Les six parties qui suivent sont indépendantes.

I Base de donnée en pySQL

L'objectif de ce sujet est de recréer en Python, en utilisant dans la mesure du possible des fonctions d'ordre supérieur, un fragment du language `SQL`. En particulier, on veut pouvoir effectuer la requête suivante:

```
SELECT name_actor FROM data_actors INNER JOIN data_movie ON known_for=title
                  WHERE rate > 7.5 ORDER BY rate DESC LIMIT 2
```

Pour cela, on modélise une base de donnée comme une liste de dictionnaires – tel que les entrées des dictionnaires d'une même liste soient identiques – et une requête comme une liste d'instructions. Exécuter une requête revient alors à appliquer, à une base de donnée initialement vide, les instructions une à une.

▸ **Question 1.** *Expliquer pourquoi, et comment, réordonner une requête pour que* `execute_query` *fonctionne correctement. Vous complèterez* `sorted_query = ...`.

▸ **Question 2.** *Pour chacune des instructions possibles, complétez les lambda fonctions. On rappelle que* `SELECT` *permet de sélectionner une colonne d'une table,* `FROM` *permet de charger une table,* `INNER JOIN` *de joindre deux tables en faisant correspondre les élements de deux colonnes,* `WHERE` *de filtrer les lignes de la table,* `ORDER BY DESC` *de trier les lignes (par ordre décroissant) et* `LIMIT` *de ne garder que les premiers résultats.*

À toute fin utile, on rappelle que: `|` *permet de faire l'union de deux dictionnaires, que la fonction* `eval(s: str)` *prend une string et renvoie l'objet python associé (typiquement une variable ou une fonction), que la fonction* `filter(f, l)` *filtre la liste* `l` *en appliquant le filtre (fonction booléenne)* `f` *et renvoie un object de type* `map` *(qu'on peut convertir en liste avec la fonction* `list`*), et enfin que* `sorted(l, key=...)` *permet de trier la liste* `l` *selon les valeurs de* `[key(x) for x in l]`.

```
my_query = [ ("SELECT", "name")
           , ("FROM", "data_actors")
           , ("INNER JOIN", "data_movie", "known_for", "title")
           , ("WHERE", "rate", lambda r: r>7.5)
           , ("ORDER BY DESC", "rate")
           , ("LIMIT", 2)
       ]

instruction_weight = { "SELECT": 0 } # TODO

instruction_pgrm = { "SELECT": lambda x: x } # TODO

def execute_query(query):
    sorted_query = query # TODO
    db = []
    print("Answer database at beginning: []")
    for instruction in sorted_query:
        print("Instruction to be performed:", instruction)
        db = [] # TODO
        print("Answer database after instruction:", db)
    return db

def check():
    return execute_query(my_query) == ["Mark Hamill", "Roy Scheider"]
```

II Représentation d'images carrées par arbres binaires

Dans cet exercice, sauf mention contraire, on considère des images carrées de largeur en pixel une puissance de deux.

▸ **Question 1.** *Expliquer comment représenter, comme vous le feriez devant vos élèves, une image à l'aide d'arbres binaires complets. Vous commencerez par rappeler ce qu'est un arbre binaire complet.*

▸ **Question 2.** *Comparez cette structure de donnée par rapport à la représentation naïve (en listes de listes) d'une image. Justifer ensuite l'intérêt de cette structure de donnée si l'on ne se restreint plus aux arbres binaires complets. Vous vous appuierez sur un exemple bien choisi.*

▸ **Question 3.** *Décrire les algorithmes récursifs permettant, pour une image stockée sous la forme d'un arbre binaire, de réaliser: une symmétrie horizontale, une symmétrie verticale, une rotation de 180°.*

▸ **Question 4.** *Décrire l'algorithme permettant de convertir une image représentée par une liste en liste en une image représentée par un arbre binaire.*

▸ **Question 5.** *Peut-on adapter l'algorithme précédent pour des images rectangulaires dont les côtés ne sont plus nécessairement des puissances de 2 ?*

III Distance de Levenshtein

La distance de Levenshtein donne une mesure de la différence entre deux chaines de caractères. Elle correpond au nombre minimal de changements (suppression, insertion ou remplacement de caractère) qu'il faut faire pour passer de l'une à l'autre. Par exemple, $\mathsf{lev}(examen, examen) = 0$ et $\mathsf{lev}(corne, corse) = 1$.

