

P, NP, NPC, and etc.

בעיה שהיא **Hard** היא בעיה שלא קיימת עבודה פתרון בזמן פולינומי.
משפט ההיררכיה של הזמן: קיימות בעיות hard כרצוננו. כלומר: בהינתן 2 פונקציות זמן $f_1 < f_2$ אזי קיימת לפחות קבוצה אחת שניתנת לחישוב בזמן f_2 אבל לא בזמן f_1 .

P: בעיה היא ב-P אם קיים אלגוריתם דטר' בזמן פולינומי להכרעתה.
NP: בעיה היא ב-NP אם קיים אלגוריתם לא-דטר' בזמן פולינומי להכרעתה.
co-NP: $L \in co-NP \Leftrightarrow \bar{L} \in NP$

Non determinism: קיימות 2 גישות שקולות להראות שבעיה היא ב-NP – כתיבת אלגוריתם לא-דטר':
1. **ארכיטקטורת ירוק-אדום**: מאפשרים פקודות goto label_i. הארכיטקטורה בוחרת את ה-labels ה"טובים" (שימוש ב-oracle) כך שהיא תוכל בסוף להגיע לאור ירוק. אם אין היא יכולה בסוף להגיע לאור ירוק היא תבחר label רנדומי ובסוף תגיע לאור האדום.
נכתוב אלגוריתם **פולינומי** שמשתמש בארכיטקטורה זו.
הערה: צריך להקפיד שבעת כתיבת האלגוריתם מספר ה-labels יהיו קבועים ולא תלויים בקלט.
2. **ניחוש**: נחש פתרון לבעיה (באופן לא-דטר') ובדוק "ע" אלגוריתם דטר' בזמן פולינומי שהפתרון נכון. אם כן – accept, אחרת – reject.
הערה: צריך להקפיד שבדיקת הניחוש תיעשה בזמן **פולינומי בקלט של הבעיה** (כלומר, צריך שאורך הניחוש יהיה חסום בפולינום באורך הקלט).

- אם קיים אלגוריתם לא דטר' לבעיה A שזמנו $f(n)$ אזי קיים אלגוריתם דטר' לבעיה A שזמנו $O(d^{f(n)})$ כאשר d זה המספר המקסימלי של goto labels באלגוריתם הלא-דטר'.

הוכחה: נבנה עץ של כל המסלולים האפשריים של התוכנית (למשל, תוך שימוש ב-BFS). עומק העץ הוא $f(n)$ כי בזמן הזה או שהיה קודקוד אחד לפחות שהגיע לירוק או שכל שאר הקודקודים הגיעו לאדום. מספר הפיצולים בכל רמה בעץ הוא כמספר ה-labels, כלומר: d . ולכן סה"כ קודקודים יהיו $d^{f(n)}$, כאשר כל קודקוד כזה הוא דטרמיניסטי. כעת נעבור סדרתית על הקודקודים עד שנמצא ירוק (במקרה הגרוע ביותר נעבור על כל הקודקודים).

רדוקציות

1. **טיורינג**: $A \leq_T^P B$ אם קיים אלגוריתם בזמן פולינומי להכרעת A תוך שימוש ב-B כ-black-box (oracle).
2. **many-one**: $A \leq_M^P B$ אם קיימת פונקציה פולינומית f כך ש- $x \in A \Leftrightarrow f(x) \in B$.
הערה: הפונקציה לא חייבת להיות חח"ע ועל.

NPC: $A \in NPC$ אם $A \in NP$ ו- $\forall B \in NP, B \leq_M^P A$.
הערה: אם לא ידוע אם $A \in NP$ אזי A היא NP-Hard.

- רדוקציית many-one היא טרנזיטיבית, כלומר: $A \leq_M^P B \text{ and } B \leq_M^P C \Rightarrow A \leq_M^P C$.
הוכחה: $[x \in A \Leftrightarrow f_1(x) \in B, x \in B \Leftrightarrow f_2(x) \in C] \Rightarrow x \in A \Leftrightarrow f_1(f_2(x)) \in C, f_1 \circ f_2 \in poly$

טענות

1. $\overline{TAUT} = SAT \quad TAUT \in co-NP \Leftrightarrow \overline{TAUT} \in NP$
2. $P = NP \Rightarrow NP = co-NP$, שכן P סגור תחת המשלים.
3. $P \in NP \cap co-NP \Rightarrow NP \cap co-NP \neq \emptyset$
4. P סגורה תחת משלים ותחת הרכבה. לא ידוע אם NP סגורה תחת משלים ותחת הרכבה (לו NP הייתה סגורה תחת הרכבה אזי היינו מפעילים את פונקציית ההיפוך על הבעיה וכך מקבלים את המשלים של הבעיה $(NP = co-NP)$).
5. אם $A \leq_T^P B$ or $A \leq_M^P B$ וידוע ש- $B \in P$, אזי $A \in P$. (29)
6. $A \leq_M^P B \Rightarrow A \leq_T^P B$ – (28) רדוקציית many-one היא מקרה פרטי של רדוקציית טיורינג.
7. $A \leq_T^P B \Rightarrow A \leq_T^P \bar{B}$
8. אם $A \leq_T^P B$ אזי $NP = co-NP$.
9. $SAT, Clique, VC, HC \in NPC$ (31, 34) $VC \leq_M^P Clique \leq_M^P VC$, $HC \leq_M^P VC$.
10. $DNF \in P$

11. $A \leq_M^P B, A \in NPC \Rightarrow B \in NP - Hard$ (many-one reductions) (טרנזיטיביות)

12. אם $A \leq_M^P B$ ו- $B \in NP$ אזי $A \in NP$ (זה לא נכון עבור רדוקציית טיורינג)

13. מתקיים תמיד ש: $A \leq_T^P \bar{A}, \bar{A} \leq_T^P A$

14. $A \leq_M^P B \Leftrightarrow \bar{A} \leq_M^P \bar{B}$

15. אם $A \in NPC$ וגם $A \in co - NP$ אזי $NP = co - NP$

16. לכל $A \in P$ ו- Σ^* מתקיים $B \neq \emptyset, A \leq_M^P B$

17. $NP \neq co - NP \Rightarrow P \neq NP$

18. $P \subseteq co - NP$

19. $A \leq_M^P \bar{A} \Leftrightarrow \bar{A} \leq_M^P A$

20. $A \in NPC \Rightarrow \bar{A} \in co - NPC = (co - NP) - C$

21. כל שפה ב-P היא שלמה למעט השפה הריקה והשפה האוניברסלית.

22. אם $A \in co - NP$ ו- $A \in NP - Hard$ אזי $A \in NP$

23. אם $A \in NPC, A \leq_M^P B$ ו- $B \leq_M^P A$ אזי $B \in NPC$

24. אם $A \in NP - Hard, A \leq_M^P B$ ו- $B \leq_M^P A$ אזי $B \in NP - Hard$

25. $A \in P$ ו- $B \in NP$ $\nRightarrow B \leq_M^P A$

26. $A, B \in NP$ $\nRightarrow B \leq_M^P A$

27. נוח לעשות רדוקציה מ-VC לבעיה אחרת שבה מחפשים נציגים מתוך קבוצה.

תרגיל: נתון $A \in P; B, C \in NP; D \in NPC; E \in Co - NP$ עבור הבעיות הבאות הכרע האם התשובה (1) שגויה, (2) נכונה רק אם $P = NP$ או (3) נכונה תמיד.

א. $B \leq_M^P \bar{B}$ נכון אם $P = NP$.

ב. $B \leq_T^P E$ נכון אם $P = NP$.

ג. $A \leq_M^P D$ נכון.

ד. $D \leq_M^P \bar{E}$ נכון אם $P = NP$.

ה. $E \leq_T^P D$ נכון.

