Gracjan Serwinowski

Nr. 264166

W ramach niniejszego sprawozdania przeprowadziłem badania nad implementowanymi algorytmami operującymi na różnych strukturach danych. Celem było zbadanie złożoności tych algorytmów oraz analiza ich wydajności na podstawie wykonanych eksperymentów. W programie skupiliśmy się na operacjach wstawiania, usuwania, wyszukiwania oraz wyświetlania elementów w różnych strukturach danych.

Plan eksperymentu:

1. Struktury danych:
   * W moim programie zaimplementowałem trzy podstawowe struktury danych: tablice, listy jednokierunkowe oraz drzewa binarne.
   * Przeanalizowałem złożoność czasową dla operacji wstawiania, usuwania, wyszukiwania i wyświetlania elementów w każdej z tych struktur.
2. Algorytmy:
   * W przypadku operacji wstawiania i usuwania elementu, zaimplementowałem algorytmy takie jak wstawianie na początku listy, wstawianie na końcu listy, wstawianie na określonej pozycji, usuwanie podanej wartości oraz algorytmy odpowiednie dla tablicy i drzewa binarnego.
   * Dla operacji wyszukiwania elementu, zaimplementowałem algorytmy takie jak wyszukiwanie liniowe.
3. Parametry eksperymentu:
   * Wielkość struktur danych: Rozpocząłem od małych struktur, a następnie zwiększałem ich rozmiar, aby zbadać wpływ na wydajność algorytmów. Zastosowana ilość danych w strukturach to: 100, 1000 i 10000 losowych, nieposortowanych elementów.
   * Generowanie danych: Wygenerowaliśmy dane w sposób losowy, za pomocą oddzielnego programu napisanego w języku C++, który losuje taką ilość danych jaka została określona przez pierwszą liczbę.
   * Pomiar czasu: Skorzystaliśmy z bibliotek programistycznych, takich jak chrono do pomiaru czasu wykonania algorytmów i zaimplementowaliśmy tą funkcję w menu „switch”, aby zliczała czasy wykonywanych operacji.

W dalszej części sprawozdania przedstawię szczegółowe implementacje algorytmów oraz wyniki pomiarów, które pozwolą nam na dokładniejszą analizę złożoności tych algorytmów i ich wydajności w zależności od sprzętu, wielkości struktur i

|  |  |  |  |
| --- | --- | --- | --- |
|  | Tablica | Lista | DrzewoBST |
| Usuwanie | 0.0085 ms | 0.0048 ms | 0.0036 ms |
| Dodawanie (na koniec) | 0.0045 ms | 0.0094 ms |  |
| Dodawanie (na początek) | 0.0055 ms | 0.0020 ms | 0.0037 ms |
| Dodawanie (losowe) | 0.0058 ms |  |  |
| Dodawanie (po indeksie) |  | 0.0078 ms |  |
| Szukanie | 0.0010 ms | 0.0072 ms | 0.0024 ms |

Tab. 1 Średni czas z 10 prób dla struktury 100 elementowej.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Tablica | Lista | DrzewoBST |
| Usuwanie | 0.0162 ms | 0.0023 ms | 0.0049 ms |
| Dodawanie (na koniec) | 0.0101 ms | 0.0600 ms |  |
| Dodawanie (na początek) | 0.0101 ms | 0.0020 ms | 0.0036 ms |
| Dodawanie (losowe) | 0.0140 ms |  |  |
| Dodawanie (po indeksie) |  | 0.0610 ms |  |
| Szukanie | 0.0026 ms | 0.0009 ms | 0.0022 ms |

Tab. 2 Średni czas z 10 prób dla struktury 1000 elementowej.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Tablica | Lista | DrzewoBST |
| Usuwanie | 0.0831 ms | 0.4413 ms | 0.0048 ms |
| Dodawanie (na koniec) | 0.0529 ms | 0.7592 ms |  |
| Dodawanie (na początek) | 0.0601 ms | 0.0027 ms | 0.0062 ms |
| Dodawanie (losowe) | 0.0587 ms |  |  |
| Dodawanie (po indeksie) |  | 0.5651 ms |  |
| Szukanie | 0.0365 ms | 0.4886 ms | 0.0031 ms |

Tab. 4 Średni czas z 10 prób dla struktury 10000 elementowej.

