数据结构与算法实验报告

实验一 背包问题的求解

1.问题描述

假设有一个能装入总体积为 T 的背包和 n 件体积分别为 w1,w2,…wn 的物品，能否从 n 件物品中挑选若干件恰好装满背包，即使 w1+w2+…+wm=T，要求找出所有满足上述条件的解。

例如：当 T=10，各件物品的体积{1,8,4,3,5,2}时，可找到下列 4 组解：

(1,4,3,2)

(1,4,5)

(8,2)

(3,5,2)。

2.实现提示

可利用回溯法的设计思想来解决背包问题。首先，将物品排成一列，然后，顺序选取物品装入背包，若已选取第 i 件物品后未满，则继续选取第 i+1 件，若该件物品“太大”不能装入，则弃之，继续选取下一件，直至背包装满为止。

如果在剩余的物品中找不到合适的物品以填满背包，则说明“刚刚”装入的物品“不合适”，应将它取出“弃之一边”，继续再从“它之后”的物品中选取，如此重复，直到求得满足条件的解，或者无解。

由于回溯求解的规则是“后进先出”，自然要用到“栈”。

进一步考虑：如果每件物品都有体积和价值，背包又有大小限制，求解背包中存放物品总价值最大的问题解---最优解或近似最优解。

3. 实现思路

采用深搜实现枚举每个物品选择的状态（选/未选），到最底层判定体积之和是否等于 T。用体积过大或过小经行剪枝优化。

4. 实验代码

```
#include<iostream>
#include<stack>
#define max 100
using namespace std;
int main()
{
    int V,n,v[max];
    bool found = false;
    cout << "请输入背包大小：";
    cin >> V;
    cout << "请输入物品个数：";
    cin >> n;
    cout << "请输入每个物品的大小：";
    for (int i = 0; i < n; i++)
    {
        cin >> v[i];
    }
```

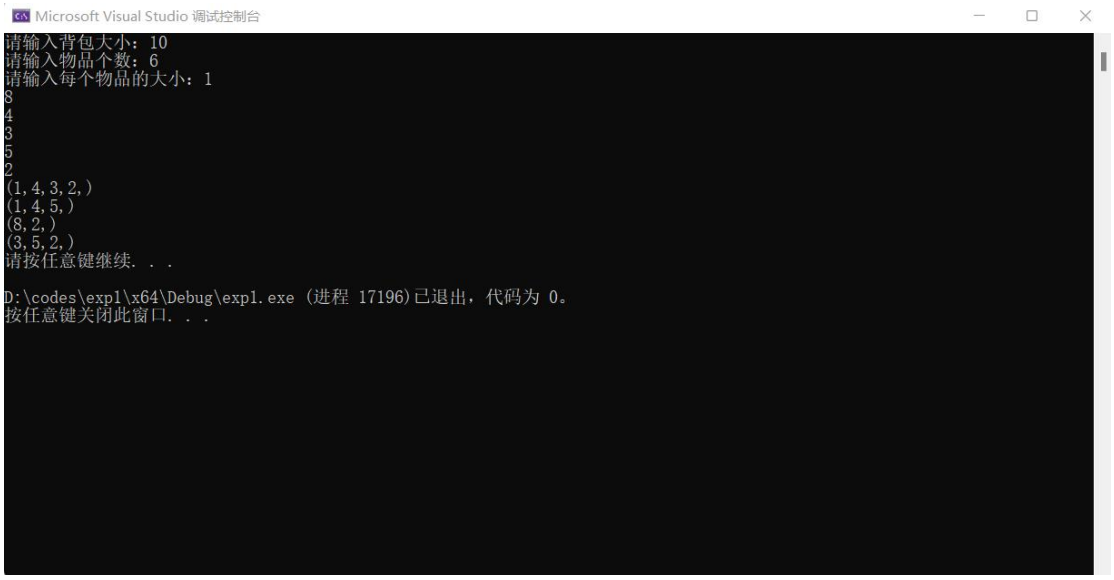
```
int rem = V,i=0;
stack<int> s;
while (1)
{
    if (rem == v[i] || (rem > v[i] && i < n)) {
        s.push(i);
        rem -= v[i];
    }
    if (rem == 0) {
        found = true;

        stack<int> p;
        while (!s.empty())
        {
            p.push(s.top());
            s.pop();
        }
        cout << "(";
        while (!p.empty())
        {
            cout << v[p.top()] << ",";
            s.push(p.top());
            p.pop();
        }
        cout << ")" << endl;

        if (s.size() == 1) {
            rem += v[s.top()];
            s.pop();
            i++;
            if (i == n) break;
        }
        else if (i == n - 1)
        {
            rem += v[s.top()];
            s.pop();
            i = s.top() + 1;
            rem += v[s.top()];
            s.pop();
        }
        else {
            i = s.top() + 1;
            rem += v[s.top()];
```

```
                s.pop();
            }
        }
        else {
            if (i == n - 1) {
                if (s.size() == 1 && s.top() == n - 1) {
                    break;
                }
                else {
                    if (i == s.top()) {
                        rem += v[s.top()];
                        s.pop();
                        i = s.top() + 1;
                        rem += v[s.top()];
                        s.pop();
                    }
                    else {
                        i = s.top() + 1;
                        rem += v[s.top()];
                        s.pop();
                    }
                }
            }
            else {
                i++;
            }
        }
    }
    if (found == false) cout << "无法装满背包" << endl;
    system("pause");
    return 0;
}
```

