Benchmark

Wygenerowano przez Doxygen 1.8.3.1

N, 2 mar 2014 13:40:01

Spis treści

Rozdział 1

Strona główna

1.1 Opis programu

Program mierzy czas wykonania algorytmu, który mnoży każdy element wektora przez 2.

Rozdział 2

Indeks przestrzeni nazw

2.1 Lista przestrzeni nazw

Tutaj znajdują się wszystkie przestrzenie nazw wraz z ich krótkimi opisami:

Rozdział 3

Indeks hierarchiczny

3.1 Hierarchia klas

Ta lista dziedziczenia posortowana jest z grubsza, choć nie całkowicie, alfabetycznie:

Rozdział 4

Indeks klas

4.1 Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

Rozdział 5

Indeks plików

5.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

Rozdział 6

Dokumentacja przestrzeni nazw

6.1 Dokumentacja przestrzeni nazw GenerateInput

Funkcje

- def GenerateInput

6.1.1 Dokumentacja funkcji

6.1.1.1 def GenerateInput.GenerateInput (*fileName* = "input.txt", *problemSize* = [10, *repeatNum* = 10)

Rozdział 7

Dokumentacja klas

7.1 Dokumentacja szablonu klasy Benchmark< ProblemType >

Klasa pozwalająca na testowanie algorytmów.

`#include <Benchmark.hpp>`

Metody publiczne

- Benchmark (int repeatNum=10)

  Konstruktor pozwalający na wybór ilości powtórzeń algorytmu dla danego rozmiaru problemu.
- void start (std::istream &isInputData=std::cin, std::istream &isCorrectOutData=std::cin)

  Rozpoczyna testowanie algorytmu.
- bool saveAsCSV (std::ostream &os=std::cout)

  Metoda pozwala zapisać wyniki testów wydajnościowych do strumienia.

Atrybuty prywatne

- ProblemType m_problem

  Przechowuje obiekt, odpowiedzialny za algorytm.
- std::vector< BenchmarkData > m_allBenchmarks

  Wektor przechowujący informacje o przeprowadzonych testach.
- unsigned int m_repeatNum

7.1.1 Opis szczegółowy

template<typename ProblemType>class Benchmark< ProblemType >

Klasa pozwalająca na testowanie algorytmów.

ProblemType musi udostępniać określony interfejs tzn. musi implementować następujące metody: void readInData(std::istream& is = std::cin) void readOutData(std::istream& is = std::cin) bool isCorrect() const void compute() void clearData()

Zalecane jest, aby dziedziczył on po klasie Problem

7.1.2 Dokumentacja konstruktora i destruktora

7.1.2.1 template<typename ProblemType > Benchmark< ProblemType >::Benchmark (int *repeatNum* = 10) [inline]

Konstruktor pozwalający na wybór ilości powtórzeń algorytmu dla danego rozmiaru problemu.

7.1.3 Dokumentacja funkcji składowych

7.1.3.1 template<typename ProblemType > bool Benchmark< ProblemType >::saveAsCSV (std::ostream & *os* = std::cout)

Metoda pozwala zapisać wyniki testów wydajnościowych do strumienia.

Format jest zgodny z CSV.

Parametry

os	- strumień wyjściowy, do którego mają zostać przekierowane wyniki

7.1.3.2 template<typename ProblemType > void Benchmark< ProblemType >::start (std::istream & *isInputData* = std::cin, std::istream & *isCorrectOutData* = std::cin)

Rozpoczyna testowanie algorytmu.

Metoda wczytuje dane wejściowe dla algorytmu, następnie poprawne dane wyjściowe. Wykonuje algorytm oraz sprawdza dane. Jeśli jest rozbieżność pomiędzy danymi wynikowymi zgłaszany jest wyjątek "Algorytm jest niepoprawny".

7.1.4 Dokumentacja atrybutów składowych

7.1.4.1 template<typename ProblemType > std::vector<BenchmarkData> Benchmark< ProblemType >::m_allBenchmarks [private]

Wektor przechowujący informacje o przeprowadzonych testach.

