AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

Algorytm listy dwukierunkowej z zastosowaniem GitHub

Autor:
Jakub Bednarek

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2023

Spis treści

1. Ogólne określenie wymagań

Celem niniejszej sekcji jest przedstawienie ogólnego zarysu wymagań oraz celów, które przyświecały projektowi implementacji listy dwukierunkowej w języku C++. Lista dwukierunkowa, będąca przedmiotem niniejszej dokumentacji, została zaprojektowana z myślą o zapewnieniu efektywnego zarządzania danymi, umożliwiając jednocześnie realizację podstawowych operacji takich jak dodawanie, usuwanie oraz przeszukiwanie elementów w strukturze danych. W dalszej części tego rozdziału zostaną omówione kluczowe aspekty funkcjonalności oraz wydajności, które zostały uwzględnione podczas procesu projektowania i implementacji listy. Dodatkowo, sekcja ta zawiera informacje na temat założeń, ograniczeń oraz strategii testowania przyjętych w ramach projektu.

1.1. Podstawowe operacje na liście

Listy dwukierunkowe, będące rozszerzeniem list jednokierunkowych, umożliwiają efektywne przeprowadzanie szeregu operacji na danych. Poniżej przedstawiono kluczowe operacje, które są dostępne w ramach implementacji listy dwukierunkowej:

1. **Dodawanie elementu:**
    - Na początku listy
    - Na końcu listy
    - W wyznaczonym miejscu listy
2. **Usuwanie elementu:**
    - Z początku listy
    - Z końca listy
    - Z wyznaczonego miejsca listy
3. **Przeglądanie listy:**
    - Przeglądanie elementów od początku do końca
    - Przeglądanie elementów od końca do początku
4. **Inne operacje:**
    - Czyszczenie listy

Celem projektu będzie także zapoznanie się z narzędziem kontroli wersji GitHub w skład którego będzie wchodziło:

- Dodawanie commitów
- Cofanie się o kilka commitów w tył
- Usunięcie jednego commita

1.2. Złożoność obliczeniowa operacji

W kontekście implementacji listy dwukierunkowej, kluczowe jest zapewnienie możliwości efektywnego dodawania i usuwania elementów nie tylko na końcu, ale również na początku struktury danych. W tradycyjnej liście jednokierunkowej operacje takie jak dodawanie lub usuwanie elementu na początku listy są realizowane w czasie stałym, czyli O(1)[1]. W przypadku listy dwukierunkowej, dodatkowe wskaźniki umożliwiają efektywne przeglądanie struktury danych w obu kierunkach, co może być szczególnie przydatne w pewnych kontekstach aplikacyjnych, takich jak algorytmy, które wymagają szybkiego dostępu do sąsiednich elementów.

1.3. Ograniczenia

Podczas projektowania i implementacji listy dwukierunkowej w języku C++ napotkano pewne ograniczenia, które wpływają na sposób użycia oraz zakres aplikacji struktury danych. Poniżej przedstawiono kluczowe ograniczenia, które zostały zidentyfikowane w ramach tego projektu:

1. **Ograniczenie typu danych:**

   Aktualna implementacja listy dwukierunkowej jest ograniczona do przechowywania wyłącznie danych typu `integer`. Oznacza to, że użytkownicy nie mogą przechowywać innych typów danych, takich jak liczby zmiennoprzecinkowe, ciągi znaków czy obiekty niestandardowe, w strukturze listy. To ograniczenie może wpływać na uniwersalność oraz możliwości ponownego użycia implementacji w różnych kontekstach aplikacyjnych.

[1]Szczegółowe omówienie złożoności obliczeniowej różnych operacji na listach można znaleźć w literaturze przedmiotu, np. w pracy Cormena et al.[1].

1.4. Testowanie

Testowanie stanowi kluczowy element procesu rozwoju oprogramowania, mający na celu zapewnienie, że zaimplementowana struktura danych oraz powiązane z nią metody działają zgodnie z oczekiwaniami i są wolne od błędów. W kontekście listy dwukierunkowej, testowanie skupia się na walidacji logicznej poprawności operacji, wydajności oraz stabilności implementacji podczas różnych scenariuszy użycia.

W funkcji main w prosty sposób zostanie przetestowane działanie metod tej struktury danych

```
#include "Lista.h"
int main()
{
  // Tworzenie nowej listy
  Lista lista;

  // Dodawanie elementow do listy
  lista.dodajNaPoczatek(150);
  lista.dodajNaPoczatek(999);
  lista.dodajNaPoczatek(324);
  lista.dodajNaKoniec(1235);
  lista.dodajPodIndex(1, 4);

  // Usuwanie elementow z listy
  lista.usunIndeks(1);
  lista.usunZKonca();
  lista.usunZPoczatku();

  // Wyswietlanie zawartosci listy
  lista.wyswietlListe();
  lista.wyswietlListeOdTylu();
  lista.wyswietlNastepny(0);
  lista.wyswietlPoprzedni(1);

  // Czyszczenie listy
  lista.wyczyscListe();

  // Wyswietlenie zawartosci pustej listy
  lista.wyswietlListe();
  // Zakonczenie programu
  return 0;
}
```

Listing 1. Test metod listy w funkcji main

2. Analiza problemu

2.1. Zastosowania listy dwukierunkowej

Lista dwukierunkowa, jako zaawansowana struktura danych, znajduje zastosowanie w wielu dziedzinach informatyki oraz w różnorodnych aplikacjach i algorytmach. Przykłady użycia mogą obejmować systemy, które wymagają efektywnego dodawania i usuwania elementów, takie jak systemy zarządzania pamięcią, algorytmy przetwarzania danych, czy też w aplikacjach, które wymagają szybkiego dostępu do sąsiednich elementów danych.

2.2. Sposób działania listy dwukierunkowej

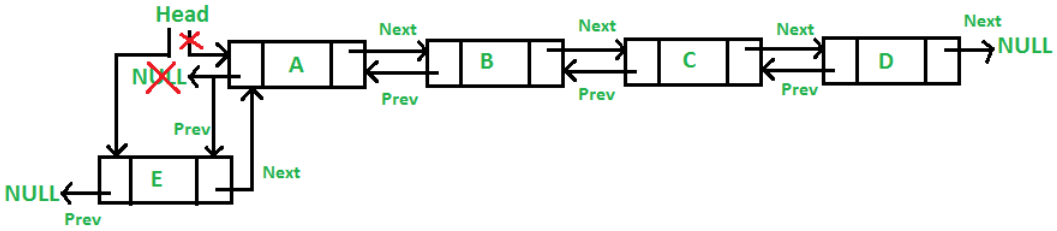
Lista dwukierunkowa jest strukturą danych, która składa się z elementów, zwanych węzłami, gdzie każdy węzeł zawiera przynajmniej dwie referencje (lub wskaźniki) do sąsiednich węzłów: jeden wskazujący na poprzedni węzeł i drugi wskazujący na następny węzeł. Pierwszy element listy jest nazywany głową, a ostatni ogonem.

Rys. 2.1. Element listy dwukierunkowej z wskaźnikami na poprzedni i następny element.

Na Rysunku 2.1 (s. 6) przedstawiono przykładową strukturę listy dwukierunkowej. Każdy węzeł zawiera dane oraz wskaźniki do sąsiednich węzłów.

