

Optimalizacja działania algorytmów sortowania - Quick Sort

Wygenerowano przez Doxygen 1.8.6

Cz, 16 kwi 2015 11:26:30

Optymalizacja działania algorytmów sortowania - Quick Sort

Mateusz Bencer, nr: 209360

16 kwietnia 2015

Sprawozdanie zawiera wnioski wyciągnięte z optymalizacji algorytmów sortowania - Quick Sort. Celem pomiarów było sprawdzenie wpływu doboru elementu osiowego (ang. pivot) na czas sortowania określonej liczby elementów w kontenerze. Na potrzeby tego zadania użyłem implementacji listy z programu Łukasza Saka. Doświadczenie z pracy nad cudzym kodem okazało się bardzo pouczające...

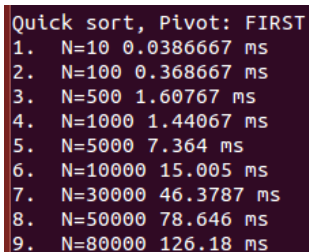
W celu możliwości implementacji algorytmu sortowania konieczne były drobne zmiany w kodzie, tj.:

- Musiałem dodać metody set oraz get do klasy lista, ponieważ niemożliwym było zaimplementowanie algorytmu, gdzie przy pobieraniu elementu został on ściągany, a podczas dodawania inne zostały przesuwane. Metodom pośrednim (sprawdzoną przeze mnie) było pobieranie wszystkich elementów do zwykłej tablicy za pomocą metody pop, posortowanie tej tablicy, i późniejsze powrotne dodanie tych elementów za pomocą metody push do listy. Jednak jak było to do przewidzenia, miało to bardzo negatywny wpływ na złożoność oraz osiągnięte rezultaty czasowe. O wiele prostsza byłaby implementacja tego algorytmu na liście powiązanej (linked list), gdzie każdy element zawiera wskaźnik do poprzedniego i następnego elementu, ale na moment kiedy pobrałem kod z gita, była tam tylko implementacja tablicowa.
- Musiałem dokonać gruntownych zmian w klasie Benchmark, która była stricte dopasowana do poprzedniego zadania (mało uniwersalna).
- Dodałem zaimplementowaną przez siebie wcześniej klasę Timer do pomiarów czasowych.

Zostało przeze mnie przetestowanych 5 sposobów doboru elementu osiowego. Pomiary zostały przeprowadzone dla 9 kolejno rosnących rozmiarach listy. Wszystkie elementy umieszczone w liście są pseudolosowe.

- 1. Pierwszy skrajny element.

Z rozważań teoretycznych wynikało, że jest to najprostsza, ale zarazem najbardziej naiwna metoda doboru elementu osiowego. W przypadku tablicy wstępnie posortowanej taki wybór może skutkować złożonością nawet $O(n^2)$. Moje pomiary (na liście z elementami losowymi) potwierdziły to.

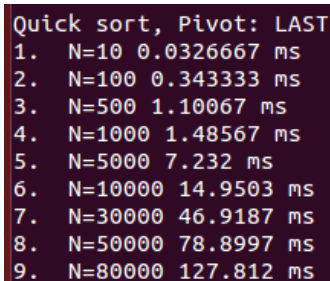


```
Quick sort, Pivot: FIRST
1. N=10 0.0386667 ms
2. N=100 0.368667 ms
3. N=500 1.60767 ms
4. N=1000 1.44067 ms
5. N=5000 7.364 ms
6. N=10000 15.005 ms
7. N=30000 46.3787 ms
8. N=50000 78.646 ms
9. N=80000 126.18 ms
```

Rysunek 1: Wyniki działania programu, gdy pivot jest pierwszym, skrajnym elementem

- 2. Ostatni skrajny element.

Wybór ostatniego elementu jest analogiczny do wyboru pierwszego skaranego elementu. W takim wypadku złożość również może wynieść $O(n^2)$. Moje pomiary również dają podobne rezultaty jak do tych z podpunktu 1.



```
Quick sort, Pivot: LAST
1. N=10 0.0326667 ms
2. N=100 0.343333 ms
3. N=500 1.10067 ms
4. N=1000 1.48567 ms
5. N=5000 7.232 ms
6. N=10000 14.9503 ms
7. N=30000 46.9187 ms
8. N=50000 78.8997 ms
9. N=80000 127.812 ms
```

Rysunek 2: Wyniki działania programu, gdy pivot jest ostatnim, skrajnym elementem

- 3. Element losowy.

Wybór elementu losowego ma na celu uśrednienie możliwości wystąpienia sytuacji najbardziej optymistyczne, jak i najbardziej pesymistycznej. Jest to metoda bardzo zapobiegawcza. Jest złożoność to około $T(n) = 2n \ln(n)$ (przy rozkładzie równomiernym). Jednak według moich pomiarów jest to metoda gorsza od dwóch poprzednich. Podejrzewam, że jest to związane z dodatkowym nakładem czasowym potrzebnym na wylosowanie elementu.

```
Quick sort, Pivot: RANDOM
1. N=10 0.048 ms
2. N=100 0.386667 ms
3. N=500 1.30567 ms
4. N=1000 1.62533 ms
5. N=5000 8.33433 ms
6. N=10000 17.263 ms
7. N=30000 54.5173 ms
8. N=50000 90.7563 ms
9. N=80000 148.93 ms
```

Rysunek 3: Wyniki działania programu, gdy piwot jest elementem losowym

- 4. Element środkowy.

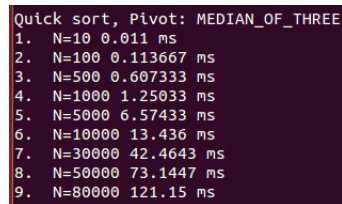
Taki wybór jest chyba najbardziej popularny, przynajmniej wśród implementacji quick sorta znalezionych przeze mnie w internecie. Zapobiega on występowaniu (w prawie wszystkich przypadkach) sytuacji najbardziej pesymistycznej, a wyliczenie elementu środkowego wymaga małych nakładów obliczeniowych. Mimo wszystko uzyskane przeze mnie wyniki są podobne do tych z wyborem elementu pierwszego i ostatniego.

```
Quick sort, Pivot: CENTER
1. N=10 0.0123333 ms
2. N=100 0.133333 ms
3. N=500 0.683333 ms
4. N=1000 1.39167 ms
5. N=5000 7.38 ms
6. N=10000 14.9877 ms
7. N=30000 46.4457 ms
8. N=50000 78.272 ms
9. N=80000 129.781 ms
```

Rysunek 4: Wyniki działania programu, gdy piwot jest elementem środkowym

- 5. Mediana z trzech.

Metoda ta polega na wyliczeniu mediany z elementu najmniejszego, największego i środkowego i jego wyborze, jako elementu osiowego. Idealną sytuacją byłoby za każdym razem obliczanie mediany z całej listy, jednak takie podejście wymaga w praktyce wstępnie posortowanej tablicy do wyznaczenia mediany... (co oczywiście jest sprzeczne z celowością sortowania). Kompromisem dającym dobre rezultaty jest właśnie wyznaczenie mediany z trzech wartości. Według moich pomiarów jest to metoda najszybsza.

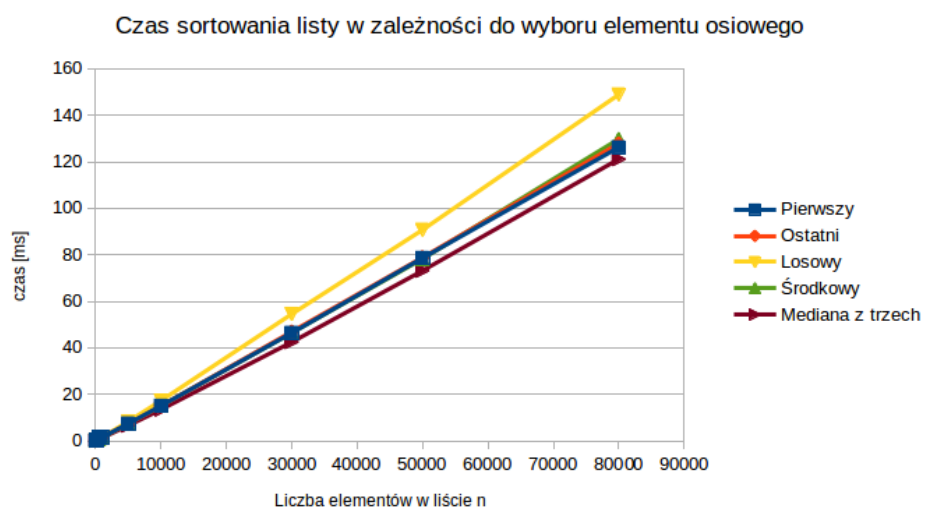


N	Time (ms)
10	0.011
100	0.113667
500	0.607333
1000	1.25033
5000	6.57433
10000	13.436
30000	42.4643
50000	73.1447
80000	121.15

