Algorithmique & programmation

Chapitre 2 : Vecteurs

Algorithmes de mise à jour

Insertion

Mise à jour

- Deux opérations
  - Insertion
  - Suppression
- Deux types de vecteurs
  - Vecteur non trié
  - Vecteur trié
- Note : on supposera toujours que **n < nmax**
  - insertion toujours possible

Insertion dans un vecteur non trié

- On insère l'élément à la fin

procédure insertfin(dr V[1..n]: vecteur; d elem : t) ;
spécification *{n ≥ 0}* ➔ *{ajout d'un élément à la fin de V}*
debproc
    n := n+1 ;
    V[n] := elem ;
finproc ;

Insertion dans un vecteur trié

- Après l'insertion le vecteur doit rester trié

- On procède en deux étapes
  1. Chercher la place que doit occuper l'élément à insérer
     - fonction **posit**
  2. Effectuer l'insertion de l'élément par décalage
     - procédure **insertplace**

procédure **insertion**

procédure insertion(dr V[1..n]:vecteur ; d elem : t) ;
spécification *{n ≥ 0, V[1..n] trié}* ➔
{insertion de elem dans V tel que V reste trié}

```
    p : entier ;
debproc
    si n = 0 alors                    {insertion dans un vecteur vide}
        V[1] := elem ;
        n := 1 ;
    sinon                             {n > 0}
        p := posit (V[1..n], elem) ;
                                      {V[1..p-1] ≤ elem < V[p..n], p ∈ [1..n+1]}
        inserplace (V[1..n], elem, p) ;
    finsi ;
finproc ;
```


Recherche de la place

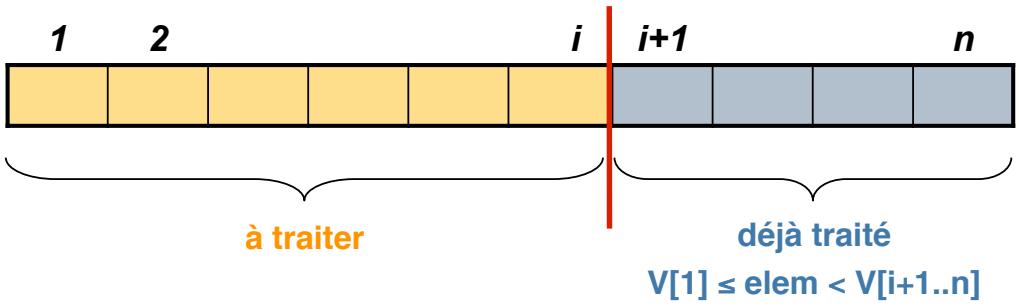
- On cherche un indice p (p ∈ [1..n+1]) tel que :

V[1..p-1] ≤ elem < V[p..n]

- C'est-à-dire
  - la place la plus « à droite » possible
  - ➔ minimum de décalages pour l'insertion
- Méthode
  - Recherche séquentielle
    - On commencera à droite !
  - Recherche dichotomique

Recherche séquentielle de la place

- Si p vaut 1 on a fini
  - V[1] > elem
- Sinon, on cherche p dans [2..n+1]
  - En schématisant :



  - On s'arrêtera dès que V[i] ≤ elem

Recherche séquentielle de la place

- Raisonnement par récurrence

Hypothèse V[1] ≤ elem < V[i+1..n]

