САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №5

по курсу «Алгоритмы и структуры данных»

Тема: Деревья. Пирамида, пирамидальная сортировка. Очередь с приоритетами.

Вариант 11

Выполнила:

Жмачинская Д.С.

К3141

Проверил: Афанасьев А.В.

Санкт-Петербург 2024 г.

# Задачи по варианту

# **Задача 2:**

# Текст задачи.

# 

# Листинг кода.

*"""Module for calculating the height of a tree from a list of parents."""*from typing import List  
  
  
class TreeHeightCalculator:  
 *"""  
 A class to calculate the height of a tree.  
 Assumes that -1 in the parents list denotes the root of the tree.  
 """* def tree\_height(self, n: int, parents: List[int]) -> int:  
 *"""  
 Calculates the height of the tree.  
  
 :param n: Number of nodes.  
 :param parents: List of parents for each node (index = node, value = parent).  
 :return: Height of the tree.  
 """* tree: List[List[int]] = [[] for \_ in range(n)]  
 root: int = -1  
  
 for node in range(n):  
 parent = parents[node]  
 if parent == -1:  
 root = node  
 else:  
 tree[parent].append(node)  
  
 def height(node: int) -> int:  
 *"""Recursively calculates the height of the tree from the given node."""* if not tree[node]:  
 return 1  
 return 1 + max(height(child) for child in tree[node])  
  
 return height(root)

**Текстовое объяснение решения.**

**Для вычисления высоты дерева используется рекурсивный алгоритм, который строит дерево на основе списка родителей. Элементы, равные `-1`, определяются как корни дерева. Для каждого узла рекурсивно определяется глубина от текущего узла до корня, сравнивая максимальную глубину для всех дочерних узлов.**

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.004003 сек | 0.30МБ |
| Пример из задачи | 0.05832 сек | 0.52 МБ |
| Пример из задачи | 0.09345 сек | 1.45 МБ |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.42455 сек | 1.99 МБ |

**Вывод по задаче.**  
Алгоритм продемонстрировал эффективное решение задачи за линейное время относительно количества узлов, позволяя корректно вычислить высоту дерева независимо от его структуры.

## Задача 7:

## Текст задачи.

# 

# Листинг кода.

*"""Module for performing heap-based sorting of an array."""*from typing import List  
  
  
class HeapSorter:  
 *"""  
 A class to perform heap sort on an array.  
 """* def heapify(self, array: List[int], size: int, root\_index: int) -> None:  
 *"""  
 Converts a subtree with root at root\_index into a max-heap.  
  
 :param array: The array to be heapified.  
 :param size: The size of the heap within the array.  
 :param root\_index: The index of the root of the subtree.  
 """* largest: int = root\_index  
 left\_child: int = 2 \* root\_index + 1  
 right\_child: int = 2 \* root\_index + 2  
  
 # Check if left child exists and is greater than root  
 if left\_child < size and array[left\_child] > array[largest]:  
 largest = left\_child  
  
 # Check if right child exists and is greater than the current largest  
 if right\_child < size and array[right\_child] > array[largest]:  
 largest = right\_child  
  
 # If the largest element is not the root, swap and continue heapifying  
 if largest != root\_index:  
 array[root\_index], array[largest] = array[largest], array[root\_index]  
 self.heapify(array, size, largest)  
  
 def heapsort(self, array: List[int]) -> None:  
 *"""  
 Performs heap sort on the provided array.  
  
 :param array: The array to be sorted.  
 """* n: int = len(array)  
  
 # Build a max-heap from the array  
 for idx in range(n // 2 - 1, -1, -1):  
 self.heapify(array, n, idx)  
  
 # Extract elements from the heap one by one  
 for idx in range(n - 1, 0, -1):  
 array[0], array[idx] = array[idx], array[0] # Swap  
 self.heapify(array, idx, 0) # Heapify the reduced heap

**Текстовое объяснение решения.**

**Алгоритм сначала преобразует входной массив в max-heap. Затем извлекаются элементы максимального значения (корень дерева), и оставшаяся часть кучи пересобирается. Итерация продолжается до полной сортировки массива.**

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.01134 сек | 1.67 МБ |
| Пример из задачи | 0.02689 сек | 1.77 МБ |
| Пример из задачи | 0.07581 сек | 1.87 МБ |
| Верхняя граница диапазона значений входных данных из текста задачи | 1.02345 сек | 1.90 МБ |

**Вывод по задаче.**

**Heapsort показал себя как эффективный алгоритм сортировки с гарантированной сложностью `O(nlogn)`. Его преимуществом является отсутствие необходимости в дополнительной памяти, так как преобразования происходят внутри исходного массива.**

**Дополнительные задачи**

## Задача 3:

# Текст задачи.

# 

# Листинг кода.

*"""Main script for processing network packets."""*import os  
from typing import List, Tuple  
from lab5.Task3.src.NetworkPacketsProcessor import NetworkPacketsProcessor  
from lab5.utils.IOHandler import IOHandler  
from lab5.utils.consts import TXT\_DIR, INPUT\_FILES\_DIR, OUTPUT\_FILES\_DIR  
from lab5.utils.decorate import measure\_time\_and\_memory  
  
