|  |  |
| --- | --- |
| **数据结构与算法 实验报告** | |
| 第四次 | |
| 图片包含 标牌  已生成极高可信度的说明 | |
|  | |
|  | |
|  | |
| **姓名** | 宋郅博 |
| **班级** | 强基数学2102 |
| **学号** | 221311457 |
| **电话** | 13561313919 |
| **Email** | 2105602479@qq.com |

任务一

建立为实现该游戏的图的抽象描述结构，包括图中顶点的意义以及存储的信息、边的意义以及存储的信息，并给出该图的逻辑示意图。

|  |
| --- |
| 顶点：表示单词，存储单词字符串。 |
| 边：表示两个单词之间可以转换（相差一个字母），无向边无需额外信息。 |
| 逻辑示意图：如 COLD → CO**R**D → C**A**RD → **W**ARD → WAR**M** |

任务二

在任务 1 的基础上，结合教材中图的抽象数据类型的定义，设计并实现一个为该游戏而使用的具体的 Graph Class。

由任务一可知，要创建的图应该是一个无向图。两个单词只有在相差一个字母的时候有边，若使用邻接矩阵来确认是否相邻，这个矩阵大概率是个稀疏矩阵，会造成很大的浪费，所以最好使用邻接表来存储相邻单词信息

以下是class中的操作（hw\_4\work2\Graph.java）：

|  |
| --- |
| addWord：向邻接表中添加一个单词 |
| addEdge：添加边的方法，参数为两个字符串，分别表示两个单词 |
| getNeighbors：获取给定单词的邻居 |
| getAllWords：获取所有单词 |
| buildGraph：根据给定的单词集合构建图 |
| generateNeighbors：生成与给定单词相邻的单词列表 |
| bfsShortestPath：使用广度优先搜索算法找到从start到end的最短路径 |

任务三

该任务中会提供一个所有长度为 5 的单词列表文件 words5.txt，需要针对提供的这个单词列表解决如下问题：

① 针对 words5.txt 文件中的单词列表，生成一个 noladder.txt 文件，该文件中记录的单词是无法和其他单词形成字梯的所有单词。

② 编写一个具有交互功能的程序，给用户随机抽两个单词（注：这两个单词必须要保证能够有字梯链），接受用户的输入，判断用户的每次输入是否是正确的，直到用户失败或者成功。（如果可能，还可以增加判断用户的成功输入是否是最短的变化链路）

3.1 问题一

创建了Task3a.java文件，在文件中导入words5.txt，生成了noladder.txt。代码如下：

package work3;

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.util.\*;

public class Task3a {

    public static void main(String[] args) throws IOException {

        Set<String> words = new HashSet<>(Files.readAllLines(Paths.get("C:\\Users\\Thomas\\Desktop\\hw\_4\\work3\\words5.txt")));

        Graph graph = Graph.buildGraph(words);

        List<String> noLadderWords = new ArrayList<>();

        for (String word : graph.getAllWords()) {

            if (graph.getNeighbors(word).isEmpty()) {

                noLadderWords.add(word);

            }

        }

        Files.write(Paths.get("noladder.txt"), noLadderWords);

    }

}

3.2 问题二

代码如下：

package work3;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.util.\*;

public class Task3b {

    public static void main(String[] args) throws IOException {

        Set<String> words = new HashSet<>(Files.readAllLines(Paths.get("C:\\Users\\Thomas\\Desktop\\hw\_4\\work3\\words5.txt")));

        Graph graph = Graph.buildGraph(words);

        WordLadderGame game = new WordLadderGame(graph);

        game.startNewGame();

        System.out.println("Start word: " + game.getStartWord());

        System.out.println("End word: " + game.getEndWord());

        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        List<String> userPath = new ArrayList<>();

        userPath.add(game.getStartWord());

        String currentWord = game.getStartWord();

        while (true) {

            System.out.println("Current word: " + currentWord);

            System.out.print("Next word: ");

            String nextWord = reader.readLine().trim().toLowerCase();

            if (!words.contains(nextWord)) {

                System.out.println("Invalid word. Try again.");

                continue;

            }

            if (!graph.getNeighbors(currentWord).contains(nextWord)) {

                System.out.println("Not a valid adjacent word. Try again.");

                continue;

            }

            if (userPath.contains(nextWord)) {

                System.out.println("Word already used. Try again.");

                continue;

            }

            userPath.add(nextWord);

            currentWord = nextWord;

            if (currentWord.equals(game.getEndWord())) {

                System.out.println("Congratulations! You reached the end.");

                if (game.isShortest(userPath)) {

                    System.out.println("You used the shortest path!");

                } else {

                    System.out.println("But it's not the shortest path.");

                }

                break;

            }

        }

    }

}

class WordLadderGame {

    private Graph graph;

    private String startWord;

    private String endWord;

    private List<String> shortestPath;

    public WordLadderGame(Graph graph) {

        this.graph = graph;

    }

    public void startNewGame() {

        List<String> words = new ArrayList<>(graph.getAllWords());

        Random random = new Random();

        while (true) {

            int startIndex = random.nextInt(words.size());

            int endIndex = random.nextInt(words.size());

            if (startIndex == endIndex) continue;

            startWord = words.get(startIndex);

            endWord = words.get(endIndex);

            shortestPath = graph.bfsShortestPath(startWord, endWord);

            if (!shortestPath.isEmpty()) {

                break;

            }

        }

    }

    public String getStartWord() {

        return startWord;

    }

    public String getEndWord() {

        return endWord;

    }

    public boolean isShortest(List<String> userPath) {

        return userPath.size() == shortestPath.size();

    }

}