Raffinages

ZEIDLER Mathieu, AUGEREAU Robin et MURUGESAPILLAI Vithursan

Département Sciences du Numérique - Première année 2022-2023 - Groupe VRM

Table des matières

Schéma du fonctionnement du Routeur avec Cache

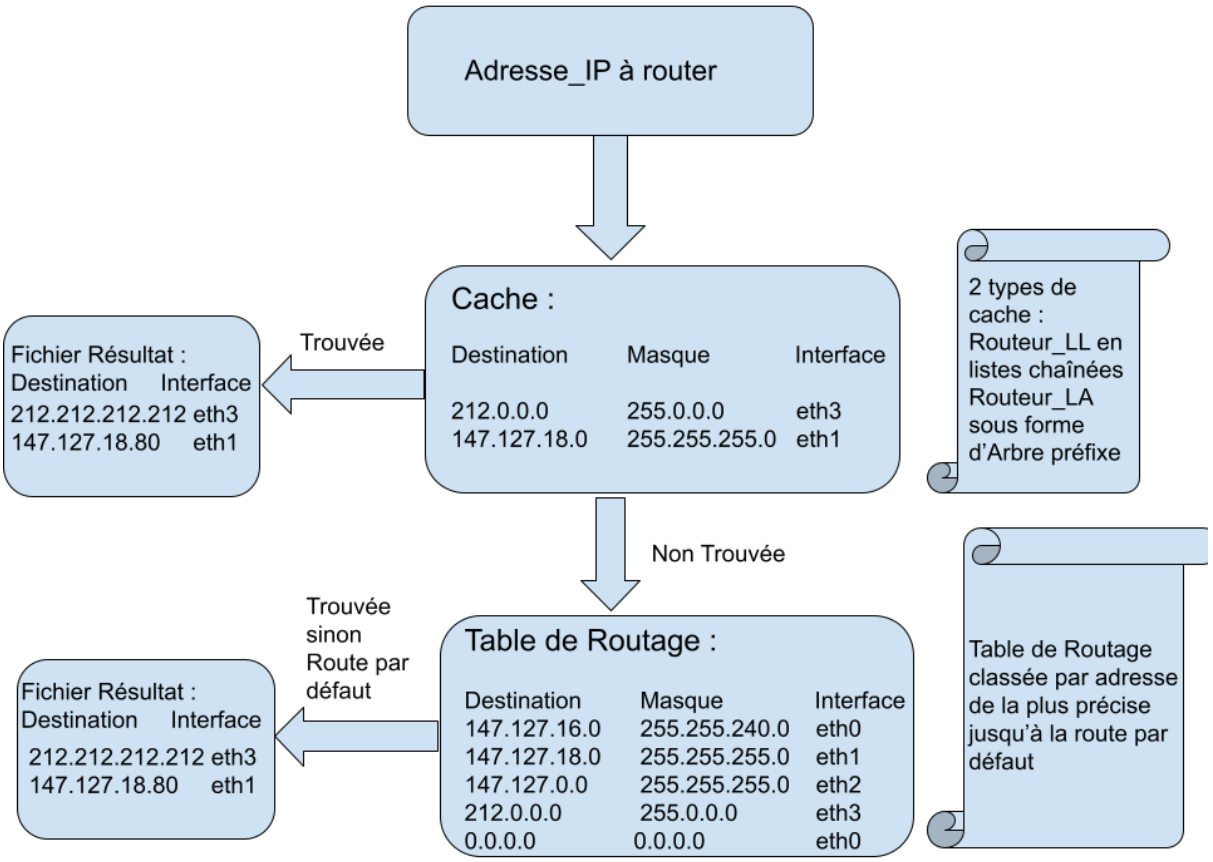


Schéma des programmes et des modules principaux

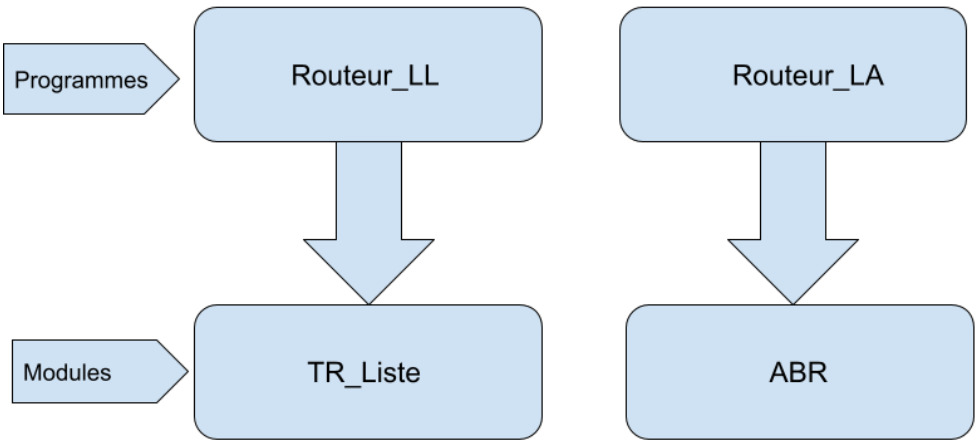
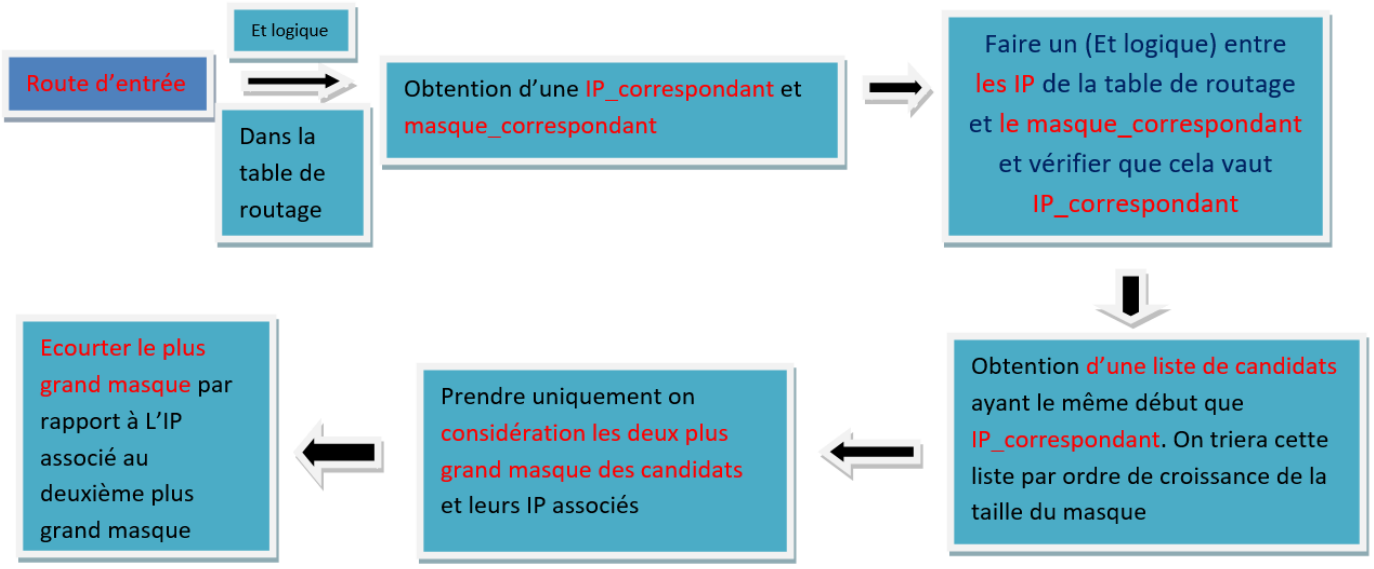


