ⵜⴰⵙⴷⴰⵡⵉⵜ ⵙⵉⴷⵉ ⵎⵓⵃⴰⵎⵎⴷ ⴱⵏ ⵄⴱⴷⵍⵍⴰⵀ
ⵜⴰⵏⵣⵍⵎⵜ ⵜⴰⵎⴰⵙⵙⵙⵜ ⴷⵀⵔ ⵍⵎⵀⵔⴰⵣ
جامعة سيدي محمد بن عبدالله
كلية العلوم ظهر المهراز
Université Sidi Mohamed Ben Abdellah
Faculté des Sciences Dhar Mahraz

Parallel Computing

Algorithme de sélection séquentiel

BDSAS

Réalisé par : JOUAL Anouar
SADIQI Omar

Introduction :

En algorithmique, un **algorithme de sélection** est une méthode ayant pour but de trouver le *k*-ième plus petit élément d'un ensemble d'objets (étant donné un ordre et un entier *k*).

La question de la sélection est un problème essentiel en algorithmique, notamment dans la recherche du maximum, du minimum et de la médiane. Plusieurs algorithmes ont été proposés et plusieurs contextes ont été étudiés : algorithmes en ligne, complexité amortie, complexité en moyenne, ensemble d'objet particuliers etc.

Le problème de la sélection est très lié aux algorithmes de tri : l'un des algorithmes classiques, Quickselect, utilise d'ailleurs le même principe que l'algorithme de tri Quicksort.

Problématique:

le problème est le suivant : étant donné un ensemble de *n* objets, un ordre sur ces objets, et un entier *k* inférieur à *n*, trouver l'objet qui est strictement plus grand qu'exactement *k* objets.

Algorithme :

Un algorithme simple consiste à commencer par trier les objets, puis de trouver le *k*-ième élément. Ceci peut-être fait avec une complexité de $O(nlog(n))$dans le pire cas, du fait de la complexité des algorithmes de tri.

On peut cependant arriver à une complexité en moyenne linéaire et même à une complexité linéaire dans le pire cas avec l'algorithme médiane des médianes[1,2].

- Complexité :

Ces algorithmes sont au centre de questions importantes en complexité algorithmique. Ils sont aussi très importants de par leurs vastes domaines d'application.

Méthode Allsums : Fonctionnalité

- Définition général :

Etant donné une suite de nombres réels n A = a1, a2, ...,an , et un entier positif k, le Problème de sélection Sum est de trouver le segment A (i *, j *) = ai * , ai + 1, ..., aj , tels que le rang de la somme : (i *, j *) = Σt = i * j * à k est sur toutes (n-1) 2 segments. Nous présentons un algorithme déterministe pour ce problème qui fonctionne en O (nlogn) temps. Le résultat précédemment connu pour ce problème est un algorithme aléatoire qui fonctionne en O (nlogn) heure prévue.

L'application de cet algorithme, nous pouvons obtenir un algorithme déterministe pour le k Sums Maximum problème, à savoir, le problème de l'énumération des k plus importants segments de somme, qui fonctionne en O (nlogn + k) temps. Les algorithmes randomisés et déterministes précédemment les plus connus pour le problème k des montants maximaux fonctionnent respectivement en O attendu (nlogn + k) temps et dans le pire des cas O (log2 + n) temps.

L’implémentation des algorithmes de sélection séquentiels :

```
{ Recherche de la presence d'une valeur entiere
dans un tableau d'entiers trie
par ordre croissant }
```

- **Methode sequentielle**

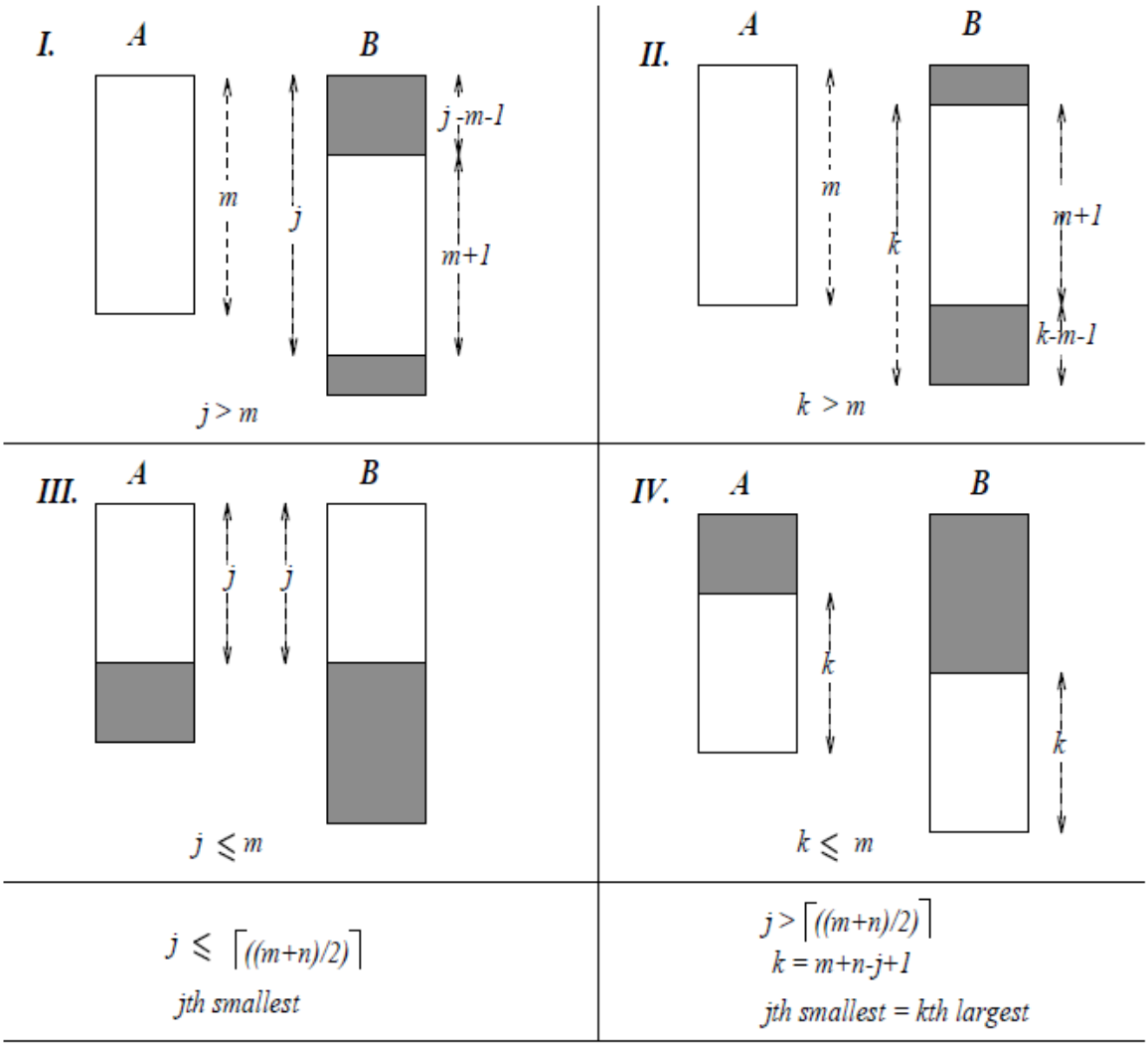
```
constante entier N <- ...

booleen fonction estPresent(v,t)
  Donnees
    v : entier
    t : Tableau[N] de entier
  Locales
    res : booleen
    run : booleen
    i : entier
  run <- vrai
  i <- 0
  tantque run = vrai faire
    si i = N alors
      run <- faux
      sinon
      si t[i] >= v alors
        run <- faux
        sinon
        i <- i+1
      fsi
    fsi
  fait
  res <- faux
  si i < N alors
    si t[i] = v alors
      res = vrai
    fsi
  fsi
  retourner res
fin fonction
```

```
/* Recherche sequentielle de la presence          */
/* d'un int dans un tableau de int trie           */


static boolean estPresent(int v,int [] t) {
  boolean run;
  boolean res;
  int i;
  run = true;
  i = 0;
  while ( run ) {
    if ( i == t.length )
      run = false;
      else  {
      if ( t[i] >= v ) {
        run = false; }
        else {
        i = i+1; } } }
  res = false;
  if ( i < t.length ) {
    if ( t[i] == v ) {
      res = true; } }
  return res;
}
```

- Version optimisée

```
/* Recherche sequentielle de la presence          */
/* d'un int dans un tableau de int trie           */


static boolean estPresent2(int v,int [] t) {
  int i = 0;
  while ( ( i != t.length ) && ( t[i] < v ) ) {
    i = i+1; }
  return ( ( i < t.length ) && ( t[i] == v ) );
```

Schéma de recherche du médiane des éléments :



La médiane des médianes :

La **médiane des médianes** est un algorithme de sélection, c'est-à-dire un algorithme pour trouver un élément dans une liste.

Bien que le tri rapide soit sous-quadratique en temps, en moyenne, il peut exiger un temps quadratique lorsque le choix du pivot est mauvais (prenons le cas d'un pivot égal au plus petit élément à chaque étape).
La solution pour le rendre O(n log(n)) dans le pire des cas est de toujours trouver des « bons » pivots. Un bon pivot est celui pour lequel on peut établir qu'une proportion constante d'éléments sont situés en dessous du pivot (respectivement au-dessus).

L'algorithme est en 3 étapes :

- L'algorithme divise la liste en groupes de cinq éléments. Ensuite, pour chaque groupe de cinq, la médiane est calculée (une opération qui peut s'effectuer en temps constant).
- L'algorithme est alors appelé récursivement sur cette sous-liste de $N/5$ éléments pour trouver la vraie médiane de ces éléments.
- Enfin, la médiane des médianes est choisie pour être le pivot. Selon la position de l'élément recherché, l'algorithme recommence avec les éléments au-dessus du pivot ou en dessous.