Inngangur að máltækni - Verkefni 6

Guðmundur Óli Norland

Hjálparaðferðir

Hér koma aðferðir sem ég skrifaði til að hjálpa mér við lausn verkefnanna:

Mjög svo tilraunakennd mállýsing sem ég bjó til

```
def get_grammar():
    """
    Aðferð sem skilar heimatilbúinni samhengisfrjálsri mállýsingu

    S - byrjunarmerki
    SMA - smáorð
    FO - fornöfn
    SL - setningarliður
    NO - nafnorð
    SO - sagnorð
    NHM - nafnháttarmerki
    PFN - persónufornafn
    ÁFN - ábendingarfornafn
    LO - lýsingarorð
    AO - atviksorð
    ST - samtenging
    """

    grammar = nltk.CFG.fromstring(
        """
S -> SMA|FO|SOSL
SMA -> AOSL|NHMSL|STSL|FO
FO ->  PFNSL|ÁFNSL|NOSL|LOSL|SMA
NOSL -> NO SO|NO SOSL|NO ST|NO STSL
SOSL -> SO NO|SO NOSL|SO PFN|SO PFNSL|SO ÁFN|SO ÁFNSL|SO LO|SO LOSL|SO AO|SO AOSL|SO NHM|SO NHMSL
NHMSL -> NHM SO|NHM SOSL
PFNSL -> PFN SO|PFN SOSL|PFN SMA
ÁFNSL -> ÁFN NO|ÁFN NOSL|ÁFN SO|ÁFN SOSL|ÁFN LO|ÁFN LOSL
LOSL -> LO NO|LO NOSL|LO FO
AOSL -> AO AO|AO AOSL|AO NHM|AO NHMSL|AO FO
STSL -> ST NO|ST NOSL|ST PFN|ST PFNSL|ST ÁFN|ST ÁFNSL|ST LO|ST LOSL|ST AO|ST AOSL|ST NHM|ST NHMSL
NO -> 'epli'|'banani'|'gúrka'|'appelsína'|'appelsínu'|'sítróna'|'sítrónan'|'hestur'|'hesti'
SO -> 'borða'|'kastaði'|'ætla'|'langar'|'vera'|'ert'
NHM -> 'að'
PFN -> 'ég'|'við'|'þú'|'mig'|'okkur'|'þig'|'þér'|'þín'|'mín'
ÁFN -> 'þessi'|'þessum' |'þetta'
LO -> 'fallegur'|'fallega'|'fallegt'|'góður'|'betri'|'bestur'|'besta'|'best'
AO -> 'alveg'|'óskaplega'
ST -> 'og'
        """
    )

    return grammar
```

Hér er aðferð sem býr til setningu út frá samhengisfrjálsri mállýsingu

```
def rand_sent_from_grammar(grammar, start):
    """
    Endurkvæm hjálparaðferð sem býr til setningu út frá samhengisfrjálsri mállýsingu

    Breytur:
```

```
    grammar: samhengisfrjáls míllýsing
    start: byrjunartákn

    Skilagildi:
    string: setning
    """
    if type(start) is str:
        return start

    pro = grammar.productions(start)
    lengd = len(pro)
    if lengd > 1:
        randint = random.randint(0, lengd - 1)
    else:
        randint = 0
    rhs = pro[randint].rhs()

    if len(rhs) == 2:
        return (
            rand_sent_from_grammar(grammar, rhs[0])
            + " "
            + rand_sent_from_grammar(grammar, rhs[1])
        )
    else:
        return rand_sent_from_grammar(grammar, rhs[0])
```

Aðferð til að mæla tíma þáttunar

```
def timeit_parser(parser, sentence, iterations=1000):
    """
    Hjálparfall til að mæla tímann sem viðkomandi þáttari er að þátta setningu.

    Breytur:
    parser (string): heiti þáttarans í nltk
    sentence (string): setning til að þátta
    iterations (int): hversu oft á að mæla tímann

    Skilagildi:
    float: mælingartíminn í sekúndum
    """

    # uppsetning; imports og breytuskilgreining
    setup = "import nltk; from __main__ import get_grammar;"
    setup += "parser = nltk.{parser}(get_grammar())".format(parser=parser)
    # það sem við keyrum 'iterations' oft
    statement = "parser.parse({sentence})".format(sentence=sentence.split())
    # byrjum að taka tímann
    time = Timer(statement, setup).timeit(iterations)
    return time
```

Og að lokum aðferð til að prenta út tré þáttunar

```
def print_tree(parser, sent, one_tree):
    """
```

```
    Hjálparaðferð sem prentar tré fyrir þáttara + setningu

    Breytur:
    parser: þáttari
    sent (string): setning
    one_tree (boolean): prenta út fyrsta tréið einungis
    """
    for tree in parser.parse(sent.split()):
        print(tree)
        if one_tree:
            break
```

Verkefni 17

verkefni 17

```python
# Write a program to compare the efficiency of a top-down chart parser
# compared with a recursive descent parser (4). Use the same grammar
# and input sentences for both. Compare their performance using the
# timeit module (see 4.7 for an example of how to do this).
def ver17():
    grammar = get_grammar()
    sent = "þessum hesti langar alveg óskaplega að borða þetta fallega epli"
    iterations = 1000

    # byrjum á að prenta tré
    td = nltk.TopDownChartParser(grammar)
    print("Top Down tré:")
    print_tree(td, sent, True)
    rd = nltk.RecursiveDescentParser(grammar)
    print("\nRecursive Descent tré:")
    print_tree(rd, sent, True)

    # tökum tímann á top down chart þáttaranum
    td_time = timeit_parser("TopDownChartParser", sent, iterations)
    # tökum tímann á recursive descent þáttaranum
    rd_time = timeit_parser("RecursiveDescentParser", sent, iterations)
    print(
        "{iterations} runs of the top down chart parser: {td_time}".format(
            iterations=iterations, td_time=td_time
        )
    )
    print(
        "{iterations} runs of the recursive descent parser: {td_time}".format(
            iterations=iterations, td_time=rd_time
        )
    )
```

Þessi kóði skilaði mér úttakinu:

```
Top Down tré:
(S
  (FO
    (ÁFNSL
      (ÁFN þessum)
      (NOSL
        (NO hesti)
        (SOSL
          (SO langar)
          (AOSL
            (AO alveg)
            (FO
              (SMA
                (AOSL
                  (AO óskaplega)
                  (FO
                    (SMA
                      (NHMSL
                        (NHM að)
                        (SOSL
```

```
                          (SO borða)
                          (ÁFNSL
                            (ÁFN þetta)
                            (LOSL (LO fallega) (NO epli)))))))))))))))))

Recursive Descent tré:
(S
  (SMA
    (FO
      (ÁFNSL
        (ÁFN þessum)
        (NOSL
          (NO hesti)
          (SOSL
            (SO langar)
            (AOSL
              (AO alveg)
              (AOSL
                (AO óskaplega)
                (NHMSL
                  (NHM að)
                  (SOSL
                    (SO borða)
                    (ÁFNSL (ÁFN þetta) (LOSL (LO fallega) (NO epli)))))))))))))
1000 runs of the top down chart parser: 14.775574275990948
1000 runs of the recursive descent parser: 0.0035160350089427084
```

