- Initialize a normal dictionary with the name “adjfreq”

- Initialize a default dictionary with the name “adjfreq” and 0 as default value (also import the library)

- Initialize a default dictionary with the name “key_to_dict” and a dictionary as default value

- Initialize a default dictionary with the name “key_to_dict” and a default dictionary(with 0 as default value) as default value

- Initialize a normal dictionary with the name “pairfreq”

```
adjfreq = {}
adjfreq = dict()
```

- Initialize a default dictionary with the name “pairfreq” and 0 as default value (also import the library)

```
from collections import defaultdict

adjfreq = defaultdict(int)
adjfreq = defaultdict(lambda:0)
```

- Initialize a default dictionary with the name “key_to_dict” and a normal dictionary as default value

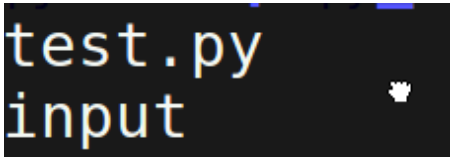
```
adjfreq = defaultdict(dict)
adjfreq = defaultdict(lambda:{})
```

- Initialize a default dictionary with the name “key_to_dict” and a default dictionary(with 0 as default value) as default value

```
adjfreq = defaultdict(lambda:defaultdict(int))
adjfreq = defaultdict(lambda:defaultdict(lambda:0))
```

```python
import sys

print(sys.argv[0])
print(sys.argv[1])

# if we run "python test.py input_file.txt", what will be printed out?
```

```python
import sys

print(sys.argv[0])
print(sys.argv[1])

# if we run "python test.py input_file.txt", what will be printed out?
```



Übung3: Kontingenztabelle

1 Aachener Agentur
1 Aachener Altbauwohnung
1 Aachener Amt

Given a file of adj and noun pairs with their frequency, complete the following code to compute pairfreq, adjfreq, and nounfreq.

```
# Häufigkeiten aus Datei einlesen
pairfreq = .... # normal dict
adjfreq = ... # defaultdict with 0 as default value
nounfreq = ... # defaultdict with 0 as default value
N = 0  # Gesamtzahl der Wortpaare
with open(input_file) as file:
    for line in file:
        ......
```

```
# Häufigkeiten aus Datei einlesen
pairfreq = {}
adjfreq = defaultdict(int)
nounfreq = defaultdict(int)
N = 0  # Gesamtzahl der Wortpaare
with open(sys.argv[1]) as file:
    for line in file:
        f,adj,noun = line.split()
        freq = int(f)
        pairfreq[adj,noun] = freq
        adjfreq[adj]   += freq
        nounfreq[noun] += freq
        N += freq
```

complete the code

```
# given
pairfreq = {("big","fish"):2, ("red","wine"):5,("white","wine"):3}
adjfreq = {"big":2, "red": 5, "white":3}
nounfreq = {"fish":2, "wine": 8}
N = 3

for wordpair in pairfreq:
    adj,noun = wordpair
    011 = ....

    01_ = ....
    02_ = ...

    0_1 = ....
    0_2 = ....

    012 = ...
    021 = ...
    022 = ...
```

```
# given
pairfreq = {("big","fish"):2, ("red","wine"):5,("white","wine"):3}
adjfreq = {"big":2, "red": 5, "white":3}
nounfreq = {"fish":2, "wine": 8}
N = 3

for wordpair in pairfreq:
    adj,noun = wordpair
    O11 = pairfreq[wordpair]

    O1_ = adjfreq[adj]
    O2_ = N - O1_

    O_1 = nounfreq[noun]
    O_2 = N - O_1

    O12 = O1_ - O11
    O21 = O_1 - O11
    O22 = O_2 - O12
```

Write code that print out each adj-noun and its log-likelihood ratio (llr), sorted from highest to lowest llr

```
# gegeben sind
pairfreq = {("big","fish"):2, ("red","wine"):5,("white","wine"):3}
llr = {("big","fish"):0.003, ("red","wine"):0.345,("white","wine"):0.00233}

# Wortpaare absteigend nach LLR-Werten sortiert ausgeben
# output format: "adj noun score"
```

Write code that print out each adj-noun and its log-likelihood ratio (llr), sorted from highest to lowest llr

```
# gegeben sind
pairfreq = {("big","fish"):2, ("red","wine"):5,("white","wine"):3}
llr = {("big","fish"):0.003, ("red","wine"):0.345,("white","wine"):0.00233}

# Wortpaare absteigend nach LLR-Werten sortiert ausgeben
# output format: "adj noun score"
for (adj,noun), score in sorted(llr.items(), key=lambda x: x[1], reverse=True):
    print(adj, noun, score)
```

Übung4: Training von Sprachidentifizierer

Compute ngramfreq dictionary

```
from collections import defaultdict

text= "Herr Präsident, verehrte Kolleginnen und Kollegen, auch ich möchte zu Beginn meines persönlichen
Redebeitrags meine Freude darüber zum Ausdruck bringen, daß Israel den Friedensprozeß mit den
Palästinensern und den Syrern wiederaufgenommen hat. Dies ist das greifbare Ergebnis des starken Willens
von Ehud Barak und seiner Regierung zur Herbeiführung des Friedens in Sicherheit."

L = 4 # Länge der verwendeten N-Gramme

# N-Gramm-Häufigkeiten(Buchstabe) berechnen
ngramfreq = ....
for ...

# expected output: ngramfreq = {'Herr': 1, 'err ': 1, 'rr P': 1, 'r Pr': 1, ' Prä': 1, ....}
```

```python
from collections import defaultdict

text= "Herr Präsident, verehrte Kolleginnen und Kollegen, auch ich möchte zu Beginn meines persönlichen
Redebeitrags meine Freude darüber zum Ausdruck bringen, daß Israel den Friedensprozeß mit den
Palästinensern und den Syrern wiederaufgenommen hat. Dies ist das greifbare Ergebnis des starken Willens
von Ehud Barak und seiner Regierung zur Herbeiführung des Friedens in Sicherheit."

L = 4 # Länge der verwendeten N-Gramme

# N-Gramm-Häufigkeiten(Buchstabe) berechnen
ngramfreq = defaultdict(int)
for i in range(len(text) - L+1):
    ngram = text[i:i+L]
    ngramfreq[ngram] += 1

# expected output: ngramfreq = {'Herr': 1, 'err ': 1, 'rr P': 1, 'r Pr': 1, ' Prä': 1, ....}
```

Thema der Prüfung ist die Berechnung der geglätteten Wahrscheinlichkeiten eines N-Gramm-Sprachmodelles über Buchstaben. Die Länge der N-Gramme ist dabei nicht fest vorgegeben. Die Wahrscheinlichkeiten sollen mit der interpolierten Backoff-Methode geglättet werden. Die Häufigkeiten der Buchstaben-N-Gramme sind dabei bereits gegeben.

Aufgabe 1) Welche Python-Datenstruktur eignet sich zur Repräsentation der bereits berechneten Häufigkeiten?

Hinweis: Es gibt hier mehrere Möglichkeiten. Wählen Sie eine aus, mit der Sie dann die folgenden Aufgaben lösen. (1 Punkt)

Discount berechnen

```
ngramfreq = {'Herr': 2, 'err ': 1, 'rr P': 4, 'r Pr': 1, ' Prä': 12}

# Berechnung des Discounts
N1 = ....
N2 = ....
```

```
ngramfreq = {'Herr': 2, 'err ': 1, 'rr P': 4, 'r Pr': 1, ' Prä': 12}

# Berechnung des Discounts
N1 = sum(1 for v in ngramfreq.values() if v == 1)
N2 = sum(1 for v in ngramfreq.values() if v == 2)
```

Aufgabe 2) Schreiben Sie eine Funktion mit dem Namen **discount**, welche die N-Gramm-Häufigkeiten als Argument bekommt und den Discount zurückgibt.

Hinweis: Den Discount erhalten Sie, indem Sie N1 durch N1 plus 2 mal N2 teilen, wobei N1 die Zahl der N-Gramme mit Häufigkeit 1 ist. (4 Punkte)

compute all context frequency given ngramfreq

```
ngramfreq = {'Herr': 2, 'err ': 1, 'rr P': 4, 'r Pr': 1, ' Prä': 12}

# Berechnung aller Kontexthäufigkeiten
contextfreq = ....
for ....

#expected output: contextfreq = {'Her': 2, 'err': 1, 'rr ': 4, 'r P': 1, ' Pr': 12}
```

$$p(w_i|w_{i-k}^{i-1}) = \frac{\max(0, f(w_{i-k}^{i}) - \delta_k)}{f(w_{i-k}^{i-1})} + \alpha(w_{i-k}^{i-1})\, p(w_i|w_{i-k+1}^{i-1})$$

Auch hier gilt: $f(w_{i-k}^{i-1}) = \sum_w f(w_{i-k}^{i-1} w)$

```
ngramfreq = {'Herr': 2, 'err ': 1, 'rr P': 4, 'r Pr': 1, ' Prä': 12}

# Berechnung aller Kontexthäufigkeiten
contextfreq = defaultdict(int)
for ngram, freq in ngramfreq.items():
    context = ngram[:-1]
    contextfreq[context] += freq
```

Schreiben Sie eine Funktion “compute_contextfreq, die “ngramfreq” dictionary as Argument bekommt und “contextfreq” zurückgibt.

Berechnen Sie der relativen Häufigkeiten mit Discount für all N-Gramme und speichern Sie die Werte in einer Dictionary “prob”.

```
ngramfreq = {'Herr': 2, 'err ': 1, 'rr P': 4, 'r Pr': 1, ' Prä': 12}
contextfreq = {'Her': 2, 'err': 1, 'rr ': 4, 'r P': 1, ' Pr': 12}
discount = 0.5


# Berechnung der relativen Häufigkeiten mit Discount
prob = ....
for ngram, f in ngramfreq.items():
    ....

# expected output: prob{'Herr': 0.002, 'err ': 0.231, ...}
```

$$p(w_i|w_{i-k}^{i-1}) = \frac{\max(0, f(w_{i-k}^{i}) - \delta_k)}{f(w_{i-k}^{i-1})} + \alpha(w_{i-k}^{i-1})\ p(w_i|w_{i-k+1}^{i-1})$$

```
ngramfreq = {'Herr': 2, 'err ': 1, 'rr P': 4, 'r Pr': 1, ' Prä': 12}
contextfreq = {'Her': 2, 'err': 1, 'rr ': 4, 'r P': 1, ' Pr': 12}
discount = 0.5


# Berechnung der relativen Häufigkeiten mit Discount
prob = {}
for ngram, f in ngramfreq.items():
    context = ngram[:-1]
    prob[ngram] = (ngramfreq[ngram] - discount) / contextfreq[context]

# expected output: prob{'Herr': 0.002, 'err ': 0.231, ...}
```

Aufgabe 3) Schreiben Sie eine Funktion **estimate_prob**, welche die N-Gramm-Häufigkeiten als Argument bekommt und dann von jeder N-Gramm-Häufigkeit den Discount abzieht und das Ergebnis durch die Kontexthäufigkeit teilt. Die Kontexthäufigkeit eines N-Grammes **g** bekommen Sie, indem Sie die Häufigkeiten aller N-Gramme summieren, die sich höchstens im letzten Element von **g** unterscheiden.

Die Werte, die für jedes N-Gramm berechnet wurden, geben Sie in einer geeigneten Datenstruktur zurück. Diese Werte werden im Folgenden als *angepasste relative Häufigkeiten* bezeichnet. (10 Punkte)

```
def estimate_prob(ngramfreq):
    contextfreq = defaultdict(int)
    for ngram, freq in ngramfreq.items():
        context = ngram[:-1]
        contextfreq[context] += freq

    prob = {}
    for ngram, f in ngramfreq.items():
        context = ngram[:-1]
        prob[ngram] = (ngramfreq[ngram] - discount) /
        contextfreq[context]

    return prob
```

Berechnen Sie N-1Gramm-Häufigkeiten

```
ngramfreq = {'Herr': 2, 'err ': 1, 'rr P': 4, 'r Pr': 1, ' Prä': 12}

# Berechnung der N-1Gramm-Häufigkeiten
sngramfreq = defaultdict(int)
for ngram, freq in ngramfreq.items():
    ....
ngramfreq = sngramfreq

# expected output: ngramfreq = {'err': 2, 'rr ': 1, 'r P': 4, ' Pr': 1, 'Prä': 12}
```

