Almog Jakov: https://github.com/AlmogJakov

דחיסת נתונים א' - דף נוסחאות

משפטים והגדרות

קוד UD: קוד שניתן לפענוח בצורה יחידה
קוד מידי: קוד שבו כל מילת קוד מפוענחת בסיום קריאתה
- קוד מידי אינו הכרחי עבור תכונת UD (למשל במחרוזת סופית נשים מחיצות בסיום קריאתה)

קוד שלם: קוד שהוא אינסופי למחצה כך שכשנפענח אותו משמאל לימין הוא תמיד יפוענח באופן UD
וגם לא נתקע כאשר נפענח אותו.
אם קוד הוא שלם אזי הוא UD.
עבור כל קוד שהוא שלם מתקיים $K(C) = 1 \leftrightarrow$. הוא לא מבוזבז ואין צמתים פנימיים בעלי בן אחד.
אורך מילת קוד ממוצעת: $E(C,P) = \sum_{i=1}^{n} p_i \cdot |c_i|$
קוד חסר רישות: לא קיימת מילת קוד שהיא רישה של מילת קוד אחרת
- אם קוד חסר רישות ניתן לבנות לו עץ
- אם קוד חסר רישות אז הוא UD
- קוד מידי $\leftrightarrow$ קוד חסר רישות

קוד בעל יתירות מינימלית: קוד C הוא קוד בעל יתירות מינימלית אם מתקיים $E(C,P) \le E(C',P)$ עבור כל קוד C' חסר רישות בעל n אותיות
אינפורמציה: $I(s_i) = -\log_2(p_i)$
- צריך לתכנן את הקוד כך שמילת קוד עבור s_i מכילה $I(s_i)$ תווים
- אם מתקיים $p_i = 1$ אז מתקיים $I(s_i) = 0$.
- אם הרצף $s_i s_j$ מופיע בהסתברות $p_i p_j$ אז $I(s_i s_j) = I(s_i) + I(s_j)$.

אנטרופיה: $H(P) = -\sum_{i=1}^{n} p_i \cdot \log_2(p_i)$
- אנטרופיה מהווה חסם תחתון (של סיביות) לאורך מילות הקוד
- את האנטרופיה מעגלים כלפי מעלה כי חייב לקחת מספר שלם של סיביות
- אנטרופיה זה בעצם ממוצע משוקלל של כמויות האינפורמציה
- לכל קוד UD מתקיים $H(P) \le E(C,P)$
- קוד חסר רישות הוא קוד UD (הפוך לא בהכרח)

קראפט: $K(C) = \sum_{i=1}^{n} 2^{-|c_i|}$
- אם $K(C) \le 1$ אזי קיים קוד חסר רישות C' כך ש: $E(C,P) = E(C',P)$ וגם $|C| = |C'|$
- אם $K(C) > 1$ אזי לא קיים קוד חסר רישות עם אותם אורכים
- אם קוד הוא UD אז $K(C) \le 1$
- קיים קוד מידי $\leftrightarrow K(C) \le 1$

:UD TEST

נגדיר: Dangling Suffix = המשלים של רישה של מילה מסוימת.
אלגוריתם:
- הכנס את כל מילות הקוד לקבוצה (נקרא למילים אלו "מילים מקוריות").
  חזור על התהליך כל עוד שלב 1 מתקיים:
  - בדוק אם קיימת מילה שהיא רישה של מילה אחרת
  - אם קיימת מילה כזו, הוסיף לקבוצה את הDangling Suffix
  - אם הDangling Suffix שהוספנו היא מילת קוד מקורית אזי הקוד לא UD

Static

ההנחה היא שההסתברויות בלתי תלויות ואחידות.
הקוד הוא סטנדרטי ולכן אין prelude.
כמו כן, השימוש הוא בתווי האסקי ולכן ההסתברות לכל תו הינה $\frac{1}{256}$
נשים לב כי האנטרופיה היא 8.

Semi-Static

כאן ההגבלה היא רק על התווים שמופיעים בטקסט במקום להשתמש בכל התווים באסקי. לכן, לאחר מעבר על הטקסט וספירת כמות התווים (ללא ספירת ההופעות של כל תו כי מניחים שההסתברות לכל תו היא $\frac{1}{n}$)
ובהנחה שמצאנו n תווים, אזי, הprelude יראה כך:
- 8 סיביות ראשונות: יהיו מספר התווים
- n*8 הסיביות הבאות: יהיו קודי האסקי של התווים
- שאר הסיביות: מילות הקוד באורך קבוע

$$E(C,P) = H(P) + \frac{8n+8}{|text\ size|}$$

Semi-Static (Self-probabilities)

בשיטה זו עוברים על הטקסט וסופרים את כמות התווים + כמות ההופעות של כל תו.
הסתברות של תו תהיה $\frac{number\ of\ occurrences\ in\ the\ text}{text\ size}$
בהנחה שמצאנו n תווים, אזי, הprelude יראה כך:
- 8 סיביות ראשונות: יהיו מספר התווים
- n*8 הסיביות הבאות: יהיו קודי האסקי של התווים
- שאר הסיביות: ביטים המייצגים את ההסתברויות של כל תו

נניח שמצאנו n תווים ונניח שצריך p סיביות בשביל לקודד הסתברות אזי

$$E(C,P) = H(P) + \frac{8n + pn + 8}{|text\ size|}$$

נעביר את התווים עם ההסתברויות השכיחות תחילה.

Unary Code

כל קידוד של תו במיקום i יהיה $1^{i-1}0$ (אינדקס נמוך להסתברות שכיחה)
אלגוריתם להפיכה לקוד שלם:
- הורד את 0 במילת הקוד האחרונה

Binary Code

קוד בינארי פשוט: מס' הסיביות המינימלי עבורו ניתן לייצג את כמות הא"ב
קוד בינארי מינימלי: נאפשר לחלק ממילות הקוד להיות באורך $\lceil \log_2 n \rceil$ ולחלק להיות באורך $\lceil \log_2 n \rceil - 1$ בצורה הבאה:

כמות מילים	אורך
$2n - 2^{\lceil \log_2 n \rceil}$	$\lceil \log_2 n \rceil$
$2^{\lceil \log_2 n \rceil} - n$	$\lfloor \log_2 n \rfloor$

כאשר מילות הקוד הקצרות בצד שמאל של העץ.

Elias

קוד C_γ:
אלגוריתם:
- סדר את האותיות לפי ההסתברויות שלהן (הכי נפוץ יהיה ראשון עם אינדקס 1).
- כל מילת קוד תראה כך: XY
  כאשר X הוא **מספר הסיביות** הנדרש על מנת לייצג את האינדקס באונארי וY הוא האינדקס עצמו בבינארי ללא ה1 המוביל (הכי שמאלי).

קוד C_δ:
זהה לחלוטין לקוד C_γ פרט לכך שX מקודד בעזרת C_γ במקום באונארי.

Golomb + Rice

קוד Rice הוא מקרה פרטי של Golomb כאשר הדלי הוא חזקה של 2. כלומר $b=2^k$.

Encode

```
Golom_encode(x, b):
  q = ⌊(x − 1)/b⌋
  r = x − q ∗ b
  p1 = Unary_encode(q + 1)
  p2 = Minimal_binary_encode(r, b)
  return p1 · p2
```

p2 היא מילת הקוד הr בא"ב בינארי **מינימלי** בגודל b. (כאשר המילים מסודרות בסדר לקסיקוגרפי ומהקצרה לארוכה).

Decode

```
Golom_decode(xy, b):
  q = Unary_decode(x) − 1
  r = Minimal_binary_decode(y)
  return r + q ∗ b
```

כאשר xy היא מילת הקוד אותה מפענחים.
x הוא הרישה (האונארית) של מילת הקוד
(ניתן לזיהוי כי מסתיים ב0)
וy זה שאר מילת הקוד

Fibonacci Code

אלגוריתם להפיכה לקוד מידי:
- חשב קוד פיבונאצ'י
- הוסף 1 לתחילת מילת קוד
- הפוך את המילים

תכונות:
- קוד פיבונאצ'י הוא קוד שלם (הוכחנו בעזרת קראפט)
- יש F_k מילות קוד באורך k+1 (כולל ה1 שמוסיפים בסוף) כאשר $F_1 = F_2 = 1$.
  מספר j המקיים $F_{k+1} \le j \le F_{k+2}$ צריך k+1 סיביות $F_{k+2} - F_{k+1} = F_k$

