第4章: 形態素解析

30. 形態素解析結果の読み込み

> 形態素解析結果（neko.txt.mecab）を読み込むプログラムを実装せよ．ただし，各形態素は表層形（surface），基本形（base），品詞（pos），品詞細分類1（pos1）をキーとするマッピング型に格納し，1文を形態素（マッピング型）のリストとして表現せよ．第4章の残りの問題では，ここで作ったプログラムを活用せよ

MeCab version: 0.996

`./neko.txt.mecab` が存在していると仮定

各行は `表層形 品詞,品詞細分類1,品詞細分類2,品詞細分類3,活用形,活用型,原形,読み,発音` の形式

```
with open('./neko.txt.mecab') as f:
    lines = f.readlines()

# 最終結果が入るリストを定義.
sentences: list[list[dict]] = []

# 1文中の形態素が格納されるリストを定義.
# 1文を読み切るたびに結果に追加して、自身は初期化する.
sentence: list[dict] = []

for line in lines:
    if line.strip() == 'EOS':
        if not sentence:
            continue
        sentences.append(sentence)
        sentence = []
    else:
        surface, others = line.split('\t')
        elms = others.split(',')
        pos, pos1, base = elms[0], elms[1], elms[6]
        sentence.append({
            'surface': surface,
            'pos': pos,
            'pos1': pos1,
            'base': base,
        })
else:
    if sentence:
        sentences.append(sentence)
```

1文目(吾輩は..)の確認


```
[
  0 : {
    "surface" : "　"
    "pos" : "記号"
    "pos1" : "空白"
    "base" : "　"
  }
  1 : {
    "surface" : "吾輩"
    "pos" : "名詞"
    "pos1" : "代名詞"
    "base" : "吾輩"
  }
  2 : {
    "surface" : "は"
    "pos" : "助詞"
    "pos1" : "係助詞"
    "base" : "は"
  }
```

```
  3 : {
    "surface" : "猫"
    "pos" : "名詞"
    "pos1" : "一般"
    "base" : "猫"
  }
  4 : {
    "surface" : "で"
    "pos" : "助動詞"
    "pos1" : "*"
    "base" : "だ"
  }
  5 : {
    "surface" : "ある"
    "pos" : "助動詞"
    "pos1" : "*"
    "base" : "ある"
  }
  6 : {
    "surface" : "。"
    "pos" : "記号"
    "pos1" : "句点"
    "base" : "。"
  }
]
```


31. 動詞

> 動詞の表層形をすべて抽出せよ.

```
verb_surface_set = set(d['surface'] for d in itertools.chain(*sentences) if d['pos'] == '動詞')
```

愚直に書くと以下の様に2重for-loopで書くことになります.

ただ今回の場合文章毎に処理する必要がなく、全形態素について処理すればいいだけなので **itertools.chain** で2重リストをフラットにして一度に処理してしまうのが良さそうです. 動詞を判定する条件も簡単なのでジェネレータ式で簡単に書きます.

set() の引数には Iterator をそのまま書けるのでジェネレータ式を直接書いています.

```
# 愚直解法
verb_surface_set_verbose = set()
for sentence in sentences:
    for morph in sentence:
        if morph['pos'] == '動詞':
            verb_surface_set_verbose.add(morph['surface'])
```

32. 動詞の基本形

> 動詞の基本形をすべて抽出せよ.

```
verb_base_set = set(d['base'] for d in itertools.chain(*sentences) if d['pos'] == '動詞')
```

31. 動詞 と同様です.

33. 「AのB」

> 2つの名詞が「の」で連結されている名詞句を抽出せよ.

```
a_of_b_list: list[str] = []
# 文を跨いで連結されていても意味がないので、文ごとの処理となります
for sentence in sentences:
```

```
    # 連続した3つの形態素に興味があるので、0番目から最後から3番目の形態素までのfor-loopにします.
    for i in range(len(sentence)-2):
        morph1 = sentence[i]
        morph2 = sentence[i+1]
        morph3 = sentence[i+2]
        # 全ての条件を and でつなぐときには all関数を使うと可読性が向上することがあります
        if all([
            morph1['pos'] == '名詞',
            morph2['surface'] == 'の' and morph2['pos1'] == '連体化',
            morph3['pos'] == '名詞',
        ]):
            a_of_b_list.append(f'{morph1["surface"]}{morph2["surface"]}{morph3["surface"]}')
```

先頭の5つ確認


```
[
  0 : "彼の掌"
  1 : "掌の上"
  2 : "書生の顔"
  3 : "はずの顔"
  4 : "顔の真中"
]
```


