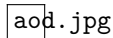


Rapport de TP 4MMAOD : Patch optimal entre deux fichiers

Nabil Bensrhier (2A ISI)
Redouane Yagouti (2A IF)

12 novembre 2019

1 Graphe de dépendances (1 point)

aod.jpg

Étant donné deux fichiers A comportant n lignes et un fichier B comportant m lignes, notre algorithme **itératif** se base principalement sur la fonction de Bellman du problème qui a été déjà fournie.

L'idée principale est de commencer à trouver les coûts correspondants à chaque transformation d'un fichier source A contenant i lignes à un fichier cible B contenant j lignes avec $i \in \{1, \dots, n\}$ et $j \in \{1, \dots, m\}$ et d'en calculer les coûts, jusqu'à arriver à calculer le coût $f(n, m)$ qui correspond au coût final de la transformation du fichier A en fichier B . On peut stocker les coûts de Bellman dans une matrice de taille (n, m) (un double pointeur en C)

On peut proposer l'algorithme **itératif** suivant pour le calcul de la matrice de Bellman :

```
Calculate BellmanMatrix( $n, m$ )  
  matrix  $\leftarrow$  matrice_vide_avec_n_lignes_et_m_colonnes  
  for  $i = 0$  to  $n$   
    for  $j = 0$  to  $m$   
      if  $i = 0$  and  $j = 0$   
        matrix[0][0]  $\leftarrow$  0  
      else if  $i = 0$   
        sum  $\leftarrow$  0  
        for  $k = 1$  to  $j$   
          sum  $\leftarrow$  sum + 10 + L( $B, k$ )  
        matrix[0][j]  $\leftarrow$  sum  
      else if  $j = 0$   
        matrix[i][0]  $\leftarrow$  10 * i  
      else  
        a  $\leftarrow$  matrix[i][j-1] + 10 + L( $B, j$ )  
        b  $\leftarrow$  matrix[i-1][j-1] + C( $i, j$ )  
        c  $\leftarrow$  matrix[i-1][j] + 10  
        matrix[i][j]  $\leftarrow$  Min(a, b, c)
```

Il nous reste alors de trouver le chemin de coût minimal dans notre matrice en partant de l'élément d'indices (n, m) jusqu'à arriver à l'origine $(0, 0)$.

2 Principe de notre programme (1 point)

On cherche le chemin optimal qui commence à partir du point de coordonnées (n, m) et qui fini au point d'origine $(0, 0)$. À partir du chemin généré on saura quelles sont les opérations qu'on doit faire sur le Patch en se servant des conditions données sur le sujet. Ainsi la fonction f peut être vue comme une matrice qu'on parcourt du point le plus bas à droite jusqu'à arriver à l'origine :

- (a) Une direction à gauche correspondra à un ajout dans le Patch. **ADD**
- (b) Une direction selon la diagonale correspondra à une substitution/aucune action dans le Patch. **NOTHING, SWAP**
- (c) Une direction en haut correspondra à une destruction dans le Patch. **DELETE**

REMARQUE :

Le plus court chemin est choisit à l'aide de la matrice générée et non directement sur la matrice. Car il faut prendre en considération les coûts de passage de $(i, j-1)$ à (i, j) , de $(i-1, j-1)$ à (i, j) et de $(i-1, j)$ à (i, j) . Le point essentiel à capter c'est qu'il faut commencer du point (n, m) et finir au point $(0, 0)$.

L'étape finale d'écriture dans le Patch consiste à suivre le chemin minimal et d'en extraire les opérations : **ADD, DELETE, NOTHING, SWAP** correspondante pour chaque coordonnée (i, j) rencontrée, où i correspond à une ligne de A et j correspond à une ligne de B .

Principe :

En général on se trouve dans des cas où les fichiers contiennent un très grand nombre de lignes.

Donc le principe est de calculer des sous matrices des coûts au lieu de calculer toute la matrice à la fois. En effet, on parcourt les fichier tout en calculant les coûts sans avoir les stocker dans une matrice, jusqu'à un seuil à partir duquel on peut stocker la dernière sous matrice de taille seuil x seuil, de telle façon que cette sous matrice est supportée par la mémoire. Ensuite, on calcule le chemin minimal jusqu'à la fin de la sous matrice soit ($i = dei$ ou $j = dej$). Enfin, les nouveaux fichier sont du début jusqu'au coordonnées de l'instruction calculée par le chemin minimal. Et on répète cette démarche jusqu'à l'arrivée à (0, 0).

