隐马尔可夫模型

安装 Graphviz

1. 安装graphviz
    - 安装文件下载地址：https://graphviz.gitlab.io/_pages/Download/Download_windows.html (https://graphviz.gitlab.io/_pages/Download/Download_windows.html)
    - 安装过程中选择将graphviz加到PATH
2. 安装python插件graphviz： pip install graphviz
3. 安装python插件pydotplus：pip install pydotplus

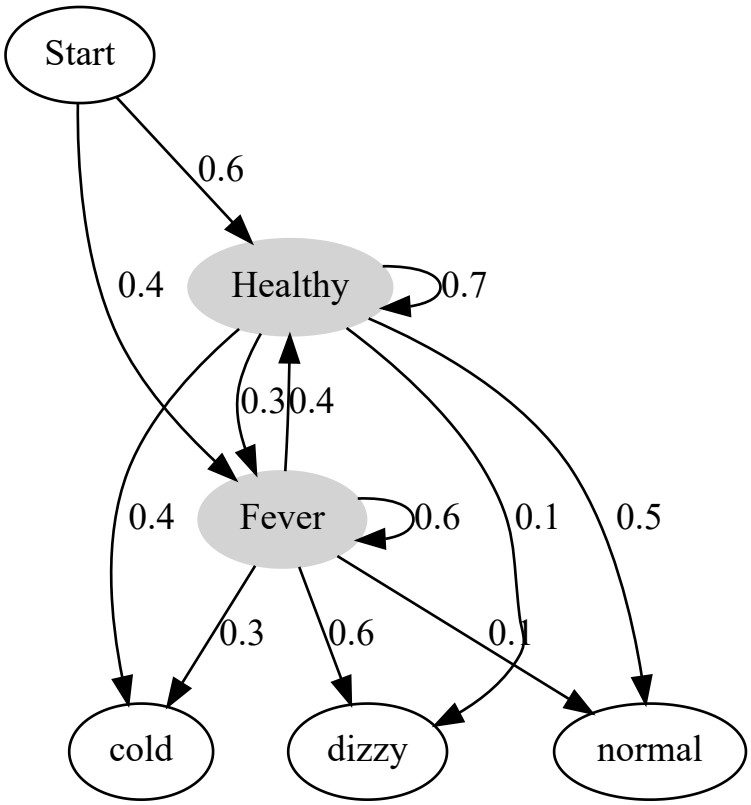
绘制流程图

In [230]:
```
import numpy as np
from collections import Counter
import logging
logging.basicConfig(level=logging.ERROR,format='%(asctime)s - [line:%(lineno)d] - %(leveln
```

In [231]:
```
states = ('Healthy', 'Fever')
start_probability = {'Healthy': 0.6, 'Fever': 0.4}
transition_probability = {
    'Healthy': {'Healthy': 0.7, 'Fever': 0.3},
    'Fever': {'Healthy': 0.4, 'Fever': 0.6},
}
emission_probability = {
    'Healthy': {'normal': 0.5, 'cold': 0.4, 'dizzy': 0.1},
    'Fever': {'normal': 0.1, 'cold': 0.3, 'dizzy': 0.6},
}
observations = ('normal', 'cold', 'dizzy')
```

In [232]:
```
from graphviz import *

g = Digraph()

g.node("Start")

for key in transition_probability:
    g.node(key, style='filled', color="lightgray")

for key in emission_probability['Healthy']:
    g.node(key)

for key in start_probability:
    g.edge("Start",key, label=str(start_probability[key]))

for key in transition_probability:
    for state in transition_probability[key]:
        g.edge(key, state, label=str(transition_probability[key][state]))

for key in emission_probability:
    for observation in emission_probability[key]:
        g.edge(key, observation, label=str(emission_probability[key][observation]))

g
```

Out[232]:



数据转换

In [233]:
```
def generate_index_map(lables):
    index_label = {}
    label_index = {}
    i = 0
    for l in lables:
        index_label[i] = l
        label_index[l] = i
        i += 1
    return label_index, index_label
```

In [234]:
```
states_label_index, states_index_label = generate_index_map(states)
observations_label_index, observations_index_label = generate_index_map(observations)
```