2.3. Przykłady operacji na liście dwukierunkowej

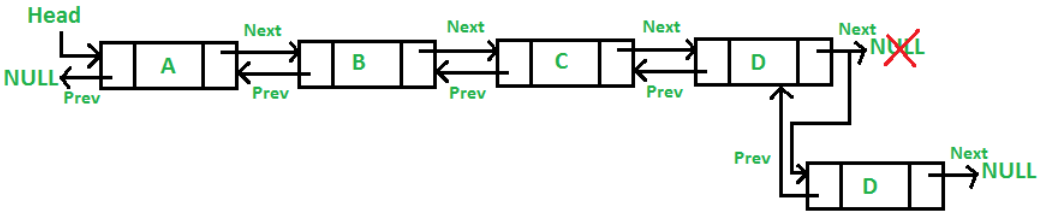
Przykład dodawania elementu na początek listy:



Rys. 2.2. Dodawanie elementu na początek listy dwukierunkowej.[2]

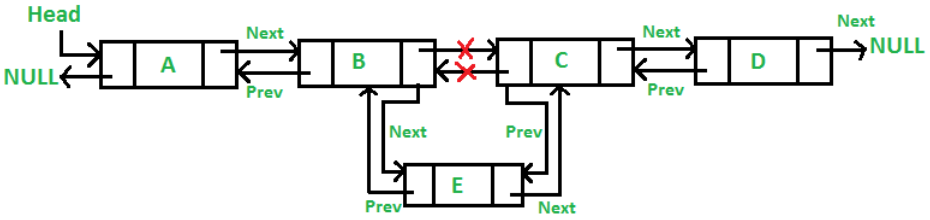
Na Rysunku 2.4 (s. 7) przedstawiono proces dodawania elementu do listy dwukierunkowej. Na rysunku widzimy węzły oznaczone literami od „A" do „D", które są ułożone w kolejności. Wskaźnik „Head" wskazuje na początek listy, a w tym przypadku na węzeł „A". Element "E"to element przygotowany do dodania do listy. Ustawiamy wskaźnik "Next"elementu E na element, na który obecnie wskazuje "Head"(element "A"). Następnie ustawiamy wskaźnik "prev"pierwszego elementu (A) na element "E". Ustawiamy wskaźnik "head"na element "E"a na końcu wskaźnik "Prev"elementu pierwszego (A) na element "E". W ten sposób dodaliśmy element na początek istniejącej listy.

Przykład dodawania elementu na koniec listy:



Rys. 2.3. Dodawanie elementu na koniec listy dwukierunkowej.[3]

Przykład dodawania elementu pod wybrany indeks listy:



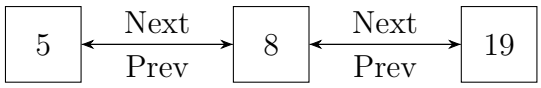
Rys. 2.4. Dodawanie elementu pod indeks listy.[4]

Przykład usuwania elementu z poczatku listy:

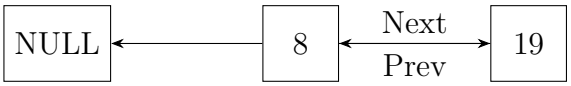
Teraz załóżmy, że chcemy usunąć element o wartości 5 z naszej listy, który jest elementem początkowym. Ilustruje to przykład na rysunku 2.5 (s. 8)

Ilustracje te pokazują w jaki sposób przebiega usuwanie elementu z początku listy. Analogicznie usuwa się element w przodu listy.

[4]Ilustracja ze strony GeeksForGeeks[2]

