МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 42

ОТЧЁТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

| старший преподаватель |  |  |  | С.Ю. Гуков |
| --- | --- | --- | --- | --- |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

| ОТЧЁТ О ЛАБОРАТОРНОЙ РАБОТЕ №6 |
| --- |
| Построение кучи |
| по курсу: |
| АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ |
|  |

РАБОТУ ВЫПОЛНИЛ

| СТУДЕНТ гр. № | 4329 |  |  |  | Д.С. Шаповалова |
| --- | --- | --- | --- | --- | --- |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

ОГЛАВЛЕНИЕ

[Цель работы 3](#_30j0zll)

[Постановка задачи 3](#_1fob9te)

[Схема алгоритма решения 4](#_3znysh7)

[Полное описание реализованной функции 4](#_gqc1nhfmq022)

[Листинг программы 5](#_2et92p0)

[Результат выполнения программы. 5](#_tyjcwt)

[ВЫВОДЫ 6](#_3dy6vkm)

# Цель работы

Написать программу, переставляющую элементы заданного массива чисел так, чтобы он удовлетворял свойству мин-куч.

# Постановка задачи

Задание: Написать программу, переставляющую элементы заданного массива чисел так, чтобы он удовлетворял свойству мин-куч. Текст задания приведён в таблице 1.

Таблица 1. Индивидуальное задание

| № | Текст задания | Вход | Выход |
| --- | --- | --- | --- |
| 0 | Написать программу, переставляющую элементы заданного массива чисел так, чтобы он удовлетворял свойству мин-куч.  **Выход:** Первая строка выхода должна содержать число обменов m, которое должно удовлетворять неравенству 0 ≤ m ≤ 4n. Каждая из последующих m строк должна задавать обмен двух элементов массива A. Каждый обмен задаётся парой различных индексов 0 ≤ i != j ≤ n−1. После применения всех обменов в указанном порядке массив должен превратиться в мин-кучу, то есть для всех 0 ≤ i ≤ n−1 должны выполняться следующие два условия:   * если 2i + 1 ≤ n−1, то A[i] < A[2i + 1]. * если 2i + 2 ≤ n−1, то A[i] < A[2i + 2].   *Ограничения.* 1 ≤ n ≤ 105; 0 ≤ A[i] ≤ 109 для всех 0 ≤ i ≤ n−1; все A[i] попарно различны; i != j. | Первая строка содержит число n. Следующая строка задаёт массив чисел A[0], ..., A[n−1]. | Переставить элементы массива так, чтобы выполнялись неравенства A[i] ≤ A[2i + 1] и A[i] ≤ A[2i + 2] для всех i. |

# Схема алгоритма решения

1. Ввод данных
2. Функция build\_min\_heap
3. Функция min\_heapify
4. Вывод перестановок

# Полное описание реализованной функции

**Ввод данных:**

n = int(input()) — считываем размер массива от пользо вателя.

A = list(map(int, input().split())) — считываем сам массив, разбивая его по пробелам и преобразуя в список целых чисел.

swaps = build\_min\_heap(A) — вызываем build\_min\_heap, передавая массив, чтобы построить мин-кучу и вернуть список перестановок.

**Функция build\_min\_heap:**

n = len(array) — число элементов в нашем массиве, которое будет использоваться для описания структуры кучи.

swaps = [] — создаем пустой список, который будет хранить индексы элементов, которые будут подвергнуты обмену.

for i in range(n // 2 - 1, -1, -1): — этот цикл начинается *с последнего родительского узла (индекс n//2 - 1) и идет до корня (индекс 0)*. Это важно, так как у каждого узла в бинарном дереве могут быть дочерние элементы, только если он не находится на самом нижнем уровне.

р - родительский, д - дочерний - [р1, р2, р3, д2, д2, д3], р2 и р3 - д1.

min\_heapify(array, n, i, swaps) — каждый раз, когда мы перебираем индекс i, мы вызываем функцию min\_heapify, чтобы проверить и восстановить мин-кучу относительно элемента с индексом i.

**Функция min\_heapify:**

smallest = i — инициализируем переменную smallest значением i. Она будет хранить индекс наименьшего элемента из текущего узла и его дочерних элементов.

left = 2 \* i + 1 — вычисляем индекс левого дочернего узла для узла i.

right = 2 \* i + 2 — вычисляем индекс правого дочернего узла для узла i.

*Проверка левого дочернего узла:*

if left < n and array[left] < array[smallest]: — проверяет, существует ли левый дочерний узел (индекс left должен быть меньше n). Если существует, то мы сравниваем его значение с текущим наименьшим узлом.

smallest = left — если значение левого дочернего узла меньше, чем значение узла, то обновляем smallest на индекс left.

*Проверка правого дочернего узла:*

if right < n and array[right] < array[smallest]: — аналогично, проверяем, существует ли правый дочерний узел и меньше ли его значение, чем текущее наименьшее.

smallest = right — если условие выполняется, обновляем smallest на индекс right.

*Обмен значений:*

if smallest != i: — если индекс smallest отличается от i, это значит, что наименьший элемент находится не в текущем узле.

array[i], array[smallest] = array[smallest], array[i] — выполняем обмен элементов в массиве на позициях i и smallest при помощи параллельного присваивания — это позволяет нам обменивать значения без использования временной переменной.

swaps.append((i, smallest)) — записываем индексы обмена в список swaps.

*Рекурсивный вызов min\_heapify:*

min\_heapify(array, n, smallest, swaps) — после обмена мы рекурсивно вызываем min\_heapify для дочернего узла smallest, чтобы восстановить свойства мин-кучи для нового узла на позиции smallest.

*Возврат к build\_min\_heap:*

После завершения цикла for и выполнения min\_heapify для всех узлов, функция build\_min\_heap возвращает список swaps, который содержит все произведенные обмены.

**Вывод перестановок:**

print(len(swaps)) — выводим количество произведенных обменов.

for swap in swaps: — начинаем цикл по списку обменов.

print(swap[0], swap[1]) — выводим индексы каждого обмена.

# Листинг программы

def min\_heapify(array, n, i, swaps):

smallest = i

left = 2 \* i + 1

right = 2 \* i + 2

# Проверяем, является ли левый и правый дочерний элемент меньшими

# не вышли за пределы массива? и родительский эл-т меньше дочернего?

# ищем эл-т, чтобы поменять с i-тым.

if left < n and array[left] < array[smallest]:

smallest = left

if right < n and array[right] < array[smallest]:

smallest = right

# Если i-тый эл-т не наименьший - значит стоит не там, меняем.

if smallest != i:

array[i], array[smallest] = array[smallest], array[i]

swaps.append((i, smallest))

print(array)

# проверяем нужно ли переместившийся эл-т опустить ещё ниже

min\_heapify(array, n, smallest, swaps)

def build\_min\_heap(array, n):

swaps = []

# Запускаем функцию восстановления кучи для каждого узла, начиная с последнего родительского

# с р3 - [р1, р2, р3, д2, д2, д3] - [р1, р2, д1, д2, д2]

for i in range(n // 2 - 1, -1, -1): # start, step, stop

min\_heapify(array, n, i, swaps)

return swaps

# Чтение входных данных

flag = False

while not flag:

try:

n = int(input())

A = list(map(int, input().split()))

flag = True

except ValueError:

print("Это не цифры, попробуйте ещё раз")

if flag and len(A)==n:

# Генерация мин-кучи

swaps = build\_min\_heap(A, n)

# Вывод результата

print(len(swaps)) # Количество обменов

for swap in swaps:

print(swap[0], swap[1]) # Индексы обменов

else:

print("Длина массива А и n не совпадают. ")

# 

# Результат выполнения программы.

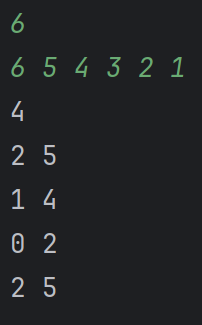


Рисунок 2.1 - 1-ый результат работы программы

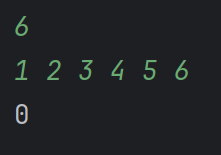
****

Рисунок 2.2 - 2-ой результат работы программы

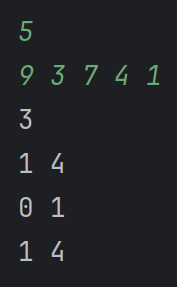


Рисунок 2.3 - 3-ой результат работы программы

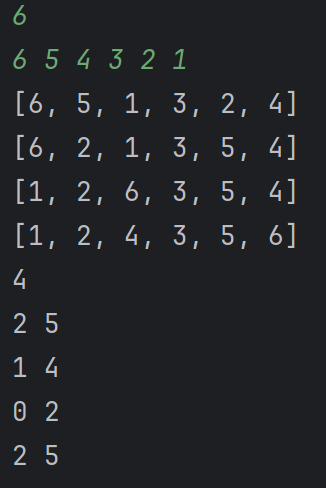


Рисунок 3.1 - 1-ый результат работы программы с отладочной информацией

# Выводы

В ходе выполнения лабораторной работы мной были освоены и изучены: понятие двоичная куча; алгоритм построения мин-кучи. Написанная программа была протестирована, полученный результат соответствует значению в примере.