  
@measure\_time\_and\_memory  
def main() -> None:  
 *"""Main function to read input, process network packets, and write output."""* current\_dir: str = os.path.dirname(os.path.abspath(\_\_file\_\_))  
 txtf\_dir: str = IOHandler.get\_path(current\_dir, TXT\_DIR)  
 input\_path: str = IOHandler.get\_path(txtf\_dir, INPUT\_FILES\_DIR, 'input.txt')  
 output\_path: str = IOHandler.get\_path(txtf\_dir, OUTPUT\_FILES\_DIR, 'output.txt')  
  
 lines: List[str] = IOHandler.read\_file(input\_path)  
 if not lines:  
 # If there are no packets, do not output anything  
 return  
  
 # First line contains buffer\_size and number of packets  
 first\_line = lines[0].strip().split()  
 if len(first\_line) == 1:  
 buffer\_size: int = int(first\_line[0])  
 n: int = 0  
 else:  
 buffer\_size: int = int(first\_line[0])  
 n: int = int(first\_line[1])  
  
 packets: List[Tuple[int, int]] = []  
 for line in lines[1:]:  
 arr, proc = line.strip().split()  
 packets.append((int(arr), int(proc)))  
  
 processor: NetworkPacketsProcessor = NetworkPacketsProcessor()  
 results: List[str] = processor.network\_packets(buffer\_size, packets)  
  
 # Convert start times to strings, keeping '-1' for dropped packets  
 output\_lines: List[str] = [time\_start for time\_start in results]  
 IOHandler.write\_file(output\_path, "\n".join(output\_lines))  
 print("Обработка завершена. Результаты записаны в output.txt")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

**Текстовое объяснение решения.**

Для моделирования обработки сетевых пакетов используется буфер фиксированного размера. Пакеты обрабатываются в порядке их поступления. Если буфер заполнен, вновь поступивший пакет отбрасывается. Время обработки каждого пакета определяется на основе времени окончания обработки предыдущего.

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.003465 сек | 0.35 МБ |
| Пример из задачи | 0.03958 сек | 0.56 МБ |
| Пример из задачи | 0.10345 сек | 1.59 МБ |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.4124 сек | 1.89 МБ |

**Вывод по задаче.**  
Алгоритм точно моделирует обработку пакетов в реальных сетях, справляясь с ограничением памяти на фиксированный буфер. Это позволяет эффективно управлять сетевой нагрузкой.

## Задача 6:

# Текст задачи.

# 

# Листинг кода.

*"""Module for processing priority queue operations."""*import heapq  
from typing import List, Tuple  
  
  
class PriorityQueueProcessor:  
 *"""  
 A class to process priority queue operations:  
 - A x: Add element x.  
 - X: Extract the minimum element.  
 - D x y: Decrease the value of the element added in line x+1 to y.  
 """* def \_\_init\_\_(self):  
 self.heap: List[Tuple[int, int]] = [] # Min-heap  
 self.element\_map: dict = {} # id → value  
 self.id\_map: dict = {} # line number → element id  
 self.removed\_elements: set = set() # Set of removed elements  
 self.next\_id: int = 0 # Unique id for each added element  
  
 def process\_operations(self, operations: List[str]) -> List[str]:  
 *"""  
 Processes a list of priority queue operations.  
  
 :param operations: List of operation strings.  
 :return: List of results for 'X' operations.  
 """* output\_results: List[str] = []  
 for index, operation in enumerate(operations):  
 tokens = operation.split()  
 command = tokens[0]  
 if command == "A":  
 # Add element  
 value = int(tokens[1])  
 heapq.heappush(self.heap, (value, self.next\_id))  
 self.element\_map[self.next\_id] = value  
 self.id\_map[index + 1] = self.next\_id  
 self.next\_id += 1  
  
 elif command == "X":  
 # Extract minimum  
 self.\_remove\_invalid\_elements()  
 if self.heap:  
 \_, element\_id = heapq.heappop(self.heap)  
 output\_results.append(str(self.element\_map[element\_id]))  
 self.removed\_elements.add(element\_id)  
 else:  
 output\_results.append("\*")  
  
 elif command == "D":  
 # Decrease element value  
 line\_number = int(tokens[1]) + 1  
 new\_value = int(tokens[2])  
 element\_id = self.id\_map[line\_number]  
 self.element\_map[element\_id] = new\_value  
 heapq.heappush(self.heap, (new\_value, element\_id))  
  
 return output\_results  
  
 def \_remove\_invalid\_elements(self) -> None:  
 *"""Removes elements from the heap that have already been removed."""* while self.heap and self.heap[0][1] in self.removed\_elements:  
 heapq.heappop(self.heap)

**Текстовое объяснение решения.**

Реализована структура минимальной кучи для операций с приоритетной очередью. Для добавления элемента производится вставка с сохранением структуры кучи. Извлечение минимального элемента выполняется путем замены корня последним элементом и восстановления свойств кучи. Уменьшение значения элемента происходит через перестройку структуры вниз или вверх.

|  |  |  |
| --- | --- | --- |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.004577 сек | 0.25 МБ |
| Пример из задачи | 0.04566 сек | 0.44 МБ |
| Пример из задачи | 0.12357 сек | 1.69 МБ |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.4675 сек | 2.00 МБ |

**Вывод по задаче.**

Алгоритм позволяет эффективно реализовывать основные операции с приоритетными очередями за логарифмическое время, что делает его подходящим для задач с высокими требованиями к производительности.

# Вывод

Общий вывод по лабораторной работе  
Лабораторная работа продемонстрировала использование различных структур данных (деревьев, кучи, очередей с приоритетами) и алгоритмов для решения задач.  
Работа позволила углубить понимание принципов работы базовых структур данных, их оптимизации и алгоритмов сортировки. Все задачи выполнены с учетом ограничений по времени и памяти, что подчеркивает их практическую применимость.