自然言語処理プログラミング勉強会 4 - 単語分割

Graham Neubig
奈良先端科学技術大学院大学 (NAIST)

単語分割とは

- 日本語や中国語、タイ語などは英語と違って単語の間に空白を使わない

単語分割を行う

- 単語分割は単語の間に明示的な区切りを入れる

単語　分割　を　行　う

必要なプログラミング技術：部分文字列

- 文字列の一部からなる部分文字列を作る方法

```
my_str = "hello world"

# Print the first 5 letters
print my_str[:5]
# Print all letters from position 6
print my_str[6:]
# Print all letters from position 3 until 7
print my_str[3:8]
```

```
$ ./my-program.py
hello
world
lo wo
```

Unicode 文字列の扱い（文字化けを防ぐ）

- unicode() と encode() 関数で UTF-8 を扱う

```
input_file = open("test_file.txt", "r")
for my_str in input_file:
    my_utf_str = unicode( my_str, "utf-8" )

    # Handle the string as a byte string
    print "str: %s %s" % (my_str[0:2], my_str[2:4])

    # Handle the string as a unicode string
    print "utf_str: %s %s" % (my_utf_str[0:2].encode("utf-8"),
                              my_utf_str[2:4].encode("utf-8"))
```

```
$ cat test_file.txt
単語分割
$ ./my-program.py
str: ?? ?
utf_str:  単語 分割
```

単語分割は難しい！

- 各文に対する多くの解釈はあるが、正解はただ 1 つ

- 正しい仮説をどうやって選ぶか？

1 つの解決策：言語モデルの利用

- 最も確率の高い仮説を選ぶ

P(農産　物　価格　安定　法)= $4.12*10^{-23}$

P(農産　物価　格安　定法) = $3.53*10^{-24}$

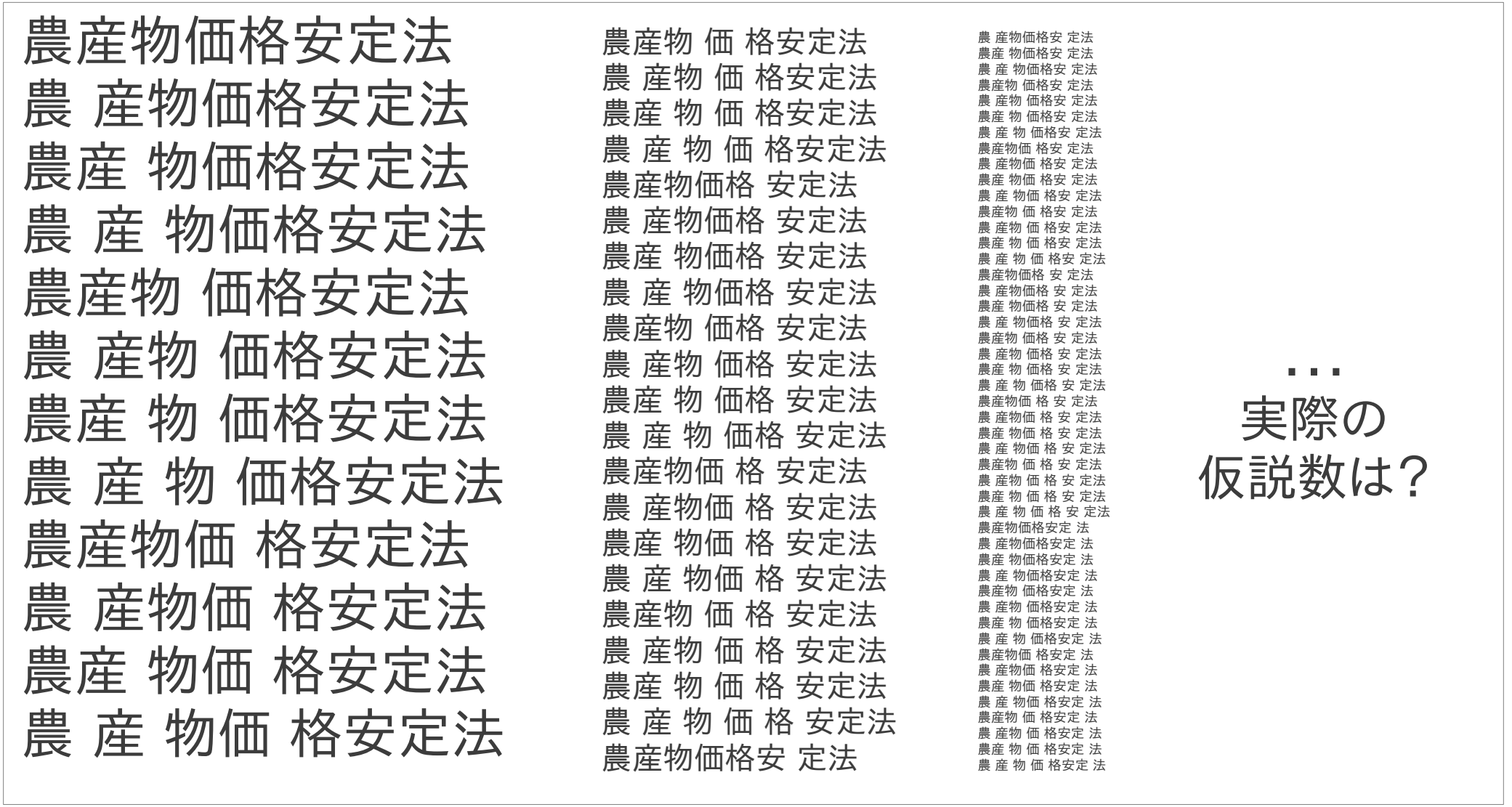
P(農産　物　価　格安　定法)= $6.53*10^{-25}$

P(農産　物　価格　安　定法)= $6.53*10^{-27}$

…

- ここで 1-gram 言語モデルを利用

問題：膨大な仮説空間



- 最も確率の高い仮説をどうやって効率良く見つけるか

俺に任せろ！

Andrew Viterbi

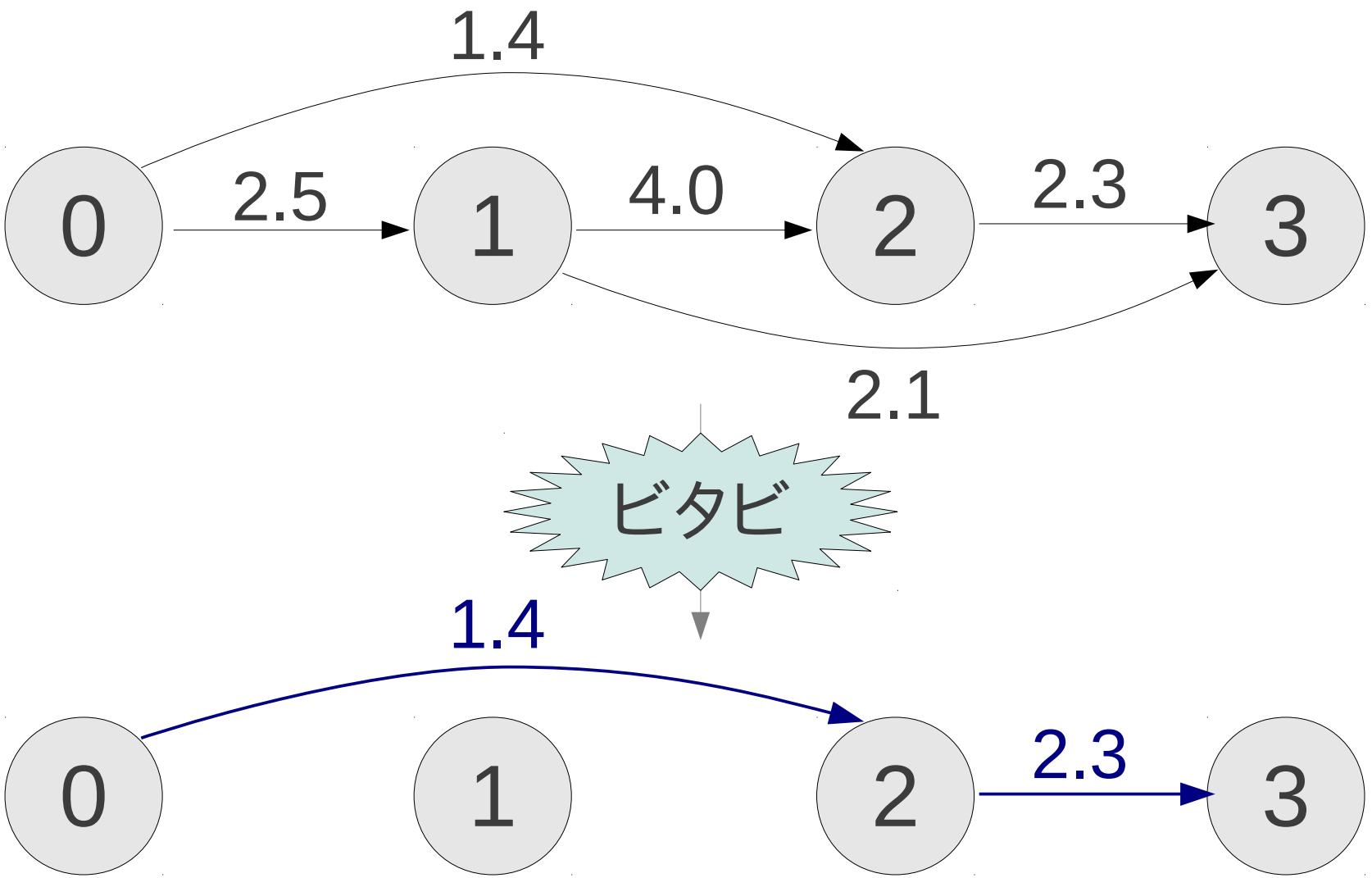
アンドリュー・ビタビ

(カリフォルニア大学 LA 校教授→ Qualcomm 代表取締役)

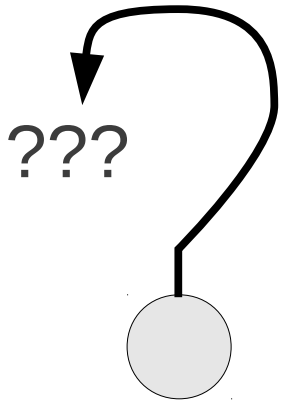
ビタビアルゴリズム

ビタビアルゴリズム

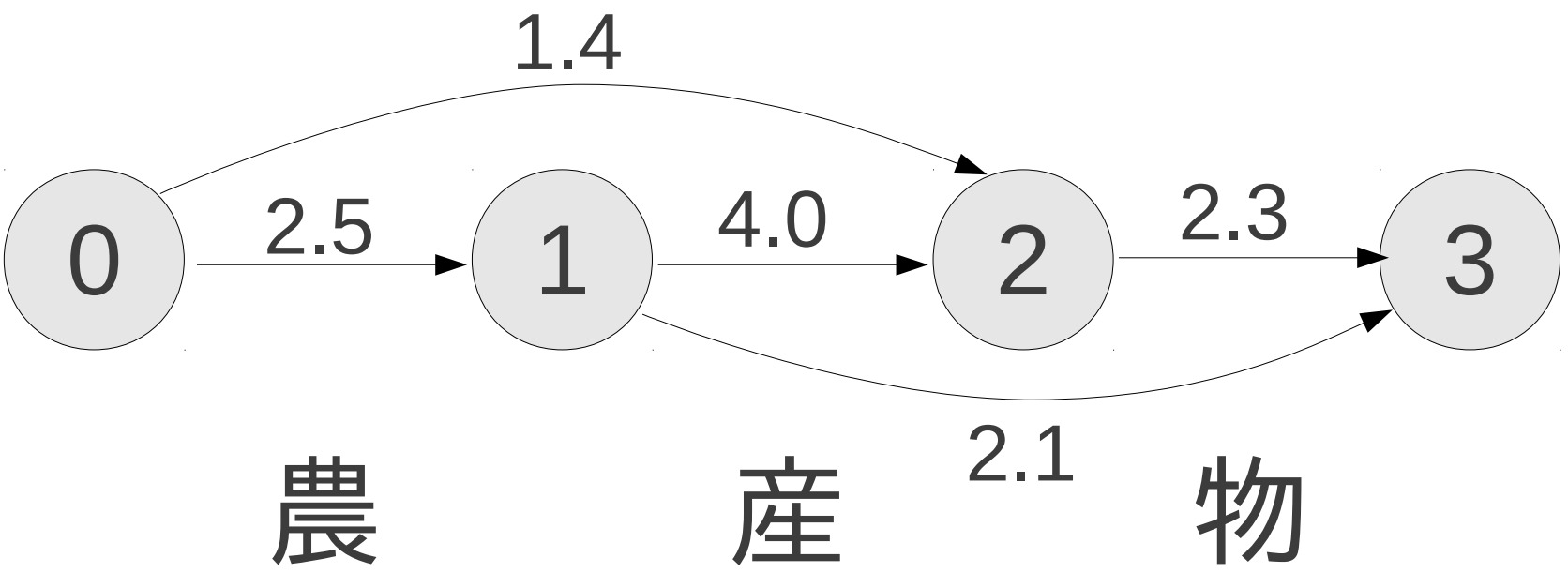
- グラフの最短経路を見つけるアルゴリズム



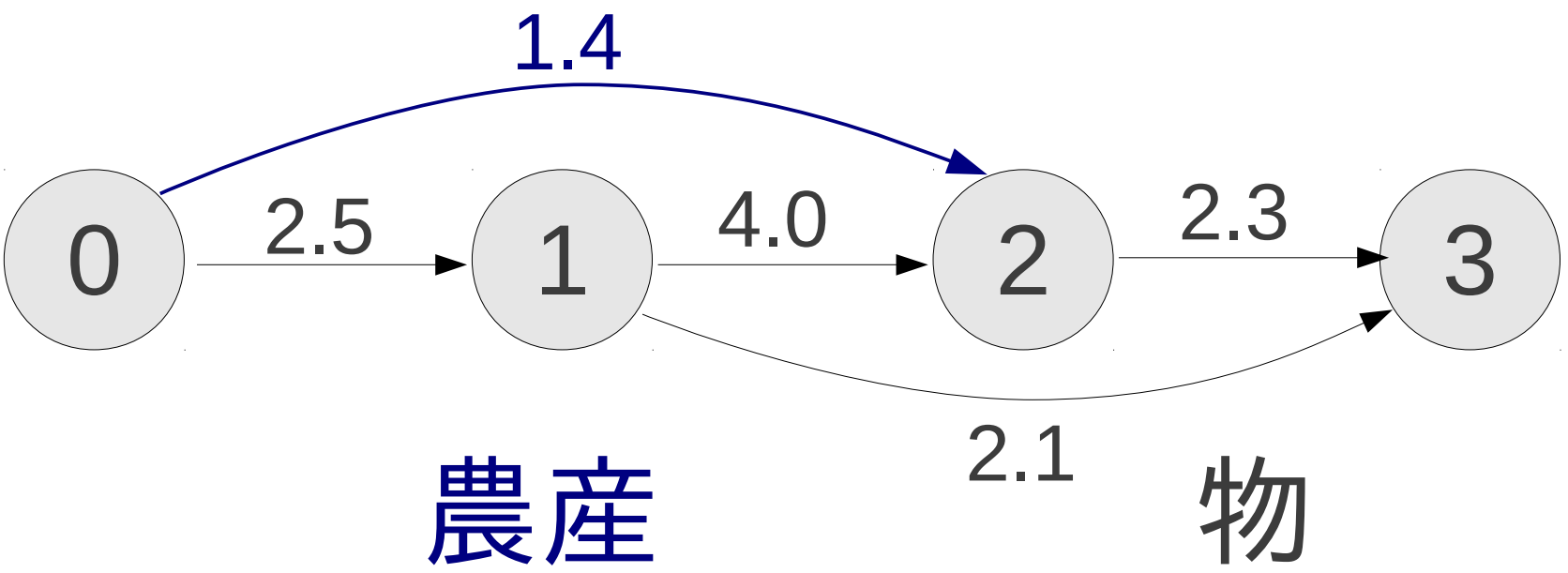
グラフ？
単語分割の話じゃなかったっけ？



グラフとしての単語分割

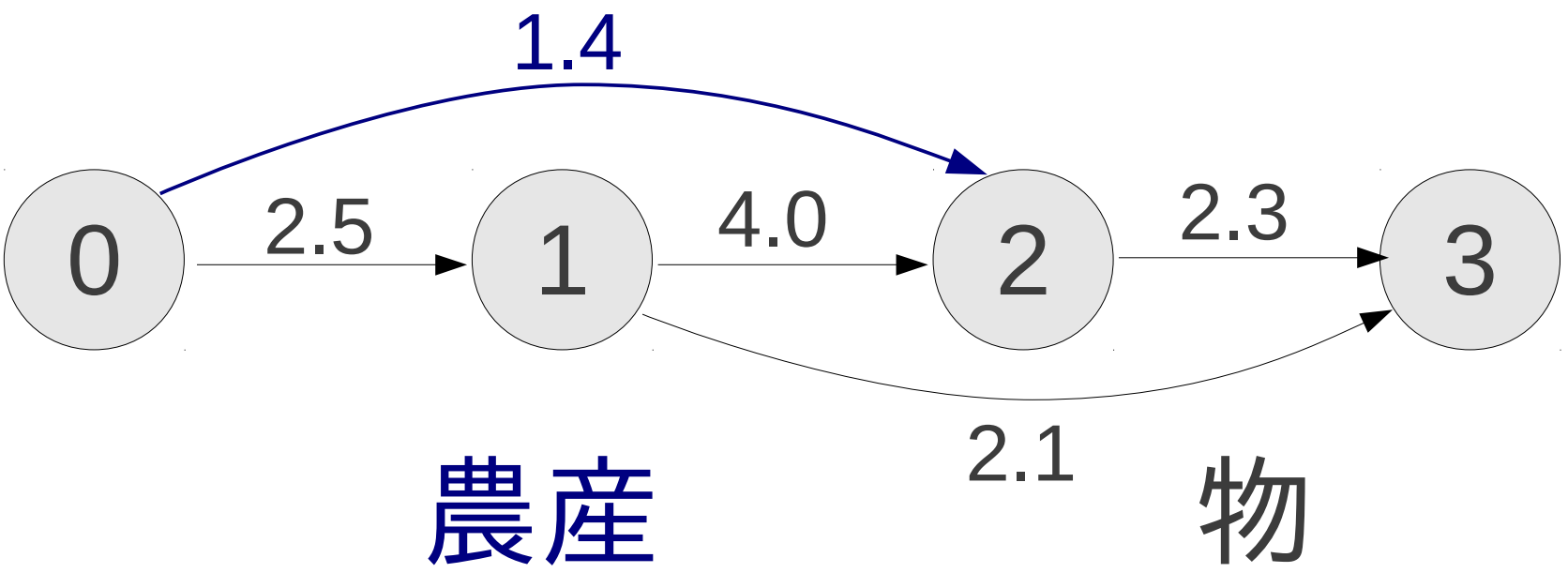


グラフとしての単語分割



- 各エッジは単語を表す

グラフとしての単語分割

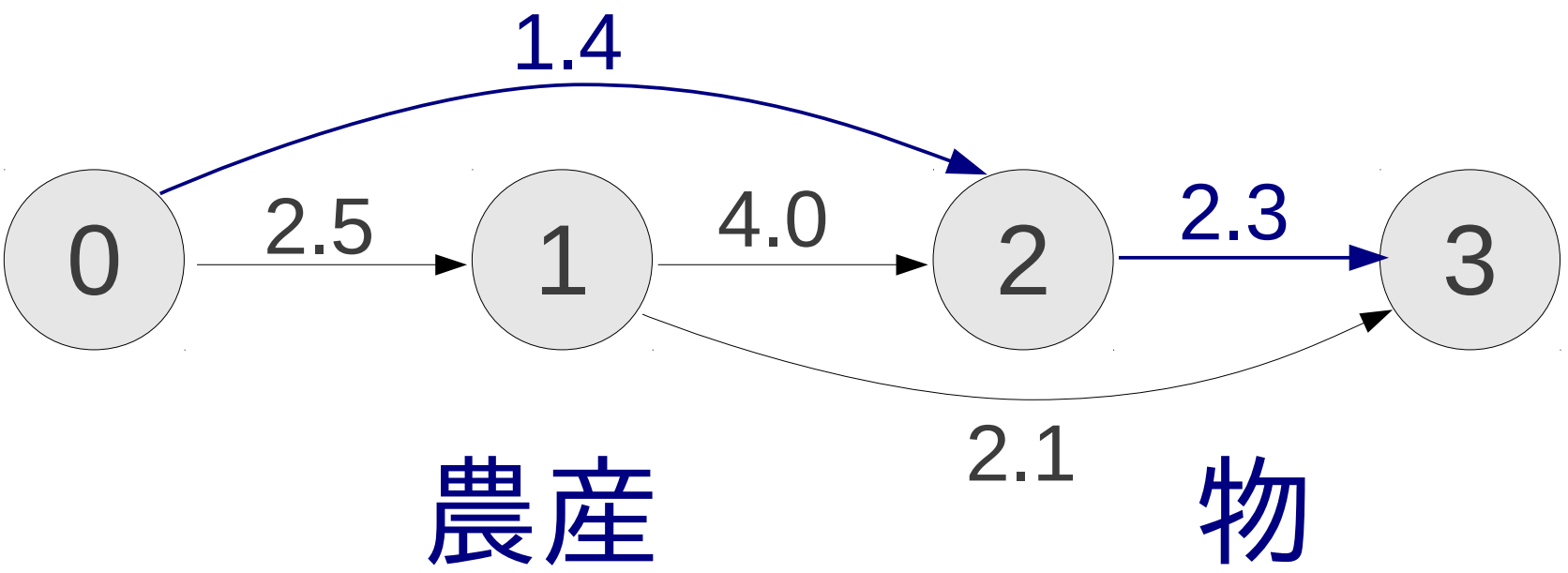


- 各エッジは単語を表す
- エッジの重みは負の対数確率

$$-\log(P(\text{農産})) = 1.4$$

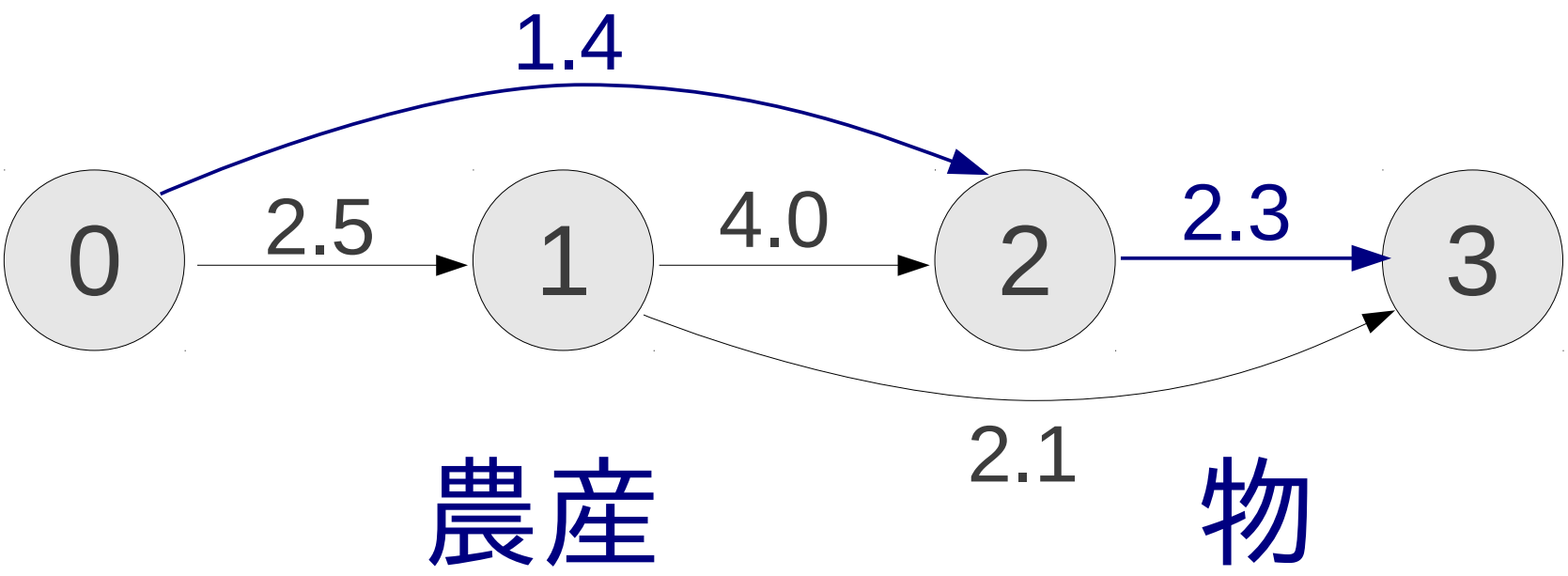
  - なぜ負？ (ヒント：最短経路)

グラフとしての単語分割



- グラフの経路は文の分割候補を表す

グラフとしての単語分割

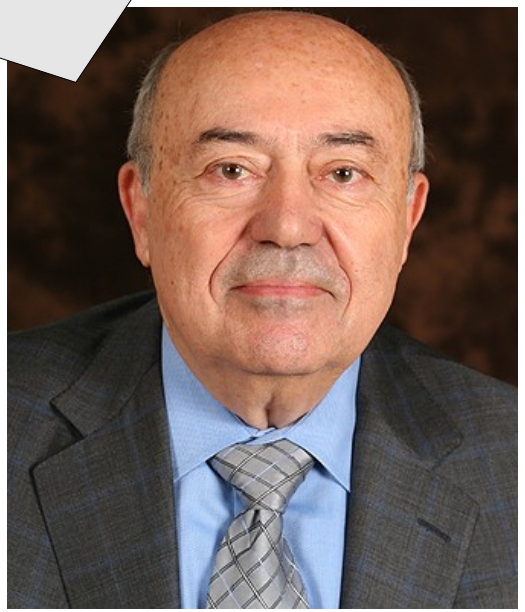


- グラフの経路は文の分割候補を表す
- 経路の重みは文の 1-gram 負対数確率

$$-\log(P(\text{農産})) + -\log(P(\text{物})) = 1.4 + 2.3 = 3.7$$

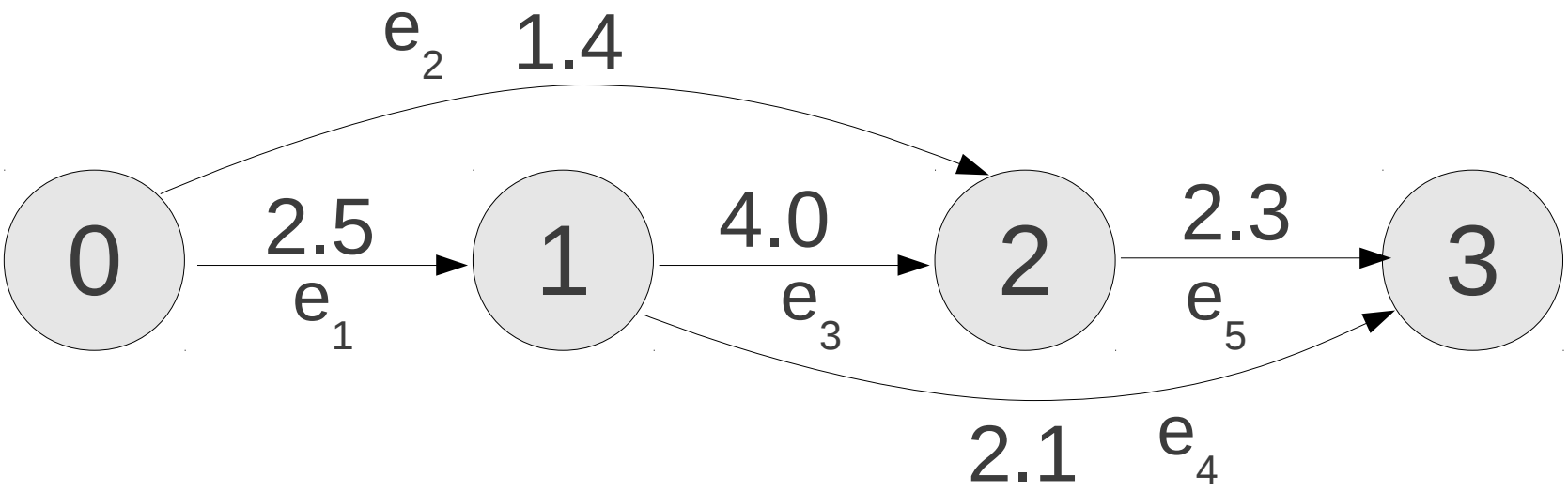
ビタビ先生、もっと教えて！

- ビタビアルゴリズムは２つのステップからなる
  - 前向きステップで、各ノードへたどる最短経路の長さを計算
  - 後ろ向きステップで、最短経路自体を構築



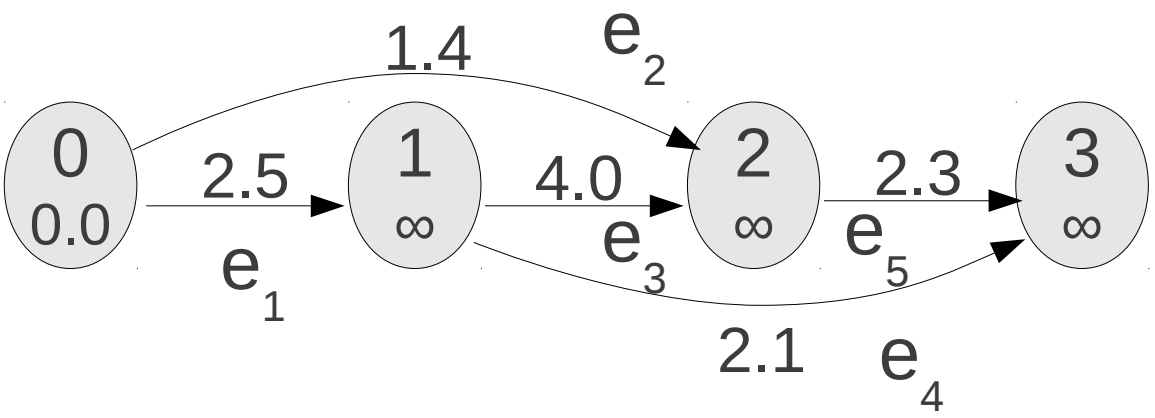
前向きステップ

前向きステップ



best_score[0] = 0
for each *node* in the *graph* (昇順)
    best_score[*node*] = ∞
    for each incoming *edge* of *node*
        score = *best_score*[*edge.prev_node*] + *edge.score*
        if *score* < *best_score*[*node*]
            best_score[*node*] = *score*
            best_edge[*node*] = *edge*

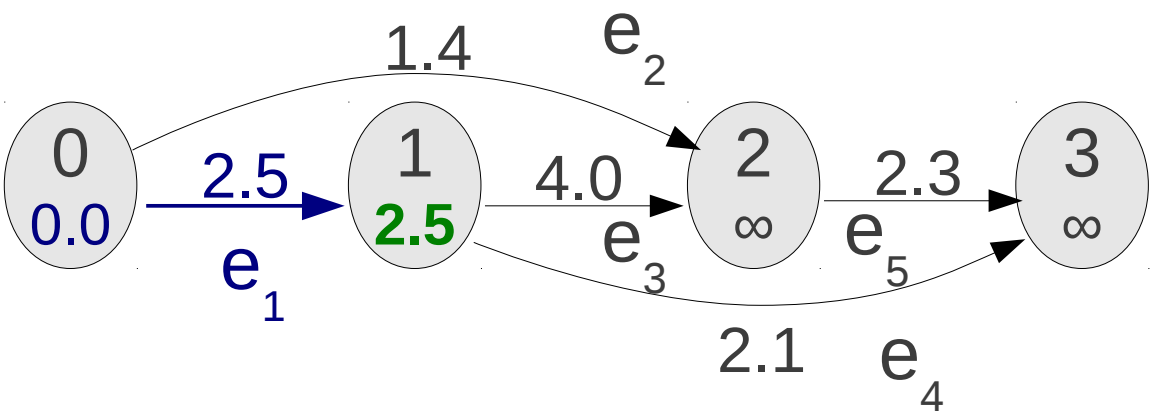
例 :



初期化 :

best_score[0] = 0

例 :



初期化 :

best_score[0] = 0

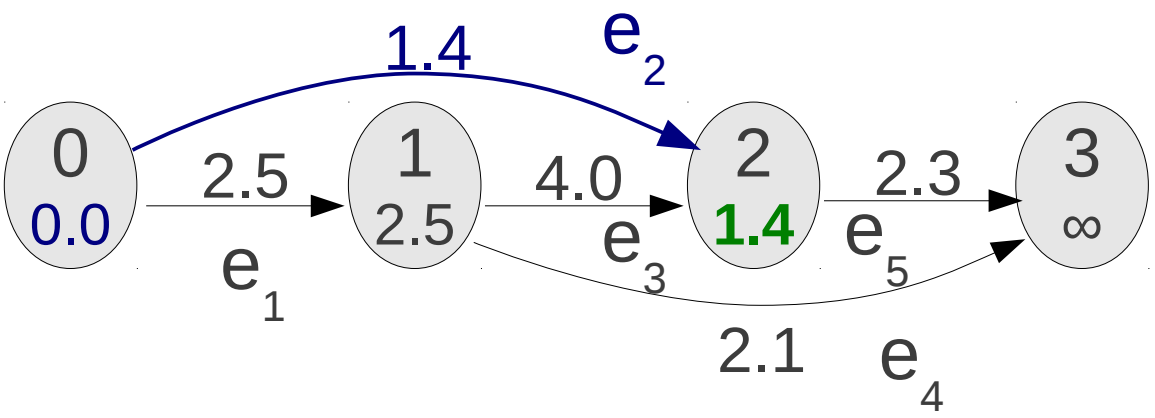
e_1 を計算 :

score = 0 + 2.5 = 2.5 (< ∞)

best_score[1] = 2.5

best_edge[1] = e_1

例 :



初期化 :

best_score[0] = 0

e_1 を計算 :

score = 0 + 2.5 = 2.5 (< ∞)

best_score[1] = 2.5

best_edge[1] = e_1

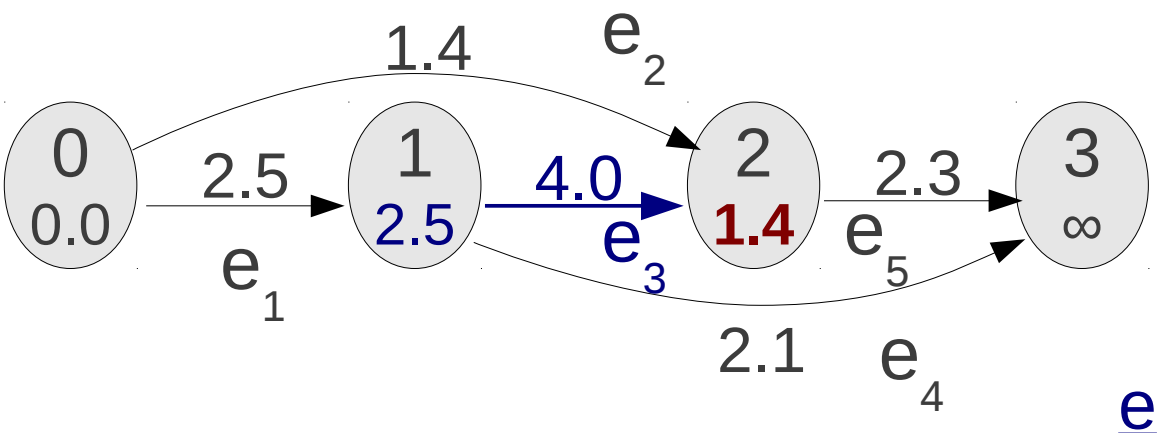
e_2 を計算 :

score = 0 + 1.4 = 1.4 (< ∞)

best_score[2] = 1.4

best_edge[2] = e_2

例 :



