**北大附中2019学段汇报**

**猴子打字项目**

作者：李天桐 谢梓涵 周翟恩和

（以上姓名不分先后）

日期：2019年6月26日

开源项目地址：[**https://github.com/bdfzoier/Monkey\_Type**](https://github.com/bdfzoier/Monkey_Type)

目录

[1 引言 2](#_Toc12452708)

[2 基本原理 3](#_Toc12452709)

[3 实现计划 5](#_Toc12452710)

[4 小组分工 6](#_Toc12452711)

[5 实现过程 6](#_Toc12452712)

[5.1 单词库 6](#_Toc12452713)

[5.2 分开单词 6](#_Toc12452714)

[5.3 存储数据 7](#_Toc12452715)

[5.4 随机一个组合 7](#_Toc12452716)

[5.5 DFS生成伪单词 8](#_Toc12452717)

[5.6 量化分析 9](#_Toc12452718)

[6 与马尔可夫链的关系？ 10](#_Toc12452719)

[6.1 什么是马尔可夫链？ 10](#_Toc12452720)

[6.2 我们对这个问题的建模与马尔可夫链是否有关系？ 10](#_Toc12452721)

[7 项目意义与展望 10](#_Toc12452722)

[8 参考文献/数据 11](#_Toc12452723)

## 1 引言

猴子打字项目是一个能自动分析学习字典，利用一阶马尔可夫链生成类似英语单词，满足英语单词基本发音和规则的字符串的程序。

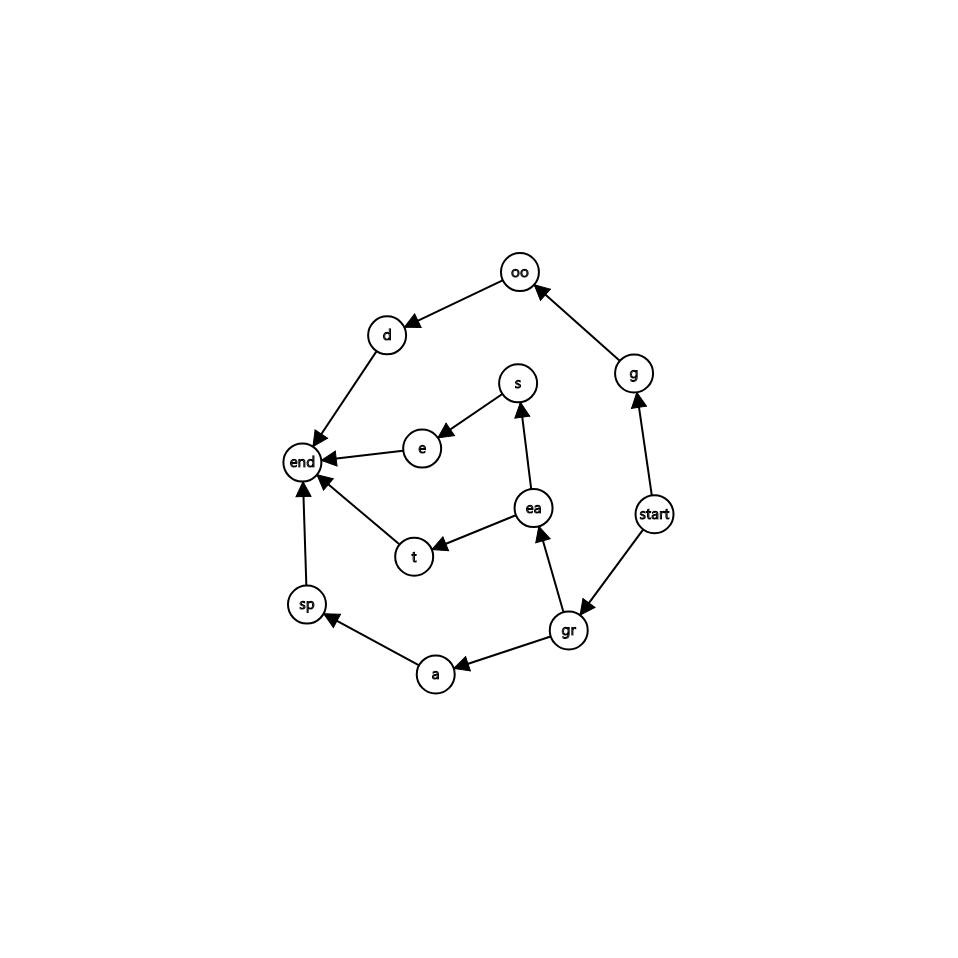
相信大家都听说过”脸滚键盘“，虽然这也是一个生成单词的方法，但是这种单词通常都不会满足发音规则等任何规则，就更不用提能不能生成真正的单词了。它的概率小于1/1000000000。

