Kopiec N-arny

Oleg Semenov

Spis treści

Wstęp

W tym projekcie stworzyłem własną implemetację kopca N-arnego.
Ową implementację napisałem w C++14.

Wspomagałem się także następującymi źródłami:
https://www.cs.cmu.edu/~eugene/teach/algs03b/works/s6.pdf
https://www.sanfoundry.com/python-program-implement-d-ary-heap/

Metody

Klasa Heap posiada następujące metody:

Konstruktor

Konstruktor przyjmuje 1-3 argumenty:

- init_data — array z danymi wejściowymi dla kopca
- d — stopień kopca (domyślnie 3)
- type — typ budowy kopca (HeapType::MIN lub HeapType::MAX, wzrastający lub malejący)

```
Heap(std::initializer_list<T> init_data, size_t d = 3,
    HeapType type = HeapType::MIN)
        : data(init_data), d(d), type(type) {
    for (size_t i = data.size() / 2; i-- > 0;) {
        heapify(i);
    }
}

explicit Heap(size_t d = 3, HeapType type = HeapType::MIN)
        : d(d), type(type) {}
```

Insert

Metoda insert pozwala na dodanie elementu do kopca:

```
void insert(T key) {
    auto index = data.size();
    data.push_back(key);

    while (index > 0) {
        auto p = parent(index);
        if (cmp(data[index], data[p])) {
            std::swap(data[p], data[index]);
        }
        index = p;
    }
}
```

Extract

Metoda extract zwraca i jednocześnie usuwa element z góry kopca:

```
T extract() {
    auto top = data.at(0);
    data.front() = data.back();
    data.pop_back();
    heapify(0);
    return top;
}
```

Front

Metoda front zwraca górny element kopca bez usuwania:

```
T front() const {
    return data.front();
}
```

Size

Metoda size zwraca rozmiar kopca:

```
size_t size() const {
    return data.size();
}
```

Empty

Metoda empty sprawdza czy kopiec jest pusty, zwraca typ boolean:

```
bool empty() const {
    return data.empty();
}
```

Operator []

Operator [] umożliwia dostęp do i-tego elementu kopca:

```
T operator[](size_t i) const {
    return data[i];
}
```

Heapify

Metoda heapify jest metodą prywatną danej klasy. Używana jest do zachowania struktury kopca w danym objekcie.

```
void heapify(size_t i) {
    auto top = i;

    for (size_t n = 0; n < d && child(i, n) < data.size(); n++) {
        if (cmp(data[child(i, n)], data[top])) {
            top = child(i, n);
        }
    }

    if (top != i) {
        std::swap(data[top], data[i]);
        heapify(top);
    }
}
```

Cmp

Metoda cmp (prywatna) służy do porównywania dwóch elementów kopca. W zależności od typu kopca (HeapType::MIN lub HeapType::MAX) sprawdza czy wejściowe elementy mają następującą właśność $a < b$ lub $a > b$, zwracając przy tym wartość booleanowską.

```
 bool cmp(T a, T b) const {
        return type == HeapType::MIN ? a < b : a > b;
    }
```

Parent

Metoda parent zwraca index rodzica elementu o indexie i.

```
size_t parent(size_t i) const {
    return (i - 1) / d;
}
```

Child

Metoda parent zwraca index n-tego dziecka elementu o indexie i.

```
size_t child(size_t i, size_t n) const {
    return i * d + n + 1;
}
```

Testy jednostkowe

```
#define IS_TRUE(x) {temp &= x; if (!x) {std::cout << __FUNCTION__ \
    << " failed on line "<< __LINE__ << std::endl; }}

void unit_test() {
    Heap<size_t> heap({5, 2, 4, 1, 3}, 3);
    bool temp = true;
    IS_TRUE((heap.front() == 1))
    IS_TRUE((heap.size() == 5))
    IS_TRUE((heap[2] == 4))
    IS_TRUE(!heap.empty())
    heap.insert(6);
    IS_TRUE((heap[heap.size() - 1] == 6))
    int extracted = heap.extract();
    IS_TRUE((extracted == 1))
    IS_TRUE((heap.front() == 2))
    if (temp) {std::cout << "All unit tests passed.";}
}
```