

中国科学技术大学计算机学院
《数据结构》报告



实验题目：栈、队列及其应用

学生姓名：王章瀚

学生学号：PB18111697

完成日期：2019 年 11 月 1 日

计算机实验教学中心制

2019 年 09 月

1 实验要求

1.1 概述

假设有一 $N \times N$ 的棋盘和 N 个皇后，请为这 N 个皇后进行布局使得这 N 个皇后互不攻击（即任意两个皇后 不在同一行、同一列、同一对角线上）。

基本要求：

1. 输入 N ，输出 N 个皇后互不攻击的所有布局；
2. 用非递归方法来解决 N -皇后问题，即自己设置栈来处理。

1.2 输入与输出

输入棋盘大小（皇后数） N

输出所有满足题目要求的棋盘。

输入输出样例：

1.3 测试数据

```
Input:
8
Output:
Q#####
#####Q###
#####Q
#####Q###
###Q#####
#####Q#
#Q#####
###Q#####
.....//所有布局方案
92 //总布局数
```

2 设计思路

本实验的基本算法如下图所示：

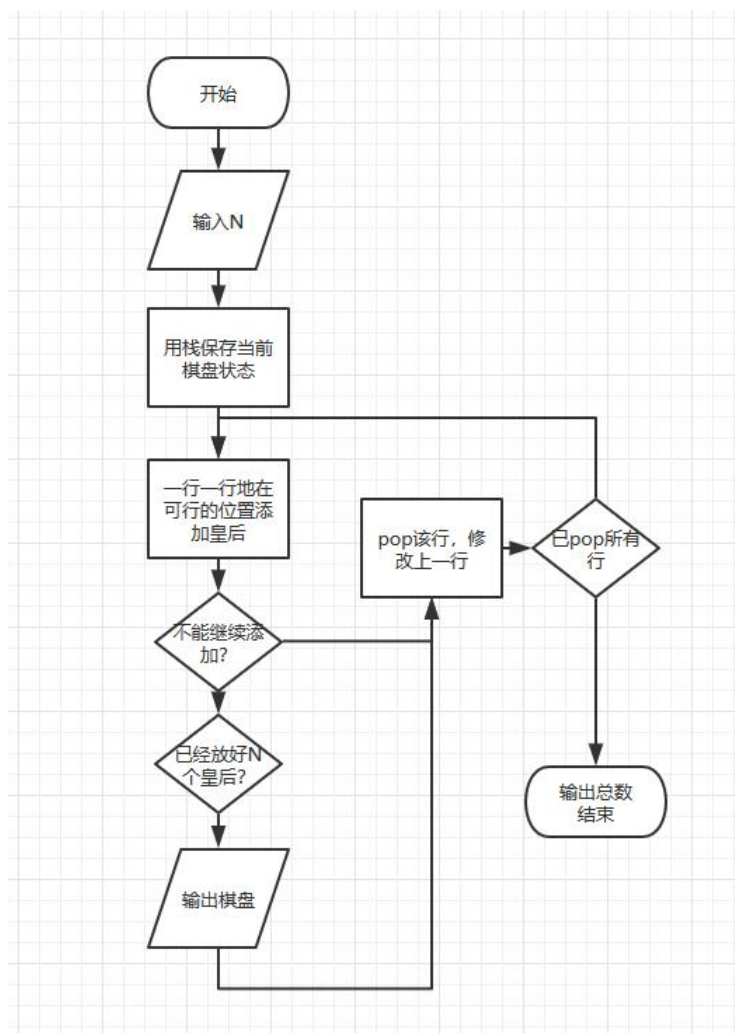


图 1: 基本算法思路

本实验中，使用栈和递归来分别实现任务。

其中，使用栈的方法时，定义了一个栈类型，后面会描述其定义。

使用递归的方法则单纯使用一个线性表来存储当前正在处理的 layout（布局）。

而这两种方法的共同点是存储布局的方式——即使用一个 N 维线性表来存储每行放的皇后的列位置。

3 关键代码讲解

由于没有复杂的函数调用关系，故将函数调用关系图略去。

且此处算法设计思路极其简单，即，逐行遍历 layout 的所有可能情况，如遇冲突，则撤销摆放，向前回溯，如遇成功，则输出，计数器自加。

3.1 栈

3.1.1 数据类型的定义

1). 布局栈的定义

```
1 // Here we use a stack to store the information of
2 // the layout.
3 class layout {
4 private:
5     int cur;
6     int size;
7     int** layout_stack;
8
9 public:
10    layout(int N) {
11        size = 0;
12        cur = -1;
13        this->N = N;
14        layout_stack = new int*[N];
15        for (int i = 0; i < N; i++) {
16            layout_stack[i] = new int[N];
17        }
18    }
19
20    int N;
21    inline int stack_size() {
22        return cur + 1;
23    }
24
25    int* top() {
26        if (cur >= 0)
27            return layout_stack[cur];
28        return nullptr;
```

```

29     }
30
31     void pop() {
32         if (cur >= 0)
33             delete layout_stack[cur--];
34     }
35
36     void push(int* cur_layout) {
37         layout_stack[++cur] = cur_layout;
38     }
39
40     int* clone(int* ori);
41     void printLayout(int* lay);
42     bool is_lastline_ok(int* lay, int row);
43     //use this function to solve the answer.
44     void solve();
45 };
46

```

3.1.2 主要算法

主要算法即将流程图内容实现一遍，过程较为简单，详细说明已在下文注释：

```

1     void solve() {
2         clock_t start, end;
3         start = clock();
4         int counter = 0;
5         int* cur_layout = new int[N];
6         cur_layout[0] = 0;
7         push(cur_layout);
8         while (1) {
9             cur_layout = clone(top());
10            if (top() == nullptr)
11                break;
12            if (is_lastline_ok(cur_layout, stack_size() - 1)) {
