

FST

FST (Finite State Transducer) 算法的概念在这篇博客中并不涉及，网上有太多的资料啦，写的都非常的不错。这里推荐这位网友的介绍：<https://www.shenyanchao.cn/blog/2018/12/04/lucene-fst/>。如果链接失效了，可以看附件中的副本。本文中，我们基于一个例子来介绍在Lucene中如何实现FST算法及应用，感谢网友 关新全 的分享，基于他的分享使得我在看源码的时候事半功倍，在此基础上，增加一些更加贴近源码的内容。同样的关新全同学分享的文章在附件中。

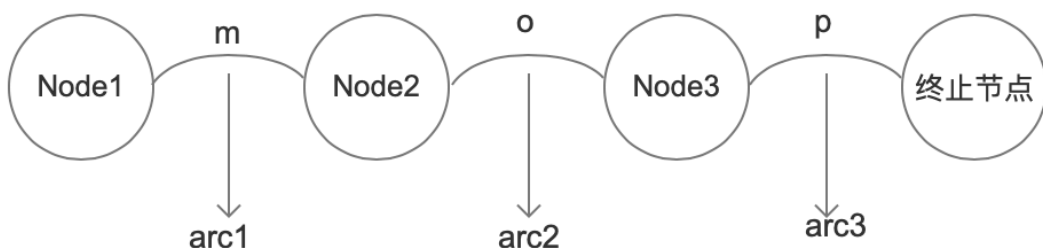
准备工作

先介绍几个重要的类跟变量

Arc< T>类

每一个Arc< T>对象封装了某个我们需要存储的数据，并且每一个Arc< T>对象属于某一个Node对象，一个Node对象可以包含多个一个Arc< T>对象，用Arc< T>[] 数组的方式存储，终止节点是没有Arc< T>对象的。Node类的子类有UnCompiledNode< T>类跟CompiledNode 类，下文会介绍

```
public static class Arc<T> {  
    public int label;  
    public Node target;  
    public boolean isFinal;  
    public T output;  
    public T nextFinalOutput;  
}
```



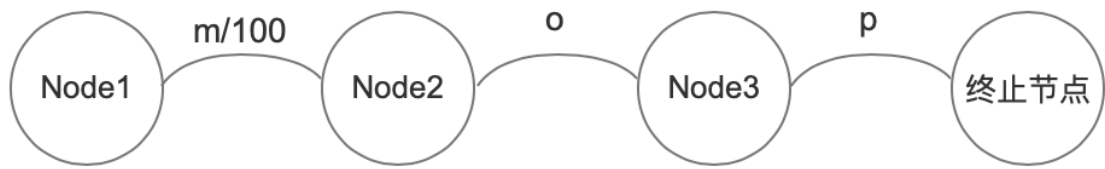
例如 “mop”，会使用三个Arc<T>对象来分别描述“m”，“o”，“p”的三个字符的信息，并且这三个对象属于三个不同的Node，如上图，描述“m”的Arc<T>对象arc1，它属于Node1节点，对象arc1中的label的值是 109，即“m”的ASCII值；对象arc1的target的值即Node2，这里暂时不讨论isFinal、nextFinalOutput的含义，后面会涉及，下面介绍下output的值：

output

output值是一个输入的附加值，比如有输入“mop”，它的附加值是100，但是对于分别封装了“m”，“o”，“p”的三个arc对象，附加值100应该存放哪个arc对象的output字段呢？下面通过一个例子来说明：

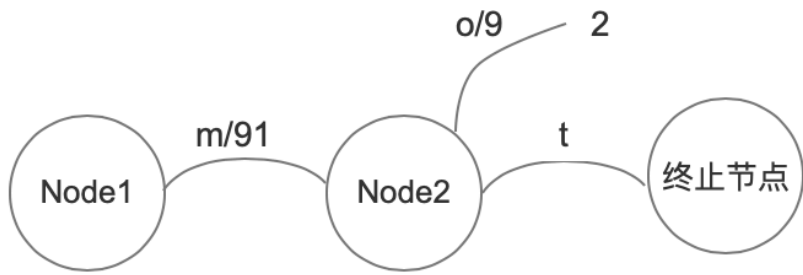
例子：输入为 “mop”，“mt”，附加值分别是100跟91。

处理“mop”， 100



只处理输入 "mop" 时，附加值是放在“m”的arc对象中。

处理“mt”， 91



上图中我们能发现，如果后面的输入跟前一个输入有相同的前缀，那么相同前缀对应的每一个arc的output都有可能被更新，在这个例子中，“mop”的附加值100被拆分为91跟9，因为“mt”的附加值是91，取两者的较小值来更新封装了“m”的arc对象中的output值。
上图中封装了“p”的arc怎么没有了，多出了一个值2？这里不用关心这一点，下文会解释。

