Opracowanie struktur danych

Szymon Kaczmarek, nr indeksu 148056

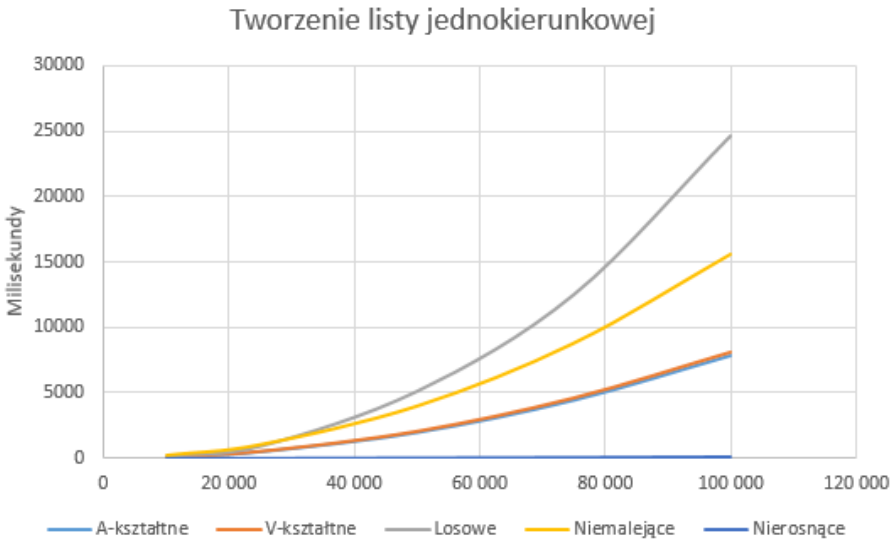
1 Wprowadzenie

Niniejsze opracowanie ma na celu przedstawienie efektywności struktur danych. Zaprezentowana będzie lista jednokierunkowa posortowana, drzewo BST oraz drzewo AVL. Struktury te zostały zaimplementowane w języku C++ (link do repozytorium GitHub).

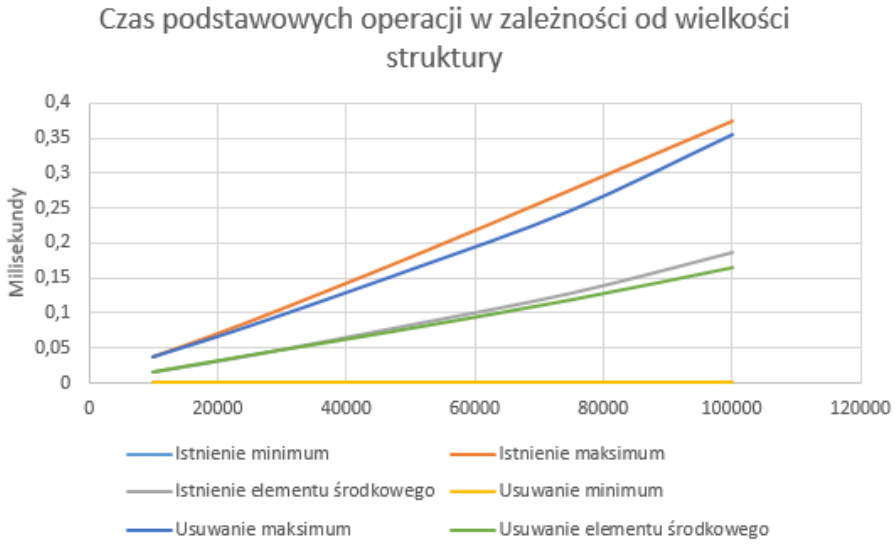
2 Lista jednokierunkowa

Lista jednokierunkowa to struktura polegająca na tym, że każdy jej element przechowuje swoją wartość oraz wskaźnik na następny element listy. Używając tej struktury przechowujemy jedynie wskaźnik na pierwszy element (tzw. głowę). Chcąc dostać się do n-tego elementu należy przeiterować się przez strukturę n-razy. Moja implementacja zawiera listę posortowaną, która ułatwia wyszukiwanie elementów kosztem czasu tworzenia. Cechy tej struktury danych to:

- Wysoka średnia złożoność dodawania elementu ($O(n)$)
- Łatwa iteracja po elementach
- Łatwe usuwanie elementu
- Wysoka średnia złożoność znajdowania elementu ($O(n)$)



Jak można zauważyć, pesymistyczą złożonością dla tworzenia listy jest $O(n^2)$. Związane jest to faktem, że aby dodać element należy wyszukać ze złożonością liniową odpowiedniego miejsca dla niego co dla n elementów daje powyższą złożoność. Optymistyczną złożonością jest natomiast $O(n)$ dla danych podanych w kolejności nierosnącej. W tym przypadku kolejne elementy zawsze są dodawane na początku listy co powoduje, że wyszukiwanie odpowiedniego miejsca jest wykonywane ze złożonością $O(1)$.



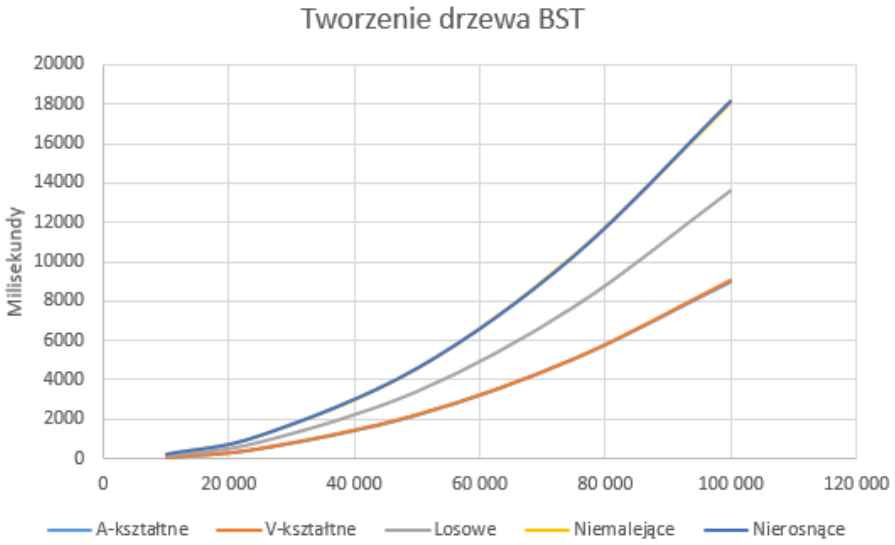
Z racji, że lista jest posortowana czasy tych operacji są w złożoności liniowej

($O(n)$). Jedynie operacje związane z minimum są wykonywane ze złożonością $O(1)$, ponieważ jest to pierwszy element listy i nie trzeba go wyszukiwać.

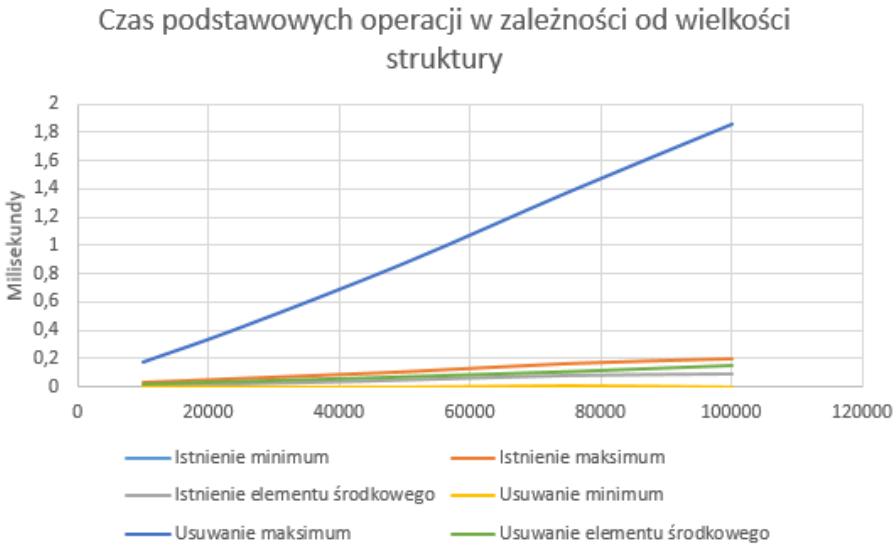
3 Drzewo BST

Drzewo BST jest usprawnieniem listy jednokierunkowej. Każdy element, zamiast przechowywać wskaźnik na kolejny, przechowuje wskaźniki na dwójkę dzieci, z których jedno posiada wartość większą od niego, a drugie wartość mniejszą. Takie usprawnienie polepsza nam złożoność wyszukiwania danych elementów, ponieważ zamiast iterować się przez całą stukturę, z każdym kolejnym elementem dzielimy sobie ją na pół. Cechy tej struktury danych to:

- Niska średnia złożoność dodawania elementu ($O(log(n))$)
- Trudna iteracja po elementach
- Trudne usuwanie elementu (przy usuwaniu elementu trzeba wybrać kolejny element, który nie zaburzy struktury drzewa)
- Niska średnia złożoność znajdowania elementu ($O(log(n))$)



Jak widać, pesymistyczna złożoność wynosi $O(n^2)$. Dla danych posortowanych oraz malejących drzewo to przyjmuje postać zwykłej listy posortowanej, stąd wysoka złożoność. Dla pozostałych danych złożoność obliczeniowa wynosi $O(n * log(n))$.

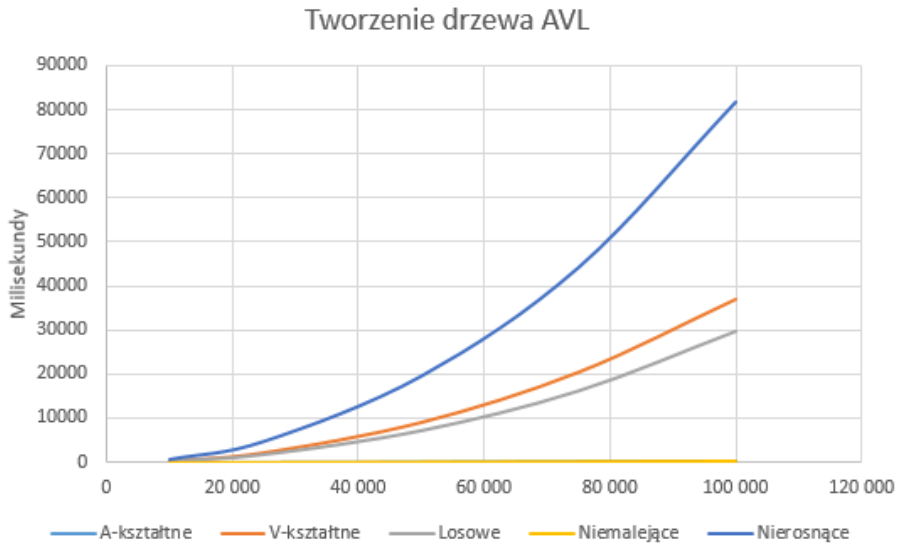


Jak widać pesymistyczna złożoność operacji usuwania elementu wynosi $O(n^2)$. Spowodowane jest to najgorszą możliwym ustawieniem drzewa. Usuwanie elementu bliskiego korzenia w tej sytuacji powoduje przepisywanie kolejnych elementów. Znalezienie elementu wtedy jest liniowe, tak samo jak przepisywanie co daje nam powyższą złożoność. Średnia złożoność jest lepsza dla tych operacji ponieważ wynosi $O(n * log(n))$.

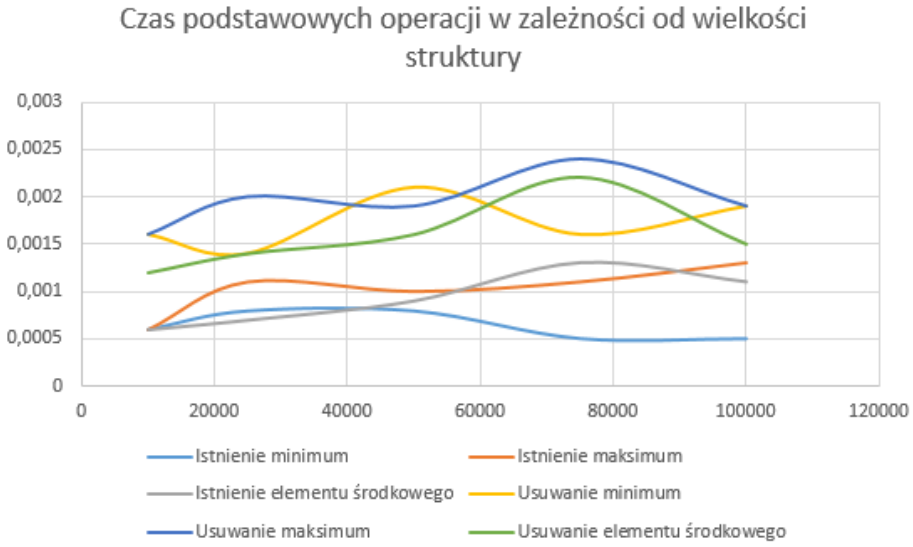
4 Drzewo AVL

Drzewo AVL jest kolejnym usprawnieniem wcześniejszych struktur. Budową przypomina drzewo BST, lecz tutaj gałęzie się automatycznie balansują, celem uniknięcia pesymistycznych przypadków gdzie wszystkie dane znajdują się w jednej linii. Cechy tego algorytmu to:

- Niska średnia złożoność dodawania elementu ($O(log(n))$)
- Trudna iteracja po elementach
- Trudne usuwanie elementu (przy usuwaniu elementu trzeba dokonać rotacji gałęzi, tak aby nie zaburzyć struktury drzewa)
- **Zapewniona** złożoność znajdowania elementu ($O(log(n))$)

