DS 2bis

Informatique de tronc commun, classe de PC*

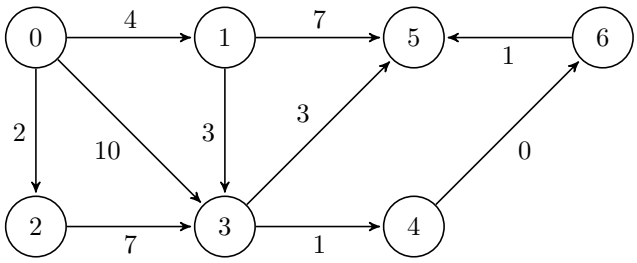
Julien Reichert

Ce devoir consiste en plusieurs parties indépendantes.

Partie A - Graphes et algorithme de Dijkstra

Nous allons considérer ici un problème de recherche de chemin optimal en poids dans un graphe, avec la contrainte supplémentaire qu'en cas d'égalité de poids le chemin utilisant le moins d'arcs sera privilégié. En cas de nouvelle égalité, n'importe quel chemin ainsi optimal pourra être retenu.

Les graphes seront représentés par liste d'adjacence et les sommets seront les entiers naturels de `0` au nombre de sommets moins un. Ainsi, pour le graphe `g` ci-après, la liste `g[1]` contient les couples `(3, 3)` et `(5, 7)`, car il existe un arc depuis le sommet appelé `1` vers le sommet appelé `3`, et que cet arc est de poids `3`, ainsi qu'un arc depuis le sommet appelé `1` vers le sommet appelé `5`, et que cet arc-là est de poids `7`.



Il est important de noter que la liste associée à un sommet n'est pas forcément croissante, ni selon les sommets ni selon les poids.

Une première possibilité pour répondre au problème posé en introduction est d'utiliser une comparaison naturelle sur les couples, les distances au sommet de références étant alors stockées en tant que couples `(d, l)` où `d` est le poids minimal actuellement recensé vers le sommet concerné et `l` le nombre minimal d'arcs empruntés par un chemin ayant ce poids. Mais pour la beauté des raisonnements informatiques et des encodages de plusieurs informations en une valeur, une autre possibilité sera explorée ici.

Nous allons donc transformer le graphe pour que les poids soient remplacés par une valeur hybride égale à dix puissance `k` fois le poids d'origine plus un, de sorte que dix puissance `k` soit strictement supérieur au nombre de sommets. Ainsi, le nouveau poids d'un chemin de taille limitée au nombre de sommets sera dix puissance `k` fois le poids d'origine du chemin plus le nombre d'arcs empruntés, permettant de distinguer par ce critère les chemins de même poids dans le graphe de base, tout en évitant des valeurs exceptionnelles dues à une retenue car s'il existe un chemin optimal en poids, il en existe aussi un qui ne passe au plus qu'une fois par un même sommet. Si le nombre de sommets est `12` et le poids d'un arc est `9`, le nouveau poids sera donc `901`.

Exercice A1 : Écrire une fonction prenant en entrée un entier positif `n` et qui détermine la plus petite puissance de dix strictement supérieure à `n`.

Attention dans l'exercice A1 : pour `n` valant `42`, c'est `100` qu'il faudra retourner, et non pas `2` qui est l'exposant.

Exercice A2 : Écrire une fonction prenant en entrée un graphe pondéré `g` donné par liste d'adjacence et qui renvoie le graphe `g2` obtenu en modifiant tous les poids selon le principe évoqué précédemment.

Exercice A3 : Ci-après figure une implémentation de l'algorithme de Dijkstra pour un graphe quelconque (qui peut être aussi bien `g2` que `g` en pratique). Compléter les lignes marquées par des points de suspension. On signale que `distances` détermine la distance minimale du sommet de référence à tout sommet et que `distancesbis` simule une file de priorité avec tous les sommets non encore totalement traités.

```
def dijkstra(g, s):
    distances = [None] * len(g)
    distancesbis = {}
    distances[s] = 0
    distancesbis[s] = 0
    while distancesbis != {}:
        mini, le_mini = None, None
        for sommet in distancesbis:
            if mini is None or distancesbis[sommet] < mini:
                mini = distancesbis[sommet]
                le_mini = sommet
        distancesbis.pop(le_mini)
        for (t, p) in g[le_mini]:
            ...
    return distances
```

Exercice A4 : Écrire alors une fonction prenant en entrée un graphe `g` et deux sommets `s` et `t` et qui détermine le poids minimal d'un chemin de `s` à `t` ainsi que la nombre minimal d'arcs d'un chemin ayant ce poids.

