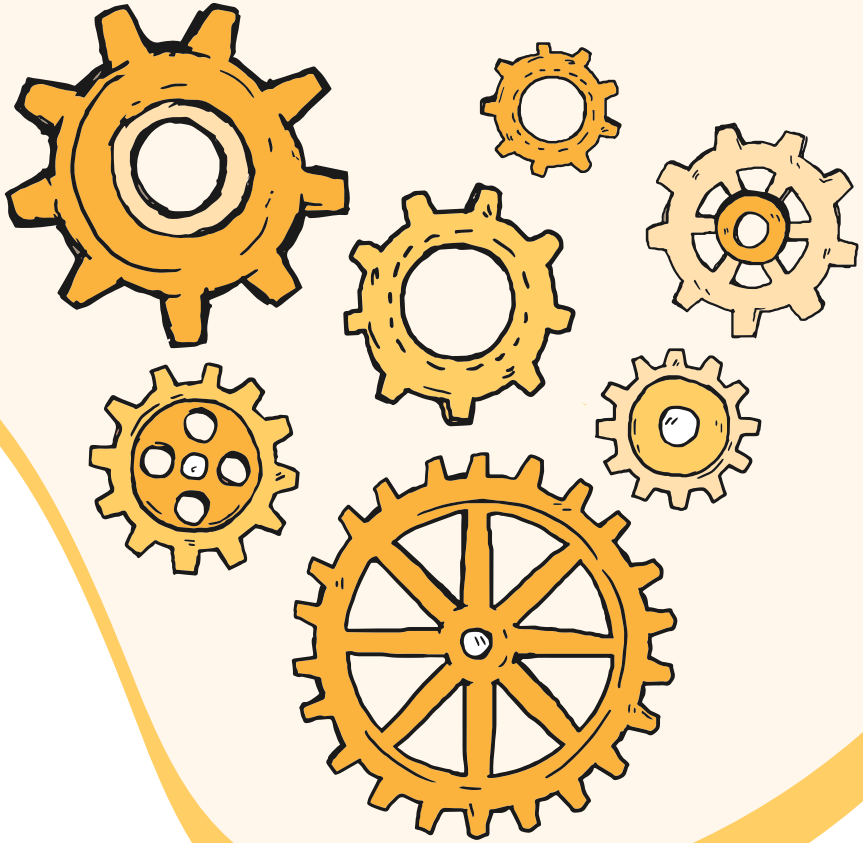
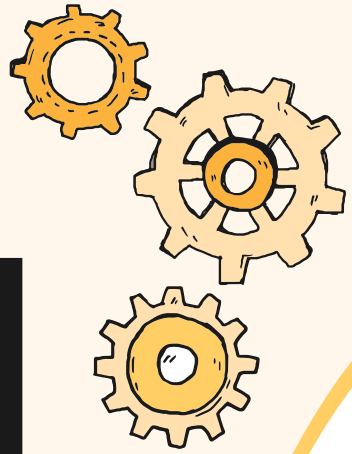
Huffman Coding

Huffman Sıkıştırma Kodlamasına Bir Bakış

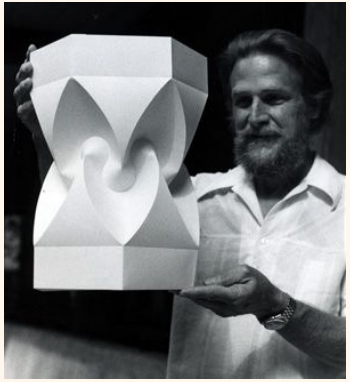


01



Huffman Kodlamanın Tarihi

Huffman Kodlama Tarihi



David Huffman, 1978

David Huffman (1925-1999) 1951'de MIT'de Bilişim Kuramı (Information theory) dersinde öğrenciydi. Profesörü Robert Fano, öğrencilere bir final sınavı veya bir dönem ödevi yazma seçeneği sundu. Huffman finale girmek istemediği için dönem ödevi üzerinde çalışmaya başladı. Makalenin konusu, en verimli (en uygun) kodu bulmaktı.

Profesör Fano'nun öğrencilerine söylemediği şey, bunun açık bir problem olduğu ve problem üzerinde kendisinin çalıştığı gerçeğiydi. Huffman, sorun üzerinde çok zaman harcadı ve çözümü bulduğunda neredeyse pes etmek üzereydi. Keşfettiği kod optimaldi, yani mümkün olan en düşük ortalama mesaj uzunluğuna sahipti. Fano'nun bu problem için geliştirdiği yöntem her zaman optimal bir kod üretmedi. Bu nedenle Huffman, profesöründen daha iyi yaptı. Ve "A Method for the Construction of Minimum-Redundancy Codes" adlı makalesi yayımlandı.

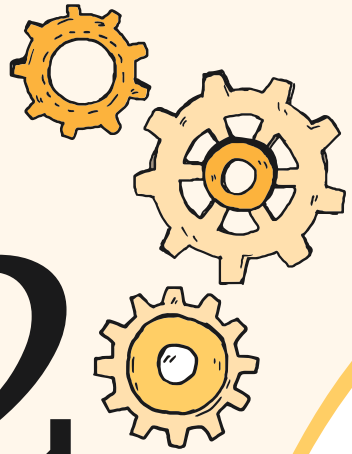
İlerleyen zamanlarda Huffman, profesörünün bununla boğuştuğunu bilseydi, muhtemelen problemi denemeyeceğini söyledi.

Robert Fano, 1975

For more detailed information

Huffman, David A. "A method for the construction of minimum-redundancy codes." *Proceedings of the IRE* 40.9 (1952): 1098-1101.

02



Huffman Kodlaması Nedir?

Huffman Kodlaması Nedir?

Huffman kodlaması bir **kayıpsız** veri sıkıştırma yöntemidir. Veride hiç kayıp olmadan veriyi sıkıştırıp tekrar açmayı sağlar.

Verinin içindeki karakterleri **frekanslarına** göre kodladığı için depolama avantajı sağlar. Frekansı yüksek olan yani daha sık kullanılan karakterler daha az, nadir kullanılan karakterler daha fazla yer kaplar. Her ne kadar günümüzde daha karmaşık ve etkili yöntemler kullanılsa da modern sıkıştırma yöntemlerinin temelinde Huffman kodlaması vardır. Huffman, bu yöntemin tek bir Unicode bloğundan oluşan veriler için en iyi yöntem olduğunu kanıtlamıştır.

Huffman kodlaması bir aç gözlü (**greedy**) algoritmadır. Aç gözlü algoritmalar; sonraki hamleyi düşünmeden, o anki en iyi hamleyi seçer.

Entropi

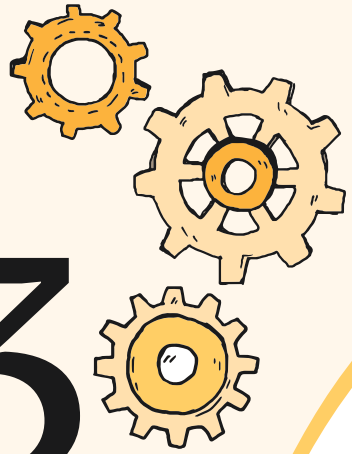
Huffman kodlama, **entropi bazlı bir algoritmadır**, yani verideki her sembolün bulunma olasılığı analiz edilerek kodlama yapılır.

Entropi, her sembolün veya semboller kümesinin öz bilgisinin ağırlıklı ortalamasıdır.

Ortalama sembol uzunluğu entropiye eşit ise, bulunabilecek en iyi kod bulunmuş demektir.

Entropi (Claude Elwood Shannon)

$$H = \sum_{i=1}^{n} P(si) \log_2 \frac{1}{P(si)}$$



03 Nasıl Çalışır?

Encoding / Decoding

Nasıl Çalışır?

Temel olarak iki aşamadan oluşur;

1- Huffman ağacı oluşturma.

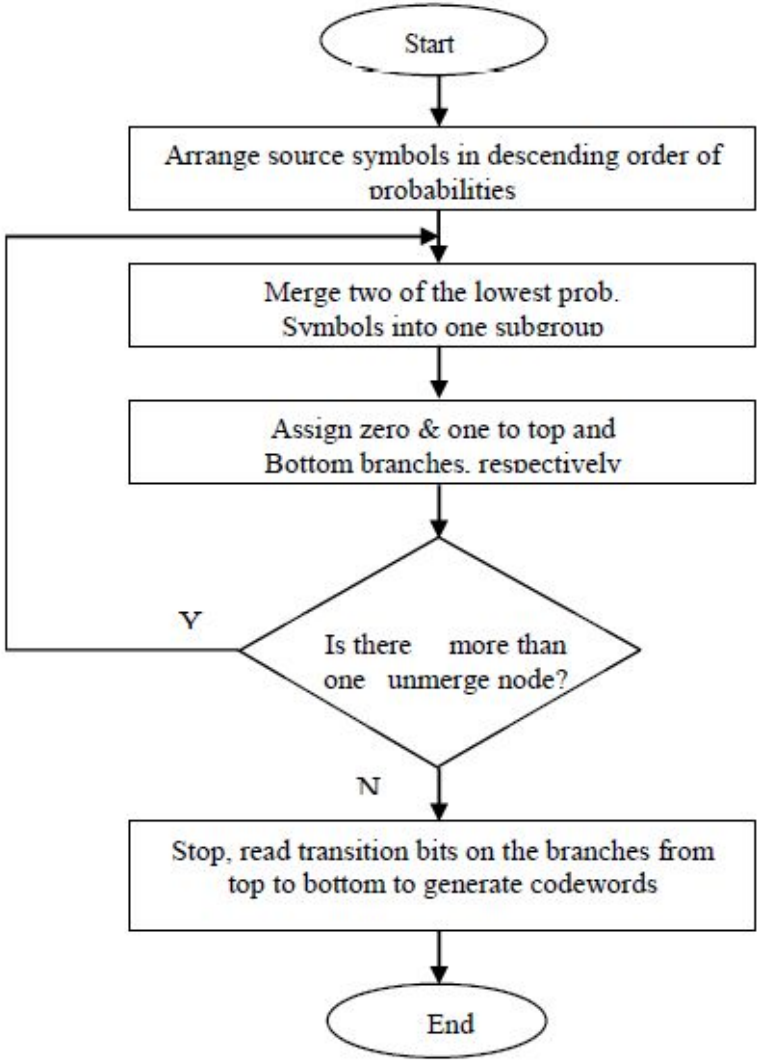
2- Ağacı gezerek karakterlere kod atamak.

-- Her farklı karakter için bir yaprak düğüm oluşturulur.

-- Frekansı en az olan iki düğüm seçilir ve bu frekans toplamlarıyla yeni bir düğüm oluşturulur.

-- Kök düğüme ulaşana kadar tekrar edilir.

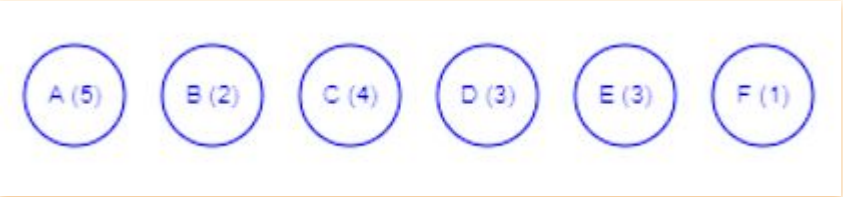
-- Ağacın sol kollarına “0”, sağ kollarına “1” atanır.


Start
Arrange source symbols in descending order of probabilities
Merge two of the lowest prob. Symbols into one subgroup
Assign zero & one to top and Bottom branches, respectively
Is there more than one unmerge node?
Y
N
Stop, read transition bits on the branches from top to bottom to generate codewords
End

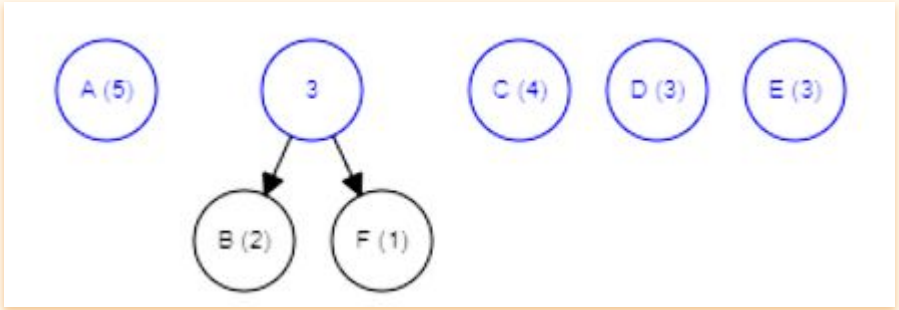
Örnek: BBCADEDDEEACCCFAAA

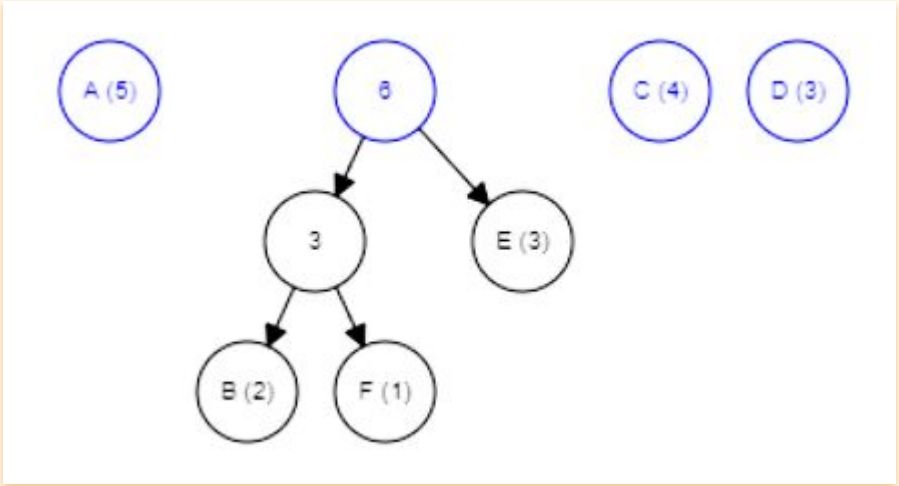
Karakter	Frekans
A	5
C	4
E	3
D	3
B	2
F	1

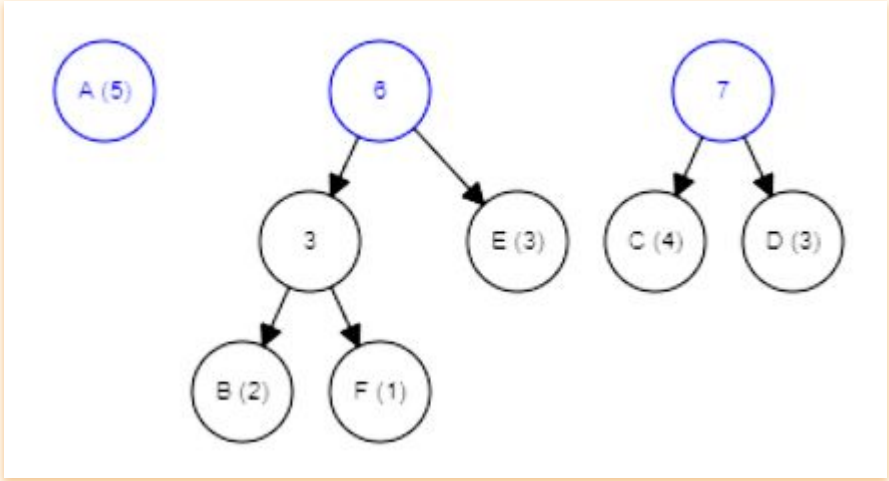
1- Her karakter için yaprak düğüm oluşturma

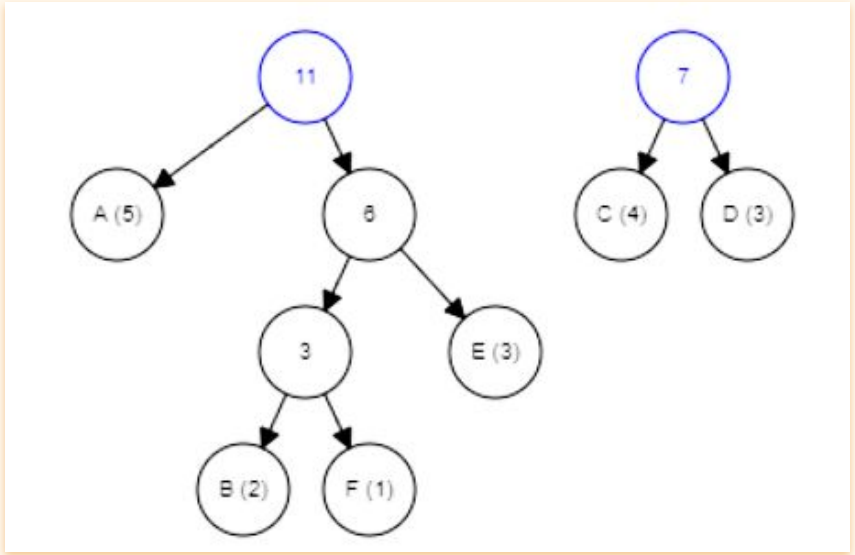


2- En az frekans değerlerine sahip iki düğümü birleştirme

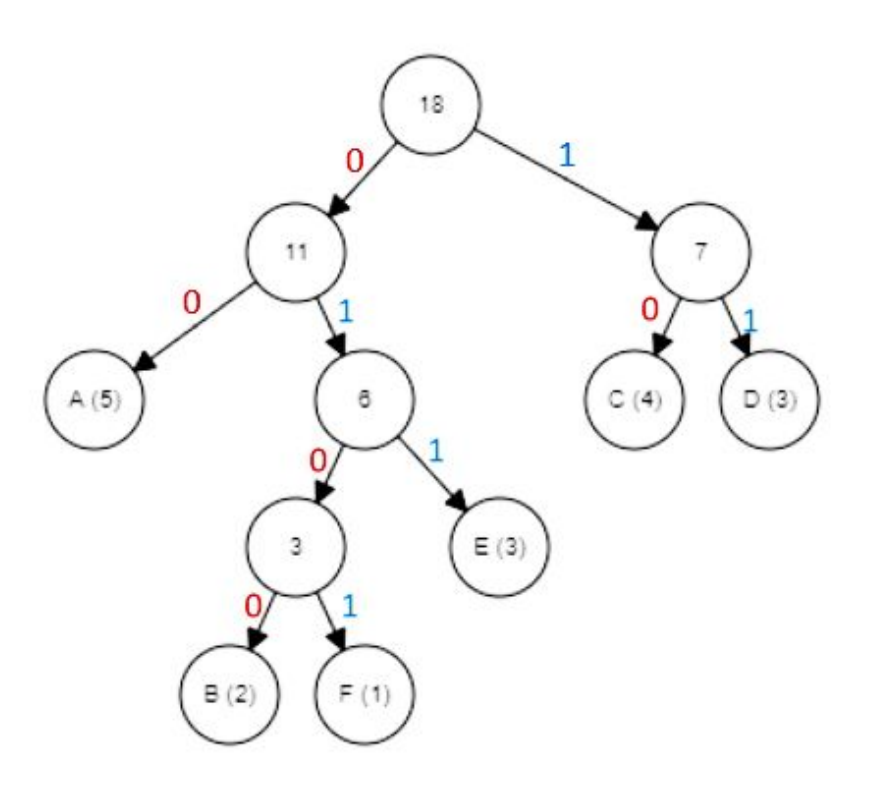





A (5)
6
7
3
E (3)
C (4)
D (3)
B (2)
F (1)


11
7
A (5)
6
C (4)
D (3)
3
E (3)
B (2)
F (1)

Tamamlanmış Ağaç



Sıkıştırılmamış	Sıkıştırılmış
A = 0100 0001	A = 00
B = 0100 0010	B = 0100
C = 0100 0011	C = 10
D = 0100 0100	D = 11
E = 0100 0101	E = 011
F = 0100 0110	F = 0101

Sıkıştırılmamış

BBCADEDDEEACCCFAAA =
01000010 01000010 01000011 01000001
01000100 01000101 01000100 01000100
01000101 01000101 01000001 01000011
01000011 01000011 01000110 01000001
01000001 01000001

Kapladığı alan = 144 bit

Sıkıştırılmış

BBCADEDDEEACCCFAAA =
01000100100011011111101101100101010
01010 00000

Kapladığı alan = 45 bit

A = 00 B = 0100 C = 10
D = 11 E = 011 F = 0101

Prefix kuralı, bir kodun diğerlerinin ön eki (prefixi) olmaması kuralına denir. Yoksa decode ederken karmaşa yaşanır

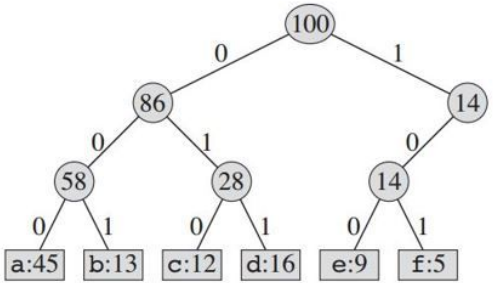
Görüldüğü üzere veri Huffman Kodlamasına göre sıkıştırılınca 99 bitlik alandan kazanç sağlanıyor.

Fixed-length encoding vs Variable-length encoding

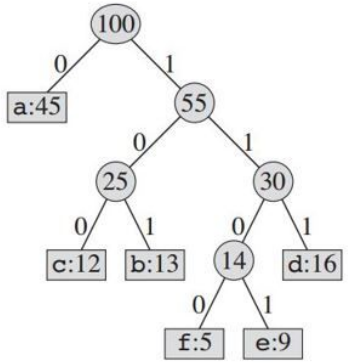
Bir Huffman ağacındaki ikiden fazla “sembol” aynı olasılığa sahip olduğunda, farklı birleştirme sıraları farklı Huffman kodları üretir.

Örneğin elimizde 100 karakterlik bir veri olduğunu düşünelim. Sıklıkları aşağıdaki gibi 6 farklı karakterimiz olsun.

Karakter	a	b	c	d	e	f
Sıklık	45	13	12	16	9	5



Fixed-length Codeword



Variable-length Codeword

Karakter	a	b	c	d	e	f
Sabit Uzunluklu Kodlama	000	001	010	011	100	101
Değişken Uzunluklu Kodlama	0	101	100	111	1101	1100

Sabit Uzunlukta kodlama - Her karaktere aynı sayıda bit kullanan bir ikili kod atanır. Kodlama işlemi kısalır yani zamandan kazanç sağlanır, ama hafızadan kaybedilir

Değişken Uzunluk kodlaması - Sabit uzunluklu kodlamanın aksine, bu şema verilen metindeki frekanslarına bağlı olarak karakterleri kodlamak için değişken sayıda bit kullanır. Hafızadan kazanç sağlanır.

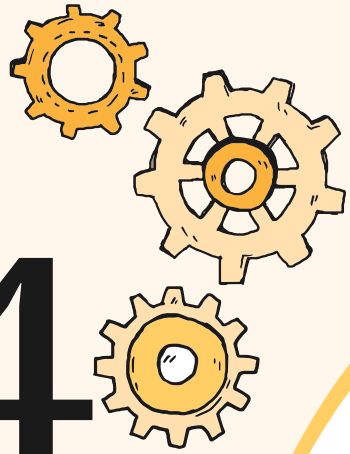
Eğer fixed-length (sabit uzunluklu) kodlamayı kullanırsak her karakter için 3 bite ihtiyacımız olacak ve bu yöntem bütün dosya için 300 bit kullanmayı gerektirecek.

Diğer kodlamada ise 45 X 1 + 13 X 3 + 12 X 3 + 16 X 3 + 9 X 4 + 5 X 4 = 224 bitlik yere ihtiyacımız olacak. Ve bu yöntem ile belleği % 25,3 daha az kullanmış oluyoruz.

Dinamik Huffman Kodlaması

Kodlama yapılırken daha önce karşılaşılan sembollerin istatistiği temel alınarak Huffman kodlaması **tek geçişli** hale getirilir. Tek geçişli prosedürün yararı, kaynağın gerçek zamanlı olarak kodlanabilmesidir, ancak yalnızca tek bir kayıp tüm kodu bozduğundan iletim hatalarına daha duyarlı hale gelir.

