

クレジット:

Mathematics and Informatics Center メディアプログラミング入門 2020 山肩洋子

ライセンス:

利用者は、本講義資料を、教育的な目的に限ってページ単位で利用することができます。特に記載のない限り、本講義資料はページ単位でクリエイティブ・コモンズ 表示-非営利-改変禁止 ライセンスの下に提供されています。

<http://creativecommons.org/licenses/by-nc-nd/4.0/>

本講義資料内には、東京大学が第三者より許諾を得て利用している画像等や、各種ライセンスによって提供されている画像等が含まれています。個々の画像等を本講義資料から切り離して利用することはできません。個々の画像等の利用については、それぞれの権利者の定めるところに従ってください。



メディアプログラミング入門

第3回：形態素解析と言語モデル

火 5 @本郷 2020年6月16日

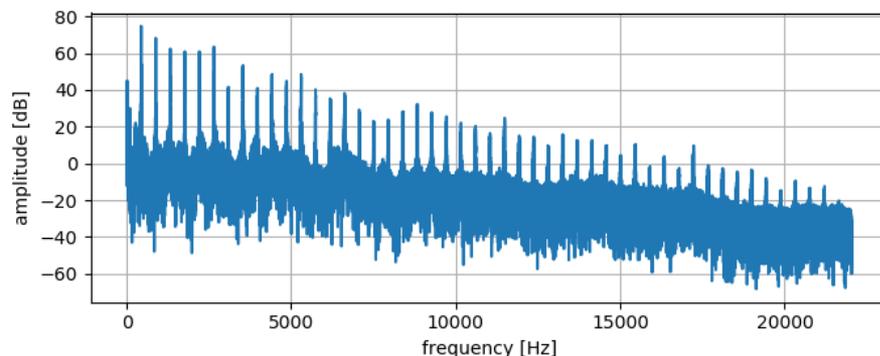
情報理工学系研究科 数理・情報教育研究センター

准教授 山肩 洋子

第2回の課題：楽器の音を分析しよう

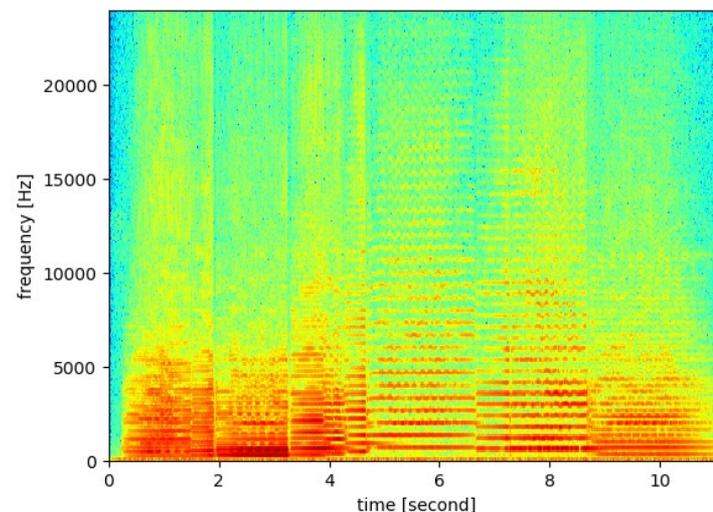
課題1：振幅スペクトルの描画

sound/Vaiolin.wavを開いて、その振幅スペクトルを描画してください。



課題2：スペクトログラムの描画

ヴァイオリンの音楽sounds/Violin-music.wavのスペクトログラムを描画してください（パラメータは教材と同じ）



第3回 テキスト解析 1 : ○○っぽい文を生成してみよう

講義内容 : Pythonでのテキストの扱いを学び、簡単な自然言語処理を体験する。自分が過去に書いたレポートからランダムな文を生成してみよう。

演習内容 : 形態素解析器の仕組みを解析した後、janomeを使って形態素解析を体験する。また、nltkを使って単語n-gramによる統計的言語モデルを学習し、ランダム文生成のプログラムを試行する

すももももものうち	
すもも	名詞, 一般, *, *, *, *, すもも, スモモ, スモモ
も	助詞, 係助詞, *, *, *, *, も, も, も
もも	名詞, 一般, *, *, *, *, もも, もも, もも
の	助詞, 連体化, *, *, *, *, の, ノ, ノ
うち	名詞, 非自立, 副詞可能, *, *, *, うち, ウチ, ウチ
EOS	

形態素解析

単語uni-gram:

“私”, “は”, “旅行”, “に”, “行き”, “たい”, “。”

単語bi-gram:

“私/は”, “は/旅行”, “旅行/に”, “に/行き”, “行き/たい”, “たい/。”

単語tri-gram:

“私/は/旅行”, “は/旅行/に”, “旅行/に/行き”, “に/行き/たい”, “行き/たい/。”

となる。

単語n-gram

「ジヨバンニはあぶなく赤いはなを切られるかも知れないと言うのでした。カムパネルラは、やどり木に云いました。」

「銀河鉄道の夜」っぽい文章

「吾輩はその後跋になったところを生れ得て歓天喜地の隙からすかして見ると三口べらべらやっているような返事をする。」

「吾輩は猫である」っぽい文章

「自分は、自分にはまるで見当がつかなかったあの時の自分の画才を信じさせたい、と思う。」

「人間失格」っぽい文章

ランダム文生成

本日の学習内容

- 形態素解析
 - 形態素解析は何に使うのか？
 - 文書の特徴分析
 - テキスト検索（インデックス検索）
 - 形態素解析の仕組み
 - コスト最小法
 - Viterbiアルゴリズムによる最尤パス推定
 - フリーの形態素解析ツールの紹介
 - 演習) 形態素解析器Janome
- 統計的n-gramモデル
 - 言語モデルとは何か、その応用先
 - 統計的n-gramの学習
 - モデルの利用
- 来週の課題

形態素解析

演習 : TextProcessing1.ipynb

何の小説？



何の小説？



文書の特徴を分析する手段の一つ：形態素解析

- **形態素**(morpheme)とは、意味を担う最小の言語要素
 - 「単語」よりも少し小さな分割
 - 「不確実」→「不（接頭語）＋確実（名詞）」
- コンピュータで形態素を自動的に制定する処理のことを**形態素解析**と呼ぶ
- 品詞や読みを特定したり、活用形を元に戻したりする目的にも使える

文「東京大学は欧米諸国の諸制度に倣った」の形態素解析結果：

東京大学	名詞,固有名詞,組織,***,東京大学,トウキョウダイガク,トーキョーダイガク
は	助詞,係助詞,****,は,ハ,ワ
欧米	名詞,固有名詞,地域,一般,**,欧米,オウベイ,オーベイ
諸国	名詞,一般,****,諸国,シヨコク,シヨコク
の	助詞,連体化,****,の,ノ,ノ
諸	接頭詞,名詞接続,****,諸,シヨ,シヨ
制度	名詞,一般,****,制度,セイド,セイド
に	助詞,格助詞,一般,***,に,ニ,ニ
倣っ	動詞,自立,**,五段・ワ行促音便,連用タ接続,倣う,ナラッ,ナラッ
た	助動詞,***,特殊・タ,基本形,た,タ,タ

形態素解析の利用先

- 単語分割 (Tokenizing)
 - 頻出語などを計算するためにはまず単語分割が必要
 - 先ほどのタグクラウドは、形態素解析結果から「名詞」のみを
あつめて、その頻度に応じて文字の大きさを決めてランダムに
配置したもの
- 品詞推定 (PoS tagging)
- 利用先：
 - **テキスト検索**
 - 読み方の推定: テキスト音声合成 (Text-to-speech, TTS)で利用
 - 「表記」から「読み」を求める「読み振り」
例「上手」→「うわて」「じょうず」「かみて」
例「正しく」→「ただしく」「まさしく」
 - 機械翻訳
 - トピック分析・文書分類
 - etc.

テキスト検索のタイプと取り得る手法

タスク：ある文書中に「パペマペ」という文字列が登場するか？

クライパペマチラキイニナコチグニチライキ**パペマペ**チマイニクゲマノケケクチスラ
セニクマポイパペマピクミクチイプピチモタセクチセイコバスニノセスイチクマニパ

事前にその言葉で検索されることが予測できない文字列
(たとえば自分のハンドルネームなど。未知語と呼ぶ) で検索する場合

→**逐次検索**

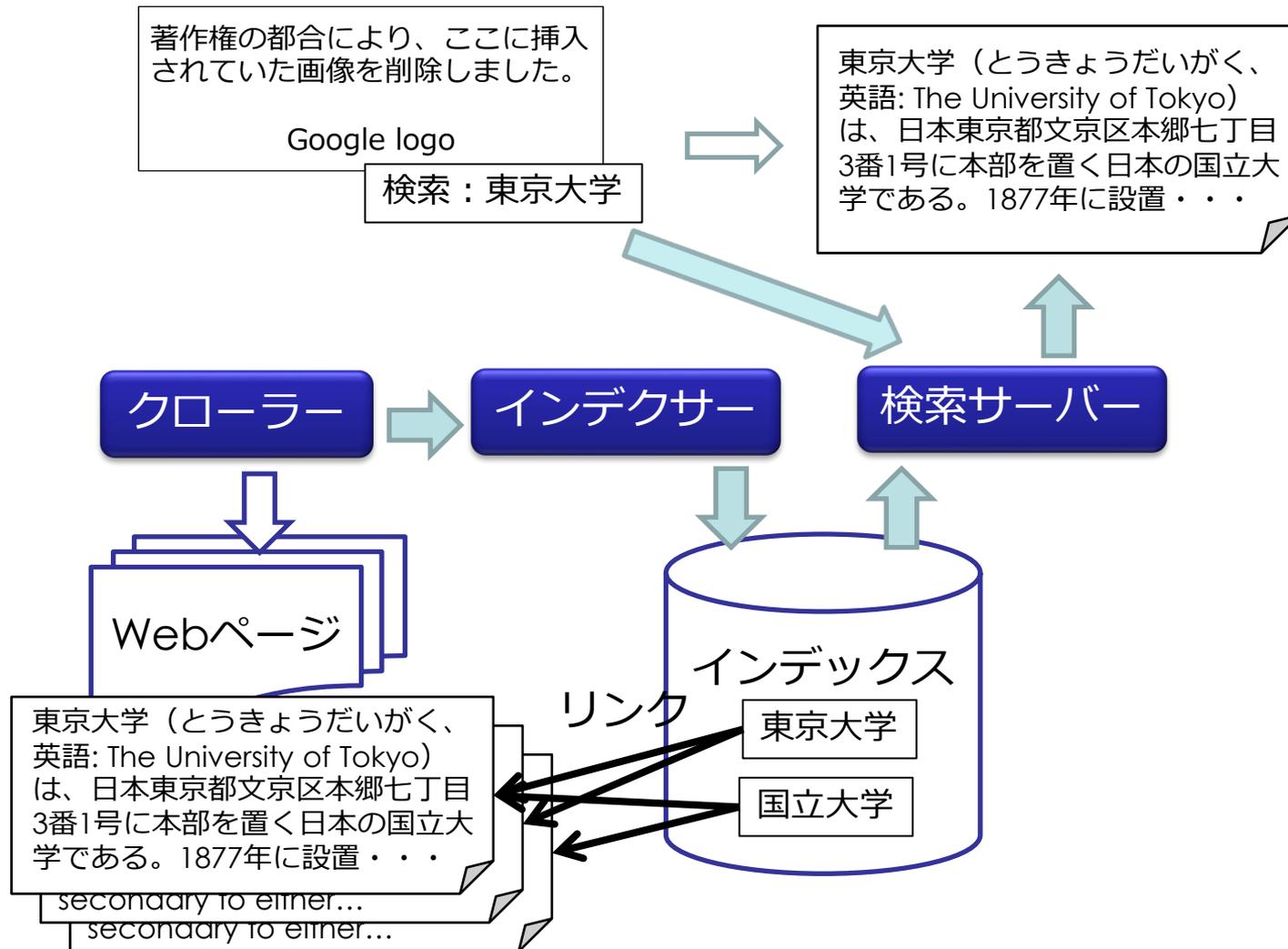
パペットマペットは、ウシとカエルの動物2匹からなる漫才コンビ。通称**パペマペ**。かわいらしい外見で自虐的なブラックユーモアをこめた掛け合いをネタとしている。
(wikipedia 「パペットマペット」より)

「パペマペ」という単語を知っていれば（検索辞書に登録されていれば）

1. あらかじめ各文書に「パペマペ」が登場するかを調べておき、
2. 「パペマペ」が検索されたら該当ページを表示すればいい

→**索引（インデックス）検索**

Web検索エンジンの仕組み



ref. (2020/06/13) Wikipedia 「東京大学」, <https://ja.wikipedia.org/wiki/%E6%9D%B1%E4%BA%AC%E5%A4%A7%E5%AD%A6>

インデクサーの仕組み

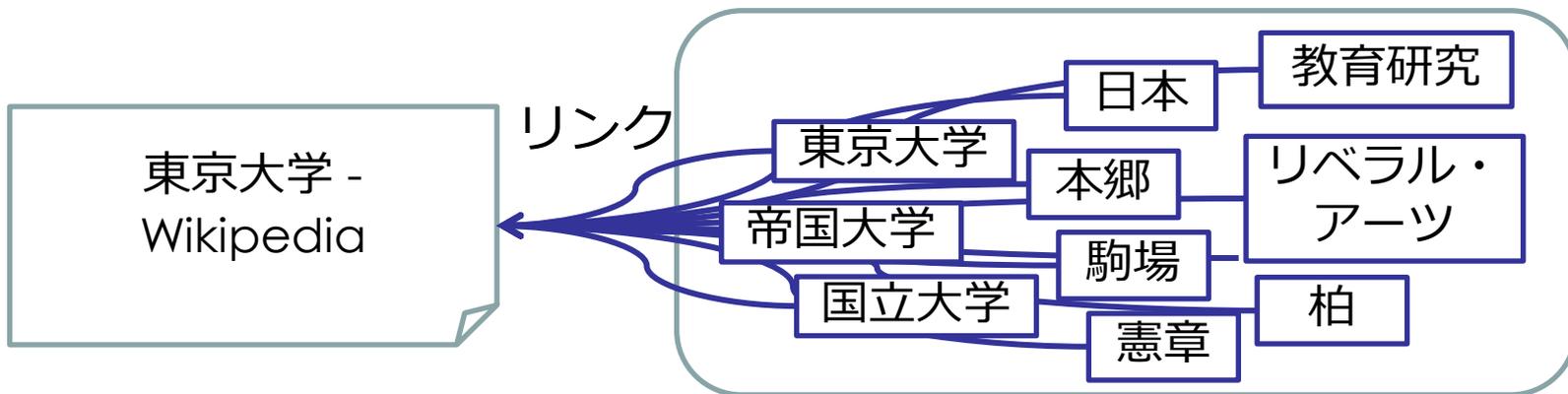
クローラーが見つめてきたあるページ「東京大学- Wikipedia」

東京大学（とうきょうだいがく、英語: The University of Tokyo）は、日本東京都文京区本郷七丁目3番1号に本部を置く日本の国立大学である。...

単語に分割し、キーワードを抽出

東京大学（とうきょうだいがく、英語：The University of Tokyo）は、日本東京都文京区本郷七丁目3番1号に本部を置く日本の国立大学である。...

インデックス



インデックスに対して検索

検索結果として出力

検索語(クエリ)

「東京大学」

インデックス

リンク

東京大学医学部
附属病院

東京大学 -
Wikipedia

東京大学
ホームページ

東京大学
総合図書館

大阪大学
ホームページ

東京大学

帝国大学

日本

...

ページランク



- Googleが開発当初に採用したランキングアルゴリズム
- Sergey Brin & Larry Page, "The Anatomy of a Large-Scale Hypertexturl Web Search Engine," Computer Networks Vol. 30, pp. 107-117, 1998.

$$PR(A) = (1 - d) + d\left(\frac{PR(T_1)}{C(T_1)} + \dots + \frac{PR(T_n)}{C(T_n)}\right)$$

- PR(A) : ページAのPageRank
 - PR(T₁) : ページAにリンクしているページT₁のPageRank
 - C(T₁) : ページT₁から出ているリンクの数 (ページAとページT₁を除く)
 - dを0<d<1の減衰係数とする。通常は0.85にセットする。
- あるページAのPageRankは、WebページAにリンクしている各ページのPageRankを、そのページから外向きのリンク (アウトリンク) 数で割った値の総和として定義
 - 実際の順位は、入力クエリとの関連度の計算が終わった後にPageRankが乗算されることにより決められる

ref. (2020/06/13) File:Sergey Brin cropped.jpg, [CC 表示 2.0](https://commons.wikimedia.org/wiki/File:Sergey_Brin_cropped.jpg), https://commons.wikimedia.org/wiki/File:Sergey_Brin_cropped.jpg

ref. (2020/06/13) File:Larry Page in the European Parliament, 17.06.2009 (cropped).jpg, [CC 表示-継承 3.0](https://commons.wikimedia.org/wiki/File:Larry_Page_in_the_European_Parliament,_17.06.2009_(cropped).jpg), [https://commons.wikimedia.org/wiki/File:Larry_Page_in_the_European_Parliament,_17.06.2009_\(cropped\).jpg](https://commons.wikimedia.org/wiki/File:Larry_Page_in_the_European_Parliament,_17.06.2009_(cropped).jpg)

インデクサーの仕組み

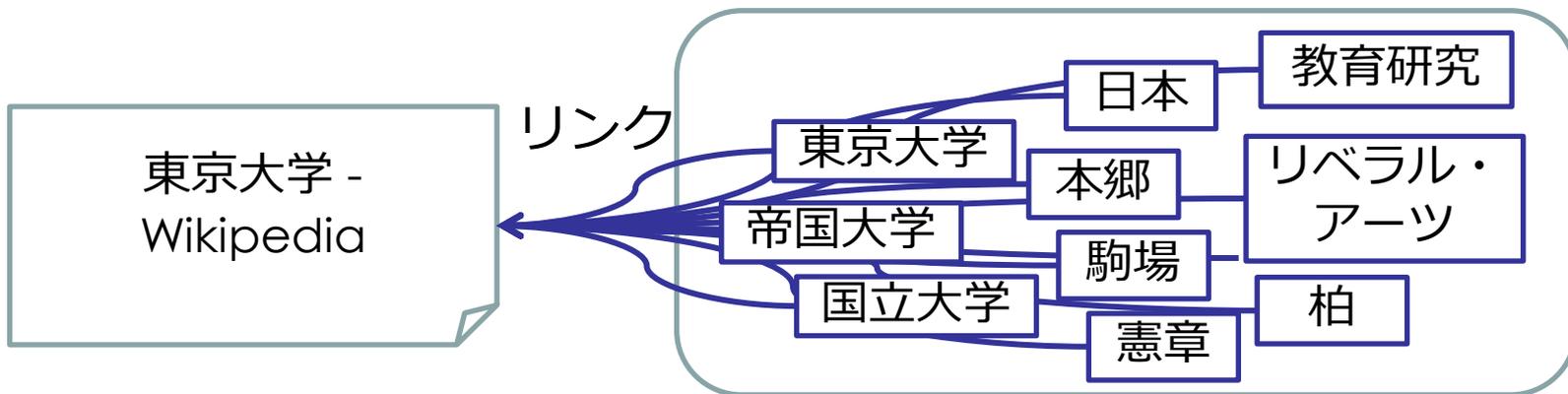
クローラーが見つめてきたあるページ「東京大学- Wikipedia」

東京大学（とうきょうだいがく、英語: The University of Tokyo）は、日本東京都文京区本郷七丁目3番1号に本部を置く日本の国立大学である。...

単語に分割し、キーワードを抽出

東京大学（とうきょうだいがく、英語：The University of Tokyo）は、日本東京都文京区本郷七丁目3番1号に本部を置く日本の国立大学である。...

インデックス



形態素解析の仕組み

日本語と英語の違い

- 英語の場合は単語分割は容易
 - Tokenization: ピリオドやダブルクォーテーションなどの記号を除けばよい
 - NLTKにはtokenizingツールがある
 - Tokenizeした後で品詞を推定：PoS tagging
 - 同じ表記の単語が複数の品詞を取る場合が多い
 - 例) "top" → 名詞 (the top of a mountain)、形容詞 (the top floor)、動詞 (top with sprinkles)
 - 例) "google" → 名詞と動詞の両方あり得る
- 日本語の場合は品詞と単語境界は同時に推定
 - 品詞のつながりやすさを目安に文字列を単語に区切る
 - 単語と品詞が決まれば、読みや発音なども同時に決まる

単語境界は必ずしも一意に決まらない！

「畜産物価格安定法」どこで区切る？

- あらゆるパターンの区切り方で区切る
- すべて辞書に登録されている単語となる区切り方が正解の候補

辞書に登録されている単語

畜産物価格安定法

畜/産物価格安定法

畜産/物価格安定法

畜/産/物価格安定法

畜産物/価格安定法

...

畜/産物/価格/安定/法

畜産/物/価格/安定/法

畜/産/物/価格/安定/法

...

畜産/物価/格安/定法

実はもっとある！
どれが正しい？

初期の試み 1 : 文法的接続可能性を使った形態素解析

品詞接続表

	名詞	助詞	形容詞	副詞	助動詞	動詞
名詞	○	○	×	×	○	×
助詞	○	×	○	○	×	○
形容詞	○	×	×	×	○	×
副詞	○	×	×	×	×	○
助動詞	○	×	×	×	○	○
動詞	○	×	×	×	○	○

- 標準的な日本語の文法では「学校へ行く」のように名詞と動詞の間には助詞が必要
- しかし口語では、助詞を抜かして名詞に動詞が後続することができるので不十分（例：「学校行く」）

例参照：“自然言語処理—基礎と応用—”，田中穂積，電子情報通信学会，1999

初期の試み2：分割数最小法

- 入力文字列を構成する単語の総数が最小になる解釈を優先する
- 和文だと割とうまく行くが、うまく行かないケースに対処することが難しい

区切りの数が最小なので
これが選ばれる

「言語学入門講座」はどこで区切る？

- 言語学 | 入門 | 講座 → 3つ
- 言語 | 学 | 入門 | 講座 → 4つ
- 言語学 | 入 | 門 | 講座 → 4つ

しかし正解が得られない場合もある

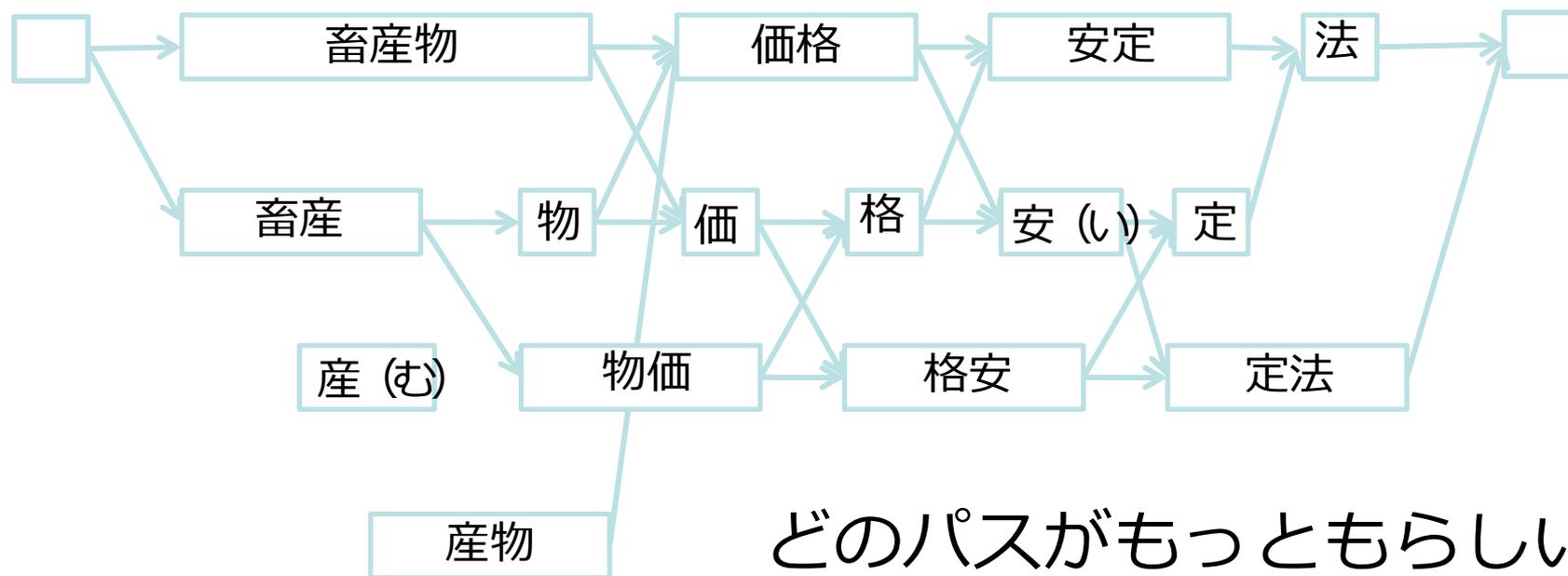
畜産物/格安/安定/法 → 4単語

畜産/物価/格安/定法 → 4単語

グラフを使った形態素解析

1. 辞書に登録されている単語を並べる
2. 隣り合うもの同士にリンクを張る
3. 文頭から始めて文末にたどり着けたものを正解とする

畜 産 物 価 格 安 定 法

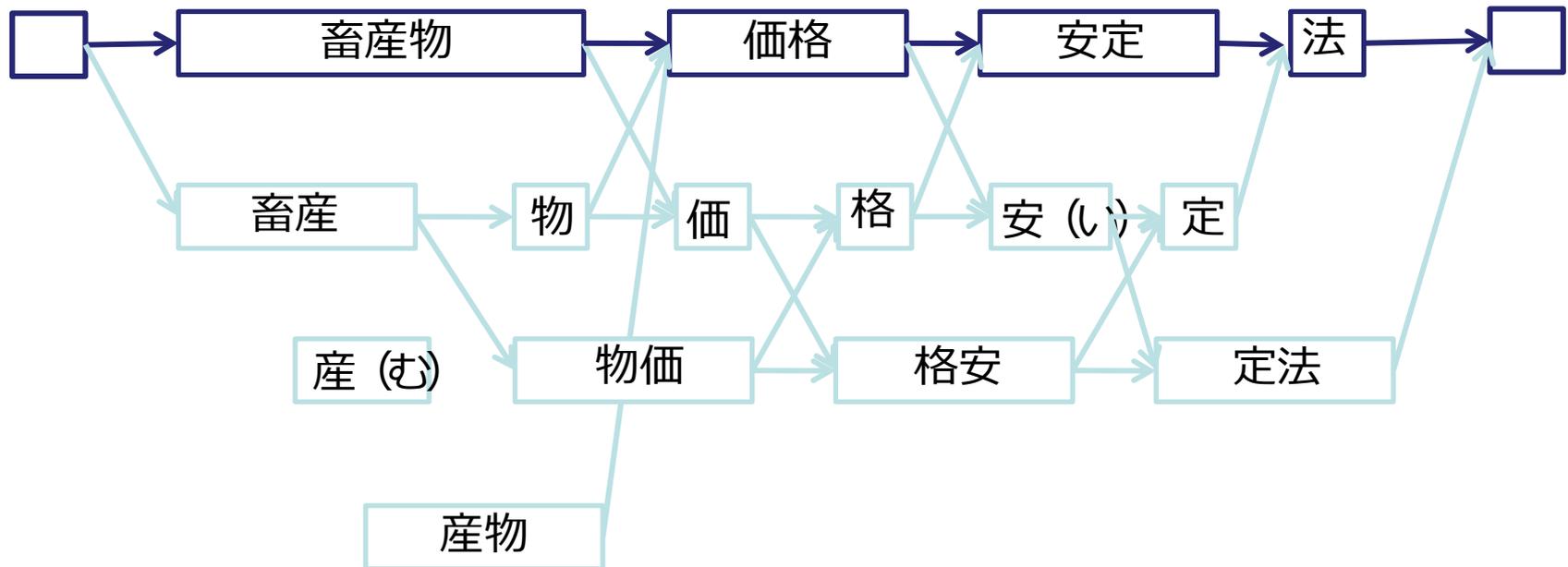


例参照：“自然言語処理—基礎と応用—”，田中穂積，電子情報通信学会，1999.

初期の試み3：最長一致法

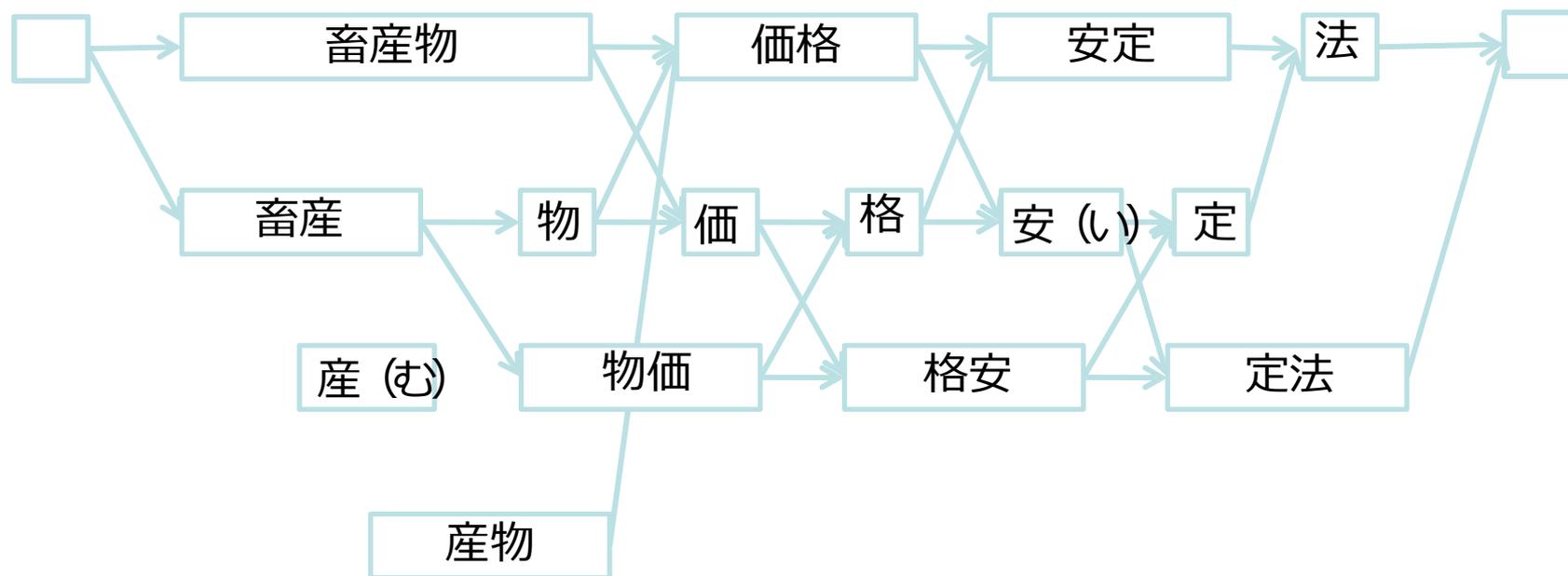
- 文字列の先頭から解析を始め、継続し得る単語のうち最長の単語を選択して先に進む
- 前半で間違った経路を選んでしまうと後で軌道修正できない

畜 産 物 価 格 安 定 法



コスト最小法

- グラフの単語(ノード)と単語間の接続(エッジ)にコストを付与
- 経路上のコストの和が最小となるような単語分割を選択

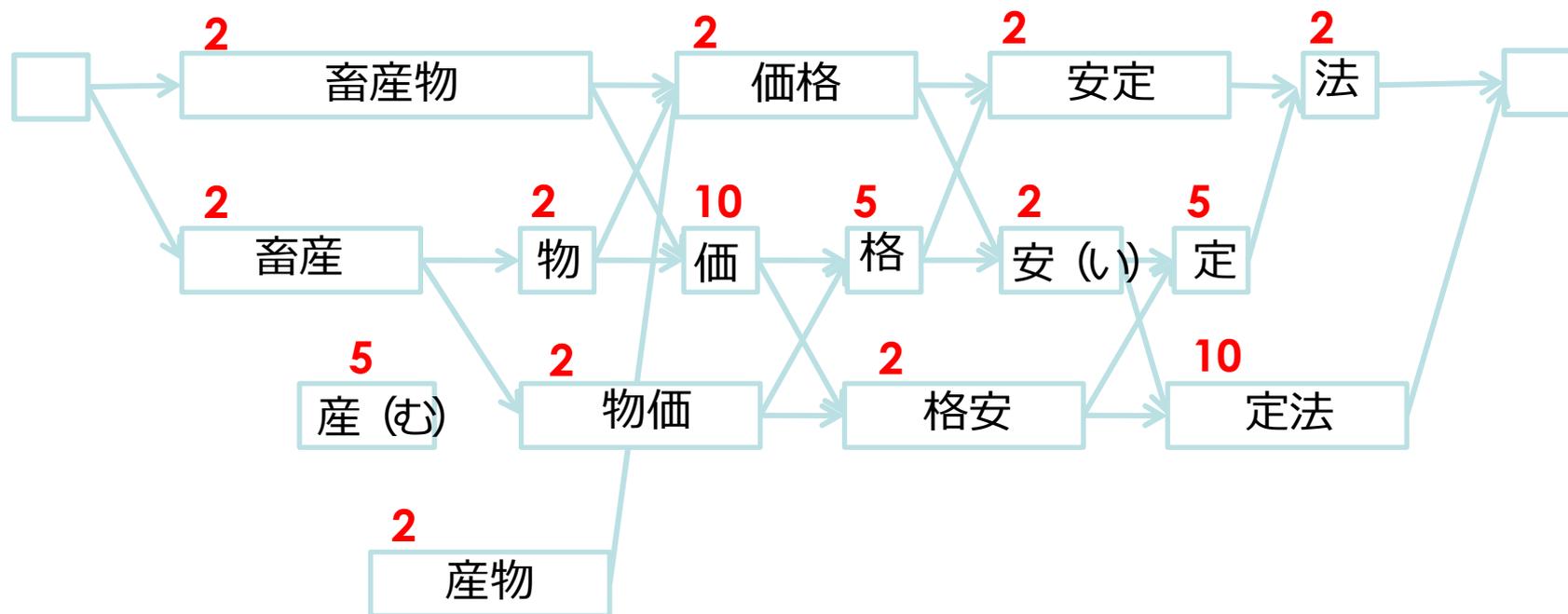


コスト最小法

- グラフの単語(ノード)と単語間の接続(エッジ)にコストを付与
- 経路上のコストの和が最小となるような単語分割を選択

コストは2種類 :

1. **生起コスト** : 稀にしか使われない単語のコストは大きくする



コスト最小法

- グラフの単語(ノード)と単語間の接続(エッジ)にコストを付与
- 経路上のコストの和が最小となるような単語分割を選択

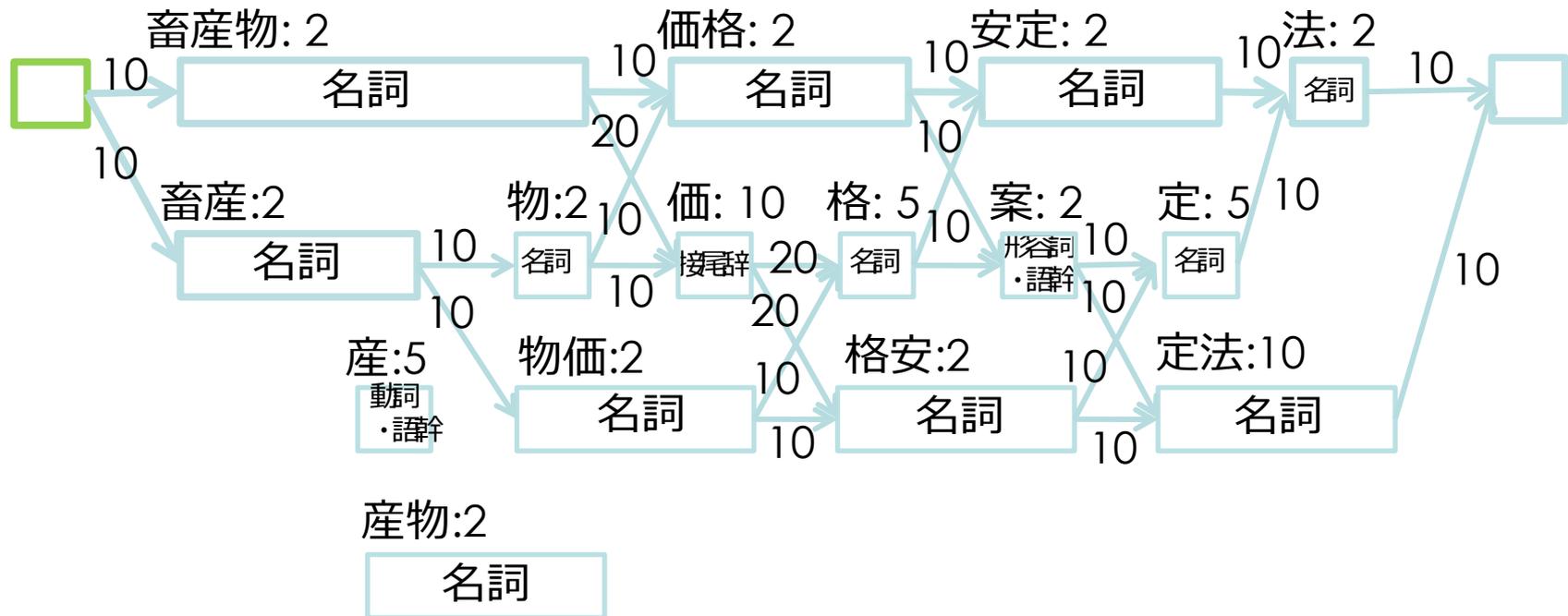
コストは2種類 :

1. **生起コスト** : 稀にしか使われない単語のコストは大きくする
2. **接続コスト** : 稀にしか接続しない単語の間のコストを大きくする



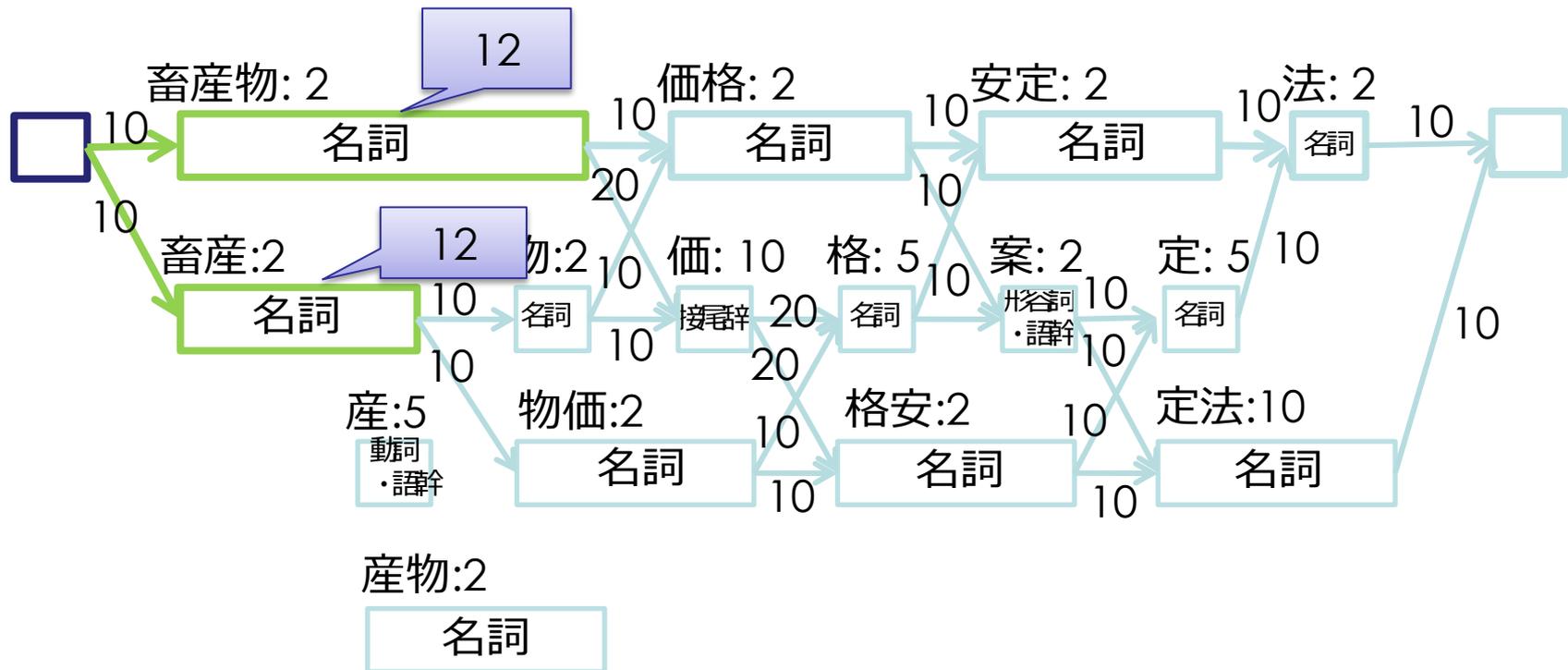
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



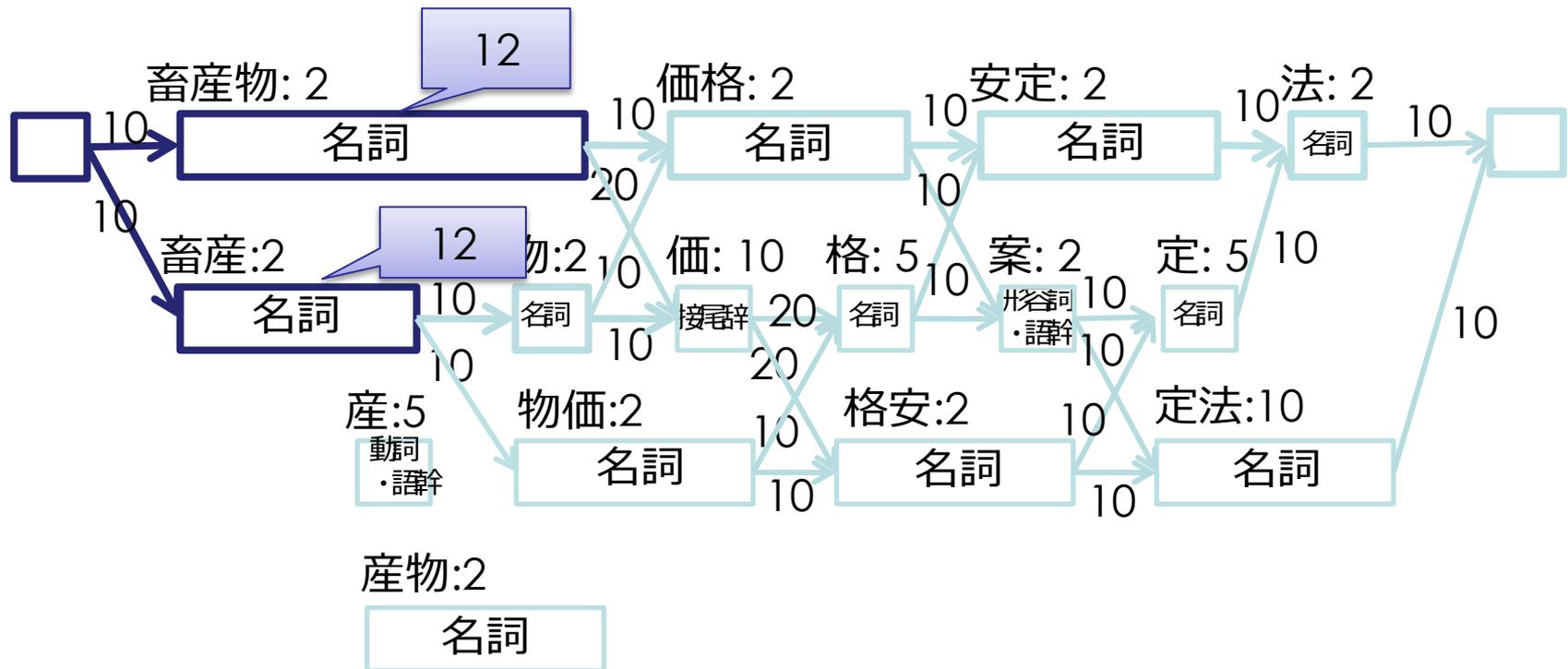
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



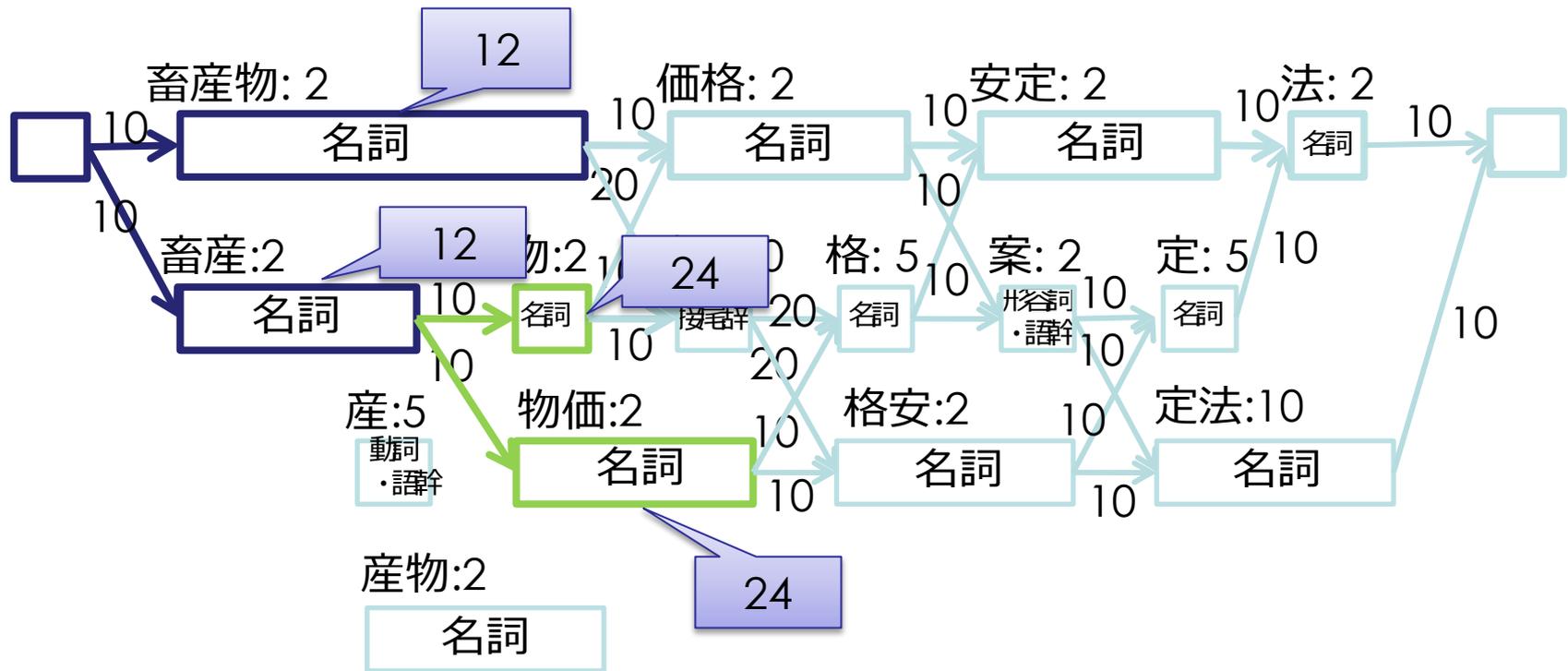
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



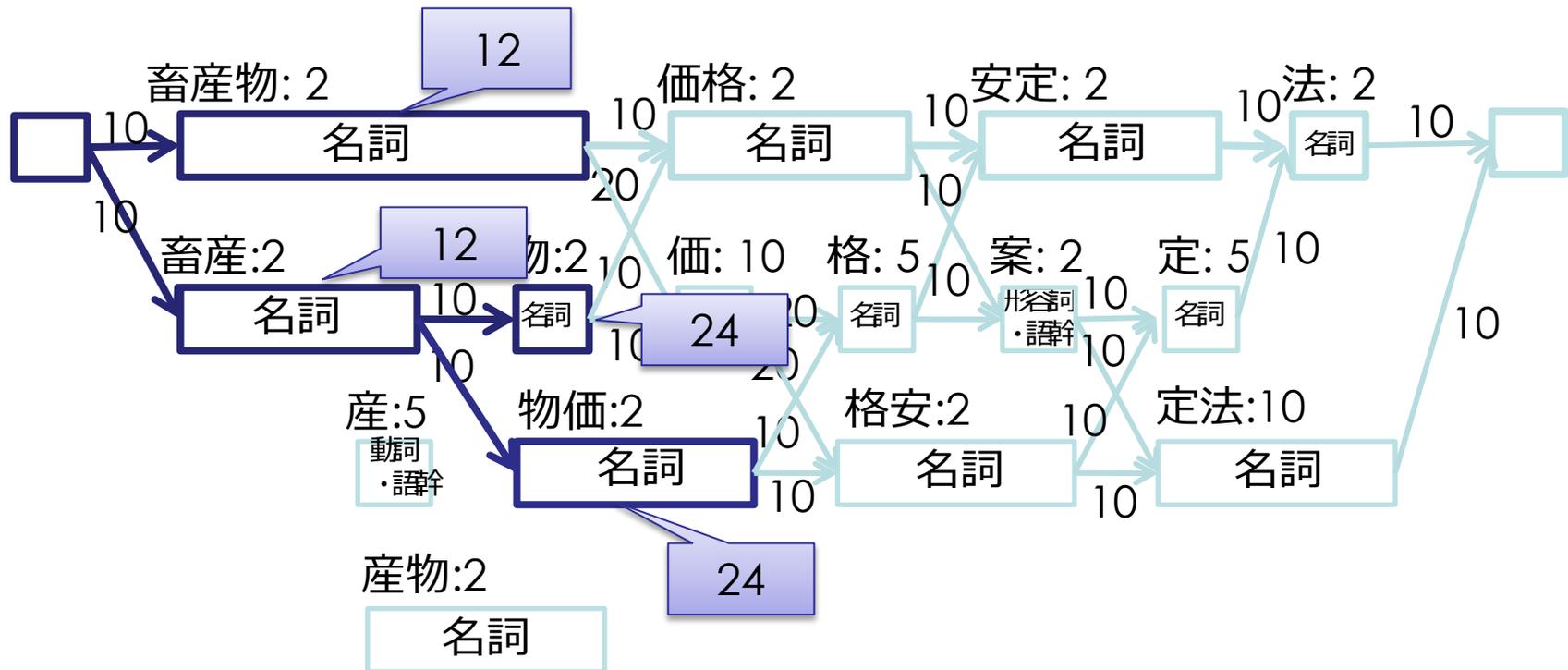
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



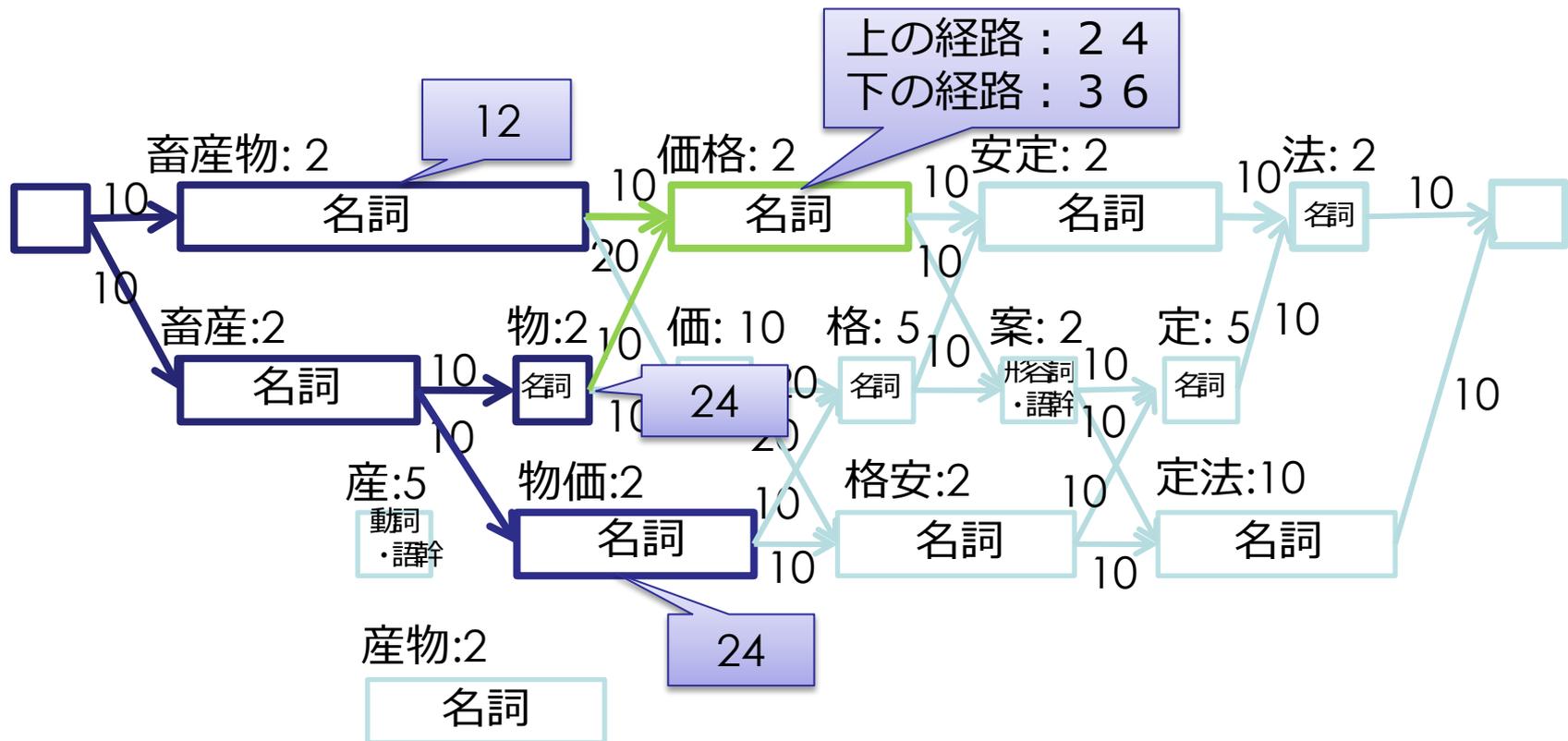
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



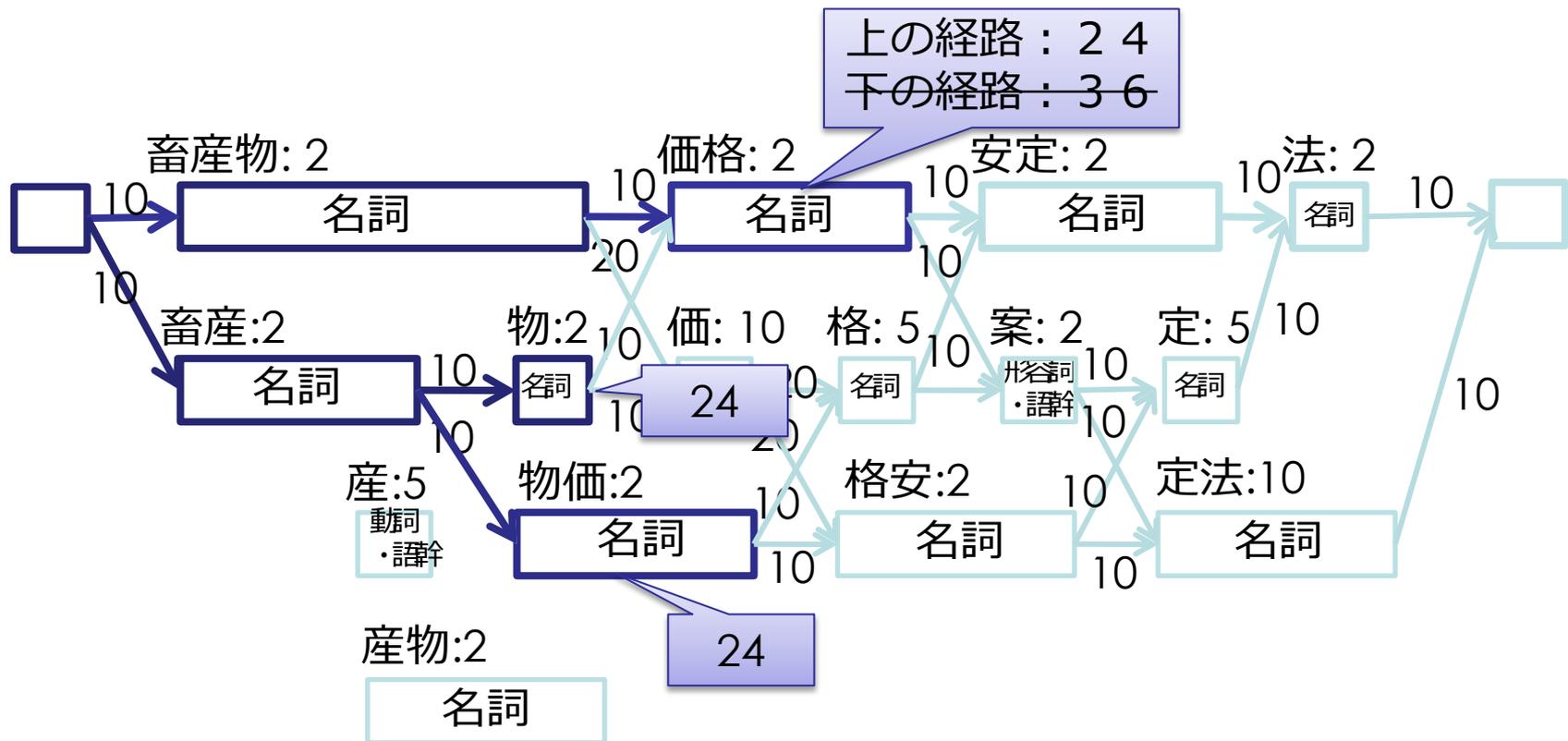
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



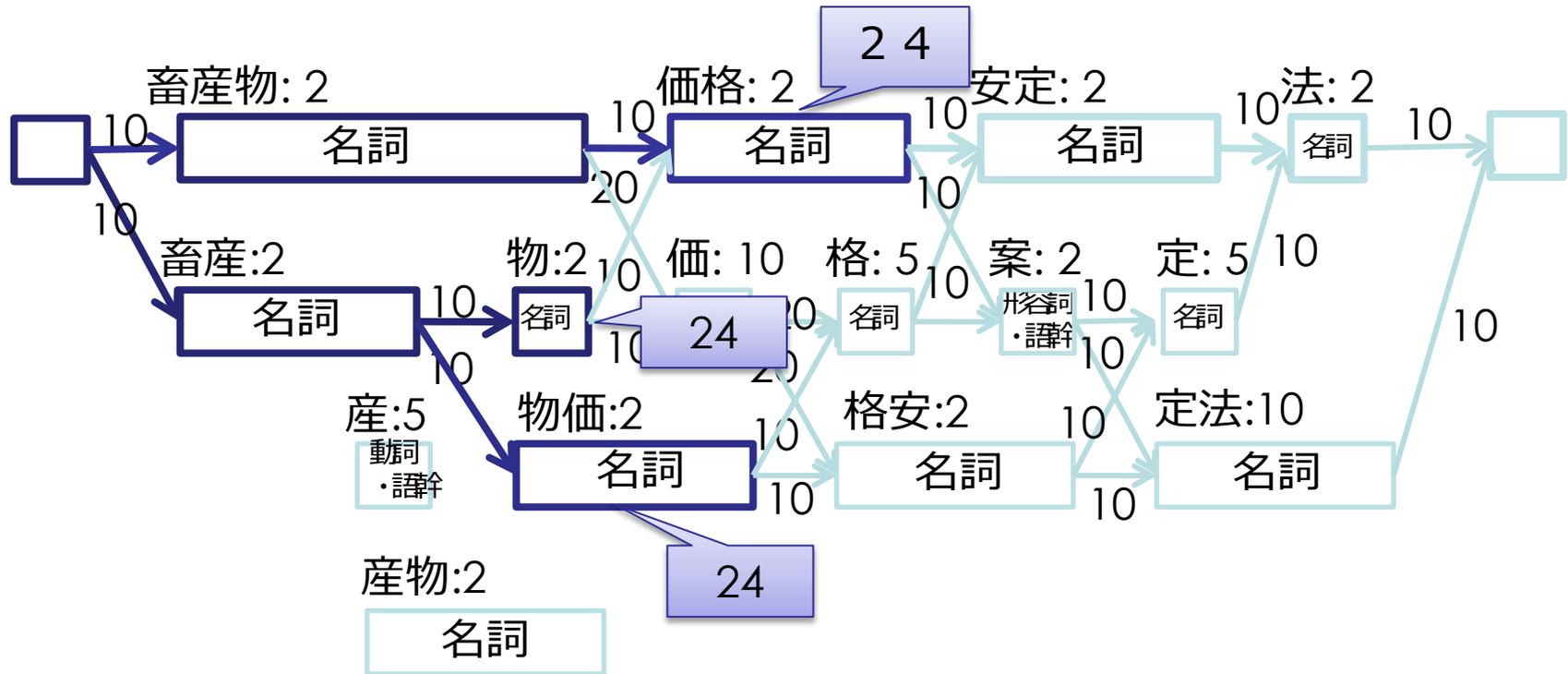
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