BytesStore bytes；

在这里，我们只需要知道在BytesStore类中有一个current[]数组，它用来存放四种数据来描述一个arc的信息：

index

当前arc对象描述的字符不是数据的最后一个字符时，会存储一个index值来指向下一个字符在current[]数组中的位置。

output

arc的output字段的值，如果output为0，那么就不用存储，在查询阶段，会对一个term的所有arc的output值进行整合，合成一个完整的值。

label

label值即我们的输入数据，比如我们的输入“mop”，对应三个label值，其中的一个label值就是109，即“m”的ASCII值。

flag

flag的值用来在读取构建后的FST时使用，它用来描述一个arc的信息，在当前版本7.5.0中，flag有以下值：

BIT_FINAL_ARC:	1	arc指向一个终止节点
BIT_LAST_ARC:	2	arc是Node节点中的最后一个arc
BIT_TARGET_NEXT:	4	上一个UnCompiledNode< T>类型的Node处理结束后新生成的CompiledNode类型的Node是当前arc的target结点，用来描述当前的arc封装的字符不是数据的最后一个字符
BIT_STOP_NODE:	8	arc的target是一个终止节点
BIT_ARC_HAS_OUTPUT:	16	arc有output值(output不为0)
BIT_ARC_HAS_FINAL_OUTPUT:	32	arc有output值，并且output的值是最终的值(下文会解释)

UnCompiledNode< T>类；

前面的图中 Node1, Node2, Node2都是UnCompiledNode<T>类的对象，即节点。节点中可以包含多个arc，并且arc的信息未存放到current[]数组中。

CompiledNode类；

CompiledNode类描述了一个arc的信息已经被写入到current[]数组中，比如上图中消失的那个封装了“p”的arc对象，以及多出的值2，这个值2描述了在current[]数组中存储封装了“p”的arc对象的数据的下标位置。

NodeHash< T>类

NodeHash< T>类提供了对节点的Hash存储，目的就是减少空间占用，每次对一个UnCompiledNode< T>对象处理前，先判断之前有没有处理过(判断Hash值是否一致)，如果有，那么就不用存储到current[]数组中，实现了后缀存储功能。

UnCompiledNode< T>[] frontier；

存放UnCompiledNode< T>类对象的数组，即存放节点的数组。上文中的图中的Node1, Node2, Node3都是frontier[]数组的数组元素

long lastFrozenNode;

构建FST时，每次处理完一个节点，就会返回一个long类型的值，如果是 终止节点那么返回固定值 -1，否则返回在current[]数组中的下标值。

算法基本步骤

步骤1：处理上一个输入与当前输入不相同Node节点，将Node节点中的所有arc对象的信息写入到current[]数组中，如果是第一个输入，直接执行下一个步骤。步骤2：将当前输入写入到frontier[]数组中。步骤3：处理上一个输入与当前输入相同的前缀值，调整output值。

例子

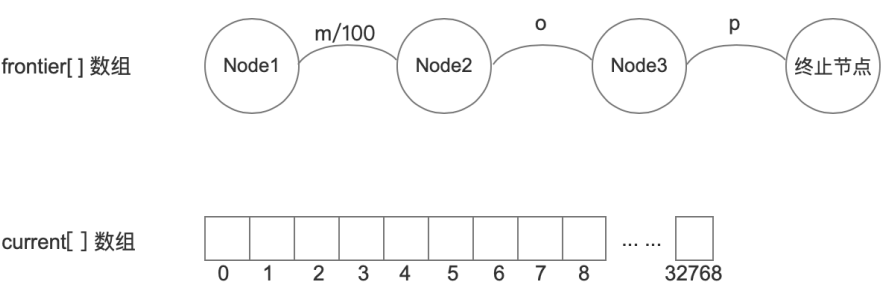
下面通过一个例子来介绍FST的构建跟查询，输入跟附加值如下：

```
String[] inputValues = {"mop", "moth", "pop", "star", "stop", "top"};
long[] outputValues = {100, 91, 72, 83, 54, 55};
```

注意的是输入顺序必须是有序的，才能获得一个最小FST。例子中的输入已经默认有序的

输入mop

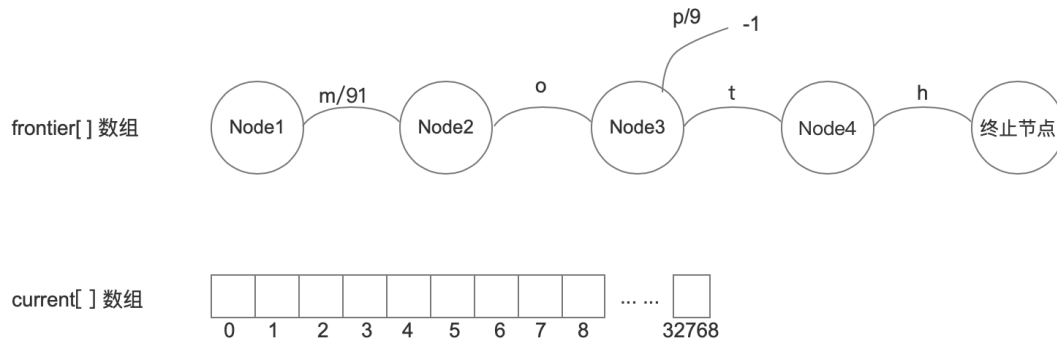
这是第一个输入，所以没有写入current[]数组的操作。



输入moth

输入moth后，我们要处理上一个输入“mop”的不相同的结点，可以看出不相同的节点是前一张图中Node3后面一个终止节点，所以要将这个节点的所有arc信息写入到current[]数组中，并获得一个在current[]中的下标值，从代码层面描述就是将 UnCompiledNode<T>对象变成CompiledNode对象，由于是终止节点，没有arc对象，所以返回值是固定值 -1。并且这个值不需要写入到current[]数组中，最后更新lastFrozenNode为 -1

以上执行的步骤1跟步骤2，剩下执行步骤3，更新Node1中“m”的附加值，由100变成91。并且“p”的附加值由0变为9。



输入pop

输入pop后，与上一个值没有相同的结点，所以我们要将Node2，Node3，Node4，终止节点的所有arc信息写入到current[]数组中，注意的是Node1中的所有arc只有在所有输入处理结束后才会处理。所有的处理的顺序为：节点间按照从后往前，节点内的arc按照写入到该结点时候的顺序，在当前情况下，处理的arc顺序应该是：终止节点->Node4 (h) -> Node3 (p -> t) ->Node2(o)

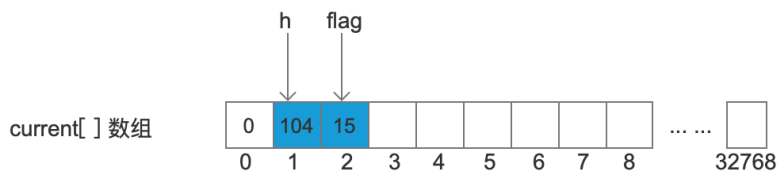
处理终止节点

终止节点返回值为固定值-1，并更新lastFrozenNode为 -1。current[]数组不变



处理Node4 (h)

Node4中只有“h”对应的arc对象，根据flag的定义，这个arc对象，它满足Node4节点中的最后一个arc (BIT_LAST_ARC)、arc的target结点的值跟lastFrozenNode一致都为-1 (BIT_TARGET_NEXT)、arc指向一个终止节点 (BIT_FINAL_ARC)、arc的target是一个终止节点 (BIT_STOP_NODE)、所以flag的值为 BIT_LAST_ARC(2) + BIT_TARGET_NEXT(4) + BIT_FINAL_ARC(1) + BIT_STOP_NODE(8) = 15。然后将flag跟“h”的ASCII值写入到current[]数组中，最后更新lastFrozenNode的值为2，该值为flag在current[]数组中的下标值。



处理Node3 (p -> t)

由于UnCompiledNode<T>类中使用 Arc<T>[] 数组存储所有的arc，所以按照存储到该数组的先后顺序进行处理。同时也意味着“t”对应的arc是Node3中的最后一条arc。

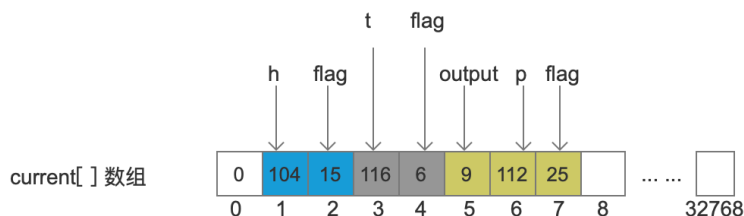
处理“p”

