**计算智能 作业1**

2021年9月23日

**要求：**

1. 在文档中说明解题思路、方法实现、求解结果等，必要时需要对结果进行分析和讨论。
2. 将源代码作为附录粘贴于作业文档末尾，并同时作为提交的附件与作业文档一起放在压缩包内备查。
3. 独立完成，严禁抄袭。

**内容：**

1. 编写程序求解下列问题：

小孩分油问题：两个小孩去打油，一人带了一个一斤的空瓶，另一个带了一个七两、一个三两的空瓶。原计划各打一斤油，可是由于所带的钱不够，只好两人合打了一斤油，在回家的路上，两人想平分这一斤油，可是又没有其它工具。试仅用三个瓶子(一斤、七两、三两)精确地分出两个半斤油来。

说明：

1) 一斤 = 10两；

2) 编程语言不限（需提交源代码和运行结果截图）。

3) 截止时间：**10月8日 0:00**。

解题思路：

在这里，由于3+7<=10，这道题我们可以把10两的瓶子视为例2.1中的大桶，来简化问题，这时候，我们同样有8个状态。



|  |  |  |
| --- | --- | --- |
| **规则号** | **规则** | **解释** |
| 1 | (E,S) and E<7 → (7,S) | 7两瓶不满时装满 |
| 2 | (E,S) and S<3 → (E,3) | 3两瓶不满时装满 |
| 3 | (E,S) and E>0 → (0,S) | 7两瓶不空时倒空 |
| 4 | (E,S) and S>0 → (E,0) | 3两瓶不空时倒空 |
| 5 | (E,S) and E>0 and E+S≤3 → (0,E+S) | 7两瓶中油全倒入3两瓶 |
| 6 | (E,S) and S>0 and E+S≤7 → (E+S,0) | 3两瓶中油全倒入7两瓶 |
| 7 | (E,S) and S<3 and E+S≥3 → (E+S-3,3) | 用7两瓶油装满3两瓶子 |
| 8 | (E,S) and E<7 and E+S≥7 → (7,E+S-7) | 用3两瓶油装满7两瓶子 |

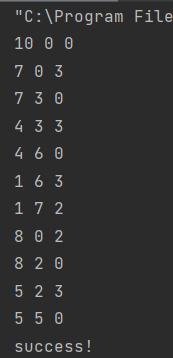
在此基础上，我们可以得到深度优先搜索和广度优先搜索的代码：

附件solution2和solution3.

// 深度优先搜索

import java.util.HashSet;  
import java.util.LinkedList;  
import java.util.Set;  
  
class State {  
 int E;  
 int S;  
  
 public State(int e, int s) {  
 E = e;  
 S = s;  
 }  
}public class Solution2 {  
 static LinkedList<State> *stack* = new LinkedList<>();  
 static Set<State> *visited* = new HashSet<>();  
  
 private static boolean isVisited(State s) {  
 for (State cur : *visited*) {  
 if (cur.E == s.E && cur.S == s.S) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 private static void move(State s) {  
 if (!*isVisited*(s)) {  
 *visited*.add(s);  
 *stack*.push(s);  
 }  
 }  
  
 public static void main(String[] args) {  
 int R = 10, E = 0, S = 0;  
 int fE = 5, fS = 0;  
 *stack*.push(new State(E, S));  
 while (!*stack*.isEmpty()) {  
 State cur = *stack*.pop();  
 E = cur.E;  
 S = cur.S;  
 System.*out*.println(R - E - S + " " + E + " " + S);  
 if (E == fE && S == fS) {  
 System.*out*.println("success!");  
 break;  
 }  
 if (E < 7) *move*(new State(7, S));  
 if (S < 3) *move*(new State(E, 3));  
 if (E > 0) *move*(new State(0, S));  
 if (S > 0) *move*(new State(E, 0));  
 if (E > 0 && E + S <= 3) *move*(new State(0, E + S));  
 if (S > 0 && E + S <= 7) *move*(new State(E + S, 0));  
 if (S < 3 && E + S >= 3) *move*(new State(E + S - 3, 3));  
 if (E < 7 && E + S >= 7) *move*(new State(7, E + S - 7));  
 }  
 }  
}

深度优先搜索搜索路径：



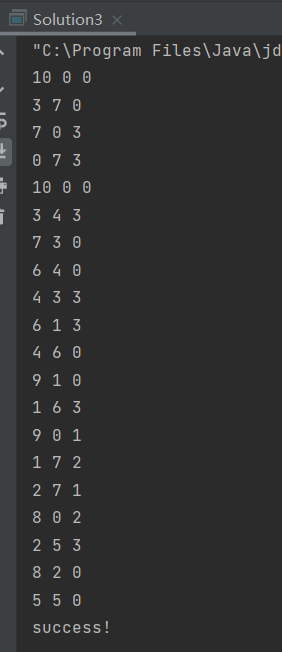
虽然深度优先搜索能找到路径，但是不一定是最短路径，而采用广度优先搜索的方法来解决问题的好处就是只要问题有解，广度优先搜索一定能在优先步骤内求解并且路径最短。

