

## 程序功能说明

- 1. 以**邻接多重表**为存储结构，实现连通无向图的输入，插入等操作。
- 2. 借助于队列类型，用非递归算法实现**广度优先遍历**。
- 3. 借助于栈类型，用**非递归**算法实现**深度优先遍历**。
- 4. 以邻接表为存储结构，建立**深度优先生成树**和**广度优先生成树**，并以树形输出生成树。
- 5. 利用 *dijkstra* 算法求某顶点到其他所有顶点的**最短路径**及其距离。
- 6. 利用 *floyd* 算法求任意顶点到其他所有顶点的距离。
- 7. 利用 *kruskal* 算法求**最小生成树**。
- 8. 以上功能均通过 *easyx* 库**可视化实现**，通过每个步骤之间插入 0.2s 的睡眠时间达到**过程可视化**的效果。

## 程序运行展示

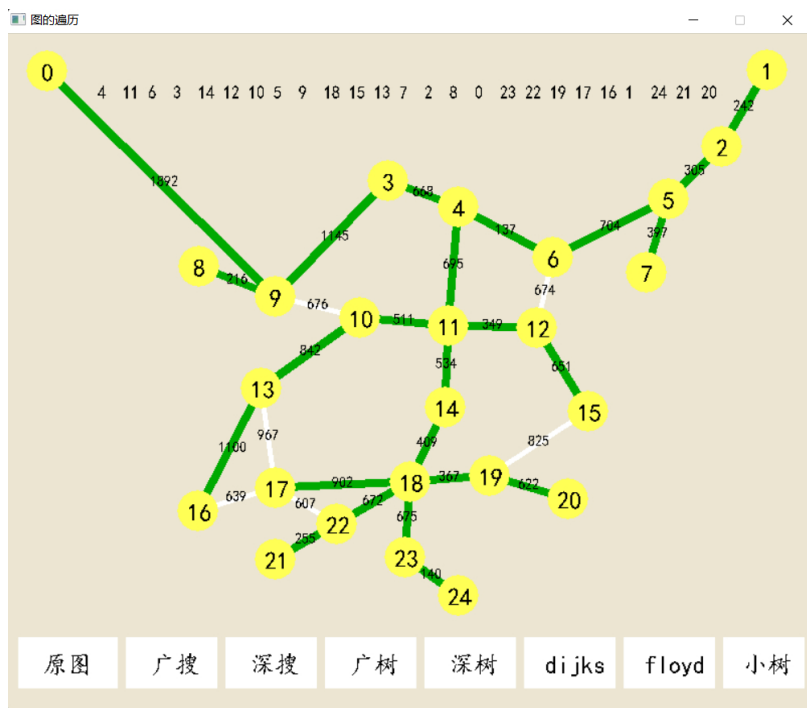
### 输入数据

```
C:\Users\linggm\Desktop\图的遍历.exe
16 17 639
17 18 902
18 19 367
19 20 622
17 22 607
18 22 672
22 21 255
18 23 675
23 24 140
0 --> 0:9 --> NULL
1 --> 1:2 --> NULL
2 --> 2:5 --> 1:2 --> NULL
3 --> 3:9 --> 3:4 --> NULL
4 --> 4:11 --> 4:6 --> 3:4 --> NULL
5 --> 5:6 --> 5:7 --> 2:5 --> NULL
6 --> 6:12 --> 4:6 --> 5:6 --> NULL
7 --> 5:7 --> NULL
8 --> 8:9 --> NULL
9 --> 9:10 --> 8:9 --> 3:9 --> 0:9 --> NULL
10 --> 10:13 --> 10:11 --> 9:10 --> NULL
11 --> 11:14 --> 11:12 --> 10:11 --> 4:11 --> NULL
12 --> 12:15 --> 11:12 --> 6:12 --> NULL
13 --> 13:17 --> 13:16 --> 10:13 --> NULL
14 --> 14:18 --> 11:14 --> NULL
15 --> 15:19 --> 12:15 --> NULL
16 --> 16:17 --> 13:16 --> NULL
17 --> 17:22 --> 17:18 --> 16:17 --> 13:17 --> NULL
18 --> 18:23 --> 18:22 --> 18:19 --> 17:18 --> 14:18 --> NULL
19 --> 19:20 --> 18:19 --> 15:19 --> NULL
20 --> 19:20 --> NULL
```