“p”对应的arc对象，它满足 arc指向一个终止节点 (BIT_FINAL_ARC)、arc的target是一个终止节点 (BIT_STOP_NODE)、arc有output值(9) (BIT_ARC_HAS_OUTPUT)、所以flag的值为 $\text{BIT_FINAL_ARC}(1) + \text{BIT_STOP_NODE}(8) + \text{BIT_ARC_HAS_OUTPUT}(16) = 25$ 。由于还有其他arc未处理，所以暂时不能更新lastFrozenNode的值。

处理“t”

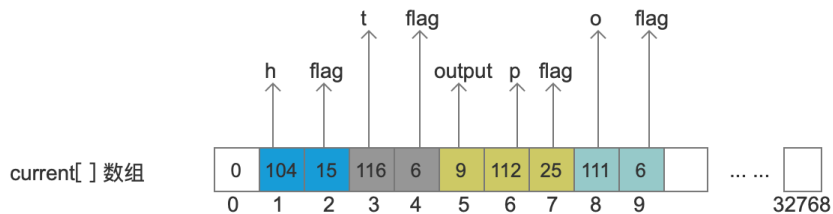
“t”对应的arc对象，它满足 Node3节点中的最后一个arc (BIT_LAST_ARC)、arc的target结点的值跟 lastFrozenNode是相同的 (2) (BIT_TARGET_NEXT) 所以flag的值为 $\text{BIT_LAST_ARC}(2) + \text{BIT_TARGET_NEXT}(4) = 6$ 。Node3中所有的arc处理结束，更新lastFrozenNode的值为7，7是Node3的第一个arc的flag在current[]数组中的下标。

这里有个注意点就是，当按照 p -> t的顺序处理结束后，会对刚才存储的数据执行逆置操作。所以“t”跟“p”对应的arc对象的数据在current[]数组中的位置就如下图：

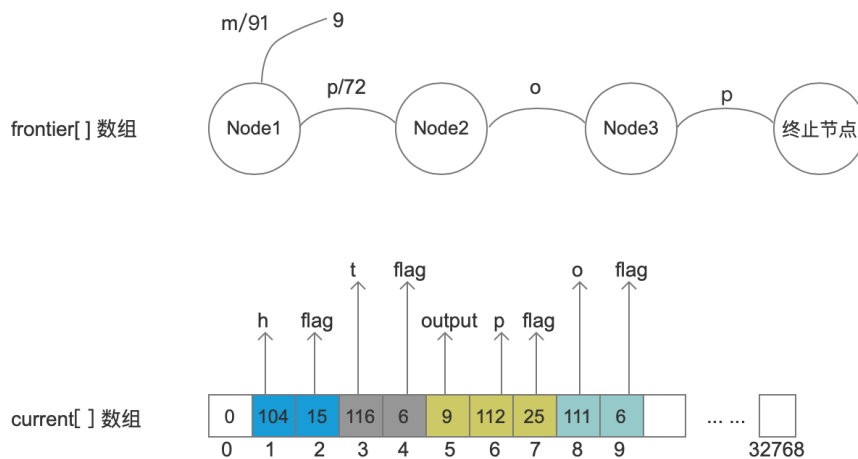


处理Node2 (o)

"o"对应的arc对象，它满足 Node2节点中的最后一个arc (BIT_LAST_ARC)、arc的target结点的值跟 lastFrozenNode是相同的，所以flag的值为 $\text{BIT_LAST_ARC}(2) + \text{BIT_TARGET_NEXT}(4) = 6$ 。更新 lastFrozenNode的值为9。 注意的是，由于"m"对应的arc的target是指向"o"，所以它的target会被更新为 9



步骤一执行结束后，把“pop”添加到`frontier[]`数组中，如下图：

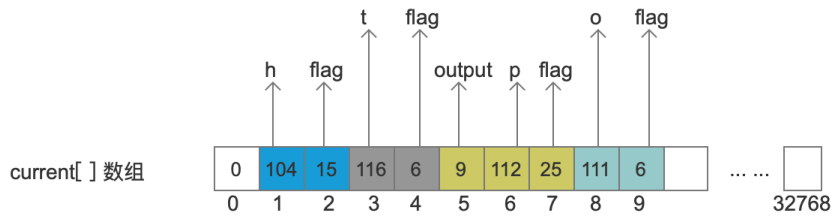


输入 star

输入star后，与上一个值没有相同的结点，所以我们要将Node2，Node3，终止节点的所有arc信息写入到 `current[]` 数组中，同样的是Node1中的所有arc只有在所有输入处理结束后才会处理。所有的处理的顺序为：节点间按照从后往前，节点内的arc按照写入到该结点时候的顺序，在当前情况下，处理的arc顺序应该是：终止节点 -> Node3 (p) -> Node2 (o)

