Stockage et exploitation de tables de routage

Paul Somson[1], Ayoub Canon[1], and Chourouk El Hassani[1]

[1]Départment Science du Numérique, INP-ENSEEIHT

***Résumé*—Ce document propose une implémentation en Ada d'un module complet pour manipuler les tables de routages. Il passera en revue une méthode pour les stocker et pour les utiliser.**

***Index Terms*—Ada, Routeur, Table de Routage**

I. Introduction

LES tables de routage sont un type de données fondamental dans le réseau (domaine de l'informatique). Leur manipulation peut être délicate, et s'avère, depuis les années 90, un enjeu crucial considérant l'explosion d'internet [1], et la litérrature montre bien un intérêt de recherche dans l'optimisation du stockage et de la manipulation de ces tables de routage [2]. L'objectif de ce document n'est pas de proposer un module optimal pour la manipulation des tables de routage, mais plutôt de donner au lecteur un aperçu des enjeux qui se présentent lors d'un tel exercice, et les solutions (intuitives) que nous proposons.

Nous commencerons alors par une présentations des différents modules implémentés (architecture de l'application), nous irons ensuite plus en détail sur certains algorithmes que nous jugeons clés, enfin nous ferons une ouverture en abordant des points plus globaux sur l'organisation du projet (approche employée faces aux problèmes, etc), chaque auteur proposera son ressenti sur l'application produite.

II. Architecture de l'application

L'application est subdivisée en plusieurs modules. Une stratégie modulaire est conseillée pour des soucis de lisibilité, mais surtout d'évolutivité. Il devient très simple avec une architecture modulaire de mettre à jour un composant ou d'en rajouter. Voici les différents modules (simplifiés) de notre application :

- Définitions des types
  - Arbre Binaire
  - Liste Chaînée Associative (LCA)
- Gestion du Cache
- Traitement de l'Information
  - Traitement des adresses IP (affichage, transformation, ...)
  - Interface humain-machine (console)
- Modules de tests
  - Stockage des résultats
  - Scripts de tests

Dans ce document, nous allons détailler chacun de ces points.

III. Définitions des types

A. Liste Chaînée Associative (LCA)

La premier type que nous avons implémenté est la liste chaînée associative.

Son principe est connu de tous chaque cellule possède deux informations :

- la valeur utile de la cellule,
- un pointeur vers la cellule suivante.

Dans le cadre de cette application, on rajoutera une troisième information, la "donnée" qui sera l'information reliée à l'adresse IP stockée dans la valeur utile.

Il y a de multiples fonctionnalités pour manipuler les listes chaînées associatives [3]. Dans le cadre de notre application, nous avons implémenté les suivantes :

- ajout à la fin,
- changement de valeur d'une cellule,
- supprimer une cellule,
- test de présence.

B. Arbre Binaire

Les arbres binaires (ou *binary search tree*) sont régulièrement utilisés pour le stockage de tables de routage et ont inspirés beaucoup de recherche à ce sujet [4]. Ils sont en effet un compromis agréable entre simplicité de compréhension et efficacité. En effet, leur représentation est triviale (Figure 1), et l'opération d'insertion de détection ou de supression d'une *leaf* est au plus de $\mathcal{O}(\log n)$ dans le cas où l'arbre binaire et tout ses sous-arbres sont équilibrés. Les adresses IPs sont stockées dans les *leaf*.

Note sur le stockage des adresses IPs

La méthode de stockage des adresses IP dans l'arbre a été modifiée. Auparavant, après avoir évalué le bit de poids fort de l'adresse, on décalait l'adresse de 1 bit vers la gauche (on multipliait l'adresse par deux), et on continuait de descendre l'arbre en utilisant cette nouvelle adresse. Une adresse 1000 stockée dans une feuille qui a pour chemin droite-gauche (10) avait donc pour adresse originale 1010 (1000 / 2 + 10 * 2).
Or, cela posait un problème fondamental lors de la détermination d'un élément le moins récemment utilisé (ou autre) : il aurait fallu se souvenir de chaque chemin, ce qui est très consommateur au grands ordres. Ainsi, on a décidé de stocker les adresse IP

entières et non modifiées dans la feuille, en évaluant le ième bit de gauche pour la navigation.

