```cpp
#include"Dijkstra.h"

//构造函数
Graph_DG::Graph_DG(int vexnum, int edge) {
    //初始化顶点数和边数
    this->vexnum = vexnum;
    this->edge = edge;
    //为邻接矩阵开辟空间和赋初值
    arc = new int*[this->vexnum];
    dis = new Dis[this->vexnum];
    for (int i = 0; i < this->vexnum; i++) {
        arc[i] = new int[this->vexnum];
        for (int k = 0; k < this->vexnum; k++) {
            //邻接矩阵初始化为无穷大
                arc[i][k] = INT_MAX;
        }
    }
}
//析构函数
Graph_DG::~Graph_DG() {
    delete[] dis;
    for (int i = 0; i < this->vexnum; i++) {
        delete this->arc[i];
    }
    delete arc;
}

// 判断我们每次输入的的边的信息是否合法
//顶点从1开始编号
bool Graph_DG::check_edge_value(int start, int end, int weight) {
    if (start<1 || end<1  || start>vexnum || end>vexnum || weight < 0) {
        return false;
    }
    return true;
}

void Graph_DG::createGraph() {
    cout << "请输入每条边的起点和终点（顶点编号从1开始）以及其权重" << endl;
    int start;
    int end;
    int weight;
    int count = 0;
    while (count != this->edge) {
        cin >> start >> end >> weight;
        //首先判断边的信息是否合法
        while (!this->check_edge_value(start, end, weight)) {
            cout << "输入的边的信息不合法，请重新输入" << endl;
            cin >> start >> end >> weight;
        }
        //对邻接矩阵对应上的点赋值
        arc[start - 1][end - 1] = weight;
        //无向图添加上这行代码
        //arc[end - 1][start - 1] = weight;
        ++count;
    }
}

void Graph_DG::print() {
    cout << "图的邻接矩阵为：" << endl;
    int count_row = 0; //打印行的标签
    int count_col = 0; //打印列的标签
    //开始打印
    while (count_row != this->vexnum) {
        count_col = 0;
        while (count_col != this->vexnum) {
            if (arc[count_row][count_col] == INT_MAX)
                cout << "∞" << " ";
            else
            cout << arc[count_row][count_col] << " ";
            ++count_col;
        }
        cout << endl;
        ++count_row;
    }
}
void Graph_DG::Dijkstra(int begin){
    //首先初始化我们的dis数组
    int i;
    for (i = 0; i < this->vexnum; i++) {
        //设置当前的路径
        dis[i].path = "v" + to_string(begin) + "-->v" + to_string(i + 1);
        dis[i].value = arc[begin - 1][i];
    }
    //设置起点的到起点的路径为0
    dis[begin - 1].value = 0;
    dis[begin - 1].visit = true;

    int count = 1;
    //计算剩余的顶点的最短路径（剩余this->vexnum-1个顶点）
    while (count != this->vexnum) {
        //temp用于保存当前dis数组中最小的那个下标
        //min记录的当前的最小值
        int temp=0;
        int min = INT_MAX;
        for (i = 0; i < this->vexnum; i++) {
            if (!dis[i].visit && dis[i].value<min) {
                min = dis[i].value;
                temp = i;
            }
        }
        //cout << temp + 1 << "  "<<min << endl;
        //把temp对应的顶点加入到已经找到的最短路径的集合中
        dis[temp].visit = true;
        ++count;
        for (i = 0; i < this->vexnum; i++) {
            //注意这里的条件arc[temp][i]!=INT_MAX必须加，不然会出现溢出，从而造成程序异常
            if (!dis[i].visit && arc[temp][i]!=INT_MAX && (dis[temp].value + arc[temp][i]) < dis[i].value) {
                //如果新得到的边可以影响其他为访问的顶点，那就就更新它的最短路径和长度
                dis[i].value = dis[temp].value + arc[temp][i];
                dis[i].path = dis[temp].path + "-->v" + to_string(i + 1);
            }
        }
    }

}
void Graph_DG::print_path(int begin) {
    string str;
    str = "v" + to_string(begin);
    cout << "以"<<str<<"为起点的图的最短路径为：" << endl;
    for (int i = 0; i != this->vexnum; i++) {
        if(dis[i].value!=INT_MAX)
        cout << dis[i].path << "=" << dis[i].value << endl;
        else {
            cout << dis[i].path << "是无最短路径的" << endl;
        }
    }
}
```