**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ**
**НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ УКРАИНЫ**
**“КИЕВСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ”**
**ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ**

**Кафедра математических методов кибернетической безопасности**

КУРСОВАЯ РАБОТА

Дисциплина: «Интеллектуальные методы обработки информации»

Направление подготовки: 8.04030101 «Прикладная математика»

Тема: «Интеллектуальные методы обработки информации»

Выполнил студент группы ФИ-51м

Кригин Валерий Михайлович

Проверила:

Бояринова Юлия Евгеньевна

_____

*(подпись)*

Оценка:

_____

**Киев 2015**

# ОГЛАВЛЕНИЕ

# 1 ЗАКОН ЦИПФА

## 1.1 Закон Ципфа

Отношение ранга слова $R$, то есть его номер в списке слов, отсортированных по частоте в порядке убывания, к частоте слова $f$, является постоянным

$$Z = R \cdot f,$$

где $f$ — частота слова в тексте, а $Z$ — коэффициент Ципфа. Значит,

$$f = \frac{Z}{R}.$$

## 1.2 Задание

Под понятием “отфильтровать текст” тут и далее будут подразумеваться следующие действия:

1) очистить текст от всех символов кроме букв и пробелов;
2) буквы привести в нижний регистр, между словами оставить по одному пробелу.
3)

В лабораторной работе нужно

1) взять текст (желательно на русском языке) длиной более нескольких сотен килобайт;
2) отфильтровать текст;
3) составить частотный словарь слов — каждому слову текста сопоставить количество его повторений в тексте;

4) отсортировать частоты в порядке убывания;
5) изобразить полученные значения на графике, выбрав логарифмический масштаб для оси ординат и абсцисс;
6) построить степенную линию тренда и убедиться, что график похож на прямую линию, за исключением, возможно, “хвостов” с обеих концов.

### 1.3 Фильтр

На Perl написан фильтр, который
1) делает заглавные буквы строчными;
2) убирает всё кроме пробелов, символов табуляций, переносов строк и т.п.;
3) превращает все символы, которе не являются буквами, в пробел, также предотвращает появление двух пробелов подряд.

Вход считывается из stdin, выход происходит в stdout.

Листинг 1.1 — filter.pl

```
1 #!/usr/bin/perl -w -CAS
2 use utf8;
3
4 $_ = lc join('', <>);
5
6 s/[^\p{L}\s]//g;
7 s/[\s]+/ /g;
8
9 print;
```

### 1.4 Частотный словарь

На Python написан скрипт, который составляет частотный словарь и выводит его в формате csv. Полученный результат можно открыть в программе для работы с электронными таблицами для построения графиков.

Вход считывается из stdin, выход происходит в stdout.

