DS 5

Informatique MP2I

Julien Reichert

Toutes les questions de programmation sont à résoudre dans le langage précisé le cas échéant.

Questions de cours

Question de cours 1 : Donner la complexité dans le meilleur des cas (rappel : ne pas utiliser $\mathcal{O}$ si ce n'est pas le pire des cas, on donnera un équivalent ou une borne inférieure) de l'algorithme naïf de recherche de motifs et une instance où cette complexité est atteinte. Idem pour le pire des cas. Il faudra également traiter la question dans deux cas : si le problème consiste à déterminer la présence d'un motif et si le problème consiste à compter le nombre d'occurrences.

Question de cours 2 : Mêmes questions pour l'algorithme de Boyer-Moore.

Question de cours 3 : Mêmes questions pour l'algorithme de Rabin-Karp.

Question de cours 4 : Montrer que l'arbre obtenu par l'algorithme de Huffman minimise la somme sur tous les caractères des produits du nombre d'apparitions de ce caractère et du nombre de bits utilisés pour l'écrire, parmi tous les codes préfixes possibles.

Question de cours 5 : Écrire dans un langage au choix une implémentation de l'algorithme de Lempel-Ziv-Welch (limité à la compression et renvoyant simplement une séquence de codes entiers).

Question de cours 6 : Soit M^n la puissance usuelle n-ième de la matrice d'adjacence M d'un graphe (S, A). Montrer que la cellule à la ligne i et la colonne j de M^n contient le nombre de chemins distincts de longueur n du i-ième sommet du graphe au j-ième sommet du graphe.

Question de cours 7 : Montrer que dans un graphe non orienté non vide, n'importe laquelle des trois propriétés suivantes est impliquée par la conjonction des deux autres : (i) le graphe n'admet pas de cycle non trivial, (ii) le graphe est connexe, (iii) le graphe a une arête de moins que le nombre de sommets.

Question de cours 8 : Montrer que tout sous-chemin d'un chemin optimal en longueur est optimal en longueur, mais raccorder deux chemins optimaux en longueur ne donne pas forcément un chemin optimal en longueur. Prouver la proposition analogue sur des chemins optimaux en valeur dans un graphe pondéré.

Question de cours 9 : On donne ci-après une implémentation de l'algorithme de Dijkstra en OCaml. En prouver la terminaison et l'adapter pour que la fonction renvoie un chemin de l'origine à une destination qui est un nouvel argument, chemin qui sera une liste de sommets contenant les deux extrémités susnommées.

```
type 'a tas = Vide | Noeud of 'a tas * 'a * 'a tas;;

let creer_fp () = Vide;;

let est_fp_vide fp = fp = creer_fp ();;

(* Attention, la structure est persistante ! *)
let rec ajouter_fp dist fp s = match fp with
| Vide -> Noeud(Vide, s, Vide)
| Noeud(g, t, d) ->
    let (nouvrac, nouvajout) = (if dist.(s) < dist.(t) then (s, t) else (t, s)) in
    Noeud(d, nouvrac, ajouter_fp dist g nouvajout);;
```

```
(* Renvoie la racine et la file sans celle-ci. *)
let rec extraire_fp dist fp = match fp with
| Vide -> failwith "File de priorité vide"
| Noeud(g, s, Vide) -> s, g
| Noeud(Vide, s, d) -> s, d
| Noeud((Noeud(gg, sg, gd) as g), s, (Noeud(dg, sd, dd) as d)) ->
(* Utilisation exceptionnelle de la syntaxe as qui simplifie l'écriture *)
    s, (if dist.(sg) < dist.(sd)
    then Noeud(snd (extraire_fp dist g), sg, d)
    else Noeud(g, sd, snd (extraire_fp dist d)));;

(* Autre possibilité : ajouter systématiquement, quitte à avoir des doublons,
et tester au moment de retirer un sommet. *)
let rec remonter_fp dist fp t = match fp with
(* attention, coût linéaire ici, une optimisation reviendrait à localiser l'élément *)
| Vide -> Vide (* Tous les cas où le sommet n'est pas trouvé dans la branche... *)
| Noeud(g, s, d) when s = t -> fp
| Noeud(Noeud(gg, sg, gd), s, d) when sg = t ->
    if dist.(t) < dist.(s) then Noeud(Noeud(gg, s, gd), t, d) else fp
| Noeud(g, s, Noeud(dg, sd, dd)) when sd = t ->
    if dist.(t) < dist.(s) then Noeud(g, t, Noeud(dg, s, dd)) else fp
| Noeud(g, s, d) ->
    let gbis = remonter_fp dist g t and dbis = remonter_fp dist d t in
    begin
      match gbis, dbis with
      | Vide, Vide -> fp
      | Noeud(gg, sg, gd), _ when sg = t && dist.(t) < dist.(s) ->
        Noeud(Noeud(gg, s, gd), t, d)
(* pas besoin de dbis, d ne peut pas être modifié alors *)
      | _, Noeud(dg, sd, dd) when sd = t && dist.(t) < dist.(s) ->
        Noeud(g, t, Noeud(dg, s, dd))
      | _, _ -> Noeud(gbis, s, dbis)
    end;;

let dijkstra graphe origine =
  let n = Array.length graphe in
    let dist = Array.make n max_int in
      dist.(origine) <- 0;
      let rec whilerecursif ouverts = if not (est_fp_vide ouverts) then begin
        let s, ouverts2 = extraire_fp dist ouverts in
          let voisins = graphe.(s) in
            let fonction_a_folder fp (t, poids) =
              if dist.(s) + poids < dist.(t) then
                begin
                  let il_faut_ajouter = dist.(t) = max_int in
                  dist.(t) <- dist.(s) + poids;
                  if il_faut_ajouter then ajouter_fp dist fp t
                  else remonter_fp dist fp t
                end
              else fp
            in whilerecursif (List.fold_left fonction_a_folder ouverts2 voisins)
       end
       in whilerecursif (ajouter_fp dist (creer_fp ()) origine);
       dist;;
```

