背包问题实验报告

计算机2306 张钊源 2236114968

1、题目：

假设有一个能装入总体积为T的背包和n件体积分别为w1,w2,…wn的物品，能否从n件物品中挑选若干件恰好装满背包，即使w1+w2+…+wm=T，要求找出所有满足上述条件的解。

2、解题思想：

（1）主要使用数据结构Stack存储当前组合和所有的解，以及Backtrack回溯算法来遍历所有可能的物品组合。

（2）自己练习了Stack的编写，使用链表完成，实现了pop()、push(i)、isEmpty()、peek()、toArray()、size()等功能。

（3）程序总体有五部分：

①public BackpackProblem(int t, int[] items)：

对BackpackProblem对象进行初始化，该对象包含容量T，物品数组items，和两个存储栈current，solutions用来存储当前正在考虑的物品组合和所有找到的符合条件的解

②private void save()：

将当前找到的解保存到 solutions 栈中。首先通过 current.toArray() 获取当前物品组合的数组，并将它们依次压入 solutions 栈中。

③private void BackTrack(int i, int sum)：

这是回溯算法的核心部分，它递归地搜索所有物品的组合。当 sum == T 时，表示找到了一个符合条件的解，此时调用 save() 将该解保存。当 sum > T 或者已经遍历完所有物品时，递归返回。否则，分别进行两次递归：一次将当前物品 i 加入组合中，另一次则跳过该物品。

④private void print()：

该方法负责从 solutions 栈中弹出所有保存的解并打印它们。打印时先将solutions 栈中的元素全部转移到临时栈 temp 中，再从 temp 中按顺序打印每个解。

⑤public void solve()：

该方法用来解题，调用 BackTrack() 从物品列表的第一个物品开始进行递归搜索，直到遍历所有解并调用 print() 输出最终结果。用户在使用时只需要设置好 T 与 items 后调用该函数即可求解，因此该函数设置为 public，其余则为 private。

3、算法介绍：

使用回溯算法：

S1:当背包恰好装满时，储存该组解入 solutions 栈，并存储该组解的物品数量，结束递归

S2:如果当前解超出背包容量或所有物品都已处理完成，结束递归

S3:对每个物品从两个分支进行遍历：第一，选择该物品，将其 push 入 current 栈，更新当前总体积并继续递归；第二，跳过该物品，直接对下一物品进行递归。通过这种遍历方式，我们可以得到所有不重复的物品组合并找出所需解

4、输入输出：

（1）*example1：*

初始化：items = {1, 8, 4, 3, 5, 2}, T = 10

输出：

(1, 4, 3, 2)

(1, 4, 5)

(8, 2)

(3, 5, 2)

（2）*example2：*

初始化：items = {5, 9, 6, 2, 3, 7, 10, 12, 1, 8, 11, 18}, T = 20

输出：

(5, 9, 6)

(5, 9, 2, 3, 1)

(5, 6, 2, 7)

(5, 6, 1, 8)

(5, 2, 3, 10)

(5, 2, 12, 1)

(5, 3, 12)

(5, 3, 1, 11)

(5, 7, 8)

(9, 6, 2, 3)

(9, 2, 1, 8)

(9, 3, 7, 1)

(9, 3, 8)

(9, 10, 1)

(9, 11)

(6, 2, 3, 1, 8)

(6, 2, 12)

(6, 2, 1, 11)

(6, 3, 10, 1)

(6, 3, 11)

(2, 3, 7, 8)

(2, 7, 10, 1)

(2, 7, 11)

(2, 10, 8)

(2, 18)

(3, 7, 10)

(7, 12, 1)

(12, 8)

(1, 8, 11)

5、总结：

（1）时间复杂度：

$O(2^N)$ （N 为物品总数）

（2）空间复杂度：

$O(N \cdot 2^N)$ （N 为物品总数）

（3）亮点：

自行尝试使用链表实现了 Stack class，加深了对链表与栈的理解，且避免了数组的扩容问题，提高效率。编写 toArray 方法将 current 中的链表转化为数组，使解便于存储。通过将每组解的物品数量存储于每组解之后，于输出前将栈整个颠倒过来，使解的顺序正确，此时物品数量在每组解之前，从而便于 print 时对每组不同解的辨别。

（4）不足：

回溯算法的时间、空间复杂度均为指数级，在 items 数量较多时，可能会导致编译时间过长、内存占用过多。

八皇后问题实验报告

计算机2306 张钊源 2236114968

1、题目：

设在初始状态下在国际象棋的棋盘上没有任何棋子（这里的棋子指皇后棋子）。然后顺序在第1行，第2行……第8行上布放棋子。在每一行中共有8个可选择的位置，但在任一时刻棋盘的合法布局都必须满足3个限制条件（1）任意两个棋子不得放在同一行（2）任意两个棋子不得放在同一列上（3）任意棋子不得放在同一正斜线和反斜线上。

2、解题思想：

（1）使用了自己尝试完成的deque包，包含两个class LinkedListDeque与ArrayDeque，以及一个interface Deque，实现了addFirst, addLast, get(i), removeFirst, removeLast等功能。在该问题中用ArrayDeque以存储所有解和通过get(i)输出解。

（2）一维数组board[]存储每行中皇后所在列的索引。由于皇后不能在同一行，所以每行只会有一列存在皇后，使用一位数组存储可以降低空间复杂度，同时使程序更加简洁。Deque solutions以数组作为其元素，存储所有board代表的解，以便实现输出随机解。通过对类常量SIZE的修改，可以实现对任意N皇后问题的求解。

