M2-DNR2I - Système / D3.js / Node.js

François Rioult`Francois.Rioult@info.unicaen.fr`

10 septembre 2013

Table des matières

Chapitre 1

Traitement de données avec Python

1.1 Pros :

- très concis, ce qui ne veut pas dire qu'il ne faut pas détailler les instructions, au risque d'être abscons
- langage objet assez complet (héritage, polymorphisme, modèles à class variable)
- énormément de librairies (mais moins qu'en Java, entre autres sur le web sémantique)
- interprété, ce qui facilite les expérimentations
- programmation fonctionnelle

1.2 Cons :

Il manque des éléments de modélisation objet comme les notions de :

- variables publiques / privées : tout est public, la convention veut qu'on préfixe les variables privées avec un tiret bas.
- interface
- operateurs

Ses avantages peuvent également s'avérer être des inconvénients :

- le code étant interprété, on peut ne découvrir des erreurs de syntaxe qu'à l'exécution
- l'interprétation du code peut nuire aux performances. Si c'est ce que l'on recherche, on préférera C++.

1.3 Conclusion

Au niveau de PHP, mais avec une syntaxe beaucoup plus puissante. Sa vocation généraliste permet des développements de projets ambitieux, ce n'est pas qu'un langage

de script.

On y retrouve la puissance de AWK mais Python facilite le développement de scripts, pour gérer des données complexes et utiliser les librairies standards.

1.4 Trucs

– la déclaration d'une méthode statique doit être précédée de `@staticmethod`.

Chapitre 2

Animation de données avec D3.js

D3 pour Data-Driven Documents permet l'animation interactive de données, ce qui facilite la navigation documentaire. C'est Mootools ou jQuery spécialisé dans la génération de graphique.

La différence essentielle est que D3 permet de maintenir un lien avec les données originales tout au long de la section : il est aisé par exemple de trier une colonne a posteriori en accédant à la liste des enregistrements dans D3 puis de demander l'instruction de tri :

```
var sortDataButton = document.getElementById('sortname');
            sortDataButton.onclick = function (event){
                d3.selectAll('.composer').sort(function (a, b){
                   return a.name.localeCompare(b.name);
                });
            };
```

C'est cet accès caractéristique qui lui confère toute sa puissance car le programmeur est déchargé de toute manipulations et se concentre sur l'accès aux données.

2.1 Principes

À partir de données JSON (c'est plus pratique comme format, mais XML marche également à grand renfort de documentGetElementById), D3.js fournit une librairie JavaScript dédiée à la visualisation de données sur différents Layouts, permet les animations et gère l'interaction.

D3 utilise massivement SVG, ce qui assure une bonne compatibilité avec les interfaces actuelles. En interaction avec JavaScript, le code est ouvert : il y a une profusion d'exemples sur le web qu'il est aisé d'adapter.

D3 s'interface particulièrement facilement avec Node.js, ce qui permet le développement d'applications ambitieuses en dehors du trio Apache/PHP/SQL. La donnée est

gérée par JSON, ce qui permet une interaction plus riche que HTTP. Les calculs sont effectués sur le client, ce qui décharge le serveur et l'infrastructure.

Les données peuvent parfaitement être générées dynamiquement par les scripts, la librairies contient tous les éléments pour faire de la géométrie.

2.2 Méthode

On s'efforcera dans un premier temps, d'adopter le découpage suivant :

1. préparation des données JSON : même si D3 fournit des filtres puissants, il est sage dans un premier temps d'accorder beaucoup de soin à la préparation des données. On génèrera avec des langages de script comme Python, des tables de jointure contenant toute l'information. Cela permet de décharger D3 des filtrages en précalculant l'effort, et facilite la conception ;
2. choisir le layout : c'est l'étape importante, fortement limitée par le type des données (multi-valuées, arbre, graphe).
3. définir l'ergonomie : animation et surtout interaction

Programmation :

1. sélection des noeuds du DOM : définir le point d'insertion des données ;
   `insert = d3.selectAll("#menu").selectAll("div");`
2. effectuer la jointure entre les noeuds sélectionnés et les données ;
   `enter = insert.data(root.items).enter();`
3. par défaut, la méthode data() retourne la jointure interne, qui est souvent vide lorsque l'on insère pour la première fois dans le DOM. Il faut donc sélectionner la partie de jointure droite (les données qui ne correspondent pas au DOM) à l'aide de enter(). Cette méthode n'instancie les nouveaux noeuds que lorsqu'elle est suivie de append(), insert(), select() et call(). La section exit() contient la jointure gauche (les noeuds du node auxquels ne sont pas affectées de données).

2.3 Layout

Les choix d'un layout est très contraint par le type de données que l'on souhaite afficher : (du plus simple au plus complexe)

données numériques : en toute généralité, on pourra valoriser des données multi-valuées ;

arbre : pour une hiérarchie, affichée de façon circulaire afin de limiter les écarts de représentations ;

graphe : c'est la forme la plus générale, utilisée par des relations.

Les sommets peuvent être valués ce qui est mis à profit dans Pack pour générer des cercles de rayon variable.

Nom D3	type	données	description
Chord	Diagramme d'accord	graphe	en entrée une matrice de relations.
Cluster	Dendogramme	graphe	
Force	interaction gravitationnelle	graphe	utilise une analogie gravitationnelle pour disposer les sommets d'un **graphe**, les arêtes permettent de fixer une distance,ce qui augmente la masse.
Histogram	histogramme	numérique	représentation statistique de l'information
Pack	hiérarchie de cercles	arbre	
Pie	camemberts ou donuts	numérique	
Stack	graphe d'évolution	numérique	pour des séries temporelles
Tree	arbre circulaire	arbre	
Treemap	pavage récursif	arbre valué	

2.4 Animation

2.5 Interaction

Chapitre 3

Node.js

On souhaite écrire un petit framework gérant des données (JSON ou XML), permettant d'effectuer des requêtes et d'afficher le contenu à l'aide de D3. Pour l'instant, le contenu de base est la requête XQuery, ici `auteurs.xq`, qui référence l'adresse absolue du fichier de données.

3.1 Données

(fichier `graph.son`) :

```
{
    "vertices": [
        {
            "attribute": 4,
            "id": 0,
            "name": "José A. Blakeley"
        },
        {
            "attribute": 11,
            "id": 1,
            "name": "Yuri Breitbart"
        },
    ]
}
```

3.2 Gestionnaire de Données

utilisation de BaseX, la base de données XML/Xquery. BaseX est un logiciel léger facile à installer (package baseX sur Ubuntu). Il utilise le remarquable langage XQuery pour les requêtes qui motive ce passage par XML. On pourra préférer MongoDB, plus

directement orienté JSON mais également plus lourd (comparable à SQL) en terme de déploiement.

```
let $return :=
        let $fileName := '/var/www/rioultf/svn/m2dnr2i-systeme/code/d3/cours/jso
        let $file := file:read-text($fileName)
        let $json := json:parse($file)
        for $ret in $json//vertices/value
        return <author>{($ret/name, $ret/attribute, $ret/id)}</author>


        return element{'json'}{$return}
```

BaseX fournit :

- un serveur, lancé en ligne de commande par `basexserver`
- une interface qui permet de tester les requêtes et naviguer dans les données : `basexgui`.

3.3 Serveur

utilisation de Node.js, serveur léger programmable en JavaScript. Ce serveur est très simple à exécuter : `node <script.js>`. Il réagit aux connexions à la machine sur un port particulier et implémente la notion de routes applicative. Node.js répond à la requête en envoyant par exemple un fichier HTML. Ensuite, il gère les requêtes asynchrone, ce qui permet de programmer en push (c'est le serveur qui contacte le client). Node.js traite particulièrement bien le JSON, qui est son unité de communication.

Node.js implémente pour cela le standard WebSocket, ce qui garantit son fonctionnement sur les plateformes qui respectent les standards.