7.1.4.2 template<typename ProblemType > ProblemType Benchmark< ProblemType >::m_problem [private]

Przechowuje obiekt, odpowiedzialny za algorytm.

Musi implementować metody, które zostały wspomniane w opisie klasy

7.1.4.3 template<typename ProblemType > unsigned int Benchmark< ProblemType >::m_repeatNum [private]

Dokumentacja dla tej klasy została wygenerowana z pliku:

- /home/mochman/Politechnika/PAMSI/benchmark-multiply-by-2/prj/inc/Benchmark.hpp

7.2 Dokumentacja struktury BenchmarkData

Struktura przechowująca informacje o pojedynczym teście wydajności.

`#include <Benchmark.hpp>`

Atrybuty publiczne

- unsigned int problemSize

  Pole zawierające rozmiar problemu.
- unsigned int number

  Pole przechowujące numer porządkowy testu.
- double usInterval

  Pole przechowujące czas wykonania algorytmu.

7.2.1 Opis szczegółowy

Struktura przechowująca informacje o pojedynczym teście wydajności.

7.2.2 Dokumentacja atrybutów składowych

7.2.2.1 unsigned int BenchmarkData::number

Pole przechowujące numer porządkowy testu.

Jest to liczba określająca, który raz jest testowany algorytm z takim samym rozmiarem problemu

7.2.2.2 unsigned int BenchmarkData::problemSize

Pole zawierające rozmiar problemu.

Z reguły jest to ilość danych wejściowych np. ilość liczb do posortowania

7.2.2.3 double BenchmarkData::usInterval

Pole przechowujące czas wykonania algorytmu.

Jednostką interwału jest mikrosekunda

Dokumentacja dla tej struktury została wygenerowana z pliku:

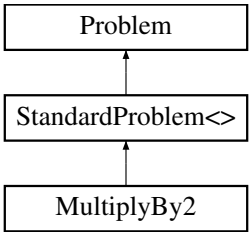
- /home/mochman/Politechnika/PAMSI/benchmark-multiply-by-2/prj/inc/Benchmark.hpp

7.3 Dokumentacja klasy MultiplyBy2

Klasa reprezentuje algorytm mnożenia tablicy przez 2.

`#include <MultiplyBy2.hpp>`

Diagram dziedziczenia dla MultiplyBy2



Metody publiczne

- void compute ()

  Przemnaża tablicę wejściową przez 2.
- ~MultiplyBy2 ()=default

Dodatkowe Dziedziczone Składowe

7.3.1 Opis szczegółowy

Klasa reprezentuje algorytm mnożenia tablicy przez 2.

7.3.2 Dokumentacja konstruktora i destruktora

7.3.2.1 MultiplyBy2::~MultiplyBy2 ( ) [default]

7.3.3 Dokumentacja funkcji składowych

7.3.3.1 void MultiplyBy2::compute ( ) [virtual]

Przemnaża tablicę wejściową przez 2.

Metoda wymnaża każdy element przez dwa i zapisuje wynik do tablicy wyjściowej

Implementuje StandardProblem<>.

Dokumentacja dla tej klasy została wygenerowana z plików:

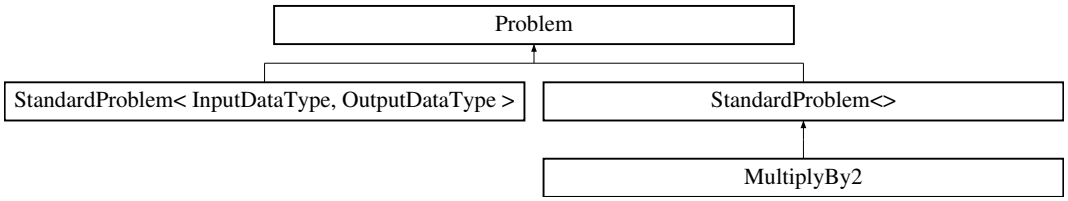
- /home/mochman/Politechnika/PAMSI/benchmark-multiply-by-2/prj/inc/MultiplyBy2.hpp
- /home/mochman/Politechnika/PAMSI/benchmark-multiply-by-2/prj/src/MultiplyBy2.cpp

7.4 Dokumentacja klasy Problem

Abstrakcyjna klasa reprezentująca problem algorytmiczny.

`#include <Problem.hpp>`

Diagram dziedziczenia dla Problem



Metody publiczne

- virtual void readInData (std::istream &is=std::cin)=0

  Wczytuje dane wejściowe algorytmu.
- virtual void readOutData (std::istream &is=std::cin)=0

  Wczytuje poprawne dane wyjściowe algorytmu.
- virtual bool isCorrect () const =0

  Sprawdza czy wynik algorytmu jest poprawny.

- virtual void compute ()=0

  Rozpoczyna pracę algorytmu.
- virtual unsigned int problemSize () const =0

  Pozwala pobrać rozmiar problemu.
- virtual void problemSize (unsigned int size)=0

  Pozwala ustawić rozmiar problemu.
- virtual void clearData ()=0

  Usuwa wszystkie dane.
- virtual ~Problem ()=default

7.4.1 Opis szczegółowy

Abstrakcyjna klasa reprezentująca problem algorytmiczny.

Definiuje interfejs podstawowych operacji dla problemu tzn. wczytanie danych, przeprowadzenie obliczeń, sprawdzenie poprawności algorytmu

7.4.2 Dokumentacja konstruktora i destruktora

7.4.2.1 virtual Problem::~Problem ( ) [virtual], [default]

7.4.3 Dokumentacja funkcji składowych

7.4.3.1 virtual void Problem::clearData ( ) [pure virtual]

Usuwa wszystkie dane.

Pozbywa się z pamięci wszyskich danych wejsciowych, poprawnych wyjsciowych itp

Implementowany w StandardProblem< InputDataType, OutputDataType > i StandardProblem<>.

7.4.3.2 virtual void Problem::compute ( ) [pure virtual]

Rozpoczyna pracę algorytmu.

Implementowany w StandardProblem< InputDataType, OutputDataType >, StandardProblem<> i MultiplyBy2.

7.4.3.3 virtual bool Problem::isCorrect ( ) const [pure virtual]

Sprawdza czy wynik algorytmu jest poprawny.

Dokonuje porównania wczytanych poprawnych danych z wynikiem algorytmu

Implementowany w StandardProblem< InputDataType, OutputDataType > i StandardProblem<>.

7.4.3.4 virtual unsigned int Problem::problemSize ( ) const [pure virtual]

Pozwala pobrać rozmiar problemu.

Implementowany w StandardProblem< InputDataType, OutputDataType > i StandardProblem<>.

7.4.3.5 virtual void Problem::problemSize (unsigned int *size*) [pure virtual]

Pozwala ustawić rozmiar problemu.

Implementowany w StandardProblem< InputDataType, OutputDataType > i StandardProblem<>.

7.4.3.6 virtual void Problem::readInData (std::istream & *is* = `std::cin`) [pure virtual]

Wczytuje dane wejściowe algorytmu.

Parametry

is	- strumień z którego mają zostać wczytane dane

Implementowany w StandardProblem< InputDataType, OutputDataType > i StandardProblem<>.

7.4.3.7 virtual void Problem::readOutData (std::istream & *is* = `std::cin`) [pure virtual]

Wczytuje poprawne dane wyjściowe algorytmu.

Parametry

is	- strumień z którego mają zostać wczytane dane

Implementowany w StandardProblem< InputDataType, OutputDataType > i StandardProblem<>.

Dokumentacja dla tej klasy została wygenerowana z pliku:

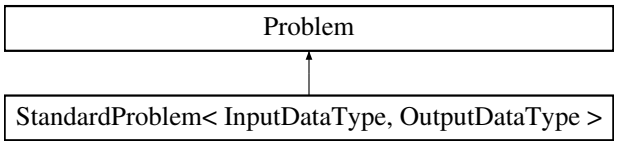
- /home/mochman/Politechnika/PAMSI/benchmark-multiply-by-2/prj/inc/Problem.hpp

7.5 Dokumentacja szablonu klasy StandardProblem< InputDataType, OutputDataType >

Klasa definiuje standardowy problem algorytmiczny.

