Patricia Trie | Fonctions de hachage | TP 4

Présenté par: Omar ALLOUCH & Charaf Eddine EL KIHAL

Implémentation et explication du code *(C et Python)*

Contrairement au TP précédent, on a décidé de présenter les deux codes (C & Python) plutôt que simplement Python (seul Python fonctionne), afin de présenter comment on a procédé et peut-être d'avoir un retour sur ce qui aurait pu être le problème.

Commençons avec C:

En partant des fichiers donné, une grande partie de code est déjà écrite, et il ne reste qu'ajouter la fonction de recherche and le Patricia Trie, et le comptage des intersections.

Le comptage est simple, il s'agit d'ouvrir le second fichier, parcourir chaque ligne / mot, et faire un appel à la fonction de recherche qu'on va voir en détail plus tard. On décide à l'aide du résultat de cette fonction si on incrémente le nombre total d'intersection ou pas.

```
int main() {
    ...
  fichier2 = fopen(f2, "r");
  if (fichier2 == NULL) {
    printf("erreur ouverture fichier %s.\n", f2);
    exit(2);
  }

  int count = 0;
```

```
  while (!feof(fichier2) && (fgets(ligne, sizeof ligne, fichier2
    nettoyage(ligne, mot);
    if (strlen(mot) != 0)
      if (recherche_Patricia(mon_Patricia, mot))
        count++;
  }
  fclose(fichier2);

  printf("Nombre de mots communs : %d\n", count);
    ...
}
```

La fonction de recherche est un peu plus compliqué, mais voici la logique qu'on a suivie :

1. On cherche la première lettre de la chaine dans la racine

   ```
   position = valeur[0] - DEBUT;
   ```

2. Si la chaine n'existe pas à la position trouvée ⇒ on retourne FALSE

   ```
   if (racine->cle[position] == NULL)
     return FALSE;
   ```

3. Si la chaine existe, on vérifie si elle est finie
4. Si la chaine est finie ⇒ on retourne TRUE

   ```
   if (strcmp(racine->cle[position], valeur) == 0 &&
         racine->fin[position] == TRUE)
     return TRUE;
   ```

5. Sinon, on cherche la première différence entre la chaine et la chaine gardée

```
indice = 0;
while (racine->cle[position][indice] == valeur[indice])
  indice++;
```

6. Si la chaine est plus petite que la chaine gardée (la différence est une lettre) ⇒ on retourne FALSE

```
if (racine->cle[position][indice] != '\0')
  return FALSE;
```

7. Si la chaine dépasse la chaine gardée (la différence est \0), il est possible que la chaine existe ⇒ on vérifie s'il existe un noeud fils. S'il n'y a pas ⇒ on retourne FALSE

```
if (racine->fils[position] == NULL)
  return FALSE;
```

8. S'il y a ⇒ On retourne le résultat de la recherche dans le fils avec le reste de la chaine.

```
i = indice;
while (valeur[i] != '\0') {
  reste[i - indice] = valeur[i];
  i++;
}
reste[i - indice] = '\0';

// on continue la recherche dans le fils
return recherche_Patricia(racine->fils[position], reste);
```

Résultats

Les résultats sont un peu bizarre, indiquant une erreur dans le code qu'on n'a pas réussi à trouver.

Par exemple, si exécute le programme avec les paramètres `corncob.txt` comme premier fichier, et `shakespeare.txt` comme second fichier, on obtient 15763, ce qui n'est exact. Si on inverse les fichiers, on obtient 15767, ce qui n'est pas le même résultat, ni le bon. Et parfois, on obtenait `Segmentation fault`.

Voyant que ce n'était pas ce qu'on attendait, on a décidé d'utiliser Python, car on y ai plus habitués.

Python

Le code Python utilise des classes plutôt que de simples structures de données pour faciliter le processus d'insertion et de recherche (inspiré du code C).

Le code Python est très bien documenté, nous ne réécrirons pas tout ici.

La principale différence est dans la structure des nœuds, on l'a rendue plus simple, en faisant en sorte que chaque nœud contienne elle-même la valeur de clé et en ayant *children* comme attribut contenant le reste des clés / mots. La logique restante est fortement inspirée du code C donné.

Résultats

L'exécution du code Python nous donne un résultat conforme à ce qu'on obtenait dans les TPs précédents : 15760.