Premier cycle Mardi 24 Juin 2008

I3 - Algorithmique

Durée : *3h00*
Documents autorisés : **AUCUN** (calculatrice comprise)

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.

1 Compréhension d'algorithmes vus en cours (5 points)

1.1 Min-max

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.

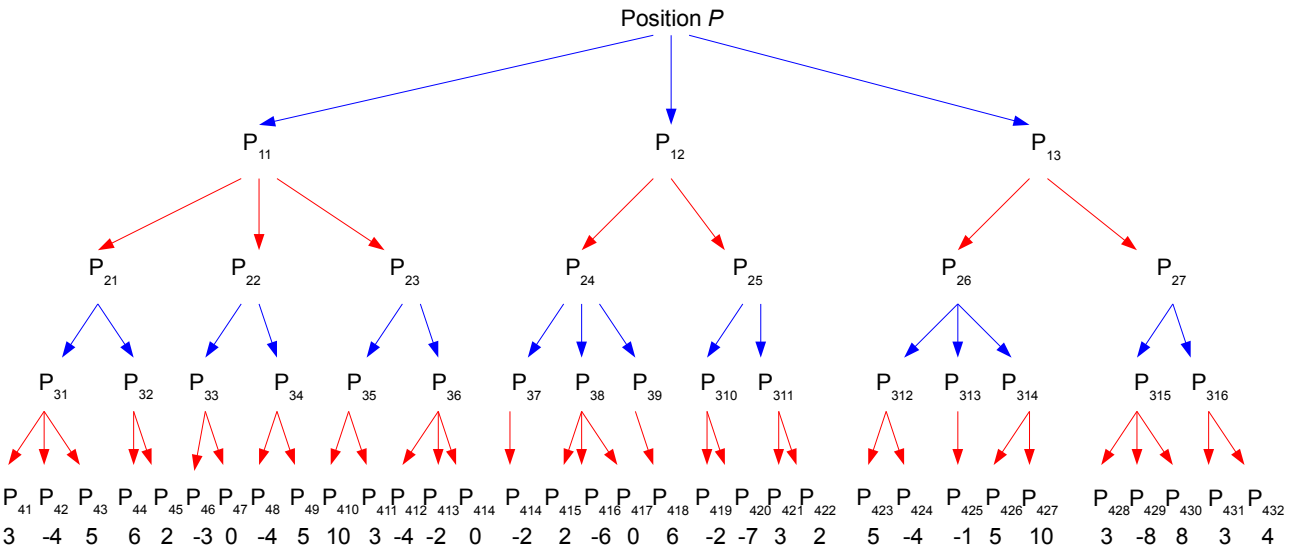

FIG. 1 – Arbre de positions

1.2 *paint*

Dans le cours sur la récursivité nous avons vu l'algorithme de la procédure *remplir* suivante :

```
procédure remplir ( E/S e : Ecran , E x,y : Naturel , ancienneCouleur, nouvelleCouleur : Couleur)
debut
    si obtenirCouleurPixel(e,x,y)=ancienneCouleur alors
        fixerCouleurPixel(e,x,y,nouvelleCouleur)
        remplir(e,x,y-1,ancienneCouleur,nouvelleCouleur)
        remplir(e,x,y+1,ancienneCouleur,nouvelleCouleur)
        remplir(e,x-1,y,ancienneCouleur,nouvelleCouleur)
        remplir(e,x+1,y,ancienneCouleur,nouvelleCouleur)
    finsi
fin
```

Après avoir recopié la figure 2 sur votre copie, numéroter les pixels qui seront coloriés suivant leur ordre d'apparition à l'écran en commençant par le pixel numéroté 1 (l'ancienne couleur est le blanc, la nouvelle le noir).

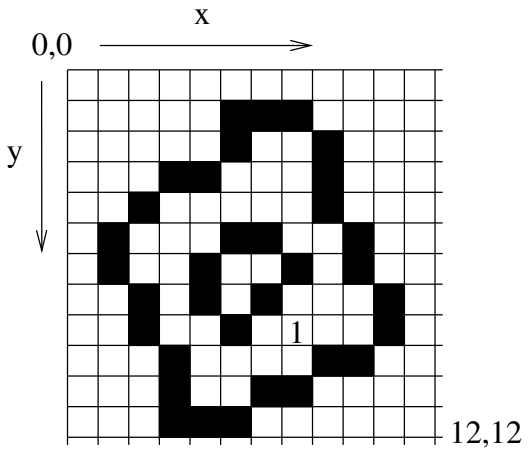

FIG. 2 – Surface à remplir

1.3 Partitionner

Après avoir rappelé en quelques lignes le principe du partitionnement d'un tableau utilisé dans le tri rapide, donner le résultat du partionnement du tableau suivant en considérant que la valeur pivot est la valeur de la première case.

5	4	3	7	10	6	9	1	3	12	9	8
1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*

2 Polynome (4 points)

Soit le type *Polynome* avec les opérations suivantes :
- **fonction** monome (a : **Reel**, n : **Naturel) :** Polynome
- **fonction** addition (p1,p2 : Polynome) **:** Polynome
- **fonction** degre (p : Polynome) **: Naturel**
- **fonction** coefficient (p : Polynome, n : **Naturel) : Reel**

Donnez le corps des fonctions suivantes :

1. **fonction** multiplicationParScalaire (p : Polynome, a : **Reel) :** Polynome
2. **fonction** soustraction (p1,p2 : Polynome) **:** Polynome
3. **fonction** multiplication (p1,p2 : Polynome) **:** Polynome
4. **fonction** derivation (p : Polynome) **:** Polynome

3 Problème du labyrinthe (11 points)[1]

L'objectif de cet exercice est d'étudier le problème du labyrinthe. Comme l'indique la figure 3, l'objectif est de trouver un algorithme permettant de trouver le chemin qui mène de l'entrée à la sortie.

En fait, un labyrinthe est composé de cases. On accède à une case à partir d'une case et d'une direction. Les directions possibles sont `Nord`, `Sud`, `Est` et `Ouest`.

Par exemple, comme le montre la figure 4, le labyrinthe précédent peut être considéré comme étant composé de 25 cases. La case numéro 6 est la case d'entrée. La case 20 est la case de sortie. La case 8 est accessible depuis la case 13 avec la direction `Nord`.

[1]Les 4 premières parties de cet exercice sont indépendantes

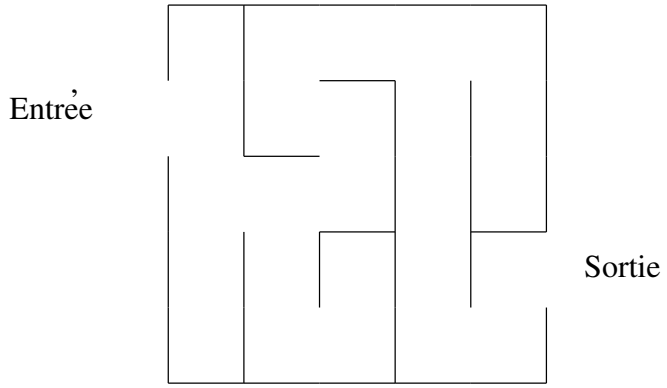

FIG. 3 – Un labytinthe

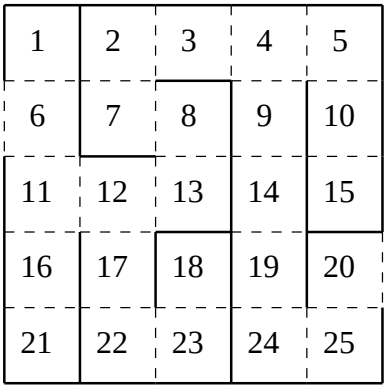

FIG. 4 – Un labytinthe composé de cases

3.1 Type direction (0,25 points)

Donner le type `Direction`.

3.2 Conception préliminaire d'un ensemble de directions (2,5 points)

À l'image du type `EnsembleDEntiers` vu en TD, on considère que l'on a le type `EnsembleDeDirections`.

1. Rappeler les opérations disponibles pour le type `EnsembleDEntiers`.
2. Donner les signatures des procédures et fonctions de ces opérations pour le type `EnsembleDeDirections`.

3.3 Conception détaillée d'une pile de directions (4 points)

Une pile est un type de données de type LIFO (*Last In First Out*), c'est-à-dire que la première donnée insérée dans une pile (on dit que l'on *empile* une donnée) sera la dernière donnée qui pourra être retirée de la pile (on dit que l'on *dépile* une donnée). Réciproquement, la dernière donnée empilée sera la première à être dépilée.

La figure 5 est une représentation graphique classique d'une pile. Dans cet exemple, c'est e_1 qui a été empilé en premier. e_2 a été empilé en deuxième, e_3 en troisième. C'est e_3 qui sera le premier à être dépilé.

Les opérations pour manipuler une pile sont les suivantes :

– créer une pile vide (sans aucun élément)
– savoir si une pile est vide
– empiler un élément dans une pile
– dépiler un élément d'une pile

En considérant que les éléments que l'on va utiliser sont des directions, on peut concevoir le type `PileDeDirections` de la façon suivante :

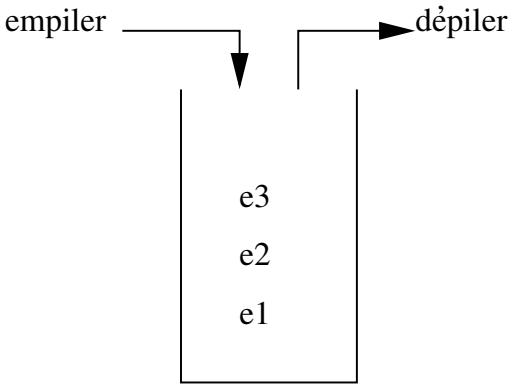

FIG. 5 – Une pile

Type PilesDeDirections **= Structure**
lesDirections : **Tableau[**1..MAX**] de** Direction
nb : **Naturel**
finstructure

La conception préliminaire de ce type donne les signatures de fonctions et procédures suivantes :
- **fonction** creerPileDeDirections **() :** PileDeDirections
- **fonction** estUnePileDeDirectionsVide (p : PileDeDirections**) : Booleen**
- **procédure** empilerDirection **(** **E/S** p : PileDeDirections , **E** d : Direction **)**
- **procédure** depilerDirection **(** **E/S** p : PileDeDirections , **S** d : Direction **)**
  ...avec $non(estUnePileDeDirectionsVide(p))$

Donner le corps de ces fonctions et procédures.

3.4 Conception préliminaire du type Labyrinthe (1,25 points)

Les opérations disponibles sur un labyrinthe sont les suivantes :
- créer un labyrinthe (rectangle d'une certaine dimension),
- obtenir la case d'entrée,
- savoir si une case est la case de sortie,
- obtenir un ensemble de directions possibles depuis une case donnée,
- obtenir la case accessible depuis une case et une direction.

Donner les signatures des fonctions et procédures correspondant aux opérations décrites ci-dessus.

3.5 Utilisation du type Labyrinthe : Algorithme du petit-poucet (4 points)

L'objectif de cette dernière partie est d'obtenir une pile de directions qui permet de trouver la sortie depuis une case. Une solution a ce problème est d'utiliser le principe du petit poucet, c'est-à-dire mettre un caillou sur les cases rencontrées. Lorsque l'on tombe sur un cul de sac ou une case déjà rencontrée, on « retourne » à la case précédente pour tester une autre direction.

Pour ne pas modifier le type Labyrinthe, plutôt que de marquer une case avec un caillou on peut ajouter une case à un ensemble (`EnsembleDEntiers` vu en TD). Pour vérifier si on a déja rencontré une case, il suffit alors de vérifier si la case est présente dans l'ensemble.

Proposer le corps de la procédure suivante qui permet de trouver le chemin de sortie (s'il existe) à partir d'une case donnée[2] :
- **procédure** calculerCheminDeSortie **(** **E** l : Labyrinthe, caseCourante : **Entier** , **E/S** casesVisitees : EnsembleDEntiers , **S** permetDAllerJusquALaSortie : **Booleen**, lesDirectionsASuivre : PileDeDirections **)**

[2]Vous pouvez vous inspirer de la procédure *remplir* du premier exercice