3 Analyse du coût théorique (3 points)

3.1 Nombre d'opérations en pire cas :

Justification : Le coût de notre programme est principalement dominé par le coût du calcul des $f(i, j)$ pour $i \leq n$ et $j \leq m$.

Pour un fichier A contenant n lignes, et une fichier B contenant m lignes, on calcul d'abord la première ligne par la donnée de $f(0, j)$, ce coût est de $n - 1$. De même on calcul le coût de calcul de la première colonne par la donnée de $f(i, 0)$ ce qui coûte $m - 1$ opérations.

Après pour calculer les autres éléments, il faut les comparer avec les voisins de haut, de gauche et diagonale : l'opération pour trouver le minimum coûte 3 comparaisons, et ceci se fait $(n - 1)(m - 1)$ fois.

Le coût est donc :

$$C(n, m) = 1 + n - 1 + m - 1 + 3(n - 1)(m - 1)$$

3.2 Place mémoire requise :

Justification : Notre premier programme est correct pour les 3 premier tests de benchmark (00 , 01 , 02), est ceci dû au fait que le pc sur lequel on fait les tests a les caractéristiques suivantes :

Mémoire : 15.5 Gio

Processeur : Intel^R Core i7-8750H CPU @ 2.20GHZ * 12

Carte graphique : GeForce GTX 1060/PCIe/SSE2

Type de système : 64 bits

Donc, on estime que la taille maximale de la matrice des coûts que le PC puisse la sauvegarder sans avoir perdre des coûts est :

un fichier $F1$ de taille 20000 lignes, de même un fichier $F2$ de taille 20000.

Et comme chaque coût est sauvegarder sur un *int*, c-à-d un espace de 4 octet.

Ce qui est équivalent à $20000^2 \cdot 32 = 12Gio$

3.3 Nombre de défauts de cache sur le modèle CO :

Justification : La matrice qu'on a créé pour la fonction de Bellman parcourt la matrice ligne par ligne, en effet on calcul la première ligne, après on calcul le premier élément de la deuxième ligne, ceci permet donc de calculer la deuxième ligne toute entière. Ainsi le défaut de cache est égale à $\frac{nm}{L}$.

4 Compte rendu d'expérimentation (2 points)

4.1 Conditions expérimentales

4.1.1 Description synthétique de la machine :

Les mesures ont été effectuée sur un processeur Intel R Core TM i7-8750H CPU @ 2.20GHz 12.

Le OS utilisé est Ubuntu.

4.1.2 Méthode utilisée pour les mesures de temps :

Les mesures ont été effectuée 5 fois de suite à chaque programme à l'aide de la commande **time** de Unix.

4.2 Mesures expérimentales

Il faut souligner que les tests sont choisis pour un seuil de 30 000 lignes, pour un PC de 16 GB de RAM. Donc le temps du calcul est rapide par rapport à un seuil de 10 000 choisi pour les PC de 4 GB de RAM, donc avant de tester il faut modifier le seuil selon les caractéristiques de chaque PC.

Le benchmark 04 est juste une question du temps, on a pas réussi à le valider mais, le temps nécessaire pour le

	coût du patch	temps min	temps max	temps moyen
benchmark_00	10345	0.023	0.027	0.025
benchmark_01	12437	3.492	3.636	3.522
benchmark_02	3559	130.54	137.23	133.54
benchmark_03	19088	437	420	429

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

trouver est estimé de 4h.

4.3 Analyse des résultats expérimentaux

5 Coût du patch en octet (1 point)

On s'inspirant de l'équation de Bellman donnée, le coût du patch en octet correspond à :

$$f(i, j) = \min \begin{cases} f(i-1, j-1) + C(i, j) \\ f(i-1, j) + 4 + \log(i) + L_i^A \\ f(i, j-1) + 4 + \log(i) + L_j^B \end{cases}$$

Avec

$$C(i, j) = \begin{cases} 0 & \text{si } A_i = B_j \\ 4 + L_j^B + L_i^A + \log(i) & \text{sinon} \end{cases}$$