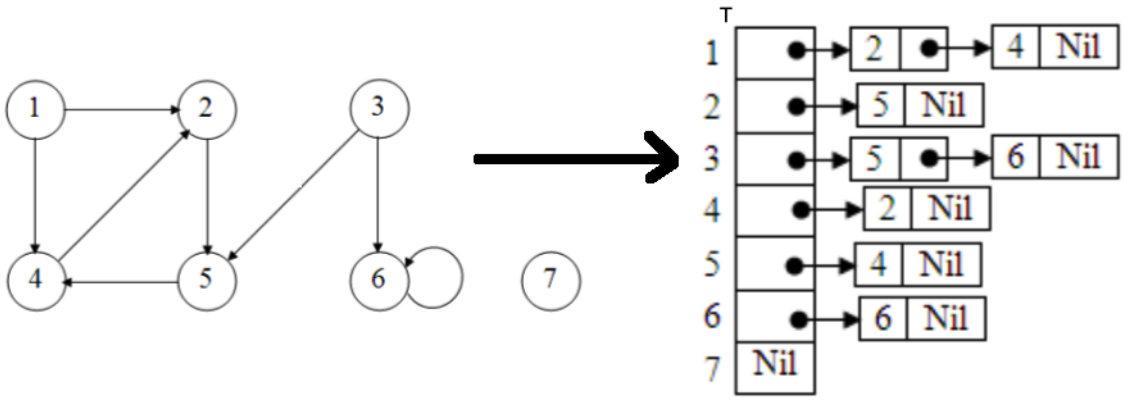
Université Mohamed Khider-Biskra
Faculté des Sciences Exactes et Sciences de la Nature et de la Vie
Département d'informatique

$2^{ème}$ année LMD — Algorithmique et Structures de données
26 Jan 2019 — 08:00-09:30

Examen de rattrapage

Exercice 1 Graphes (6 pts : 1 + 3 + 2)
Soit un graphe représenté par des listes d'adjacence dans un tableau de pointeurs T comme schématisé ci-dessous :



1. Donner la déclaration des structures de données nécessaires à la représentation d'un tel graphe en mémoire.
2. Écrire une procédure permettant de parcourir le graphe en profondeur d'abord.
3. Écrire une fonction Chemin(i,j) permettant de retourner vrai s'il existe un chemin entre i et j et faux sinon.

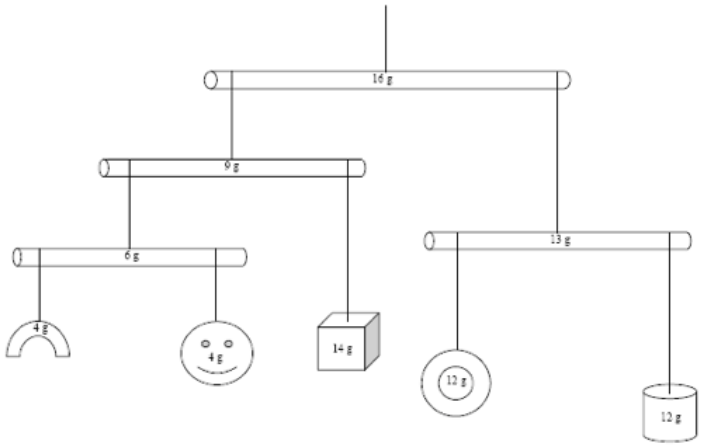
Exercice 2 Structures en table (6 pts : 1 + 2.5 + 2.5)
On souhaite maintenir en mémoire un dictionnaire Français-Anglais de N mots sous forme d'un tableau de deux champs : le mot en français et sa traduction en Anglais.

Mot Fr	Traduction Ang
Ordinateur	Computer
Université	University
:	:
:	:
Mémoire	Memory

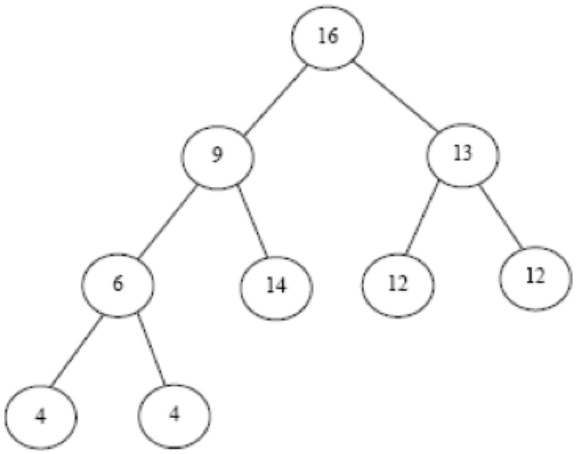
Pour accélérer la recherche dans la table, on utilise la méthode de Hashcoding avec une fonction de Hachage H. Et pour gérer les collisions, on utilise la méthode de chaînage interne séparé sur M enregistrements supplémentaires.

1. Donner la déclaration des structures de données nécessaires à la représentation d'un tel dictionnaire en mémoire ainsi que leur initialisation.
2. Écrire la procédure Insérer(MotFr,MotAng) permettant d'isérer un mot avec sa traduction dans le dictionnaire.
3. Écrire la procédure Supprimer(MotFr,MotAng) permettant de supprimer un mot avec sa traduction du dictionnaire.

Exercice 3 Arbres binaires (8 pts : 2 + 3 + 1)
On considère un arbre binaire représentant un mobile : chaque feuille correspond à un objet accroché à l'une de ses extrémités, et chaque nœud intermédiaire correspond à une barre de support. L'information associée à chaque nœud est son poids en grammes.
Exemple de mobile :



L'arbre binaire correspondant à ce mobile est le suivant :



On vous demande d'écrire les procédures et fonctions suivantes :

1. La fonction **Poids(M)** qui retourne le poids d'un mobile M donné.
2. La fonction **EstEquilibré(M)** qui retourne vrai si en accrochant M est un mobile équilibré. On dit qu'un mobile est équilibré si le poids de son sous-arbre gauche est égale au poids de son sous arbre droit et qui sont eux même équilibrés (le mobile de la figure est équilibré).
3. La fonction **ResteEquilibréG(M,x)** qui retourne vrai si le mobile équilibré M reste équilibré après l'ajout de l'objet de poids x à son sous arbre gauche ; et faux sinon.

★★★ Bonne chance ★★★

Corrigé type

Exercice 1 Graphes (6 pts : 1 + 3 + 2)

1. Déclaration des structures de données nécessaires à la représentation du graphe en mémoire :

```
Type  TMaillon = structure
        Val : Entier ;
        Suivant : Pointeur(TMaillon) ;
Fin ;
Var  Graphe : Tableau[1..N] de Pointeur(TMaillon) ;
```

2. Procédure permettant de parcourir le graphe en profondeur d'abord :

```
Procédure ParcoursProfondeur();
var  Visité : Tableau[1..N] de booléen ;
              i : entier ;
Procédure DFS( x : entier);
Début
    Visité[x]← Vrai ;
    Ecrire(x) ;
    P← Graphe[x] ;
    Tant que (P ≠ Nil) faire
        Si (Visité[Valeur(P)]=Faux) Alors
            DFS(Valeur(P)) ;
        Fin Si;
        P← Suivant(P) ;
    Fin TQ;
Fin;
Début
    Pour i de 1 à N faire
        Visité[i]← Faux ;
    Fin Pour;

    Pour i de 1 à N faire
        Si (Visité[i]=Faux) Alors
            DFS(i) ;
        Fin Si;

    Fin Pour;


Fin;
```

3. Fonction Chemin(i,j) permettant de retourner vrai s'il existe un chemin entre i et j et faux sinon :

```
Fonction Chemin( i,j : entier) : booléen;
var  Visité : Tableau[1..N] de booléen ;
Procédure DFS( x : entier);
Début
    Visité[x]← Vrai ;
    P← Graphe[x] ;
    Tant que (P ≠ Nil) faire
        Si (Visité[Valeur(P)]=Faux) Alors
            DFS(Valeur(P)) ;
        Fin Si;
        P← Suivant(P) ;
    Fin TQ;
Fin;
Début
    Pour i de 1 à N faire
        Visité[i]← Faux ;
    Fin Pour;
    DFS(i) ;
    Chemin ← Visité[j] ;
Fin;
```

Exercice 2 Structures en table (6 pts : 1 + 2.5 + 2.5)
On souhaite maintenir en mémoire un dictionnaire Français-Anglais de N mots sous forme d'un tableau de deux champs : le mot en français et sa traduction en Anglais.

Mot Fr	Traduction Ang
Ordinateur	Computer
Université	University
:	:
:	:
Mémoire	Memory

Pour accélérer la recherche dans la table, on utilise la méthode de Hashcoding avec une fonction de Hachage H. Et pour gérer les collisions, on utilise la méthode de chaînage interne séparé sur M enregistrements supplémentaires.