### 原图展示

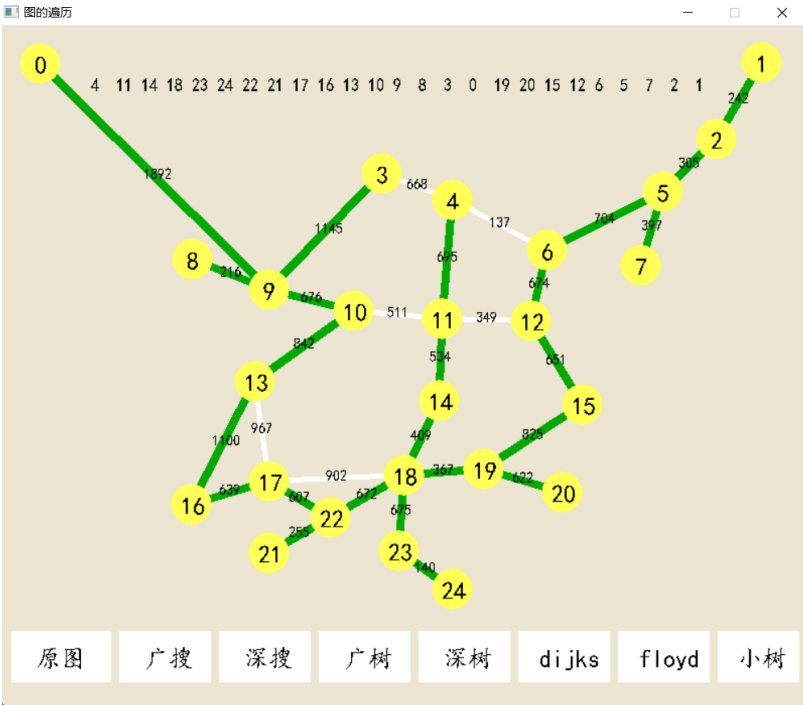


## 广度优先搜索



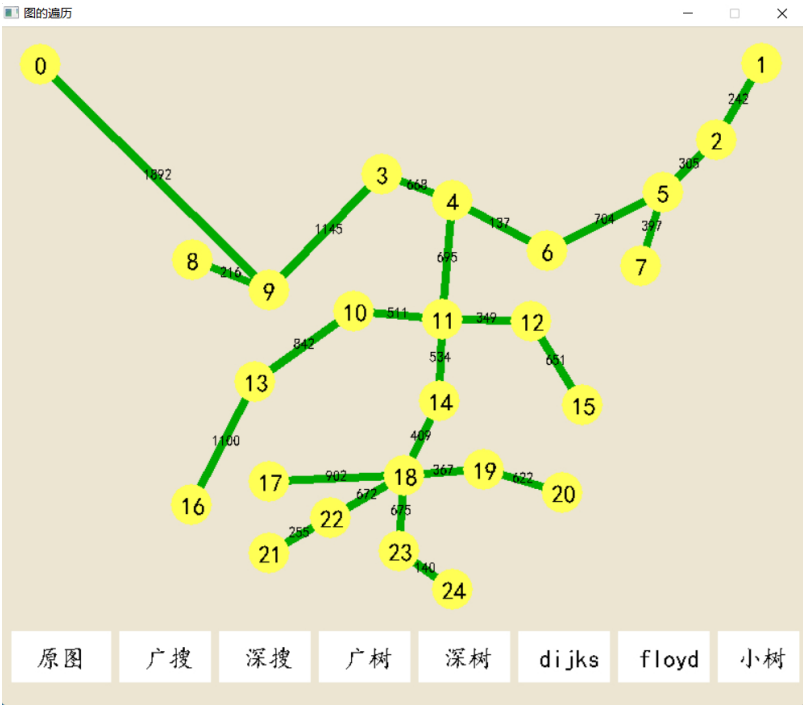
点击对应按钮，输入起点，开始搜索。

# 深度优先搜索

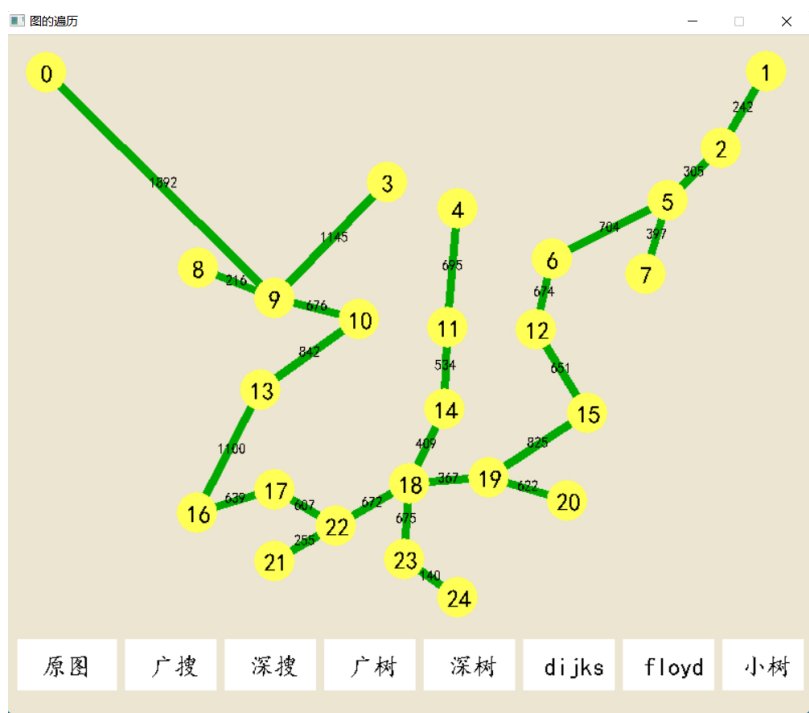


点击对应按钮，输入起点，开始搜索。

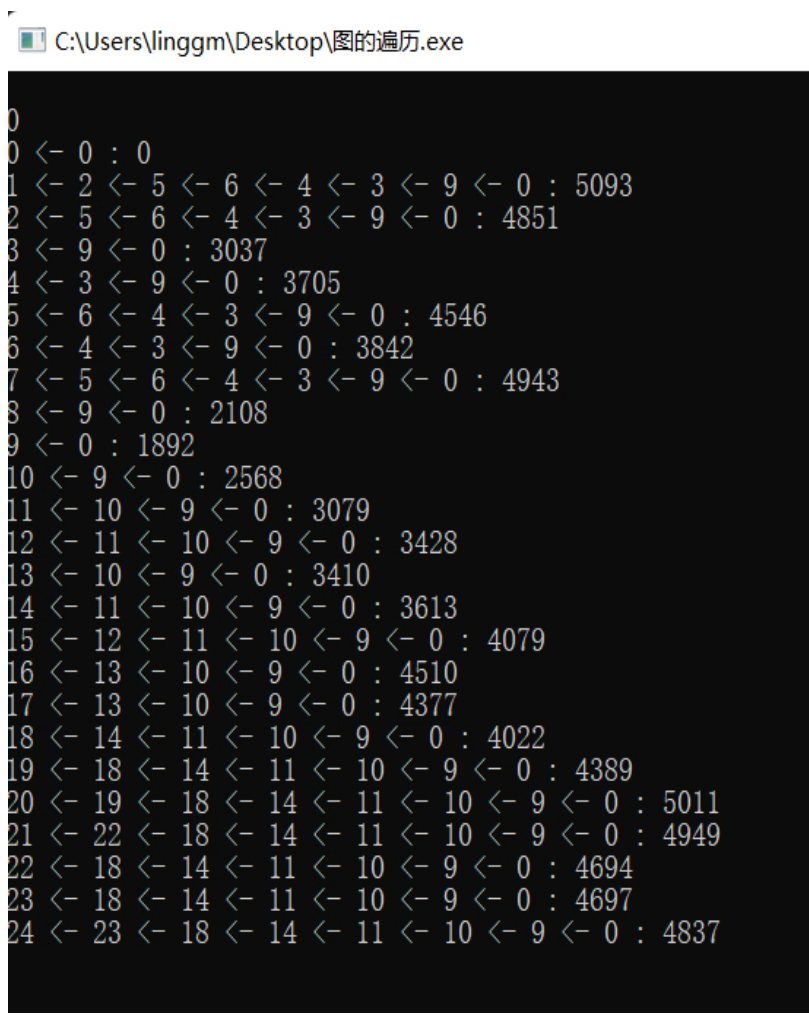
# 广度优先生成树



# 深度优先生成树



# dijkstra



```
C:\Users\linggm\Desktop\图的遍历.exe

4
0 <- 9 <- 3 <- 4 : 3705
1 <- 2 <- 5 <- 6 <- 4 : 1388
2 <- 5 <- 6 <- 4 : 1146
3 <- 4 : 668
4 <- 4 : 0
5 <- 6 <- 4 : 841
6 <- 4 : 137
7 <- 5 <- 6 <- 4 : 1238
8 <- 9 <- 3 <- 4 : 2029
9 <- 3 <- 4 : 1813
10 <- 11 <- 4 : 1206
11 <- 4 : 695
12 <- 6 <- 4 : 811
13 <- 10 <- 11 <- 4 : 2048
14 <- 11 <- 4 : 1229
15 <- 12 <- 6 <- 4 : 1462
16 <- 13 <- 10 <- 11 <- 4 : 3148
17 <- 18 <- 14 <- 11 <- 4 : 2540
18 <- 14 <- 11 <- 4 : 1638
19 <- 18 <- 14 <- 11 <- 4 : 2005
20 <- 19 <- 18 <- 14 <- 11 <- 4 : 2627
21 <- 22 <- 18 <- 14 <- 11 <- 4 : 2565
22 <- 18 <- 14 <- 11 <- 4 : 2310
23 <- 18 <- 14 <- 11 <- 4 : 2313
24 <- 23 <- 18 <- 14 <- 11 <- 4 : 2453
```

点击对应按钮，输入起点，开始搜索。

# floyd

```
选择 C:\Users\linggm\Desktop\图的遍历.exe

0 5093 4851 3037 3705 4546 3842 4943 2108 1892 2568 3079 3428 3410 3613 4079 4510 4377 4022 4389 5011 4949 4694 4697 4837
