CENG 201 Veri Yapıları 6: Binary Heap ve Huffman Ağaçları

Öğr.Gör. Şevket Umut ÇAKIR

Pamukkale Üniversitesi

Hafta 6

Anahat

Öncelik Kuyruğu(Priority Queue) Tanım

Tanım

Öncelik kuyruğu ekleme(insert) ve en küçüğü silme(deleteMin) işlemlerine sahip bir veri yapısıdır. deleteMin işlemi normal kuyruk yapısındaki dequeue işlemi gibi çalışır ama her zaman en küçük değere sahip elemanı listeden çıkarır.

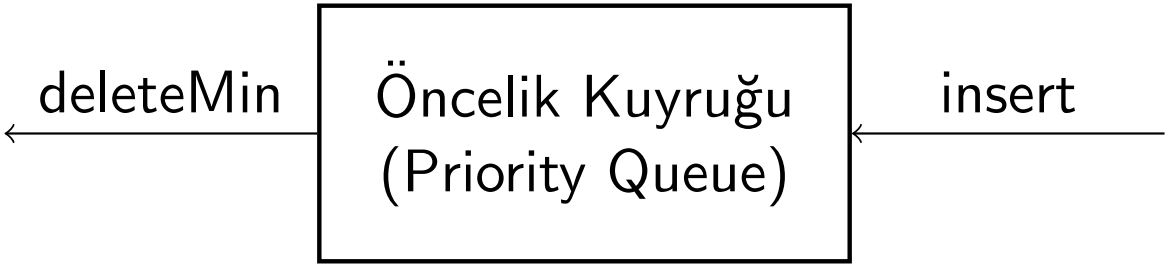


Figure: Öncelik kuyruğu modeli

İkili Yığın/Binary Heap

Tanım

Öncelik kuyruklarının gerçekleştirilmesi için ikili ağaç yapısında verimli bir veri yapısıdır.

Heap aşağıdaki özellikleri sağlar:

- Heap tam ikili ağaç(complete binary tree) yapısındadır
- Bir düğümdeki anahtar değeri çocukarının anahtar değerinden daha küçüktür(Minimum Heap)
- Tam ikili ağaç yapısında olduğu için dizilerle temsil edilebilir, sol çocuk: $2i$, sağ çocuk: $2i + 1$
- Ekleme ve silme işlemlerinde bozulan heap yapısını oluşturmak için **Heapify** işlemi yapılır

Heap Örneği

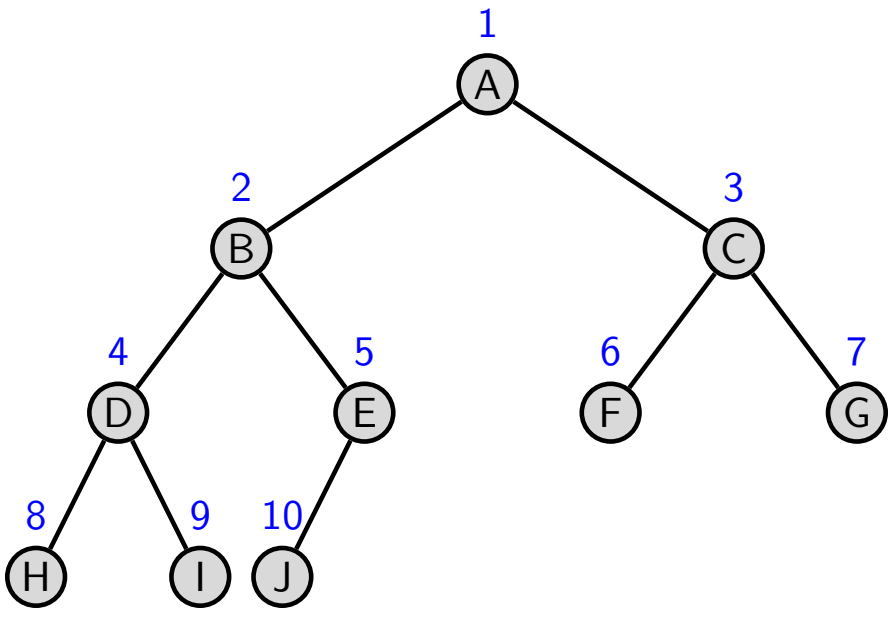


Figure: Heap Örneği

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Değer		A	B	C	D	E	F	G	H	I	J			

Ekleme

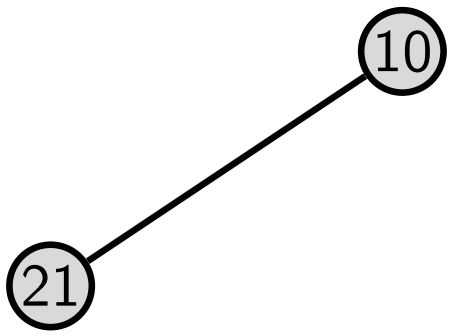
- Ekleme işlemi tam ikili ağacın en alt seviyesinin en sağına gerçekleştirilir
- Eğer Heap özelliği bozulmuşsa ebeveyn ile eklenen çocuk yer değiştirir, bu işlem köke kadar devam eder

Ekleme Örneği: 10

~~10~~, 21, 12, 16, 5, 8, 65, 81, 15, 4 değerlerini boş bir ikili yığına ekleyelim

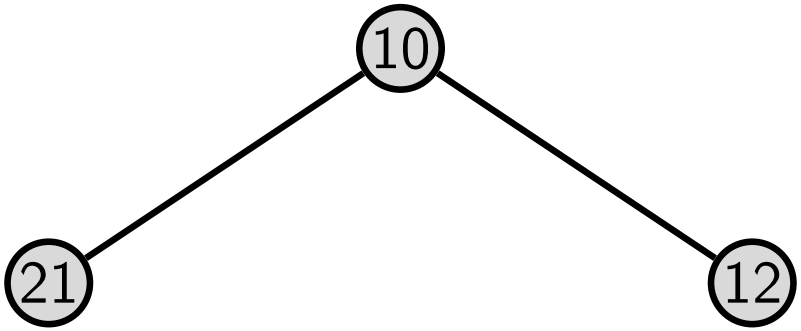
Ekleme Örneği: 21

~~10~~, ~~21~~, 12, 16, 5, 8, 65, 81, 15, 4 değerlerini boş bir ikili yığına ekleyelim



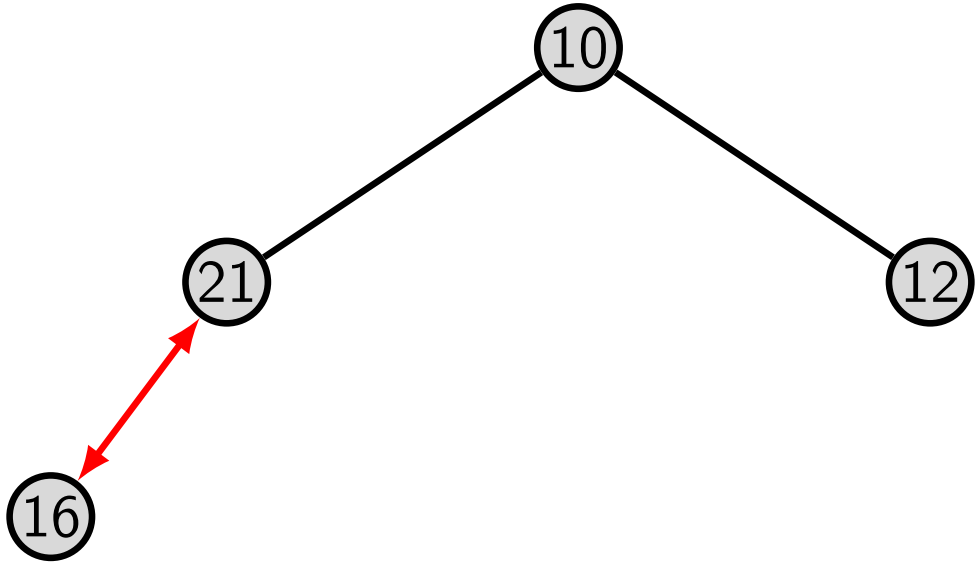
Ekleme Örneği: 21

~~10~~, ~~21~~, ~~12~~, 16, 5, 8, 65, 81, 15, 4 değerlerini boş bir ikili yığına ekleyelim



