Concours Centrale-Supélec, France

Session 2019, filière MP

Option informatique

Durée : 4 heures

Ce que traite le sujet

- Partie I : code binaire de Gray
- Partie II : énumération des combinaisons

Sujet reproduit sans retouche à partir de la page suivante.

Résumé des parties, chapitres de cours liés et autres années :
prepahelp.com/annales/centrale/mp/2019/info

Prepa Help ne revendique aucun droit sur le sujet, propriété de l'organisateur du concours.

Option informatique

MP

2019

CONCOURS CENTRALE•SUPÉLEC

4 heures

Calculatrice autorisée

Les candidat devront utiliser le langage Caml, à l'exclusion de tout autre, pour traiter ce sujet. Ils devront donner le type de chaque fonction écrite.

Dans tout le problème, n désigne un entier naturel non nul.

Si $x \in \{0;1\}$, $\bar{x}$ désigne $1-x$.

On note $\oplus$ l'opérateur *ou exclusif* (également appelé *XOR*). Il est défini sur $\{0;1\}$ par la table suivante :

	0	**1**
0	0	1
1	1	0

On prolonge cette définition à $\mathbb{N}$ de la manière suivante.

Soit $(x,y) \in \mathbb{N}^2$ vérifiant $x < 2^p$ et $y < 2^p$ où p est un entier naturel non nul. On décompose x et y en base 2 :

$$x = \sum_{k=0}^{p} a_k 2^k \qquad \text{et} \qquad y = \sum_{k=0}^{p} b_k 2^k,$$

où les coefficients a_k et b_k sont des éléments de $\{0;1\}$. On définit alors $x \oplus y$ par :

$$x \oplus y = \sum_{k=0}^{p} (a_k \oplus b_k) 2^k.$$

I Code binaire de Gray

On cherche à obtenir tous les n-uplets composés de 0 et 1 de deux manières différentes.

Ces n-uplets seront implémentés à l'aide de listes.

I.A – Dans cette sous-partie, on obtient ces n-uplets dans l'ordre lexicographique.

Q 1. Écrire une fonction `suivant` transformant un n-uplet en son suivant dans l'ordre lexicographique. Par exemple, si $t = (0;1;0;0;1;1;1)$, après exécution de `suivant(t)`, on a $t = (0;1;0;1;0;0;0)$.

Cette fonction modifie la liste qu'elle reçoit en argument. De plus elle renvoie un booléen, valant `vrai` si elle a pu déterminer un n-uplet suivant, ou bien `faux` si le n-uplet fourni en argument était le dernier. Dans ce dernier cas, la valeur de ce n-uplet après exécution de la fonction est non spécifiée.

Q 2. Écrire une fonction affichant tous les n-uplets dans l'ordre lexicographique.

On pourra commencer par écrire une fonction affichant les éléments d'un n-uplet :

```
affiche_nuplet : int list -> unit
```

I.B – Pour certaines applications, par exemple pour éviter des états transitoires intermédiaires dans les circuits logiques ou pour faciliter la correction d'erreur dans les transmissions numériques, on souhaite que le passage d'un n-uplet au suivant ne modifie qu'un seul bit. Dans un brevet de 1953, Frank Gray[1] définit un ordre des n-uplets possédant cette propriété.

Pour produire la liste des n-uplets dans l'ordre de Gray, un algorithme consiste à partir de la liste des 1-uplets $(0,\ 1)$ et à construire la liste des $(n+1)$-uplets à partir de celles de n-uplets en ajoutant un 0 en tête de chaque n-uplet puis un 1 en tête de chaque n-uplet en parcourant leur liste à l'envers. On obtient ainsi pour $n = 2$, la liste $(00,\ 01,\ 11,\ 10)$, pour $n = 3$, la liste $(000,\ 001,\ 011,\ 010,\ 110,\ 111,\ 101,\ 100)$, etc.

Les n-uplets sont représentés par des listes, une liste de n-uplets sera donc une liste de listes.

Q 3. Écrire une fonction `ajout` telle que si a est un entier et l une liste de n-uplets d'entiers, `ajout a l` renvoie une liste contenant les éléments de l auxquels on a ajouté a en tête.

[1] Frank Gray (1887-1969) était un chercheur travaillant chez Bell Labs ; il a en particulier produit de nombreuses innovations dans le domaine de la télévision.

BY-NC-SA

Q 4. Écrire deux fonctions mutuellement récursives `monte` et `descend` prenant en argument un entier n et renvoyant les n-uplets dans l'ordre de Gray. L'une les renverra de 00...0 à 10...0, l'autre en sens inverse.

Q 5. Évaluer la complexité de ces fonctions en termes d'appels à `monte` et `descend`.

Q 6. Décrire une façon simple d'améliorer cette complexité.

I.C – Dans tout ce qui suit, l'expression *représentation binaire*, désigne la représentation traditionnelle en base 2. Étant donné $k \in \mathbb{N}^*$, on a de manière unique

$$k = \sum_{i=0}^{p} a_i 2^i \qquad \text{avec} \qquad p \in \mathbb{N}, \quad a_i \in \{0;1\}, \quad a_p \neq 0$$

La représentation binaire canonique de k est $a_p \cdots a_0$ (le bit de poids fort, qui est non nul, en premier). On dira que tout n-uplet constitué d'un nombre quelconque de 0 suivis de la représentation binaire canonique de k est une représentation binaire de k.

On définit une fonction g sur $\mathbb{N}$ de la manière suivante. Pour $k \in \mathbb{N}$ et n tel que $k < 2^n$, on considère la liste dans l'ordre de Gray des 2^n n-uplets ; on indice cette liste de 0 à $2^n - 1$; $g(k)$ est le nombre dont une représentation binaire est le n-uplet d'indice k.

Par exemple, si on énumère les 3-uplets selon l'ordre de Gray, on a (000, 001, 011, 010, 110, 111, 101, 100). Dans cette liste, l'élément d'indice 0 est 000 donc $g(0) = 0$, le 3-uplet d'indice 7 est 100 donc $g(7) = 4$.

Soit n un entier naturel et k un entier compris entre 2^n et $2^{n+1} - 1$: $k = 2^n + r$ avec $0 \leqslant r < 2^n$.

Q 7. Démontrer que $g(k) = 2^n + g(2^n - 1 - r)$.

Q 8. En déduire que, si la représentation binaire de k est $b_n \cdots b_0$ et si on pose $b_{n+1} = 0$, la représentation binaire de $g(k)$ est $a_n \cdots a_0$ où, pour tout j entre 0 et n, $a_j = b_j \oplus b_{j+1}$.

Q 9. Exprimer, pour $k \in \mathbb{N}$, $g(k)$ en fonction de k (à l'aide de $\oplus$).

Q 10. Montrer que g est une bijection de $\mathbb{N}$ dans $\mathbb{N}$ et, avec les notations précédentes, exprimer b_j en fonction de a_k, $k \geqslant j$.

I.D – On cherche à réaliser les fonctions g et g^{-1} avec des circuits logiques. Une porte logique sera représentée par un rectangle contenant son nom (AND, OR, XOR, NOT) et on indiquera les entrées et sorties par des flèches. Par exemple :



On se place d'abord dans le cas $n = 3$.

Q 11. Donner un circuit logique à 3 entrées représentant les trois bits d'un entier k inférieur ou égal à 7 et à trois sorties représentant les trois bits de $g(k)$.

On pourra utiliser la porte logique XOR.

Q 12. Donner un circuit pour l'opération inverse.

Q 13. Si $n \geqslant 2$, donner un circuit permettant de passer d'un nombre k à n bits à $g(k)$. Faire de même pour l'opération inverse en utilisant le moins possible de portes. Préciser le nombre de portes utilisées.

II Enumération des combinaisons

On souhaite maintenant parcourir toutes les combinaisons de p éléments pris dans un ensemble de cardinal n avec $0 \leqslant p \leqslant n$. On supposera que cet ensemble est celui des n premiers entiers naturels. On le note E_n ; $E_n = \{0, ..., n-1\}$. Les éléments d'une combinaison de E_n seront systématiquement considérés dans l'ordre croissant.

II.A – On se place dans le cas $p = 3$.

Q 14. Écrire une fonction d'argument n affichant les combinaisons de 3 éléments pris dans $\{0, ..., n-1\}$; on souhaite que ces combinaisons apparaissent dans l'ordre lexicographique. Par exemple pour $n = 4$: $[0, 1, 2]$, $[0, 1, 3]$, $[0, 2, 3]$, $[1, 2, 3]$.

II.B – On revient au cas p entier quelconque entre 1 et n.

Q 15. Donner la première et la dernière combinaison de p éléments de E_n ($p \leqslant n$) lorsqu'on énumère ces combinaisons dans l'ordre lexicographique.

On suppose que $c_0 \cdots c_{p-1}$ est une combinaison de E_n vérifiant $c_0 < \cdots < c_{p-1}$, stockée sous la forme d'un vecteur ou d'un tableau ; on suppose également que cette combinaison n'est pas la dernière.

Q 16. Écrire une fonction `comb_suivante` permettant de transformer une combinaison en la combinaison suivante dans l'ordre lexicographique.

On pourra commencer par chercher le plus petit indice j tel que $c_{j+1} > c_j + 1$.

BY-NC-SA

Q 17. En déduire une fonction d'arguments n et p, énumérant et affichant toutes les combinaisons de p éléments de E_n dans l'ordre lexicographique.

Q 18. Déterminer le nombre de combinaisons affichées avant d'arriver à la combinaison $c_0 \cdots c_{p-1}$.

Q 19. En déduire que tout nombre entier naturel non nul N peut s'écrire sous la forme

$$N = \sum_{k=1}^{p} \binom{n_k}{k}$$

où $0 \leqslant n_1 < n_2 < \cdots < n_p$. On dira qu'il s'agit d'une décomposition combinatoire de degré p pour l'entier N.

Q 20. Démontrer que si m et p sont deux entiers naturels tels que $p \leqslant m$, alors

$$\sum_{k=0}^{p-1} \binom{m-k}{p-k} = \binom{m+1}{p} - 1$$

On pourra commencer par vérifier que, pour k entre 0 et $p-1$, $\binom{m-k}{p-k} = \binom{m-k+1}{m-p+1} - \binom{m-k}{m-p+1}$.

Q 21. En déduire l'unicité, pour $N \in \mathbb{N}$, d'une décomposition combinatoire de degré p pour N.

II.C – On suppose que l'on dispose de n ($n \geqslant 3$) objets $o_0, ..., o_{n-1}$ et d'un sac dont la charge maximale est M. Chaque objet o_i a une valeur v_i et une masse m_i. Soit p un entier entre 1 et $n-2$. On veut choisir p objets parmi les n de façon à ce que la masse totale de ces p objets soit inférieure à M et leur valeur totale soit la plus grande possible.

Q 22. De quelle manière peut-on utiliser la fonction ci-dessus pour résoudre ce problème ?

Q 23. Évaluer le nombre d'additions effectuées (on ne s'intéressera qu'aux additions entre m_i et entre v_i).

II.D – Pour améliorer cet algorithme, on souhaite passer d'une combinaison à une autre en ne faisant que deux modifications.

Q 24. Expliquer comment représenter une combinaison de p éléments pris parmi n par un n-uplet de 0 et 1.

Q 25. Modifier les fonctions `monte` et `descend` de la question 4 pour qu'elles renvoient les n-uplets représentant les combinaisons de p éléments pris parmi n, dans le même ordre que précédemment.

Q 26. Déterminer le premier et le dernier n-uplet renvoyé par l'appel `monte n p`.

Q 27. Montrer qu'entre deux éléments successifs de la liste de n-uplets renvoyés par `monte` ou `descend`, seuls deux bits changent. Montrer que cette propriété est également vraie entre le dernier et le premier n-uplets.

Soit a l'un de nos n-uplets, et soit a' son successeur. On admet qu'il existe un unique entier j, $1 \leqslant j \leqslant \max(p, n-p)$ tel que $g^{-1}(a) - g^{-1}(a') \equiv 2^j \bmod 2^n$

Q 28. En déduire un algorithme simple pour passer d'un n-uplet contenant p bits à 1 au suivant. Le décrire et le programmer sous la forme d'une fonction `suivant`.

II.E –

Q 29. En déduire un algorithme qui donne un remplissage optimal du sac. Cet algorithme aura comme entrée :

— les valeurs des n objets,

— les masses des n objets, dans le même ordre,

— l'entier p,

— la masse maximum M.

Il rendra en résultat les p indices des objets choisis.

On ne demande pas la programmation explicite de cet algorithme.

• • • FIN • • •

BY-NC-SA

Travailler les chapitres de ce sujet

Prepa Help suit le programme de MP chapitre par chapitre : cours détaillé, méthodes, pièges, exercices et problèmes corrigés pas à pas, tirés des concours. L'essai gratuit donne accès aux chapitres offerts de ta filière, sans carte bancaire.

Chapitres liés à ce sujet

- Langage OCaml : expressions, types, filtrage, récursivité
- Récursivité
- Architecture de l'ordinateur et représentation des nombres
- Complexité des algorithmes itératifs et récursifs
- Calcul algébrique : sommes, produits, binôme et systèmes linéaires

Ouvrir l'essai gratuit

Scanne le code ou clique sur le bouton.

prepahelp.com/#essai

Une question sur le concours ou sur ta filière : **écrire sur WhatsApp**

Les autres années et épreuves : prepahelp.com/annales/centrale/mp