但是，利用机器学习，可以让几乎所有生成的单词都可以发音，同时可以把生成真单词的概率提升到20%，这就略有些困难，本论文就提供了一个可共借鉴的生成思路。

关键词：机器学习、马尔可夫链、后缀数组（SA）、生成单词

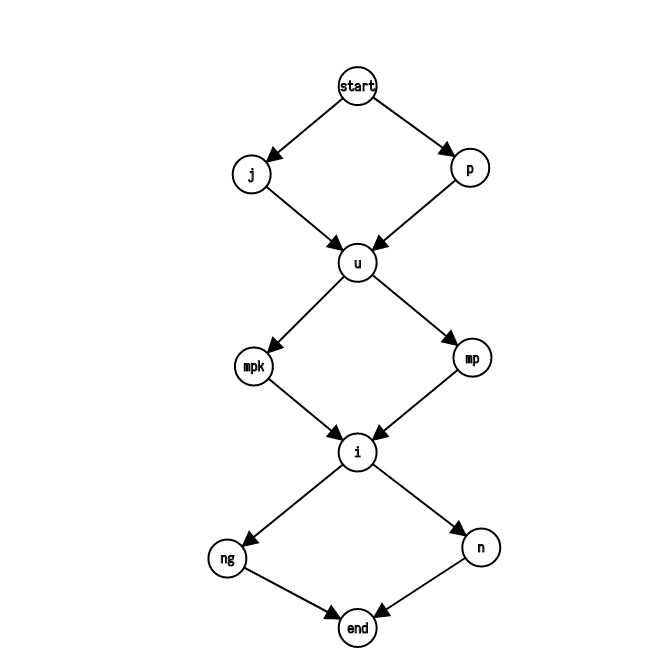
## 2 基本原理

采用一个词库，把所有单词拆成连续元音+连续辅音+连续元音+连续辅音……的组合，比如traditional=tr+a+d+i+t+io+n+a+l。然后建有向无权图，将第i个组合向第i+1个组合连上一条边，从start节点向第1个组合连一条边，从最后一个组合向end节点连1条边。比如当我们采用了great,grasp,grease,good这4个单词的词库，那么我们建成的图将会是如下形式：



采用类似网络流建模的方式，我们列出Start及End节点，造点连边后根据概率和随机生成的长度生成单词。把路径上经过的所有节点连起来，就是一个新单词了。显然，词的数量越多，可能产生的新单词的数量也就越多。~~貌似这4个单词还没法生成任何新单词~~

再举一个例子，pumpkin和jumping生成的图是这样的



不难从图中看到，一共有8种方式从起点出发并到达重点，对应生成的单词分别为：

jumping  
jumpkin  
jumpking  
jumpin  
pumping  
pumpin  
pumpkin  
pumpking

## 3 实现计划

实现时，我们分为4步实现这个项目：

第1步，寻找一个合适的单词库。

第2步，编写一个专门用来分开所有单词的程序，这个程序将会把所有的单词拆分成续元音+连续辅音+连续元音+连续辅音……的组合。

第3步，编写用于存储数据的结构体，后面所有函数封装到这个结构体里面。

第4步，编写一个用来随机寻找一个组合的程序，我们在这里做到了按概率比例来随机生成：例如有5个单词有sp+oi的组合，1个单词有sp+e的组合，那么我们编写的程序随机生成60000个组合，将会有大约50000个sp+oi和10000个sp+e。

