Rapport
Projet UE d'ouverture

compression du arbre binaire recherche

Qiwei XIAN, Ruiwen WANG

28 novembre 2019

Table des matières

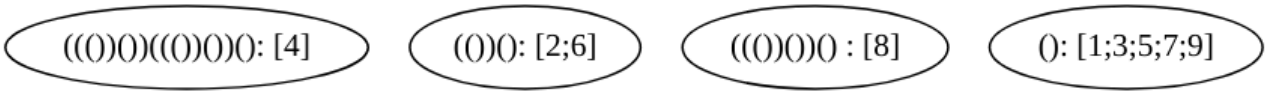
1 Objectif

L'objectif du jeu est de construire une structure compressée depuis les arbres binaires de recherche(ABR) en utilisant 2 façon (list et map) de mettre des noeuds isomorphes à même noeud.Grâce à cette structure,nous pouvions gagner d'esapce mémoire. Afin d'étudier l'efficacité de recherche et le taux de compression, on va les étudier par rapport à complexité en temps et l'occupation d'espace mémoire.

2 Methode réaliste

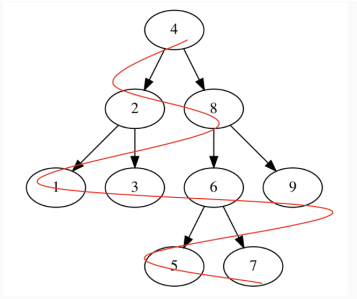
2.1 Algorithme pour compresser ABR en `arbre_compression_liste`

Compression de l'arbre binaire 1. Mettre en ensemble les nœuds isomorphes dans une liste et on peut obtenir pair list :

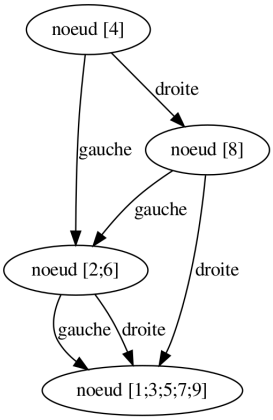


2. Parcourir la liste de pair et construire les nœuds internes de l'arbre compression liste et les stocker dans un table de hachage.

3. Parcourir ABR en largeur afin de connecter les nœuds



Recherche une valeur dans un `arbre_compression_liste` :
Parcourir la liste de noeud. S'il existe une valeur $i, i > v$. alors on va fils gauche. Sinon on va fils droite.



2.2 Algorithme pour compresser ABR en `arbre_compression_map`

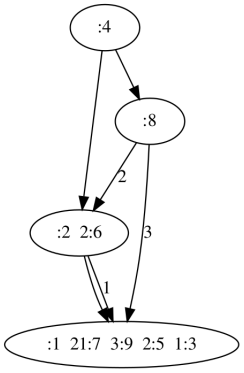
Compression de l'arbre binaire

- Associer les clés de l'ABR et les mots de parenthèse correspondant dans un table hachage.
- Mettre les parenthèses et les noeuds vides de l'arbre compression map dans un table hachage.
- Parcourir l'ABR en profondeur, on met à jour le noeud actuel, son père , un compteur de nombre de noeud isomorphe et le chemin des étiquette qu'on a passé, pour chaque noeud vide qu'on a stocké dans un hashtable, on ajoute le clé de noeud actuel à la fois connecter lui et son père. Pour insérer un clé, il y a 3 situations :

  Le noeud est nouvelle structure, on construire un noeud de compression et mettre le mot vide comme le clé et la valeur est celle de noeud ABR. En plus créer un lien entre son père et ce noeud, l'étiquette de lien est aussi mot vide.

  Le noeud est isomorphe et son père n'a pas encore le lien. Incrémenter le compteur de nombre de noeud isomorphe, et créer le lien entre ce noeud et son père. L'étiquette est la valeur du compteur. En plus on merge ce noeud avec un autre noeud isomorphe.

  Le noeud est isomorphe et son père a déjà eu le lien. Il faut concaténer le chemin qu'on a mis à jour et cette étiquette. En plus on ajoute la valeur de ABR dans le noeud isomorphe et sa clé est égale à résultat de concaténation.
- Continuer à parcourir en profondeur.



Recherche une valeur dans un `arbre_compression_liste` :

- Clé initial est mot vide.
- Comparer la valeur de but avec la valeur qui est associé du clé dans le nœud actuel. Si la valeur de but est plus petite, on parcourt le fils gauche, la clé devient le résultat de la concaténation de clé précédent et l'étiquette de transition gauche.
- Sinon on parcourt le fils droite.

2.3 Modules

`Abr.ml`→ la définition de Abr et ses méthodes
`main.ml`→la fonction main
`com_list_abr.ml`→la définition de `arbre_compression_liste` et les méthodes
`com_map_abr.ml`→la définition de `arbre_compression_map` et les méthodes

`generation_list.ml`→les fonctions pour générer une liste aléatoire.
`Util.ml`→les fonctions de print makefile

2.4 Structure des données

```
type abr = Null_abr
| Abr of int * abr * abr
```

le type abr est un stucture classique de l'arbre binaire de recherche

```
type pair = Null_pair
| Pair of string * ((int list) ref)
```

le type pair est un structure contient des parenthèses et une list des clés du noeuds isomorphes

```
type compressor = Null_com
| Compressor of (compressor ref) * (int list) * (compressor ref)
```

le type pair est un structure d'arbre consiste à des noeuds contient une liste des clés du noeuds isomorphes

```
type compressor_map =Null_com_map
| Compressor_map of ((compressor_map ref) * eti ref * (string, int)Hashtbl.t
* eti ref * (compressor_map ref))
and eti = Null_eti | Eti of string
```

le type pair est un structure d'arbre consiste à des noeuds contient une hashmap liée des étiquettes et des clés du noeuds isomorphes

```
type 'a transition = Transition of ('a * 'a)
```

Le type transition est un structure consiste à afficher un arbre en format pdf, il représent un arête entre deux nœuds.

2.5 Les fonctions détaillées

(1) **Abr.ml**
- (a) construct_ast list
  val construct_ast : int list → abr = <fun>
  construire un ABR par une liste int
- (b) search arbre valeur
  val search : abr → int → bool = <fun>
  rechercher une valeur dans un ABR
- (c) displayAST (root : abr)
  val displayAST : abr → unit = <fun>
  Afficher un ABR

(2) **Com_list_abr.ml**

(a) tree_traversal (abr : abr)
val tree_traversal : abr → pair list ref * (int, string) Hashtbl.t ref = <fun>
Parcourir ABR, rendre un hashtable qui associe la valeur de nœud et le mot de parenthèse. une liste de pair.

(b) tree_traversal (abr : abr)
val tree_traversal : abr → pair list ref * (int, string) Hashtbl.t ref = <fun>
Parcourir ABR, rendre un hashtable qui associe la valeur de nœud et le mot de parenthèse. une liste de pair.

(c) make_compressor (pair : pair)
val make_compressor : pair → compressor = <fun>
Construire un nœud de l'arbre_compressor_list

(d) pairList_to_map (list : pair list)
val pairList_to_map : pair list → (string, compressor ref) Hashtbl.t ref = <fun>
Transférer la liste de pair à un hashtable qui associe le mot de parenthèse et la référence de nœud de l'arbre_compression_list

(e) connect_node (abr : abr) (lib : (string, compressor ref) Hashtbl.t ref) (value_mot : (int, strin g) Hashtbl.t ref)
Parcourir ABR afin de connecter les nœuds de l'arbre_compression_list

(f) compress_ast (abr : abr) val compress_ast : abr → compressor = <fun>
Compresser un ABR et rendre l'arbre_compression_list

(g) search (com : compressor) (value : int)
val search : compressor → int → bool = <fun>
Vérifier si l'arbre_compression_list contient une élement "value"

(h) displayCompressor (root : compressor)
val displayCompressor : compressor → unit = <fun>
Afficher un arbre_compression_list en format PDF

(3) **Com_map_abr.ml**

(a) vide_compressor_map()
val vide_compressor_map : unit → compressor_map ref = <fun>

(b) add (com : compressor_map ref) (key :string) (value :int)
val add : compressor_map ref → string → int → unit = <fun>
Ajouter la clé et la valeur de nœud ABR dans le nœud de l'arbre_compression_map

(c) tree_traversal_map (abr : abr)
val tree_traversal_map : abr →(int, string) Hashtbl.t ref * (string, compressor_map ref) Hashtbl.t ref =<fun>
Parcourir le ABR et rendre la référence d'un hashtable qui associe la valeur de nœud ABR et le mot de parenthèse, ainsi que la référence d'un hashtable qui associe le mot de parenthèse et le nœud de l'arbre_compression_map.

(d) connect_node (abr :abr) (mot_noeud : (string, compressor_map ref) Hashtbl.t ref) (value_mot : (int, string) Hashtbl.t ref)
val connect_node : abr → (string, compressor_map ref) Hashtbl.t ref → (int, string) Hashtbl.t ref → unit = <fun>
Construire et connecter les noeuds de l'arbre_compression_map en cours de parcours ABR en profondeur.

(e) compress_map_ast abr compress_map_ast abr
val compress_map_ast : abr → compressor_map = <fun>

Compresser ABR en l'arbre_compression_map

(f) search (root : compressor_map) (value : int)
val search : compressor_map → int → bool = <fun>
Vérifier si la value existe dans l'arbre_compression_map

(g) displayCompressorMap (root : compressor_map)
val displayCompressorMap : compressor_map → unit = <fun>
Afficher l'arbre_compression_map en format PDF

(4) **Generation_list.ml**

(a) remove_at list n
val remove_at : 'a list → int → 'a list = <fun>

(b) makelist n
val makelist : int → int list = <fun>

(c) extraction_alea l p
val extraction_alea : 'a list → 'a list → 'a list * 'a list = <fun>

(d) gen_permutation n
val gen_permutation : int → int list = <fun>
Générer aléatoirement une liste de longueur n

(5) **Util.ml**

(a) print_list l
val print_list : int list → unit = <fun>
Afficher les éléments dans la liste int

(b) print_list_list (l : (int list) list)
val print_list_list : int list list → unit = <fun>
Afficher les éléments dans la liste de liste int

(c) print_pair (p : pair)
val print_pair : pair → unit = <fun>
Afficher les éléments de la liste de pair