Schéma de la méthode d'optimisation du cache



Méthode d'écourtement du masque pour une optimisation du cache et ajout dans le cache

Les deux éléments en considérations dans la liste des candidats sont : avec (a, b, c', a, b, c', x, y, y' des octets de bits)

Destination	masque	interface
a.b.c.0	x.x.y.0	eth1
a.b.c'.0	x.x.y'.0	eth2

Indication :

c < c' avec le dernier bit à 1 de c' qui est plus loin du bit le plus fort que celui de c.

y < y' avec y' qui contient plus de bit à 1 que y.

On écourte alors y' par rapport a c de la manière suivante :

Avec c = a.b.10101000.0 (le dernier 1 en i)

y' deviendra y'modifié = a.b.111111000.0 (le dernier 1 en i+1)

Ajout dans le cache :

IP : (IP d'entrée and x.x.y'modifié.0)

Masque : x.x.y'modifié.0

Interface : interface_correspondant

Liste des modules et des sous programmes utilisés

- **TR_liste :**

Ce module permet de créer des listes afin de stocker à la fois une table de routage ou un cache sous ses 3 politiques d'utilisation

- **Initialiser_liste**(Sda: out T_liste)
- **Est_Vide_liste** (Sda : T_liste)
- **Cle_Presente**(Sda : T_liste ; IP : T_IP)
- **Taille_liste** (Sda : in T_liste)
- **Enregistrer_liste** (Sda : in out T_liste ; IP : in T_IP ; Interfac : in T_Interface ; Masque : in T_IP)

-- Supprimer une donnée repérée par son IP dans une Sda.

- **Supprimer_par_destination_liste** (Sda : in out T_liste ; Destination : in T_IP)
- **La_Donnee_Inter_liste** (Sda : in T_liste ; Destination : in T_IP)
- **La_Donnee_Masque_liste** (Sda : in T_liste ; Destination : in T_IP)
- **Vider_liste** (Sda : in out T_liste)

-- Incrémenter le compteur de toutes les règles de la table de routage afin de savoir laquelle a été la moins récemment utilisée

- **Incrementer_rec** (Sda : in T_liste)

-- Incrémenter le compteur d'une règle en la repérant par son IP (comparaison avec le et logique)

- **Incrementer_freq** (Sda : in T_liste ; IP : T_IP)

-- Renvoyer l'adresse ip de la règle de l'interface dont le compteur associé est le plus grand

- **Max** (Sda : in T_liste)

-- Renvoyer l'adresse ip de la règle de l'interface dont le compteur associé est le plus petit

- **Min** (Sda : in T_liste)

-- Supprimer la règle située à la fin de la liste (en queue)

- **Supprimer_fin**(Sda : in out T_liste)

-- Ajouter une règle au début de la liste

- **Ajouter_regle_liste**(Sda : in out T_liste ; IP : in T_IP ; Interfac : in T_Interface ; Masque : in T_IP)

-- Renvoie la dernière adresse IP de destination de la liste où il y a un match avec IPentree

- **TraiterIP_liste**(Sdad : in T_liste; IPentree : in T_IP)

-- Savoir si la fonction TraiterIP_liste renverra une règle correspondante présente dans le cache

- **Cle_Match_liste** (Sda : in T_liste ; Destination : in T_IP)

-- Appliquer un traitement (Traiter) pour chaque couple d'une Sda.

- **Pour_Chaque_liste** (Sda : in T_liste)

-- Affiche la position du dernier 1 en partant de la gauche de l'IP match de IPentree écrit en binaire, dont le masque est l'avant dernier le plus long (sert à l'optimisation du cache)

- **AjouterCache**(Sdad : in T_liste; IPentree : in T_IP)

- **ABR :**

Ce module permet de créer et manipuler le cache sous format arbre préfixe

- **Est_vide** (Arbre : in T_ABR)
- **Est_Feuille**(Arbre : in T_ABR)

-- Permet de renvoyer l'interface associée à un adresse IP

- **La_Donnee_Inter**(Arbre : in T_ABR, IP : in T_IP, Masque : in T_IP, Interface : in T_Interface)

-- Permet de savoir si une adresse IP est présente dans le cache (arbre)

- **Est_presente_LA** (Arbre : in T_ABR, IP : in T_IP, Masque : in T_IP, Interface : in T_Interface)
- **Initialiser_cache_Arbre** (Arbre : in T_ABR)
- **Ajouter** (Arbre : in out T_ABR, IP : in T_IP) --Ajout d'une règle au cache

-- Équilibrage de l'arbre avec suppression branches inutiles

- **Elaguer**(Arbre : out T_ABR)

-- Permet de connaître la taille du cache (Arbre)

- **Taille**(Arbre : in T_ABR)
- **Supprimer**(Arbre : in out T_ABR) -- Transformation d'une Feuille en Noeud

- **CLI :**

Ce module permet de gérer les lignes de commande et contient les différentes opérations sur le type adresse IP

-- Lire les commandes de l'utilisateur entrées en ligne de commande lors de l'exécution du programme

- **Lecture_lcommande** (stats_commande : in out Integer; politique : in out integer; taille_cache : in out Integer; fichier_table : in out Unbounded_String; fichier_paquet : in out Unbounded_String; fichier_resultat : in out Unbounded_String);

