1. 背景知识

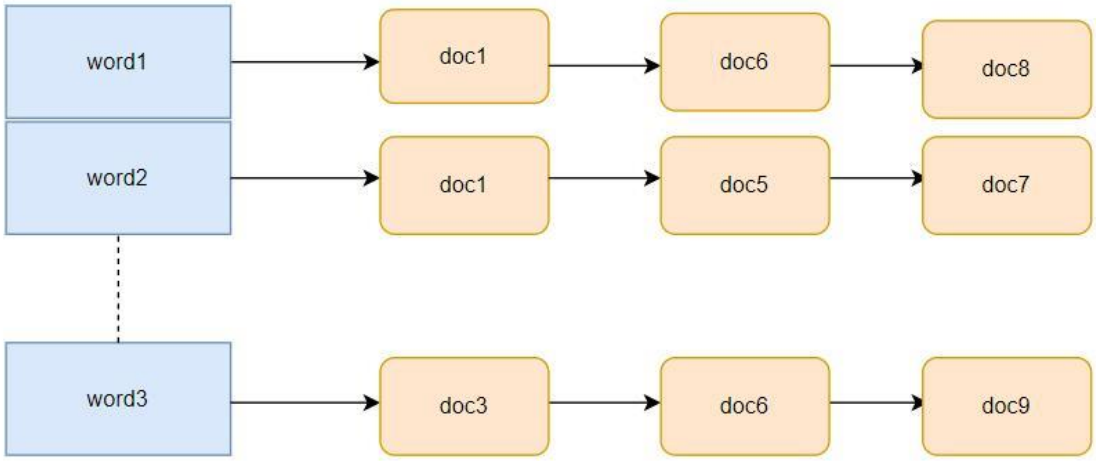
在互联网时代，Google，百度等搜索引擎（Search Engine）在我们日常的信息需求（Information need）中发挥着不可或缺的作用。搜索引擎的实现一般可分为以下 4 个模块：

（1）网络爬虫（Web Spider）：主要是从互联网上采集海量网页数据，保存在数据库中，为构建索引从而为搜索引擎提供支持。

（2）索引构建：主要是对网页数据进行清洗、分词、建立倒排索引（Inverted Index）。我们一般把句子分词后单词称为词条。比如“深圳技术大学在深圳坪山”可以分词为“深圳技术大学/在/深圳/坪山“。其中分隔符中每个单词或词语可以统称为词条。

遍历整个文档集合，得到每个词条和出现词条的文档列表，就构成了倒排索引的数据结构（类似树的孩子链表表示法，如下图所示）。

单项索引数据为：词条(word)：出现词条的文档 ID1，出现词条的文档 ID2,…,出现词条的文档 ID3。



（3）检索模块：主要根据查询表达式E，检索出候选文档集合C。常用的检索技术有布尔检索，比如

以上图为例：

检索 word1，则应该返回 doc1,doc6,doc8

检索 word1 AND word2 则应该返回 doc1

检索 word3 OR (word1 AND word2) 则应该返回 doc1，doc3,doc6,doc9,。

检索 word3 OR word AND word2 则应该返回 doc1,doc6。

（4）排序模块：使用人工规则或者排序学习（Learning to Rank）技术，综合多种因素（查询文档匹配得分、网页 Pagerank 值，标题匹配得分、时间等）对候选文档集合C进行排序，最后返回给用户。

2.题目要求

综合以上过程，搜索引擎使用了大量数据结构和算法，请你结合以上背景，独立完成以下练习。

要求：给定分词后的 10 篇维基百科文章（10 种水果的介绍），每篇文章为一个独立的文本文件，文件名为标题。

（1）扫描文件，建立倒排索引（Inverted Index），其结构如上图所示。

提示：索引使用链表结构。其中可以对文档名进行排序，并设置文档 ID。

为了加快找到词条，可采用在搜索中讲到各类算法和数据结构，比如

二分查找、Hash 查找或者 B 树等查找算法。（附加）

倒排索引的定义：也常被称为反向索引、置入档案或反向档案，是一种索引方法，被用来存储在全文搜索下某个单词在一个文档或者一组文档中的存储位置的映射。它是文档检索系统中最常用的数据结构

建立倒排索引的步骤：

步骤 1:读取一整条句子到变量 str 中;

步骤 2:从句子的尾端读取一个字到变量 word（单词）中;

步骤 3:在字典查找 word 中保存的单词。如果不存在则保存 word 并转到步骤 4，否则转到步骤 5;

