**Дисциплина «Алгоритмы решения прикладных задач»**

**Рабочая тетрадь 6.**

**Структуры данных (часть 1)**

|  |  |
| --- | --- |
| **Теоретический материал** | |
| Структура данных — это способ организации информации для более эффективного использования. В программировании структурой обычно называют набор данных, связанных определённым образом. Например, массив — это структура. Со структурой можно работать: добавлять данные, извлекать их и обрабатывать, например изменять, анализировать, сортировать. Для каждой структуры данных — свои алгоритмы. Работа программиста — правильно выбирать уже написанные готовые либо писать свои.  Главное свойство структур данных в том, что у любой единицы данных должно быть чёткое место, по которому её можно найти. Как определяется это место и как происходит поиск, зависит от конкретной структуры.  Характеристики структур данных следующие:  ● Данные в памяти представлены определённым образом, который однозначно позволяет определить структуру.  ● Чаще всего внутрь структуры можно добавить элемент или извлечь оттуда. Это свойство не постоянное — бывают структуры, которые нельзя изменять после создания.  ● Существуют алгоритмы, которые позволяют взаимодействовать с этой структурой.  При этом данных необязательно должно быть много. Массив из одного элемента — уже структура данных.  **1. Массив (Array)**  Одна из самых простых структур данных, которая встречается чаще всего. Именно на массивах основаны многие другие структуры данных: списки, стеки, очереди.  Для простоты восприятия можно считать, что массив — это таблица. Каждый его элемент имеет индекс — «адрес», по которому этот элемент можно извлечь. В большинстве языков программирования индексы начинаются с нуля. То есть первый элемент массива имеет индекс не [1], а [0]. Данные в массиве можно просматривать, сортировать и изменять с помощью специальных операций.  *Массивы бывают двух видов:*  **● *Одномерные.*** У каждого элемента только один индекс. Можно представить это как строку с данными, где одного номера достаточно, чтобы чётко определить положение каждой переменной.  ***● Многомерные.*** У каждого элемента два или больше индексов. По сути, это комбинация из нескольких одномерных массивов, то есть вложенная структура.    *Как применяют массивы:*  ● В качестве блоков для более сложных структур данных. Массивы предусмотрены в синтаксисе большинства языков программирования, и на их основе удобно строить другие структуры.  ● Для хранения несложных данных небольших объёмов.  ● Для сортировки данных.  **2. Динамический массив (Dynamic array)**  В классическом массиве размер задан заранее — мы точно знаем, сколько в нём индексов. А динамический массив — это тот, у которого размер может изменяться. При его создании задаётся максимальная величина и количество заполненных элементов. При добавлении новых элементов они сначала заполняются до максимальной величины, а при превышении сразу создаётся новый массив, с большей максимальной величиной.  Элементы в динамический массив можно добавлять без ограничений и куда угодно. Однако, если добавлять их в середину, остальные придётся сдвигать, что занимает много времени. Поэтому лучше всего динамический массив работает при добавлении элементов в конце.  *Как применяют динамические массивы:*  ● В качестве блоков для структур данных.  ● Для хранения неопределённого количества элементов.  **3. Связный список (Linked list)**  Ещё одна базовая структура данных, которую, как и массивы, используют для реализации других структур. Связный список — это группа из узлов. В каждом узле содержатся:  ● Данные.  ● Указатель или ссылка на следующий узел.  ● В некоторых списках — ещё и ссылка на предыдущий узел.  В итоге получается список, у которого есть чёткая последовательность элементов. При этом сами элементы более разрозненны, чем в массиве, поскольку хранятся отдельно. Быстро перемещаться между элементами списка помогают указатели.    *Как применяют связные списки:*  ● Для построения более сложных структур данных.  ● Для реализации файловых систем.  ● Для формирования хэш-таблиц.  ● Для выделения памяти в динамических структурах данных.  **4. Стек (Stack)**  Эта структура данных позволяет добавлять и удалять элементы только из начала. Она работает по принципу LIFO — Last In, First Out (англ. «последним пришёл — первым ушёл»). Последний добавленный в стек элемент должен будет покинуть его раньше остальных.    *Как применяют стеки:*  ● Для реализации рекурсии.  ● Для вычислений постфиксных значений.  ● Для временного хранения данных, например истории запросов или изменений.  **5. Очередь (Queue)**  Этот вид структуры представляет собой ряд данных, как и стек. Но в отличие от него она работает по принципу FIFO — First In, First Out (англ. «первым пришёл — первым ушёл»). Данные добавляют в конец, а извлекают из начала.    Бывают неклассические, двусторонние очереди (деки). В них можно добавлять элементы и извлекать их из начала и конца структуры. Элементы посередине недоступны.  **Как применяют очереди:**  ● Для реализации очередей, например на доступ к определённому ресурсу.  ● Для управления потоками в многопоточных средах.  ● Для генерации значений.  ● Для создания буферов. | |
| **Пример 1** | |
| ***Задача:*** | |
|  | Написать на языке C++ программу, в которой создается и распечатывается связный список из произвольного количества элементов (большего 2). |
| ***Решение:*** | |
|  |  |
| ***Ответ:*** | |
|  |  |
| **Задание 1** | |
| ***Задача:*** | |
|  | Создать массив натуральных чисел произвольного размера *n*, [*a*1, ..., *an*.] Определить и вывести на экран количество элементов массива:   |  | | --- | | а) являющихся нечетными числами; | | б) являющихся квадратами четных чисел; | | в) кратных 3 и не кратных 5; | | г) удовлетворяющих условию *;* | | д) удовлетворяющих условию *;* | | е) имеющих четные порядковые номера и являющихся нечетными числами. | |
| ***Решение:*** | |
|  | import math as m    def factorial(n):  solve = 0  for i in range(1, n):  solve \*= i;  return solve;    a= []  for i in range(1,100):  a.append(i)  count = 0  for i in a:  if i % 2 == 0:  count += 1  print(count)  count = 0  for i in a:  if m.sqrt(i) % 2 == 0:  count += 1  print(count)  count = 0  for i in range(len(a)):  if (m.pow(2,i)) < a[i] < factorial(i):  count += 1  print(count)  count = 0  for i in range(len(a)):  try:  if a[i] < ((a[i-1] + a[i+1]) / 2):  count += 1  except IndexError:  pass  print(count)  count = 0  for i in range(len(a)):  if ((i + 1) % 2 == 0) and (a[i] % 2 != 0):  count += 1  print(count) |
| ***Ответ:*** | |
|  |  |
| **Задание 2** | |
| ***Задача:*** | |
|  | Создать массив натуральных чисел произвольного размера *n*, [*a*1, ..., *an*.]. Найти количество и сумму тех элементов массива, которые делятся на 5 и не делятся на 7. |
| ***Решение:*** | |
|  | summ = 0  count = 0  a = [int(x) for x in range(1,100)]  for i in a:  if (i % 5 == 0) and (i % 7 != 0):  summ+=i  count += 1  print(f'Count: {count}, summa: {summ}') |
| ***Ответ:*** | |
|  |  |
| **Задание 3** | |
| ***Задача:*** | |
|  | Создать массив целых чисел произвольного размера *n*, [*a*1, ..., *an*.]. Имеются ли в массиве:  а) два идущих подряд нулевых члена;  б) три идущих подряд нулевых члена? |
| ***Решение:*** | |
|  | def check(a):  nul = a[0]  flaga = 0  flagb = 0  for i in range(len(a)):  try:  if (a[i] == a[i+1]) and (a[i] == nul):  print("a: YES")  flaga+= 1  break  except IndexError:  pass  for i in range(len(a)):  try:  if (a[i] + a[i+1] + a[i+2]) / 3 == nul:  print("b:Yes")  flagb += 1  break  except IndexError:  pass  if not(flaga):  print("a: Not")  if not(flagb):  print("b: Not")  return;  a = [5,1,5,5,2,0,8]  check(a) |
| ***Ответ:*** | |
|  |  |
| **Задание 4** | |
| ***Задача:*** | |
