Algorithmique & programmation

Chapitre 3 : Fichiers séquentiels

Algorithme traitant un seul fichier

Accès par position

Accès à un élément d'un fichier

- Deux type d'accès
  - **par position**
    - on connaît le rang **k** de l'élément dans le fichier
      - *n'avait aucun intérêt sur un vecteur (accès direct)*
    - sur n'importe quel type de fichier, trié ou non
  - **associatif**
    - on connaît la valeur **val** de l'élément et on cherche la première occurrence
    - le type de fichier, tiré ou non trié, aura son importance

Accès au k^{ième} élément

on suppose k > 0

- Première version : f = f_1^{k-1} || f_k^n
  - Deux parties

1. Parcours des k-1 premiers éléments de f
   - $f^- = f_1^{k-1}$
2. Ensuite :
   - si $f^+ = <>$ alors pas de k^{ième} élément
   - si $f^+ \neq <>$ le k^{ième} élément est le premier de f^+

Première partie ($f^- = f_1^{k-1}$)

- Raisonnement par récurrence

Hypothèse $f^- = f_1^{i-1}$

➤ i = k ⇨ $f^- = f_1^{k-1}$ ✳

➤ i < k

➤➤ $f^+ = <>$ ⇨ *le k^{ième} n'existe pas* ✳

➤➤ $f^+ \neq <>$ ⇨ lire (f, c) ; i := i + 1 ; ➡ ***H***

Itération tantque (i < k) et non fdf (f) faire...

Initialisation relire (f) ; i := 1 ; ➡ ***H***

Condition de sortie du tantque

- tantque (i < k) et non fdf (f) faire...

- À la sortie
  - ¬ ((i < k) et ¬fdf (f))

  = ¬(i < k) ou ¬(¬fdf (f))

  = (i ≥ k) ou fdf (f)

  = (i = k + 1) ou fdf (f)

Accès au k^{ième} élément (calcul du résultat)

- Tableau de sortie

i = k	fdf(f)	résultat
vrai	vrai	trouvé := faux
vrai	faux	trouvé := vrai ; lire(f, valk)
faux (i≤n)	vrai	faux
faux (i≤n)	faux	impossible (tantque)

- trouvé est vrai lorsque non fdf(f)
  - **trouvé := non fdf(f)** ;

procédure **accèsk1** (première version)

procédure accesk1 (d f : fichier de t ; d k : entier ;
r trouvé : booléen ; **r** valk : t) ;
spécification *{k > 0}* ➔ *{(trouvé , valk = x_k) v (¬trouvé , valk indéfini)}*
c : t ; i : entier ;
debproc
relire (f) ;
trouvé := faux ;
i := 1 ;
tantque (i < k) **et non fdf** (f) **faire**
lire (f , c) ;
i := i + 1
finfaire ;
si non fdf (f) alors
trouvé := vrai ;
lire (f , valk) ;
finsi ;
finproc ;

procédures **accesk** (fichier d'entiers)

```
-- Importer le module générique Sequential_io, use inutile
with Sequential_Io;

package P_Fentier is
   -- Instancier le paquetage Sequential_Io
   -- Pour manipuler des fichiers de Integer
   package P_Entier_Io is new Sequential_Io(Integer);
   use P_Entier_Io;

   -- Première version de la procédure
   procedure accesk1(F : in P_Entier_Io.File_Type ; k : in
     integer ; trouve : out boolean ; valk : out integer);


   -- Seconde version de la procédure
   procedure accesk1(F : in P_Entier_Io.File_Type ; k : in
     integer ; trouve : out boolean ; valk : out integer);

end P_Fentier;
```

procédure **accesk1**

```
procedure accesk1(F : in P_Entier_Io.File_Type ; k : in
    integer ; trouve : out boolean ; valk : out integer) is

--spec  {k  >  0}=>{(trouvé  ,  valk  =  x_k)  v  (¬trouvé  ,  valk
    indéfini)}

  c : integer ;   i : integer ;
begin
  reset(F) ;          --on se positionne au début du fichier
  trouve := false ;   --initialisation de trouve à faux
  i := 1 ;            --on se prépare à lire le premier de F
  while (i < k) and not End_Of_File(F) loop
    read (F , c) ;
    i := i + 1 ;
  end loop ;
  if not End_Of_File(F) then
    trouvé := true ;
    read (f , valk) ;
  end if ;
end accesk1 ;
```


Accès au k^{ième} élément

on suppose k > 0

- Deuxième version : $f = f_1^k \,||\, f_{k+1}^n$
  - Deux parties

1. Parcours des k premiers éléments de f
   - $f^- = f_1^k$
2. Ensuite : soit Df^- le dernier élément de f^-
   - si Df^- n'existe pas alors pas de k^{ième} élément
   - si Df^- existe, c'est le k^{ième} élément

Première partie ($f^- = f_1^k$)

- Raisonnement par récurrence

Hypothèse $f^- = f_1^{i-1}$

➤ $i = k + 1$ ⇨ $f^- = f_1^k$ ✳

➤ $i \leq k$

➤➤ $f^+ = <>$ ⇨ *le $k^{ième}$ n'existe pas* ✳

➤➤ $f^+ \neq <>$ ⇨ lire (f, c) ;
i := i + 1 ; ➡ ***H***

Itération tantque (i ≤ k) et non fdf (f) faire...

Initialisation relire (f) ; i := 1 ; ➡ ***H***

Condition de sortie du tantque

- tantque (i ≤ k) et non fdf (f) faire...

- À la sortie
  - ¬ ((i ≤ k) et ¬fdf (f))

  = ¬(i ≤ k) ou ¬(¬fdf (f))

  = (i > k) ou fdf (f)

  = (i = k + 1) ou fdf (f)

Accès au k^{ième} élément (calcul du résultat)

- Tableau de sortie

i = k + 1	fdf(f)	résultat
vrai	vrai	c=x_k=Df- ⇨ trouvé:=vrai; valk:=c
vrai	faux	c=x_k=Df- ⇨ trouvé:=vrai; valk:=c
faux (i<k + 1)	vrai	trouvé:=faux
faux (i<k + 1)	faux	impossible (tantque)

- trouvé est vrai lorsque **i = k + 1**
  - **trouvé := i = k + 1 ;**

procédure **accèsk2**

procédure accesk2 (d f : fichier de t ; d k : entier ;
r trouvé : booléen ; **r** valk : t) ;
spécification *{k > 0}* ➔ *{(trouvé , valk = x_k) v (¬trouvé , valk indéfini)}*
c : t ; i : entier ;
debproc
relire (f) ;
trouvé := faux ;
i := 1 ;
tantque (i ≤ k) **et non fdf** (f) **faire**
lire (f , c) ;
i := i + 1
finfaire ;
si i = k + 1 alors
trouvé := vrai ;
valk := c ;
finsi ;
finproc ;

procédure **accesk2**

```
procedure accesk2(F : in P_Entier_Io.File_Type ; k : in
    integer ; trouve : out boolean ; valk : out integer) is

--spec  {k  >  0}=>{(trouve  ,  valk  =  x_k)  v  (¬trouve  ,  valk
    indéfini)}

  c : integer ;   i : integer ;
begin
  reset(F) ;      --on se positionne au début du fichier
  trouve := false ;  --initialisation de trouve à faux
  i := 1 ;     --on se prépare à lire le premier de F
  while (i <= k) and not End_Of_File(F) loop
    read (F , c) ;
    i := i + 1 ;
  end loop ;
  if i=k+1 then
    trouvé := true ;
    valk := c ;
  end if ;
end accesk2 ;
```