Devoir d'informatique numéro 1

le jeudi 5 octobre, durée deux heures

Compression et décodage d'un message

Mines-Ponts 24

Les questions qui suivent ont été sélectionnées dans un énoncé du concours Mines-Ponts en en éliminant certaines. Ne soyez donc pas surpris que les numéros des questions ne soient pas consécutifs. Les deux parties du problème peuvent être traitées indépendamment et la seconde concerne un graphe pondéré.

Rappels concernant le langage Python : il n'est pas possible d'utiliser des fonctions internes à Python sur les listes ou les chaînes de caractères telles que **min**, **max**, `count`, `remove` ... Seules les instructions basiques telles que **len**`(liste)`, `liste.append(e)` sont autorisées. Le tranchage ou slicing ainsi que la concaténation sont également permis. Les programmes doivent être commentés lorsque c'est nécessaire pour justifier les choix. Il n'est pas utile de rappeler la signature des fonctions demandées dans les différentes questions.

On s'intéresse au problème de communication entre deux personnes, nommées Alice et Bob qui cherchent à s'envoyer un message au travers d'un canal de communication (une bande de fréquences radio par exemple).

Avant d'être lu par Bob, le message original d'Alice passe par plusieurs étapes que nous allons séparer de la manière suivante :

— une phase de **compression**, durant laquelle Alice cherche à trouver la représentation la plus compacte possible du message,
— une phase d'**encodage** durant laquelle le message compressé est transformé en une succession de symboles transmissibles au travers du canal de communication utilisé,
— une phase de **transmission** durant laquelle le message encodé circule sur le canal de communication et est susceptible de subir une altération,
— une phase de **décodage** durant laquelle Bob décode le message qu'il a reçu, le message lui apparaît alors sous la forme compressée,
— une phase de **décompression** durant laquelle Bob applique l'opération réciproque de la compression opérée par Alice.

Ce modèle est décrit par le schéma de la Figure 1 :

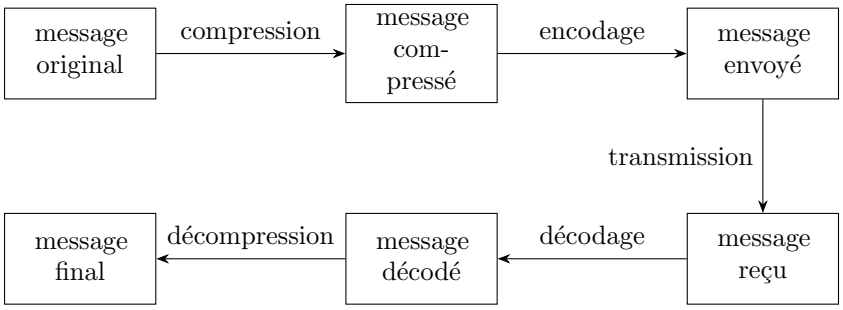


FIGURE 1 – Schématisation du modèle de communication considéré ici.

Dans cette épreuve, nous allons nous intéresser uniquement à deux phases : la compression du message d'origine par Alice en un message compact et le décodage par Bob d'un message transmis, potentiellement entaché d'erreurs.

1.1 Analyse du texte source

L'objet de cette partie est d'analyser le contenu d'une chaîne de caractères `s` afin de déterminer :

— les caractères utilisés par la chaîne `s` ;

— le nombre d'occurrences de chacun.

❑ **Q2** – Écrire une fonction nommée `nbCaracteres(c:str,s:str)->int` qui prend comme argument un caractère `c`, une chaîne `s` et qui renvoie le nombre d'occurrences (c'est-à-dire le nombre d'apparitions) de `c` dans `s`. La fonction doit avoir une complexité linéaire en `n`, la longueur de la chaîne `s`.

