**数据结构与算法大作业**

**22062322 数学 许耀坤**

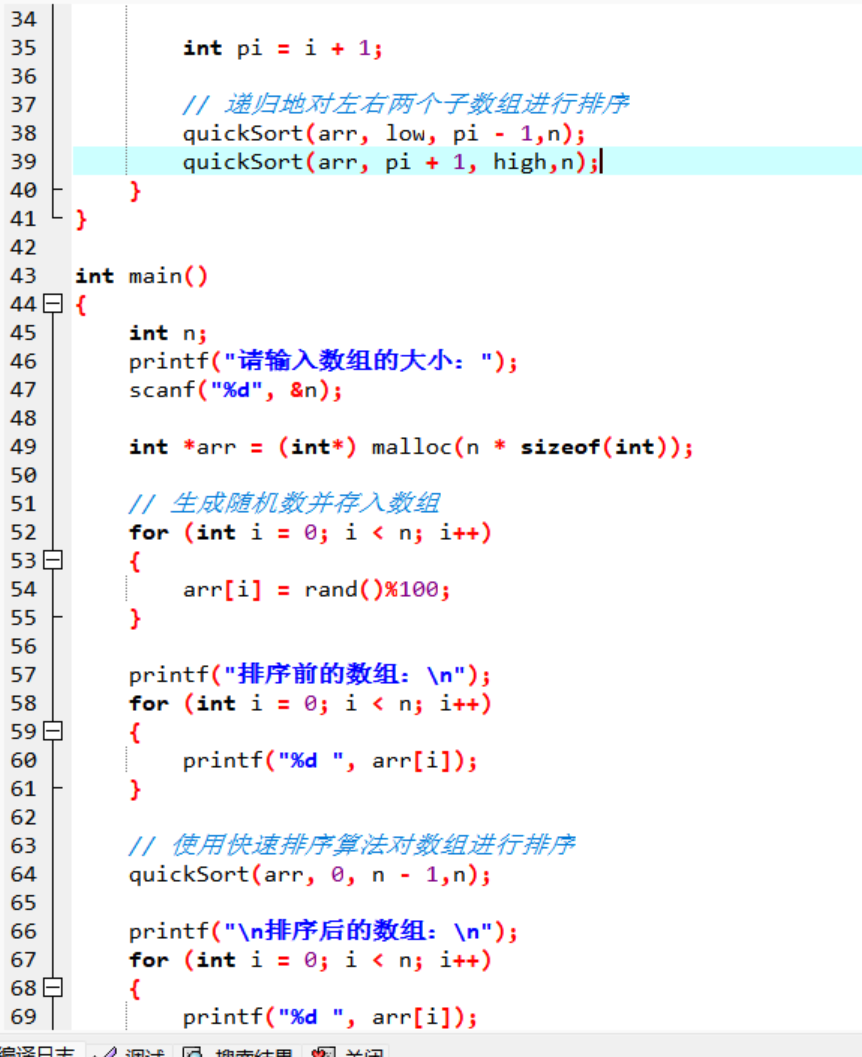
**第一部分 五种排序算法**

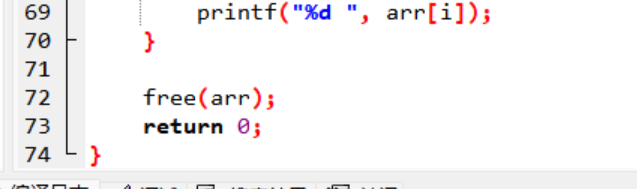
一、快速排序

快速排序是一种常用的排序算法，它基于分治的思想，通过将数组分成较小的子数组来进行排序。其基本思想是选择一个基准元素，将数组分成两个子数组，比基准元素小的元素放在左边，比基准元素大的元素放在右边，然后分别对左右两个子数组进行递归排序，最后将左子数组、基准元素和右子数组合并起来。

下面给出具体代码：







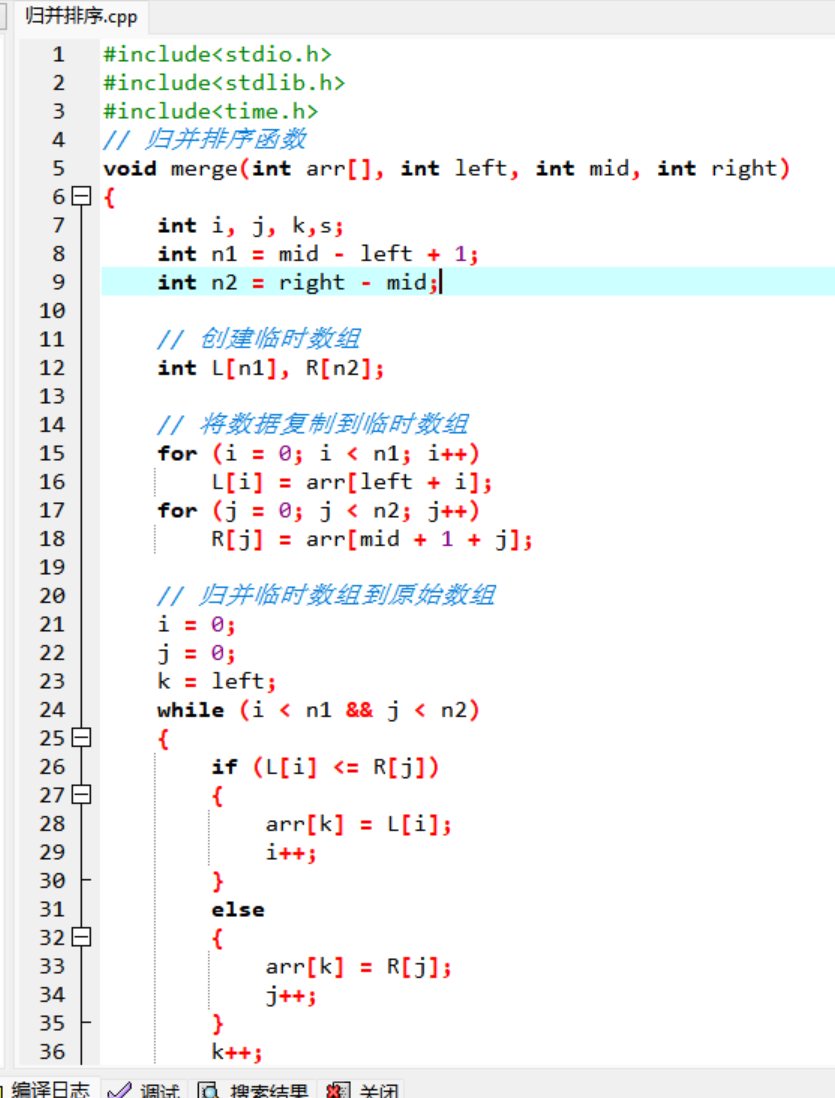
根据上面给出的代码，我们选取n=20时运行，此时我们可以得出相应的运行结果及具体迭代过程：



