Types
Type Politique_de_tri **est** énumération (FIFO, LRU, LFU)

Type T_Arguments **est** enregistrement (Taille: Entier;
Politique: Politique_de_tri;
Statistiques: Booléen;
Routage: Unbounded_String;
Paquets: Unbounded_String;
Resultats: Unbounded_String)

Type T_Adresse_IP **est** mod 2**32

Type T_LCA **est** pointeur sur T_Cellule;

Type T_Cellule **est** enregistrement (Destination: T_Adresse_IP;
Masque: T_Adresse_IP;
eth: Unbounded_String;
Dernière_Utilisation: Entier;
Nb_Utilisation: Entier;
Suivant: T_LCA)

Type T_Donnee **est** enregistrement (Destination: T_Adresse_IP;
Masque: T_Adresse_IP;
eth : Unbounded_String)

Utilitaire

R5 : Comment “Convertir un T_Adresse_Ip en String” ?*

Initialiser un string de 0 **in:** destination : T_Adresse_IP; **out:** chaine_binaire : String
destination_bis = destination;
Pour numero_bit de Taille (Chaine_Binaire)) à 1 **faire**
    Si destination_bis mod 2 = 1 **alors**
        chaine_binaire(numero_bit) = '1';
    Fin Si
    destination_bis = destination_bis / 2;
Fin pour
Retourner chaine_binaire;

R6 : Comment “Initialiser un string de 0” ?

chaine_binaire = String (1 à Taille (destination));
Pour i de 1 à Taille (destination) **faire**
    chaine_binaire(i) = ‘0’;
Fin Pour

R5 : Comment “Convertir un String en T_Adresse_IP” ?

```
resultat = 0;
Pour i de 1 à Taille (binaire) faire
        caractere = String (binaire)(i);
        chaine = Unbounded_String (“0” & caractere);
        resultat = resultat * 2 + T_Adresse_IP (String (chaine));
Fin Pour
Retourner resultat;
```

R6 : Comment “Compter le nombre de 1 dans deux chiffres binaires” ?

```
Si m1 and 2**i /= 0 alors
        compteur1 := compteur1 + 1;
Sinon si m2 and 2**i /= 0 alors
        compteur2 := compteur2 + 1;
Sinon
        Rien;
Fin si
```

R4 : Comment “Obtenir le ième octet” ?

```
Selon i dans
        1 => a = adresse / 256**3;
        2 => a = (adresse mod 256**3) / 256**2;
        3 => a = ((adresse mod 256**3) mod 256**2) / 256;
        4 => a = ((adresse mod 256**3) mod 256**2) mod 256;
        Autres => a = 0
Fin Selon

Retourner Entier (a);
```

R6 : Comment “Calculer la longueur du masque” ?

```
compteur = 0;
i = 0;
Tant Que i <= 31 ET (masque ET 2**(31-i)) /= 0 faire
        compteur = compteur + 1;
        i = i + 1;
Fin Tant Que
Retourner compteur;
```

R6 : Comment “Créer un masque” ?

```
binaire = “”;
Pour i de 1 à longueur faire
        binaire = binaire & “1”;
Fin Pour
Pour i de longueur à 31 faire
        binaire = binaire & “0”;
```

Fin Pour
Retourner Convertir un string en T_Adresse_IP (binaire);

RAFFINAGE COMMUN

R0 : Comment “Faire fonctionner le routeur” ?

Initialiser les paramètres du routeur	**out:** Arguments : T_Arguments
Lire la table de routage	**in**: Arguments; **out**: Table : T_LCA
Traiter les instructions du fichier paquets	**in:** Arguments, Table

R1 : Comment “Initialiser les paramètres du routeur” ?

Initialiser les paramètres du routeur par défaut **out:** Arguments

```
i =1;
Tant que i <= Nombre_Arguments faire
        Selon Lire_Arguments(i) dans
                ‘-c’ =>
                        Arguments.taille = Lire_Arguments(i+1);
                        i = i + 2;
                ‘-P’ =>
                        Arguments.politique = Lire_Arguments(i+1);
                        i = i + 2;
                ‘-s’ =>
                        Arguments.statistiques = Vrai;
                        i = i +1;
                ‘-S’ =>
                        Arguments.statistiques = Faux;
                        i = i +1;
                ‘-t’ =>
                        Arguments.routage = Lire_Arguments(i+1);
                        i = i + 2;
                ‘-p’ =>
                        Arguments.paquets = Lire_Arguments(i+1);
                        i = i + 2;
                ‘-r’ =>
                        Arguments.resultats = Lire_Arguments(i+1);
                        i = i + 2;
        Fin Selon
Fin Tant Que
```

R1 : Comment “Lire la table de routage” ?

