11.15

2.应用题

（4）根据迪杰斯特拉算法得出的表格如下

终点	**i=1**	**i=2**	**i=3**	**i=4**	**i=5**	**i=6**
b	15 (a,b)	15 (a,b)	15 (a,b)	15 (a,b)	15 (a,b)	**15 (a,b)**
c	**2 (a,c)**					
d	12 (a,d)	12 (a,d)	11 (a,c,f,d)	**11 (a,c,f,d)**		
e	∞	10 (a,c,e)	**10 (a,c,e)**			
f	∞	**6 (a,c,f)**				
g	∞	∞	16 (a,c,f,g)	16 (a,c,f,g)	**14 (a,c,f,d,g)**	
S 终点集	{a,c}	{a,c,f}	{a,c,e,f}	{a,c,d,e,f}	{a,c,f,e,d,g}	{a,b,c,d,e,f,g}

3.算法设计题

（5）

> 采用邻接表存储结构，设计一个算法，判别无向图中任意给定的两个顶点之间是否存在一条长度为k的简单路径。

完整代码如下：

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

#define MAXSIZE 100
//边
typedef struct ArcNode {
    int adjvex;
    struct ArcNode* nextarc;
} ArcNode;
//顶点
typedef struct VertexNode {
    int data;//编号
    ArcNode* firstarc;
} VertexNode;

typedef struct {
    VertexNode vertices[MAXSIZE];
    int visited[MAXSIZE];
    int vexnum, arcnum;//顶点数量，边数量
} ALGraph;
```

```
ALGraph createGraph(int vexnum, int arcnum) {
    ALGraph G;
    G.vexnum = vexnum;
    G.arcnum = arcnum;

    for (int i = 0; i < vexnum; i++) {
        G.vertices[i].data = i;//顶点编号
        G.vertices[i].firstarc = NULL;
        G.visited[i] = 0;
    }

    printf("Enter the arcs in the format (start end):\n");
    for (int i = 0; i < arcnum; i++) {
        int start, end;
        scanf("%d %d", &start, &end);

        ArcNode* arc = (ArcNode*)malloc(sizeof(ArcNode));
        //连接构造图
        arc->adjvex = end;
        arc->nextarc = G.vertices[start].firstarc;
        G.vertices[start].firstarc = arc;

        arc = (ArcNode*)malloc(sizeof(ArcNode));
        arc->adjvex = start;
        arc->nextarc = G.vertices[end].firstarc;
        G.vertices[end].firstarc = arc;
    }

    return G;
}

int exist_path_len(ALGraph G, int i, int j, int k) {
    if (i == j && k == 0)
        return 1; // 找到了一条路径，且长度符合要求
    else if (k > 0) {
        G.visited[i] = 1;
        ArcNode* p = G.vertices[i].firstarc;
        while (p) {
            int l = p->adjvex;
            if (!G.visited[l] && exist_path_len(G, l, j, k - 1))//递归查找，每次将
剩余路径减一
                return 1;
            p = p->nextarc;
        }
        G.visited[i] = 0; //若没有找到则将结点恢复至未访问状态
    }
    return 0; // 没找到
}

int main(){
    int vexnum, arcnum;
    printf("Enter the number of vertices in the graph: ");//图的顶点数
    scanf("%d", &vexnum);
```

```
    printf("Enter the number of arcs in the graph: ");//图的边数
    scanf("%d", &arcnum);

    ALGraph graph = createGraph(vexnum, arcnum);//创建图

    int start, end, length;
    printf("Enter the start vertex: ");//起点
    scanf("%d", &start);
    printf("Enter the end vertex: ");//终点
    scanf("%d", &end);
    printf("Enter the required path length: ");//路径长度
    scanf("%d", &length);

    if (exist_path_len(graph, start, end, length))
        printf("There exists a simple path of length %d between vertex %d and
%d.\n", length, start, end);//存在
    else
        printf("There does not exist a simple path of length %d between vertex %d
and %d.\n", length, start, end);//不存在

    return 0;
}
```

主要算法部分在exist_path_len函数中

逻辑思路：

首先，通过 createGraph 函数创建了一个邻接表表示的有向图。用户需要输入顶点数量和边数量，并按照指定格式输入每条边的起点和终点。

调用 exist_path_len 函数来判断是否存在从起点到终点的长度为指定值的简单路径。参数包括当前顶点 i、目标顶点 j、剩余路径长度 k，返回值为整数。如果当前顶点 i 等于目标顶点 j 且剩余路径长度 k 等于 0，说明找到了一条满足要求的路径，返回 1。否则，如果剩余路径长度 k 大于 0，将当前顶点 i 标记为已访问（即将 visited[i] 赋值为 1），并遍历顶点 i 的邻接顶点。对于每个未访问过的邻接顶点 l，递归调用 exist_path_len 函数，并将剩余路径长度减 1。如果在任意一次递归调用中找到了满足要求的路径，返回 1。最后，恢复当前顶点 i 的未访问状态（即将 visited[i] 赋值为 0），并返回 0，表示没有找到满足要求的路径。

最后，根据 exist_path_len 函数的返回值，在主函数中输出相应的结果。

程序测试结果如下：

```
Enter the number of vertices in the graph: 6
Enter the number of arcs in the graph: 7
Enter the arcs in the format (start end):
0 1
0 2
0 3
1 2
1 3
1 4
4 5
Enter the start vertex: 0
Enter the end vertex: 5
Enter the required path length: 4
There exists a simple path of length 4 between vertex 0 and 5.
```