-- Convertir une règle au format "IP MASQUE INTERFACE" en trois chaînes de caractères

- **parse_regle**(Regle: Unbounded_String ; Front_Half : in out Unbounded_String ; Middle_Half : in out Unbounded_String ; Back_Half : in out Unbounded_String);

-- Retirer des caractères nuisibles à la lecture de la chaîne

- **Remove_char**(Back_Half : in out Unbounded_String)

-- Convertir une IP au format chaîne de caractères en adresse IP au format T_Adresse_IP

- **parse_ip**(Regle: Unbounded_String)

-- Convertir une adresse IP au format T_Adresse_IP en chaîne de caractères

- **deparse_ip**(IP1: in T_Adresse_IP)

-- Savoir si Masque1 < Masque2

- **inf**(Masque1 : T_Adresse_IP; Masque2 : T_Adresse_IP)

-- Renvoyer la dernière position de 1 dans Masque en partant de la gauche du masque

- **Pos_dernier_1**(Masque : in T_Adresse_IP) construire_masque(Taille : in Integer)

Variables et types utilisés

- **Pour tous les modules :**

```
type T_Adresse_IP est modulo 2**32  -- Former une adresse_IP de 32 bits
```

- **Pour le module TR_Liste :**

```
-- Création du type T_Cellule, pour faire un encapsulage technique des informations techniques
T_regle_d est un ENREGISTREMENT
        destination : T_IP = T_Adresse_IP
        masque : T_IP = T_Adresse_IP
        interface : T_Interface = chaîne de caractères
Fin ENREGISTREMENT              -- ce module est implanté génériquement

-- Création du type T_Cellule, un maillon de la liste chaînée
T_Cellule est nouveau ENREGISTREMENT
        regle : T_regle,
        compteur : Entier
        Suivant : T_liste
Fin ENREGISTREMENT

T_liste est pointeur vers T_Cellule

cache est T_liste
```

- **Pour le module de l'Arbre :**

```
type T_Regle est un ENREGISTREMENT de
        IP : T_IP
        Masque : T_IP
        Interface : T_Interface
Fin ENREGISTREMENT      -- ce module est implanté génériquement

type T_ABR est un ENREGISTREMENT de
        Regle : T_Regle
        SAG : T_ABR         -- SAG = Sous-Arbre-Gauche
        SAD : T_ABR         -- SAD = Sous-Arbre-Droit
        bool : Boolean      -- Boolean pour reconnaître un noeud (Faux) et une Feuille (Vrai)
        VA : Entier          -- VA = Valeur d'Ancienneté, compteur pour la politique LRU
Fin ENREGISTREMENT

type T_ABR a accès à T_Noeud
```

Structure générale commune aux routeurs

R0 : Faire fonctionner le routeurLL **R1 : Comment "Faire fonctionner le routeurLL" ?** Lire les lignes de commandes Initialiser_liste(TR) Lire et traiter les données du fichier des paquets Choisir d'afficher les statistiques	stats_commande (Entier) , politique (Entier), taille_cache (Entier) ,fichier_table (Unbounded String) ,fichier_resultat (Unbounded String) , fichier_paquet (Unbounded String) : in out TR (TR_liste) : out TR (TR_liste) : in nb_demande_route (Unbounded String) ,nb_faut_cache (Unbounded String) : in
R0 : Faire fonctionner le routeurLA **R1 : Comment "Faire fonctionner le routeurLA" ?** Lire les lignes de commandes Initialiser_cache_LCA (Arbre : T_ABR) Lire et traiter les données du fichier des paquets Choisir d'afficher les statistiques	stats_commande (Entier) , politique (Entier), taille_cache (Entier) ,fichier_table (Unbounded String) ,fichier_resultat (Unbounded String) , fichier_paquet (Unbounded String) : in out T_ABR out TR : in nb_demande_route (Unbounded String) ,nb_faut_cache (Unbounded String) : in
R2 : Comment "Lire les lignes de commandes" ? **Pour** Arg Dans 1..Nombre_Argument **Faire** **Si** Argument(Arg) = "-s" **Alors** stats_commande <- 1; *--Afficher statistiques* **SinonSi** Argument(Arg)= "-S" **Alors** stats_commande <- 0; *--Ne pas afficher statistiques* **SinonSi** Argument(Arg) = "-p" **Alors** **Si** Argument(Arg+1) = "FIFO" **Alors** politique <- 0; *--politique FIFO* **SinonSi** Argument(Arg+1) = "LRU" **Alors** politique <- 1; *--politique LRU* **Sinon** politique <- 2; *--politique LFU*	

FinSi
SinonSi Argument(Arg) = "-c" **Alors**
taille_cache <- Integer'value(Argument(Arg+1)); *--taille_cache du cache*
SinonSi Argument(Arg) = "-t" **Alors**
fichier_table <- To_Unbounded_String(Argument(Arg+1));
SinonSi Argument(Arg) = "-P" **Alors**
fichier_paquet <- To_Unbounded_String(Argument(Arg+1));
SinonSi Argument(Arg) = "-r" **Alors**
fichier_resultat <- To_Unbounded_String(Argument(Arg+1));
Sinon
Rien
FinSi
FinPour	

R2 : Comment "Lire et traiter les données du fichier des paquets"? *--Initialisation des listes LCA servant pour les politiques LRU et LFU* **Si** taille_cache /= 0 **Alors** Initialiser_liste(cache); --Initialiser_cache_Arbre(Arbre) avec routeurLA **FinSi** *--Ouverture des fichiers contenant les paquets et de sortie qui contiendra les bonnes interfaces* Ouvrir les fichiers de paquets et de résultats **Répéter** Numero_Ligne <- Integer (Line (Type_Fichier_Paquet)); *--On récupère le numéro de la ligne de commande dans le fichier paquets* Adresse_IP <- Lire(Type_Fichier_Paquet); Adresse_IP <- Trim (Adresse_IP, Both); Distinguer les valeurs prises par Adresse_ip **Jusqu'A** Fin_Of_File (Type_Fichier_Paquet); **Fin Faire;**	fichier_paquet (Unbounded String), fichier_resultat (Unbounded String) : in Adresse_ip (Unbounded String) : in
R2 : Comment "Choisir d'afficher les statistiques" ? **Si** stats_commande = 1 **Alors** Ecrire("Statistiques du routeur : "); Afficher_nb_demande_route(nb_demande_route); **Si** taille_cache /= 0 **Alors** taux_defaut_cache <- float(nb_defaut_cache) / float(nb_demande_route);	

