EISTI Ecole Internationale des Sciences du Traitement de l'Information

ING1 2009-2010

Algorithmique II

Durée : 2 heures. Aucun document autorisé

1) Piles (4pts)

Dans cet exercice, on ne considère que des piles qui contiennent des entiers.

QUESTIONS :

1-1) Écrire un algorithme pour déplacer les entiers d'une pile dans une autre de telle sorte que dans la pile résultat les nombres pairs soient au dessous des nombres impairs.

1-2) Écrire un algorithme pour copier dans la pile résultat les nombres pairs contenus dans la pile initiale. Le contenu de la pile initiale après exécution de l'algorithme ne doit pas avoir été modifié. Les nombres pairs dans la pile résultat doivent être dans l'ordre où ils apparaissent dans la pile initiale.

2) Graphes : algorithme de Marimont (7pts)

Dans de nombreuses applications, on souhaite savoir si un graphe possède ou non des circuits. L'algorithme de Marimont permet de résoudre ce problème avec une complexité "raisonnable". Il est basé sur la propriété qui suit :

Soit G un graphe orienté. Les deux propositions suivantes sont alors équivalentes :

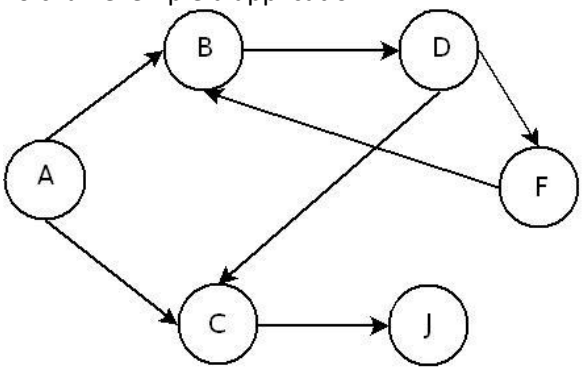
1) G est sans circuit

2) G et tous ses sous-graphes possèdent au moins un **point d'entrée (sommet sans prédécesseur)** et au moins un **point de sortie (sommet sans successeur)**

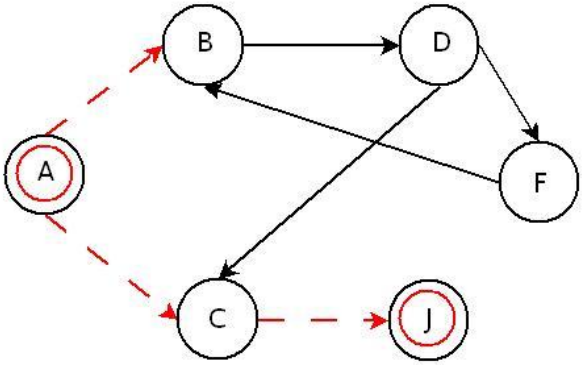
Un exemple d'écriture de l'algorithme de Marimont dans notre contexte est donc le suivant. Soit H le **sous-graphe de G qui contient tous ses sommets non encore marqués**, E l'ensemble des points d'entrée de H, et S l'ensemble des points de sortie de H.

H ← G // à l'initialisation, aucun sommet de G n'est marqué
E ← Ensemble des points d'entrée de H
S ← Ensemble des points de sortie de H
TANTQUE (H a des sommets ET E et S ne sont pas vides) FAIRE
    On marque tous les sommets appartenant à E
    On marque tous les sommets appartenant à S
    H ← Sous-graphe de H qui contient tous les sommets non marqués
    E ← Ensemble des points d'entrée de ce nouveau H
    S ← Ensemble des points de sortie de ce nouveau H
FINTANTQUE
L'algorithme rend vrai si et seulement si le dernier sous graphe H n'a plus de sommets

Voici un exemple d'application :

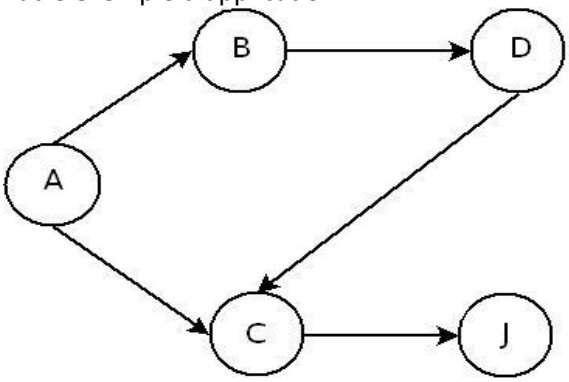


H <-- G
E <-- {A}
S <-- {J}

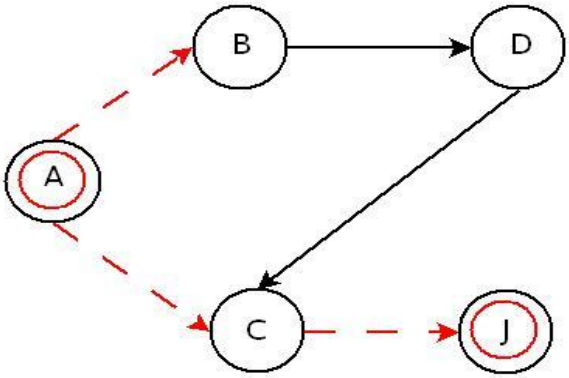


H <-- H \ {A,J}
E <-- {}
S <-- {C}
Fin de l'itération (E est vide)
H a encore des sommets
Il y a donc au moins un circuit

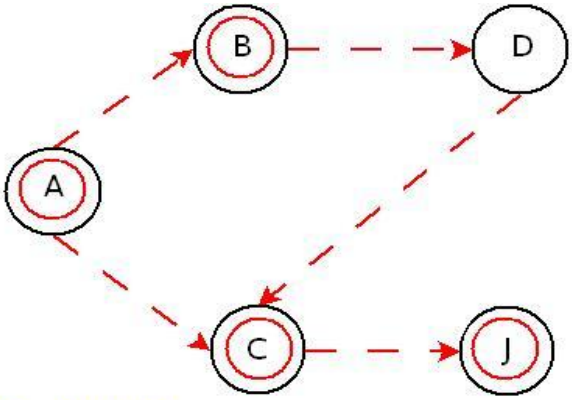
Autre exemple d'application :



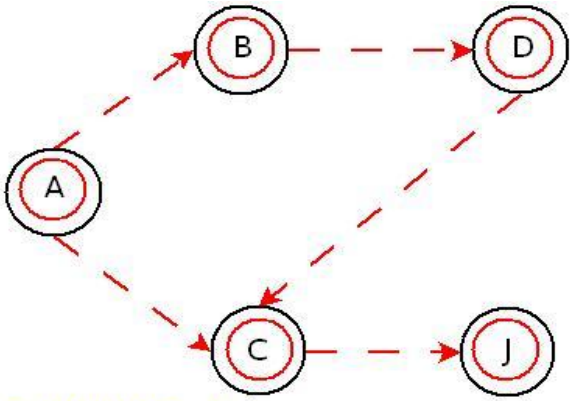
H <-- G
E <-- {A}
S <-- {J}



H <-- H \ {A,J}
E <-- {B}
S <-- {C}



H <-- H \ {B,C}
E <-- {D}
S <-- {D}



H <-- H \ {D} <-- {}
E <-- {}
S <-- {}
Fin de l'itération
H n'a plus de sommets
Il n'y a donc pas de circuit

QUESTIONS :

2-1) Écrire l'opération d'extension qui construit le sous-graphe constitué de tous les sommets non marqués du graphe sur lequel on a appelé l'opération.

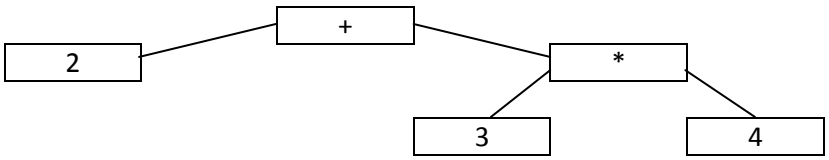
2-2) Écrire l'opération d'extension qui construit l'ensemble des points d'entrée du graphe sur lequel on a appelé l'opération.

2-3) Écrire l'opération d'extension qui construit l'ensemble des points de sortie du graphe sur lequel on a appelé l'opération.

2-4) Écrire l'opération d'extension qui exécute l'algorithme de Marimont. Le résultat sera booléen. "Vrai" signifiera que le graphe sur lequel on a appelé l'opération contient au moins un circuit.

3) Résolution d'un problème (9pts)

On considère une expression numérique simple. On suppose que cette expression n'est formée que de constantes numériques et des quatre opérateurs +, -, *, /. Il s'agit dans ce problème de transformer cette expression sous forme d'un arbre (voir schéma ci-dessous) puis de l'évaluer.
L'expression 2+3 * 4 se transforme



On modélise ce problème à l'aide d'un nouveau type abstrait nommé ExpSim_ExpArb.
On définit les opérations de base comme suit :

1. Une opération nommée **creerExp** qui crée un nouvel objet en recevant en paramètre l'expression simple sous forme d'un vecteur de Termes (indicé de 1 à ???). L'expression doit être syntaxiquement correcte.
2. Une opération nommée **recNbTermes** qui permet de récupérer le nombre de termes de l'expression simple
3. Une opération nommée **recTerme** qui permet de récupérer le $n^{ième}$ de l'expression simple.
4. Une opération nommée **insererTermes** qui reçoit en paramètres un entier r, un terme de type opérateur et un terme de type opérande. Cette opération insère dans l'expression simple l'opérateur et l'opérande juste après le $r^{ième}$ terme de l'expression simple.

QUESTIONS :

3-1) Donner la signature de ces opérations avec les pré-conditions et les axiomes.

3-2) Ecrire l'opération d'extension nommée **evaluerExp** qui permet d'évaluer l'expression simple en utilisant sa représentation sous forme d'arbre. Dans cette question, on suppose que l'opération nommée **recExpArb** qui permet de récupérer l'expression sous forme d'arbre est déjà écrite. On donnera en justifiant la complexité.

3-3) On suppose maintenant qu'en plus des constantes on peut avoir des variables dans l'expression. Proposer une autre définition des opérations de base qui intègre cette nouvelle possibilité.

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 Terme
Modèlisation 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