**综合设计报告**

**课程名称：**  数据结构

**题 目：**  皇后问题

**指导教师：**  蔡英

**设计起始日期：** 2013.xx.xx-2013.xx.xx

**学 院：**  计算机学院

**系 别：** 计算机科学与技术系

**学生姓名：**  刘鸿喆

**班级/学号：**  计科1201/2012011107

**成 绩：**

**一、需求分析**

* 实验目的：通过求解皇后问题，熟悉深度优先搜索法DFS（回溯法（Backtracking Algorithms）技术。
* 实验内容：
  + 由n2个方块排成n行n列的正方形称为n元棋盘。如果两个皇后位于n元棋盘上的同一行、同一列或同一对角线上，则称它们在互相攻击。现要找出使棋盘上n个皇后互不攻击的布局。
  + 编制程序解决上述问题，以n=6运行程序，输出结果。

**二、概要设计**

**1、数据结构**

栈：代替调用栈，使用非递归方法

**2、使用算法**

深度优先搜索法DFS

**三、详细设计**

**1、数据结构详细设计**

栈：

位置栈：Stack<Position> \_posStack;

临时栈：Stack<Position> \_temStack;

**2、算法**

定义一个数组A[n],A[i]表示第i行上的皇后所在的列数,i=0,1,...,n-1

第i行与第j行上的皇后互不攻击的充要条件是：|A[i]-A[j]|==|i-j|

初始时,各行皇后均放在第0列,然后从第0行开始逐行布局

设前i-1行已布好,现考虑第i行皇后的位置,从其当前位置A[i]开始向右探察

若A[i]<=n-1,则检查i行皇后与前i-1行皇后是否互不攻击

若有攻击，则i行皇后右移一位，重复这个过程

若无攻击，则

若i<n-1，继续布下一行的皇后

若i=n-1，输出布局，然后将n-1行皇后右移一位，重复这个过程，寻找另一种布局

若A[i]>n-1，则将i行皇后放在第0列，回退一行，考虑第i-1行皇后与前i-2行皇后互不攻击的下一个位置。若已退到-1行，则结束

**四、调试分析**

**1、调试过程中遇到的问题**

无。

**2、算法的时空分析**（n=皇后数）

时间：O(n^n)

空间：O(n^2)

**五、使用说明和测试结果**

**1、使用说明**

无。

**2、测试结果**

输出

size: 1, count: 1

size: 2, count: 0

size: 3, count: 0

size: 4, count: 2

size: 5, count: 10

size: 6, count: 4

size: 7, count: 40

size: 8, count: 92

size: 9, count: 352

size: 10, count: 724

size: 11, count: 2680

**六、心得体会**

Np问题不要碰，盲目DFS而不剪枝是大坑。

**七、附录**

**1、程序**

// #define \_\_DEBUG\_\_

#include <iostream>

#include "include/chessboard.h"

int main(int argc, char const \*argv[]) {

Chessboard chessboard;

for (int i = 1; i <= 11; i++) {

chessboard = Chessboard(i);

std::cout << "size: " << i << ", count: " << chessboard.amountOfQueens() << std::endl;

}

return 0;

}

#include <iostream>

#include "position.h"

#include "stack.h"

class Chessboard {

public:

Chessboard() : size(0) {}

Chessboard(int size) : size(size) {}

int amountOfQueens() {

int i, j, count = 0, layer;

Position pos;

for (i = 0; i < size; i++) {

\_posStack.push(Position(0, i));

#ifdef \_\_DEBUG\_\_

std::cout << "layer: " << 0 << ", push: " << i << std::endl;

#endif

j = 0;

while (!\_posStack.empty()) {

layer = \_posStack.top().x + 1;

if (layer == size) {

count++;

j = \_posStack.top().y + 1;

\_posStack.pop();

} else {

for (; j < size; j++) {

pos = Position(layer, j);

if (\_available(pos)) {

\_posStack.push(pos);

#ifdef \_\_DEBUG\_\_

std::cout << "layer: " << layer << ", push: " << j << std::endl;

#endif

j = 0;

break;

}

}

if (j == size) {

j = \_posStack.top().y + 1;

\_posStack.pop();

}

}

}

}

return count;

}

private:

bool \_available(Position pos) {

Position \_pos;

int i;

bool available = true;

while (!\_posStack.empty()) {

\_pos = \_posStack.top();

\_temStack.push(\_pos);

\_posStack.pop();

if (\_pos.x == pos.x ||

\_pos.y == pos.y ||

\_pos.x + \_pos.y == pos.x + pos.y ||

\_pos.x - \_pos.y == pos.x - pos.y) {

available = false;

}

}

while (!\_temStack.empty()) {

\_pos = \_temStack.top();

\_posStack.push(\_pos);

\_temStack.pop();

}

return available;

}

int size;

Stack<Position> \_posStack;

Stack<Position> \_temStack;

};

class Position {

public:

Position() : x(0), y(0) {}

Position(int x, int y) : x(x), y(y) {}

int x;

int y;

};

/\*\*

\* @file stack.h

\* @brief Data structure: stack

\* @author Darren Liu (MSR.B, msr-b)

\* @date 2013/10/13

\* @copyright Copyright (c) 2013 Darren Liu. All rights reserved.

\*/

#include <iostream>

#include <cstdlib>

/\*\*

\* @brief Data structure: stack

\*/

template <typename \_Ty>

class Stack {

public:

/\*\*

\* @brief constructor of stack

\*/

Stack() : \_top(NULL), \_count(0) {}

/\*\*

\* @brief push element into the stack

\*

\* @param data (const \_Ty&) the element

\*

\* @return succeeded or not

\* @retval true push succeded

\* @retval false push failed

\*/

bool push(const \_Ty &data) {

StackNode \*top = new StackNode(data, \_top);

if (!top) {

return false;

}

\_top = top;

\_count++;

return true;

}

/\*\*

\* @brief remove the top element

\*

\* @return succeed or not

\* @retval true pop succeded

\* @retval false pop failed

\* @see \_Ty top() const

\*/

bool pop() {

if (empty()) {

return false;

}

StackNode \*top = \_top;

\_top = \_top -> next;

delete top;

\_count--;

return true;

}

/\*\*

\* @brief test whether the stack is empty

\*

\* @return empty or not

\* @retval true empty

\* @retval false not empty

\*/

bool empty() const {

return (\_top) ? false : true;

}

/\*\*

\* @brief clear all entries in the stack

\*/

void clear() {

while (!empty()) {

pop();

}

}

/\*\*

\* @brief return the top element

\*

\* @return top element

\*/

\_Ty top() const {

if (!\_top) {

std::cerr << "No elements in stack." << std::endl;

exit(EXIT\_FAILURE);

}

return \_top -> data;

}

/\*\*

\* @brief count of entries

\*

\* @return count of entries

\*/

int count() const {

return \_count;

}

/// @brief destructor of stack

~Stack() {

for (StackNode \*top = \_top; top; top = \_top) {

\_top = \_top -> next;

delete top;

}

}

private:

class StackNode {

public:

StackNode(const \_Ty &data, StackNode \*next = NULL) : data(data), next(next) {}

\_Ty data;

StackNode \*next;

};

/// @brief count

int \_count;

/// @brief top pointer

StackNode \*\_top;

};