初期化 :

best_score[0] = 0

e_1 を計算 :

score = 0 + 2.5 = 2.5 (< ∞)

best_score[1] = 2.5

best_edge[1] = e_1

e_2 を計算 :

score = 0 + 1.4 = 1.4 (< ∞)

best_score[2] = 1.4

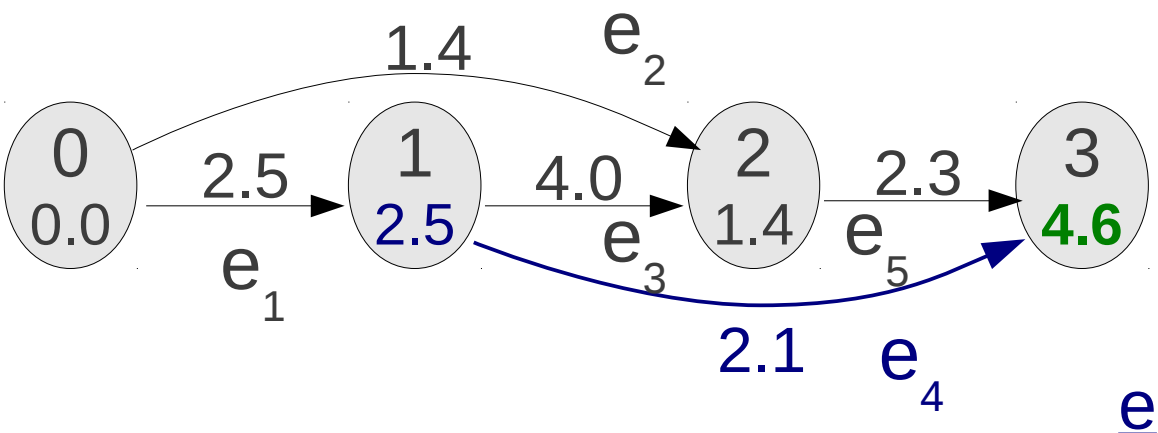
best_edge[2] = e_2

e_3 を計算 :

score = 2.5 + 4.0 = 6.5 (> 1.4)

変更なし !

例 :



初期化 :

best_score[0] = 0

e_1 を計算 :

score = 0 + 2.5 = 2.5 (< ∞)

best_score[1] = 2.5

best_edge[1] = e_1

e_2 を計算 :

score = 0 + 1.4 = 1.4 (< ∞)

best_score[2] = 1.4

best_edge[2] = e_2

e_3 を計算 :

score = 2.5 + 4.0 = 6.5 (> 1.4)

変更なし !

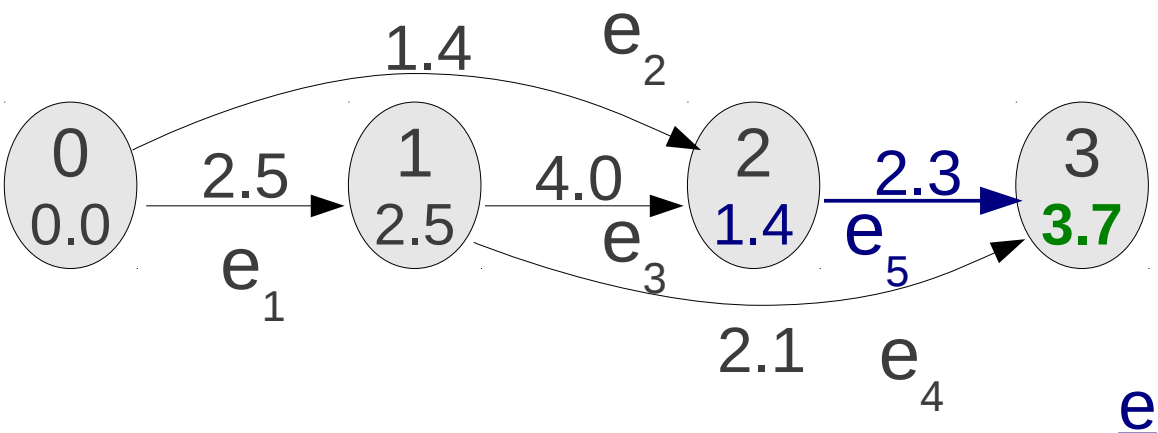
e_4 を計算 :

score = 2.5 + 2.1 = 4.6 (< ∞)

best_score[3] = 4.6

best_edge[3] = e_4

例 :



初期化 :
best_score[0] = 0

e_1 を計算 :
score = 0 + 2.5 = 2.5 (< ∞)
best_score[1] = 2.5
best_edge[1] = e_1

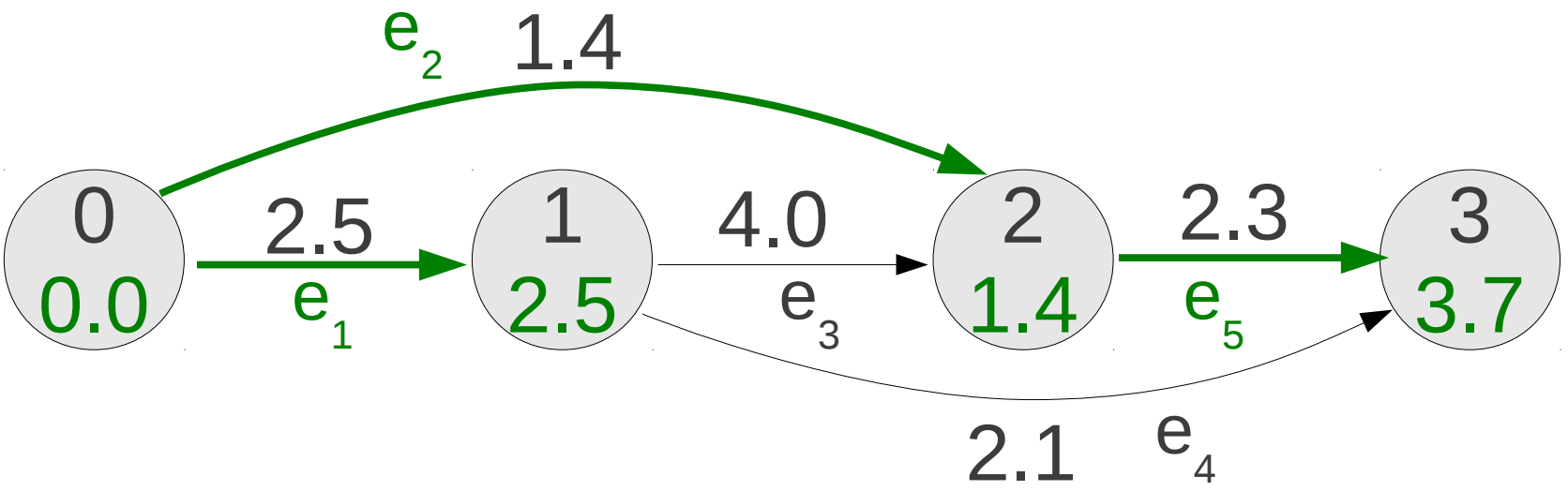
e_2 を計算 :
score = 0 + 1.4 = 1.4 (< ∞)
best_score[2] = 1.4
best_edge[2] = e_2

e_3 を計算 :
score = 2.5 + 4.0 = 6.5 (> 1.4)
変更なし !

