Raport z laboratorium nr: 6

Imię i nazwisko autora: Grzegorz Piśkorski

Nr indeksu: 405879

1. **Najważniejszy fragment kodu z laboratoriów:**

```
def insert_new_node(self, elem):
    found = self.search(elem)
    if found:
        return None
    else:
        n = len(self.list)
        pot_root = round(elem)
        if elem - pot_root >= 0:
            pot_root += 0.5
        else:
            pot_root -= 0.5

        pot_index = binary_search(self.list, 0, n - 1, pot_root, True)
        if pot_index >= n:
            self.list.append(Node(pot_root))
            self.list[pot_index].insert_data(Node(elem))
        else:
            if pot_root == self.list[pot_index].val:
                self.list[pot_index].insert_data(Node(elem))
            else:
                self.list.insert(pot_index, Node(pot_root))
                self.list[pot_index].insert_data(Node(elem))
```

Uzasadnienie wyboru:

Wybrałem operację dodającą do struktury nowy element. Moim zdaniem jest to najważniejsza część mojego kodu, z tego względu, że ona nam tworzy naszą strukturę. Bez niej musielibyśmy każdemu elementowi (który chcemy dodać) własnoręcznie znaleźć odpowiednie miejsce w strukturze i odpowiednio poprzepinać wszystkie Nody, aby zachowana była zaplanowana struktura.

Podsumowanie

Zadanie polegało na skonstruowaniu zadanej struktury i oszacowaniu jej złożoności obliczeniowej. Pomiary czasów dla poszczególnych operacji za każdym razem dla każdej operacji okazywały się bardzo małe (w moim przypadku było to 0.0 dla każdej operacji nawet dla 1 000 000 wylosowanych liczb).

Założenia: h = wysokość największego drzewa w strukturze

n = liczba korzeni w tablicy

Insert(x):

Jak wiemy, dodanie nowego elementu do drzewa BST jest proporcjonalna do jego wysokości, zatem O(h). My jednak musimy najpierw znaleźć odpowiednie miejsce w strukturze i w najgorszym przypadku powiększy tablicę naszych korzeni. O ile dodanie nowego elementu na koniec list przy pomocy append zajmuje czas jednostkowy. O tyle dodanie elementu gdzieś w środku listy przy pomocy list.insert(), według dokumentacji pythona zajmuje O(n). A odpowiednie miejsce znajduję przy pomocy binary searcha. Reasumując, złożoność naszej operacji Insert(x) wynosi:

O(logn + n + h) – znajdujemy miejsce na nowy węzeł + powiększamy naszą tablicę korzeni + wstawiamy element do znalezionego korzenia drzewa BST

Minimum(x):

x – skoro na starcie mamy dany korzeń i nie musimy się martwić znalezieniem korzenia, w którym musimy znaleźć minimum to złożoność będzie taka sama jak wyszukiwanie minimum w zwykłym drzewie BST to jest: O(h) w najgorszym przypadku.

Maximum(x):

Dokładnie ta sama sytuacji co przy operacji Minimum(x), zatem złożoność wynosi O(h).

Search(x):

W tym wypadku najpierw musimy znaleźć odpowiedni korzeń, w którym może znajdować się nasz szukany element (ponownie korzystam z binary searcha), czyli mamy O(logn). A następnie zgodnie z teorią szukamy elementu w znalezionym korzeniu w czasie O(h). Zatem złożoność tej operacji to O(logn + h)