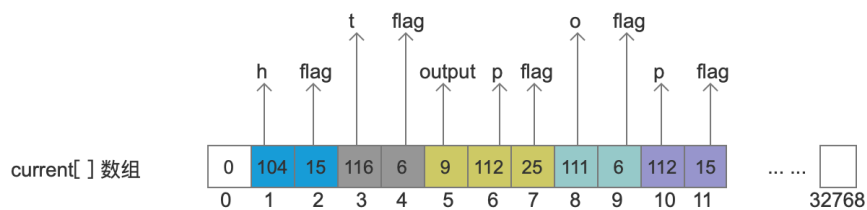
处理终止节点

终止节点返回值为固定值-1，并更新lastFrozenNode为 -1。current[]数组不变



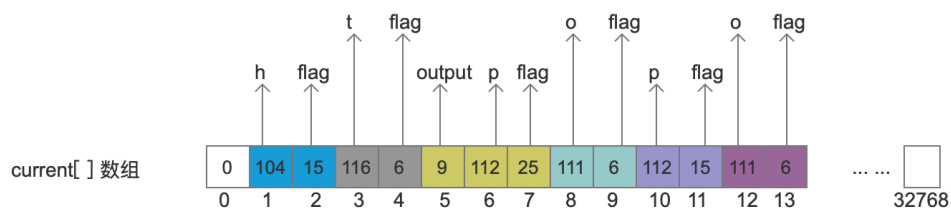
处理Node3 (p)

"p"对应的arc对象，它满足 Node3节点中的最后一个arc (BIT_LAST_ARC)、arc指向一个终止节点 (BIT_FINAL_ARC)、arc的target结点的值跟lastFrozenNode是相同的 (BIT_TARGET_NEXT)、arc的target是一个终止节点 (BIT_STOP_NODE)、所以flag的值为 $\text{BIT_FINAL_ARC}(1) + \text{BIT_LAST_ARC}(2) + \text{BIT_TARGET_NEXT}(4) + \text{BIT_STOP_NODE}(8) = 15$ 。

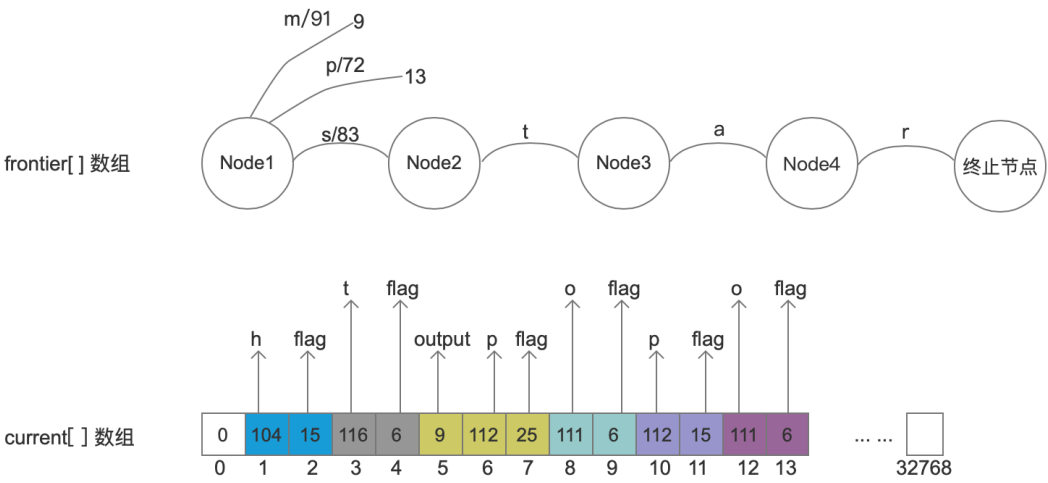


处理Node2 (o)

