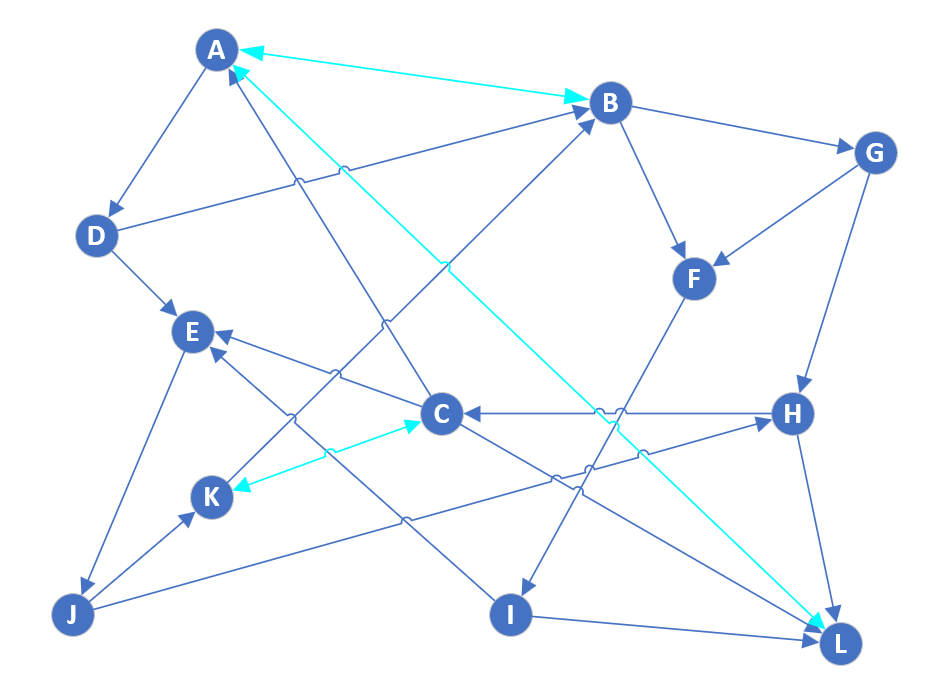
张宇杰 2022113573

以下程序均以该图作为输入：蓝色为单向边，浅蓝色为双向边

图形用Visio Professional绘制



输入序列如下：

可见于文件./my-graph/graph\_1.txt

1. +v A
2. +v B
3. +v C
4. +v D
5. +v E
6. +v F
7. +v G
8. +v H
9. +v I
10. +v J
11. +v K
12. +v L
13. +e A D 1
14. +e C A 1
15. +e A L 1
16. +e L A 1
17. +e A B 1
18. +e B A 1
19. +e D B 1
20. +e K B 1
21. +e B F 1
22. +e B G 1
23. +e C E 1
24. +e C K 1
25. +e K C 1
26. +e H C 1
27. +e C L 1
28. +e D E 1
29. +e E J 1
30. +e I E 1
31. +e F I 1
32. +e G F 1
33. +e G H 1
34. +e H L 1
35. +e J H 1
36. +e J K 1
37. +e I L 1

**一**、分别实现无向图（或有向图）的邻接矩阵和邻接表存储结构的建立算法，分析和比较其建立算法的时间复杂度以及存储结构的空间占用情况

**五**、以适当的方式输入图的顶点和边，并显示相应的结果。要求顶点不少于 10个，边不少于 13 条（图的规模越大越好）

**有向图，邻接矩阵**

运行程序：

1. #include <iostream>
2. #include <fstream>
3. #include ".\src\MyGraph"
4. using namespace MyGraph;
5. using namespace std;
6. int main()
7. {
8. system("chcp 65001");
9. ifstream command("./graph\_1.txt");
10. AdjacencyMatrixGraph G1;
11. G1.commandSequence(command);
12. G1.showMatrix();
13. cout << endl;
14. }

运行结果：

日历

描述已自动生成

分析：

时间复杂度：每加一次边，对邻接矩阵进行一次修改，修改是O(1)的，故为O(E)

空间占用：显然为O(V2)

**有向图，邻接表**

运行程序：

1. #include <iostream>
2. #include <fstream>
3. #include ".\src\MyGraph"
4. using namespace MyGraph;
5. using namespace std;
6. int main()
7. {
8. system("chcp 65001");
9. ifstream command("./graph\_1.txt");
10. AdjacencyListGraph G1;
11. G1.commandSequence(command);
12. G1.showList();
13. cout << endl;
14. }

