

学生实验报告



课程名称\_\_\_数据结构与算法 \_

题目名称\_\_\_过河\_\_

小组成员\_\_ 32001272-徐彬涵 \_\_\_

（学号-姓名）

\_ 32001269-徐若萱 \_

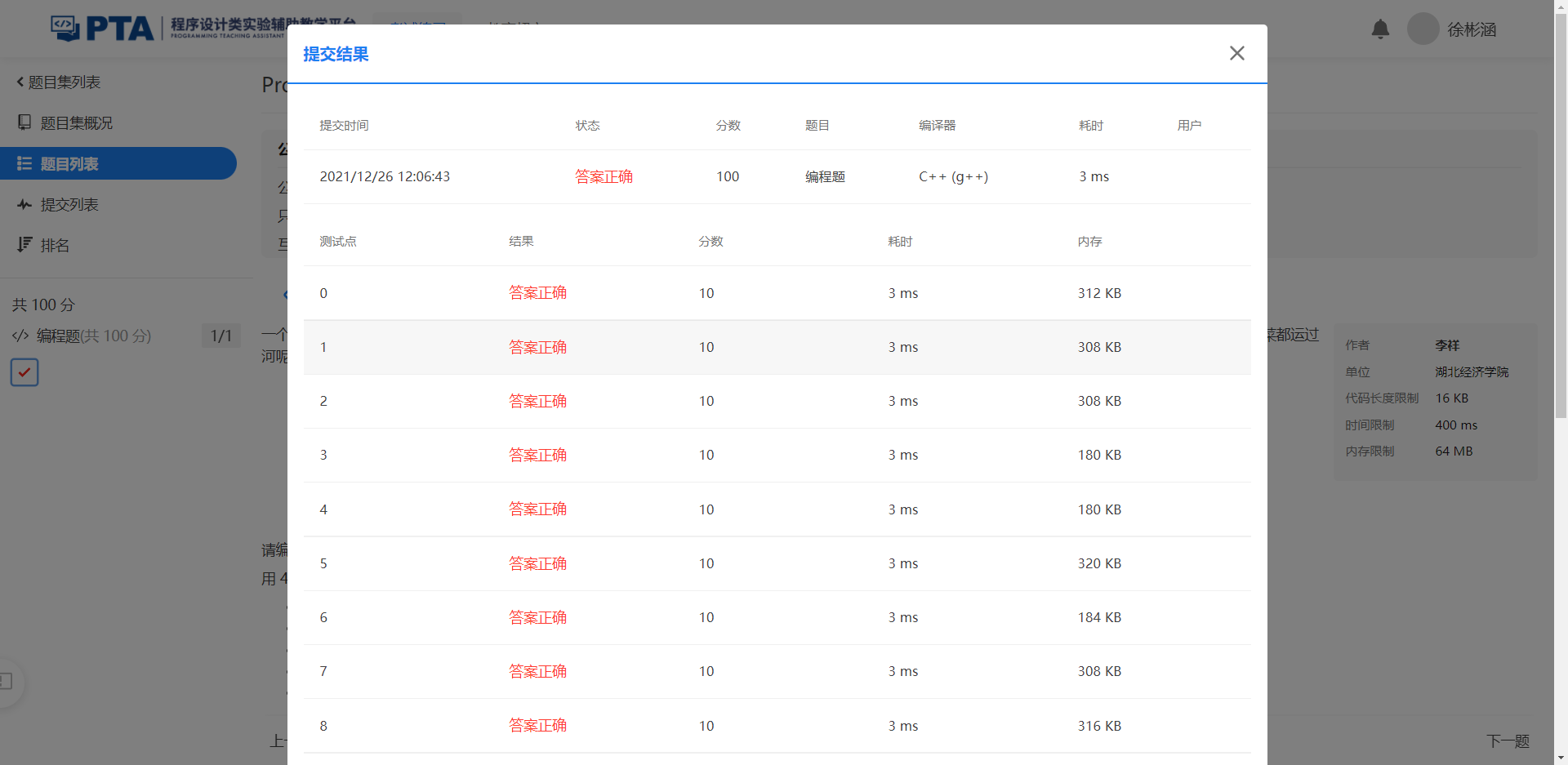
\_ (互评版本请不要填写本行内容) \_

任课教师\_\_ \_

2021 年12月 25 日

# 零.完成情况

**PINTIA评测成绩：**



**成员分工及贡献情况：**

5分 徐彬涵：编写代码测试样例

5分 徐若萱：编写报告

# 一. 问题描述

# 题目：过河

一个人要将一匹狼、一只羊、一筐菜运到河对岸。但是他的船太小了，一次只能带一样。当他不在时，狼要吃羊、羊要吃菜。怎样才能安全地把它们都运过河呢？

|  |
| --- |
|  |

请编写程序，找出过河的全部方案。

用 4 个字母分别表示人、狼、羊、菜的状态：

* 用 M 表示人在，用 . 表示人不在
* 用 W 表示狼在，用 . 表示狼不在
* 用 G 表示羊在，用 . 表示羊不在
* 用 C 表示菜在，用 . 表示菜不在
* 用 -> 表示船从左岸去右岸，用 <- 表示船从右岸去左岸。

举例说明：

若人狼羊菜全在左岸，则状态表示为

MWGC -> ....

若人将羊运到右岸，左岸只剩狼和菜，则状态表示为

.W.C <- M.G.

#### 输入格式

初始状态  
终止状态

#### 输出格式

若有解，则输出所有解法。  
每一个解法由若干行组成，每行表示一个状态。  
不同的解法之间空一行。  
若无解，则输出 None。

**题目保证**所输入的初始状态和终止状态不同，不会发生狼吃羊、羊吃菜的情况，且船只和人在一起。

为使输出的解法排列顺序一致，要求人每次渡河时都按以下顺序进行尝试：

* 不带东西过河
* 带狼过河
* 带羊过河
* 带菜过河

并且要求输出的每个解法中不允许出现两行相同的状态。

#### 输入样例1

MWGC -> ....

.... <- MWGC

#### 输出样例1

MWGC -> ....

.W.C <- M.G.

MW.C -> ..G.

...C <- MWG.

M.GC -> .W..

..G. <- MW.C

M.G. -> .W.C

.... <- MWGC

MWGC -> ....

.W.C <- M.G.

MW.C -> ..G.

.W.. <- M.GC

MWG. -> ...C

..G. <- MW.C

M.G. -> .W.C

.... <- MWGC

# 二.数据结构定义与分析

**分析问题**，根据分析写出数据结构定义

**该问题可以采用回溯遍历的方式对每一种方式进行遍历，只需建立一个结构体来存放人狼羊菜的分布情况，这里我利用四位二进制的方式来表示分布情况，**

**然后用结构体数组（里面含有一个int值表示四位二进制数）来存储可能的情况**

**四位二进制数**

**0表示在左岸**

**1表示在右岸**

**将四位二进制转为十进制即可较方便的存储**

**船的行进方式无需特殊标记，因为船一定需要人来划，所以人在哪边船就在那边**

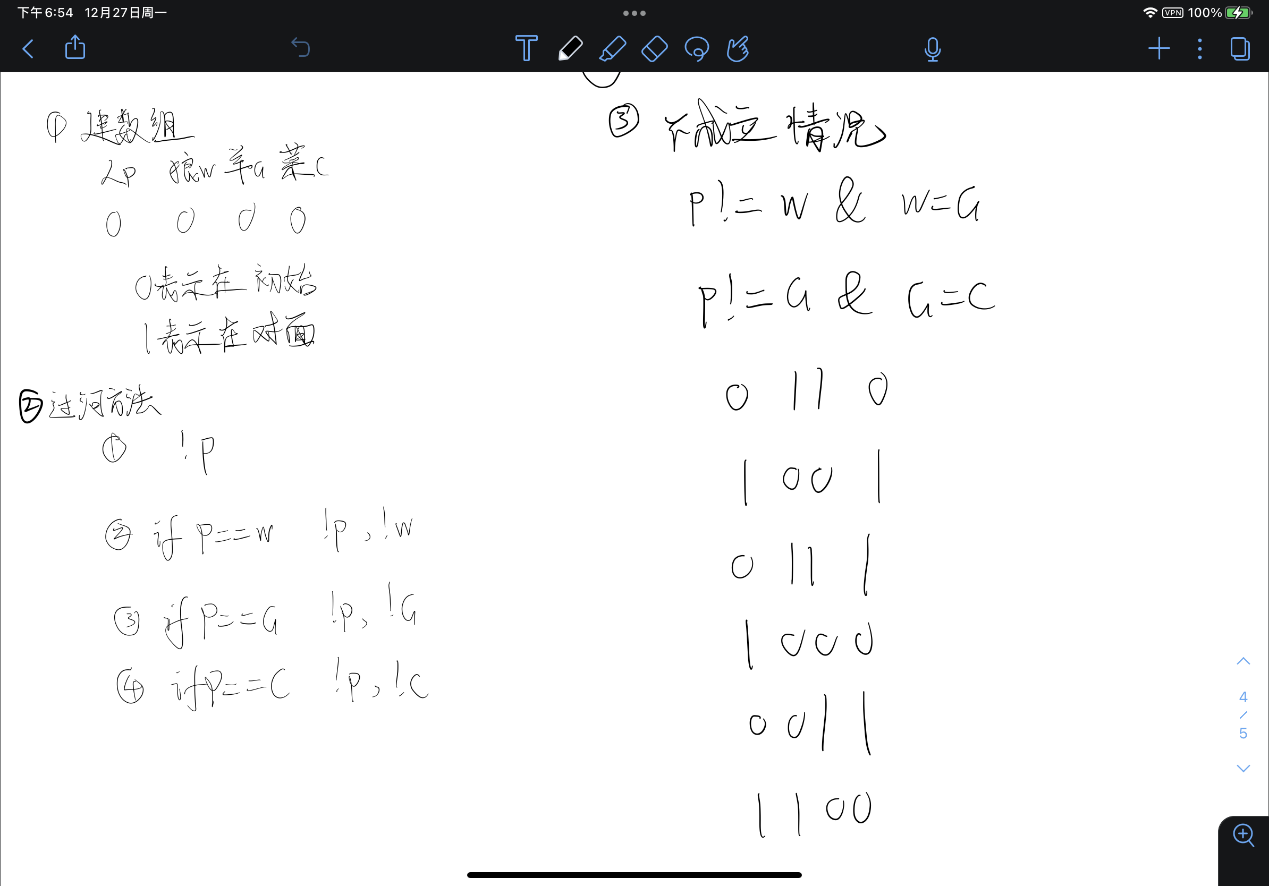
# 三、算法分析

写出算法的主要思想

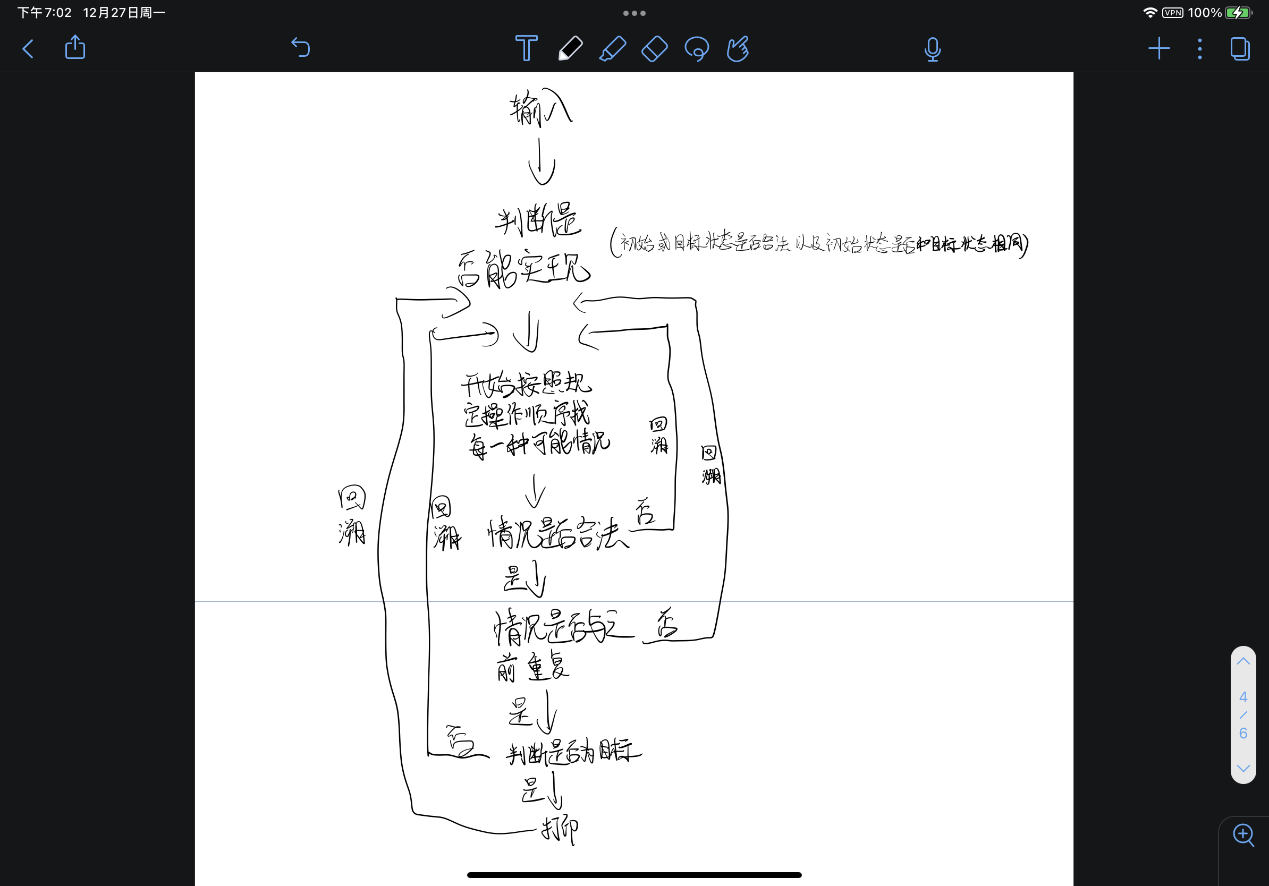
**首先初始化，将输入的字符串转化为数值储存进初始结构体，将目标数值存入target，然后进行递归回溯处理，每次都按照题目要求进行处理，然后分支继续递归，如果不符合要求则回溯，将该情况进行“剪枝”处理。如果target一致则进行打印，最后输出所有结果。**

**无法完成的情况只有初始情况和目标情况一样的，和初始情况不合法，以及目标情况不合法这三种，所以仅需在输入之后进行判断，而不需要在递归时进行判断**

**大致示意图**



**流程图**



# 四. 算法的时间复杂度分析

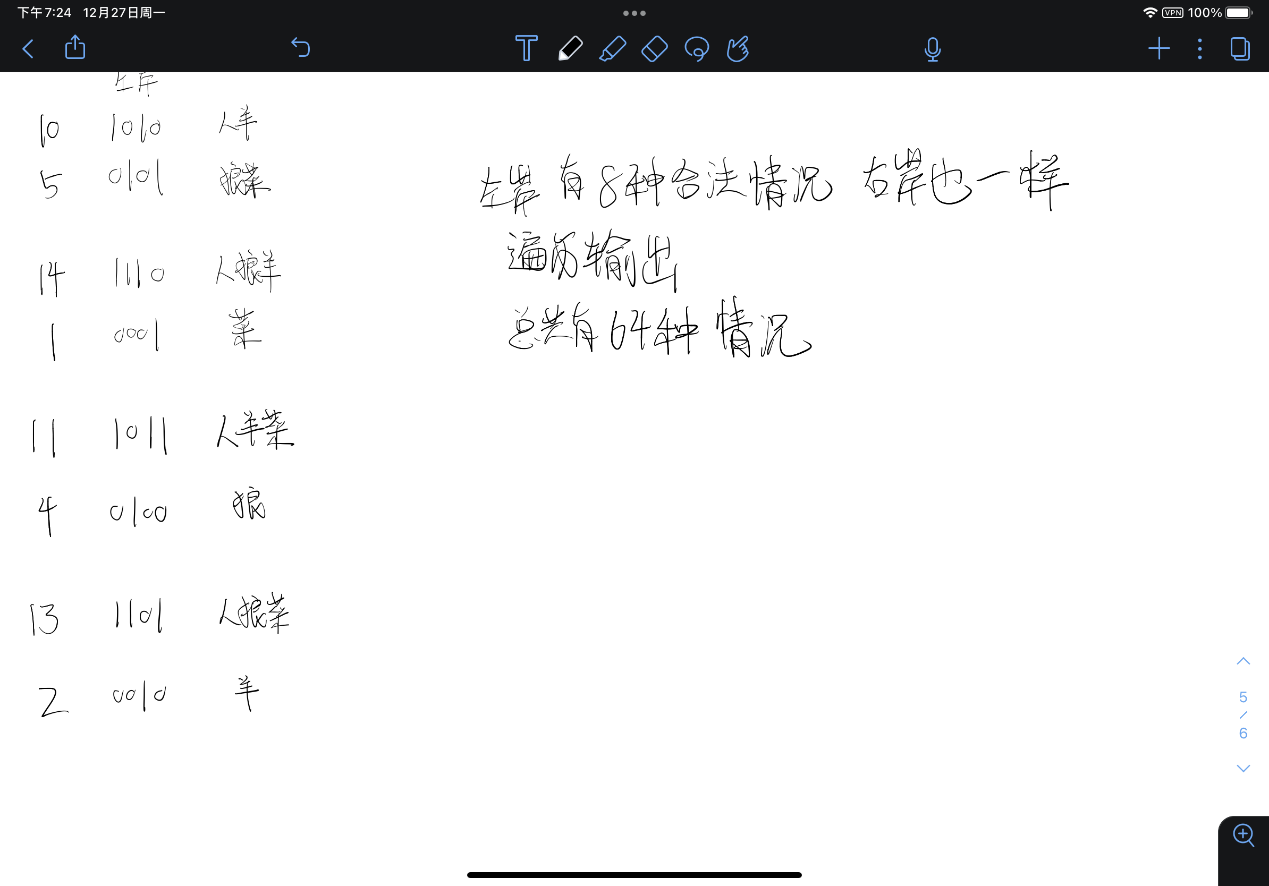
分析算法的时间复杂度以及分析过程

**核心算法是递归，运用了两个函数来判断是否重复和是否合法，判断重复的时间复杂度为O(n)，是否合法的为O(1)，递归回溯算法的最坏情况时间复杂度为O(!n)，总的时间复杂度可以归为O(!n)**

# 五、测试用例的选择原因以及测试结果

写出测试用例，详细阐述每个测试用例的选择原因，测试结果可以截图，（注意截图不要出现姓名、学号等信息）

其实我们可以知道，左边合法的情况只有这几种



测试程序请见

**https://paste.ubuntu.com/p/W6hwzr2FPT/**

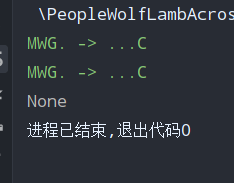
test.cpp文件压缩包里有，双重保险，但我是用CLION写的，所以用devc++打开可能会乱码，所有建议打开上面的网址复制代码

下面再给出几组不成立案例

初始情况和目标情况相同

MWG. -> ...C

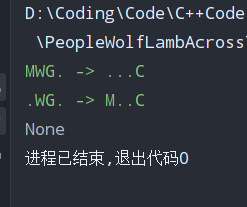
MWG. -> ...C



目标情况为非法情况

MWG. -> ...C

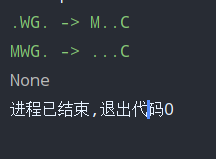
.WG. -> M..C



初始情况为非法情况

.WG. -> M..C

MWG. -> ...C



**六、存在问题及可能原因**

算法可能存在的问题，若有，请写出解决原因与方法。

暂未发现问题

# 七、附源代码以及代码相应注释

我是用CLION写的，所以用devc++打开可能会乱码，所有建议打开下面的网址复制代码

**https://paste.ubuntu.com/p/VZT7Y2CfrQ/**

源代码

1. #include<stdio.h>
2. **bool** orz = **false**;//用于实现在每一种情况输出后隔一行再输出下一种情况的标志
3. **struct** state{
4. **int** sit;
5. };
6. //使用二进制分布来判断人狼羊菜
7. **void** Init(**struct** state \*start, **int** \*target){
8. **char** x;
9. **int** sum = 0;
10. scanf("%c",&x);
11. **if**(x == 'M') sum += 1 << 3;
12. scanf("%c",&x);
13. **if**(x == 'W') sum += 1 << 2;
14. scanf("%c",&x);
15. **if**(x == 'G') sum += 1 << 1;
16. scanf("%c",&x);
17. **if**(x == 'C') sum += 1;
18. **while** (x != '\n')
19. x = getchar();
20. start->sit = sum;
21. \*target = 0;
22. scanf("%c",&x);
23. **if**(x == 'M') \*target += 1 << 3;
24. scanf("%c",&x);
25. **if**(x == 'W') \*target += 1 << 2;
26. scanf("%c",&x);
27. **if**(x == 'G') \*target += 1 << 1;
28. scanf("%c",&x);
29. **if**(x == 'C') \*target += 1;
30. **while** (x != '\n')
31. x = getchar();
32. }
33. //判断是否是非法情况
34. **bool** IsWrong(**int** test){
35. **int** p, w, g, c;
36. p = (test >> 3) % 2;
37. w = (test >> 2) % 2;
38. g = (test >> 1) % 2;
39. c = test % 2;
40. **if**((p != w && w == g) || (p != g && g == c))
41. **return** **true**;
42. **else** **return** **false**;
43. }
44. //判断是否重复
45. **bool** IsRepeat(**int** test,**struct** state \*start, **int** flag){
46. **for**(**int** i = 0; i < flag; ++i){
47. **if**(start[i].sit == test)
48. **return** **true**;
49. }
50. **return** **false**;
51. }
52. //打印结果
53. **void** print(**struct** state \*start, **int** flag){
54. **if**(orz) printf("\n");
55. orz = **true**;
56. **for**(**int** i = 0; i < flag; ++i){
57. **int** p, w, g, c;
58. **char** out[13];
59. p = (start[i].sit >> 3) % 2;
60. w = (start[i].sit >> 2) % 2;
61. g = (start[i].sit >> 1) % 2;
62. c = start[i].sit % 2;
63. //        printf("%d%d%d%d\n", p, w, g, c);
64. out[4] = out[7] = ' ';
65. out[12] = '\0';
66. **if**(p){
67. out[0] = 'M';
68. out[8] = '.';
69. out[5] = '-';
