动态规划与中文分词

动态规划：概念

- 动态规划(dynamic programming)是通过组合子问题的解来求解原问题(在这里，"programming"指的是一种表格法，并非编写计算机程序)
- 动态规划应用于子问题重叠的情况，即不同的子问题具有公共的子子问题(子问题的求解是递归进行的，将其划分为更小的子子问题)
- 动态规划对于每个子子问题只求解一次，将其解保存在一个表格中，从而无需每次求解一个子子问题都重新计算，避免了不必要的操作

动态规划：概念

- 动态规划通常用来求解最优化问题(optimization problem)，这类问题可以有很多可行解，每个解都有一个值，我们希望寻找具有最优值(最小值或最大值)的解，我们称这样的解为问题的一个最优解(an optimal solution)，而不是最优解(the optimal solution)，因为可能多个解都达到最优值

- 适用范围：
  - 一类优化问题
  - 可分为多个相关子问题
  - 子问题的解被重复使用

动态规划：条件

- 优化子结构(optimal substructure)
  - 当一个问题的优化解包含了子问题的优化解时，我们说这个问题具有优化子结构
  - 缩小子问题集合，只需那些优化问题中包含的子问题，降低实现复杂性
  - 优化子结构使我们能自下而上地完成求解过程
- 重叠子问题(subtities)
  - 在问题的求解过程中，很多子问题的解将被多次使用

动态规划：设计步骤

1. 刻画一个最优解的结构特征
2. 递归地定义最优解的值
3. 计算最优解的值，通常采用自底向上的方法
4. 利用计算出的信息构造一个最优解

- 步骤1-3是动态规划算法求解问题的基础。如果我们仅仅需要一个最优解的值，而非解本身，可以忽略步骤4。如果需要做步骤4，有时需要在执行步骤3的过程中维护一些额外信息，以便用来构造最优解

最大概率分词

- 最大概率分词是一种基本的统计分词方法
- 从统计的角度来看
  - 分词问题的输入是一个字串：$C = c_1, c_2, \dots, c_n$
  - 输出是一个词串：$S = w_1, w_2, \dots, w_m$
  - 其中要求$m \leq n$
- 对于一个特定的字符串C，会有多个切分方案S对应，最大概率分词就是在这些S中找出概率最大的一个切分方案，将其看作对于输入字符串切分的最有可能的词序列

最大概率分词

- 由贝叶斯公式可得：

$$Seg(C) = argmax_{S\in G} P(S|C) = argmax_{S\in G} \frac{P(C|S)P(S)}{P(C)}$$

- 其中P(c)是字符串在语料库中出现的概率，为一个固定值
- 从词串恢复到字串只有一种方式，因此$P(C|S) = 1$
- 因此，比较$P(S_1|C)$和$P(S_2|C)$的大小就变成了比较$P(S_1)$和$P(S_2)$的概率

最大概率分词

- 在最大概率分词中，假设每个词的概率是上下文无关的，也就是unigram
- 因此$P(S) = P(w_1, w_2, \dots, w_m) = P(w_1) \times P(w_2) \times \cdots \times P(w_m)$
- 其中，$P(w_i) = \frac{w_i\text{在词典中出现的次数}n_i}{\text{词典中总词数}N}$

最大概率分词

- 在实际操作中，如果字符串比较长，分词的形式就会非常多，计算量和长度呈指数增长
- F(n)表示一个有n个字的句子包含的全部分词方法
- $F(n) = 1 + F(n-1) + F(n-2) + \cdots + F(1)$
  - $F(1) = 1$、$F(2) = 2$、$F(3) = 4$、$F(4) = 8$
  - $F(n) = 2F(n-1)$
- 因此，$F(n) = 2^{n-1}$
- 如果将词频看做距离，则求解最佳切分的方法等价于在2^{n-1}的解空间中寻找一种最佳的切分方法使得路径最大
- 因此引入动态规划算法

最大概率分词

- 主要分为3步：

1. 构造前缀词典
2. 利用前缀词典对输入的句子进行切分，得到所有的切分可能，根据切分位置，构造有向无环图
3. 通过动态规划，计算得到最大概率路径，也就得到了最终的切分形式