Verkefni 18

verkefni 18

```
# Compare the performance of the top-down, bottom-up,
# and left-corner parsers using the same grammar and
# three grammatical test sentences. Use timeit to log
# the amount of time each parser takes on the same sentence.
# Write a function that runs all three parsers on all three sentences,
# and prints a 3-by-3 grid of times, as well as row and column totals.
# Discuss your findings.
def ver18():
    grammar = get_grammar()
    # fyrsta röðin í töflunni (hausinn)
    table = PrettyTable(["Þáttari", "Setning 1", "Setning 2", "Setning 3", "Samtals"])

    # þrjár ólíkar setningar
    sents = []
    sents.append("þú ert hestur")
    sents.append("ég ætla að vera besta sítrónan")
    sents.append("banani og gúrka að borða appelsínu og okkur langar að vera best")

    # þáttararnir sem við ætlum að prófa
    parsers = [
        nltk.TopDownChartParser(grammar),
        nltk.BottomUpChartParser(grammar),
        nltk.LeftCornerChartParser(grammar),
    ]
    parsers_strings = [
        "TopDownChartParser",
        "BottomUpChartParser",
        "LeftCornerChartParser",
    ]
    # fylki til að halda utan um heildartíma setninga
    sent_total_time = [0, 0, 0]
    # ítrum í gegnum setningarnar fyrir hvern þáttara og geymum tímana í töflunni
    for j, parser in enumerate(parsers_strings):
        # heildartími þáttarans
        total = 0
        row = [parser]
        for i, sent in enumerate(sents):
            # prentum tré fyrir hvern þáttara + setningu
            print(
                "{parser} þáttunartré fyrir setningu {i}:".format(
                    parser=parser, i=i + 1
                )
            )
            print_tree(parsers[j], sent, True)
            print()
            # tökum svo tímann
            time = timeit_parser(parser, sent, 1000)
            total += time
            sent_total_time[i] += round(Decimal(time), 6)
            row.append(round(Decimal(time), 6))
        row.append(round(Decimal(total), 6))
        table.add_row(row)
    # bætum við heildartíma setninga í töfluna
    table.add_row(["Samtals"] + sent_total_time + [sum(sent_total_time)])
```

```
    print("Tafla:\n", table)
```

Þessi kóði skilaði mér úttakinu:

```
TopDownChartParser þáttunartré fyrir setningu 1:
(S (FO (PFNSL (PFN þú) (SOSL (SO ert) (NO hestur)))))

TopDownChartParser þáttunartré fyrir setningu 2:
(S
  (FO
    (PFNSL
      (PFN ég)
      (SOSL
        (SO ætla)
        (NHMSL
          (NHM að)
          (SOSL (SO vera) (LOSL (LO besta) (NO sítrónan))))))))

TopDownChartParser þáttunartré fyrir setningu 3:
(S
  (FO
    (NOSL
      (NO banani)
      (STSL
        (ST og)
        (NOSL
          (NO gúrka)
          (SOSL
            (SO borða)
            (NOSL (NO appelsínu) (STSL (ST og) (PFN okkur)))))))))

BottomUpChartParser þáttunartré fyrir setningu 1:
(S (FO (PFNSL (PFN þú) (SOSL (SO ert) (NO hestur)))))

BottomUpChartParser þáttunartré fyrir setningu 2:
(S
  (FO
    (PFNSL
      (PFN ég)
      (SOSL
        (SO ætla)
        (NHMSL
          (NHM að)
          (SOSL (SO vera) (LOSL (LO besta) (NO sítrónan))))))))

BottomUpChartParser þáttunartré fyrir setningu 3:
(S
  (FO
    (NOSL
      (NO banani)
      (STSL
        (ST og)
        (NOSL
          (NO gúrka)
          (SOSL
```

```
            (SO borða)
            (NOSL (NO appelsínu) (STSL (ST og) (PFN okkur)))))))))

LeftCornerChartParser þáttunartré fyrir setningu 1:
(S (FO (PFNSL (PFN þú) (SOSL (SO ert) (NO hestur)))))

LeftCornerChartParser þáttunartré fyrir setningu 2:
(S
  (FO
    (PFNSL
      (PFN ég)
      (SOSL
        (SO ætla)
        (NHMSL
          (NHM að)
          (SOSL (SO vera) (LOSL (LO besta) (NO sítrónan)))))))))

LeftCornerChartParser þáttunartré fyrir setningu 3:
(S
  (FO
    (NOSL
      (NO banani)
      (STSL
        (ST og)
        (NOSL
          (NO gúrka)
          (SOSL
            (SO borða)
            (NOSL (NO appelsínu) (STSL (ST og) (PFN okkur)))))))))

Tafla:
 +----------------------+-----------+-----------+-----------+-----------+
|        Þáttari        | Setning 1 | Setning 2 | Setning 3 |  Samtals  |
+----------------------+-----------+-----------+-----------+-----------+
|   TopDownChartParser  |  6.046793 | 11.571594 |  9.092019 | 26.710407 |
|  BottomUpChartParser  |  1.712500 |  4.712589 |  7.534075 | 13.959164 |
| LeftCornerChartParser |  0.668018 |  2.339154 |  3.750441 |  6.757613 |
|        Samtals        |  8.427311 | 18.623337 | 20.376535 | 47.427183 |
+----------------------+-----------+-----------+-----------+-----------+
```