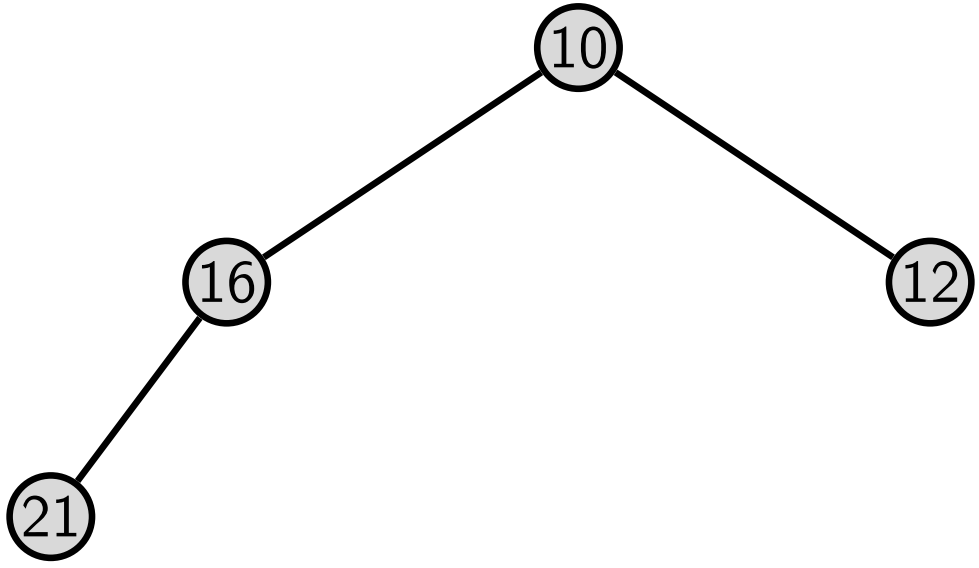
Ekleme Örneği: 16

~~10~~, ~~21~~, ~~12~~, ~~16~~, 5, 8, 65, 81, 15, 4 değerlerini boş bir ikili yığına ekleyelim



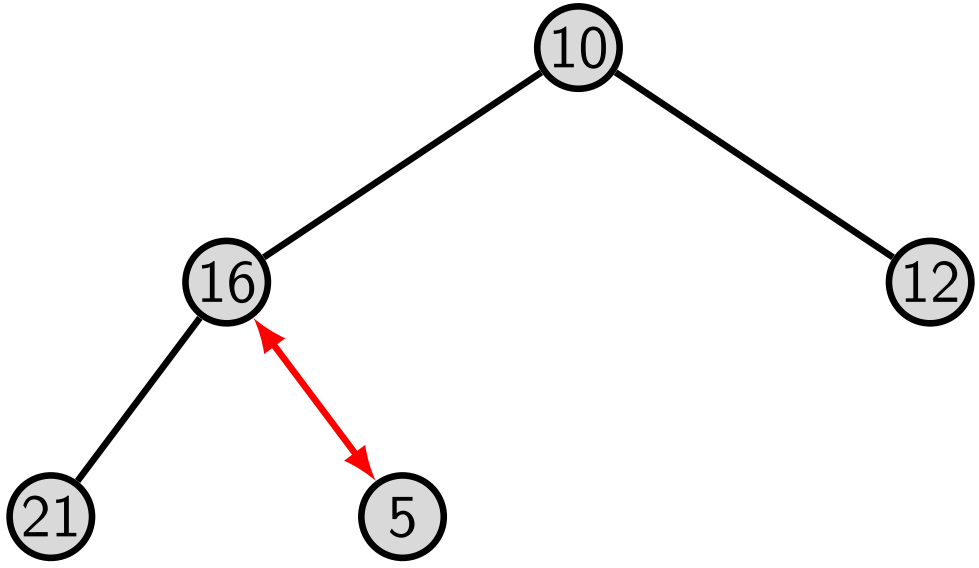
Ekleme Örneği: 16

~~10~~, ~~21~~, ~~12~~, ~~16~~, 5, 8, 65, 81, 15, 4 değerlerini boş bir ikili yığına ekleyelim



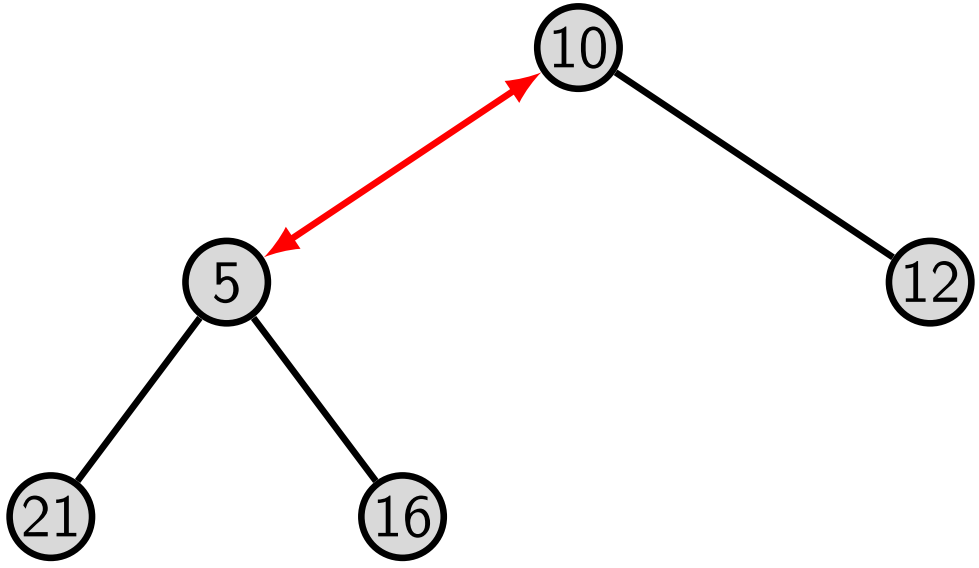
Ekleme Örneği: 5

~~10~~, ~~21~~, ~~12~~, ~~16~~, ~~5~~, 8, 65, 81, 15, 4 değerlerini boş bir ikili yığına ekleyelim



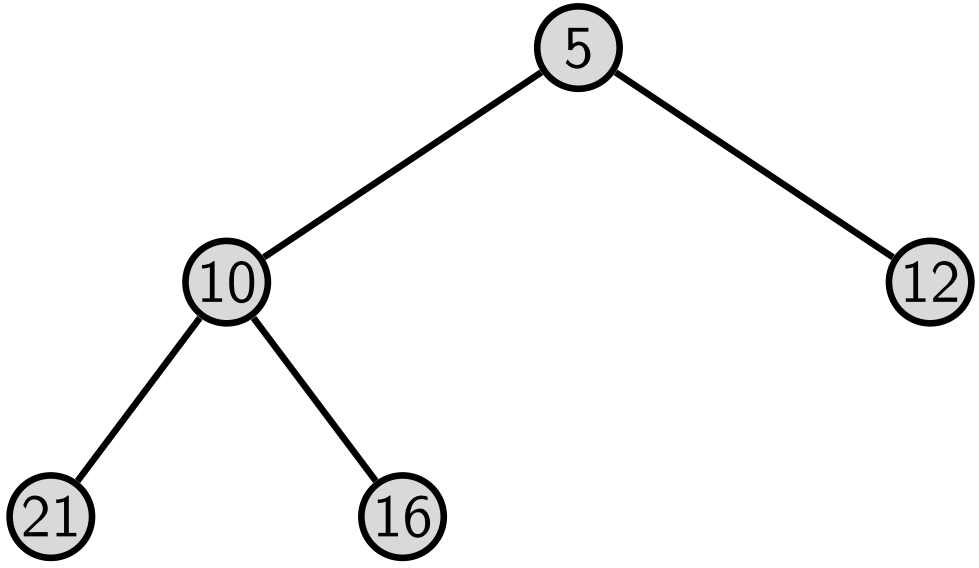
Ekleme Örneği: 5

~~10~~, ~~21~~, ~~12~~, ~~16~~, ~~5~~, 8, 65, 81, 15, 4 değerlerini boş bir ikili yığına ekleyelim



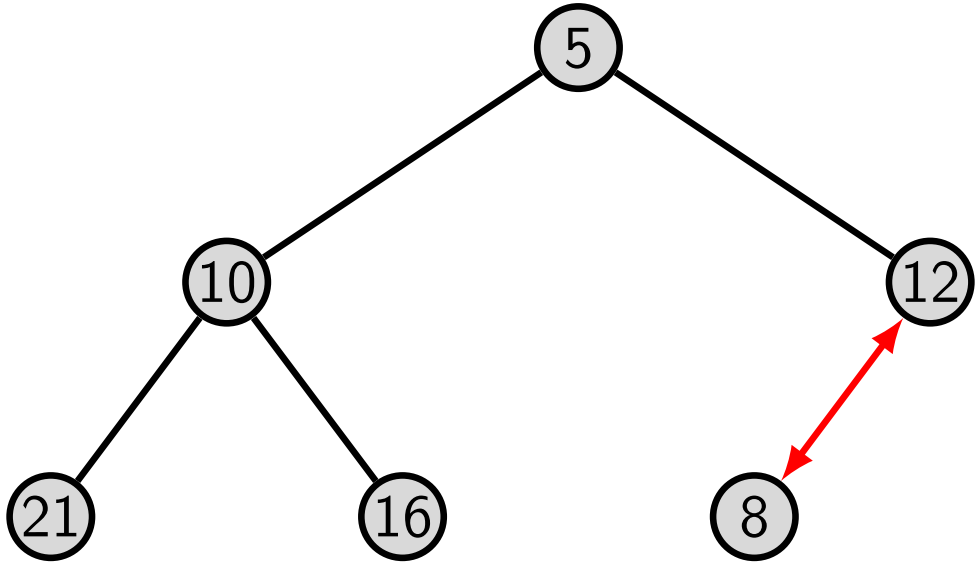
Ekleme Örneği: 5

~~10~~, ~~21~~, ~~12~~, ~~16~~, ~~5~~, 8, 65, 81, 15, 4 değerlerini boş bir ikili yığına ekleyelim



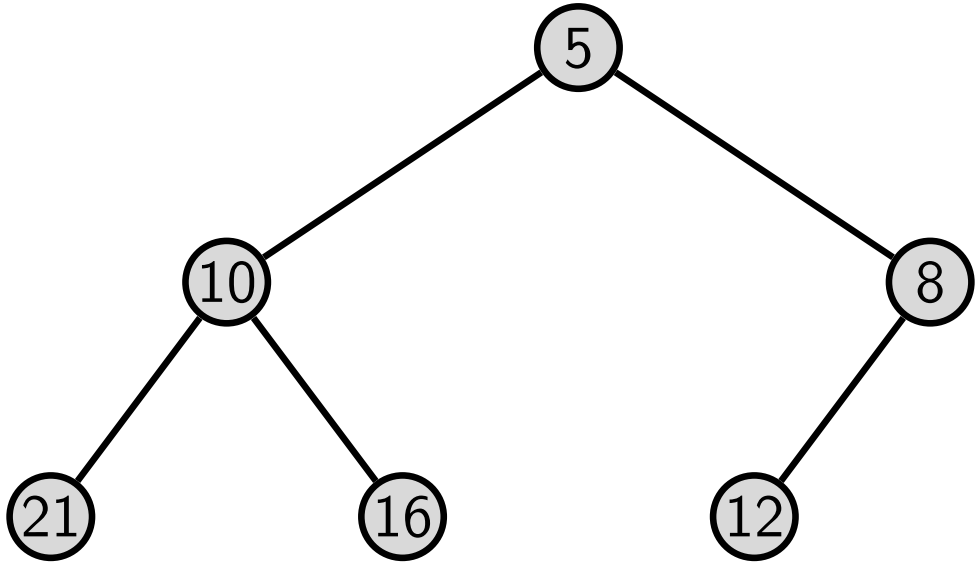
Ekleme Örneği: 8

~~10~~, ~~21~~, ~~12~~, ~~16~~, ~~5~~, ~~8~~, 65, 81, 15, 4 değerlerini boş bir ikili yığına ekleyelim



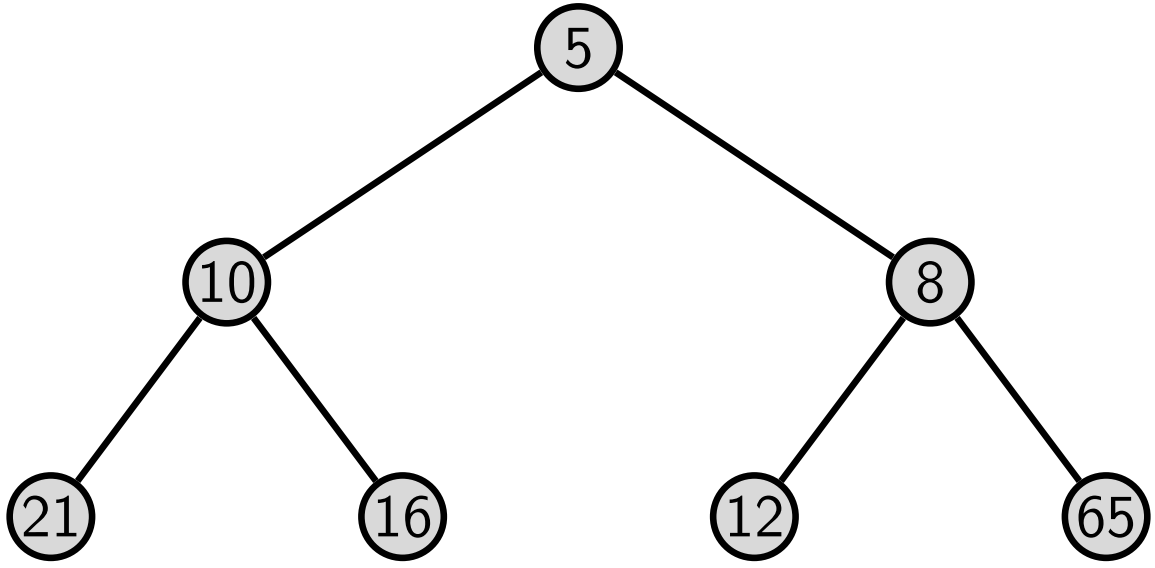
Ekleme Örneği: 8

~~10~~, ~~21~~, ~~12~~, ~~16~~, ~~5~~, ~~8~~, 65, 81, 15, 4 değerlerini boş bir ikili yığına ekleyelim



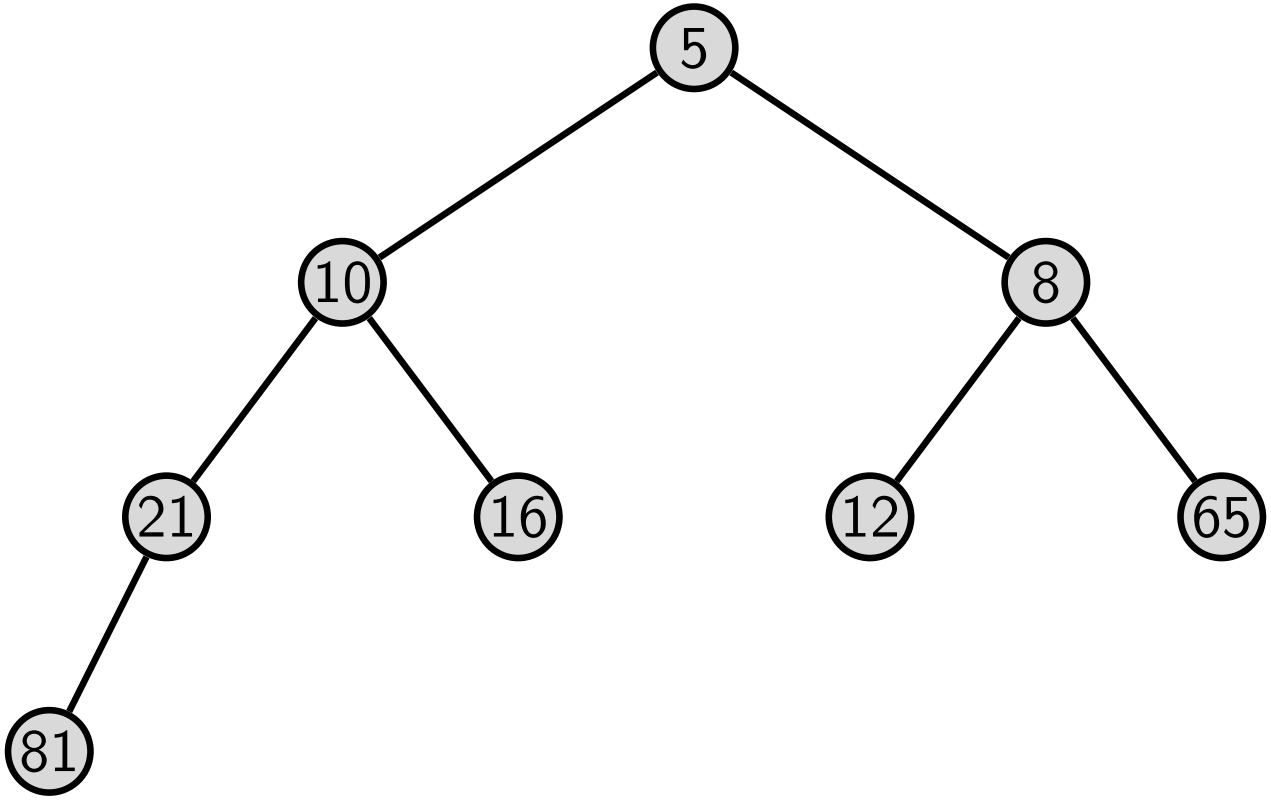
Ekleme Örneği: 65

~~10~~, ~~21~~, ~~12~~, ~~16~~, ~~5~~, ~~8~~, ~~65~~, 81, 15, 4 değerlerini boş bir ikili yığına ekleyelim



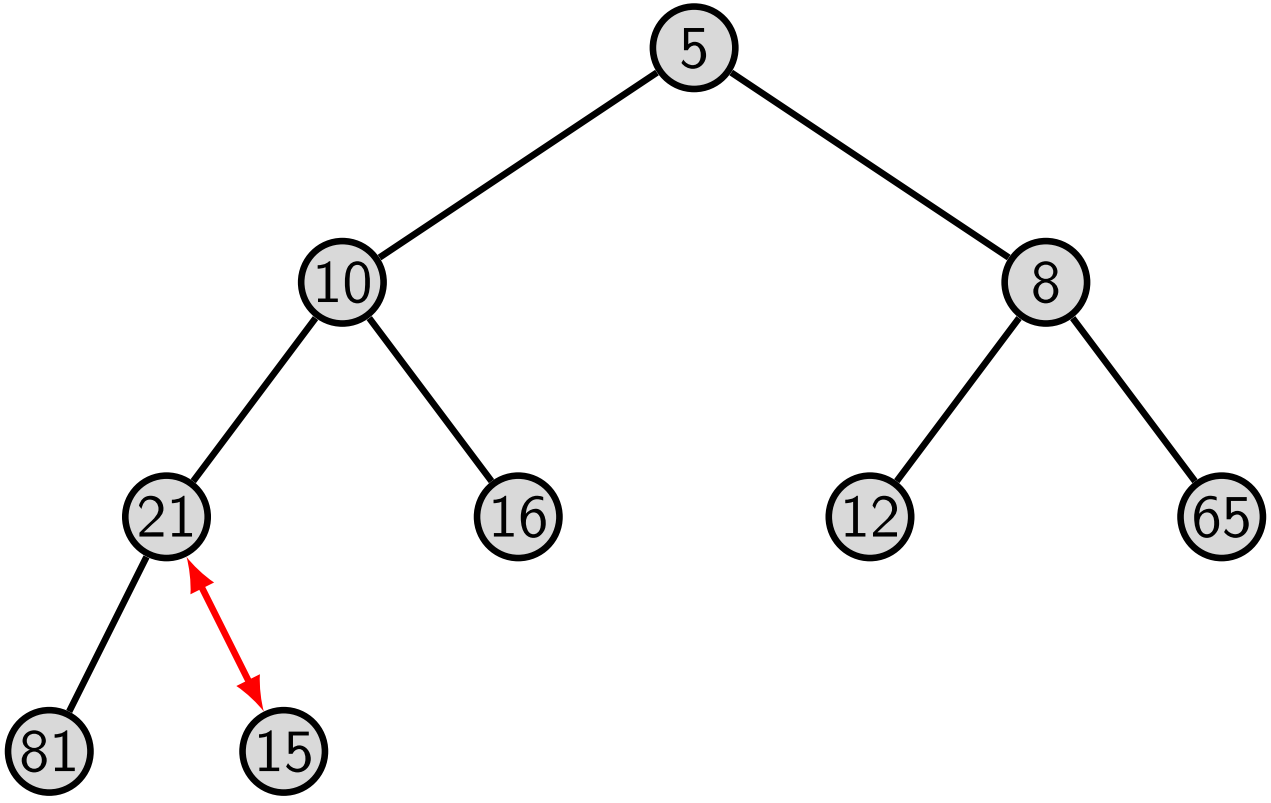
Ekleme Örneği: 81

~~10~~, ~~21~~, ~~12~~, ~~16~~, ~~5~~, ~~8~~, ~~65~~, ~~81~~, 15, 4 değerlerini boş bir ikili yığına ekleyelim



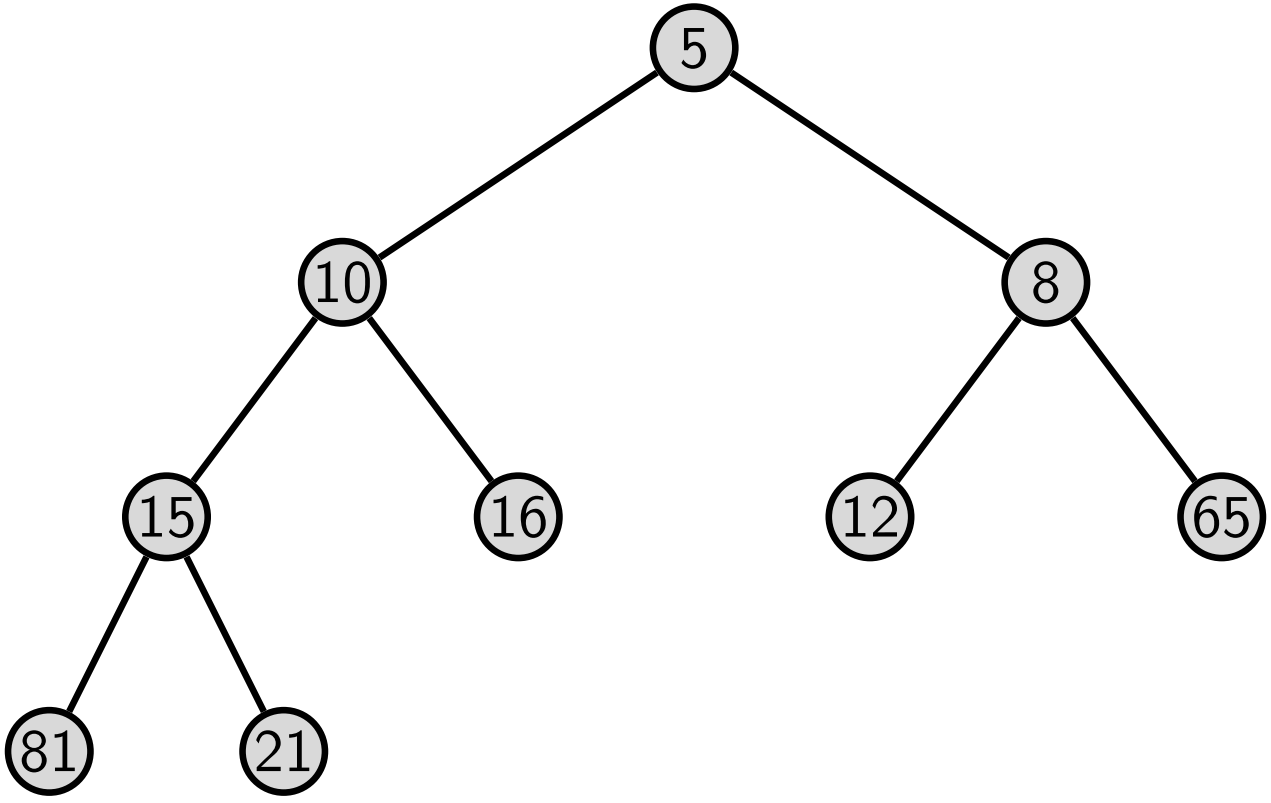
Ekleme Örneği: 15

~~10~~, ~~21~~, ~~12~~, ~~16~~, ~~5~~, ~~8~~, ~~65~~, ~~81~~, ~~15~~, 4 değerlerini boş bir ikili yığına ekleyelim



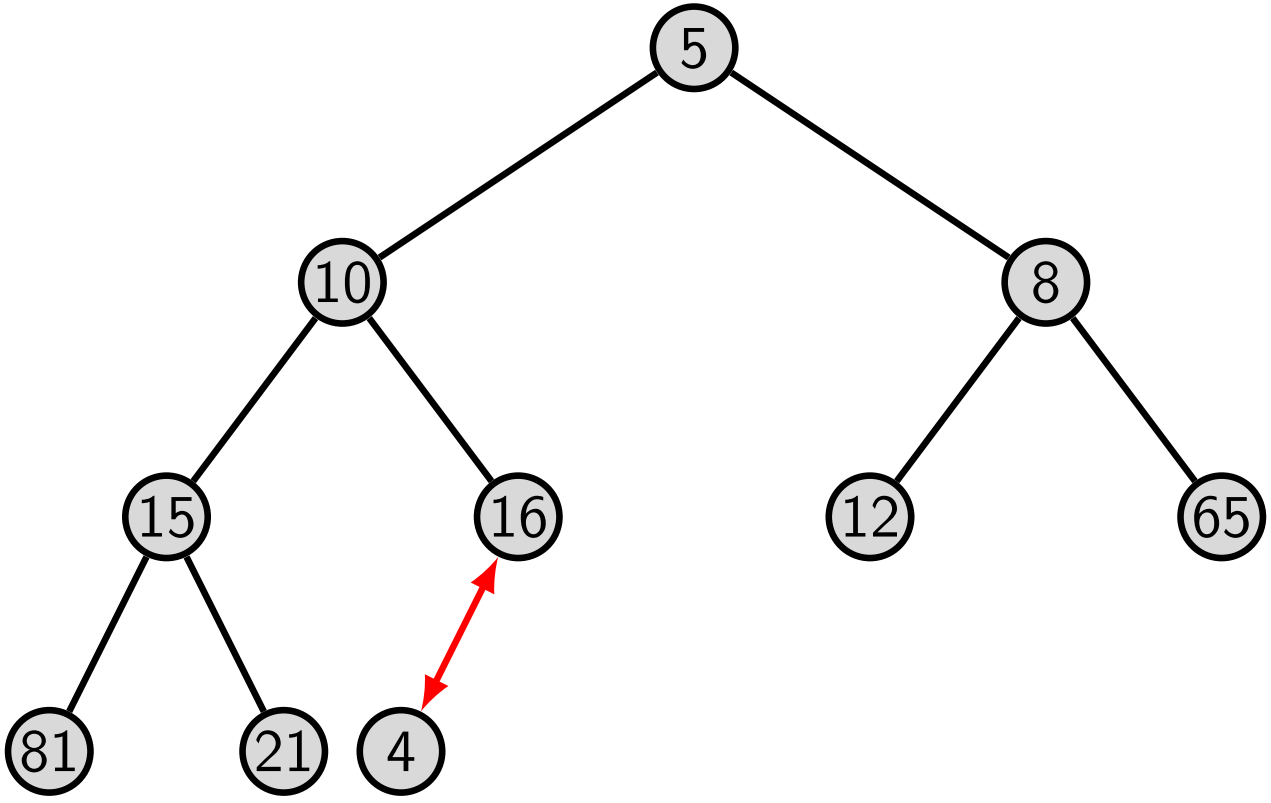
Ekleme Örneği: 15

~~10~~, ~~21~~, ~~12~~, ~~16~~, ~~5~~, ~~8~~, ~~65~~, ~~81~~, ~~15~~, 4 değerlerini boş bir ikili yığına ekleyelim



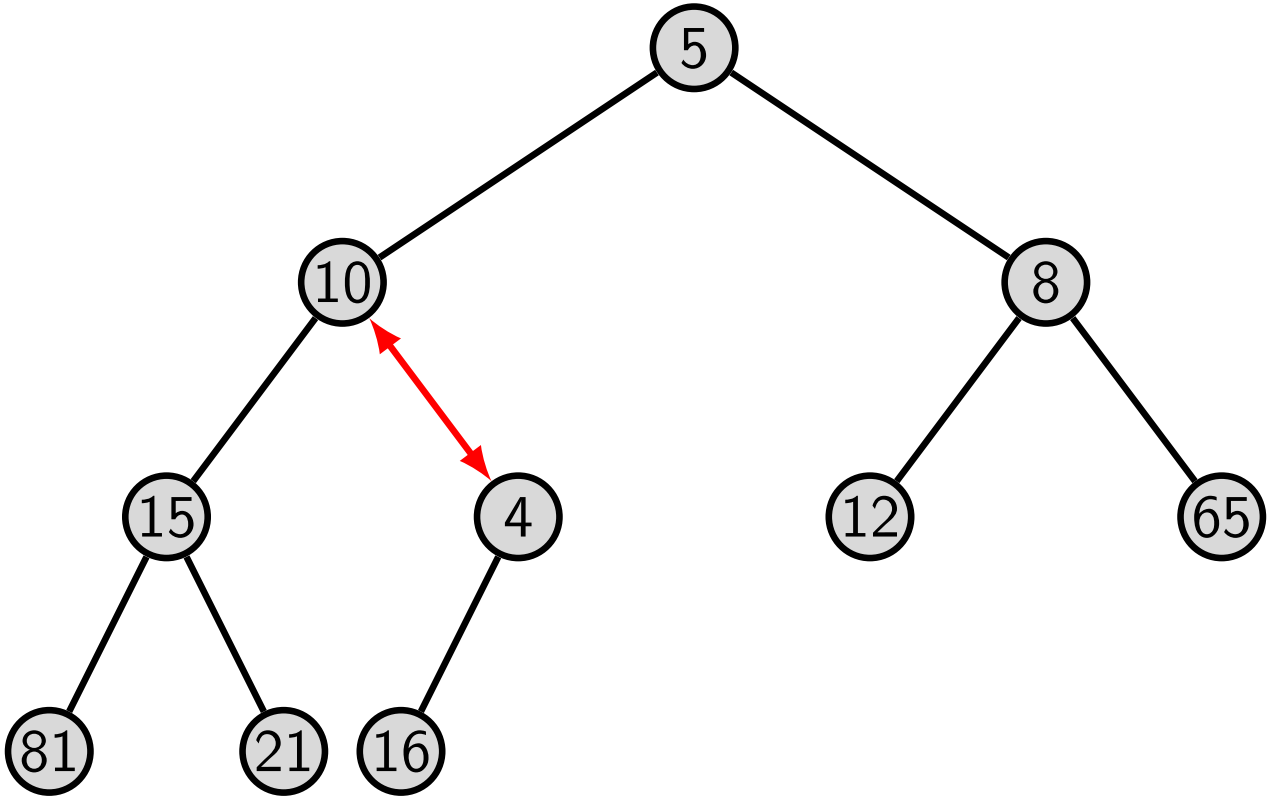
Ekleme Örneği: 4

~~10~~, ~~21~~, ~~12~~, ~~16~~, ~~5~~, ~~8~~, ~~65~~, ~~81~~, ~~15~~, ~~4~~ değerlerini boş bir ikili yığına ekleyelim



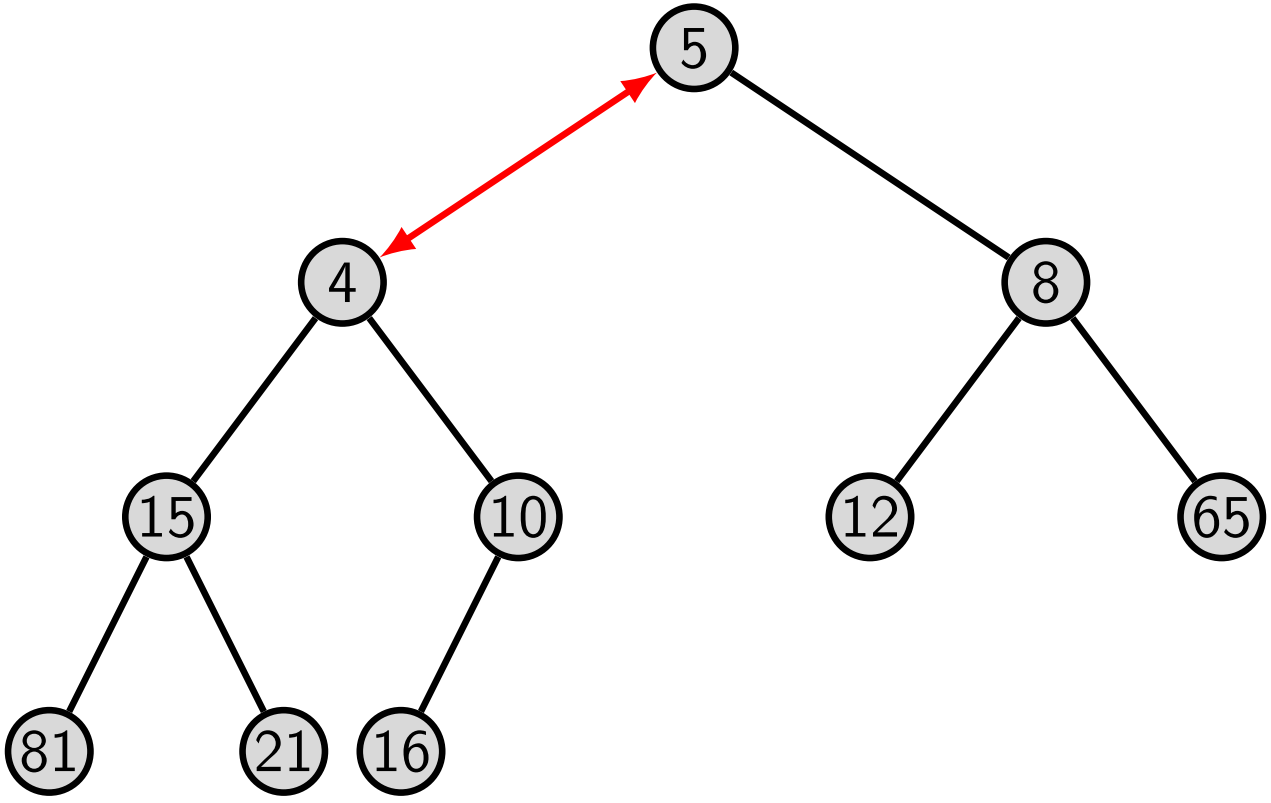
Ekleme Örneği: 4

~~10~~, ~~21~~, ~~12~~, ~~16~~, ~~5~~, ~~8~~, ~~65~~, ~~81~~, ~~15~~, ~~4~~ değerlerini boş bir ikili yığına ekleyelim