最大概率分词：词典

- 以“去北京大学玩”为例，作为待分词的输入文本
- 离线统计的词典形式如下：
- 每一行有3列，第一列是词，第二列是词频，第三列是词性

```
北京大学 2053 nt
大学 20025 n
去 123402 v
玩 4207 v
北京 34488 ns
北 17860 ns
京 6583 ns
大 144099 a
学 17482 n
```

最大概率分词：前缀词典构建

- 前缀词典的构建首先将离线词典中的词及词频提取出来，再分别获取每个词的前缀词，如果已经存在于前缀词典中，则不处理，如果该前缀词不在前缀词典中，则将其词频置为0，便于后续构建有向无环图

```python
# 前缀词典构建
def gen_pfdict(filename):
    lfreq = {}  # 保存前缀词典中的词及词频
    ltotal = 0  # 保存总的词数
    with open(filename) as fp:
        line = fp.readline()
        while len(line) > 0:
            # 保存离线词典中的词及词频
            word, freq = line.split()[0:2]
            freq = int(freq)
            lfreq[word] = freq
            ltotal += freq
            # 对于离线词典中的每个词，获取其前缀词
            for ch in range(len(word)):
                wfrag = word[:ch + 1]
                if wfrag not in lfreq:
                    lfreq[wfrag] = 0
            line = fp.readline()

    return lfreq, ltotal
```

最大概率分词：前缀词典构建

- 该函数对于前面离线词典的输出为：

```
北京大学  2053
北   17860
北京 34488
北京大    0
大学 20025
大   144099
去   123402
玩   4207
京   6583
学   17482
```

最大概率分词：DAG构建

- 有向无环图(directed acyclic graphs)，简称DAG，是一种图的数据结构，表示没有环的有向图

- 采用dict结构存储DAG，最终的DAG是以{k : [k , j , ..] , m : [m , p , q] , ...}的字典结构存储，其中k和m为词在文本sentence中的位置，k对应的列表存放的是文本中以k开始且词sentence[k: j + 1]在前缀词典中的以k开始j结尾的词的列表，即列表存放的是sentence中以k开始的可能的词语的结束位置，这样通过查找前缀词典就可以得到词

