Ramaneek Gill 1000005754

CSC373 Feb,3,2015

Problem Set 4

IDEA:
Always want to split the bigger chunks of the array and always want to split as close to the midpoint as possible to minimize the length of substrings.

Define Array:

A[p] = an array of ordered breakpoints from $p_1, \dots, p_k$ such that splitting a string s at breakpoints in the order of the array A minimizes the total time required.

Recurrence Relation:

A[0] = breakpoint p such that it is closest to $\left\lfloor \frac{len(s)}{2} \right\rfloor$
Note: at each I'th iteration there is i+1 substrings
A[i] = breakpoint p such that it is the closest $\left\lfloor \frac{len(s_{maxlen})}{2} \right\rfloor$ where s is the substring (out of i+1 different substrings) with the largest length.

Iterative Algorithm:

```
Algorithm(s, A, k=0):   #to start off the algorithm set k to 0

# get the first breakpoint
diff = infinity
For i = range(0, len(p)):
        If abs(p[i] - ⌊len(s)/2⌋) < diff
                A[k] = p[i]

P = P – A[k]  #remove the breakpoint from the list since it has been used

substrings[0] = s[  : A[k]]
substrings[1] = s[len(substring[k]) : ]

if len(substring[0]) >  len(substring[1]):
        big_string = substring[0]
        little_string = substring[1]
else:
        big_string = substring[1]
        little_string = substring[0]

A[k+1] = Algorithm(Big_string, A, k+1)[k+1]     #since algorithm returns an array we can index it
A[k+2] = Algorithm(little_string, A, k+1)[k+2]
Return A
```

This algorithm will always pick a breakpoint as close as to the middle of the string as possible and will always operate on the bigger split first. This way we try to make the list operations resemble binary search as close as possible to try and achieve half split strings each recursive call which reduces the amount of time to operate on each substring.