L'utilitaire `npm` (Node.js packet manager) installe en local les modules nécessaire. Ici :

```
$ npm basex  express  socket.io  socket.io-client  xml2json
```

```
//-------------------------------------------------------------------------------
//---------------------------- Mini XQuery FrameWork ----------------------------
//-------------------------------------------------------------------------------
// npm install basex socket.io express socket.io-client xml2json

var fs = require('fs');
var express = require('express');
var app = express();

//-------------=-----------------------------------------------------------------
//---- comportement du serveur web : servir le fichier serverHTML ---------------
//-------------------------------------------------------------------------------
app.get('/*', function(req, res) {
```

```
    res.setHeader('Content-Type', 'text/html');
    console.log("requestied ", req.route.params[0]);
    var serverHTML = 'html/' + req.route.params[0] + '.html';

    fs.readFile(__dirname + '/' + serverHTML,
            function(err, data) {
                if (err) {
                    res.writeHead(500);
                    return res.end('Error loading ' + serverHTML);
                }
                res.end(data);
                console.log('page ' + serverHTML + ' envoyee !');
            });
});
//------------------------------------------------------------------------------
/// si on ecoute sur le port 80, il faut lancer 'sudo node app.js' (port < 100)
var port = 8080;
var io = require('socket.io').listen(app.listen(port), {log: false});
//, {log: false});

//------------------------------------------------------------------------------
// traitement des donnees
var basex = require('./node_modules/basex/index');
var fs = require('fs');
var http = require('http');
basex.debug_mode = false;

//var d3 = require("d3");

//------------------------------------------------------------------------------
function loadfile(src) {
    return fs.readFileSync(__dirname + "/" + src, 'utf-8');
}

//------------------------------------------------------------------------------
function print(err, reply) {
    if (err) {
        console.log("Error: " + err);
    } else {
        //console.log(reply);
        var json = JSON.parse(reply.result);
        console.log(json);
    }
}

//------------------------------------------------------------
```

```
//------------ traitement des connexions ----------------
//-------------------------------------------------------
var parser = require('xml2json');
io.sockets.on('connection', function(socket) {
    //console.log(socket);
    // message
    socket.on('require', function(data) {
        console.log('require', data);

        var session = new basex.Session("localhost", 1984, "admin", "admin");
        var fileName = 'xq/' + data.file + '.xq';
        console.log('fileName', fileName);
        var cmd = loadfile(fileName);
        var query = session.query(cmd);
        query.execute(function(err, reply) {
            var json = parser.toJson(reply.result);
            console.log("sending message " + data.message, json);
            socket.emit(data.message, json);
        });
        query.close();
        session.close();
    });
});

console.log('info', 'Server is running on port ' + port);
```

3.4 Client

utilisation de D3.js. Cette librairie génère dynamiquement des représentations riches, animées et interactive pour les données.

```
<html>
    <head>
        <title>Blank Page for d3js Experimentation</title>
        <script src="http://127.0.0.1/rioultf/svn/m2dnr2i-systeme/code/d3/cours/
        <script src="socket.io/socket.io.js"></script>
        <link type="text/css" href="http://127.0.0.1/rioultf/svn/m2dnr2i-systeme
    </head>
    <body>
        <header>
            <menu>
                <div id="menu"></div>
            </menu>
        </header>
        <section id="content">
```

