哈尔滨工业大学计算学部

实验报告

课程名称：数据结构与算法

课程类型：专业核心基础课（必修）

实验项目：图型结构及其应用

实验题目：最短路径算法

实验日期：2024 年 5 月 12 日

班级：22WL022

学号：2022110829

姓名：杨明达

设计成绩	报告成绩	任课老师
		张岩

一、实验目的

最短路径问题研究的主要有：单源最短路径问题和所有顶点对之间的最短路径问题。在计算机领域和实际工程中具有广泛的应用，如集成电路设计、GPS/游戏地图导航、智能交通、路由选择、铺设管线等。本实验要求设计和实现 Dijkstra 算法和 Floyd-Warshall 算法，求解最短路径问题。

二、实验要求及实验环境

实验要求：

1. 实现单源最短路径的 Dijkstra 算法，输出源点及其到其他顶点的最短路径长度和最短路径。

2. 实现全局最短路径的 Floyd-Warshall 算法。计算任意两个顶点间的最短距离矩阵和最短路径矩阵，并输出任意两个顶点间的最短路径长度和最短路径。

3. 利用 Dijkstra 或 Floyd-Warshall 算法解决单目标最短路径问题：找出图中每个顶点 v 到某个指定顶点 c 最短路径。

4. 利用 Dijkstra 或 Floyd-Warshall 算法解决单顶点对间最短路径问题：对于某对顶点 u 和 v，找出 u 到 v 和 v 到 u 的一条最短路径。

5. 以文件形式输入图的顶点和边，并以适当的方式展示相应的结果。要求顶点不少于 10 个，边不少于 15 个（图的规模越大越好）。

实验环境：Windows 11 && Visual Studio Code

三、设计思想（本程序中的用到的所有数据类型的定义，主程序的流程图及各程序模块之间的调用关系、核心算法的主要步骤）

1．逻辑设计

1.1 主程序的流程图

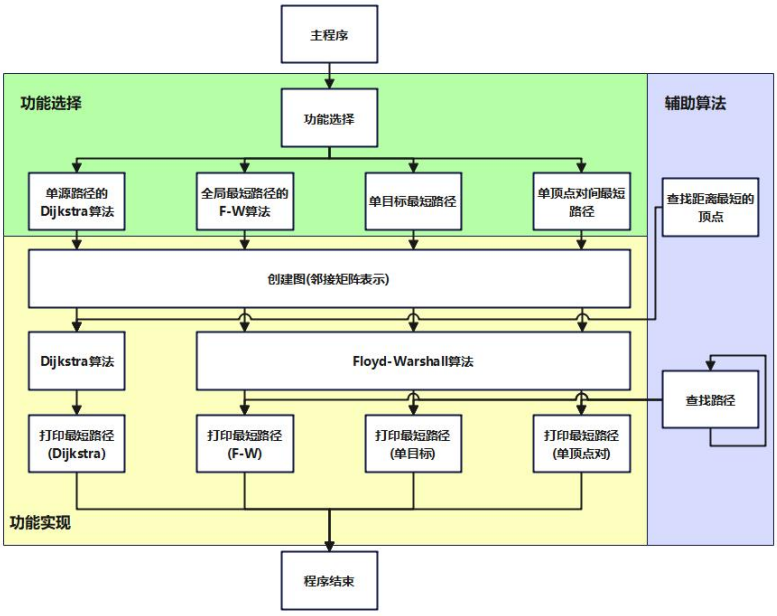

图 1 主程序流程图

1.2 各程序模块的调用关系

（1）主函数调用创建图、Dijstra 算法、Floyd-Warshall 算法、打印最短路径(F-W)、打印最短路径(单目标)、打印最短路径(单顶点对)函数。

（2）从 Dijstra 算法函数中调用查找距离最短顶点的功能函数。

（3）从打印最短路径(F-W)函数中调用查找路径函数。

（4）从打印最短路径(单目标)函数中调用查找路径函数。

（5）从打印最短路径(单顶点对)函数中调用查找路径函数。

（6）从查找路径函数中调用查找路径函数，即递归调用。

1.3 核心算法

（1）创建图（基于邻接矩阵）

在本次实验中，基于邻接矩阵创建图是实现整体功能的基础。在函数中，首先读取 Graph.txt 文件，将顶点数、边数依次读入存储结构中。接着对邻接矩阵初始化，读入顶点信息。最后读取各个边的信息和权值并放入邻接矩阵中。

```
1.  void Create_Graph(MTGraph *G) {
2.      FILE *fp;
3.      fp = fopen("Graph.txt", "r");
4.      fscanf(fp, "%d", &G->n);
5.      fscanf(fp, "%d", &G->e);
6.      for (int i = 0; i < G->n; i++) {
7.          for (int j = 0; j < G->n; j++) {
8.              G->edge[i][j] = inf;
9.          }
10.     }
11.     for (int i = 0; i < G->n; i++) {
12.         G->edge[i][i] = 0;
13.     }
14.     for (int i = 0; i < G->n; i++) {
15.         fscanf(fp, "%d", &G->vertex[i]);
16.     }
17.     for (int i = 0; i < G->e; i++) {
18.         int u, v, w;
19.         fscanf(fp, "%d %d %d", &u, &v, &w);
20.         G->edge[u][v] = w;
21.     }
22.     fclose(fp);
23. }
```

（2）Dijstra 算法

在本次实验中，Dijstra 是查找最短路径的核心算法之一，利用贪心思想。在函数中，首先，对三个矩阵 D、P、S 进行初始化，将 D 赋值为邻接矩阵中对应的边权值，将 P 赋值为指定的源点，将 S 赋值为 0。接着，将 S[source]赋值为 1,然后进入二重循环，实现最短路径的查找。在第一层循环中（循环顶点数），寻找距离整体 S 最短距离的顶点 w，将 w 放入 S 集中，然后进入第二层循环（循环顶点数），检查 S 集内的顶点，将原有距离和引入 w 后的距离进行比较，取二者最小的为最短距离，然后更新 D 最短路径长度和 P 路径中的上一个顶点。

```
1.  void Dijkstra(MTGraph *G, int source, int D[], int P[], int S[]) {
2.      for (int i = 0; i < G->n; i++) {
3.          D[i] = G->edge[source][i];
4.          P[i] = source;
5.          S[i] = 0;
6.      }
7.      S[source] = 1;
8.      for (int i = 0; i < G->n - 1; i++) {
9.          int w = MinCost(G, D, S);
10.         S[w] = 1;
11.         for (int v = 0; v < G->n; v++) {
12.             if (!S[v]) {
13.                 int sum = D[w] + G->edge[w][v];
14.                 if (sum < D[v]) {
15.                     D[v] = sum;
16.                     P[v] = w;
17.                 }
18.             }
19.         }
20.     }
21. }
```

（3）Floyd-Warshall 算法

在本次实验中，Floyd-Warshall 是查找最短路径的核心算法之一，利用动态规划思想。首先，对最短路径长度矩阵和最短路径矩阵进行初始化，将最短路径长度赋值为邻接矩阵的边权重值，将最短路径赋值为-1。接着进入三重循环，不断在 i 到 j 的路径上，引入 w，比较原有距离和引入 w 后的距离，取二者最小的为最短路径，并更新最短路径信息。

```
1.  void Floyd(MTGraph *G, int D[][NumVertices], int P[][NumVertices]) {
```

```
        for (int i = 0; i < G->n; i++) {
            for (int j = 0; j < G->n; j++) {
                D[i][j] = G->edge[i][j];
                P[i][j] = -1;
            }
        }
        for (int k = 0; k < G->n; k++) {
            for (int i = 0; i < G->n; i++) {
                for (int j = 0; j < G->n; j++) {
                    if (D[i][k] + D[k][j] < D[i][j]) {
                        D[i][j] = D[i][k] + D[k][j];
                        P[i][j] = k;
                    }
                }
            }
        }
    }
```

2．物理设计（即存储结构设计）

本实验主要采用邻接矩阵表示图，该存储结构包含四个数据，分别为顶点表（int 类型），边表（int 类型），顶点数（int 类型），边数（int 类型）。邻接矩阵表示的示意图如下图所示。

图 2 邻接矩阵表示存储

四、测试结果（包括测试数据、结果数据及结果的简单分析和结论，可以用截图得形式贴入此报告）

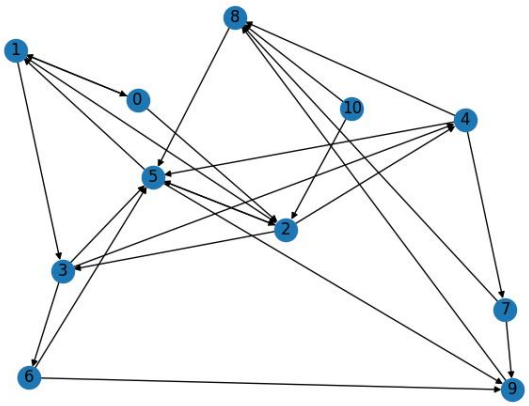

图 3 构造的有向图

1.实现单源最短路径的 Dijkstra 算法

```
请选择功能实现:
1.实现单源最短路径的Dijkstra算法;
2.实现全局最短路径的Floyd-Warshall算法;
3.利用Floyd-Warshall算法解决单目标最短路径问题;
4.利用Floyd-Warshall算法解决单顶点对间最短路径问题;
0.退出
1
请输入源点:7
源点7到顶点0:最短路径为:0<-1<-5<-8<-7  最短路径长度为:21
源点7到顶点1:最短路径为:1<-5<-8<-7  最短路径长度为:18
源点7到顶点2:最短路径为:2<-5<-8<-7  最短路径长度为:15
源点7到顶点3:最短路径为:3<-2<-5<-8<-7  最短路径长度为:18
源点7到顶点4:最短路径为:4<-2<-5<-8<-7  最短路径长度为:19
源点7到顶点5:最短路径为:5<-8<-7  最短路径长度为:8
源点7到顶点6:最短路径为:6<-3<-2<-5<-8<-7  最短路径长度为:22
源点7到顶点8:最短路径为:8<-7  最短路径长度为:1
源点7到顶点9:最短路径为:9<-7  最短路径长度为:2
源点7到顶点10:无最短路径
```

图 4 最短路径（Dijkstra）

2.实现全局最短路径的 Floyd-Warshall 算法

```
请选择功能实现:
1.实现单源最短路径的Dijkstra算法;
2.实现全局最短路径的Floyd-Warshall算法;
3.利用Floyd-Warshall算法解决单目标最短路径问题;
4.利用Floyd-Warshall算法解决单顶点对间最短路径问题;
0.退出
1
请输入源点:7
源点7到顶点0:最短路径为:0<-1<-5<-8<-7  最短路径长度为:21
源点7到顶点1:最短路径为:1<-5<-8<-7  最短路径长度为:18
源点7到顶点2:最短路径为:2<-5<-8<-7  最短路径长度为:15
源点7到顶点3:最短路径为:3<-2<-5<-8<-7  最短路径长度为:18
源点7到顶点4:最短路径为:4<-2<-5<-8<-7  最短路径长度为:19
源点7到顶点5:最短路径为:5<-8<-7  最短路径长度为:8
源点7到顶点6:最短路径为:6<-3<-2<-5<-8<-7  最短路径长度为:22
源点7到顶点8:最短路径为:8<-7  最短路径长度为:1
源点7到顶点9:最短路径为:9<-7  最短路径长度为:2
源点7到顶点10:无最短路径
请选择功能实现:
1.实现单源最短路径的Dijkstra算法;
2.实现全局最短路径的Floyd-Warshall算法;
3.利用Floyd-Warshall算法解决单目标最短路径问题;
4.利用Floyd-Warshall算法解决单顶点对间最短路径问题;
0.退出
2
```

