# L3-Intership: Formal Proof of Uno and Yagiura's Algorithm

Fleury Mathias

ENS Rennes

31 décembre 2013

FIGURE – Arianne 5 (La Presse)

- preuve formelle de ~~programme~~
- preuve formelle d'*algorithme* :
  - génétique
  - théorie des graphes : décomposition modulaire

1. Décomposition modulaire de graphe modulaire
   - Permutation
   - Décomposition modulaire
   - Tracé du graphe de permutations
   - Particularité
2. Algorithme
   - Plus naïf...
   - Algorithme quadratique
   - Vers l'algorithme linéaire
   - L'algorithme linéaire
3. Démonstration
   - Coq
   - Version quadratique
   - Démonstration
4. Conclusion

# Plan

1. Décomposition modulaire de graphe modulaire
   - Permutation
   - Décomposition modulaire
   - Tracé du graphe de permutations
   - Particularité
2. Algorithme
   - Plus naïf...
   - Algorithme quadratique
   - Vers l'algorithme linéaire
   - L'algorithme linéaire
3. Démonstration
   - Coq
   - Version quadratique
   - Démonstration
4. Conclusion

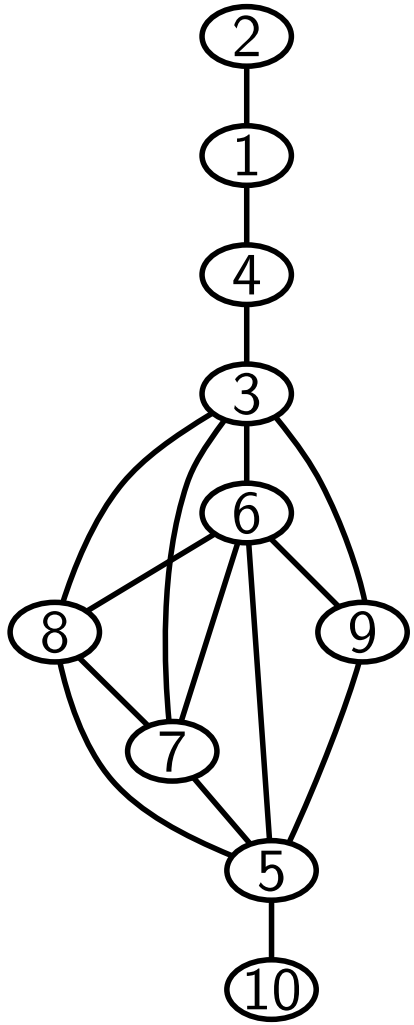
# Définition de la décomposition modulaire

**Module**

Pour $G = (S, A)$.
$M \subset S$, on ne peut pas distinguer les éléments de $M$ pour $S \setminus M$ :

$$\forall (u, v) \in M^2, \forall x \in A \setminus M,$$

$x$ adjacent à $u$ et $v$ ou aucun




(a) Graphe de permutation

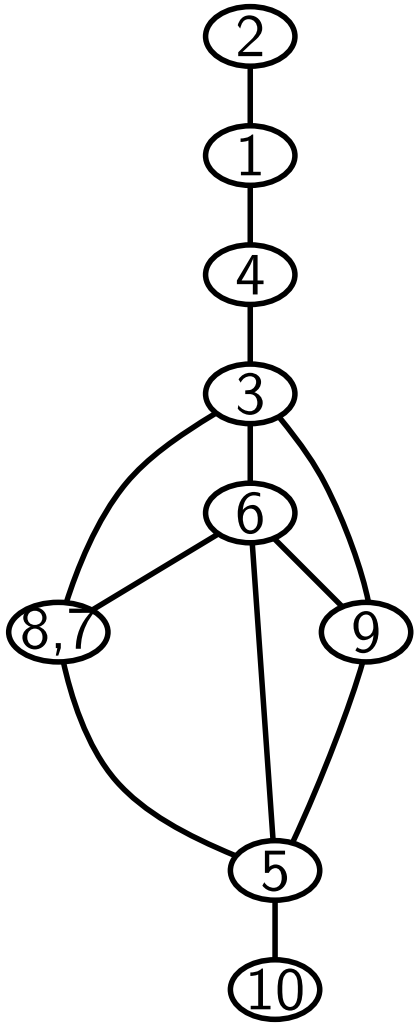
# Définition de la décomposition modulaire

**Module**

Pour $G = (S, A)$.
$M \subset S$, on ne peut pas distinguer les éléments de $M$ pour $S \setminus M$ :

$$\forall (u, v) \in M^2, \forall x \in A \setminus M,$$

$x$ adjacent à $u$ et $v$ ou aucun




(a) Graphe de permutation

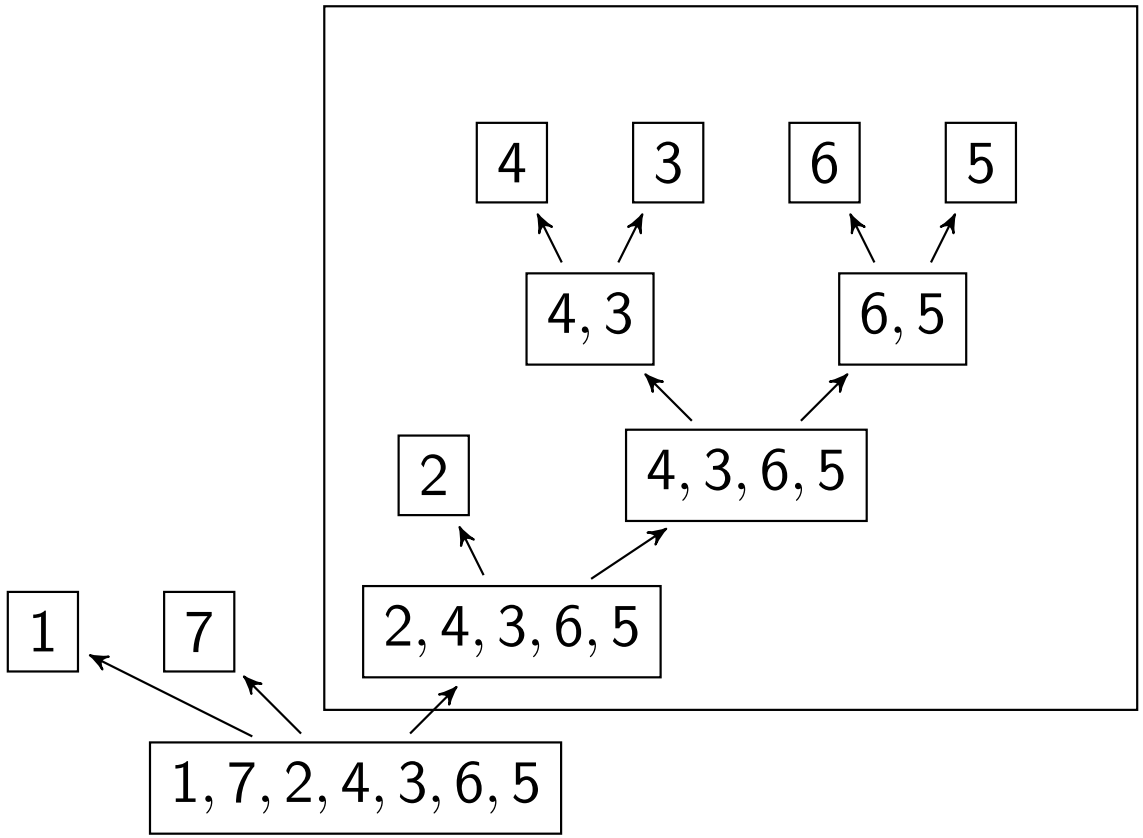
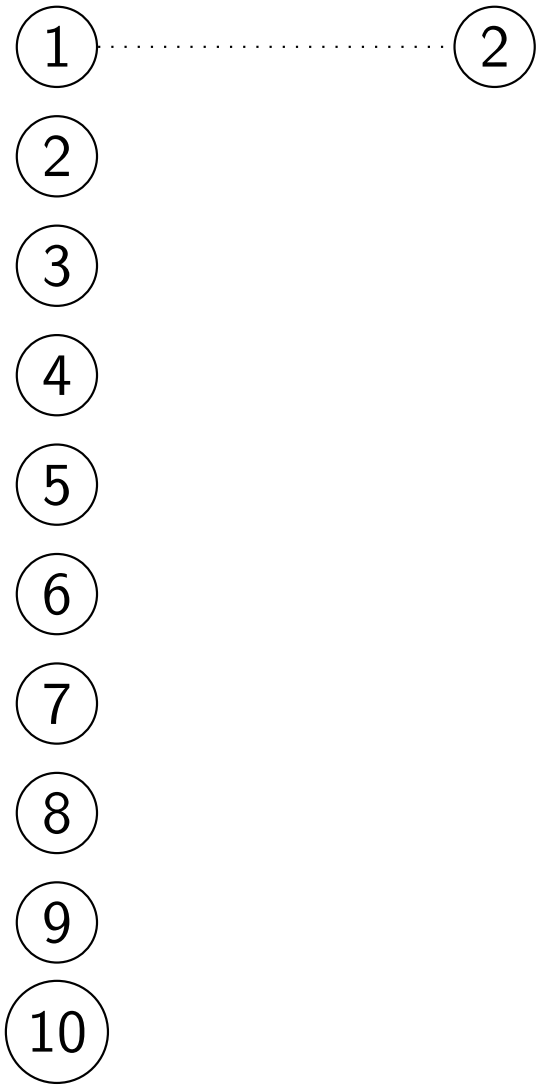



FIGURE – Décomposition modulaire

# Tracé de la permutation

| $i$ | 1 | 2 | 3 | ... | 10 |
|---|---|---|---|---|---|
| $\sigma$ | 2 | | | ... | |

(a) Graphe de permutation

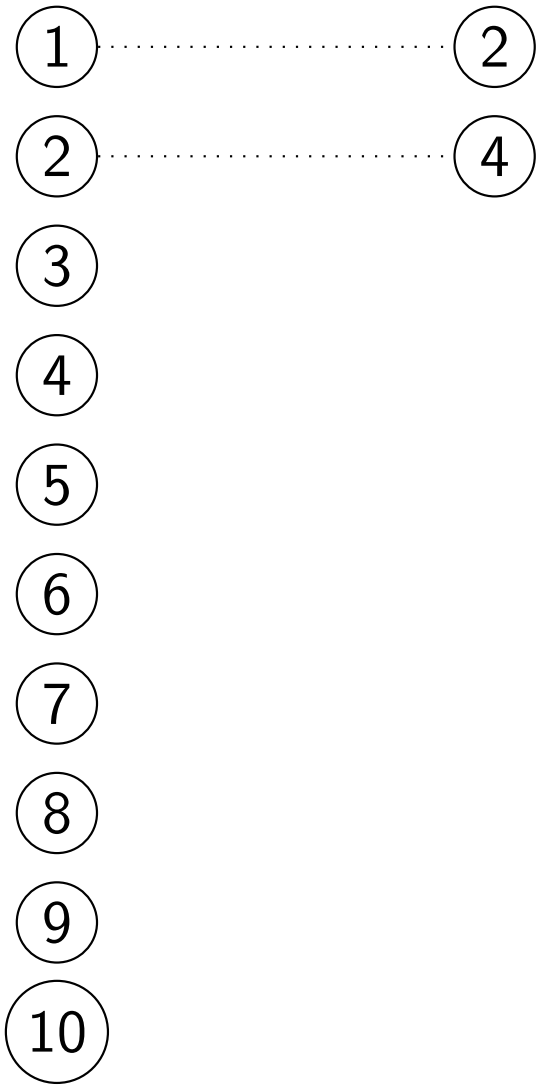



(b) Permutation

# Tracé de la permutation

| $i$ | 1 | 2 | 3 | ... | 10 |
|---|---|---|---|---|---|
| $\sigma$ | 2 | 4 | | ... | |

(a) Graphe de permutation

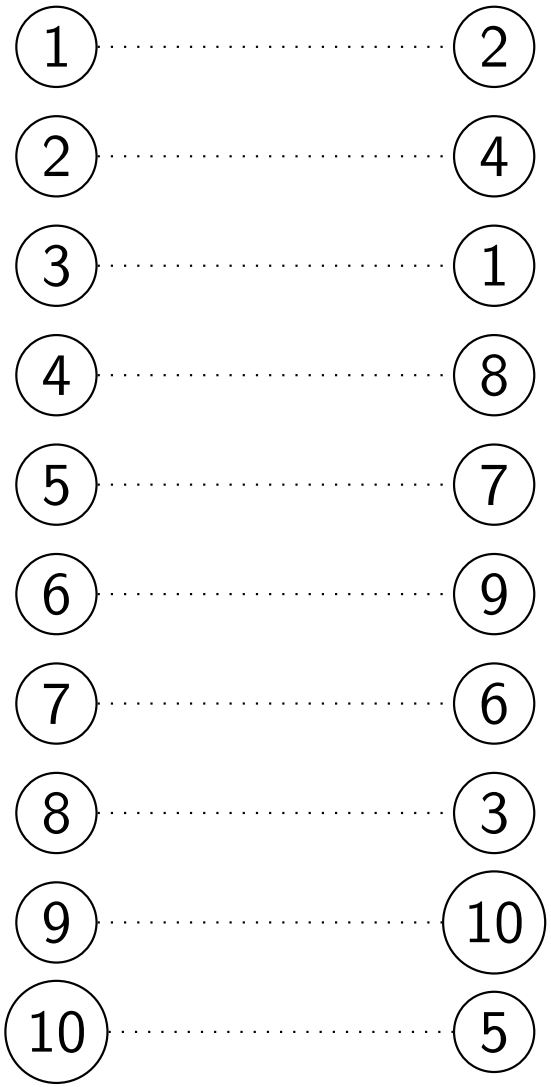



(b) Permutation

# Tracé de la permutation

| $i$ | 1 | 2 | 3 | ... | 10 |
|---|---|---|---|---|---|
| $\sigma$ | 2 | 4 | 1 | ... | 5 |

(a) Graphe de permutation

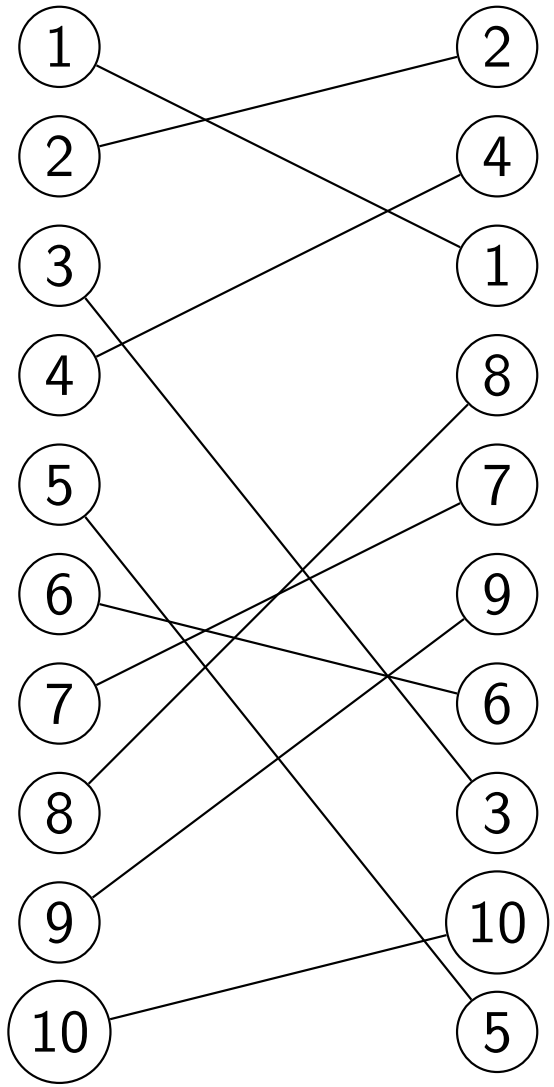



(b) Permutation

# Tracé de la permutation

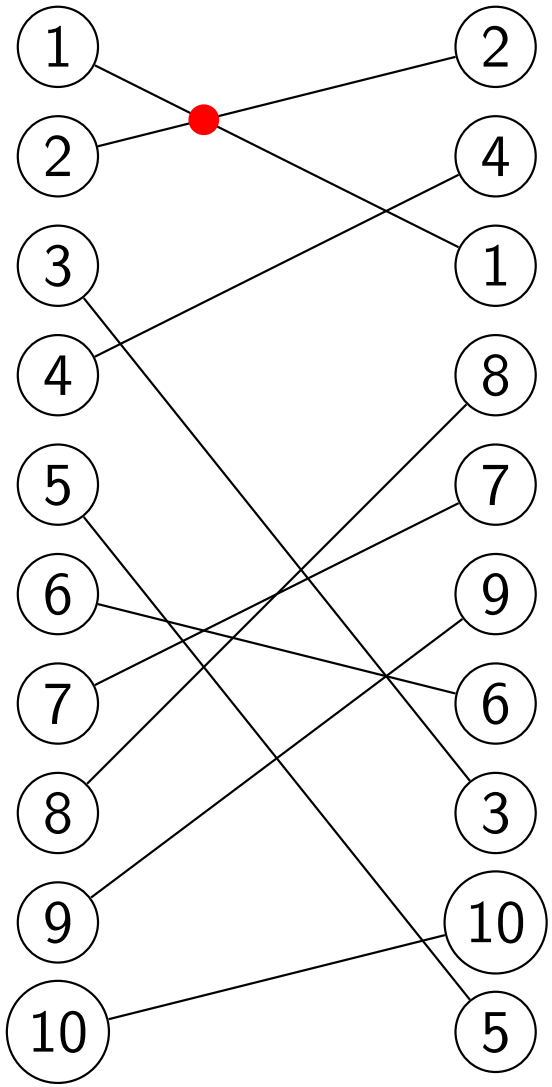
| $i$ | 1 | 2 | 3 | ... | 10 |
|---|---|---|---|---|---|
| $\sigma$ | 2 | 4 | 1 | ... | 5 |

(a) Graphe de permutation

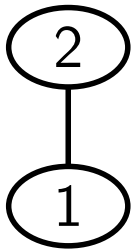



(b) Permutation

# Lien avec les graphes de permutations

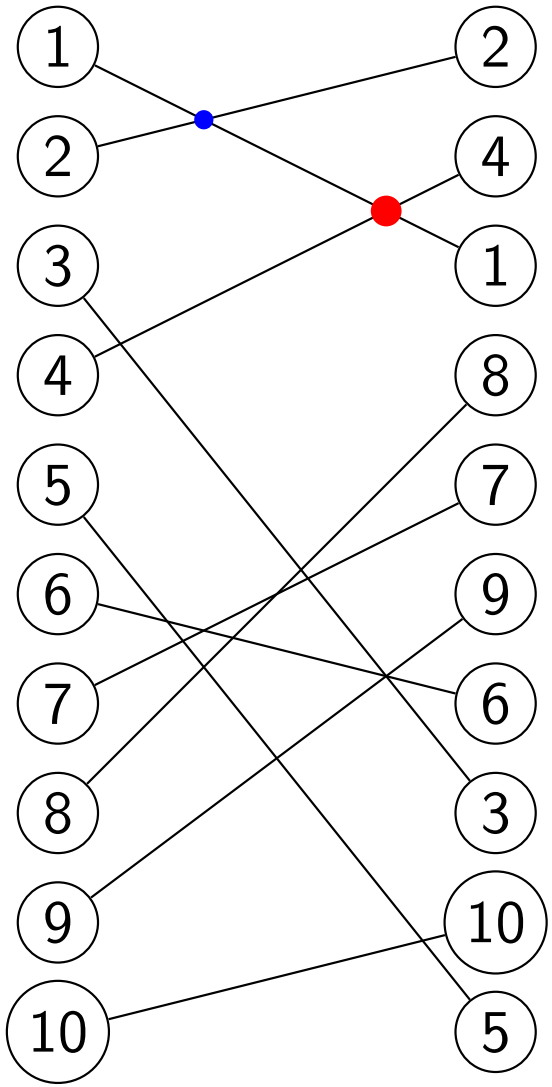



(a) Permutation

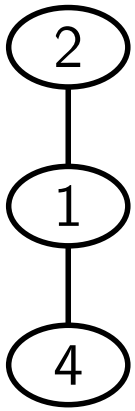



(b) Graphe de permutation

# Lien avec les graphes de permutations

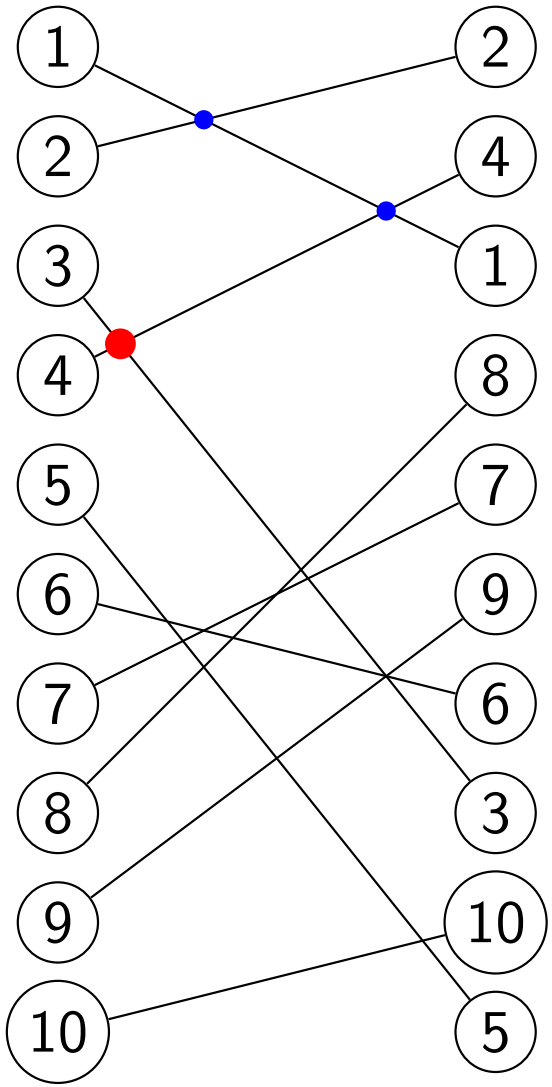



(a) Permutation

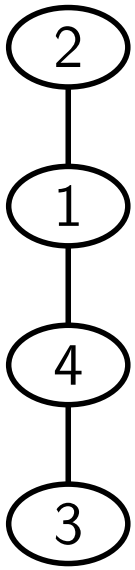



(b) Graphe de permutation

# Lien avec les graphes de permutations

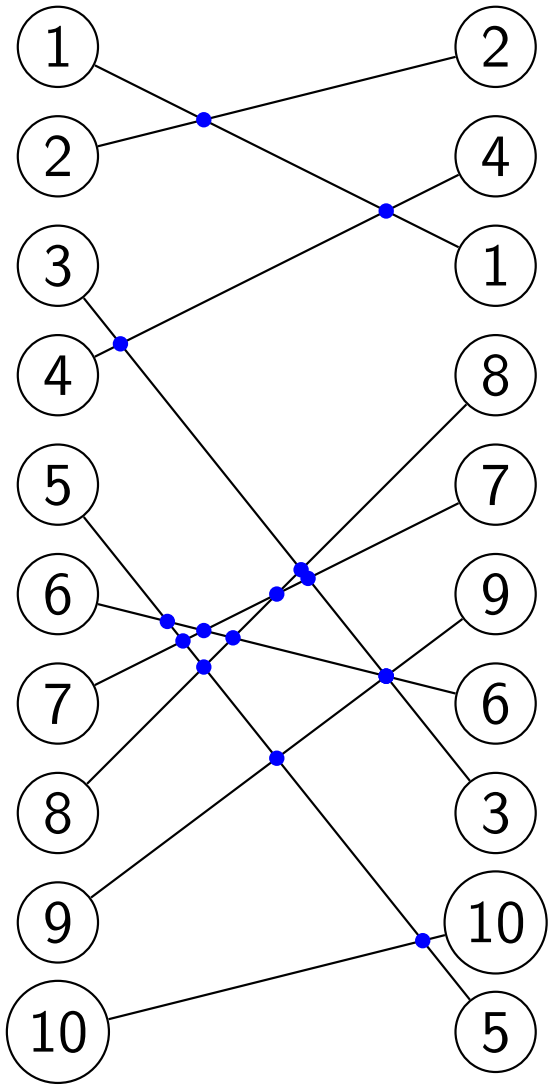



(a) Permutation

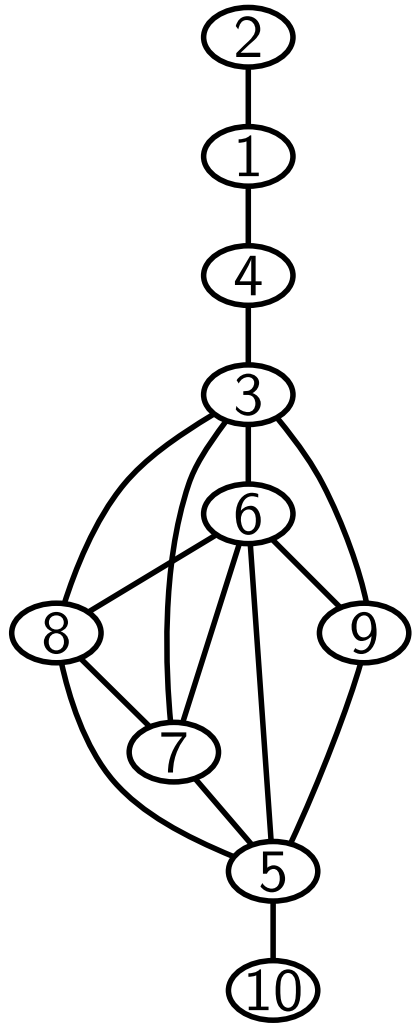



(b) Graphe de permutation

# Lien avec les graphes de permutations




(a) Permutation

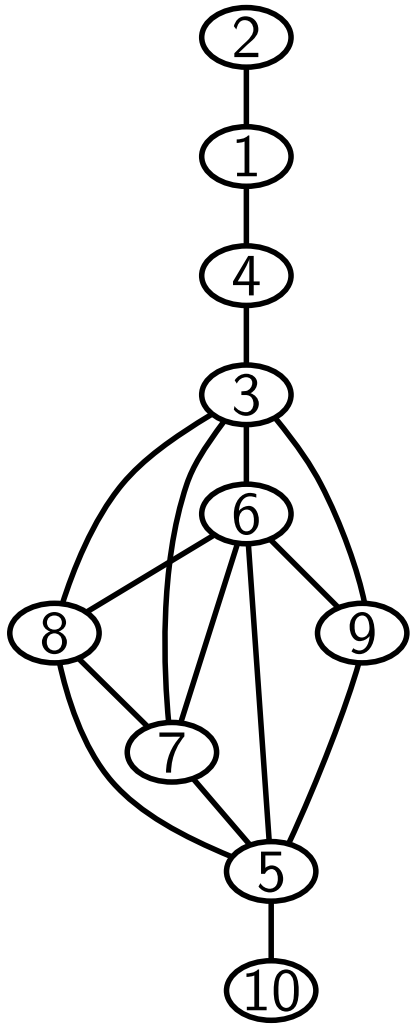



(b) Graphe de permutation

# Caractérisation

- Recherche des images de segments par une permutation
- image de segment :
  - 4
  - $4; 3; 6; 7; 8; 9; 5 \equiv 3; 4; \ldots; 9$
- non : $4; 6, \ldots$

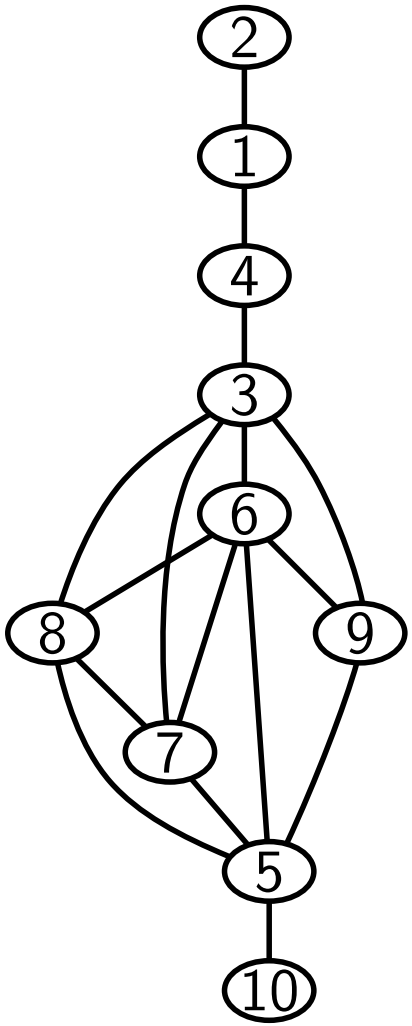
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $\sigma$ | 2 | 4 | 1 | 8 | 7 | 9 | 6 | 3 | 10 | 5 |



# Caractérisation

- Recherche des images de segments par une permutation
- image de segment :
  - 4
  - $4; 3; 6; 7; 8; 9; 5 \equiv 3; 4; \ldots; 9$
- non : $4; 6, \ldots$

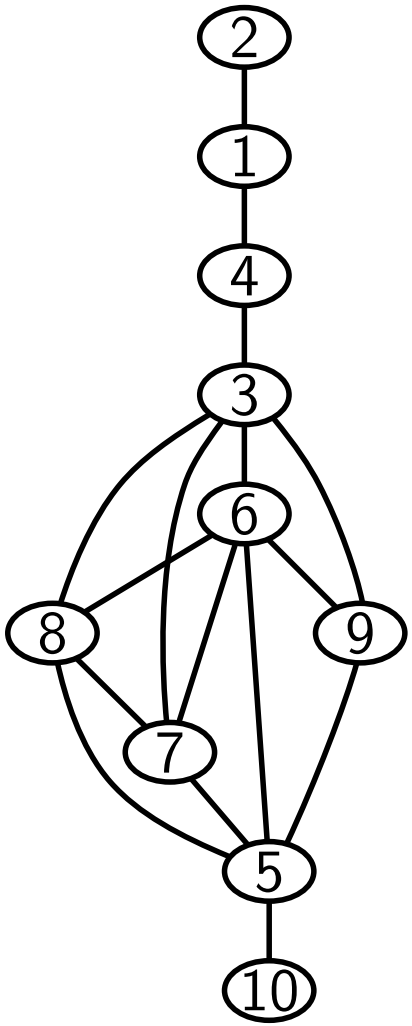
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $\sigma$ | 2 | 4 | 1 | 8 | 7 | 9 | 6 | 3 | 10 | 5 |



# Caractérisation

- Recherche des images de segments par une permutation
- image de segment :
  - 4
  - $4; 3; 6; 7; 8; 9; 5 \equiv 3; 4; \ldots; 9$
- non : $4; 6, \ldots$

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $\sigma$ | 2 | 4 | 1 | 8 | 7 | 9 | 6 | 3 | 10 | 5 |



# Plan

1. Décomposition modulaire de graphe modulaire
   - Permutation
   - Décomposition modulaire
   - Tracé du graphe de permutations
   - Particularité
2. Algorithme
   - Plus naïf...
   - Algorithme quadratique
   - Vers l'algorithme linéaire
   - L'algorithme linéaire
3. Démonstration
   - Coq
   - Version quadratique
   - Démonstration
4. Conclusion

# Algorithme très naïf

- pour les intervalles possibles $\mathcal{S}$ $[\mathcal{O}(n^2)]$ :
- recherche $m = \min \mathcal{S}$ et $M = \max \mathcal{S}$, $[\mathcal{O}(n)]$
- teste d'appartenance : $\forall i \in [\![m; M]\!], i \in \mathcal{S}$ $[\mathcal{O}(n \times n)]$
- $\rightarrow \mathcal{O}(n^5)$

**Caractérisation des intervalles**

$I \subset [\![1; n]\!]$ $I$ est un intervalle $\iff$ $\max I - \min I + 1 = |I|$

$I = \{4; 3; 6; 7; 8; 9; 5\}$ : $9 - 3 + 1 = 7$.

# Algorithme très naïf

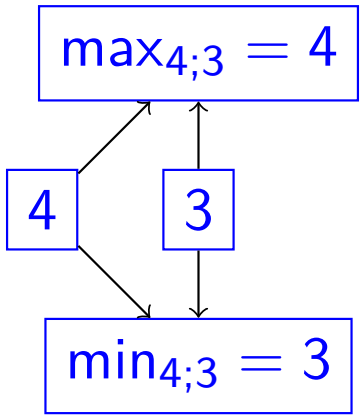
- pour les intervalles possibles $\mathcal{S}$ $[\mathcal{O}(n^2)]$ :
- recherche $m = \min \mathcal{S}$ et $M = \max \mathcal{S}$, $[\mathcal{O}(n)]$
- teste d'appartenance : $\forall i \in [\![m; M]\!], i \in \mathcal{S}$ $[\mathcal{O}(n \times n)]$
- $\rightarrow \mathcal{O}(n^5)$

**Caractérisation des intervalles**

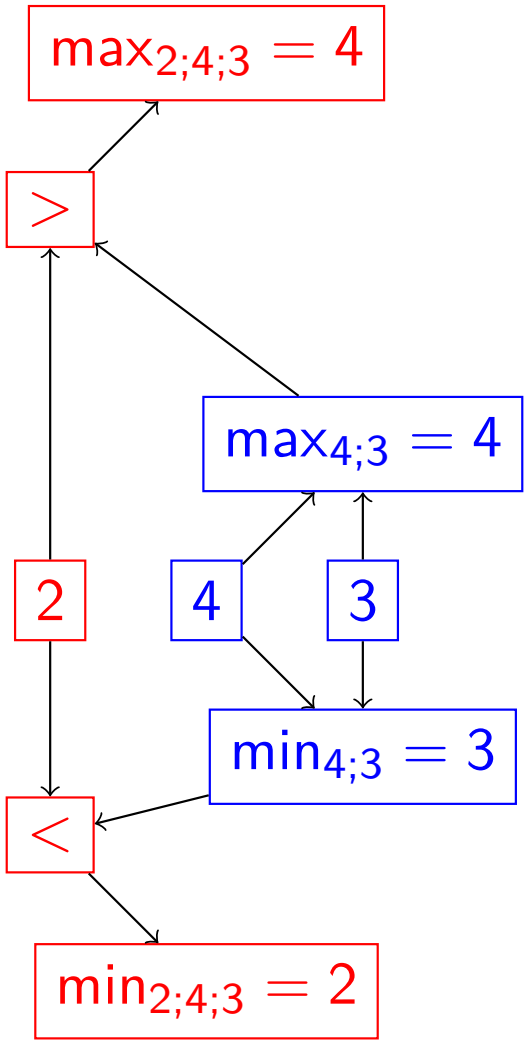
$I \subset [\![1; n]\!]$ $I$ est un intervalle $\iff$ $\max I - \min I + 1 = |I|$

$I = \{4; 3; 6; 7; 8; 9; 5\}$ : $9 - 3 + 1 = 7$.

# Algorithme quadratique



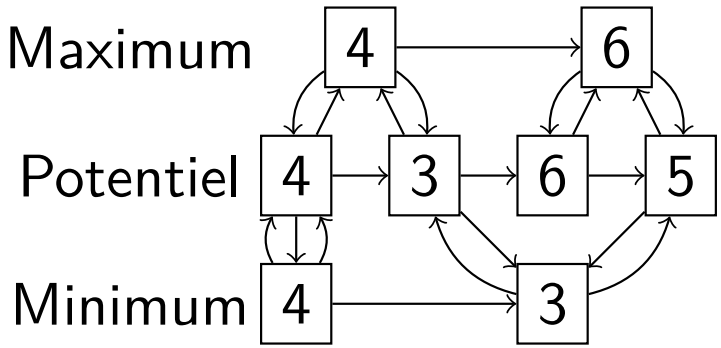
# Algorithme quadratique



# Intervalles communs

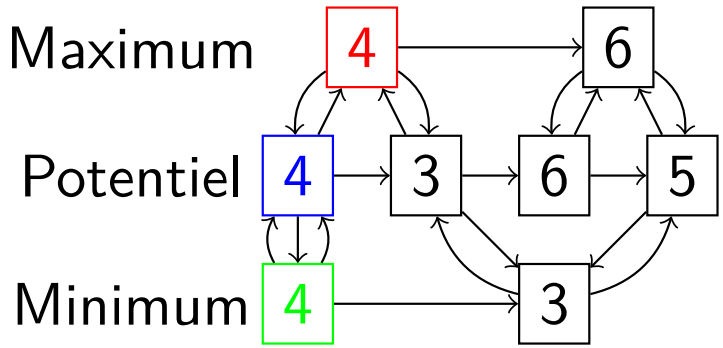
- Takeaki Uno and Mutsunori Yagiura
- *Algorithmica* $\mathcal{O}(n + K)$
- structure de donnée
- Binh-Minh Bui Xuan, Michel Habib, Christophe Paul
- $\mathcal{O}(n)$

# Intervalles communs




Found interval :

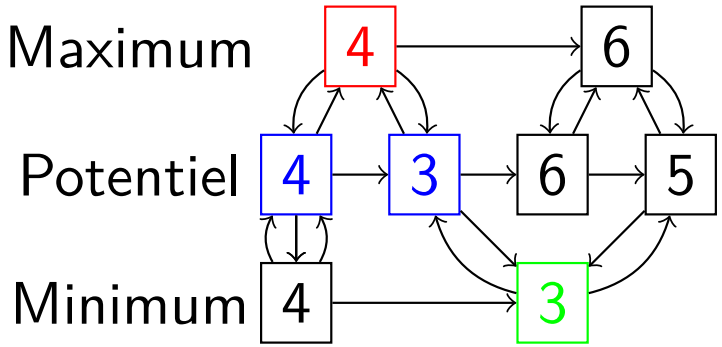
$\emptyset$




Found interval :

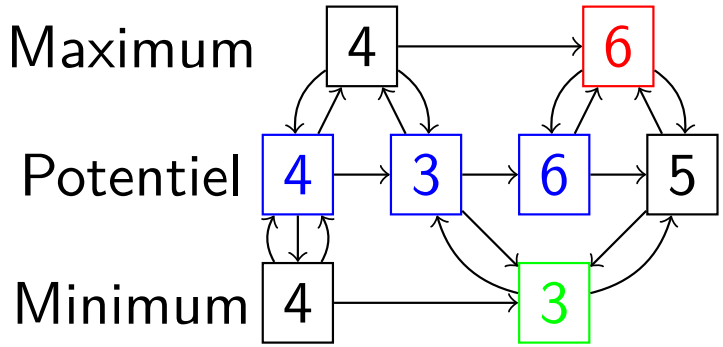
$\{\{4\}\}$

# Intervalles communs




Found interval :

$\{\{4\}; \{4; 3\}\}$




Found interval :

$\{\{4\}; \{4; 3\}\}$

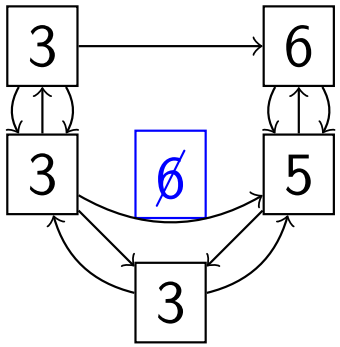
# Version linéaire

⏩ Passer l'algorithme linéaire

Maximum

Potentiel

Minimum

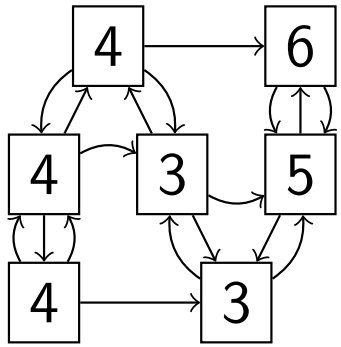



$\{5\}; \{6\}; \{6; 5\}; \{3\}$

Maximum

Potentiel

Minimum




$\{5\}; \{6\}; \{6; 5\}; \{4\}$
$\{4; \ldots; 3\}; \{4; \ldots; 5\}$

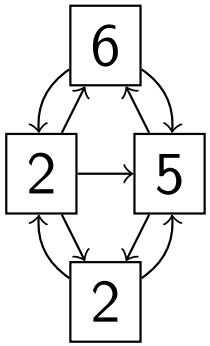
# Version linéaire

⏩ Passer l'algorithme linéaire

Maximum

Potentiel

Minimum

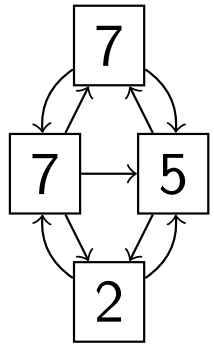



$\{5\}; \{6\}; \{6; 5\}; \{4\}$
$\{4 \ldots 3\}; \{4 \ldots 5\}; \{2 \ldots 6\}$

Maximum

Potentiel

Minimum




$\{5\}; \{6\}; \{6; 5\}; \{4\}$
$\{4 \ldots 3\}; \{4 \ldots 5\}; \{2 \ldots 6\}$
$\{7 \ldots 5\}$

# Plan

1. Décomposition modulaire de graphe modulaire
   - Permutation
   - Décomposition modulaire
   - Tracé du graphe de permutations
   - Particularité
2. Algorithme
   - Plus naïf...
   - Algorithme quadratique
   - Vers l'algorithme linéaire
   - L'algorithme linéaire
3. Démonstration
   - Coq
   - Version quadratique
   - Démonstration
4. Conclusion

# Pourquoi Coq ?

## Coq

1. Preuve formelle
2. Écrit en OCaml  100.000 lignes de codes
3. théorie des types :
   - Correspondance de Curry-Howard :

     démo $\iff \varphi$ : Théorèmes $\to$ Théorème
   - Calculus of (Inductive) Constructions or Coc

## Réussites

- compilateur CompCert C (X. Leroy)
- théorème des quatre couleurs (G. Gonthier)

Figure – Logo de Coq

# Pourquoi Coq ?

## Coq

1. Preuve formelle
2. Écrit en OCaml  100.000 lignes de codes
3. théorie des types :
   - Correspondance de Curry-Howard :

     démo $\iff \varphi$ : Théorèmes $\to$ Théorème
   - Calculus of (Inductive) Constructions or Coc

## Réussites

- compilateur CompCert C (X. Leroy)
- théorème des quatre couleurs (G. Gonthier)

Figure – Logo de Coq

# Pourquoi Coq ?

## Coq

1. Preuve formelle
2. Écrit en OCaml  100.000 lignes de codes
3. théorie des types :
   - Correspondance de Curry-Howard :

     $$\text{démo} \iff \varphi : \text{Théorèmes} \to \text{Théorème}$$

   - Calculus of (Inductive) Constructions or Coc

## Réussites

- compilateur CompCert C (X. Leroy)
- théorème des quatre couleurs (G. Gonthier)

Figure – Logo de Coq

# Pourquoi Coq ?

## Coq

1. Preuve formelle
2. Écrit en OCaml  100.000 lignes de codes
3. théorie des types :
   - Correspondance de Curry-Howard :

$$\text{démo} \iff \varphi : \text{Théorèmes} \to \text{Théorème}$$

   - Calculus of (Inductive) Constructions or Coc

## Réussites

- compilateur CompCert C (X. Leroy)
- théorème des quatre couleurs (G. Gonthier)

Figure – Logo de Coq

# Pourquoi Coq ?

## Coq

1. Preuve formelle
2. Écrit en OCaml  100.000 lignes de codes
3. théorie des types :
   - Correspondance de Curry-Howard :

$$\text{démo} \iff \varphi : \text{Théorèmes} \to \text{Théorème}$$

   - Calculus of (Inductive) Constructions or Coc

## Réussites

- compilateur CompCert C (X. Leroy)
- théorème des quatre couleurs (G. Gonthier)

Figure – Logo de Coq

# Pourquoi Coq ?

## Coq

1. Preuve formelle
2. Écrit en OCaml  100.000 lignes de codes
3. théorie des types :
   - Correspondance de Curry-Howard :

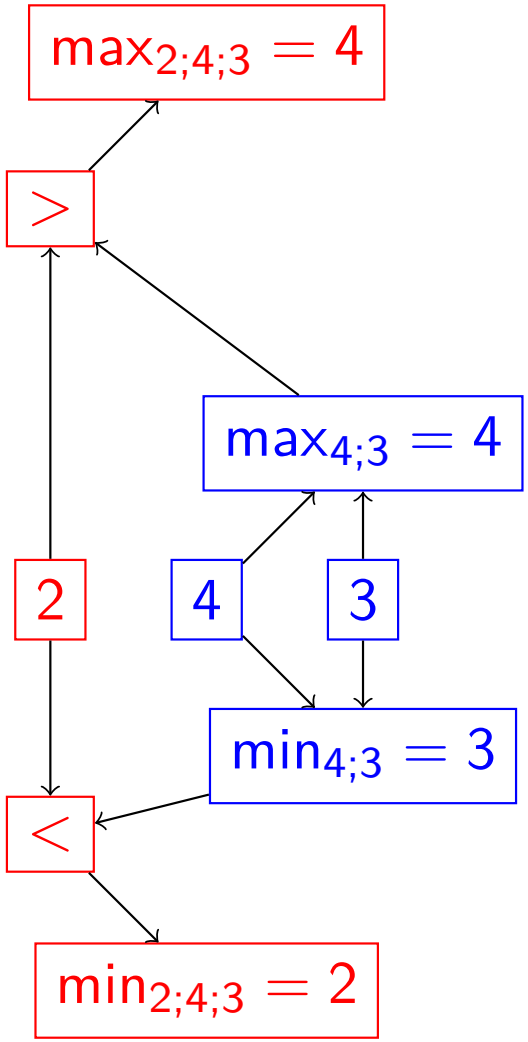
$$\text{démo} \iff \varphi : \text{Théorèmes} \to \text{Théorème}$$

   - Calculus of (Inductive) Constructions or Coc

## Réussites

- compilateur CompCert C (X. Leroy)
- théorème des quatre couleurs (G. Gonthier)

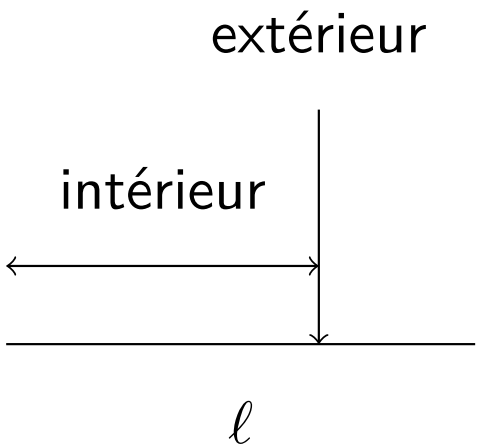
Figure – Logo de Coq




- liste
- des valeurs (maximum et minimum)

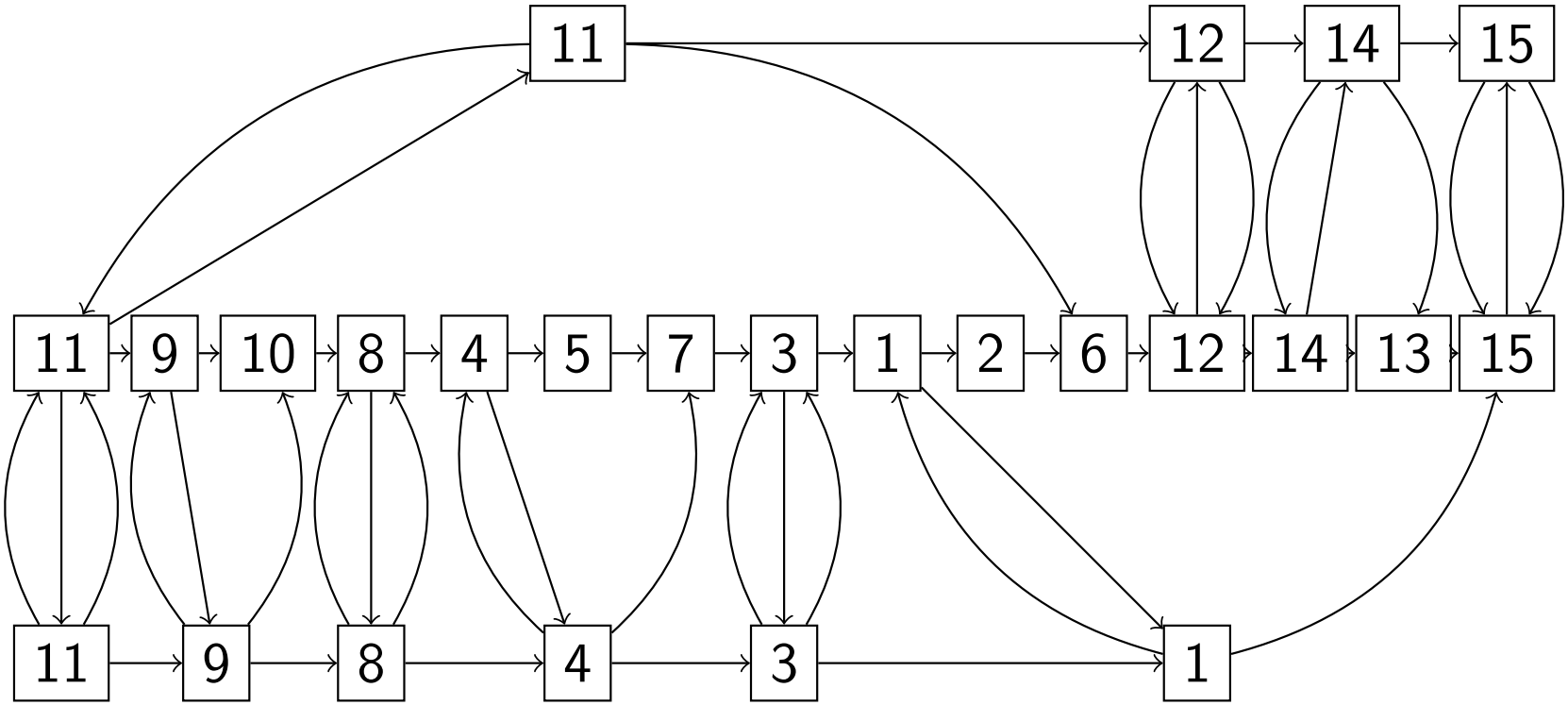
## Programmation

- boucle « intérieur »
- boucle extérieur




**Theorem** outer_l_n2_proof :
∀ (ℓ : list ℕ) (δ : ℕ),
outer_loop_n2 ℓ
δ = outer_loop ℓ δ.

# Démonstration




- liste : simple
- pointeurs : compliqué

# Difficultés de la preuve

## Pointeurs

### Numérotation des listes

```
Definition Numbered_List (A : Type) : Type :=
    | nil_num : Numbered_List A
    | L_num : A → ℕ → Numbered_List A →
Numbered_List A.
```

## Terminaison des programmes

```
Fixpoint f (n : ℕ ) : ?
 := f (S n)
```

# Plan

1. Décomposition modulaire de graphe modulaire
   - Permutation
   - Décomposition modulaire
   - Tracé du graphe de permutations
   - Particularité
2. Algorithme
   - Plus naïf...
   - Algorithme quadratique
   - Vers l'algorithme linéaire
   - L'algorithme linéaire
3. Démonstration
   - Coq
   - Version quadratique
   - Démonstration
4. Conclusion

# Conclusion

1. Stage :
   - travail d'algorithmique intéressant
   - manque de temps pour finir la preuve
2. Présentation
   - preuve formelle de plus en plus utilisé
   - outil complexe, pas un miracle

Questions ?

- Aller au maintien de la structure
- Aller à application génétique
- Aller à Compcert
- Aller au théorème des quatre couleurs
- Exemple de code Coq

# Plan

5 Annexe
- Lien avec l'ordre des gènes
- CompCert
- Théorème des quatre couleurs
- Exemple de code Coq

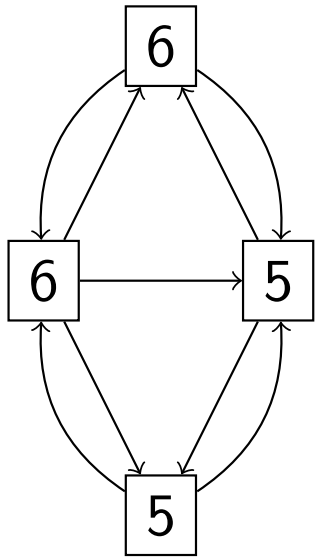
# Structure de donnée, exemple quadratique

⏩ Passer maintien de la structure

Maximum

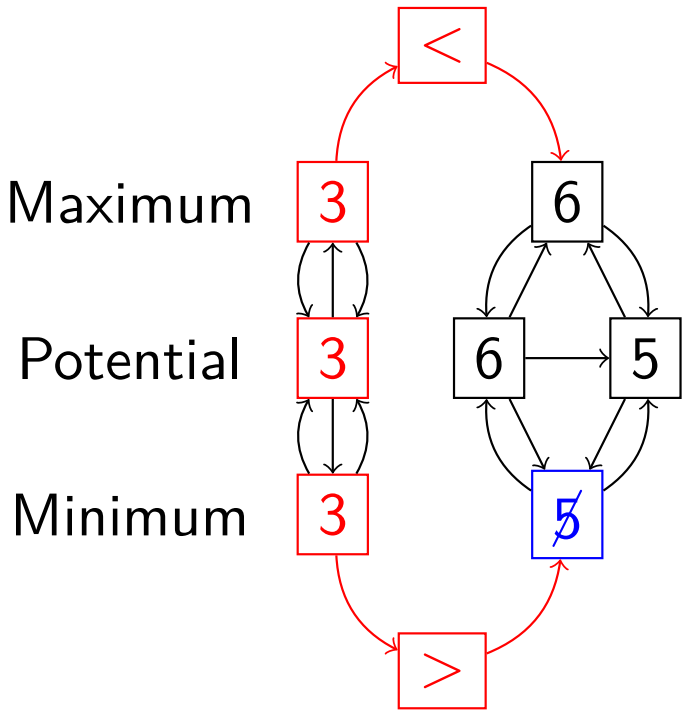
Potential

Minimum



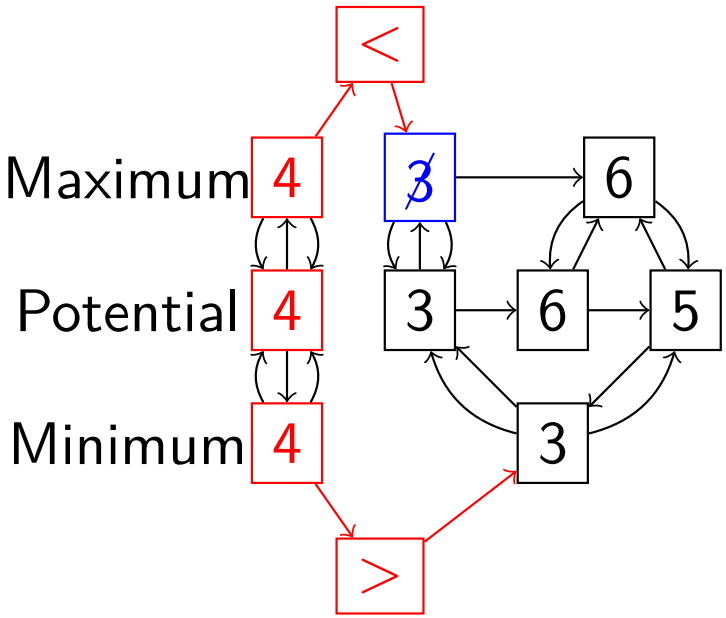
# Structure de donnée, exemple quadratique

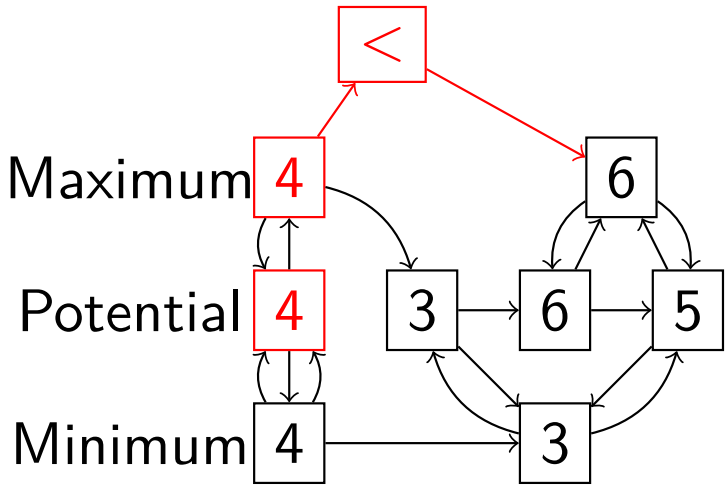
⏩ Passer maintien de la structure

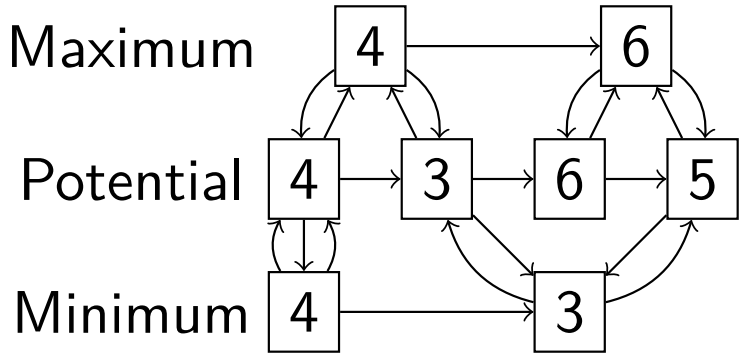


# Structure de donnée, exemple quadratique

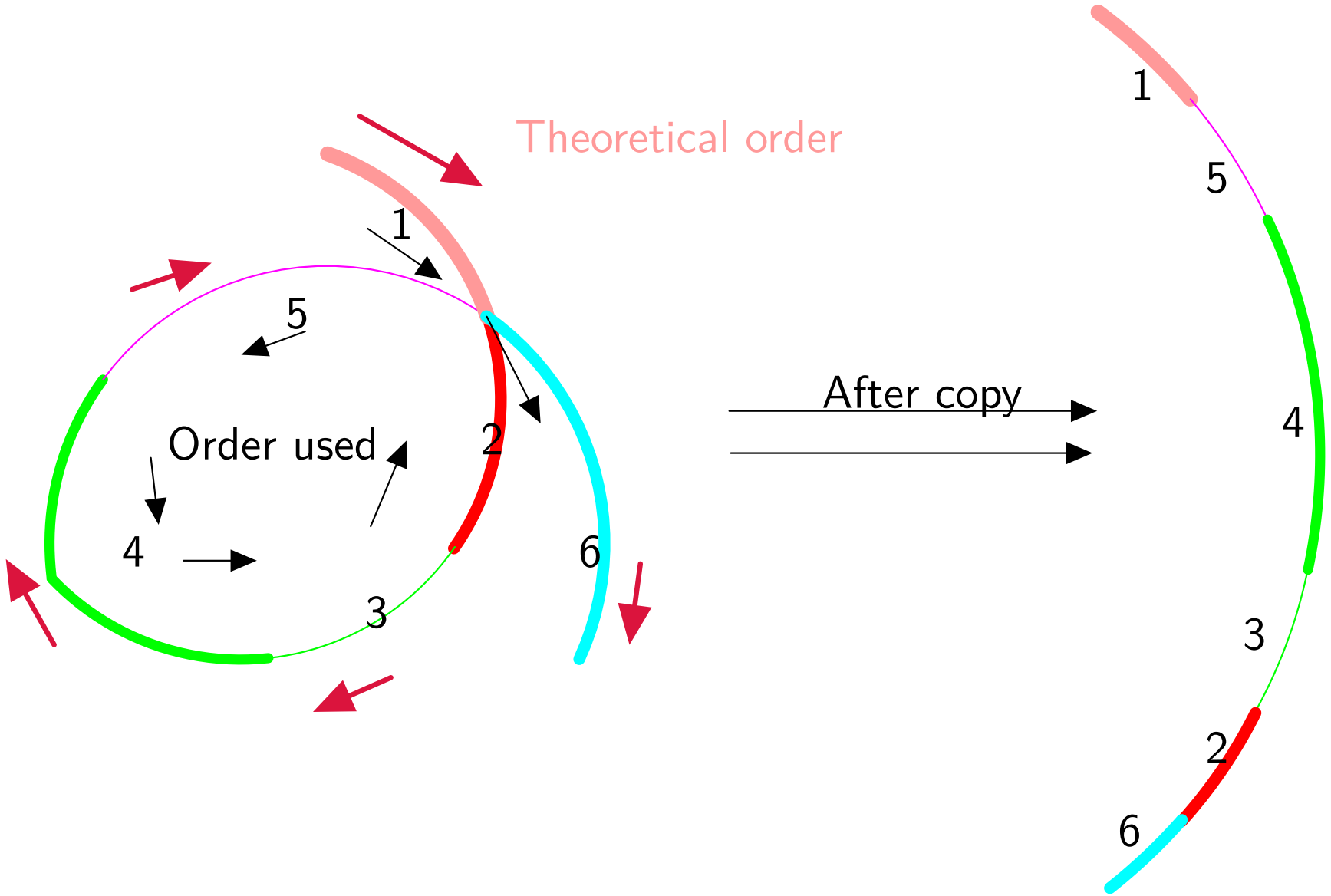
⏩ Passer maintien de la structure

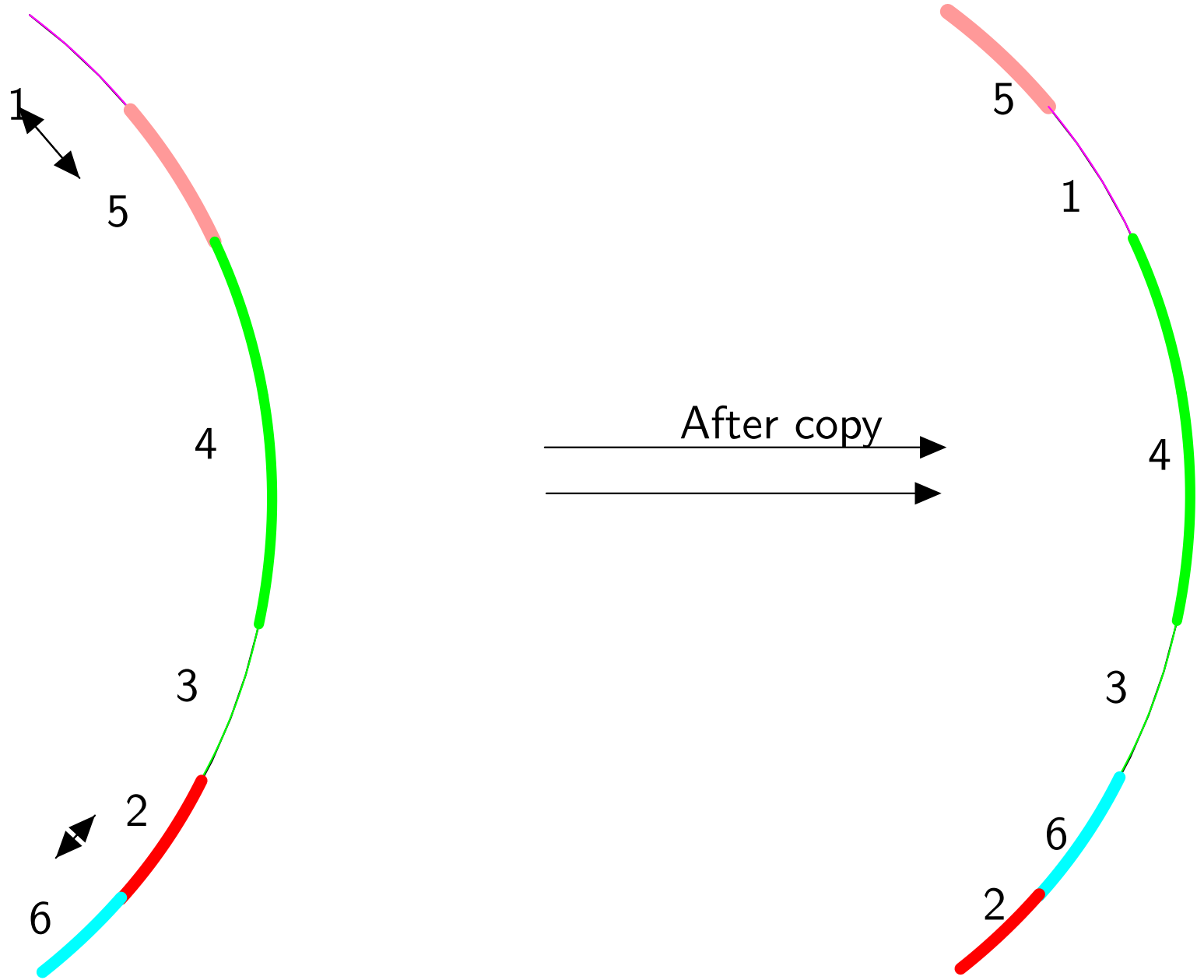


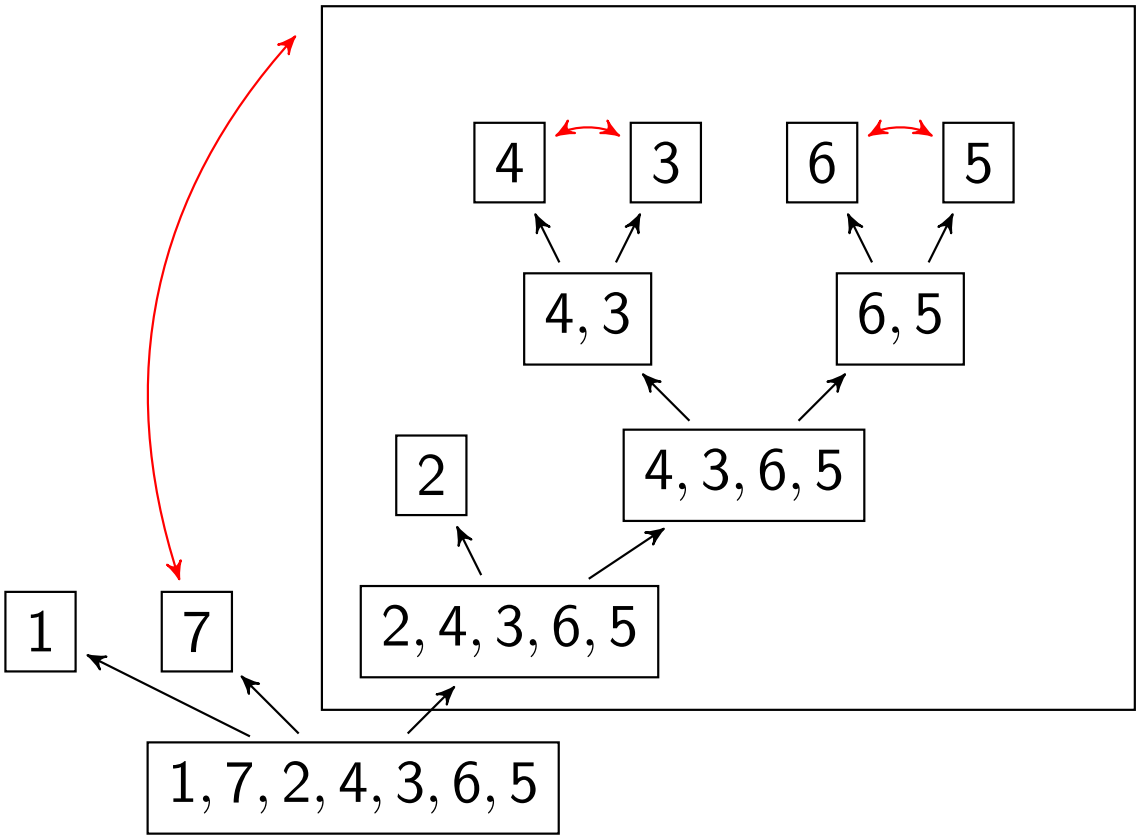



Genetic Adam et Eva


Theoretical order
1
5
2
6
3
4
Order used
After copy
1
5
4
3
2
6


1
5
4
3
2
6
After copy
5
1
4
3
6
2


4
3
6
5
4, 3
6, 5
2
4, 3, 6, 5
1
7
2, 4, 3, 6, 5
1, 7, 2, 4, 3, 6, 5

▸ Liste annexe

## description

- Compilateur prouvé à 90%

## Semantic preservation theorem :

For all source programs S and compiler-generated code C, if the compiler, applied to the source S, produces the code C, without reporting a compile-time error, then the observable behavior of C is one of the possible observable behaviors of S.

## Pourquoi ?

- compilateur compliqué
- beaucoup de bugs
- négligeable par rapport aux bugs de programmation
- sauf si applications critiques

## Pourquoi pas ?

- comportement indéfini : *Undefined behavior*
- 32 bits
- librairie standard non prouvée

# CompCert

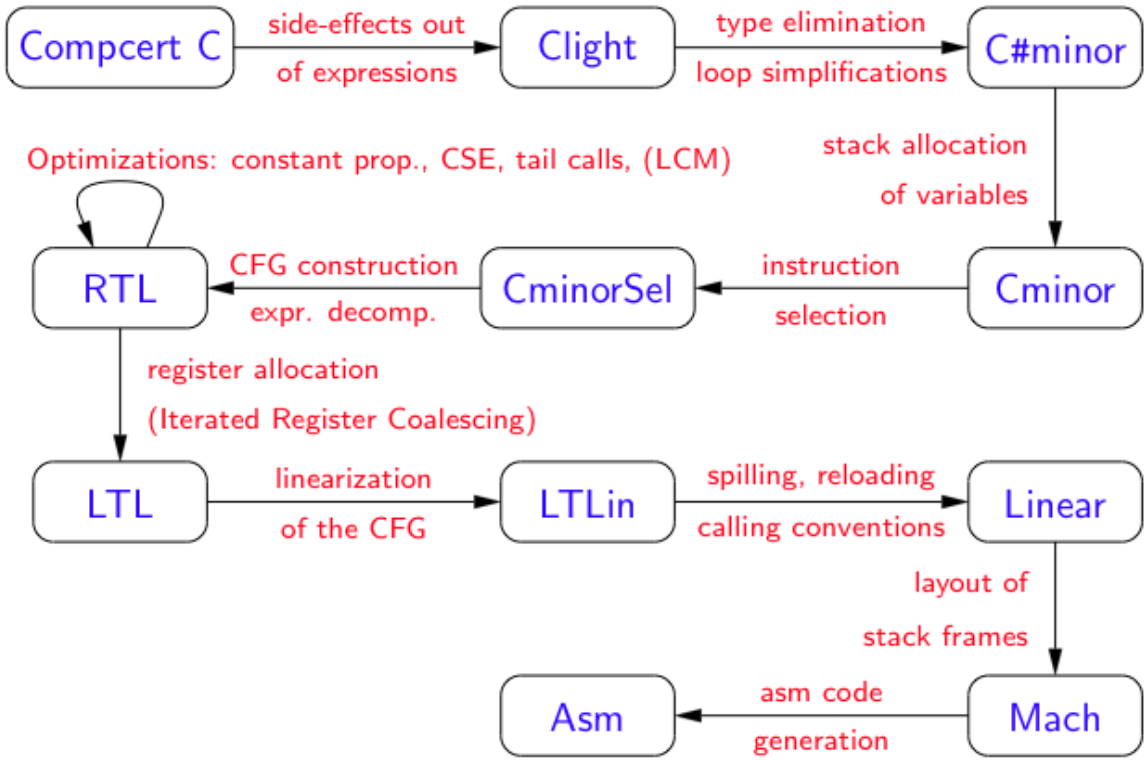



FIGURE – Étape de compilation (site internet de CompCert C)

▸ Liste annexe

# Théorème des 4 couleurs

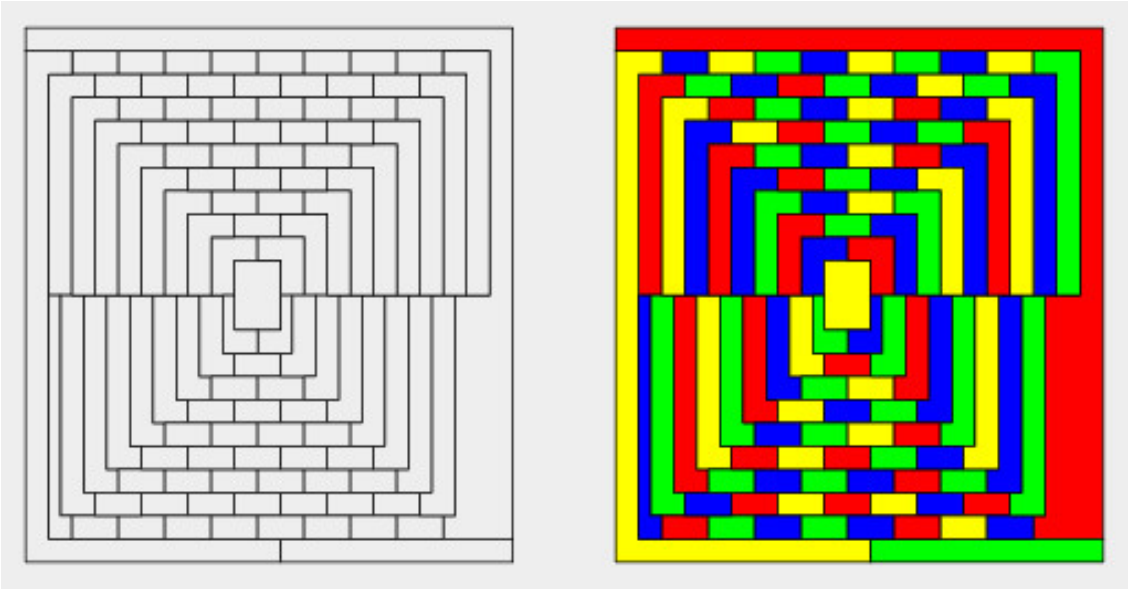


FIGURE – Martin Gardner (1975) Poisson d'avril (MathWorld)

- Guthrie 1852
- de Morgan a présenté le problème
- Appel et Haken 1977
- Gonthier 2004

▸ Liste annexe

**Theorem** outer_l_pr :
∀ (ℓ : list ℕ)
(δ : ℕ),
outer_l_n2 ℓ δ
= outer_l ℓ δ.

1 subgoals, subgoal 1 (ID 34)

---

∀ (ℓ : list ℕ)
(δ : ℕ),
outer_l_n2 ℓ δ
= outer_l ℓ δ

```
Theorem outer_l_pr :
∀ (ℓ : list ℕ)
    (δ : ℕ),

outer_l_n2 ℓ δ
  = outer_l ℓ δ.
Proof.
```

```
1 subgoals, subgoal 1 (ID 34)
──────────────────────────────
∀ (ℓ : list ℕ)
    (δ : ℕ),
  outer_l_n2 ℓ δ
= outer_l ℓ δ
```

```
Theorem outer_l_pr :
∀ (ℓ : list ℕ)
    (δ : ℕ),

outer_l_n2 ℓ δ
  = outer_l ℓ δ.
Proof.
 intros ℓ δ.
```

```
1 subgoals, subgoal 1 (ID 36)

ℓ : list ℕ
δ : ℕ
──────────────────────────────
outer_l_n2 ℓ δ
 = outer_l ℓ δ
```

```
Theorem outer_l_pr :
∀ (ℓ : list ℕ)
      (δ : ℕ),

outer_l_n2 ℓ δ
    = outer_l ℓ δ.
 Proof.
  intros ℓ δ.
  induction δ.
```

```
2 subgoals, subgoal 1 (ID 39)

ℓ : list ℕ
──────────────────────────────────────
outer_loop_n2 ℓ 0
  = outer_loop ℓ 0

subgoal 2 (ID 42) is:
outer_l_n2 ℓ (S δ)
 = outer_l ℓ (S δ)
```

```
Theorem outer_l_pr :
∀ (ℓ : list ℕ)
    (δ : ℕ),

outer_l_n2 ℓ δ
  = outer_l ℓ δ.
Proof.
  intros ℓ δ.
  induction δ.
  simpl.
```

```
2 subgoals, subgoal 1 (ID 39)

ℓ : list ℕ
──────────────────────────────
[] = []

subgoal 2 (ID 42) is:
outer_l_n2 ℓ (S δ)
 = outer_l ℓ (S δ)
```

```
Theorem outer_l_pr :
∀ (ℓ : list ℕ)
    (δ : ℕ),

outer_l_n2 ℓ δ
  = outer_l ℓ δ.
Proof.
  intros ℓ δ.
  induction δ.
  simpl.
  reflexivity.
```

```
1 subgoals, subgoal 1 (ID 42)

ℓ : list ℕ
δ : ℕ
IHδ : outer_l_n2 ℓ δ
  = outer_l ℓ δ
============================
outer_l_n2 ℓ (S δ)
= outer_l ℓ (S δ)
```

```
Theorem outer_l_pr :
∀ (ℓ : list ℕ)
    (δ : ℕ),

outer_l_n2 ℓ δ
  = outer_l ℓ δ.
Proof.
 intros ℓ δ.
 induction δ.
 simpl.
 reflexivity.

 simpl.
```

```
1 subgoals, subgoal 1 (ID 48)

ℓ : list ℕ
δ : ℕ
IHδ : outer_l_n2 ℓ δ
  = outer_l ℓ δ
============================
inner_l_n2 ℓ (S δ)
  ++ outer_l_n2 ℓ δ =
inner_l ℓ (S δ)
  ++ outer_l ℓ δ
```

```
Theorem outer_l_pr:
∀ (ℓ : list ℕ)
      (δ:ℕ),

outer_l_n2 ℓ δ
    = outer_l ℓ δ.
Proof.
 intros ℓ δ.
 induction δ.
 simpl.
 reflexivity.

 simpl.
 apply list_eq_cons.
```

```
2 subgoals, subgoal 1 (ID 49)

ℓ : list ℕ
δ : ℕ
IHδ : outer_l_n2 ℓ δ
  = outer_l ℓ δ
──────────────────────────────
inner_l_n2 ℓ (S δ)
  = inner_l ℓ (S δ)

subgoal 2 (ID 50) is:
 outer_l_n2 ℓ δ
  = outer_l ℓ δ
```

```
Theorem outer_l_pr:
∀ (ℓ : list ℕ)
      (δ:ℕ),

outer_l_n2 ℓ δ
    = outer_l ℓ δ.
Proof.
 intros ℓ δ.
 induction δ.
 simpl.
 reflexivity.

 simpl.
 apply list_eq_cons.
 apply inner_l_n2_p.
```

```
1 subgoals, subgoal 1 (ID 50)

ℓ : list ℕ
δ : ℕ
IHδ : outer_l_n2 ℓ δ
 = outer_l ℓ δ
──────────────────────────────
outer_l_n2 ℓ δ
 = outer_l ℓ δ
```

```
Theorem outer_l_pr :
∀ (ℓ : list ℕ)
      (δ : ℕ),

outer_l_n2 ℓ δ
    = outer_l ℓ δ.
Proof.
 intros ℓ δ.
 induction δ.
 simpl.
 reflexivity.

 simpl.
 apply list_eq_cons.
 apply inner_l_n2_p.
 assumption.
Qed.
```