Cartouche du document

Année : ING 1
Matière : Algorithmique I
Activité : Travail dirigé

Objectifs

L'objectif porte sur le concept de type abstrait, les conteneurs Vecteur, Liste, Pile, File, Map, Arbre, Graphe et des algorithmes associés.

Tous les documents sont interdits.

La durée de l'examen est de 2 heures.

Le rendu est une copie papier.

Sommaire des exercices

Corps des exercices

1 - Eléments chimiques

Énoncé :

Les éléments chimiques ont été classifiés dans le tableau de Mendeleiev. Dans ce tableau, les éléments chimiques sont répartis dans des groupes et des périodes. Il y a 18 groupes et 7 périodes. De plus, un élément chimique est décrit par un symbole, un nom et un numéro (numéro atomique). Un numéro est compris entre 1 et 118.

Exemple du Lithium

- symbole : LI
- nom : Lithium
- numéro : 3

Pour représenter un élément chimique, on utilisera le type abstrait suivant :

```
TYPE ABSTRAIT ElementChimique
Concept
Ce type abstrait modélise un élément chimique du tableau de
Mendeleiev
Opérations de base
Constructeur ElementChimique : creerEC(symbole Chaine, nom Chaine,
numero Entier, groupe Entier, periode Entier) : ElementChimique
Observateur ElementChimique : recSymbole() : Chaine
Observateur ElementChimique : recNom() : Chaine
Observateur ElementChimique : recNumero() : Entier
Observateur ElementChimique : recGroupe() : Entier
```

```
Observateur ElementChimique : recPeriode() : Entier
```

Question 1)

Énoncé de la question

barême 1pt : Définir les préconditions (quand il y en a) de ces opérations en ne tenant compte que des informations données dans le texte.

Solution de la question

définie(creerEC(symb,nom,numero,groupe,periode))⇒ 1 <= numero <= 118

définie(creerEC(symb,nom,numero,groupe,periode)) ⇒ 1 <= periode <= 7

définie(creerEC(symb,nom,numero,groupe,periode)) ⇒ 1 <= groupe <= 18

Question 2)

Énoncé de la question

Définir les axiomes de ces opérations.

Solution de la question

barême 1pt

Deux solutions sont acceptables :

1) Dans le constructeur, les caractéristiques de l'objet créé sont des copies des paramètres.

```
creerEC(symb,nom,numero,groupe,periode).recSymbole().estEgal(symb)
creerEC(symb,nom,numero,groupe,periode).recNom()estEgal(nom)
creerEC(symb,nom,numero,groupe,periode).recNumero().estEgal(numero)
creerEC(symb,nom,numero,groupe,periode).recGroupe().estEgal(groupe)
creerEC(symb,nom,numero,groupe,periode).recPeriode().estEgal(periode)
```

2) Dans le constructeur, les références des caractéristiques de l'objet créé sont égales aux paramètres.

```
creerEC(symb,nom,numero,groupe,periode).recSymbole() = symb
creerEC(symb,nom,numero,groupe,periode).recNom() = nom
creerEC(symb,nom,numero,groupe,periode).recNumero() = numero
creerEC(symb,nom,numero,groupe,periode).recGroupe() = groupe
creerEC(symb,nom,numero,groupe,periode).recPeriode() = periode
```

Question 3)

Énoncé de la question

1) Ecrire une opération qui reçoit une liste d'éléments chimiques et qui supprime de cette liste les éléments chimiques dupliqués. On considère que deux éléments chimiques sont identiques quand leurs symboles sont égaux.

2) Etablir la complexité de cette opération.

Solution de la question

barême 3pts

```
Transformateur Liste suppSymbolesDupliques()
```

```
Transformé lec
Références locales
l1, l2 Liste
symb Chaine
Début
 l1 <-- lec
 Tantque non l1.estVide() Faire
  l2 <-- l1.reste()
  symb <-- l1.tete().recSymbole()
  Tantque non l2.estVide() Faire
   Si l2.tete().recSymbole().estEgal(symb) Alors
    l2.supprimer()
   Sinon
    l2 <-- l2.reste()
   Fin Si
  Fin Tantque
 Fin Tantque
retourner lec
Fin
```

Question 4)

Énoncé de la question

Dans le tableau de Mendeleiev, le symbole est un identifiant d'un élément chimique. On vous demande d'écrire une opération qui reçoit le tableau de Mendeleiev sous forme d'une liste d'éléments chimiques et renvoie une map constituée de ces éléments. Chaque élément de la map est défini comme suit :

- la clé est le symbole de l'élément chimique.
- la valeur est le quadruplet (nom,numero,groupe,periode)

On doit envisager que la liste puisse contenir des éléments dupliqués comme définis dans la question précédente.

Solution de la question

barême 3pts

Il faut d'abord définir un type abstrait pour gérer le quadruplet (nom,numero,groupe,periode).

```
TYPE ABSTRAIT ElementChimiquePartiel
Concept
Ce type abstrait modélise un élément chimique du tableau de
Mendeleiev à l'exception du symbole.
Opérations de base
Constructeur ElementChimiquePartiel : creerECP(nom Chaine, numero
Entier, groupe Entier, periode Entier) : ElementChimiquePartiel
Observateur ElementChimiquePartiel : recNom() : Chaine
Observateur ElementChimiquePartiel : recNumero() : Entier
```

```
Observateur ElementChimiquePartiel : recGroupe() : Entier
Observateur ElementChimiquePartiel : recPeriode() : Entier

Constructeur Map creerMapSymboles(Liste symboles)
Références locales
mec Map
symb Chaine
ec ElementChimique
ecp ElementChimiquePartiel
Debut
mec <-- mapVide()
Tantque non symboles.estVide() Faire
 ec <-- symboles.tete()
 symb <-- ec.recSymbole()
 Si non mec.existeCle(symb) Alors
  ecp <--
  creerECP(ec.recNom(),ec.recNumero(),ec.recGroupe(),ec.recPeriode())
  mec.ajouter(symb,ecp)
 Fin Si
 symboles <-- symboles.reste()
Fin Tantque
retourner mec
Fin
```

2 - Accessibilité

Énoncé :

Soit G un graphe orienté de sommets S et d'arêtes A. Soir R(u) l'ensemble des sommets accessibles à partir de u, avec u ∈ S. On définit min(u) comme étant le plus petit numéro des sommets de R(u) (on rappelle que dans le type Abstrait Graphe les sommets sont numérotés de 1 à n avec n = |S|).

Question 1)

Énoncé de la question

Ecrire un algorithme permettant de calculer min(u) pour un sommet u donné.

Solution de la question

barême 3pts

On a écrit (en td) les opérations de parcours dans un graphe à partir d'un sommet donné. On prend les versions qui sont des transformateurs du graphe. Ces transformateurs marquent les sommets accessibles à partir de u. Dans notre solution on utilisera le parcours en largeur.

```
Observateur Graphe recMinIndiceAccessible(u Entier) : Entier
Observé gr
Références locales
min, i Entier
```

```
Debut
 gr.parcoursEnLargeur(u)
 n <-- gr.recNbSommets()
 min <-- n + 1
 Pour i <-- 1 à n pas 1
  Si gr.estMarque(i) et i < min Alors
   min <-- i
  Fin Si
 Fin Pour
 retourner min
Fin
```

Fin

Question 2)

Énoncé de la question

En déduire un algorithme qui calcule min(u) pour tous les sommets $u \in S$.

Solution de la question

barême 2pts

```
Observateur Graphe recVectMinIndiceAccessible() : Vecteur
Observé gr
Références locales
i, n Entier
vMin Vecteur
Debut
 n <-- gr.recNbSommets()
 vMin <-- creerVecteur(1,n)
 Pour i <-- 1 à n pas 1
  vMin.affVal(i,gr.recMinIndiceAccessible(i))
 Fin Pour
 retourner vMin
Fin
```

Question 3)

Énoncé de la question

Evaluer la complexité de votre algorithme. Justifier votre réponse. (indication : il est possible de trouver un tel algorithme en O(|S| + |A|)).

Solution de la question

barême 3pts

Dans l'opération recMinIndiceAccessible, on a fait :

- un parcours en largeur qui a une complexité en O(|S| + |A|);

• une recherche de minimum dans un vecteur de sommets. Dans le pire des cas on a une complexité en O(|S|)

L'opération recMinIndiceAccessible a donc une complexité en O(|S| + |A|).

L'opération recVectMinIndiceAccessible appelle l'opération recMinIndiceAccessible autant de fois qu'il y a de sommets. L'opération recVectMinIndiceAccessible a donc une complexité en $O(|S|^2 + |S|.|A|)$.

3 - Algorithme de Prim

Énoncé :

L'algorithme de Prim est une variante de l'algorithme du calcul de l'arbre couvrant de poids minimum (comme Kruskal), qui s'inspire de la technique utilisée pour calculer le plus court chemin via l'algorithme de Dijkstra. Le principe de l'algorithme est le suivant :

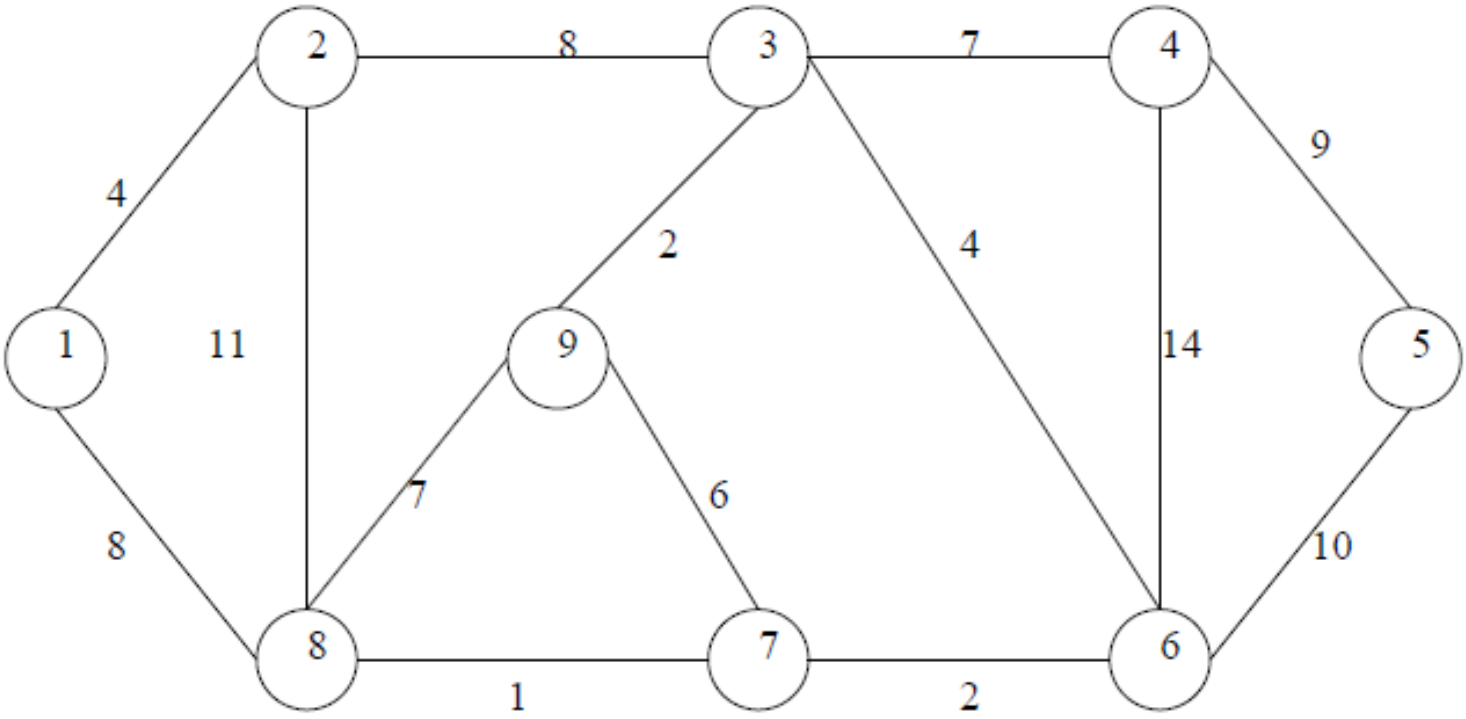
- Partir d'un sommet arbitraire r.
- A chaque étape, ajouter une arête minimale pour relier l'arbre en construction à un sommet isolé.
- L'algorithme termine lorsqu'il n'y a plus de sommet isolé.

Nous vous proposons d'associer un vecteur de distances aux sommets et de marquer à chaque fois les sommets qui ne sont plus isolés. Pour chaque sommet u, distance(u) est le poids minimal d'une arête reliant u à un sommet de l'arbre en construction. Par convention, distance(u) = ∞ si une telle arête n'existe pas. Un vecteur pere(u) désignant le parent de u dans l'arbre sera également mis à jour dans l'algorithme.

