PROBLÈME PRÉLIMINAIRE #1

COMPÉTITION DE PROGRAMMATION DE ROBOTIQUE CRC

Un programme de

AEST EAST

Version 1.0

QUELQUES NOTES

- Les règles complètes sont dans la section 4 du livret des règlements.
- Vous avez jusqu'au **dimanche 17 novembre, 23h59** pour remettre votre code.
- N'hésitez pas à utiliser le forum de programmation sur le discord de la CRC pour poser vos questions et discuter des problèmes. Il est là pour ça!
- **On vous donne des fichiers modèles faciles à utiliser pour votre code et pour faire vos tests. Vous devez les utiliser pour résoudre le problème!**

UTILISATION DU FICHIER MODÈLE

- Le fichier de test appelle la fonction associée avec en paramètre les informations du test et compare sa sortie avec ce qui est attendu pour vous permettre de voir si les tests réussissent. **Tout votre code (sauf fonctions additionnelles que vous créez) devrait être écrit dans la fonction prévue à cet effet.**
- Les points mis dans le document indiquent la difficulté et le pointage attribué pour la réussite pour chaque défi. Ce problème préliminaire aura une valeur globale de 2% du défi principal.

STRUCTURE

Une petite mise en situation comme celle-ci explique les fondements de chaque défi et offre les bases nécessaires pour résoudre celui-ci.

Spécification d'entrée et de sortie:

Contient les caractéristiques des entrées fournies ainsi que les critères attendus pour les sorties du programme.

Exemple d'entrée et de sortie:

Contient un exemple d'entrée, parfois constitué lui-même de plusieurs sous-exemples, pour que vous puissiez tester votre programme. Chaque exemple de sortie donne la réponse attendue pour l'entrée correspondante.

Explication de la première sortie:

Décortique davantage le défi en expliquant comment la première entrée est traitée et en montrant le chemin menant à cette réponse.

TakTiques avancées!

Votre équipe est en finale de la compétition TakTik 2025 et votre copilote veut transmettre des TakTiques cruciales pour le jeu. Afin de ne pas donner d'informations à vos adversaires, vous décidez d'échanger des messages cryptés. Comme le pilote n'a pas le temps de les décoder et de se concentrer sur le jeu, vous lui faites un système de décryptage automatique. Vous devez donc programmer toutes les portions de décryptage du message et les assembler. Ainsi, vous aurez un avantage sur toutes les autres équipes de la compétition et aurez plus de chances de gagner TakTik 2025!

Partie 1: Le bas de l'alphabet (20 points)

Pour faire le décryptage du message, vous devez sélectionner une seule lettre par phrase. Vous devez trouver la lettre de chaque phrase étant la première dans l'alphabet. Pour votre code secret, les lettres majuscules passeront en priorité sur les lettres minuscules. Donc, la priorité pour chaque phrase devrait être: A, B, C, [...], Z, a, b, c, [...], z.
Note: N'oubliez pas que les majuscules et minuscules n'ont pas la même valeur **ascii** pour la même lettre.

Spécification d'entrée et de sortie:

Vous recevrez en entrée un texte dans une variable de type *string* dont les phrases sont séparées par des signes de ponctuation terminaux (. ! ?). Vous devrez donner en sortie une variable de type *array* contenant des valeurs de type *string* de la lettre choisie pour chaque phrase dans l'entrée.

Exemple d'entrée:

"my c**a**t always finds a way to sneak inside! **E**very bird loves flying under the bright sky. he enjoys **c**ycling with his dog every morning."

"Samuel veut écrire du code en **P**ython. ton script est trop **c**ool! utilise-t-il **d**es modules externes?"

"**W**ait 1 second, he's not here anymore. j**e** ne le vois plus! me neither, where **d**id he go? je pense qu'il s'est perdu; cette ville est tellement **b**elle."

Exemple de sortie:

```
['a', 'E', 'c']
['P', 'c', 'd']
['W', 'e', 'd', 'b']
```

Explication de la première sortie:

Notons qu'il y a trois phrases dans la première entrée.

- Dans la première phrase, on regarde chaque lettre pour trouver la plus petite. On commence par ‘m’, suivi de ‘y’, qui est plus loin dans l’alphabet, donc on conserve ‘m’ comme valeur.
- Ensuite, on prend la lettre ‘c’, qui est la plus basse. Donc, on garde ‘c’.
- Puis, on croise 'a': on garde ‘a’ comme plus petite lettre.
- On peut regarder chaque lettre, jusqu'à ce qu'on atteigne le point '.' pour confirmer que la lettre 'a' a en fait la valeur la plus petite de cette phrase.

