Süre: 70 Dakika

**Manisa Celal Bayar Üniversitesi Yazılım Mühendisliği Bölümü**
# YZM 2116 – Veri Yapıları Dersi

## Final Sınavı Örnek Soruları
Bahar 2018

| Adı ve Soyadı | YANIT ANAHTARI | Öğrenci Numarası | |
|---|---|---|---|
| Grubu | | İmza | |
| Tarih | | Not | /100 |

**Soru#1 (20 puan):** Aşağıda bir ***hash tablosu*** verilmiştir:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| | | | | | | |

a) Yukarıda verilen tabloya herhangi bir *k* nesnesi, **H(*k*)=*k* mod 7** hash fonksiyonuna göre yerleştirilmektedir ve hashing işlemi sonucunda farklı anahtarlara sahip iki eleman için aynı değer üretildiği (çakışma olduğu) takdirde, **doğrusal ölçüm** (*linear probing*) yöntemi kullanılmaktadır. **Buna göre, yukarıdaki hash tablosuna 8, 33, 15, 26, 22 anahtar değerlerinin yerleştirilmesi sonucu oluşacak tabloyu aşağıda belirtiniz:**

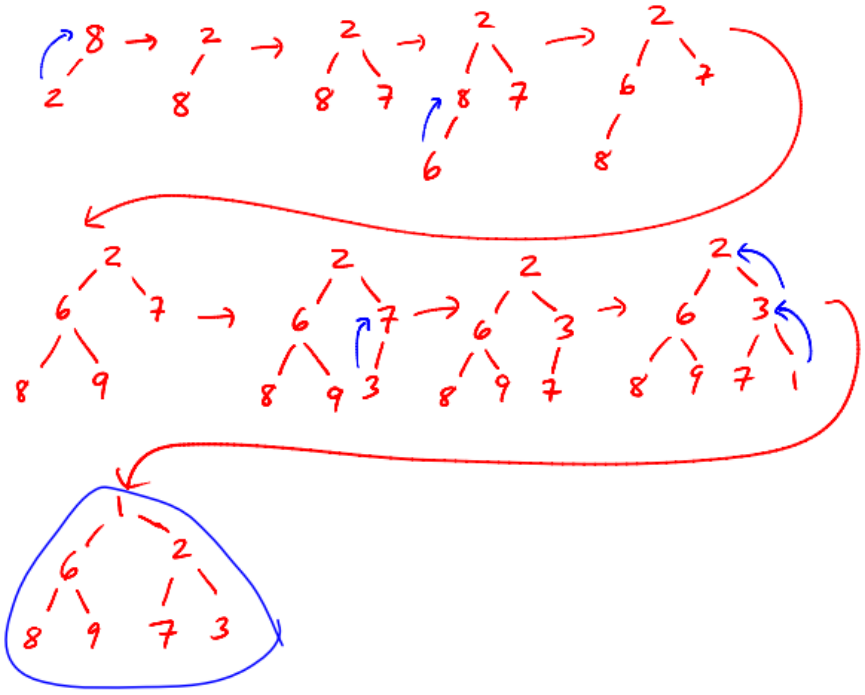
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| | 8 | 15 | 22 | | 33 | 26 |

b) Yukarıda verilen tabloya herhangi bir *k* nesnesi, **H(*k*)=*k* mod 7** hash fonksiyonuna göre yerleştirilmektedir ve hashing işlemi sonucunda farklı anahtarlara sahip iki eleman için aynı değer üretildiği (çakışma olduğu) takdirde, **karesel ölçüm** (*quadratic probing*) yöntemi kullanılmaktadır. **Buna göre, yukarıdaki hash tablosuna 6, 13, 14, 21, 28 anahtar değerlerinin yerleştirilmesi sonucu oluşacak tabloyu aşağıda belirtiniz:**

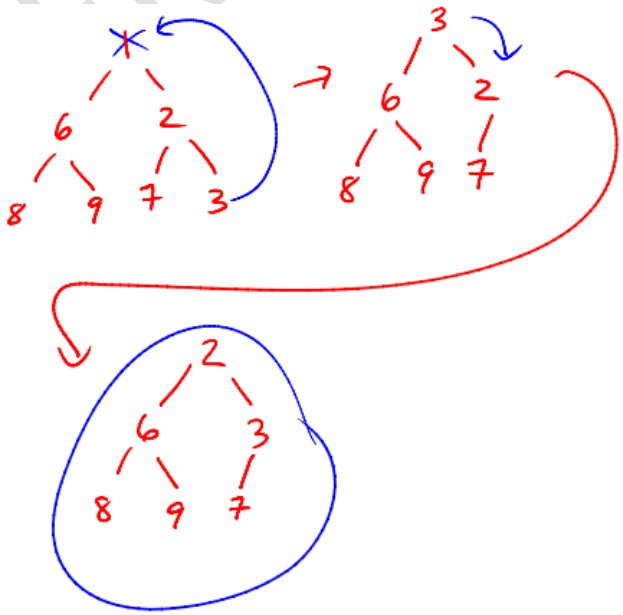
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 13 | 14 | 28 | | 21 | | 6 |

**Soru#2 (20 puan):** {8, 2, 7, 6, 9, 3, 1} **elemanlarının sırasıyla eklenmesi ile oluşan ve başlangıçta boş olan bir** min-heap (*minimum yığın ağacı*) **veri yapısını adım adım çizerek oluşturunuz. Oluşturduğunuz yığın ağacı yapısı üzerinde, 1 kez `deleteMin` (minimum elemanın silinmesi)** işleminin gerçekleşmesi sonucu oluşan yapıya ilişkin ağacı ve tüm ara adımları çizerek gösteriniz.

