# 哈尔滨工业大学计算机科学与技术学院

# 2017 年秋季学期《软件工程》

# Lab 1：结对编程


| 姓名 | 班级/学号 | 联系方式 |
| --- | --- | --- |
| 马玉坤 | 36101/1150310618 | mayukuner@126.com/18845895386 |
| 张宁 | 36101/1150310607 | 3294349775@qq.com/17390605885 |

# 目 录

# 1 实验要求

实现从文本文件中读取数据并根据要求生成图结构，可视化展示有向图，并在其上进行查询桥接词，计算最短路径，随机游走和利用桥接词生成新文本，并且实时展示各操作的结果。于此同时在主函数之中，需要实现固定功能的几个函数。

# 2 待求解问题描述与数学模型

## 2.1 读入文本并生成有向图

- 输入数据：输入文件路径（通过文件框选择或者手动输入）
- 输出数据：由文本文件生成的有向图
- 约束条件：
  1. 有向图中每个节点对应一个单词
  2. 单词不区分大小写
  3. 仅当单词 w1 和单词 w2 在文本文件中相邻时，w1 与 w2 有连边，且边权为 w1 与 w2 相邻的次数展示有向图

## 2.2 存储有向图（可选功能 1）

- 输入数据：存储图像路径及待存储情况
- 输出数据：图像文件，并保存在磁盘中
- 约束条件：用户已打开输入文本文件且输出文件路径合法

## 2.3 查询桥接词

- 输入数据：图结构及待查询的两个单词 word1 与 word2
- 输出数据：两个单词在图中的所有桥接词
- 约束条件：
  1. 当两个单词在图中不存在时，输出提示信息
  2. 当两个单词在图中存在时，输出所有的 word3，使得 word1 与 word3 有连边，word3 与 word2 有连边

## 2.4 根据 bridge word 生成新文本

- 输入数据：图结构及待查询的文本
- 输出数据：在查询文本中的相邻单词之间插入桥接词并输出新的文本

- 约束条件：
  1. 相邻单词无桥接词时，两个单词之间不插入任何单词
  2. 当两个单词有一个桥接词时，两个单词间插入该桥接词
  3. 当两个单词有多个桥接词时，两个单词间插入这些桥接词中的随机选择的任意一个
  4. 单词不存在时，输出提示信息

## 2.5 计算两个单词之间的最短路径

- 输入数据：图结构及待查询的两个单词 word1 和 word2
- 输出数据：在输入文本的生成图中，查询 word1 到 word2 对应的节点的最短路径，并显示最短路径、最短路径长度和不同的最短路径的数目
- 约束条件：
  1. 当单词在图中不存在时，输出提示信息
  2. 当 word1 与 word2 不可达时，输出提示信息
  3. 当 word1 与 word2 有多条最短路时，输出字典序最小的一条最短路径

## 2.6 计算出两个单词之间的所有的最短路径（可选功能 3）

- 功能说明：由于图中两点的最短路径数目为指数级别，因此我们没有直接全部显示所有路径，而是实现了按照字典序从小到大的顺序查看所有最短路径的功能
- 输入数据：图结构及待查询的两个单词 word1 和 word2 以及字典序顺序 k
- 输出数据：在输入文本的生成图中，查询 word1 到 word2 对应的节点的最短路径，并显示字典序第 k 小的最短路
- 约束条件：
  1. 当单词在图中不存在时，输出提示信息
  2. 当 word1 与 word2 不可达时，输出提示信息

## 2.7 随机游走

- 输入数据：生成图
- 输出数据：从随机选择的起点开始随机游走所经过的单词构成的文本
- 约束条件：
  1. 起点随机选择
  2. 每条边只访问一次
  3. 到达某个点时，随机选择一条未访问过的该点的出边进行遍历
  4. 当某个点的出边没有未访问过的边时，游走过程结束

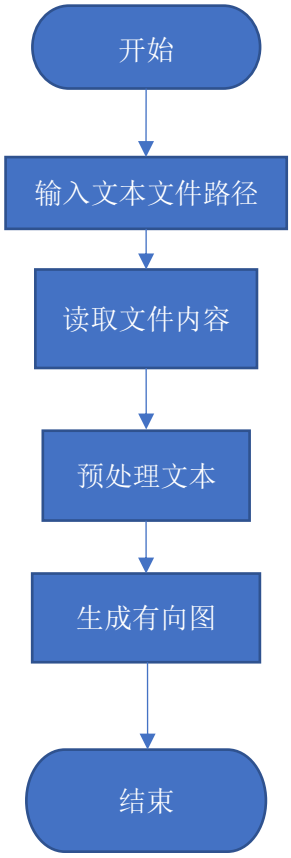
# 3 算法与数据结构设计

## 3.1 设计思路与算法流程图

### 3.1.1 读入文本并生成有向图

首先根据输入文件路径获得输入文件的内容，然后使用 String 类的 replaceAll()函数与正则表达式，将除了字母与空格以外的字符替换成空格，然后将连续的空格合并为一个空格，之后使用 trim()方法删除两端空格。之后使用 split()方法，将文本用空格分割成单词，从而将整个文本文件拆分成单词的序列。

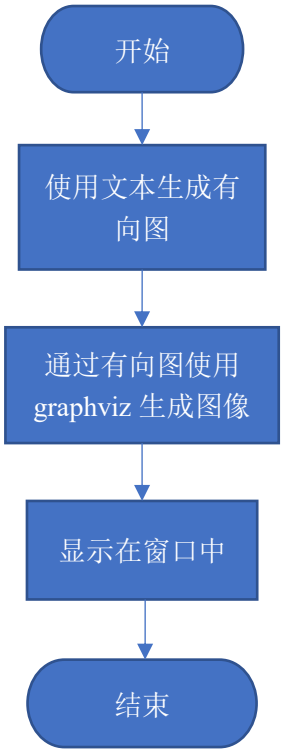
之后遍历单词序列，设两个相邻的单词分别为 word1 和 word2，首先查询 word1-word2 这条边是否已经在图中，如果已经在图中，那么修改 word1 与 word2 的连边边权即可，否则新建一条 word1 连向 word2 的边。




### 3.1.2 展示有向图

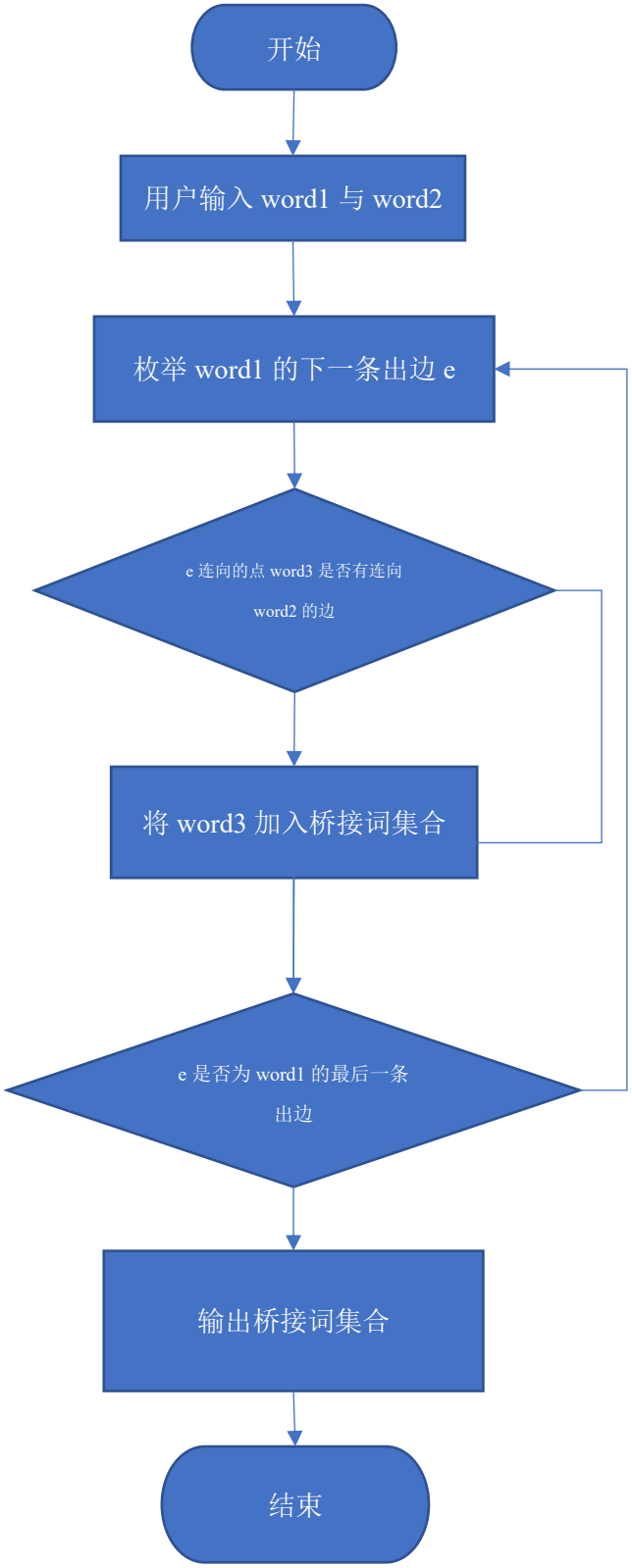
将自定义的图，调用 graphviz-java 的构造方法，生成一张图，并在窗口中显示给用户。由于 graphviz-java 渲染图像的效率十分低，因此仅当用户打开用于图像显示的窗口时，程序

