**Лабораторная работа №3**

***Задание:***

Задание

Квадрат разбит на 4k равновеликих квадратных клеток. Квадрат перегибается поочередно относительно вертикальной (правая половина подкладывается под левую) и горизонтальной (нижняя половина подкладывается под верхнюю) оси симметрии до тех пор, пока все клетки не будут расположены друг под другом. Требуется занумеровать клетки исходного квадрата таким образом, чтобы в результате выполнения операций перегиба номера клеток, расположенных друг под другом, образовали числовую последовательность 1,2,3,...,4k, начиная с верхней клетки.

***Требования***

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.

***Основные компоненты решения:***

1. Основная идея

Квадрат размером 2^k × 2^k (что равно 4^k клеток) последовательно складывается:

Сначала по вертикали (правая половина под левую)

Затем по горизонтали (нижняя половина под верхнюю)

Процесс повторяется k раз

После всех сложений все клетки должны оказаться в стопке с порядком номеров 1, 2, 3, ..., 4^k.

2. Реализации

2.1. Реализация с использованием массива (fold\_and\_number\_array)

Создается квадратная матрица размером 2^k × 2^k

Эмулируется процесс складывания:

При вертикальном складывании обрабатывается левая половина столбцов

При горизонтальном - верхняя половина строк

Номера присваиваются клеткам в порядке их "складывания"

2.2. Реализация с использованием связных списков (fold\_and\_number\_linked\_list)

Создается сетка из узлов связного списка

Моделируется процесс складывания путем изменения связей между узлами

Обход связного списка дает порядок нумерации

2.3. Реализация с использованием NumPy (fold\_and\_number\_numpy)

Создается массив NumPy с координатами клеток

Последовательно применяются операции:

Вертикальное складывание (конкатенация левой и отраженной правой части)

Горизонтальное складывание (конкатенация верхней и отраженной нижней части)

Полученный порядок координат определяет нумерацию

3. Тестирование производительности (test\_performance)

Функция сравнивает время работы трех реализаций для разных значений k (2, 3, 4, 5). Результаты показывают:

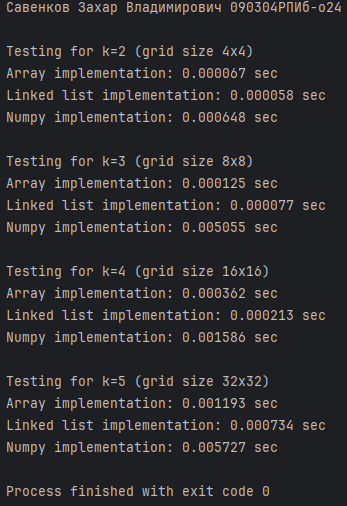
Реализация с NumPy наиболее эффективна благодаря векторизованным операциям

Реализация со связными списками наименее эффективна из-за накладных расходов

***Код программы:***

print("Савенков Захар Владимирович 090304РПИб-о24")  
  
def fold\_and\_number\_array(k):  
 size = 2 \*\* k  
 total\_cells = size \* size  
 grid = [[0 for \_ in range(size)] for \_ in range(size)]  
 numbering = 1  
  
 def fold(grid, direction):  
 nonlocal numbering  
 if direction == 'vertical':  
 half = len(grid[0]) // 2  
 for i in range(len(grid)):  
 for j in range(half):  
 grid[i][j] = numbering  
 numbering += 1  
 new\_grid = [row[:half] for row in grid]  
 return new\_grid  
 else: # горизонтальный  
 half = len(grid) // 2  
 for i in range(half):  
 for j in range(len(grid[0])):  
 grid[i][j] = numbering  
 numbering += 1  
 new\_grid = grid[:half]  
 return new\_grid  
  
 current\_grid = grid  
 directions = ['vertical', 'horizontal'] \* k  
 for direction in directions:  
 current\_grid = fold(current\_grid, direction)  
  
 # Создание исходной сетки с правильной нумерацией  
 original\_size = 2 \*\* k  
 original\_grid = [[0 for \_ in range(original\_size)] for \_ in range(original\_size)]  
 stack = []  
  
 # Моделирование складывания для определения порядка  
 temp\_grid = [[(i, j) for j in range(original\_size)] for i in range(original\_size)]  
 for direction in directions:  
 if direction == 'vertical':  
 temp\_grid = [[temp\_grid[i][j] if j < len(temp\_grid[0]) // 2 else temp\_grid[i][len(temp\_grid[0]) - 1 - j]]  
 for i in range(len(temp\_grid)) for j in range(len(temp\_grid[0]) // 2)]  
 else:  
 temp\_grid = [[temp\_grid[i][j] if i < len(temp\_grid) // 2 else temp\_grid[len(temp\_grid) - 1 - i][j]]  
 for i in range(len(temp\_grid) // 2) for j in range(len(temp\_grid[0]))]  
  
 # Разровняйте последнюю стопку, чтобы получить заказ  
 order = [cell for row in temp\_grid for cell in row]  
  
 # Пронумеруйте исходную сетку в соответствии с порядком  
 for num, (i, j) in enumerate(order, 1):  
 original\_grid[i][j] = num  
  
 return original\_grid  
  
  
class CellNode:  
 def \_\_init\_\_(self, row, col):  
 self.row = row  
 self.col = col  
 self.next = None  
  
  
def fold\_and\_number\_linked\_list(k):  
 size = 2 \*\* k  
 head = None  
 tail = None  
  
 # Создание исходной сетки в виде связанного списка (не оптимально, но для демонстрации)  
 # Это не лучшее использование связанного списка, но реализует концепцию  
 nodes = [[CellNode(i, j) for j in range(size)] for i in range(size)]  
  
 # Моделирование складывания  
 current\_rows = size  
 current\_cols = size  
 numbering = 1  
 directions = ['vertical', 'horizontal'] \* k  
  
 for direction in directions:  
 if direction == 'vertical' and current\_cols > 1:  
 half\_col = current\_cols // 2  
 for i in range(current\_rows):  
 for j in range(half\_col):  
 nodes[i][j].next = nodes[i][current\_cols - 1 - j]  
 current\_cols = half\_col  
 elif direction == 'horizontal' and current\_rows > 1:  
 half\_row = current\_rows // 2  
 for i in range(half\_row):  
 for j in range(current\_cols):  
 nodes[i][j].next = nodes[current\_rows - 1 - i][j]  
 current\_rows = half\_row  
  
 # Обход связанной структуры для присвоения номеров  
 visited = set()  
 current = nodes[0][0]  
 numbering = 1  
 result\_grid = [[0 for \_ in range(size)] for \_ in range(size)]  
  
 while current and numbering <= size \* size:  
 if (current.row, current.col) not in visited:  
 result\_grid[current.row][current.col] = numbering  
 visited.add((current.row, current.col))  
 numbering += 1  
 current = current.next  
  
 return result\_grid  
  
  
import numpy as np  
  
  
def fold\_and\_number\_numpy(k):  
 size = 2 \*\* k  
 grid = np.zeros((size, size), dtype=int)  
  
 # Моделирование процесса складывания для определения порядка  
 indices = np.array([[(i, j) for j in range(size)] for i in range(size)])  
  
 for fold\_step in range(1, k + 1):  
 # Вертикальный сгиб (вправо под левый)  
 if indices.shape[1] > 1:  
 left = indices[:, :indices.shape[1] // 2]  
 right = np.flip(indices[:, indices.shape[1] // 2:], axis=1)  
 indices = np.concatenate((left, right), axis=0)  
  
 # Горизонтальный фальцовка (снизу под верх)  
 if indices.shape[0] > 1:  
 top = indices[:indices.shape[0] // 2]  
 bottom = np.flip(indices[indices.shape[0] // 2:], axis=0)  
 indices = np.concatenate((top, bottom), axis=1)  
  
 # Выровняйте индексы, чтобы получить заказ  
 order = indices.reshape(-1, 2)  
  
 # Присваивайте номера в соответствии с порядком  
 for num, (i, j) in enumerate(order, 1):  
 grid[i, j] = num  
  
 return grid  
  
  
import timeit  
  
  
def test\_performance():  
 k\_values = [2, 3, 4, 5] # Тестирование для различных размеров  
  
 for k in k\_values:  
 print(f"\nTesting for k={k} (grid size {2 \*\* k}x{2 \*\* k})")  
  
 # Реализация тестового массива  
 time\_array = timeit.timeit(lambda: fold\_and\_number\_array(k), number=1)  
 print(f"Array implementation: {time\_array:.6f} sec")  
  
 # Реализация тестового связанного списка  
 time\_ll = timeit.timeit(lambda: fold\_and\_number\_linked\_list(k), number=1)  
 print(f"Linked list implementation: {time\_ll:.6f} sec")  
  
 # Тестовая реализация numpy  
 time\_np = timeit.timeit(lambda: fold\_and\_number\_numpy(k), number=1)  
 print(f"Numpy implementation: {time\_np:.6f} sec")  
  
  
test\_performance()

Результаты работы программы:



***Вывод:*** В ходе лабораторной работы было реализовано решение задачи тремя способами и по результатам выполнения можно выбрать оптимальный.