Mathématiques discrètes
Tri fusion

1 Principe du tri

Il s'agit de trier une liste chainée de valeur entière. Pour cela, le tri fusion découpe la liste en deux sous listes les trie séparément puis les fusionne. Le trie de chaque sous liste reffet appel au trie fusion jusqu'à n'obtenir que des valeurs singulières.

L'algorithme se base sur l'idée que les comparaisons sont couteuses en temps et qu'il faut donc les minimiser en n'ayant à trier que des sous-listes qui sont déjà triées.

Cet algorithme reprend le principe du "diviser pour mieux règner".

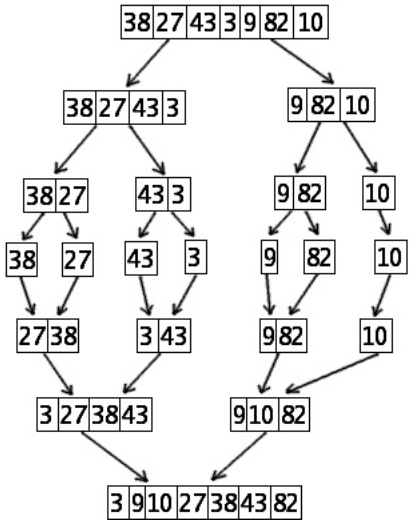

FIGURE 1 – Principe du tri fusion

2 Protocole de validation

L'avantage des fonctions récursives est qu'elles sont courtes à écrire et donc qu'un bug ne peut se trouver que dans un nombre limité de lignes de code. Un problème survient cependant lorsque l'on souhaite suivre l'état de ces fonctions.

Avec une fonction non récursive, il est facile de mettre des points d'arrêts dans le programme pour suivre l'état des variables. Dans un programme récursif

les valeurs sont stockées sur la pile et il est difficile de suivre les étapes intermédiaires.

Pour vérifier le bon fonctionnement de nos fonctions, nous avons fait des tests itératifs. Nous avons analysé le rôle de chaque fonction et vérifié ensuite indépendamment qu'elles produisent le résultat attendu avant de réaliser la prochaine fonction.

Une fois l'algorithme entièrement écrit, nous avons testé son fonctionnement avec plusieurs jeux de tests qui nous semblaient pertinents. Nous avons testé l'algorithme sur des listes déjà triées, non-triées dans le pire des cas. Nous avons fait aussi varier le nombre d'éléments de la liste pour vérifier les cas limites. Une liste avec un seul élément, deux éléments. Enfin nous avons testé l'algorithme sur des jeux de données aléatoires.

De façon pragmatique, nous avons fait un parcours séquentiel complet de la liste résultante pour vérifier que chaque élément est plus petit que l'élément suivant.

3 Complexité théorique

Le cout de la division d'une liste est de l'ordre de n, celui de la fusion de liste de respectivement n et m éléments est au pire de $(n + m)$ opérations. Ces deux opérations sont effectuées un même nombre de fois en effet on parle de diviser ou de multiplier par deux la taille d'une liste pour revenir à un liste de la même taille qu'initialement.

On approxime ce nombre à $ln(n)$ fois, en effet on découpe à chaque fois une liste en deux jusqu'à arriver à des éléments de taille un, et $2^(log(n)) = n$ morceaux de taille 1 (ou log est le logarithme néperien en base 2)

L'algorithme est donc en $O(n\ ln(n))$. Cependant, cette méthode utilise des appels de fonctions de manière récursive. Cela a pour conséquence que chaque appel de fonction demande la sauvegarde du contexte d'execution, ça prend du temps et de la mémoire.

4 Protocole de mesures

Pour mesurer de façon pertinante le temps d'execution de l'algorithme, nous devons mesurer uniquement l'appel de fonction au tri fusion. De façon naïve nous aurions pu mesurer le temps d'execution de l'appel du programme grâce logiciel "time" disponible sous linux, mais nous aurions alors aussi prit en compte le temps d'allocation et de liberation de mémoire. Si ce temps peut être considéré comme négligeable pour un nombre d'éléments faible, il devient pratiquement équivalent au temps du trie pour de grands ensembles de valeurs.

Nous avons donc choisi de mesurer le temps d'execution de notre algorithme directement dans le corps du programme grâce à la librairie "time.h". On prend deux mesures, une avant d'appeler la fonction triFusion et une autre une fois

l'appel terminé. On fait la soustraction de la dernière mesure avec la première pour obtenir le temps mis par notre algorithme pour s'executer.

5 Étude des cas

5.1 Cas d'une liste déjà triée

Une liste triée représente le cas optimal pour la fonction de tri. En effet elle effectue seulement les opérations de division et de fusion sans avoir à déplacer les éléments.

5.2 Cas d'une liste triée en sens inverse

La liste triée en sens inverse est aussi un cas très rapide même si cela est moins évident au premier abord. Tous les éléments d'une sous liste seront plus grand que tous les éléments de l'autre. Le programme fusionnera les deux listes en économisant des comparaisons.

Exemple : $[5]->[4]->[3]->[2]$ est décomposé en [5] — [4] — [3] — [2] puis les deux listes $[4]->[5]$ et $[2]->[3]$ sont fusionnés.

Le programme va comparer 4 à 2 (ajouter 2) puis 4 à 3 (ajouter 3) et ajouter d'un coup la liste restante. Nous avons donc 2 tests 3 insertions.

Dans le cas du jeu de données suivant : $[5]->[3]->[2]->[4]$ pour les sous listes $[3]->[5]$ et $[2]->[4]$. Le programme compare 2 et 3. Ajoute 2. Compare 3 et 4, ajoute 4. Compare 4 et 5, ajoute 4 puis 5. Soit 3 tests et 4 insertions.

Pour fusionner deux listes de n éléments triées en ordre inverse on aura donc une complexité de l'ordre de n et non pas de $2n$ éléments.

5.3 Cas d'une liste de valeurs aléatoire

Enfin, le jeu aléatoire représente le cas moyen, mais aussi le pire cas de ces jeux d'essai.

6 Résultats des mesures

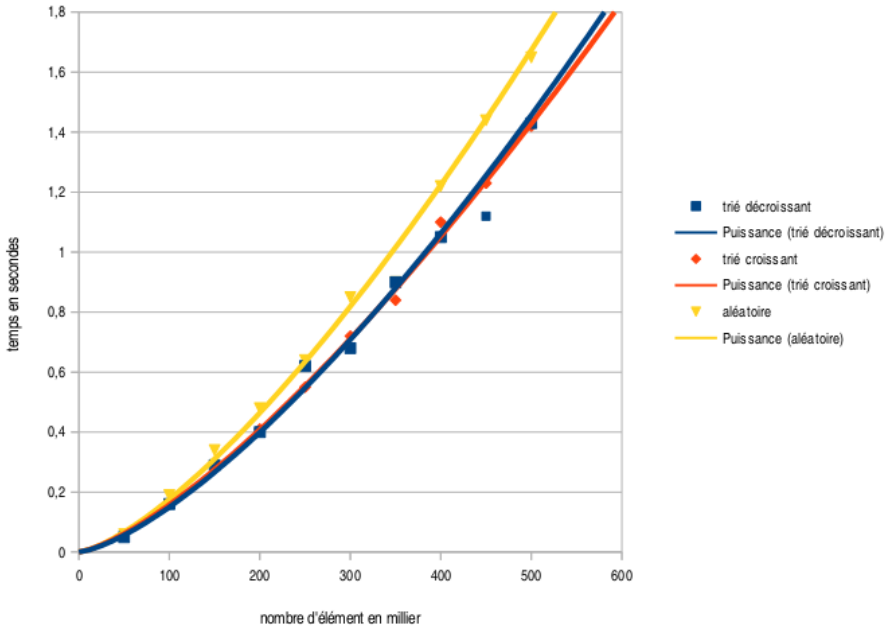

FIGURE 2 – Résultat de performance du tri fusion

Première remarque, les appels récursifs de la fonction et le nombre limité d'appel de la pile ne nous permet pas de trier des piles de plus de 600 000 éléments. Ensuite sous la barre des 50 000 éléments le temps est de l'ordre de cinq centième de seconde, on choisira donc de ne pas étudier des tris inférieurs à 500 000 éléments pour des raisons de précision.

Les courbes de tendances obtenues sont bien logarithmiques. Jusqu'aux 200 000 éléments à trier la fonction de tri est aussi performante quelque soit le jeu d'essai. Cette information est intéressante car d'autres algorithmes ont de grands écarts de performance entre le meilleur et le pire des cas. Le tri fusion nous permet une bonne approche du temps de traitement quelque soit la liste à trier.

Au delà des 300 000 éléments, les courbes de nombres aléatoires et de ceux de listes déjà trier commencent à se distinguer. Même si la complexité est toujours de $O(n\ ln(n))$, la constante varie d'un jeu à l'autre et c'est celle ci qui explique la création de cet écart. De même le tri des listes déjà triées par ordre croissant et décroissant ont une constante différente, cependant les tests montrent que celle ci est insignifiante.

De plus, plus les listes choisies sont grandes plus l'influence du ln est faible. On s'approche alors d'une complexité en $O(n)$.

7 Conclusion

L'algorithme de tri fusion est parmi les tris les plus performants, surtout en comparaison à d'autres algorithmes qui sont eux en moyenne en $O(n^2)$ pour les moins rapides, et en $O(n\ log(n))$ pour les plus rapides.

Le plus grand défaut de cet algorithme est qu'il utilise la mémoire de la pile qui a une taille limité. C'est ce qui nous empêche de tester ses résultats sur des listes de plus de 600 000 éléments, or cet algorithme est sensé se démarquer par ses performance sur de grands ensembles. Pour palier à ce problème de mémoire il est possible d'utiliser une méthode itérative du tri fusion, mais le trie n'est alors plus en place, il faut une autre liste pour contenir les valeurs triées.