才会调用 graphviz-java 进行图像的渲染并显示。当用于图像显示的窗口关闭时，程序不会调用 graphviz-java 进行图像的渲染。




### 3.1.3 查询桥接词

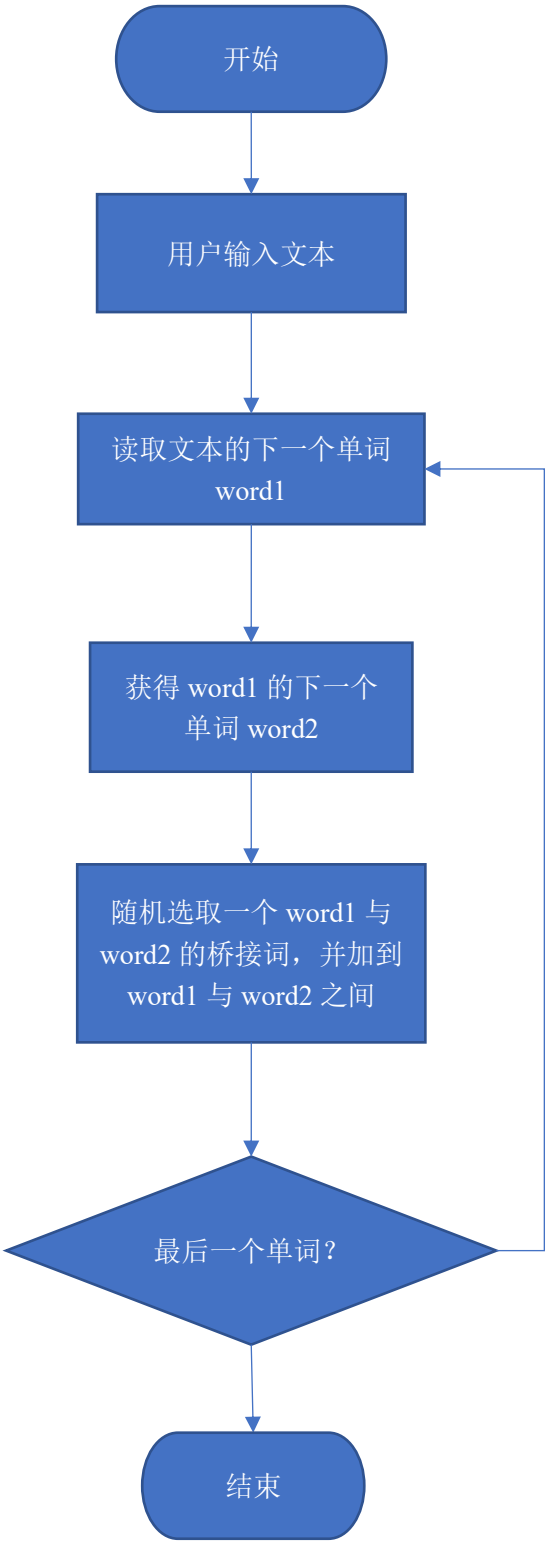
设输入的两个单词为 word1 和 word2。遍历 word1 连出的边(word1, word3)，判断 word3 是否有连向 word2 的边。




### 3.1.4 根据 bridge word 生成新文本

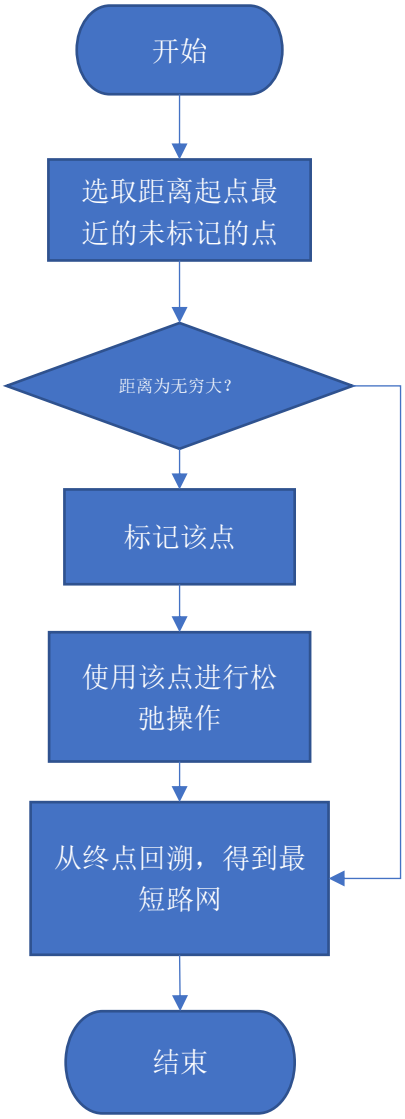
读入文本，使用“读入文本并生成有向图”中的方法，将读入的文本格式化，得到单词

序列。之后，遍历单词序列，调用查询桥接词方法，获得单词序列中任意相邻的单词的所有桥接词，并随机选择一个插入到两个单词之间。由此来生成新文本。



### 3.1.5 计算两个单词之间的最短路径

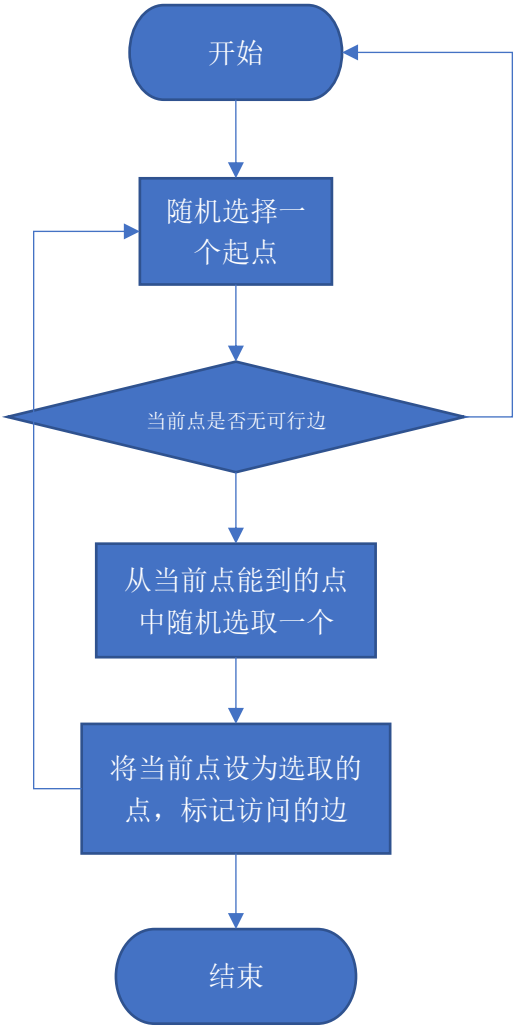
在输入文件的生成图中，设起点为 s，终点为 t。使用迪杰斯特拉（Dijkstra）算法求 s 到 t 的最短路。在求解最短路过程中，需要记录每个点的可选前驱（p 为 u 的可选前驱，当且仅当 s 到 p 的最短路长度加上 p 到 u 的边长等于 s 到 u 的最短路长度），由此可以生成最短路径并求出最短路长度以及不同的最短路径的个数。