Það sem við sjáum úr þessari töflu er að almennt séð eykst tíminn eftir því sem setningin er lengri. Áhugavert er að sjá að þótt að TopDownChart sé langhægastur; er hann sá einni sem minnkar tímann frá setningu 2 til 3.

Verkefni 30

Hér er aðferð sem býr til setningu út frá samhengisfrjálsri míllýsingu

```
def rand_sent_from_grammar(grammar, start):
    """
    Endurkvæm hjálparaðferð sem býr til setningu út frá samhengisfrjálsri míllýsingu

    Breytur:
    grammar: samhengisfrjáls míllýsing
    start: byrjunartákn

    Skilagildi:
    string: setning
    """
    if type(start) is str:
        return start

    pro = grammar.productions(start)
    lengd = len(pro)
    if lengd > 1:
        randint = random.randint(0, lengd - 1)
    else:
        randint = 0
    rhs = pro[randint].rhs()

    if len(rhs) == 2:
        return (
            rand_sent_from_grammar(grammar, rhs[0])
            + " "
            + rand_sent_from_grammar(grammar, rhs[1])
        )
    else:
        return rand_sent_from_grammar(grammar, rhs[0])
```

verkefni 30

```
# Write a function that takes a grammar (such as the one defined in 3.1)
# and returns a random sentence generated by the grammar. (Use grammar.start()
# to find the start symbol of the grammar; grammar.productions(lhs) to get the
# list of productions from the grammar that have the specified left-hand side;
# and production.rhs() to get the right-hand side of a production.)
def ver30():
    grammar = get_grammar()
    for i in range(10):
        print(rand_sent_from_grammar(grammar, grammar.start()))
```

Þessi kóði gaf t.d. þetta úttak (vil benda á að þessa míllýsingu má bæta mjög svo):

```
að langar
og hestur og
að borða
óskaplega bestur appelsína og óskaplega alveg að ætla
óskaplega að
sítróna og þú óskaplega að
alveg óskaplega alveg
og þessum
við ert við ætla hestur ætla
alveg alveg að kastaði ég borða fallegt
```

Verkefni 19

Read up on "garden path"sentences. How might the computational work of a parser relate to the difficulty humans have with processing these sentences? http://en.wikipedia.org/wiki/Garden_path_sentence

---

Garden-path áhrifin eiga sér stað þegar setning inniheldur tvíræða/óljósa merkingu sem lesandinn túlkar á ákveðinn hátt, sem svo breytist algjörlega þegar hann hefur klárað setninguna og les hana aftur. *The complex houses married and single soldiers and their families.* þetta er gott dæmi um slíka setningu. Flestir lesa *complex* sem lýsingarorð og *houses* sem nafnorð, hinsvegar er það ekki rökrétt þegar öll setningin hefur verið lesin. *Complex* er nafnorð (byggingasamstæða) og *houses* er sögn (að hýsa). Í þessu felst hvers vegna þáttarar og menn eiga sameiginlega erfitt með slíkar setningar. Þáttarinn/lesandinn les setninguna orð fyrir orð og reynir að mynda samhengi úr því sem komið er. Semsagt þegar einungis *The complex houses* er lesið mun þáttarinn/heilinn reyna að mynda samhengi úr þessum orðum sem hann er vanastur að sjá. Einungis eftir að öll setningin hefur verið lesin og heildarsamhenginu náð er hægt að meta hvort þáttunin/lesskilningurinn hafi verið réttur.

