Lignes répétées

Projet d'algorithmique

Alexandre Petit

Licence Informatique, 2ème année

UFR des sciences et technique de Rouen

Analyse préalable au développement du programme

Dans une implantation classique, la taille d'un fichier est limitée à `LONG_MAX = 2147483647` caractères. L'alphabet usuel, l'ASCII comprend `UCHAR_MAX + 1 = 256` caractères distincts.

Dans la suite de notre analyse, mathématique, on assimilera toute valeur numérique supérieure à 2^{64} à la valeur infinie. De façon symétrique, on assimilera toute valeur inférieure à 10^-5 à la valeur nulle, 0.

L'objet de ce programme est de rechercher des occurrences répétées de lignes dans un ensemble de textes. Nous nommerons *motif* ces éléments.

Facteurs

Les attributs de premiers ordres sont :

1. la **taille du fichier** de référence (en octets), autrement dit le nombre de caractères sur l'alphabet ASCII ;
2. le **nombre de lignes** du fichier de référence ;

que nous considérerons dans leur majoration. De ces deux éléments on déduit :

1. la **longueur moyenne** de chaque ligne, c'est-à-dire le nombre de caractères par motif.

Enfin, dans le but de dégager un ordre d'idées de la distribution des données pour un volume important de texte, nous tenterons de fixer certains repères quant aux paramètres suivants :

1. le **nombre de motifs** (nombre d'entrées dans la table)
2. le **nombre d'occurrences** (nombre de lignes à mémoriser)

Les cas extrêmes, comme par exemple un fichier ayant un nombre maximal de lignes de longueur 1, sont des cas ayant de faibles probabilités de survenir, mais ils sont possibles. Il est donc intéressant de les expliciter afin de révéler certaines majorations.

Dans la suite la mention *de référence* (ou *du fichier de référence*) sera omise pour alléger la rédaction.

Distribution des données

- La taille du fichier est considérée comme maximale dans l'ensemble de l'analyse.
- Le nombre de lignes est considéré comme
    - **cas extrême A (minoration)** : une unique ligne de `LONG_MAX` caractères ;
    - **cas extrême B (majoration)** : `LONG_MAX` lignes composées d'un unique caractère ;
    - dans les cas moyens, on considèrera des lignes allant de 10 à 80 caractères.

- Le nombre de motifs (nombre d'entrées dans la table) :
  - **cas extrême 1 (minoration)** : un unique motif est répété à chaque ligne du fichier.
  - **cas extrême 2 (majoration)** : dans sa majoration, le *nombre de motifs potentiels* découle directement de la taille des lignes et de leur nombre.
  - dans les cas moyens, on se propose d'évaluer le système pour plusieurs valeurs intermédiaires ;
- Le nombre d'occurrences des motifs est sans aucun doute la valeur la plus incertaine et donc la plus délicate à approcher. Cela étant, elle nous intéresse.
  - **cas extrême α (max)** : parmis l'ensemble des motifs potentiels, un unique motif est répété à chaque ligne du fichier. Le nombre d'occurence est alors maximal.
  - **cas extrême β (min)** : chaque motif est distinct (dans la limite du nombre de motifs potentiels). Le nombre d'occurrence de chaque motif est alors minimal.
  - cas maximal pour un motif quelconque : majoré par le nombre de lignes ;
  - cas minimal pour un motif quelconque : minoré par 1 de façon constante ;
  - cas moyens : distribution linéaire, exponentielle, quadratique, aléatoire

Remarque : les cas extrême A et B proposent des situation de choix pour tester les limites du programme. L'archive du projet contient deux sous-programmes - placés dans le dossier `tools/` - qui permettent de générer des fichiers textes que l'on pourra assimiler à ces cas de figure.

Application numérique

Posons $l_{max} =$ `LONG_MAX`

et $a_{max} =$ `UCHAR_MAX + 1` .

nombre de caractères / lignes	nombre de lignes	motifs potentiels	nombre moyen d'occurrences de chaque motif potentiel	nombre maximal d'occurences d'un motif effectif
c	$n = l_{max}/c$	$m_p = (a_{max})^c$	n/m_p	n
l_{max}	1	$256^{l_{max}} = \infty$	0	1
1	2147483647	$256^1 = 256$	8388607	2147483647
2	1073741824	$256^2 \simeq 7.10^4$	16384	1073741824
3	715827882	$256^3 \simeq 2.10^6$	42	715827882
5	429496729	$256^5 \simeq 10^{12}$	0,0004	429496729
10	214748364	$256^{10} = \infty$	0	214748364
40	53687091	$256^{40} = \infty$	0	53687091
80	26843545	$256^{80} = \infty$	0	26843545

Conclusion

Il en ressort que pour une composition parfaitement aléatoire, les chances pour un motif potentiel d'apparaître même une fois dans le texte sont infiniment petites dès lors que les motifs ont une longueur moyenne de 5 caractères, ce qui est peu. D'autre part, on remarque que le nombre d'occurrences peut atteindre de très grandes valeurs, et ce même pour des motifs de longueur relativement élevée.

Sans plus de précision sur la nature des données, on peut supposer, avec une certaine précaution, que le nombre d'occurrences d'un motif a de fortes chances d'être de faible valeur. Inutile donc de prévoir à l'avance des espaces de stockage de grande taille pour le référencement du nombre d'occurrences.

Choix des structures de donnée

En vue de parvenir à choisir des structures de données adaptés aux besoins du programme, résumons l'objet du projet. La principale tache du programme est d'identifier la position des lignes de texte qui sont répétées plusieurs fois, au sein d'un même fichier, ou à travers différents fichiers. Pour chaque ligne, il sera donc nécessaire de mémoriser les fichiers dans lesquelles elle apparaît, puis pour chacun de ces fichiers, les lignes auxquelles elle apparaît. Afin de structurer ces informations, il semble intéressant de faire appel à une table de hashage. En effet, une table de hashage permet d'accéder à des informations par association, autrement dit par clé. Nous articulerons donc les informations relatives aux motifs relevés autour de cette structure. Aussi, afin de conserver une trace des clés de la table, les motifs seront stockés dans une liste.

Une fois le stockage des motif passé en revue, on peut désormais s'intéresser aux autres données essentielles du programme qui sont les noms de fichiers et les listes d'occurences des motifs pour chaque fichier.

Le noms des fichiers à analyser seront stockés dans une liste centralisée. De cette façon, on pourra associer à chaque fichier un identifiant unique qui sera son indice dans la liste.

Enfin, à chaque motif sera associé une liste de relevés d'informations. Chaque relevé sera propre à un fichier spécifique et contiendra la liste des occurrences du motif dans ce fichier. Afin d'identifier le fichier auquel est associé le relevé, on fera usage d'un identifiant fichier qui correspond à l'indice du fichier dans la liste centralisée.

Synthèse

- Table de hashage (registre)
- Liste des fichiers
- Liste des relevés d'informations (numéros de lignes par fichier)
- Liste des numéros de lignes
- Liste des motifs

Notons que la table de hashage sera utilisée pour des insertions exclusivement, il n'est donc pas nécessaire d'implanter le retrait. Ainsi, le module du cours d'algorithmique convient parfaitement.

Pour les listes, bien que chacune travail avec un type de donnée différent, elles se présentent comme similaire dans l'usage. On choisira donc d'implanter un tableau dynamique polymorphe qui servira à l'ensemble. Des solutions alternatives pourront toujours être apportées en cours de projet en cas de manquement.

Modules développés

- **ftrack** : relevé d'informations (liste des occurences d'un motif pour un fichier) ;
- **dyna** (dynamic array) : tableau dynamique polymorphe avec implantation du tri-rapide ;
- **hashtbl** : *repris du cours d'algorithmique et inchangé* ;
- **Intracker** : le module principal, offre une structure de donnée pour stocker les paramètres du programme et implante les algorithmes d'analyse et d'affichage ;
- **Inscanner** : le module consacré à la récupération de lignes de texte dans un flot donné, propose des fonctionnalités de transformation et de filtrage du contenu lu ;

- **quick_sort** : implante le tri rapide. Ce module est repris du *tp 3* d'algorithmique et semblable à la fonction standard `qsort`.

Illustration

Ci-dessous figure une illustration de l'état de l'application après analyse des trois fichiers textes suivants :

fichier1

```
dune
dune005
yAgA$$$
```

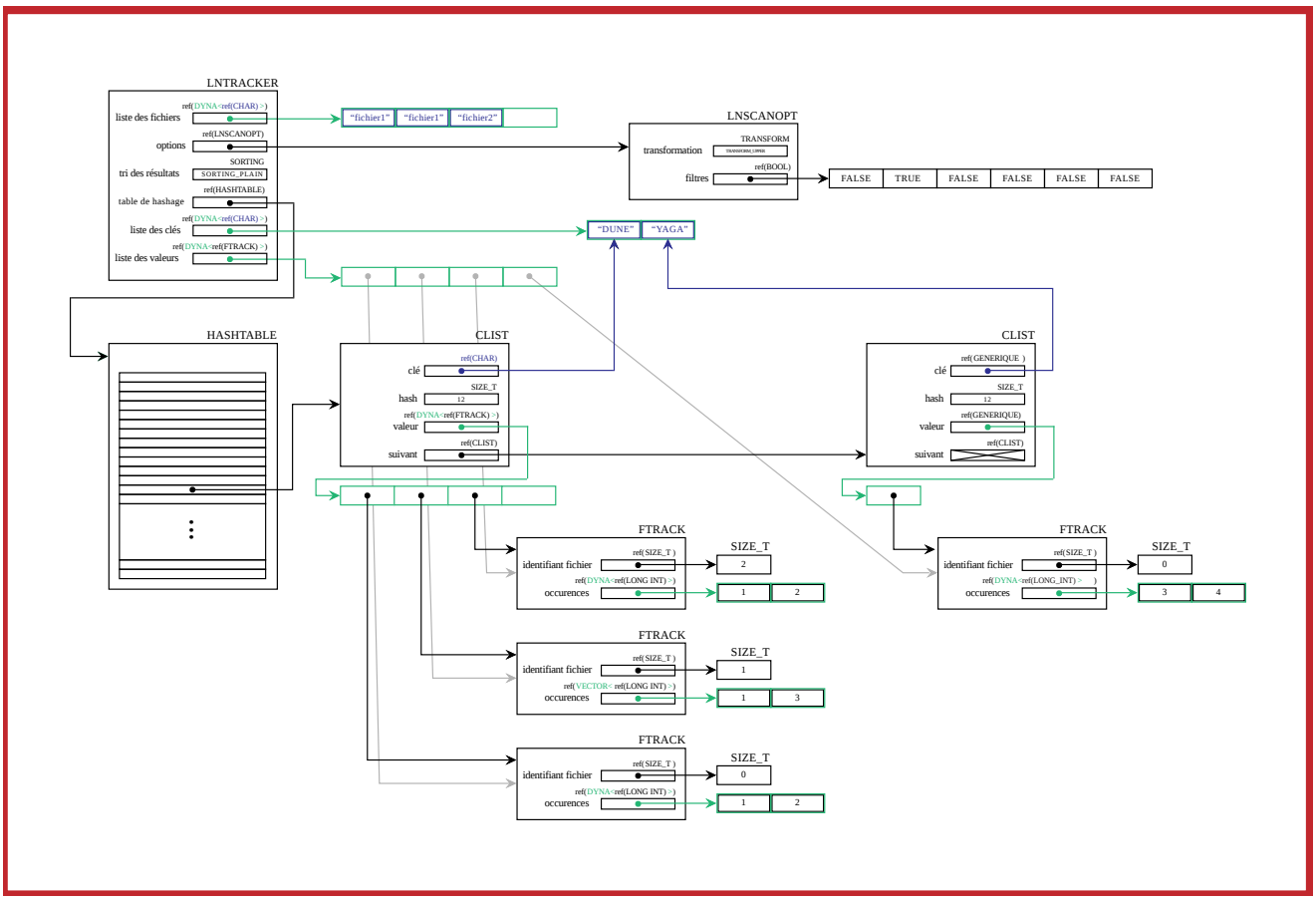
fichier2

```
DUNE45
45DUNE
vrille
```

fichier3

```
Dune
Dune
Cerf
```

Dans cet exemple, le programme a été initialisé avec l'option de transformation en capitales de majuscules et un filtre restreint aux caractères alphabétiques. Les schémas reprennent en grande partie les conventions développées lors du cours d'algorithmique. Une seule entorce a été faite à ces conventions pour simplifier l'affichage des tableaux dynamique (tracés en vert) lorsque les valeurs pointées sont de type *char* * (type représenté en bleu) ou *long int*. Les données mémorisées par les tableau dynamiques étant homogènes pour chaque utilisation, le type des données est explicité dans le type du tableau dynamique à la façon des objets *vector* en *C++*.



Le tableau dynamique qui contient la liste des noms de fichier met en évidence la façon dont le tableau est redimensionné au fur et à mesure que les données s'accumulent. Ici, la tableau est parti d'une capacité de 1 élément pour évoluer vers une capacité de 2 lorsque le deuxième fichier a été détécté, puis 4 pour faire place au troisième fichier.

Limites

La taille du buffer utilisé pour la lecture des lignes a été fixée à `STRINGLEN_MAX = 65535` aussi toute ligne longueur supérieure à cette limite sera tronquée. Cependant, si les préfixes de deux lignes de `STRINGLEN_MAX` caractères sont identiques, il peut être raisonnable (prêter attention quand même au contexte dans lequel le programme est employé) de penser que les lignes sont identiques dans leur intégralité. Effectivement, la probabilité technique que deux lignes de 65535 caractères soient identiques est de 1 pour 65535^{256}. Alors plus ..

Astuces

Inialiser srand à l'aide d'un timesamp afin d'avoir de nouvelles valeurs aléatoires pour de chaque exécution.

```
#include <time.h>
srand((unsigned int) time(NULL));
```

puis pour un premier balayage de l'application et de sa robustesse, glisser une évaluation d'un modulo de `rand()` dans la macro-fonction consacré à la gestion des erreurs.

```
#define ON_VALUE_GOTO(expr, value, label)     \
    if (rand() % 10 == 0) {                   \
      goto label;                             \
    }                                         \
    if ((expr) == (value)) {                  \
      goto label;                             \
    }
```