In [235]:
```
observations_index_label
```

Out[235]:
```
{0: 'normal', 1: 'cold', 2: 'dizzy'}
```

In [236]:
```
states_label_index
```

Out[236]:
```
{'Healthy': 0, 'Fever': 1}
```

In [237]:
```
states_index_label
```

Out[237]:
```
{0: 'Healthy', 1: 'Fever'}
```

In [238]:
```
def convert_observations_to_index(observations, label_index):
    list = []
    for o in observations:
        list.append(label_index[o])
    return list


def convert_map_to_vector(map, label_index):
    v = np.empty(len(map), dtype=float)
    for e in map:
        v[label_index[e]] = map[e]
    return v


def convert_map_to_matrix(map, label_index1, label_index2):
    m = np.empty((len(label_index1), len(label_index2)), dtype=float)
    for line in map:
        for col in map[line]:
            m[label_index1[line]][label_index2[col]] = map[line][col]
    return m
```

In [239]:
```
A = convert_map_to_matrix(transition_probability, states_label_index, states_label_index)
B = convert_map_to_matrix(emission_probability, states_label_index, observations_label_ind
observations_index = convert_observations_to_index(observations, observations_label_index)
pi = convert_map_to_vector(start_probability, states_label_index)
```

In [240]:
```
A
```

Out[240]:
```
array([[0.7, 0.3],
       [0.4, 0.6]])
```

In [241]:
```
B
```

Out[241]:
```
array([[0.5, 0.4, 0.1],
       [0.1, 0.3, 0.6]])
```

In [242]:
```
observations_index
```

Out[242]:
```
[0, 1, 2]
```

In [243]:
```
pi
```

Out[243]:
```
array([0.6, 0.4])
```

生成样本

根据初始状态概率向量采样第一个时刻的状态

In [244]:
```
def generate_initial_state(start_probability):
    rd = np.random.rand()
    if rd <= start_probability[states_index_label[0]]:
        return 0
    else:
        return 1

states_index_label[generate_initial_state(start_probability)]
```

Out[244]:
```
'Healthy'
```

In [245]:
```
# 测试
c = Counter()
for i in range(10000):
    c[states_index_label[generate_initial_state(start_probability)]] += 1

print(c)
```

```
Counter({'Healthy': 6034, 'Fever': 3966})
```

根据状态转移概率矩阵第i行的概率向量采样下一时刻的状态

In [246]:
```
def generate_transition_state(transition_probability, current_state):
    rd = np.random.rand()
    if rd <= transition_probability[current_state][states_index_label[0]]:
        return 0
    else:
        return 1

states_index_label[generate_transition_state(transition_probability, "Healthy")]
```

Out[246]: 'Healthy'

In [247]:
```
# 测试
c = Counter()
for i in range(10000):
    c[states_index_label[generate_transition_state(transition_probability, "Healthy")]] +=

print(c)

c = Counter()
for i in range(10000):
    c[states_index_label[generate_transition_state(transition_probability, "Fever")]] += 1

print(c)
```

```
Counter({'Healthy': 6932, 'Fever': 3068})
Counter({'Fever': 5967, 'Healthy': 4033})
```

根据发射概率矩阵采样观察

In [248]:
```
def generate_observation(emission_probability, current_state):
    rd = np.random.rand()
    value1 = emission_probability[current_state][observations_index_label[0]]
    value2 = emission_probability[current_state][observations_index_label[1]]
    if rd <= value1:
        return 0
    elif rd > value1 and rd <= value1 + value2:
        return 1
    else:
        return 2

observations_index_label[generate_observation(emission_probability, "Healthy")]
```

Out[248]: 'normal'

In [249]:
```
# 测试
c = Counter()
for i in range(100000):
    c[observations_index_label[generate_observation(emission_probability, "Fever")]] += 1

print(c)
```

```
Counter({'dizzy': 60094, 'cold': 29851, 'normal': 10055})
```

生成序列

In [250]:
```
def generate(length):
    hidden_states = []
    observations = []
    current_state = states_index_label[generate_initial_state(start_probability)]
    hidden_states.append(current_state)
    observations.append(observations_index_label[generate_observation(emission_probability
    for i in range(1, length):
        current_state = states_index_label[generate_transition_state(transition_probabilit
        hidden_states.append(current_state)
        observations.append(observations_index_label[generate_observation(emission_probabi
    return hidden_states,observations

states,observations = generate(5)
```

