

COURS DE WEBMINING

P. Fernandez-Blanco

IUT d'Aix-en-Provence

Plan du cours

- **Les limitations des systèmes de bases de données \Rightarrow Motivation pour les systèmes avec de la recherche d'information (en anglais, "Information Retrieval")**
- **Recherche d'information :**
 - Indexation
 - Mesure de similarité
- **Recherche sur le web :**
 - Algorithme Page Rank
 - Algorithme HITS

Contexte

- **Webmining** : processus d'extraction d'informations contenues dans les documents stockés sur internet.
- **Techniques associées** :
 - Traitement automatique du langage naturel (Natural Language Processing)
 - > ingénierie linguistique
 - Recherche d'information (Information Retrieval)
 - > retrouver des informations dans un corpus
 - > recherche d'informations sur le web à l'aide d'un moteur de recherche
- **Quelques applications** :
 - Analyse de la consommation
 - Techniques de marketing
 - Amélioration de service

Les bases de données comparées à la recherche d'information

- Format des données :
 - BD : données structurées. Sémantique claire, basée sur un modèle formel
 - RI : non-structurées - texte libre
- Requêtes :
 - BD : formelles (par exemple, SQL)
 - IR : souvent - langues naturelles (recherche de mots-clés)
- Résultats :
 - BD : résultat exact
 - IR : résultats pertinents, parfois sans rapport

La recherche d'information

- But : trouver les documents les plus pertinents pour une requête donnée
- Notions :
 - collections de documents
 - requête (ce dont l'utilisateur a besoin comme information)
 - notion de pertinence

Problème de la RI

Recherche d'information

Auteur



concepts

concepts

Termes de la recherche

⇔
sont-ils pareils ?

Termes du document

Quel type d'information

- texte(documents)
- XML et documents structurés
- images
- audio (effets, chansons)
- vidéo
- code source
- applications/services web

Types d'information souhaitée

- rétrospective
 - « rechercher le passé »
 - requêtes différentes dans une collection statique de documents
- prospectives (filtrage)
 - « rechercher les tendances »
 - requête statique dans une collection dynamique de documents
 - dépendance temporelle

Modèles de RI

- L'approche ensembliste classique BD :

L'ensemble des documents s'obtient par une série d'opérations (union, intersection, passage au complémentaire). Correspond au langage de requêtes SQL.

- L'approche algébrique (ou vectorielle) :

Les contenus des documents sont représentés mathématiquement par des vecteurs -> notion d'espace vectoriel et de proximité entre documents

- L'approche probabiliste :

C'est l'estimation de la probabilité de pertinence d'un document par rapport à une requête.

Recherche basée sur le contenu, modèle de l'espace vectoriel

- Les requêtes et les documents sont représentés en tant que vecteurs de termes de l'index
- Indexer les documents dans la collection (offline)
- Traiter chaque requête
- La similarité est calculée entre deux vecteurs
 - les documents (réponses) : ordonnés selon la similarité (avec la requête)

Le fonctionnement d'un moteur de recherche.

Etapes :

- **L'exploration ou crawl** : faite par un robot d'indexation qui parcourt récursivement tous les hyperliens qu'il trouve en récupérant les ressources intéressantes. L'exploration est lancée depuis une ressource pivot, comme une page d'annuaire web.
- **L'indexation des ressources récupérées** : extraire les mots considérés comme significatifs du corpus de documents à explorer. Les mots extraits sont enregistrés dans une BD. Un dictionnaire inverse ou l'index permet de retrouver rapidement dans quel document se situe un terme significatif donné.
- **La recherche** : un algorithme identifie dans le corpus documentaire (en utilisant l'index), les documents qui correspondent le mieux aux mots contenus dans la requête. Les résultats des recherches sont donnés par ordre de pertinence supposée.

Robot d'indexation

Robot d'indexation : crawler, automatic indexer, bot, Web spider, Web robots

- Comment obtenir tout ce qu'il y a à indexer ?
 - Partir de sites connus ou d'un annuaire
 - Suivre les liens depuis chaque site
 - Répéter le processus

Le robot Google : Google Bot

On détaille ici le fonctionnement du robot d'indexation de Google. On distingue plusieurs fonctions.

