TP Crawling

François Lecerf

1. Crawls élémentaires avec wget

1.1 Téléchargez la page http://www.freepatentsonline.com/8591332.html (description d'un brevet) avec wget. Ouvrez le fichier ainsi téléchargé et comparez-le à la page Web d'origine. Qu'observez-vous ?

Le fichier téléchargé n'a pas les informations de mise en page (celles ci se trouvent dans les feuilles de style que téléchargerait un navigateur web)

```
sh question1.py
```

1.2 Avec les options « -r --wait=1 », wget permet de télécharger de manière récursive un ensemble de pages Web, avec un délai de politesse d'une seconde entre deux requêtes vers le même serveur. Essayez cette option pour télécharger un ensemble de descriptions de brevets depuis FPO. Que constatez-vous ? Vous pouvez interrompre le crawl avec CTRL+C.

wget télécharge de façon récursive toutes les pages qu'il trouve à partir de la première page. Le traitement n'en finit pas car il télécharge trop de pages (par défaut wget prend 5 sous niveaux)

```
sh question2.sh
```

1.4 wget dispose d'une option « -np » permettant de ne télécharger que les pages Web contenues dans le même répertoire que la page Web initiale (ou dans un sous-répertoire de celui-ci). Essayez de mettre en œuvre cette option pour télécharger des articles de FPO, notamment celui de la partie 1. Vous devriez parvenir à récupérer une partie d'entre eux, mais il reste des requêtes inutiles faites par le crawler. Commentez.

On évite de télécharger des pages web situées au dessus du niveau d'arborescence de requête avec l'option np.

Malgré cela, wget télécharge beaucoup de pages qui n'ont aucune pertinence par rapport à notre requête:

- des images ./images
- du javascript ./js

Les liens contenant des brevêts ont un formatage spécifique que l'on exploitera via une regex à la question suivante.

```
sh question4.sh
```

1.5 wget dispose d'une option « --reject-regex » permettant de ne pas télécharger les pages correspondant à un certain motif (expression régulière); utiliser cette option pour exclure des pages qui vous semblent inutiles en indiquant quel(s) motifs de caractère(s) les URL que vous souhaitez rejeter comporte(nt) (attention, si vous utilisez des caractères spéciaux pour les expressions régulières, il faut les préfixer d'un antislash). Laissez le crawl travailler un moment : combien avez-vous récupérés de description de brevets? Réussissez-vous à ne récupérer que des descriptions de brevet ? Si oui, avec quelle expression régulière ?

J'ai préféré utiliser une requête qui ne renvoit que les liens qui vérifient la regex (-accept-regex):

```
regex='/[0-9]{7}.html'
```

J'ai pu récupérer 150 brevets en 2 minutes

```
sh question5.sh
```

2. Crawl systématique par programme Python

Nous allons maintenant utiliser Python3 pour faire un crawl plus systématique des articles qui nous intéressent. Téléchargez le fichier tasktimer.py et un exemple d'utilisation dans etape0.py à ces adresses :

- https://perso.telecom-paristech.fr/moissina/TPCRAWL/etape0.py
- https://perso.telecom-paristech.fr/moissina/TPCRAWL/modules/tasktimer.py
  (ici vous pouvez aussi envisager d'utiliser scrapy)

2.1 Commencez par faire un programme minimal qui fait la requête http://www.freepatentsonline.com/result.html?sort=relevance&srch=top&query_txt=video&submit=&patents=on sur le site FPO pour charger (par exemple en utilisant urllib) et sauver dans un fichier la réponse du serveur FPO à cette requête. Cette requête

interroge le serveur pour obtenir une liste de brevets concernant la vidéo.

Appelez le programme etape1.py

tasktimer.py

```
# -*- encoding: utf-8 -*-
# test access to the ISTEX API
#  author: JC Moissinac (c) 2016 inspired by J.F.Sebastian

from threading import Event, Thread
from functools import partial


def call_repeatedly(interval, max_iter, func, *args):
    stopped = Event()

    def loop(max_iter):
        internalstop = False
        while (not stopped.wait(interval)) and (not internalstop) and \
              (max_iter > 0):
            # the first call is in `interval` secs
            internalstop = func(*args)
            max_iter = max_iter - 1
    thread = Thread(target=partial(loop, max_iter=max_iter))
    thread.start()
    thread.join()
    return stopped.set

# The event is used to stop the repetitions:
# cancel_future_calls = call_repeatedly(5, print, "Hello, World")
# do something else here...
# cancel_future_calls() # stop future calls
```

etape1.py

```
# -*- encoding: utf-8 -*-
from tasktimer import call_repeatedly
from urllib.request import urlopen
from urllib.parse import urlencode
import re


def getHtml(html, params):
    params = urlencode(params)
    response = urlopen(html + "?" + params)
    return response



html = 'http://www.freepatentsonline.com/result.html'
```

```
params = {'sort': 'relevance',
          'srch': 'top',
          'query_txt': 'video',
          'submit': '',
          'patents': 'on'
          }
outputFile = "../data/search.html"


response = getHtml(html, params)

with open(outputFile, mode='w') as file:
    file.write(response.read().decode('utf-8'))
```