### 3.1.6 随机游走

当用户点击“开始/重新开始”时，在图所有的点中，随机找到一个点，并以此为起点开始随机游走。当用户点击“下一步”时，程序在该点的出边中，随机选取一条没有访问过的

边进行遍历，直到到达一个没有未访问过的出边的点为止。




## 3.2 数据结构设计

## 3.2.1 生成图结构

### 3.2.1.1 WordGraph

```
/**
 *
 *WordGraph 类用于存储文本生成的有向图
 *nodes记录了单词到节点对象的对应关系
 *WordGraph的构造函数初始化nodes为空HashMap，并初始化颜色为黑色
 */
public class WordGraph {
```

```
    public HashMap<String, WordNode> nodes;
    public String color;
    public WordGraph();
}
```

#### 3.2.1.2 WordNode

```
/**
*
*WordNode 类是存储图像中的节点的
*除了一个类变量，每一个WordNode对象包含七个属性，三个方法
*属性HashMap记录了和本节点相邻单词到节点对象的对应关系
*text 记录节点的单词文本
*color 记录节点的颜色
*node 用来传递给graphviv-java库从而绘制图像
*pre 记录最短路算法之中节点的可行前驱
*distance 记录了每个顶点距离起点的最短路径的距离
*visited 记录节点在最短路算法中是不是已经被访问过了
*countPaths 从该点到达终点的不同的最短路数目
*addEdge 添加一条边
*hasEdge 判断一条边是否存在
*compareTo 比较函数，排序用
*/
public class WordNode implements Comparable<WordNode> {
    static final int INF = Integer.MAX_VALUE;
    public HashMap<String, WordEdge> edges;
    public String text;

    // for painting
    public Color color;
    public Node node;

    // for calculating the shortest path
    public HashSet<WordNode> pre;
    public TreeSet<WordNode> suc;
    public int distance;
    public boolean visited;
    public int countPaths;


    public WordNode(String _text);

    public void addEdge(WordNode v);
    public boolean hasEdge(String text);
```

```
    public int compareTo(WordNode rhs);
}
```

#### 3.2.1.3 WordEdge

```
/**
 *
 *WordEdge 类是存储图像中的边
 *每一个WordEdge对象包含四个属性，两个方法
 *to 记录有向边指向的节点
 *weight记录边上的权重
 *color 记录边的颜色
 *visited 记录边在随机游走过程之中是不是被访问过了
 */
public class WordEdge {
    public WordNode to;
    public int weight;
    public Color color;
    public boolean visited;

    public WordEdge(WordNode _to);

    public void incWeight();
}
```

## 3.3 算法时间复杂度分析

### 3.3.1 读入文本并生成有向图

这个部分的时间复杂度为线性时间复杂度，与读入的文本大小同阶。对于每对相邻的单词，需要使用 HashSet 修改或新建一条边，由于 HashSet 查询和修改的时间复杂度为 O(1)，所以设文本大小为 n，则时间复杂度为 O(n)。

### 3.3.2 存储有向图（可选功能 1）

有向图存储过程需要调用 graphviz-java 库，设 graphviz-java 库生成图像的复杂度为 O(f(n))，那么该过程的时间复杂度则为 O(f(n))。

### 3.3.3 查询桥接词

设输入的单词为 word1 和 word2，那么遍历 word1 的所有出边的代价为 O(|e[word1]|)，其中

e[word]表示 word 表示的节点的出边集合，“||”表示绝对值符号。对于每条出边(word1, word3)，查询 word3 是否有连向 word2 的边的时间复杂度为 O(1)。因此该过程的总时间复杂度为 O(|e[word1]|)。

### 3.3.4 根据 bridge word 生成新文本

设输入的文本的单词数目为 k，设图中的点数为 n，边数为 m。那么该过程需要查询对这 k-1 对相邻的单词查询 bridge word，总时间复杂度平均情况下为 O(km/n)，其中 m/n 为生成图中点的平均出度，O(m/n)即随机选一个点并遍历其出边的复杂度。最坏情况下，时间复杂度为 O(km)。

### 3.3.5 计算两个单词之间的最短路径

在该过程中，我们使用了未加堆优化的迪杰斯特拉算法，求解 s 出发的单源最短路的时间复杂度为 O(n^2)，其中 n 为图中顶点的个数。

### 3.3.6 计算出两个单词之间所有最短路径字典序第 k 小的路径（可选功能 3）

在该过程中，我们使用了动态规划算法，再求解完最短路径后，构建最短路网（一条边(u,v)在最短路网中，当且仅当 s 到 u 的最短路长度加上边(u,v)的长度等于 s 到 v 的最大路长度）。最短路网是一个有向无环图（DAG），在这个有向无环图上使用动态规划算法，即可求出字典序第 k 小的最短路，时间复杂度为 O(n+m)，n 表示图中的点数，m 表示图中的边数。