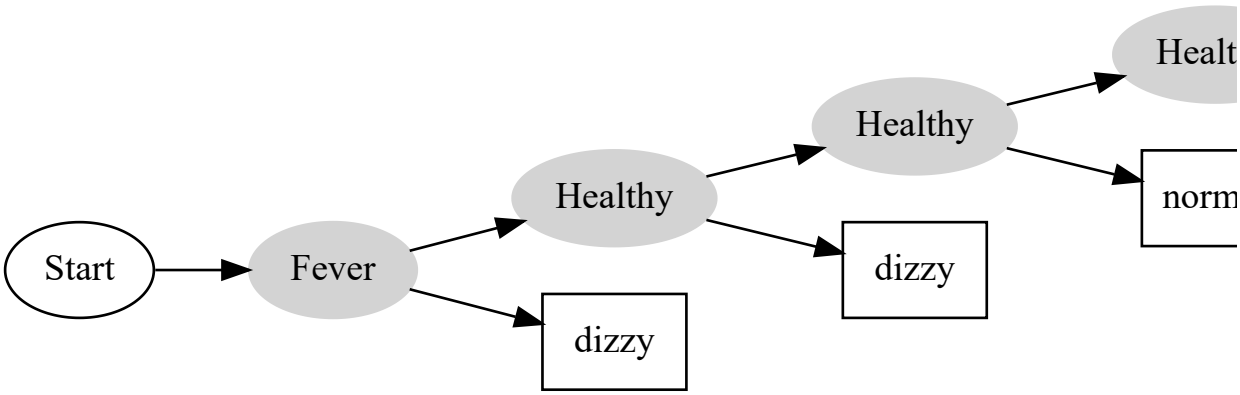
In [251]:
```
list(zip(states,observations))
```

Out[251]:
```
[('Fever', 'dizzy'),
 ('Healthy', 'dizzy'),
 ('Healthy', 'normal'),
 ('Healthy', 'normal'),
 ('Healthy', 'cold')]
```

绘制流程图

In [252]:
```
g = Digraph()
g.attr(rankdir='LR')

g.node("Start")
for i,sta in enumerate(states):
    g.node(str(i+1)+ "_" + sta, color='lightgray', label=sta, style='filled')

for i,obs in enumerate(observations):
    g.node(str(i+1)+ "_" + obs, color='black',shape="box", label=obs)

g.edge("Start",str(1) + "_" + states[0])

for i in range(len(states)-1):
    g.edge(str(i+1) + "_" + states[i],str(i+2) + "_" + states[i+1])

for i in range(len(observations)):
    g.edge(str(i+1) + "_" + states[i],str(i+1) + "_" + observations[i])
g
```

Out[252]:



viterbi 算法

In [253]:
```
def viterbi(trainsition_probability,emission_probability,pi,obs_seq):

    # 隐藏状态个数
    N = np.array(trainsition_probability).shape[0]

    # 观测序列的观测个数，即时刻个数
    T = len(obs_seq)

    # 每个时刻每个状态对应的局部最优状态序列的概率值
    delta = np.zeros((N,T))

    # 保存每个时刻每个状态取到最大概率的前置节点
    phi = np.zeros((N,T),dtype = int)

    #初始状态
    delta[:,0] = pi*np.transpose(emission_probability[:,obs_seq[0]])
    print("delta[:,0]",delta[:,0])

    for t in range(1,T):
        list_max=[]
        for n in range(N):
            # 计算时刻t,状态为n的所有单个路径的概率值
            delta_ti = delta[:,t-1]*np.transpose(trainsition_probability[:,n])
            print("delta_ti",delta_ti)
            # 保存最大概率
            list_max.append(np.max(delta_ti))
            print("list_max",list_max)
            # 保存最大概率对应的前置节点
            phi[n,t] = np.argmax(delta_ti)
            print("phi[%d, %d]"%(n,t), phi[n,t])
        delta[:,t] = np.array(list_max)*np.transpose(emission_probability[:,obs_seq[t]])
        print("delta[:,%d]"%t,delta[:,t])

    result = np.zeros(T)
    result[T-1] = np.argmax(delta[:,T-1])
    print("result[%d]"%(T-1), result[T-1])
    for t in range(T-2,-1,-1):
        result[t] = phi[int(result[t+1]),t+1]
        print("result[%d]"%t, result[t])
    return result
```