5. 实验结果



实验二 农夫过河问题的求解

1.问题描述

一个农夫带着一只狼、一只羊和一棵白菜，身处河的南岸。他要把这些东西全部运到北岸。他面前只有一条小船，船只能容下他和一件物品，另外只有农夫才能撑船。如果农夫在场，则狼不能吃羊，羊不能吃白菜，否则狼会吃羊，羊会吃白菜，所以农夫不能留下羊和白菜自己离开，也不能留下狼和羊自己离开，而狼不吃白菜。请求出农夫将所有的东西运过河的方案。

2.实现提示

求解这个问题的简单方法是一步一步进行试探，每一步搜索所有可能的选择，对前一步合适的选择后再考虑下一步的各种方案。要模拟农夫过河问题，首先需要对问题中的每个角色的位置进行描述。可用 4 位二进制数顺序分别表示农夫、狼、白菜和羊的位置。用 0 表在南岸，1 表示在北岸。例如，整数 5 (0101)表示农夫和白菜在南岸，而狼和羊在北岸。

现在问题变成：从初始的状态二进制 0000(全部在河的南岸)出发，寻找一种全部由安全状态构成的状态序列，它以二进制 1111(全部到达河的北岸)为最终目标。总状态共 16 种(0000 到 1111)，(或者看成 16 个顶点的有向图)可采用广度优先或深度优先的搜索策略---得到从 0000 到 1111 的安全路径。

以广度优先为例：整数队列---逐层存放下一步可能的安全状态；Visited[16]数组标记该状态是否已访问过，若访问过，则记录前驱状态值---安全路径。

最终的过河方案应用汉字显示出每一步的两岸状态。

3. 实现思路

首先需要选择一个对问题中每个角色的位置进行描述的方法。一个很方便的办法是用四位二进制数顺序分别表示农夫、狼、白菜和羊的位置。例如用 0 表示农夫或者某东西在河的南岸，1 表示在河的北岸。因此可以列举出 16 种情景，其中有 6 种情形是不安全的。从初始状态二进制 0000(全部在河的南岸) 出发，寻找一种全部由安全状态构成的状态序列，它以二进制 1111(全部到达河的北岸) 为最终目标，并且在序列中的每一个状态都可以从前一状态通过农夫划船过河的动作到达。

4. 实验代码

```
#include<iostream>
using namespace std;
const int VertexNum = 16;
typedef struct
{
    int farmer; // 农夫
    int wolf; // 狼
    int sheep; // 羊
    int veget; // 白菜
}Vertex;
typedef struct
{
    int vertexNum;
    Vertex vertex[VertexNum];
    bool Edge[VertexNum][VertexNum];
}AdjGraph;
bool visited[VertexNum] = { false };
int retPath[VertexNum] = { -1 };


int locate(AdjGraph* graph, int farmer, int wolf, int sheep, int veget)
{
    for (int i = 0; i < graph->vertexNum; i++)
    {
        if (graph->vertex[i].farmer == farmer && graph->vertex[i].wolf == wolf &&
graph->vertex[i].sheep == sheep && graph->vertex[i].veget == veget)
            return i;
    }
    return -1;
}


bool isSafe(int farmer, int wolf, int sheep, int veget)
{

    if (farmer != sheep && (wolf == sheep || sheep == veget))    return false;
    else      return true;
}


// 判断状态
bool isConnect(AdjGraph* graph, int i, int j)
{
    int k = 0;
```