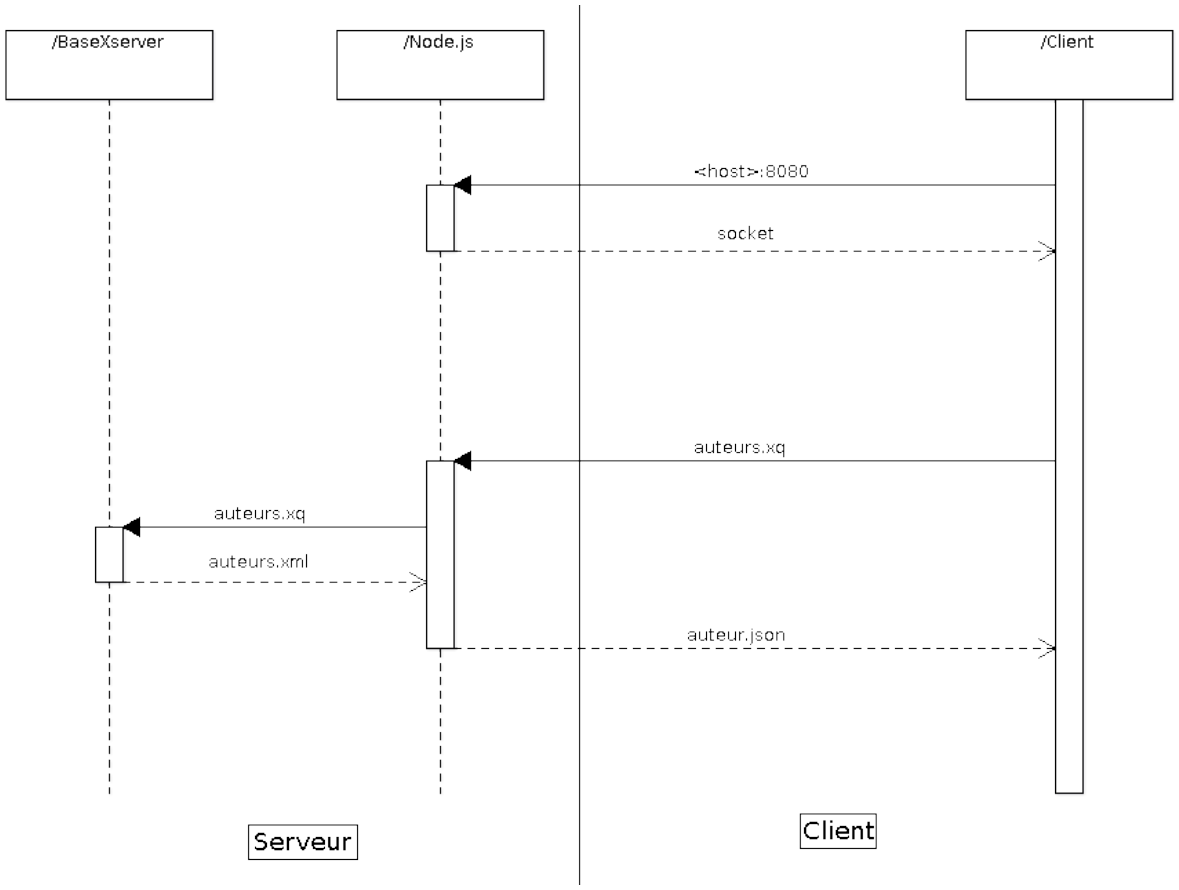
```
    <div id="d3-content"></div>
    <div id="nconf"></div>
</section>
<footer>
    cf. <a href="http://www.ultrasaurus.com/sarahblog/2012/03/d3-js-expe
        d3.js experiments in the console</a>
</footer>

<script>
    //-------------------------------------------
    function insertAuthorNames(root) {
        var json = JSON.parse(root);
        //console.log("json", json.json);
        var insert = d3.selectAll("#d3-content").selectAll("div");
        var authors = insert.data(json.json.author).enter();
        authors.append("div")
                .attr("class", "box")
                .append("div")
                .attr("class", "author-name")
                .attr("id", function(d) {
            return "authorId-" + d.id;
        })
                .on('mouseover.tooltip', function(d) {
            d3.select("#authorId-" + d.id + " text").remove();
            d3.select('#authorId-' + d.id)
                    .append('text')
                    .attr('class', 'name-rollover')
                    .text('  --> ' + d.attribute + " conf.")
                    .transition()
                    .style('opacity', 0)
                    .duration(500)
                    .style('opacity', '100%')
                    ;
        })
                .on('mouseout.tooltip', function(d) {
            d3.select("#authorId-" + d.id + " text")
                    .transition()
                    .duration(500)
                    .style('opacity', 0)
                    .remove();
        })
                .text(function(d) {
            return d.name;
        });

        // attribut des auteurs
```

```
    d3.selectAll('.box')
            .append('div')
            .style("width", function(d) {
        return (d.attribute / 2) + "em";
    })
            .attr("class", "author-attribute")
            .on('mouseover.tooltip', function(d) {
        d3.select("text.name-rollover").remove();
        d3.selectAll('footer')
                .append('text')
                .attr('class', 'name-rollover')
                .text(d.attribute + " conf.")
                ;
    })
            .on('mouseout.tooltip', function(d) {
        d3.select("text.name-rollover")
                .transition()
                .duration(500)
                .style('opacity', 0)
                .remove();
    })
            .text(function(d) {
        return d.attribute;
    });



}
//-------------------------------------------
function insertMenu(root) {
    var json = JSON.parse(root);
    console.log("json", json.json.value);
    var insert = d3.selectAll("#menu").selectAll("div");
    var menu = insert.data(json.json.menu).enter();
    menu.append("div")
            .attr("class", "menu")
            .append("a")
            .attr("href", function(d) {
        return d.name;
    })
            .text(function(d) {
        return d.label
    });
}
//-------------------------------------------
// adresse regulierement ecoutee
```

```
            var socket = io.connect('http://127.0.0.1:8080');

            socket.on('authors', function(data) {
                insertAuthorNames(data);
                socket.emit('ack');
            });

            socket.on('menu', function(data) {
                insertMenu(data);
                socket.emit('ack');
            });

            socket.emit('require', {file: 'menu', message: 'menu'});
            socket.emit('require', {file: 'auteurs', message:'authors'});
        </script>
    </body>
</html>
```

