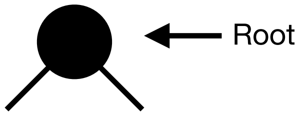
4ο Σετ Ασκήσων

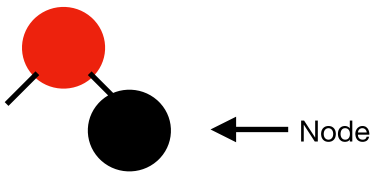
Άσκηση 1.α:

Έχουμε ένα δυαδικό δέντρο T 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 :

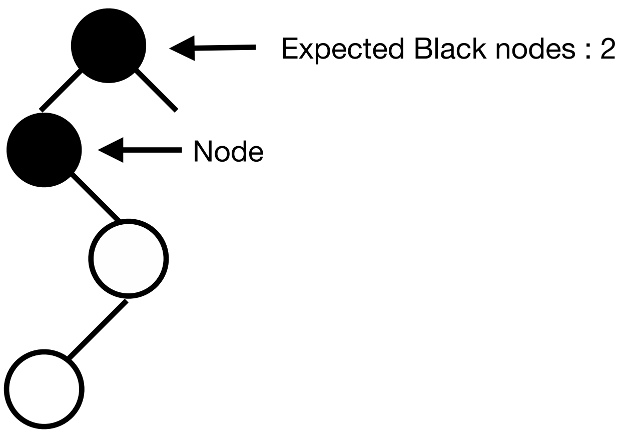
Αρχικά η ρίζα του δέντρου πρέπει να είναι μαύρη βάση των ιδιοτήτων άρα η ρίζα βάφεται μαύρη. Και περιμένουμε ότι πηγαίνοντας σε ένα φύλλο θα συναντήσουμε height(n)/2 μαυρα nodes.



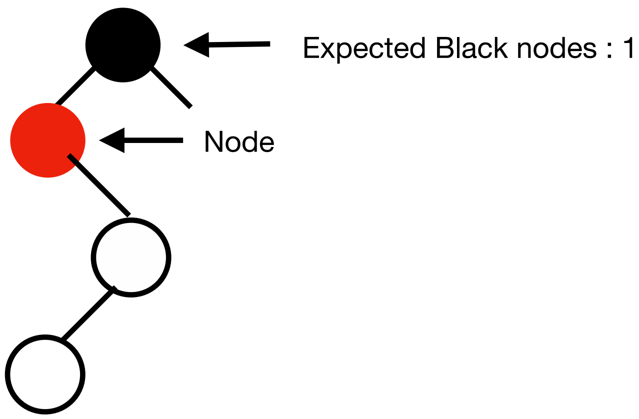
Στην συνέχεια ελέγχουμε αν ο πατέρας του node στο οποίο βρισκόμαστε είναι κόκκινος. Αν είναι τότε χρωματίζουμε το node μάυρο. Και συνεχίζοντας στο μονοπάτι αυτό περιμένουμε να βρουμε όσα μαύρα περίμενε και ο πατέρας αυτού του node.



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.

```
If n == root:
        n.color = black
        n.expected_black = height(n) / 2
else if n.parent == red:
        n.color = black
        n.expected_black = n.parent.expected_black
else if n.parent == black:
        if min(n) == n.parent.expected_black:
                n.color = black
        else min(n) > n.parent.expected_black:
                n.color = red
        n.expected_black = n.parent.expected_black -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(l=1; l < n; l++) Heapify(A[0..l]);
}
```

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

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

l = 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

l = 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

I = 4 -> The leaves 29,15,22,14,12 don’t have Childs so heapify makes no change.

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

I = 5 -> The leaves 29,15,22,14,12,13 don’t have Childs so heapify makes no change.

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

I = 6 -> The leaves 29,15,22,14,12,13,10 don’t have Childs so heapify makes no change.

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

I = 7 -> Node 6 is the only node that has children and it has only a right child and Heapify doesn’t swap 6 with its child 29 because 6 < 29.

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

I = 8 -> Node 9,6 is the only nodes that have children and Heapify doesn’t swap 6 with its child 29 because 6 < 29 and doesn’t swap 9 with its child because 9<22, 9<15.

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

I = 9 -> Node 8,9,6, is the only nodes that have children and Heapify doesn’t swap 6 with its child 29 because 6 < 29 and doesn’t swap 9 with its child because 9<22, 9<15 and doesn’t swap 8 with its child because 8<14, 8<12.

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

I = 10 -> Node 5,8,9,6, is the only nodes that have children and Heapify doesn’t swap 6 with its child 29 because 6 < 29 and doesn’t swap 9 with its child because 9<22, 9<15 and doesn’t swap 8 with its child because 8<14, 8<12 and doesn’t swap 5 with its child because 5<10, 5<13.

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

I = 11 -> Node 2,5,8,9,6, is the only nodes that have children and Heapify doesn’t swap 6 with its child 29 because 6 < 29 and doesn’t swap 9 with its child because 9<22, 9<15 and doesn’t swap 8 with its child because 8<14, 8<12 and doesn’t swap 5 with its child because 5<10, 5<13 and doesn’t swap 2 with its child because 2<6, 2<9.

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

I = 12 -> Node 2,5,8,9,6, is the only nodes that have children and Heapify doesn’t swap 6 with its child 29 because 6 < 29 and doesn’t swap 9 with its child because 9<22, 9<15 and doesn’t swap 8 with its child because 8<14, 8<12 and doesn’t swap 5 with its child because 5<10, 5<13 and doesn’t swap 2 with its child because 2<6, 2<9 but heapify will swap node 11 with 5 because 5<11.

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

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

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

```
Void swap(Table H, int j, int k){
        node tmp = H[ j ];
        H[ j ] = H[ k ];
        H[ k ] = tmp;
}
```

```
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);

}
```