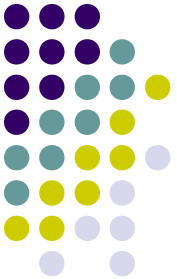
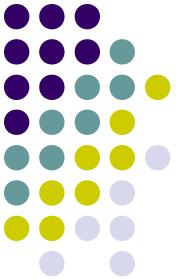


大規模テキスト処理を支える形態素解析技術

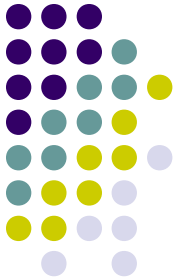
Google 株式会社　　工藤 拓

第80回 人工知能学会 知識ベースシステム研究会 (SIG-KBS)

自己紹介



- 2003年: NAIST 博士後期課程修了
  - 統計的自然言語処理
  - 機械学習
  - データマイニング
- 2004年: NTTコミュニケーション科学基礎研究所入所　リサーチアソシエイト
  - グラフ構造に対する機械学習手法
- 2005年~ Google株式会社ソフトウェアエンジニア
  - Web検索 (サーチクォリティーチーム)



目次

- 形態素解析の技術 (アカデミック)
  - 辞書引きのアルゴリズム、データ構造
  - 曖昧性の解消
  - 今後の課題
- MeCabの開発裏話 (ソフトウェア開発)
  - 歴史
  - 設計方針
  - 汎用テキスト変換ツールとしての MeCab
- これから

形態素解析

- 文を単語に区切り、品詞を同定する処理
  - 全文検索 文書分類 テキストマイニング
- 以下の３つの処理
  - 単語への分かち書き(tokenization)
  - 活用語処理(stemming, lemmatization)
  - 品詞同定(part-of-speech tagging)

```
すもも  名詞,一般,*,*,*,*,すもも,スモモ,スモモ
も      助詞,係助詞,*,*,*,*,も,モ,モ
もも    名詞,一般,*,*,*,*,もも,モモ,モモ
も      助詞,係助詞,*,*,*,*,も,モ,モ
もも    名詞,一般,*,*,*,*,もも,モモ,モモ
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
。      記号,句点,*,*,*,*,。,。,。
```

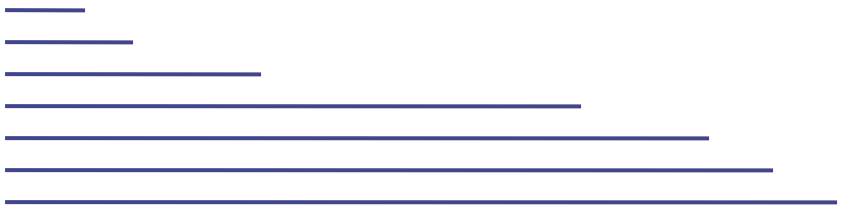
形態素解析の技術

- 基本的な処理: 辞書から単語を引いて、与えられた文と照合し、最も自然な単語列を求める
  - 辞書引き
    - 入力文は単語毎に区切られていない
    - どの文字列を辞書引きするか自明ではない
  - 曖昧性の解消
    - すべての可能な単語の組合せから(何らかの基準で)最適な単語列を発見する
    - 基準の定義

日本語処理のための辞書の要件

- 単語の区切りが明確でないので、先頭から何文字までが単語なのかわからない

奈良先端科学技術大学院大学情報科学研究科

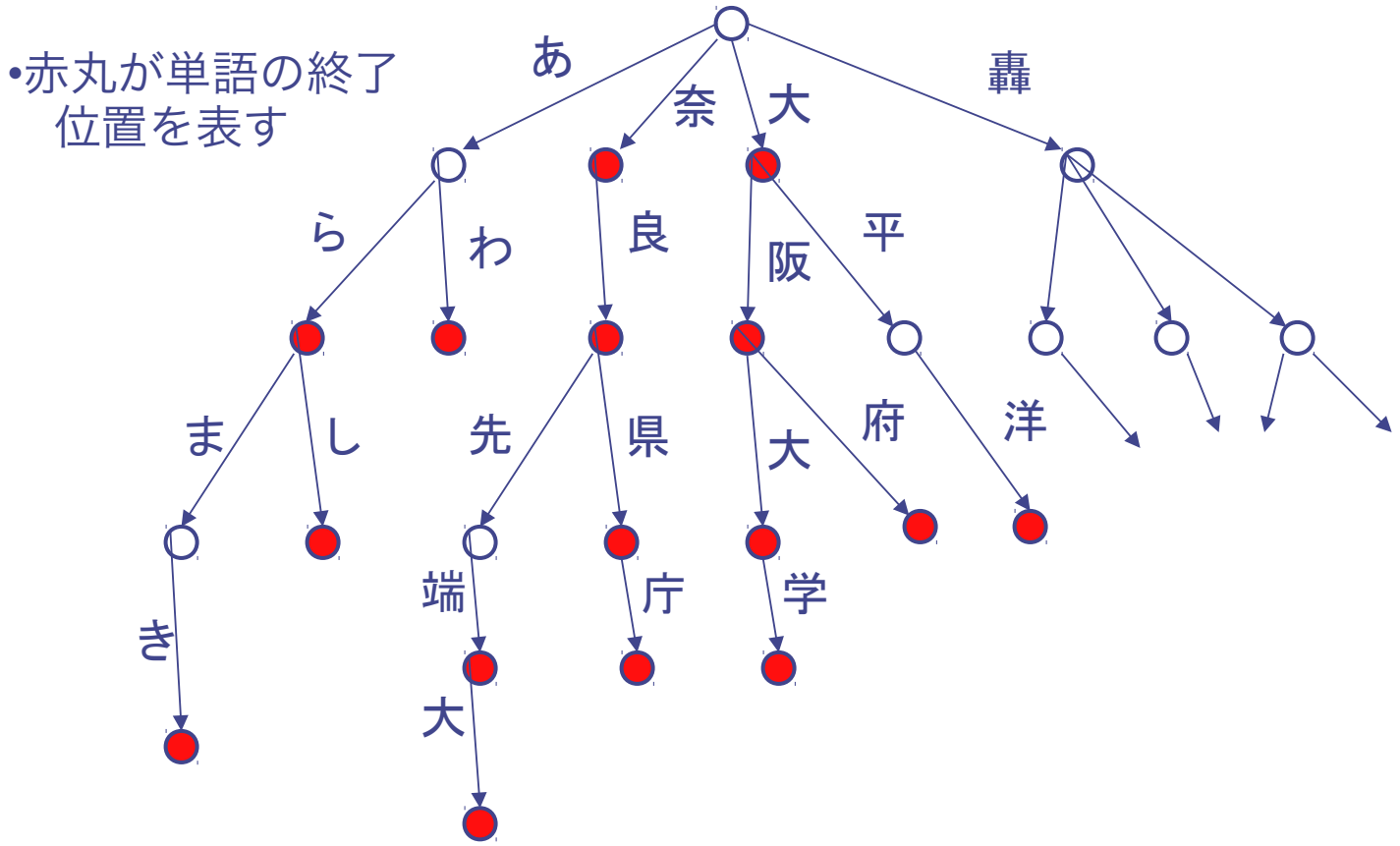


- 単純な方法(hash)だと $(文長)^2$ の辞書引きが発生!

```
$str = "奈良先端科学技術大学院大学情報科学研究科";
 for (my $i = 0; $i < length($str); ++$i) {
    for (my $j = 1; $j < length($str) - $i; ++$j) {
        my $key = substr($str, $i, $j);
        if (defined $dic{$key}) {
            ...;
        ...
```

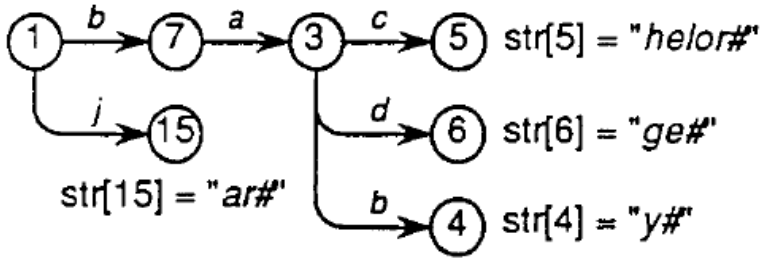
ハッシュデータベース、 RDB は辞書として使えない

辞書検索のためのデータ構造：TRIE



- 対象文字列の先頭から文字を順番にたどるだけ
- 辞書引き終了のタイミングが自動的にわかる
- 入力文字列の長さに比例した時間 O(文長) で探索が可能
- さまざまな実装

Double-Array(ダブル配列) TRIE



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
BASE	4	0	1	-15	-1	-12	1	0	0	0	0	0	0	0	-9
CHECK	0	0	7	3	3	3	1	0	0	0	0	0	0	0	1

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
TAIL	h	e	l	o	r	#	?	?	a	r	#	g	e	#	y	#

POS=17

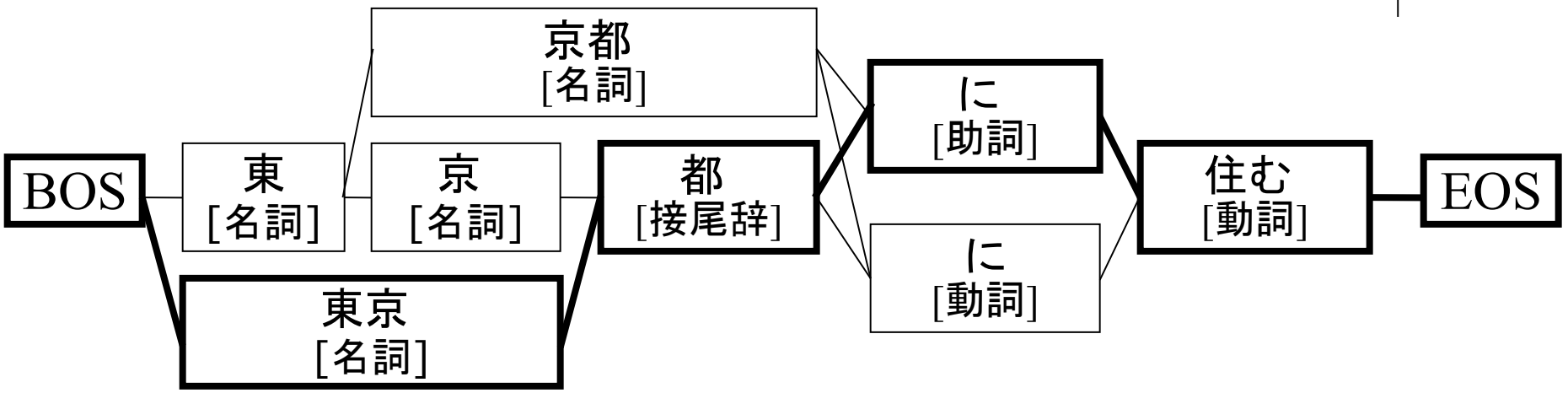
An efficient implementation of Trie Structures, Aoe et al 92 より引用

{#=-1, a=2, b=3, c=4...}

```
int n = 1
for (int i = 0; i < strlen(key); ++i) {

  int k = BASE[n] + charcode(key[i]);
  if (CHECK[k] != n) break;

  // 見つかった!
  if (BASE[k] < 0)
     printf ("%d\n",  - BASE[k]);

  n = k;
}
```

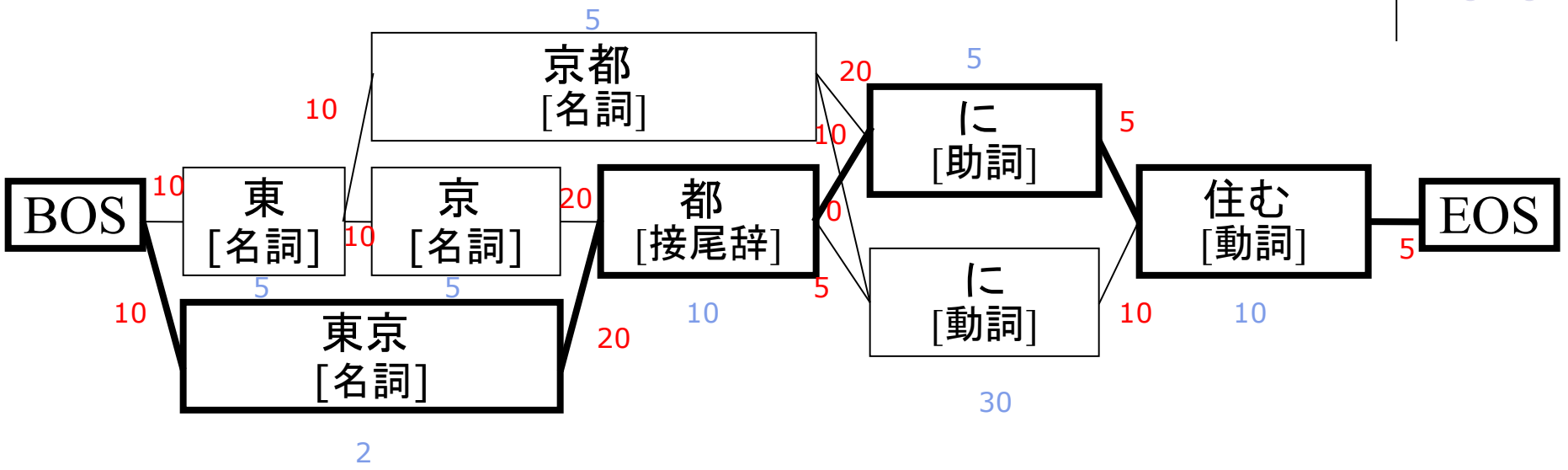
- MeCabに採用 (後に ChaSen も)
  - 利点: (知る限り)最も高速
  - 欠点: 辞書サイズが大きい, 構築が面倒

曖昧性の解消



- 規則(ヒューリスティックス)に基づく手法 (80年代)
  - 最長一致:　長い単語を優先　(KAKASI)
  - 分割数最小:　文全体の単語の数を最小にする候補
  - 文節数最小:　文全体の文節数を最小にする候補
- 多くの場合曖昧性を解決できない