（3）程序主体位于class EightQueens，有三部分：

①private boolean judge(int row)：

数组board[]的使用默认了同一行中不存在两个皇后，因此只需判断列与斜线。我们对行序号row前的所有行进行遍历：第一，如果此前有行中存在与该行相同的列序号，则说明存在两皇后位于同一列，不符合条件；第二，如果存在*abs*(row - i) == *abs*(board[row] - board[i])，则说明斜率绝对值为1，存在两皇后位于同一斜线，不符合条件。若不属于这两种情况，则皇后的位置合法，判断程序通过。

②private void backtrack(int row)：

该问题的算法核心与背包问题相似，也为递归回溯的思路。该方法能够遍历每种摆放的可能性，以类似于枚举的方式解决问题。算法详情见算法介绍部分。

③public void printSolution(int[] solution)：

对当前数组对应的解进行输出。以 Q 代表皇后所在位置，以·代表空格。使用二重循环输出，当 board 中皇后所在列与当前列相等时，可知皇后位置位于此处。

（4）public void solve()用于求解该问题，能够将所有解储存于 solutions 中，同时得出解法总数。

（5）制作了两个继承于 EightQueens 的 class Allsolutions 与 RandomSolution，分别实现输出所有解与输出随机解。

3、算法介绍：

使用递归：

S1：当 row == SIZE 时，说明所有皇后已完成排列，将该组解加入 solutions 中，结束递归。

S2：对该 row 的每个 column 进行遍历，将皇后置于每个位置进行尝试，如果位置合法，则进入递归，对下一行进行操作

4、输出：（RandomSolution）

Total solutions: 92

One solution:

```
· · · Q · · · ·
· · · · · · · Q
Q · · · · · · ·
· · · · Q · · ·
· · · · · · Q ·
· Q · · · · · ·
· · · · · Q · ·
· · Q · · · · ·
```

5、总结：

（1）时间复杂度：O(N!) （N = SIZE）

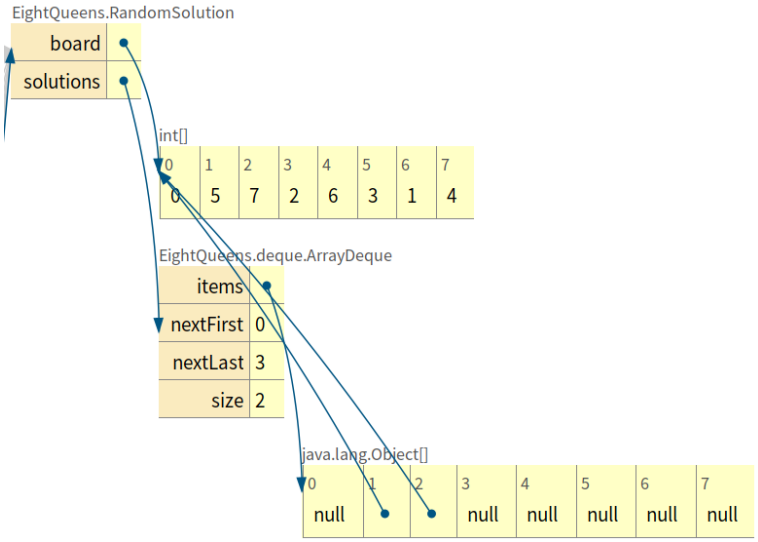
（2）空间复杂度：O(N) （N = SIZE）

（3）该题完成心得：

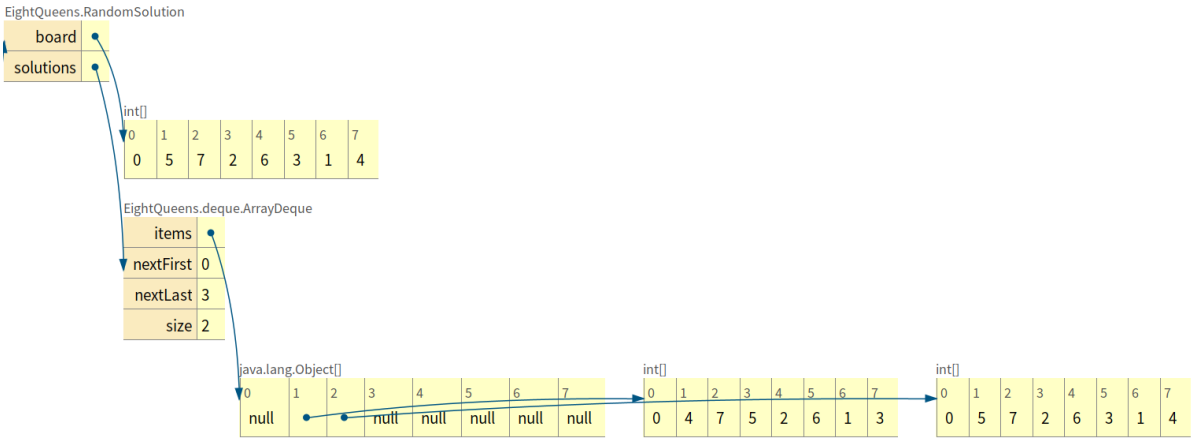
①通过 deque 包的编写，我的 java 编写能力提升，对双端循环队列有了更深的理解。

