

# AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Wydział Nauk Inżynierskich  
Katedra Informatyki

## DOKUMENTACJA PROJEKTOWA ZAAWANSOWANE PROGRAMOWANIE

### **...Algorytm listy dwukierunkowej z zastosowaniem GitHub...**

Autor:  
Kamil Gruca

Prowadzący:  
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

# Spis treści

<b>1. Ogólne określenie wymagań</b>	<b>3</b>
<b>2. Analiza problemu</b>	<b>4</b>
2.1. Działanie algorytmu . . . . .	4
2.1.1. Dodawanie elementów na początek listy . . . . .	5
2.1.2. Dodawanie elementów na koniec listy . . . . .	5
2.1.3. Dodawanie elementów pod wskazany indeks . . . . .	6
2.1.4. Usuwanie elementu z początku listy . . . . .	6
2.1.5. Usuwanie elementu z końca listy . . . . .	6
2.1.6. Usuwanie elementu z wskazanego indeksu listy . . . . .	6
2.1.7. Wyświetlanie elementów listy . . . . .	7
2.2. Zastosowanie list dwukierunkowych . . . . .	7
<b>3. Projektowanie</b>	<b>8</b>
3.1. Języki programowania wykorzystane w projekcie . . . . .	8
3.2. Narzędzia które zostały wykorzystane w projekcie . . . . .	8
3.3. System kontroli wersji GIT . . . . .	8
3.3.1. korzystanie z Gita . . . . .	9
<b>4. Implementacja</b>	<b>11</b>
4.1. Rozłożenie metody Dodanie elementu na koniec . . . . .	11
4.2. Rozłożenie metody do wyświetlania elementu po wskazanym indeksie . . . . .	12
4.3. Scenariusze Git . . . . .	14
<b>5. Wnioski</b>	<b>15</b>
5.1. Wnioski płynące z listy dwukierunkowej . . . . .	15
5.2. Wnioski płynące z Gita . . . . .	15
5.3. Podsumowanie . . . . .	15
<b>Literatura</b>	<b>16</b>
<b>Spis rysunków</b>	<b>17</b>
<b>Spis listingów</b>	<b>18</b>

## 1. Ogólne określenie wymagań

Wymagania w przypadku listy dwukierunkowej to funkcje jakie lista dwukierunkowa będzie spełniać

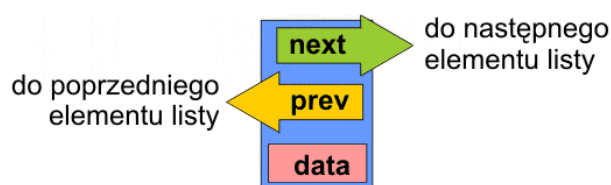
- **Lista dwukierunkowa** - Elementy listy posiadają wskaźnik do następnego i poprzedniego elementu dzięki czemu można przeglądać listę w obie strony
- **Dodawanie elementów** - Program umożliwia dodawanie elementów do listy: na początku, na końcu, oraz pod wskazany indeks.
- **Usuwanie elementu** - Program umożliwia usuwanie elementów z listy: na początku, na końcu, oraz ze wskazanego indeksu.
- **Wyświetlanie elementów** - Program wyświetla całą listę, w odwrotnej kolejności, następny element po wskazanym indeksie, oraz poprzedni element po wskazanym indeksie
- **Czyszczenie listy** - Program usuwa całą listę
- **Projekt wieloplikowy** - Kod źródłowy jest podzielony na pliki źródłowe które zawierają implementacje funkcji i metody klas, oraz na pliki nagłówkowe które zawierają deklaracje klas i metod.

Projekt ma również zawierać pełną dokumentację oraz zawierać zdalne repozytorium w (w tym przypadku Github)

## 2. Analiza problemu

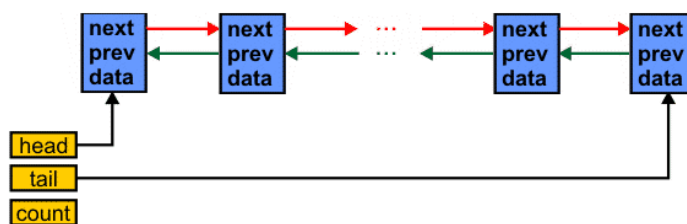
### 2.1. Działanie algorytmu

Lista jest sekwencyjną strukturą danych, która składa się z ciągu elementów tego samego typu. Dostęp do elementów listy jest sekwencyjny – tzn. z danego elementu listy możemy przejść do elementu następnego lub do poprzedniego. Dojście do elementu  $i$ -tego wymaga przejścia przez kolejne elementy od pierwszego do docelowego. Takie zachowanie się tej struktury jest konsekwencją budowy jej elementów. Oprócz samych danych każdy element listy przechowuje zwykle dwa wskaźniki – do elementu następnego listy oraz do elementu poprzedzającego na liście.



Rys. 2.1. Element listy dwukierunkowej

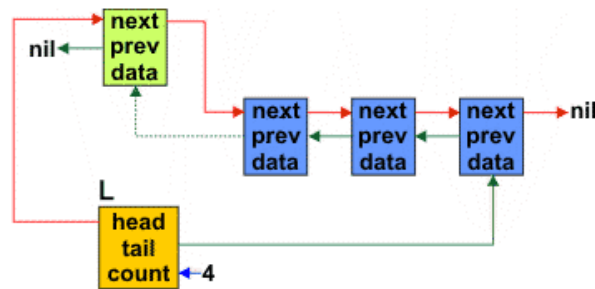
Lista dwukierunkowa posiada swój obiekt reprezentujący który zawiera wskaźnik do pierwszego elementu (head), wskaźnik do ostatniego elementu listy (tail) oraz licznik elementów w liście (count). Przykładowa lista dwukierunkowa:



Rys. 2.2. Lista dwukierunkowa

### 2.1.1. Dodawanie elementów na początek listy

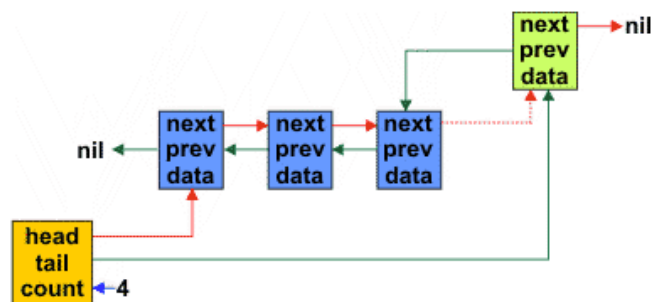
Aby dodać nowy element na początek listy należy utworzyć dynamicznie nowy element w pamięci, następnie umieścić w nim dane, w poli prev trzeba zapisać adres zerowy ponieważ pierwszy element listy nie posiada poprzednika, w poli next należy umieścić adres przechowywany przez pole head, w ten sposób następnikiem nowego elementu stanie się obecny pierwszy element listy, w poli head należy umieścić adres nowego elementu listy i na końcu zwiększyć pole count o 1.



Rys. 2.3. Dodawanie elementu na początek listy

### 2.1.2. Dodawanie elementów na koniec listy

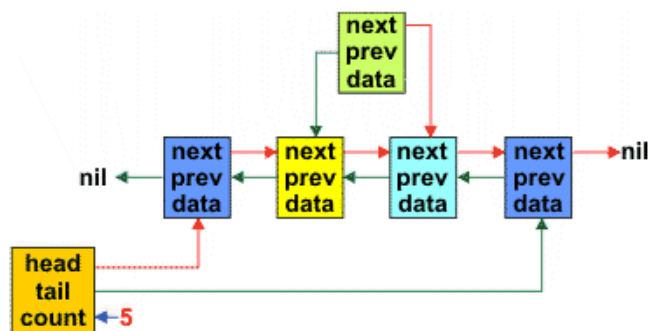
Należy utworzyć w pamięci dynamicznie nowy element i umieścić w nim dane, w poli next umieścić adres zerowy, do pola prev przenieść zawartość pola tail, a w poli tail umieścić adres nowego elementu i na koniec zwiększyć licznik count o 1.



Rys. 2.4. Dodawanie elementu na koniec listy

### 2.1.3. Dodawanie elementów pod wskazany indeks

Przy dodawaniu elementu pod wskazany indeks przeszukujemy listę żeby znaleźć element o indeksie o jeden niższym od docelowego indeksu, następnie modyfikujemy wskaźniki prev oraz next aby dołączyć nowy element, po tych operacjach zwiększamy licznik count o 1.



Rys. 2.5. Dodawanie elementu pod wskazany indeks

### 2.1.4. Usuwanie elementu z początku listy

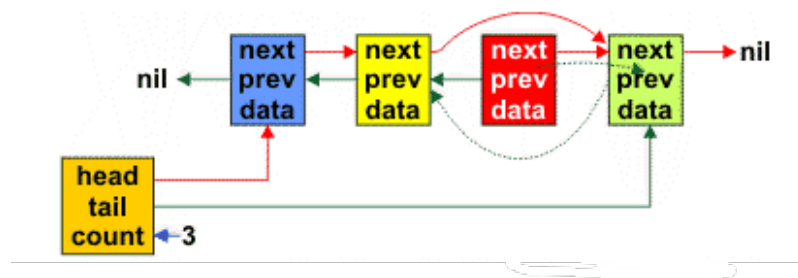
Aby usunąć element z początku listy należy ustawić wskaźnik head na następny element listy, następnie zaaktualizować wskaźnik prev nowego pierwszego elementu na adres zerowy po tym usuwamy element z pamięci i zmniejszamy licznik count o 1.

### 2.1.5. Usuwanie elementu z końca listy

Aby usunąć element z końca listy należy ustawić wskaźnik tail na poprzedni element, następnie zaaktualizować pole next nowego ostatniego elementu na adres zerowy, po tym usuwamy element z pamięci i zmniejszamy licznik count o 1.

### 2.1.6. Usuwanie elementu z wskazanego indeksu listy

Aby usunąć element z wybranego indeksu należy przeszukać listę aby znaleźć wybrany element do usunięcia, następnie zaaktualizować pola next oraz prev sąsiednich elementów i usunąć element, na końcu należy zmniejszyć licznik count o 1.



Rys. 2.6. Usuwanie elementu z listy

### 2.1.7. Wyświetlanie elementów listy

Aby wyświetlić elementy listy możemy użyć do tego wskaźników `next` lub `prev` w zależności od tego w którą stronę chcemy się poruszać i wyświetlać wartości elementów listy. Aby wyczyścić listę należy ustawić wartość `head` oraz `tail` na 0 co oznacza że lista jest pusta.

## 2.2. Zastosowanie list dwukierunkowych

- Listy dwukierunkowe są dobre do implementacji historii działań w edytorze tekstu, dzięki nim można łatwo nawigować między poprzednimi i przyszłymi stanami.
- Listy dwukierunkowe są używane w systemach operacyjnych do zarządzania kolejkami procesów lub zarządzania pamięciami.
- Listy dwukierunkowe mogą być wykorzystywane do reprezentowania struktur takich jak wskaźniki na rekordy, umożliwiając łatwe poruszanie się do poprzednich i następnych rekordów bez potrzeby przeszukiwania całej bazy.
- Listy dwukierunkowe mogą być używane do reprezentacji sąsiadów w wierzchołkach grafu, gdzie konieczne jest szybkie przechodzenie w obu kierunkach, szczególnie w przypadku nieskierowanych grafów.

## 3. Projektowanie

### 3.1. Języki programowania wykorzystane w projekcie

W tym projekcie tak jak i było w wymaganiach wykorzystałem tylko i wyłącznie język C++. Język ten jest doskonałym wyborem do implementacji listy dwukierunkowej ponieważ, pozwala on na manualne zarządzanie pamięcią za pomocą operatorów new i delete, umożliwia to dynamiczne przydzielanie pamięci dla elementów listy. C++ jest również dobrym wyborem ponieważ obsługuje programowanie obiektowe, co pozwala na stworzenie klas do reprezentacji węzłów listy i samej listy. Umożliwia to łatwiejsze zarządzanie kodem, a także wprowadzenie metod operacyjnych

### 3.2. Narzędzia które zostały wykorzystane w projekcie

W tym projekcie do pisania kodu wykorzystałem program Visual Studio 2022. Program ten to jedno z najpopularniejszych zintegrowanych środowisk programistycznych (IDE), które oferuje wiele funkcji i narzędzi ułatwiających programowanie w C++. Visual Studio ma wbudowaną obsługę systemu kontroli wersji Git, co ułatwia zarządzanie kodem źródłowym, śledzenie zmian i współpracę z innymi programistami. Visual Studio oferuje zaawansowane wsparcie dla języka C++ oraz standardowej biblioteki szablonów (STL), co ułatwia implementację i korzystanie z różnorodnych struktur danych i algorytmów.

Visual Studio używa kompilatora Microsoft Visual C++ (MSVC), który jest rozwijany i utrzymywany przez firmę Microsoft. MSVC jest jednym z najpopularniejszych kompilatorów C++ i jest szczególnie dobrze zintegrowany z ekosystemem Windows. MSVC dostarcza narzędzia do debugowania, które pozwalają na analizowanie błędów w kodzie źródłowym. Umożliwia śledzenie wartości zmiennych, ustawianie punktów przerwania oraz analizowanie stosu wywołań, co jest przydatne w przypadku złożonych aplikacji.

### 3.3. System kontroli wersji GIT

W całym projekcie na bieżąco wykorzystywałem system kontroli wersji GIT. Jest to system kontroli wersji, który pozwala na śledzenie zmian w plikach oraz współpracę z innymi programistami. Git jest rozproszonym systemem, co oznacza, że każdy programista posiada pełną kopię repozytorium na swoim lokalnym komputerze, a nie tylko jedną centralną wersję. Ważnym aspektem GITa jest to że przechowuje on pełną historię zmian co pozwala na łatwe przywracanie wcześniejszych

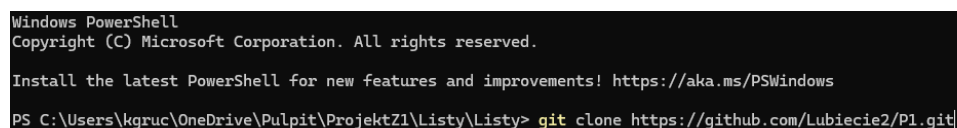


wersji kodu, kolejnym ważnym aspektem Gita jest to że pozwala on na tworzenie gałęzi (branchy) które pozwalają na równoległą pracę nad różnymi wersjami kodu które nie wpływają na główną wersję kodu

W projekcie korzystałem z Githuba do zarządzania systemem kontroli wersji GIT. GitHub to serwis, w którym możesz przechowywać swój kod źródłowy, jak i zarządzać całym procesem wytwarzania oprogramowania: od ukrywania plików źródłowych przed publicznością dzięki repozytoriom prywatnym, przez możliwość ich pobierania dzięki komendzie `git clone`, po prośbę o wprowadzenie zmian dzięki opcji tworzenia pull request

### 3.3.1. korzystanie z Gita

- Aby skopiować repozytorium do naszego lokalnego repozytorium (w tym wypadku z GitHuba) należy użyć komendy `git clone` [link do repozytorium]



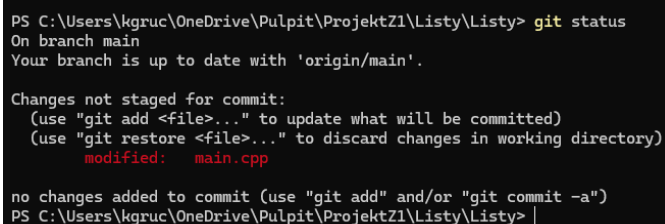
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\kgruc\OneDrive\Pulpit\ProjektZ1\Listy\Listy> git clone https://github.com/Lubiecie2/P1.git
```

Rys. 3.1. git clone

- Teraz możemy wprowadzić zmiany w kodzie. Po tym sprawdzamy w jakich plikach zaszły zmiany za pomocą komendy `git status`



```
PS C:\Users\kgruc\OneDrive\Pulpit\ProjektZ1\Listy\Listy> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   main.cpp

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\kgruc\OneDrive\Pulpit\ProjektZ1\Listy\Listy> |
```

Rys. 3.2. git status

- W tym momencie możemy dodać pliki które zostały zmienione za pomocą komendy `git add`. następnie za pomocą komendy `git commit -m "Komentarz"` commitujemy nasze pliki

