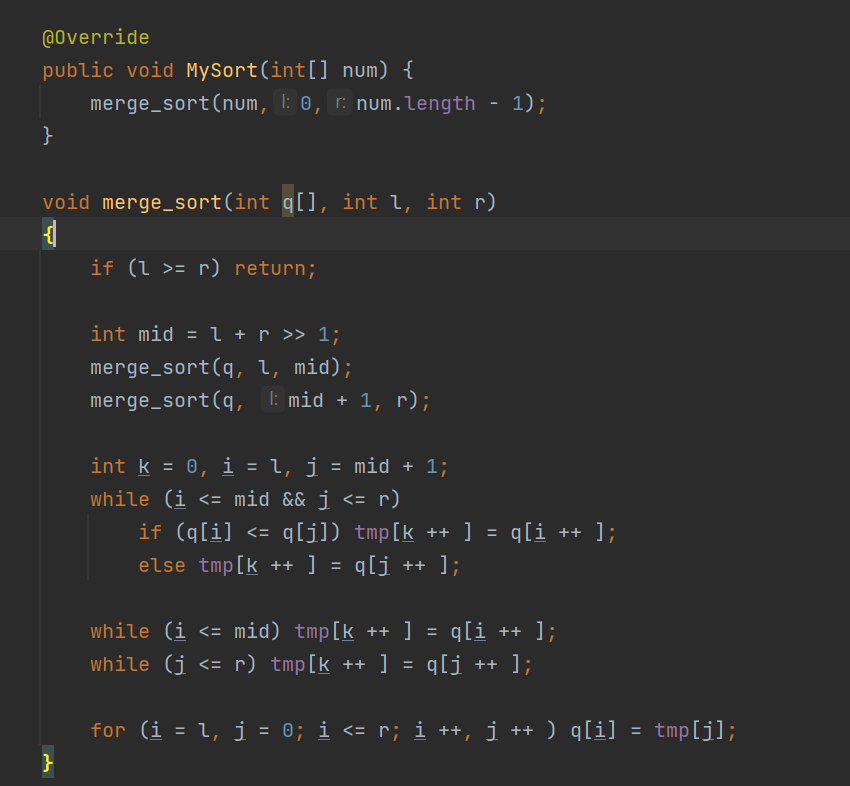
最后，将r1数组中的元素复制回原始序列r的对应位置。

根据基本思想，在合并有序序列时调用Merge函数，将结果存入r1，然后再把r1复制回给r,递归进行以上操作，排序就完成了。

归并排序是稳定的排序算法，具有良好的适应性，可以用于对链式结构和顺序存储结构的数据进行排序。其时间复杂度为O(nlogn)，空间复杂度为O(n)。



·**代码实现和运行结果**

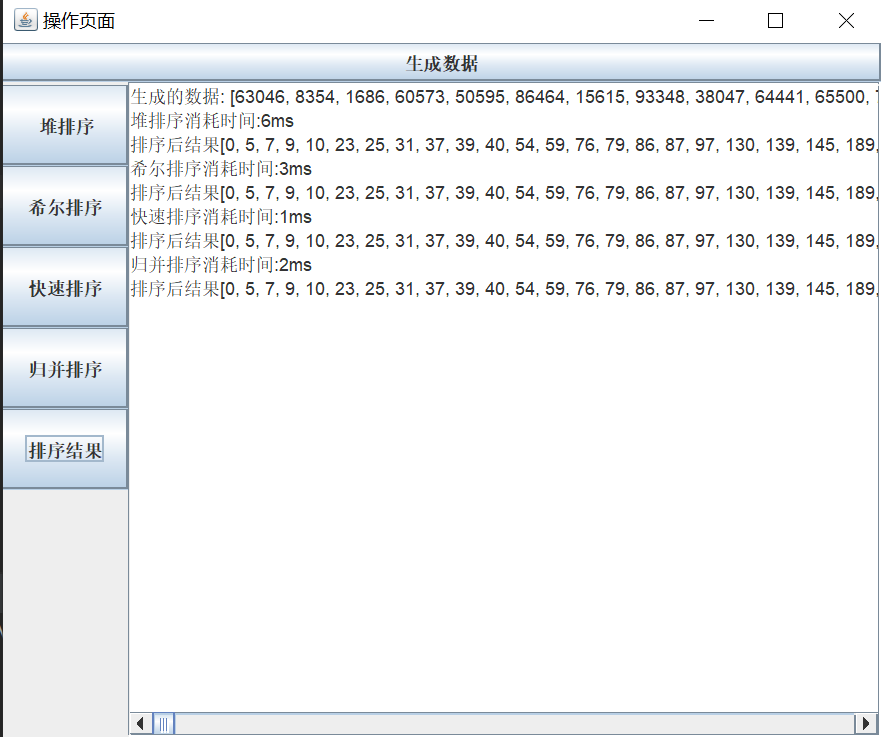


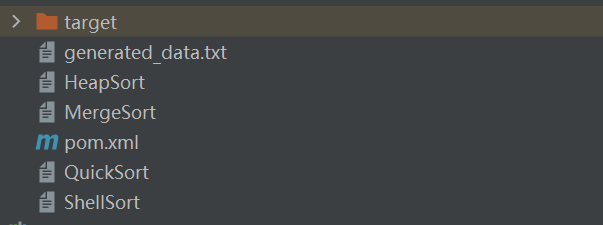
## 1.2 四种排序算法比较

### 1.2.1 时间复杂度分析

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 排序方法 | 时间复杂度（平均） | 时间复杂度  （最好） | 时间复杂度  （最坏） | 空间复杂度 | 稳定性 |
| 希尔排序 |  |  |  |  | 不稳定 |
| 快速排序 |  |  |  |  | 不稳定 |
| 堆排序 |  |  |  |  | 不稳定 |
