

**Lab report**

|  |  |
| --- | --- |
| **Course**: | Class Libraries and Data Structures |
| **Semester**: | 1st semester of the academic year **2021-2022** |
| **Major**: | Software Engineering |
| **Class**: | 2020 |
| **Student Name**: | 温长锟 |
| **Student ID:** | 222020321062106 |
| **Teacher:** | ZHAO, Hengjun (赵恒军) |

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | Backtracking Framework | | | |
| Date | | Oct 25，2021 | Type | | □Confirmatory  √ Design  □Comprehensive |
| 1. **Objective & Requirements**    1. Understand the design and implementation principles of recursion and backtracking algorithm to solve a practical problem    2. Understand the object-oriented design of backtracking framework using C++, especially the iterator inner class    3. Grasp the use of the backtracking framework to solve a specific problem, i.e. the 8-Queen problem. | | | | | |
| 1. **Experimental environment (**platform and software**)**   Windows 7 (or higher versions) + Visual Studio 2010 (or higher versions) | | | | | |
| 1. **Experimental content and design** (Main Content, Procedure, Codes and Results)   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.    Requirement:   1. You should use the backtrack framework to solve the 8-Queen problem. 2. You can start from a fixed position, e.g. (0,0), or ask the user to input the position of the first queen 3. The codes in 4 of the files are fixed and you are not allowed to modify them 4. main.cpp 5. backtrack.h 6. backtrack.cpp 7. application.h 8. The codes in 3 of the files are to be implemented： 9. position.h 10. position.cpp 11. application.cpp (including the iterator)   Content & Design   1. use the backtrack framework to solve the 8-Queen problem.      1. start from a fixed position, e.g. (0,0)      1. Do not change the following four files   main.cpp  backtrack.h  backtrack.cpp  application.h   1. The codes in 3 of the files are implemented   position.h  position.cpp  application.cpp  Procedure & Codes  Step1.implemented position.h    Step2.implemented position.cpp    Step3 implemented application.cpp    CODE:  #include "application.h"  #include <iostream>  using namespace std;  //board  const int SIZE = 8;  bool board[SIZE][SIZE];  void Application::initialize()  {  for (int i = 0; i < SIZE; i++)  for (int j = 0; j < SIZE; j++)  board[i][j] = false;  board[0][0] = true;  }  Position Application::getStartPosition()  {  //please implement  for (int j = 0; j < SIZE; j++)  {  if (board[0, j])  return Position(0, j);  }  }  bool Application::isValid(const Position& p)  {  int row = p.getRow();  int col = p.getCol();  int count = 0;  for (int i = 0; i < row; i++) {  if (board[i][col] == true) {  return false;  }  }  for (int i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--) {  if (board[i][j] == true) {  return false;  }  }  for (int i = row - 1, j = col + 1; i >= 0 && j < SIZE; i--, j++) {  if (board[i][j] == true) {  return false;  }  }  return true;  }  void Application::extend(const Position& p)  {  if (isValid(p))  {  board[p.getRow()][p.getCol()] = true;  }  };  bool Application::success(const Position& p)  {  return (p.getRow() == SIZE );    }  void Application::goBack(const Position& p)  {  board[p.getRow()][p.getCol()] = false;  }  void Application::print()  {  for (int row = 0; row < SIZE; row++)  {  for (int column = 0; column < SIZE; column++)  cout << board[row][column] << ' ';  cout << endl;  }  }  struct itrFields  {  int row, col;  int dir\_x, dir\_y;  };  Application::Iterator::Iterator()  {  itrCurrPosPtr = NULL;  }  Application::Iterator::Iterator(const Position& Pos)  {  itrFields \*itrPtr = new itrFields;    itrPtr->row = Pos.getRow();  itrPtr->col = Pos.getCol();  itrPtr->dir\_x = Pos.getRow();  itrPtr->dir\_y = 0;  itrCurrPosPtr = itrPtr;    }  Position Application::Iterator::getNextPosition()  {  itrFields\* itrPtr = (itrFields\*)itrCurrPosPtr;  int nextRow = itrPtr->row,  nextCol = itrPtr->col;  switch (itrPtr->row)  {  case 0:nextRow = 0; break;  case 1:nextRow = 1; break;  case 2:nextRow = 2; break;  case 3:nextRow = 3; break;  case 4:nextRow = 4; break;  case 5:nextRow = 5; break;  case 6:nextRow = 6; break;  case 7:nextRow = 7; break;  }  switch (itrPtr->col++)  {  case 0:nextCol = 0; break;  case 1:nextCol = 1; break;  case 2:nextCol = 2; break;  case 3:nextCol = 3; break;  case 4:nextCol = 4; break;  case 5:nextCol = 5; break;  case 6:nextCol = 6; break;  case 7:nextCol = 7; break;  }  Position next(nextRow + 1, nextCol);  return next;  }  bool Application::Iterator::noNextPosition()  {  if (((itrFields\*)itrCurrPosPtr)->dir\_x > 7||((itrFields\*)itrCurrPosPtr)->dir\_y > 7)  return true;  return false;  }  #ifndef POSITION\_H  #define POSITION\_H  class Position  {  private:  int row,col;  public:  Position();  Position(int row, int column);  void setPosition(int row, int column) ;  int getRow() const ;  int getCol() const;  };  #endif  #include "position.h"  //please implement  Position:: Position(int x, int y)  {  this->row = x; this->col = y;  };  void Position::setPosition(int row, int col) {  this->row = row; this->col = col;  };  Position::Position()  {  row = 0; col = 0;  };  int Position::getRow() const {  return row;  };  int Position::getCol() const {  return col;  };  Results: | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results   and summing up the harvest and the existing problems）  Under the guidance of Mr. Zhao, I understand the design and implementation principle of recursive and backtracking algorithms. Mr. Zhao listed the maze backtracking problem in the experimental class, which is that I have a deeper understanding of recursion and backtracking algorithms, and also understand how to use recursion and backtracking algorithms to solve practical problems.In this experiment, I used C + + programming language to understand the surface user design of backtracking framework through the internal classes of iterators in C + +. This experiment also uses the backtracking framework to solve the specific eight queens problem. | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |