江西财经大学

信息管理学院综合性实验报告

课程名称：	数据结构与算法课程实践		
题　　目：	校园导游系统设计		
小组成员			
成员姓名：	周旭兵	学号：	0222807
成员姓名：	邹睿坤	学号：	0222668
成员姓名：	彭子康	学号：	0222836
成员姓名：	郭小平	学号：	0224898
教学班：			
指导教师：			
开课学期：	2023-2024 学年第 2 学期		

（时间：2024 年 4 月 1 日 —— 2024 年 4 月 16 日）

目录

正文结构说明

一、需求分析

1. 问题描述

该系统的主要设计目标是为江西财经大学麦庐校区的访客提供一个人性化、简单易用的电子导游服务。这个系统旨在通过电子地图与实际环境的对比观察，帮助初次来访的用户轻松地在校园内导航，发现和了解校园的主要地点和服务设施，包括教学楼、图书馆、学生宿舍、运动场所、食堂等。此外，系统应能提供校园的基本信息和各个景点的介绍，增加用户对校园的了解和兴趣。

2. 用户需求（从使用者的视角）

从使用者的视角出发，用户需求可以分为以下几类：

首页菜单功能：

用户可以在电子地图上搜索特定的地点，并获取到该地点的详细信息和如何到达的指引。首页菜单应该列出系统提供的各种功能选项，以便用户根据自己的需求选择相应的功能。例如，查看游览路线、查询景点间最短路径、查询景点间所有路径、学校景点介绍、景点信息查询等菜单界面应该有清晰的布局和格式，使用户能够快速了解每个功能选项，并能够轻松地进行选择。

查看导航路线服务：

提供校园内的地点信息、开放时间、联系方式等实用信息。系统需要接受用户输入的起始景点和目标景点，以便确定导航的起点和终点。系统应该对用户输入的起始景点和目标景点进行合法性检查，确保输入的景点编号在系统中存在，系统根据用户输入的起始景点和目标景点，在校园地图中查询并计算出两个景点之间的导航路线。

学校景点介绍：

查询特定景点的信息：用户应能够通过输入景点编号或名称来获取该景点的详细信息，包括名称、介绍等。
显示所有景点信息：用户应能够查看所有校园景点的信息列表，以便了解校园的整体情况。
信息展示格式：信息展示应该清晰易读，包括景点编号、名称和介绍等内容。
错误处理：系统应该能够处理用户可能输入的错误，如不存在的景点编号或名称

3. 功能分析（从开发者的视角）

从开发者的视角进行功能分析，系统主要应包括以下功能模块：

地图展示和导航模块：

用户可以通过搜索来获取地点信息和导航路线。用户可以通过搜索来获取地点信息和导航路线。用户可以通过系统查看校园的地图，显示各个景点的位置分布。地图展示应该直观清晰，能够让用户快速了解校园布局。

信息查询模块：

提供关于校园地点、活动、历史背景的详细信息查询功能。用户可以通过输入景点编号或名称来查询特定景点的详细信息。

查询结果应包括景点名称、位置、介绍等相关信息，

用户可以查看系统中所有校园景点的信息列表，以便了解校园的整体情况。

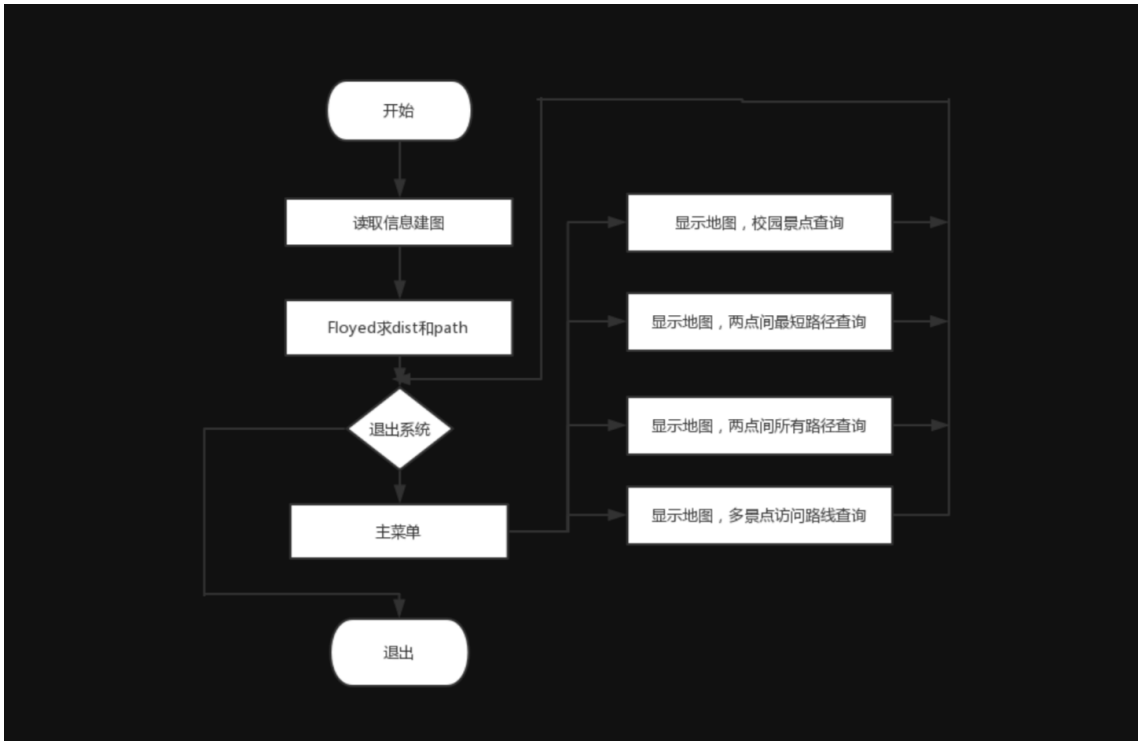
每个景点信息应包括编号、名称、位置、介绍等内容

用户交互模块：

设计简洁直观的用户界面，使用户能够轻松理解和操作。
界面布局合理，功能模块分类清晰，用户可以快速找到需要的功能。
显示菜单：用户交互模块首先会显示系统的菜单，菜单中列出了用户可以选择的各项功能，以便用户了解可以进行的操作。
获取用户选择：用户可以通过输入菜单中对应功能的编号来选择要执行的操作。用户交互模块负责获取用户的选择，并将选择传递给其他模块进行处理。
执行相应功能：根据用户的选择，用户交互模块调用其他模块中相应的函数或方法来执行对应的功能。比如，根据用户选择调用查看游览路线函数、查询景点间最短路径函数等。
显示结果：在执行完用户选择的功能后，用户交互模块负责将结果显示给用户。这可能是景点信息、路径信息或其他用户感兴趣的信息。
循环执行：用户交互模块会循环执行以上步骤，直到用户选择退出系统。这样用户就可以连续进行多次操作，而不需要重启系统。
通过以上功能，用户交互模块实现了用户与系统之间的信息交互，使用户能够方便地使用系统提供的功能。

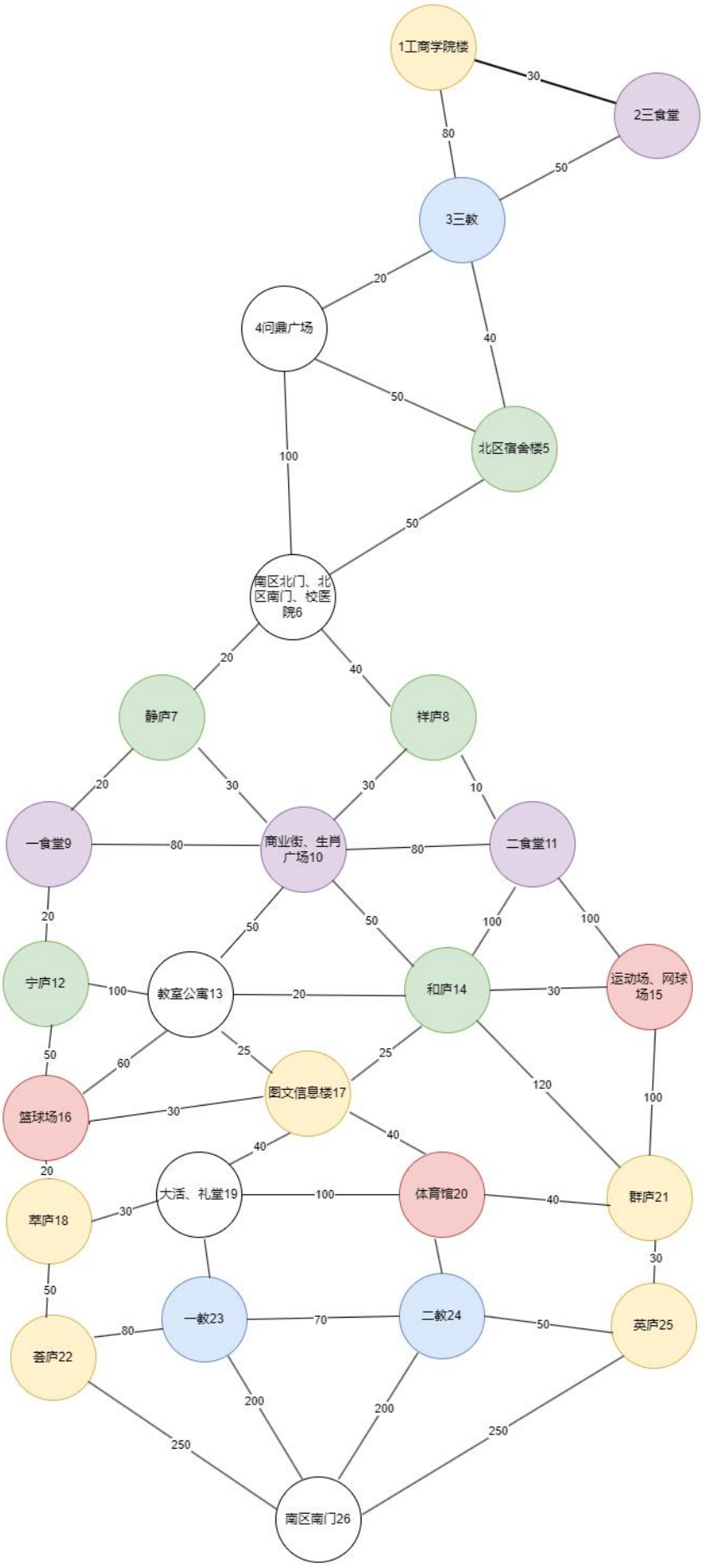
二、概要设计

1. 系统框架



2. 数据模型

首先是进行校园地图的概念性设计，将地图抽象为无向带权图，使接下来的任务更加高效方便：


1工商学院楼
2三食堂
3三教
4问鼎广场
北区宿舍楼5
南区北门、北区南门、校医院6
静庐7
祥庐8
一食堂9
商业街、生肖广场10
二食堂11
宁庐12
教室公寓13
和庐14
运动场、网球场15
篮球场16
图文信息楼17
大活、礼堂19
体育馆20
群庐21
萃庐18
荟庐22
一教23
二教24
英庐25
南区南门26
30
80
50
20
40
100
50
20
40
20
30
30
10
20
80
80
20
50
50
100
100
100
50
60
25
25
120
30
40
40
20
30
100
40
50
80
70
50
200
200
250
250

共建模出 26 个校园地点与 47 条边。
然后是数据模型的分析：

1.景点信息 (vexsinfo):

position: 景点的编号，用整数表示。
name: 景点的名称，字符串类型，最多 32 个字符。
introduction: 景点的介绍，字符串类型，最多 256 个字符。

2.边的权值信息 (arcell):

adj: 表示边的权值，整数类型，用于表示景点之间的距离或成本。Infinity 表示两景点之间无直接连接。

3.邻接矩阵 (adjmatrix):

存储了景点之间的连接关系和距离信息。
使用二维数组表示，大小为 MaxVertexNum x MaxVertexNum，其中 MaxVertexNum 是顶点的最大数量。

4.图结构信息 (mgraph):

vexs: 顶点向量，存储了所有景点的信息。
arcs: 邻接矩阵，存储了景点之间的连接关系和距离信息。
vexnum: 图中顶点的数量。
arcnum: 图中边的数量。

3. UI 界面设计（建议是：CLI 模式下纯文本的 UI 操作界面）

由于是 C 语言实现，用户界面可能限于控制台应用程序。UI 设计将集中于如何使控制台输入输出用户友好。

以上是一个文本形式的 UI 界面设计，它采用了简洁明了的排版，清晰地展示了系统提供的各项功能。用户可以通过输入相应功能的编号来选择要执行的操作。整体设计遵循了简洁直观、易于理解的原则，以提高用户的操作效率和体验。

三、详细设计

1. 数据设计

```
typedef struct arcell
{
    int adj;
}arcell, adjmatrix[MaxVertexNum][MaxVertexNum];
```

arcell 结构体定义了校园内景点间路径的权值信息。其中：
int adj 表示路径的权值。在校园导游系统中，这个权值了代表路径的距离。例如：

```
strcpy_s(c.vexs[9].name, "一食堂");
strcpy_s(c.vexs[11].name, "二食堂");
```

```
c.arcs[9][11].adj = 300;
```

说明二食堂与一食堂之间的变的权值是 300，代表了这两个景点间的距离为 300m。
这个数值越大说明两个景点之间的实际距离越远，数值越小代表两个景点间的实际距离越近。

```
typedef struct vexsinfo
{
    int position;
    char name[32];
    char introduction[256];
}vexsinfo;
```

vexsinfo 结构体用于存储校园内单个景点的详细信息。其中：
int position: 用于标识景点的唯一编号。在导游系统中，每个景点应有一个唯一的编号以方便管理和查询。而 char name[32]用于存储景点的名称，如“一食堂”、“二食堂”等。

```
strcpy_s(c.vexs[9].name, "一食堂");
strcpy_s(c.vexs[11].name, "二食堂");
```

其中一食堂的编号为 9，二食堂的编号为 11。
char introduction[256]用于提供景点的详细介绍，如历史背景、特色景观、开放时间、参观注意事项等。这部分信息有助于用户深入了解景点，提升游览体验。例如：

```
strcpy_s(c.vexs[10].name, "商业街、生肖广场");
strcpy_s(c.vexs[10].introduction, "收递快递处");
```

图中 10 号景点“商业街，生肖广场”的介绍为“快递收发处”。

```
typedef struct mgraph
{
    vexsinfo vexs[MaxVertexNum];
    adjmatrix arcs;
    int vexnum, arcnum;
}mgraph;
```

mgraph 结构体是整个校园导游系统的图数据结构，它将所有景点及其相互关系组织在一起。其中：

vexsinfo vexs[MaxVertexNum]是一个固定大小的数组，用于存储校园内所有景点的详细信息。数组的每个元素都是一个 vexsinfo 结构体，对应一个具体的景点。

adjmatrix arcs 是 arcell 类型的二维数组，在代码中作为邻接矩阵，用于表示景点间的连接关系以及对应的权值。邻接矩阵的行和列分别对应于景点编号。对于矩阵中的元素 (i, j)，其值 arcs[i][j].adj 表示从景点 i 到景点 j 的权值。例如：

```
for (i = 1; i <= key; i++)
    for (j = 1; j <= key; j++)
        c.arcs[i][j].adj = Infinity;
c.arcs[1][2].adj = 150;
c.arcs[1][3].adj = 100;
c.arcs[2][3].adj = 190;
c.arcs[3][4].adj = 150;
c.arcs[3][5].adj = 100;
c.arcs[4][6].adj = 300;
c.arcs[4][5].adj = 200;
c.arcs[5][6].adj = 160;
c.arcs[6][7].adj = 100;
c.arcs[6][8].adj = 110;
```

```
c.arcs[18][20].adj = 100;
for (i = 1; i <= key; i++)
    for (j = 1; j <= key; j++)
        c.arcs[j][i].adj = c.arcs[i][j].adj;
return c;
```

这是对邻接矩阵的初始化（省略了中间部分代码），使用两层嵌套循环初始化给定大小（由变

量 key 确定，key 为定点个数，定义为了 26）的邻接矩阵。循环变量 i 和 j 分别从 1 遍历到 key。在每次循环中，将矩阵元素 c.arcs[i][j].adj 赋值为 Infinity（Infinity 定义为 2000）。目的是确保在后续操作之前，图中的每条边都具有表示无穷大的初始权值，有助于区分尚未赋值的真实边与不存在的边。而后运用无向图的对称性，对于每个元素 c.arcs[i][j]，将其权值赋给对应的对角线元素 c.arcs[j][i]。

int vexnum: 存储当前校园内景点的实际数量，即已使用的 vexsinfo 数组元素个数。

int arcnum: 记录校园内路径的总数，即邻接矩阵中非零权值的元素个数。

2. 算法设计

2.1 查看游览路线算法（迪杰斯特拉算法）

```
void shortestpath_dij(mgraph c)
{
    int v0,v,w,k=1,min,t,p;
    int final[MaxVertexNum];//final[w]=1表示已经求得顶点V0到Vw的最短路径
    int Patharc[MaxVertexNum];//用于存储最短路径下标的数组
    int ShortPathtable[MaxVertexNum];//用于存储到各点最短路径的权值和
    printf("\n请输入一个起始景点的编号: ");
    scanf("%d",&v0);printf("\n\n");
    while(v0<0||v0>key)
    {
        printf("\n您输入的景点编号不存在\n");
        printf("请重新输入: ");
        scanf("%d",&v0);
    }
```

这部分代码首先定义了一些变量和数组：

int v0 是用户输入的起始顶点的编号；

int v, w, k, min, t, p 都是用于循环和计算的临时变量。其中，v 和 w 通常用作循环变量，k 用于存储当前找到的最近的顶点，min 用于存储当前的最短距离，t 和 p 用于在打印最短路径时作为临时变量；

int final[MaxVertexNum]数组用于标记已经找到最短路径的顶点。如果 final[w] 为 1，表示已经找到了从起始顶点 v0 到顶点 w 的最短路径；

int Patharc[MaxVertexNum]数组用于存储最短路径中的顶点编号。Patharc[w] 存储的是从 v0 到 w 的最短路径中，w 的前一个顶点；

int ShortPathtable[MaxVertexNum]数组用于存储到各点最短路径的权值和。ShortPathtable[w] 存储的是从 v0 到 w 的最短路径的长度。

用户被提示输入一个起始景点的编号，该编号被存储在 v0 中。如果输入的编号无效（即小于 0 或大于 key），则会提示用户重新输入。

```
    //初始化数据
    for(v=1;v≤c.vexnum;v++)//数组0还是空出来
    {   final[v]=0;//全部顶点初始化为未找到最短路径
        ShortPathtable[v]=c.arcs[v0][v].adj;//将与v0点有连线的顶点加
上权值
        Patharc[v]=0;//初始化路径数组p为0
     }
     ShortPathtable[v0]=0;  // V0至v0的路径为0
     final[v0]=1;//V0至v0不需要路径
```

