重 庆 大 学

学 生 实 验 报 告

实验课程名称 算法设计与分析

开课实验室 DS1501

学 院 大数据与软件学院 年级 2021 专业班软件工程 X 班

学 生 姓 名 XXX 学 号 2021XXXX

开 课 时 间 2022 至 2023 学年第 二 学期

总 成 绩	
教师签名	XXX

大数据与软件学院制

《算法设计与分析》实验报告

开课实验室：DS1501　　2023 年 5 月 30 日

学院	大数据与软件学院	年级、专业、班	2021 级软件工程 X 班	姓名	XXX	成绩	
课程名称	算法设计与分析	实验项目名　　称	回溯法实验和分枝限界法实验	指导教师	XXX		
教师评语	教师签名： 年　月　日						

一、 实验目的

- 掌握回溯法算法框架，掌握分枝限界法的特点和算法框架。
- 熟练掌握“满足方程解问题求解算法”的实现，熟练掌握“4 皇后问题求解算法”的实现。
- 主要任务：实现教材配套实验指导书“第 2 章 2.5.4 小节 求解满足方程解问题”和“第 2 章 2.6.1 小节 求解 4 皇后问题”。

二、使用仪器、材料

PC 微机 Lenovo Legion R9000P2021H；
Windows11 操作系统;
Clion2023 编译环境；

三、实验步骤

2.5.4 实验 4 求解满足方程解问题

编写一个实验程序,求出 a、、c、d、e,满足 a * b-c * d 十 e=1 方程,其中所有变量的取值为 1~5 并且均不相同。

解:

本题相当于求出 1~5 的一个排列,满足方程要求。采用解空间为排列数的回溯算法框架。

2.6.1 实验 1 求解 4 皇后问题

编写一个实验程序,采用队列式和优先队列式分枝限界法求解 4 皇后问题的一个解,分析这两种方式的求解过程，比较创建的队列结点个数。

解:

(1)队列式分枝限界法：

采用队列式分枝限界法求解,只需要设计一个普通队列,如果已经放置了 i 个皇后,考察第 i+1 个皇后时需要判断是否有冲突。为此,在每个结点中存放搜索到该结点为止所有放好的皇后。由于每行只能放一个皇后,只需要保存皇后的列位置。声明队列中的结点类型 NodeType。

首先将根结点(虚结点,其 row=-1)进 qu 队。队列 qu 不空时循环:出队结点 e,考察 i=e.row+1 行的子结点,仅仅将与 e.cols 不冲突的子结点进队。由于需要求所有的解,不发生冲突就是剪枝条件。当出队结点 e 时有 e.row=n-1；

(2)优先队列式分枝限界法：

采用优先队列式分枝限界法求解,需要设计一个优先队列,以当前结点的 row 为限界,即 row 越大越好,修改 NodeType。

四、实验过程原始记录(数据、图表、计算等)

2.5.4 实验 4 求解满足方程解问题

编写一个实验程序,求出 a、、c、d、e,满足 a * b-c * d 十 e=1 方程,其中所有变量的取值为 1~5 并且均不相同。

源代码：

```
#include <iostream>

int ans[5];   // 定义长度为5的整型数组ans，用于存放排列组合的结果
int n=5;      // 定义常量n，表示数组的长度

// 打印出数组中的元素，按照一定格式展示
void showanswer(int ans[]){
    std::cout<<ans[0]<<"*"<<ans[1]<<"-"<<ans[2]<<"*"<<ans[3]<<
             "-"<<ans[4]<<" = 1"<<std::endl;
}

// 递归函数，用于生成数组ans的所有排列组合，并在满足条件时调用showanswer函数打印出结果
void resolve(int i){
    if(i==n){  // 当i=n时，说明已经生成了一组排列组合，判断是否符合条件并输出
        if(ans[0]*ans[1]-ans[2]*ans[3]-ans[4]==1){
            showanswer(ans);
        }
    }
    else{   // 当i<n时，继续向下递归生成排列组合
        for(int j=i;j<n;j++){
            std::swap( &: ans[i], &: ans[j]);  // 将第i个元素与第j个元素交换
            resolve( i: i+1);  // 继续递归生成下一个位置的元素
            std::swap( &: ans[i], &: ans[j]);  // 将第i个元素与第j个元素交换回来，方便下一次循环生成其他排列组合
        }
    }
}

int main() {
    // 首先将数组ans初始化为1~5的整数序列
    for(int i=0;i<n;i++){
        ans[i]=i+1;
    }
    // 输出提示信息，并调用resolve函数生成所有排列组合
    std::cout<<"2.5.4问题求解结果为： "<<std::endl;
    resolve( i: 0);
    return 0;   // 程序结束，返回0表示成功执行
}
```

