**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

Курсовая РАБОТА

**по дисциплине «Алгоритмы и структуры данных»**

Тема: Реализация структуры данных

|  |  |  |
| --- | --- | --- |
| Студент гр. 3343 |  | Коршков А.А. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Коршков А.А. | | |
| Группа 3343 | | |
| Тема работы: Реализация структуры данных | | |
| Исходные данные:  Вариант 4.2 | | |
| Содержание пояснительной записки: разделы «Содержание», «Введение», «Задание», «Исследование», «Описание реализованной программы», «Анализ полученных результатов», «Заключение», «Список использованных источников», «Приложение А» | | |
| Предполагаемый объем пояснительной записки:  Не менее 22 страниц. | | |
| Дата выдачи задания: 06.11.2024 | | |
| Дата сдачи реферата: 19.12.2024 | | |
| Дата защиты реферата: 19.12.2024 | | |
| Студент гр. 3343 |  | Коршков А.А. |
| Преподаватель |  | Иванов Д.В. |

# **АННОТАЦИЯ**

В ходе курсовой работы создана программа, реализующая хранение данных весов тушек. Исходя из требований задания выбрана и реализована подходящая по параметрам структура данных. Для добавления данных в структуру данных используется массив. Результатом работы является класс MaxBinaryHeap, отвечающий за хранение данных о весах тушек. Также были созданы юнит-тесты с использованием фреймворка pytest для различных сценариев работы со структурой данных, простой интерфейс командной строки для демонстрации работы со структурой данных.

**СОДЕРЖАНИЕ**

ОГЛАВЛЕНИЕ

[АННОТАЦИЯ 3](#_Toc185459065)

[ВВЕДЕНИЕ 5](#_Toc185459066)

[ЗАДАНИЕ 6](#_Toc185459067)

[ИССЛЕДОВАНИЕ 7](#_Toc185459068)

[ОПИСАНИЕ РЕАЛИЗОВАННОЙ ПРОГРАММЫ 8](#_Toc185459069)

[Описание структуры данных 8](#_Toc185459070)

[Описание тестов 9](#_Toc185459071)

[Описание главной функции 10](#_Toc185459072)

[АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ 11](#_Toc185459073)

[ЗАКЛЮЧЕНИЕ 12](#_Toc185459074)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 13](#_Toc185459075)

[ПРИЛОЖЕНИЕ А 14](#_Toc185459076)

[ИСХОДНЫЙ КОД ПРОГРАММ 14](#_Toc185459077)

# **ВВЕДЕНИЕ**

Цель курсовой работы – выбор и реализация необходимой структуры данных, которая оптимально подходит под требования поставленного задания. Для выполнения этой цели необходимо решить следующие задачи:

1. Сравнить существующие различные структуры данных, которые могут подойти для выполнения задания.
2. Изучить недостатки и преимущества этих структур в плане использования памяти, сложность для различных операций.
3. На основе полученной информации выбрать структуру данных.
4. Реализовать выбранную структуру данных.

# **ЗАДАНИЕ**

Вариант 2

В магазине мясопродуктов продается бройлер. Покупатели предпочитают тушки побольше, поэтому всегда выбирают курицу максимального веса. Каждый день в магазин поступает новая партия бройлеров, которую можно представить в виде списка весов тушек. Прежде владелец для каждого покупателя взвешивал каждую доступную тушку и выбирал из них максимальную по весу. Со временем он понял, что это неоптимальный вариант.

Помогите владельцу с решением данной проблемы путем выбора и реализации оптимальной структуры данных для хранения весов тушек.

# **ИССЛЕДОВАНИЕ**

Для решения поставленной задачи необходимо выбрать оптимальную структуру данных. Первичный отбор среди всех структур данных осуществлялся по двум критериям, сформулированным из условия задачи: быстрое получение бройлера максимального веса и быстрая вставка информации о новой партии бройлеров в виде массива весов. В результате отсеивания наименее оптимальных для задачи структур, осталось две: макс-куча и АВЛ дерево.

Исследование этих двух структур показало, что вставка в макс-кучу имеет временную сложность алгоритма вставки порядка O(log n), а получение данных по убыванию является простой задачей – необходимо получать данные из корня макс-кучи O(1).