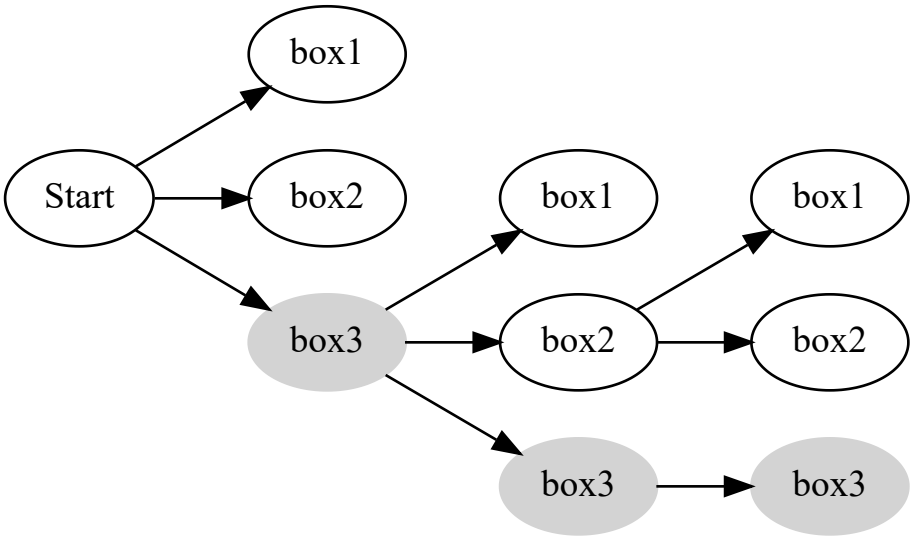
In [254]:
```
def viterbi_with_flowchart(trainsition_probability,emission_probability,pi,states, observa

    g = Digraph()
    g.attr(rankdir='LR')

    # 隐藏状态个数
    N = np.array(trainsition_probability).shape[0]

    # 观测序列的观测个数，即时刻个数
    T = len(obs_seq)

    # 每个时刻每个状态对应的局部最优状态序列的概率值
    delta = np.zeros((N,T))

    # 保存每个时刻每个状态取到最大概率的前置节点
    phi = np.zeros((N,T),dtype = int)

    #初始状态
    delta[:,0] = pi*np.transpose(emission_probability[:,obs_seq[0]])
    logging.debug("delta[:,0]: " + str(delta[:,0]))

    for index,state in enumerate(states):
        g.node(str(0) + "_" + state, label=state)
        g.edge("Start", str(0) + "_" + state)

    for t in range(1,T):
        logging.debug("\t 第一层for循环：t = %d"%t)
        for index,state in enumerate(states):
            g.node(str(t) + "_" + state, label=state)
        list_max=[]
        for n in range(N):
            logging.debug("\t\t 第二层for循环：n = %d"%n)
            # 计算时刻t,状态为n的所有单个路径的概率值
            delta_ti = delta[:,t-1]*np.transpose(trainsition_probability[:,n])
            logging.debug("\t\tdelta_ti: "+ str(delta_ti))
            # 保存最大概率
            list_max.append(np.max(delta_ti))
            logging.debug("\t\tlist_max: "+ str(list_max))
            # 保存最大概率对应的前置节点
            phi[n,t] = np.argmax(delta_ti)
            logging.debug("\t\tphi[%d, %d]: "%(n,t)+str(phi[n,t]))
            g.edge(str(t-1)+"_"+states[phi[n,t]], str(t) + "_" + states[n])
        delta[:,t] = np.array(list_max)*np.transpose(emission_probability[:,obs_seq[t]])
        logging.debug("\tdelta[:,%d]: "%t + str(delta[:,t]))

    result = np.zeros(T)
    result[T-1] = np.argmax(delta[:,T-1])
    g.node(str(T-1) + "_" + states[int(result[T-1])], label=states[int(result[T-1])], sty
    logging.debug("result[%d]: "%(T-1)+str(result[T-1]))
    for t in range(T-2,-1,-1):
        result[t] = phi[int(result[t+1]),t+1]
        logging.debug("result[%d]: "%t+str(result[t]))
        g.node(str(t) + "_" + states[int(result[t])], label=states[int(result[t])], style=
    return result,g
```