运行结果：

```
2.5.4问题求解结果为：
3*4-2*5-1 = 1
3*4-5*2-1 = 1
4*3-2*5-1 = 1
4*3-5*2-1 = 1

进程已结束,退出代码0
```

2.6.1 实验 1 求解 4 皇后问题

编写一个实验程序,采用队列式和优先队列式分枝限界法求解 4 皇后问题的一个解,分析这两种方式的求解过程，比较创建的队列结点个数。

源代码：

（1）队列式分枝限界法：

```
#include <iostream>
#include <vector>
#include <queue>

constexpr int NofQueen = 4; // 定义皇后数量
// 定义结点的数据结构
struct NodeType{
    int number; // 结点编号，用于调试和输出
    int row; // 皇后所在行，范围从 -1 到 NofQueen-1，其中 -1 表示未确定行
    std::vector<int> column; // 皇后所在列，长度为 NofQueen，存储每个皇后所在的列号
};

// 打印结点信息，用于调试和输出
void shownode(const NodeType& n){
    if(n.row!=-1){
        std::cout<<"编号: "<<n.number<<"  ,对应位置("<<n.row<<","<<n.column[n.row]<<")"<<std::endl;
    }
    else{
        std::cout<<"编号: "<<n.number<<"  ,对应位置("<<n.row<<",*)"<<std::endl;
    }
}

// 检查是否可以在 (x,y) 放置皇后
bool check(const std::vector<int>& column,int x,int y){
    for(int p=0; p<x; p++){
        if(column[p]==y || abs( x: column[p]-y)==abs( x: p-x)){
            return false;
        }
    }
    return true;
}
```

```
// 解决 N 皇后问题的函数
void resolve(){
    int count = 1; // 计数器，用于给每个结点编号
    int x,y;
    NodeType n1,n2;
    std::queue<NodeType> qu; // 定义一个队列用于存储结点

    // 初始化第一个结点
    n1.number = count++; // 给结点编号，计数器加 1
    n1.row = -1; // 第一个皇后未确定行，初始化为 -1
    qu.push( x: n1); // 将第一个结点放入队列中
    std::cout<<"进队：  ";
    shownode( n: n1);
```

```
    while(!qu.empty()){ // 循环直到队列为空
        n1 = qu.front(); // 取出队头结点
        qu.pop(); // 弹出队头结点
        std::cout<<"出队： ";
        shownode( n: n1);
        if(n1.row==NofQueen-1){ // 如果已经放置了 NofQueen 个皇后，得到一组解
            std::cout<<"产生一个解为： ";
            for(x=0;x<NofQueen;x++){
                std::cout<<"["<<x+1<<","<<n1.column[x]+1<<"] "; // 输出每个皇后所在的位置
            }
            std::cout<<std::endl;
            return ; // 返回
        }
        else{
            for(y=0;y<NofQueen;y++){ // 在下一行中尝试放置皇后
                x=n1.row+1; // 下一行的行号
                if(check(n1.column,x,y)){ // 检查是否可以放置皇后
                    n2.number = count++; // 给结点编号，计数器加 1
                    n2.row = x; // 更新皇后所在行
                    n2.column = n1.column; // 复制上一行的列号
                    n2.column.push_back(y); // 在下一行中放置皇后
                    qu.push( x: n2); // 将新结点放入队列中
                    std::cout<<"  进队子结点： ";
                    shownode( n: n2);
                }
            }
        }
    }
}
```

```
// 主函数
int main(){
    std::cout<<"2.6.1 "<<std::endl<<NofQueen<<"皇后问题求解过程为："<<std::endl;
    resolve(); // 解决 N 皇后问题并输出求解过程和最终结果
    return 0;
}
```