La première concerne la découverte de nouvelles URLs :

- avec des listings RSS (Really Simple Syndication) : un fichier texte au format XML utilisé pour extraire d'un site web du contenu régulièrement mis à jour.
- directe dans un formulaire de plan de site fichiers sitemaps
- Crawl et analyse des liens sur les pages

Google Bot – Niveaux de crawl

Pour récupérer le contenu de ces nouvelles pages, il y a 3 couches de pages :

- **couche de base** : la plupart des pages du web. Elles sont crawlées avec une fréquence liée à la fréquence de mise à jour du contenu ainsi qu'à leur PageRank.
- **couche quotidienne** : un petit nombre de pages crawlées de façon quotidienne.
- **couche temps réel** : un nombre très petit de pages crawlées avec une fréquence de l'ordre de la minute, l'heure,... (ex. l'actualité)

Le robot sait gérer ces 3 couches de pages à crawler.

Pour chaque page, un score de crawl est calculé, et ce score détermine à quelle couche la page sera associée.

GoogleBot – Analyse des liens

Le robot récupère une page, son contenu et les liens => analysés

- l'analyse du contenu d'une page :
 - calcul des signatures
 - une à partir de l'URL de la page
 - une à partir du contenu de la page
 - détection de contenu dupliqué au niveau d'une page et d'un site
- l'analyse du texte des liens et du texte autour des liens

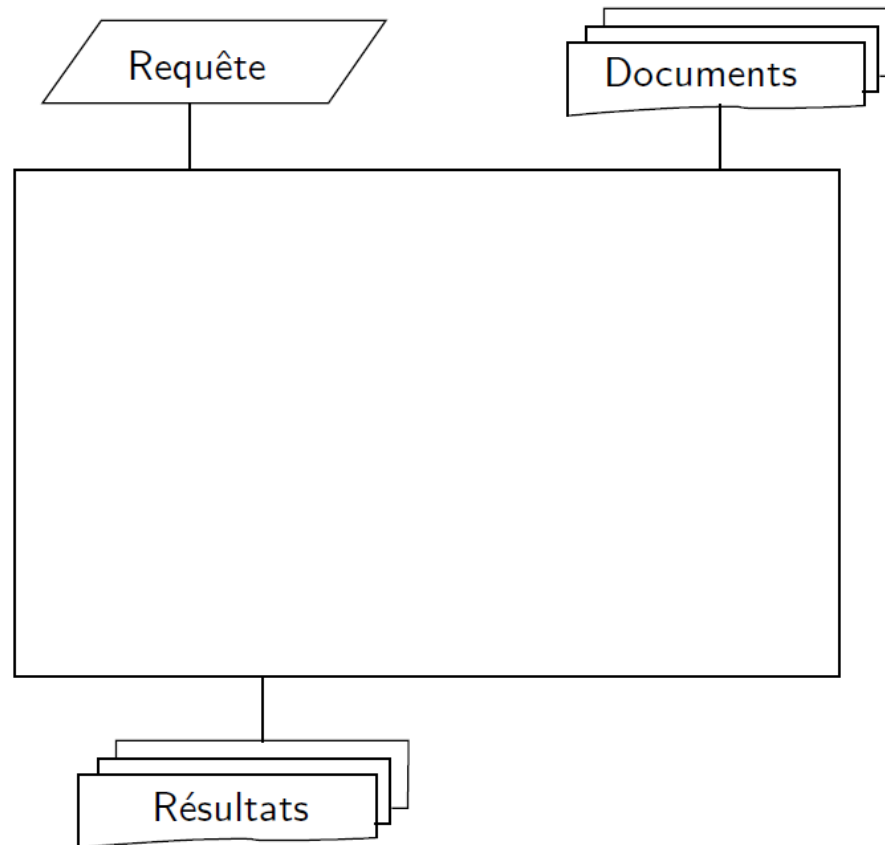
Robot d'indexation - fonctionnement

- initialisation de la queue des URLs avec les URLs de départ
- tant qu'on peut encore visiter
 - prendre l'URL disponible suivante depuis la queue
 - télécharger le contenu et marquer l'URL comme visitée
 - extraire les hyperliens depuis le document nouvellement téléchargé et les
 - rajouter dans la queue, s'ils satisfont les critères nécessaires
 - réévaluer les conditions pour continuer à visiter des sites (profondeur maximale, nombre maximal de documents récupérés, temps maximal d'exécution, queue d'URLs vide, etc.
 - attendre « un peu » avant de continuer (pour ne pas « assommer » le serveur)

