Analyse de fichier client

Le problème

- entreprise avec 10 millions de clients,
  - client identifié par un numéro de 11 111 111 à 99 999 999.
  - fichier séquentiel représente l'historique résumé des factures des 12 derniers mois.
- fournir deux listes:
  - liste des 100 plus gros clients,
    ceux dont le montant total des factures des 12 derniers mois est le plus élevé,
  - liste des 100 meilleurs payeurs,
    ceux dont le délai de paiement moyen a été le plus court sur la même période.

Les structures (1)


```
type T_Num_Client is range 11_111_111..99_999_999;
type T_Somme is range 0..1_000_000_00;                --1 million de francs
type T_Date is range 0..36500;                -- depuis le 1/1/1950, jusque 2050
type T_Resume is
    record
        Numero : T_Num_Client;
        Montant : T_Somme; -- en centimes
        Date_Fact, Date_Paiement : T_Date;
    end record;
package SIO is new Ada.Sequential_IO (T_Resume); use SIO;
Fichier_Resume : SIO.File_Type;  -- fichier historique
    -- en ordre croissant des dates de paiement,
    -- 1 million d'enregistrements, concernant 100 000 clients.
    -- lecture du fichier prend 20µs par enregistrement,
```

Les structures (2)

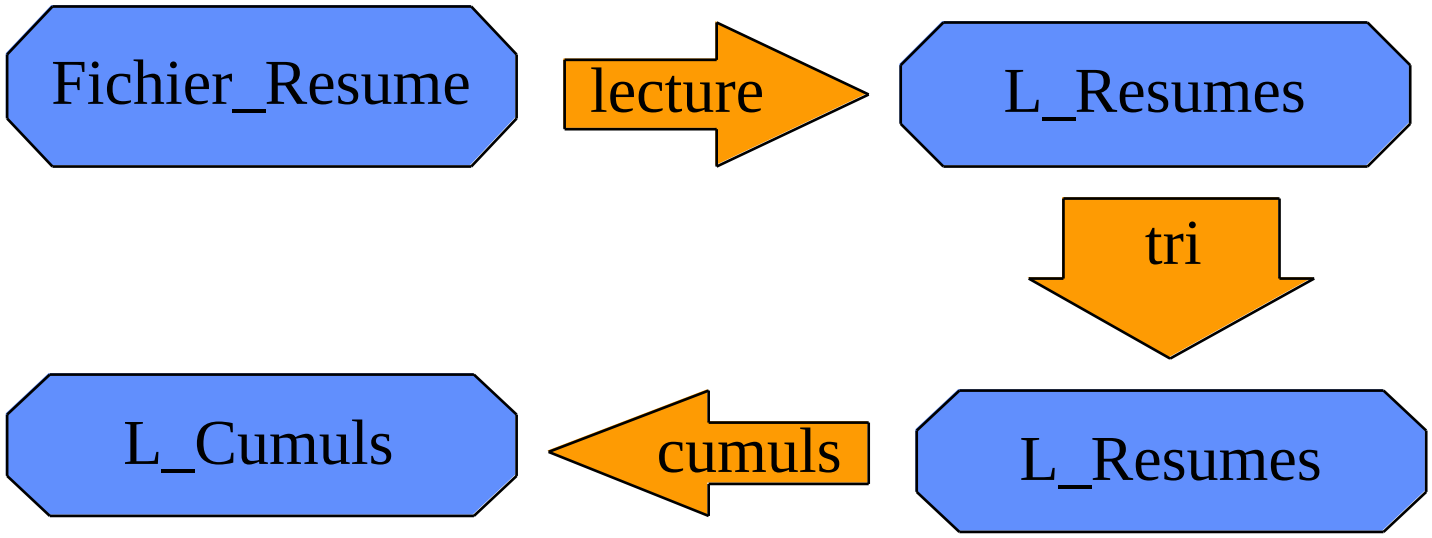

```
type T_Nb_Fact is range 0..400;      -- 400 factures par client
type T_Cumul is
    record
        Numero : T_Num_Client;
        Nb_Fact : T_Nb_Fact; -- nombre de factures
        Montant : T_Somme; -- total
        Delais : T_Date; -- total des délais
    end record;
package PL_Cumuls is new Listes_Contigues (Element=> T_Cumul);
use PL_Cumuls;
L_Cumuls : PL_Cumuls.Liste (100_000);
-- le cumul des valeurs prend 10µs par enregistrement
-- une comparaison ou un transfert prend 1µs.
```

Première partie (1)

package PL_Resumes **is new** Listes_Contigues(Element=>T_Resume);

use PL_Resumes;

L_Resumes : PL_Resumes.Liste (1_000_000);

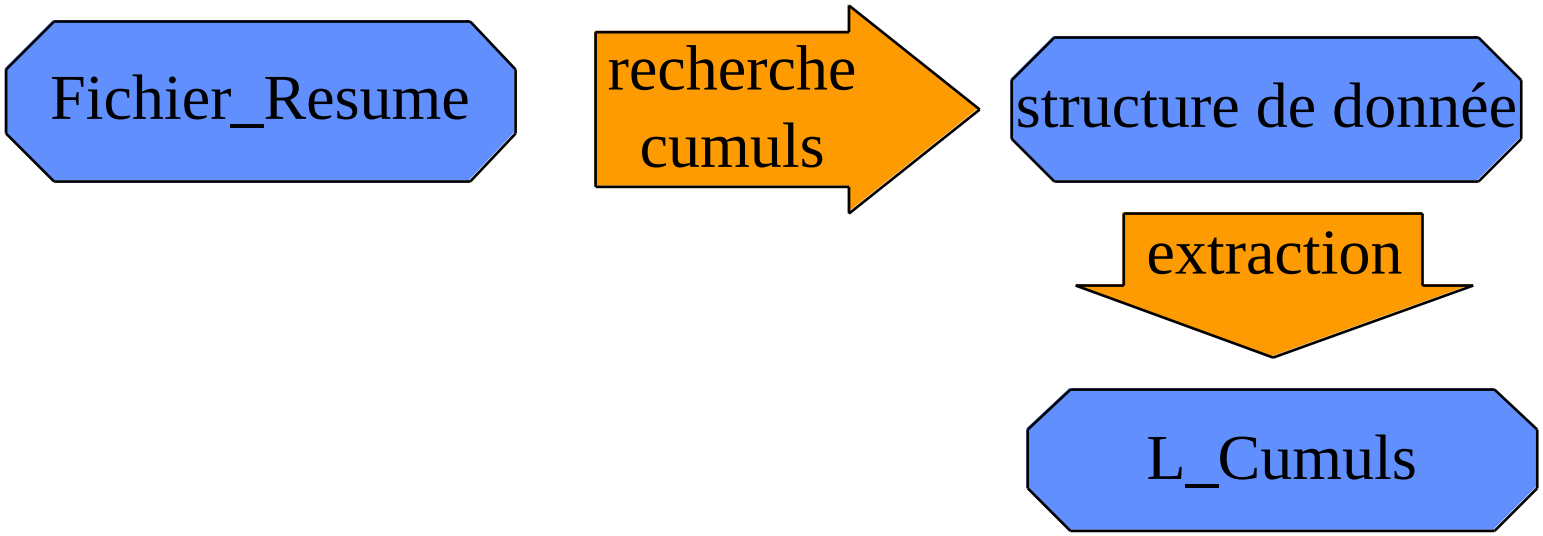


Première partie (2)

- lecture du fichier dans la liste
  - lecture: 1 000 000 * 20 µs => 20 secondes
- algorithme de tri: rapide ou tas
  - tri: n log n => 20 * 1 000 000 * (2c + 1t) = 60 secondes
- parcours séquentiel style "Unique"
  - création au fur et à mesure nouveaux clients
  - cumul pour les factures d'un même client qui se suivent
  - => 1 000 000 * 10 µs = 10 secondes
- soit 1 à 2 minutes, 12 à 16 Mo en mémoire centrale

Deuxième partie (1)

- utilisation d'une structure interne de cumul
  - liste contigue, quelconque ou ordonnée
  - liste chaînée, quelconque ou ordonnée
  - arbre de recherche, H équilibré ou non



Deuxième partie (2)

- lecture fichier (20 s.) et cumuls valeurs (10 s.) => 30 s.
- Selon la structure (n factures, p clients)
  - liste contigue en ordre quelconque, adjonctions au bout
    - recherche séquentielle p*n => 10 heures
  - liste ordonnée
    - recherche dichotomique n log p => 17 s.
    - insertions p*p/4 => 40 mn
  - liste chaînée => idem a liste contigue quelconque
  - arbre AVL
    - recherche arbre et adjonction: n log p => 17 s.
    - liste par exploration => moins d'1 s.
- AVL => de l'ordre de 1 mn.

