EISTI Ecole Internationale des Sciences du Traitement de l'Information

ING1 2010-2011

Algorithmique II

Durée : 2 heures. Tout document papier autorisé. Chaque fois que vous utilisez un type abstrait vu en cours vous devez respecter scrupuleusement la signature des opérations de ce type.

1. Une pile limitée (5 pts)

Dans cet exercice, on s'intéresse à un nouveau type de conteneur : Pile à taille limitée. Ce conteneur marche exactement comme une pile à ceci près que le nombre d'éléments dans le conteneur est borné. Cette borne est fixée lors de la création de la pile. Pour simplifier on supposera que cette pile contient des entiers. A tout moment les deux premiers éléments de la pile contiennent la taille maximum de la pile (au sommet) et le nombre d'éléments empilés (juste en dessous du sommet). A chaque ajout dans la pile, on a donc deux cas de figures

- La pile n'est pas pleine et l'élément est ajouté en mode LIFO comme dans une pile quelconque
- La pile est pleine et dans ce cas, on retire l'élément le plus récent de la pile pour laisser la place au nouvel élément. Le nouvel élément est alors ajouté en mode LIFO.

Questions :

1-1) (1 pt) Écrire dans le type Pile, l'opération d'extension qui permet de créer une pile de taille limitée.

```
Reponse : Constructeur Pile PileVideLimitee(taille : Entier) : Pile
Références locales
p : Pile
 p ← pileVide()
 p.empiler(0)
 p.empiler(taille)
 retourner p
Fin constructeur
```

1-2) (2 pts) Ecrire dans le type Pile, l'opération d'extension qui permet de supprimer un élément dans une pile de taille limitée.

```
Reponse : definie (p.empilerLimitee(e))⇔ p.sommet() > p.depiler().sommet()
Transformateur Pile empilerLimitee(Element e) : Pile
Transforme p
Références locales
taille, nbEmpiles : Entier
 taille ← p.sommet()
 p.depiler()
 nbEmpiles ← p.sommet()
 p.depiler()
 p.empiler(e)
 p.empiler(nbEmpiles + 1)
 p.empiler(taille)
 retourner p
Fin transformateur
```

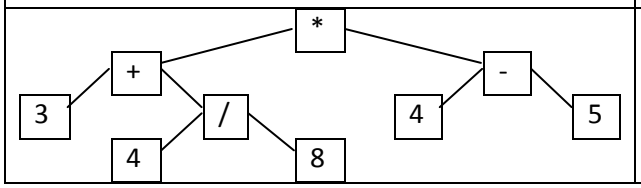
1-3) (2 pts) Ecrire dans le type Pile, l'opération d'extension qui permet d'ajouter un élément dans une pile de taille limitée

```
Reponse : definie (p.depilerLimitee(e))⇔ p.depiler().sommet() > 0
Transformateur Pile depilerLimitee() : Pile
Transforme p
Références locales
taille, nbEmpiles : Entier
 taille ← p.sommet()
 p.depiler()
 nbEmpiles ← p.sommet()
 p.depiler()
 p.depiler()
 p.empiler(nbEmpiles - 1)
 p.empiler(taille)
 retourner p
Fin transformateur
```

2. Expression numérique : Arbre binaire ou expression post-fixée (7pts)

Dans cet exercice, on considère des expressions numériques simples. Une telle expression contient des constantes réelles et les opérateurs *, /, +, -. Pour évaluer une expression de ce type, on peut la fournir en entrée sous diverses formes. On se limitera aux deux formes : arbre binaire et expression post-fixée.

Exemple l'expression (3 + (4/8)) * (4-5) se représente comme suit :

Arbre binaire	Expression post-fixée
	3 4 8 / + 4 5 - *

Questions

2-1) (1 pt) Donner la signature de l'opération d'extension convertirEABenEPF qui a pour paramètre implicite une expression numérique représentée en arbre binaire d'objets de type Terme (voir annexe) et qui retourne une liste contenant cette même expression représentée en post-fixée.

Réponse : Observateur ArbreKAire convertirEABenEPF () : Liste

2-2) (6 pts) Écrire l'algorithme de cette opération.

Réponse : Il suffit de parcourir l'arbre binaire en profondeur en prenant le fils à droite en premier puis le fils à gauche pour respecter l'ordre des opérandes.

```
Observateur ArbreKAire convertirEABenEPF () : Liste
Observé eAB
Références  locales
eL : Liste
 el ← listeVide()
 el.convertirEABenEPF Rec(eAB)
 retourner el
Fin Observateur
```

```
Transformateur Liste convertirEABenEPF (eAB : ArbreKAire) : Liste
Observé eL
Références  locales
eABGauche, eABDroite : ArbreKAire

  eL.ajouter(eAB.recRacine())
  eABGauche ← eAB.recEnfant(1)
  eABDroite ← eAB.recEnfant(2)
  Si non eABGauche.estVide() Alors
     eL.convertirEABenEPF(eABDroite)
     eL.convertirEABenEPF(eABGauche)
  FinSi
  retourner eL
Fin Transformateur
```

3. Graphe : Chaîne de Markov (8 pts)

Dans cet exercice, on modélise l'évolution d'une maladie. Cette maladie est décrite par k états. On observe l'évolution de cette maladie à des intervalles réguliers. On a observé cette évolution sur un échantillon d'individus pendant un grand laps de temps. A l'aide de ces observations, on sait établir pour chaque couple d'états (k_i,k_j) un nombre $p_{i,j}$ qui est la probabilité pour un individu de passer de l'état k_i à l'état k_j entre deux instants d'observations.
On suppose que les changements d'états sont indépendants. Cela signifie que si un malade est dans l'état k_0 et si on note $p_{0,1}$ la probabilité de passer de l'état k_0 à l'état k_1 puis $p_{1,2}$ la probabilité de passer de l'état k_1 à l'état k_2 alors la probabilité de passer de l'état k_0 à l'état k_1 puis de l'état k_1 à l'état k_2 est égale à $p_{0,1} * p_{1,2}$. Plus généralement si un malade est dans l'état k_0 un instant donné alors la probabilité pour qu'il passe respectivement par les états k_1, …, k_n est égale à $\prod_{i=1}^{n} p_{i-1,i}$.
Pour modéliser cette maladie, on utilisera un graphe orienté et valué dont les sommets sont les différents états. Deux états k_i et k_j sont réliés si et seulement si $p_{i,j} > 0$. Dans ce cas la valuation de l'arc est la probabilité $p_{i,j}$.

Questions

3-1)(facile : 2 pts) Dans le type Graphe, écrire l'algorithme de l'opération nommée verifMarkov qui permet de vérifier qu'un graphe représente bien l'évolution d'une maladie comme définie ci-dessus.

Réponse : Il faut vérifier que toutes les valuations des arcs du graphe sont des probabilités (réels entre 0 et 1). En étant plus rigoureux, il faut aussi vérifier que pour un état donné, la somme des valuations des arcs dont l'origine est cet état est égale à 1.

3-2)(difficile : 3 pts) On considère un graphe représentant l'évolution d'une maladie. Ecrire l'opération (signature, pré-condition, algorithme) nommée probaPasserEtatImpossible qui reçoit un état k et qui renvoie un ensemble d'états E. Pour chaque état j de E, il est impossible de passer de l'état k à l'état j dans un laps de temps fini.

Réponse :

Signature : Observateur Graphe probaPasserEtatImpossible(k : Entier) : Ensemble

Pré-condition : aucune

Algorithme en français : Cette opération reçoit le graphe d'évolution et un état d de départ. Elle initialise un ensemble E contenant tous les sommets (ie : tous les états de la maladie). A partir de l'état d, elle fait un parcours récursif (avec marquage de sommet pour éviter les cycles) du graphe (en profondeur ou en largeur peu importe) en passant chaque fois en paramètre le

sommet courant et l'ensemble E. A chaque appel récursif, on enlève de E le sommet courant. A la fin de la récursivité, E contient les états demandés.

3-3) (très difficile : 3 pts) On considère un graphe représentant l'évolution d'une maladie. Ecrire l'opération (signature, pré-condition, algorithme) nommée probaPasserEtatPossible qui reçoit un état k et une probabilité p et qui renvoie un ensemble d'états E. Pour chaque état j de E, il existe une évolution de la maladie qui permet de passer de l'état k à l'état j dans un laps de fini avec une probabilité supérieure ou égale à p.

Réponse :

Signature : Observateur Graphe probaPasserEtatPossible(k : Entier) : Ensemble

Pré-condition : aucune

Algorithme en français : Cette opération reçoit le graphe d'évolution et un état d de départ. Elle initialise un ensemble E à l'ensemble vide. A partir de l'état d, elle fait un parcours récursif (avec marquage de sommet pour éviter les cycles) du graphe (en profondeur ou en largeur peu importe) en passant chaque fois en paramètres le sommet courant, l'ensemble E, la probabilité p et une probabilité pProduit. Au premier appel, on prend 1 comme valeur pour pProduit. Pour chaque appel récursif, la nouvelle valeur du paramètre pProduit est égale au produit de l'ancienne valeur de pProduit par la valuation de l'arc associé à l'appel récursif. Pour chaque appel, on ajoute dans E le sommet courant si la probabilité pProduit est supérieure ou égale à p. De plus si lors d'un appel récursif, on constate pProduit < p alors on arrête la récursivité. En effet, il est inutile de continuer car à chaque appel, on multiplie pProduit par une probabilité qui est un nombre inférieur ou égale à 1, donc pProduit ne peut diminuer ou garder la même valeur. A la fin de la récursivité, E contient les états demandés.

