

Contrôle continu 4

(3 pages)

Questions de cours (6 points)

- Dans une table de hashage, pourquoi peut-on ne pas avoir accès à une clé en $O(1)$
 - dans le cas d'adressage ouvert ?
Dans le cas de l'adressage ouvert, les clés sont stockées dans la table. Si il y a des collisions les éléments sont replacés en utilisant une fonction de sondage et en cherchant des places libres dans la table. Ainsi les éléments qui ne sont pas à « leur place » génèrent des collisions supplémentaires. Ainsi si la table est très pleine ou si les fonctions de hachage donne toujours le même résultat alors l'accès devient $O(n)$.
 - dans le cas d'adressage chaîné ?
Dans le cas de l'adressage chaîné, les clés sont stockées dans des listes extérieures à la table. Si il y a des collisions les éléments sont placés dans la même liste. Ainsi si la fonction de hachage donne toujours le même résultat alors l'accès devient $O(n)$.

- Peut-on avoir un arbre binaire de recherche dont les sommets sont des tas min? Si oui écrire une fonction permettant de comparer deux tas. Si non, expliquez pourquoi.
Oui puisque tout somme contient un objet quelconque. Cependant pour que cela soit un arbre binaire de recherche on doit pouvoir comparer deux tas (avoir un opérateur \leq). On peut par exemple comparer les deux valeurs des tas.

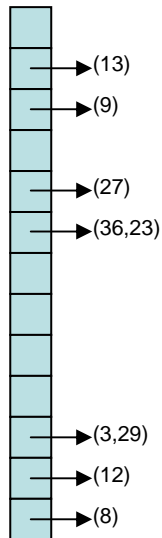
fonction $op :: \leq (ref\ t1, t2 : tas\ d'objets) : boolean ;$
début
 retourner(valeur(t1) <= valeur(t2))
fin

- On considère une table de hashage munie la fonction de hashage sur la clé x

$$h(x) = (3x + 1) \bmod M$$
 - Quelle sont les index possibles pour le tableau de la table de hashage ?
Les valeurs sont dans l'intervalle $[0..M-1]$

 - Quel problème génère le choix de $M=12$
Si $M=12$, pour les valeurs $(3x+1) \bmod 12$ on a l'ensemble de valeurs $\{1,4,7,10\}$. On ne couvrira donc pas tout l'intervalle possible. Cela provient du fait que 12 n'est pas un nombre premier.

 - On suppose que $M=13$, dessinez la structure de données, après la suite d'insertion suivante (23,13,8,12,27,9,29,36,3) dans le cas de l'adressage chaîné.



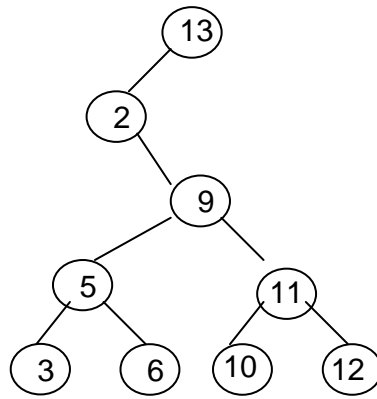
- On suppose que $M=13$, dessinez la structure de données, après la suite d'insertion suivante (23,13,8,12,27,9,29,36,3). On utilisera la fonction de sondage

$$h(x) = (5x + 1) \bmod M$$

36
13
9
27
23
3
29
12
8

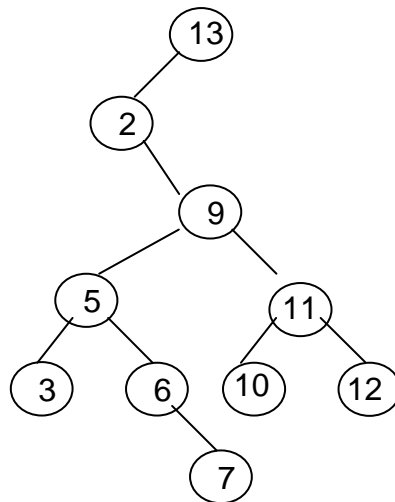
Exercice 2 (6 points)

- Soit l'arbre binaire de recherche suivant

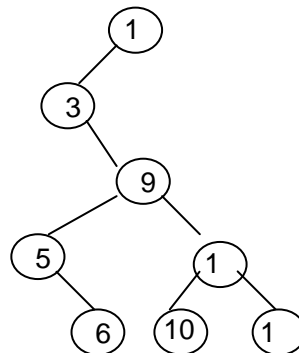


Pour chaque question, on considère que l'arbre à modifier est celui de la figure ci-dessus. On donnera si nécessaire les détails des opérations.

1. Donnez l'arbre après insertion de 7

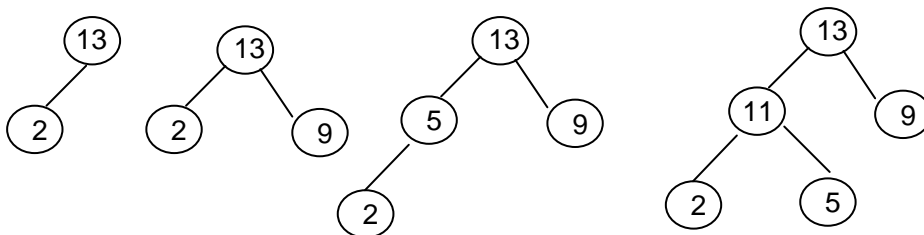


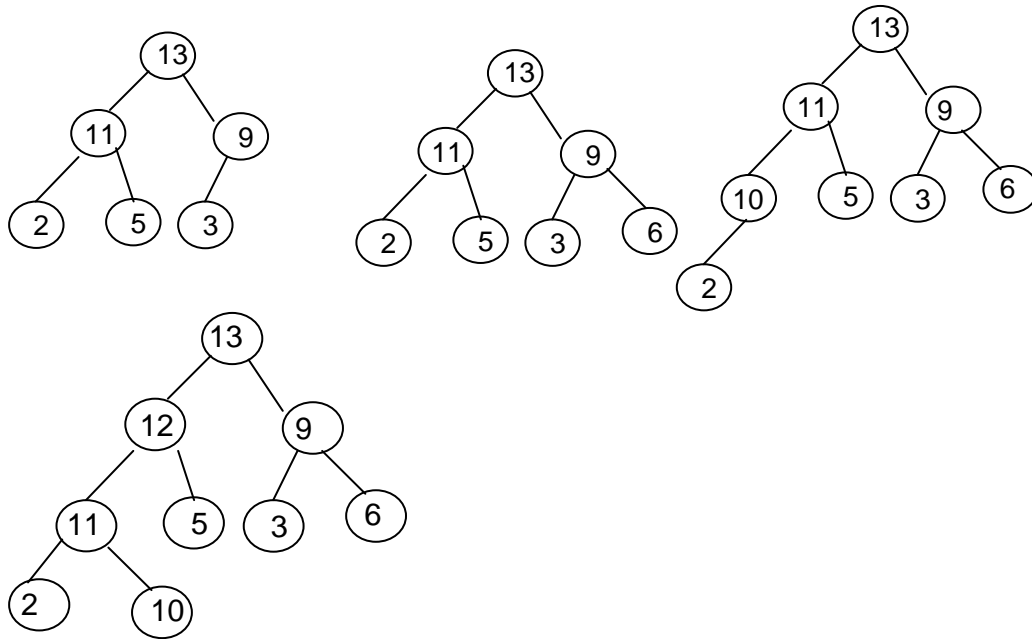
2. Donnez l'arbre après suppression de 2



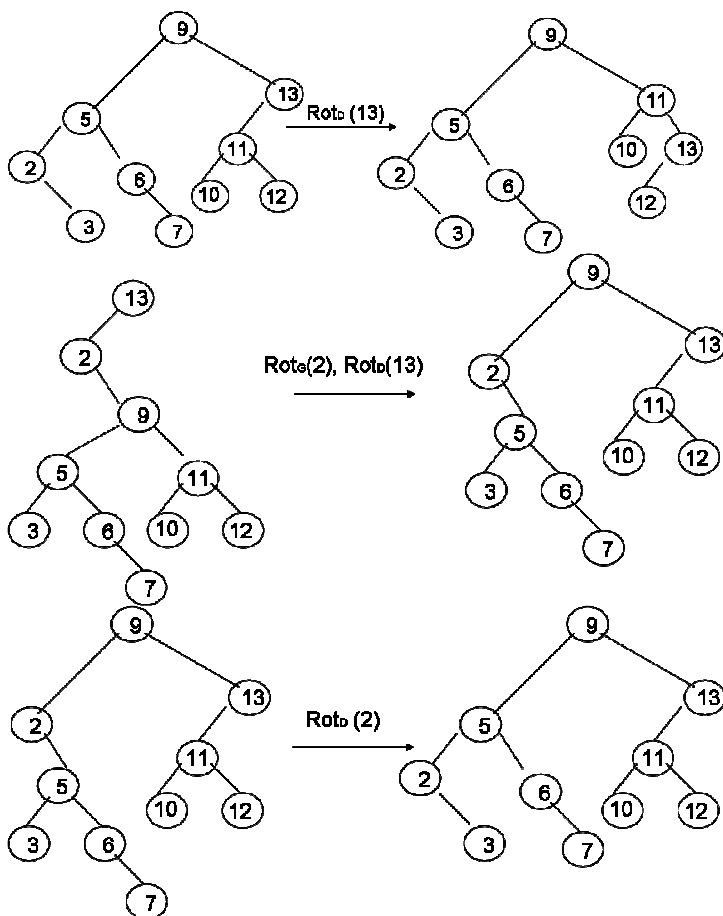
3. On considère la suite des clés dans l'ordre hiérarchique construire le tas max correspondant.

Clés dans l'ordre hiérarchique : 13,2,9,5,11,3,6,10,12



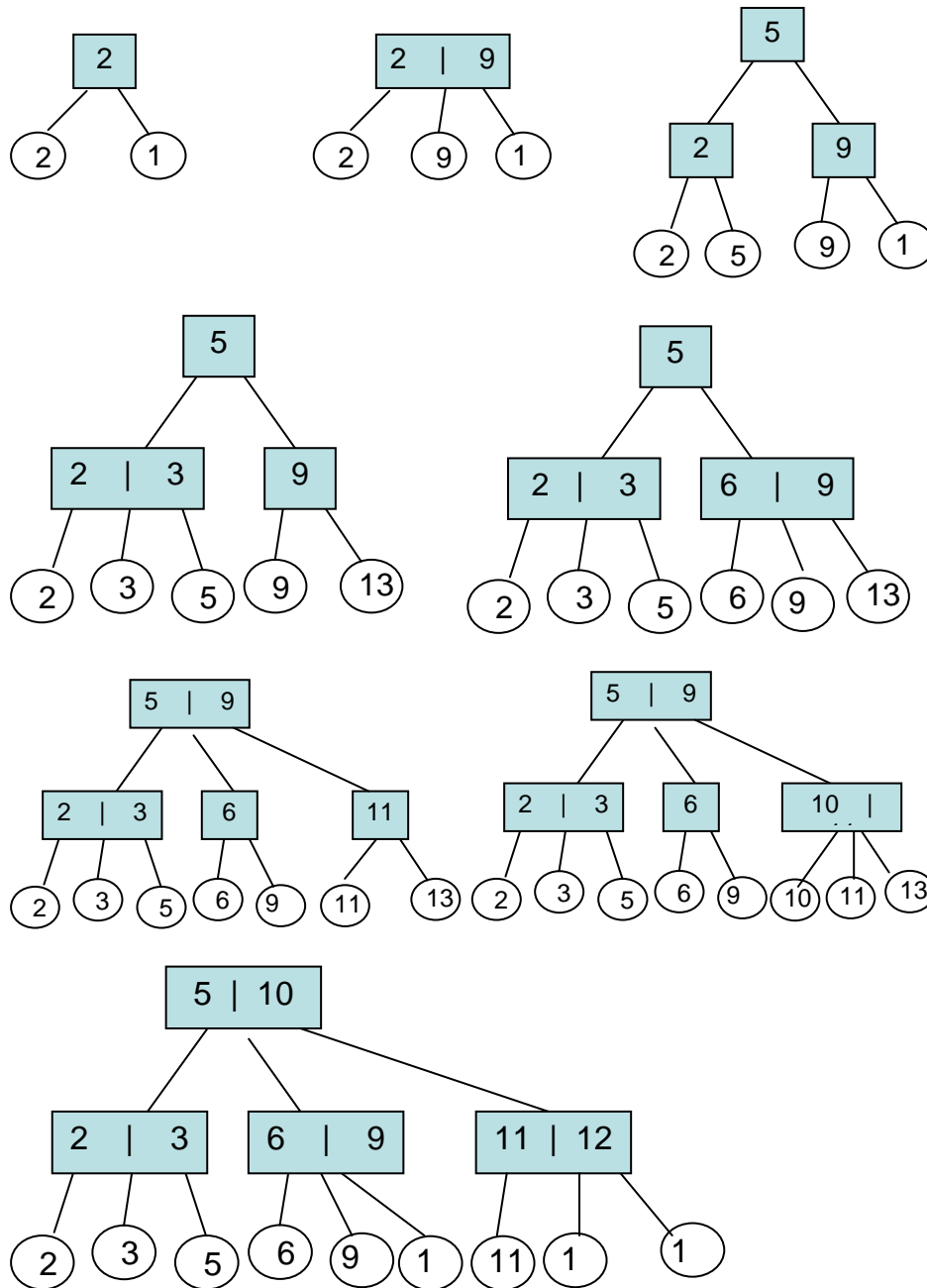


4. Equilibrez l'arbre binaire de recherche, on donnera les étapes et les rotations.



5. On considère la suite des clés dans l'ordre préfixe construire le 2-3 arbre correspondant.

Clé ordre préfixe : 13,2,9,5,3,6,11,10,12



Exercice 2 (8 points)

Un dispositif dispose de deux robots pour traiter des commandes. Une commande est identifiée par un nom sur 30 caractères et une date sous la forme [jour,mois,année]. Le premier robot (R1) mémorise les commandes dès qu'elles lui parviennent. Le second robot (R2) traite les commandes.

1 – Quelle structure de données pertinente peut-on utiliser pour accéder en temps constant aux identificateurs des commandes ?

Une table de hashage !

2 – Quelle structure de données peut utiliser R1 pour que R2 traite les commandes dans le même ordre où elles arrivent à R1 ?

Une file !

3 – Décrivez ces structures en détaillant leurs éléments.

Type FC : file de entier ;/ on stocke les valeurs de hashage*/*

L'entier dans la structure est une valeur de hachage de l'élément de commande.

Commande=structure

Code :entier ; /* obtenu par la fonction commandeToEntier */

Nom :tableau[1..30] de char ;

Date :structure

Jour :entier ;

Mois :entier ;

Année : entier ;

Finstructure

finstructure

Si on choisit un adressage ouvert on a donc une table de hachage de Commande. Si C est une commande la clé doit être calculée à partir de C.code. Il faut donc modifier les fonctions de hachage en conséquence..

4 – Ecrire les fonctions *ajouterCommande* (utilisée par R1) et *supprimerCommande* utilisé par R2.

Fonction *ajouterCommande*(ref F :FC ; ref T : tableHash de commande ;
ref C :commande) :vide

Début

Enfiler(F,ajouter(C,T))

Fin

Fonction *supprimerCommande*(ref F :FC ; ref T : tableHash de commande) :vide

Début

Supprimer(valeur(F),T) ;

Défiler(F)

Finsi

5 – Modifier la fonction *ajouterCommande* pour éviter les doublons.

Fonction *ajouterCommande*(ref F :FC ; ref T : tableHash de commande ;
ref C :commande) :vide

Début

Si chercher(T,C)==NIL alors

Enfiler(F,ajouter(C,T))

finsi

Fin

Listes simplement chaînées (listeSC)

fonction valeur(val L:liste d'objet):objet;

fonction debutListe(val L:liste d'objet);

fonction suivant(val L:liste d'objet);

fonction listeVide(val L:liste d'objet): boolean;

fonction créerListe(ref L:liste d'objet):vide;

fonction insérerAprès(ref L:liste d'objet; val x:objet;):vide;

fonction insérerEnTete(ref L:liste d'objet val x:objet):vide;

fonction supprimerAprès(ref L:liste d'objet):vide;

fonction supprimerEnTete(ref L:liste d'objet):vide;

Listes doublement chaînées (listeDC)

fonction finListe(val L:liste d'objet):vide;

fonction précédent(val L::liste d'objet): vide;

Piles

fonction valeur(ref P:pile de objet):objet;

```
fonction fileVide(ref P:pile de objet):booléen;  
fonction créerPile(P:pile de objet) :vide  
fonction empiler(ref P:pile de objet;val x:objet):vide;  
fonction dépiler(ref P:pile de objet):vide;  
fonction detruirePile(ref P:pile de objet):vide;
```

Files

```
fonction valeur(ref F:file de objet):objet;  
fonction fileVide(ref F:file de objet):booléen;  
fonction créerFile(F:file de objet);vide;  
fonction enfiler(ref F:file de objet;val x:objet):vide;  
fonction défiler (ref F:file de objet):vide;  
fonction detruireFile(ref F:file de objet):vide;
```

Arbres binaires

```
fonction getValeur(val S:sommet):objet;  
fonction filsGauche(val S:sommet):sommet;  
fonction filsDroit(val S:sommet):sommet;  
fonction pere(val S:sommet):sommet;  
fonction setValeur(ref S:sommet;val x:objet):vide;  
fonction ajouterFilsGauche(ref S:sommet,val x:objet):vide;  
fonction ajouterFilsDroit(ref S:sommet,x:objet):vide;  
fonction supprimerFilsGauche(ref S:sommet):vide;  
fonction supprimerFilsDroit(ref S:sommet):vide;  
fonction detruireSommet(ref S:sommet):vide;  
fonction créerArbreBinaire(val Racine:objet):sommet;
```

Arbres planaires

```
fonction valeur(val S:sommetArbrePlanaire):objet;  
fonction premierFils(val S:sommetArbrePlanaire):sommetArbrePlanaire;  
fonction frere(val S:sommetArbrePlanaire):sommetArbrePlanaire;  
fonction pere(val S:sommetArbrePlanaire):sommetArbrePlanaire;  
fonction créerArborescence(val racine:objet):sommetArbrePlanaire;  
fonction ajouterFils(ref S:sommetArbrePlanaire,val x:objet):vide;  
fonction supprimerSommet(ref S:sommetArbrePlanaire):vide;  
fonction créerArbreBPlaniare(val Racine:objet):sommet;
```

Arbres binaire de recherche

```
fonction ajouter(ref A:arbreBinaire d'objets, val v:objet):vide;  
fonction supprimer(val A: arbreBinaire d'objets, val v:objet):vide;
```

Tas

```
fonction valeur(ref T:tas d'objet): objet;  
fonction ajouter(ref T:tas de objet, val v:objet):vide;  
fonction supprimer(val T:tas de objet):vide;  
fonction creerTas(ref T:tas,val:v:objet):vide;  
fonction detruireTas(ref T:tas):vide;
```

Dictionnaire

```
fonction appartient((ref d:dictionnaire,val M::mot):booléen;  
fonction creerDictionnaire(ref d: dictionnaire):vide ;  
fonction ajouter(ref d:dictionnaire,val M::mot):vide;  
fonction supprimer(ref d:dictionnaire,val M:mot):vide;  
fonction detruireDictionnaire(ref d:dictionnaire):vide;
```

Table de Hachage

```
fonction chercher(ref T:tableHash de clés, val v:clé):curseur;
```

```
fonction créerTableHachage(ref T: tableHash de clé, ref h:fonction):vide;  
fonction ajouter(ref T:tableHash de clé,val x:clé):booleen;  
fonction supprimer((ref T:tableHash de clé,val x:clé):vide;
```