Deuxième partie (3)


```
with Type_Fichier_Client; use Type_Fichier_Client;
function Type_Fichier_Client.La_Cle (E: in T_Cumul) return T_Num_Client is
begin return E.Numero;  end;

with Type_Fichier_Client; use Type_Fichier_Client;
with Type_Fichier_Client.La_Cle;
with Arbres_Binaires.De_Recherche;
package Arbres_Binaires.AR_Cumuls is
    new Arbres_Binaires.De_Recherche (T_Cumul, T_Num_Client);

with Arbres_Binaires.AR_Cumuls;
with Arbres_Binaires.De_Recherche.AVL;
package Arbres_Binaires.AR_Cumuls.PA_Cumuls is
    new Arbres_Binaires.AR_Cumuls.AVL;

use Arbres_Binaires.AR_Cumuls.PA_Cumuls;
A_Cumuls : Arbre_AVL (25);
```

création des arbres de recherche de cumuls

création des arbres AVL de cumuls

déclaration de notre arbre

Deuxième partie (4)

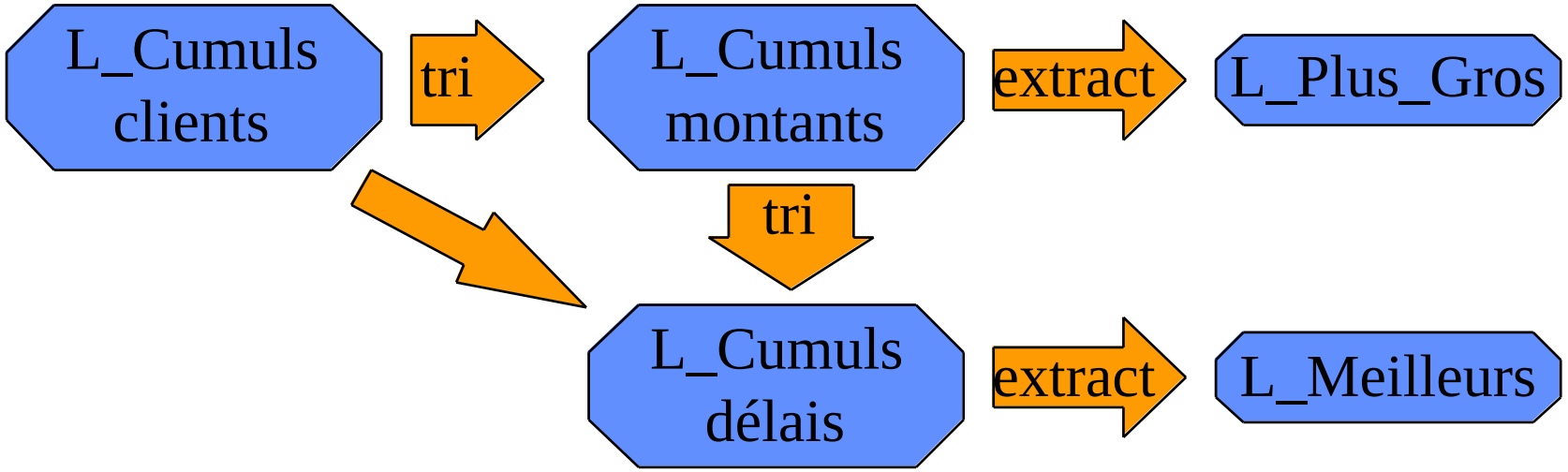

```
procedure Construire_Arbre_Cumuls (A_Cumuls: in out Arbre_AVL) is
    Resume : T_Resume; Cumul : T_Cumul;
begin Vider (A_Cumuls);                          -- partir d'un arbre vide
    while not End_Of_File (Fichier_Resume) loop
      Read (Fichier_Resume, Resume);             -- lecture fichier
      begin
        Positionner (A_Cumuls, Sur => Resume.Numero);    -- recherche
        Cumul := Contenu_Courant (A_Cumuls);     -- ancienne valeur
        Cumul.Nb_Fact := Cumul.Nb_Fact + 1; ...  -- etc, modification
        Changer_Courant (A_Cumuls, Cumul);       -- mise à jour
      exception when Erreur_Specification =>     -- non trouve dans l'arbre
        Ajouter (A_Cumuls, (Resume.Numero, ... );    -- création
      end;
    end loop;
end Construire_Arbre_Cumuls;
```

Deuxième partie (5)


```
procedure Fixer_L_Cumuls (A_Cumuls : in out Arbre_AVL;
                          L_C : in out PL_Cumuls.Liste) is
begin
    Tronquer (L_C, 0);                                  -- vider la liste
    Positionner_Sur_Premier (A_Cumuls);
    while not A_La_Fin (A_Cumuls) loop
       Prolonger (L_C, Contenu_Courant (A_Cumuls));
       Passer_Au_Suivant (A_Cumuls);        -- la remplir avec les nœuds
    end loop;
end Fixer_L_Cumuls;
```

Troisième partie (1)

- trier pour obtenir seulement les 100 intéressants
  - trier tout => p log p soit 5 secondes
  - tri par sélection, arrêter après classement des 100
    - par tas p + 100 log p soit 0,3 s.
- tri croissant/décroissant => extraction des 100 derniers



Troisième partie (2)


```
package PL_Clients is new Listes_Contigues (T_Num_Clients);
use PL_Clients;
L_Plus_Gros, L_Meilleurs : PL_Clients.Liste (100);
procedure Les_100_Derniers (L1 : in PL_Cumuls.Liste;
                                 L2 : in out PL_Cumuls.Liste) is
begin
    Tronquer (L2, 0);
    for I in reverse Longueur (L1) - 99..Longueur (L1) loop
       Prolonger (L2, Ieme (L1, I));
    end loop;
 end Les_100_Derniers;
```

Troisième partie (3)


```
generic
    with function "<" (U, V : in T_Cumul) return Boolean is <>;
procedure Tri_Plus_Grands (La_Liste : in out PL_Cumuls.Liste;
                                      Sur : in Positive);

function Comp_Montants (U, V : in T_Cumul) return Boolean is
begin                          -- comparaison pour obtenir les 100 plus gros clients
    return U.Montant > V.Montant;
end;

procedure Tri_Montants is new Tri_Plus_Grands (Comp_Montants);

function Comp_Delais (U, V : in T_Cumul) return Boolean is
begin                          -- comparaison pour obtenir les 100 meilleurs clients
    return U.Delais / U.Nb_Fact < V.Delais / V.Nb_Fact;
end;

procedure Tri_Delais is new Tri_Plus_Grands (Comp_Delais);
```

Troisième partie (4)


```
procedure Tri_Plus_Grands (La_Liste : in out PL_Cumuls.Liste;
                           Sur : in Positive) is
   package Tas is new PL_Cumuls.En_Tas; use Tas;
   Long : Natural := Longueur (La_Liste);  Dernier : Positive; U : T_Cumul;
 begin
   Ordonner_Le_Tas (La_Liste, Long);       -- construction du tas
    if Sur >= Long then Dernier := 2;          -- en fait demande de tri complet
   else Dernier := Long - Sur + 1;
   end if;
   for K in reverse Dernier .. Long loop   -- extraction du tas
      U := Ieme(La_Liste, K);--permuter racine et dernière feuille
      Changer_Ieme (La_Liste, K, Ieme (La_Liste, 1));
      Changer_Ieme (La_Liste, 1, U);
      Ajuster_Le_Tas (La_Liste, 1, K - 1);      -- corriger le tas
   end loop;
end Tri_Plus_Grands;
```

Troisième partie (5)


```
procedure Extraire_Les_Listes (L : in out PL_Cumuls.Liste;
                               Plus_Gros, Meilleurs : in out PL_Cumuls.Liste) is
begin
    Tri_Montants (L, 100);
    Les_100_Derniers (L, Plus_Gros);
    Tri_Delais (L, 100);
    Les_100_Derniers (L, Meilleurs);
end Extraire_Les_Listes;
```