Rapport de TP 4MMAOD : Patch optimal entre deux fichiers

BOYER Quentin (groupe étudiant$_1$)

7 novembre 2019

1 Graphe de dépendances (1 point)

Dessinez le graphe de dépendances des appels.

L'algorithme est alors de trouver le plus court chemin depuis un point $(i, j) tq i = 0 et/ou j = 0$. Cela donne donc en fonction du chemin le patch optimal

2 Principe de notre programme (1 point)

Pour pouvoir trouver le plus court chemin je commence par creer une liste de tout les chemins au depart d'un des bords avec $i = 0$ ou $j = 0$. Ensuite on fait un tas avec ces chemins. A chaque iteration suivante on prends le chemin le plus court, on l'enleve du tas. Grace a ce chemin on produit trois chemin en incrementant i, j puis i et j. Si un chemin $c1$ arrive sur une case sur laquelle un autre chemin est deja passe, on sait par notre invariant qu'il existe un autre moyen d'arriver a ce point, et qu'il est plus court. On peut donc supprimer $c1$, puisque il sera pire que le chemin qui est deja passe par la. Sinon on ajoute le chemin dans notre tas

On s'arrete des que le plus court chemin arrive sur la case (m, n). On a alors trouve le plus court chemin pour arriver au patch optimal

De plus pour calculer les couts on stock non pas un tableau des lignes mais un tableau qui donne l'indice de la fin de la ligne i, cela permet ainsi d'avoir un cout faible pour trouver la longeure d'une ligne

3 Analyse du coût théorique (3 points)

On commence par calculer la liste des indices de fin de ligne, ca coute $m + n$ dans tout les cas.

3.1 Nombre d'opérations en pire cas :

Justification : Dans le pire cas il faut calculer tout les chemins possibles, c'est a dire passer par chaque case exactement une fois, cela fait donc $m \times n$ iterations. A chaque iteration on produit environ 1 chemin, puisque si rentre en collision avec les autres chemins, au total on est donc en $\Theta(m \times n)$

3.2 Place mémoire requise :

Justification : Dans chaque case on doit stocker de quelle cases viens t'on, pour pouvoir refaire le chemin dans le sens inverse. Cela fait donc dans le pire cas autant de place requise que de chemins, soit $\Theta(m \times n)$.

3.3 Nombre de défauts de cache sur le modèle CO :

Justification : Pour les defauts de cache, dans le pire cas a chaque fois qu'on essaye d'acceder a une case elle n'est pas dans le cache, si par exemple les chemins les plus courts sont dans des parties totalement differentes de la grille. Cela coute donc $\Theta(m \times n)$

3.4 Quand arrive le pire cas

Si les fichiers ne sont pas tres differents on va rester majoritairement autour de la diagonale. Cela change donc beacoup la donne, puisque on va faire des defauts de cache que lorsque on a trop avance dans la diagonale, on va calculer et garder en memoire qu'environ la diagonale.

Si $m \approx n$ on a la diagonale de longeure m, cela donne donc un algorithme quasiment lineaire en la taille des fichiers, et m/L defauts de cache environ

4 Compte rendu d'expérimentation (2 points)

4.1 Conditions expérimentaless

4.1.1 Description synthétique de la machine :

Le CPU de la machine est un AMD Ryzen 2600, avec 16GB de RAM DDR4. Il tourne sur Arch Linux, i3. Utilisation standard du PC avec des logiciels usuels d'ouvert comme un navigateur web.

4.1.2 Méthode utilisée pour les mesures de temps :

J'ai utilise le benchTester pour relever les temps de chaque benchmark, user puisque c'est un programme mono-thread et que cela enleve le temps de l'OS. l'un apres l'autre en attendant. J'ai recommence 5 fois.

4.2 Mesures expérimentales

	coût du patch	temps min	temps max	temps moyen
benchmark_00	10419	0.01	0.03	0.02
benchmark_01	13222	5.00	5.99	5.27
benchmark_02	3633	3.00	3.40	3.09
benchmark_03	19121	6.56	7.15	6.73
benchmark_04	3726	3.13	3.32	3.22

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

On voit en annexe un graphe de temps de calcul en fonction de la taille de fichier, avec des fichiers qui different avec environ 30% de probabilite par ligne

4.3 Analyse des résultats expérimentaux

On voit avec le graphe en annexe que le temps semble en effet etre entre quadratique et lineaire.

5 Coût du patch en octet (1 point)

Il suffit juste de changer la fonction de cout en disant que cela coute aussi cher de supprimer que d'ajouter, et quasmient deux fois plus cher de faire une subsititution. Le cout est alors approximativement la longeure de la ligne somme avec le nombre de chiffres pour representer le numero de la ligne. Cela change peu le probleme.

6 Bonus : Pourrait-on utiliser une technique de blocking ? (1 point)

Avec l'algorithme tel que presente non, en effet on risque de ne pas trouver le chemin optimal puisque l'algorithme ne sait que aller dans le coin d'un bloc. Pour faire du blocking ca serait de couper les fichiers en parties de plus en plus petites et essayer de transformer chacune des parties d'entree en chacune des parties d'entrees. Le cout de recoller chaque partie risque de s'averer assez haut

Annexe

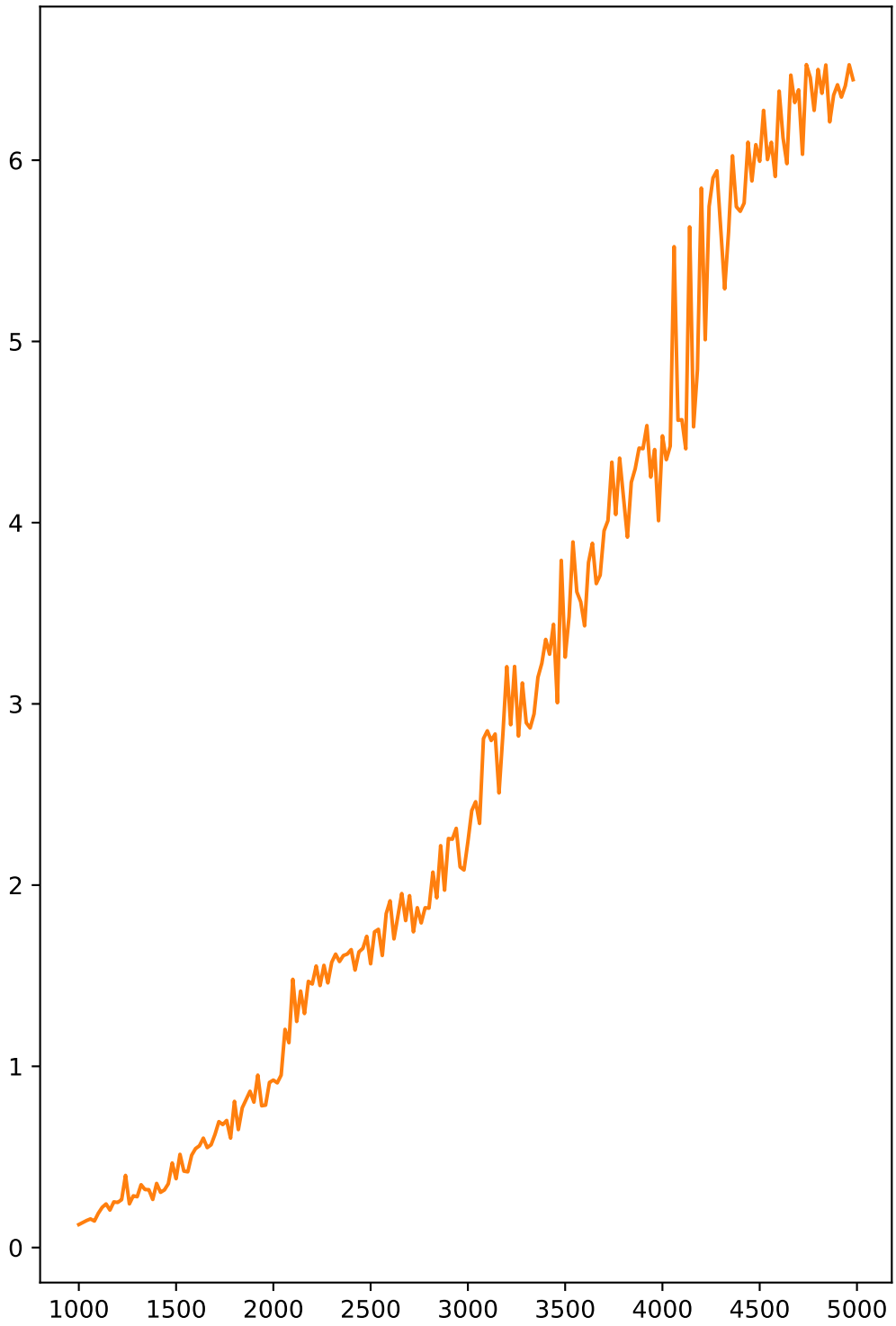

FIGURE 2 – Mesures des temps minimum en fonction de la longeure