`#include <StandardProblem.hpp>`

Diagram dziedziczenia dla StandardProblem< InputDataType, OutputDataType >



Metody publiczne

- virtual void readInData (std::istream &is=std::cin)
  Patrz Problem::readInData.
- virtual void readOutData (std::istream &is=std::cin)
  Patrz Problem::readOutData.
- virtual bool isCorrect () const
  Patrz Problem::isCorrect.
- virtual void compute ()=0
  Patrz Problem::compute.
- virtual unsigned int problemSize () const
  Patrz Problem::problemSize.
- virtual void problemSize (unsigned int size)
  Patrz Problem::problemSize(unsigned int)
- virtual void clearData ()
  Usuwa wszystkie dane.

Atrybuty chronione

- std::vector< InputDataType > m_inputData

  Wektor przechowujący dane wejściowe.
- std::vector< OutputDataType > m_outputData

  Wektor przechowujący dane wyjściowe.
- std::vector< OutputDataType > m_correctOutputData

  Wektor przechowujący poprawne dane wyjściowe.

Atrybuty prywatne

- unsigned int m_problemSize

  Pole przechowujące rozmiar problemu.

7.5.1 Opis szczegółowy

template<typename InputDataType = double, typename OutputDataType = double>class StandardProblem< InputDataType, OutputDataType >

Klasa definiuje standardowy problem algorytmiczny.

Poprzez standardowy problem algorytmiczny rozumiem problem, w którym wczytuje się rozmiar problemu a następnie tablicę tego samego typu. Wynikiem jest zbiór liczb tego samego typu o tej samej liczności Przykładem może być np. mnożenie tablicy przez stałą, sortowanie itp.

7.5.2 Dokumentacja funkcji składowych

7.5.2.1 template<typename InputDataType , typename OutputDataType > void StandardProblem< InputDataType, OutputDataType >::clearData ( ) [virtual]

Usuwa wszystkie dane.

Pozbywa się z pamięci wszyskich danych wejsciowych, poprawnych wyjsciowych itp

Implementuje Problem.

7.5.2.2 template<typename InputDataType = double, typename OutputDataType = double> virtual void StandardProblem< InputDataType, OutputDataType >::compute ( ) [pure virtual]

Patrz Problem::compute.

Implementuje Problem.

Implementowany w MultiplyBy2.

7.5.2.3 template<typename InputDataType , typename OutputDataType > bool StandardProblem< InputDataType, OutputDataType >::isCorrect ( ) const [virtual]

Patrz Problem::isCorrect.

Implementuje Problem.

7.5.2.4 template<typename InputDataType = double, typename OutputDataType = double> virtual unsigned int StandardProblem< InputDataType, OutputDataType >::problemSize ( ) const [inline], [virtual]

Patrz Problem::problemSize.

Implementuje Problem.

7.5.2.5 template<typename InputDataType = double, typename OutputDataType = double> virtual void StandardProblem< InputDataType, OutputDataType >::problemSize (unsigned int *size*) [inline], [virtual]

Patrz Problem::problemSize(unsigned int)

Dodatkowo zmiana powoduje wyczyszczenie wszystkich danych oraz rezerwację pamięci o odpowiedniej wielkości

Implementuje Problem.

7.5.2.6 template<typename InputDataType , typename OutputDataType > void StandardProblem< InputDataType, OutputDataType >::readInData (std::istream & *is* = std::cin) [virtual]

Patrz Problem::readInData.

Implementuje Problem.

7.5.2.7 template<typename InputDataType , typename OutputDataType > void StandardProblem< InputDataType, OutputDataType >::readOutData (std::istream & *is* = std::cin) [virtual]

Patrz Problem::readOutData.

Implementuje Problem.

7.5.3 Dokumentacja atrybutów składowych

7.5.3.1 template<typename InputDataType = double, typename OutputDataType = double> std::vector<OutputDataType> StandardProblem< InputDataType, OutputDataType >::m_correctOutputData [protected]

Wektor przechowujący poprawne dane wyjściowe.