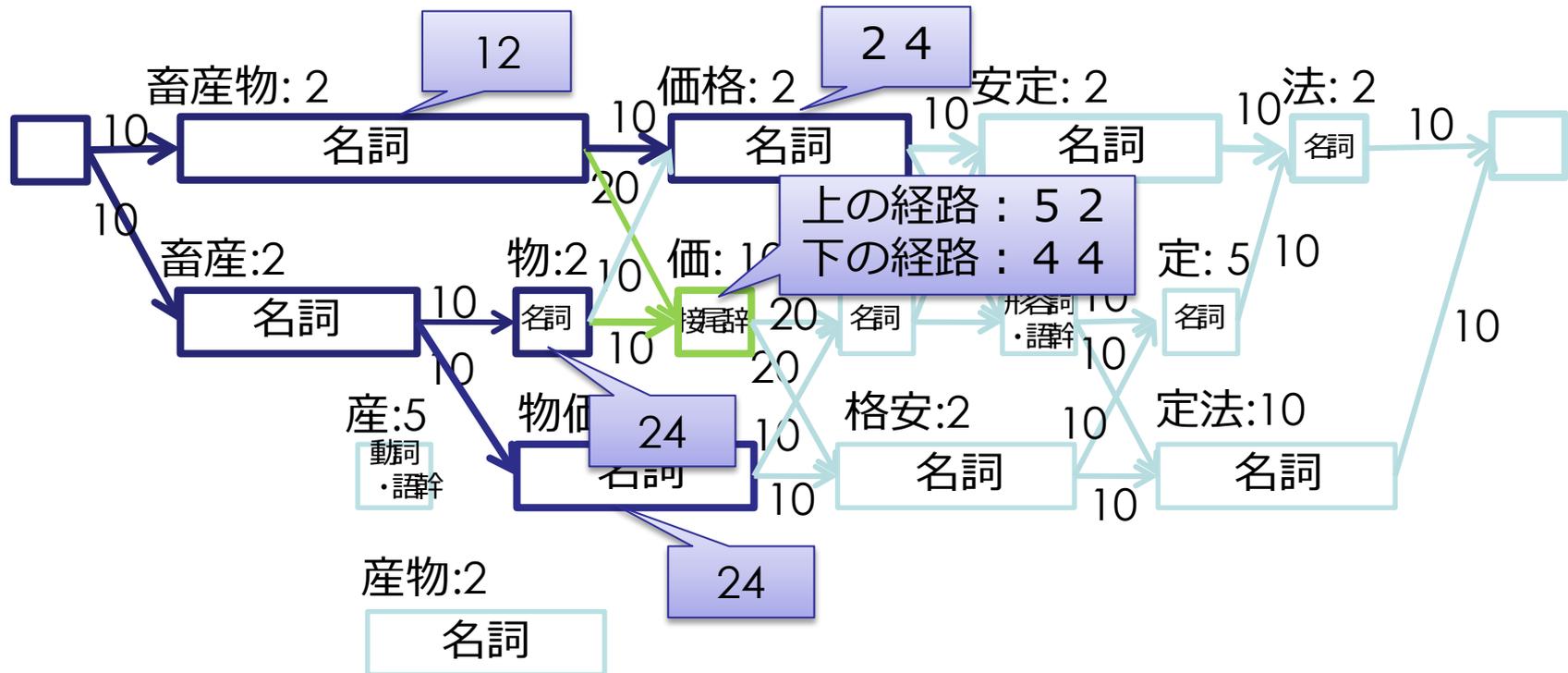
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



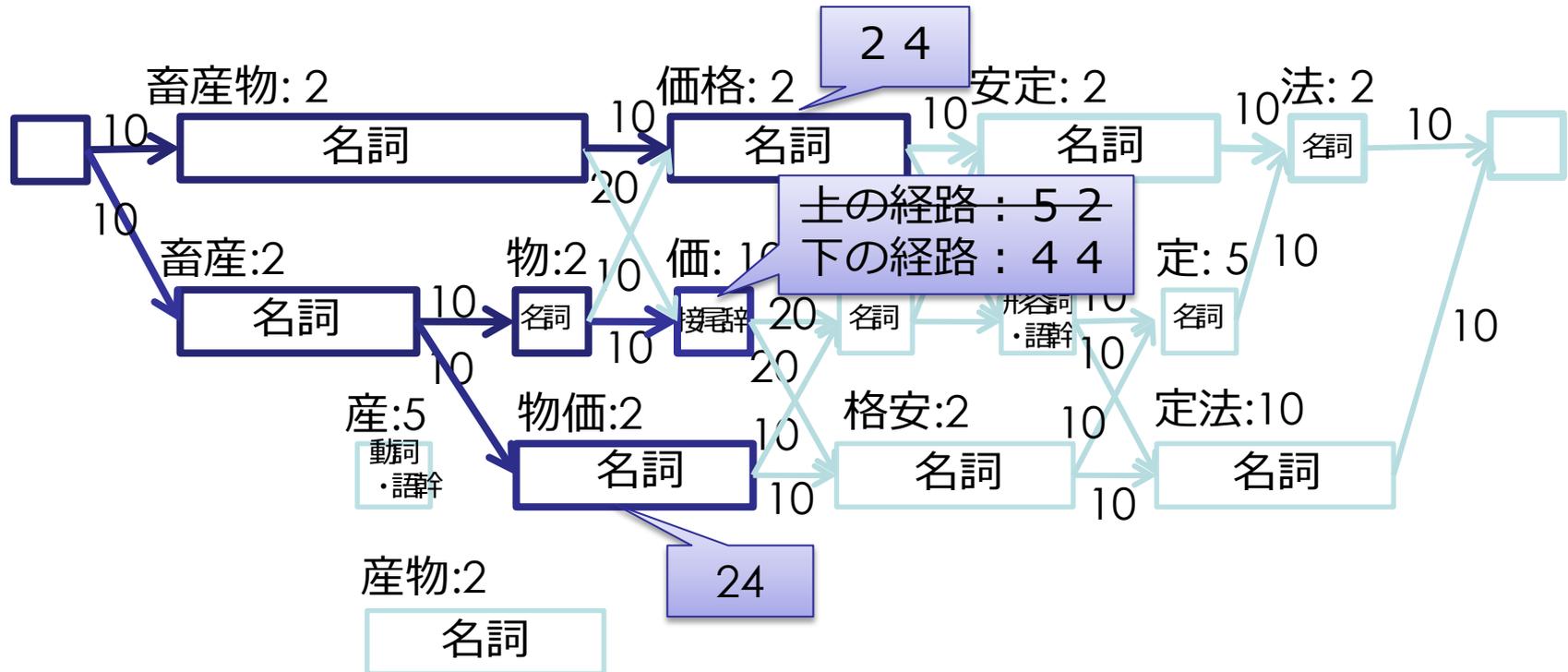
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



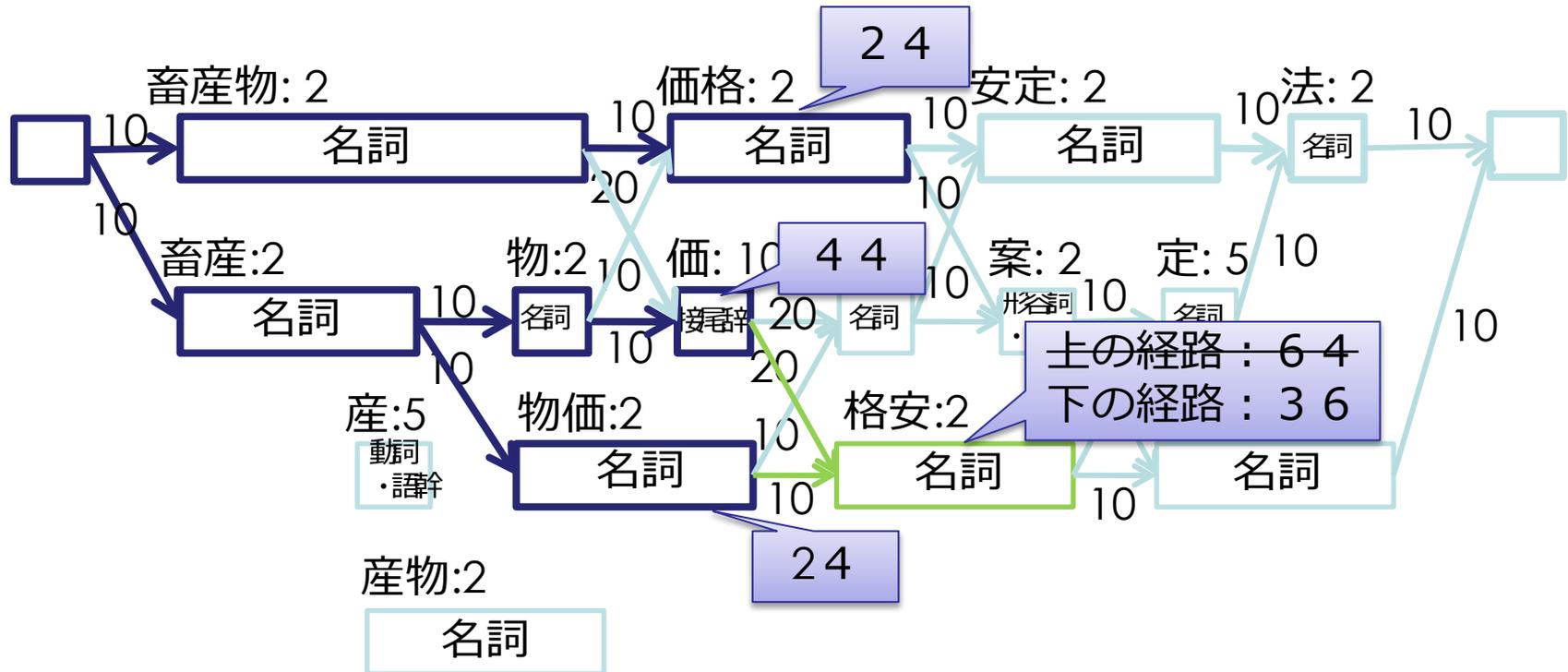
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



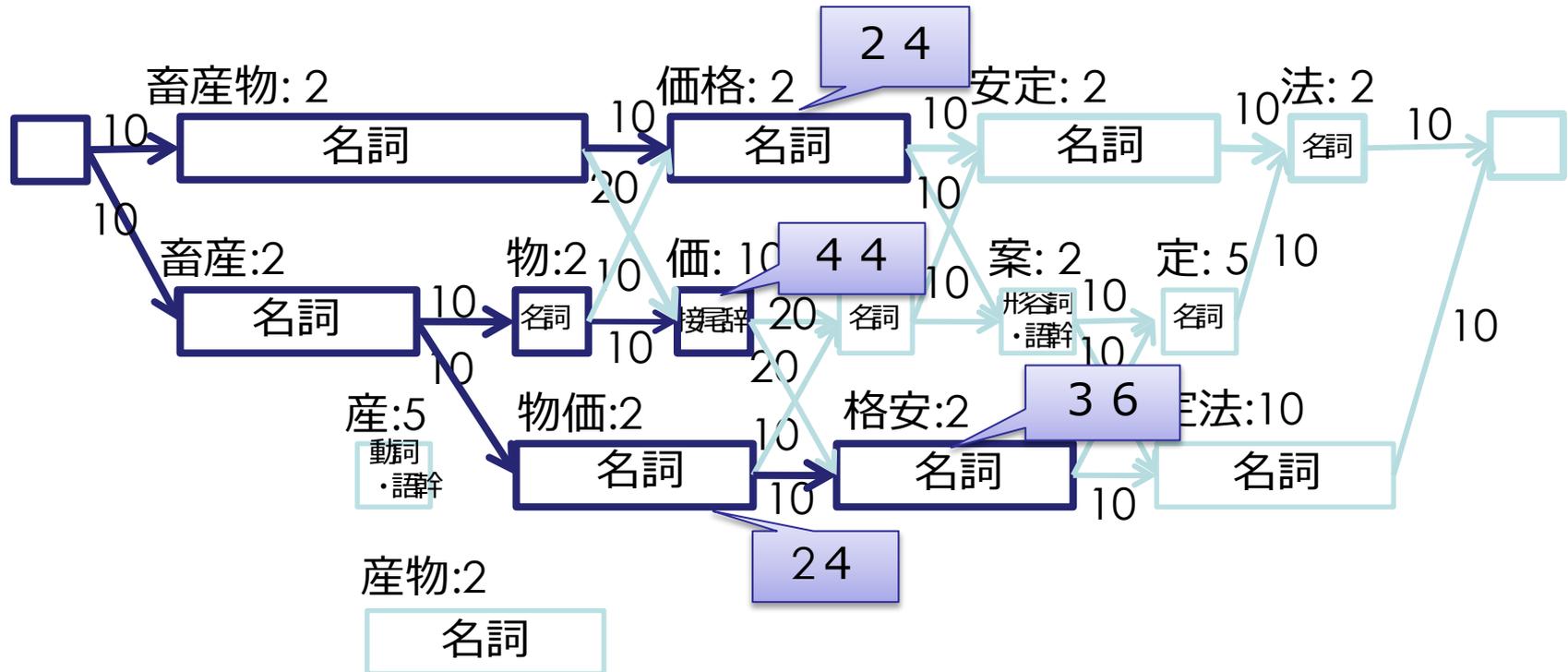
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



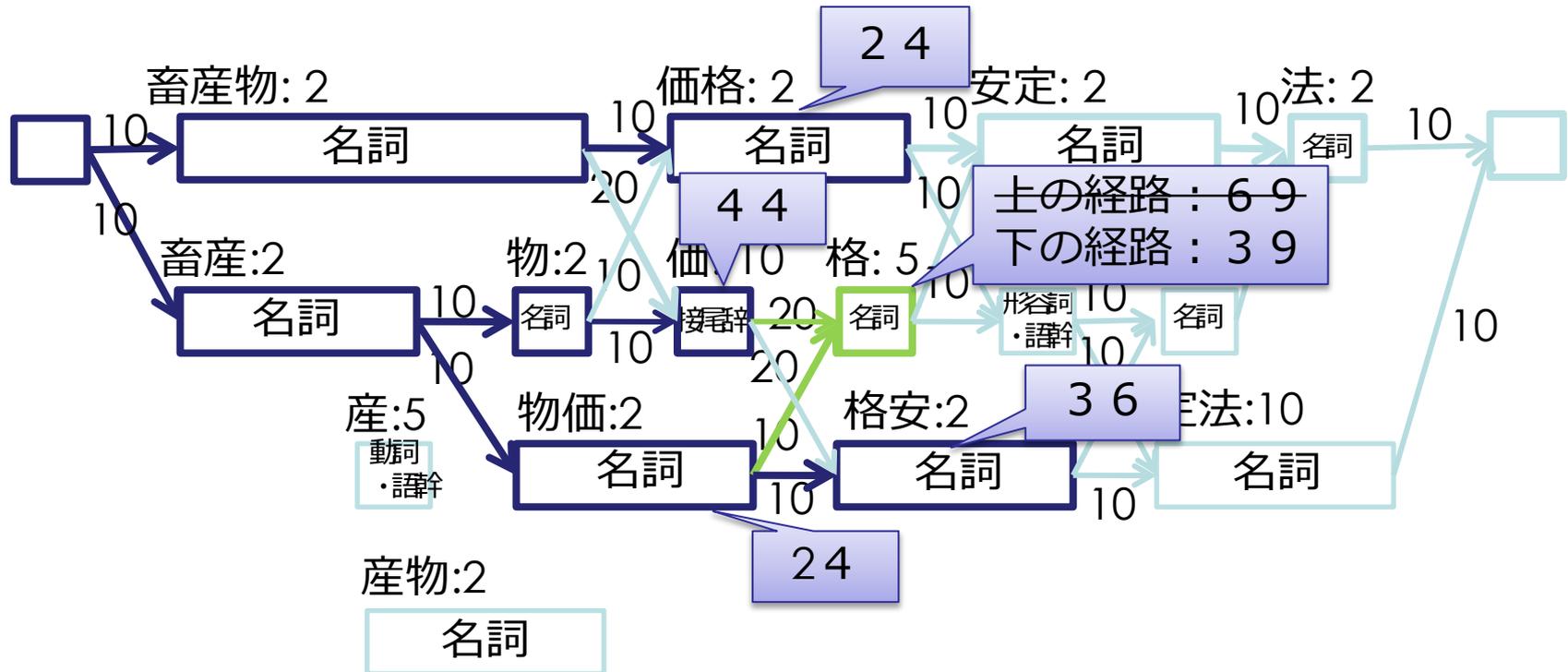
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



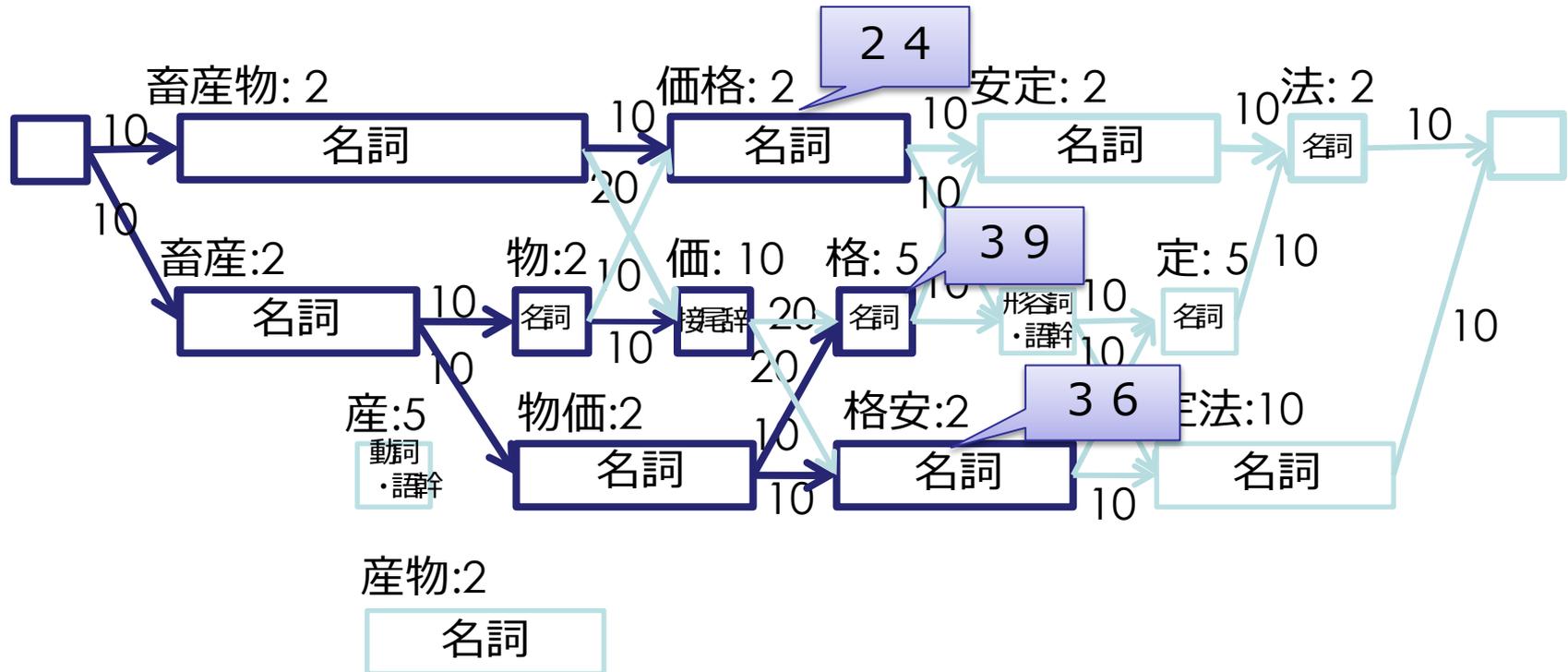
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



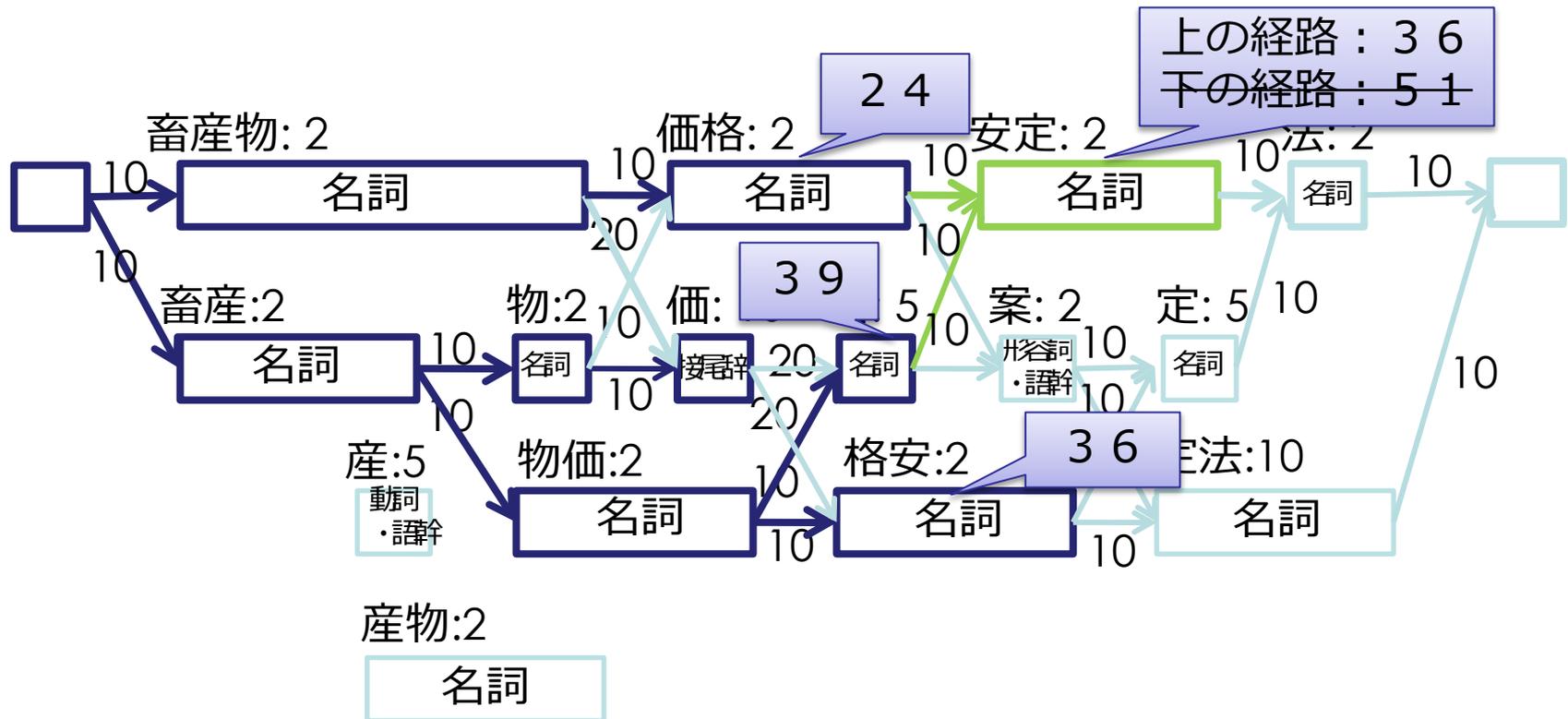
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