这部分代码初始化了 final、Patharc 和 ShortPathtable 数组。final 数组用于标记已经找到最短路径的顶点，Patharc 数组用于存储最短路径中的顶点编号，ShortPathtable 数组用于存储到各点最短路径的权值和。

```
     //开始主循环，每次求得V0到某个v顶点的 最短路径
     for(v=1;v≤c.vexnum;v++)
     {  min=Infinity;
        for(w=1;w≤c.vexnum;w++)//找出最近的顶点和权值
        {   if(!final[w]&&ShortPathtable[w]<min)//有边
            {   k=w;
                min=ShortPathtable[w];
             }
         }
         final[k]=1;//将目前找到的最近的顶点置1
         //修正当前最短路径及距离
         for(w=1;w≤c.vexnum;w++)
         {  //如果经过v顶点的路径比现在这条路径的长度短的话，更新
            if(!final[w]&&(min+c.arcs[k]
[w].adj<ShortPathtable[w]))
            {   ShortPathtable[w]=min+c.arcs[k][w].adj;//修改当前路
径长度
                Patharc[w]=k;  //存放前驱结点（像糖葫芦）
            }
          }
     }
```

这部分代码是 Dijkstra 算法的主循环。在每次循环中，函数首先找到当前未被访问过的顶点中，距离 v0 最近的顶点 k，然后更新 final[k] 为 1，表示已经找到了从 v0 到 k 的最短路径。然后，函数会更新从 v0 到所有其他未被访问过的顶点的距离。如果通过顶点 k 到某个顶点 w 的距离比当前的距离短，那么就更新距离，并将 k 保存为 w 的前驱节点。

```
    //打印最短路径
    for(t=1;t≤c.vexnum;t++)
    {  p=t;
       if(t≠v0)//反向输出
       {   printf("%d%s",t,c.vexs[t].name);
           for(w=1;w≤c.vexnum;w++)
           {   if(Patharc[p]≠0)
               {   printf("←—%d%s",Patharc[p],c.vexs[p].name);
                   p=Patharc[p];
               }
           }
           printf("←—%d%s",v0,c.vexs[v0].name);
           printf("\n总路线长为%d米\n\n",ShortPathtable[t]);
       }
    }
}
```

这部分代码输出从 v0 到所有其他顶点的最短路径和路径长度。注意，路径是反向输出的，即从目标顶点开始，通过前驱节点一直回溯到起始顶点。

2.2 查询两景点间的最短路径算法（floyd 算法）

```
void floyd(mgraph c)//一种暴力破解获取最短路径的算法
{
    int i, j, k;
    for (i = 1; i <= key; i++)//将图的邻接矩阵赋值给 shortest二维数组，将矩阵pathh全部初始化为-1
    {
        for (j = 1; j <= key; j++)
        {
            shortest[i][j] = c.arcs[i][j].adj;
            pathh[i][j] = j;
        }
    }
```

首先，这段代码初始化了两个二维数组 shortest 和 pathh，它们分别用于存储最短路径的权值和路径信息。这两个数组的维度都是顶点个数 key。通过两个嵌套的循环，将图的邻接矩阵中的路径长度赋值给 shortest，并将 pathh 的值初始化为终点的顶点编号。

shortest[i][j] = c.arcs[i][j].adj; // 将邻接矩阵中的权值复制给最短路径矩阵

pathh[i][j] = j; // 初始化路径矩阵，路径矩阵中的每个元素表示从顶点 i 到顶点 j 的路径中，j 的前一个顶点是 j 自身

```
for (k = 1; k <= key; k++)//核心操作，完成了以k为中间点对所有的顶点对（i,j）进行检测和修改
{
    for (i = 1; i <= key; i++)
    {
        for (j = 1; j <= key; j++)
        {
            if (shortest[i][j] > shortest[i][k] + shortest[k][j])
            {
                shortest[i][j] = shortest[i][k] + shortest[k][j];
                pathh[i][j] = pathh[i][k];//记录一下所走的路 //P数组用来存放前驱顶点
            }
        }
    }
}
```

接下来，算法进入三重循环，遍历所有顶点对 (i, j)，以及顶点 k。在每一轮循环中，算法检查是否存在通过顶点 k 的路径比直接连接顶点 i 和 j 的路径更短。如果是，则更新 shortest[i][j] 和 pathh[i][j]，将路径长度和路径信息更新为经过顶点 k 的路径。

这样，经过三重循环的迭代，最终 shortest 数组中存储的就是所有顶点对之间的最短路径的权值，而 pathh 数组中存储的是最短路径上的顶点信息。

复杂度：Floyd-Warshall 算法的时间复杂度为 O(n^3)，其中 n 是顶点的数量。这是因为算法使用三重循环遍历所有可能的顶点对，对于每一对顶点，都要检查是否存在更短的路径。

2.3 查询景点间所有路径算法

```
//4.查找两景点间的景点的所有路径
int allpath(mgraph c)
{
    int k, i, j, m, n;
    printf("\n\n请输入您想要查询的两个景点的编号: \n\n");
    scanf_s("%d%d", &i, &j); printf("\n\n");
    m = locatevex(c, i);
    n = locatevex(c, j);
    d[0] = m;
    for (k = 0; k < key; k++)
        visited[k] = 0;
    visited[m] = 1;
    path(c, m, n, 0);
    return 1;
}
```