(d) print_pair_list p_List
val print_pair_list : pair list → unit = <fun>

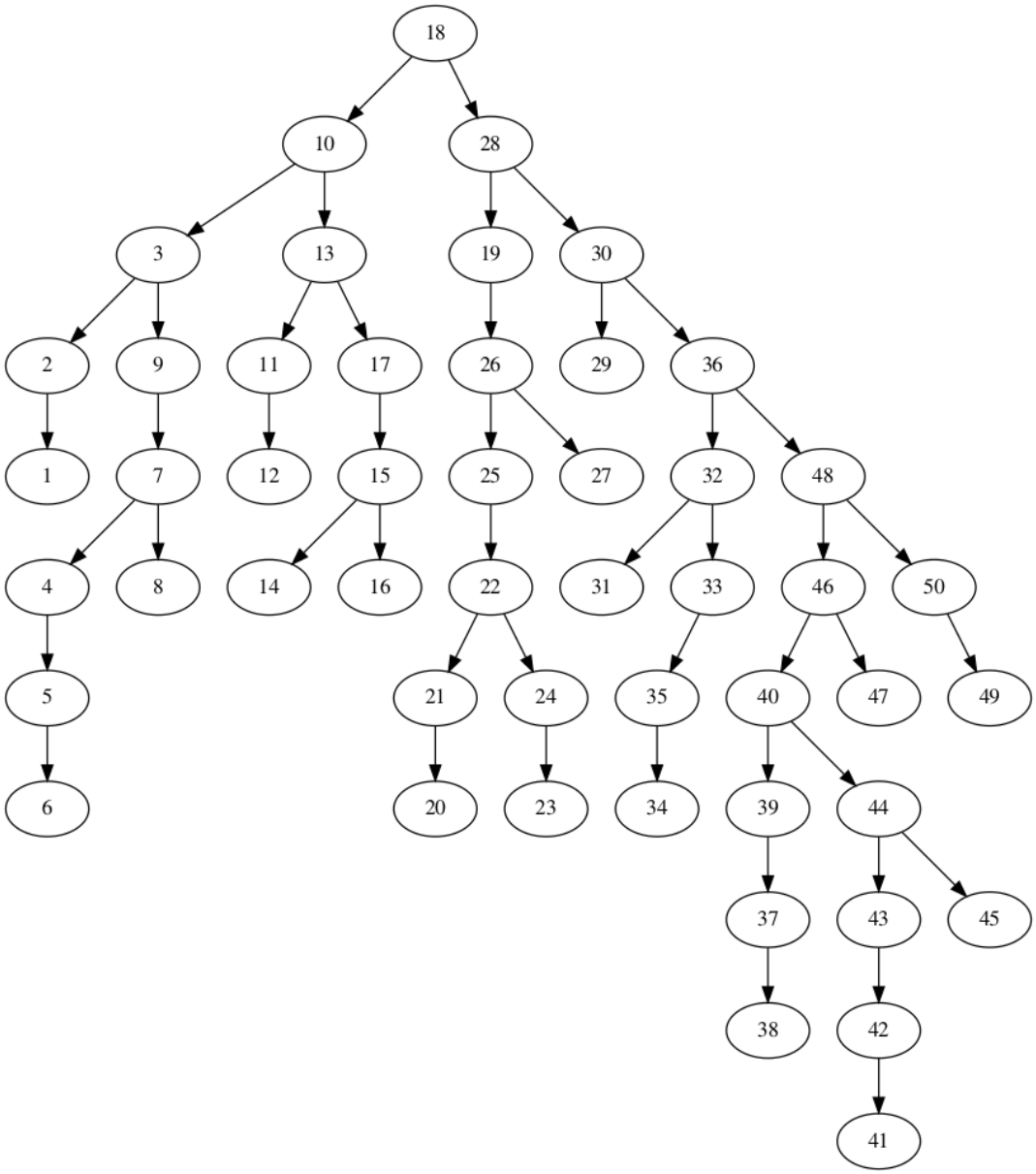
(e) parse_integers s
val parse_integers : Scanf.Scanning.file_name → int list = <fun>
Transférer le contenue de stdin en une liste int