②在处理解时，我对 reference type 有了更深的理解：若直接将 board 存入 solutions，其实是将 board 的地址加入 solutions。在 board 改变时，solutions 中的解也会随之改变，因此需使用 board.clone()，来加入每组解的副本。

addLast(board)：



addLast(board.clone())：



③通过尝试更大的 SIZE，我认识到回溯法在时间上的不足。这种方法的本质是暴力枚举，时间曲线呈指数型上升，一旦问题复杂，耗费的时间便会快速增长。

（4）亮点：

①使用 ArrayDeque 以数组形式来存储所有解，而非直接存储在数组中，这样使只输出一组解成为可能。

②利用 Random 库实现了随机解的输出，提供两种不同的输出模式，既能获取全部解，也能获取某个随机的解。

约瑟夫环问题实验报告

计算机2306 张钊源 2236114968

1、题目：

设编号为1，2，…，n(n>0)个人按顺时针方向围坐一圈，每人持有一个正整数密码。开始时任意给出一个报数上限m，从第一个人开始顺时针方向自1起顺序报数，报到m时停止报数，报m的人出列，将他的密码作为新的m值，从他在顺时针方向上的下一个人起重新自1报数；如此下去直到所有人全部出列为止。

2、解题思想：

该题的解决以循环链表为基础，将所有人通过链表相连构成一个环状结构，在经过m次报数时，将对应的人移出链表，将对应序号存储入people数组，并更新报数次数m与总人数n。在构造count方法时，尝试了迭代法与递归法两种不同方式。

（1）基本介绍：

创建Node class来设置链表的节点，N为人数，m为初始报数上限，数组password存储每个人持有的密码，first为第一个人，people存储出列顺序，peopleIndex为出列总人数。初始化问题对象public Josephus(int N, int m, int[] password)，设置其N，m，password.

（2）链表功能添加：

①private void initial()：根据N与password初始化链表，将每个人及其密码加入链表，构建为循环链表。

②private int get(int index)：获取index位置的密码，用于更新报数值m。

③private void remove(int index)：让index位置的人出列，将其密码记录在people数组中，同时使出列总人数peopleIndex加1。

（3）主要功能count的实现:

用迭代法和递归法两种方法分别实现了count，具体见算法介绍。

（4）printPassword 与 solve：

print 用于输出结果，solve 用于将解题过程打包封装。

3、算法介绍：

（1）private void count_Iteratively(int m, int n)：

设置 m，n 为形参，其中 m 为每次报数上限值，n 为当前人数。

利用循环，直接通过计算确定每次报数最终出列的人的序号，获取其密码用以更新 m 值，同时 n-1，完成操作后将其移除链表。

（2）private void count_Recursively(int m, int n, int index, int count)：

设置 m，n，index，count 四个形参，其中 m 为每次报数上限值，n 为当前人数，index 为当前人的序号，count 为当前报数值。

S1：当该次报数值 count 达到 m 时，完成当前递归，进行将当前人出列并存储序号的操作，并将 count 恢复至 0。

S2:当 n==0 时，说明所有人均已出列，递归结束。

S3:每次递归时，index 向前一位，count+1，代表报数的过程。

4、输入输出：

输入：

7 20

3 1 7 2 4 8 4

输出：

Answer solved by iteration: 8 3 2 4 1 7 4

（Answer solved by recursion: 8 3 2 4 1 7 4）

5、总结：

（1）时间复杂度：O（N^2）

（2）空间复杂度：O（N）

（3）在使用链表完成该题后，我发现代码过于冗长、时间复杂度较高，且输出的是每个人的密码值，而非他们对应的序号。因此我采用数组与递推公式设计了 class JosephusOptimized，通过数学计算直接锁定每次出列人的序号，进行输出。该算法时间复杂度为 O(N)，且代码更加简洁明了，避免了不必要的部分。但我认为使用链表完成的代码也有可取之处，故而一起保留下来。

此时输出为出列者的序号：6 1 4 2 7 3 5

（4）此后，我思考了链表实现的不足之处，在于 remove 与 get 函数每次都需要从头遍历链表，因此，我重新编写了 class Josephus_v2，将链表重新编排，在创建对象时进行初始化，并且加入了 pre 指针以时刻跟踪报数过程，避免了对链表的多次遍历。在递归时，我放弃了逐个递归模拟报数过程，用数学计算获取目标位置从而在对链表的一圈遍历之内得到出列者。此时时间复杂度降低为 O（N）。

（5）通过该题的完成，我对链表与数组二者的优劣有了更深入的认识，并且加强了自己简化代码，优化时间、空间复杂度的观念，提升了我的编程能力。

二叉排序树与平衡二叉树实验报告

计算机2306 张钊源 2236114968

1、题目：

分别采用二叉链表和顺序表作存储结构，实现对二叉排序树与平衡二叉树的操作。

2、解题思想：

（1）初始化：

先明确每个class需要实现的功能，如insert、delete、search等，设置BinaryTree interface来起统筹、提醒作用。设置树的结点class Node，为每个点配置data，leftchild，rightchild。在AVL的实现中，需要对左右子树高度进行比较，因此增加参数height。

（2）BST的二叉链表实现：

①插入：对于每个结点，其左子树结点值小于其值，右子树结点值大于其值。在操作过程中，调用递归方法，若插入的data小于当前结点的data，则向左递归，否则向右递归，直至找到null结点后，插入新结点。

②删除：在递归遍历树查找到需要删除的元素后，分为三种情况：一，叶结点的删除，只用将其变为null即可；二，若有left child或right child，需要将其子结点提升至被删除结点；三，若有两个子结点，则可通过查找左子树的最大节点或右子树的最小节点替代被删除节点的值，然后递归删除替代节点。

