LogCluster 是一个开源的基于 perl 语言的命令行日志分析工具，能够从大量的蕴含了事件的日志数据文件中挖掘出有意义的日志模式并对日志进行聚类，通过传入一系列的参数和参数值，来改变 LogCluster 的聚类算法分析效果。

使用该工具时可传入的主要参数说明如下表所示：

参数	说明
--input=<file_path>	要进行分析的日志文件路径
--support=<support>	● 整数。 ● 控制结果输出，只有聚到一个类的日志行数不少于 support 的时候才会输出展示。 ● 默认值为 100
--separator=<word_separator_regexp>	● 正则表达式 ● 用来对一条日志中的字符串进行分隔处理的分隔符正则表达式。 ● 默认值为空格
--wweight=<word_weight_threshold>	● 0~1 之间的浮点数 ● 簇之间通过单词之间的相互依赖权重来判断能否连接合并，若超过该值则两个簇可合二为一。【单词之间的相互依赖权重的具体计算方法见后文】 ● 默认值为 0.5
--weightf=<word_weight_function>	● 正整数 ● 计算*单词依赖权重*的方程，如果输入为 1，则使用函数一进行计算；如果输入大于 1，则使用函数二进行计算。【函数一和函数二的具体说明见后文】 ● 默认值为 1
--wfreq=<word_frequency_threshold>	● 0~1 之间的浮点数 ● 若一个单词的出现频度 f > word_frequency_threshold，则该单词被视为*高频词*。【单词频度的具体计算过程见后文】 ● 默认值为 0.5
--write_words=<word_file_path>	● 记录*高频词*的文档路径 ● 当工具检测到*高频词*以后，会将其写入该文档 ● 默认路径为 logcluser 工具目录下的 /WriteFiles.words.txt

关于参数 wweight、weightf 的具体说明：

（1） wweight：单词之间*依赖度*由两个单词同时出现的日志数目度量。假设 a，b 为*高频词*，a 在 m 条日志中出现，其中 n 条日志同时出现了 b，那么 a 相对于 b 的*依赖度*定义为

$$dep(a, b) = n / m$$

注意：dep(a, a)=1 ， 0<dep(a, b)<1

dep(a, b) 越高，则说明 a 与 b 同时出现的可能性越大。从而引出*单词依赖权重*的定义。假设 w1,…,wk 分别是一条*固定句式*中的单词。那么*单词依赖权重*定义为

$$f1(wi) = \sum_{j=1} k \ dep(wj, wi) / k$$

f1(wi) 越小，说明 wi 与其他单词一同出现的可能性越小，则 wi 为 token 的可能性越高。**wweight 为*单词依赖度*阈值，若 f1(wi)<wweight，则 wi 为 token，将会从*固定句式*中删除。**

（2） weightf：上述 f1(w) 存在一些显著的缺点，如果*固定句式*中单词数量较少或者一个单词重复出现多次，f1(w) 的值会变得很大,不能正确反应单词之间的*依赖度*。为克服上述缺点，作者提出 f2(w)。假设 U 为*固定句式*中单词去重后的集合，p 为集合中单词总数，即 p = |U|。例如，对于句式：interface *{1,1} down: interface *{1,2} fault，其 U= { interface, down, fault }，p=3。

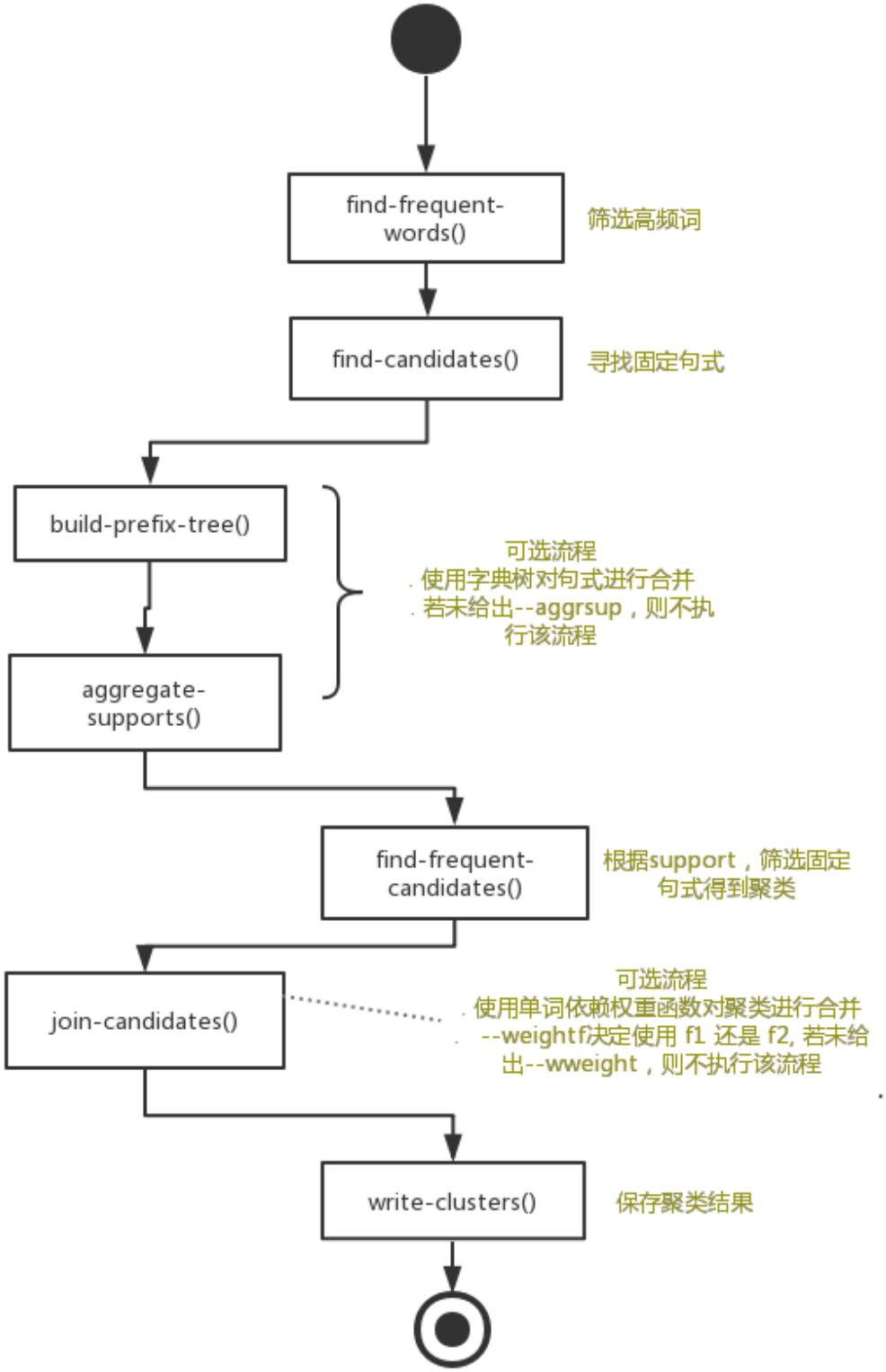
f2(w) 定义如下

$$f2(w) = ((\sum v \in U \ dep(v, w)) - 1) / (p - 1), \quad if\ p > 1$$

$$f2(w) = 1, \quad if\ p = 1$$

如果 weightf=1，则使用 f1(wi)进行计算；如果 weightf >1，则使用 f2(wi)进行计算

算法流程:


find-frequent-words()
筛选高频词
find-candidates()
寻找固定句式
build-prefix-tree()
aggregate-supports()
可选流程
. 使用字典树对句式进行合并
. 若未给出--aggrsup，则不执行该流程
find-frequent-candidates()
根据support，筛选固定句式得到聚类
join-candidates()
可选流程
. 使用单词依赖权重函数对聚类进行合并
. --weightf决定使用 f1 还是 f2, 若未给出--wweight，则不执行该流程
write-clusters()
保存聚类结果

算法分析:

find_frequent_words()

读入所有日志文件，统计日志中所有单词的出现频数，若出现频数>support，则该单词视为*高频词*，存入 fwords 字典中。若给出了 --write_words 参数，在函数结束后会将*高频词*写入指定路径（默认路径为/WriteFiles/words.txt）

考虑下面的日志

User bob login from 10.0.0.1
User alice login from 10.0.0.1
User jim login from 10.0.0.2
User Srv Admin login from 10.0.0.3

假设 support=3，那么 User, login, from 就被视作*高频词*

find_candidates()

对每条日志寻找其中蕴含的*固定句式*，区分出其中的参数。作者认为*固定句式*由*高频词*构成，非高频词视作参数。对每一条日志，candidate 为日志中出现的*高频词序列*，*vars* 数组保存*高频词*之间参数的单词数目。然后以 candidate 作为 key，合并具有相同*高频词序列*的句式，存入 candidates 字典中，统计*固定句式*出现的频数。例如，以上述日志第一条为例，

User bob login from 10.0.0.1

其*高频词序列* candidate 为：*User-> login-> from* ，*vars* 数组为[0, 1, 0, 1]，*（User 前面没有参数，故 vars[0]=0, User 与 login 之间有一个参数 bob，vars[1]=1）*
得到第一条日志的句式一为

*User *{1} login from *{1}*

而对于上述第四条日志，

User Srv Admin login from 10.0.0.3

其*高频词序列* candidate 为 *User-> login-> from*，*vars* 数组为**[0, 2, 0, 1]** *(Srv Admin 占两个单词)*，得到第四条日志的句式二为

*User *{2} login from *{1}*

统计日志文件，可得句式一出现频数为 3（前三条日志），句式二出现频数为 1（第四条日志）。由于句式一和句式二的*高频词序列*相同，故合并句式一与句式二，总出现频数为 4，得到最终的*固定句式*如下

*User *{1,2} login from *{1,1}*

**{1,2} 表示该参数单词数目最少为 1，最多为 2。*

最后若给出了--weightf 参数，在函数结束后会计算*固定句式*中单词的*依赖度 dep(a,b)*，方便后续使用*单词依赖权重*函数进行聚类合并。

aggregate_supports();

若给出了-- aggrsup 参数，则在执行 *find_frequent_candidates()*前先执行该函数。首先调用 *build_prefix_tree()* 将*固定句式*依次插入字典树中。

字典树每个节点由高频词和其对应的 vars[i] 构成，例如对于上述*固定句式*

*User *{1,2} login from *{1,1}*

*可改写成 *{0,0} User *{1,2} login *{0,0} from *{1,1}*

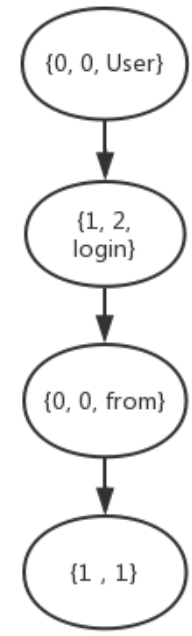
将会产生节点 {0, 0, User}，{1, 2, login}，{0, 0, from}，{1, 1} 依次插入*字典树*。在插入的过程中，如果在该路径上已经存在相同的节点，则增加节点的计数，否则创建新的节点插入。

考虑下述情况，共有两条*固定句式*

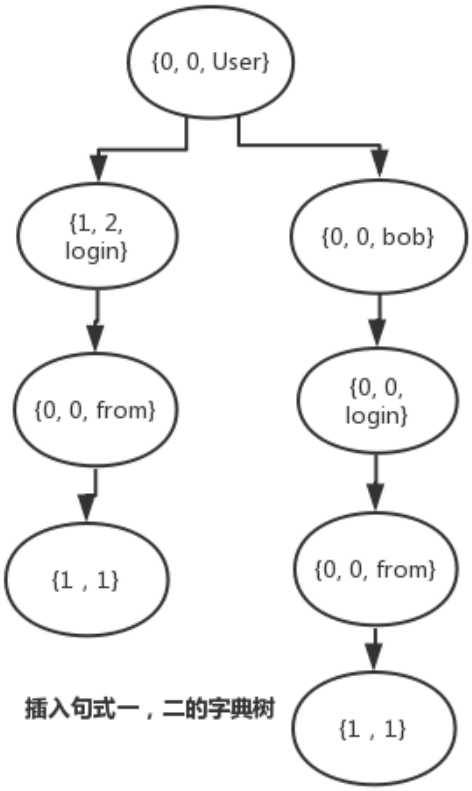
*User *{1,2} login from *{1,1}*

*User bob login from * {1,1}*

插入句式一*字典树*如下方右图，插入句式一，二的*字典树*如左图。



插入句式一的字典树



插入句式一，二的字典树

构建完*字典树*后，aggregate_supports 对每一条*固定句式*调用 find_more_specific() 寻找该*固定句式*的更为特例的情况（如上，句式二就是句式一的更为特例的情况），将其合并，得到合并后的*固定句式*。

find_frequent_candidates()

对上述得到的*固定句式*进行筛选，若*固定句式*出现频数 > support 则找到一个聚类，删除出现频数小于 support 的*固定句式*.

join_candidates()

若给出了--wweight 参数，在执行完 *find_frequent_candidates() 后*则会执行该函数。首先根据 weightf 选择*单词依赖权重*函数 f(w)，将 f(w) < wweight 的单词从*固定句式*中删除。(f(w) 的计算公式参见 wweight, weightf 的参数详解）然后合并 *删除单词后固定句式*相同的聚类,统计聚类中日志总数,得到最终的聚类结果。