Numérique et sciences informatiques

Lycée Hoche

année scolaire 2025-2026

Contents

1 Présentation du problème sur un exemple

Nous disposons d'une clé USB qui est déjà bien remplie et sur laquelle il ne reste que 5 GO d'espace libre.Nous souhaitons copier sur cette clé des fichiers vidéos pour l'emporter en voyage.Chaque fichier a une taille et chaque vidéo a une durée.La durée n'est pas proportionnelle à la taille car les fichiers sont de formats différents, certaines vidéos sont de grande qualité, d'autres sont très compressées.
Le tableau qui suit présente les fichiers disponibles avec les durées données en minutes.

Fichier	Durée	Taille
Vidéo 1	114	4,57 GO
Vidéo 2	32	630 MO
Vidéo 3	20	1,65 GO
Vidéo 4	4	85 MO
Vidéo 5	18	2,15 GO
Vidéo 6	80	2,71 GO
Vidéo 7	5	320 MO

1.1 Représentation des données

Ecrire une liste de 3-uplets nommée *videos* contenant l'ensemble des fichiers.

2 Méthode de la force brute:mise en oeuvre d'un algorithme

Le principe est simple.Il faut tester tous les cas possibles. Une méthode est d'associer le chiffre 1 à un fichier s'il est choisi et le chiffre 0 sinon.
Nous obtenons ainsi un nombre entier écrit en binaire avec sept chiffres.Le nombre 1001100 signifie que nous avons choisi les fichiers 1 ,4 et 5.Le nombre 1111111 signifie qu'on a choisi tous les fichiers.
A chaque nombre correspond exactement une possibilité pour construire une partie de l'ensemble des 7 chiffres.

Quel est le nombre total de possibilités?.....

2.1 Ensemble des parties

Nous souhaitons écrire une fonction qui prend en paramètre un ensemble , (dans notre problème un ensemble de 7 fichiers) , et renvoie l'ensemble des parties qui peuvent être constituées.

On considérera ici l'ensemble des 3-uplets des vidéos ci-dessus sous la forme d'une liste:
E = [('vidéo1', 114 , 4.57),('vidéo2', 32, 0.63),('vidéo3', 20, 1.65),('vidéo4', 4, 0.085),
('vidéo5', 18, 2.15),('vidéo6', 80, 2.71),('vidéo7', 5, 0.32)]

Nous avons besoin de l'écriture d'un nombre en binaire.Soit nous programmons une fonction , soit nous utilisons la fonction bin de Python.

1. Ecrire une telle fonction nommé *int_to_bin* dont la spécification est donnée ci-dessous.

```
def int_to_bin(n,nb):
    '''
    Entrées :n et nb sont de type int
    n est le nombre à convertir en binaire
    nb est le nombre de bits utilisés

    CU:l'algorithme utlisé est celui de la division euclidienne

    Sortie : la fonction retourne une chaîne
    de caractères contenant la représentation
    binaire de n
    '''
    ch=""
    #lignes à compléter
    while ..... :
        r = ............
        n = ............
        ch = str(r) +...
    #On complète éventuellement avec des zéros à gauche
    #pour former une chaîne de nb caractères
    ch = ..............................
    return ch
```

2. Nous pouvons alors écrire une fonction *ens_des_parties* qui prend en paramètre un ensemble d'objets et envoie une liste dont chaque élément est une partie de l'ensemble.

```
def ens_des parties(ensemble):
    """
    Entree :ensemble est une liste de p-uplets
    Sortie:.....................#à compléter
    """
    #nombre d'éléments
    nb = ..........
    #nombre de parties
    n = ...........
    #ensemble des parties
    parties = [ ]
    #écrire deux boucles imbriquées for pour former une partie
    #puis l'adjoindre à l'ensembles des parties "parties"
    for i in range (1,n):
        ch = int_to_bin(.....) # écriture de i sur nb bits
        partie= ....
        for j in range(len(ch)):
            if ch[j]== "1":
                partie.append(......)
        parties.append(partie)

    return ..........
```

