Département STPI 17 juin 2013

I3 - Algorithmique

Durée : *3h*
Documents autorisés : **AUCUN** (calculatrice comprise) [1]

Remarques :
- Veuillez lire attentivement les questions avant de répondre.
- Le barème donné est un barème indicatif qui pourra évoluer lors de la correction.
- Rendez une copie propre.
- N'utilisez pas de crayon à papier.

1 Question de cours (4 points)

1.1 Tris (2 points)

Démontrez que le tri rapide est en :
- Complexité dans le meilleur des cas en $\Omega(n * log_2(n))$
- Complexité dans le pire des cas en $O(n^2)$

Solution proposée :

Voir cours

1.2 Algorithme min-max (2 points)

Après avoir rappelé en quelques lignes le principe de l'algorithme min-max, quel est le coup qui sera choisi par cet algorithme sur l'arbre présenté par la figure 1. Justifiez.

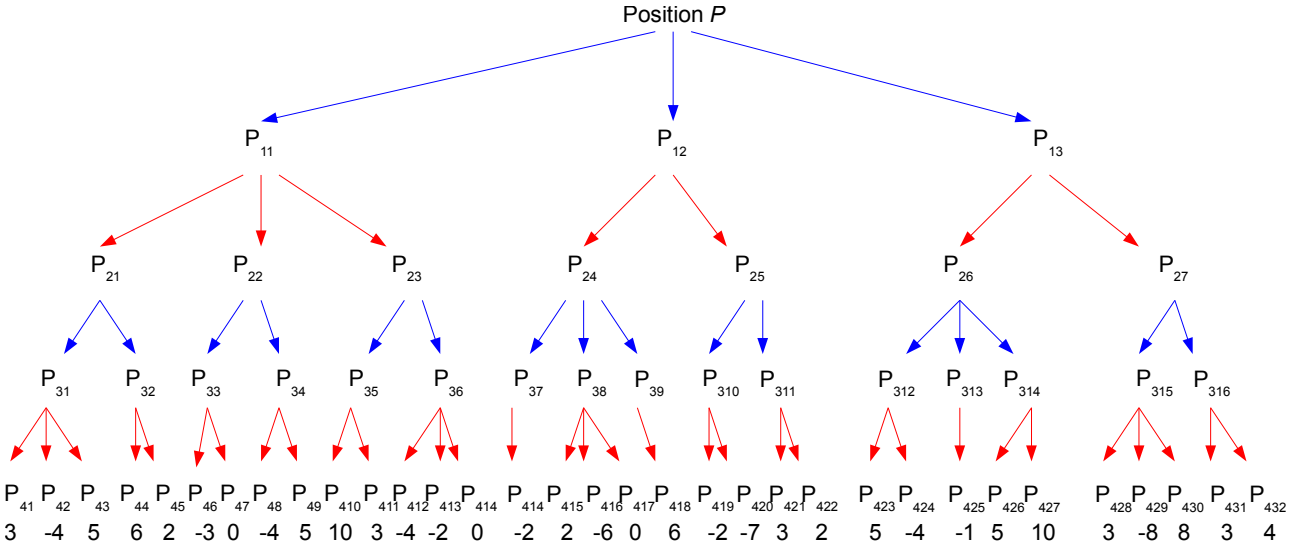
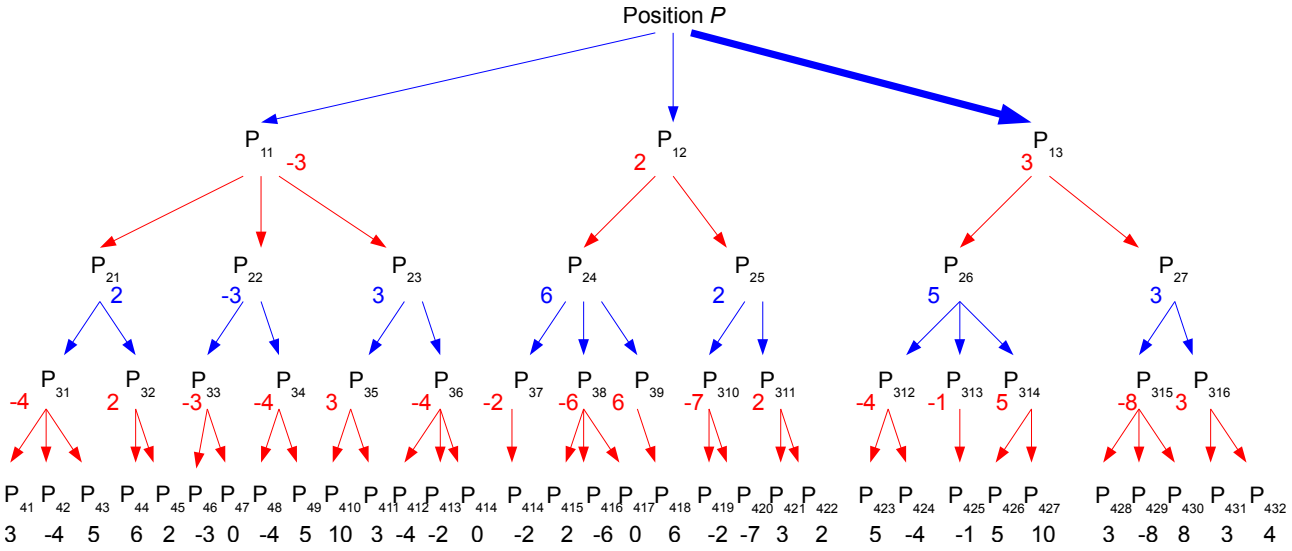


FIGURE 1 – Arbre de positions

Solution proposée :

1. Sauf les dictionnaires pour les étudiants non francophones



Solution proposée :

Voir cours

2 Question de TD (4 points)

Pour rappel, le type ListeChaineeDEntiers est défini de la façon suivante :

Type ListeChaineeDEntiers **=** ^ NoeudDEntier
Type NoeudDEntier **= Structure**
entier : **Entier**
listeSuivante : ListeChaineeDEntiers
finstructure

Et nous l'utilisons à l'aide des fonctions et procédures suivantes :

- **fonction** listeVide () **:** ListeChaineeDEntiers
- **fonction** estVide (uneListe : ListeChaineeDEntiers) **: Booleen**
- **procédure** ajouter (**E/S** uneListe : ListeChaineeDEntiers,**E** element : Entier)
- **fonction** obtenirEntier (uneListe : ListeChaineeDEntiers) **:** Entier

  ⌊**précondition(s)** non(estVide(uneListe))
- **fonction** obtenirListeSuivante (uneListe : ListeChaineeDEntiers) **:** ListeChaineeDEntiers

  ⌊**précondition(s)** non(estVide(uneListe))
- **procédure** fixerListeSuivante (**E/S** uneListe : ListeChaineeDEntiers,**E** nelleSuite : ListeChaineeDEntiers)

  ⌊**précondition(s)** non(estVide(uneListe))
- **procédure** supprimerTete (**E/S** l :ListeChaineeDEntiers)

  ⌊**précondition(s)** non estVide(l)
- **procédure** supprimer (**E/S** uneListe : ListeChaineeDEntiers)

1. Écrire une fonction booléenne itérative, estPresent, qui permet de savoir si un entier est présent dans une liste chaînée.
2. Écrire une fonction booléenne récursive, estPresent, qui permet de savoir si un entier est présent dans une liste chaînée.

Solution proposée :

Voir TD

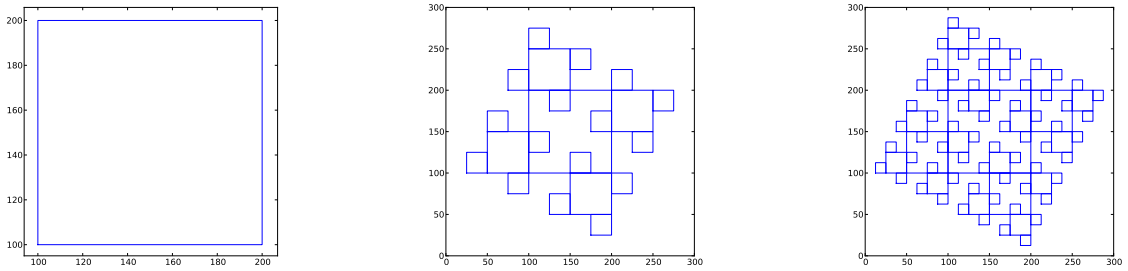
3 Dessin récursif (4 points) [2]

Supposons que la procédure suivante permette de dessiner un carré sur un graphique (variable de type `Graphique`) :

– **procédure** carre **(E/S** g : Graphique,**E** x,y,cote **Reel)**

L'objectif est de concevoir une procédure `carres` qui permet de dessiner sur un graphique des dessins récursifs tels que présentés par la figure 2. La signature de cette procédure est :

– **procédure** carres **(E/S** g : Graphique,**E** x,y,cote : **Reel**, n : **NaturelNonNul)**

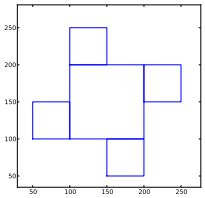


(a) $carres(100, 100, 100, 1)$ (b) $carres(100, 100, 100, 3)$ (c) $carres(100, 100, 100, 4)$

FIGURE 2 – Résultats de différents appels de la procédure carres

1. Dessinez le résultat de l'exécution de $carres(100, 100, 100, 2)$.
2. Donnez l'algorithme de la procédure `carres`.

Solution proposée :

1. 
2. Algorithme

```
procédure carres (E/S g : Graphique,E x,y,cote : Reel, n : NaturelNonNul)
debut
    si n≥1 alors
        carre(x,y,cote)
        carres(x-cote/2,y,cote/2,n-1)
        carres(x,y+cote,cote/2,n-1)
        carres(x+cote,y+cote/2,cote/2,n-1)
        carres(x+cote/2,y-cote/2,cote/2,n-1)
    finsi
fin
```

2. Inspiré de `http://www-fourier.ujf-grenoble.fr/~parisse/giac/doc/fr/casrouge/casrouge018.html`

4 Traitements d'image (8 points)

Une image bitmap en 256 niveaux de gris est représentée par des pixels. A chaque pixel sont associés des coordonnées (x, y) et un entier (entre 0 et 255), déterminant la couleur (0 pour noir ; 255 pour blanc). Une image peut être utilisée à l'aide du type `ImageNB` et des fonctions et procédures suivantes :

- **fonction** imageNB **(**largeur,hauteur : **NaturelNonNul**, couleurFond : 0..255**) :** ImageNB
- **fonction** largeur **(**im : ImageNB**) : NaturelNonNul**
- **fonction** hauteur **(**im : ImageNB**) : NaturelNonNul**
- **fonction** obtenirCouleur **(**im : ImageNB,x,y : **Naturel) :** 0..255

  ⌊**précondition(s)** x≤0 et x<largeur(im) et y≤0 et y<hauteur(im)
- **procédure** fixerCouleur **(E/S** im : ImageNB,**E** x,y : **Naturel**, couleur : 0..255**)**

  ⌊**précondition(s)** x≤0 et x<largeur(im) et y≤0 et y<hauteur(im)

Le pixel de coordonnées $(0, 0)$ se trouvant en haut à gauche de l'image.

En traitement d'image, un filtre est une fonction qui calcule une image destination à partir d'une image source. Chaque pixel de l'image destination est fonction des couleurs des pixels de l'image source et optionnellement des couleurs des pixels de l'image destination déjà calculées.

Filtre moyenneur

Le principe du filtre moyenneur est que la couleur de chaque pixel x, y de l'image destination est égale à la moyenne des couleurs des pixels appartenant à une fenêtre carrée centrée en x, y de l'image source. La largeur de cette fenêtre carrée est obligatoirement impaire. L'image destination semble alors floue.

La figure 3 présente l'utilisation d'un filtre moyenneur avec une fenêtre de taille 5 (exemple issue de `http://perso.telecom-paristech.fr/~maitre/BETI//filtres_lin_nlin/filtres.html`).

(a) Image source (b) Image destination

FIGURE 3 – Utilisation d'un filtre moyenneur de taille 5

Pour calculer un filtre moyenneur, il faut donc être capable d'identifier les bornes (coin haut gauche et bas droit) des pixels de la fenêtre pour une coordonnée donnée. Il faut aussi calculer la moyenne des pixels à l'intérieur de cette fenêtre. Et pour calculer cette moyenne, il faut être capable de calculer la somme de niveaux de gris des pixels de cette fenêtre.

Questions

1. Analyse : Proposez une analyse descendante du problème.
2. Conception préliminaire : Donnez les signatures des fonctions et procédures issues de votre analyse.
3. Conception détaillée : Donnez les algorithmes de toutes les fonctions et procédures de votre conception préliminaire.

Solution proposée :

1. Analyse :
   - filtreMoyenneur : ImageNB × **NaturelNonNul** ⇸ ImageNB
     - bornesFenetre : **NaturelNonNul** × **NaturelNonNul** × **NaturelNonNul** ⇸ **NaturelNonNul** × **NaturelNonNul** × **NaturelNonNul** × **NaturelNonNul**
       - min : **Entier** × **Entier** → **Entier**
       - max : **Entier** × **Entier** → **Entier**
     - moyenneFenetree : ImageNB × **NaturelNonNul** × **NaturelNonNul** × **NaturelNonNul** × **NaturelNonNul** ⇸ 0..255
       - sommeFenetree ImageNB × **NaturelNonNul** × **NaturelNonNul** × **NaturelNonNul** × **NaturelNonNul** ⇸ **NaturelNonNul**
2. CP :

   fonction filtreMoyenneur (im : ImageNB, tailleFenetre : **NaturelNonNul) :** ImageNB

   ⌊**précondition(s)** impair(tailleFenetre)

   procédure bornesFenetre **(E** im : ImageNB,x,y : **NaturelNonNul**, tailleFenetre : **NaturelNonNul**, **S** x1,y1,x2,y2 : **NaturelNonNul)**

   ⌊**précondition(s)** x≤0 et x<largeur(im) et y≤0 et y<hauteur(im) et impair(tailleFenetre)

   fonction moyenneFenetree (im : imageNB, x1 ,y1, x2, y2 **NaturelNonNul) :** 0..255

   ⌊**précondition(s)** x1≤0 et x1<largeur(im) et y1≤0 et y1<hauteur(im) et x2≤0 et x2<largeur(im) et y2≤0 et y2<hauteur(im)

   fonction sommeFenetree (im : imageNB, x1 ,y1, x2, y2 **NaturelNonNul) :** 0..255

   ⌊**précondition(s)** x1≤0 et x1<largeur(im) et y1≤0 et y1<hauteur(im) et x2≤0 et x2<largeur(im) et y2≤0 et y2<hauteur(im)
3. CD :

   procédure bornesFenetre **(E** im : ImageNB, x,y : **NaturelNonNul**, tailleFenetre : **NaturelNonNul**, **S** x1,y1,x2,y2 : **NaturelNonNul)**

   ⌊**précondition(s)** x≤0 et x<largeur(im) et y≤0 et y<hauteur(im) et impair(tailleFenetre)

   debut
   x1 ← max(0,x - tailleFenetre div 2)
   y1 ← max(0,y - tailleFenetre div 2)
   x2 ← min(largeur(im),x + tailleFenetre div 2)
   y2 ← min(largeur(im),y + tailleFenetre div 2)
   fin

   fonction filtreMoyenneur (im : ImageNB, tailleFenetre, x1,y1,x2,y2 : **NaturelNonNul) :** ImageNB

   ⌊**précondition(s)** impair(tailleFenetre)

   Déclaration i,j : **Naturel**
   res : ImageNB

   debut
   im ← imageNB(largeur(im),hauteur(im),0)
   pour i ←0 **à** largeur(im)-1 **faire**
   pour j ←0 **à** hauteur(im)-1 **faire**
   bornesFenetre(i,j,tailleFenetre,x1,y1,x2,y2)
   fixerCouleur(res,i,j,moyenneFenetree(im,x1,y1,x2,y2))
   finpour
   finpour
   retourner res
   fin

   fonction moyenneFenetree (im : imageNB, x,y : **Naturel**, tailleFenetre : **NaturelNonNul) :** [0..255]

   ⌊**précondition(s)** x≤0 et x<largeur(im) et y≤0 et y<hauteur(im) et impair(tailleFenetre)

Déclaration i,j,somme,nbPixel : **Naturel**

debut
retourner somme div (x2-x1)*(y2-y1)
fin
fonction sommeFenetree **(**im : imageNB, x,y : **Naturel**, tailleFenetre : **NaturelNonNul) :** [0..255]
⌊**précondition(s)** x≤0 et x<largeur(im) et y≤0 et y<hauteur(im) et impair(tailleFenetre)
Déclaration i,j,somme,nbPixel : **Naturel**

debut
somme ← 0
pour i ←x1 **à** x2 **faire**
pour j ←y1 **à** y2 **faire**
somme ← somme+obtenirCouleur(im,i,j)
finpour
finpour
retourner somme
fin