| 归并排序 |  |  |  |  | 稳定 |

## 1.3 最终页面效果演示



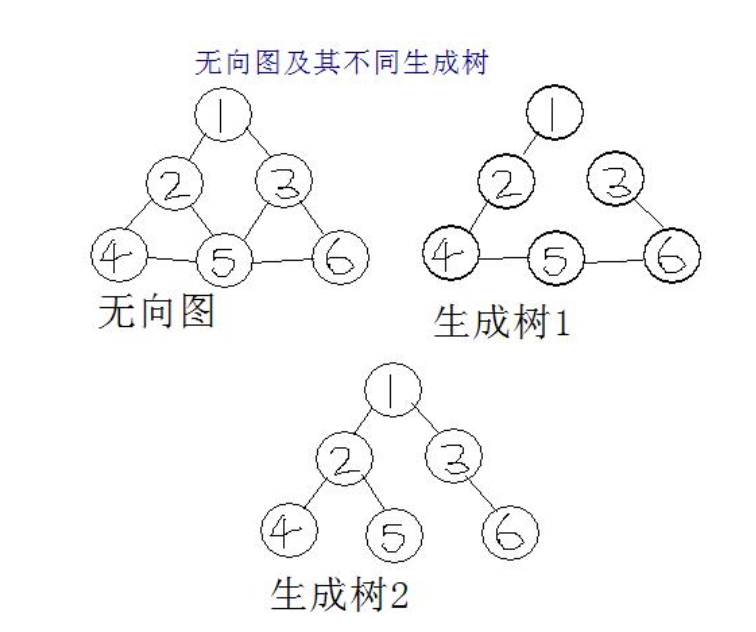


# 二．管道建设最佳方案

## 2.1 最小生成树算法

### 2.1.1 最小生成树概念

在一给定的无向图G = (V, E) 中，(u, v) 代表连接顶点 u 与顶点 v 的边，而 w(u, v) 代表此的边权重，若存在 T 为 E 的子集（即）且为无循环图，使得的 w(T) 最小，则此 T 为 G 的最小生成树。最小生成树其实是最小权重生成树的简称。（简而言之就是把一个图变成一棵树，并且树中的边权和最小）



### 2.1.2 Prim算法

Prim算法是一种用于在加权连通图中找到最小生成树的贪心算法。最小生成树是图的一棵生成树，它的所有边的权值之和最小。Prim算法以一个初始节点开始，逐步将与当前生成树相连的权值最小的边添加到生成树中，直到所有的节点都被连接。

以下是Prim算法的详细步骤

1. 选择初始节点： 从图中选择一个初始节点作为生成树的根节点。
2. 初始化： 将该节点标记为已访问，并将与该节点相连的所有边加入一个优先队列（最小堆）中。
3. 重复以下步骤，直到所有节点都被访问：

