

《数据结构》上机实验报告

第 5 次上机

学号：

姓名：

学院： 信息科学学院

专业： 计算机科学

教师：

日期： 2018.10.12

1. 实验要求

1.上机之前应做好充分准备，认真思考所需的上机题目，提高上机效率。

2.独立上机输入和调试自己所编的程序，切忌抄袭、拷贝他人程序。

3.上机结束后，整理出实验报告。书写报告时，重点放在实验的方法、思路以及总结反思上，以达到巩固课堂学习、提高动手能力的目的。

1. 实验内容

八皇后问题：

设在初始状态下在国际象棋棋盘上没有任何棋子(皇后)。然后顺序在第1行，第2行，…。第8行上布放棋子。在每一行中有8个可选择位置，但在任一时刻，棋盘的合法布局都必须满足3个限制条件，即任何两个棋子不得放在棋盘上的同一行、或者同一列、或者同一斜线上。

设计要求：

1、试用递归的方法编写算法，求解并输出此问题的所有合法布局。

2、试用非递归的方法编写算法，求解并输出此问题的所有合法布局。

1. 实验步骤（写出问题分析或者算法思路）

递归思路：使用深度优先搜索一层一层地检索，对每一层来说从左往右遍历出当前情况下允许放置的位置。对于这一层的每个位置都采用先假设放置一个，然后更新位置可用状态后再看下一层，无论这一种放置方法是否成功，最后都会回到同一层，还原位置可用状态后再看同一层的下一个可选位置。详见函数dfs\_recursion。

关于位置是否可用的状态记录，采用ppt上的对角线判断法，用两个大小为2\*n的数组记录2n条对角线的使用情况，并且用一个大小为n的数组记录纵轴的使用情况，这样就可以用O(1)时间去查询放置状态和更改放置状态了。

复杂度的估计：因为是每一层每一个的遍历，而且上层的放置状态会影响到下层的放置状态，所以之前遍历得到的成功状态并没有对后面程序有借鉴意义。第一层有n种放置方法，对应地第二层就有(n-2)或(n-3)种放置方法，对应第三层就有(n-3)或(n-4)或(n-5)种放置方法，可以看出耗费的时间是 <n! ，尽管有对角线数组帮助剪枝，但是在n比较大的时候，这种程度的剪枝是微不足道的，所以算法的复杂度是O(n!)的。这是一个非常大的复杂度，因为我在测试n=11的时候就耗费了差不多有1min的时间，而且伴随着n的增加，检索的时间是呈现指数型得增长，在n>=12的时候就太慢了。

非递归思路：

考虑走迷宫的办法，从0层开始走，第一层的每一个点都能走，走了之后会对后面能走的点有约束。使用栈来记录下每个点能到达的每一个新状态，每次都取栈顶元素出来，看这一状态能否继续向下走。详见函数dfs\_not\_recursion。

栈用了stl<stack>库的栈定义。

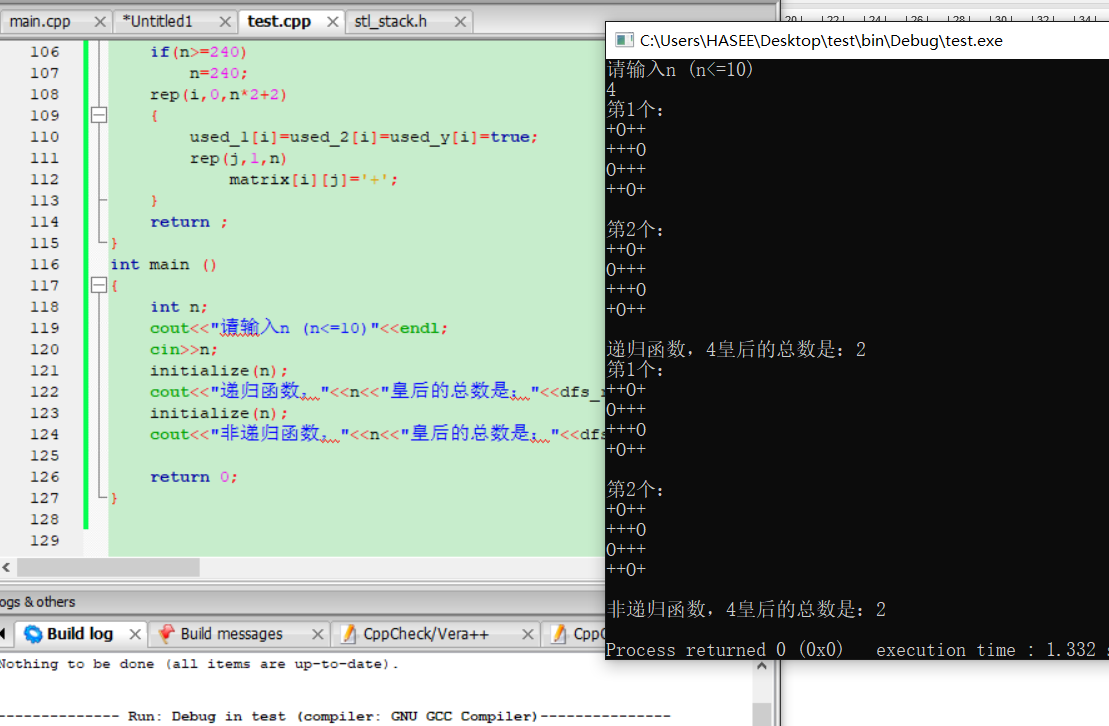
每种状态包括了5个变量，th表示当前的层数，qe数组记录的是第i层走的是哪一列的点（方便输出，如果不要求输出棋盘的话，能删去加快速度），col,md,sd分别表示列、两个对角线的占用情况。注意采用了状态压缩的办法存储。

