BCS009 数据结构与算法设计　　作业 4　　石雨宸 D23090120503

1. 设有 n 个人站成一排，从左向右的编号分别为 1～n，现在从左往右报数“1，2，1，2，…”，数到“1”的人出列，数到“2”的立即站到队伍的最右端。报数过程反复进行，直到 n 个人都出列为止。要求给出他们的出列顺序。

例如，当 n=8 时，初始序列 1 2 3 4 5 6 7 8；则出列顺序为 1 3 5 7 2 6 4 8

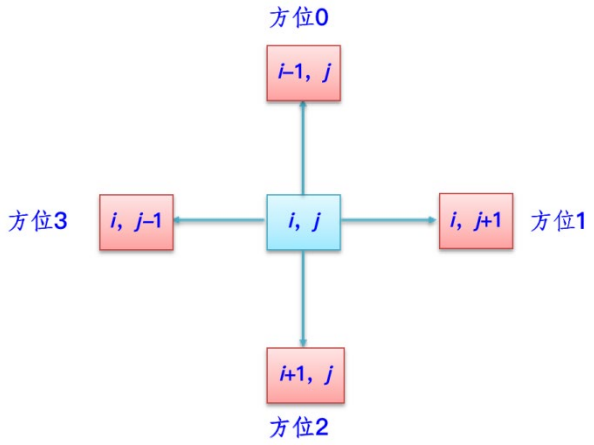
```c
#include <stdio.h>
#include <stdlib.h>

// 定义节点
typedef struct Node
{
    int data;
    struct Node *next;
} Node;

//定义队列
typedef struct Queue
{
    Node *front;
    Node *rear;
} Queue;

// 创建新节点
Node *newNode(int data)
{
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->data = data;
    temp->next = NULL;
    return temp;
}

// 创建新队列
Queue *createQueue()
{
    Queue *q = (Queue *)malloc(sizeof(Queue));
    q->front = q->rear = NULL;
    return q;
```

```
}

// 将节点加入队列
void enqueue(Queue *q, int data)
{
    // 如果队列为空则创建一个front 和rear 都指向的节点
    Node *temp = newNode(data);
    if (q->rear == NULL)
    {
        q->front = q->rear = temp;
        return;
    }
    //更新rear 指针
    q->rear->next = temp;
    q->rear = temp;
}

// 从队列中取出节点
int dequeue(Queue *q)
{
    if (q->front == NULL)
        return -1;

    Node *temp = q->front;
    q->front = q->front->next;

    if (q->front == NULL)
        q->rear = NULL;

    int data = temp->data;
    free(temp);
    return data;
}

// 模拟出列过程
void simulate(int n)
{
    Queue *q = createQueue();
    for (int i = 1; i <= n; i++)
    {
        enqueue(q, i);
    }
```

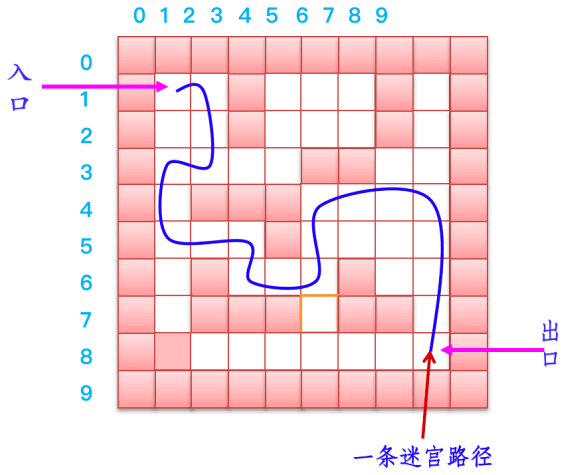
```c
    // 记录当前报数
    int count = 1;
    while (q->front != NULL)
    {
        int person = dequeue(q);
        if (count == 1)
        {
            printf("%d ", person);
            count = 2;
        }
        else
        {
            enqueue(q, person);
            count = 1;
        }
    }
    printf("\n");
}

int main()
{
    int n;
    printf("Enter the number of people: ");//设定队列人数
    scanf("%d", &n);
    simulate(n);
    return 0;
}
```