Afficher_nb_defaut_cache(nb_defaut_cache); Afficher_taux_defaut_cache(taux_defaut_cache); **FinSi** **Sinon** Rien **FinSi**	
R3 : Comment "Ouvrir les fichiers de paquets et de résultat" ? Nom_Fichier_Paquet <- fichier_paquet; Nom_Fichier_Resultat <- fichier_resultat; Create (Type_Fichier_Resultat, Out_File, To_String (Nom_Fichier_Resultat)); Open (Type_Fichier_Paquet, In_File, To_String (Nom_Fichier_Paquet));	
R3 : Comment "Distinguer les valeurs prises par Adresse_ip*"?* *-- Commande table* **Si** Adresse_IP = To_Unbounded_String("table") **Alors** Afficher la table de routage dans la console *-- Commande fin* **SinonSi** Adresse_IP = To_Unbounded_String("fin") **Alors** Arrêter le programme *--Commande cache* **SinonSi** Adresse_IP = To_Unbounded_String("cache") **Alors** Afficher le cache dans la console **Sinon** Traiter une IP **FinSi**	Adresse_ip (Unbounded String) : in Adresse_ip (Unbounded String) : in Adresse_ip (Unbounded String) : in Adresse_ip (Unbounded String) : in
R4 : Comment "Afficher la table de routage dans la console " ? Ecrire("----------"); Ecrire("table (ligne" & integer'image(Numero_Ligne) & ")"); Ecrire("----------"); AfficherListe(TR); Ecrire("----------");	
R4 : Comment "Arrêter le programme"? Ecrire("----------"); Ecrire("fin (ligne" & Numero_Ligne & ")"); Ecrire("----------"); Quitter;	
R4 : Comment "Afficher le cache dans la console"? Ecrire("----------"); Ecrire("cache (ligne" & Numero_Ligne & ")"); Ecrire("----------");	

AfficherListe(cache); Ecrire("----------");	
R4 : Comment “Traiter une IP pour le routeur LL”? -- Comment “mettre à jour le cache (LRU) avec un routeur_LA“? cf Routeur_LA *--Incrémentation du compteur de nombre de routes* nb_demande_route <- nb_demande_route + 1; Adresse_IP <- Trim (Adresse_IP, Right); *--Traitement de l'adresse IP* IP_actuelle <- parse_IP(Adresse_IP); *---L'utilisateur a demandé un cache* **Si** taille_cache /= 0 **Alors** Utiliser le routeur avec un cache **Sinon** Retourner l’interface correspondante dans la table de routage **FinSi**	Ip_actuelle (T_Adresse_IP) : in Ip_actuelle (T_Adresse_IP) : in
R5 : Comment “Utiliser le routeur avec un cache” ? *--1er cas : l'IP a une règle correspondante dans le cache* **Si** PresCache(cache, IP_actuelle) **Alors** Mettre à jour le cache si l’IP a un match dans le cache Interf <- La_Donnee_Inter_liste(cache,DestTemp); IPdest <- Adresse_IP; Ecrire l'IP destination et l'interface utilisée dans le fichier résultat *--2eme cas : l'IP n’a de règle correspondante dans le cache* **Sinon** Mettre à jour le cache si l’IP n’a de match dans le cache **FinSi**	cache, IP_actuelle (T_Adresse_IP) : in, out Interf (Unbounded String) ,IPdest (Unbounded String) : out cache (TR_liste) ,IP_actuelle (T_Adresse_IP) : in out
R5 : Comment “*Retourner l’interface du match dans la table de routage”?* DestTemp <- TraitIPListe(TR,parse_ip(Adresse_IP)); Interf <- La_Donnee_Inter_liste(TR,DestTemp); IPdest <- Adresse_IP; Texte_temp <- IPdest & ' ' & Interf; Ecrire(Type_Fichier_Resultat, Texte_temp); Nouvelle_Ligne (Type_Fichier_Resultat);	
R6 : Comment “Mettre à jour le cache si l’IP a un match dans le cache”? **Selon** politique **Dans** 0 => *--cache FIFO : ne rien faire* DestTemp <- TraitIPListe(cache,IP_actuelle); 1 => *--cache LRU*	

estTemp <- TraitIPListe(cache,IP_actuelle); Incrementer_rec(cache); 2 => *--cache LFU* DestTemp <- TraitIPListe(cache,IP_actuelle); Incrementer_freq(cache,IP_actuelle); Autres => Rien **FinSelon**	cache (TR_liste) : in cache (TR_liste) : in
R6 : Comment "Ecrire l'IP destination et l'interface utilisée dans le fichier résultat" ? Texte_temp <- IPdest & " | " & Interf; Ecrire(Type_Fichier_Resultat, Texte_temp); Nouvelle_ligne (Type_Fichier_Resultat);	
R6 : Comment "Mettre à jour le cache si l'IP n'a de match dans le cache" ? *--Recherche de la règle correspondante directement dans la table de routage* DestTemp <- TraitIPListe(TR,IP_actuelle); *--Définition de la règle correspondante* Interf <- La_Donnee_Inter_liste(TR,DestTemp); Masquevf <- La_Donnee_Masque_liste(TR, DestTemp); IPdest <- Adresse_IP; Ecrire l'IP destination et l'interface utilisée dans le fichier résultat Construire la règle modifiée Mettre à jour le cache selon qu'il soit plein nb_defaut_cache <- nb_defaut_cache + 1;	 Interf, Ipdest : in cache (TR_liste) ,IP_actuelle (T_Adresse_IP) : in cache (TR_liste) , Masquevf (T_Adresse_IP) , IP_actuelle (T_Adresse_IP) : in out
R7 : Comment "*Construire la règle modfiée"?* **Si** Taille_liste(cache) = 0 **Alors** taille_masque <- Pos_dernier_1(IP_actuelle); **Sinon** taille_masque <- AjCache(cache,IP_actuelle); *--emplacement du dernier 1 dans l'avant dernier masque le plus long* **FinSi** Masquevf <- construire_masque(taille_masque); IP_actuelle <- (IP_actuelle ET Masquevf);	
R7 : Comment "*Mettre à jour le cache selon qu'il soit plein"?* *--Vérifions que le cache ne soit pas plein*	

