Algorytmy 2
Laboratorium: tablica dynamiczna

Przemysław Klęsk

14 października 2020

1 Cel

Celem zadania jest wykonanie implementacji struktury danych nazywanej *tablicą dynamiczną* (ang. *dynamic array*). Jest to jedna z najbardziej podstawowych liniowych struktur danych i jednocześnie baza do tworzenia bardziej zaawansowanych struktur takich jak np. kopiec czy tablica mieszająca. Nazwa — tablica dynamiczna — wynika z faktu, że struktura ta w sposób dynamiczny powiększa swój rozmiar, gdy zachodzi taka potrzeba. Przyjmuje się pewien stały współczynnik powiększania (np. 1.5, 2.0) i po wyczerpaniu aktualnego rozmiaru tablicy próba dodania kolejnego elementu wymaga następujących czynności:

- zaalokowania większego (ciągłego) fragmentu pamięci na nową tablicę (stanowiącego np. odpowiednio 150% lub 200% poprzedniego rozmiaru),
- przepisania dotychczasowej zawartości ze starego miejsca w nowe,
- dopisania nowego elementu,
- uwolnienia pamięci zajmowanej przez starą tablicę.

Tym samym operacja dodawania pojedynczego elementu w przypadku optymistycznym (i typowym) ma stałą złożoność obliczeniową — $\Theta(1)$, zaś w przypadku pesymistycznym (opisanym powyżej) ma złożoność liniową — $\Theta(n)$. Należy jednocześnie zauważyć, że w związku z geometrycznym wzrostem rozmiaru tablicy, kolejne momenty wymagające przepisania zawartości, czyli kosztu liniowego, będą zachodziły coraz rzadziej, z częstością gasnącą również geometrycznie (wykładniczo). Tym samym można pokazać matematycznie, że tzw. *zamortyzowana* złożoność obliczeniowa operacji dodawania pozostaje *stała* — $\Theta(1)$ — co jest ważnym faktem motywującym powszechne stosowanie tej struktury danych. Należy przy tym zwrócić uwagę na podstawową zaletę używania tablicy czyli stały czasu dostępu do dowolnego elementu — $\Theta(1)$ — w odróżnieniu np. od list, gdzie czas dostępu jest liniowy.

Dodatkowym celem zadania jest wykonanie odpowiednich pomiarów czasowych obrazujących wspomniane powyżej fakty tyczące złożoności obliczeniowej. W ramach głównego eksperymentu, polegającego na dodawaniu dużej liczby elementów do tablicy dynamicznej, chcielibyśmy obserwować: czasy kolejnych pesymistycznych operacji dodawania, czas całkowity i czas zamortyzowany.

2 Instrukcje, wskazówki, podpowiedzi

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

2. Struktura (lub klasa) reprezentująca tablicę dynamiczną powinna zawierać: informacje o aktualnie faktycznym i maksymalnym rozmiarze oraz właściwą tablicę z danymi (lub wskaźnikami na dane) — deklaracyjnie np.: `T* array;`, gdzie `T` jest dowolnym typem.

3. Można przyjąć początkowy maksymalny rozmiar tablicy równy 1 i współczynnik rozszerzania równy 2.0.

4. **Interfejs tablicy dynamicznej** powinien udostępniać następujące funkcje / metody:

   (a) dodanie nowego elementu na końcu tablicy (argument: dane),

   (b) zwrócenie danych i-tego elementu tablicy (argument: indeks i żądanego elementu (numerując od zera); wynik: dane i-tego elementu lub niepowodzenie w razie indeksu poza zakresem),

   (c) ustawienie (podmiana) danych i-tego elementu tablicy (argument: indeks i żądanego elementu (numerując od zera) oraz nowe dane; wynik: pusty lub niepowodzenie w razie indeksu poza zakresem),

   (d) czyszczenie tablicy tj. usunięcie wszystkich elementów,

   (e) zwrócenie napisowej reprezentacji tablicy — np. funkcja / metoda `to_string(...)` (format wyniku wg uznania programisty, może zawierać np. aktualny rozmiar tablicy, aktualny maksymalny rozmiar tablicy, wypis pewnej liczby elementów początkowych / końcowych, opcjonalnie adres tablicy w pamięci; argumenty: również wg uznania programisty — np. liczba elementów do wypisania, wskaźnik na funkcję wypisującą pojedynczy rekord danych).

   (f) bąbelkowe posortowanie tablicy (argument: wskaźnik na komparator lub brak argumentu przy założeniu istnienia przeciążonego operatora `<` lub `>`); uwaga: sortowanie ma odbywać się w miejscu.

5. W programie można wykorzystać ogólne wskazówki z poprzedniego zadania dotyczące:

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

3 Zawartość funkcji `main()`

Główny eksperyment zawarty w funkcji `main()` ma polegać na dodawaniu dużej liczby elementów (danych) do tablicy dynamicznej, np. rzędu 10^7. Towarzyszyć mają temu pomiary czasowe. Poniższy listing pokazuje schemat eksperymentu (proszę traktować go jako poglądowy przykład):

```
int main()
{
    ...
    dynamic_array<some_object*>* da = new dynamic_array<some_object*>(); // stworzenie tablicy

    const int order = 7; // rzad wielkosci rozmiaru dodawanych danych
    const int n = pow(10, order); // rozmiar danych

    // dodawanie do tablicy
    clock_t t1 = clock();
    double max_time_per_element = 0.0; // najgorszy zaobserwowany czas operacji dodawania
    for (int i = 0; i < n; i++) {
        some_object* so = ... // losowe dane
        clock_t t1_element = clock();
        da->add(so);
        clock_t t2_element = clock();
        double time_per_element = ... // obliczenie czasu pojedynczej operacji dodawania
        if (time_per_element > max_time_per_element)
        {
            ... // odnotowanie nowego najgorszego czasu i komunikat informacyjny na ekran (przy ktorym
                indeksie mialo to miejsce)
        }
    }
    clock_t t2 = clock();
    ... // wypis na ekran aktualnej postaci tablicy (skrotowej) i pomiarow czasowych (czas calkowity i
        zamortyzowany)

    da->clear(true); // czyszczenie tablicy wraz z uwalnianiem pamieci danych
    delete da;
    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 $\{2, 3, 3.5, 4, 4.5, 5\}$).