③中序遍历：通过递归遍历左子树、访问当前节点、遍历右子树，可以得到节点值的升序排列。

④平均查找长度：递归遍历树的每个节点，计算每个节点的深度，并统计所有节点的总深度与节点数目，然后求出平均值。

（3）BST的顺序表实现：

①存储方法：采用ArrayList结构，可以动态调整大小，便于树的扩展。在数组中，从0开始记录结点序列，假如某结点序列为i，那么其parent为（i-1）/2，其left child为2 * I + 1，其right child为2 * I + 2。

②插入：若数组不足以容纳新结点，自动扩充数组并以 null 填充。插入操作与链式 BST 思路基本一致。

③删除：在进行删除时，我尝试采用了迭代的方法完成查找工作，直接对 index 进行修改。在找到删除结点后，我分为四种情况讨论：无子结点，直接用 null 替换；有 left child 或 right child，先以之替换，后递归调用 delete 以删除替换结点；有两个子结点，找到左子树的最右结点替换，并递归删除。删除成功后 break 退出循环。

④中序遍历与 ASL：基本同上。在中序遍历时需考虑 index 是否超载，及时对数组进行扩容。

（4）AVL 的实现：

与链式 BST 基础思路一致，但需要注意计算平衡因子（左子树高度-右子树高度），在失衡（平衡因子绝对值>1）时及时进行旋转操作。在 LL（左子树左高）时，对失衡结点右旋；在 RR（右子树右高）时，对失衡结点左旋；在 LR（左子树右高）时，先对失衡结点的 left child 左旋，再对失衡结点右旋；在 RL（右子树左高）时，先对失衡结点的 right child 右旋，再对失衡结点左旋。在进行插入、删除时，要实时更新每个结点的高度，并检查是否失衡。

（5）图形化实现：

我尝试使用工具 Graphviz 完成树结构的可视化实现。将树的结构存储在.dot 文件后，Graphviz 可通过读取.dot 文件画出树的大致形状。在 AVL 树中，我尝试在每次 insert 以及 delete 后生成对应的树结构图，能够清楚地看到树的创建过程。

3、算法分析：

算法的核心思想为树的递归，总体思路已经介绍，在此分析各 method 的时间复杂度。

（1）**插入：**

插入元素时，时间复杂度为 O(h)，其中 h 是树的高度。最坏情况下，当树退化为链表时，h 接近 n，插入操作的时间复杂度为 O(n)。但在 AVL 树中，插入的时间复杂度为 O(log n)。

（2）**删除：**

删除操作类似插入操作，最坏情况下的时间复杂度为 O(h)，即 O(n)。如果树保持平衡，时间复杂度为 O(log n)。

（3）**中序遍历：**

中序遍历树的时间复杂度为 O(n)，因为每个节点都需要访问一次。

（4）**ASL：**

计算 ASL 时需要遍历整个树，与树的平衡性密切相关。对于 AVL 与较平衡的 BST，ASL 时间复杂度接近 O(log n)；而对于极度不平衡的树（如形成链表结构），ASL 时间复杂度接近 O(n)。

（5）**可视化生成：**

生成.dot 文件并通过 Graphviz 工具生成树的可视化图形。这个过程的时间复杂度为 O(n)，因为每个节点和边都需要被写入文件。

（6）**空间复杂度：**都涉及到树的递归调用栈，为 O(log n)。

4、输入输出：

（1）LinkedBST：

输入：16 30 2 21 8 6 1 23 7 15 10 9 5 17 12

输出：（最后一步前的输出省略）

After inserting 12:

1 2 5 6 7 8 9 10 12 15 16 17 21 23 30

Average Search Length: 3.8

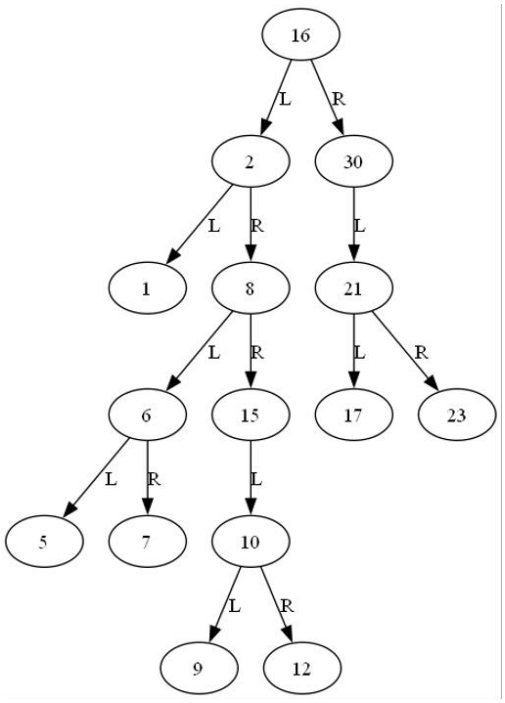
The BST image is saved in binary_tree.png.

Enter element to delete from BST:

15

In-order Traversal after Deletion:

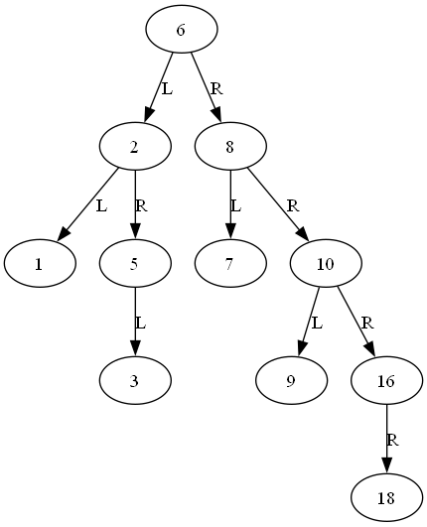
1 2 5 6 7 8 9 10 12 16 17 21 23 30



（2）ArrayBST：

输入：6 2 1 5 8 10 3 9 16 3 18 7 7 10

输出：（最后一步前的输出省略）



After inserting 10:

1 2 3 5 6 7 8 9 10 16 18

Average Search Length: 3.090909090909091

The BST image is saved in binary_tree1.png.