从优先队列中选择最小权值边： 从优先队列中取出权值最小的边，该边连接了生成树的一个节点和一个未访问的节点。

将未访问的节点标记为已访问： 将未访问的节点标记为已访问，并将与该节点相连的所有边加入优先队列。

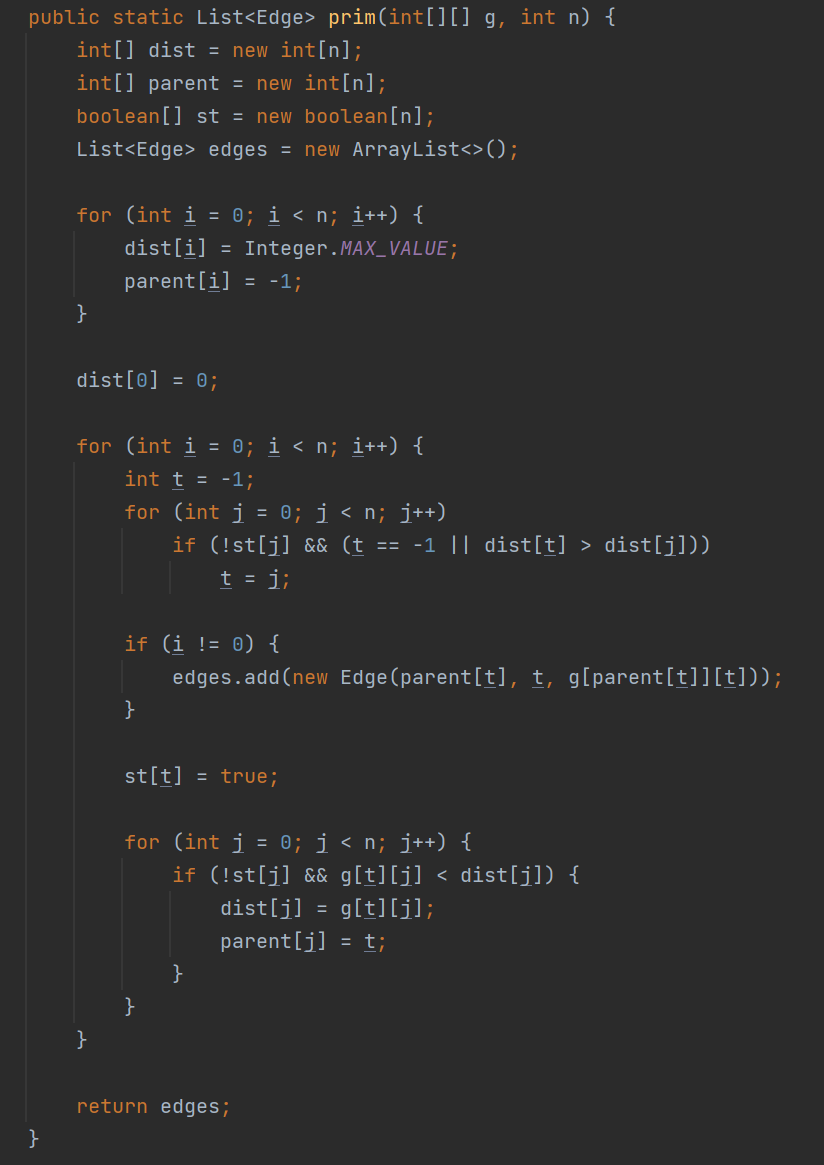
将选中的边加入生成树： 将选中的边加入生成树的边集合。

生成最小生成树： 重复步骤3直到所有节点都被访问，此时生成树的边集合即为最小生成树。

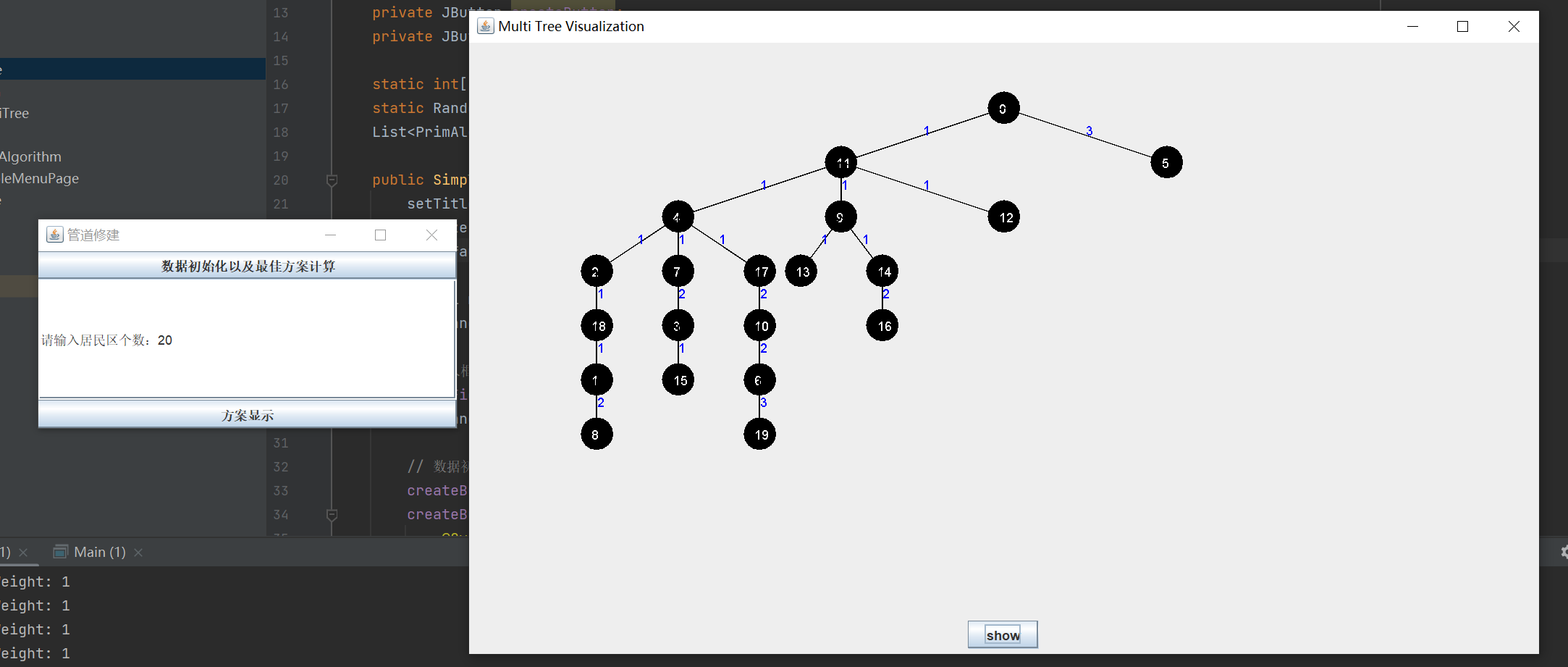
Prim算法的核心思想是每次都选择连接生成树和未访问节点的权值最小的边。由于Prim算法每次只关注当前生成树的边，因此在边的数量比较大的图中，Prim算法的效率较高。同时，Prim算法可以用于处理稠密图和稀疏图。

总体来说，Prim算法的流程是很直观的，但要注意实现时的数据结构选择，通常需要使用堆（优先队列）来高效地找到当前最小的边。