图 5-a 最短路径（Floyd-Warshall）

```
最短路径长度矩阵为:
   0    2    3    5    6    7    9   10   10    9  inf
   3    0    1    3    4    5    7    8    8    7  inf
  18   15    0    3    4    5    7    8    8    7  inf
  15   12    9    0    1    2    4    5    5    4  inf
  14   11    8   11    0    1   15    4    4    3  inf
  13   10    7   10   11    0   14   15    3    2  inf
  15   12    9   12   13    2    0   17    2    1  inf
  21   18   15   18   19    8   22    0    1    2  inf
  20   17   14   17   18    7   21   22    0    9  inf
  21   18   15   18   19    8   22   23    1    0  inf
  23   20    7   10   11   10   14   15    3   12    0

最短路径矩阵为:
  -1   -1    1    1    3    4    3    4    9    5   -1
  -1   -1   -1   -1    3    4    3    4    9    5   -1
   5    5   -1   -1   -1    4    3    4    9    5   -1
   5    5    5   -1   -1    4   -1    4    9    5   -1
   5    5    5    5   -1   -1    5   -1    9    5   -1
   1   -1   -1    2    2   -1    3    4    9   -1   -1
   5    5    5    5    5   -1   -1    5    9   -1   -1
   8    8    8    8    8    8    8   -1   -1   -1   -1
   5    5    5    5    5   -1    5    5   -1    5   -1
   8    8    8    8    8    8    8    8   -1   -1   -1
   8    8   -1    2    2    8    3    4   -1    8   -1

源点0到顶点1:最短路径为:0->1  最短路径长度为:2
源点0到顶点2:最短路径为:0->1->2  最短路径长度为:3
源点0到顶点3:最短路径为:0->1->3  最短路径长度为:5
源点0到顶点4:最短路径为:0->1->3->4  最短路径长度为:6
源点0到顶点5:最短路径为:0->1->3->4->5  最短路径长度为:7
源点0到顶点6:最短路径为:0->1->3->6  最短路径长度为:9
源点0到顶点7:最短路径为:0->1->3->4->7  最短路径长度为:10
源点0到顶点8:最短路径为:0->1->3->4->5->9->8  最短路径长度为:10
源点0到顶点9:最短路径为:0->1->3->4->5->9  最短路径长度为:9
源点0到顶点10:无最短路径
源点1到顶点0:最短路径为:1->0  最短路径长度为:3
源点1到顶点2:最短路径为:1->2  最短路径长度为:1
源点1到顶点3:最短路径为:1->3  最短路径长度为:3
源点1到顶点4:最短路径为:1->3->4  最短路径长度为:4
源点1到顶点5:最短路径为:1->3->4->5  最短路径长度为:5
源点1到顶点6:最短路径为:1->3->6  最短路径长度为:7
源点1到顶点7:最短路径为:1->3->4->7  最短路径长度为:8
源点1到顶点8:最短路径为:1->3->4->5->9->8  最短路径长度为:8
源点1到顶点9:最短路径为:1->3->4->5->9  最短路径长度为:7
源点1到顶点10:无最短路径
源点2到顶点0:最短路径为:2->4->5->1->0  最短路径长度为:18
```

