Jour 12 – L'instruction < et la récursion

Nous avons vu le jour 2 l'instruction > qui permet d'obtenir les nodes référencés par les ways. Si le lot de données courant contient des relations, cette instruction inclut les nodes et ways référencés par les relations, ainsi que les nodes de ces ways. L'instruction > permet donc de produire un résultat complet : toutes les **références** sont « résolues ». Overpass parle de *récursion descendante*.

Ainsi la requête suivante produit le circuit de randonnée, les ways référencés par la relation et les nodes référencés par ces ways.

```
relation
  [type=route]
  [route=hiking]
  [name="Circuit de Haute-Perche"];
out;
>;
out;
```

Observez le résultat dans l'onglet Données et comparez avec les données obtenues en remplaçant les trois dernières lignes par l'instruction `out geom;` : dans un cas on récupère les ways et les nodes avec leurs tags, dans l'autre les coordonnées sont intégrées à la relation.

Il existe également l'instruction < qui a l'effet inverse : sont produits les ways et relations qui référencent un node ou un way du lot de données courant : c'est la *récursion ascendante*. On peut alors compléter le résultat avec l'instruction > pour inclure les ways et nodes référencés. Exemple avec pour élément de départ l'Avenue Arthus-Princé à Chaumes-en-Retz :

```
area["ref:INSEE"=44005];
way[name="Avenue Arthus-Princé"](area);
<; out;
>; out;
```

Le résultat est conséquent ! Pour y voir clair supprimez la dernière ligne pour ne voir que les relations référencées par la rue et complétez l'instruction out par l'option `tags` pour ne voir que les tags, dans l'onglet Données :

```
area["ref:INSEE"=44005];
way[name="Avenue Arthus-Princé"](area);
<; out tags;
```

On observe une ligne de transport, la relation associatedStreet de la rue, et 4 itinéraires de randonnée. Si on s'intéresse aux seules randonnées, ont peut appliquer un filtre sur les relations obtenues avec l'instruction <.

```
area["ref:INSEE"=44005];
way[name="Avenue Arthus-Princé"](area);
<;
rel._[route=hiking];
```

```
out geom;
```

Quelle est cette syntaxe `rel._` !?! La **variable `._`** désigne le lot de données courant, `rel._` désigne donc les relations de ce lot de données, sur lesquelles on applique le filtre de tag. On obtient le même résultat en utilisant une variable nommée :

```
area["ref:INSEE"=44005];
way[name="Avenue Arthus-Princé"](area);
< -> .refs;
rel.refs[route=hiking];
out geom;
```

On peut également utiliser une variable pour le lot de données en « entrée » de l'instruction <, sur lequel la récursion opère :

```
area["ref:INSEE"=44005];
way[name="Avenue Arthus-Princé"](area)->.rue;
.rue < -> .refs;
rel.refs[route=hiking];
out geom;
```

Bien sûr les variables peuvent aussi être utilisées avec l'instruction >. L'exemple suivant est équivalent à la première requête de ce tuto :

```
relation
  [type=route]
  [route=hiking]
  [name="Circuit de Haute-Perche"]->.rel;
.rel out;
.rel > -> .refs;
.refs out;
```

Notez l'utilisation des variables **avant** l'instruction `out` : la requête **retourne le contenu de la variable** et non le lot de données courant. Ce n'est pas forcément utile mais cela explicite bien le fonctionnement des variables dans Overpass.

Exercices

- Trouvez les rues de Chaumes-en-Retz avec un passage piéton.
- Trouvez les randonnées connectées au Circuit de Haute-Perche, c'est-à-dire ayant un node ou un way en commun.
- Idem mais sans le Circuit de Haute-Perche dans le résultat.
- Trouvez les tables de pique-nique à proximité du Circuit de Haute-Perche.
- Que signifie la syntaxe `(._;>;);` générée par Overpass Turbo pour “réparer une requête” ?

Au fait, vous pouvez désactiver les messages « Données incomplètes » dans les paramètres généraux, sous le choix du serveur Overpass…