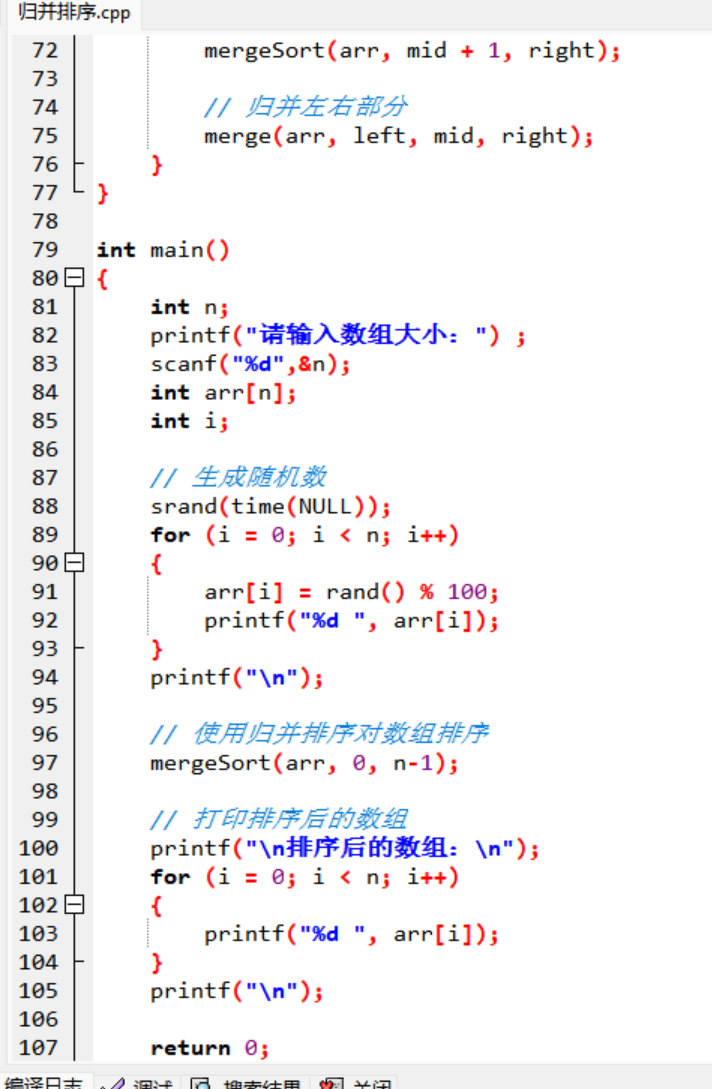
1. 并归排序

归并排序是一种分治算法，它将待排序的数组分成两个子数组，分别对这两个子数组进行排序，然后将排序后的子数组合并成一个有序的数组。其基本思想是先递归地将数组不断二分，直到每个子数组只有一个元素，然后再将这些子数组两两合并，直到最终得到一个有序的数组。

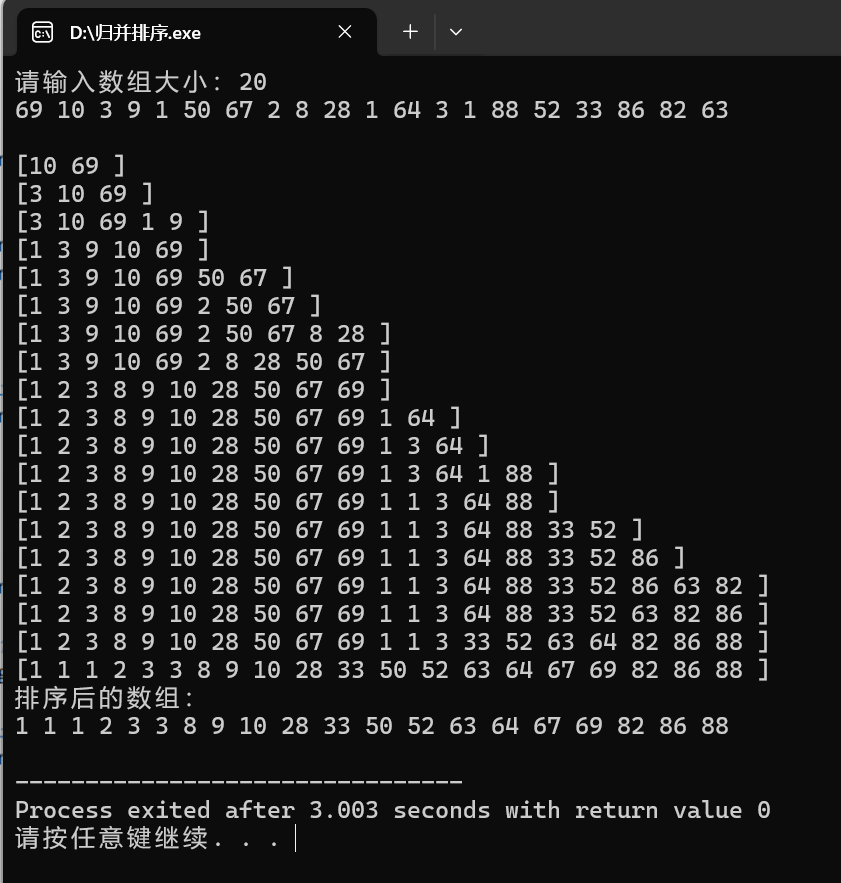
下面给出具体代码：







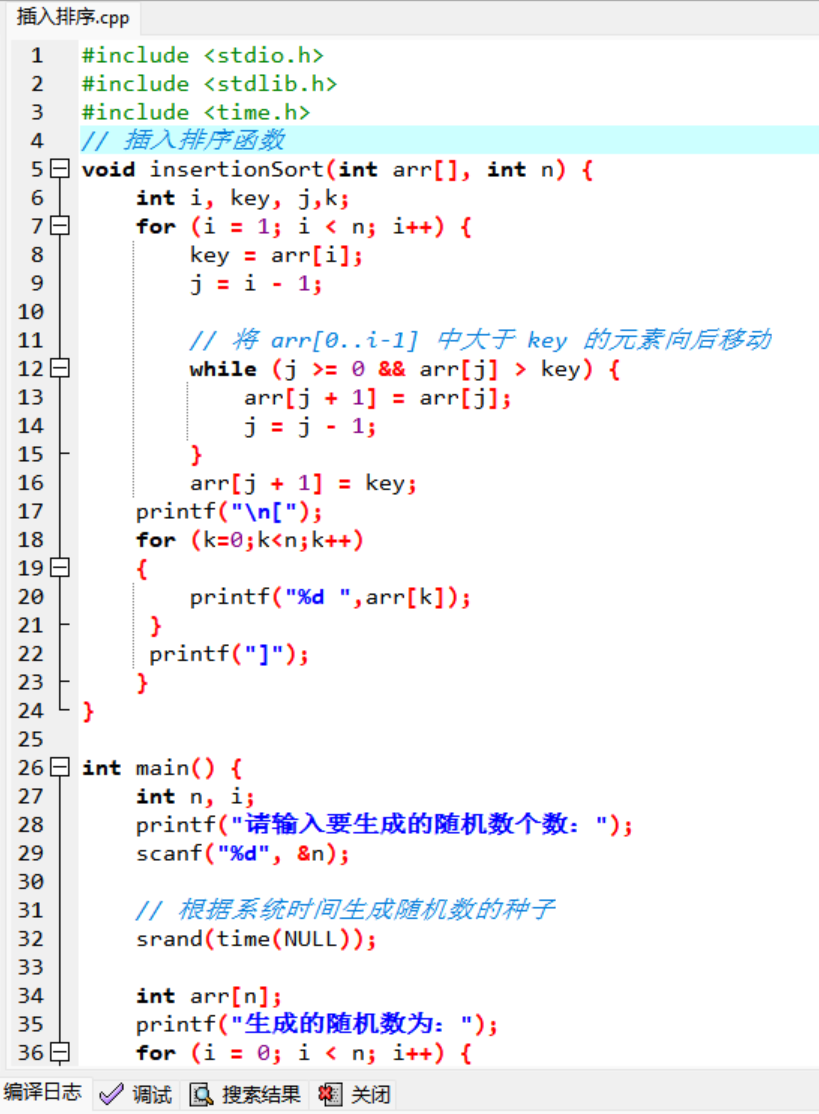
根据上面给出的代码，我们选取n=20时运行，此时我们可以得出相应的运行结果及具体迭代过程：