### 2.1.3 代码实现



## 2.2 页面实现以及效果演示



# 三．相关文件操作以及页面设计

## 3.1 文件操作

PrintWriter 和 Scanner 是 Java 中用于文件操作的两个常用类。它们分别用于输出（写入）和输入（读取）文本文件。下面详细解释一下这两个类的用法：

### 3.1.1 PrintWriter

PrintWriter 是用于向文件写入文本数据的类。它提供了多种写入数据的方法，使得文件写入操作变得简便。

PrintWriter 写入方法：

print 和 println：用于写入字符串，println 会在写入字符串后自动换行。

printf：类似于 System.out.printf，格式化写入数据。

### 3.1.2 Scanner

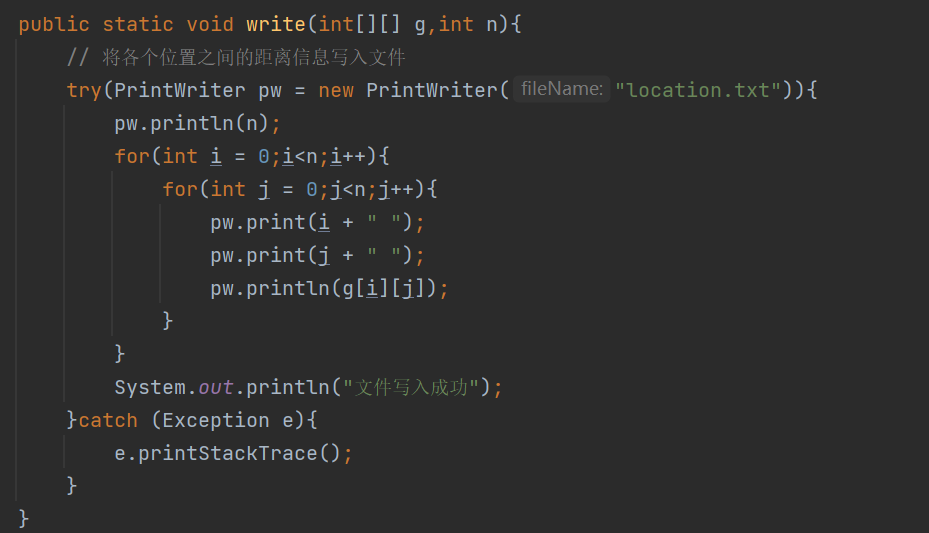
Scanner 类用于从输入源（例如文件、输入流、字符串等）读取数据。它提供了各种 next 方法，可用于读取不同类型的数据。

