I Croisements

On modélise ici un réseau routier par un ensemble de *croisements* et de *voies* reliant ces croisements. Les voies partent d'un croisement et arrivent à un autre croisement. Ainsi, pour modéliser une route à double sens, on utilise deux voies circulant en sens opposés.
La base de données du réseau routier est constituée des relations suivantes :

- Croisement(id, longitude, latitude)
- Voie(id, longueur, id_croisement_debut, id_croisement_fin)

Dans la suite on considère c l'identifiant (`id`) d'un croisement donné.

❑ **Q26** – Écrire la requête SQL qui renvoie les identifiants des croisements atteignables en utilisant une seule voie à partir du croisement ayant l'identifiant c.

❑ **Q27** – Écrire la requête SQL qui renvoie les longitudes et latitudes des croisements atteignables en utilisant une seule voie, à partir du croisement c.

❑ **Q28** – Que renvoie la requête SQL suivante ?

```
1  SELECT V2.id_croisement_fin
2  FROM   Voie as V1
3  JOIN   Voie as V2
4  ON     V1.id_croisement_fin = V2.id_croisement_debut
5  WHERE  V1.id_croisement_debut = c
```

II Typographie

Voici la définition de quelques termes utiles pour la suite :

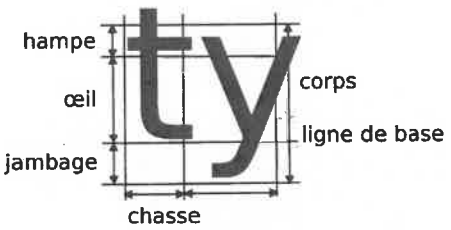
— un **caractère** est un signe graphique d'un système d'écriture, par exemple le *caractère latin a majuscule* « A ». Le standard Unicode donne à chaque caractère un nom et un identifiant numérique, appelé *point de code*, que nous appellerons ci-après simplement *code*. Le code de « A » dans la représentation Unicode est 65. La version 13.0 publiée en mars 2020 répertorie 143 859 caractères couvrant 154 systèmes d'écriture, modernes ou historiques comme les hiéroglyphes ;

— un **glyphe** est un dessin particulier représentant un caractère, par exemple pour le *caractère latin a majuscule* : A (roman) *A* (italique) $\mathcal{A}$ (caligraphié) **A** (gras), **A** (courrier)...

— une **police** de caractères est un ensemble coordonné de glyphes incluant différentes variantes (style roman ou italique, graisse...) et permettant de représenter un texte complet dans un système d'écriture. La police de ce document est *Computer Modern*, la police par défaut de LaTeX ;

— une **famille** est un groupe de polices. La classification Vox-ATypI, proposée par Maximilien Vox en 1952 et adoptée par l'Association typographique internationale, contient 11 familles. La police *Computer Modern* fait partie de la famille *Didone*.

Le corps du glyphe est sa hauteur, la chasse est sa largeur. Le corps est décomposé en trois parties : l'œil qui contient typiquement les petites lettres, le jambage et la hampe qui recouvrent les dépassements en dessous ou au dessus de l'œil. La limite inférieure de l'œil est la ligne de base. Elle définit l'alignement des caractères. La chasse peut être fixe (polices monospaces) ou variable.



La description vectorielle d'un glyphe est définie de la façon suivante :

— un **point** `p` est repéré par ses coordonnées (abscisse, ordonnée) dans le plan orthonormé classique, et sera représenté par une liste de deux flottants ;

— une **multi-ligne** `l` est une séquence de points reliés par des segments, représentée par une liste de points, éventuellement restreinte à un seul point ;

— la **description vectorielle** `v` d'un glyphe est un ensemble non vide de multi-lignes, représenté par une liste de multi-lignes.

Les descriptions vectorielles seront supposées normalisées de sorte que la ligne de base corresponde à l'ordonnée 0, que la hauteur de l'œil soit 1, et enfin que le glyphe soit collé à l'abscisse 0, sans dépassement vers les abscisses négatives.

Concrètement, la description vectorielle d'un glyphe est une liste de listes de listes de 2 flottants. À titre d'illustration, voici une description vectorielle d'un glyphe, composée de deux multi-lignes .

```
v = [ [ [ 0.25, 1.0 ], [ 0.25, -1.0 ], [ 0.0, -1.0 ] ], [ [ 0.25, 1.25 ] ] ]
```

~~Partie II – Gestion de polices de caractères vectorielles~~

Une base de données stocke les informations liées aux polices de caractères dans 4 tables ou relations.