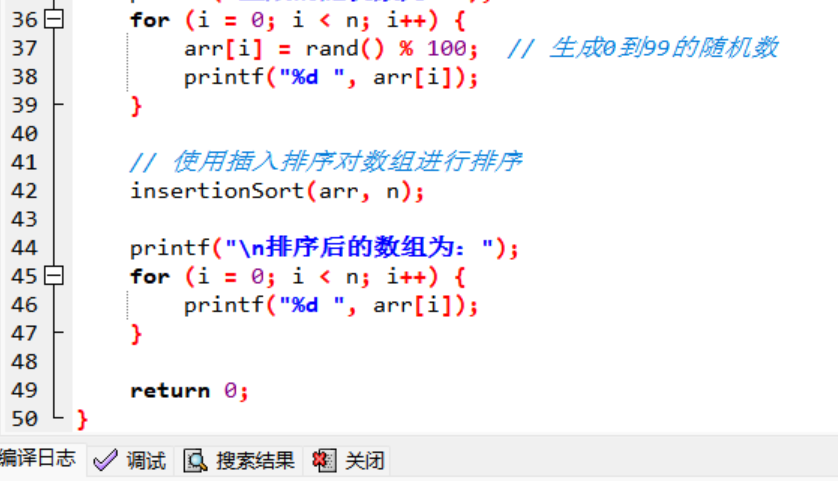


1. 插入排序

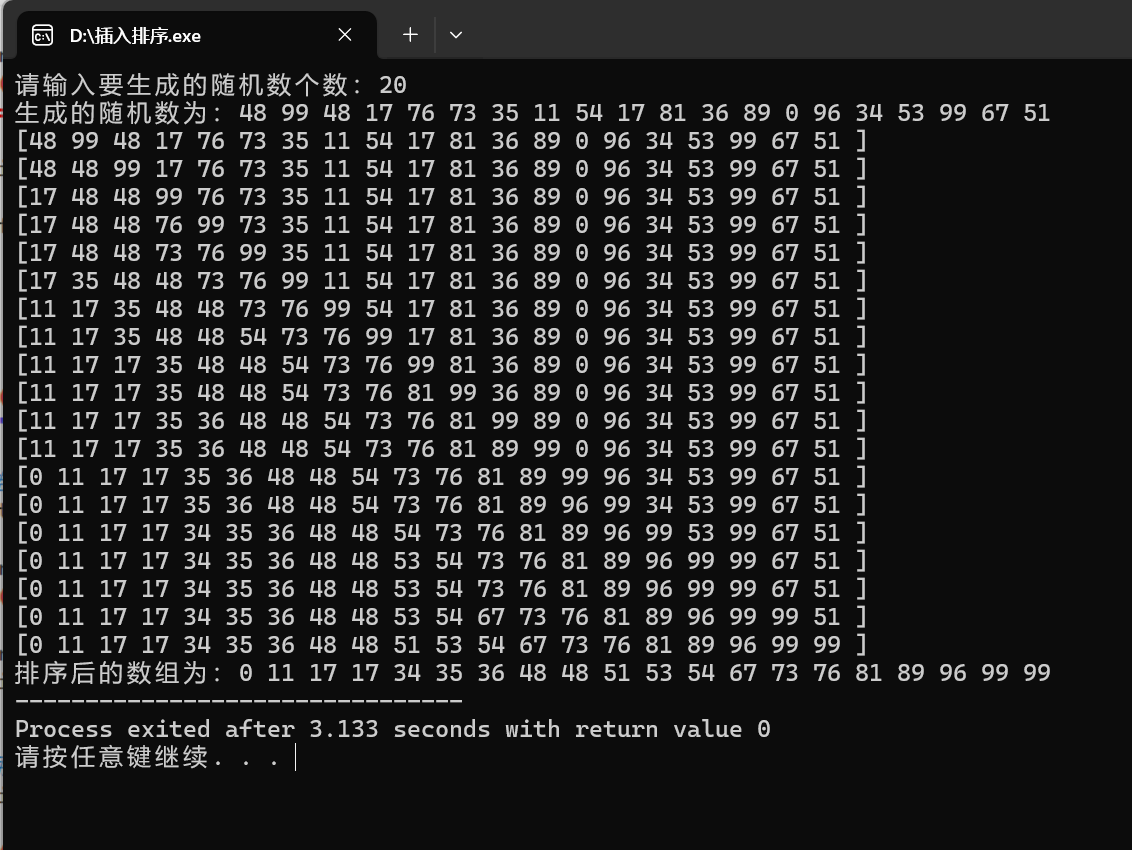
插入排序是一种简单且常用的排序算法，它通过构建有序序列，对未排序的元素逐个进行插入，从而实现排序。其基本思想是将数组分成已排序和未排序两部分，每次从未排序部分选择一个元素，将其插入到已排序部分的适当位置，使得已排序部分依然保持有序。