Exercice A5 : Expliquer les adaptations qu'il faudra faire au niveau des programmes précédents si on avait voulu en priorité le chemin passant par le plus petit nombre d'arcs, et en cas d'égalité celui de poids moindre.

Partie B - Algorithme des k plus proches voisins

On redonne ci-dessous une version de l'algorithme des k plus proches voisins dans un cas assez général où la fonction de distance est en argument, ce qui permet de considérer en particulier une dimension quelconque.

```
def kNN(les_points, point_sup, k, dist):
    distances = [(dist(point[0], point_sup), point[1]) for point in les_points]
    pass
    etiquettes = dict()
    for _, etiq in distances[:k]:
        if etiq in etiquettes:
            etiquettes[etiq] += 1
        else:
            etiquettes[etiq] = 1
    rep = etiq # Initialisation pertinente plutôt que None
    for e in etiquettes: # Moche de réutiliser le nom etiq, mais ça marcherait
        if etiquettes[e] > etiquettes[rep]:
            rep = e
    return rep
```

Exercice B1 : L'instruction `pass` a remplacé une ligne de code cruciale. Quel était l'effet de cette ligne ?

Exercice B2 : Écrire une fonction de calcul de la distance euclidienne (à dimension fixée ou non, au choix) qui puisse être compatible avec la fonction `kNN`. Écrire alors un élément possible de la liste `les_points`, en expliquant la structure.

Exercice B3 : Que vaut précisément `etiq`, dans la ligne qui initialise `rep`, par rapport aux arguments de la fonction ?

Exercice B4 : On souhaite modifier la fonction de sorte que plus un des k plus proches voisins est proche, plus il va contribuer pour le choix de son étiquette. La formule suggérée est de le compter pour l'inverse de la distance au point supplémentaire, en traitant le cas particulier d'une distance nulle : dans ce cas l'étiquette sera forcément retenue (mathématiquement cohérent...). On admettra que les points fournis sont distincts deux à deux. Écrire une nouvelle version de la fonction `kNN`. On pourra signaler les modifications à faire, pour aller plus vite.

Partie C - Bases de données

On considère une table `Resultats` pouvant matérialiser des épreuves quelconques. Les attributs sont limités à une date `Date` de type chaîne de caractères de taille limitée à 20, un identifiant `Id` et à un score `Score`, tous les deux de type entier. Le couple `(Date,Id)` est une clé primaire.

Un exemple d'enregistrement est donc `("2023-03-02", 42, 19)`, pour un score de `19` réalisé par la personne d'identifiant `42` le 2 mars 2023.

Pour chacun des exercices, si on demande de récupérer des informations, il n'y a pas de pénalité au cas où des informations supplémentaires sont collectées, sous réserve de ne pas provoquer d'ambiguïté voire d'erreur SQL.

On utilisera exclusivement une version allégée de SQL (malgré une certaine tolérance pour MySQL).

Exercice C1 : Écrire une requête qui permet de récupérer le score de la personne d'identifiant `19` à l'épreuve d'hier.

Exercice C2 : Écrire une requête qui permet de récupérer toutes les dates des épreuves auxquelles la personne d'identifiant `5` a participé.

Exercice C3 : Écrire une requête qui permet de récupérer le nombre de dates où un résultat a été enregistré.

Exercice C4 : Écrire une requête qui permet de récupérer le score maximum réalisé, la date à laquelle il a été réalisé et l'identifiant de la personne qui l'a réalisé.

Exercice C5 : Écrire une requête qui permet de récupérer le score maximum réalisé à une date notée `d`.

Exercice C6 : Écrire une requête qui permet de déterminer le nombre de personnes ayant fait mieux qu'un score `s` donné à une date notée `d`.

Exercice C7 : Écrire une requête qui permet de déterminer le nombre de personnes ayant fait mieux que le score d'une personne dont l'identifiant `n` est donné à une date notée `d`.

Exercice C8 : Écrire une requête qui permet de récupérer l'ensemble des couples d'identifiants de personnes telles que le score de la première soit strictement supérieur au score de la seconde, le tout à une date notée `d`.

Exercice C9 : Déduire des exercices qui précèdent une requête qui permet d'obtenir le classement à la date `d` [1] en attribuant la même place à des égalités (l'ordre d'affichage en cas d'égalités n'importe pas). On comprendra que la place est à inclure dans le résultat de la requête.

Exercice C10 : Écrire une requête qui permet de récupérer les identifiants des personnes ayant participé à toutes les épreuves existantes.

1. ou le classement général si on préfère