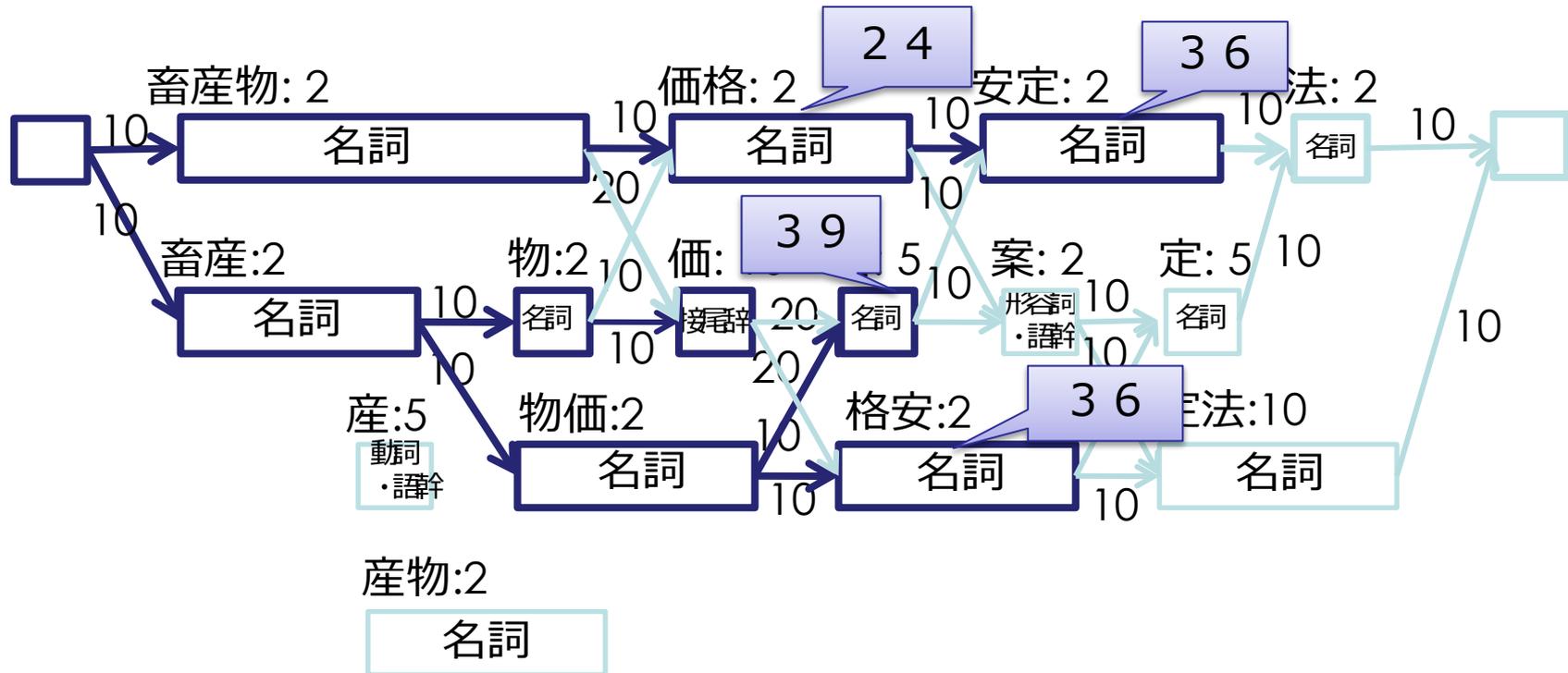
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



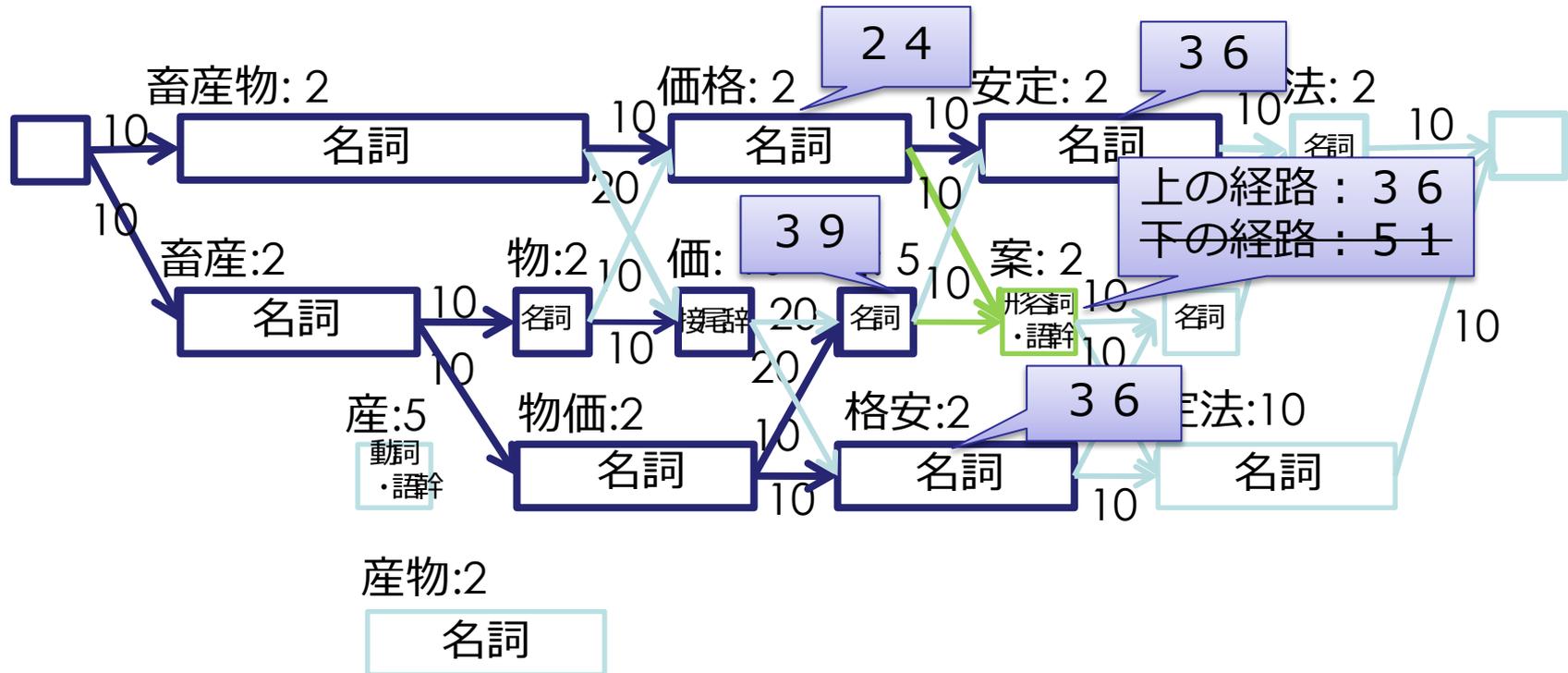
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



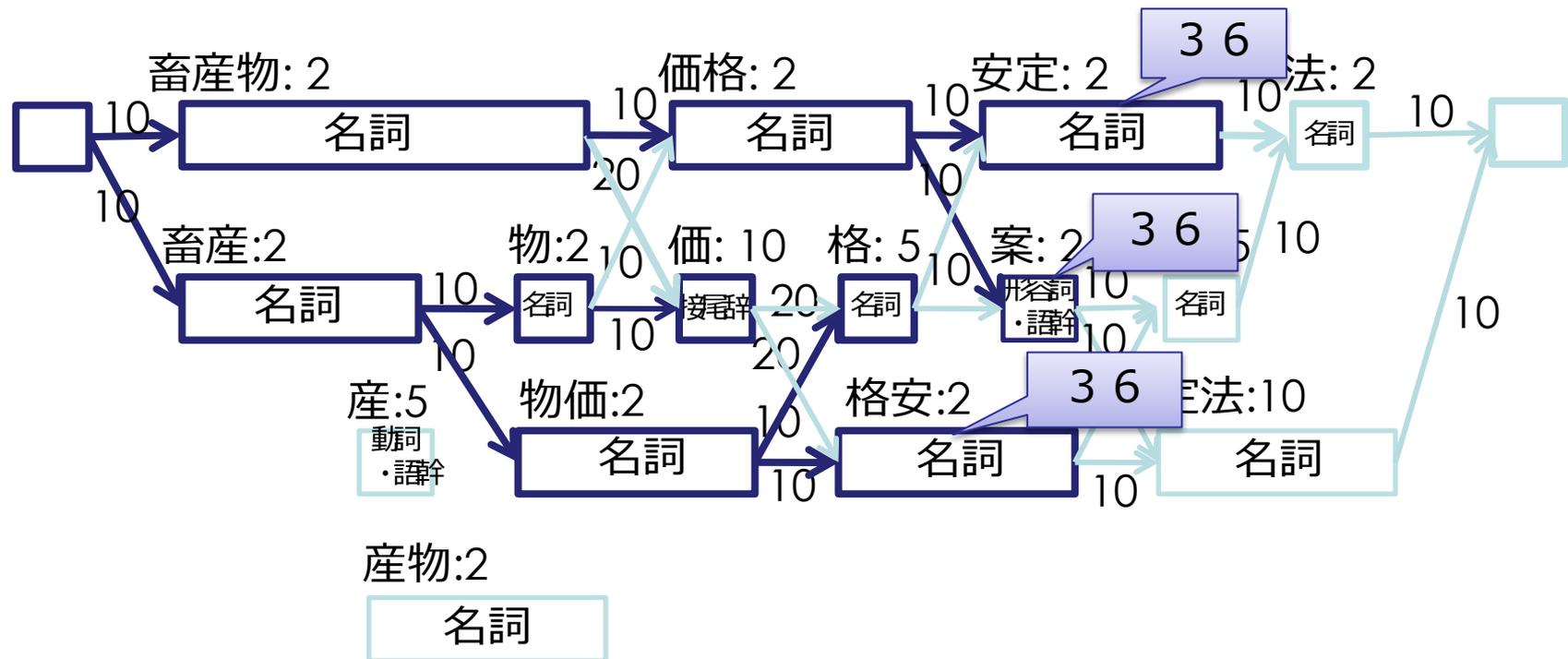
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



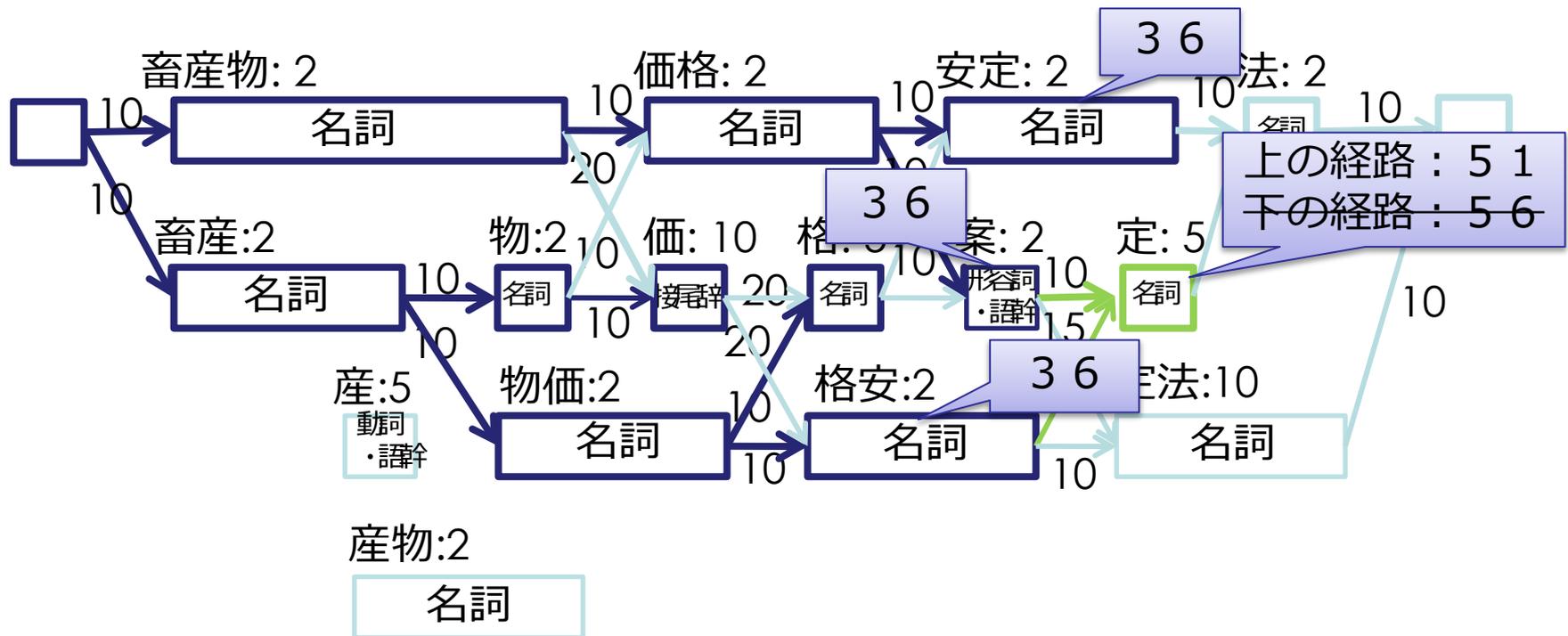
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



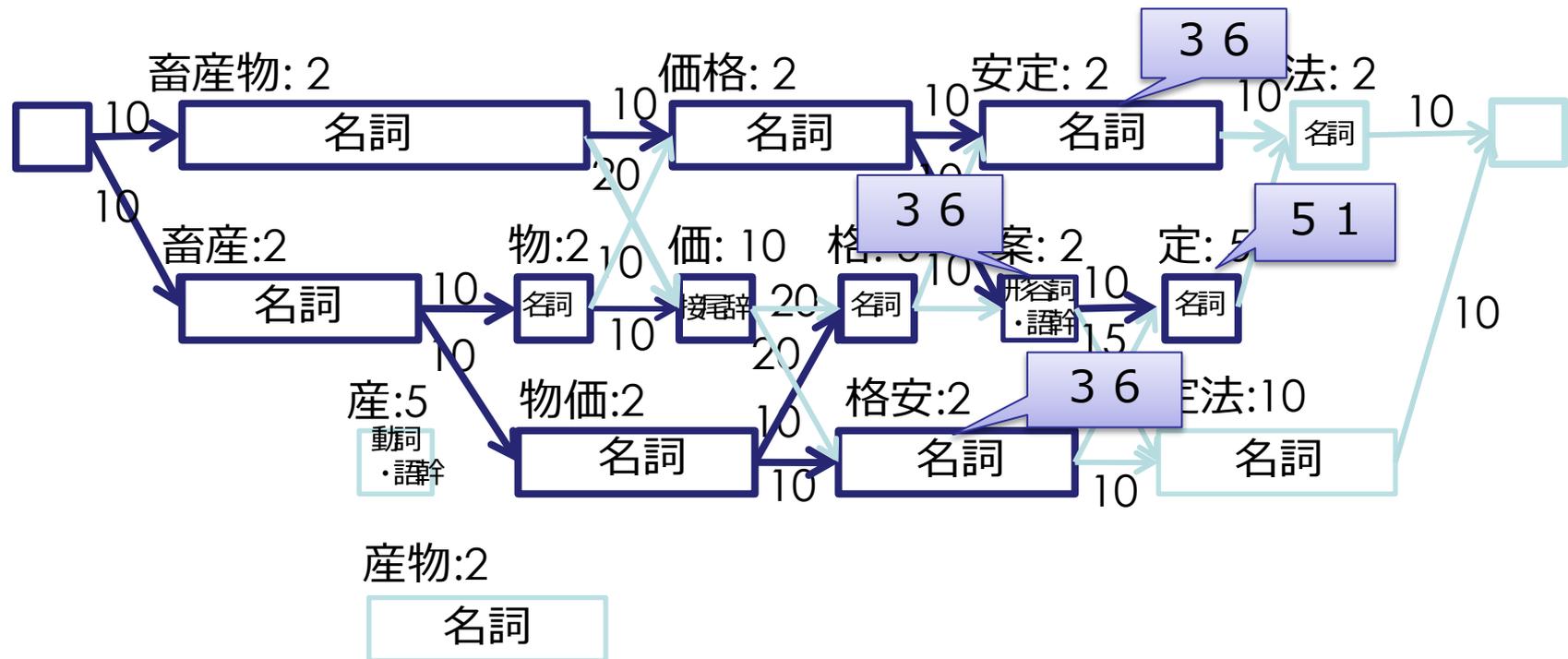
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



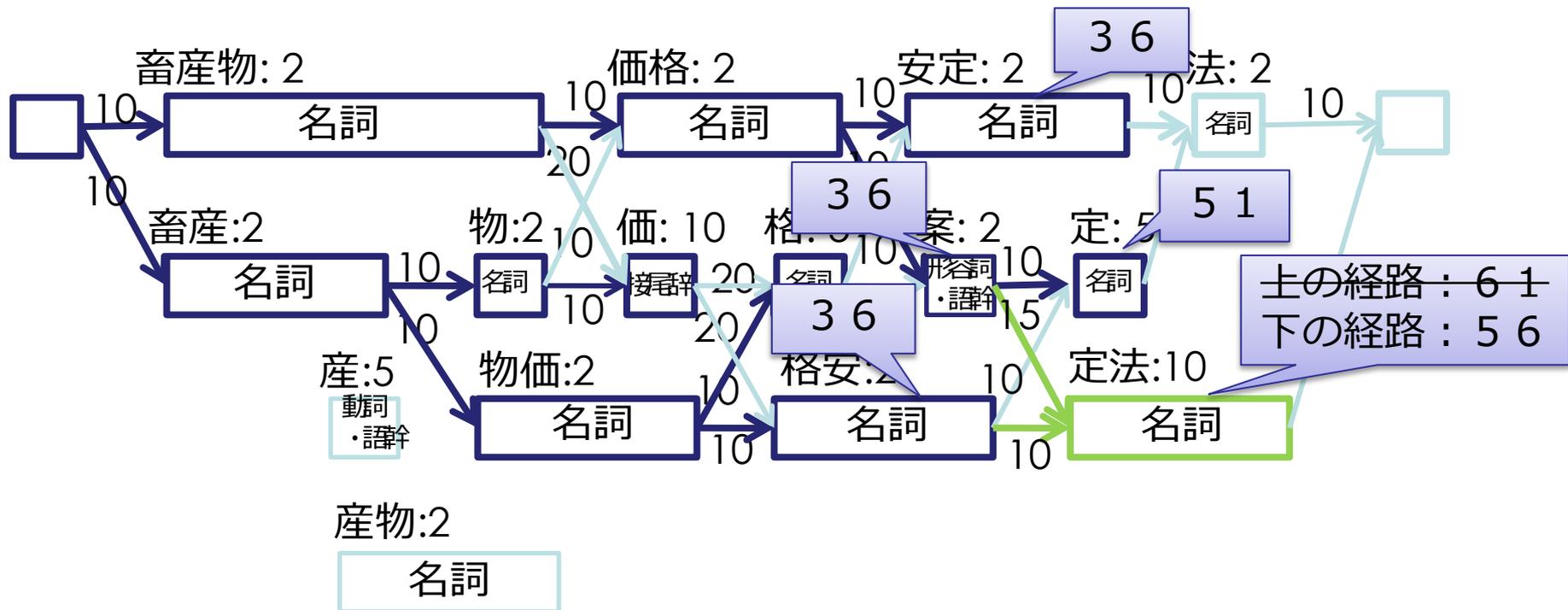
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



