Dans ce sujet, on s'intéresse au concept de ***code correcteur*** qui permet de communiquer de manière fiable, malgré la présence inévitable de bruit sur les canaux de transmission. De nos jours, les codes correcteurs sont utilisés intensivement sur tous les réseaux de communication (internet, téléphonie ...).

Remarques concernant la programmation

Tableaux. Un appel à « `Array.make n x` » renvoie un tableau de longueur `n` dont tous les éléments sont égaux à `x`.

Nombres aléatoires. Le module `Random` permet de générer des nombres aléatoires. En particulier, la fonction « `Random.float: float -> float` » prend en entrée un nombre $x \geqslant 0$ et renvoie un nombre aléatoire choisi uniformément dans $[\,0\,;x\,]$.

Par exemple, la fonction « `alea: unit -> bool` » renvoie `true` avec probabilité 1/3 et `false` avec probabilité 2/3. L'appel à `Random.self_init` permet à OCaml d'initialiser le générateur de nombres aléatoires. Dans la suite, on suppose que cet appel a déjà été effectué, vous n'avez donc pas besoin de le réécrire sur votre copie.

```
Random.self_init();;

let alea(): bool =
  let p = Random.float 1. in
  p < 1./.3.;;
```

Caractères et chaînes de caractères. En OCaml, un caractère (type `char`) se définit en utilisant deux apostrophes (exemples : `'a'`, `'A'`, `'2'`) et une chaîne de caractères (type `string`) se définit avec des guillemets (exemples : `""`, `"Ceci est une chaîne de caractères"`, `"00100100"`).

Étant donnée une chaîne de caractères « `s: string` », sa taille est donnée par « `String.length s` » et le caractères d'indice `i` par `s.[i]` (qui est de type `char`).

La fonction `String.init` prend en entrée un entier `n` ainsi qu'une fonction « `f: int -> char` » et renvoie la chaîne de caractères de taille `n` dont le caractère d'indice `i` est « `f i` ». Par exemple, le code ci-contre affiche la chaîne de caractères `"Tp4!!!"`.

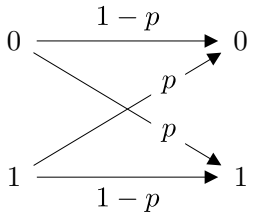
```
let generer_string () =
  let f i = match i with
    | 0 -> 'T'
    | 1 -> 'p'
    | 2 -> '3'
    | _ -> '!'
  in
  String.init 6 f;;
print_string (generer_string());;
```

1 Introduction

Alice souhaite envoyer à Bob un message composé de zéros et de uns. Malheureusement, le seul canal de communication auquel ils ont accès est bruité, ce qui signifie qu'un message donné en entrée du canal est légèrement modifié avant d'en atteindre la sortie. Le but des deux protagonistes est de mettre en place un protocole permettant à Bob de retrouver avec une bonne probabilité le message envoyé par Alice.

Dans tout le sujet, on note $\mathbb{F}_2 = \{0, 1\}$. Un élément de $\mathbb{F}_2$ est appelé un ***bit*** et on dira qu'un bit b a été ***flippé*** s'il a été remplacé par $(1 - b)$ (0 devient 1 et 1 devient 0). Un ***mot binaire*** de longueur $n \in \mathbb{N}^*$ est un n-uplet $x \in \mathbb{F}_2^n$. On se place dans le cas particulier où le canal de communication entre Alice et Bob est un ***canal binaire symétrique*** de paramètre p (voir définition qui suit).

Soit $p \in [\,0\,;1\,]$. Un bit envoyé via un canal binaire symétrique de paramètre p est flippé avec probabilité p et reste intact avec probabilité $1-p$. En d'autres termes, lorsqu'Alice envoie le bit 0 (resp. 1), Bob reçoit le bit 0 (resp. 1) avec probabilité $1-p$ et le bit 1 (resp. 0) avec probabilité p. De plus, quand Alice envoie plusieurs bits via le canal, chacun des bits est flippé avec probabilité p de manière indépendante.



En OCaml, un mot binaire est représenté par une chaîne de caractères « `x: string` » composée uniquement de caractères `'0'` et `'1'`.

