软件工程实验报告

实验环境

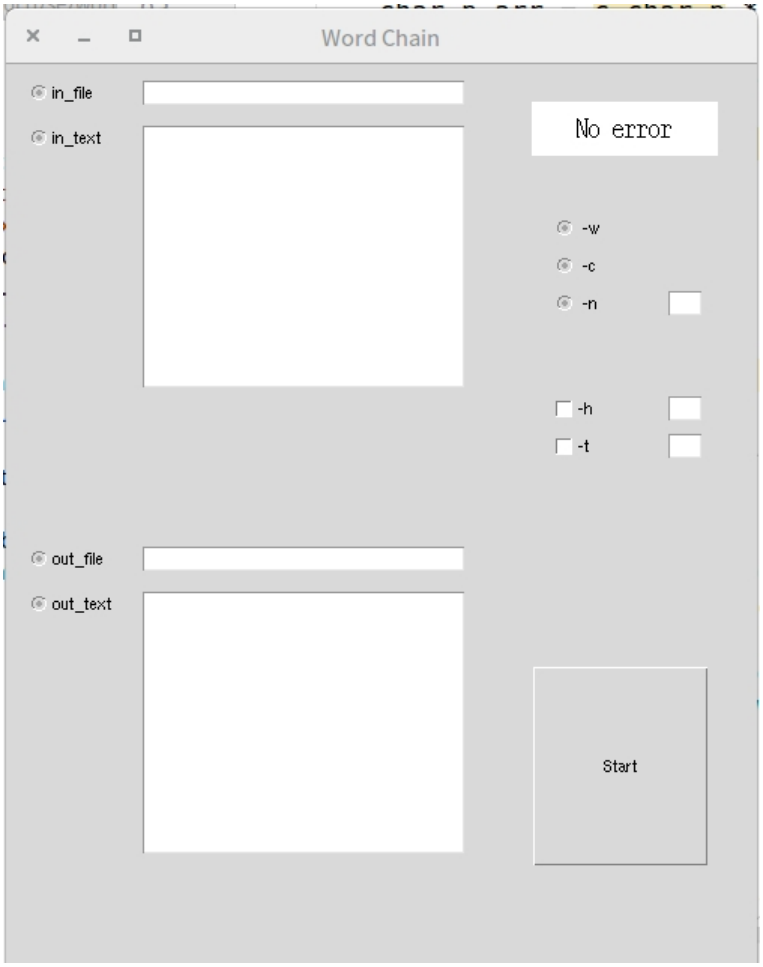
基础算法实现：

1. IDE： Visual Studio 2017. VSCode
2. 操作系统：WIndows 10. Ubuntu 16.04
3. 运行环境:Linux Ubuntu16.04

GUI 与编译

GUI：

- GUI 界面展示:



- GUI 功能:
  - 输入:可选择文件输入,直接输入两种方式.文件输入直接选择 in_file,并在 in_file 栏里输入待输入文件的绝对路径或者相对路径;直接输入则选择 in_text,并在 in_text 栏中直接输入内容,如果没有选择输入方式,则默认以 in_text 方式输入.
  - 输出:可选择文件输出,直接输出两种方式.文件输出直接选择 out_file,并在 out_file 栏里输入待输出文件的绝对路径或者相对路径;直接输出则选择 out_text,并在 out_text 栏中直接输出内容,如果没有选择输出方式,则错误栏会给予报错.
  - 参数选择:'-w','-c','-n'这三种参数互斥,只可以选择一种,若都不选,则默认'-w','-h','-t'这两种参数可以任意选择,'-n','-h','-t'这三种参数若选择,则需要在后面的栏中输入相应的信息,'-n'后输入数字,'-h','-t'后输入[a-zA-z]的字母,若输入不符合要求,则会在错误栏报错
- 错误信息
  - Lack parameter: -n, -t, -h 参数选择了但未输入相应信息

- parameter error: 参数输入错误
- in_file illegal: 输入文件不合法
- in_file not exist: 输入文件不存在
- lack input: 选择了 in_text 输入,但未输入内容
- input illegal: 输入内容不合法
- where to output: 选择了 out_file,但未输入路径

编译:

```
> cd wordChain/src
> g++ -fPIC -c WordChain.cpp -std=c++11
> g++ -shared -o libtest.so WordChain.o
> mv libtest.so ../GUI
> cd ../GUI
> pyinstaller -w -F gui.py
> mv dist/gui .
> ./gui
```

或
直接进入 wordChain/目录,然后输入

```
> shell compile.sh
```

运行 gui 需要在 GUI/目录下

若 gui 运行报错则,尝试按转 lib/目录下的库文件
若依然报错,可直接进入 GUI/目录,输入

```
> python gui.py
```

运行

基础算法实现

数据结构：

- 本次实验的数据结构采用了图的存储方式：

1. 实验中，每一个单词的主要关键信息如下：
    - 单词的首尾字母
    - 单词的长度
    - 单词的内容

    从这些信息可以看出，我们可以采用图的方式来进行存储：
    - 每一个节点表示字母，所以图的规模可以被限制。
    - 用一条有向边表示一个单词，边的相关信息为单词的长度和内容。