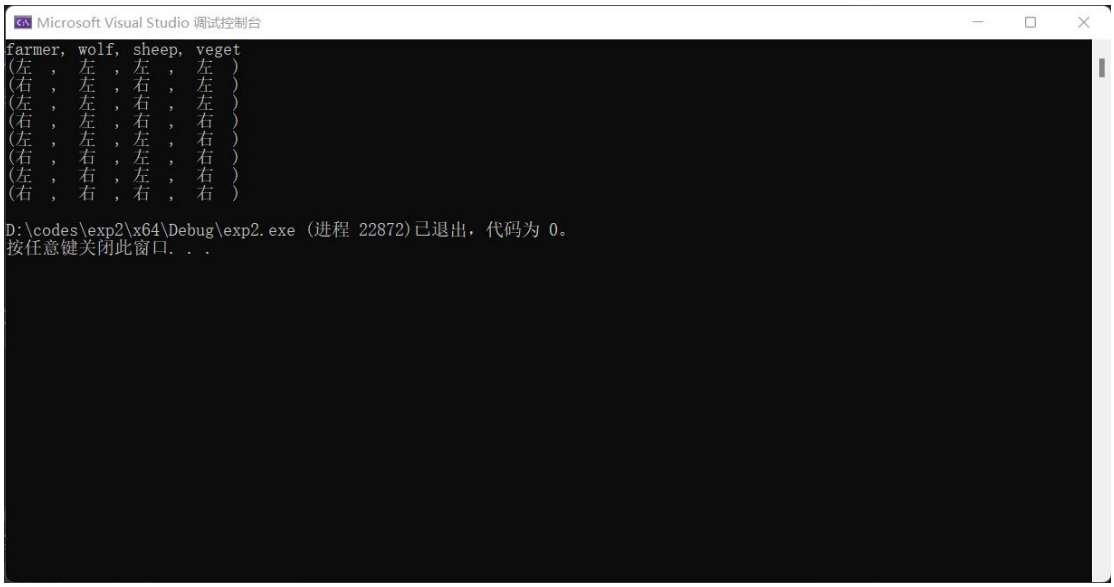
```
    if (graph->vertex[i].wolf != graph->vertex[j].wolf)      k++;
    if (graph->vertex[i].sheep != graph->vertex[j].sheep)    k++;
    if (graph->vertex[i].veget != graph->vertex[j].veget)    k++;
    if (graph->vertex[i].farmer != graph->vertex[j].farmer && k <= 1)     return true;
    else    return false;
}


void CreateG(AdjGraph* graph)
{
    int i = 0; int j = 0;
    for (int farmer = 0; farmer <= 1; farmer++)
    {
        for (int wolf = 0; wolf <= 1; wolf++)
        {
            for (int sheep = 0; sheep <= 1; sheep++)
            {
                for (int veget = 0; veget <= 1; veget++)
                {
                    if (isSafe(farmer, wolf, sheep, veget))
                    {
                        graph->vertex[i].farmer = farmer;
                        graph->vertex[i].wolf = wolf;
                        graph->vertex[i].sheep = sheep;
                        graph->vertex[i].veget = veget;
                        i++;
                    }
                }
            }
        }
    }
    graph->vertexNum = i;
    for (i = 0; i < graph->vertexNum; i++)
    {
        for (j = 0; j < graph->vertexNum; j++)
        {
            if (isConnect(graph, i, j))      graph->Edge[i][j] = graph->Edge[j][i] = true;
            else        graph->Edge[i][j] = graph->Edge[j][i] = false;
        }
    }
    return;
}
string judgement(int state)
{
```

```
    if (state == 0)  return "左  ";
    else return  "右  ";
}
void printPath(AdjGraph* graph, int start, int end)
{
    int i = start;
    cout << "farmer" << ", wolf" << ", sheep" << ", veget" << endl;
    while (i != end) {
        cout << "(" << judgement(graph->vertex[i].farmer) << ",  " <<
            judgement(graph->vertex[i].wolf) << ", " << judgement(graph->vertex[i].sheep)
<< ",  " <<
            judgement(graph->vertex[i].veget) << ")";
        cout << endl;
        i = retPath[i];
    }
    cout << "(" << judgement(graph->vertex[i].farmer) << ",  " <<
        judgement(graph->vertex[i].wolf) << ", " << judgement(graph->vertex[i].sheep) <<
",  " <<
        judgement(graph->vertex[i].veget) << ")";
    cout << endl;
}
// DFS
void dfsPath(AdjGraph* graph, int start, int end)
{
    int i = 0;
    visited[start] = true;
    if (start == end)   return;
    for (i = 0; i < graph->vertexNum; i++)
    {
        if (graph->Edge[start][i] && !visited[i])
        {
            retPath[start] = i;
            dfsPath(graph, i, end);
        }
    }
}
int main()
{
    AdjGraph graph;
    CreateG(&graph);
    int start = locate(&graph, 0, 0, 0, 0);
    int end = locate(&graph, 1, 1, 1, 1);
    dfsPath(&graph, start, end);
    if (visited[end])
```

```
    {
        printPath(&graph, start, end);
        return 0;
    }
    return -1;
}
```