Enter element to delete from BST:

18

In-order Traversal after Deletion:

1 2 3 5 6 7 8 9 10 16

（3）AVL：

输入：6 5 2 8 7 1 3 10 9 4

输出：

Generated graph for tree after inserting 4 at

AVL_Trees_Graphs\AVL_10_1734535340566.dot

After inserting 4:

1 2 3 4 5 6 7 8 9 10

Average Search Length: 2.9

Enter element to delete from AVL:

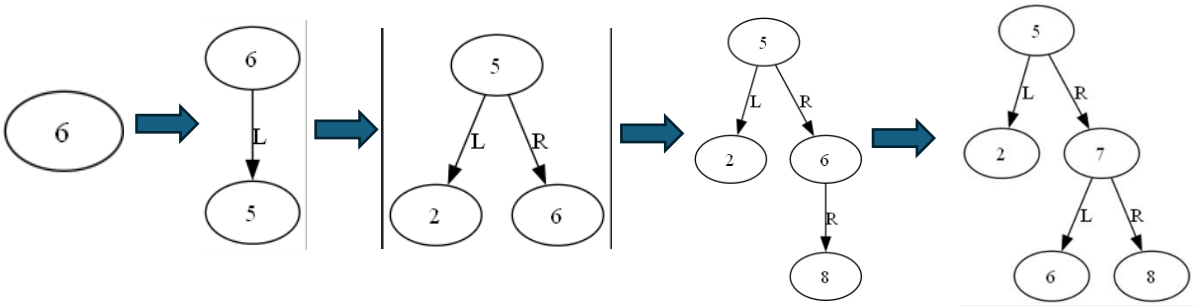
5

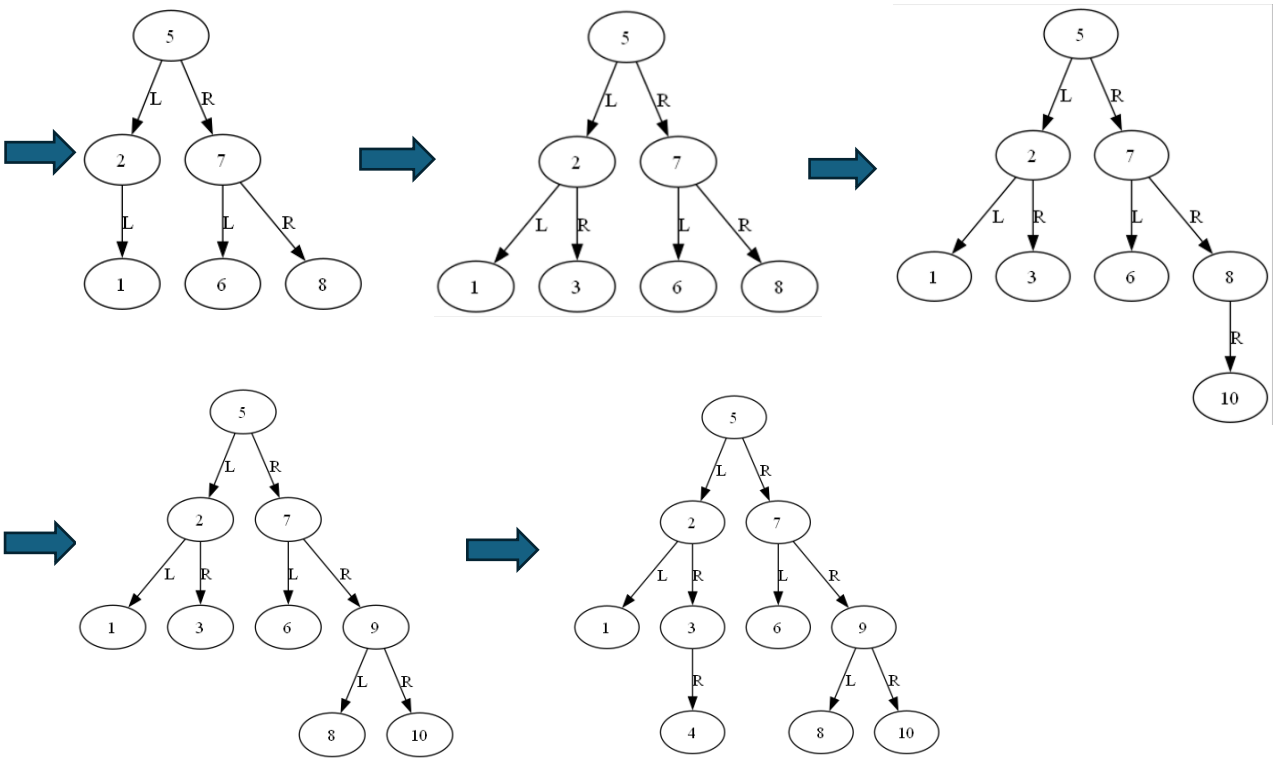
Generated graph for tree after deleting 5 at

