Algorithmique et bases de la programmation

Durée : *1h30*
Documents autorisés : **AUCUN**

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 sur votre copie.

1 Compilation d'un projet C (3 points)

Soit un projet C de création d'un correcteur orthographique composé des fichiers suivants :
- dans le répertoire *include* : *mot.h*, *collection.h*, *ensemble.h*, *ensembleDeMots.h*, *dictionnaire.h*, *correcteur.h* ;
- dans le répertoire *src* : *mot.c*, *ensemble.c*, *ensembleDeMots.c*, *dictionnaire.c*, *correcteur.c*, *main.c*, *motTU.c*, *dictionnaireTU.c*, *correcteurTU.c*.

Nous voulons que :
- les résultats de la compilation des *.c* soient dans le répertoire *src* ;
- l'exécutable, de nom *asispell*, soit créé dans le répertoire *bin* ;
- deux librairies statiques *libdictionnaire.a* et *libcorrecteur.a* soient créées dans le répertoire *lib* ;
- que les tests unitaires soient créées dans le répertoire *tests*.

Donnez l'ensemble des commandes *gcc* et *ar* qui, depuis le répertoire racine du projet, permet d'obtenir ce résultat (la compilation devra afficher tous les *warning* et vérifier que le code C est ISO).

Solution proposée :

```
$ gcc -o src/mot.o -Iinclude -Wall -pedantic src/mot.c // idem avec tous les .c
$ ar -r lib/libdictionnaire.a mot.o ensemble.o ensembleDeMots.o
dictionnaire.o
$ ar -r lib/libcorrecteur.a correcteur.o
$ gcc -o bin/asispell -Llib main.o -ldictionnaire
-lcorrecteur
$ gcc -o tests/motTU -Llib motTU.o -ldictionnaire // pour dictionnaire
$ gcc -o tests/motTU -Llib motTU.o -ldictionnaire -lcorrecteur
```

2 Recherche dichotomique (6 points)

1. Écrire une fonction itérative, rechercheDichotomique, qui détermine par dichotomie le plus **grand** indice d'un élément, (dont on est sûr de la présence) dans un tableau d'entiers t (de n entiers significatifs) triés dans l'ordre croissant. Il peut y avoir des doublons (ou plus) dans le tableau. Vous justifierez vos choix.
2. Même question mais cette fois en récursif. Votre algorithme est récursif terminal ou non terminal ? Justifiez.

Solution proposée :

fonction rechercheDichotomique (t : **Tableau**[1..MAX] **d'Entier** ; n : **NaturelNonNul** ; element : **Entier**) : **NaturelNonNul**

⌊**précondition(s)** $\exists\ 1 \leq i \leq n$, t[i] = element

Déclaration d,f,m : **NaturelNonNul**

debut
d ← 1
f ← n
m ← (d + f) div 2
tant que f-d > 1 **faire**
si t[m] = element **alors**

```
            d ← m
        sinon
            si t[m] < element alors
                d ← m+1
            sinon
                f ← m-1
            finsi
        finsi
        m ← (d + f) div 2
    fintantque
    si t[f]=e alors
        retourner f
    sinon
        retourner d
    finsi
fin
fonction rechercheDichotomiqueR (t : Tableau[1..MAX] d'Entier ; d,f : NaturelNonNul ; element : Entier)
: NaturelNonNul
    Déclaration  m : NaturelNonNul
debut
    si f-d > 1 alors
        m ← (d + f) div 2
        si t[m] = element alors
            retourner rechercheDichotomiqueR(m,f)
        sinon
            si t[m] < element alors
                retourner rechercheDichotomiqueR(m+1,f)
            sinon
                retourner rechercheDichotomiqueR(d,m-1)
            finsi
        finsi
    sinon
        si t[f]=e alors
            retourner f
        sinon
            retourner d
        finsi
    finsi
fin
```

C'est un algorithme récursif terminal

3 L'algorithme des $k-moyennes$ (11 points)

Soit un ensemble de n points. L'objectif de l'algorithme des $k-moyennes$ ($k-means$ en anglais) est de regrouper ces n points dans k ensembles S_i de points similaires (k est un paramètre de l'algorithme). On dit qu'un point pt_1 est similaire à un point pt_2 si la distance entre pt_1 et pt_2 est faible.