5. 实验结果



实验四　八皇后问题

1.问题描述

设在初始状态下在国际象棋的棋盘上没有任何棋子（这里的棋子指皇后棋子）。然后顺序在第 1 行，第 2 行……第 8 行上布放棋子。在每一行中共有 8 个可选择的位置，但在任一时刻棋盘的合法布局都必须满足 3 个限制条件（1）任意两个棋子不得放在同一行（2）任意两个棋子不得放在同一列上（3）任意棋子不得放在同一正斜线和反斜线上。

2.基本要求

编写求解并输出此问题的一个合法布局的程序。

3、实现提示：

在第 i 行布放棋子时，从第 1 列到第 8 列逐列考察。当在第 i 行第 j 列布放棋子时，需要考察布放棋子后在行方向、列方向、正斜线和反斜线方向上的布局状态是否合法，若该棋子布放合法，再递归求解在第 i+1 行布放棋子；若该棋子布放不合法，移去这个棋子，恢复布放该棋子前的状态，然后再试探在第 i 行第 j+1 列布放棋子。

4. 实验思路

我们一行一行的尝试在某个位置放置一个皇后，然后检查这个位置是否能放皇后，若果不能放就直接选择下一个位置，不让搜索树继续往下发展。 若该位置能放，才继续往下发展放其他的皇后。

判断该位置上是否可以放该皇后：

q[]数组记录某列已经放过了皇后。q[i]= 1 代表第 i 列已经放了一个皇后。

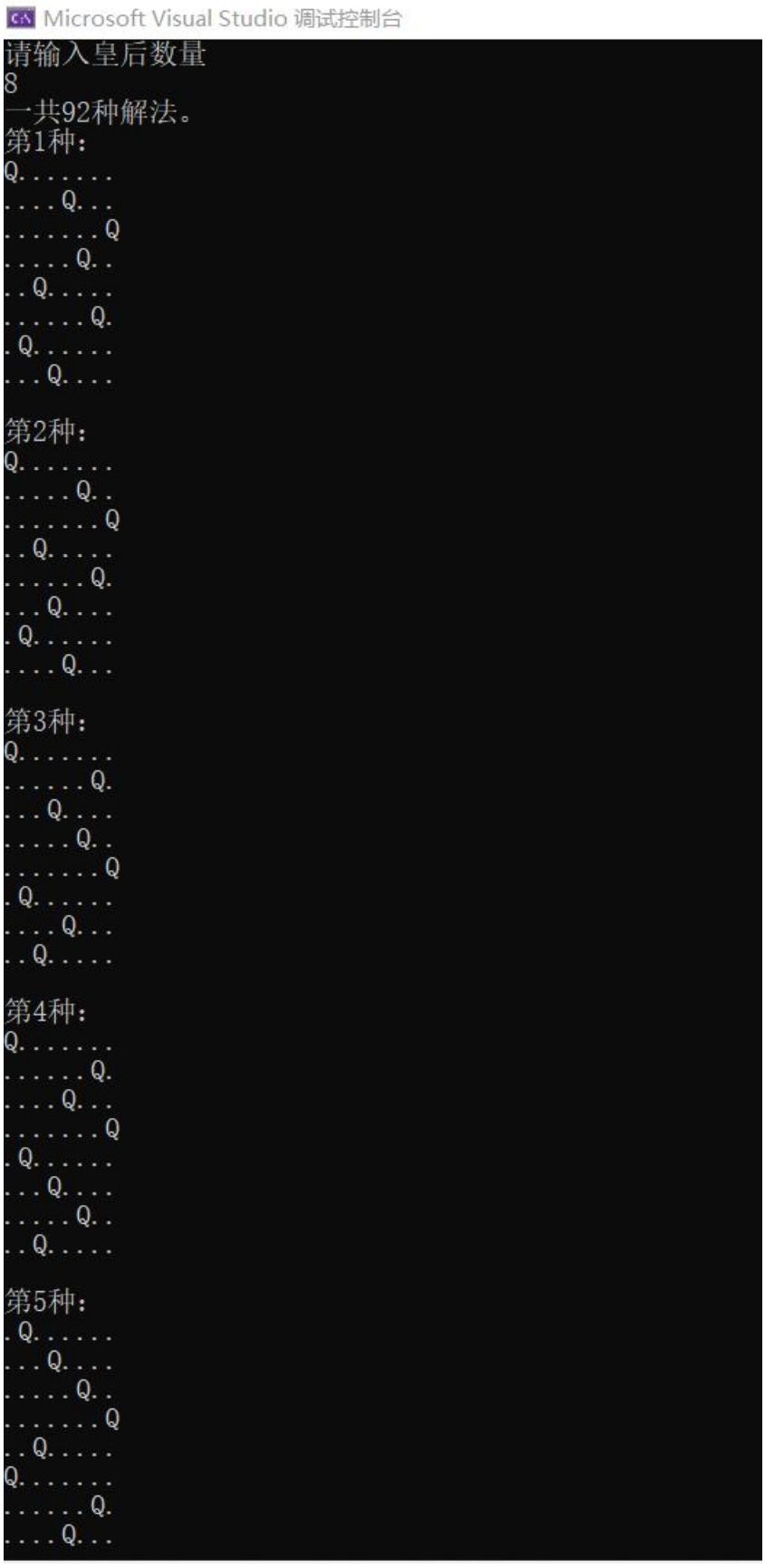
d[] 数组来记录某正对角线已经放置过了皇后。

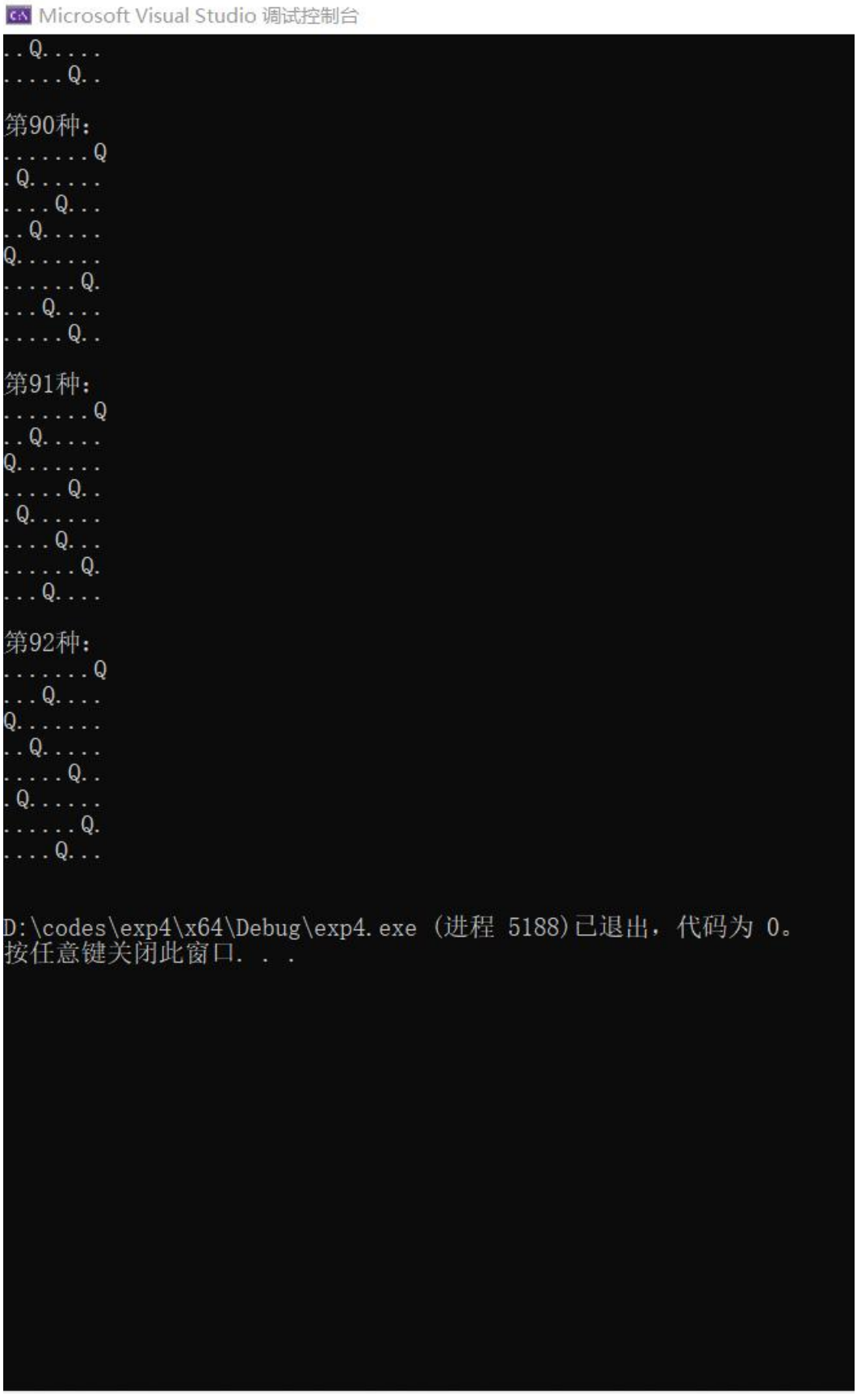
ud[]数组来记录某反对角线已经放置过了皇后。

5. 实验代码

```
#include<iostream>
#include<vector>
using namespace std;
    void dfs(int i, vector<string>& strs, vector<bool>& q, vector<bool>& d, vector<bool>&
ud, vector<vector<string>>& ans, int n) {
        if (i == n) {
            ans.push_back(strs);
            return;
        }
        //选择一个位置进行放置
        for (int j = 0; j < n; j++) {
            if (!q[j] && !d[i + j] && !ud[n - i + j]) {
                q[j] = d[i + j] = ud[n - i + j] = true;
                strs[i][j] = 'Q';
                dfs(i + 1, strs, q, d, ud, ans, n);
                strs[i][j] = '.';
                q[j] = d[i + j] = ud[n - i + j] = false;
            }
        }
    }
    vector<vector<string>> solveNQueens(int n) {
        vector<bool> cols(n), d(25), ud(25);
        vector<vector<string>> ans;
        vector<string> strs(n, string(n, '.'));
        dfs(0, strs, cols, d, ud, ans, n);
        return ans;
    }
int main() {
    int n;
    cout << "请输入皇后数量" << endl;
    cin >> n;
    vector<vector<string>> solve=solveNQueens(n);
    cout << "一共" << solve.size() << "种解法。" << endl;
    for (int i = 0; i < solve.size(); i++)
    {
        cout << "第" << i + 1 << "种：" << endl;
        for (int j = 0; j < solve[i].size(); j++) {