5093 0 242 2056 1388 547 1251 944 3417 3201 2594 2083 1925 3436 2617 2576 4536 3928 3026 3393 4015 3953 3698 3701 3841
4851 242 0 1814 1146 305 1009 702 3175 2959 2352 1841 1683 3194 2375 2334 4294 3686 2784 3151 3773 3711 3456 3459 3599
3037 2056 1814 0 668 1509 805 1906 1361 1145 1821 1363 1479 2663 1897 2130 3763 3208 2306 2673 3295 3233 2978 2981 3121

3705 1388 1146 668 0 841 137 1238 2029 1813 1206 695 811 2048 1229 1462 3148 2540 1638 2005 2627 2565 2310 2313 2453
4546 547 305 1509 841 0 704 397 2870 2654 2047 1536 1378 2889 2070 2029 3989 3381 2479 2846 3468 3406 3151 3154 3294
3842 1251 1009 805 137 704 0 1101 2166 1950 1343 832 674 2185 1366 1325 3285 2677 1775 2142 2764 2702 2447 2450 2590
4943 944 702 1906 1238 397 1101 0 3267 3051 2444 1933 1775 3286 2467 2426 4386 3778 2876 3243 3865 3803 3548 3551 3691
2108 3417 3175 1361 2029 2870 2166 3267 0 216 892 1403 1752 1734 1937 2403 2834 2701 2346 2713 3335 3273 3018 3021 3161

1892 3201 2959 1145 1813 2654 1950 3051 216 0 676 1187 1536 1518 1721 2187 2618 2485 2130 2497 3119 3057 2802 2805 2945

2568 2594 2352 1821 1206 2047 1343 2444 892 676 0 511 860 842 1045 1511 1942 1809 1454 1821 2443 2381 2126 2129 2269