|  | Создать связанный список произвольного размера, элементы которого содержат целые числа. Посчитать сумму всех целых чисел в связанном списке. |
| ***Решение:*** | |
|  | class Node:  def \_\_init\_\_(self, data):  self.next = None  self.data = data    def append\_to\_linked\_list(self, val):  end = Node(val) # создание нового узла  n = self # ставим первый элемент  while n.next:  n = n.next # пока ну None  n.next = end    def print\_linked\_list(self):  n = self # Начать с первого узла  while n:  print(n.data)  n = n.next  def summ(self):  n = self  count = 0  while n:  count += n.data  n = n.next  print(count)    if \_\_name\_\_ == "\_\_main\_\_":  linked\_list = Node(1)  linked\_list.append\_to\_linked\_list(2)  linked\_list.append\_to\_linked\_list(3)  linked\_list.append\_to\_linked\_list(4)  linked\_list.summ()  linked\_list.print\_linked\_list() |
| ***Ответ:*** | |
|  |  |
| **Задание 5\*** | |
| ***Задача:*** | |
|  | Создать массив действительных чисел. Выяснить, является ли он упорядоченным по убыванию. |
| ***Решение:*** | |
|  | def is\_descending(arr):  # Проверка, является ли массив упорядоченным по убыванию  return all(arr[i] >= arr[i + 1] for i in range(len(arr) - 1))    # Создание массива действительных чисел  my\_array = [5.0, 4.0, 3.0, 2.0, 1.0]    # Проверка и вывод результата  if is\_descending(my\_array):  print("Массив упорядочен по убыванию.")  else:  print("Массив не упорядочен по убыванию.") |
| ***Ответ:*** | |
|  |  |
| **Задание 6\*** | |
| ***Задача:*** | |
|  | Для связного списка реализовать следующие операции в виде функций:   * Операция, проверяющая список на пустоту. * Три операции добавления объекта в список (в начало, конец или внутрь после любого (n-го) элемента списка); * Операция, возвращающая количество элементов в списке; * Операция доступа к списку, состоящему из всех элементов исходного списка, кроме первого. |
| ***Решение:*** | |
|  | class Node:  def \_\_init\_\_(self, data=None):  self.data = data  self.next = None      class LinkedList:  def \_\_init\_\_(self):  self.head = None    def is\_empty(self):  return self.head is None    def add\_to\_start(self, data):  new\_node = Node(data)  new\_node.next = self.head  self.head = new\_node    def add\_to\_end(self, data):  new\_node = Node(data)  if not self.head:  self.head = new\_node  return  current = self.head  while current.next:  current = current.next  current.next = new\_node    def add\_after\_nth(self, n, data):  new\_node = Node(data)  current = self.head  for \_ in range(n - 1):  if current is None:  print("Error: Index out of range")  return  current = current.next  new\_node.next = current.next  current.next = new\_node    def count\_elements(self):  count = 0  current = self.head  while current:  count += 1  current = current.next  return count    def get\_sublist\_without\_first(self):  if not self.head or not self.head.next:  return None  new\_head = self.head.next  current = new\_head  while current.next:  current = current.next  current.next = None  return new\_head      # Пример использования  linked\_list = LinkedList()    print("Пусто:", linked\_list.is\_empty())    linked\_list.add\_to\_end(10)  linked\_list.add\_to\_end(20)  linked\_list.add\_to\_start(5)    print("Количесвто элементов:", linked\_list.count\_elements())    linked\_list.add\_after\_nth(1, 15)    print("Количество элементов после вставки:", linked\_list.count\_elements())    sublist\_head = linked\_list.get\_sublist\_without\_first()  print("Все элементы кроме первого:")  current = sublist\_head  while current:  print(current.data, end=" ")  current = current.next |
| ***Ответ:*** | |
| **Задание 7\*** | |
| ***Задача:*** | |