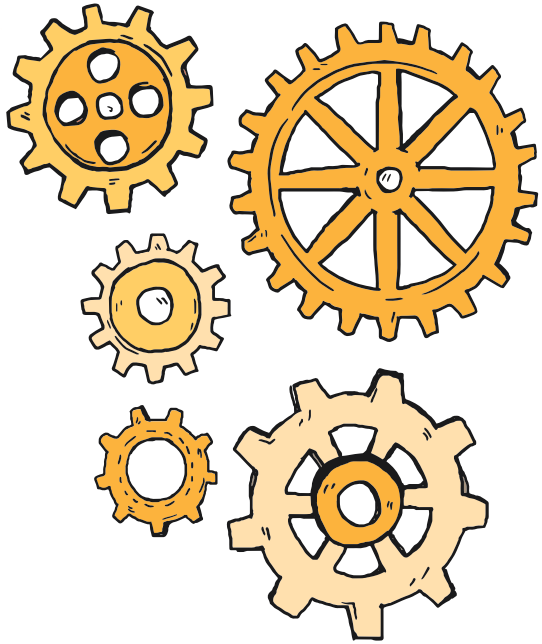
(İleride gösterilecek örneklerde iki geçişli huffman kodlaması kullanıldı)



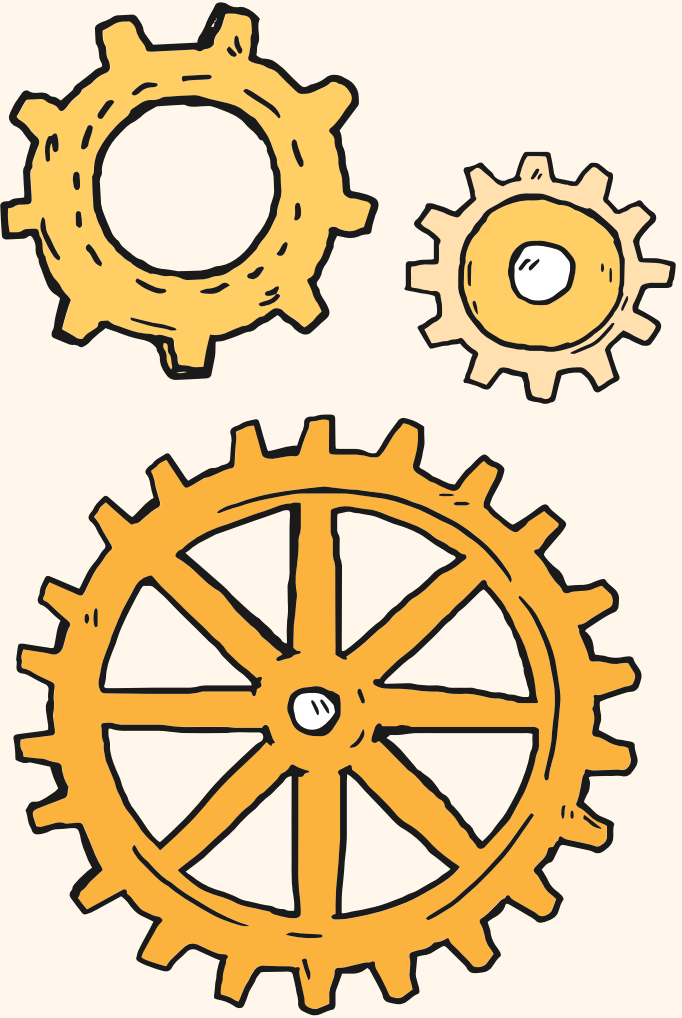
04 Pseudocode

Algoritmayı nasıl
implemente ederiz?

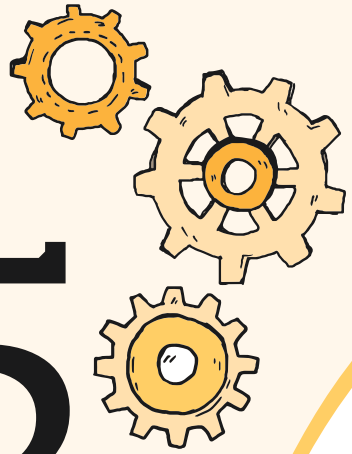
Building Huffman Tree



```
HUFFMAN(C)
1  n ← |C|
2  Q ← C
3  for i ← 1 to n − 1
4        do allocate a new node z
5           left[z] ← x ← EXTRACT-MIN(Q)
6           right[z] ← y ← EXTRACT-MIN(Q)
7           f[z] ← f[x] + f[y]
8           INSERT(Q, z)
9  return EXTRACT-MIN(Q)          ▷ Return the root of the tree.
```



1. Priority queue oluşturulur. **(Min heap yapısı)**
2. Internal node oluşturulur **(2 child node ve toplam frekans içerir.)**
3. Kuyruktan en küçük iki eleman extract edilir.
4. ***3. Adımda*** çıkarılan iki node, ***2. adımda*** oluşturulan internal node içerisine eklenir. Toplam frekans yazılır.
5. Oluşturulan internal node tekrar priority kuyruğuna eklenir.
6. ***2-5 adımları*** kuyrukta tek eleman kalana değin tekrarlanır ((n-1) tane => n tane leaf node binary tree de bağlanmak için n-1 internal node ihtiyaç duyar.)