13                // no conflict here
14                // whether finished
15                if (stack_size() == N) {
16                    counter++;
17                    cout << counter << endl;
18                    printLayout(cur_layout);
19                }

```

```

20         else {
21             cur_layout[stack_size()] = 0;
22             push(cur_layout);
23             continue;
24         }
25     }
26     // need a recall
27     delete[] cur_layout;
28     cur_layout = top();
29     if (cur_layout == nullptr)
30         break;
31     while (cur_layout[stack_size() - 1] == N - 1) {
32         // tried all circumstance in this row
33         pop();
34         cur_layout = top();
35         if (cur_layout == nullptr)
36             break;
37     }
38     // found a row that the queen can be move rightward.
39     if (cur_layout != nullptr) {
40         top()[stack_size() - 1]++;
41     }
42 }
43 end = clock();
44 cout << "以上共" << counter << "种" << endl;
45 cout << "耗时" << (double)(end - start) / CLOCKS_PER_SEC << "秒" <<
endl;
46
47 }
48

```

3.2 递归

3.2.1 主要算法

主要算法即将流程图内容实现一遍，过程较为简单，详细说明已在下文注释：

```

1 // recursion method
2 int counter = 0;
3 void recursion(int* cur_layout, int row, const int N) {
4     if (row == N) {
5         counter++;

```

```

6      // print the layout.
7      cout << counter << endl;
8      for (int i = 0; i < N; i++) {
9          for (int j = 0; j < N; j++) {
10             if (j == cur_layout[i]) cout << 'Q';
11             else cout << '#';
12         }
13         cout << endl;
14     }
15     cout << endl;
16     return;
17 }
18 for (int i = 0; i < N; i++) {
19     cur_layout[row] = i;
20     bool flag = true;
21     // try all circumstances.
22     for (int j = 0; j < row; j++) {
23         if (cur_layout[j] == cur_layout[row]
24             || abs(cur_layout[j] - cur_layout[row]) == row - j) {
25             flag = false;
26             break;
27         }
28     }
29     // if ok, try next row. Here is the recursion.
30     if(flag)
31         recursion(cur_layout, row + 1, N);
32 }
33 return;
34 }
35

```

4 调试分析

4.1 问题发现与解决

在使用栈来保存布局时，由于指针的各种调用出现了一些混乱，经过 Debug 模式的观察，最终查出错误并解决。

4.2 算法的时空复杂度分析

由于遍历了几乎所有情况，而每行只需要尝试前面几行没有试过的，因此时间复杂度约为 $o(n \times (n-1) \times \cdots \times 1) = o(n!)$.