Initialiser_LCA (Table);

ligne = “”;
Tant Que ligne n’est pas vide **faire**
    ligne = Lire_Ligne (Arguments.routage);
    Donnee = Interpréter une ligne de la table de routage **in:** Table, ligne
    Enregistre_LCA (Donnee, Table);
Fin Tant Que

R1 : Comment “Traiter les instructions du fichier paquets” ?

Initialiser_LCA (Cache);
Arrêt = Faux;
i_ligne_paquet = 1;
Tant Que ligne n’est pas vide ET NON Arrêt **faire**
    ligne = Lire_Ligne (Arguments.paquets);
    Interpréter une ligne du fichier paquets **in:** Table, ligne, i_ligne_paquet, Arguments; **in out:** Cache
    i_ligne_paquet = i_ligne_paquet + 1;
Fin Tant Que

R2 : Comment “Initialiser les paramètres du routeur par défaut” ?

Arguments.Taille = 10;
Arguments.Politique = FIFO;
Arguments.Statistiques = Vrai;
Arguments.Routage = “table.txt”;
Arguments.Paquets = “paquets.txt”;
Arguments.Resultats = “resultats.txt”;

R2 : Comment “Interpréter une ligne de la table de routage” ?

Pour i de 1 à Taille (ligne) **faire**
    Interpréter le ième caractère de la table de routage **in:** ligne, i : Entier;
    out: destination : T_Adresse_IP, masque : T_Adresse_IP, eth : String
Fin Pour
Donnee = (destination, masque, eth);

R2 : Comment “Interpréter une ligne du fichier paquets” ?

Nb_defauts_cache = 0;
Nb_demandes_routes = 0;
Taux_defauts_cache = 0.0;

Selon ligne **faire**
    “table” =>
        Pour_Chaque_LCA (Table, Afficher la route);
    “cache” =>
        Pour_Chaque_LCA (Cache, Afficher la route);
    “stat” =>
        Afficher les statistiques du cache **in:** Nb_defauts_cache, Nb_demandes_routes, Taux_defauts_cache

“fin” =>
Arrêt = Vrai;
“” => Rien;
défaut =>
Trouver la route correspondant au paquet **in:** ligne, i_ligne_paquet, Arguments, Table;
in out: Nb_defauts_cache, Nb_demandes_routes, Taux_defauts_cache, Cache
– raffinages spécifiques

Fin Selon

R3: Comment “Afficher la route” ?

Pour i de 1 à 3 **faire**
longueur = Taille (Unbounded_String (Obtenir le ième octet (destination, i)))
ligne = ligne & (Unbounded_String (Obtenir le ième octet (destination, i))) (2 à longueur) & “.”
Fin Pour
longueur = Taille (Unbounded_String (Obtenir le ième octet (destination, 4)))
ligne = ligne & (Unbounded_String (Obtenir le ième octet (destination, 4))) (2 à longueur) & “ “

Pour i de 1 à 3 **faire**
longueur = Taille (Unbounded_StringObtenir le ième octet (masque, i)))
ligne = ligne & (Unbounded_StringObtenir le ième octet (masque, i))) (2 à longueur) & “.”
Fin Pour
longueur = Taille (Unbounded_StringObtenir le ième octet (masque, i)))
ligne = ligne & (Unbounded_StringObtenir le ième octet (masque, 4))) (2 à longueur) & “ “ & eth

Afficher (ligne);
Afficher_Nouvelle_Ligne;

R3 : Comment “Afficher les statistiques du cache” ?

Afficher (« Nombre de défauts de cache : Nb_defauts_cache. »)
Afficher (« Nombre de demandes de routes : Nb_demandes_routes. »)
Afficher (« Taux de défauts de cache : Taux_defauts_cache. »)

R3 : Comment “Interpréter le ième caractère d’une ligne de la table de routage” ?

Si ligne (i) = ‘.’ OU ligne (i) = ‘ ‘ **alors**
Interpréter la destination et le masque d’une ligne de la table de routage **out:** destination, masque
Sinon Si i = Taille (ligne) **alors**
Interpréter l’interface d’une ligne de la table de routage **out:** eth
Sinon
Rien;
Fin Si

R4 : Comment “Interpréter la destination et le masque d’une ligne de la table de routage” ?