Problème - Harry Potter and the Philosophiæ Doctor

Le problème ci-après regroupe des exercices faisant principalement appel à des connaissances algorithmiques et se composant d'exercices totalement indépendants (sauf utilisation de numéros bis).

Première année - Harry Potter à l'école des Mines (de Nancy)

Harry Potter, ayant nouvellement découvert sa nature de sorcier, est amené par Hagrid au chemin de traverse. Il y découvre un système monétaire particulier, avec trois types de pièces : les noises, les mornilles (29 noises) et les gallions (17 mornilles). Promis, je n'invente rien !

Question P1.1 : En considérant une somme à payer, exprimée en noises (entier positif, seul argument), et en admettant que la quantité de chaque pièce à disposition n'est pas un problème, on cherche à minimiser le nombre de pièces à échanger de part et d'autre (donc paiement et rendu de monnaie) dans la transaction. Écrire une fonction en C qui résout le problème.

Les étudiants de première année de Poudlard sont répartis dans quatre maisons, et tout porte à croire qu'il y a autant d'étudiants dans chaque maison. En tout cas, cela donne une idée.

Question P1.2 : Écrire en OCaml une fonction prenant en argument une liste de taille un multiple de quatre (pas à vérifier) et retournant la liste de tous les quadruplets de listes de tailles identiques qui partitionnent la liste de départ en respectant l'ordre au sein de la liste de départ.

Par exemple, pour la liste `[0; 1; 2; 3; 4; 5; 6; 7]` (avec des nombres pour simplifier), la fonction retournera `[([0; 1], [2; 3], [4; 5], [6, 7]); ([0; 1], [2; 3], [4; 6], [5; 7]); ...]` (liste tronquée car de taille 2520).

Question P1.2bis : D'ailleurs, combien y a-t-il de telles partitions possibles en fonction de la taille de la liste, notée $4n$?
[Ceci est un exercice de mathématiques et la formule doit être justifiée !]

Deuxième année - Harry Potter et la chambre au CROUS

Un certain Tom aime bien jouer sur les mots et mélanger des lettres. Ceci étant, pour le côté dramatique, mieux vaut que les choses aillent à une certaine allure.

Question P2.1 : Écrire en OCaml une fonction prenant en argument deux chaînes de caractères et renvoyant la distance maximale (en nombre d'indices) entre deux caractères identiques, en considérant la manière d'apparier les caractères identiques qui minimise cette distance maximale. On supposera sans le vérifier que la deuxième chaîne est une anagramme de la première.

Par exemple, pour les chaînes `"EXPELLIARMUS"` et `"SUPERMAXILLE"`, la réponse est `11` (distance entre les `S`).