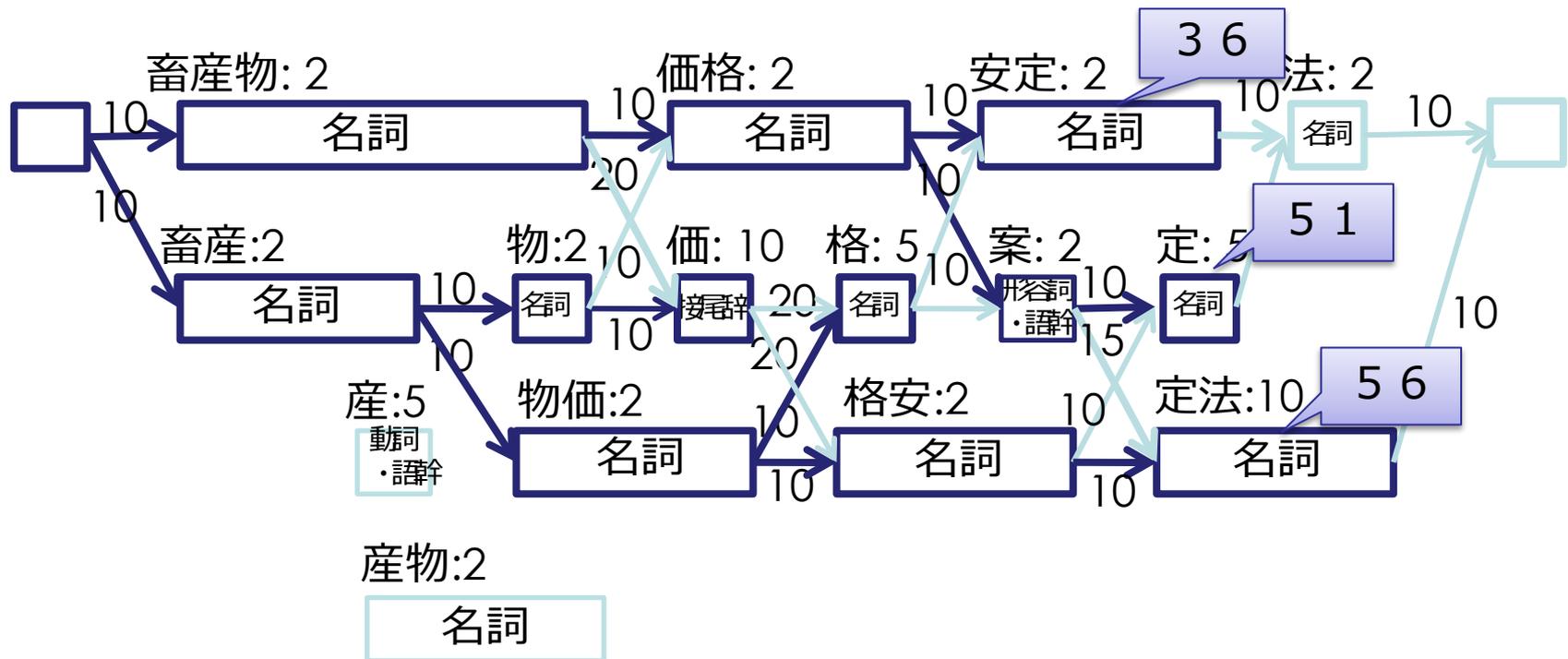
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



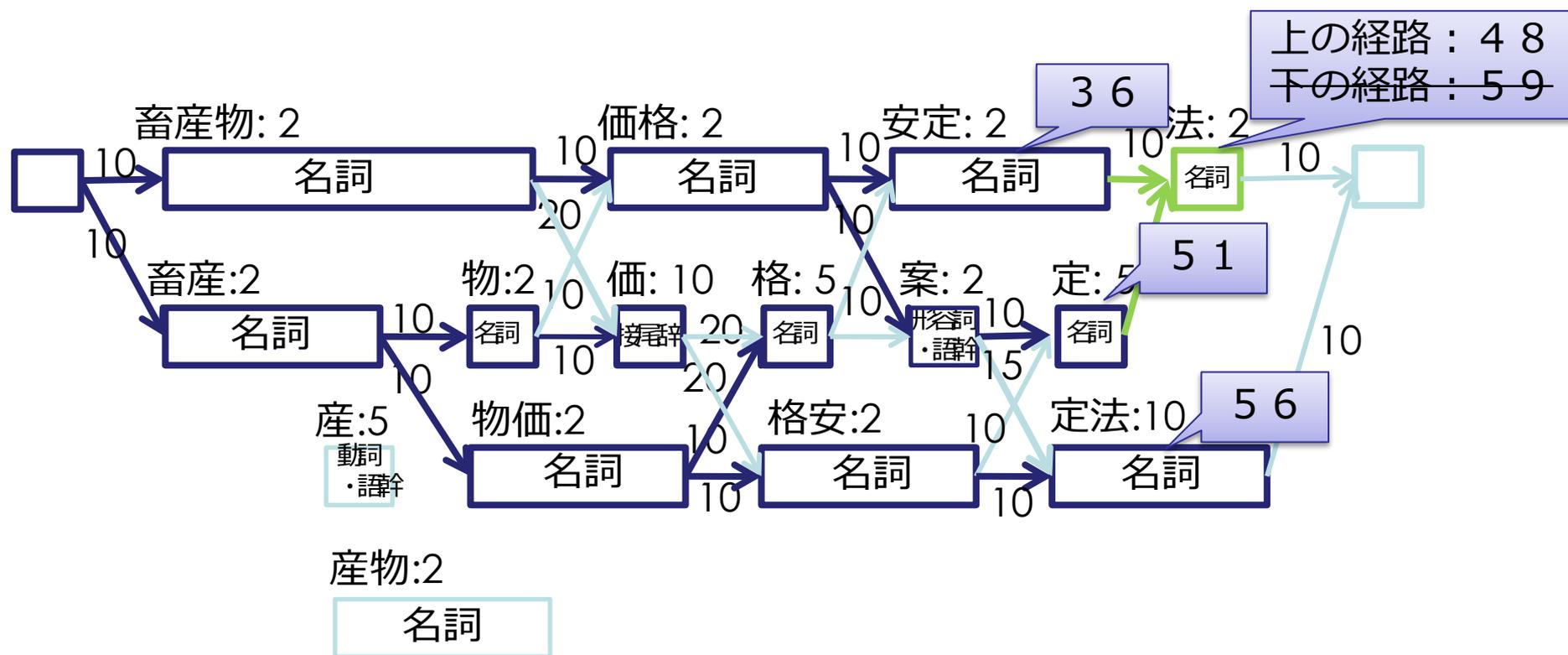
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



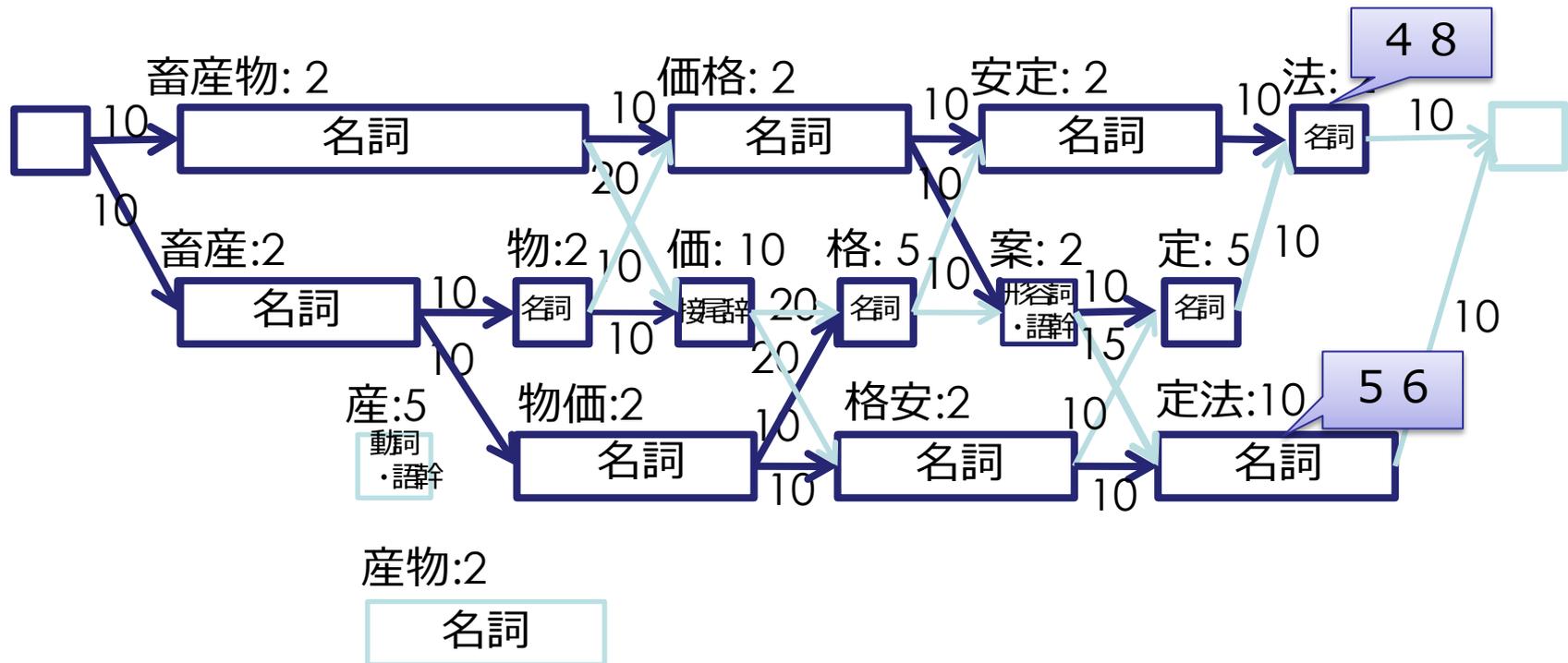
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



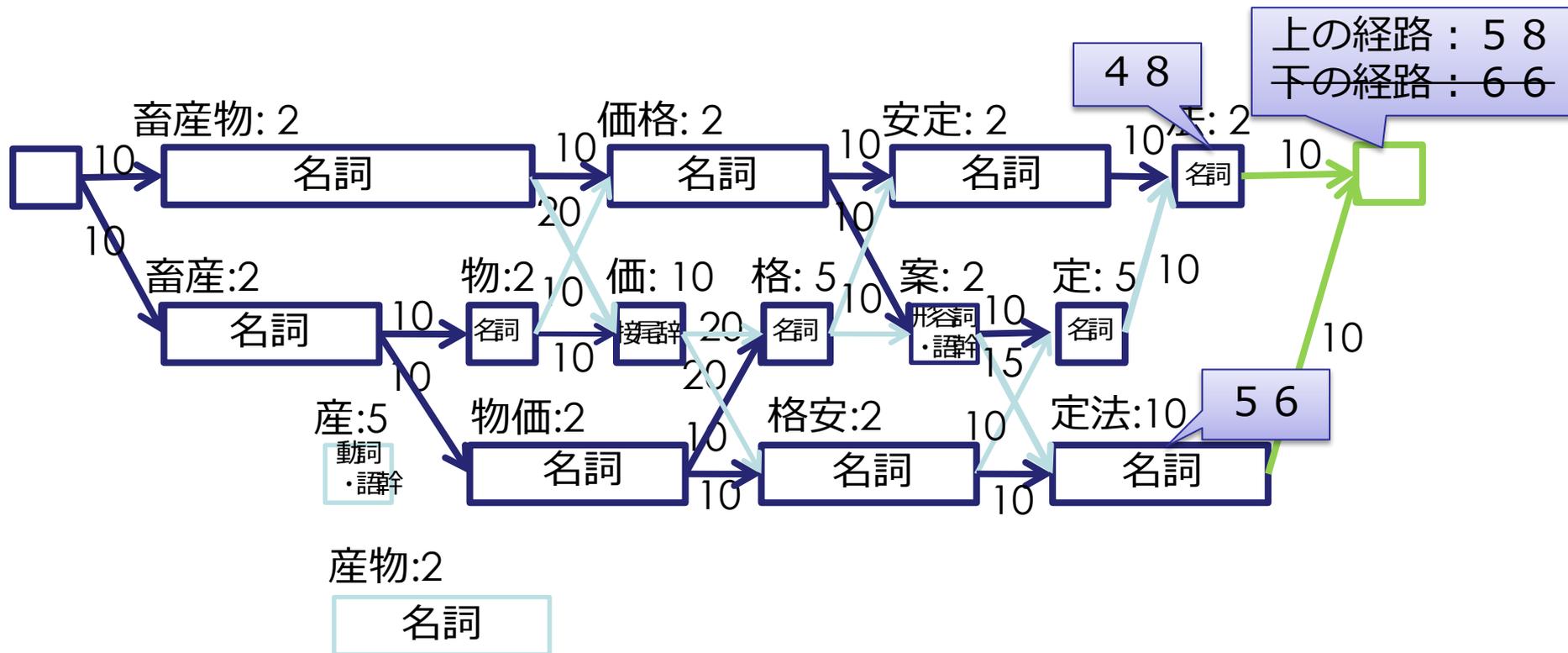
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



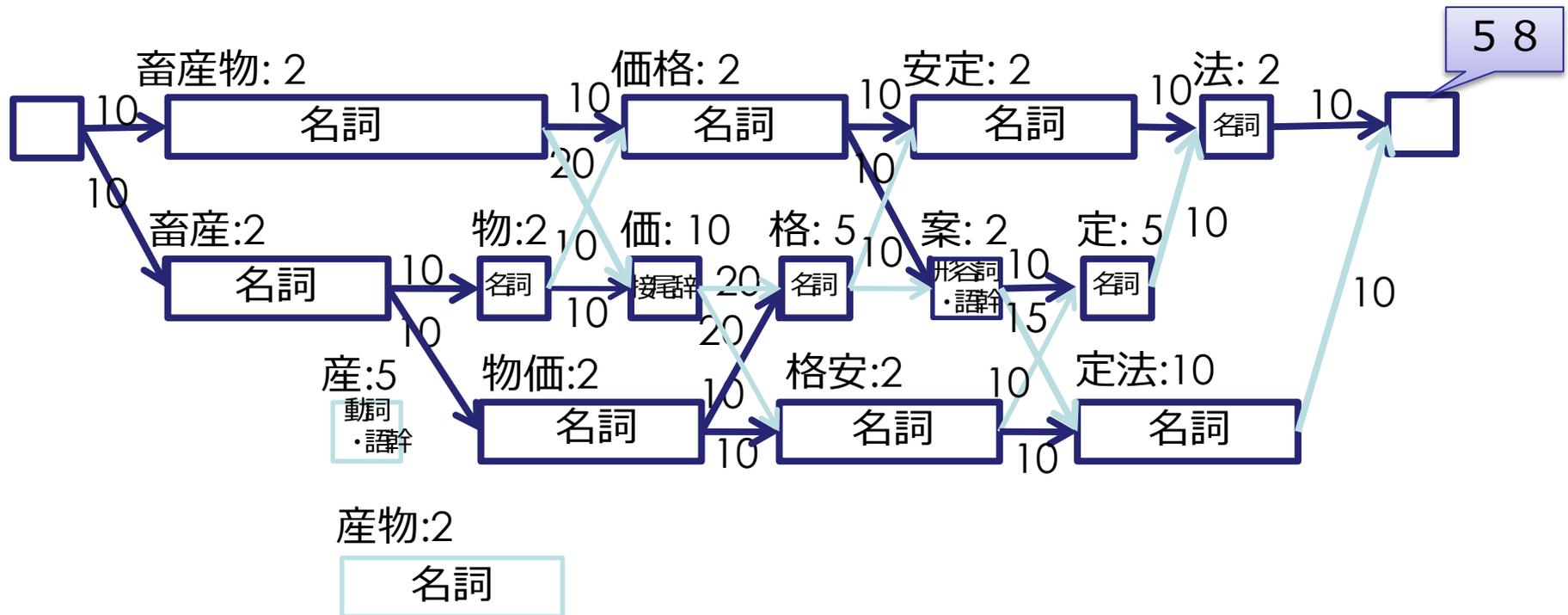
Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



Viterbi アルゴリズム

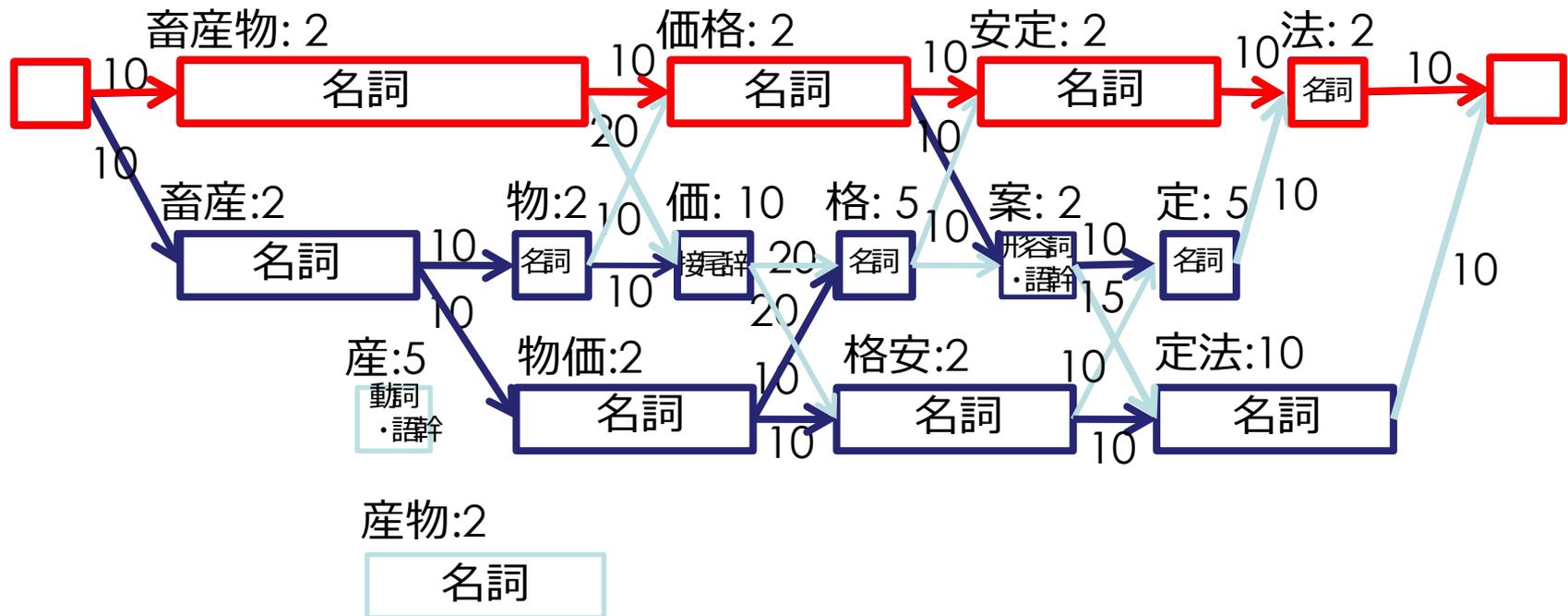
- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する



Viterbi アルゴリズム

- コストが最も低い経路の形態素解析が解
- 経路をどうやって探すか？（全て計算するのは大変！）
- 各状態において、コストが最も低い経路のコストを採用する

後ろから太い枝をたどると経路は唯一...



