Informatique 1
L1 Portail IE
L1 Portail MA

Responsables:

pierre-alain.fouque@univ-rennes1.fr

patrick.derbez@univ-rennes1.fr

Notion de données structurées

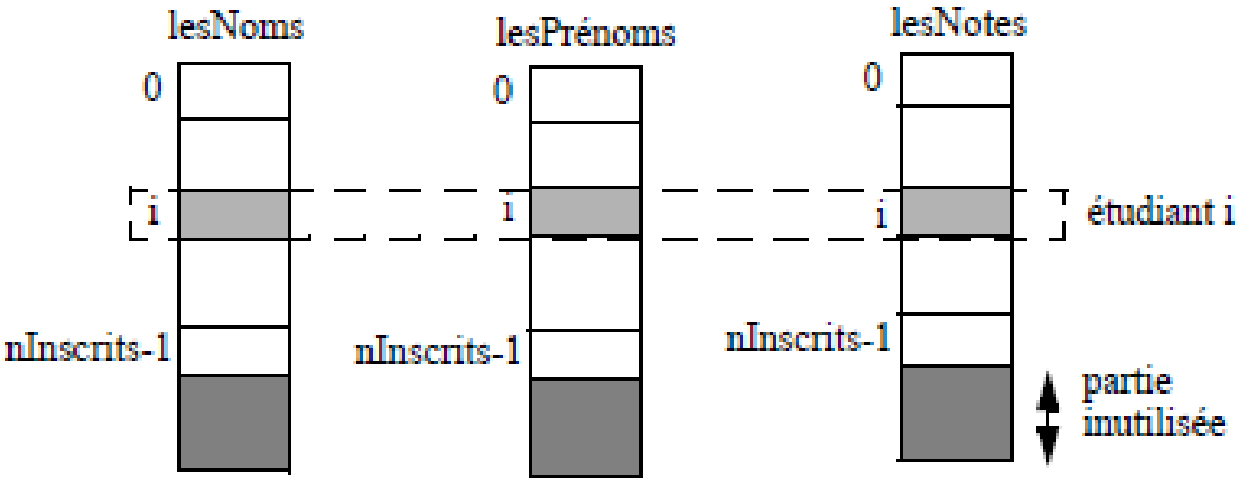
Exemple illustratif:

- On veut stocker des données représentant des étudiants et des notes. Pour chaque étudiant on a:
  - son nom
  - son prénom
  - sa note

- On veut pouvoir saisir des données représentant des étudiants et saisir les notes pour ces étudiants

Notion de données structurées

Exemple illustratif: une solution à trois tableaux



Notion de données structurées

Exemple illustratif: inconvénients

- les informations liées à un étudiants sont réparties dans plusieurs variables
- si on veut passer en paramètre les informations liées à un étudiant, cela multiplie les paramètres
- on ne peut pas rendre en résultat d'une fonction l'ensemble des informations d'un étudiants

Dans cet exemple, il y a 3 champs (nom, prénom, note).

Avec *n* champs, il faudrait *n* tableaux différents!

Notion de données structurées

- Le principe des données structurées consiste à rassembler des informations (généralement hétérogènes) dans une entité appelée **structure**.

- Exemple: cette structure **`Etudiant`** contient les données **`nom`**, **`prenom`** et **`note`**

```
// donnée structurée Inscrit
public class Etudiant {

    public String nom;
    public String prenom;
    public int note;

}
```

Notion de données structurées

```
// donnée structurée Etudiant
public class Etudiant {

    public String nom;
    public String prenom;
    public int note;

}
```

- La structure **`Etudiant`** devient un nouveau type et est utilisable comme tous les autres types (int, boolean, String,…). **`Etudiant`** est un type dit « complexe » (son passage de paramètre s'effectue par copie de référence)
- On dit que la structure **`Etudiant`** aggrège les champs **`nom`**, **`prenom`**, **`note`**.

Notion de données structurées

Avec les données structurées:

- on peut créer des variables qui représentent une donnée structurée : `Etudiant etu = new Etudiant();`
- on peut affecter des valeurs aux champs de la donnée structurée : `etu.nom = "Ferry";`
- on peut récupérer les valeurs des champs de la donnée structurée : `String nom = etu.nom;`
- on peut utiliser ce type `Etudiant` comme n'importe quel autre type

Notion de données structurées

- exemple d'utilisation:

```
public static void essai() {

  Etudiant etu = new Etudiant();  // Création d'une structure Etudiant

  etu.nom = "Ferry";  // affectation des valeurs
  etu.prenom = "Jules";
  etu.note = 17;

  afficher(etu); // appel à la fonction afficher

}
// passage d'une structure de données en paramètre
public static void afficher(Etudiant e) {
  System.out.println("Nom  =" + e.nom);
  System.out.println("Prem =" + e.prenom);
  System.out.println("Note =" + e.note);
}
```

Notion de données structurées

- exemple d'utilisation:

```
// une fonction peut retourner une (reference sur) une structure de
données
public static Etudiant creerEtudiant(String nom, String prenom) {
  Etudiant neuf = new Etudiant();
  neuf.nom = nom;
  neuf.prenom = prenom;
  return neuf;
}

// une structure de données est un type complexe
// le passage de paramètres se fait par référence.
// Donc une modification sur le paramètre formel impacte
// le paramètre effectif
public static void modifierNote(Etudiant e, int note) {
  e.note = note;
}
```