3079 2083 1841 1363 695 1536 832 1933 1403 1187 511 0 349 1353 534 1000 2453 1845 943 1310 1932 1870 1615 1618 1758
3428 1925 1683 1479 811 1378 674 1775 1752 1536 860 349 0 1702 883 651 2802 2194 1292 1476 2098 2219 1964 1967 2107
3410 3436 3194 2663 2048 2889 2185 3286 1734 1518 842 1353 1702 0 1887 2353 1100 967 1869 2236 2858 1829 1574 2544 2684

3613 2617 2375 1897 1229 2070 1366 2467 1937 1721 1045 534 883 1887 0 1534 1950 1311 409 776 1398 1336 1081 1084 1224
4079 2576 2334 2130 1462 2029 1325 2426 2403 2187 1511 1000 651 2353 1534 0 2733 2094 1192 825 1447 2119 1864 1867 2007

4510 4536 4294 3763 3148 3989 3285 4386 2834 2618 1942 2453 2802 1100 1950 2733 0 639 1541 1908 2530 1501 1246 2216 2356

4377 3928 3686 3208 2540 3381 2677 3778 2701 2485 1809 1845 2194 967 1311 2094 639 0 902 1269 1891 862 607 1577 1717
4022 3026 2784 2306 1638 2479 1775 2876 2346 2130 1454 943 1292 1869 409 1192 1541 902 0 367 989 927 672 675 815
4389 3393 3151 2673 2005 2846 2142 3243 2713 2497 1821 1310 1476 2236 776 825 1908 1269 367 0 622 1294 1039 1042 1182
5011 4015 3773 3295 2627 3468 2764 3865 3335 3119 2443 1932 2098 2858 1398 1447 2530 1891 989 622 0 1916 1661 1664 1804
```