案例一

In [255]:
```
states = ("box1","box2","box3")
observations = ('red','white')
trainsion_probility = np.array([[0.5, 0.2, 0.3],
                                [0.3, 0.5, 0.2],
                                [0.2, 0.3, 0.5]])
emission_probility = np.array([[0.5, 0.5],
                               [0.4, 0.6],
                               [0.7, 0.3]])
pi = np.array([0.2,0.4,0.4])
obs_seq = np.array([0,1,0])

res,g = viterbi_with_flowchart(trainsion_probility, emission_probility, pi, states, observ
print(res)
print("观测序列：", ", ".join(map(lambda x: observations[int(x)], obs_seq)))
print("隐藏序列：", ", ".join(map(lambda x: states[int(x)], res)))
g
```

```
[2. 2. 2.]
观测序列： red, white, red
隐藏序列： box3, box3, box3
```

Out[255]:



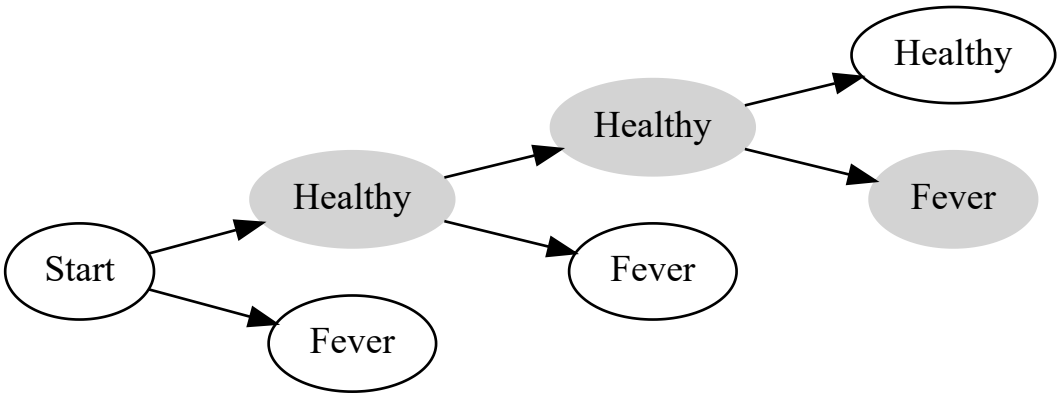
案例2

In [256]:
```
states = ('Healthy', 'Fever')
start_probability = {'Healthy': 0.6, 'Fever': 0.4}
transition_probability = {
    'Healthy': {'Healthy': 0.7, 'Fever': 0.3},
    'Fever': {'Healthy': 0.4, 'Fever': 0.6},
}
emission_probability = {
    'Healthy': {'normal': 0.5, 'cold': 0.4, 'dizzy': 0.1},
    'Fever': {'normal': 0.1, 'cold': 0.3, 'dizzy': 0.6},
}
observations = ('normal', 'cold', 'dizzy')

states_label_index, states_index_label = generate_index_map(states)
observations_label_index, observations_index_label = generate_index_map(observations)

A = convert_map_to_matrix(transition_probability, states_label_index, states_label_index)
B = convert_map_to_matrix(emission_probability, states_label_index, observations_label_ind
observations_index = convert_observations_to_index(observations, observations_label_index)
pi = convert_map_to_vector(start_probability, states_label_index)
```

In [257]:
```
obs_seq = np.array([0,1,2])
#res = viterbi(A, B, pi, obs_seq)
res,g = viterbi_with_flowchart(A, B, pi, states, observations, obs_seq)
print("观测序列：", ", ".join(map(lambda x: observations[int(x)], obs_seq)))
print("隐藏序列：", ", ".join(map(lambda x: states[int(x)], res)))
g
```

观测序列： normal, cold, dizzy
隐藏序列： Healthy, Healthy, Fever

Out[257]:



结巴分词

In [259]:
```
# 概率值都是取对数之后的结果
from prob_start import P as start_p # 状态初始概率
from prob_trans import P as trans_p # 状态转移概率
from prob_emit import P as emit_p   # 状态发射概率

MIN_FLOAT = -3.14e100

PrevStatus = {
    'B': 'ES',
    'M': 'MB',
    'S': 'SE',
    'E': 'BM'
}
```