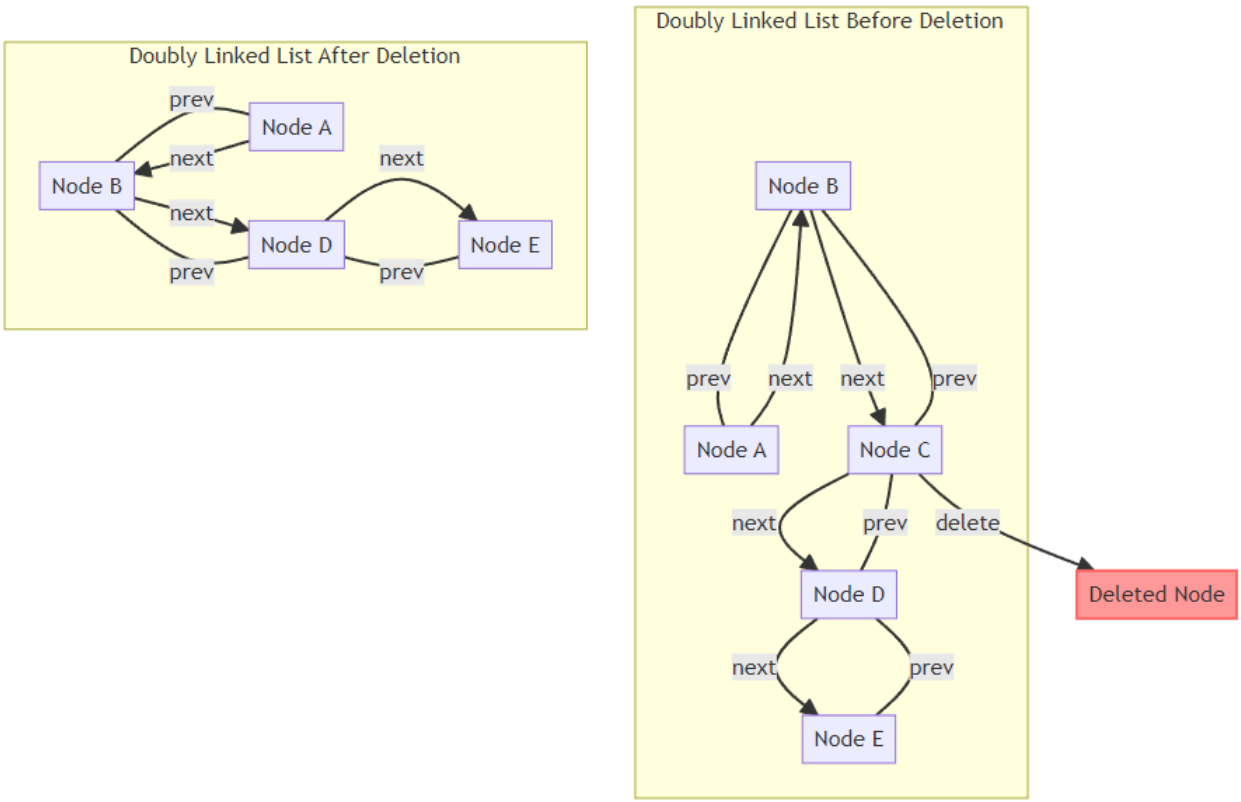


Rys. 2.5. Lista przed usunieciem przodu



Rys. 2.6. Lista po usunieciu przodu

Przykład usuwania elementu poprzez podanie indeksu



Rys. 2.7. Usuwanie elementu po indeksie

Ilustracja 2.7 (s. 8) przedstawia wygląd listy przed i po zastosowaniu usuwania wybranego przez użytkownika elementu listy. Usuwanie takie to proces rozpoczęcia przechozenia listy pod początku aż do miejsca wybranego przez użytkownika a następnie łączenia elementu poprzedzającego go z elementem który występuje po nim.

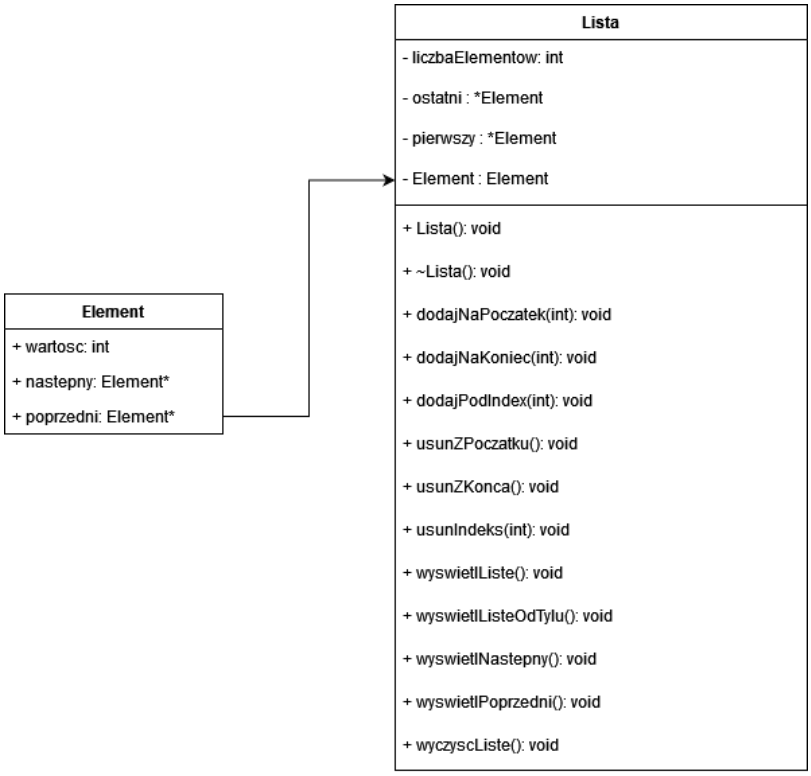
3. Projektowanie

3.1. Narzędzia i Technologie

Realizacja projektu listy dwukierunkowej w języku C++ została przeprowadzona przy użyciu zestawu narzędzi i technologii, które wspierają efektywne i nowoczesne praktyki programistyczne. Poniżej przedstawiono główne komponenty wykorzystane w trakcie prac nad projektem:

- **Język Programowania:** Projekt został zrealizowany w **C++**, języku programowania oferującym bogaty zestaw funkcji i bibliotek, które wspierają zarówno programowanie obiektowe, jak i proceduralne, umożliwiając tworzenie wydajnych i optymalnych struktur danych.
- **Środowisko Programistyczne:** Do implementacji oraz testowania kodu źródłowego użyto **Visual Studio 2022** - zintegrowane środowisko programistyczne (IDE) od Microsoft.
- **Kompilator:** Kod źródłowy został skompilowany przy użyciu **MSVC** (Microsoft Visual C++), który jest częścią środowiska Visual Studio 2022. MSVC jest kompilatorem optymalizującym, który może być używany do tworzenia aplikacji na różne platformy i architektury.
- **Kontrola Wersji:** W celu efektywnego zarządzania wersjami kodu źródłowego wykorzystano system kontroli wersji **Git**. Repozytorium projektu zostało umieszczone na platformie **GitHub**, co ułatwia śledzenie zmian oraz zarządzanie zadaniami.
- **Biblioteki Dodatkowe:** W trakcie realizacji projektu nie korzystano z zewnętrznych bibliotek, wszystkie funkcjonalności zostały zaimplementowane przy użyciu standardowej biblioteki C++.

3.2. Diagram UML klasy implementujaca liste



Rys. 3.1. Schemat UML klasy "Lista"

Rysunek 3.1 (s. 10) przedstawia diagram UML klasy implementującej działanie listy dwukierunkowej.

4. Implementacja

4.1. Struktura kodu

Implementacja listy dwukierunkowej została zrealizowana w języku programowania C++, który jest szeroko stosowany do różnorodnych zastosowań, od systemów wbudowanych po aplikacje desktopowe, ze względu na swoją wydajność i elastyczność. Struktura danych, jaką jest lista dwukierunkowa, pozwala na efektywne dodawanie i usuwanie elementów z dowolnego miejsca listy, gdyż każdy element (węzeł) zawiera wskaźniki zarówno do swojego poprzednika, jak i następnika.