这个过程就不赘述，跟上面的逻辑没区别

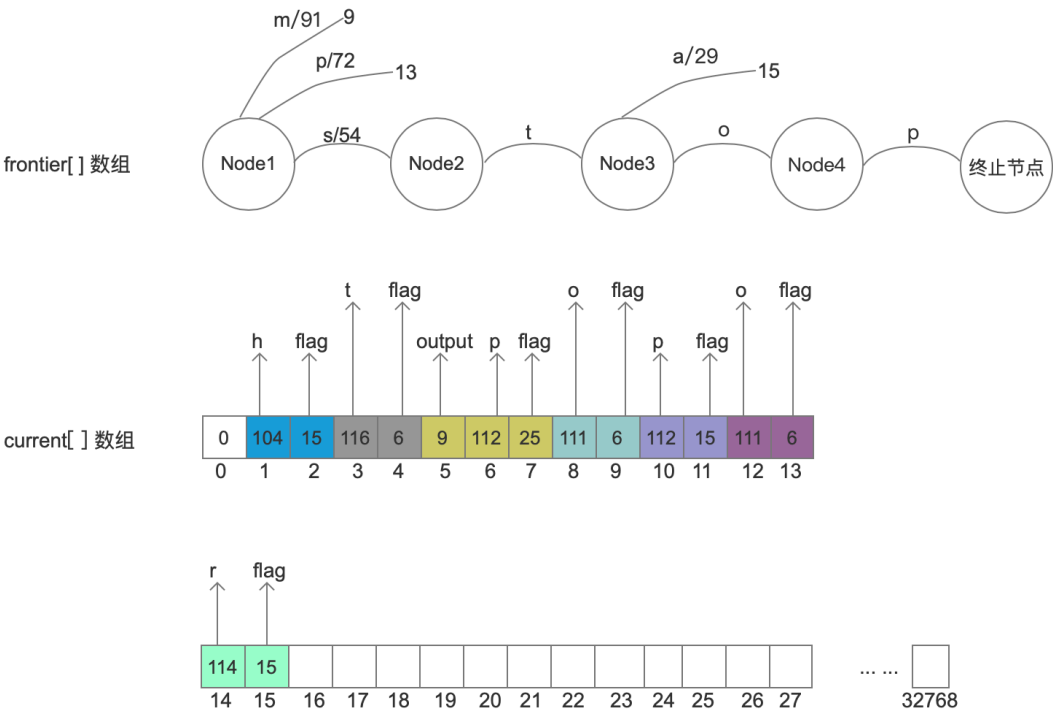


将“star”写入到frontier[]数组中



输入stop

这个过程就不赘述了，跟上面的逻辑没区别



输入top

输入top后，与上一个值没有相同的结点，所以我们要将Node2，Node3，Node4，终止节点的所有arc信息写入到current[]数组中，注意的是Node1中的所有arc只有在所有输入处理结束后才会处理。所有的处理的顺序为：节点间按照从后往前，节点内的arc按照写入到该结点时候的顺序，在当前情况下，处理的arc顺序应该是：终止节点->Node4 (p) -> Node3 (a -> o) ->Node2(t)

处理终止节点

终止节点返回值为固定值-1，并更新lastFrozenNode为 -1。current[]数组不变

处理Node4 (p)

处理“p”对应的arc对象时，根据NodeHash提供的后缀存储发现，之前存储“pop”的第二个“p”对应的UnCompiledNode<T> 类型的对象Hash值相同，所以这里直接就可以使用已经处理（compile）过的的CompiledNode对象，所以current[]没有改变。

处理Node3 (a -> o)

处理“a”

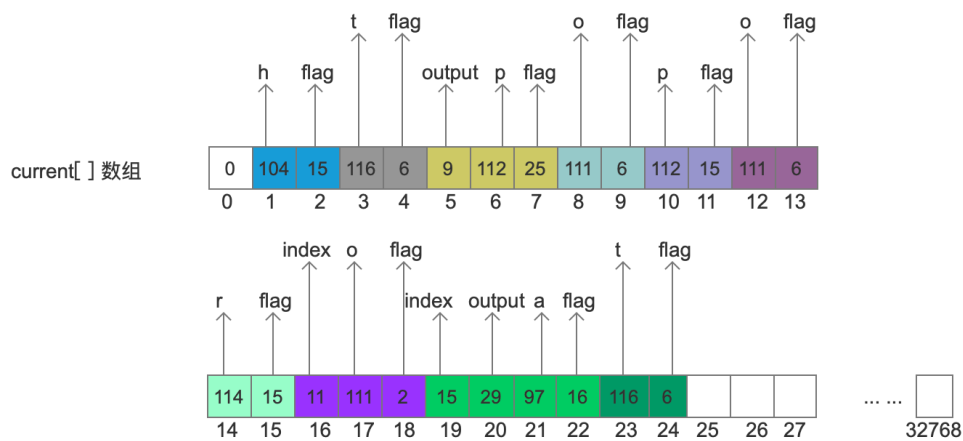
“a”对应的arc对象，它满足 arc有output值(29) 所以flag的值是BIT_ARC_HAS_OUTPUT(16) = 16，另外a的target的值是大于0的，说明“a”不是数据的最后一个字符，这个值描述下一个字符在current[]数组中的位置15，值为。