第5步，也就是展示用程序，将会从start节点随机一个路径到end节点：每一次采用随机一个组合的方法找到下一步，然后用递归的方法找到路径，并输出。

第6步，进行量化分析，也就是类似于测算生成出的10000个单词中，有多少个是真正的英文单词。这样，就可以得出大致的相似度以及这个程序到底有多“聪明”。

## 4 小组分工

周翟恩和负责完成第2、3、4、5步，撰写论文；

谢梓涵负责完成API，对代码进行常数优化，撰写论文；

李天桐负责完成第1、6步，撰写论文。

## 5 实现过程

### 5.1 单词库

我们采用了英语四级词汇，大约有4300个。事实上，我们之前采用过10000个单词甚至更大的一个7000000个单词的大型词库，但经过人工查询后发现很多单词都不是正常的英文单词（如地名，带“-”的单词，或者没有元音的单词），这样生成出来的“单词”反而更不像正常的英语单词。因此，我们选择了英语四级词汇作为我们机器学习的用具。另外，单词量过大也不容易快速求解~~以及保护电脑健康。~~

### 5.2 分开单词

我在这里采用了暴力拆解的方法，直接把所有的字母组合拆出来当作图的节点，并记录下一个节点。我们把这一部分以及3、4部分写成了三个头文件。

这里的时间复杂度为，其中代表单词的长度，平均长度大约在10~15左右。

### 5.3 存储数据

为了存储和处理方便，以及节省空间，我们将每个元/辅音组利用谢梓涵通过Splay全都映射为一个整数。

我们存储数据采用的是邻接表，即将每个节点的所有子节点全部顺序存下在一个数组里。

### 5.4 随机一个组合

在生成单词中遇到一个问题便是由于最终可以生成的单词过多，甚至由于图中的自环（o后面可以接n，n后面可以接o），会导致我们在生成的过程中出现死循环，所以我们需要一个强有力的方法来在搜索过程中进行筛选，放弃某些情况，避免我们程序发生死循环。

由于可以生成的单词数量随着深度的增加，呈指数级增长，所以最有效的一个筛选方法就是控制搜索的深度，即单词由几个声/韵母组构成。 另一个方法即是只随机生成部分的单词，而这里遇到的问题就是，因为空间问题，我们不得不把所有相同的节点压缩到1个节点，但是我们在节点对象中封装了一个prop变量，用于存储压缩前该组合在其父节点的所有出现的次数。而它的生成概率将与prop成正比。具体来说，第个元素的生成概率即为。但是c++中关于随机数的函数只有一个，其功能也十分简陋，即简单的生成一个随机的整数，那么我们如何实现上述的操作呢？

对于一个数组，如下

|  |  |  |  |
| --- | --- | --- | --- |
| index→ | 1 | 2 | 3 |
| value | 10 | 31 | 12 |
| prop | 5 | 1 | 4 |

我们的解决办法是对其prop属性求前缀和，如下

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| index→ | 0 | 1 | 2 | 3 |
| sum | 0 | 5 | 6 | 10 |

然后先生成一个随机数，这里假设是157，将其对sum[3]，即所有项的和取余，得到7。

发现sum[3]>7>=sum[2]，所以这次随机得到的结果便是下标为3，值为value[3]，即12这个数。

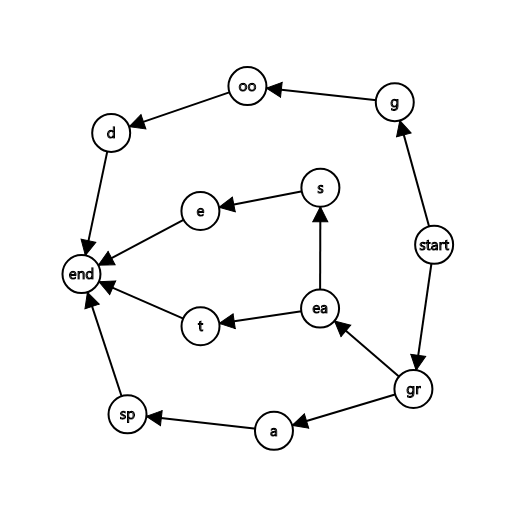
当这个数字对取余时，其可以为中的任意一个数，每个数概率相等，而如果该数在[sum[i-1],sum[i])之间，共有prop[i]个数，所以其概率恰好为。

