TD15. Révisions MP2I

I. Représentations numériques

Exercice 1. Les entiers
1. Donner la représentation binaire de 56.
2. Sur n bits, quel intervalle d'entiers non signés peut-on représenter ?
3. Écrire une fonction OCaml `binaire : int -> int list` telle que `binaire x` renvoie la liste des bits de la représentation binaire de `x`, du bit de poids fort au bit de poids faible.
4. Écrire une fonction `decimal : int list -> int` qui prend en entrée une liste de bits et renvoie l'entier dont cette liste est la représentation binaire. On veillera à ce que la fonction proposée linéaire en la taille de la liste.
5. Sur n bits en compléments à 2, quel intervalle d'entiers peut-on représenter ?
6. Donner la représentation en complément à 2 sur 8 bits de -27.

Exercice 2. Les flottants
Les flottants à simple précision sont représentés en machine sur 32 bits, en suivant la norme `IEEE754` :

- le premier bit représente le signe ;
- les 8 bits suivants représentent l'exposant ;
- les 23 derniers bits représentent la mantisse.

Le flottant de signe s, d'exposant e et de mantisse m vaut $2^{e-127} \times 1, m$ si $s = 0$, et $-2^{e-127} \times 1, m$ si $s = 1$.

1. Donner la représentation décimale du flottant $1\underbrace{0\ldots011}_{8\text{ bits}}\ \underbrace{1010\ldots0}_{23\text{ bits}}$.
2. Donner la représentation flottante de $12,4$.
3. On appelle précision machine la distance ε qui sépare 1 et le flottant le plus proche qui lui est strictement supérieur. Que vaut ε ?

II. Programmation dynamique

Exercice 3. Problème du rendu de monnaie
On s'intéresse au problème suivant :
Pour $0 \leq k \leq n$ et $0 \leq s \leq S$, on note $nb(k, s)$ le nombre minimal de pièces parmi $\{p_1, \ldots, p_k\}$ nécessaire pour atteindre s (et $+\infty$ si ce n'est pas possible).

1. $\forall\ 0 \leq s \leq S$, que vaut $nb(0, s)$?
2. $\forall\ 0 \leq k \leq n$, que vaut $nb(k, 0)$?
3. Établir une relation de récurrence vérifiée par les $nb(k, s)$.
4. En déduire un fonction C `int rendu_monnaie(int S, int* p, int n)` prenant en entrée une somme S à atteindre, un tableau des pièces de taille n, et renvoyant le nombre minimum de pièces à fournir pour atteindre S.
On implémentera la méthode de programmation dynamique de façon ascendante.

Exercice 4. Palindromes (CCINP 2023)
Soit Σ un alphabet fini contenant au moins deux lettres. On note $u = u_0 \cdots u_{n-1}$ un mot sur Σ, composé de n lettres $u_i \in \Sigma$, $i \in [\![0, n-1]\!]$. La longueur de u est notée $|u|$. Pour $0 \leq i < j \leq n$, on note $u[i, j]$ le mot $u_i \cdots u_{j-1}$. L'ensemble des mots construits sur Σ et contenant le mot vide ε est noté Σ^*.

Définition 1 (Miroir)
Soit $u \in \Sigma^*$. Le miroir de $u = u_0 \dots u_{n-1}$, noté $\bar{u}$, est le mot $\bar{u} = u_{n-1} \dots u_0$. Par convention, $\bar{\varepsilon} = \varepsilon$.
Définition 2 (Palindrome)
$u \in \Sigma^*$ est un palindrome si et seulement si $u = \bar{u}$.

Par convention, le mot vide ε n'est pas considéré comme un palindrome.

On recherche le nombre de palindromes facteurs d'un mot $u \in \Sigma^*$ (comptés avec les multiplicités éventuelles), soit le cardinal de l'ensemble $\{(i, j),\ 0 \leq i < j \leq |u|,\ u[i, j] = u[\bar{i}, j]\}$. Dit autrement, on cherche le nombre de palindromes contenus dans u.

1. Si $\Sigma = \{a, b\}$, donner le nombre de palindromes contenus dans le mot $u = babb$.

En parcourant naïvement les lettres d'un mot u donné, on peut proposer un algorithme en pseudo-code permettant de compter le nombre de palindromes de u (**Algorithme 1**).

Algorithme 1 - Décompte naïf du nombre de palindromes contenus dans un mot donné.

```
Entrées : Un mot u.
Sorties : Le nombre de palindromes contenus dans u.
début
    nb = 0
    pour i=0 à |u|-1 faire
        pour j=0 à |u|-1 faire
            estPalindrome=True
            pour k=i à j − 1 faire
                si u[i] ≠ u[j − k − 1] alors
                    estPalindrome=False
                    Sortir du pour k
            si estPalindrome alors
                nb = nb+1
    retourner nb
```

2. Évaluer la complexité dans le pire des cas de l'**Algorithme 1** en fonction de $|u|$.

Pour améliorer cette première idée, on utilise le paradigme de la programmation dynamique. Pour $u \in \Sigma^*$, on définit un tableau de booléens P de taille $(|u| + 1) \times (|u| + 1)$ tel que $P[i][j]$ vaut vrai si $u[i, j]$ est un palindrome. On a donc pour tout $i \in [\![0, |u| - 1]\!]$, $P[i][i + 1] = True$.

3. Soit $u[i, j]$ un mot. À quelles conditions sur u_i, u_{j-1} et $u[i + 1, j - 1]$ le mot $u[i, j]$ est-il un palindrome ?
4. En déduire une relation de récurrence vérifiée par les coefficients de P.
5. Écrire un algorithme de programmation dynamique en pseudo-code résolvant le problème. Évaluer sa complexité.

III. Algorithmes du texte

Exercice 5. Algorithme de Huffman
On considère un texte dont les occurences des lettres sont données par le tableau suivant :

a	b	c	d	e	f	g
18	10	5	3	9	29	28

1. Construire l'arbre de compression de Huffman associé à ce texte. On disposera les feuilles pour qu'elles soient dans l'ordre alphabétique de gauche à droite.
2. Compresser le mot $u = aabfecdag$.
3. Décompresser l'encodage $x = 100001001111101100$.

Exercice 6. Algorithme de Rabin-Karp
On considère un mot sur l'alphabet $\Sigma = \{a, b, c, d\}$, dont les lettres sont numérotées par la table suivante

a	b	c	d
0	1	2	3

On définit la fonction de hachage d'un mot $u = u_0 \dots u_{n-1}$ par :

$$h(u) = \sum_{i=0}^{n-1} u_i 4^{n-1-i} \mod 7$$

1. Donner la valeur de $h(bcc)$.
2. Combien de comparaisons de caractères ont lieu lorsqu'on cherche le mot *bcc* dans le texte *abadacbcc* selon l'algorithme de Rabin-Karp ?

Exercice 7. Algorithme de Boyer-Moore
On considère le mot $u = concours$.
1. Donner la table de décalage du mot u selon l'algorithme de Boyer-Moore.
2. Combien de décalages et de comparaisons de caractères ont lieu lorsqu'on cherche le mot u dans le texte suivant selon l'algorithme de Boyer-Moore ?

$$t = \texttt{jefinislecours,jecoursauxconcours}$$

Exercice 8. Algorithme de LZW
1. Dérouler manuellement la compression et la décompression de la chaîne "abracadabra" avec l'algorithme LZW.
2. Étudier le comportement de l'algorithme LZW dans le cas où le texte à compresser est constitué de N occurrences d'un unique caractère.

IV. Bases de données

Exercice 9. Étude de tournois en langage SQL (CCMP 2025)
On étudie des *tournois* d'échecs lors desquels s'affrontent exactement deux joueurs (A et B), au cours de plusieurs matchs. Le vainqueur du tournoi est le joueur qui a gagné le plus grand nombre de matchs. Le rôle de chaque joueur (*Blanc* ou *Noir*) peut changer entre chaque match. On dispose d'une base de donnée SQL composée de trois tables (les clés primaires sont soulignées)

- Joueurs(<u>JoueurID</u>, NomJ)
- Tournois(<u>TournoiID</u>, NomT, JoueurAID, JoueurBID)
- Matchs(<u>MatchID</u>, TournoiID, RoleJoueurA, VainqueurID)

dont voici certains extraits :

JoueurID	NomJ
2	Alice
7	Bob
8	Charlie

TournoiID	NomT	JoueurAID	JoueurBID
82	12Dec	2	8
59	17Fev	7	8
61	29Mai	8	2

MatchID	TournoiID	RoleJoueurA	VainqueurID
119	82	Blanc	2
934	82	Noir	8
925	82	Noir	2
384	61	Blanc	2

Par exemple, le tournoi du 12 décembre a opposé Alice et Bob, il s'est joué en 3 matchs. Alice, qui a gagné deux des trois matchs, a été victorieuse.

1. Écrire une requête SQL renvoyant le nombre de matchs joués dans chaque tournoi.
2. Écrire une requête SQL renvoyant le nombre de tournois gagnés par chaque joueur.
3. Écrire une requête SQL renvoyant le nombre de fois où chaque joueur a joué le rôle de *Blanc*.