Selon nb_points_parcourus **dans**
0 => destination := destination + ligne (i_point+1 à i-1)

1, 2, 3 => destination := destination*256 + ligne (i_point+1 à i-1)
4 => masque := masque + ligne (i_point+1 à i-1)
5, 6, 7 => masque := masque*256 + ligne (i_point+1 à i-1)
Fin Selon
nb_points_parcourus := nb_points_parcourus + 1;
i_point := i;

R4 : Comment "Interpréter l'interface d'une ligne de la table de routage" ?

i_espace_eth = Taille (ligne)
Tant Que ligne (i_espace_eth) /= ' ' **faire**
i_espace_eth = i_espace_eth - 1;
Fin Tant Que
Eth := ligne (i_espace_eth à (Taille (ligne)));

R5 : Comment "Trouver la route assurant la cohérence du cache" ?

parcours_table = Table;
meilleure_longueur = 0;

Pour i de 1 à Taille (Table) **faire**
masque_temp = (paquet ET parcours_table.Destination^) OU ((NON paquet) ET (NON parcours_table.destination^);
longueur = Calculer la longueur du masque (masque_temp)
masque_temp = Creer un masque (longueur + 1);

Déterminer si cette route a une plus grande longueur de masque que les précédentes **in:** longueur, paquet, masque_temp, eth;
in out: meilleure_longueur;
out: route_meilleure_cohérence

parcours_table = parcours_table.Suivant^;
Fin Pour

R6 : Comment "Déterminer si cette route a une plus grande longueur de masque que les précédentes" ?

Si longueur >= meilleure_longueur **faire**
meilleure_longueur = longueur
route_meilleure_cohérence = (paquet ET masque_temp, masque_temp, eth);
Sinon
Rien;
Fin Si

RAFFINAGE LCA

R3 : Comment "Trouver la route correspondant au paquet" ?

Convertir String en T_Adresse_IP **in:** ligne; **out:** paquet

Choisir la meilleure route pour ce paquet — **in:** Cache, Table, paquet;
out: meilleure_route : T_Donnee, route_enregistrée_dans_cache : Bouléen;
in out: Nb_defauts_cache

Nb_demandes_route = Nb_demandes_route + 1;
Taux_defauts_cache = Flottant (Nb_defauts_cache) / Flottant (Nb_demandes_route);

Écrire dans le fichier résultats — **in:** meilleure_route, Arguments

Mettre à jour le cache — **in:** Table, i_ligne_paquet, meilleure_route, Arguments, route_enregistrée_dans_cache, paquet;
in out: Cache

R4 : Comment "Choisir la meilleure route pour ce paquet" ?

route_enregistrée_dans_cache = Vrai;
Trouver la meilleure route pour ce paquet dans une table — **in:** Cache, ligne; **out:** meilleure_route
exception
    Aucune_Route_Valide => meilleure_route = Trouver la meilleure route pour ce paquet dans une table — **in:** Table, ligne; **out:** meilleure_route
        route_enregistrée_dans_cache = Faux;
        Nb_defauts_cache = Nb_defauts_cache + 1;

R4 : Comment "Écrire dans le fichier résultats" ?

Pour i de 1 à 3 **faire**
    longueur = Taille (Unbounded_String (Obtenir le ième octet (destination, i)))
    ligne = ligne & (Unbounded_String (Obtenir le ième octet (destination, i))) (2 à longueur) & "."
Fin Pour
longueur = Taille (Unbounded_String (Obtenir le ième octet (destination, 4)))
ligne = ligne & (Unbounded_String (Obtenir le ième octet (destination, 4))) (2 à longueur) & " " & eth

Écrire (ligne, Arguments.resultats);
Écrire_Nouvelle_Ligne (Arguments.resultats);

R4 : Comment "Mettre à jour le cache" ?

