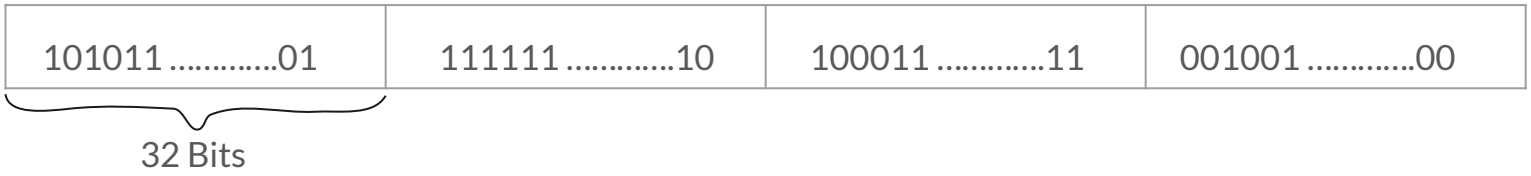
SORBONNE UNIVERSITÉ

Projet d'Algorithmique Avancée

BOUZOURINE Hichem 21319982
MAOUCHE Mounir 21315128

Echauffement

Structure des clés



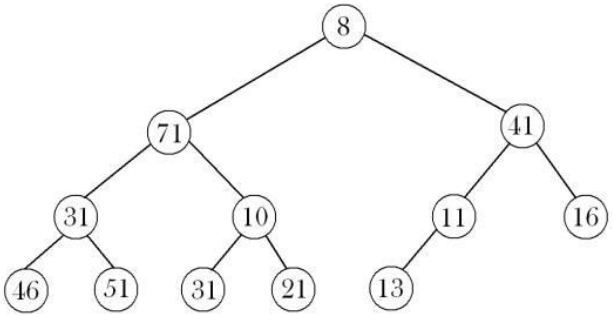
Prédicat Inf: Compare deux clés de 128 bits en convertissant en valeurs non signées et en comparant les entiers 32 bits pour déterminer si la première clé est strictement inférieure à la seconde.

Prédicat Sup: Renvoie false dès qu'elle détecte une différence au niveau d'un bit, indiquant ainsi que les clés ne sont pas égales, sinon elle retourne true.

Tas min

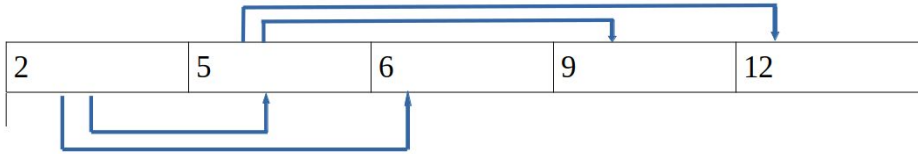
Tas sur Arbre

Représenté avec des Noeuds contenant une **Clé128** et **2 descendants**.



Tas sur Tableau

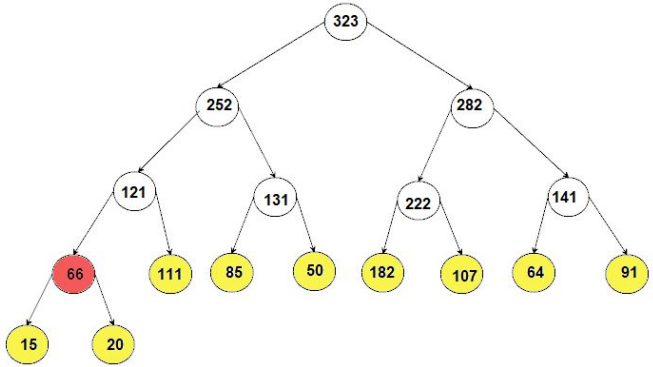
Représenté avec un tableau de **Clés128** bits,



Fonctions implémentées:

- SupprMin
- Ajout
- AjoutsIteratifs

Fonction Construction



323	252	282	121	131	222	141	66	111	85	50	182	107	64	91	15	20
323	252	282	121	131	222	141	15	111	85	50	182	107	64	91	66	20
323	252	282	121	131	222	64	15	111	85	50	182	107	141	91	66	20
323	252	282	121	131	107	141	15	111	85	50	182	222	64	91	66	20

Fonction Union

1. Récupérer la liste des éléments des deux **Tas**.
2. Insérer dans un “**Set**”.
3. Utiliser la fonction **Construction** pour obtenir l’union optimisé

Complexité

SupprMin: Θ(log (n))

1. Chercher la valeur la plus à droite dans le dernier niveau d'un arbre **log(n)**.
2. Rétablir l'ordre d'infériorité à partir de la racine, aussi **(log n)**

Ajout: Θ(log (n))

1. Identifier la position d'insertion pour une nouvelle valeur dans un arbre binaire équilibré log(n).
2. ajuster l'arbre pour maintenir sa structure correcte (les remontées), (log n) opérations.

AjoutsItératifs: Θ(n.log (n))

Appel de la fonction 'Ajout' **n** fois:

Complexité

Construction: Θ(n)

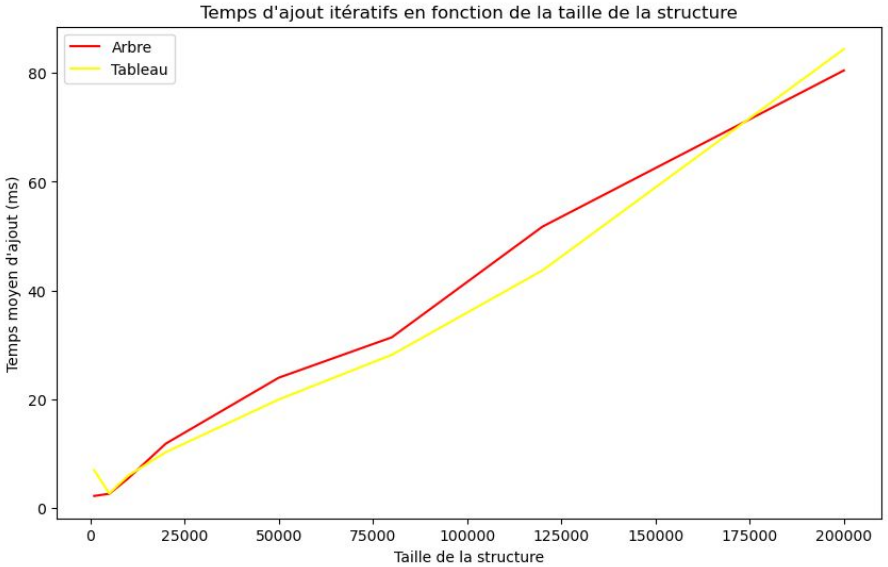
$$\begin{aligned}\sum_{h=0}^{\lfloor \log n \rfloor} \left(\frac{n}{2^h} \cdot h \right) &= n \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h} \\ &\leq n \sum_{h=0}^{\infty} \frac{h}{2^h} \\ &\leq O(n) \\ &= O(n)\end{aligned}$$

Union: Θ(n + m)

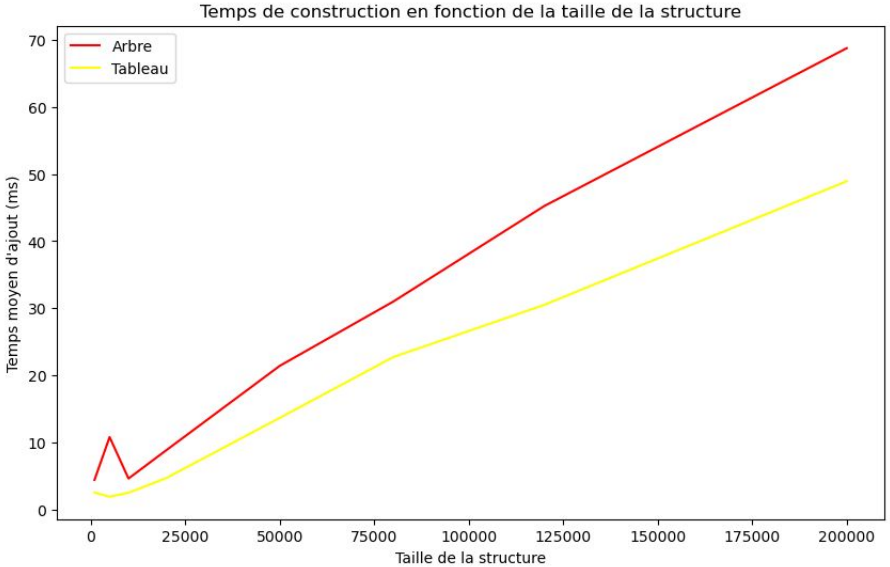
1. 2 tas de taille **n** et **m**. On parcourt chacun en une complexité de **Θ(n)** et **Θ(m)**.
2. construire la liste des élément (sans doublons) en complexité de Θ(n + m)
3. avec la fonction construction, puisqu'elle est linéaire, la complexité finale sera en Θ(n + m)