（2）优先队列式分枝限界法：

```
#include <iostream>
#include <vector>
#include <queue>

constexpr int NofQueen = 4; // 定义皇后数量
// 定义结点的数据结构
struct NodeType{
    int number; // 结点编号，用于调试和输出
    int row; // 皇后所在行，范围从 -1 到 NofQueen-1，其中 -1 表示未确定行
    std::vector<int> column; // 皇后所在列，长度为 NofQueen，存储每个皇后所在的列号
    bool operator < (const NodeType &s) const{ // 定义小于运算符，便于使用优先队列
        return row<s.row; // 按照 row 值从小到大排序
    }
};

// 打印结点信息，用于调试和输出
void shownode(const NodeType& n){
    if(n.row!=-1){
        std::cout<<"编号: "<<n.number<<"  ,对应位置("<<n.row<<","<<n.column[n.row]<<")"<<std::endl;
    }
    else{
        std::cout<<"编号: "<<n.number<<"  ,对应位置("<<n.row<<",*)"<<std::endl;
    }
}

// 检查是否可以在 (x,y) 放置皇后
bool check(const std::vector<int>& column,int x,int y){
    for(int p=0; p<x; p++){
        if(column[p]==y || abs( x: column[p]-y)==abs( x: p-x)){
            return false;
        }
    }
    return true;
}
```

```
// 解决 N 皇后问题的函数
void resolve(){
    int count = 1; // 计数器，用于给每个结点编号
    int x,y;
    NodeType n1,n2;
    std::priority_queue<NodeType> qu; // 定义一个队列用于存储结点

    // 初始化第一个结点
    n1.number = count++; // 给结点编号，计数器加 1
    n1.row = -1; // 第一个皇后未确定行，初始化为 -1
    qu.push( x: n1); // 将第一个结点放入队列中
    std::cout<<"进队：  ";
    shownode( n: n1);
```

```
    while(!qu.empty()){ // 循环直到队列为空
        n1 = qu.top(); // 取出优先级最高的节点（即 row 值最小的节点）
        qu.pop(); // 弹出队头结点
        std::cout<<"出队： ";
        shownode( n: n1);
        if(n1.row==NofQueen-1){ // 如果已经放置了 NofQueen 个皇后，得到一组解
            std::cout<<"产生一个解为： ";
            for(x=0;x<NofQueen;x++){
                std::cout<<"["<<x+1<<","<<n1.column[x]+1<<"] "; // 输出每个皇后所在的位置
            }
            std::cout<<std::endl;
            return ; // 返回
        }
        else{
            for(y=0;y<NofQueen;y++){ // 在下一行中尝试放置皇后
                x=n1.row+1; // 下一行的行号
                if(check(n1.column,x,y)){ // 检查是否可以放置皇后
                    n2.number = count++; // 给结点编号，计数器加 1
                    n2.row = x; // 更新皇后所在行
                    n2.column = n1.column; // 复制上一行的列号
                    n2.column.push_back(y); // 在下一行中放置皇后
                    qu.push( x: n2); // 将新结点放入队列中
                    std::cout<<"  进队子结点： ";
                    shownode( n: n2);
                }
            }
        }
    }
}
```

```
// 主函数
int main(){
    std::cout<<"2.6.1 "<<std::endl<<NofQueen<<"皇后问题求解过程为："<<std::endl;
    resolve(); // 解决 N 皇后问题并输出求解过程和最终结果
    return 0;
}
```

运行结果：

（1）队列式分枝限界法：

```
2.6.1
4皇后问题求解过程为：
进队：  编号：1  ,对应位置(-1,*)
出队：  编号：1  ,对应位置(-1,*)
  进队子结点：  编号：2  ,对应位置(0,0)
  进队子结点：  编号：3  ,对应位置(0,1)
  进队子结点：  编号：4  ,对应位置(0,2)
  进队子结点：  编号：5  ,对应位置(0,3)
出队：  编号：2  ,对应位置(0,0)
  进队子结点：  编号：6  ,对应位置(1,2)
  进队子结点：  编号：7  ,对应位置(1,3)
出队：  编号：3  ,对应位置(0,1)
  进队子结点：  编号：8  ,对应位置(1,3)
出队：  编号：4  ,对应位置(0,2)
  进队子结点：  编号：9  ,对应位置(1,0)
出队：  编号：5  ,对应位置(0,3)
  进队子结点：  编号：10  ,对应位置(1,0)
  进队子结点：  编号：11  ,对应位置(1,1)
出队：  编号：6  ,对应位置(1,2)
出队：  编号：7  ,对应位置(1,3)
  进队子结点：  编号：12  ,对应位置(2,1)
出队：  编号：8  ,对应位置(1,3)
  进队子结点：  编号：13  ,对应位置(2,0)
出队：  编号：9  ,对应位置(1,0)
  进队子结点：  编号：14  ,对应位置(2,3)
出队：  编号：10  ,对应位置(1,0)
  进队子结点：  编号：15  ,对应位置(2,2)
出队：  编号：11  ,对应位置(1,1)
出队：  编号：12  ,对应位置(2,1)
出队：  编号：13  ,对应位置(2,0)
  进队子结点：  编号：16  ,对应位置(3,2)
出队：  编号：14  ,对应位置(2,3)
  进队子结点：  编号：17  ,对应位置(3,1)
出队：  编号：15  ,对应位置(2,2)
出队：  编号：16  ,对应位置(3,2)
产生一个解为：  [1,2] [2,4] [3,1] [4,3]

进程已结束,退出代码0
```

（2）优先队列式分枝限界法：

```
2.6.1
4皇后问题求解过程为：
进队：  编号：1  ,对应位置(-1,*)
出队：  编号：1  ,对应位置(-1,*)
  进队子结点：  编号：2  ,对应位置(0,0)
  进队子结点：  编号：3  ,对应位置(0,1)
  进队子结点：  编号：4  ,对应位置(0,2)
  进队子结点：  编号：5  ,对应位置(0,3)
出队：  编号：2  ,对应位置(0,0)
  进队子结点：  编号：6  ,对应位置(1,2)
  进队子结点：  编号：7  ,对应位置(1,3)
出队：  编号：6  ,对应位置(1,2)
出队：  编号：7  ,对应位置(1,3)
  进队子结点：  编号：8  ,对应位置(2,1)
出队：  编号：8  ,对应位置(2,1)
出队：  编号：4  ,对应位置(0,2)
  进队子结点：  编号：9  ,对应位置(1,0)
出队：  编号：9  ,对应位置(1,0)
  进队子结点：  编号：10  ,对应位置(2,3)
出队：  编号：10  ,对应位置(2,3)
  进队子结点：  编号：11  ,对应位置(3,1)
出队：  编号：11  ,对应位置(3,1)
产生一个解为：  [1,3] [2,1] [3,4] [4,2]

进程已结束,退出代码0
```

对比结果：

队列式分枝限界法是一种广度优先搜索算法，它生成所有可能的解，并将它们按照深度顺序放入一个队列中。该算法会尝试每种可能性，直到找到一个解或者队列为空为止。这种算法可以保证找到最优解，但是当问题规模增大时，其时间复杂度会指数级增长，因此它不适合解决较大规模的问题。

相比之下，优先队列式分枝界限法使用了一个优先队列来保存未扩展的节点，其中优先级由节点的界限函数确定。每次选择具有最小界限函数值的节点进行扩展，以此加速搜索过程。这种算法可以避免探索与当前最优解无关的节点，从而减少搜索空间，并在更短的时间内找到较优解。但是，这种算法并不能保证找到全局最优解。

而在本题中，从结果可以看出,采用队列式分枝限界法求解 4 皇后问题的一个解,生成的队列结点个数为 16 个,而采用优先队列式时生成的队列结点个数为 11 个,明显采用优先队列式分枝限界法更为高效。

因为 4 皇后问题有多个解，所以虽然本次尝试中，两种方法得到的解不同,但都满足题目要求。

五、实验结果及分析

结果都已对应显示在原始数据记录中，结果都与预期的分析符合。