נכתב ע"י צבי מינץ ויואב גרוס

מרצה :פרופ' דנה שפירא

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

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 $\forall C: H(P) \le E(C,P)$

קראפט: אם $K(C) = \sum_{i=1}^{n} 2^{-|c_i|} \le 1$ 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** ⇐ קוד **UD** (ניתן לפענוח בצורה יחידה)

קוד מיידי ⇔ קוד חסר רישות, קוד **UD** ⇐/ קוד **חסר רישות**

מבחן לזיהוי קוד יחודי: (אם"ם)

- Examine all pairs of codewords:

1. Construct a list of all codewords.
2. 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:
   - I. You get a dangling suffix that is an **original** codeword → the code is not UD
   - II. There are no more unique dangling suffixes → the code is UD

מודלים: (ללמוד ולעשות הנחות על מבנה הטקסט)

יש 2 סוגים של דחיסות – Lossless $ex.Text$ ו-Lossy $ex.Video$

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

קודים: (רוצים למצוא קודים בסדר גודל לוגרתמי)

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

Elias : C_γ **(גמה):** החלק הראשון זה Unary **של מספר הביטים** ליצוג X והחלק השני זה קוד הבינארי של X ללא '1' הראשון

C_δ **(דלתא):** החלק הראשון זה C_γ **של מספר הביטים** ליצוג X והחלק השני זה קוד הבינארי של X ללא '1' הראשון

סדרי גודל:

C_γ: $1 + 2 \cdot \lfloor \log_2 x \rfloor$ bits | C_δ: $1 + 2\lfloor \log_2 \log_2 2x \rfloor + \lfloor \log_2 x \rfloor$ bits

אלגוריתם לפענוח אליאס-D-ary:

1. הוסף שורש עם D-ary לעץ.
2. צמתים פנימיים עם התפלגות 0 נמחקים.
3. התחלת של הצמתים (בגודל k) לך מקשרים.
4. $x + n = 1 + (D-1)k$ מגדיר D שם אותיות לפי הפסים הבא המשוב.

Golomb_encode(x,b)

```
q←(x-1) div b;
r←x-q·b;
l = Unary_encode(q+1);
n = Minimal_binary_encode(r,b); return l·n
```

Golomb_decode(b)

```
q←Unary_decode()-1;
r←Minimal_binary_decode(b);
return r+q·b;
```

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

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

חלק שני: k הביטים הנמוכים של הייצוג הבינרי של X-1

Shannon-Fano Algorithm: (לא תמיד הכי יעיל)

מיין 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 $(l_1, \dots, l_n)$

1. $maxlength \leftarrow$ האורך של המילה הארוכה ביותר
2. $for\ i = 1\ to\ maxlength\ do\ \{num[i] = 0\}$
3. $for\ i = 1\ to\ n\ do\ \{num[l_i] + +\}$
4. $firstcode[maxlength] \leftarrow 0$
5. $for\ i = maxlength - 1\ downto\ 1\ do$

5.1. $firstcode[i] \leftarrow (firstcode[i+1] + num[i+1])/2$

6. $for\ i = 1\ to\ maxlength\ do$

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

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

7.1. $codeword[i] \leftarrow nextcode[l_i]$

7.2. $symbol[l_i, nextcode[l_i] - firstcode[l_i]] \leftarrow i$

7.3. $nextcode[l_i] + +$

אלגוריתם לפענוח של הפמן קנוני ע"י הטבלאות:

1. $v \leftarrow nextInputBit()$// קבלת הביט הראשון
2. $i = 1$
3. $while\ v < firstcode[i]\ do$

3.1. $v \leftarrow 2v + nextInputBit()$

3.2. $i + +$

אם יצאנו מהלולאה אזי v מכילה מילת קוד תקינה//

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

אלגוריתם לעדכון עץ הפמן דינמי:

```
q = leaf(x_t)
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
```

Skeleton

הגדרות לאלגוריתמים של עץ skeleton:

$base(i) = 2(base(i-1) + n_{i-1})$

$seq(i) = seq(i-1) + n_{i-1}$

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

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++
```

קידוד - אריתמטי [0,1]

$low\_bound(s_i) \leftarrow \sum_{j=1}^{i-1} p_j$

$high\_bound(s_i) \leftarrow \sum_{j=1}^{i} p_j$