e_4 を計算 :
score = 2.5 + 2.1 = 4.6 (< ∞)
~~best_score[3] = 4.6~~
~~best_edge[3] = e_4~~

e_5 を計算 :
score = 1.4 + 2.3 = 3.7 (< 4.6)
best_score[3] = 3.7
best_edge[3] = e_5

前向きステップの結果：

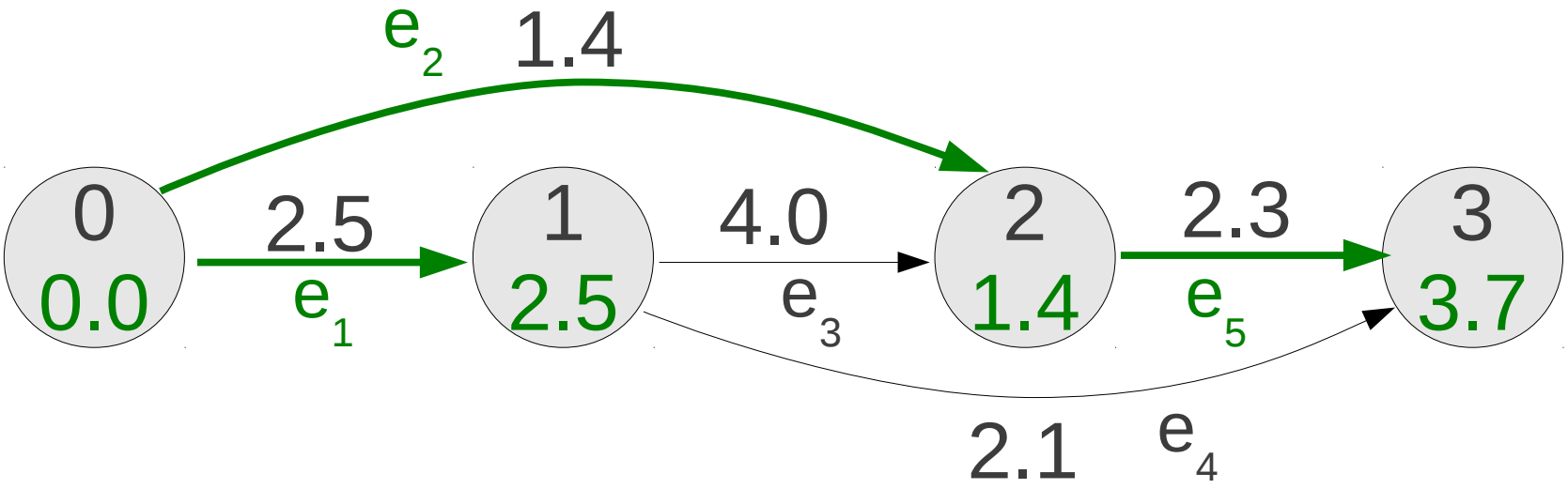


best_score = (0.0, 2.5, 1.4, 3.7)

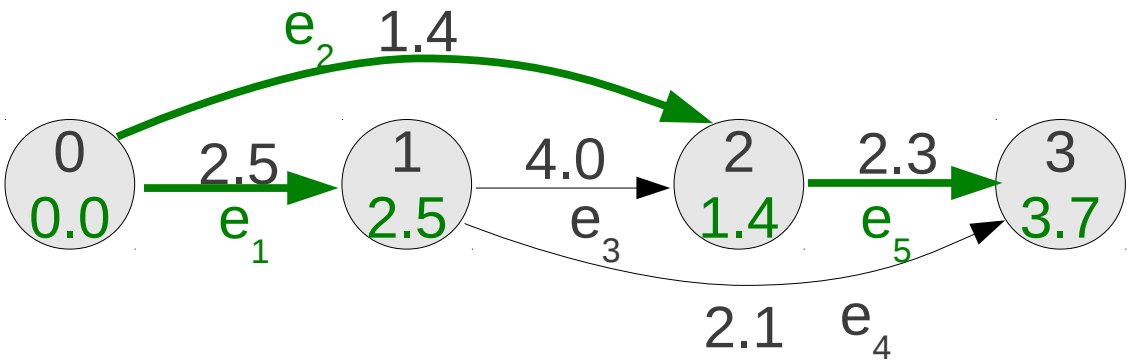
best_edge = (NULL, e_1, e_2, e_5)

後ろ向きステップ

後ろ向きステップのアルゴリズム



best_path = [ ]
next_edge = *best_edge*[*best_edge*.length – 1]
while *next_edge* != NULL
    add *next_edge* to *best_path*
    next_edge = *best_edge*[*next_edge.prev_node*]
reverse best_path

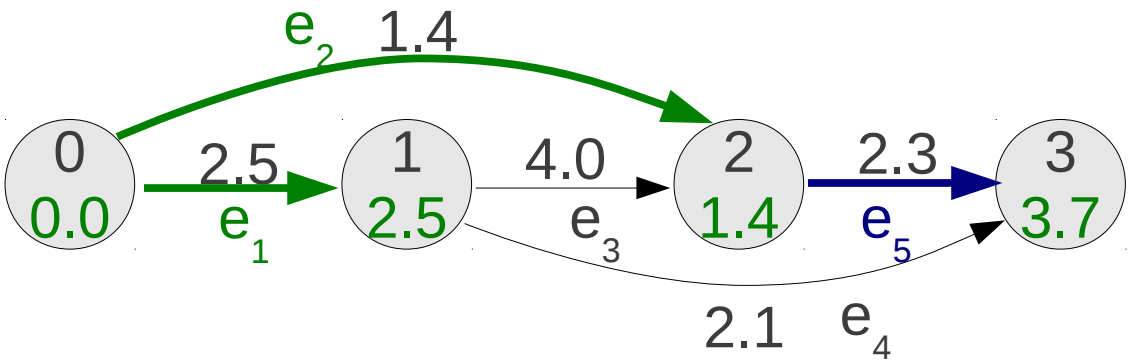


初期化 :

best_path = []

next_edge = best_edge[3] = e_5

後ろ向きステップの例



初期化 :

best_path = []

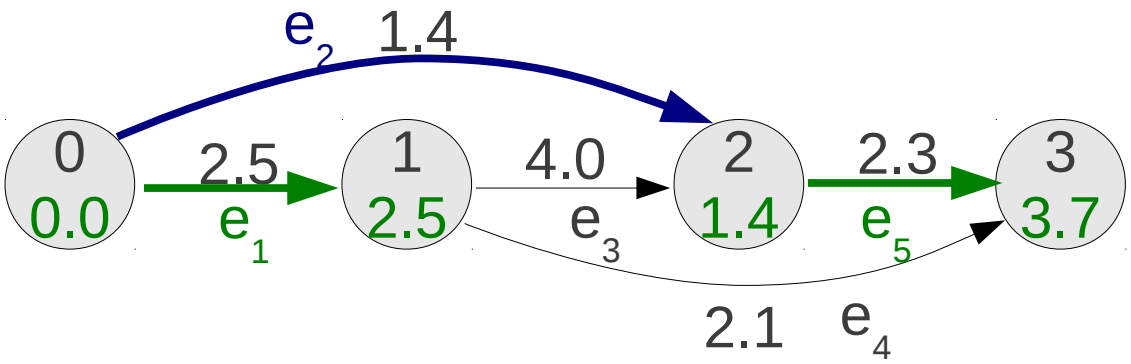
next_edge = best_edge[3] = e_5

e_5 を計算 :

best_path = [e_5]

next_edge = best_edge[2] = e_2

後ろ向きステップの例



初期化 :

best_path = []

next_edge = best_edge[3] = e_5

e_5 を計算 :

best_path = [e_5]