▸ **Question 1.** *Que vaut* $\mathsf{lev}(niche, chiens)$ *?*

▸ **Question 2.** *Proposer une définition inductive de la distance de Levenshtein entre deux mots (pas nécessairement de même taille).*

▸ **Question 3.** *En déduire un algorithme récursif pour calculer* lev*.*

▸ **Question 4.** *Critiquer cet algorithme. Vous pourrez faire un parallèle avec une implémentation naïve de la fonction de Fibonacci.*

▸ **Question 5.** *Proposer, brièvement, une amélioration de l'algorithme. Comment s'appelle le mécanisme que vous avez utilisé ?*

▸ **Question 6.** *Quelle utilisation de cette distance pouvez-vous imaginer ? Vous pourrez par exemple proposer des rafinements de cette distance dans le cadre de cette utilisation.*

IV Arbre couvrant minimal

Soit $G = (S, A)$ un graphe non orienté dont les arêtes sont pondérées. Un arbre couvrant de poids minimal est sous-graphe T de G tel que T est un arbre connexe ayant pour sommets les sommets de G, et tel que la somme des poids des arêtes qui le compose soit minimale.

▸ **Question 1.** *A-t-on unicité de l'arbre couvrant minimal d'un graphe ?*

On considère deux algorithmes, connus respectivement comme l'algorithme de Prim et l'algorithme de Kruskal, qui permettent d'obtenir un arbre couvrant minimal.

- **Prim.** L'*arbre couvrant* contient initialement un sommet du graphe. Itérativement, on ajoute à l'*arbre couvrant* une arête du graphe de poids minimal *de sorte qu*'il reste un arbre (on ajoute donc une arête adjacente à l'arbre existant qui ne crée pas de cycle). On retourne l'*arbre couvrant*.
- **Kruskal.** Le *graphe couvrant* contient initialement un sommet du graphe. Itérativement, on ajoute au *graphe couvrant* une arête du graphe de poids minimal *de sorte qu*'elle ne crée pas de cycle dans le *graphe couvrant*. On retourne le *graphe couvrant*, qui s'avère être un arbre couvrant.

▸ **Question 2.** *Déroulez l'exécution de ces algorithmes sur un graphe exemple.*

▸ **Question 3.** *Formalisez une des deux algorithmes. Vous pourrez vous inspirer de l'algorithme de Dijkstra et de sa file de priorité.*

▸ **Question 4.** *Quelle est la complexité de ces algorithmes ?*

▸ **Question 5.** *Justifiez que le graphe retourné par l'algorithme de Kruskal est un arbre couvrant.*

▸ **Question 6.** *Montrez que l'algorithme de Kruskal renvoie un arbre couvrant de poids minimal. Vous pourrez considérer l'invariant suivant: en notant H le sous-graphe de G sélectionné à une itération de l'algorithme donnée, il existe un arbre couvrant minimal de G qui contient H et ne contient aucune des arêtes jusqu'alors rejetées par l'algorithme.*

V Tri inconscient

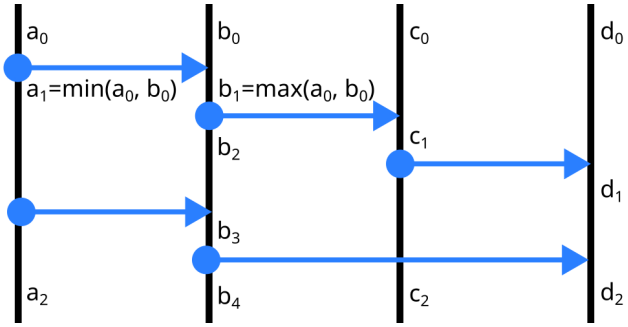
On se place dans le cadre où un utilisateur accède à des données chiffrées sur un serveur. Remarquons d'abord que cela n'empêche pas le serveur d'obtenir des informations sur les données (chiffrées) qu'il stocke.

