Rapport - Projet de XML

Nathan COZZO - 71801617
Mehmet YALCIN - 22211019

May 22, 2023

Contents

1 Conversion en XML d'un arbre au format CSV

1.1 Le script python

La première partie de notre projet a consisté à lire les fichiers au format CSV à savoir **treeoflife_nodes.csv** représentant les nœuds d'un arbre phylogénétique et **treeoflife_links.csv** représentant les liens entre ses nœuds. Pour se faire, nous avons écrit un script python lisant ces deux fichiers et renvoyant sur la sortie standard un fichier au format XML. Il ne reste plus qu'à utiliser la redirection ">" pour envoyer le résultat dans un fichier.Bien évidement, notre script python permet également de lire d'autres fichiers ayant la même forme que les fichiers cités précédemment, en respectant l'ordre des paramètres : le fichier nœud puis le fichier lien. On obtient la commande suivante :

```
$ ./script.py *_nodes.csv *_links.csv > fichier.xml
```

Expliquons rapidement comment nous avons écrit ce script python. Afin de lire correctement les fichiers csv, nous avons utilisé la bibliothèque standard csv. Au début, nous pensions utiliser la méthode split de python afin de séparer les éléments entre virgule. Malheureusement, cette technique n'est pas suffisante car le nom des nœuds peuvent contenir une virgule. C'est le cas, par exemple, du nœud avec l'id n°60296 portant le nom suivant *"Brenierea, Bauhinia"* Enfin pour lire les lignes du fichier csv, on utilise un reader puis une boucle foreach sur ce reader.

1.2 Format choisi

Dans cette partie, nous allons parler de la forme du fichier xml que l'on obtient grâce au script python. Nous avons fais une première version avant d'arriver à la version finale.

La première version a été :

```
<tree>
        <node id=' '>
                <child idref=' '/>
                ...
                <child idref=' '/>
        </node>
        <node> ... </node>
</tree>
```

Pour réaliser ce schéma, il a fallut enregistrer les nœuds et les liens dans un dictionnaire python pour faire la correspondance entre "node" et "child". Or, cela fait un premier traitement sur les données. Le but du cours étant de manipuler et transformer des fichier XML, nous pensons qu'il ne faut pas de premier traitement dans le script mais uniquement une lecture puis une réécriture simple en XML. On obtient donc la version finale sans traitement dans le script :

```
<tree>
```

```
        <nodes >
                <node node_id=' ' .../>
                ...
                <node node_id=' ' .../>
        </nodes>
        <links>
                <link source_node_id=' ' target_node_id=' ' />
                ...
                <link/>
        </links>
</tree>
```

1.3 Notre schéma XSD

Enfin notre XSD valide la version finale, mais nous avons aussi un XSD qui valide la première car elle a été réalisée en premier.

Le schéma XSD doit vérifier :

- l'unicité de id ref sinon il y a un risque que la première transformation boucle
- qu'on a bien un élément "nodes" contenant une séquence potentiellement infinie d'element "node"
- qu'on a bien un élément "links" contenant une séquence potentiellement infinie d'élément "link" et un élément "links"
- les attributs importants de l'élément "node" : node_id et name
- les attributs importants de l'élément "link" : source_node_id et target_node_id

Pour faire la vérification, on utilise xmllint.

2 Conversion des données XML en SVG

2.1 Première transformation : la transformation initiale

On commence avec un arbre qui contient un élément "nodes" et un élément "links". L'élément "nodes" contient des éléments node avec les informations du noeud en attribut et l'élément "links" contient des éléments "link" avec les informations d'un lien en attribut. Cet arbre est la version finale expliquée précédemment. Ce que l'on obtient après la transformation est un élément tree qui contient plusieurs éléments node qui contiennent eux des éléments child. En effet, c'est notre version initiale expliquée dans la partie précédente.

2.2 Deuxième transformation : des nœuds dans des nœuds

L'objectif de cette transformation et de réécrire l'arbre de façon à avoir des nœuds dans des nœuds afin de faciliter le comptage de la profondeur de l'arbre. La racine représenterait alors la source de l'arbre, c'est un nœud qui n'est pointé par aucun autre. Alors comment faire pour trouver la source de l'arbre ? Nous avons utilisé l'expression xPath suivante :

```
tree/node except tree/key('preg',node/child/@idref)
```

Puis on a une règle sur les éléments "node" qui recopie simplement l'élément et ses attributs que l'on applique au début sur le "node" source obtenue avec l'expression xPath.

Nous avons eu cette idée de transformation car nous avons voulu essayer d'utiliser l'élément "xsl:number" avec l'attribut level="multiple" vu dans le dernier cours :[exemple d'utilisation dans le dossier brouillon : transformation-number.xsl].

