Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2**

Линейные структуры данных

Вариант № 7

Преподаватель \_\_\_\_\_\_\_\_\_\_ И. В. Матковский

подпись, дата

Студент КИ20-08Б 032049025 \_\_\_\_\_\_\_\_\_\_ А. Н. Головань

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2021

1. **Цель работы**

Научиться работать с различными видами линейных структур данных.

1. **Порядок выполнения работы**

Разработать для решения поставленной задачи алгоритм; реализовать полученный алгоритм с использованием линейных структур данных заданных типов. Одна из заданных структур должна быть реализована самостоятельно, без использования готовых библиотек; вторая структура может быть, как реализована самостоятельно, так и взята из STL.

1. **Задание**

Удалить из числовой последовательности все отрицательные элементы.

1. **Код программы**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112 | import os  from time import time  from random import randint  os.system("cls")  class Node:    # Function to initialize the node object  def \_\_init\_\_(self, data):  self.data = data # Assign data  self.nextnode = None # Initialize next as null  self.backnode = None # Initialize next as null  def push(self, data):  while self.nextnode != None:  self = self.nextnode  self.nextnode = Node(data)  self.nextnode.nextnode = None  self.nextnode.backnode = self  def delete(self):  if self.backnode == None:  self = self.nextnode  self.backnode = None  return self    if self.nextnode == None:  self.backnode.nextnode = None  self = None  return    self.backnode.nextnode = self.nextnode  self.nextnode.backnode = self.backnode  def print\_nodes(self):  print(self.data,end = " ")  while self.nextnode != None:  self = self.nextnode  print(self.data,end = " ")  def print\_rev\_nodes(self):  global begin  global end  print(self.data)  while self.backnode != None:  self = self.backnode  print(self.data)    def list\_del\_neg(a):  begin = a  while a.nextnode != None:  a = a.nextnode    if (a.data<0):  new\_a = a.delete()  if new\_a != None:  a = new\_a  if begin.data <0:  begin = begin.nextnode    return begin  def massive\_del\_neg(b):  a = b.copy()  it = len(a)-1  while it>=0:  if a[it]<0:  del a[it]  it -= 1      return a      count = 20  print("count - > {}".format(count))  massive = [] #объявлем list  for i in range(count):  massive.append(randint(-9,9)) # добавляем в list  a = Node(massive[0]) # Объявляем список  for i in range(1,len(massive)):  a.push(massive[i])  print(massive)  start\_time = time()  massive = massive\_del\_neg(massive)  end\_time = time()  print("massive delete for {} sec".format(end\_time - start\_time))  print(massive)  print("\n")  a.print\_nodes() #выводим список  start\_time = time()  a = list\_del\_neg(a)  end\_time = time()  print("\nlist delete for {} sec".format(end\_time - start\_time))  a.print\_nodes() #выводим список |

1. **Теоретические оценки сложности алгоритмов**

Данная программа использует две разные формы хранения данных – массив и двусвязный список.

1. В случае с массивом, временная и пространственная сложность будет составлять т.к. Ситуация с удалением абсолютно аналогична ситуации с добавлением. В лучшем случае мы удаляем последний элемент массива; тогда его достаточно просто очистить, не трогая остальную часть.
2. Для двусвязного списка временная и пространственная сложность будет составлять т.к. нет необходимости сдвигать элементы массива, поддерживая наложенные на их положение ограничения, достаточно лишь удалить элемент и исправить нужные связи.
3. **Экспериментальные оценки сложности алгоритмов**

Таблица 1 - зависимость времени от объема входных данных

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Количество  элементов | 10 | 100 | 1000 | 10000 | 100000 |
| Массив, с | 0 | 0 | 0 | 0 | 0 |
| Двусвязный список, с | 0 | 0 | 0 | 0.007 | 0.0137 |