**Sprawozdanie**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Obraz zawierający tekst, symbol, godło, logo  Opis wygenerowany automatycznie* | Państwowa Akademia Nauk Stosowanych w Nysie | | | Wydział Nauk Technicznych | |
| Laboratorium Zarządzanie Danych Informacyjnymi | | | | |
| Kierunek: | Informatyka | Rok studiów nr: | 2 | Semestr nr: | 3 |
| Rok akademicki: | 2024/2025 | Grupa administracyjna: | L1 | Grupa  ćwiczeniowa: | - |

SPRAWOZDANIE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nr ćwiczenia | Temat ćwiczenia | | | |
| - | Aplikacja Bazodanowa w języku Java | | | |
| Wpisz termin  złożenia  sprawozdania |
| 11.01.2025 |
| Data faktycznego  złożenia  sprawozdania |
| (nie wypełniaj) |
| Wykonawcy | Nazwisko | Imię | Nr indeksu | Ocena |
| Biernat | Mateusz | 27725 | (Nie wypełniane w trybie online) |
|  |  |  |  |
|  |  |  |  |

**Uwaga**: Umieszczenie danych osobowych wykonawców stanowi grupowe i nieodwołalne oświadczenie, że są oni/one   
(i tylko oni/one) współautorami przedstawionego sprawozdania. Późniejsza zmiana składu zespołu wykonawców nie będzie możliwa.

Nie wypełniać  
 przy składaniu online

Data i podpis prowadzącego ćwiczenia

1. **Część Teoretyczna**

**Wyszukiwanie binarne**

**Wyszukiwanie binarne** jest algorytmem służącym do efektywnego znajdowania elementu w posortowanej tablicy. Działa na zasadzie dzielenia zbioru na pół w każdej iteracji, dzięki czemu szybko zawęża obszar poszukiwań.

**Działanie:**

* Wyszukiwanie binarne zaczyna od porównania szukanego elementu z elementem środkowym w tablicy.
* Jeśli szukany element jest równy elementowi środkowemu, wyszukiwanie kończy się sukcesem.
* Jeśli szukany element jest mniejszy od elementu środkowego, algorytm kontynuuje wyszukiwanie w lewej połowie tablicy.
* Jeśli szukany element jest większy, algorytm kontynuuje wyszukiwanie w prawej połowie tablicy.
* Proces ten powtarza się, aż element zostanie znaleziony lub zakres poszukiwań będzie pusty (co oznacza, że elementu nie ma w tablicy).

**Złożoność:**

* Czasowa: **O(log n)**, ponieważ w każdej iteracji zakres wyszukiwania jest dzielony na pół.
* Pamięciowa: **O(1)**, ponieważ nie wymaga dodatkowej pamięci poza zmiennymi pomocniczymi.

**Wyszukiwanie liniowe**

**Wyszukiwanie liniowe** jest najprostszym algorytmem wyszukiwania, który polega na przeszukiwaniu tablicy po kolei, sprawdzając każdy element z osobna.

**Działanie:**

* Algorytm przechodzi przez wszystkie elementy w tablicy, porównując każdy z nich z szukanym elementem.
* Jeśli znajdzie element, który jest równy szukanemu, algorytm kończy wyszukiwanie i zwraca jego indeks.
* Jeśli algorytm przejdzie całą tablicę, a element nie zostanie znaleziony, zwraca informację o braku elementu.

**Złożoność:**

* Czasowa: **O(n)**, gdzie n to liczba elementów w tablicy, ponieważ w najgorszym przypadku trzeba sprawdzić każdy element.
* Pamięciowa: **O(1)**, ponieważ nie wymaga dodatkowej pamięci oprócz zmiennych pomocniczych.

**Łańcuchowe haszowanie (Chaining)**

**Łańcuchowe haszowanie** to technika rozwiązywania kolizji w tabelach haszujących. Polega na przechowywaniu elementów, które mają ten sam hash w postaci łańcucha (np. listy powiązanej), co pozwala na skuteczne zarządzanie kolizjami.

**Działanie:**

* Dla każdego elementu obliczany jest hash za pomocą funkcji haszującej, który wskazuje na indeks w tablicy.
* Jeśli dwa lub więcej elementów mają ten sam hash (tzn. kolidują), są przechowywane w tym samym miejscu w tabeli, ale w formie listy (łańcucha).
* Kiedy dochodzi do dodania nowego elementu, jest on dodawany na koniec listy powiązanej w tym samym miejscu tablicy.
* Podobnie, podczas wyszukiwania, algorytm najpierw oblicza hash, a potem przeszukuje listę powiązaną w danym miejscu.

**Złożoność:**

* Czasowa:
  1. W najlepszym przypadku (gdy brak kolizji) wynosi **O(1)**, ponieważ element trafia na unikalny indeks.
  2. W najgorszym przypadku (wszystkie elementy mają ten sam hash) wynosi **O(n)**, ponieważ cała lista w jednym miejscu tablicy musi zostać przeszukana.
* Pamięciowa: **O(n)**, ponieważ każde miejsce w tabeli może przechowywać listę powiązaną, co wymaga dodatkowej pamięci.

**Tabele haszujące**

**Tablica haszująca** to struktura danych, która służy do przechowywania i szybkiego wyszukiwania elementów, przy użyciu funkcji haszującej, która mapuje każdy element na unikalny indeks w tablicy.

**Działanie:**

* **Funkcja haszująca**: Każdy element wstawiany do tablicy haszującej jest najpierw przetwarzany przez funkcję haszującą. Funkcja ta przypisuje elementowi wartość liczbową (hash), która służy jako indeks w tablicy.
* **Kolizje**: Jeżeli dwa elementy mają ten sam hash (kolizja), stosuje się różne techniki rozwiązywania kolizji, takie jak:
  1. **Łańcuchowanie** (Chaining): Przechowywanie wielu elementów w tym samym miejscu tablicy w postaci listy.
  2. **Przeszukiwanie otwarte** (Open addressing): Szukanie innego miejsca w tablicy na podstawie z góry ustalonej strategii (np. liniowe, kwadratowe, podwójne haszowanie).
* **Złożoność**:
  1. Czasowa: W idealnym przypadku (brak kolizji) wynosi **O(1)**, ponieważ dostęp do elementu odbywa się bezpośrednio przez indeks.
  2. Pamięciowa: **O(n)**, gdzie n to liczba elementów, ponieważ każdy element musi być przechowywany w tablicy.

**Zastosowania:**

* Tablice haszujące są szeroko stosowane w wielu dziedzinach informatyki, w tym w bazach danych, kompilatorach, systemach rozproszonych i wielu algorytmach, gdzie szybkie wyszukiwanie, wstawianie i usuwanie elementów jest kluczowe.

1. Dół formularza
2. **Interfejs graficzny aplikacji**

**Obraz zawierający tekst, zrzut ekranu, numer, Równolegle

Opis wygenerowany automatycznie**

Aplikacja składa się z tabeli, która ma stały rozmiar, jednakże jest ona wyposażona w scrollbar dzięki czemu użytkownik może przesuwać się po tabeli.   
Poniżej znajdują się pola tekstowe:

* Typu tekstowego:
  + Imię
  + Nazwisko
  + Nr Telefonu
* Typu liczbowego:
  + Wiek
  + Wzrost
* Typu Binarnego
  + Status Studenta
  + Zatrudnienie

Po wypełnieniu pól tekstowych i po naciśnięciu przycisku „Dodaj użytkownika” rekord zostanie dodany do tabeli, następnie jej widok zostanie automatycznie przeładowany dzięki czemu użytkownik natychmiast zaobserwuje efekt wykonanej operacji.

