Algorithmique & programmation

Chapitre 2 : Vecteurs

Vecteur trié

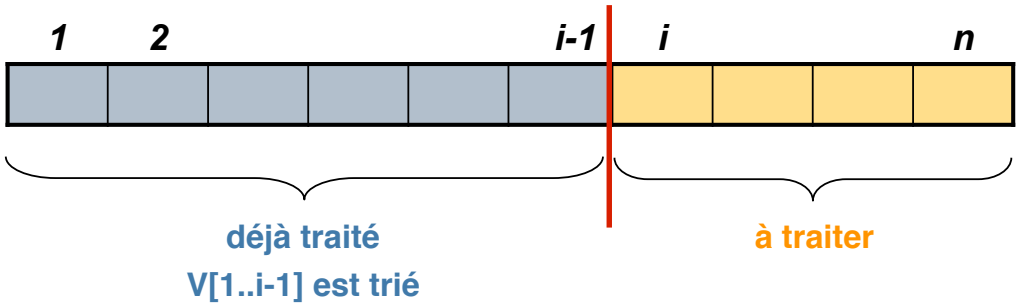
Recherche séquentielle triée

Vecteur trié (ordonné)

- Définition
  - un vecteur vide (n=0) est trié
  - un vecteur à un seul élément (n=1) est trié
  - un vecteur V[1..n], n>1, est ordonné si
    - $\forall i \in [2..n], V[i-1] \leq V[i]$
- Définition récursive
  - un vecteur vide (n=0) est trié
  - un vecteur à un seul élément (n=1) est trié
  - **si** (V[1..i-1] trié et $V[i-1] \leq V[i]$)
    - **alors** V[1..i] trié pour $i \in [2..n]$

Vérifier qu'un vecteur est trié

- Recherche de l'hypothèse
  - Si on est encore en train d'essayer de savoir si un vecteur est trié, c'est que la partie du vecteur que l'on a examinée est triée
  - En schématisant :



Vérifier qu'un vecteur est trié

- Raisonnement par récurrence

Hypothèse V[1..i-1] est trié

➤ i = n+1 ⇨ *V[1..n] est trié*
⇨ résultat = vrai ✳

➤ i ≤ n ⇨

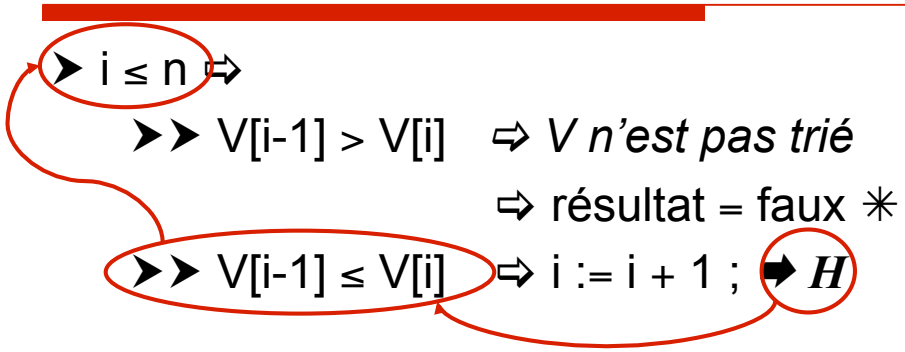
➤➤ V[i-1] > V[i] ⇨ *V n'est pas trié*
⇨ résultat = faux ✳

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

Trouver l'itération ?

Chemin depuis le retour à l'hypothèse vers les conditions

Trouver l'itération



- Il faut que les deux conditions soient vérifiées
  - i ≤ n et V[i-1] ≤ V[i]

Attention !!!!!

