Le sujet est composé de deux exercices entièrement indépendants.

Exercice I

Vous avez été sélectionné(e) pour participer au jeu « Cherchez les Clés du Paradis (CCP) ». Le jeu se déroule en deux épreuves, au cours desquelles vous devez collecter des clés vertes. A l'issue de chacune d'entre elles, vous passez à l'épreuve suivante en cas de succès et êtes éliminé(e) en cas d'échec.

1 - Première épreuve

Jean-Pierre Pendule, le célèbre animateur, vous accueille pour la première épreuve. Il vous explique la règle du jeu. Devant vous, deux boîtes et sur chacune d'entre elles une inscription. Chacune des boîtes contient soit une clé verte, soit une clé rouge. Vous devez choisir l'une des boîtes : si le résultat est une clé rouge, alors vous quittez le jeu, si c'est une clé verte vous êtes qualifié(e) pour l'épreuve suivante.

Jean-Pierre Pendule dévoile les inscriptions sur chacune des boîtes et vous affirme qu'elles sont soit vraies toutes les deux, soit fausses toutes les deux :

— sur la boîte 1, il est écrit « Une au moins des deux boîtes contient une clé verte » ;
— sur la boîte 2, il est écrit « Il y a une clé rouge dans l'autre boîte ».

Dans toute cette partie, on note V_i la proposition affirmant qu'il y a une clé verte dans la boîte i.

1) Donner une formule de la logique des propositions représentant la phrase écrite sur la boîte 1.
2) Donner de même une formule de la logique des propositions pour l'inscription de la boîte 2.
3) Donner une formule représentant l'affirmation de l'animateur. Simplifier cette formule de sorte à n'obtenir qu'une seule occurrence de chaque V_i.
4) Quel choix devez-vous faire pour continuer le jeu à coup sûr ?

2 - Deuxième épreuve

Vous voici arrivé(e) à la dernière épreuve. Jean-Pierre Pendule vous dévoile une troisième boîte et vous explique les règles du jeu. Dans une des boîtes se cache la clé qui vous permet de remporter la victoire finale. Dans une autre boîte se cache une clé rouge qui vous fait tout perdre. La dernière boîte est vide. Encore une fois, chacune des boîtes porte une inscription :

— sur la boîte 1, il est écrit « La boîte 3 est vide » ;
— sur la boîte 2, il est écrit « La clé rouge est dans la boîte 1 » ;
— sur la boîte 3, il est écrit « Cette boîte est vide ».

L'animateur affirme que l'inscription portée sur la boîte contenant la clé verte est vraie, celle portée par la boîte contenant la clé rouge est fausse. L'inscription affichée sur la boîte vide est aussi vraie.

5) Donner une formule de la logique des propositions pour chaque inscription. On pourra introduire des propositions R_i affirmant que la clé rouge se trouve dans la boîte i.
6) Donner une formule de la logique des propositions synthétisant l'information que vous apportée l'animateur.
7) En supposant que la clé verte est dans la boîte 2, montrer par l'absurde que l'on aboutit à une incohérence.
8) Donner alors la composition des trois boîtes.

Exercice II

Nous proposons dans cette partie d'étudier une méthode de compression de données. L'algorithme proposé ici implémente plusieurs couches d'arrangement de données et de compressions successives, utilisées dans l'ordre suivant pour la compression et l'ordre inverse pour la décompression :

(i) Transformation de Burrows-Wheeler,

(ii) Codage par plages (RLE),

Soit Σ un alphabet de symboles, de cardinal $|\Sigma| = h$. On munit Σ d'une relation d'ordre $\leq$.

Pour tout entier naturel k, on note Σ^k l'ensemble des mots de longueur k construits à partir de Σ. Il est muni de la relation d'ordre lexicographique induite par la relation d'ordre $\leq$.

Pour $\mu \in \Sigma^k$, on note $|\mu| = k$ la taille de μ et $|\mu|_a$ le nombre d'occurrences de $a \in \Sigma$ dans μ.

Un mot $\mu \in \Sigma^k$ sera représenté par une liste de caractères. On définit donc le type

```
type mot = char list
```

Les paragraphes suivants étudient les algorithmes et propriétés de ces deux phases, chacun d'entre eux pouvant être abordé **de manière indépendante**.

1 - Transformation de Burrows-Wheeler (BWT)

Soit $\mu \in \Sigma^k$ un mot. La transformation BWT réalise une permutation des symboles de μ de sorte que les symboles identiques sont regroupés dans de longues séquences. Cette transformation n'effectue pas de compression, mais prépare donc à une compression plus efficace.

Dans la suite, nous étudions le codage et le décodage d'un mot transformé par cette opération.

Phase de codage

On rajoute à la fin de μ un marqueur de fin (par convention noté |, inférieur par $\leq$ à tous les autres symboles de Σ). Dans toute la suite, $\hat{\mu}$ désigne le mot auquel on a ajouté le symbole |.

1) Pour $\mu = turlututu$, construire une matrice M dont les lignes sont les différentes permutations circulaires successives du mot $\hat{\mu}$. Les permutations seront ici envisagées par décalage à droite des caractères.

2) Écrire une fonction `circulaire : mot` $\rightarrow$ `mot` qui réalise une permutation à droite d'un mot μ donné en entrée.

3) Écrire une fonction `matrice_mot : mot` $\rightarrow$ `mot list` qui construit la matrice M à partir d'un mot passé en entrée. La valeur de retour est une liste de mots. Cette fonction utilisera la fonction `circulaire`.

Une permutation des lignes de M est alors effectuée, de sorte à classer les lignes par ordre lexicographique. On note $M' = P \cdot M$ la matrice obtenue, P étant la matrice de permutation.

4) Donner les matrices P et M' dans le cas du mot $\hat{\mu}$, pour $\mu = turlututu$.

Pour construire la matrice de permutation P, il faut trier la liste des mots définissant M.

5) Écrire une fonction `compare : mot -> mot -> bool` tel que `compare mot1 mot2` indique si le premier argument est inférieur au deuxième (pour l'ordre lexicographique). On pourra utiliser les opérations `< , > , <=` et `>=` entre deux éléments de type `char`.

6) Écrire une fonction récursive `tri : mot list -> lot list` qui réalise le tri d'une liste de mots. On pourra réaliser un tri par insertion ou un autre tri en expliquant clairement l'algorithme.

7) En déduire une fonction `matrice_mot_triée : mot` $\rightarrow$ `mot list` qui construit M' à partir de μ.

8) Pour $\mu \in \Sigma^k$, donner le nombre de comparaisons de symboles nécessaires au pire des cas, pour trier deux permutations circulaires du mot μ.

9) En déduire la complexité dans le pire des cas pour le tri des k permutations circulaires d'un mot $\mu \in \Sigma^k$ (exprimée en nombre de comparaisons de symboles).

La transformation BWT consiste alors à coder le mot μ par la dernière colonne de la matrice M' obtenue à partir de $\hat{\mu}$. On note $BWT(\mu)$ ce codage.

10) Écrire une fonction `codageBWT : mot` $\rightarrow$ `mot` qui encode un mot passé en entrée. On utilisera une fonction récursive pemettant de récupérer le dernier symbole de chacun des mots de M'. Donner le codage du mot $\mu = turlututu$.

Phase de décodage

Pour décoder un mot codé par BWT, il est nécessaire de reconstruire itérativement M' à partir de la seule donnée du mot code $BWT(\mu)$. Par construction, $BWT(\mu)$ est la dernière colonne de M'.

On pose ici comme exemple $BWT(\mu) = edngvnea|$.

11) Construire, à partir de la seule donnée de $BWT(\mu)$, la première colonne de M'. Justifier le principe de construction.

La dernière et la première colonne de M' donnent alors tous les sous-mots de longueur 2 du mot μ.

12) Proposer un algorithme permettant d'obtenir la deuxième colonne de M'. Donner cette deuxième colonne pour $BWT(\mu) = edngvnea|$.

13) On dispose à l'itération i des $(i-1)$ premières colonnes de M' et de sa dernière colonne. Proposer un principe algorithmique permettant de construire la i-ième colonne de M'.

14) En déduire un algorithme itératif permettant de reconstruire M' puis μ.

15) Quel décodage obtient-on pour le mot $BWT(\mu)$ proposé ?

2 - Codage par plages RLE

Le codage RLE (Run Length Coding), ou codage par plages, est une méthode de compression dont le principe est de remplacer dans une chaîne de symboles une sous-chaîne de symboles identiques par le couple constitué du nombre de symboles identiques et du symbole lui même. Par exemple, la chaîne « aaababb » est compressée en $[(3, `a`), (1, `b`), (1, `a`), (2, `b`)]$.

16) Écrire une fonction `rle : mot -> (int*char) list` qui code un mot passé en entrée par codage RLE.

17) Écrire une fonction `decodeRLE : (int*char) list -> mot` qui décode une liste issue du codage RLE d'un mot.