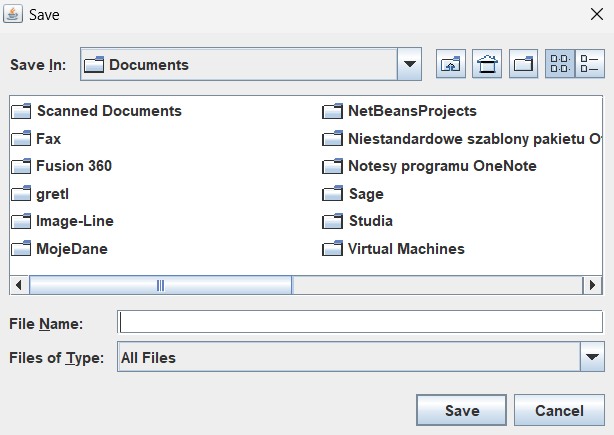
W dolnej części aplikacji znajduję się 10 przycisków funkcyjnych:

* *Generuj użytkowników* – po naciśnięciu przycisku wyświetla się okno, z zapytaniem jaką ilość rekordów ma zostać wygenerowaniu. Po zatwierdzeniu odpowiedniej ilości przez użytkownika widok tabeli zostanie zaktualizowany o wygenerowane losowo rekordy.

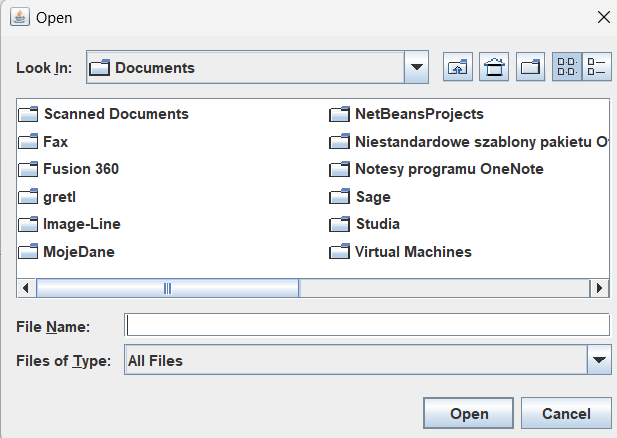
Obraz zawierający tekst, zrzut ekranu, oprogramowanie, wyświetlacz

Opis wygenerowany automatycznie

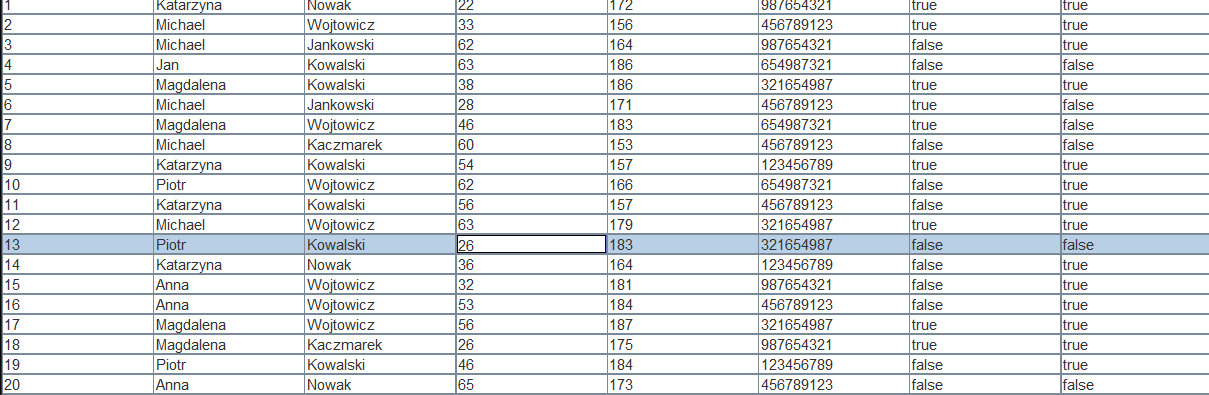
* *Eksportuj do CSV –* po naciśnięciu wyświetla się okno file chooser, dzięki któremu możemy określić lokalizacje zapisywanego pliku w formacie CSV.

**

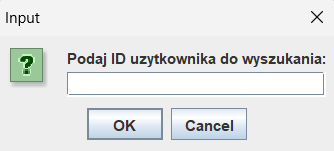
* *Importuj do CSV – po naciśnięciu wyświetla się okno file chooser, dzięki któremu możemy wyszukać oraz wczytać pliki w formacie CSV.*

**

* *Pokaz wszystkich użytkowników – po naciśnięciu wyświetla się pełna lista użytkowników*.
* *Usuń wszystkich użytkowników –* po naciśnięciu usunięta zostaje cała tabela.
* *Usuń użytkownika –* po naciśnięciu usunięty zostaje uprzednio zaznaczony rekord tabeli ( zaznaczony rekord podświetla się na niebiesko ).

**

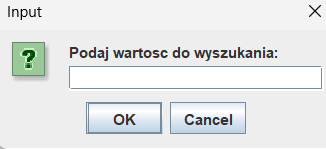
* *Wyszukaj użytkownika* – po naciśnięciu wyświetla się okno z zapytaniem jakie ID użytkownika nas interesuje, po zatwierdzeniu rezultatem wyszukiwania jest wyświetlenie się konkretnego użytkownika

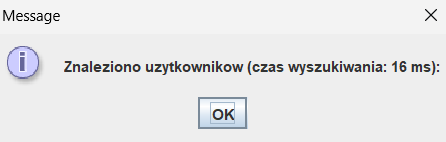
**

* *Wyszukiwanie liniowe –* po naciśnięciu wyświetla się okno z zapytaniem wg której kolumny tabeli mają być wyszukiwane rekordy, po zatwierdzeniu wyświetla się kolejne okno, tym razem zapytaniem o frazę kluczową, następnie zostaną wyświetlone rekordy, które zawierają w danej kolumnie frazę kluczową – funkcjonalność oparta jest o wyszukiwanie liniowe.
* *Wyszukiwanie binarne –* po naciśnięciu wyświetla się okno z zapytaniem wg której kolumny tabeli mają być wyszukiwane rekordy, po zatwierdzeniu wyświetla się kolejne okno, tym razem zapytaniem o frazę kluczową, następnie zostaną wyświetlone rekordy, które zawierają w danej kolumnie frazę kluczową – funkcjonalność oparta jest o wyszukiwanie binarne.

*Obraz zawierający tekst, zrzut ekranu, oprogramowanie, wyświetlacz

Opis wygenerowany automatycznie*

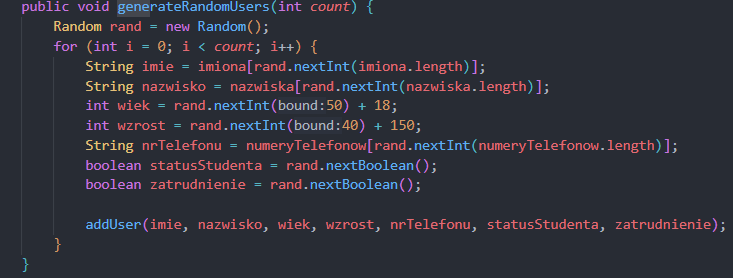
**

**

* *Wyszukiwanie łańcuchowe –* po naciśnięciu wyświetla się okno z zapytaniem wg której kolumny tabeli mają być wyszukiwane rekordy, po zatwierdzeniu wyświetla się kolejne okno, tym razem zapytaniem o frazę kluczową, następnie zostaną wyświetlone rekordy, które zawierają w danej kolumnie frazę kluczową – funkcjonalność oparta jest o wyszukiwanie łańcuchowe.

1. **Opis funkcjonalności**

*Generowanie losowych użytkowników* oparte jest o funkcje, która w sposób losowy wybiera imię, nazwisko oraz numer telefonu z tablic, które uprzednio zostały przygotowane. Wiek, wzrost, status studenta oraz zatrudnienie uzupełniany jest w wartości z odpowiedniego przedziału. Dla wieku są to wartości od 18 do 68, dla wzrostu od 150 do 190, pola status studenta oraz zatrudnienie to wartości True lub False.

****

*Eksport do CSV* - funkcjonalność wykonana jest w oparciu o klasę BufferedWriter, która służy do zapisywania tekstu do pliku. Została ona użyta tutaj ze względu na efektywniejszy sposób zapisy aniżeli w przypadku FileWriter. Klasa działa na zasadzie przechowywania danych w buforze, które są zapisywane dopiero po jego zapełnieniu co zmniejsza ilość operacji wejścia/wyjścia. Funkcja przechodzi przez wszystkich użytkowników, konwertuje dane na ciąg znaków, zapisuje a następnie przechodzi do kolejnej linii. Kończymy działanie funkcji metodą .close(), która zamyka strumień i zapisuje pozostałe dane znajdujące się w buforze do pliku.

*Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie*

*Import do CSV* – Funkcja oparta na klasie BufferedReader, która podobnie jak BufferedWriter używa buforowania, aby zmniejszyć ilość operacji wejścia wyjścia.

Zapisujemy do zmiennej line odczyt z całej linii tekstu następnie do utworzonej tablicy data umieszczamy elementy z całej linii, użyta jest tu metoda .split(), która rozdziela nam wyrazy z linii w oparciu o podany separator, którym w tym wypadku jest przecinek. Następnie w oparciu o te dane tworzona jest nowa instancja obiektu User oraz jest ona dodawana do tablicy users

Obraz zawierający zrzut ekranu

Opis wygenerowany automatycznie

*Usuń wszystkich użytkowników*  - funkcjonalność oparta o metodę .clear(), która czyści listę użytkowników.

Obraz zawierający tekst, Czcionka, zrzut ekranu, linia

Opis wygenerowany automatycznie

*Usuń użytkownika*  - funkcjonalność oparta o metodę list .removeIf(), która usuwa element po spełnieniu konkretnego warunku, który w tym wypadku jest nr ID danego użytkownika. Obiekt User posiada metodę, która zwraca jest ID.

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

*Usuń wybranego użytkownika -*  funkcjonalność oparta jest o pobieraniu wartości id zaznaczonego rekordu w tabeli, a następnie na tej podstawie usuwany jest użytkownik z listy użytkowników oraz w widoku tabeli usuwany jest rekord danego użytkownika.

W przypadku gdy naciśniemy na przycisk usuwania wybranego użytkownika a nie mamy zaznaczonego rekordu wyświetli się komunikat „Wybierz użytkownika do usunięcia”.

Obraz zawierający tekst, zrzut ekranu, Czcionka

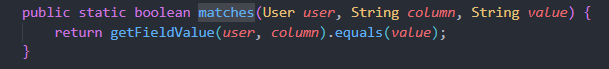
Opis wygenerowany automatycznie

*Wyszukiwanie liniowe*  - funkcjonalność wykonana w oparciu o podstawowe metody. Jako argumenty funkcji wczytujemy listę wszystkich użytkowników, nazwę kolumny względem, której dokonujemy operacji wyszukiwania oraz wartość, której szukamy.

Następnie używając metody pomocniczej matches wczytujemy podane wartości do funkcji, która używa kolejnej pomocniczej funkcji getFieldValue, która znajduje wszystkie informacje o danych rekordzie, w przypadku gdy .equals() wykaże nam wartość true zwracamy dany rekord i dodajemy go do naszej listy rezultatów, która zawiera tylko i wyłącznie rekordy, które są wynikiem naszych wyszukiwani.

*Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie*

**

*Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie*

*Wyszukiwanie binarne*  w danej funkcjonalności został zaimplementowany standardowy algorytm wyszukiwania binarnego z dodatkową funkcją sortującą, gdyż posortowane dane są warunkiem koniecznym do zaimplementowania danej funkcji sortującej . Początkowo przed wykonaniem poniżej metody wykonuje się sortowanie listy użytkowników względem podanej kolumny.

Wykorzystujemy metodę .sort() z klasy List oraz Comparator.comparing() do posortowania kolekcji obiektów na podstawie dynamicznie określonego pola.

Następnie przypisujemy wartość left = 0 oraz right – długość tablicy minus 1, wyznaczamy środek, wartość środkową, jeśli wartość środkowa jest poszukiwana zwracamy ją, następnie sprawdzamy czy mamy powtórzenia danej wartości, sprawdzamy do deklarując indeks „i” większy oraz mniejszy od wartości środkowej. W przypadku gdy wartość środkowa nie jest równa wartości szukanej sprawdzamy wartość powyżej oraz poniżej wartości środkowej.

*Obraz zawierający tekst, zrzut ekranu, wyświetlacz, oprogramowanie

Opis wygenerowany automatycznie*

*Wyszukiwanie łańcuchowe* – jest to funkcjonalność, która została oparta na dostępnym rozwiązaniu w języku Java wykorzystującym hashMapy, która budowana jest w tle, a następnie możemy z niej korzystać przyśpieszając działanie wyszukiwania.

*Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie*

*Wyświetlanie wyników wyszukiwania*  jest to dodatkowa funkcjonalność, wyświetlająca wyniki wyszukiwania liniowego, binarnego oraz łańcuchowego. Funkcjonalność w przypadku pustej listy rezultatów zwraca nam komunikat o braku wyników, natomiast w przypadku gdy lista nie jest pusta zwracam nam wiadomość o znalezieniu użytkowników. Warto również zaznaczyć, że na potrzeby testowania wydajności poszczególnych metod wyszukiwania w poniższej metodzie został zaimplementowany czas wyszukiwania, który obliczany jest w każdej metodzie wyszukiwania, a następnie przekazywany jest jako parametr funkcji.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

*Klasy zawarte w aplikacji*

UserApp – klaas całej aplikacji

UserMenager – klasa zawierająca tablice imion, nazwisk oraz metody pozwalające dokonywać operacji

User – klasa poszczególnego rekordu.

1. Porównanie szybkości działania poszczególnych algorytmów wyszukiwania

*Wyszukiwanie liniowe*

Wraz ze wzrostem liczby rekordów czas operacji wydłuża się i nie ma tutaj żadnego znaczenia typ pola, względem którego przeszukujemy zbiór danych, gdyż w tej metodzie funkcja przechodzi przez wszystkie elementy listy. Złożoność obliczeniowa tej metody to O(n)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ilość rekordów w tys.** | 100 | 300 | 500 | 1000 | 5000 |
| **ID** | 23 | 38 | 81 | 75 | 211 |
| **Imię** | 7 | 33 | 38 | 77 | 250 |
| **Wiek** | 8 | 22 | 59 | 50 | 250 |
| **Zatrudnienie** | 6 | 23 | 51 | 88 | 300 |
| **rodzaj wyszukiwania** | liniowe | liniowe | liniowe | liniowe | liniowe |

*Wyszukiwanie binarne*

Podczas pierwszej próby wyszukania zbioru danych rekordów względem jakiejkolwiek kolumny czas operacji jest dłuższy nawet dwadzieścia razy. Jest to spowodowane faktem, iż wyszukiwanie binarne wymaga posortowanych danych, zatem do czasu wyszukiwania, którego złożoność obliczeniowa wynosi O(log n ), musimy również dodać czas użytego algorytmu sortowaniem którego złożoność obliczeniowa to O(n log n ).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| PIERWSZE WYSZUKANIE | | | | | |
| **ilość rekordów w tys.** | 100 | 300 | 500 | 1000 | 5000 |
| **ID** | 41 | 49 | 69 | 158 | 1000 |
| **Imie** | 112 | 147 | 225 | 412 | 2491 |
| **Wiek** | 115 | 271 | 604 | 875 | 4644 |
| **Zatrudnienie** | 64 | 248 | 376 | 753 | 2743 |
| **rodzaj wyszukiwania** | binarne | binarne | binarne | binarne | binarne |

Podejmując się operacji po raz kolejny w trakcie trwania programu czasy ulegają nawet pięciokrotnemu skróceniu. W tym przypadku algorytm ten w większości przypadków osiąga czasy zbliżone to wyszukiwania liniowego. Jednakże warto tutaj zaznaczyć, że w każdym wykonaniu algorytm sprawdza czy dane są posortowane. W przypadku posortowanych danych sam czas operacji sortowania jest szybszy od liniowego – prezentowany czas sumuje czas sprawdzenia czy dane są posortowane oraz wyszukiwanie.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| PONOWNE WYSZUKANIE | | | | | |
| **ilość rekordów w tys.** | 100 | 300 | 500 | 1000 | 5000 |
| **ID** | 9 | 20 | 50 | 79 | 450 |
| **Imie** | 15 | 26 | 53 | 92 | 486 |
| **Wiek** | 18 | 45 | 81 | 169 | 731 |
| **Zatrudnienie** | 21 | 69 | 119 | 230 | 1100 |
| **rodzaj wyszukiwania** | binarne | binarne | binarne | binarne | binarne |
|  |  |  |  |  |  |

*Wyszukiwanie łańcuchowe*

Metoda wyszukiwania łańcuchowego w pierwszej próbie wyszukania osiąga czasy zbliżone do wyszukiwania linowego raz wyszukiwania binarnego – w ponownych próbach. Natomiast dzięki zaimplementowaniu hashmapy, które generowane są w tle podczas pierwszego wykonania algorytmu na danej kolumnie. Sprawia to, że złożoność obliczeniowa w najbardziej optymistycznym scenariuszu ( bez kolizji ) osiąga wartość O(1).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ilość rekordów w tys.** | 100 | 300 | 500 | 1000 | 5000 |
| **ID** | 22 | 30 | 50 | 62 | 378 |
| **Imię** | 6 | 20 | 21 | 68 | 222 |
| **Wiek** | 8 | 15 | 22 | 50 | 264 |
| **Zatrudnienie** | 7 | 22 | 49 | 76 | 448 |
| **rodzaj wyszukiwania** | łańcuchowe | łańcuchowe | łańcuchowe | łańcuchowe | łańcuchowe |

Wnioski:

Sortowanie liniowe mimo, iż jest najprostsze w implementacji to przy większych zbiorach danych zaczyna być nieefektywny. W tym przypadku wyszukiwanie łańcuchowe ze złożonością O(1) sprawdza się wyśmienicie, jednakże dane muszą mieć już uprzednio stworzoną strukturę. Również w porównaniu z wyszukiwaniem binarnym liniowe przy większych zbiorach danych zaczyna prezentować się niezbyt korzystnie. Jednakże warto tu zaznaczyć, że by wyszukiwanie te trwało mniej czasu trzeba zaimplementować mechanizm tworzenia pomocniczych tablic, które zawierają posortowane dane.

Podsumowując, czasy wyszukiwania dla poszczególnych algorytmów – pomijając już czasy sortowania to ( zakładając, że struktura indeksowa oraz posortowane tablice pomocnicze istnieją) :

Liniowe – O(N)

Binarne – O(lon N)

Łańcuchowe - O(1)