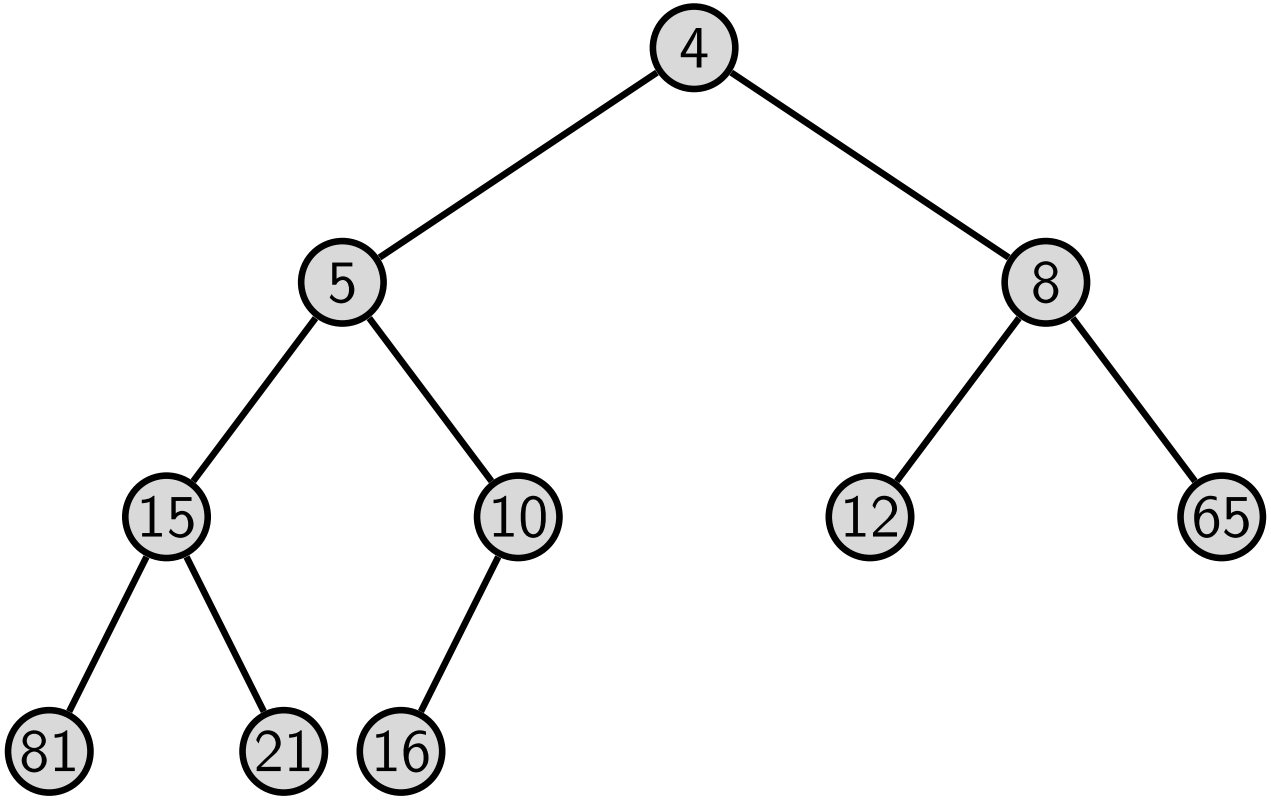
Ekleme Örneği: 4

~~10~~, ~~21~~, ~~12~~, ~~16~~, ~~5~~, ~~8~~, ~~65~~, ~~81~~, ~~15~~, ~~4~~ değerlerini boş bir ikili yığına ekleyelim



Ekleme Örneği: 4

~~10~~, ~~21~~, ~~12~~, ~~16~~, ~~5~~, ~~8~~, ~~65~~, ~~81~~, ~~15~~, ~~4~~ değerlerini boş bir ikili yığına ekleyelim

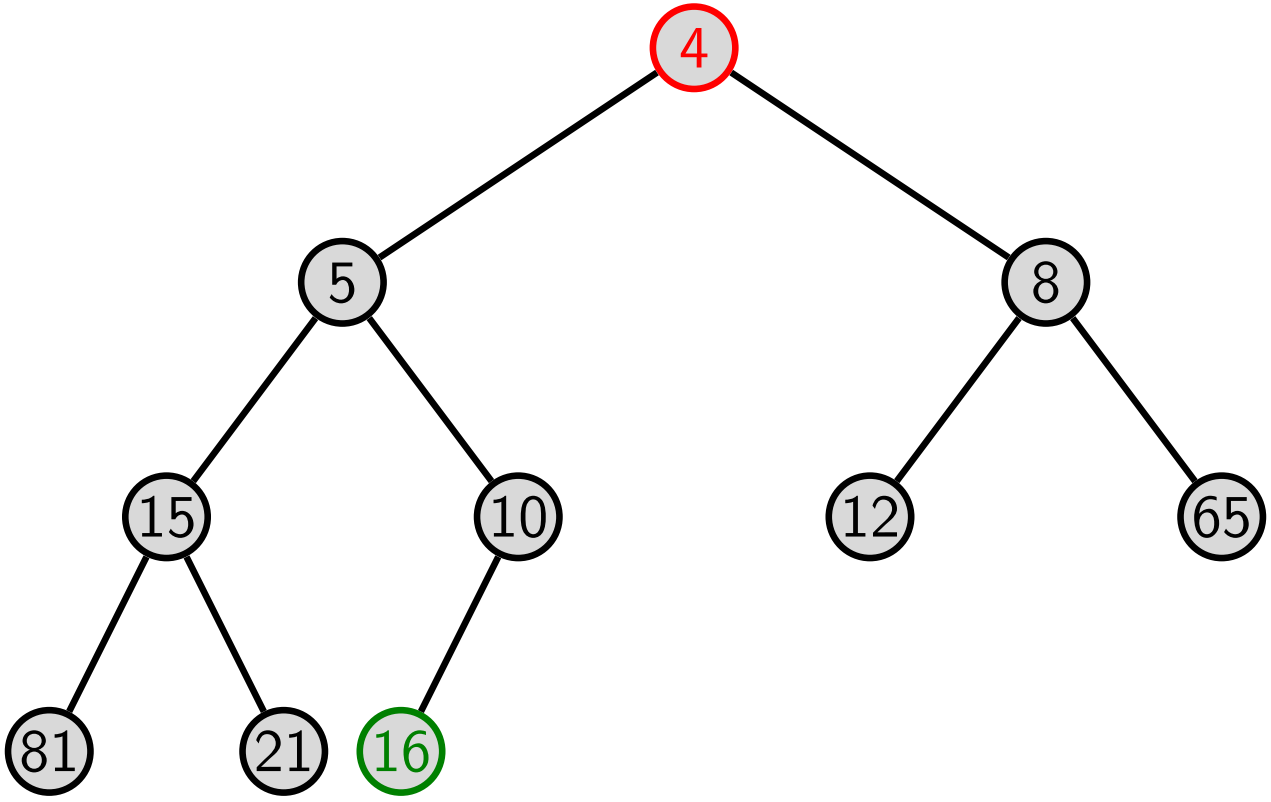


Silme

- Silme işlemi kökten yapılır(deleteMin)
- Kökteki elemanın yerine en alt seviyenin en sağındaki eleman çıkartılıp yazılır
- Heap özelliğini sağlamak için aşağı doğru çocuklarından hangisi küçükse onunla yer değiştirir