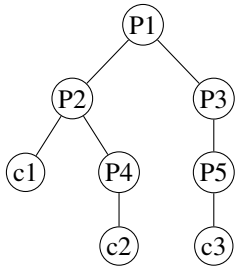

FIGURE 1 – Exemple d'arbre binaire

Le principe de l'arbre est très simple, il consiste en un type "*node*" que l'on définit. Ce type possède plusieurs informations :

— une "clé", qui contient l'information utile de la variable, ici il s'agira de l'adresse IP,
— une "valeur", qui est un type intermédiaire, il contient la valeur associée à l'adresse IP ainsi que son nombre de consultation, ce champ n'est pas natif dans les arbres binaires. Nous l'utilisons uniquement pour cette appliquation précise,
— un "fils gauche" et un "fils droit", qui pointent vers les fils respectifs du *node*.

Il y a deux types de *nodes* :

— les *nodes* racines, qui contiennent un ou deux *child*, on dit que chaque node racine est la racine d'un "sous-arbre binaire", ainsi, chaque arbre binaire est par construction récursif,
— les *nodes* "feuilles" ou *leaf* qui ne possèdent pas d'enfant.

Il existe de multiples opérations sur les arbres binaires [5]. Nous n'en avons implenté qu'une sélection :

— suppression d'une *node* basé sur la clé,
— l'ajout d'une *node* dans l'arbre,
— le traitement de chaque *node* de l'arbre,
— la suppression d'un arbre.

Ces fonctionnalités sont suffisante dans le cadre du stockage d'une table de routage.

IV. Deux algorithmes clés

Dans cette partie on explore plus en détails deux algorithmes réalisant les fonctionnalités décrites précédemment.

A. LCA : Suppression de donnée

La première fonction que nous allons détailler est la suppression d'une cellule en connaissant sa clé. On parle ici non pas d'une suppression de la donnée mais réellement de libérer l'espace mémoire associé à la cellule. Sur les figures 2 et 3, vous pouvez voir le fonctionnement de cet algorithme lorsque l'on souhaite enlever la cellule de clé "127.0.0.11".

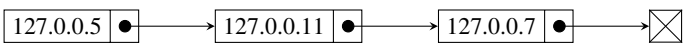

FIGURE 2 – Exemple de LCA pour notre application

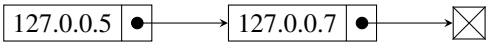

FIGURE 3 – LCA après suppression de la cellule ayant pour clé "127.0.0.11"

L'algorithme 1 présente l'implémentation algorithmique de cette fonctionnalité.

Algorithm 1: procedure Supprimer()

```
input : Sda : T_LCA, Cle : T_CLE
output: Sda : T_LCA
Description Supprime la donnée associée à une clé
 dans la LCA
T_LCA Aux ← Sda
if Sda = NULL then
 | raise Cle_Absente_Exception
else if Sda.All.Cle = Cle then
 | Sda ← Sda.All.Suivant
 | Free(Aux)
else
 | Supprimer(Sda.All.Suivant, Cle)
end
```

B. Arbre Binaire : Suppression de donnée

Nous allons à présent voir l'analogue de la fonction de suppression d'une donnée dans une LCA mais dans un arbre binaire.

Comme nous l'avons vu précédemment, il existe plusieurs types de *nodes*, les "racines" et les "*leaf*". Bien sûr, la méthode pour supprimer une racine d'une *leaf* diffère. On peut par ailleurs subdiviser les racines en deux sous-catégories :

— les racines avec un seul enfant (droit ou gauche),
— les racines avec deux enfants.

Nous avons alors trois types distincs de *nodes* pour les arbres binaires, c'est à dire trois méthodes de suppressions.

1) Cas où la node est une leaf: C'est le cas le plus simple, car il ne nécessite pas de restructuration de l'arbre binaire. Ainsi, on libère simplement la mémoire de la cellule en question, sans toucher au reste de l'arbre binaire. Par exemple, prenons l'arbre en figure 1, on souhaite supprimer la cellule "c3", le résultat de l'algorithme est indiqué en figure 4. On note toutefois, qu'ici nous ne nous intéressons pas au caractère équilibré de l'arbre binaire. Si l'on souhaitait garder l'abre binaire équilibré (pour bénificier notamment de la complexité d'accès au pire de $\mathcal{O}(\log n)$) il faudrait modifier l'algorithme pour rééquilibrer l'arbre binaire après la suppression.

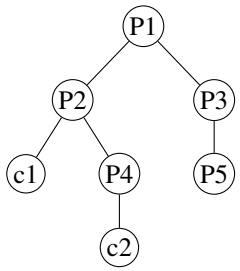

FIGURE 4 – Suppression d'une donnée *leaf* d'un arbre binaire

L'algorithme est alors très simple, comme expliqué précédemment, il s'agit simplement de libérer la mémoire de la cellule en question.

Algorithm 2: procedure Supprimer_leaf()

```
input : Arbre : T_LA, Cle : T_ADRESSEIP
output: Arbre : T_LA
Description Supprime la donnée associée à une clé
 dans l'arbre binaire (cas où la clé est une leaf)
if Arbre.All.Gauche = NULL and Arbre.All.Droite =
 NULL then
 | Free(Arbre)
end
```

2) Cas où la node n'a qu'un enfant: Un autre cas très simple à gérer, puisqu'il consiste simplement à remplacer la *node* à libérer par son enfant gauche ou droit. Appliqué à l'abre binaire en figure 1, en supprimant la cellule "P3", on obtient alors le résultat en figure 5. Ici encore, nous ne nous attardons pas sur l'équilibrage de l'arbre binaire.

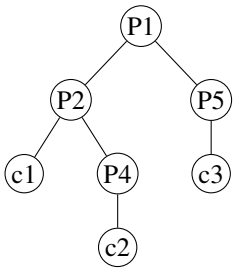

FIGURE 5 – Suppression d'une donnée racine avec un seul enfant d'un arbre binaire

Une fois encore, l'algorithme est très simple (algorithme 3). La subtilité est de bien faire attention à mettre la cellule courant dans une variable temporaire pour éviter une perte non contrôlée de données.

Algorithm 3: procedure Supprimer_racine1()

```
input : Arbre : T_LA, Cle : T_ADRESSEIP
output: Arbre : T_LA
Description Supprime la donnée associée à une clé
 dans l'arbre binaire (cas où la clé est une leaf)
A_Detruire ← Arbre
if Arbre.All.Gauche = NULL and Arbre.All.Droite !=
 NULL then
 | Arbre ← Arbre.All.Droite
 | Free(A_Detruire)
else if Arbre.All.Gauche != NULL and Arbre.All.Droite
 = NULL then
 | Arbre ← Arbre.All.Gauche
 | Free(A_Detruire)
end
```

V. TEST DES FONCTIONNALITÉS ET DE L'APPLICATION

Pour tester tout le programme, nous avons réalisé des fichiers de tests propre pour le fonctionnement des listes chaînées associatives et des arbres binaires. L'idée générale est basé sur deux stratégies :

— on affiche le log dans le terminal au fur et à mesure de l'exécution,

— utiliser la fonction *assert()* de ada pour tester d'éventuelles erreur de formattage ou d'exécution. En effet cette fonction arrête le programme en cas de non validation de la condition. Si le programme arrive à la dernière ligne sans problèmes, alors cela conditionne que l'application fonctionne correctement.

VI. ORGANISATION DE L'ÉQUIPE

Pour ce projet, nous avons utilisé git comme outil de versionning, et GitHub pour la plateforme. Nous avons reparti le travail en fonction des types de données. L'un se focalisait sur les listes chaînées associatives, un autre sur les arbres binaires, et enfin le dernier membre s'est chargé de la mise en oeuvre de tous les modules.

VII. AMÉLIORATIONS POSSIBLES

Comme indiqué en introduction, l'intérêt pour les tables de routage et l'optimisation de leur stockage / manipulation était vif dans les années 90.

Il semblerait que la litérrature converge vers l'utilisation des arbres binaires, c'est donc la solution que nous decidons de garder pour entrevoir une amélioration à notre application.

Draves *et al,* proposent une représentation optimale pour la minimisation du nombre de préfixe [6]. Leur travail suggère une amélioration de 60% de la performance, et se base sur des algortihmes relativement simple mais séquencés en trois "passages" ou *pass* successifs. Ce procédé est appellé *Optimal Routing Table Constructor* (ORTC), a été testé sur de nombreuses bases publiques, attestant de sa viabilité. Cela semble être une bonne première piste d'amélioration.

Un problème peut également surgir lors de la manipulation de table à très grande échelle. Il peut alors devenir pratique, voire obligatoire, de séparer les données sur plusieurs espaces de stockages différents (ou "agents"). C'est ce qu'on appelle alors un problème de décentralisation. La décentralisation est au coeur de beaucoup de recherche contemporaine et révèle de nombreux enjeux mathématiques [7], mais surtout sur des thèmes pratiques comme l'optimisation [8] et l'apprentissage profond [9]. Les tables de routage n'ont pas fait exception, et l'émergence de ces enjeux de décentralisation ont ravivé un certain intérêt dans l'optimisation de ces tables à grande échelle [10]. Ces notions de décentralisations font intervenir inévitablement des notions de graphes et donc de topologie (géométrie du réseaux d'agents).

Il pourrait être alors intéressant de repenser notre application pour être capable de manipuler de très larges tables de données.

Enfin, nous pouvons proposer de réaliser cette application en C, et ainsi gagner significativement en performance. Les tables de routages étant conceptuellement très proches du matériel, certaines implémentations hardwares [11] sont possible pour encore plus accélérer le processus de traitement.

VIII. Discussion et bilans personnels

A. Ayoub

Dans l'ensemble, j'ai trouvé ce projet très intéressant et instructif. Le plus gros challenge a été le travail en équipe, il est parfois compliqué de se mettre d'accord sur certains points et de correctement se partager le travail. De plus, les subtilités du language Ada m'ont forcé plusieurs fois à penser différemment l'implémentation de mes idées. Cependant, ce projet m'a permis d'appliquer ce que j'avais appris en cours dans un projet plus concret que les TP.

B. Paul

C'était un projet passionnant qui a su me mettre en difficulté. J'ai beaucoup apprécié les subtilités des algorithmes mis en place pour l'arbre et le cache, et j'en ai tiré de grandes leçons. Le travail en groupe étant une difficulté, ce projet m'a aidé à développer mon esprit d'équipe, et j'en garde un souvenir très positif.

C. Chourouk

Ce projet était instructif mais plutôt difficile au début . En fin de compte, j'ai réussi à dépasser mes difficultés initiales grâce à un travail d'équipe solide et à une meilleure compréhension des exigences du projet .Ce projet m'a permis d'acquérir des compétences plus approfondies en programmation impérative.Il m'a aussi permis de prendre des initiatives et travailler au sein d'un groupe.

IX. Conclusion

En conclusion nous avons réalisé une application basique permettant de stocker et de manipuler des tables de routage en Ada. Notre approche algorithmique permettra au lecteur de facilement implémenter la même application dans n'importe quel autre langage de programmation.

Nous rappellons que l'objectif de ce document est simplement de donner un aperçu des techniques et enjeux recontrés lors de la manipulation de telles tables, et qu'en aucun cas il s'agit ici d'une implémentation optimale, tant sur la performance, que sur le scalabilité et la sécurité.

Merci d'avoir lu notre rapport. <

Références

[1] Tian Bu, Lixin Gao, Don Towsley et al. "On routing table growth". In : *ACM SIGCOMM Computer Communication Review* 32.1 (2002), p. 77.

[2] Keith Sklower. "A tree-based packet routing table for Berkeley unix." In : *USENIX Winter*. T. 1991. 1991, p. 93-99.

[3] Nick Parlante. "Linked list basics". In : *Stanford CS Education Library* 1 (2001), p. 25.

[4] Sartaj Sahni et Kun Suk Kim. "An O (logn) dynamic router-table design". In : *IEEE Transactions on Computers* 53.3 (2004), p. 351-363.

[5] Erkki Mäkinen. "A survey on binary tree codings". In : *The Computer Journal* 34.5 (1991), p. 438-443.

[6] Richard P Draves et al. "Constructing optimal IP routing tables". In : *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*. T. 1. IEEE. 1999, p. 88-97.

[7] Radu Alexandru Dragomir, Mathieu Even et Hadrien Hendrikx. "Fast stochastic bregman gradient methods : Sharp analysis and variance reduction". In : *International Conference on Machine Learning*. PMLR. 2021, p. 2815-2825.

[8] Angelia Nedić, Alex Olshevsky et Michael G Rabbat. "Network topology and communication-computation tradeoffs in decentralized optimization". In : *Proceedings of the IEEE* 106.5 (2018), p. 953-976.

[9] Michael Kamp et al. "Efficient decentralized deep learning by dynamic model averaging". In : *Joint European conference on machine learning and knowledge discovery in databases*. Springer. 2019, p. 393-409.

[10] Christoph Lenzen et Boaz Patt-Shamir. "Fast routing table construction using small messages". In : *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. 2013, p. 381-390.

[11] Anthony J McAuley et Paul Francis. "Fast routing table lookup using CAMs". In : *IEEE INFOCOM'93 The Conference on Computer Communications, Proceedings*. IEEE. 1993, p. 1382-1391.