下面给出具体代码：





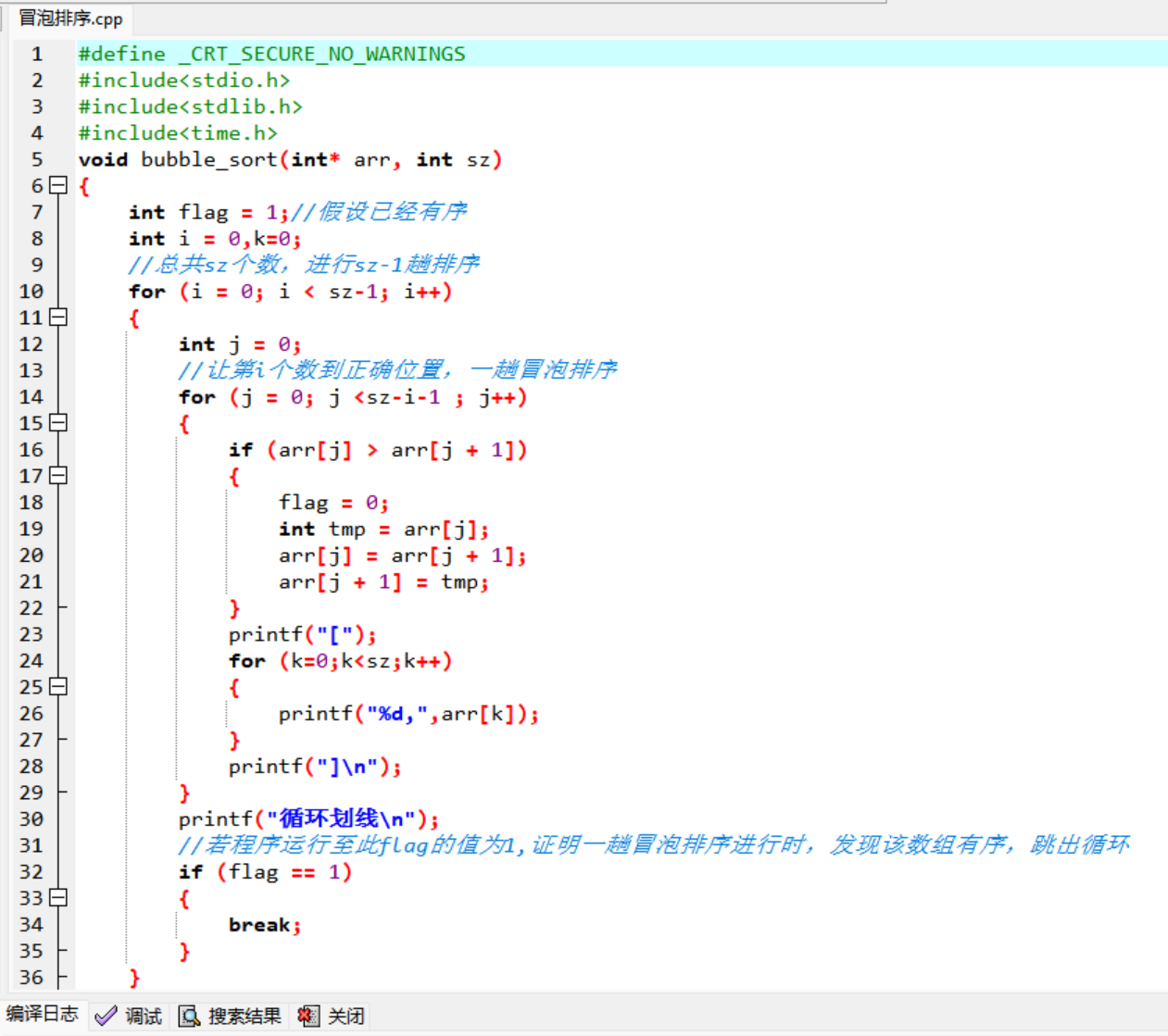
根据上面给出的代码，我们选取n=20时运行，此时我们可以得出相应的运行结果及具体迭代过程：



1. 冒泡排序

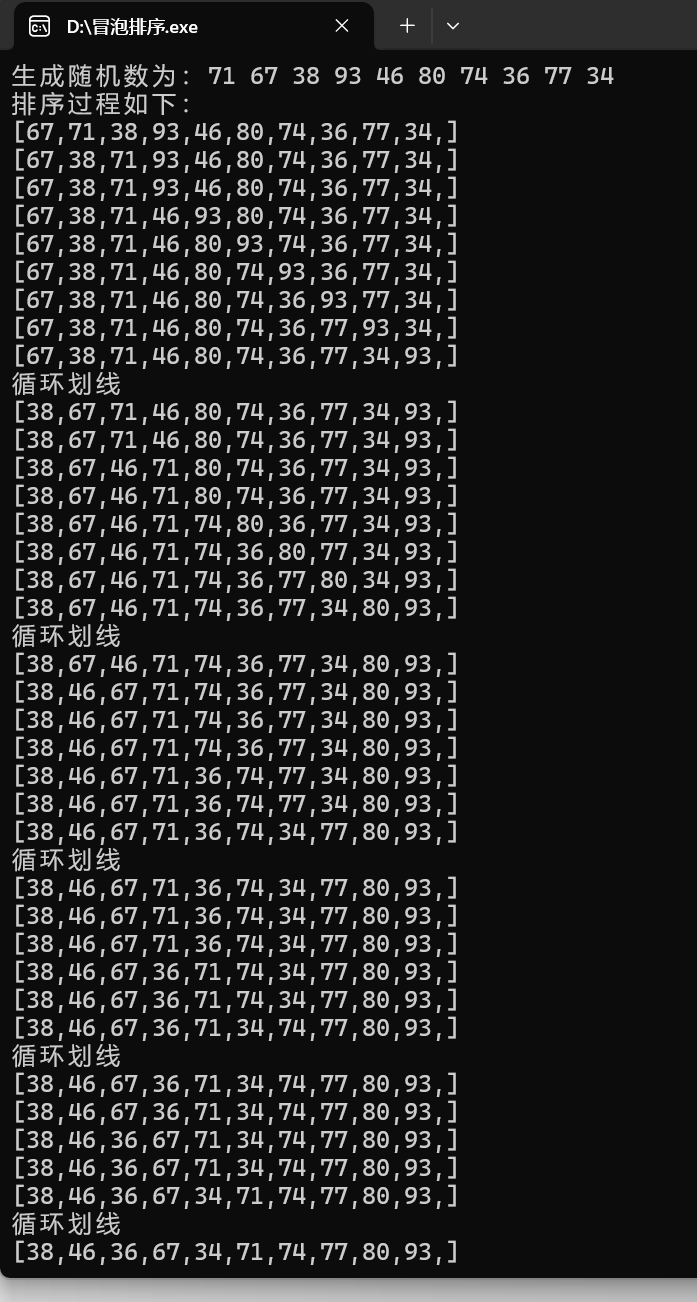
冒泡排序是一种简单且常用的排序算法，它通过比较相邻的元素并交换它们的位置，将较大的元素逐渐“冒泡”到右侧，从而实现排序。其基本思想是每次从数组的第一个元素开始，比较相邻的两个元素，如果顺序不对则交换它们的位置，这样每一轮比较都会将当前未排序部分的最大值移动到最右边。重复进行这个过程，直到整个数组排序完成。

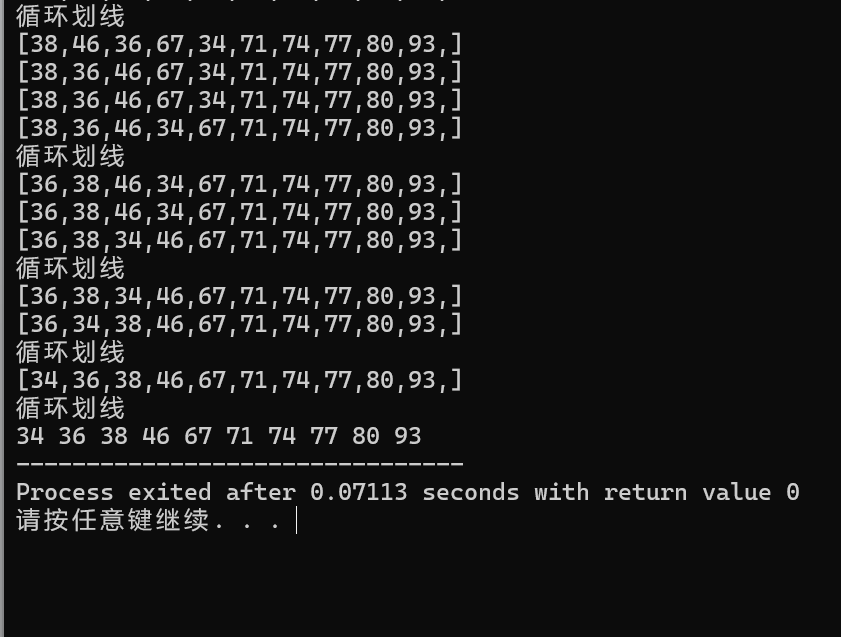
下面给出具体代码：





根据上面给出的代码，我们选取n=10时运行，此时我们可以得出相应的运行结果及具体迭代过程：

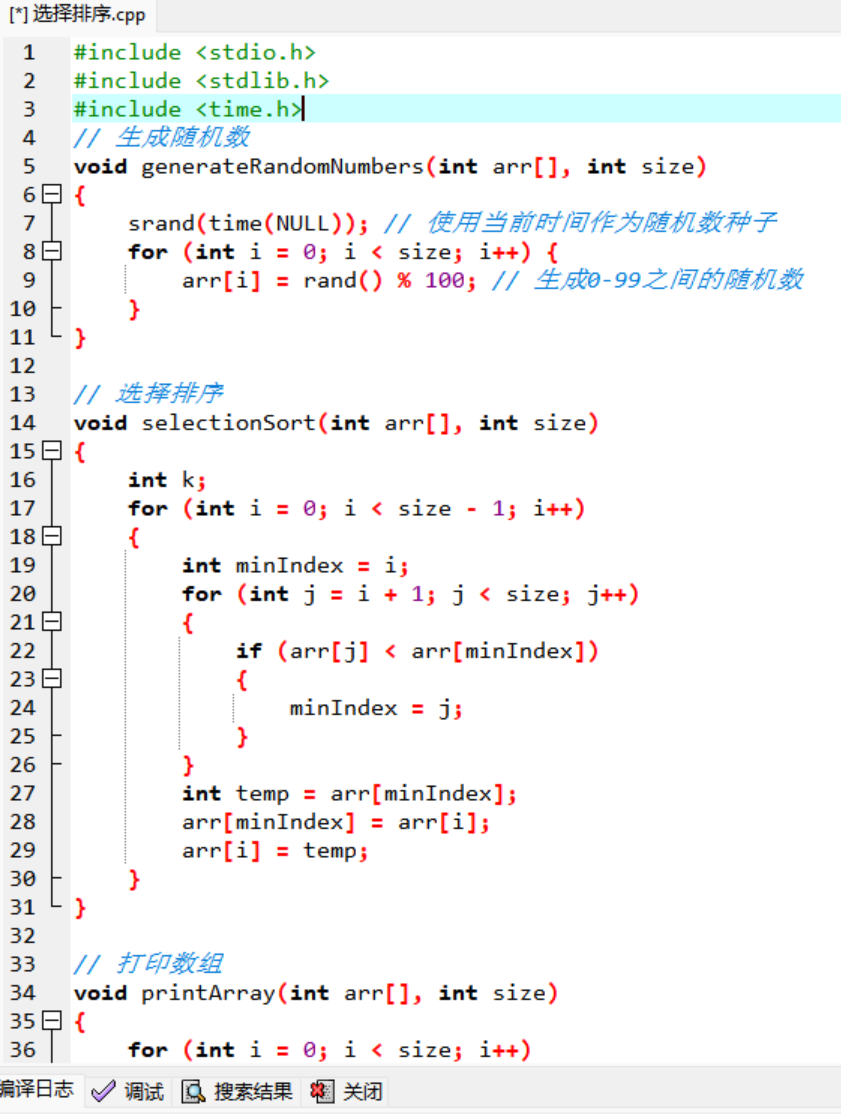


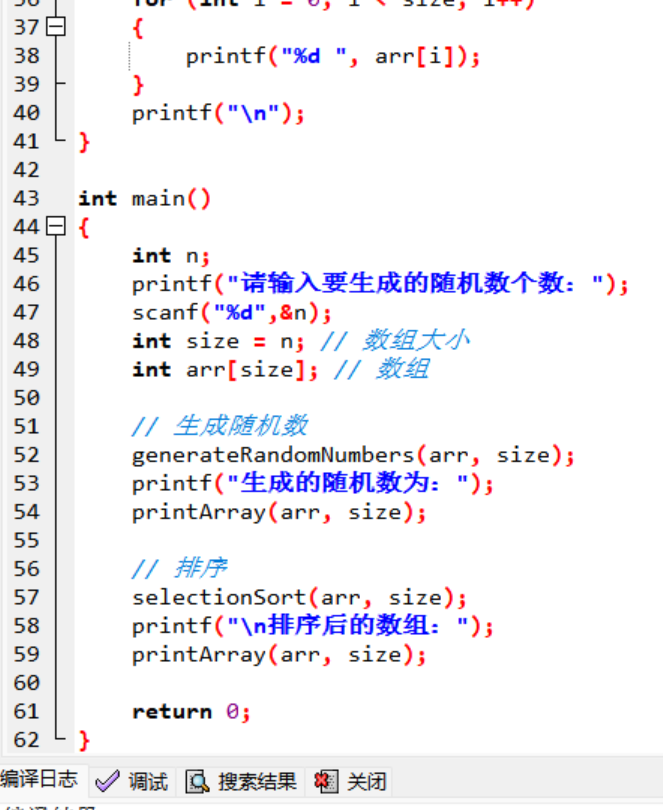


1. 选择排序

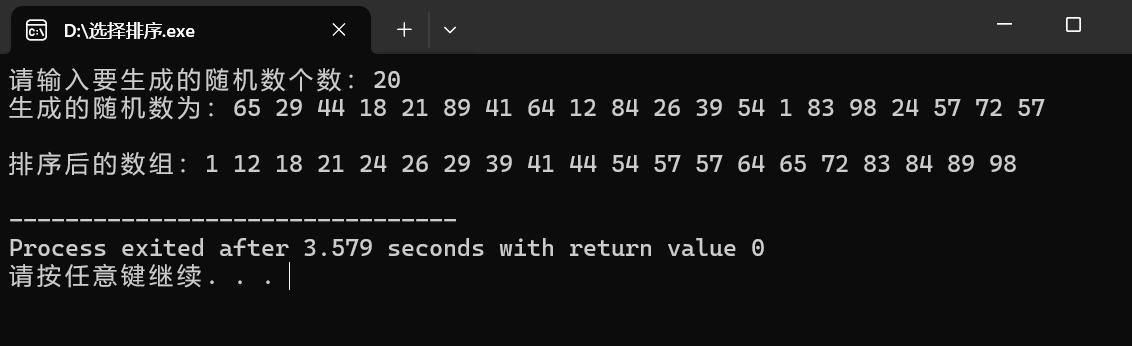
选择排序是一种简单且常用的排序算法，它的基本思想是每次从未排序的元素中选择最小（或最大）的元素，将其与未排序部分的第一个元素交换位置，从而逐步构建有序序列。具体实现时，通过不断选择最小（或最大）的元素，并将其放置到正确的位置上，直到所有元素都排好序。