2. 给定一个 M×N 的迷宫图、入口与出口、行走规则。求一条从指定入口到出口的路径。 所求路径必须是简单路径，即路径不重复。

行走规则：上、下、左、右相邻方块行走。其中（*i*, *j*）表示一个方块



例如，*M*=8，*N*=8，图中的每个方块，用空白表示通道，用阴影表示障碍物。为了算法方便，一般在迷宫外围加上了一条围墙。



2.1 请用栈求解迷宫问题；

```c
#include <stdio.h>
#include <stdlib.h>

// 定义8*8 的迷宫
#define ROWS 8
#define COLS 8

typedef struct
{
    int x, y;
```

```c
} Position;

typedef struct
{
    Position positions[ROWS * COLS];
    int top;
} Stack;
// 初始化栈
void initStack(Stack *stack)
{
    stack->top = -1;
}
// 判断栈是否为空
int isStackEmpty(Stack *stack)
{
    return stack->top == -1;
}
// 入栈
void push(Stack *stack, Position pos)
{
    stack->positions[++stack->top] = pos;
}
// 出栈
Position pop(Stack *stack)
{
    return stack->positions[stack->top--];
}
// 查看栈顶位置
Position peek(Stack *stack)
{
    return stack->positions[stack->top];
}
// 判断是否可以移动
int isValidMove(int maze[ROWS][COLS], int
visited[ROWS][COLS], int x, int y)
{
    return x >= 0 && x < ROWS && y >= 0 && y < COLS &&
maze[x][y] == 0 && !visited[x][y];
}

void printPath(Stack *stack)
{
```

```
    for (int i = 0; i <= stack->top; i++)
    {
        printf("(%d, %d)-> ", stack->positions[i].x,
stack->positions[i].y);
    }
    printf("finish\n");
}

void solveMaze(int maze[ROWS][COLS], Position start,
Position end)
{
    int visited[ROWS][COLS] =
{0};                                // 标记已经访问过的位置
    int directions[4][2] = {{0, 1}, {1, 0}, {0, -1}, {-1,
0}}; // 设置移动方位
    Stack stack;
    initStack(&stack);

    push(&stack, start);
    visited[start.x][start.y] = 1;

    while (!isStackEmpty(&stack))
    {
        // 检查当前路径的位置
        Position current = peek(&stack);

        if (current.x == end.x && current.y == end.y)
        {

            printf("Path: ");
            printPath(&stack);
            return;
        }

        int moved = 0;
        for (int i = 0; i < 4; i++)
        {
            // 计算新的x,y 坐标
            int newX = current.x + directions[i][0];
            int newY = current.y + directions[i][1];
            // 判断是否可以移动
            if (isValidMove(maze, visited, newX, newY))
```

```
            {
                // 将新位置入栈
                push(&stack, (Position){newX, newY});
                visited[newX][newY] = 1;
                moved = 1;
                break;
            }
        }

        if (!moved)
        {
            pop(&stack);
        }
    }

    printf("No path found.\n");
}

int main()
{
    // 定义迷宫，0 表示可以通过，1 表示障碍物
    int maze[ROWS][COLS] = {
        {0, 0, 1, 0, 0, 0, 1, 0},
        {0, 0, 1, 0, 0, 0, 1, 0},
        {0, 0, 0, 0, 1, 1, 0, 0},
        {0, 1, 1, 1, 0, 0, 0, 0},
        {0, 0, 0, 1, 0, 0, 0, 0},
        {0, 1, 0, 0, 0, 1, 0, 0},
        {0, 1, 1, 1, 0, 1, 1, 0},
        {1, 0, 0, 0, 0, 0, 0, 0}};
    // 设置起点和终点
    Position start = {0, 0};
    Position end = {7, 7};

    solveMaze(maze, start, end);

    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  SQL CONSOLE  COMMENTS
Path: (0, 0)-> (0, 1)-> (1, 1)-> (2, 1)-> (2, 0)-> (3, 0)-> (4, 0)-> (4, 1)-> (4, 2)-> (5, 2)-> (5, 3)-> (5, 4)-> (6, 4)-> (7, 4)-> (7
, 5)-> (7, 6)-> (7, 7)-> finish
```