运行结果：

图片包含 表格

描述已自动生成

分析：

时间复杂度：每加一次边，向对应邻接表插入一条边，而vector尾插的均摊时间复杂度为O(1)，故整体时间复杂度为O(E)

空间占用：显然为O(V+E)

**二**、实现无向图（或有向图）的邻接矩阵和邻接表两种存储结构的相互转换算法

**邻接矩阵转邻接表**

运行程序：

1. #include <iostream>
2. #include <fstream>
3. #include ".\src\MyGraph"
4. using namespace MyGraph;
5. using namespace std;
6. int main()
7. {
8. system("chcp 65001");
9. ifstream command("./graph\_1.txt");
10. AdjacencyMatrixGraph G1;
11. G1.commandSequence(command);
12. G1.showMatrix();
13. cout << endl;
14. AdjacencyListGraph G2 = matrixToList(G1);
15. G2.showList();
16. cout << endl;
17. }

运行结果：

文本

描述已自动生成

结果正确

分析：时间复杂度为O(V2)

**邻接表转邻接矩阵**

运行程序：

1. #include <iostream>
2. #include <fstream>
3. #include ".\src\MyGraph"
4. using namespace MyGraph;
5. using namespace std;
6. int main()
7. {
8. system("chcp 65001");
9. ifstream command("./graph\_1.txt");
10. AdjacencyListGraph G1;
11. G1.commandSequence(command);
12. G1.showList();
13. cout << endl;
14. AdjacencyMatrixGraph G2 = listToMatrix(G1);
15. G2.showMatrix();
16. cout << endl;
17. }

运行结果：

文本

描述已自动生成

结果正确

分析：时间复杂度为O(V+E)

**三**、在上述两种存储结构上，分别实现无向图（或有向图）的深度优先搜索(递归和非递归)和广度优先搜索算法。并以适当的方式存储和展示相应的搜索结果，包括：深度优先或广度优先生成森林（或生成树）、深度优先或广度优先序列和深度优先或广度优先编号。并分析搜索算法的时间复杂度和空间复杂度。