Rysunek 5: Wyniki działania programu, gdy piwot jest medianą z trzech elementów

Dla lepszego zobrazowania otrzymanych wyników umieszczam jeszcze wykres ze wszystkimi metodami.

Możemy zaobserwować, że wszystkie metody dają dość podobne rezultaty.



Rysunek 6: Wykres przedstawiający czasy sortowania, przy różnym wyborze piwota

Dla liczb losowych najlepsza jednak (według mojej implementacji) wydaje się być metoda mediany z trzech elementów.

Spis treści

1 Indeks hierarchiczny	1
1.1 Hierarchia klas	1
2 Indeks klas	3
2.1 Lista klas	3
3 Indeks plików	5
3.1 Lista plików	5
4 Dokumentacja klas	7
4.1 Dokumentacja szablonu klasy Benchmark< T >	7
4.1.1 Opis szczegółowy	7
4.1.2 Dokumentacja konstruktora i destruktora	7
4.1.2.1 Benchmark	7
4.2 Dokumentacja szablonu klasy Kolejka< TYP >	8
4.2.1 Opis szczegółowy	8
4.3 Dokumentacja szablonu klasy Lista< TYP >	8
4.3.1 Opis szczegółowy	9
4.4 Dokumentacja szablonu klasy Stos< TYP >	9
4.4.1 Opis szczegółowy	9
4.5 Dokumentacja klasy Timer	10
4.5.1 Opis szczegółowy	10
4.5.2 Dokumentacja konstruktora i destruktora	10
4.5.2.1 Timer	10
4.5.3 Dokumentacja funkcji składowych	10
4.5.3.1 diffTimeMs	10
4.5.3.2 startTimer	11
4.5.3.3 stopTimer	11
5 Dokumentacja plików	13
5.1 Dokumentacja pliku /home/mateusz/git/209365/Lab3/prj/inc/Benchmark.hh	13
5.1.1 Opis szczegółowy	13
5.2 Dokumentacja pliku /home/mateusz/git/209365/Lab3/prj/inc/Kolejka.hh	13

5.2.1	Opis szczegółowy	14
5.3	Dokumentacja pliku /home/mateusz/git/209365/Lab3/prj/inc/Lista.hh	14
5.3.1	Opis szczegółowy	14
5.4	Dokumentacja pliku /home/mateusz/git/209365/Lab3/prj/inc/Stos.hh	14
5.4.1	Opis szczegółowy	14
5.5	Dokumentacja pliku /home/mateusz/git/209365/Lab3/prj/inc/Timer.hh	14
5.5.1	Opis szczegółowy	15
5.6	Dokumentacja pliku /home/mateusz/git/209365/Lab3/prj/src/Timer.cpp	15
5.6.1	Opis szczegółowy	15
Indeks		16

Rozdział 1

Indeks hierarchiczny

1.1 Hierarchia klas

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

Benchmark< T >	7
Lista< TYP >	8
Kolejka< TYP >	8
Stos< TYP >	9
Timer	10

Rozdział 2

Indeks klas

2.1 Lista klas

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

Benchmark< T >	Klasa do przeprowadzenia testów na kontenerach danych	7
Kolejka< TYP >	Klasa Kolejka	8
Lista< TYP >	Klasa Lista	8
Stos< TYP >	Klasa Stos	9
Timer	Klasa do pomiaru różnicy czasów	10