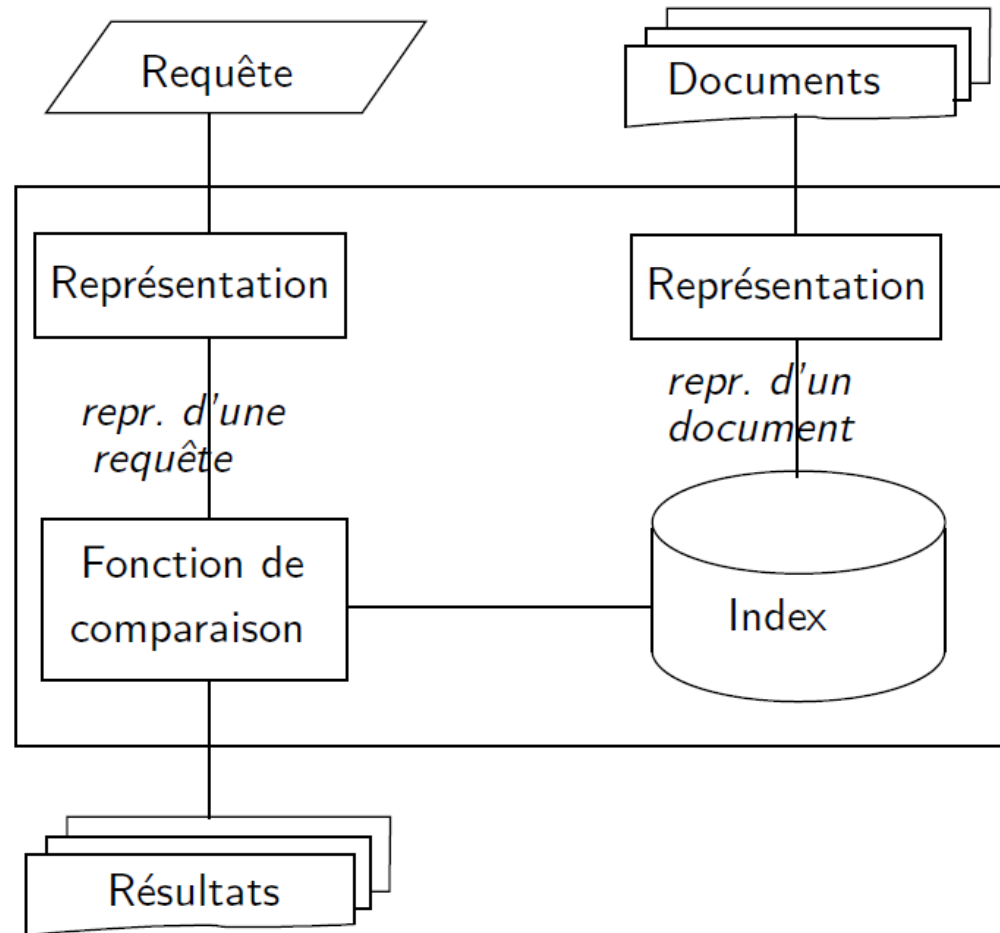
Recherche basée sur le contenu

- contenu pris en compte
 - meta-données balises de description des pages qui renseignent le robot => ne sont plus trop utilisées
 - liens
 - contenu txt donné par un analyseur de pdf, html, .doc,...
- contenu ignoré
 - robots.txt : un fichier demandant aux robots de ne pas indexer les pages mentionnées.
 - directive NOFOLLOW utilisée dans les blogs pour les commentaires-spam `anchor text`
 - pages situées en profondeur de plus de trois niveaux
 - pages dynamiques souvent pas plus d'un niveau. Elles sont indexées plus lentement que les pages statiques.
- "bot spoofer"
 - Pour aider les développeurs web : un logiciel qui imite un robot d'indexation et voit les pages d'un site comme le robot.

