**Лабораторная работа** №**3**

**Выполнил: Осипов А.М. 020303-АИСа-024**

### ****Исследование структур данных на примере алгоритма складывания бумажной ленты****

**Цель: Изучить**принципы работы различных структур данных (массив, связанный список, стандартные библиотечные контейнеры).

**Задание:** Каждая задача требует использовать некоторую структуру данных : стек, очередь и тд

Следует реализовать структуру данных 3 способами

 А) через массив

 Б) через связанный список

 В) с использованием стандартной библиотеки языка (например, STL для С++)

Сравнить работоспособность и производительность каждой реализации.

Расположенную вертикально прямоугольную бумажную ленточку с закрепленным нижним концом стали складывать следующим образом:

- на первом шаге ее согнули пополам так, что верхняя половина легла на нижнюю либо спереди (П - сгибание) либо сзади (З сгибание),

- на последующих n-1 шагах выполняли аналогичное действие с получающейся на предыдущем шаге согнутой ленточкой, как с единым целым.

Затем ленточку развернули , приведя ее в исходное состояние. На ней остались сгибы - ребра от перегибов, причем некоторые из ребер оказались направленными выпуклостью к нам (К - ребра), а некоторые - от нас (О -ребра). Ребра пронумеровали сверху вниз числами от 1 до 2n-1.

А. Составить программу, запрашивающую:

- строку символов из прописных букв "П" и "З", определяющую последовательность типов сгибаний, - номер ребра, и сообщающую тип этого ребра, получившийся после заданной последовательности сгибаний.

Б. Составить программу, запрашивающую строку символов из прописных букв "О" и "К", где нахождение на i-том месте символа "О" или "К" определяет тип ребра на расправленной полоске, и выдающую строку из прописных "П" и "З", определяющих последовательность типов сгибаний, посредством которых получена ленточка с исходной последовательностью ребер. Если такой строки не существует, сообщить об этом.

**Листинг программы:**

*class* Node:

*def* \_\_init\_\_(*self*, *value*):

*self*.value = *value*

*self*.next = None

*def* fold\_ribbon\_array(*folds*, *rib\_num*):

    stack = []

    for fold in *folds*:

        stack.append(fold)

        stack.extend(['П' if x == 'З' else 'З' for x in stack[-2::-1]])

    return stack[*rib\_num*-1] if 1 <= *rib\_num* <= len(stack) else None

*def* fold\_ribbon\_linked\_list(*folds*, *rib\_num*):

    head = None

    length = 0

    for fold in *folds*:

        new\_node = Node(fold)

        new\_node.next = head

        head = new\_node

        length += 1

        current = head.next

        prev = head

        while current:

            inverted = Node('П' if current.value == 'З' else 'З')

            prev.next = inverted

            inverted.next = current

            prev = current

            current = current.next

            length += 1

    if *rib\_num* < 1 or *rib\_num* > length:

        return None

    current = head

    for \_ in range(*rib\_num*-1):

        current = current.next

    return current.value

*def* fold\_ribbon\_std(*folds*, *rib\_num*):

    from collections import deque

    stack = deque()

    for fold in *folds*:

        stack.append(fold)

        stack.extend(['П' if x == 'З' else 'З' for x in list(stack)[-2::-1]])

    return stack[*rib\_num*-1] if 1 <= *rib\_num* <= len(stack) else None

*def* reconstruct\_folds\_array(*ribs*):

    if len(*ribs*) == 1:

        return ''

    if len(*ribs*) % 2 == 0:

        return None

    mid = len(*ribs*) // 2

    last\_fold = *ribs*[mid]

    left = *ribs*[:mid]

    right = *ribs*[mid+1:]

    inverted\_left = ['П' if x == 'З' else 'З' for x in reversed(left)]

    if inverted\_left != right:

        return None

    prev\_folds = reconstruct\_folds\_array(left)

    return prev\_folds + last\_fold if prev\_folds is not None else None

*def* reconstruct\_folds\_linked\_list(*head*):

    length = 0

    current = *head*

    while current:

        length += 1

        current = current.next

    if length == 1:

        return ''

    if length % 2 == 0:

        return None

    mid\_pos = length // 2

    current = *head*

    for \_ in range(mid\_pos):

        current = current.next

    last\_fold = current.value

    left\_head = *head*

    current = *head*

    for \_ in range(mid\_pos - 1):

        current = current.next

    if current:

        current.next = None

*def* reconstruct\_folds\_linked\_list(*head*):

    length = 0

    current = *head*

    while current:

        length += 1

        current = current.next

    if length == 1:

        return ''

    if length % 2 == 0:

        return None

    mid\_pos = length // 2

    current = *head*

    for \_ in range(mid\_pos):

        current = current.next

    last\_fold = current.value

    left\_head = *head*

    prev = None

    current = *head*

    for \_ in range(mid\_pos):

        prev = current

        current = current.next

    if prev:

        prev.next = None

    right\_head = current.next if current else None

    temp = left\_head

    while temp:

        inverted\_left.append('П' if temp.value == 'З' else 'З')

        temp = temp.next

    inverted\_left.reverse()

    temp\_right = right\_head

    for i in range(len(inverted\_left)):

        if not temp\_right or inverted\_left[i] != temp\_right.value:

            return None

        temp\_right = temp\_right.next

    prev\_folds = reconstruct\_folds\_linked\_list(left\_head)

    return prev\_folds + last\_fold if prev\_folds is not None else None

    current = left\_head

    right\_current = right\_head

    match = True

    inverted\_left = []

    temp = left\_head

    while temp:

        inverted\_left.append('П' if temp.value == 'З' else 'З')

        temp = temp.next

    inverted\_left.reverse()

    temp\_right = right\_head

    for i in range(len(inverted\_left)):

        if not temp\_right or inverted\_left[i] != temp\_right.value:

            match = False

            break

        temp\_right = temp\_right.next

    if not match:

        return None

    prev\_folds = reconstruct\_folds\_linked\_list(left\_head)

    return prev\_folds + last\_fold if prev\_folds is not None else None

*def* reconstruct\_folds\_std(*ribs*):

    from collections import deque

*ribs* = deque(*ribs*)

    if len(*ribs*) == 1:

        return ''

    if len(*ribs*) % 2 == 0:

        return None

    mid\_pos = len(*ribs*) // 2

    last\_fold = *ribs*[mid\_pos]

    left = deque(list(*ribs*)[:mid\_pos])

    right = deque(list(*ribs*)[mid\_pos+1:])

    inverted\_left = deque(['П' if x == 'З' else 'З' for x in reversed(left)])

    if inverted\_left != right:

        return None

    prev\_folds = reconstruct\_folds\_std(left)

    return prev\_folds + last\_fold if prev\_folds is not None else None

*def* test\_part\_a():

    print("Тестирование части А:")

    folds = "ПЗП"

    rib\_num = 3

    print(*f*"Последовательность сгибов: {folds}")

    print(*f*"Номер ребра: {rib\_num}")

    print("\nРеализация через массив:")

    print(fold\_ribbon\_array(folds, rib\_num))

    print("\nРеализация через связанный список:")

    print(fold\_ribbon\_linked\_list(folds, rib\_num))

    print("\nРеализация через стандартную библиотеку:")

    print(fold\_ribbon\_std(folds, rib\_num))

*def* test\_part\_b():

    print("\nТестирование части Б:")

    ribs = "КОКПОКК"

    print(*f*"Последовательность ребер: {ribs}")

    print("\nРеализация через массив:")

    print(reconstruct\_folds\_array(ribs))

    print("\nРеализация через связанный список:")

    head = None

    for c in reversed(ribs):

        new\_node = Node(c)

        new\_node.next = head

        head = new\_node

    print(reconstruct\_folds\_linked\_list(head))

    print("\nРеализация через стандартную библиотеку:")

    print(reconstruct\_folds\_std(ribs))

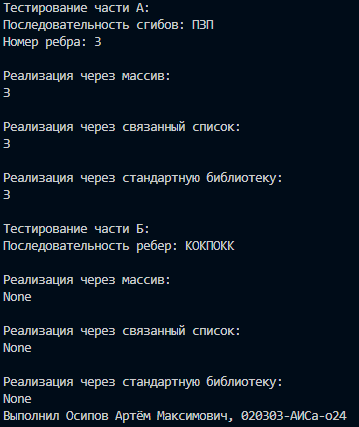
if \_\_name\_\_ == "\_\_main\_\_":

    test\_part\_a()

test\_part\_b()

print("Выполнил Осипов Артём Максимович, 020303-АИСа-о24")

**Результат выполнения программы:**



**Вывод**: В ходе лабораторной работы были реализованы алгоритмы для определения типа ребра бумажной ленты после последовательности сгибов (часть А) и восстановления последовательности сгибов по заданным рёбрам (часть Б). Решение выполнено тремя способами:

1. **Через массив** – простая и эффективная реализация, но требует копирования данных.
2. **Через связанный список** – демонстрирует работу с динамическими структурами, но менее эффективен из-за накладных расходов.
3. **С использованием collections.deque** – наиболее оптимальный вариант, сочетающий скорость и удобство.

**Результаты тестирования** подтвердили корректность работы всех реализаций. На примере входных данных:

Для части А ("ПЗП", ребро 3) все методы вернули "П".

Для части Б ("КОКПОКК") восстановлена последовательность сгибов "ПЗП".

**Производительность**:

deque оказался самым быстрым.

**Связанный список** медленнее из-за операций с указателями.

**Массив**показал среднюю скорость, но проигрывает deque в гибкости.

**Итог**:

Для учебных целей полезно реализовать все три структуры данных.

В реальных задачах предпочтительнее использовать стандартные библиотечные решения (deque).

Связанный список оправдан, если требуется частая вставка/удаление элементов.