05

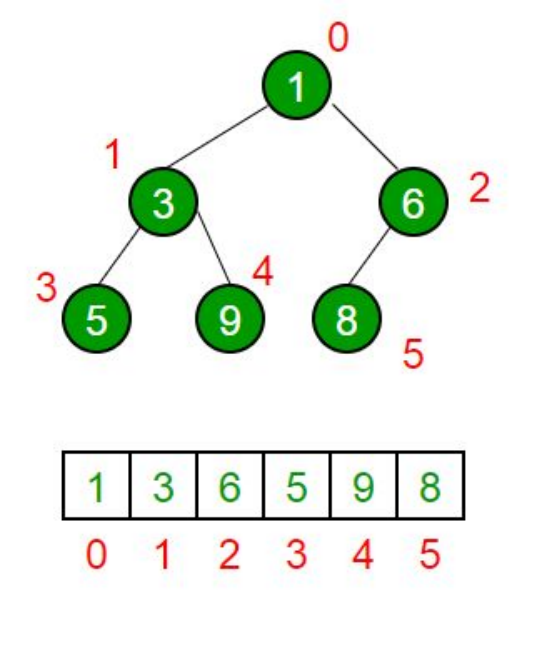
Time Complexity Analysis

Big-O Analysis of the Huffman Encoding Algorithm

Min-Heap Kullanımı

Huffman Coding algoritması anlatılırken bahsedilen min-priority queue yapısını bir çok şekilde kurmak mümkün fakat biz kendi implementasyonumuzda binary min-heap kullandık.

Min-heap her parent düğümün kendi child düğüm değerlerinden küçük veya eşit olduğu bir veri yapısıdır.



Big-O Analysis of Min Heap

01

Building a heap

O(nlog(n))

(William's Method)

02

Insertion

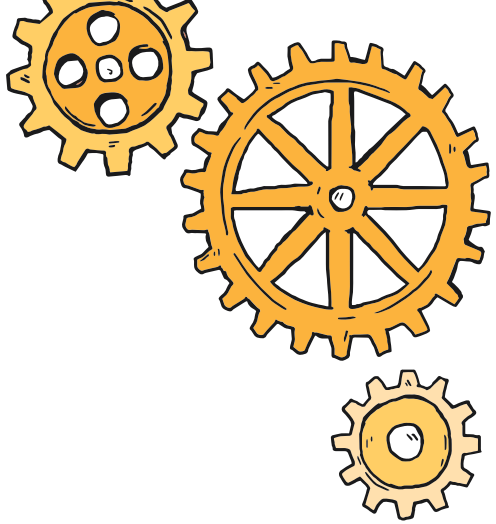
O(log(n))

03

Extraction min val

O(log(n))

Time Complexity Analysis



```
HUFFMAN(C)
1  n ← |C|
2  Q ← C
3  for i ← 1 to n − 1
4       do allocate a new node z
5          left[z] ← x ← EXTRACT-MIN(Q)        O(logn)
6          right[z] ← y ← EXTRACT-MIN(Q)       O(logn)
7          f[z] ← f[x] + f[y]                  O(1)
8          INSERT(Q, z)                        O(logn)
9  return EXTRACT-MIN(Q)                       ▷ Return the root of the tree.
```

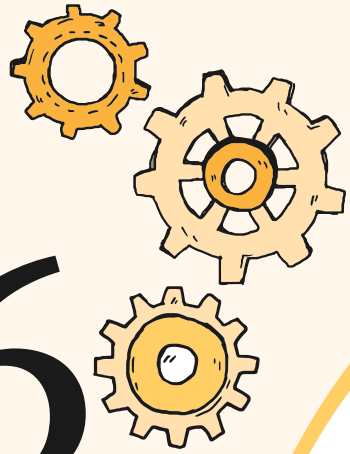
n-1 kere

$T = (n-1)(3 \cdot O(\log n) + O(1))$

Big O of Huffman Coding => $O(n \log n)$

For more detailed information

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 16.3, pp. 385–392.



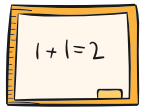
06

Kullanım Alanları

Huffman Kodlaması Kullanım Alanları

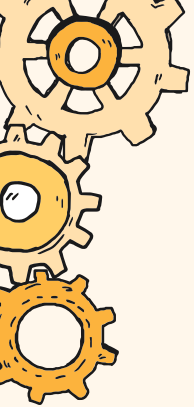
Ses

Metin



Resim

Video

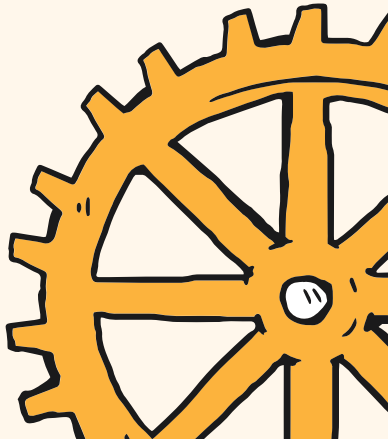


Gerçek Hayattaki Kullanım Alanları

Her tür dosya formatı veriyi sıkıştırarak tutar. Huffman Algoritması ile her veriyi kodlamak mümkün.

Huffman, geniş anlamda müzik, görüntü ve belirli iletişim protokollerini kodlamak için kullanılır.

