Algorithmique avancée et programmation C

Durée : *1h15*
Documents autorisés : **AUCUN**

Remarques :
— Veuillez lire attentivement les questions avant de répondre ;
— Rendez une copie propre ;
— N'utilisez pas de crayon à papier sur votre copie ;
— Pour chaque section comportant des questions, il est indiqué, entre parenthèses, le nombre de compétences évaluées ;
— Vous trouverez en annexe les signatures de fonctions et procédures sur des collections, SDD nécessaires à certains exercices.

1 QCM (3 compétences)

Répondez au QCM ci joint (à rendre avec votre copie). La note de chaque question peut varier de -1 à $+1$.

Soit B le nombre de bonnes réponses à une question et soit M le nombre de mauvaises réponses à cette même question ($B+M$ est égal au nombre de réponses à la question). Chaque bonne réponse cochée rapporte $1/B$ points et chaque mauvaise réponse $-1/M$ points.

Solution proposée :

Attendus d'apprentissages disciplinaires évalués

— DEV001 : Compiler et linker un programme C (options de base de gcc)
— DEV008 : Traduire des passages de paramètre algorithme en passage de paramètre C
— DEV009 : Utiliser les pointeurs, tableaux et chaînes de caractères en C

2 Les arbres de Fenwick

Soit une suite de valeurs $v_i, i \in 1..n$ qui évoluent dans le temps. Dans de nombreux domaines, il est nécessaire de calculer la somme des m premières valeurs (nommée somme préfixes). Avec un algorithme naïf cette opération est en $O(n)$.

« Un arbre de Fenwick est une structure de données qui permet de mettre à jour efficacement des valeurs et de calculer des sommes préfixes dans un tableau de valeurs.

Cette structure a été proposée par Boris Ryabko en 1989, avec une modification supplémentaire publiée en 1992. Elle est devenue connue sous le nom d'arbre de Fenwick après que Peter Fenwick l'ait décrite dans un article de 1994.

Comparé à un tableau plat de valeurs, l'arbre de Fenwick réalise un bien meilleur équilibre entre deux opérations : la mise à jour des valeurs et le calcul des sommes préfixes. » (Wikipédia)

2.1 Principe

Un arbre de Fenwick est un arbre binaire équilibré, tel que les nœuds ont soit 0 fils (les feuilles) soit exactement 2 fils. Un nœud d'un arbre de Fenwick représente un intervalle ou une valeur. Un nœud non feuille représente un intervalle $(d..f)$ et stocke la somme des valeurs sur cet intervalle $\sum_{i=d}^{f} v_i$. Le fils gauche d'un nœud non feuille représente l'intervalle de gauche $(d..(d+f)\ div\ 2)$, et son fils droit l'intervalle de droite $((d+f)\ div\ 2+1..f)$. Une feuille représente une valeur v_i. S'il y a n valeurs de référencées, l'intervalle dans la racine de l'arbre est $1..n$.

On calcule la somme préfixes d'un indice j ($\sum_{i=1}^{j} v_i$) de la manière suivante :
— si le nœud courant n'est pas une feuille, on compare l'indice j à l'intervalle $d..f$ courant. Trois cas apparaît alors :
  — $j < d$: dans ce cas la somme préfixes vaut 0 ;
  — $j \geq f$: dans ce cas la somme préfixes vaut la valeur qu'il y a dans le nœud courant (la somme des valeurs contenu dans les deux fils) ;
  — $j \geq d$ et $j < f$: dans ce cas la somme préfixes vaut l'addition de la somme préfixes de j pour le sous arbre gauche et de la somme préfixes de j pour le sous arbre droit.
— si le nœud courant est une feuille, la somme préfixe vaut v_i.

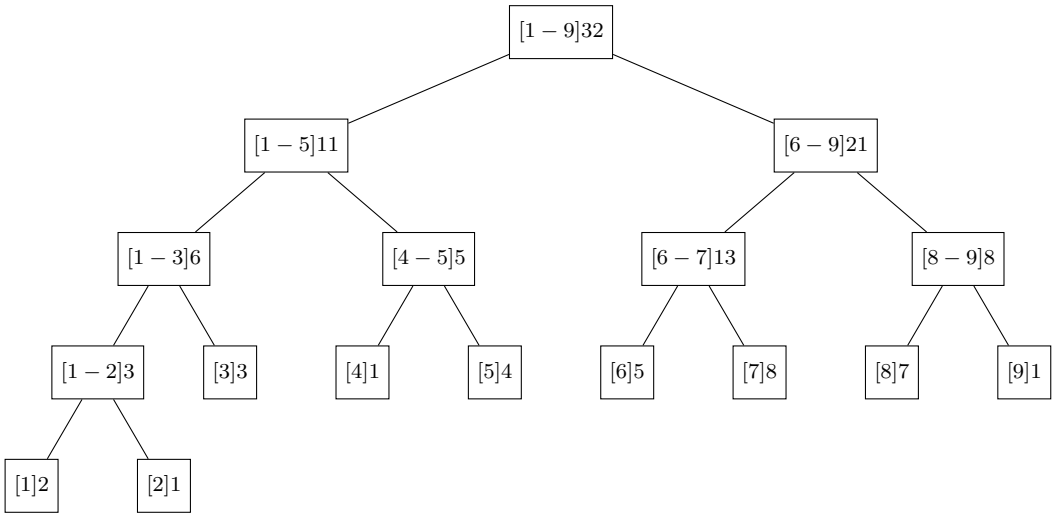
Soit les 9 valeurs v_i suivantes :

2	1	3	1	4	5	8	7	1
1	2	3	4	5	6	7	8	9

1. Dessinez l'arbre de Fenwick pour ces valeurs. Vous penserez à indiquer pour les nœuds non feuille les intervalles et les sommes des valeurs sur ces intervalles. Pour les feuilles vous indiquerez i et v_i.
Solution proposée :

Attendus d'apprentissages disciplinaires évalués

— AN004 : Comprendre et appliquer des consignes algorithmiques sur un exemple



2. En utilisant l'arbre que vous venez de dessiner, calculez la somme préfixes pour l'indice 7. Expliquez votre démarche.
Solution proposée :

Attendus d'apprentissages disciplinaires évalués

— AN004 : Comprendre et appliquer des consignes algorithmiques sur un exemple

— On part de la racine, 7 est au sein de l'intervalle 1..9, il faut donc additionner la somme préfixes pour le fils gauche et la somme préfixes pour le fils droit :
  — Pour le fils gauche, l'intervalle est de 1..5, 7 est donc supérieur à 5, cette somme vaut donc 11.
  — Pour le fils droit, l'intervalle est 6..9, il faut donc additionner la somme préfixe pour le fils gauche et la somme préfixe pour le fils droit :
    — Pour le fils gauche, l'intervalle est de 6..7, $7 \geq 7$, cette somme vaut donc 13.
    — Pour le fils droit, l'intervalle est de 8..9, $7 < 8$, donc la somme vaut 0

Finalement la somme préfixes calculée est $11 + 13$ soit 24

2.2 Analyse (4 compétences)

On considère que les valeurs stockées sont de type entier et qu'elles sont indicées à partir de 1.

Soit le TAD `ArbreFenwick` pour lequel on peut :

1. créer un arbre de Fenwick à partir d'un nombre de valeurs, toutes nulles ;
2. créer un arbre à partir d'une liste de valeurs ;
3. savoir si un arbre de Fenwick est une feuille ;
4. obtenir les deux arbres de Fenwick (fils gauche et droit) lorsque l'arbre de Fenwick n'est pas une feuille ;
5. obtenir l'indice d'une valeur lorsque l'arbre de Fenwick est une feuille ;
6. obtenir les indices de l'intervalle lorsque l'arbre de Fenwick n'est pas une feuille ;
7. obtenir la valeur se trouvant à la racine d'un arbre de Fenwick : v_i si c'est une feuille, $\sum_{i=d}^{f} v_i$ sinon
8. additionner un entier à une valeur identifiée à partir de son indice.

Décrivez ce TAD (sans les axiomes et sémantiques).

Solution proposée :

Attendus d'apprentissages disciplinaires évalués

- AN201 : Identifier les dépendances d'un TAD
- AN203 : Savoir si une opération identifiée fait partie du TAD à spécifier
- AN204 : Formaliser des opérations d'un TAD
- AN205 : Formaliser les préconditions d'une opération d'un TAD

Nom: ArbreFenwick

Utilise: **Booleen**, **NaturelNonNul**, **Entier**, Liste

Opérations:

afAPartirDeNZeros:	**NaturelNonNul** → ArbreFenwick
afAPartirDeValeurs:	Liste<**Entier**> ⇸ ArbreFenwick
estUneFeuille:	ArbreFenwick → **Booleen**
obtenirFilsGauche:	ArbreFenwick ⇸ ArbreFenwick
obtenirFilsDroit:	ArbreFenwick ⇸ ArbreFenwick
obtenirIndice:	ArbreFenwick ⇸ **NaturelNonNul**
obtenirIntervalle:	ArbreFenwick ⇸ **NaturelNonNul** × **NaturelNonNul**
obtenirValeur:	ArbreFenwick → **Entier**
additionner:	ArbreFenwick × **NaturelNonNul** × **Entier** ⇸ ArbreFenwick

Préconditions:

afAPartirDeValeurs(l):	non estVide(l)
obtenirFilsGauche(af):	non estUneFeuille(af)
obtenirFilsDroit(af):	non estUneFeuille(af)
obtenirIndice(a):	estUneFeuille(af)
obtenirIntervalle(af):	non estUneFeuille(af)
additionner(af,i,v):	(estUneFeuille(af) et obtenirIndice(af)=i) ou (non estUneFeuille(af) et avec d,f ← obtenirIntervalle(af), $i \in d..f$)

2.3 Conception préliminaire (2 compétences)

Proposez les signatures des fonctions et procédures correspondant aux opérations du TAD `ArbreFenwick`.

Solution proposée :

Attendus d'apprentissages disciplinaires évalués

- CP003 : Choisir entre une fonction et une procédure
- CP004 : Concevoir une signature (préconditions incluses)

- **fonction** afAPartirDeNZeros **(**n : **NaturelNonNul) :** ArbreFenwick
- **fonction** afAPartirDeValeurs **(**vs : Liste<**Entier**>**) :** ArbreFenwick
  - ⌊**précondition(s)** non estVide(l)
- **fonction** estUneFeuille **(**af : ArbreFenwick**) : Booleen**
- **fonction** obtenirFilsGauche **(**af : ArbreFenwick**) :** ArbreFenwick
  - ⌊**précondition(s)** non estUneFeuille(af)
- **fonction** obtenirFilsDroit **(**af : ArbreFenwick**) :** ArbreFenwick
  - ⌊**précondition(s)** non estUneFeuille(af)
- **fonction** obtenirIndice **(**af : ArbreFenwick**) : NaturelNonNul**
  - ⌊**précondition(s)** estUneFeuille(af)
- **fonction** obtenirIntervalle **(**af : ArbreFenwick**) : NaturelNonNul** × **NaturelNonNul**
  - ⌊**précondition(s)** non estUneFeuille(af)
- **fonction** obtenirValeur **(**af : ArbreFenwick**) : Entier**
- **procédure** additionner **(E/S** af : ArbreFenwick, **E** i : **NaturelNonNul**, v : **Entier)**
  - ⌊**précondition(s)** (estUneFeuille(af) et obtenirIndice(af)=i) ou (non estUneFeuille(af) et avec d,f ← obtenirIntervalle(af), $i \in d..f$)

2.4 Conception détaillée (7 compétences)

On décide de représenter le type `ArbreFenwick` à l'aide de la SDD `ArbreBinaire` (cette SDD est rappelée en Annexe) de la façon suivante :

Type Contenu = **Structure**
val : **Entier**
iDebut : **NaturelNonNul**
iFin : **NaturelNonNul**
finstructure
Type ArbreFenwick = ArbreBinaire<Contenu>

Tel que :

— *val* est la valeur si le noeud courant est une feuille, ou si ce n'est pas une feuille, *val* est égale à la somme des valeurs contenu dans ses deux sous arbres ;
— *iDebut*, *iFin* sont les indices de l'intervalle, ou l'indice $iDebut = iFin$ pour les feuilles.

On rappelle que l'arbre de Fenwick permet de mettre à jour efficacement une des valeurs v_i. Cette mise à jour doit impacter les éléments concernés de l'arbre. Donnez l'algorithme itératif de la fonction ou de la procédure qui permet d'additionner un entier à une valeur désignée par son indice (opération numéro 8 de la question 2.2)

Solution proposée :

Attendus d'apprentissages disciplinaires évalués

— CP003 : Choisir entre une fonction et une procédure
— CP004 : Concevoir une signature (préconditions incluses)
— CD002 : En tant qu'utilisateur, respecter une signature
— CD003 : Utiliser le principe d'encapsulation
— CD005 : Écrire un pseudo code lisible (indentation, identifiant significatif)
— CD006 : Choisir la bonne itération
— CD009 : Écrire un algorithme qui résout le problème

procédure additionnerAuNoeud **(E/S** af : ArbreFenwick, **E** v : **Entier)**
Déclaration contenu : Contenu
debut
contenu ← obtenirElement(af)
contenu.val ← contenu.val+v
fixerElement(af,contenu)
fin
procédure additionner **(E/S** af : ArbreFenwick, **E** i : **NaturelNonNul**, v : **Entier)**
⌊**précondition(s)** (estUneFeuille(af) et obtenirIndice(af)=i) ou (non estUneFeuille(af) et avec d,f ← obtenirIntervalle(af), $i \in d..f$)
Déclaration afCourant : ArbreFenwick
contenu : Contenu
debut
afCourant ← af
tant que non estUneFeuille(af) **faire**
additionnerAuNoeud(afCourant,v)
contenu ← obtenirElement(af)
si i≤(contenu.iDebut+contenu.iFin) div 2 **alors**
afCourant ← obtenirFilsGauche(afCourant)
sinon
afCourant ← obtenirFilsDroit(afCourant)
finsi
fintantque
additionnerAuNoeud(afCourant,v)
fin

Note : af aurait pu avoir un passage de paramètre en entrée car ce n'est pas le pointeur qui est modifié mais les nœuds pointés. Le choix du passage de paramètre en entrée/sortie nous oblige à utiliser une variable locale ($afCourant$) pour parcourir l'arbre.

2.5 Utilisation : somme préfixes (9 compétences)

Donnez l'algorithme récursif de la fonction ou de la procédure qui permet de calculer la somme préfixes des i premières valeurs d'un arbre de Fenwick. Si i est plus grand que le nombre de valeurs alors la somme préfixes sera la somme de toutes ces valeurs.

Votre algorithme récursif est-il terminal ? Qu'est-ce que cela implique ? Justifiez

Solution proposée :

Attendus d'apprentissages disciplinaires évalués

- CP003 : Choisir entre une fonction et une procédure
- CP004 : Concevoir une signature (préconditions incluses)
- CD002 : En tant qu'utilisateur, respecter une signature
- CD003 : Utiliser le principe d'encapsulation
- CD005 : Écrire un pseudo code lisible (indentation, identifiant significatif)
- CD009 : Écrire un algorithme qui résout le problème
- CD201 : Identifier et résoudre le problème des cas non récursifs
- CD202 : Identifier et résoudre le problème des cas récursifs
- CD203 : Identifier une récursivité terminale et non terminale et ce que cela implique

fonction calculerSommePrefixes (af : ArbreFenwick, i : **NaturelNonNul**) **: Entier**
  Déclaration d,f : **NaturelNonNul**
debut
  si estUneFeuille(af) **alors**
    retourner obtenirValeur(af)
  finsi
  d,f ← obtenirIntervalle(af)
  si i<d **alors**
    retourner 0
  finsi
  si i≥f **alors**
    retourner obtenirSomme(af)
  finsi
  retourner calculerSommePrefixes(obtenirFilsGauche(af),i) + calculerSommePrefixes(obtenirFilsDroit(af),i)
fin

Il s'agit d'un algorithme non terminal (car il y a l'un des deux appels récusrifs n'est pas la dernière instruction. Il est donc impossible de dérécursiver automatiquement cet algorithme.

Annexes

En cours nous avons conçu le SDD `ArbreBinaire` de la façon suivante :

Type ArbreBinaire = ^ Noeud
Type Noeud = **Structure**
  lElement : Element
  filsGauche : ArbreBinaire
  filsDroit : ArbreBinaire
finstructure

Que l'on peut utiliser à l'aide des fonctions et procédure suivantes :

- **fonction** arbreBinaire **() :** ArbreBinaire
- **fonction** estVide (a : ArbreBinaire) **: Booleen**
- **fonction** ajouterRacine (fg,fd : ArbreBinaire,e : Element) **:** ArbreBinaire
- **fonction** obtenirElement (a : ArbreBinaire) **:** Element
  ⌊**précondition(s)** non estVide(a)
- **fonction** obtenirFilsGauche (a : ArbreBinaire) **:** ArbreBinaire
  ⌊**précondition(s)** non estVide(a)
- **fonction** obtenirFilsDroit (a : ArbreBinaire) **:** ArbreBinaire
  ⌊**précondition(s)** non estVide(a)
- **procédure** fixerElement (**E** a : ArbreBinaire, e : Element)

⌊**précondition(s)** non estVide(a)

— **procédure** fixerFilsGauche **(E** a : ArbreBinaire, ag : ArbreBinaire**)**

⌊**précondition(s)** non estVide(a)

— **procédure** fixerFilsDroit **(E** a : ArbreBinaire, ad : ArbreBinaire**)**

⌊**précondition(s)** non estVide(a)

— **procédure** supprimerRacine **(E/S** a : ArbreBinaire, **S** fg,fd : ArbreBinaire**)**

⌊**précondition(s)** non estVide(a)

— **procédure** supprimer **(E/S** a : ArbreBinaire**)**