Rozdział 3

Indeks plików

3.1 Lista plików

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

/home/mateusz/git/209365/Lab3/prj/inc/ Benchmark.hh	
Deklaracja i definicja (razem, bo szablon) klasy Benchmark	13
/home/mateusz/git/209365/Lab3/prj/inc/ Kolejka.hh	
Definicja klasy Kolejka	13
/home/mateusz/git/209365/Lab3/prj/inc/ Lista.hh	
Definicja klasy Lista	14
/home/mateusz/git/209365/Lab3/prj/inc/ Pivot.hh	??
/home/mateusz/git/209365/Lab3/prj/inc/ quick_sort.hh	??
/home/mateusz/git/209365/Lab3/prj/inc/ Stos.hh	
Definicja klasy Stos	14
/home/mateusz/git/209365/Lab3/prj/inc/ Timer.hh	
Plik zawierający deklaracje klasy Timer służącej do pomiaru różnicy czasów	14
/home/mateusz/git/209365/Lab3/prj/src/ Timer.cpp	
Plik zawierający definicje funkcji klasy Timer służącej do pomiaru różnicy czasów	15

Rozdział 4

Dokumentacja klas

4.1 Dokumentacja szablonu klasy `Benchmark< T >`

Klasa do przeprowadzenia testów na kontenerach danych.

```
#include <Benchmark.hh>
```

Metody publiczne

- `Benchmark` (unsigned int *test_list_sizes, unsigned int size, unsigned int number_of_memeasurements)
Konstruktor klasy `Benchmark` pomiarający wykładnik liczby określającej dla jakiej liczby elementów testujemy kontener.
- void `testQuickSort` (`Lista< T >` *list, PIVOT piv)

4.1.1 Opis szczegółowy

```
template<class T>class Benchmark< T >
```

Klasa do przeprowadzenia testów na kontenerach danych.

Klasa odpowiada za pomiar czasu koniecznego do umieszczenia określonej liczby danych na stosie, kolejce oraz liście

4.1.2 Dokumentacja konstruktora i destruktora

4.1.2.1 `template<typename T > Benchmark< T >::Benchmark (unsigned int * test_list_sizes, unsigned int size, unsigned int number_of_memeasurements)`

Konstruktor klasy `Benchmark` pomiarający wykładnik liczby określającej dla jakiej liczby elementów testujemy kontener.

Parametry

<i>testPower</i>	Liczba określająca wykładnik 10. Cała liczba ($10^{(\text{testPower})}$) określa liczbę elementów, które będą wkładane do kontenerów.
<i>Liczby</i>	w tablicy określają jaką ilość danych (jak dużą listę) będziemy kolejno sortować (w celu pomiaru potrzebnego czasu dla każdej wielkości)

Dokumentacja dla tej klasy została wygenerowana z pliku:

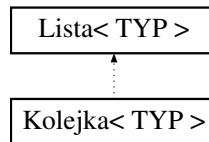
- `/home/mateusz/git/209365/Lab3/prj/inc/Benchmark.hh`

4.2 Dokumentacja szablonu klasy Kolejka< TYP >

Klasa [Kolejka](#).

```
#include <Kolejka.hh>
```

Diagram dziedziczenia dla Kolejka< TYP >



Metody publiczne

- void **PUSH** (TYP liczba)
- int **POP** ()
- void **SHOW** ()
- unsigned int **SIZE** ()

4.2.1 Opis szczegółowy

```
template<typename TYP>class Kolejka< TYP >
```

Klasa [Kolejka](#).

Klasa ta modeluje nam Kolejke Składa się z pól klasy [Lista](#) oraz metod PUSH, POP, SIZE, SHOW Klasa w calosci wykorzystuje implementacje listy

Dokumentacja dla tej klasy została wygenerowana z pliku:

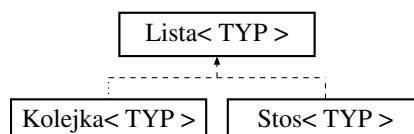
- </home/mateusz/git/209365/Lab3/prj/inc/Kolejka.hh>

4.3 Dokumentacja szablonu klasy Lista< TYP >

Klasa [Lista](#).

```
#include <Lista.hh>
```

Diagram dziedziczenia dla Lista< TYP >



Metody publiczne

- void **Rozmiar** ()
- void **PUSH** (TYP liczba, unsigned int index)
- void **SET** (TYP liczba, unsigned int index)
- void **Powiekszenie_Pamieci** ()
- TYP **POP** (unsigned int index)

- void **Zmniejszenie_Pamieci** ()
- TYP **GET** (unsigned int index)
- unsigned int **SIZE** ()
- void **SHOW** ()

4.3.1 Opis szczegółowy

```
template<typename TYP>class Lista< TYP >
```

Klasa [Lista](#).

Klasa ta modeluje nam Listę wartości typu TYP Składa się z pól:

Parametry

in	<i>*tab</i>	- tablica naszych liczb;
in	<i>poczatek</i>	- pierwsza liczba w naszej tablicy
in	<i>koniec</i>	- ostatnia liczba w naszej tablicy
in	<i>_rozmiar_listy</i>	- rozmiar stworzonej tablicy dynamicznej

Dokumentacja dla tej klasy została wygenerowana z pliku:

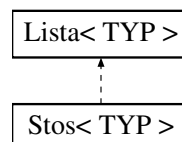
- /home/mateusz/git/209365/Lab3/prj/inc/[Lista.hh](#)

4.4 Dokumentacja szablonu klasy Stos< TYP >

Klasa [Stos](#).

```
#include <Stos.hh>
```

Diagram dziedziczenia dla Stos< TYP >



Metody publiczne

- void **PUSH** (TYP liczba)
- int **POP** ()
- void **SHOW** ()
- unsigned int **SIZE** ()

Dodatkowe Dziedziczone Składowe

4.4.1 Opis szczegółowy

```
template<typename TYP>class Stos< TYP >
```

Klasa [Stos](#).

Klasa ta modeluje nam [Stos](#) Składa się z pól klasy [Lista](#) które zostaną użyte oraz metod PUSH, POP, SIZE, SHOW Klasa w całości wykorzystuje implementację listy

Dokumentacja dla tej klasy została wygenerowana z pliku:

- </home/mateusz/git/209365/Lab3/prj/inc/Stos.hh>

4.5 Dokumentacja klasy Timer

Klasa do pomiaru różnicy czasów.

```
#include <Timer.hh>
```

Metody publiczne

- [Timer](#) ()
Konstruktor zerujący parametry.
- void [startTimer](#) ()
Zmierzenie czasu rozpoczęcia pomiaru.
- void [stopTimer](#) ()
Zmierzenie czasu zakończenia pomiaru.
- double [diffTimeMs](#) ()
Funkcja zwracająca różnicę czasu pomiędzy czasem rozpoczęcia i zakończenia pomiaru.

4.5.1 Opis szczegółowy

Klasa do pomiaru różnicy czasów.

Klasa pozwala na pomiar czasów w danych momentach oraz na zwrócenie czasu, który upłynął pomiędzy tymi momentami

4.5.2 Dokumentacja konstruktora i destruktor

4.5.2.1 `Timer::Timer ()`

Konstruktor zerujący parametry.

Konstruktor ten odpowiada za zerowania zmiennych startu i stopu w celu możliwości późniejszego sprawdzenia, czy pomiary czasu konieczne do wyznaczenia różnicy zostały zrealizowane.

4.5.3 Dokumentacja funkcji składowych

4.5.3.1 `double Timer::diffTimeMs ()`

Funkcja zwracająca różnicę czasu pomiędzy czasem rozpoczęcia i zakończenia pomiaru.

Różnica czasu zwracana jest w milisekundach.

Warunek wstępny

Czas zakończenia pomiaru musi być większy (późniejszy) od czasu jego rozpoczęcia

Zwraca

Zwracana jest różnica czasu zrzutowana do typu double

4.5.3.2 void Timer::startTimer ()

Zmierzenie czasu rozpoczęcia pomiaru.

Funkcja zapamiętuje bieżący czas, jako czas rozpoczęcia pomiaru.

4.5.3.3 void Timer::stopTimer ()

Zmierzenie czasu zakończenia pomiaru.

Funkcja zapamiętuje bieżący czas, jako czas zakończenia pomiaru.