Si Taille (Cache) = Arguments.Taille **alors**
    Trier le cache — **in out:** Cache
Sinon
    Rien;
Fin Si

Enregistrer éventuellement la route dans le cache — **in:** Table, ligne, route_enregistrée_dans_le_cache, meilleure_route;
in out: Cache; **out:** route_meilleure_cohérence

Actualiser les statistiques d'utilisation de la route — **in:** route_meilleure_cohérence, i_ligne_paquets; **in out:** Cache

R5 : Comment "Trouver la meilleure route pour ce paquet dans une table" ?

```
parcours_table = Table;
meilleur_nombre_de_bits_a_1 = 0;
a_trouvé_une_route  = Faux;

Pour i de 1 à Taille (Table) faire
        Traiter la correspondance entre le paquet et la destination de la table
                in: paquet, parcours_table, nb_de_bits_du_masque_a_1;
                out: a_trouvé_une_route, nb_de_bits_du_masque_a_1,
                                                meilleure_route;
                in out: meilleur_nombre_de_bits_a_1
        parcours_table = parcours_table.Suivant^;
Fin Pour

Lever une erreur si aucune route ne correspond
        in: a_trouvé_une_route
```

R5 : Comment “Trier le cache” ?

```
Selon Arguments.politique dans
        FIFO => Trier le cache en FIFO      in out: Cache
        LRU => Trier le cache en LRU        in out: Cache
        LFU => Trier le cache en LFU        in out: Cache
Fin Selon
```

R5 : Comment “Actualiser les statistiques d’utilisation de la route” ?

```
Si Cache.Destination^ = route_meilleure_coherence.destination^ ET NON Est_Vide_LCA (Cache) alors
        Cache.Dernière_Utilisation^ = i_ligne_paquets;
        Cache.Nb_Utilisations^ = Cache.Nb_Utilisations^ + 1;
Sinon Si NON Est_Vide_LCA (Cache)
        Actualiser les statistiques d’utilisation de la route     in: route_meilleure_coherence, i_ligne_paquets; out: Cache.Suivant^
Sinon
        Rien;
Fin Si
```

R5: Comment “Enregistrer éventuellement la route dans le cache” ?

```
Si NON route_enregistrée_dans_le_cache alors
        eth = meilleure_route.eth^;
        Trouver la route assurant la cohérence du cache     in: Table, ligne, eth; out: route_meilleure_cohérence
        Enregistrer_LCA (Cache, route_meilleure_coherence.destination^, route_meilleure_coherence.masque^, eth);
Sinon
        Rien;
Fin Si
```

R6 : Comment “Traiter la correspondance entre le paquet et la destination de la table” ?

```
nb_de_bits_du_masque_a_1 = 0;
Si paquet ET parcours_table.Masque^ = parcours_table.Destination^ alors
        Compter les 1 dans un nombre binaire (parcours_table.Masque^)          out: nb_de_bits_du_masque_a_1
```

```
        a_trouvé_une_route = Vrai;

        Déterminer si cette route est meilleure que les précédentes      in: nb_de_bits_du_masque_a_1, parcours_table
                                                                         in out: meilleur_nombre_de_bits_a_1; out: meilleure_route;
Sinon
        Rien;
Fin Si
```

R6 : Comment “Lever une erreur si aucune route ne correspond” ?

```
Si NON a_trouvé_une_route alors
        lever Aucune_Route_Valide;
Sinon
        Rien;
Fin Si
```

R6 : Comment “Trier le cache en FIFO” ?

```
première_case = Cache;
Cache = Cache.Suivant^;
Libérer (première_case);
```

R6 : Comment “Trier le cache en LRU” ?

```
courant = Cache;
destination_LRU = courant.Destination^;
numero_LRU = courant.Dernière_Utilisation^;
Tant Que NON Est_Vide_LCA (courant) faire
        Déterminer si cette route est plus ancienne que les précédentes    in: courant; in out: numero_LRU, destination_LRU;
        courant = courant.Suivant^;
Fin Tant Que

Supprimer_LRU (Cache, destination_LRU);
```

R6 : Comment “Trier le cache en LFU” ?

```
courant = Cache;
destination_LFU = courant.Destination^;
frequence_LFU = courant.Nb_Utilisations^;
Tant Que NON Est_Vide_LCA (courant) faire
        Déterminer si cette route est moins fréquemment utile que les précédentes      in: courant; in out: fréquence_LFU, destination_LFU;
        courant = courant.Suivant^;
Fin Tant Que

Supprimer_LCA (Cache, destination_LFU);
```

R7 : Comment “Déterminer si cette route est meilleure que les précédentes” ?

```
Si nb_de_bits_du_masque_a_1 >= meilleur_nombre_de_bits_a_1 alors
```

```
        meilleur_nombre_de_bits_a_1 = nb_de_bits_du_masque_a_1;
        meilleure_route = (parcours_table.Destination^, parcours_table.Masque^, parcours_table.Eth^);
Sinon
        Rien;
Fin Si
```

