RAPPORT PROJET 2022 -2023

EXPLOITATION DES TABLES DE ROUTAGE

Le présent document a pour finalité de dévoiler les étapes de réflexion (à savoir les réflexions conduites sur l'architecture et les types de données appropriés, les difficultés rencontrées, les solutions élaborées, les concepts rejetés) rencontrées au cours de l'exploitation d'une table de routage en langage Ada, ainsi que les perspectives envisagées pour optimiser ou poursuivre davantage le projet.

Auteurs du projet

BOUCHAMA Ayoub
ELGUERRAOUI Oussama
MARZOUGUI Achraf

Encadrant

MENDIL Ismail

Groupes du projet

E / F

Promotion

2022/2023

Deadline du projet

22/11/2022 - 14/01/2023

Objectif du projet

L'objectif de ce projet est d'implanter, évaluer et comparer plusieurs manières de stocker et exploiter la table de routage d'un routeur.

Résultats attendus :

On s'attend à ce que les différentes méthodes aient été mises en œuvre et testées et que les résultats de ces tests aient permis de déterminer la méthode qui offre les meilleures performances en termes de temps de réponse de consommation de ressources et de fiabilité. On s' attend également à ce que ces résultats soient présentés et comparés de manière claire et concise, afin de permettre aux décideurs de prendre une décision informée sur la meilleure approche à adopter pour stocker et exploiter la table de routage de leur routeur.

INTRODUCTION

Le projet que nous avons entrepris vise à développer et à évaluer des modèles de gestion de la table de routage d'un routeur en utilisant le langage de programmation Ada et en mettant en œuvre une solution de stockage basée sur les listes chaînées. Notre objectif est de fournir une analyse comparative des performances de ces différentes approches afin de déterminer laquelle offre les meilleures caractéristiques en termes de temps de réponse, de consommation de ressources et de fiabilité.

La gestion de la table de routage est un élément clé de la performance d'un routeur, car elle contient les informations nécessaires pour acheminer les paquets de données vers leur destination finale. Il est donc crucial de disposer d'une méthode efficace pour stocker et gérer les entrées de cette table de manière à minimiser les temps de latence et les erreurs de transmission.

Parmi les différentes approches de stockage qui peuvent être utilisées, les listes chaînées sont une option couramment adoptée en raison de leur flexibilité et de leur efficacité en termes de temps et de mémoire. Cependant, il convient de noter que l'utilisation de listes chaînées peut entraîner une augmentation de la consommation de ressources et une diminution des performances si elles ne sont pas utilisées de manière appropriée.

Dans le cadre de ce projet, nous avons choisi de mettre en œuvre une solution de gestion de la table de routage basée sur les listes chaînées en utilisant le langage Ada. Nous avons également décidé de comparer les performances de cette solution avec celles de différentes stratégies de gestion de cache, telles que LRU, LFU et FIFO. Ainsi, nous espérons être en mesure de déterminer laquelle de ces approches offre les meilleures caractéristiques en termes de temps de réponse, de consommation de ressources et de fiabilité.

Le principal objectif de notre projet est de développer et d'évaluer des modèles de gestion de la table de routage d'un routeur en utilisant le langage Ada et en mettant en œuvre une solution de stockage basée sur les listes chaînées. Nous souhaitons également comparer les performances de cette solution avec celles de différentes stratégies de gestion de cache afin de déterminer laquelle offre les meilleures caractéristiques en termes de temps de réponse, de consommation de ressources et de fiabilité.

Pour atteindre ces objectifs, nous avons prévu de suivre les étapes suivantes :

