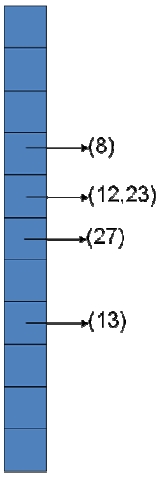
Contrôle continu 4
(3 pages)

Questions de cours (5 points)

- A quoi sert une table de hashage ? Pour n clés, comparez l'encombrement mémoire entre l'adressage ouvert et l'adressage chainé.
  à accéder à des clés plus rapidement. En général le temps moyen d'accès à n éléments est en O(n). L'adressage chainé stocke la clé à l'extérieur de la table dans une liste doublement chainée (2 pointeurs pour une clé). L'adressage ouvert stocke la clé à l'intérieur de la table. L'adressage chainé prend donc plus d'espace que l'adressage ouvert.
- Peut-on avoir un arbre binaire de recherche dont les sommets sont des tas Max? Si oui écrire une fonction permettant de comparer deux tas. Si non, expliquez pourquoi.
  Pour qu'un type d'objet soit la valeur d'un sommet d'ABR il faut que le type soit doté d'un ordre total. On peut comparer deux tas en comparant la valeur des racines donc un tas peut être la valeur d'un sommet.
  fonction op ::=<(ref T1,T2 : tas d'objet) :booleen ;
  retourner(valeur(T1)<valeur(T2))
- 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 index sont dans l'intervalle[0..M-1]
  - Quel problème génère le choix de M=8 ?
    Aucun car toutes les valeurs de l'intervalle sont atteintes.
  - On suppose que M=11, dessinez la structure de données, après la suite d'insertion suivante **(**23,13,8,12,27**)** dans le cas de l'adressage chainé.

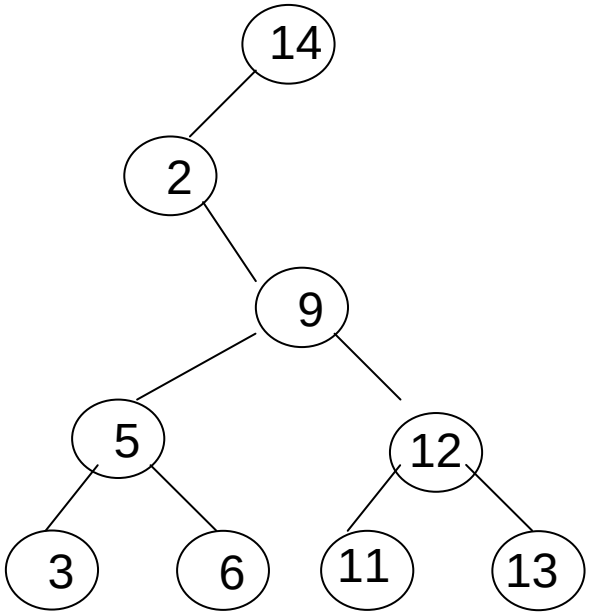


  - On suppose que M=11, dessinez la structure de données, après la suite d'insertion suivante **(**23,13,8,12,27**)**. On utilisera la fonction de sondage

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

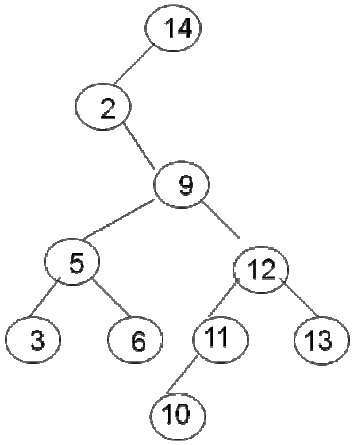
Exercice 2 (5 points)

- Soit l'arbre binaire de recherche suivant

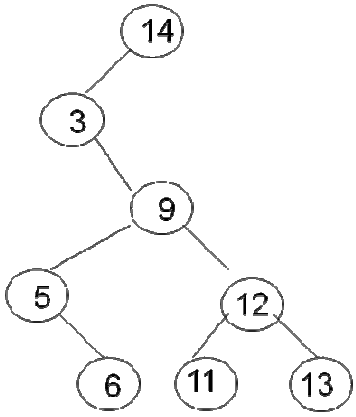


Pour les questions 1,2 on considère que l'arbre à modifier est celui de la figure ci-dessus. On donnera les détails des opérations.

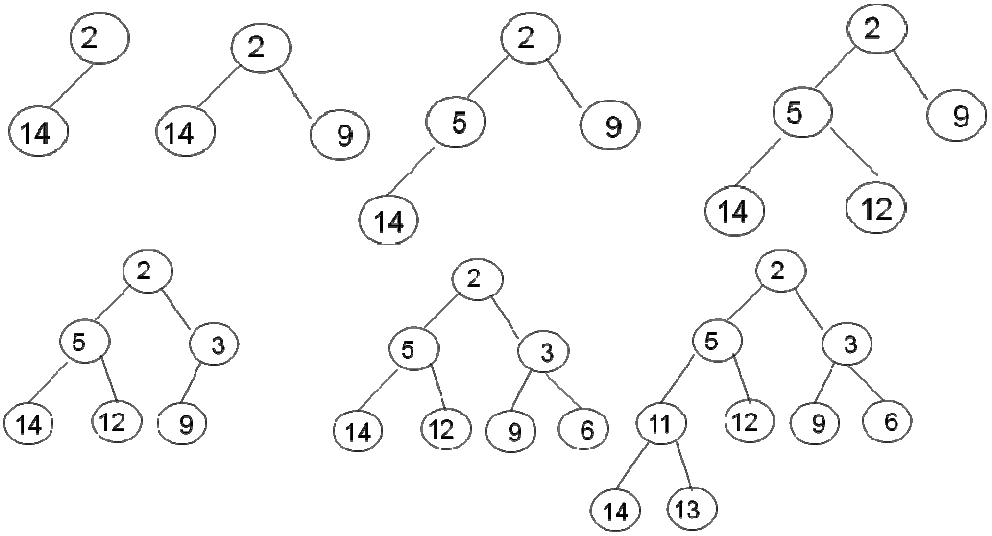
1. Donnez l'arbre après insertion de 10, on expliquera comment il est obtenu.
   On cherche un chemin dans l'arbre à partir de la racine en comparant la valeur du sommet avec la valeur à insérer. Si la valeur est supérieure (resp. <=) on descend sur le fils droit (resp. gauche). Lorsque le sommet n'a pas de fils, on créée un fils à droite (resp. à gauche).



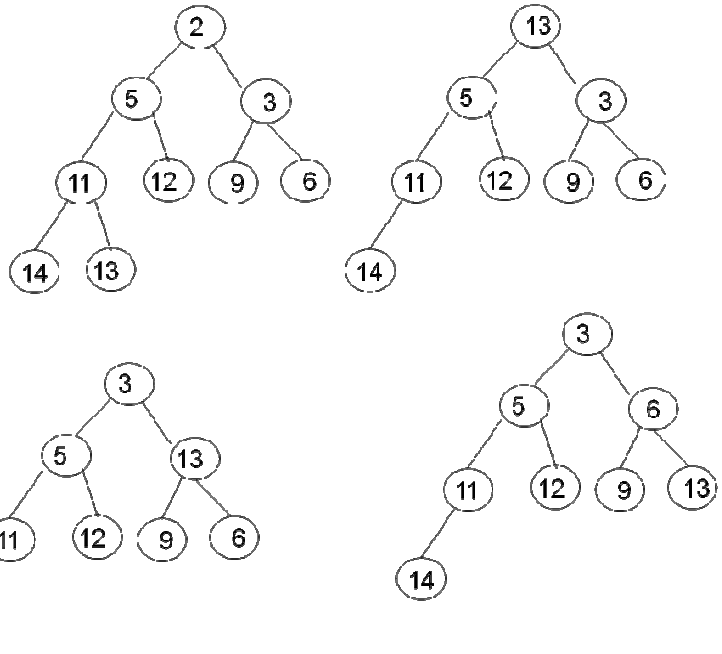
2. Donnez l'arbre après suppression de 2, on expliquera comment il est obtenu.
   On remonte à la place de 2 la plus petite des valeurs plus grandes que 2 soit 3 et on supprime la feuille.



3. On considère la suite des clés dans l'ordre hiérarchique construire le tas min correspondant. On dessinera le tas après chaque insertion d'une clé.
   Les valeurs dans l'ordre hiérarchique sur l'arbre de départ sont : 14,2,9,5,12,3,6,11,13 .



4. Montrer l'effet de la primitive supprimer sur le tas ainsi obtenu



Exercice 2 (4 points)

1 - Ecrire une fonction tasToListe qui construit une liste doublement chaînée L contenant tous les éléments d'un tas-Max T dans l'ordre croissant. On utilisera les primitives du type listes doublement chaînées et du type tas.

```
fonction tasToListe(val T : tas d'objet) :listeDC d'objet ;
  var L:listeDC d'objet;
  début
    creerListe(L) ;
    tantque valeur(T)!=NULL faire
      insererEnTete(L,valeur(T)) ;
      supprimer(T) ;
    fintantque
    retourner(L)
  fin
```

2 – Décrire les changements dans votre fonction si la liste est simplement chainée.

L'algorithme ci-dessus ne nécessite aucun changement.

Listes simplement chainées (listeSC)

fonction valeur(val L:liste d'objet):objet;
fonction debutListe(val L:liste d'objet) :vide ;
fonction suivant(val L:liste d'objet) :vide ;
fonction listeVide(val L:liste d'objet): booleen;
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 chainées (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;

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;

File de priorité

fonction changeValeur(ref T:tas d'objet,val s:sommet,val v:objet):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;