1 Correction du TD 11

Première demi-heure : Construction d'une liste aléatoire

Exercice 1 :

```
import random
def mot_alea (l) :
    l = [ chr(97+random.randint(0,25)) for i in range(l) ]
    return "".join(l)

taille  = 20
N       = 100000
mots    = [ mot_alea(10) for _ in range (N) ]
```

Exercice 2 :

```
import time
debut = time.clock()
for k in cherche :
    i = mots.index(k)
fin = time.clock()
print ("recherche simple",fin - debut)
```

Seconde demi-heure : Recherche dichotomique

Exercice 3 :

```
# recherche dichotomique
def dicho (mots, x) :
    a = 0
    b = len(mots)-1
    while a < b :
        m = (a+b)//2
        t = mots[m]
        if t < x : b = m-1
        elif t == m :
            return
        else :
            a = m+1
    return a

mots.sort()

debut = time.clock()
for k in cherche :
    i = dicho(mots, k)
fin = time.clock()
print ("dichotomie",fin - debut)
```

Soit N le nombre de mots dans la liste :
– coût de la recherche simple : $O(N)$

– coût de la recherche dichotomique : $O(\ln N)$

Troisième demi-heure : Trie

Soit L la longueur maximale des mots et C le nombre de lettres distinctes, avec un trie, le coût de la recherche est majoré par : $O(L \ln C)$.

Exercice 4 :

```
def build_trie(mots) :
    trie = { }
    for m in mots :
        r = trie
        for c in m :
            if c not in r : r[c] = { }
            r = r[c]
    return trie
```

Troisième demi-heure : Recherche dans un trie

Exercice 5 :

```
def lookup(trie, m) :
    r = trie
    for c in m :
        if c in r :
            r = r[c]
        else :
            return False
    return True
```

Exercice 6 :

```
debut = time.clock()
for k in cherche :
    i = lookup(trie, k)
fin = time.clock()
```

Pour aller plus loin ou pour ceux qui ont fini plus tôt

```
import random, time

def mot_alea (l) :
    l = [ chr(97+random.randint(0,25)) for i in range(l) ]
    return "".join(l)

for N in [100,200,500,1000,2000,5000,10000,20000,50000,100000,200000] :
    for taille in [5,10,20,50,100] :
        n       = 1000
```

```
mots    = [ mot_alea(10) for _ in range (N) ]
cherche = [ mot_alea(10) for _ in range(0,n) ]
mots   += cherche

if True :
    # recherche dichotomique
    def dicho (mots, x) :
        a = 0
        b = len(mots)-1
        while a < b :
            m = (a+b)//2
            t = mots[m]
            if t < x : b = m-1
            elif t == m :
                return
            else :
                a = m+1
        return a

    mots.sort()

    debut = time.clock()
    for k in cherche :
        i = dicho(mots, k)
    fin = time.clock()
    print ("dichotomie\t",N,"\t",taille,"\t",(fin - debut)/len(cherche))

if True :
    # recherche dichotomique
    dico = { m:0 for m in mots }

    debut = time.clock()
    for k in cherche :
        i = dico[k]
    fin = time.clock()
    print ("dictionnaire\t",N,"\t",taille,"\t",(fin - debut)/len(cherche))

if True :
    # trie

    def build_trie(mots) :
        trie = { }
        for m in mots :
            r = trie
            for c in m :
                if c not in r : r[c] = { }
                r = r[c]
        return trie

    def lookup(trie, m) :
        r = trie
        for c in m :
            if c in r :
                r = r[c]
            else :
                return False
        return True

    trie = build_trie(mots)
```

```
        debut = time.clock()
        for k in cherche :
            assert lookup(trie, k)
        fin = time.clock()
        print ("trie\t",N,"\t",taille,"\t",(fin - debut)/len(cherche))

    if False :
        # recherche simple
        debut = time.clock()
        for k in cherche :
            i = mots.index(k)
        fin = time.clock()
        print ("recherche simple\t",N,"\t",taille,"\t",(fin - debut)/len(cherche))
```

fin correction TD ?? □