Лабораторная 3

Тема 1.3. Структуры данных. Работа в VS Code со структурами данных «бинарная\_биноминальная куча/куча Фибоначчи/хеш-таблицы»

**Аналитический отчет по лабораторной работе: Реализация и сравнение структур данных на Python, C++ и Java**

**1. Бинарная и биномиальная кучи**

**1.1. Особенности структур**

* **Бинарная куча**: полное бинарное дерево, поддерживающее свойство кучи. Эффективна для операций вставки и извлечения минимума за O(log n).
* **Биномиальная куча**: коллекция биномиальных деревьев. Поддерживает слияние куч за O(1).

**1.2. Реализации**

**Python**

*# Бинарная куча через heapq*

import heapq

heap = []

heapq.heappush(heap, 5)

heapq.heappush(heap, 2)

min\_val = heapq.heappop(heap)

*# Собственный класс биномиальной кучи (упрощенный)*

class BinomialHeap:

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

self.trees = []

def merge(self, other):

*# Логика слияния деревьев*

pass

**C++**

*// Бинарная куча через priority\_queue*

#include <queue>

std::priority\_queue<int, std::vector<int>, std::greater<int>> minHeap;

minHeap.push(5);

minHeap.push(2);

int minVal = minHeap.top();

minHeap.pop();

*// Биномиальная куча (используется готовая реализация из примера)*

**Java**

*// Бинарная куча через PriorityQueue*

import java.util.PriorityQueue;

PriorityQueue<Integer> minHeap = new PriorityQueue<>();

minHeap.add(5);

minHeap.add(2);

int minVal = minHeap.poll();

*// Биномиальная куча (реализация из примера)*

**2. Куча Фибоначчи**

**2.1. Особенности**

* Поддерживает операции вставки, слияния и уменьшения ключа за амортизированное O(1).

**2.2. Реализации**

**Python**

*# Реализация через класс (упрощенная)*

class FibonacciHeap:

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

self.min\_node = None

self.count = 0

def insert(self, key):

*# Логика вставки*

pass

**C++**

*// Использование готовой реализации из примера*

FibonacciHeap<int> fibHeap;

fibHeap.insert(10);

fibHeap.insert(20);

int min = fibHeap.getMin();

**Java**

*// Использование PriorityQueue для эмуляции (не полноценно)*

PriorityQueue<Integer> fibHeap = new PriorityQueue<>();

fibHeap.add(10);

fibHeap.add(20);

int min = fibHeap.poll();

**3. Хеш-таблицы**

**3.1. Особенности**

* Обеспечивают среднюю сложность O(1) для вставки, удаления и поиска.

**3.2. Реализации**

**Python**

*# Встроенный словарь*

hash\_map = {}

hash\_map["key"] = "value"

val = hash\_map.get("key")

*# Собственная реализация*

class MyHashMap:

def \_\_init\_\_(self, size=10):

self.size = size

self.buckets = [[] for \_ in range(size)]

def \_hash(self, key):

return hash(key) % self.size

def put(self, key, value):

*# Логика вставки с разрешением коллизий*

pass

**C++**

*// Через unordered\_map*

#include <unordered\_map>

std::unordered\_map<std::string, int> hashMap;

hashMap["key"] = 123;

int val = hashMap["key"];

*// Собственная реализация (из примера)*

**Java**

*// Через HashMap*

import java.util.HashMap;

HashMap<String, Integer> hashMap = new HashMap<>();

hashMap.put("key", 123);

int val = hashMap.get("key");

*// Собственная реализация*

class MyHashMap<K, V> {

private LinkedList<Entry<K, V>>[] buckets;

public void put(K key, V value) {

*// Логика вставки*

}

}

**4. Сравнение реализаций на разных языках**

| Структура данных | Python | C++ | Java |
| --- | --- | --- | --- |
| **Бинарная куча** | heapq (min-heap) | priority\_queue | PriorityQueue |
| **Биномиальная куча** | Реализация вручную | Готовая реализация | Готовая реализация |
| **Куча Фибоначчи** | Реализация вручную | Готовая реализация | Эмуляция через PriorityQueue |
| **Хеш-таблица** | dict | unordered\_map | HashMap |

**Сравнительный анализ производительности операций**

| Структура данных | Вставка | Поиск минимума | Удаление минимума | Особенности |
| --- | --- | --- | --- | --- |
| **Бинарная куча** | O(log n) | O(1) | O(log n) | Простая реализация, эффективное использование памяти |
| **Биномиальная куча** | O(1) | O(log n) | O(log n) | Быстрое слияние куч |
| **Куча Фибоначчи** | O(1) | O(1) | O(log n) | Лучшая амортизированная сложность |
| **Хеш-таблица** | O(1) | O(1) | O(1) | Зависит от хеш-функции и коллизий |

**5. Вывод**

* **Python**: удобен для прототипирования, имеет встроенные модули (heapq, dict), но сложные структуры требуют ручной реализации.
* **C++**: предоставляет мощные STL-контейнеры, подходит для высокопроизводительных приложений.
* **Java**: богатая стандартная библиотека (PriorityQueue, HashMap), строгая типизация.