4ο Σετ Ασκήσων

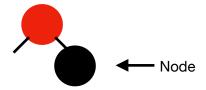
Άσκηση 1.α:

Έχουμε ένα δυαδικό δέντρο Τ για το οποίο γνωρίζουμε ότι, για κάθε κόμβο του, το μήκος του μακρύτερου μονοπατιού είναι από τον κόμβο σε ένα εξωτερικό φύλλο είναι το πολύ διπλάσιο του μήκους του συντομότερου μονοπατιού από αυτόν τον κόμβο σε ένα φύλλο. Αυτό σημαίνει ότι πληρεί τις προϋποθέσεις για να γίνει ένα Red-Black Tree. Και ο τρόπος που θα το χρωματίζαμε είναι ο εξής:

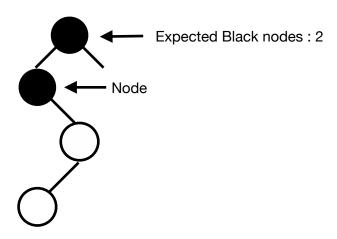
Αρχικά η ρίζα του δέντρου πρέπει να είναι μαύρη βάση των ιδιοτήτων άρα η ρίζα βάφεται μαύρη. Και περιμένουμε ότι πηγαίνοντας σε ένα φύλλο θα συναντήσουμε height(n)/2 μαυρα nodes.



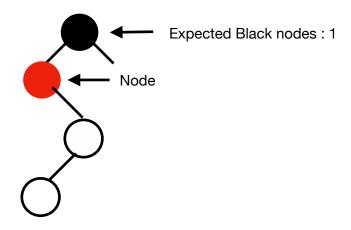
Στην συνέχεια ελέγχουμε αν ο πατέρας του node στο οποίο βρισκόμαστε είναι κόκκινος. Αν είναι τότε χρωματίζουμε το node μάυρο. Και συνεχίζοντας στο μονοπάτι αυτό περιμένουμε να βρουμε όσα μαύρα περίμενε και ο πατέρας αυτού του node.



Αν ο πατέρας του node που ειμαστε ειναι μαυρου χρωματος τοτε αν, min(n) (η συντομότερη απόσταση από το node που βρισκόμαστε σε ένα φύλλο) είναι ίση με τον αριθμό από μαύρα nodes που περιμενε ο πατερας αυτου του node τοτε το node αυτο θα χρωματιστεί μαύρο.



Αλλιώς αν, min(n) είναι μεγαλύτερο από τον αριθμό από μαύρα nodes που περιμενε ο πατερας αυτου του node τοτε το node αυτο θα χρωματιστεί κόκκινο.



Τέλος, ότι και να γίνει το node που βρισκόμαστε θα περιμένει να βρει όσα μαύρα nodes περιμενε ο πατερας του -1.

Άσκηση 1.β:

}

29	15	22	14	12	13	10	6	9	8	5	2	11	4

InitializeHeap(Table A[0..n-1]{

 $for(I=1;\ I< n;\ I++)\ Heapify(A[0..I]);$

I = 1 -> The leaves 29,15 don't have Childs so heapify makes no change.

20	15	22	14	10	10	10	6	0	0		2	44	1
29	10	22	14	12	10	10	O	9	0	S S		11	4

I = 2 -> The leaves 29,15,22 don't have Childs so heapify makes no change.

29	15	22	14	12	13	10	6	9	8	5	2	11	4

I = 3 -> The leaves 29,15,22,14 don't have Childs so heapify makes no change.

29	15	22	14	12	13	10	6	9	8	5	2	11	4
----	----	----	----	----	----	----	---	---	---	---	---	----	---

= 4 -	> The I	eaves	29,15,	ZZ, 14,	12 0011			0 00	Jupy			arigo.	
29	15	22	14	12	13	10	6	9	8	5	2	11	4
							_						
= 5 -	> The I	eaves	29,15,	22,14,	12,13 c	don't h	ave Ch	nilds so	heap	ify mal	kes no	chang	e.
29	15	22	14	12	13	10	6	9	8	5	2	11	4
= 6 -	> The I	eaves	29,15,	22,14,	12,13,1	10 don	't have	Childs	s so he	eapify r	nakes	no cha	ange.
29	15	22	14	12	13	10	6	9	8	5	2	11	4
hild 2 29	15 > Node 29 beca 15			•						•		•	4 vith it 4
= 9 -	> Node	e 8,9,6	, is the	only r	odes t	hat ha	ve chil	dren aı	nd Hea	apify de	oesn't	swap (3 with
ts chi	> Node ld 29 b i't swap	ecaus	e 6 < 2	9 and		't swap	9 with					•	
ts chi doesn 29 = 10 vith it and do becau	ld 29 b 1't swap 15 -> Noo s child oesn't use 5<1	de 5,8, 29 be swap 8	e 6 < 2 h its ch 14 9,6, is cause 6 3 with i 3.	9 and hild beautiful beaut	doesn' cause 8 13 ly node and do d beca	t swap 8<14, 6 10 es that besn't use 8<	9 with 8<12. 6 have of swap 9	9 childre 9 with i	8 n and ts child	5 Heapify d beca	9<22, 9 2 y does use 9< ap 5 w	9<15 a 11 n't swa 22, 9 ith its c	ap 6 15 child
ts chi doesn 29 = 10 vith it	ld 29 b i't swap 15 -> Noo is child oesn't	de 5,8, 29 bes	e 6 < 2 h its ch 14 9,6, is cause 6 3 with i	9 and hild beautiful beaut	doesn' cause 8 13 ly node and do	t swap 8<14, a 10 es that besn't	9 with 8<12. 6 have o	n its ch 9 childre 9 with i	8 n and tts chil	5 Heapify	9<22, 9 2 y does use 9<	9<15 a 11 n't swa	ap 6
ts chi doesn 29 = 10 vith it and do ecau 29 = 11 vith it	ld 29 b 1't swap 15 -> Noo s child oesn't use 5<1	de 5,8, 29 bees swap 8 0, 5<1 22 de 2,5, 29 bees swap 8	e 6 < 2 h its ch 14 9,6, is cause (3 with i 3. 14 8,9,6, i cause (3 with i	9 and hild bed 12 the on 6 < 29 ts child ts child ts child the confidence of the con	ly node and do beca	es that besn't use 8<	9 with 8<12. 6 have 6 swap 9 14, 8 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	ohildre 9 childre 9 with i 12 and 9 e childi 9 with i	8 n and ts child does 8 ren and ts child does	5 Heapify d becan't swa	y does use 9- ap 5 w 2 oify doe use 9- ap 5 w	9<15 a 11 n't swa <22, 9< ith its c 11 esn't st <22, 9<	ap 6 (15) child 4 wap (
ts chi doesn 29 = 10 with it and do 29 = 11 with it	15 15 -> Noos child oesn't se 5<1 15 -> Noos child oesn't se child oesn't se child oesn't	de 5,8, 29 bees swap 8 0, 5<1 22 de 2,5, 29 bees swap 8	e 6 < 2 h its ch 14 9,6, is cause (3 with i 3. 14 8,9,6, i cause (3 with i	9 and hild bed 12 the on 6 < 29 ts child ts child ts child the confidence of the con	ly node and do beca	es that besn't use 8<	9 with 8<12. 6 have 6 swap 9 14, 8 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	ohildre 9 childre 9 with i 12 and 9 e childi 9 with i	8 n and ts child does 8 ren and ts child does	5 Heapify d becan't swa	y does use 9- ap 5 w 2 oify doe use 9- ap 5 w	9<15 a 11 n't swa <22, 9< ith its c 11 esn't st <22, 9<	ap 6 (15) child 4 wap (
ts chi doesn 29 = 10 vith it and doecau 29 = 11 vith it and doecau 29 = 12 vith it and doecau	ld 29 b i't swap 15 -> Noo s child oesn't ise 5<1 -> Noo s child oesn't ise 5<1	de 5,8, 29 bees swap 8 0, 5<1 22 de 2,5, 29 bees swap 8 0, 5<1 22 de 2,5, 29 bees swap 8 0, 5<1 22	e 6 < 2 h its ch 14 9,6, is cause (3 with i 3. 14 8,9,6, i cause (3 with i 3 and 14 8,9,6, i cause (3 with i 3 and 14	9 and hild bed 12 the on 6 < 29 ts child doesn 12 s the 6 < 29 ts child doesn ts child	doesn' cause 8 13 ly node and do beca 't swap 13 only no and do beca 't swap 13 only no and do beca 't swap 13	t swap 8<14, 8 10 es that besn't use 8< 10 des th besn't use 8< 2 with 10 des th besn't use 8<	o 9 with 8<12. 6 have 6 swap 9 14, 8<6 14, 8<6 at have 8 14, 8<6 at have 9 14, 8<6 a	children with in child bed	8 n and ts child does ts child does ause ts child does ts child does	bause 9 5 Heapify d becan't swa 2 < 6, 2 < 5 d Heap d becan't swa 2 < 6, 2 < 5	y does y does use 9- ap 5 w 2 oify doe use 9- ap 5 w 2	9<15 a 11 n't swa <22, 9< ith its o 11 esn't si <22, 9< ith its o ith its o ith its o	ap 6 15 child 4 wap 15 child

