Rappels relecture rapide du dernier cours

- Les applications des parcours :
  - parcours simples : préfixe, suffixe, infixe, étendus aux arbres N-aires
  - parcours en largeur ... enrichi sur des paires <noeuds,profondeur>
  - parcours avec valeur retour (contains, similaire, copie ...)
  - parcours et transmission d'arguments (chemin, liste des noeuds vus ...)
- Des applications pratiques :
  - Encodage d'un arbre dans un tableau
  - et réciproquement
  - Utilisation de listes d'arbres (génération des arbres de hauteur $\leq h$)

Université de Paris

Rappels relecture rapide du dernier cours

- Les applications des parcours :
  - parcours simples : préfixe, suffixe, infixe, étendus aux arbres N-aires
  - parcours en largeur ... enrichi sur des paires <noeuds,profondeur>
  - parcours avec valeur retour (contains, similaire, copie ...)
  - parcours et transmission d'arguments (chemin, liste des noeuds vus ...)
- Des applications pratiques :
  - Encodage d'un arbre dans un tableau
  - et réciproquement
  - Utilisation de listes d'arbres (génération des arbres de hauteur $\leq h$)
- Pas mal de choses à assimiler.
  Aujourd'hui : 3ème et dernière séance sur les arbres, exercices

Université de Paris

Rappels relecture rapide du dernier cours

- Les applications des parcours :
  - parcours simples : préfixe, suffixe, infixe, étendus aux arbres N-aires
  - parcours en largeur ... enrichi sur des paires <noeuds,profondeur>
  - parcours avec valeur retour (contains, similaire, copie ...)
  - parcours et transmission d'arguments (chemin, liste des noeuds vus ...)
- Des applications pratiques :
  - Encodage d'un arbre dans un tableau
  - et réciproquement
  - Utilisation de listes d'arbres (génération des arbres de hauteur $\leq h$)
- Pas mal de choses à assimiler.
  Aujourd'hui : 3ème et dernière séance sur les arbres, exercices
- Semaine prochaine : révision

Université de Paris

Initiation à la programmation Java
IP2 - Séance No 11

Yan Jurski

Université de Paris

Exercice 1

- Arbre **complet** : chaque niveau de l'arbre est complètement rempli.

Exercice

Ecrire une méthode qui détecte si un arbre binaire est complet ou non.

Université de Paris

Exercice 1

- Arbre **complet** : chaque niveau de l'arbre est complètement rempli.

Exercice

Ecrire une méthode qui détecte si un arbre binaire est complet ou non.

Fichier Arbre.java

```
public boolean estComplet(){
 if (racine==null) return true;
 else return racine.estComplet();
}
```

Université de Paris

Exercice 1

- Arbre **complet** : chaque niveau de l'arbre est complètement rempli.

Exercice

Ecrire une méthode qui détecte si un arbre binaire est complet ou non.

Fichier Arbre.java

```
public boolean estComplet(){
 if (racine==null) return true;
 else return racine.estComplet();
}
```

Fichier Noeud.java

```
public boolean estComplet(){
 if (fg==null && fd==null) return true;
 if (fg==null || fd ==null) return false;
 return ( fg.estComplet()
          && fd.estComplet()
          && fg.hauteur()==fd.hauteur() // on a déjà écrit hauteur()...
 ); // fin du return
}
```

Exercice 1 - seconde solution

- Arbre **complet** : chaque niveau de l'arbre est complètement rempli.

Exercice

Ecrire une méthode qui détecte si un arbre binaire est complet ou non.

- Il existe une relation hauteur v.s. nombre de noeuds . . .
  qu'on peut exploiter :

Université de Paris

Exercice 1 - seconde solution

- Arbre **complet** : chaque niveau de l'arbre est complètement rempli.

Exercice

Ecrire une méthode qui détecte si un arbre binaire est complet ou non.

- Il existe une relation hauteur v.s. nombre de noeuds . . .
  qu'on peut exploiter :

Fichier Arbre.java

```
public boolean estComplet(){
 return ( nbNoeud()==Math.pow(2,hauteur()) -1 ); // nbNoeud déjà écrite
}
```

Université de Paris

Exercice 1

- Arbre **complet** : chaque niveau de l'arbre est complètement rempli.

Exercice

Ecrire une méthode qui détecte si un arbre binaire est complet ou non.

- Mais il existe aussi une relation hauteur v.s. nombre de feuilles . . .

Université de Paris

Exercice 1

- Arbre **complet** : chaque niveau de l'arbre est complètement rempli.

Exercice

Ecrire une méthode qui détecte si un arbre binaire est complet ou non.

- Mais il existe aussi une relation hauteur v.s. nombre de feuilles . . .

Fichier Arbre.java

```
public boolean estComplet_vers2(){
 return ( nbFeuilles()==Math.pow(2,hauteur()-1) ); // écrire nbFeuilles
}
```

Université de Paris

Exercice 1

- Arbre **complet** : chaque niveau de l'arbre est complètement rempli.

Exercice

Ecrire une méthode qui détecte si un arbre binaire est complet ou non.

Fichier Arbre.java

```
public boolean estComplet_vers2(){
 return ( nbFeuilles()==Math.pow(2,hauteur()-1) ); // écrire nbFeuilles
}
public int nbFeuilles(){
 if (racine==null) return 0; else return racine.nbFeuilles();
}
```

Fichier Noeud.java

```
public int nbFeuilles(){
 if (fg==null && fd==null) return 1;
 int nb=0;
 if (fg != null) nb += fg.nbFeuilles();
 if (fd != null) nb += fd.nbFeuilles();
 return nb;
}
```

Exercice 2

Exercice

Quelle est la moyenne des profondeurs des feuilles d'un arbre binaire ?

- profondeur de la racine=1, profondeur d'un de ses fils=2 etc . . .

Université de Paris

Exercice 2

Exercice

Quelle est la moyenne des profondeurs des feuilles d'un arbre binaire ?

- profondeur de la racine=1, profondeur d'un de ses fils=2 etc ...

Il faut calculer $\sum_{f \in feuilles} profondeur(f)$ et diviser par $nbFeuilles$

Exercice 2

Exercice

Quelle est la moyenne des profondeurs des feuilles d'un arbre binaire ?

- profondeur de la racine=1, profondeur d'un de ses fils=2 etc ...

Il faut calculer $\sum_{feuilles\ f} profondeur(f)$ et diviser par *nbFeuilles*

Fichier Arbre.java

```
public double moyProfFeuilles(){
 if (racine==null) return 0;
 return ( racine.sumProfFeuilles(1) / nbFeuilles() );
}
```

- Décision : passage en argument de la profondeur courante

Université de Paris

Exercice 2

Exercice

Quelle est la moyenne des profondeurs des feuilles d'un arbre binaire ?

Fichier Arbre.java

```
public double moyProfFeuilles(){
 if (racine==null) return 0;
 return ( racine.sumProfFeuilles(1) / nbFeuilles() );
}
```

Fichier Noeud.java

```
public int sumProfFeuilles(int p){ // p : profondeur courante
 if (estFeuille()) return p;
 int rep=0;
 if (fg!=null) rep += fg.sumProfFeuilles(p+1);
 if (fd!=null) rep += fd.sumProfFeuilles(p+1);
 return rep;
}
```

Université de Paris

Arbres étendus - 1

- Dans le cas des listes simplement chaînées :
  - La notion de prédécesseur s'imposait, mais était non représentée
  - $\Rightarrow$ on a ajouté un attribut **Noeud précédent**
    - surcoût dans les opérations de maintien
    - gain sur certaines opérations complexes

Université de Paris

Arbres étendus - 1

- Dans le cas des listes simplement chaînées :
  - La notion de prédécesseur s'imposait, mais était non représentée
  - $\Rightarrow$ on a ajouté un attribut **Noeud précédent**
    - surcoût dans les opérations de maintien
    - gain sur certaines opérations complexes
- Cas des arbres :
  - La notion de père s'impose, mais n'est pas encore représentée
  - $\Rightarrow$ on peut ajouter un attribut **Noeud père**

Fichier Noeud.java

```
public class Noeud {
 private E content; // sans importance
 private Noeud filsG;
 private Noeud filsD;
 private Noeud père;
}
```

Université de Paris

Arbres étendus - 1

- Dans le cas des listes simplement chaînées :
  - La notion de prédécesseur s'imposait, mais était non représentée
  - $\Rightarrow$ on a ajouté un attribut **Noeud précédent**
    - surcoût dans les opérations de maintien
    - gain sur certaines opérations complexes
- Cas des arbres :
  - La notion de père s'impose, mais n'est pas encore représentée
  - $\Rightarrow$ on peut ajouter un attribut **Noeud père**

Fichier Noeud.java

```
public class Noeud {
 private E content; // sans importance
 private Noeud filsG;
 private Noeud filsD;
 private Noeud père;
}
```

- Attention à la cohérence au moment de la construction !

Université de Paris

Arbres étendus - Application

Exercice

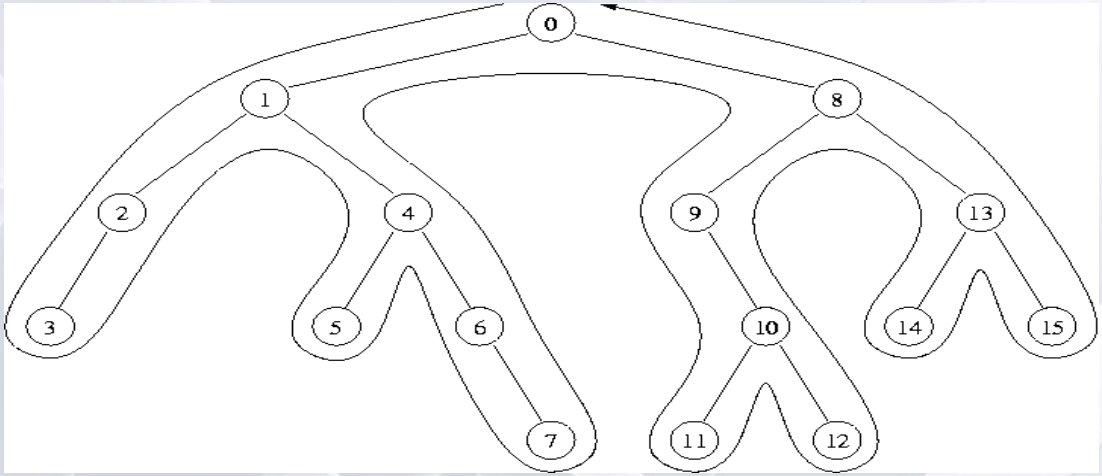
Soit m un mot composé des lettres 'g' , 'd' ou 'u' (pour up)
Ecrivez une méthode d'arbre, qui affiche le contenu porté par le nœud obtenu en suivant le chemin indiqué par m et en partant de la racine.

Université de Paris

Arbres étendus - Application

Exercice

Soit *m* un mot composé des lettres 'g' , 'd' ou 'u' (pour up)
Ecrivez une méthode d'arbre, qui affiche le contenu porté par le nœud obtenu en suivant le chemin indiqué par *m* et en partant de la racine.



- **afficheOrienté("ddudugduudd")** donne 15

Arbres étendus - Application

Exercice

Soit m un mot composé des lettres 'g' , 'd' ou 'u' (pour up)
Ecrivez une méthode d'arbre, qui affiche le contenu porté par le nœud obtenu en suivant le chemin indiqué par m et en partant de la racine.

Fichier Arbre.java

```
public void afficheOrienté (String dir){
 if (dir==null || racine ==null) return;
 racine.afficheOrienté(dir);
}
```

Université de Paris

Arbres étendus - Application

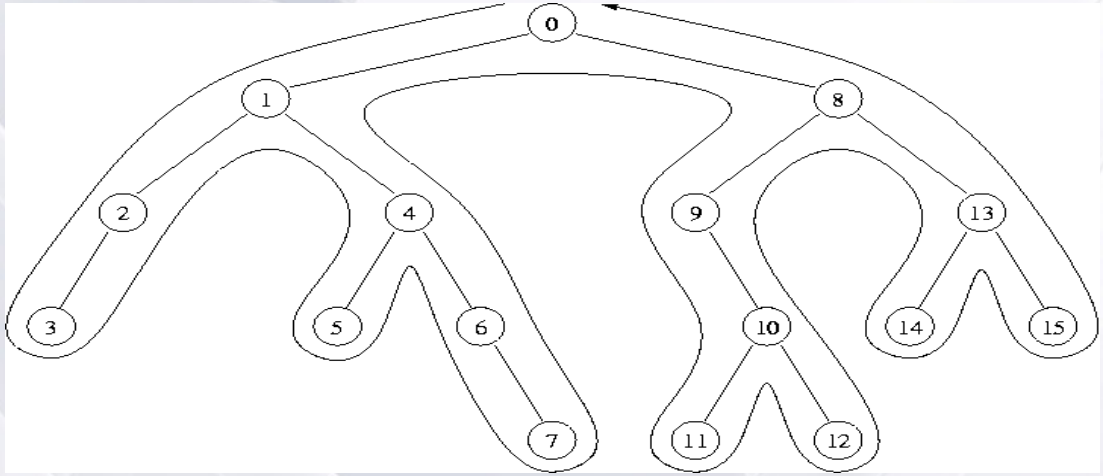
Exercice

Soit *m* un mot composé des lettres 'g' , 'd' ou 'u' (pour up)
Ecrivez une méthode d'arbre, qui affiche le contenu porté par le nœud obtenu en suivant le chemin indiqué par *m* et en partant de la racine.

Fichier Noeud.java

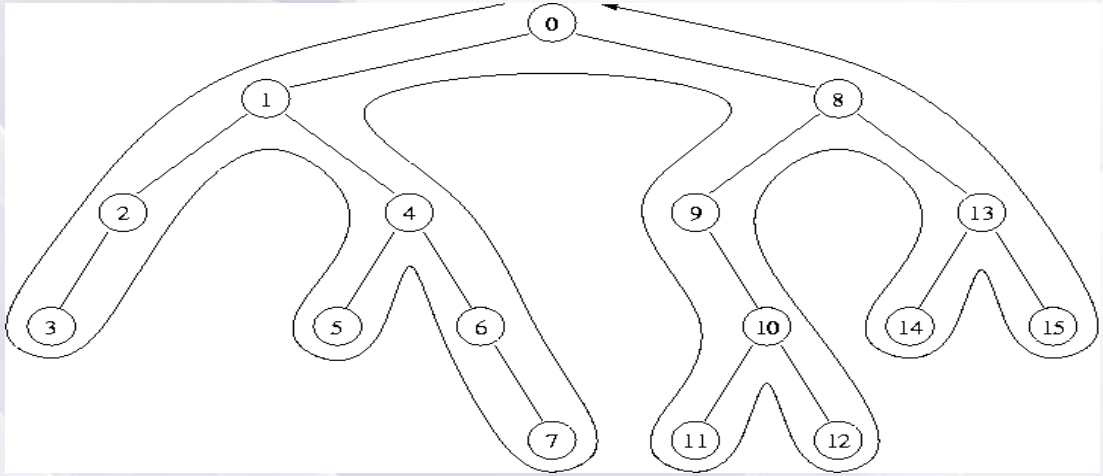
```
public void afficheOriente (String dir){
 if (dir.length()==0) {
  System.out.println(content.toString());
  return;
 }
 char d=dir.charAt(0);
 dir=dir.substring(1);
 if (d=='g' && filsG != null) filsG.afficheOrienté(dir);
 else if (d=='d' && filsD != null) filsD.afficheOrienté(dir);
 else if (d=='u' && père != null) père.afficheOrienté(dir);
 // sinon rien d'autre
}
```

Arbres étendus - Application 2 (un peu abstraite)



- Rappel : lors d'un parcours en profondeur un nœud est visité 3 fois
  - à la descente
  - à la remontée du fils gauche
  - à la remontée du fils droit
- Définissons un objet **ParcoursArbre** qui :
  - maintient le prochain nœud à visiter dans un parcours en profondeur
  - possède les méthodes :
    - **int numPassage()** qui donne le numéro de la visite de ce noeud
    - **Noeud whoIsNext()** retourne le prochain noeud à visiter
    - **void moveNext()** avance dans le parcours en cours (rien si terminé)

