Informatique Tronc Commun
Les indispensables

I Représentation des nombres

Q1. Quels entiers peut-on coder sur 1 octet ?

Q2. Un pixel est codé sur trois niveaux de couleur (RGB), chacun représenté par un entier compris entre 0 et 255. Quelle est la mémoire nécessaire pour stocker une image de 1024 pixels par 768 ?

II Listes (++++)

On considère une liste `L` de flottants.

Q3. Écrire une fonction `somme(L:list)->float` qui prend en entrée la liste `L` et qui renvoie la somme des éléments de `L`.

Q4. Écrire une fonction `moyenne(L:list)->float` qui prend en entrée la liste `L` et qui renvoie la moyenne des éléments de `L`.

Q5. Écrire une fonction `variance(L:list)->float` qui prend en entrée la liste `L` et qui renvoie la variance des éléments de `L`.

Q6. Écrire une fonction `max(L:list)->float` qui prend en entrée la liste `L` et qui renvoie le maximum des éléments de `L`.

Q7. Écrire une fonction `rang_min(L:list)->int` qui prend en entrée la liste `L` et qui renvoie le rang du minimum des éléments de `L`.

Q8. La liste de listes de deux éléments, `L`, est du type `[[ch1,a1],[ch2,a2],...]`, où les premiers éléments de chaque liste `ch1`, `ch2`, sont des chaines de caractères et les deuxièmes éléments `a1`, `a2`, de chaque sous liste est un flottant.

(a) Écrire une fonction `somme2(L:list[list])->float` qui prend en entrée la liste `L` et qui renvoie la somme des valeurs `ai`.

(b) Écrire une fonction `moyenne2(L:list[list])->float` qui prend en entrée la liste `L` et qui renvoie la moyenne des valeurs `ai`.

(c) Écrire une fonction `max2(L:list[list])->float` qui prend en entrée la liste `L` et qui renvoie la valeur maximale des `ai`.

(d) Écrire une fonction `min(L:list[list])->str` qui prend en entrée la liste `L` et qui renvoie la chaine `chi` de valeur `ai` minimale.

Q9. Écrire une fonction booléenne `recherche(L:list,e:int)->bool` qui prend en entrée une liste `L` d'entiers et un entier `e` si l'élément et qui renvoie si un élément ou non dans la liste. Quelle est la complexité de cette fonction ?

III Les dictionnaires

On considère un dictionnaire `D`, les clés sont des chaines de caractères, et les valeurs des entiers.

Q10. Écrire une fonction `max_dico(D:dict)->float` qui prend en entrée le dictionnaire `D` et qui renvoie la valeur maximale.

Q11. Écrire une fonction `cle_min_dico(D:dict)->int` qui prend en entrée le dictionnaire `D` et qui renvoie la clé de valeur minimale.

Q12. Écrire une fonction `moy_dico(D:dict)->float` qui prend en entrée le dictionnaire `D` et qui renvoie la moyenne des valeurs.

Q13. Écrire une fonction `valeur(D:dict,e:int)->list[str]` qui prend en entrée le dictionnaire `D` et un entier `e` et qui renvoie la liste des clés de valeurs `e`.

Q14. Écrire une fonction `occurrences(texte:str)->dict` qui prend en argument une chaîne de caractère `texte` et renvoie un dictionnaire dont les clés sont les lettres qui apparaissent dans le texte et les valeurs le nombre d'occurrences de ces lettres.

Par exemple : `occurrences('ACCTAGCCCTA')` renverra `{'A':3,'C':5,'T':2,'G':1}`.

IV Tris

On souhaite trier une liste, par ordre croissant.

À adapter pour trier une liste par ordre décroissant, ou une liste de liste selon l'un des éléments des sous listes...

On donne ici quelques exemples de tris classiques. Aucun n'est rigoureusement au programme, tout en l'étant tous... Les algorithmes ne sont pas à connaître mais il faudrait pouvoir les écrire une fois l'algorithme rappelé.

IV.1 Tri fusion

Le tri fusion est un **tri récursif** basé sur le principe de « diviser pour régner » : un problème initial sur n données est divisé en deux sous-problèmes portant sur $\frac{n}{2}$ données si possible.

Ici, il s'agit de passer du tri d'une liste de n éléments aux tris de deux listes contenant moitié moins d'éléments. On trie alors la première moitié de la liste, la seconde moitié de la liste, et on fusionne ces deux listes triées en une seule liste triée en utilisant une fonction annexe.

Q15. Écrire une fonction `fusion(L1:list,L2:list)->list` qui prend en entrée deux listes triées dans le même ordre et renvoie une liste triée résultant de la fusion des deux listes.

Q16. Écrire la fonction récursive `tri_fusion(L:list)->list` qui renvoie la liste triée en utilisant l'algorithme du tri fusion.

IV.2 Tri rapide

Le tri rapide est un tri récursif dans lequel on divise un problème initial en deux sous-problèmes.

On choisit un élément, que l'on notera `p`, appelé **pivot** (qui peut être, le premier élément de la liste, le dernier, ...), de l'enlever de la liste et de partitionner la liste en deux sous-tableaux : une liste contenant les éléments strictement inférieurs à `p` et une liste contenant les éléments strictement supérieurs à `p`. On trie récursivement chacune des deux listes et on rassemble tout.

Q17. Écrire la fonction `tri_rapide(L:list)->list` qui renvoie la liste `L` triée.

IV.3 Tri bulles

On considère une liste `L` à n éléments. Le tri à bulles consiste à faire remonter les éléments les plus grands en permutant successivement les éléments du tableau : on parcourt le tableau, et à chaque fois que l'élément de gauche est strictement supérieur à l'élément de droite, on les permute. À la fin de ce parcours, le plus grand élément du tableau est en dernière position. On recommence le parcours du tableau pour trier les $n-1$ éléments, puis les $n-2$ éléments, etc. Le nom « tri à bulles » vient du fait que les éléments les plus grands remontent plus vite, comme les bulles dans l'eau.

Q18. Écrire une fonction itérative `tri_bulles(L:list)->list` qui trie une liste selon l'algorithme du tri à bulles.

Q19. Écrire une fonction récursive `tri_bulles_rec(L:list)->list` qui trie une liste selon l'algorithme du tri à bulles.

V Dichotomie

Q20. Comment doit-être la liste pour utiliser l'algorithme de dichotomie pour déterminer si un entier est présent dans une liste d'entier ?

Q21. Rappeler le principe de la dichotomie pour déterminer si un entier est présent dans une liste d'entier.

Q22. Écrire une fonction booléenne qui prend en entrée une liste `L` d'entiers et un entier `e` si l'élément et qui renvoie si un élément ou non dans la liste par dichotomie.

Q23. Quelle est la complexité de cette fonction ?

VI Algorithme glouton

Les **algorithmes gloutons** suivent une stratégie simple : **lorsqu'à chaque étape, un choix doit être fait, c'est le choix optimal à ce moment qui est fait.**

Un problème d'optimisation a deux caractéristiques : une fonction que l'on doit maximiser ou minimiser et une série de contraintes auxquelles il faut satisfaire. On peut essayer de résoudre ce type de problème en écrivant un algorithme qui énumère toutes les possibilités afin de trouver la meilleure. C'est un algorithme très simple mais souvent inutilisable à cause du coût. L'objectif d'un algorithme glouton est d'obtenir une solution rapidement, mais qui ne sera pas toujours la solution optimale.

À chaque étape exécutée par un algorithme, se présente un ensemble de choix et un algorithme glouton fait le meilleur choix parmi les propositions. **Un choix glouton est donc un choix localement optimal. La question est de savoir si en faisant une série de choix localement optimaux, on finit par aboutir à une solution optimale. C'est parfois le cas mais pas toujours.**

Prenons l'exemple classique du rendu de monnaie. On considère le système de monnaie de la zone euro, et nous avons les pièces suivantes (en centimes) : $S = (1, 2, 5, 10, 20, 50, 100, 200)$.

Le problème du rendu de monnaie consiste à trouver le nombre de pièces de chaque type à rendre pour arriver à la somme à rendre, en minimisant le nombre de pièces utilisées.

La contrainte est le fait que le montant total des pièces rendues est égal à la somme à rendre.

On cherche, par valeur décroissante en partant de la pièce qui a la plus forte valeur, la première pièce qui a une valeur inférieure ou égale à la somme à rendre r. On prend cette pièce, on retranche sa valeur v à r. On recommence en partant de la pièce prise en cherchant cella qui a une valeur inférieure ou égale à la nouvelle somme à rendre $r - v$. On prend cette pièce, et ainsi de suite jusqu'à arriver à une somme à rendre nulle.

Q24. Mettre en œuvre l'algorithme à la main pour rendre 8 centimes, 7 €, 12 €.

