Merge Sort

3

Wygenerowano za pomocą Doxygen 1.12.0

Rozdział 1

Indeks klas

1.1 Lista klas

Tutaj znajdujÄ siÄ klasy, struktury, unie i interfejsy wraz z ich krÃ^{3}tkimi opisami:

Rozdział 2

Indeks plików

2.1 Lista plików

Tutaj znajduje się lista wszystkich plików wraz z ich krótkimi opisami:

Rozdział 3

Dokumentacja klas

3.1 Dokumentacja klasy MergeSort

Klasa implementuj¹ca algorytm MergeSort.

```
#include <MergeSort.h>
```

Statyczne metody publiczne

- static void Sort (std::vector< int > &tab)

  Funkcja sortuj¹ca tablicê przy u¿yciu algorytmu MergeSort.
- static void Divide (std::vector< int > &tab, int l, int r)

  Funkcja dziel¹ca tablicê na dwie czêci do dalszego sortowania.
- static void Merge (std::vector< int > &tab, int l, int mid, int r)

  Funkcja scala dwie posortowane czêci tablicy w jedn¹ posortowan¹ ca³oœæ.

3.1.1 Opis szczegÃ³Åowy

Klasa implementuj¹ca algorytm MergeSort.

Definicja w linii 7 pliku MergeSort.h.

3.1.2 Dokumentacja funkcji skÅadowych

3.1.2.1 Divide()

```
void MergeSort::Divide (
            std::vector< int > & tab,
            int l,
            int r)  [static]
```

Funkcja dziel¹ca tablicê na dwie czêci do dalszego sortowania.

Funkcja ta dzieli tablicê na dwie czêci na podstawie indeksów lewego (l) i prawego (r) granicy. Nastêpnie wywo³uje rekurencyjnie sortowanie dla obu czêci.

Parametry

tab	Referencja do tablicy, która ma zostaæ podzielona.
l	Indeks pocz¹tkowy lewej czêci tablicy.
r	Indeks koñcowy prawej czêci tablicy.

Definicja w linii 10 pliku MergeSort.cpp.

3.1.2.2 Merge()

```
void MergeSort::Merge (
            std::vector< int > & tab,
            int l,
            int mid,
            int r)  [static]
```

Funkcja scala dwie posortowane czêci tablicy w jedn¹ posortowan¹ ca^{3}oæ.

Ta funkcja ^{3}¹czy dwie posortowane czêci tablicy w jeden ci¹g w sposób wydajny, zachowuj¹c porz¹dek rosn¹cy. Operacja scalania jest kluczow¹ czêci¹ algorytmu MergeSort.

Parametry

tab	Referencja do tablicy, w której odbywa siê scalanie.
l	Indeks pocz¹tkowy lewej czêci tablicy.
mid	Indeks rodkowy, który dzieli tablicê na dwie czêci.
r	Indeks koñcowy prawej czêci tablicy.

Definicja w linii 21 pliku MergeSort.cpp.

3.1.2.3 Sort()

```
void MergeSort::Sort (
            std::vector< int > & tab)  [static]
```

Funkcja sortuj¹ca tablicê przy u¿yciu algorytmu MergeSort.

Ta funkcja jest wywo^{3}ywana w celu rozpoczêcia procesu sortowania tablicy. U¿ywa algorytmu MergeSort, który dzieli tablicê na mniejsze czêci i scala je w posortowan¹ ca^{3}oæ.

Parametry

tab	Referencja do tablicy, która ma zostaæ posortowana.

Definicja w linii 3 pliku MergeSort.cpp.

Dokumentacja dla tej klasy zostaÅa wygenerowana z plikÃ^{3}w:

- MergeSort/MergeSort.h
- MergeSort/MergeSort.cpp

Rozdział 4

Dokumentacja plików

4.1 Dokumentacja pliku MergeSort/MergeSort.cpp

```
#include "MergeSort.h"
```

4.2 MergeSort.cpp

Idź do dokumentacji tego pliku.

```
00001 #include "MergeSort.h"
00002
00003 void MergeSort::Sort(std::vector<int>& tab) {
00004
00005     if (tab.size() <= 1) return; // <-- Jeśli tablica ma jeden lub mniej elementów, nie ma potrzeby
      sortować
00006     Divide(tab, 0, tab.size() - 1); // <-- Rozpoczynamy dzielenie tablicy od indeksu 0 do ostatniego
      elementu
00007
00008 }
00009
00010 void MergeSort::Divide(std::vector<int>& tab, int l, int r) { // <-- Funkcja jest odpowiedzialana za
      dzielenie tablicy na mniejsze podtablice
00011
00012
00013     if (l >= r) return; // <-- Jeśli lewy indeks jest większy lub równy prawemu, oznacza to, że nie ma
      już co dzielić (tablica ma tylko jeden element)
00014         int mid = l + (r - l) / 2; // <-- Obliczamy indeks środkowy
00015         Divide(tab, l, mid);       // <-- Rekurencyjnie dzielimy lewą część tablicy
00016         Divide(tab, mid + 1, r);   // <-- Rekurencyjnie dzielimy prawą część tablicy
00017         Merge(tab, l, mid, r);     // <-- Scalanie obu posortowanych części tablicy
00018
00019 }
00020
00021 void MergeSort::Merge(std::vector<int>&tab, int l, int mid, int r) {
00022
00023     std::vector<int> left(tab.begin() + l, tab.begin() + mid + 1); // <-- Tworzenie tablicy która
      zawiera elementy lewa do środka
00024     std::vector<int> right(tab.begin() + mid + 1, tab.begin() + r + 1); // <-- Tworzenie tablicy która
      zawiera elementy środka do końca
00025
00026      int i = 0; // <-- indeks dla lewej podtablicy
00027      int j = 0; // <-- indeks dla lewej prawej
00028      int k = l; // <-- indeks dla lewej głównej
00029
00030      while (i < left.size() && j < right.size()) { // <-- Scalanie obu podtablic to tablicy głównej
      dopóki są elementy w obu podtablicach
00031
00032          if (left[i] <= right[j]) { // <-- Jeśli element w lewej podtablicy jest mniejszy lub równy
      elementowi w prawej
00033
00034              tab[k++] = left[i++]; // <-- Wstawiamy element z lewej podtablicy do głównej tablicy
00035
```

```
00036             }
00037
00038         else {
00039
00040              tab[k++] = right[j++]; // <-- Wstawiamy element z prawej podtablicy do głównej tablicy
00041
00042             }
00043
00044         }
00045
00046     while (i < left.size()) { // <-- Jeżeli pozostały elementy w lewej podtablicy, wstawiamy je do
      głównej tablicy
00047
00048          tab[k++] = left[i++]; // <-- Przenosimy pozostałe elementy z lewej podtablicy
00049
00050         }
00051
00052     while (j < right.size()) { // <-- Jeśli pozostały elementy w prawej podtablicy, wstawiamy je do
      głównej tablicy
00053
00054          tab[k++] = right[j++]; // <-- Przenosimy pozostałe elementy z prawej podtablicy
00055
00056         }
00057
00058 }
00059
00060
```

4.3 Dokumentacja pliku MergeSort/MergeSort.h

```
#include <vector>
```

Komponenty

- class MergeSort

  Klasa implementuj¹ca algorytm MergeSort.

4.4 MergeSort.h

Idź do dokumentacji tego pliku.

```
00001 #pragma once
00002 #include <vector>
00007 class MergeSort {
00008
00009     public:
00018     static void Sort(std::vector<int>& tab);
00029     static void Divide(std::vector<int>& tab, int l, int r);
00041     static void Merge(std::vector<int>& tab, int l, int mid, int r);
00042
00043 };
```

4.5 Dokumentacja pliku MergeSortTest/test.cpp

```
#include "C:/Users/Adrian/Desktop/MergeSort/MergeSort/MergeSort.cpp"
#include "C:/Users/Adrian/Desktop/MergeSort/MergeSort/MergeSort.h"
#include <gtest/gtest.h>
```

Funkcje

- TEST (Tests, correct)
- TEST (Tests, reverseOrder)
- TEST (Tests, randomOrder)
- TEST (Tests, onlyNegative)
- TEST (Tests, negativeANDpositive)
- TEST (Tests, empty)
- TEST (Tests, oneNumber)
- TEST (Tests, duplicates)
- TEST (Tests, negativeDuplicates)
- TEST (Tests, twoNumbers)
- TEST (Tests, overHundred)
- TEST (Tests, allTypesHundred)

4.5.1 Dokumentacja funkcji

4.5.1.1 TEST() [1/12]

```
TEST (
            Tests ,
            allTypesHundred )
```

Definicja w linii 117 pliku test.cpp.

4.5.1.2 TEST() [2/12]

```
TEST (
            Tests ,
            correct )
```

Definicja w linii 5 pliku test.cpp.

4.5.1.3 TEST() [3/12]

```
TEST (
            Tests ,
            duplicates )
```

Definicja w linii 68 pliku test.cpp.

4.5.1.4 TEST() [4/12]

```
TEST (
            Tests ,
            empty )
```

Definicja w linii 50 pliku test.cpp.

4.5.1.5 TEST() [5/12]

```
TEST (
            Tests ,
            negativeANDpositive )
```

Definicja w linii 41 pliku test.cpp.

4.5.1.6 TEST() [6/12]

```
TEST (
            Tests ,
            negativeDuplicates )
```

Definicja w linii 77 pliku test.cpp.

4.5.1.7 TEST() [7/12]

```
TEST (
            Tests ,
            oneNumber )
```

Definicja w linii 59 pliku test.cpp.

4.5.1.8 TEST() [8/12]

```
TEST (
            Tests ,
            onlyNegative )
```

Definicja w linii 32 pliku test.cpp.

4.5.1.9 TEST() [9/12]

```
TEST (
            Tests ,
            overHundred )
```

Definicja w linii 95 pliku test.cpp.