3.5 Avantages

L'intérêt d'utiliser un gestionnaire XML/Xquery :
- on ne transmet que les données utiles à l'application
- on décharge un peu le client qui doit faire moins de calcul sur les données
- on utilise Xquery, d'une expressivité remarquable pour exprimer les requêtes



3.6 Inconvénients

À la différence du développement traditionnel Apache/PHP/Sql, un serveur programmé par Node.js peut facilement planter, ce qui rend la conception particulièrement exigeante, en particulier en matière de test.

De plus, on ne gère pas que les routes, comme on le ferait avec PHP, mais aussi l'interactivité en webSocket. La conception est donc au moins deux fois plus complexe.

3.7 Difficultés

- Node.js sert un fichier HTML et pour le navigateur, l'environnement d'accès aux fichiers n'existe pas. Les inclusions de code (entre autres D3) doivent se faire en mode URI :
  `<script src="http://127.0.0.1/rioultf/.../node_modules/d3/d3.min.js"></scrip`
- en revanche, pour socket.io,
  `<script src="socket.io/socket.io.js"></script>`

– Node.js exécute des requêtes Xquery et récupère du XML qu'il faut transformer en JSON et transmettre au client :

```
var session = new basex.Session("localhost", 1984, "admin", "admin");
var xq = "auteurs.xq";
var cmd = loadfile(xq);
var query = session.query(cmd);
query.execute(function(err, reply) {
    var json = parser.toJson(reply.result);
    console.log(json);
    socket.emit('message', json);
});
query.close();
session.close();
```

– le client récupère une chaîne de caractères représentant du JSON, qu'il faut transformer en objet JSON pour l'intégrer dans D3 :

Microframework : express (cf. http ://www.stanford.edu/class/cs98si/slides/nodejs.html)

3.8 Intégration de XQuery avec BaseX

3.9 Intégration de D3

Chapitre 4

Crawling avec Scrapy

Scrapy est un framework applicatif pour aspirer des sites web et extraire des données structurées.