In [289]:
```
def viterbi(obs, states, start_p, trans_p, emit_p):
    V = [{}]  # tabular
    path = {}
    for y in states:  # init
        V[0][y] = start_p[y] + emit_p[y].get(obs[0], MIN_FLOAT)
        path[y] = [y]
    #print(path)
    for t in range(1, len(obs)):
        V.append({})
        newpath = {}
        for y in states:
            em_p = emit_p[y].get(obs[t], MIN_FLOAT)
            (prob, state) = max(
                [(V[t - 1][y0] + trans_p[y0].get(y, MIN_FLOAT) + em_p, y0) for y0 in Prev
            V[t][y] = prob
            newpath[y] = path[state] + [y]
        path = newpath
        #print(path)

    (prob, state) = max((V[len(obs) - 1][y], y) for y in 'ES')

    return (prob, path[state])

sentence = "商品和服务"
print(viterbi(sentence, "BMES", start_p, trans_p, emit_p))
```

```
(-30.294867244593323, ['B', 'E', 'S', 'B', 'E'])
```

In [290]:
```
def __cut(sentence):
    global emit_P
    prob, pos_list = viterbi(sentence, 'BMES', start_p, trans_p, emit_p)
    begin, nexti = 0, 0
    # print pos_list, sentence
    for i, char in enumerate(sentence):
        pos = pos_list[i]
        if pos == 'B':
            begin = i
        elif pos == 'E':
            yield sentence[begin:i + 1]
            nexti = i + 1
        elif pos == 'S':
            yield char
            nexti = i + 1
    if nexti < len(sentence):
        yield sentence[nexti:]
print(list(__cut(sentence)))
```

['商品', '和', '服务']

练习

利用MSR语料计算初始状态概率向量π,状态转移概率矩阵A和发射概率矩阵B,构建分词器并进行性能评测。

In [296]:
```
import os,sys
import re
from math import log
sighan05 = "../第二课/第二届国际中文分词评测/icwb2-data/"
msr_dict = os.path.join(sighan05, 'gold', 'msr_training_words.utf8')
msr_test = os.path.join(sighan05, 'testing', 'msr_test.utf8')
msr_output = os.path.join(sighan05, 'testing', 'msr_output.txt')
msr_gold = os.path.join(sighan05, 'gold', 'msr_test_gold.utf8')
train=open(sighan05+'training/msr_training.txt')
```

In [272]:

```
def translate_dot(dic):           #计算概率并取log
    s=sum(dic.values())
    for i in dic.keys():
        if dic[i]==0:
            continue
        dic[i]=log(dic[i]/s)


def make_word_count(y,ci):        #将词插入状态字典
    if ci in emit_p[y].keys():
        emit_p[y][ci]+=1
    else:
        emit_p[y][ci]=1


def make_train_count(line):
    line=line.lstrip()            #去除字符串最开始的空格
    i=0
    y=''
    while(i<len(re.sub("\\s+", " ", line))):
        if re.sub("\\s+", " ", line)[i]==' ':      #跳过中间空格
            i=i+1
            continue
        if i==0:                                   #判断第一个字符的状态
            if re.sub("\\s+", " ", line)[i+1]==' ':
                start_p['S']+=1
                y=y+'S'
                make_word_count('S',re.sub("\\s+", " ", line)[i])
            else:
                start_p['B']+=1
                y=y+'B'
                make_word_count('B',re.sub("\\s+", " ", line)[i])
        else:                                      #判断后续字符的状态
            if re.sub("\\s+", " ", line)[i-1]==' ':
                if re.sub("\\s+", " ", line)[i+1]==' ':
                    trans_p[y[-1]]['S']+=1
                    y=y+'S'
                    make_word_count('S',re.sub("\\s+", " ", line)[i])
                else:
                    trans_p[y[-1]]['B']+=1
                    y=y+'B'
                    make_word_count('B',re.sub("\\s+", " ", line)[i])
            else:
                if re.sub("\\s+", " ", line)[i+1]==' ':
                    trans_p[y[-1]]['E']+=1
                    y=y+'E'
                    make_word_count('E',re.sub("\\s+", " ", line)[i])
                else:
                    trans_p[y[-1]]['M']+=1
                    y=y+'M'
                    make_word_count('M',re.sub("\\s+", " ", line)[i])
        i+=1
```