Um p von reduziertem Kontext zu berechnen

$$p(w_i|w_{i-k}^{i-1}) = \frac{\max(0, f(w_{i-k}^{i}) - \delta_k)}{f(w_{i-k}^{i-1})} + \alpha(w_{i-k}^{i-1})\, p(w_i|w_{i-k+1}^{i-1})$$

```
ngramfreq = {'Herr': 2, 'err ': 1, 'rr P': 4, 'r Pr': 1, ' Prä': 12}

# Berechnung der N-1Gramm-Häufigkeiten
sngramfreq = defaultdict(int)
for ngram, freq in ngramfreq.items():
    sngramfreq[ngram[1:]] += freq
ngramfreq = sngramfreq

# expected output: ngramfreq = {'err': 2, 'rr ': 1, 'r P': 4, ' Pr': 1, 'Prä': 12}
```

Aufgabe 4) Schreiben Sie eine Funktion **compute_backoff_factors**, welche die Tabelle mit den angepassten relativen Häufigkeiten aus der vorherigen Aufgabe als Argument erhält und die Backoff-Faktoren für die verschiedenen Kontexte berechnet, indem Sie die Wahrscheinlichkeiten aller N-Gramme in der Tabelle summiert, die bis auf ein zusätzliches letztes Element mit dem Kontext-N-Gramm identisch sind. Die Summe wird dann von 1 abgezogen, um den Backoff-Faktor des Kontextes zu erhalten. Die Tabelle mit den Backoff-Faktoren geben Sie zurück. (6 Punkte)

$$p(w_i|w_{i-k}^{i-1}) = \frac{\max(0, f(w_{i-k}^{i}) - \delta_k)}{f(w_{i-k}^{i-1})} + \alpha(w_{i-k}^{i-1})\, p(w_i|w_{i-k+1}^{i-1})$$

Aufgabe 4) Schreiben Sie eine Funktion **compute_backoff_factors**, welche die Tabelle mit den angepassten relativen Häufigkeiten aus der vorherigen Aufgabe als Argument erhält und die Backoff-Faktoren für die verschiedenen Kontexte berechnet, indem Sie die Wahrscheinlichkeiten aller N-Gramme in der Tabelle summiert, die bis auf ein zusätzliches letztes Element mit dem Kontext-N-Gramm identisch sind. Die Summe wird dann von 1 abgezogen, um den Backoff-Faktor des Kontextes zu erhalten. Die Tabelle mit den Backoff-Faktoren geben Sie zurück. (6 Punkte)

```
def compute_backoff_factors(prob):
    backoff = defaultdict(lambda: 1.0)
    for ngram, p in prob.items():
        kontext = ngram[:1]
        backoff[kontext] -= p
    return backoff
```

complete the code

```
prob = {'err': 0.2, 'rr ': 0.1, 'r P': 0.4, ' Pr': 0.01, 'Prä': 0.012}

# Dictionary prob in json Datei "parameter" speichern
with open.....:
    .....


# Load das Dictionary "prob" aus "parameter" json Datei
with open.....:
    prob = .....
```

```
prob = {'err': 0.2, 'rr ': 0.1, 'r P': 0.4, ' Pr': 0.01, 'Prä': 0.012}

# Dictionary prob in json Datei "parameter" speichern
with open("parameter","wt") as file:
    json.dump(prob, file)


# Load das Dictionary "prob" aus "parameter" json Datei
with open("parameter") as file:
    prob = json.load( file )
```

Aufgabe 5) Schreiben Sie eine Funktion **get_prob**, welche ein N-Gramm als Argument erhält und die geglättete Wahrscheinlichkeit des letztes Elementes des N-Grammes gegeben die vorherigen Elemente zurückgibt. Dazu muss zu der angepassten relativen Häufigkeit des N-Grammes das Produkt aus Backoff-Faktor und geglätteter Backoff-Wahrscheinlichkeit addiert werden. Die Backoff-Wahrscheinlichkeit wird dabei rekursiv mit derselben Funktion **get_prob** berechnet.