La figure 1 montre l'application de l'algorithme des $k-moyennes$ sur un ensemble de 15 points avec $k=3$. On obtient ici trois ensembles S_1, S_2 et S_3 de respectivement 6 points, 4 points et 5 points.

Dans cet algorithme les ensembles S_i sont caractérisés par leur isobarycentre g_i. Il y a deux étapes principales :

1. l'association de chaque point pt à un ensemble S_i en fonction de l'isobarycentre g_i le plus proche de pt (à l'issue de cette étape, tous les points appartiennent donc à un ensemble S_i)
2. le calcul des nouveaux isobarycentres g'_i de chacun des ensembles S_i

Lorsque le système se stabilise (les isobarycentres sont toujours les mêmes), on a regroupé les points pt en k ensembles S_i de points optimalement similaires.

De manière plus détaillée et plus formel, le principe de cet algorithme est le suivant :

1. Choisir aléatoirement k points g_i (avec $i \in 1..k$) dans la zone couverte par les points pt_j donnés (avec $j \in 1..n$). L'ensemble G contient les k isobarycentres g_i.

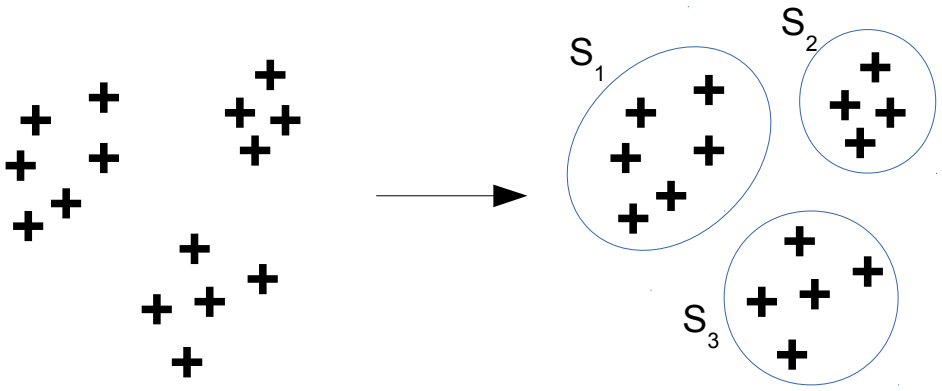


FIGURE 1 – Application de la l'algorithme des $k-moyennes$ (avec $k=3$)

2. Répartir chaque point pt_j dans l'ensemble S_i dont l'isobarycentre g_i est le plus proche de pt_j.
3. Calculer les isobarycentre g'_i (formant l'ensemble G') des ensembles S_i.
4. Si les ensembles de points G et G' sont identiques, alors les ensembles S_i sont la solution au problème. Sinon, on recommence depuis l'étape 2.

3.1 Utilisation du TAD ensemble (2 points)

Nous allons utiliser le TAD Ensemble vu en cours :

Nom: Ensemble
Paramètre: Element
Utilise: **Booleen**,**Naturel**
Opérations:

ensemble:	$\rightarrow$ Ensemble
ajouter:	Ensemble $\times$ Element $\rightarrow$ Ensemble
retirer:	Ensemble $\times$ Element $\rightarrow$ Ensemble
estPresent:	Ensemble $\times$ Element $\rightarrow$ **Booleen**
cardinalite:	Ensemble $\rightarrow$ **Naturel**
union:	Ensemble $\times$ Ensemble $\rightarrow$ Ensemble
intersection:	Ensemble $\times$ Ensemble $\rightarrow$ Ensemble
soustraction:	Ensemble $\times$ Ensemble $\rightarrow$ Ensemble

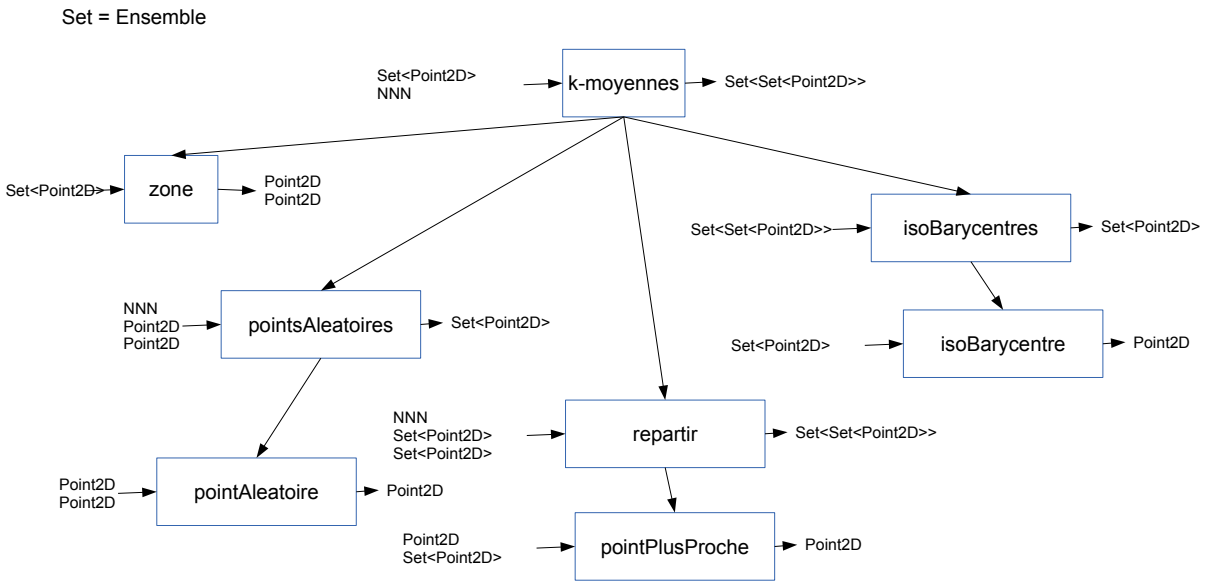
Axiomes:
- ajouter(ajouter(s,e),e)=ajouter(s,e)
- retirer(ajouter(s,e),e)=s
- estPresent(ajouter(s,e),e)
- non estPresent(retirer(s,e),e)
- cardinalite(ensemble())=0
- cardinalite(ajouter(s,e))=1+cardinalite(s) et non estPresent(s,e)
- cardinalite(ajouter(s,e))=cardinalite(s) et estPresent(s,e)
- ...

Donnez les signatures des fonctions et procédures permettant d'utiliser ce TAD.

3.2 Analyse du problème (4 points)

On considère posséder le TAD *Point2D* vu en cours. À partir du principe algorithmique exposé précédemment, donnez l'analyse descendante du problème sachant qu'il prendra en entrée un ensemble de points et un naturel non nul et qu'il calculera un ensemble d'ensembles de points.

Solution proposée :



3.3 Conception préliminaire (2 point)

Proposez les signatures des fonctions et procédures de votre analyse descendante.

3.4 Conception détaillée (3 points)

Donnez l'algorithme de la fonction ou procédure se trouvant au sommet de votre analyse descendante.

Solution proposée :

fonction k-moyennes (pts : Set<Point2D>, k : **NaturelNonNul) :** Set<Set<Point2D>>

Déclaration pt1,pt2 : Point2D
clusters : Set<Set<Point2D>>

debut
zone(pts,pt1,pt2)
G' ← pointsAleatoire(k,pt1,pt2)
repeter
G ← G'
clusters ← repartir(k,pts,G)
G' ← isoBarycentres(clusters)
jusqu'a ce que G=G'
retourner clusters
fin