Projet de synthèse (semestre 3)

Implémentation des algorithmes pour trouver toutes les intersections d'un ensemble de segments en 2D

Giorgio Lucarelli

giorgio.lucarelli@univ-lorraine.fr

octobre 2024

UNIVERSITÉ DE LORRAINE | UFR MATHÉMATIQUES INFORMATIQUE MÉCANIQUE ET AUTOMATIQUE

Objectif

- Utiliser les connaissances acquises pendant la L1 et la L2 en
  - Maths Discrètes
  - Programmation C
  - Algorithmique et Structures des Données
  - Récursivité
  - Outils Système

Organisation

- travailler en équipes de **2 personnes**
- **les membres de chaque équipe appartiennent au même groupe TP**
- **déclaration des équipes :** à la fin du cours ou par émail (giorgio.lucarelli@univ-lorraine.fr) jusqu'à dimanche 13 octobre minuit
- sujet sur arche

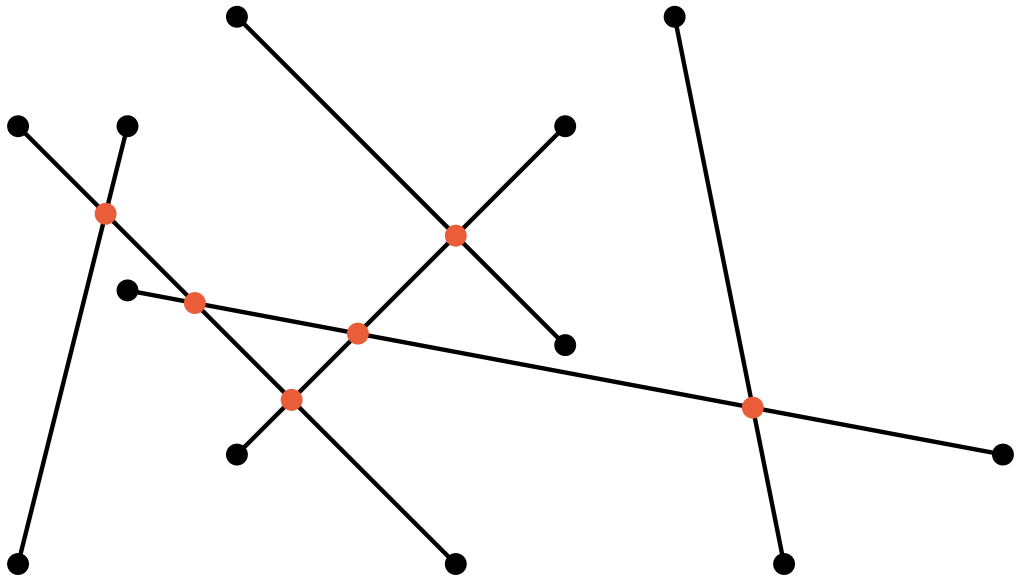
Emploi du temps

- 4h CM : aujourd'hui
  - présentation du projet et des outils
- 18h TP
  - 1ère séance : introduction sur GIT
  - séances 2–9 : suivi du projet
  - intervenants : Bochra DJAHEL (TP3, TP4, TP5), Giorgio LUCARELLI (TP1, TP2)
- 4h TP + 4h TD : soutenances
  - chaque équipe présentera son projet pendant 40 minutes
  - vous devrez être présents que pendant votre présentation

Sujet : trouver tous les points d'intersection d'un ensemble des segments sur le plan

Entrée : un ensemble de segments $S = \{s_1, s_2, \ldots, s_n\}$

Sortie : un ensemble $P = \{p_1, p_2, \ldots, p_k\}$ avec les points d'intersection entre les segments de S

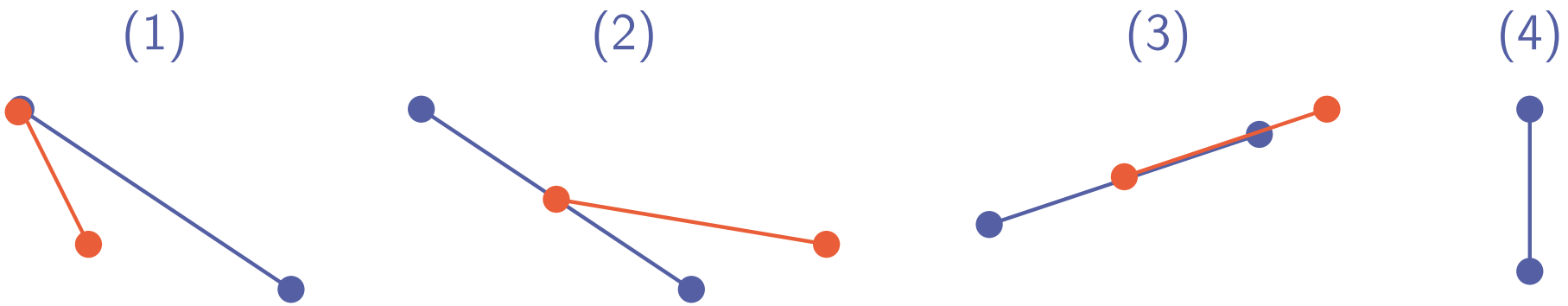


Définitions

- Point = (abscisse, ordonnée)
- Segment = {extrémité 1, extrémité 2} = {début, fin}

Hypothèses

- Configurations **non** étudiées :
  (on suppose que notre entrée ne contient pas ces configurations)
  (1) Deux segments ne peuvent pas partager une même extrémité.
  (2) Le point d'intersection de deux segments ne peut pas coïncider avec l'extrémité d'un de ces segments.
  (3) L'intersection de deux segments ne peut être un (sous-)segment ; situation qui arrive quand les deux segments sont colinéaires et partiellement recouvrants.
  (4) Un segment ne peut pas être parallèle à l'axe des ordonnées.



Algorithmes à implémenter

1. Algorithme glouton
2. Algorithme de balayage
   - proposé par Bentley et Ottmann en 1979

Algorithme glouton

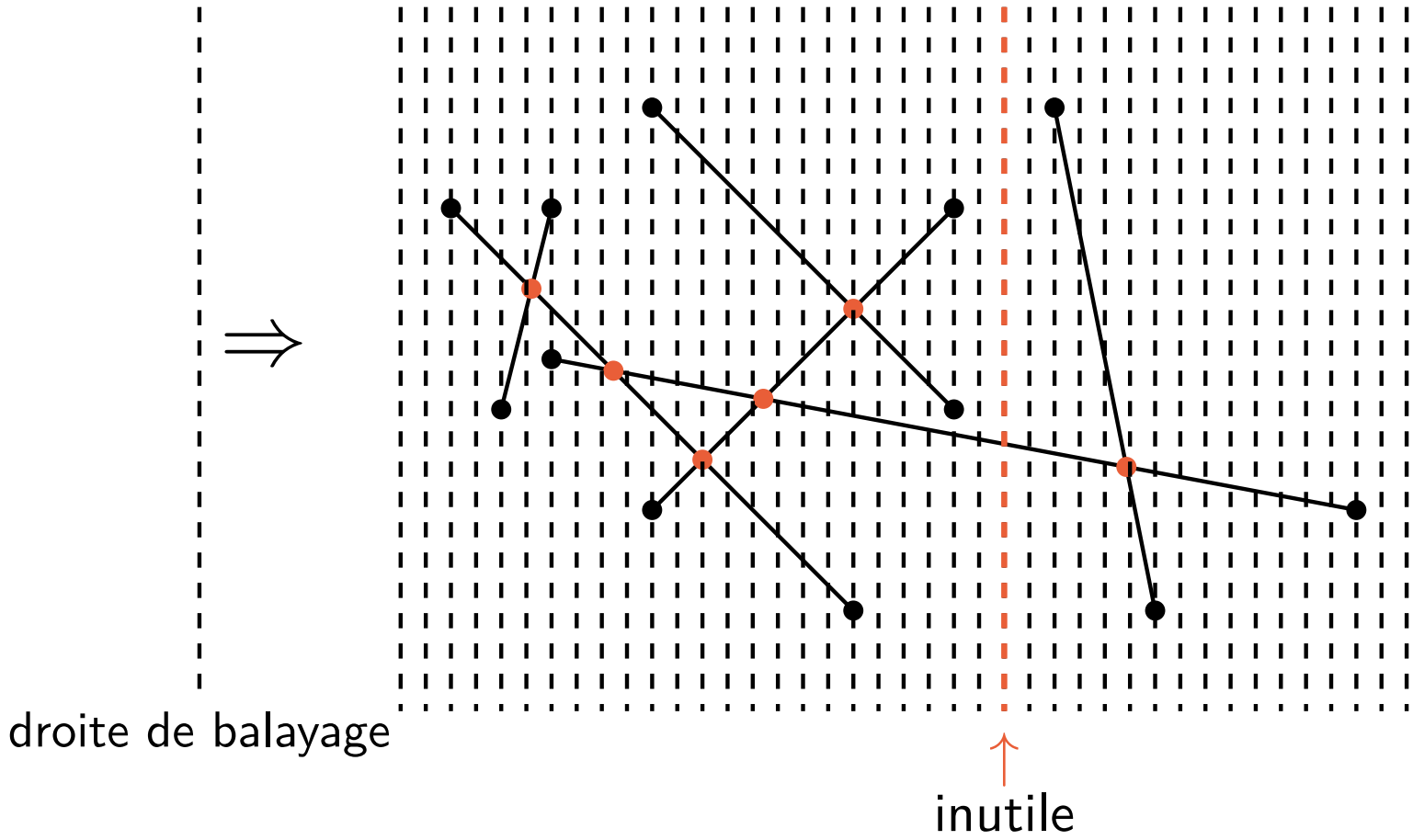
1: Initialisation : $P \leftarrow \emptyset$
2: **pour** chaque couple s_i, s_j de segments de S, avec $s_i \neq s_j$ **faire**
3: **si** s_i intersecte avec s_j **alors**
4: $p \leftarrow$ point d'intersection entre s_i et s_j
5: $P \leftarrow P \cup \{p\}$
6: **retourne** P

- Algorithme exhaustif et très simple
- **Complexité ?**

Algorithme de balayage (en anglais : sweep-line algorithm)

droite de balayage

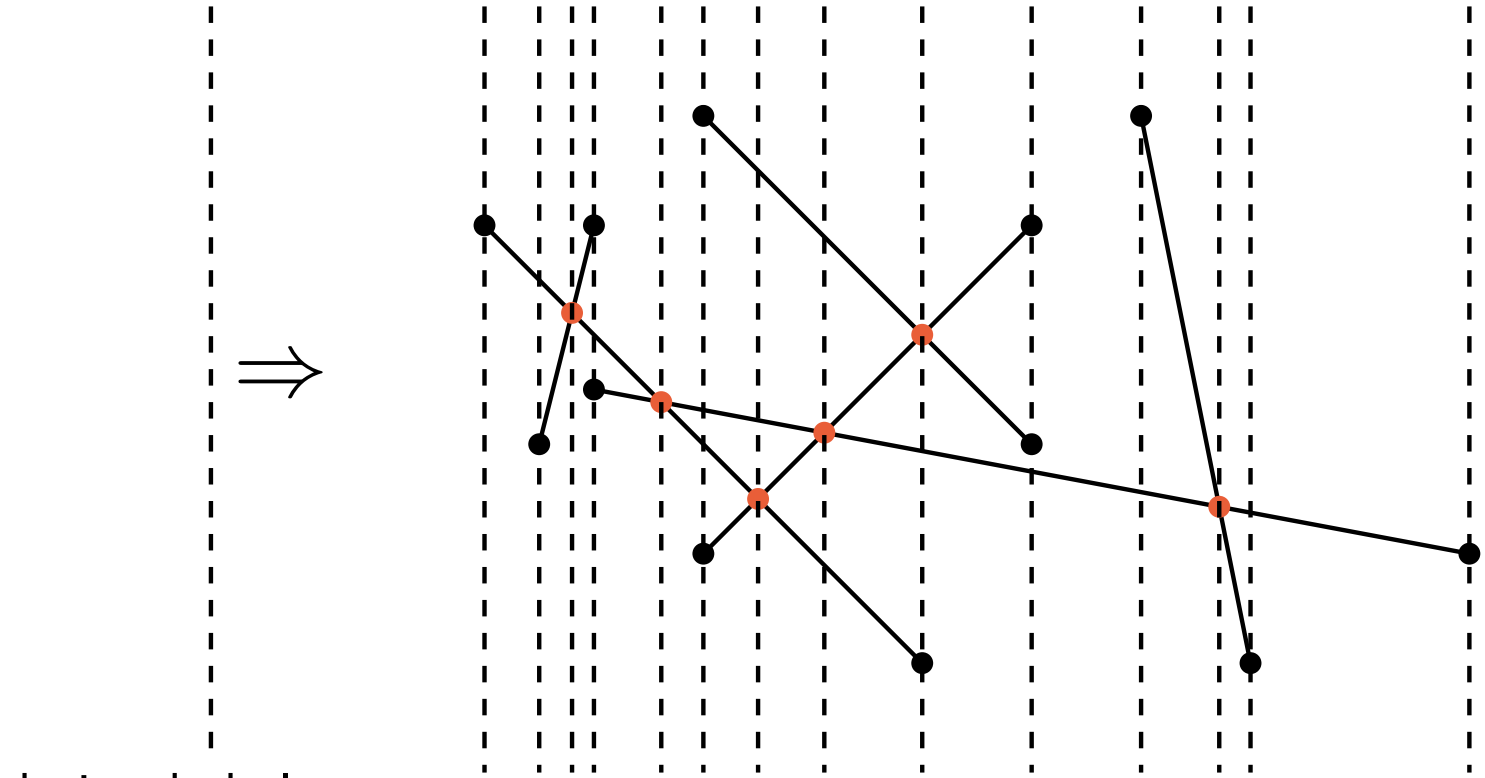
- droite parallèle à l'axe des ordonnées
- se déplace de gauche à droit



Algorithme de balayage (en anglais : sweep-line algorithm)

droite de balayage

- droite parallèle à l'axe des ordonnées
- se déplace de gauche à droit

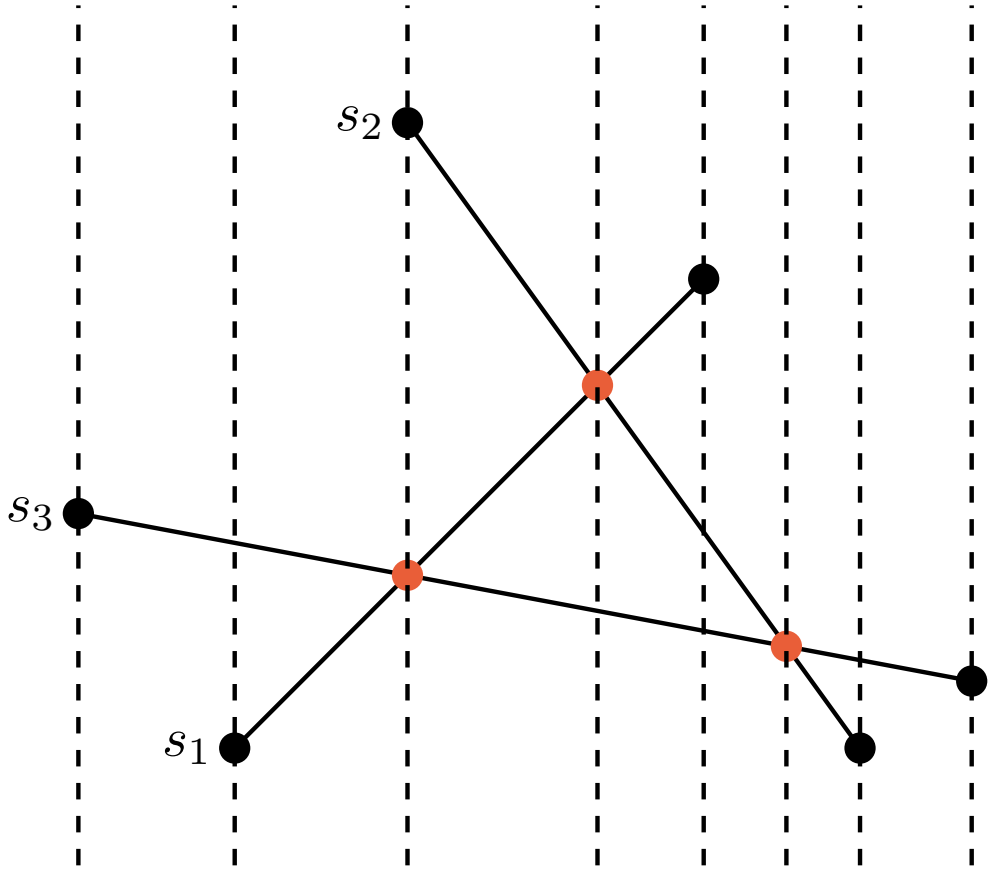


Événements

- événement $=$ point
- 3 **types** d'événements
  - début d'un segment
  - fin d'un segment
  - point d'intersection
- **priorité** entre les événements (**ordre** d'événements)
  - événement $e_1 = (x_1, y_1)$
  - événement $e_2 = (x_2, y_2)$
  - e_1 a plus grande priorité que e_2 (e_1 précède e_2 : $e_1 \prec e_2$) si
    1. $x_1 < x_2$ ou
    2. $x_1 = x_2$ et $y_1 > y_2$

Ordre des événements : exemple

$$\text{début}(s_3) \prec \text{début}(s_1) \prec \text{début}(s_2) \prec \text{intersection}(s_3,s_1) \prec \text{intersection}(s_2,s_1) \prec \text{fin}(s_1) \prec \text{intersection}(s_2,s_3) \prec \text{fin}(s_2) \prec \text{fin}(s_3)$$



Algorithme de balayage : description générale (version 1)

1: **pour** chaque événement e suivant l'ordre spécifié **faire**
2: **si** e est du type "début segment" **alors**
3: `gestion_événement_début()`
4: **sinon si** e est du type "fin segment" **alors**
5: `gestion_événement_fin()`
6: **sinon si** e est du type "intersection" **alors**
7: `gestion_événement_intersection()`

Problème : comment trouver l'ensemble d'événements ?

- début d'un segment, fin d'un segment : font partie de l'entrée
- points d'intersection : font partie de la sortie ! ! !

Comment trouver l'ensemble d'événements ?

- **Initialisation** de la structure d'événements
  - ajouter les événements "début d'un segment" et "fin d'un segment"
- Détermination des événements du type "point d'intersection"
  - **dynamiquement** pendant l'exécution de l'algorithme
  - à l'intérieur des procédures `gestion_événement_début()`, `gestion_événement_fin()` et `gestion_événement_intersection()`
- **Structure de données :** file de priorité
  - dynamique
  - ordonnée

Algorithme de balayage : description générale (version 2)

1: `file_de_priorité events` $\leftarrow$ `initialisation(S)`
   // Initialisation de la file de priorité `events` avec les événements
   // du type "début segment" et "fin segment"
2: **tant que** `events` != $\emptyset$ **faire**
3: $e \leftarrow$ extraire l'événement avec la plus grande priorité dans `events`
4: **si** e est du type "début segment" **alors**
5: `gestion_événement_début()`
6: **sinon si** e est du type "fin segment" **alors**
7: `gestion_événement_fin()`
8: **sinon si** e est du type "intersection" **alors**
9: `gestion_événement_intersection()`

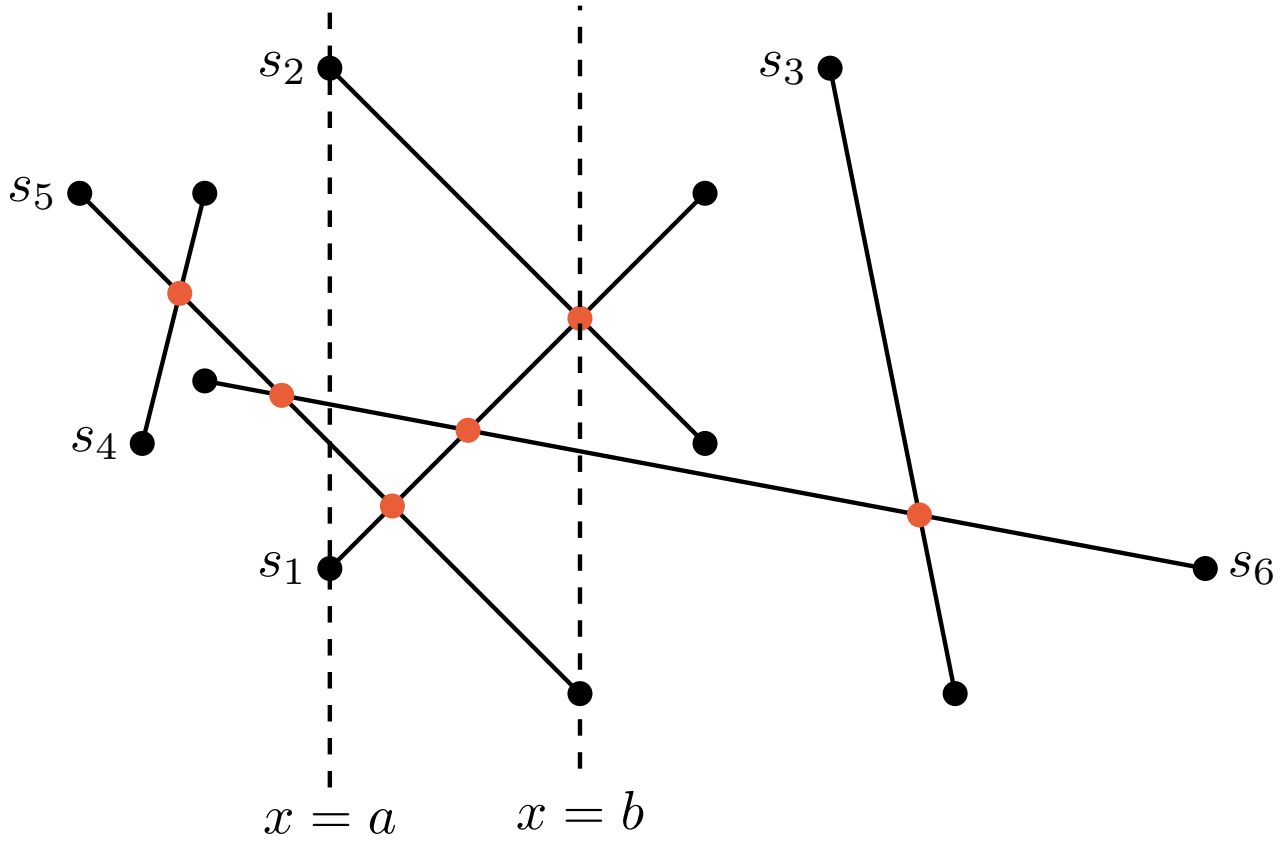
Question : Comment/quand déterminer un événement "intersection" ?

Droite de balayage

- un segment s est appelé **actif** par rapport à une droite de balayage si la droite intersecte le segment
- **état de la droite de balayage à une position x :**
  l'ensemble des segments actifs **triés** par ordre d'ordonnée à x décroissant
- **convention #1 :** si la fin d'un segment s survient à la position x, alors s n'appartient pas à l'état de la droite de balayage à x
- **convention #2 :** si un point d'intersection entre les segments s_i et s_j survient à la position x, alors l'ordre entre s_i et s_j est le même comme pour la position $x + \epsilon$, pour un $\epsilon > 0$ très petit

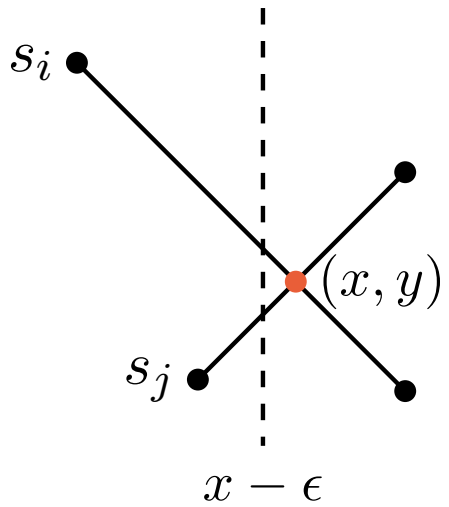
Droite de balayage

- **exemples**
  - $x = a$: état $= \langle s_2, s_6, s_5, s_1 \rangle$
  - $x = b$: état $= \langle s_1, s_2, s_6 \rangle$



Comment/quand déterminer un événement “intersection” ?

- **Observation :** si deux segments s_i, s_j intersectent à (x, y), alors s_i et s_j sont consécutifs à l'état de la droite de balayage à la position $x - \epsilon$, pour un $\epsilon > 0$ très petit.



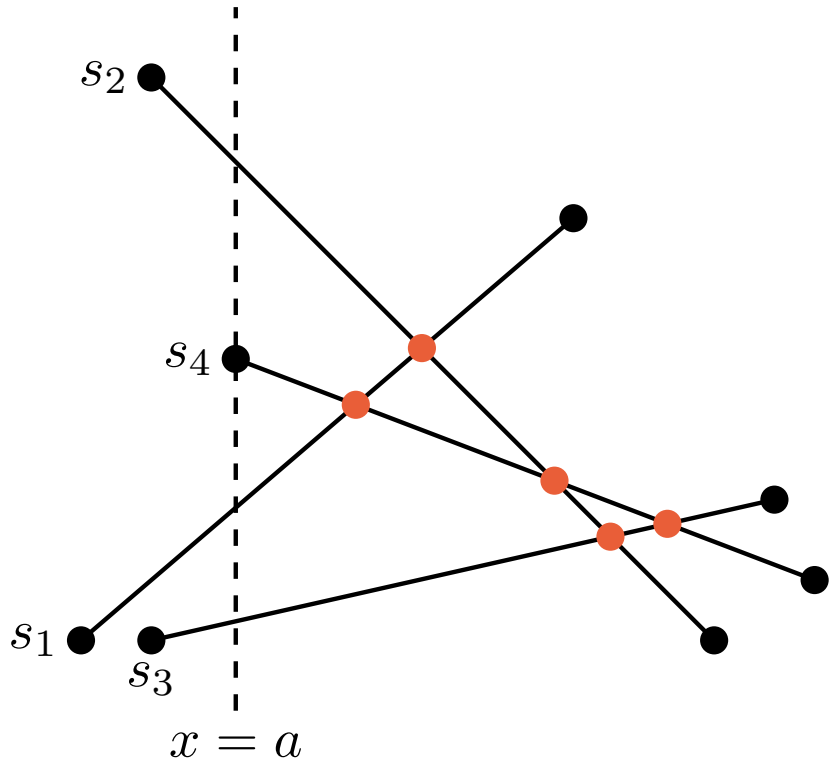
- **Observation (reformulée) :** si deux segments s_i, s_j intersectent à l'événement e, alors il y a un événement $e' \prec e$ pour lequel s_i et s_j sont consécutifs à l'état de la droite de balayage à e'.

Comment/quand déterminer un événement “intersection” ?

Il suffit d’examiner **à chaque événement** tous les couples des **segments consécutifs** dans l’état de la droite de balayage

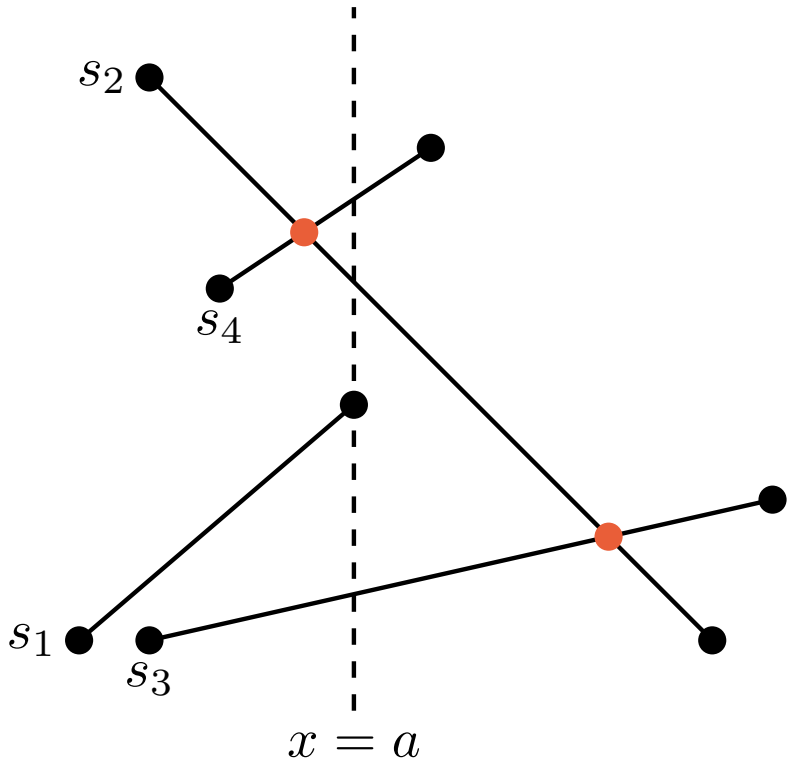
mais on peut faire mieux ...

- **événement "début d'un segment"**
  - état juste avant $x = a$: $\langle s_2, s_1, s_3 \rangle$
  - état à $x = a$: $\langle s_2, \mathbf{s_4}, s_1, s_3 \rangle$
  - examiner l'existence des points d'intersection **uniquement** entre les segments qui deviennent consécutifs après l'addition du nouveau segment dans l'état de la droite de balayage : s_2, s_4 et s_4, s_1



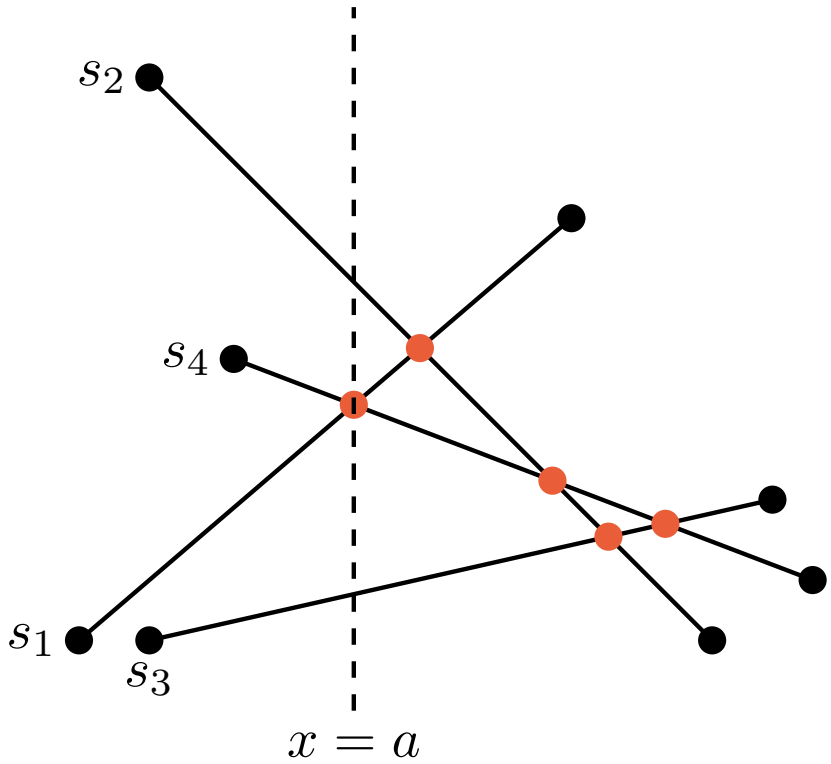
mais on peut faire mieux ...

- **événement "fin d'un segment"**
  - état juste avant $x = a$: $\langle s_4, s_2, \mathbf{s_1}, s_3 \rangle$
  - état à $x = a$: $\langle s_4, s_2, s_3 \rangle$
  - examiner l'existence des points d'intersection **uniquement** entre les segments qui deviennent consécutifs après la suppression du segment de l'état de la droite de balayage : s_2, s_3



mais on peut faire mieux ...

- **événement "intersection"**
  - état juste avant $x = a$: $\langle s_2, \mathbf{s_4}, \mathbf{s_1}, s_3 \rangle$
  - état à $x = a$: $\langle s_2, \mathbf{s_1}, \mathbf{s_4}, s_3 \rangle$
  - examiner pour des points d'intersection **uniquement** entre les segments qui deviennent consécutifs après l'échange des segments dans l'état de la droite de balayage : s_2, s_1 et s_4, s_3

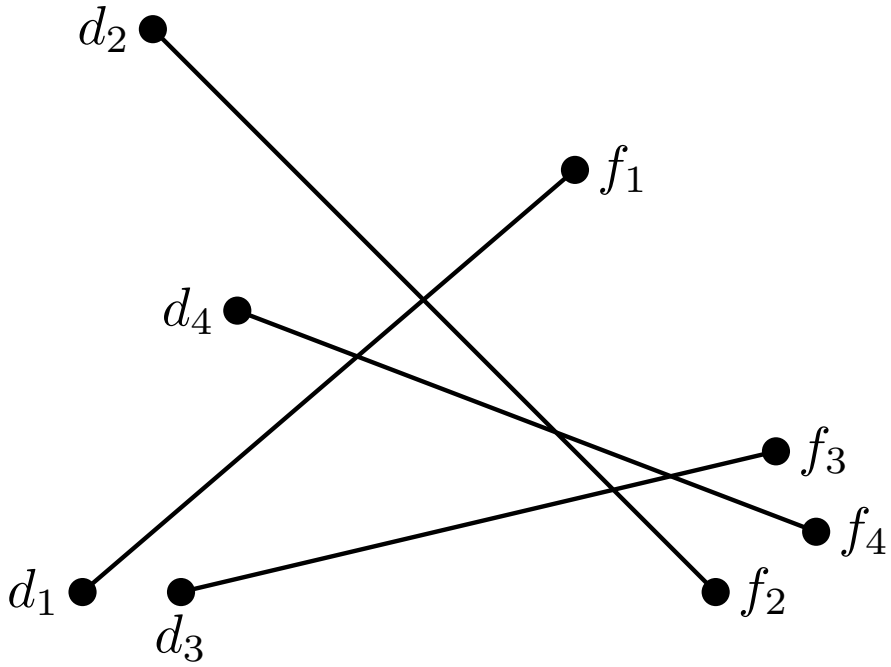


mais on peut faire mieux ...

Conclusion : il suffit d'examiner l'existence des points d'intersection **uniquement** entre les segments qui deviennent consécutifs après la modification de l'état de la droite de balayage

Pourquoi on a fait tout ça ?
Complexité du test : constante

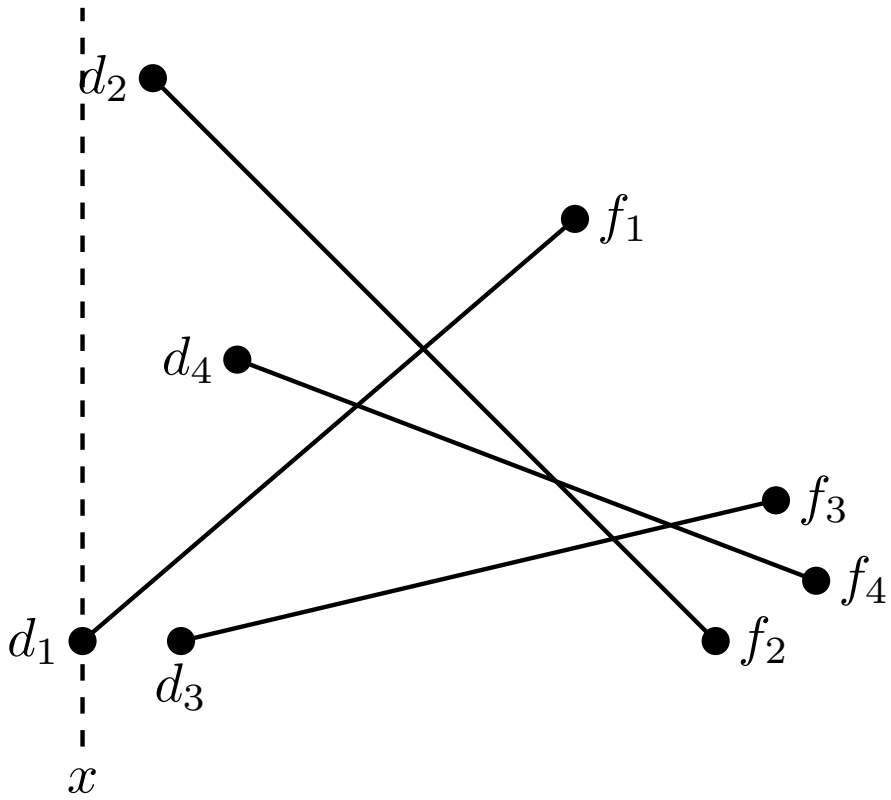
Exemple d'exécution



Initialisation

- file de priorité : $d_1 \prec d_2 \prec d_3 \prec d_4 \prec f_1 \prec f_2 \prec f_3 \prec f_4$
- état $= \langle\rangle$
- intersections trouvées : $\{\}$

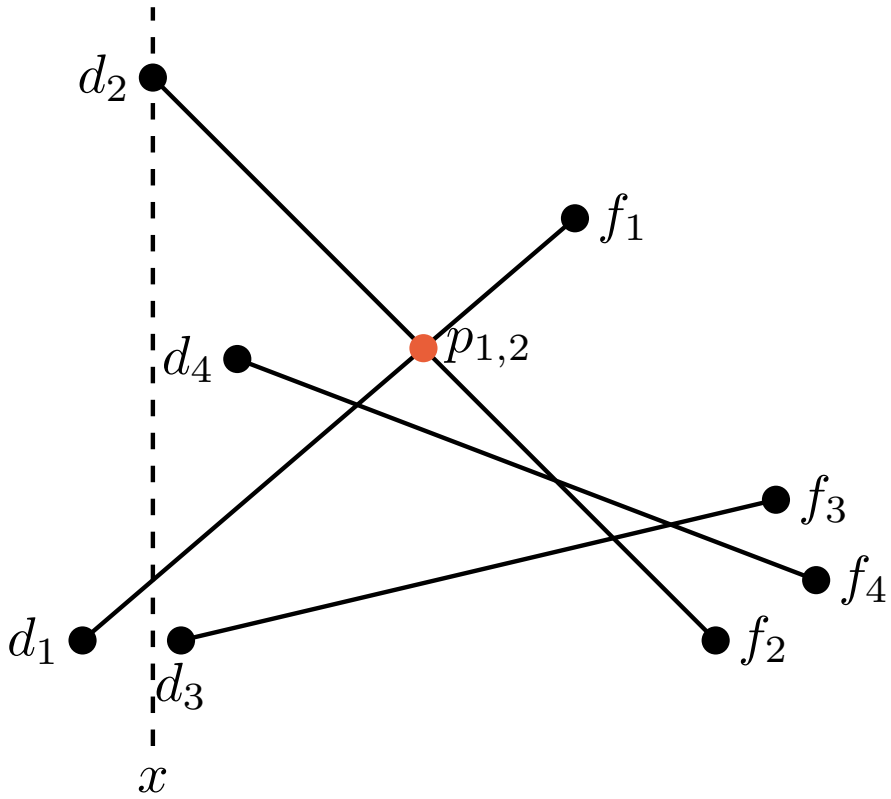
Exemple d'exécution



$x = d_1$

- file de priorité : $d_2 \prec d_3 \prec d_4 \prec f_1 \prec f_2 \prec f_3 \prec f_4$
- état $= \langle s_1 \rangle$
- intersections trouvées : $\{\}$

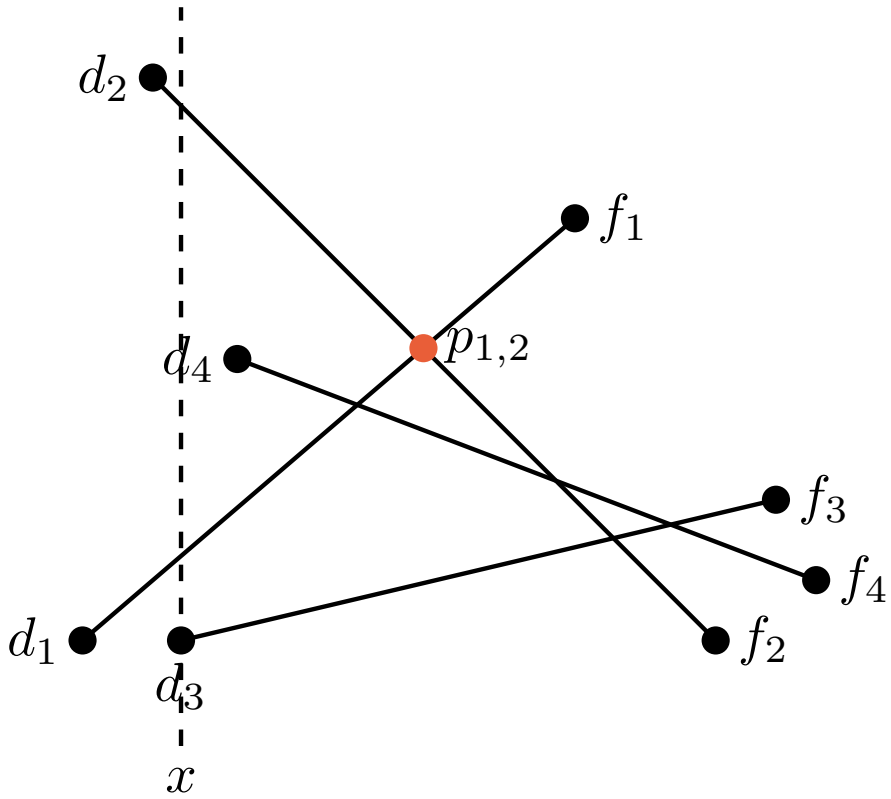
Exemple d'exécution



$x = d_2$

- file de priorité : $d_3 \prec d_4 \prec p_{1,2} \prec f_1 \prec f_2 \prec f_3 \prec f_4$
- état $= \langle s_2, s_1 \rangle$
- intersections trouvées : $\{p_{1,2}\}$

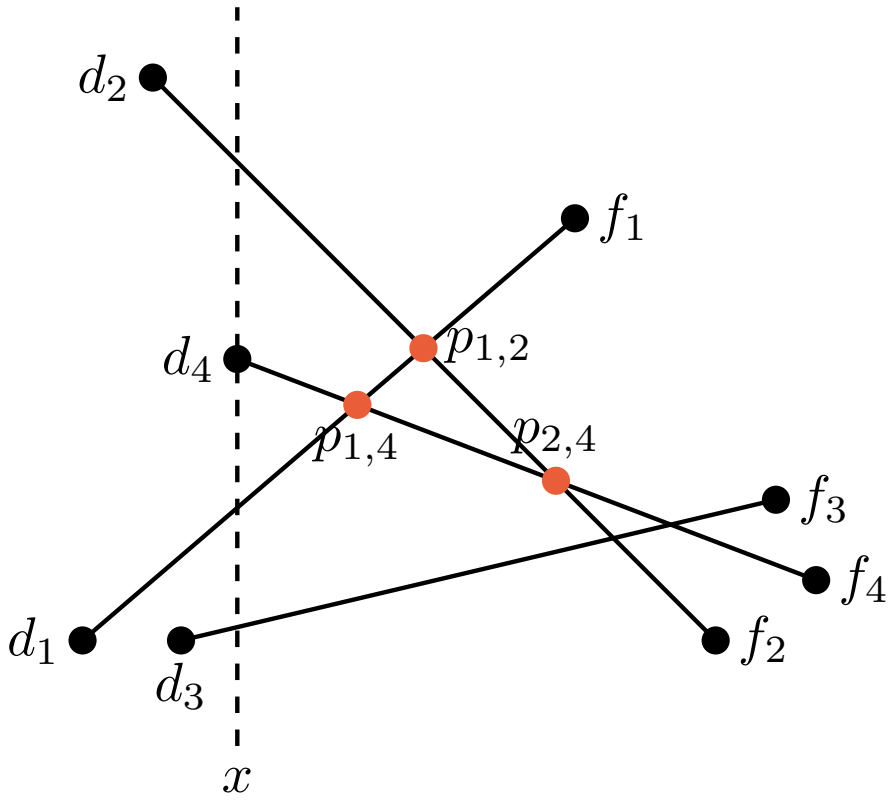
Exemple d'exécution



$x = d_3$

- file de priorité : $d_4 \prec p_{1,2} \prec f_1 \prec f_2 \prec f_3 \prec f_4$
- état $= \langle s_2, s_1, s_3 \rangle$
- intersections trouvées : $\{p_{1,2}\}$

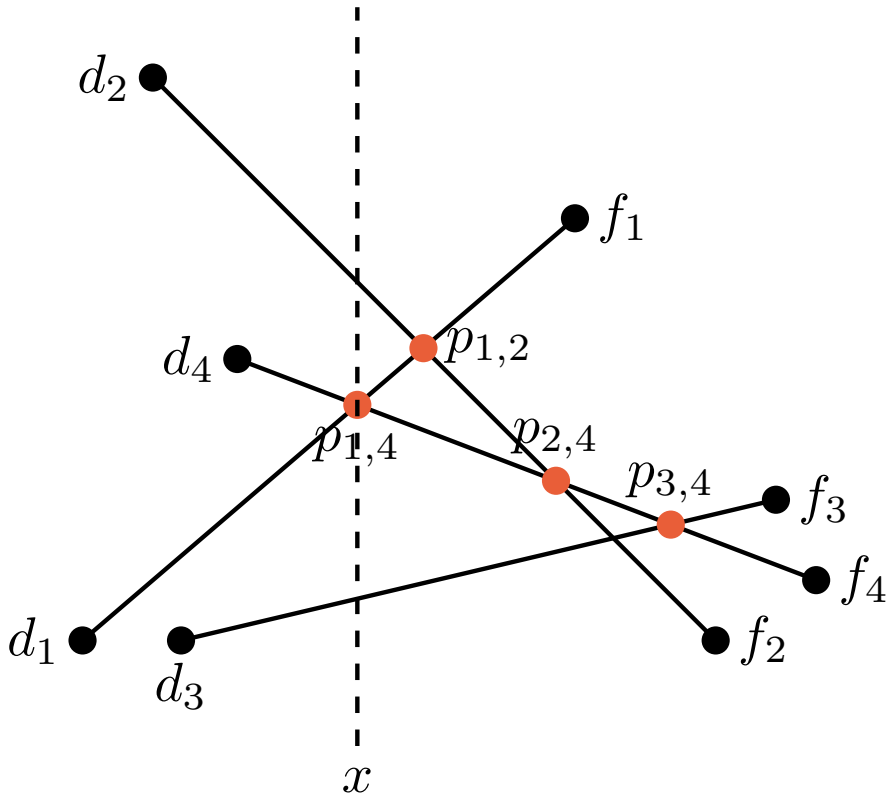
Exemple d'exécution



$x = d_4$

- file de priorité : $p_{1,4} \prec p_{1,2} \prec p_{2,4} \prec f_1 \prec f_2 \prec f_3 \prec f_4$
- état $= \langle s_2, s_4, s_1, s_3 \rangle$
- intersections trouvées : $\{p_{1,2}, p_{1,4}, p_{2,4}\}$

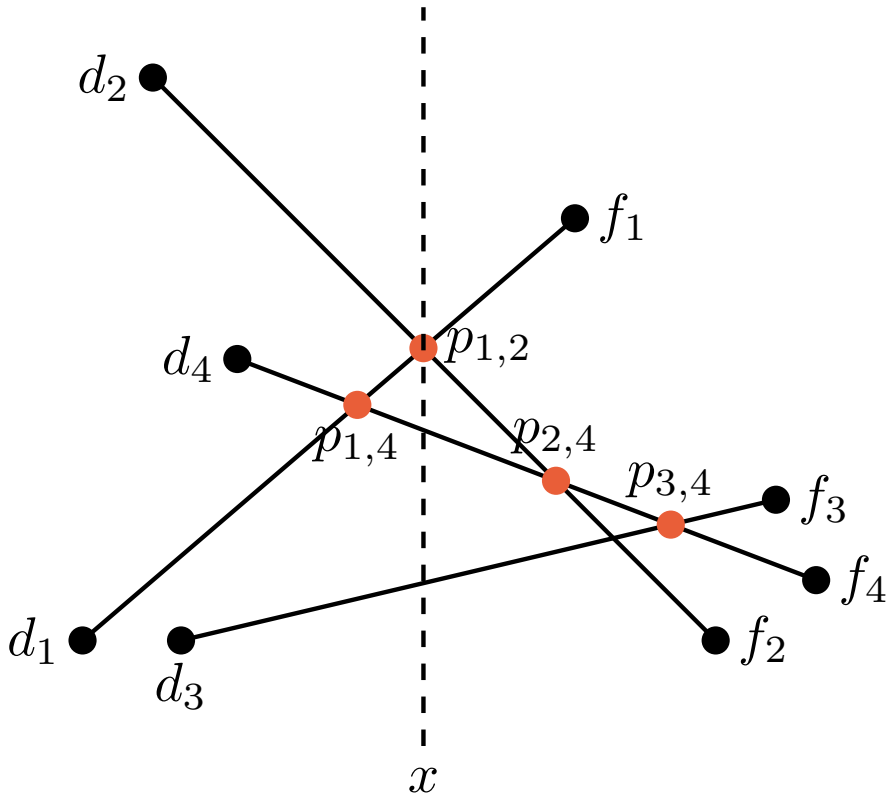
Exemple d'exécution



$x = p_{1,4}$

- file de priorité : $p_{1,2} \prec p_{2,4} \prec f_1 \prec p_{3,4} \prec f_2 \prec f_3 \prec f_4$
- état $= \langle s_2, s_1, s_4, s_3 \rangle$
- intersections trouvées : $\{p_{1,2}, p_{1,4}, p_{2,4}, p_{3,4}\}$

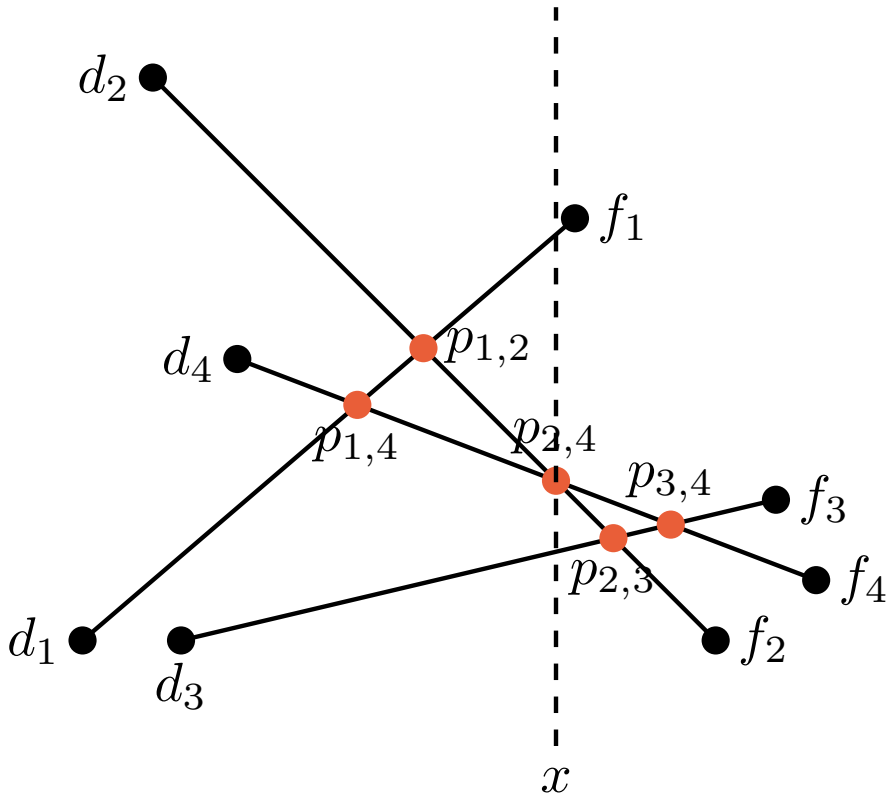
Exemple d'exécution



$x = p_{1,2}$

- file de priorité : $p_{2,4} \prec f_1 \prec p_{3,4} \prec f_2 \prec f_3 \prec f_4$
- état $= \langle s_1, s_2, s_4, s_3 \rangle$
- intersections trouvées : $\{p_{1,2}, p_{1,4}, p_{2,4}, p_{3,4}\}$

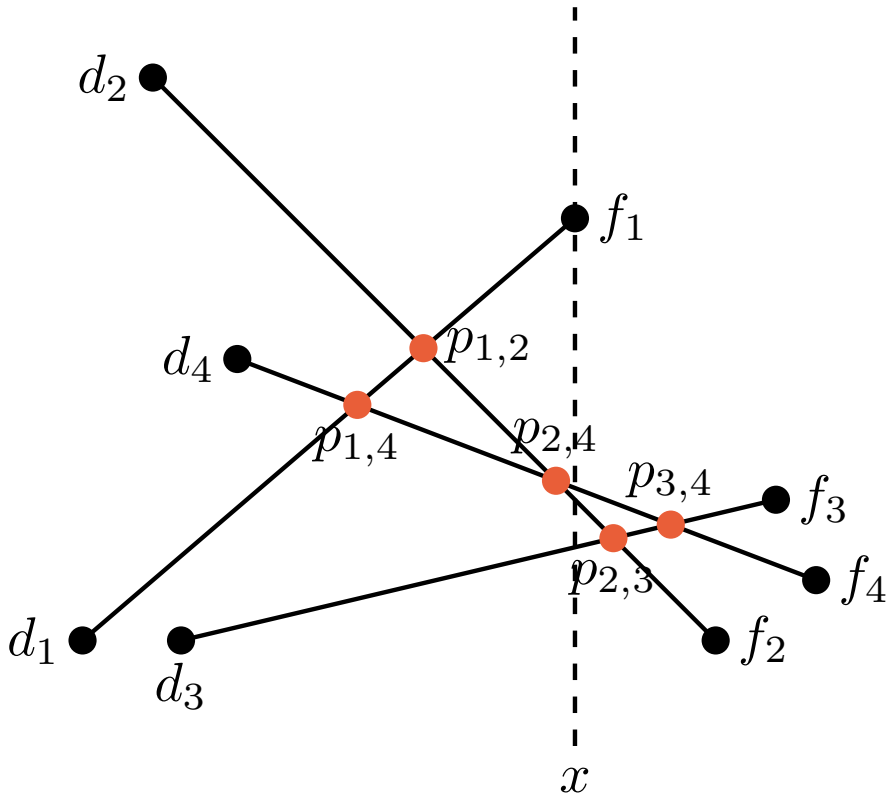
Exemple d'exécution



$x = p_{2,4}$

- file de priorité : $f_1 \prec p_{2,3} \prec p_{3,4} \prec f_2 \prec f_3 \prec f_4$
- état $= \langle s_1, s_4, s_2, s_3 \rangle$
- intersections trouvées : $\{p_{1,2}, p_{1,4}, p_{2,4}, p_{3,4}, p_{2,3}\}$

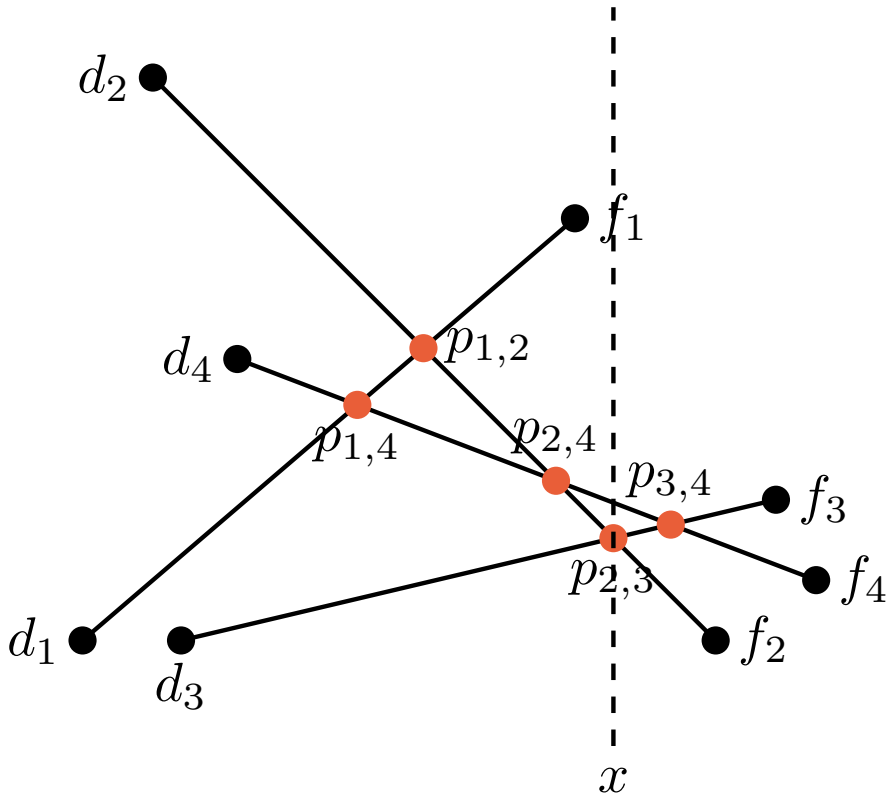
Exemple d'exécution



$x = f_1$

- file de priorité : $p_{2,3} \prec p_{3,4} \prec f_2 \prec f_3 \prec f_4$
- état $= \langle s_4, s_2, s_3 \rangle$
- intersections trouvées : $\{p_{1,2}, p_{1,4}, p_{2,4}, p_{3,4}, p_{2,3}\}$

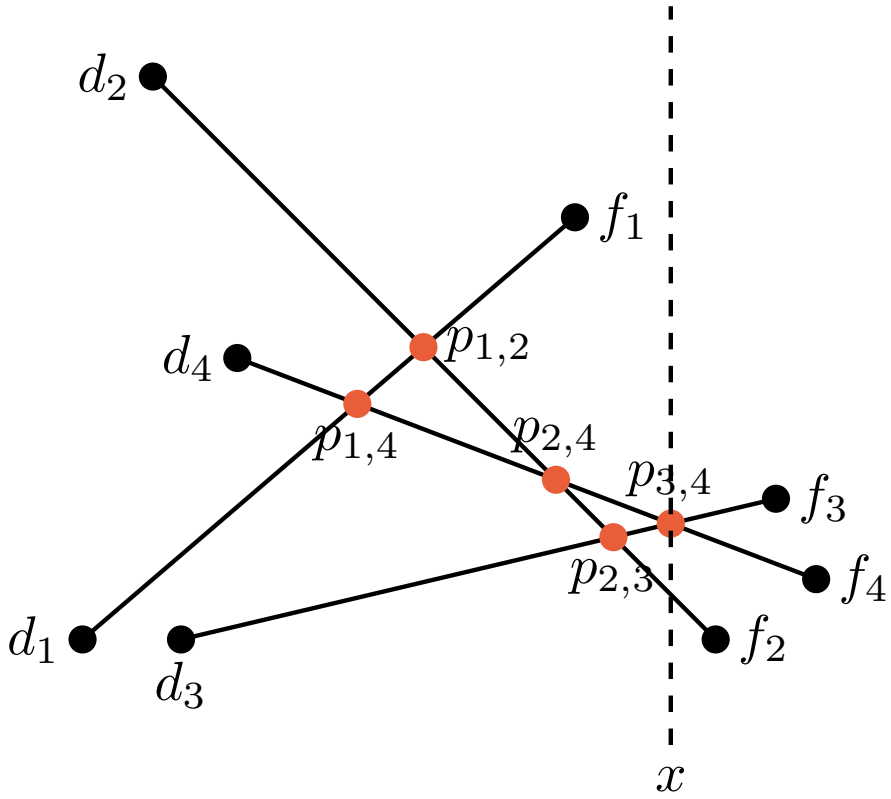
Exemple d'exécution



$x = p_{2,3}$

- file de priorité : $p_{3,4} \prec f_2 \prec f_3 \prec f_4$
- état $= \langle s_4, s_3, s_2 \rangle$
- intersections trouvées : $\{p_{1,2}, p_{1,4}, p_{2,4}, p_{3,4}, p_{2,3}\}$

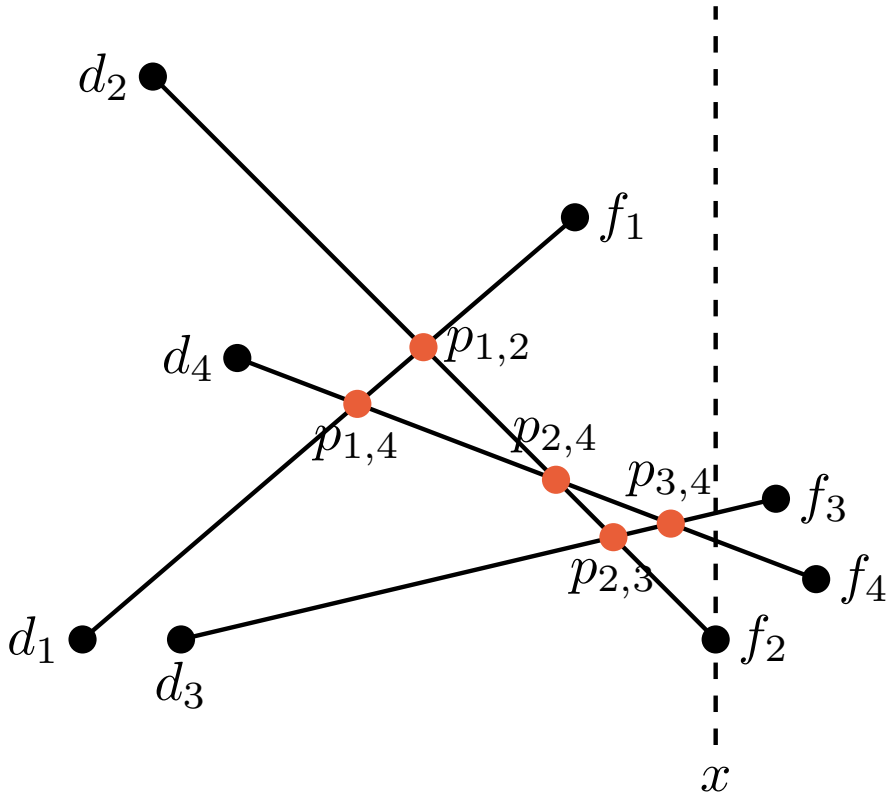
Exemple d'exécution



$x = p_{3,4}$

- file de priorité : $f_2 \prec f_3 \prec f_4$
- état $= \langle s_3, s_4, s_2 \rangle$
- intersections trouvées : $\{p_{1,2}, p_{1,4}, p_{2,4}, p_{3,4}, p_{2,3}\}$

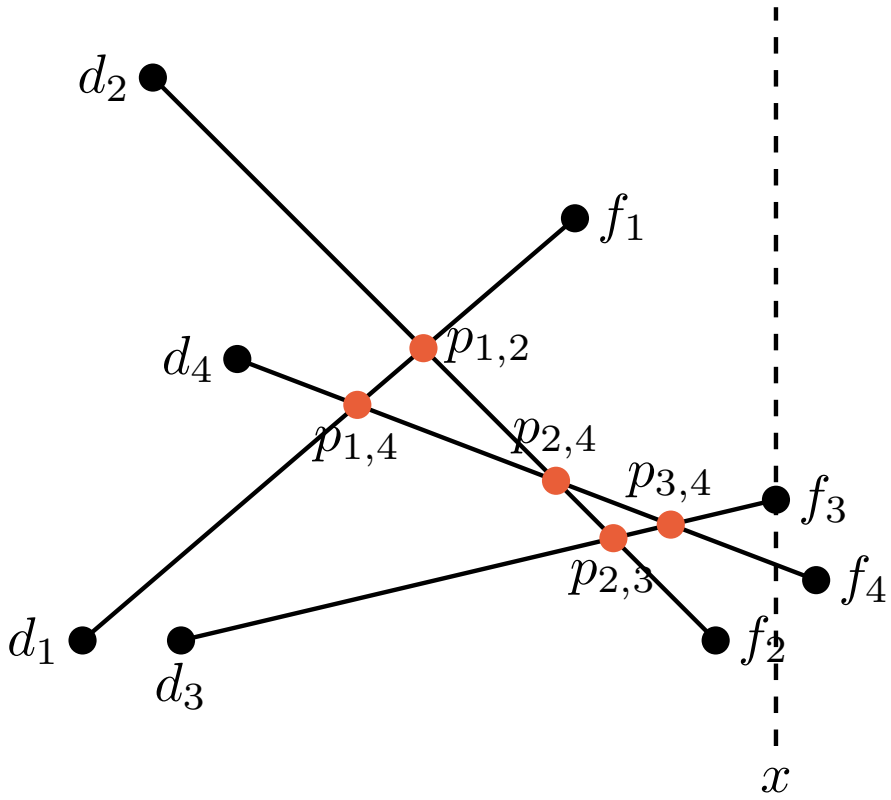
Exemple d'exécution



$x = f_2$

- file de priorité : $f_3 \prec f_4$
- état $= \langle s_3, s_4 \rangle$
- intersections trouvées : $\{p_{1,2}, p_{1,4}, p_{2,4}, p_{3,4}, p_{2,3}\}$

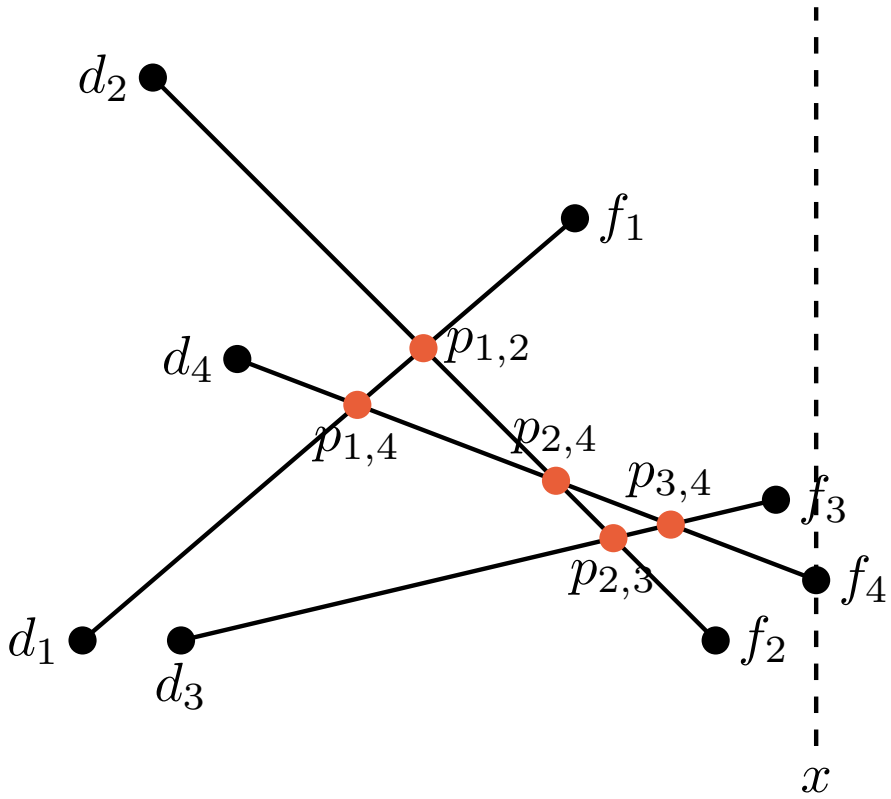
Exemple d'exécution



$x = f_3$

- file de priorité : f_4
- état $= \langle s_4 \rangle$
- intersections trouvées : $\{p_{1,2}, p_{1,4}, p_{2,4}, p_{3,4}, p_{2,3}\}$

Exemple d'exécution



$x = f_4$

- file de priorité : $\emptyset$
- état $= \langle\rangle$
- intersections trouvées : $\{p_{1,2}, p_{1,4}, p_{2,4}, p_{3,4}, p_{2,3}\}$

Structures des données

- ensemble de segments (entrée)
  - **liste chaînée**
- ensemble de points d'intersection (sortie)
  - **liste chaînée**
- file de priorité : `événements`
  - **arbre binaire de recherche**
- `état` : structure triée
  - **liste chaînée triée**

Objectif final du projet

- Évaluer la **performance** des deux algorithmes : glouton et BentleyOttmann

Méthodologie

- ✓ Analyse de complexité théorique pour l'utilisation de différentes structures de données
- ✓ Expériences pour valider l'analyse théorique
  - les données vont être générées par vous en utilisant de scripts shell

- **squelette fourni**
  - organisation du code
  - définition des `struct` utilisées
  - prototypes des fonctions
  - prototypes des structures des données
- **à coder**
  - les primitives de différentes structures
  - l'algorithme glouton
  - l'algorithme BentleyOttmann

Attention ! Vous n'avez pas le droit de modifier les prototypes des fonctions et des structures fournis.

Type générique de données

- `void*`
  - objectif : définir de données génériques
  - le type est précisé dans l'utilisation
  - exemple : une liste chaînée peut contenir des entiers ou des nombres réels ou des segments ou ...

Exemple d'utilisation

```
void * data;
int i = 3;
data = &i;
printf("data = %d\n", *((int*) data));
```

Allocation de la mémoire de façon dynamique

- `void * malloc(size_t size);`
  - `size` : la taille en octets de la mémoire à allouer
  - renvoie : un pointeur vers l'adresse mémoire allouée
  - si l'allocation a échoué, renvoie `NULL` $\Rightarrow$ faire le test

Exemple 1

```
double * pi = (double*) malloc(sizeof(double));
if (pi == NULL) {fprintf(stderr, "Erreur!"); exit(1);}
*pi = 3.14159265359;
printf("pi = %f\n", *pi);
```

Exemple 2

```
int * T = (int*) malloc(3 * sizeof(int));
if (T == NULL) {fprintf(stderr, "Erreur!"); exit(1);}
T[0] = 0; T[1] = 1; T[2] = 2;
for (int i = 0; i < 3; i++)
    printf("T[%d] = %d\n", i, T[i]);
```

Allocation de la mémoire de façon dynamique

- Quand ?
  1. on ne connaît pas la taille à l'avance
  2. on veut gérer des données de taille importante
  3. on veut que la mémoire d'une variable reste allouée même en dehors du sous-programme dans lequel elle est déclarée

Allocation de la mémoire de façon dynamique

Exemple de cas (3)

```
int * f() {
    int A[3];
    A[0] = 0; A[1] = 1; A[2] = 2;
    return A;
}

int main() {
    int * T;
    T = f();
    for (int i = 0; i < 3; i++)
        printf("T[%d] = %d\n", i, T[i]);
}
```

- Avertissement pendant la compilation

```
warning: function returns address of local variable
```

- Exécution : Segmentation fault (core dumped)

Allocation de la mémoire de façon dynamique

Exemple de cas (3) – correction

```
int * f() {
    int * A = (int*) malloc(3 * sizeof(int));
    if (A == NULL) {fprintf(stderr, "Erreur!"); exit(1);}
    A[0] = 0; A[1] = 1; A[2] = 2;
    return A;
}

int main() {
    int * T;
    T  = f();
    for (int i = 0; i < 3; i++)
        printf("T[%d] = %d\n", i, T[i]);
}
```

- Résultat

```
T[0] = 0
T[1] = 1
T[2] = 2
```

Libération de la mémoire

- `void free(void *ptr);`
  - libération de la mémoire accédée par le pointeur `ptr`

Attention !

Pour chaque appel à `malloc` lors de l'exécution d'un programme il faut un appel à `free`

Libération de la mémoire

Exemple

```
int main() {
    int * T = (int*) malloc(3 * sizeof(int));
    if (T == NULL) {
        fprintf(stderr, "Erreur!");
        exit(1);
    }
    T[0] = 0; T[1] = 1; T[2] = 2;
    for (int i = 0; i < 3; i++)
        printf("T[%d] = %d\n", i, T[i]);
    free(T);
}
```

Exemple : tableau de 2 dimensions (3 x 2)

```
int main() {
    int ** T = (int**) malloc(3 * sizeof(int*));
    for (int i = 0; i < 3; i++)
        T[i] = (int*) malloc(2 * sizeof(int));
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 2; j++)
            T[i][j] = i * j;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 2; j++) {
            printf("%d ", T[i][j]);
        }
        printf("\n");
    }
    for (int i = 0; i < 3; i++)
        free(T[i]);
    free(T);
}
```

Pointeurs de fonction

- Permettent de stocker une référence vers une fonction
- Déclaration

```
void (*pointeurFonction)(void);
int (*pointeurFonction2)(void);
int (*pointeurFonction3)(double, double);
```

- Utilisation

```
void carre(int n) {
    printf("n^2 = %d\n", n*n);
}
int main(void) {
    void (*ptrFonction)(int); // déclaration
    ptrFonction = carre;      // affectation
    ptrFonction(5);           // utilisation
    return 0;
}
```

Autres remarques sur le code C

- Utilisation du mot-clé `static` pour certains sous-programmes
  - sous-programmes privés dans le fichier où ils sont définis
  - on ne peut pas les appeler dehors de ce fichier
  - pas de prototype dans le fichier `.h`
  - objectif : protéger nos structures de données
- Déclaration de paramètres de fonctions avec le mot-clé `const`
  - objectif : garantir que l'argument ne sera pas modifié par la fonction
  - exemple : une procédure qui affiche un sommet ne doit pas le modifier

Compilation & Exécution

- `Makefile` fourni (à modifier si besoin)

- `make` : compilation
- `make run <paramètres>` : compilation & exécution
- `make clean` : suppression de fichiers intermédiaires
- `make clean run` : nettoyage, compilation & exécution
- `make memorycheck` : utilisation du programme `valgrind` pour examiner l'état de la mémoire à la fin de l'exécution de votre programme

Compilation & Exécution (suite)

- `make test` : compilation & exécution des tests (c'est à vous de les écrire)
- `make deletetest` : suppression de l'exécutable qui lance les tests
- `make testmemorycheck` : utilisation du programme `valgrind` pour examiner l'état de la mémoire à la fin de l'exécution de vos tests

- `make jni` : compilation & création de la bibliothèque JNI nécessaire pour l'interface graphique
- `make deletelib` : suppression du fichier de la bibliothèque JNI

Présentation du canevas...

- `include/util.h` et `src/util.c` (code fourni)

Présentation du canevas...

- `include/util.h` et `src/util.c` (code fourni)
- `include/rational.h` et `src/rational.c`

Nombres rationnels

- problèmes de **précision** liés à l'utilisation des nombres réels
- il n'y a pas de type nombre rationnel en C
- on définie notre type :

```
struct Rational {
    long long numerator;   // numérateur
    long long denominator;   // dénominateur
};
```

Nombres rationnels

- simplifier un nombre rationnel :

  le numérateur et le dénominateur doivent être premiers entre eux

- **pourquoi ?**
  éviter de dépasser l'espace de stockage des entiers longs (8 octets)

- appliquer `PGCD(num,den)` après chaque opération

- **exemple**
  - $r = \frac{15}{6}$
  - $PGCD(15, 6) = 3$
  - simplification : $r = \frac{15/3}{6/3} = \frac{5}{2}$

Présentation du canevas...

- `include/util.h` et `src/util.c` (code fourni)
- `include/rational.h` et `src/rational.c`
- `include/geometry.h` et `src/geometry.c`

Points et Segments

```
struct Point {
    struct Rational x;  // abscisse
    struct Rational y;  // ordonnée
};
```

```
struct Segment {
    struct Point * endpoint1;  // début et fin
    struct Point * endpoint2;  // du segment
};
```

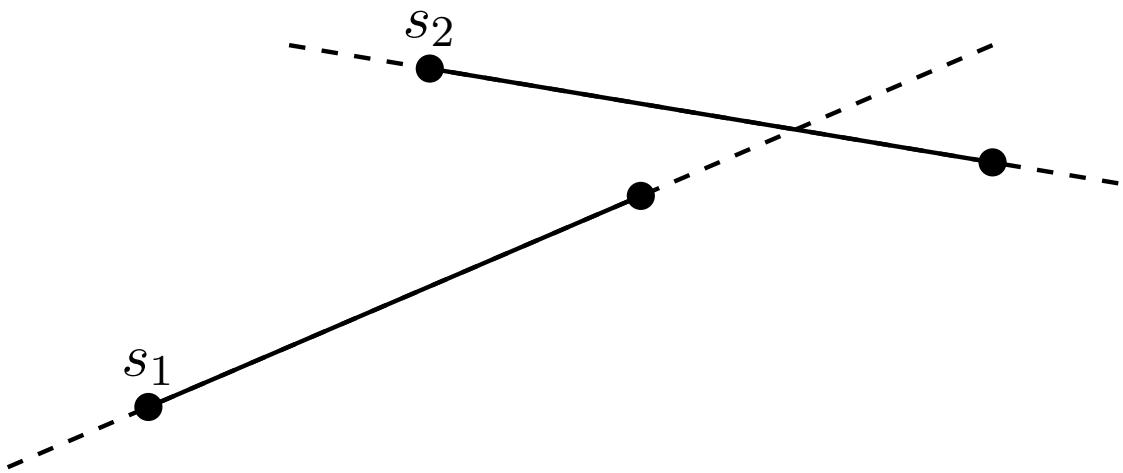
Points et Segments : utilisation

```
struct Rational x1 = {3, 2};
struct Rational y1 = {1, 1};
struct Point * p1 = new_point(x1, y1);

struct Rational x2 = {5, 2};
struct Rational y2 = {3, 1};
struct Point * p2 = new_point(x2, y2);

struct Segment * s = new_segment(p1, p2);
```

Intersection de deux segments



- **attention !** s_1 et s_2 ne s'intersectent pas même si les droites directrices intersectent
- quelles sont les conditions d'avoir une intersection de deux segments ?
- pour rappel, pas de segment vertical

Fonctions importantes

```
// Définition d'ordre
int point_precedes(const void * p1,
                   const void * p2);
int segment_precedes(const struct Segment * s1,
                     const struct Segment * s2,
                     struct Rational x0);

// Gestion d'intersections
int do_intersect(const struct Segment * s1,
                 const struct Segment * s2);
struct Point * get_intersection_point(
                 const struct Segment * s1,
                 const struct Segment * s2);
```

Présentation du canevas...

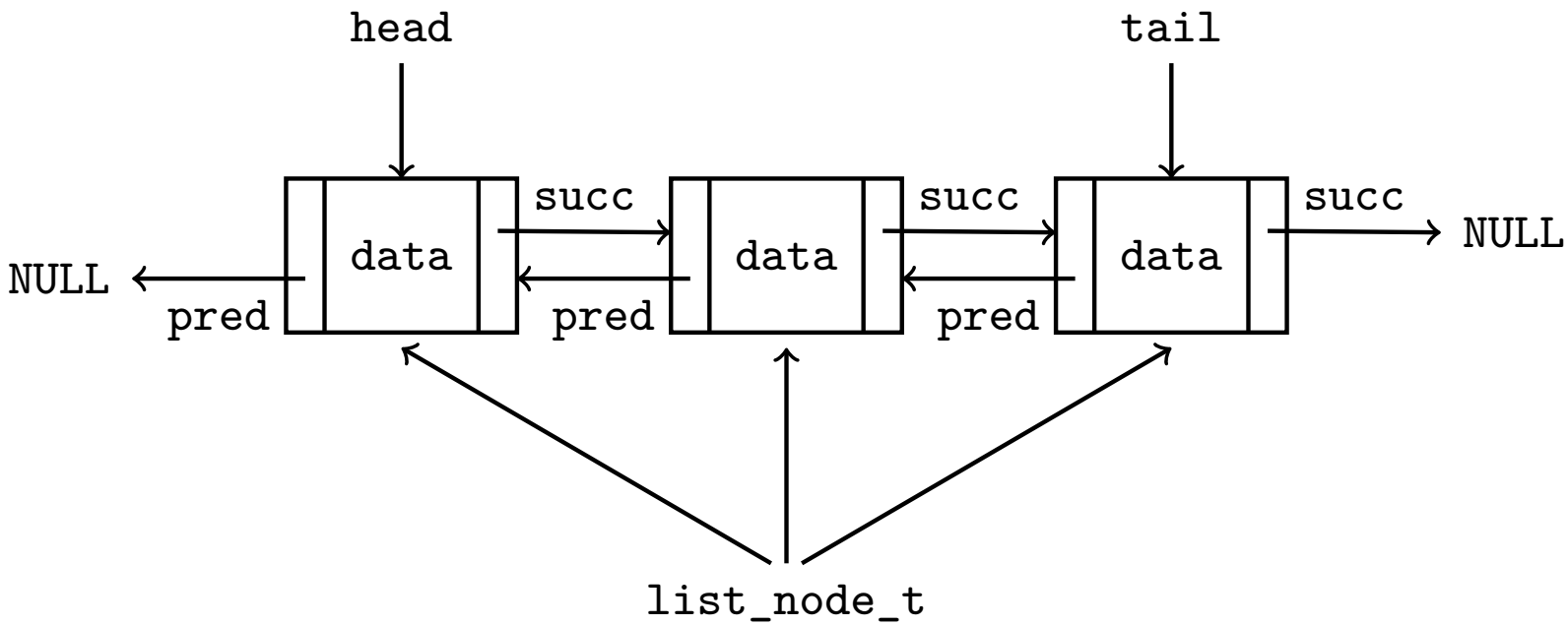
- `include/util.h` et `src/util.c` (code fourni)
- `include/rational.h` et `src/rational.c`
- `include/geometry.h` et `src/geometry.c`
- `include/list.h` et `src/list.c`

Listes chaînées

```
struct list_node_t {
    void * data;
    struct list_node_t * successor;
    struct list_node_t * predecessor;
};
```

```
struct list_t {
    struct list_node_t * head,
    struct list_node_t * tail;
    unsigned int size;
};
```

Listes chaînées



Listes chaînées

```
struct list_node_t {
    void * data;
    struct list_node_t * successor;
    struct list_node_t * predecessor;
};
```

- **Pourquoi ?**
  implémenter une seule liste et l'utiliser avec de différentes données
- **Comment afficher une liste ?**
  utiliser des pointeurs de fonction

```
void view_list(const struct list_t * L,
               void (*viewData)(const void *));
```

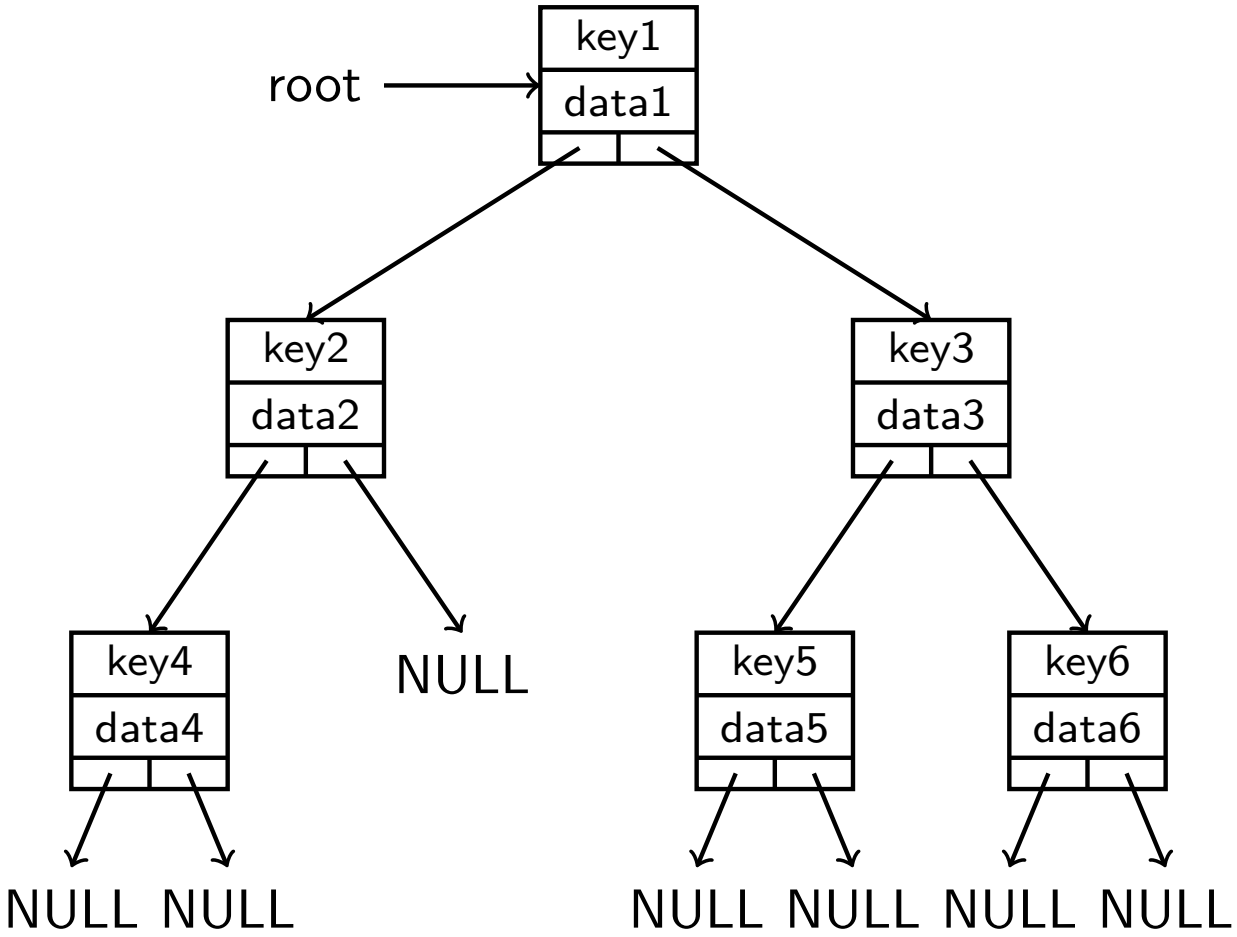
Présentation du canevas...

- `include/util.h` et `src/util.c` (code fourni)
- `include/rational.h` et `src/rational.c`
- `include/geometry.h` et `src/geometry.c`
- `include/list.h` et `src/list.c`
- `include/tree.h` et `src/tree.c`

Arbres Binaires de Recherche

- arbres **binaires** : chaque nœud a 0 ou 1 ou 2 fils
- pour chaque nœud : une valeur (ou donnée) et une **clé**
- arbres **ordonnés** par rapport aux clés
- pour chaque nœud :
  - toutes les clés du fils gauche sont inférieures à sa clé
  - toutes les clés du fils droit sont supérieures à sa clé
- parcours **infixé** $\Rightarrow$ liste ordonnée des clés

Arbres Binaires de Recherche



- ordre des clés ? $\text{key4} \prec \text{key2} \prec \text{key1} \prec \text{key5} \prec \text{key3} \prec \text{key6}$
- **structure récursive**

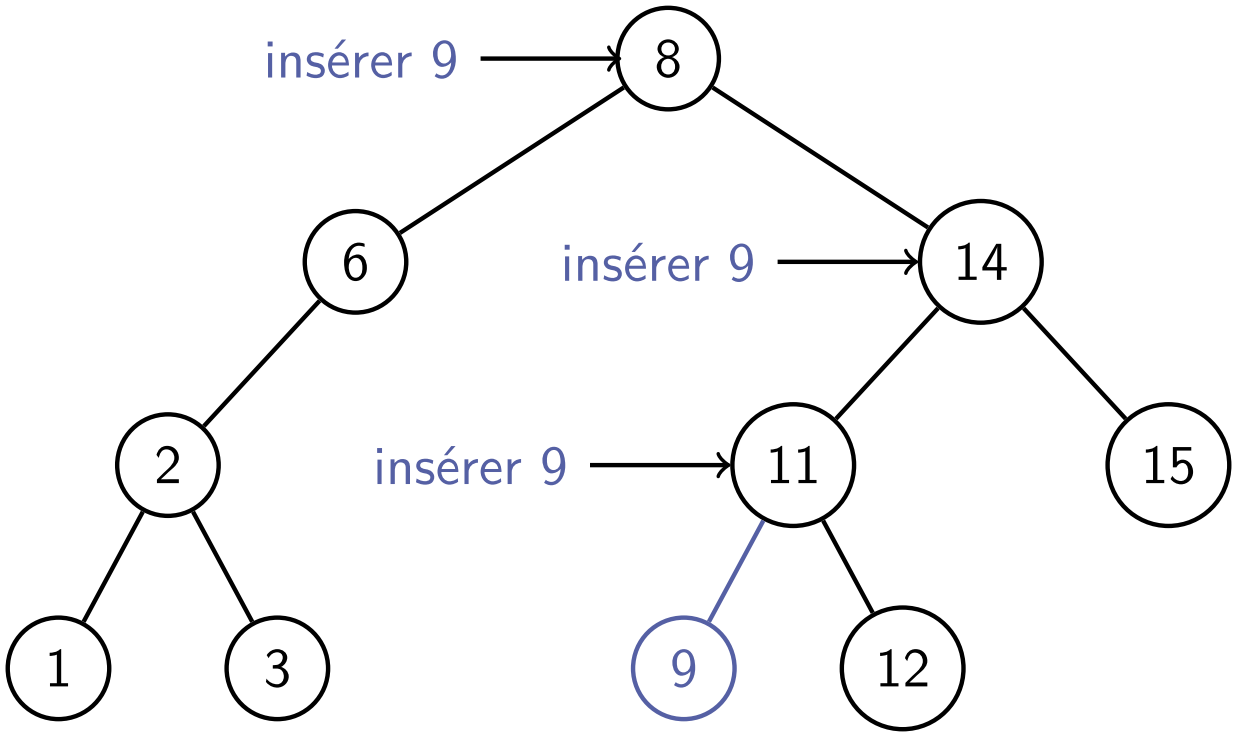
Arbres Binaires de Recherche en C

```c
struct tree_node_t {
    void * key;
    void * data;
    struct tree_node_t * left;
    struct tree_node_t * right;
};


struct tree_t {
    struct tree_node_t * root;
    unsigned int size;
};
```

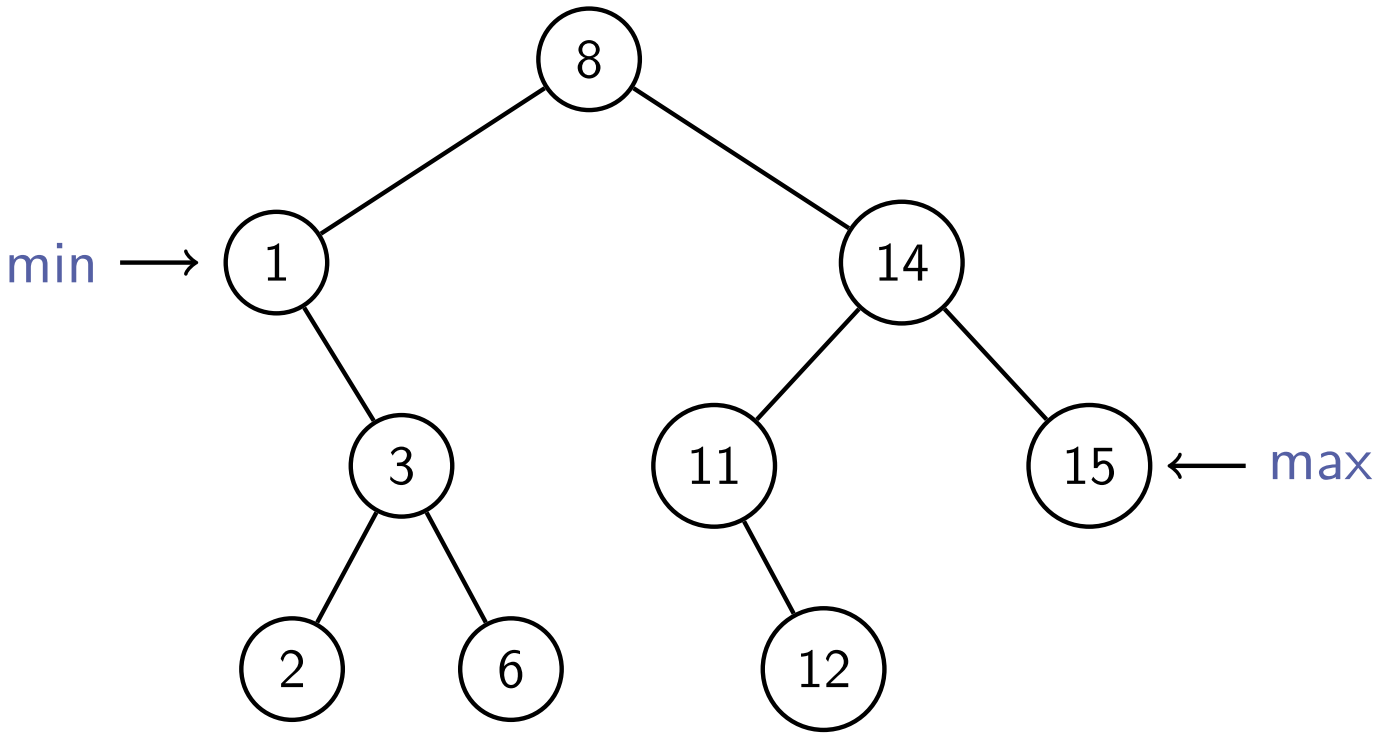
Arbres Binaires de Recherche : Insertion

- Descendre de façon récursive soit à gauche soit à droit
- Insérer le nouveau élément toujours comme une nouvelle feuille



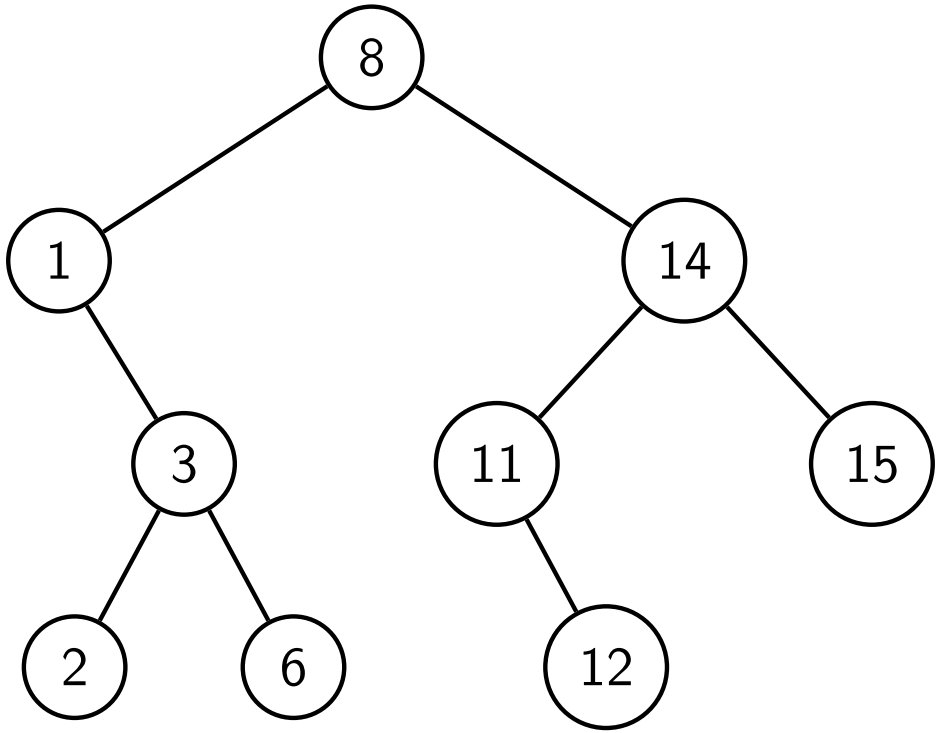
Arbres Binaires de Recherche : recherche/min/max

- **recherche** : même principe que l'insertion
- **min** : descendre à gauche jusqu'à que le fils gauche n'existe pas
- **max** : descendre à droite jusqu'à que le fils droit n'existe pas



Arbres Binaires de Recherche : successeur (prédécesseur)

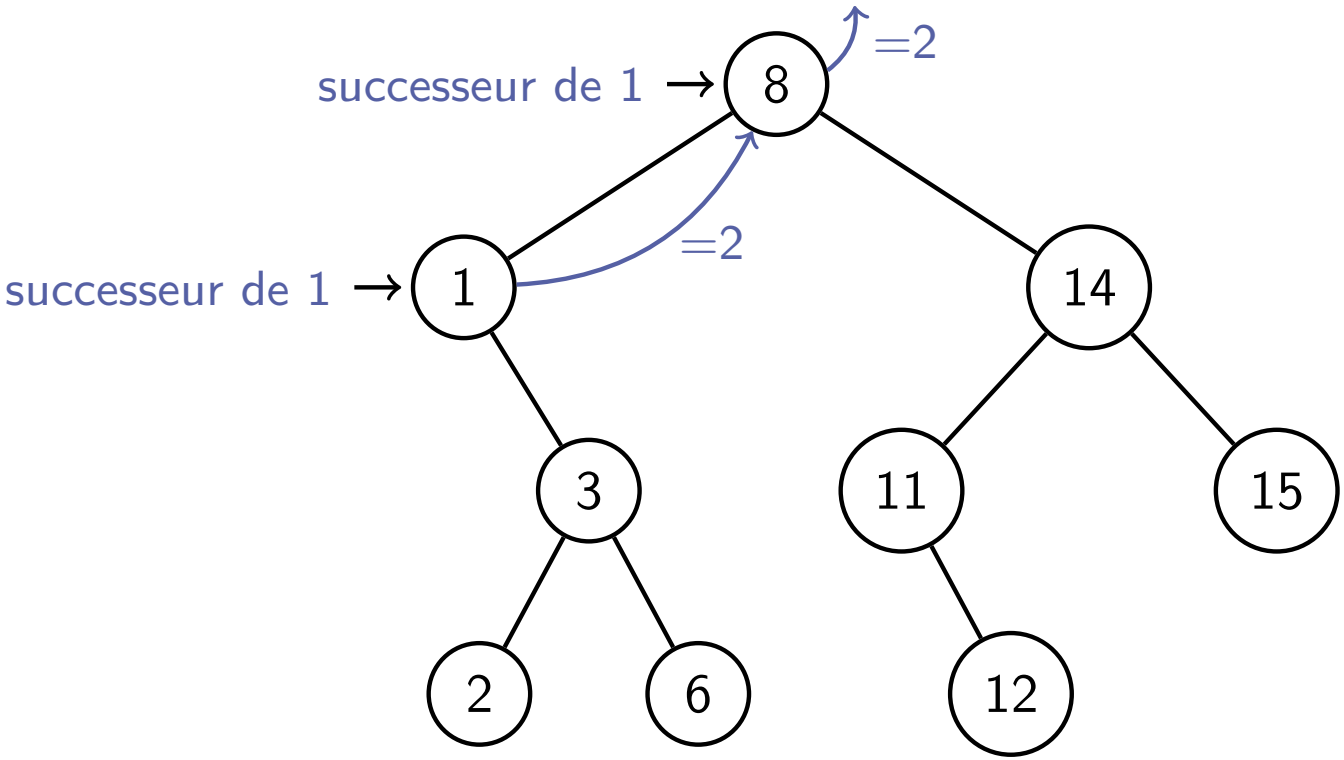
- **successeur** du nœud v : le nœud avec la clé suivante
  (= le nœud avec la plus petit clé qui est plus grande que la clé de v)



- successeur(3) = 6
- successeur(1) = 2
- successeur(6) = 8

Arbres Binaires de Recherche : successeur (prédécesseur)

- Descendre de façon récursive jusqu'au nœud v pour lequel on cherche le successeur
- 2 cas à considérer

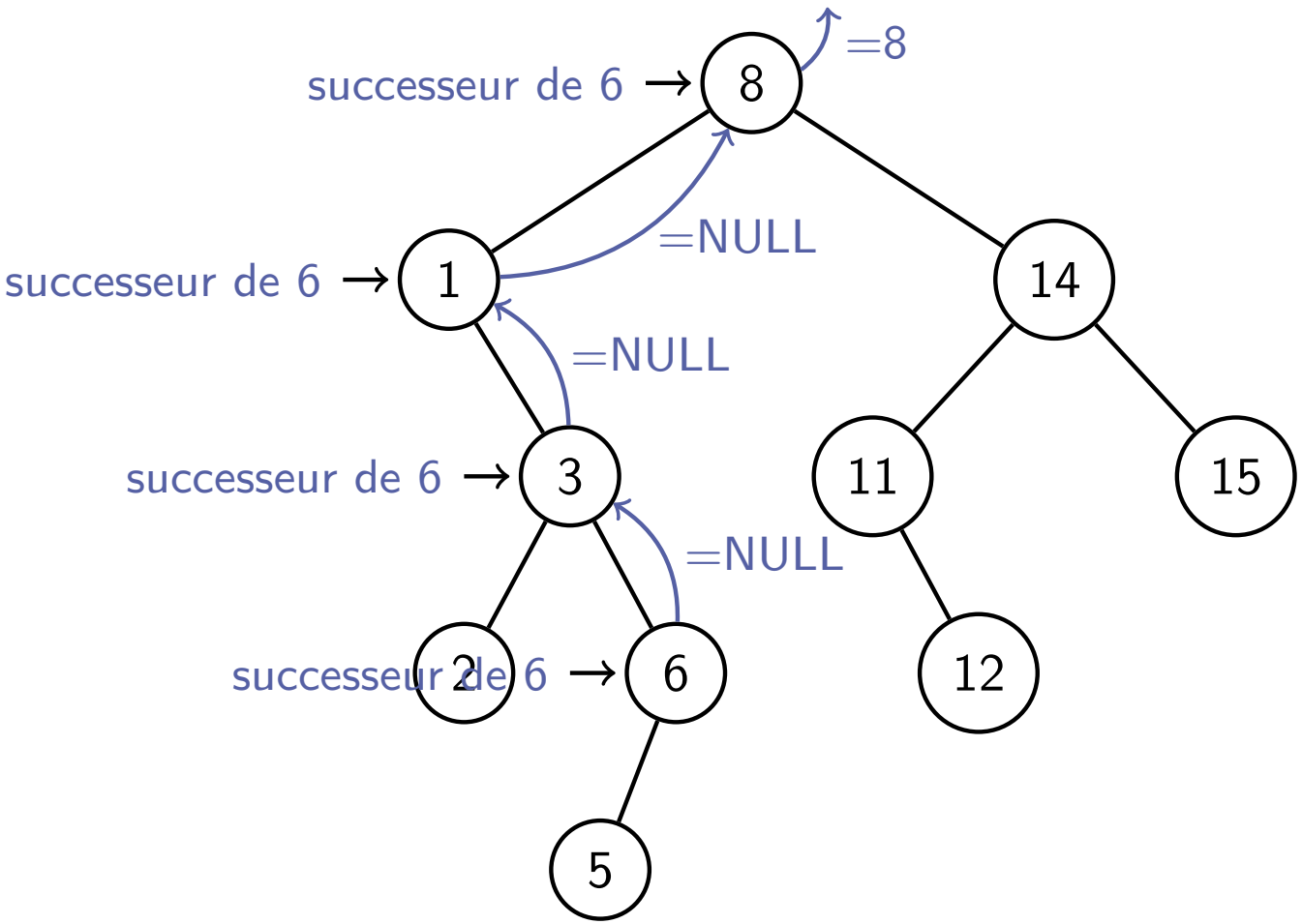


- Cas 1 : le fils droit de v existe $\Rightarrow$

$$\text{successeur}(v) = \{\text{min clé de son sous-arbre droit}\}$$

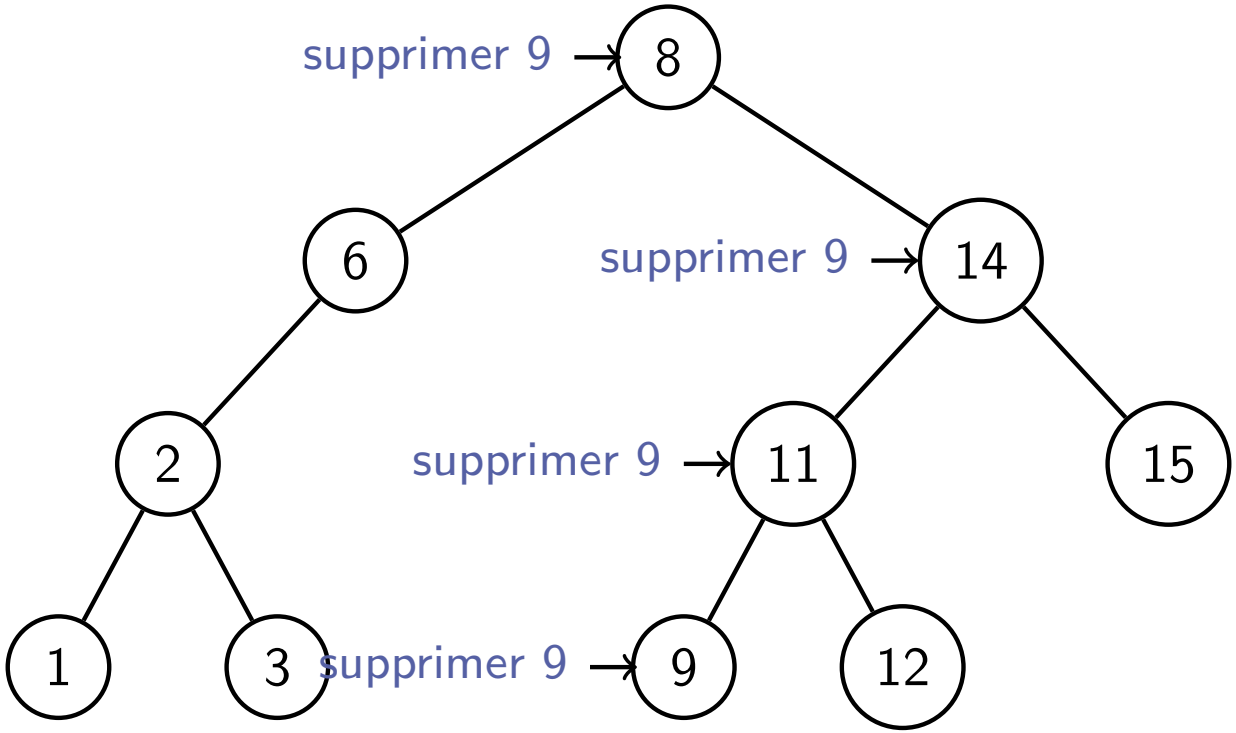
Arbres Binaires de Recherche : successeur (prédécesseur)

- Cas 2 : le fils droit de v n'existe pas
- En remontant de la récursion, le premier nœud qui a descendu à gauche est le successeur de v



Arbres Binaires de Recherche : Suppression

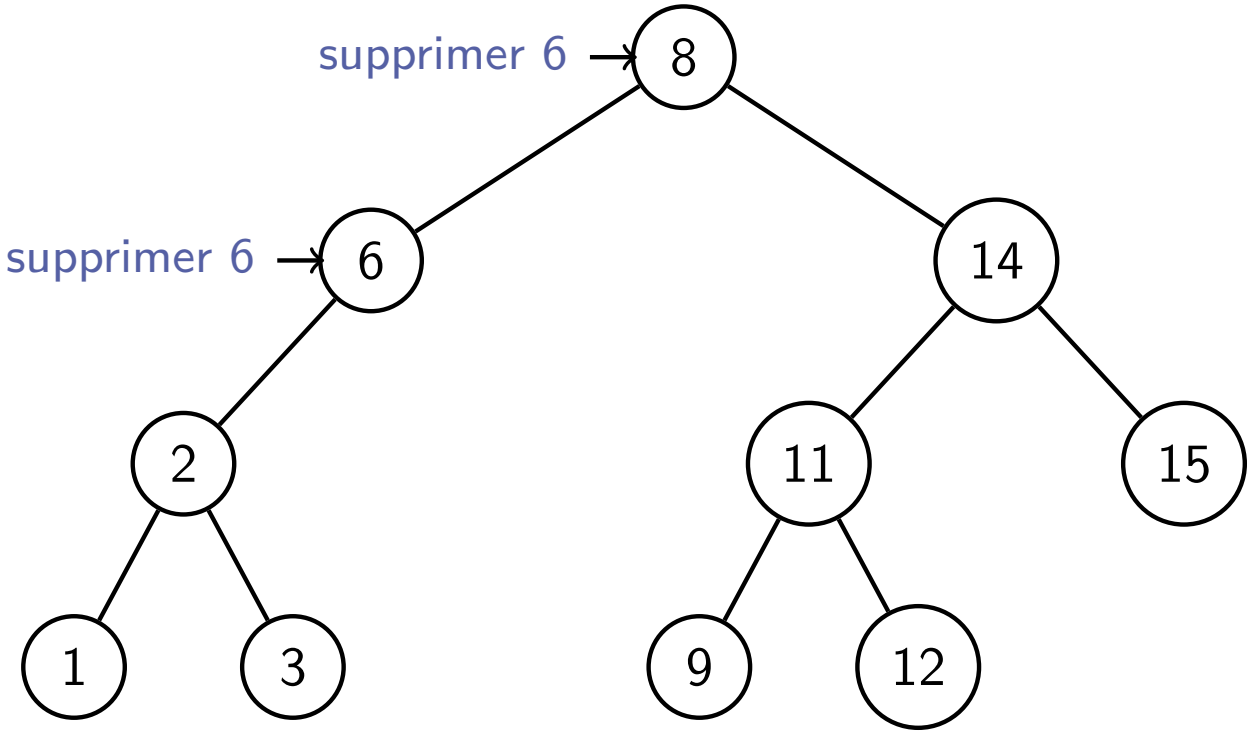
- Descendre de façon récursive soit à gauche soit à droit
- 3 cas à considérer



- Cas 1 : le nœud à supprimer est une feuille

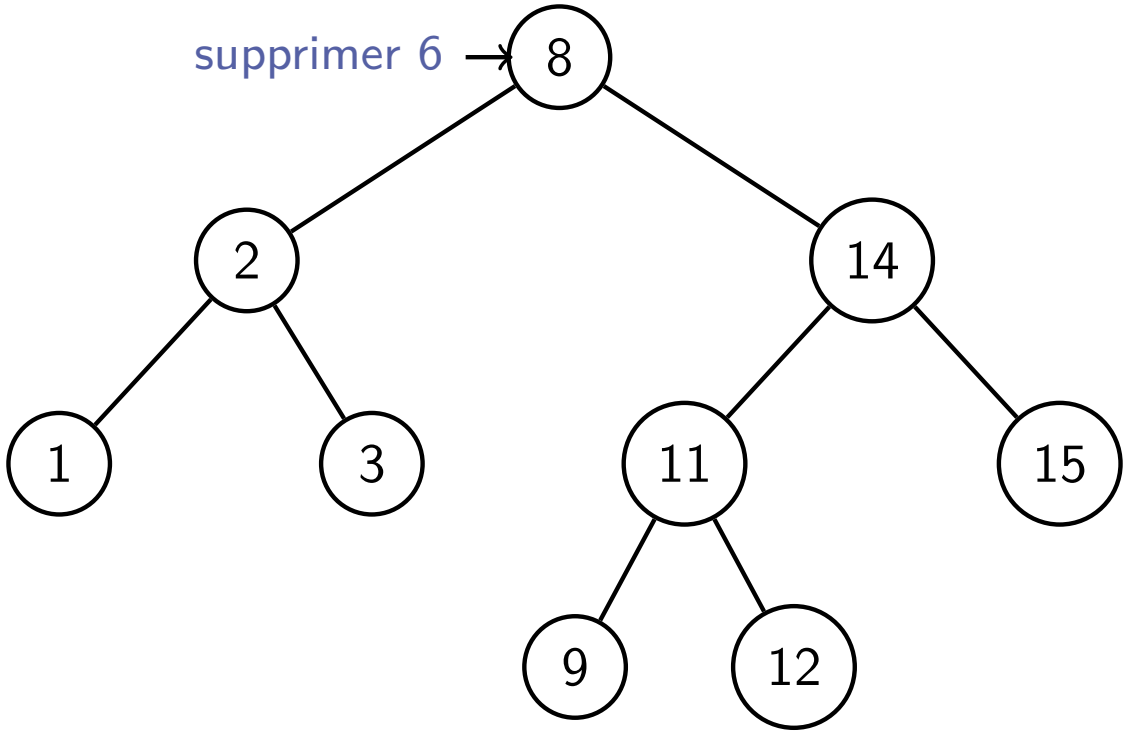
Arbres Binaires de Recherche : Suppression

- Cas 2 : le nœud à supprimer a un seul fils



Arbres Binaires de Recherche : Suppression

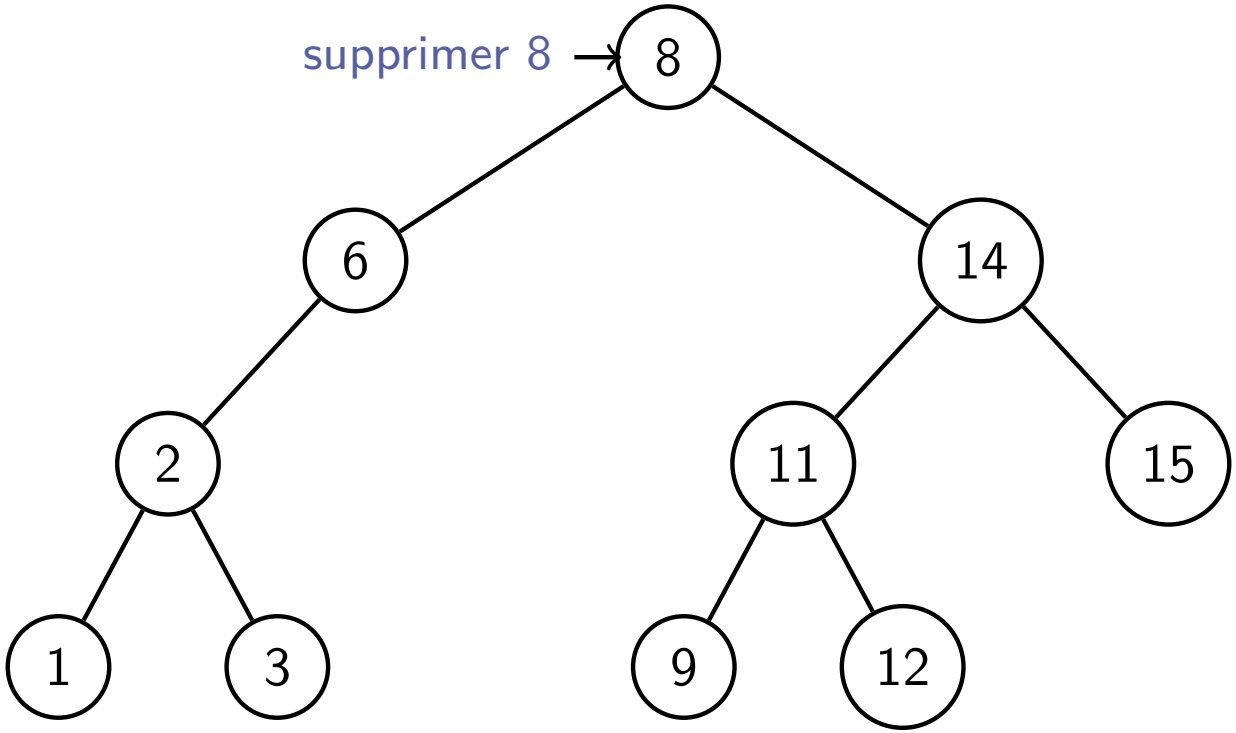
- Cas 2 : le nœud à supprimer a un seul fils



- Lier son père avec son fils unique

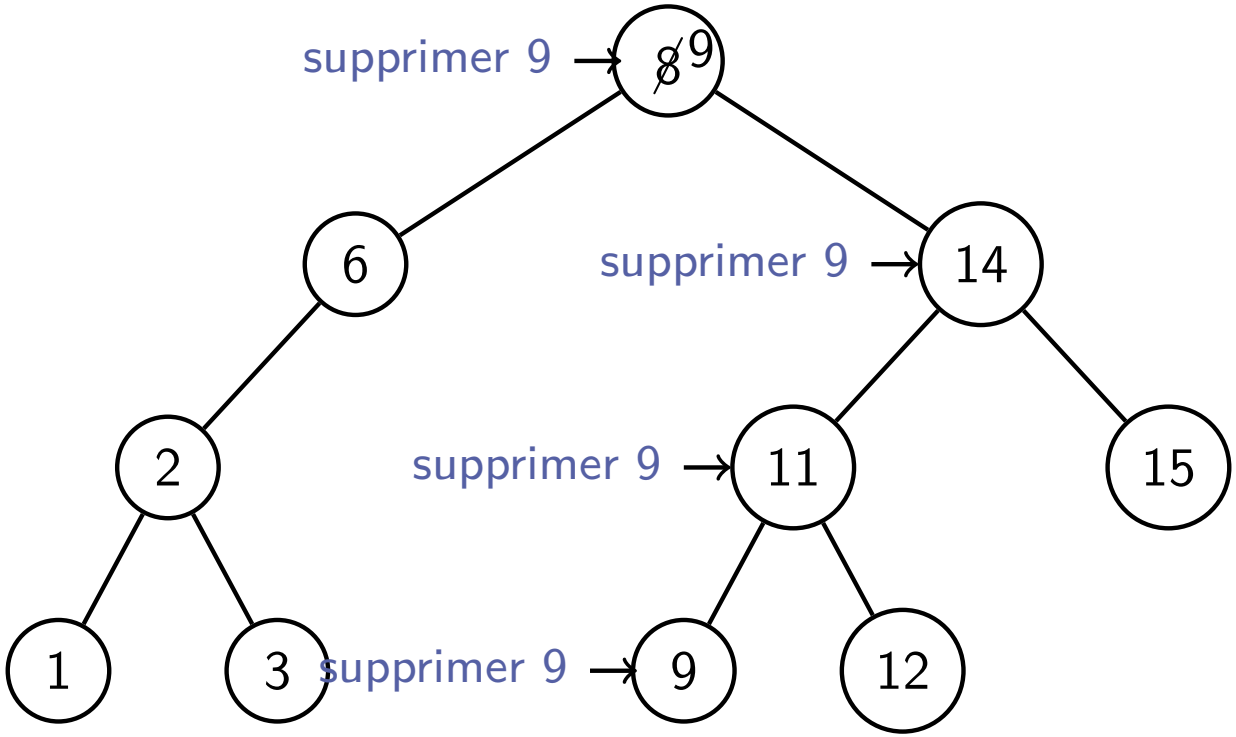
Arbres Binaires de Recherche : Suppression

- Cas 3 : le nœud à supprimer a deux fils



Arbres Binaires de Recherche : Suppression

- Cas 3 : le nœud à supprimer a deux fils



- Remplacer ses données et sa clé avec ceux de son successeur
- Supprimer récursivement le nœud de son successeur

Présentation du canevas...

- `include/util.h` et `src/util.c` (code fourni)
- `include/rational.h` et `src/rational.c`
- `include/geometry.h` et `src/geometry.c`
- `include/list.h` et `src/list.c`
- `include/tree.h` et `src/tree.c`
- `include/algo.h` et `src/algo.c`

Gestion des fichiers

```
// ranger dans une liste les segments stockés
// dans le fichier texte de nom input_filename
struct list_t * load_segments(
        const char * input_filename);

// ranger dans un fichier texte de nom
// output_filename les points de la liste
// intersections
void save_intersections(
        const char * output_filename,
        const struct list_t * intersections);
```

Format du fichier de segments

- chaque ligne $\Rightarrow$ un segment `S=(P1,P2)`
- format : `P1.x,P1.y P2.x,P2.y`
- les points sont séparés par une espace (pas d'autre espace)

```
1/1,1/1 4/1,4/1
1/1,5/1 4/1,2/1
5/1,5/1 6/1,0/1
-1/1,0/1 0/1,4/1
-1/1,4/1 3/1,0/1
8/1,1/1 0/1,5/2
```

Format du fichier des points d'intersection

- chaque ligne $\Rightarrow$ un point `P=(x,y)`
- format : `P.x,P.y`
- pas d'espace entre l'abscisse et l'ordonnée

```
3/1,3/1
3/2,3/2
40/19,40/19
40/7,10/7
-1/5,16/5
8/13,31/13
```

Présentation du canevas...

- `include/util.h` et `src/util.c` (code fourni)
- `include/rational.h` et `src/rational.c`
- `include/geometry.h` et `src/geometry.c`
- `include/list.h` et `src/list.c`
- `include/tree.h` et `src/tree.c`
- `include/algo.h` et `src/algo.c`
- `src/main.c` (code fourni)

Présentation du canevas...

- `include/util.h` et `src/util.c` (code fourni)
- `include/rational.h` et `src/rational.c`
- `include/geometry.h` et `src/geometry.c`
- `include/list.h` et `src/list.c`
- `include/tree.h` et `src/tree.c`
- `include/algo.h` et `src/algo.c`
- `src/main.c` (code fourni)
- Interface graphique : `java/` (code fourni)

Interface graphique

- JavaFX (cours d'Interfaces Graphique du semestre 4)
- Fonctionnalités :
  - choisir un fichier contenant un ensemble des segments
  - afficher les segments
  - choisir l'algorithme de calcul des intersections des segments : l'algorithme glouton ou l'algorithme de Bentley et Ottmann
  - appliquer l'algorithme choisi en appelant le code que vous allez implémenter en C
  - afficher les points d'intersections

Comment faire ?

- Java Native Interface : appeler une bibliothèque écrite en C par un code Java
  - `http://web.archive.org/web/20120419230023/http://java.sun.com/docs/books/jni/html/start.html`

- Terminal BASH sous Linux
  (supposons qu'on se trouve sur le dossier canevas)

```
make jni   // compiler la bibliothèque en C
cd java/   // se déplacer sur le dossier java
make       // compiler le code en Java
make run   // exécuter l'interface graphique
```

Présentation du canevas...

- `include/util.h` et `src/util.c` (code fourni)
- `include/rational.h` et `src/rational.c`
- `include/geometry.h` et `src/geometry.c`
- `include/list.h` et `src/list.c`
- `include/tree.h` et `src/tree.c`
- `include/algo.h` et `src/algo.c`
- `src/main.c` (code fourni)
- Interface graphique : `java/` (code fourni)
- Scripts BASH : `script/`

Scripts BASH

- Pour faire quoi ?
  - génération des instance aléatoires
  - exécution des expériences
  - analyse des résultats
- cours “Outils Système”
- découvrir nouvelles commandes (`seq`, `getopts`, etc)
- pour exécuter `run_experiments.sh` il faut :
  - finir le code C
  - compiler le code fourni du fichier `src/expe.c` avec

```
make expe
```

Introduction sur GIT

Sauvegarde : le bon vieux temps...

- Problèmes basiques :
  - “Oh, mon disque est tombé en panne.” / “Quelqu’un a volé mon ordinateur portable !”
  - “@#% ! !, je viens de supprimer un fichier important !”
  - “Oops, j’ai introduit un bogue il y a longtemps dans mon code, comment il était avant ?”
- Solutions :
  - Copier :

    ```
    $ cp -r ~/project/ ~/backup/
    ```
  - Sauvegarder l’histoire :

    ```
    $ cp -r ~/project/ ~/backup/project-2013-02-02
    ```
  - ...

Travailler en équipe : le bon vieux temps

- Problèmes basiques : Plusieurs personnes travaillant sur le même ensemble de fichiers
  1. “Hey, tu as modifié le même fichier que moi, comment on fusionne ?”
  2. “Tes modifications sont brisées, ton code ne compile même pas. Corrige tes modifications avant de me l’envoyer !”
- Solutions :
  - Deux personnes ne travaillent jamais en même temps sur le même fichier. $\Rightarrow$ Ne marche pas bien...
  - Plusieurs personnes travaillent sur le même répertoire $\Rightarrow$ Douloureux à cause de (2) ci-dessus.
  - Plusieurs personnes travaillent pour éviter les conflits et fusionnent plus tard.

Fusion : Problème et Solution

- Ma version

```
#include <stdio.h>

int main () {
  printf("Hello");

  return EXIT_SUCCESS;
}
```

- Ta version

```
#include <stdio.h>

int main () {
  printf("Hello!\n");

  return 0;
}
```

- Version initiale

```
#include <stdio.h>

int main () {
  printf("Hello");

  return 0;
}
```

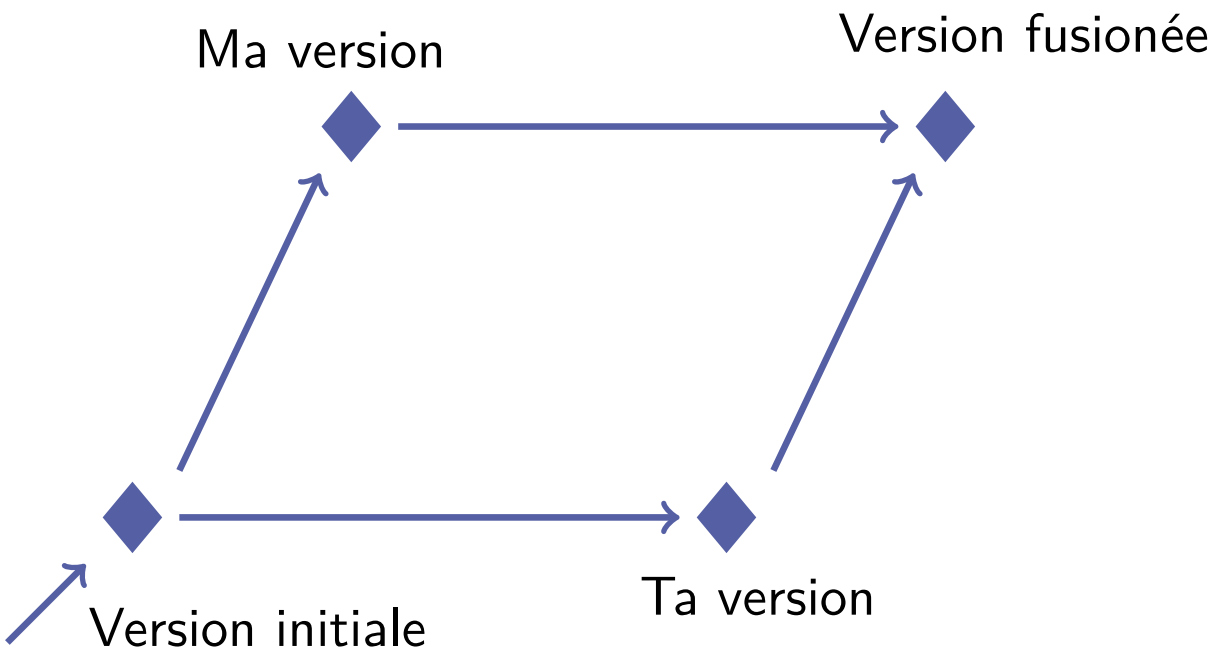
Cette fusion peut être faite pour vous de façon automatique

La fusion repose sur l'histoire !

Travail en équipe lié aux sauvegardes

Fusion

Espace de révisions possibles en 2D



Système de contrôle de version

- Garder une trace de l'histoire :
  - `commit` = image de l'état actuel,
  - Méta-données (nom d'utilisateur, date, message,...) enregistrées avec `commit`.
- Utile pour travailler en équipe (fusion).
  - Chaque utilisateur travaille sur sa propre copie,
  - L'utilisateur télécharge explicitement les modifications des autres quand il le souhaite.

Git : Basic concepts

- Chaque répertoire de travail contient :
  - Les fichiers sur lesquels vous travaillez (comme d'habitude)
  - L'histoire ou "dépôt" (dans le répertoire `.git/`)
- Opérations basiques :
  - git clone : obtenir une copie d'un dépôt existant (fichiers + histoire)
  - git commit : créer une nouvelle révision dans un répertoire
  - git pull : télécharger des révisions à partir d'un répertoire
  - git push : envoyer des révisions vers un répertoire
  - git add, git rm and git mv : dire à Git quels fichiers doivent être répertoriés
  - git status : savoir ce qui se passe
- Pour le Projet de Synthèse :
  - Chaque équipe crée un dépôt partagé sur `https://gitlab.univ-lorraine.fr/`.
  - Le répertoire `.git/` fait partie de l'évaluation

Conseils d'utilisation de Git (pour les débutants)

- N'échangez jamais de fichiers en dehors du contrôle de Git (email, clé USB, ...), sauf si vous savez vraiment ce que vous faites.
- Utilisez toujours `git commit` avec le paramètre `-a`.
- Exécutez un `git push` après chaque `git commit -a` (utilisez `git pull` si besoin).
- Exécutez `git pull` régulièrement pour rester synchroniser avec vos coéquipiers. Il faut faire un `git commit -a` avant le `git pull` (ceci afin d'éviter de mélanger des modifications manuelles avec des fusions).
- Ne faites pas de modifications inutiles à votre code. Ne laissez pas votre éditeur de reformater le code qui n'est pas écrit par vous.

Séance Machine sur GIT

- Énoncé sur arche
- À terminer en libre service pendant le premier TP
- cf. aussi `https://git-scm.com/book/fr/v2`

Proposition d'organisation des séances TP

1. GIT
2. nombres rationnels
3. listes chaînées
4. structures géométriques et algorithme glouton
5. arbres binaires de recherche
6. arbres binaires de recherche
7. algorithme Bentley-Ottmann
8. algorithme Bentley-Ottmann
9. scripts

Évaluation

- CC1 (coef. 0.3)
  - Utilisation du GIT : 5%
  - Utilisation des primitives : 5%
  - Nombres rationnels : 5%
  - Listes doublement chaînées : 10%
  - Structures géométriques : 5%
- CC3 (coef. 0.4)
  - Algorithme glouton : 5%
  - Arbres binaires de recherche : 20%
  - Algorithme Bentley-Ottmann : 15%
- CC2 (coef. 0.3)
  - Scripts & Expériences : 10%
  - Rapport : 10%
  - Soutenance : 10%

Évaluation

Il n'y aura pas de seconde chance !

Le dépôt du projet

- Dans un seul fichier compressé (`.zip`, `.rar`, `.tar.gz`, etc) :
  1. le dossier `canevas` avec votre code
     a. le dossier `.git/`
     b. le code en C (fichiers .h et .c, Makefile)
     c. les scripts BASH
  2. un rapport de maximum 5 pages (format pdf imposé)
- Un seul projet soumis par équipe
- **Date limite :** dimanche **5 janvier, 23h59**
  - pénalité d'un point par jour de retard
  - CC1 & CC3 : ABI si aucune soumission avant le 8 janvier, 23h59
  - CC2 : ABI si aucune soumission avant le 8 janvier, 23h59, et absence dans la soutenance
- **Attention** au plagiat !
  - les projets seront passer par un détecteur de plagiat de code

Soutenance

- semaine de 13 janvier
- 40 minutes par équipe
  - 15 minutes de présentation (avec des diapositives, powerpoint ou autre)
  - 15 minutes des questions sur le code
  - 10 minutes des questions générales et délibération
- tous les membres de l'équipe participeront
- différente note pour chaque membre possible
- présence uniquement pendant la soutenance de votre équipe

Consignes pour le rapport

- **5 pages maximum** (page de garde exclue)
- le rapport n'est pas une documentation du code
- introduction / conclusions
- expliquer par exemple :
  - Quels outils vous avez utilisé ?
  - Quelles fonctions importantes supplémentaires vous avez introduit et pourquoi ?
  - Quelles étaient les difficultés que vous avez rencontrées ?
  - Quelles améliorations envisageriez vous ?
- **évaluation de performance**
  - théorique – complexité (remplir et justifier les tableaux demandés au sujet)
  - Quelle structure de données fonctionne le mieux en termes de temps d'exécution pour de petits instances ? Pour de instances moyennes ? Pour de grandes instances ? Pourquoi ?

Conseils générales

- utiliser GIT correctement et régulièrement
- partager les tâches avec vos coéquipiers
- écrire des commentaires utiles
- définir des nouvelles fonctions si besoin
- travailler en dehors des heures TP
- **contacter moi rapidement** si vous avez un problème avec votre équipe
- **ne pas modifier** les prototypes des fonctions et des structures fournies

Tester – tester – tester ! ! !
chaque fonction séparément et progressivement (avant continuer)

- A titre indicatif
  - .c du projet : 1350 lignes
  - mes fonctions de test : 2150 lignes

Définir

1. fonctions de test
2. données de test

Fonctions de test – exemple listes

```
int compare_lists(struct list_t * l1, int * l2[], int size) {
    if (get_list_size(l1) != size) return 0;
    if (get_list_size(l1) != 0 && !get_list_head(l1)) return 0;
    if (!get_list_head(l1)) return 1;

    struct list_node_t * curr = get_list_head(l1);
    int i = 0;
    while (curr != NULL) {
        if (get_list_node_data(curr) != l2[i])
            return 0;
        curr = get_successor(curr);
        i++;
    }

    curr = get_list_tail(l1);
    i = size-1;
    while (curr != NULL) {
        if (get_list_node_data(curr) != l2[i])
            return 0;
        curr = get_predecessor(curr);
        i--;
    }
    return 1;
}
```

Données de test – exemple listes

```
void test_list_insert_last() {
    int * i1 = malloc(sizeof(int));
    int * i2 = malloc(sizeof(int));
    int * i3 = malloc(sizeof(int));
    *i1 = 1;
    *i2 = 2;
    *i3 = 3;

    struct list_t * L = new_list();
    int * tab[3];
    tab[0] = i1; tab[1] = i2; tab[2] = i3;

    list_insert_last(L, i1);
    if (compare_lists(L, tab, 1) == 0) printf("problème");
    list_insert_last(L, i2);
    if (compare_lists(L, tab, 2) == 0) printf("problème");
    list_insert_last(L, i3);
    if (compare_lists(L, tab, 3) == 0) printf("problème");
}
```

Questions ? ? ?