Pontificia Universidad Católica Madre y Maestra
Facultad de Ciencias de la Ingeniería

Tarea #1:

Corte Minimo (Kager)

Presentado Por:

Angel González (2013-0157)

Materia:

Diseno y Analisis de Algoritmos

Profesor:

Juan Núñez.

Marco Teorico

> En informática, un montón es una estructura de datos basada en árbol especializado que satisface la propiedad montón: Si A es un nodo padre de B, entonces la llave (el valor) de nodo A se ordena con respecto a la clave de nodo B con el mismo orden de aplicar en todo el montón. Un montón pueden ser clasificados ya sea como un "máximo de almacenamiento dinámico" o un "min montón". En un montón max, las claves de nodos padre son siempre mayores que o iguales a las de los niños y la tecla más alta se encuentra en el nodo raíz. En un montón min, las claves de nodos padre son menores o iguales a las de los niños y la tecla más baja está en el nodo raíz. Montones son cruciales en varios algoritmos de grafos eficientes, como el algoritmo de Dijkstra, y en el heapsort algoritmo de clasificación. Una implementación común de un montón es el montón binario, en el que el árbol es un árbol binario completo (ver figura).

En un montón, el elemento más alto (o más bajo) prioridad siempre se almacena en la raíz. Un montón no es una estructura ordenada y puede ser considerada como parcialmente ordenado. Como es visible desde el montón-diagrama, no existe una relación particular entre los nodos en cualquier nivel, incluso entre los hermanos. Cuando un montón es un árbol binario completo, que tiene una altura más pequeña posible, un montón con N nodos siempre tiene log N altura. Un montón es una estructura de datos útil cuando se necesita para eliminar el objeto con la prioridad más alta (o más baja).

Codigo

```
from heapq import heappush as push, heappop as pop
import time


def medians(numbers):
    numbers = iter(numbers)
    less, more = [], []
    first = next(numbers)
    yield first
    second = next(numbers)
    push(less, - min(first, second))
    push(more, max(first, second))
    while True:
        current = (more[0] if len(less) < len(more)
                   else - less[0] if len(less) > len(more)
        else (more[0] - less[0]) / 2)
        yield current
        number = next(numbers)
        if number <= current:
            push(less, - number)
        else:
            push(more, number)
        small, big = ((less, more) if len(less) <= len(more)
                      else (more, less))
        if len(big) - len(small) > 1: push(small, - pop(big))


if __name__ == '__main__':
        start_time = time.time()
        f = open("prob_median.txt", 'r')
        line_list = f.readlines()
        lst = [int(elem) for elem in line_list]
        print(tuple(medians(lst)))
        print("--- %s seconds ---" % (time.time() - start_time))
```

Corrida

(6331, 4562, 2793, 4562, 2793, 2216, 2793, 2548, 2793, 2548, 2793, 3542, 4292, 4385, 4479, 5082, 5147, 4813, 4479, 4813, 4479, 4385, 4479, 4385, 4292, 4213, 4135, 4213, 4292, 4213, 4292, 4213, 4135, 3873, 3611, 3545, 3480, 3136, 3480, 3545, 3611, 3873, 4135, 3873, 3611, 3545, 3611, 3873, 4135, 3873, 3611, 3545, 3611, 3545, 3611, 3545, 3555, 3517, 3555, 3583, 3611, 3583, 3555, 3530, 3555, 3583, 3555, 3530, 3555, 3583, 3611, 3873, 3611, 3873, 3719, 3927, 4135, 4158, 4181, 4158, 4135, 3927, 3719, 3927, 4135, 4023, 3912, 4023, 3912, 4023, 3912, 3922, 3912, 3922, 3912, 3922, 3912, 3815, 3912, 3922, 3912, 3922, 3932, 4033, 3932, 3922, 3932, 4033, 3932, 3922, 3932, 3922, 3932, 3922.....5001, 5001, 5001, 5001, 5001, 5001, 5002, 5001, 5002, 5001, 5001, 5000, 5001, 5000, 5001, 5000, 5001, 5001, 5002, 5002, 5002, 5001, 5001, 5000, 5000, 5000)

--- 0.0552470684052 seconds ---