下表展示了所用时间的表格。

皇后数	4	5	6	7	8	9	10	11	12
栈/秒	0.0000736	0.0001495	0.0007673	0.0027767	0.0104837	0.048083	0.228105	1.2051	6.8581
递归/秒	0.0000035	0.0000092	0.0000458	0.0001814	0.0006057	0.0029883	0.0137117	0.0739021	0.421939

下图红色是栈，蓝色是递归。灰色是用指数函数拟合的结果。

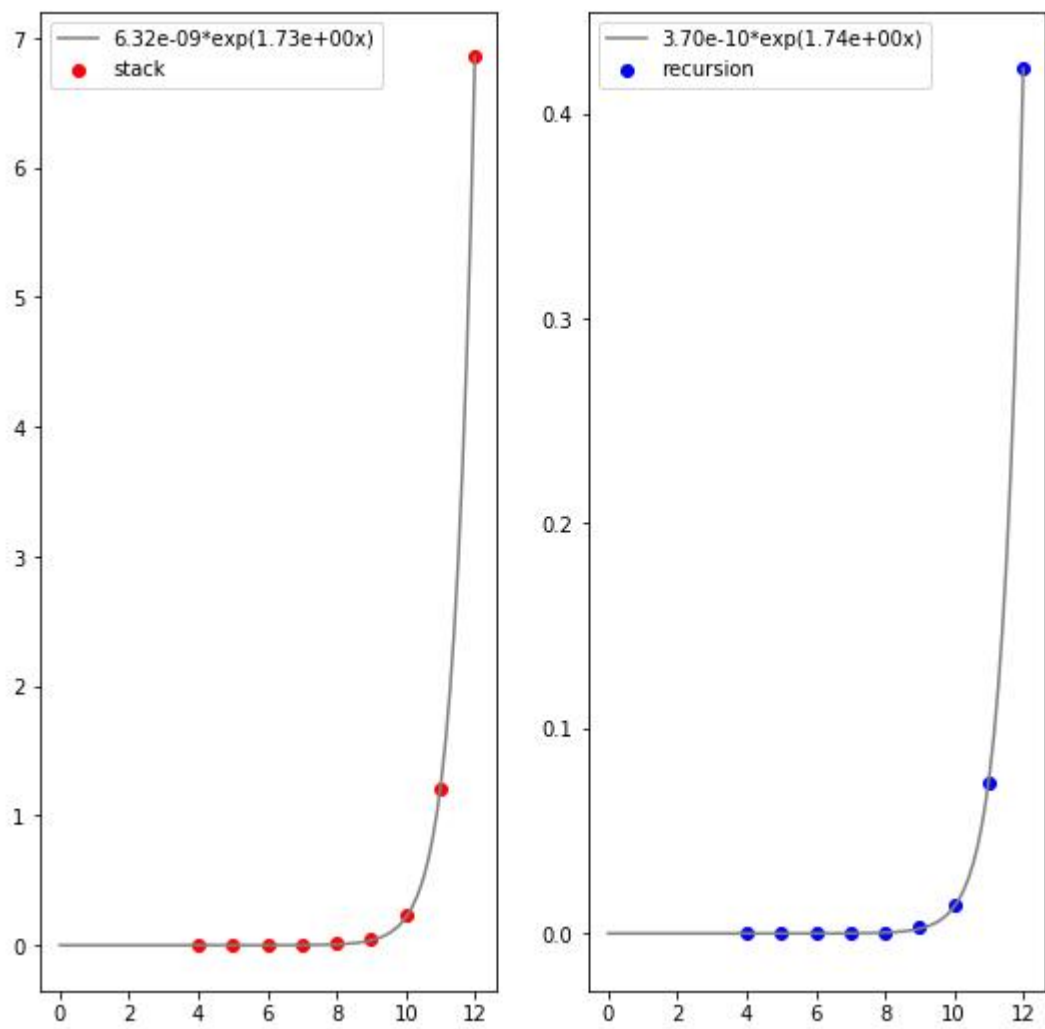


图 2: time

空间复杂度：对于栈方式，最多同时有 n 个大小为 n 的线性表存储所有布局，故为 $o(n^2)$. 而对于递归，则只用了一个线性表存储，故为 $o(n)$.

5 代码测试

5.1 data1

```
Input:
8
Output:
8
栈解法:
1
Q#####
####Q###
#####Q
#####Q##
##Q#####
#####Q#
#Q#####
###Q####
...
92
以上共 92 种
耗时 0.842 秒

递归解法:
1
Q#####
####Q###
#####Q
#####Q##
##Q#####
#####Q#
#Q#####
###Q####
...
92
以上共 92 种
耗时 0.965 秒
```

截图如下:

```
Microsoft Visual Studio 调试控制台

\\#\#\#\#\#Q\#
\\#\#Q\#\#\#\#
\\#\#\#\#\#Q\#\#

91
\\#\#\#\#\#\#Q
\\#\#Q\#\#\#\#\#
2\\#\#\#\#\#\#\#
\\#\#\#\#\#Q\#\#
\\#Q\#\#\#\#\#\#
\\#\#\#Q\#\#\#\#
\\#\#\#\#\#Q\#\#
\\#\#\#Q\#\#\#\#

92
\\#\#\#\#\#\#Q
\\#\#\#Q\#\#\#\#
2\\#\#\#\#\#\#\#
\\#\#Q\#\#\#\#\#
\\#\#\#\#\#Q\#\#
\\#Q\#\#\#\#\#\#
\\#\#\#\#\#Q\#\#
\\#\#\#\#Q\#\#\#\#

以上共92种
耗时0.965秒
```

图 3: data1

5.2 data2

Input:

4

Output:

4

栈解法:

1

#Q##

###Q

Q###

##Q#

2

##Q#

Q###

###Q

#Q##

以上共 2 种

耗时 0.003 秒

递归解法:

1

#Q##

###Q

Q###

##Q#

2

##Q#

Q###

###Q

#Q##

以上共 2 种

耗时 0.002 秒

截图如下:

```
Microsoft Visual Studio  
##Q##  
  
2  
##Q##  
Q###  
###Q  
#Q##  
  
以上共2种  
耗时0.003秒  
  
递归解法:  
1  
\\#Q\\#\\#  
\\#\\#\\#Q  
Q\\#\\#\\#  
\\#\\#Q\\#  
  
2  
\\#\\#Q\\#  
Q\\#\\#\\#  
\\#\\#\\#Q  
\\#Q\\#\\#  
  
以上共2种  
耗时0.002秒
```

图 4: data2

6 实验总结

通过本次的实验，本人更加熟悉了栈的使用。理解了怎么将一个递归的程序用栈来表示。

但由于算法原因，N 不能过大。尝试使用 fork/join，N=16 时依然需要一分多钟。

```
16 Queens' solutions amount is 14772512  
using 98584ms
```

图 5: concurrent

7 附录

7.1 N 皇后问题

```
1 #include <iostream>  
2 #include <cmath>
```

```

3  #include <ctime>
4  #include <windows.h>
5  using namespace std;
6
7
8  // Here we use a stack to store the information of
9  // the layout.
10 class layout {
11 private:
12     int cur;
13     int size;
14     int** layout_stack;
15
16 public:
17     layout(int N) {
18         size = 0;
19         cur = -1;
20         this->N = N;
21         layout_stack = new int*[N];
22         for (int i = 0; i < N; i++) {
23             layout_stack[i] = new int[N];
24         }
25     }
26
27     int N;
28     inline int stack_size() {
29         return cur + 1;
30     }
31
32     // get the top of stack
33     int* top() {
34         if (cur >= 0)
35             return layout_stack[cur];
36         return nullptr;
37     }
38
39     // pop the top of the stack
40     void pop() {
41         if (cur >= 0)
42             delete layout_stack[cur--];
43     }
44
45     // push cur_layout to the top of the stack
46     void push(int* cur_layout) {
47         layout_stack[++cur] = cur_layout;
48     }

```

```

49 // clone a layout and return it.
50 int* clone(int* ori) {
51     if (ori == nullptr)
52         return nullptr;
53     auto temp = new int[N];
54     for (int i = 0; i < N; i++) {
55         temp[i] = ori[i];
56     }
57     return temp;
58 }
59
60
61 // print the layout.
62 void printLayout(int* lay) {
63     for (int i = 0; i < N; i++) {
64         for (int j = 0; j < N; j++) {
65             if (j == lay[i]) cout << 'Q';
66             else cout << '#';
67         }
68         cout << endl;
69     }
70     cout << endl;
71 }
72
73 // to determine whether the last line in lay is ok.
74 bool is_lastline_ok(int* lay, int row) {
75     for (int j = 0; j < row; j++) {
76         if (lay[j] == lay[row] || abs(lay[j] - lay[row]) == row - j)
77     {
78         return false;
79     }
80     return true;
81 }
82
83 void solve() {
84     LARGE_INTEGER start, nFreq, end;
85     QueryPerformanceFrequency(&nFreq);
86     QueryPerformanceCounter(&start);
87     int counter = 0;
88     int* cur_layout = new int[N];
89     cur_layout[0] = 0;
90     push(cur_layout);
91     while (1) {
92         cur_layout = clone(top());
93         if (top() == nullptr)

```

```

94         break;
95     if (is_lastline_ok(cur_layout, stack_size() - 1)) {
96         // no conflict here
97         // whether finished
98         if (stack_size() == N) {
99             counter++;
100             cout << counter << endl;
101             printLayout(cur_layout);
102         }
103         else {
104             cur_layout[stack_size()] = 0;
105             push(cur_layout);
106             continue;
107         }
108     }
109     // need a recall
110     delete[] cur_layout;
111     cur_layout = top();
112     if (cur_layout == nullptr)
113         break;
114     while (cur_layout[stack_size() - 1] == N - 1) {
115         // tried all circumstance in this row
116         pop();
117         cur_layout = top();
118         if (cur_layout == nullptr)
119             break;
120     }
121     // found a row that the queen can move rightward.
122     if (cur_layout != nullptr) {
123         top()[stack_size() - 1]++;
124     }
125 }
126 QueryPerformanceCounter(&end);
127 cout << "以上共" << counter << "种" << endl;
128 cout << "耗时" << (double)(end.QuadPart - start.QuadPart) / (
129     double)nFreq.QuadPart << "秒" << endl;
130 }
131
132 };
133
134
135 // recursion method
136 int counter = 0;
137 void recursion(int* cur_layout, int row, const int N) {
138     if (row == N) {

```



```

139     counter++;
140     // print the layout.
141     cout << counter << endl;
142     for (int i = 0; i < N; i++) {
143         for (int j = 0; j < N; j++) {
144             if (j == cur_layout[i]) cout << 'Q';
145             else cout << "#";
146         }
147         cout << endl;
148     }
149     cout << endl;
150     return;
151 }
152 for (int i = 0; i < N; i++) {
153     cur_layout[row] = i;
154     bool flag = true;
155     // try all circumstances.
156     for (int j = 0; j < row; j++) {
157         if (cur_layout[j] == cur_layout[row]
158             || abs(cur_layout[j] - cur_layout[row]) == row - j) {
159             flag = false;
160             break;
161         }
162     }
163     // if ok, try next row. Here is the recursion.
164     if(flag)
165         recursion(cur_layout, row + 1, N);
166 }
167 return;
168 }
169
170
171 int main()
172 {
173     int N;
174     cin >> N;
175
176     cout << "栈解法: \n";
177     layout lay(N);
178     lay.solve();
179
180     cout << "\n递归解法: \n";
181     int* cur_layout = new int[N];
182     LARGE_INTEGER start, nFreq, end;
183     QueryPerformanceFrequency(&nFreq);
184     QueryPerformanceCounter(&start);

```

```
185         recursion(cur_layout, 0, N);
186         QueryPerformanceCounter(&end);
187         cout << "以上共" << counter << "种" << endl;
188         cout << "耗时" << (double)(end.QuadPart - start.QuadPart) / (double)
nFreq.QuadPart << "秒" << endl;
189     }
190
191
```