Heap

Основная теория

В чем суть структуры

Heap - структура данных, представляющая собой бинарное дерево, для которого выполняются следующие свойства:

- **Heap.Parent >= Heap.Child** - свойство макс-кучи (**Max-Heap**), то есть самый большой элемент кучи лежит в ее корне
- **Heap.Parent <= Heap.Child** - свойство мин-кучи (**Min-Heap**). Наименьший элемент кучи лежит в ее корне.

Идейная реализация

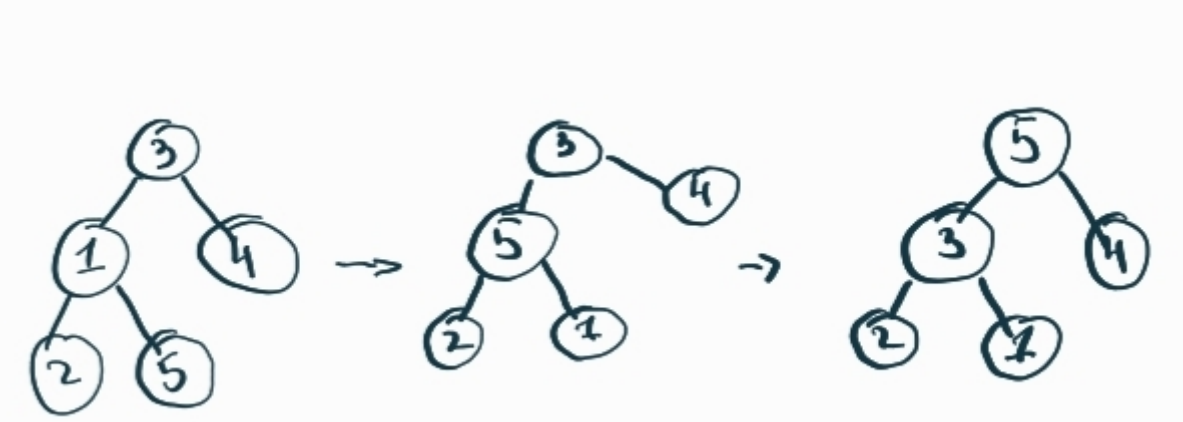
Операции над структурой и асимптотическая сложность

Операции для кучи:

- `Parent()` - метод возвращает индекс родителя для элемента
- `Left()` - возвращает индекс левого потомка
- `Right()` - возвращает индекс правого потомка
- `Max-Heaplify()` - операция для восстановления свойств кучи. Функция принимает объект кучи и индекс элемента, для которого мы хотим восстановить свойства кучи. Данная функция просеивает элемент вниз. Она выбирает набольший элемент среди элемента и его детей `heap[i]`, `heap[left]`, `heap[right]`. Если `heap[i]` - максимальный из трех, то выходим, иначе делаем `swap` максимального элемента и текущего (`heap[i]`, `heap[largest]`) рекурсивно вызываем для `heap[largest]`.
- `BuildHeap()` - функция принимает один аргумент - массив, и возвращает правильную кучу. Для реализации мы должны вызвать метод `MaxHeapify` для всех элементов (кроме листов) начиная с самых нижних, то есть от `heap.size / 2` до `0`.

Иллюстрации

Процесс построения кучи из массива:



HeapSort

Сортировка на куче. Нам дан массив из чисел. Сначала мы строим из него макс-кучу. Поскольку максимальный элемент находится на самом верху кучи, то мы всегда можем получить его значение.

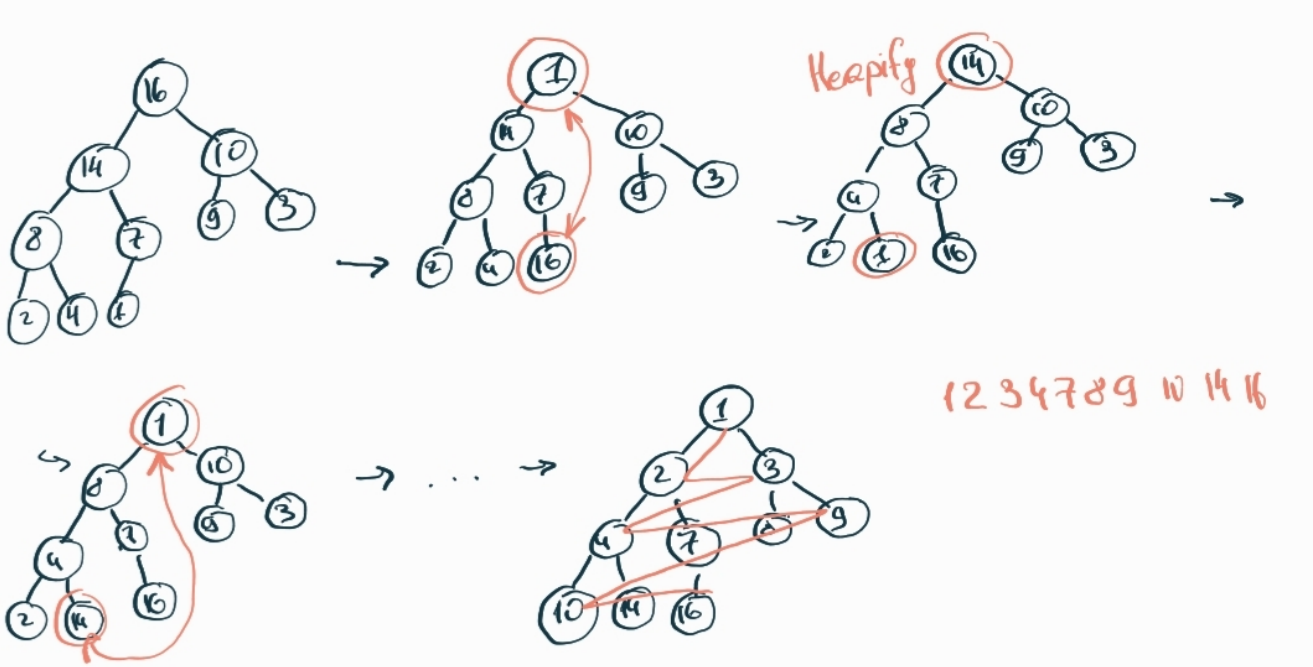
Алгоритм:

1. строим макс-кучу
2. делаем цикл **for** чтобы пробежаться по каждому элементу из кучи и свапнуть его с самым первым. Внутри цикла **for** мы уменьшаем размер кучи на 1, чтобы не учитывать самый большой элемент, который мы только свапнули.
3. На каждой итерации цикла вызываем функцию `MaxHeapify()` чтобы восстановить свойства кучи, поскольку мы только что ее сломали, положим минимальный элемент на ее верх.

В результате у нас получается массив, отсортированный по возрастанию.

Асимптотическая сложность сортировки: **O(n logn)** - **n** раз пробегаемся по элементам и для каждого вызываем функцию `MaxHeapify` сложность которой **log n**.

Пример сортировки:



Примеры кода

Код для методов кучи

Инициализация кучи. Функция построения кучи из массива.

```
// Max-Heap
// the largest element in the top of heap
type HeapArray struct {
    data []int
    size int
}

func NewHeapArray(size int) *HeapArray {
    return &HeapArray{
        size: size,
        data: make([]int, size),
    }
}

func BuildHeap(numbers []int) *HeapArray {
    size := len(numbers)

    heap := &HeapArray{
        size: size,
        data: make([]int, 0),
    }
    heap.data = append(heap.data, numbers...)
    for i := size/2 + 1; i >= 0; i-- {
        heap.MaxHeapify(i)
    }
```

```
    return heap
}
```

Методы кучи

```
func (h *HeapArray) Left(parent int) (int, error) {
    if parent*2+1 >= h.size {
        return 0, ErrNoLeftChild
    }

    return parent*2 + 1, nil
}

func (h *HeapArray) Right(parent int) (int, error) {
    if parent*2+2 >= h.size {
        return 0, ErrNoRightChild
    }
    return parent*2 + 2, nil
}

func (h *HeapArray) MaxHeapify(elem int) {
    largest := elem

    left, err := h.Left(elem)
    if err == nil && h.data[left] > h.data[elem] {
        largest = left
    }

    right, err := h.Right(elem)
    if err == nil && h.data[right] > h.data[largest] {
        largest = right
    }
    // if one of child is bigger than parent - swap elements
    if largest != elem {
        h.data[elem], h.data[largest] = h.data[largest], h.data[elem]
        h.MaxHeapify(largest)
    }
}
```

Реализация HeapSort

```
func HeapSort(numbers []int) []int {
    heap := BuildHeap(numbers)
    for i := heap.size - 1; i >= 0; i-- {
        heap.data[0], heap.data[i] = heap.data[i], heap.data[0]
        heap.size--
        heap.MaxHeapify(0)
    }
```

```
    return heap.data
}
```

Ресурсы

- Кормен "**Алгоритмы: построение и анализ**" стр.188
- Heap (data structure) - Wikipedia
- Пирамидальная сортировка (HeapSort) / Хабр (habr.com)