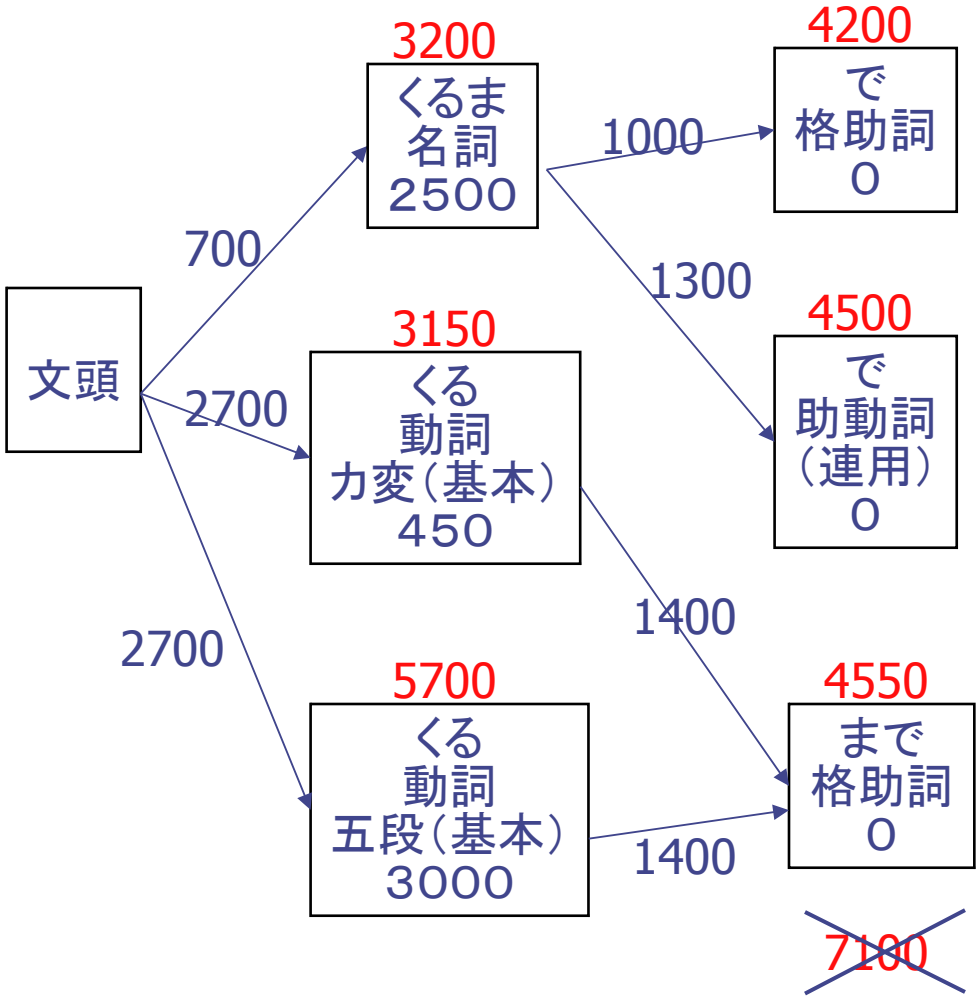
最小コスト法



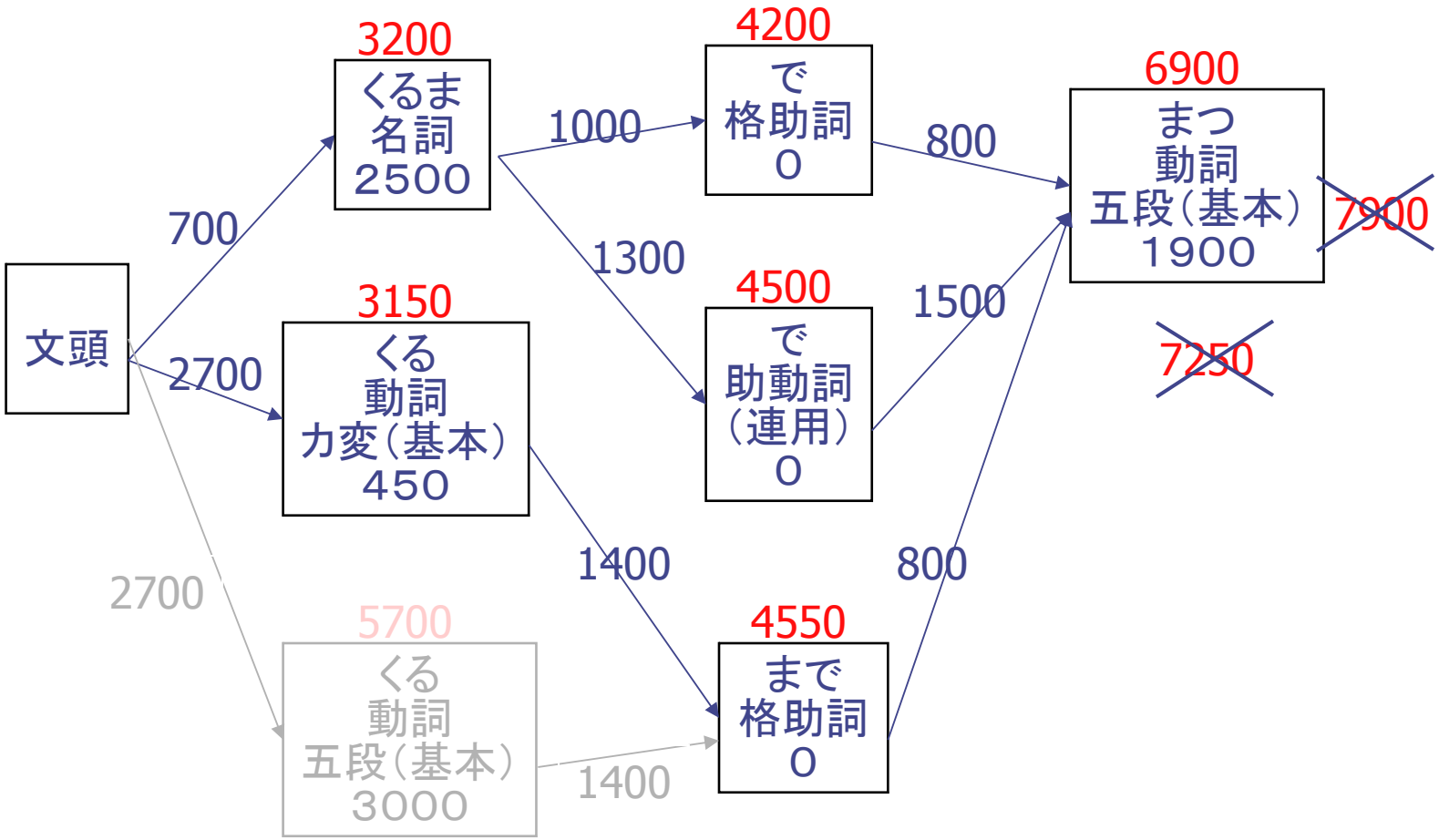
連接コスト: 二つの単語のつながりやすさ
生起コスト: 一つの単語の出現しやすさ

- 連接コストと生成コストの和が最小になる解
- コストはなんらかの方法で決定 (後述)
- Viterbi アルゴリズム (動的計画法の一種) で O(文長) で探索可能

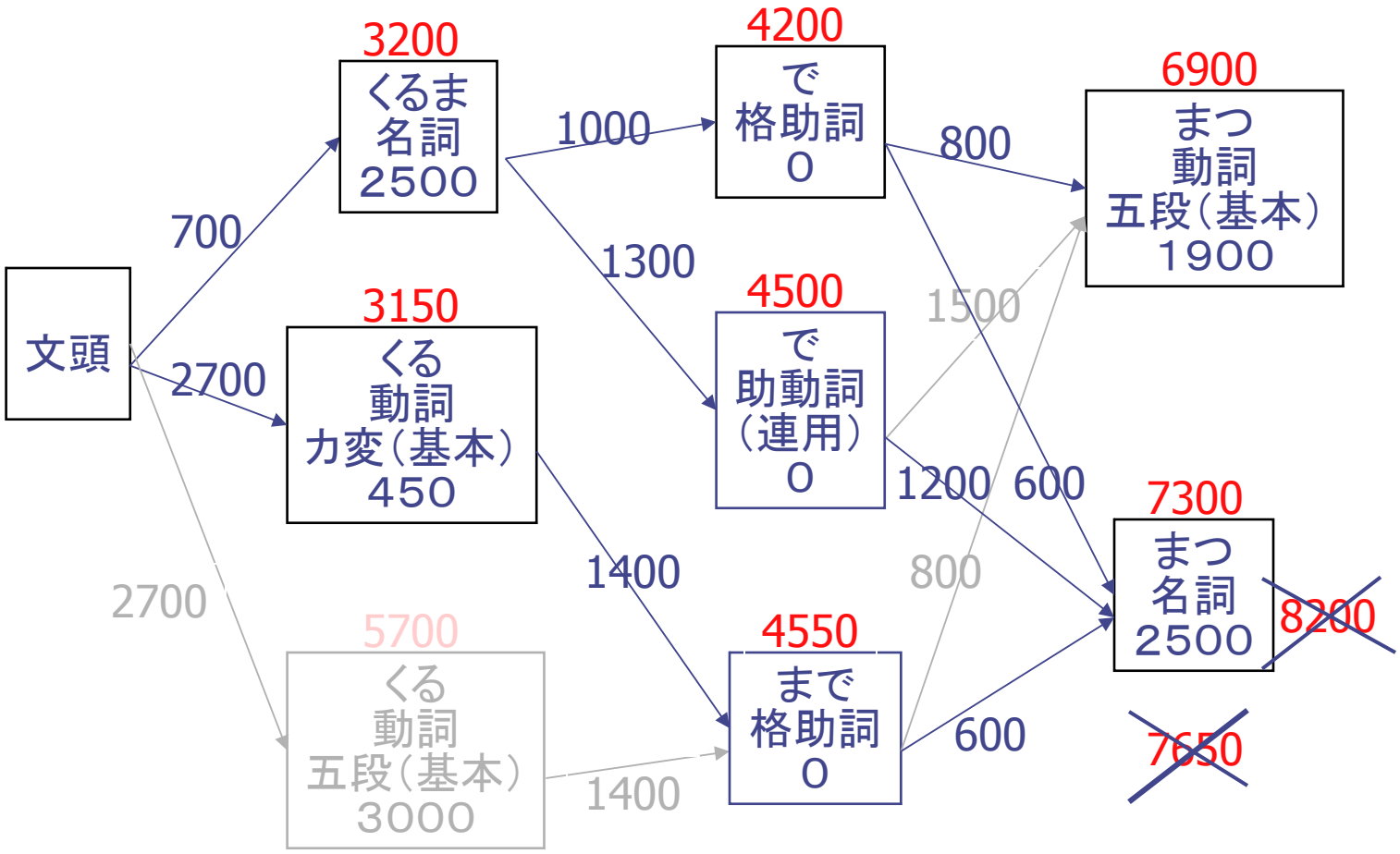
最小コスト法 (Viterbi アルゴリズム)