Arbres étendus - Application 2 (un peu abstraite)



- **int numPassage()** qui donne le numéro de la visite de ce noeud
- **Noeud whoIsNext()** retourne le prochain noeud à visiter
- **void moveNext()** avance dans le parcours en cours (rien si terminé)

- Exemple pour bien comprendre :

```
ParcoursArbre p=new ParcoursArbre(ArbreDeBranche4);
System.out.println(p.whoIsNext().getVal()); // Noeud 4
System.out.println(p.numPassage); // 1er passage
p.moveNext(); // exploration de son fils gauche
p.moveNext(); // exploration de son fils gauche
System.out.println(p.whoIsNext().getVal()); // Noeud 5
System.out.println(p.numPassage); // 2nd passage
```

Université de Paris

Implémentation de ParcoursArbre

Fichier ParcoursArbre.java

```
public class ParcoursArbre{
 private Noeud prochain;
 private int numVisite;
 public ParcoursArbre(Arbre a){
  prochain = a.getRacine();
  numVisite=1;
 }
 public Noeud whoIsNext(){ return prochain; }
 public int numPassage(){ return numVisite; }
 public void moveNext(){
    ...
 }
}
```

Université de Paris

Implémentation de ParcoursArbre

Fichier ParcoursArbre.java

```
public void moveNext(){
 if (prochain==null) return;
 switch(numVisite){
   case 1 :
     if (prochain.getFilsG()!=null) prochain=prochain.getFilsG();
     else numVisite++;
     break;
   case 2 :
     if (prochain.getFilsD()!=null) {
       prochain=prochain.getFilsD(); numVisite=1;
     } else numVisite++;
     break;
   default : // reste 3, cas où le lien au père est utile
     if ( (prochain.père != null) && (prochain.père.filsG==prochain) )
         numVisite=2;
     else numVisite=3; // car il est fils droit
     prochain=prochain.père;
     break;
 }
}
```

Arbres étendus - Application 2

Exercice (suite)

Ajoutez à **ParcoursArbre** une méthode **void moveNextPrefixe()** qui poursuit le parcours commencé, et se positionne sur le prochain noeud qui sera vu en préfixe

Arbres étendus - Application 2

Exercice (suite)

Ajoutez à **ParcoursArbre** une méthode **void moveNextPrefixe()** qui poursuit le parcours commencé, et se positionne sur le prochain noeud qui sera vu en préfixe

Fichier ParcoursArbre.java

```
public void moveNextPrefixe(){
 do {
  moveNext();
 } while(prochain != null && numVisite != 1);
}
```

Université de Paris

Arbres étendus - Application 2

Exercice (suite)

Ajoutez à **ParcoursArbre** une méthode **void moveNextPrefixe()** qui poursuit le parcours commencé, et se positionne sur le prochain noeud qui sera vu en préfixe

Fichier ParcoursArbre.java

```
public void moveNextPrefixe(){
 do {
  moveNext();
 } while(prochain != null && numVisite != 1);
}
```

- On peut ainsi imaginer **moveNextInfixe()** ou **moveNextSuffixe()**

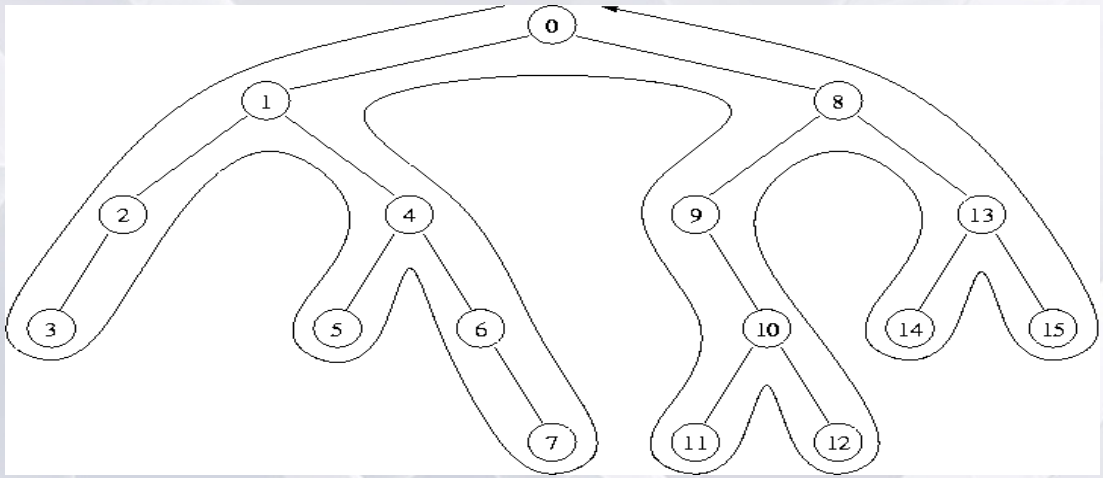
Université de Paris

Exercice 3

- **ParcoursArbre** permet d'abstraire un parcours dans un algorithme.

Exercice

Ecrivez une méthode **paireSommeEgale(int n)** qui affiche tous les couples de valeurs d'un arbre binaire d'entiers, dont la somme vaut *n*



exemple : **paireSommeEgale(5)** affiche ici dans l'ordre de votre choix :

$$(0,5)\ (5,0)\ (1,4)\ (4,1)\ (2,3)\ (3,2)$$

Université de Paris

Exercice 3

Exercice

Ecrivez une méthode **paireSommeEgale(int n)** qui affiche tous les couples de valeurs d'un arbre binaire d'entiers, dont la somme vaut *n*

Fichier Arbre.java

```
public void paireSommeEgale(int n){
 ParcoursArbre p1=new ParcoursArbre(this); Noeud n1=p1.whoIsNext();
 while(n1 != null) {
  ...
  p1.moveNextPrefixe();
  n1=p1.whoIsNext();
 }
}
```

Université de Paris

Exercice 3

Exercice

Ecrivez une méthode **paireSommeEgale(int n)** qui affiche tous les couples de valeurs d'un arbre binaire d'entiers, dont la somme vaut *n*

Fichier Arbre.java

```
public void paireSommeEgale(int n){
 ParcoursArbre p1=new ParcoursArbre(this); Noeud n1=p1.whoIsNext();
 while(n1 != null) {
  ParcoursArbre p2=new ParcoursArbre(this); Noeud n2=p2.whoIsNext();
  while(n2 != null) {
   ...
   p2.moveNextPrefixe();
   n2=p2.whoIsNext();
  }
  p1.moveNextPrefixe();
  n1=p1.whoIsNext();
 }
}
```

Exercice 3

Exercice

Ecrivez une méthode **paireSommeEgale(int n)** qui affiche tous les couples de valeurs d'un arbre binaire d'entiers, dont la somme vaut *n*

Fichier Arbre.java

```
public void paireSommeEgale(int n){
 ParcoursArbre p1=new ParcoursArbre(this); Noeud n1=p1.whoIsNext();
 while(n1 != null) {
  ParcoursArbre p2=new ParcoursArbre(this); Noeud n2=p2.whoIsNext();
  while(n2 != null) {
   int v2=n2.getVal();
   int v1=n1.getVal();
   if (v1+v2==n) System.out.println( v1 + "," + v2 );
   p2.moveNextPrefixe();
   n2=p2.whoIsNext();
  }
  p1.moveNextPrefixe();
  n1=p1.whoIsNext();
 }
}
```

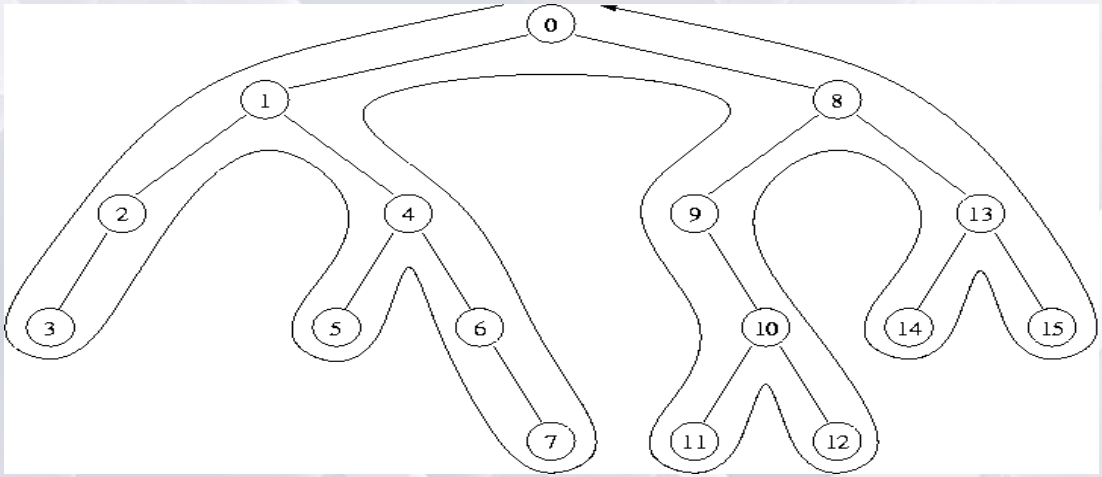
- Il est possible que vous ne soyez pas à l'aise avec **ParcoursArbre**
- Il s'agit d'un exercice, ce n'est pas qq chose de standard

Université de Paris

Exercice 4 - Version 2

Exercice

Ecrivez une méthode **paireSommeEgale(int n)** qui affiche tous les couples de valeurs d'un arbre binaire d'entiers, dont la somme vaut *n*. Sans utiliser **ParcoursArbre**



- On ne peut pas échapper à un double parcours

Université de Paris

Exercice 4 - Version 2

Exercice

Ecrivez une méthode **paireSommeEgale(int n)** qui affiche tous les couples de valeurs d'un arbre binaire d'entiers, dont la somme vaut *n*.
Sans utiliser **ParcoursArbre**

Fichier Arbre.java

```
public void paireSommeEgale(int n){
 if (racine==null) return;
 racine.paireSommeEgale(n,racine);
}
```

Fichier Noeud.java

```
public void paireSommeEgale(int n,Noeud x){ // x : arbre d'origine
 if (x==null) return;
 if (x.contient(n-content)) System.out.println(content, n-content);
 if (filsG != null) filsG.paireSommeEgale(n,x); // x : tout l'arbre
 if (filsD != null) filsD.paireSommeEgale(n,x);
}
```

Exercice 4 - Version 2

Exercice

Ecrivez une méthode **paireSommeEgale(int n)** qui affiche tous les couples de valeurs d'un arbre binaire d'entiers, dont la somme vaut *n*.
Sans utiliser **ParcoursArbre**

Fichier Arbre.java

```
public void paireSommeEgale(int n){
 if (racine==null) return;
 racine.paireSommeEgale(n,racine);
}
```

Fichier Noeud.java

```
public void paireSommeEgale(int n,Noeud x){ // x : arbre d'origine
 if (x==null) return;
 if (x.contient(n-content)) System.out.println(content, n-content);
 if (filsG != null) filsG.paireSommeEgale(n,x); // x : tout l'arbre
 if (filsD != null) filsD.paireSommeEgale(n,x);
}
```

- Pas mal, mais cette solution ne marche pas dans le cas de doublons (s'il y a plusieurs noeuds qui portent $n - content$)

Exercice 4 - Version 3

Exercice

Ecrivez une méthode **paireSommeEgale(int n)** qui affiche tous les couples de valeurs d'un arbre binaire d'entiers, dont la somme vaut *n*.
Sans utiliser **ParcoursArbre**

Fichier Arbre.java

```
public void paireSommeEgale(int n){
 if (racine==null) return;
 racine.paireSommeEgale(n,racine);
}
```

Fichier Noeud.java

```
public void paireSommeEgale(int n,Noeud x){
 if (x==null) return;
 x.trouveCorrespondance(content,n); // content transmis pour affichage
 if (filsG != null) filsG.paireSommeEgale(n,x);
 if (filsD != null) filsD.paireSommeEgale(n,x);
}
```

Exercice 4 - Version 3

Fichier Arbre.java

```
public void paireSommeEgale(int n){
 if (racine==null) return;
 racine.paireSommeEgale(n,racine);
}
```

Fichier Noeud.java

```
public void paireSommeEgale(int n,Noeud x){
 if (x==null) return;
 x.trouveCorrespondance(content,n); // content transmis pour affichage
 if (filsG != null) filsG.paireSommeEgale(n,x);
 if (filsD != null) filsD.paireSommeEgale(n,x);
}
public void trouveCorrespondance(int v1,int n){
 if (v1+content==n) System.out.println(v1 + " , " + content);
 if (filsG != null) filsG.trouveCorrespondance(v1,n);
 if (filsD != null) filsD.trouveCorrespondance(v1,n);
}
```

Université de Paris

Exercices 5 et 6 - facultatif

Exercice

Ecrivez une méthode **décomposeTripletSomme(int n)** qui affiche tous les choix possibles de 3 valeurs contenues dans un arbre binaire, dont la somme est n

Exercice

Ecrivez une méthode **décomposeSomme(int n, int p)** qui affiche tous les choix possibles de p valeurs contenues dans un arbre binaire, dont la somme est n

Université de Paris

Exercices 5 et 6 - facultatif

Exercice

Ecrivez une méthode **décomposeTripletSomme(int n)** qui affiche tous les choix possibles de 3 valeurs contenues dans un arbre binaire, dont la somme est n

Exercice

Ecrivez une méthode **décomposeSomme(int n, int p)** qui affiche tous les choix possibles de p valeurs contenues dans un arbre binaire, dont la somme est n

... Probablement qu'ici l'abstraction de parcoursArbre sera appréciée ...

Université de Paris

Résumé provisoire

Nous avons :

- Illustré à nouveau des parcours simples
  - pour les arbres complets

Université de Paris

Résumé provisoire

Nous avons :

- Illustré à nouveau des parcours simples
  - pour les arbres complets
- Illustré à nouveau un parcours avec passage d'argument
  - la moyenne des profondeurs des feuilles

Université de Paris

Résumé provisoire

Nous avons :

- Illustré à nouveau des parcours simples
  - pour les arbres complets
- Illustré à nouveau un parcours avec passage d'argument
  - la moyenne des profondeurs des feuilles
- Envisagé l'utilisation d'un lien vers le père
  - cas simple de mots construits sur 'g', 'd', et 'u' pour up

Université de Paris

Résumé provisoire

Nous avons :

- Illustré à nouveau des parcours simples
  - pour les arbres complets
- Illustré à nouveau un parcours avec passage d'argument
  - la moyenne des profondeurs des feuilles
- Envisagé l'utilisation d'un lien vers le père
  - cas simple de mots construits sur 'g', 'd', et 'u' pour up
- Traité une application non triviale de ce lien
  - construction d'un objet un peu théorique de parcours

Université de Paris

Résumé provisoire

Nous avons :

- Illustré à nouveau des parcours simples
  - pour les arbres complets
- Illustré à nouveau un parcours avec passage d'argument
  - la moyenne des profondeurs des feuilles
- Envisagé l'utilisation d'un lien vers le père
  - cas simple de mots construits sur 'g', 'd', et 'u' pour up
- Traité une application non triviale de ce lien
  - construction d'un objet un peu théorique de parcours
- Traité à fond un exercice nécessitant deux parcours du même arbre

Université de Paris

Exercice 7

Exercice

On défini un arbre dont les noeuds portent chacuns deux valeurs entières a et b, avec $a < b$. Ils représentent donc des intervalles.
On souhaite que dans l'arbre chaque noeud soit une sur-approximation des noeuds inférieurs, au sens où son intervalle englobe (contienne) les autres.
Ecrire une méthode qui teste si l'arbre est correctement étiqueté pour réaliser ces sur-approximations.

Université de Paris

Exercice 7

Exercice

On défini un arbre dont les noeuds portent chacuns deux valeurs entières a et b, avec $a < b$. Ils représentent donc des intervalles.
On souhaite que dans l'arbre chaque noeud soit une sur-approximation des noeuds inférieurs, au sens où son intervalle englobe (contienne) les autres.
Ecrire une méthode qui teste si l'arbre est correctement étiqueté pour réaliser ces sur-approximations.

Fichier Noeud.java

```
public class Noeud{
 private int a;
 private int b;
 private Noeud filsG;
 private Noeud filsD;
}
```

Université de Paris

Exercice 7

Exercice

On défini un arbre dont les noeuds portent chacuns deux valeurs entières a et b, avec $a < b$. Ils représentent donc des intervalles.
On souhaite que dans l'arbre chaque noeud soit une sur-approximation des noeuds inférieurs, au sens où son intervalle englobe (contienne) les autres.
Ecrire une méthode qui teste si l'arbre est correctement étiqueté pour réaliser ces sur-approximations.

Fichier Noeud.java

```
public class Noeud{
 private int a;
 private int b;
 private Noeud filsG;
 private Noeud filsD;
}
```

- La seule difficulté est de comprendre que
  - Le *min* des a des sous arbres doit être dans $[a, b]$
  - Le *max* des b des sous arbres doit être dans $[a, b]$
- Des parcours simples suffisent

Université de Paris

Exercice 7

Fichier Noeud.java

```
public int minA(){
 int rep=this.a;
 if (filsG != null) rep=Math.min(rep,filsG.minA());
 if (filsD != null) rep=Math.min(rep,filsD.minA());
 return rep;
}
```

Exercice 7

Fichier Noeud.java

```
public int minA(){
 int rep=this.a;
 if (filsG != null) rep=Math.min(rep,filsG.minA());
 if (filsD != null) rep=Math.min(rep,filsD.minA());
 return rep;
}
public int maxB(){
 int rep=this.b;
 if (filsG != null) rep=Math.max(rep,filsG.maxB());
 if (filsD != null) rep=Math.max(rep,filsD.maxB());
 return rep;
}
```

Université de Paris

Fichier Noeud.java

```
public int minA(){
 int rep=this.a;
 if (filsG != null) rep=Math.min(rep,filsG.minA());
 if (filsD != null) rep=Math.min(rep,filsD.minA());
 return rep;
}
public int maxB(){
 int rep=this.b;
 if (filsG != null) rep=Math.max(rep,filsG.maxB());
 if (filsD != null) rep=Math.max(rep,filsD.maxB());
 return rep;
}
public boolean surApproxime(){
 if (filsG != null)
   && (filsG.minA()<=a || filsG.maxB()>=b || !filsG.surApproxime())
   return false;
 if (filsD != null)
   && (filsD.minA()<=a || filsD.maxB()>=b || !filsD.surApproxime())
   return false;
 return true;
}
```

Question cette variante est-elle correcte ? (à voir chez vous)

```
public boolean surApproxime(){
 if (filsG != null)
  && (filsG.a<=a || filsG.b>=b || !filsG.surApproxime())
   return false;
 if (filsD != null)
  && (filsD.a<=a || filsD.b>=b || !filsD.surApproxime())
   return false;
 return true;
}
```

Université de Paris

Exercice 8

Exercice

On défini un arbre dont les noeuds portent chacuns deux valeurs entières a et b, avec $a < b$. Ils représentent donc des intervalles.
On souhaite que dans l'arbre chaque noeud soit une sur-approximation des noeuds inférieurs, au sens où son intervalle englobe (contienne) les autres.
On veux forcer cette sur-approximation : l'arbre étant ce qu'il est, faites en sorte que chaque noeud impose à ses fils de se mettre en conformité s'il ne l'est pas. C'est à dire de modifier au mieux ses valeurs pour rentrer dans le cadre de ces approximations.

Université de Paris

Exercice 8

Exercice

On défini un arbre dont les noeuds portent chacuns deux valeurs entières a et b, avec $a < b$. Ils représentent donc des intervalles.
On souhaite que dans l'arbre chaque noeud soit une sur-approximation des noeuds inférieurs, au sens où son intervalle englobe (contienne) les autres.
On veux forcer cette sur-approximation : l'arbre étant ce qu'il est, faites en sorte que chaque noeud impose à ses fils de se mettre en conformité s'il ne l'est pas. C'est à dire de modifier au mieux ses valeurs pour rentrer dans le cadre de ces approximations.

- La transmission d'information se fait du haut vers le bas
- $\Rightarrow$ on ajoute des paramètres au parcours

Université de Paris

Exercice 8

Fichier Arbre.java

```
public void comply(){
    if (racine !=null) racine.comply();
}
```

Exercice 8

Fichier Arbre.java

```
public void comply(){
    if (racine !=null) racine.comply();
}
```

Fichier Noeud.java

```
public void comply(){
    comply(a,b);
}
```

Université de Paris

Exercice 8

Fichier Arbre.java

```
public void comply(){
    if (racine !=null) racine.comply();
}
```

Fichier Noeud.java

```
public void comply(){
    comply(a,b);
}
public void comply(int x,int y){
    if (a < x) a=x;
    if (b > y) b=y;
    if (filsG!=null) filsG.comply(a,b);
    if (filsD!=null) filsD.comply(a,b);
}
```

Université de Paris

Exercice 9

Exercice

On considère deux arbres binaires a et b portant des valeurs entières.
Les noeuds de ces arbres disposent d'un lien vers leurs pères.
On souhaite "greffer" l'arbre b à la place du premier noeud de a qui dans un parcours infixe porte la même valeur que la racine de b.
Ecrivez cette méthode

Université de Paris

Exercice 9

Exercice

On considère deux arbres binaires a et b portant des valeurs entières.
Les noeuds de ces arbres disposent d'un lien vers leurs pères.
On souhaite "greffer" l'arbre b à la place du premier noeud de a qui dans un parcours infixe porte la même valeur que la racine de b.
Ecrivez cette méthode

- Attention : la racine peut changer
- Attention : une seule greffe (au premier noeud de valeur égale)

Université de Paris

Exercice 9 - solution 1 avec ArbreParcours

Fichier Arbre.java

```
public void greffe(Arbre b){
 ParcoursArbre p=new ParcoursArbre(this);
 p.nextInfixe(); // si vous l'avez écrit ...
 Noeud n=p.whoIsNext();
 while(n!=null) { // on regarde partout
  if(...){ // on l'a trouvé !?
  ...
  }
  p.nextInfixe(); // on continue à chercher
  n=p.whoIsNext();
 }
}
```

Exercice 9 - solution 1 avec ArbreParcours

Fichier Arbre.java

```
public void greffe(Arbre b){
 ParcoursArbre p=new ParcoursArbre(this);
 p.nextInfixe(); // si vous l'avez écrit ...
 Noeud n=p.whoIsNext();
 while(n!=null) { // on le cherche
  if(n.getVal()==b.racine.getVal()){ //on l'a trouvé !
   if (n==racine) {racine=b.racine; b.racine=null; return;}
   ...
  }
  p.nextInfixe();
  n=p.whoIsNext();
 }
}
```

Exercice 9 - solution 1 avec ArbreParcours

Fichier Arbre.java

```
public void greffe(Arbre b){
 ParcoursArbre p=new ParcoursArbre(this);
 p.nextInfixe(); // si vous l'avez écrit ...
 Noeud n=p.whoIsNext();
 while(n!=null) { // on le cherche
  if(n.getVal()==b.racine.getVal()){ //on l'a trouvé !
   if (n==racine) {racine=b.racine; b.racine=null; return;}
   Noeud p=n.getPère();
   if (p.getFilsG()==n) p.setFilsG(b.racine);
   if (p.getfilsD()==n) p.setFilsD(b.racine);
   b.racine.setPère(p);
   b.racine=null;
   return; // on s'arrête
  }
  p.nextInfixe();
  n=p.whoIsNext();
 }
}
```

Exercice 9 - solution 2 en modifiant le parcours infixe

Exercice

"greffer" l'arbre b à la place du premier noeud de a qui dans un parcours infixe porte la même valeur que la racine de b.

- Attention : la racine peut changer
- Attention : une seule greffe (au premier noeud de valeur égale)

Université de Paris

Exercice 9 - solution 2 en modifiant le parcours infixe

Exercice

"greffer" l'arbre *b* à la place du premier noeud de *a* qui dans un parcours infixe porte la même valeur que la racine de *b*.

- Attention : la racine peut changer
- Attention : une seule greffe (au premier noeud de valeur égale)

Fichier Noeud.java - rappel du parcours Infixe

```
public void afficheInfixe(){
 if (filsG!=null) filsG.afficheInfixe();
 System.out.print(val) ;
 if (filsD!=null) filsD.afficheInfixe();
}
```

- Le traitement ne doit pas avoir lieu s'il a déjà eu lieu sur le fils gauche
- Il faut une valeur retour pour le savoir
- L'arbre *a* aussi aimerait savoir si la racine doit être modifiée

Université de Paris

Exercice 9 - solution 2 en modifiant le parcours infixe

Exercice

"greffer" l'arbre *b* à la place du premier noeud de *a* qui dans un parcours infixe porte la même valeur que la racine de *b*.

- Attention : la racine peut changer
- Attention : une seule greffe (au premier noeud de valeur égale)

Fichier Noeud.java - rappel du parcours Infixe

```
public void afficheInfixe(){
 if (filsG!=null) filsG.afficheInfixe();
 System.out.print(val) ;
 if (filsD!=null) filsD.afficheInfixe();
}
```

- Le traitement ne doit pas avoir lieu s'il a déjà eu lieu sur le fils gauche
- Il faut une valeur retour pour le savoir
- L'arbre *a* aussi aimerait savoir si la racine doit être modifiée
- $\Rightarrow$ retournons le noeud résultat s'il y a eu modif, null sinon

Université de Paris

Exercice 9 - solution 2 en modifiant le parcours infixe

Fichier Arbre.java

```
public void greffe(Arbre b){
 if ( racine==null || b.racine==null) return;
 Noeud res=racine.greffe(b.racine);
 // concrètement le resultat sera soit null, soit racine, soit b
 if (res!=null) { racine=res; b.racine=null; }
}
```

Université de Paris

Fichier Arbre.java

```
public void greffe(Arbre b){
 if ( racine==null || b.racine==null) return;
 Noeud res=racine.greffe(b.racine);
 // concrètement le resultat sera soit null, soit racine, soit b
 if (res!=null) { racine=res; b.racine=null; }
```

Fichier Noeud.java

```
public Noeud greffe(Noeud b){
 Noeud res=null;
 if (filsG!=null) res=filG.greffe(b);
 if (res != null) return this; // il a été trouvé à gauche
 if (val==b.val) { // greffe à faire
  if (père==null) return b; // cas de la racine
  Noeud p=père;
  if (p.filsG==this) {p.filsG=b;}
  if (p.filsD==this) {p.filsD=b;}
  b.père=p;
  return b;
 }
 if (filsD!=null) res=filsD.greffe(b);
 if (res==null) return null;
 return this;
}
```