❑ **Q3** – Pour déterminer la liste des caractères utilisés à l'intérieur d'une chaîne `s` on utilise la fonction définie ci-dessous :

```
1 def listeCaracteres(s:str):
2     listeCar = []
3     n = len(s)
4     for i in range(n):
5         c = s[i]
6         if not(c in listeCar):
7             listeCar.append(c)
8     return listeCar
```

Que renvoie cette fonction lorsque `s='abaabaca'` ? Expliquer succinctement le principe de fonctionnement de cette fonction.

❑ **Q4** – En fonction de la longueur `n` de la chaîne et du nombre `k` de caractères distincts dans celle-ci, déterminer la complexité asymptotique dans le pire des cas de la fonction de la question **Q3**. Par exemple pour `s='abaabaca'`, on a $n = 8$ et $k = 3$. On négligera la complexité des `append` mais pas celle des tests d'appartenance de la forme `i in L`. Autrement dit, la ligne 7 est considérée comme étant de complexité constante et la ligne 6 de complexité linéaire en la longueur de la liste `listeCar`.

❑ **Q5** – On définit alors une fonction `analyseTexte(s:str)->list`

```
1 def analyseTexte(s:str):
2     R = []
3     l = listeCaracteres(s)
4     for i in range(len(l)):
5         c = l[i]
6         R.append((c, nbCaracteres(c, s)))
7     return R
```

Expliquer ce que fait cette fonction et donner la valeur renvoyée par la commande `analyseTexte('babaaaabca')`.

❑ **Q6** – En fonction de la longueur n de `s` et du nombre k de caractères distincts présents dans `s`, (autrement dit k est la longueur de `listeCaracteres(s)`), donner une estimation de la complexité asymptotique dans le pire des cas de la fonction `analyseTexte`.

La suite de l'énoncé utilise le concept de fréquence statistique d'apparition d'un caractère. C'est un nombre compris entre 0 et 1, défini par le nombre d'occurrences de ce caractère dans les textes écrits en français, divisé par le nombre total de caractères de ces textes. J'ai supprimé la partie 1.2 qui utilise des requêtes SQL pour déterminer ces fréquences.

1.3 Compression

La compression par codage arithmétique consiste à représenter une chaîne de caractères `s` par un nombre réel déterminé à l'intérieur de l'intervalle $[0; 1[$.

Initialement, on attribue à chaque caractère utile une portion de l'intervalle $[0; 1[$ proportionnelle à sa fréquence d'occurrences. Par exemple, pour un alphabet à 5 lettres `'abcde'`, on pourrait avoir un tableau comme ci-dessous :

Caractère	`'a'`	`'b'`	`'c'`	`'d'`	`'e'`
Fréquence	0.2	0.1	0.2	0.4	0.1
Intervalle	$[0; 0.2[$	$[0.2; 0.3[$	$[0.3; 0.5[$	$[0.5; 0.9[$	$[0.9; 1[$

La chaîne de caractères `s` est codée en partant de l'intervalle $[0; 1[$. À chaque caractère successif de celle-ci, on affine cet intervalle en ne considérant que la portion correspondant au caractère lu.

Si par exemple la chaîne à coder est `s = 'dac'` :

— on obtient d'abord l'intervalle $[0.5; 0.9[$ correspondant au caractère `'d'` ;

— le caractère `'a'` détermine alors le sous-intervalle $[0.50; 0.58[$ de $[0.5; 0.9[$ correspondant à la portion associée au caractère `'a'`.

— le caractère `'c'` détermine enfin l'intervalle $[0.524; 0.540[$.

La figure qui suit illustre ce processus.

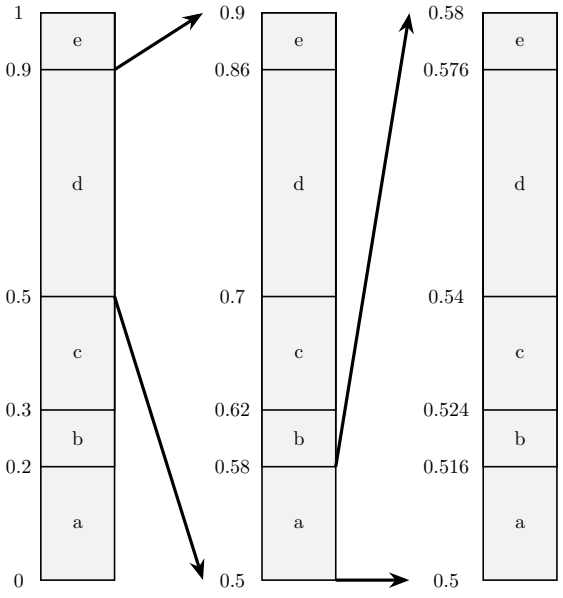


FIGURE 2 – Encodage de `s = 'dac'`

❑ **Q10** – En considérant la table des fréquences précédente, proposer l'intervalle correspondant à la chaîne `s='bac'`.

On suppose disposer d'une fonction `codeCar(car:str,g:float,d:float)->(float,float)` qui prend en argument un caractère `car` et les deux extrémités d'un intervalle $[g, d[$ et qui renvoie un tuple composé des extrémités du sous-intervalle de $[g, d[$ déterminé par le caractère `car`. En reprenant l'illustration précédente `codeCar('b',0,1)` produit `(0.2,0.3)` et `codeCar('a',0.5,0.9)` produit `(0.5,0.58)`.

❑ **Q11** – Écrire une fonction `codage(s:str)->(float,float)` prenant en argument la chaîne `s` et fournissant en réponse le tuple `(g,d)` constitué des deux extrémités de l'intervalle $[g, d[$ produit par l'algorithme de codage précédent.

Le codage arithmétique consiste alors à coder la chaîne `s` par un flottant `x` choisi arbitrairement à l'intérieur de l'intervalle $[g, d[$.

1.4 Décodage

Pour effectuer le décodage d'un flottant `x`, il suffit de repérer dans quelle succession d'intervalles celui-ci se trouve.

À titre d'exemple, reprenons le tableau précédent et considérons le nombre `x=0.123`

Caractère	`'a'`	`'b'`	`'c'`	`'d'`	`'e'`
Fréquence	0.2	0.1	0.2	0.4	0.1
Intervalle	$[0; 0.2[$	$[0.2; 0.3[$	$[0.3; 0.5[$	$[0.5; 0.9[$	$[0.9; 1[$

Puisque `x` appartient à l'intervalle $[0; 0.2[$, le premier caractère est un `'a'`. Puisque `x` appartient au sous-intervalle $[0.10; 0.18[$, le caractère suivant est un `'d'`, *etc.*

❑ **Q12** – Déterminer le caractère qui suit `'ad'` dans la chaîne codée par `x=0.123` en spécifiant le sous-intervalle qui a permis de décoder ce caractère.

Le codage présenté ci-dessus pose une difficulté : deux chaînes distinctes peuvent être représentées par le même code. Par exemple, les chaînes `'b'` et `'ba'` son représentées par 0.2 *et* 0.20 *qui sont égaux. On voit que le problème est associé à la borne gauche égale à 0.*

Une solution possible pour résoudre le problème précédent consiste à introduire un caractère nouveau signifiant la fin de la chaîne de caractères.

Nous conviendrons de désigner ce caractère par `'#'`. Ce caractère se voit attribuer une plage non vide au voisinage de 0. Dans la suite, on suppose que la table des fréquences est adaptée de sorte à prendre en compte la présence de ce nouveau caractère.

On suppose disposer, en plus de la fonction `codeCar(car,g,d)` précédente, d'une fonction `decodeCar(x:float,g:float,d:float)->str` qui détermine le caractère correspondant à la valeur `x` quand celle-ci est comprise dans la plage de $[g, d[$.
Par exemple `decodeCar(0.123,0,1)` donne `'a'` tandis que `decodeCar(0.123,0,0.2)` donne le caractère `'d'`.

❑ **Q14** – Ecrire une fonction `decodage(x:float)->str` produisant la chaîne de caractères `s` déterminée par la valeur de `x` (avec le caractère `'#'` compris).

2 Décodage du message reçu par Bob à l'aide de l'algorithme de Viterbi

2.1 Modélisation du canal de communication par un graphe

Dans cette partie, nous allons désormais considérer que le message compressé par Alice a été envoyé au travers d'un canal de communication. À cette fin, et indépendamment de la phase de compression étudiée dans la première partie, le message a subi une deuxième phase de transformation, dite d'encodage (cf Figure 1).

Le message envoyé sur le canal est une suite de **symboles** à valeurs dans un **alphabet**, noté Σ, comportant K symboles. Le choix d'un alphabet efficace n'est pas l'objet de notre étude et constitue un sujet à part entière.

Suite au passage dans le canal de communication, le message envoyé subit une **altération** de sorte que Bob reçoit une séquence de symboles de Σ qui ne correspond pas nécessairement à celle qui a été émise.

Dans cette partie, nous allons voir une approche permettant à Bob de décoder le message reçu et de potentiellement corriger quelques erreurs liées à la transmission du message et à la connaissance *a priori* de la propension du canal de communication à altérer les symboles du message lors de la transmission.

La modélisation proposée est la suivante :

— Bob observe une suite de N symboles $obs_0, \dots, obs_{N-1}$, que nous allons représenter par une liste Python $Obs = [obs_0, \dots obs_{N-1}]$,

— pour simplifier, on supposera que l'alphabet Σ est un ensemble de K entiers consécutifs commençant à 0, de sorte que $\Sigma = [\![0, K-1]\!]$. Par exemple si $K = 3$ et $N = 8$, un message valide reçu par Bob pourrait être `[2,0,0,2,1,1,0,0]`.

— chacun des symboles observés obs_t correspond à l'altération d'un symbole s_t envoyé par Alice. On note $[s_0, \dots s_{N-1}]$ le message original ; pour reprendre l'exemple précédent, Alice pourrait avoir envoyé `[2,0,0,2,1,1,2,0]`.

— on connaît, pour chaque paire $(i, j) \in \Sigma^2$, la probabilité $E_{i,j}$ que le canal altère le symbole j en un symbole i. On stocke ces probabilités dans une liste de listes `E` ; autrement dit, `E[i][j]` est la probabilité conditionnelle d'observer le symbole i sachant que le symbole j a été émis.

Ici (par exemple) on pourrait considérer : $E = \begin{pmatrix} 0.7 & 0.2 & 0.3 \\ 0.2 & 0.7 & 0.1 \\ 0.1 & 0.1 & 0.6 \end{pmatrix}$, représentée par la liste de listes : `E = [[0.7,0.2,0.3],[0.2,0.7,0.1],[0.1,0.1,0.6]]`.

Le fait que `E[2][0]` vaille 0.1 signifie donc que la probabilité que le symbole observé par Bob soit un 2 sachant qu'Alice a émis un 0 est de 0.1.

— on suppose également que le symbole courant s_t envoyé par Alice a une incidence sur le symbole suivant s_{t+1} qu'elle peut envoyer, au même titre que dans une langue comme le français, la probabilité d'observer un `'t'` dans un mot, n'est pas la même suivant que le caractère précédent est un `'e'` ou un `'z'`.

Ainsi pour chaque couple de symboles $(i, j) \in \Sigma^2$, on suppose que l'on connaît la probabilité d'émettre le symbole j à l'instant $t + 1$ sachant que symbole i a été émis à l'instant t. On suppose également que cette probabilité ne dépend pas de t.

L'information concernant ces probabilités de transition d'un symbole à l'autre peut se stocker dans une matrice P de taille $K \times K$, que l'on représente informatiquement par une liste de listes `P`. Chaque entrée `P[i][j]` donne la probabilité qu'Alice émette le symbole j à l'instant $t + 1$ sachant qu'elle a émis le symbole i à l'instant t. En d'autres termes, `P[i][j]` $= P_{s_t=i}(s_{t+1} = j)$.

On prendra ici à titre d'exemple : $P = \begin{pmatrix} 0.3 & 0.2 & 0.5 \\ 0.4 & 0.4 & 0.2 \\ 0.2 & 0.3 & 0.5 \end{pmatrix}$, représentée par la liste de listes : `P=[[0.3,0.2,0.5],[0.4,0.4,0.2],[0.2,0.3,0.5]]`.

Le fait que `P[2][0]` vaille 0.2 signifie donc que la probabilité que le symbole envoyé par Alice à l'instant $t + 1$ soit un 0 sachant que celui envoyé à l'instant t est un 2 vaut 0.2.

Nous allons désormais nous intéresser au problème du décodage : étant donné la liste $Obs = [obs_0, \ldots, obs_{N-1}]$ des symboles observés par Bob, quelle séquence $\tilde{s}_0, \ldots, \tilde{s}_{N-1}$ est la plus probable ?

En d'autres termes, $\tilde{s}_0, \ldots, \tilde{s}_{N-1}$ est l'*estimation* la plus probable faite par Bob du message d'origine $s_0, \ldots, s_{N-1}$, étant donné les observations $obs_0, \ldots, obs_{N-1}$.

La modélisation précédente peut se représenter à l'aide d'un graphe défini comme suit (voir Figure 3 pour un exemple) :

— on crée un sommet $S_{i,j}$ pour chaque symbole possible $0 \leqslant i \leqslant K-1$ et chaque indice d'observation $0 \leqslant j \leqslant N-1$. Chaque couche verticale dans le graphe correspond à un caractère dans le message. Chaque strate horizontale correspond à un symbole.

— au niveau de la j-ème couche verticale, les sommets $S_{i,j}$ pour $j < N-1$ ont pour successeurs les états $S_{k,j+1}$ pour tous les symboles k possibles,

— par commodité, on ajoute un état source σ correspondant au début du message décodé et un état cible τ correspondant à la fin du message, ces états étant respectivement reliés à la première et la dernière couche,

— le décodage du message envoyé par Alice correspond à un chemin entre σ et τ dans ce graphe. A chaque sommet du chemin correspond une lettre décodée. Par exemple, le chemin passant par $S_{0,0}, S_{2,1}, S_{0,2}, S_{1,3}$, correspond au décodage de $[0, 2, 0, 1]$.

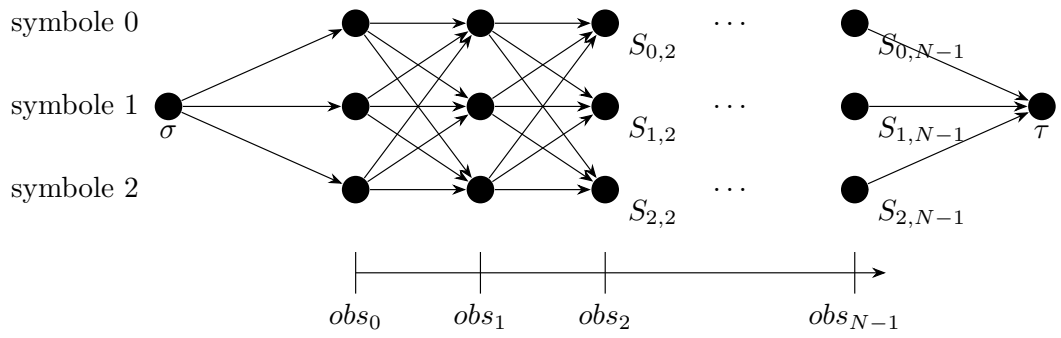


FIGURE 3 – Illustration du modèle de décodage considéré ici.

À chacune des observations obs_i on peut faire correspondre K états. Chacun de ces états correspond à l'un des symboles potentiellement émis par Alice (ici $K = 3$). Les états sont organisés en N couches successives. Chaque état est noté $S_{i,j}$; j correspond à l'indice dans la suite des symboles émis et i au symbole correspondant. Les états de la couche j et les états de la couche $j + 1$ sont reliés par un arc de j vers $j + 1$. On ajoute également, par commodité, un état source σ et un état cible τ respectivement reliés à la première et à la dernière couche.

❑ **Q15** – En fonction de N et de K, donner le nombre de sommets et d'arcs du graphe illustré par la Figure 3. On ne comptera pas les sommets source σ et cible τ, ni les arcs partant du sommet source σ ni ceux arrivant à la cible τ.

On choisit désormais de pondérer chaque arc par la probabilité de transiter par cet arc. Autrement dit :

— les arcs issus de la source σ vers $S_{i,0}$ sont pondérés par $E_{obs_0,i}$ la probabilité d'observer le symbole obs_0 sachant que le symbole i a été émis par Alice ;
— les arcs arrivant à la cible τ sont pondérés par 1 (en fin de message, on transite forcément vers l'état final),
— les arcs internes entre $S_{i,j}$ et $S_{k,j+1}$ sont pondérés par la probabilité $E_{obs_{j+1},k}P_{i,k}$.

La probabilité d'un chemin $\sigma S_{i_0,0}S_{i_1,1}\dots S_{i_{N-1},N-1}\tau$ entre σ et τ est le **produit** des probabilités des arcs qui le composent.

L'objectif va être de trouver le chemin de **probabilité maximale** dans ce graphe entre le sommet source σ et le sommet cible τ.

❑ **Q17** – On revient dans le cas général, N et K sont désormais quelconques. Indiquer combien il existe de chemins entre σ et τ (un ordre de grandeur utilisant la notation O ou Θ est accepté). Préciser si un algorithme d'exploration exhaustive est envisageable dans ce cas.

2.2 Stratégie gloutonne

Pour pouvoir implémenter correctement la recherche du chemin de probabilité maximale, il est utile de disposer d'une fonction auxiliaire qui sera utilisée dès que nécessaire.

❏ **Q18** – Pour une liste `liste`, on appelle argument du maximum et on note `argMax` tout indice i tel que `liste[i]` soit maximal. Proposer une fonction `maximumListe(liste:[float])->(float,int)` qui prend en entrée une liste de nombres et qui renvoie la valeur du maximum de la liste ainsi que le plus petit argument du maximum, *i.e.* le premier indice auquel cette valeur maximale apparaît.

On souhaite appliquer un algorithme glouton pour trouver le chemin de probabilité maximale entre le sommet source σ et le sommet cible τ. On rappelle qu'un algorithme glouton cherche, à chaque étape, à faire le choix localement optimal. Ici, si à une étape on se retrouve au sommet $S_{i,j}$, il s'agit de choisir l'arc de plus forte probabilité partant de ce sommet.

Dans un premier temps on écrit une fonction `initialiserGlouton(Obs:[[int]], E:[[float]], K:int)->int` qui permet d'initialiser l'algorithme glouton en trouvant le sommet le plus probable parmi $S_{i,0}$ pour i variant entre 0 et $K-1$. Pour cela il faut regarder la colonne `Obs[0]` de `E` et relever l'indice de la plus grande valeur.

```
1 def initialiserGlouton(Obs, E, K):
2     probasInitiales = [E[Obs[0][i]] for i in range(K)]
3     s, symbole = maximumListe(probasInitiales)
4     return symbole
```

❏ **Q19** – Proposer une fonction `glouton(Obs:[int],P:[[float]],E:[[float]],K:int,N:int)->[int]` qui renvoie la liste d'états obtenue par l'approche gloutonne. Même si cela n'est pas nécessaire, K, N seront des arguments de cette fonction.

❏ **Q21** – Indiquer le chemin renvoyé par l'algorithme glouton appliqué à la Figure 4. Conclure quant à l'optimalité de l'approche.