**P.S. 在本次程序设计中，邻接表存储的图AdjacencyListGraph与邻接矩阵存储的图AdjacencyMatrixGraph均共用了父类Graph的各种搜索方法**

**故以下的程序以及运行结果中，仅展示对邻接表存储的图AdjacencyListGraph的搜索结果（因为两种存储结构的代码一致，结果也一致）**

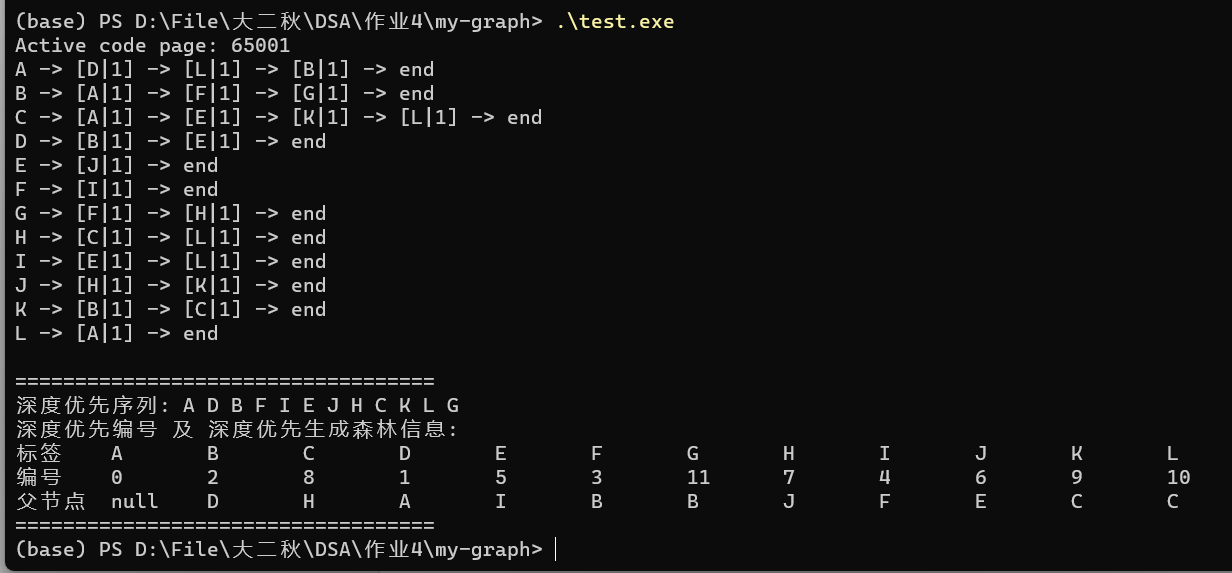
**仅在分析复杂度时讨论二者的不同**

**深度优先搜索（递归）**

运行程序：

1. #include <iostream>
2. #include <fstream>
3. #include ".\src\MyGraph"
4. using namespace MyGraph;
5. using namespace std;
6. int main()
7. {
8. system("chcp 65001");
9. ifstream command("./graph\_1.txt");
10. AdjacencyListGraph G1;
11. G1.commandSequence(command);
12. G1.showList();
13. cout << endl;
14. G1.dfs();
15. cout << endl;
16. }

输出结果：



转换为图形：

图表

描述已自动生成

图表

描述已自动生成

分析：

*时间复杂度*：分为两个部分：访问每个节点花费的时间，以及在每个节点找邻居花费的时间

**对邻接矩阵**：

1：V个节点需要O(V)

2：对邻接矩阵，对于节点i，需要扫描第i行的每一个元素才能找到所有的邻居，需要O(V)

故总复杂度O(V^2)

**对邻接表：**

1：n个节点需要O(V)

2：对邻接表，总共的找邻居时间复杂度就是遍历边表的时间复杂度。对于有向图：O(V)

故总复杂度O(V+E)

*空间复杂度：*

无论是邻接矩阵还是邻接表，递归工作栈的最大深度为V，故空间复杂度为O(V)

**深度优先搜索（非递归）**

运行程序：

1. #include <iostream>
2. #include <fstream>
3. #include ".\src\MyGraph"
4. using namespace MyGraph;
5. using namespace std;
6. int main()
7. {
8. system("chcp 65001");
9. ifstream command("./graph\_1.txt");
10. AdjacencyListGraph G1;
11. G1.commandSequence(command);
12. G1.showList();
13. cout << endl;
14. G1.dfs\_no\_rec();
15. cout << endl;
16. }

运行结果：

文本

描述已自动生成

与递归实现的dfs一致

分析：复杂度同递归实现的dfs

**广度优先搜索**

运行程序：

1. #include <iostream>
2. #include <fstream>
3. #include ".\src\MyGraph"
4. using namespace MyGraph;
5. using namespace std;
6. int main()
7. {
8. system("chcp 65001");
9. ifstream command("./graph\_1.txt");
10. AdjacencyListGraph G1;
11. G1.commandSequence(command);
12. G1.showList();
13. cout << endl;
14. G1.bfs();
15. cout << endl;
16. }

运行结果：

文本

描述已自动生成

转换为图形：

图表

描述已自动生成

图表, 雷达图

描述已自动生成

分析：

*时间复杂度*：两部分：访问每个节点花费的时间，在每个节点找邻居花费的时间

**对邻接矩阵：**

1：V个节点需要O(V)；

2：由于是邻接矩阵，对于节点i，需要扫描第i行的每一个元素，需要O(V)；

总复杂度O(V^2)

**对邻接表：**

1：n个节点需要O(V)

2：对邻接表，总共的找邻居时间复杂度就是遍历边表的时间复杂度。对于有向图：O(V)

故总复杂度O(V+E)

*空间复杂度：*

无论是邻接矩阵还是邻接表，工作队列的最大长度为V，故空间复杂度为O(V)

**四**、（2）对于有向图，采用“邻接表”存储结构，设计和实现计算每个顶点入度、

出度和度的算法，并分析其时间复杂度

运行程序：

1. #include <iostream>
2. #include <fstream>
3. #include ".\src\MyGraph"
4. using namespace MyGraph;
5. using namespace std;
6. int main()
7. {
8. system("chcp 65001");
9. ifstream command("./graph\_1.txt");
10. AdjacencyListGraph G1;
11. G1.commandSequence(command);
12. G1.showList();
13. cout << endl;
14. G1.showDegree();
15. cout << endl;
16. }

运行结果：

文本

描述已自动生成

绘制在图上：

蓝色的球

低可信度描述已自动生成

分析：

时间复杂度：

要想知道一个节点的入度与出度，对于邻接表来说，必须遍历整个边表才能得知一个节点的入度与出度。时间复杂度为O(E)

当然，求解所有节点的入度与出度的时间复杂度也是O(E)，需要占用空间O(V)