

אוריאל  
דוד  
בשומרון  
הוא רשע של צא ולכן סיפא מתנדות היא צ  
אם ה א ח מילת קוד מקורית - הקוד לא יחזור

נכתב ע"י צבי מינץ ויואב גרוס

מרצה: פרופ' דנה שפירא

מבוא: (בסוף רוצים קוד  $K(C) \leq 1$  ו- $E(C, P)$  מינמלי)

אורך מילת הקוד הממוצעת:  $E(C, P) = \sum_{i=1}^n p_i \cdot |c_i|$

אינפורמציה:  $I(s_i) = -\log_2 p_i$

אנטרופיה (חסם תחתון):  $H(P) = -\sum_{i=1}^n p_i \log_2 p_i$

כלומר  $\forall C: H(P) \leq E(C, P)$

קראפט: אם  $K(C) = \sum_{i=1}^n 2^{-|c_i|} \leq 1$  אזי קיים קוד  $C'$

כך ש-  $|C| = |C'|, E(C, P) = E(C', P)$  ו- $C'$  פרפיקסי

אם  $K(C) > 1$  אזי לא קיים קוד פרפיקסי עם האורכים

קוד שלם:  $K(C) = 1$ . קוד שלם יוצר עץ מלא

קוד חסר רישות  $\Leftarrow$  קוד  $UD$  (ניתן לפענוח בצורה יחידה)

קוד מיידי  $\Leftrightarrow$  קוד חסר רישות, קוד  $UD \nLeftarrow$  קוד חסר רישות

מבחן לזיהוי קוד יחודי: (אם"ם)

- Examine all pairs of codewords:
- Construct a list of all codewords.
- If there exist a codeword, a, which is a prefix of another codeword, b, add the dangling suffix to the list (if it is not there already), until:
  - You get a dangling suffix that is an original codeword  $\rightarrow$  the code is not UD
  - There are no more unique dangling suffixes  $\rightarrow$  the code is UD

מודלים: (ללמוד ולעשות הנחות על מבנה הטקסט)

יש 2 סוגים של דחיסות - Lossy, Lossless

ex.Video ex.Text

קוד סטטי:  $p_i = \frac{1}{|\Sigma|}$  כאשר  $|\Sigma| = 256 (Ascii)$  (אין prelude)

קוד סמי סטטי:  $p_i = \frac{1}{|\Sigma|}$  כך  $\#$  תווים שונים בטקסט  $|\Sigma|$

ולכן  $E(C, P) = H(P) + \frac{8 \cdot |\Sigma| + 8}{|Text Size|}$  (8 יכול להשתנות - כאן מתבסס על מודל אסקי)

סמי סטטי עם הסתברויות עצמיות:  $p_i = \frac{v_i}{m}$  כאשר  $v_i$  מספר

ההופעות של  $s_i$  בטקסט בגודל  $m$ . נציין ב- $|p|$  כמה עולה לעביר

הסתברות ולכן  $E(C, P) = H(P) + \frac{8 \cdot |\Sigma| + |\Sigma| \cdot |p| + 8}{|Text Size|}$

קודים: (רוצים למצוא קודים בסדר גודל לוגרתימי)

קודים מסדר ראשון: ישנה תלות בין כל זוג תווים

Unary Code:  $Unary(X) = 1^{X-1} \cdot 0$

Binary Code: כל סימן מייצג מילת קוד באורך  $\lceil \log_2 n \rceil$

Minimal Binary Code: אם יש  $n$  תווים אזי  $n - 2^{\lceil \log_2 n \rceil}$  יהיו

באורך  $\lceil \log_2 n \rceil$  ביטים ושאר  $2^{\lceil \log_2 n \rceil} - 2n$  באורך  $\lceil \log_2 n \rceil$

Elas:  $C_\gamma$  (גמה): החלק הראשון זה Unary של מספר הביטים

ליצוג  $X$  והחלק השני זה קוד הבינארי של  $X$  ללא '1' הראשון

$C_\delta$  (דלתא): החלק הראשון זה  $C_\gamma$  של מספר הביטים ליצוג  $X$

והחלק השני זה קוד הבינארי של  $X$  ללא '1' הראשון

דרי גודל:

$C_\gamma: 1 + 2 \cdot \lceil \log_2 x \rceil$  bits |  $C_\delta: 1 + 2 \lceil \log_2 \log_2 2x \rceil + \lceil \log_2 x \rceil$  bits

Golomb encode(x,b)

$q \leftarrow (x-1) \div b;$

$l = \text{Unary\_encode}(q+1);$

$n = \text{Minimal\_binary\_encode}(r,b); \text{return } l \cdot n$

Rice code: נבחר את  $b$  להיות  $2^K$  עבור  $k$  כלשהו.

חלק ראשון: קוד אונרי של ( $1 + (x-1)$  right shift  $k$  bits)

Shannon-Fano Algorithm: (לא תמיד הכי יעיל)  $X-1$

מיינ את ההסתברויות בסדר יורד.

כל עוד קבוצת ההסתברות מכילה יותר מתו 1:

נחלק את הקבוצה ל2 חלקים כך שסכום ההסתברויות בכל

חלק פחות או יותר זהה. -קבוצה אחת תקבל 1, והשנייה 0.

קוד הפמן: אלגוריתם לבניית עץ קנוני

אחרי הרצת הפמן קיבלנו את כל האורכים  $(l_1, \dots, l_n)$

האורך של המילה הארוכה ביותר  $\leftarrow \text{maxlength}$

for  $i = 1$  to  $\text{maxlength}$  do {num[i] = 0}

for  $i = 1$  to  $n$  do {num[l<sub>i</sub>] ++}

firstcode[ $\text{maxlength}$ ]  $\leftarrow 0$

for  $i = \text{maxlength} - 1$  downto 1 do

5.1. firstcode[i]  $\leftarrow$  (firstcode[i + 1] +

num[i + 1])/2

6. for  $i = 1$  to  $\text{maxlength}$  do

6.1. nextcode[i]  $\leftarrow$  firstcode[i]

7. for  $i = 1$  to  $n$  do

7.1. codeword[i]  $\leftarrow$  nextcode[l<sub>i</sub>]

7.2. symbol[l<sub>i</sub>, nextcode[l<sub>i</sub>] - firstcode[l<sub>i</sub>]]  $\leftarrow i$

7.3. nextcode[l<sub>i</sub>] ++

אלגוריתם לפענוח של הפמן קנוני ע"י הטבלאות:

1.  $v \leftarrow \text{nextInputBit}()$  קבלת הביט הראשון

2.  $i = 1$

3. while  $v < \text{firstcode}[i]$  do

3.1.  $v \leftarrow 2v + \text{nextInputBit}()$

3.2.  $i ++$

// יצאנו מהלולאה אזי  $v$  מכילה מילת קוד תקינה

4. return symbol[i, v - firstcode[i]]

אלגוריתם לעדכון עץ הפמן דינמי:

q = leaf(x<sub>i</sub>)

if (q is the 0-node)

replace q by a parent 0-node with two 0-node children;

q = left child;

if q is a sibling of a 0-node

interchange q with the highest numbered leaf

of the same weight;

increment q's weight by 1;

q = parent of q

while q is not root

interchange q with the highest numbered node

of the same weight;

increment q's weight by 1;

q = parent of q

increment q's weight by 1

הגדרות לאלגוריתמים של עץ skeleton

base(i) = 2(base(i - 1) + n<sub>i-1</sub>)

seq(i) = seq(i - 1) + n<sub>i-1</sub>

diff(i) = base(i) - seq(i)

Decoding Algorithm

1. tree\_pointer <- root

2. i <- 1

3. start <- 1

4. while i < length\_of\_string

4.1. if string[i] = 0

4.1.1. tree\_pointer <- left(tree\_pointer)

4.2. else tree\_pointer <- right(tree\_pointer)

4.3. if value(tree\_pointer) > 0

4.3.1. codeword <- string[start ...

(start + value(tree\_pointer) - 1)]

4.3.2. output <- table[l(codeword) -

diff(value(tree\_pointer))]

4.3.3. tree\_pointer <- root

4.3.4. start <- start + value(tree\_pointer)

4.3.5. i <- start

4.4. else i ++

1. tree\_pointer <- root

2. i <- start <- 1

3. while i < length\_of\_string

3.1. if string[i] = 0 tree\_pointer <-

left(tree\_pointer)

3.2. else tree\_pointer <- right(tree\_pointer)

3.3. if value(tree\_pointer) > 0

3.3.1. len <- value(tree\_pointer)

3.3.2. codeword <- string[start ... (start + len - 1)]

3.3.3. if flag(tree\_pointer) = 1 and

2l(codeword) >= base(len + 1)

3.3.3.1. codeword <- string[start ... (start + len)

3.3.3.2. len ++

3.3.4. output <- table[l(codeword) - diff(len)]

3.3.5. tree\_pointer <- root

3.3.6. i <- start <- start + len

3.4. else i ++

Skeleton

low\_bound(s<sub>i</sub>)  $\leftarrow \sum_{j=1}^i p_j$   
high\_bound(s<sub>i</sub>)  $\leftarrow \sum_{j=i}^n p_j$

low <- 0.0

high <- 1.0

while input symbols remain{

range <- high - low

Get symbol

high <- low + high\_bound(symbol) \* range

low <- low + low\_bound(symbol) \* range

}

Output any value in [low, high)

