Heap, heap sort, priority queue

1. Heap

- Heap-ът е структура от данни, в която елементите са сортирани в нарастващ или намаляващ ред. В имплементацията на heap се използва пълно двоично дърво (complete binary tree). Пълното дърво е такова дърво, чиито нива са изцяло запълнени, с изключение на последното, в което може и да липсват един или повече елементи (последното също може да е запълнено изцяло). Съществуват два основни вида heap: max-heap и min-heap. Елементите в двоичното дърво са така наредени, че винаги родителят е по-голям (или равен) от двете деца (max-heap) или по-малък (min-heap). Съответно в корена на дървото при max-heap стои най-голямата стойност, а при min-heap – най-малката. Предимството на тази структура от данни е бързото взимане на максимален или минимален елемент.

- Основните операции, които heap-ът поддържа са:

    - heapify - пренарежда елементите, така, че свойствата на heap-а да се запазят

    - insertion – добавяне на елемент в heap-а

    - deletion – изтрива най-горния елемент и пренарежда heap-а

    - peek – връща най-горния елемент без да го премахва

- Сложностите на тези операции са:

    - heapify – O(log N)

    - insertion – O(log N)

    - deletion – O(log N)

    - peek – O(1)

- Кога би ни било полезно да използваме heap?

    - Heap-ът се използва в имплементацията на priority queue

    - Използваме го и при алгоритъма за сортиране heap sort

    - При имплементация на алгоритъма на Дикстра за най-къс път в граф

    - Когато имаме нужда да подредим елементите си по даден приоритет

Примерна имплементация на max-heap можете да намерите тук:
https://github.com/Bacekalki/SDP/blob/main/Heap.cpp

Можете да забележите, че в имплементацията на heap-а използвам вектор, за да репрезентирам пълното двоично дърво. Това е възможно, защото дървото е пълно и знаем, че лявото дете на елемента с индекс index е на позиция 2*index + 1, а дясното – 2 * index + 2. Родителят на елемента с индекс index е на позиция (index – 1) / 2. Разбира се, вместо

vector<int>, можем да използваме вектор с наш custom обект, стига този обект да има метод за сравняване, например предефинирани опереатори за сравнение.

2. Heap sort

- Heap sort е comparison-based алгоритъм за сортиране, който използва имплементация на binary heap, т.е heap, който използва пълно двоично дърво. В описанието се използва max-heap. Алгоритъмът се състои от следните стъпки:

1. Построяваме heap от подадения като агрумент вектор чрез heapify
2. Повтаряме следните стъпки докато в heap-а не остане само 1 елемент:
    - Сменяме местата на root-а на heap-а и последния елемент
    - Махаме последния елемент, защото той вече е на правилното място
    - Отново викаме heapify, за да пренаредим heap-а

- Сложност на heap sort
    - best case – O(N logN)
    - average case – O(N logN)
    - worst case – O(N logN)
- Предимства на heap sort
    - heap sort гарантира O(N logN) сложност
    - heap sort не изисква допълнителна памет спрямо броя на елементите, които трябва да се сортират (O(1) memory complexity).
    - Сравнително прост е и е лесно приложим, ако искаме да използваме priority queue

- Недостатък на heap-sort е, че дори да му подадем сортиран масив, сложността ще остане O(N logN).

Примерна имплементация на heap sort можете да намерите тук:

https://github.com/Bacekalki/SDP/blob/main/HeapSort.cpp

Визуална демонстрация на heap sort можете да намерите тук:

https://youtu.be/Q1yi1eaqN7I?si=DZNS2W9rpv_gZWkc

3. Priority queue

- Priority queue е абстрактна структура от данни, която пази елементи, заедно с техен приоритет. Тази структура позволява добавяне на елемент, премахване на най-горния елемент (този с най-висок или най-нисък приоритет) и достъпване на най-горния елемент без премахване. Има няколко начина, по които може да се направи имплементация на priority queue, като тази в стандартната библиотека на C++(std::priority_queue) използва max-heap. В действителност двете структури от данни наистина много си приличат и затова използването на max-heap при имплементация на priority queue е доста интуитивно. Разликата помежду им е, че priority queue-то е абстрактна структура от данни, чието описание е фокусирано върху методите, които трябва да се поддържат, а изборът на стркутурата от данни, която ще се използва е оставен на нас. Можем да гледаме на priority queue-то като на интерфейс. Heap-ът от своя страна си е конкретна структура от данни с конкретна имплементация чрез пълно двоично дърво.

- Ако искаме да използваме priority queue с наши custom обекти трябва да предоставим метод за сравняване, чрез който обектите да могат да се подредят в опашката.

- Операциите, които поддържа имплементацията в стандартната библиотека на C++ са:

    - push(element) – вкарва елемента в опашката
    - pop() – премахва най-горния елемент
    - top() – достъпва най-горния елемент без да го премахва
    - empty() – връща true, ако опашката е празна
    - size() – връща броя елементи в опашката

Credits:

https://www.youtube.com/@CC_ACADEMY

https://www.geeksforgeeks.org

https://www.javatpoint.com/

Изготвено от:

Калоян Джуджев, ФМИ