2.2 请问队列求解迷宫问题；

```c
#include <stdio.h>
#include <stdbool.h>

#define N 8
#define M 8

typedef struct
{
    int x, y;
} Point;

typedef struct
{
    Point points[N * M];
    int front, rear; // 设定队首和队尾
} Queue;

// 初始化队列
void initQueue(Queue *q)
{
    q->front = q->rear = 0;
}
// 判断队列是否为空
bool isEmpty(Queue *q)
{
    return q->front == q->rear;
}
// 入队
bool enqueue(Queue *q, Point p)
{
    if ((q->rear + 1) % (N * M) == q->front)
    {
        return false; // Queue is full
    }
    q->points[q->rear] = p;
    q->rear = (q->rear + 1) % (N * N);
    return true;
}
// 出队
```

```
bool dequeue(Queue *q, Point *p)
{
    if (isEmpty(q))
    {
        return false;
    }
    *p = q->points[q->front];
    q->front = (q->front + 1) % (N * M);
    return true;
}
// 检查当前节点在迷宫内且未被访问过
bool isValid(int x, int y, int maze[N][N], bool
visited[N][M])
{
    return (x >= 0 && x < N && y >= 0 && y < M &&
maze[x][y] == 0 && !visited[x][y]);
}

void printPath(Point path[N][M], Point end)
{
    Point current = end;
    printf("Path: ");
    while (path[current.x][current.y].x != -1 &&
path[current.x][current.y].y != -1)
    {
        printf("(%d, %d) <- ", current.x, current.y);
        current = path[current.x][current.y];
    }
    printf("Start\n");
}
// 使用广度优先搜索查找路径
bool bfs(int maze[N][M], Point start, Point end)
{
    int rowNum[] = {-1, 0, 0, 1}; // 行方向，左右
    int colNum[] = {0, -1, 1, 0}; // 列方向，上下

    bool visited[N][M] = {false};
    Point path[N][M];//存储路径的前驱节点

    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
```

```
        {
            path[i][j] = (Point){-1, -1};
        }
    }
    // 设置起点已访问
    visited[start.x][start.y] = true;

    Queue q;
    initQueue(&q);
    enqueue(&q, start);

    while (!isEmpty(&q))
    {
        Point curr;
        dequeue(&q, &curr);

        // 判断是否到达终点，到达则打印路径
        if (curr.x == end.x && curr.y == end.y)
        {
            printPath(path, end);
            return true;
        }
        // 遍历当前节点的四个相邻节点
        for (int i = 0; i < 4; i++)
        {
            int x = curr.x + rowNum[i];
            int y = curr.y + colNum[i];

            if (isValid(x, y, maze, visited))
            {
                visited[x][y] = true;
                path[x][y] = curr;
                Point adj = {x, y};
                enqueue(&q, adj);
            }
        }
    }

    printf("No path found from (%d, %d) to (%d, %d)\n",
start.x, start.y, end.x, end.y);
    return false;
}
```

```cpp
int main()
{
    // 定义迷宫，0 表示可以通过，1 表示障碍物
    int maze[N][M] = {
        {0, 0, 1, 0, 0, 0, 1, 0},
        {0, 0, 1, 0, 0, 0, 1, 0},
        {0, 0, 0, 0, 1, 1, 0, 0},
        {0, 1, 1, 1, 0, 0, 0, 0},
        {0, 0, 0, 1, 0, 0, 0, 0},
        {0, 1, 0, 0, 0, 1, 0, 0},
        {0, 1, 1, 1, 0, 1, 1, 0},
        {1, 0, 0, 0, 0, 0, 0, 0}};

    // 设置起点和终点
    Point start = {0, 0};
    Point end = {7, 7};

    bfs(maze, start, end);

    return 0;
}
```