Drzewo binarne i czerwono-czarne

Czwartek 12:15-15:00

Mgr inż. Andrzej Wytyczak-Partyka

Szymon Płaneta

Spis treści

1. Zadanie

Wstęp

Zadaniem było utworzenie drzewa binarnego (które samo się balansuje) oraz drzewa czerwono-czarnego. Następnie dokonanie ich porównania poprzez przetestowanie czasów wykonywania operacji dodawania elementów oraz znajdowania elementu o zadanej wartości.

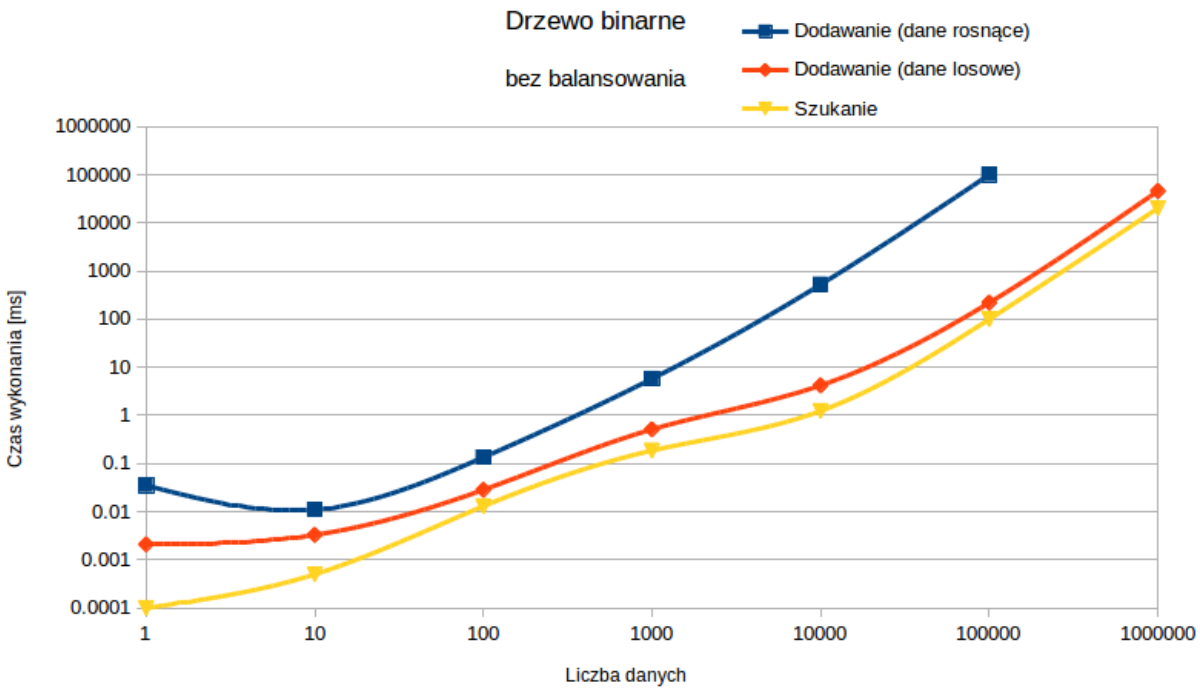
2. Drzewo binarne

Implementacja

W podstawowej wersji drzew binarnego zaimplementowałem jedynie niezbędne pola oraz metody - każdy węzeł zawiera jedynie przechowywaną wartość oraz wskaźniki na swoje lewe i prawe dziecko (które w przypadku braku dzieci wskazują NULL). Taka forma implementacji grozi występowaniem przypadków pesymistycznych - wtedy gdy dodawane wartości są stale rosnące lub stale malejące, drzewo zostaje zdegradowane do listy - dodanie kolejnego elementu lub znalezienie elementu o wartości maksymalnej (lub minimalnej, zależnie od przypadku) wymaga wtedy przejścia przez wszystkie węzły drzewa - koszt takich operacji wynosi więc $O(n)$.

Przeprowadzone testy

Przeprowadzono testy operacji wstawiania oraz wyszukiwania wartości. Wyniki przedstawiono poniżej:



Rys. 1: Czasy operacji wstawiania i wyszukiwania n elementów w drzewie binarnym

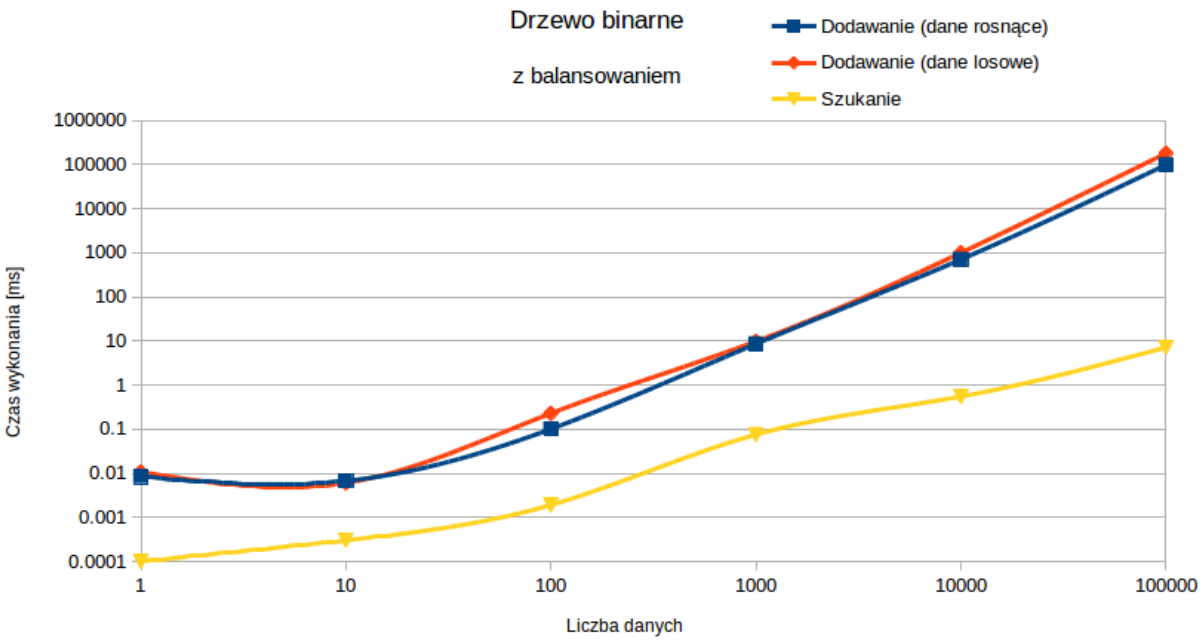
3. Drzewo binarne z balansowaniem

Implementacja

Drzewo binarne przedstawione w poprzednim rozdziale rozbudowałem o pole przechowujące informację o ilości elementów znajdujących się w drzewie, oraz metody obliczania wysokości drzewa, oraz jego balansowania (do balansowania należało zaimplementować metody rotacji w lewo oraz w prawo). Wybrany algorytm balansowania to algorytm DSW - dokonuje on rotacji drzewa w prawo aż do momentu uzyskania listy, a następnie rotacjami w lewo buduje on zrównoważone drzewo. Algorytm balansowania ma złożoność czasową $O(n)$ Taka implementacja pozwala na zachowywanie rozsądnej wysokości drzewa, jednak należy zastanowić się nad tym, kiedy należy dokonywać balansowania. Jeżeli balansowanie drzewa będzie dokonywane po wstawieniu każdego elementu, czasy dodawania ogromnie wzrosną - nonsensem jest zamieniać całe drzewo (zawierające np. 10^6 elementów) na listę, a następnie je odbudowywać przy dodawaniu każdego nowego elementu. Problem ten rozwiązałem dodając warunek:
if($height(root) > 2 * \log_2 iloscwezlow$)
balance(tree)
Pozwala on na uniknięcie używania algorytmu balansowania przy każdej operacji dodawania, jednak nadal ma on swoje minusy - algorytm obliczania wysokości drzewa działa rekurencyjnie i również wpłynie na czasy wykonywania obliczeń. Można rozwiązać ten problem, dodając kolejny warunek - np. sprawdzanie w/w warunku jedynie dla określonych ilości węzłów znajdujacych się aktualnie w drzewie.

Przeprowadzone testy

Przeprowadzono testy operacji wstawiania oraz wyszukiwania wartości. Wyniki przedstawiono poniżej:



Rys. 2: Czasy operacji wstawiania i wyszukiwania n elementów w drzewie binarnym z balansowaniem

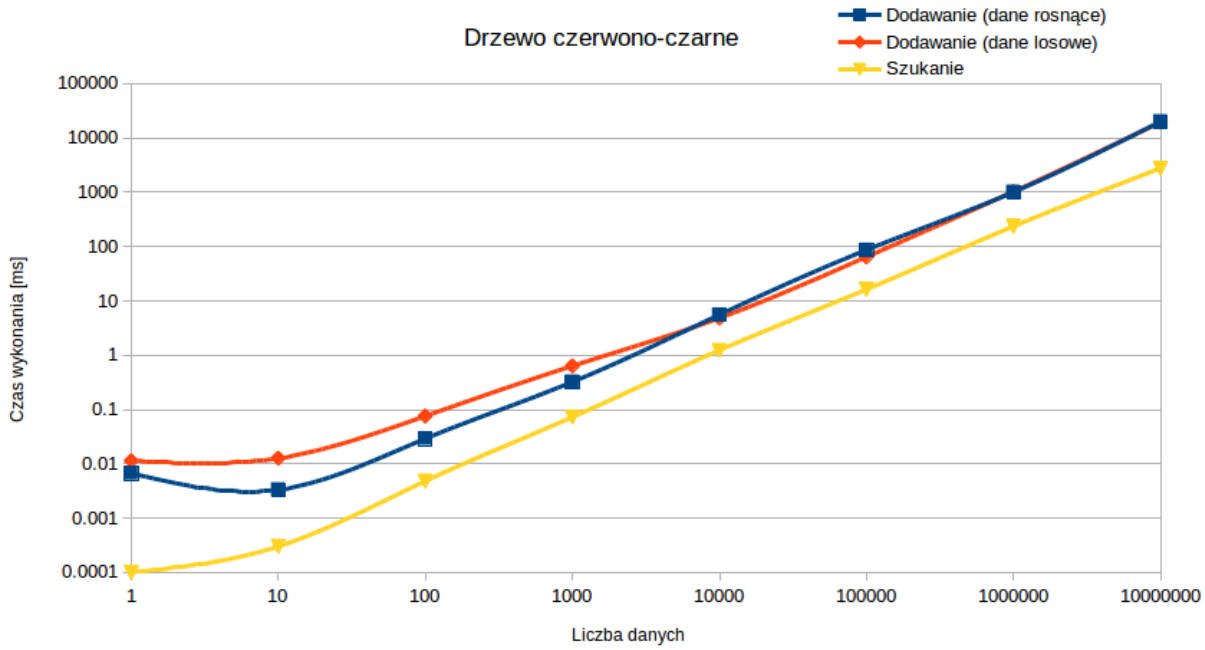
4. Drzewo czerwono-czarne

Implementacja

Drzewo czerwono-czarne to bardziej rozbudowana wersja drzewa binarnego, pozwalająca na efektywne utrzymywanie wysokości drzewa nie większej niż $2\log n + 1$, gdzie n to liczba węzłów w drzewie. Pozwala to na wykonywanie podstawowych operacji w czasie $O(\log n)$.
Każdy węzeł w drzewie czerwono-czarnym posiada dodatkową informację o swoim kolorze (czerwony lub czarny) oraz wskaźnik na swojego rodzica. W skład struktury drzewa wchodzi również dodatkowy węzeł strażnik (sentinel, kolor czarny). Wskazują na niego wszystkie wskaźniki, które w drzewie binarnym miałyby wartość NULL. Implementacja operacji dodawania wymaga sprawdzania warunków drzewa czerwono-czarnego przy każdym dodawaniu i w przypadku konieczności dokonywanie rotacji oraz poprawianie kolorów węzłów, jednak operacje te działaja w czasie $O(n)$.

