Отчет по лабораторной работе номер 3

По теме: Исследование алгоритма перестановки карточек для достижения чередующейся цветовой последовательности.

**Цель работы:**

Имеется n черных и белых карточек, сложенных в стопку. Карточки раскладываются на стол в одну линию следующим образом: первая кладется на стол, вторая под низ стопки, третья- на стол, четвертая - под низ стопки и т.д., пока все карточки не будут выложены на стол. Каким должно быть исходное расположение карточек в стопке, чтобы разложенные на столе карточки чередовались по цвету: белая, черная, белая, черная и т.д.

**Код программы:**

import time

from collections import deque

#Выполнил Шакула Дмитрий Андреевич 090301-ПОВа-о24

# A) Через массив (Python list)

def initial\_deck\_array(n):

# строим желаемую последовательность на столе: W, B, W, B, …

S = ['W' if i % 2 == 0 else 'B' for i in range(n)]

D = []

# обратный процесс для восстановления исходной колоды

for c in reversed(S):

if D:

# переставляем нижнюю карту наверх

D.insert(0, D.pop())

# вставляем текущую карту наверх

D.insert(0, c)

return D

# B) Через собственный связанный список

class Node:

def \_\_init\_\_(self, val):

self.val = val

self.next = None

class LinkedList:

def \_\_init\_\_(self):

self.head = None

self.tail = None

def prepend(self, val):

node = Node(val)

if not self.head:

self.head = self.tail = node

else:

node.next = self.head

self.head = node

def pop\_tail(self):

if not self.head:

raise IndexError("pop from empty list")

if self.head is self.tail:

val = self.head.val

self.head = self.tail = None

return val

cur = self.head

# ищем предпоследний

while cur.next is not self.tail:

cur = cur.next

val = self.tail.val

cur.next = None

self.tail = cur

return val

def to\_list(self):

res = []

cur = self.head

while cur:

res.append(cur.val)

cur = cur.next

return res

def initial\_deck\_linked(n):

S = ['W' if i % 2 == 0 else 'B' for i in range(n)]

D = LinkedList()

for c in reversed(S):

if D.head:

x = D.pop\_tail()

D.prepend(x)

D.prepend(c)

return D.to\_list()

# C) Через стандартную библиотеку (collections.deque)

def initial\_deck\_deque(n):

S = ['W' if i % 2 == 0 else 'B' for i in range(n)]

D = deque()

for c in reversed(S):

if D:

D.appendleft(D.pop())

D.appendleft(c)

return list(D)

# Симуляция выкладывания: верхняя на стол, следующая под низ

def deal(deck):

D = deque(deck)

result = []

while D:

result.append(D.popleft())

if D:

D.append(D.popleft())

return result

# Проверка чередования W, B, W, B, …

def is\_alternating(seq):

return all(seq[i] == ('W' if i % 2 == 0 else 'B') for i in range(len(seq)))

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

try:

n = int(input("Введите n (количество карточек): ").strip())

if n < 1:

raise ValueError

except ValueError:

print("Ошибка: нужно целое число n ≥ 1.")

exit(1)

implementations = [

("Array/list", initial\_deck\_array),

("LinkedList", initial\_deck\_linked),

("Deque", initial\_deck\_deque),

]

print(f"\nСравнение производительности при n = {n}\n")

print(f"{'Имплементация':<15} {'Build(s)':>10} {'Deal(s)':>10} {'Correct':>10}")

print("-" \* 50)

for name, func in implementations:

# измеряем время построения

t0 = time.time()

deck = func(n)

t1 = time.time()

# измеряем время выкладывания

dealt\_seq = deal(deck)

t2 = time.time()

build\_time = t1 - t0

deal\_time = t2 - t1

correct = is\_alternating(dealt\_seq)

print(f"{name:<15} {build\_time:10.6f} {deal\_time:10.6f} {str(correct):>10}")

print("\nПример исходной колоды (сверху→вниз) для Array/list:")

print(initial\_deck\_array(n))

print("\nПервые элементы выложенной последовательности:")

print(deal(initial\_deck\_array(n))[:min(20,n)])

print("#Выполнил Шакула Дмитрий Андреевич 090301-ПОВа-о24")

**Результаты выполнения:**