В АВЛ-дерево все операции выполняются в среднем за O(log n) за счёт выполнения балансировки после вставки/удаления элемента. Эта реализация неплохо себя показывает, когда необходимо на все операции иметь примерно равную сложность.

По условию задачи продавец использовал список всех тушек и искал максимальный элемент. В результате операция нахождения максимума занимал O(n). При очень небольших объёмах данных (около 10 элементов) этот способ может быть достаточным для выполнения задачи. Однако если количество элементов намного больше, то данный способ становится слишком неэффективным.

Было решено использовать макс-кучу, так как в условии задачи четко указано, что необходимо получать максимальный вес тушки за минимальное и извлекать данную позицию из ассортимента, это основные требования задачи. Найти бройлер максимального размера можно за O(1), но после того, как достали максимальный элемент необходимо выполнить “просеивание”, чтобы восстановить кучу. Однако это всё намного эффективнее, чем способ по условию задачи. Чтобы добавить бройлер в позицию нужно после вставки выполнить.

# **ОПИСАНИЕ РЕАЛИЗОВАННОЙ ПРОГРАММЫ**

## Описание структуры данных

Класс MaxBinaryHeap представляет реализованную по заданию структуру данных. Это бинарная макс куча, о котором говорилось в Исследовании. Конструктор ничего не принимает, создаётся как новый объект.

Описание полей:

* \_\_heap – массив, который является основой для операций макс-кучи.

Описание методов:

* \_parent - Вычисляет индекс родительского узла для заданного индекса в куче.
* \_left\_child - Вычисляет индекс левого дочернего узла для заданного индекса в куче.
* \_right\_child - Вычисляет индекс правого дочернего узла для заданного индекса в куче.
* \_\_iter\_\_ – инициализирует итератор по структуре (в порядке убывания баллов студентов)
* \_\_next\_\_ – возвращает следующего студента по ходу итерации. Выбрасывает StopIteration при достижении конца структуры
* \_left\_child - Вычисляет индекс левого дочернего узла для заданного индекса в куче.
* \_right\_child - Вычисляет индекс правого дочернего узла для заданного индекса в куче.
* \_swap - Меняет местами элементы в куче по заданным индексам.
* \_\_heapify\_up - Восстанавливает свойства кучи, перемещая элемент вверх по куче, если он больше своего родителя.
* \_\_heapify\_down - Восстанавливает свойства кучи, перемещая элемент вниз по куче, если он меньше своих дочерних элементов.
* insert - Добавляет новый элемент в кучу. После добавления вызывает метод \_\_heapify\_up.
* extract\_max - Извлекает и возвращает максимальный элемент из кучи. После извлечения вызывает метод \_\_heapify\_down.
* peek\_max - Возвращает максимальный элемент из кучи без его удаления.
* \_\_len\_\_ - Возвращает количество элементов в куче.
* is\_empty - Проверяет, является ли куча пустой.
* \_\_str\_\_ - Возвращает строковое представление кучи, где элементы разделены запятыми.

Также были созданы дополнительные функции для работы с вводом/выводом в консоли и взаимодействием с пользователем:

* clear\_console - Очищает консоль экрана. Использует команду cls для Windows и clear для Unix-подобных систем.
* get\_answer - Получает выбор пользователя из консольного меню. Возвращает целое число от 1 до 4. Если ввод некорректен, выбрасывает исключение ValueError.
* deliver\_broilers - Добавляет новые бройлеры в инвентарь магазина на основе пользовательского ввода. Преобразует ввод в список весов и добавляет их в приоритетную очередь.
* handle\_delivery - Управляет процессом поставки бройлеров в магазин. Запрашивает у пользователя веса бройлеров, добавляет их в очередь и обрабатывает ошибки ввода.
* handle\_sale - Управляет продажей бройлера с максимальным весом. Выводит вес проданного бройлера и удаляет его из очереди. Если очередь пуста, сообщает об этом.
* print\_broilers - Отображает текущий инвентарь бройлеров в магазине. Если очередь не пуста, выводит список бройлеров.

## Описание тестов

Для тестирования корректности работы структуры созданы юнит-тесты, проверяющие различные сценарии работы со структурой, в том числе «крайние случаи». Используется библиотека pytest, запуск тестов выполняется командой «pytest tests.py».

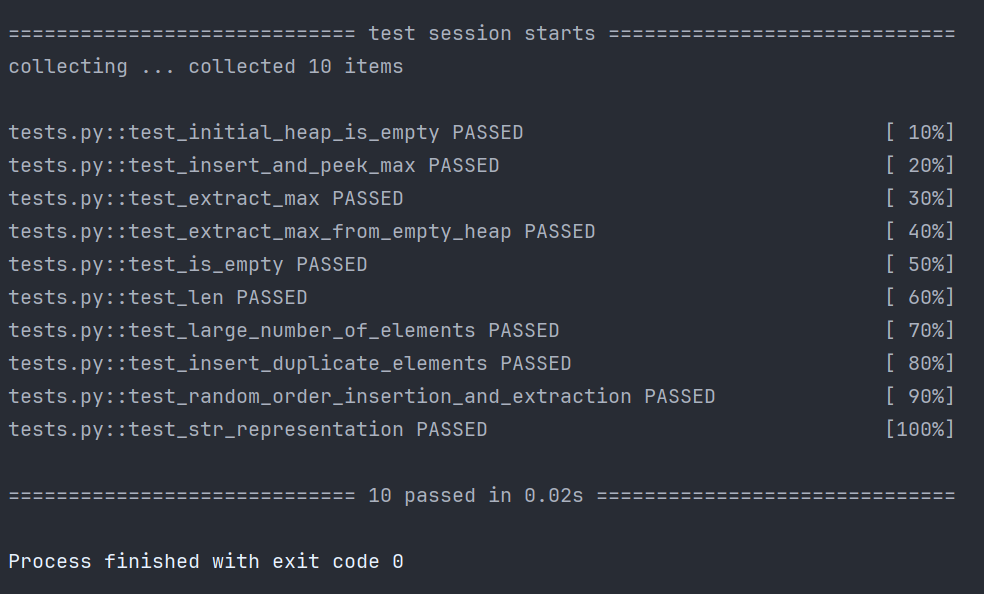


Рисунок 1 – Результат выполнения тестов

## Описание главной функции

Для демонстрации взаимодействия со структурой создан небольшой интерфейс командной строки.

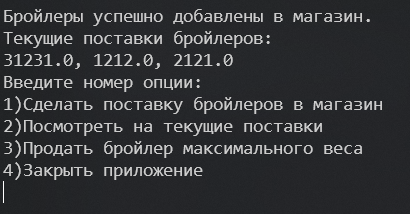


Рисунок 2 – Демонстрация интерфейса командной строки.

Полный разработанный программный код (структуры, главной функции и тестов) см. в Приложении А.

# **АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ**

Для исследования скорости работы структуры было измерено время работы операции вставки на различных объемах данных. Сводные результаты измерений представлены в Таблице 1. Все значения времени в таблице представлены в секундах.

Таблица 1 – Результаты исследования

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Операция, c | Размер выборки | | | | |
| 100 | 1000 | 10000 | 100000 | 1000000 |
| Извлечение макс. элемента | 0.000009 | 0.000010 | 0.000016 | 0.000021 | 0.000044 |
| Нахождение макс. элемента | 0.000001 | 0.000002 | 0.000003 | 0.000003 | 0.000002 |
| Вставка нового элемента | 0.000007 | 0.000008 | 0.000009 | 0.000021 | 0.000023 |

Как видно из данных Таблицы 1, операция извлечения максимального элемента при любом из представленных объемов данных занимает менее, чем 10-4 секунды, что позволяет сделать вывод о подтверждении теоретических данных и заключить, что вставка в структуру выполняется за O(log n) по времени.

Также нахождение максимального элемента для любых размеров выборки происходит менее, чем за 10-5 секунд. Это доказывает, что нахождения максимума выполняется примерно за O(1).

Вставка нового элемента выполняется менее, чем за 10-5 секунд, что также подтверждает сложность данной операции O(log n).

Вся приведённая выше информация подтверждает заявленную в теоретическом исследовании гипотезу о эффективности структуры для решения задачи, поставленной в рамках варианта курсовой работы.

# **ЗАКЛЮЧЕНИЕ**

Курсовая работа успешно выполнена. Результаты практических измерений времени работы структуры подтвердили ее заявленную эффективность. В процессе разработки был реализован весь необходимый функционал в соответствии с требованиями курсовой работы. Кроме того, создан простой интерфейс командной строки для демонстрации работы структуры, написаны и выполнены юнит-тесты.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Двоичная куча // ИТМО URL: https://neerc.ifmo.ru/wiki/index.php?title=Двоичная\_куча (дата обращения: 12.12.2024).