图 5-b 最短路径（Floyd-Warshall）

3.利用 Floyd-Warshall 算法解决单目标最短路径问题

```
请选择功能实现:
1.实现单源最短路径的Dijkstra算法;
2.实现全局最短路径的Floyd-Warshall算法;
3.利用Floyd-Warshall算法解决单目标最短路径问题;
4.利用Floyd-Warshall算法解决单顶点对间最短路径问题;
0.退出
3
请输入目标点:7
源点0到目标点7:最短路径为:0->1->3->4->7  最短路径长度为:10
源点1到目标点7:最短路径为:1->3->4->7  最短路径长度为:8
源点2到目标点7:最短路径为:2->4->7  最短路径长度为:8
源点3到目标点7:最短路径为:3->4->7  最短路径长度为:5
源点4到目标点7:最短路径为:4->7  最短路径长度为:4
源点5到目标点7:最短路径为:5->2->4->7  最短路径长度为:15
源点6到目标点7:最短路径为:6->5->2->4->7  最短路径长度为:17
源点8到目标点7:最短路径为:8->5->2->4->7  最短路径长度为:22
源点9到目标点7:最短路径为:9->8->5->2->4->7  最短路径长度为:23
源点10到目标点7:最短路径为:10->2->4->7  最短路径长度为:15
```

图 6 单目标最短路径（Floyd-Warshall）

4.利用 Floyd-Warshall 算法解决单顶点对间最短路径问题

```
请选择功能实现:
1.实现单源最短路径的Dijkstra算法;
2.实现全局最短路径的Floyd-Warshall算法;
3.利用Floyd-Warshall算法解决单目标最短路径问题;
4.利用Floyd-Warshall算法解决单顶点对间最短路径问题;
0.退出
4
请输入两个顶点:5 7
源点5到顶点7:最短路径为:5->2->4->7  最短路径长度为:15
源点7到顶点5:最短路径为:7->8->5  最短路径长度为:8
```

图 7 单顶点对间最短路径（Floyd-Warshall）

五、经验体会与不足

经验体会：通过实现最短路径问题的编码实现，熟练掌握单源最短路径问题和所有顶点对之间的最短路径问题，熟练掌握了两个重要求解最短路径的算法，即 Kijkstra 算法和 Floyd-Warshall 算法。

不足：程序中的一些算法编写的不够高效简洁，还存在时间和空间的不必要消耗，并且程序的输出端口编写的也存在不规范之处。

六、附录：源代码（带注释）

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

#define VertexData int
#define EdgeData int
#define NumVertices 100
#define inf 99999
/*---------------邻接矩阵表示---------------*/
typedef struct {
    VertexData vertex[NumVertices];            // 顶点表
    EdgeData edge[NumVertices][NumVertices];  // 邻接矩阵——边表
    int n;                                     // 图的顶点数
    int e;                                     // 图的边数
} MTGraph;
// 创建图(从 txt 文件中读取)
void Create_Graph(MTGraph *G) {
    FILE *fp;
    fp = fopen("Graph.txt", "r");
    fscanf(fp, "%d", &G->n);
    fscanf(fp, "%d", &G->e);
    for (int i = 0; i < G->n; i++) {
        for (int j = 0; j < G->n; j++) {
            G->edge[i][j] = inf;
        }
    }
    for (int i = 0; i < G->n; i++) {
        G->edge[i][i] = 0;
    }
    for (int i = 0; i < G->n; i++) {
        fscanf(fp, "%d", &G->vertex[i]);
    }
    for (int i = 0; i < G->e; i++) {
        int u, v, w;
        fscanf(fp, "%d %d %d", &u, &v, &w);
        G->edge[u][v] = w;
    }
    fclose(fp);
}
// 查找最短距离的顶点
```

```
int MinCost(MTGraph *G, int D[], int S[]) {
    int temp = inf;
    int w = 0;
    for (int i = 0; i < G->n; i++) {
        if (!S[i] && D[i] < temp) {
            temp = D[i];
            w = i;
        }
    }
    return w;
}
// Dijkstra 算法
void Dijkstra(MTGraph *G, int source, int D[], int P[], int S[]) {
    for (int i = 0; i < G->n; i++) {
        D[i] = G->edge[source][i];
        P[i] = source;
        S[i] = 0;
    }
    S[source] = 1;
    for (int i = 0; i < G->n - 1; i++) {
        int w = MinCost(G, D, S);
        S[w] = 1;
        for (int v = 0; v < G->n; v++) {
            if (!S[v]) {
                int sum = D[w] + G->edge[w][v];
                if (sum < D[v]) {
                    D[v] = sum;
                    P[v] = w;
                }
            }
        }
    }
}
// 打印 Dijkstra 算法的最短路径(给定源点,打印源点到其他顶点的最短路径和长度)例如:源点为 1，目标点为 2，输出为 2<-1
void Print_Dijkstra(MTGraph *G, int source, int D[], int P[]) {
    for (int i = 0; i < G->n; i++) {
        if (i != source) {
            printf("源点%d 到顶点%d:", source, i);
            if (D[i] != inf) {
                printf("最短路径为:");
                printf("%d", i);
```

```
                    int k = P[i];
                    while (k != source) {
                        printf("<-%d", k);
                        k = P[k];
                    }
                                printf("<-%d    最短路径长度为:%d\n", source, D[i]);
                } else {
                    printf("无最短路径\n");
                }
            }
        }
    }
    // Floyd 算法
    void Floyd(MTGraph *G, int D[][NumVertices], int P[][NumVertices]) {
        for (int i = 0; i < G->n; i++) {
            for (int j = 0; j < G->n; j++) {
                D[i][j] = G->edge[i][j];
                P[i][j] = -1;
            }
        }
        for (int k = 0; k < G->n; k++) {
            for (int i = 0; i < G->n; i++) {
                for (int j = 0; j < G->n; j++) {
                    if (D[i][k] + D[k][j] < D[i][j]) {
                        D[i][j] = D[i][k] + D[k][j];
                        P[i][j] = k;
                    }
                }
            }
        }
    }
    // 递归查找路径
    void Path(int i, int j, int P[][NumVertices]) {
        int k = P[i][j];
        if (k != -1) {
            Path(i, k, P);
            printf("%d->", k);
            Path(k, j, P);
        }
    }
    // 打印 Floyd 算法的最短路径(打印最短距离矩阵、最短路径矩阵和任意两点之间的最短路径和长度)
```

```
122. void Print_Floyd(MTGraph *G, int D[][NumVertices], int P[][NumVertices]) {
123.     printf("最短路径长度矩阵为:\n");
124.     for (int i = 0; i < G->n; i++) {
125.         for (int j = 0; j < G->n; j++) {
126.             if (D[i][j] == inf) {
127.                 printf("   inf ");
128.             } else {
129.                 printf("%4d ", D[i][j]);
130.             }
131.         }
132.         printf("\n");
133.     }
134.     printf("\n");
135.     printf("最短路径矩阵为:\n");
136.     for (int i = 0; i < G->n; i++) {
137.         for (int j = 0; j < G->n; j++) {
138.             printf("%4d ", P[i][j]);
139.         }
140.         printf("\n");
141.     }
142.     printf("\n");
143.     for (int i = 0; i < G->n; i++) {
144.         for (int j = 0; j < G->n; j++) {
145.             if (i != j) {
146.                 printf("源点%d 到顶点%d:", i, j);
147.                 if (D[i][j] != inf) {
148.                     printf("最短路径为:");
149.                     printf("%d->", i);
150.                     Path(i, j, P);
151.                                 printf("%d   最短路径长度为:%d\n", j, D[i][j]);
152.                 } else {
153.                     printf("无最短路径\n");
154.                 }
155.             }
156.         }
157.     }
158. }
159. // 利用 Floyd-Warshall 算法解决单目标最短路径问题(给定目标点，打印源点到目标点的最短路径和长度)
160. void Single_Target(MTGraph *G, int D[][NumVertices], int P[][NumVertices]) {
161.     int target;
```

```
    printf("请输入目标点:");
    scanf("%d", &target);
    for (int i = 0; i < G->n; i++) {
        if (i != target) {
            printf("源点%d 到目标点%d:", i, target);
            if (D[i][target] != inf) {
                printf("最短路径为:");
                printf("%d->", i);
                Path(i, target, P);
                                printf("%d    最 短 路 径 长 度为:%d\n", target, D[i][target]);
            } else {
                printf("无最短路径\n");
            }
        }
    }
}
// 利用 Floyd-Warshall 算法解决单顶点对间最短路径问题(给定两个顶点，打印两个顶点之间的最短路径和长度)
void Single_Pair(MTGraph *G, int D[][NumVertices], int P[][NumVertices]) {
    int u, v;
    printf("请输入两个顶点:");
    scanf("%d %d", &u, &v);
    printf("源点%d 到顶点%d:", u, v);
    if (D[u][v] != inf) {
        printf("最短路径为:");
        printf("%d->", u);
        Path(u, v, P);
        printf("%d  最短路径长度为:%d\n", v, D[u][v]);
    } else {
        printf("无最短路径\n");
    }
    printf("源点%d 到顶点%d:", v, u);
    if (D[v][u] != inf) {
        printf("最短路径为:");
        printf("%d->", v);
        Path(v, u, P);
        printf("%d  最短路径长度为:%d\n", u, D[v][u]);
    } else {
        printf("无最短路径\n");
    }
}
int main() {
```

```
    while (1) {
        int function_option;
        printf(
            "请选择功能实现:\n1.实现单源最短路径的Dijkstra算法;\n2."
            "实现全局最短路径的 Floyd-Warshall 算法;\n3.利用Floyd-"
            "Warshall 算法解决单目标最短路径问题;\n4.利用Floyd-"
            "Warshall 算法解决单顶点对间最短路径问题;\n0.退出\n");
        scanf("%d", &function_option);
        if (function_option == 1) {
            MTGraph *G = (MTGraph *)malloc(sizeof(MTGraph));
            Create_Graph(G);
            int D[NumVertices];  // 源点 1 到源点 i 的当前最短路径长度
            int P[NumVertices];  // 源点 1 到源点 i 的当前最短路径上，最后经过的顶点
            int S[NumVertices];  // 存放源点和已生成的终点，实际上是 bool 类型
            int source;
            printf("请输入源点:");
            scanf("%d", &source);
            Dijkstra(G, source, D, P, S);
            Print_Dijkstra(G, source, D, P);
        } else if (function_option == 2) {
            MTGraph *G = (MTGraph *)malloc(sizeof(MTGraph));
            Create_Graph(G);
             int D[NumVertices][NumVertices];  // 最短路径长度矩阵
             int P[NumVertices][NumVertices];  // 最短路径矩阵
            Floyd(G, D, P);
            Print_Floyd(G, D, P);
        } else if (function_option == 3) {
            MTGraph *G = (MTGraph *)malloc(sizeof(MTGraph));
            Create_Graph(G);
             int D[NumVertices][NumVertices];  // 最短路径长度矩阵
```

```
233.                int P[NumVertices][NumVertices];  // 最短路径矩阵
234.                Floyd(G, D, P);
235.                Single_Target(G, D, P);
236.            } else if (function_option == 4) {
237.                MTGraph *G = (MTGraph *)malloc(sizeof(MTGraph));
238.                Create_Graph(G);
239.                int D[NumVertices][NumVertices];  // 最短路径长度矩阵
240.                int P[NumVertices][NumVertices];  // 最短路径矩阵
241.                Floyd(G, D, P);
242.                Single_Pair(G, D, P);
243.            } else if (function_option == 0) {
244.                break;
245.            } else {
246.                printf("输入错误，请重新输入\n");
247.            }
248.        }
249.        return 0;
250. }
```