2.2 Faire un programme –inspiré des programmes etape0 et etape1- qui charge une liste de pages. Appelez ce programme etape2. Maintenant, au lieu de sauver le contenu reçu, vous allez commencer à analyser le contenu. Pour commencer vous pouvez utiliser pytidylib pour corriger les défauts éventuels des pages chargées (optionnellement, si votre compte permet d'utiliser BeautifulSoup, vous pouvez faire les choses avec cet outil). Ensuite, vous allez utiliser html5lib pour accéder à des parties de la page. Avec le XPATH './/xh :a', récupérez tous les tags présents dans la page (xh est le préfixe associé au namespace http://www.w3.org/1999/xhtml).

- Combien y a-t-il de liens dans la page initiale puis dans d'autres pages que votre programme visite?

Il y a 50 liens par page

- Ajoutez chaque lien trouvé à la liste des liens à charger.

Rajouté dans un set

- Ajoutez une condition d'arrêt lorsque 30 pages html ont été chargées (vous pouvez tester avec d'autres valeurs que 30).

Clause max_results

- A ce moment-là, combien de pages ont été visitées ?

30 pages de 50 liens soit 1500 brevets

- Combien de liens sont dans la file d'attente des liens à traiter ?

1500

Donnez le nom etape3.py à ce programme.

etape3.py

```
# -*- encoding: utf-8 -*-
from bs4 import BeautifulSoup
from tasktimer import call_repeatedly
from urllib.request import urlopen
from urllib.parse import urlencode
import re


def getSearchResults(params, max_results):
    '''
    Return max_result first outputs
    '''
    patentLinks = set()
    for p in range(1, max_results):
        params['p'] = p
        searchPage = getHtml(query, params)
        tmp_set = getResultContents(searchPage)
        patentLinks = patentLinks.union(tmp_set)
    return patentLinks


def getHtml(html, params):
    params = urlencode(params)
    response = urlopen(html + "?" + params)
    return response


def getResultContents(page):
    '''
    input:
        page: html search contents (string)
    output:
        links: links matching regex (set)
    '''
    tmp_links = set()
    try:
        # soup = BeautifulSoup(page, 'html.parser')
        soup = BeautifulSoup(page, 'html5lib')
        tmp_tags = soup.find_all("div", attrs={"class": "legacy-container"})
        # tmp_tags = soup.find_all("table", attrs={"class": "listing_table"})
        tmp_links = set([a.get("href") for a in tmp_tags[0].find_all('a') if
                         a.get("href") is not None])
        return tmp_links
    except AttributeError as e:
        print(e)
        return tmp_links


domain = 'http://www.freepatentsonline.com'
query = domain + '/result.html'
params = {'sort': 'relevance',
          'srch': 'top',
          'query_txt': 'video',
```

```
          'patents': 'on'
          }
max_results = 30
outputFile = "../data/search.html"


to_process = getSearchResults(params, max_results)
to_process = list(map(lambda url: domain + url, to_process))

# write output
with open(outputFile, mode='w') as file:
    for link in to_process:
        file.write(link + '\n')
```

2.3 Ajouter au programme précédent du code pour éviter de charger plusieurs fois la même URL.

Ce n'est pas nécessaire : les sorties se trouvent dans un set, et sont donc déjà dédoublonnées

2.4 Ajouter au programme précédent du code:

- pour déterminer si une page est une page de description d'un brevet (+> une page qui nous intéresse)

On filtrera les liens morts en testant le retour de urlopen. Les pages contiennent toutes un brevet (filtré en amont)