אלגוריתמים לפתירת הבעיה (של החסרון שאין מילה באורך 1):
1. אלגוריתם 1:
   - ניקח רק את המילים שמתחילות ב1 (לאחר שהפכנו)
   - נוריד את הסיבית האחרונה (שהיא 1 כי מסתיים ב11)
2. אלגוריתם 2 (שקולה לראשונה):
   - נוריד את הסיבית האחרונה (שהיא 1 כי מסתיים ב11)
   - נוסיף 10 להתחלה של כל מילת קוד
   - לבסוף נוסיף את המילה 1 כמילת הקוד הראשונה

תכונות של האלגוריתמים:
- מכילות מילת קוד אחת בלבד באורך 1.
- פרט למילה הראשונה, ישנן F_{k-2} מילות קוד באורך k.
- ניתן להוכיח שתמיד יותר טוב מקוד C_δ וקוד C_γ.

Shannon Code

$\log_2 \frac{1}{p} \le \left\lceil \log_2 \frac{1}{p} \right\rceil \le \log_2 \frac{1}{p} + 1$ אז: $H(P) \le L \le H(P) + 1$
Coding blocks of symbols:

$$H\left(p(s_i, s_j)\right) = 2H(p(s))$$

החולשה של שיטה זו שאנו יכולים לקבל עץ שאינו מלא.
encoding full binary tree:
בעץ מלא עם n עלים יש n-1 צמתים פנימיים
צריך 2n-1 ביטים כדי לקודד עץ בינארי עם n עלים
כל קוד חסר רישות ניתן לייצוג ע"י עץ
קידוד עץ:
- 8 סיביות ראשונות: מספר האלף בית
- n*8 סיביות הבאות: קידודי האסקי עבור כל אות (נניח שהאותיות המתקבלות מסודרות מהעלה השמאלי לעלה הכי ימני לפי הסדר)
- נעביר מחרוזת המייצגת את העץ כך שכל סיבית היא אינדיקטור אם צומת מסוים הוא פנימי (1) או עלה (0).

Shannon-Fano codes: אלגוריתם:
- סדר את ההסתברויות מהגדול לקטן.
- חלק את ההסתברויות לשני קבוצות כך שההפרש בניהם יהיה מינימלי ככל הניתן (אם החלוקה אינה ודאית ויש התלבטות עבור הסתברות מסוימת אז ניקח אותה לקבוצה השמאלית שבה האורך קצר יותר)
- לקבוצה השמאלית ניתן את הספרה 0 ולימנית את הספרה 1.

קוד שאינו תמיד אפקטיבי.

Huffman Code (Minimum Redundancy Coding)

אלגוריתם:

```
Huffman(Σ):
  n = |Σ|
  Q1 = Σ // min heap
  For i=1 to n-1:
    Do allocate-node(Z) // new node
    z.left = x = Extract_min(Q1)
    z.right = y = Extract_min(Q1)
    z.weight = x..weight + y.weight
    insert(Q1,z)
  return Extract_min(Q1)
```

משפט: בהינתן משקלים $w_1, \dots, w_n$ הופמן מקצה אורכי קוד $l_1, \dots, l_n$ כך ש $\sum_{i=1}^{n} w_i l_i$ הוא מינימלי.
למה 1: קוד אופטימלי עבור קובץ תמיד מיוצג ע"י עץ בינארי מלא כך שלכל צומת פנימית יש 2 ילדים.
למה 2: בעץ אופטימלי ה2 ילדים עם המשקלים הכי נמוכים נמצאים ברמה הנמוכה ביותר.
למה 3: בעץ אופטימלי ה2 ילדים עם המשקלים הכי נמוכים יכולים להיות אחים.
בהחלפת 0/1 עבור צמתים פנימיים הופמן יכול לתת 2^n קודים שונים.