אלגוריתם לפענוח קוד אריתמי בטווח [0,1]:

encoded <- Get (encoded number)

do{ Find symbol whose range contains encoded

Output the symbol

range <- high(symbol) - low(symbol)

encoded <- (encoded - low(symbol))/range

} until (EOF)

קידוד - אריתמי [0,1]

low <- 0.0

high <- 1.0

while input symbols remain{

range <- high - low

Get symbol

high <- low + high\_bound(symbol) \* range

low <- low + low\_bound(symbol) \* range

}

Output any value in [low, high)

אלגוריתם לפענוח קוד אריתמי בטווח [0,1]:

encoded <- Get (encoded number)

do{ Find symbol whose range contains encoded

Output the symbol

range <- high(symbol) - low(symbol)

encoded <- (encoded - low(symbol))/range

} until (EOF)

קידוד - אריתמי [0,1]

low <- 0.0

high <- 1.0

while input symbols remain{

range <- high - low

Get symbol

high <- low + high\_bound(symbol) \* range

low <- low + low\_bound(symbol) \* range

}

Output any value in [low, high)

אלגוריתם לפענוח קוד אריתמי בטווח [0,1]:

encoded <- Get (encoded number)

do{ Find symbol whose range contains encoded

Output the symbol

range <- high(symbol) - low(symbol)

encoded <- (encoded - low(symbol))/range

} until (EOF)

קידוד - אריתמי [0,1]

low <- 0.0

high <- 1.0

while input symbols remain{

range <- high - low

Get symbol

high <- low + high\_bound(symbol) \* range

low <- low + low\_bound(symbol) \* range

}

Output any value in [low, high)

אלגוריתם לפענוח קוד אריתמי בטווח [0,1]:

encoded <- Get (encoded number)

do{ Find symbol whose range contains encoded

Output the symbol

range <- high(symbol) - low(symbol)

encoded <- (encoded - low(symbol))/range

} until (EOF)

קידוד - אריתמי [0,1]

low <- 0.0

high <- 1.0

while input symbols remain{

range <- high - low

Get symbol

high <- low + high\_bound(symbol) \* range

low <- low + low\_bound(symbol) \* range

}

Output any value in [low, high)

אלגוריתם לפענוח קוד אריתמי בטווח [0,1]:

encoded <- Get (encoded number)

do{ Find symbol whose range contains encoded

Output the symbol

range <- high(symbol) - low(symbol)

encoded <- (encoded - low(symbol))/range

} until (EOF)

קידוד - אריתמי [0,1]

low <- 0.0

high <- 1.0

while input symbols remain{

range <- high - low

Get symbol

high <- low + high\_bound(symbol) \* range

low <- low + low\_bound(symbol) \* range

}

Output any value in [low, high)

אלגוריתם לפענוח קוד אריתמי בטווח [0,1]:

encoded <- Get (encoded number)

do{ Find symbol whose range contains encoded

Output the symbol

range <- high(symbol) - low(symbol)

encoded <- (encoded - low(symbol))/range

} until (EOF)

קידוד - אריתמי [0,1]

low <- 0.0

high <- 1.0

while input symbols remain{

range <- high - low

Get symbol

high <- low + high\_bound(symbol) \* range

low <- low + low\_bound(symbol) \* range

}

Output any value in [low, high)

דחיסת דקדוק Re-Pair

כל עוד ישנו צמד תווים שחוזר על עצמו

בטקסט:

מצא את הזוג השכיח ביותר והחלף אותו

במשתנה חדש.

2. אם משתנה מופיע פעם אחת בצד ימין - נמחק אותו.

1. קוראים את הטקסט תו אחרי תו ומוסיפים ל.

בטקסט:

2. אם צמד תווים מופיע פעמיים בצד ימין הוא יוחלף

בכלל חדש.

1. קוראים את הטקסט תו אחרי תו ומוסיפים ל.

בטקסט:

2. אם צמד תווים מופיע פעמיים בצד ימין הוא יוחלף

בכלל חדש.

1. קוראים את הטקסט תו אחרי תו ומוסיפים ל.

בטקסט:

2. אם צמד תווים מופיע פעמיים בצד ימין הוא יוחלף

בכלל חדש.

1. קוראים את הטקסט תו אחרי תו ומוסיפים ל.

בטקסט:

2. אם צמד תווים מופיע פעמיים בצד ימין הוא יוחלף