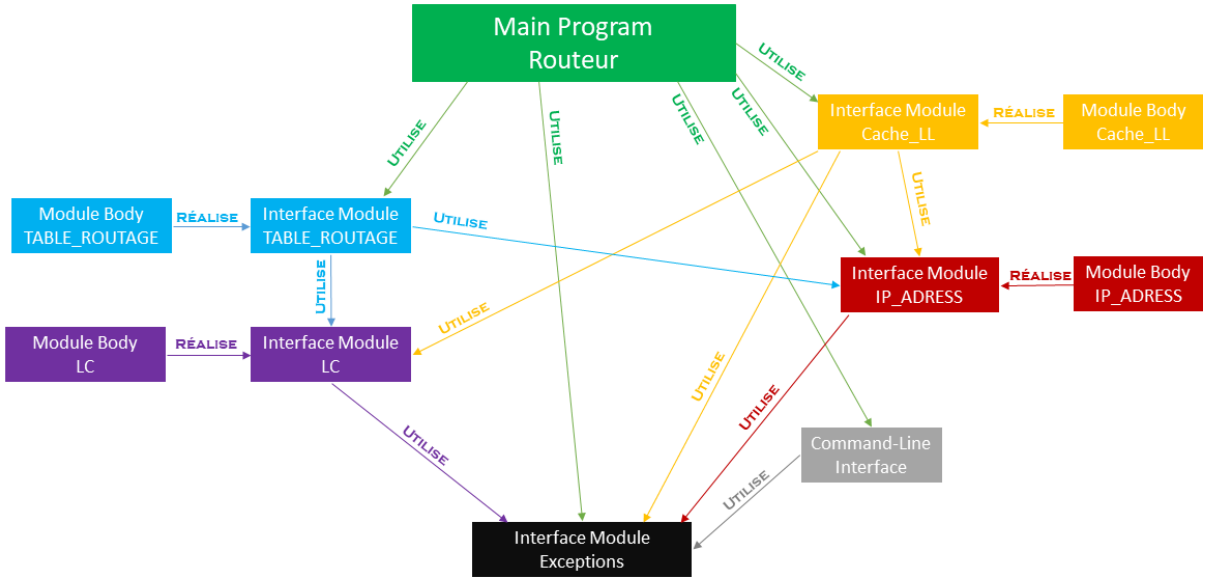
- Développement des modèles de gestion de la table de routage en utilisant le langage Ada et en mettant en œuvre une solution de stockage basée sur les listes chaînées.
- Mise en place d'un environnement de test et de simulation pour évaluer les performances des modèles développés.
- Génération de scénarios de test représentatifs des différents types de trafic auxquels peut être confronté un routeur dans des conditions réelles.
- Tester et évaluer les performances de ces modèles en utilisant différentes stratégies de gestion de cache (LRU, LFU, FIFO).
- Exécution des tests et collecte des données de performance.
- Analyse des résultats des tests et comparaison des performances des différentes approches testées.
- Rédaction d'un rapport final présentant les résultats de l'étude et les recommandations pour l'utilisation de la meilleure approche de gestion de la table de routage.

PLAN

I. Présentation de la conception architecturale et de la sélection des types de données

Architecture du système :

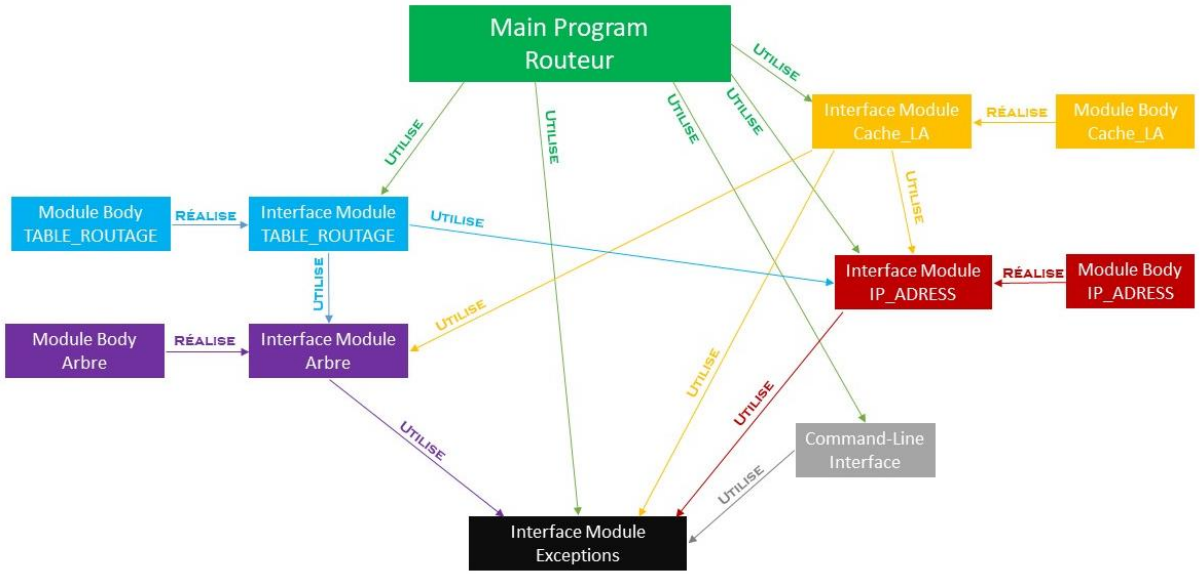
I. Système utilisant un cache avec la structure de donnée : liste chaînée



Le système utilise un cache avec une structure de données de type liste chaînée pour stocker temporairement des données afin de les rendre plus rapidement accessibles. Cette technique est souvent utilisée pour accélérer l'accès à des données fréquemment utilisées ou à des données qui sont coûteuses à recréer ou à récupérer. La liste chaînée est utilisée pour stocker les données dans l'ordre où elles ont été ajoutées au cache, ce qui permet de gérer facilement l'ajout et la suppression de données du cache.