            cout << solve[i][j].c_str() << endl;
        }
        cout << endl;
    }
    return 0;
}
```

6. 实验结果


Microsoft Visual Studio 调试控制台
请输入皇后数量
8
一共92种解法。
第1种：
Q.......
....Q...
.......Q
.....Q..
..Q.....
......Q.
.Q......
...Q....
第2种：
Q.......
.....Q..
.......Q
..Q.....
......Q.
...Q....
.Q......
....Q...
第3种：
Q.......
......Q.
...Q....
.....Q..
.......Q
.Q......
....Q...
..Q.....
第4种：
Q.......
......Q.
....Q...
.......Q
.Q......
...Q....
.....Q..
..Q.....
第5种：
.Q......
...Q....
.....Q..
.......Q
..Q.....
Q.......
......Q.
....Q...


Microsoft Visual Studio 调试控制台
..Q.....
.....Q..
第90种：
.......Q
.Q......
....Q...
..Q.....
Q.......
......Q.
...Q....
.....Q..
第91种：
.......Q
..Q.....
Q.......
.....Q..
.Q......
....Q...
......Q.
...Q....
第92种：
.......Q
...Q....
Q.......
..Q.....
.....Q..
.Q......
......Q.
....Q...
D:\codes\exp4\x64\Debug\exp4.exe (进程 5188)已退出，代码为 0。
按任意键关闭此窗口. . .

实验六 教学计划编制问题

1.问题描述

大学的每个专业都要制定教学计划。假设任何专业都有固定的学习年限，每学年含两学期，每学期的时间长度和学分上限值均相等。每个专业开设的课程都是固定的，而且课程在开设时间的安排必须满足先修关系。每门课程有哪些先修课程是确定的，可以有任意多门，也可以没有。每门课恰好占一个学期。试在这样的前提下设计一个教学计划编制程序。

2.基本要求

(1)输入参数包括：学期总数，一学期的学分上限，每门课的课程号(固定占 3 位的字母数字串)、学分和直接先修课的课程号。

(2)允许用户指定下列两种编排策略之一：一是使学生在各学期中的学习负担尽量均匀；二是使课程尽可能地集中在前几个学期中。

(3)若根据给定的条件问题无解，则报告适当的信息；否则，将教学计划输出到用户指定的文件中。计划的表格格式自行设计。

3、实现提示：

可设学期总数不超过 12，课程总数小于 100。如果输入的先修课程号不在该专业开设的课程序列中，则作为错误处理。

4.测试数据

学期总数：6；学分上限：10；该专业共开设 12 门课，课程号从 C01 到 C12，学分顺序为 2，3，4，3，2，3，4，4，7，5，2，3。先修关系见下图。