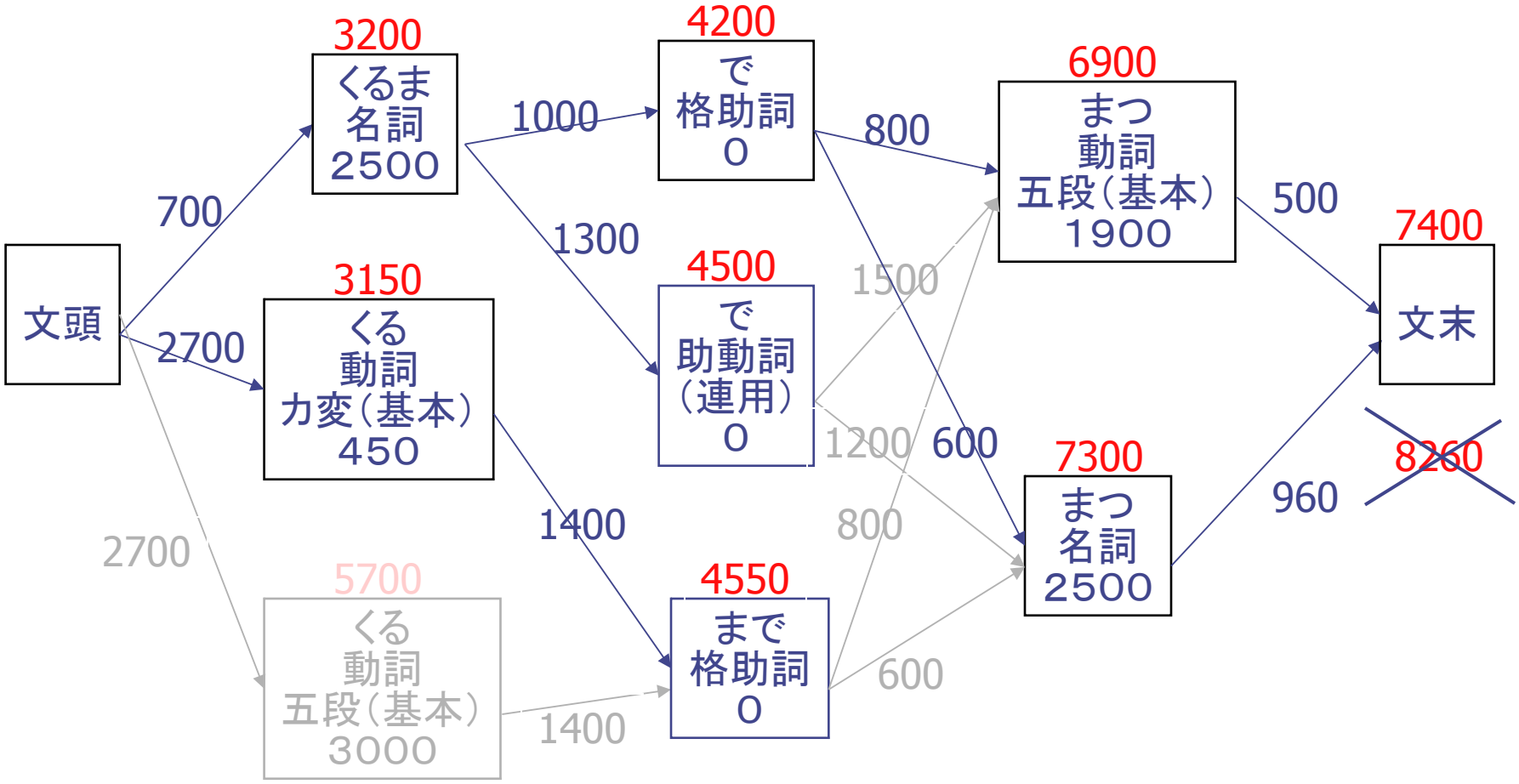
最小コスト法 (Viterbi アルゴリズム)



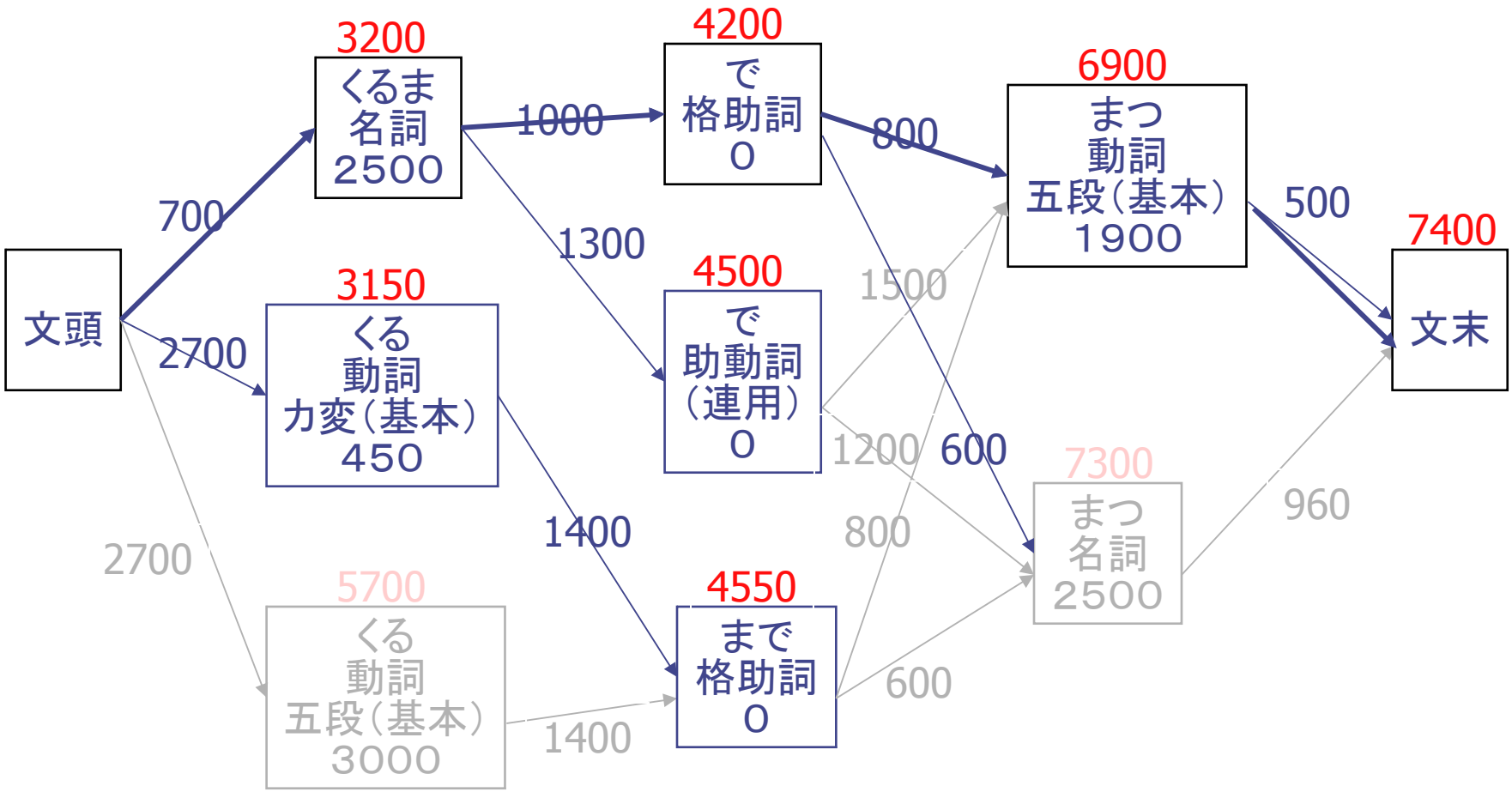
最小コスト法 (Viterbi アルゴリズム)