4. Annexes

TYPE ABSTRAIT Pile
Opérations de base
Constructeur Pile : pileVide() : Pile
Transformateur Pile : empiler(Element e) : Pile
Transformateur Pile : depiler() : Pile
Observateur Pile : sommet() : Element
Observateur Pile : estVide() : Booleen

TYPE ABSTRAIT Liste
Opérations de base
Constructeur Liste : listeVide() : Liste
Transformateur Liste : ajouter(e Element) : Liste
Transformateur Liste : supprimer() : Liste
Observateur Liste : reste() : Liste
Observateur Liste : tete() : Element
Observateur Liste : estVide() : Booleen

TYPE ABSTRAIT Terme
Modélisation d'un terme opérande ou opérateur dans n'importe quel expression

Opérations de base
Constructeur Terme : creerTermeOperande(REEL operande) : Terme
Constructeur Terme : creerTermeOperateur(CHAINE operateur) : Terme
Observateur Terme : estOperateur() : BOOLEEN
Observateur Terme : recOperateur() : CHAINE
Observateur Terme : recOperande() : REEL

Axiomes
Pré-conditions
definie(t.recOperateur()) <==> t.estOperateur()
definie(t.recOperande()) <==> Non t.estOperateur()

Post-conditions
creerTermeOperateur(operateur).estOperateur()
creerTermeOperande(operande).estOperande()
creerTermeOperande(operande).recOperande() = operande
creerTermeOperateur(operateur).recOperateur() = operateur

TYPE ABSTRAIT ArbreKAire
Constructeur ArbreKAire : arbreVide() : ArbreKAire
Constructeur ArbreKAire : creerArbre(Element, Entier) : ArbreKAire
Transformateur ArbreKAire : modifierRacine (Element) : ArbreKAire
Transformateur ArbreKAire : affEnfant (Entier, ArbreKAire): ArbreKAire
Transformateur ArbreKAire : supprimerFeuille(Entier) :ArbreKAire
Observateur ArbreKAire : estVide () : Booleen
Observateur ArbreKAire : recRacine() : Element
Observateur ArbreKaire : existeParent() : Booleen
Observateur ArbreKAire : recEnfant(Entier) : ArbreKAire
Observateur ArbreKAire : recParent() : ArbreKAire
Observateur ArbreKAire : recArite() : Entier

TYPE ABSTRAIT Ensemble
Un objet de ce type permet de modèliser les ensembles avec les opérations ensemblistes connues telles que l'appartenance, l'intersection, l'union, la différence, l'inclusion et le cardinal

Opérations de base
Constructeur Ensemble : ensembleVide() : Ensemble
Transformateur Ensemble : ajouter(Element) : Ensemble
Transformateur Ensemble : supprimer(Element) : Ensemble
Observateur Ensemble : choisir () --> Element
Observateur Ensemble : estVide() --> Booleen
Observateur Ensemble : appartient(Element) : Booleen
Observateur Ensemble : cardinal() : Entier

TYPE ABSTRAIT Graphe
Ce type permet de modéliser les graphes. Les sommets sont numérotés de 1 à n.

Opérations de base
Constructeur Graphe : creerGraphe(Entier nbSommets) : Graphe
Transformateur Graphe : ajouterArete(Arete a) : Graphe
Transformateur Graphe : marquer(Entier noS) : Graphe
Transformateur Graphe : demarquer(Entier noS) : Graphe
Observateur Graphe : estMarque(Entier noS) : Booleen
Observateur Graphe : recAretes() : Vecteur
Observateur Graphe : recNbSommets() : Entier
Observateur Graphe : recNbAretes() : Entier
Observateur Graphe : recArete(Entier noSD, Entier noSA) : Arete

Opérations d'extension
// Les prédécesseurs sont rangés par ordre croissant de numéros
Observateur Graphe : recPredecesseurs(Entier noS) : Vecteur

// Les successeurs sont rangés par ordre croissant de numéros
Observateur Graphe : recSuccesseurs(Entier noS) : Vecteur

TYPE ABSTRAIT Arete
Concept : Ce type permet de modéliser les arêtes d'un graphe

Opérations de base
Constructeur Arete : creerArete(Entier o, Entier d) : Arete
Observateur Arete : recOrigine() : Entier
Observateur Arete : recDestination() : Entier

TYPE ABSTRAIT AreteValuee HERITE DE Arete
Ce type permet de modéliser les arêtes valuées d'un graphe. On représente les sommets par des numéros entre 1 et n où n est le nombre de sommets.

Opérations de base
Constructeur AreteValuee : creerAreteValuee(Entier o, Entier d, Reel val) : AreteValuee
Observateur AreteValuee : recValuation() : Reel