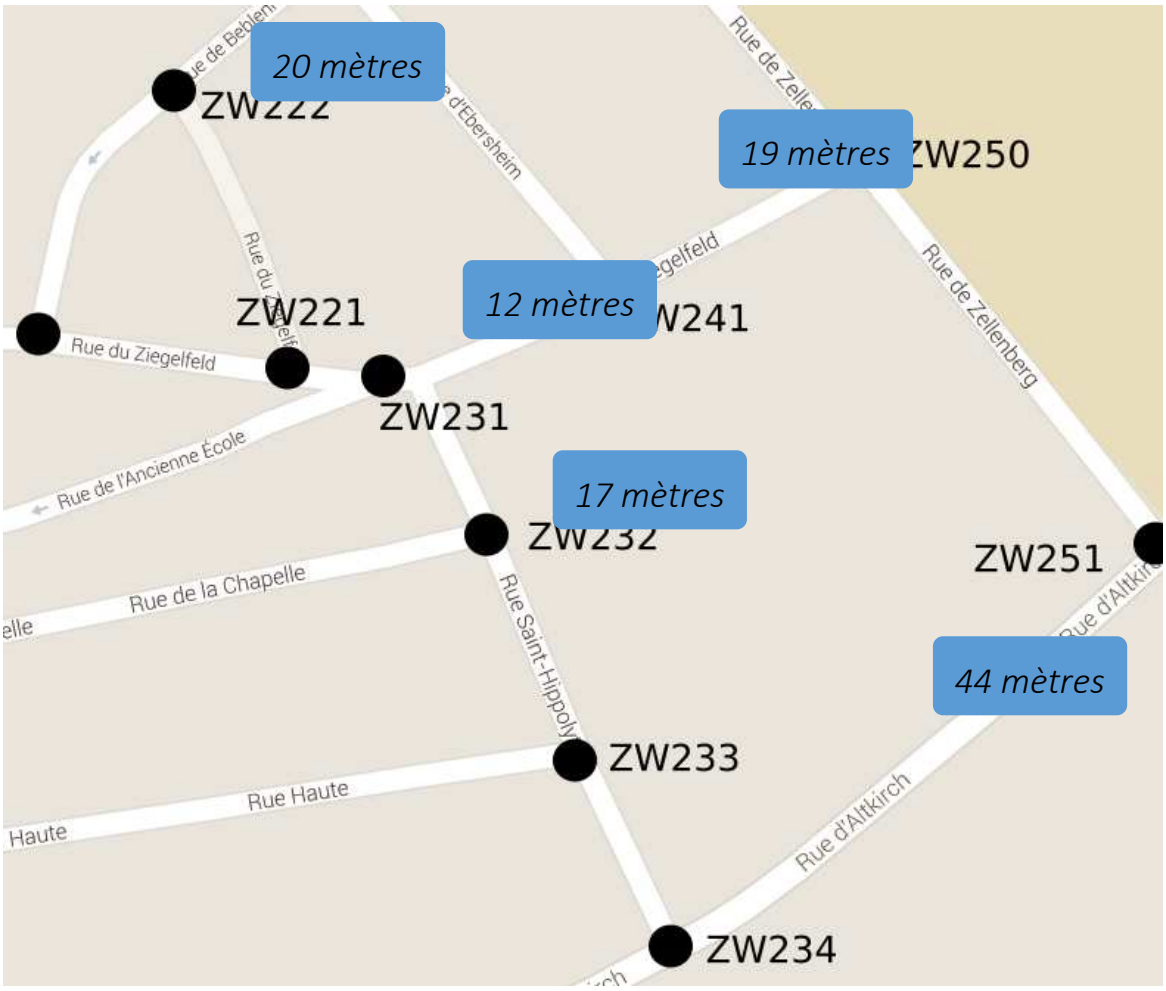
Travaux Pratiques
séance nr. 6 - Graphe de voierie urbaine (suite)

Ce TP propose une suite au TP numéro trois précédent.

On souhaite toujours modéliser un réseau urbain. Pour cela on construit un graphe valué dont les sommets sont les carrefours et les arêtes les rues. Chaque carrefour possède un identifiant : 2 lettres suivi de 3 chiffres (un nombre). Chaque rue possède un nom (une chaîne de caractères de longueur arbitraire), mais aussi une **longueur en mètres**.

Le schéma ci-dessous présente toujours le même exemple, mais avec des distances entre carrefours.



On utilisera toujours une représentation par liste d'adjacence, mais on ajoute à chaque arc une distance en mètres (ici c'est le champ ***lg***):

```
typedef struct strarc {unsigned int icf, irue, lg; struct strarc *suc;} Strarc, *ListeSom;

typedef struct {unsigned int icf; ListeSom lsucc, lpred; }  Sommet;

typedef struct strsom {Sommet  c; struct strsom *suiv;} Strsom, *GrapheUrbain;
```

3.1 Adapter les opérations de création et destruction d'un ***GrapheUrbain***., d'adjonction / suppressions de rue et de carrefour, d'adjonction d'une arête au graphe (il faut un paramètre de distance en plus). Les autres opérations sont identiques: existence d'une rue, existence d'un carrefour, existence d'une connexion entre deux carrefours, etc.

3.2 Ecrire à présent une opération permettant d'obtenir la liste de tous les carrefours que l'on peut atteindre par un chemin à partir d'un carrefour ***icar*** donné. On utilisera un **parcours en profondeur d'abord** avec un **tableau** pour faire le marquage des sommets déjà visités.

```
ListeSom  atteindre(GrapheUrbain g, unsigned int icar) ;
```

La liste renvoyée est de type ***ListeSom***. Elle contient tous les carrefours ***k*** que l'on peut atteindre à partir du carrefour ***icar***. Dans cette liste, le champ ***lg*** de chaque maillon ne contiendra non plus la longueur de l'arc, mais **la distance parcourue** depuis ***icar*** pour atteindre ***k***. Utiliser une opération récursive auxiliaire.

3.3 Ecrire une opération qui renvoie **tous les chemins** qui permettent d'atteindre un carrefour ***iarrivee*** à partir d'un carrefour de départ ***idepart***. Cette opération renvoie **une liste de listes**, càd une liste de chemins, un chemin étant de type ***ListeSom***.

Pour tous les chemins ***ListeSom*** renvoyés, la tête sera icf=***idepart*** et la queue icf=***iarrivee***. Le champ ***lg*** est utilisé comme précédemment pour indiquer la longueur du chemin parcouru. Donc lg=0 pour la tête de liste. Pour la queue de liste, ***lg*** contient la distance parcourue pour atteindre ***iarrivee***.

Pour renvoyer une liste de ***ListeSom***, on ne créera pas de nouveau type liste. On réutilisera simplement la structure ***Strsom***, qui est déjà une liste de liste : dans le champ ***c*** de type ***Sommet,*** icf=***idepart*** et on utilisera ***lsucc*** pour encoder les chemins (lpred=NULL ne sera pas utilisé). Du coup, la fonction renvoie un ***GrapheUrbain*** :

```
GrapheUrbain tousleschemins(GrapheUrbain g, unsigned int idepart, unsigned int iarrivee);
```

3.4 Implanter une opération ***pluscourtchemin*** qui, à partir d'un carrefour ***idepart*** donné, renvoie le chemin de plus courte distance permettant d'atteindre un autre carrefour ***iarrivee***. Cette opération renvoie une ***ListeSom***. Le champ ***lg*** contient la longueur parcourue. Utiliser l'opération précédente (et non pas Dijkstra) : il suffit de rechercher dans la liste de listes, celle dont le maillon en queue contient la valeur ***lg*** la plus petite :

```
ListeSom pluscourtchemin(GrapheUrbain g, unsigned int idepart, unsigned int iarrivee);
```