Représentation Graphique de la complexité

AjoutsIteratifs

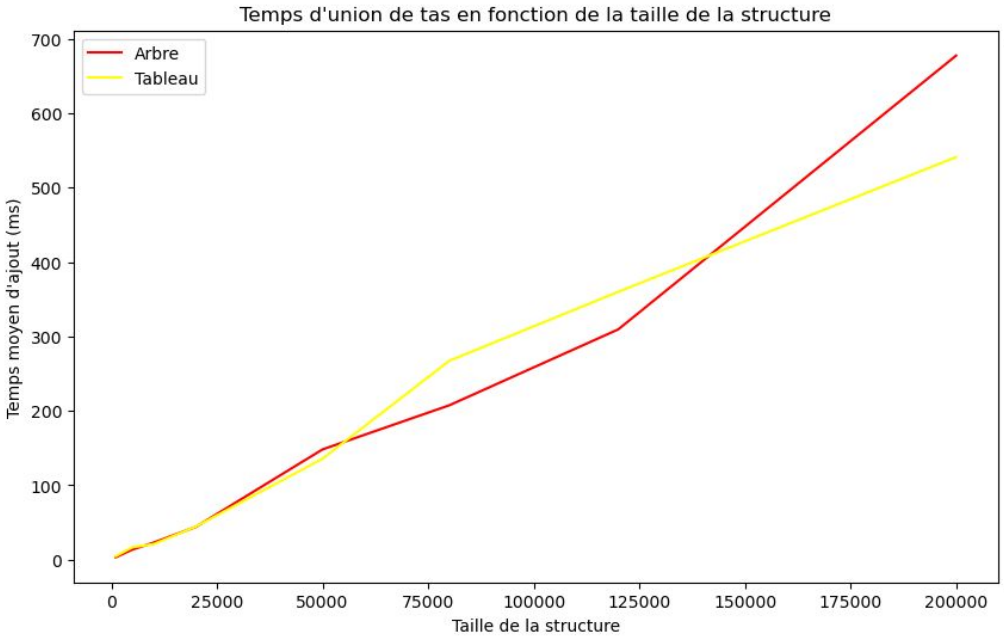


Construction



Représentation Graphique de la complexité

Union



Files binomiales

Files Binomiales

Suite de Tournois Binomiaux de taille décroissante. Elle possède les primitives:

— **union** : Fait l'union de 2 tournoi binomial de même taille.

— **décapite** : Renvoie la file binomiale obtenue en supprimant la racine du tournoi binomial

— **file** : Renvoie la file binomiale réduite au tournoi binomial courant

— **getFils** : Renvoie le fils présent à l'indice donnée en paramètre

Files Binomiales

Fonction implémentées:

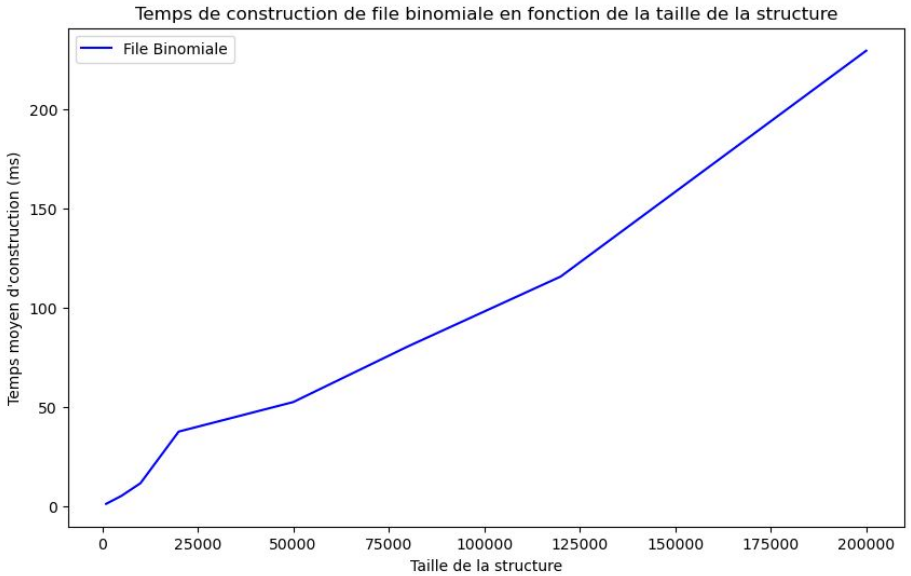
Union: dans l'union de deux files binomiales il faut regarder est-ce qu'on a un retenu ou non, puis on applique l'algorithme qui se trouve dans le cours.

Ajout: Pour faire l'ajout dans les files binomiales, on se base sur l'union de la file avec un tournoi binomiale.

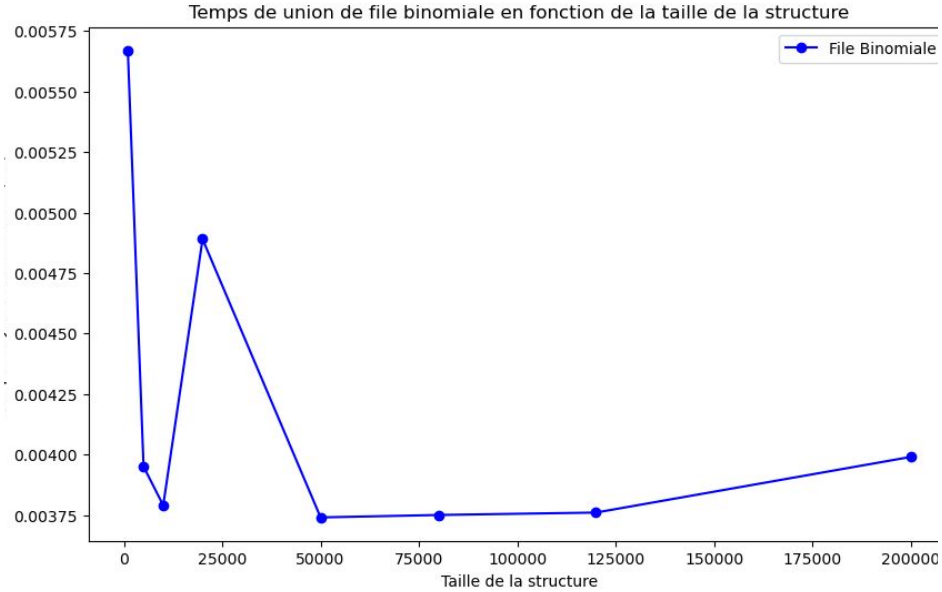
SupprMin: on se base aussi sur l'union, car on va supprimer un élément et on retourne la file sans cet tournois.

Représentation Graphique de la complexité

Construction



Union



Hachage MD5

Présentation:

MD5 est une méthode de hachage cryptographique qui permet de chiffrer un message sur code unique et de taille réduite (128 bits) tout en minimisant les collisions.

MD5 calcule l'empreinte numérique d'un fichier et opère de la manière suivante:

Principe du hachage MD5

Prétraitement :

Ajouter un padding au message original pour avoir une longueur = 512*k - 64
Un bloc final de 64 bits est ajouté pour stocker la longueur originale du message avant le padding.

-> On se retrouve avec un message de longueur multiple de 512

Initialisation

Initialiser 4 registres A,B,C,D de 32 bits chacun avec des constantes spécifiques.

Traitement:

Chaque bloc de 512 bits est traité à travers une série de 64 étapes itératives, en utilisant des fonctions non linéaires, des opérations de décalage, de "ou exclusif", et des additions modulo 2^32 pour mélanger les bits du message.

Les variables A, B, C, et D sont mises à jour à chaque étape du traitement du bloc en fonction des résultats intermédiaires.

Finalisation

La valeur de hachage finale est obtenue en concaténant les valeurs finales des variables A, B, C, et D.

Arbre de recherche

L’arbre AVL est adéquat car :

Il Garantit que pour chaque sous-arbre, la différence de profondeur de ses fils est au plus de 1.

-> Profondeur maximale d’une feuille = hauteur de l’arbre = log n

-> Recherche d’une valeur dans l’arbre en O(log n) comparaisons

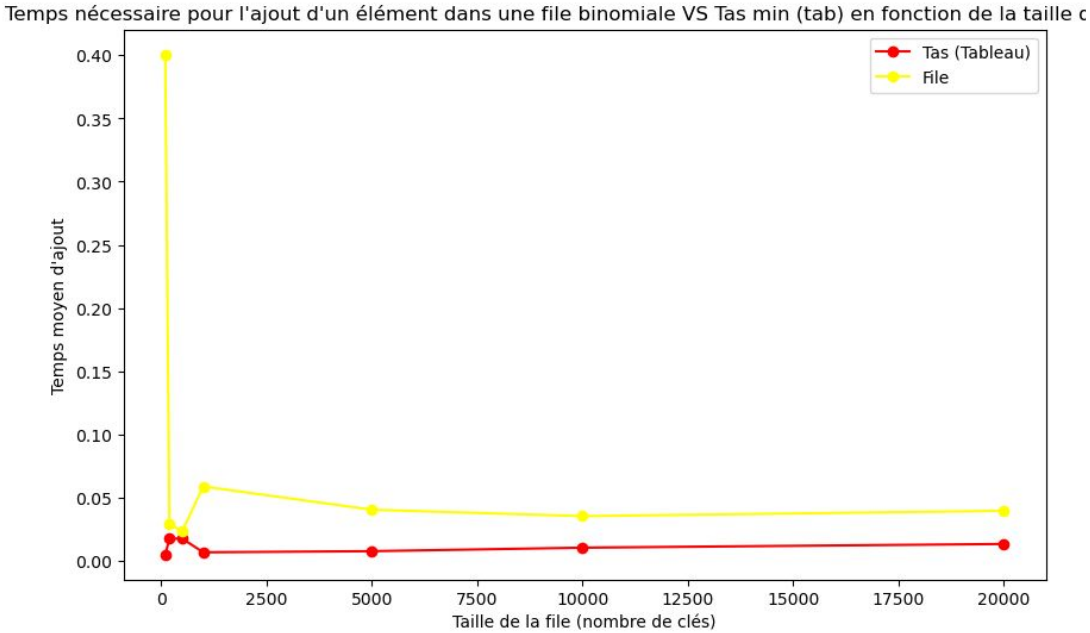
Etude expérimentales

Données de l’oeuvre
de Shakespeare

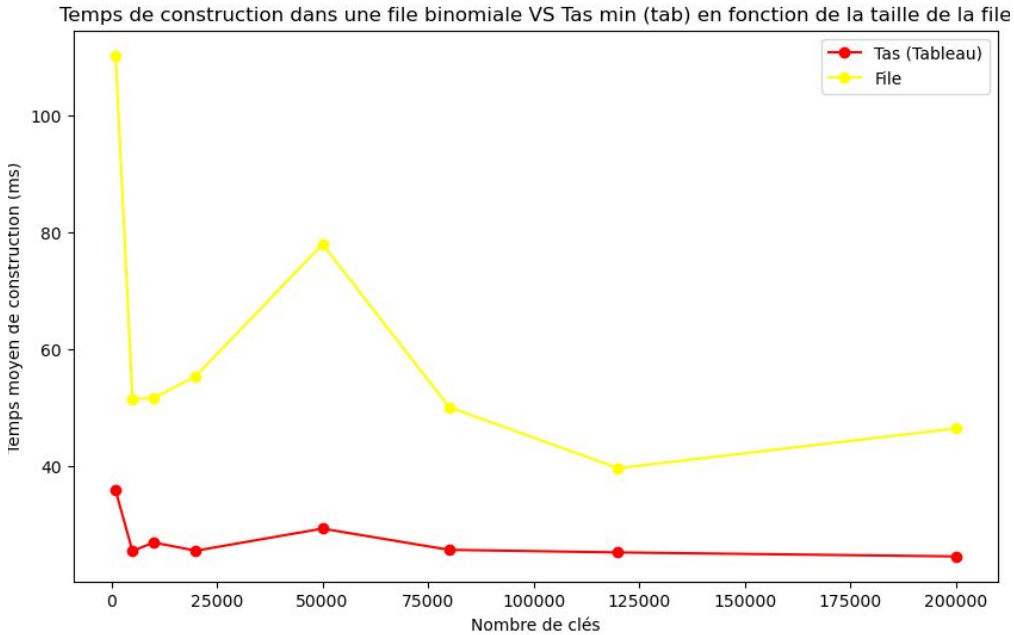
Collisions

Après avoir utiliser le Hachage MD5 sur les données de Shakespear, on remarque qu'il y a pas de collision dans les mots, car la probabilité de trouver 2 mots en collision en utilisant le hachage MD5 est $1.47*10^{-29}$

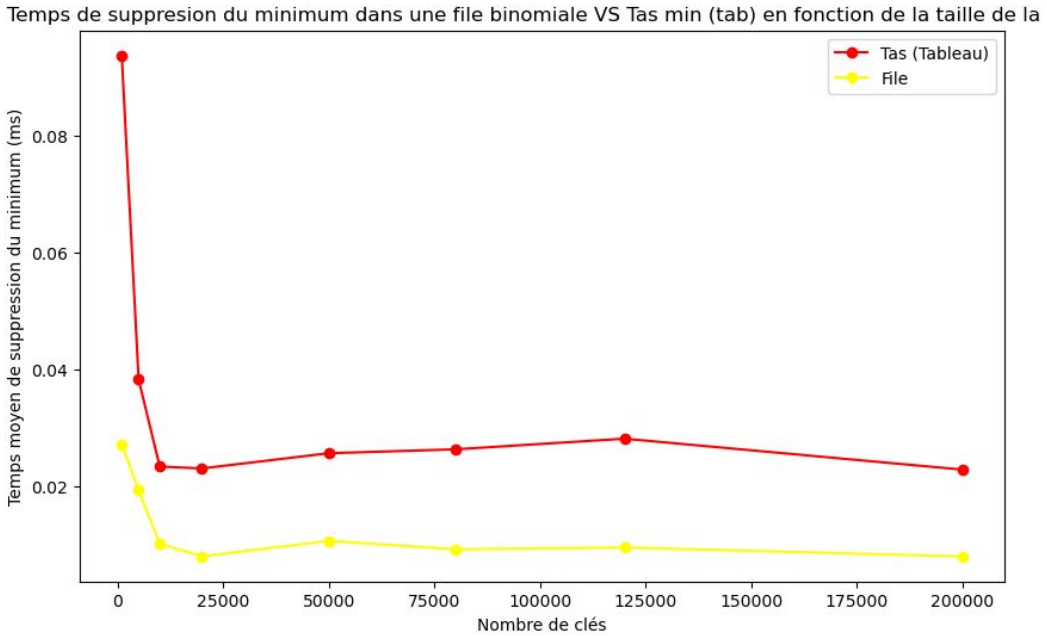
Evaluation de l'ajout



Evaluation de la Construction



Evaluation de la suppression du minimum



Evaluation de l'Union