分析报告

任务拆解表

分类	任务名称	起止时间	详情	难点	改进想法
分析	学习策略及函数要求	0:00:00~0:02:26	学习雨课堂上对两种策略的描述及函数的输入输出要求	对策略和函数要求的理解	无
分析	函数一逻辑梳理	0:02:27~0:06:02	通过文字描述函数一实现对应功能的逻辑，有助于编码	无	无
分析	函数二逻辑梳理	0:06:03~0:07:59	通过文字描述函数二实现对应功能的逻辑，有助于编码	无	无
编码	编写函数一	0:08:00~0:16:07	根据逻辑梳理文本编写函数一，实现其既定功能	编写时会出现逻辑错误,较难发现	借助 copilot 辅助修改
编码	编写函数二	0:16:08~0:18:46	根据逻辑梳理文本编写函数二，实现其既定功能	编写时会出现逻辑错误,较难发现	借助 copilot 辅助修改
验证	设计测试样例	0:18:47~0:23:36	将两个数独通过矩阵的形式写入程序	无	无
编码	编写主函数	0:23:47~0:27:51	编写程序分别把两个测试样例交给函数一和函数二处理	需要调整输出格式以满足规范美观要求	可以通过 ai 工具辅助，对输出格式的代码进行指导
验证	测试	0:27:51~0:29:07	运行程序查看输出结果是否满足要求	无	无
调试	调试	0:29:07~0:32:26	发现结果不符合预期，通过在各个位置添加语句进行测试	有时需要花费较长时间才能找到问题	通过 ai 辅助提示在哪些地方设置断点进行调试

开发任务具体分析

一、分析阶段

该阶段主要包含对数独策略和函数要求进行学习及用文本梳理函数一和函数二的编写思路。

（1）对数独策略和函数要求进行学习

任务要求

- 学习两种数独策略的基本思路
- 了解两个函数的输入输出及实现要求

具体实现

- 两种数独的策略可以总结为：从已知的值出发确定未知值；从未知值的候选集出发确定未知值。
- 两个函数的实现逻辑可以总结为：确定某区域只能某个单元格填写某值，即某区域只有一个单元格候选集有该值；确定某单元格只能填写某值，即通过排除该空单元格所在行、列和九宫格已有值，当该空单元格的候选集只有一个数时，可以确认该单元格。

（2）函数一逻辑梳理

任务要求

- 梳理函数一的输入输出及功能

具体实现

- 通过文本描述，明确函数一的具体实现逻辑，为后续的编写作准备

（3）函数二逻辑梳理

基本与（2）相同，此处不再赘述

二、编码阶段

该阶段主要是函数一、函数二和主函数的编写。

（1）编写函数一

任务要求

- 正确实现 lastRemainingCellInference 函数的功能

具体实现

- 声明答案矩阵，并进行初始化处理
- 通过行、列和九宫格已有的数对各个位置的候选集进行处理

• 实现函数一主要功能，通过分析区域候选集确定单元格须填写的数

（2）编写函数二

任务要求

• 正确实现 possibleNumberInference 函数的功能

具体实现

• 声明答案矩阵，并进行初始化处理
• 通过行、列和九宫格已有的数对各个位置的候选集进行处理
•实现函数二主要功能，通过分析每个空单元格的候选集确定单元格须填写的数

（3）编写主函数

任务要求

• 编写主函数进行样例的测试

具体实现

• 将样例作为形参送入函数并获得输出
• 编写输出代码输出样例的结果

三、验证阶段

（1）设计测试样例

任务要求

• 设计不同的测试样例对函数功能进行测试

具体实现

• 将两个数独转化成矩阵形式

（2）测试

任务要求

• 对两个函数的功能进行测试

具体实现

• 检查两个函数的输出是否符合预期
• 若不符合预期，进入调试阶段

四、调试阶段

（1）调试

任务要求

• 通过调试找出函数的问题

具体实现

- 分析结果不符合预期可能的原因
- 在各处设置断点进行错误排查
- 排查出错误后再次验证函数输出

可优化的点

（1）可以再细分任务，减小任务颗粒度。

（2）函数一与函数二对答案数组的部分处理相同，可以抽象出一个新的函数进行简化。

（3）调试不熟练，不能娴熟地通过设置断点排除错误，应进一步强化自身 debug 技能。

代码段展示

```
#include<iostream>
#include<vector>
#include<unordered_set>
using namespace std;

vector<vector<unordered_set<int>>> lastRemainingCellInference (vector<vector<int>>& cells) //确定某个区域(行、列、九宫格)只有某格可以填入某数字
{
    vector<vector<unordered_set<int>>>
ans(9,vector<unordered_set<int>>(9,unordered_set<int>{1,2,3,4,5,6,7,8,9}));
    for(int i=0;i<9;i++)//初始化
    {
```

```
    for(int j=0;j<9;j++)
    {
        if(cells[i][j]!=0)
            ans[i][j].clear(); //如果该格已经填入数字，则该格的集合清空
    }
}
int change_flag=1;//标记答案数组是否发生更改
while(change_flag)
{
    change_flag=0;
    for(int i=0;i<9;i++)
    {
        for(int j=0;j<9;j++)
        {
            for(int k=0;k<9;k++)//从[i][j]中删除行有的数字
            {
                if(k!=j&&cells[i][k]!=0)
                {
                    if(ans[i][j].erase(cells[i][k]))
                    {
                        change_flag=1;
                    }
                }
            }
            for(int k=0;k<9;k++)//从[i][j]中删除列有的数字
            {
                if(k!=i&&cells[k][j]!=0)
                {
                    if(ans[i][j].erase(cells[k][j]))
                    {
                        change_flag=1;
                    }
                }
            }
            for(int k=0;k<9;k++)//从[i][j]中删除九宫格有的数字
            {
                int x=i/3*3+k/3;
                int y=j/3*3+k%3;
                if((x!=i||y!=j)&&cells[x][y]!=0)
                {
                    if(ans[i][j].erase(cells[x][y]))
                    {
                        change_flag=1;
                    }
```

```
    }
}

//若该行只有该单元格能填该数字，则确认该数字位置
for(int k=1;k<=9;k++)
{
    int count=0;
    for(int m=0;m<9;m++)
    {
        if(m!=j&&ans[i][m].find(k)!=ans[i][m].end())
        {
            count++;
        }
    }
    if(count==0&&ans[i][j].find(k)!=ans[i][j].end())
    {
        cells[i][j]=k;
        ans[i][j].clear();
        change_flag=1;
        break;
    }
    //若该列只有该单元格能填该数字，则确认该数字位置
    count=0;
    for(int m=0;m<9;m++)
    {
        if(m!=i&&ans[m][j].find(k)!=ans[m][j].end())
        {
            count++;
        }
    }
    if(count==0&&ans[i][j].find(k)!=ans[i][j].end())
    {
        cells[i][j]=k;
        ans[i][j].clear();
        change_flag=1;
        break;
    }
    //若该九宫格只有该单元格能填该数字，则确认该数字位置
    count=0;
    for(int m=0;m<9;m++)
    {
        int x=i/3*3+m/3;
        int y=j/3*3+m%3;
        if((x!=i||y!=j)&&ans[x][y].find(k)!=ans[x][y].end())
```

```
                    {
                        count++;
                    }
                }
                if(count==0&&ans[i][j].find(k)!=ans[i][j].end())
                {
                    cells[i][j]=k;
                    ans[i][j].clear();
                    change_flag=1;
                    break;
                }
            }
        }
    }
    return ans;
}
```

```
vector<vector<unordered_set<int>>> possibleNumberInference (vector<vector<int>>& cells) //确定某个单元格只能填那个数字
{
    vector<vector<unordered_set<int>>>
ans(9,vector<unordered_set<int>>(9,unordered_set<int>{1,2,3,4,5,6,7,8,9}));
    for(int i=0;i<9;i++)//初始化
    {
        for(int j=0;j<9;j++)
        {
            if(cells[i][j]!=0)
                ans[i][j].clear(); //如果该格已经填入数字，则该格的集合清空
        }
    }
    int change_flag=1;//标记答案数组是否发生更改
    while(change_flag)
    {
        change_flag=0;
        for(int i=0;i<9;i++)
        {
            for(int j=0;j<9;j++)
            {
                for(int k=0;k<9;k++)//从[i][j]中删除行有的数字
                {
                    if(k!=j&&cells[i][k]!=0)
                    {
                        if(ans[i][j].erase(cells[i][k]))
```

```
                    {
                        change_flag=1;
                    }
                }
            }
            for(int k=0;k<9;k++)//从[i][j]中删除列有的数字
            {
                if(k!=i&&cells[k][j]!=0)
                {
                    if(ans[i][j].erase(cells[k][j]))
                    {
                        change_flag=1;
                    }
                }
            }
            for(int k=0;k<9;k++)//从[i][j]中删除九宫格有的数字
            {
                int x=i/3*3+k/3;
                int y=j/3*3+k%3;
                if(x!=i&&y!=j&&cells[x][y]!=0)
                {
                    if(ans[i][j].erase(cells[x][y]))
                    {
                        change_flag=1;
                    }
                }
            }
        }
    }
    //寻找有没有单元格只有一个数字可以填入
    for(int i=0;i<9;i++)
    {
        for(int j=0;j<9;j++)
        {
            if(cells[i][j]==0&&ans[i][j].size()==1)
            {
                cells[i][j]=*ans[i][j].begin();//填入该数字
                ans[i][j].clear();//清空该单元格的集合
                change_flag=1;
            }
        }
    }
}
return ans;
```

```
}
```

```
int main()//设计测试样例
{
    vector<vector<int>> cells1(9,vector<int>(9,0));
    cells1={{2,0,0,0,7,0,0,3,8},{0,0,0,0,0,6,0,7,0},{3,0,0,0,4,0,6,0,0},
            {0,0,8,0,2,0,7,0,0},{1,0,0,0,0,0,0,0,6},{0,0,7,0,3,0,4,0,0},
            {0,0,4,0,8,0,0,0,9},{0,6,0,4,0,0,0,0,0},{9,1,0,0,6,0,0,0,2}};
    vector<vector<int>> cells2(9,vector<int>(9,0));
    cells2={{0,7,0,0,0,8,0,2,9},{0,0,2,0,0,0,0,0,4},{8,5,4,0,2,0,0,0,0},
            {0,0,8,3,7,4,2,0,0},{0,0,0,0,0,0,0,0,0},{0,0,3,2,6,1,7,0,0},
            {0,0,0,0,9,0,6,1,2},{2,0,0,0,0,0,4,0,0},{1,3,0,6,0,0,0,7,0}};
    vector<vector<unordered_set<int>>> ans1=lastRemainingCellInference(cells1);
    vector<vector<unordered_set<int>>> ans2=lastRemainingCellInference(cells2);
    vector<vector<unordered_set<int>>> ans3=possibleNumberInference(cells1);
    vector<vector<unordered_set<int>>> ans4=possibleNumberInference(cells2);
    for(int i=0;i<9;i++)
    {
        for(int j=0;j<9;j++)
        {
            if(cells1[i][j]==0)
            {
                cout<<"{";
                for (auto item : ans1[i][j])
                {
                    cout<<item<<" ";
                }
                cout<<"} ";
            }
            else
            {
                cout<<cells1[i][j]<<" ";
            }
        }
        cout<<endl;
    }
}
```