2.3 Troisième transformation : calcul des positions

Il a fallu comprendre comment construire la représentation graphique avant de faire des transformations : Quelles sont les éléments à faire apparaître pour construire les fichiers svg pour le rectangle et pour le cercle.

Nous avons commencé par numéroter les feuilles en partant du n°0 qui représente la feuille la plus à gauche dans l'arbre.Nous avons appelé cet numérotation "x" car elle fais penser à l'axe des x dans la représentation rectangulaire. Pour pouvoir faire cette numérotation, nous avons utiliser "xsl:number" avec l'attribut level="any" et l'attribut count qui compte uniquement les feuilles.

Maintenant, il faut que tous les noeuds aient un attribut "x", comment faire ? L'idée est de partir des feuilles puis de remonter vers la racine. Mais comment faire ça en XSLT ? Nous avons utiliser la notion de variable. En effet, nous avons enregistré dans une variable le résultat d'un apply template sur les fils puis lu ce résultat ce qui permet de faire partir des feuilles les coords et les remontes petit à petit. On oublie pas de faire une copy du résultat de la variable sinon aucun résultat n'apparaît. Pour le calcul, on a juste a prendre le x du fils de gauche (noté xmin) et le x du fils de droite (noté xmax) puis les et diviser par 2 pour obtenir le x du noeud actuel. $x = \frac{xmin+xmax}{2}$

Ensuite, nous avons un attribut y qui permet de calculer la profondeur de l'arbre. Pour calculer la profondeur de chaque nœuds, nous avons utilisé une autre notion à savoir les paramètres dans des templates. On a le paramètre "profondeur" que l'on incrémente à chaque appel récursif en commençant par la racine qui vaut 0.

2.4 Explication des représentations

2.4.1 Représentation rectangulaire

Tous les nœuds tirent un trait vertical vers le bas. Et les nœuds qui ne sont pas des feuilles(qui n'ont pas de fils) tirent un trait horizontal du fils le plus à gauche au fils le plus à droite (correspondant aux attributs xmin et xmax dans le code)

L'espace entre les feuilles est égale à la longueur du document divisé par le nombre de feuilles. La longueur des traits verticales est égale à la largeur du document divisé par la profondeur max de l'arbre. Pour avoir la position x sur le graphique, on multiplie x par l'espace entre les feuilles. Et pour le y, on multiplie la profondeur par la longueur des traits à la verticales.

2.4.2 Représentation circulaire

On utilise les mêmes informations que celles qu'on utilise pour faire la représentation graphique circulaire, ce qui change, c'est la façon de tracer les traits.

La profondeur dans l'arbre représente le rayon dans la représentation graphique en cercle. Le nœud source est au centre. L'espace entre les feuilles est alors $\frac{2\pi}{nbfeuille}$. Le rayon de notre cercle est la taille du document divisé par 2. Donc on place le centre au coordonnée (rayon,rayon). Pour calculer l'angle, on prend le x et on multiplie par l'espace entre les feuilles. Puis on calcul le premier rayon et le suivant grâce au y. Et on trace un arc de cercle grâce a xmin, xmax, et le plus grand rayon actuel.

3 Résumé des transformations

Voici un résumé de toutes les transformations :

- **transformation_initiale.xsl** : permet de traiter le résultat du script python pour avoir un élément "tree" qui contient des éléments "node" qui contiennent des éléments "child".
- **transformation_source.xsl** : permet de traiter le résultat de la **transformation_initiale.csv** Cette transformation permet d'obtenir des éléments "node" qui contiennent des éléments "node" en identifiant le nœud source qui devient la racine du document.
- **transformation_position.xsl** : permet de traiter le résultat de la transformation **transformation_source.xsl**. Cette transformation permet de calculer les positions x, y ainsi que les xmin et xmax utiles pour faire le graphique.
- **transformation_rectangle.xsl** : permet de traiter le résultat de la transformation **transformation_position.xsl**. Cette transformation permet d'obtenir un fichier SVG pour la représentation rectangulaire.

- **transformation_cercle.xsl** : permet de traiter le résultat de la transformation **transformation_position.xsl**. Cette transformation permet d'obtenir un fichier SVG pour la représentation circulaire.

4 Conclusion

Pour conclure, nous avons réussi dans un premier temps à lire les deux fichiers au format csv afin d'obtenir un fichier xml. Puis, uniquement grâce à une suite de transformation XSLT, nous avons réussi à obtenir les deux représentations graphiques attendues avec un temps de traitement assez court.

Bonus : Le sujet nous demande de faire 2 fichiers XSLT différents pour les représentations, mais nous aurions pu aussi utiliser la notion des modes et faire des règles avec des modes différents : rectangle ou cercle. Puis on aurait généré les deux fichiers SVG avec un seul fichier XSLT.