Huffman (Canonical)

אלגוריתם לעץ הופמן קנוני משוך שמאלה (בהינתן קוד הופמן):
בהינתן אותיות $\{i\}$ ואורך אופטימלי $\{l_i\}$

```
maxlength=max{l_i}
// find the number of codewords of each length
For l = 1 to maxlength
  num[l]=0
For i = 1 to n
  num[l_i]++
// store the first codeword
Firstcode[maxlength]=0
For l=maxlength-1 downto 1
  firstcode[l] = (firstcode[l+1]+num[l+1])/2
For l=1 to maxlength
  nextcode[l] = firstcode[l]
For i=1 to n
  codeword[i]=nextcode[l_i]
  symbol[l_i,nextcode[l_i]-firstcode[l_i]] = i
  nextcode[l_i]++
```

ייצוג עץ הופמן קנוני:
1. העבר את האותיות לפי הסדר (מהארוך לקצר ובסדר לקסיקוגרפי)
2. העבר טבלה שבה יש את רק את האותיות שבתחילת כל בלוק (כלומר, המילה הראשונה בכל אורך [מהארוכה אל הקצרה]) כך שהטבלה מכילה את האותיות הנ"ל ואת הקידוד שלה.

אלגוריתם פענוח הודעה בינארית (שימוש במטריצת symbol שבנינו):

```
maxlength=max{l_i}
v=nextInputBit()
l = 1
while v < firstcode[l] do
  v = 2v+nextInputBit()
  l++
// the integer v is a valid codeword of l bits
return symbol[l,v-firstcode[l]]
// the index of the decoded symbol
```

הגדרה שקולה לעצי הופמן קנוני:
אלגוריתם לבניית עץ הופמן קנוני משוך ימינה.
- תחילה, נפעיל את אלגוריתם הופמן הרגיל
- לאחר מכן, נמיין את מילות הקוד שקיבלנו מהאלגוריתם הרגיל ע"פ האורכים של מילות הקוד
- לכל מילת קוד ניתן הסתברות 2^{-l_i} כאשר l_i היא אורך מילת הקוד
- לכל מילה נגדיר מספר $k = \sum_{j=1}^{i-1} 2^{-l_i}$. כלומר, עבור מילת קוד i אנו סוכמים את ההסתברויות של מילות הקוד שהאינדקס שלהן קטן ממש מהאינדקס i
- מילת הקוד החדשה תהיה l_i הספרות שאחרי הנקודה העשרונית של המספר k הנ"ל

Huffman (D-ary)

לעץ הופמן די-ארי עם n צמתים פנימיים יש $(D-1) \cdot n + 1$ עלים
אלגוריתם:
- חשב מה מספר העלים n_0 שצריך להוסיף כך שמתקיים $n + n_0 = (D-1) \cdot n' + 1$
- הפעל הופמן עבור D עלים בכל שלב
- תן ערך $0, \dots, D-1$ לכל קשת (הספירה מתחילה מ0)
- התאם עלה להסתברות שלו (וזרוק את העלים עם ה0)

Almog Jakov: https://github.com/AlmogJakov

דחיסת נתונים א' - דף נוסחאות

Huffman (Adaptive)

Sibling property:
- משקל צומת שווה לסכום המשקל של שני ילדיו.
- הצמתים ניתנים לסידור ע"פ משקלם בסדר ≥ כך שמתקיים שהקודקודים 2j-1 ו-2j אחים.

אלגוריתם:
- הכנס לעץ עם תדירות 0.
- קודד ורק אז עדכן את העץ.

אלגוריתם העדכון: (כאשר x הוא העלה שהגענו אליו באחד משלבי הקידוד וכעת עלינו לעדכן)

```
q = leaf(x)
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 weight by 1;
  q = parent of q;
while q!= root
  interchange q with the highest numbered node of the same weight;
  increment q weight by 1;
  q = parent of q;
increment q weight by 1;
```

Huffman (Skeleton)

אלגוריתם:
- צור קוד הופמן קנוני משוך ימינה
- הסר את כל תתי עצים השלמים שגובהם לכל הפחות 1.
- תן לצומת פנימי ערך 0. ותן לעלים אורך מילות הקוד מהשורש (הכללי) עד לעלים (של תת העץ שמחקנו).

הגדרות:

n_i = מספר מילות הקוד באורך i

$m = min\{i|n_i > 0\}$. כלומר, זאת המילה עם האינדקס המינימלי.