AVL_Trees_Graphs\AVL_Delete_5_1734535343812.dot

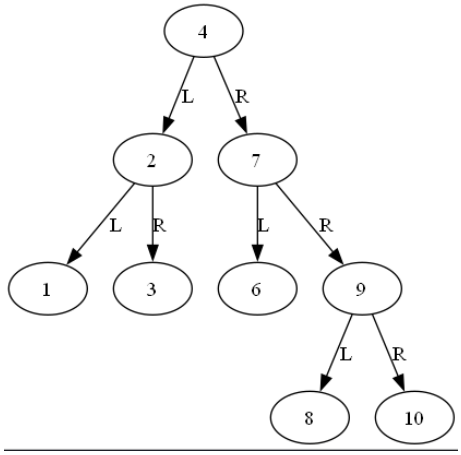
In-order Traversal after Deletion:

1 2 3 4 6 7 8 9 10





删除 5 后：



5、总结：

（1）通过本次对 BST 与 AVL 的多种实现，我对树的结构、BST 的原理、ArrayList 的应用、树的旋转等知识有了更深刻的理解，掌握了这些知识的核心要点及其代码实现。

（2）探索了树结构可视化实现的可能性，对 Graphviz 工具有了初步的了解。

（3）在实现 ArrayBST 时，我被索引越界问题等数组问题困扰许久。我先尝试用普通的数组完成，发现在树扩展后，数组的动态扩容较为麻烦，因此转而采

用 ArrayList 结构。我大致阅读了 ArrayList 的关键方法，并查阅资料理解了如何使用这些方法。在中序遍历时，发生了索引越界现象，这使我意识到这个问题，并了解了 ensureCapacity（index）的使用，在 insert 与 inorderTraversal 之前检查数组容量并及时扩容，避免了这个问题。在实现 delete 时，我尝试使用了 while 循环来构建一个迭代的结构对树进行遍历，我认为这对减少递归带来的时间、空间占用有一定的效果。

（4）我认为本次实验的完成较为完善，但仍存在一些改进空间，如在生成 AVL 树每次插入后的图片时，可采用多线程的方法提高效率。在后续的学习过程中，我也会尝试继续完善这些代码，并且精进对使用的算法、方法的掌握。

全国交通咨询模拟系统实验报告

计算机2306 张钊源 2236114968

1、题目：

出于不同目的的旅客对交通工具有不同的要求。例如，因公出差的旅客希望在旅途中的时间尽可能地短，出门旅游的游客则期望旅费尽可能省，而老年旅客则要求中转次数最少。编制一个全国城市间的交通咨询程序，为旅客提供两种或三种最优决策的交通咨询。

2、解题思想

本系统的实现基于图论中的有向图模型，将城市作为图的节点，交通连接（火车和飞机）作为图的边。通过构建高效的图结构，系统能够快速进行城市和交通连接的增删操作。同时，利用Dijkstra算法分别实现最快路线和最省钱路线的查找功能，以满足用户的不同需求。

（1）基本介绍

系统主要由以下class组成：

① TransportType：枚举类型，定义交通工具类型（火车、飞机）。

② City：表示城市，包含城市名称。

③ Transport：表示交通连接，包含出发地、目的地、交通工具类型、出发时间、到达时间、费用及火车编号等信息。

④ Graph：图结构，使用邻接表存储城市及其交通连接。

⑤ Path：表示一条路径，包含当前城市、所经过的交通连接列表、总费用和总时间。

⑥ RouteFinder：路径查找器，基于图结构实现最快路线和最省钱路线的查找。

⑦ TrafficConsultationSystem：主系统class，提供用户交互界面和功能模块。

（2）图结构设计

系统采用邻接表来存储图结构，以便高效地管理城市及其交通连接。class Graph 中使用 Map<String, List<Transport>>来表示每个城市及其连接的交通方式。这种结构在空间上较为节省，适合稀疏图的场景。

（3）路径查找算法

系统使用 Dijkstra 算法分别实现两种最优路线的查找：

① 最快路线：以总时间作为权重，优先选择总时间最短的路径。

② 最省钱路线：以总费用作为权重，优先选择总费用最低的路径。

3、算法介绍

（1）最快路线查找（findFastestRoute）

① 使用 PriorityQueue，按照路径的总时间进行排序。

② 从起始城市开始，逐步扩展到相邻城市，计算每条路径的累计时间和费用。

③ 记录每个城市到达的最佳时间，避免重复计算。

④ 当到达目的城市时，返回当前路径。

（2）最省钱路线查找（findCheapestRoute）

① 使用 PriorityQueue，按照路径的总费用进行排序。

② 从起始城市开始，逐步扩展到相邻城市，计算每条路径的累计费用和时间。

③ 记录每个城市到达的最佳费用，避免重复计算。

④ 当到达目的城市时，返回当前路径。

4、输入输出：

--- 全国城市交通咨询系统 ---

1. 编辑城市信息

2. 编辑交通时刻表

3. 查询最优路线

4. 退出

请输入您的选择: *3*

请输入起始城市: *北京*

请输入目的城市: *上海*

请输入最优决策（1 为最快/2 为最便宜）：*2*

请输入交通工具类型（火车/飞机）：*火车*

最优路线费用为 450.0 元。

路线详情:

火车编号: G1001

  火车: 北京 -> 天津, 出发时间 08:00, 到达时间 09:30, 费用: 100.0

火车编号: G1002

  火车: 天津 -> 徐州, 出发时间 10:00, 到达时间 12:30, 费用: 150.0

火车编号: G1003

  火车: 徐州 -> 上海, 出发时间 13:00, 到达时间 15:30, 费用: 200.0

--- 全国城市交通咨询系统 ---

1. 编辑城市信息

2. 编辑交通时刻表

3. 查询最优路线

4. 退出

请输入您的选择: *2*

--- 编辑交通时刻表 ---

1. 添加交通连接

2. 删除交通连接

3. 列出所有交通连接

4. 返回主菜单

请输入您的选择: *3*

当前所有交通连接列表:

火车: 成都 -> 西安, 出发时间 21:00, 到达时间 01:00, 费用: 400.0, 火车编号: G1009

火车: 成都 -> 深圳, 出发时间 02:00, 到达时间 06:00, 费用: 450.0, 火车编号: G1028

火车: 成都 -> 杭州, 出发时间 17:00, 到达时间 03:00, 费用: 450.0, 火车编号: G1044

火车: 成都 -> 南京, 出发时间 21:00, 到达时间 07:00, 费用: 450.0, 火车编号: G1048

火车: 成都 -> 北京, 出发时间 18:30, 到达时间 06:30, 费用: 600.0, 火车编号: G1050

飞机: 成都 -> 上海, 出发时间 10:00, 到达时间 12:30, 费用: 350.0

飞机: 成都 -> 南京, 出发时间 09:00, 到达时间 11:00, 费用: 450.0

火车: 上海 -> 徐州, 出发时间 16:00, 到达时间 18:30, 费用: 200.0, 火车编号: G1004

火车: 上海 -> 南京, 出发时间 19:00, 到达时间 22:00, 费用: 200.0, 火车编号: G1017

火车: 上海 -> 杭州, 出发时间 09:00, 到达时间 10:30, 费用: 150.0, 火车编号: G1019

火车: 上海 -> 广州, 出发时间 07:00, 到达时间 15:00, 费用: 400.0, 火车编号: G1033

火车: 上海 -> 西安, 出发时间 20:00, 到达时间 10:00, 费用: 600.0, 火车编号: G1042

火车: 上海 -> 重庆, 出发时间 19:00, 到达时间 09:00, 费用: 550.0, 火车编号: G1046

火车: 上海 -> 西安, 出发时间 07:30, 到达时间 19:30, 费用: 600.0, 火车编号: G1051

飞机: 上海 -> 广州, 出发时间 12:00, 到达时间 14:00, 费用: 350.0

飞机: 上海 -> 北京, 出发时间 22:00, 到达时间 00:00, 费用: 300.0

火车: 广州 -> 杭州, 出发时间 20:00, 到达时间 04:00, 费用: 400.0, 火车编号: G1021

火车: 广州 -> 深圳, 出发时间 09:00, 到达时间 10:30, 费用: 200.0, 火车编号: G1024

火车: 广州 -> 上海, 出发时间 16:00, 到达时间 00:00, 费用: 400.0, 火车编号: G1034

火车: 广州 -> 北京, 出发时间 20:00, 到达时间 08:00, 费用: 700.0, 火车编号: G1036

火车: 广州 -> 南京, 出发时间 21:00, 到达时间 09:00, 费用: 500.0, 火车编号: G1056

飞机: 广州 -> 深圳, 出发时间 15:00, 到达时间 16:00, 费用: 200.0

飞机: 广州 -> 天津, 出发时间 01:00, 到达时间 03:30, 费用: 600.0

火车: 徐州 -> 上海, 出发时间 13:00, 到达时间 15:30, 费用: 200.0, 火车编号: G1003

火车: 徐州 -> 天津, 出发时间 19:00, 到达时间 21:30, 费用: 150.0, 火车编号: G1005

火车: 天津 -> 徐州, 出发时间 10:00, 到达时间 12:30, 费用: 150.0, 火车编号: G1002

火车: 天津 -> 北京, 出发时间 22:00, 到达时间 23:30, 费用: 100.0, 火车编号: G1006

火车: 天津 -> 杭州, 出发时间 17:30, 到达时间 03:30, 费用: 500.0, 火车编号: G1054

飞机: 天津 -> 广州, 出发时间 19:00, 到达时间 21:30, 费用: 600.0

火车: 西安 -> 成都, 出发时间 16:00, 到达时间 20:00, 费用: 400.0, 火车编号: G1008

火车: 西安 -> 北京, 出发时间 02:00, 到达时间 10:00, 费用: 500.0, 火车编号: G1010

火车: 西安 -> 南京, 出发时间 07:00, 到达时间 17:00, 费用: 350.0, 火车编号: G1039

火车: 西安 -> 上海, 出发时间 05:00, 到达时间 19:00, 费用: 600.0, 火车编号: G1041

火车: 西安 -> 上海, 出发时间 20:30, 到达时间 08:30, 费用: 600.0, 火车编号: G1052

火车: 西安 -> 重庆, 出发时间 23:00, 到达时间 11:00, 费用: 500.0, 火车编号: G1058

飞机：西安 -> 北京，出发时间 06:00，到达时间 08:00，费用：300.0

火车：重庆 -> 南京，出发时间 17:00，到达时间 21:00，费用：300.0，火车编号：G1013

火车：重庆 -> 深圳，出发时间 16:00，到达时间 20:00，费用：400.0，火车编号：G1026

火车：重庆 -> 上海，出发时间 04:00，到达时间 18:00，费用：550.0，火车编号：G1045

火车：重庆 -> 西安，出发时间 10:00，到达时间 22:00，费用：500.0，火车编号：G1057

飞机：重庆 -> 南京，出发时间 16:00，到达时间 18:30，费用：500.0

飞机：重庆 -> 深圳，出发时间 06:00，到达时间 08:00，费用：400.0

火车：杭州 -> 南京，出发时间 08:30，到达时间 11:30，费用：250.0，火车编号：G1011

火车：杭州 -> 广州，出发时间 11:00，到达时间 19:00，费用：400.0，火车编号：G1020

火车：杭州 -> 上海，出发时间 05:00，到达时间 06:30，费用：150.0，火车编号：G1022

火车：杭州 -> 深圳，出发时间 14:00，到达时间 20:00，费用：500.0，火车编号：G1030

火车：杭州 -> 成都，出发时间 06:00，到达时间 16:00，费用：450.0，火车编号：G1043

火车：杭州 -> 天津，出发时间 06:30，到达时间 16:30，费用：500.0，火车编号：G1053

飞机：杭州 -> 西安，出发时间 13:00，到达时间 15:30，费用：400.0

火车：北京 -> 天津，出发时间 08:00，到达时间 09:30，费用：100.0，火车编号：G1001

火车：北京 -> 西安，出发时间 07:00，到达时间 15:00，费用：500.0，火车编号：G1007

火车：北京 -> 南京，出发时间 06:30，到达时间 14:30，费用：450.0，火车编号：G1015

火车：北京 -> 广州，出发时间 05:00，到达时间 19:00，费用：700.0，火车编号：G1035

火车：北京 -> 深圳，出发时间 09:00，到达时间 23:00，费用：800.0，火车编号：G1037

火车：北京 -> 成都，出发时间 05:30，到达时间 17:30，费用：600.0，火车编号：G1049

飞机：北京 -> 上海，出发时间 09:00，到达时间 11:00，费用：300.0

火车：深圳 -> 广州，出发时间 07:00，到达时间 08:30，费用：200.0，火车编号：G1023

火车：深圳 -> 重庆，出发时间 11:00，到达时间 15:00，费用：400.0，火车编号：G1025

火车：深圳 -> 成都，出发时间 21:00，到达时间 01:00，费用：450.0，火车编号：G1027

火车：深圳 -> 杭州，出发时间 07:00，到达时间 13:00，费用：500.0，火车编号：G1029

火车：深圳 -> 南京，出发时间 21:00，到达时间 05:00，费用：600.0，火车编号：G1031

火车：深圳 -> 北京，出发时间 00:00，到达时间 14:00，费用：800.0，火车编号：G1038

飞机：深圳 -> 重庆，出发时间 17:00，到达时间 19:00，费用：400.0

飞机：深圳 -> 广州，出发时间 04:00，到达时间 05:00，费用：200.0

火车：南京 -> 重庆，出发时间 12:00，到达时间 16:00，费用：300.0，火车编号：G1012

火车: 南京 -> 杭州, 出发时间 22:00, 到达时间 01:00, 费用: 250.0, 火车编号: G1014

火车: 南京 -> 上海, 出发时间 15:30, 到达时间 18:30, 费用: 200.0, 火车编号: G1016

火车: 南京 -> 北京, 出发时间 23:00, 到达时间 07:00, 费用: 450.0, 火车编号: G1018

火车: 南京 -> 深圳, 出发时间 06:00, 到达时间 12:00, 费用: 600.0, 火车编号: G1032

火车: 南京 -> 西安, 出发时间 18:00, 到达时间 04:00, 费用: 350.0, 火车编号: G1040

火车: 南京 -> 成都, 出发时间 10:00, 到达时间 20:00, 费用: 450.0, 火车编号: G1047

火车: 南京 -> 广州, 出发时间 08:00, 到达时间 20:00, 费用: 500.0, 火车编号: G1055

飞机: 南京 -> 成都, 出发时间 12:30, 到达时间 14:30, 费用: 450.0

（其他功能略去，可进入程序尝试）

5、总结：

（1）时间复杂度（Dijkstra 算法）：

O(N log N + E)，其中 N 为城市数量，E 为交通连接数量。

（2）空间复杂度：

①图结构存储：O(N + E)

②PriorityQueue 的使用和路径记录：O(N)

（3）完成本题对我来说是一个比较大的挑战。本题的其中一个难点在于设计系统功能，需要理清每个 class 需要包含的元素、功能，并选择相应数据结构，逐步实现并完善界面，确保交互正常。在完成过程中，许多问题会在调试的过程中出现，如跨日行程的负时间异常，设置 trainID 以避免同一行程选取时发生冲突等，这让我在编写程序的同时能够发现之前没有考虑到的地方并对其进行缝补、修葺，加深了我对该问题不同情况分析的理解。每个 class 的设计思路大概为：先设置 enum 类来确定交通方式，再设置 Transport 类确定该系统所需的行程信息。在 class Graph 中，我采用 Hashmap 来存储邻接表，在 map 中以城市作为结点，Transport（包含时间、费用、类型、trainID 等要素）为边来构建，实现添加、删除功能。在 class Path 中，我将交通工具、时间、费用放在一起，以便比较得出最优路径。之后利用 Path 生成 PriorityQueue，使用 Dijkstra 算法比较得出最优路径。

（4）通过本次实验，我深入理解了图论在实际问题中的应用，尤其是Dijkstra算法在路径优化中的重要性。交互系统的设计与实现不仅增强了我的java编程能力，还提升了我在算法优化和数据结构选择方面的思考能力，用户交互界面的设计也让我认识到用户体验的重要性，完成本题对我的编程思维有较大锻炼。