自然言語処理
第3回

中田尚

演習課題（5点）

締切：4/21 23:59

- 実質的には任意課題と同じ
- レベル1（3点）
  - 入力ファイルの概要とファイルサイズ
  - ソースコード（LMSと同じ部分は省略可能）
  - 出力された文章（20文字以上）
- レベル2（+2点）
  - 出力された文章の考察
    - 良かったのか悪かったのか？
    - n-gramの性質を踏まえた考察

- 単語分割して単語単位のn-gramとしても良いが加点はない

n-gramの性質

- コーパス全体の平均的統計情報
- 1-gramの場合
  - 文字/単語の出現頻度
    - **日本語ではひらがなが圧倒的**
    - 漢字に比べて種類が少なく、かつ、助詞や活用で使用率が高い（約3割）
    - い、ん、か、し がtop4という調査が多め
- 2-gram
  - 2文字の出現頻度
  - やはり圧倒的に「ひらがな2文字」になる
- 「る。」のように句読点になることもある

n-gramの性質

- 2-gram
  - 2文字の出現頻度
  - やはり圧倒的に「ひらがな2文字」になる
    - 「い○」だと「いる」
    - 「る○」だと「るこ」（「すること」等）
    - 「こ○」だと「こと」
    - 「と○」だと「とい」（「という」等）
    - 「い○」に戻る
  - つまり、「いることといることといること・・・」となる
- レポートは上記を踏まえて自分の言葉で書く

レポート講評

- 「2-gramでは文脈を考慮できない」
  - 文脈を考慮できないのは間違いではない
  - 考慮できないから、どうなのか？
- 「あうある」のような結果
  - あり得ないはずなので、、、
- 「確率的に選べば良くなる」
  - 本当にそう思ってる？どこかで見たのを書いただけ？
- 「3-gram, 4-gramと増やせば良くなる」
  - 10-gramでやってみるとよい
- 「1MBを30回コピーした」
  - え、、、それは1MBの頻度を30倍するのと同じです。
- 「○×」は「○×□」や「○×△」のように頻出するのでbigramで出現回数を「○×」は頻出するが「×□」は、、

ChatGPT 4o作

- 日本語の文字単位2-gram（bigram）を分析すると、上位に頻出するのは「いる」「るこ」「こと」「とい」など、主にひらがな2文字の機能語的な連なりである。これを順に接続すると「いることといること…」といった無限ループ的な非文構造が生まれる。これは、bigramが前後の文脈や構造を無視して単純な隣接文字の出現頻度に依存するため、意味的な整合性が保証されないことが主な原因である。日本語では助詞・補助動詞などの頻度が高く、それがn-gramに偏りをもたらす。このように、意味を持つ文章生成には、構文構造や語彙間の意味関係を考慮する必要があり、文字単位のn-gramでは限界があることが分かる。
- **生成AIの台頭＝これと同等以上が即座に求められる世界である**

文字体系／文字コードと形態素解析

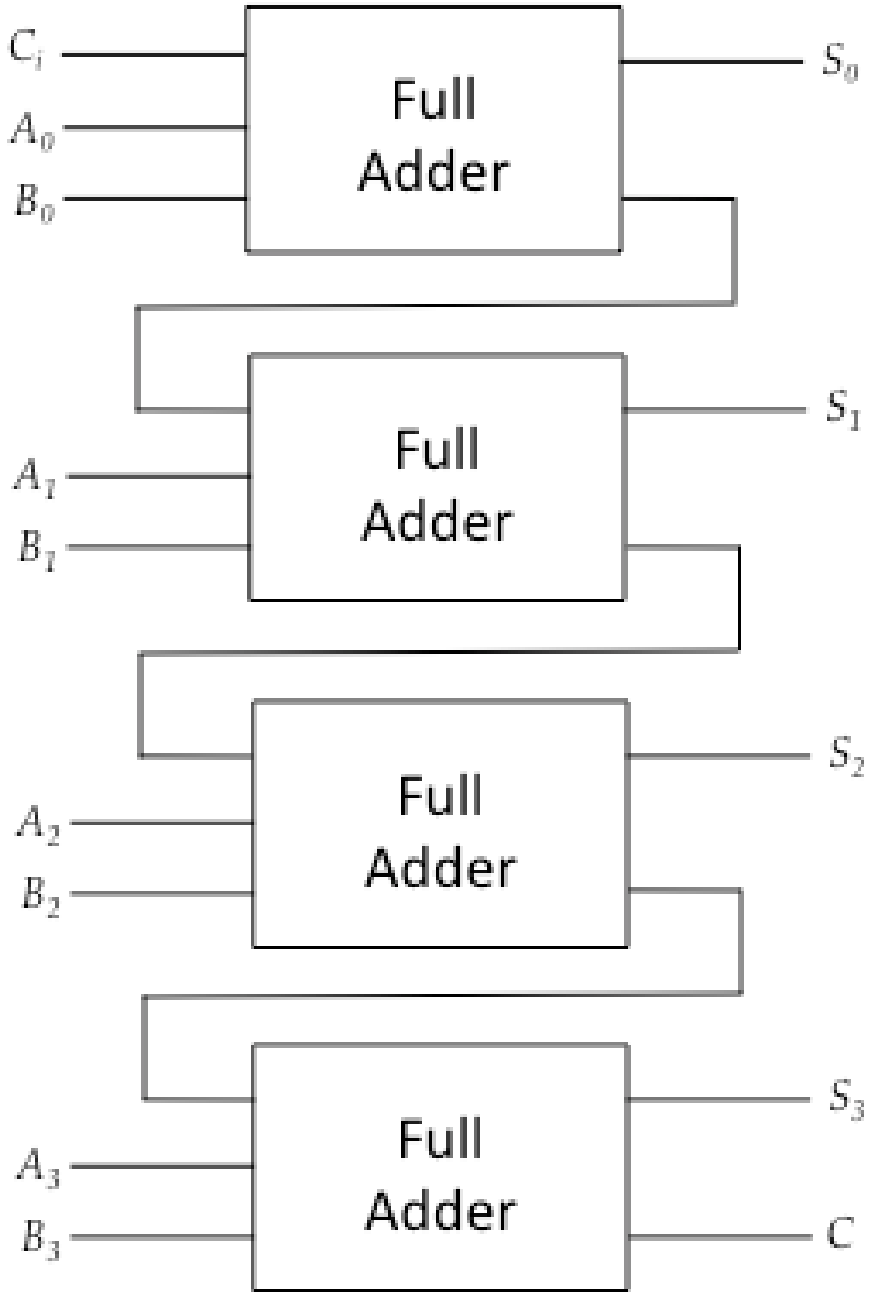
- データとしての文字列
  - そもそもデータとは
  - 文字列とは
- 文字体系／文字コード
  - ASCII
  - 多バイト文字
  - UTF
- 形態素解析とは
  - 日本語の形態素解析
  - 英語の形態素解析

そもそもデータとは

- 復習
  - ビット：0と1の2進数の最小単位
  - 8ビット＝1バイト
    - intは4バイト＝32bit
- 1000は2進数で 0011 1110 1000