Projekt został rozdzielony na plik zawierający deklaracje klasy **Lista.h** i plik zawierający implementacje w **Lista.cpp**

```
1  /**
2   * Klasa implementujaca liste dwukierunkowa.
3   */
4  class Lista
5  {
6  public:
7      Lista();
8      ~Lista();
9
10     /**
11      * Dodanie elementu na poczatek listy.
12      * @param wartosc Wartosc, ktora ma zostac dodana do listy.
13      */
14     void dodajNaPoczatek(int wartosc);
15
16     /**
17      * Dodanie elementu na koniec listy.
18      * @param wartosc Wartosc, ktora ma zostac dodana do listy.
19      */
20     void dodajNaKoniec(int wartosc);
21
22     /**
23      * Dodanie elementu pod index.
24      * @param index Numer indeksu pod ktorym zostanie wstawiony
    nowy element.
25      * @param wartosc Wartosc, ktora ma zostac dodana do listy.
26      */
27     void dodajPodIndex(int index, int wartosc);
28
29     /**
30      * Usuniecie pierwszego elementu z listy.
```

```
31      */
32     void usunZPoczatku();
33 
34     /**
35      * Usuniecie ostatniego elementu z listy.
36      */
37     void usunZKonca();
38 
39     /**
40      * Usuniecie elementu pod index.
41      * @param index Numer indeksu elementu, ktory ma zostac
   usuniety.
42      */
43     void usunIndeks(int index);
44 
45     /**
46      * Wyswietlenie listy od poczatku.
47      */
48     void wyswietlListe();
49 
50     /**
51      * Wyswietlenie listy od konca.
52      */
53     void wyswietlListeOdTylu();
54 
55     /**
56      * Wyswietlenie kolejnego elementu listy po elemencie o zadanym
       indeksie.
57      * @param index Numer indeksu elementu, po ktorym ma zostac
   wyswietlony kolejny element.
58      */
59     void wyswietlNastepny(int index);
60 
61     /**
62      * Wyswietlenie poprzedniego elementu listy przed elementem o
   zadanym indeksie.
63      * @param index Numer indeksu elementu, przed ktorym ma zostac
   wyswietlony poprzedni element.
64      */
65     void wyswietlPoprzedni(int index);
66 
67     /**
68      * Usuniecie wszystkich elementow z listy.
69      */
70     void wyczyscListe();
```

```
71
72
73 private:
74     /**
75      * Struktura definiujaca element listy.
76      */
77     struct Element
78     {
79         int wartosc; /* Wartosc przechowywana przez element. */
80         Element* nastepny; /* Wskaznik na nastepny element listy.
       */
81         Element* poprzedni; /* Wskaznik na poprzedni element listy.
        */
82     };
83     Element* pierwszy; /* Wskaznik na pierwszy element listy. */
84     Element* ostatni; /* Wskaznik na ostatni element listy. */
85     int liczbaElementow;
86 };
```

