Официальный сайт суда

Парсинг и загрузка данных

Текстовые файлы постановлений

Словарь постановлений

Формирование грубой выборки

Список всего, что похоже на ссылки

Сопоставление грубой выборки с словарем постановлений. Формирование чистой выборки

Ориентированный граф взаимных ссылок

Выдача результата в виде графа и его отрисовки

Ориентированный граф взаимных ссылок

Визуализация графа взаимных ссылок

user

Обращается к официальному сайту Конституционного суда РФ и парсит оттуда уникальные названия постановлений, даты их принятия, ссылки на их тексты

GetResolutionHeaders

No argument

return value: {'uid': {'date': 'date string', 'url': 'link to pdf file', 'title':'string - full title of document'}}

Словарь:
Ключ - строка, Uid - уникальное название постановления, например '32-П/2018'.
Значения - словарь:
ключ - строка, 2 возможных значения:
'date' - дата публикации, 'link'- ссылка на текст публикации (в формате pdf)
'title' - полное название документа

Получает на вход словарь заголовков (от метода GetResolutionHeaders). Для каждого постановления загружает файл текста постановления и преобразует его в текстовый формат. Возвращает словарь заголовков, записывая в url путь к файлу

LoadResolutionsTexts

{'uid': {'date': 'date string', 'url': 'link to pdf file', 'title': 'full title of document'}}
decreesHeaders

return value: {'uid': {'date' : 'date string', 'url' : 'path to text file' }}
write to files: texts of resolutions. Each file name is the resolution uid. Format of files is txt.

GetRudeLinks

string uid
string textFileName

{'uid':[strings : everything that is a bit like the link to resolution]}

Получает на вход UID постановления и имя файла с его текстом. Отсекает часть текста "Особые мнения". Из оставшейся части выбирает все фразы, хоть слегка похожие на ссылку на постановление

Словарь из одной пары, где ключ - uid, а значение - список строк, возможно являющихся ссылками.

GetRudeLinksForMultipleDocuments

{'uid' : {'data': 'smth', 'url':'filenames', 'title' : 'Full title of documets '}}

{'uid':[strings : everything that is a bit like the link to resolution]}

Получает словарь постановлений и для каждого постановления в нем вызывает GetRudeLinks, формируюя результирующий словарь

Словарь постановлений.

Словарь, где ключи - uid, а значения - списки строк, возможно являющихся ссылками.

ProcessPeriod

string earlierDate
string laterDate

write to file: graph as list of nodes and adjacency matrix
write to file and show to user: picture of the graph

Получает период (как начальную и конечную дату) и возвращает граф цитирования для всех постановлений за этот период. Граф сохраняется в файл как список вершин и матрица смежности, а так же отрисовывается и выводится на экран. Картинка тоже сохраняется в файл.

StartProcessWith

'uid'
int depth

write ot file: graph as list of nodes ans adjacency matrix
write to file and show to user: picture of graph

Начинает рекурсивную обработку постановлений, начиная с заданного. depth - глубина рекурсии, то есть число постановлений в цепочке цитирований. depth = 1 - поиск только ссылок из начального документа

GetCleanLinks

{'uid':[strings : everything that is a bit like the link to resolution]}
{'uid' : {'data': 'smth', 'url':'filenames', 'title' : 'Full title of documets '}}

{'uid':[strings :final links]})

Получает грубую выборку и словарь заголовков постановлений. Возвращает список подтвержденных ссылок

GetLinksGraph

{'uid' : {'data': 'smth', 'url':'filenames'}}

([string uids], [[adjacency matrix
]])

Получает словарь заголовков с уже скачанными файлами, формирует по нему ориентированный граф следующим способом: по заголовку формируется грубая выборка (методом GetRudeSamples), по ней формируется чистая выборка (методом GetCleanSamples). Из массивов цитированных постановлений формируется матрица смежности графа, где узлами являются uid постановлений, а направления ребер идут от цитирующего к цитируемому постановлению.

WebCrawler

-имя участника

-имя участника

Link

from: uid, str
to: uid, str
context: words around the link, str
count: count of documents, int

constructor(from, context)
Add(context)

DocumentHeader

uid: str
source_url: web url, str
full_title: titles of the documents, str or list of str
document_type: str
count: count of the documents, int
date: Date
text_location: str or list of str

constructor(uid, documentType, sourceUrl, fullTitle, textLocation=None)
Add(sourceUrl, fullTitle, textLocation=None)

Can be:
filename- name of local file with text of the document;
None - text not received yet;
database: just a word 'database' which indicates that text of the document can be received from database by uid
if a few database text location specified, after the word 'database' should be '|' and exactly uid, which allow to receive text of the document from database.

//iterators

GraphHeaderFilter

allowedCountTo
allowedCountFrom

HeaderFilter

allowedTypes: set of types strings
alloweDates: set of Date

constructor (types:list or set of types str, dates: list or set of Date)
constructor (types:list or set of types str, firstDate: Date, lastDate:Date)
check_header(h: header): bool
get_filtered_headers(headers: dict of uid : header): dict of uid: header

LinkGraph

nodes: list of header
edges: list of tuple: (uid, uid, weight)

get_subgraph(headerFilters, linkFilters): link_graph

GraphEdgeFilter

allowedTypesFrom: set of types str
allowedTypesTo: set of types str
weightDiapozon: tupe (min weight, max weight)

-имя участника

LinkFinding

-имя участника
get_rough_links_for_multiple_documents(headers, webCrawler)
get_rough_links(header, webCrawle)

LinkAnalyzing

get_clean_links(links, headers):{linkStr: links}
get_links_graph(links): Link_graph

ApiModule

process_period(firstDate, lastDate, headersFilters=None, edgeFilters=None, nodeFilters=None)

start_process_with(uid, depth, headersFilters=None, edgeFilters=None, nodeFilters=None))