BM 210 Algoritma Analizi
(Analysis of Algorithms)

Hazırlayan: M.Ali Akcayol
Gazi Üniversitesi
Bilgisayar Mühendisliği Bölümü

Konular

- Veri Sıkıştırma
- Kayıplı ve Kayıpsız Sıkıştırma
- Sabit Genişlikli / Değişken Genişlikli Kodlama
- Huffman Algoritması (Greedy Algorithms)
  - Kodlama ağacının oluşturulması
  - Kodlamanın yapılması
  - Sıkıştırılmış bilginin tekrar elde edilmesi

Veri Sıkıştırma

Sıkıştırma niçin kullanılır ?

1- Alan gereksinimini azaltmak için

- Hard disklerin boyutu büyümekle birlikte, yeni programların ve data dosyalarınında boyutu büyümektedir ve alan ihtiyacı artmaktadır
- 3.5" floppy diskler hala kullanılmaktadır ve alanları çok azdır

2- Zaman ve bant genişliği gereksinimini azaltmak için

- Birçok program ve dosya internetten download edilmektedir
- Birçok kişi düşük hızlı bağlantıyı kullanmaktadır
- Sıkıştırılmış dosyalar transfer süresini kısaltmakta ve birçok kişinin aynı server'ı kullanımına izin vermektedir

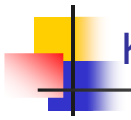
Kayıplı ve Kayıpsız Sıkıştırma

1- Kayıplı sıkıştırma

- Bilginin bir kısmı geri elde edilmez (MP3, JPEG, MPEG)
- Genellikle ses ve görüntü uygulamalarında kullanılır
- Çok büyük ses ve görüntü dosyalarında çok iyi sıkıştırma yapar ve kullanıcı kalitedeki azalmayı farketmez

2- Kayıpsız sıkıştırma

- Orijinal dosya tekrar tam olarak elde edilir (D(C(X)) = X, burada C sıkıştırılan dosyayı, D ise açılan dosyayı ifade eder)
- .txt, .exe, .doc, .xls, .java, .cpp vs. gibi dosyalarda kullanılır
- Ses ve görüntü dosyalarındada kullanılabilir ancak kayıplı sıkıştırma kadar yüksek sıkıştırma oranına sahip olmaz



Kayıpsız Sıkıştırma

Kayıpsız sıkıştırma temel olarak 3 algoritma altında toplanır

- **Huffman** – her karakter için değişken genişlikli bir kelime kodu (codeword) kullanır
- LZ77 - "sliding" window kullanır ve bir grup karakteri bir zamanda sıkıştırır
- LZ78 / LZW – daha önce karşılaşılan işaretleri saklayan bir sözlük kullanır ve bir grup karakteri aynı anda sıkıştırır

Kayıpsız sıkıştırma uygulamaları

- unix compress, gif: LZW
- pkzip, zip, winzip, gzip: Huffman + LZ77

Kodlama Teorisi (Coding Theory) Fixed Length ve Variable Length

Toplam 1.000.000.000 karakterden oluşan bilginin sadece 6 harften {a,b,c,d,e,f} oluştuğunu varsayalım. Karakterlerin kullanım sıklıkları ise aşağıdaki gibi olsun.

a	*b*	*c*	*d*	*e*	*f*	
.45	.13	.12	.16	.09	.05	**Frekans**
000	001	010	011	100	101	**Fixed length**
0	101	100	111	1101	1100	**Variable length**

Fixed length $3 \bullet 1.000.000.000 = 3.000.000.000$ bits $\approx$ **3GB**

Variable length $(.45 \bullet 1 + .13 \bullet 3 + .12 \bullet 3 + .16 \bullet 3 + .09 \bullet 3 + .05 \bullet 3) \bullet 1.000.000.000$
$= 2.240.000.000$ bits $\approx$ **2.24GB**

Karakterlerin Kodunu Çözme (Decode)

E	0
T	11
N	100
I	1010
S	1011

11010010010101011

T E N N I S

Prefix code

E	0
T	10
N	100
I	0111
S	1010

100100101010

Belirsiz

Prefix code

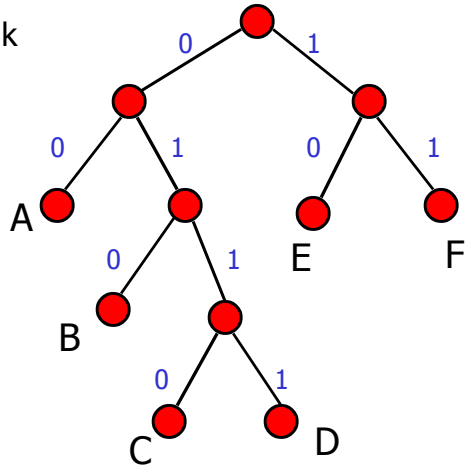
- Bir codeword başka bir codeword'ün prefix'i olamaz
- Kod çözülmesinde teklik olmalı

A	00	1	00
B	010	01	10
C	011	001	11
D	100	0001	0001
E	11	00001	11000
F	101	000001	101

Prefix code ve ikilik ağaç

- Prefix code'ların ikilik ağaçla gösterimi

A	00
B	010
C	0110
D	0111
E	10
F	11



Kodlama için gereken boyut

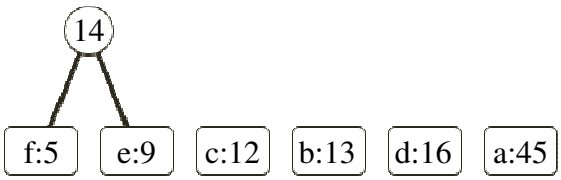
- $f(c)$ – c karakterinin dosya içindeki frekansını gösterirsin
- $d_T(c)$ – c karakterinin bulunduğu yaprağın seviyesini gösterirsin (kod boyutu)
- Kodlama için gereken dosyanın bit olarak boyutu $B(T)$ aşağıdaki gibi ifade edilir

$$B(T) = \sum_{c \in C} f(c) d_T(c)$$

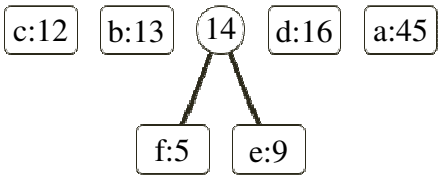
Kodlama ağacının oluşturulması

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

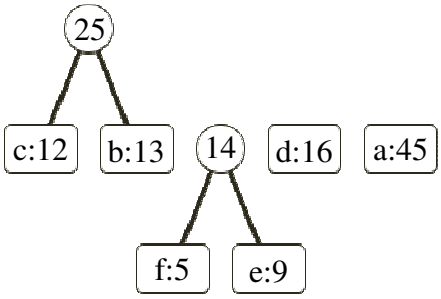
Kodlama ağacının oluşturulması



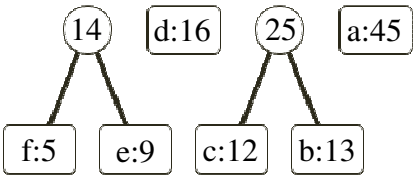
Kodlama ağacının oluşturulması



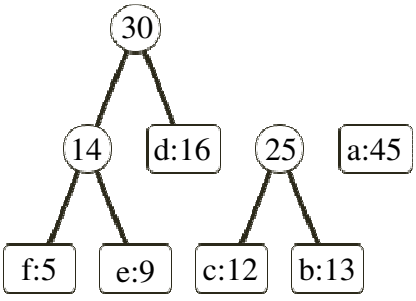
Kodlama ağacının oluşturulması



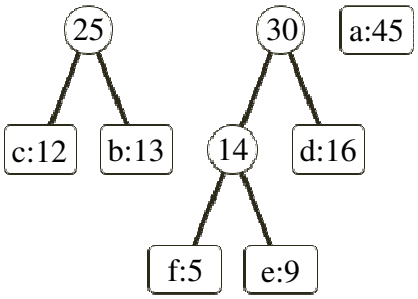
Kodlama ağacının oluşturulması



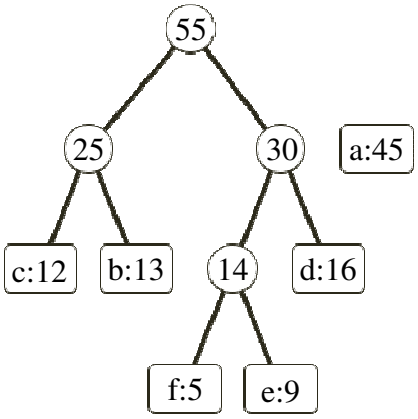
Kodlama ağacının oluşturulması



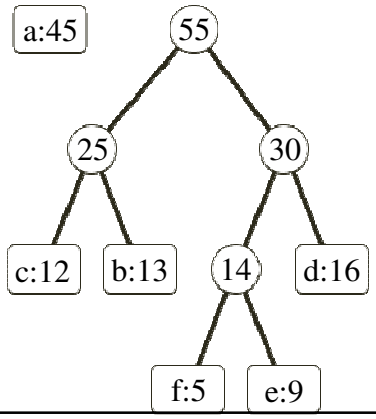
Kodlama ağacının oluşturulması



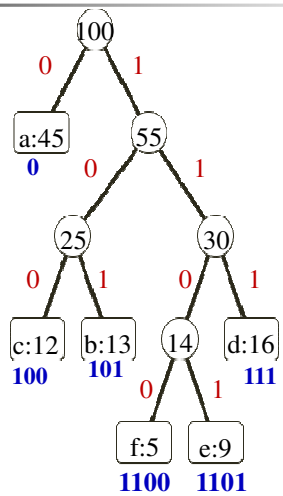
Kodlama ağacının oluşturulması



Kodlama ağacının oluşturulması



Kodlama ağacının oluşturulması



Huffman algoritması

```
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.
```

Farklı karakter sayısı

Min-Priority Queue

En düşük frekanslı iki node

İki node'un toplamı yeni node

Yeni node kuyruktaki yerine ekleniyor

Çalışma süresi

- Uygun veri yapısı olarak min-heap kullanılabilir
- Heap oluşturma O(*lg n)* ve for döngüsünde *n-1* adet işlem yapılır
- Toplam çalışma süresi O(*n lg n*) olur

Dönem Ödevi

- Verilen bir metin dosyasını sıkıştıran programı Huffman Algoritmasıyla yazınız
- Program C#'ta hazırlanacak kullanıcı arayüzüne sahip olacaktır
- Hazırlanan proje 17 mayıs tarihinde sınıfta sunulacaktır