4.1 Vocabulaire

item conteneurs remplis par les données aspirées

spiders un nom, des URL de départ, une méthode d'analyse (`parse`) du résultat

request demande de parcours d'une URL

response réponse fournie par une requête

referrer acteur initiant une requête, `none` pour une URL de départ

4.2 Scrapy shell

Pour tester ses programmes, on peut utiliser Scrapy en mode shell interactif (l'installation de *ipython* est fortement recommandée) :

```
$ scrapy shell <url>
```

On obtient alors une console Python, qui permet :

- d'ouvrir le résultat dans un navigateur :
  `view(response)`
- d'effectuer des requêtes XPath sur le contenu :
  `hxs.select('//div[@class="content"]').extract()`
  `...`

4.3 Fonctionnement

On crée un nouveau projet scrapy comme suit :

```
$ scrapy startproject <projet>
```

On obtient les fichiers suivants :

```
projet/
|-- projet                      # module python
|   |-- __init__.py
|   |-- items.py                # fichier d'items
|   |-- pipelines.py            # fichier de pipelines
|   |-- settings.py             # fichier de réglages
|   `-- spiders                 # dossier spiders
|       `-- __init__.py
`-- scrapy.cfg                  # fichier de configuration
```

4.3.1 Items

Les items sont des conteneurs remplis par les données aspirées. Ce sont des dictionnaires python.

Dans un premier temps, on se contentera de définir les items à récupérer dans le fichier `items.py` :

```
class ProjetItem(Item):
    # define the fields for your item here like:
    # name = Field()
    title = Field()
    link = Field()
    desc = Field()
```

Les items sont extraits à l'aide de requêtes `XPath` [1], par exemple :

```
/html/head/title
/html/head/title/text():
//td
//div[@class="mine"]
```

Scrapy fournit donc des sélecteurs `XPath`, le plus général étant `hxs` (hyperText XPath Selector), auxquels on peut appliquer les méthodes suivantes :

select(<XPath>) retourne un nouveau selecteur en appliquant la requête XPath

extract() retourne une chaîne unicode

re(<regexp>) retourne une liste de chaîne unicode par application de l'expression régulière `regexp` [2].

1. `http://www.mulberrytech.com/quickref/xpath2.pdf`
2. `http://www.night-ray.com/regex.pdf`

4.3.2 Spiders

Les *spiders* sont des classes définies par l'utilisateur pour récupérer des informations sur un domaine. Ils définissent une liste initiale d'URL à explorer, comment suivre les liens et comment extraire les items du contenu récupéré.

Dans une classe `BaseSpider` d'un fichier `myspider.py` (par exemple) du dossier `spider`, il y a trois attributs principaux à définir :

name identifie le spider

start_urls une liste d'URL de départ

parse() une méthode appelée sur l'objet `Response` de chaque URL de départ. Cette méthode extrait les items et suit les URL (`Request`).

Par exemple, le spider suivant décode les descriptions d'une liste :

```
    def parse(self, response):
        hxs = HtmlXPathSelector(response)
        sites = hxs.select('//ul/li')
        for site in sites:
            desc = site.select('text()').extract()
            for d in desc:
                if len(d.strip()):
                    print d.strip()
```

Pour lancer le crawler, on exécute la commande `scrapy crawl <spider>`.
La méthode `parse()` est normalement utilisée pour remplir les items :

```
from scrapy.spider import BaseSpider
from scrapy.selector import HtmlXPathSelector
from projet.items import ProjetItem

class MySpider(BaseSpider):
    name = "myspider"
    allowed_domains = ["dmoz.org"]
    start_urls = [
        "http://www.dmoz.org/Computers/Programming/Languages/Python/Books/",
        "http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/"
    ]

    def parse(self, response):
        hxs = HtmlXPathSelector(response)
        sites = hxs.select('//ul/li')
        items = []
        for site in sites:
            item = ProjetItem()
            item['title'] = site.select('a/text()').extract()
            item['link'] = site.select('a/@href').extract()
            item['desc'] = '';
```

```
            for d in site.select('text()').extract():
                if len(d.strip()):
                    item['desc'] += d.strip()
            items.append(item)
        return items
```

Enfin, on peut exporter le résultat au format `JSON` :

```
$ scrapy crawl myspider -o items.json -t json
```

Chapitre 5

Présentation des données DBLP

DBLP est une base de données en ligne qui recense les publications des chercheurs en informatique (cf. `http://dblp.uni-trier.de/`). La base complète est disponible à `http://dblp.uni-trier.de/xml`).