其实递归的方法也是一种走迷宫的搜索办法，只是搜索的顺序有变化。复杂度还是O(n!)的。

1. 程序清单（源程序代码等）

main.h:

1. #include<iostream>
2. #include <stdio.h>
3. #include <malloc.h>
4. #include <fstream>
5. #include <string>
6. #include <stack>
7. using namespace std;
8. #define rep(a,b,c) for(int a=b;a<=c;a++)
9. #define ll long long
10. int cnt;
11. bool used\_y[250]; //纵轴
12. bool used\_1[250]; //主对称轴
13. bool used\_2[250]; //反对称轴
14. char matrix[250][250];
15. int dfs\_recursion(int th,int tar)//递归
16. {
17. int ans=0;
18. if(th==tar+1)
19. {
20. cout<<"第"<<++cnt<<"个："<<endl;
21. rep(i,1,tar)
22. {
23. rep(j,1,tar)
24. cout<<matrix[i][j];
25. cout<<endl;
26. }
27. cout<<endl;
28. return 1;
29. }
30. else
31. {
32. rep(i,1,tar)
33. {
34. if(used\_y[i] && used\_1[i-th+tar] && used\_2[i+th])
35. {
36. used\_y[i]=used\_1[i-th+tar]=used\_2[i+th]=false;
37. matrix[th][i]='O';
38. ans=ans+dfs\_recursion(th+1,tar);
39. matrix[th][i]='+';
40. used\_y[i]=used\_1[i-th+tar]=used\_2[i+th]=true;
41. }
42. }
43. return ans;
44. }
45. }
46. struct state
47. {
48. int th;
49. ll col,md,sd;
50. int qe[15];
51. };
52. int dfs\_not\_recursion(int tar) //非递归
53. {
54. int ans=0;
55. state start;
56. rep(i,1,tar)
57. start.qe[i]=0;
58. start.col=start.md=start.sd=0LL;
59. start.th=0;
60. stack <state> q;
61. q.push(start);
62. while(!q.empty())
63. {
64. state tmp=q.top();
65. q.pop();
66. if(tmp.th==tar)
67. {
68. ans++;
69. cout<<"第"<<++cnt<<"个："<<endl;
70. rep(i,1,tar)
71. {
72. rep(j,1,tmp.qe[i]-1)
73. cout<<"+";
74. cout<<"O";
75. rep(j,tmp.qe[i]+1,tar)
76. cout<<"+";
77. cout<<endl;
78. }
79. cout<<endl;
80. }
81. else
82. {
83. rep(i,1,tar)
84. {
85. if(! (tmp.col&(1LL<<i) || tmp.md &(1LL<<(i+tmp.th+1)) || tmp.sd&(1LL<<(-i+tar+tmp.th+1))) )
86. {
87. state ne=tmp;
88. ne.col += 1LL<<i;
89. ne.md += (1LL<<(i+tmp.th+1));
90. ne.sd += (1LL<<(-i+tar+tmp.th+1));
91. ne.qe[tmp.th+1]=i;
92. ne.th=tmp.th+1;
93. q.push(ne);
94. }
95. }
96. }
97. }
98. return ans;
99. }
100. void initialize(int n)
101. {
102. cnt=0;
103. if(n>=240)
104. n=240;
105. rep(i,0,n\*2+2)
106. {
107. used\_1[i]=used\_2[i]=used\_y[i]=true;
108. rep(j,1,n)
109. matrix[i][j]='+';
110. }
111. return ;
112. }
113. int main ()
114. {
115. int n;
116. cout<<"请输入n (n<=10)"<<endl;
117. cin>>n;
118. initialize(n);
119. cout<<"递归函数，"<<n<<"皇后的总数是："<<dfs\_recursion(1,n)<<endl;
120. initialize(n);
121. cout<<"非递归函数，"<<n<<"皇后的总数是："<<dfs\_not\_recursion(n)<<endl;
122. return 0;
123. }
124. 运行结果（程序运行时的结果说明或运行截图等）



1. 总结（实验中遇到的问题、取得的经验、感想等）

1 递归的写法更简单。有写错的风险，要考虑更多的问题。

2 n皇后问题的算法的复杂度是O(n!)的，不能进行n>=13的计算。

3 n皇后问题的递归算法是深度优先算法，这种算法一般用于解决迷宫、爬虫等问题，但本质还是一种穷举的办法，有时可以结合动态规划可以化简。但是我不能找到n皇后里高效的化简办法。（如果不要求打印每个结果、只求状态总数的话可能有更多的办法）