R7 : Comment “Déterminer si cette route est plus ancienne que les précédentes” ?

```
Si courant.Dernière_Utilisation^ > numero_LRU alors
        numero_LRU = courant.Dernière_Utilisation^;
        destination_LRU = courant.Destination^;
Sinon
        Rien;
Fin Si
```

R7 : Comment “Déterminer si cette route est moins fréquemment utile que les précédentes” ?

```
Si courant.Nb_Utilisations^ < frequence_LFU alors
        frequence_LFU = courant.Nb_Utilisations^;
        destination_LFU = courant.Destination^;
Sinon
        Rien;
Fin Si
```

RAFFINAGE ARBRE

R3 : Comment “trouver la route correspondant aux paquets ?”

```
Nb_Demande_Route = Nb_Demande_Route + 1
Meilleure_route_precoherence = choisir la meilleure route (table, Cache_Arbre, convertir_adresse_entier(Ligne_Paquets), Nb_Defaut_Cache)
Taux_Defaut_Cache = Nb_Defaut_Cache / Nb_Demande_Route
Meilleure_route = Coherence(table, convertir_adresse_entier(Ligne_Paquets), Meilleure_route.eth) in : table  in : Meilleure_route
                                                                                                  in : Ligne_Paquet
                                                                                                       in : Meilleure_route
Si Dans_Cache(Liste_Cache, Meilleure_route) alors
        Mise_a_jour_LRU(Liste_Cache, Meilleure_route)   in out : Liste_Cache  in : Meilleure_route

SinonSi Taille(Liste_Cache) < Options.Taille alors
        Enregistrer_LCA(Liste_Cache, Meilleure_route.Destination, Meilleure_route.Masque, Meilleure_route.eth)
        enregistrer la route dans un arbre(Cache_Arbre, Meilleure_route.Destination, Meilleure_route.Masque, Meilleure_route.eth)
                                                                                                  in out : Cache_Arbre
                                                                                                  in : Meilleure_route
Sinon
```

Route_A_supprimer = Trier LRU l'arbre (Liste_Cache) **in out** : Liste_Cache
Enregistrer(Liste_Cache, Meilleure_route.Destination, Meilleure_route.Masque, Meilleure_route.eth)
Supprimer_Arbre(Cache_Arbre, Route_A_supprimer) **in out** : Cache_Arbre **in** : Route_A_supprimer
enregistrer la route dans un arbre(Cache_Arbre, Meilleure_route.Destination, Meilleure_route.Masque, Meilleure_route.eth)
in out : Cache_Arbre
in : Meilleure_route

Fin si
ligne = creer_ligne(convertir_adresse_entier(Ligne_Paquets), Meilleure_route.eth)
Ecrire dans le fichier (Sortie, ligne)
Nouvelle ligne (Sortie)

R4 : Comment "enregistrer la route dans un arbre ?"

indice_de_parcours = 1
enregistrer_arbre(Arbre, adresse, masque, eth, indice_de_parcours) **in out** : Arbre **in** : adresse **in** : masque **in** : eth **in out** : indice_de_parcours

R4 : Comment "choisir la meilleure route ?"

Fin_Boucle = Faux
indice = 1
parcours_Cache = Cache
paquet_bit = binaire(paquet)
presence = Faux
Tant Que non Fin_Boucle **faire**
rechercher dans le cache(parcours_cache, presence, Fin_boucle, route, paquet_bit, indice) **in out** : parcours_cache **in out** : presence
in out : Fin_boucle **out** : route **in** : paquet_bit **in out** : indice
Fin Tant Que
Si presence **alors**
retourner route
Sinon
retourner trouver la meilleure route pour ce paquet dans une table (Table_routage, Paquet) **in** : Table_routage **in** : Paquet
Fin si

R4 : Comment "supprimer une destination dans un arbre ?"

indice_pere = 1
supprimer l'élément du cache(Cache, Adresse, indice_pere) **in out** : Cache **in** : Adresse **in out** : indice_pere
supprimer les noeuds vides au dessus de l'élément supprimé(Cache, indice_pere, Adresse) **in out** : Cache **in** : Adresse **in out** : indice_pere

R5 : Comment "enregistrer une route dans un arbre ?"

Si Arbre = null **alors**
Créer une nouvelle cellule arbre.destination^ (adresse, masque, eth, null, null)
Sinon Si Arbre droite = null et Arbre gauche = null **alors**
enregistrer lorsque l'arbre ne possède ni fils droit ni fils gauche (Arbre, destination_bit, indice_de_parcours, masque, eth)
in out : Arbre **in** : destination_bit **in out** : indice_de_parcours **in** : masque **in** : eth
Sinon Si Arbre droite /= null et Arbre gauche /= null **alors**
enregistrer lorsque l'arbre possède un fils droit et un fils gauche (Arbre, adresse, destination_bit, indice_de_parcours, masque, eth)

in out : Arbre **in** : destination_bit **in out** : indice_de_parcours **in** : masque **in** : eth
Sinon Si Arbre droite /= null et Arbre gauche = null **alors**
enregistrer lorsque l'arbre possède un fils droit mais pas un fils gauche (Arbre, adresse, destination_bit, indice_de_parcours, masque, eth)
in out : Arbre **in** : destination_bit **in out** : indice_de_parcours **in** : masque **in** : eth
Sinon Si Arbre droite = null et Arbre gauche /= null **alors**
enregistrer lorsque l'arbre possède un fils gauche mais pas un fils droit (Arbre, adresse, destination_bit, indice_de_parcours, masque, eth)
in out : Arbre **in** : destination_bit **in out** : indice_de_parcours **in** : masque **in** : eth
Sinon
Rien;
Fin Si

R5 : Comment "rechercher dans le cache ?"

Si parcours_Cache /= null **alors**
continuer à chercher si noeud vide et s'arrêter sinon(parcours_cache, presence, Fin_Boucle, paquet_bit, indice)
in out : parcours_cache **in out** : presence **in out** : Fin_Boucle **in** : paquet_bit **in out** : indice
Sinon
Fin_boucle = Vrai;
Fin si

R5 : Comment "supprimer l'élément du cache ?"

Si Cache = null **alors**
Rien;
Sinon
supprimer si l'élément peut etre dans le cache(Cache, adresse, indice_pere) **in out** : Cache **in** : adresse **in out** : indice_pere
Fin Si

R5 : Comment "supprimer les noeuds vides au dessus de l'élément supprimé ?"

Pour indice_suppression reverse de 1 à indice_pere **faire**
indice_progression = 1
descendre l'arbre et supprimer le dernier Noeud Vide(Cache, indice_pere, indice_progression, Adresse)
in out : Cache **in out** : indice_pere **in out** : indice_progression **in** : Adresse
Fin Pour

R6 Comment "enregistrer lorsque l'arbre ne possède ni fils droit ni fils gauche ?"

Si Arbre.destination^= String_en_adresse_IP(destination_bit) **alors**
rien
Sinon
stockage = (Arbre.destination^, Arbre.masque^, Arbre.eth^)
Arbre.eth = "Noeud_vide"
destination_arbre = binaire(Arbre.destination^)
créer un noeud au niveau du bit de différence(Arbre, destination_bit, destination_arbre, indice_de_parcours, stockage, masque, eth)
in out : Arbre **in** : destination_bit **in** : destination_arbre **in out** : indice_de_parcours **in** : stockage **in** : masque **in** : eth

Fin Si

R6 Comment “enregistrer lorsque l’arbre possède un fils droit et un fils gauche ?”

Si destination_bit(indice_de_parcours) = ‘0’ **alors**
indice_de_parcours = indice_de_parcours + 1
enregistrer une route dans un arbre(Arbre gauche, adresse, masque, eth, indice_de_parcours)
in out : Arbre gauche **in** : adresse **in** : masque **in** : eth **in out** :indice_de_parcours
Sinon
indice_de_parcours = indice_de_parcours + 1
enregistrer une route dans un arbre(Arbre droite, adresse, masque, eth, indice_de_parcours)
in out : Arbre droite **in** : adresse **in** : masque **in** : eth **in out** :indice_de_parcours
Fin Si

R6 Comment “enregistrer lorsque l’arbre possède un fils droit mais pas un fils gauche ?”

Si Arbre gauche.eth^ /= "Noeud_Vide" and Arbre gauche.destination^ = String_En_Adresse_IP(Destination_bit) **alors**
Rien
Sinon Si Destination_bit(indice_de_parcours) = '0' and Arbre gauche.eth^ /= "Noeud_Vide" **alors**
Stockage = (Arbre gauche.destination^, Arbre gauchel.Masque^, Arbre .gauche.eth^)
destination_arbre = binaire(Arbre gauche.Destination^)
Arbre gauche.eth = “Noeud_vide”
créer un noeud au niveau du bit de différence(Arbre gauche, destination_bit, destination_arbre, indice_de_parcours, stockage, masque, eth);
in out : Arbre gauche **in** : destination_bit **in** : destination_arbre **in out** : indice_de_parcours **in** : stockage **in** : masque **in** : eth
Sinon Si Destination_bit(indice_de_parcours) = '0' and Arbre gauchel.eth^ = "Noeud_Vide" **alors**
Indice_de_parcours = indice_de_parcours + 1
enregistrer une route dans un arbre (Arbre gauche, adresse, masque, Eth, indice_de_parcours);
in out : Arbre gauche **in** : adresse **in** : masque **in** : eth **in out** :indice_de_parcours
Sinon
Arbre droite = Nouvelle cellule (String_En_Adresse_IP (destination_bit), Masque, Eth, null, null)
Fin Si

R6 Comment “enregistrer lorsque l’arbre possède un fils gauche mais pas un fils droit ?”

Si Arbre droite.eth^ /= "Noeud_Vide" and Arbre droite.Destination^ = String_En_Adresse_IP(Destination_bit) **alors**
Rien
Sinon Si To_String(Destination_bit)(indice_de_parcours) = '1' and Arbre.all.droite.eth /= "Noeud_Vide" **alors**
Stockage = (Arbre droite.Destination^, Arbre droite.Masque^, Arbre droite.eth^)
destination_arbre = binaire(Arbre droite.Destination^)
Arbre droite.eth = “Noeud_vide”
créer un noeud au niveau du bit de différence(Arbre droite, destination_bit, destination_arbre, indice_de_parcours, stockage, masque , eth)
in out : Arbre droite **in** : destination_bit **in** : destination_arbre **in out** : indice_de_parcours **in** : stockage **in** : masque **in** : eth
Sinon Si To_String(Destination_bit)(indice_de_parcours) = '1' and Arbre.all.droite.eth = "Noeud_Vide" **alors**
Indice_de_parcours = indice_de_parcours + 1
enregistrer une route dans un arbre (Arbre droite, adresse, masque, Eth, indice_de_parcours)
in out : Arbre droite **in** : adresse **in** : masque **in** : eth **in out** :indice_de_parcours

Sinon
Arbre gauche := Nouvelle cellule (String_En_Adresse_IP (destination_bit), Masque, Eth, null, null);
Fin Si

R6 :Comment “continuer à chercher si noeud vide et s'arrêter sinon ?

Si parcours_Cache.all.eth /= To_Unbounded_String("Noeud_vide") **alors**
rechercher la route si une destination a été trouvé (parcours_cache, presence, Fin_Boucle, route, paquet_bit, indice)
in out : parcours_cache **in out** : presence **in out** : Fin_Boucle **in out** : route **in** : paquet_bit **in out** : indice
Sinon
rechercher la route si on a trouvé Noeud Vide (parcours_cache, presence, Fin_Boucle, route, paquet_bit, indice)
in out : parcours_cache **in out** : presence **in out** : Fin_Boucle **in out** : route **in** : paquet_bit **in out** : indice
Fin Si

R6 : Comment “supprimer si l’élément peut etre dans le cache ?”

paquet_bit := binaire(Adresse);
Si Cache.all.eth = To_Unbounded_String("Noeud_Vide") **alors**
rechercher si on trouve noeud vide lors de la suppression(Cache, adresse, indice_pere, paquet_bit)
in out : Cache **in** : adresse **in out** : indice_pere **in** : paquet_bit
Sinon Si Cache.all.Destination = Adresse **alors**
Libérer(Cache)
Sinon
Rien
Fin Si

R6 Comment “descendre l’arbre et supprimer le dernier Noeud Vide ?”

Si indice_progression < indice_pere **alors**
descendre dans l’arbre(Cache, indice_pere, indice_progression, Adresse)
in out : Cache **in out** : indice_pere **in out** : indice_progression **in** : Adresse
Sinon Si binaire(Adresse)(indice_progression + 1) = '0' **et** Cache droite = null **alors**
Libérer(Cache.all.gauche)
Sinon Si binaire(Adresse)(indice_progression + 1) = '1' **et** Cache gauche = null **alors**
Libérer(Cache.all.droite)
Fin si

R7 : Comment “créer un noeud au niveau du bit de différence ?”

Si destination_bit(indice_de_parcours) = destination_arbre(indice_de_parcours) **et** indice de parcours < 32 **alors**
créer des noeuds vides (Arbre, indice_de_parcours, destination_bit)
in out : Arbre **in out :** indice_de_parcours **in** : destination_bit
poursuivre la création de noeuds vides(Arbre, destination_bit, destination_arbre, indice_de_parcours, stockage, masque, eth)
in out : Arbre **in** : destination_bit **in** : destination_arbre **in out** : indice_de_parcours **in** : stockage **in** : masque **in** : eth
Sinon
créer deux branches gauche et droite après le nœud de différence(Arbre, indice_de_parcours, stockage, destination_bit, masque, eth)
in out : Arbre in out : indice_de_parcours in : stockage in : destination_bit in : masque in : eth
Fin Si

R7 : Comment “rechercher la route si une destination a été trouvé ?”

Si (Paquet and Parcours_Cache.all.Masque) = Parcours_Cache.all.Destination **alors**
    presence = Vrai
    Fin_Boucle = Vrai
    route := (parcours_Cache.Destination^, parcours_Cache.Masque^, parcours_Cache.eth^)
Sinon
    Fin_Boucle = Vrai
Fin Si

R7 : Comment “rechercher la route si on a trouvé Noeud Vide ?”

Si To_string(paquet_bit)(indice) = '0' **Sinon**
    Parcours_Cache = Parcours_Cache.all.gauche
    indice = indice + 1
Sinon
    Parcours_Cache := Parcours_Cache.all.droite
    indice = indice + 1;
Fin Si

R7 : Comment “rechercher si on trouve noeud vide lors de la suppression ?”

Si paquet_bit(indice_pere) = '0' **alors**
    supprimer l’élément du cache(Cache gauche, adresse, indice_pere + 1)
in out : Cache gauche **in** : adresse **in** : indice_pere
Sinon
    supprimer l’élément du cache(Cache droite, adresse, indice_pere + 1)
in out : Cache gauche **in** : adresse **in** : indice_pere
Fin Si

R7 : Comment “descendre dans l’arbre ?”

Si To_String(binaire(Adresse))(indice_progression) = '0' **Alors**
    indice_progression := indice_progression + 1
    descendre l’arbre et supprimer le dernier Noeud Vide(Cache gauche, indice_pere, indice_progression, Adresse)
in out : Cache gauche **in out** : indice_pere **in out** : indice_progression **in** : Adresse

Sinon
    indice_progression := indice_progression + 1
    descendre l’arbre et supprimer le dernier Noeud Vide(Cache droite, indice_pere , indice_progression ,Adresse)
in out : Cache droite **in out** : indice_pere **in out** : indice_progression **in** : Adresse
Fin Si

R8 : Comment “créer des noeuds vides”

Si Destination_bit(indice_de_parcours) = 0
    Arbre gauche.interface = “noeud vide”
Sinon
    Arbre droite.interface = “noeud vide”

Fin Si

R8 : Comment “poursuivre la création de noeuds vides ?”

Si destination_bit(indice_de_parcours) = ‘0’ **alors**
indice_de_parcours = indice_de_parcours + 1
créer un noeud au niveau du bit de différence(Arbre droite, destination_bit, destination_arbre, indice_de_parcours, stockage, masque, eth)
in out : Arbre droite **in** : destination_bit **in** : destination_arbre **in out** : indice_de_parcours **in** : stockage **in** : masque **in** : eth
Sinon
indice_de_parcours = indice_de_parcours + 1
créer un noeud au niveau du bit de différence(Arbre gauche, destination_bit, destination_arbre, indice_de_parcours, stockage, masque, eth)
in out : Arbre gauche **in** : destination_bit **in** : destination_arbre **in out** : indice_de_parcours **in** : stockage **in** : masque **in** : eth
Fin Si

R8 : Comment “créer deux branches gauche et droite après le nœud de différence ?”

Si binaire(stockage.destination)(indice de parcours) = 0 **alors**
L’interface de Arbre est un noeud vide
Créer une nouvelle cellule arbre_gauche.destination^ (Stockage)
Créer une nouvelle cellule arbre_droite.destination^ (String_en_adresse_IP(destination_bit), Masque, Interface)
Sinon
L’interface de Arbre est un noeud vide
Créer une nouvelle cellule arbre_droite.destination^ (Stockage)
Créer une nouvelle cellule arbre_gauche.destination^ (String_en_adresse_IP(destination_bit), Masque, Interface)
Fin Si