Moteur de recherche de texte basé sur l'algorithme KMP

Auteur:

Chuchen PENG

Fei YANG

l'algorithme KMP

L'algorithme de Knuth-Morris-Pratt (souvent abrégé par algorithme KMP) est un algorithme de recherche de sous-chaîne, permettant de trouver les occurrences d'une chaîne P dans un texte S. Sa particularité réside en un pré-traitement de la chaîne, qui fournit une information suffisante pour déterminer où continuer la recherche en cas de non-correspondance. Cela permet à l'algorithme de ne pas ré-examiner les caractères qui ont été précédemment vérifiés, et donc de limiter le nombre de comparaisons nécessaires.

L'algorithme a été inventé par Knuth et Pratt, et indépendamment par J. H. Morris en 1975.

Introduction du moteur

- Le moteur met en œuvre la capacité de trouver n'importe quel mot ou énoncé en parallèle à partir d'une bibliothèque de textes, en retournant le nombre maximum d'énoncés associés qui peuvent être mis en correspondance dans chaque texte.
- La mise en œuvre du programme est divisée en plusieurs parties principales, à savoir：
  - Lecture de bibliothèques de textes, pré-traitement de bibliothèques de textes
  - Lecture de mots clés, découpage de toutes les sous-chaînes consécutives d'un mot clé
  - La correspondance par Kmp
  - Recherche multithread

Lecture et pré-traitement des textes

```
def getText(txtfile):
    txt = open(txtfile,'r').read()
    txt = txt.lower()    #Changer tous les mots du texte en lettres minuscules
    for ch in '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~':
        txt = txt.replace(ch, ' ')  #Remplacer les caractères spéciaux dans le texte par des espaces
    return txt
```

Pour faciliter la correspondance, tous les mots du texte de la bibliothèque sera converti en lettres minuscules, et le mot-clé saisi ensuite devra être converti en minuscules en même temps. Ce projet ne fait pas de correspondance pour les caractères spéciaux, uniformément remplacés par des espaces.

La correspondance par Kmp

Ici, la correspondance KMP implémente qu'étant donné la chaîne parentale et la sous-chaîne
Renvoie la position de la sous-chaîne dans la chaîne parentale, et si elle n'est pas présente, renvoie -1.

```python
def kmp(mom_string,son_string):
    # Passer une chaîne parente et une sous-chaîne
    # Renvoie la première position de la sous-chaîne correspondante, s'il n'y a pas de correspondance, renvoie -1
    test=''
    if type(mom_string)!=type(test) or type(son_string)!=type(test):
        return -1
    if len(son_string)==0:
        return 0
    if len(mom_string)==0:
        return -1
    #Trouvez le tableau 'next'
    next=[-1]*len(son_string)
    if len(son_string)>1:# # Le if ici est dans le cas où la liste est hors limites.
        next[1]=0
        i,j=1,0
        while i<len(son_string)-1:
            if j==-1 or son_string[i]==son_string[j]:
                i+=1
                j+=1
                next[i]=j
            else:
                j=next[j]
```

Saisir le pré-traitement des mots-clés

```
keyword1=input().lower()
keyword1=keyword1.split(' ')
#Rechercher des sous-chaînes consécutives
str_set=[keyword1[i:i + x + 1] for x in range(len(keyword1)) for i in range(len(keyword1) - x)]
#Recherche à partir de la plus longue sous-chaîne
str_set=str_set[::-1]
```

Si l'entrée est une phrase, de séparer tous les mots de cette phrase, parce que comme un moteur de recherche, de retourner tous les textes pertinents, le mot-clé d'entrée dans ou hors du texte, de sorte que la recherche de tous les mots clés de sous-chaînes consécutives, et de la plus longue sous-chaîne, c'est-à-dire, le mot-clé lui-même pour commencer la recherche, une fois trouvé, les suivants n'ont pas besoin de trouver.

Recherche multithread

```
threads = []
for txt in filelist1:
    t = threading.Thread(target=search_in_txt,args=('./text/'+txt,str_set))
    threads.append(t)
for i in range(len(filelist1)):
    threads[i].start()
for i in range(len(filelist1)):
    threads[i].join()
```

Lorsque la base de texte est énorme, la recherche multithread peut augmenter de manière significative la vitesse de calcul, avec un taux d'accélération théorique de N fois (N est le nombre de cœurs de processeurs).

Recherche du texte

```
def search_in_txt(txt,str_set):
    mom_string=getText(txt)
    for item in str_set:
        kw=' '.join(item)
        if kmp(mom_string,kw)>=0:
            print(txt+" contain: "+kw+'\n')
            #Rechercher à partir du plus long, puis revenir une fois trouvé, il n'est pas nécessaire de rechercher les plus courts
            return
```

Les paramètres d'entrée de cette fonction sont respectivement le texte et l'ensemble des sous-chaînes.
La sortie est la plus longue sous-chaîne contenue dans le texte.

Démonstration

- Entrée *soldiers*， retour hamlet.txt

```
input keywords: (for example:soldiers, or Muggle)
soldiers
./text/Hamlet.txt contain: soldiers
```

- Entrée *Muggle*， retour tous les documents contenant “Muggle"

```
input keywords: (for example:soldiers, or Muggle)
Muggle
./text/1.Harry Potter and the Philosopher's Stone.txt contain: muggle
./text/7.Harry Potter and the Deathly Hallows.txt contain: muggle
./text/3.Harry Potter and the Prisoner of Azkaban.txt contain: muggle
./text/4.Harry Potter and the Goblet of Fire.txt contain: muggle
./text/2.Harry Potter and the Chamber of Secrets.txt contain: muggle


./text/6.Harry Potter and The Half-Blood Prince.txt contain: muggle
./text/5.Harry Potter and the Order of the Phoenix.txt contain: muggle
```

Démonstration

- Entrée “*he has chosen Draco in revenge*”

```
input keywords: (for example:soldiers, or Muggle)
he has chosen Draco in revenge
./text/6.Harry Potter and The Half-Blood Prince.txt contain: he has chosen draco in revenge

./text/Hamlet.txt contain: he has

./text/1.Harry Potter and the Philosopher's Stone.txt contain: he has

./text/2.Harry Potter and the Chamber of Secrets.txt contain: he has

./text/3.Harry Potter and the Prisoner of Azkaban.txt contain: he has

./text/4.Harry Potter and the Goblet of Fire.txt contain: he has

./text/7.Harry Potter and the Deathly Hallows.txt contain: he has

./text/5.Harry Potter and the Order of the Phoenix.txt contain: he has
```

- Cette phrase se trouve dans la partie 6, mais tous les autres textes contiennent également “he has”