Przeprowadzone testy

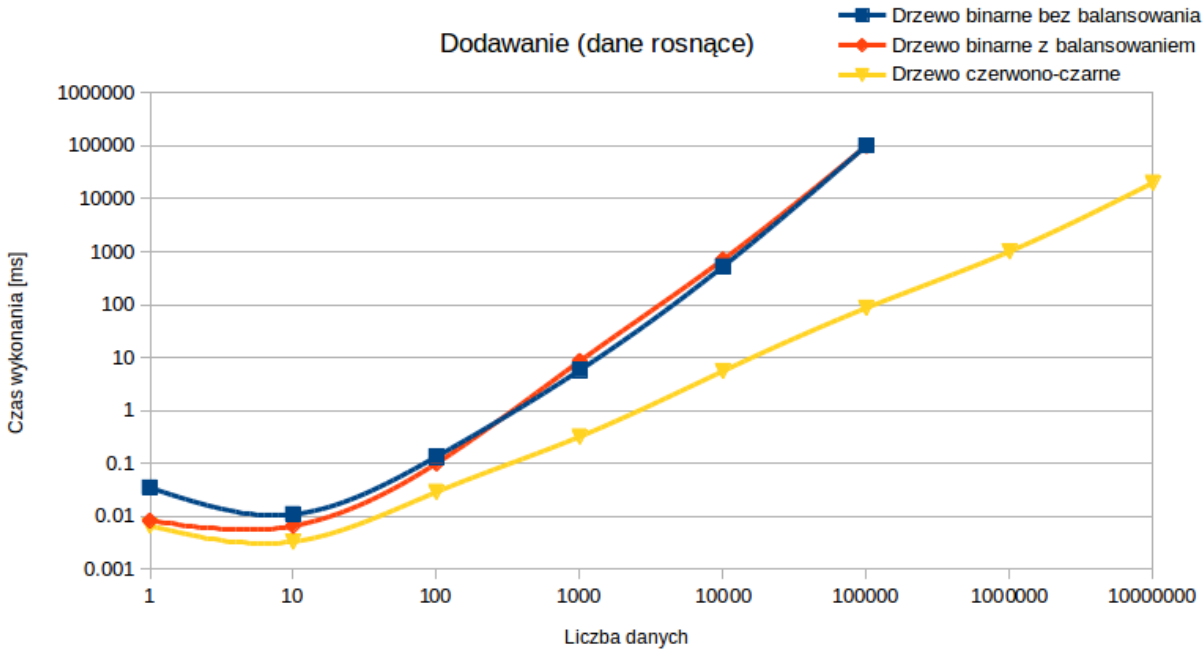
Przeprowadzono testy operacji wstawiania oraz wyszukiwania wartości. Wyniki przedstawiono poniżej:



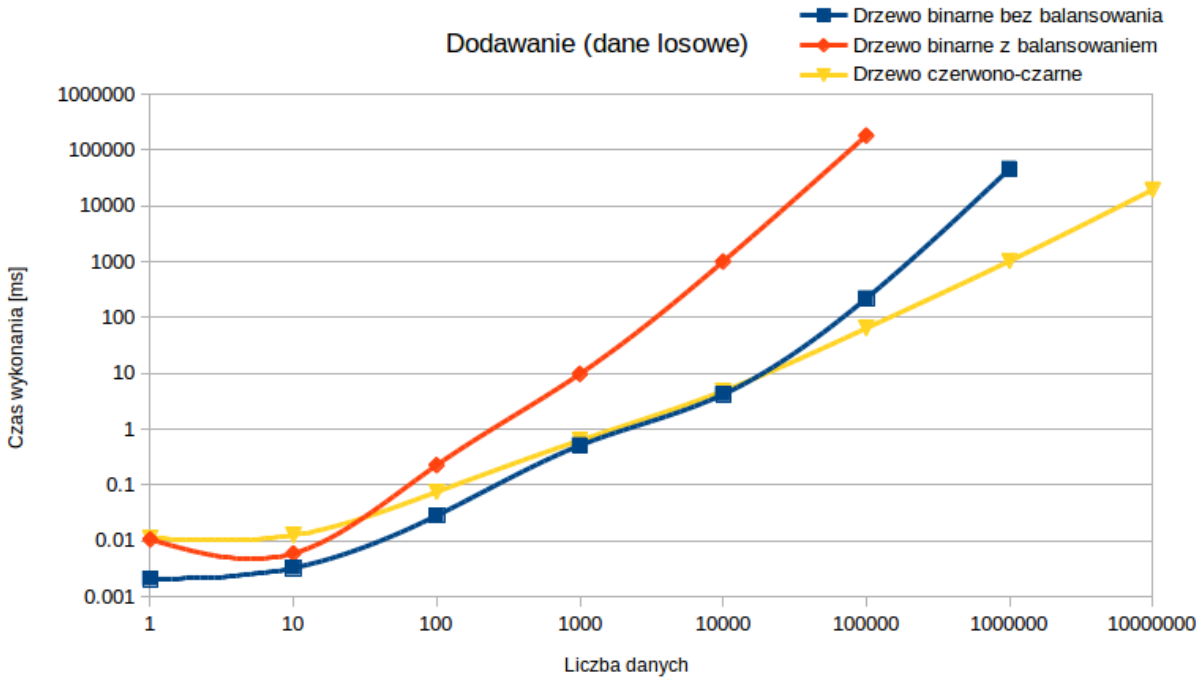
Rys. 3: Czasy operacji wstawiania i wyszukiwania n elementów w drzewie czerwono-czarnym

5. Porównanie

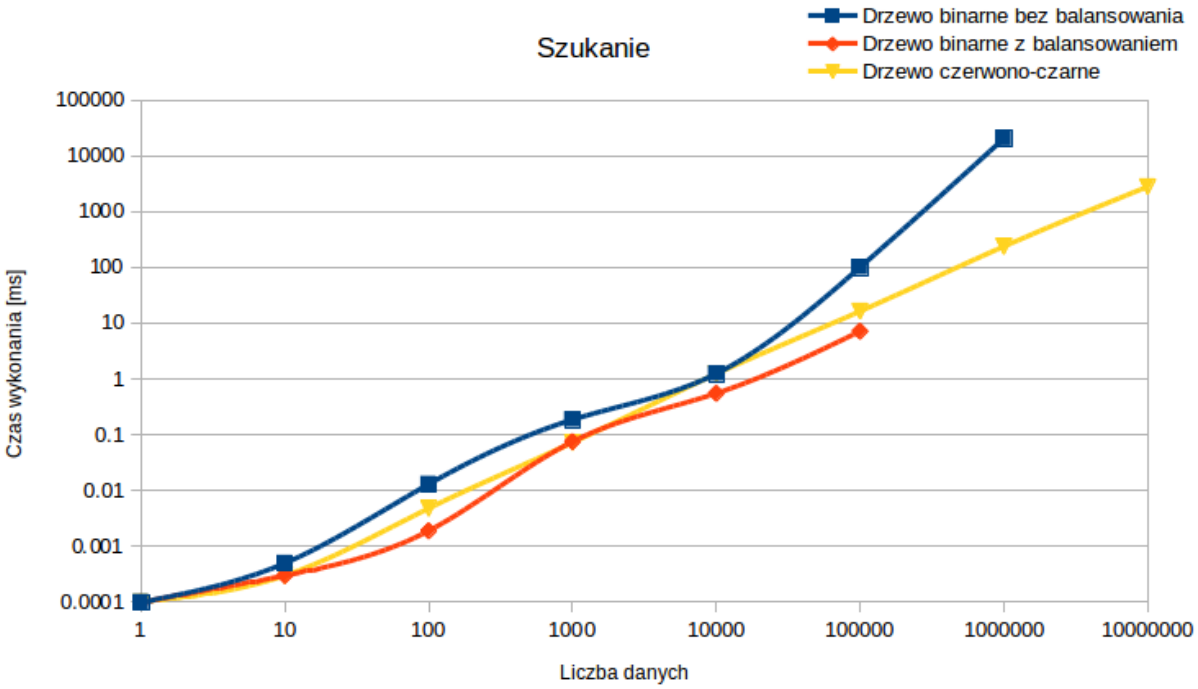
Poniżej przedstawiono czasy działania poszczególnych algorytmów na wspólnych wykresach.



Rys. 4: Czasy operacji wstawiania danych rosnących dla wszystkich drzew



Rys. 5: Czasy operacji wstawiania danych losowych dla wszystkich drzew



Rys. 6: Czasy operacji wyszukiwania elementów dla wszystkich drzew

6. Wnioski

Drzewo czerwono-czarne działa zdecydowanie najszybciej, oraz nie posiada przypadków pesymistycznych. W większości zastosowań będzie to zdecydowanie lepszy wybór, niż zwykłe binarne drzewo poszukiwań. Jeżeli jesteśmy pewni, że w konkretnym zastosowaniu nie przytrafią nam się przypadki pesymistyczne, być może wystarczy zastosować zwykłe drzewo binarne bez balansowania. Pod względem operacji dodawania najgorzej wypadło drzewo binarne z balansowaniem - działa znacznie wolniej od drzewa czerwono-czarnego, a od drzewa binarnego bez balansowania wypada lepiej jedynie w przypadkach pesymistycznych dla dużej ilości danych. Wykorzystanie drzewa binarnego z balansowaniem może być uzasdanione w przypadku, gdy wiemy, że znaczną większością wykonywanych operacji będą operacje wyszukiwania, a dodawanie będzie odbywać się naprawdę bardzo rzadko. Można zastanowić się nad poprawą implementacji drzewa binarnego z balansowaniem - być może w przypadku wykorzystania bardziej optymalnych algorytmów, czasy operacji wstawiania danych ulegną poprawie.