En vérité, les lettres étaient mises sur plusieurs lignes...

Question P2.1bis : Même exercice avec un tableau de chaînes de caractères, la distance étant calculée en tant que distance euclidienne, et les coordonnées étant l'indice de la chaîne où le caractère est pris et l'indice au sein de la chaîne. Les chaînes peuvent commencer par des espaces qui ne seront pas à traiter mais qui auront un effet sur les coordonnées des autres caractères de la chaîne où elles apparaissent, et certaines chaînes pourront être vides. **On pourra se contenter de décrire les adaptations à faire sur le programme précédent à condition que celui-ci soit effectivement écrit.**

Un nombre incalculable de mèmes font honneur à la tendance d'Albus Dumbledore à donner des points à Gryffondor à tort et à travers. Mais curieusement, c'est toujours juste assez pour passer devant...

Question P2.2 : Écrire en C une fonction qui prend en argument un entier noté `s`, un tableau d'entiers `t` et sa taille `n` et qui calcule la plus petite valeur strictement supérieure à `s` qui puisse s'obtenir en additionnant des éléments à des indices différents dans `t` (les éléments pouvant être égaux par ailleurs). La valeur de retour de la fonction devra permettre de reconstruire cette somme et on prendra soin d'expliquer comment.

Dans l'idée, si le tableau n'avait pas contenu que des entiers mais des couples (entier, chaîne), on voudrait bien pouvoir récupérer les chaînes de caractères associées, par exemple « 50 points pour avoir respiré », « 10 points pour les jolies chaussures »...

Troisième année - Harry Potter et le prisonnier de son immeuble qui manque le TD

L'offre de cours s'étoffe et l'emploi du temps ne permet pas de suivre tous les enseignements que l'on voudrait (on dirait les études supérieures). Fort heureusement, le don d'ubiquité est envisageable...

Question P3.1 : Écrire en OCaml une fonction qui prend en argument un tableau de cours (type défini ci-après) et qui détermine le nombre minimum d'endroits où il faut parfois pouvoir être à la fois pour suivre des cours à hauteur (au moins) d'un degré de satisfaction (qui est aussi en argument).

Le type est le suivant : `type cours = { debut : int ; fin : int ; satisfaction : int}`. On considère qu'il faut être à deux endroits à la fois lorsque l'on veut suivre deux cours dont les intervalles ouverts respectifs délimité par les débuts et fins correspondants ont une intersection non vide. Les degrés de satisfaction seront toujours positifs (ou nuls dans le cas de la divination). Début et fin sont des entiers, comme ils correspondent à un point dans le temps on peut considérer que les jours s'enchaînent (par exemple `32` = le mardi à huit heures du matin).

Quatrième année - Harry Potter et la coupe des établissements de Prologin où Poinca a brillé

Arrivé à Poudlard, Harry avait constaté que les escaliers étaient du genre à se déplacer, bloquant parfois le passage et modifiant les itinéraires. Le château n'en fait certes qu'à sa tête, mais c'est l'occasion de trouver un exercice qui se place dans un scénario où les mouvements peuvent être prédits, par exemple grâce à l'acquisition de la carte du maraudeur (ou dans un certain labyrinthe...).

Question P4.1 : Considérons un graphe pondéré dynamique : les sommets et les arcs sont toujours les mêmes, mais le poids de chaque arc (s, s') est décrit par une fonction $f_{(s,s')}$ prenant en argument le temps passé depuis le début de l'expérience (entier naturel) et renvoyant un entier strictement positif (temps mis pour aller de la source à la destination), **avec la propriété que pour tout arc (s, s') et pour tous t_1, t_2 tels que $t_1 < t_2$ on a $t_1 + f_{(s,s')}(t_1) \leq t_2 + f_{(s,s')}(t_2)$, autrement dit attendre avant de prendre un arc ne permet pas d'arriver strictement plus tôt**. Une adaptation de l'algorithme de Dijkstra permet alors de déterminer, à partir d'un sommet d'origine fixé, quel est le premier moment où chaque sommet peut être rejoint. Expliquer (éventuellement par des commentaires dans le code) cette adaptation et l'implémenter en OCaml. On pourra se servir de la version fournie dans la partie cours.

Cinquième année - Harry Potter et l'ordre structurel que presque personne ne sait définir

Lors de la première éviction de Dumbledore, une citation célèbre a été prononcée. La deuxième éviction est l'occasion de faire décoder un message (l'A. D. est prudente et communique de manière secrète).