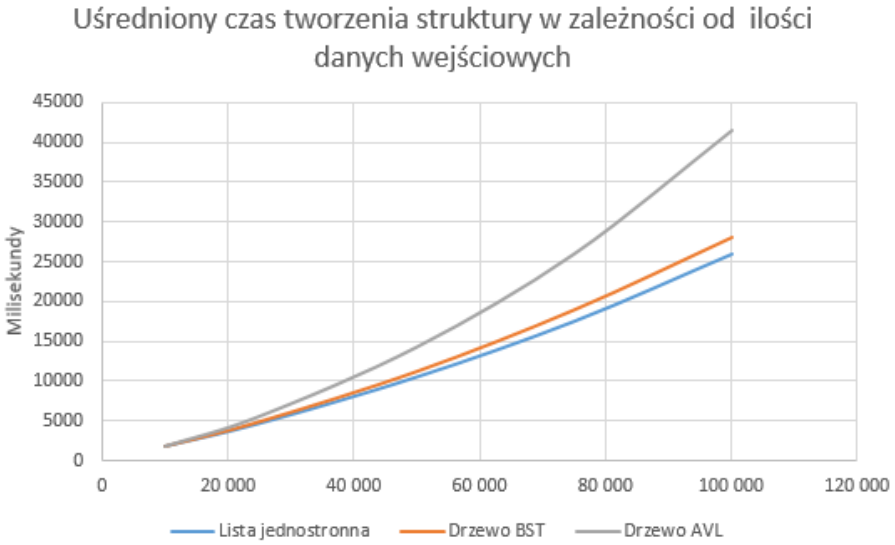


Różnica w czasach tworzenia drzewa AVL polega na ciągłym równoważeniu dodawanych danych. Można przyjąć, że złożoność tworzenia tej struktury jest równa $O(n * log(n))$.

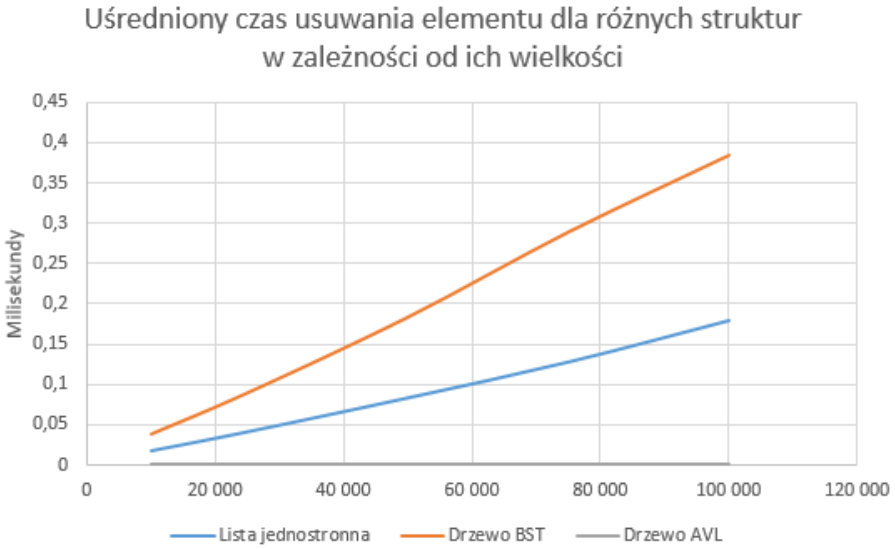


Podstawowe operacje mają czasy bardzo niskie i zbliżone do siebie, a fluktuacje są spowodowane równoważeniem się drzewa oraz niedokładnością pomiarową. Wszystkie te operacje mają zapewnioną złożoność $O(log(n))$, która wynika z faktu zrównoważenia drzewa.

5 Zestawienie



Struktury listy oraz drzewa BST tworzą się w podobnym czasie, ponieważ są one pozbawione dodatkowej funkcji równoważącej. Drzewo AVL posiada takową przez co czas tworzenia takiej struktury jest większy. Drzewo AVL, kosztem czasu tworzenia struktury, zapewnia optymalną złożoność dla operacji na nim. Jest to mała cena w porównaniu do zysków czasowych podczas pracy na tej strukturze.



Jak widać czas usuwania elementu jest o wiele niższy w drzewie AVL od reszty struktur. Dzięki stałemu równoważeniu jest stale zapewniona najlepsza złożoność $O(log(n))$.

6 Zakończenie

Zostały przedstawione i zestawione trzy struktury danych. Każda z nich została omówiona, tak samo jak różnice między nimi. Rozpatrzone zostały również ich cechy.

Spis treści

1 Wprowadzenie **1**

2 Lista jednokierunkowa **1**

3 Drzewo BST **3**

4 Drzewo AVL **4**

5 Zestawienie **6**

6 Zakończenie **7**