Question 1)

Énoncé de la question

On considère le graphe suivant :



Compléter le vecteur distance qui suit jusqu'à la fin de l'exécution qui part du sommet 1 :

Sommets	1	2	3	4	5	6	7	8	9
Initialisation	0	∞	∞	∞	∞	∞	∞	∞	∞
Itération n° 1	0	**4**	∞	∞	∞	∞	∞	8	∞
Itération n° 2	0	4	8	∞	∞	∞	∞	8	∞
Itération n° 3	0	4	8	7	∞	4	∞	8	2

Solution de la question

barême 1pt

Itération	1	2	3	4	5	6	7	8	9	Pivot
j = 0	0	∞	∞	∞	∞	∞	∞	∞	∞	-
j = 1	0	**4**	∞	∞	∞	∞	∞	8	∞	2
j= 2	0	4	**8**	∞	∞	∞	∞	8	∞	3
j = 3	0	4	8	7	∞	4	∞	8	**2**	9
j = 4	0	4	8	7	∞	**4**	6	7	2	6
j = 5	0	4	8	7	10	4	**2**	7	2	7
j = 6	0	4	8	7	10	4	2	**1**	2	8
j = 7	0	4	8	**7**	10	4	2	1	2	4
j = 8	0	4	8	7	**9**	4	2	1	2	5

Question 2)

Énoncé de la question

Dans ce qui suit on suppose qu'il existe dans le type abstrait Vecteur, une opération **v.recIndiceMin(gr Graphe)**. Chaque indice du vecteur v est un numéro de sommet du graphe gr. On considère le sous ensemble S' de ces indices constitué des sommets non marqués. Cette opération renvoie parmi les indices de S' l'indice de la case qui contient la plus petite valeur du vecteur v.

Rappel : MAXINT représente le plus grand entier que l'on peut définir dans la machine.

Compléter l'algorithme de Prim suivant :

```
Observateur Graphe algoPrim(r: Entier):Vecteur
Observe: GP
References locales:
 i,j,k,u,v : Entier
 G : Graphe
 succ,parent,distance : Vecteur
debut
 G <-- copie(GP)
 distance <-- creerVecteur(1,G.recNbSommets())
 parent   <-- creerVecteur(1,G.recNbSommets())
 distance <-- distance.affVal(r,0)
 k <-- 1
 Tantque k <= G.recNbSommets() Faire
  distance.affVal(k,MAXINT)
  k <-- k+1
 Fin Tantque
 j <-- 1
 Tantque j <= G.recNbSommets() Faire
  //u <-- distance.recIndiceMin()
  u <--  distance.recIndiceMin (G)
  G <-- G.marquer(u)
  succ <-- G.recSuccesseurs(u)
  i <-- 1
  Tantque i <= succ.borneSup() Faire
   // A Completer
  Fin Tantque
  j <-- j+1
 Fin Tantque
 retourner pere
fin
```

Solution de la question

barême 3pts

```
Observateur Graphe algoPrim(r: Entier):Vecteur
Observe: GP
References locales:
 i,j,k,u,v : Entier
 G : Graphe
 succ,parent,distance : Vecteur
debut
```

```
 G <-- copie(GP)
 distance <-- creerVecteur(1,G.recNbSommets())
 parent <-- creerVecteur(1,G.recNbSommets())
 distance <-- distance.affVal(r,0)
 k <-- 1
 Tantque k <= G.recNbSommets() Faire
  distance.affVal(k,MAXINT)
  k <-- k+1
 Fin Tantque
 j <-- 1
 Tantque j <= G.recNbSommets() Faire
  //u <-- distance.recIndiceMin()
  u <--  distance.recIndiceMin (G)
  G <-- G.marquer(u)
  succ <-- G.recSuccesseurs(u)
  i <-- 1
  Tantque i <= succ.borneSup() Faire
   v<--succ.recVal(i)
   Si non(G.estMarque(v)) et gr.recArete(u,v).recValuation()<
   distance.recVal(v) Alors
    pere.affVal(v,u)
    distance.affVal(v,gr.recArete(u,v)).recValuation())
   Fin Si
   i <-- i+1
  Fin Tantque
  j <-- j+1
 Fin Tantque
 retourner pere
fin
```

Annexes

```
Type abstrait Vecteur
Constructeur Vecteur : creerVecteur(Entier bi, Entier bs) :
Vecteur
Transformateur Vecteur : affVal(Entier i,Element e) : Vecteur
Observateur Vecteur : recVal(Entier i) : Entier
Observateur Vecteur : estInitialise(Entier i) : Booleen
Observateur Vecteur : borneInf() : Entier
Observateur Vecteur : borneSup() : Entier

Type abstrait Liste
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 Map
Constructeur Map : mapVide() : Map
Transformateur Map : ajouter(c Element, v Element) : Map
Transformateur Map : supprimer(c Element) : Map
Observateur Map : recVal(c Element) : Element
Observateur Map : existeCle(c Element) : Booleen
Observateur Map : existeValeur(v Element) : Booleen
Observateur Map : cardinal() : Entier
Observateur Map : recCles() : Liste

Type abstrait Graphe
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
Opération d'extension
Observateur Graphe : recSuccesseurs(Entier s) : Vecteur
```