Le programme principal routeur utilise plusieurs modules afin de pouvoir fonctionner correctement. Le module **Cache_LL** permet de gérer le cache du routeur, c'est-à-dire la mémoire tampon qui stocke temporairement les routes déjà utilisées afin de pouvoir y accéder rapidement par la suite et qui utilise le module **LC** qui permet d'effectuer les différentes opérations sur les listes chainees. Le module **TABLE_ROUTAGE** permet de gérer la table de routage du routeur, qui contient l'ensemble des routes connues par le routeur et qui lui permet de déterminer vers quelle interface envoyer les paquets. Le module **IP_ADRESS** gère les adresses IP, c'est-à-dire les identifiants numériques qui permettent d'identifier les différents appareils connectés au réseau. Le module **CLI** (Command Line Interface) permet de gérer les arguments de la ligne de commande, c'est-à-dire les options et les paramètres qui peuvent être passés au programme lors de son exécution. Enfin, le module **Exceptions** gère les exceptions, c'est-à-dire les erreurs qui peuvent survenir lors de l'exécution du programme et qui nécessitent une gestion spécifique. Tous ces modules sont utilisés par le programme principal routeur afin de pouvoir exécuter ses différentes fonctionnalités de manière optimale et de gérer les éventuels problèmes qui peuvent survenir.

II. Système utilisant un cache avec la structure de donnée : Arbre



Un système qui utilise un cache avec un arbre est un système informatique qui utilise une structure de données en arbre pour stocker temporairement des données dans un cache afin de les rendre plus facilement accessibles. Cette technique est souvent utilisée pour améliorer les performances de l'application en réduisant les temps d'accès aux données et en réduisant la charge sur le système de stockage principal.

Le programme principal routeur utilise plusieurs modules afin de pouvoir fonctionner correctement. Le module **Cache_LA** permet de gérer le cache du routeur, c'est-à-dire la mémoire tampon qui stocke temporairement les routes déjà utilisées afin de pouvoir y accéder rapidement par la suite et qui utilise le module **Arbre** qui permet d'effectuer les différentes opérations sur un arbre. Le module **TABLE_ROUTAGE** permet de gérer la table de routage du routeur, qui contient l'ensemble des routes connues par le routeur et qui lui permet de déterminer vers quelle interface envoyer les paquets. Le module **IP_ADRESS** gère les adresses IP, c'est-à-dire les identifiants numériques qui permettent d'identifier les différents appareils connectés au réseau. Le module **CLI** (Command Line Interface) permet de gérer les arguments de la ligne de commande, c'est-à-dire les options et les paramètres qui peuvent être passés au programme lors de son exécution. Enfin, le module **Exceptions** gère les exceptions, c'est-à-dire les erreurs qui peuvent survenir lors de l'exécution du programme et qui nécessitent une gestion spécifique. Tous ces modules sont utilisés par le programme principal routeur afin de pouvoir exécuter ses différentes fonctionnalités de manière optimale et de gérer les éventuels problèmes qui peuvent survenir.

Modules crées :

- **TABLE_ROUTAGE :** permet de définir les opérations qui vont être utilisées dans la manipulation de la table de routage.
- **LC :** permet de définir une table de routage sous forme d'une liste chainée.
- **IP_ADRESS :** permet de définir le type Adresse IP et les opérations utilisées pour manipuler une Adresse IP.
- **Cache_LL :** permet de construire le cache version liste chainée.
- **CLI** (Command Line Interface) **:** permet d'afficher le guide d'utilisation de l'application
- **Arbre :** permet de définir le type arbre et les opérations qui vont être utilisées pour le manipuler.
- **Cache_LA :** permet de construire le cache version arbre.
- **Exceptions :** permet de gérer les exceptions.

Packages utilisés :

- Ada.Text_IO
- Ada.Integer_Text_IO ;
- Ada.Strings.Unbounded ;
- Ada.Strings;
- Ada.Exceptions ;
- Ada.Command_Line ;

Les types de données :

- **Dans Cache_LL :**

```
private

        type T_Cellule is record
            IP_Interface : Unbounded_String;
            N_CALLS_IP : Integer;
        end record;

        package ADRESS_CELLULE_LL is new LC (T_Cle => T_Adresse_IP, T_Donnee => T_Cellule);
        use ADRESS_CELLULE_LL;

        type T_Cache is record
            L : T_LC;
            Size_Max : Integer;
            N_CALLS : Integer;
            N_ERRORS : Integer;
        end record;

end Cache_LL;
```

- **Dans LC :**

```
76  private
77
78      type T_Cellule;
79
80      type T_LC is access T_Cellule;
81
82      type T_Cellule is
83          record
84              Cle : T_Cle;
85              Donnee : T_Donnee;
86              Suivant : T_LC;
87          end record;
88
89  end LC_ROUTAGE;
90
```

- **Dans TABLE_ROUTAGE :**

```
7
8   package TABLE_ROUTAGE is
9
10      Type T_Donnee is record
11        Masque : T_Adresse_IP;
12        Destination : Unbounded_String;
13      end record;
```

- **Dans Arbre :**

```
38
39  private
40
41   type T_Noeud;
42
43   type T_LA is access T_Noeud;
44
45  type T_Noeud is
46     record
47         Cle : T_Adresse_IP ;
48         Valeur: Unbounded_String;
49         Visite_Noeud : Integer;
50         Fils_Gauche : T_LA ;
51         Fils_Droit : T_LA ;
52     end record;
53
54  end Arbre;
```

➢ **Dans Cache_LA :**

```
private

   type T_Cache is record
      Arbre : T_LA; -- Arbre
      Size : Integer; -- Taille de l'arbre
      Size_Max : Integer; -- Taille max de l'arbre
      Visite_Noeud : Integer; -- Nombre de visite de noeud
      Errors : Integer; -- Nombre d'erreurs
   end record;

end Cache_LA;
```

➢ **Dans IP_ADRESS**

```
package IP_ADRESS is

  Type T_Adresse_IP is mod 2**32;
```

II. Présentation des principaux algorithmes et des tests des programmes

CACHE VERSION LISTE CHAINÉE

➔ Module IP_ADRESS

- **Nom :** Show_IP
  Entrée : IP : adresse IP à afficher (T_Adresse_IP)

  Sortie : Aucune

  Sémantique : Afficher l'adresse IP sous forme de chaîne de caractères "999.999.999.999"

- **Nom :** Add_IP
  Entrée :
  - File : fichier dans lequel ajouter l'adresse IP (File_Type)
  - IP : adresse IP à ajouter (T_Adresse_IP)

  Sortie : Aucune

  Sémantique : Ajouter l'adresse IP au fichier.

- **Nom :** Get_IP
  Entrée :
  - File : fichier à partir duquel récupérer l'adresse IP (File_Type)

  Sortie : l'adresse IP récupérée (T_Adresse_IP)

  Sémantique : Récupérer une adresse IP du fichier.

➔ Module Cache_LL

- **Nom :** Initialiser_Cache_LL
  Entrée :
  - Cache : cache à initialiser (T_Cache)
  - Size_Max : taille maximale du cache (Integer)

  Sortie : Aucune.

  Sémantique : Initialise le cache en mettant la taille maximale à Size_Max et en vidant la liste chaînée de cache.

- **Nom :** Afficher_Statistiques_LL
  Entrée :
  - Cache : cache dont on veut afficher les statistiques (T_Cache)

  Sortie : Aucune.

  Sémantique : Affiche le nombre total d'appels et le nombre de défauts du cache.

- **Nom :** Afficher_Cache_LL
  Entrée :
  - Cache : cache à afficher (T_Cache)
  - Politique : politique de remplacement utilisée (Unbounded_String)

  Sortie : Aucune.

  Sémantique : Affiche le contenu du cache, en indiquant la politique de remplacement utilisée. Pour chaque entrée, affiche l'adresse IP de la destination et l'adresse IP de l'interface de sortie associée, ainsi que le nombre d'appels de cette entrée.

- **Nom :** Lire_Cache_LL

  Entrée :

  - Cache : cache dans lequel chercher (T_Cache)
  - Destination : adresse IP de la destination à chercher (T_Adresse_IP)
  - Politique : politique de remplacement utilisée (Unbounded_String)

  Sortie :

  - IP_Interface : adresse IP de l'interface trouvée (Unbounded_String)
  - Found : booléen indiquant si la destination a été trouvée ou non (Boolean)

  Sémantique : Cherche l'adresse IP de l'interface correspondant à la destination dans le cache. Si la destination est trouvée, la fonction retourne l'adresse IP de l'interface dans IP_Interface et Found vaut True. Sinon, Found vaut False.

- **Nom :** Ajouter_Cache_LL

  Entrée :

  - Cache : cache dans lequel ajouter l'entrée (T_Cache)
  - Destination : adresse IP de la destination (T_Adresse_IP)
  - IP_Interface : adresse IP de l'interface de sortie associée (Unbounded_String)
  - Politique : politique de remplacement utilisée (Unbounded_String)

  Sortie : aucune

  Sémantique : Ajoute l'entrée (Destination, IP_Interface) au cache en respectant la politique de remplacement spécifiée. Si l'entrée existe déjà, son compteur d'appels est incrémenté. Si le cache est plein, l'entrée qui doit être remplacée selon la politique est supprimée.

- **Nom :** Incrementer_Defauts_LL

  Entrée :

  - Cache : cache dans lequel incrémenter le nombre de défauts (T_Cache)

  Sortie : Aucune

  Sémantique : Incrémente le nombre de défauts du cache de 1.

- **Nom :** Taille_Cache_LL

  Entrée :

  - Cache : cache dont on veut connaître la taille (T_Cache)

  Sortie :

  - Taille : taille du cache (Integer)

  Sémantique : Retourne la taille du cache (c'est-à-dire le nombre d'éléments qu'il contient).

- **Nom :** Est_Vide_Cache_LL

  Entrée :

- Cache : cache à tester (T_Cache)

Sortie :

- Est_Vide : booléen indiquant si le cache est vide ou non (Boolean)

Sémantique : Retourne True si le cache est vide, False sinon.

- **Nom :** Nb_Appels_Destination_LL
  Entrée :
    - Cache : cache dans lequel chercher (T_Cache)
    - Destination : adresse IP de la destination à chercher (T_Adresse_IP)

  Sortie : nombre d'appels associé à l'entrée (Destination, IP_Interface) dans le cache (Integer)

  Sémantique : Retourne le nombre d'appels associé à l'entrée (Destination, IP_Interface) dans le cache. Si l'entrée n'existe pas, retourne 0.

→ Module LC

- **Nom :** Initialiser
  Entrée :
    - TRT : table de routage à initialiser (T_LC)

  Sortie : Aucune.

  Sémantique : Initialise la table de routage TRT en la mettant vide.

- **Nom :** Est_Vide
  Entrée :
    - TRT : table de routage à tester (T_LC)

  Sortie :
    - Est_Vide : booléen indiquant si la table de routage est vide ou non (Boolean)

  Sémantique : Retourne True si la table de routage est vide, False sinon.

- **Nom :** Taille
  Entrée :
    - TRT : table de routage à tester (T_LC)

  Sortie :
    - Taille : nombre d'éléments dans la table de routage (Integer)

  Sémantique : Retourne le nombre d'éléments dans la table de routage TRT.

- **Nom :** Enregistrer

Entrée :

- TRT : table de routage à modifier (T_LC)
- Cle : clé de l'élément à enregistrer (T_Cle)
- Donnee : donnée de l'élément à enregistrer (T_Donnee)

Sortie : Aucune

Sémantique : Enregistre la donnée Donnee associée à la clé Cle dans la table de routage TRT. Si la clé est déjà présente dans la table de routage, sa donnée est changée.

- **Nom :** Supprimer
  Entrée :
  - TRT : table de routage à modifier (T_LC)
  - Cle : clé de l'élément à supprimer (T_Cle)

  Sortie : Aucune

  Sémantique : Supprime l'élément associé à la clé Cle dans la table de routage TRT. Génère une exception Cle_Absente_Exception si la clé n'est pas utilisée dans la table de routage.

- **Nom :** Cle_Presente
  Entrée :
  - TRT : table de routage à tester (T_LC)
  - Cle : clé à tester (T_Cle)

  Sortie :
  - Cle_Presente : booléen indiquant si la clé est présente dans la table de routage (Boolean)

  Sémantique : Retourne True si la clé Cle est présente dans la table de routage TRT, False sinon.

- **Nom :** La_Donnee
  Entrée :
  - TRT : table de routage à tester (T_LC)
  - Cle : clé de l'élément dont on veut la donnée (T_Cle)

  Sortie :
  - La_Donnee : donnée associée à la clé dans la table de routage (T_Donnee)

  Sémantique : Retourne la donnée associée à la clé Cle dans la table de routage TRT. Génère une exception Cle_Absente_Exception si la clé n'est pas utilisée dans la table de routage.

- **Nom :** Vider
  Entrée :
  - TRT : table de routage à vider (T_LC)

  Sortie : Aucune

  Sémantique : Supprime tous les éléments de la table de routage TRT.

- **Nom :** Enregistrer_Fin
  Entrée :
  - TRT : table de routage à modifier (T_LC)
  - Cle : clé de l'élément à enregistrer (T_Cle)
  - Donnee : donnée de l'élément à enregistrer (T_Donnee)

  Sortie : Aucune

  Sémantique : Enregistre l'élément avec la clé Cle et la donnée Donnee à la fin de la table de routage TRT.

- **Nom :** Suppression_Haute
  Entrée :
  - TRT : table de routage à modifier (T_LC)

  Sortie : Aucune

  Sémantique : Supprime l'élément le plus haut de la table de routage TRT.

- **Nom :** Element_En_Indice
  Entrée :
  - TRT : table de routage à tester (T_LC)
  - Indice : indice de l'élément à retrouver (Integer)
  - Cle : variable dans laquelle la clé de l'élément sera stockée (T_Cle)
  - Donnee : variable dans laquelle la donnée de l'élément sera stockée (T_Donnee)

  Sortie : Aucune

  Sémantique : Met dans les variables Cle et Donnee la clé et la donnée de l'élément à l'indice Indice dans la table de routage TRT.

- **Nom :** Pour_Chaque
  Entrée :
  - TRT : table de routage à parcourir (T_LC)

  Sortie :

Sémantique : Parcourt chaque élément de la table de routage TRT et exécute la procédure Traiter avec la clé et la donnée de chaque élément en entrée. Si une exception est générée lors de l'exécution de Traiter, affiche un message d'erreur.

→ Module TABLE_ROUTAGE

- **Nom :** Show_Line
  Entrée :
  - Cle : clé de l'élément à afficher (T_Adresse_IP)
  - Donnee : donnée de l'élément à afficher (T_Donnee)

  Sortie : Aucune

  Sémantique : Affiche une ligne de la table de routage avec la clé Cle et la donnée Donnee sous le format "cle: donnee".

- **Nom :** Show_Table
  Entrée :
  - Table : table de routage à afficher (T_LC)
  - Line : numéro de la ligne en cours d'affichage (Integer)

  Sortie : Aucune

  Sémantique : Affiche l'intégralité de la table de routage Table, en affichant chaque ligne avec la procédure Show_Line et en mettant à jour le numéro de la ligne en cours d'affichage Line.

- **Nom :** Compare_Table
  Entrée :
  - Table : table de routage à tester (T_LC)
  - IP : adresse IP à trouver dans la table de routage (T_Adresse_IP)

  Sortie :
  - Compare_Table : interface de sortie associée à l'adresse IP dans la table de routage (Unbounded_String)

  Sémantique : Retourne l'interface de sortie associée à l'adresse IP dans la table de routage Table. Si l'adresse IP n'est pas trouvée dans la table de routage, retourne une chaîne vide.

- **Nom :** Initialiser_Table
  Entrée :
  - Table : table de routage à initialiser (T_LC)
  - File_Table : fichier contenant les éléments à ajouter à la table de routage (File_Type)

  Sortie : Aucune

Sémantique : Initialise la table de routage Table en ajoutant les éléments contenus dans le fichier File_Table, qui doit être au format "adresse_IP masque interface_de_sortie". Chaque ligne du fichier correspond à un élément de la table de routage.

➔ Module CLI

- **Nom :** User_Guide
  Entrée : Aucune

  Sortie : Aucune

  Sémantique : Affiche le guide de l'utilisateur sur l'utilisation des options du programme.

- **Nom :** Argument_Parsing
  Entrée :
  - Argument_Count : le nombre d'arguments passés au programme (Integer)
  - Cache_Size : la taille du cache (Integer)
  - Policy : la politique de remplacement du cache (Unbounded_String)
  - Stat : indique si les statistiques doivent être affichées (Boolean)
  - Data_File_Name : le nom du fichier contenant les routes de la table de routage (Unbounded_String)
  - In_Package_File_Name : le nom du fichier contenant les paquets à router (Unbounded_String)
  - Out_Result_File_Name : le nom du fichier contenant les résultats (adresse IP destination du paquet et interface utilisée) (Unbounded_String)

  Sortie : Aucune

  Sémantique : Analyse les arguments passés au programme et met à jour les variables Cache_Size, Policy, Stat, Data_File_Name, In_Package_File_Name et Out_Result_File_Name en fonction des options spécifiées. Si une option invalide est spécifiée, affiche un message d'erreur et quitte le programme. Si aucune option n'est spécifiée, utilise les valeurs par défaut.

CACHE VERSION ARBRE PRÉFIXÉ

(En Plus des fonctions / Procédures définies précédemment on ajoute)

➔ Module Arbre

- **Nom :** Free
  Entrée : Aucune

  Sortie : Aucune

Sémantique : Constructeur de déallocation incontrôlée qui permet de libérer la mémoire allouée pour un noeud de l'arbre.

- **Nom :** Initialiser
  Entrée : Arbre (T_LA)

  Sortie : Aucune

  Sémantique : Procédure qui initialise l'arbre en le définissant comme nul.

- **Nom :** Taille
  Entrée : Arbre (T_LA)

  Sortie : Taille (Integer)

  Sémantique : Fonction qui retourne la taille de l'arbre en comptant le nombre de noeuds.

- **Nom :** Supprimer
  Entrée : Courant (Integer), Arbre (T_LA)

  Sortie : Aucune

  Sémantique : Procédure qui supprime un noeud de l'arbre en utilisant une clé donnée. Elle utilise également les sous-programmes Minimum_Recursion et Supprimer_Par_Cle.

- **Nom :** Traiter
  Entrée : Courant (Integer), Cle (T_Adresse_IP), Arbre (T_LA)

  Sortie : Valeur (Unbounded_String), Found (Boolean)

  Sémantique : Procédure qui cherche une adresse dans l'arbre et retourne la valeur associée ainsi que l'indicateur Found.

- **Nom :** Enregistrer
  Entrée : Courant (Integer), Cle (T_Adresse_IP), Valeur (Unbounded_String), Arbre (T_LA)

  Sortie : Aucune

  Sémantique : Procédure qui enregistre une clé et une valeur dans l'arbre. Elle commence par vérifier si la clé existe déjà dans l'arbre, et si c'est le cas, elle met à jour la valeur associée. Sinon, elle crée un nouveau noeud avec la clé et la valeur données et l'insère à la bonne place dans l'arbre en respectant l'ordre des clés.

- **Nom :** Afficher
  Entrée : Arbre (T_LA)

Sortie : Aucune

Sémantique : Procédure qui affiche l'arbre en parcourant ses noeuds de manière infixe.

➔ Module Cache_LA

- **Nom :** Initialiser_Cache_LA
  Entrée : Un enregistrement T_Cache et un entier Taille_Max
  Sortie :
  Sémantique : Initialise les champs de l'enregistrement T_Cache avec les valeurs fournies en entrée.

- **Nom :** Afficher_Statistiques_LA
  Entrée : Un enregistrement T_Cache
  Sortie :
  Sémantique : Affiche les statistiques de l'enregistrement T_Cache.

- **Nom :** Lire_Cache_LA
  Entrée :
  - Un enregistrement T_Cache
  - Une adresse IP de type T_Adresse_IP,
  - Une chaîne de caractères Unbounded_String
  - Un booléen

  Sortie : Aucune
  - Une chaîne de caractères Unbounded_String
  - Un booléen

  Sémantique : Cherche l'adresse IP dans l'arbre de l'enregistrement T_Cache. Si l'adresse est trouvée, met à jour le booléen en sortie et renvoie l'interface associée dans la chaîne de caractères. Sinon, met à jour le booléen en sortie et renvoie une chaîne vide.

- **Nom :** Afficher_Cache_LA
  Entrée : Un enregistrement T_Cache
  Sortie : Aucune
  Sémantique : Affiche les éléments de l'arbre de l'enregistrement T_Cache.

LES TESTS DE PROGRAMMES

Pour évaluer les performances des programmes du routeur, nous avons mis en place les protocoles de test suivants :

- Tester le cache en initialisant les versions liste chainée et arbre, puis en appliquant les politiques LRU, FIFO, LFU, en y ajoutant les entrées 0, 1, 2, 3, 4 et en effectuant un affichage des résultats.
- Effectuer un test de la procédure User_Guide dans le module Cli, en utilisant un test_cli.
- Tester le module arbre.

- Évaluer les fonctions du module IP_ADRESS en utilisant une adresse IP et un fichier texte ("ip_test.txt").
- Tester les fonctions et procédures du module LC.
- Tester la table de routage et les sous-programmes qui permettent de la manipuler en utilisant un fichier texte "table_routage_test.txt" et une adresse IP donnée par l'utilisateur.

III. Les difficultés rencontrées et les solutions apportées

Pour plus de détails, nous avons eu des difficultés à comprendre et à utiliser les commandes nécessaires pour configurer les tables de routage. Cela comprenait la compréhension des différents protocoles de routage, ainsi que les commandes pour ajouter, supprimer et modifier les entrées de table.

Nous avons également eu des difficultés à implémenter le module arbre pour les tables de routage, car il s'agissait d'une nouvelle méthode pour nous. Il a fallu comprendre comment organiser les informations dans l'arbre et comment l'utiliser pour effectuer des recherches rapides.

Nous avons egalement rencontré des problèmes dans le parcours du fichier table.txt pour initialiser la table de routage.

Enfin, nous avons rencontré des erreurs lors de l'exécution du programme en ce qui concerne le routage, cela pouvait être dû à des erreurs de configuration ou des erreurs dans l'implémentation de l'arbre et du cache, ce qui a nécessité de rechercher les erreurs et de les corriger pour que le routage fonctionne correctement.

IV. L'état d'avancement et les perspectives d'amélioration

1. L'état d'avancement

A ce stade, nous sommes convaincus que l'ensemble du projet a été correctement implémenté. Les tests unitaires des modules ont été effectués avec succès et donc le routage des paquets contenus dans le fichier d'entrée s'est également déroulé sans problème. Bien que le travail soit considéré comme achevé, il est possible d'optimiser certains aspects de la mise en œuvre pour améliorer les performances en termes de temps d'exécution et d'utilisation de la mémoire.

2. Les perspectives d'amélioration

Dans le cadre des perspectives d'amélioration, nous avons déjà implémenté différents systèmes de cache tels que LRU (Least Recently Used), LFU (Least Frequently Used) et FIFO (First In First Out) pour optimiser les performances de routage. Ces algorithmes de remplacement de cache permettent de gérer efficacement l'espace de cache limité et de maximiser l'utilisation des entrées de routage les plus fréquemment utilisées. Il a été constaté que ces algorithmes ont amélioré les performances de routage en termes de temps

d'exécution et d'utilisation de la mémoire. Cependant, il est encore possible de continuer à étudier d'autres algorithmes de remplacement de cache pour voir s'ils peuvent apporter des améliorations supplémentaires.

D'autres perspectives d'amélioration pourraient inclure :

- L'intégration de protocoles de routage dynamique tels que OSPF (Open Shortest Path First) ou BGP (Border Gateway Protocol) pour permettre au routeur de mettre à jour automatiquement les entrées de table de routage en fonction des changements dans le réseau. Ces protocoles permettent de maintenir une table de routage à jour en échangeant des informations de routage avec d'autres routeurs dans le réseau.
- La mise en place de fonctionnalités de sécurité telles que la filtration de paquets, la mise en place de pare-feux ou la mise en place de VPN pour protéger le réseau contre les menaces potentielles. Ces fonctionnalités permettent de limiter l'accès au réseau et de protéger les données sensibles contre les attaques malveillantes.
- L'utilisation de technologies de virtualisation pour permettre à plusieurs instances de routeur de fonctionner sur une seule machine physique. Cela permet de maximiser l'utilisation des ressources matérielles et de faciliter la gestion des routeurs.
- L'implémentation d'une interface utilisateur graphique pour faciliter la configuration et la surveillance du routeur pour les utilisateurs non techniques.

Il est important de noter que ces améliorations pourraient être complexes à mettre en œuvre et pourraient nécessiter des recherches supplémentaires et des tests approfondis pour s'assurer qu'elles fonctionnent correctement. Il est important de bien planifier et de tester soigneusement toutes les modifications apportées au projet pour s'assurer qu'elles ne causent pas d'erreurs ou de bugs.

BILAN

En résumé, ce mini-projet a été un succès dans la réalisation des objectifs fixés en début de projet. Nous avons réussi à implémenter un routeur fonctionnel capable de lire les entrées de table de routage à partir d'un fichier et de diriger les paquets vers la bonne destination en utilisant ces entrées. Les participants ont également bénéficié de l'apprentissage et du développement de leurs compétences en programmation, en analyse de réseaux et en gestion de projet. Nous avons également mis en place des algorithmes de cache pour améliorer les performances de routage en termes de temps d'exécution et d'utilisation de la mémoire. Enfin, nous avons identifié des perspectives d'amélioration pour le projet telles que l'intégration de protocoles de routage dynamique, la mise en place de fonctionnalités de sécurité, l'utilisation de technologies de virtualisation et l'implémentation d'une interface utilisateur graphique. Il est important de noter que le projet nécessitera des tests et une maintenance régulière pour s'assurer que tout fonctionne correctement et pour intégrer les améliorations futures.

MISE À JOUR

Nous avons procédé à la correction et à l'optimisation de certains modules de test, notamment test_cache_ll, test_cache_la et test_ip_address. Nous avons également ajouté une procédure nommée To_T_Adresse_IP au module IP_ADRESS pour permettre la transformation d'une entrée de quatre entiers en une adresse de type T_Adresse_IP. Nous avons également ajouté une fonction appelée Line_Split au module TABLE_ROUTAGE, permettant la séparation d'une ligne de type Unbounded_String en un tableau contenant des Unbounded_String en utilisant un délimiteur spécifique. Enfin, nous avons ajouté une fonction nommée Add_Line qui prend une ligne, l'analyse avec la fonction Line_Split et enregistre les composants dans la table de routage. Nous avons également ajouté des petites fonctions et procedures pour retrouver les differentes composantes du cache puisqu’il est privé pour les tests. Bref, nous avons procédé à la correction des petits détails dans nos codes pour qu’il soit fonctionnel à 100 pour 100.