Question P5.1 : Une chaîne de caractères a été compressée puis sérialisée par l'algorithme de Lempel-Ziv-Welch en utilisant des codes sur 9 bits réorganisés pour un découpage en octets. On donne ci-après le résultat en tant que suite de codes ASCII (pour éviter d'avoir à mettre la table ASCII en annexe, notamment). Quelle était la chaîne initiale ?

Le résultat en question : 44, 155, 206, 162, 3, 185, 164, 216, 108, 16, 24, 77, 135, 51, 120, 128, 204, 105, 55, 25, 4, 7, 67, 65, 132, 232, 32, 52, 25, 77, 135, 8, 28, 22, 15, 9, 59, 152, 79, 39, 49, 1, 136, 202, 32, 51, 154, 78, 198, 83, 116, 34, 44, 72, 55, 153, 228, 7, 35, 164, 140, 233, 12, 137, 155, 206, 114, 99, 185, 162, 24, 97, 57, 154, 225, 166, 243, 144, 128, 210, 116, 23, 0.

Sixième année - Harry Potter et le fils illégitime du gendre de Stanislas

[Précédemment intitulé « Harry Potter et le sous-titre parodique que je n'arrive pas à inventer »]

Le nouveau capitaine de l'équipe de Gryffondor de Quidditch a fort à faire cette année-là avec les départs de ses membres importants à la suite de l'obtention de leur diplôme (ou pas...). Même si certains postes semblent évidents à attribuer, des essais sont nécessaires.

Imaginons que les sélections sont terminées, mais que suite à divers événements (empoisonnement, retenues, etc.) des changements et réattributions soient nécessaires. Pour éviter une lourde défaite, il serait bon d'utiliser pour chaque poste des joueurs adéquats. C'est pour cela que chaque joueur va avoir une liste de préférences pour chaque poste sous la forme d'une note de zéro à dix (zéro étant l'inadéquation et dix l'adéquation). Il s'agit donc d'attribuer à chaque joueur un poste de sorte de maximiser la somme des scores d'adéquation.

Question P6.1 : Résoudre à la main l'instance proposée ci-après. Il n'y a pas unicité de la réponse optimale car « Batteur 1 » et « Batteur 2 » sont par exemple interchangeables. Il suffit de trouver un appariement qui maximise le score.

Joueur	Gardien	Poursuiveur 1	Poursuiveur 2	Poursuiveur 3	Batteur 1	Batteur 2	Attrapeur
Demelza	1	3	3	3	1	1	5
Ginny	4	5	5	5	2	2	8
Harry	5	3	3	3	2	2	10
Jimmy	2	0	0	0	4	4	1
Katie	2	3	3	3	3	3	4
Ritchie	2	0	0	0	5	5	3
Ron	6	3	3	3	0	0	3

On remarque que pour un poste où plusieurs joueurs sont attendus, il a suffi de dupliquer une colonne avec le même score d'adéquation pour chacun. C'est la même adaptation que celle qu'on ferait pour utiliser le problème des mariages stables et l'algorithme de Gale-Shapley pour une affectation dans une école où plusieurs places sont disponibles, par exemple (coucou Parcoursup !).

La résolution du problème des affectations se fait traditionnellement par ce qui est appelé « l'algorithme hongrois ». On peut considérer qu'il existe deux manières d'exécuter la méthode associée : à la main, il est plus aisé de manipuler la matrice des scores afin de faire apparaître un zéro par ligne et par colonne, mais s'il s'agit de programmer on passera plutôt par des graphes, avec la notion de chemins augmentants de l'algorithme d'Edmonds.

Ainsi donc, pour une résolution à la main, voici la recette de cuisine, extraite et adaptée d'un TP de première année de tronc commun de 2021/2022.

Puisqu'il s'agit de sélectionner n éléments, un par ligne et un par colonne, afin de maximiser la somme totale, on peut supposer sans perte de généralité que le maximum de chaque ligne est 0.

En pratique, il s'agit de retrancher dans chaque ligne (resp. colonne) la valeur du maximum de la ligne (resp. colonne) à chaque cellule, ce qui se fait sans perte de généralité car une répartition qui maximise la somme continue de la maximiser lorsque tous les joueurs voient le score associé à un poste diminué de la même constante, ainsi que lorsque pour un joueur particulier le score associé à chaque poste subit également la même modification.

