Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Автоматизованих Систем Обробки Інформації та Управління

Лабораторна робота № 6

з дисципліни «Теорія алгоритмів»

на тему:

**" Піраміди”**

Виконала:

студентка гр. ІС-02

Павлущенко Ольга

Викладач:

Новікова П. А.

Київ – 2020

**Завдання**

В даній роботі необхідно розв'язати наступну задачу визначення послідовності медіан для заданого вхідного масиву. Нагадаємо, що медіаною для масиву називається елемент, який займає середнє положення у відсортованому масиві. Так, якщо кількість елементів у масиві непарна, то медіана одна та індекс її у відсортованому масиві визначається як [n/2] (де n — розмір вхідного масиву). Якщо кількість елементів у масиві парна, то медіан буде дві та їх індекси визначаються за формулами [n/2] та [n/2] + 1.

Задача формулюється наступним чином. Нехай заданий вхідний масив A = [x1, ..., xN]. Припустимо, що елементи масиву поступають на вхід програми послідовно: в кожний момент часу розглядається новий елемент xi. Необхідно для кожного i (від 1 до N) визначити медіану підмасиву A' = [x1, ..., xi], тобто медіану для масиву елементів, які були отримані програмою на даний момент часу. Необхідно розв’язати цю задачу, використовуючі структури даних пірамід і так, щоб кожна медіана визначалась за час O(log(i)).

**Програмний код**

def main():

H\_low =[]

H\_high = []

Input=[]

input("input\_10.txt",Input)

sort\_input(Input,"is02\_OlhaPavlushchenko\_06\_output.txt",H\_low,H\_high) #запис у файл

def input(file\_name,Input): # зчитування файлу

file = open(file\_name, 'r')

k = 0

lines = file.readlines()

for str in lines:

if k == 0:

n = int(str)

k = 1

else:

Input.append(int(str))

file.close()

def sort\_input(A, filename,H\_low,H\_high):

file = open(filename, 'w')

n = len(A)

for i in range(0, n):

AddElement(A[i],H\_low,H\_high)

file.write(get\_median(H\_low,H\_high))

file.close()

def Left(i): #визначення індексу лівого нащадка елементу

return i\*2

def Right(i): #визначення індексу правого нащадка елементу

return i\*2 + 1

def MaxHeapify(A,i):

p = Left(i)-1

q = Right(i)-1

if p < len(A) and A[p] > A[i-1]: #якщо введений індекс лежить у межах піраміди і значення більше, ніж вхідне

largest = p

else:

largest = i-1

if q < len(A) and A[q]>A[largest]: #якщо введений індекс лежить у межах піраміди і значення більше, ніж попереднє

largest = q

if largest != i-1: #якщо індекс не дорівнює вхідному

A[i-1],A[largest] = A[largest],A[i-1] #зміна елементів місцями

MaxHeapify(A, largest) #повторний виклик процедури

def HeapExtractMax(A):

if len(A) > 0: #якщо піраміда не порожня

max = A[0] #визначення найбільшого елементу

A.pop(0) #видалення найбільшого елементу

return max

def MinHeapify(A,i):

p = Left(i)-1

q = Right(i)-1

if p < len(A) and A[p] < A[i-1]:

lowest = p

else:

lowest = i-1

if q < len(A) and A[q] < A[lowest]:

lowest = q

if lowest != i-1:

A[i-1],A[lowest] = A[lowest],A[i-1]

MinHeapify(A, lowest)

def HeapExtractMin(A):

if len(A) > 0:

min = A[0] #мінімальне значення з неспадної піраміди

A.pop(0) #видалення найменшого елементу

return min

def BuildMaxHeap(A):

heap\_size = len(A)

for i in range(heap\_size//2, 0, -1):

MaxHeapify(A, i)

def BuildMinHeap(A):

heap\_size = len(A)

for i in range(heap\_size//2, 0, -1):

MinHeapify(A, i)

def AddElement(x,H\_low,H\_high):

if (len(H\_low) > 0 and H\_low[0] > x) or len(H\_low) == 0:

H\_low.append(x)

BuildMaxHeap(H\_low)

else:

H\_high.append(x)

BuildMinHeap(H\_high)

if len(H\_low) - len(H\_high) == 2: #якщо розмір H\_low став більшим на 2 за розмір H\_high

max = HeapExtractMax(H\_low) #взяти найбільний з H\_low елемент

H\_high.insert(0, max) #вставити в іншу піраміду

BuildMaxHeap(H\_low)

BuildMinHeap(H\_high)

else:

if len(H\_high) - len(H\_low) == 2: #якщо розмір H\_high став більшим на 2 за розмір H\_low

min = HeapExtractMax(H\_low) #взяти найменший з H\_high елемент

H\_low.insert(0, min) #вставити в іншу піраміду

BuildMinHeap(H\_high)

BuildMaxHeap(H\_low)

def get\_median(H\_low,H\_high):

if (len(H\_low) - len(H\_high)) % 2 == 0: #якщо загальна кількість елементів парна

result = str(H\_low[0]) + " " + str(H\_high[0]) #повернути найбільший з H\_low і найменший з H\_high

else:

if len(H\_low) > len(H\_high): #якщо елементів в H\_low більше, ніж у H\_high

result = str(H\_low[0]) #повернути найбільший елемент з цієї піраміди

else:

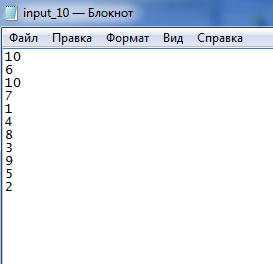
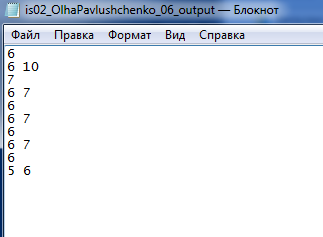
result = str(H\_high[0]) #повернути найменший елемент з цієї піраміди

result += '\n'

return result

main()

**Результати**

**Висновок**

В результаті виконання даної лабораторної роботи було досліджено будову та функціонування пірамід, незростаючі і неспадні піраміди.

Послідовність медіан була знайдена за допомогою за допомогою структури даних – піраміди. Було створено дві різних піраміди: неспадна, що містила елементи меншої половини масиву, і незростаюча, яка містила елементи більшої половини масиву.

Завдяки підтримці властивостей цих пірамід, процедури виконання яких викликалися при будь-яких змінах рекурсивно, останній елемент меншої половини і перший елемент другої половини завжди залишалися кореневими вузлами пірамід, що забезпечувало легкий та швидкий доступ до них. Під час кожної ітерації поточний елемент порівнювався з цими елементами і відповідно додавався до однієї з пірамід. Після цього виконувалась підтримка інваріанту цикла, що зберігала різницю у довжинах пірамід не більше за 1, а отже і розташування медіан.