2.2 Fonctions annexes

Pour chaque partie , nous devons calculer la durée totale et la taille des fichiers constituant la partie. Ecrire deux fonctions *duree_totale* et *taille_totale* qui réalisent cette tâche en complétant les corpus proposés ci-dessous.

```
def duree_totale(liste):
    d = 0
    ..........#à compléter
    ..........
    return d
```

```
def taille_totale(liste):
    t = 0
    ..........#à compléter
    ..........
    return t
```

2.3 Programme final

Pour le programme final, nous écrivons une fonction *recherche* qui prend en paramètres un ensemble de parties et la contrainte, et renvoie la partie correspondant au meilleur choix satisfaisant la contrainte après avoir parcouru toutes les parties.

1. Ecrire une telle fonction en complétant le corpus proposé ci-dessous

```
def recherche(ens_parties,contrainte):
    duree_max=0
    solution= [ ]
    for partie in ..............:
        duree =...................
        taille = .................
        if taille <=.......and  ............:
            duree_max = ......
            solution = .......

    return solution, duree_max
```

2. Il reste à écrire une fonction nommée *force_brute* qui appelle les fonctions précédentes. Compléter alors le corpus proposé ci-dessous.

```
def force_brute(fichiers , taille_max):
    '''
    entrées: -fichiers est l'ensemble des vidéos
             -taille_max est la contrainte
    sortie : le tuple solution (vidéos choisies et durée optimale)
    '''
    parties =.........
    return ..........
```

3. On termine en affichant les résultats de la démarche de force brute.

```
choix = force_brute(videos,5)
#à compléter
duree_totale = ............
choix_fichiers  =..........
print(choix_fichiers , duree_totale)
```

Page suivante : Utilisation d'un algorithme glouton

3 Utilisation d'un algorithme glouton

Un fichier vidéo est représenté par une liste comme :

[('vidéo1', 114 , 4.57),('vidéo2', 32, 0.63),('vidéo3', 20, 1.65),('vidéo4', 4, 0.085),
('vidéo5', 18, 2.15),('vidéo6', 80, 2.71),('vidéo7', 5, 0.32)]

Nous avons besoin de trois fonctions prenant en paramètre un tel fichier et renvoyant soit la durée, soit l'inverse de la taille (si la taille est minimale, son inverse est maximale), soit le rapport durée/taille.

```
def duree(fichier):
    return fichier[1]
def taille(fichier):
    return 1 / fichier[2]
def rapport(fichier):
    return fichier[1] / fichier[2]
```

Fonction sorted

Nous définissons une fonction *glouton* qui prend en paramètres une liste de fichiers , une taille maximale(celle que peut stocker la clé USB) et le type de choix utilisé (par durée , par taille, ou par duree/taille).

```
def sacaDos_glouton(liste,taille_max,choix):
    """
    Entrées:
    liste : une liste de fichiers
    taille_max : capacité maximale de stockage de la clé usb
    choix : le type de choix utilisé
    (par durée , par taille, ou par duree/taille).

    Sortie :
    reponse : la liste des fichiers solutions
    totale_duree : la duree totale maximale obtenue
    """

    #On construit une nouvelle liste en triant la liste passée en
    #paramètre suivant le type de choix utilisé
    #avec la fonction  sorted

    triee = sorted(liste, key = choix , reverse=True)
    #Initialisation de variables
    reponse = [ ]
    totale_duree = 0
    taille =0

    #La liste triée est parcourue et les noms des fichiers
    # sont ajoutés un par un dans la variable reponse,
    # tant que la #taille totale ne dépasse pas la taille maximale

     i=0
     while ..........and..........:
         nom,d,t = triee[ i ]
         if taille + t <= ........:
              reponse.append(....)
              taille = ..........
              total_duree = ......
         i=......
     return reponse , totale_duree
```

Nous exécutons la fonction *glouton* en précisant le type de choix utilisé.

»>print(sacaDos_glouton(videos,5,duree))

»>print(sacaDos_glouton(videos,5,taille))

»>print(sacaDos_glouton(videos,5,rapport))