34. 名詞の連接

> 名詞の連接（連続して出現する名詞）を最長一致で抽出せよ.

```
# 最長の連接を与える形態素のリストを用意
max_morphs: list[dict] = []
for sentence in sentences:
    # カウンタを用意
    i = 0
    # 一時的な連接を保持する形態素のリスト
    morphs: list[dict] = []
    while i < len(sentence):
        morph = sentence[i]
        if morph['pos'] == '名詞':
            # 名詞が続く場合は一時変数に追加
            morphs.append(morph)
        else:
            # 名詞が終わった場合は、暫定の最長のものと比較して更新可否判定
            if morphs and len(morphs) > len(max_morphs):
                max_morphs = morphs
            morphs = []
        i += 1
```

確認

manyaslip'twixtthecupandthelip

35. 単語の出現頻度

> 文章中に出現する単語とその出現頻度を求め，出現頻度の高い順に並べよ

語/単語の定義について十分な理解がなく、以下のコードは全て形態素 := 単語 とみなしたようなコードになっています

助詞等を排除する前処理や複数の連続する名詞の形態素を一定のルールでマージするような処理が必要な可能性があります.

```
# 何かをカウントするときには、自動的にキーを初期化してくれる defaultdict を使用すると便利です
d: defaultdict = defaultdict(int)
# 前述の通り文章ごとに考える必要がないので itertools.chain でフラットにして1重のfor-loopで処理
for morph in itertools.chain(*sentences):
    d[morph['surface']] += 1
# 出現数でソート
occurrences = list(sorted([(k, v) for k, v in d.items()], key=lambda x: x[1], reverse=True))
```

先頭5単語を確認


```
[
```

```
  0 : [
    0 : "の"
    1 : 9194
  ]
  1 : [
    0 : "。"
    1 : 7486
  ]
  2 : [
    0 : "て"
    1 : 6868
  ]
  3 : [
    0 : "、"
    1 : 6772
  ]
  4 : [
    0 : "は"
    1 : 6420
  ]
]
```


最後の出現数でソートする部分 (**occurrences** を計算する部分) のコードですが、愚直解法は以下のようになります。これを簡潔に書いたものと思ってください.

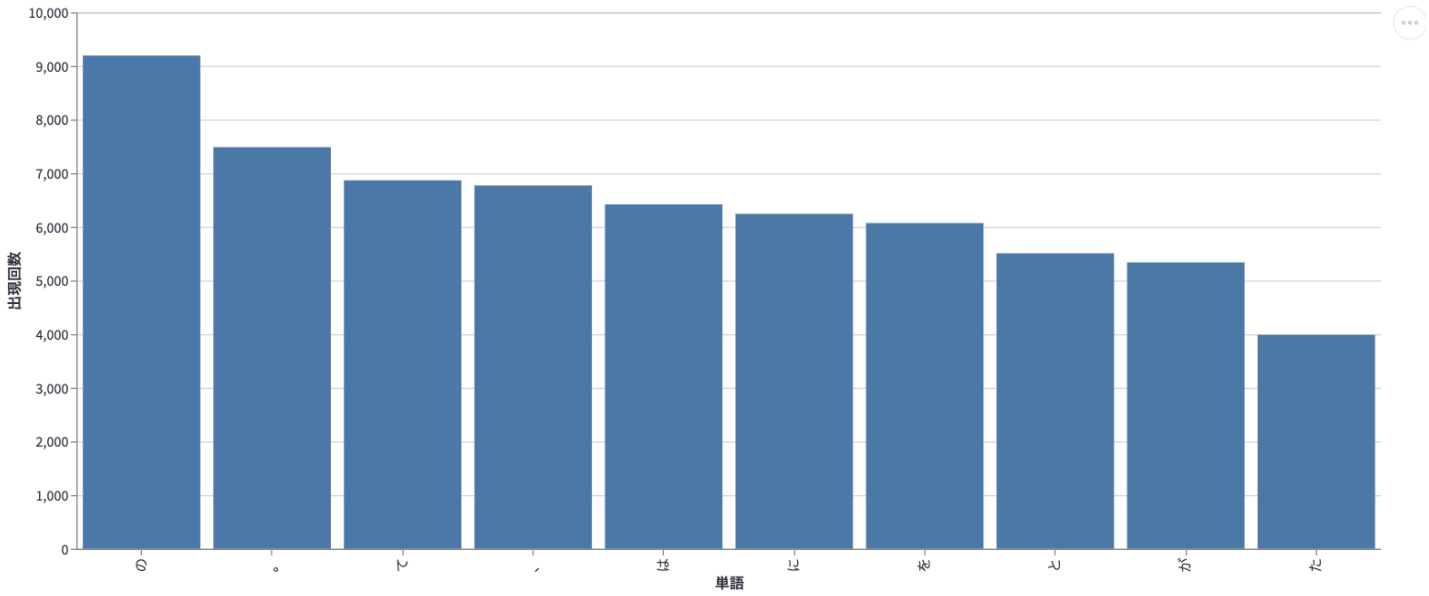
```python
# カウンタ(辞書) を、出現単語と出現回数を示すタプルのリストに変換する
tuples: list[Tuple[str, int]] = []
for k, v in d.items():
    tuples.append((k, v))
# タプルのリストを タプルの1番目の値(:= 出現回数) を元にソートします
occurrences_verbose = sorted(tuples, key=lambda t: t[1], reverse=True)
```