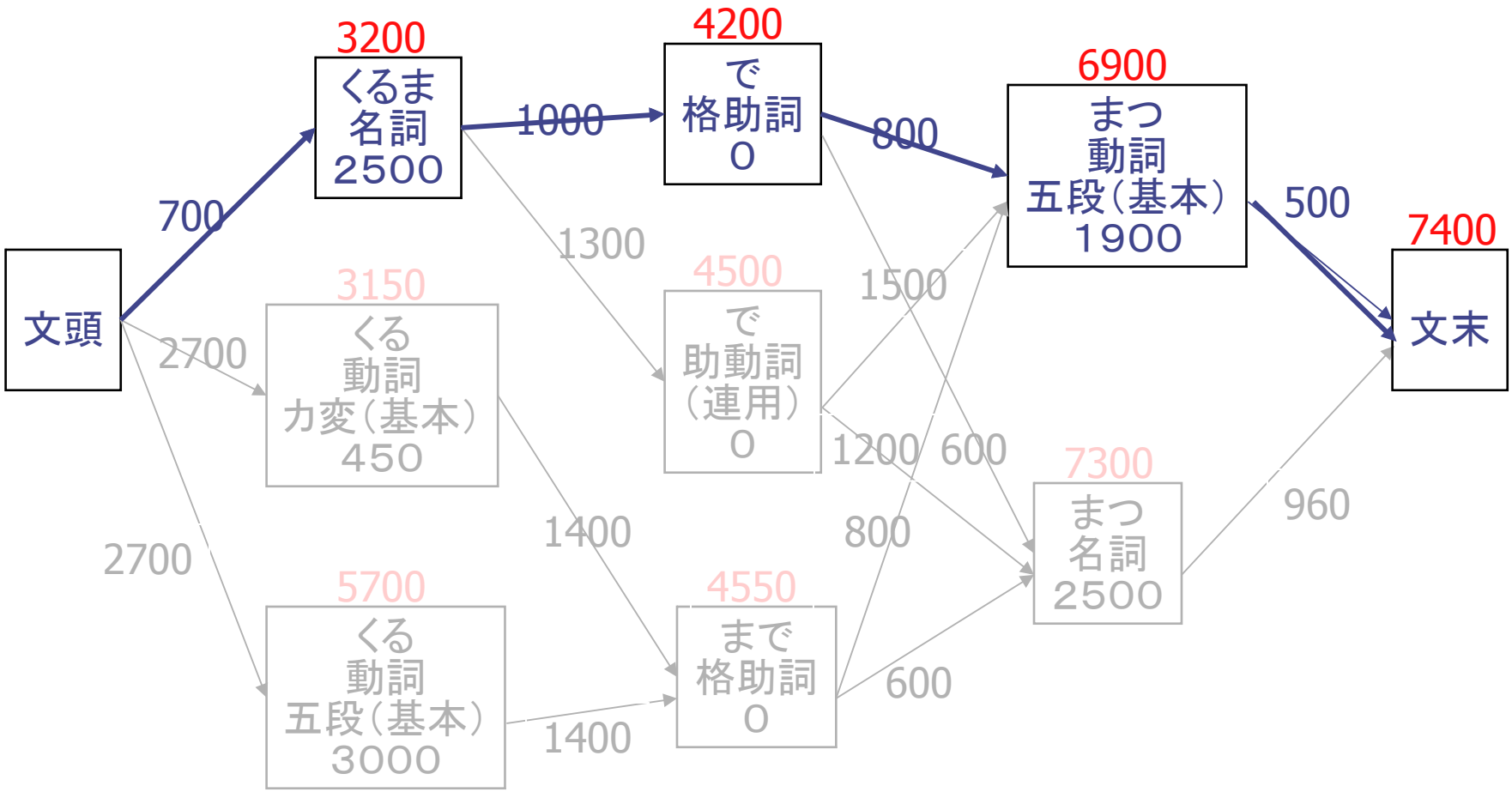
最小コスト法 (Viterbi アルゴリズム)



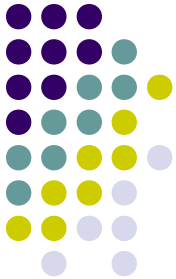
最小コスト法 (Viterbi アルゴリズム)



最小コスト法 (Viterbi アルゴリズム)

