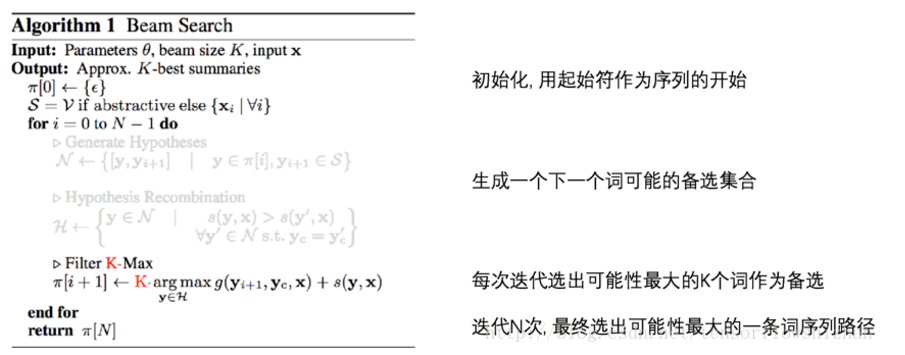
### **Beam Search生成摘要(decode)**

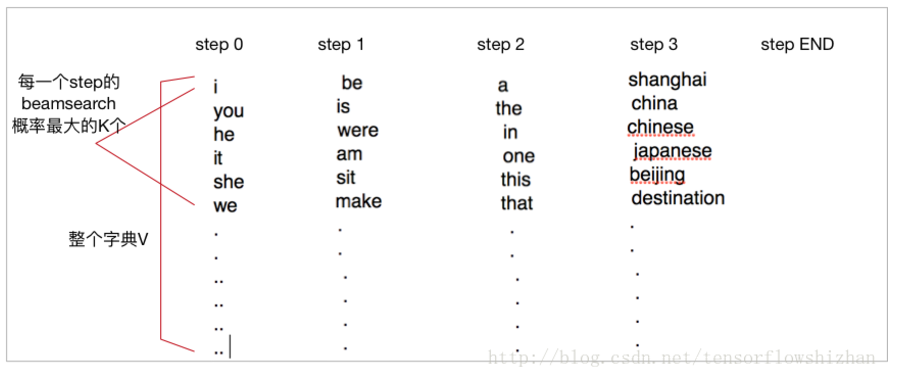
我们现在回到生成摘要的问题. 回顾前面, 我们的目标是找到:

  
 Y是长度为N的序列y组成的集合, 如果字典中的单词数量是V的话, 我们要生成的这个摘要就有种可能性. 因为我们这里已经做了处理, 只根据前面的C个已经预测出的单词yc来预测下一个词yi+1. 这样算法复杂度变成了O(). 但是即使是这样, 这个算法也太复杂了.   
 使用维特比译码需要O().复杂度获得精确的解. 然而在实际中V太大使得问题难解. 一个替代方法是使用贪婪解来近似获得argmax, 只保证每次前进的一小步是概率最大的.   
 在精确解和贪婪解方法之间取一个折中, 就是beam-search束搜索解码器(Algorithm1), 它在保持全量字典V的同时, 在输出摘要的每一个位置上将自己限制在K个潜在的假设内. 这种beam-search方法在神经机器翻译模型NMT也很常用. Beam search算法展示如下:

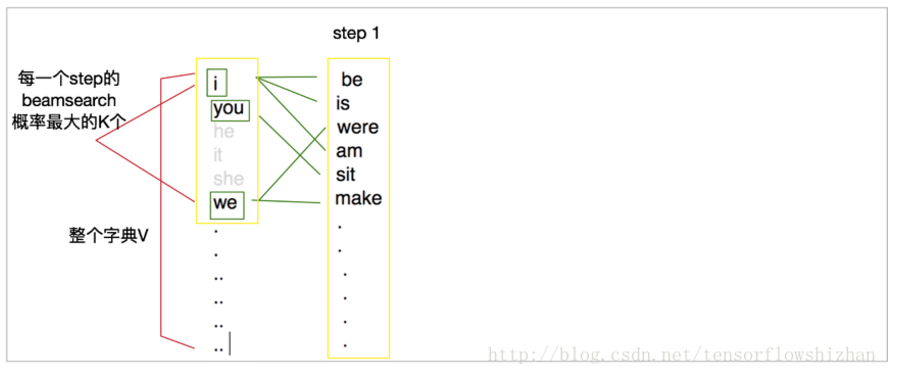
  
IMG_256  
参数说明:   
N: 摘要的长度   
K: beam的尺寸   
V: 字典里所有单词的数量   
C: 关注的词序列的长度

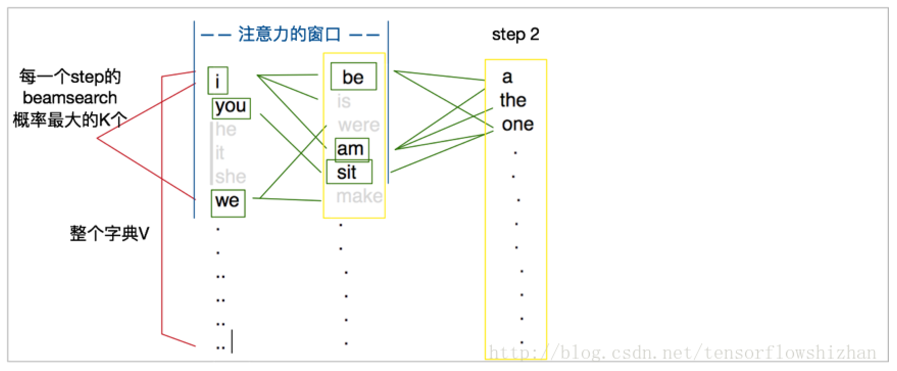
#### Beam search案例

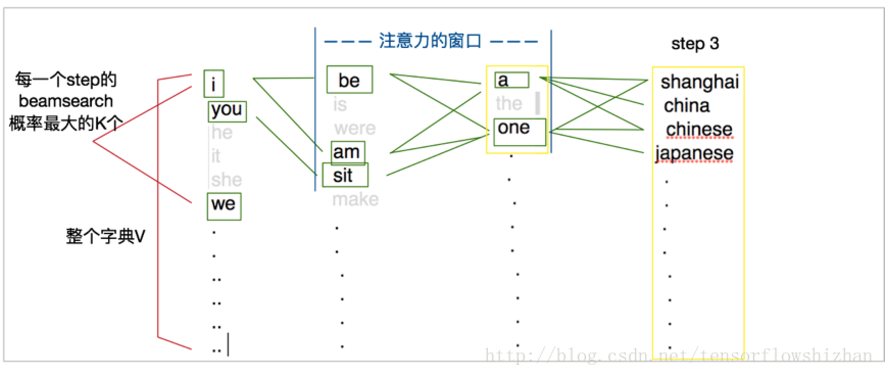
下面举一个简单的例子来说明beam search算法的运行过程. 在这个例子里, 摘要长度N=4, beam的大小K=6, 注意力窗口大小C=2, 模型最理想的结果是‘i am a chinese’. Beamsearch的每一次迭代都从字典V里找K个最大的可能.

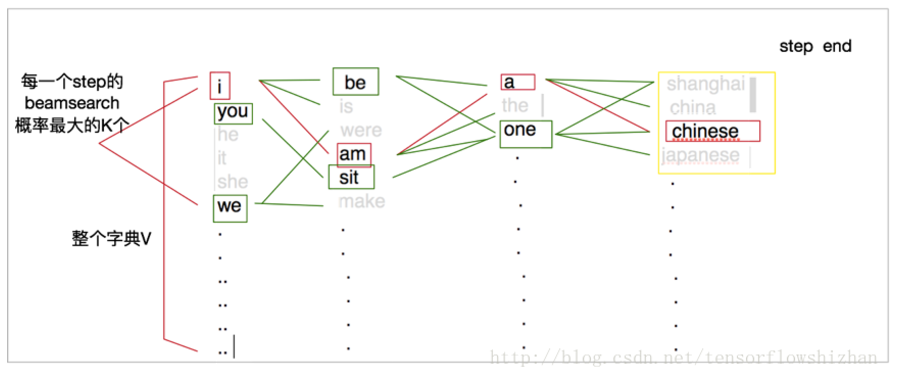
  
Step1: 预测前C个词的时候窗口溢出的部分需要进行padding操作, 预测第1个词的时候我们选出K个词符.

  
Step2: 预测第2个词的时候, 我们选出新的K个词符, 对应K条备选路径. 前一阶段概率低的路径和词符, 被抛弃掉.

  
Step3: 重复前面的过程.

  
Step4: 每次beam search不一定能选出不同的K个词, 但是每次beam search都找到最优的前K个路径, 路径可以有重叠.

  
Step5: 迭代N次, 最终选出可能性最大的一条词序列路径

  
下面是对Beam Search算法的详细分析, 对原文的Algorithm 1逐条进行解释.

#### Beam Search算法分析

1.π[0]是可以用规定好的起始符号

<s>

来初始化. 在训练和生成摘要时, 窗口Q和C沿着文本滑动如果超出范围, 用起始符号

<s>

做padding.

2.如果模型是abstraction-based, 输出y的备选集合是整个字典, 如果希望摘要的单词全部从原文中抽取, 那么词典由输入正文x的所有单词构成.

3.我们会设定一个最大输出长度N, 算法会进行N轮迭代.

(1)现已有K个假设, 每一个假设都对应一条路径; 对每一个假设, 我们从字典S(有V 个单词)中选出K个单词作为备选.

(2)在字典中寻找, 搜索其他单词, 如果计算的到的state值比当前集合中的任意一个大, 就把它保留下来.

(3)当每一个假设都遍历完整个字典S, 就会产生K×K条路径, 我们在这些路径中选择概率最大的K个路径作为下一次迭代的基础.(每一条路径都保留了之前i?1个节点对应的单词)

4.当N次迭代进行完后, 我们只剩下了K条路径, 最后在从这其中选出1条概率最大的即可.

5.路径所经历的所有节点即为摘要的单词. 如果这中间遇到了停止符

<e>

摘要就是从

<s>

到

<e>

如果没有

<e>

出现, 摘要的最大长度就是N.

Beam Search的运算复杂度从O()变成了O(KNV), 因为V>>N和K, 加速效果非常显著. 束搜索依据已经计算好的路径以及当前的V个备选值, 计算出最优的K的值. 最新的K个最优值都保留着相应路径上之前的所有的节点.