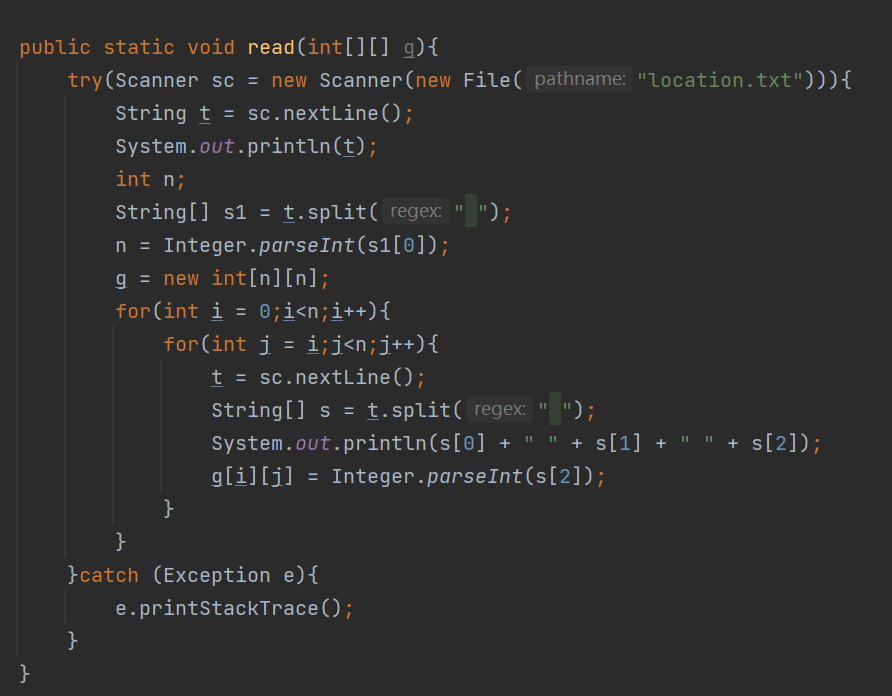
Scanner 读取方法：

next 和 nextLine：用于读取字符串。

nextInt、nextDouble 等：用于读取特定类型的数据。

hasNext、hasNextLine 等：用于检查是否还有下一个输入项





## 页面设计

### AWT 库

AWT 是 Java 最早的 GUI 库，提供了一组与平台无关的 API，用于创建窗口、按钮、文本框等基本 GUI 组件。AWT 的组件是直接使用本地平台的 GUI 控件，因此在不同平台上可能有不同的外观。

AWT 主要有以下几个核心概念：

组件（Component）： 表示用户界面的元素，如按钮、文本框等。

容器（Container）： 用于包含和组织组件的容器，如窗口、面板等。

布局管理器（Layout Manager）： 用于管理容器中组件的排列方式，例如 BorderLayout、FlowLayout、GridLayout 等。

### 3.2.2 Swing 库

Swing 是在 AWT 的基础上发展起来的一套 GUI 工具包，提供了丰富的组件、更好的外观和跨平台一致性。Swing 的组件是纯 Java 实现的，而不是直接使用本地平台的控件，因此在不同平台上具有相同的外观。

Swing 的主要特点包括：

轻量级组件（Lightweight Components）： Swing 组件是纯 Java 实现，不依赖于本地平台的 GUI 控件，因此更加灵活和跨平台。

面向对象设计： Swing 使用面向对象的设计，提供了丰富的类和接口，使得 GUI 的构建更加模块化和可扩展。

Pluggable Look and Feel（PLAF）： 允许开发者选择应用程序的外观，例如 Metal、Nimbus、Windows 等。

事件驱动（Event-Driven）： Swing 使用事件模型来处理用户交互，开发者可以注册事件监听器来响应用户的操作。

图形渲染（Graphics Rendering）： Swing 提供了自定义绘制的能力，可以通过继承组件并重写 paintComponent 方法来自定义组件的外观。