Par la suite, si une répartition permet de sélectionner n zéros, c'est gagné. Sinon il s'avère qu'on peut couvrir tous les zéros en traçant moins de n lignes et colonnes.

La transformation à appliquer est la suivante : une fois ces zéros couverts, prendre le maximum des éléments n'appartenant à aucune ligne ni colonne tracée, le soustraire dans toutes les **lignes non tracées** et l'additionner sur toutes les **colonnes tracées**.

Le maximum de la matrice est alors zéro une fois de plus, certains zéros ayant changé de place, et on retente de sélectionner n zéros, répétant l'autre opération sinon, jusqu'à ce que l'algorithme termine (la terminaison est garantie, mais la preuve va au-delà des attendus du jour).

Septième année - Harry Potter et les exercices de la mort

Pas de spoiler, pas de mise en contexte. Ce sera juste aussi difficile que de détruire des horcruxes.

Question P7.1 : Écrire en OCaml une fonction qui prend en argument un entier `d` entre 1 et 31 (pas à vérifier) et un entier `j` entre 0 et 6 représentant un jour de la semaine (0 = dimanche, 1 = lundi, etc.) et qui retourne un couple `(a, m)` où `m` est un numéro de mois (1 = janvier, etc.) et `a` une année, de sorte que la prochaine fois où le `d` du mois est un jour représenté par `j` sera au mois représenté par `m` en l'an `a`, ceci évalué à partir du premier mai 2024.

Question P7.2 : Écrire en C une fonction qui prend en argument une chaîne de caractères `s` et un tableau de chaînes de caractères (ainsi que sa taille), toutes les chaînes du tableau étant de la même taille (pas à vérifier) et retournant l'indice au sein du tableau de la chaîne qui apparaît le plus souvent en tant que facteur dans `s`. En cas d'égalité, on renverra n'importe lequel des indices à égalité, de même si aucun facteur n'apparaît. Il est obligatoire d'utiliser l'algorithme de Rabin-Karp en choisissant une fonction de hachage arbitrairement.

Question P7.3 : On considère un tableau `t` d'entiers dont la taille est notée `n`. On suppose que ce tableau contient une et une seule fois chacun des `n` premiers entiers naturels et ceci ne sera pas à vérifier. Le tableau représente donc une permutation σ au sens où $\sigma(i)$ est représenté par `t[i]`. Le cours de maths permet de dire que toute permutation est composée d'un produit (au sens de la composition) de cycles à support disjoint (définition intuitive qu'on peut comparer à des composantes fortement connexes dans un graphe de degré sortant 1). Décrire (en pseudo-code ou implémenter l'algorithme en C ou en OCaml) un algorithme permettant de déterminer si la permutation représentée par `t` ne contient qu'un seul cycle (pas forcément de taille `n`, et s'il n'y a que des points fixes la réponse sera positive quand même). Prouver sa terminaison le cas échéant et calculer sa complexité en temps et en espace. Viser l'excellence : temps linéaire et espace constant.

Question P7.4 : Décrire (en pseudo-code ou implémenter l'algorithme en C ou en OCaml) un algorithme comptant le nombre de chemins eulériens dans un graphe. Prouver sa terminaison le cas échéant et calculer sa complexité en temps et en espace.

Question P7.5 : On signale ou rappelle qu'une clique dans un graphe non-orienté est un sous-graphe complet du graphe et qu'une clique maximale d'un graphe est une clique de ce graphe qui n'est strictement incluse dans aucune autre clique du même graphe. Décrire (en pseudo-code ou implémenter l'algorithme en C ou en OCaml) un algorithme construisant à partir d'un graphe non orienté le « graphe des cliques » du graphe, défini comme le graphe non orienté dont les sommets sont les cliques maximales du graphe en argument et les arêtes relient deux sommets correspondant à des cliques telles qu'au moins une arête existe entre un sommet de l'une et un sommet de l'autre.