Dokumentacja dla tej klasy została wygenerowana z plików:

- </home/mateusz/git/209365/Lab3/prj/inc/Timer.hh>
- </home/mateusz/git/209365/Lab3/prj/src/Timer.cpp>

Rozdział 5

Dokumentacja plików

5.1 Dokumentacja pliku /home/mateusz/git/209365/Lab3/prj/inc/Benchmark.hh

Deklaracja i definicja (razem, bo szablon) klasy [Benchmark](#).

```
#include "../inc/Lista.hh"
#include "../inc/quick_sort.hh"
#include "../inc/Timer.hh"
#include <iostream>
#include <cstdlib>
#include <cstdio>
#include <ctime>
#include <string>
```

Komponenty

- class [Benchmark< T >](#)

Klasa do przeprowadzenia testów na kontenerach danych.

5.1.1 Opis szczegółowy

Deklaracja i definicja (razem, bo szablon) klasy [Benchmark](#). [Benchmark.hh](#)

5.2 Dokumentacja pliku /home/mateusz/git/209365/Lab3/prj/inc/Kolejka.hh

Definicja klasy [Kolejka](#).

```
#include <iostream>
#include "Lista.hh"
```

Komponenty

- class [Kolejka< TYP >](#)

Klasa Kolejka.

5.2.1 Opis szczegółowy

Definicja klasy [Kolejka](#). Plik zawiera definicje klasy [Kolejka](#), która będzie strukturą naszych danych. Klasa ta posiada szablony, dzięki czemu możemy pracować na różnych typach danych

5.3 Dokumentacja pliku /home/mateusz/git/209365/Lab3/prj/inc/Lista.hh

Definicja klasy [Lista](#).

```
#include <iostream>
```

Komponenty

- class [Lista](#)< TYP >
Klasa [Lista](#).

5.3.1 Opis szczegółowy

Definicja klasy [Lista](#). Plik zawiera definicje klasy [Lista](#) która będzie strukturą danych opartą na tablicy dynamicznej

5.4 Dokumentacja pliku /home/mateusz/git/209365/Lab3/prj/inc/Stos.hh

Definicja klasy [Stos](#).

```
#include <iostream>
#include "Lista.hh"
```

Komponenty

- class [Stos](#)< TYP >
Klasa [Stos](#).

5.4.1 Opis szczegółowy

Definicja klasy [Stos](#). Plik zawiera definicje klasy [Stos](#), która będzie strukturą naszych danych. Klasa ta posiada szablony, dzięki czemu możemy pracować na różnych typach danych

5.5 Dokumentacja pliku /home/mateusz/git/209365/Lab3/prj/inc/Timer.hh

Plik zawierający deklaracje klasy [Timer](#) służącej do pomiaru różnicy czasów.

```
#include <ctime>
```

Komponenty

- class [Timer](#)
Klasa do pomiaru różnicy czasów.

5.5.1 Opis szczegółowy

Plik zawierający deklaracje klasy [Timer](#) służącej do pomiaru różnicy czasów. [Timer.h](#)

5.6 Dokumentacja pliku /home/mateusz/git/209365/Lab3/prj/src/Timer.cpp

Plik zawierający definicje funkcji klasy [Timer](#) służącej do pomiaru różnicy czasów.

```
#include "../inc/Timer.hh"  
#include <iostream>
```

5.6.1 Opis szczegółowy

Plik zawierający definicje funkcji klasy [Timer](#) służącej do pomiaru różnicy czasów. [Timer.cpp](#)

Skorowidz

/home/mateusz/git/209365/Lab3/prj/inc/Benchmark.hh,
[13](#)

/home/mateusz/git/209365/Lab3/prj/inc/Kolejka.hh, [13](#)

/home/mateusz/git/209365/Lab3/prj/inc/Lista.hh, [14](#)

/home/mateusz/git/209365/Lab3/prj/inc/Stos.hh, [14](#)

/home/mateusz/git/209365/Lab3/prj/inc/Timer.hh, [14](#)

/home/mateusz/git/209365/Lab3/prj/src/Timer.cpp, [15](#)

Benchmark

Benchmark, [7](#)

Benchmark< T >, [7](#)

diffTimeMs

Timer, [10](#)

Kolejka< TYP >, [8](#)

Lista< TYP >, [8](#)

startTimer

Timer, [10](#)

stopTimer

Timer, [11](#)

Stos< TYP >, [9](#)

Timer, [10](#)

diffTimeMs, [10](#)

startTimer, [10](#)

stopTimer, [11](#)

Timer, [10](#)