Programme etape4.py

```
# -*- encoding: utf-8 -*-
from bs4 import BeautifulSoup
import pandas as pd
from tasktimer import call_repeatedly
from urllib.request import urlopen
from urllib.parse import urlencode, unquote
import re


# fonction appelée périodiquement
def urlcall(toBeProcessed):
    if toBeProcessed["to_process"]:
        crawl_page(toBeProcessed["domain"],
                   toBeProcessed["to_process"],
                   toBeProcessed["processed"])
        print(toBeProcessed["to_process"])
        print(toBeProcessed["processed"])
        print("Not done yet")
        return False
    else:
        print("Done")

        return True
```

```
def getSearchResults(params, max_results):
    '''
    Return max_result first outputs
    '''
    patentLinks = set()
    for p in range(1, max_results):
        params['p'] = p
        searchPage = getHtml(query, params)
        tmp_set = getResultContents(searchPage)
        patentLinks = patentLinks.union(tmp_set)
    return patentLinks


def getHtml(html, params):
    params = urlencode(params)
    response = urlopen(html + "?" + params)
    return response


def getResultContents(page):
    '''
    input:
        page: html search contents (string)
    output:
        links: links matching regex (set)
    '''
    tmp_links = set()
    try:
        # soup = BeautifulSoup(page, 'html.parser')
        soup = BeautifulSoup(page, 'html5lib')
        tmp_tags = soup.find_all("div", attrs={"class": "legacy-container"})
        # tmp_tags = soup.find_all("table", attrs={"class": "listing_table"})
        tmp_links = set([a.get("href") for a in tmp_tags[0].find_all('a') if
                         a.get("href") is not None])
        return tmp_links
    except AttributeError as e:
        print(e)
        return tmp_links


def getPatentLinks(page):
    '''
    input:
        page: html patent contents (string)
    output:
        links: other patents linked to this patent (set)
    '''
    tmp_links = set()
    # pattern = re.compile(r'^/wiki/(?!.*redlink=1).*$', re.IGNORECASE)
    pattern = re.compile(r'.*[0-9]{7}.html$', re.IGNORECASE)

    try:
```

```
        soup = BeautifulSoup(page, 'html.parser')
        tmp_links = set([a.get("href") for a in soup.find_all('a') if
                         a.get("href") is not None])
        tmp_links = set([link for link in tmp_links if pattern.match(link)])
        return tmp_links
    except AttributeError as e:
        print(e)
        return tmp_links


def crawl_page(domain, to_process, processed):
    '''
    input:
        domain: root of the domain (string)
        to_process: pages that must be seen (set)
        processed: pages that were already seen (set)
    output:
       Finished: True/False
    '''
    input_link = to_process.pop()
    output_links = getPatentLinks(urlopen(domain + input_link))

    processed.add(input_link)
    to_process = to_process | (output_links - processed)

    return to_process is None


def getPatentContents(page):
    '''
    input:
        page: html patent contents (string)
    output:
        links: dictionary giving patents informations
    '''
    disp_elm_titles = set(
          ["Title:", "Inventors:", "Abstract:", "Application Number:",
           "Publication Date:", "Filing Date:", "Export Citation:",
           "Assignee:", "Primary Class:", "Other Classes:",
           "International Classes:"])

    tmp_dict = dict()
    # pattern = re.compile(r'^/wiki/(?!.*redlink=1).*$', re.IGNORECASE)
    try:
        soup = BeautifulSoup(page, 'html.parser')
        tmp_tags = soup.find_all("div", attrs={"class": "disp_doc2"})

        for tag in tmp_tags:
            key_tag = tag.find("div", attrs={"class": "disp_elm_title"})

            if (key_tag is not None):
                key = unquote(key_tag.text).strip()
```

```
                if key in disp_elm_titles:
                    value_tag = tag.find("div",
                                         attrs={"class": "disp_elm_text"})
                    if (value_tag is not None):
                        value = unquote(value_tag.text).strip()
                        tmp_dict[key] = value

        return tmp_dict

    except AttributeError as e:
        print(e)
        return tmp_dict


if __name__ == '__main__':

    domain = 'http://www.freepatentsonline.com'
    query = domain + '/result.html'
    params = {'sort': 'relevance',
              'srch': 'top',
              'query_txt': 'video',
              'patents': 'on'
              }
    max_results = 5
    outputFile = "../data/export.json"

    to_process = getSearchResults(params, max_results)
    processed = set()

    call_repeatedly(0.5, 60, urlcall, {"to_process": to_process,
                                       "processed": processed,
                                       "domain": domain})

    processed = list(map(lambda url: domain + url, processed))

    patent_contents = []
    for link in processed:
        patent_contents.append(getPatentContents(urlopen(link)))
    df_patents = pd.DataFrame(patent_contents)

    df_patents["Filing Date:"] = pd.to_datetime(df_patents["Filing Date:"],
                                                format="%m/%d/%Y")
    df_patents["Publication Date:"] = pd.to_datetime(
                                               df_patents["Publication Date:"],
                                               format="%m/%d/%Y")

    df_patents.to_json(path_or_buf=outputFile, orient='records')
    print(df_patents)
```

2.5 A l'aide de html5lib et d'expressions régulières sélectionnez certaines URL qui respectent certains modèles (vous pourrez

mettre au point vos expressions régulières à l'aide de https://regex101.com/). Programme etape6.py. Vous décrirez quels liens sont supposés sélectionner vos expressions.

Déjà réalisé dans l'étape 4

getPatentLinks: on conserve les chaines se terminant par 7 chiffres et .html

```
pattern = re.compile(r'.*[0-9]{7}.html$', re.IGNORECASE)
```

2.6 Choisissez d'autres informations dans ces pages dont vous allez chercher à récupérer la valeur (nom d'auteur de brevet ? entreprise ?...). Sauvez les données obtenues dans une structure json.

Déjà réalisé dans l'étape 4

2.7 Mettez en place un processus qui va chercher, sur les principes précédents, dans FPO, au moins 3 brevets dont un des inventeurs est français et qui a à voir avec le multimédia.

Programme etape8.py. Vous indiquerez comment lancer le processus et créerez un fichier nommé multimedia.txt qui contiendra les 3 URLs des pages décrivant ces brevets, chacune sur une ligne.