```
PS C:\Users\kgruc\OneDrive\Pulpit\ProjektZ1\Listy\Listy> git add .
PS C:\Users\kgruc\OneDrive\Pulpit\ProjektZ1\Listy\Listy> git commit -m "aktualna wersja programu"
[main 1a18ec8] aktualna wersja programu
1 file changed, 20 insertions(+), 9 deletions(-)
PS C:\Users\kgruc\OneDrive\Pulpit\ProjektZ1\Listy\Listy> |
```

**Rys. 3.3.** git commit

- Teraz możemy wypchnąć nasze pliki do zdalnego repozytorium za pomocą komendy `git push`

```
PS C:\Users\kgruc\OneDrive\Pulpit\ProjektZ1\Listy\Listy> git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 461 bytes | 461.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/Lubiecie2/P1.git
259c08b..1a18ec8 main -> main
PS C:\Users\kgruc\OneDrive\Pulpit\ProjektZ1\Listy\Listy> |
```

**Rys. 3.4.** git commit

## 4. Implementacja

### 4.1. Rozłożenie metody Dodanie elementu na koniec

Funkcja Lista Dodanie na koniec wchodzi w skład klasy replisty i służy do dodawania nowego elementu na koniec listy dwukierunkowej.

- Funkcja przyjmuje jeden argument całkowity `v`, który reprezentuje wartość jaka zostanie dodana

```
1 void replisty::Lista_Dodanie_na_koniec(char v)
```

**Listing 1.** Metoda dodanie na koniec

- Tworzony jest dynamicznie nowy element który ma ustawioną wartość `prev` i `next` jako `nullptr`

```
1 elisty* nws = new elisty(v);
```

**Listing 2.** Metoda dodanie na koniec

- Nowy element listy jest dodawany na koniec więc jego wskaźnik `next` jest ustawiany na wartość `nullptr` ponieważ nie ma on następnego elementu. Do pola `prev` nowego elementu jest przenoszona zawartość pola `tail` w ten sposób nowy ostatni element będzie powiązany z obecnie ostatnim elementem

```
1 nws->next = nullptr;  
2 nws->prev = tail;
```

**Listing 3.** Metoda dodanie na koniec

- Pętla sprawdza, czy zmienna `tail` jest ustawiona, jeśli zmienna `tail` ma wartość niezerową to nowy element zostaje przypisany jako następny element obecnego końca listy. Jeśli zmienna `tail` ma wartość `nullptr`, oznacza to że lista jest pusta i nowy element zostaje zarówno jej pierwszym jak i ostatnim elementem, w takim przypadku zmienna `head` zostaje ustawiona na nowy element

```
1 if (tail != nullptr) {  
2     tail->next = nws;  
3 }  
4 else {  
5     head = nws;  
6 }
```

**Listing 4.** Metoda dodanie na koniec

- Zmienna tail zostaje ustawiona tak by wskazywała na nowo dodany element, po tych operacjach licznik count zmniejszany jest o 1

```
1 tail = nws;
2 count--;
```

Listing 5. Metoda dodanie na koniec

Efekt metody:

```
int main()
{
    replisty lista;

    lista.Lista_Dodanie_na_koniec('3');
    lista.Lista_Dodanie_na_koniec('4');

    std::cout << "Aktualna kolejnosc listy po dodaniu na koniec: ";
    lista.Lista_wyswietl();
};
```

Konsola debugowania progra x + v

Aktualna kolejnosc listy po dodaniu na koniec: 3 4

C:\Users\kgruc\OneDrive\Pulpit\ProjektZ1\Listy\x64\Debu  
Naciśnij dowolny klawisz, aby zamknąć to okno...

Rys. 4.1. metoda dodanie na koniec

## 4.2. Rozłożenie metody do wyświetlania elementu po wskazanym indeksie

Metoda wyświetlanie następnego elementu służy do wyświetlania następnego elementu w liście po wskazanym indeksie

- Funkcja przyjmuje jeden argument całkowity v, który reprezentuje indeks po którym zostanie wyświetlony następny element.

```
1 void replisty::Lista_Wyswietlanie_Nastepnego_elementu(int index)
```

Listing 6. Metoda wyswietlanie elementu po indeksie

- Zmienna obecnyelement jest wskaźnikiem który wskazuje na element do którego prowadzi wartość head czyli na pierwszy element listy

```
1 elisty* obecny_element = head;
```

Listing 7. Metoda wyswietlanie elementu po indeksie

- Pętla for jest używana do przejścia przez listę, aby dojść do elementu o wskazanym indeksie, pętla w każdej iteracji przemieszcza wskaźnik aż do elementu o podanym indeksie

```

1   for (int i = 1; i < index; i++) {
2       obecny_element = obecny_element->next;
3   }

```

**Listing 8.** Metoda wyświetlanie elementu po indeksie

- Instrukcja warunkowa sprawdza czy istnieje następny element. Jeśli wskaźnik next nie jest równy nullptr oznacza to że istnieje następny element, w przeciwnym przypadku nie ma następnego elementu.

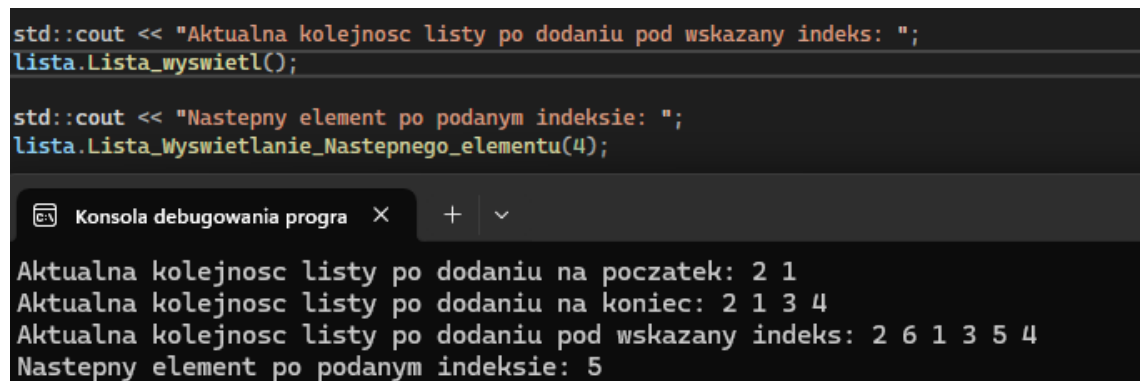
```

1   if (obecny_element->prev != nullptr) {
2       std::cout << obecny_element->prev->data << std::endl;
3   }
4   else {
5       std::cout << "Brak poprzedniego elementu" << std::endl;
6   }

```

**Listing 9.** Metoda wyświetlanie elementu po indeksie

Efekt metody:



```

std::cout << "Aktualna kolejnosc listy po dodaniu pod wskazany indeks: ";
lista.Lista_Wyswietl();

std::cout << "Nastepny element po podanym indeksie: ";
lista.Lista_Wyswietlanie_Nastepnego_elementu(4);

```

Konsola debugowania progra

```

Aktualna kolejnosc listy po dodaniu na poczatek: 2 1
Aktualna kolejnosc listy po dodaniu na koniec: 2 1 3 4
Aktualna kolejnosc listy po dodaniu pod wskazany indeks: 2 6 1 3 5 4
Nastepny element po podanym indeksie: 5

```

**Rys. 4.2.** metoda wyświetlanie wyrazu po indeksie

### 4.3. Scenariusze Git

Podczas realizowania projektu musieliśmy przetestować różne scenariusze korzystania z Gita

- Cofnięcie ostatniego commita

```
PS C:\Users\kgruc\pz\P1\Projekt_Zaawansowany_1> git reset --hard HEAD~1
HEAD is now at 8a21d28 Merge branch 'beta'
PS C:\Users\kgruc\pz\P1\Projekt_Zaawansowany_1> git branch
  beta
* main
PS C:\Users\kgruc\pz\P1\Projekt_Zaawansowany_1> git push origin HEAD --force
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Lubiecie2/Projekt_Zaawansowany_1.git
+ a6cab1c...8a21d28 HEAD -> main (forced update)
PS C:\Users\kgruc\pz\P1\Projekt_Zaawansowany_1>
```

Rys. 4.3. metoda dodanie na koniec

- Korzystanie z brancha pobocznego do tworzenia próbnych wersji projektu a następnie scalanie go do głównej gałęzi



Rys. 4.4. Network graph

- Pobranie projektu z GitHuba do innej lokalizacji na komputerze, dopisanie kolejnej metody w pobranym projekcie, zapisać zmiany na serwerze. Pobranie zmian z GitHuba do pierwszej lokalizacji, dopisanie nowej metody i zapisanie zmiany na serwer.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\kgruc\OneDrive\Pulpit\druga> git clone https://github.com/Lubiecie2/P1.git
```