### 3.3.7 随机游走

在该过程中，每条边至多经过一次，因此最坏情况下时间复杂度为 O(m)，m 表示图中的边数。

# 4 实验与测试

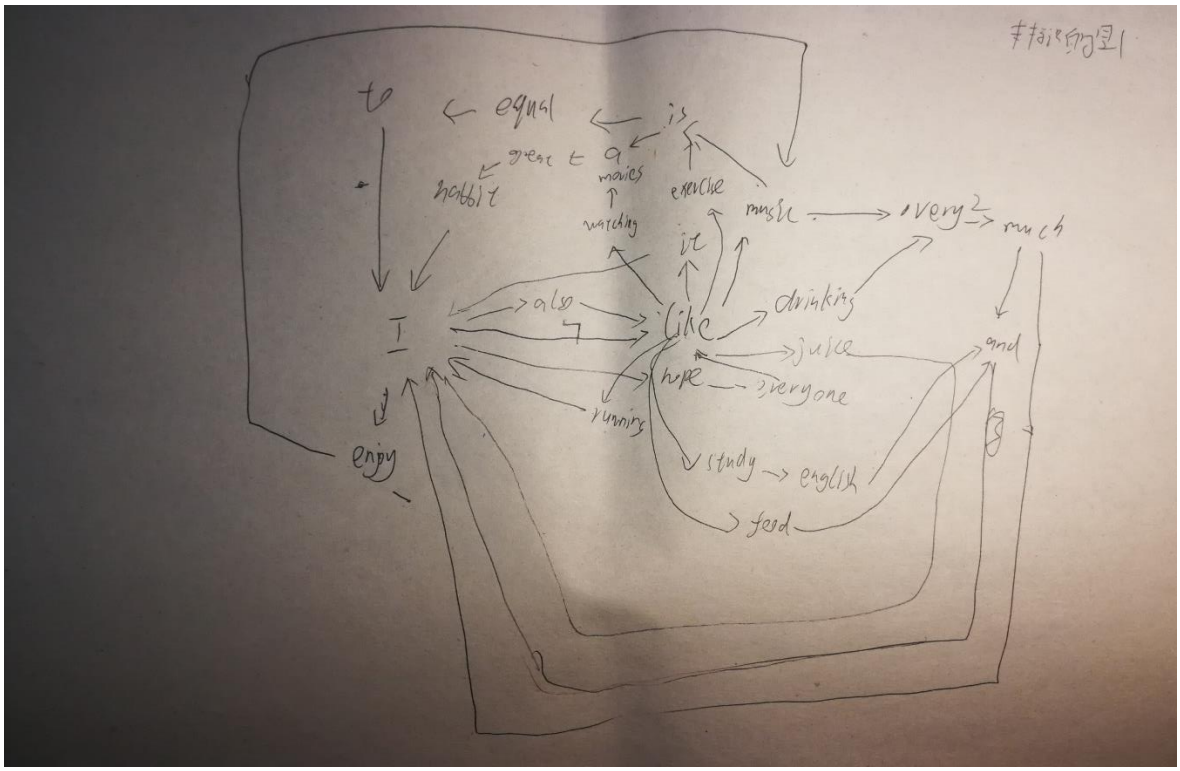
## 4.1 读取文本文件并展示有向图

### 4.1.1 文本文件内容

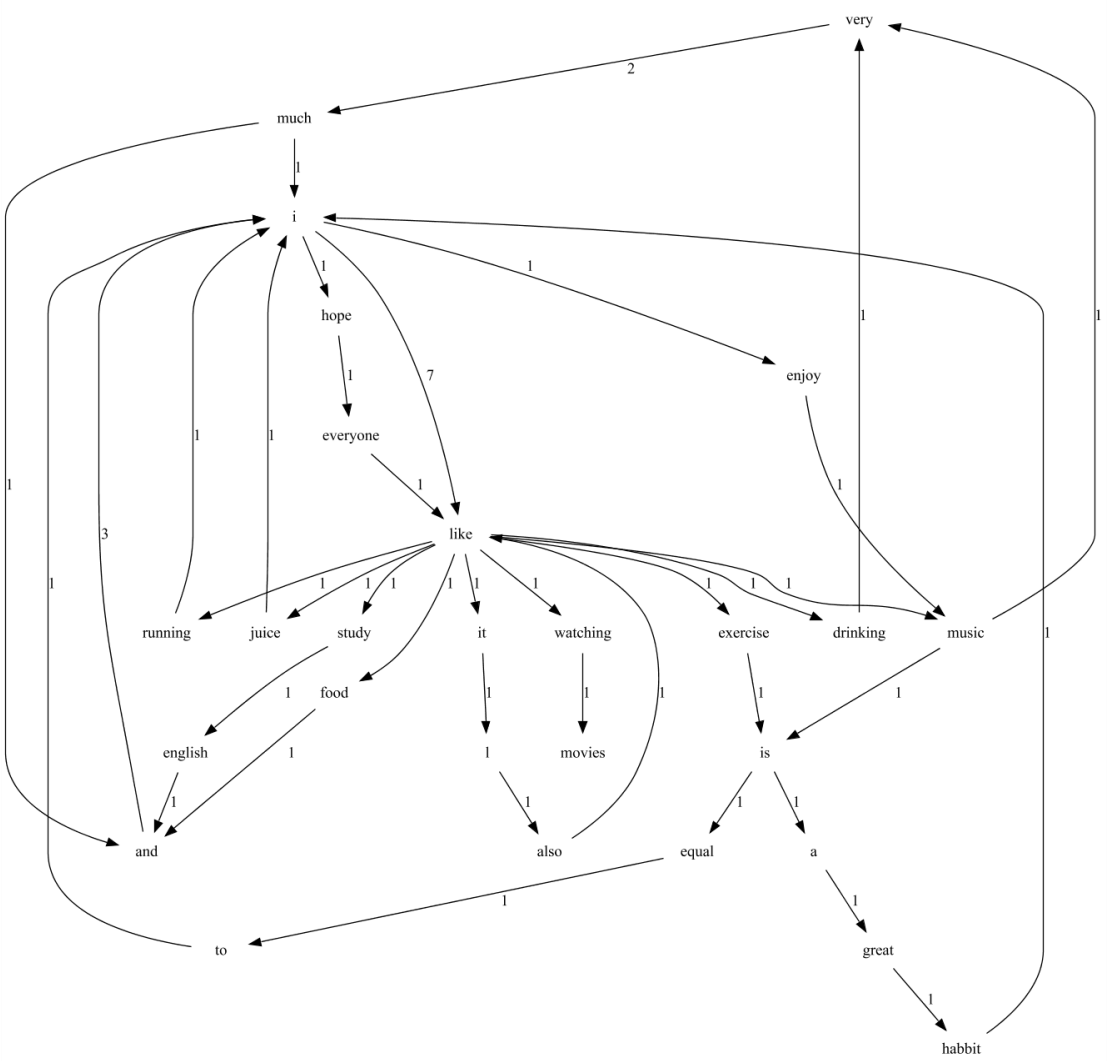
I like music is equal to i enjoy music very much.
I like drinking very much and i like juice.

I hope everyone like it l also like exercise is a great habbit i like running i like study english and i like food and i like watching movies.

## 4.1.2 期望生成的图（手工计算得到）

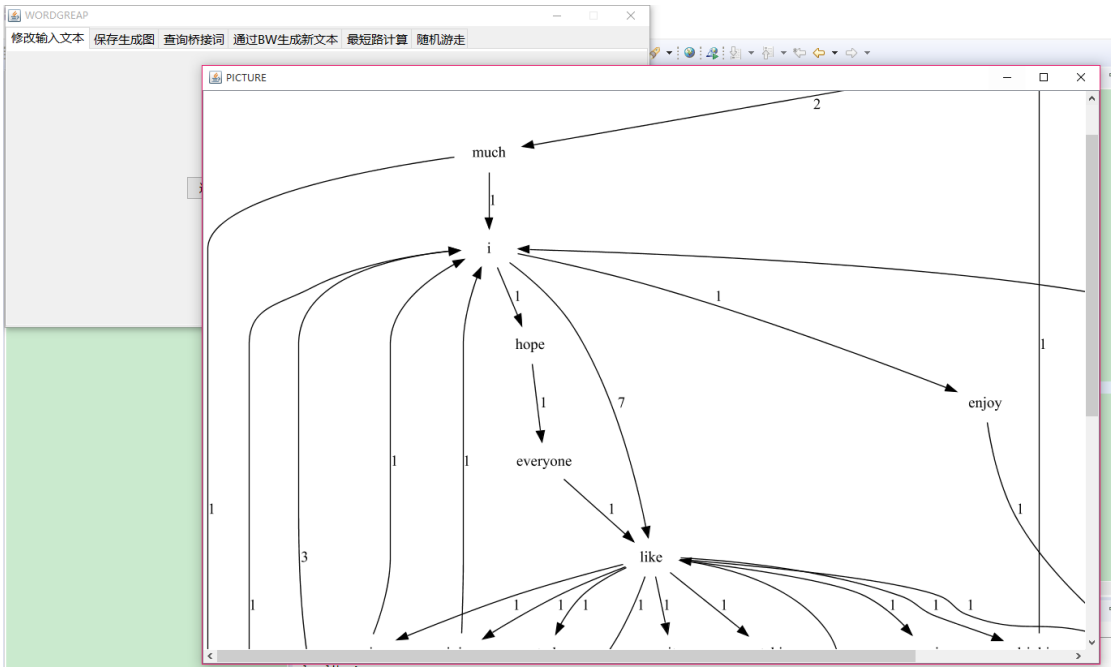


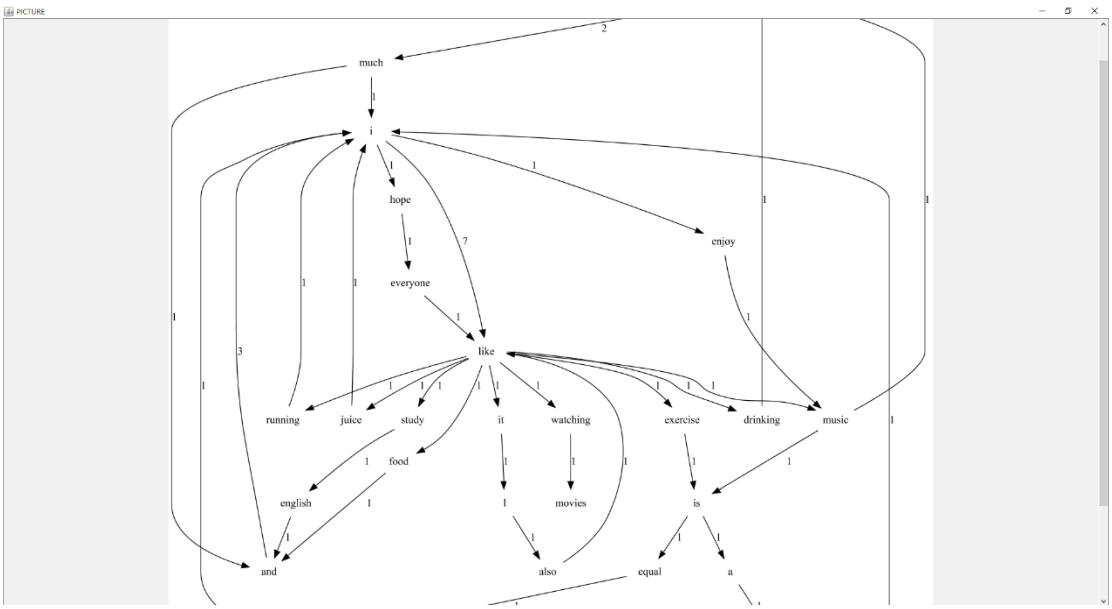

## 4.1.3 程序实际生成图


very
much
i
hope
enjoy
everyone
like
running
juice
study
it
watching
exercise
drinking
music
food
english
movies
is
and
also
equal
a
to
great
habbit

