TP Algorithmique et Optimisation Discrète

PORA - LEDUCQ

October 2018

1 Programme

Question 5 :

Le programme contient un seul fichier .c. On a définit une structure paragraphe qui nous à permise pas la suite de créer un tableau de chaîne de caractère contenant tous les mots du paragraphe qu'on est en train de traiter. On a utilisé des variables globales pour définir des conditions d'arrêt notamment pour séparer le texte d'entrée en paragraphe et pour arrêter le programme lorsqu'on arrive à la fin du fichier c'est la variable IND (avec mmap on ne peut faire une boucle while qui s'arrête à EOF mais comparer la taille du fichier et l'indice globale de parcours du fichier).
Par la suite, pour chaque paragraphe, on crée un tableau contenant donc les mots du paragraphes, un tableau de tous les cout minimaux par chaque sous-paragraphes commençant par le mot i pour tout i indice du tableau des mots du paragraphes. Les coûts min sont calculés avec mémoïsation.
De plus, on crée un autre tableau qui stocke le tableau des pointeur parent (tableau argmin) de i pour i un indice dans le tableau de mot du paragraphe. Les coût minimaux sont calculé avec la fonction coutminimal1 (c'est par ailleurs aussi cette fonction qui remplit le tableau des pointeur parent argmin).
Une fois cela fait on connait précisément par quel mot devrait commencer chaque ligne du paragraphe justifié. Ainsi on peut utiliser la fonction justification() qui permet d'écrire dans le fichier de sortie des lignes justifiées avec le bon nombre(M) de caractères. On libère tous les tableaux créés et le programme écrit avec succès dans la sortie stderr.

Determinons l'espace mémoire nécessaire à notre programme afin de justifier un texte de $\#caractères$ caractères : nous utilisons une première structure de liste chainée afin de stocker l'ensemble des mots d'un paragraphe chaque mot est stocké sur un espace mémoire de taille M. On donc $\#caractères$ cellules de listes chainées ayant chacun un mot de taille M.
Ensuite nous avons un tableau de mot de taille $\#caractères$ qui référence les mots de la liste chainée précédente.
Enfin pour la justification afin de créer la ligne justifié on la stocke on utilise un

espace n supplémentaire.
On a donc besoin d'un espace memoire de taille $O(\#caractères)$.

Déterminons le nombre d'opérations effectués par notre programme dans le pire cas. On se place toujours dans le cas de la justification d'un texte de n mots sur une longueur M.
Pour récupérer chaque mot nous utilisons une boucle while nous avons donc n opérations qui correspondent à la condition du while. pour recupérer chacun des caractères des mots nous effectuons des comparaisons afin de savoir si on rencontre des separateurs de mots ce qui ajoute $\#caractères$ opérations.
Pour ce qui concerne du calcul de coût minimal, on utilise deux fonctions : *coutminimal*1 et *badness*. Le calcul de coût minimal se fait par mémoïsation. Pour chaque mot indicé i du tableau, on appelle la fonction *coutminimal*1. Celle-ci utilise le tableau tab coutmin qu'elle a en argument pour calculé le coût minimal du sous paragraphe suffixe commençant par le mot i en utilisant les coût minimauux calculés pour les sous paragraphe suffixe commençant par des mots se trouvant après le mot indicé i (on fait une recherche de minimumu en utilisant la fonction d'évalutation *badness*()). On appelle n fois la fonction *coutminimal*1. Celle si boucle sur les mots se situant, dans le texte, après le mot d'entrée. On a donc un coût pour cette partie du programme en : $O(n * \sum_{i=0}^{n} i) = O(n^3)$.
Le calcul du coût en mémoire demande de la place en mémoire supplémentaire, en effet on stocke deux tableaux de taille n contenant des entiers. Ceux sont les tableaux tab argmin (le tableau contenant les pointeurs parents) et le tableau tab coutmin contenant les coût minimaux des sous paragraphes suffixes commençant par tous les mots du texte.

Remarque : c.f. Annexe pour le détail du raisonnement du calcul du coût minimal.

Pour la justification du texte nous avons n opérations ce qui correspond à une condition de boucle while et au parcours des mots. Il y a également $\#caractères$ pour placer chacun des caractères et ajouter les espaces sans dépasser M caractères. Finalement nous avons $O(n^3)$ opérations.

Déterminons les défauts de localité produits par notre cache. Nous avons utilisés *mmap* donc le texte dans notre programme a le même coût d'accès que si il s'agissait d'un tableau de caractères. Tout d'abord nous récupérons les mots du texte en procédant une lecture caractères par caractères on a donc $O(\frac{\#caractères}{L})$ défauts de caches.
Nous recopions ensuite chacun des mots dans un tableau de mots ce qui nous fait $O(\frac{n}{l})$ défauts de localité.
Pour la partie sur le calcul de coût minimum : on utilise trois tableaux qu'on appelle en même temps. Pour le tableau de mot, on peut compter : $O(\frac{n^3}{L^2}$

défauts de localité. Pour les tableaux de coût min et d'argmin : on les appelle n : on peut compter $O(\frac{n}{L})$ pour les deux. Au total, le calcul de coût minimal exige $O(\frac{n^3}{L^2})$ défauts de localité.
Enfin pour la j de l'appeL'utilisation l de la fonction. ustification de texte nous parcourons une nouvelle fois l'ensemble des caractères ce qui correspond a $O(\frac{\#caractères}{L})$ défauts de caches.
Finalement nous avons $O(\frac{n^3}{L^2})$ défauts de localité.

Question 6 :
Tableau des temps d'execution pour M = 100:

Texte	Minimum en s	Moyen en s	Maximum en s
court-plusieurs-paragraphes-ISO-8859-1.in	0.04	0.06	0.08
court-un-paragraphe.in	0.04	0.04	0.05
court-un-paragraphe-8859-1.in	0.03	0.04	0.06
foo.in	0.03	0.03	0.03
foo.iso8859-1.in	0.03	0.03	0.03
foo2.iso8859-1.in	0.02	0.03	0.03
longtempsjemesuis.ISO-8859.in	0.04	0.07	0.08

Comparons avec le nombre de mots et de caratères dans les textes

Texte	Nombre de caractères	Nombres de mots
court-plusieurs-paragraphes-ISO-8859-1.in	3014	527
court-un-paragraphe.in	633	117
court-un-paragraphe-8859-1.in	633	117
foo.in	51	25
foo.iso8859-1.in	81	36
foo2.iso8859-1.in	164	72
longtempsjemesuis.ISO-8859.in	3681	622

On peut remarquer une correlation entre le temps d'éxécution et la longueur des textes. Il est difficile de donner la dépendance entre ces deux tableaux car l'échantillon est petit et de plus le temps d'éxécution est assez peu précis.

Analysons le nombre de defauts de caches :

Texte	Nombre de défauts de caches
court-plusieurs-paragraphes-ISO-8859-1.in	55 000
court-un-paragraphe.in	27 000
court-un-paragraphe-8859-1.in	27 000
foo.in	7 000
foo.iso8859-1.in	7 000
foo2.iso8859-1.in	8 000
longtempsjemesuis.ISO-8859.in	4 423 000

Les premiers textes semblent donner des résultats coherrents mais pas le dernier sans qu'on puisse l'expliquer.

2 Annexe

Dans cette annexe, nous allons détailler le raisonnement mis en place pour le calcul du coût minimal et l'utilisation des pointeurs parents. Remarque : les indices font références à la position des mots dans le tableau. On commence avec la fonction $badness(i, j)$ qui nous dit combien ça serait "mal" d'utiliser les mots de i à j comme une ligne. C'est une fonction de pénalité.
On a :
Maintenant on découpe le paragraphe en sous paragraphe suffixe. Il y a n sous paragraphe suffixe. On note $DP(i)$ le coût minimal pour le sous paragraphe suffixe commençant par le mot indicé i.
On a alors : $DP(i) = min(DP(j) + badness(i, j)$ pour j allant de $i+1$ à $n-1)$.
Avec pour condition d'arrêt : $DP(n-1) = 0$.
De plus, on note :
$parent(i) = argmin(DP(j) + badness(i, j)$ pour j allant de $i + 1$ à $n - 1)$.
Ainsi, on sait que la première ligne commence au mot indicé 0. La deuxième ligne doit commencer par le mot indicé $parent(0)$, la troisième commence au mot indicé $parent(parent(0))$ ...