这段代码定义了一个名为 allpath 的函数，功能是从给定的校园导游系统图中查询从用户指定的起点景点到终点景点的所有可达路径。

首先使用 locatevex 函数检查用户输入的景点编号是否有效。其中 locatevex 函数如下：

```
int locatevex(mgraph c, int v)
{
    int i;
    for (i = 1; i <= c.vexnum; i++)
        if (v == c.vexs[i].position)  return i;
    return -1;
}
```

如果找到景点，返回景点序号 i ，没有找到这个景点则返回-1。

然后初始化路径追踪变量，int d[]用于存储路径上的景点编号。将路径的起点 m 存入 d[0]，作为路径追踪的起始点。并初始化全局变量 visited[]，这是一个数组，长度等于图中的顶点数 26。将所有元素（对应每个景点）初始化为 0，表示未访问过任何顶点。然后将起点 m 对应的 visited[m] 设置为 1，表示已访问起点。最后使用 path 函数来搜索从起点 m 到终点 n 的所有路径。

```
void path(mgraph c, int m, int n, int k)
{
    int s, t = k + 1, length = 0;
    if (d[k] == n && k < 26)
    {
        for (s = 0; s < k; s++)
        {
            length = length + c.arcs[d[s]][d[s + 1]].adj;
        }

        {
            for (s = 0; s < k; s++)
            {
                printf("%d%s--->", d[s], c.vexs[d[s]].name);
            }
            printf("%d%s  ", d[s], c.vexs[d[s]].name);
            printf("总路线长为%d米\n\n", length);
        }
    }
    else
```

```
    {
        s = 1;
        while (s <= c.vexnum)
        {
            if ((c.arcs[d[k]][s].adj < Infinity) && (visited[s] == 0))
            {
                visited[s] = 1;
                d[k + 1] = s;
                path(c, m, n, t);
                visited[s] = 0;
            }
            s++;
        }
    }
```

其中：

mgraph c: 图结构，包含所有景点信息和它们之间的连接关系。

int m: 路径的起点景点编号。int n: 路径的终点景点编号。

int k: 当前路径深度，同时也是 d[] 数组的下标，用于记录路径上的景点顺序。

int s, t = k + 1 作用是：s 作为临时变量在后续循环中使用；t 初始化为 k + 1，表示路径下一位顶点对应的 d[] 数组下标。

int length = 0: 用于累计当前路径的总长度，初始化为 0。

首先检查当前路径末尾的顶点 d[k] 是否等于终点 n，并且路径长度（顶点数）不超过 26。如果是，则计算路径长度，方法是使用循环遍历 d[] 数组，累加相邻顶点间边的权值。

```
for (s = 0; s < k; s++)
{
    length = length + c.arcs[d[s]][d[s + 1]].adj;
}
```

然后输出路径，遍历 d[] 数组，依次打印路径上的景点编号和名称，并输出路径总长度。

如果顶点 d[k] 不等于终点 n，则进入递归搜索：

使用 s 作为计数器，从 1 遍历至 c.vexnum（图中的顶点个数）。

如果从当前路径末尾顶点 d[k] 到顶点 s 之间存在边，且顶点 s 未被访问过（visited[s] == 0）。

则将顶点 s 标记为已访问（visited[s] = 1），并将顶点 s 添加到路径中（d[t] = s），并将递归深度增加（path(c, m, n, t)）。在回溯时，撤销对顶点 s 的访问标记（visited[s] = 0），以便继续搜索其他路径。

2.4 景点信息查询算法

```
void seeabout(mgraph c)
{
    int k;
    printf("\n请输入要查询的景点编号: ");
    scanf_s("%d", &k);
    while (k <= 0 || k > key)
    {
        printf("\n您所输入的景点编号不存在! ");
        printf("\n请重新输入景点编号: ");
        scanf_s("%d", &k);
    }
    printf("\n\n编号: %-4d\n", c.vexs[k].position);
    printf("\n景点名称: %-10s\n", c.vexs[k].name);
    printf("\n详细介绍: %-80s\n\n", c.vexs[k].introduction);
}
```

首先使用 scanf_s 读取用户输入的景点编号 k。再启动一个 while 循环，检查 k 是否在有效范围内（即 1 <= k <= key）。key 表示图中景点的最大编号（26）。

如果用户输入的编号不在有效范围内，将提示错误，并让用户再次输入数据。当用户输入的有效景点编号满足条件后，跳出循环，开始输出该景点的详细信息。其中 %-4d、%-10s、%-80s 指定了相应字段的最小宽度和对齐方式。

2.5 学校景点介绍算法

```
void browsecompus(mgraph c)
{
    int i;
    printf(" \n\n编号        景点名称                    简介\n");
    printf("_____________________________________________________________________________\n");
    for (i = 1; i <= key; i++)
    {
        if (c.vexs[i].position != -1)
            printf("%-10d%-25s%-80s\n", c.vexs[i].position, c.vexs[i].name, c.vexs[i].introduction);
    }
    printf("_____________________________________________________________________________\n");
}
```