下面给出具体代码：





根据上面给出的代码，我们选取n=20时运行，此时我们可以得出相应的运行结果：



这五种排序方法可以按照运行速度从快到慢的顺序进行排序：

1. 归并排序：归并排序的平均时间复杂度为O(nlogn)，因此在大多数情况下，它是这五种排序方法中最快的。

2. 快速排序：快速排序的平均时间复杂度也为O(nlogn)，但是在实际应用中，它通常比归并排序略慢一些。

3. 插入排序：插入排序的平均时间复杂度为O(n^2)，在处理较小规模的数据时，插入排序的性能可能会比快速排序和归并排序更好。

4. 选择排序：选择排序的平均时间复杂度也为O(n^2)，它比插入排序稍慢一些，因为它需要进行更多的比较操作。

5. 冒泡排序：冒泡排序的平均时间复杂度也为O(n^2)，它是这五种排序方法中最慢的，因为它需要进行更多的交换操作。

需要注意的是，以上排序方法的运行速度取决于输入数据的规模和特征。在实际应用中，选择适合特定情况的排序方法非常重要。

1. **两种遍历算法**
2. 广度遍历

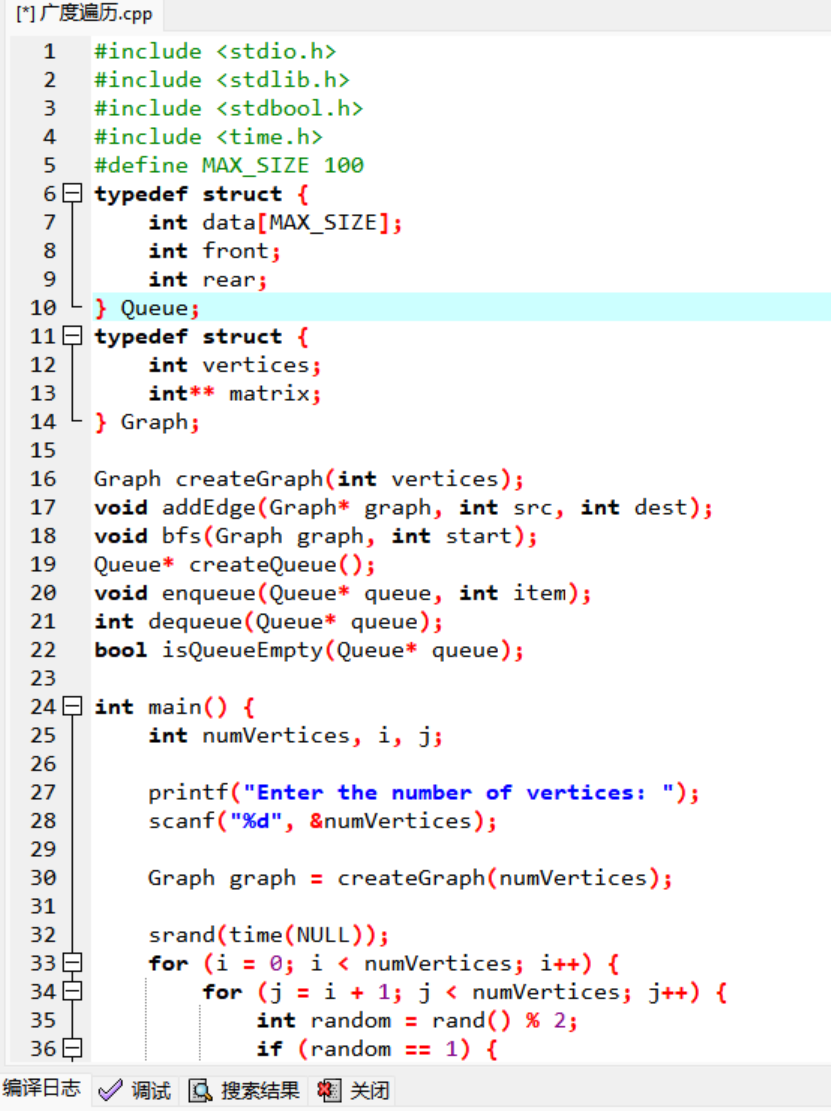
广度优先遍历（BFS）是一种图的遍历算法，用于遍历或搜索图中的节点。BFS从图的起始节点开始，逐层地遍历图的节点，先访问起始节点的所有邻接节点，然后再逐层访问邻接节点的邻接节点，直到遍历完所有可达节点。