$base(i)$ = ערך מספרי של מילת הקוד הראשונה באורך i
- $base(m) = 0$
- $base(i) = 2(base(i-1) + n_{i-1})$

$B_s(k)$ = התצוגה של המספר k בצ ביטים בדיוק (0 משמאל אם צריך).
- מילת הקוד באינדקס j באורך i (עבור $j = 0,1,\dots n_i - 1$) היא $B_i(base(i) + j)$

$seq(i)$ = מספר האינדקס של המילה הראשונה באורך i
- $seq(m) = 0$
- $seq(i) = seq(i-1) + n_{i-1}$

$I(w)$ = הערך המספרי של הייצוג הבינארי w
- אם w באורך l אז מתקיים $w = B_l(I(w))$

מכאן: $I(w) - base(l)$ = האינדקס הפנימי של מילת הקוד w **בתוך הבלוק** של מילות הקוד באורך l

ולכן: $seq(l) + I(w) - base(l)$ = האינדקס הכללי של מילת הקוד w (כללי = ביחס לכלל מילות הקוד). אינדקס כללי:

$I(w) - diff(l)$ כאשר $diff(l) = base(l) - seq(l)$

אלגוריתם פענוח:

```
tree_pointer=root
i=1
start=1
while i < length_of_string
  if string[i]=0
    tree_pointer=left(tree_pointer)
  else
    tree_pointer=right(tree_pointer)
  if value(tree_pointer)>0
    codeword=string[start…(start+value(tree_pointer)-1)]
    output=table[I(codeword)-diff[value(tree_pointer)]]
    tree_pointer=root
    start=start+value(tree_pointer)
    i=start
  else i++
```

Arithmetic Code (Static)

אלגוריתם קידוד:

```
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)
```

- גודל האינטרבל הוא כפל הסתברויות כל אות בהודעה שקידדנו.

אלגוריתם פענוח (עבור מספר n):

```
Find symbol whose range contains n
Output the symbol
Range = high(symbol) – low(symbol)
Encoded = (encoded-low(symbol))/range
```

Arithmetic Code (Adaptive)

$N(a)$ = מספר התדירויות שהאות a הופיעה.

$P(a)$ = ההסתברות לקבל את a.

כאשר: $P(a) = \frac{N(a)+1}{N(a)+N(b)+N(c)+3}$

וכן לכל אות באלף בית.

LZ77

(offset,length,symbol)

אלגוריתם קידוד:

```
p=1 // next char to be coded
while (there is text to be coded):
  search for the longest match for S[p…] in s[p-w…p-1]
  // suppose match at pos m with len = 1
  output the triple (p-m,l,s,s[p+1])
  p = p+l+1
```

LZSS

(offset,length) [1] או בתווים בודדים [0].

LZS

(offset1,length1), (offset2,length2)

LZ78

אלגוריתם:
- בנה מילון D שיכיל בהתחלה רק את המילה הריקה (שהאינדקס שלה הוא 0)
- הכנס תתי מחרוזת (של המחרוזת שנרצה לקודד) ותן לה את האינדקס הבא בתור שפנוי.
- נסה "לתפוס" את המילה הכי ארוכה שכבר במילון ונקודד את האינדקס שלה + הקידוד של התו הבא. במקביל,
- הוסף למילון את אותה הרישא בשרשור התו הבא שמקודדים

שיטה לדעת כמה ביטים צריך לקידוד מס': נשאל – "כמה ביטים צריך בשביל לקודד x מספרים" כאשר x הוא האינדקס באותה שורה

LZW

אלגוריתם קידוד: (טבלה התחלתית גדולה ממש מגודל הא"ב)

```
Dictionary=single characters
w=first char of input
repeat {
  k=next char
  if(EOF)
    output code(w)
  else if (w·k)∈Dictionary
    w = w·k
  else
    output code(w)
    // output in num of bits
       corresponding to table size
    Dictionary.add(w·k)
    // Only enlarge the table when
       entering information and when it is full
    w = k
}
```

אלגוריתם פענוח:

חשוב: המפענח תמיד מגדיל את המילון ברגע שהמילון מתמלא

```
Initialize table with single character strings
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
  Enter Translation(OLD)·C to the string table
  // Increase the table as soon as the
     dictionary fills up
  OLD = NEW
}
```

Run length Code

זוג סדור (num,symbol) ע"פ רצף של תווים.

Binary Run length Code:

escape codeword = ספרה מקסימלית.

