# Algorithmique & programmation

Chapitre 2 : Vecteurs

Algorithmes de mise à jour

Suppression

## Suppression dans un vecteur trié

- On procède en deux étapes
  1. Chercher la place de l'élément à supprimer
     - fonction **positsup**
  2. Effectuer la suppression de l'élément par décalage
     - procédure **tasser**

- La procédure positionnera un booléen qui dira si la suppression a pu être effectuée ou non

# procédure **supp**

**procédure** supp (**dr** V[1..n] : vecteur ; d **elem** : t ; **r possible** : booléen) ;
**spécification** *{n > 0}* ➔
*{(possible, suppression d'une occurrence de elem dans V)*
*v (¬possible, suppression impossible)}*

```
    p : entier ;
debproc
    possible := faux ;
    p := positsup (V[1..n], elem) ;
    si p > 0 alors
        tasser (V[1..n], p) ;
        possible := vrai ;
    finsi ;
finproc ;
```



# Recherche de la place

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

**V[p] = elem, elem < V[p+1..n]**
**ou bien, si elem ∉ V[p+1..n] alors p=0**

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

# Recherche de la place séquentielle

- Vers l'hypothèse
  - On cherche d'abord p tel que :
    - V[1..p] ≤ elem < V[p+1..n]
  - ensuite, il suffira de vérifier si V[p] = elem



# Recherche de la place séquentielle

- Raisonnement par récurrence

***Hypothèse*** V[1] ≤ elem < V[i+1..n]

- V[i] < elem ⇨ p := 0 ; ✳
- V[i] = elem ⇨ p := i ; ✳
- V[i] > elem ⇨ i := i - 1 ; ➡ ***H***

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

***Initialisation***

- V[1] > elem ⇨ p := 0 ; ✳
- V[1] ≤ elem ⇨ i := n ; ➡ ***H***

# fonction **positsup** séquentielle

**fonction** positsup (d V[1..n] : vecteur ; d elem : t) : entier;
**spécification** *{n > 0, V[1..n] trié}* ➔ *{résultat = p, (elem ∉ V, p = 0)*
*v (V[p] = elem, elem < V[p+1..n], p∈[1..n])}*

```
    p, i : entier ;
debfonc
    p := 0 ;
    si V[1] ≤ elem alors
        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]}
        si V[i] = elem alors
            p := i ;
        finsi ;
    finsi ;
    retour p ;
finfonc ;
```



# fonction **positsup** dichotomique

**fonction** positsup (d V[1..n] : vecteur ; d elem : t) : entier;
**spécification** *{n > 0, V[1..n] trié}* ➔ *{résultat = p, (elem ∉ V, p = 0)*
*v (V[p] = elem, elem < V[p+1..n], p∈[1..n])}*

```
    p, inf, sup, m : entier ;
debfonc
    p := 0 ;
    si V[n] = elem alors
        p := n ;
    sinonsi V[n] > elem alors

        {recherche dichotomique}




    finsi ;
    retour p ;
finfonc ;
```

# fonction **positsup** dichotomique

```
    sinonsi V[n] > elem alors
      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 ;                         {V[1..inf-1] ≤ elem < V[sup..n]}
      finfaire ;
      {V[1..inf-1]≤elem<V[sup..n], inf=sup➔V[1..sup-1]≤elem<V[sup ..n]}
      si (sup>1) et alors (V[sup-1]=elem) alors
        p := sup - 1;
      finsi ;
    finsi ;
    retour p ;
finfonc ;
```



# procédure **tasser**

**procédure** tasser (**dr** V[1..n] : vecteur ; d p : entier) ;

**spécification** *{n > 0, p ∈ [1..n]}* ➔

*{retassement des éléments d'indice > p}*

```
debproc
    pour j := p + 1 haut n faire
        V[j-1] := V[j] ;
    finfaire ;
    n := n - 1 ;
finproc ;
```

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