步骤 4:如果是字典中最大单词或者超过最大单词数（认定为新词），从句尾去掉该单词，返回步骤 2;

步骤 5:继续读取前一个字到 word 中，转到步骤 3。

（2）实现**布尔检索模型**，简单起见：操作符使用&、| 和() 分别表示 AND，OR 和()。要求输入检索表达式，检索返回的按照文档 ID 大小进行排序，以及原文文档中第一个查询关键词在原文第一次匹配的上下文（前后 4 或 8 个字）。比如，查询表达式为：苹果，则返回结果可以是：

1(文档编号)：…每天一个苹果，补充足…

3(文档编号):….人们都说苹果是一种营….

布尔模型：

文档(用 D 表示)—— 一个文档被表示为关键字的集合;

查询式（用 Q 表示）—— 用于表示用户查询的关键词的布尔组合，并用“与(And)、或(or)、非(Not)”链接起来，且用括号知名优

先次序；

以下有两个文档：

文档 1：a b c f g h；

文档 2：a f b x y z；

要找出出现 a 或者 b 但一定要出现 z 的文档，可以将查询表示为布尔表达式 Q=(a∨b) ∧z,并转换成析取范式 qDNF=(1, 0,1) ∨(0,1,1) ∨(1,1,1)；

文档 1 和文档 2 的三元组对应值分别为(1,1,0)和(1,1,1)；经过匹配,将文档 2 返回；

方法步骤：

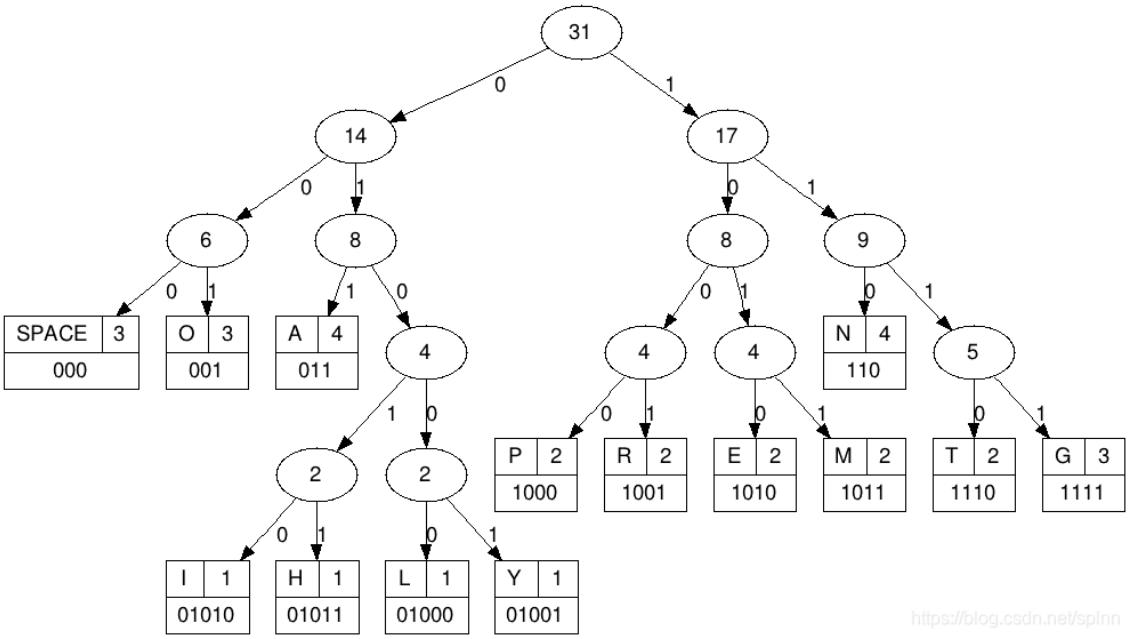
步骤一：读取表达式创建链表；

步骤二：将链表里的数与检索的字符串对应索引链表遍历，如果链表里的数出现,则打印输出这个字符串出现的 txt 文件下对应行的句子。

（3）在 10 个文件中，字母出现的频率是不相同的，为了使经常出现的词条具有较短编码、较少出现的词条具有较长编码。请你根据本学期数据结构内容实现上述文档的压缩和解压缩算法。

原理：

在计算机数据处理中，霍夫曼编码使用变长编码表对源符号（如文件中的一个字母）进行编码，其中变长编码表是通过一种**评估来源符号出现机率**的方法得到的，出现机率高的字母使用较短的编码，反之出现机率低的则使用较长的编码，这便使编码之后的字符串的平均长度、期望值降低，从而达到无损压缩数据的目的。