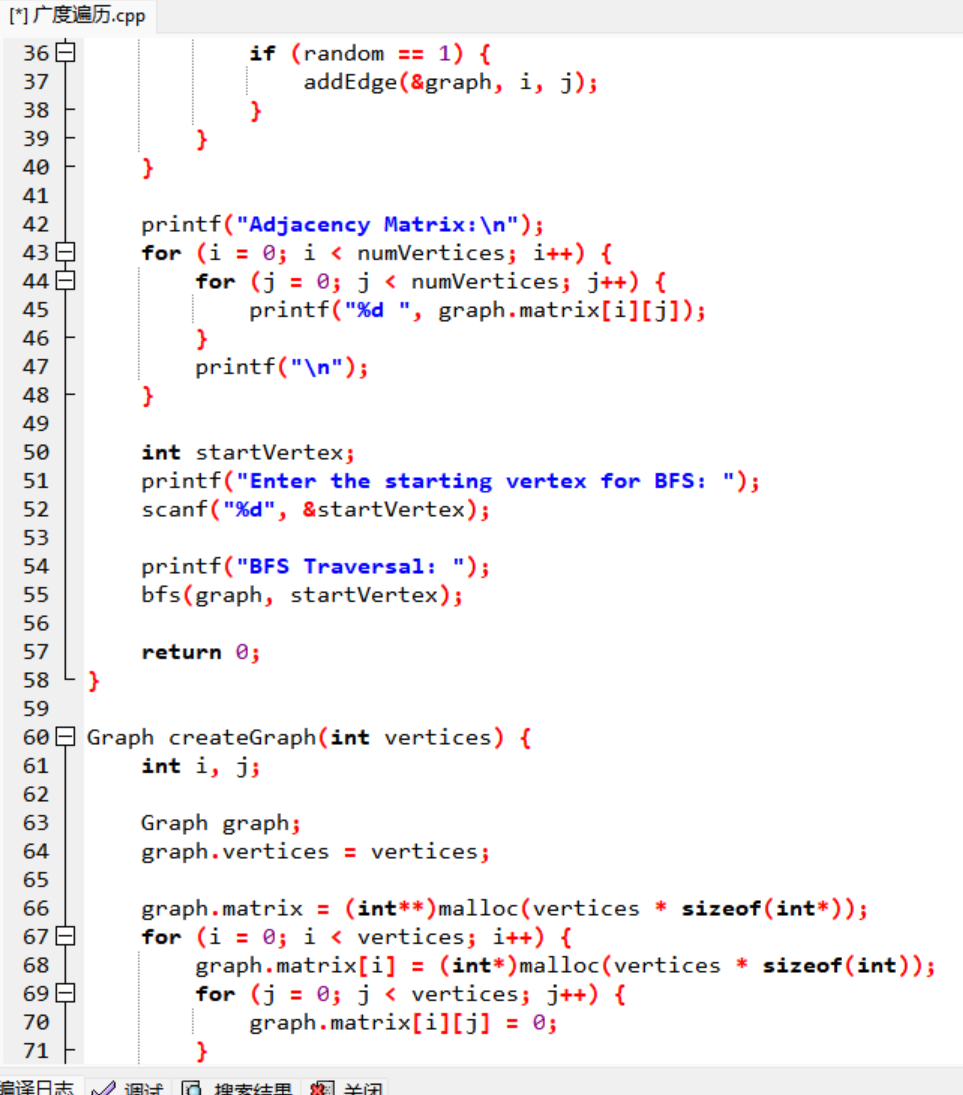
BFS的实现通常使用队列来辅助进行节点的访问。具体步骤如下：

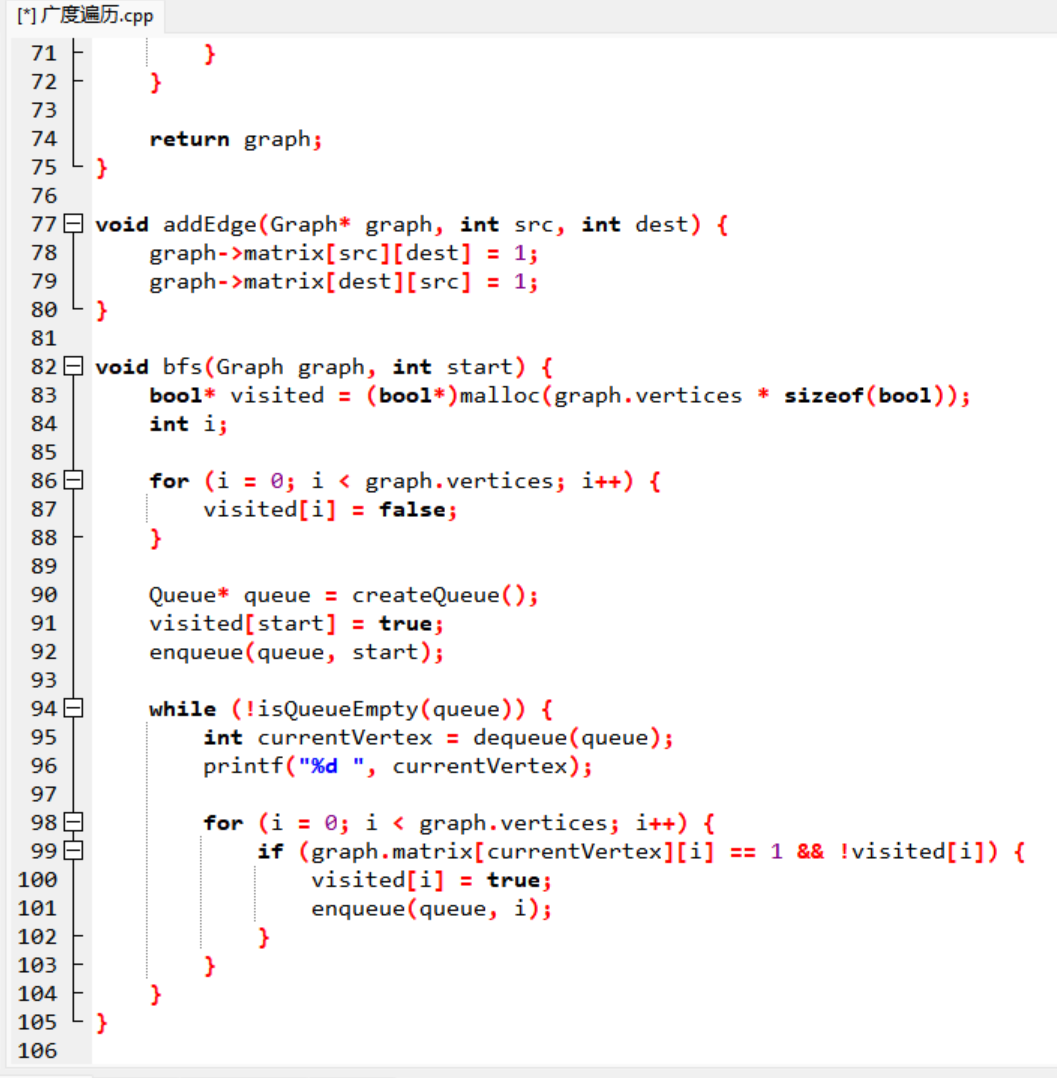
1. 将起始节点放入队列中。
2. 从队列中取出一个节点，并访问该节点。
3. 将该节点的所有未访问过的邻接节点放入队列中。
4. 重复步骤2和步骤3，直到队列为空。