Si Taille_liste(cache) < taille_cache + 1 **Alors** Ajouter_regle_liste(cache, IP_actuelle , Interf , Masquevf); nb_defaut_cache <- nb_defaut_cache + 1; *--Si le cache est plein, alors il faut distinguer selon les politiques* **Sinon** **Selon** politique **Dans** 0 => Mettre à jour le cache avec la politique FIFO 1 => Mettre à jour le cache avec la politique LRU 2 => Mettre à jour le cache avec la poliqiue LFU Autres => Rien **FinSelon** **FinSi**	cache (TR_liste) , IP_actuelle (T_Adresse_IP) : in out cache (TR_liste), IP_actuelle (T_Adresse_IP) : in out cache (TR_liste) , IP_actuelle (T_Adresse_IP) : in out
R8 : Comment "*Mettre à jour le cache avec la politique FIFO*"? Supprimer_fin(cache); Ajouter_regle_liste(cache,IP_actuelle , Interf , Masquevf);	
R8 : Comment "*Mettre à jour le cache avec la politique LRU*"? *--Supprimer la donnée la moins récemment utilisée du cache et de la liste* minc <- Min(cache); Supprimer_par_destination_liste(cache,minc); *--Ajouter la nouvelle règle au cache et à la liste* Ajouter_regle_liste(cache,IP_actuelle , Interf , Masquevf); Incrementer_rec(cache);	
R8 : Comment "*Mettre à jour le cache avec la politique LFU*"? *--Supprimer la donnée la moins fréquemment utilisée du cache et de la liste* maxc <- Max(cache); *--Ajouter la nouvelle règle au cache et à la liste* Supprimer_par_destination_liste(cache,maxc); Ajouter_regle_liste(cache,IP_actuelle , Interf , Masquevf); Incrementer_freq(cache, IP_actuelle);	
Procédures et Fonctions utilisées dans les raffinages : **R0 :** Initialiser une table de routage au format TR_liste **R1 : Comment " Initialiser une table de routage au format TR_liste" ?** PROCEDURE Initialiser_liste(TR) (définition dans le module TR_liste)	

R0 : Afficher le nombre de défauts de cache sur demande
R1 : Comment “Afficher le nombre de défauts de cache sur demande“?

```
PROCEDURE Afficher_nb_defaut_cache(nb_defaut_cache : Integer) est
Debut
Ecrire("Nombre de défauts du cache : " & nb_defaut_cache'Image);
Nouvelle_ligne;
Fin Afficher_nb_defaut_cache;
```

R0 : Afficher le nombre de demandes de route sur demande
R1 : Comment “Afficher le nombre de demandes de route sur demande”?

```
PROCEDURE Afficher_nb_demande_route(nb_demande_route : Integer) est
Debut
Ecrire("Nombre de demandes de route : " & nb_demande_route'Image);
Nouvelle_ligne;
Fin Afficher_nb_demande_route;
```

R0 : Afficher le taux de défaut du cache sur demande
R1 : Comment “Afficher le taux de défaut du cache sur demande”?

```
PROCEDURE Afficher_taux_defaut_cache(taux_defaut_cache : Float) est
Debut
Ecrire("Taux de défault du cache :"&taux_defaut_cache'Image);
Nouvelle_ligne;
Fin Afficher_taux_defaut_cache;
```

Spécification du programme routeur_LL

Fonctions/Procédures utilisées dans le module TR_liste :

R0 : Initialiser une liste
R1 : Comment " Initialiser une liste" ?

```
Procedure Initialiser_liste(Sda: out T_liste) est
Debut
   Sda<-Null;
Fin Initialiser_liste;
```

R0 : Affirmer si une liste est vide
R1 : Comment " Affirmer si une liste est vide" ?

```
fonction Est_Vide_liste (Sda : T_liste) return Boolean est
Debut
     return (Sda = null);
Fin Est_Vide_liste;
```

R0 : Retourner la taille de la liste
R1 : Comment " Retourner la taille de la liste" ?

```
fonction Taille_liste (Sda : in T_liste) return Entier est
Debut
  Si Sda = Null Alors
   return 0;
  Sinon
   return 1 + Taille_liste (Sda.all.Suivant);
  FinSi
Fin Taille_liste;
```

R0 : Enregistrer une adresse
R1 : Comment "Enregistrer une adresse" ?

```
   Procedure Enregistrer_liste (Sda : in out T_liste ; IP : in T_IP ; Interfac : in T_Interface ;
Masque : in T_IP) est
       Nouv_Cellule : T_liste ;
       Regleb : T_Regle;
   Debut
       Si Est_Vide_liste(Sda) Alors
           Regleb.interfac <-Interfac ;
```

```
            Regleb.IP <-IP;
            Regleb.Masque <- Masque ;
            Nouv_Cellule <-new T_Cellule'(Regleb, 0 ,Sda);
            Sda <-Nouv_Cellule;
        Sinon
            Enregistrer_liste(Sda.all.Suivant, IP, Interfac , Masque);
        FinSi
    Fin Enregistrer_liste;
```

R0 : Renvoyer l'interface correspondante à une IP

R1 : Comment "Envoyer l'interface correspondante à une IP" ?

```
  fonction La_Donnee_Inter_liste (Sda : in T_liste ; Destination : in T_IP) return T_Interface
est
  Debut
      Si Est_Vide_liste(Sda) Alors
        raise Cle_Absente_Exception;
      SinonSi Sda.all.Regle.IP = Destination Alors
        return Sda.Regle.interfac;
      Sinon
        return La_Donnee_Inter_liste(Sda.all.suivant,Destination);
      FinSi
  Fin La_Donnee_Inter_liste;
```

R0 : Renvoyer le masque correspondante à une IP

R1 : Comment "Renvoyer le masque correspondante à une IP" ?

```
  fonction La_Donnee_Masque_liste (Sda : in T_liste ; Destination : in T_IP) return T_IP est
  Debut
      Si Est_Vide_liste(Sda) Alors
        raise Cle_Absente_Exception;
      SinonSi Sda.all.Regle.IP = Destination Alors
        return Sda.all.Regle.Masque;
      Sinon
        return La_Donnee_Masque_liste(Sda.all.suivant,Destination);
      FinSi
  Fin La_Donnee_Masque_liste;
```

R0 : Supprimer une règle de la liste en fonction de son IP

R1 : Comment "Supprimer une règle de la liste en fonction de son IP" ?

```
  Procedure Supprimer_par_destination_liste (Sda : in out T_liste ; Destination : in T_IP)
est
    Miroir :  T_liste;
  Debut
      Si Est_Vide_liste(Sda) Alors
        Null;
      SinonSi iSda.all.Regle.IP = Destination Alors
        Miroir<-Sda;
        Sda<-Sda.all.Suivant;
        Free_liste(Miroir);
      Sinon
        Supprimer_par_destination_liste(Sda.all.Suivant, Destination);
      FinSi
  Fin Supprimer_par_destination_liste;
```

R0 : Vider une liste

R1 : Comment " Vider une liste" ?

```
  Procedure Vider_liste (Sda : in out T_liste) est
    Miroir :  T_liste <-Sda;
  Debut
    Si Est_Vide_liste(Sda) Alors
      Null;
    Sinon
      Miroir<-Sda;
      Sda<-Sda.all.Suivant;
      Free_liste(Miroir);
      Vider_liste(Sda);
    FinSi
  Fin Vider_liste;
```

R0 : Incrémenter la liste pour tous ses maillons

R1 : Comment " Incrémenter la liste pour tous ses maillons" ?

```
  Procedure Incrementer_rec (Sda : in out T_liste) est
  Debut
    Si Est_Vide_liste(Sda) Alors
      Null;
    Sinon
```

```
      Sda.all.Compteur <-Sda.all.Compteur +1 ;
      Sda <-Sda.all.Suivant;
    FinSi
  Fin Incrementer_rec;
```

R0 : Incrémenter le compteur de la règle qui possède la destination IP

R1 : Comment “Incrémenter le compteur de la règle qui possède la destination IP” ?

```
  Procedure Incrementer_freq (Sda : in out T_liste ; IP : T_IP) est
  Debut
    TantQue Est_Vide_liste(Sda) Faire
      Si IP = Sda.all.Regle.IP Alors
        Sda.all.Compteur <-Sda.all.Compteur +1 ;
      FinSi
      Sda <-Sda.all.Suivant ;
    Fin TantQue;
  Fin Incrementer_freq;
```

R0 : Renvoyer l’adresse IP associée au compteur max

R1 : Comment “ Renvoyer l’adresse IP associée au compteur max” ?

```
  fonction Max (Sda : in T_liste) return T_IP est
  compteur_temp : Integer <-Sda.all.Compteur ;
  ip_temp : T_IP <-Sda.all.Regle.IP;
  Miroir : T_liste <-Sda ;
  Debut
    TantQue Miroir.all.Suivant /= Null Faire
      Si Miroir.all.Compteur > compteur_temp Alors
        compteur_temp <-Miroir.all.Compteur ;
        ip_temp <-Miroir.all.Regle.IP;
      FinSi
      Miroir <-Miroir.all.Suivant ;
    Fin  TantQue;
    return ip_temp ;
  Fin Max;
```

R0 : Renvoyer l'adresse IP associée au compteur min

R1 : Comment "Renvoyer l'adresse IP associée au compteur min" ?

```
  fonction Min (Sda : in T_liste) return T_IP est
  compteur_temp : Integer <-Sda.all.Compteur ;
  ip_temp : T_IP <-Sda.all.Regle.IP;
  Miroir : T_liste <-Sda ;
  Debut
    TantQue Miroir.all.Suivant /= Null Faire
      Si Miroir.all.Compteur < compteur_temp Alors
        compteur_temp <-Miroir.all.Compteur ;
        ip_temp <-Miroir.all.Regle.IP;
      FinSi
      Miroir <-Miroir.all.Suivant ;
    Fin TantQue;
    return ip_temp ;
  Fin Min;
```

R0 : Ajouter une règle au début de la liste

R1 : Comment "Ajouter une règle au début de la liste" ?

```
  Procedure Ajouter_regle_liste(Sda : in out T_liste ; IP : in T_IP ; Interfac : in T_Interface ;
Masque : in T_IP) est
    Nouv_Cellule : T_liste ;
    Regleb : T_Regle;
    Debut
      Regleb.interfac <-Interfac ;
      Regleb.IP <-IP;
      Regleb.Masque <- Masque ;
      Nouv_Cellule <-new T_Cellule'(Regle => Regleb, Compteur => 0, Suivant => Sda);
      Sda <-Nouv_Cellule;
  Fin Ajouter_regle_liste;
```

R0 : Renvoyer l'IP du match avec le masque le plus long

R1 : Comment "Renvoyer l'IP du match avec le masque le plus long" ?

```
  fonction TraiterIP_liste(Sdad : in T_liste; IPentree : in T_IP) return T_IP est
  MasqueTemp : T_IP;
  IPtemp : T_IP;
  Miroir : T_liste <-Sdad;
  Premiere : Boolean <-False; ---savoir si une adresse a déjà été affectée à MasqueTemp
  Debut
    Pour I allant 1..Taille_liste(Sdad) Faire
```

```
        Si non Premiere Alors
            Si comparer(IPentree, Miroir.all.Regle.Masque, Miroir.all.Regle.IP) Alors
              Premiere <-True;
              MasqueTemp <-Miroir.all.Regle.Masque ;
              IPtemp <-Miroir.all.Regle.IP ;
            FinSi
        Sinon
          Si comparer(IPentree, Miroir.all.Regle.Masque, Miroir.all.Regle.IP) and
inf(MasqueTemp,Miroir.all.Regle.Masque) Alors
            MasqueTemp <-Miroir.all.Regle.Masque ;
            IPtemp <-Miroir.all.Regle.IP ;
          FinSi
        FinSi
        Miroir <-Miroir.all.Suivant;
      Fin Pour;
      return IPtemp;
    Fin TraiterIP_liste;
```

R0 : Renvoyer si l'IP a un match dans la liste

R1 : Comment "Renvoyer si l'IP a un match dans la liste" ?

```
  fonction Cle_Match_liste (Sda : in T_liste ; Destination : in T_IP) return Boolean est
  Debut
      Si Sda = Null Alors
        return False;
      SinonSi comparer(Destination, Sda.all.Regle.Masque, Sda.all.Regle.IP) Alors
        return True ;
      SinonSi Sda.all.Suivant /= Null Alors
        return Cle_Match_liste(Sda.all.Suivant, Destination);
      FinSi
     return False;
  Fin Cle_Match_liste;
```

R0 : Répéter un traitement pour chaque maillon de la liste

R1 : Comment "Répéter un traitement pour chaque maillon de la liste" ?

```
  Procedure Pour_Chaque_liste (Sda : in T_liste) est
  Debut
    Si Sda=Null Alors
      Null;
    Sinon
      Afficher(Sda.all.Regle.IP, Sda.all.Regle.Masque, Sda.all.Regle.interfac);
```

```
      Pour_Chaque_liste(Sda.all.Suivant);
    FinSi
  Fin Pour_Chaque_liste ;
```

R0 : Supprimer le dernier maillon de la liste
R1 : Comment “Supprimer le dernier maillon de la liste” ?

```
  Procedure Supprimer_fin(Sda : in out T_liste) est
  Debut
    Si Sda.all.Suivant = Null Alors
      Free_liste(Sda);
    Sinon
      Supprimer_fin(Sda.all.Suivant);
    FinSi
  Fin Supprimer_fin;
```

R0 : Renvoyer si une règle est présente contenant une certaine IP
R1 : Comment “Renvoyer si une règle est présente contenant une certaine IP” ?

```
 fonction Cle_Presente(Sda : T_liste ; IP : T_IP) return Boolean est
  Debut
    Si Sda = Null Alors
      return False;
    SinonSi Sda.all.Regle.IP = IP Alors
      return True ;
    SinonSi Sda.all.Suivant /= Null Alors
      return Cle_Presente(Sda.all.Suivant, IP);
    FinSi
    return False;
  Fin Cle_Presente;
```

-- Pour l’optimisation du cache
R0 : Construire la nouvelle règle à ajouter au cache
R1 : Comment “Construire la nouvelle règle à ajouter au cache” ?

```
  fonction AjouterCache(Sdad : in T_liste; IPentree : in T_IP) return Integer est
    IPtempM : T_IP;
    MasqueTempM : T_IP;
    IPtemp : T_IP;
```

```
    MasqueTemp : T_IP;
    PremiereM : Boolean <-False; ---savoir si une adresse a déjà été affectée à
MasqueTempM et IPtempM
    Premiere : Boolean <-False; ---savoir si une adresse a déjà été affectée à MasqueTemp
et IPtemp
    Miroir : T_liste <-Sdad;
    compteur : Integer;
  Debut
    Pour I allant de 1..Taille_liste(Sdad) Faire
      Si not Premiere Alors
          Si comparer(IPentree, Miroir.all.Regle.Masque, Miroir.all.Regle.IP) Alors
            MasqueTemp <-Miroir.all.Regle.Masque ;
            IPtemp <-Miroir.all.Regle.IP ;
            Premiere <-True;
          FinSi
      Put_Line("Premiere");
      Sinon
        Si comparer(IPentree, Miroir.all.Regle.Masque, Miroir.all.Regle.IP) and
inf(MasqueTemp,Miroir.all.Regle.Masque) Alors
          Si non PremiereM Alors
            Si inf(MasqueTemp, Miroir.all.Regle.Masque) Alors
              MasqueTempM <-Miroir.all.Regle.Masque ;
              IPtempM <-Miroir.all.Regle.IP ;
            Sinon
              MasqueTempM <-MasqueTemp;
              IPtempM <-IPtemp;
              MasqueTemp <-Miroir.all.Regle.Masque ;
              IPtemp <-Miroir.all.Regle.IP ;
            FinSi
            PremiereM <-True;
          SinonSi comparer(IPentree, Miroir.all.Regle.Masque, Miroir.all.Regle.IP) and
inf(MasqueTempM,Miroir.all.Regle.Masque) Alors
            Si inf(MasqueTempM,MasqueTemp) Alors
              MasqueTempM <-MasqueTemp;
              IPtempM <-IPtemp;
              MasqueTemp <-Miroir.all.Regle.Masque ;
              IPtemp <-Miroir.all.Regle.IP ;
            Sinon
              MasqueTempM <-Miroir.all.Regle.Masque ;
              IPtempM <-Miroir.all.Regle.IP ;
```

```
            FinSi
          FinSi
        FinSi
      FinSi
      Miroir <-Miroir.all.Suivant;
     Fin Pour;
     Si not PremiereM Alors
      compteur <-Pos_dernier_1(IPtempM);
      Sinon
      compteur <-Pos_dernier_1(IPtemp);
      FinSi
     return compteur;
Fin AjouterCache;
```

Spécification du programme routeur_LA

Algorithmes sous la forme de Raffinages	Flots de données
R4 : Comment "mettre à jour le cache (LRU) avec un routeur_LA"?	
Si l'adresse_IP est présente dans le cache **Alors**	Arbre : in T_ABR, IP : in T_IP
Renvoyer la règle	Arbre : in T_ABR, IP : in T_IP , Masque : T_IP in , Interface : in T_Interface
-- Initialiser la valeur d'ancienneté pour la politique LRU	
VA = 0	
Mettre à jour les compteurs VA	
Sinon	
-- Si le cache n'est pas plein	
Si Taille (Arbre) /= Capacite_cache **Alors**	Capacite_cache : in Entier
Ajouter la règle dans le cache	Arbre : in out T_ABR, IP : in T_IP , Masque : T_IP in , Interface : in T_Interface
VA = 0	
Mettre à jour les compteurs VA	
Sinon	
Rechercher l'adresse IP utilisée la moins récemment dans le cache	Arbre : in T_ABR
Supprimer cette adresse_IP	Arbre : in out T_ABR
Ajouter la nouvelle règle dans le cache	Arbre : in out T_ABR, IP : in T_IP, Masque : T_IP in, Interface : in T_Interface
VA = 0	
Mettre à jour les compteurs VA	
FinSi	
FinSi	
R5 : Comment "savoir si l'adresse_IP est présente dans le cache ?"	
--on procède d'une manière récursive	
Si Est_vide(Arbre) **Alors**	
Lève l'exception Cle_Absente	
Sinon	
-- S'il s'agit d'un noeud et que le premier bit de l'IP est un 0, on va vers la gauche	
Si non Est_Feuille(Arbre) et (IP et POIDS_FORT) < 1 **Alors**	IP : in T_IP

-- on décale ensuite le bit à comparer pour que ce soit le suivant lorsque la fonction s'appliquera à nouveau IP <- IP*2 Est_Presente_LA(Arbre.^SAG , IP) --Si le premier bit est un 1 et qu'il s'agit d'un noeud **SinonSi** non Est_Feuille(Arbre) et (IP et POIDS_FORT) = 1 **Alors** IP<- IP*2 Est_Presente_LA(Arbre.^SAD , IP) --Feuille et que l'IP correspond **SinonSi** Est_Feuille(Arbre) et Arbre.^Regle.IP = IP **Alors** Ecrire(Vrai) **Sinon** Ecrire(Faux) **FinSi** **FinSi**	Arbre.^SAG : in T_ABR Arbre.^SAD : in T_ABR
R5 : Comment "Renvoyer la règle du cache, l'interface correspondant à l'adresse_IP ?" **Si** Est_vide(Arbre) **Alors** Lève l'exception Cle_Absente **Sinon** **Si** non Est_Feuille(Arbre) et (IP et POIDS_FORT) < 1 **Alors** IP <- IP*2 La_Donnee_Inter(Arbre.^SAG , IP) **SinonSi** non Est_Feuille(Arbre) et (IP et POIDS_FORT) = 1 **Alors** IP<- IP*2 La_Donnee_Inter(Arbre.^SAD , IP) **SinonSi** Est_Feuille(Arbre) et Arbre.^Regle.IP = IP **Alors** Ecrire(Arbre.^Regle.Interface) **Sinon**	

Lève l'exception Cle_Absente **FinSi** **FinSi** --Ce raffinage est long à cause de toutes les possibilités à prendre en compte **R5 : Comment "Ajouter une règle dans le cache ?"** --Création d'une nouvelle Feuille avec un Boolean Vrai **Si** Est_Vide(Arbre) **Alors** Création d'une Feuille **Sinon** -- Si le bit de poids fort vaut 0 on se retrouve à gauche du tableau **Si** (IP et POIDS_FORT) < 1 Alors -- Si l'IP de la feuille ne correspond pas alors il faut créer un nouveau Noeud **Si** Est_Feuille(Arbre) et Arbre.^Regle.IP =/ IP **Alors** Creation d'un noeud à gauche (Arbre.^SAG) IP <- IP*2 Ajouter(SAG) --Si l'IP correspond le boolean devient vrai et le Noeud se transforme en Feuille **SinonSi** Est_Feuille(Arbre) et Arbre.^Regle.IP = IP **Alors** Arbre.^bool = Vrai **Sinon** IP <- IP*2 Ajouter(SAG) **FinSi** --idem à droite **Sinon** **Si** Est_Feuille(Arbre) et Arbre.^Regle.IP =/ IP **Alors** Creation d'un noeud à droite (Arbre.^SAD) IP <- IP*2 Ajouter(SAD) **SinonSi** Est_Feuille et Arbre.^Regle.IP = IP **Alors** Arbre.^bool = Vrai **Sinon** IP <- IP*2 Ajouter(SAD) **FinSi** **FinSi** **FinSi**	Arbre : out T_ABR, IP : in T_IP , Masque : T_IP in , Interface : in T_Interface Arbre.^SAG : out T_ABR, IP : in T_IP , Masque : T_IP in , Interface : in T_Interface Arbre.^bool : out Boolean Arbre.^SAD : out T_ABR, IP : in T_IP , Masque : T_IP in , Interface : in T_Interface Arbre.^bool : out Boolean

R5 : Comment “Rechercher l’adresse IP la moins utilisée dans le cache ?”

```
-- le but étant de parcourir toutes les feuilles
Pour i allant de 1 à Capacite_Cache Faire
    -- Recherche de la VA la plus élevée parmi les feuilles
    Si Est_Feuille(Arbre) = Vrai Alors
        Si Arbre.^VA > VA_max Alors
            VA_max <- Arbre.^VA
            Regle_max  <- Arbre.^Regle
        Sinon
            Rien
    Sinon
        Rien
    FinSi
FinPour
```

R5 : Comment “Mettre à jour les compteurs VA ?”

```
Arbre.^VA <- Arbre.^VA + 1
--Utilise la récursivité pour traiter tout l’arbre
Maj_VA (Arbre.^SAG)
Maj_VA(Arbre.^SAD)
```

R5 : Comment “Supprimer l'adresse IP utilisée la moins récemment ?”

```
-- On réutilise la fonction qui recherche l’IP ancienne
Regle_max <- Rechercher_IP_Ancienne(Arbre)
Arbre.^bool <- False
Elaguer(Arbre)
```

R6 : Comment “ Créer une feuille ?”

```
-- On définit une variable Regleb de type T_Regle pour stocker les informations relatives à la règle à mettre dans la feuille
Regleb.interfac <- Interface
Regleb.IP <- IP
Regleb.Masque <- Masque
-- Pour un noeud le boolean est Vrai
Arbre <- Nouv T_Noeud'(Regleb, null, null, Vrai, 0)
```

R6 : Comment “ Creer un noeud “ ?

```
Regleb.interfac <- Interface
Regleb.IP <- IP
Regleb.Masque <- Masque
```

-- Pour un noeud le boolean est Faux Arbre <- Nouv T_Noeud'(Regleb, null, null, Faux, 0)	
Fonctions/Procédures utilisées dans le module ABR :	Données en entrée des fonctions/procédures :
#Est_vide **R0 : Vérifier que l'arbre est vide** **R1 : Comment "vérfiier que l'arbre est vide"?**	
Ecrire (Arbre = null)	Arbre : in T_ABR
#Est_Feuille **R0 : Vérfiier que l'arbre (noeud) est une feuille** **R1 : Comment "vérfiier que l'arbre (noeud) est une feuille"?**	
Si Est_Vide(Arbre) **Alors** Ecrire (Faux) **SinonSi** Est_Vide(Arbre.^SAG) et Est_Vide(Arbre.^SAD) **Alors** Ecrire (Vrai) **Sinon** Ecrire (Faux) **FinSi**	Arbre : in T_ABR, Arbre.^SAG : in T_ABR, Arbre.^SAD : in T_ABR
#Taille **R0 : Renvoyer la Taille de l'arbre** **R1 : Comment " Renvoyer la Taille de l'arbre ?"**	
Si Est_Vide(Arbre) **Alors** Taille <- 0 **Sinon** Taille <- Taille(Arbre.^SAG) + Taille(Arbre.^SAD) + 1 **FinSi** Ecrire(Taille)	Arbre : in T_ABR
-- procédure difficile à réaliser **#Elaguer** **R0 : Elaguer l'arbre en supprimant les noeuds inutiles qui le déséquilibre** **R1 : Comment " Elaguer l'arbre en supprimant les noeuds inutiles qui le déséquilibre ?"**	
Pour i allant de 0 à Capacite_Cache **Faire** Parcourir une branche jusqu'au bout	Arbre : in out T_ABR

-- S'il n'y a que des noeuds reliés à aucun autre noeud alors on peut supprimer la branche **Si** non Est_Feuille(Arbre) et non Arbre.^SAG **Alors** Free (Arbre) Free (SAG) Free (SAD) **FinSi** **FinPour** --procédure incomplète **#Initialiser_cache_Arbre** **R0 : Initialiser l'Arbre** **R1 : Comment "initialiser l'Arbre"?** Arbre <- Nouv T_Noeud'(Regle, null, null, Vrai, 0)	Arbre : out T_ABR