Die angepasste relative Häufigkeit jedes N-Grammes kann in dem globalen Dictionary **prob** nachgeschlagen werden. Diese Werte wurden in der vorherigen Aufgabe berechnet. Das Dictionary **prob** enthält aber auch die Werte für alle kürzeren N-Gramme. Die Backoff-Faktoren des Kontextes jedes N-Grammes können in einem Dictionary **backoff** nachgeschlagen werden. Die Rekursion zur Berechnung der geglätteten Wahrscheinlichkeiten soll bei Unigrammen enden.

Geben Sie bei dieser Aufgabe zusätzlich an, welche Defaultwerte die beiden Dictionaries prob und backoff liefern sollten, falls ein Key nicht definiert ist.

$$p(w_i|w_{i-k}^{i-1}) = \frac{\max(0, f(w_{i-k}^{i}) - \delta_k)}{f(w_{i-k}^{i-1})} + \alpha(w_{i-k}^{i-1})\, p(w_i|w_{i-k+1}^{i-1})$$

Aufgabe 5) Schreiben Sie eine Funktion **get_prob**, welche ein N-Gramm als Argument erhält und die geglättete Wahrscheinlichkeit des letztes Elementes des N-Grammes gegeben die vorherigen Elemente zurückgibt. Dazu muss zu der angepassten relativen

```
def get_prob( ngram ):
    if len(ngram) == 0:
        # Unigrammwahrscheinlichkeiten werden mit einer uniformen Verteilung geglättet
        return 1.0 / 1000 # Es wird von 1000 möglichen Zeichen ausgegangen
    context = ngram[0:-1] # Kontext = N-Gramm ohne letzes Zeichen
    ngram2 = ngram[1:]    # Backoff-NGramm = N-Gramm ohne erstes Zeichen
    p = ngram_prob.get(ngram, 0.0)
    bof = backoff.get(context, 1.0)
    bop = get_prob(ngram2, lang)
    return p + bof * bop
```

Aufgabe 9) Programmieraufgabe 1

Sie sollen ein Programm schreiben, das ein einfaches Hidden-Markow-Modell auf Trainingsdaten trainiert. Die Daten befinden sich in der Datei **data**, wobei jede Zeile genau einen Satz enthält. Die Zeile beginnt mit der Wortfolge. Dann folgt ein Tabulatorzeichen und dann die Tagfolge. Einzelne Wörter (bzw. Tags) sind durch Leerzeichen getrennt.

Schritte:

- Lesen Sie die Datei Zeile für Zeile ein. Extrahieren Sie die Häufigkeiten aller Wort-Tag-Paare, aller Wörter, aller Tagpaare und aller (Vorgänger-)Tags. Für die Extraktion der Tagpaar-Häufigkeiten müssen Sie Grenztags hinzufügen.
- Schätzen Sie die Kontext-Wahrscheinlichkeiten $p(t|t')$ und die lexikalischen Wahrscheinlichkeiten $p(w|t)$ mit relativen Häufigkeiten (ohne Glättung).

Aufgabe 10) Programmieraufgabe 2

Schreiben Sie eine Funktion **forward(words)**, welche eine Wortfolge (ohne Endesymbol) als Argument erhält und den Forward-Algorithmus für einen Bigramm-Tagger implementiert und die Tabelle mit den Forward-Wahrscheinlichkeiten zurückgibt.

$$\begin{aligned} \alpha_t(0) &= \begin{cases} 1 & \text{falls t das Startsymbol ist} \\ 0 & \text{sonst} \end{cases} \\ \alpha_t(i) &= \sum_{t'} \alpha_{t'}(i-1)\; p(t|t')\; p(w_i|t) \text{ für } 1 \leq i \leq n+1 \end{aligned}$$

Die Funktion besteht aus drei Schleifen über die Wortpositionen, Tags und Vorgängertags. Gegeben ist die Funktion **contextprob(t1,t2)**, welche die Wahrscheinlichkeit von *t2* gegeben *t1* zurückgibt, und die Funktion **lexprobs(w)**, welche ein Dictionary zurückliefert, welches die möglichen Tags des Wortes w als Keys und die Wahrscheinlichkeiten $p(w|tag)$ als Values hat.

(Vergessen Sie nicht das Endesymbol. Die Funktion **lexprobs(.)** liefert für das Endesymbol das Endetag mit der Wahrscheinlichkeit 1.) (7 Punkte)