UNIVERSITAT DE LLEIDA

Escola Politècnica Superior

Grau en Enginyeria Informàtica

Estructures de dades

PRÀCTICA 4:
Heap Queue

Marc Cervera Rosell 47980320C

Diego Martínez de Sanjuan 48051017S

Data de lliurament: dia 07 de Desembre de 2021.

INDEX

1. Funcions

1.1. add

Després de crear un Triplet amb els paràmetres de la funció i el timeStamp corresponent, afegeix el Triplet al final de l'ArrayList i incrementa el timeStamp per cada operació d'inserció i extracció que fa. Això servirà per diferenciar les prioritats iguals.

A partir de l'últim element va comprovant que tingui pare i que sigui de prioritat inferior. Per comparar dos Triplet utilitza el mètode compareTo (expliucat més endavant). En cas de ser superior, s'intercanviaràn.

Per fer aquest intercanvi, fa ús d'un mètode auxiliar anomenat swap.

Cal observar que el compareTo compara dues instàncies de Triplet. Perquè pugui realitzar bé la comparació, també implementa un altre mètode auxiliar anomenat getParent, amb un funcionament idèntic al mètode estàtic parent ja donat, però en comptes de retornar l'índex del pare, retorna el Triplet del pare.

1.2. remove

La funció remove, guarda el primer valor de l'array (el més prioritari) en una instància Triplet que retornarà al final de la funció i es col·loca l'últim element de l'array a la primera posició. Aquí és quan és crida a la funció recursiva removeCheck per organitzar l'array i col·locar aquest element a la seva posició correcta.

La funció removeCheck comença mirant si l'element que li passen té fill esquerra, si en té, el guarda en una instància anomenada biggestTriplet. Després mira si té fill dret, i si en té, compara el fill esquerra amb el dret. Si el dret és superior, és associat a la variable biggestTriplet fent que aquesta variable solament contingui el fill més gran. Una vegada tenim el fill més gran, comparem el pare (element que ens han passat) amb el fill més gran i si el pare és inferior a ell s'intercanvien posicions amb el swap mencionat anteriorment i es torna a cridar a la funció.

1.3. compareTo

1. Tracta el cas de les prioritats null. Si una prioritat és null, l'altra per força serà superior (i viceversa).
2. Compara prioritats diferents amb el mètode equals.
3. Compara prioritats iguals i tracta els timeStamp de cada Triplet. Retornarà el timeStamp de menys valor, és a dir, el que ha arribat primer.

1.4. Size

Retorna el nombre d'elements de l'ArrayList. Utilitza el mètode size() ja implementat.

El fet de restar 1 en el valor de retorn, és degut al fet que la primera posició de l'ArrayList no s'usa. A causa del seu desús, s'ha pres la decisió de disseny d'establir la posició amb índex 0 de l'ArrayList amb un valor i una prioritat igual a null.

1.5. Element

Tracta l'excepció i retorna el valor corresponent V del primer Triplet de l'ArrayList (posició 1). Aquest mètode no cal que garanteixi que sigui max-heap, d'això se n'asseguren els mètodes d'inserció i extracció.

2. Tests

S'ha implementat una classe de test per cada tipus de test, és a dir, s'ha implementat una classe pels tests del add(), una per als tests del remove(), una per als tests del element() i una per als tests del size().

Per al test del add(), s'han realitzat els casos explicats en el compareTo, per comprovar que aquest últim funciona correctament. S'ha afegit una comparació de prioritat null amb una altra prioritat null.

A més, a l'apartat 1) del compareTo, s'ha afegit un test amb prioritats desordenades per corroborar que el mètode add reordena correctament.

Per al test del remove(), inicialment s'omple la cua amb elements amb un valor i una prioritat, després es testeja que quan es crida la funció remove retorna el element amb mes prioritat i s'ordena l'array de manera correcta.