- Dans la deuxième phrase, on croise les lettres 'E', 'v', 'e', 'r', 'y', et enfin 'b'. **'E' et 'e' n'ont pas la même valeur ascii!**
- La lettre 'E' correspond à la valeur ascii la plus basse, donc on choisit le 'E'.
- On vérifie ensuite le reste des lettres (jusqu'au point d'exclamation '!') et on ne trouve pas de lettre ayant une valeur plus basse que 'E'.

- Dans la troisième phrase, après avoir analysé toutes les lettres de façon similaire, on voit que la lettre 'c' est la plus petite.

Pour le premier exemple, on renvoie un *array* contenant 3 valeurs de type *string* : la lettre ayant la valeur ascii la plus faible pour chaque phrase. Cela nous donne `['a', 'E', 'c']` comme sortie.

Partie 2: Voyelles et consonnes (20 points)

Vous voulez rendre votre message plus sécuritaire et, dans chaque phrase, vous avez décidé de faire un système de calcul caché. Vous voulez compter le nombre de voyelles et de consonnes dans chacune des phrases que vous recevez. Vous multipliez ces deux sommes ensemble pour obtenir la valeur numérique de la phrase. Les voyelles sont spécifiquement (a,e,i,o,u,y). N'oubliez pas que les lettres ayant des accents doivent être considérées dans le calcul (ex: è, ê et è comptent comme des voyelles).

Spécification d'entrée et de sortie:

En entrée, vous recevez une variable de type *string* contenant un texte de plusieurs phrases pour lesquelles vous devrez trouver la valeur associée.
En sortie, vous devrez donner une liste d'entiers (variables de types *int*) qui sont les valeurs associées à chaque phrase.

Exemple d'entrée:

"My cat always finds a way to sneak inside! Every bird loves flying under the bright sky. Fred enjoys cycling with his dog every morning."
"Pourquoi est-ce que j'ai le réflexe de regarder mes mains lorsque j'échappe un frisbee?"
"Maybe the elevator will arrive faster if I keep pressing the button. Je me suis lavé les mains après avoir cuisiné, mais elles sentent encore l'ail."

Exemple de sortie:

```
[270, 312, 350]
[1216]
[759, 986]
```

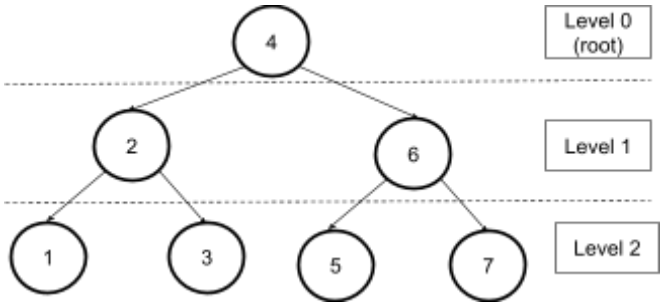
Explication de la première sortie:

Dans la première phrase "My cat always finds a way to sneak inside!", il y a 15 voyelles et 18 consonnes. On multiplie ces deux sommes pour obtenir 270, qui est donc la valeur associée à la première phrase. Pour la seconde phrase nous avons 13 voyelles et 24 consonnes. On trouve donc la valeur associée de 312. Pour la troisième phrase, nous avons 14 voyelles et 25 consonnes qui nous donnent une valeur associée de 350.

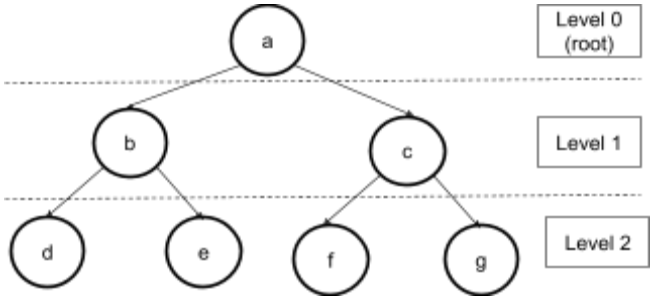
Partie 3: Arbre (40 points)

En informatique, les arbres sont des structures de données très utiles pour organiser de l'information. Dans un arbre, la valeur en haut occupe le niveau 0 et cette valeur est la racine de l'arbre. Nous pouvons placer les éléments de niveau 1 sous la racine de niveau 0. Tout nouvel élément ajouté à l'arbre doit avoir un lien avec un élément du niveau supérieur. Dans notre contexte, nous allons utiliser un arbre avec 2 branches par élément et seulement les niveaux 0 à 2.