加算器（寄り道）

- 2進数の加算は
- AとBの位を揃えて、1bitずつ足し算をし、繰り上がりを隣に送る



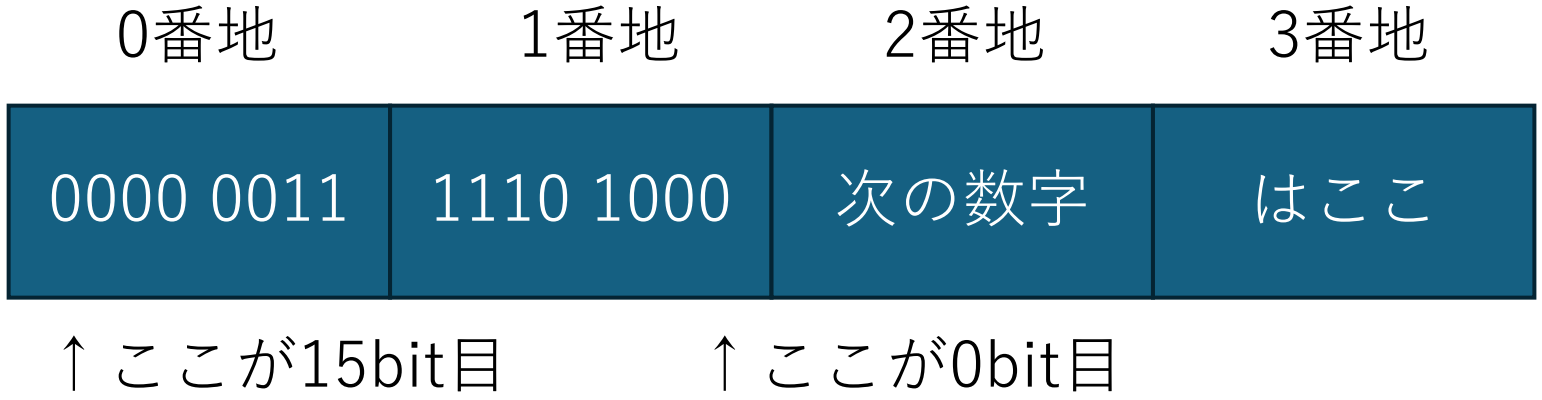
メモリに保存する

- メモリはC言語の配列のようなもの
  - 箱が複数並んでいる
  - 箱には番号がついている（アドレス）
  - 普通は1箱＝1バイト＝8bit（バイトアドレス）

- 1000= 0011 1110 1000 を保存するには2種類考えられる
  - 見たまま並べる

ビッグエンディアン

- 1000= 0011 1110 1000 を保存するには2種類考えられる
  - 見たまま（読み上げる順序に）並べる



  - 番地とbitの並びが逆！

読み上げたときに最後が大きなアドレスで終わる＝ビッグエンディアン

リトルエンディアン

- 1000= 0011 1110 1000 を保存するには2種類考えられる
  - 番地とbitの並びをあわせる

  - 計算に使う順に読み込まれる
  - 加算器にもそのまま接続可能

読み上げたときに最後が小さなアドレスで終わる＝リトルエンディアン

ビッグとリトル

- ビッグエンディアン
  - インターネット
  - 普通の文章の見た目の順序で送り出すイメージ
- リトルエンディアン
  - 現代のほぼすべてのコンピュータ
  - 計算は下位の桁から行うイメージ
- 歴史的にはプロセッサはビッグエンディアンとリトルエンディアンが競合していた
  - 有名なプロセッサで最後のビッグエンディアンはPowerMac G5

文字列とエンディアン

- 1バイト単位であればどちらでも同じ
  - 2バイト以上の固定長であればエンディアンの違いは重要
  - 1,2,3,,,バイトの可変長であればどちらでも同じ

文字コード

- そもそも文字コードとは
- 文字コードの歴史
  - ASCII
  - 多バイト文字
  - UTF-8
    - エンディアン
  - 文字化けやフォント

文字コードとは

- 数字と文字の対応表
  - "A"は65 など
  - 文字コードとしての最大はUnicodeの0x10FFFF（111万4,111）

- ASCII（American Standard Code for Information Interchange）
  - 7bit＝128文字
  - 現代のアルファベット＋数字の世界標準
  - ほとんどの文字コードはASCIIの拡張
  - A-Z、a-z、0-9が連続して並んでいる
    - EBCDIC（A-Z、a-zが連続していない）
  - 周辺機器を制御するための文字（制御文字）も含まれる

多バイト文字

- 1バイトは256種類なので、ひらがなはともかく、漢字を扱うことは全くできない
  - 他の非英語圏でも同様
  - 各国で様々な文字コードが作られた
  - その結果、同じファイルであっても文字コードがわからなければ何が書いてあるのかわからない
  - 日本語だけでも複数の文字コードがある
- 日本語の文字コードの例
  - JIS(ASCIIと日本語を特殊文字で切り替え)
  - Shift JIS（1バイト目は後半128文字の一部、2バイト目は全部で表現）
  - EUC-JP（1バイト目も2バイト目も後半128文字で表現）

その他の言語

- 多くの言語は、絵→絵文字→象形文字→現在の文字となったと考えられている
  - 「ハングル」は人間が考えて作った文字体系母音と子音を組み合わせて文字ができている
    - ルールさえ覚えれば少ない学習量ですべて読める
- 古典的な文字の分類
  - 表意文字：文字が意味を表す
    - 漢字など
  - 表音文字：文字が発音を表す
    - 音節文字：ひらがな、カタカナ（1つで1つの発音）
    - 音素文字：アルファベット（母音＋子音の組み合わせ）

言語と文字

- 一般に言語と文字は一対一に対応しない
  - 漢字は中国、日本、韓国（CJKと呼ばれる）等で使われる
- 多くの言語でアルファベット対応（ラテン文字転写）が決められている

  ±∂ð≠∈→ **namaste**
  - コンピュータでの入力やパスポート標記などに使われる

分かち書き

- 単語間を空白で区切る書き方
  - 分かち書きをする：英語など（90%以上の言語）
  - 分かち書きをしない：日本語、中国語、タイ語など

  - 分かち書きされていれば単語分割は簡単

Unicode

- 全世界の文字を単一の文字コードで表現しよう
  - 当初は16bit固定長を目指していた
    - 足りなかったので、21bitになった
  - 21bitを表現する方法に複数ある
    - UTF（Unicode Transformation Format）
    - UTF-8（現在の主流）
      - 1～4バイトASCII互換、主な漢字は3バイト
    - UTF-16
      - 2バイトまたは4バイト
      - ほとんどが2バイトになるので、日本語等ではUTF-8よりも小さくなる可能性がある
    - UTF-32
      - 4バイト固定
      - 単に21bit並べて残りを0にするので単純だが、ファイルサイズは大きくなる

Unicodeの正規化

- 濁点を表す文字がある
  - 「ぁ」 のように任意の文字と結合して濁音を作ることができる
  - その結果「が」と「が」のように本来の濁音に2種類の表現が可能
- 正規化に複数種類がある
  - NFD（全部分解する）
  - NFC（全部結合する）
  - これにより見た目が同じ文字は同じ文字コードになる
- OS間の問題
  - MacOSはNFDでWindowsはNFCなので、USBメモリ等で問題が発生する
    - 検索で見つけられない
    - 見た目が同じファイルが作れたりする

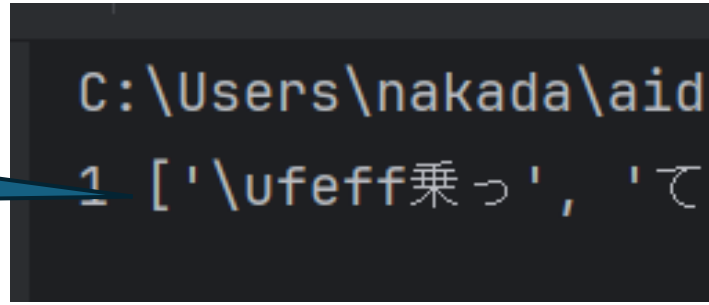
Unicode（その他）

- 絵文字も多数登録されている
  - OSによって見た目が違うものも多いので注意
  - 文化によっても違う
    -  Tofu on fire
- 肌の色を変更できる（Skin Tone Modifiers）
  - 👍🏻 👍 👍🏿