36. 頻度上位10語

> 出現頻度が高い10語とその出現頻度をグラフ（例えば棒グラフなど）で表示せよ

pandasとaltairを使います

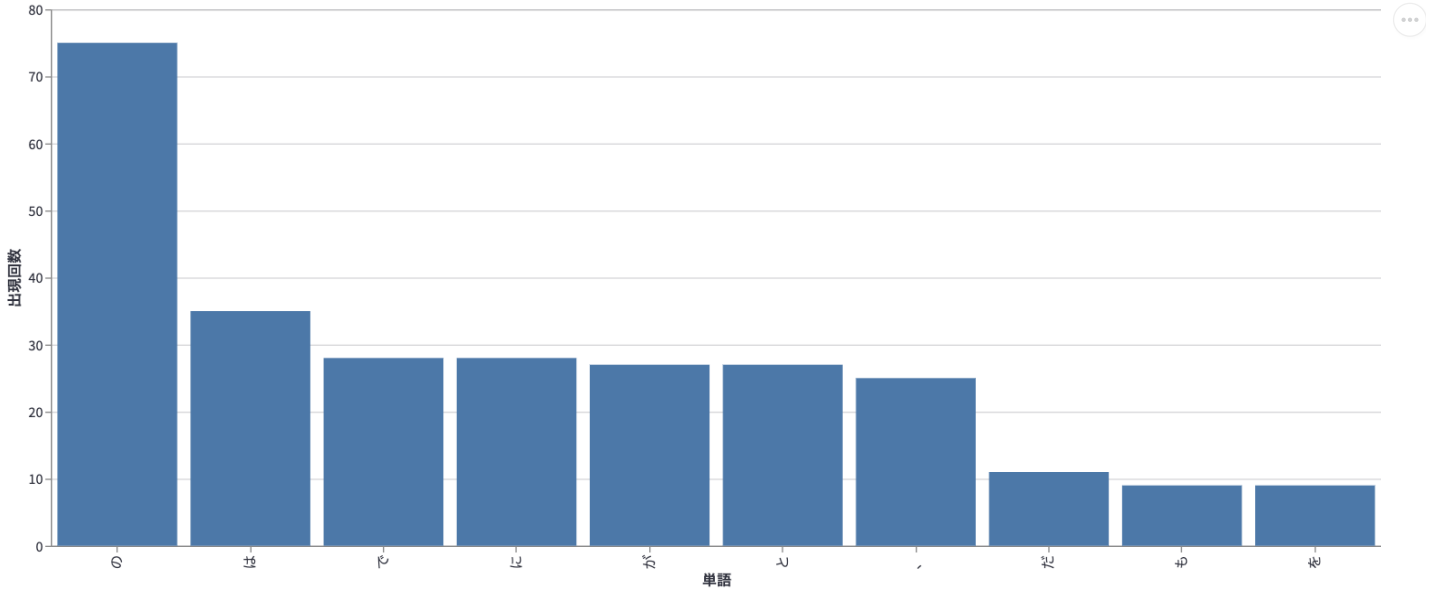
```python
df = pd.DataFrame({
    '単語': [x[0] for x in occurrences[:10]],
    '出現回数': [x[1] for x in occurrences[:10]],
})
c = alt.Chart(df).mark_bar().encode(
    x=alt.X("単語:O", sort='-y'),
    y=alt.Y("出現回数:Q"),
).properties(height=500)
st.altair_chart(c, use_container_width=True)
```



37. 「猫」と共起頻度の高い上位10語

「猫」とよく共起する（共起頻度が高い）10語とその出現頻度をグラフ（例えば棒グラフなど）で表示せよ

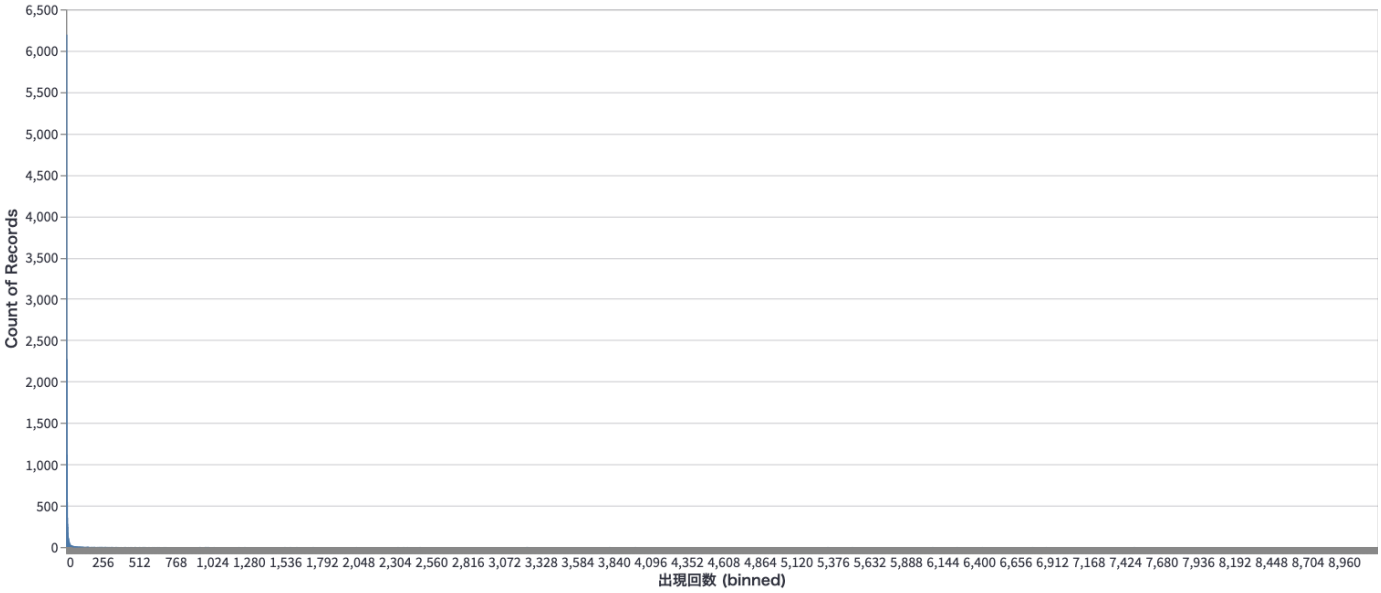
```
# 猫と共起する単語のカウンタを初期化
cat_d: defaultdict = defaultdict(int)
for sentence in sentences:
    for i, morph in enumerate(sentence):
        # 自身が '猫' でなくて、前後いずれかが '猫' のものがカウント対象
        if morph['surface'] != '猫' and (
            (i > 0 and sentence[i-1]['surface'] == '猫') or
            (i < len(sentence) - 1 and sentence[i+1]['surface'] == '猫')
        ):
            cat_d[morph['surface']] += 1
# 出現数でソート（先述した方法）
cat_co_occurrences = list(sorted([(k, v) for k, v in cat_d.items()], key=lambda x: x[1], reverse=True))
# グラフ描画
df = pd.DataFrame({
    '単語': [x[0] for x in cat_co_occurrences[:10]],
    '出現回数': [x[1] for x in cat_co_occurrences[:10]],
})
c = alt.Chart(df).mark_bar().encode(
    x=alt.X("単語:O", sort='-y'),
    y=alt.Y("出現回数:Q"),
).properties(height=500)
st.altair_chart(c, use_container_width=True)
```



38. ヒストグラム

> 単語の出現頻度のヒストグラムを描け．ただし，横軸は出現頻度を表し，1から単語の出現頻度の最大値までの線形目盛とする．縦軸はx軸で示される出現頻度となった単語の異なり数（種類数）である

```
df = pd.DataFrame({
    '単語': [x[0] for x in occurrences],
    '出現回数': [x[1] for x in occurrences],
})
c = alt.Chart(df).mark_bar().encode(
    x=alt.X(
        "出現回数:Q",
        bin=alt.Bin(step=1, extent=[0, occurrences[0][1]]),
    ),
    y=alt.Y("count():Q"),
).properties(height=500)
st.altair_chart(c, use_container_width=True)
```



39. Zipfの法則

> 単語の出現頻度順位を横軸，その出現頻度を縦軸として，両対数グラフをプロットせよ

```
c = alt.Chart(df).mark_bar().encode(
    x=alt.X(
        "出現回数:Q",
        scale=alt.Scale(type='log')
    ),
    y=alt.Y(
        "count():Q",
        scale=alt.Scale(type='log')
    ),
).properties(height=500)
st.altair_chart(c, use_container_width=True)
```