Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

**Лабораторна робота №** 2

по дисципліні «Теорія алгоритмів»

на тему: Дослідження списків

|  |  |
| --- | --- |
| Виконав:  студент групи ІА-з91  Москаленко В.В.  Дата здачі \_\_\_\_\_\_\_\_\_\_\_\_  Захищено з балом \_\_\_\_\_\_\_\_\_\_\_ | Перевірив:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Київ 2020

1. Постановка проблеми.

У цій лабораторній роботі нам необхідно створити три класи, для кожного типу структур даних, а також спільний інтерфейс. Методи добавлення, вилучення, заміщення і пошуку розмістити всередині цих класів. Щоби отримати достовірні дані результату спочатку ми заповнимо структури даних великою кількістю випадкових елементів, переконавшись, що елемент, використовуваний для пошуку не буде міститися серед випадково згенерованих. Для кожної операції підрахувати середній час виконання, порівняти для різних алгоритмів. Для даної задачі є однозначний розв’язок, який переважно реалізовується за рахунок мови С на більш низькому рівні. Так як нам потрібна висока оптимізація для коректних операції вставки видалення у великих об’ємах даних.

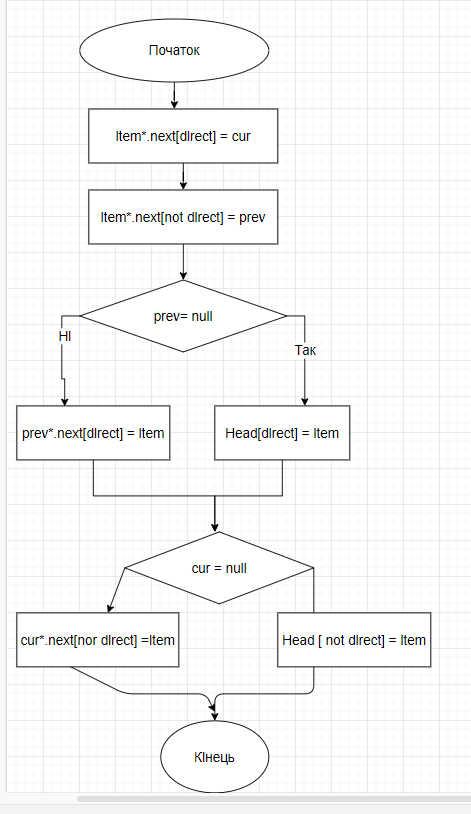
1. Побудова моделі.

Для усіх трьох структур даних ми будемо використовувати стандартні структури даних всередині цих класів. В структуру даних буде входити поле Data та один або два вказівники, в залежності це буде однобічно чи двобічно зв’язаний список, також буде входити базовий конструктор даних, тобто буде комбінація кількох структур. Для кожної операції ми будемо робити окремий метод, який потім будемо викликати в основній функції.

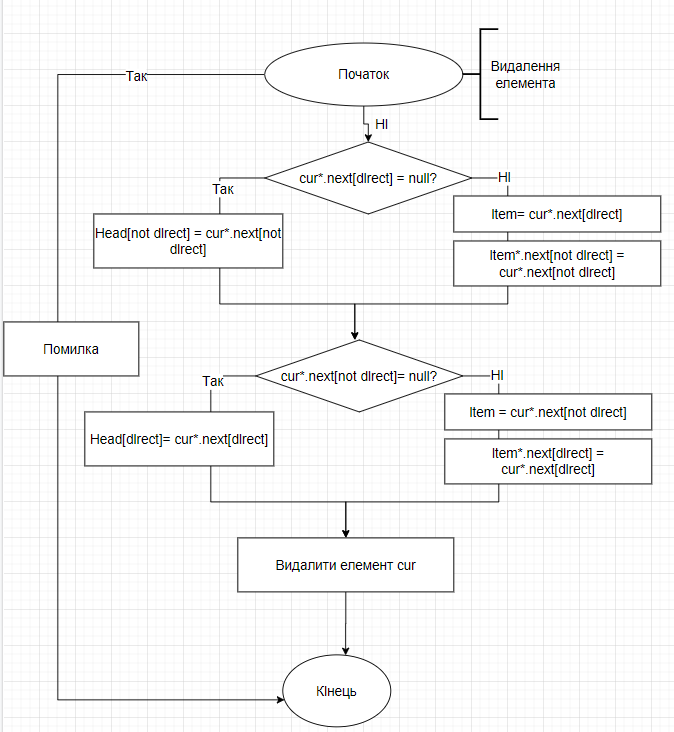
1. Розроблення алгоритму

Алгоритм для даної задачі полягає у використані структури даних з даними вказівниками базовим конструктором і примінення її до конкретих алгоритмів. Розглянемо алгоритм на прикладі однобічно зв’язаного списка. Для прикладу візьме функції вставки. У випадку звичайної вставки ми будемо вставляти дані в кінець списку, перевіряючи чи кінець списку є null, і відповідно будемо створювати наступний елемент за допомогою прирівнення вказівника на кінець списка, до того значення яке ми хочемо вставити. У випадку якщо у списку є 0 елементів, то наш елемент стає першим. Для другого варіанту ми будемо використовувати функцію вставки на початок списку. Відповідно, якщо список є пустим то ми вставляємо наш елемент на перше місце, якщо список є заповненим ми ставимо вказівник нашого елементу на перший елемент нашого списку, таким чином стаючи у голові списку. У випадку функції вставки в попередньо визначене місце ми літеруємо список до конкретного елементу і ставимо вказівник попереднього елементу на наш новий елемент а вказівник нашого елементу на той елемент на який вказував попередній елемент. Так само реалізуємо всі наступні методи для всіх трьох структур даних.

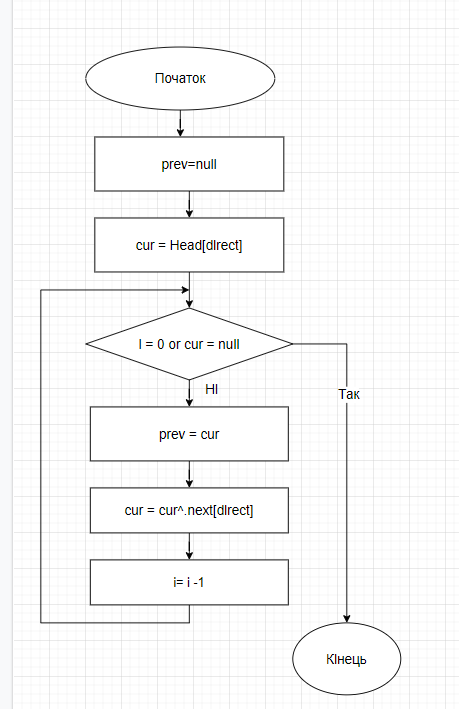
**Алгоритм пошуку в двобічно зв’язаному списку**



**Алгоритм видалення в двобічно зв’язаному списку**

****

**Алгоритм заміщення в двобічно зв’язаному списку**



1. Перевірка правильності алгоритму.

Пересвідчимось, що наш алгоритм працює для всіх коректних вхідних даних, на прикладі операцій вставки в однобічно зв’язаний список. Ми передбачили варіанти коли список є пустим і коли список є заповненим. Функція вставки не має циклів, тому алгоритм має низьку імовірність за циклювання. Існуючі дані не псуються. За рахунок того, що а початку ми перевіряємо чи список є пустим або наповненим і в кінці повертаємо список. У випадку, якщо нам потрібно викати вставку у попередньо визначене місце, ми зберігаємо вказівник на наступний елемент в тимчасовій змінній. Таким чином ми не втрачаємо другу частину списка.

5. Реалізація алгоритму.

import java.io.\*;

public class LinkedList {

Node head;

static class Node {

int data;

Node next;

Node(int d)

{

data = d;

next = null;

}

}

public static LinkedList insert(LinkedList list, int data)

{

Node new\_node = new Node(data);

new\_node.next = null;

if (list.head == null) {

list.head = new\_node;

}

else {

Node last = list.head;

while (last.next != null) {

last = last.next;

}

last.next = new\_node;

}

return list;

}

public static void printList(LinkedList list)

{

Node currNode = list.head;

System.out.print("\nLinkedList: ");

while (currNode != null) {

System.out.print(currNode.data + " ");

currNode = currNode.next;

}

System.out.println("\n");

}

public static LinkedList deleteByKey(LinkedList list, int key)

{

Node currNode = list.head, prev = null;

if (currNode != null && currNode.data == key) {

list.head = currNode.next;

System.out.println(key + " found and deleted");

return list;

}

while (currNode != null && currNode.data != key) {

prev = currNode;

currNode = currNode.next;

}

if (currNode != null) {

prev.next = currNode.next;

System.out.println(key + " found and deleted");

}

if (currNode == null) {

System.out.println(key + " not found");

}

return list;

}

public static LinkedList deleteAtPosition(LinkedList list, int index)

{

Node currNode = list.head, prev = null;

if (index == 0 && currNode != null) {

list.head = currNode.next;

System.out.println(index + " position element deleted");

return list;

}

int counter = 0;

while (currNode != null) {

if (counter == index) {

prev.next = currNode.next;

System.out.println(index + " position element deleted");

break;

}

else {

prev = currNode;

currNode = currNode.next;

counter++;

}

}

if (currNode == null) {

System.out.println(index + " position element not found");

}

return list;

}

public static void main(String[] args)

{

LinkedList list = new LinkedList();

list = insert(list, 1);

list = insert(list, 2);

list = insert(list, 3);

list = insert(list, 4);

list = insert(list, 5);

list = insert(list, 6);

list = insert(list, 7);

list = insert(list, 8);

printList(list);

deleteByKey(list, 1);

printList(list);

deleteByKey(list, 4);

printList(list);

deleteByKey(list, 10);

printList(list);

deleteAtPosition(list, 0);

printList(list);

deleteAtPosition(list, 2);

printList(list);

deleteAtPosition(list, 10);

printList(list);

}

}

1. Перевірка програми.

Даний програму ми тестували на різних вхідних даних, як у вигляді чисел так і вигляді строк. Перевіряли операції вставки, видалення, заміщення. І наступної перевірки даної структури даних порівнюючи паралельно заміри часу.

1. Вивід програми