Listing 2. Struktura pliku Lista.h

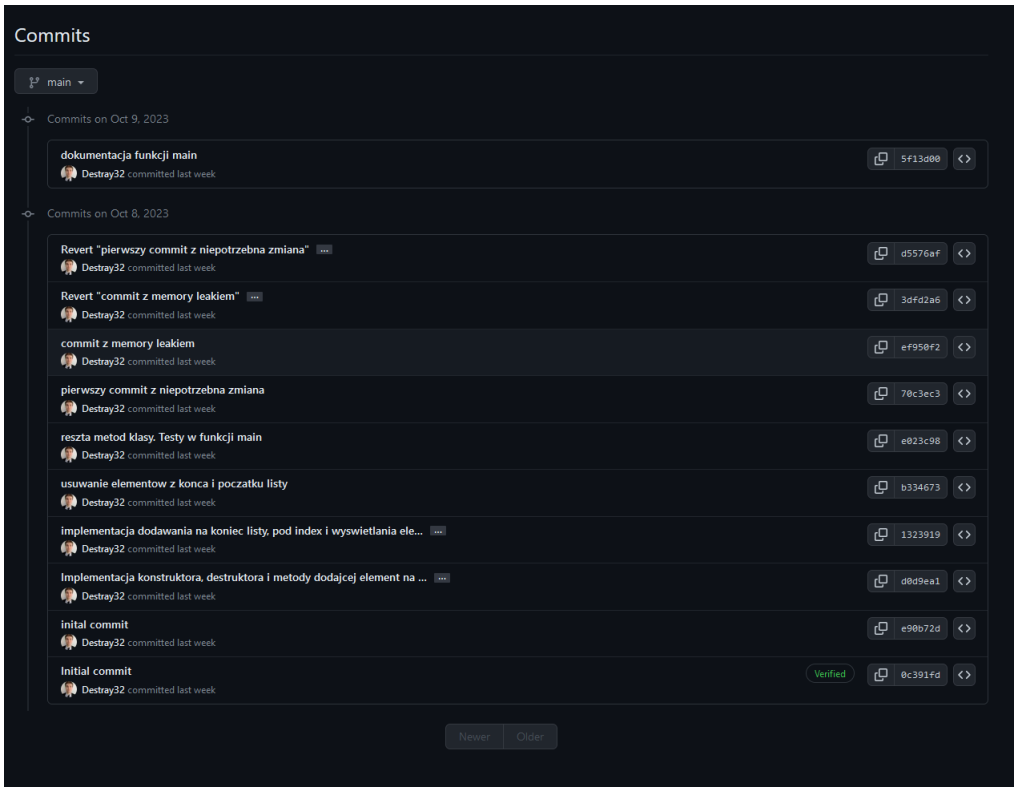
4.2. Mechanizm obsługi błędów

Implementacja listy dwukierunkowej została zrealizowana w języku programowania C++, który jest szeroko stosowany do różnorodnych zastosowań, od systemów wbudowanych po aplikacje desktopowe, ze względu na swoją wydajność i elastyczność. Struktura danych, jaką jest lista dwukierunkowa, pozwala na efektywne dodawanie i usuwanie elementów z dowolnego miejsca listy, gdyż każdy element (węzeł) zawiera wskaźniki zarówno do swojego poprzednika, jak i następnika.

4.3. Repozytorium GitHub

Do przechowywania aktualnej wersji programu w tym projekcie został wykorzystany GitHub, który umożliwia sprawne zarządzanie wersją programu. Zmiany w programie były commitowane na bieżąco do repozytorium w celu możliwości powrotu do wcześniejszej wersji kodu.

Na rysunku 4.1 (s. 14) widoczna jest historia commitow repozytorium pod koniec rozwijania projektu. W historii widoczne jest cofnięcie się o dwa commity, które cofa wskaźnik HEAD na dwa wczesniejsze commity przy okazji wypychając na repozytorium dwa commity



Rys. 4.1. Historia commitów w repozytorium GitHub

5. Wnioski

5.1. Podsumowanie

Projekt listy dwukierunkowej w języku C++ umożliwił zdobycie cennych doświadczeń w tworzeniu i zarządzaniu strukturami danych a także zarządzaniem pamięcią dynamiczną. Lista dwukierunkowa, będąca jedną z fundamentalnych struktur danych, stanowi ważny element w nauce algorytmów i struktur danych, dostarczając jednocześnie praktycznych umiejętności, które mogą być wykorzystane w różnorodnych dziedzinach informatyki.

1. **Zrozumienie struktur danych**
    - Implementacja listy dwukierunkowej pozwoliła na pogłębienie wiedzy na temat działania tej struktury danych, w tym na zrozumienie, jak zarządzać wskaźnikami i pamięcią w kontekście dodawania i usuwania elementów.
2. **Nauka kontroli wersji GitHub**
    - Projekt umożliwił poznanie narzędzia GitHub i tego jak umożliwia kontrolowanie akutalnego kodu projektu

Bibliografia

[1] Thomas H. Cormen i in. *Introduction to Algorithms*. 3rd. MIT Press, 2009. ISBN: 978-0-262-03384-8.

[2] *Strona internetowa GeeksForGeeks*. URL: https://www.geeksforgeeks.org/introduction-and-insertion-in-a-doubly-linked-list/?ref=lbp (term. wiz. 15. 10. 2023).

Spis rysunków

Spis tabel

Spis listingów