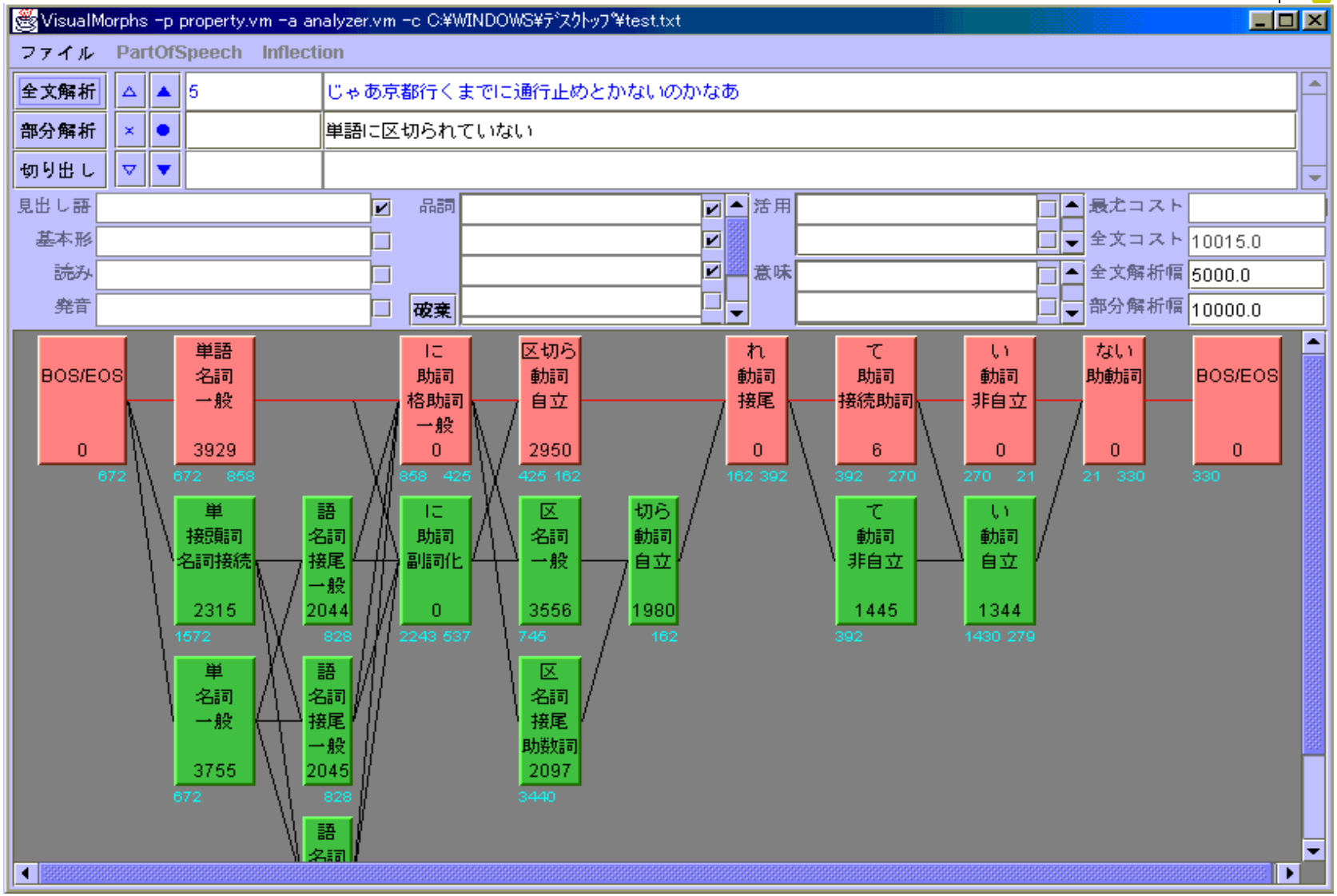


コストの決定方法

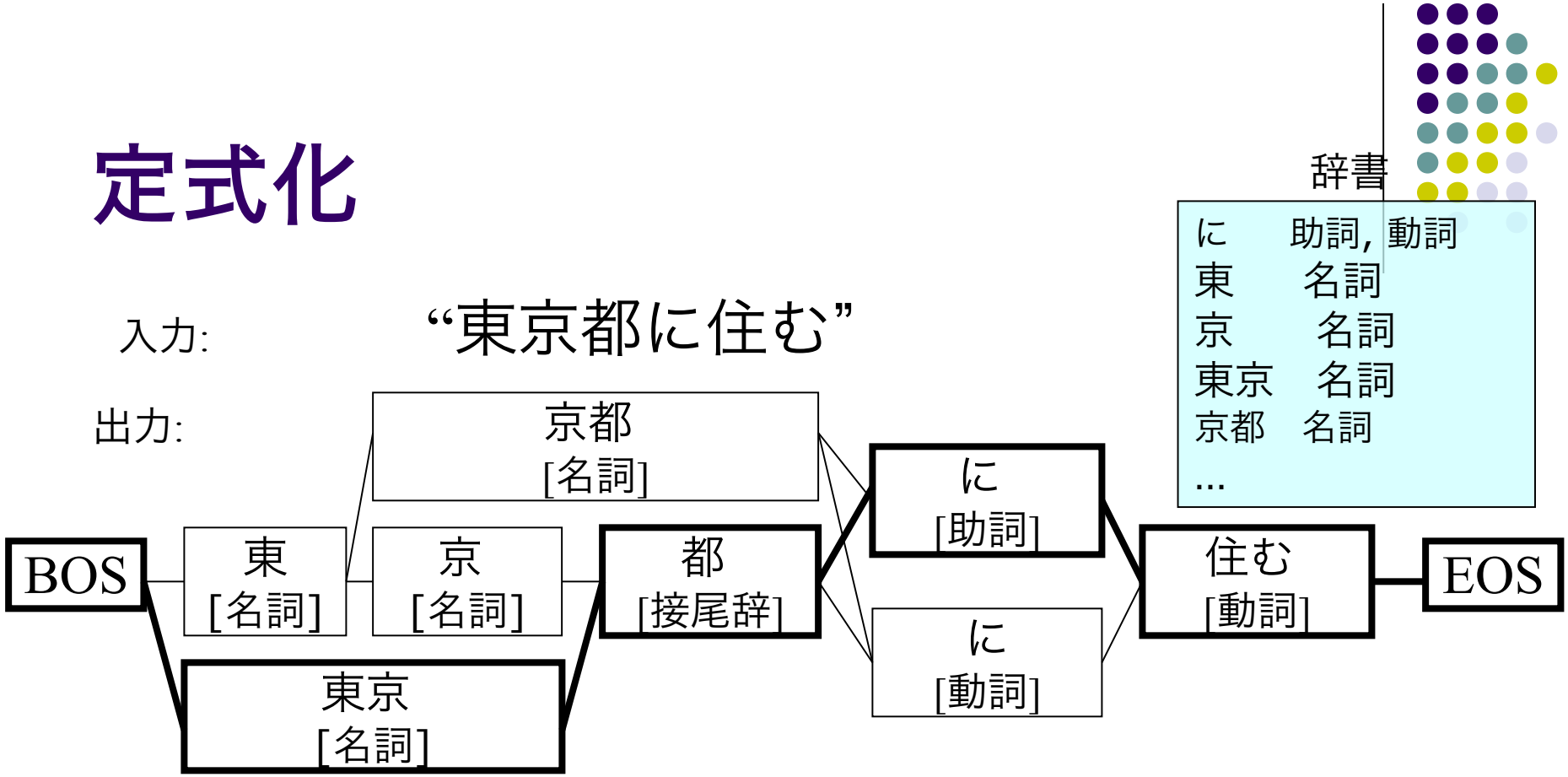


- 人手でガンバル (90年代はじめ)
  - 試行錯誤の連続, かなり大変
  - 客観的評価が難しい
- 統計処理
  - 教師なし推定 (生テキストからのみ推定)
    - 超低コスト
    - わかちき程度ならうまくいく
  - 教師あり推定 (正解データを作って推定)
    - 現代の形態素解析器の主流
    - HMM, CRF

正解データ作成ツール (VisualMorphs)



定式化



形態素解析 ＝ 可能な経路から正しい経路を１つ選ぶタスク

入力文 X

出力系列 $Y = (\langle w_1, t_1 \rangle, \ldots, \langle w_{\#Y}, t_{\#Y} \rangle)$

Hidden Markov Model (HMM)

- テキストを生成するような生成的確率モデル
- 形態素解析を**間接的に**解いている
- 頻度を数えるだけなので実装は超簡単
- EMアルゴリズムを用い生テキストからも推定可能

$$P(Y,X) = P(T,W) = P(W \mid T)P(T)$$

$$\approx \prod_i P(w_i \mid t_i)P(t_i \mid t_{i-1})$$

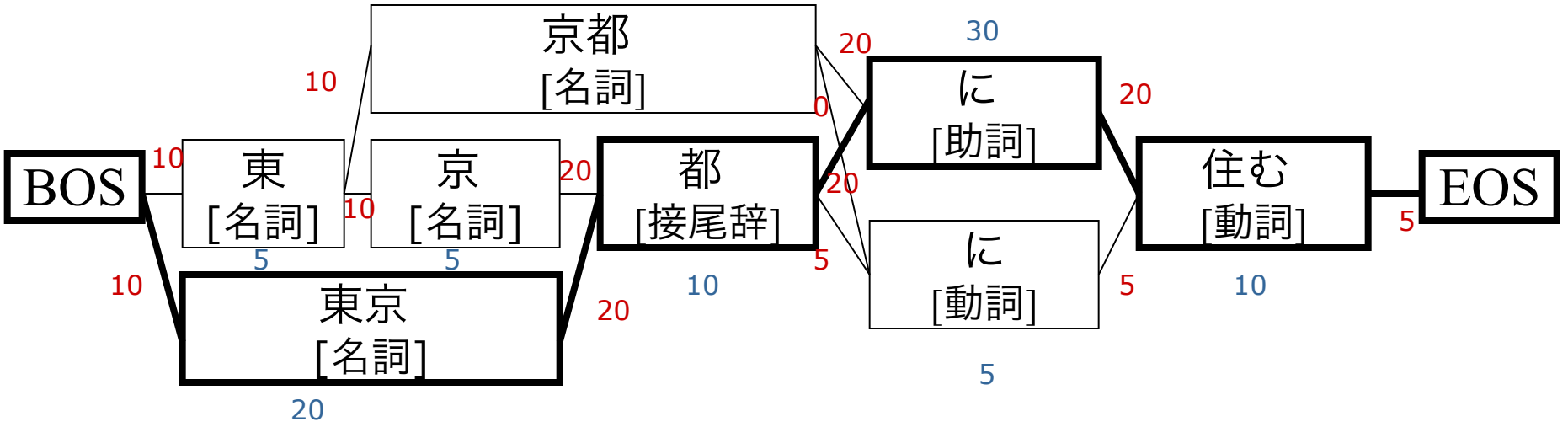
負の対数化

$$\sum_i [-\log P(w_i \mid t_i) - \log P(t_i \mid t_{i-1})]$$

生起コスト　　連接コスト

$$P(w_i \mid t_i) = f(w_i, t_i) / f(t_i)$$

コスト最小法 = 線形モデル



BOS-名詞　名詞-接尾　名詞-東京

大域素性ベクトル **F**(Y,X) = (… … 1 … … 1 … … 1 … 1 … 0 …)