Indexation et Recherche : La boîte noire "RI"



Indexation et Recherche : La boîte noire "RI"

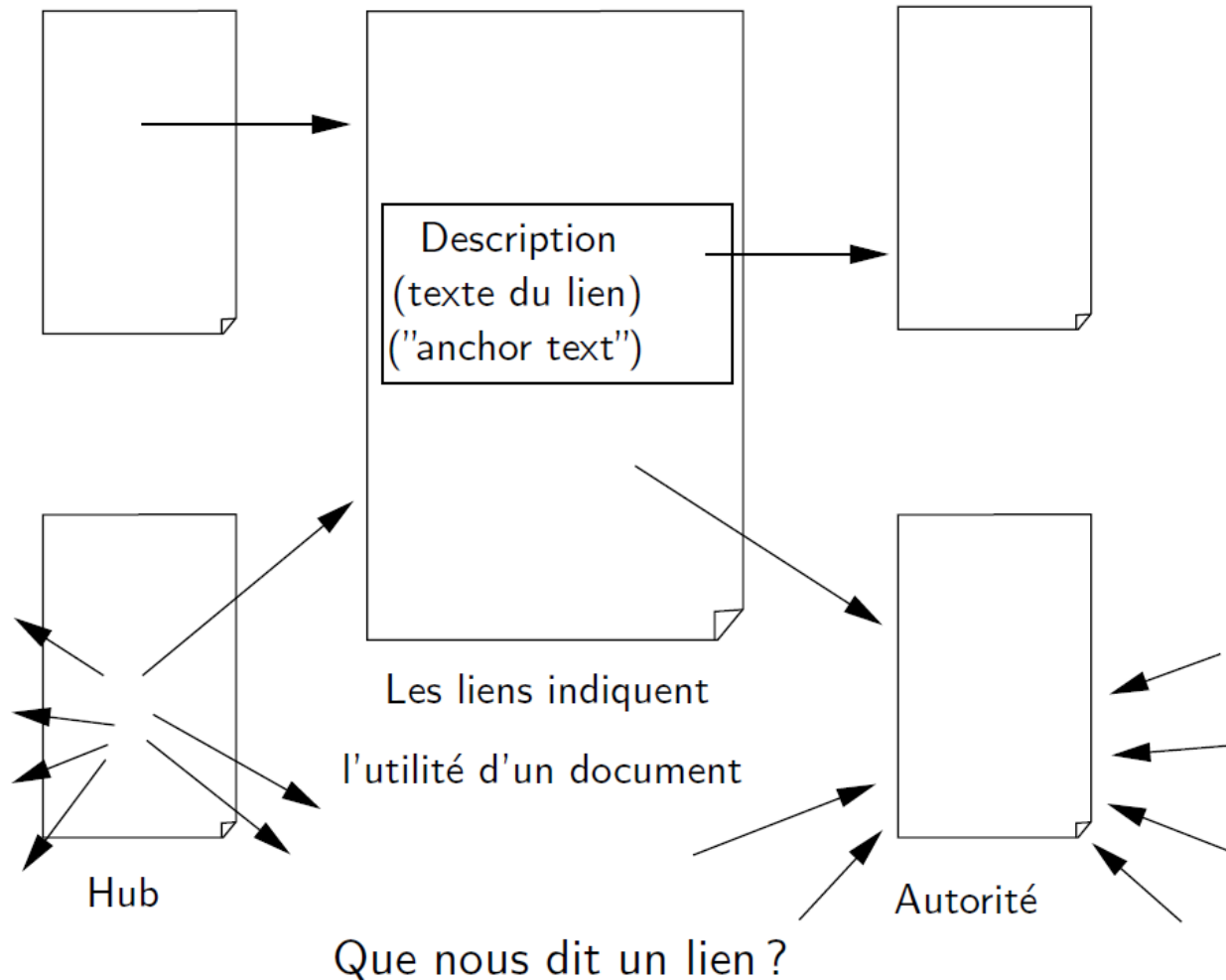


Amélioration de la recherche basée sur le contenu (Google)

Les caractéristiques clés de Google

- Utilisation des indexes basés sur le contenu
 - Indexes peuvent être partitionnés sur plusieurs machines ou bien
 - les données sont dupliquées sur les machines et les requêtes sont distribuées parmi celles-ci
- PageRank
 - ordonner les documents trouvés, en utilisant les liens entre pages comme indice de pertinence