4.5.1.10 TEST() [10/12]

```
TEST (
            Tests ,
            randomOrder )
```

Definicja w linii 23 pliku test.cpp.

4.5.1.11 TEST() [11/12]

```
TEST (
            Tests ,
            reverseOrder )
```

Definicja w linii 14 pliku test.cpp.

4.5.1.12 TEST() [12/12]

```
TEST (
            Tests ,
            twoNumbers )
```

Definicja w linii 86 pliku test.cpp.

4.6 test.cpp

Idź do dokumentacji tego pliku.

```
00001 #include "C:/Users/Adrian/Desktop/MergeSort/MergeSort/MergeSort.cpp"
00002 #include "C:/Users/Adrian/Desktop/MergeSort/MergeSort/MergeSort.h"
00003 #include <gtest/gtest.h>
00004
00005 TEST(Tests, correct) {
00006
00007     std::vector<int> t = { 1, 2, 3, 4, 5 };
00008     MergeSort::Sort(t);
00009     std::vector<int> expected = { 1, 2, 3, 4, 5 };
00010     ASSERT_EQ(t, expected);
00011
00012 }
00013
00014 TEST(Tests, reverseOrder) {
00015
00016     std::vector<int> t = { 10, 9, 8, 7, 6 };
00017     MergeSort::Sort(t);
00018     std::vector<int> expected = { 6, 7, 8, 9, 10 };
00019     ASSERT_EQ(t, expected);
00020
00021 }
00022
00023 TEST(Tests, randomOrder) {
00024
00025     std::vector<int> t = { 3, 1, 4, 5, 2 };
00026     MergeSort::Sort(t);
00027     std::vector<int> expected = { 1, 2, 3, 4, 5 };
00028     ASSERT_EQ(t, expected);
00029
00030 }
00031
00032 TEST(Tests, onlyNegative) {
00033
00034     std::vector<int> t = { -12, -8, -4, -16, -2 };
00035     MergeSort::Sort(t);
00036     std::vector<int> expected = { -16, -12, -8, -4, -2 };
00037     ASSERT_EQ(t, expected);
00038
00039 }
00040
00041 TEST(Tests, negativeANDpositive) {
00042
00043     std::vector<int> t = { -18, 30, -41, 1, -2 };
00044     MergeSort::Sort(t);
00045     std::vector<int> expected = { -41, -18, -2, 1, 30 };
00046     ASSERT_EQ(t, expected);
00047
00048 }
00049
00050 TEST(Tests, empty) {
00051
00052     std::vector<int> t = {};
```

```
00053     MergeSort::Sort(t);
00054     std::vector<int> expected = {};
00055     ASSERT_EQ(t, expected);
00056
00057 }
00058
00059 TEST(Tests, oneNumber) {
00060
00061     std::vector<int> t = { 37 };
00062     MergeSort::Sort(t);
00063     std::vector<int> expected = { 37 };
00064     ASSERT_EQ(t, expected);
00065
00066 }
00067
00068 TEST(Tests, duplicates) {
00069
00070     std::vector<int> t = { 3, 6, 5, 5, 6 ,3 };
00071     MergeSort::Sort(t);
00072     std::vector<int> expected = { 3, 3, 5, 5, 6 ,6 };
00073     ASSERT_EQ(t, expected);
00074
00075 }
00076
00077 TEST(Tests, negativeDuplicates) {
00078
00079     std::vector<int> t = { -10, -3, -5, -10, -5 ,-3 };
00080     MergeSort::Sort(t);
00081     std::vector<int> expected = { -10, -10, -5, -5, -3, -3 };
00082     ASSERT_EQ(t, expected);
00083
00084 }
00085
00086 TEST(Tests, twoNumbers) {
00087
00088     std::vector<int> t = { 1,2 };
00089     MergeSort::Sort(t);
00090     std::vector<int> expected = { 1,2 };
00091     ASSERT_EQ(t, expected);
00092
00093 }
00094
00095 TEST(Tests, overHundred) {
00096
00097     std::vector<int> t(101);
00098     std::vector<int> expected(101);
00099
00100     for (int i = 0; i < 101; i++) {
00101
00102         expected[i] = i * 2;
00103     }
00104
00105     for (int i = 0; i < 101; i++) {
00106
00107         t[i] = i * 2;
00108     }
00109
00110     t[0] = 2;
00111     t[1] = 0;
00112     MergeSort::Sort(t);
00113     ASSERT_EQ(t, expected);
00114
00115 }
00116
00117 TEST(Tests, allTypesHundred) {
00118
00119     std::vector<int> t(201);
00120     std::vector<int> expected(201);
00121
00122     for (int i = -100; i < 101; i++) {
00123
00124         int index = i + 100;
00125         expected[index] = i * 2;
00126         t[index] = i * 2;
00127
00128     }
00129
00130     t[20] = 16;
00131     expected[20] = 16;
00132     t[20] = 6;
00133     expected[20] = 6;
00134     std::sort(expected.begin(), expected.end());
00135     MergeSort::Sort(t);
00136     ASSERT_EQ(t, expected);
00137
00138 }
```

Skorowidz