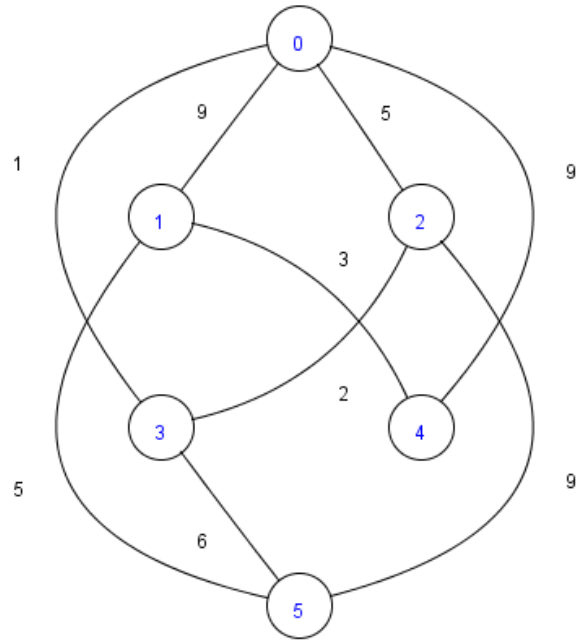
**Yığın ağacı:**

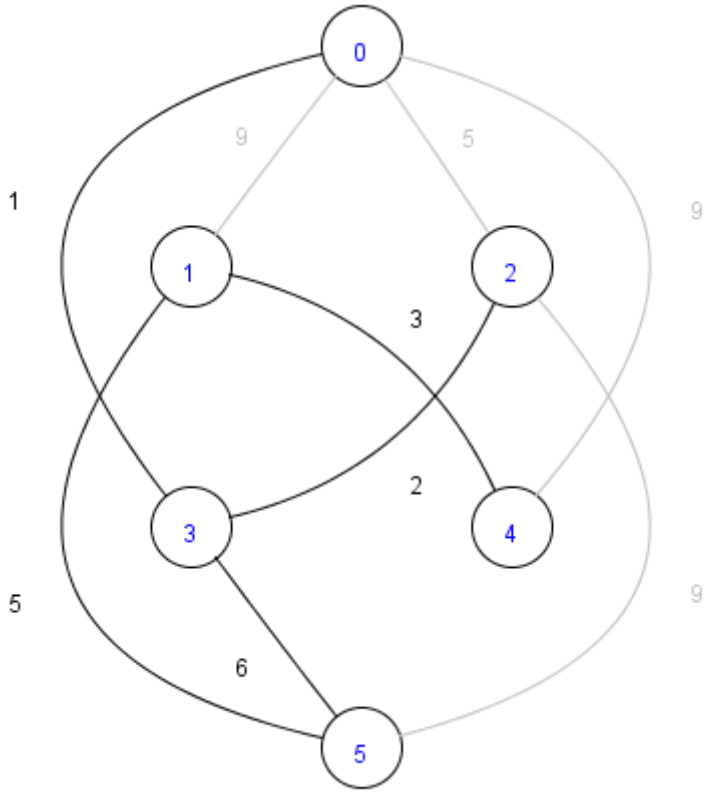


**1 kez `deleteMin` işletildikten sonra:**

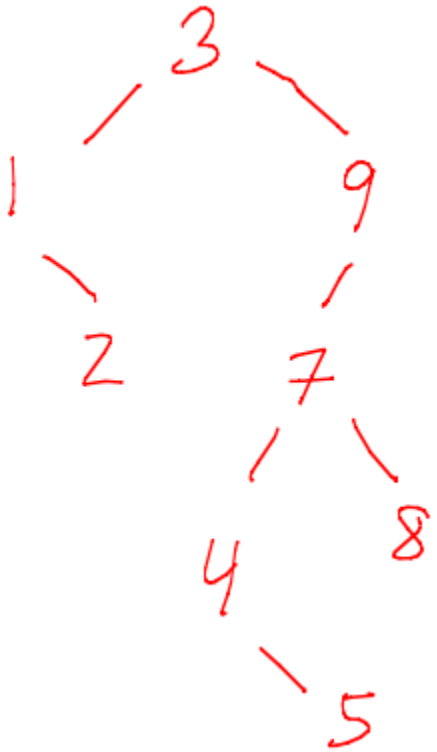


**<u>Soru#3 (15 puan):</u> Aşağıda verilen çizge üzerinde *<u>Kruskal algoritmasını</u>* işleterek <u>minimum kapsayan ağaç problemini</u> çözünüz. Adımlarınızı gösteriniz. Çözüm sonucu elde ettiğiniz minimum kapsayan ağacı şekil üzerinde belirtiniz.**

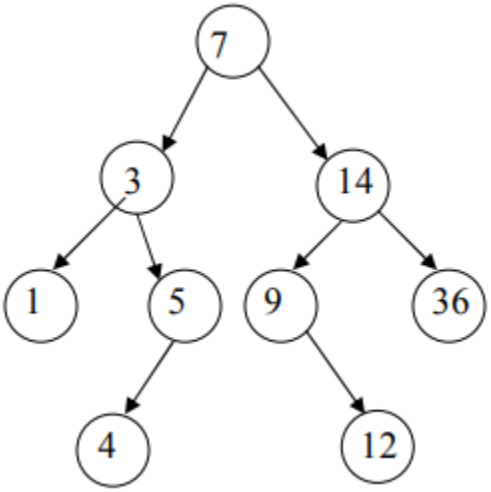


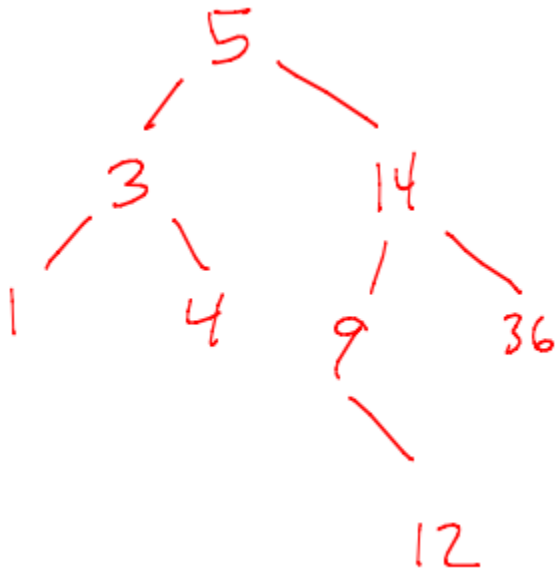
**<u>Soru#4 (10 puan):</u>** {3, 9, 1, 7, 4, 5, 8, 2} **elemanlarının sırasıyla eklenmesi ile oluşan ve başlangıçta boş olan bir** *ikili arama ağacını (binary search tree)* **<u>adım adım çizerek</u> oluşturunuz.**