コストの設定方法

- 人手で与える
 - 90年代初めに試み
 - 手間がかかり、かつ客観的評価が困難
 - Juman (京大NLP研) は人手にこだわり
- **コーパス (大量の言語データ) から学習する**
 - ここでのコーパスは人手で形態素解析したもの

ChaSen形式のコーパス

私	ワタシ	私	名詞-代名詞-一般
の	ノ	の	助詞-連体化
テーマ	テーマ	テーマ	名詞-一般
について	ニツイテ	について	助詞-格助詞-連語
お話し	オハナシ	お話し	名詞-サ変接続
する	スル	する	動詞-自立 サ変・スル 基本形

- MecabやJanomeはCRF (Conditional Random Field) によりモデル化
 - 単語列 X が与えられたとき、その解析結果の形態素列を Y とすると、 X が与えられた時の Y の条件付確率 $P(Y | X)$ を学習
 - $P(Y | X)$ が最大になるような Y を解とする

よく使われるフリーの形態素解析器

- Juman++
 - 辞書を人手で整備→語彙サイズは小さいが解析結果がリッチ
 - 高精度だが速度は若干劣る (Juman++はJumamnに比べ高速化)
 - 手法: RNN(Recurrent Neural Network) 深層学習の一種
- MeCab/**Janome (演習で使用)**
 - アルゴリズムがシンプルでモジュール化もされており、実装が容易、Webでサンプルプログラムがたくさん見つかる
 - 手法: CRF (Conditional Random Field)
- KyTea
 - 特殊なドメインの文書を解析する際は、自分でコーパスを作ってモデル学習するのがいいが、その場合はKyTeaが簡単
 - 手法: SVM (Support Vector Machine)
- Juman/ChaSen
 - Juman++やMeCabに代替わり
 - 手法: HMM (Hidden Markova Model)

結果や精度はコーパスや辞書にも依存

- 形態素・構文情報付き言語コーパス
 - 京都大学テキストコーパス：毎日新聞（原文が有料）
 - 国立国語研究所 書き言葉均衡コーパス（BCCWJ）（有料）
- 辞書：品詞体系が異なる
 - IPA辞書: IPA 品詞体系に基づく辞書
 - ChaSen, MeCab, Janomeが採用
 - JUMAN辞書：Juman辞書
 - Jumanが採用
 - 設定によってMecabでも利用できる
- コストはコーパスから学習
 - **コーパスに現れがちな文の解析は精度が高い**
 - ゼロから学習はあまりやらない（大規模コーパスは基本有料）
 - ドメイン適応したい場合はまず辞書の追加（演習に含まれます）、より高精度を目指すなら学習済みモデルを再学習する

演習 : TextProcessing1.ipynb

Janomeを使った形態素解析

- Pythonの標準ライブラリのみで構成
- プリケーション
 - 辞書などを読み込んだオブジェクト
 - 初期化コストが高いため、同じプログラム内ではこのオブジェクトを使いまわす

```
from janome.token
```

```
t = Tokenizer()
```

```
line = '東京大学は欧米諸国の諸制度に倣った、日本国内で初の近代的な大学として設立された。'
```

```
for token in t.tokenize(line):  
    print(token)
```



東京大学	名詞,固有名詞,組織,***,東京大学,トウキョウダイガク,トーキョーダイガク
は	助詞,係助詞,****,は,ハ,ワ
欧米	名詞,固有名詞,地域,一般,**,欧米,オウベイ,オーベイ
諸国	名詞,一般,****,諸国,シヨコク,シヨコク
の	助詞,連体化,****,の,ノ,ノ
諸	接頭詞,名詞接続,****,諸,シヨ,シヨ
制度	名詞,一般,****,制度,セイド,セイド
に	助詞,格助詞,一般,***,に,ニ,ニ
倣っ	動詞,自立,**,五段・ワ行促音便,連用タ接続,倣う,ナラッ,ナラッ
た	助動詞,***,特殊・タ,基本形,た,タ,タ

Janomeの標準出力形式 例 1 (名詞)

東京大学 名詞,固有名詞,組織,*,*,*,東京大学,トウキョウ
ダイガク,トーキョーダイガク

表層形 : 「東京大学」

品詞 : 名詞

品詞細分類1 : 固有名詞

品詞細分類2 : 組織

品詞細分類3 : *

活用型 : *

活用形 : *

*は登録なし

原形 : 東京大学

読み : ウキョウダイガク

発音 : トーキョーダイガク

IPA品詞体系

音声認識ではこの情報を利用

IPA品詞体系 1

(間投、フィラー、感動詞、記号、形容詞、助詞、助動詞)

品詞ID	分類	例
0	その他,間投	「あ」「ア」のみ
1	フィラー	「えーと」「なんか」など
2	感動詞	「うむ」「お疲れさま」「トホホ」
3	記号,アルファベット	「A-z」
4	記号,一般	「?」「!」「¥」
5	記号,括弧開	「(」「【」など
6	記号,括弧閉	「)」「】」など
7	記号,句点	「。」「.」のみ
8	記号,空白	「 」のみ
9	記号,読点	「,」「,」のみ
10	形容詞,自立	「美しい」「楽しい」
11	形容詞,接尾	「ったらしい」「っぼい」
12	形容詞,非自立	「づらい」「がたい」「よい」

品詞ID	分類	例
13	助詞,格助詞,一般	「の」「から」「を」
14	助詞,格助詞,引用	「と」のみ
15	助詞,格助詞,連語	「について」「とかいう」
16	助詞,係助詞	「は」「こそ」「も」「や」
17	助詞,終助詞	「かしら」「ぞ」「っけ」「わい」
18	助詞,接続助詞	「て」「つつ」「および」「ので」
19	助詞,特殊	「かな」「けむ」「にゃ」
20	助詞,副詞化	「と」「に」のみ
21	助詞,副助詞	「くらい」「なんか」「ばっかり」
22	助詞,副助詞/並立助詞/終助詞	「か」のみ
23	助詞,並立助詞	「とか」「だの」「やら」
24	助詞,連体化	「の」のみ
25	助動詞	「ます」「らしい」「です」

参照: 「形態素解析ツールの品詞体系」 <http://www.unixuser.org/~euske/doc/postag/>

IPA品詞体系2

(接続詞、接頭詞、動詞、副詞、名詞)

品詞ID	分類	例
26	接続詞	「だから」「しかし」
27	接頭詞,形容詞接続	「お」「まっ」
28	接頭詞,数接続	「計」「毎分」
29	接頭詞,動詞接続	「ぶっ」「引き」
30	接頭詞,名詞接続	「最」「総」
31	動詞,自立	「投げる」
32	動詞,接尾	「しまう」「ちゃう」 「願う」
33	動詞,非自立	「しまう」「ちゃう」 「願う」
34	副詞,一般	「あいかわらず」「多分」
35	副詞,助詞類接続	「こんなに」「そんなに」
36	名詞,サ変接続	「インプット」「悪化」
37	名詞,ナイ形容詞語幹	「申し訳」「仕方」
26	接続詞	「だから」「しかし」

品詞ID	分類	例
38	名詞,一般	普通名詞。
39	名詞,引用文字列	「いわく」のみ
40	名詞,形容動詞語幹	「健康」「安易」「駄目」
41	名詞,固有名詞,一般	
42	名詞,固有名詞,人名,一般	
43	名詞,固有名詞,人名,姓	
44	名詞,固有名詞,人名,名	
45	名詞,固有名詞,組織	「株式会社〇〇」
46	名詞,固有名詞,地域,一般	「東京」
47	名詞,固有名詞,地域,国	「日本」
48	名詞,数	「0」「一」
49	名詞,接続詞的	「〇対〇」「〇兼〇」
50	名詞,接尾,サ変接続	「(可視)化」
51	名詞,接尾,一般	「感」「観」「性」
52	名詞,接尾,形容動詞語幹	「的」「げ」「がち」
53	名詞,接尾,助数詞	「個」「つ」「本」「冊」

参照：「形態素解析ツールの品詞体系」 <http://www.unixuser.org/~euske/doc/postag/>

Janomeの標準出力形式 例2 (動詞)

倣っ 動詞,自立,*,*,五段・ワ行促音便,連用夕接続,倣う,ナラッ,ナラッ

表層形：「倣っ」

品詞：動詞

品詞細分類1：自立

品詞細分類2：* *は登録なし

品詞細分類3：*

活用型：五段・ワ行促音便

活用形：連用夕接続

原形：倣う

読み：ナラッ

発音：ナラッ

テキスト解析をする際は、同じ意味の単語でも別の単語として数えられるのを防ぐため、事前に原形に戻すほうが良い場合がある

分かち書き

- 単語分割のみを行うモード
- シンプルなテキスト解析ではよく用いられる

```
line = '東京大学は欧米諸国の諸制度に倣った、日本国内で初の近代的な大学として設立された。'
```

```
print(t.tokenize(line, wakati=True))
```

```
['東京大学', 'は', '欧米', '諸国', 'の', '諸', '制度', 'に', '倣', 'っ', 'た', ',', ' ', '日本', '国内', 'で', '初', 'の', ' ', '近代', '的', 'な', '大学', 'として', '設立', 'さ', 'れ', 'た', '。']
```

未知語に対するユーザ定義辞書の追加

Mathematics and Informatics Center メディアプログラミング入門 2020 山肩洋子 [CC BY-NC-ND](#)

• 未知語とは

- コーパスに一度も現れないもの
- 特殊なドメインでのみ使われる単語 (ex. 「駒場キャンパス」)

• ユーザが定義した辞書を追加することが可能

- csv形式で書いた辞書ファイルを作成して読み込み
- 形式は「表層形,左文脈ID,右文脈ID,生起コスト,品詞,品詞細分類1,品詞細分類2,品詞細分類3,活用型,活用形,原形,読み,発音」
- 左文脈ID、右文脈IDは-1にすれば自動的に割り振られる
- 生起コストは低ければ低いほど出現しやすくなる
(試してみてください)
- MeCabのIPA辞書をダウンロードして、その中に含まれる辞書ファイルから、似たような使われ方をする単語を探し、IDやコストをコピーしてもいい

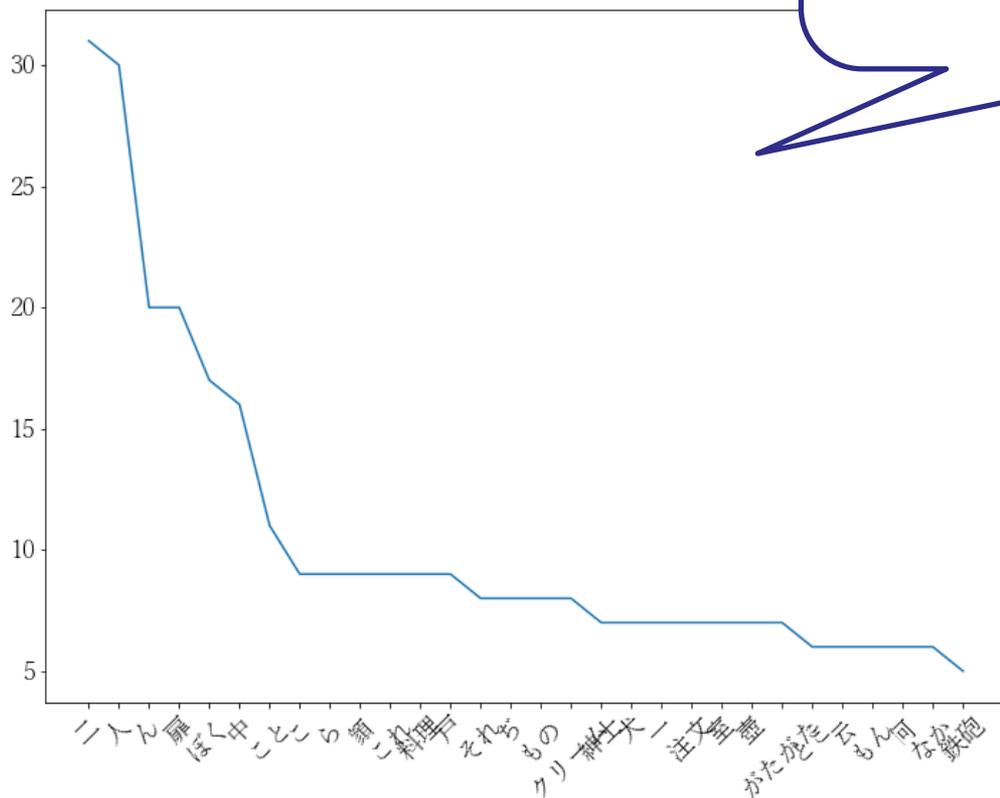
駒場キャンパス,-1,-1,1000,名詞,固有名詞,地名,東大のキャンパス名,*,*,こまば
きゃんぱす,コマバキャンパス,コマバキャンパス

解析の際にはここ以外は見えていないので、
他は好きに設定しても問題は起きない

ただし、辞書は貴重な言語資源なので
汎用性を考えてほしい！

「注文の多い料理店」名詞のヒストグラム

文字が「□」に化けてしまった方は、
掲示板をご覧ください！
'FontChecker.ipynb'
を置きました。



新しい語彙概念：サブワード

- 異なり単語をそのまま語彙とすると語彙サイズが爆発
 - 特にDNNでは語彙サイズを大きくできない
 - 汎用自然言語解析モデルBERTは標準語彙数32,000語
 - 「角川 国語辞典」は75,000語を収録
 - 高頻度語のみに注目すれば語彙サイズを小さくできるが、低頻度語を無視することになる
 - 無視された低頻度語だけでなく、学習コーパスに一度も現れない単語も扱えない（未知語となる）
- Byte pair encoding (BPE)：低頻度語を共通する部分文字列でまとめることで語彙数を削減
 - 「〇〇山」「〇〇川」「〇〇er」「〇〇る」「〇〇太郎」
 - 「〇〇」部分を"##"で表現（英語におけるステミングと同じ）
 - 例「赤門はかつて加賀藩上屋敷があった本郷キャンパスにある。」
→ 「'赤', '##門', 'は', 'かつて', '加賀', '藩', '上', '##屋敷', 'が', 'あつ', 'た', '本郷', 'キャンパス', 'に', 'ある', '。'」
 - 「赤門」「上屋敷」などの語彙は辞書にないため、「赤」と「##門」、「上」と「##屋敷」で代用される

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units. In Proc. of the 54th Annual Meeting of the Association for Computational Linguistics, pp. 1715-1725, 2016. ([PDF](#))

言語モデル

演習 : TextProcessing2.ipynb

言語モデルとは何か？

- 文が与えられると、その文が**どの程度起こりやすいか (=文の生起確率)**を返すモデル
- I 個の単語からなる文を $W = w_1 w_2 \dots w_I$ とすると

$\langle s \rangle$ は文頭、 $\langle /s \rangle$ は文末を表す

w_0 w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8 w_9
 $W = \langle s \rangle$ 単純作業を計算機で行う $\langle /s \rangle$

文の生起確率： $P(W)$

何に使うのか？

最終的に**自然な文を生成**するシステム

音声認識

音声



単純作業を
計算機で...

機械翻訳

英語

Do simple
tasks with ...



単純作業を
計算機で...

かな漢字変換

キーボード入力

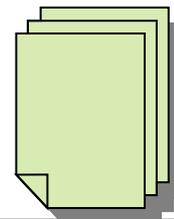
たんじゅんさ
ぎょうをけ...



単純作業を
計算機で...

OCR

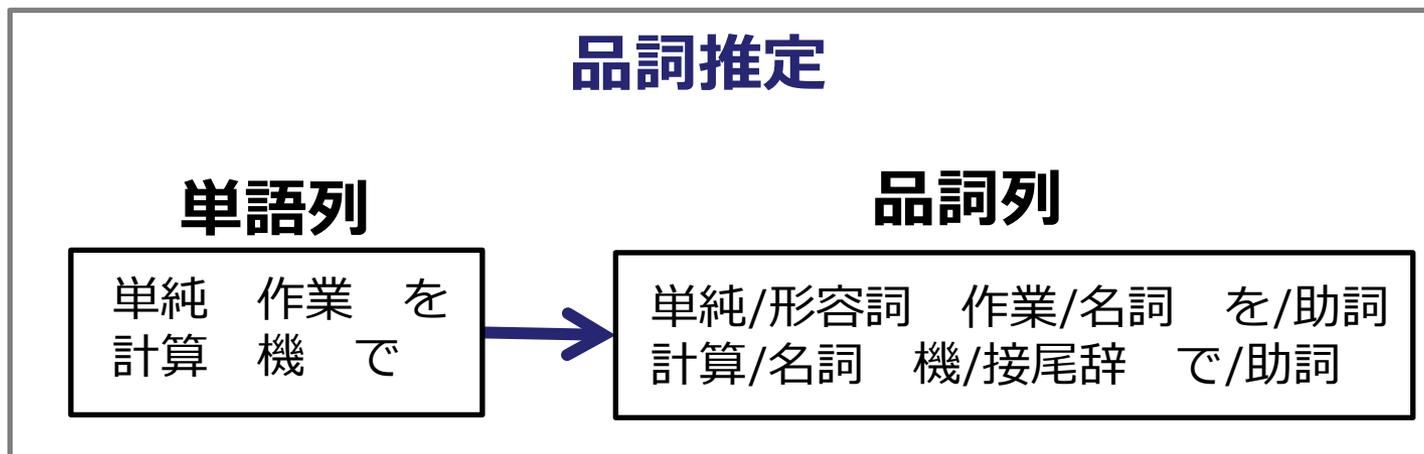
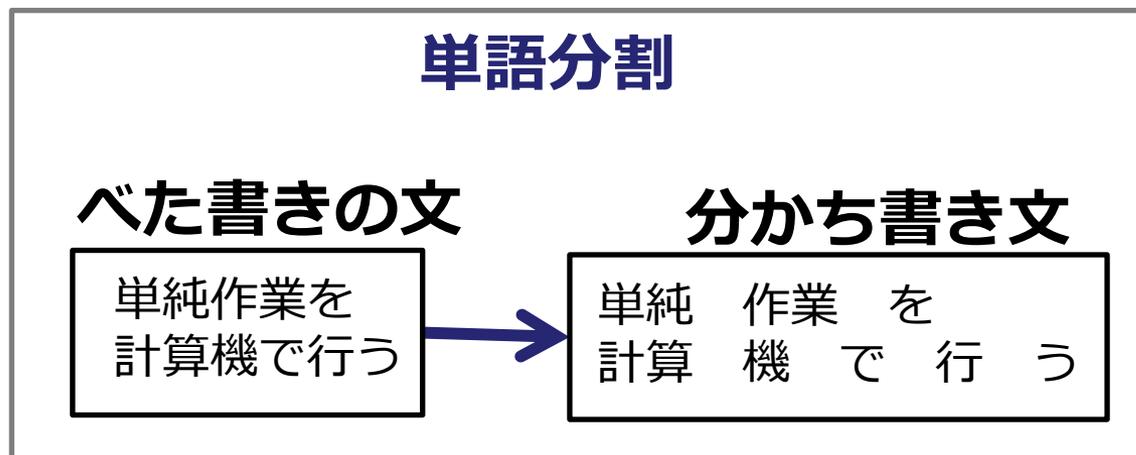
文書



単純作業を
計算機で...

何に使うのか？

文の解析を行うシステム



$P(W)$ をどうやって計算したらいい?

$W = \overset{w_0}{\langle s \rangle} \overset{w_1}{\text{単純}} \overset{w_2}{\text{作業}} \overset{w_3}{\text{を}} \overset{w_4}{\text{計算}} \overset{w_5}{\text{機}} \overset{w_6}{\text{で}} \overset{w_7}{\text{行}} \overset{w_8}{\text{う}} \overset{w_9}{\langle /s \rangle}$

例えばある単語「**作業**」に注目すると...

- 文頭からその単語の直前までの文「 $\langle s \rangle$ 単純」の生起確率を $P(\langle s \rangle \text{ 単純})$ とする
- 「 $\langle s \rangle$ 単純」の後に**作業**が続く事後確率を $P_{LM}(\text{作業} | \langle s \rangle \text{ 単純})$ とすると、単語列 ($\langle s \rangle$ 単純 **作業**) の生起確率 $P(\langle s \rangle \text{ 単純 } \text{作業})$ は

$$P(\langle s \rangle \text{ 単純 } \text{作業}) = P(\langle s \rangle \text{ 単純}) \times P_{LM}(\text{作業} | \langle s \rangle \text{ 単純})$$

これも同じやり方で計算できるよね! ?