2. АВЛ-дерево // ИТМО URL: https://neerc.ifmo.ru/wiki/index.php?title=АВЛ-дерево (дата обращения: 12.12.2024).

3. Куча (структура данных) // Codechick URL: https://codechick.io/tutorials/dsa/dsa-heap (дата обращения: 12.12.2024).

4. АВЛ-дерево // Codechick URL: https://codechick.io/tutorials/dsa/dsa-trees (дата обращения: 12.12.2024).

# **ПРИЛОЖЕНИЕ А**

# **ИСХОДНЫЙ КОД ПРОГРАММ**

Название файла:main.py

"""

The main file.

"""

from modules.max\_binary\_heap import MaxBinaryHeap

from modules.console\_functions import get\_answer, clear\_console, handle\_delivery, handle\_sale, print\_broilers

def main() -> None:

    """

    The main function of program.

    """

    pq: MaxBinaryHeap = MaxBinaryHeap()

    clear\_console()

    print("Здравствуйте, милорд! Чтобы вы хотели сделать?")

    while True:

        try:

            print("Введите номер опции:")

            ans: int = get\_answer()

        except ValueError:

            clear\_console()

            print(

                "Ошибка! Неправильный ввод.\n"

                "Пожалуйста, введите число от 1 до 4."

            )

            continue

        if ans == 1:

            clear\_console()

            handle\_delivery(pq)

            print\_broilers(pq)

        elif ans == 2:

            clear\_console()

            print\_broilers(pq)

        elif ans == 3:

            clear\_console()

            handle\_sale(pq)

            print\_broilers(pq)

        elif ans == 4:

            exit(0)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Название файла:console\_functions.py

"""

This file contains functions to work with console in meat's shop.

"""

from os import name as os\_name

from os import system

from modules.max\_binary\_heap import MaxBinaryHeap

def clear\_console() -> None:

    """

    Clears the console screen.

    This function uses the appropriate system command to clear the console

    screen based on the operating system. It uses 'cls' for Windows (nt)

    and 'clear' for Unix-based systems.

    Parameters:

    None

    Returns:

    None

    """

    system("cls" if os\_name == "nt" else "clear")

def get\_answer() -> int:

    """

    This function retrieves a user's choice from the console menu.

    Parameters:

    None

    Returns:

    int: The user's choice as an integer. The value must be between 1 and 4.

         If the user enters an invalid choice, a ValueError is raised.

    """

    ans = int(

        input(

            "1)Сделать поставку бройлеров в магазин\n"

            "2)Посмотреть на текущие поставки\n"

            "3)Продать бройлер максимального веса\n"

            "4)Закрыть приложение\n"

        )

    )

    if not 1 <= ans <= 4:

        raise ValueError("Answer must be between 1 and 4")

    return ans

def deliver\_broilers(pq: MaxBinaryHeap) -> MaxBinaryHeap:

    """

    Adds new broilers to the shop's inventory based on user input.

    This function prompts the user to enter the weights of broilers,

    converts the input to a list of floats, and inserts each weight

    into the provided priority queue.

    Parameters:

    pq (MaxBinaryHeap): The priority queue representing the current inventory of broilers.

    Returns:

    MaxBinaryHeap: The updated priority queue with new broilers added.

    Raises:

    ValueError: If the user input cannot be converted to float values.

    """

    weights = list(map(float, input().replace(",", ".").split()))

    for i in weights:

        pq.insert(i)

    return pq

def handle\_delivery(pq: MaxBinaryHeap) -> None:

    """

    Handles the delivery of broilers to the shop.

    This function manages the process of adding new broilers to the shop's inventory.

    It prompts the user to input the weights of the broilers, attempts to add them

    to the priority queue, and handles any input errors.

    Parameters:

    pq (MaxBinaryHeap): The priority queue representing the current inventory of broilers.

    Returns:

    None

    Side effects:

    - Modifies the provided MaxBinaryHeap object by adding new broiler weights.

    - Prints messages to the console for user interaction and feedback.

    - Clears the console screen during the process.

