AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

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

Autor:
Seweryn Homoncik

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań

1.1. Określenie celu pracy

Celem jest opracowanie i zaimplementowanie dwukierunkowej listy dynamicznej (czyli listy, która przechowuje elementy w sposób rozproszony na stercie) w języku C++ z użyciem klasy, która umożliwia operacje na elementach listy za pomocą wskaźników. Klasa, reprezentująca strukturę listy dwukierunkowej, zapewni odpowiednią funkcjonalność do efektywnego zarządzania elementami listy i manipulowania nimi. W wyniku pracy nad programem powstanie klasa, która będzie realizowała funkcjonalności typowe dla listy dwukierunkowej, a każda z metod będzie miała konkretne przeznaczenie:

- Dodawanie elementów:

  - Metoda dodająca element na początek listy powinna umieścić nowy węzeł przed istniejącym pierwszym węzłem i aktualizować wskaźniki początku listy.

  - Metoda dodająca element na koniec listy powinna umieścić nowy węzeł na końcu listy i odpowiednio zaktualizować wskaźniki.

  - Możliwość dodania elementu pod wskazany indeks pozwoli na elastyczne modyfikowanie listy, gdzie elementy będą umieszczane na dowolnej pozycji.

- Usuwanie elementów:

  - Metoda usuwająca element z początku listy będzie odpowiedzialna za usunięcie pierwszego węzła i aktualizację wskaźników.

  - Podobnie metoda usuwająca element z końca listy usunie ostatni węzeł.

  - Możliwość usunięcia elementu pod wskazanym indeksem zapewni elastyczność i pozwoli na precyzyjną manipulację listą.

- Wyświetlanie elementów:

  Metody umożliwiające wyświetlanie listy od początku do końca oraz w odwrotnej kolejności będą prezentować pełną zawartość listy w obu kierunkach, co ułatwi wizualizację danych. Wyświetlenie następnego lub poprzedniego elementu w stosunku do wybranego pozwoli użytkownikowi nawigować po liście, co jest przydatne np. do tworzenia interfejsów opartych na listach.

- Czyszczenie listy:

Metoda czyszcząca całą listę zwolni pamięć zajmowaną przez wszystkie elementy, aby zapobiec wyciekom pamięci.

Program powinien zostać przetestowany w funkcji main, aby sprawdzić, czy wszystkie metody działają poprawnie i zgodnie z założeniami. Oczekujemy, że program poprawnie obsłuży dodawanie, usuwanie i przeglądanie elementów w obu kierunkach oraz sprawnie zwolni pamięć po zakończeniu działania.

2. Analiza problemu

2.1. Zastosowanie algorytmu listy dwukierunkowej

Dwukierunkowe listy są szeroko stosowane w wielu aplikacjach i strukturach danych, w których wymagany jest dynamiczny dostęp do elementów i możliwość łatwego przemieszczania się między nimi w obu kierunkach. Algorytm dwukierunkowej listy dynamicznej (lub inaczej listy podwójnie powiązanej) jest wykorzystywany w następujących obszarach:

- Implementacje edytorów tekstu i arkuszy kalkulacyjnych: Podczas edycji długich dokumentów dwukierunkowa lista może przechowywać kolejne linie lub fragmenty tekstu, pozwalając na szybkie przechodzenie między nimi w obu kierunkach.

- Systemy operacyjne: Listy dwukierunkowe są używane do implementacji list procesów, zarządzania zadaniami, czy buforów w systemach operacyjnych, gdzie przechodzenie do poprzedniego lub następnego elementu jest konieczne.

- Struktury danych cache'owe: W przypadku algorytmów typu LRU (Least Recently Used) cache, listy dwukierunkowe pomagają w szybkim usuwaniu najrzadziej używanych elementów z pamięci podręcznej. Navigacja po historii: W przeglądarkach internetowych oraz w systemach nawigacyjnych lista dwukierunkowa jest idealna do implementacji nawigacji „do przodu" i „do tyłu".

2.2. Sposób działania programu/algorytmu

Algorytm listy dwukierunkowej składa się z węzłów, które przechowują wartość oraz wskaźniki do poprzedniego i następnego węzła. Dzięki temu można łatwo przemieszczać się po liście w obu kierunkach. Program dodaje nowe węzły na początku, na końcu lub w określonej pozycji w liście, aktualizując wskaźniki sąsiednich węzłów, aby poprawnie odwoływały się do nowo dodanego elementu. Algorytm usuwa wybrany węzeł (z początku, końca lub wskazanej pozycji), zmieniając wskaźniki poprzedniego i następnego węzła, aby zapełnić „lukę" po usuniętym elemencie. Program przechodzi przez każdy węzeł w liście, wyświetlając jego wartość, albo w kolejności od początku do końca, albo odwrotnie, wykorzystując wskaźniki w obu kierunkach. Algorytm usuwa każdy węzeł, zwalniając zajmowaną przez niego pamięć, a następnie ustawia wskaźniki początkowy i końcowy na wartość null, aby zasygnalizować, że lista jest

pusta.

2.3. Sposób wykorzystania algorytmu

- Inicjalizacja listy: Tworzymy obiekt klasy listy dwukierunkowej, który początkowo nie zawiera żadnych elementów (pusta lista). Operacje na liście:
- Dodajemy elementy, na przykład na początek, koniec, lub w konkretne miejsce w liście, w zależności od wymagań aplikacji.

  Wywołujemy operacje przeglądania listy od początku lub w odwrotnej kolejności, by przechodzić przez dane zgodnie z potrzebami aplikacji (np. wyświetlanie historii działań). Dla systemów czasu rzeczywistego (takich jak aplikacje przechowujące aktywną listę zadań) lista może być aktualizowana dynamicznie poprzez dodawanie lub usuwanie elementów, gdy zmienia się stan programu. Kiedy lista nie jest już potrzebna, wywoływana jest metoda czyszczenia, aby usunąć wszystkie elementy i zwolnić pamięć.

3. Projektowanie

3.1. Opis środowiska GitHub

GitHub[1] to platforma hostingowa, która umożliwia przechowywanie i zarządzanie kodem źródłowym projektów przy użyciu systemu kontroli wersji Git. Pozwala na współpracę nad projektami, śledzenie zmian w kodzie oraz synchronizowanie pracy. Dzięki GitHub, każdy członek zespołu może tworzyć oddzielne "gałęzie"(branch), na których wprowadza zmiany, a następnie łączy je z głównym projektem poprzez "pull requesty"po zaakceptowaniu przez innych.

GitHub jest kluczowym narzędziem w pracy zespołów open-source, gdzie kod projektów jest publicznie dostępny, a każdy użytkownik może zaproponować zmiany lub aktualizacje. Jest także platformą społecznościową, gdzie deweloperzy mogą tworzyć profile, śledzić innych programistów oraz przeglądać i uczyć się z publicznie dostępnych repozytoriów.

3.2. Język programowania - C++

C++ to jeden z najpopularniejszych i najbardziej wszechstronnych języków programowania, który jest używany w szerokim zakresie dziedzin, od systemów operacyjnych po aplikacje multimedialne, gry komputerowe, a także oprogramowanie naukowe. Język ten został zaprojektowany jako rozszerzenie języka C, wprowadzając paradygmaty programowania obiektowego, co czyni go bardziej elastycznym i umożliwia tworzenie bardziej złożonych programów i aplikacji.

- Podstawowe cechy języka C++: Wprowadza klasy, dziedziczenie, polimorfizm, abstrakcję oraz enkapsulację. Programowanie obiektowe pozwala na organizowanie kodu w struktury zwane obiektami, co ułatwia zarządzanie złożonymi aplikacjami. C++ jest językiem kompilowanym, co oznacza, że kod źródłowy jest przekształcany do postaci kodu maszynowego przed wykonaniem. Dzięki temu aplikacje napisane w C++ charakteryzują się dużą wydajnością, co czyni język idealnym do tworzenia aplikacji, gdzie szybkość działania jest krytyczna, jak np. w grach, symulacjach czy systemach wbudowanych. Język ten daje programistom pełną kontrolę nad pamięcią i zasobami systemowymi. Umożliwia ręczne zarządzanie pamięcią za pomocą operatorów new i delete, co pozwala na optymalizację zużycia zasobów, ale także wprowadza ryzyko błędów, takich jak wycieki pamięci.

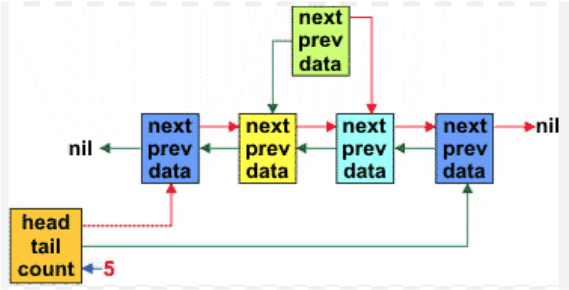
[1] https://github.com/

C++ wspiera różne style programowania, w tym programowanie proceduralne, obiektowe, a także funkcyjne, co sprawia, że jest bardzo elastycznym językiem. Programiści mogą wybrać najlepszy sposób rozwiązywania problemu, dostosowując paradygmat do specyfiki aplikacji. Oferuje bogaty zestaw bibliotek standardowych (Standard Template Library, STL), które zawierają gotowe implementacje struktur danych (np. wektory, listy, mapy) oraz algorytmy (np. sortowanie, wyszukiwanie). STL umożliwia programistom szybkie tworzenie wydajnych aplikacji bez konieczności pisania kodu od podstaw. Programy napisane w C++ mogą być uruchamiane na różnych systemach operacyjnych, takich jak Windows, Linux czy macOS, dzięki dostosowanym kompilatorom i narzędziom dostarczanym przez różne platformy. Jest używany w tworzeniu aplikacji na urządzenia mobilne, komputery stacjonarne, a także w systemach wbudowanych.

- Zalety C++:
  - Wydajność – Język ten pozwala na tworzenie bardzo wydajnych aplikacji, dzięki kompilacji do kodu maszynowego i bezpośredniemu dostępowi do pamięci.

  - Kontrola nad sprzętem – C++ umożliwia bezpośrednią manipulację pamięcią i rejestrami, co jest kluczowe w programowaniu systemów i aplikacji wbudowanych.

  - Wszechstronność – C++ jest stosowane w bardzo wielu dziedzinach: od gier komputerowych po oprogramowanie używane w przemyśle kosmicznym.



Rys. 3.1. Działanie listy dwukierunkowej[1]

4. Implementacja

Program implementuje dwukierunkową listę dynamiczną w języku C++, która umożliwia operacje na elementach listy, takie jak dodawanie, usuwanie, wyświetlanie i czyszczenie listy. Poniżej omówione są główne komponenty programu i działanie algorytmu listy dwukierunkowej:

4.1. Struktura Wezel

Wewnętrzna struktura Wezel definiuje pojedynczy węzeł listy dwukierunkowej:

- dane – przechowuje wartość elementu.
- poprzedni – wskaźnik do poprzedniego węzła.
- nastepny – wskaźnik do następnego węzła.
- Konstruktor – inicjuje węzeł z wartością wartosc i ustawia wskaźniki poprzedni i nastepny na nullptr.

4.2. Klasa ListaDwukierunkowa

Klasa ta przechowuje wskaźniki poczatek i koniec, które wskazują odpowiednio na pierwszy i ostatni element listy. Dzięki dwukierunkowym wskaźnikom możliwa jest iteracja listy w obu kierunkach.

Metody dodawania elementów:

- dodajNaPoczatek – Dodaje nowy węzeł na początku listy. Jeśli lista jest pusta, nowy węzeł staje się jedynym elementem (poczatek i koniec wskazują na ten sam węzeł). W przeciwnym razie aktualizuje wskaźniki tak, aby nowy węzeł był nowym początkiem.

  Wezel nowyWezel = new Wezel(wartosc);* - Tworzenie nowego węzła

  nowyWezel-¿nastepny = poczatek; - Nowy węzeł wskazuje na aktualny początek

  poczatek-¿poprzedni = nowyWezel; - Aktualny początek wskazuje na nowy węzeł jako poprzedni

  poczatek = nowyWezel; - Początek listy przesuwa się na nowy węzeł

```
void dodajNaPoczatek(int wartosc) {
        Wezel* nowyWezel = new Wezel(wartosc);
        if (!poczatek) {
            poczatek = koniec = nowyWezel;
        }
        else {
            nowyWezel->nastepny = poczatek;
            poczatek->poprzedni = nowyWezel;
            poczatek = nowyWezel;
        }
    }
```

Listing 1. Metoda dodająca element na początek listy

- dodajNaKoniec – Dodaje nowy węzeł na końcu listy. Działa podobnie jak poprzednia metoda, ale umieszcza element na końcu, aktualizując wskaźnik koniec.

  nowyWezel-¿poprzedni = koniec; - Nowy węzeł wskazuje na aktualny koniec

  koniec-¿nastepny = nowyWezel; - Aktualny koniec wskazuje na nowy węzeł jako następny

  koniec = nowyWezel; - Koniec listy przesuwa się na nowy węzeł

```
void dodajNaKoniec(int wartosc) {
        Wezel* nowyWezel = new Wezel(wartosc);
        if (!koniec) {
            poczatek = koniec = nowyWezel;
        }
        else {
            nowyWezel->poprzedni = koniec;
            koniec->nastepny = nowyWezel;
            koniec = nowyWezel;
        }
    }
```

Listing 2. Metoda dodająca element na koniec listy

- dodajNaIndeksie – Dodaje nowy węzeł na określonej pozycji. Jeśli indeks jest zerowy, dodaje element na początku. Jeśli indeks jest poza zakresem, dodaje na końcu listy. W innym przypadku umieszcza węzeł na wskazanej pozycji, dostosowując wskaźniki poprzedni i nastepny.

  if (indeks == 0) - Jeśli indeks to 0, dodaj na początek

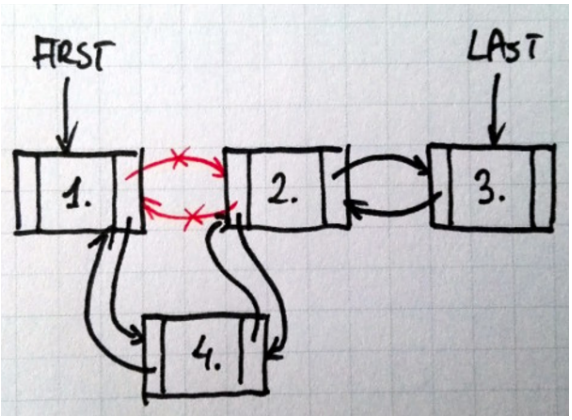
  if (!obecny) - Jeśli indeks jest poza końcem listy, dodaj na koniec

nowyWezel-¿nastepny = obecny-¿nastepny; - Ustawia wskaźnik następny dla nowego węzła

nowyWezel-¿poprzedni = obecny; - Ustawia wskaźnik poprzedni dla nowego węzła

```
void dodajNaIndeksie(int indeks, int wartosc) {
    if (indeks == 0) {
        dodajNaPoczatek(wartosc);
        return;
    }
    Wezel* obecny = poczatek;
    for (int i = 0; obecny && i < indeks - 1; i++) {
        obecny = obecny->nastepny;
    }
    if (!obecny) {
        dodajNaKoniec(wartosc);
    }
    else {
        Wezel* nowyWezel = new Wezel(wartosc);
        nowyWezel->nastepny = obecny->nastepny;
        nowyWezel->poprzedni = obecny;
        if (obecny->nastepny) {
            obecny->nastepny->poprzedni = nowyWezel;
        }
        obecny->nastepny = nowyWezel;
        if (obecny == koniec) {
            koniec = nowyWezel;
        }
    }
}
```

Listing 3. Metoda dodająca element na indeksie



Rys. 4.1. Dodawanie elementu do listy dwukierunkowej[2]

Metody usuwania elementów:

- usunZPoczatku – Usuwa węzeł z początku listy, przesuwając poczatek na następny element. Jeśli lista staje się pusta, ustawia koniec na nullptr.

  if (!poczatek) return; - Jeśli lista jest pusta, nic nie robi

  Wezel temp = poczatek;* - Zapamiętuje wskaźnik na początek

  poczatek = poczatek-¿nastepny; - Przesuwa początek na następny element

  delete temp; - Usuwa pierwotny początek

```
void usunZPoczatku() {
        if (!poczatek) return;
        Wezel* temp = poczatek;
        poczatek = poczatek->nastepny;
        if (poczatek) {
            poczatek->poprzedni = nullptr;
        }
        else {
            koniec = nullptr;
        }
        delete temp;
    }
```

Listing 4. Metoda usuwająca element z początku listy

- usunZKonca – Usuwa węzeł z końca listy, przesuwając koniec na poprzedni element.

  koniec = koniec-¿poprzedni; - Przesuwa koniec na poprzedni element

  delete temp; - Usuwa pierwotny koniec

```
void usunZKonca() {
        if (!koniec) return;
        Wezel* temp = koniec;
        koniec = koniec->poprzedni;
        if (koniec) {
            koniec->nastepny = nullptr;
        }
        else {
            poczatek = nullptr;
        }
        delete temp;
    }
```

Listing 5. Metoda usuwająca element z końca listy

- usunNaIndeksie – Usuwa węzeł na określonej pozycji. Jeśli indeks wskazuje na pierwszy element, usuwany jest początek. Jeśli indeks jest poza zakresem, operacja jest ignorowana.

  if (obecny-¿poprzedni) obecny-¿poprzedni-¿nastepny = obecny-¿nastepny; - Aktualizuje wskaźnik następny

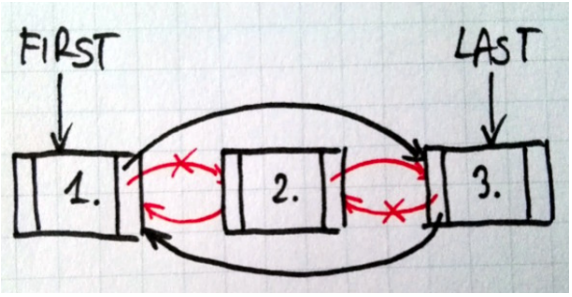
  if (obecny-¿nastepny) obecny-¿nastepny-¿poprzedni = obecny-¿poprzedni; - Aktualizuje wskaźnik poprzedni

  if (obecny == poczatek) poczatek = obecny-¿nastepny; - Jeśli usuwany jest początek, przesuwa początek

  if (obecny == koniec) koniec = obecny-¿poprzedni; - Jeśli usuwany jest koniec, przesuwa koniec

```
void usunNaIndeksie(int indeks) {
    if (indeks == 0) {
        usunZPoczatku();
        return;
    }
    Wezel* obecny = poczatek;
    for (int i = 0; obecny && i < indeks; i++) {
        obecny = obecny->nastepny;
    }
    if (!obecny) return;
    if (obecny->poprzedni) obecny->poprzedni->nastepny = obecny
   ->nastepny;
    if (obecny->nastepny) obecny->nastepny->poprzedni = obecny
   ->poprzedni;
    if (obecny == poczatek) poczatek = obecny->nastepny;
    if (obecny == koniec) koniec = obecny->poprzedni;
    delete obecny;
}
```

Listing 6. Metoda usuwająca element na indeksie



Rys. 4.2. Usuwanie elementu z listy dwukierunkowej[2]

Wyświetlanie listy:

- wyswietl – Wyświetla elementy listy od początku do końca.

  while (obecny) - Przechodzi przez listę *std::cout « obecny-¿dane « "";* - Wyświetla wartość

```
void wyswietl() const {
        Wezel* obecny = poczatek;
        while (obecny) {
            std::cout << obecny->dane << " ";
            obecny = obecny->nastepny;
        }
        std::cout << std::endl;
    }
```

Listing 7. Metoda wyświetlająca listę

- wyswietlOdKonca – Wyświetla elementy listy w odwrotnej kolejności, zaczynając od końca.

  while (obecny) - Przechodzi przez listę od końca

```
void wyswietl() const {
        Wezel* obecny = poczatek;
        while (obecny) {
            std::cout << obecny->dane << " ";
            obecny = obecny->nastepny;
        }
        std::cout << std::endl;
    }

```

Listing 8. Metoda wyświetlające listę od końca

Czyszczenie listy:

- wyczysc – Usuwa wszystkie elementy listy, wywołując usunZPoczatku aż lista będzie pusta.

```
void wyczysc() {
    while (poczatek) {
        usunZPoczatku();
    }
}
```

Listing 9. Metoda usuwająca zawartość listy

5. Wnioski

Implementacja algorytmu listy dwukierunkowej w tym projekcie pozwoliła na praktyczne zapoznanie się z dynamiczną strukturą danych oraz mechanizmami zarządzania pamięcią w C++. Lista dwukierunkowa umożliwia wydajne dodawanie i usuwanie elementów na początku i końcu listy dzięki wskaźnikom poczatek i koniec. Dzięki dwukierunkowym wskaźnikom, lista dwukierunkowa umożliwia zarówno przechodzenie od początku do końca listy, jak i w przeciwnym kierunku. Jest to szczególnie przydatne w aplikacjach, gdzie istotne jest odwrotne przetwarzanie danych lub szybki dostęp do obu końców listy.

Projekt przedstawia prawidłowe zarządzanie pamięcią w dynamicznych strukturach danych. Każdy węzeł był tworzony dynamicznie, co wymagało odpowiedniego usuwania go z pamięci. Implementacja metod wyczysc oraz destruktora zapewniła brak wycieków pamięci.

Bibliografia

[1] *Źródło schematu listy dwukierunkowej.* URL: https://eduinf.waw.pl/inf/alg/001_search/0087.php (term. wiz. 14. 11. 2024).

[2] *Źródło schematów dodawania i usuwania elementów z listy dwukierunkowej.* URL: https://www.samouczekprogramisty.pl/struktury-danych-lista-wiazana/ (term. wiz. 20. 11. 2024).

Spis rysunków

Spis listingów