文頭から順に計算していくとすると...

$$P(\langle s \rangle \text{ 単純 } \text{作業} \text{ を } \text{計算} \text{ 機 } \text{ で } \text{行} \text{ う } \langle /s \rangle) = P_{LM}(\text{単純} | \langle s \rangle) P_{LM}(\text{作業} | \langle s \rangle \text{ 単純}) P_{LM}(\text{を} | \langle s \rangle \text{ 単純 } \text{作業}) \dots P_{LM}(\langle /s \rangle | \langle s \rangle \text{ 単純 } \text{作業} \text{ を } \text{計算} \text{ 機 } \text{ で } \text{行} \text{ う})$$

式にすると・・・

$$P(\langle s \rangle \text{ 単純作業を計算機で行う } \langle /s \rangle) = \\ P_{LM}(\text{単純} | \langle s \rangle) P_{LM}(\text{作業} | \langle s \rangle \text{ 単純}) P_{LM}(\text{を} | \langle s \rangle \text{ 単純 作業}) \dots \\ P_{LM}(\langle /s \rangle | \langle s \rangle \text{ 単純 作業を計算機で行う})$$

$$P(W) = P_{LM}(w_0) P_{LM}(w_1 | w_0) \dots P_{LM}(w_{N+1} | w_0 \dots w_I)$$

$$= \prod_{i=0}^{I+1} P_{LM}(w_i | w_0 \dots w_{i-1})$$

ある単語 **その直前の単語列**

$\sum_{i=0}^{I+1}$ の掛け算バージョン
 $i=0$ (文頭) から $i=I+1$ (文末) までかけ合わせる

条件付確率はコーパスから学習！

- $P_{LM}(w_i|w_0 \dots w_{i-1})$ は大量の文（コーパス）から学習
- あらゆる単語の w_0, w_1, \dots, w_i の組み合わせについて $P_{LM}(w_i|w_0 \dots w_{i-1})$ を計算しておく → そんなの絶対ムリ！
 - 例え学習コーパスが数億文からなるとしても、
文頭から全く同じ文はほとんどない！
 - 「**彼**は単純作業を計算機で行う」
 - 「**彼女**は単純作業を計算機で行う」
- データがスパース（疎）になる
 $P(\text{単純作業を計算機で行う}) = 1/\text{数億}$

解決策：**直前の数単語**の連結しか考慮しない

解決策：直前の数単語の連結しか考慮しない

$$P(\langle s \rangle \text{ 単純 作業 を 計算機 で 行う } \langle /s \rangle) = \\ P_{LM}(\text{単純} | \langle s \rangle) P_{LM}(\text{作業} | \langle s \rangle \text{ 単純}) P_{LM}(\text{を} | \langle s \rangle \text{ 単純 作業}) \dots \\ P_{LM}(\langle /s \rangle | \langle s \rangle \text{ 単純 作業 を 計算機 で 行う})$$



直前の1単語のみを考慮するとすると...

$$P(\langle s \rangle \text{ 単純 作業 を 計算機 で 行う } \langle /s \rangle) = \\ P_{LM}(\text{単純} | \langle s \rangle) P_{LM}(\text{作業} | \langle s \rangle \text{ 単純}) P_{LM}(\text{を} | \langle s \rangle \text{ 単純 作業}) \dots \\ P_{LM}(\langle /s \rangle | \langle s \rangle \text{ 単純 作業 を 計算機 で 行う})$$



$$P(\langle s \rangle \text{ 単純 作業 を 計算機 で 行う } \langle /s \rangle) \approx \\ P_{LM}(\text{単純} | \langle s \rangle) P_{LM}(\text{作業} | \text{単純}) P_{LM}(\text{を} | \text{作業}) \dots P_{LM}(\langle /s \rangle | \text{う})$$

2単語のつながりのみを考慮するバージョンを単語bi-gramと呼ぶ

統計的単語n-gramによる言語モデル

オリジナルの言語モデル

$$P(W) = P_{LM}(w_0)P_{LM}(w_1|w_0) \dots P_{LM}(w_{I+1}|w_0 \dots w_I)$$

$$= \prod_{i=0}^{I+1} P_{LM}(w_i | w_0 \dots w_{i-1})$$

ある単語 **その直前のすべての単語列**

統計的単語bi-gramによる言語モデル

$$P(W) \cong P_{LM}(w_1|w_0)P_{LM}(w_2|w_1) \dots P_{LM}(w_{I+1}|w_I)$$

$$= \prod_{i=0}^{I+1} P_{LM}(w_i | w_{i-1})$$

ある単語 **その直前の1単語**

直前の文字をいくつまで考慮するか？

- 1-gram (ユニグラム): 履歴を考慮しない

$$P(W) = \prod_{i=0}^{I+1} P_{LM}(w_i)$$

- 2-gram (バイグラム): 直前の1単語からの連結

$$P(W) = \prod_{i=0}^{I+1} P_{LM}(w_i | w_{i-1})$$

- 3-gram (トライグラム): 直前の2単語からの連結

$$P(W) = \prod_{i=0}^{I+1} P_{LM}(w_i | w_{i-2} w_{i-1})$$

$P(\langle s \rangle \text{ 単純 作業 を 計算機 で 行う } \langle /s \rangle) \approx$

$$P_{LM}(\text{単純} | \langle s \rangle \langle s \rangle) P_{LM}(\text{作業} | \langle s \rangle \text{単純}) P_{LM}(\text{を} | \text{単純 作業}) \dots \\ P_{LM}(\text{う} | \text{を 行}) P_{LM}(\langle /s \rangle | \text{行う})$$

直前の文字をいくつまで考慮するか？

- 1-gram (ユニグラム): 履歴を考慮しない

$$P(W) = \prod_{i=0}^{I+1} P_{LM}(w_i)$$

- 2-gram (バイグラム): 直前の1単語からの連結

$$P(W) = \prod_{i=0}^{I+1} P_{LM}(w_i | w_{i-1})$$

- 3-gram (トライグラム): 直前の2単語からの連結

$$P(W) = \prod_{i=0}^{I+1} P_{LM}(w_i | w_{i-2} w_{i-1})$$

$$P(W) = \prod_{i=0}^{I+1} P_{LM}(w_i | w_{i-n+1} \cdots w_{i-1})$$

最尤推定による統計的単語1-gramモデルの計算

- 直前の文字は考慮しない
- ある単語 w_i がコーパス中にどの程度の頻度で発生するか？

$$P_{LM}(w_i) = \frac{c_L(w_i)}{\sum_{\tilde{w}} c_L(\tilde{w})}$$

← 学習コーパスにおける
 w_i の数

← 学習コーパスの総単
語数

- コーパスの総単語数=1,000,000、「単純」=200の場合

$$P_{LM}(\text{単純}) = \frac{200}{1,000,000} = 0.0002$$

計算してみよう！統計的1-gramモデル

1-gram確率を計算せよ.

ただし, 下記4文を学習コーパスとする

単純 作業 を 計算機 で 行う </s>
単純 作業 は 退屈 だ </s>
単純 な 問題 </s>
今日 は いい 天気 だ </s>

単語は全部で23個

$$P_{LM}(\text{単純}) = 3/23$$

$$P_{LM}(\text{作業}) = 2/23$$

$$P_{LM}(\text{を}) = 1/23$$

$$P_{LM}(\text{計算機}) = 1/23 \quad P_{LM}(\text{で}) = 1/23$$

$$P_{LM}(\text{行}) = 1/23$$

$$P_{LM}(\text{う}) = 1/23 \quad P_{LM}(\text{</s>}) = 4/23$$

$$P(\text{単純 作業 を 計算機 で 行う </s>}) = (3*2*4)/(23^8)$$

最尤推定による統計的単語 2-gramモデルの計算

- ある単語 w_{i-1} の後にある単語 w_i が続く確率

$$P_{LM}(w_i|w_{i-1}) = \frac{c_L(w_{i-1}w_i)}{c_L(w_{i-1})}$$

学習コーパス中で
 $w_{i-1} w_i$ という2単語が
並んで現れた数



学習コーパス中の
 w_{i-1} の総数



- 「単純」の数=200、「単純 作業」の数=40の場合

$$P_{LM}(\text{作業}| \text{単純}) = \frac{40}{200} = 0.2$$

計算してみよう！統計的2-gramモデル

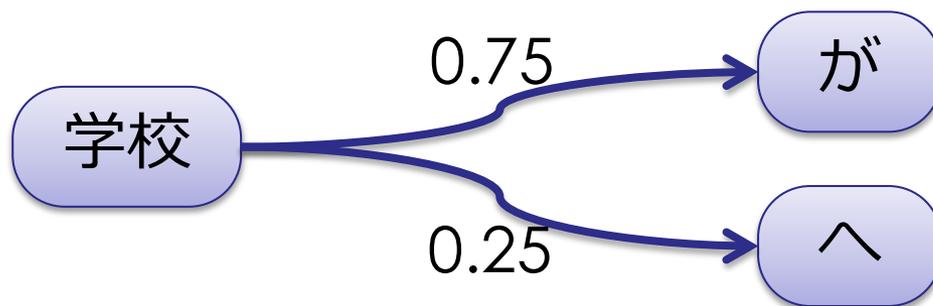
学習コーパス

私/は/学校/へ/行っ/た/。
私/は/学校/が/好き/だ/。
私/は/学校/が/楽しい/。
学校/が/私/の/オアシス/だ/。

「学校」の出現回数： $C(\text{“学校”})=4$
「学校が」の出現回数： $C(\text{“学校が”})=3$
「学校へ」の出現回数： $C(\text{“学校へ”})=1$

$$P(\text{“が”}|\text{“学校”}) = \frac{C(\text{“学校が”})}{C(\text{“学校”})} = \frac{3}{4} = 0.75$$

$$P(\text{“へ”}|\text{“学校”}) = \frac{C(\text{“学校へ”})}{C(\text{“学校”})} = \frac{1}{4} = 0.25$$



計算してみよう！統計的3-gramモデル

学習コーパス

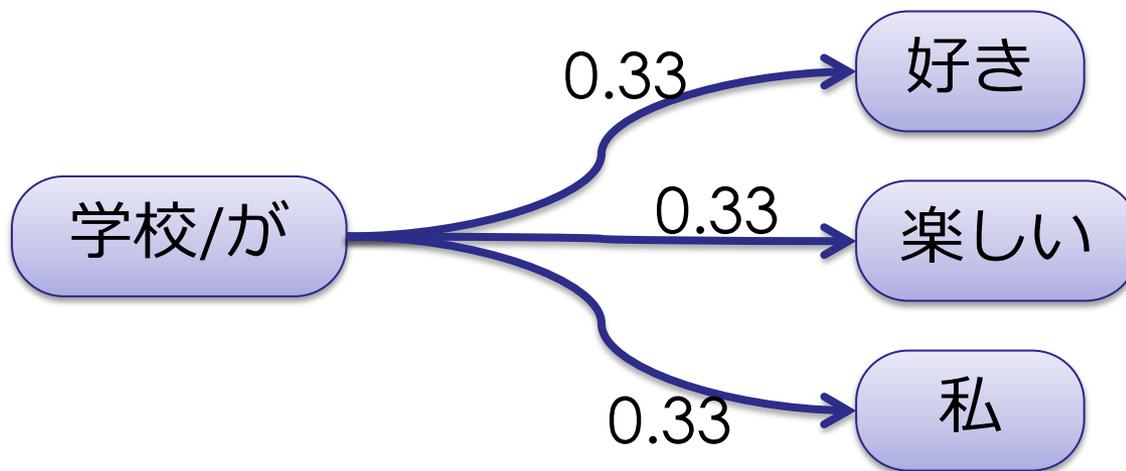
私/は/学校/へ/行っ/た/。
私/は/学校/が/好き/だ/。
私/は/学校/が/楽しい/。
学校/が/私/の/オアシス/だ/。

「学校が」の出現回数： $C(\text{“学校が”})=3$

「学校が好き」の出現回数：

$C(\text{“学校が好きだ”})=1$

$$P(\text{“好き”}|\text{“学校/が”}) = \frac{C(\text{“学校/が/好き”})}{C(\text{“学校/が”})} = \frac{1}{3} = 0.33$$



頻度 0 の問題点

単純 作業 を 計算機 で 行う </s>
単純 作業 は 退屈 だ </s>
単純 な 問題 </s>
今日 は いい 天気 だ </s>

$P_{LM}(\text{だ}|\text{単純})$ は0になってしまう…

- 学習サイズとモデルの複雑さのジレンマ
 - nは大きければ大きいほど表現力は高い
 - nが大きいと学習データにサンプルがない可能性が上がる
- 複数のn-gramを重みをつけて足し合わせる方法もある
(線形補完)

$$P_{\text{uni-gram}}(w_i|w_{i-1}) = \alpha P_{LM}(w_i) + \beta P_{\text{bi-gram}}(w_i|w_{i-1}) + (1 - \alpha - \beta) P_{\text{tri-gram}}(w_i|w_{i-2}w_{i-1})$$

uni-gram の重み **uni-gram** 確率 **bi-gram** の重み **bi-gram** 確率 **tri-gram** の重み **tri-gram** 確率

演習 : TextProcessing2.ipynb

NLTK (Natural Language Tool-Kitを使った統計的n-gram

- “text/miyazawa_wakati.txt”には宮沢賢治の作品147品の本文をjanomeを使って分かち書きして収録

2. n-gram生成 (uni-gram, bi-gram, tri-gramを生成)

入力 : 「その明け方の空の下、ひるの鳥でもゆかない高いところを
するどい霜のかけらが風に流されてサラサラ」

unigram [('<BOP>', ('その'), ('明け方'), ('の'), ('空'), ('の'), ('下'), ('、'), ('ひる'), ('の'), ('鳥'), ('で'), ('も'), ('ゆか'), ('ない'), ('高い'), ('ところ'), ('を'), ('するどい'), ('霜'), ('の'), ('かけ'), ('ら'), ('が'), ('風'), ('に'), ('流さ'), ('れ'), ('て'), ('サラサラ'), ('サラサラ'), ('南'), ('の'), ('ほう'), ('へ'), ('とん'), ('で'), ('ゆき'), ('まし'), ('た'), ('。'), ('<EOP>'))]

bigram [('<BOP>', 'その'), ('その', '明け方'), ('明け方', 'の'), ('の', '空'), ('空', 'の'), ('の', '下'), ('下', '、'), ('、', 'ひる'), ('ひる', 'の'), ('の', '鳥'), ('鳥', 'で'), ('で', 'も'), ('も', 'ゆか'), ('ゆか', 'ない'), ('ない', '高い'), ('高い', 'ところ'), ('ところ', 'を'), ('を', 'するどい'), ('するどい', '霜'), ('霜', 'の'), ('の', 'かけ'), ('かけ', 'ら'), ('ら', 'が'), ('が', '風'), ('風', 'に'), ('に', '流さ'), ('流さ', 'れ'), ('れ', 'て'), ('て', 'サラサラ'), ('サラサラ', 'サラサラ'), ('サラサラ', '南'), ('南', 'の'), ('の', 'ほう'), ('ほう', 'へ'), ('へ', 'とん'), ('とん', 'で'), ('で', 'ゆき'), ('ゆき', 'まし'), ('まし', 'た'), ('た', '。'), ('。', '<EOP>')]

trigram [('<BOP>', 'その', '明け方'), ('その', '明け方', 'の'), ('明け方', 'の', '空'), ('の', '空', 'の'), ('空', 'の', '下'), ('の', '下', '、'), ('下', '、', 'ひる'), ('、', 'ひる', 'の'), ('ひる', 'の', '鳥'), ('の', '鳥', 'で'), ('鳥', 'で', 'も'), ('で', 'も', 'ゆか'), ('も', 'ゆか', 'ない'), ('ゆか', 'ない', '高い'), ('ない', '高い', 'ところ'), ('高い', 'ところ', 'を'), ('ところ', 'を', 'するどい'), ('を', 'するどい', '霜'), ('するどい', '霜', 'の'), ('霜', 'の', 'かけ'), ('の', 'かけ', 'ら'), ('かけ', 'ら', 'が'), ('ら', 'が', '風'), ('が', '風', 'に'), ('風', 'に', '流さ'), ('に', '流さ', 'れ'), ('流さ', 'れ', 'て'), ('れ', 'て', 'サラサラ'), ('て', 'サラサラ', 'サラサラ'), ('サラサラ', 'サラサラ', '南'), ('サラサラ', '南', 'の'), ('南', 'の', 'ほう'), ('の', 'ほう', 'へ'), ('ほう', 'へ', 'とん'), ('へ', 'とん', 'で'), ('とん', 'で', 'ゆき'), ('で', 'ゆき', 'まし'), ('ゆき', 'まし', 'た'), ('まし', 'た', '。'), ('た', '。', '<EOP>')]

2.2 n-gramの出現回数

uni-gram頻度
 ジョバンニ: 205

bi-gram頻度
 ジョバンニ

->	は: 123
->	が: 27
->	の: 21
->	も: 10
->	に: 6
->	、: 5
->	さん: 4
->	を: 3
->	たち: 3
->	と: 1
->	や: 1
->	まで: 1

合計

uni-gram頻度	->	玄: 1	->	胸: 1	
ジョバンニは	->	立つ: 1	->	びつくり: 1	
->	、: 33	->	高く: 1	->	困: 1
->	思は: 7	->	われ: 1	->	たしかに: 1
-> 合計	まるで: 5	->	帽子: 1	->	こんな: 1
->	思ひ: 5	->	せ: 1	->	首: 1
->	まつ: 5	->	なんとも: 1	->	坊: 1
->	もう: 4	->	走り: 1	->	川下: 1
->	すぐ: 2	->	ぢ: 1	->	生: 1
->	その: 2	->	町: 1	->	熱: 1
->	何: 2	->	眼: 1	->	どうしても: 1
->	俄: 2	->	一: 1	->	だんだん: 1
->	窓: 2	->	叫び: 1	->	もうす: 1
->	橋: 2	->	まだ: 1	->	あぶなく: 1
->	なんだか: 2	->	なぜ: 1	->	そつ: 1
->	云: 2	->	どんどん: 1	->	自分: 1
->	また: 2	->	いきなり: 1	->	唇: 1
->	勢: 1	->	みんな: 1	->	力強く: 1
->	手: 1	->	わくわく: 1	->	叫ん: 1
->	拾: 1	->	かすか: 1		
->	おじぎ: 1	->	それ: 1		
->	靴: 1	->	(: 1		

3. 言語モデルの学習

```
from nltk.lm import Vocabulary
from nltk.lm.models import MLE

# 読み込んだ小説集の語彙（異なり単語）を収集
# Vocabularyは1次元のリストを受け取るが、wordsは2次元のリストなので、
# wordsを内包表記で2次元から1次元に変換してからVocabularyに渡しています
vocab = Vocabulary([item for sublist in words for item in sublist])

# 語彙の一覧を表示させたいなら下2行のコメントを有効にする
# for v in sorted(vocab.counts):
#     print('%t{:s}'.format(v))

print('Vocabulary size: ' + str(len(vocab))) # 語彙サイズ（単語の種類数）

text_trigrams = [ngrams(word, 3) for word in words] # tri-gramを計算

n = 3
lm = MLE(order = n, vocabulary = vocab) # 最尤推定法 (Maximum Likelihood Estimation)による言語モデルの準備
lm.fit(text_trigrams) # 上で計算したtri-gramを使って言語モデルを学習
```

Vocabulary size: 22793

4. 統計的n-gramの活用

['ジヨバンニ', 'は']に続く単語のtri-gram確率

すべて足すと1になる

、: 0.268293	立つ: 0.008130	胸: 0.008130
思は: 0.056911	高く: 0.008130	びつくり: 0.008130
まるで: 0.040650	われ: 0.008130	困: 0.008130
思ひ: 0.040650	帽子: 0.008130	たしかに: 0.008130
まつ: 0.040650	せ: 0.008130	こんな: 0.008130
もう: 0.032520	なんとも: 0.008130	首: 0.008130
すぐ: 0.016260	走り: 0.008130	坊: 0.008130
その: 0.016260	ぢ: 0.008130	川下: 0.008130
何: 0.016260	町: 0.008130	生: 0.008130
俄: 0.016260	眼: 0.008130	熱: 0.008130
窓: 0.016260	一: 0.008130	どうしても: 0.008130
橋: 0.016260	叫び: 0.008130	だんだん: 0.008130
なんだか: 0.016260	まだ: 0.008130	もうす: 0.008130
云: 0.016260	なぜ: 0.008130	あぶなく: 0.008130
また: 0.016260	どんどん: 0.008130	そつ: 0.008130
勢: 0.008130	いきなり: 0.008130	自分: 0.008130
手: 0.008130	みんな: 0.008130	唇: 0.008130
拾: 0.008130	わくわく: 0.008130	力強く: 0.008130
おじぎ: 0.008130	かすか: 0.008130	叫ん: 0.008130
靴: 0.008130	それ: 0.008130	
玄: 0.008130	(: 0.008130	

4.2 ランダム文生成

- context = ['ジヨバンニ', 'は']に続く単語をtri-gramからランダムに出力
- さらに直前の2単語から次の単語を次々とつないでいく
- あるtri-gramが選ばれる確率は、そのtri-gram確率に依存
- 実行するたびに新しい文が生成

['ジヨバンニ', 'は', '立つ', 'て', 'ある', '。']

['ジヨバンニ', 'は', 'だんだん', '大きく', 'なっ', 'た', 'の', 'です', '。']

['ジヨバンニ', 'は', '自分', 'で', '光る', 'やつ', 'な', 'ん', 'だ', 'ねえ', '。']

['ジヨバンニ', 'は', '思は', 'ず', '二', '人', '一緒', 'に', '念', 'を', '入れ', 'て', '行く', 'ん', 'だ', '。']

['ジヨバンニ', 'は', 'だんだん', '言', 'も', '粗末', 'に', 'なっ', 'て', 'から', '、', 'こんな', 'こと', 'を', 'きか', 'ない', 'こども', 'ら', 'の', '芸術', 'は', '少年', '少女', '期', 'の', '雨', 'の', '中', 'だ', 'から', '<EOP>']

5. 文の生起確率 $P(W)$ の算出

$$P(W) \approx \prod_{i=0}^{I+1} P_{LM}(w_i | w_{i-1})$$

- 文 W に対して、tri-gramを文頭からかけ合わせていく
- 途中1個でも0があると $P(W)$ は常にゼロになる！
 - コーパスが十分大きければそんなことは起こらないはず！
 - どんな文 W に対しても $P(W)$ 0にならないよう、コーパスに一度も現れないtri-gramにも微小な確率を付与しておく
(演習では 10^{-8} としている)

こちらの圧勝！

実験結果：

$P(\text{'ジヨバンニは何げなく答えました。'}) = 3.12480 \times 10^{-5}$

$P(\text{'何げなくジヨバンニは答えました。'}) = 6.65888 \times 10^{-25}$

- 使用できるフォントは環境によって異なる
 - Windows: 'Yu Mincho'など
 - MacOS: 'AppleGothic'など
 - Colaboratory: 'IPAGothic'をインストールして使用
- 皆さんの環境で使用できるフォント
 - FontChecker.ipynbで使用できるフォントを調べる
 - フォントが一部「□」になってしまう
 - キャッシュ(拡張子が.cacheのもの) を削除