Левченко Мария Игоревна УИБО-09-24

**Куча**-это абстрактная структура данных, представляющая собой специализированное двоичное дерево (или набор деревьев), которое удовлетворяет свойству упорядоченности. Обычно используется для реализации очереди с приоритетом, где элемент с наивысшим (или наименьшим) приоритетом всегда находится в корне. Кучи позволяют эффективно выполнять операции вставки, извлечения и удаления корневого элемента.

**Бинарная куча** представляет собой полное бинарное дерево, удовлетворяющее свойству кучи: для max-кучи каждый узел больше или равен своим потомкам, для min-кучи — меньше или равен. Используется для реализации очередей с приоритетом.

Операции: добавление элемента, извлечение элемента, удаление элемента.

Реализация: в виде одномерного массива. Левый потомок вершины с индексом i имеет индекс 2i+1, правый — 2i+2. Корень дерева — элемент с индексом 0.

Python:

import heapq

heap = []

Java:

import java.util.PriorityQueue;

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

C++:

#include <queue>

std::priority\_queue<int> heap;

**Биноминальная куча-** это набор биномиальных деревьев — объектов, задаваемых рекуррентно. Биномиальное дерево нулевого ранга состоит из одной вершины. Биномиальное дерево ранга k представляет собой вершину и k детей, ранг которых последовательно уменьшается с k− 1 до 0.

Свойства: ключ каждой вершины не меньше ключа её родителя, все биномиальные деревья имеют разный размер.

Принцип работы: кроме стандартных операций очереди с приоритетом (добавления, удаления, извлечения минимума, изменения ключей) предоставляет дополнительную операцию слияния двух куч.

Операции: вставка нового элемента — создаёт новую кучу с новым узлом, извлечение минимального элемента, слияние двух биномиальных куч — объединяет корневые списки обеих куч в порядке возрастания их степеней, сравнивает градусы корней и сначала связывает деревья с меньшими градусами.

Python:

class BinomialNode:

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

self.key = key # Значение узла

self.degree = 0 # Количество дочерних узлов (порядок дерева)

self.child = None # Ссылка на первого ребенка

self.sibling = None # Ссылка на следующего брата (следующее дерево в списке)

class BinomialHeap:

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

self.head = None # Голова списка корневых деревьев

Java:

class BinomialNode {

int key; // Значение узла

int degree; // Количество дочерних узлов

BinomialNode child; // Ссылка на первого ребенка

BinomialNode sibling; // Ссылка на следующего брата

}

class BinomialHeap {

BinomialNode head; // Голова списка корневых деревьев

}

C++:

struct BinomialNode {

int key; // Значение узла

int degree; // Количество детей (порядок дерева)

BinomialNode\* child; // Указатель на первого ребенка

BinomialNode\* sibling; // Указатель на следующего брата

};

class BinomialHeap {

BinomialNode\* head; // Указатель на голову списка корней

};

**Куча Фибоначчи**— это набор деревьев, упорядоченных в соответствии со свойством неубывающей пирамиды: ключ каждого узла не меньше ключа его родительского узла.

Особенности: деревья не имеют заданной формы, в крайнем случае куча может содержать каждый элемент в отдельном дереве, дочерние узлы объединены указателями left и right в один циклический двусвязный список дочерних узлов, корни всех деревьев связаны указателями left и right в циклический двусвязный список корней, для всей кучи хранится указатель на узел с минимальным ключом — корень одного из деревьев.

Операции: вставка нового элементая, объединение двух куч, извлечение минимального узла, уменьшение.

Амортизированный анализ: амортизированное время работы операций над кучей Фибоначчи постоянно, за исключением удаления элемента (чаще всего — удаления минимального элемента) — O(log n), где n— размер кучи.

Python:

class FibonacciNode:

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

self.key = key # Значение узла

self.degree = 0 # Количество дочерних узлов

self.marked = False # Флаг потери ребенка (для каскадного вырезания)

self.child = None # Ссылка на любого ребенка

self.left = self # Левая ссылка в циклическом списке

self.right = self # Правая ссылка в циклическом списке

class FibonacciHeap:

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

self.min\_node = None # Указатель на узел с минимальным ключом

self.count = 0 # Количество узлов в куче

Java:

class FibonacciNode {

int key; // Значение узла

int degree; // Количество дочерних узлов

boolean marked; // Флаг потери ребенка

FibonacciNode child; // Ссылка на любого ребенка

FibonacciNode left; // Левая ссылка в циклическом списке

FibonacciNode right; // Правая ссылка в циклическом списке

}

class FibonacciHeap {

FibonacciNode min\_node; // Ссылка на узел с минимальным ключом

int count; // Количество узлов в куче

}

C++:

struct FibonacciNode {

int key; // Значение узла

int degree; // Количество дочерних узлов

bool marked; // Флаг потери ребенка

FibonacciNode\* child; // Указатель на любого ребенка

FibonacciNode\* left; // Левая ссылка в циклическом списке

FibonacciNode\* right; // Правая ссылка в циклическом списке

};

class FibonacciHeap {

FibonacciNode\* min\_node; // Указатель на узел с минимальным ключом

int count; // Количество узлов в куче

};

**Хеш-таблица**— структура данных, реализующая интерфейс ассоциативного массива. Она позволяет хранить пары «ключ, значение» и выполнять три операции: добавление новой пары, поиск и удаление пары по ключу

Компоненты хеш-таблицы: хеш-функция, бакеты, методы разрешения коллизий.

Принцип работы: выполнение операции в хеш-таблице начинается с вычисления хеш-функции от ключа. Хеш-код играет роль индекса в массиве, а зная индекс, можно выполнить требующуюся операцию

Виды: с цепочками, с открытой адресацией.

Операции: вставка элемента, поиск элемента, удаление элемента.

Python:

hash\_table = {}

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

Java:

import java.util.HashMap;

HashMap<String, String> hashTable = new HashMap<>();

hashTable.put("key", "value");

C++:

#include <unordered\_map>

std::unordered\_map<std::string, std::string> hash\_table;

hash\_table["key"] = "value";

Рассмотренные структуры данных решают разные задачи оптимизации хранения и обработки информации, но объединяет их одно: стремление обеспечить быстрый доступ к данным и эффективное выполнение операций. Кучи (бинарная, биномиальная, Фибоначчи) в первую очередь связаны с реализацией приоритетных очередей. Они различаются по организации элементов: от простого бинарного дерева (бинарная куча) до системы деревьев с возможностью эффективного слияния (биномиальная) и структуры с отложенной обработкой (Фибоначчи). Эти различия определяют сложность операций: чем сложнее структура, тем выгоднее становится работа с большими объёмами данных и многократными объединениями куч. Хэш-таблицы решают иную задачу — поиск и доступ к элементам по ключу за константное время. Они не поддерживают упорядоченность, но идеально подходят там, где требуется быстрый поиск, вставка и удаление. Таким образом, выбор структуры определяется характером задачи: для работы с приоритетами и сортировкой — различные виды куч; для быстрого ассоциативного доступа — хэш-таблицы. Знание особенностей этих структур позволяет подбирать оптимальный инструмент, снижая затраты по времени и памяти и делая программы более эффективными.