Verkefni 28

verkefni 28

```python
# Process each tree of the Treebank corpus sample nltk.corpus.treebank and extract
# the productions with the help of Tree.productions(). Discard the productions that
# occur only once. Productions with the same left hand side, and similar right hand
# sides can be collapsed, resulting in an equivalent but more compact set of rules.
# Write code to output a compact grammar.
def ver28():
    treebank = nltk.corpus.treebank
    # ítrum yfir öll tréin og geymum reglurnar í lista
    productions = []
    for fileid in treebank.fileids():
        ps = treebank.parsed_sents(fileid)
        for tree in ps:
            productions += tree.productions()
    # notum bara það sem kemur oftar en einu sinni
    productions = [x for x, count in Counter(productions).items() if count > 1]
    # sækjum lista af öllum ólíkum millitáknum
    millitakn = list(set([x.lhs() for x in productions]))
    # sýum út reglur fyrir millitákn sem eru með alltof margar reglur
    productions_new = []
    for m in millitakn:
        # sækjum allar reglurnar fyrir þetta millitákn
        m_productions = [x for x in productions if x.lhs() == m]
        lengd = len(m_productions)
        threshold = 20
        if lengd > threshold:
            sample = random.sample(m_productions, k=int(lengd / 10))
            productions_new += sample
        else:
            productions_new += m_productions

    # búum til cfg fyrir reglurnar
    grammar = nltk.CFG(nltk.grammar.Nonterminal("S"), productions_new)
    # prófum að búa til random setningar og prenta þær út
    for i in range(20):
        print(rand_sent_from_grammar(grammar, grammar.start()))
```

Þessi kóði skilaði mér úttakinu:

```
*-21
Springs earns
Its force
around these
plus
*T*-77
-LCB-
through worst
On immediately
through also
Securities illegally
``
since Improvement scientists
through stability sounds
```

```
our mainly THE carried soon better
cautious Engineers
*T*-76
Any strings Vargas shrinks
Its Only
ranged
```