# kruskal



# 部分关键代码及其说明

## 存储结构

```
template <typename T>
struct EdgeNode{
    bool mark; //是否被处理过
    T vtx1;
    T vtx2;
    EdgeNode<T> *path1; //与vtx1关联的下一条边
    EdgeNode<T> *path2; //与vtx2关联的下一条边
    int weight; //权重
    EdgeNode(T v1, T v2): vtx1(v1), vtx2(v2), path1(nullptr), path2(nullptr), weight(0), mar
    EdgeNode(T v1, T v2, int w): vtx1(v1), vtx2(v2), path1(nullptr), path2(nullptr), weight(
};

template <typename T>
struct VertexNode{
    T data;
    int x, y;
    EdgeNode<T> *fout; //第一条边
    VertexNode(): data(0), x(0), y(0), fout(nullptr){}
    VertexNode(T data): data(data), x(0), y(0), fout(nullptr){}
    VertexNode(T data, EdgeNode<T> *n): data(data), x(0), y(0), fout(n){}
};

template <typename T>
struct Graph{
    vector<vector<int>>> adj; //邻接矩阵
    vector<vector<int>>> adjlist; //邻接表
    VertexNode<T> **node; // 邻接多重表
    int order; //顶点数量
    int size; //边的数量

    Graph(int o, int s): order(o), size(s){
        node = new VertexNode<T>*[o];
        adj.resize(o);
        adjlist.resize(o);
        for(int i = 0; i < order; i++){
            node[i] = new VertexNode<T>();
            adj[i].resize(o, INF);
            adj[i][i] = 0;
        }
    }
};
```

每个部分代表什么含义，在注释上有说明。

# 广度优先搜索部分代码

```
void bfs(T start){
    // 边框线条颜色函数
    setlinecolor(GREEN);
    // 边框线条形式函数
    setlinestyle(PS_SOLID, 8);
    if(start < 0 || start >= order)
        return;
    queue<T> que;
    vector<bool> visit(order, false);
    que.push(start);
    visit[start] = true;
    int counter = 0;
    settextstyle(18, 8, "楷体");
    while(!que.empty()){
        T t = que.front();
        cout << t << " ";
        outtextxy(90+25*counter++, 50, to_string(t).c_str() );
        que.pop();
        EdgeNode<T> *tmp = node[t]->fout;
        while(tmp != nullptr){

            if(tmp->vtx1 == t){
                if(!visit[tmp->vtx2]){
                    visit[tmp->vtx2] = true;
                    tmp->mark = true;
                    line(node[tmp->vtx1]->x, node[tmp->vtx1]->y, node[tmp->vtx2]->x, node[tmp->vtx2]->y);
                    Sleep(200);
                    que.push(tmp->vtx2);
                }
                tmp = tmp->path1;
            }
            else{
                if(!visit[tmp->vtx1]){
                    visit[tmp->vtx1] = true;
                    tmp->mark = true;
                    line(node[tmp->vtx1]->x, node[tmp->vtx1]->y, node[tmp->vtx2]->x, node[tmp->vtx2]->y);
                    Sleep(200);
                    que.push(tmp->vtx1);
                }
                tmp = tmp->path2;
            }
        }
    }
    cout << endl;
}
```