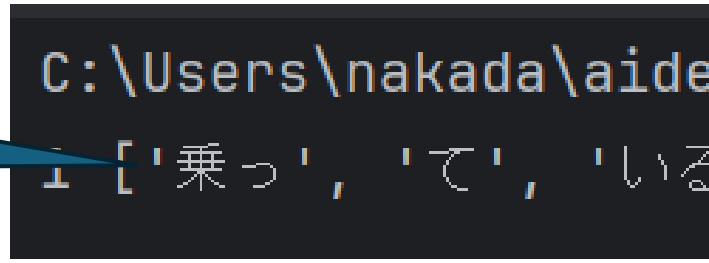
エンディアンとBOM

- UTF-16とUTF-32 にはビッグエンディアンとリトルエンディアンの2種類がある
  - ファイル単体で見分ける仕組みが必要
  - BOM（Byte Order Mark）
  - UTF-8：EF BB BF（FE FFのUTF-8表現）
  - UTF-16 BE：FE FF
  - UTF-16 LE：FF FE
  - UTF-32 BE：00 00 FE FF
  - UTF-32 LE：FE FF 00 00

utf-8でBOM
が見えている



encoding="utf-8-sig" とすると
BOMが自動で除去される



utf-8の文字数

- 1文字が1バイトだったり3バイトだったりする
- Pythonでは意識せずに1文字単位で操作できる
- C言語ではcharとwcharがあり、wcharならばバイト数を気にする必要はない

文字化けやフォント

- 文字コードに関する問題
- 文字化け
  - 文字コードを間違えて読み込んだ
- 中華フォント → → → → → →
  - UTFは日中韓台の文字を統一して扱う
  - 同じ漢字でも字体は各国で異なる
  - 言語を指定する方法はあるが不適切だと他国の文字に見える
  - 原因
    - 日本語対応アプリ等の開発現場に日本人がいない
    - 多言語化が容易になった弊害

https://note.com/usagimaruma/n/n87b2530627ea

形態素解析

- ginza等を使えば簡単にできる

- それでは面白くない

n-gramを使った形態素解析

- 形態素解析＝単語分割、品詞特定、活用や原形特定
- ここでは単語分割だけ

- 任意課題で作ったn-gramを元に、出現確率が最も高くなるように単語に分割する
  - 「し」の出現回数≧「した」の出現回数なので注意
    - 10倍以上違っても不思議ではない
    - 出現確率で計算する　例：ABの次にABCとなる確率
    - $P(A,B,C)=\frac{C(A,B,C)}{C(A,B)}$　　※P( )は確率、C( )は出現回数
    - 数学的に正しくは以下のようにABが前提条件であることを明記する
    - $P(A,B,C|A,B)=\frac{C(A,B,C)}{C(A,B)}$

古典的形態素解析

- MeCab、Juman、KyTea、Chasen
- 辞書を用いる
  - 辞書がないと読みや品詞を特定出来ない
- 活用はルールベースで認識
- 前後の文脈も見て判断する
  - 1文の中での文脈が中心

