Rapport Algorithmique 2

Génération de textes aléatoires

L'objet de chacun des modules

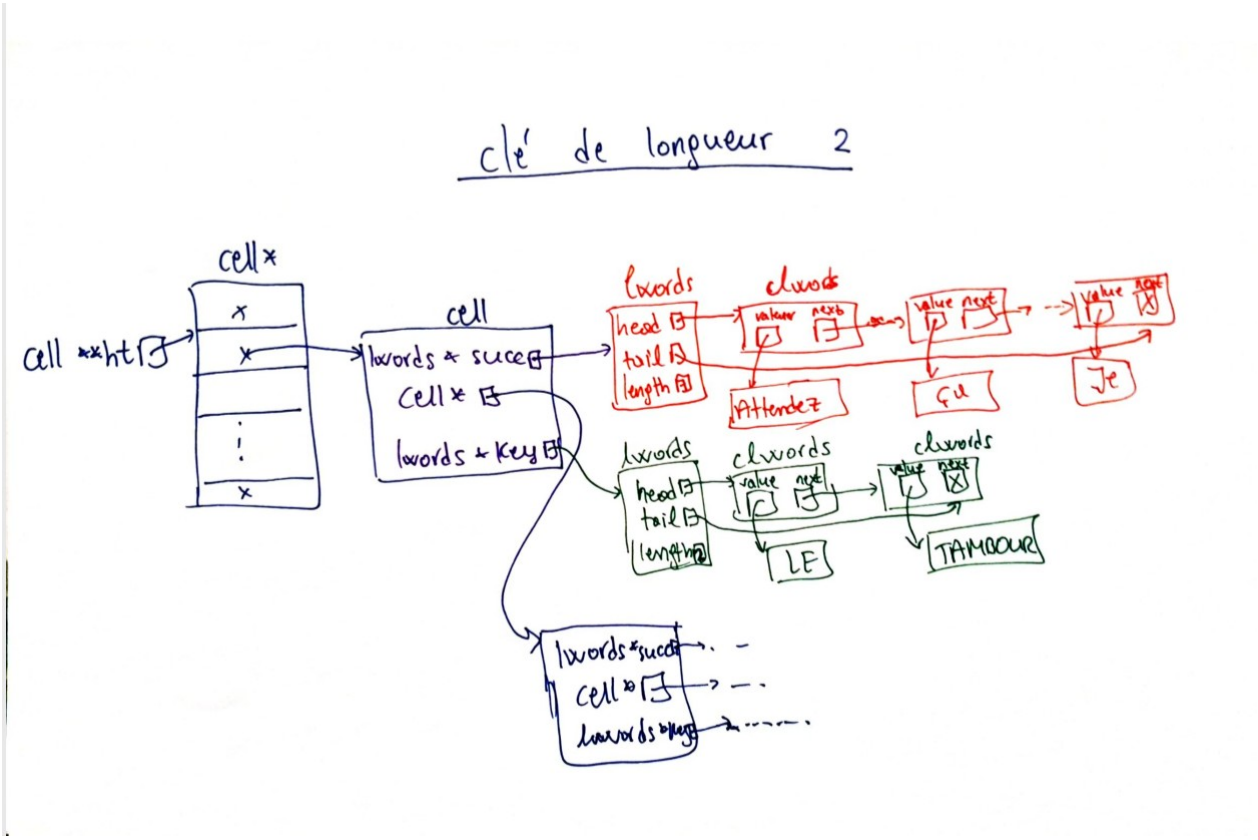
1. hashtable, holdall → modules déjà données en CM et aussi en TP, nous les utilisons sans les modifier pour stoker les clés et les successeurs associés.
2. lwords → module utilisé pour l'implatation des différentes fonctions sur les listes dynamiques simplément chaîné avec pointeur de tête et de queue et un champ représentant la longueur de la liste.
3. main → fichier contenant le programme principale où nous avons fait l'implatation de l'algorithme de génération de textes aléatoires et qui utilise tout les autres modules.

Déscription de l'implantation

Dans le fichier main.c ce trouve toute l'implatation de l'algorithme demandé.
Nous avons utilisé un type énumére pour les différentes options que l'utilisateur peut entrer sur la ligne de commande. L'utilisateur doit rentrer soit le nom du programme suivi des options -nNUMBER -lNUMBER , soit le nom du programme suivi des options -nNUMBER --display-table ainsi que le texte lu sur l'entrée. L'utilisateur peut toujours entrer le nom du programme suivi de --help s'il veut de l'aide pour l'execution de main.c.
Pour garder la valeur de NUMBER nous avons utilisé la fonction de la biliotheque <string.h> appelé *atoi* qui converti un string en un int et nous avons fait cela à partir de la deuxiéme indice du argv[k] entré sur la ligne de commande pour chaque option car nous savons que les deux premiers indice de argv[k] correspondent aux caratères '-' et 'n'.

Après nous avons codé la phase de lecture et de construction de la table de l'algorithme demandé et aussi la phase de la génération aléatoire.
Nous avons pensé d'utiliser des listes dynamiques simplément chaîné avec pointeur de tête et de queue et aussi un champ qui représente la loungeur de la liste pour garder toutes les clés et les listes de successeurs associées à chaque clé. Ainsi, beaucoup de nos fonctions sur ce type de liste s'éxécutent en temps constant. Nous utilisons un tableau de hachage initialisé à la table de hachage vide avec l'aide de la fonction *hashtable_empty* qui prend en paramètres la fonction de comparaison pour les listes défini dans le module *lwords.h* et la fonction de pré-hachage conseillées par Kernighan et Pike traduite pour les listes. Nous utilisons trois fourretouts, un pour garder les listes de clés, un pour garder les listes des succésseurs et un autre pour garder tous les mots lues sur l'entrée standard. Tous les mots que nous lisons sont stockés dans le tableau string[STRING_LENGTH_MAX + 1]. Nous avons suivi l'algorithme expliqué dans le sujet et après avoir bien compris le TP8 ainsi que ce que nous avons fait en CM sur les tables de hachage, nous sommes arrivés à la conclusion que le principe d'associer à une clé sa liste de succésseurs, est pareil avec ce que nous avons fait en tp en lisant des mots et à chaque mot lui associer son nombre d'occurence. Mais dans notre cas, nous n'avons pas des mots mais des listes de mots. Donc nous avons compris qu'il suffisait de faire 2 fonctions, une pour la comparasion des listes (tp8 comparasion des mots avec strcmp) et nous avons changé la fonction local *str_display* à *key_succ_display* qui, au lieu de calculer le h pour un mot, il calcule le h pour tous les mots qui sont dans une liste. De même nous avons modifié les fonctions *rfree*, nommée maintenant *rfree__word*, qui fait la désallocation d'un mot et *rfree__list* qui fait la désallocation d'une liste. Ces dernières

fontions sont utilisées pour faire l'affichage de la table clé-successeurs (*holdall_apply_context*) quand l'utilisateur entre les options -nNUMBER - - display-table. Elles font aussi les désallocations des fourtouts car dans les fourretouts hakey et hasucc nous n'avons plus des mots mais des listes de mots. Le dessin illustre ce que nous avons expliqué jusqu'à maintenant.



Exemples

Sur la ligne de commande :

- ➔ time valgrind ./main -n2 < ../textes/abcd.txt --display-table
  allocation / desallocation : 182
  temps d'execution : 0m0,460s
- ➔ time valgrind ./main -n2 -l100 < ../textes/lesmiserables.txt
  allocation : 4,538,398
  temps d'execution : 0m12,346s

Le dernier fait beaucoup d'allocations car nous allouons pour les listes de clés et les listes de successeurs et nous mettons chaqu'une des listes dans un fourretout donc nous allouons pour les fourretouts, pour les mots et pour le fourretout des mots et finalement nous allouons pour la table d'hachage. Le temps d'éxécution est un peu plus grand car le fichier est de taille plus grande de plus, au début, nous affichons les messages pour les mots qui vont être découpés et donc le temps des testes incrémente le temps d'éxécution.

Limites éventuelles du programme

Nous ne lisont pas la germe du générateur pseudo-aléatoire sur l'entrée standard car nous voulions avoir l'option --display-table avec -nNUMBER mais commme il ne peux lire que deux arguments sur la ligne de commande, il ignorait le reste après avoir lu ses deux premiers. Aussi nous avons fait des tests mais il y avait toujours un problème quand nous avons les trois options donc nous avons décidé de forcer l'utilisateur à entrer un nombre défini d'arguments sur la ligne de commande.
De plus, nous avons un problème avec les désallocations. Quand il affiche la table toutes les désallocations sont faitent correctement mais quand nous voulons qu'il affiche la génération du texte aléatoire, il ne fait pas correctement les désalloations. Donc nous avons un problème avec les disposes dans la parite de gnération aléatoire de texte mais nous n'avons pas réussi à le résoudre.