▸ **Question 1.** *Supposons qu'un hôpital cherche souvent la liste de ses patients sur une tranche d'âge. Lors de connexions successives au serveur, l'hôpital demande accès aux fichiers patients (a, b, c), puis (b, c, d), puis (c, d). Que peut déduire le serveur ?*

On s'intéresse maintenant à des algorithmes qui ne sont pas vulnérables à ce genre d'attaque. En particulier, on va se concentrer sur un algorithme de tri *inconscient*. Un algorithme est dit *inconscient* si pour toutes entrées x et y, l'algorithme appliqué à chacune de ces entrées effectue des accès mémoires indistingables.

▸ **Question 2.** *Donnez un exemple de tri qui n'est pas inconscient. Donnez un exemple de tri inconscient.*

On peut intuiter de la question précédente que l'opération de base d'un tri inconscient semble être le compareAndSwap: étant donné un couple (x, y), l'opérateur renvoie (x, y) lorsque $x < y$ et (y, x) sinon. On appelle *réseau de comparateur* un réseau d'opérateurs compareAndSwap sur des entrées fixées. Voici, comment les représenter graphiquement :



On appelle *taille* d'un réseau le nombre de ses comparateurs, et *profondeur* d'un réseau le nombre maximal de comparateur par lesquels une entrée peut passer.

▸ **Question 3.** *Représenter le réseau de comparaison pour le tri inconscient donné en question 2 sur un jeu de quatre entrées. Quelle est, asymptotiquement, la taille et la profondeur du réseau ?*

V.1 Tri de séquences bitoniques

Une séquence binaire est dite *bitonique* si c'est une séquence croissante-puis-décroissante, ou une permutation circulaire du cas précédent. Formellement, c'est un mot de $0^+1^*0^* + 1^+0^*1^*$. On appelle *half-cleaner* le réseau de taille d'entrée $2n$ formé des n comparateurs $\{(i, n+i) \mid i \in [\![1, n]\!]\}$. On admettra que, pour une entrée bitonique, les deux moitiées de la sortie du half-cleaner sont bitoniques. De plus, une des deux moitiées est uniquement constituées de 0 ou de 1.

▸ **Question 4.** *En déduire un réseau de comparateur (récursif) pour du tri binaires bitoniques.*

V.2 Tri de séquences binaires quelconques

▸ **Question 5.** *Proposer un réseau de comparateurs (récursif) pour du tri de séquences binaires quelconques. Quelle est sa taille ?*

VI En vrac

▸ **Question 1.** *À la suite des analogies* files/files d'attente *et* piles/piles d'assiettes*, une élève vous demande si les files de priorité sont analogues aux files VIP dans les parcs d'attractions. Que lui répondez-vous ?*

▸ **Question 2.** *On associe souvent le problème du parcours Hamiltonien à celui du voyageur de commerce. Proposez, après avoir redéfini ce parcours, un "problème illustré" pour le problème du parcours Eulérien.*

▸ **Question 3.** *Proposer un circuit pour 4 bits d'entrées qui renvoie* 1 *si et seulement si* $\overline{b_3 b_2 b_1 b_0}^{10}$ *(l'interprétation décimale du mot de quatre bit en entrée,* b_0 *étant le bit de poids faible) est un multiple de 3. Vous utiliserez une technique générique, basée sur les tables de vérité.*

▸ **Question 4.** *Sur la rive d'un fleuve se trouvent un loup, une chèvre, un chou et un passeur. Dans ce problème bien connu, l'objectif est de faire passer tout ce petit monde sur l'autre rive. Le passeur dispose d'une barque sur laquelle il ne peut embarquer qu'un seul des autres protagonistes. Mais attention, en l'abscence du passeur, la chèvre mangerait le choux, et le loup la chèvre.*

Proposer une modélisation du problème par une structure informatique classique, et proposer un algorithme de résolution de l'énigme (que vous ferez tourner à la main).