- ***La première fois que i ≤ n devient faux i = n + 1***
- **Peut-on consulter V[n + 1] ?**
  - **NON !!!!!!!**

Trouver l'itération

- Il faut que les deux conditions soient vérifiées
  - i ≤ n **et** (V[i-1] ≤ V[i])
- On doit éviter
  - d'évaluer (V[i-1] ≤ V[i]) lorsque i ≤ n est faux
- Solution ?
  - Remplacer le **et** par un **et alors**
  - Pour obtenir

    i ≤ n **et alors** (V[i-1] ≤ V[i])

- *Commentaire*
  - *Ce n'est pas de l'optimisation, c'est **obligatoire** !!!*

Vérifier qu'un vecteur est trié

- Raisonnement par récurrence

Hypothèse V[1..i-1] est trié

- i = n+1 ⇨ *V[1..n] est trié*
  ⇨ résultat = vrai ✳
- i ≤ n ⇨
  - V[i-1] > V[i] ⇨ *V n'est pas trié*
    ⇨ résultat = faux ✳
  - V[i-1] ≤ V[i] ⇨ i := i + 1 ; ➡ ***H***

Itération tantque (i ≤ n) **et alors** (V[i-1] ≤ V[i]) faire …

Trouver l'initialisation ?

Connaît-on un vecteur pour lequel on sait qu'il est trié ?

Trouver l'initialisation

- Connaît-on un vecteur pour lequel on sait sans calcul qu'il est trié ?

Oui !!

- Le vecteur à un élément (pourquoi pas le vecteur vide ?)
  - V[1..1]
- *Rappel*

Hypothèse V[1..i-1] est trié

- Quelle valeur de **i** pour avoir **V[1..1]** ?
  - Réponse : **i doit valoir 2**

Initialisation i := 2 ; ➡ ***H***

Vérifier qu'un vecteur est trié

- Raisonnement par récurrence

Hypothèse V[1..i-1] est trié

➤ i = n+1 ⇨ *V[1..n] est trié*
⇨ résultat = vrai ✳

➤ i ≤ n ⇨

➤➤ V[i-1] > V[i] ⇨ *V n'est pas trié*
⇨ résultat = faux ✳

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

Itération tantque (i ≤ n) **et alors** (V[i-1] ≤ V[i]) faire …

Initialisation i := 2 ; ➡ ***H***

fonction **trié**

```
fonction trié (d V[1..n] : vecteur) : booléen ;
spécification { } ➔ {résultat = V est trié}
        i : entier ;
debfonc
        i := 2 ;
        {V[1..i-1] trié}
        tantque (i ≤ n) et alors (V[i-1] ≤ V[i]) faire
                i := i +1 ;
        finfaire ;
        {(i > n)) ou sinon (V[i-1] > V[i])}
        il faut rendre vrai ou faux
finfonc ;
```

fonction **trié** (calcul du résultat)

- Condition de sortie du tantque
  - *{(i > n) ou sinon (V[i-1] > V[i])}*
- La fonction doit rendre vrai ou faux (raisonnement)
  - Soit on a l'intuition du calcul
  - Soit on fait un tableau de sortie

i > n	**V[i-1] > V[i]**	**résultat**
vrai	non examiné	vrai
faux (i≤n)	vrai	faux
faux (i≤n)	faux	impossible (tantque)

fonction **trié** (calcul du résultat)

- Lecture du tableau de sortie

i > n	**V[i-1] > V[i]**	**résultat**
vrai	non examiné	vrai
faux (i≤n)	vrai	faux
faux (i≤n)	faux	impossible (tantque)

- Le résultat est vrai lorsque i > n, faux sinon
  - **retour** i > n ;

fonction **trié** générique

fonction trié (d V[1..n] : vecteur) : booléen ;
spécification *{ }* ➔ *{résultat = V est trié}*
    i : entier ;
debfonc
    i := 2 ;
    {V[1..i-1] trié}
    tantque (i ≤ n) **et alors** (V[i-1] ≤ V[i]) **faire**
        i := i +1 ;
    finfaire ;
    {(i > n)) ou sinon (V[i-1] > V[i])}
    retour i > n ;
finfonc ;

fonction **trié** (vecteur d'entiers)

