Algorithmique
Partiel n° 1

Info-Spé – Epita

D.S. 314060.4 BW (22 déc. 2009 - 10 :00)

Consignes (à lire) :

- □ Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
  - Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
  - Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
  - Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
  - Aucune réponse au crayon de papier ne sera corrigée.
- □ La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 et que les points de présentation (2 au maximum) sont retirés de cette note.
- □ **Les algorithmes :**
  - Tout algorithme doit être écrit dans le langage Algo (pas de C, Caml ou autre).
  - Tout code Algo non indenté ne sera pas corrigé.
  - Tout ce dont vous avez besoin (types, routines) est indiqué en **annexe** (page 5) !
- □ Durée : 2h00

Exercice 1 (CC – *3 pts*)

Soit le graphe `G=<S,A>` non orienté avec :

```
          S={1,2,3,4,5,6,7,8,9,10}
       et A={{1,3},{1,5},{1,6},{1,7},{2,4},{2,8},{3,5},{3,6},{3,7},
             {4,8},{4,9},{4,10},{6,7},{8,9},{8,10},{9,10}}
```

1. Représenter le graphe correspondant à `G`.
2. Donner le tableau `Degré` tel que $\forall\ i \in [1, Card(S)]$, `Degré[i]` soit égal au degré de `i` dans `G`.
3. Représenter (dessiner) la forêt couvrante associée au parcours en profondeur du graphe `G`. *Représenter aussi les autres arcs en les qualifiant à l'aide d'une légende.* On considérera le sommet 1 comme base du parcours et les successeurs en ordre croissant.

Exercice 2 (Largeur et principe – *6 pts*)

Écrire le principe et l'algorithme abstrait de la procédure `aff_arc_larg` correspondant à l'affichage des arcs rencontrés lors du parcours largeur d'un graphe orienté. Pour l'algorithme, vous ne ferez que la procédure de parcours qui traite des successeurs d'un sommet donné. Vous n'avez pas à donner la boucle itérative extérieure qui vérifie que tous les sommets aient été visités.

Exercice 3 (Arbres AA – *5 pts*)

Un *arbre AA* (Arne Andersson tree) est un dérivé des arbres bicolores, avec une propriété supplémentaire : les nœuds rouges ne peuvent être ajoutés qu'en fils droit. En d'autres termes, un nœud rouge ne peut pas être un fils gauche.

Les arbres AA sont donc une simulation des arbres 2-3 plutôt que des arbres 2-3-4 : il n'y a que des 2-nœuds et des 3-nœuds, les 3-nœuds étant représentés penchés à droite.

Ces arbres sont implémentés avec le type suivant :

```
types
    /* déclaration type t_element */
    t_aAA = ↑ t_noeudAA
    t_noeudAA = enregistrement
        t_element    cle
        t_aAA    fg, fd
        entier    niveau
    fin enregistrement t_noeudAA
```

où `niveau` représente le niveau, de bas en haut, dans l'arbre 2-3 correspondant (les feuilles au niveau 1). Deux clés appartenant au même 3-nœud de l'arbre 2-3 ont donc le même niveau (la plus grande clé, considérée comme rouge, étant dans le fils droit de la plus petite) dans l'arbre AA.

Par exemple, l'arbre 2-3 de la figure 1 sera représenté par l'arbre AA de la figure 2 : les clés "rouges" sont celles qui ont le même niveau que leur père (10, 8 et 14 dans l'exemple) et sont toujours à droite.

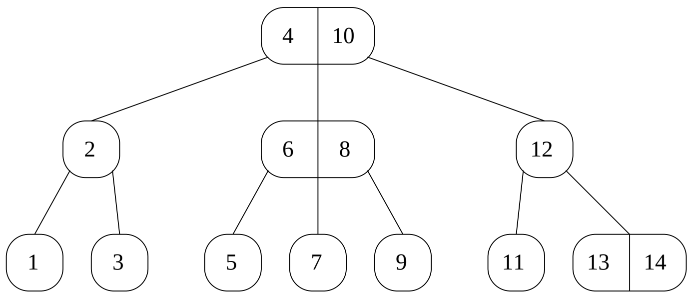


Fig. 1 – Arbre 2-3

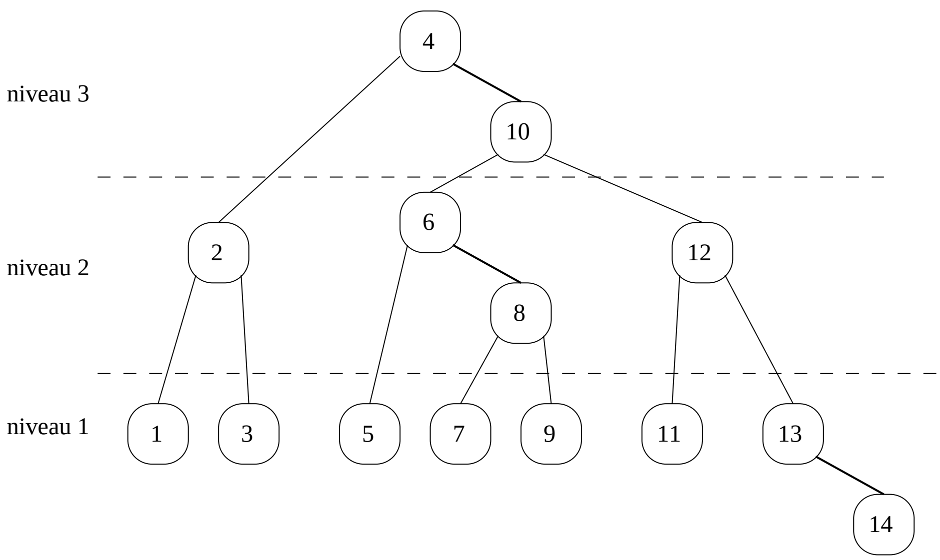


Fig. 2 – Arbre AA

Pour l'arbre AA, le niveau représente le nombre de liens gauches à suivre avant d'atteindre un fils vide :

– Le "niveau" de chaque feuille est donc 1.
– Tout nœud de niveau supérieur doit avoir au moins un fils droit de même niveau (considéré comme rouge) ou de niveau inférieur.
– Il ne peut pas avoir 3 nœuds de même niveau sur un même chemin (un nœud ne peut avoir le même niveau que son grand père).
– Un fils gauche est obligatoirement de niveau inférieur à son père (pas de fils rouge à gauche).

Il n'y a que deux transformations pour maintenir les propriétés de l'arbre après ajout ou suppression :

Skew : Une rotation droite utilisée lorsque une insertion ou une suppression crée un fils gauche rouge (le fils gauche a le même niveau que son père). Les deux nœuds conservent le même niveau.

Split Une rotation gauche utilisée lorsque l'on se retrouve avec deux nœuds rouges consécutifs (3 nœuds de même niveau consécutifs à droite). La racine voit son niveau incrémenté (correspond à un éclatement).

On supposera les procédures `skew` et `split`, prenant toutes les deux un arbre AA en paramètre, implémentées. Elles effectuent la rotation sur la racine de l'arbre passé en paramètre, ainsi que la mise à jour éventuelle des niveaux.

Le principe de l'insertion de la valeur x dans l'arbre AA A est le suivant :

– La descente se fait comme pour l'insertion d'une clé en feuille dans un arbre binaire de recherche. La nouvelle feuille est insérée au niveau 1. Si la valeur x est déjà présente, elle n'est pas insérée.

– En remontant :
  – de la droite : s'il a 3 nœuds de même niveau (toujours à droite), on effectue un "split". Par exemple, insérer la valeur 15 dans l'arbre de la figure 2.
  – de la gauche : il faut s'assurer que le fils gauche n'a pas le même niveau que le nœud actuel (par exemple si on insère 0 dans l'arbre 2, si c'est le cas on effectue un "skew". Dans ce cas uniquement, on vérifie que le petit-fils droit (après transformation) n'a pas le même niveau lui aussi, auquel cas, un "split" est nécessaire. Ce dernier cas peut-être illustré par l'ajout de $12,5$ dans l'arbre 2.

  Les transformations pouvant se propager jusqu'à la racine.

Deux exemples d'insertion sont donnés en annexe (page 6).

Question : Compléter la fonction `insert_AA` qui insère une clé dans un arbre AA, sauf si celle-ci est déjà présente. La fonction retourne un booléen indiquant si l'insertion a eu lieu.

Exercice 4 (Bipartite graph – *6 pts*)

Un graphe biparti est un graphe non orienté $G =< S, A, C >$, dans lequel S peut être partitionné en deux ensembles S_1 et S_2 tels que $(u, v) \in A$ implique soit que $u \in S_1$ et $v \in S_2$, soit que $u \in S_2$ et $v \in S_1$. Aucune arête ne doit relier deux sommets d'un même ensemble.

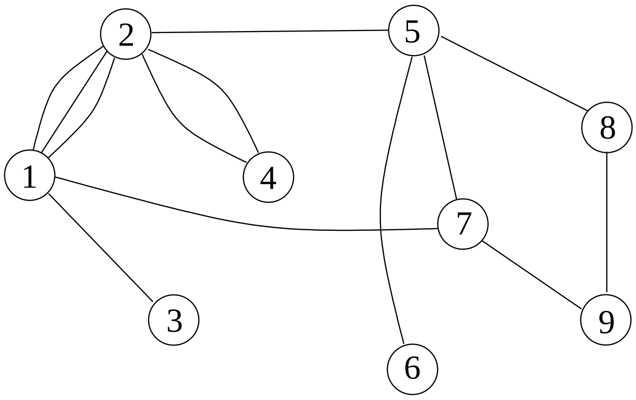


FIG. 3 – Graphe G_3

1. Le graphe de la figure 3 est-il biparti ? Si oui, donner les deux ensembles S_1 et S_2.
2. On veut tester si un graphe, en représentation dynamique, est biparti. Pour cela on utilise un **parcours profondeur**. La fonction ci-dessous est la fonction d'appel. Écrire la fonction `test_rec`.

Spécifications :

La fonction `biparti (t_graphe_d` G`)` retourne un booléen indiquant si le graphe non orienté G est biparti.

```
algorithme fonction biparti : booleen
    parametres locaux
        t_graphe_d        G

    variables
        t_vect_entiers   marque
        entier           i
        t_listsom        ps
debut
    pour i ← 1 jusqu'a G.ordre faire
        marque[i] ← 0
    fin pour
    ps ← G.lsom
    tant que ps <> NUL faire
        si marque[ps↑.som] = 0 alors
            marque[ps↑.som] ← 1
            si non test_rec (ps, marque) alors
                retourne faux
            fin si
        fin si
        ps ← ps↑.suiv
    fin tant que
    retourne vrai
fin algorithme fonction biparti
```

Annexes

Graphes orientés : opérations du type abstrait

SORTE Graphe

UTILISE Sommet, Entier, Booléen

OPERATIONS

graphe_vide	: → Graphe
ajouter-le-sommet _ à _	: Sommet × Graphe → Graphe
ajouter-l'arc <_,_> à _	: Sommet × Sommet × Graphe → Graphe
_ est-un-sommet-de _	: Sommet × Graphe → Booléen
<_,_> est-un-arc-de _	: Sommet × Sommet × Graphe → Booléen
d°+ de _ dans _	: Sommet × Graphe → Entier
_ ème-succ-de _ dans _	: Entier × Sommet × Graphe → Sommet
d°- de _ dans _	: Sommet × Graphe → Entier
_ ème-pred-de _ dans _	: Entier × Sommet × Graphe → Sommet
retirer-le-sommet _ de _	: Sommet × Graphe → Graphe
retirer-l'arc <_,_> de _	: Sommet × Sommet → Graphe

Files : opérations du type abstrait

SORTE
File

UTILISE
Booléen, Élément

OPÉRATIONS

file-vide	:	→ File
est-vide	:	File → Booléen
enfiler	:	File × Élément → File
défiler	:	File → File
premier	:	File → Élément

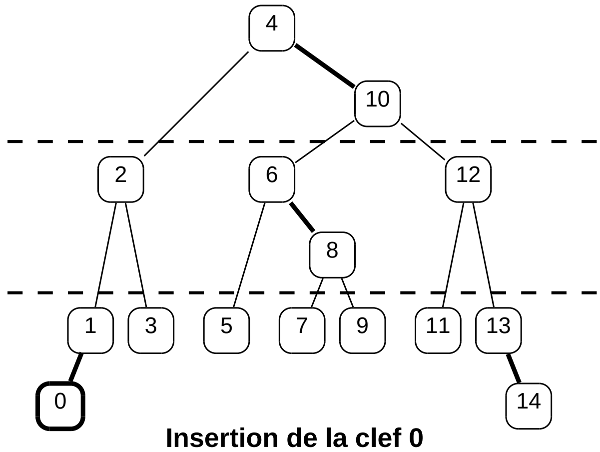
Représentation dynamique des graphes

```
types
    t_listsom = ↑ s_som
    t_listadj = ↑ s_ladj
    s_som     = enregistrement
        entier      som
        t_listadj   succ
        t_listadj   pred
        t_listsom   suiv
    fin enregistrement s_som
    s_ladj      = enregistrement
        t_listsom   vsom
        entier      nbliens
        t_listadj   suiv
    fin enregistrement s_ladj
    t_graphe_d  = enregistrement
        entier     ordre
        booleen    orient
        t_listsom  lsom
    fin enregistrement t_graphe_d
```

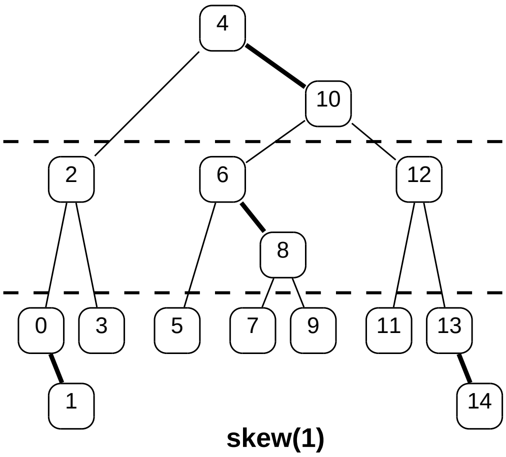
Autres types utiles

```
constantes
    Max = /* une valeur suffisante !*/
types
    t_vect_entiers  = Max entier
    t_vect_booleens = Max booleen
    t_vect_reels = Max reel
```

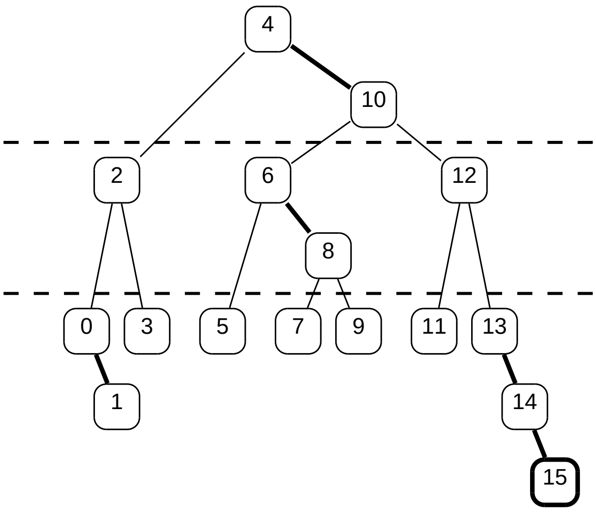
Exemples d'insertions dans un arbre AA



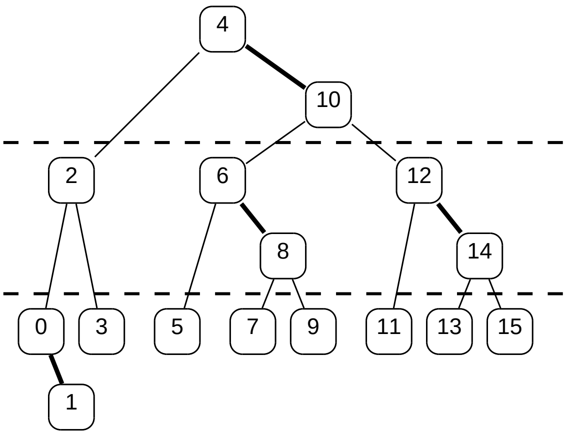
Insertion de la clef 0



skew(1)



Insertion de la clef 15



split(13)