我们来观察广度优先搜索的结果：

// 广度优先搜索

import java.util.HashSet;  
import java.util.LinkedList;  
import java.util.Set;  
  
  
public class Solution3 {  
 static LinkedList<State> *queue* = new LinkedList<>();  
 static Set<State> *visited* = new HashSet<>();  
 private static boolean isVisited(State s) {  
 for(State iterator : *visited*) {  
 if (iterator.E == s.E && iterator.S == s.S) {  
 return true;  
 }  
 }  
 return false;  
 }  
 private static void move(State s) {  
 if (!*isVisited*(s)) {  
 *visited*.add(s);  
 *queue*.addLast(s);  
 }  
 }  
 public static void main(String[] args) {  
 int R = 10, E = 0, S = 0;  
 int fE = 5, fS = 0;  
 *queue*.addLast(new State(E, S));  
 while (!*queue*.isEmpty()) {  
 State cur = *queue*.removeFirst();  
 E = cur.E;  
 S = cur.S;  
 System.*out*.println(R - E - S + " " + E + " " + S);  
 if (E == fE && S == fS) {  
 System.*out*.println("success!");  
 break;  
 }  
 if (E < 7) *move*(new State(7, S));  
 if (S < 3) *move*(new State(E, 3));  
 if (E > 0) *move*(new State(0, S));  
 if (S > 0) *move*(new State(E, 0));  
 if (E > 0 && E + S <= 3) *move*(new State(0, E + S));  
 if (S > 0 && E + S <= 7) *move*(new State(E + S, 0));  
 if (S < 3 && E + S >= 3) *move*(new State(E + S - 3, 3));  
 if (E < 7 && E + S >= 7) *move*(new State(7, E + S - 7));  
 }  
  
 }  
}

结果：



我们看到，广度优先搜索虽然找了最优解，但是搜索过程明显较大。但在本问题中，在问题不复杂的情况下，广度优先搜索能找到最好的结果。我们来看下最好的结果，去掉搜索过程，只看最优路线，代码见solution。

算法表述：

1、初始状态S(10, 0, 0)加入队列

2、判断队列是否为空，若是则任务搜索失败，否则转到3

3、取队首状态，广度优先搜索队首状态之后的每一中状态

4、若状态为（5，5，0），返回正确结果，搜索结束

5、若状态已经搜索过，直接跳过，否则到6

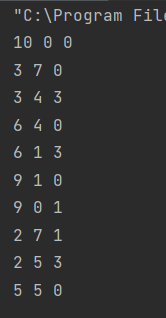
6、加入队列末尾，重复3-6过程。

代码：

// 记录搜索结果的广度优先搜索

import java.util.Arrays;  
import java.util.LinkedList;  
  
class Path {  
 private int[] Node;  
 private int pre;  
  
 public Path(int[] node, int pre) {  
 Node = node;  
 this.pre = pre;  
 }  
  
 public int[] getNode() {  
 return Node;  
 }  
  
 public int getPre() {  
 return pre;  
 }  
}  
  
public class Solution {  
 private static final int[] *cap* = {10, 7, 3};  
 private static Path *start* = new Path(new int[]{10, 0, 0}, -1);  
 private static Path[] *que* = new Path[1000];  
 private static boolean[][][] *tag* = new boolean[11][8][4];  
  
 public static void main(String[] args) {  
 Path end = *bfs*(*start*, *que*);  
 Path temp;  
 LinkedList<String> output = new LinkedList<>();  
 if (end != null) {  
 temp = end;  
 while (temp.getPre() != -1) {  
 output.add(temp.getNode()[0] + " " + temp.getNode()[1] + " " + temp.getNode()[2]);  
 temp = *que*[temp.getPre()];  
 }  
 output.add(temp.getNode()[0] + " " + temp.getNode()[1] + " " + temp.getNode()[2]);  
 }  
 while (output.size() != 0) {  
 System.*out*.println(output.removeLast());  
 }  
 }  
  
 public static void pour(int i, int j, int[] node) {  
 if (node[i] >= (*cap*[j] - node[j])) {  
 node[i] -= *cap*[j] - node[j];  
 node[j] = *cap*[j];  
 } else {  
 node[j] += node[i];  
 node[i] = 0;  
 }  
 }  
  
 public static Path bfs(Path start, Path[] que) {  
 int front = 0;  
 int tail = 0;  
 que[tail] = start;  
 tail += 1;  
 int[] cur = start.getNode();  
 *tag*[cur[0]][cur[1]][cur[2]] = true;  
  
 while (front <= tail) {  
 Path path = que[front];  
 for (int i = 0; i < 3; i++) {  
 if (path.getNode()[i] != 0) {  
 for (int j = 0; j < 3; j++) {  
 int[] temp = Arrays.*copyOf*(path.getNode(), 3);  
 *pour*(i, (i + j) % 3, temp);  
 if (temp[0] == 5 && temp[1] == 5) {  
 que[tail] = new Path(temp, front);  
 return que[tail];  
 }  
 if (*tag*[temp[0]][temp[1]][temp[2]]) {  
 continue;  
 }  
 *tag*[temp[0]][temp[1]][temp[2]] = true;  
 que[tail] = new Path(temp, front);  
 tail += 1;  
  
 }  
  
 }  
 }  
 front += 1;  
 }  
 return null;  
 }  
}

运行结果



可以看到，最优结果只用了9步，而深度优先搜索得到的结果需要10步，但是，为了搜索到这9步最优结果，广度优先搜索搜索路径为19步，在实际中，我们可以根据需要进行取舍和权衡。