I = 13 -> Node 4 > 2 so when Heapify is called it swaps 4 with 2.

Άσκηση 2.α:

Το μεγαλύτερο πλήθος εσωτερικών κόμβων σε ένα κοκκινόμαυρο δέντρο με μαύρο ύψος κ είναι 2^(2κ) - 1. Σε αυτήν την περίπτωση στο μακρύτερο μονοπάτι οι κόμβοι πάνε εναλαξ μαύροι και κόκκινοι και ο καθένας έχει παιδιά τα οποία κάνουν το ίδιο.

Το μικρότερο πλήθος εσωτερικών κόμβων σε ένα κοκκινόμαυρο δέντρο με μαύρο ύψος κ είναι 2^κ - 1. Σε αυτήν την περίπτωση στο μικρότερο μονοπάτι οι κόμβοι είναι όλοι μαύροι.

Άσκηση 2.β:

Το μεγαλύτερο πλήθος κόμβων σε ένα σωρό με ύψος h είναι 2^h -1 καθώς ένα τέτοιο δέντρο είναι ένα πλήρες δέντρο(συμπληρώνεται από αριστερά προς τα δεξιά) και άρα θα καταλήξει να είναι τέλειο.

Το μικρότερο πλήθος κόμβων σε ένα σωρό ύψους h είναι 2^(h-1) καθώς ένα τέτοιο δέντρο θα ήταν ένα τέλειο δέντρο ύψους h-1 και θα είχε ο αριστερότερος κόμβος ένα αριστερό παιδί ώστε να είναι το δέντρο ύψους h.

Άσκηση 2.γ:

Ένας σωρός με 256 κλειδιά θα αντιπροσωπεύει ένα τέλειο δέντρο ύψους 8 του οποίου το αριστερότερο παιδί έχει ένα αριστερό παιδί. Σε αυτήν την σωρό σίγουρα το μικρότερο κλειδί βρίσκεται στην θέση 0, άρα οι πιθανές θέσεις για το 3 μικρότερο κλειδί είναι στην θέση 1 ή 2 του heap.

Δεν μπορούμε να είμαστε τόσο ακριβείς στο που είναι το μεγαλύτερο κλειδί αλλά μπορούμε να είμαστε σίγουροι ότι είναι ένα εσωτερικό φύλλο εξαιτίας τησ ιδιότητας της σωρού ότι οι πατρικοί κόμβοι πρέπει να είναι μικρότεροι ίσοι από τα παιδιά τους. Άρα οι πιθανές θέσεις θα είναι από 192 - 255.

Άσκηση 3:

```
parent(int m) return (m + 4) / 8 * 4 - 4;
leftchild(int m) return 2m-n;
rightchild(int m) return 2m-n-1;
child(Table H, int m) return H[leftchild(m)] > H[rightchild(m)]? H[leftchild(m)]: H[rightchild(m)];
Void checkParentHeap(Table H, int j){
        node current = H[ j ];
       while(true){
               if (current.parent == nill || current.parent.key < current.key){
                       return;
               swap(H, j, parent( j ));
               current = H[ parent( j )];
       }
}
Void checkChildHeap(Table H, int j){
        node root = H[0];
        while(true){
               if(child(j) == nill || H[j].key < child(j).key){
                       return;
               swap(H, j, child( j ));
               root= H[ child( j ) ];
       }
}
Void HeapSwapElement( Table H, int j, int k){
        swap(H, j, k);
        if( H[ j ].parent == nill ){
               checkChildHeap(H, j);
        }else checkParentHeap(H, j);
        if( H[k].parent == nill ){
               checkParentHeap(H, k);
        }else checkChildHeap(H, k);
}
```