    """

    print("Сделать поставку бройлеров в магазин.\n" "Введите через пробел вес бройлеров:")

    while True:

        try:

            pq = deliver\_broilers(pq)

            clear\_console()

            print("Бройлеры успешно добавлены в магазин.")

            break

        except ValueError:

            clear\_console()

            print(

                "Ошибка! Некорректный ввод веса!\n"

                "Введите ещё раз вес каждого бройлера через пробел."

            )

def handle\_sale(pq: MaxBinaryHeap) -> None:

    """

    Handles the sale of a broiler from the shop's inventory.

    This function attempts to sell the broiler with the maximum weight from the

    priority queue. It prints the weight of the sold broiler and removes it

    from the queue. If the queue is empty, it informs that the inventory is empty.

    Parameters:

    pq (MaxBinaryHeap): The priority queue representing the current inventory of broilers.

    Returns:

    None

    Side effects:

    - Modifies the provided MaxBinaryHeap object by removing the maximum element.

    - Prints messages to the console about the sale or empty inventory status.

    """

    try:

        print("Продан бройлер", pq.peek\_max())

        pq.extract\_max()

    except IndexError:

        print("Ассортимент пуст.")

def print\_broilers(pq: MaxBinaryHeap):

    """

    Displays the current inventory of broilers in the shop.

    This function checks if the priority queue of broilers is not empty.

    If there are broilers in the inventory, it prints the current list

    of broilers.

    Parameters:

    pq (MaxBinaryHeap): The priority queue representing the current inventory of broilers.

    Returns:

    None

    Side effects:

    - Prints the current inventory of broilers to the console if not empty.

    """

    if not pq.is\_empty():

        print("Текущие поставки бройлеров:")

        print(pq)

Название файла:max\_binary\_heap.py

"""

This file contains the implementation of a max binary heap.

"""

class MaxBinaryHeap:

    """

    This class implements a max binary heap.

    """

    def \_\_init\_\_(self, heap) -> None:

        """

        Initializes a new instance of the MaxBinaryHeap class.

        This constructor sets up an empty list to represent the heap.

        """

        self.\_\_heap: list = []

        if isinstance(heap, list) or isinstance(heap, tuple):

            for i in heap: self.insert(i)

    def \_parent(self, index: int) -> int:

        """

        Calculates the index of the parent node in a binary heap.

        Args:

            index (int): The index of the current node.

        Returns:

            int: The index of the parent node.

        """

        return (index - 1) // 2

    def \_left\_child(self, index: int) -> int:

        """

        Calculates the index of the left child node in a binary heap.

        Args:

            index (int): The index of the current node. This index is 0-based.

        Returns:

            int: The index of the left child node. If the left child does not exist,

                 returns the index of the last node in the heap.

        """

        return 2 \* index + 1

    def \_right\_child(self, index: int) -> int:

        """

        Calculates the index of the right child node in a binary heap.

        Args:

            index (int): The index of the current node. This index is 0-based.

        Returns:

            int: The index of the right child node. If the right child does not exist,

                 returns the index of the last node in the heap.

        """

        return 2 \* index + 2

    def \_swap(self, i: int, j: int) -> None:

        """

        Swaps the elements at indices i and j in the heap.

        This method is used to maintain the heap property when elements are moved

        up or down the heap.

        Parameters:

        - i (int): The index of the first element to swap.

        - j (int): The index of the second element to swap.

        Returns:

        - None: This method does not return a value. It modifies the heap in-place.

        """

        self.\_\_heap[i], self.\_\_heap[j] = self.\_\_heap[j], self.\_\_heap[i]

    def \_\_heapify\_up(self, index: int) -> None:

        """

        Performs a heapify operation on the max heap

        by moving the element at the given index upwards until the heap property is satisfied.

        Args:

            index (int): The index of the element to heapify upwards. This index is 0-based.

        Returns:

            None: This method does not return a value. It modifies the heap in-place.

        """

        parent: int = self.\_parent(index)

        if index > 0 and self.\_\_heap[parent] < self.\_\_heap[index]:

            self.\_swap(parent, index)

            self.\_\_heapify\_up(parent)

    def \_\_heapify\_down(self, index: int) -> None:

        """

        Performs a heapify operation on the max heap by moving the element

        at the given index downwards until the heap property is satisfied.

        Args:

        - index (int): The index of the element to heapify downwards. This index is 0-based.