Liens entre les documents



La recherche basée sur les liens

- **Liens internes** : entre deux pages différentes d'un même site
- **Liens externes** : un lien pointant depuis un site vers des pages quelconques d'un autre site.
 - ⊙ **Sortants** : Ils aident les moteurs de recherche à mieux comprendre le contenu du site vers le quel votre lien pointe.
 - ⊙ **Entrants** : les obtenir d'un annuaire, par négociation

Les liens sortant d'un site - règles :

- peu de liens externes sortants (environ 5), sinon le site est considéré comme étant un annuaire.
- doivent être en rapport avec le thème du site ou de la page qui héberge le lien

La recherche basée sur les liens : PageRank

Notion d'« autorité »

- Autorité : page référencée par beaucoup d'autres pages, c'est une page dont le contenu est de qualité (sur un sujet précis).
-> Une autorité a beaucoup de liens entrants
- L'intuition : des sites de moins bonne qualité n'ont pas autant de sites qui pointent vers eux

La recherche basée sur les liens : HITS (Hypertext Induced Topic Search)

Notion de « Hub » et « autorité » :

- Autorité : page référencée par beaucoup d'autres pages.
- Hub (ou concentrateur) : Page contenant des listes de bonnes références, c'est une page qui guide les utilisateurs vers les pages d'autorité.
-> un hub a beaucoup de liens sortants
- L'intuition : des sites de moins bonne qualité n'ont pas autant de sites de bonne qualité qui pointent vers eux

Simple crawler

Une instance CrawlerURL - URL parcourue par le crawler, pour chaque document retrouvé elle est créée dans SimpleCrawler

- ⊙ : `::computeURL()` : Crée l'objet URL (en Java) à partir de la chaîne de caractères.

- ⊙ : `::getURL()` : Rend l'objet URL (en Java) créé par : `::computeURL()`, utilisé dans `::setRawContent()`

- ⊙ : `::getDepth()` : Rend le niveau de profondeur courant (limiter le parcours), utilisé dans `::addUrlsToUrlQueue()`

- ⊙ : `::isAllowedToVisit()` : Rend le booléen qui dit si cet URL est permise pour aller chercher le contenu du document. Appelée dans `SimpleCrawler::doWeHavePermissionToVisit()`

- ⊙ : `::isVisited()` : Rend le booléen qui dit si on est déjà passé par là. Appelé par `CrawlerURL::isUrlAlreadyVisited()`.

- ⊙ : `::getUrlString()` : Utilisé dans SimpleCrawler, pour obtenir la chaîne de caractères qui représente l'URL (pour gérer leur liste, les afficher, etc.).

- ⊙ : `::toString()` : Surcharge de la méthode Java générique.

Document - Parser JSOUP : Parse-document-from-string

public class CrawlerUrl { //données et méthodes pour le contenu téléchargé depuis l'URL }

1. <http://jsoup.org/cookbook/input/parse-document-from-string>
2. private Document htmlJsoupDoc ;

Document : Elements + TextNodes + nodes

1. Jsoup.parse(String html, String baseUri)
2. public static Document parse(String html,String baseUri)
3. html - HTML to parse ; baseUri - base URI du document pour résoudre les liens relatifs.
4. Throws : MalformedURLException -l'URL demandé n'est pas HTTP ou HTTPS URL
 - htmlJsoupDoc = Jsoup . parse(htmlText,baseURL) ;

Document - Parser JSOUP : Parse-document-from-string

```
public CrawlerUrl(String urlString, int depth)  
public void setRawContent(String htmlText)
```

- Ce que fait le parser jsoup :
 - En entrée : string JAVA contenant un HTML
 - But : analyser le HTML pour retrouver son contenu, le vérifier/modifier si besoin
 - L'analyseur manipule <http://jsoup.org> :
 - unclosed tags (e.g. `<p>Blah -> <p>Blah</p>`)
 - implicit tags (e.g. `<td>Table data</td> -> <table><tr><td>`)
 - créant la structure de document HTML : `head +title+ body.(text())`
- L'objet « document »
 - En sortie du parser : document
 - Document = Elements + TextNodes
 - Chaîne d'héritage :
 - Document extends Element extends Node.
 - TextNode extends Node

Document - Parser JSOUP : Parse-document-from-string

Extraire des data avec jsoup -> utiliser un sélecteur de syntaxe pour extraire des Elements

- Méthodes avec des sélecteurs CSS ou similaires à jquery :

- `Element.select(String selector)`
- `Elements.select(String selector)`

- Un « select » renvoie une liste d'Elements

- Quelques sélecteurs :

- `tagname : ". title()"`

`private String title ; title = htmlJsoupDoc . title() ;`

- `[attribute] : "[href]"`

`Elements hrefJsoupLinks = htmlJsoupDoc . select("a[href]") ;`

- attributs qui commencent par la valeur: `[attr^=value]`
- attributs qui finissent par la valeur: `[attr$=value]`
- attributs qui contiennent la valeur: `[attr*=value]` par exemple `[href*= /path/]`

`Elements ImagesPNGs = htmlJsoupDoc .select("img[src$=.png]")`

Document - Parser JSOUP : Les données des Elements

Retrouver les données (les attributs, le texte, et le HTML) stockées dans Elements après l'analyse d'un document Document `htmlJsoupDoc`

- pour obtenir la valeur d'un attribut : méthode `Node.attr(String key)`
- pour le texte sur un élément : `Element.text ()`
`private String niceText ; niceText = htmlJsoupDoc . body() . text() ;`
- pour le HTML : `Element.html ()`, ou `Node.outerHtml ()`

```
Elements hrefJsoupLinks = htmlJsoupDoc . select("a[href]");
for (Element link : hrefJsoupLinks) {
    String thisLink = link.attr("abs:href");
    if(thisLink . startsWith("http://")) {
        System.out.println("JSOUP Found: " + thisLink);
        linkList . add(thisLink);
    }
}
```

SimpleCrawler – parcourir le web

Problème :

- aller sur le web retrouver un document HTTP (comme votre browser préféré)
- récupérer depuis ce document les liens pointant vers d'autres documents
- aller retrouver ces documents également
- continuer tout ceci tout en
 - gardant trace du "trajet" parcouru, pour ne pas boucler (par exemple si D1->D2->D3->D1)
 - respectant les éventuelles interdictions posées par les administrateurs des sites distants (fichier robots.txt)
- `httpClient` - retrouve des documents depuis des serveurs HTTP –on s'en crée un objet et exécute() des requêtes (classe `HttpGet`)
- `Queue<CrawlerUrl> urlQueue` - une liste dans laquelle on rajoute au fur et à mesure les liens des doc. téléchargés
- `visitedUrls` - une table de hachage pour noter le parcours au fur et à mesure : on y rajoute les URLs visités, un par un, et on la consulte pour savoir si on est déjà passé par là
- `sitePermissions` - pour chaque site distant, la liste des pages à ne pas explorer, donnée dans le fichier distant robots.txt de ce site-la

SimpleCrawler – parcourir le web

Methode pour le parcours :

- dans `SimpleCrawler::crawl()` on fait ceci :
 - on initialise la liste `urlQueue` avec l'url initiale
 - tant que la liste n'est pas vide et qu'on n'est pas au max des docs vus
 1. on extrait l'URL-element `U` qui est en tête d'`urlQueue`,
 2. on le valide (`U` ne doit pas déjà avoir été visité, etc.)
 3. on récupère depuis le web le doc `D` pointe par `U`, utilisant `httpClient.execute()`
 4. on marque `U` comme ayant été visité
 5. on traite `D` (analyse avec `Jsoup::parse()`, obtention liens)
 6. on sauvegarde localement le texte de `D`
 7. on insère les liens obtenus depuis `D` dans `urlQueue`
- Si au pas 2 `U` est invalidée, on itère la boucle
- Les pas 3,4,5 sont dans `SimpleCrawler::getContent(CrawlerUrl url)`
- Le pas 4 : appelle la méthode de `SimpleCrawler` qui met `U` dans `SimpleCrawler::visitedUrls` et qui appelle ensuite pour `U` la méthode `CrawlerUrl::setIsVisited()`
- Le pas 5 : appelle (avec le texte `D` en paramètre) la méthode de `CrawlerUrl` qui appelle `Jsoup::parse()` dessus et récupère ensuite le titre, le corps, les liens, etc.

SimpleCrawler – parcourir le web

Exemple de parcours :

- Supposons que notre urlQueue contient ceci : $U_1, U_2, \dots U_n$
- Faisons tourner la boucle de `SimpleCrawler::crawl()`
 - on extrait U_1 - donc urlQueue contient maintenant $U_2, \dots U_n$
 - on valide U_1 - supposons qu'elle est bien valide
 - on récupère depuis le web le doc D_1 pointé par U_1
 - U_1 est mise dans la table de hachage `SimpleCrawler::visitedUrls` et marquée comme visitée avec `CrawlerUrl::setIsVisited()`
 - on traite D_1 obtenant ses liens sortants $U_1^D, U_2^D, \dots U_k^D$
 - on sauvegarde D_1 localement
 - on insère ces liens dans la liste, donc urlQueue contient maintenant $U_2, U_3, \dots U_n, U_1^D, U_2^D, \dots U_k^D$
 - on itère
 - on extrait ainsi U_2 - donc urlQueue contient maintenant $U_3, \dots U_n, U_1^D, U_2^D, \dots U_k^D$
 - on valide U_2 ,
 - si elle est valide, on récupère D_2
 - etc.

SimpleCrawler – parcourir le web - CrawlerUrl

- donnée-membre URL url, utilisant la classe Java URL.
 - entité de base pour gérer les noms d'hôte, le port, l'adresse elle-même, les tests pour savoir si on a le droit d'y aller, etc.
- donnée-membre booléenne isVisited
 - Mise à vrai par `CrawlerUrl::setIsVisited()`
 - sa valeur est rendue par `CrawlerUrl::isVisited()`
- donnée-membre `List<String> linkList`
 - contient les liens sortants du document pointé par cette URL et téléchargé (dans SimpleCrawler)
 - rendue par `CrawlerUrl::getLinks()` (par exemple appelée dans `SimpleCrawler::addUrlsToUrlQueue()`)
 - peuplée suite à l'analyse du texte d'un document avec `CrawlerUrl::setRawContent(String htmlText)`
 - reçoit le texte HTML tel qu'il vient d'être téléchargé
 - l'analyse avec `Jsoup::parse()`
 - en extrait le titre avec `Jsoup::title()` et le corps avec `Jsoup::body()`
 - en extrait les liens dans une boucle parcourant les éléments HTML href
 - met ces liens dans la donnée-membre linkList etc.

JavaDoc - mot clé this

```
public class Point {  
    public int x = 0;  
    public int y = 0;  
    //constructor  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

JavaDoc - Structures de données Crawler - Listes

la liste des URLs à parcourir :

```
Queue<CrawlerUrl>= new LinkedList<CrawlerUrl>
```

```
Interface Collection<E> // l'interface "racine" -----//
    boolean isEmpty() // renvoie VRAI si la collection ne contient aucun element
    int size() // renvoie le nombre d'elements de la collection

Interface Queue<E> // herite de Collection<E> -----//
    boolean add(E e) // rajoute l'element e a la queue, renvoyant VRAI si succes
    E remove() // retrouve l'element qui est en tete de la queue, le rend
                // en retour, et l'enleve egalement depuis la queue

class LinkedList // implementation de Interface List -----//
                // qui a son tour implemente Collection
```

Documentation Java - Structures de données - Table de hachage

- `private Map<String, CrawlerUrl> visitedUrls`
- `private Map<String, Collection<String> > sitePermissions`

```
Interface Map<K,V> // associe valeurs V a des cles K (uniques) -----//
                    // mieux eviter de modifier les cles une fois dedans
                    // permet d'enumerer les cles, les valeurs ou encore
                    // les paires (cle, valeur)
int size()         // renvoie le nombre total de paires (cle, valeur)
boolean containsKey(Object cle) // VRAI si cette cle est dans la 'map'
                                // i.e. la map a une paire (cle,valeur)
V put(K cle, V valeur) // associe valeur a cle dans la map
V get(Object cle)      // renvoie la valeur associee a cle si elle existe
                        // dans la map, et sinon elle renvoie null
```

Documentation Java - Structures de données - Table de hachage

```
public SimpleCrawler(...)
```

- this.visitedUrls=new HashMap<String,CrawlerUrl>() ;
- this.sitePermissions=new HashMap<String,Collection<String> >() ;

```
static interface Map.Entry<K,V> // classe declaree dans la classe Map
                                // pour représenter les paires
```

```
Set<Map.Entry<K,V>> entrySet() // renvoie l'ensemble des paires
```

```
class HashMap<K,V> // implementation de Interface Map -----//
                    // en tant que table de hachage
                    // on l'utilise pour instancier nos Map d'URLs
                    // visites, respectivement de permissions de site
```

Documentation Java - Structures de données - Énumération

```
public interface Set<E> // collection d'elements sans duplicata ---//  
    extends Collection<E>
```

```
Iterator<E> iterator() // renvoie un iterateur -- enumerer les elements
```

```
public interface Iterator<E> // un iterateur pour une collection -- //  
    boolean hasNext()          // VRAI s'il existe un element suivant  
    E next()                   // renvoie l'element suivant
```

```
// Donc par exemple, apres avoir rempli une (hash)map, on peut recuperer  
// ses entrees avec entrySet(), prendre un iterateur dessus et les  
// enumerer une par une pour les examiner, les traiter, etc.
```

Documentation Java – Entrées/(sorties)

```
public abstract class InputStream // la superclasse de toutes -----//
    extends Object                // les classes representant des
    implements Closeable           // flots d'entree d'octets

public class FileInputStream      // utilisee pour lire un fichier
    extends InputStream

FileInputStream(String cheminFichier) // constructeur

public final class Scanner       // analyseur lexical simple avec
    extends Object               // lequel nous lirons depuis des
    implements Iterator<String>  // fichiers (e.g. ligne par ligne)

Scanner(InputStream source)      // constructeur -- nous allons l'utiliser pour lire
    // depuis un FileInputStream

boolean hasNextLine()            // VRAI s'il y a une ligne dispo

String nextLine()                // avance le scanner avec toute une ligne et la
    //renvoie

void close()                     // ferme le scanner
```

Documentation Java – (Entrées)/sorties

```
public abstract class Writer    // classe abstraite pour ecrire --//
    extends Object              // sur des flots de caracteres
    implements Appendable, Closeable, Flushable

public void write(String str)   // ecrit la chaine de caracteres str

public Writer append(CharSequence csq) //ecrit en rajout la chaine de chrs
    // (notez que la classe String implemente l'interface CharSequence)

public class OutputStreamWriter // "pont" pour encoder les caracteres
    extends Writer              // correctement (Unicode, etc.)

public class FileWriter          // classe pour ecrire "simplement"
    extends OutputStreamWriter   // dans des fichiers texte

FileWriter(String cheminFichier) // constructeur

public class BufferedWriter      // pour ecrire du texte sur un flot
    extends Writer              // de sortie de caracteres

BufferedWriter(Writer out)      // constructeur -- on va l'utiliser avec des FileWriter
    // pour ecrire

void flush() // vide les tampons
void close() // ferme le BufferedWriter
```


Documentation Java – Chaînes de caractères

```
public final class StringBuilder // chaine de caracteres -----//
    extends Object                // modifiable
    implements Serializable, CharSequence

StringBuilder append(CharSequence s) // rajoute la chaine de caracteres

String toString() // renvoie le 'String' contenant la chaine de caracteres construite

public final class String // chaine de caracteres constantes
    extends Object
    implements Serializable, Comparable<String>, CharSequence
    int length() // renvoie la longueur de la chaine
    String[] split(String regexp) // decompose le String en lexemes selon l'expression
                                   // rationnelle regexp, par exemple :

String a = "aaa bbb ccc";
String[] t = a.split("\\s+"); // fera t[0]="aaa" t[1]="bbb",
// t[2]="ccc"
// la concatenation de String se fait avec l'operateur +:
String c = "aa" + " " + "bb";
```

Documentation Java – HTTP, URL

```
public final class URL // represente une URL -----//
    extends Object
    implements Serializable

URL(String chaineDeChrUrl) // constructeur

String getHost()           // extrait la partie hote de l'url

String getPath()           // extrait la partie chemin de l'url

int getPort()              // extrait le numero de port de l'url

int hashCode()             // cree un entier pour la table de hachage
```