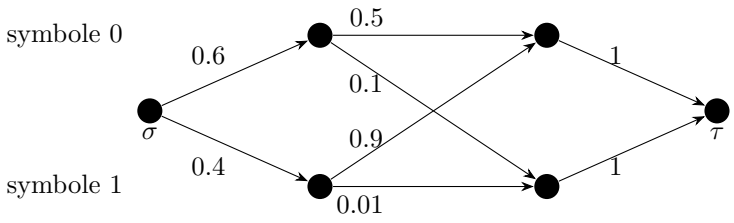


FIGURE 4 – Que donne l'algorithme glouton ici ?

❏ **Q22** – Expliquer en quoi rechercher un chemin de probabilité maximale pourrait se transformer en un problème de recherche de plus court chemin dans un graphe pondéré à poids positifs. Préciser alors quel algorithme pourrait être utilisé.

La recherche d'un chemin se fait en réalité selon l'algorithme de Viterbi. Il utilise une méthode de programmation dynamique, concept que nous étudierons dans un prochain chapitre.

Base de données photographique

variations sur un sujet du concours Centrale-Supélec

On étudie une base de données destinée à faciliter la gestion d'une banque de photographies numériques prises par les membres d'un groupe d'amis. Parmi les personnes de ce groupe, certaines sont photographes, mais d'autres se contentent de figurer sur les images en faisant un beau sourire. On établit une liste de mots-clés susceptibles de décrire le contexte dans lequel chaque photo a été prise : mer, montagne, sport, mariage, etc. Une même photo peut être décrite par plusieurs mots-clés, par exemple « mer » et « sport » pour une photo de planche à voile. Dans cette base de données, on crée tout d'abord une table `Motcle`,une table `Photo` et une table `Personne`.

MotCle	
MC_id	integer
MC_texte	string

Photo	
PH_id	integer
PH_date	string
PH_larg	integer
PH_haut	integer
PH_auteur	integer
PH_fichier	string

Personne	
PE_id	integer
PE_sexe	string
PE_prenom	string

— Table `Personne` : `PE_id` est l'identifiant d'une personne, `PE_sexe` vaut `'M'` ou `'F'` et `PE_prenom` est son prénom.

— Table `Photo` : `PH_id` est l'identifiant de la photo, `PH_date` la date de prise de vue (chaîne de caractère de la forme `'AAAA-MM-JJ'`, `PH_larg` et `PH_haut` sa largeur et sa hauteur en pixels, `PH_auteur` l'identifiant de son auteur (que l'on peut ainsi retrouver dans la table `Personne`), `PH_fichier` le nom du fichier qui la contient.

— Table `MotCle` : `MC_id` est un identifiant et `MC_texte` un mot-clé en langue française.

1. Citer une clé primaire pour chacune des trois tables. Dans la table `Photo`, citer un clé étrangère et indiquer à quelle table elle fait référence.

2. Écrire une requête renvoyant le prénom de toutes les personnes de sexe féminin.

3. Écrire une requête donnant les identifiants de toutes les photographies dont le rapport hauteur sur largeur vaut exactement 4/3.

4. Écrire une requête comptant le nombre de prénoms portés par des personnes de sexe masculin. Si plusieurs personnes portent le même prénom, on compte ce prénom une seule fois.

5. Écrire une requête renvoyant la date de la photo la plus ancienne.

Pour établir un lien entre les mots-clés, les photos et les personnes, on se place dans le cadre du modèle entité-association (MEA). On considère les associations suviantes :

— l'assocation *décrire* entre la table `MotCle` et la table `Photo` ; elle est définie par la proposition « un mot-clé décrit une photo », par exemple le mot-clé « montagne » décrit la photo d'identifiant 143 ;

— l'association *être de l'auteur de*, définie par la proposition « une personne est l'auteur d'une photographie » ;

— l'association *être présent* définie par la proposition « une personne est présente sur une photographie ».