Rys. 4.5. Pobranie projektu do innej lokalizacji

## 5. Wnioski

### 5.1. Wnioski płynące z listy dwukierunkowej

- Lista dwukierunkowa składa się z elementów, z których każdy zawiera dane oraz dwa wskaźniki: jeden wskazujący na poprzedni element `prev`, a drugi na następny element `next`.
- Możliwość przeglądania elementów w obie strony, co może być przydatne w różnych algorytmach i strukturach.
- Umożliwia łatwe dodawanie i usuwanie elementów z dowolnej pozycji w liście (początek, koniec, środek) bez potrzeby przesuwania innych elementów.
- Przy operacjach na wskaźnikach istnieje ryzyko błędów związanych z nieprawidłowym aktualizowaniem wskaźników, co może prowadzić do uszkodzenia struktury listy.

### 5.2. Wnioski płynące z Gita

- Git umożliwia śledzenie wszystkich zmian w kodzie źródłowym, co ułatwia prace i utrzymanie historii projektu
- Git pozwala na tworzenie gałęzi (branch), co umożliwia równoległe rozwijanie funkcji, testowanie nowych pomysłów lub naprawianie błędów bez zakłócania głównej wersji kodu.

### 5.3. Podsumowanie

Podczas tworzenia tego projektu nauczyłem się jak tworzyć nowe elementy listy, jak dodawać elementy na różny sposób do listy, jak usuwać elementy z listy na różny sposób oraz jak wyświetlać liste na kilka sposobów. Podczas tego projektu nauczyłem się również jak korzystać z Githuba, jak commitować, tworzyć nowe branchy a następnie scalać je do jednej wersji kodu. Listy dwukierunkowe to jedno z bardziej zaawansowanych struktur danych co w niektórych momentach tworzenia kodu sprawiało u mnie pewne problemy.

## Bibliografia

- [1] *Serwis Edukacyjny w I-LO w Tarnowie*. URL: [https://eduinf.waw.pl/inf/alg/001\\_search/0084.php](https://eduinf.waw.pl/inf/alg/001_search/0084.php) (term. wiz. 12.10.2024).
- [2] *binarnie.pl*. URL: <https://binarnie.pl/lista-dwukierunkowa/> (term. wiz. 13.10.2024).
- [3] *embeddeddev.pl*. URL: <http://www.embeddeddev.pl/menu-lcd-listy-jedno-dwukierunkowe/> (term. wiz. 13.10.2024).
- [4] *programistaJAVA*. URL: <https://programistajava.pl/listy-jako-struktura-danych/> (term. wiz. 14.10.2024).



## Spis rysunków

2.1. Element listy dwukierunkowej . . . . .	4
2.2. Lista dwukierunkowa . . . . .	4
2.3. Dodawanie elementu na początek listy . . . . .	5
2.4. Dodawanie elementu na koniec listy . . . . .	5
2.5. Dodawanie elementu pod wskazany indeks . . . . .	6
2.6. Usuwanie elementu z listy . . . . .	7
3.1. git clone . . . . .	9
3.2. git status . . . . .	9
3.3. git commit . . . . .	10
3.4. git commit . . . . .	10
4.1. metoda dodanie na koniec . . . . .	12
4.2. metoda wyswietlanie wyrazu po indeksie . . . . .	13
4.3. metoda dodanie na koniec . . . . .	14
4.4. Network graph . . . . .	14
4.5. Pobranie projektu do innej lokalizacji . . . . .	14

## Spis listingów

1.	Metoda dodanie na koniec . . . . .	11
2.	Metoda dodanie na koniec . . . . .	11
3.	Metoda dodanie na koniec . . . . .	11
4.	Metoda dodanie na koniec . . . . .	11
5.	Metoda dodanie na koniec . . . . .	12
6.	Metoda wyswietlanie elementu po indeksie . . . . .	12
7.	Metoda wyswietlanie elementu po indeksie . . . . .	12
8.	Metoda wyswietlanie elementu po indeksie . . . . .	13
9.	Metoda wyswietlanie elementu po indeksie . . . . .	13