```
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 אריתמטי 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)
```

דחיסת דקדוק Sequitur

קוראים את הטקסט תו אחרי תו ומוסיפים לS.

1. אם צמד תווים מופיע פעמיים בצד ימין הוא יוחלף בכלל חדש.
2. אם משתנה מופיע פעם אחת בצד ימין – נמחק אותו.

Tunstall (בהינתן מילון עם מילות קוד באורך קבוע, יש להתאים מחרוזות)

הכנס למילון את Σ.

כל עוד נותר מקום לS תווים נוספים:

1. עבור d מילת הקוד עם ההסתברות המקסי' במילון, הכנס את $\{d\sigma : \sigma \in \Sigma\}$ למילון.
2. אם $d \notin \Sigma$ – מחק את d מהמילון.

קוד פיבונאצ'י 1

כתוב את המספר בבסיס פיבונאצ'י, הוסף 1 משמאל, הפוך את המספר.

תכונות: קוד מלא, פרפיקסי, מיידי. כל F_k מילות הקוד בטווח $F_{k+1} \le j < F_{k+2}$ הן באורך $k+1$.

קוד פיבונאצ'י 2

$Fib2(1) = 1$, עבור $Fib2(n+1)$:

כתוב את n בבסיס פיבונאצ'י, הפוך את המספר, הוסף 10 משמאל.

דחיסת דקדוק Re-Pair

כל עוד ישנו צמד תווים שחוזר על עצמו בטקסט:

מצא את הזוג השכיח ביותר והחלף אותו במשתנה חדש.

דחיסה מילונית

דחיסה סטטית – אפשר ע"י גרידי (אם σ נמצא אז לחפש את $\sigma \cdot \sigma_2$ וכו' ואז להתחיל מחדש),אפשר למצוא אופטימלי ע"י רדוקציה לגרף. דחיסה **אדפטיבית** – המילון נבנה בזמן הקידוד

LZ77 – עם Sliding Window, Look Ahead Buffer ומכיל שלשות (off,len,symbol) כאשר לקדד תו בודד זה $(0,0,\sigma)$

אלגוריתם קידוד LZ77 :

1. p←1 // The next character to be coded
2. while there is text remaining to be coded

2.1. search for the longest match for S[p..] in S[p-W...p-1] if the match occurs at position m with length l

2.2 Output the triple (p-m, l,S[p+l])

2.3 p←p+l+1

הצבה עצמית – כאשר $length > offset$

LZSS – שיפור, משתמש בעץ בינארי מאוזן (AVL) ומחזיק ביט + (off,len) או ביט + אסקי (תו בודד)

LZW – שימוש בעץ trie, מחזיק Index | Phrase ומאותחל עם תווים בודדים, בנוסף מחזיק טבלה w | k

אלגוריתם לקידוד:

```
w ← first char of input
repeat
k ← next char
  if (EOF) output code(w)
  else
    if ((w · k) ∈ Dictionary) then w ← w · k
    else    output code(w)
            Dictionary ← w · k
            w ← k
```

אלגוריתם פענוח ל LZW: אתחל מילון עם תוים בודדים

```
OLD = first input code
output translation of OLD
while not end of input stream{
    NEW = next input code
    if NEW is not in the string table
        S = translation of OLD
        S = S · C
    else
        S = translation of NEW
    output S
    C = first character of S
    Translation(OLD) · C to the string table
    OLD = NEW }
```

מדיר בהתחלה: [0,1] ואז רשומים Interval: $p(\sigma) = \frac{1}{|\Sigma|}$ עבור כל $\sigma \in \Sigma$

בהמשך לכל תו x בטקסט נעדכן את הטווח לתו x לפי הנוסחאות: $N(x)$, $p(x) = \frac{N(x)}{N}$

לאחר מכן נחלק את הטווח low_bound(x), low+range*high_bound(x)) לפי low_bound(x)-low, high_bound(x)-low מכוב הקודם

$\sigma \in \Sigma$ לכל עבור $N(\sigma), p(\sigma) = \frac{N(\sigma)+1}{N+|\Sigma|}$ כאשר

פוסט Range Coding הלקח מאוחר יותר

אלגוריתם פענוח ל Range Coding:

```
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 אריתמטי 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)
```

חסר רישות:

ϵ

0

10

110

...

x אינו חסר רישות – הקוד לא יוחלט לגבי x אם קיים σ אם x הוא תחילית של $x\sigma$