

Retour sur equals 1/2

```
class Paire {
String w1, w2 ;
Paire (String w1, String w2) {
    this.w1 = w1 ; this.w2 = w2 ;
}
}
```

```
...
Paire p= new Paire ("un","deux");
Paire q= new Paire ("un","deux");
String s="un", t="un";
System.out.println(p.equals(q)+
    ", "+s.equals(t));
```

Affiche: *false,true*

- Pourquoi?
 - ▶ Lors de l'appel d'une méthode, JAVA cherche une déclaration de méthode qui corresponde à cette signature.
 - ▶ Ici, il n'y a pas de méthode equals dans la classe Paire.
 - ▶ Puisque tout objet, et donc Paire, hérite de la classe Object, JAVA cherche dans la classe Object. Or dans Object, **class** on peut lire:

```
public boolean equals(Object obj) {return (this==obj);}
```

- L'implémentation par défaut est donc l'égalité physique.
- Pour les String, la méthode equals est redéfinie.

Retour sur equals 1/2

```
class Paire {
String w1, w2 ;
Paire (String w1, String w2) {
    this.w1 = w1 ; this.w2 = w2 ;
}
public boolean equals(Paire p){
    return w1.equals(p.w1)
    && w2.equals(p.w2) ;
}
}
```

```
...
Paire p= new Paire ("un","deux");
Paire q= new Paire ("un","deux");
String s="un", t="un";
System.out.println(p.equals(q)+
    ", "+s.equals(t));
```

Affiche: *true,true*

- Ce problème est donc résolu.
- Bilan:
 - ▶ Utiliser equals est toujours mieux que d'utiliser ==.
 - ▶ Mais ce n'est pas toujours suffisant.
- Mais alors, pourquoi la semaine dernière n'avait-on pas fait comme cela?

Retour sur le code de la librairie standard

```
public V get(Object key) {
    if (key == null)
        return getForNullKey();
    int hash = hash(key.hashCode());
    for (Entry<K,V> e = table[indexFor(hash, table.length)];
        e != null;
        e = e.next) {
        Object k;
        if (e.hash == hash && ((k = e.key) == key
            || key.equals(k)))
            return e.value;}
    return null;}
}
```

- Il est écrit `key.equals(k)`, où `k` et `key` sont des Objects.
- Même en ayant défini **public boolean equals(Paire p)** dans la classe Paire, JAVA ira chercher la déclaration **public boolean equals(Object obj)** dans la classe Object, car `key` n'est pas une Paire, mais un Object.
- L'égalité utilisée est donc toujours l'égalité physique.

Que retenir de tout cela?

- Utiliser equals est toujours mieux que d'utiliser ==.
- Pour utiliser un `HashMap<K,V>`, il faut s'assurer que
 - ▶ la méthode equals est correctement définie pour les objets de la classe `K`.
 - la redéfinir si besoin avec un code du type:

```
public boolean equals(Object o) {
    Paire p = (Paire)o ;
    return w1.equals(p.w1) && w2.equals(p.w2) ; }
```
 - ▶ la méthode hashCode est correctement définie pour les objets de la classe `K`.
 - la redéfinir si besoin, en définissant:

```
public int hashCode() {...}
```
 - ▶ Un contrat avec JAVA:
 - lorsque `k.equals(k')` est vrai, on doit avoir `k.hashCode() == k'.hashCode()` pour `k` et `k'` de type `K`.
- Quelque part, la "vraie" sémantique de get implémentée est:
 - ▶ La méthode `V get(K k)` retourne la donnée associée à une clé `k'`, si il existe une clé `k'` avec `k'.equals(k)`, **null** sinon.

Définir un type abstrait proprement

- On veut définir le type abstrait "sac"

```
class Sac {
    ...
    Sac() { ... } //Construire un sac vide
    boolean EstVide() { ... } //Tester si un sac est vide
    void Ajouter(int e) { ... } //Ajouter un entier à un sac
    int Enlever() { ... } //Enlever un entier d'un sac
}
```

- Pour le faire proprement en JAVA, on peut définir

```
interface Sac {
    boolean EstVide(); //Tester si un sac est vide
    void Ajouter(int e); //Ajouter un entier à un sac
    int Enlever(); //Enlever un entier d'un sac
}
```

5

Ce qui permet de l'utiliser

- A ce moment là, sac n'est pas une classe, mais on peut l'utiliser comme un type...

```
static void count(Sac s) {
    for (int i=0; !s.EstVide(); i++) {
        s.Enlever();
    }
}
```

- ... sans se préoccuper de l'implémentation.

6

Créer une implémentation

- On utilise le mot clé **implements**.

Implémentation 1:

```
class File implements Sac {
    private int[] t;
    private int sp;
    File() { t=new int[100]; sp=0; }
    public boolean EstVide() {
        return (sp <= 0); }
    int
    Tete() {return t[sp-1];}
    public void Ajouter(int e) {
        t[sp++] = e;}
    public int Enlever() {
        return t[--sp];} } }
```

Implémentation 2:

```
class File implements Sac {
    private int[] t;
    private int in,out;
    File() { t=new int[100];
        in=out=0;}
    public boolean EstVide() {
        return (in == out); }
    public void Ajouter(int e) {
        t[out]=e; out=(out +1)% 100;}
    public int Enlever() {
        int c=t[in]; in=(in+1) % 100;
        return c;} } }
```

7

Intérêts

- On peut travailler sans se préoccuper de l'implémentation.
 - ce qui permet le travail en groupe, ou modulaire.
 - et de remplacer une implémentation par une autre sans modifier le reste du programme.
- Le compilateur refuse de compiler si une des méthodes de l'**interface** est manquante,
 - et donc aide à vérifier que l'implémentation de chacune des fonctionnalités est bien présente.
- Note: Les méthodes déclarées dans l'interface doivent être qualifiées de **public**.

8

Cours 5: Arbres

Olivier Bournez

bournez@lix.polytechnique.fr
LIX, Ecole Polytechnique

INF421-b

Bases de la programmation et de l'algorithmique

9

Aujourd'hui

Les graphes

Les arbres

Les arbres binaires

Programmer des arbres binaires

Parcours d'arbres

Les arbres généraux

Le codage de Huffman

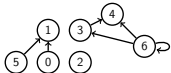
10

Un graphe orienté

- Un *graphe orienté* (digraph) est donné par un couple $G = (V, E)$, où
 - ▶ V est un ensemble.
 - ▶ $E \subset V \times V$.

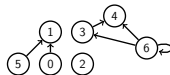
- Exemple:

- ▶ $V = \{0, 1, \dots, 6\}$.
- ▶ $E = \{(0, 1), (3, 4), (5, 1), (6, 3), (6, 4), (6, 6)\}$.



11

Vocabulaire

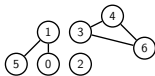


- Les éléments de V sont appelés des *sommets* (parfois aussi des *nœuds*).
- Les éléments e de E sont appelés des *arcs*.
- Si $e = (u, v)$, u est appelé *la source* de e , v est appelé *la destination* de e .
- Remarque:
 - ▶ Les boucles (les arcs (u, u)) sont autorisées.

12

Un graphe

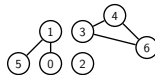
- Un *graphe*¹ est donné par un couple $G = (V, E)$, où
 - ▶ V est un ensemble.
 - ▶ E est un ensemble de paires $\{u, v\}$ avec $u, v \in V$.
- On convient de représenter une paire $\{u, v\}$ par (u, v) ou (v, u) .
- Autrement dit, (u, v) et (v, u) dénotent la même arête.
- Exemple:
 - ▶ $V = \{0, 1, \dots, 6\}$
 - ▶ $E = \{(0, 1), (3, 4), (5, 1), (6, 3), (6, 4)\}$.



¹Lorsqu'on ne précise pas, par défaut, un graphe est non-orienté.

13

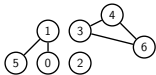
Vocabulaire



- Les éléments e de E sont appelés des *arêtes*. Si $e = (u, v)$, u et v sont appelés les *extrémités* de e .
- Remarque: (sauf autre convention explicite)
 - ▶ Les boucles ne sont pas autorisées.

14

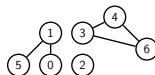
Vocabulaire (cas non-orienté)



- u et v sont dits *voisins* s'il y a une arête entre u et v .
- Le *degré* de u est le nombre de voisins de u .

15

Vocabulaire (cas non-orienté)

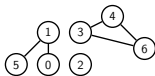


- Un *chemin du sommet s vers le sommet t* est une suite e_0, e_1, \dots, e_n de sommets telle que $e_0 = s$, $e_n = t$, $(e_{i-1}, e_i) \in E$, pour tout $1 \leq i \leq n$.
 - ▶ n est appelé la *longueur* du chemin, et on dit que t est *joignable* à partir de s .
 - ▶ Le chemin est dit *simple* si les e_i sont distincts deux-à-deux.
 - ▶ Un *cycle* est un chemin de longueur non-nulle avec $e_0 = e_n$.
- Le sommet s est dit à *distance n* de t s'il existe un chemin de longueur n entre s et t , mais aucun chemin de longueur inférieure.

16

Composantes connexes (cas non-orienté)

- Prop. La relation "être joignable" est une relation d'équivalence.
- Les classes d'équivalence sont appelées les *composantes connexes*.



- Un graphe est dit *connexe* s'il n'y a qu'une seule classe d'équivalence.
 - ▶ Autrement dit, tout sommet est joignable à partir de tout sommet.

17

Les graphes sont partout!

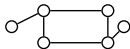
- Beaucoup de problèmes se modélisent par des objets et des relations entre objets.
- Exemples:
 - ▶ Le graphe routier.
 - ▶ Les réseaux informatiques.
 - ▶ Le graphe du web.
- Beaucoup de problèmes se ramènent à des problèmes sur les graphes.
- Théorie des graphes:
 - ▶ Euler, Hamilton, Kirchhoff, König, Edmonds, Berge, Lovász, Seymour,...
- Les graphes sont omniprésents en informatique.

18

Théorie des graphes. Exemple 1: graphes planaires.

- Un graphe est dit *planaire* s'il peut se représenter sur un plan sans qu'aucune arête n'en croise une autre.

- Planaire:





- Non-planaire: K_5



$K_{3,3}$



- Théorème [Kuratowski-Wagner] Un graphe fini est planaire ssi il ne contient pas de sous-graphe qui soit une expansion de K_5 ou de $K_{3,3}$.

- ▶ Une expansion consiste à ajouter un ou plusieurs sommets sur une ou plusieurs arêtes (exemple:  devient 

19

Théorie des graphes. Exemple 2: Coloriage de graphe.

- Allouer des fréquences GSM correspond à colorier les sommets d'un graphe.
 - ▶ sommets: des émetteurs radio.
 - ▶ arête entre u et v : le signal de u perturbe v ou réciproquement.
 - ▶ couleur: fréquence radio.
- Le problème de *coloriage d'un graphe*: colorier les sommets d'un graphe de telle sorte qu'il n'y ait aucune arête entre deux sommets d'une même couleur.



Un coloriage avec 4 couleurs

20

Théorie des graphes. Exemple 2: Coloriage de graphe.

- Théorème: Appel et Haken (76): tout graphe planaire est coloriable avec 4 couleurs (preuve avec 1478 cas critiques).
 - ▶ Robertson, Sanders, Seymour, Thomas, Gonthier, Werner.
- On ne connaît aucun algorithme pour déterminer si un graphe général est coloriable avec 4 couleurs (même avec 3) qui fonctionne en temps polynomial (le problème est *NP-complet*).

21

Applications des graphes. Exemple 3: évaluer une page de la toile

- Brevet "Method for Node Ranking in a Linked Database" (Université Stanford, Janvier 1997).
- En simplifiant.
 - ▶ On considère le graphe des liens entre pages.
 - ▶ Modèle théorique:
 - Avec probabilité $0 < c < 1$, un surfeur abandonne la page actuelle et recommence sur une des n pages du web, choisie de manière équiprobable.
 - Avec probabilité $1 - c$, le surfeur suit un des liens de la page actuelle j , choisie de manière équiprobable parmi tous les liens l_j émis.
 - ▶ Quelle que soit la distribution initiale, on convergera vers une unique distribution stationnaire.
 - ▶ On évalue une page par la probabilité de cette page dans la distribution stationnaire.

22

Aujourd'hui

Les graphes

Les arbres

Les arbres binaires

Programmer des arbres binaires

Parcours d'arbres

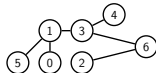
Les arbres généraux

Le codage de Huffman

23

Les arbres sont partout!

- Un graphe² connexe sans cycle est appelé un *arbre* (libre).
- Un graphe² sans-cycle est appelé une *forêt*:
 - ▶ chacune de ses composantes connexes est un arbre.
- Dès qu'on a des objets, des relations entre objets, et pas de cycle, on a donc un arbre ou une forêt.



- Les arbres sont omniprésents en informatique.

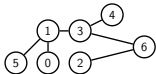
²non-orienté.

24

Une caractérisation & Quelques propriétés

Soit $G = (V, E)$ un graphe³. Les propriétés suivantes sont équivalentes:

- G est un arbre (libre).
- Deux sommets quelconques de V sont connectés par un unique chemin simple.
- G est connexe, mais ne l'est plus si on enlève n'importe laquelle de ses arêtes.
- G est connexe, et $|E| = |V| - 1$.
- G est sans cycle, et $|E| = |V| - 1$.
- G est sans cycle, mais ne l'est plus si l'on ajoute n'importe quelle arête.

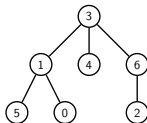


³non-orienté.

26

Les arbres poussent de haut en bas en informatique

- On distingue souvent un sommet que l'on appelle *sa racine*.
- On dessine un arbre
 - ▶ en plaçant la racine tout en haut.
 - ▶ puis en plaçant les sommets à distance i de la racine à la ligne i .
- Exemple: pour l'arbre libre précédent, en prenant le sommet d'étiquette 3 comme racine.



26

Encore du vocabulaire

- Pour tout sommet u , il existe un unique chemin (simple) entre la racine r et u .
- Tout sommet a sur ce chemin est appelé *un ancêtre* de u . u est dit un *descendant* de a .
- L'avant dernier sommet p sur ce chemin est appelé *le père* de u . u est appelé *un fils* de p .
- Le *sous-arbre* de racine u est l'arbre induit par les descendants de u .
- *L'arité* d'un sommet est le nombre de ses fils.
- Un sommet qui n'a pas de fils est appelé *une feuille*.
- La *hauteur d'un sommet* est sa distance à la racine r .
- La *hauteur d'un arbre* est la hauteur de la feuille la plus haute.

27

Une vision récursive: cas général

- Un arbre correspond à un couple formé
 1. d'un sommet particulier, appelé sa racine,
 2. et d'une partition des sommets restants en un ensemble d'arbres.
- Cette vision permet de faire des preuves inductives.
 - ▶ Exemple: montrer qu'un arbre dont tous les sommets sont d'arité au moins 2 possède plus de feuilles que de sommets internes.
- Mais reste informatiquement encore un peu compliqué.

28

Aujourd'hui

Les graphes

Les arbres

Les arbres binaires

Programmer des arbres binaires

Parcours d'arbres

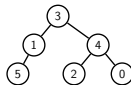
Les arbres généraux

Le codage de Huffman

29

Une vision récursive: arbres binaires

- Plus simple: Les arbres binaires.



- Un arbre binaire est
 - soit vide
 - soit l'union disjointe d'un sommet, appelé sa *racine*, d'un arbre binaire, appelé *sous-arbre gauche*, et d'un arbre binaire, appelé *sous-arbre droit*.

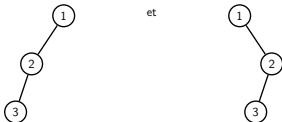
- Autrement dit, un arbre binaire d'entiers est solution de l'équation:

$$A = \text{null} \sqcup (A \times \text{int} \times A)$$

30

Attention: Arbre binaire \neq Arbre d'arité ≤ 2 .

- Un arbre *binaire* a tous ses sommets d'arité 0, 1 ou 2.
- Mais les concept d'arbre de degré ≤ 2 et d'arbre binaire sont différents.
- En fait,



ne sont pas les même arbres binaires, car

$$(((\emptyset, 3, \emptyset), 2, \emptyset), 1, \emptyset) \neq (\emptyset, 1, ((\emptyset, 3, \emptyset), 2, \emptyset))$$

31

Bornes sur la hauteur et nombre de sommets

- Dans un arbre binaire de hauteur h contenant n sommets, on a

$$\lceil \log_2 n \rceil \leq h \leq n - 1.$$

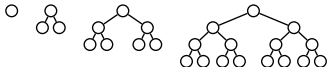
- Si on préfère:

$$h + 1 \leq n \leq 2^{h+1} - 1.$$

- Arbres avec h maximal.



- Arbres avec n maximal.



32

Aujourd'hui

Les graphes

Les arbres

Les arbres binaires

Programmer des arbres binaires

Parcours d'arbres

Les arbres généraux

Le codage de Huffman

33

Définir un arbre binaire en JAVA

- Slogan:



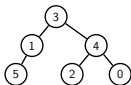
- On code un arbre vide par la référence **null**, et un arbre non-vidé par une instance de la classe Arbre.

```
class Arbre {  
    int val; Arbre gauche, droite;  
    Arbre (Arbre gauche, int val, Arbre droite) {  
        this.gauche = gauche;  
        this.val = val;  
        this.droite = droite; }  
}
```

34

Construire un arbre binaire

```
new Arbre (  
    new Arbre (  
        new Arbre (null,5,null),  
        1,  
        null),  
    3,  
    new Arbre (  
        new Arbre (null,2,null),  
        4,  
        new Arbre (null,0,null)))
```



35

Calculer la hauteur d'un arbre binaire

- La programmation d'algorithmes sur les arbres binaires est intrinsèquement récursive, car un arbre binaire est solution de l'équation

$$A = \text{null} \cup (A \times \text{int} \times A)$$

- Exemple: Calcul de la hauteur. Équations récursives, en notant \emptyset l'arbre vide, et en notant (G, r, D) un arbre binaire non-vidé.

$$\text{Hauteur}((\emptyset, r, \emptyset)) = 0$$

$$\text{Hauteur}((G, r, \emptyset)) = 1 + \text{Hauteur}(G)$$

$$\text{Hauteur}((\emptyset, r, D)) = 1 + \text{Hauteur}(D)$$

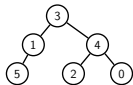
$$\text{Hauteur}((G, r, D)) = 1 + \max(\text{Hauteur}(G), \text{Hauteur}(D))$$

```
static int hauteur(Arbre a) {  
    if (a.gauche== null && a.droite==null) return 0;  
    if (a.gauche==null) return 1+hauteur(a.droite);  
    if (a.droite==null) return 1+hauteur(a.gauche);  
    return 1 + Math.max(hauteur(a.gauche), hauteur(a.droite));  
}
```

36

Se repérer dans un arbre binaire

- Tout sommet s'identifie de façon unique par le chemin de la racine vers ce sommet.
- Ce chemin est une succession d'arêtes orientées à gauche (que l'on peut coder par 0) ou à droite (que l'on peut coder par 1).
- Et donc tout sommet s'identifie de façon unique par un mot binaire (éventuellement vide).
- Exemple
 1. le sommet 5, se code par 00,
 2. le sommet 2, se code par 10,
 3. le sommet 4, se code par 1,
 4. le sommet 3, se code par ϵ (le mot vide).



37

Récupérer un sous-arbre

- Un chemin est aussi une liste d'entiers (valant 0 ou 1).
- Équations récursives, en notant \emptyset l'arbre vide et la liste vide, et en notant (G, r, D) un arbre binaire non-vide, et (x, L) une liste non-vide.

$$\begin{aligned} \text{SousArbre}(\emptyset, \emptyset) &= \emptyset \\ \text{SousArbre}((G, r, D), \emptyset) &= (G, r, D) \\ \text{SousArbre}((G, r, D), (0, L)) &= \text{SousArbre}(G, L) \\ \text{SousArbre}((G, r, D), (1, L)) &= \text{SousArbre}(D, L) \end{aligned}$$

```
static Arbre sousArbre(Arbre a, Liste c) {
    if (a==null & c==null) return null;
    if (c==null) return a;
    if (c.contenu==0) return sousArbre(a.gauche,c.suivant);
    if (c.contenu==1) return sousArbre(a.droite,c.suivant);
    throw new Error("Chemin non valide");
}
```

38

Aujourd'hui

Les graphes

Les arbres

Les arbres binaires

Programmer des arbres binaires

Parcours d'arbres

Les arbres généraux

Le codage de Huffman

39

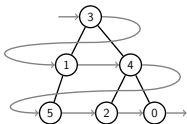
Parcourir un arbre

- Beaucoup d'algorithmes sur les arbres nécessitent de parcourir (traiter) tous les sommets
 - ▶ Exemple: Tester l'existence d'une valeur particulière dans un arbre.
 - ▶ Exemple: Afficher un arbre.
- Il existe une terminologie standard pour qualifier les parcours.
 - ▶ On peut parcourir de gauche à droite, ou de droite à gauche.
 - ▶ Une fois ce choix fait, on distingue les parcours
 - en largeur.
 - en profondeur:
 - préfixe,
 - infixe,
 - suffixe.

40

Parcourir en largeur

- Parcours en largeur d'abord.
 - ▶ on parcourt par distance croissante à la racine.



- Si le traitement d'un sommet consiste à l'afficher, on affichera dans l'ordre 3, 1, 4, 5, 2, 0.
- C'est peut-être le parcours le plus naturel, mais c'est le plus délicat à programmer avec les arbres binaires.

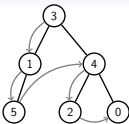
41

Parcourir en profondeur

- Parcours en profondeur d'abord:
 - ▶ on parcourt récursivement.
- Mais il reste trois possibilités
 - ▶ Préfixe: traiter la racine, parcourir le sous-arbre gauche, puis le sous-arbre droit.
 - ▶ Infixe: parcourir le sous-arbre gauche, traiter la racine, parcourir les sous-arbre droit.
 - ▶ Suffixe (appelé aussi postfixe): parcourir le sous-arbre gauche, le sous-arbre droit, puis traiter la racine.

42

Parcours en profondeur: préfixe

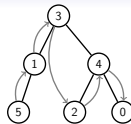


- Si le traitement d'un sommet consiste à l'afficher, on affichera dans l'ordre 3, 1, 5, 4, 2, 0.

```
static void Affiche(Arbre a) {  
    if (a==null) return;  
    System.out.println(a.val+" ");  
    Affiche(a.gauche);  
    Affiche(a.droite);  
}
```

43

Parcours en profondeur: infixe

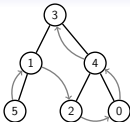


- Si le traitement d'un sommet consiste à l'afficher, on affichera dans l'ordre 5, 1, 3, 2, 4, 0.

```
static void Affiche(Arbre a) {  
    if (a==null) return;  
    Affiche(a.gauche);  
    System.out.println(a.val+" ");  
    Affiche(a.droite);  
}
```

44

Parcours en profondeur: postfixe



- Si le traitement d'un sommet consiste à l'afficher, on affichera dans l'ordre 5, 1, 2, 0, 4, 3.

```
static void Affiche(Arbre a) {  
    if (a==null) return;  
    Affiche(a.gauche);  
    Affiche(a.droite);  
    System.out.println(a.val+" ");  
}
```

45

De l'itératif? Parcourir en largeur?

- Voici un algorithme générique itératif de parcours d'arbre.

```
static void parcours(Arbre a) {  
    Sac s = new Sac(); // Le sac contient les sous-arbres  
    s.Ajouter(a); // encore à traiter  
    while (!s.EstVide()) {  
        Arbre t = s.Enlever(); // On traite un sommet  
        if (t != null) {  
            System.out.println(t.val);  
            s.Ajouter(t.gauche); // Et on stocke le travail futur  
            s.Ajouter(t.droite);  
        }  
    }  
}
```

- Si le sac est implémenté
 - ▶ par une pile, on parcourt en profondeur (préfixe).
 - ▶ par une file, on parcourt en largeur.

46

Aujourd'hui

Les graphes

Les arbres

Les arbres binaires

Programmer des arbres binaires

Parcours d'arbres

Les arbres généraux

Le codage de Huffman

47

Petit résumé

Il y a beaucoup de points de vue sur les arbres.

- le point de vue de la théorie des graphes:
Un arbre est un graphe (non-orienté) connexe sans cycle, avec possiblement un sommet distingué, que l'on appelle racine.
- le point de vue récursif.

$$A = \text{null} \uplus (A \times \text{int} \times A)$$

- le point de vue JAVA.



48

Les arbres k -aires

- Les points de vue précédents se généralisent aux arbres quelconques.
 - ▶ un sommet peut posséder cette fois un nombre quelconque de fils.
- Définition récursive:
 - ▶ L'arbre vide est un arbre.
 - ▶ Si L est une liste d'arbres, alors (r, L) est un arbre.
- Les algorithmes et notions précédentes se généralisent facilement⁴.

```
class Arbre {
    int val;
    ListeArbres fils;
    Arbre (int val, ListeArbres fils) {
        this.val = val;
        this.fils = fils;}}

class ListeArbres { ... }
```

⁴Il faut être souple sur la terminologie "liste" plus haut. La liste des fils peut éventuellement s'implémenter par un tableau, selon les circonstances.

49

Aujourd'hui

Les graphes

Les arbres

Les arbres binaires

Programmer des arbres binaires

Parcours d'arbres

Les arbres généraux

Le codage de Huffman

50

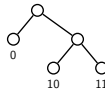
Les deux cours à venir

- Dans les deux cours à venir, nous verrons quelques applications des arbres.
- Commençons par une utilisation des arbres binaires pour la compression de données.
- Le codage de Huffman est utilisé (en post-traitement) dans les codages JPEG et MP3.
- Nous allons voir que le codage de Huffman est optimal pour un codage caractère par caractère (et si l'on "oublie" qu'il faut transmettre l'arbre).

51

Codes préfixes

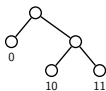
- On a vu qu'on pouvait associer un mot binaire (constitué que de 0 et de 1) à tout sommet d'un arbre binaire.



- Un ensemble fini de mots est appelé *un code préfixe* s'il correspond aux mots associés aux feuilles d'un arbre binaire.
 - ▶ Exemple: $\{0, 10, 11\}$ est le code préfixe associé à cet arbre.

52

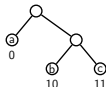
Codes préfixes



- Prop (admis). Tout mot binaire constitué de 0,1 admet au plus une décomposition comme concaténation de mots d'un code préfixe: cette décomposition, quand elle existe s'obtient en lisant le mot de gauche à droite.
 - ▶ Exemple: 01001100 se décompose en 0, 10, 0, 11, 0, 0.

53

Coder en binaire lettre par lettre



- Soit un ensemble fini de lettres (un *alphabet*) Σ .
- A partir d'un arbre binaire dont les feuilles sont les lettres de Σ , on peut coder sans ambiguïté tout mot sur Σ par un mot binaire.
 - ▶ Exemple: pour l'alphabet $\Sigma = \{a, b, c\}$, en utilisant l'arbre plus haut, *ababc* se code en binaire par 01001011 de longueur 8.

54

Coder en binaire lettre par lettre

- Une fois fixé un arbre binaire, la longueur du codage d'un mot ne dépend que des fréquences d'apparition de chacune des lettres dans ce mot.
 - ▶ Exemple: la longueur du codage du mot *ababc* est donné par $2 \times$ la longueur du codage de *a* + $2 \times$ la longueur du codage de *b* + $1 \times$ la longueur du codage de *c*.
- Connaissant ces fréquences d'apparition déterminer le code préfixe qui donne le code de longueur minimal?
- Un ensemble fini de mots est appelé *un code préfixe complet* s'il correspond aux mots associés aux feuilles d'un arbre binaire, dont tous les sommets internes (= qui ne sont pas des feuilles) sont d'arité 2.

55

Algorithme de Huffman

- Initialisation: pour chaque lettre,
 - ▶ on construit un arbre réduit à sa racine;
 - ▶ on fixe la valeur de cette racine à la fréquence de la lettre.
- Itération:
 - ▶ on fusionne les deux arbres dont les racines ont les valeurs minimales. C'est-à-dire: on construit un arbre qui possède comme fils ces deux arbres. La nouvelle racine a pour valeur la somme des valeurs des racines de ces deux arbres.
- Chaque itération réduit de un le nombre d'arbres. Le résultat final est un arbre binaire.
- Cet arbre binaire possède la propriété d'être optimal parmi les codes préfixes complets (exercice).

56

En pratique

- Il faut aussi transmettre l'arbre.
- Il existe une version dynamique de l'algorithme (voir poly).
- Il existe des techniques de compression sans perte de données plus efficaces (Zip-Lempel, codages arithmétiques) qui ne s'imposent pas de coder caractère par caractère.
- On peut aussi utiliser l'algorithme de Huffman pour coder par paires de caractères (ou k -uplets), plutôt que caractère par caractère.
- Complément: *L'entropie* de Shanon (l'information) d'une source d'information correspond à \inf sur k de la longueur moyenne du codage d'une lettre par Huffman. Cela correspond aussi à $-\sum_i p_i \log p_i$, si p_i est la fréquence de la lettre i .