**Дисциплина «Алгоритмы решения прикладных задач»**

**Рабочая тетрадь 1.2**

**Бинарный поиск**

|  |  |  |
| --- | --- | --- |
| **Теоретический материал** | | |
| Двоичный (бинарный) поиск (также известен как метод деления пополам или дихотомия) — классический алгоритм поиска элемента в отсортированном массиве, использующий дробление массива на половины.  Алгоритм включает в себя следующие шаги:   1. Определение значения элемента в середине массива. Полученное значение сравнивается с искомым элементом. 2. Если искомый элемент меньше значения середины, то поиск осуществляется в первой половине элементов, иначе — во второй. 3. Поиск сводится к тому, что вновь определяется значение серединного элемента в выбранной половине и сравнивается с искомым элементом. 4. Процесс продолжается до тех пор, пока не будет найден элемент, равный искомому, или не станет пустым интервал для поиска.   Существует ряд особенностей, которые необходимо учитывать при программной реализации кода:   1. Пусть first и last – индексы элементов, являющихся левым и правым концом отрезка поиска на некотором шаге алгоритма. В случае больших массивов код (first + last) / 2 для поиска середины отрезка может быть ошибочен, если first и last по отдельности умещаются в свой тип, а first+last — нет. Для обхода этого ограничения можно использовать выражение first + (last - first) / 2, которое привет к такому же результату, но позволит не выйти за границы типа данных. 2. Необходимо тестировать код для случаев пустого массива. Массива из одного элемента и другие частные случаи в зависимости от условий задачи. 3. Массив может содержать несколько экземпляров искомого элемента. В зависимости от условия задачи, необходимо учитывать требования поиска первого экземпляра элемента, последнего экземпляра элемента, произвольного экземпляра и т.п. | | |
| **Пример 1.1** | | |
| ***Задача:*** | | |
|  | Написать программу для бинарного поиска элемента в отсортированном массиве | |
| ***Решение:*** | | |
|  |  | |
| ***Ответ:*** | | |
|  |  | |
| **Задание 1.1** | | |
| ***Задача:*** | | |
|  | | Написать программу для бинарного поиска элемента в отсортированном массиве. В случае, если элемент не найден, указать позицию, в которую можно выполнить вставку элемента (чтобы массив оставался отсортированным) и выполнить вставку |
| ***Решение:*** | | |
|  | | def binary\_search(lst, num):      l = 0      r = len(lst) - 1      while l <= r:          mid = (r + l) // 2          if lst[mid] > num:              r = mid - 1          else:              l = mid + 1          if lst[l] == num:              print(f'Элемент найден, его индекс {lst.index(num)}')              break      else:          print(f'Элемент не найден, но он мог бы хорошо вписаться на индекс {lst.index(l) + 1}, именно туда мы его и впишем')          lst.insert(lst.index(l) + 1, num)          print(f'Теперь наш список выглядит вот так: {lst}')  lst = [1, 2, 4, 5, 6, 8]  binary\_search(lst, 3) |
| ***Ответ:*** | | |
|  | |  |
| **Задание 1.2** | | |
| ***Задача:*** | | |
|  | | Массив arr называется горным, если выполняются следующие свойства:   * 1. В массиве не менее трех элементов   2. Существуют i (0 < i < arr.length – 1) что:      + arr[0] < arr[1] < ... < arr[i - 1] < arr[i]      + arr[i] > arr[i + 1] > ... > arr[arr.length - 1]   Иными словами, в массиве есть пик (или несколько пиков) такие, что остальные элементы убывают влево и вправо относительно пика.   1. проверить, является ли массив arr горным. 2. если да – с помощью бинарного поиска вернуть индекс первого пика. |
| ***Решение:*** | | |
|  | |  |
| ***Ответ:*** | | |
|  | | def find\_peak(arr):      left, right = 0, len(arr) - 1      while left < right:          mid = left + (right - left) // 2          if arr[mid] > arr[mid + 1]:              right = mid          else:              left = mid + 1      return left if left > 0 and left < len(arr) - 1 else -1  def is\_mountain\_array(arr):      if len(arr) < 3:          return False      peak\_index = find\_peak(arr)      return peak\_index != -1  # Пример использования:  arr = [1, 3, 5, 4, 2]  if is\_mountain\_array(arr):      peak\_index = find\_peak(arr)      print(f"Массив является горным. Индекс первого пика: {peak\_index}")  else:      print("Массив не является горным.") |
| **Задание 1.3** | | |
| ***Задача:*** | | |
|  | | Массив отсортирован по возрастанию (элементы могут повторяться). С помощью алгоритма бинарного поиска выяснить, каких чисел в массиве больше – положительных или отрицательных. Если больше положительных чисел – вывести их количество, если же больше отрицательных, то вывести их количество |
| ***Решение:*** | | |
|  | |  |
| ***Ответ:*** | | |
|  | | def binary\_search(lst):      l = 0      r = len(lst)      minus = 0      plus = 0      while l < r:          mid = (l + r) // 2          if lst[mid] < 0:              minus += mid - l + 1              l = mid + 1          else:              plus += r - mid + 1              r = mid - 1      if plus > minus:          print(f'Положительных чисел больше, их {plus}')      else:          print(f'Отрицательных чисел больше, их {minus}')    lst = [-100, -90, -60, -10, 5, 6, 7, 8, 9, 10]  binary\_search(lst) |
| **Задание 1.4\*** | | |
| ***Задача:*** | | |
|  | | Дан целочисленный массив nums. Постройте целочисленный массив counts, где counts[i]— количество меньших элементов справа от nums[i].  **Пример**:  *Ввод: nums = [5,2,6,1]*  *Вывод: [2,1,1,0]*  *Объяснение:*  *Справа от 5 находятся 2 меньших элемента (2 и 1).*  *Справа от 2 находится только 1 элемент меньшего размера (1).*  *Справа от 6 находится 1 элемент поменьше (1).*  *Справа от 1 находится 0 элементов меньшего размера.* |
| ***Решение:*** | | |
|  | |  |
| ***Ответ:*** | | |
|  | | def binary\_search(lst):      lst1 = [0 for i in lst]      for i in range(len(lst) - 1):          for j in range(i + 1, len(lst)):              if lst[i] > lst[j]:                  lst1[i] += 1      print(lst1)  lst = [5, 2, 6, 1]  binary\_search(lst) |