Algorytmy 2
Laboratorium: kopiec binarny

Przemysław Klęsk

30 września 2019

1 Cel

Celem zadania jest wykonanie implementacji struktury danych nazywanej *kopcem binarnym* (ang. *binary heap*). Kopiec binarny zorientowany na maksimum jest drzewem binarnym, w którym każdy węzeł ma klucz większy lub równy kluczom dzieci[1]. Typowa implementacja kopca jest oparta na tablicy dynamicznej, do której węzły drzewa binarnego reprezentującgo kopiec są wpisane kolejno poziomami (wszystkie poziomy kopca poza ostatnim muszą być pełne). Dzięki powyższym warunkom proste operacje arytmetyczne pozwalają określić indeks rodzica dowolnego węzła oraz indeksy jego dzieci. Przy numerowaniu od zera, rodzic węzła i znajduje się pod numerem $\lfloor (i-1)/2 \rfloor$, a dzieci pod numerami $2i+1$, $2i+2$. Dwa popularne zastosowania kopca to: sortowanie przez kopcowanie (ang. *heap sort*) i znajdowanie najkrótszych ścieżek (np. algorytmy Dijkstry i A^*).

W ramach niniejszego zadania badane będą czasy wykonywania dwóch najistotniejszych operacji na kopcu tj.: dodawania nowego elementu oraz pobierania elementu maksymalnego (wraz z usunięciem go) — tzw. operacja *poll*. Obie te operacje mają złożoność obliczeniową $O(\log n)$, ponieważ naprawianie kopca wymaga pracy proporcjonalnej co najwyżej do jego wysokości. Tym samym wykonywanie tych operacji dla n elementów prowadzi do czasu rzędu $O(n \log n)$.

Uwaga: sortowanie na kopcu *nie* będzie realizowane w ramach tego zadania. Dodatkowe funkcje (metody) potrzebne do tego celu pojawią się w ramach późniejszego zadania porównującego sortowanie przez kopcowanie z sortowaniem przez zliczanie (ang. *counting sort*) i sortowaniem kubełkowym (ang. *bucket sort*) — czyli tzw. sortowaniami pseudoliniowymi.

2 Instrukcje, wskazówki, podpowiedzi

1. Podobnie jak w poprzednich zadaniach dozwolone są implementacja strukturalna lub obiektowa, przy czym ponownie wymagane jest użycie mechanizmu szablonów (`template`) języka C++ dla zachowania ogólności.

[1] W kopcu zorientowanym na minimum: klucz rodzica $\leqslant$ klucze dzieci.

2. Implementacja może wykorzystywać wcześniej wykonany własny kontener tablicy dynamicznej (np. poprzez dołączenie odpowiedniego pliku nagłówkowego `.h` i umieszczenie wewnątrz struktury / klasy kopca pola typu tablica dynamiczna). Przy czym nie jest to bezwzględny wymóg.

3. Na potrzeby tego zadania **interfejs kopca binarnego** powinien udostępniać następujące funkcje / metody:

   (a) dodanie nowego elementu (argumenty: dane i informacja lub komparator definiujące klucz porządkowania),

   (b) pobranie i usunięcie elementu maksymalnego tj. korzenia kopca (argument: informacja lub komparator definiujące klucz porządkowania — potrzebne do naprawienia kopca po usunięciu korzenia; wynik: dane związane z elementem maksymalnym lub niepowodzenie w razie pustego kopca),

   (c) czyszczenie kopca tj. usunięcie wszystkich elementów,

   (d) zwrócenie napisowej reprezentacji kopca — np. funkcja / metoda `to_string(...)` (argumenty i format wyniku wg uznania programisty — można wykorzystać m.in. napis pochodzący z tablicy dynamicznej, którą kopiec opakowuje),

   (e) przekopcowanie rekurencyjne w górę poczynając od podanego indeksu (argumenty: indeks oraz informacja lub komparator definiujące klucz porządkowania),

   (f) przekopcowanie rekurencyjne w dół poczynając od podanego indeksu (argumenty: indeks oraz informacja lub komparator definiujące klucz porządkowania).

   Punkty (e), (f) reprezentują narzędziowe funkcje naprawcze (na użytek wewnętrzny kopca) — nie powinny być one używane bezpośrednio przez użytkownia zaś wywoływane jedynie z wnętrza pozostałych funkcji.

4. W programie można wykorzystać ogólne wskazówki z poprzednich zadań dotyczące:

   - dynamicznego zarządania pamięcią (`new`, `delete`) — w szczególności przemyślenia miejsc odpowiedzialnych za uwalnianie pamięci danych,
   - wydzielenia implementacji interfejsu kopca do odrębnego pliku `.h`,
   - pracy z napisami (użycie typu `std::string`),
   - pomiaru czasu (funkcja `clock()` po dołączeniu `#include <time.h>`),
   - użycia wskaźników na funkcje,
   - generowania losowych danych (funkcje `rand()` i `srand(...)`).

3 Zawartość funkcji `main()`

Główny eksperyment zawarty w funkcji `main()` ma polegać na wielokrotnym dodawaniu coraz większej liczby elementów (danych) do kopca (rzędy wielkości od 10^1 aż do 10^7), a następnie pobieraniu (wraz z

usuwaniem) elementu maksymalnego w kopcu, aż do opróżnienia kopca. Należy raportować czasy dodawania i pobierania (całkowite i średnie).

Poniższy listing pokazuje schemat eksperymentu (proszę traktować go jako poglądowy przykład):

```
int main()
{
    ...
    const int MAX_ORDER = 7; // maksymalny rzad wielkosci dodawanych danych

    binary_heap<some_object*>* bh = new binary_heap<some_object*>();
    for (int o = 1; o <= MAX_ORDER; o++)
    {
        const int n = pow(10, o); // rozmiar danych

        // dodawanie do kopca
        clock_t t1 = clock();
        for (int i = 0; i < n; i++) {
            some_object* so = ... // losowe dane
            bh->add(so, some_objects_cmp);
        }
        clock_t t2 = clock();
        ... // wypis na ekran aktualnej postaci kopca (skrotowej) oraz pomiarow czasowych

        // pobieranie (i usuwanie) elementu maksymalnego z kopca
        t1 = clock();
        for (int i = 0; i < n; i++) {
            some_object* polled = bh->poll(some_objects_cmp); // argument: wskaznik na komparator
            ... // ewentualny wypis na ekran danych elementu pobranego (przy malym eksperymencie)
            delete polled;
        }
        t2 = clock();
        ... // wypis na ekran aktualnej postaci kopca (kopiec pusty) oraz pomiarow czasowych

        bh->clear(); // czyszczenie kopca (tak naprawde ,,zresetowanie'' tablicy dynamicznej
             opakowanej przez kopiec)
    }
    delete bh;
    return 0;
}
```

4 Sprawdzenie antyplagiatowe — przygotowanie wiadomości e-mail do wysłania

1. Kod źródłowy programu po sprawdzeniu przez prowadzącego zajęcia laboratoryjne musi zostać przesłany na adres *algo2@zut.edu.pl*.

2. Plik z kodem źródłowym musi mieć nazwę wg schematu: `nr_albumu.algo2.nr_lab.main.c` (plik może mieć rozszerzenie `.c` lub `.cpp`). Przykład: `123456.algo2.lab06.main.c` (szóste zadanie laboratoryjne studenta o numerze albumu 123456). Jeżeli kod źródłowy programu składa się z wielu plików, to należy stworzyć jeden plik, umieszczając w nim kody wszystkich plików składowych.

3. Plik musi zostać wysłany z poczty ZUT (zut.edu.pl).

4. Temat maila musi mieć postać: `ALGO2 IS1 XXXY LAB06`, gdzie `XXXY` to numer grupy (np. `ALGO2 IS1 210C LAB06`).

5. W pierwszych trzech liniach pliku z kodem źródłowym w komentarzach muszą znaleźć się:

   - informacja identyczna z zamieszczoną w temacie maila (linia 1),
   - imię i nazwisko autora (linia 2),
   - adres e-mail (linia 3).

6. Mail nie może zawierać żadnej treści (tylko załącznik).

7. W razie wykrycia plagiatu, wszytkie uwikłane osoby otrzymają za dane zadanie ocenę 0 punktów (co jest gorsze niż ocena 2 w skali $\{2, 3, 3.5, 4, 4.5, 5\}$).