**两图一致**

### 4.1.4 实际运行界面



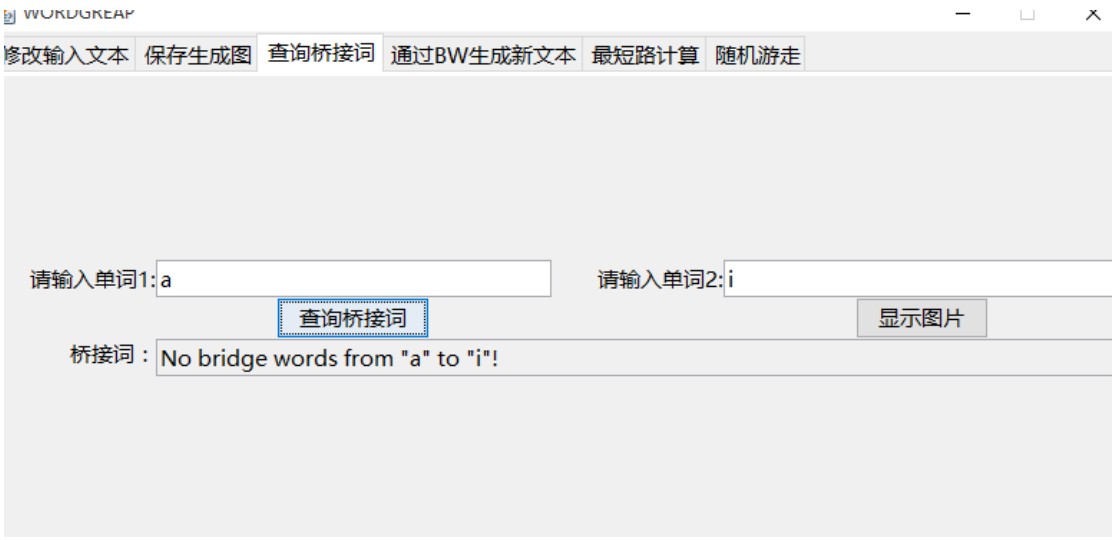



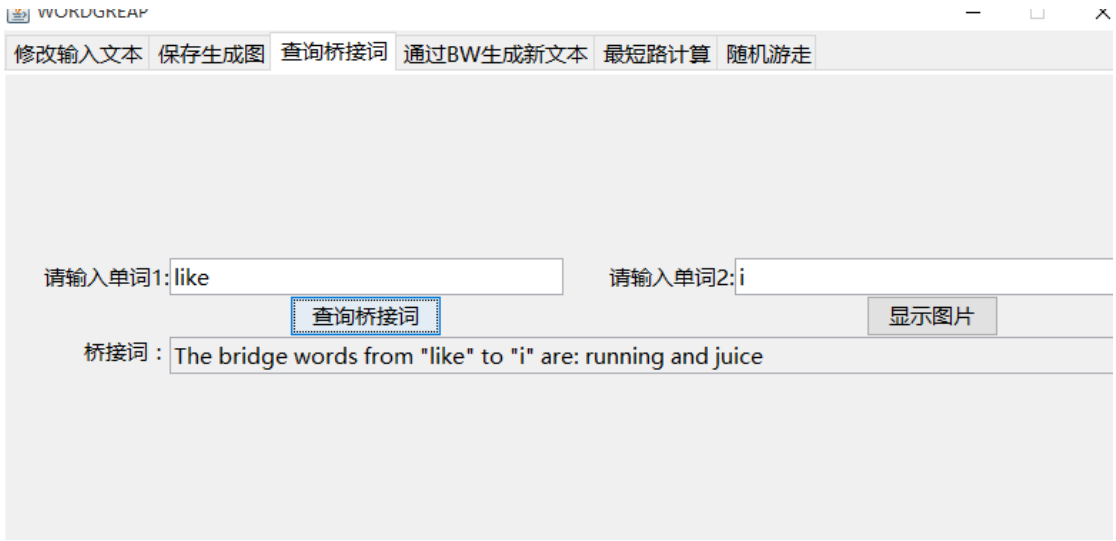

## 4.2 查询桥接词

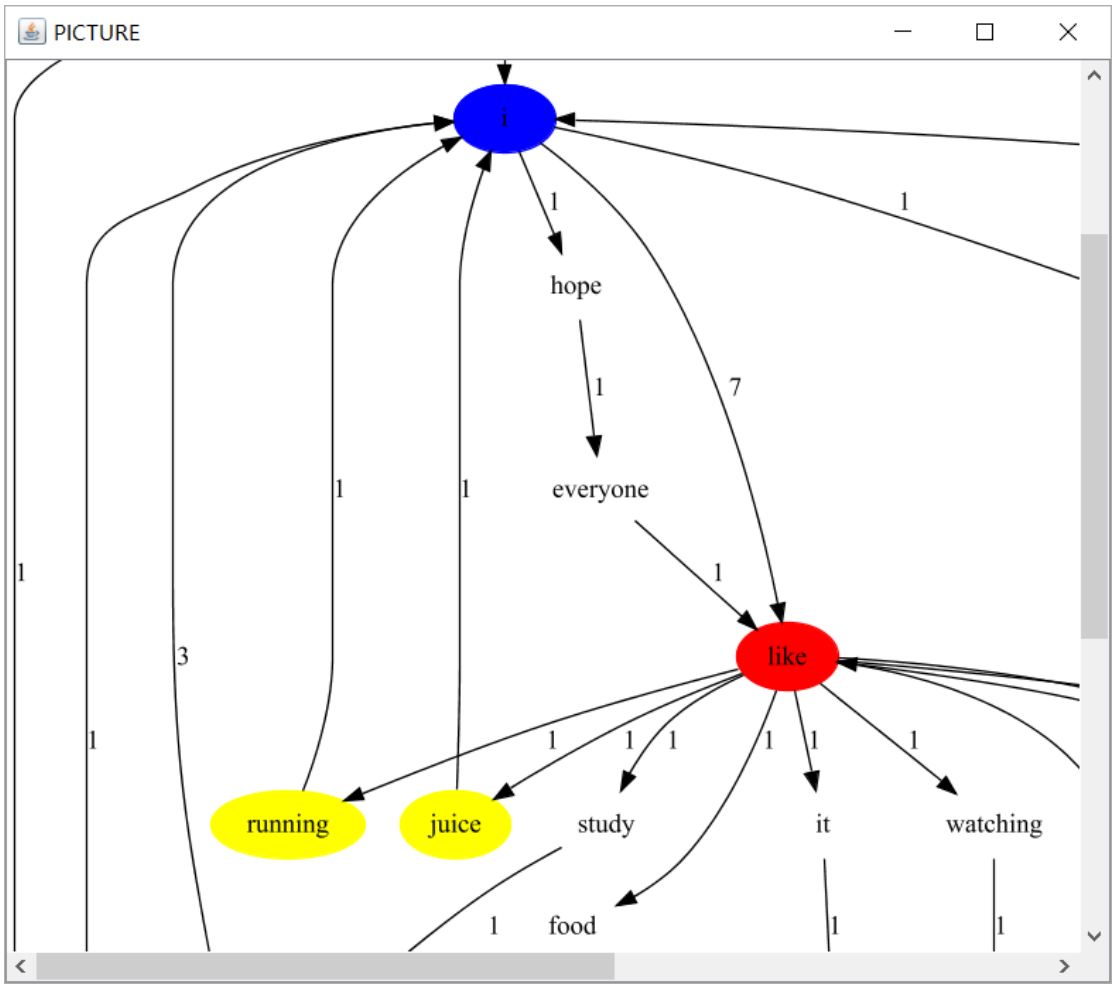
| 序号 | 输入（2 个单词） | 期望输出 | 实际输出 | 运行是否正确 |
|---|---|---|---|---|
| 1 | a i | No bridge words from "a" to "i"! | No bridge words from "a" to "i"! | 正确 |

| | | | | |
|---|---|---|---|---|
| 2 | Like i | The bridge words from "like" to "i" are: running and juice | The bridge words from "like" to "i" are: running and juice | 正确 |
| 3 | Very i | The bridge word from "very" to "i" is: much | The bridge word from "very" to "i" is: much | 正确 |
| 4 | Very ggg | No "ggg" in the graph! | No "ggg" in the graph! | 正确 |
| 5 | Verydd ggg | No "verydd" and "ggg" in the graph! | No "verydd" and "ggg" in the graph! | 正确 |