コストベクトル **Λ** = (… … 5 … … 20 … …20 … 10 .. 0 …)

Yのコスト値 = **Λ**・ **F**(Y, X)

素性ベクトルとコストベクトルの内積

Conditional Random Fields

- P(Y|X) を単一の指数分布モデルで表現

$$P(Y \mid X) = \frac{\exp(-\boldsymbol{\Lambda} \cdot \mathbf{F}(Y, X))}{Z_X}$$

Y のコスト

$$Z_X = \sum_{Y' \in \Psi(X)} \exp(-\boldsymbol{\Lambda} \cdot \mathbf{F}(Y', X))$$

$\Psi(X)$ ：出力経路の候補集合 (すべての Y)

- 推定戦略の違い
  - HMM: 同時確率　　P(Y, X)
  - CRF:　条件付き確率　P(Y|X)
- CRFは形態素解析を**直接的に**解いている

CRFのパラメータ推定

- 山登り法による最尤推定（関数の最大値探索問題）

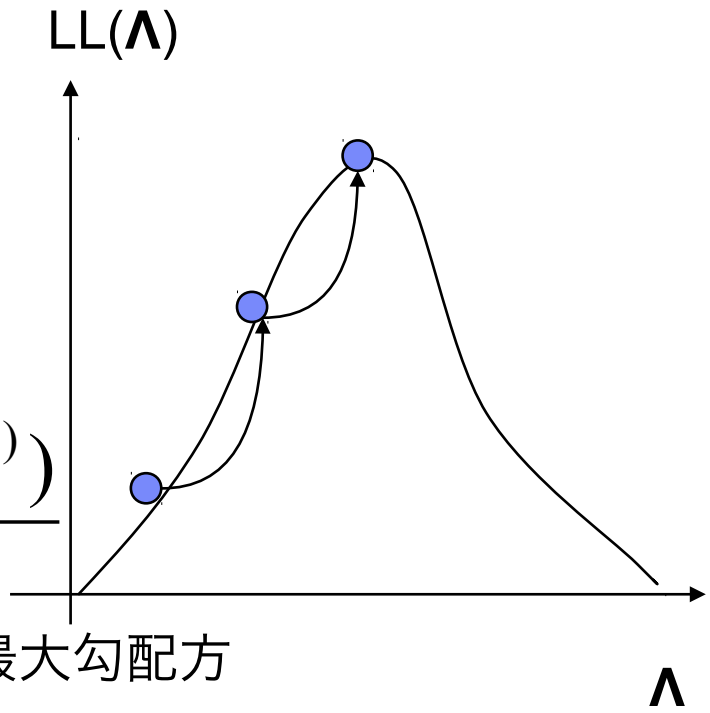
1. 現在のパラメータの対数尤度(LL)の計算

$$LL(\Lambda) = \sum_{i \in \text{training data}} \log P_{\Lambda}(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)})$$

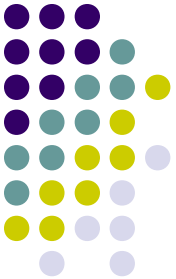
2. 偏微分(勾配)の計算　（0なら終了）

$$\frac{LL(\Lambda)}{\partial \Lambda} = \sum_{i \in \text{training data}} \frac{\log P_{\Theta}(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)})}{\partial \Lambda}$$

3. 勾配方向へパラメータの更新（たとえば、最大勾配方向）して、1に戻る



CRFのパラメータ推定 cont'd



- パラメータ推定の簡易版 (パーセプトロン)

```
Λ = 0        ・・・初期化
For I = 1 .. T    学習回数
  Forall  X  in 学習データ
    Y_opt = DoViterbi(X, Λ)      ・・・ 現在のパラメータで解析
    Λ = Λ - [ F(Y_正解, X) – F(Y_opt, X) ]    ・・・パラメータの更新
  End
End
```

- 正解データが出力されるようにパラメータを調節
- CRF
  - $F(Y\_正解) - E_{\_P(Y|X)}\ [F(Y, X)]$
  - 期待値計算→ ForwardBackward アルゴリズム

今後の課題

- 未知語処理:　辞書にない単語の対応
  - 現在の実装
    - 字種に基づくヒューリスティックス
    - カタカナ連続を「疑似単語」として登録
    - 字種の定義等は外部ファイルに記述
  - 問題点
    - 無駄な疑似単語生成:　99%以上は無駄、解析速度に大きく影響
    - 疑似単語生成の精密なコントロール
      - 必要な時に正しい単位で生成

オープンソース形態素解析器

- 松本教授による prolog プロトタイプ
- 妙木,黒橋氏による C 実装
- コスト決定に2ヵ月

辞書の再編成

92年 0.6　94年 0.6　96年 3.1　03年 4.0　05年 5.1

JUMAN

パトリシアTRIE
back port

96.09.30
ChaSen

97年 1.0

- 浅原氏によるコスト推定法の研究
- 北内氏によるトライグラム拡張

03年 2.3.3

- 松本教授がNAISTに
- 統計処理によるコスト推定
- パトリシアTRIEによる高速化
- たつを氏が中心に開発

ダブル配列
back port

02.12.4
MeCab

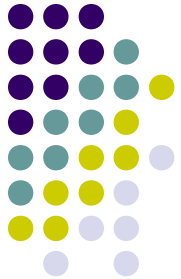
06年 0.9

- ダブル配列による高速化
- 機能限定、よりシンプルに
- 開発言語を C++ に変更
- オブジェクト指向で書き直し

03年
Sen
MeCabの
Java port

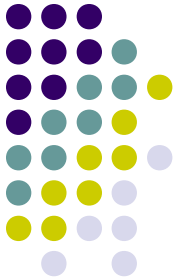
06年 1.2.2

MeCab の設計方針



- 辞書とシステムの完全分離
  - 自然言語の複雑さはシステムではなく辞書/コストとして外部定義
  - システムは日本語を知らない
  - grep “名詞” *.cpp としても何も出てこない :-)
  - システムは「ひらがな」「カタカナ」の区別すら知らない　(文字種の情報もすべて外部定義)
  - 他の言語も辞書さえあれば解析可能
- 解析速度を犠牲にしない
  - 事前に計算できることはすべてやっておく
  - 辞書やコスト値はすべてバイナリデータ
  - ディスクの使い方は富豪的

