Tableau de bord (d'après le concours Polytechnique)

La durée du devoir est de 3h. Les fonctions dont on demande une implémentation sont à écrire dans le langage OCaml. Toutes les fonctions de la bibliothèque standard sont autorisées. Cela inclue en particulier les fonctions de la forme `List.XXX`, incluant (mais non limitées à) celles rappelées ci-dessous.

`List.length : 'a list -> int`

Renvoie le nombre d'éléments dans la liste fournie en argument. Complexité linéaire en la taille de la liste ($O(n)$).

`List.hd : 'a list -> 'a`

Renvoie le premier élément de la liste fournie en argument. Complexité constante ($O(1)$).

`List.tl : 'a list -> 'a`

Renvoie la liste en argument privée de son premier élément. Complexité constante ($O(1)$).

`List.rev : 'a list -> 'a list`

Retourne une nouvelle liste contenant les éléments de la liste fournie en argument en ordre inverse. Complexité linéaire en la taille de la liste ($O(n)$).

`List.init : int -> (int -> 'a) -> 'a list`

« `List.init n f` » crée la liste `[f 0; f 1; ...; f (n-1)]` à `n` éléments obtenus en appelant la fonction `f` successivement avec les entiers de `0` à `n-1` (dans cet ordre).

`List.map : ('a -> 'b) -> 'a list -> 'b list`

« `List.map f lst` » renvoie la liste `[f`a_0`; f`a_1`; ...; f`a_{n-1}`]` où les a_i sont les éléments de la liste `lst`. L'ordre d'évaluation des $f\,a_i$ n'est pas spécifiée.

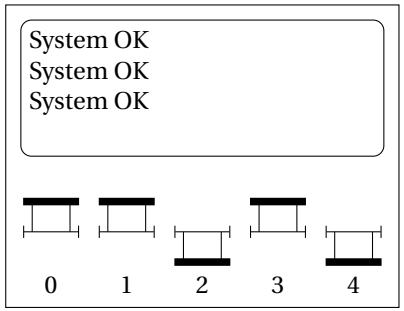
Toute fonction peut librement être réutilisée dans les questions suivantes, y compris les fonctions demandées par l'énoncé que vous n'êtes pas parvenus à écrire.

Il est demandé d'attacher une attention toute particulière à la clarté et la lisibilité des fonctions. On attend d'une fonction qu'elle soit juste, mais également qu'elle ne soit pas inutilement compliquée. Pour toute fonction de plus de quelques lignes, il convient d'en expliquer clairement le fonctionnement (ce que les noms désignent, ce que font les éventuelles fonctions auxilliaires et la signature de celles-ci, etc.). Il est recommandé de proposer des invariants de boucles chaque fois que cela fait sens.

1 Introduction

Nous considérons un système commandé par un tableau de bord comportant N interrupteurs, chacun pouvant être baissé ou levé. On désire tester ce système (pour le valider ou pour effectuer une opération de maintenance) en essayant mécaniquement chacune des 2N configurations possibles pour l'ensemble des interrupteurs.

Le coût de cette opération, qu'elle soit réalisée par un opérateur humain ou par un robot, sera le nombre total de mouvements d'interrupteurs nécessaires. Nous supposons que chaque fois qu'un interrupteur est commuté le système effectue un diagnostic automatiquement et instantanément. Les interrupteurs sont indexés de 0 à N – 1.



2 Parties de $\mathbb{N}$

Nous appelons *partie* un sous-ensemble fini de l'ensemble $\mathbb{N}$ des entiers naturels. Un élément d'une partie est appelé *indice*. La *différence symétrique* de deux parties P et Q est définie par :

$$P \,\Delta\, Q = (P \setminus Q) \cup (Q \setminus P) = (P \cup Q) \setminus (P \cap Q)$$

On vérifie facilement que la différence symétrique est commutative et associative, ce que l'on ne demande pas de démontrer. Pour tout entier n positif ou nul, nous notons $I_n = \{0, \ldots, n-1\}$ l'ensemble des entiers naturels inférieurs strictement à n.

Une partie sera représentée par une liste d'indices distincts **apparaissant dans l'ordre croissant des entiers**. On utilisera les types suivants :

```
type indice == int
type partie == indice list
```

1. Proposer une fonction `valid`, de signature `partie -> bool`, prenant en argument de type `partie` et renvoyant un booléen indiquant s'il valide toutes les hypothèses (indices

positifs, distincts, rangés par ordre croissant).

Dans la suite, tous les objets de type `partie` fournis en argument sont réputés vérifier ces hypothèses (il n'est pas demandé de le contrôler, tout comportement de la fonction est admissible si ce n'est pas le cas), et tous les objets de type `partie` retournés par vos fonctions devront les vérifier également.

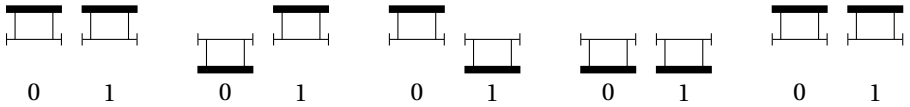
2. Écrire la fonction `delta`, de signature `partie -> partie -> partie` qui réalise la différence symétrique de 2 parties. Le nombre d'opérations ne devra pas excéder $O(m+n)$, où m et n sont les cardinaux des arguments. Nous rappelons que dans toute liste de type `partie` les indices sont distincts et doivent apparaître dans l'ordre croissant des entiers.

3 Énumération des parties par incrément

À toute partie P, nous associons l'entier $e(\mathrm{P}) = \sum_{i\in \mathrm{P}} 2^i$ (avec la convention $e(\emptyset) = 0$). Nous définissons le *successeur* de P comme l'unique partie dont l'entier associé est $(e(\mathrm{P})+1)$. Par exemple, le successeur de la partie P = `[0; 1; 3]` ($e(\mathrm{P}) = 2^0+2^1+2^3 = 11$) est la partie `[2; 3]` ($e(\mathrm{P}) = 2^2+2^3 = 12$).

3. Écrire la fonction `succ`, de signature `partie -> partie` qui prend en argument une partie et renvoie le successeur de la partie. Le nombre d'opérations ne devra pas excéder $O(l)$ où l est le plus petit indice absent dans la partie donnée en argument.

Dans le cadre du problème des interrupteurs, on associe à chacune des configurations possibles la partie formée des indices des interrupteurs baissés. En application de ce mode d'énumération des parties, nous voulons réaliser le test de toutes les configurations d'interrupteurs. Au début et à la fin du test tous les interrupteurs seront levés. Par exemple, pour N = 2, on considère la séquence suivante :



4. Proposer une fonction `sequence` de signature `int -> partie list` prenant en argument le nombre d'interrupteurs et renvoyant la liste des listes des indices des interrupteurs à commuter pour réaliser la totalité du test pour les N interrupteurs et qui examine les configurations dans l'ordre défini par le successeur. Par exemple, pour N = 2, le résultat attendu sera `[[0]; [0; 1]; [0]; [0; 1]]` : en partant de tous les interrupteurs levés, on commence par abaisser l'interrupteur 0 pour obtenir la première configuration, puis on remonte l'interrupteur 0 tout en baissant l'interrupteur 1 pour obtenir la seconde configuration, puis on baisse à nouveau l'interrupteur 0 pour obtenir la troisième configuration, puis on relève les deux interrupteurs pour revenir à la configuration où tous les interrupteurs sont levé.

5. Exprimer, en fonction de N, le nombre total d'interrupteurs à commuter pour réaliser le test de cette manière.

4 Énumération des parties par un code de Gray

Nous notons $\langle u_0,\dots,u_{l-1}\rangle$ une suite finie de l entiers. La *concaténation* de deux suites finies de longueur l et l' respectivement est une suite finie de longueur $l+l'$ définie par $\langle u_0,\dots,u_{l-1}\rangle \cdot \langle u'_0,\dots,u'_{l'-1}\rangle = \langle u_0,\dots,u_{l-1},u'_0,\dots,u'_{l'+1}\rangle$.

La suite vide, notée $\langle\rangle$, est la suite de longueur 0. Une suite finie U est préfixe d'une autre suite finie V s'il existe une suite finie W telle que $\mathrm{V} = \mathrm{U}\cdot\mathrm{W}$ (autrement dit U est le début de V).

Pour tout entier positif ou nul n, nous considérons la suite finie $\mathrm{T}(n)$ de longueur $2n-1$ définie par $\mathrm{T}(0) = \langle\rangle$ et $\mathrm{T}(n+1) = \mathrm{T}(n)\cdot\langle n\rangle\cdot\mathrm{T}(n)$. Nous avons par exemple $\mathrm{T}(1) = [0]$, $\mathrm{T}(1) = [0,1,0]$ et $\mathrm{T}(3) = [0,1,0,2,0,1,0]$.

Pour tout entier i positif ou nul nous notons t_i le (i + 1)$^{\text{e}}$ élément de $\mathrm{T}(n)$ s'il existe. Puisque $\mathrm{T}(n)$ est préfixe de $\mathrm{T}(n+1)$, la suite $(t_i)_{i\geqslant 0}$ est définie sans ambiguïté.

Enfin, nous posons $\mathrm{S}_0 = \emptyset$ et pour tout entier i positif ou nul nous définissons l'ensemble $\mathrm{S}_{i+1} = \mathrm{S}_i \ \Delta\ \{t_i\}$.

6. Donner la valeur de T(4).

7. Donner la valeur de S_i pour tout i inférieur ou égal à 15.

Nous voulons montrer que les S_i peuvent être utilisés pour énumérer les parties de I_n, et ainsi résoudre notre problème d'interrupteurs.

8. Donner la valeur de S_{2^n-1} pour tout $n \geqslant 0$.

9. Montrer que pour tout $n > 0$ et tout $i < 2^n$, on a $\mathrm{S}_{2^n+i} = \mathrm{S}_i \ \Delta\ \{n-1, n\}$.

10. En déduire que pour tout $n \geqslant 0$ l'ensemble $\mathrm{P}_n = \{\mathrm{S}_0, \mathrm{S}_1, \dots, \mathrm{S}_{2^n-1}\}$ est l'ensemble des parties de I_n.

Revenons au problème des interrupteurs. Comme dans la section précédente, nous imposons que les interrupteurs soient levés au début et à la fin du test.

11. En s'inspirant des résultats de cette partie, proposer une fonction `sequence_gray` de signature `int -> partie list` prenant en argument le nombre d'interrupteurs et renvoyant la liste des listes des indices des interrupteurs à commuter pour réaliser la totalité du test pour les N interrupteurs.

12. Quel est le coût du test avec cette méthode (c'est-à-dire le nombre total d'interrupteurs à commuter) ? Peut-on réaliser le test à un coût moindre ?

Pour tout $i > 0$, on note $\min(\mathrm{S}_i)$ le plus petit élément de S_i.

13. Donner une expression de t_i en fonction de S_i pour i impair.

14. En déduire une fonction `succ_gray` de signature `partie -> partie` qui prend en argument une partie et retourne celle qui la suit immédiatement dans l'ordre défini par la suite $(\mathrm{S}_i)_{i\geqslant 0}$.

5 Système défaillant

Chaque interrupteur baissé active une composante du système, et un mauvais fonctionnement de l'alimentation électrique provoque une défaillance dès que plus de K interrupteurs sur les N sont baissés.

15. Écrire une fonction `test` de signature `partie -> partie -> indice liste` qui renvoie une liste d'interrupteurs à commuter, de taille minimale, permettant de passer d'une configuration non défaillante à une autre sans provoquer de défaillance. Les arguments de ce programme seront la partie de départ et la partie cible. Par exemple, pour K = 4, on peut passer de `[1; 3; 4]` à `[0; 4; 5]` en commutant successivement les interrupteurs 1, 0, 5 et 3 (à aucun moment il n'y a strictement plus de 4 interrupteurs baissés). `[1; 0; 5; 3]` est donc un résultat acceptable (mais ce n'est pas le seul). On remarquera que la liste des interrupteurs à commuter n'est pas nécessairement une partie (dans notre exemple, les indices ne sont pas par ordre croissant).

L'*inverse* d'une suite finie T est obtenue en prenant ses éléments dans l'ordre inverse, nous la notons $\tilde{\mathrm{T}}$. Soit $\mathrm{T}(n,k)$ la suite définie pour tout $k \geqslant 1$ et pour tout $n \geqslant 1$ par

$$\begin{cases} \mathrm{T}(1,k) = \langle 0 \rangle \\ \mathrm{T}(n+1,1) = \mathrm{T}(n,1) \cdot \langle n-1, n \rangle \\ \mathrm{T}(n+1,k+1) = \mathrm{T}(n,k+1) \cdot \langle n \rangle \cdot \tilde{\mathrm{T}}(n,k) \end{cases}$$

16. Proposer une fonction `t` de signature `int -> int -> int list` prenant en arguments n, k et renvoyant la liste $\mathrm{T}(n,k)$.

Soit k un entier strictement positif. Pour tout entier i positif ou nul nous notons $t_{k,i}$ le $(i+1)^{\mathrm{e}}$ élément de $\mathrm{T}(n,k)$ s'il existe. Puisque $\mathrm{T}(n,k)$ est préfixe de $\mathrm{T}(n+1,k)$, la suite $\left(t_{k,i}\right)_{i \geqslant 0}$ est définie sans ambiguïté.

Enfin, nous posons $\mathrm{S}_{k,0} = \emptyset$ et pour tout entier i positif ou nul nous définissons l'ensemble $\mathrm{S}_{k,i+1} = \mathrm{S}_{k,i} \ \Delta \ \left\{t_{k,i}\right\}$.

17. Exprimer la longueur $l_{n,k}$ de $\mathrm{T}(n,k)$ en fonction de n, de k et du nombre $s_{n,k}$ de parties de I_n de cardinal inférieur ou égal à k.

18. Montrer que pour tout $k \geqslant 1$ et tout $n \geqslant 1$ l'ensemble $\mathrm{P}_{n,k} = \left\{\mathrm{S}_{k,0}, \mathrm{S}_{k,1}, \ldots, \mathrm{S}_{k,l_{n,k}}\right\}$ est l'ensemble des parties de I_n de cardinal inférieur ou égal à k.

19. Écrire un programme `test_panne` de signature `int -> int -> indice list` qui renvoie une liste de $l_{\mathrm{N,K}} + 1$ interrupteurs à commuter permettant de vérifier toutes les configurations non défaillantes sans provoquer de défaillance, en commençant et en finissant avec des interrupteurs tous levés. Les entiers N et K sont donnés en arguments.

20. Montrer que le coût d'un test commençant et en finissant avec des interrupteurs tous levés et ne provoquant pas de défaillance ne peut pas être inférieur à $l_{\mathrm{N,K}} + 1$.