### 4.2.1 实际运行界面





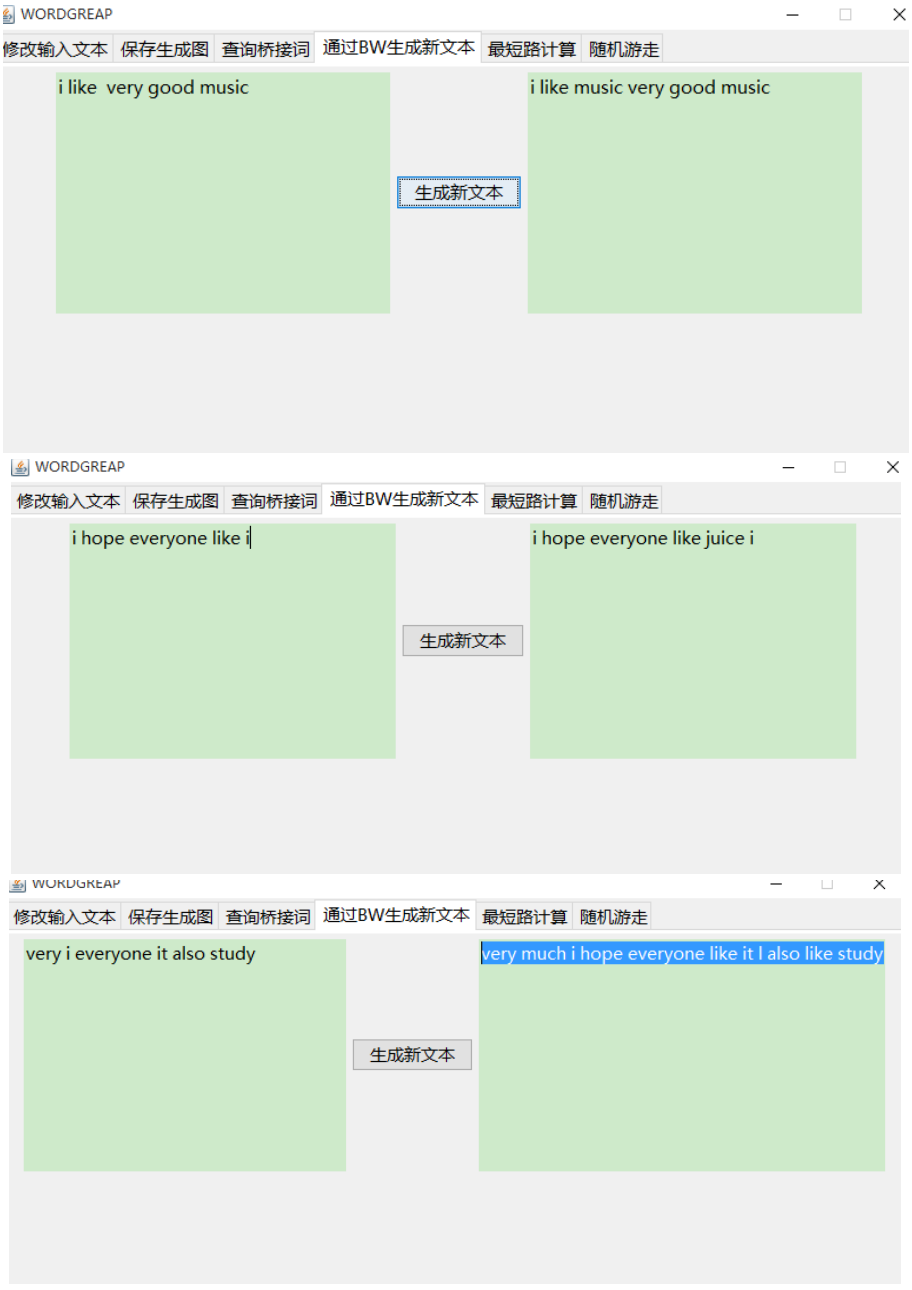


## 4.3 根据桥接词生成新文本

| 序号 | 输入（一行文本） | 期望输出 | 实际输出 | 运行是否正确 |
|---|---|---|---|---|
| 1 | i like very good | i like music very good | i like music very | 正确 |

| | | | | |
|---|---|---|---|---|
| | music | music | good music | |
| 2 | i hope everyone like i | i hope everyone like juice i | i hope everyone like juice i | 正确 |
| 3 | very i everyone it also study | very much i hope everyone like it l also like study | very much i hope everyone like it l also like study | 正确 |