Листинг 1.2 — counter.py

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  from sys import stdin
5  from os import linesep
6
7  words = ' '.join([l.strip() for l in stdin]).split(' ')
8
9  counts = {}
10 for key in set(words):
11     counts[key] = 0
12 for w in words:
13     counts[w] += 1
14
15 result = sorted(counts.iteritems(),key=lambda x: x[1],
16                  reverse=True)
17
18 print linesep.join('%s,%d'%r for r in result)
```

## 1.5 График

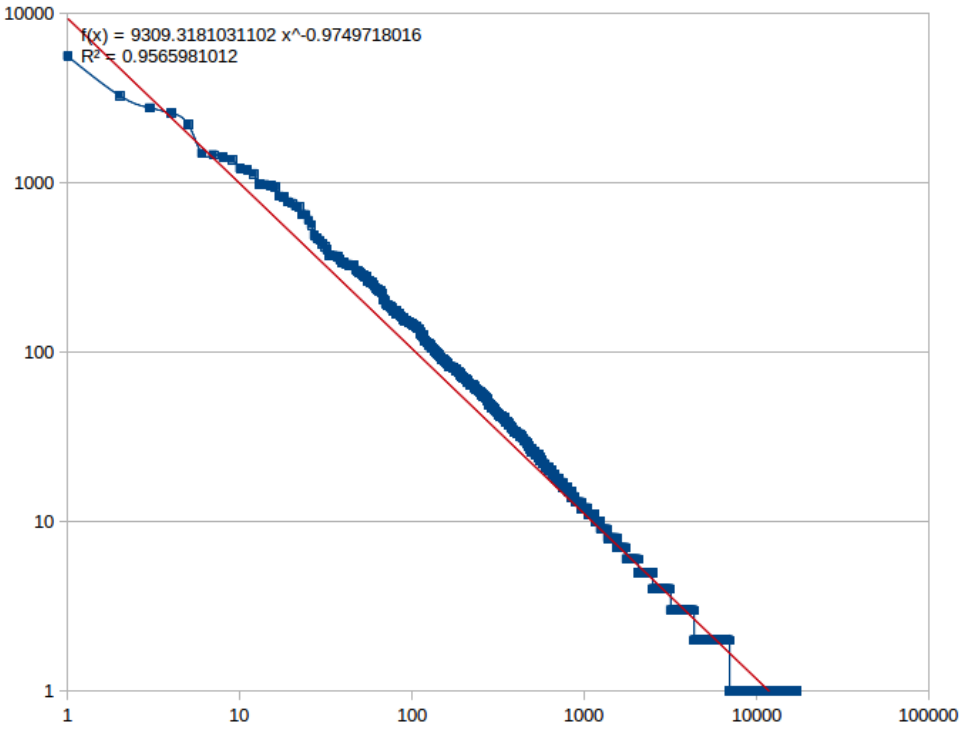


Рисунок 1.1 — Результат

# 2 ЗАКОН ХИПСА

## 2.1 Закон Хипса

Объём словаря уникальных слов $\nu(n)$ для текста длиной $n$ связан с длиной текста следующим соотношением

$$\nu(n) = \alpha \cdot n^{\beta},$$

где $\alpha$ и $\beta$ — эмпирические константы, которые разнятся от языка к языку, и для европейских языков колеблятся в пределах от 10 до 100 и от 0.4 до 0.6 соответственно.

## 2.2 Задание

В лабораторной работе нужно

1) взять текст (желательно на русском языке) длиной более нескольких сотен килобайт;
2) отфильтровать текст;
3) построить зависимость количества уникальных слов в тексте от его размера; для этого достаточно использовать один и тот же текст, изымать из него всё больше и больше слов с каждой итерацией, и подсчитывать число уникальных слов на каждом шаге;
4) изобразить полученные значения на графике;
5) построить степенную линию тренда и убедиться, что полученные параметры $\alpha$ и $\beta$ близки к теоретическим значениям.

### 2.3 Фильтр

На Perl написан фильтр, который

1) делает заглавные буквы строчными;
2) убирает всё кроме пробелов, символов табуляций, переносов строк и т.п.;
3) превращает все символы, которе не являются буквами, в пробел, также предотвращает появление двух пробелов подряд.

Вход считывается из stdin, выход происходит в stdout.

Листинг 2.1 — filter.pl

```
1 #!/usr/bin/perl -w -CAS
2 use utf8;
3
4 $_ = lc join('', <>);
5
6 s/[^\p{L}\s]//g;
7 s/[\s]+/ /g;
8
9 print;
```

### 2.4 Частотный словарь

На Python написан скрипт, который считает зависимость между объёмом текста и объёмом словаря уникальных слов и выводит его в формате csv. Полученный результат можно открыть в программе для работы с электронными таблицами для построения графиков.

Листинг 2.2 — counter.py

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  from sys import stdin
5
6  words = ' '.join([l.strip() for l in stdin]).split(' ')
7
8  found = []
9
10 for i, w in enumerate(words):
11     if w not in found:
12         found.append(w)
13     print '%d,%d'%(i+1, len(found))
```

### 2.5 График

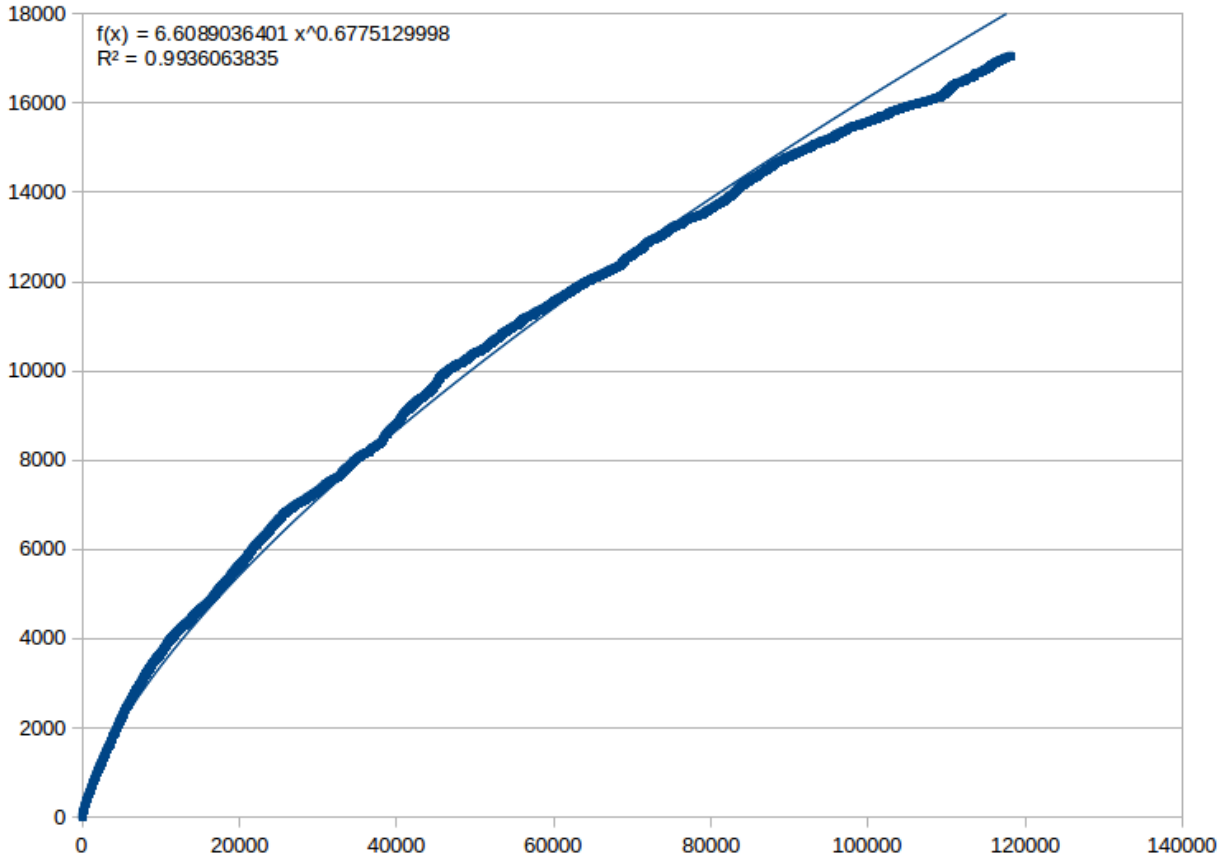


Рисунок 2.1 — Результат

# 3 $TF - IDF$

## 3.1 $TF - IDF$

Для $i$ слова ($n$-граммы) индексы $TF$ и $IDF$ считаются по следующим формулам, где $D$ — множество документов, $n_k$ — количество повторений $k$ слова ($n$-граммы) в текущем документе

$$TF_i = \frac{n_i}{\sum_k n_k},$$
$$IDF_i = \log \frac{|D|}{|\{d \mid t_i \in d \in D\}|}.$$

Сам индекс $TF - IDF$ является произведением индексов $TF$ и $IDF$

$$TF - IDF_i = TF_i \cdot IDF_i$$

## 3.2 Задание

### 3.2.1 Основное задание

В лабораторной работе нужно

1) взять текст (желательно на русском языке) длиной более нескольких сотен килобайт;
2) отфильтровать текст;
3) подсчитать $TF - IDF$ для каждого слова;
4) изобразить полученные результаты в виде таблицы, отсортировав по значению $TF - IDF$ в порядке убывания.

То же самое нужно проделать с биграммами и триадами слов. Например,

в тексте “мама мыла раму” биграммы следующие: “мама мыла” и “мыла раму”.

### 3.2.2 Стоп-слова (шумовые слова)

Стоп-слова — те слова, которые не несут смысловую нагрузку. К ним относятся предлоги, частицы и прочее, если анализируемый документ не является учебником русского языка.

Список стоп-слов можно найти в интернете. Например, в разделе 12.9.4 Full-Text Stopwords документации к MySQL 5.5 находится список англоязычных шумовых слов.

Для увеличения скорости и уменьшения объёма обрабатываемых данных

1) при подсчёте $TF - IDF$ для слов можно выбросить из рассмотрения те, которые находятся в списке стоп-слов; например, слово “не” имеет мало смысла в сказке о царе Салтане, чего не скажешь о слове “лебедь”;
2) при подсчёте $TF - IDF$ для биграмм следует исключать те биграммы, которые содержат в себе шумовые слова; например, биграмма “я пришёл” имеет мало смысловой нагрузки, но биграмма “пришёл домой” скажет больше;
3) при подсчёте $TF - IDF$ для триад следует исключать те элементы, которые оканчиваются или начинаются на шумовые слова; скажем, “и она решила” мало о чём говорит, триада “она решила пойти” скажет больше, но “решила пойти домой” несёт определённый смысл.

