IFT1015 Programmation 1

Algorithms avec les tableaux

Matériel produit par Marc Feeley et Aaron Courville

Revoir: Tableaux

- Un tableau peut être construit en énumérant tous ces éléments entre « **[** » et « **]** »
- L'accès à un élément, l'**indexation**, se fait avec la syntaxe : *‹expression›* **[***‹expression›***]**

```
var couleurs = ["rouge", "vert", "bleu"];

var lotto649 = [34, 10, 23, 1, 49, 22];

var noel = [2012, 12, 25];

print( couleurs[1] ); // imprime vert
couleurs[1] = "jaune";
print( couleurs[1] ); // imprime jaune
```

Traitements de tableaux

- Traitements sur les tableaux que nous avons déjà vu:
  - imprimer tous les éléments d'un tableau
  - renverser le contenu d'un tableau
  - trouver la position d'une valeur dans un tableau
  - imprimer la valeur minimale d'un tableau

Autres traitements de tableaux

- Autres traitements importants sur les tableaux :
  - shuffler (mélanger un tableau)
  - insertion/enlèvement d'un élément d'un tableau
  - trier un tableau par ordre croissant
  - recherche d'un élément dans un tableau trié (en ordre croissant)
- Cette semaine: nous allons nous concentrer sur des algorithmes pour effectuer ces traitements.

Shuffler (mélanger) un tableau

```
/* echange t[from] et t[to] aucune verification que les indices sont valides */
var swap = function(t,from,to) {
    ++swap.calls;
    var temp = t[from];
    t[from] = t[to];
    t[to] = temp;
};

// shuffle d'un tableau t
var shuffle = function (t) {
    if (t && t.length) {
       for (var i=0; i<t.length; ++i) {
         var j = Math.floor(Math.random() * t.length);
         if (i !== j) swap(t,i,j);
       }
    }
    return t;
};

var test_shuffle = function() {
   var tab = [];
   for (var i = 0; i < 20; ++i) tab[i] = i;
   print("longeur: ", tab.length, ",  tab: " , tab);
   shuffle(tab);
   print("longeur: ", tab.length, ",  tab: " , tab);
};
test_shuffle();
```

Insertion d'un élément

```
/* ajout (en place) d'une valeur val dans un tableau t Ã  l'indice ind
 * si ind n'est pas specifie, ajout en fin, ex: insert([1,3,5,7],4,2) -> [1,3,4,5,7] */
var insert = function (t, val, ind) {
    if (ind === undefined) return t.push(val); // ajout en fin
    if ((ind >= 0) && (ind <= t.length)) {
      var i;
      for (i = t.length; i> ind; --i) t[i] = t[i-1];  t[i] = val;
    }
    return t;
};

/* egalite de deux tableaux
 * egalite des longueurs et egalite 2 a 2 des valeurs */
var egal = function(t1, t2) {
    if (t1.length !== t2.length) return false;
    for (var i = 0; i<t1.length; ++i)
    if (t1[i] != t2[i])
        return false;
    return true;
};

var test_insert = function () {
    var tab1 = [11, 22, 33, 44, 3, 4, 6, 8, 9, 10, 2, 7];
    insert(tab1,7,2);
    assert( egal(tab1,[11,22,7,33,44,3,4,6,8,9,10,2,7]), "echec insert (1)");
    insert(tab1,7,0);
    assert( egal(tab1,[7,11,22,7,33,44,3,4,6,8,9,10,2,7]), "echec insert (2)");
    insert(tab1, 7);
    assert( egal(tab1,[7,11,22,7,33,44,3,4,6,8,9,10,2,7,7]), "echec insert (3)");
    insert(tab1,8,15);
    assert( egal(tab1,[7,11,22,7,33,44,3,4,6,8,9,10,2,7,7,8]), "echec insert (4)");
};

test_insert();
```

Enlèvement d'un élément

```
/* retrait (en place) de nb valeurs (default 1) Ã  l'indice ind
 * ex: retrait([1,2,2,2,3,4,5],1,2) -> [1,2,3,4,5] */
var remove = function (t, ind, nb) {
    var n = nb || 1;
    if (ind < t.length) {
        for (var i=ind+n; i<t.length; ++i) t[i-n] = t[i];
        t.length -= Math.min(t.length-ind,n);
    }
    return t;
};

/* elimine les repetitions successives dans un tableau t
 * t est modifie (en place) */
var elimine_repetition = function(t) {
    for (var i = 0; i < t.length-1; ++i) {
      var nb = 0;
      for (var j = i+1; (j < t.length) && (t[j] == t[i]); ++j) nb++;
      if (nb) remove(t,i,nb);
    }
};

var test_remove = function() {
    var tab = [1,2,2,2,3,4,5];
    remove(tab,1,2);
    assert( egal(tab, [1,2,3,4,5]), "echec remove (1)"); // besoin fonction egal qui est pas ici.
    remove(tab,0);
    assert( egal(tab, [2,3,4,5]), "echec remove (2)");   // besoin fonction egal qui est pas ici.
};

var test_elimine_repetition = function() {
    var t = [1,2,2,2,3,4,4,5,6,7,7,6,6,6,6,6,6];
    elimine_repetition(t);
    assert (egal(t,[1,2,3,4,5,6,7,6]), "echec elimine_repetition");
};

test_remove();
test_elimine_repetition();
```

Copie un tableau

```
/* retourne une copie du tableau t */
var copie = function(t,deb,fin) {

    if (deb === undefined) deb = 0;
    if (fin === undefined) fin = t.length-1;

    var c = new Array(fin-deb+1);
    for (var i=0; i<c.length; i++) c[i] = t[i+deb];
    return c;
};

/* egalite de deux tableaux
 * egalite des longueurs et egalite 2 a 2 des valeurs */
var egal = function(t1, t2) {
    if (t1.length !== t2.length) return false;
    for (var i = 0; i<t1.length; ++i)
   if (t1[i] != t2[i])
       return false;
    return true;
};

var test_copie = function() {
   assert(egal(copie([1,2,3,4,6]),[1,2,3,4,6]), "test copie (1) rate");
};

test_copie();
```

Trier un tableau

- Beaucoup d'algorithms (nous allons voir certains d'entre eux):
  - tri par sélection
  - tri par insertion
  - tri à bulle

  } ceux-ci, nous verrons cette semaine
  (ils sont lent mais simple de comprendre et coder)

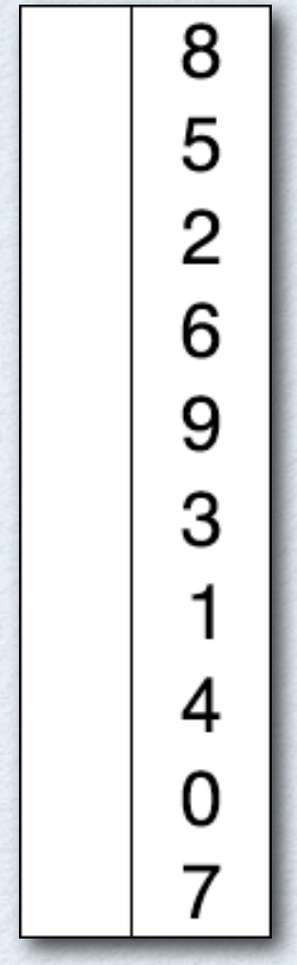
  - tri rapid
  - tri par fusion

  } ceux-ci, nous verrons dans quelques semaine
  (ils sont plus performant)

- Les performances des algorithmes ne sont pas tous pareils.

Tri par sélection

- Parcourt le tableau à trier du début à la fin.
- Tri par sélection fonctionne en choisissant l'élément minimum et le placer à la tête de la partie non triés du tableau.
- À chaque itération, de la partie non triée du tableau se contracte par un seul élément.



Tri par sélection

```
// Procédure qui trie un tableau en ordre croissant in-situ

var trier = function (t) {  // tri par sélection
    for (var i=0; i<t.length-1; i++) {
        var m = positionMin(t, i);
        var temp = t[i];
        t[i] = t[m];
        t[m] = temp;
    }
};

var positionMin = function (t, debut) {
    // suppose que t.length > debut
    var posMin = debut;
    for (var i=debut+1; i<t.length; i++) {
        if (t[i] < t[posMin]) {
            posMin = i;
        }
    }
    return posMin;
};

var tab1 = [5, 8, 3, 7];
var tab2 = ["poire", "orange", "pomme"];
trier(tab1);  print(tab1);  // imprime : 3,5,7,8
trier(tab2);  print(tab2);  // imprime : orange,poire,pomme
```

Tri par insertion

- Parcourt le tableau à trier du début à la fin.
- Au *i*-ème élément: (les éléments qui le précèdent sont déjà triés) insérer le *i*-ème élément à sa place parmi ceux qui précèdent.
- Puis décaler les éléments afin de pouvoir effectuer l'insertion.
- En pratique, ces deux actions sont fréquemment effectuées en une passe, qui consiste à faire « remonter » l'élément au fur et à mesure jusqu'à rencontrer un élément plus petit.

Le tri par insertion est cependant considéré comme le tri le plus efficace sur des entrées de petite taille.

Tri par insertion

```
/* tri (ordre croissant) par insertion */

var triInsertion = function(t) {

   if (t && t.length) {
      var memo,j;
      for (var i=1; i<t.length; ++i) {
         memo = t[i];
         for (j=i-1; (j >= 0) && (t[j] > memo); --j) {
            t[j+1] = t[j];
         }
         t[j+1] = memo;
      }
   }
   return t;
};

var tab1 = [5, 8, 3, 7];
var tab2 = ["poire", "orange", "pomme"];
triInsertion(tab1);  print(tab1);  // imprime : 3,5,7,8
triInsertion(tab2);  print(tab2);  // imprime : orange,poire,pomme
```

Tri à bulle

- Parcourt le tableau à trier du début à la fin.
- Tri à bulle fonctionne en comparant les éléments voisins (en ordre) et commuter leurs valeurs si elles sont dans le mauvais ordre.
- L'algorithm continue jusqu'à ce que (boucle `do...while`) on pass par tous le tableau sans commuter les voisins.

Tri à bulle

```
// Procédure qui trie un tableau en ordre croissant in-situ

var trier = function (t) { // tri bulle

    do {
        var echange = false;

        for (var i=0; i<t.length-1; i++) {
            if (t[i+1] < t[i]) {
                var temp = t[i];
                t[i] = t[i+1];
                t[i+1] = temp;
                echange = true;
            }
        }
    } while (echange);

};

var tab1 = [5, 8, 3, 7];
var tab2 = ["poire", "orange", "pomme"];

trier(tab1);  print(tab1);  // imprime : 3,5,7,8
trier(tab2);  print(tab2);  // imprime : orange,poire,pomme
```

Test: quel algo?

6 5 3 1 8 7 2 4 ⟵ Tri à bulle

Tri par insertion ⟶

Recherche dans un tableau trié

- Il y a deux façon de chercher dans un tableau pour un element en particulier:
  - Recherche linéaire
    - lent pour les long tableau (linéaire en les number d'element)
    - généralement applicable
  - Recherche dichotomique (binary search)
    - très vite pour les grands tableaux
    - efficacité dépend de la présence d'un tableau trié

Recherche linéaire

- Idée simple: Parcourt le tableau du début à la fin en comparant chaque élément avec la valeur cible. Arrêter, lorsque la valeur se trouve.

Recherche linéaire

```
var to_insert = function(i) { return -i-1; }; // voir convention

/* linearSearch
 * recherche lineaire (de la gauche vers la droite) de val dans t
 * return l'indice de val dans t si val existe
 *        et l'opposÃ© de l'indice -1 sinon                    */
 var linearSearch = function(t,val) {
    if (t && t.length) {
      var i=0;
      while ((i<t.length) && (t[i] < val)) i++;
      return ((i<t.length) && (t[i] === val))? i : to_insert(i);
    }
    return to_insert(0); // tableau vide
};

/* genere un nombre entier entre from (defaut: 0) et to (defaut 10) inclus */
var alea = function(from, to) {
    var deb = from || 0;
    var fin = to || 10;
    return deb + Math.floor(Math.random() * (fin-deb+1));
};

var test1 = function(search) {
   // tableau trie en ordre croissant
   var t = new Array(alea(5,10));
   t[0] = alea(0,3);
   for (var i=1; i<t.length; ++i) t[i] = t[i-1] + alea(0,3);

   for (i=0; i<10; ++i) {
      var val = alea(1,t.length*2);
      var res = search(t,val);
      if (res >=0) print (val , " se trouve dans ", t, " a l'indice ", res);
      else         print (val, " devrait etre insere dans " , t , " a l'indice ", to_insert(res));
   }
}; // test1
test1(linearSearch);
```

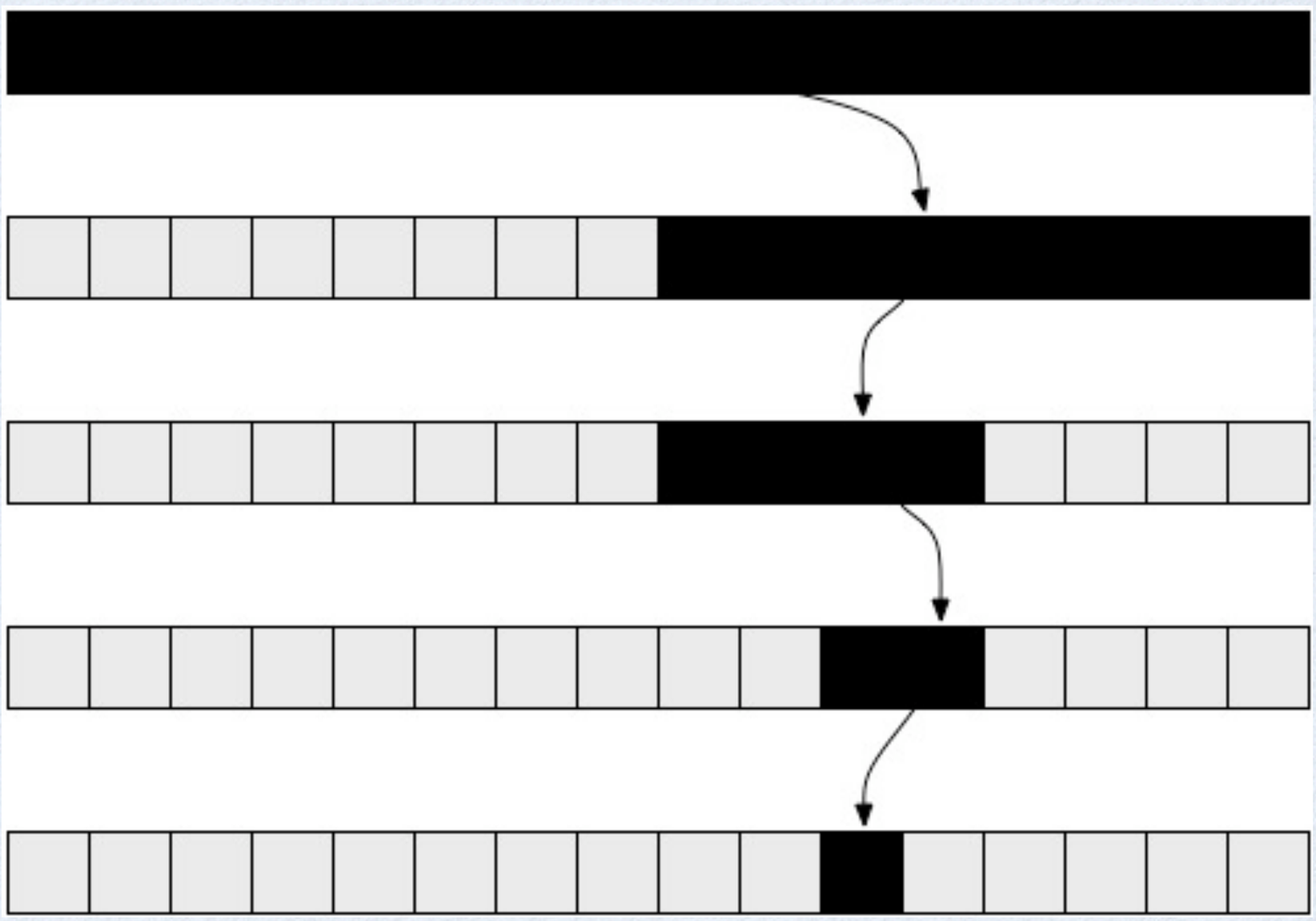
Recherche dichotomique

Algorithm:

- Commence avec un tableau trié (en order croissant).
- À chaque étape, on coupe le tableau en deux parties (pas forcément égales).
- Si le valeur cible est plus petit que le valeur au point de partage, continue le recherche à la première partie.
- Si le valeur cible est plus grand que le valeur au point de partage, continue le recherche à la deuxième partie.
- Répéter sur la partie choisie

Recherche dichotomique

Algorithm en image:



Recherche dichotomique

```
var to_insert = function(i) { return -i-1; }; // voir convention

/* binarySearch
 * recherche dichotomique  */
var binarySearch = function(t, val) {
    var deb = 0;
    var fin = t.length-1;
    while (deb <= fin) {
     var mil = deb + Math.floor((fin-deb) / 2);
     if (t[mil] == val) return mil;
     if (t[mil] > val) fin = mil-1;
     else              deb = mil+1;
    } // deb <= fin
    return to_insert(deb); // place ou inserer
};
/* genere un nombre entier entre from (defaut: 0) et to (defaut 10) inclus */
var alea = function(from, to) {
    var deb = from || 0;
    var fin = to || 10;
    return deb + Math.floor(Math.random() * (fin-deb+1));
};
var test1 = function(search) {
   // tableau trie en ordre croissant
   var t = new Array(alea(5,10));
   t[0] = alea(0,3);
   for (var i=1; i<t.length; ++i) t[i] = t[i-1] + alea(0,3);
   for (i=0; i<10; ++i) {
      var val = alea(1,t.length*2);
      var res = search(t,val);
      if (res >=0) print (val , " se trouve dans ", t, " a l'indice ", res);
      else         print (val, " devrait etre insere dans " , t , " a l'indice ", to_insert(res));
   }
}; // test1
test1(binarySearch);
```