➤ V[i] ≤ elem ⇨ p := i + 1 ; ✳
➤ V[i] > elem ⇨ i := i - 1 ; ➡ ***H***

Itération tantque (V[i] > elem) faire …

Initialisation

➤ V[1] > elem ⇨ p := 1 ; ✳
➤ V[1] ≤ elem ⇨ i := n ; ➡ ***H***

fonction **posit** séquentielle

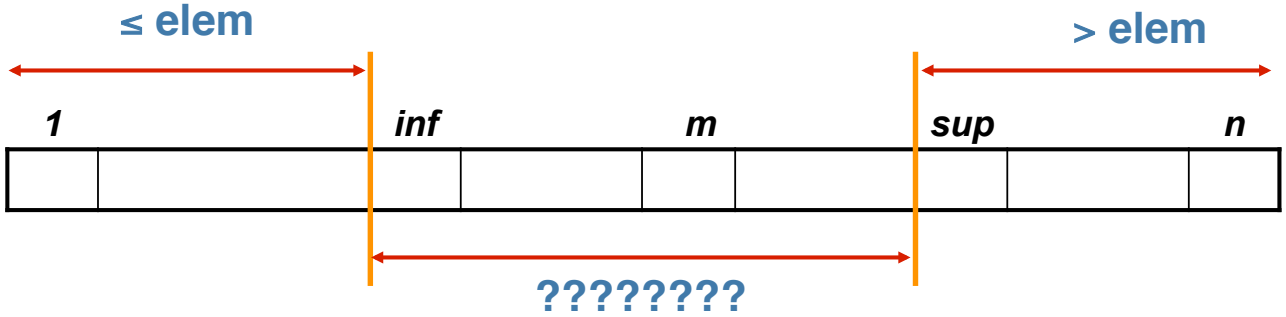
fonction posit (d V[1..n] : vecteur ; d elem : t) : entier ;
spécification *{n > 0, V[1..n] trié}* ➔
{résultat = p, p ∈ [1..n+1], V[1..p-1] ≤ elem < V[p..n]}
p, i : entier ;
debfonc
si V[1] > elem **alors**
p := 1 ; *{elem < V[p..n]}*
sinon *{V[1] ≤ elem}*
i := n ; *{V[1] ≤ elem < V[i+1..n]}*
tantque V[i] > elem **faire** *{V[1] ≤ elem < V[i..n]}*
i := i - 1 ; *{V[1] ≤ elem < V[i+1..n]}*
finfaire ; *{V[1..i] ≤ elem < V[i+1..n]}*
p := i + 1 ;
finsi ;
retour p ; *{V[1..p-1] ≤ elem < V[p..n], p ∈ [1..n+1]}*
finfonc ;

Recherche dichotomique de la place

- Traiter d'abord le cas p = n + 1
- Ensuite, on cherche p dans [1..n] tel que

V[1..p-1] ≤ elem < V[p..n]

- En schématisant :



Recherche dichotomique de la place

Raisonnement par récurrence

Hypothèse V[1..inf-1] ≤ elem < V[sup..n]

- inf = sup ⇨ p := sup ; ✳
- inf < sup

  m := (inf + sup) div 2 ;

  - V[m] ≤ elem ⇨ inf := m + 1 ; ➡ ***H***
  - V[m] > elem ⇨ sup := m ; ➡ ***H***

Itération tantque (inf < sup) faire …

Initialisation

- V[n] ≤ elem ⇨ p := n + 1 ; ✳
- V[n] > elem ⇨ inf := 1 ; sup := n ➡ ***H***

fonction **posit** dichotomique

```
fonction posit (d V[1..n] : vecteur ; d elem : t) : entier ;
spécification {n>0,V[1..n] trie} ➔ {résultat=p, p∈[1..n+1], (V[1..p-1]≤elem<V[p..n]}
      p, inf, sup, m : entier ;
debfonc
   si V[n] ≤ elem alors
      p := n + 1 ;                          {V[1..p-1] ≤ elem}
   sinon
      inf := 1 ; sup := n ;                 {V[1..inf-1] ≤ elem < V[sup..n]}
      tantque inf < sup faire
            m := (inf + sup) div 2 ;
            si V[m] ≤ elem alors
                inf := m + 1 ;              {V[1..inf-1] ≤ elem}
            sinon
                sup := m ;                  {V[sup..n] > elem}
            finsi ;
      finfaire ;                            {V[1..sup-1] ≤ elem < V[sup ..n]}
      p := sup ;                            {V[1..p-1] ≤ elem < V[p..n]}
   finsi ;
   retour p ;
finfonc ;
```

procédure **insertplace**

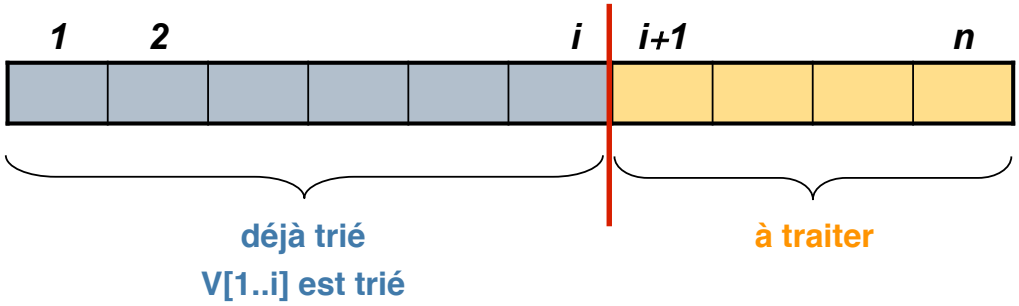
procédure inserplace (**dr** V[1..n] : vecteur ; d **elem** : t ; d **p** : entier) ;
spécification *{V[1..n] trié, p ∈[1..n+1]}* ➔
{insertion de **elem** *à la place* **p** *après décalage des éléments de V[p..n]}*

```
debproc
    pour i := n bas p faire  {décalage à droite}
        V[i+1] := V[i] ;
    finfaire ;
    n := n+1 ;
    V[p] := elem ;
finproc ;
```

- Décalage à droite à partir du dernier !
- Forme abrégée du pour … : V[p+1..n+1] := V[p..n] ;

Tri par insertion

- Supposons triés les i (1 ≤ i ≤ n) premiers éléments du vecteur V



- Il suffit d'insérer V[i+1] à sa place dans V[1..i] pour que V[1..i+1] soit trié
  - On continue comme cela jusqu'à n

Tri par insertion

☐ Raisonnement par récurrence

Hypothèse V[1..i] trié , V[i+1..n] non traité

➤ i = n ⇨ V[1..n] est trié ✳
➤ i < n ⇨ insertion (V[1..i], V[i+1]) ; ➡ ***H***

Itération tantque (i < n) faire …

Initialisation i := 1 ; ➡ ***H***

procédure **triinsert**

procédure triinsert (**dr** V[1..n]:vecteur) ;
spécification *{n > 0}* ➔ *{V[1..n] trié}*
    i : entier ;
debproc
    i := 1 ;
    {V[1..i] trié, V[i+1..n] non traité}
    tantque i < n **faire**
        insertion (V[1..i], V[i+1])
        {V[1..i] trié,V[i+1..n] non traité} ;
    finfaire ;
    {(i = n, V[1..i] trié, V[i+1..n] non traité) ➔ *V[1..n] trié}*
finproc ;

Côut de **triinsert**

- insertion est appelé pour i = 1 à n-1
- À chaque appel de insertion
  - 1 appel de recherche de position dans V[1..i]
  - 1 appel de inserplace
- Recherche de la position
  - Méthode séquentielle :
    - de 1 à i comparaisons
    - en moyenne ≈ i/2 accès
  - Méthode dichotomique
    - au plus $\log_2 i$ comparaisons
    - en moyenne ≈ $\log_2 i$ accès

si i = 1024
alors i/2 = 512
alors que $\log_2 i = 10$

Côut de **triinsert**

- insertplace : 2(i-p+1) accès pour le décalage, et 1 accès pour l'insertion ; p ∈ [1..i+1]
  - nb accès ∈ [1..2i+1]
  - en moyenne ≈ i accès
- Au total
  - Séquentiel

$$\sum_{i=1}^{n-1}(i/2+i) = 3n(n-1)/4 \approx 3n^2/4$$ accès

  - Dichotomique

$$\sum_{i=1}^{n-1}(\log_2 i + i)$$ accès **(très inférieur pour n grand)**

Côut de **triinsert**

- Avec la méthode de recherche **séquentielle**
  - le tri par insertion est comparable au tribulle optimisé

- Avec la méthode de recherche **dichotomique**
  - le tri par insertion est **nettement plus efficace si n est grand**