## 3.3 Фильтр

На Perl написан фильтр, который

1) делает заглавные буквы строчными;

2) убирает всё кроме пробелов, символов табуляций, переносов строк и т.п.;
3) превращает все символы, которе не являются буквами, в пробел, также предотвращает появление двух пробелов подряд.

Вход считывается из stdin, выход происходит в stdout.

Листинг 3.1 — filter.pl

```
1 #!/usr/bin/perl -w -CAS
2 use utf8;
3
4 $_ = lc join('', <>);
5
6 s/[^\p{L}\s]//g;
7 s/[\s]+/ /g;
8
9 print;
```

### 3.4 Счётчик $TF - IDF$

На Python написан скрипт, который считает $TF-IDF$ для слов и выводит их в формате csv.

Полученный результат можно открыть в программе для работы с электронными таблицами для сортировки и фильтрации.

Листинг 3.2 — counter.py

```
1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3
4 from sys import stdin, argv
```

```
5  from os import linesep
6  from math import log
7  from stoplist import stop_list
8
9
10 def get_count(words):
11     tfs = {}
12     for key in set(words):
13         tfs[key] = 0
14     for w in words:
15         tfs[w] += 1
16     return tfs
17
18 def group_n_grams(words, n):
19     if n < 2:
20         return [w for w in words if w not in stop_list]
21     return ['␣'.join(w for w in words[i:i+n])
22             for i in range(len(words)-n)
23             if words[i+n-1] not in stop_list
24                and words[i] not in stop_list]
25
26 if __name__ == '__main__':
27     n_grams_length = 1
28
29     if len(argv) > 1:
30         n_grams_length = int(argv[1])
31
```

```
32      texts = ([l.strip().split('␣') for l in stdin])
33      names = map(lambda text: text[0], texts)
34      texts = map(lambda text: group_n_grams(text[1:],
35                                             n_grams_length), texts)
36
37
38      tfs = map(get_count, texts)
39
40
41      idf = {}
42      for word in set(sum(texts, [])):
43          idf[word] = 0
44
45
46      for tf in tfs:
47          for word in tf:
48              idf[word] += 1
49
50
51      logN = log(len(texts))
52      for word in idf:
53          idf[word] = logN - log(idf[word])
54
55
56      tf_idfs = []
57      for i, tf in enumerate(tfs):
58          tf_idfs.append({})
```

```
59          for word in tf:
60              tf_idfs[i][word] = tf[word] * idf[word] / len(tf)
61
62      result = [(names[i], word, tf_idf[word])
63                for i, tf_idf in enumerate(tf_idfs)
64                for word in tf_idf]
65      result = sorted(result, key=lambda x: x[2], reverse=True)
66      print linesep.join('%s,%s,%f'%(r) for r in result)
```

### 3.5 Результат

На 3.1 изображены первые 18 строк таблицы со значениями $TF - IDF$ для слов из 144 документов автора Льва Николаевича Толстого, 27 документов Фёдора Михайловича Достоевского и 31 документа Александра Сергеевича Пушкина, отсортированных по значению $TF - IDF$ в порядке убывания.

Объём документов Толстого 18MB, Достоевского 7.6MB, Пушкина — 2.8MB. Фильтрация происходит соответственно 10.3, 3.2 и 2 секунды. Далее каждый документ имеет только один перенос строки, который говорит об окончании документа, и их можно объединить в один файл. Подсчёт $TF - IDF$ происходит за 6.5 секунд, на выходе получается *.csv* файл объёмом 39MB.

