M2 CCI – Algorithmique – Devoir surveillé

Durée 2h, sans documents **30 janvier 2024**

NE PAS RECOPIER les énoncés des questions. Ne pas perdre de temps à un soin excessif de la présentation. Les questions sont indépendantes. Le barème est indicatif.

1. À propos de combinaisons

On veut produire toutes les combinaisons de deux entiers pris parmi ceux d'une séquence **S non vide** d'entiers **distincts deux à deux**. Le résultat est une séquence **C** de couples <e1, e2> tels que : e1 ≠ e2 et si <e1, e2> ∈ **C** alors <e2, e1> ∉ **C**. Aucun ordre n'est imposé entre les couples du résultat.

Par exemple pour S = [5, 3, 10, 7], C = [<5,3>, <5, 10>, <5, 7>, <3, 10>, <3, 7>, <10, 7>].

Q1 **[4 points]**

On définit deux fonctions :

CoupleE : type <E1 : entier, E2 : entier>

LesCouples : fonction (x : entier, S : séquence non vide d'entier)
→ séquence non vide de CoupleE
{Séquence formée de tous les couples comportant x et un élément de S.
Par exemple : LesCouples (10, [2, 13, 4]) = [<10,2>, <10,13>, <10,4>].}

LesComb : fonction (S : séquence non vide d'entier) → séquence de CoupleE
{Séquence formée des combinaisons de 2 entiers de S (cf ci-dessus).}

— **Donner des équations de récurrence** définissant la fonction **LesCouples**
— **Donner des équations de récurrence** définissant **LesComb** en utilisant **LesCouples**.

2. Existence d'un chemin dans un arbre binaire

Etant donné un arbre binaire **A**, la *valeur d'un chemin* de **A** est la séquence $[\mathbf{e_1, e_2, \ldots, e_n}]$ (n≥1) des étiquettes des nœuds du chemin. Dans cette séquence, l'ordre correspond à la relation père-fils : $\forall \mathbf{i} \in [\mathbf{1 \ldots n\text{-}1}]$, $\mathbf{e_{i+1}}$ est fils de $\mathbf{e_i}$.

On donne une séquence **non vide** d'entiers **S**, et un arbre binaire d'entiers **A**. On veut répondre à la question : *existe-t-il dans **A** un chemin de valeur **S** ?* Pour cela, on étudie une question intermédiaire : *existe-t-il dans **A** un chemin **issu de la racine** (dont le premier nœud est la racine) et de valeur **S** ?*

Q2 [5 points]

On donne le lexique suivant :

ExisteCh : fonction (A : arbre binaire d'entier, S : séquence non vide d'entier) → booléen
{vrai ⇔ il existe dans A un chemin de valeur S.}

ExisteChRac : fonction (A : arbre binaire d'entier, S : séquence non vide d'entier) → booléen
*{vrai ⇔ il existe dans A un chemin **issu de la racine** et de valeur S.}*

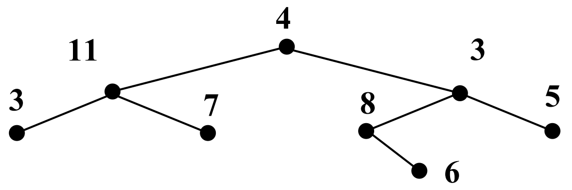
— Donner des **équations de récurrence** définissant la fonction **ExisteCh** en utilisant la fonction **ExisteChRac**.
— Donner des **équations de récurrence** définissant la fonction **ExisteChRac**.

3. Les préfixes de poids P dans un arbre

Q3 [11 points]

Étant donné un arbre binaire d'entiers **A**, le *poids* d'un chemin de **A** est la somme des éléments de ce chemin ; un chemin de **A** ne commence pas nécessairement à la racine de **A**. Un *préfixe* de **A** est un chemin de **A** dont le premier élément est la racine de **A**.

Exemple : soit **A** l'arbre ci-dessous :



Un **préfixe** de **A** de poids **15** est : **[4, 11]**
Les **préfixes** de **A** de poids **15** sont : **[[4, 11], [4, 3, 8]]**
Il y a 2 **chemins** de **A** de poids **14** : **[11, 3]** et **[8, 6]**

On définit :

EntPos : type entier > 0
ArbPos : type arbre binaire d'EntPos

Indication : Si **Z** est un préfixe de poids **P** de l'un des fils de **A,** et, si **A** est de racine **r**, alors $\mathbf{r_o Z}$ est un préfixe de **A** de poids **P+r**.