使用 for 循环遍历图结构 c 中的所有景点，循环变量 i 从 1 到 key（key=26)。循环结束后，所有有效景点的信息将展示在屏幕上。

3. UI 交互设计

```c
void mainwork()
{
    menu();
    int choice;
    campus = initgraph();
    do
    {
        printf("请输入您的选择: ");
        scanf_s("%d", &choice);
        switch (choice)
        {
        case 1:system("cls"); menu(); break;
        case 2:system("cls"); shortestpath_dij(campus); break;
        case 3:system("cls"); shortdistance(campus); break;
        case 4:system("cls"); allpath(campus); break;
        case 5:system("cls"); browsecompus(campus); break;
        case 6:system("cls"); seeabout(campus); break;
        case 0:system("cls"); printf("谢谢使用\n"); break;
        default:printf("未找到该功能，请输入有效选项！\n"); break;
        }
    } while (choice);
}
```

函数首先调用 menu() 函数，显示系统菜单：

```c
void menu()
{
    printf("┌──────────────────────────────────────────────────┐\n");
    printf("│ ╭═══════════════════════════════════════════╮    │\n");
    printf("│ ||        欢 迎 使 用 校 园 导 游 系 统        ||  │\n");
    printf("│ ╰═══════════════════════════════════════════╯    │\n");
    printf("│                      欢迎来到                    │\n");
    printf("│                     江西财经大学                 │\n");
    printf("│                     菜 单 选 择                  │\n");
    printf("│  ************************************************ │\n");
    printf("│  *  1.菜单                **   2.查看游览路线（迪）*  │\n");
    printf("│  ************************************************ │\n");
    printf("│  *  3.查询景点间最短路径  **   4.查询景点间所有路径*   │\n");
    printf("│  ************************************************ │\n");
    printf("│  *  5.学校景点介绍        **   6.景点信息查询      *  │\n");
    printf("│  ************************************************ │\n");
    printf("│  *  0.退出                **   第一小组0v0         *  │\n");
    printf("│  ************************************************ │\n");
    printf("└──────────────────────────────────────────────────┘\n");
}
```

再通过 switch case 函数，根据用户选择的值，使用 cls 清屏后执行相应的操作，例如：
用户输入 4，将调用 allpath(campus)查询所有路径。
用户输入 0，则退出系统并输出感谢信息。

四、 调试与测试

概述系统实现过程，对遇到的问题及其解决过程进行详述，内容包括：

- 开发、调试过程中遇到的问题及解决方法。
- 说明测试所用的测试数据和测试方法，包括输入和输出。这里的测试数据应该完整和严格，最好多于需求分析中要求的各项功能实现。
- 总结测试运行结果。

将代码在 vscode 中运行时会出现 vscode 集成终端无法显示完整 menu 菜单以及终端中文乱码问题：

对于集成终端无法显示完整 menu 菜单问题，我们最终通过在.vscode 文件夹里 launch.json 文件中的"configurations"中添加一行"externalConsole": true 解决此问题，最终的 launch.json 内容如下：

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "(gdb) Launch",
            "type": "cppdbg",
            "request": "launch",
            "program":
"${fileDirname}\\${fileBasenameNoExtension}.exe",
            "args": [],
            "stopAtEntry": false,
            "cwd": "${workspaceFolder}",
            "environment": [],
            "externalConsole": true,
            "MIMode": "gdb",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for
gdb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": true
                }
            ],
            "preLaunchTask": "C/C++: gcc.exe 生成活动文件"
        }
    ]
}
```

但是使用本地终端后，vscode 会出现终端中文乱码问题，我们通过[最简单优雅地解决 VsCode 的命令行显示中文乱码问题_vscode 中文乱码优雅-CSDN 博客]了解到了 Windows 终端一般默

认为 GBK 编码格式，而 vscode 中代码为 UTF-8 编码格式，因此若要在 vscode 中调试可以通过在 main 函数中添加 system("chcp 65001")来解决此问题，而在 vs 中不需要此步骤，最终的 main 函数效果如下：