На 3.2 изображены первые 18 строк таблицы с биграммами, а на 3.3 изображены первые 18 строк таблицы с триадами.

| № | Книга | Слово | $TF - IDF$ |
|---|---|---|---|
| 1 | TolstoiVorobei | воробей | 0.910196 |
| 2 | TolstoiEchizayac | ёж | 0.723855 |
| 3 | TolstoiVorobei | лён | 0.717333 |
| 4 | TolstoiTelenoknaldu | телёнок | 0.649992 |
| 5 | TolstoiLetuchayamysh | летучая | 0.645765 |
| 6 | PushkinKamennyigost | гуан | 0.625634 |
| 7 | TolstoiVolgaiVazuza | волга | 0.616940 |
| 8 | TolstoiFilipok | филипок | 0.603935 |
| 9 | TolstoiShatiDon | шат | 0.591983 |
| 10 | TolstoiShakalyislon | слон | 0.570469 |
| 11 | TolstoiVolgaiVazuza | вазуза | 0.570408 |
| 12 | TolstoiPesnyaprosrachenienarekeChernoi | bis | 0.523350 |
| 13 | TolstoiZaicyilyagushki | зайцы | 0.517134 |
| 14 | TolstoiLetuchayamysh | мышь | 0.507136 |
| 15 | PushkinKamennyigost | дон | 0.471297 |
| 16 | TolstoiMyshi | кота | 0.467739 |
| 17 | TolstoiShatiDon | дон | 0.454054 |
| 18 | TolstoiSobakaieeten | собака | 0.441651 |

Рисунок 3.1 — Результат для слов

| № | Книга | Биграмма | $TF - IDF$ |
|---|---|---|---|
| 1 | TolstoiLetuchayamysh | летучая мышь | 2.051165 |
| 2 | PushkinKamennyigost | дон гуан | 0.803516 |
| 3 | TolstoiMyshi | кота спастись | 0.558765 |
| 4 | PushkinKamennyigost | дона анна | 0.461864 |
| 5 | TolstoiSobakaieeten | бросила своё | 0.408328 |
| 6 | TolstoiSobakaieeten | своё мясо | 0.408328 |
| 7 | TolstoiSobakaieeten | тень собака | 0.408328 |
| 8 | TolstoiSobakaieeten | своё волною | 0.408328 |
| 9 | TolstoiSobakaieeten | мясо несёт | 0.408328 |
| 10 | TolstoiShatiDon | шат иваныч | 0.407217 |
| 11 | TolstoiShatiDon | дон иваныч | 0.407217 |
| 12 | TolstoiVolk | ай ай | 0.372557 |
| 13 | TolstoiVorobei | лён воробей | 0.366087 |
| 14 | TolstoiSobakaieeten | зубах несла | 0.355009 |
| 15 | TolstoiSobakaieeten | волною унесло | 0.355009 |
| 16 | TolstoiSobakaieeten | собака шла | 0.355009 |
| 17 | TolstoiSobakaieeten | собака мясо | 0.355009 |
| 18 | TolstoiSobakaieeten | кинулась отнимать | 0.355009 |

Рисунок 3.2 — Результат для биграмм

| № | Книга | Триада | $TF-IDF$ |
|---|---|---|---|
| 1 | TolstoiOtecisynovya | отец и сыновья | 0.865335 |
| 2 | TolstoiMyshi | коту на шею | 0.663533 |
| 3 | TolstoiZaicyilyagushki | зайцы и лягушки | 0.629335 |
| 4 | TolstoiTelenoknaldu | телёнок на льду | 0.497650 |
| 5 | TolstoiSobakaieeten | своё волною унесло | 0.408328 |
| 6 | TolstoiSobakaieeten | бросила своё мясо | 0.408328 |
| 7 | TolstoiSobakaieeten | несёт она бросила | 0.408328 |
| 8 | TolstoiSobakaieeten | тень собака шла | 0.408328 |
| 9 | TolstoiSobakaieeten | собака мясо несёт | 0.408328 |
| 10 | TolstoiVorobei | птицы не послушались | 0.393205 |
| 11 | TolstoiShakalyislon | шакалы и слон | 0.384593 |
| 12 | TolstoiSobakaieeten | унесло и осталась | 0.355009 |
| 13 | TolstoiSobakaieeten | собаки того мяса | 0.355009 |
| 14 | TolstoiSobakaieeten | воде и подумала | 0.355009 |
| 15 | TolstoiSobakaieeten | зубах несла мясо | 0.355009 |
| 16 | TolstoiSobakaieeten | мясо и кинулась | 0.355009 |
| 17 | TolstoiSobakaieeten | дощечке через речку | 0.355009 |
| 18 | TolstoiSobakaieeten | несла мясо увидала | 0.355009 |

Рисунок 3.3 — Результат для триад

# 4 ГРАФИЧЕСКОЕ ПРЕДСТАВЛЕНИЕ СЕТИ СЛОВ

## 4.1 Задание

Изобразить граф, отображающий взаимосвязи между словами, биграммами и триадами. Привести его матрицу весов.