Z tymi dany jest porównywane wyjście algorytmu i sprawdzana poprawność algorytmu.

7.5.3.2 template<typename InputDataType = double, typename OutputDataType = double> std::vector<InputDataType> StandardProblem< InputDataType, OutputDataType >::m_inputData [protected]

Wektor przechowujący dane wejściowe.

7.5.3.3 template<typename InputDataType = double, typename OutputDataType = double> std::vector<OutputDataType> StandardProblem< InputDataType, OutputDataType >::m_outputData [protected]

Wektor przechowujący dane wyjściowe.

Są to dane wygenerowane poprzez metodę compute()

7.5.3.4 template<typename InputDataType = double, typename OutputDataType = double> unsigned int StandardProblem< InputDataType, OutputDataType >::m_problemSize [private]

Pole przechowujące rozmiar problemu.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- /home/mochman/Politechnika/PAMSI/benchmark-multiply-by-2/prj/inc/StandardProblem.hpp

7.6 Dokumentacja klasy Timer

Klasa mierząca długość interwału czasu.

`#include <Timer.hpp>`

Metody publiczne

- double nsInterval () const

  Oblicza długość odcinka czasu w nanosekundach.
- double usInterval () const

  Oblicza długość odcinka czasu w mikrosekundach.
- double msInterval () const

  Oblicza długość odcinka czasu w milisekundach.
- double sInterval () const

  Oblicza długość odcinka czasu w sekundach.
- void start ()

  Zaczyna odmierzać czas.
- void stop ()

  Kończy odmierzać czas.

Statyczne metody publiczne

- static double precision ()

  Zwraca precyzję zegara w sekundach.

Atrybuty prywatne

- std::chrono::high_resolution_clock::time_point m_start

  Przechowuje informacje o chwili od której odliczać czas.
- std::chrono::high_resolution_clock::time_point m_end

  Przechowuje informacje o chwili do której odliczać czas.

7.6.1 Opis szczegółowy

Klasa mierząca długość interwału czasu.

Do funkcjonowania używa high_resolution_clock z z biblioteki standardowej C++.

7.6.2 Dokumentacja funkcji składowych

7.6.2.1 double Timer::msInterval ( ) const

Oblicza długość odcinka czasu w milisekundach.

Jeśli nie została użyta metoda start() to czas liczony jest od tzw epoch. Jeśli zostanie użyta metoda end() a start() nie to pojawi się ujemny czas liczony od epoch

7.6.2.2 double Timer::nsInterval ( ) const

Oblicza długość odcinka czasu w nanosekundach.

Jeśli nie została użyta metoda start() to czas liczony jest od tzw epoch. Jeśli zostanie użyta metoda end() a start() nie to pojawi się ujemny czas liczony od epoch UWAGA! Nie na każdym systemie dokładność czasu jest tak duża!

7.6.2.3 double Timer::precision ( ) [static]

Zwraca precyzję zegara w sekundach.

7.6.2.4 double Timer::sInterval ( ) const

Oblicza długość odcinka czasu w sekundach.

Jeśli nie została użyta metoda start() to czas liczony jest od tzw epoch. Jeśli zostanie użyta metoda end() a start() nie to pojawi się ujemny czas liczony od epoch

7.6.2.5 void Timer::start ( ) [inline]

Zaczyna odmierzać czas.

Metoda zapisuje do prywatnego pola aktualną chwilę, od której ma być liczony czas

7.6.2.6 void Timer::stop ( ) [inline]

Kończy odmierzać czas.

Metoda zapisuje do prywatnego pola aktualną chwilę, do której ma być liczony czas

7.6.2.7 double Timer::usInterval ( ) const

Oblicza długość odcinka czasu w mikrosekundach.

Jeśli nie została użyta metoda start() to czas liczony jest od tzw epoch. Jeśli zostanie użyta metoda end() a start() nie to pojawi się ujemny czas liczony od epoch

7.6.3 Dokumentacja atrybutów składowych

7.6.3.1 std::chrono::high_resolution_clock::time_point Timer::m_end [private]

Przechowuje informacje o chwili do której odliczać czas.

