

# 实验六

## 15331265

### 任嘉昕

#### 一:实验内容

图的存储,图的遍历.

#### 二:实验要求

- 1.编写函数分别建立图的邻接矩阵和邻接表,要求能从键盘输入图.
- 2.编写图的 DFS 和 BFS 算法
- 3.编写主函数测试(使用字符单选形式)

#### 三:实验思路

首先构造点类,一个点的数据域有:存储的数据值(**data**),在图中的编号(**order**),存邻接的点数据(**next**)(为了将来删除方便使用了存储数据而没有存编号虽然本次实验没有用到删除函数),权值大小(**weight**).

然后分别构造邻接矩阵的图类和邻接链表的图类以及对应的 **DFS** 和 **BFS**.邻接矩阵的边通过一个 **matrix** 来存储,而邻接链表的边关系通过对应点的一个链表存.

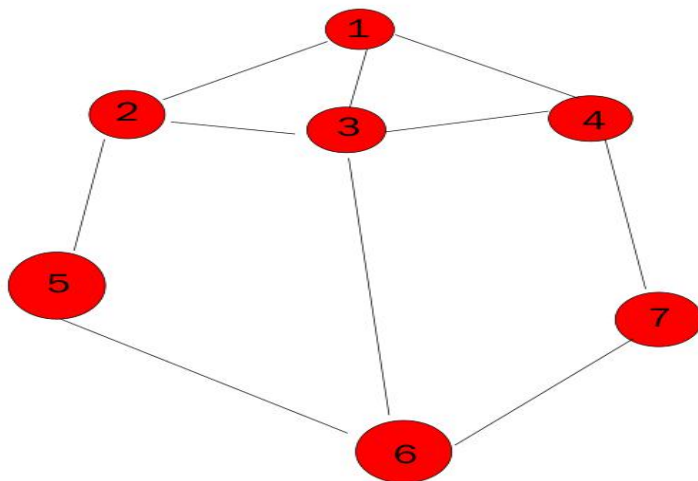
在 **BFS** 中使用队列遍历时考虑到有不连通图的情况,在最外围有添加了一层循环判断是否全部已经遍历过,否则就进入下一个子图遍历.

**DFS** 中为了考虑不连通图,每次进入函数时多增加一个参数 **part** 用

来记录第一个用来遍历的节点,当回溯到第一个节点但是却没有全部遍历的时候就进入下一个子图。

实验测试:

标准测试图如下:



分析:

图中圆形内 1 2 3 4 5 6 7 代表数据

他们的编号分别为 0 1 2 3 4 5 6.

默认从编号 0 开始遍历且按相邻元素编号大小顺序判断  
则

DFS 应得到

(1)(2 3 4 7 6 5)

BFS 得到

(1)(2 3 4)(5 6 7)

注 ()内部的数字顺序可以出现交换

实际运行

首先是邻接矩阵

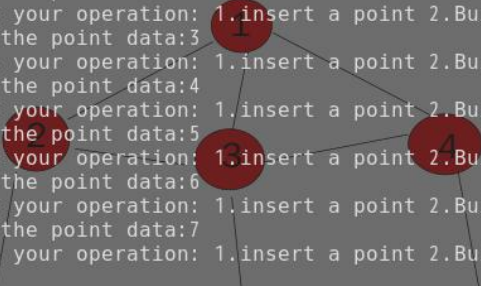
```
[rjx@JX graph]$ ./a.out
Select the build form : 1.Matrix 2.List 3.exit 1
Start test Matrixed Graph
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:□
```

输入点

```

Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:1
Input the point data:1
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:1
Input the point data:2
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:1
Input the point data:3
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:1
Input the point data:4
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:1
Input the point data:5
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:1
Input the point data:6
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:1
Input the point data:7
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:

