

# AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych  
Katedra Informatyki

## DOKUMENTACJA PROJEKTOWA ZAAWANSOWANE PROGRAMOWANIE

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

Autor:  
Blavitskyi Mykola

Prowadzący:  
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

# Spis treści

<b>1. Ogólne określenie wymagań</b>	<b>4</b>
1.1. Cel Projektu . . . . .	4
<b>2. Analiza problemu</b>	<b>5</b>
2.1. Gdzie używa się algorytmu listy dwukierunkowej? . . . . .	5
2.2. Sposób działania programu/algorytmu . . . . .	5
2.3. Przykład wykorzystania algorytmu . . . . .	5
2.4. Mnożenie macierzy . . . . .	5
2.5. Implementacja w programie C++ . . . . .	6
<b>3. Projektowanie</b>	<b>9</b>
3.1. Wprowadzenie . . . . .	9
3.2. Wykorzystane narzędzia . . . . .	9
3.3. Sposób użycia narzędzi . . . . .	9
3.4. Git . . . . .	9
3.5. Doxygen . . . . .	10
3.6. Overleaf . . . . .	11
<b>4. Implementacja</b>	<b>12</b>
4.1. Wprowadzenie . . . . .	12
4.2. Implementacja algorytmu . . . . .	12
4.3. Struktura Node . . . . .	12
4.4. Klasa DoublyLinkedList . . . . .	12
4.4.1. Dodawanie elementów . . . . .	12
4.4.2. Usuwanie elementów . . . . .	13
4.5. Ciekawe fragmenty kodu . . . . .	14
4.6. Wyniki działania algorytmu . . . . .	14
<b>5. Wnioski</b>	<b>16</b>
5.1. Wnioski Końcowe . . . . .	16
<b>Literatura</b>	<b>17</b>

<b>Spis rysunków</b>	<b>17</b>
<b>Spis tabel</b>	<b>18</b>
<b>Spis listingów</b>	<b>19</b>

# 1. Ogólne określenie wymagań

## 1.1. Cel Projektu

Celem projektu jest stworzenie funkcjonalnej implementacji listy dwukierunkowej w języku C++. Projekt ten ma na celu nie tylko rozwijanie umiejętności programistycznych w zakresie używania języka C++, ale również zapoznanie się z najlepszymi praktykami inżynierii oprogramowania, takimi jak:

- **Zrozumienie struktur danych:** Przez implementację listy dwukierunkowej studenci zdobędą praktyczną wiedzę na temat struktur danych oraz ich zastosowań w programowaniu.
- **Dokumentacja kodu:** Projekt wymaga użycia narzędzia Doxygen do generowania dokumentacji, co pozwoli na naukę jak tworzyć czytelne i zrozumiałe opisy dla kodu, co jest kluczowe w pracy zespołowej.
- **Kontrola wersji:** Użycie systemu kontroli wersji Git i platformy GitHub umożliwi studentom zrozumienie, jak zarządzać zmianami w projekcie, współpracować w zespole oraz stosować dobre praktyki przy commitowaniu i dokumentowaniu kodu.
- **Testowanie i rozwój:** Projekt wymaga przetestowania różnych scenariuszy oraz implementacji nowych funkcji, co sprzyja rozwijaniu umiejętności analitycznego myślenia oraz rozwiązywania problemów.

W wyniku realizacji projektu studenci zdobędą praktyczną wiedzę, która jest niezbędna w późniejszej karierze zawodowej jako programiści, a także nauczą się jak dokumentować swoje projekty oraz efektywnie zarządzać kodem źródłowym.

## 2. Analiza problemu

### 2.1. Gdzie używa się algorytmu listy dwukierunkowej?

Lista dwukierunkowa ma wiele zastosowań w programowaniu i informatyce, w tym:

- **Implementacja kolejek i stosów:** Umożliwia szybkie dodawanie i usuwanie elementów.
- **Aplikacje multimedialne:** Używana w odtwarzaczach wideo i muzyki do zarządzania listą utworów.
- **Edycja tekstu:** Pomaga w przechowywaniu znaków tekstu, co ułatwia wstawianie i usuwanie.
- **Wyszukiwanie:** Optymalizuje przeszukiwanie danych w bazach danych.

### 2.2. Sposób działania programu/algorytmu

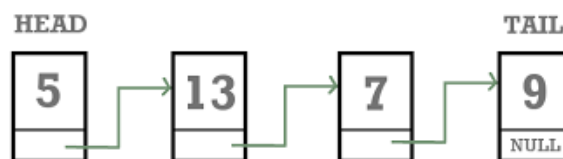
Algorytm listy dwukierunkowej w C++ składa się z następujących elementów:

- **Struktura węzła (Node):** Każdy węzeł zawiera dane oraz wskaźniki do poprzedniego i następnego węzła.
- **Klasa listy dwukierunkowej (DubleLinkedList):** Klasa zarządza całą listą i zawiera metody do dodawania, usuwania oraz wyświetlania elementów.

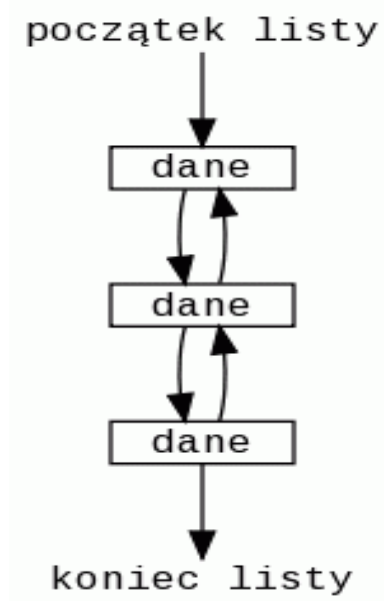
### 2.3. Przykład wykorzystania algorytmu

Aby zobrazować działanie algorytmu listy dwukierunkowej, rozważmy przykład mnożenia macierzy.

### 2.4. Mnożenie macierzy



Rys. 2.1. Jednokierunkowa Lista



Rys. 2.2. Lista Algorytmu i Struktury Danych



Rys. 2.3. Jedno i dwukierunkowe Listy

## 2.5. Implementacja w programie C++

Poniżej przedstawiamy przykładowy kod implementujący algorytm listy dwukierunkowej w C++:

```

1 #include <iostream>
2 #include <cstdlib>
3 #include <ctime>
4
5 using namespace std;
6
7 // Struktura wezla
8 struct Node {
9     int data; // Wartość wezła
10    Node* prev; // Wskaznik na poprzedni wezeł
11    Node* next; // Wskaznik na następny wezeł
12
13    Node(int value) : data(value), prev(nullptr), next(nullptr) {}
14 };
15
  
```

```
16 // Klasa listy dwukierunkowej
17 class DupleLinkedList {
18 private:
19     Node* head; // Wskaznik na pierwszy element listy
20     Node* back; // Wskaznik na ostatni element listy
21
22 public:
23     DupleLinkedList() : head(nullptr), back(nullptr) {}
24
25     // Metody klasy
26     void addToHead(int value);
27     void addToBack(int value);
28     void display();
29     // Inne metody...
30 };
31
32 // Implementacja metod
33 void DupleLinkedList::addToHead(int value) {
34     Node* newNode = new Node(value);
35     if (!head) {
36         head = back = newNode;
37     } else {
38         newNode->next = head;
39         head->prev = newNode;
40         head = newNode;
41     }
42 }
43
44 void DupleLinkedList::addToBack(int value) {
45     Node* newNode = new Node(value);
46     if (!back) {
47         head = back = newNode;
48     } else {
49         back->next = newNode;
50         newNode->prev = back;
51         back = newNode;
52     }
53 }
54
55 void DupleLinkedList::display() {
56     Node* current = head;
57     while (current) {
58         cout << current->data << " ";
59         current = current->next;
60     }
```

```
61     cout << endl;  
62 }
```

**Listing 1.** Fragment kodu - dodawanie elementów



## 3. Projektowanie

### 3.1. Wprowadzenie

W niniejszym dokumencie opisano narzędzia i techniki, które zostaną wykorzystane w projekcie C++. Projekt będzie realizowany w języku C++ i z wykorzystaniem nowoczesnych narzędzi programistycznych.

### 3.2. Wykorzystane narzędzia

W projekcie zastosowane zostaną następujące narzędzia:

- **Język C++:** To język programowania wysokiego poziomu, który jest powszechnie używany w aplikacjach systemowych, gier oraz w programowaniu aplikacji wieloplatformowych.
- **Visual Studio 2022:** Zintegrowane środowisko programistyczne (IDE) od Microsoft, które ułatwia rozwijanie aplikacji w języku C++. Oferuje wiele narzędzi do debugowania, testowania oraz zarządzania projektami.
- **Git:** System kontroli wersji, który umożliwia śledzenie zmian w kodzie źródłowym. Umożliwia współpracę w zespołach oraz efektywne zarządzanie wersjami kodu.
- **GitHub:** Platforma oparta na systemie Git, która umożliwia hostowanie kodu, współpracę nad projektami oraz zarządzanie zadaniami.
- **Doxygen:** Narzędzie do generowania dokumentacji z kodu źródłowego. Umożliwia tworzenie dokumentacji w różnych formatach, takich jak HTML, PDF czy LaTeX, na podstawie specjalnych komentarzy w kodzie.
- **Overleaf:** Aplikacja webowa do edytowania dokumentów w LaTeX. Umożliwia łatwe współdzielenie dokumentów oraz ich wspólną edycję w czasie rzeczywistym.

### 3.3. Sposób użycia narzędzi

### 3.4. Git

Git to system kontroli wersji, który umożliwia zarządzanie historią zmian w kodzie źródłowym. Aby efektywnie korzystać z Git, wykonaj następujące kroki:

- **Inicjalizacja repozytorium:** Użyj komendy `git init` w katalogu projektu, aby utworzyć nowe repozytorium.
- **Dodawanie plików:** Użyj `git add .`, aby dodać wszystkie pliki do repozytorium.
- **Tworzenie commitów:** Użyj `git commit -m "Opis zmian"`, aby zapisać zmiany z odpowiednim komentarzem.
- **Tworzenie gałęzi:** Użyj `git branch nazwa_gałęzi`, aby utworzyć nową gałąź.
- **Przełączanie gałęzi:** Użyj `git checkout nazwa_gałęzi`, aby przełączyć się na inną gałąź.
- **Wysyłanie zmian na GitHub:** Użyj `git push origin nazwa_gałęzi`, aby wysłać zmiany na zdalne repozytorium.

### 3.5. Doxygen

Doxygen jest narzędziem do generowania dokumentacji z kodu źródłowego. Aby z niego skorzystać:

- **Instalacja:** Zainstaluj Doxygen na swoim systemie.
- **Konfiguracja:** Utwórz plik konfiguracyjny `Doxyfile` przy użyciu polecenia `doxygen -g`.
- **Dodawanie komentarzy:** Używaj specjalnych komentarzy w kodzie źródłowym, aby opisać funkcje, klasy i zmienne. Przykład:

```
/**
 * @brief Funkcja dodaje dwie liczby.
 * @param a Pierwsza liczba.
 * @param b Druga liczba.
 * @return Suma a i b.
 */
int add(int a, int b) {
    return a + b;
}
```

- **Generowanie dokumentacji:** Użyj polecenia `doxygen` `Doxyfile`, aby wygenerować dokumentację w formacie HTML lub PDF.

### 3.6. Overleaf

Overleaf to platforma do edytowania dokumentów w LaTeX. Aby z niej skorzystać:

- **Rejestracja:** Załóż konto na stronie Overleaf.
- **Tworzenie projektu:** Utwórz nowy projekt i wybierz szablon dokumentu.
- **Edycja:** Edytuj dokument w edytorze tekstu. Zmiany są automatycznie zapisywane.
- **Współpraca:** Zaproś innych użytkowników do współpracy, udostępniając link do projektu.
- **Eksport:** Eksportuj dokument do formatu PDF lub innego formatu, korzystając z opcji eksportu.

## 4. Implementacja

### 4.1. Wprowadzenie

W niniejszym dokumencie przedstawiono implementację algorytmu listy dwukierunkowej w języku C++. Opisano ciekawe fragmenty kodu oraz wyniki działania programu.

### 4.2. Implementacja algorytmu

Implementacja listy dwukierunkowej składa się z dwóch głównych części: struktury węzła (Node) oraz klasy listy dwukierunkowej (DubleLinkedList).

### 4.3. Struktura Node

Struktura Node reprezentuje pojedynczy element listy dwukierunkowej. Składa się z trzech pól: `data` (przechowującego wartość), `prev` (wskaźnika na poprzedni element) oraz `next` (wskaźnika na następny element).

```
1 struct Node {  
2     int data;  
3     Node* prev;  
4     Node* next;  
5  
6     Node(int value) : data(value), prev(nullptr), next(nullptr) {}  
7 };
```

**Listing 2.** Fragment kodu - struktura Node

### 4.4. Klasa DubleLinkedList

Klasa `DubleLinkedList` zarządza operacjami na liście dwukierunkowej. Obejmuje metody dodawania, usuwania oraz wyświetlania elementów.

#### 4.4.1. Dodawanie elementów

Metody `addRandomToHead` i `addRandomToBack` umożliwiają dodawanie losowych elementów na początek i koniec listy.

```
1 void addRandomToHead() {  
2     int value = rand() % 100; // losowanie w zakresie 0-99  
3     Node* newNode = new Node(value);
```

```
4     if (!head) {
5         head = back = newNode;
6     } else {
7         newNode->next = head;
8         head->prev = newNode;
9         head = newNode;
10    }
11 }
12
13 void addRandomToBack() {
14     int value = rand() % 100;
15     Node* newNode = new Node(value);
16     if (!back) {
17         head = back = newNode;
18     } else {
19         newNode->prev = back;
20         back->next = newNode;
21         back = newNode;
22     }
23 }
```

Listing 3. Fragment kodu - dodawanie elementów

#### 4.4.2. Usuwanie elementów

Metody `removeFromHead`, `removeFromBack` i `removeAt` umożliwiają usuwanie elementów z listy.

```
1 void removeFromHead() {
2     if (!head) return;
3     Node* temp = head;
4     head = head->next;
5     if (head) {
6         head->prev = nullptr;
7     } else {
8         back = nullptr;
9     }
10    delete temp;
11 }
12
13 void removeAt(int index) {
14     if (index == 0) {
15         removeFromHead();
16         return;
17     }
```

```

18     Node* current = head;
19     int pos = 0;
20     while (current && pos < index) {
21         current = current->next;
22         pos++;
23     }
24     if (!current) {
25         cout << "Indeks poza zakresem " << endl;
26         return;
27     }
28     if (current->prev) current->prev->next = current->next;
29     if (current->next) current->next->prev = current->prev;
30     if (current == back) back = current->prev;
31     delete current;
32 }

```

Listing 4. Fragment kodu - usuwanie elementów

## 4.5. Ciekawe fragmenty kodu

Ciekawe fragmenty kodu obejmują m.in. metodę `display`, która wyświetla wszystkie elementy listy, oraz `clear`, która czyści listę.

```

1 void display() {
2     Node* current = head;
3     while (current) {
4         cout << current->data << " ";
5         current = current->next;
6     }
7     cout << endl;
8 }
9
10 void clear() {
11     while (head) {
12         removeFromHead();
13     }
14 }

```

Listing 5. Fragment kodu - wyświetlanie i czyszczenie listy

## 4.6. Wyniki działania algorytmu

Po uruchomieniu programu użytkownik może zobaczyć wyniki działania algorytmu, takie jak:

- Lista z losowymi wartościami,
- Lista w odwrotnej kolejności,
- Lista po usunięciu elementów,
- Lista po wyczyszczeniu.

Przykład działania programu pokazuje, że algorytm poprawnie zarządza operacjami na liście dwukierunkowej i efektywnie wykonuje dodawanie, usuwanie oraz wyświetlanie elementów.

```
1 int main() {
2     DupleLinkedList list;
3
4     list.addRandomToHead();
5     list.addRandomToHead();
6     list.addRandomToBack();
7     list.insertRandomAt(1);
8
9     cout << "Lista z losowymi wartosciami: ";
10    list.display();
11
12    cout << "Lista w odwrotnej kolejnosci: ";
13    list.displayReverse();
14
15    list.removeFromHead();
16    list.removeFromBack();
17    list.removeAt(0);
18
19    cout << "Lista po usunieciu elementow: ";
20    list.display();
21
22    list.clear();
23    cout << "Lista po wyczyszczeniu: ";
24    list.display();
25
26    return 0;
27 }
```

**Listing 6.** Fragment kodu - przykład działania

## 5. Wnioski

### 5.1. Wnioski Końcowe

W ramach przeprowadzonego projektu zrealizowano implementację algorytmu listy dwukierunkowej w języku C++. Projekt ten dostarczył wielu cennych informacji na temat efektywności oraz zastosowań tego typu struktur danych w praktycznych aplikacjach.

- **Zrozumienie algorytmu:** Implementacja listy dwukierunkowej pozwoliła na lepsze zrozumienie jej struktury oraz sposobu działania. Dzięki zastosowaniu wskaźników do zarządzania elementami, możliwe jest szybkie dodawanie oraz usuwanie węzłów bez potrzeby przeszukiwania całej listy.
- **Efektywność operacji:** Analizowane operacje takie jak dodawanie, usuwanie oraz wyświetlanie elementów wykazały wysoką efektywność algorytmu. Odpowiednie zarządzanie pamięcią i wskaźnikami przyczyniło się do minimalizacji czasu wykonywania tych operacji.
- **Przejrzystość kodu:** Stosowanie zrozumiałych nazw zmiennych oraz szczegółowych komentarzy w kodzie pozwoliło na łatwiejsze zrozumienie logiki działania algorytmu. To podejście sprzyja również przyszłej rozbudowie oraz utrzymywaniu kodu.
- **Możliwości rozwoju:** Zrealizowany projekt otwiera możliwość dalszej rozbudowy, na przykład przez implementację dodatkowych funkcji, takich jak sortowanie czy wyszukiwanie. Istnieje również potencjał do połączenia listy dwukierunkowej z innymi strukturami danych w celu zwiększenia ich funkcjonalności.
- **Praktyczne zastosowania:** Lista dwukierunkowa znajduje szerokie zastosowanie w różnych dziedzinach, takich jak tworzenie baz danych, zarządzanie kolejkami, czy aplikacje wymagające dynamicznego zarządzania danymi. Zastosowania te podkreślają znaczenie tej struktury w realnych projektach.

**Podsumowanie:** Przeprowadzony projekt ukazuje, że lista dwukierunkowa jest nie tylko skutecznym narzędziem do zarządzania danymi, ale również doskonałym sposobem na naukę podstaw algorytmiki i programowania w języku C++. Uzyskane wyniki oraz wnioski stanowią solidną podstawę do dalszej pracy nad bardziej złożonymi projektami i algorytmami.



## Spis rysunków

2.1. Jednokierunkowa Lista . . . . .	5
2.2. Lista Algorytmy i Struktury Danych . . . . .	6
2.3. Jedno i dwukierunkowe Listy . . . . .	6

## **Spis tabel**

## Spis listingów

1.	Fragment kodu - dodawanie elementów . . . . .	6
2.	Fragment kodu - struktura Node . . . . .	12
3.	Fragment kodu - dodawanie elementów . . . . .	12
4.	Fragment kodu - usuwanie elementów . . . . .	13
5.	Fragment kodu - wyświetlanie i czyszczenie listy . . . . .	14
6.	Fragment kodu - przykład działania . . . . .	15