1. GZIP, PKZIP (winzip) ve BZIP2 gibi dosya arşivleme programlarında ve DEFLATE algoritmasında,
2. Kayıpsız JPEG, MPEG, PNG görüntü dosyalarında, Flate-ZIP algoritmasında
3. MP3 ve FLAC dosyalarında Huffman kodlamasına rastlanır.



DEFLATE ALGORİTMASI

1. İlk olarak PKZIP ile ortaya çıktı. Phil Katz tarafından geliştirildi.
2. Huffman encoding + LZ77 sıkıştırma birleşimi
3. Ücretsiz olduğu için birçok alanda kullanıldı:
   Winzip, Jar dosyaları, PNG fotoğraf sıkıştırması etc.
   C (Zlib), C++ (7-Zip/AdvanceCOMP), Pascal (Paszlib), Java (java.util.zip), .NET 2.0 (Zip-Ada)

4. Deflate algoritması Huffman ağacının etkili kodlama yapamayacak kadar büyüdüğünü gördüğünde, yeni bir Huffman ağacı oluşturmak için o bloğu sonlandırarak yeni bir blok başlatır.

   Sıkıştırılacak olan veri birbirini takip eden bloklar kümesi olarak düşünülür.
   Her blok için oluşturulan Huffman ağacı bir önceki ve bir sonraki bloktan bağımsızdır.
   Sıkıştırılabilen blokların büyüklüğü değişkendir.

Resim Sıkıştırma

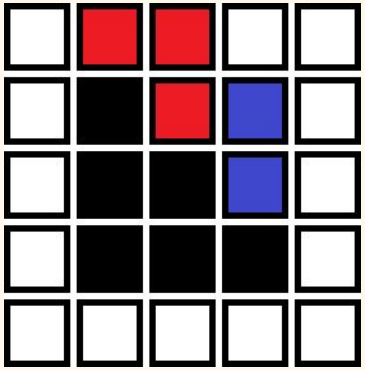
Kayıpsız (lossless) JPEG sıkıştırması, Huffman algoritmasını saf haliyle kullanır.

Kayıpsız JPEG, büyük tıbbi ekipman üreticileri tarafından desteklenen **DICOM** standardının bir parçası olarak tıp cihazlarında yaygındır.

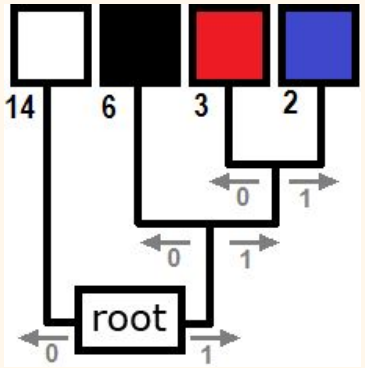
- Ultrason makineleri,
- nükleer rezonans görüntüleme makineleri,
- MRI makineleri ve
- elektron mikroskoplarında kullanılır.

Kayıpsız JPEG algoritmasının varyasyonları, bilgileri kaybetmeden bir kameranın görüntü sensöründen veri kaydettiği için fotoğraf meraklıları arasında popüler olan RAW formatında da kullanılır.

Orijinal (sıkıştırılmamış) görüntü temsili 8 bit / piksel kullanır. Örneğin; **5x5** piksellik bir görüntü için **200 bit** alan harcanır.



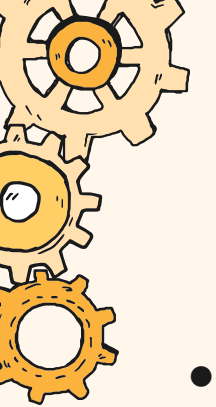
Önce görselde kaç farklı renk pikseli olduğu sayılır, sonra bulunma olasılığına göre sıralanır. Renklerin renk kodları saklanır, böylece decode edilebilir.



Şimdi kökten en uzaktaki renkler en az sıklıkta olacak şekilde bir ağaç inşa ederek renkleri bir araya getiriyoruz. Sağda renklerin frekansları ve ağaçtan elde edilen bit kodları bulunmaktadır.

14 x 1 + 6 x 2 + 3 x 3 + 2 x 3 = **41bit**

color	freq.	bit code
	14	0
	6	10
	3	110
	2	111



Video Sıkıştırma

HUFFYUV

- Ben Rudiak-Gould tarafından 90'lı yıllarda geliştirilmiş popüler kayıpsız hareketli görüntü (video) sıkıştırma algoritması.
- Sıkıştırma ve açma işlemlerinde hızlı bir algoritmadır.
- Her çerçeveyi bir önceki çerçeveden bağımsız olarak kodlar (Çerçeve-içi (intra-frame) kodlama). Kullandığı model Kayıpsız JPEG ile benzerdir. Huffman kodlaması temel olarak kullanılır.
- Kaynak kodu açık, dağıtımı serbesttir.

Dinlediğiniz İçin Teşekkürler

Zeynep Sena Yurdadur
Zeynep Gök
Eda Nur Var
Zeynep Çetin

https://zeynep-sena.github.io/Huffman-Coding