Allur kóðinn

```
import nltk
import random

from collections import Counter
from decimal import Decimal
from prettytable import PrettyTable
from timeit import Timer


def get_grammar():
    """
    Aðferð sem skilar heimatilbúinni samhengisfrjálsri míllýsingu

    S - byrjunarmerki
    SMA - smáorð
    FO - fornöfn
    SL - setningarliður
    NO - nafnorð
    SO - sagnorð
    NHM - nafnháttarmerki
    PFN - persónufornafn
    ÁFN - ábendingarfornafn
    LO - lýsingarorð
    AO - atviksorð
    ST - samtenging
    """

    grammar = nltk.CFG.fromstring(
        """
S -> SMA|FO|SOSL
SMA -> AOSL|NHMSL|STSL|FO
FO ->  PFNSL|ÁFNSL|NOSL|LOSL|SMA
NOSL -> NO SO|NO SOSL|NO ST|NO STSL
SOSL -> SO NO|SO NOSL|SO PFN|SO PFNSL|SO ÁFN|SO ÁFNSL|SO LO|SO LOSL|SO AO|SO AOSL|SO NHM|SO NHMSL
NHMSL -> NHM SO|NHM SOSL
PFNSL -> PFN SO|PFN SOSL|PFN SMA
ÁFNSL -> ÁFN NO|ÁFN NOSL|ÁFN SO|ÁFN SOSL|ÁFN LO|ÁFN LOSL
LOSL -> LO NO|LO NOSL|LO FO
AOSL -> AO AO|AO AOSL|AO NHM|AO NHMSL|AO FO
STSL -> ST NO|ST NOSL|ST PFN|ST PFNSL|ST ÁFN|ST ÁFNSL|ST LO|ST LOSL|ST AO|ST AOSL|ST NHM|ST NHMSL
NO -> 'epli'|'banani'|'gúrka'|'appelsína'|'appelsínu'|'sítróna'|'sítrónan'|'hestur'|'hesti'
SO -> 'borða'|'kastaði'|'ætla'|'langar'|'vera'|'ert'
NHM -> 'að'
PFN -> 'ég'|'við'|'þú'|'mig'|'okkur'|'þig'|'þér'|'þín'|'mín'
ÁFN -> 'þessi'|'þessum' |'þetta'
LO -> 'fallegur'|'fallega'|'fallegt'|'góður'|'betri'|'bestur'|'besta'|'best'
AO -> 'alveg'|'óskaplega'
ST -> 'og'
        """
    )

    return grammar
```

```
def rand_sent_from_grammar(grammar, start):
    """
    Endurkvæm hjálparaðferð sem býr til setningu út frá samhengisfrjálsri míllýsingu

    Breytur:
    grammar: samhengisfrjáls míllýsing
    start: byrjunartákn

    Skilagildi:
    string: setning
    """
    if type(start) is str:
        return start

    pro = grammar.productions(start)
    lengd = len(pro)
    if lengd > 1:
        randint = random.randint(0, lengd - 1)
    else:
        randint = 0
    rhs = pro[randint].rhs()

    if len(rhs) == 2:
        return (
            rand_sent_from_grammar(grammar, rhs[0])
            + " "
            + rand_sent_from_grammar(grammar, rhs[1])
        )
    else:
        return rand_sent_from_grammar(grammar, rhs[0])


def timeit_parser(parser, sentence, iterations=1000):
    """
    Hjálparfall til að mæla tímann sem viðkomandi þáttari er að þátta setningu.

    Breytur:
    parser (string): heiti þáttarans í nltk
    sentence (string): setning til að þátta
    iterations (int): hversu oft á að mæla tímann

    Skilagildi:
    float: mælingartíminn í sekúndum
    """

    # uppsetning; imports og breytuskilgreining
    setup = "import nltk; from __main__ import get_grammar;"
    setup += "parser = nltk.{parser}(get_grammar())".format(parser=parser)
    # það sem við keyrum 'iterations' oft
    statement = "parser.parse({sentence})".format(sentence=sentence.split())
    # byrjum að taka tímann
    time = Timer(statement, setup).timeit(iterations)
    return time


def print_tree(parser, sent, one_tree):
```

```
    """
    Hjálparaðferð sem prentar tré fyrir þáttara + setningu

    Breytur:
    parser: þáttari
    sent (string): setning
    one_tree (boolean): prenta út fyrsta tréið einungis
    """
    for tree in parser.parse(sent.split()):
        print(tree)
        if one_tree:
            break


# Write a program to compare the efficiency of a top-down chart parser
# compared with a recursive descent parser (4). Use the same grammar
# and input sentences for both. Compare their performance using the
# timeit module (see 4.7 for an example of how to do this).
def ver17():
    grammar = get_grammar()
    sent = "þessum hesti langar alveg óskaplega að borða þetta fallega epli"
    iterations = 1000

    # byrjum á að prenta tré
    td = nltk.TopDownChartParser(grammar)
    print("Top Down tré:")
    print_tree(td, sent, True)
    rd = nltk.RecursiveDescentParser(grammar)
    print("\nRecursive Descent tré:")
    print_tree(rd, sent, True)

    # tökum tímann á top down chart þáttaranum
    td_time = timeit_parser("TopDownChartParser", sent, iterations)
    # tökum tímann á recursive descent þáttaranum
    rd_time = timeit_parser("RecursiveDescentParser", sent, iterations)
    print(
        "{iterations} runs of the top down chart parser: {td_time}".format(
            iterations=iterations, td_time=td_time
        )
    )
    print(
        "{iterations} runs of the recursive descent parser: {td_time}".format(
