AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

...Algorytm listy dwukierunkowej z zastosowaniem GitHub...

Autor:
Jakub Marek Tokarczyk

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań

Pierwszym celem projektu jest wykonanie programu listy dwukierunkowej opartej na stercie. Działanie listy ma zostać zaimplementowane w klasie oraz ma zawierać metody odpowiadające za:

- Dodanie elementu na początku listy
- Dodanie elementu na końcu listy
- Dodanie elementu pod wskazany indeks
- Usunięcie elementu z początku listy
- Usunięcie elementu z końca listy
- Usunięcie elementu z pod wskazanego indeksu
- Wyświetlanie listy
- Wyświetalnie listy w odwrotnej kolejności
- Wyświetlanie następnego elementu
- Wyświetlanie poprzedniego elementu
- Czyszczenie całej listy

Działanie klasy i metody ma zostać przetestowane w funkcji main.
Drugim celem projektu jest zapoznanie się z GitHub i zainstalowanie wtyczki GitHub do wybranego środowiska i używać GitHub przy pisaniu programu Należy stworzyć na nim konto i zapisywać postępy nad projektem. Przy oddaniu projektu zaprezentować:

- Co najmniej 5 commit'ów
- Cofnięcie się o dwa commity.
- Usunięcie jednego commita
- Usunięcie pliku z katalogu projektu, aby się nie kompilował oraz ściągnięcie ostatniej poprawnej wersji z GitHub, aby projekt znów się kompilował.

2. Analiza problemu

Lista dwukierunkowa to struktura danych, która umożliwia poruszanie się w obu kierunkach: do przodu i do tyłu. Każdy element listy zawiera trzy składniki: dane (data), wskaźnik na poprzedni element (prev) oraz wskaźnik na następny element (next).

Rys. 2.1. Przykład

2.1. Gdzie używa się algorytmu listy dwukierunkowej?

Lista dwukierunkowa jest wykorzystywana w wielu zastosowaniach, gdzie wymagane jest szybkie i łatwe poruszanie się po elementach zarówno do przodu, jak i do tyłu np:

- Systemy zarządzania pamięcią
- Historia przeglądania w przeglądarkach internetowych
- Kolejki z priorytetem

2.2. Sposób działania algorytmu listy dwukierunkowej

Lista dwukierunkowa składa się z węzłów, z których każdy przechowuje dane oraz dwa wskaźniki:

- next - odnosi się do kolejnego elementu w liście.
- prev - odnosi się do poprzedniego elementu w liście.

Dodawanie elementu na początku listy. Podobnie jak w przypadku jednokierunkowej listy liniowej, zakładamy, że operacja dodawania nowego elementu do dwukierunkowej listy liniowej będzie wykonywana wyłącznie na niepustej liście. Dodatkowo, po pomyślnym wykonaniu tej operacji, wskaźnik listy powinien wskazywać na jej pierwszy element. Jeśli jednak nie uda się utworzyć nowego elementu, stan listy powinien

pozostać niezmieniony. Przy implementacji operacji dodawania nowego elementu do listy należy rozważyć trzy główne przypadki:

- element jest dodawany na początku listy i zostanie jej pierwszym elementem
- element jest dodawany wewnątrz listy
- element jest dodawany na końcu listy i zostanie jej ostatnim elementem

Każdy z tych przypadków zostanie zaimplementowany w osobnych funkcjach pomocniczych, które będą wywoływane w ramach jednej głównej funkcji odpowiedzialnej za dodawanie elementu do dwukierunkowej listy liniowej. Przedstawienie kodów źródłowych funkcji pomocniczych poprzedzi ich szczegółowy opis.

```
void add_at_back(struct list_node *last_node,
                                 struct list_node *new_node)
{
    last_node->next = new_node;
    new_node->previous = last_node;
}
```

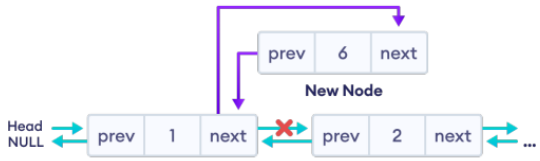
Rys. 2.2. Dodawanie na końcu

Funkcja addatback() przyjmuje dwa wskaźniki jako argumenty. Pierwszy wskaźnik odnosi się do bieżącego ostatniego elementu listy, a drugi do nowego elementu, który ma zostać dodany na końcu listy. W wierszu nr 4 adres nowego elementu zostaje zapisany w polu next aktualnego końcowego elementu, co powoduje, że nowy element staje się ostatnim w liście. Jednak na tym etapie nie jest jeszcze w pełni zintegrowany z listą. Dlatego w wierszu nr 5 adres elementu wskazywanego przez lastnode zostaje zapisany w polu previous nowego elementu. Po wykonaniu tych operacji nowy element staje się integralną częścią listy jako jej ostatni element.

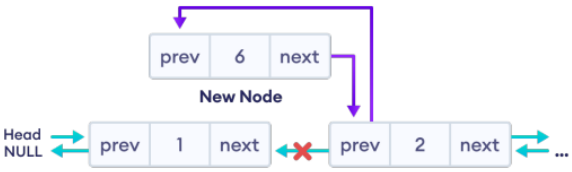
Operacja usuwania elementu z dwukierunkowej listy liniowej różni się nieco od analogicznej operacji w liście jednokierunkowej. Choć istnieją pewne podobieństwa, różnice są na tyle istotne, że implementacja tej operacji jest bardziej złożona. Podobnie jak w przypadku listy jednokierunkowej, zakładamy, że: jeśli lista była jednoelementowa, po usunięciu elementu powinna przestać istnieć; jeśli zawierała więcej elementów, powinna zostać pomniejszona o jeden element, zachowując spójną strukturę; jeśli natomiast element do usunięcia nie istnieje, stan listy powinien pozostać niezmieniony.

Możemy także wstawić element pod wskazany indeks łącząc ze sobą wskaźnik prev

nowego elemnentu z wskaźnikiem next poprzedzającego indeks.



Rys. 2.3. Łączenie z poprzednim elementem



Rys. 2.4. Łączenie z następnym elementem

3. Projektowanie

W projekcie będę korzystać z następujących narzędzi:

- Visual Studio 2022 - Środowisko programistyczne, które będzie wykorzystywane do pisania, kompilowania i debugowania kodu. Visual Studio 2022 zapewnia pełne wsparcie dla języka C++ oraz zaawansowane funkcje debugowania i zarządzania projektem.
- Język programowania - C++, Programowanie w języku C++ pozwoli na efektywną implementację algorytmów oraz struktur danych, takich jak lista dwukierunkowa. Język ten zapewnia dużą kontrolę nad pamięcią i wydajnością, co jest istotne w wielu projektach algorytmicznych i systemowych.
- Git: System kontroli wersji, który będzie wykorzystywany do zarządzania wersjami kodu, śledzenia zmian. Git umożliwia łatwe zarządzanie historią projektu i integrację z platformami takimi jak GitHub.

3.1. Sposób używania git

Git to system kontroli wersji, który pozwala na śledzenie i zarządzanie historią zmian w plikach projektu. Aby rozpocząć korzystanie z Gita, należy wykonać kilka podstawowych kroków, które obejmują konfigurację, tworzenie repozytorium oraz zarządzanie wersjami plików. Na początku trzeba zainstalować Git na swoim komputerze. Git jest dostępny dla systemów operacyjnych Windows, macOS i Linux, a instalację można pobrać ze strony oficjalnej. Po instalacji warto skonfigurować swoje dane użytkownika, aby Git wiedział, kto dokonał zmian w repozytorium. Po skonfigurowaniu Gita, można rozpocząć pracę nad repozytorium. Pierwszym krokiem jest utworzenie nowego repozytorium. Kiedy repozytorium jest gotowe, możemy zacząć pisanie kodu. Git umożliwia synchronizowanie lokalnych zmian z zewnętrznym repozytorium, co oznacza że możemy wysłać i pobrać zmiany kodu. Aby pobrać najnowsze zmiany z repozytorium, używamy komendy git pull a aby wysłać nasze zmiany na serwer, używamy git push. Dzięki temu systemowi każdy programista może swobodnie pracować nad projektem, a Git zapewnia kontrolę nad historią zmian, umożliwiając powrót do poprzednich wersji plików.

4. Implementacja

```
class DoublyLinkedList {
private:
  struct Node {
    int data;
    Node* prev;
    Node* next;
    Node(int val) : data(val), prev(nullptr), next(nullptr) {}
  };
```

Listing 1. Struktura węzła

Przedstawiony fragment kodu opisuje strukturę węzła listy dwukierunkowej w klasie DoublyLinkedList. Węzeł jest podstawowym elementem listy, który przechowuje dane oraz wskaźniki do poprzedniego i następnego węzła.

```
public:
  DoublyLinkedList() : head(nullptr), tail(nullptr) {}

  ~DoublyLinkedList() {
    Clear();
  }
    void AddAtBeg(int value) {
      Node* newNode = new Node(value);
      if (!head) {
        head = tail = newNode;
      }
      else {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
      }
    }
    void AddAtEnd(int value) {
      Node* newNode = new Node(value);
      if (!tail) {
        head = tail = newNode;
      }
      else {
        newNode->prev = tail;
        tail->next = newNode;
        tail = newNode;
      }
    }
```

Listing 2. Konstrukor dekonstruktor dodawanie elementu na początek i koniec listy

W przedstawionym fragmencie listingu znajduje się konstruktor, który inicjalizuje pustą listę, ustawiając wskaźniki head i tail na nullptr, oraz dekonstruktor, który usuwa wszystkie elementy listy, wywołując metodę Clear(), aby zwolnić pamięć. Metoda AddAtBeg(int value) pozwala na dodanie nowego elementu na początek listy. Z kolei metoda AddAtEnd(int value) umożliwia dodanie elementu na koniec listy.

```
void display() const {
 if (!head) {
  std::cout << "The list is empty.\n";
  return;
 }
 Node* current = head;
 while (current) {
  std::cout << current->data << " ";
  current = current->next;
 }
 std::cout << std::endl;
}
void DisplayReverse() const {
 if (!tail) {
  std::cout << "The list is empty.\n";
  return;
 }
 Node* current = tail;
 while (current) {
  std::cout << current->data << " ";
  current = current->prev;
 }
 std::cout << std::endl;
}
```

Listing 3. Wypisywanie listy

Metoda display() wypisuje zawartość listy od jej początku do końca, iterując przez kolejne węzły za pomocą wskaźnika next. Z kolei metoda DisplayReverse() wypisuje elementy w odwrotnej kolejności, zaczynając od końca listy i poruszając się wstecz za pomocą wskaźnika prev. Obie metody sprawdzają, czy lista jest pusta, i w takim przypadku wyświetlają odpowiedni komunikat.

```
void menu() {
  std::cout << "\nMenu:\n";
  std::cout << "1. Add item at the beginning of the list.\n";
  std::cout << "2. Add item at the end of the list.\n";
  std::cout << "3. Add item at a specified index.\n";
```

```
6   std::cout << "4. Delete item from the beginning of the list.\n";
7   std::cout << "5. Delete item from the end of the list.\n";
8   std::cout << "6. Delete item from a specified index.\n";
9   std::cout << "7. Display list\n";
10  std::cout << "8. Display reversed list.\n";
11  std::cout << "9. Clear list.\n";
12  std::cout << "0. Exit.\n";
13  std::cout << "Your choice: ";
14 }
```

Listing 4. Menu

Funkcja menu() wyświetla użytkownikowi listę dostępnych opcji do zarządzania listą dwukierunkową. Umożliwia m.in. dodawanie elementów na początku, końcu lub w określonym miejscu listy, usuwanie elementów z tych samych pozycji, wyświetlanie listy w kolejności normalnej lub odwrotnej, a także czyszczenie całej listy. Opcja 0 pozwala na zakończenie programu.

```
1     void Clear() {
2       while (head) {
3         RemoveFromBeg();
4       }
5     }
```

Listing 5. Wyczyszcze listy

Metoda Clear() usuwa wszystkie elementy z listy dwukierunkowej, wywołując w pętli metodę RemoveFromBeg() do momentu, aż lista stanie się pusta

5. Wnioski

Tworzenie listy dwukierunkowej było niezwykle ciekawym projektem, który udało mi się zrealizować przy wykorzystaniu znanych mi narzędzi języka C++. Dzięki temu zadaniu lepiej zrozumiałem, jak działa taka struktura danych oraz w jaki sposób można się po niej poruszać. Dodatkowo nauczyłem się korzystać z platformy GitHub, która oferuje bardzo przydatne narzędzie do kontroli wersji, ułatwiające zarządzanie i śledzenie zmian w kodzie.

Spis rysunków

Spis tabel

Spis listingów