TD 1

1. Algorithmes de recherche

1. Écrire une fonction `present(n:int,L:list)->bool` qui prend en argument un entier `n` et une liste d'entiers `L` et qui renvoie `True` ou `False` selon que `n` appartient à `L` ou non. Cette fonction devra avoir une complexité constante dans le meilleur des cas (donc qui ne dépend pas de la taille de la liste `L`).
2. Un tableau `T` à deux dimensions (n lignes et p colonnes) est représenté par une liste de listes : `T[i]` est la liste des éléments de la $i^{\text{ème}}$ ligne et `T[i][j]` est donc l'élément de la $i^{\text{ème}}$ ligne et $j^{\text{ème}}$ colonne.
   Écrire une fonction `present2D(n:int,T:list)->bool` similaire à celle de la question précédente, qui teste si l'entier `n` appartient au tableau d'entiers `T`.

2. Autour du maximum

1. Écrire une fonction `Max(L:list)->float` qui renvoie la valeur du plus grand élément de la liste de réels `L`, supposée non vide.
2. Écrire une fonction `ToutSurMax(L:list)->list` qui renvoie une liste `[M,nb,pos]`, où `M` est le plus grand élément de `L`, `nb` son nombre d'occurrences et `pos` la liste de ses positions (les indices). Cette fonction ne devra faire qu'un seul parcours de la liste `L`
3. Écrire une fonction `MaxDico(d:dic)->str` qui prend en argument un dictionnaire non vide dont les valeurs sont des réels et qui renvoie la clé, supposée être une chaine de caractères, correspondant à la plus grande des valeurs de `d`.

3. Utilisations de dictionnaires

1. On donne à des élèves un QCM dont les bonnes réponses sont stockées dans un dictionnaire `reponses` de la forme `{'Q1':'a','Q2:'c',...}` (ce qui signifie que la bonne réponse à la première question est la réponse a, celle à la deuxième c,...). La copie d'un élève est stockée elle aussi dans un dictionnaire `eleve` de la forme `{'Q1':'a','Q3:'d',...}` (ce qui signifie qu'il a répondu a à la question 1, d à la 3,... ; s'il n'a pas répondu à une question, elle ne figure pas dans ce dictionnaire). Le barème est le suivant : 2 points pour une bonne réponse, -1 pour une réponse fausse et 0 en cas de non réponse. Écrire une fonction `note(eleve:dic,reponses:dic)->int` qui prend en argument deux dictionnaires de la forme précédente et renvoie la note de l'élève.
2. Écrire une fonction `union(d1:dic,d2:dic)->dic` qui prend en argument deux dictionnaires et qui renvoie un autre dictionnaire `d` qui réunit `d1` et `d2` : si une clé correspond à des valeurs différentes dans `d1` et `d2`, la valeur correspondante dans `d` sera un tuple avec les deux valeurs. Si par exemple on a `d1={'un':1,'deux':2,'trois':3}` et `d2={'un':1,'deux':[2],'quatre':4}`, la fonction devra renvoyer `{'un': 1, 'deux': (2, [2]), 'trois': 3, 'quatre': 4}`

Pour une chaîne de caractères `ch`, on appelle mot de longueur k, toute chaîne de k caractères consécutifs extraite de `ch` : les mots de longueur 4 de `'concours'` sont `'conc'`, `'onco'`, `'ncou'`, `'cour'` et `'ours'`.

3. Écrire une fonction `mot(k:int,ch:str)->dic` qui prend en argument un entier `k` non nul et une chaîne de caractère et qui renvoie un dictionnaire dont les clefs sont les mots de longueur k issus de `ch` et les valeurs leur nombre d'occurrences : `mot(3,'abracadabra')` devra par exemple renvoyer `{'abr': 2, 'bra': 1, 'rac': 1, 'aca': 1, 'cad': 1, 'ada': 1, 'dab': 1}`.
4. En déduire une fonction `MoinsFrequentMot(k:int,ch:str)->str` qui renvoie un mot de longueur k qui apparaît le moins souvent, mais au moins une fois, dans `ch` (si plusieurs mots sont possibles, on en renverra un seul).
   Évaluer la complexité de cette fonction en fonction de N, la longueur de la chaîne `ch`.
5. Une chaîne de caractères est l'anagramme d'une autre si elles sont constituées des mêmes caractères dans un ordre différent. Écrire une fonction `anagramme(ch1:str,ch2:str)->bool` qui prend en argument deux chaînes de caractères non vides et qui renvoie `True` ou `False` selon qu'elles sont anagramme l'une de l'autre ou non.