- soit V un vecteur d'entier de type **vectInt**

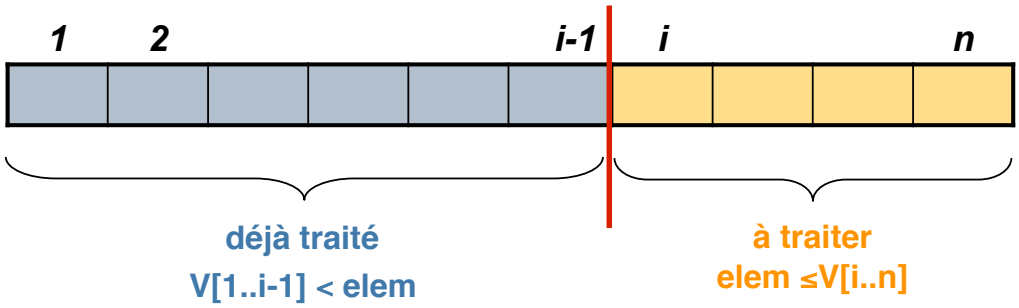
```
function triéInt (V : in vectInt) retrun boolean is
--{ } ➔ {résultat = V est trié}
  i : integer ;
begin
  i := V'First + 1 ;
  --{V(1..i-1) trié}
  while (i ≤ V'Last) and then (V(i-1) ≤ V(i)) loop
    i := i +1 ;
  end loop ;
  --{(i > V'Last)) ou sinon (V(i-1) > V[i])}
  return i > n ;
end triéInt ;
```

Accès à un élément (vecteur trié)

- On veut savoir si l'élément **elem** est présent dans le vecteur V

fonction accètrié1 (d V[1..n] : vecteur ; d elem : t) : booléen ;
spécification *{n ≥ 0, V[1..n]trié}* ➔*{résultat = elem ∈ V[1..n]}*

- Si on cherche encore, c'est que tous les éléments que l'on a examinés sont **< elem**



Accès à un élément (vecteur trié)

- Raisonnement par récurrence

Hypothèse V[1..i-1] < elem

➤ i = n+1 ⇨ *elem ∉ V[1..n]*
⇨ résultat = faux ✳

➤ i ≤ n ⇨
➤➤ V[i] ≥ elem ⇨ *elem ∈ V[1..n] ssi V[i] = elem*
⇨ résultat = (V[i] = elem) ✳
➤➤ V[i] < elem ⇨ i := i + 1 ; ➡ ***H***

Itération tantque (i ≤ n) **et alors** (V[i] < elem) faire …

Initialisation i := 1 ; ➡ ***H***

fonction **accèstrié1**

fonction accètrié1 (d V[1..n] : vecteur ; d elem : t) : booléen ;
spécification *{n ≥ 0, V[1..n]trié}* ➔*{résultat = elem ∈ V[1..n] }*
    i : entier ;
debfonc
    i := 1 ;
    {V[1..i-1] < elem}
    tantque (i ≤ n) **et alors** (V[i] < elem) **faire**
        {V[1..i] < elem}
        i := i + 1 ;
        {V[1..i-1] < elem}
    finfaire ;
    {(i > n) ou sinon (V[i] ≥ elem)} {en particulier i = n + 1}
    il faut rendre vrai ou faux
finfonc ;

fonction **accèstrié1** (calcul du résultat)

- Condition de sortie du tantque
  - *{(i > n) ou sinon (V[i] ≥ elem)}*
- La fonction doit rendre vrai ou faux (raisonnement)
  - Soit on a l'intuition du calcul
  - Soit on fait un tableau de sortie

i > n	V[i] ≥ elem	résultat
vrai	non examiné	faux
faux (i≤n)	vrai	vrai ssi V[i] = elem
faux (i≤n)	faux	impossible (tantque)

fonction **accèstrié1** (calcul du résultat)

- Lecture du tableau de sortie

i > n	V[i] ≥ elem	résultat
vrai	non examiné	faux
faux (i≤n)	vrai	vrai ssi V[i] = elem
faux (i≤n)	faux	impossible (tantque)

- Le résultat est vrai lorsque i ≤ n **et alors** V[i] = elem, faux sinon
  - **retour** i ≤ n **et alors** V[i] = elem ;

fonction **accèstrié1**

fonction accètrié1 (d V[1..n] : vecteur ; d elem : t) : booléen ;
spécification *{n ≥ 0, V[1..n]trié}* ➔*{résultat = elem ∈ V[1..n] }*
    i : entier ;
debfonc
    i := 1 ;
    {V[1..i-1] < elem}
    tantque (i ≤ n) **et alors** (V[i] < elem) **faire**
        {V[1..i] < elem}
        i := i + 1 ;
        {V[1..i-1] < elem}
    finfaire ;
    {(i = n + 1) ou sinon (V[i] ≥ elem)}
    retour (i ≤ n) et alors (V[i] = elem) ;
finfonc ;

fonction **accèstrié1** (vecteur d'entiers)

- soit V un vecteur d'entier de type **vectInt**

```
function accètrié1Int (V : in vectInt ;
                       elem : in integer) return boolean is
--{n ≥ 0, V trié} ➔ {résultat = elem ∈ V}
   i : integer ;
begin
   i := V'First ;
   --{V(1..i-1) < elem}
   while (i ≤ V'Last) and then (V(i) < elem) loop
      i := i + 1 ;
   end loop ;
   --{(i = V'Last + 1) ou sinon (V(i) ≥ elem)}
   return (i ≤ n) and then (V(i) = elem) ;
end accèstrié1Int ;
```


Accès à un élément (vecteur trié non vide)

- S'arrêter sur l'avant dernier élément pour éviter le **et alors**

fonction accètrié2 (d V[1..n] : vecteur ; d elem : t) : booléen ;
spécification *{**n > 0**, V[1..n]trié}* ➔*{résultat = elem ∈ V[1..n] }*
    i : entier ;
debfonc
    i := 1 ;
    tantque (**i < n**) **et** (V[i] < elem) **faire**
        i := i + 1 ;
    finfaire ;
    {(i = n) ou sinon (V[i] ≥ elem)}
    il faut rendre vrai ou faux
finfonc ;

fonction **accèstrié2** (calcul du résultat)

- Tableau de sortie

i = n	V[i] ≥ elem	résultat
vrai	vrai	vrai ssi V[i] = elem
vrai	faux	faux
faux (i<n)	vrai	vrai ssi V[i] = elem
faux (i<n)	faux	impossible (tantque)

- Le résultat est vrai lorsque V[i] = elem quelle que soit la valeur de i, faux sinon
  - **retour** V[i] = elem ;

fonction **accèstrié2**

fonction accètrié2 (d V[1..n] : vecteur ; d elem : t) : booléen ;
spécification *{n > 0, V[1..n]trié}* ➔*{résultat = elem ∈ V[1..n] }*
    i : entier ;
debfonc
    i := 1 ;
    tantque (**i < n**) **et** (V[i] < elem) **faire**
        i := i + 1 ;
    finfaire ;
    {(i = n) ou sinon (V[i] ≥ elem)}
    retour V[i] = elem ;
finfonc ;

fonction **accèstrié2** (vecteur d'entiers)

- soit V un vecteur d'entier de type **vectInt**

```
fonction accètrié2Int (V : in vectInt ;
                       elem : in integer) returns boolean is
--{V non vide trié} ➔ {résultat = elem ∈ V}
  i : integer ;
begin
  i := V'First ;
  while (i < V'Last) and (V(i) < elem) loop
    i := i + 1 ;
  end loop ;
  --{(i = V'Last) ou sinon (V(i) ≥ elem)}
  return V(i) = elem ;
end accètrié2Int ;
```


Coût d'une recherche séquentielle

- Accès à un élément indicé (V[i]) très coûteux par rapport à un élément non indicé
- La comparaison entre deux éléments indicés nécessite deux accès (V[i] et V[j])

- Si le vecteur V[1..n] n'est **pas trié** il faut :
  - n comparaisons (accès) si elem ∉ V
  - n÷2 comparaisons en moyenne si elem ∈ V

- Si le vecteur V[1..n] est **trié** il faut
  - n÷2 comparaisons en moyenne si elem ∈ ou ∉ V