1. Écrire une fonction « `canal: float -> string -> string` » qui prend en entrée une probabilité p ainsi qu'un mot binaire x, et renvoie le mot binaire y obtenu à la sortie du canal binaire symétrique de paramètre p.
2. Supposons qu'Alice envoie un mot binaire $x \in \mathbb{F}_2^n$ via le canal binaire symétrique de paramètre p. Quelle est la probabilité que Bob reçoive exactement x (c'est à dire qu'aucun bit n'ait été flippé) ?

Dans le but de communiquer de manière fiable, Alice et Bob décident d'utiliser des ***codes correcteurs***. L'idée est de ne s'autoriser à envoyer via le canal que certains mots binaires. Soit $k \in \mathbb{N}^*$ et $n \in \mathbb{N}^*$ deux entiers tels que $n \geqslant k$. On suppose disposer d'un ensemble $\mathcal{M} \subset \mathbb{F}_2^n$ associé à une bijection $\varphi : \mathbb{F}_2^k \to \mathcal{M}$. Les éléments de $\mathcal{M}$ sont appelés ***mots de code*** ; ce sont ces mots binaires (et seulement eux) qui peuvent être envoyés via le canal. Voici le protocole adopté par Alice et Bob :

(Étape 1) Alice découpe son message original en plusieurs blocs, chaque bloc étant composé de k bits. Dans le reste du protocole, chaque bloc est traité indépendamment, on suppose donc sans perte de généralité que le message d'Alice est un mot binaire $w \in \mathbb{F}_2^k$.

(Étape 2) Alice envoie $x = \varphi(w) \in \mathbb{F}_2^n$ à Bob via le canal binaire symétrique qui reçoit $y \in \mathbb{F}_2^n$. Par définition du canal binaire symétrique, y est donc égal à x où chaque bit a été flippé avec probabilité p de manière indépendante (voir question 1).

(Étape 3) Bob utilise y pour déterminer l'élément $\hat{x} \in \mathcal{M}$ qui lui semble être le plus probable pour x.

(Étape 4) Bob calcule $\hat{w} = \varphi^{-1}(\hat{x})$.

Finalement, la communication est un succès lorsque $w = \hat{w}$.

2 Code de répétition

Soit $n \in \mathbb{N}^*$ un entier impair. Dans un premier temps, nous allons étudier le ***code de répétition*** à n bits pour lequel $k = 1$. L'ensemble des mots de code $\mathcal{M}$ et la bijection $\varphi : \mathbb{F}_2 \to \mathcal{M}$ sont définis par :

$$\mathcal{M} = \Big\{(0,0,\ldots,0),(1,1,\ldots,1)\Big\} \qquad\qquad \varphi : \begin{cases} 0 \mapsto (0,0,\ldots,0) \\ 1 \mapsto (1,1,\ldots,1) \end{cases}$$

Lors de l'étape 3 de la partie 1, Bob doit déterminer quel est l'élément de $\mathcal{M}$ qu'Alice lui a envoyé. Il choisit d'utiliser un ***vote de majorité***, c'est à dire que si y comporte plus de 0 que de 1 alors il décide que $\hat{x}$ vaut $(0,0,\ldots,0)$, sinon il décide que $\hat{x}$ vaut $(1,1,\ldots,1)$. Par exemple, le tableau ci-dessous présente pour $n = 3$ les huit valeurs possibles de y et les valeurs de $\hat{x}$ et $\hat{w}$ correspondantes :

y	$(0,0,0)$	$(0,0,1)$	$(0,1,0)$	$(0,1,1)$	$(1,0,0)$	$(1,0,1)$	$(1,1,0)$	$(1,1,1)$
$\hat{x}$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(1,1,1)$	$(0,0,0)$	$(1,1,1)$	$(1,1,1)$	$(1,1,1)$
$\hat{w}$	0	0	0	1	0	1	1	1

3. (a) Écrire une fonction « `phi: int -> string -> string` » qui prend en entrée l'entier n ainsi que le message $w \in \mathbb{F}_2$ et renvoie le mot de code $x = \varphi(w) \in \mathbb{F}_2^n$. Si la chaîne de caractères donnée en entrée n'est ni `"0"`, ni `"1"`, votre fonction déclenchera une erreur.

(b) Écrire une fonction « `phi_inv: int -> string -> string` » qui prend en entrée l'entier n ainsi qu'un mot binaire $\hat{x} \in \mathcal{M}$ et renvoie $\hat{w} = \varphi^{-1}(\hat{x})$. Si la chaîne de caractères donnée en entrée ne représente pas un élément de $\mathcal{M}$, votre fonction déclenchera une erreur.

4. Écrire une fonction « `vote_maj: string -> string` » qui prend en entrée un mot binaire y et renvoie le mot binaire $\hat{x}$ obtenu par un vote de majorité. On pourra supposer sans le vérifier que la chaîne de caractères donnée en entrée est de taille impaire et est composée uniquement de zéros et de uns.

5. (a) Déterminer la probabilité $S(r, p)$ que la communication soit un succès pour un code de répétition à $n = 2r + 1$ bits. On exprimera le résultat à l'aide d'une somme en fonction de r et de p.

   (b) Calculer $S(r + 1, p)$ en fonction de $S(r, p)$. En déduire la discussion de leur comparaison.

Afin de vérifier expérimentalement la valeur obtenue à la question 5a, on décide de simuler le protocole de communication entre Alice et Bob et d'en calculer le taux de succès. Une simulation du protocole se déroule de la manière suivante :

→ On génère un message initial w valant `"0"` avec probabilité 1/2 et valant `"1"` avec probabilité 1/2.

→ On calcule les mots binaires x, y, $\hat{x}$ et $\hat{w}$ comme décrit dans la partie 1.

→ On vérifie si la communication est un succès.

L'idée est alors d'effectuer k simulations du protocole avec k suffisamment grand. Le taux de succès (nombre de transmissions réussies divisé par le nombre total de transmissions) sera environ égal à la probabilité calculée à la question 5a.

6. Écrire une fonction « `simul: float -> int -> int -> float` » qui prend en entrée la probabilité p ainsi que deux entiers (n, m), et renvoie le taux de succès du protocole de communication lorsqu'on envoie m bits à l'aide d'un code de répétition à n bits. On pourra supposer sans le vérifier que $n, m \in \mathbb{N}^*$ et que n est impair.

3 Codes linéaires

3.1 Définitions et Notations

Soit $G = (V \cup C, A)$ un graphe biparti. On note $v_0, v_1, \ldots, v_{n-1}$ les éléments de V (plus tard, chacun de ces sommets représentera un bit d'un mot binaire) et $c_0, c_1, \ldots c_{m-1}$ les éléments de C appelés les ***noeuds de parité***. Pour un sommet $s \in V \cup C$, on note $\Gamma(s)$ l'ensemble des voisins de s. Puisque G est biparti, si $s \in V$ alors $\Gamma(s) \subset C$ et si $s \in C$ alors $\Gamma(s) \subset V$.

Soit $x = (x_0, \ldots, x_{n-1}) \in \mathbb{F}_2^n$ un mot binaire et $c \in C$ un noeud de parité. On dit que c est ***satisfait*** par x si $\left\{ i \in [\![0; n-1]\!] : v_i \in \Gamma(c) \text{ et } x_i = 1 \right\}$ est de cardinal pair ; dans le cas contraire, on dit que c est ***non satisfait*** par x. On appelle ***syndrome*** de x et on note $\sigma(x) \in \mathbb{F}_2^m$ la suite de bits $\sigma(x) = (\sigma_0, \sigma_1, \ldots, \sigma_{m-1})$ où pour chaque $j \in [\![0; m-1]\!]$:

$$\begin{cases} \sigma_j = 0 \text{ si } c_j \text{ est satisfait par } x, \\ \sigma_j = 1 \text{ si } c_j \text{ est non satisfait par } x. \end{cases}$$

Les figures 1, 2, 3, 4 permettent d'illustrer ces définitions. Soit $G = (V \cup C, A)$ le graphe biparti de la figure 1 dans lequel $|V| = 7$ et $|C| = 3$. Alors :

$$\Gamma(v_4) = \{c_0, c_2\}, \qquad \Gamma(c_0) = \{v_0, v_1, v_2, v_4\}.$$

Pour dessiner la figure 2, on a considéré le mot binaire $x = (1, 0, 0, 0, 0, 0, 1)$ et le syndrome associé $\sigma(x) = (1, 0, 1)$. Le sommet v_i est grisé sur la figure lorsque $x_i = 1$ et le sommet c_j est grisé lorsque $\sigma_j = 1$. Un noeud de parité est donc grisé lorsqu'il est non satisfait, c'est à dire lorsque son nombre de voisins grisés est impair. De même :

→ Pour la figure 3 : $x = (0,1,0,1,0,1,0)$ et $\sigma(x) = (1,1,1)$.
→ Pour la figure 4 : $x = (0,0,0,0,1,0,1)$ et $\sigma(x) = (1,0,0)$.

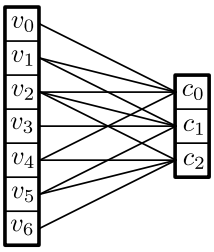


FIGURE 1

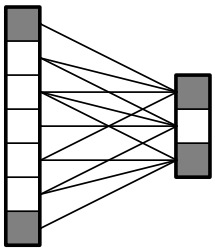


FIGURE 2

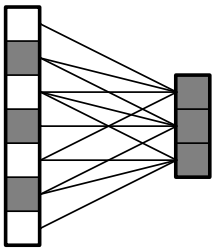


FIGURE 3

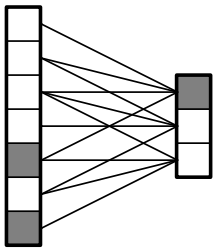


FIGURE 4

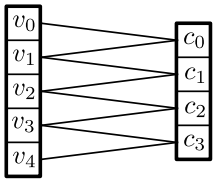


FIGURE 5

Soit $G = (V \cup C, A)$ un graphe biparti. On définit l'ensemble des mots de code associés à G par :

$$\mathcal{M} = \Big\{ x \in \mathbb{F}_2^n : \sigma(x) = (0,0,\ldots,0) \Big\}.$$

On parle alors de ***code correcteur linéaire*** et G est appelé le ***graphe de Tanner*** de $\mathcal{M}$. Par exemple, avec le graphe de la figure 1, on obtient :

$$\begin{aligned}\mathcal{M} = \{&(0,0,0,0,0,0,0); (0,0,0,1,0,1,1); (0,0,1,0,1,1,1); (0,0,1,1,1,0,0);\\ &(0,1,0,0,1,1,0); (0,1,0,1,1,0,1); (0,1,1,0,0,0,1); (0,1,1,1,0,1,0);\\ &(1,0,0,0,1,0,1); (1,0,0,1,1,1,0); (1,0,1,0,0,1,0); (1,0,1,1,0,0,1);\\ &(1,1,0,0,0,1,1); (1,1,0,1,0,0,0); (1,1,1,0,1,0,0); (1,1,1,1,1,1,1)\}\end{aligned}$$

Remarque. L'entier k et la fonction φ associés au code correcteur peuvent être obtenus à partir de G en utilisant des outils d'algèbre linéaire. Nous ne détaillerons pas cela ici.

Pour $(b_1, b_2) \in \mathbb{F}_2^2$, on note $b_1 \oplus b_2$ (lire "b_1 xor b_2") l'élément de $\mathbb{F}_2$ défini par :

$$0 \oplus 0 = 0, \qquad 0 \oplus 1 = 1, \qquad 1 \oplus 0 = 1, \qquad 1 \oplus 1 = 0.$$

L'opérateur $\oplus$ est donc commutatif et associatif puisqu'il s'agit simplement d'une somme modulo 2. On étend cette opération aux éléments de $\mathbb{F}_2^n$ par :

$$(a_0, a_1, \ldots, a_{n-1}) \oplus (b_0, b_1, \ldots, b_{n-1}) = (a_0 \oplus b_0, a_1 \oplus b_1, \ldots, a_{n-1} \oplus b_{n-1}).$$

7. Déterminer l'ensemble $\mathcal{M}$ dans le cas du graphe de Tanner de la figure 5. Justifier.
8. Soit G un graphe de Tanner et $(x, y) \in (\mathbb{F}_2^n)^2$. Montrer que :
$$\sigma(x \oplus y) = \sigma(x) \oplus \sigma(y).$$

En OCaml, un graphe de Tanner sera représenté par une variable « `g : tanner` » telle que `g.m` est le nombre de noeuds de parité et `g.adj` est un tableau de listes de taille n tel que `g.adj.(i)` est la liste des voisins du sommet v_i.

```
type tanner = {m: int; adj: int list array;};;
```

Par exemple, le graphe de Tanner de la figure 1 est représenté par :

```
{m = 3; adj = [|[0]; [0;1]; [0;1;2]; [1]; [0;2]; [1;2]; [2]|]}
```

9. (a) Écrire une fonction « `syndrome: tanner -> string -> int array` » qui calcule le syndrome du mot binaire donné en entrée. On pourra supposer sans le vérifier que la chaîne de caractères est de la bonne taille et composée uniquement de zéros et de uns. Votre fonction doit renvoyer un tableau dont les éléments appartiennent à $\{0, 1\}$.
   (b) En déduire une fonction « `liste_mots_code: tanner -> string list` » qui renvoie la liste de tous les mots de code du code correcteur associé au graphe de Tanner.

3.2 Algorithme de décodage

Pour $i \in [\![0; n-1]\!]$, on définit $e_i \in \mathbb{F}_2^n$ le mot binaire $e_i = (\delta_{i,0}, \delta_{i,1}, \ldots, \delta_{i,n-1})$ où $\delta_{i,j}$ est le symbole de Kronecker : $\delta_{i,j} = 1$ si $j = i$ et $\delta_{i,j} = 0$ sinon. Soit $x = (x_0, x_1, \ldots, x_{k-1}) \in \mathbb{F}_2^k$. On appelle ***poids*** de x et on note $\|x\|$ le nombre de composantes non-nulles de x : $\|x\| = \sum_{j=0}^{k-1} x_j$.

Dans cette partie, on s'intéresse à l'étape 3 de la partie 1 : étant donnée un mot binaire $y \in \mathbb{F}_2^n$ reçu par Bob, il s'agit de retrouver le mot de code $x \in \mathcal{M}$ qui a été envoyé par Alice. Pour cela, on propose d'utiliser l'algorithme dit "bit-flip". À chaque étape, cet algorithme flippe l'un des bits du mot binaire en s'assurant que cette opération fasse décroître le poids du syndrome. Notez que cet algorithme n'est performant que si le graphe de Tanner est bien choisi (voir par exemple la partie 4).

Algorithme "bit-flip" (Gallager, 1962)

Entrée : un mot binaire $y \in \mathbb{F}_2^n$.
Sortie : un mot binaire $\hat{x} \in \mathbb{F}_2^n$.

$\hat{x}_0 = y$; $k = 0$
tant que $\left[\exists i \in [\![0; n-1]\!] : \left\|\sigma(\hat{x}_k \oplus e_i)\right\| < \left\|\sigma(\hat{x}_k)\right\|\right]$ **faire**
  Choisir arbitrairement l'un de ces i.
  Soit $\hat{x}_{k+1} = \hat{x}_k \oplus e_i$
  $k \leftarrow k + 1$
fin tant que
renvoyer $\hat{x}_k$

---

10. Écrire une fonction « `bit_flip : tanner -> string -> string` » qui applique l'algorithme bit-flip au mot binaire donné en entrée.

4 Codes expanseurs

Dans cette partie, nous allons étudier un cas particulier de codes correcteurs pour lesquels l'algorithme bit-flip est performant. Soit $G = (V \cup C, A)$ un graphe biparti et $E \subset V \cup C$ un sous-ensemble des sommets de G. On appelle ***voisinage*** de E et on note $\Gamma_0(E)$ l'ensemble des sommets qui sont voisins d'au moins un sommet de E (Γ a été défini dans la partie 3.1) :

$$\Gamma_0(E) = \bigcup_{e \in E} \Gamma(e).$$

On appelle ***voisinage unique*** de E et on note $\Gamma_{\mathrm{u}}(E)$ l'ensemble des sommets qui sont voisins d'exactement un sommet de E :

$$\Gamma_{\mathrm{u}}(E) = \left\{ s \in \Gamma_0(E) : \text{ il existe un unique } e \in E \text{ tel que } s \in \Gamma(e) \right\}$$

On appelle ***voisinage multiple*** de E et on note $\Gamma_{\mathrm{m}}(E)$ l'ensemble des sommets de $\Gamma_0(E)$ qui sont voisins de plusieurs sommets de E :

$$\Gamma_{\mathrm{m}}(E) = \Gamma_0(E) \setminus \Gamma_{\mathrm{u}}(E) = \left\{ s \in \Gamma_0(E) : \exists (e, e') \in E^2, e \neq e', s \in \Gamma(e) \cap \Gamma(e') \right\}.$$

Soit $d_V \in \mathbb{N}^*$ un entier strictement positif et $\gamma, \delta \in]\,0\,;1\,]$. On dit que G est un ***graphe expanseur*** si tous les sommets de V ont pour degré d_V et :

$$\forall E \subset V : \left[|E| \leqslant \gamma |V| \Rightarrow |\Gamma_0(E)| \geqslant (1 - \delta) d_V |E| \right].$$

On admet l'existence de graphes expanseurs.

11. Soit G un graphe expanseur et $E \subset V$ tel que $|E| \leqslant \gamma|V|$. Montrer que :

$$|\Gamma_{\mathrm{u}}(E)| \geqslant d_V|E|(1-2\delta) \qquad \text{et} \qquad |\Gamma_{\mathrm{m}}(E)| \leqslant d_V|E|\delta.$$

Comme dans la partie 3.1, on note $(v_0, v_1, \ldots, v_{n-1})$ les éléments de V et $(c_0, c_1, \ldots c_{m-1})$ les éléments de C. Étant donné un mot binaire $x = (x_0, x_1, \ldots, x_{n-1}) \in \mathbb{F}_2^n$ (resp. $z = (z_0, z_1, \ldots, z_{m-1}) \in \mathbb{F}_2^m$), on appelle ***support*** de x (resp. ***support*** de z) et on note $supp(x)$ (resp. $supp(z)$) le sous-ensemble de V (resp. C) défini par :

$$supp(x) = \Big\{v_i : i \in [\![0, n-1]\!], x_i = 1\Big\}, \qquad supp(z) = \Big\{c_j : j \in [\![0, m-1]\!], z_j = 1\Big\}.$$

Soit $x \in \mathbb{F}_2^n$ un mot de code et $y \in \mathbb{F}_2^n$ le mot binaire obtenu lorsque x est envoyé via le canal binaire symétrique. On exécute l'algorithme bit-flip sur l'entrée y, et on note $f \in \mathbb{N}$ le nombre de tours de boucle. Soient $i_0, i_1, \ldots, i_{f-1}$ les indices sélectionnés à chaque tour de boucle par l'algorithme. En particulier, pour tout $k \in [\![0, f-1]\!]$:

$$\|\sigma(\hat{x}_k \oplus e_{i_k})\| < \|\sigma(\hat{x}_k)\|$$

On appelle ***support d'exécution*** l'ensemble $U = supp(x \oplus y) \cup \{v_{i_0}, v_{i_1}, \ldots, v_{i_{f-1}}\}$.

12. Supposons $\delta < \dfrac{1}{4}$.
    (a) Montrer que $f \leqslant \|\sigma(x \oplus y)\|$, puis que $|U| \leqslant \|x \oplus y\|(1 + d_V)$.
    (b) Soit $z \in \mathbb{F}_2^n$ un mot binaire tel que $0 < \|z\| \leqslant \gamma|V|$. Montrer qu'il existe $e \in \mathbb{F}_2^n$ tel que :

$$\|e\| = 1 \qquad \text{et} \qquad \|\sigma(z \oplus e)\| \leqslant \|\sigma(z)\| - d_V(1 - 4\delta).$$

    (c) En déduire que si $\|x \oplus y\| \leqslant \dfrac{\gamma}{1 + d_V}|V|$ alors l'algorithme bit-flip renvoie x.

Remarque. Étant donné que $e = x \oplus y$ est l'erreur introduite par le canal lors de la transmission, on a montré que si le poids de e est suffisamment petit (mais quand même linéaire en le nombre de bits), alors l'algorithme bit-flip parvient à corriger e.