**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()** （N为物品总数）

（2）空间复杂度：

**O(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）

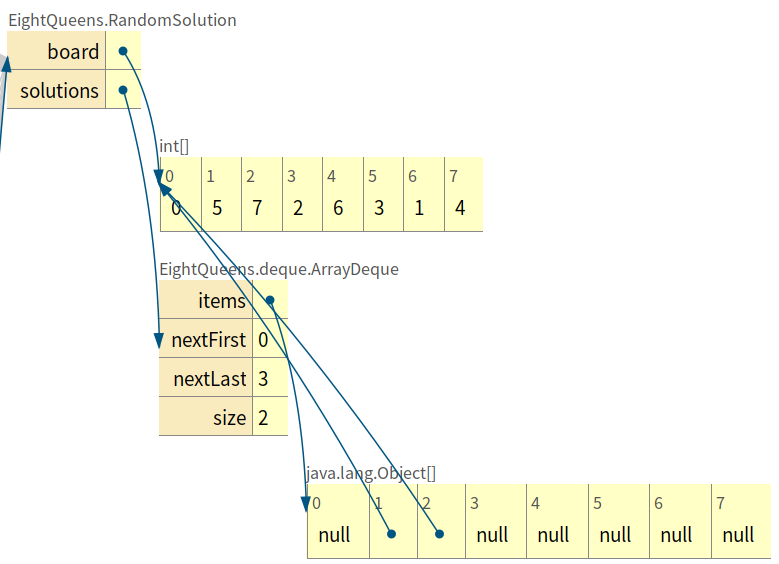
（2）空间复杂度：O(N) （N = SIZE）

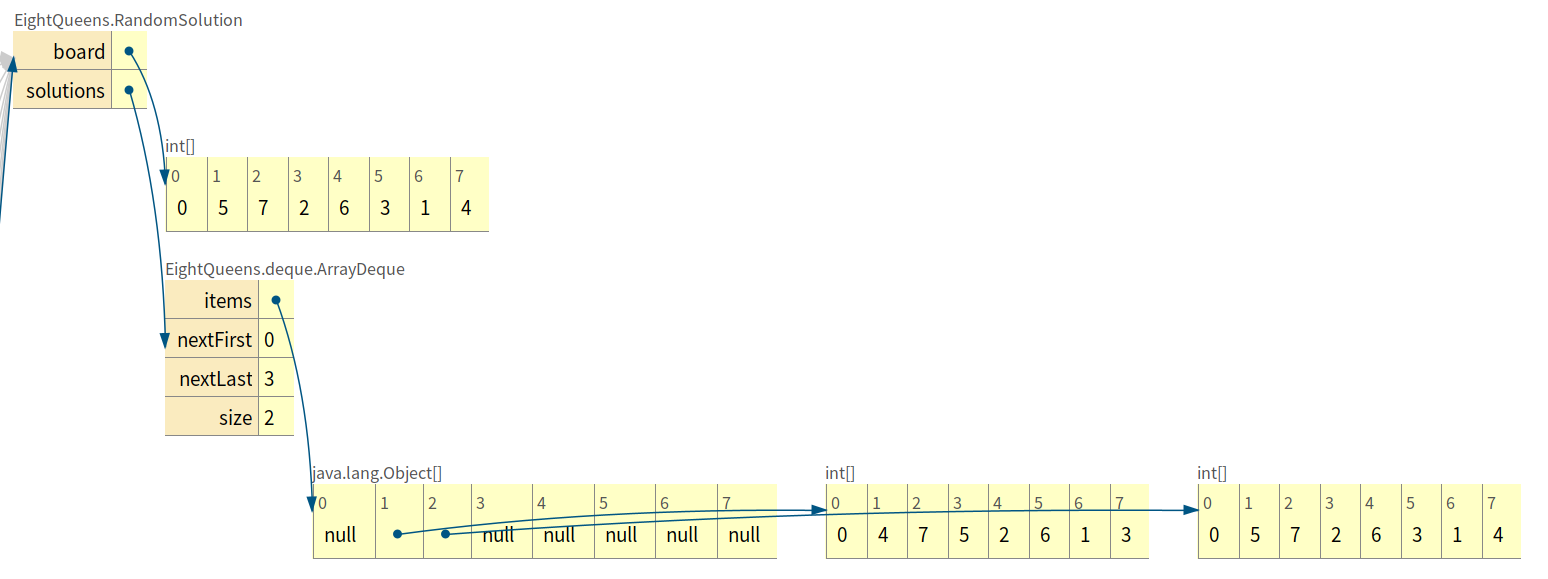
（3）该题完成心得：

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（N²）

（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。