**Soru#5 (10 puan):** Aşağıda bir **ikili arama ağacı** (*binary search tree*) verilmiştir:




**Buna göre, yukarıda verilen ikili arama ağacında "7" değerinin silinmesi sonrası oluşan ağaç yapısını, ara aşamaları da çizerek belirtiniz.**



**Soru#6 (25 puan): Verilen bir ikili arama ağacının heap (yığın ağacı) veri yapısı özelliği taşıyıp taşımadığını belirleyen bir algoritma yazınız.** [Gerçekleştirimde, C#, Java, C vb. diller kullanılabilir. Çözüm, sözde kod ile algoritmik olarak da ifade edilebilir.]

```
/* Java program to checks if a binary tree is max heap ot not */

// A Binary Tree node
class Node
{
    int key;
    Node left, right;

    Node(int k)
    {
        key = k;
        left = right = null;
    }
}

class Is_BinaryTree_MaxHeap
{
    /* This function counts the number of nodes in a binary tree */
    int countNodes(Node root)
    {
        if(root==null)
            return 0;
        return(1 + countNodes(root.left) + countNodes(root.right));
    }

    /* This function checks if the binary tree is complete or not */
    boolean isCompleteUtil(Node root, int index, int number_nodes)
    {
        // An empty tree is complete
        if(root == null)
            return true;

        // If index assigned to current node is more than
        // number of nodes in tree, then tree is not complete
        if(index >= number_nodes)
            return false;

        // Recur for left and right subtrees
        return isCompleteUtil(root.left, 2*index+1, number_nodes) &&
               isCompleteUtil(root.right, 2*index+2, number_nodes);
    }

    // This Function checks the heap property in the tree.
    boolean isHeapUtil(Node root)
    {
        //  Base case : single node satisfies property
        if(root.left == null && root.right==null)
            return true;

        //  node will be in second last level
        if(root.right == null)
        {
            //  check heap property at Node
            //  No recursive call , because no need to check last level
            return root.key >= root.left.key;
```

```
        }
        else
        {
            //  Check heap property at Node and
            //  Recursive check heap property at left and right subtree
            if(root.key >= root.left.key && root.key >= root.right.key)
                return isHeapUtil(root.left)  && isHeapUtil(root.right);
            else
                return false;
        }
    }

    //  Function to check binary tree is a Heap or Not.
    boolean isHeap(Node root)
    {
        if(root == null)
            return true;

        // These two are used in isCompleteUtil()
        int node_count = countNodes(root);

        if(isCompleteUtil(root,      0      ,      node_count)==true      &&
isHeapUtil(root)==true)
            return true;
        return false;
    }

    // driver function to test the above functions
    public static void main(String args[])
    {
        Is_BinaryTree_MaxHeap bt = new Is_BinaryTree_MaxHeap();

        Node root = new Node(10);
        root.left = new Node(9);
        root.right = new Node(8);
        root.left.left = new Node(7);
        root.left.right = new Node(6);
        root.right.left = new Node(5);
        root.right.right = new Node(4);
        root.left.left.left = new Node(3);
        root.left.left.right = new Node(2);
        root.left.right.left = new Node(1);

        if(bt.isHeap(root) == true)
            System.out.println("Given binary tree is a Heap");
        else
            System.out.println("Given binary tree is not a Heap");
    }
}
```