    进一步抽象后可以得出，相关信息可以抽象为用矩阵（二维数组）来存储：

    - alphamatrix[i][j]的值表示：以第 i 个字母开始、第 j 个字母结束的单词数量，字母的顺序按照字母表来排列。
    - 接下来的两个 vector 二维数组分别存储单词的长度和内容。

2. 数据结构和相关算法在 Prefix.cpp 中的类：Words 实现。
    - 类的基本存储单元如下：

```
class Words
{
private:
    vector <string> wordMatrix[26][26];  //用于保存字符串的向量矩阵，[i][j]中表示从i开始j结束的所有单词组成的向量
    vector <int> wordSizeMatrix[26][26]; //用于保存单词长的向量矩阵，[i][j]中表示从i开始j结束的所有单词长度的向量
    int alphaMatrix[26][26];
    int total_word_num;                  //保存总单词数
```

    - 其中 wordMatrix 和 wordSizeMatrix 中的元素按顺序一一对应，且按照从小到大的顺序排列（即最大的元素在 vector 的末尾）。

算法思想：

由于采用了图的存储方式，因此非常自然地想到了采用深度优先搜索（DFS）的方式来遍历生成的图，从而得到符合要求的单词链。

- 根据需求描述，可以将四种不同的用户需求转变为不同约束条件下的 DFS：
    - 寻找最大单词数的单词链： 可以转化为在边权为 1 的图上找最长路径的 DFS 问题，在代码中通过
    - 寻找字母数最多的单词链： 可以转化为在边权不恒为 1 的图上找最长路径的 DFS 问题。
    - 寻找指定开头字母/结尾字母的单词链： 转化为从特定行开始、以特定列结束的 DFS 问题。
    - 寻找所有指定长度的单词链：转化为深度限定的 DFS 问题。

其中第三种要求可以和其他三种要求混合使用，而在没有第三种条件的限制下，需要考虑 alphaMatrix 的每一行，因为单词链可能以任何一个字母开始，所以建立一个循环来查找所有符合条件的单词链。

算法部分的主要实现在 WordChain.cpp 中

- 在实际情况中，由于文件中单词数可能会很大，因此在实际搜索中，在单词数很大时，我们可以限定搜索时间或搜索深度，从而降低时间开销，满足用户的基本需求。
- 其他的关键点：
  - 重复单词的检测：

    由于读取的文件中可能有重复的单词，因此需要在插入单词时检查是否有重复的单词，但是如果和所有已经读取的单词均进行一次比较，则时间复杂度会提升，因此这一部分需要进行优化。

    考虑到本实验中采用的数据结构，因此可以将比较的范围缩小为首尾字母相同、长度相同的 string 类变量，从而缩小了比较范围，提高了函数效率。

    以下代码段实现了按单词长度升序插入 vector 并进行重复单词的检测：

  - 命令行的处理：

    不同的用户需求体现在不同的输入参数中，但是如果输入正确（错误处理在下一部分有所提及），那么命令行的格式相对规整：

    命令行读入的 argv[]中，最后一个元素一定是文件路径，如果当前读入的命令行参数-h, -t,则接下来的参数一定是单个字母，如果输入-n，则下一个参数一定是一个数字，-w，-c 可以在任意位置单独出现。

    所以从参数的第 1~argc-2 个元素，进行上述检查：

错误处理：

本次实验中，需要对程序的健壮性进行考量，所以对于一些错误输入的情况，需要进行特殊处理：

- 文件不存在或者文件为空：

  这两种情况均只用对文件进行简单的判断即可实现。
- 命令行格式出错或者存在非法组合：

由于本工程的用户需求主要体现在命令行参数上，因此需要考虑命令行输入错误的情况：

1. 命令行参数组合错误：

- 此处由 gui 操作限制, '-w', '-c', '-n'参数只可以选择一个,否则会相互矛盾
- 若'-n', '-h', '-t'参数后没有将相应的补充信息,则 gui 会报错,并给予错误提示
- '-w', '-c', '-n'这三个参数必须选择一个,若都没有则默认'-w'

2. 命令行参数内容错误：

- 对'-c', '-h'和'-n'后加的参数给予限定,若不符合要求,则报错

- 单词数为零：

这一部分通过 Words 类中新增加的变量 total_word_num 实现，在每一次向 vector 矩阵中插入单词时计数。在文件读取完成之后检查 total_word_num，如果最终结果为 0，则直接报错：此时文件中没有合法单词。