Лабораторная работа 5 - Деревья. Пирамида (Heap). Очередь с приоритетами

Цель

Разобрать пирамиду (двоичную кучу) и очередь с приоритетами, а также применить их в задачах: построение min-heap и моделирование планировщика заданий.

Задания и решения

Задание 4 - Построение пирамиды (min-heap) и список обменов

По входу задается массив. Нужно построить min-heap и вывести все обмены индексов, которые выполнялись при построении.

Решение сделано через просеивание вниз (sift_down):

- идем по вершинам снизу вверх от `n//2 - 1` до `0`
- для каждой вершины делаем `sift_down`, меняя местами с минимальным из детей
- каждый обмен записывается в список swaps
- индексы выводятся 0-based

```
def sift_down(a, i, n, swaps):
    while True:
        min_i = i
        l = 2 * i + 1
        r = 2 * i + 2

        if l < n and a[l] < a[min_i]:
            min_i = l
        if r < n and a[r] < a[min_i]:
            min_i = r

        if min_i == i:
            break

        a[i], a[min_i] = a[min_i], a[i]
        swaps.append((i, min_i))
        i = min_i


def main():
    with open("n1/input.txt", "rb") as f:
        data = f.read().split()

    n = int(data[0])
    a = list(map(int, data[1:1 + n]))

    swaps = []
```

```
    for i in range(n // 2 - 1, -1, -1):
        sift_down(a, i, n, swaps)

    out_lines = [str(len(swaps))]
    out_lines.extend(f"{i} {j}" for i, j in swaps)

    with open("n1/output.txt", "w", encoding="utf-8") as f:
        f.write("\n".join(out_lines))


if __name__ == "__main__":
    main()
```

Проверено на нескольких input-файлах. Вывод совпадает с ожидаемым.

Задание 5 - Очередь с приоритетами (планировщик заданий)

Дано `n` потоков и `m` заданий с длительностями. Нужно для каждого задания вывести:

- номер потока, который его возьмет
- время старта этого задания

Идея - имитируем события "поток освободился" через очередь с приоритетами:

- для каждого потока храним `(next_free_time, thread_id)`
- приоритет меньше, если меньше `next_free_time`, а при равенстве меньше `thread_id`
- берем вершину кучи - это поток, который освободится раньше всех (если одновременно несколько - по минимальному id)
- выводим его `thread_id` и `next_free_time`, затем обновляем `next_free_time += duration` и просеиваем вниз

Куча реализована вручную (без `collections.deque` и без готовых приоритетных очередей).

```
class MinHeap:
    def __init__(self, arr):
        self.a = arr
        self.n = len(arr)
        for i in range(self.n // 2 - 1, -1, -1):
            self._sift_down(i)

    @staticmethod
    def _less(x, y):
        if x[0] != y[0]:
            return x[0] < y[0]
        return x[1] < y[1]

    def _sift_down(self, i):
        a = self.a
        n = self.n
        while True:
```

```
            min_i = i
            l = 2 * i + 1
            r = 2 * i + 2

            if l < n and self._less(a[l], a[min_i]):
                min_i = l
            if r < n and self._less(a[r], a[min_i]):
                min_i = r

            if min_i == i:
                break

            a[i], a[min_i] = a[min_i], a[i]
            i = min_i

    def top(self):
        return self.a[0]

    def replace_top(self, item):
        self.a[0] = item
        self._sift_down(0)


def main():
    with open("n2/input.txt", "rb") as f:
        data = f.read().split()

    n = int(data[0])
    m = int(data[1])
    times = list(map(int, data[2:2 + m]))

    heap = MinHeap([(0, i) for i in range(n)])

    out = []
    for t in times:
        free_time, thread_id = heap.top()
        out.append(f"{thread_id} {free_time}")
        heap.replace_top((free_time + t, thread_id))

    with open("n2/output.txt", "w", encoding="utf-8") as f:
        f.write("\n".join(out))


if __name__ == "__main__":
    main()
```

Проверено на input-файлах из папки `n2`, а также на нескольких вариантах, где:

- все задания нулевой/малой длительности
- одновременно освобождаются несколько потоков
- `n > m` и `n < m`

Во всех проверках порядок потоков и времена старта получаются корректными.