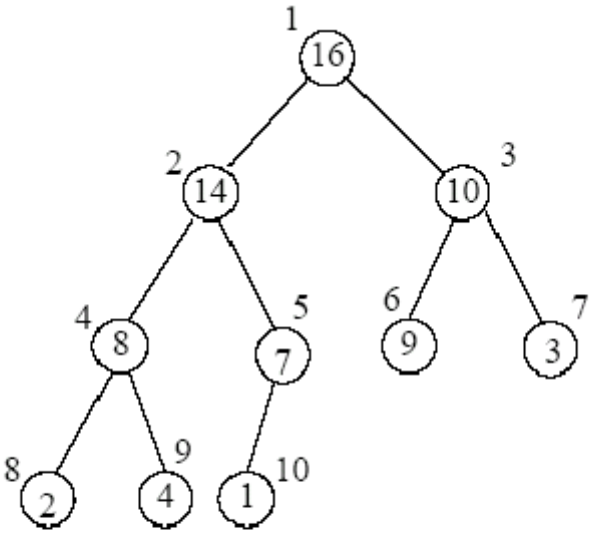
ESTRUCTURA DE DADES:PRÀCTICA 4
HEAPS



1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

Universitat De Barcelona

Curs 2017-2018

Pau Bernabé Constans

NIUB: 20081736

Pablo Laiz

EXERCICI 1

En aquest exercici havíem de fer les operacions bàsiques del Heap:

`constructor`: construeix el Heap buit
`size`: retorna el nombre de nodes que hi ha en el Heap
`empty`: retorna cert si el Heap esta buit, fals en cas contrari

`insert`: afegeix un nou element al Heap. Aquesta funcio rep la clau i el valor/s d'aquesta clau. En aquest mètode he fet la crida d'un mètode creat per mi. S'anomena upheap i funciona de manera que si tenim el heap amb nodes, es cridarà i farà l'ordrenació en el heap de l'últim node inserit.

`min`: retorna la clau minima del Heap. En aquest he decidit retornar el index 0 del heap, ja que com que ordena, òbviament el mínim és heap[0].

`minValues`: retorna els valors de la clau minima del Heap. He fet el toString i retorna el valor del mínim.

`removeMin`: elimina el node minim del Heap. En aquest també he creat un mètode anomenat downheap. Es crida després de fer el swap entre el primer-últim i esborrar-lo(últim). Segueix la mateixa dinàmica que l'upheap però com diu el nom, en un sentit invers.

`printHeap`: imprimeix per consola tot el Heap. Fent un recorregut imprimir els nodes en ordre
`search:` Introduint una clau, aquest busca en el vector i si la troba, retorna el toString del valor desitjat.

Decidiu vosaltres el TAD per representar dels nodes del Heap i justifiqueu la vostra decisio a la memòria.
He utilitzat el TADMovie, ja que en el 2 haviem d'utilitzar-lo explícitament i així podia aprofitar gran part de codi.

EXERCICI 2

En aquest exercici havíem d'implementar el HeapMovieFinder, una versió en heap del BSTMovieFinder.

• `appendMovies(filename)`: Aquest metode rep el nom d'un fitxer i emmagatzema el seu contingut a una estructura de dades. Utilitzava el mètode que
• `insertMovie(movieID, title, rating)`: Aquest metode rep les dades d'una pel・licula i fa la insercio a l' estructura de dades corresponent.
• `showMovie(movieID)`: Aquest metode rep un `movieID` i retorna un sol string amb les dades associades a la pel・licula.
• `findMovie(movieID)`: Aquest metode mostra per pantalla el titol de `movieID`, junt amb tota la informacio de la pel·licula.

Els mètodes són els mateixos però hem d'implementar-los amb el Heap, la majoria m'hi he fixat i el appendmovies, insertmovie els he aprofitat totalment. Només el showMovie i el Findmovie han variat respectivament al anterior.

Un mètode meu era el adquirir la profunditat, bàsicament he fet un upheap que situava el fill a la posició última del vector i anava pujant fins que aquest arribava a 0(primera posició). Mentres anava comptant.

En el mètode de llegir i comparar el heap amb la cerca de pel·lícules he aprofitat el search i daquesta manera ens mostra les repetides com a vector i el nombre de repetides.

He utilitzat el remove min per a fer la impressió de les pel·lícules, perquè cada cop el que farem és treure el mínim i d'aquesta manera imprimir-lo i tenir una llista completament ordenada de forma creixent.

Si ho fèiem amb el print de l'exercici 1 i un recorregut el que passava era que les pel·lícules sortien un ordre no creixent, sinó que ho imprimia com a un arbre HEAP.

EXERCICI 3: AVALUACIÓ DE LES ESTRUCTURES

1. Compteu el temps de generació de l'estructura per les dos llistes de pel·licules de diferent grandària. Feu les dues proves en el mateix ordinador i en condicions similars.

En el fitxer de les pel·lícules de mida petita em dona un temps de:

Elapsed time: 0.001994 s

En el fitxer de les pel·lícules de mida gran em dona un temps de:

Elapsed time: 0.124942 s

2. Compteu el temps d'acces (cerca) de les pel· licules de `cercaPelicules.txt` amb el heap inicialitzat en base als dos fitxers donats.

En el fitxer de les pel·lícules de mida petita em dona un temps de cerca de:

Elapsed time: 0.045355

En el fitxer de les pel·lícules de mida gran em dona un temps de cerca de:

Elapsed time: 1.68511

Raoneu els resultats de temps obtinguts.

És obvi que els fitxers grans tindran un major cost degut al seu tamany que es aproximadament 10 vegades el petit.

Indiqueu quin es el cost computacional teoric de les operacions d'insercio i cerca en el heap.

El cost d'inserir un element en un vector és O(1), però hem de tenir en compte que hem de fer upheap per a cada inserció, així que el cost total és de O(logn).

El cost de cerca en el heap, en el meu cas és de O(n), ja que recorrem tot el vector fins a trobar l'element desitjat.