最大概率分词：DAG构建

- 算法流程如下：

从前往后依次遍历文本的每个位置，对于位置k，首先形成一个片段，这个片段只包含位置k的字，然后就判断该片段是否在前缀词典中：

1.如果这个片段在前缀词典中：

　　1.1 如果词频大于0，就将这个位置i追加到以k为key的一个列表中；

　　1.2 如果词频等于0，则表明前缀词典存在这个前缀，但是统计词典并没有这个词，继续循环；

2.如果这个片段不在前缀词典中，则表明这个片段已经超出统计词典中该词的范围，则终止循环；

3.然后该位置加1，然后就形成一个新的片段，该片段在文本的索引为[k:i+1]，继续判断这个片段是否在前缀词典中

最大概率分词：DAG构建

- 代码实现如下：

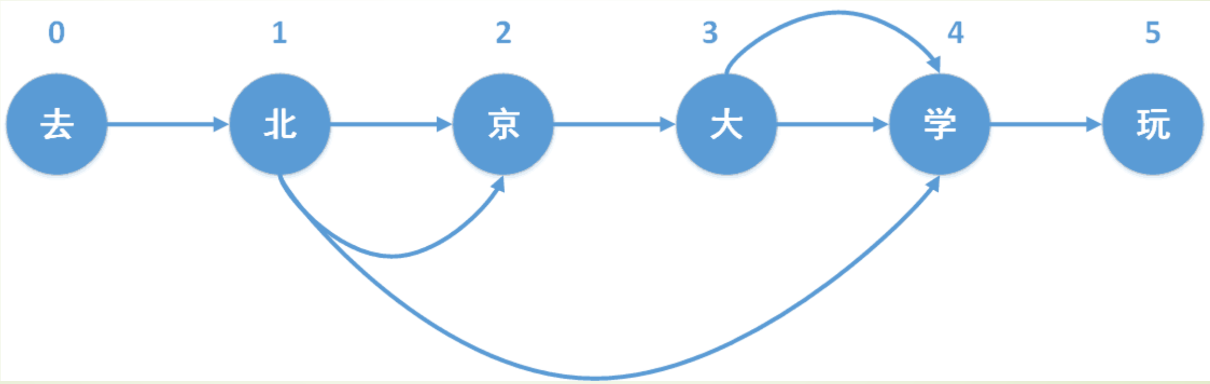
```python
def get_DAG(sentence, lfreq):
    DAG = {}
    N = len(sentence)
    for k in range(N):
        tmplist = []
        i = k
        frag = sentence[k]
        while i < N and frag in lfreq:
            if lfreq[frag] > 0:
                tmplist.append(i)
            i += 1
            frag = sentence[k:i + 1]
        if not tmplist:
            tmplist.append(k)
        DAG[k] = tmplist
    return DAG
```

最大概率分词：DAG构建

- 对于句子“去北京大学玩”，函数的输出结果为：

```
0:[0]
1:[1, 2, 4]
2:[2]
3:[3, 4]
4:[4]
5:[5]
```

- 对应的DAG为：



最大概率分词：最大概率路径计算

- DAG中的每个节点，都是带权的，对于在前缀词典中的词语，其权重就是它的词频，我们想要求得$route = (w_1, w_2, \dots, w_n)$使得$\sum weight(w_i)$最大

- 如果使用动态规划求解，需要满足两个条件
  - 重复子问题
  - 最优子结构

最大概率分词：最大概率路径计算

- 重复子问题：
- 对于节点w_i和其可能存在的多个后继节点w_j和w_k
  - 任意通过w_i到达w_j的路径的权重=该路径通过w_i的路径权重+w_j的权重
  - 任意通过w_i到达w_k的路径的权重=该路径通过w_i的路径权重+w_k的权重
- 也就是对于拥有公共前驱节点w_i的节点w_j和w_k，需要重复计算达到w_i的路径的概率

最大概率分词：最大概率路径计算

- 最优子结构
- 对于整个句子的最优路径R_{max}和一个末端节点w_x，对于其可能存在的多个前驱$w_i, w_j, w_k, \dots$设到达w_i, w_j, w_k的最大路径分别是$R_{max_i}, R_{max_j}, R_{max_k}, \dots$有

$$R_{max} = \max\left(R_{max_i}, R_{max_j}, R_{max_k}, \dots\right) + \mathrm{weight}(w_x)$$

- 于是问题转化为求解$R_{max_i}, R_{max_j}, R_{max_k}, \dots$
- 组成了最优子结构，子结构里面的最优解是全局最优解的一部分

最大概率分词：最大概率路径计算

- 构建自底向上的动态规划过程，从从sentence的最后一个字（N-1）开始倒序遍历sentence的每个字（idx）的方式，计算子句sentence[idx ~ N-1]的概率对数得分。然后将概率对数得分最高的情况以（概率对数，词语最后一个位置）这样的元组保存在route中。
- 函数中，logtotal为构建前缀词频时所有的词频之和的对数值，这里的计算都是使用概率对数值，可以有效防止下溢问题。

最大概率分词：最大概率路径计算

- 代码实现如下：

```python
def calc(sentence, DAG, route, lfreq, ltotal):
    N = len(sentence)
    route[N] = (0, 0)
    logtotal = log(ltotal)
    for idx in range(N - 1, -1, -1):
        route[idx] = max((log(lfreq[sentence[idx:x + 1]] or 1) - logtotal + route[x + 1][0], x) for x in DAG[idx])
```

- 其结果输出为：

```
6:(0, 0)
5:(-4.477290894284794, 5)
4:(-7.530159813019676, 4)
3:(-7.394350100223548, 4)
2:(-11.423900230630986, 2)
1:(-9.672029455141175, 4)
0:(-10.770622835238974, 0)
```

- 由此可知，最终的结果输出为：去/北京大学/玩