SORTOWANIE KOPCOWE | JAVA

WSTĘP

Sortowanie kopcowe (heapsort) jest względnie szybkim algorytmem sortowania – jego czas działania wynosi O(nlogn). Korzysta on ze struktury danych nazywanej kopcem.

KOPIEC

Kopiec jest pełnym lub prawie pełnym (pełnym na wszystkich poziomach z wyjątkiem ostatniego, który jest wypełniony od lewej strony do pewnego miejsca) drzewem binarnym z dodatkową cechą nazywaną **własnością kopca**. Zakłada ona, że każdy, który nie jest liściem posiada większą wartość od swoich dzieci. Kopce można rozpatrywać jako tablicę – wartości węzłów są wtedy kolejno przepisywane poziomami drzewa.

W algorytmie *heapsort* korzystam z poniżych funkcji działających na kopcu:

* left otrzymuje indeks węzła i zwraca indeks jego lewego dziecka

int left(int i) {

return 2\*i + 1;

}

* right otrzymuje indeks węzła i zwraca indeks jego prawego dziecka

int right(int i) {

        return 2\*i + 2;

    }

* Funkcja heapify dostaje jako argumenty kopiec w postaci tablicy, ilość jego elementów oraz indeks konkretnego elementu. Ma ona za zadanie przywrócić warunek kopca dla danego elementu. Na początku zostaje wybrana największa wartość z trójki *– rodzic, lewe dziecko, prawe dziecko*. Jeżeli A[i] ma największą wartość to warunek kopca dla tego elementu jest spełniony i możemy zakończyć działanie funkcji. W innym przypadku wartość węzła o indeksie i zostaje zamieniona z większą wartością jego dzieci, a następnie funkcja heapify zostaje rekurencyjnie wykonana na tym dziecku

void heapify(int A[], int n, int i) {

        int l = left(i);

        int r = right(i);

        int largest = i;

        if(l < n && A[l] > A[largest]) {

            largest = l;

        }

        if(r < n && A[r] > A[largest]) {

            largest = r;

        }

        if(largest != i) {

            int t = A[i];

            A[i] = A[largest];

            A[largest] = t;

            heapify(A, n, largest);

        }

    }

* buildHeap

void buildHeap(int A[]) {

        for(int i = A.length/2-1; i >= 0; i--) {

            heapify(A, A.length, i);

        }

    }

Funkcja buildHeap odpowiada za zbudowanie kopca z dowolnie ułożonej tablicy. Dzieje się to poprzez wywołanie funkcji heapify kolejno dla każdego elementu kopca niebędącego liściem (wykonanie operacji heapify dla liścia nie miałoby sensu, gdyż liście zawsze spełniają warunek kopca). Aby poddrzewa aktualnego elementu zawsze były kopcami wykonujemy operację od dołu drzewa – końca tablicy.

FUNKCJA HEAPSORT

void heapSort(int A[]) {

        buildHeap(A);

        int i;

        for(i = A.length - 1; i >= 1; i--) {

            int t = A[i];

            A[i] = A[0];

            A[0] = t;

            heapify(A, i, 0);

        }

    }

Funkcja *heapSort* w pierwszym kroku buduje kopiec z przekazanej do niej tablicy. Następnie wartość z korzenia kopca zostaje zamieniona z ostatnim miejscem tablicy i ignorowana do końca działania funkcji. Następnie na nowym korzeniu zostaje wywołana funkcja heapify w celu uzyskania kopca (nie biorąc już w ogóle pod uwagę ostatniego elementu) Proces ten jest powtarzany aż do „wyrzucenia” na koniec wszystkich elementów. W takim przypadku otrzymujemy posortowaną tablicę.

WAŻNE CECHY ALGORYTMU

* sortuje w miejscu
* nie jest stabilny
* ma złożoność czasową O(nlogn)