Q25. Compléter la fonction `monnaie(p:list,r:int)->list` en Python.

```
def monnaie(p,r):
    """
    Arguments :
    p : liste des pièces du système de monnaie
    r : somme à rendre
    Retour :
    sol : liste du nombre de pièces à rendre pour chaque pièce du système de monnaie
    """
    assert type(p)==list and type(r)==int
    assert r>=0
    n=len(p) # nbre de pièces
    i=........ # on part de la pièce la plus grosse
    sol=........ # initialisation du nbre de pièces dans chaque type
    while ........ : # tant que la somme à rendre n'est pas nulle
        while ........ : # tant que la pièce testée est plus grande que le montant à rendre, on passe à la suivante (plus petite)
            i=........
        sol[i] = ................ # on ajoute une pièce i à rendre
        r=................ # on enlève le montant de la pièce ajoutée à la somme à rendre
    return sol
```

On considère les valeurs des pièces et billets en euros `p=[1,2,5,10,20,50,100,200,500]`.

On représente par une liste `C` le nombre de chaque pièces/billets qu'un commerçant dispose dans sa caisse (par exemple : `[32,37,17,12,13,5,1,0,0]` signifie qu'il a 32 pièces de 1 €, 37 pièces de 2 €, etc).

Q26. Écrire une fonction `renducaisse(p:list,C:list,r:int)->list` qui détermine la façon de rendre la somme `r`, avec ce qu'il y a de disponible dans la caisse. On renverra une liste avec la quantité de chaque pièce/billet qu'il devra rendre.

VII Algorithmes fondamentaux indispensables en 2e année

VII.1 Programmation dynamique

Q27. Écrire une fonction `mat_nulle(n:int,m:int)->list[list]` qui crée un tableau de `n` lignes et de `m` colonnes remplies de 0.

Exemple simple : calcul du coefficient binomial avec $\binom{n}{k} = \begin{cases} 1 & \text{si } k = n \text{ ou } k = 0 \\ n & \text{si } k = 1 \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{sinon} \end{cases}$

Q28. En utilisant la mémoïsation, écrire la fonction `binome_mem1(k:int,n:int,d={}:dict)->int` qui renvoie la valeur de $\binom{n}{k}$.

Q29. En utilisant la mémoïsation, écrire la fonction `binome_mem2(k:int,n:int)->int` qui renvoie la valeur de $\binom{n}{k}$.

On définira une sous-fonction `cb(i:int,j:int)->int` au sein de `binome_mem2` qui calcule $\binom{j}{i}$ et complète un dictionnaire `d` dont les clés sont les 2-uplets `(i,j)` et la valeur associée $\binom{j}{i}$. Ce dictionnaire `d` est une variable locale pour `binome_mem2`, globale pour `cb`.

Q30. Écrire une fonction `binome_asc(k:int,n:int)->list[list]` qui calcule le coefficient binomial en utilisant l'approche ascendante. Pour cela, on remplira un tableau `tab` (de `k+1` colonnes et de `n+1` lignes) après l'avoir initialisé avec des 0. La case ligne `j` et colonne `i` contient $\binom{j}{i}$.

VII.2 IA

Q31. Écrire la fonction `distance(X:list,Y:list)->float` qui renvoie la distance entre les deux listes `X` et `Y`.

Q32. Écrire la fonction `Distances(X:list,Z:list[list])->list[float]` qui renvoie la liste des distances entre `X` et chaque liste de la liste `Z`. On proposera deux versions : la première en utilisant la fonction précédente, la deuxième sans l'utiliser.

Q33. Écrire la fonction `barycentre(L:list[list])->list` qui prend en entrée une liste de listes de coordonnées, et renvoie les coordonnées du barycentre de ces points sous la forme d'une liste.

VII.3 Théorie des jeux

Q34. Écrire la fonction`transpose(G:dict)->dict` qui prend en entrée un graphe `G` représenté sous la forme d'un dictionnaire des successeurs et renvoie le dictionnaire des précédesseurs.

Q35. Écrire la fonction `degres_sortants(G:dict)->dict` qui prend en entrée le graphe `G` représenté sous la forme d'un dictionnaire des successeurs et renvoie un dictionnaire des degrés sortants des sommets de `G`.

Q36. Écrire une fonction `predecesseurs(G:dict,j:int)->list` qui renvoie la liste de tous les prédécesseurs du sommet `j` dans le graphe `G` représenté sous la forme d'un dictionnaire des successeurs.