Manel Carrillo Maíllo (1633426)
1r de MatCad
Examen Final

2. a PSEUDOCODI.

1. FUNCIÓ ESBUIDA.

Aquesta funció rep la Cua que conté el binary heap com a paràmetre i el seu objectiu és saber si la cua és buida o no.

Per saber-ho, mirem la variable nlevels de la seva estructura que indica la quantitat de nivells que té el binary heap de manera que si aquest és 0, vol dir que no hi ha cap node a la cua i ho fem saber retornant un int, en el meu cas un 1, 0.

2. FUNCIÓ DESENCUA.

Aquesta funció rep com a paràmetres la Cua que conté el binary heap i l'estructura noderouting status que conté el cost.

L'objectiu d'aquesta funció és substituir el node arrel del binary heap i canviar-lo amb l'últim node, esborrant aquest últim tenint en compte diferents situacions per poder mantenir la propietat de forma.

Per començar s'ha localitzat el valor dels dos nodes igualant dues variables a les posicions dels nodes a la cua , pel cas de l'arrel level 0 i índex 0 i en cas de l'últim utilitzant l'estructura cua amb els nlevels i last_level_size.

En segon lloc, s'ha de posar el valor del node de la última posició a l'arrel igualant la cua a la posició 0 0 amb la variable de l'últim nivell que hem guardat abans.

Ara analitzem les possibles situacions, hem de tindre en compte la situació on traiem l'últim element d'un nivell, de manera que no només hem de calcular els nodes del nivell i igualar a last_level_size i reduir un level nlevels (primer redueixo, després calculo last level size).

Per veure si un nivell s'acaba en fer el desencua, comparem last level size de manera que si aquest és diferent d'1, es redueix una posició, si no, es redueix una posició i un nivell. Després de fer això, hem de veure si quan hem desencuat ens hem quedat només amb el node arrel o no de manera que si el nlevels és diferent de 1 que seria el de l'arrel, hem de fer un heapify down per actualitzar l'arbre amb el node últim que hem col·locat al principi

de manera que per a fer-ho creem una estructura element i li indiquem que el seu índex i level és 0, ja que és l'arrel.

3. FUNCIO REENCUA.

Aquesta funció rep com a paràmetres un node, l'estructura node-routing status que conté el cost i la cua binary heap.

Aquesta funció es fa amb l'objectiu de solucionar el cas en què s'hagi trobat un camí més curt i l'arbre hagi perdut la seva propietat heap, de manera que per a mentir-la hem de fer servir dues funcions, ja que no cal afegir cap node, perquè aquest es troba dins l'arbre.

Per poder mantenir l'arbre ben estructurat mantenint la propietat hem de fer un heapify up amb el node que s'ha actualitzat de manera que per trobar-ho dins l'arbre utilitzem una funció similar a la funció buscapunt que és look up que a través del node introduït per paràmetre et retorna l'element que es buscava.

4. FUNCIÓ ENCUA.

Aquesta funció rep el node que es vol incorporar a la cua, l'estructura node-routing status que conté el cost i la cua binary heap.

L'objectiu d'aquesta funció és afegir un node mantenint la propietat de forma tenint en compte si la cua és buida o no i tenint en compte si ha de crear un nou nivell o no pel fet que l'anterior estigui ple i finalment fer un heapify up per actualitzar la seva posició a l'arbre si fa falta.

Per començar el que fem és veure si la cua és buida o no, si la cua és buida aquest node entrat com a paràmetre es guarda a la posició 0 0 de la cua i s'actualitza el nivell i el nombre de nodes de l'arbre indicant que és 1.

Si la cua no és buida hem de diferenciar si hem de crear un nou nivell o no, de manera que comparem el last level size de l'arbre sense haver-hi encuat amb la quantitat total de nodes que pot tindre aquell nivell fent l'operació 2^nlevels-1.

Si són iguals, hem de crear un nou nivell i per a fer-ho hem de reservar memòria dels nodes que entraran en aquest nivell fent un malloc amb la quantitat de nivells nova (1 més que l'anterior) i actualitzem el nombre de nodes de l'últim nivell a 1 i el nlevels li sumem una unitat.

En canvi, si no hem de crear un nou nivell, guardem el node a la última nova posició i actualitzem només el last_level_size.

I per últim en qualsevol dels tres casos, fer un heapify up amb l'element del node que hem inserit**.**

5 FUNCIÓ LOOK UP.

Funció que rep com a paràmetre un node i la cua i el seu objectiu és el mateix que la funció buscapunt d'altres entregues que és trobar un node dintre de l'arbre.

En aquest cas hem de mirar tot l'arbre de forma piramidal, nivell a nivell fins a arribar al final.

Per a fer això es necessita dos fors, un que vagi nivell a nivell i un altre que vagi node a node dintre d'aquest nivell fins a descobrir-ho. Per si es dona el cas en què no es troba he afegit una cerca (tot i que crec que no caldria) per indicar que no s'ha trobat el node dintre de l'arbre.

Si es troba el node, guardem en una estructura element el nivell i el node on hem trobat el node.

6. FUNCIÓ FILL ESQUERRA

Aquesta funció rep un node o element el qual correspondrà al node pare del fill esquerre que volem trobar amb aquesta equació i també rep la cua.

En aquest cas hem de tindre en consideració el cas en què el pare del fill o l'element entrat com a paràmetre es troba a l'últim nivell de l'arbre de manera que no tindrà fills.

Si té fills, indiquem el nivell i la posició del fill, la posició és dos cops la del pare i el nivell és un més que el del pare.

7.FUNCIÓ FILL DRET

Aquesta funció rep un node o element el qual correspondrà al node pare del fill dret que volem trobar amb aquesta equació i també rep la cua.

En aquest cas hem de tindre en consideració el cas en què el pare del fill o l'element entrat com a paràmetre es troba a l'últim nivell de l'arbre de manera que no tindrà fills .

Si té fills, indiquem el nivell i la posició del fill, la posició és dos cops la del pare més una unitat i el nivell és un més que el del pare.

8. FUNCIÓ PARE

Aquesta funció rep un node o element el qual correspondrà al node fill del pare que volem trobar amb aquesta equació i també rep la cua.

En aquest cas hem de tindre en consideració el cas en què fill o l'element entrat com a paràmetre no sigui l'arrel, ja que aquest no té pare.

Si en té, el pare es troba la meitat de la posició del fill i a un nivell menys.

9.FUNCIÓ GETCOST

Aquesta funció rep com a paràmetre un node i l'estructura node-routing status que conté el cost.

Aquesta funció té l'objectiu de simplificar i fer més visual aquelles funcions que utilitzen el cost com pot ser el heapify up o el down.

Per poder retornar el cost del node, el que fem és accedir a nods el qual té el cost de cada element de la cua i retornar-ho.

10. FUNCIÓ HEAPIFY UP

Aquesta funció rep com a paràmetres un element de la cua, la cua binary heap i l'estructura node-routing status que conté el cost.

L'objectiu d'aquesta funció és mantindré les propietats i l'estructura d'un binary heap, fent que si el valor del node pare és més gran que el del fill s'intercanviïn. Això es fa de manera consecutiva fins que el pare sigui més petit que el fill, d'aquesta manera es manté les propietats.

Per a fer aquesta funció he creat una variable indicadora que el pare és més petit que el fill per poder parar el bucle d'actualitzacions. De manera que es fa un while que pararà en el moment en què aquesta variable canviï de valor. Dintre del while es va comparant el cost del pare i el fill (cridant a les funcions anteriors per trobar el cost i el pare del node entrat per paràmetre) de manera que si el cost del pare és més gran que el del fill, el node fill passa a la posició del pare i el node pare a la posició del fill i continuem mirant fins que tot estigui actualitzat.

11 FUNCIÓ HEAPIFY DOWN

Aquesta funció rep com a paràmetres un element de la cua, la cua binary heap i l'estructura node-routing status que conté el cost.

L'objectiu d'aquesta funció es mantindré les propietats i l'estructura d'un binary heap, fent que si el valor del node pare és més gran que el del fill s'intercanviïn. Això es fa de manera consecutiva fins que el pare sigui més petit que el fill, d'aquesta manera es manté les propietats.

Per a fer aquesta funció he creat una variable indicadora que el pare és més petit que el fill per poder parar el bucle d'actualitzacions. De manera que es fa un while que pararà en el moment en què aquesta variable canviï de valor.

Dintre del while tindrem en compte diferents situacions, la primer situació és el cas en què el pare és més gran que els dos fills, però el dret és més petit que el pare.

En aquesta situació el fill dret i el pare s'intercanvien. A la situació contrària on els fills són més petits que el pare, però el fill esquerre és més gran que el pare, aquests dos s'intercanvien.

Si per contra el pare només és més gran que el fill dret, aquests dos s'intercanvien i si és el fill esquerre l'únic fill més peti que el pare, aquests dos s'intercanvien.