LLMにおける形態素解析

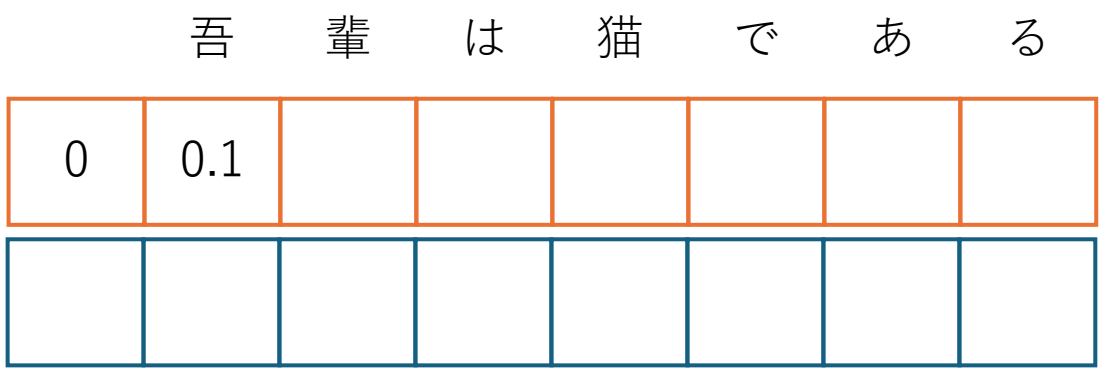
- トークン化
- 単語をIDに変換するだけ
  - それ以外の処理はLLMの内部で行われる
  - 明示的に品詞等を特定するのではなく、全部含めて処理する

n-gramと動的計画法による単語分解

- ChatGPT o4-mini-highの実行結果
  - 本/日/は/晴/天/な/り/。/吾/輩/は/猫/で/あ/る/が/、/名/前/は/ま/だ/無/い/で/す
  - 全部切れてる・・
- 私の結果
  - 本日は/晴天/なり。/吾/輩は/猫であ/るが、/名前/はまだ/無い/です
  - 「吾/輩は」の分析
    - hist2["吾輩"]/hist1["吾"]=0.03
    - hist2["輩は"]/hist1["輩"])=0.12
    - 「吾輩」の次は「は」であるが、「吾」の次は「吾輩」以外にたくさんある

動的計画法

- 計算用の配列を用意する（2つ）
  - 最初は0
  - 吾の確率は0.1だっとする
  - 吾/輩と吾輩を比較する
    - 吾/輩は P(吾)+P(輩) 0.1+0.05=0.15とする
    - 吾輩は C(吾輩)/C(吾) 0.03（前ページ参照）
    - 大きい方を採用する



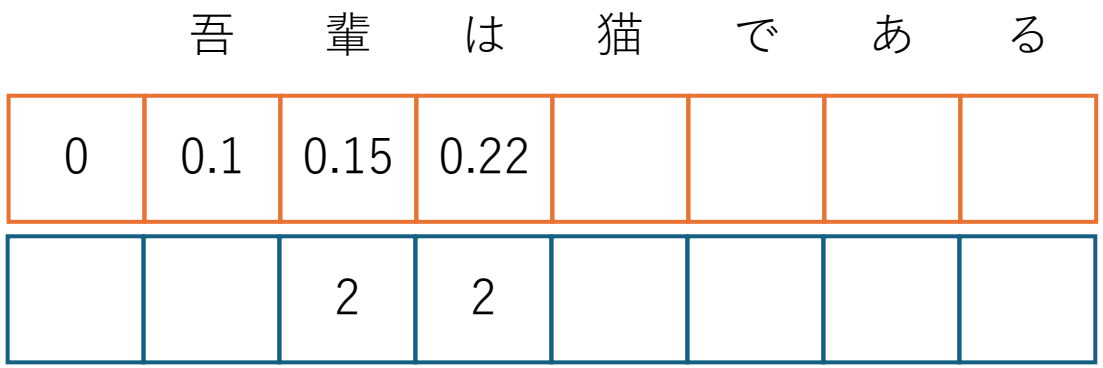
動的計画法

- 計算用の配列を用意する（2つ）
  - 最初は0
  - 吾の確率は0.1だっとする
  - 吾/輩と吾輩を比較する
    - 吾/輩は P(吾)+P(輩) 0.1+0.05=0.15とする
    - 吾輩は C(吾輩)/C(吾) 0.03（前ページ参照）
    - 大きい方を採用する
    - 下の配列に「輩」は2文字の最終文字であることを記録

吾	輩	は	猫	で	あ	る
0	0.1	0.15					
		2					

動的計画法

- 計算用の配列を用意する（2つ）
  - 「（吾輩）/は」と「吾/輩は」「吾輩は」を比較する
    - （吾輩）が「吾輩」か「吾/輩」かは気にしなくて良いのがポイント
  - 前の結果から「吾/輩は」が選ばれたとする
    - 数字は0.1+0.12=0.22となる
    - 「輩は」が2文字なので、下段は2



動的計画法

- 計算用の配列を用意する（2つ）
  - 「（吾輩は）/猫」と「（吾輩）/は猫」「（吾）/輩は猫」を比較する
    - （吾輩は）が0.22、（吾輩）が0.15、（吾）が0.1
  - 0.22+xと0.15+yと0.1+zを比較する
    - x,y,zは計算する
    - 結果として0.31になったとする（根拠なし）

	吾	輩	は	猫	で	あ	る
	0	0.1	0.15	0.22	0.31			
			2	2	1			

動的計画法

- 計算用の配列を用意する（2つ）
  - 同様に「で」は結果として0.35になったとする（根拠なし）

	吾	輩	は	猫	で	あ	る
0	0.1	0.15	0.22	0.31	0.35		
		2	2	1	2		

動的計画法

- 計算用の配列を用意する（2つ）
  - 「（吾輩は猫で）/あ」と「（吾輩は猫）/であ」「（吾輩は）/猫であ」を比較する
    - （吾輩は猫で）が0.35、（吾輩は猫）が0.31、（吾輩は）が0.22
  - 0.35+P(あ)
  - 0.31+C(であ)/C(で)
  - 0.22+C(猫であ)/C(猫で)
  - の3つを計算して最大を採用する「/猫であ/」になるのは3番目が最大

	吾	輩	は	猫	で	あ	る
0	0.1	0.15	0.22	0.31	0.35	0.48	
		2	2	1	2	3	

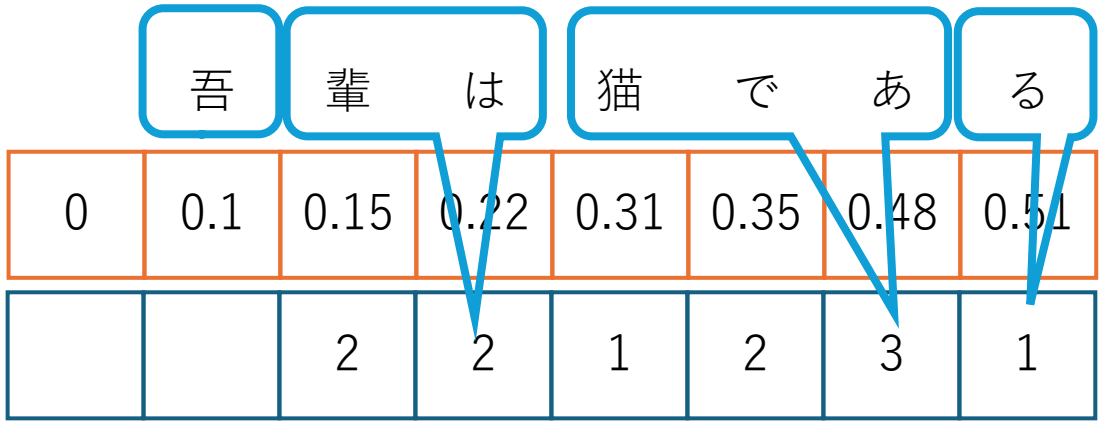
動的計画法

- 計算用の配列を用意する（2つ）
  - 最後も適当に埋めた

	吾	輩	は	猫	で	あ	る
0	0.1	0.15	0.22	0.31	0.35	0.48	0.51
		2	2	1	2	3	1

動的計画法

- 結果は後ろから辿る



次回予告

- 5/13
  - 文法の解析
  - そろそろ小テスト？