处理“o”

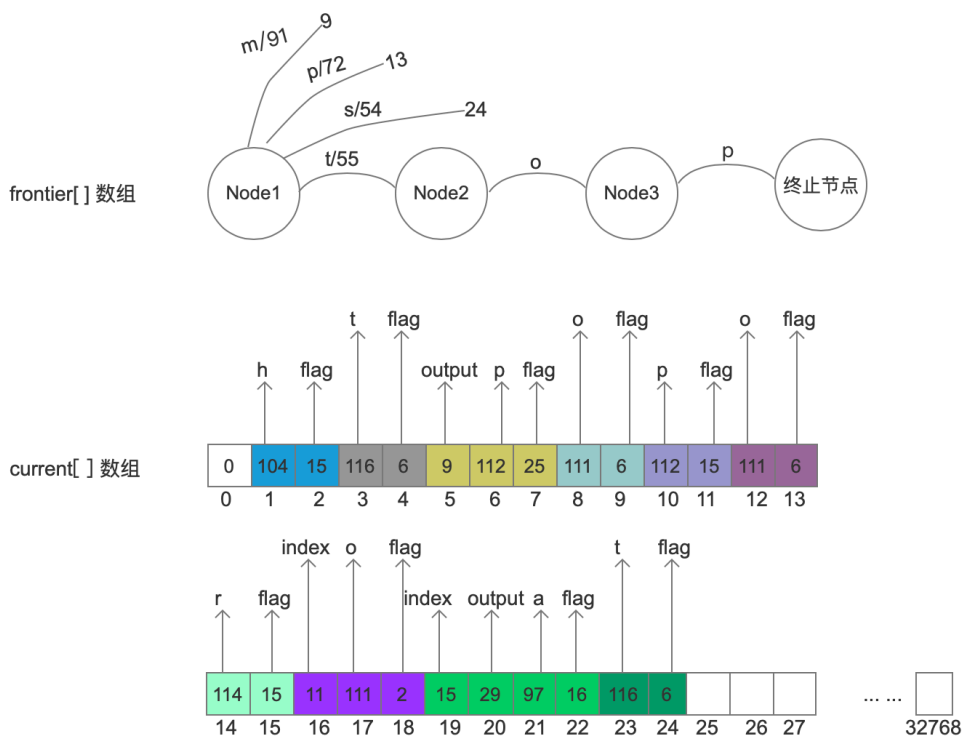
“o”对应的arc对象，它满足 Node3节点中的最后一个arc (BIT_LAST_ARC)，所以flag的值是2，另外a的target的值是大于0的，说明“o”不是数据的最后一个字符，这个值描述下一个字符在current[]数组中的位置，值为11。

处理Node2 (t)

“t”对应的arc对象，它满足 Node2节点中的最后一个arc (BIT_LAST_ARC)、arc的target结点的值跟lastFrozenNode是相同的 (BIT_TARGET_NEXT)，所以flag的值为 BIT_LAST_ARC(2) + BIT_TARGET_NEXT(4) = 6。最后更新lastFrozenNode的值为24。

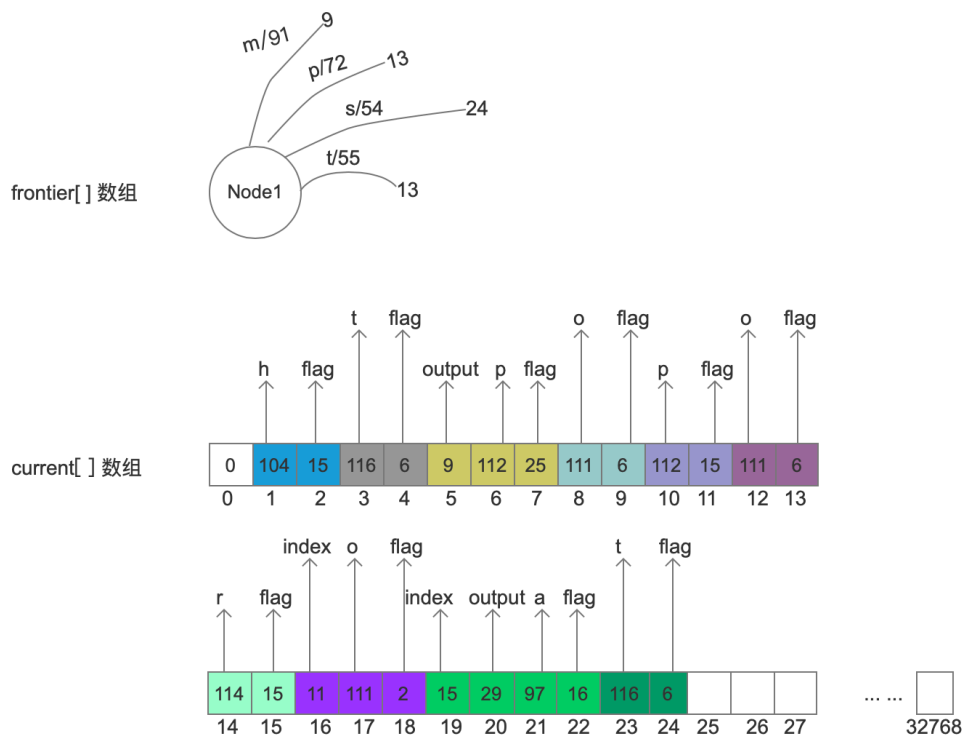


将“top”写入到frontier[]数组中



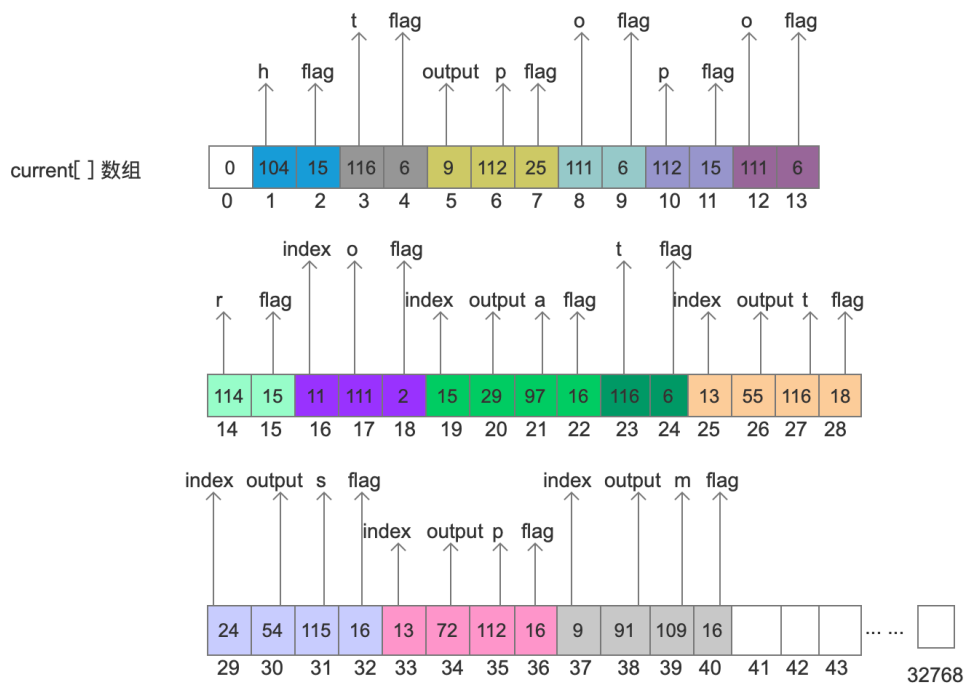
处理top

“top”是最后的输入，所以在将“top”写入到frontier[]数组后，直接处理，处理过程不赘述，跟上面的逻辑没什么区别，由于“top”跟“pop”有2个相同的后缀值，所以处理完“top”后，current[]数组没有变化



最后处理Node1节点

不赘述，跟前面的逻辑是差不多的



结语

本篇博客介绍了FST的构建过程，在随后的博客中会介绍如何使用current[]数组来恢复数据。

[点击下载](#)Markdown文件