α -Approximations: אלגוריתם A בזמן פולינומלי כך ש- $C(A) \leq \alpha \cdot C(OPT(A))$

2-Approximation for VC

```

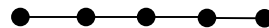
1   $V_c \leftarrow \emptyset$ 
2  while  $E \neq \emptyset$ 
3      Pick  $(a, b) \in E$ 
4       $V_c \leftarrow V_c \cup \{a, b\}$ 
5      Delete from E any edge incident on a or b
    
```

בהנחה שהקודקודים הם $\{1, \dots, n\}$ אזי ניצור טבלה (רשימת שכנים) כך שהורדת כל קשת תיקח $O(1)$ והורדת כל הקשתות תיקח $O(|E|)$.

אם הקודקודים אינם $\{1, \dots, n\}$ אזי ניתן לשנות את השמות ב- $O(n \log n)$ (ע"י מיון הקודקודים ואז שינוי השמות ל- $\{1, \dots, n\}$).

לכן, זמן האלגוריתם הוא $O(n \log n + |E|)$.

כיוון שאנחנו בוחרים כל הזמן להוסיף 2 קודקודים ל-VC אזי במקרה הגרוע ביותר אנו מגיעים ל-VC שגדול פי 2 מ- OPT . לדוגמה:



Pattern Matching

:Pattern Matching with Don't Cares

- לא ניתן להשתמש באוטומט או בטבלת עדים לפתור את הבעיה, שכן ה- $\emptyset$ "הורסים" את הטרגיטיביות עליה בנויות שיטות אלו.
 - השיטות שמסתמכות על טרגיטיביות: טבלת עדים, אוטומט, עצי מלים.
- נפתור ע"י כפל פולינומים, כאשר נגדיר את הכפל בין שני תווים כך: $a \cdot b = \begin{cases} 0 & , a = b \\ 1 & , a \neq b \end{cases}$ כאשר $\emptyset \cdot a = 0$ תמיד. הפתרון הזה נותן את מספר השגיאות לכל מיקום.

עבור $\Sigma = \{0, 1\}$

על-מנת לבצע את כפל הפולינומים ע"י FFT צריך שהכפל יהיה סגור לכן נבצע: $\bar{T} \cdot P^R + T \cdot \bar{P}^R$ לפי הכפל שהגדרנו לעיל ונפתור את הבעיה בזמן $O(n \log m)$.

עבור א"ב גדול יותר:

שלב ראשון:

נשים לב שלא משנה כמה הא"ב גדול, הוא תמיד חסום ע"י $\sigma = \min(m+1, |\Sigma|)$ (וזאת כיוון שאנו מעוניינים במismatch: ניתן להחליף כל אות בטקסט שאינה מופיעה בתבנית באות אחרת יחידה השייכת לא"ב).

הערה: אם אנחנו מחפשים less-than-matching אזי ניתן לחסום את הא"ב ב- $\sigma = \min(2m+1, |\Sigma|)$ (כל אות בטקסט שאינה בתבנית ניתן להחליף באיזושהי אות יחידה שהיא בטווח של 2 אותיות בתבנית).

את ההחלפה של התווים בטקסט ניתן לעשות ב- $O(n \log m)$ וזאת בהנחה ש- $\pi = \{x \in \Sigma \mid x \in P\}$ ממין (נעשה את ההחלפה ע"י חיפוש בינארי על π).

שלב שני:

נקודת כל אות בא"ב בצורה בינארית (כל תו $\leftarrow \log \sigma$ ביטים), כאשר $\emptyset$ יקודד ע"י $\log \sigma$. כלומר, ניצור טקסט ותבנית חדשים, ולכן נקט:

$$\left. \begin{array}{l} T \rightarrow T', |T| = n \rightarrow |T'| = n \log \sigma \\ P \rightarrow P', |P| = m \rightarrow |P'| = m \log \sigma \end{array} \right\} \begin{array}{l} O(N \log M) = O(n \log \sigma \log(m \log \sigma)) = O(n \log \sigma (\log m + \log \log \sigma)) \\ = O(n \log \sigma \log m + n \log \sigma \log \log \sigma) = O(n \log m \log \sigma) \end{array}$$

בעיה: האלגוריתם קטע לא סופר את השגיאות נכון (הוא סופר מספר ביטים שגוי ולא מספר תווים). (עמ"נ למצוא **התאמה** נצטרך להסתכל בוקטור המכפלה הסופית בתוצאה רק בגבול האינטגרלי של הקידוד, כלומר בערך המתקבל בוקטור כל $\log \sigma$ מקומות, כלומר במקומות $1, 1 + \log \sigma, 1 + 2 \log \sigma, \dots$ (אם במקומות האלו יש $0 \leftarrow$ יש התאמה בטקסט)).