(i) Nombre de préfixes de poids P

— Donner des **équations de récurrence** définissant :

NbPrefP : fonction (A : ArbPos, P : EntPos) → entier ≥ 0
{Nombre de préfixes de poids P dans A.}

(ii) Nombre de chemins de poids P

— Donner des **équations de récurrence**, utilisant la fonction **NbPrefP** et définissant :

NbChP : fonction (A : ArbPos, P : EntPos) → entier ≥ 0
{Nombre de chemins de poids P dans A.}

(iii) Construction d'un préfixe de poids P

— Donner des **équations de récurrence** définissant :

UnPrefP : fonction (A : ArbPos, P : EntPos) → séquence d'EntPos
{Construit un préfixe de poids P dans A ou la séquence vide s'il n'y en a pas.}

(iv) Construction de tous les préfixes de poids P

On définit la fonction suivante :

PlusG : fonction (e **:** EntPos, SS : séquence de séquence d'EntPos)
→ séquence de séquence d'EntPos

{Séquence de séquences obtenue à partir de SS en ajoutant e à gauche de toutes les séquences de SS. ***Attention, on ne veut ajouter e à gauche que si SS n'est pas vide, donc PlusG(e, []) = []****.}*

— Donner des **équations de récurrence** définissant **PlusG**.

— Donner des **équations de récurrence** définissant :

LesPrefP : fonction (A : ArbPos, P : EntPos) → séquence de séquence d'EntPos
{Construit la séquence de tous les préfixes de poids P dans A ou la séquence vide s'il n'y en a pas.}

M2 CCI – Algorithmique

AL – DS 3 : des exemples de solutions **30 janvier 2024**

1. À propos de combinaisons

Q1 [4 points]

(1) LesCouples(e1, [e2]) = [<e1,e2>]
(2) LesCouples(e1, e2$_0$S) = <e1, e2>$_0$LesCouples(e1, S) *{S non vide}*

(1) LesComb([e]) = []
(2) LesComb(e$_o$S) = LesCouples(e, S) & LesComb(S) *{S non vide}*

2. Existence d'un chemin dans un arbre binaire

Q2 [5 points]

ExisteCh(/\, S) = faux *{S non vide}*
ExisteCh(/G, r, D\, S) = *{S non vide}*
    ExisteChRac(/G, r, D\, S) ou ExisteCh(G, S) ou ExisteCh(D, S)

ExisteChRac (/\, S) = faux *{S non vide}*
ExisteChRac (/G, r, D\, [e]) = (r=e)
ExisteChRac (/G, r, D\, e$_o$S) = *{S non vide}*
    (r=e) et (ExisteChRac(G, S) ou ExisteChRac(D, S))

3. Les préfixes de poids P dans un arbre

Q3 [11 points

(i) Nombre de préfixes de poids P [2.5 points]

(1) NbPréfP (/ \, P) = 0
(2) NbPréfP (/G, r, D\, P) =
    soit dif = P - r dans
        selon dif
            dif < 0 : 0
            dif = 0 : 1
            dif > 0 : NbPréfP(G, dif) + NbPréfP(D, dif)

(ii) Nombre de chemins de poids P [1.5 points]

(1) NbChP (/ \, P) = 0
(2) NbChP (/G, r, D\, P) = NbPrefP(/G, r, D\, P) + NbChP(G, P) + NbChP(D, P)

(iii) Construction d'un préfixe de poids P [3 points]

(1) UnPrefP(/ \, P) = [ ]
(2) UnPrefP(/G, r, D\, P) =
    soit dif = P - r dans
        selon dif
            dif < 0 : [ ]
            dif = 0 : [r]
            dif > 0 : soit C=UnPrefP(G, dif)
                dans si C≠[ ] alors r$_o$C
                    sinon
                        soit C=UnPrefP(D, dif)
                        dans si C≠[ ] alors r$_o$C
                            sinon []

(iv) Construction de tous les préfixes de poids P

[1 point]

(1) PlusG(e, []) = []
(2) PlusG(e, s_oS) = (e_os)$_o$PlusG(e, S)

[3 points]

(1) LesPrefP(/ \, P) = []
(2) LesPrefP(/G, r, D\, P) =
soit dif = P - r dans
selon dif
dif < 0 : []
dif = 0 : [[r]]
dif > 0 : PlusG(r, LesPrefP(G, dif)) & PlusG(r, LesPrefP(D, dif))