            iterations=iterations, td_time=rd_time
        )
    )


# Compare the performance of the top-down, bottom-up,
# and left-corner parsers using the same grammar and
# three grammatical test sentences. Use timeit to log
# the amount of time each parser takes on the same sentence.
# Write a function that runs all three parsers on all three sentences,
# and prints a 3-by-3 grid of times, as well as row and column totals.
# Discuss your findings.
def ver18():
    grammar = get_grammar()
```

```
    # fyrsta röðin í töflunni (hausinn)
    table = PrettyTable(["Þáttari", "Setning 1", "Setning 2", "Setning 3", "Samtals"])

    # þrjár ólíkar setningar
    sents = []
    sents.append("þú ert hestur")
    sents.append("ég ætla að vera besta sítrónan")
    sents.append("banani og gúrka borða appelsínu og okkur")

    # þáttararnir sem við ætlum að prófa
    parsers = [
        nltk.TopDownChartParser(grammar),
        nltk.BottomUpChartParser(grammar),
        nltk.LeftCornerChartParser(grammar),
    ]
    parsers_strings = [
        "TopDownChartParser",
        "BottomUpChartParser",
        "LeftCornerChartParser",
    ]
    # fylki til að halda utan um heildartíma setninga
    sent_total_time = [0, 0, 0]
    # ítrum í gegnum setningarnar fyrir hvern þáttara og geymum tímana í töflunni
    for j, parser in enumerate(parsers_strings):
        # heildartími þáttarans
        total = 0
        row = [parser]
        for i, sent in enumerate(sents):
            # prentum tré fyrir hvern þáttara + setningu
            print(
                "{parser} þáttunartré fyrir setningu {i}:".format(
                    parser=parser, i=i + 1
                )
            )
            print_tree(parsers[j], sent, True)
            print()
            # tökum svo tímann
            time = timeit_parser(parser, sent, 1000)
            total += time
            sent_total_time[i] += round(Decimal(time), 6)
            row.append(round(Decimal(time), 6))
        row.append(round(Decimal(total), 6))
        table.add_row(row)
    # bætum við heildartíma setninga í töfluna
    table.add_row(["Samtals"] + sent_total_time + [sum(sent_total_time)])
    print("Tafla:\n", table)


# Write a function that takes a grammar (such as the one defined in 3.1)
# and returns a random sentence generated by the grammar. (Use grammar.start()
# to find the start symbol of the grammar; grammar.productions(lhs) to get the
# list of productions from the grammar that have the specified left-hand side;
# and production.rhs() to get the right-hand side of a production.)
def ver30():
    grammar = get_grammar()
    for i in range(10):
```

```
        print(rand_sent_from_grammar(grammar, grammar.start()))


# Process each tree of the Treebank corpus sample nltk.corpus.treebank and extract
# the productions with the help of Tree.productions(). Discard the productions that
# occur only once. Productions with the same left hand side, and similar right hand
# sides can be collapsed, resulting in an equivalent but more compact set of rules.
# Write code to output a compact grammar.
def ver28():
    treebank = nltk.corpus.treebank
    # ítrum yfir öll tréin og geymum reglurnar í lista
    productions = []
    for fileid in treebank.fileids():
        ps = treebank.parsed_sents(fileid)
        for tree in ps:
            productions += tree.productions()
    # notum bara það sem kemur oftar en einu sinni
    productions = [x for x, count in Counter(productions).items() if count > 1]
    # sækjum lista af öllum ólíkum millitáknum
    millitakn = list(set([x.lhs() for x in productions]))
    # sýum út reglur fyrir millitákn sem eru með alltof margar reglur
    productions_new = []
    for m in millitakn:
        # sækjum allar reglurnar fyrir þetta millitákn
        m_productions = [x for x in productions if x.lhs() == m]
        lengd = len(m_productions)
        threshold = 20
        if lengd > threshold:
            sample = random.sample(m_productions, k=int(lengd / 10))
            productions_new += sample
        else:
            productions_new += m_productions

    # búum til cfg fyrir reglurnar
    grammar = nltk.CFG(nltk.grammar.Nonterminal("S"), productions_new)
    # prófum að búa til random setningar og prenta þær út
    for i in range(20):
        print(rand_sent_from_grammar(grammar, grammar.start()))


ver17()
ver18()
ver30()
ver28()
```