Pour ce problème, comme il y a 3 niveaux (0, 1 et 2) et que chacun est complet, nous avons un total de 7 valeurs. En effet, le niveau 0 contient seulement la racine, le niveau 1 contient deux éléments et chacun de ces éléments se séparent en 2 pour donner 4 éléments au niveau 2. Tout cela pour un total de 7 éléments. Dans un arbre, les valeurs sont organisées du plus petit au plus grand en le regardant de gauche à droite. Voici un arbre avec les valeurs de 1 à 7.



Dans le contexte de notre décryptage, nous voulons afficher notre message sur l'arbre en prenant chaque valeur (1 à 7 dans cet exemple) et en la remplaçant par sa lettre associée dans l'arbre. Voici un exemple avec les nombres: [1, 2, 3, 4, 5, 6, 7] et les lettres associées ['d', 'b', 'e', 'a', 'f', 'c', 'g']. Dans cet exemple, d est associé à 1, b est associé à 2 et ainsi de suite jusqu'à g qui est associé à 7. Suite à la substitution des valeurs par les lettres associées, on obtient ce nouvel arbre.



Il y a plusieurs moyens de parcourir un arbre, mais pour notre décryptage, nous allons choisir de lire cet arbre en procédant par niveau, ce qui se nomme un parcours en largeur (**Breadth First Search**). Nous allons donc regarder chaque niveau l'un après l'autre et lire les valeurs de la gauche vers la droite. Avec l'arbre précédent, nous aurions le résultat:
['a', 'b', 'c', 'd', 'e', 'f', 'g'].

En résumé, nous allons construire un arbre avec notre liste de 7 nombres en entrée, les remplacer par les lettres correspondantes et finalement en faire la lecture avec la méthode de parcours en largeur.

Pour plus d'info sur les arbres en général, voici deux ressources intéressantes.
https://www.freecodecamp.org/news/all-you-need-to-know-about-tree-data-structures-bceacb85490c/
https://www.geeksforgeeks.org/introduction-to-tree-data-structure/

Spécification d'entrée et de sortie:

En entrée vous recevrez deux listes contenant chacune 7 éléments. La première liste sera les nombres à placer dans l'arbre et la seconde liste sera les lettres associées à chaque nombre.
En sortie, vous donnerez les 7 lettres dans un *array* organisées par un parcours en largeur de l'arbre.

Exemples d'entrée:

[31, 44, 14, 33, 43, 26, 57], ['i', 'o', 'n', 'u', 'c', 's', 'e']

[2, 4, 6, 5, 3, 1, 7], ['b', 'a', 'c', 'f', 'e', 'd', 'g']

[14, 41, 10, 59, 99, 109, 90], ['u', 'e', 'l', 'x', 'p', 't', 'f']

Exemples de sortie:

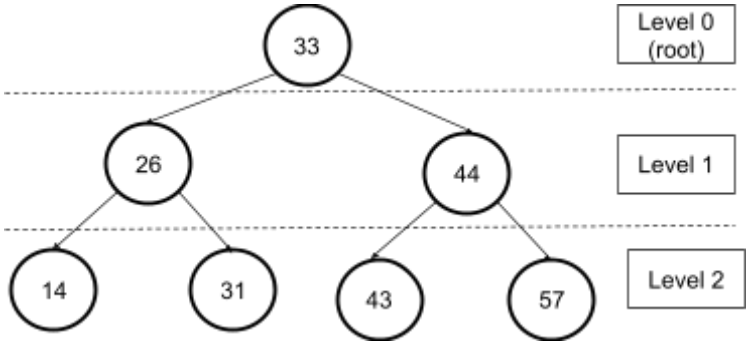
['u', 's', 'o', 'n', 'i', 'c', 'e']

['a', 'b', 'c', 'd', 'e', 'f', 'g']