5.实验代码

```cpp
#include<iostream>
#include<map>
#include<vector>
#include<stack>
#pragma warning(disable:4996)
using namespace std;

struct node
{
    char name[33];
    int xf;
};

int all_terms, t_max_xf, t_V, t_E;
vector<node> G[1009];
map<string, int> mp;

void create_graph()
{
    int i;
    printf("\t\t\txhw-教学编制-exp6\n");
    printf("输入学期总数: ");
    scanf("%d", &all_terms);
    printf("请输入学期的学分上限: ");
    scanf("%d", &t_max_xf);
    printf("请输入教学计划的课程数: ");
    scanf("%d", &t_V);
    printf("请输入各个课程的先修课程的总和(边总数): ");
    scanf("%d", &t_E);
    printf("请输入%d 个课程的课程号(最多 30 个字符,小写字母 c+数字如 c10)\n", t_V);
    node data;
    for (i = 1; i <= t_V; i++)
    {
        printf("请输入第%d 个: ", i);
        scanf(" %s", data.name);
        G[i].push_back(data);
        mp[G[i][0].name] = i;
    }
    printf("请输入%d 个课程分别对应的学分值:\n", t_V);
    for (i = 1; i <= t_V; i++) scanf("%d", &G[i][0].xf);
    printf("请输入下列课程的先修课程(输入以#结束)\n");
    char s[33];
    for (i = 1; i <= t_V; i++)
    {
```

```
        printf(" %s 的先修课程: ", G[i][0].name);
        while (true)
        {
            scanf(" %s", s);
            if (s[0] == '#') break;
            G[i].push_back(G[mp[s]][0]);
        }
    }
    printf("\t\t\t 录入数据成功\n");
}

void display()
{
    int i, j;
    printf("有向图\n");
    printf("%d 个顶点", t_V);
    for (i = 1; i <= t_V; ++i) printf("%s%c", G[i][0].name, i == t_V ? '\n' : ' ');
    printf("%d 条弧边:\n", t_E);
    for (i = 1; i <= t_V; i++)
    {
        int k = G[i].size();
        for (j = 0; j < k; j++)
            printf("%s---->%s\n", G[i][0].name, G[i][j].name);
    }
}

void solve1(int ans[])
{
    int q = 1, cnt = 0;
    while (q <= all_terms)
    {
        int num = t_V / all_terms;
        printf("\n 第%d 个学期应学课程: ", q);
        while (num--)
        {
            printf("%s%c", G[ans[cnt++]][0].name, num != 0 ? ' ' : '\n');
        }
        if (q == all_terms) printf("OK Over!\n");
        q++;
    }
}

void solve2(int ans[])
{
```

```
    int q = 1, cnt = 0;
    while (q <= all_terms)
    {
        int C = G[ans[cnt]][0].xf;
        printf("\n 第%d 个学期应学课程: ", q);
        while (cnt < t_V && C <= t_max_xf)
        {
            printf("%s ", G[ans[cnt]][0].name);
            if (cnt + 1 < t_V) C = C + G[ans[cnt + 1]][0].xf;
            cnt++;
        }
        if (cnt >= t_V || q == all_terms)
        {
            cout << endl;
            printf("OK Over!\n");
            break;
        }
        q++;
    }
}

void topo_sort()
{
    int i, j, vis[1009];
    memset(vis, 0, sizeof(vis));
    for (i = 1; i <= t_V; i++)
    {
        int k = G[i].size();
        for (j = 1; j < k; j++)
            vis[mp[G[i][j].name]]++;
    }
    int ans[1009], cnt = 0;
    memset(ans, 0, sizeof(ans));
    stack<int> s;
    for (i = 1; i <= t_V; i++)
    {
        if (!vis[i]) s.push(i);
    }
    while (!s.empty())
    {
        int cur = s.top(); s.pop();
        ans[cnt++] = cur;
        int k = G[cur].size();
        for (j = 1; j < k; j++)
```

```
        {
            int num = mp[G[cur][j].name];
            vis[num]--;
            if (!vis[num]) s.push(num);
        }
    }
    if (cnt != t_V) puts("Error!");
    else
    {
        puts("OK!");
        while (true)
        {
            printf("\n\t\t\t 请选择功能:\n");
            printf("\t\t\t1.均匀\n");
            printf("\t\t\t2.集中\n");
            printf("\t\t\t3.退出\n");
            int sel;
            scanf("%d", &sel);
            switch (sel)
            {
            case 1: solve1(ans); break;
            case 2: solve2(ans); break;
            }
            if (sel == 3) break;
        }
    }
}

int main()
{
    create_graph();
    display();
    topo_sort();
    return 0;
}
```

6.实验结果

```
/*
6
10
12
16
c1
c2
c3
```

```
c4
c5
c6
c7
c8
c9
c10
c11
c12
2 3 4 3 2 3 4 4 7 5 2 3
#
c1 #
c1 c2 #
c1 #
c3 #
c11 #
c5 c3 #
c3 c6 #
#
c9 #
c9 #
c9 c10 c1 #
*/
```

