

**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**: |  |
| **Student ID:** |  |
| **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  与寻路问题相似，都是每到一位置都判断是否可以放置，不过寻路问题是可以四个方向选择下一位置，而八皇后问题是在下一行的七个位置进行选择，如果发现下一行的每个位置都不能放置，那就撤回到上一层，说明上一个位置有问题。  八皇后是横竖斜都不行，所以我的方案是每放置一个皇后都要将其横竖斜都改为自己的标记，而每撤回一个皇后都要将其标记撤回，这就要每个皇后都有不同的标记，或者将每个皇后的标记位置都存起来，这样撤回的时候就可以真正地返回到上一状态。  至于下一行尝试到哪一个，这也是需要记录的，就存在迭代器（指向当前皇后）里，这样每次撤回到上一行就能知道该从什么位置继续。  #include "application.h"  #include <iostream>  using namespace std;  #define NULL 0  //board  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() {}  //从第一行从第一个开始依次往后尝试,这样只能找出来第一个皇后放在(0,0)的情况  Position Application::getStartPosition()  {  return Position(0, 0);    }  bool Application::isValid(const Position& p) //只有为0才有效  {  int row = p.getRow();  int column = p.getColumn();  //不超出范围，且值为0，才可以放置  return (row >= 0 && row <= 7) && (column >= 0 && column <= 7) && (maze[row][column] == 0);  }  void Application::progress(const Position& p) //每放一个皇后，标记自己为9，行列斜为当前皇后所在行数  {  //检测是否为0，若是0，则改为当前行数的值，如第二行的皇后就全改为2，第三行的皇后就全改为为3  //若不是0，那就不变，因为已经被其他皇后控制了  //行列控制：  for (int i = 0; i <= 7; i++) {  if (maze[p.getRow()][i] == 0) {  maze[p.getRow()][i] = p.getRow() + 1; //行占有  }  }  for (int i = 0; i <= 7; i++) {  if (maze[i][p.getColumn()] == 0) {  maze[i][p.getColumn()] = p.getRow() + 1; //列占有  }  }  //斜线控制  //主对角线  int row, column;//找到起点  //主对角线占有  if (p.getRow() > p.getColumn()) {  row = p.getRow() - p.getColumn();  column = p.getColumn() - p.getColumn();  }  else {  row = p.getRow() - p.getRow();  column = p.getColumn() - p.getRow();  }  while (row <= 7 && column <= 7) { //任何一边到边界都算是结束  if (maze[row][column] == 0) {  maze[row][column] = p.getRow()+1;  }  row++;  column++;  }  //副对角线  if ((p.getRow() + p.getColumn()) <= 3) {  row = p.getRow() - p.getRow();  column = p.getColumn() + p.getRow();  }  else {  row = p.getRow() - (7 - p.getColumn());  column = 7;  }    while (row <= 7 && column >= 0) {  if (maze[row][column] == 0) {  maze[row][column] = p.getRow() + 1;  }  row++;  column--;  }  maze[p.getRow()][p.getColumn()] = 9;//皇后所在位置标记为9  }  bool Application::success(const Position& p)  {  if (p.getRow() == 7) { //若当前位置为第七行，那说明这个地方已经放下了皇后，也就是成功了  return true;  }  else {  return false;  }  }  //返回时将此时尝试的pos标记为0,还要将当前皇后占有的格数撤回，全部遍历一遍，等于当前行数的全部改成0  void Application::goBack(const Position& p)  {  for (int i = 0; i < 8; i++) {  for (int j = 0; j < 8; j++) {  if (maze[i][j] == p.getRow()+1) {  maze[i][j] = 0;  }  }  }  maze[p.getRow()][p.getColumn()] = 0;  }  void Application::print() //打印  {  for (int i = 0; i < 8; i++)  {  for (int j = 0; j < 8; j++) {  if (maze[i][j] != 9) { //避免修改maze本身，因为之后每找出一次结果都要print，防止回溯时maze被修改了  cout << 0 << " ";  }  else {  cout << maze[i][j] << " ";  }    }  cout << endl;  }  }  //======================================================================  struct itrPosition  {  int row;  int column;  int turn;  };  //application iterator  Application::Iterator::Iterator()  {  currItrPosPtr = NULL; //将无类型指针指向空  }  Application::Iterator::Iterator(const Position& currPos) //以行为单位，所以让迭代器指向这一行  {  itrPosition\* p = new itrPosition;  p->row = currPos.getRow();  p->column = 0;  p->turn = 0;  currItrPosPtr = p; //将无类型指针指向迭代器，此时的类型为itrPosition\*  }  Position Application::Iterator::getNextPosition() //这里要改成按照行进行遍历  {  int row = ((itrPosition\*)currItrPosPtr)->row;  int column = ((itrPosition\*)currItrPosPtr)->column;  int turn = ((itrPosition\*)currItrPosPtr)->turn;  //int turn = ((itrPosition\*)currItrPosPtr)->turn;  row++; //转到下一行，但不存到迭代器中，这样就能每次迭代都在下一行  column = turn; //计数器记录列数  turn++; //计数器加1  ((itrPosition\*)currItrPosPtr)->turn = turn; //计数器的值每次都要更新  ((itrPosition\*)currItrPosPtr)->column = column; //列数更新  return Position(row, column); //return the new position  }  bool Application::Iterator::noNextPosition() //当前行尝试完毕  {  return ((itrPosition \*)currItrPosPtr)->turn > 7;  }  Application::Iterator::~Iterator()  {  delete (itrPosition\*)currItrPosPtr;  }  结果： | | | | | |
| 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？） 2. 对每一个application，迭代器的实现都是不一样的，而我们想要让application的各个声明都高度抽象，这样只需要根据应用场景修改函数具体实现就好。所以定义了一个无类型的指针，指向未知的用于找到下一位置的信息（也就是迭代器真正的数据成员），像本题中它是指向了一个自定义的结构体itrPosition，类型自适应，里面除了位置，还有一个状态量（计数器），这是本题的应用场景，而切换到别的应用场景，这个无类型指针就会指向其它要封装的信息，这样这个无类型指针相当于一个接口，可以随意实例化（不是很严谨，但就是那个意思），保证了Application的抽象。 3. 每success一次不能直接停止程序，要先打印出来，再接着回溯，寻找下一个放法，直到撤回至第一行，返回false，本次backtrack结束。先将开始位置为(0,0)的所有情况都试出来，当全部试完，返回false（其实肯定不会返回true，因为success不返回true，但返回什么也无所谓），就开始尝试开始位置为(0,1)的所有情况，如此循环到开始位置为(0,7)时的所有情况也寻找完毕。   修改的函数如下：  Position Application::getStartPosition()  {  //return Position(0, 0);  static int c = 0; //静态变量，只有第一次调用会初始化  return Position(0, c++); //每一次调用此函数都能让c加1，也就是不断挪初始位置  }  bool BackTrack::backTrack(const Position& currPos)  {    app.progress(currPos);  if (app.success(currPos)) {  app.print(); //每次成功都打印一次  cout << endl;  globalcount++; //方法种数加1  app.goBack(currPos); //继续回溯  return false; //当作没有找到，返回false  }  else  {  Application::Iterator itr(currPos);  Position nextPos;  while (!itr.noNextPosition())  {  nextPos = itr.getNextPosition();  if (app.isValid(nextPos) && backTrack(nextPos)) {//recursive call if valid  return true;  }    }  //cannot reach goal from currPos  app.goBack(currPos); //backtrack  return false; //from currPos, no solution  }  }  int main()  {  Application app;  BackTrack btr(app);  for (int i = 0; i <= 7; i++) {  btr.backTrack(app.getStartPosition());  }  cout << btr.globalcount;  return 0;  }  这里定义了一个globalcount，是backtrack类的公有成员，用于记八皇后放置方法的数目。    92是尝试出的解法总数，前面打印了每一种解法。 | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |