AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

Testy jednostkowe

Autor:
Jakub Marek Tokarczyk

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań

1.1. Cel projektu

Projekt ma na celu zapoznanie z zasadami algorytmów sortowania, a także umożliwia zrozumienie efektywności i zastosowania algorytmu MergeSort w kontekście dużych zbiorów danych. Kolejnym celem jest nauczenie się technik pisania testów jednostkowych przy użyciu narzędzia Google Test w Visual Studio. Poprzez stworzenie testów jednostkowych dla różnych scenariuszy projekt ma na celu rozwój kompetencji w zakresie weryfikacji poprawności działania algorytmu. Testami należy sprawdzić czy algorytm:

- **Zachowuje tablicę niezmienioną, gdy jest już posortowana rosnąco:** należy sprawdzić, czy algorytm poprawnie obsługuje przypadek, gdy tablica jest już posortowana w porządku rosnącym. Algorytm nie powinien zmieniać elementów w tej tablicy,
- **Potrafi posortować tablicę, która jest posortowana w odwrotnej kolejności:** test ma sprawdzić, czy algorytm poprawnie sortuje tablicę, która jest posortowana w porządku malejącym (czyli w odwrotnej kolejności do rosnącej),
- **Poprawnie sortuje losową tablicę liczb:** w tym teście należy podać algorytmowi tablicę liczb w przypadkowej kolejności i sprawdzić, czy algorytm poprawnie posortuje te liczby,
- **Poprawnie sortuje tablicę tylko z liczbami ujemnymi:** należy sprawdzić, jak algorytm radzi sobie z tablicą zawierającą tylko liczby ujemne,
- **Poprawnie sortuje tablicę z liczbami ujemnymi i dodatnimi:** test ma sprawdzić, jak algorytm radzi sobie z tablicą, która zawiera zarówno liczby ujemne, jak i dodatnie,
- **Obsługuje pustą tablicę bez rzucania wyjątkiem:** ten test ma sprawdzić, czy algorytm prawidłowo obsługuje przypadek pustej tablicy. Algorytm nie powinien rzucać wyjątkiem, lecz po prostu zwrócić pustą tablicę,
- **Nie zmienia tablicy, która zawiera tylko jeden element:** test ma sprawdzić, czy algorytm nie zmienia tablicy, jeśli zawiera ona tylko jeden element. Algorytm nie powinien dokonywać żadnych operacji w takim przypadku,

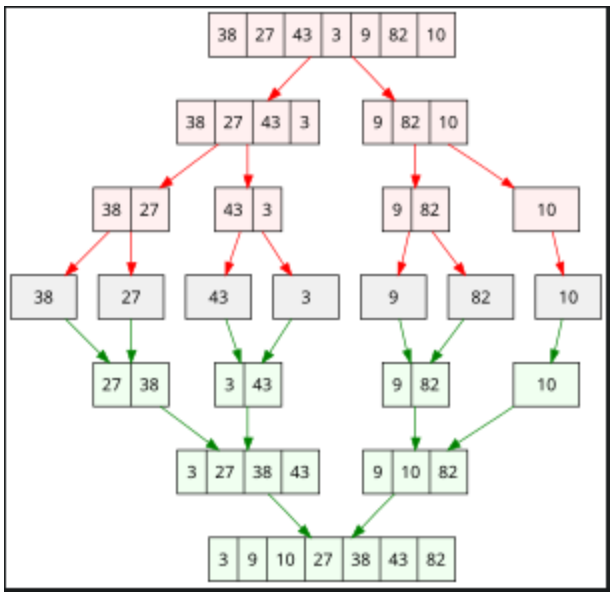
- **Poprawnie sortuje tablicę z duplikatami liczb:** należy sprawdzić, czy algorytm poprawnie sortuje tablicę zawierającą duplikaty liczb. Powinien on posortować duplikaty w odpowiedniej kolejności, zachowując ich wystąpienia,
- **Poprawnie sortuje tablicę ujemną z duplikatami:** test sprawdza, czy algorytm prawidłowo obsługuje tablicę, która zawiera tylko liczby ujemne i duplikaty,
- **Poprawnie sortuje tablicę z liczbami ujemnymi, dodatnimi oraz duplikatami:** należy sprawdzić, czy algorytm poprawnie sortuje tablicę, która zawiera zarówno liczby ujemne, dodatnie, jak i duplikaty,
- **Poprawnie sortuje tablicę zawierającą tylko dwa elementy w kolejności rosnącej:** ten test sprawdza, czy algorytm poprawnie sortuje tablicę, która zawiera dokładnie dwa elementy w nieposortowanej kolejności,
- **Poprawnie sortuje dużą tablicę zawierającą ponad 100 elementów:** w tym teście należy sprawdzić, czy algorytm jest w stanie poradzić sobie z dużą tablicą zawierającą ponad 100 elementów i poprawnie je posortować,
- **Poprawnie sortuje dużą tablicę zawierającą ponad 100 elementów z liczbami ujemnymi, dodatnimi oraz duplikatami:** test sprawdza, czy algorytm potrafi prawidłowo posortować dużą tablicę, która zawiera liczby ujemne, dodatnie i duplikaty.

Aby przeprowadzić te testy, należy zaimplementować algorytm MergeSort w klasie, a także stworzyć odpowiednią funkcję main, która będzie wywoływała sortowanie. Następnie, w osobnym pliku testowym, za pomocą Google Test należy napisać testy jednostkowe, które będą sprawdzały każde z powyższych założeń.

2. Analiza problemu

2.1. Sortowanie przez scalanie

Sortowanie przez scalanie (MergeSort)[1] jest jednym z najczęściej stosowanych algorytmów sortowania w informatyce. Jest to algorytm o stabilnej złożoności czasowej O(nlogn), co czyni go bardzo wydajnym, zwłaszcza w przypadku dużych zbiorów danych. W przeciwieństwie do algorytmu sortowania bąbelkowego czy sortowania przez wstawianie, MergeSort gwarantuje dobrą wydajność w przypadku nawet bardzo dużych zbiorów danych. Algorytm ten jest używany wszędzie tam, gdzie istotna jest stabilność sortowania i przewidywalna wydajność.



Rys. 2.1. Przykład

2.2. Gdzie używa się MergeSort?

Sortowanie przez scalanie znajduje zastosowanie w wielu dziedzinach informatyki i technologii. Jego najważniejsze zastosowania obejmują:

- **Sortowanie dużych zbiorów danych:** MergeSort jest szczególnie przydatny w przypadku bardzo dużych zbiorów danych, których rozmiar przekracza pamięć operacyjną. Algorytm jest bardzo efektywny w sortowaniu danych przechowywanych na dyskach twardych lub w systemach rozproszonych, ponieważ działa efektywnie w kontekście operacji wejścia/wyjścia,

[1] MergeSort[1]

- **Sortowanie w systemach baz danych:** algorytm MergeSort jest wykorzystywany w wielu systemach baz danych do sortowania rekordów, ponieważ jego stabilność (gdy elementy o tej samej wartości zachowują swoje pierwotne pozycje) jest przydatna w przypadkach, gdy potrzebne jest zachowanie porządku w innych kolumnach podczas sortowania według jednej z nich,

- **Analiza danych i algorytmy przetwarzania:** w zastosowaniach takich jak analiza danych, algorytmy rekomendacji, przetwarzanie danych w czasie rzeczywistym, MergeSort może być używany do efektywnego posortowania dużych zestawów danych wejściowych,

- **Wykorzystywanie w algorytmach dziel i zwyciężaj:** MergeSort jest klasycznym przykładem algorytmu działającego na zasadzie „dziel i zwyciężaj", który jest szeroko stosowany w wielu dziedzinach, od rozwiązywania problemów związanych z grafami po zadania związane z przetwarzaniem obrazów.

2.3. Sposób działania MergeSort

MergeSort działa w oparciu o podejście „dziel i zwyciężaj", które polega na tym, że problem sortowania jest dzielony na mniejsze podproblemy, które są łatwiejsze do rozwiązania, a następnie te podproblemy są scalane w rozwiązanie końcowe. Podstawowe kroki algorytmu MergeSort to:

1. Podział tablicy:

    - jeśli tablica zawiera więcej niż jeden element, jest dzielona na dwie mniejsze tablice,
    - proces dzielenia powtarza się, aż każda podtablica zawiera tylko jeden element, ponieważ tablica z jednym elementem jest już posortowana

2. Scalanie podtablic:

    - rozpoczyna się proces scalania tych jednoelementowych tablic w posortowane podtablice. Proces scalania polega na porównywaniu elementów z dwóch posortowanych tablic i tworzeniu z nich jednej większej tablicy posortowanej,
    - Dwa posortowane zbiory są łączone, zachowując porządek rosnący (lub malejący, w zależności od wymagań),

3. Rekursja:

- podziały i scalania są realizowane rekurencyjnie, co oznacza, że algorytm działa na coraz mniejszych podzbiorach, aż osiągnie tablicę jednoelementową,
- każde scalanie jest operacją, która łączy dwie posortowane tablice w jedną,

2.4. Testy jednostkowe

Test jednostkowy[2] metoda testowania tworzonego oprogramowania poprzez wykonywanie testów weryfikujących poprawność działania pojedynczych elementów (jednostek) programu– np. metod lub obiektów w programowaniu obiektowym lub procedur w programowaniu proceduralnym. Testowany fragment programu poddawany jest testowi, który wykonuje go i porównuje wynik (np. zwrócone wartości, stan obiektu, zgłoszone wyjątki) z oczekiwanymi wynikami tak pozytywnymi, jak i negatywnymi (niepowodzenie działania kodu w określonych sytuacjach również może podlegać testowaniu).

Zaletą testów jednostkowych jest możliwość wykonywania na bieżąco w pełni zautomatyzowanych testów na modyfikowanych elementach programu, co umożli wia często wychwycenie błędu natychmiast po jego pojawieniu się i szybką jego lokalizację zanim dojdzie do wprowadzenia błędnego fragmentu do programu. Testy jednostkowe są również formą specyfikacji. Z powyższych powodów są szczególnie popularne w programowaniu ekstremalnym.

[2]Test jednostkowy[2]

3. Projektowanie

3.1. W projekcie będę korzystać z następujących narzędzi

- **Visual Studio 2022**[3]**:** środowisko programistyczne, które będzie wykorzystywane do pisania, kompilowania i debugowania kodu. Visual Studio 2022 zapewnia pełne wsparcie dla języka C++ oraz zaawansowane funkcje debugowania i zarządzania projektem,

- **Język programowania C++**[4]**:** programowanie w języku C++ pozwoli na efektywną implementację algorytmów oraz struktur danych, takich jak lista dwukierunkowa. Język ten zapewnia dużą kontrolę nad pamięcią i wydajnością, co jest istotne w wielu projektach algorytmicznych i systemowych,

- **Git**[5]**:** system kontroli wersji, który będzie wykorzystywany do zarządzania wersjami kodu, śledzenia zmian. Git umożliwia łatwe zarządzanie historią projektu i integrację z platformami takimi jak GitHub,

- **GitHub**[6]**:** platforma oparta na systemie Git, która umożliwia hostowanie repozytoriów kodu w chmurze. GitHub nie tylko zapewnia zdalne przechowywanie projektów, ale także oferuje zaawansowane narzędzia do współpracy, takie jak pull requesty, które umożliwiają przeglądanie i zatwierdzanie zmian przed ich włączeniem do głównej wersji kodu. Platforma oferuje również funkcje zarządzania projektami, takie jak issues do śledzenia błędów i zadań, a także wikis i documentation, które pomagają w tworzeniu pełnej dokumentacji projektu. GitHub wspiera również integrację z narzędziami CI/CD, co pozwala na automatyzację procesów budowania i testowania aplikacji.

- **Google Test**[7]**:** framework do pisania testów jednostkowych w języku C++. Jest to narzędzie opracowane przez Google, które umożliwia programistom tworzenie, organizowanie i uruchamianie testów jednostkowych w celu weryfikacji poprawności kodu. Google Test jest szeroko stosowany w profesjonalnym tworzeniu oprogramowania, ponieważ zapewnia solidne narzędzia do testowania, umożliwiając automatyczne sprawdzanie, czy poszczególne części aplikacji działają zgodnie z oczekiwaniami.

[3] Visual Studio 2022[3]
[4] C++[4]
[5] Strona główna systemu kontroli wersji Git[5]
[6] GitHub[6]
[7] Google Test[7]

3.2. Sposób używania git

Git to system kontroli wersji, który pozwala na śledzenie i zarządzanie historią zmian w plikach projektu. Aby rozpocząć korzystanie z Gita, należy wykonać kilka podstawowych kroków, które obejmują konfigurację, tworzenie repozytorium oraz zarządzanie wersjami plików. Na początku trzeba zainstalować Git na swoim komputerze. Git jest dostępny dla systemów operacyjnych Windows, macOS i Linux, a instalację można pobrać ze strony oficjalnej. Po instalacji warto skonfigurować swoje dane użytkownika, aby Git wiedział, kto dokonał zmian w repozytorium. Po skonfigurowaniu Gita, można rozpocząć pracę nad repozytorium. Pierwszym krokiem jest utworzenie nowego repozytorium. Kiedy repozytorium jest gotowe, możemy zacząć pisanie kodu. Git umożliwia synchronizowanie lokalnych zmian z zewnętrznym repozytorium, co oznacza że możemy wysłać i pobrać zmiany kodu. Aby pobrać najnowsze zmiany z repozytorium, używamy komendy git pull a aby wysłać nasze zmiany na serwer, używamy git push. Dzięki temu systemowi każdy programista może swobodnie pracować nad projektem, a Git zapewnia kontrolę nad historią zmian, umożliwiając powrót do poprzednich wersji plików.

4. Implementacja

```
class MergeSort {
public:
    void sort(int arr[], int size);
    void mergeSort(int arr[], int left, int right);
    void merge(int arr[], int left, int mid, int right);
    void showArray(const int arr[], int size);
};
```

Listing 1. Klasa MergeSort

Listing 1

- class MergeSort: definicja klasy MergeSort, która będzie zawierać metody do sortowania tablicy za pomocą algorytmu sortowania przez scalanie. Klasa ta posiada cztery metody:

1. sort to główna metoda sortująca, która wywołuje rekurencyjne dzielenie i scalanie,
2. mergeSort to metoda rekurencyjnie dzieląca tablicę na mniejsze podtablice,
3. merge to metoda, która scala dwie posortowane tablice w jedną,
4. showArray to pomocnicza metoda wyświetlająca tablicę po sortowaniu.

```
void MergeSort::sort(int arr[], int size) {
    mergeSort(arr, 0, size - 1);
}
```

Listing 2. Implementacja metody sort

Listing 2.

- **mergeSort(arr, 0, size - 1):** wywołanie metody mergeSort w celu zainicjowania procesu sortowania. Przekazujemy pełny zakres tablicy (od indeksu 0 do size - 1), aby rozpocząć proces dzielenia tablicy na mniejsze podtablice.

```
void MergeSort::mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
```

```
7           merge(arr, left, mid, right);
8       }
9   }
```

Listing 3. Implementacja metody mergeSort

Listing 3.

- **Rekurencyjne dzielenie tablicy:** metoda mergeSort dzieli tablicę na dwie części, aż do osiągnięcia fragmentów, które mają jeden element. Działa to poprzez obliczenie punktu środkowego mid i wywoływanie funkcji rekurencyjnie na lewą i prawą część tablicy.
- **merge(arr, left, mid, right):** po podzieleniu tablicy, metoda wywołuje merge, która scala posortowane podtablice.

```
1  void MergeSort::merge(int arr[], int left, int mid, int right) {
2      int n1 = mid - left + 1;
3      int n2 = right - mid;
4
5      int* leftArray = new int[n1];
6      int* rightArray = new int[n2];
7
8      for (int i = 0; i < n1; ++i) leftArray[i] = arr[left + i];
9      for (int i = 0; i < n2; ++i) rightArray[i] = arr[mid + 1 + i];
10
11     int i = 0, j = 0, k = left;
12     while (i < n1 && j < n2) {
13         if (leftArray[i] <= rightArray[j]) {
14             arr[k] = leftArray[i];
15             ++i;
16         } else {
17             arr[k] = rightArray[j];
18             ++j;
19         }
20         ++k;
21     }
22
23     while (i < n1) {
24         arr[k] = leftArray[i];
25         ++i;
26         ++k;
27     }
28
29     while (j < n2) {
```

```
30          arr[k] = rightArray[j];
31          ++j;
32          ++k;
33      }
34
35      delete[] leftArray;
36      delete[] rightArray;
37  }
```

Listing 4. Implementacja metody merge

Listing 4.

- **Tworzenie tymczasowych tablic:** zanim połączymy dwie podtablice, tworzymy dwie tymczasowe tablice: leftArray i rightArray, które przechowują odpowiednio lewą i prawą część podzielonej tablicy.
- **Scalanie elementów:** w tej metodzie następuje właściwe scalanie: elementy z obu tablic są porównywane i układane w odpowiedniej kolejności w oryginalnej tablicy arr.
- **delete[] leftArray i delete[] rightArray:** na końcu zwalniamy pamięć przydzieloną dla tymczasowych tablic.

```
1  void MergeSort::showArray(const int arr[], int size) {
2      for (int i = 0; i < size; ++i) {
3          std::cout << arr[i] << " ";
4      }
5      std::cout << std::endl;
6  }
```

Listing 5. Metoda pomocnicza showArray

Listing 5.

- **Wyświetlanie tablicy:** ta metoda przechodzi przez wszystkie elementy tablicy i wypisuje je na ekranie, umożliwiając wizualizację efektów sortowania.

```
1  TEST(MergeSortTests, AlreadySorted) {
2      int arr[] = {1, 2, 3, 4, 5};
3      int size = sizeof(arr) / sizeof(arr[0]);
4
5      MergeSort sorter;
6      sorter.sort(arr, size);
```

```
7
8       int expected[] = {1, 2, 3, 4, 5};
9       EXPECT_EQ(std::vector<int>(arr, arr + size), std::vector<int>(
        expected, expected + size));
10  }
```

Listing 6. Testy jednostkowe z Google Test

Listing 6.

- **Testowanie już posortowanej tablicy:** ten test sprawdza, czy algorytm MergeSort poprawnie obsługuje już posortowaną tablicę. W tym przypadku wynik powinien być identyczny jak wejściowa tablica.
- **EXPECTEQ:** funkcja asercji z Google Test, która porównuje dwie tablice (wejściową i oczekiwaną). Jeżeli tablice są różne, test zakończy się niepowodzeniem.

```
1   TEST(MergeSortTests, ReverseSorted) {
2       int arr[] = {5, 4, 3, 2, 1};
3       int size = sizeof(arr) / sizeof(arr[0]);
4
5       MergeSort sorter;
6       sorter.sort(arr, size);
7
8       int expected[] = {1, 2, 3, 4, 5};
9       EXPECT_EQ(std::vector<int>(arr, arr + size), std::vector<int>(
        expected, expected + size));
10  }
```

Listing 7. Testy jednostkowe z Google Test

Listing 7.

- **Testowanie tablicy posortowanej w odwrotnej kolejności:** w tym przypadku testujemy, czy algorytm potrafi posortować tablicę, która jest w odwrotnej kolejności.
- **Google Test:** jak poprzednio, funkcja EXPECTEQ porównuje wynik z oczekiwanym.

5. Wnioski

- Tematy poruszone w projekcie pozwoliły mi zdobyć cenne doświadczenie zarówno w zakresie testów jednostkowych, jak i pracy z platformą GitHub. To było moje pierwsze spotkanie z testami jednostkowymi, które są niezwykle ważnym narzędziem w procesie tworzenia oprogramowania. Zrozumiałem, jak ogromną rolę pełnią testy jednostkowe w zapewnianiu jakości kodu. Dzięki nim możliwe jest wczesne wykrywanie błędów, co znacząco ułatwia ich naprawę i zmniejsza ryzyko pojawienia się problemów w późniejszych fazach projektu. Testowanie jednostkowe pozwala na automatyczne sprawdzanie poprawności działania poszczególnych części aplikacji, co w praktyce przekłada się na mniejszą liczbę błędów produkcyjnych oraz większą stabilność kodu.

- Praca z Google Test, która była częścią tego projektu, nauczyła mnie, jak pisać i organizować testy, jak testować różnorodne przypadki, a także jak analizować wyniki testów w celu poprawy jakości kodu. Wykonywanie testów jednostkowych pozwala na szybsze iteracje w procesie programowania i zapewnia większą pewność, że wprowadzone zmiany nie wprowadzą nowych błędów do systemu.

- Projekt dał mi również szansę na zapoznanie się z pracą na repozytoriach GitHub, co okazało się bardzo wartościowe. Do tej pory głównie pracowałem na swoich lokalnych repozytoriach, ale teraz miałem okazję poznać proces „forkowania" repozytoriów. Jest to bardzo przydatna funkcjonalność, która pozwala na stworzenie kopii repozytorium i swobodne pracowanie na niej, bez ryzyka zmiany oryginalnego projektu. Dzięki temu można testować własne rozwiązania, wprowadzać zmiany i później, jeśli zajdzie taka potrzeba, zaproponować swoje poprawki do głównego repozytorium (tzw. pull request). To narzędzie jest kluczowe w pracy zespołowej, szczególnie w projektach open-source, gdzie wiele osób może pracować równocześnie nad różnymi funkcjami lub poprawkami.

Podsumowując, projekt ten był dla mnie nie tylko okazją do nauki narzędzi, które pomogły w pisaniu testów jednostkowych, ale także pozwolił na zdobycie praktycznego doświadczenia w pracy z platformą GitHub. Zdecydowanie uważam, że te umiejętności będą niezwykle pomocne w mojej dalszej pracy programistycznej, szczególnie w kontekście pracy zespołowej oraz zapewniania jakości kodu w profesjonalnych projektach.

Bibliografia

[1] *Sortowanie przez scalanie*. URL: https://www.algorytm.edu.pl/algorytmy-maturalne/sortowanie-przez-scalanie.html#google_vignette (term. wiz. 08. 10. 2022).

[2] *Pisanie testów jednostkowych*. URL: https://learn.microsoft.com/pl-pl/visualstudio/test/writing-unit-tests-for-c-cpp?view=vs-2022 (term. wiz. 08. 10. 2022).

[3] *Visual Studio 2022*. URL: https://visualstudio.microsoft.com/pl/vs/getting-started/ (term. wiz. 10. 12. 2024).

[4] *Nauka c++*. URL: https://www.w3schools.com/cpp/ (term. wiz. 03. 12. 2024).

[5] *Git*. URL: https://git-scm.com/ (term. wiz. 09. 11. 2024).

[6] *Github*. URL: https://github.com/ (term. wiz. 03. 12. 2024).

[7] *Google Test*. URL: https://google.github.io/googletest/ (term. wiz. 03. 12. 2024).

Spis rysunków

Spis tabel

Spis listingów