D:\codes\exp6\x64\Debug\exp6.exe

```
                    xhw-教学编制-exp6
输入学期总数: 6
请输入学期的学分上限: 10
请输入教学计划的课程数: 12
请输入各个课程的先修课程的总和(边总数): 16
请输入12个课程的课程号(最多30个字符,小写字母c+数字如c10)
请输入第1个: c1
请输入第2个: c2
请输入第3个: c3
请输入第4个: c4
请输入第5个: c5
请输入第6个: c6
请输入第7个: c7
请输入第8个: c8
请输入第9个: c9
请输入第10个: c10
请输入第11个: c11
请输入第12个: c12
请输入12个课程分别对应的学分值:
2
3
4
3
2
3
4
4
7
5
2
3
请输入下列课程的先修课程(输入以#结束)
 c1的先修课程: #
 c2的先修课程: c1 #
 c3的先修课程: c1 c2 #
 c4的先修课程: c1 #
 c5的先修课程: c3 #
 c6的先修课程: c11 #
 c7的先修课程: c5 c3 #
 c8的先修课程: c3 c6 #
 c9的先修课程: #
 c10的先修课程: c9 #
 c11的先修课程: c9 #
 c12的先修课程: c9 c10 c1 #
                    录入数据成功
有向图
12个顶点c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12
16条弧边:
c1---->c1
c2---->c2
c2---->c1
c3---->c3
```

```
D:\codes\exp6\x64\Debug\exp6.exe
c7---->c5
c7---->c3
c8---->c8
c8---->c3
c8---->c6
c9---->c9
c10---->c10
c10---->c9
c11---->c11
c11---->c9
c12---->c12
c12---->c9
c12---->c10
c12---->c1
OK!

                        请选择功能:
                        1.均匀
                        2.集中
                        3.退出
1

第1个学期应学课程: c12 c10

第2个学期应学课程: c8 c6

第3个学期应学课程: c11 c9

第4个学期应学课程: c7 c5

第5个学期应学课程: c3 c2

第6个学期应学课程: c4 c1
OK Over!

                        请选择功能:
                        1.均匀
                        2.集中
                        3.退出
2

第1个学期应学课程: c12 c10
第2个学期应学课程: c8 c6 c11
第3个学期应学课程: c9
第4个学期应学课程: c7 c5 c3
第5个学期应学课程: c2 c4 c1
OK Over!

                        请选择功能:
                        1.均匀
                        2.集中
                        3.退出
```

实验十三、迷宫问题

1、问题描述：

迷宫实验是取自心理学的一个古典实验。在该实验中，把一只老鼠从一个无顶大盒子的门放入，在盒中设置了许多墙，对行进方向形成了多处阻挡。盒子仅有一个出口，在出口处放置一块奶酪，吸引老鼠在迷宫中寻找道路以到达出口。对同一只老鼠重复进行上述实验，一直到老鼠从入口到出口，而不走错一步。老鼠经多次试验终于得到它学习走迷宫的路线。

2、设计功能要求：

迷宫由 m 行 n 列的二维数组设置，0 表示无障碍，1 表示有障碍。设入口为（1，1），出口为（m，n），每次只能从一个无障碍单元移到周围四个方向上任一无障碍单元。编程实现对任意设定的迷宫，求出一条从入口到出口的通路，或得出没有通路的结论。

算法输入：代表迷宫入口的坐标

算法输出：穿过迷宫的结果。 算法要点：创建迷宫，试探法查找路。

3. 实验思路

深搜寻找路径，回溯。

4. 实验代码

```
#include<iostream>
using namespace std;
const int MAXN = 20;
const int MAXM = 20;

const int dx[] = { -1,0,0,1 };
const int dy[] = { 0,-1,1,0 };

int n, m;
char G[MAXN + 5][MAXM + 5];

bool dfs(int x, int y) {
    if (G[x][y] != '0') return false;
    G[x][y] = '*';
    if (x == n && y == m) return true;
    for (int i = 0; i < 4; ++i)
        if (dfs(x + dx[i], y + dy[i])) return true;
    G[x][y] = '0';
    return false;
}

int main() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            cin >> G[i][j];
```

```
    for (int i = 0; i <= n + 1; ++i)
        G[i][0] = G[i][m + 1] = '1';
    for (int i = 0; i <= m + 1; ++i)
        G[0][i] = G[n + 1][i] = '1';
    if (dfs(1, 1)) {
        Cout<<endl;
        cout << "----- * is the way -----" << endl;
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= m; j++)
            {
                cout << G[i][j];
            }
            cout << endl;
        }
    }
    else cout << "无解" << endl;
    return 0;
}
```

5. 实验结果

```
/*
5 5
01000
00010
01100
01001
00100
*/
```