## 4.2 Прорисовка графа

Листинг 4.1 — draw.py

```
1  from graph_tool.all import *
2  from words import data as words
3  from bigrams import data as bigrams
4  from trigrams import data as trigrams
5  from math import log10, log
6  from sys import argv
7
8
9  def clear_entries(entries, containers, treshold=0):
10     non_needed_entries = set(entries.keys())
11     non_needed_containers = set(containers.keys())
12     for entry in entries:
13         if entries[entry] <= treshold:
14             continue
15         exists = False
16         for container in containers:
17             if entry in container:
```

```
18                  if container in non_needed_containers:
19                      non_needed_containers.remove(container)
20                  non_needed_entries.remove(entry)
21                  break
22      for container in non_needed_containers:
23          del containers[container]
24      for entry in non_needed_entries:
25          del entries[entry]
26
27
28  def get_vertex(g, name, word, vertices):
29      # If we already have this vertex, just use it from cache
30      if name in vertices:
31          return vertices[name]
32      # Otherwise we have to create new one
33      a = g.add_vertex()
34      word[a] = name
35      # Add the vertex to cache
36      vertices[name] = a
37      return a
38
39
40  def build_graph(g, entries, containers,
41                  weight, word, color, vertices,
42                  treshold=0, last_word=-1, width_scale = 3.0,
43                  entry_color='red', container_color='red'):
44      min_tfidf = min(containers.values())
```

```
45      def add_to_container(entry, container):
46          a = get_vertex(g, entry, word, vertices)
47          color[a] = entry_color
48          b = get_vertex(g, container, word, vertices)
49          color[b] = container_color
50          e = g.add_edge(a, b)
51          # Set weight for new edge (tf-idf)
52          # Just empirical formula
53          raw_weight = containers[container]/min_tfidf
54          weight[e] = log(log(raw_weight)+1) * width_scale + 1
55      for entry in entries:
56          if entries[entry] < treshold:
57              continue
58          for container in containers:
59              if entry in container:
60                  add_to_container(entry, container)
61      return
62
63
64  def init_graph():
65      # Create directed graph
66      g = Graph(directed=True)
67      # Create 'weight' property for edge:
68      # will contain tf-idf
69      weight = g.new_edge_property('float')
70      # Create 'word' property for vertex:
71      # will contain string with current word
```

```
72      word = g.new_vertex_property('string')
73      color = g.new_vertex_property('string')
74      return g, weight, word, color, {}
75
76
77  if __name__ == '__main__':
78      img_name='output.png'
79      if len(argv) > 2:
80          if argv[1] in ['-w', '--word']:
81              words = dict(words.items()[:int(argv[2])])
82              clear_entries(words, bigrams)
83          elif argv[1] in ['-b', '--bigram']:
84              bigrams = dict(bigrams.items()[:int(argv[2])])
85              clear_entries(bigrams, trigrams)
86          elif argv[1] in ['-t', '--triad']:
87              trigrams = dict(trigrams.items()[:int(argv[2])])
88              clear_entries(bigrams, trigrams)
89      if len(argv) > 3:
90          img_name = argv[3]
91      g, weight, word, color, vertices = init_graph()
92      # Dictionary with existent vertices (cache)
93      clear_entries(words, bigrams)
94      clear_entries(bigrams, trigrams)
95      build_graph(g, entries=words, containers=bigrams,
96                  color=color, weight=weight, word=word,
97                  vertices=vertices,
98                  entry_color='red', container_color='blue')
```

```
 99      build_graph(g, entries=bigrams, containers=trigrams,
100                  color=color, weight=weight, word=word,
101                  vertices=vertices,
102                  entry_color='blue', container_color='purple')
103      # Draw the graph;
104      # Weight is responsible for edges widths
105      # Word contains labels for vertices
106      # graph_draw(g, vertex_font_size=10, edge_pen_width=weight,
107      #              vertex_text=word, vertex_fill_color=color,
108      #              vertex_text_position=0, output=img_name,
109      #              output_size=(300, 500))
110      # Alternative
111      # graph_draw(g, edge_pen_width=weight, vertex_text=word,
112      #              node_first=True, vertex_text_position=0,
113      #              vertex_size=20, vertex_shape='double_square')
114      # Or even
115      state=minimize_nested_blockmodel_dl(g)
116      draw_hierarchy(state, vertex_text=word,
117              vertex_text_position=1, edge_pen_width=weight,
118              vertex_fill_color=color,
119              output=img_name, output_size=(800, 600))
```