The Burrows Wheeler Transform

אלגוריתם:
- בנה מטריצה מסדר nxn (כאשר הטקסט באורך n). כך שהשורות יהיו כל הטרנספורמציות הציקליות של הטקסט. כלומר, השורה הראשונה תתחיל באות הראשונה של הטקסט. השורה השניה תתחיל מהאות השניה של הטקסט וכו' (ממשיכים לשכתב את הטקסט בצורה מעגלית בסוף השורה)
- מיין את השורות בסדר לקסיקוגרפי (לפי האלף בית (abcdefghijklmnopqrstuvwxyz)
- החזר את העמודה האחרונה במטריצה (מלמעלה למטה) + אינדקס של השורה המקורית של הטקסט (ספירה מ-0).

פענוח:

אלגוריתם פענוח:

```
BWT(L, index):
  F = sort(L)
  M = Index
  For (i=0; i < n; i++)
    T[i] = F[M]
    M = S[M]
```

Move To Front

אלגוריתם:

קודד את האינדקס של כל תו ואז העביר אותו לראש הרשימה.

בחזרה לאלגוריתם BWT:

אלגוריתם:
- נפעיל על ההודעה את אלגוריתם BWT
- לאחר מכן נפעיל על העמודה האחרונה (שאנו מעבירים) את אלגוריתם Move To Front
- על הפלט של Move To Front נפעיל דחיסה סטטיסטית הופמן/אלייאס/RunLength/אריתמטי
- לבסוף נוכל להעביר את האינדקס שהאלגוריתם BWT פלט + הקידוד (הסופי) של העמודה

PPM (+EP)

$H(X|Y) \le H(X)$

הוכחנו בהרצאה כי אנטרופיה מותנת קטנה-שווה לאנטרופיה רגילה.

כלומר, $H(X|Y) \le H(X)$

אלגוריתם:

נוסיף לטבלה תו $ שערכו יהיה מספר האלף בית באותו הקשר ונקודד אותו על מנת לרדת רמה.

אם הגענו לרמה 1- זה אומר שצריכים לקרוא תו חדש ולכן אנו מצפים לקידוד שלו.

התו $ יהיה חלק מהאלף בית וההסתברות לקודד תו חדש מהאלף בית תהיה אחידה.

עבור קידוד של תו קיים ההסתברות לקידוד תהיה אחידה על פני יחס ההופעות של תו מתוך ההופעות של שאר התווים באותה רמה.

בכל שלב נסכם בסוגריים מה ההסתברות לקידוד התו. מספר הסיביות הדרוש יהיה $\lceil \log_2 \frac{1}{p} \rceil$.

חשוב לזכור: תמיד נעדכן את כל הרמות לאחר קריאת תו.

Exclusion Principle (EP):

אלגוריתם לחישוב הסתברויות ברמה מסוימת $0 \le k < kmax$:

```
estimate_prob(k) {
  sum=count[$];
  for (i=0; i<=|Σ|; i++)
    if k==kmax
      count[i]=count[a_i | previous k symbols]
    else if (k>=0)
      count[i]=(count[a_i | previous k+1 symbols]==0)*count[a_i | previous k symbols]
    else
      count[i]=(count[a_i]==0)
    Sum+=count[i]
  for (i=1; i<=|Σ|; i++)
    p(encode a_i) = count[i]/sum
  p(encode $) = count[$]/sum
}
```

Grammer Compression

דטרמיניסטי (אין הסתעפויות בגזירה) ואסור שיהיה מעגל בגזירות.

Sequiter:
- תווים עוקבים לא יופיעו יותר מפעם אחת
- חוק צריך להופיע לפחות פעמיים

אלגוריתם:

כל עוד לא הגענו לEOF:
- אם ישנם 2 תווים צמודים שחוזרים על עצמם – צור כלל חדש
- אם ישנו חוק שמשתמשים בו פעם אחת – הסר אותו והכנס את התוכן למופע היחיד שלו

Re-Pair

אלגוריתם:
- מצא זוג תווים הכי נפוץ. לדוגמה, ab.
- צור כלל. לדוגמה, $A \rightarrow ab$.
- החלף את כל מופעי הזוג בכלל שיצרת. לפי הדוגמה הנ"ל החלף את מופעי הזוג בכלל A.
- חזור ל1 כל עוד מצאת זוג שמופיע **יותר** מפעם אחת (בחר את הכי נפוץ).