Sprawozdanie z Projektu nr 2

Po ostatnim projekcie spodobała mi się idea programowania obiektowego w takich projektach. Zacząłem więc od ustawienia sobie wszystkich potrzebnych plików i katalogów

- Drzewo BST jest zaprogramowane w katalogu ***/BST*** (nagłówek jak i .cpp)
- Lista dozwiązaniowa jest zaprogramowane w katalogu ***/LinkedList*** (nagłówek jak i .cpp)
- Tablica z hasowaniem jest zaprogramowane w katalogu ***/HashTable*** (nagłówek jak i .cpp)

Na wstępie jeszcze powiem tyle że program kompilowałem własnoręcznie w prostym IDE za pomocą MAKEFILE i konsoli typu bash.

Cały projekt zacząłem od struktury LinkedList – Listy z dowiązaniem. Przypomniałem sobie kształt całej struktury z zajęć. Przekształciłem na obiekt typu klasa, pododawałem metody do wstawienia danych i wypisywania na ekran. Myślę, że ta struktura była najłatwiejsza ze wszystkich.

Drugą w kolei zacząłem robić drzewo BST. Tutaj problemy się zaczęły. Niestety w takiego typu strukturach ważną częścią jest rekurencja, która niestety u mnie jest na poziomie podłogi…

Najwięcej czasu poświęciłem na zrozumienie jak działa jakakolwiek rekurencja a dopiero potem jak wygląda algorytm drzewa w programowaniu. Nad drzewem spędziłem większą część całego projektu.

Na koniec tablica haszująca. Tu nie było nic trudnego. Zwykła tablica o typie Listy z poprzedniego podpunktu. Mając cały algorytm Listy tutaj jedynie musiałem go użyć. Jedynym problemem było wymyślenie w jaki sposób haszować liczby w taki sposób, aby ich index był w przedziale 0 – n i zawsze wskazywał na tą samą liczbę, np. przekazujemy liczbę 980723412 i jej indexem ma być załóżmy liczba 123 która jest indexem tej liczby w tablicy. Więc za każdym razem, gdy wywołujemy tą funkcję dla tej samej liczby ma zwracać tą samą wartość. W moim przypadku rozwiązanie było proste jednak trochę nad tym siedziałem. Funkcja haszująca zwracała resztę z dzielenia liczby przekazywanej z argumentu przez n (ilość elementów w tablicy) Tym sposobem byłem pewny, że index nie będzie ani ujemny ani > n. Nic prostszego.

Aby program miał działać dla różnego n wystarczy w pliku main.cpp zmienić #define n na jakąś inną wartość. Poniżej prezentuję czasy dla odpowiednio n = 10 000, 20 000, 30 000, 100 000.

N = 10 000

```
Wprowadzanie danych do drzewa BST...
Gotowe! Czas: 0.00279333s
Wprowadzanie danych do listy...
Gotowe! Czas: 0.342374s
Wprowadzanie danych do tablicy z hashowaniem...
Gotowe! Czas: 0.000644417s
Wyszukiwanie danych w drzewie BST...
Gotowe! Czas: 0.00251929s
Wyszukiwanie danych w liscie...
Gotowe! Czas: 0.64285s
Wyszukiwanie danych w tablicy z hashowaniem...
Gotowe! Czas: 0.000388083s
```

N = 20 000

```
Wprowadzanie danych do drzewa BST...
Gotowe! Czas: 0.00586517s
Wprowadzanie danych do listy...
Gotowe! Czas: 1.29826s
Wprowadzanie danych do tablicy z hashowaniem...
Gotowe! Czas: 0.0012015s
Wyszukiwanie danych w drzewie BST...
Gotowe! Czas: 0.00548604s
Wyszukiwanie danych w liscie...
Gotowe! Czas: 2.56211s
Wyszukiwanie danych w tablicy z hashowaniem...
Gotowe! Czas: 0.000823375s
```

N = 30 000

```
Wprowadzanie danych do drzewa BST...
Gotowe! Czas: 0.00971967s
Wprowadzanie danych do listy...
Gotowe! Czas: 2.86745s
Wprowadzanie danych do tablicy z hashowaniem...
Gotowe! Czas: 0.00184233s
Wyszukiwanie danych w drzewie BST...
Gotowe! Czas: 0.010263s
Wyszukiwanie danych w liscie...
Gotowe! Czas: 5.7231s
Wyszukiwanie danych w tablicy z hashowaniem...
Gotowe! Czas: 0.00125396s
```

N = 100 000

```
./out
Wprowadzanie danych do drzewa BST...
Gotowe! Czas: 0.0367459s
Wprowadzanie danych do listy...
Gotowe! Czas: 31.947s
Wprowadzanie danych do tablicy z hashowaniem...
Gotowe! Czas: 0.00668788s
Wyszukiwanie danych w drzewie BST...
Gotowe! Czas: 0.0387339s
Wyszukiwanie danych w liscie...
Gotowe! Czas: 64.3099s
Wyszukiwanie danych w tablicy z hashowaniem...
Gotowe! Czas: 0.00537821s
```

Obserwacje:
Najwięcej czasu na wprowadzenie danych jak i wyszukanie zajmuję liście, nic dziwnego. W liście, aby dodać element musimy przejść po kolei po niej. Gdy lista jest pusta dodajemy na początek, Potem musimy przejść do 2 elementu itd. Aby dodać 100-tny element musimy przejść przez 99 elementów, a ta lista za każdym razem się zwiększa. Jednak to był pikuś w porównaniu to wyszukiwania w liście elementów. Aby wyszukać dany element musimy przejść w najgorszym wypadku od początku listy do jej końca.
Drugie miejsce zajmuję drzewo BST, tutaj wprowadzanie danych trwa krócej, gdyż sprawdzamy tylko gałąź, która nam pasuję.
Najszybciej działa tablica z haszowaniem. Oczywista oczywistość, wprowadzając liczbę do tablicy od razu znamy jej położenie, liczymy sobie jej index funkcją, która wykonuje proste obliczenie, po czym od razu w to miejsce ją dodajemy na koniec listy. Jeśli w danym indexie jest już jakaś wartość idziemy niżej, jeśli jest pusto zapisujemy na „wierzchu". Wszystko prosto, łatwo i szybko!

Tak jak w poprzednim projekcie zachęcam do githuba: https://github.com/matelko123/ALGO-Projekt2