МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное

образовательное учреждение

высшего образования

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №13

**по дисциплине**

**“ Алгоритмы построение и анализ”**

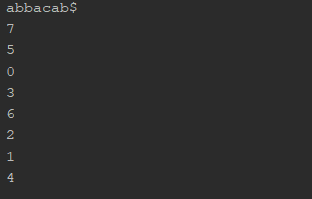
Выполнил: студент гр. ФИб-3302-51-00 ё \_\_\_\_\_\_\_\_\_\_\_

Проверил: доцент кафедры ПМиИ Разова Е. В. \_\_\_\_\_\_\_\_\_\_\_

Киров 2020

**Задание 1.**

Реализовать алгоритм Карккайнена-Сандерса.



fulltext = "abbacab"  
  
def create\_sa(text):  
 # построить s12  
 s12\_unsorted = create\_s12(text)  
  
 # отсортировать s12 суффиксы  
 s12\_with\_ranks = sort(text, s12\_unsorted)  
 s12\_ranks = [value[0] for value in s12\_with\_ranks]  
  
 if len(set(s12\_ranks)) == len(s12\_ranks):  
 s12 = [value[1] for value in sorted(s12\_with\_ranks, key=lambda x: x[0])]  
 else:  
 sa\_s12 = create\_sa(s12\_ranks)  
 s12 = [s12\_with\_ranks[idx][1] for idx in sa\_s12]  
  
 # отсортировать s0 суффиксы  
 s0 = create\_s0(text, s12)  
  
 # создать массив для слияния и слить их  
 inverse\_sa = create\_inverse\_sa(len(text), s12)  
 merged\_sa = merge\_sa(text, s0, s12, inverse\_sa)  
  
 for xx in range(len(merged\_sa)):  
 print(merged\_sa[xx])  
 return merged\_sa  
  
  
# создать массив s12  
def create\_s12(text):  
 s1 = []  
 s2 = []  
  
 # заполнение массивов триплетов  
 for idx in range(len(text)):  
 if (idx % 3) == 1:  
 s1.append(idx)  
 elif (idx % 3) == 2:  
 s2.append(idx)  
 return s1 + s2  
  
  
# сортировка s12 суффиксов  
def sort(text, sa, s0=False):  
 triplet\_idx = 2  
 if s0:  
 triplet\_idx = 0  
 sorted\_sa = sa  
  
 # сортировка массива sa  
 for i in range(triplet\_idx, -1, -1):  
 buckets = dict()  
  
 for text\_idx in sorted\_sa:  
 triplet = get\_triplet(text, text\_idx)  
  
 checked\_i = i if i < len(triplet) else len(triplet) - 1  
  
 if triplet[checked\_i] not in buckets:  
 buckets[triplet[checked\_i]] = []  
  
 buckets[triplet[checked\_i]].append(text\_idx)  
  
 sorted\_sa = []  
  
 for key in sorted(buckets.keys()):  
 sorted\_sa += buckets[key]  
  
 # получим ранги  
 ranks = dict()  
 rank = 1  
  
 for text\_idx in sorted\_sa:  
 triplet = ''.join([str(x) for x in get\_triplet(text, text\_idx)])  
 if triplet not in ranks:  
 ranks[triplet] = rank  
 rank += 1  
  
 # совмещаем s12 и ранги  
 s12\_with\_ranks = [(ranks.get(''.join([str(x) for x in get\_triplet(text, idx\_original)])), idx\_original) for idx\_original in sa]  
  
 return s12\_with\_ranks  
  
  
def get\_triplet(text, idx):  
 triplet = []  
 for text\_idx in range(idx, min(idx+3, len(text))):  
 triplet.append(text[text\_idx])  
  
 while len(triplet) < 3 and isinstance(triplet, str):  
 triplet.append("$")  
  
 return triplet  
  
  
# сортируем s0 суффиксы  
def create\_s0(text, s12):  
 s0\_unsorted = []  
  
 for idx\_text in s12:  
 if ((idx\_text - 1) % 3) == 0:  
 s0\_unsorted.append(idx\_text - 1)  
  
 if len(text) % 3 == 1:  
 s0\_unsorted.insert(0, len(text) - 1)  
  
 # сортируем s0 - но только по первой букве  
 s0\_with\_ranks = sort(text, s0\_unsorted, s0=True)  
  
 # извлекаем значения для s0  
 s0 = [x[1] for x in sorted(s0\_with\_ranks, key=lambda x: x[0])]  
 return s0  
  
"""  
слить s0 и s12  
"""  
  
# создаем обратный sa для слияния  
def create\_inverse\_sa(len\_text, sa):  
 inverse\_sa = [-1] \* len\_text  
  
 for sa\_idx in range(len(sa)):  
 inverse\_sa[sa[sa\_idx]] = sa\_idx  
  
 return inverse\_sa  
  
  
def merge\_sa(text, s0, s12, inverse\_sa):  
 sa = []  
 idx\_s0 = 0  
 idx\_s12 = 0  
  
 while (idx\_s0 + idx\_s12) < (len(s0) + len(s12)):  
 if idx\_s0 >= len(s0):  
 sa.append(s12[idx\_s12])  
 idx\_s12 += 1  
 continue  
  
 if idx\_s12 >= len(s12):  
 sa.append(s0[idx\_s0])  
 idx\_s0 += 1  
 continue  
  
 current\_s0 = s0[idx\_s0]  
 current\_s12 = s12[idx\_s12]  
  
 # проверяем первый символ  
 if text[current\_s0] > text[current\_s12]:  
 sa.append(current\_s12)  
 idx\_s12 += 1  
 continue  
  
 if text[current\_s0] < text[current\_s12]:  
 sa.append(current\_s0)  
 idx\_s0 += 1  
 continue  
  
 if text[current\_s0] == text[current\_s12]:  
  
 i = 1  
 while True:  
 # проверяем с помощью inverse\_sa  
 if (current\_s0 + i) % 3 == 0 or (current\_s12 + i) % 3 == 0:  
 # проверяем наличие второго символа  
 if text[current\_s0 + i] > text[current\_s12 + i]:  
 sa.append(current\_s12)  
 idx\_s12 += i  
 break  
  
 if text[current\_s0 + i] < text[current\_s12 + i]:  
 sa.append(current\_s0)  
 idx\_s0 += i  
 break  
  
 if text[current\_s0 + i] == text[current\_s12 + i]:  
 i += 1  
 continue  
 continue  
  
 if inverse\_sa[current\_s0 + i] > inverse\_sa[current\_s12 + i]:  
 sa.append(current\_s12)  
 idx\_s12 += 1  
 break  
  
 if inverse\_sa[current\_s0 + i] < inverse\_sa[current\_s12 + i]:  
 sa.append(current\_s0)  
 idx\_s0 += 1  
 break  
  
  
 return sa  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 create\_sa(fulltext)