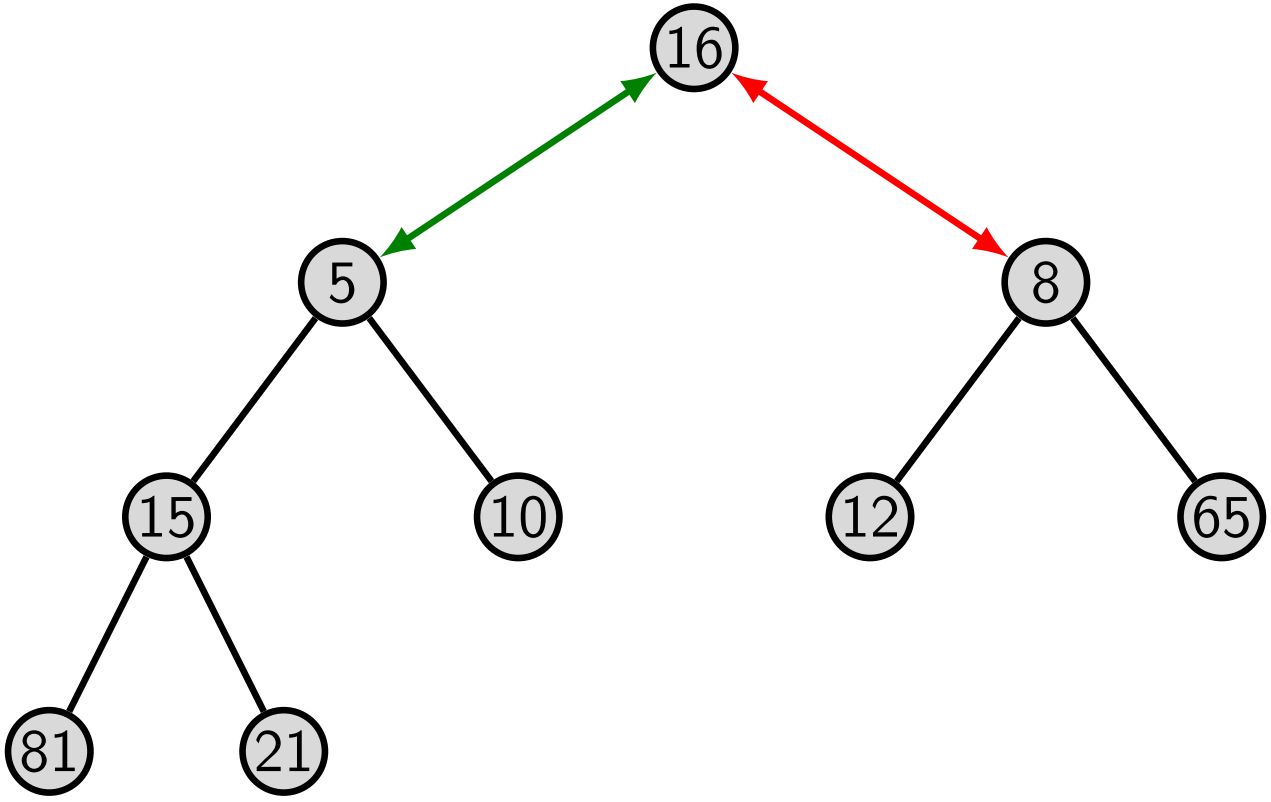
Silme Örneği

Kökteki 4 değerini silme



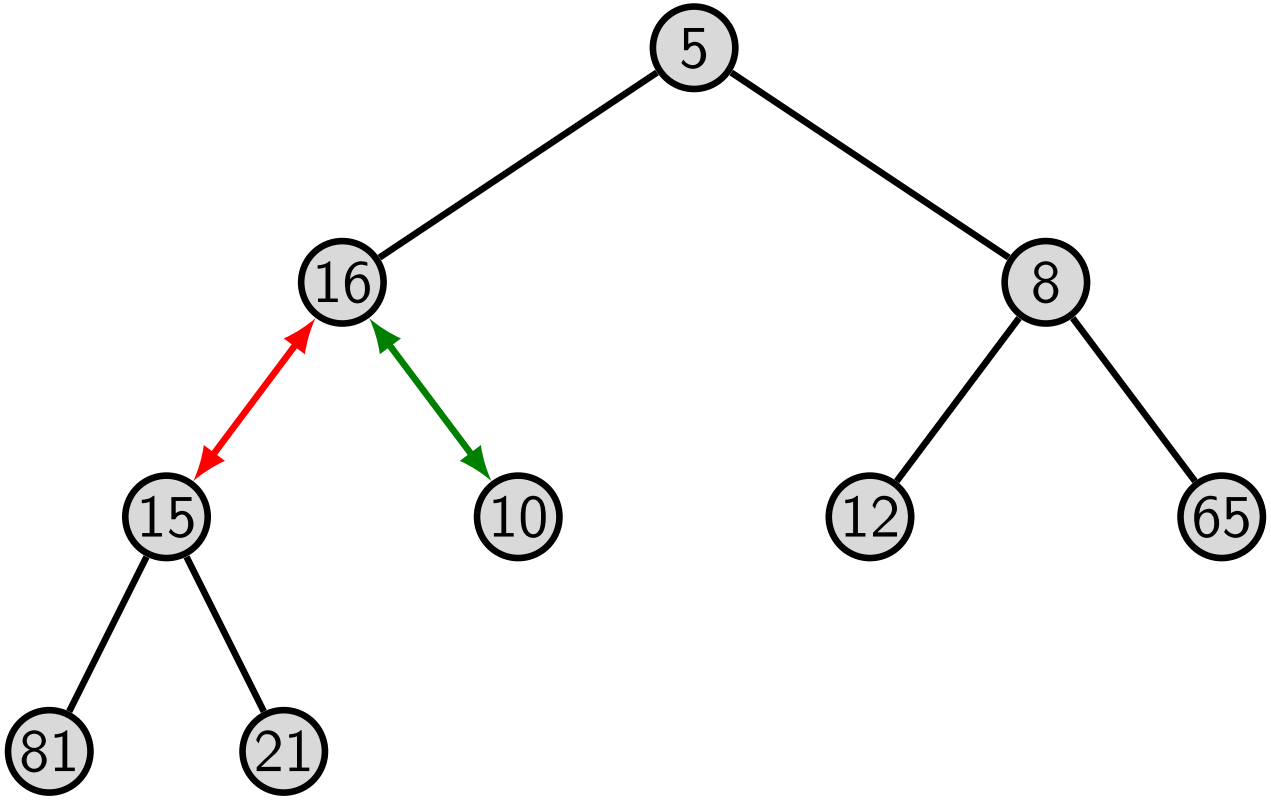
Silme Örneği

Kökteki 4 değerini silme



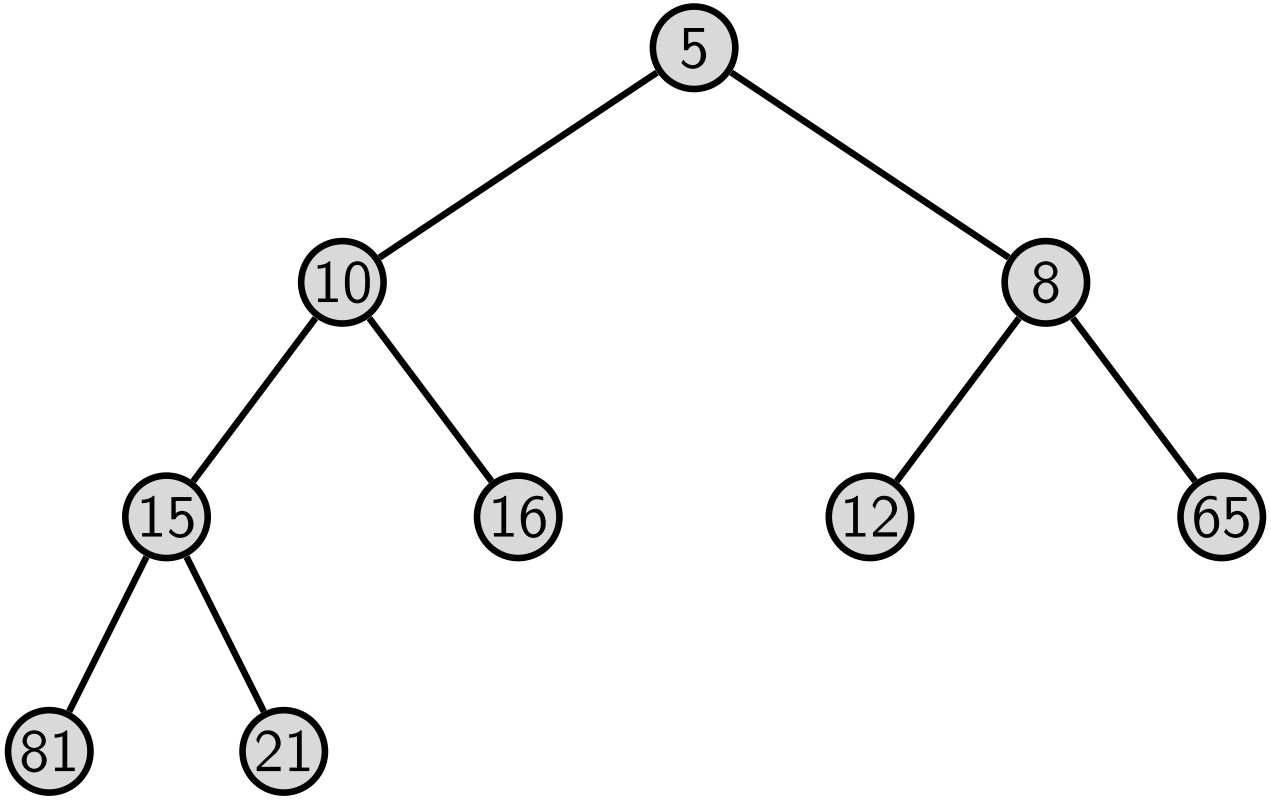
Silme Örneği

Kökteki 4 değerini silme



Silme Örneği

Kökteki 4 değerini silme



Görselleştirme

- İkili Yığın(Binary Heap) için
  http://www.cs.usfca.edu/ galles/visualization/Heap.html

Huffman Kodlama

Huffman Kodlama

Huffman kodlama, kayıpsız bir veri sıkıştırma yöntemidir. Genellikle bir metin içindeki karakterler ikili(binary) bir koda dönüştürülür ve bu kodun boyutunun orjinal metnin boyutundan daha küçük olması beklenir. Tüm karakterlerin frekans(belirme sayısı) değeri hesaplanır ve bu değerler kullanılarak ikili bir ağaç oluşturulur.

Huffman Ağacı Oluşturma Algoritması

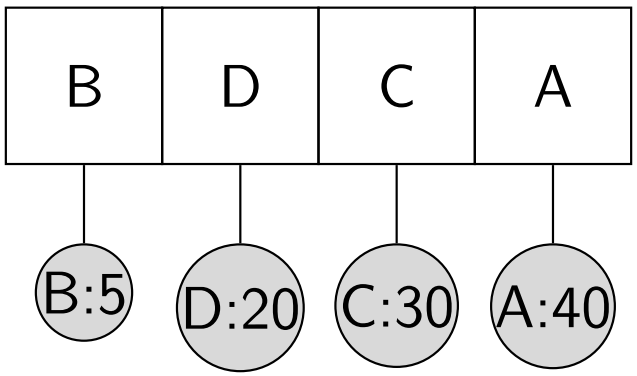
Düğümleri saklamak için öncelik kuyruğu oluştur(pq)
Bütün karakterleri frekans değerine göre kuyruğa ekle
while *Kuyrukta birden fazla eleman olduğu sürece* **do**
    n1=Kuyruktan bir düğüm çek
    n2=Kuyruktan bir düğüm çek
    Sol çocuğu n1, sağ çocuğu n2 ve frekansı toplamları olacak şekilde yeni bir düğüm oluştur ve kuyruğa ekle
end
Kuyruktaki tek değer ağacın kökünü verir

Algorithm 1: Huffman Ağacı Oluşturma Algoritması

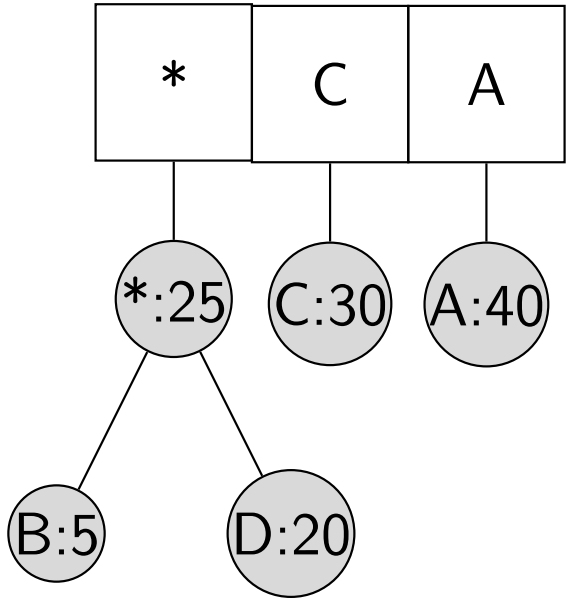
Huffman Oluşturma Örneği

İçinde 4 karakter bulunan ve frekans değerleri aşağıdaki şekilde olan bir metin için Huffman ağacını oluşturalım.