方法：对文件内出现的字符进行统计，以出现的次数为关键字，建立哈夫曼树，实现对字符的编码，将字符转换成一个 01 序列，因为一个字符占用一个字节，8 个 bit，而一个字节可以存八位的 01 编码，所以用位编码替换文件内的字符，既完成了文件的压缩。为了能够解码，压缩时把哈夫曼树和 01 序列一起存进文件中。解码时，将存进压缩文件里的哈夫曼树还原，通过文件里的 01 序列对哈夫曼树进行查找，即可还原出原文件。

（4）在互联网上，用户典型的浏览行为是在网页 1 上，看到了感兴趣的内容，然后点击跳转到网页 2 或者网页 3 继续浏览。跳转到不同网页的概率是不相同的，跳转概率越大，则认为两个网页跳转距离越近。那么可以定义两个网页之间的距离为：

$$dis(i,j) = \frac{1}{p(i|j)}$$

假如已经知道上述 10 个网页的不同跳转概率，见以下跳转概率矩阵 M，假设用户需要知道所有从某个网页到另外一个网页最短跳转距离，以下是浮

点数代表纵轴 1-10 号文档到横轴 1-10 文档的跳转概率，请你设计算法求解所有网页之间最短跳转距离。

```
      1，2，      3，      4，      5，      6，      7，      8，      9，      10
1   ([1.0000, 0.8530, 0.8636, 0.1771, 0.2546, 0.1262, 0.2162, 0.0488, 0.2061,0.1165],
2   [0.8530, 1.0000, 0.2819, 0.4287, 0.3330, 0.3598, 0.8237, 0.7063, 0.9565,0.4772],
3   [0.8636, 0.2819, 1.0000, 0.0429, 0.8887, 0.0297, 0.5160, 0.7141, 0.7820,0.9492],
4   [0.1771, 0.4287, 0.0429, 1.0000, 0.4003, 0.0611, 0.7951, 0.2877, 0.2106,0.5884],
5   [0.2546, 0.3330, 0.8887, 0.4003, 1.0000, 0.9987, 0.0640, 0.5897, 0.8763,0.6273],
6   [0.1262, 0.3598, 0.0297, 0.0611, 0.9987, 1.0000, 0.9034, 0.9996, 0.2579,0.2076],
7   [0.2162, 0.8237, 0.5160, 0.7951, 0.0640, 0.9034, 1.0000, 0.3385, 0.2499,0.6091],
8   [0.0488, 0.7063, 0.7141, 0.2877, 0.5897, 0.9996, 0.3385, 1.0000, 0.4253,0.8842],
9   [0.2061, 0.9565, 0.7820, 0.2106, 0.8763, 0.2579, 0.2499, 0.4253, 1.0000,0.0953],
10  [0.1165, 0.4772, 0.9492, 0.5884, 0.6273, 0.2076, 0.6091, 0.8842, 0.0953,1.0000]])
```

思路：题目定义了两个网页跳转概率与距离的函数，要求解所有网页之间的最短跳转距离。显然是求多源最短路径问题，应用图论中的弗洛伊德算法可以解决。

输出结果：

```
1        1.17233 1.15794 3.50497 2.28318 3.28448 2.38637 2.55831 2.21781 2.21146
1.17233 1        2.32425 2.33263 2.18664 2.32096 1.21403 1.41583 1.04548 2.09556
1.15794 2.32425 1        2.75304 1.12524 2.12654 1.93798 1.40036 1.27877 1.05352
3.50497 2.33263 2.75304 1        2.49813 2.36463 1.2577  2.83049 3.37811 1.69952
2.28318 2.18664 1.12524 2.49813 1        1.0013  2.10823 1.69578 1.14116 1.59413
3.28448 2.32096 2.12654 2.36463 1.0013  1        1.10693 1.0004  2.14246 2.13137
2.38637 1.21403 1.93798 1.2577  2.10823 1.10693 1        2.10733 2.25951 1.64177
2.55831 1.41583 1.40036 2.83049 1.69578 1.0004  2.10733 1        2.35128 1.13097
2.21781 1.04548 1.27877 3.37811 1.14116 2.14246 2.25951 2.35128 1        2.33229
2.21146 2.09556 1.05352 1.69952 1.59413 2.13137 1.64177 1.13097 2.33229 1
```