### 4.3.1 实际运行截图

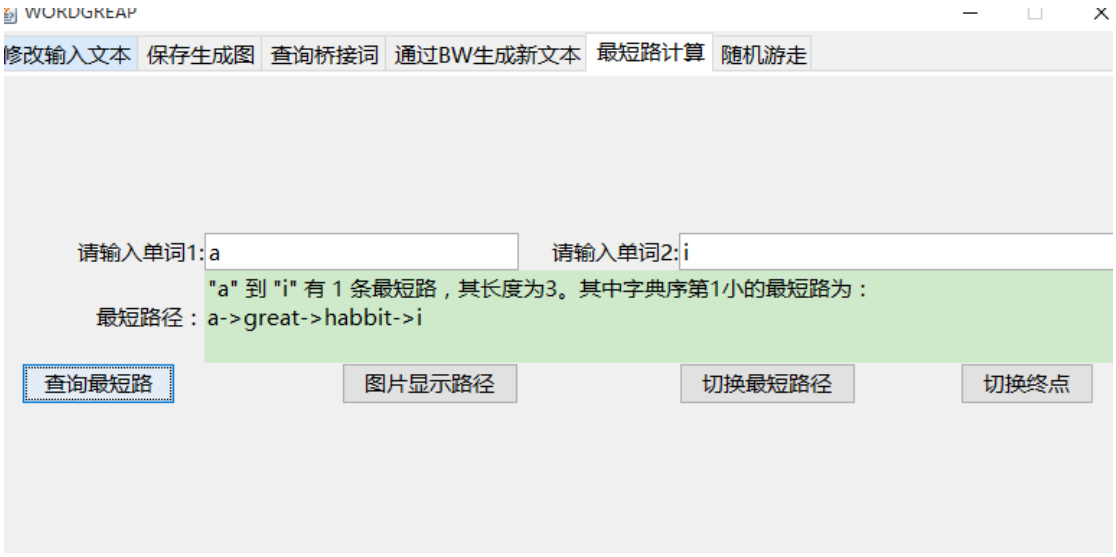


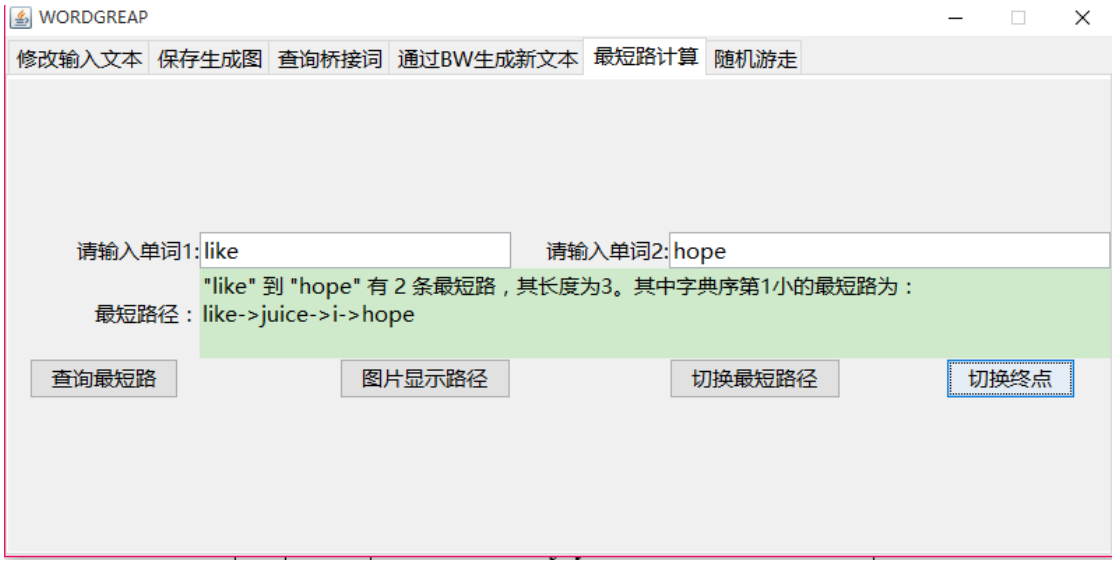
## 4.4 计算最短路径

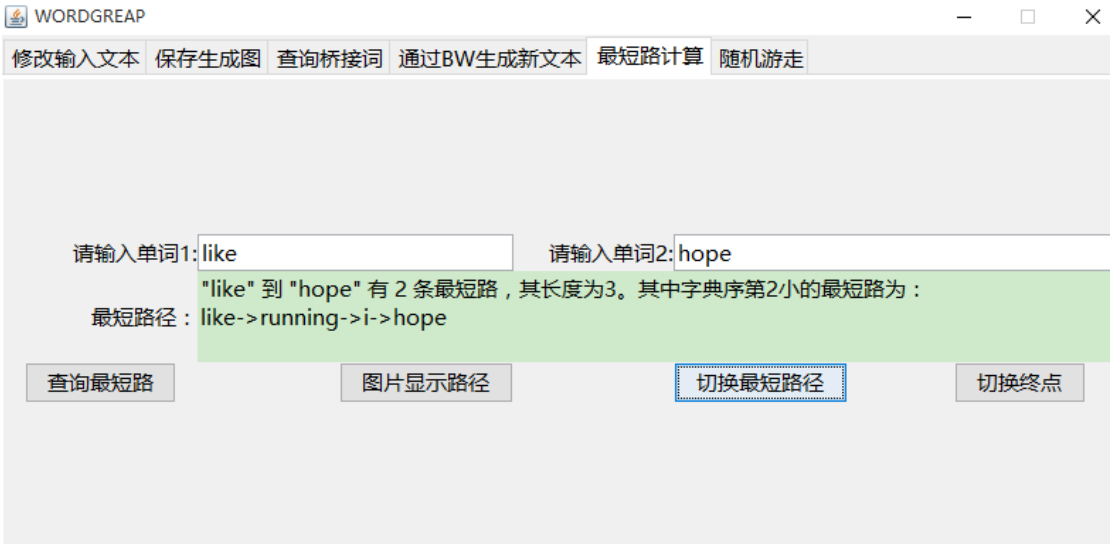
我们实现了切换不同路径的功能，和切换终点的功能。

| 序号 | 输入（两个单词、或一个单词） | 期望输出 | 实际输出 | 运行是否正确 |
|---|---|---|---|---|
| 1 | a i | "a" 到 "i" 有 1 条最短路，其长度为 3。其中字典序第 1 小的最短路为：<br>a->great->habbit->i | "a" 到 "i" 有 1 条最短路，其长度为 3。其中字典序第 1 小的最短路为：<br>a->great->habbit->i | 正确 |
| 2 | a english | "a" 到 "english" 有 1 条最短路，其长度为 8。其中字典序第 1 小的最短路为：<br>a->great->habbit->i->hope->everyone->like->study->english | "a" 到 "english" 有 1 条最短路，其长度为 8。其中字典序第 1 小的最短路为：<br>a->great->habbit->i->hope->everyone->like->study->english | 正确 |
| 3 | Like<br>Hope | "like" 到 "hope" 有 2 条最短路，其长度为 3。其中字典序第 1 小的最短路为：<br>like->juice->i->hope<br><br>"like" 到 "hope" 有 2 条最短路，其长度为 3。其中字典序第 2 小的最短路为：<br>like->running->i->hope | "like" 到 "hope" 有 2 条最短路，其长度为 3。其中字典序第 1 小的最短路为：<br>like->juice->i->hope<br><br>"like" 到 "hope" 有 2 条最短路，其长度为 3。其中字典序第 2 小的最短路为：<br>like->running->i->hope | 正确 |

### 4.4.1 实际运行截图


WORDGREAP
修改输入文本
保存生成图
查询桥接词
通过BW生成新文本
最短路计算
随机游走
请输入单词1: a
请输入单词2: i
"a" 到 "i" 有 1 条最短路，其长度为3。其中字典序第1小的最短路为：
最短路径：a->great->habbit->i
查询最短路
图片显示路径
切换最短路径
切换终点


WORDGREAP
修改输入文本
保存生成图
查询桥接词
通过BW生成新文本
最短路计算
随机游走
请输入单词1: like
请输入单词2: hope
"like" 到 "hope" 有 2 条最短路，其长度为3。其中字典序第1小的最短路为：
最短路径：like->juice->i->hope
查询最短路
图片显示路径
切换最短路径
切换终点

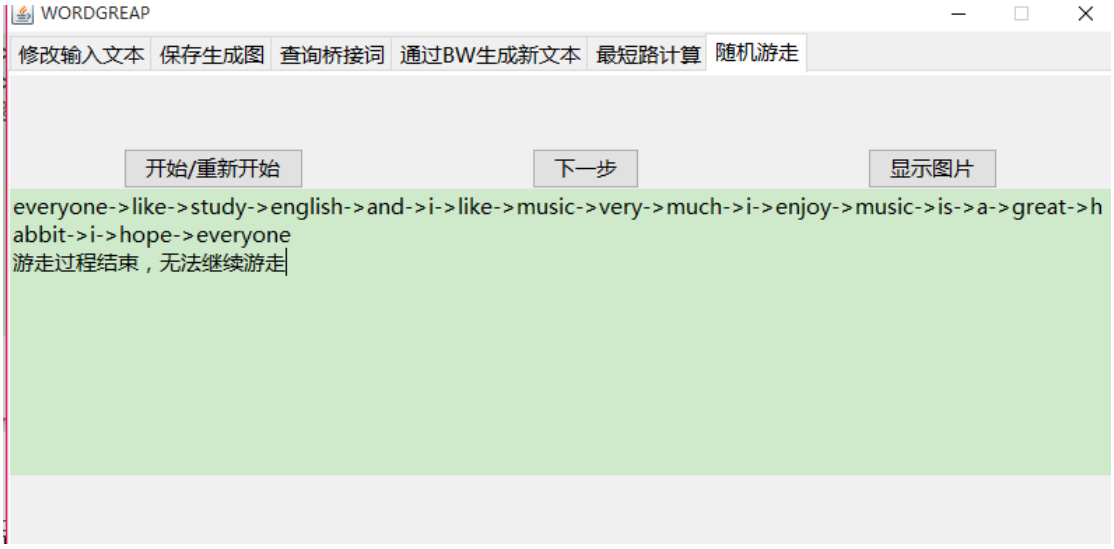


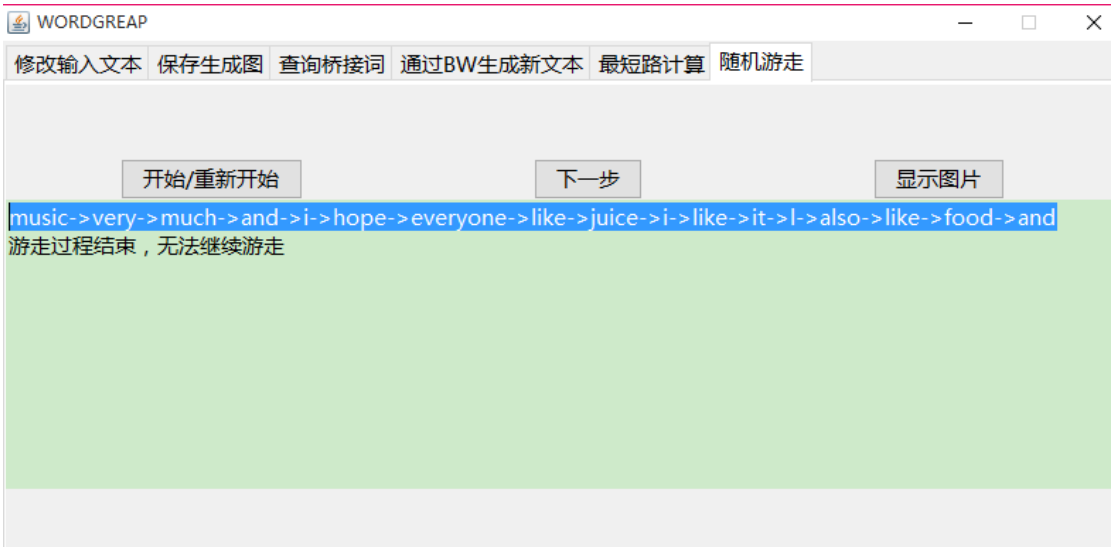

## 4.5 随机游走

注释：点击开始/重新开始选择一个节点，点击下一步选择与之相邻的下一个节点，点击显示图片将走过的路径标识在图片之中。

| 序号 | 实际输出 | 程序运行是否正确 |
|---|---|---|
| 1 | everyone->like->study->english->and->i->like->music->very->much->i->enjoy->music->is->a->great->habbit->i->hope->everyone | 正确 |
| 2 | music->very->much->and->i->hope->everyone->like->juice->i->like->it->l->also->like->food->and | 正确 |

