# Python program for implementation of suffix array operations

import time

import functools

from colored import fg, attr

class Suffix:

def \_\_init\_\_(self, index, suff):

self.index = index

self.suff = suff

# A comparison function used by sort() to compare two suffixes

def cmp(a, b):

if a.suff < b.suff:

return -1

elif a.suff > b.suff:

return 1

else:

return 0

# A utility function to build the suffix array

def build\_suffix\_array(txt, n):

# A structure to store suffixes and their indexes

suffixes = [Suffix(i, txt[i:]) for i in range(n)]

suffixes.sort(key=functools.cmp\_to\_key(cmp))

suffix\_arr = [suffixes[i].index for i in range(n)]

return suffix\_arr

# A utility function to print the suffix array

def print\_suffix\_array(T, suffix\_arr):

suffixes = []

for i in suffix\_arr:

suffixes.append(T[i:])

print("Suffix Array:", suffix\_arr)

print("Suffixes:", suffixes)

# A utility function to search about a pattern

def search(pat, txt, suffArr, n):

m = len(pat)

l = 0

r = n - 1

indices = []

# Do simple binary search for the pat in txt using the built suffix array

while l <= r:

# Find the middle index of the current subarray

mid = l + (r - l) // 2

# Get the substring of txt starting from suffArr[mid] and of length m

res = txt[suffArr[mid]:suffArr[mid] + m]

# If the substring is equal to the pattern

if res == pat:

# Add the index to the list

indices.append(suffArr[mid])

# Check for more occurrences of the pattern to the left of the current index

j = mid - 1

while j >= l and txt[suffArr[j]:suffArr[j] + m] == pat:

indices.append(suffArr[j])

j -= 1

# Check for more occurrences of the pattern to the right of the current index

j = mid + 1

while j <= r and txt[suffArr[j]:suffArr[j] + m] == pat:

indices.append(suffArr[j])

j += 1

return indices

# If the substring is less than the pattern

if res < pat:

# Move to the right half of the subarray

l = mid + 1

else:

# Move to the left half of the subarray

r = mid - 1

# If the pattern is not found

print("Pattern not found")

return indices

# build lcp array

def build\_lcp\_table(txt, suffix\_arr):

n = len(txt)

lcp = [0] \* n

rank = [0] \* n

for i in range(n):

rank[suffix\_arr[i]] = i

k = 0

for i in range(n):

if rank[i] == n - 1:

k = 0

continue

j = suffix\_arr[rank[i] + 1]

while i + k < n and j + k < n and txt[i + k] == txt[j + k]:

k += 1

lcp[rank[i]] = k

if k > 0:

k -= 1

return lcp

# A utility function to print lcp array

def print\_lcp\_array(txt, suffix\_arr, lcp):

print("LCP Array:")

print(lcp)

for i in range(len(lcp)):

suffix1 = txt[suffix\_arr[i]:]

suffix2 = txt[suffix\_arr[i+1]:] if i < len(lcp)-1 else None

message = f"lcp[{i}] = Longest Common Prefix of \"{suffix1}\" and \"{suffix2}\" = {lcp[i]}"

print(message)

# A utility function to find lcs using sa and lcp

def find\_lcs\_using\_sa\_lcp(s, sa, lcp):

max\_lcp = max(lcp)

lcs\_list = []

for i in range(0,len(s)-1):

if lcp[i] == max\_lcp:

lcs\_candidate = s[sa[i]:sa[i]+max\_lcp]

if lcs\_candidate not in lcs\_list:

lcs\_list.append(lcs\_candidate)

return lcs\_list

# A utility function to find lrs using sa and lcp

def find\_repeated\_substring(s, sa, lcp):

n = len(s)

result = ""

for i in range(0, n-1):

if lcp[i] >= 1:

length = lcp[i]

j = i + 1

while j < n and lcp[j] >= length:

j += 1

count = j - i + 1

if count >= 3:

repeated\_substring = s[sa[i]:sa[i]+length]

if len(repeated\_substring) > len(result):

result = repeated\_substring

return result

# A utility function to build inverse suffix array

def inverse\_suffix\_array(suffix\_array):

n = len(suffix\_array)

inverse\_array = [0] \* n

for i in range(n):

inverse\_array[suffix\_array[i]] = i

return inverse\_array

# A utility function to find shortest\_factors

def shortest\_factors\_using\_lgCandidat(T):

n=len(T)

TS=build\_suffix\_array(T,n)

HTR=build\_lcp\_table(T,TS)

ITS=inverse\_suffix\_array(TS)

resultat = []

# construire la table lgCandidat

# lgCandidat une table n + 1

lgCandidat1=[]

for i in range(0,n):

y=int(ITS[i])+1

if y>=len(HTR) :

y=int(ITS[i])

ss=HTR[ITS[i]]

dd=HTR[y]

lgCandidat1.insert(i, 1 + max(ss,dd))

else:

ss = HTR[ITS[i]]

dd = HTR[y]

lgCandidat1.insert(i, 1 + max(ss, dd))

# filtrer les minimums

i = 0

print("lgCandidat :",lgCandidat1)

#(lgCandidat1[n-1])

while (i + lgCandidat1[i]) <= n :

if lgCandidat1[i] <= lgCandidat1[i+1]:

resultat.append(T[i:i+lgCandidat1[i]])

i=i+1

return resultat

# A utility function to find super maximale repeats

def find\_supermaximal\_repeats(s, sa, lcp):

n = len(s)

max\_lcp = max(lcp)

supermaximal\_repeats = []

for i in range(n):

if lcp[i] >= max\_lcp:

lcp\_length = lcp[i]

j = i + 1

while j < n and lcp[j] >= lcp\_length:

j += 1

for k in range(i, j):

if sa[k] + lcp\_length < n and lcp[k] >= max\_lcp:

candidate\_repeat = s[sa[k]:sa[k]+lcp\_length]

if candidate\_repeat not in supermaximal\_repeats:

supermaximal\_repeats.append(candidate\_repeat)

return supermaximal\_repeats

# A utility function to find longest common factors between two texts

def longest\_common\_factor(txt1, txt2):

# Concatenate the two texts with a special character to distinguish their boundaries

txt = txt1 + '$' + txt2 + '#'

n = len(txt)

# Build the suffix array and LCP table for the concatenated text

suffix\_arr = build\_suffix\_array(txt, n)

htr = build\_lcp\_table(txt, suffix\_arr)

# Initialize the variables to store the maximum length and its starting position

max\_len = 0

start\_pos = -1

# Iterate over the LCP table to find the maximum length and its starting position

for i in range(n-1):

if suffix\_arr[i] < len(txt1) and suffix\_arr[i+1] > len(txt1):

if htr[i] > max\_len:

max\_len = htr[i]

start\_pos = suffix\_arr[i]

# Return the longest common factor if it exists, otherwise return an empty string

if max\_len > 0:

return txt[start\_pos:start\_pos+max\_len]

else:

return ''

# Driver program to test above functions

if \_\_name\_\_ == "\_\_main\_\_":

print("--------------------Suffix array operations---------------------")

print("Veuillez choisir le traitement a faire :")

print("1- Table des suffixes")

print("2- Recherche Exacte d'un motif")

print("3- Table HTR")

print("4- Le plus long facteur répété")

print("5- Les facteurs qui se répètent au moins 3 fois")

print("6- Table ITS")

print("7- Les plus courts facteurs uniques avec la table des lgCandidat")

print("8- Les répétitions super-maximales")

print("9- Le plus long facteur entre deux textes")

print("10- Exit")

print("----------------------------------------------------------------")

choix = int(input("choix : "))

while choix > 10:

color = fg('red')

print(color + "donner une valeur entre 1 et 10" + attr('reset'))

choix = int(input("choix : "))

continu = True

txt = "CAGAGACGGCGGAGAAATCGATTGCAACTT"

while(continu):

if(choix == 1):

start\_time = time.time()

n = len(txt)

print("Suffix array for :", txt)

suffix\_arr = build\_suffix\_array(txt, n)

print\_suffix\_array(txt, suffix\_arr)

end\_time = time.time()

execution\_time = end\_time - start\_time

print("Execution time:", execution\_time, "ms")

if(choix == 2):

start\_time = time.time()

n = len(txt)

suffix\_arr = build\_suffix\_array(txt, n)

motif = "GCAACTT"

mat = search(motif,txt,suffix\_arr,n)

print("Les indices de début du motif trouvé:",mat)

end\_time = time.time()

execution\_time = end\_time - start\_time

print("Execution time:", execution\_time, "ms")

if(choix == 3):

start\_time = time.time()

n = len(txt)

suffix\_arr = build\_suffix\_array(txt, n)

print("HTR array for :", txt)

lcp = build\_lcp\_table(txt,suffix\_arr)

print\_lcp\_array(txt,suffix\_arr,lcp)

end\_time = time.time()

execution\_time = end\_time - start\_time

print("Execution time:", execution\_time, "ms")

if(choix == 4):

start\_time = time.time()

n = len(txt)

suffix\_arr = build\_suffix\_array(txt, n)

lcp = build\_lcp\_table(txt,suffix\_arr)

lcs = find\_lcs\_using\_sa\_lcp(txt, suffix\_arr, lcp)

print("Le plus long facteur répété est : ",lcs)

end\_time = time.time()

execution\_time = end\_time - start\_time

print("Execution time:", execution\_time, "ms")

if(choix == 5):

start\_time = time.time()

n = len(txt)

suffix\_arr = build\_suffix\_array(txt, n)

lcp = build\_lcp\_table(txt,suffix\_arr)

rss = find\_repeated\_substring(txt, suffix\_arr, lcp)

print("Les plus long facteurs répétés au moins 3 fois sont :",rss)

end\_time = time.time()

execution\_time = end\_time - start\_time

print("Execution time:", execution\_time, "ms")

if(choix == 6):

start\_time = time.time()

n = len(txt)

suffix\_arr = build\_suffix\_array(txt, n)

its = inverse\_suffix\_array(suffix\_arr)

print("ITS : ",its)

end\_time = time.time()

execution\_time = end\_time - start\_time

print("Execution time:", execution\_time, "ms")

if(choix == 7):

start\_time = time.time()

lgc = shortest\_factors\_using\_lgCandidat(txt)

print("Les plus courts facteurs uniques sont :",lgc)

end\_time = time.time()

execution\_time = end\_time - start\_time

print("Execution time:", execution\_time, "ms")

if(choix == 8):

start\_time = time.time()

n = len(txt)

suffix\_arr = build\_suffix\_array(txt, n)

lcp = build\_lcp\_table(txt,suffix\_arr)

ssr = find\_supermaximal\_repeats(txt, suffix\_arr, lcp)

print("Les répétitions super-maximales sont :",ssr)

end\_time = time.time()

execution\_time = end\_time - start\_time

print("Execution time:", execution\_time, "ms")

if(choix == 9):

start\_time = time.time()

txt1 = "ACATTCTAGGATTACCAAGCTCCTGCAGAT"

txt2 = "TATAAGCCCTAGACTAGTTATTGTCTGGGA"

lcf = longest\_common\_factor(txt1, txt2)

print(lcf)

end\_time = time.time()

execution\_time = end\_time - start\_time

print("Execution time:", execution\_time, "ms")

if(choix == 10):

color = fg('green')

print(color + 'Thank you very much sir :)' + attr('reset'))

exit()

con = input("Voulez-vous continuer ? [yes/no] : ")

if(con == 'yes'):

continu = True

print("--------------------Suffix array operations---------------------")

print("Veuillez choisir le traitement a faire :")

print("1- Table des suffixes")

print("2- Recherche Exacte d'un motif")

print("3- Table HTR")

print("4- Le plus long facteur répété")

print("5- Les facteurs qui se répètent au moins 3 fois")

print("6- Table ITS")

print("7- Les plus courts facteurs uniques avec la table des lgCandidat")

print("8- Les répétitions super-maximales")

print("9- Le plus long facteur entre deux textes")

print("10- Exit")

print("----------------------------------------------------------------")

choix = int(input("choix : "))

while choix > 10:

color = fg('red')

print(color + "donner une valeur entre 1 et 10" + attr('reset'))

choix = int(input("choix : "))

else:

continu = False

color = fg('green')

print(color + 'Thank you very much sir :)' + attr('reset'))

#---------------------------------------------------------------------------------------

"""

--------------pour calculer le temps d'exécution de n'importe quel traitement-----------

import time

start\_time = time.time()

<votre traitement>

end\_time = time.time()

execution\_time = end\_time - start\_time

print("Execution time:", execution\_time, "ms")

----------------------------------------------------------------------------------------

--------------pour trier les suffixes selon l'ordre lexicographique---------------------

import functools

suffixes.sort(key=functools.cmp\_to\_key(cmp))

-----------------------------------------------------------------------------------------

--------------pour colorer les messages--------------------------------------------------

from colored import fg, attr

color = fg('red')

print(color + "donner une valeur entre 1 et 10" + attr('reset'))

color = fg('green')

print(color + 'Thank you very much sir :)' + attr('reset'))

-----------------------------------------------------------------------------------------

"""