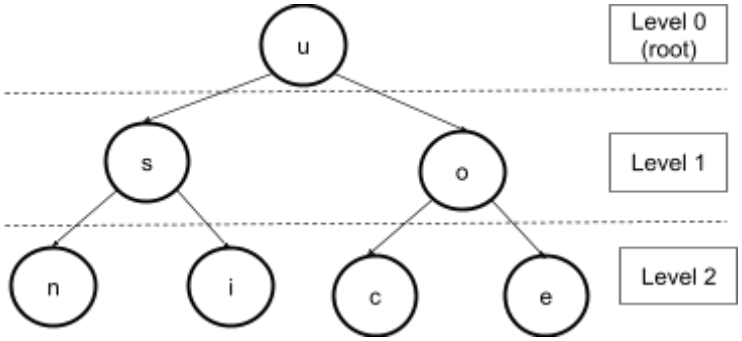
['x', 'u', 'p', 'l', 'e', 'f', 't']

Explication de la première sortie:

Pour le premier exemple, nous avons les nombres et lettres associées:
[31, 44, 14, 33, 43, 26, 57]
['i', 'o', 'n', 'u', 'c', 's', 'e']
Nous allons commencer par placer les nombres au bon endroit dans l'arbre en ordre croissant. Ainsi nous allons avoir les valeurs de gauche à droite pour la plus petite à la plus grande. Voici l'arbre une fois que les nombres sont placés.



On peut maintenant remplacer les nombres par leur lettre associée. On obtient l'arbre suivant:



Une fois l'arbre construit avec les lettres qui remplacent les nombres, nous pouvons en faire le parcours en largeur (Breadth First Search) et obtenir le résultat:
['u', 's', 'o', 'n', 'i', 'c', 'e']

Partie 4: Afficher les arbres (15 points)

Maintenant que vous êtes capable d'organiser les éléments d'un *array* sous forme d'arbre, vous devez trouver une manière d'afficher cet arbre. Vous recevrez encore une liste de 7 valeurs. À partir de cette liste, vous allez devoir afficher les éléments de la liste sur plusieurs lignes, formant un affichage en deux dimensions! Les éléments de la liste seront placés dans le même ordre que dans l'*array* de la sortie de la partie 3, soit par étage (parcours en largeur/breadth first search). L'arbre visuel doit être composé des éléments de la liste, d'espaces (blank space) [code ascii 32] et des flèches dans l'arbre (↘ et ↙) [unicode 2198 et unicode 2199]. Pour vous aider à faire l'affichage, un exemple avec les valeurs de 1 à 7 vous sera fourni. Vous pouvez vous servir de cet arbre pour faire votre affichage.

Spécification d'entrée et de sortie:

En entrée, vous recevrez un array des lettres à afficher en forme d'arbre.
En sortie, vous devez fournir un array de string qui forme le visuel de l'arbre.

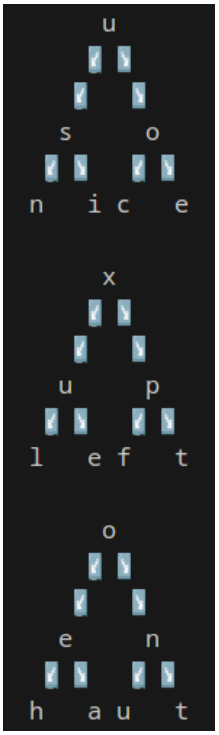
Exemple d'entrée:

['u', 's', 'o', 'n', 'i', 'c', 'e']
['x', 'u', 'p', 'l', 'e', 'f', 't']
['o', 'e', 'n', 'h', 'a', 'u', 't']

Exemple de sortie:

* Le formatage de google docs n'est pas le même que celui de la console

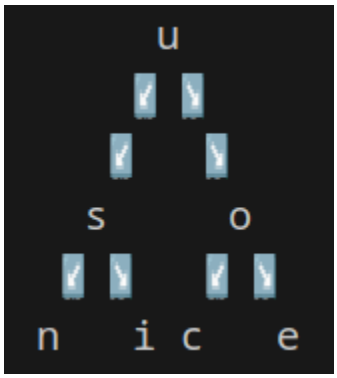
```
[
"      u       ",
"    ↙ ↘       ",
"   ↙   ↘      ",
"  s     o     ",
" ↙ ↘   ↙ ↘    ",
"n   i c   e"]

[
"      x       ",
"    ↙ ↘       ",
"   ↙   ↘      ",
"  u     p     ",
" ↙ ↘   ↙ ↘    ",
"l   e f   t"]
```



```
[
"      o       ",
"    ↙ ↘       ",
"   ↙   ↘      ",
"  e     n     ",
" ↙ ↘   ↙ ↘    ",
"h   a u   t"]
```

Explication de la première sortie:

Pour le premier exemple, nous avons en entrée le tableau de lettres: [‘u’, ‘s’, ‘o’, ‘n’, ‘i’, ‘c’, ‘e’]. On peut mettre chacune des valeurs dans l’arbre en faisant un parcours en largeur (BFS) ou en remplaçant dans l’exemple fourni. Une fois les valeurs remplacées, on peut voir le résultat en utilisant la fonction d’aide print_tree() qui nous donne l’affichage suivant:



On peut retourner l‘array de string avec les valeurs remplacées donc nous retournons:

```
[
"      u       ",
"    ↙ ↘       ",
"   ↙   ↘      ",
"  s     o     ",
" ↙ ↘   ↙ ↘    ",
"n   i c   e"]
```

Le formatage de Google Docs et de la console diffèrent, donc servez-vous de l’exemple donné pour vous aider à réussir le formatage et ainsi passer les tests.

Partie 5: Décoder les messages secrets (15 points)

Dans cette partie, vous devez combiner toutes les parties précédentes pour décoder les messages cryptés. Vous recevrez donc une variable de type *string* de 7 phrases. Pour ces phrases vous devrez trouver les lettres avec la valeur ascii la plus basse (comme dans la section 1) et aussi la multiplication du nombre de voyelles et de consonnes (comme la partie 2). Vous devrez la mettre en arbre puis l'afficher avec l'aide des parties 3 et 4. Ainsi, vous aurez le message secret décodé que vous pourrez afficher à votre pilote durant la partie. C'est la dernière étape pour vous donner un avantage lors de TakTik 2025!

Spécification d'entrée et de sortie:

Vous allez recevoir une valeur de type *string* contenant 7 phrases, que vous devez décoder pour cette partie.
En sortie, vous devrez donner un *array* de variables de type *string* qui représente l'affichage en arbre du message secret que vous avez décodé.

Exemple d'entrée:

"**T**ommy! **A**pproche! **I**ci ils font des gros robots! **T**hey are for the TakTik 2025 competition. **K**oooooIII robots, that seems like so much fun! **K**evin might also be interested to join the robotics team with us. **S**weet, if both of you are joining, we will have a super good team and might win the event!"

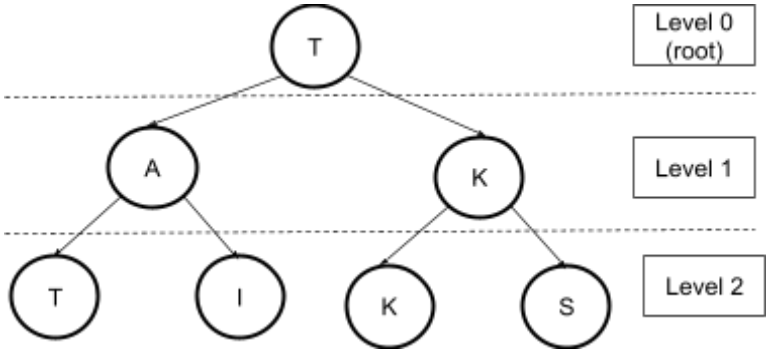
Exemple de sortie:

```
[
"      T       ",
"    ↙ ↘       ",
"   ↙   ↘      ",
"  A     K     ",
" ↙ ↘   ↙ ↘    ",
"T   I K   S"]
```

Explication de la première sortie:

- Dans ce problème nous prenons en entrée les 7 phrases, et trouvons 7 lettres associées en suivant les règles de la partie 1. Nous obtenons les lettres suivantes: ['T', 'A', 'I', 'T', 'K', 'K', 'S'].
- Nous prenons par la suite les nombres des voyelles et consonnes de chaque phrase. Nous obtenons les résultats de multiplication suivants: [6, 15, 120, 221, 330, 672, 1170].

- Une fois que nous avons ces deux informations, nous pouvons construire un arbre avec les valeurs des deux premières parties.



- En utilisant le *parcours en largeur* (BFS), nous pouvons récupérer les lettres: [‘T’, ‘A’, ‘K’, ‘T’, ‘I’, ‘K’, ‘S’]
- Une fois que nous avons les en ordre pour notre message final, nous pouvons le formater pour l’afficher et obtenir:

```
[
"      T       ",
"    ↙ ↘       ",
"   ↙   ↘    ",
"  A      K   ",
" ↙ ↘   ↙ ↘ ",
"T   I K   S"]
```

Ceci nous donne le visuel suivant dans une console: