Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Алгоритми та структури даних

Лабораторна робота №7

**«Зв’язані списки»**

Виконала:

студентка групи ІО-64

Бровченко А. В.

Перевірив:

Саверченко В. Г.

Київ

2016 р.

**Теоретичні відомості**

Зв'язаний список в [програмуванні](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) — одна з найважливіших [структур даних](https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B8_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85), в якій елементи лінійно впорядковані, але порядок визначається не номерами елементів, а [вказівниками](https://uk.wikipedia.org/wiki/%D0%92%D0%BA%D0%B0%D0%B7%D1%96%D0%B2%D0%BD%D0%B8%D0%BA_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), які входять в склад елементів списку та вказують на наступний за даним елемент (в однозв'язаних або однобічно зв'язаних списках) або на наступний та попередній елементи (в двозв'язаних або двобічно зв'язаних списках). Список має «голову» — перший елемент та «хвіст» — останній елемент.

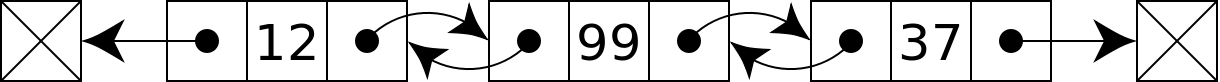
Зв'язані списки мають серію переваг порівняно з [масивами](https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D0%B8%D0%B2_(%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)). Зокрема, в них набагато ефективніше (незалежно від кількості елементів) виконуються процедури додавання та вилучення елементів. Натомість, масиви набагато кращі в операціях, які потребують безпосереднього доступу до кожного елементу, що у випадку зі зв'язаними списками неможливо та потребує послідовного перебору усіх елементів, які передують даному.

**Однобічно зв’язаний список**

https://upload.wikimedia.org/wikipedia/commons/3/37/Singly_linked_list.png

В однобічно зв'язаному списку, який є найпростішим різновидом зв'язаних списків, кожний елемент складається з двох полів: data або даних, та вказівника next на наступний елемент. Якщо вказівник не вказує на інший елемент (інакше: next = NULL), то вважається, що даний елемент — останній в списку.

**Двобічно зв’язаний список**



В двобічно зв'язаному списку елемент складається з трьох полів — вказівника на попередній елемент prev, поля даних data та вказівника next на наступний елемент. Якщо prev=NULL, то в елемента немає попередника (тобто він є «головою» списку), якщо next=NULL, то в нього немає наступника («хвіст» списка).

**Кільцевий список**

В кільцевому списку перший та останній елемент зв'язані. Тобто, поле prev голови списка вказує на хвіст списка, а поле next хвоста списка вказує на голову списка.

**Код програми**

**class** Node:  
  
 **def** \_\_init\_\_(self, data, prev, next):  
 self.data = data  
 self.prev = prev  
 self.next = next  
  
  
**class** DoubleLinkedList:  
  
 **def** \_\_init\_\_(self):  
 self.head = **None** self.tail = **None  
  
 def** add\_node(self, data):  
 new\_node = Node(data, **None**, **None**)  
  
 **if** self.head **is None**:  
 self.head = self.tail = new\_node  
 **else**:  
 new\_node.prev = self.tail  
 new\_node.next = **None** self.tail.next = new\_node  
 self.tail = new\_node  
  
 **def** delete\_node(self, num):  
 count = 0  
 current\_node = self.head  
  
 **while True**:  
 **if** count == num:  
 **if** current\_node.prev **is not None**:  
 current\_node.prev.next = current\_node.next  
 current\_node.next.prev = current\_node.prev  
 **else**:  
 self.head = current\_node.next  
 current\_node.next.prev = **None  
 break** count += 1  
 current\_node = current\_node.next  
  
 **def** show(self):  
 **if** self.head **is None**:  
 print(**None**)  
 **else**:  
 current\_node = self.head  
 **while** current\_node **is not None**:  
 **if** current\_node.next **is None**:  
 print(current\_node.data, end=**" "**)  
 **break  
 else**:  
 print(current\_node.data, end=**" <---> "**)  
 current\_node = current\_node.next  
  
 **def** show\_head(self):  
 print(self.head)  
  
 **def** len\_of(self):  
 current\_node = self.head  
  
 **if** current\_node **is None**:  
 count = 0  
 **else**:  
 count = 0  
 **while** current\_node **is not None**:  
 current\_node = current\_node.next  
 count += 1  
  
 **return** count  
  
  
**while True**:  
 n = 0  
 option = int(input(**"\n----------------------------------------\n"  
 "Оберіть операцію:\n"  
 "1 - створити двозв'язний список\n"  
 "2 - додати новий елемент до списку\n"  
 "3 - видалити елемент зі списку\n"  
 "4 - показати список\n"  
 "0 - завершити роботу\n"  
 "----------------------------------------\n"  
 "Ваш вибір: "**))  
 print(**'\n'**)  
  
 **if** option == 1:  
 l = DoubleLinkedList()  
 print(**'Список створено.'**)  
  
 **elif** option == 2:  
 el = input(**'Введіть елемент, який треба додати -> '**)  
 l.add\_node(el)  
  
 **elif** option == 3:  
 el = int(input(**'Введіть номер елемента, який треба видалити -> '**))  
 l.delete\_node(el)  
 print(**'Елемент додано в кінець списку.'**)  
  
 **elif** option == 4:  
 l.show()  
  
 **elif** option == 0:  
 **raise** SystemExit

**Висновок:**

Двобічне зв'язування потребує більше пам'яті на елемент та більших обчислювальних затрат на виконання елементарних операцій. Але такими списками легше маніпулювати, тому що вони дозволяють проходження по списку в обох напрямах, що є необхідною умовою функціонування деяких [алгоритмів](https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC).