一、【实验目的】

1、  理解图的存储结构与基本操作；

2、熟悉图的深度度优先遍历和广度优先遍历算法

3、掌握图的单源最短路径算法

二、【实验内容】

1.根据下图（图见实验12）邻接矩阵，编程实现该图的深度与广度优先遍历算法，输出遍历序列。

2.单源节点最短路径问题

问题描述:求从有向图的某一结点出发到其余各结点的最短路径。

基本要求:

（1）有向图采用邻接矩阵表示。

（2）单源节点最短路径问题采用Dijkstra算法。

（3）输出有向图中从源结点T到其余各结点的最短路径和最短路径值。

三、实验源代码

 Graph.h

#define MAX\_VERTICES 6

int matrix[MAX\_VERTICES][MAX\_VERTICES];

int visited[MAX\_VERTICES] = {0};

typedef struct Node {

    int data;

    struct Node \*next;

} Node;

typedef struct Queue {

    Node \*front;

    Node \*rear;

} Queue;

void initQueue(Queue \*q)

{

    q->front = q->rear = NULL;

}

int isEmpty(Queue \*q)

{

    return q->front == NULL;

}

void enQueue(Queue \*q, int data)

{

    Node \*newNode = (Node\*)malloc(sizeof(Node));

    newNode->data = data;

    newNode->next = NULL;

    if (isEmpty(q)) {

        q->front = q->rear = newNode;

    } else {

        q->rear->next = newNode;

        q->rear = newNode;

    }

}

int deQueue(Queue \*q)

{

    int data = q->front->data;

    Node \*temp = q->front;

    q->front = q->front->next;

    free(temp);

    return data;

}

void print(int v)

{

    printf("%d ", v);

}

void initMatrix()

{

    for (int i = 0; i < MAX\_VERTICES; i++) {

        for (int j = 0; j < MAX\_VERTICES; j++) {

            matrix[i][j] = 0;

        }

    }

}

void addEdge(int from, int to)

{

    matrix[from][to] = 1;

}

void dfs(int v)

{

    visited[v] = 1;

    print(v);

    for (int i = 0; i < MAX\_VERTICES; i++) {

        if (matrix[v][i] == 1 && visited[i] == 0) {

            dfs(i);

        }

    }

}

void bfs(int v)

{

    Queue q;

    initQueue(&q);

    visited[v] = 1;

    print(v);

    enQueue(&q, v);

    while (!isEmpty(&q)) {

        int cur = deQueue(&q);

        for (int i = 0; i < MAX\_VERTICES; i++) {

            if (matrix[cur][i] == 1 && visited[i] == 0) {

                visited[i] = 1;

                print(i);

                enQueue(&q, i);

            }

        }

    }

}

int dist[MAX\_VERTICES];

// Dijkstra

void dijkstra(int source)

{

    for (int i = 0; i < MAX\_VERTICES; i++) {

        dist[i] = INT\_MAX;

        visited[i] = 0;

    }

    dist[source] = 0;

    for (int i = 0; i < MAX\_VERTICES; i++) {

        int minDist = INT\_MAX, minIndex = -1;

        for (int j = 0; j < MAX\_VERTICES; j++) {

            if (!visited[j] && dist[j] <minDist) {

                minDist = dist[j];

                minIndex = j;

            }

        }

        visited[minIndex] = 1;

        for (int k = 0; k < MAX\_VERTICES; k++) {

            if (!visited[k] && matrix[minIndex][k] != 0 && dist[minIndex] != INT\_MAX

                && dist[minIndex] + matrix[minIndex][k] < dist[k]) {

                dist[k] = dist[minIndex] + matrix[minIndex][k];

            }

        }

    }

}

void printPath(int source)

{

    printf("输出%d最短路径\n", source);

    for (int i = 0; i < MAX\_VERTICES; i++) {

        if (i == source) continue;

        if (dist[i] == INT\_MAX) {

            printf("无法到达最短路径%d\n", i);

        } else {

            printf("可到达%d最短路径", i);

            printf("%d",source);

            int cur = i;

            while (cur != source) {

                printf(" < %d", cur);

                cur = (dist[cur] == INT\_MAX) ? cur : (dist[cur] + matrix[cur][source] == dist[source]) ? source : cur-1;

            }

            printf(",最短路径值为%d\n", dist[i]);

        }

    }

}

Main.c

#include <stdio.h>

#include <stdlib.h>

#include"graph13.h"

int main()

{

    initMatrix();

    addEdge(0, 2);

    addEdge(0, 3);

    addEdge(1, 0);

    addEdge(1, 4);

    addEdge(2, 1);

    addEdge(2, 5);

    addEdge(4, 3);

    addEdge(5, 3);

    addEdge(5, 4);

    printf("深度遍历\n");

    dfs(0);

    printf("\n");

    for (int i = 0; i < MAX\_VERTICES; i++) {

        visited[i] = 0;

    }

    printf("\n");

    printf("广度遍历结果\n");

    bfs(0);

    printf("\n");

//以上是问题1的结果

//一下是问题二的结果

  // 初始化邻接矩阵

    for (int i = 0; i <MAX\_VERTICES; i++) {

        for (int j = 0; j < MAX\_VERTICES; j++) {

            matrix[i][j] = 0;

        }

    }

    // 添加边

    matrix[0][2] = 5;

    matrix[0][3] = 30;

    matrix[1][0] = 2;

    matrix[1][4] = 8;

    matrix[2][1] = 15;

    matrix[2][5] = 7;

    matrix[4][3] = 4;

    matrix[5][3] = 10;

    matrix[5][4] = 18;

    int source = 0; // 源结点

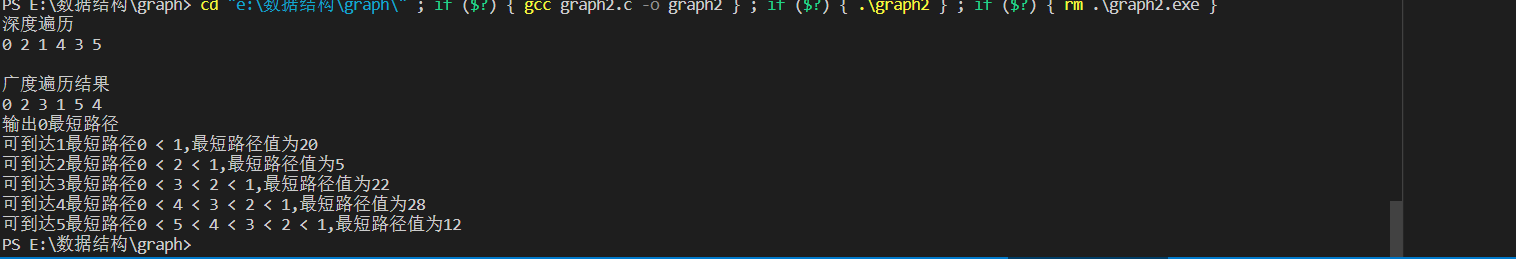
    dijkstra(source); // 求解最短路径

    printPath(source); // 输出最短路径

    return 0;

}

四、实验结果



五、实验总结

1. 为了图更有一般性，我用了一个addedge函数来实现，让图变得动态，而不是只有那么固定的静态数组
2. 对于dijkstra算法求解最短路径问题，依据我的理解，本质还是用广度收缩来搞定问题，得到一个最小生成树来输出
3. 输出的工作也比较麻烦，也写一个printpath函数来解决