In [273]:
```
start_p={'B':0,'S':0,'E':0,'M':0}
trans_p={'B': {'E': 0, 'M': 0},
 'E': {'B': 0, 'S': 0},
 'M': {'E': 0, 'M': 0},
 'S': {'B': 0, 'S': 0}}
emit_p={'B':{},'S':{},'E':{},'M':{}}
for line in train:
    make_train_count(line)

for i in trans_p.keys():
    translate_dot(trans_p[i])
for i in emit_p.keys():
    translate_dot(emit_p[i])
translate_dot(start_p)
```

In [291]:
```
# 测试例子
test_cases = ['项目的研究',
              '商品和服务',
              '研究生命起源',
              '当下雨天地面积水',
              '结婚的和尚未结婚的',
              '欢迎新老师生前来就餐']
for sentence in test_cases:
    print(list(__cut(sentence)))
```

```
['项目', '的', '研究']
['商品', '和', '服务']
['研究', '生命', '起源']
['当下', '雨天', '地面', '积水']
['结婚', '的', '和', '尚', '未', '结婚', '的']
['欢迎', '新', '老师', '生前', '来', '就餐']
```

In [292]:
```
#速度测评
import time
pressure = 10000
sentence='项目的研究'
def evaluate_speed(text):
    start_time = time.time()
    for i in range(pressure):
        __cut(sentence)
    elapsed_time = time.time() - start_time
    seg_speed = len(text) * pressure / 10000 / elapsed_time
    print('%.2f 万字/秒' % (seg_speed))
    return seg_speed
evaluate_speed(sentence)
```

```
2453.96 万字/秒
```

Out[292]:
```
2453.957406974023
```

In [293]:

```
import re

def to_region(segmentation: str) -> list:
    """
    将分词结果转换为区间
    :param segmentation: 商品 和 服务
    :return: [(0, 2), (2, 3), (3, 5)]
    """
    region = []
    start = 0
    for word in re.compile("\\s+").split(segmentation.strip()):
        end = start + len(word)
        region.append((start, end))
        start = end
    return region


def prf(gold: str, pred: str, dic) -> tuple:
    """
    计算P、R、F1
    :param gold: 标准答案文件，比如“商品 和 服务”
    :param pred: 分词结果文件，比如“商品 和服 务”
    :param dic: 词典
    :return: (P, R, F1, OOV_R, IV_R)
    """
    A_size, B_size, A_cap_B_size, OOV, IV, OOV_R, IV_R = 0, 0, 0, 0, 0, 0, 0
    with open(gold,encoding="utf-8") as gd, open(pred,encoding="utf-8") as pd:
        for g, p in zip(gd, pd):
            A, B = set(to_region(g)), set(to_region(p))
            A_size += len(A)
            B_size += len(B)
            A_cap_B_size += len(A & B)
            text = re.sub("\\s+", "", g)
            for (start, end) in A:
                word = text[start: end]
                if word in dic:
                    IV += 1
                else:
                    OOV += 1

            for (start, end) in A & B:
                word = text[start: end]
                if word in dic:
                    IV_R += 1
                else:
                    OOV_R += 1
    p, r = A_cap_B_size / B_size * 100, A_cap_B_size / A_size * 100
    return p, r, 2 * p * r / (p + r), OOV_R / OOV * 100, IV_R / IV * 100
```

In [297]:
```
import collections
msr = os.path.join(sighan05, 'training', 'msr_training.utf8')

f = collections.Counter()
with open(msr,encoding="utf-8") as src:
    total=0
    for line in src:
        line = line.strip()
        for word in line.split('  '):
            # word = word.strip()
            # if len(word) < 2: continue
            f[word] += 1
            total+=1
```

In [298]:
```
with open(msr_gold,encoding="utf-8") as test, open(msr_output, 'w', encoding="utf-8") as
    for line in test:
        output.write("  ".join(list(__cut(re.sub("\\s+", "", line)))))
        output.write("\n")
print("P:%.2f R:%.2f F1:%.2f OOV-R:%.2f IV-R:%.2f" % prf(msr_gold, msr_output, f))
```

```
P:77.86 R:79.87 F1:78.85 OOV-R:36.90 IV-R:81.04
```

In [ ]:
```

```