6. Quelle est la classe fonctionnelle (un-à-un, un-à-plusieurs ou plusieurs-à-plusieurs) de chacune de ces trois associations ?

7. La base de données contient 5 tables, c'est à dire deux tables en plus de celles déjà présentées plus haut, et son modèle physique est représenté sur la figure (1). Expliquer, dans le cadre du MEA, le rôle des tables `Decrit` et `Present`. Pourquoi l'association *être l'auteur de* ne donne-t-elle pas lieu à l'introduction d'une table spécifique ? On veillera à apporter une réponse précise utilisant à bon escient le vocabulaire prévu par le programme officiel.

8. Écrire une requête qui compte le nombre de photos prises par « Alice » ou « Bernard ».

9. Écrire une requête qui renvoie la liste, par ordre alphabétique, de tous les noms de fichiers des photos prises par « Alain ».

10. Écrire une requête qui fournit l'identifiant et la date des photographies prises avant 2016 et associées au mot-clé « surf ».

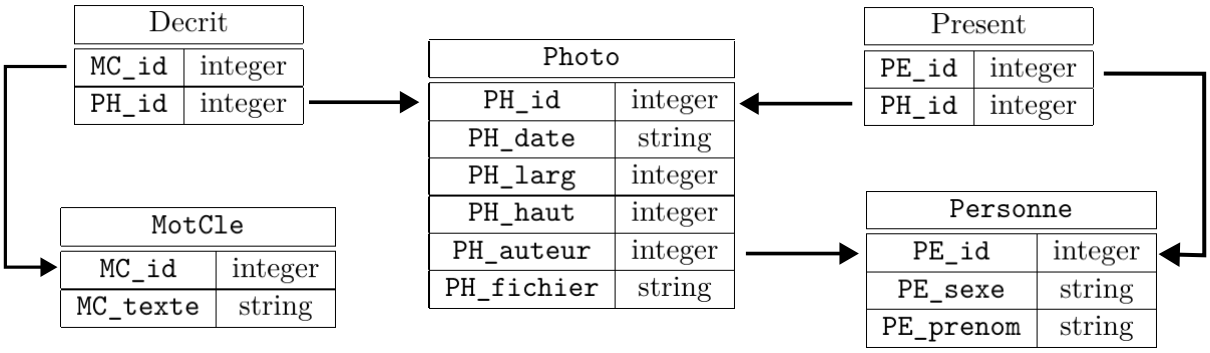


FIGURE 1 – Modèle de la base de données

11. Écrire une requête qui compte le nombre de photos prises par des femmes en 2020 et décrites par le mot-clé « voiture ».

12. Écrire une requête qui renvoie la largeur moyenne des photos décrites par le mot-clé « mer ».

13. Écrire une requête qui donne le prénom de l'auteur et l'identifiant de tous les selfies, c'est-à-dire les photographies sur lesquelles l'auteur est présent.

14. Écrire une requête qui sélectionne toutes les photographies sur lesquelles sont présents « Alice » et «Bernard», à l'exclusion de toute autre personne.

15. Écrire une requête qui affiche, à côté du prénom de chaque photographe, le nombre de photographies qu'il a prises, en triant les résultats par nombres décroissants de photographies.

16. Écrire une requête qui affiche, à côté de chaque mot-clé, la largeur moyenne des photos qu'il décrit, en se limitant à ceux qui décrivent au moins 10 photos.

17. Pour chaque photographe, compter le nombre de personnes distinctes qu'il a photographiées. On écrira ce nombre à côté du prénom du photographe.

18. On décide de dater l'entrée de chaque personne dans le groupe d'amis par la date de la photo la plus ancienne sur laquelle elle figure. Écrire une requête qui dresse le palmarès des prénoms des trois personnes les plus anciennes, avec leur date d'entrée.

19. Écrire une requête qui affiche toutes les paires de personnes ayant pris une photo le même jour.