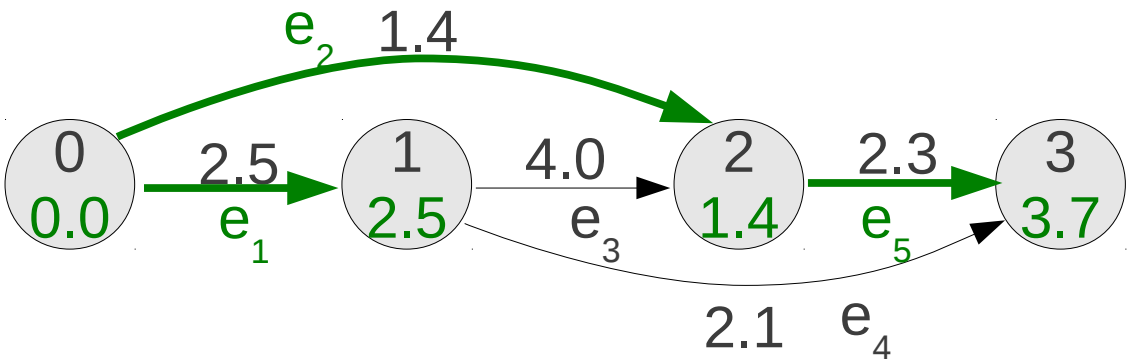
next_edge = best_edge[2] = e_2

e_2 を計算 :

best_path = [e_5, e_2]

next_edge = best_edge[0] = NULL

後ろ向きステップの例



初期化 :

best_path = []

next_edge = best_edge[3] = e_5

e_5 を計算 :

best_path = [e_5]

next_edge = best_edge[2] = e_2

e_5 を計算 :

best_path = [e_5, e_2]

next_edge = best_edge[0] = NULL

逆順に並べ替え :

best_path = [e_2, e_5]

必要なプログラミング技術：配列を逆順にする

- エッジの順番を逆にする必要がある

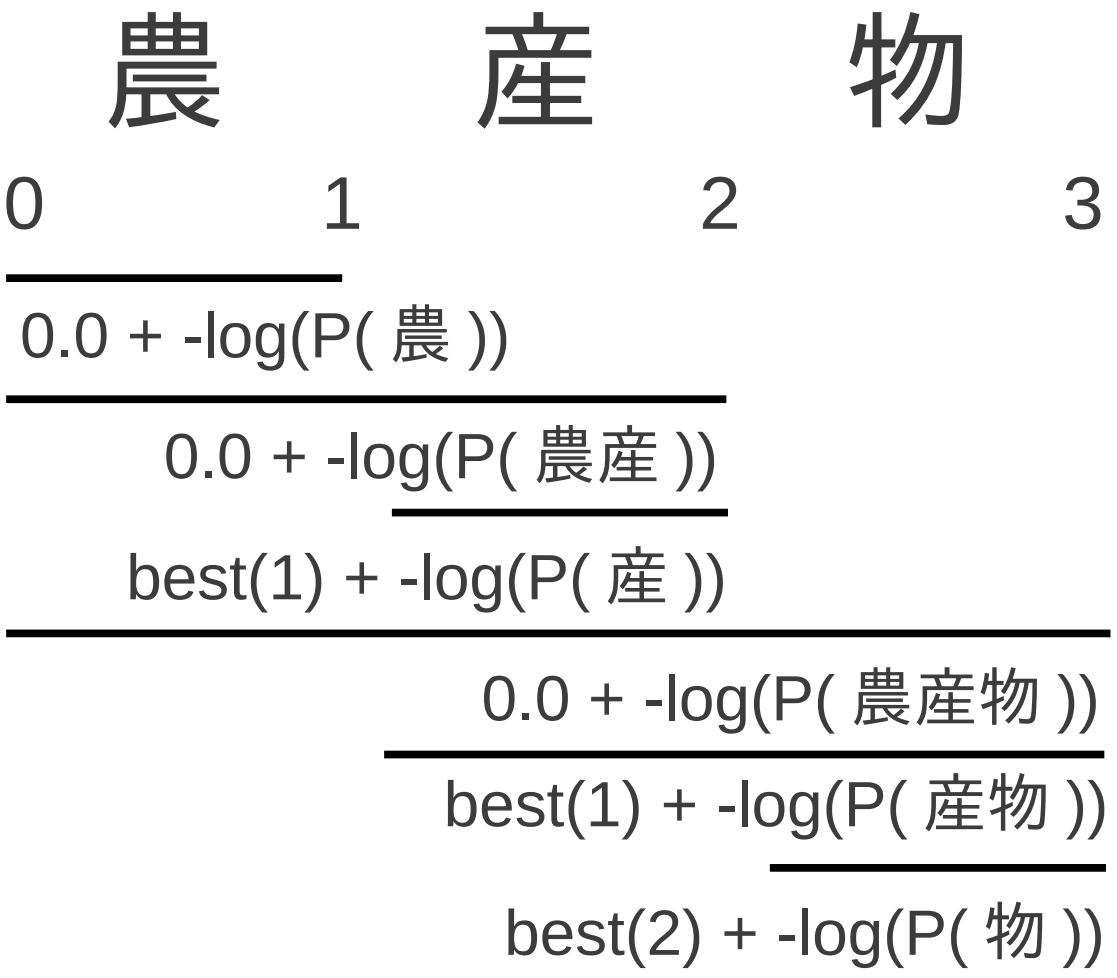
```
my_list = [ 1, 2, 3, 4, 5 ]
my_list.reverse()

print my_list
```

```
$ ./my-program.py
[5, 4, 3, 2, 1]
```

ビタビアルゴリズムを用いた単語分割

単語分割の前向きステップ



注：未知語モデル

- 1-gram モデル確率は以下のように定義した

$$P(w_i) = \lambda_1 P_{ML}(w_i) + (1-\lambda_1)\frac{1}{N}$$

- 全ての未知語に対して同一の確率を付与
  P_{unk}(“ｐｒｏｏｆ”) = 1/N
  P_{unk}(“校正（こうせい、英：ｐｒｏｏｆ”) = 1/N
- 単語分割に悪影響！
  - （未知語が１つでもあれば分割しない）
- 解決策：
  - もっと良いモデルを作成（少し難しいが、高精度）
  - 未知語の長さを１に限定（簡単、今回はこれを利用）

単語分割アルゴリズム (1)

load a map of *unigram* probabilities # 1-gram モデルの演習課題から

for each *line* in the *input*
  # 前向きステップ
  remove newline and **convert** *line* with “unicode()”
  best_edge[0] = *NULL*
  best_score[0] = *0*
  for each *word_end* in [1, 2, …, length(line)]
    best_score[*word_end*] = 10^{10} # とても大きな値に設定
    for each *word_begin* in [0, 1, …, *word_end* – 1]
      word = *line*[*word_begin*:*word_end*] # 部分文字列を取得
      if *word* is in *unigram* **or** length(*word*) = 1 # 既知語か長さ 1
        prob = P_{uni}(*word*) # 1-gram 演習と同じ
        my_score = *best_score*[*word_begin*] + -log(*prob*)
        if my_score < best_score[*word_end*]
          best_score[word_end] = my_score
          best_edge[*word_end*] = (word_begin, word_end)

単語分割アルゴリズム (2)

```
#  後ろ向きステップ
words = [ ]
next_edge = best_edge[ length(best_edge) – 1 ]
while next_edge != NULL
    #  このエッジの部分文字列を追加
    word = line[next_edge[0]:next_edge[1] ]
    encode word with the “encode()” function
    append word to words
    next_edge = best_edge[ next_edge[0] ]
words.reverse()
join words into a string and print
```

演習課題

演習課題

- 単語分割プログラムを作成
- テスト
  - モデル : `test/04-unigram.txt`
  - 入力 : `test/04-input.txt`
  - 正解 : `test/04-answer.txt`
- `data/wiki-ja-train.word` を使って学習した 1-gram モデルで、 `data/wiki-ja-test.txt` を分割
- 分割精度を以下のスクリプトで評価
  `script/gradews.pl data/wiki-ja-test.word my_answer.word`
- F 値 (F-meas) を報告

チャレンジ

- data/big-ws-model.txt に入っている、より大きなテキストで学習されたモデルを利用した分割精度を計る
- 未知語モデルの改善
- 2-gram モデルを使った単語分割