Question P7.6 : On considère une variante de la recherche de motifs où le but est de trouver une occurrence la plus proche possible d'une position donnée, notée `i` (à égalité de distance on pourra prendre la première). Un algorithme naïf pourra lancer la recherche en `i`, puis en `i-1`, puis en `i+1`, puis en `i-2`, etc. L'algorithme de Rabin-Karp s'imagine aussi, en mémorisant deux hachés et avec une formule de passage d'un haché au suivant mais aussi au précédent. L'exercice ici consiste à adapter l'algorithme de Boyer-Moore. Décrire (en pseudo-code ou implémenter l'algorithme en C ou en OCaml) un algorithme pour cette tâche en respectant l'esprit de l'algorithme de Boyer-Moore.

Et on termine par un autre message d'Albus Dumbledore !

Question P7.7 : Une chaîne de caractères a été compressée puis sérialisée par l'algorithme de Huffman selon la méthode du TP 11. On donne ci-après le résultat en hexadécimal. Quelle était la chaîne initiale ?

Le résultat en question : 24 05 8E E0 B7 5C CB 7A 92 97 5B 16 52 D1 61 BA 51 17 31 DF 3F 98 4B C7 D6 02

On rappelle la méthode du TP 11 :

- Sérialiser l'arbre en faisant un parcours en profondeur : on produit un 0 pour chaque nœud interne et un 1 pour chaque feuille, ce 1 étant suivi du code ASCII du caractère à la feuille correspondante.
- Écrire tous les bits de l'encodage du texte initial.
- Écrire dans les trois derniers bits un nombre n en binaire pour signaler que seuls $6+n$ bits sont à considérer sur les deux derniers octets.

Annexe

À Poinca, une aide sera toujours apportée à ceux qui la demandent.

On rappelle à toute fin utile des fonctions du module `Hashtbl` avec des informations sur leur spécification :

- `Hashtbl.create n` crée une table de hachage avec `n` places pour commencer, mais en adaptant si besoin (donc on devine `n` sans qu'il n'y ait de risque si l'estimation est mauvaise) ;
- `Hashtbl.add th cle valeur` ajoute une association à la table de hachage, en masquant une éventuelle clé déjà existante (l'autre valeur sera de nouveau accessible en cas de retrait de ce qui l'a masqué) ;
- `Hashtbl.find th cle` détermine la valeur associée à la clé dans la table de hachage, en déclenchant l'erreur `Not_found` si la clé est absente ;
- `Hashtbl.mem th cle` détermine si la clé est présente dans la table de hachage ;
- `Hashtbl.remove th cle` retire une occurrence de la clé dans la table de hachage s'il y en a une (sinon la fonction n'a pas d'effet) ;
- `Hashtbl.replace th cle valeur` remplace la valeur associée à la clé dans la table de hachage par une nouvelle valeur (une éventuelle valeur masquée n'est pas impactée) en ajoutant la clé si elle n'y était pas encore.
- `Hashtbl.find_opt th cle` agit comme la fonction `find`, mais retourne une option pour éviter de lever une exception si la clé est absente ;
- `Hashtbl.iter f th` appelle la fonction fournie, prenant des clés et des valeurs (dans cet ordre) en argument, à tous les éléments de la table de hachage, sans contrôle sur l'ordre, sachant que si des clés sont masquées par d'autres clés identiques, elles subiront aussi la fonction (et on sait que ce sera dans l'ordre inverse de leur apparition dans la table de hachage).

Pour la manipulation des chaînes en C, on propose les fonctions suivantes :

- `atoi(chaine)` renvoie l'entier représenté dans la chaîne en argument (ne pas l'utiliser si la chaîne ne correspond pas exactement à un entier), la fonction est `atof` pour renvoyer un flottant de type `double` ;
- `sprintf(chaine, "%d", n)` écrit dans la chaîne en premier argument l'entier en troisième argument, ce qui écrase la chaîne et suppose que sa taille soit suffisante, le format devient `"%f"` pour un flottant ;
- `strcat(chaine1, chaine2)` écrit à la suite de la chaîne en premier argument la chaîne en deuxième argument (même remarque sur la taille) ;
- `strcpy(chaine1, chaine2)` écrit au début de la chaîne en premier argument la chaîne en deuxième argument (idem).

Quelques codes ASCII utiles : 32 pour l'espace, 46 pour le point, 65 pour le A, puis l'alphabet dans l'ordre jusqu'à 90 pour le Z, 97 pour le a puis l'alphabet dans l'ordre jusqu'à 122 pour le z.

Enfin, sachez que le professeur accepte dans sa classe ceux qui ont obtenu Effort Exceptionnel aux questions de cours et qui souhaitent s'inscrire.