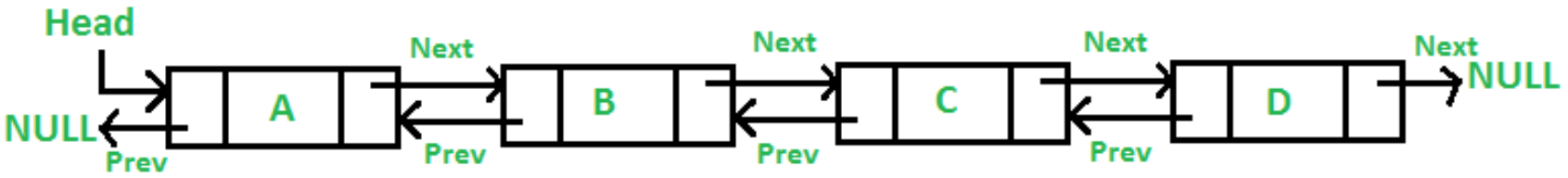
W. 4.3
Listy dwukierunkowe
(doubly linked list)

Idea

- W przypadku listy jednokierunkowej mogliśmy się łatwo przesuwać w przód listy przez wskaźnik do następnego elementu.
- Problemem jest przesuwanie się wstecz. Należy ponownie wrócić na początek i ponownie przejść listę. W najgorszym wypadku, dla ostatniego elementu o indeksie **n** należy ponownie przejść przez **n-1** elementów, żeby otrzymać element poprzedni.
- Problem przejścia do poprzedniego elementu listy były klasy O(1) gdybyśmy mieli strukturę wskaźników wskazujących na poprzedni element listy.
- Taką podwójną strukturę wskaźników next, previous wykorzystuje się w liście dwukierunkowej, którą będziemy omawiać w tej części.
- Zapamiętaj:
  - Lista jest abstrakcyjną strukturą danych która umożliwia sekwencyjny dostęp do danych.
  - Lista jednokierunkowa lub dwukierunkowa jest tylko sposobem implementacji (wykonaniem) tej abstrakcyjnej struktury danych!

Idea



Implementacja w języku C++

Węzeł

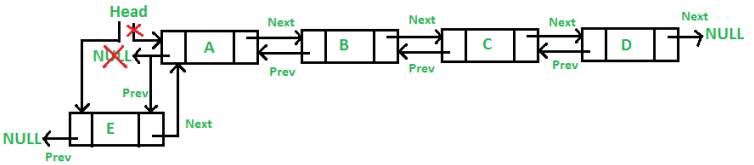
- struct **Node** {
- int **data**;
- struct Node* **next**;
- struct Node* **prev**;
- };

- **data** -przechowywana dana typu int; Typ tego pola możemy parametryzować przy użyciu szablonów.
- **Next** – wskaźnik na następny węzeł;
- **Prev** – wskaźnik na poprzedni węzeł;

Wstawianie

Wstawianie na początek - push

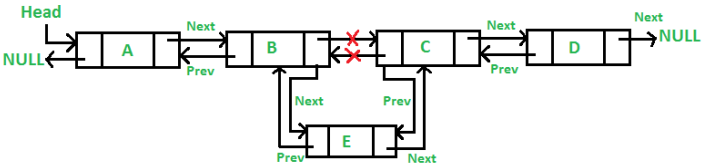
- void **push**(struct Node** head_ref, int new_data)
- { struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
-
- new_node->data = new_data;
- new_node->next = (*head_ref);
- new_node->prev = NULL;
-
- if ((*head_ref) != NULL)
- (*head_ref)->prev = new_node;
-
- (*head_ref) = new_node; }



- Kroki:
  - Alokujemy węzeł – malloc;
  - Ustawiamy atrybuty węzła:
    - Data, next, prev
  - Ustawiamy głowę listy na nowy element;

Wstawianie za dany element - **insertAfter**

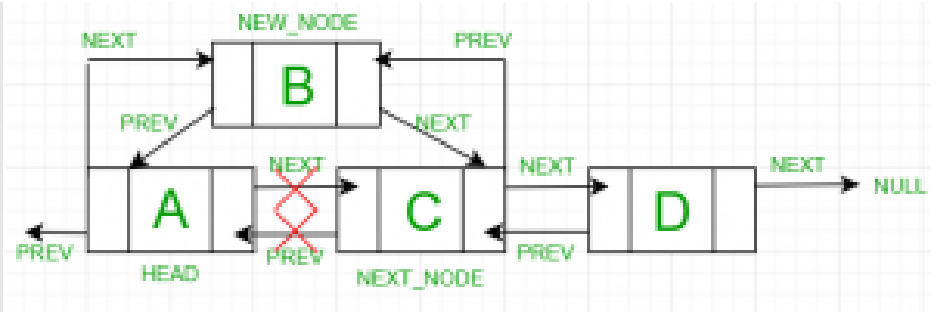
- void **insertAfter**(struct Node* prev_node, int new_data)
- {
- /*1. check if the given prev_node is NULL */
- if (prev_node == NULL) { printf("the given previous node cannot be NULL"); return; }
-
- /* 2. allocate new node */
- struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
-
- /* 3. put in the data */
- new_node->data = new_data;
-
- /* 4. Make next of new node as next of prev_node */
- new_node->next = prev_node->next;
-
- /* 5. Make the next of prev_node as new_node */
- prev_node->next = new_node;
-
- /* 6. Make prev_node as previous of new_node */
- new_node->prev = prev_node;
-
- /* 7. Change previous of new_node's next node */
- if (new_node->next != NULL) new_node->next->prev = new_node; }



- Procedura wstawiania nie różni się od tej dla listy jednokierunkowej. Należy jedynie dodatkowo ustawić atrybut **prev.**

Wstaw przed – insertBefore

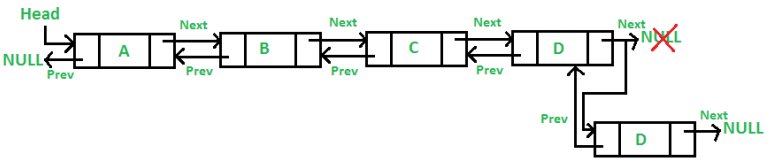
```
void insertBefore(struct Node** head_ref, struct Node* next_node, int new_data)
{      /*1. check if the given next_node is NULL */
    if (next_node == NULL) {
        printf("the given next node cannot be NULL");
        return;  }
    /* 2. allocate new node */
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));

    /* 3. put in the data */
    new_node->data = new_data;

    /* 4. Make prev of new node as prev of next_node */
    new_node->prev = next_node->prev;

    /* 5. Make the prev of next_node as new_node */
    next_node->prev = new_node;

    /* 6. Make next_node as next of new_node */
    new_node->next = next_node;

    /* 7. Change next of new_node's previous node */
    if (new_node->prev != NULL)
        new_node->prev->next = new_node;
    /* 8. If the prev of new_node is NULL, it will be
       the new head node */
    else
        (*head_ref) = new_node; }
```



- W tym przypadku mamy przewagę listy dwukierunkowej nad jednokierunkową, gdyż mamy bezpośredni wskaźnik do poprzedniego elementu w węźle listy.
- Zastanów się, jak zaimplementować taka funkcję dla listy jednokierunkowej. Jaka jest złożoność takiego podejścia.

Dodaj na koniec - append

- void **append**(Node** head_ref, int new_data)
- { /* 1. allocate node */
- Node* new_node = new Node();
- Node* last = *head_ref; /* used in step 5*/
-
- /* 2. put in the data */
- new_node->data = new_data;
-
- /* 3. This new node is going to be the last node, so make next of it as NULL*/
- new_node->next = NULL;
-
- /* 4. If the Linked List is empty, then make the new
- node as head */
- if (*head_ref == NULL) { new_node->prev = NULL; *head_ref = new_node; return; }
-
- /* 5. Else traverse till the last node */
- while (last->next != NULL) last = last->next;
-
- /* 6. Change the next of last node */
- last->next = new_node;
-
- /* 7. Make last node as previous of new node */
- new_node->prev = last;
-
- return; }



- Jest to analogiczna funkcja do tej z listy jednokierunkowej.

Zadania

- Zaimplementuj:
  - Usuwanie pierwszego elementu z listy.
  - Usuwanie ostatniego elementu z listy.
  - Usuwanie dowolnego elementu z listy.
  - Iteracja po liście w przód i w tył.
- Wskazówka:
  - Wykorzystaj analogiczne funkcje dla listy jednokierunkowej pamiętając o właściwym ustawieniu wskaźnika **prev**.

Literatura dodatkowa

- https://www.geeksforgeeks.org/doubly-linked-list/
- Rozdział 5 - Algorytmy. Struktury danych i techniki programowania.
- Rozdziały 10 – T. Cormen, ‘Wprowadzenie do algorytmów’, PWN
- Rozdział 9 - Data Structures and Algorithms using Python.

Koniec