Лабораторная работа №2

Выполнила: Ковалевская А.М. 020303-АИСа-024

Нескалярные типы данных: списки

Цель: исследование особенностей реализации и эффективности операций с нескалярными структурами данных (списками) в различных представлениях: массивы, связные списки и стандартные библиотечные реализации.

Задание:

Форма тела задана матрицей A размерности M x N. Элементы матрицы - натуральные числа. Элемент A (i,j) соответствует высоте горизонтальной квадратной площадки размера 1 x 1 относительно нижнего основания.Нижнее основание формы горизонтально.

Определить объем невытекшей воды, если

a) Тело опускается полностью в воду, затем поднимается;

После подъема вода останется только во внутреннем углублении, над элементом A(2,2)=1.

Объем воды равен 1.

b) В позицию (i_0,j_0) выливается объем воды V.

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

Следует реализовать структуру данных 3 способами

A) через массив

Б) через связанный список

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

Сравнить работоспособность и производительность каждой реализации.

Листинг программы:

```
import heapq
from collections import deque
import time
def trap_rain_water(height_map):
    """Расчет объема воды после погружения (задача а)"""
    if not height_map or not height_map[0]:
        return 0

    M, N = len(height_map), len(height_map[0])
    visited = [[False]*N for _ in range(M)]
    heap = []

    for i in range(M):
        for j in [0, N-1]:
            heapq.heappush(heap, (height_map[i][j], i, j))
            visited[i][j] = True
    for j in range(N):
```

```
    for i in [0, M-1]:
        if not visited[i][j]:
            heapq.heappush(heap, (height_map[i][j], i, j))
            visited[i][j] = True

    water = 0
    directions = [(-1,0), (1,0), (0,-1), (0,1)]
    max_height = 0

    while heap:
        height, x, y = heapq.heappop(heap)
        max_height = max(max_height, height)

        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if 0 <= nx < M and 0 <= ny < N and not visited[nx][ny]:
                visited[nx][ny] = True
                if height_map[nx][ny] < max_height:
                    water += max_height - height_map[nx][ny]
                heapq.heappush(heap, (height_map[nx][ny], nx, ny))

    return water

def pour_water(matrix, i0, j0, volume):
    """Имитация налива воды в конкретную позицию (задача b)"""
    M, N = len(matrix), len(matrix[0])
    water = [[0]*N for _ in range(M)]
    water[i0][j0] += volume
    q = deque()
    q.append((i0, j0))
    visited = [[False]*N for _ in range(M)]
    visited[i0][j0] = True

    directions = [(-1,0), (1,0), (0,-1), (0,1)]

    while q:
        i, j = q.popleft()
        current_total = matrix[i][j] + water[i][j]
        min_neighbors = []
        min_height = float('inf')

        for dx, dy in directions:
            ni, nj = i + dx, j + dy
            if 0 <= ni < M and 0 <= nj < N:
                total = matrix[ni][nj] + water[ni][nj]
                if total < current_total:
                    if total < min_height:
                        min_height = total
                        min_neighbors = [(ni, nj)]
                    elif total == min_height:
                        min_neighbors.append((ni, nj))

        if min_neighbors:
            transfer = min((current_total - min_height), water[i][j]) / len(min_neighbors)
            water[i][j] -= transfer * len(min_neighbors)

            for ni, nj in min_neighbors:
                water[ni][nj] += transfer
                if not visited[ni][nj]:
                    q.append((ni, nj))
                    visited[ni][nj] = True

    return sum(sum(row) for row in water)
```

```
def main():
    print("=== ТЕСТИРОВАНИЕ РЕШЕНИЙ ===")

    # Тест задачи a
    test_case_a = [
        [1, 4, 3],
        [2, 1, 2],
        [3, 2, 1]
    ]

    print("\nТест задачи a (погружение в воду):")
    start = time.time()
    result_a = trap_rain_water(test_case_a)
    print(f"Рассчитанный объем воды: {result_a}")
    print(f"Время выполнения: {time.time() - start:.6f} сек")

    # Тест задачи b
    test_case_b = [
        [1, 1, 1],
        [1, 0, 1],
        [1, 1, 1]
    ]

    print("\nТест задачи b (налив воды в центр):")
    start = time.time()
    result_b = pour_water(test_case_b, 1, 1, 1)
    print(f"Итоговый объем воды: {result_b:.1f}")
    print(f"Время выполнения: {time.time() - start:.6f} сек")

    print("\nДополнительный тест производительности (1000x1000):")
    large_matrix = [[5]*1000 for _ in range(1000)]
    start = time.time()
    large_result = trap_rain_water(large_matrix)
    print(f"Время расчета для большой матрицы: {time.time() - start:.2f} сек")

    print("Ковалевская А.М., 020303-АИСа-о24")

if __name__ == "__main__":
    main()
```

Результат выполнения программы:

```
= RESTART: C:\Users\alena\OneDrive\Desktop\lab3.py
=== ТЕСТИРОВАНИЕ РЕШЕНИЙ ===

Тест задачи a (погружение в воду):
Рассчитанный объем воды: 1
Время выполнения: 0.003418 сек

Тест задачи b (налив воды в центр):
Итоговый объем воды: 1.0
Время выполнения: 0.004333 сек

Дополнительный тест производительности (1000x1000):
Время расчета для большой матрицы: 0.97 сек
Ковалевская А.М., 020303-АИСа-о24
```

Вывод: стандартные библиотечные реализации списков демонстрируют оптимальную производительность за счет внутренних оптимизаций, тогда как самописные структуры на базе массивов и связных списков эффективны только в специфических сценариях с ограниченным набором операций.