(6) **main.ml**

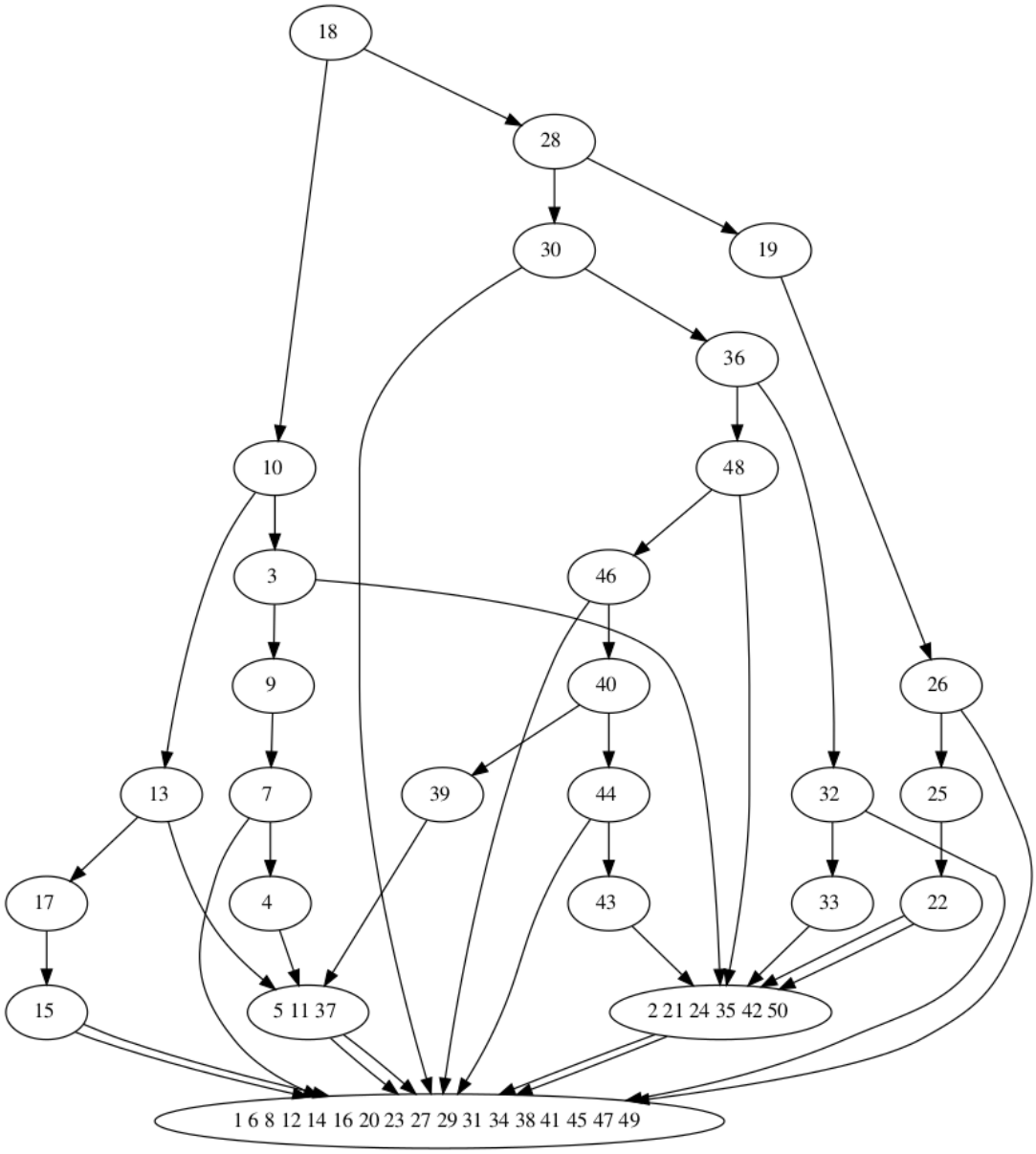
(a) space f list
val space : ('a → 'b) → 'a → unit = <fun>
Calculer l'espace mémoire occupé par la fonction f

(b) time_search f x y
val time_search : ('a → 'b → 'c) → 'a → 'b → 'c = <fun>
Calculer le temps d'exécution de la fonction f

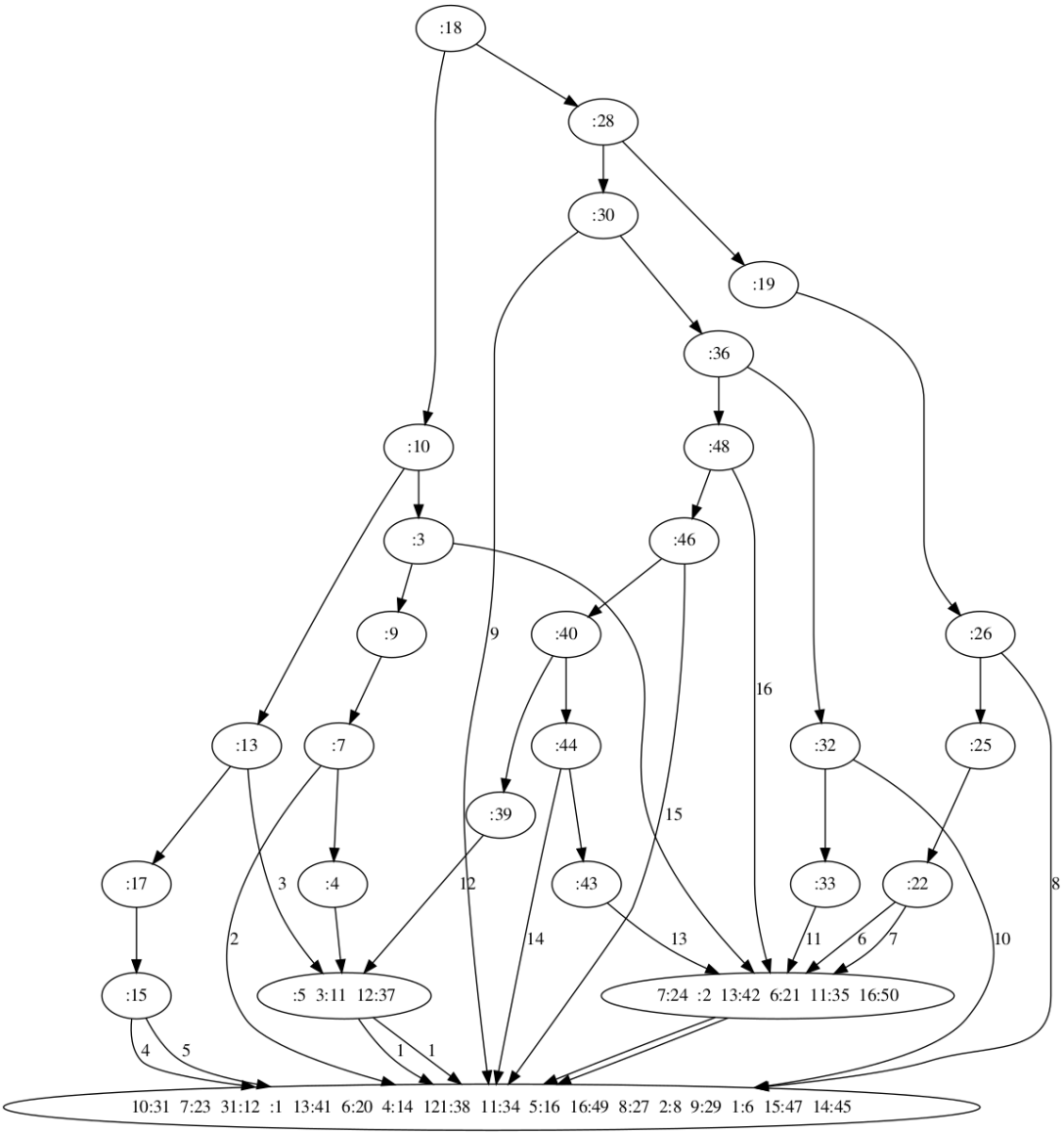
3 Resultats obtenus



Arbre binaire de recherche à tester



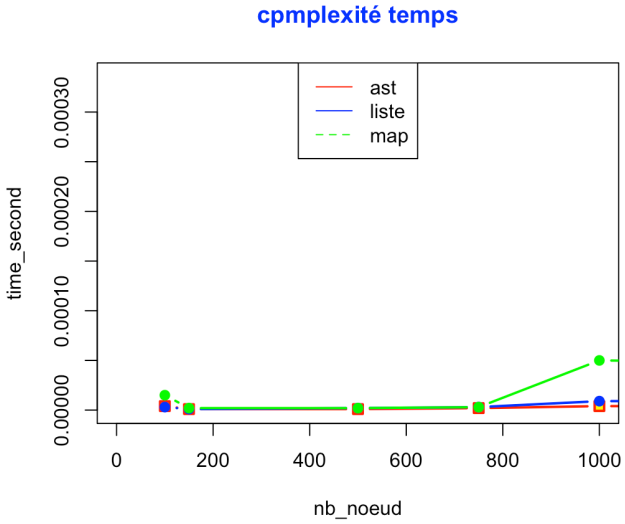
ABR compressé version liste

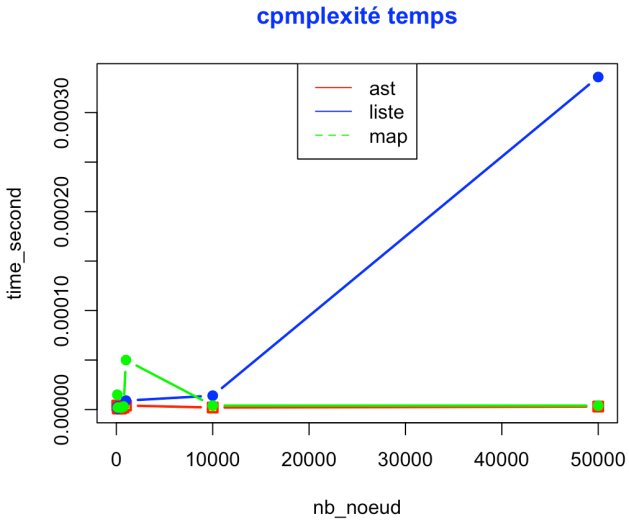


ABR compressé version map

4 Experimentations Réponse à l'exercices demandés

Question 3.11
Effectuer une etude experimentale de complexite en temps des algorithmes de recherche. Pour ce faire, on combinera les sections 1. et 2. du devoir.
Agrementer l'etude de graphiques et de commentaires.
Réponse





On a testé le pire cas de faire une recherche pour chaque structure.
Sur les deux schéma , on peut observer que quand on tester de faire rechercher une valeur dans les 3 structures : globalement la complexité du temps est :

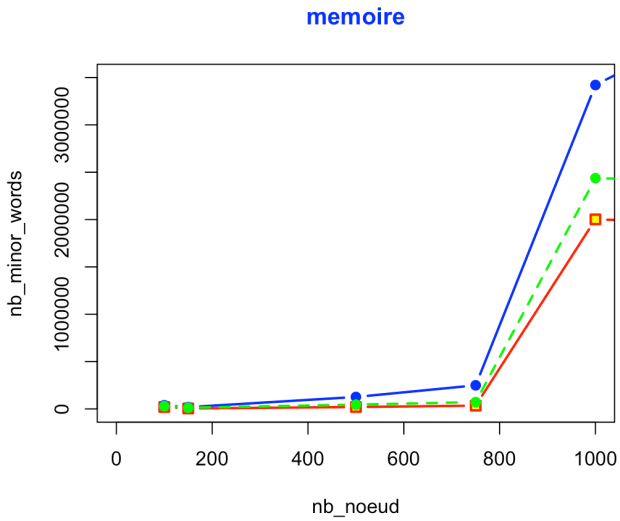
$$ABR < ABR\ Compress\ map < ABR\ Compress\ Liste$$

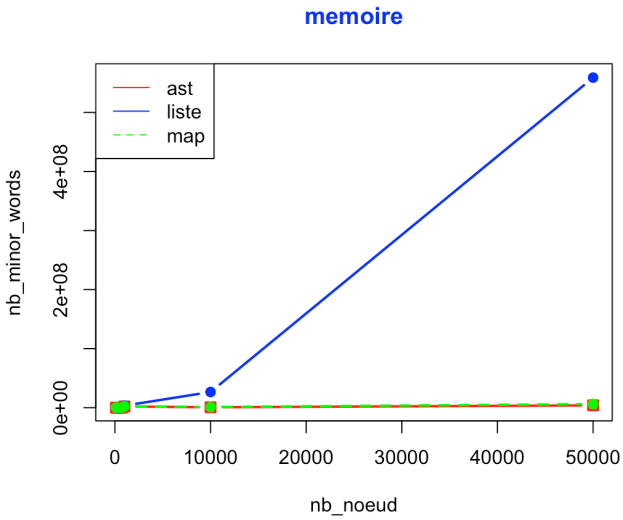
sauf que il y a une exception en nœud =100 et en nœud =1000 ABR Compressé map.
Sa complexité a montrée beaucoup que les autres , on pense que c'est parce que car la liste avec nœud 100 et la liste avec nœud 1000 sont triée et croissante donc hauteur du ABR compressé a augmenté.

Question 3.12

Effectuer une etude experimentale de complexite en espace du gain de la compression des ABR.
Agrementer l'etude de graphiques et de commentaires.

Réponse





On a testé le nombre de minor_word occupé par chaque structure.
Sur les deux schéma , on peut observer que : globalement la complexite d'espace mémoire est :

$$ABR < ABR\ Compress\ map < ABR\ Compress\ Liste$$

entre 0 et 1000 noueds , les trois structures
sa complexité a montrée beaucoup que les autre , on pense que c'est parce que car la liste avec nœud 100 et la liste avec nœud 1000 sont triée et croissante donc hauteur du ABR compressé a augmenté.

Question 3.13

Pour chacun des fichiers de tests, construire l'ABR compresse associe a la liste d'entiers puis calculer le nombre de nœuds internes et le nombre moyen de cles par nœud interne.

Réponse

nombre des nœuds / nombre de cles par nœuds : 100 : 100 nœuds / 1 clé par nœuds
150 : 59 nœuds / 2.5423 clés par nœuds
500 : 176 nœuds / 2.8409 clés par nœuds
750 : 258 nœuds / 2.906 clés par nœuds
1000 : 1000 nœuds / 1 clé par nœuds
10000 :2497 nœuds / 4.0048 clés par nœuds
50000 :10846 nœuds / 4.6099 clés par nœuds

5 Disscussion Difficulités rencontrés

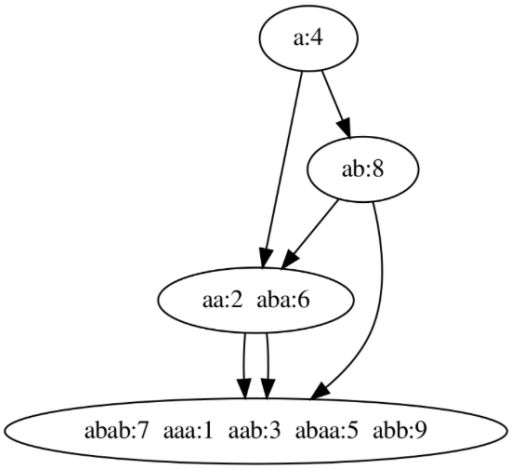
1.Tout d'abord,on ne savait pas utilisé quel façon pour calculer la complexité d'espace mémoire ,on a cherché plein de méthodes mais il en a beaucoup ne marchent pas , donc à la fin , on utilise la Module GC pour le déterminer.
On a éssayé d'utiliser l'outil online `OCaml Memory Profiler` mais on n'arrive pas trouver le bon compileur.

```
ruiwenwang@RuiwendeMBP ~ % opam switch create 4.01.0+ocp1-20150202
[ERROR] No compiler matching '4.01.0+ocp1-20150202' found, use 'opam switch list
-available' to see what is available, or use '--packages' to select packages exp
licitly.
```

Pour la suite ,on a éssayé d'utilisé l'outil extérieur `Ocamlviz` :
mais il est rassemble de manquer un fichier ,donc aussi on n'arrive pas l'utiliser.

```
File "camlp4/pa_ocamlviz.ml", line 1:
Error: I/O error: camlp4of 'camlp4/pa_ocamlviz.ml' > /var/folders/rv/2_8v4s8s0q1
01mm0tlkgr9400000gn/T/ocamlppdf3319
make: *** [camlp4/pa_ocamlviz.cmo] Error 2
```

2.



Tout d'abord, on a construit une structure comme ci-desus :
on a utilisé chaine de caractère comme les étiquettes ; mais pendant on faisait la partie `Experimentations`.
On a trouvé qu'il prends beaucoup d'espace mémoire à casue des chaines de caractère ,donc on a cherché une structure plus éfficace que cela , vous pouvez voir dans la partie `Algorithme`.

3. On a utilisé <<Makefile>> pour simplifier compilation et l'exécution du code.
ex. make ast pour produire un PDF qui contient l'image de cette abr
4. Par rapport à l'occupation d'esapce mémoire , n'import `ABR_compressé_liste`
et ABR compressé map ils prennent plus d'espace mémoire ,indiqué par le nombre de `minor words`, on pensait c'est à cause de langage Ocaml utilisent beaucoup des fonctions récursives donc les packages ne peuvent pas bien être collecté.