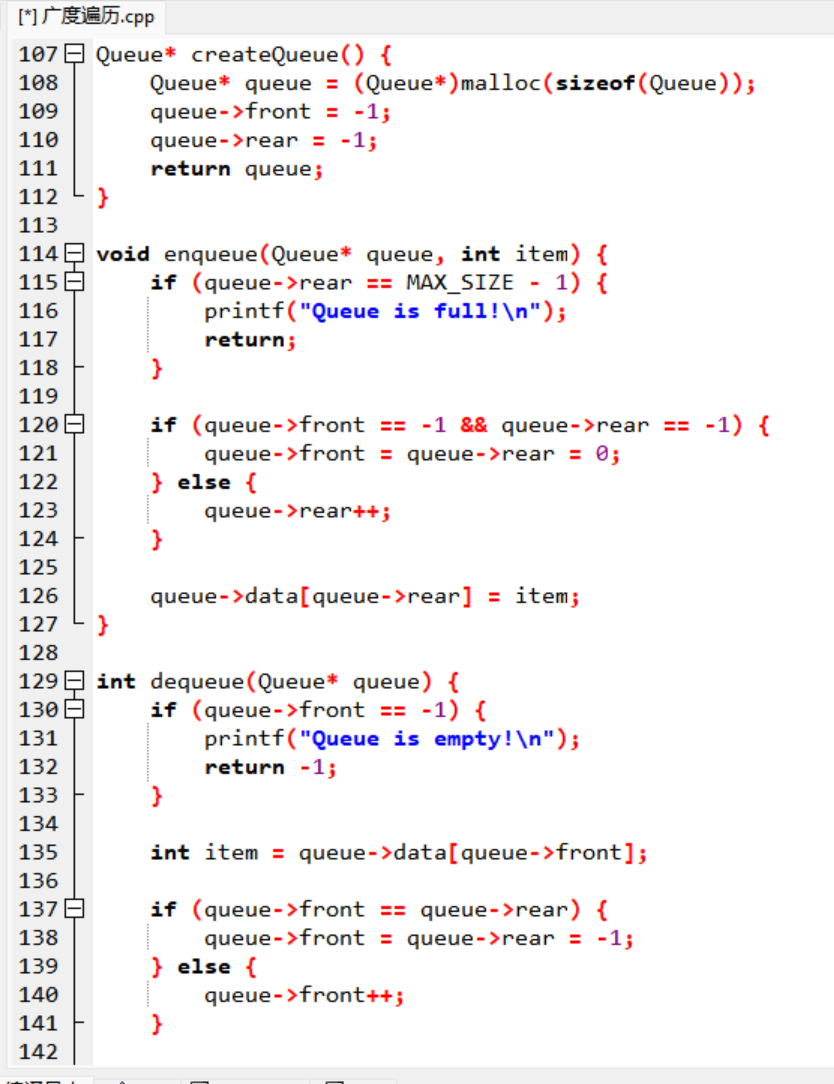
BFS可以应用于许多问题，例如查找图中的最短路径、查找图中的连通分量等。在实际应用中，BFS还可以用于解决迷宫问题、寻找社交网络中的关系等。需要注意的是，BFS遍历的图可以表示为邻接矩阵或邻接表的形式。根据实际情况，选择适合的数据结构来存储图的信息，以便实现BFS算法。

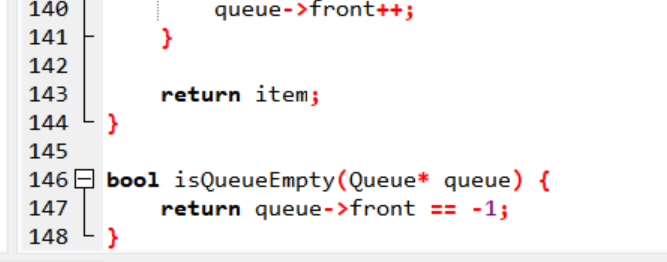
具体代码如下：



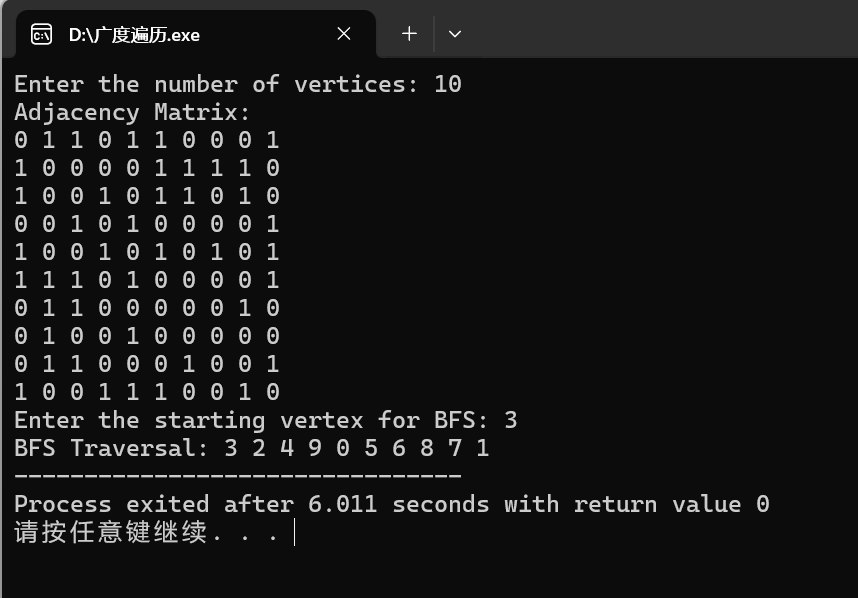








随机生成一个10阶的邻接矩阵，并对这个矩阵进行遍历，以3为起始点



1. 深度遍历

深度优先遍历（DFS）是一种图的遍历算法，用于遍历或搜索图中的节点。DFS从图的起始节点开始，沿着一条路径尽可能深地遍历，直到无法继续深入，然后回溯到上一个节点，继续探索其他未访问过的路径。DFS的实现通常使用递归或栈来辅助进行节点的访问。具体步骤如下：

1. 访问起始节点，并将其标记为已访问。

2. 从起始节点出发，选择一个未访问过的邻接节点。

3. 递归地对选中的邻接节点进行深度优先遍历，即重复步骤1和步骤2。

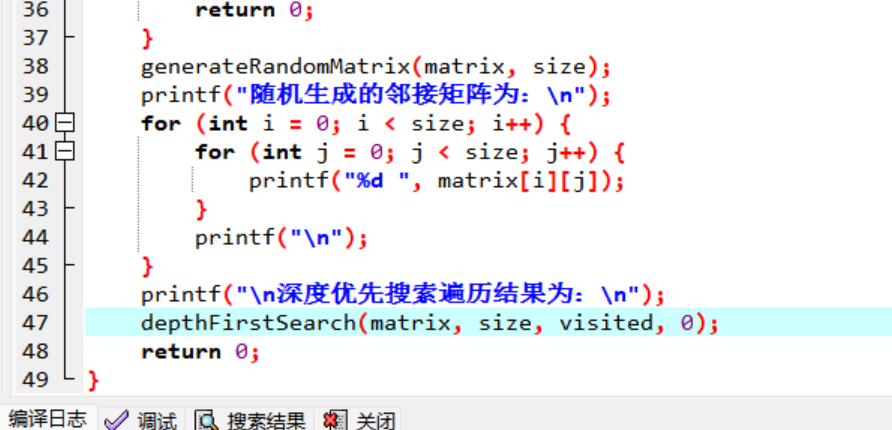
4. 当无法继续深入时，回溯到上一个节点，继续探索其他未访问过的路径。

DFS可以应用于许多问题，例如查找图中的连通分量、判断图中是否存在环、进行拓扑排序等。在实际应用中，DFS还可以用于解决迷宫问题、生成图的最小生成树等。

需要注意的是，DFS遍历的图可以表示为邻接矩阵或邻接表的形式。根据实际情况，选择适合的数据结构来存储图的信息，以便实现DFS算法。

具体代码如下：





随机生成一个10阶的邻接矩阵，并对这个矩阵进行遍历：