7.6.3.2 std::chrono::high_resolution_clock::time_point Timer::m_start [private]

Przechowuje informacje o chwili od której odliczać czas.

Dokumentacja dla tej klasy została wygenerowana z plików:

- /home/mochman/Politechnika/PAMSI/benchmark-multiply-by-2/prj/inc/Timer.hpp
- /home/mochman/Politechnika/PAMSI/benchmark-multiply-by-2/prj/src/Timer.cpp

Rozdział 8

Dokumentacja plików

8.1 Dokumentacja pliku /home/mochman/Politechnika/PAMSI/benchmark-multiply-by-2/prj/doxygen/MainPage.dox

8.2 Dokumentacja pliku /home/mochman/Politechnika/PAMSI/benchmark-multiply-by-2/prj/inc/Benchmark.hpp

```
#include <vector>
#include <iostream>
#include "Timer.hpp"
```

Komponenty

- struct BenchmarkData

  Struktura przechowująca informacje o pojedynczym teście wydajności.
- class Benchmark< ProblemType >

  Klasa pozwalająca na testowanie algorytmów.

Funkcje

- std::ostream & operator<< (std::ostream &os, BenchmarkData &benchData)

  Pozwala wyświetlić zawartość struktury.

8.2.1 Dokumentacja funkcji

8.2.1.1 std::ostream & operator<< (std::ostream & *os,* BenchmarkData & *benchData*)

Pozwala wyświetlić zawartość struktury.

Format jest następujący: rozmiar problemu,numer porządkowy,czas wykonania

8.3 Dokumentacja pliku /home/mochman/Politechnika/PAMSI/benchmark-multiply-by-2/prj/inc/MultiplyBy2.hpp

```
#include "StandardProblem.hpp"
```

Komponenty

- class MultiplyBy2

  Klasa reprezentuje algorytm mnożenia tablicy przez 2.

8.4 Dokumentacja pliku /home/mochman/Politechnika/PAMSI/benchmark-multiply-by-2/prj/inc/Problem.hpp

```
#include <iostream>
```

Komponenty

- class Problem

  Abstrakcyjna klasa reprezentująca problem algorytmiczny.

8.5 Dokumentacja pliku /home/mochman/Politechnika/PAMSI/benchmark-multiply-by-2/prj/inc/StandardProblem.hpp

```
#include <vector>
#include "Problem.hpp"
```

Komponenty

- class StandardProblem< InputDataType, OutputDataType >

  Klasa definiuje standardowy problem algorytmiczny.

8.6 Dokumentacja pliku /home/mochman/Politechnika/PAMSI/benchmark-multiply-by-2/prj/inc/Timer.hpp

```
#include <chrono>
```

Komponenty

- class Timer

  Klasa mierząca długość interwału czasu.

8.7 Dokumentacja pliku /home/mochman/Politechnika/PAMSI/benchmark-multiply-by-2/prj/src/main.cpp

```
#include "Benchmark.hpp"
#include "MultiplyBy2.hpp"
#include <string>
#include <fstream>
#include <thread>
```

Funkcje

- int main (int argc, char ∗argv[])

8.7.1 Dokumentacja funkcji

8.7.1.1 int main (int *argc,* char ∗ *argv[]*)

8.8 Dokumentacja pliku /home/mochman/Politechnika/PAMSI/benchmark-multiply-by-2/prj/src/MultiplyBy2.cpp

```
#include "MultiplyBy2.hpp"
```

8.9 Dokumentacja pliku /home/mochman/Politechnika/PAMSI/benchmark-multiply-by-2/prj/src/Timer.cpp

```
#include "Timer.hpp"
```

8.10 Dokumentacja pliku /home/mochman/Politechnika/PAMSI/benchmark-multiply-by-2/prj/tools/GenerateInput.py

Przestrzenie nazw

- namespace GenerateInput

Funkcje

- def GenerateInput.GenerateInput

Skorowidz