### 4.5.1 实际运行界面




# 5 编程语言与开发环境

Java DK 版本: 1.8.0_144
Eclipse IDE 版本 ：Eclipse IDE for Java Developers Oxygen Release (4.7.0)

# 6 结对编程

## 6.1 分组依据

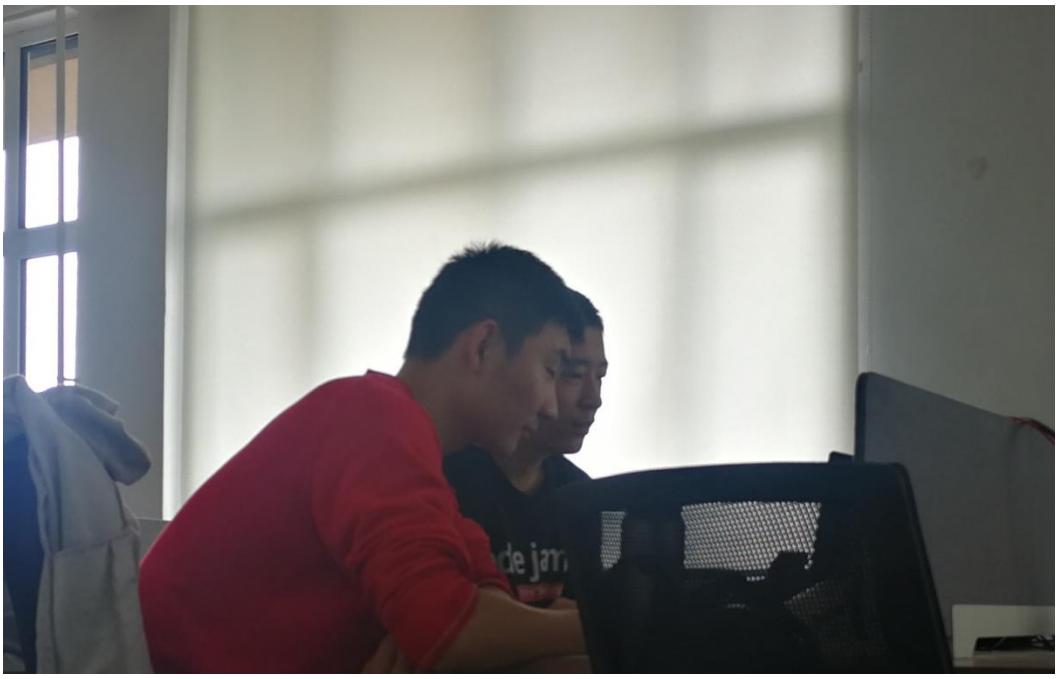
本小组中两个成员为朋友、同学和室友关系，两个人关系比较好，互相了解，能够在结对编程之中实现更好的沟通，马玉坤写代码能力，数据结构分析能力比较强，学习能力很强，有很丰富的项目经验；张宁的数学计算能力很强，并且积极承担责任，能够担任团队中的任

务。

## 6.2 角色切换与任务分工

| 日期 | 时间(HH:MM -- HH:MM) | “驾驶员” | “领航员” | 本段时间的任务 |
|---|---|---|---|---|
| 2017.9.10 | 9:00 – 12:00 | 张宁 | 马玉坤 | 编写数据读取的功能 |
| 2017.9.10 | 14:00 – 17:00 | 马玉坤 | 张宁 | 图像生成、桥接词查询和新文本生成 |
| 2017.9.12 | 15:00 – 17:00 | 马玉坤 | 张宁 | 最短路和随机游走功能 |
| 2017.9.13 | 13:45 – 15:30 | 张宁 | 马玉坤 | 编写一部分的 gui 界面对页面进行美化和布局。 |
| 2017.9.13 | 19:00 – 23:00 | 马玉坤 | 张宁 | 最后的性能优化 |

## 6.3 工作照片




code jam

## 6.4 工作日志

由领航员负责记录，记录结对编程期间的遇到的问题、两人如何通过交流合作解决每个问题的。可增加表格的行。

| 日期/时间 | 问题描述 | 最终解决方法 | 两人如何通过交流找到解决方法 |
|---|---|---|---|
| 2017.9.10 | 不知道如何可视化展示单词构成的图 | 使用了一个外部库 graphviz-java | 首先通过互联网查询 graphviz 相关信息，并查询到 |
| 2017.9.10 | 不知道如何对输入文本进行预处理 | 使用正则表达式对字符串处理 | 交流 Java 的学习心得，并一起在参考资料中查找字符串的方法 |
| 2017.9.13 | Gui 页面布局混乱 | 使用了网格袋布局来对组件进行布局 | 一起在参考资料中查找 GUI 设计的方案和布局格式，并一起画图设计讨论 |
| 2017.9.13 | Graphviz 画图太慢 | 我们默认不显示图片，除非用户点击显示图片按钮 | 一起测试每个模块的运行时间，并找到短板进行讨论解决 |
| 2017.9.10 | 如何存储由单词组成的图 | 使用 HashMap 数据结构 | 通过数据结构课的前提知识，进行存储结构的讨论 |
| 2017.9.10 | 如何将路径以不同的颜色显示在图片之中 | 我们给每一个节点和边添加一个变量来记录颜色 | 对图结构进行了讨论 |
| 2017.9.13 | Graphviz 画图出现错误 | 调整了 graphviz-java 库的调用方法 | 查阅了 graphviz 的资料，了解了 graphviz-java 库的原理和高级使用方法，从而调整了调用方法 |
| 2017.9.13 | 如何查询所有最短路中字典序第 k 小的路径 | 使用动态规划算法 | 使用算法课学习的知识，进行讨论 |

# 7 小结

对本次实验过程和结果的思考，包括但不限于对以下问题的思考：

- 与两人分别编程相比，是否体验到编程效率的提高、编程质量的提高？
  - 我们两个人通过实验一的结对编程实践，体会到了结对编程的魅力，相比于一个人编程，代码质量与编程速度有了明显的提升。

- 你认为结对编程的优势在哪里、有什么不适应之处？它的适用场合是什么？
  - **优势**：我们觉得结对编程优势很明显，可以减少很多的 bug，降低思维难度，加快开发速度。
  - **劣势**：但结对编程时，每个人对 IDE 的使用习惯不太相同，使用一台机器进行结对编程时，可能会产生不适。此外，由于小组内成员平日的学业较为繁忙，

能凑在一起进行结对编程的时间并不多，时间安排不是很灵活。

- **适应场合：**小项目，时间段，迭代周期快的项目开发适合结对编程。

- 你认为为何在敏捷开发中倡导结对编程？
  - 敏捷开发过程之中要求迭代速度尽可能快，并不重视写文档，重视代码质量，所以结对编程可以明显提升开发速度，提高代码质量，达到快速迭代的目的。

- 未来的软件开发任务中，你是否愿意使用结对编程？
  - **马玉坤：**我个人非常喜欢与人合作，在未来可能的软件开发任务中，我更倾向于使用结对编程。这可以大大减少我犯低级错误的概率，从而提升编码效率。
  - **张宁：**我愿意和合适的人一起结对编程，来达到互相监督，提高编程质量，加快开发速度的目的。