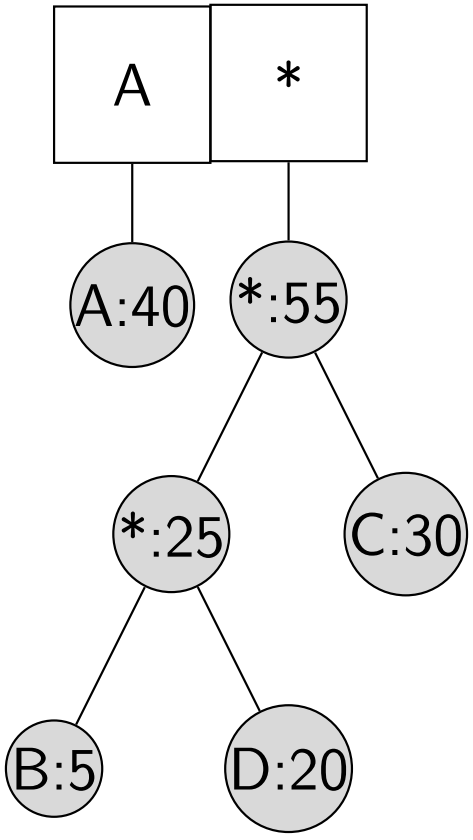
Karakter	**A**	**B**	**C**	**D**
Frekans	40	5	30	20



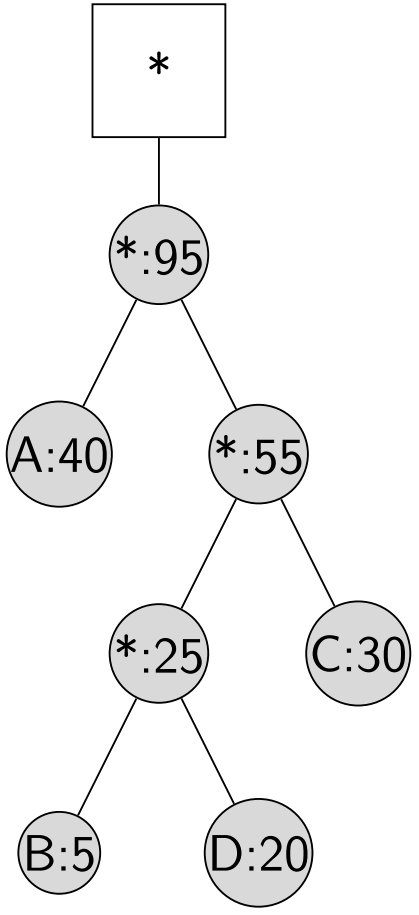
Huffman Oluşturma Örneği



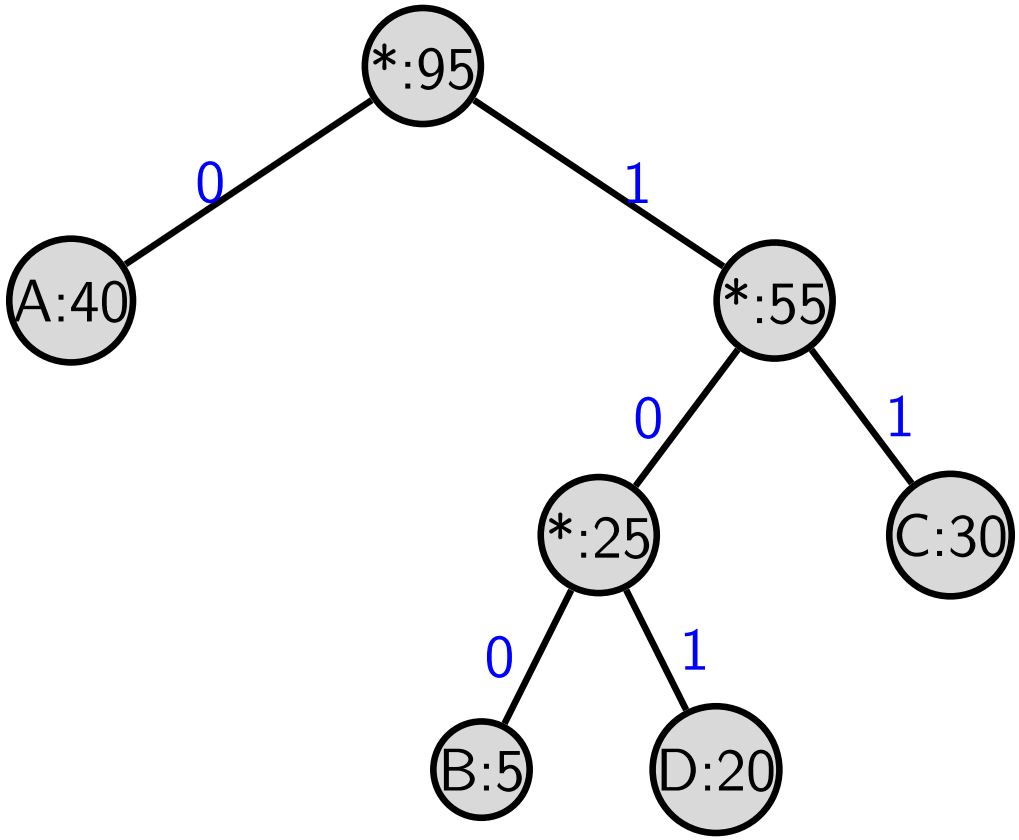
Huffman Oluşturma Örneği



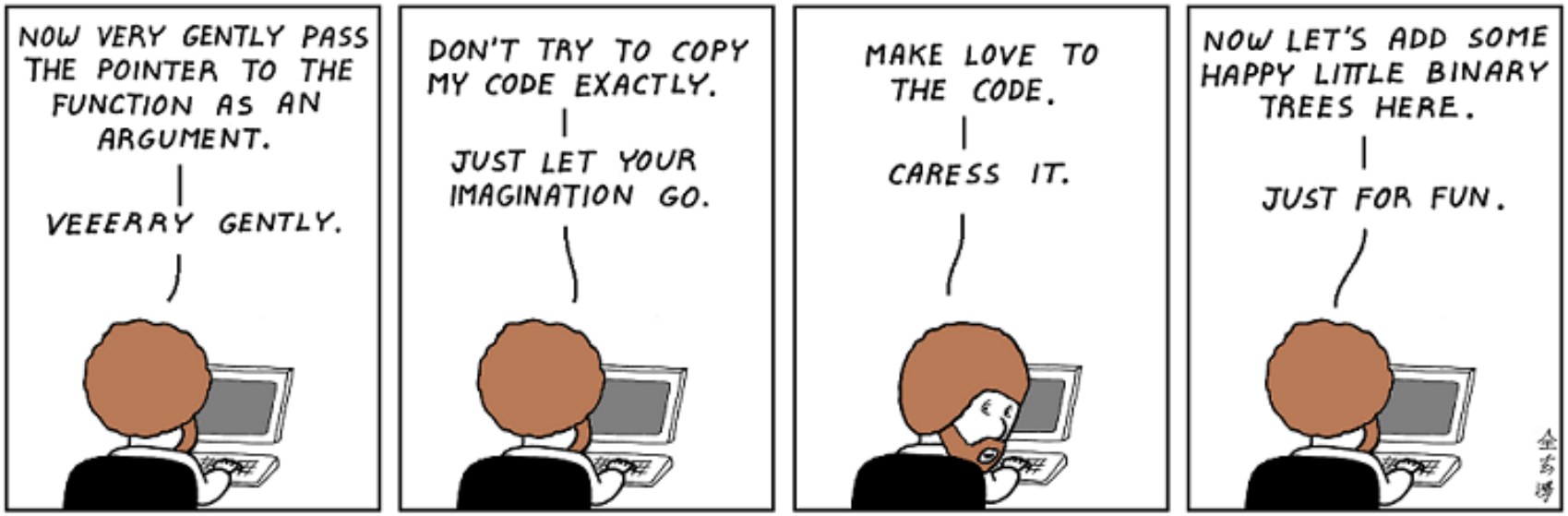
Huffman Oluşturma Örneği



Oluşan Huffman Ağacı



Karakter	**A**	**B**	**C**	**D**
Kod	0	100	11	101



1

[1] abstrusegoose