Αναφορά Εργασίας Running Median

Γεώργιος Παπουτσάς

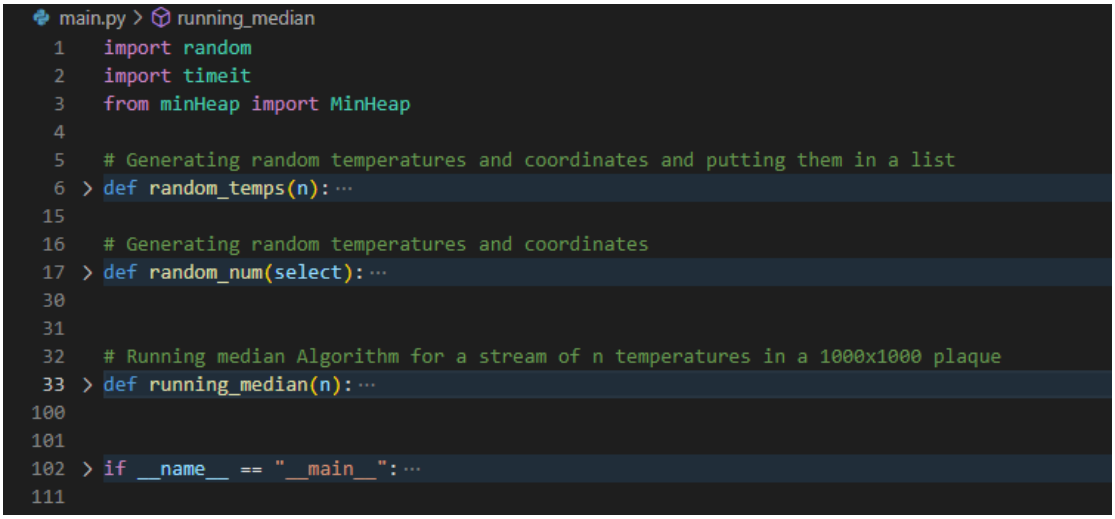
ΑΜ: 1083738

3ο έτος

Ο κώδικας απαρτίζεται από δύο αρχεία την main.py που περιέχει το κύριο πρόγραμμα και την υλοποίηση του αλγορίθμου και το minHeap.py που περιέχει την heap δομή που χρησιμοποιήθηκε. Το minHeap.py υπήρχε στο eclass και η χρήση του επιτράπηκε και για αυτό το λόγο δεν θα αναλυθεί στην αναφορά.

Το main.py:

Overview:



Η random_temps(n) και η random_num(select) δημιουργούν τυχαίες θερμοκρασίες σε τυχαίες συντεταγμένες. Πιο συγκεκριμένα:

```python
# Generating random temperatures and coordinates and putting them in a list
def random_temps(n):
    temps = []
    coords = []
    random.seed(1083738)
    for i in range(n):
        temps.append(random_num("temp"))
    for i in range(n):
        coords.append(random_num("coord"))

    with open("temps.txt", "w") as f:
        for i in range(n):
            f.write(f"{coords[i]} {temps[i]}\n")
    return temps, coords

# Generating random temperatures and coordinates
def random_num(select):

    # match select:
    #     case "temp":
    #         return float(f"{random.uniform(-30.00, 60.00):.2f}")
    #     case "coord":
    #         return tuple((random.randint(0, 999), random.randint(0, 999)))

    # For older versions of Python that don't have match case syntax
    if select == "temp":
        return float(f"{random.uniform(-30.00, 60.00):.2f}")
    elif select == "coord":
        return  tuple((random.randint(0, 999), random.randint(0, 999)))
```

Ο αλγόριθμος running median χρησιμοποιώντας ένα
min heap και ένα max heap με τα απαραίτητα σχόλια

```python
# Running median Algorithm for a stream of n temperatures in a 1000x1000 plaque
def running_median(n):
    lst = []

    min_h = MinHeap() # min heap
    max_h = MinHeap() # max heap with negation of min heap
    med = 0
    # temps, coords = random_temps(n)

    t1 = timeit.default_timer()
    for i in range(n):
        # Generating random temperature at a specific coordinate
        coord = random_num("coord")
        temp = random_num("temp")

        # Getting the temperatures from a list where the temperatures have already been generated
        # coord = coords[i]
        # temp = temps[i]

        # Check if the coordinate is already in either heap and update the temperature if so
        if max_h.isInMinHeap(coord):
            # max_h.changeKey((coord, -temp))
            max_h.deleteKey((coord, 0))

        elif min_h.isInMinHeap(coord):
            # min_h.changeKey((coord, temp))
            min_h.deleteKey((coord, 0))


        # For each new temperature, insert it into one of the heaps based on its value
        # -If the temperature is less than the median, insert it into the max heap
        # -If the temperature is greater than the median, insert it into the min heap
        if temp < med:
            max_h.insert((coord, -temp))
        else:
            min_h.insert((coord, temp))

        # Rebalance the heaps to ensure that the difference in size is at most 1
        # -If the number of elements in the  max-heap is more than one greater than the min-heap, remove the root
        # element of the max-heap and insert it into the min-heap
        # -If the number of elements in the min-heap is more than one greater than the max-heap, remove the root
        # element of the min-heap and insert it into the max-heap
        if max_h.size - min_h.size > 1:
            r = max_h.extractMin()
            min_h.insert((r[0], -r[1]))

        if min_h.size - max_h.size > 1:
            r = min_h.extractMin()
            max_h.insert((r[0], -r[1]))


        # Compute the median based on the root elements of the two heaps
        # -If the heaps are of equal size, the median is the average of the two root elements
        # -Otherwise, the median is the root element of the larger heap
        if max_h.size == min_h.size:
            med = (min_h.getMin()[1] - max_h.getMin()[1]) / 2
        elif max_h.size > min_h.size:
            med = -max_h.getMin()[1]
        else:
            med = min_h.getMin()[1]

        # Optional displaying for smaller sets of data
        # print(f"-----------{i}--------------")
        # print("min heap: ", end=" ")
        # min_h.display()
        # print("max heap: ", end=" ")
        # max_h.display()
        # print(f"median : {med}")
        # print("-------------------------\n")

    # Optional part for recording the temperatures in a file
        lst.append(med)
    with open("output.txt", "w") as f:
        for i in range(len(lst)):
            f.write(f"{lst[i]:.2f}\n")


    print(f"Median of {n} random temperatures : {med}")
    print(f"Time to execute : {timeit.default_timer() - t1:.2f}")
```

Η main:

```python
if __name__ == "__main__":
    try:
        n1 = 250_000            #
        n2 = 500_000            # Varying amounts of temperature readings
        n3 = 1_000_000          #
        random.seed(1083738)
        running_median(n1)
    except KeyboardInterrupt:
        quit()

```

Αποτελέσματα για 250_000, 500_000 και 1_000_000 όταν το πρόγραμμα τρέξει σε υπολογιστή στο ΚΥΠΕΣ:

```
Median of 250000 random temperatures : 15.02
Time to execute : 3.62
```

```
Median of 500000 random temperatures : 15.0
Time to execute : 8.68
```

```
Median of 1000000 random temperatures : 15.02
Time to execute : 23.58
```