Sprawozdanie z projektu programistycznego

Znajdowanie elementów powtarzających się

Imię i nazwisko: Szymon Ostrowski
Numer grupy: P06
Data: 4 grudnia 2024

Środowisko: Code::Blocks IDE, język C++

GitHub jako repozytorium kodu.

Spis treści

1 Wstęp

Celem projektu jest zaimplementowanie dwóch metod znajdowania duplikatów w tablicy liczb całkowitych. Programy zostały napisane w języku C++ i testowane w środowisku Code::Blocks IDE. Pierwotnie implementacja została zrealizowana w jednym programie, ale w celu lepszego zrozumienia działania obu metod (brute force i tablicy pomocniczej) zostały one rozdzielone na dwa odrębne programy. W sprawozdaniu opisano problem, zastosowane podejście teoretyczne, szczegóły implementacji oraz wyniki testów.

Zauważalnym celem projektu jest także porównanie wydajności obu podejść, a także analiza ich złożoności czasowej i pamięciowej, co pozwala na wyciągnięcie wniosków dotyczących ich zastosowania w praktyce.

2 Opis problemu

Zadanie polega na znalezieniu duplikatów w tablicy liczb całkowitych. Istnieje wiele różnych podejść do rozwiązania tego problemu, które mogą się różnić zarówno złożonością czasową, jak i wymaganiami dotyczącymi pamięci. W zależności od specyfiki danych wejściowych i wymagań dotyczących wydajności, wybór odpowiedniej metody może się różnić. W tym projekcie skupimy się na dwóch popularnych metodach:

- Metoda **brute force**, która sprawdza każdą możliwą parę elementów tablicy. Jest to podejście bardzo proste w implementacji, ale niewydajne dla dużych danych.
- Metoda z wykorzystaniem **tablicy pomocniczej**, która przechowuje liczbę wystąpień każdego elementu w osobnej tablicy. To podejście jest bardziej wydajne, ale wymaga dodatkowej przestrzeni pamięciowej.

Programy zostały zaimplementowane jako dwa oddzielne pliki, które odczytują dane wejściowe z pliku tekstowego, przetwarzają je i zapisują wyniki do pliku wynikowego. Kluczowym elementem w obydwu metodach jest porównanie liczby operacji wykonywanych w zależności od rozmiaru danych wejściowych.

3 Podstawy teoretyczne

3.1 Złożoność czasowa

Złożoność czasowa obu metod różni się w zależności od zastosowanego podejścia:

- **Metoda brute force** ma złożoność czasową $O(n^2)$, ponieważ wymaga przejścia przez wszystkie możliwe pary elementów w tablicy. Dla dużych danych wejściowych metoda ta staje się bardzo nieefektywna, ponieważ liczba operacji rośnie wykładniczo wraz ze wzrostem liczby elementów w tablicy.
- **Metoda tablicy pomocniczej** ma złożoność czasową $O(n)$, ponieważ każdy element tablicy jest analizowany tylko raz. Jednakże, ta metoda wymaga dodatkowej przestrzeni pamięciowej na przechowanie tablicy pomocniczej. Jest to bardziej efektywne podejście, ale może być problematyczne, gdy dostępna pamięć jest ograniczona.

3.2 Złożoność przestrzenna

- **Metoda brute force** wymaga stałej przestrzeni $O(1)$, ponieważ nie wykorzystuje dodatkowych struktur danych. Jednakże, jej słabością jest wysoka złożoność czasowa. - **Metoda z tablicą pomocniczą** ma złożoność przestrzenną $O(n)$, ponieważ wymaga dodatkowej tablicy do przechowywania liczb wystąpień elementów.

4 Szczegóły implementacji

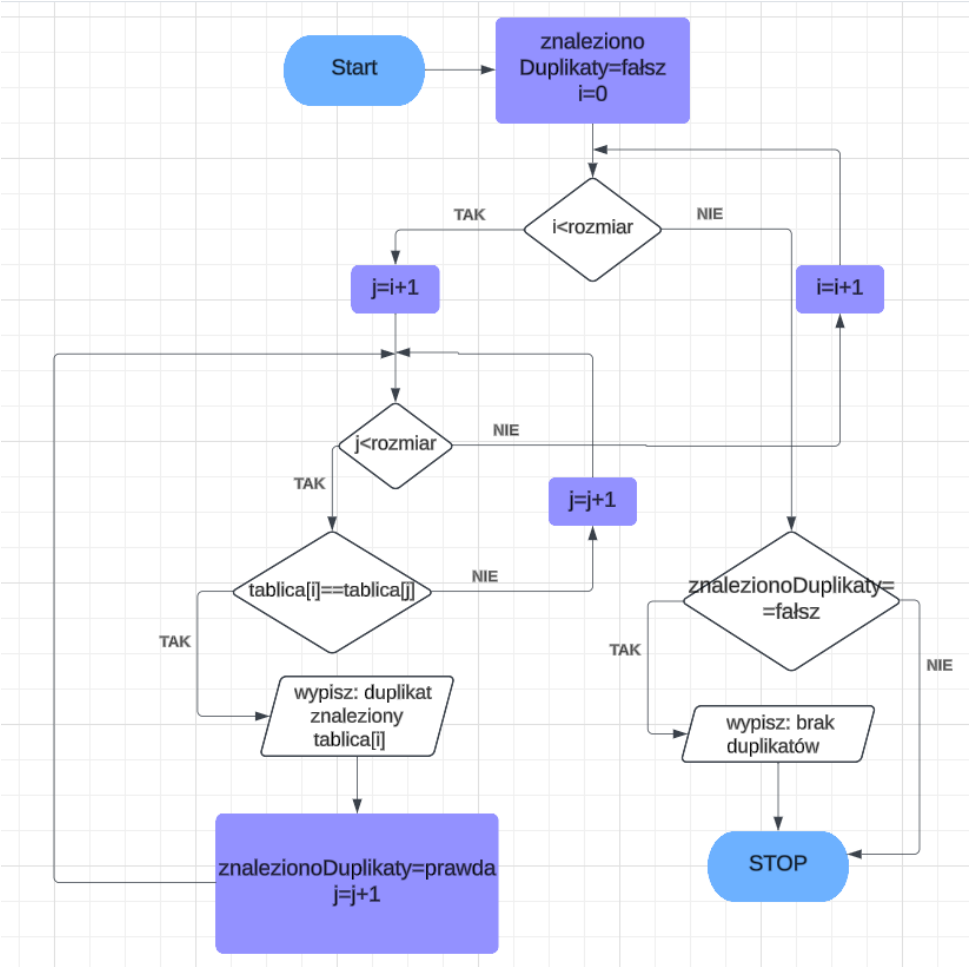
Obie metody zostały zaimplementowane w dwóch oddzielnych programach:

- Program `brute_force.cpp` implementuje metodę brute force.
- Program `hash_map.cpp` implementuje metodę z tablicą pomocniczą.

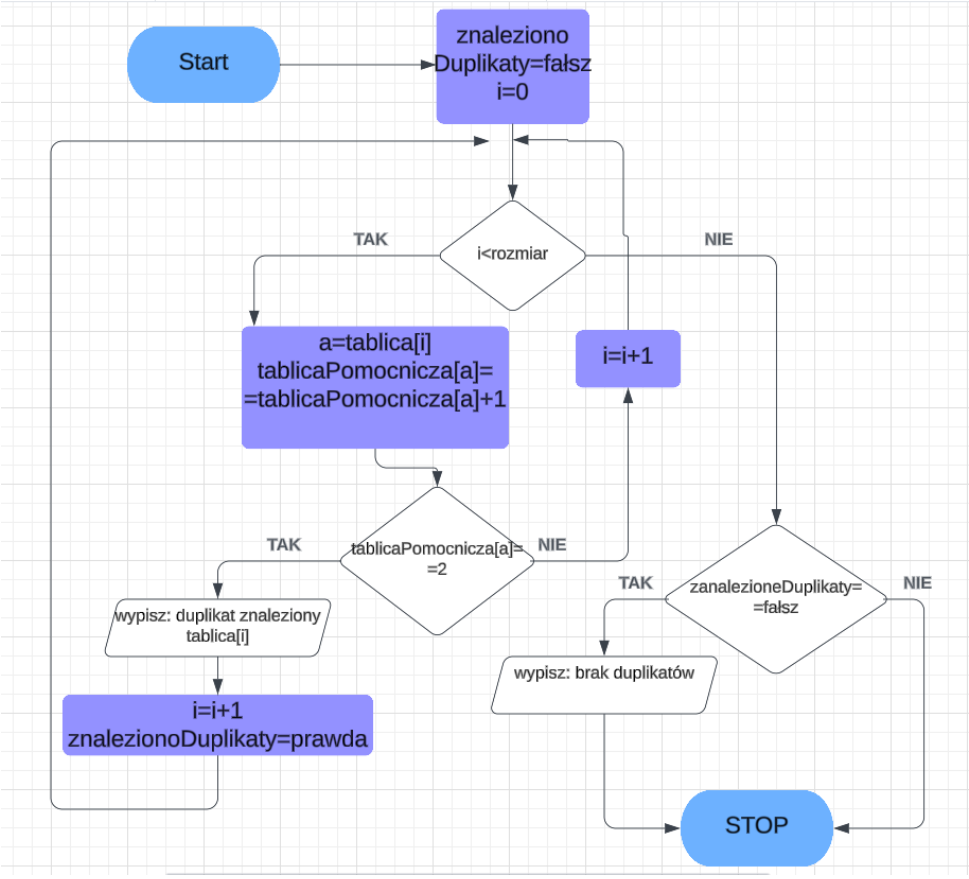
W obu programach dane wejściowe są odczytywane z pliku tekstowego, który zawiera tablicę liczb całkowitych. Programy przetwarzają dane, znajdują duplikaty, a następnie zapisują wyniki do pliku wyjściowego. W obu przypadkach wynikiem jest lista znalezionych duplikatów. Programy zostały również przetestowane na różnych danych wejściowych, aby sprawdzić ich wydajność.

4.1 Schemat blokowy algorytmu

W tym miejscu znajdują się schematy blokowe dla obu metod. Pierwszy pokazuje algorytm brute force, a drugi algorytm z tablicą pomocniczą.



Rysunek 1: Schemat blokowy algorytmu brute force



Rysunek 2: Schemat blokowy algorytmu z tablicą pomocniczą

4.2 Pseudokod algorytmu

Metoda brute force Pseudokod dla metody brute force:

```
Funkcja znajdzDuplikatyBruteForce(tablica, rozmiar, plikWyjsciowy):
Utwórz zbiór duplikaty
Ustaw znalezionoDuplikaty na fałsz
Dla i od 0 do rozmiar-1:
Dla j od i+1 do rozmiar-1:
Jeśli tablica[i] == tablica[j] i tablica[i] nie znajduje się w zbiorze duplikatów:
Zapisz do plikWyjsciowy "Duplikat znaleziony (brute force):", tablica[i]
Dodaj tablica[i] do zbioru duplikatów
Ustaw znalezionoDuplikaty na prawda
Jeśli znalezionoDuplikaty jest fałsz:
Zapisz do plikWyjsciowy "Brak duplikatów (brute force)"
```

Metoda z tablicą pomocniczą Pseudokod dla metody z tablicą pomocniczą:

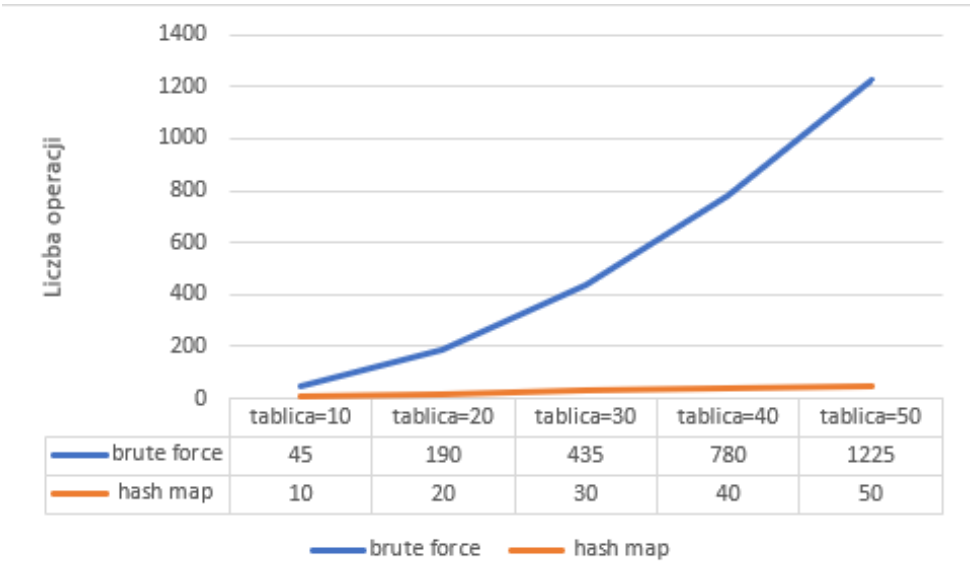
```
Funkcja znajdzDuplikatyTablicaPomocnicza(tablica, rozmiar, plikWyjsciowy):
Utwórz zbiór duplikaty
Utwórz dynamiczną tablicę tablicaPomocnicza o rozmiarze równym rozmiar
Wypełnij tablicaPomocnicza zerami
Ustaw znalezionoDuplikaty na fałsz
Dla i od 0 do rozmiar-1:
Zwiększ wartość tablicaPomocnicza[tablica[i]] o 1
Jeśli tablicaPomocnicza[tablica[i]] == 2 i tablica[i] nie znajduje się w zbiorze dupli
Zapisz do plikWyjsciowy "Duplikat znaleziony (tablica pomocnicza):", tablica[i]
Dodaj tablica[i] do zbioru duplikatów
Ustaw znalezionoDuplikaty na prawda
Jeśli znalezionoDuplikaty jest fałsz:
Zapisz do plikWyjsciowy "Brak duplikatów (tablica pomocnicza)"
```

5 Testowanie

Programy zostały przetestowane na różnych zbiorach danych, obejmujących zarówno małe, jak i duże tablice. W szczególności testy obejmowały:

- Tablice z losowymi liczbami

Wyniki testów są przedstawione poniżej:



Rysunek 3: Porównanie ilości operacji metod brute force i tablicy pomocniczej

6 Wnioski i podsumowanie

Podczas realizacji projektu porównaliśmy dwie metody znajdowania duplikatów w tablicach: brute force i tablicę pomocniczą. Chociaż metoda brute force jest łatwa w implementacji, to jej złożoność czasowa $O(n^2)$ sprawia, że nie nadaje się do pracy z dużymi zbiorami danych. Z kolei metoda z tablicą pomocniczą, choć wymaga dodatkowej przestrzeni pamięciowej, jest zdecydowanie bardziej wydajna, szczególnie w przypadku dużych tablic.

Z analizy wyników testów wynika, że metoda brute force jest praktyczna tylko dla małych rozmiarów danych, natomiast dla większych tablic znacznie lepszym rozwiązaniem jest metoda z tablicą pomocniczą. Czas wykonania metody brute force rośnie wykładniczo z rozmiarem tablicy, podczas gdy czas wykonania metody z tablicą pomocniczą rośnie liniowo.