MeCab の設計方針

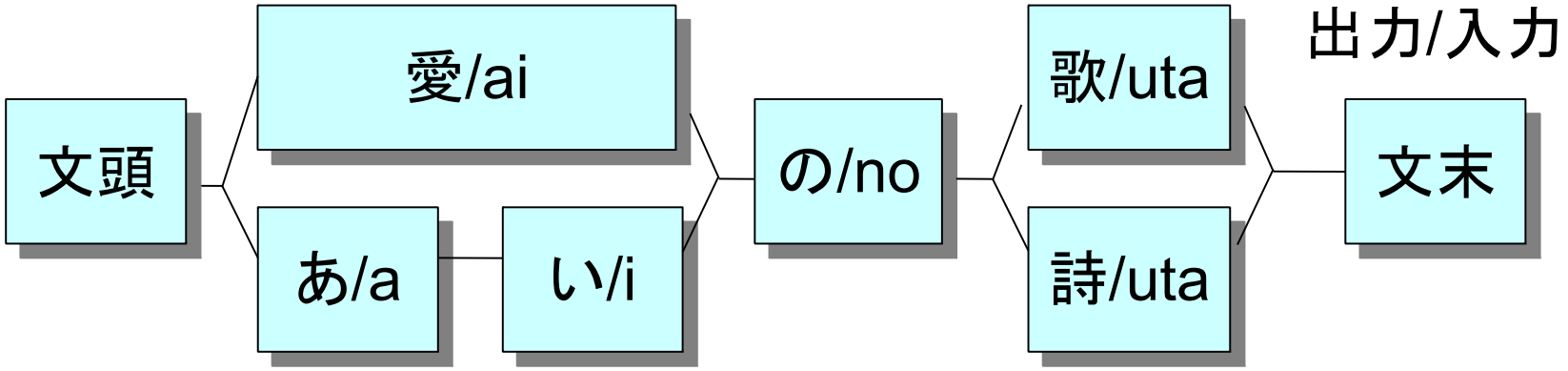


- 機能の選別
  - 前処理/後処理でできることはやらない
    - ChaSen の機能過多の反省
      - 文字コード変換, 改行処理, 連結品詞, 注釈, ChaSenサーバ
    - かわりに API を充実
      - C/C++, Perl, Java, Python, Ruby, C# ...
  - 解析器にしかできない機能を提供
    - N-best 解, 制約つき解析, ソフト分かち書き(後述)

```
use MeCab;
my $str = "すももももももものうち";
my $mecab = new MeCab::Tagger("");
for (my $n = $mecab->parseToNode($str);
      $n; $n = $n->{next}) {
    printf "%s\t%s\n", $n->{surface}, $n->{feature};
}
```

汎用テキスト処理ツール

- MeCab は日本語形態素解析器だけではない
  - 汎用的に作っています!
- テキスト → テキストの汎用変換ツール
  - 仮名漢字変換 (mecab-skkserv, AJAX IME)
  - ローマ字→ひらがな
  - 文字コード変換 (ちと強引)
  - 適切に辞書/コスト値を作れば実現可能!



MeCab の辞書

1. dic.csv (辞書定義)

```
の,166,166,8487,助詞,格助詞,一般,*,*,*,の,ノ,
京都,1306,1306,1849,名詞,固有名詞,地域,一般,*,*,京都,キョウト,キョート
桜,1304,1304,7265,名詞,固有名詞,人名,名,*,*,桜,サクラ,サク
....
```

- 単語, 左文脈id, 右文脈id, 単語生起コスト, 素性列(CSV)
- 素性は任意の情報(品詞,活用,読み等)をCSVで記述

2. matrix.def (連接コスト定義)

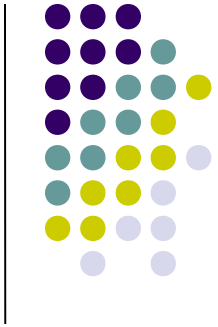
```
1306  166    -2559
1304 1303     401
166   1304     608
```

左文脈id　右文脈id　単語連接コスト

単語連接コスト
単語生起コスト

3. char.def (文字の定義)
4. unk.def (未知語処理の定義)
5. dicrc (出力フォーマット等)

AutoLink



自動的にリンクが張られる機能です
MeCabで実現できます。

1. dic.csv (辞書定義)

```
リンク,0,0,-500,http://foo.com/
MeCab,0,0,-200,http://mecab.sourceforge.jp/
Google,0,0,-100,http://www.google.com/
....
```

- 連接は一状態
- 単語の長さに対し指数的に
  小さくなるコスト
- 素性にリンク先 URL

2. matrix.def (連接コスト定義)

```
0 0 0
```

連接は使わないので一状態
コスト 0

3. char.def (文字の定義)
4. unk.def (未知語処理の定義)
 → デフォルト 1 文字 1未知語

5. dicrc

```
node-format-autolink = <a href="%H">%M<a>
unk-format-autolink = %M
```

%M: 単語 (入力)
%H: 素性 (出力)

T9風 予測入力

1/あ	2/か	3/さ
4/た	5/な	6/は
7/ま	8/や	9/ら
	0/わ	

入力: 1681 → おはよう, 241 → くどう
語呂合わせ: 1192 → 哀楽, 794 → 森田

1. dic.csv (辞書定義)

```
1,10,10,0,オ
2,11,11,0,カ
2,12,12,0,ガ
....
```

- 単語(入力): 数字
- 文脈id: すべての
  カタカナ文字に対応

2. matrix.def (連接コスト定義)

```
10 9 2505
10 10 396
10 11 606
10 12 964
...
```

- wikipedia を mecab で解析
- 単語の読みと頻度を取得
- カタカナのつながりやすさ
  をコスト化
- 「日本語らしさ」

3. char.def (文字の定義)
4. unk.def (未知語処理の定義)
 → デフォルト 1文字 1未知語

5. dicrc

```
node-format-katakana = %H
unk-format-katakana = %M
```

%M: 単語 (入力)
%H: 素性 (出力)

子音入力

dmdkry → だめだこりゃ
tnkyhu → てんきよほう
kdutk → くどうたく

1. dic.csv (辞書定義)

```
a,6,6,0,ア
k,15,15,0,カ
k,17,17,0,キ
py,137,137,0,ピャ
```

- 単語(入力): 母音無しローマ字
- 文脈id: すべての
  カタカナ文字に対応

2. matrix.def (連接コスト定義)

```
6 15 796
6 17 675
6 137 3121
6 138 3121
...
```

- wikipedia を mecab で解析
- 単語の読みと頻度を取得
- カタカナのつながりやすさ
  をコスト化
- 「日本語らしさ」

3. char.def (文字の定義)
4. unk.def (未知語処理の定義)
→ デフォルト 1文字 1未知語

5. dicrc

```
node-format-katakana = %H
unk-format-katakana = %M
```

%M: 単語 (入力)
%H: 素性 (出力)

MeCab の素性フィールドの利用

- 辞書の素性は CSV なら何でも可能
- 単語にさまざまな付加情報を付与
  - 意味情報
  - 関西弁
  - スパムスコア

```
の,166,166,8487,助詞,格助詞,一般,*,*,*,の
桜,1304,1304,7265,名詞,固有名詞,人名,名,*,*,桜,サクラ
....
```

- スパムフィルタの例
  - 通常のスパムフィルタ
    - MeCab で解析 → 単語の抽出
    - 単語をキーにスパムスコア辞書をルックアップ
  - 辞書引きが2回!　辞書の付加情報として持っておけばMeCabだけでスパムスコアリングが可能

ルー語変換

今日は良い天気です →
トゥデイはグッドゥウェザーです

1. dic.csv (辞書定義)

```
美しい,43,43,4225,ビューティフル
燃える,619,619,6565,バーンする
誘い出す,731,731,6469,ルアーする
は,261,261,3774,は
```

- mecab-ipadic を編集
- 品詞の部分にルー語
- ルー語がないものは単語そのまま

2. matrix.def (連接コスト定義)

```
0 831 348
0 832 -1264
0 833 -1168
0 834 -1193
...
```

- mecab-ipadic そのまま

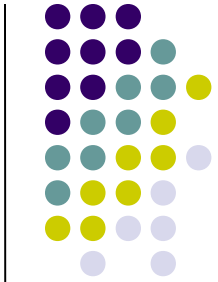
3. char.def (文字の定義)
4. unk.def (未知語処理の定義)
→ mecab-ipadic そのまま

5. dicrc

```
node-format-lou = %H
unk-format-lou = %M
```

%M: 単語 (入力)
%H: 素性 (出力)

まとめ



- MeCabの技術
  - 辞書引き
    - 通常の hash は使えない
    - TRIE
  - 曖昧性の解消
    - 最小コスト法
    - 統計処理による正解データからの推定
- 設計方針
- 汎用性
  - テキスト変換ツール
  - 意外な使い方