1. Donner la déclaration des structures de données nécessaires à la représentation d'un tel dictionnaire en mémoire ainsi que leur initialisation.

```
Type  Mot = structure
        MotFr,MotAng : Chaine de caractères ; // initialisés à ”
        Suivant : entier ; // initialisé à -1
Fin ;
Var  Dictionnaire : Tableau[1..N] de Mot ;
```

2. Écrire la procédure Insérer(MotFr,MotAng) permettant d'isérer un mot avec sa traduction dans le dictionnaire.

```
Procédure Insérer( MotFr,MotAng : Chaine de caractères);
var  i,K : entier
Début
    i← H(MotFr) ;
    Tant que (Dictionnaire[i].MotFr≠MotFr et Dictionnaire[i].Suivant ≠ -1) faire
        i ← Dictionnaire[i].Suivant ;
    Fin TQ;
    Si (Dictionnaire[i].MotFr=MotFr) Alors
        Ecrire('Le mot existe déja') ;
    Sinon
        Si (Dictionnaire[i].MotFr=") Alors
            Dictionnaire[i].MotFr ← MotFr ;
            Dictionnaire[i].MotAng ← MotAng ;
        Sinon
            K ← N-M ;
            Tant que (K ≤ N et Dictionnaire[K].Mot ≠ " ) faire
                K ← K + 1 ;
            Fin TQ;
            Si (K > N) Alors
                Ecrire('Le mot ne peut être inséré') ;
            Sinon
                Dictionnaire[K].MotFr ← MotFr ;
                Dictionnaire[K].MotAng ← MotAng ;
                Dictionnaire[i].Suivant ← K ;
            Fin Si;
        Fin Si;
    Fin Si;

Fin;
```

3. Écrire la procédure Supprimer(MotFr,MotAng) permettant de supprimer un mot avec sa traduction du dictionnaire.

```
Procédure Supprimer( MotFr : Chaine de caractères);
var  i,K, Preci : entier
Début
    i← H(MotFr) ;
    Preci ← -1 ;
    Tant que (i ≠ -1 et Dictionnaire[i].MotFr≠MotFr) faire
        Preci ← i ;
        i ← Dictionnaire[i].Suivant ;
    Fin TQ;
    Si (i = -1) Alors
        Ecrire('Le mot n'existe pas') ;
    Sinon
        Tant que (Dictionnaire[i].Suivant ≠ -1) faire
            K ← Dictionnaire[i].Suivant ;
            Dictionnaire[i].MotFr ← Dictionnaire[K].MotFr ;
            Dictionnaire[i].MotAng ← Dictionnaire[K].MotAng ;
            Preci ← i ;
            i ← K ;
        Fin TQ;
        Dictionnaire[i].MotFr ← '' ;
        Si (Preci ≠ -1 ) Alors
            Dictionnaire[Preci].Suivant = -1 ;
        Fin Si;
    Fin Si;
Fin;
```

Exercice 3 Arbres binaires (8 pts : 3 + 3 + 2)

```
Type  TNoeud = structure
        Valeur : entier ;
        FG,FD : Pointeur(TNoeud) ;
Fin ;
Var  Dictionnaire : Tableau[1..N] de Mot ;
```

1. Fonction **Poids(M)** :

```
Fonction Poids( M : Pointeur(TNoeud)) : Entier;
Début
    Si (M=Nil) Alors
        Poids ← 0 ;
    Sinon
        Poids ← Valuer(M) + Poids(FG(M)) + Poids(FD(M)) ;
    Fin Si;
Fin;
```

2. Fonction **EstEquilibré(M)**

```
Fonction EstEquilibré( M : Pointeur(TNoeud)) : Booleén;
Début
    Si (M=Nil) Alors
        EstEquilibré ← Vrai ;
    Sinon
        EstEquilibré ← (Pois(FG(M))=Poids(FD(M))) et EstEquilibré(FG(M)) et EstEquilibré(FD(M)) ;
    Fin Si;
Fin;
```

3. Fonction **ResteEquilibréG(M,x)** :

```
Fonction ResteEquilibré( M : Pointeur(TNoeud), x : entier) : Booleén;
Début
    Si (x=0 ou M=Nil) Alors
        ResteEquilibré ← Vrai ;
    Sinon
        ResteEquilibré ← Faux ; ;
    Fin Si;
Fin;
```