tp_l8dn003_20_03

March 4, 2020

0.1 TP n°3: recherche dichotomique et évaluation des tris

0.1.1 recherche linéaire vs dichotomique

1. On part d'un lexique ("bag of words") constitué à partir d'un texte comme dans le TP n°2 (liste de mots).
2. Implémenter une fonction de recherche qui reçoit un mot en paramètre, et dit si oui ou non ce mot est dans le lexique.
3. Ajouter à cette fonction un "compteur" de comparaisons: la fonction de recherche doit non seulement retourner un booléen, mais en plus la valeur de ce compteur.
4. En procédant à un tirage aléatoire des mots dans un autre texte, déterminer les nombres moyen, min et max de comparaisons.
5. Implémenter une deuxième fonction de recherche qui procède par *dichotomie* dans la liste préalablement triée (par `sorted()` ou par `quicksort()` ou par `bubblesort()`)
6. Ajouter à cette fonction un compteur pour déterminer le nombre de comparaisons.
7. Comparer les nombres moyen, min et max de comparaisons dans les deux cas.

0.1.2 bénéfice combiné quicksort() + recherche dichotomique

1. On utilise maintenant des mesures de temps de calcul pour comparer plusieurs algorithmes (cf cellule "mesure du temps")
2. Pour simplifier les choses, on travaille sur des listes d'entiers générées aléatoirement: on cherche toujours à implémenter une fonction de recherche.
3. Comparer les performance d'un programme qui fait une recherche linéaire dans une liste non triée avec celle d'un programme qui réalise d'abord un tri par `quicksort()` et fait ensuite une recherche dichotomique.
4. Dans le cas précédent, réalise un tri pour chaque recherche, ce qui est inutilement coûteux. On va chercher à savoir au bout de combien de recherches c'est plus intéressant de faire un tri une fois et de faire ensuite des recherches dichotomiques.

[1]:
```
def charge_fichier(nf):
    lgns = []
    with open(nf, "r", encoding="utf8") as f:
        for ligne in f:
            lgns.append(ligne.strip())
    return lgns
def charge_lexique(nf):
    liste_lignes = charge_fichier(nf)
    lexique = []
```

```
    for l in liste_lignes:
        for w in l.split():
            if w not in lexique:
                lexique.append(w)
    return lexique
```

[2]:
```
# Fichiers disponibles : demo-file-utf8.txt (16023 word-forms)
#                        pensees_pascal_utf8.txt (8842 word forms)
lex0 = charge_lexique("demo-file-utf8.txt")
print(len(lex0))
lex1 = charge_lexique("pensees_pascal_utf8.txt")
print(len(lex1))
```

```
16023
8842
```

[3]:
```
# Réponse à la question 2
def rech_lineaire(lex,m):
    for w in lex:
        if m == w: return True
    return False
```

[4]:
```
# Mesure du temps (de calcul)
import time

temps = []

for w in lex1:
    t1 = time.process_time()
    rech_lineaire(lex0,w)
    t2 = time.process_time()
    temps.append(t2-t1)

print(min(temps)*1000, max(temps)*1000, sum(temps)/len(temps)*1000)
```

```
0.0009999999992515995 1.4170000000000016 0.33466048405338655
```