Rapport de TP 4MMAOD : Justification optimale de texte

BONNECHERE Jean-Baptiste (ISI G_1)
PRETRE–HECKENROTH Raphaël (IF G_1)

11 novembre 2018

1 Principe de notre programme (1 point)

Notre programme utilise la méthode récursive, avec mémoïsation, et applique l'équation de Bellman associée aux mots du texte source. Nous avons choisi de stocker, pour chaque mot k, le coût minimal pour la suite du paragraphe dans le cas où k est en début de ligne. On mémorise également la liste des indices telle que si le mot k est en début de ligne dans le paragraphe n, alors `indices[n][k]` est le rang du mot qui commencera la ligne suivante avec le coût moindre pour la suite du paragraphe. La lecture des caractères du fichier s'effectue à l'aide de la commande `mmap`. L'écriture dans le fichier sortie s'effectue via la commande `fprintf`.

2 Analyse du coût théorique (1.5 points)

2.1 Nombre d'opérations (comparaions/min) en pire cas :

Justification : Pour chaque paragraphe p, pour chaque mot m_0 du paragraphe, le programme boucle sur les mots m_1 qui peuvent être sur la même ligne que m_0 et effectue au plus une comparaison par mot m_1. Chaque ligne comporte au maximum $\frac{L}{2}$ mots. Donc le nombre de comparaisons par rapport au min est de l'ordre de $\frac{mL}{2}$ et est donc en $\Theta(m)$.

2.2 Place mémoire requise :

Justification : Tout d'abord, on mémorise tous les mots (i.e. tous les caractères plus les caractères de fin de chaine) du texte source. Pour chaque mot mémorisé on associe un coût (`long long`) pour le reste du paragraphe et un indice (`long long`) pour le « meilleur »mot qui commencerait la ligne suivante. Le programme a donc besoin de $\Theta(8(n+m)+2\times 64n)$ bits de mémoire.

2.3 Nombre de défauts de cache sur le modèle CO :

Justification : Lors de la lecture du fichier texte, on effectue $\frac{n}{Z}$ défauts de cache. De la même manière, lors de l'écriture du fichier, on effectue $\frac{n}{Z}$ défauts de cache. De plus, on calcule le nombre de caractères pour un ensemble de mots (de cardinal < $\frac{L}{2}$) pour chaque ligne. Ce qui fait donc pour chaque ligne $\frac{L^2}{2}$ défauts de cache. Il y a au plus $\frac{2m}{L}$ lignes. On obtient donc un nombre de défauts de cache total de : $\frac{2n}{Z}+mL$ (pas en accord avec les tests effectués par la suite).

3 Compte rendu d'expérimentation (2.5 points)

3.1 Description de la machine et conditions expérimentalesi (0.5 point)

Les tests ont été lancés depuis `pcserveur.ensimag.fr` à l'aide de la commande `time`. Afin d'éviter que la mémoire en cache ne fausse les temps d'exécution, nous avons choisi d'exécuter les tests à l'aide de ce script shell :

```
#!/bin/bash
for ((i=1; i<=5; i++))
do
  for ((j=200; j<=2000; j=j+200))
    do
      echo ""
      echo $j
      time ./bin/AODjustify $j ./Benchmark/ALaRechercheDuTempsPerdu-1paragraphe-debut
    done
done
```

3.2 Mesures expérimentales (1 point)

Nous avons choisi d'effectuer les tests sur le premier tome d'*A la recherche du temps perdu* (avec un unique paragraphe) afin de réduire les temps d'exécution qui s'avéraient très longs lorsque M était grand pour la totalité du fichier de base.

longueur ligne (M)	valeur de justification	temps elapsed min	temps elapsed max	temps elapsed moyen	temps user moyen	temps system moyen
200	321541	1.271	1.345	1.3064	1.1744	0.014
400	161337	3.349	3.432	3.3908	3.2466	0.0126
600	108974	6.362	6.442	6.4106	6.2952	0.0148
800	79626	10.35	10.506	10.435	10.3042	0.0164
1000	66626	15.34	15.486	15.4026	15.2866	0.0136
1200	59227	21.241	21.59	21.3768	21.2478	0.0148
1400	45523	28	28.284	28.2096	28.13	0.0144
1600	44216	36.081	36.616	36.2694	36.1046	0.0138
1800	37260	44.878	45.335	45.0612	44.866	0.0126
2000	29021	54.579	54.876	54.676	54.5234	0.0154

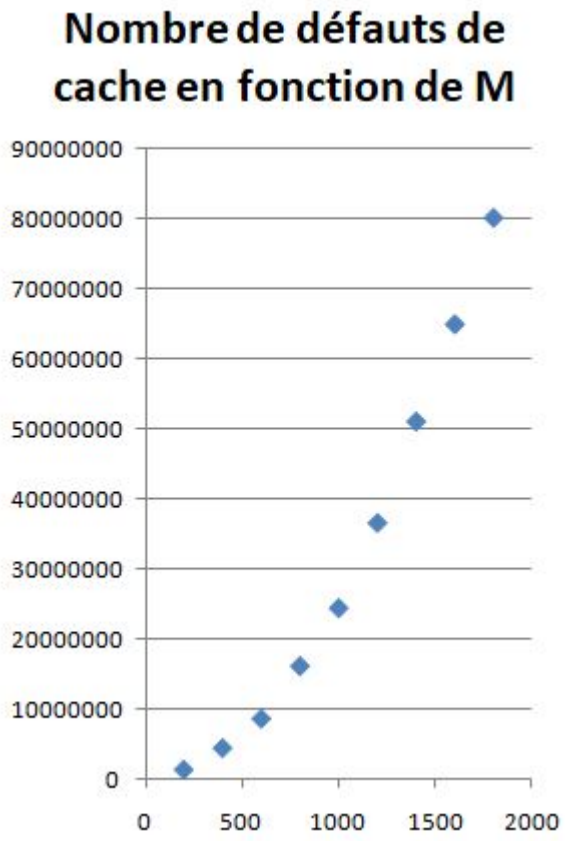
FIGURE 1 – Mesures des temps minimum, maximum et moyen de 5 exécutions.

longueur ligne (M)	#instructions (travail)	#défauts au total	#défauts en lecture	#défauts en écriture
200	1 229 878 451	1 247 462	1 075 025	172 437
400	3 269 528 073	4 350 354	4 178 061	172 293
600	6 264 371 434	8 55 089	8 382 562	172 527
800	10 208 438 518	16 069 214	15 896 493	172 721
1000	15 100 998 481	24 338 319	24 157 286	181 033
1200	20 940 603 281	36 506 405	36 319 864	333 407
1400	27 727 114 125	50 987 485	50 795 602	191 883
1600	35 460 243 204	64 875 371	64 659 765	215 606
1800	44 142 797 463	80 081 314	79 770 284	311 030

FIGURE 2 – Mesures des défauts de cache avec `valgrind -tool=cachegrind`.

3.3 Analyse des résultats expérimentaux (1 point)

Graphiquement, le nombre de cache semble suivre une évolution en $O(n^3)$. Cela remet donc bien en question le calcul effectué plus tôt.



Conclusion

L'algorithme implémenté renvoie le résultat attendu. Néanmoins le temps d'exécution semble trop élevé, notamment lorsque M est très grand. Il y a sûrement des améliorations à faire concernant le calcul du nombre de caractères pour un ensemble de mots (qui est parfois effectué plusieurs fois pour un même ensemble de caractères). De plus, l'écriture dans le fichier de sortie s'effectue sans utiliser la fonction `mmap`, ce qui peut provoquer un surplus d'opérations.