

**Lab Report**

**实验报告**

|  |  |
| --- | --- |
| **Course**: | Class Libraries and Data Structures |
| **Semester**: | 1st semester of the academic year **2023-2024** |
| **Major**: | Software Engineering |
| **Class**: | 2022 |
| **Student Name**: | 朱昊 |
| **Student ID:** | 222022321062008 |
| **Teacher:** | ZHAO, Hengjun (赵恒军) |

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | Recursion and Backtracking Framework  递归和回溯框架 | | | |
| Date | | Oct，2023 | Type | | ☑Confirmatory （验证确认型）  ☑Design（设计型）  🗆Comprehensive（综合型） |
| 1. **Objective & Requirements（实验目的）**    1. Understand the design and implementation principles of recursion and backtracking algorithm to solve a practical problem   理解递归和回溯算法的原理和实现方法   * 1. Understand the object-oriented design of backtracking framework using C++, especially the iterator inner class   理解采用面向对象思想设计通用回溯算法框架的思路，尤其是其中基于迭代器的通用算法的设计思想   * 1. Grasp the use of the backtracking framework to solve a specific problem, i.e. the 8-Queen problem.   掌握面向对象回溯框架的使用，以 8皇后问题为例培养利用回溯框架解决实际问题的能力 | | | | | |
| 1. **Experimental environment (**platform and software**)（实验环境）**   Windows 7 (or higher versions) + Visual Studio 2010 (or higher versions) | | | | | |
| 1. Experimental content and design (Main Content, Procedure, Codes and Results)（此部分应包含每一个实验内容的详细设计，含实验思路、详细实验步骤、核心代码说明等）   Task 1  A chessboard has eight rows and eight columns. In the game of chess, the queen is the most powerful piece: she can attack any piece in her row, any piece in her column, and any piece in either of her diagonals. Develop and validate a program to place eight queens on a chessboard in such a way that no queen is under attack from any other queen.    在8行乘8列的棋盘上，放置8枚皇后棋子，使得任意两个皇后棋子不可互相攻击，即不在同一行、同一列和同一对角线上。如上图所示。  Requirement（要求）:   1. You should use the backtrack framework to solve the 8-Queen problem.   使用所提供通用回溯框架解决该问题。   1. The codes in 4 of the files are fixed and you are not allowed to modify them 2. main.cpp 3. backtrack.h 4. backtrack.cpp 5. application.h   以上4个文件中的代码不允许做任何修改。   1. The codes in 3 of the files are to be implemented： 2. position.h 3. position.cpp 4. application.cpp (including the iterator)   以上3个文件中的代码需要根据实际问题进行实现    #include "application.h"  #include <iostream>  using namespace std;  #define NULL 0  //board  // 创建一个8乘8的棋盘  int maze[8][8] = { {0, 0, 0, 0, 0, 0, 0, 0,} ,  {0, 0, 0, 0, 0, 0, 0, 0} ,  {0, 0, 0, 0, 0, 0, 0, 0} ,  {0, 0, 0, 0, 0, 0, 0, 0} ,  {0, 0, 0, 0, 0, 0, 0, 0} ,  {0, 0, 0, 0, 0, 0, 0, 0} ,  {0, 0, 0, 0, 0, 0, 0, 0} ,  {0, 0, 0, 0, 0, 0, 0, 0} };  void Application::initialize() {}  Position Application::getStartPosition()  {  return Position(0, 0);// 起始位置人从（0，0）开始  }  bool Application::isValid(const Position& p)  {  int row = p.getRow();  int column = p.getColumn();  int newrow = p.getRow(); // newrow,newcolumn主要用于判断这个点在正副两条斜线上是//否有皇后  int newcolumn = p.getColumn();  int min = 0;  //新加的： 提前设置好三个分别判断该点行，列，和斜线上是否有皇后的布尔类型变量  bool noqueenrow = true;  bool noqueencolumn = true;  bool noqueenoblique = true;  for (int i = 0; i < 8; i++) {  if (maze[i][p.getColumn()] == 1) {// 通过循环该行的所有元素，来判断是否有皇后  noqueenrow = false;  }  if (maze[p.getRow()][i] == 1) {// 通过循环该列上的所有元素，来判断是否有皇后  noqueencolumn = false;  }  }  //主斜线，通过循环，把newrow和newcolumn两个变量的值循环到p这个点的主斜线的最后一个点的值  while (newrow < 7 && newcolumn < 7) {  newrow++;  newcolumn++;  }  // 判断那个变量的数值小，就令min等于其值，min主要是用与确定主斜线最后一个点循环到主斜线第一个点要循环的次数。  