        Returns:

        - None: This method does not return a value. It modifies the heap in-place.

        Local Variables:

        - left (int): The index of the left child node.

        - right (int): The index of the right child node.

        - largest (int): The index of the largest node between the current node and its children.

        The method compares the values of the current node

        and its children, and swaps the current node with the largest child if necessary.

        The method then recursively calls itself on the largest child

        until the heap property is satisfied.

        """

        left: int = self.\_left\_child(index)

        right: int = self.\_right\_child(index)

        largest: int = index

        if left < len(self.\_\_heap) and self.\_\_heap[left] > self.\_\_heap[largest]:

            largest = left

        if right < len(self.\_\_heap) and self.\_\_heap[right] > self.\_\_heap[largest]:

            largest = right

        if largest != index:

            self.\_swap(index, largest)

            self.\_\_heapify\_down(largest)

    def insert(self, value) -> None:

        """

        Inserts a new element into the priority queue.

        Args:

        - value (int or float): The value to be inserted into the priority queue.

        Returns:

        - None: This method does not return a value. It modifies the priority queue in-place.

        Raises:

        - IndexError: If the priority queue is empty,

        attempting to insert a new element will raise an IndexError.

        This method appends the new element to the end of the heap

        and then calls the \_\_heapify\_up method to restore the heap property.

        The \_\_heapify\_up method moves the new element upwards in the heap

        until the heap property is satisfied.

        """

        if type(value) != int and type(value) != float:

            raise ValueError("Value must be an integer or float.")

        self.\_\_heap.append(value)

        self.\_\_heapify\_up(len(self.\_\_heap) - 1)

    def extract\_max(self):

        """

        Extracts and returns the maximum value from the priority queue.

        This method removes and returns the maximum value from the priority queue. If the priority

        queue is empty, it raises an IndexError. After extracting the maximum value, the method

        restores the heap property by calling the \_\_heapify\_down method.

        Parameters:

        - None: This method does not take any parameters.

        Returns:

        - float: The maximum value in the priority queue.

        Raises:

        - IndexError: If the priority queue is empty,

        attempting to extract a maximum value will raise an IndexError.

        """

        if self.is\_empty():

            raise IndexError("Priority queue is empty")

        max\_value = self.\_\_heap[0]

        self.\_\_heap[0] = self.\_\_heap[-1]

        self.\_\_heap.pop()

        self.\_\_heapify\_down(0)

        return max\_value

    def peek\_max(self):

        """

        Retrieves the maximum value from the priority queue without removing it.

        This method returns the maximum value in the priority queue. If the priority queue is empty,

        it raises an IndexError. The maximum value is not removed from the priority queue, so the

        priority queue remains unchanged.

        Parameters:

        - None: This method does not take any parameters.

        Returns:

        - int: The maximum value in the priority queue.

        Raises:

        - IndexError: If the priority queue is empty,

        attempting to retrieve a maximum value will raise an IndexError.

        """

        if self.is\_empty():

            raise IndexError("Priority queue is empty")

        return self.\_\_heap[0]

    def \_\_len\_\_(self) -> int:

        """

        Returns the number of elements in the priority queue.

        This method returns the number of elements in the priority queue. The priority queue is

        represented by a binary heap, and the length of the heap is equal to the number of elements

        in the priority queue.

        Parameters:

        - None: This method does not take any parameters.

        Returns:

        - int: The number of elements in the priority queue.

        """

        return len(self.\_\_heap)

    def is\_empty(self) -> bool:

        """

        Checks if the priority queue is empty.

        This method returns True if the priority queue is empty, and False otherwise. The priority

        queue is represented by a binary heap, and the length of the heap is equal to zero when

        the priority queue is empty.

        Parameters:

        - None: This method does not take any parameters.

        Returns:

        - bool: True if the priority queue is empty, False otherwise.

        """

        return len(self) == 0

    def \_\_str\_\_(self: object) -> str:

        """

        Returns a string representation of the priority queue.

        This method returns a string representation of the priority queue, where each element in the

        heap is separated by a comma and a space. The string representation is enclosed in square brackets

        and does not include any quotation marks or parentheses.

        Parameters:

        - None: This method does not take any parameters.

        Returns:

        - str: A string representation of the priority queue.

        """

        return ", ".join(map(str, self.\_\_heap))