|  | Для связных списков L1 и L2 (произвольного размера) описать функции, которые:  а) проверяет на равенство списки L1 и L2:  б) определяет, входят ли все элементы списка L1 в список L2;  в) проверяет, есть ли в списке L1 хотя бы два одинаковых элемента;  г) переносит в конец непустого списка L1 его первый элемент;  д) переносит в начало непустого списка L1 его последний элемент;  е) добавляет в конец списка L1 все элементы списка L2;  ж) переворачивает список L1, т.е. изменяет ссылки в этом списке так, чтобы его элементы оказались расположенными в обратном порядке;  з) оставляет в списке *L* только первые вхождения одинаковых элементов. |
| ***Решение:*** | |
|  |  |
| ***Ответ:*** | |
|  |  |
| **Задание 8\*** | |
| ***Задача:*** | |
|  | Создать список из заданного количества элементов. Выполнить циклический сдвиг этого списка на N элементов вправо или влево. |
| ***Решение:*** | |
|  | def create\_list(num\_elements):  # Создание списка из заданного количества элементов  return list(range(1, num\_elements + 1))    def cyclic\_shift(lst, shift\_count):  # Циклический сдвиг списка на shift\_count элементов вправо  shift\_count %= len(lst)  return lst[-shift\_count:] + lst[:-shift\_count]    # Пример использования  num\_elements = 6  my\_list = create\_list(num\_elements)    print("Исходный список:", my\_list)    # Выполнение циклического сдвига вправо на 2 элемента  shifted\_list\_right = cyclic\_shift(my\_list, 2)  print("Циклический сдвиг вправо на 2 элемента:", shifted\_list\_right)    # Выполнение циклического сдвига влево на 3 элемента  shifted\_list\_left = cyclic\_shift(my\_list, -3)  print("Циклический сдвиг влево на 3 элемента:", shifted\_list\_left) |
| ***Ответ:*** | |
|  |  |
| **Задание 9\*** | |
| ***Задача:*** | |
|  | Используя стек (например, std::stack, доступный в #include <stack>) напечатать содержимое текстового файла, выписывая буквы каждой его строки в обратном порядке. |
| ***Решение:*** | |
|  | #include <iostream>  #include <fstream>  #include <stack>  #include <string>    int main() {  std::ifstream inputFile("example.txt"); // Заменить "example.txt" на путь к текстовому файлу    if (!inputFile.is\_open()) {  std::cerr << "Unable to open the file." << std::endl;  return 1;  }    std::string line;  std::stack<char> charStack;    while (std::getline(inputFile, line)) {  // Заполняем стек буквами из строки в обратном порядке  for (char c : line) {  charStack.push(c);  }    // Выводим буквы из стека (в обратном порядке)  while (!charStack.empty()) {  std::cout << charStack.top();  charStack.pop();  }    std::cout << std::endl;  }    inputFile.close();    return 0;  } |
| ***Ответ:*** | |
| **Задание 10\*** | |
| ***Задача:*** | |
|  | Сформировать файл из натуральных чисел и с помощью очереди (например, std::queue, доступный в #include <queue>) за один просмотр файла напечатать элементы файла в следующем порядке: сначала все однозначные числа, затем двузначные, сохраняя исходный порядок чисел в каждой из этих групп. |
| ***Решение:*** | |
|  | #include <iostream>  #include <fstream>  #include <queue>  #include <string>    int main() {  std::ifstream inputFile("numbers.txt"); // Заменить "numbers.txt" на путь к файлу с числами    if (!inputFile.is\_open()) {  std::cerr << "Unable to open the file." << std::endl;  return 1;  }    std::queue<int> singleDigitQueue;  std::queue<int> doubleDigitQueue;    int number;  while (inputFile >> number) {  if (number >= 10) {  // Двузначные числа помещаем в соответствующую очередь  doubleDigitQueue.push(number);  } else {  // Однозначные числа помещаем в другую очередь  singleDigitQueue.push(number);  }  }    inputFile.close();    // Выводим однозначные числа в порядке их появления в файле  while (!singleDigitQueue.empty()) {  std::cout << singleDigitQueue.front() << " ";  singleDigitQueue.pop();  }    // Выводим двузначные числа в порядке их появления в файле  while (!doubleDigitQueue.empty()) {  std::cout << doubleDigitQueue.front() << " ";  doubleDigitQueue.pop();  }    std::cout << std::endl;    return 0;  } |
| ***Ответ:*** | |
|  |  |