### 5.5 DFS生成伪单词

DFS就是正常的深度优先搜索。先通过随机生成的组合来建成一个有向图，拥有**start**节点及**end**节点。深度优先搜索即从**start**节点开始搜索。由于图可能存在环状结构，因此可以通过给定的深度（单词部分的长度）来判断单词是否结束。通过此种方法，可以按照某种顺序生成出伪单词。

这一步的原始数据完全依赖于上一步的随机单词。如果随机选出的单词出现了o接n，n接o的情况，生成出的单词将会出现像nonononononono一样的单词，所以对于dfs，我们同样需要对深度进行限制。

如下图情况：



这样，如果深度为4，从start节点开始进行DFS搜索的序列为：start → g → oo → d → end ← d ← oo ← g ← start → gr → ea → s → e ← s ← ea → t → end ← t ← ea ← gr → a → sp → end ← sp ← a ← gr ← start （注：→表示此深度优先搜索（DFS）算法的搜索顺序，而←表示回溯过程）。

最后在结尾时，必须进行判断是否到达end节点来决定是否输出DFS生成的伪单词，如果是因为深度限制导致的搜索停止则不能输出。（不然测试时发现测试时发现会生成一大堆相同的单词，或者不完整（刚生成了一半）的单词就被输出了）

### 5.6 量化分析

量化分析部分，我们还需要两个部分： 部分1，分别判断单词是否存在我们使用的小词库，以及网上找到的可看为包含所有单词（479000个单词）的大词库。 部分2，计数。

对于部分1，我们首先设计了一个的暴力算法，在谢梓涵经过字典树优化之后直接将复杂度降到了。~~不论是保护电脑健康能力还是~~时间和空间占用量都有所减少。

对于部分2，我们在测试中采用了深度为4~10的随机生成函数。

经过计数，我们发现在生成的一千个单词中，有214个是英语单词（存在大词库中），有47个是4300个单词的小词库中有的单词，而剩下的单词大多都与真正的英语单词十分相像。（所有生成的单词见[这里](https://raw.githubusercontent.com/bdfzoier/Monkey_Type/master/result/test2-mn4mx10-1000words-loop100/gen.txt)）

## 6 与马尔可夫链的关系？

### 6.1 什么是马尔可夫链？

马尔可夫链指的是一条链，当在这条链上进行随机移动时，第i次移动不受到任何历史移动的影响。对于我们进行单词生成，或者生成有意义的句子这类算法时，而下一个移动到达的的地方是在该节点的所有子节点中随机选择的，每一个子节点被选中的概率与其出现的次数成正比。

### 6.2 我们对这个问题的建模与马尔可夫链是否有关系？

在这里，我们可以发现，其实我们建出的图并不是传说中的“无权图”，假设一条边从u连向v，那么它的权值实际上是，其中为所有的后继节点。同时，该图满足，其中表示边的权值。

可见，我们建的模型其实就是一个马尔可夫链，因为我们这个图是静态的，所以每次移动不会受到任何历史移动的影响。

## 7 项目意义与展望

这个项目可能在拼写检查中有所意义，因为生成单词的逆操作其实就是检查某个单词是否合法，如果我们的程序能获得足够的单词数据，就能判断某个单词是否在我们生成的图中存在。

如果利用这种算法改进拼写检查的算法，便不需要将所有的单词都存储下来（因为我们的算法在存储过程中仅仅会保留单词之间有差异的部分），能有效的节省存储空间。

不过如果需要实现到这一步，我们需要大幅提高我们生成单词过程的精确度，才能提供足够的准确拼写检查。

## 8 参考文献/数据

1. <https://github.com/dwyl/english-words> 一个较大的英语单词库
2. <https://blog.csdn.net/songzitea/article/details/8864122> 利用马尔可夫链生成有意义的句子。