

Struktury danych i złożoność Obliczeniowa

Zadanie projektowe nr 1: Badanie efektywności operacji dodawania, usuwania oraz wyszukiwania elementów w różnych strukturach danych.



5 kwietnia 2017

Łukasz Broll

225972

Spis treści

[Tablica 2](#_Toc479105230)

[usuwanie i dodawanie 2](#_Toc479105231)

[początek tablicy 2](#_Toc479105232)

[koniec tablicy 2](#_Toc479105233)

[dowolne miejsce w tablicy 2](#_Toc479105234)

[wyszukiwanie 2](#_Toc479105235)

[Lista dwukierunkowa 2](#_Toc479105236)

[usuwanie i dodawanie 2](#_Toc479105237)

[początek/koniec listy 2](#_Toc479105238)

[dowolne miejsce na liście 3](#_Toc479105239)

[wyszukiwanie 3](#_Toc479105240)

[Kopiec 3](#_Toc479105241)

[usuwanie i dodawanie elementu wg klucza 3](#_Toc479105242)

[wyszukiwanie 3](#_Toc479105243)

[Operacje na tablicy 4](#_Toc479105244)

[Usuwanie z początku tablicy 4](#_Toc479105245)

[Usuwanie z końca tablicy 4](#_Toc479105246)

[Usuwanie losowego elementu tablicy 4](#_Toc479105247)

[Dodawanie elementu na początek tablicy 5](#_Toc479105248)

[Dodawanie elementu na koniec tablicy 5](#_Toc479105249)

[Dodawanie w losowym miejscu elementu w tablicy 5](#_Toc479105250)

[Wyszukiwanie w tablicy 6](#_Toc479105251)

[Operacje na liście dwukierunkowej 6](#_Toc479105252)

[Usuwanie pierwszego elementu listy 6](#_Toc479105253)

[Usuwanie ostatniego elementu listy 6](#_Toc479105254)

[Usuwanie losowego elementu listy 7](#_Toc479105255)

[Dodawanie elementu na początek listy 7](#_Toc479105256)

[Dodawanie elementu na koniec listy 7](#_Toc479105257)

[Dodanie elementu za losowym indeksem w liście 8](#_Toc479105258)

[Wyszukiwanie elementu w liście 8](#_Toc479105259)

[Operacje na kopcu 8](#_Toc479105260)

[Usuwanie elementów z kopca 8](#_Toc479105261)

[Dodawanie elementów do kopca 9](#_Toc479105262)

[Wyszukiwanie elementów w kopcu 9](#_Toc479105263)

[operacje na drzewie RBT 9](#_Toc479105264)

[usuwanie elementów z drzewa 9](#_Toc479105265)

[dodawanie elementów do drzewa 10](#_Toc479105266)

[wyszukiwanie elementu w drzewie 10](#_Toc479105267)

Wstęp teoretyczny

Czasową złożoność obliczeniową określamy jako ilość czasu niezbędnego do rozwiązania problemu w zależności od liczby danych wejściowych. Jest to jeden z najważniejszych parametrów charakteryzujących algorytm. Decyduje on o efektywności całego programu. Podstawowymi zasobami systemowymi uwzględnianymi w analizie algorytmów są czas działania oraz obszar zajmowanej pamięci. Na złożoność czasową składają się dwie wartości: **pesymistyczna**, czyli taka, która charakteryzuje najgorszy przypadek działania oraz **oczekiwana**. Złożoność czasową oznaczamy jako O(n).

# Tablica

## usuwanie i dodawanie

### początek tablicy

Implementacja dodawania na początek tablicy polega na przypisaniu do indeksu zerowego nowej wartości i przekopiowaniu starej tablicy. Natomiast usuwanie z początku tablicy polega nam relokacji tablicy bez pierwszego elementu. Złożoność czasowa takiego algorytmu to O(n) – zależy wprost proporcjonalnie od ilości elementów. Relokacja, dzięki której korzysta się tylko z wymaganej ilości pamięci nie wpływa na złożoność czasową – czas relokacji jest wprost proporcjonalny do rozmiaru tablicy.

### koniec tablicy

Operacje na końcu tablicy mają stałą złożoność obliczeniową wynoszącą O(1). Z uwagi na bezpośrednią relokację tablicy podczas wykonywania funkcji, zmierzony czas wykonywania algorytmu zależy głównie od rozmiaru tablicy i czasu jej relokacji.

### dowolne miejsce w tablicy

Czas wykonania operacji zależy od położenia modyfikowanego elementu w tablicy. W założeniu pesymistycznym złożoność obliczeniowa wynosi O(n) tj., gdy funkcja losująca indeks zwróci wartość maksymalną - ostatni element tablicy.

## wyszukiwanie

W założeniu pesymistycznym złożoność obliczeniowa wynosi O(n) tj., gdy funkcja losująca indeks zwróci wartość maksymalną - ostatni element tablicy.

# Lista dwukierunkowa

## usuwanie i dodawanie

### początek/koniec listy

Czas usunięcia lub dodania elementu będzie stały – złożoność obliczeniowa O(1), ponieważ wskaźniki na początek (\*head) i koniec (\*tail) listy gwarantują szybkie wykonanie dodawania i usuwania.

### dowolne miejsce na liście

Tak, jak w przypadku listy, czas wykonania operacji zależy od położenia modyfikowanego elementu. W założeniu pesymistycznym złożoność obliczeniowa wynosi O(n) tj., gdy funkcja losująca indeks zwróci wartość maksymalną – algorytm musi wtedy dojść do poprzednika elementu i wykonać odpowiednio wstawienie lub wycięcie.

### wyszukiwanie

W założeniu pesymistycznym złożoność obliczeniowa wynosi O(n) tj., gdy funkcja losująca indeks zwróci wartość maksymalną - ostatni element listy.

# Kopiec

### usuwanie i dodawanie elementu wg klucza

Głównym warunkiem wpływającym na złożoność obliczeniową operacji na kopcu jest spełnienie warunku kopca. Zarówno dodanie, jak i usunięcie elementu może wymagać odtworzenia struktury, zatem w pesymistycznym przypadku złożoność obliczeniowa zależy od liczby poziomów drzewa tj. O(n\*log2n). Jeśli warunek kopca nie jest naruszony, wtedy złożoność czasowa jest stała – dodawany element zostaje na ostatnim liściu, co nie jest regułą w przypadku usuwania.

### wyszukiwanie

Jako, że struktura kopca zapisana jest w tablicy, to w założeniu pesymistycznym złożoność obliczeniowa wynosi O(n) tj., gdy funkcja losująca indeks zwróci wartość maksymalną - ostatni element tablicy.

# Drzewo RBT

Zrównoważona struktura drzewa gwarantuje, że drzewo o n wewnętrznych węzłach nigdy nie będzie miało więcej poziomów niż 2\*lg2(n-1), więc wszystkie operacje wykonywane na drzewie mają pesymistyczną złożoność czasową O(lg2n).

Plan eksperymentu

* Tablica jest dynamiczna i relokowana przy usuwaniu i dodawaniu elementów, a relokacja jest uwzględniana przy każdym pomiarze czasu.
* Tablica elementów kopca jest dynamicznie tworzona z odpowiednim zapasem, ale nie jest relokowana przy każdej operacji.
* Każda operacja jest testowana na strukturze zawierającej odpowiednio 1000, 2000, 5000, 10000 i 20 000 elementów. Dla każdego pomiaru generowana jest nowa populacja. Wyniki ze 100 pomiarów są uśredniane. Wyjątek stanowi drzewo RBT, gdzie dodatkowo przeprowadzono operacje na strukturze 50000, 100000 i 200000, celem lepszej prezentacji wyników.
* Elementem struktur jest liczba typu *integer* generowana losowo z maksymalnego obsługiwanego zakresu < -16383 ; 16383 > za pomocą funkcji *rand() ,*
* Pomiarów czasu dokonano za pomocą funkcji:

*BOOL QueryPerformanceCounter ( \_\_out LARGE\_INTEGER \*lpPerformanceCount );*

*Źródło licznika:* http://cpp0x.pl/forum/temat/?id=21331

* Do pomiaru czasu nie były wliczane funkcje generujące dane.
* Funkcje wyszukujące losowe dane w strukturach zapisywały wyniki tylko dla tych elementów, które znajdowały się w strukturze.
* Wyświetlanie struktury drzewa zostało zrealizowane na gotowym szablonie.

*Źródło szablonu: http://eduinf.waw.pl/inf/alg/001\_search/0112.php*

Wyniki eksperymentu

# Operacje na tablicy

## Usuwanie z początku tablicy

|  |  |
| --- | --- |
| **Rozmiar** | **czas [ms]** |
| 1000 | 0,0487 |
| 2000 | 0,0604 |
| 5000 | 0,1023 |
| 10000 | 0,1677 |
| 20000 | 0,2596 |

## Usuwanie z końca tablicy

|  |  |
| --- | --- |
| **rozmiar** | **czas [ms]** |
| 1000 | 0,0490 |
| 2000 | 0,0624 |
| 5000 | 0,1087 |
| 10000 | 0,1595 |
| 20000 | 0,2695 |

## Usuwanie losowego elementu tablicy

|  |  |
| --- | --- |
| **rozmiar** | **czas [ms]** |
| 1000 | 0,0428 |
| 2000 | 0,0599 |
| 5000 | 0,1176 |
| 10000 | 0,1702 |
| 20000 | 0,2835 |

## Dodawanie elementu na początek tablicy

|  |  |
| --- | --- |
| **rozmiar** | **czas [ms]** |
| 1000 | 0,0471 |
| 2000 | 0,0660 |
| 5000 | 0,0919 |
| 10000 | 0,1447 |
| 20000 | 0,2452 |

## Dodawanie elementu na koniec tablicy

|  |  |
| --- | --- |
| **rozmiar** | **czas[ms]** |
| 1000 | 0,0488 |
| 2000 | 0,0618 |
| 5000 | 0,1073 |
| 10000 | 0,1611 |
| 20000 | 0,2601 |

## Dodawanie w losowym miejscu elementu w tablicy

|  |  |
| --- | --- |
| **rozmiar** | **czas[ms]** |
| 1000 | 0,0458 |
| 2000 | 0,0610 |
| 5000 | 0,1006 |
| 10000 | 0,1376 |
| 20000 | 0,2583 |

## Wyszukiwanie w tablicy

|  |  |
| --- | --- |
| **rozmiar** | **czas[ms]** |
| 1000 | 0,0221 |
| 2000 | 0,0243 |
| 5000 | 0,0398 |
| 10000 | 0,0334 |
| 20000 | 0,0584 |

# Operacje Na liście dwukierunkowej

## Usuwanie pierwszego elementu listy

|  |  |
| --- | --- |
| **rozmiar** | **czas[ms]** |
| 1000 | 0,0281 |
| 2000 | 0,0275 |
| 5000 | 0,0270 |
| 10000 | 0,0273 |
| 20000 | 0,0300 |

## Usuwanie ostatniego elementu listy

|  |  |
| --- | --- |
| **rozmiar** | **czas[ms]** |
| 1000 | 0,0275 |
| 2000 | 0,0281 |
| 5000 | 0,0278 |
| 10000 | 0,0294 |
| 20000 | 0,0272 |

## Usuwanie losowego elementu listy

|  |  |
| --- | --- |
| **rozmiar** | **czas[ms]** |
| 1000 | 0,5338 |
| 2000 | 0,6612 |
| 5000 | 0,6083 |
| 10000 | 0,6168 |
| 20000 | 0,5732 |

|  |  |
| --- | --- |
| **rozmiar** | **czas[ms]** |
| 1000 | 0,0241 |
| 2000 | 0,0308 |
| 5000 | 0,0295 |
| 10000 | 0,0253 |
| 20000 | 0,0242 |

## Dodawanie elementu na początek listy

## Dodawanie elementu na koniec listy

|  |  |
| --- | --- |
| **rozmiar** | **czas[ms]** |
| 1000 | 0,0238 |
| 2000 | 0,0243 |
| 5000 | 0,0248 |
| 10000 | 0,0233 |
| 20000 | 0,0244 |

## Dodanie elementu za losowym indeksem w liście

|  |  |
| --- | --- |
| **rozmiar** | **czas[ms]** |
| 1000 | 0,6188 |
| 2000 | 0,6086 |
| 5000 | 0,6739 |
| 10000 | 0,5786 |
| 20000 | 0,7563 |

## Wyszukiwanie elementu w liście

|  |  |
| --- | --- |
| **rozmiar** | **czas[ms]** |
| 1000 | 0,5980 |
| 2000 | 0,6268 |
| 5000 | 0,6292 |
| 10000 | 0,6439 |
| 20000 | 0,8493 |

# Operacje na kopcu

## Usuwanie elementów z kopca

|  |  |
| --- | --- |
| **rozmiar** | **czas[ms]** |
| 1000 | 0,0347 |
| 2000 | 0,0419 |
| 5000 | 0,0964 |
| 10000 | 0,1794 |
| 20000 | 0,3548 |

## Dodawanie elementów do kopca

|  |  |
| --- | --- |
| **rozmiar** | **czas[ms]** |
| 1000 | 0,0408 |
| 2000 | 0,0600 |
| 5000 | 0,0934 |
| 10000 | 0,1639 |
| 20000 | 0,2428 |

## Wyszukiwanie elementów w kopcu

|  |  |
| --- | --- |
| **rozmiar** | **czas[ms]** |
| 1000 | 0,0230 |
| 2000 | 0,0241 |
| 5000 | 0,0262 |
| 10000 | 0,0295 |
| 20000 | 0,0359 |

# operacje na drzewie RBT

## usuwanie elementów z drzewa

|  |  |
| --- | --- |
| **rozmiar** | **czas [ms]** |
| 1000 | 0,0285 |
| 2000 | 0,0279 |
| 5000 | 0,0279 |
| 10000 | 0,0331 |
| 20000 | 0,0448 |
| 50000 | 0,0364 |
| 100000 | 0,0481 |
| 200000 | 0,0584 |

## dodawanie elementów do drzewa

|  |  |
| --- | --- |
| **rozmiar** | **czas[ms]** |
| 1000 | 0,0293 |
| 2000 | 0,0273 |
| 5000 | 0,0263 |
| 10000 | 0,0268 |
| 20000 | 0,0277 |
| 50000 | 0,0271 |
| 100000 | 0,0316 |
| 200000 | 0,0337 |

## wyszukiwanie elementu w drzewie

|  |  |
| --- | --- |
| **rozmiar** | **czas[ms]** |
| 1000 | 0,0219 |
| 2000 | 0,0203 |
| 5000 | 0,0216 |
| 10000 | 0,0204 |
| 20000 | 0,0227 |

Wnioski

Maksymalny zakres wartości losowanego elementu nieznacznie wydłużył czas operacji, ale nie miał wpływu na samo działanie algorytmu. W przypadku kopca i drzewa zależnie od wartości dodanego elementu, może być konieczne przywrócenie porządku struktury. Dynamiczna alokacja pamięci w przypadku struktury tablicowej jest istotna w prawidłowym działaniu programu, jednak nieprawidłowe jej zaimplementowanie skutkuje niestabilnością programu, spowodowana najczęściej wyciekiem pamięci.

* Operacje na tablicy:

Wszystkie operacje na tablicy są wprost proporcjonalne do jej rozmiaru. Główny wpływ na ujednolicenie wyników ma wymuszona relokacja po każdej operacji. Gdyby pominąć relokację, czasy usuwania i dodawania z końca tablicy wykonywały by się ze stałym czasem. Natomiast czasy dodawani i usuwania z wnętrza tablicy zleżałyby od indeksu w którym należy wykonać operację – im dalej, tym dłużej. Ze względu na przyjęty sposób odmierzania czasu oraz relokacji tablicy, wyniki pomiarów posiadają znaczny błąd pomiarowy.

* Operacje na liście:

Dzięki wskaźnikom na głowę i ogon listy operacje na końcu i początku listy (dodawanie i usuwanie z końca lub początku) mają stałe czasy. Czas usunięcia lub wstawienia elementu do wnętrza listy uzależniony jest od położenia modyfikowanego elementu na liście. Podobnie jak w tablicy, średnie czasy wykonywania reszty operacji rosną wraz z rozmiarem listy.

* Operacje na kopcu i drzewie

Czas trwania algorytmów używanych w kopcu również wzrastają proporcjonalnie do jego rozmiaru oraz wszystkie operacje na drzewach czerwono – czarnych. Jednak wykonując operację na drzewie znacznie zwiększono zakres badanych struktur, ponieważ algorytmy działały zbyt szybko i nie można było zaobserwować dokładnych wyników.

Badane w projekcie struktury mają różne zastosowania. Tablice umożliwiają bezpośredni dostęp do indeksu, jednak przetwarzanie danych trwa znacznie dłużej niż w przypadku listy, która dzięki strukturze wskaźników świetnie sprawdza się w kolejkach i stosach. Kopiec jest dobrą strukturą do sortowania danych i szybkiego ich wyszukiwania. Natomiast największą zaletą drzew czerwono - czarnych jest zrównoważona oraz uporządkowana struktura, pozwalająca na szybką modyfikację elementów.

Bibliogirafia

1. *Algorytmy, struktury danych*, autor: mgr Jerzy Wałaszek, www: http://eduinf.waw.pl/inf/alg/001\_search
2. Drzewo czerwono-czarne, Wikipedia, www: https://pl.wikipedia.org/wiki/Drzewo\_czerwono-czarne
3. Kopiec (binarny) autor: Michał Karpiński, www: http://informatyka.wroc.pl/node/433