```c
int main()
{
    system("chcp 65001"); //cmd chcp 命令切换字符编码为 utf-8 以显示中文,在VS中无需此行代码
    system("mode con: cols=140 lines=130");//程序页面长宽
    system("SchoolGraph.png");//调用校园地图图片
    mainwork(); //主要工作函数
    return 0;
}
```

三、系统总结与比较分析

1. 系统总结

1. 系统功能的完备程度和正确性：

该系统提供了多项功能，包括景点信息查询、最短路径查询、所有路径查询等，覆盖了用户在校园导游过程中可能需要的主要功能。
景点信息查询能够准确展示景点的名称和介绍，最短路径查询和所有路径查询能够正确计算出路径并给出准确的路径长度。
系统功能实现了用户需求，且在用户输入错误信息时能够进行错误提示和用户引导，保证了系统的正确性。

2. 数据结构对系统功能支持的有效性：

使用了邻接矩阵作为图的表示方式，这种数据结构简单直观，并且能够有效支持系统中的路径查询功能。
顶点信息和边的权值信息都被封装在了图的数据结构中，使得系统能够方便地对景点信息和路径信息进行管理和查询。

3. 算法运行效率：

系统中使用了迪杰斯特拉算法和弗洛伊德算法来计算最短路径，这两种算法在不同情况下有不同的运行效率：
迪杰斯特拉算法适用于求解单源最短路径问题，适用于稀疏图，其时间复杂度为 O(V^2)，其中 V 为顶点数。
弗洛伊德算法适用于求解任意两点间的最短路径，适用于稠密图，其时间复杂度为 O(V^3)，

其中 V 为顶点数。

虽然弗洛伊德算法的时间复杂度较高，但由于系统中的顶点数较小（26 个），因此对于这个规模的数据集，算法的运行效率仍然可以接受。
综上所述，该系统在功能完备性、正确性、数据结构有效性和算法运行效率等方面表现良好，能够满足用户在校园导游过程中的基本需求，并具有一定的可扩展性和性能优化空间。

2．比较研究

其他方法：
1.邻接表：
邻接表是另一种常用的图的表示方法，它通过链表来存储每个顶点的邻接点，适用于稀疏图，节省了存储空间。
在本题中，如果使用邻接表来表示图，可能会更节省存储空间，因为大学校园内景点之间的连接并不是非常密集，很多景点可能只与少数几个景点相连。
邻接表的查询效率略低于邻接矩阵，但对于大规模数据集，其存储空间和查询效率优势更为明显。

2.比较分析：
邻接矩阵适用于稠密图，而邻接表适用于稀疏图。在大学校园导游系统中，由于景点数量有限，且景点之间的连接关系并不是非常密集，因此邻接表可能更适合用于表示图。
使用邻接表存储图的话，可能需要更多的指针操作，而邻接矩阵则直接通过数组索引即可访问，因此在查询效率上可能稍有差异。
但是，无论是邻接矩阵还是邻接表，都可以有效支持系统中的功能，具体选择取决于系统的需求和数据特点。

四、附录

1．系统使用手册

江西财经大学校园导游系统使用手册

---

1. 系统简介
江西财经大学校园导游系统是一个帮助用户探索校园的应用程序。该系统提供了景点查询、路径导航等功能，让用户更方便地了解校园内的各个景点。

---

2. 安装与运行
本系统无需安装，只需下载可执行文件并运行即可开始使用。

---

3. 主菜单
系统启动后，您将看到一个主菜单，其中包含以下选项：
查看菜单
查看游览路线（迪杰斯特拉算法）
查询景点间最短路径（弗洛伊德算法）
查询景点间所有路径
学校景点介绍

景点信息查询
退出

---

4. 使用说明

查看游览路线（迪杰斯特拉算法）
选择此选项后，系统将要求您输入起始景点编号，然后显示从起始景点到其他景点的最短路径和距离。

查询景点间最短路径（弗洛伊德算法）
输入两个景点的编号，系统将显示这两个景点之间的最短路径和距离。

查询景点间所有路径
输入两个景点的编号，系统将显示这两个景点之间的所有路径及其长度。

学校景点介绍
您可以查看所有景点的名称和简介。

景点信息查询
输入景点编号，系统将显示该景点的详细信息，包括名称、介绍等。

退出
选择此选项退出系统。

---

5. 注意事项
在输入景点编号时，请确保输入的编号在有效范围内。
在进行路径查询时，请确保输入的景点编号存在。

2．小组成员任务划分

我们将任务划分分为文档撰写、代码编写、其他任务三方面：

文档撰写：

周旭兵完成详细设计中算法设计的查看游览路线算法（迪杰斯特拉算法）、调试与测试板块内容；
周旭兵、彭子康合作完成需求分析板块内容；
彭子康完成概要设计、系统使用手册板块内容；
邹睿坤完成详细设计的数据设计、算法设计中的查询两景点间所有路径算法、UI 交互设计板块内容；
郭小平、彭子康合作完成系统总结与比较分析板块内容；
郭小平完成详细设计中算法设计的景点信息查询算法、学校景点介绍算法板块内容。

代码编写：

周旭兵负责查看游览路线算法（迪杰斯特拉算法）的编写；
周旭兵与邹睿坤合作完成图的顶点与边的创建；
邹睿坤负责查询两景点间所有路径算法的编写；

彭子康负责查询两景点间的最短路径（floyd 算法）、menu 菜单的编写；
郭小平负责景点信息查询算法、学校景点介绍算法的编写；
Main 函数、数据结构的定义由小组合作讨论完成。

其他任务：

周旭兵与邹睿坤合作完成图的概念设计

3．成员个人总结和体会

该内容尚未完成