70. out[6] = '>';
71. }
72. **else**{
73. out[0] = '.';
74. out[8] = 'M';
75. out[5] = '<';
76. out[6] = '-';
77. }
78. **if**(w){
79. out[1] = 'W';
80. out[9] = '.';
81. }
82. **else**{
83. out[1] = '.';
84. out[9] = 'W';
85. }
86. **if**(g){
87. out[2] = 'G';
88. out[10] = '.';
89. }
90. **else**{
91. out[2] = '.';
92. out[10] = 'G';
93. }
94. **if**(c){
95. out[3] = 'C';
96. out[11] = '.';
97. }
98. **else**{
99. out[3] = '.';
100. out[11] = 'C';
101. }
102. printf("%s\n",out);
103. }
104. }
105. //flag即表示寻找第flag步操作
106. **void** Move(**int** target,**struct** state \*start, **int** flag){
107. //    printf("%d\n", start[0].sit);
109. **if**(start[flag-1].sit == target){
110. print(start, flag);
111. }
112. **int** p, w, g, c;
113. p = (start[flag-1].sit >> 3) % 2;
114. w = (start[flag-1].sit >> 2) % 2;
115. g = (start[flag-1].sit >> 1) % 2;
116. c = start[flag-1].sit % 2;
118. **int** sum;
119. //    直接过河(不带任何东西)
120. sum = (!p << 3) + (w << 2) + (g << 1) + c;
121. **if**(!IsRepeat(sum, start, flag) && !IsWrong(sum)){
122. start[flag].sit = sum;
123. Move(target, start, flag+1);
124. }
125. //    带狼过河
126. **if**(p == w){
127. sum = (!p << 3) + (!w << 2) + (g << 1) + c;
128. **if**(!IsRepeat(sum, start, flag) && !IsWrong(sum)){
129. start[flag].sit = sum;
130. Move(target, start, flag+1);
131. }
132. }
133. //    带羊过河
134. **if**(p == g){
135. sum = (!p << 3) + (w << 2) + (!g << 1) + c;
136. **if**(!IsRepeat(sum, start, flag) && !IsWrong(sum)){
137. start[flag].sit = sum;
138. Move(target, start, flag+1);
139. }
140. }
141. //    带菜过河
142. **if**(p == c){
143. sum = (!p << 3) + (w << 2) + (g << 1) + !c;
144. **if**(!IsRepeat(sum, start, flag) && !IsWrong(sum)){
145. start[flag].sit = sum;
146. Move(target, start, flag+1);
147. }
148. }
149. }
150. **int** main() {
151. **struct** state state[1000];
152. **int** target;
153. Init(&state[0], &target);
154. **if**(state[0].sit == target || IsWrong(target) || IsWrong(state[0].sit))
155. printf("None");
156. **else**
157. Move(target,state, 1);
158. //    printf("%d\n",state[1].sit);
159. //    printf("%d\n",target);
160. **return** 0;
161. }



# 八、实验创新点或合理化建议

本次实验的创新之处以及是否可以有更好的建议使得题目更加有实用意义等。

创新的利用了计算机的回溯算法很好的解决问题

题目可以加一个输出最短步骤的要求来增加实用意义