Famille décrit les familles de polices, avec `fid` la clé primaire entière et `fnom` leur nom.

Police décrit les polices de caractères disponibles, avec `pid` la clé primaire entière, `pnom` le nom de la police et `fid` de numéro de sa famille.

Caractere décrit les caractères, avec `code` la clé primaire entière, `car` le caractère lui-même, `cnom` le nom du caractère.

Glyphe décrit les glyphes disponibles, avec `gid` la clé primaire entière, `code` le code du caractère correspondant au glyphe, `pid` le numéro de la police à laquelle le glyphe appartient, `groman` un booléen vrai pour du roman et faux pour de l'italique et `gdesc` la description vectorielle du glyphe.

Voici un extrait du contenu de ces tables.

Famille

fid	fnom
1	Humane
2	Garalde
3	Réale
4	Didone
5	Mécane
6	Linéale
...	...

Police

pid	pnom	fid
1	Centaur	1
2	Garamond	2
3	Times New Roman	3
4	Computer Modern	4
...	...	...
21	Triangle	6
...	...	...

Caractere

code	car	cnom
65	A	lettre majuscule latine a
66	B	lettre majuscule latine b
...	...	...
97	a	lettre minuscule latine a
98	b	lettre minuscule latine b
99	c	lettre minuscule latine c
...	...	...

Glyphe

gid	code	pid	groman	gdesc
1	65	20	True	`[ [ [0, 0], [1, 2], [2, 0] ], [ [0.5, 1], [1.5, 1] ] ]`
2	65	20	False	`[ [ [0, 0], [2, 2], [2, 0] ], [ [1, 1], [2, 1] ] ]`
...	...	...	...	...
501	97	21	True	`[ [ [0, 0], [0.5, 1], [1, 0], [0, 0] ] ]`
502	98	21	True	`[ [ [0, 2], [0, 0], [1, 0.5], [0, 1] ] ]`
503	99	21	True	`[ [ [1, 1], [0, 0.5], [1, 0] ] ]`
504	100	21	True	`[ [ [1, 2], [1, 0], [0, 0.5], [1, 1] ] ]`
...	...	...	...	...

❑ **Q3** Proposer une requête en SQL sur cette base de données pour compter le nombre de glyphes en roman (cf. description précédente).

❑ **Q4** Proposer une requête en SQL afin d'extraire la description vectorielle du caractère *A* dans la police nommée Helvetica en italique.

❑ **Q5** Proposer une requête en SQL pour extraire les noms des familles qui disposent de polices et leur nombre de polices, classés par ordre alphabétique.

Marchons, marchons, marchons...

Ce sujet propose d'appliquer des techniques informatiques à l'étude de trois marches de nature différente :

— une marche concrète (partie I - Randonnée)
— une marche stochastique (partie II - Mouvement brownien d'une petite particule)
— une marche auto-évitante (partie III - Marche auto-évitante)

Les trois parties sont indépendantes. L'annexe à la fin du sujet (pages 8 à 10) fournit les canevas de code en Python que vous devrez reprendre dans votre copie pour répondre aux questions de programmation.

Partie I. Randonnée

Lors de la préparation d'une randonnée, une accompagnatrice doit prendre en compte les exigences des participants. Elle dispose d'informations rassemblées dans deux tables d'une base de données :

— la table `Rando` décrit les randonnées possibles – la clef primaire entière `rid`, son nom, le niveau de difficulté du parcours (entier entre 1 et 5), le dénivelé (en mètres), la durée moyenne (en minutes) :

rid	rnom	diff	deniv	duree
1	La belle des champs	1	20	30
2	Lac de Castellane	4	650	150
3	Le tour du mont	2	200	120
4	Les crêtes de la mort	5	1200	360
5	Yukon Ho !	3	700	210
...	...	...	...	...

— la table `Participant` décrit les randonneurs – la clef primaire entière `pid`, le nom du randonneur, son année de naissance, le niveau de difficulté maximum de ses randonnées :

pid	pnom	ne	diff_max
1	Calvin	2014	2
2	Hobbes	2015	2
3	Susie	2014	2
4	Rosalyn	2001	4
...	...	...	...

Donner une requête SQL sur cette base pour :

❑ Q1 – Compter le nombre de participants nés entre 1999 et 2003 inclus.

❑ Q2 – Calculer la durée moyenne des randonnées pour chaque niveau de difficulté.

❑ Q3 – Extraire le nom des participants pour lesquels la randonnée n°42 est trop difficile.

❑ Q4 – Extraire les clés primaires des randonnées qui ont un ou des homonymes (nom identique et clé primaire distincte), sans redondance.