1. Wyszukiwanie w liście:
   * Złożoność: O(n)
   * Algorytm przegląda listę sekwencyjnie od początku do końca, porównując wartość szukanego elementu z każdym elementem listy.
   * W najgorszym przypadku, gdy szukany element znajduje się na końcu lub nie ma go w liście, konieczne jest przejście przez wszystkie elementy.
   * Wyszukiwanie w liście ma liniową złożoność czasową, co oznacza, że czas potrzebny do znalezienia elementu rośnie proporcjonalnie do liczby elementów w liście.
2. Wyszukiwanie w tablicy:
   * Złożoność: O(n)
   * Algorytm przegląda tablicę sekwencyjnie od początku do końca, porównując wartość szukanego elementu z każdym elementem tablicy.
   * Podobnie jak w przypadku wyszukiwania w liście, czas potrzebny do znalezienia elementu rośnie proporcjonalnie do liczby elementów w tablicy.
   * Wyszukiwanie w tablicy również ma złożoność liniową, co oznacza, że może być mniej efektywne dla dużych tablic.
3. Wyszukiwanie w drzewie BST:
   * Złożoność: O(log n) w przypadku zrównoważonego drzewa BST
   * Wyszukiwanie w drzewie BST opiera się na porównywaniu wartości szukanego elementu z wartościami węzłów drzewa.
   * Dzięki strukturze drzewa BST, gdzie lewe poddrzewo zawiera wartości mniejsze, a prawe poddrzewo zawiera wartości większe, algorytm może skutecznie porównywać i przechodzić do odpowiedniego poddrzewa.
   * W przypadku zrównoważonego drzewa BST, każde porównanie pozwala wykluczyć połowę pozostałych węzłów, co prowadzi do efektywnego podziału drzewa.
   * W rezultacie, czas potrzebny do znalezienia elementu w drzewie BST rośnie logarytmicznie w stosunku do liczby węzłów drzewa.
   * Jednakże, w najgorszym przypadku, gdy drzewo BST jest niezrównoważone (np. degeneruje się do listy), złożoność czasowa wyszukiwania może wynosić O(n), co jest równoważne sekwencyjnemu przeszukiwaniu.

Wnioski:

* Wyszukiwanie w drzewie BST jest zazwyczaj bardziej efektywne niż sekwencyjne przeszukiwanie w liście lub tablicy, szczególnie dla dużych zbiorów danych.
* Wyszukiwanie w tablicy i liście ma złożoność liniową, co oznacza, że czas potrzebny do znalezienia elementu rośnie proporcjonalnie do liczby elementów.
* Wybór odpowiedniego algorytmu zależy od kontekstu i charakterystyki danych, takich jak rozmiar zbioru, częstotliwość operacji wyszukiwania, dodawania i usuwania elementów, a także oczekiwana wydajność.

Wnioski te dotyczące efektywności implementowanych algorytmów wskazują na istotne różnice w czasie wykonywania operacji wyszukiwania elementów. Wyszukiwanie liniowe w liście i tablicy ma złożoność czasową O(n), co oznacza, że czas potrzebny do znalezienia elementu rośnie proporcjonalnie do liczby elementów. Z kolei wyszukiwanie w drzewie BST oferuje złożoność czasową O(log n) w przypadku zrównoważonego drzewa. Oznacza to, że czas potrzebny do znalezienia elementu rośnie logarytmicznie w stosunku do liczby węzłów drzewa. W praktyce, wybór odpowiedniego algorytmu zależy od konkretnych wymagań i charakterystyki danych. Wyszukiwanie w drzewie BST sprawdza się szczególnie dobrze dla dużych zbiorów danych, gdzie czas wykonywania operacji jest istotny. Natomiast wyszukiwanie liniowe w liście lub tablicy może być wystarczające dla mniejszych zbiorów lub sytuacji, gdzie kolejność elementów nie jest istotna. Ważne jest również uwzględnienie innych operacji, takich jak wstawianie, usuwanie i wyświetlanie elementów, które również mają wpływ na efektywność algorytmów. Na podstawie analizy złożoności tych operacji można dokonać bardziej kompleksowej oceny i wyboru odpowiedniego algorytmu w zależności od konkretnych potrzeb. Wnioski te mogą posłużyć jako podstawa do dalszych eksploracji i badań nad efektywnością różnych algorytmów w zależności od kontekstu i wymagań aplikacji. Optymalny wybór algorytmu może przyczynić się do poprawy wydajności, oszczędności zasobów i lepszej obsługi danych w programach i systemach informatycznych.