On travaille sur un extrait de ces données :

attributes_maping.txt contient la liste des acronymes des conférences et une note d'importance de ces conférences, attribuées par l'expert.

```
2 DMKD
1 SAC
3 ICDM
1 CommunACM
...
```

vertices_mapping.txt liste les chercheurs

```
------Authors-----

vertex_id,DBLP_id,name

0,6,José A. Blakeley
1,7,Yuri Breitbart
2,8,Hector Garcia-Molina
3,9,Abraham Silberschatz
...
```

DBLPattr2723.txt pour différents instants temporels (T0 à T8), on liste pour chaque chercheur son nombre de publication dans chaque conférence :

```
T0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 2 2 0 3 0 0 0 0 0 1 0 0
2 0 0 0 0 3 0 0 0 0 0 2 0 0 0 0 0 0 3 2 0 0 0 0 0 0 1 3 0 5 0 6 0 0 0 0 0 0
...
```

DBLPgraph2723.txt pour chaque chercheur aux différents instants temporels, on recense ses co-auteurs :

```
T0
0 425 67 35 308
1 240 2 3 100 261 359 425 74 364 2028 174 175 16 17 115 366 76 1497 2524 285
2 1 3 5 6 8 267 659 20 366 601 25 2341 39 425 1965 2102 1339 2109 74 76 468
...
```

Chapitre 6

TP

6.1 Transformation des données DBLP au format JSON

6.1.1 Préliminaires

Réalisez quelques commandes awk/shell pour vérifier que les données sont correctes.

6.1.2 Script Python

Réaliser un script python pour transformer les données en JSON. Ce script sera lancé par la commande :

```
./preprocess.py DBLPgraph2723.txt vertices.txt conferences.txt attributes.txt >
```

Le cannevas est le suivant :

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import json
#-----------------------------------------------------------------
class MyEncoder(json.JSONEncoder):
    def default(self, o):
        return o.__dict__

#-----------------------------------------------------------------
if len(sys.argv) != 5:
    sys.stderr.write('error usage : ' + sys.argv[0] + ' <edges> <vertices> <conf
    sys.stderr.write('transforms a text graph into JSON\n')
    sys.exit(1)

edges = Edges(sys.argv[1])
```

```
attributes = Attributes(sys.argv[4])
vertices = Vertices(sys.argv[2], attributes)
conferences = Conferences(sys.argv[3])
graph = ...

print MyEncoder().encode(graph)
```

Le résultat à obtenir est le suivant :

```
{
    "edges": [
        {
            "source": 275,
            "target": 776
        },
        {
            "source": 141,
            "target": 170
        },
        {
            "source": 7,
            "target": 24
        },
...
    "vertices": [
        {
            "attribute": 4,
            "id": 0,
            "name": "José A. Blakeley"
        },
        {
            "attribute": 11,
            "id": 1,
            "name": "Yuri Breitbart"
        },
...
    "conferences": {
        "conferences": [
            {
                "index": 0,
                "name": "DMKD",
                "value": "2"
            },
            {
                "index": 1,
                "name": "SAC",
                "value": "1"
```

```
            },
...
```

6.1.3 Indications

- on peut utiliser node.js pour effectuer un pretty-print du JSON :
  ```
  node -e "console.log(JSON.stringify(JSON.parse(require('fs')\
   .readFileSync(process.argv[1])), null, 4));" data.json
  ```
- cannevas d'une classe pour construire un objet à partir d'un fichier :
  ```
  import sys

  class Attributes:
      def __init__(self, fileName):
          try:
              file = open(fileName, 'r')
              for line in file:
                  ...
          except IOError, e:
              sys.stderr.write("I/O error openning attributes file\n" + fileNa
              sys.exit(2)
  ```
- python dispose des méthodes `strip()` pour retirer les espaces en début et fin de ligne et `split(separateur)` pour créer un tableau à partir d'une chaine, découpée selon un séparateur.
- la fonction `set(<tableau>)` retourne un ensemble construit à partir d'un tableau, i.e. une structure exempte de doublons.