if (newrow < newcolumn) {  min = newrow;  }  else {  min = newcolumn;  }  // 开始从主斜线最后一个点循环到第一个点，每次循环判断有没有皇后，一旦有就令noqueenoblique = false;然后跳出循环  for (int i = 0; i < min + 1; i++) {  if (maze[newrow - i][newcolumn - i] == 1) {  noqueenoblique = false;  break;  }  }  //副斜线，道理同主斜线，但是min值应该是newcolumn - newrow;  //重新等于p的位置  newrow = p.getRow();  newcolumn = p.getColumn();  while (newrow < 7 && newcolumn < 7) {  newrow--;  newcolumn++;  }  min = newcolumn - newrow;  for (int i = 0; i < min + 1; i++) {  if (maze[newrow + i][newcolumn - i] == 1) {  noqueenoblique = false;  break;  }  }  //还要加上判断noqueenrow noqueencolumn noqueenoblique全为false  return (row >= 0 && row <= 7) && (column >= 0 && column <= 7) && maze[row][column] == 0 && noqueenrow == true && noqueencolumn == true && noqueenoblique == true;  }  void Application::progress(const Position& p)//让走过的地方发生改变, 皇后问题就是让这个点为皇后（1）  {  maze[p.getRow()][p.getColumn()] = 1;  }  bool Application::success(const Position& p)//判断是否成功  {  int queennum = 0;  for (int i = 0; i < 8; i++) {  for (int j = 0; j < 8; j++) {  if (maze[i][j] == 1) {  queennum++;  }  }  }  //如果有8个皇后了就代表成功  return queennum == 8;  }  void Application::goBack(const Position& p) //修改该位置的值  {    maze[p.getRow()][p.getColumn()] = 0;  }  void Application::print()  {  for (int i = 0; i < 8; i++)  {    for (int j = 0; j < 8; j++)  cout << maze[i][j] << " ";  cout << endl;  }  }  //======================================================================  struct itrPosition  {  int row;  int column;  //int turn;//方向  int count;//需要用来计数，因为我的getNextPosition（）固定从这个点的下一列的第一个元素开始返回，count存储是这是第几次了。  };  //application iterator  Application::Iterator::Iterator()  {  currItrPosPtr = NULL;  }  Application::Iterator::Iterator(const Position& currPos)//有参构造  {  itrPosition\* p = new itrPosition;  p->row = currPos.getRow();  p->column = currPos.getColumn();  p->count = 0;  currItrPosPtr = p;  }  Position Application::Iterator::getNextPosition()//根据函数体里的计算，返回下一个位置，调用一次，方向就会+1  {  int row = ((itrPosition\*)currItrPosPtr)->row;  int column = ((itrPosition\*)currItrPosPtr)->column;  /\*int turn = ((itrPosition\*)currItrPosPtr)->turn;\*/  int count = ((itrPosition\*)currItrPosPtr)->count;  //下面这个if就保证了,是从原来位置的下一列最上面的元素开始，并且每回到原来的点一次，那么返回的位置就会是原来位置的下一列的第count行元素位置。  if (count < 8) {  column++;  row = count;  count++;  }  ((itrPosition\*)currItrPosPtr)->count = count;  return Position(row, column); //return the new position，返回新位置  }  bool Application::Iterator::noNextPosition()  {  return ((itrPosition\*)currItrPosPtr)->count > 7;//若count=8了说明没有下一列的元素都试过了，没有新位置了  }  Application::Iterator::~Iterator()  {  delete (itrPosition\*)currItrPosPtr;  } | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems）（此部分应包含实验结果，对实验结果的分析，实验收获的总结，实验中存在问题的讨论等；另外，需要回应一下如下思考题：1.在Application的内部类Iterator中为什么封装了一个无类型指针，其作用是什么？2. 如何修改使你的程序以找出这个棋盘上所有可能的八个皇后放置位置？）   本实验只修改了application.cpp里的代码，成功在终端打印了起始位置为（0，0）的八皇后问题的一个解。这次试验让我很好地理解采用面向对象思想设计通用回溯算法框架的思路，尤其是其中基于迭代器的通用算法的设计思想以及掌握面向对象回溯框架的使用。这次以 8皇后问题为例培养了我利用回溯框架解决实际问题的能力。  思考题：   1. 因为该算法是通用算法，要想让算法具备适用于更多场景，那么不明确指向某一数据类型，而能强转成任何类型的指针void更加适合。   Void类型的指针不是指向空，恰恰相反，任何类型的指针都可以分配给一个void指针。只需要将void指针解引用前将void指针明确转换为一个具体的指针类型。    2 main函数修改如下：  ，  需要在backtrack.h里添加一个methodnum变量记录一共有多少可能解。    再对回溯算法进行小修改即可      如上图一共产生了92种结果。 | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |