Cours 8

Graphes orientés
Files de priorité

Jean-Jacques.Levy@inria.fr

http://jeanjacqueslevy.net

tel: 01 39 63 56 89

secrétariat de l'enseignement:

Catherine Bensoussan

cb@lix.polytechnique.fr

Aile 00, LIX

tel: 01 69 33 34 67

http://www.enseignement.polytechnique.fr/informatique/

Plan

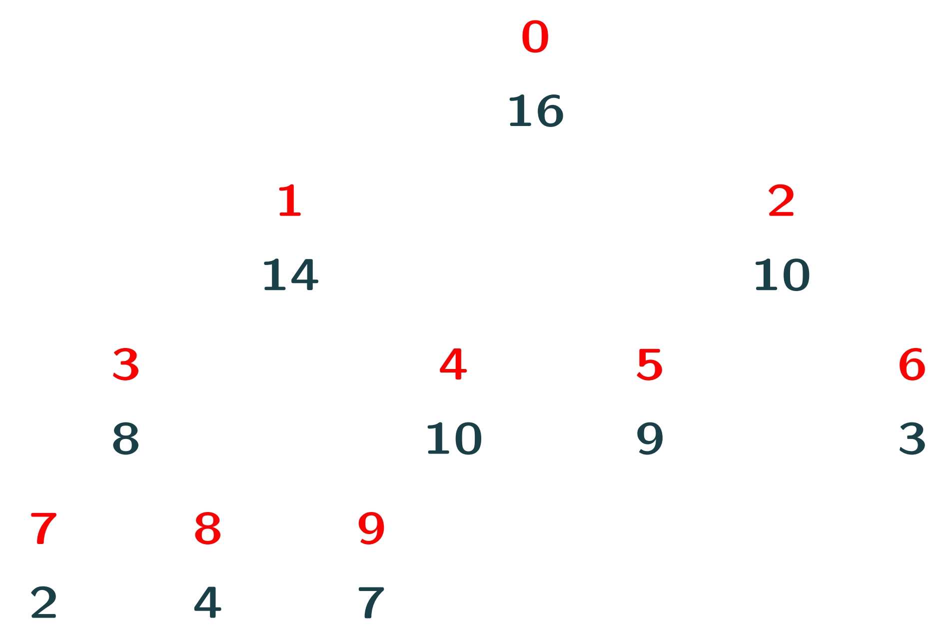
1. Files de priorité
2. Heapsort
3. Plus court chemin
4. Composantes fortement connexes
5. Warshall

Structure de tas

Représentation par des arbres binaires quasi-parfaits, dont tout noeud a une valeur supérieure à celle de ses fils.

Arbres représentés par des tableaux, “**tas**” (*heap*).

i	0	1	2	3	4	5	6	7	8	9
$a[i]$	16	14	10	8	10	9	3	2	4	7



Files de priorités

Représentation et ajout d'un élément

```
class FileP {
  int[] t; int taille;
  FileP (int n) { t = new int[n]; taille = 0; }

  static void ajouter (FileP f, int v)  {
    if (f.taille >= f.t.length)
      throw new Error ("File pleine.");
    ++f.taille; int i = f.taille - 1;
    while( i > 0 && f.t[(i - 1)/2] < v ) {
        f.t[i] = f.t[(i - 1)/2];
        i = (i - 1)/2;
    }
    f.t[i] = v;
  }

  static int premier (FileP f) { return f.t[0]; }
```

Ajout

16

14 10

8 10 9 3

2 4 7

16

14 10

8 10 9 3

2 4 7 15

16

14 10

8 15 9 3

2 4 7 10

16

15 10

8 14 9 3

2 4 7 10

Files de priorités

```
static int supprimer (FileP f) {
  if (f.taille <= 0)
    throw new Error ("File vide.");
  int res = f.t[0];
  f.t[0] = f.t[f.taille - 1]; --f.taille;
  int v = f.t[0]; int i;
  for (i = 0; 2*i + 1 < f.taille; ) {
      int j = 2*i + 1;
      if (j + 1 < f.taille && f.t[j + 1] > f.t[j])   ++j;
      if (v >= f.t[j])  break;
      f.t[i] = f.t[j]; i = j;
  }
  f.t[i] = v;
  return res;
}
```

Exercice 1 Complexité de l'ajout et de la suppression?

Suppression

16

15 10

8 14 9 3

2 4 7 10

10

15 10

8 14 9 3

2 4 7

15

10 10

8 14 9 3

2 4 7

15

14 10

8 10 9 3

2 4 7

Le tri par tas

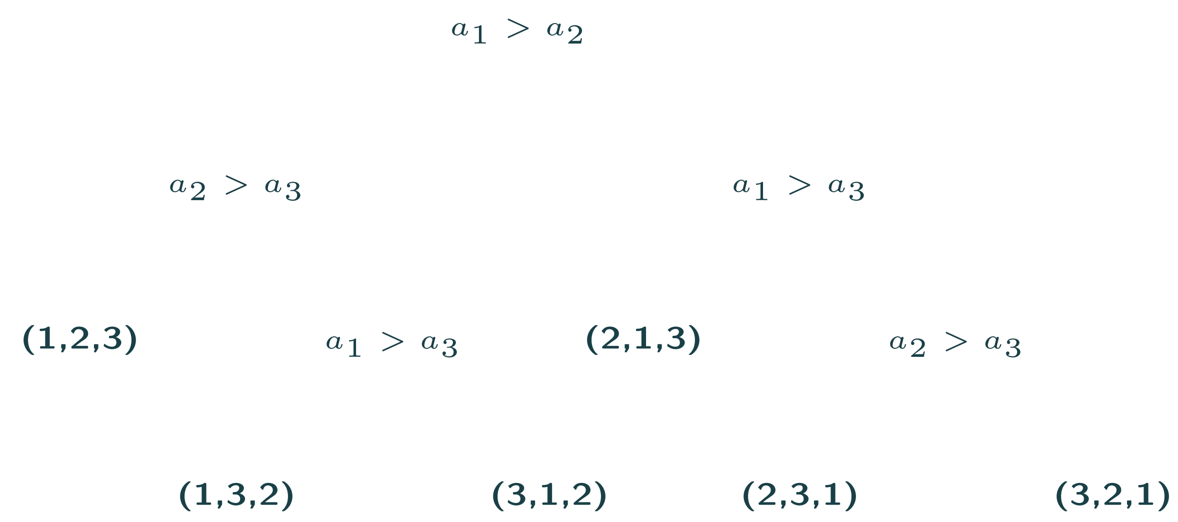
```
static void triParTas (int[ ] a) {
  FileP f = new FileP (a.length);
  for (int i = 0; i < a.length; ++i)
    ajouter (f, a[i]);
  for (int i = a.length - 1; i >= 0; --i)
    a[i] = supprimer(f);
}
```

- **la file et le tableau peuvent être confondus.**
- **complexité en $O(n \log n)$, mais en fait un des plus mauvais tris existants (la constante devant le $n \log n$ est élevée)**

Optimalité du tri

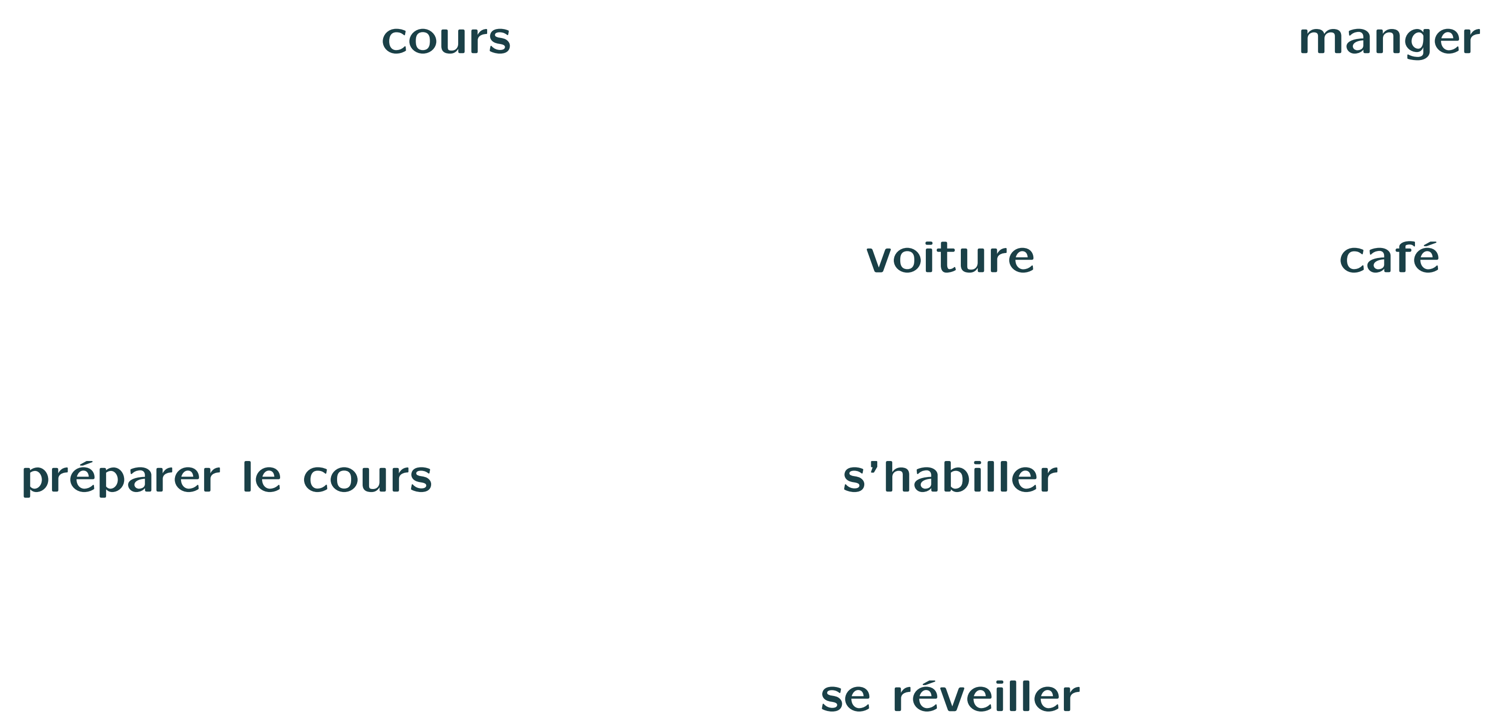
Théorème 1 Le tri de n éléments, fondé uniquement sur les comparaisons des éléments deux à deux, fait au moins $O(n \log n)$ comparaisons.

Démonstration Tout arbre de décision pour trier n éléments a $n!$ feuilles représentant toutes les permutations possibles. Un arbre binaire de $n!$ feuilles a une hauteur de l'ordre de $\log n! \simeq n \log n$ par la formule de Stirling.



Tri topologique

Tester si un graphe $G = (V, E)$ est acyclique et donner une liste des noeuds $\langle n_i \mid i = 1..n \rangle$ dans un ordre compatible avec celui du graphe (n_i successeur de n_j implique $i < j$).



(tri topologique = trouver un ordre total compatible avec un ordre partiel donné)

Tri topologique

Ordre de lecture
des chapîtres
du livre de
H. P. Barendregt
The λ-calculus:
its syntax and semantics
North-Holland, 1980

Détection de cycles dans un graphe orienté

```
static boolean acyclique (Graphe g) {
  int n = g.succ.length;
  boolean[] vu = new boolean[n];
  boolean[] enCours = new boolean[n];
  for (int i=0; i < g.succ.length; ++i) vu[i] = enCours[i] = false;
  for (int i=0; i < g.succ.length; ++i)
    if ( !vu[i] && cycliqueEn (g, i, vu, enCours) )
        return false;
  return true;
}

static boolean cycliqueEn (Graphe g, int i, boolean[] vu, boolean[] enCours) {
  enCours[i] = true;
  for (Liste ls = g.succ[i]; ls != null; ls = ls.suivant) {
    int x = ls.val;
    if ( enCours[x] || !vu[x] && cycliqueEn (g, x, vu, enCours) )
        return true;
  }
  enCours[i] = false; vu[i] = true;
  return false;
}
```

Exercice 2 Programmer le tri topologique. Complexité?

Fermeture transitive

Le graphe est représenté par sa matrice M de connexion (matrice d'adjacence). Il s'agit de calculer

$$M^* = M^0 + M + M^2 + M^3 + \cdots$$

Les chemins dont tous les noeuds sont distincts ont une longueur strictement inférieure à n. Il suffit de calculer

$$M^* = M^0 + M + M^2 + M^3 + \cdots M^{n-1}$$

La multiplication de matrice $n \times n$ a une complexité $O(n^3)$. La fermeture transitive peut donc se faire en $O(n^4)$ opérations.

Faire mieux?

Algorithme de Warshall

Enumération des chemins selon le plus grand des noeuds intermédiaires par lesquels ils passent.

$$C_{-1} = M$$
$$C_k = C_{k-1} \cup \{(x,y) \mid (x,k) \in C_{k-1}, (k,y) \in C_{k-1}\}$$

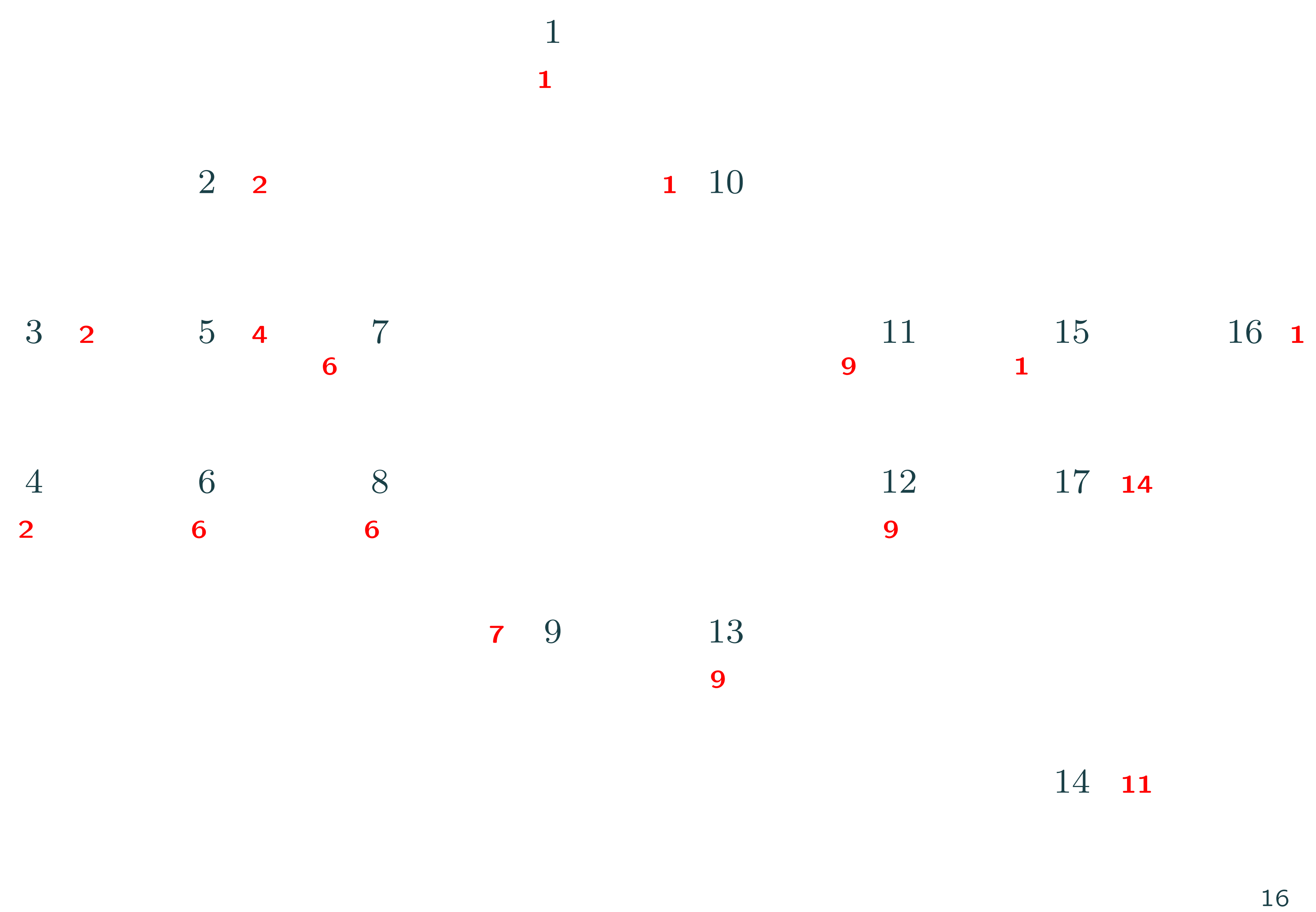
```
static Graphe fermetureTransitive (Graphe g) {
  int n = g.m.length;
  Graphe c = copieGraphe (g);
  for (int k = 0; k < n; ++k)
    for (int i = 0; i < n; ++i)
      for (int j = 0; j < n; ++j)
          c.m[i][j] = c.m[i][j] || (c.m[i][k] && c.m[k][j]);
  return c;
}
```

Exercice 3 Complexité?

Composante fortement connexe

Dans un graphe non orienté, connexité = facile. Dans un graphe orienté, composantes **fortement connexes** = parties maximales où toute paire de noeuds distincts est reliée par un chemin.

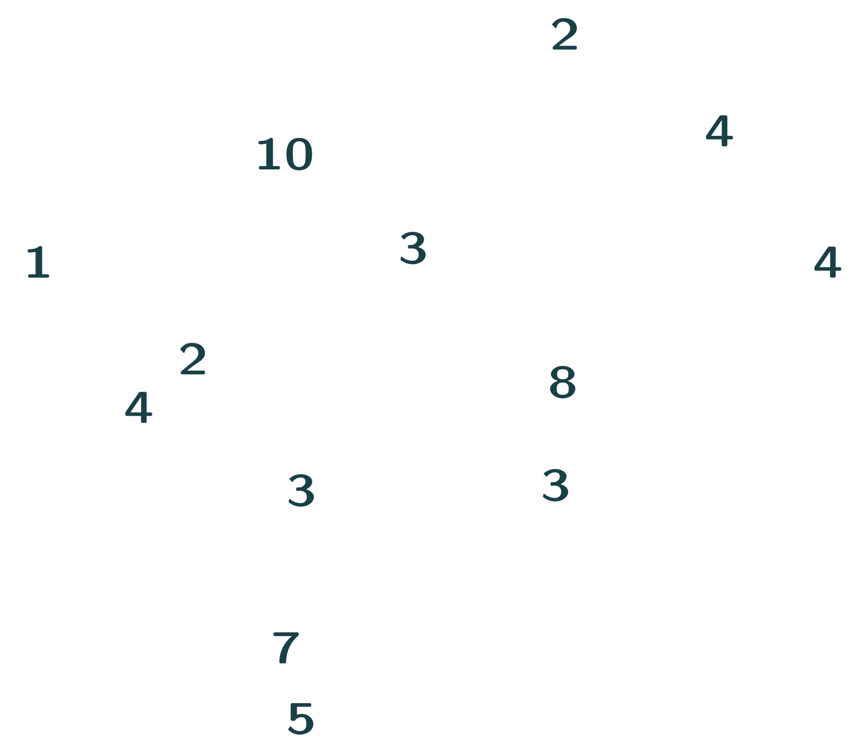
Points d’attache (cf. biconnexité)



Calcul des composantes fortement connexes

```
static int attache (Graphe g, int k) {
  num[k] = ++id; Pile.empiler(k, p);
  int min = id;
  for (Liste ls = g.succ[k]; ls != null; ls = ls.suivant) {
    int x = ls.val; int m;
    if (num[x] == -1) m = attache (g, x);
    else m = num[x];
    if (m < min) min = m;
  }
  if (min == num[k]) {
    int y;
    do {
      y = Pile.depiler(p);
      System.out.print (y + " ");
      num[y] = g.succ.length;
    } while (y != k);
    System.out.println();
  }
  return min;
}
```

Graphe valué



Chaque arc a un poids (distance).

Plus court chemin dans un graphe

- graphe valué a une notion de distance (≥ 0) attachée à tout arc
- on suppose les distance positives $d(i,j) \geq 0$ pour tout i,j.

- calculer le plus court chemin entre deux noeuds i et j dans un graphe valué
- $dist_{i,j} = \min\{dist_{i,k} + d(k,j) \mid 0 \leq k < n\}$

- Attention: problème différent de calculer le plus court chemin entre tous les noeuds.

Algorithme de Dijkstra

```
static int[] plusCourtChemin (GrapheV g, int s, int d) {
  int n = g.succ.length; FileP q = new FileP (n);
  int[] dMin = new int[n]; int[] prec = new int[n];
  for (int i = 0; i < n; ++i) { dMin[i] = Integer.MAX_VALUE; prec[i] = -1; }
  dMin[s] = 0;
  FileP.ajouter (s, 0, q);
  while (q.taille != 0) {
    int k = FileP.supprimer (q);
    if (k == d) return prec;
    for (ListeSucc ls = g.succ[k]; ls != null; ls = ls.suivant) {
      int x = ls.val; int w = ls.dist;
      if (dMin[x] > dMin[k] + w) {
        dMin[x] = dMin[k] + w;
        FileP.modifier (x, dMin[x]);
        prec[x] = k;
      }
    }
  }
  return prec;
}
```

Algorithme de Dijkstra – bis

Exercice 4 Complexité de l'algorithme de Dijkstra

Exercice 5 Complexité en recherchant le minimum simplement (sans file de priorité)?

Exercice 6 Comment transformer l'algorithme de Dijkstra pour le trouver le plus court chemin d'un noeud à tous les autres dans le graphe.

Exercice 7 Idem en plus court chemin depuis tous les noeuds jusqu'à un même noeud destination.

Exercice 8 Idem pour les plus courts chemins entre tous les noeuds.

En TD – problèmes

- **Analyse de dépendances**
- **Commande** *Make*
- **Composante fortement connexes**