Notion de données structurées

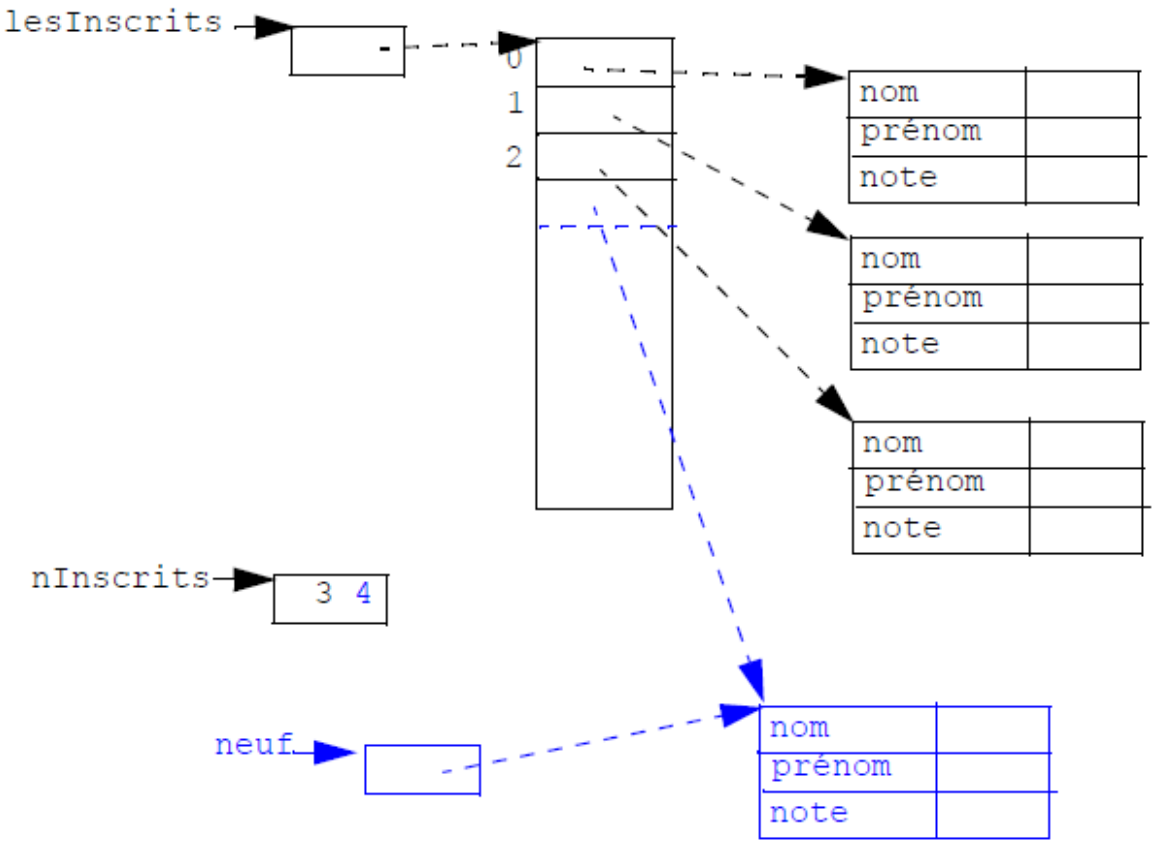
- en reprenant notre problème initial avec des données structurées, on peut créer un tableau d'Etudiants
- et ensuite effectuer des traitements sur ce tableau

```
// création d'un tableau d'étudiants
Etudiant []lesInscrits = new Etudiant [maxInscrits];
int nInscrits = 0;


// on ajoute l'étudiant dans le tableau
lesInscrits[nInscrits] = creerEtudiant("Ferry",
"Jules");
// et on incrémente le nombre d'inscrits
nInscrits++;
```

```
//  calcul de la moyenne une fois
// les données saisies
double somme = 0.0;
for (int i=0; i < nInscrits; i++) {
  somme = somme + lesInscrits[i].note;
}
moyenne = somme / nInscrits;
```

Notion de données structurées

- ajout d'un étudiant dans le tableau



Composition de données structurées

Les données structures peuvent être composées (eg une structure de structure)

- Definition d'un **Pixel**

```
// donnée structurée Pixel
public class Pixel{
        public int r; // quantité de rouge
        public int v; // quantité de vert
        public int b; // quantité de bleu
}
```

Composition de données structurées

- Défintion d'une **Image** composée d'un ensemble de **Pixels**

```
// donnée structurée Image
public class Image {
        public int hauteur; // hauteur de l'image
        public int largeur; // largeur de l'image
        public Pixel []tabPixels; // tableau de pixel

}
```

Création d'une image

- **On se donne une fonction pour créer une image à partir de sa taille hauteur x largeur**

```
// fonction de creation d'une image (blanche par défaut)
public static Image creerImage(int h, int l) {
        // on cree l'image
        Image monImage = new Image();
        // on initialise les dimensions
        monImage.largeur = l; monImage.hauteur = h;
        // on alloue en mémoire le tableau de pixels
       monImage.tabPixels = new Pixel[l*h]; // tableau de pixel
        // on crée les pixels dans le tableau
        for ( int i=0; i <l*h; i++) {
                monImage.tabPixels[i] = new Pixel();
                monImage.tabPixels[i].r = 0;
                monImage.tabPixels[i].v = 0;
                monImage.tabPixels[i].b = 0;
        }
}
```

Accès à un pixel (x,y) de l'image

- **Pour accèder à un pixel**

```
Image monImage = creerImage(320,200); // creation d'une image
Pixel p = monImage.tabPixels[0]; // accès au premier pixel
p.r = 255;                       // on met le premir pixel en rouge
monImage.tabPixels[0].r = 255;   // idem… accès à la composante r,
du tableau tabPixels, dans l'image monImage

Pixel q = monImage.tabPixels[10*monImage.largeur + 15]; // accès
au pixel à l'abscisse 10 et l'ordonnée 15
```

- **On se donne une fonction pour simplifier l'accès…**

```
// accès à un pixel
// precondition 0 <= x < largeur, 0 <= y < hauteur
public static Pixel getPixel (Image i, int x, int y) {
        return i.tabPixels[y*i.largeur + x];
}
```

Calcul du gradient d'une image en un point

- **Gradient vertical par différences finies (sur la composante rouge)**

```
int gradientVertical(Image i, int x, int y) {
        Pixel p1 = getPixel(i, x, y-1);
        Pixel p2 = getPixel(i, x, y+1);
        return p2.r – p1.r;
        // ou return getPixel(i,x,y+1).r – getPixel(i,x,y-1).r;
}
```

Calcul du gradient d'une image

- **Appliqué en tout point de l'image (sauf bordures), le gradient permet la détection des contours**

Cacher de l'information dans une image

Une image banale …

… mais un message est caché dedans!

Cacher de l'information dans une image

Idée: afficher le bit de poid faible des couleurs d'un ensemble de pixels

Complètement invisible à l'oeil nu ...

LSB

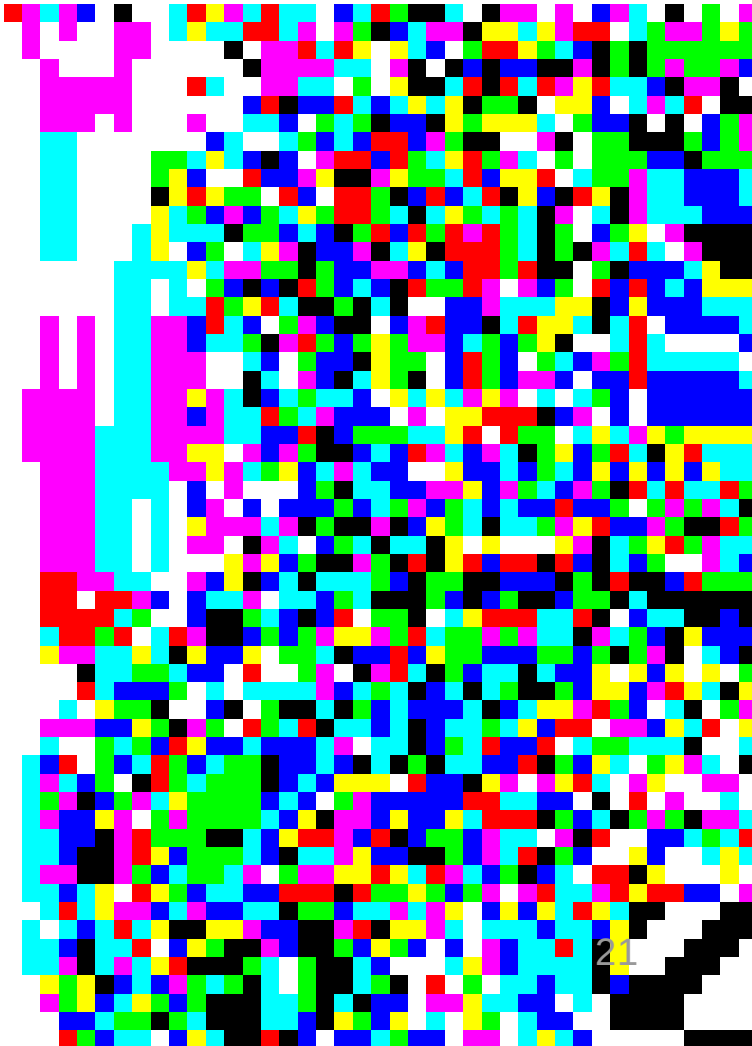
Idée: afficher le bit de poid faible des couleurs d'un ensemble de pixels

Complètement invisible à l'oeil nu ...

… mais identifiable avec les bons outils

LSB

Quelque chose se cache dans le coin supérieur gauche :)



Root-me.org - challenges de stéganographie

Conclusion

- Notions abordées en programmation impérative
  - Instructions et structures de contrôle (itératives conditionnelles)
  - Fonctions et passage de paramètres
  - Tableaux (une et deux dimensions)
  - Structures de données

- A suivre (pour L1IE):
  - Structures de données évoluées (listes chainées, files, piles, arbres)
  - Autres paradigmes de programmation (fonctionnel/objet)

Représentation d'un ensemble d'entiers

On cherche à créer une structure de données qui contient un « ensemble d'entiers » sans doublons (i.e. il n'existe pas deux entiers identiques dans la structure).

- on veut aussi les fonctionnalités suivantes:
  - récupérer le cardinal de l'ensemble (le nombre d'éléments qu'il contient)
  - rechercher si un élément donné appartient à l'ensemble
  - calculer l'union de deux ensembles E et F (l'ensemble de tous les éléments de E et tous les éléments de F)
  - calculer la différence entre l'ensemble E et l'ensemble F (l'ensemble des éléments de E sans les éléments de F)
  - calculer l'intersection de deux ensembles (ensemble des éléments qui appartiennent à la fois à E et à F)

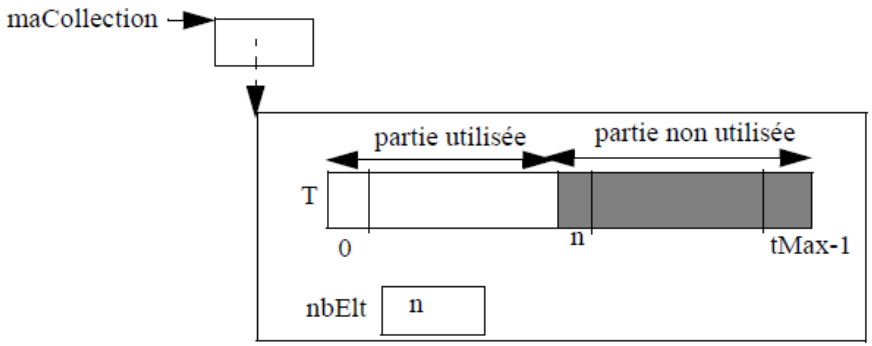
Représentation d'un ensemble d'entiers

Solution:

- on crée une structure de données qui contient un champ T de type tableau et un champ nbElt de type entier.
- le tableau est « partiellement » rempli, c'est-à-dire:
  - qu'il possède une taille tMax (nombre maximum d'entiers qu'on peut stocker dans le tableau)
  - et on connait le nombre d'éléments (nbElt) actuellement stockés dans le tableau (nbElt <= tMax).
  - au départ le tableau est vide (nbElt = 0)

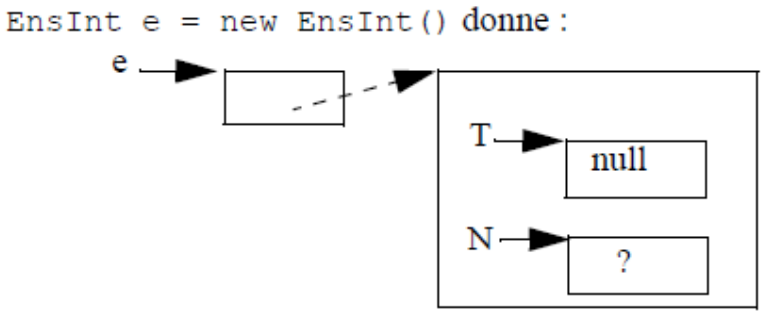
Représentation d'un ensemble d'entiers

```
// un ensemble d'entiers est une structure de données
// contenant un champ T de type tableau d'entiers
// et un champ nbElt de type entier (nb elemets du tableau)
public class EnsInt {

   public int [] T; // T[0..nbElt-1] ne contient pas de doublons
   public int nbElt;  // 0 <= nbElt <= T.length

}
...

EnsInt maCollection = new EnsInt();
```



Création d'un ensemble d'entiers vide

- dans la déclaration de notre structure de données, on ne fixe pas la taille du tableau (on ne fait que déclarer les types dont on a besoin)
- Lorsqu'on crée une instance du type EnsInt, le tableau d'entiers n'est pas initialisé (ni le nombre d'éléments)



(Il faut faire ce travail manuellement)

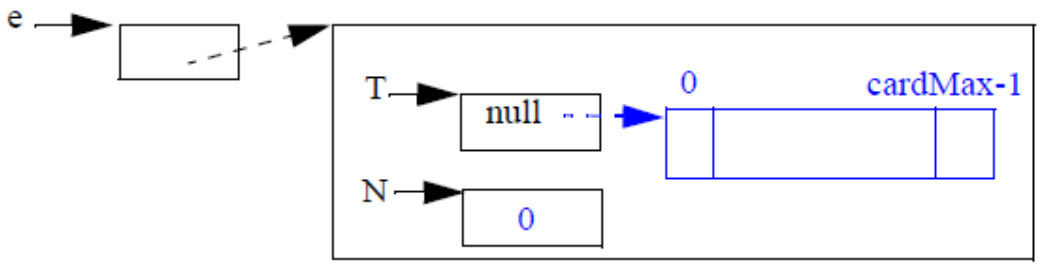
Création d'un ensemble d'entiers vide

- on se donne une fonction initEnsInt qui prend une cardinalité (le nombre max d'éléments que pourra stocker le tableau) et construit un nouvel EnsInt.

```
// construit la structure de donnée, alloue le tableau
// en mémoire, et initialise le nombre d'éléments
// effectivement stockés à 0
public static EnsInt initEnsInt (int maxCard) {
  EnsInt e = new EnsInt();
  e.T = new int [maxCard];
  e.nbElt = 0;
  return e;
}
```

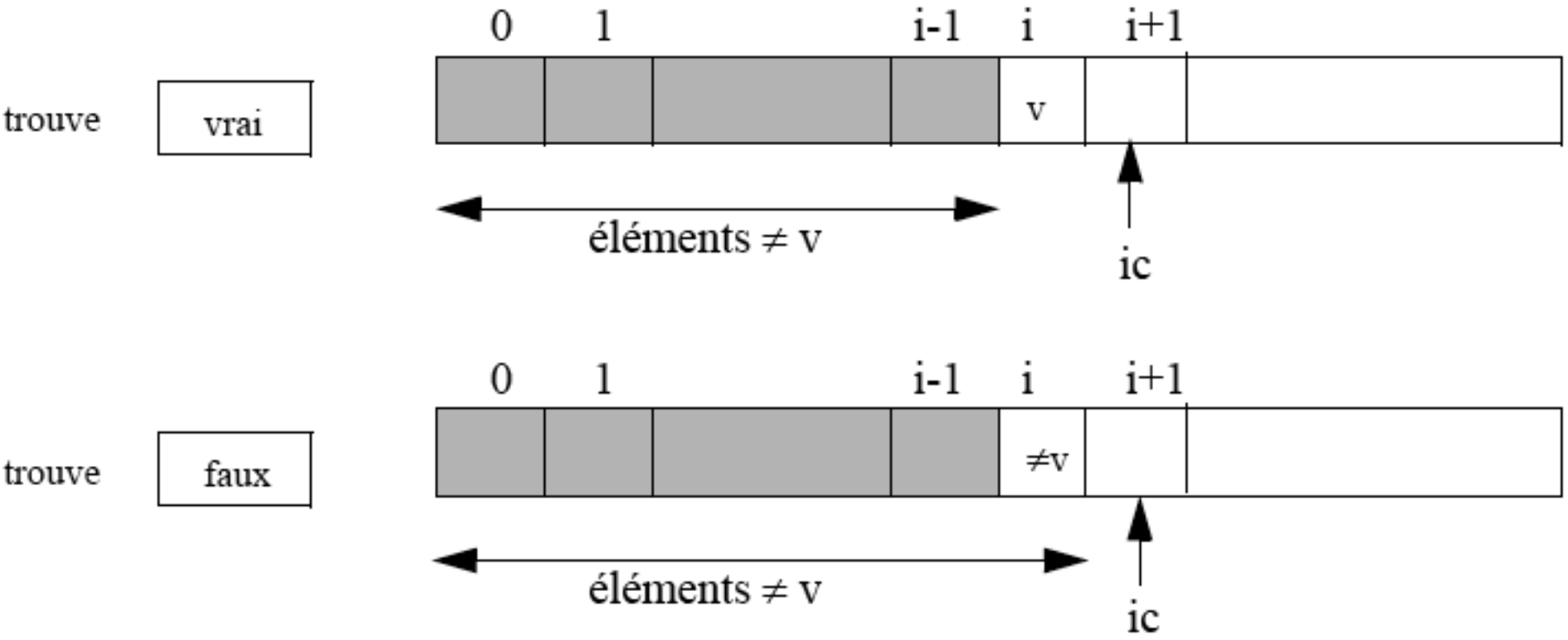
Création d'un ensemble d'entiers vide

- on se donne une fonction initEnsInt qui prend une cardinalité (le nombre max d'éléments que pourra stocker le tableau) et construit un nouvel EnsInt.



Recherche d'un élément dans un ensemble d'entiers (parcours séquentiel)

- On choisit de modéliser par un booléen ***trouve*** le fait d'avoir trouvé v.
- On a deux états possibles si on traite l'élément à l'indice i



Recherche d'un élément dans un ensemble d'entiers (parcours séquentiel)

- On choisit de modéliser par un booléen ***trouve*** le fait d'avoir trouvé v.
- On a deux états possibles si on traite l'élément d'indice i

```
public static boolean appartientA(EnsInt ensemble, int valeur) {
  boolean trouve = false;
  int i=0;
  while (i < ensemble.nbElt && !trouve) {
   trouve = (ensemble.T[i] == valeur);
   ++i;
  }
  return trouve;
}
```

Insertion d'un nouvel élément dans un ensemble d'entier

- Il faut vérifier que cet élément n'existe pas déjà.

```
public static void ajouter (EnsInt ensemble, int valeur) {
  if (!appartientA(ensemble, valeur)) {
    ensemble.T[ensemble.nbElt] = valeur;
    ensemble.nbElt = ensemble.nbElt + 1;
  }
}
```

Insertion d'un nouvel élément dans un ensemble d'entier

- Il faut vérifier que cet élément n'existe pas déjà.

```
public static void ajouter (EnsInt ensemble, int valeur) {
  if (!appartientA(ensemble, valeur)) {
    ensemble.T[ensemble.nbElt] = valeur;
    ensemble.nbElt = ensemble.nbElt + 1;
  }
}
```

- Ne pas oublier de vérifier que le tableau n'est pas plein!

```
public static void ajouter (EnsInt ensemble, int valeur) {
  if (ensemble.nbElt < ensemble.T.length && !appartientA(ensemble, valeur) )
  {
    ensemble.T[ensemble.nbElt] = valeur;
    ensemble.nbElt = ensemble.nbElt + 1;
  }
}
```

Union d'ensembles d'entiers

- Faire l'union de deux ensembles d'entiers A et B consiste à créer un nouvel ensemble d'entiers contenant tous les éléments de A et de B
- Ici, l'algorithmique est simple, nous n'analyserons pas l'itération.

```
public static EnsInt union(EnsInt A, EnsInt B) {
  // initialisation du tableau résultat
  EnsInt resultat = initEnsInt(A.T.length + B.T.length);

  // on ajoute les éléments de A dans résultat
  for (int i = 0; i < A.nbElt; i++) {
    ajouter(resultat, A.T[i]);
  }

  // on ajoute les éléments de B dans résultat
  for (int i = 0; i < B.nbElt; i++) {
    ajouter(resultat, B.T[i]);
  }

  // on retourne le tout
  return resultat;
}
```

Union d'ensembles d'entiers

- Un peu d'optimisation

```
public static EnsInt union(EnsInt A, EnsInt B) {
  // initialisation du tableau résultat
  EnsInt resultat = initEnsInt(A.T.length + B.T.length);

  // on copie A dans résultat
  for (int i = 0; i < A.nbElt; i++) {
    resultat.T[i] = A.T[i];
  }
  resultat.nbElt = A.nbElt;

  // on ajoute les éléments de B dans résultat
  for (int i = 0; i < B.nbElt; i++) {
    ajouter(resultat, B.T[i]);
  }

  // on retourne le tout
  return resultat;
}
```

Différence entre deux ensembles d'entiers

- Faire la différence entre l'ensemble d'entiers A et l'ensemble d'entiers B consiste à sélectionner tous les éléments de A qui ne sont pas dans B

```
public static EnsInt difference(EnsInt A, EnsInt B) {
  // initialisation du tableau résultat
  EnsInt resultat = initEnsInt(A.T.length);

  // on parcourt tous les éléments de A
  // si l'élément n'est dans B, on l'ajoute à résultat
  for (int i = 0; i < A.nbElt; i++) {
    if (!appartientA(B, A.T[i])) {
      ajouter(resultat, A.T[i]);
    }
  }

  // on retourne le tout
  return resultat;
}
```

Différence entre deux ensembles d'entiers

- Encore un peu d'optimisation

```
public static EnsInt difference(EnsInt A, EnsInt B) {
  // initialisation du tableau résultat
  EnsInt resultat = initEnsInt(A.T.length);

  // on parcourt tous les éléments de A
  // si l'élément n'est dans B, on l'ajoute à résultat
  for (int i = 0; i < A.nbElt; i++) {
    if (!appartientA(B, A.T[i])) {
      resultat.T[resultat.nbElt++] = A.T[i];
    }
  }

  // on retourne le tout
  return resultat;
}
```

Intersection entre deux ensembles d'entiers

- Faire l'intersection entre l'ensemble deux ensembles d'entiers A et B consiste à sélectionner tous les éléments qui sont présents à la fois dans A et dans B

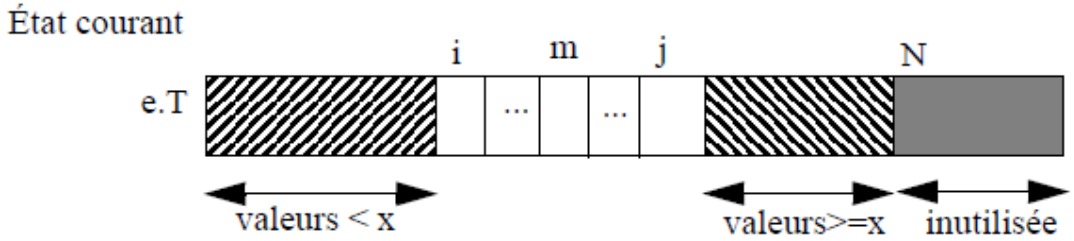
```
public static EnsInt intersection(EnsInt A, EnsInt B) {
  // initialisation du tableau résultat
  EnsInt resultat = initEnsInt(A.T.length);

  // on parcourt tous les éléments de A
  // si l'élément est aussi dans B, on l'ajoute à résultat
  for (int i = 0; i < A.nbElt; i++) {
    if (appartientA(B, A.T[i])) {
      ajouter(resultat, A.T[i]);
    }
  }

  // on retourne le tout
  return resultat;
}
```

Remarques

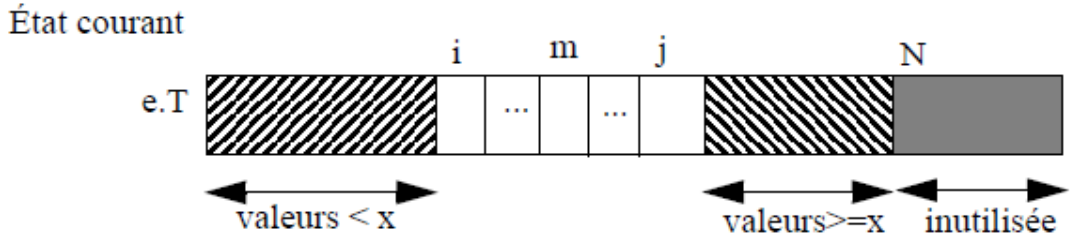
- Cas algorithmes garantissent qu'aucun élément n'est en doublon dans les tableaux résultats (parce que la fonction **ajouter** est utilisée, et celle-ci vérifie que l'élément n'est pas déjà présent).
- Et ces algorithmes impliquent souvent une recherche d'appartenance à l'ensemble d'entiers, ce qui est coûteux à la longue.
- Par exemple l'algorithme intersection effectue, dans le pire des cas:
  - A.nbElt tests d'appartenance sur B.nbElt
  - A.nbElt ajouts sur resultat.nbElt (qui dans le pire cas croit linéairement)
  - donc la complexité en considérant m=A.nbElt, n=B.nbElt est de n*m+n*(n+1)/2
  - Or, le dernier cours nous montre qu'une recherche dichotomique dans un tableau trié est de complexité log2 n (n étant le nombre d'éléments du tableau).
- Modifions les algorithmes en considérant que l'Ensemble d'entier est trié et réflechissons un peu…

Insertion dans un ensemble d'entiers trié

- Commençons par modifier la fonction **ajouter** afin d'insérer un élément en bonne position dans l'ensemble d'entiers triés en utilisant une technique dichotomique (on appelera cette fonction: **inserer**)
- Pour cela, on utilise le milieu m = (i + j)/2 à chaque itération



Insertion dans un ensemble d'entiers trié



Analyse:

- on a traité une partie du problème, c'est-à-dire qu'on doit insérer l'élément x en bonne place dans le tableau entre les indices i et j (et on sait que pour les tranches T[0..i-1] et T[j+1..N-1] il n'est pas possible d'insérer x en bonne place).
- Traiter cas courant :
- si x < e.T[m], on fait j = m-1
  - si x > e.T[m], on fait i = m+1
  - si x == e.T[m], alors x existe déjà (on ne doit pas l'insérer)

Insertion dans un ensemble d'entiers trié

- Passer au suivant
  - on calcule m = (i+j) / 2
- Condition de continuité : (i <= j) && (!trouve)
  - Tant qu'il reste des éléments à traiter (i <= j)
  - ET tant qu'on a pas encore trouvé l'élément
- Traitements finaux
  - Si (trouve) alors on ne fait pas l'insertion
  - Si (!trouve), alors on ajoute l'élément x dans le tableau à l'index i+1 (ce qui veut dire qu'on décale tous les éléments à partir de l'index i+1).

Insertion dans un ensemble d'entiers trié

```
public static void inserer (EnsInt ensemble, int x) {
  int i = 0;
  int j = ensemble.nbElt-1;
  int m = (i + j) / 2;
  boolean trouve = false;
  // on recherche si x est dans le tableau
  while (i <= j && !trouve) {
    if (x == ensemble.T[m]) {
      trouve = true;
    } else if (x < ensemble.T[m]) {
      j = m - 1;
    } else {
      i = m + 1;
    }
    m = (i + j) / 2;
  }
  // si non trouve alors on décale les élements à droite pour insérer x
  if (!trouve) {
    for (int k = ensemble.nbElt; k > i+1; k--) {
      ensemble.T[k] = ensemble.T[k-1];
    }
    ensemble.T[i+1] = x;
    ensemble.nbElt = ensemble.nbElt+1;
  }
}
```

Union d'ensembles sur des ensembles triés

- Une approche naïve peut consister à écrire:

```
public static EnsInt unionTrie(EnsInt A, EnsInt B) {
  // initialisation du tableau résultat
  EnsInt resultat = initEnsInt(2*maxCard);

  // on ajoute les éléments de A dans résultat
  for (int i = 0; i < A.nbElt; i++) {
    inserer(resultat, A.T[i]);
  }

  // on ajoute les éléments de B dans résultat
  for (int i = 0; i < B.nbElt; i++) {
    inserer(resultat, B.T[i]);
  }

  // on retourne le tout
  return resultat;
}
```

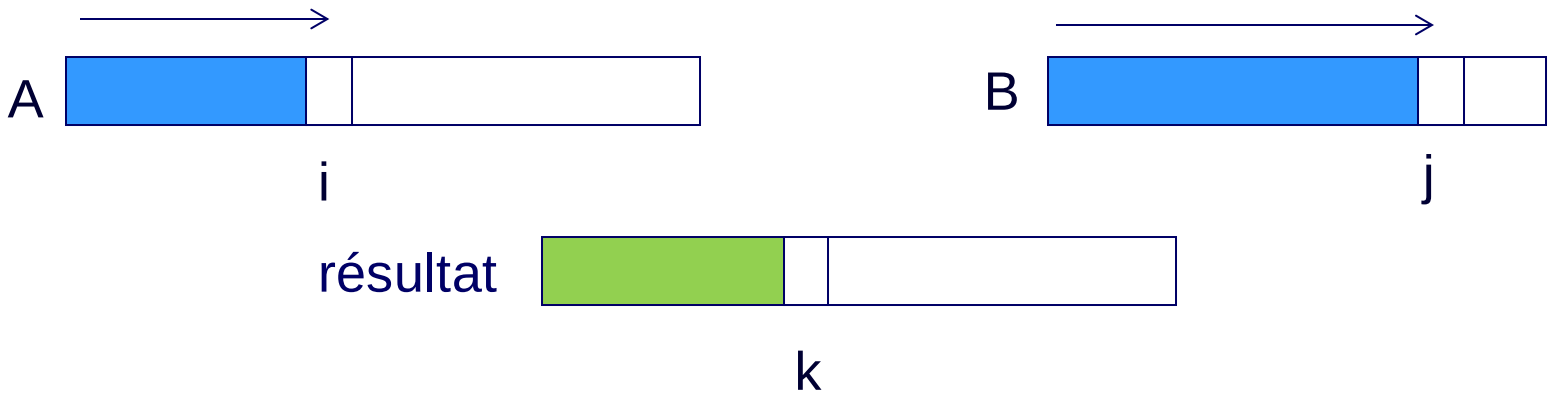
Union d'ensembles sur des ensembles triés

- Dans ce cas la complexité est de A.nbElt+B.nbElt fois le nombre d'appels à la fonction **inserer** (qui a pour complexité log2 n). Dans le pire cas, n= A.nbElt+B.nbElt. La complexité est donc de n log2 n.

- Or en réfléchissant, il est possible de parcourir les deux ensembles d'entiers en parallèle et d'ajouter les éléments au fur et à mesure dans le tableau résultat. La complexité est alors linéaire en fonction de n.

Union d'ensembles sur des ensembles triés

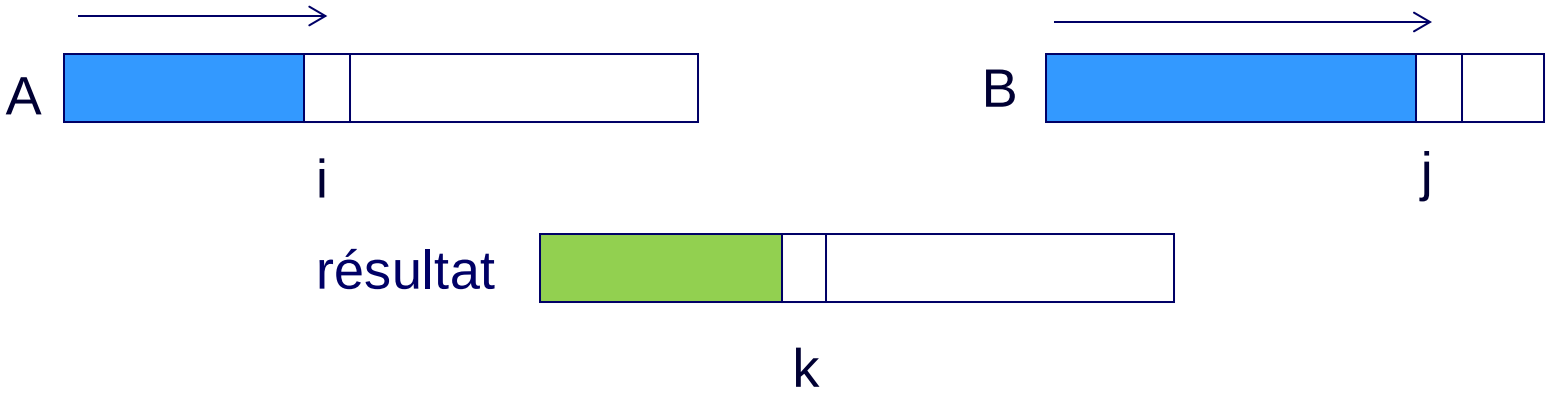
Principe:



Hypothèse: on a traité une partie du problème, c'est-à-dire qu'on a inséré tous les éléments de A.T[0..i-1] et B.T[0-j-1] dans l'ordre trié dans le tableau résultat

Union d'ensembles sur des ensembles triés

Principe:



Traiter cas courant

- si A.T[i] < B.T[j] alors résultat.T[k] = A.T[i]; i = i + 1;
- si A.T[i] > B.T[j] alors résultat.T[k] = B.T[i]; j = j + 1;
- si A.T[i] == B.T[j] alors résultat.T[k] = A.T[i]; i = i + 1; j = j + 1;

Passer au suivant

- k = k + 1;

Union d'ensembles sur des ensembles triés

Attention! Que faire lorsque i == A.nbElt ou j == B.nbElt ?

Il faut ajouter ces cas dans l'itérative!

- si (j == B.nbElt) || (j < B.nbElt && A.T[i] < B.T[j])
  alors résultat.T[k] = A.T[i]; i = i + 1;
- si (i == A.nbElt) || (i < A.nbElt && A.T[i] > B.T[j])
  alors résultat.T[k] = B.T[i]; j = j + 1;
- si (i < A.nbElt) && (j < B.nbElt) && A.T[i] == B.T[j]
  - alors résultat.T[k] = A.T[i]; i = i + 1; j = j + 1;

Passer au suivant

- k = k + 1;

Condition de continuté

- (i < A.nbElt) || (j < B.nbElt)

Union d'ensembles sur des ensembles triés

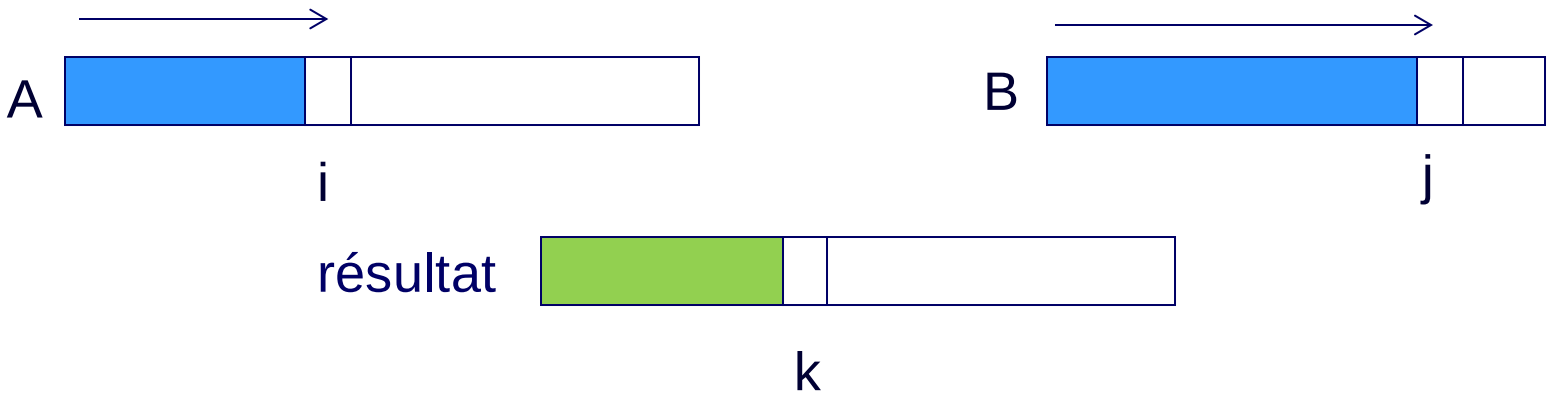
```
public static EnsInt unionTrieLineaire(EnsInt A, EnsInt B) {
  // initialisation du tableau résultat
  EnsInt resultat = initEnsInt(2*maxCard);

  int i = 0;
  int j = 0;
  int k = 0;
  while (i < A.nbElt || j < B.nbElt) {
    if (j == B.nbElt || j < B.nbElt && A.T[i] < B.T[j] ) { // si on avance sur A
      resultat.T[k] = A.T[i];
      i = i + 1;
    }
    else if (i == A.nbElt || i < B.nbElt && A.T[i] > B.T[j]) { // si on avance sur B
      resultat.T[k] = B.T[j];
      j = j + 1;
    }
    else { // si on avance sur A et B
      resultat.T[k] = A.T[i];
      i = i + 1;
      j = j + 1;
    }
    k = k + 1;        // passer au suivant
  }
  resultat.nbElt = k; // mise à jour du nombre d'éléments
  return resultat;
}
```

Union d'ensembles sur des ensembles triés

```
public static EnsInt unionTrieLineaire(EnsInt A, EnsInt B) {
  // initialisation du tableau résultat
  EnsInt resultat = initEnsInt(2*maxCard);

  int i = 0;
  int j = 0;
  int k = 0;
  while (i < A.nbElt && j < B.nbElt) {
    if (A.T[i] <= B.T[j] ) { // si on avance sur A
      resultat.T[k] = A.T[i];
      if (A.T[i] == B.T[j]) j = j+1; // si on avance aussi sur B
      i = i + 1;
    }
    else { // si on avance sur B
      resultat.T[k] = B.T[j];
      j = j + 1;
    }
    k = k + 1;        // passer au suivant
  }
  for (; i < A.nbElt; ++i) resultat.T[k++] = A.T[i];
  for (; j < B.nbElt; ++j) resultat.T[k++] = B.T[j];
  resultat.nbElt = k; // mise à jour du nombre d'éléments
  return resultat;
}
```

Différence entre deux ensembles triés

On suit une procédure similaire pour la différence entre l'ensemble d'entiers triés A et l'ensemble d'entiers triés B



Hypothèse: on a traité une partie du problème, c'est-à-dire qu'on a ajouté dans le tableau résultat tous les éléments A.T[0..i-1] sans ceux qui apparaissaient aussi dans B.T[0..j-1]

Différence entre deux ensembles triés

On suit exactement le même raisonnement:

Traiter cas courant

- si A.T[i] < B.T[j] alors résultat.T[k] = A.T[i]; i = i + 1; k = k + 1;
- si A.T[i] > B.T[j] alors j = j + 1;
- si A.T[i] == B.T[j] alors i = i + 1; j = j + 1;

Passer au suivant

Condition de continuté

- (i < A.nbElt)

Différence entre deux ensembles triés

Attention! Que faire lorsque j == B.nbElt?

On ajoute tous les éléments restants de A! L'itérative devient donc

Traiter cas courant

- si (j < B.nbElt) alors
  si A.T[i] < B.T[j] alors résultat.T[k] = A.T[i]; i = i + 1; k = k + 1;
  si A.T[i] > B.T[j] alors j = j + 1;
  si A.T[i] == B.T[j] alors i = i + 1; j = j + 1;
- sinon
  résultat.T[k] = A.T[i]; i = i + 1; k = k + 1;

Condition de continuté

- (i < A.nbElt)

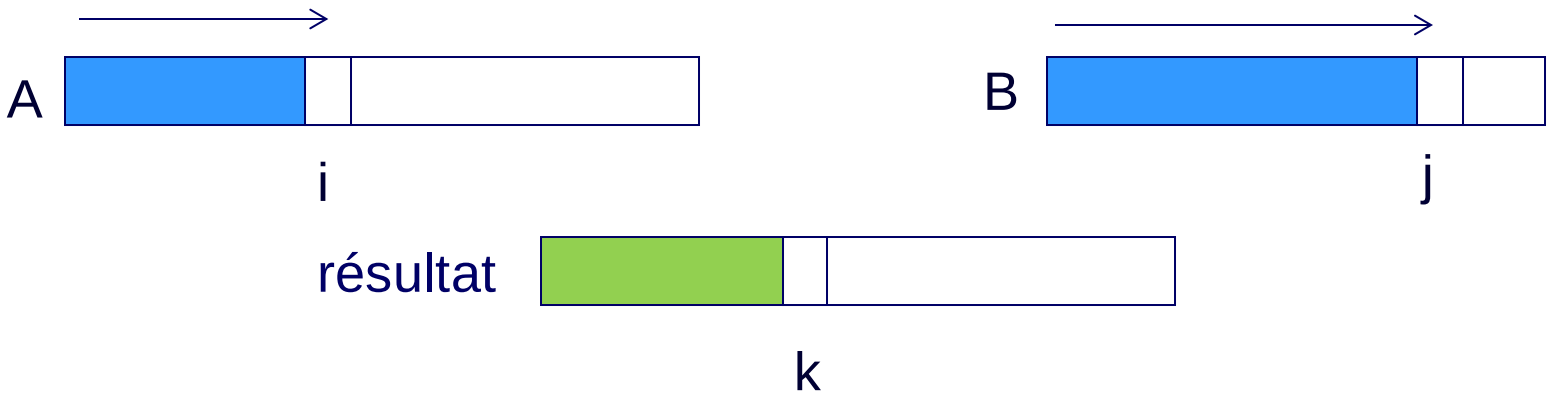
Différence entre deux ensembles triés

En complexité linéaire…

```
public static EnsInt differenceTrie(EnsInt A, EnsInt B) {
  // initialisation du tableau résultat
  EnsInt resultat = initEnsInt(maxCard);

  int i = 0;
  int j = 0;
  int k = 0;
  while (i < A.nbElt && j < B.nbElt) {
      if (A.T[i] < B.T[j]) {      // on a trouvé un élément de A pas dans B
          resultat.T[k] = A.T[i]; i = i + 1; k = k + 1;
       }
      else {          // on ne sait pas si A est dans B, on avance dans B
          if (A.T[i] == B.T[j]) i = i + 1;             // on a trouvé un élément de A dans B,
on ne l'ajoute pas
          j = j + 1;
       }
  }
  for (; i < A.nbElt; ++i) {                // il ne reste plus d'elements dans B, on recopie A
      resultat.T[k++] = A.T[i];
  }
  resultat.nbElt = k; return resultat;
}
```

Intersection entre deux ensembles triés

On suit une procédure similaire pour l'intersection entre deux ensembles d'entiers triés A et B



Hypothèse: on a traité une partie du problème, c'est-à-dire qu'on a ajouté dans le tableau résultat tous les éléments A.T[0..i-1] qui apparaissaient aussi dans B.T[0-j-1]

Intersection entre deux ensembles triés

On suit exactement le même raisonnement:

Traiter cas courant

- si A.T[i] < B.T[j] alors i = i + 1; // on ne trouvera pas A.T[i] dans B
- si A.T[i] > B.T[j] alors j = j + 1; // on doit chercher A.T[i] dans B

// si on a le même entier dans les deux listes, on l'ajoute au resultat

- si A.T[i] == B.T[j] alors résultat.T[k] = A.T[i]; i = i + 1; j = j + 1; k = k + 1;

Passer au suivant

Condition de continuité

- (i < A.nbElt & j < B.nbElt) // tant qu'on peut parcourir les deux tableaux

Intersection entre deux ensembles triés

```
public static EnsInt intersectionTrie(EnsInt A, EnsInt B) {
  // initialisation du tableau résultat
  EnsInt resultat = initEnsInt(maxCard);

  int i = 0;
  int j = 0;
  int k = 0;
  while (i < A.nbElt && j < B.nbElt) {
    if (A.T[i] < B.T[j]) {   // on ne trouvera pas A.T[i] dans B
      i = i + 1;
    }
    else if (A.T[i] > B.T[j]) {  // on doit chercher A.T[i] dans B
      j = j + 1;
    }
    else { // on a trouvé le meme entier dans A et B
      resultat.T[k] = A.T[i];
      i = i + 1;
      j = j + 1;
      k = k + 1;
    }
  }
  resultat.nbElt = k;
  return resultat;
}
```