Combien d'objets dans une image?

Jean-Jacques Lévy
INRIA

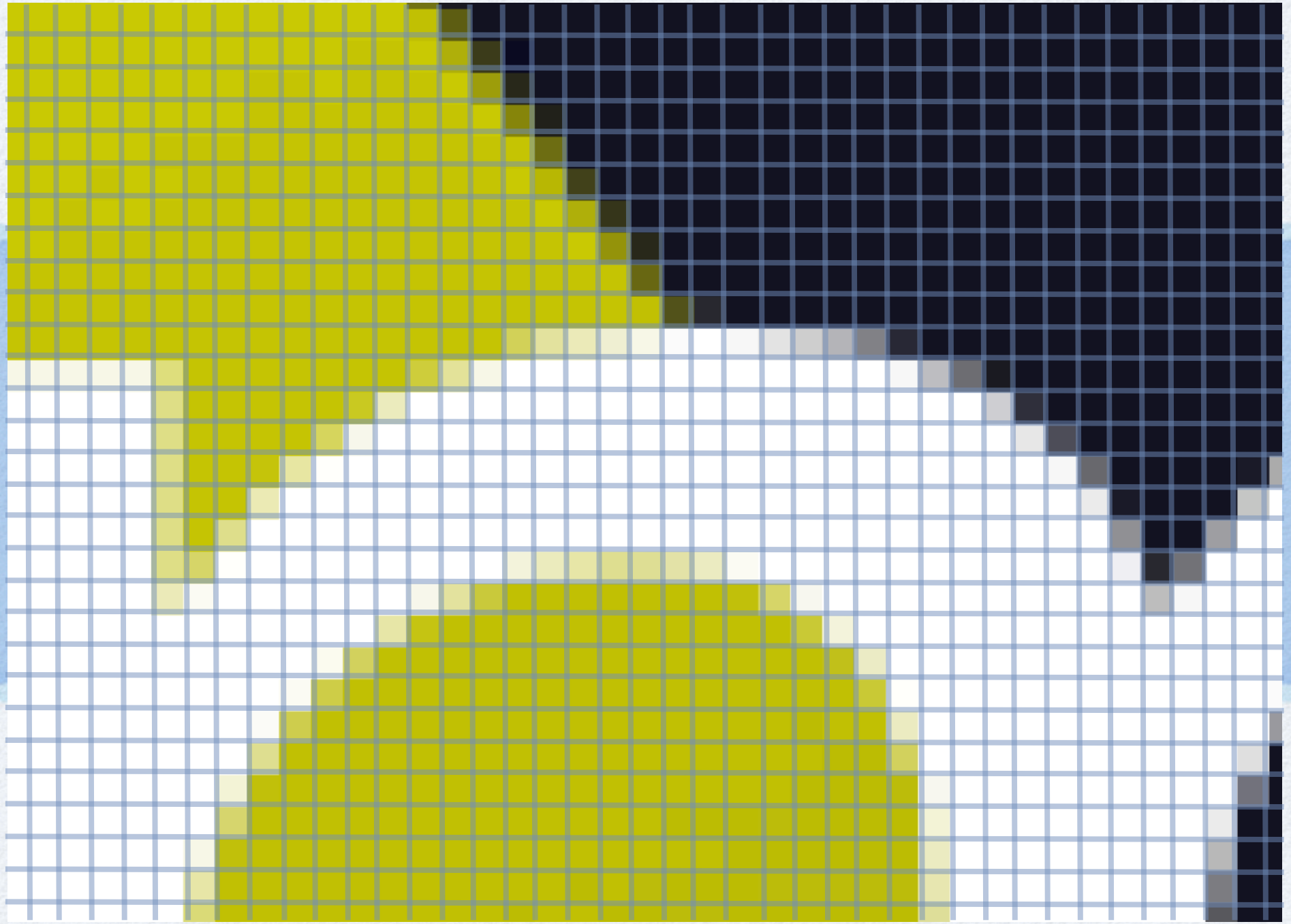
CENTRE DE RECHERCHE COMMUN

INRIA
MICROSOFT RESEARCH


Combien d'objets
dans une image?
Jean-Jacques Lévy
INRIA
CENTRE DE RECHERCHE
COMMUN
INRIA
MICROSOFT RESEARCH


Combien
dans une


mb



PIXELS

(pictures elements)

800

1200

10^6

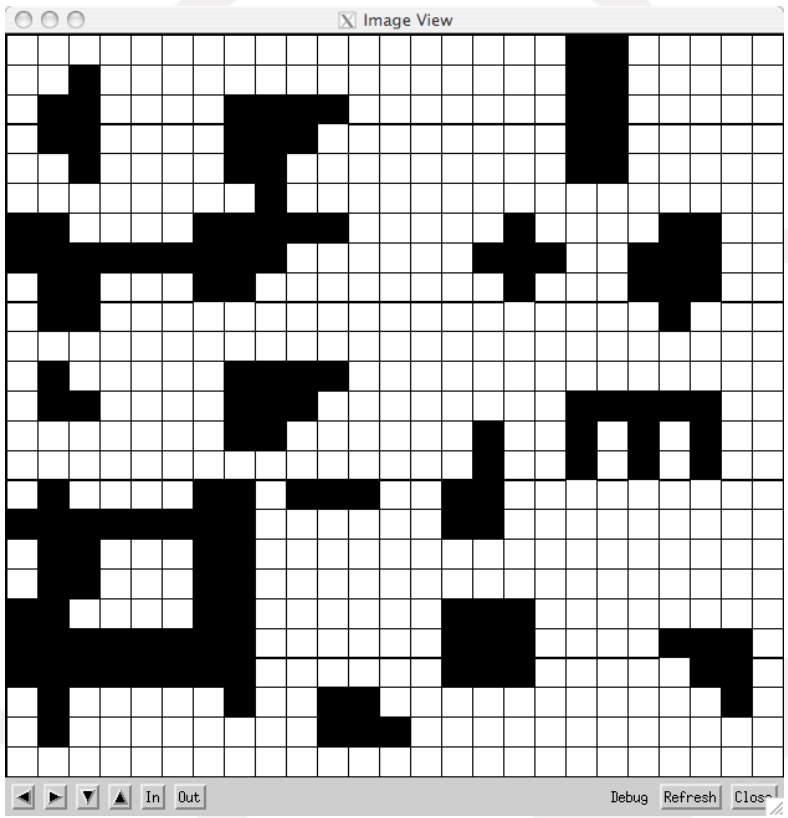
1Mpix

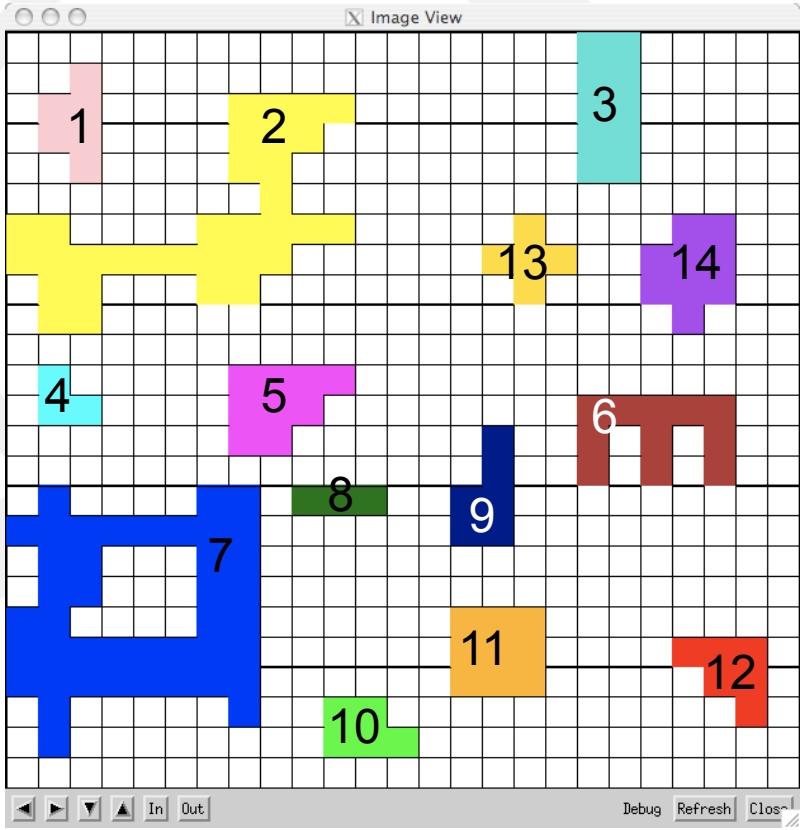
Problème et Algorithme

Qu'est ce qu'un objet?

- ensemble de pixels contigus similaires
  - similaire ?
- simplification
  - images niveau de gris (255 valeurs)
  - 0 = noir, 255 = blanc
  - similaires = contigus avec valeurs voisines
- donner un numéro différent à chaque objet
- le nombre d'objets est le numéro max

Étiquetage





15 objets dans cette figure

Algorithme naïf

1) choisir un point non parcouru

2) parcourir tous les pixels connexes similaires

3) et recommencer tant qu'il y a un point non parcouru

Complexité élevée:

- trouver un pixel non parcouru
- explorer la zone connexe (quelle direction?)

Algorithme

1) première passe

- balayer les pixels en donnant un nouveau numéro à tout nouvel objet

2) deuxième passe

- générer les équivalences dues aux contiguités découvertes au cours du balayage

3) troisième passe

- compter le nombre de classes d'équivalence

Complexité:

- parcourir 2 fois l'image (coût linéaire)
- gérer efficacement les classes d'équivalences

Animation

système Polka d’animation d’algorithmes

Classes d’équivalence

CENTRE DE RECHERCHE COMMUN
INRIA MICROSOFT RESEARCH

Classes d'équivalences

"Union-Find"

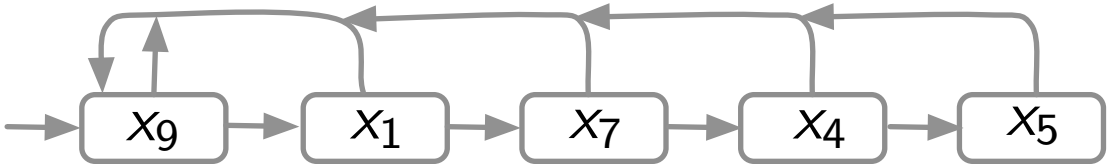
- objets $x_1, x_2, \ldots, x_n$
- équivalences
- trouver la classe de x_p
- 3 opérations:
  - $\texttt{NEW}(x)$ nouvel objet
  - $\texttt{FIND}(x)$ trouver représentant canonique
  - $\texttt{UNION}(x, y)$ fusionner 2 classes d'équivalence

Classes d'équivalences

"Union-Find"

- 3 opérations:
  - $\texttt{NEW}(x)$ nouvel objet
  - $\texttt{FIND}(x)$ trouver représentant canonique
  - $\texttt{UNION}(x, y)$ fusionner 2 classes d'équivalence

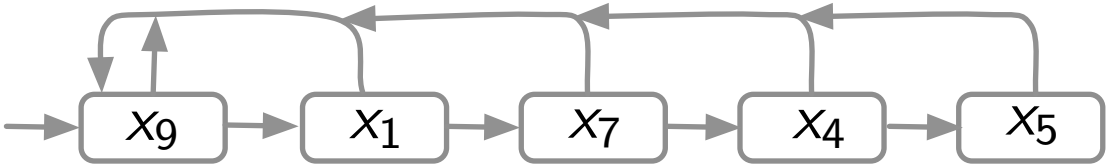
- représentation par des listes chaînées
  - canonique en tête de liste
  - tous les éléments de la classe pointent sur lui
  - exemple: $x_7 = x_4, x_1 = x_9, x_5 = x_1, x_5 = x_4$

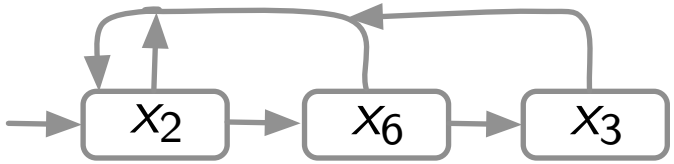


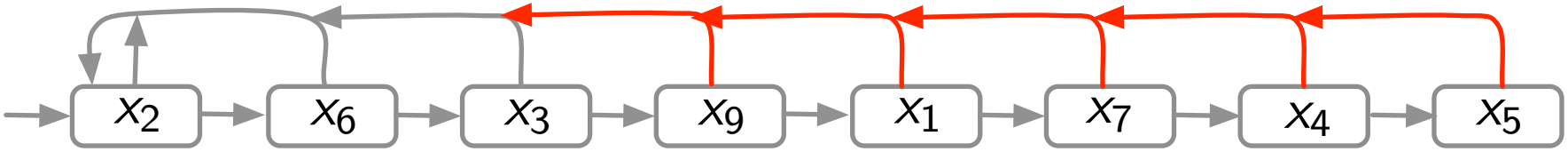
Classes d'équivalences

"Union-Find"

- opération délicate
  - `UNION`(x, y) fusionner 2 classes d'équivalence





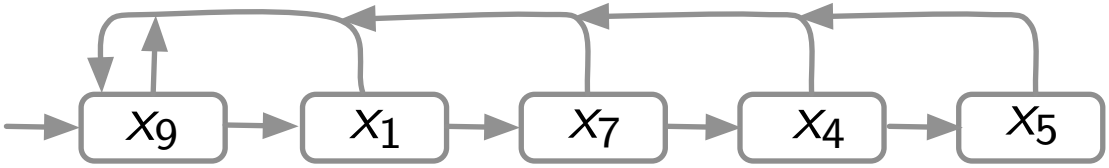


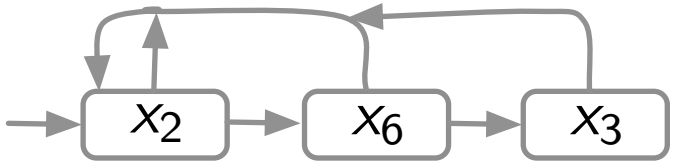
$O(n)$ opérations

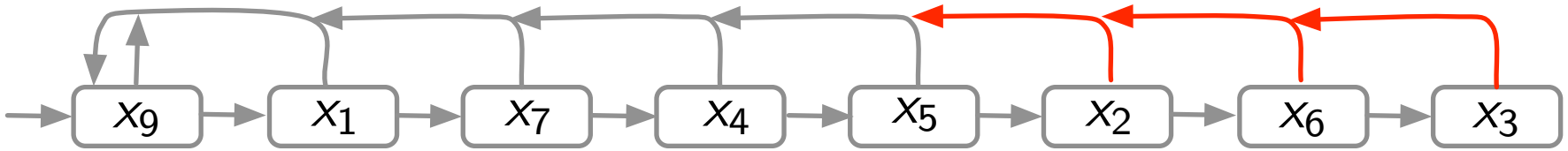
Classes d'équivalences

"Union-Find"

- opération délicate
  - `UNION`(x, y) fusionner 2 classes d'équivalence





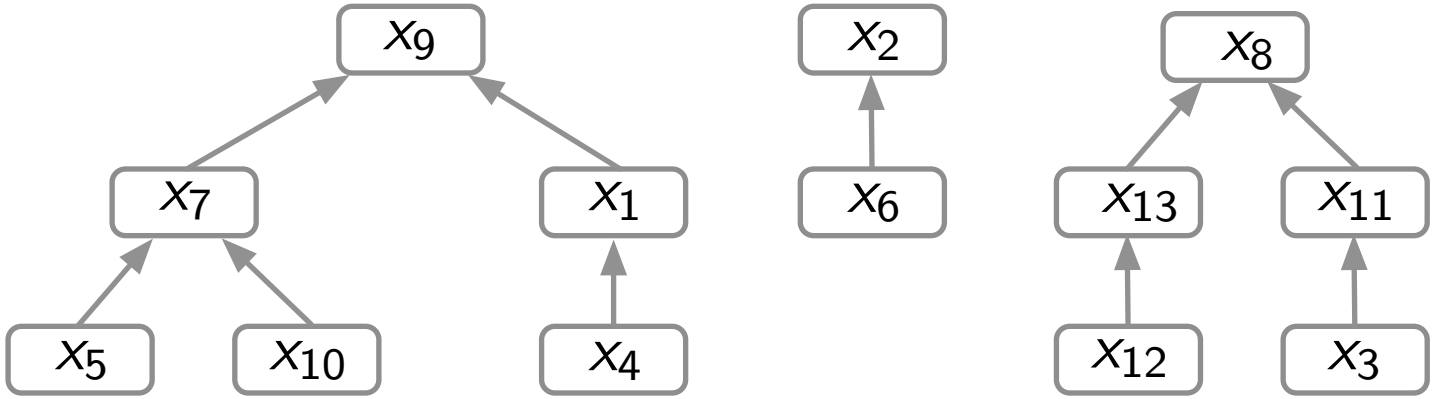


$O(\log n)$ opérations

Classes d'équivalences

"Union-Find"

- avec une arborescence



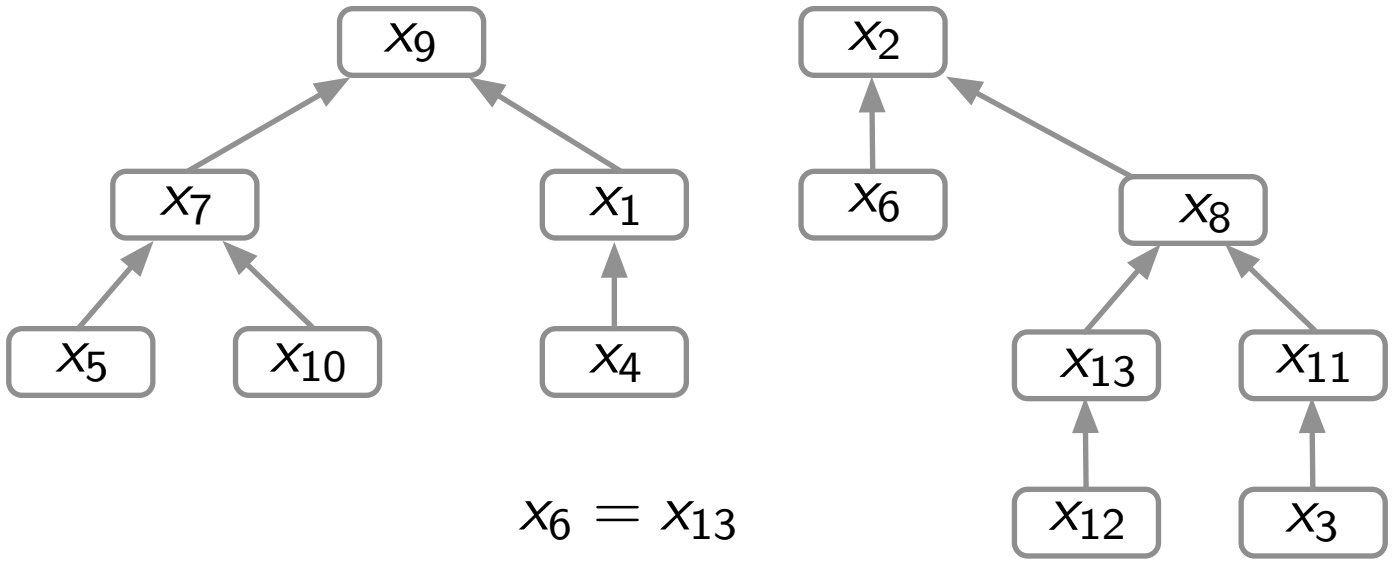
- représentée par le tableau des ancètres directs:

père	0	9	2	11	1	7	2	9	8	9	7	8	13	8

Classes d'équivalences

"Union-Find"

- la fusion crée des déséquilibres
  - `UNION`(x, y) fusionner 2 classes d'équivalence

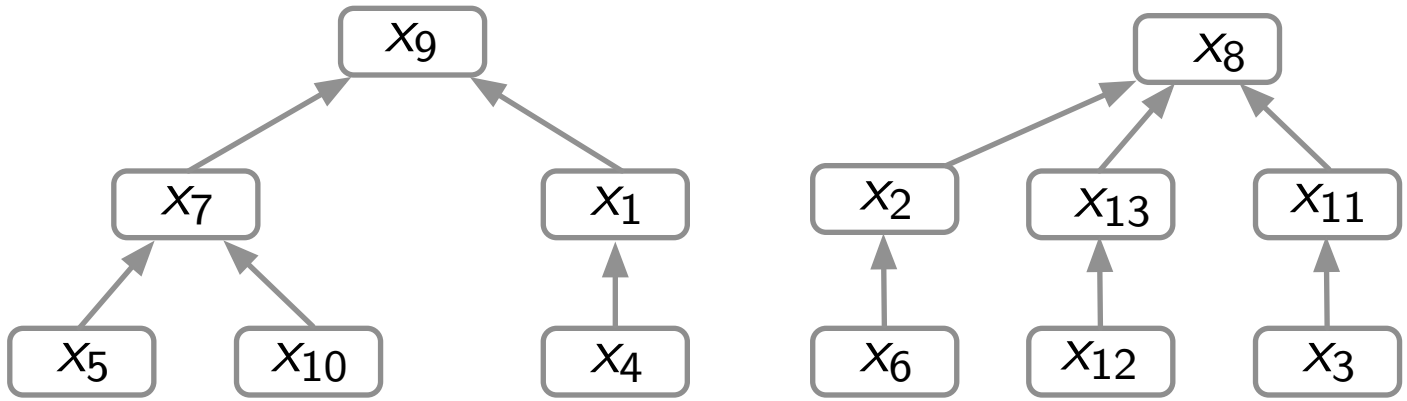


$x_6 = x_{13}$

Classes d'équivalences

"Union-Find"

- équilibrer l'arborescence
  - `UNION`(x, y) fusionner 2 classes d'équivalence



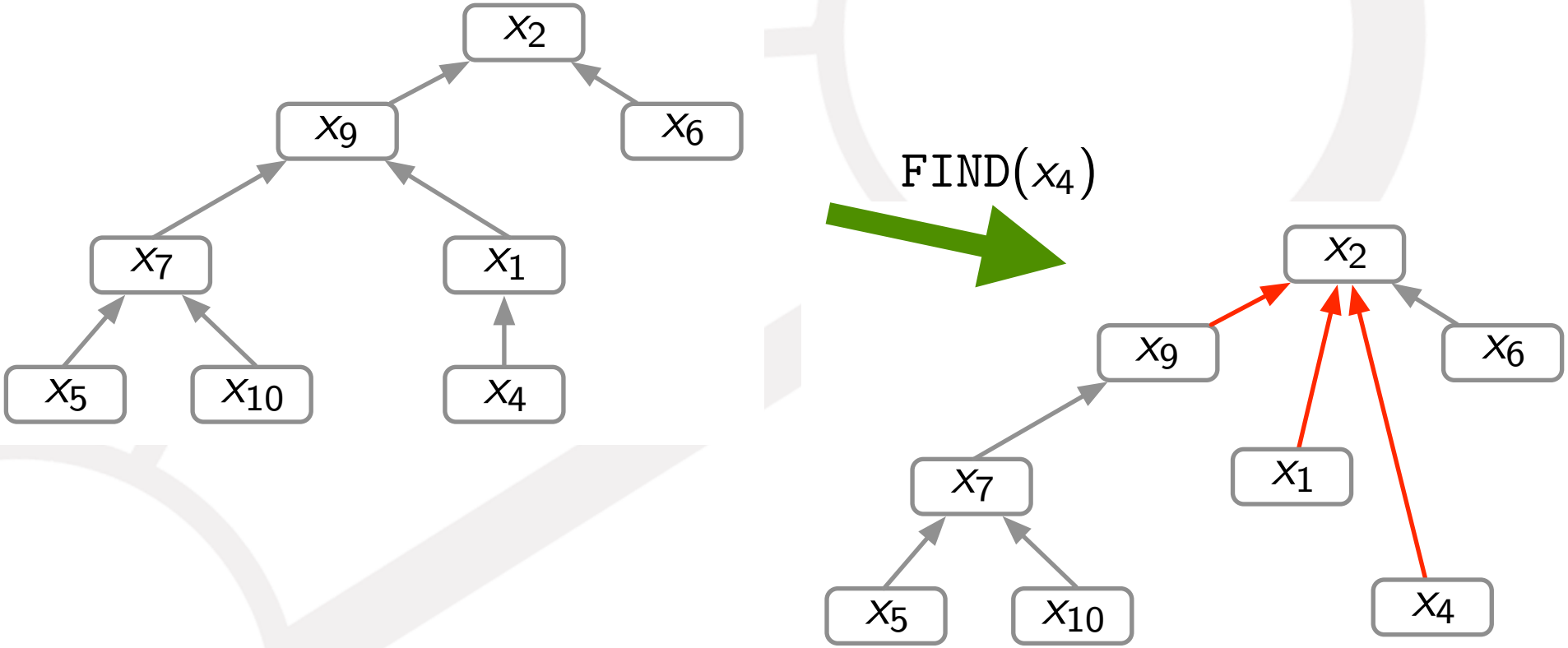
il suffit de maintenir la hauteur des arbres dans chaque noeud

$O(\log n)$ opérations pour `FIND`(x)

Classes d'équivalences

"Union-Find"

- compression du chemin vers le canonique
  - `FIND`(x) a un effet de bord sur l'arborescence



Classes d'équivalences

"Union-Find"

- la programmation de la compression est récursive

```
function FIND (x) =
  if pere[x] == x then return x;
  else {
    int y = FIND (pere[x]);
    pere[x] = y;
    return y;
  };
```

ZZZ: écrire cette fonction sans récursion est bien plus délicat

Classes d'équivalences

"Union-Find [R. Tarjan]"

- Si on fusionne en équilibrant et si on trouve le canonique en compressant les chemins d'accès, la complexité pour m opérations est $O(\alpha(m, n))$

$$\alpha(m, n) = \min\{i \mid i \geq 1,\ A(i, \lfloor m/n \rfloor) > \log n\}$$

où la fonction d'Ackermann $A(m, n)$ est définie par

$$\begin{aligned} A(0, n) &= n + 1 \\ A(m, 0) &= A(m - 1, 1) \\ A(m, n) &= A(m - 1, A(m, n - 1)) \end{aligned}$$

D'où $\alpha(m, n) \simeq 4$

la fonction d'Ackermann croit plus vite que toutes les fonctions récursives primitives (i.e. les programmes avec des boucles for)

Complexité

Soit N le nombre de pixels

1) première passe

- balayer les pixels en donnant un nouveau numéro à tout nouvel objet

$$O(N)$$

2) deuxième passe

- générer les équivalences dues aux contiguités découvertes au cours du balayage

$$O(N)$$

3) troisième passe

- compter le nombre k de classes d'équivalence

$$O(N \times \alpha(N, k))$$

Démo

Programme

```
let main () =
  let nargs = Array.length (Sys.argv) in
  let framefmt = if nargs >= 2 then Sys.argv.(1) else "" in
  if nargs >= 3 then threshold := int_of_string Sys.argv.(2);
  open_graph framefmt;

  let ncols = read_int() in
  let nlignes = read_int() in
  let b = bmap_read nlignes ncols in
  bmap_display b;
  pause();

  let ob = new_obmap b in
  bmap_display (color_obmap ob);
  pause();

  let p = init_set (!nObjects+1) in
  compute_equalities p b ob;
  let ob2 = apply_equalities_to p ob in
  bmap_display (color_obmap ob2);
  pause();
;;

main();;
```

Programme

```
let marge = 0;;

let bmap_display b =
  let bi = make_image b in
    draw_image bi marge marge;;

let bmap_read nlignes ncols =
  let b = Array.make_matrix nlignes ncols 0 in
  for i = 0 to nlignes - 1 do
    let s = read_line() in
    let xs = ref (Str.split (Str.regexp "[ \t]+") s) in
    for j = 0 to ncols - 1 do
      let c = int_of_string (List.hd !xs) in
        b.(i).(j) <- rgb c c c;
        xs := List.tl !xs;
    done;
  done;
  b;;

let pause () =
  match wait_next_event [Button_down] with
      _ -> () ;;
```

Programme

```
let nObjects = ref (-1);;
let new_object () =
  incr nObjects;
  !nObjects;;

let threshold = ref 7;;

let same_gray_color b i j i0 j0 w =
  (* b matrice w x h *)
  (* 0 <= i < w, 0 <- j < h, -1 <= i0 < w, -1 <= j0 < w *)
  if i0 < 0 || j0 < 0 || j0 >= w then false else
    abs ((b.(i).(j) land 0xff) - (b.(i0).(j0) land 0xff)) < !threshold ;;

let new_obmap b =
  let h = Array.length b and
      w = Array.length b.(0) in
  let ob = Array.make_matrix h w 0 in
  for i = 0 to h - 1 do
    for j = 0 to w - 1 do
      ob.(i).(j) <-
        if same_gray_color b i j (i-1) j w then ob.(i-1).(j) else
        if same_gray_color b i j i (j-1) w then ob.(i).(j-1) else
        if same_gray_color b i j (i-1) (j-1) w then ob.(i-1).(j-1) else
        if same_gray_color b i j (i-1) (j+1) w then ob.(i-1).(j+1) else
          new_object();
    done;
  done;
  ob;;

let color_obmap ob =
  let h = Array.length ob and
      w = Array.length ob.(0) in
  let cb = Array.make_matrix h w 0 in
  for i = 0 to h - 1 do
    for j = 0 to w - 1 do
```

RECHERCHE COMMUN
INRIA MICROSOFT RESEARCH

Programme

```
let compute_equalities p b ob =
  let h = Array.length ob and
      w = Array.length ob.(0) in
  for i = 0 to h - 1 do
    for j = 1 to w - 1 do
      let c = ob.(i).(j) and c' = ob.(i).(j-1) in
      if c <> c' && same_gray_color b i j i (j-1) w then
        union p c c';
    done
  done;;

let apply_equalities_to p ob =
  let h = Array.length ob and
      w = Array.length ob.(0) in
  for i = 0 to h - 1 do
    for j = 0 to w - 1 do
      ob.(i).(j) <- find p ob.(i).(j);
    done
  done;
  ob;;
```

Programme

```
let init_set n = Array.init n (function i -> i);;

let rec find p x =
  if p.(x) = x then x else
    let y = find p (p.(x)) in
    p.(x) <- y;
    y;;

let union p x y =
  p.(find p y) <- p.(find p x) ;;
```

Union-Find = 8 lignes

Programme entier = 246 lignes

Conclusion

Conclusion

- Derrière d'anodins problèmes, il y a des **algorithmes** ou des **fonctions logiques** intéressants

- Le problème algorithmique ne représente qu'une **infime** partie du programme (ici 3%)

- Les techniques de programmation **s'apprennent** (récursion, utilisation des bibliothèques, structuration du programme). Cela prend du temps de les maitriser.

- C'est **amusant** ...

CENTRE DE RECHERCHE COMMUN

INRIA
MICROSOFT RESEARCH