AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

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

Autor:
Tomasz Wąchała

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2023

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 programem do kontroli wersji GitHub. Należy stworzyć na nim konto i zapisywać postępy nad projektem. Przy oddaniu projektu zaprezentować:

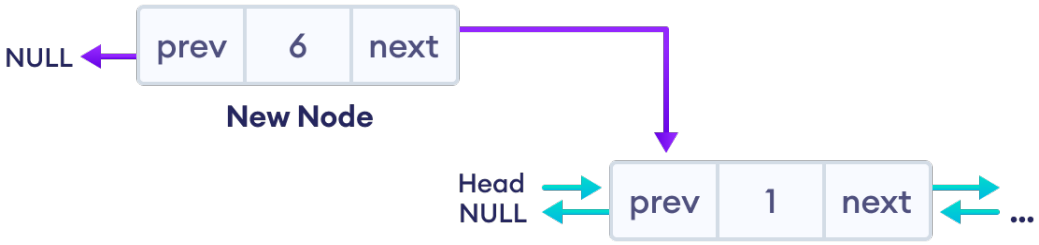
- Co najmniej 5 commit'ów
- Co najmniej jedno cofnięcie się o dwa commit'y
- Usunięcie jednego commit'a

2. Analiza problemu

Lista dwukierunkowa jest to struktura danych po której możemy poruszać się w dwóch kierunkach, do przodu i do tyłu. Każdy element zawiera wskaźnik na poprzedni (prev) lub następny (next) element w liście oraz dane (data) Rys. 2.1 (s. 4).

Rys. 2.1. Element listy.[1]

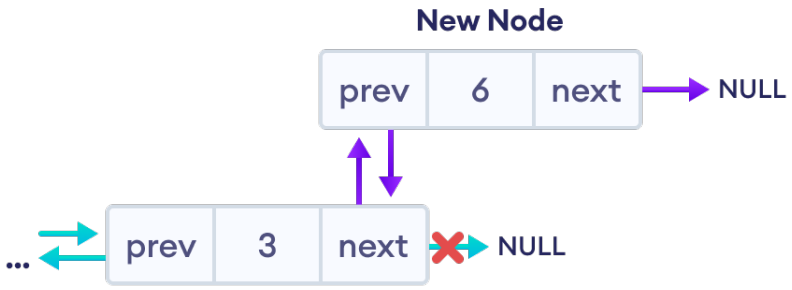
Na liście dwukierunkowej można przeprowadzać wiele operacji, takich jak dodawanie elementu na początek listy. Polega ono na połączeniu ze sobą wskaźnika next nowo stworzonego elementu ze wskaźnikiem prev elementu z początku listy (head) w taki sposób aby na siebie wskazywały Rys. 2.2 (s. 4).



Rys. 2.2. Dodawanie elementu na początek.[1]

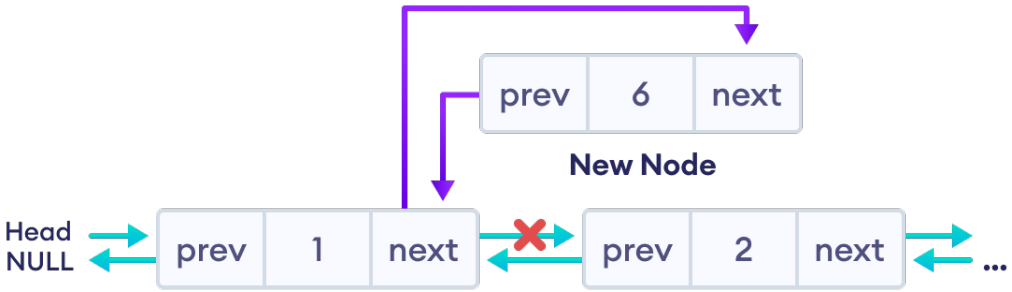
[1]Infografika ze strony https://www.programiz.com/dsa/doubly-linked-list[1].

Kolejną operacją jest dodanie elementu na koniec. Przebiega ona podobnie jak z dodawaniem na początek z różnicą, że łączymy ostatni element (tail) z nowo stworzonym Rys. 2.3 (s. 5).



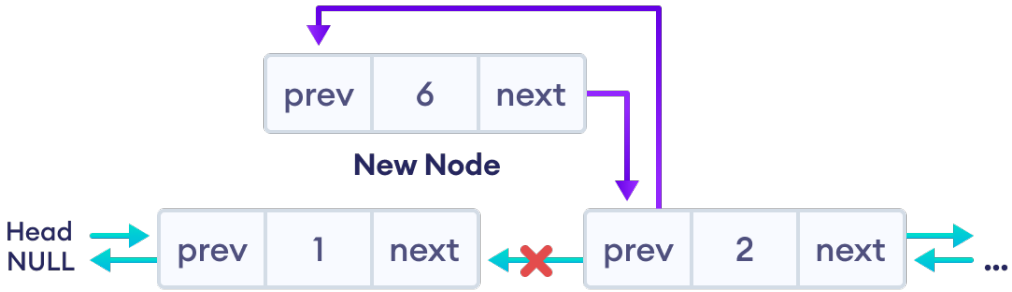
Rys. 2.3. Dodawanie elementu na koniec.[1]

Możemy także wstawić element pod wskazany indeks łącząc ze sobą wskaźnik prev nowego elemnentu z wskaźnikiem next poprzedzającego indeks Rys. 2.4 (s. 5) oraz wskaźnik next z wskaźnikiem prev następnego po wskazanym indeksie Rys. 2.5 (s. 6).



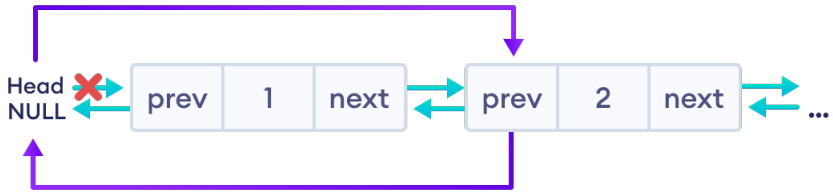
Rys. 2.4. Łączenie z poprzedzającym elementem.[1]

[1]Infografika ze strony https://www.programiz.com/dsa/doubly-linked-list[1].

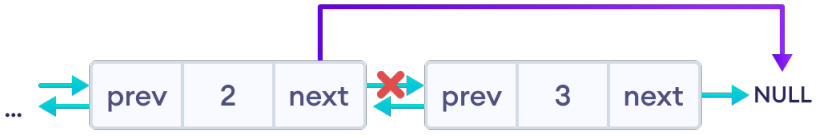


Rys. 2.5. Łączenie z następnym elementem.[1]

W analogiczny sposób przeprowadza się operację usuwania elementów z listy dwukierunkowej. Pierwszy i ostatni element usuwa się poprzez wpisywanie wartości NULL, żeby na nic nie wskazywały, do wskaźników elementów head i tail 2.6 i 2.7 (s. 6). Inaczej wygląda usuwanie wybranego elementu, gdzie łączy się ze sobą wskaźniki poprzedzającego i następującego elementu aby ten, którego chcemy się pozbyć nie należał już do listy 2.8 (s. 7).



Rys. 2.6. Usuwanie pierwszego elementu.[1]



Rys. 2.7. Usuwanie ostatniego elementu.[1]

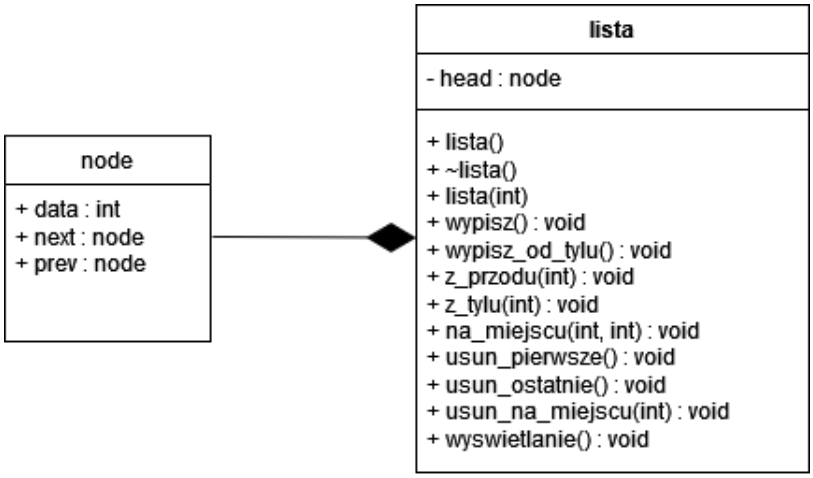
[1]Infografika ze strony https://www.programiz.com/dsa/doubly-linked-list[1].

Rys. 2.8. Usuwanie wybranego elementu.[1]

[1]Infografika ze strony https://www.programiz.com/dsa/doubly-linked-list[1].

3. Projektowanie

Projekt wykonany będzie w programie Virtual Studio Code z użyciem języka C++. Kompilacja będzie przeprowadzana za pomocą dodatku do VS Code o nazwie Code Runner wykorzystującego wcześniej zainstalowany kompilator MinGW. Kolejne wersje projektu będą zapisywane na platformie GitHub.



Rys. 3.1. Diagram klasy.

Klasa będzie zawierać strukturę node odpowiadającą poszczególnym elementom listy. Działanie wszystkich metod klasy będzie przetestowane w funkcji main.

4. Implementacja

```
class lista
{
private:
    struct node {
        int value;
        node *next;
        node *prev;
    } *head;
```

Listing 1. Struktura elementów listy.

Kod 1 (s. 9) przedstawia strukturę elementu listy, znajduje się ona w klasie lista w części prywatnej. Na końcu, po strukturze, tworzy się pierwszy element listy, o nazwie head.

```
public:
    lista(){
        head = new node();
        head->value = 0;
        head->next = NULL;
        head->prev = NULL;
    };
    ~lista(){
        node* temp = head;
        while(temp != NULL){
            node* temp2 = temp;
            temp = temp->next;
            delete temp2;
        }
        head = NULL;
        cout << "Usuwanie listy" << endl;
    };
    lista(int value){
        head = new node();
        head->value = value;
        head->next = NULL;
        head->prev = NULL;
    };
```

Listing 2. Konstruktory i destruktory

Kod 2 (s. 9) przedstawia konstruktory i destruktory. Konsturktor parametrowy i bezparametrowy tworzą pierwszy element listy o nazwie head za pomocą struktury node. Przypisywane mu są wartość value w zależności czy była podana podczas

tworzenia listy oraz wartości NULL do wskaźników prev i next gdyż nie ma innych elementów listy niż początkowy. Desturktor idzie odpowiednio po każdym kolejnym elemencie listy od początku i z pomocą zmiennych temp i temp2 usuwa wszystkie jej elementy.

```
1    void wypisz(){
2        node* temp = head;
3        while(temp != NULL){
4            cout << temp->value << " ";
5            temp = temp->next;
6        }
7        cout << endl;
8    };
9    void wypisz_od_tylu(){
10       node* temp = head;
11       while(temp->next != NULL){
12           temp = temp->next;
13       }
14       while(temp != NULL){
15           cout << temp->value << " ";
16           temp = temp->prev;
17       }
18       cout << endl;
19   };
```

Listing 3. Wypisywanie listy

Kod 3 (s. 10) przedstawia metody służące do wypisywania listy od początku i od końca. Wypisywanie od początku przebiega od pierwszego elementu (head), wypisuje go, przechodzi do następnego i o ile nie jest pusty wypisuje ponownie i powtarza proces aż do przejścia całej listy. Wypisywanie od końca działa na podobnej zasadzie z tym, że najpierw idzie się do ostatniego elemenut listy (tail) i od niego wypisuje się każdy poprzedni element aż natrafi na koniec.

```
1    void z_przodu(int value){
2        node* temp = new node();
3        temp->value = value;
4        temp->next = head;
5        temp->prev = NULL;
6        head->prev = temp;
7        head = temp;
8    };
9    void z_tylu(int value){
10       node* temp = head;
11       while(temp->next != NULL){
```

```
            temp = temp->next;
        }
        node* nowy = new node();
        nowy->value = value;
        nowy->next = NULL;
        nowy->prev = temp;
        temp->next = nowy;
    };
    void na_miejscu(int value, int index){
        node* temp = head;
        for(int i = 0; i < index-1; i++){
            temp = temp->next;
        }
        node* nowy = new node();
        nowy->value = value;
        nowy->next = temp->next;
        nowy->prev = temp;
        temp->next = nowy;
    };
```

Listing 4. Dodawanie elementów listy

Kod 4 (s. 10) przedstawia dodawanie elementu na początek, koniec lub wskazane miejsce do listy. W odpowiednie wskaźniki prev i next wpisuje się adresy elementów które mają być ze sobą połączone. Usuwanie działa podobnie ale zamiast wpisywać we wskaźniki adresy wpisuje się NULL aby nie łączyły się z innymi elementami.

```
    void wyswietlanie(){
        cout << "Aktualny element: " << head->value << endl;
        int wybor;
        cin >> wybor;

        switch(wybor){
            case 0:
                break;
            case 1:
                if(head->prev != NULL){
                    head = head->prev;
                }
                wyswietlanie();
                break;
            case 2:
                if(head->next != NULL){
                    head = head->next;
                }
                wyswietlanie();
```

```
20                  break;
21              default:
22                  cout << "Nie ma takiej opcji" << endl;
23                  wyswietlanie();
24                  break;
25          }
26      }
```

Listing 5. Dodawanie elementów listy

Kod 5 (s. 11) przedstawia metodę, której zadaniem jest wyświetalnie następnego bądź poprzedniego elementu. Wypisuje ona pierwszy element z listy i za pomocą instrukcji switch pozwala wbyierać pomiędzy przejściem na następny, poprzedni element oraz wyjście wybierając 0.

5. Wnioski

Tworzenie listy dwukierunkowej było bradzo ciekawym projektem, który byłem w stanie za pomocą znanych mi już narzędzi w języku C++ stworzyć. Dzięki niemu jestem w stanie sobie wyobrazić jak taka lista wygląda oraz w jaki sposób po niej się porusza. Nauczyłem się także używać bardzo przydatnego narzędzia do kotroli wersji znajdującego się na platformie GitHub[2].

[2]GitHub https://github.com[2].

Bibliografia

[1] *Strona internetowa Programiz*. URL: https://www.programiz.com/dsa/doubly-linked-list (term. wiz. 13. 10. 2023).

[2] *Strona internetowa GitHub*. URL: https://github.com (term. wiz. 15. 10. 2023).

Spis rysunków

Spis tabel

Spis listingów