```

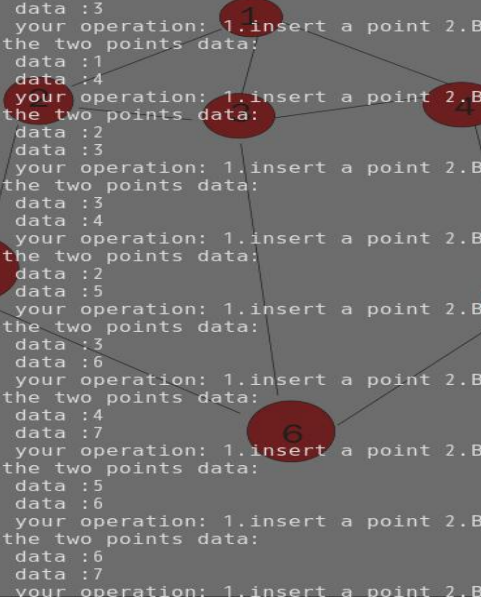


输入边(使用边上两点的数据)

```

Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:2
Input the two points data:
Point1 data :1
Point2 data :2
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:2
Input the two points data:
Point1 data :1
Point2 data :3
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:2
Input the two points data:
Point1 data :1
Point2 data :4
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:2
Input the two points data:
Point1 data :2
Point2 data :3
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:2
Input the two points data:
Point1 data :3
Point2 data :4
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:2
Input the two points data:
Point1 data :2
Point2 data :5
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:2
Input the two points data:
Point1 data :3
Point2 data :6
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:2
Input the two points data:
Point1 data :4
Point2 data :7
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:2
Input the two points data:
Point1 data :5
Point2 data :6
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:2
Input the two points data:
Point1 data :6
Point2 data :7
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:

```



遍历

```

Select the method of travel: (B) => BFS , (D) => DFS :B
1234567
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:3
Select the method of travel: (B) => BFS , (D) => DFS :D
1234765
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:

```

和预想的一致。

测试邻接表

```

Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:4
MatrixG test end
Select the build form : 1.Matrix 2.List 3.exit:2
Start test Listed Graph
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:

```

输入点和边操作和上面的一致(截取了部分输入)

```
Start test Listed Graph
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:1
Input the point data:1
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:1
Input the point data:2
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:1
Input the point data:3
```

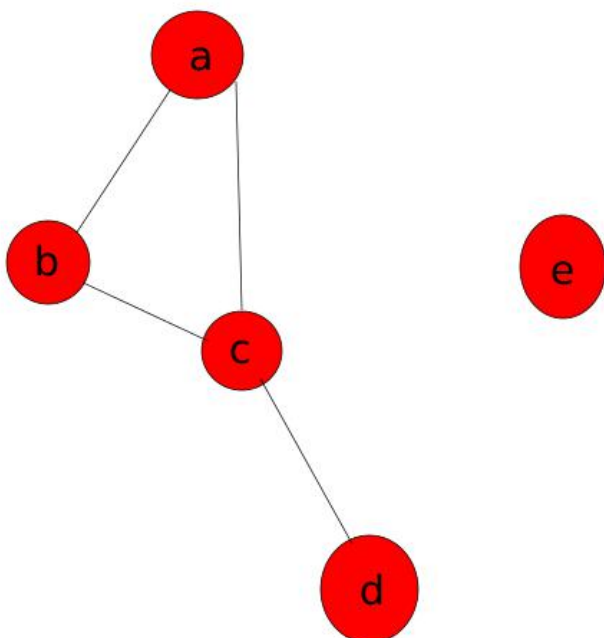
```
POINT2 data :7
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:2
Input the two points data:
Point1 data :5
Point2 data :6
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:2
Input the two points data:
Point1 data :5
```

遍历

```
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:3
Select the method of travel: (B) => BFS , (D) => DFS :B
1234567
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:3
Select the method of travel: (B) => BFS , (D) => DFS :D
1234765
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:
```

和刚才的一致完成标准测试.

非标准测试



图中 a b c d e 编号分别为 0 1 2 3 4  
遍历结果为

```

Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:2
Input the two points data:
Point1 data :c
Point2 data :d
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:3
Select the method of travel: (B) => BFS , (D) => DFS :D
abcde
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:3
Select the method of travel: (B) => BFS , (D) => DFS :B
abcde
Detect your operation: 1.insert a point 2.Build an edge 3.travel the graph 4.exit:

```

可见可以实现非连通图遍历。

## 实验总结

这次实验内容不难但是还是有很多地方值得探讨,比如开始的时候应当想到使用继承机制实现 **2** 个图类,就没有必要写大量的冗余代码,图的遍历的算法也很类似没有必要写两个.非联通图的处理还可以使用虚根来解决不用每次遍历完一个部分再遍历节点一次找没有遍历的节点。