利用队列结构实现图的广度优先搜索（存储结构为邻接多重表）



# 深度优先搜索部分代码

```
void dfs(T start){
    // 边框线条颜色函数
    setlinecolor(GREEN);
    // 边框线条形式函数
    setlinestyle(PS_SOLID, 8);
    if(start < 0 || start >= order)
        return;
    stack<T> sta;
    vector<bool> visit(order, false);
    vector<EdgeNode<T>*> rec(order);
    for(int i = 0; i < order; i++){
        rec[i] = node[i]->fout;
    }
    sta.push(start);
    cout << start << ' ';
    visit[start] = true;
    int counter = 0;
    settextstyle(18, 8, "楷体");
    outtextxy(90+25*counter++, 50, to_string(start).c_str() );
    while(!sta.empty() ){
        T t = sta.top();
        EdgeNode<T> *tmp = rec[t];
        if(tmp == nullptr){
            sta.pop();
        }
        else{
            if(tmp->vtx1 == t){
                if(!visit[tmp->vtx2]){
                    visit[tmp->vtx2] = true;
                    tmp->mark = true;
                    sta.push(tmp->vtx2);
                    cout << tmp->vtx2 << ' ';
                    line(node[tmp->vtx1]->x, node[tmp->vtx1]->y, node[tmp->vtx2]->x, node[tmp->vtx2]->y);
                    Sleep(200);
                    outtextxy(90+25*counter++, 50, to_string(tmp->vtx2).c_str() );
                }
                rec[t] = tmp->path1;
            }
            else if(tmp->vtx2 == t){
                if(!visit[tmp->vtx1]){
                    visit[tmp->vtx1] = true;
                    tmp->mark = true;
                    sta.push(tmp->vtx1);
                    cout << tmp->vtx1 << ' ';
                    line(node[tmp->vtx1]->x, node[tmp->vtx1]->y, node[tmp->vtx2]->x, node[tmp->vtx2]->y);
                    Sleep(200);
                    outtextxy(90+25*counter++, 50, to_string(tmp->vtx1).c_str() );
                }
            }
        }
    }
}
```

```
        rec[t] = tmp->path2;
    }
}
cout << endl;
}
```

利用栈结构实现图的深度优先搜索（存储结构为邻接多重表）

# dijkstra算法

```
void dijkstra(int start){
    if(start < 0 || start >= order)
        return;
    vector<int> dist(order, INF);
    vector<int> pre(order, -1);
    vector<bool> visit(order, false);
    visit[start] = true;
    for(int i = 0; i < order; i++){
        dist[i] = adj[start][i];
        if(dist[i] < INF && dist[i] != 0)
            pre[i] = start;
    }
    for(int e = 1; e < order; e++){
        int minn = INF, min_pos = -1;
        for(int i = 0; i < order; i++){
            if(!visit[i] && dist[i] < minn){
                minn = dist[i];
                min_pos = i;
            }
        }
        if(min_pos == -1)
            break;
        visit[min_pos] = true;
        for(int i = 0; i < order; i++){
            if((long long)dist[min_pos] + adj[min_pos][i] < dist[i]){
                dist[i] = dist[min_pos] + adj[min_pos][i];
                pre[i] = min_pos;
            }
        }
    }
    for(int i = 0; i < order; i++){
        int tmp = pre[i];
        cout << i;
        if(tmp == -1){
            cout << " <- " << start;
        }
        while(tmp != -1){
            cout << " <- " << tmp;
            tmp = pre[tmp];
        }
        if(dist[i] == INF)
            cout << " : " << "INF" << endl;
        else
            cout << " : " << dist[i] << endl;
    }
}
```

利用 *dijkstra* 实现图的单源点最短路径搜索（存储结构为邻接矩阵）

## floyd算法

```
void floyd(){
    int **dist = new int*[order];
    for(int i = 0; i < order; i++){
        dist[i] = new int[order];
        for(int j = 0; j < order; j++){
            dist[i][j] = adj[i][j];
        }
    }
    for(int e = 0; e < order; e++){
        for(int i = 0; i < order; i++){
            for(int j = 0; j < order; j++){
                if((long long)dist[i][e] + dist[e][j] < dist[i][j]){
                    dist[i][j] = dist[i][e] + dist[e][j];
                }
            }
        }
    }
    for(int i = 0; i < order; i++){
        for(int j = 0; j < order; j++){
            cout << dist[i][j] << " ";
        }
        cout << endl;
    }
    for(int i = 0; i < order; i++){
        delete[] dist[i];
    }
    delete[] dist;
}
```

利用 *floyd* 实现图的多源点最短路径搜索（存储结构为邻接矩阵）

## 程序运行方式简要说明

1. 通过 *EasyX* 库开发图形化交互界面，进行图的可视化展示，**通过插入睡眠时间实现算法过程的可视化。**
2. **鼠标左键按下按键**实现对应功能。如点击“深度优先搜索”后，根据要求输入起点，然后**图形窗口逐步展示算法的过程**，最后得出结果。
3. 广度优先搜索通过**队列**实现，深度优先搜索通过**栈**实现，两种搜索都在**邻接多重表**上进行。
4. *dijkstra* , *kruskal* 和 *floyd* 算法在**邻接矩阵**上进行。
5. 输出对应优先生成树时，在原图的基础上展示。