פתרון:

לכל תו בתבנית נבצע מכפלה בשביל לבדוק את מספר השגיאות שנגרם לכל מקום בו נציב את התבנית על הטקסט. החישוב יהיה כדלהלן: נגדיר: $\chi_\sigma(a) = \begin{cases} 1 & , \sigma = a \\ 0 & , \sigma \neq a \end{cases}$

לכל $\sigma \in \pi$ נחשב $V_\sigma = \chi_{\bar{\sigma}}(T) \cdot \chi_\sigma(P^R)$ בדיקת מספר ההתנגשויות של σ בתבנית למשהו שהוא לא σ . ובסוף נסכום: $V \leftarrow \sum_{\sigma \in \pi} V_\sigma$ כאשר: $V_\sigma[i] =$ מס' השגיאות כאשר במקום i מול התו σ . $V[i] =$ מס' השגיאות כאשר התבנית מתחילה במקום i בטקסט.

סה"כ:

עבור א"ב חסום נקבל: $O(n|\pi| \log m) = O(n \log m)$

עבור א"ב לא חסום נקבל: $O(n|\pi| \log m) = O(nm \log m)$ שזה גרוע מהנאיבי $O(nm)$.

פתרון – Kosaraju – Divide & Conquer

ב-Kosaraju עובדים עם טקסט בגודל $2m$. איך?
 מחלקים את הטקסט בגודל n למקטעים בגודל $2m$ (סה"כ $n/2m$ מקטעים כאלו). כיוון שגודל התבנית הוא m עשויה להיות התאמה שתהיה בין 2 מקטעים. לכן נחלק לעוד $n/2m$ בגודל $2m$ כאשר כל מקטע כזה מתחיל מאמצע מקטע אחד עד אמצע המקטע שאחריו. סה"כ יש n/m מקטעים.

כעת ניתן לבדוק התאמות בזמן $O(mf(m))$ עבור כל מקטע ולכן סה"כ עבור טקסט בגודל n זה יעלה לנו:
 $O(n/m \cdot m \cdot f(m)) = O(n \cdot f(m))$

צריך לראות שניתן לעבור ממיקום בטקסט המקורי למיקום במקטע:
 עבור מקום i בטקסט המקורי צריך למצוא באיזה מקטע הוא נמצא ואיפה בתוך המקטע הוא נמצא. 2 מקרים:
 1. אם $i \bmod 2m \leq m$:

א. המקטע הרצוי הוא מקטע מספר $\lfloor i/2m \rfloor$ (המקטע הראשון הוא מקטע מספר 0).
 ב. המיקום במקטע הוא: $i \bmod 2m$.

2. אם $i \bmod 2m > m$:
 א. המקטע הרצוי הוא מקטע מספר $\lfloor i - m/2m \rfloor$.
 ב. המיקום במקטע הוא: $(i - m) \bmod 2m$.

האלגוריתם: (הטקסט הוא בגודל $2m$ ולכן סה"כ גודל הטקסט והתבנית הוא $O(m)$).
שלב ראשון:

אות שכיחה היא אות שמופיעה יותר מ- $\sqrt{m}$ פעמים (בטקסט ובתבנית). יש לכל היותר $3\sqrt{m}$ אותיות שכיחות. עבורם נפעיל את אלגוריתם ה-mismatch בנפרד והעלות תהיה $O(m\sqrt{m} \log \sqrt{m}) = O(m\sqrt{m} \log m)$

שלב שני:

את שאר האותיות בטקסט ובתבנית נכתוב כשלושה: <אינדקס, {T/P}>, תוך (עמ"נ לאפשר מעבר מהיר בין שלושה לאות המקורית ניצור מצביע דו-כיווני ביניהן).
 נסדר את כל השלושות לקסיקוגרפית לפי התו.
 נחלק את השלושות לקבוצות בגודל $O(\sqrt{m})$, כך שלא נפריד בין אותיות זהות לקבוצות שונות, כלומר אם אחרי $\sqrt{m}$ תווים האות עוד לא הסתיימה, נמשיך את הקבוצה עד שהאות תסתיים (כיוון שהאות אינה שכיחה אזי היא לא מופיעה יותר מ- $\sqrt{m}$ ולכן גודל הקבוצה יישאר $O(\sqrt{m})$).
 מספר הקבוצות הקיימות הוא $O(\sqrt{m})$.

לכל קבוצה נבחר נציג ובבנה טקסט ותבנית חדשים עם הנציגים בלבד. גודל הא"ב יהיה כמספר הנציגים - $O(\sqrt{m})$.
 למצוא mismatch על הטקסט והתבנית החדשים ייקח $O(m\sqrt{m} \log \sqrt{m}) = O(m\sqrt{m} \log m)$
 כל שגיאה שספרנו בטקסט ובתבנית החדשים הייתה קיימת גם בטקסט ובתבנית הישנים, אולם יש שגיאות שלא ספרנו (שגיאות בתוך הקבוצה עצמה: למשל, אם החלפנו את a ואת b באותו נציג אזי לא ספרנו את השגיאות של a מול b).

פתרון: לכל מיקום i בטקסט נעביר את הקבוצה שלו (השלושות) ונשווה את איבריה עם האיבר בטקסט. נשים לב שצריך להשוות רק איבר שהוא במקורו בתבנית (לא צריך לספור שגיאות של טקסט מול טקסט) ושהוא שונה מהתו בטקסט. אם האות שונה והיא נמצאת במקום j בתבנית, אזי נקדם את השגיאה במקום $j - i$.
 את זה נעשה לכל תו בטקסט ולכן הזמן: $O(m\sqrt{m})$.

סה"כ: $O(m \log m) + O(m\sqrt{m} \log m) + O(m\sqrt{m}) = O(m\sqrt{m} \log m)$

ועבור טקסט בגודל n : $O(n\sqrt{m} \log m)$

- ניתן להגיע גם ל- $O(n\sqrt{m \log m})$ אם נגדיר איבר שכיח להיות איבר שמופיע יותר מ- m/k פעמים ונגדיר גודל קבוצה להיות $O(m/k)$, כאשר $k = \sqrt{m/\log m}$

FFT

כפל רגיל עולה $O(n^2)$. כפל מטריצות $n \times n$ עולה גם $O(n^2)$.
 נשתמש ב-DFT (divide & conquer) לפתור ב- $O(n \log n)$.
 DFT עובד בשדה המרוכבים: $(r, \theta) = a + bi$, $r = \sqrt{a^2 + b^2}$, $a = r \cos \theta$, $b = r \sin \theta$, $\theta = \arctan(b/a)$
 $(r_1, \theta_1)(r_2, \theta_2) = (r_1 r_2, \theta_1 + \theta_2)$

שורשי יחידה: $x^n = 1$

$$(r, \theta)^n = (r^n, n\theta) = 1 + 0i \Rightarrow r^n = 1, n\theta = 0 \Rightarrow r = 1, \theta = \frac{2\pi j}{n}, j = 0, 1, \dots, n-1$$

- אם w הוא שורש יחידה n -י אזי לכל j w^j הוא גם כן שורש יחידה n -י.
הוכחה: $(w^j)^n = (w^n)^j = 1^j = 1$

הגדרה: w ייקרא **פרימיטיבי** אם:

- w הוא שורש יחידה n -י.
- w יוצר את כל שורשי היחידה, כלומר: $1, w, \dots, w^{n-1}$ כולם שונים.

תכונות:

- סכום כל שורשי היחידה ה- n הוא 0 (עבור $n \geq 2$).
- אם w הוא שורש יחידה n -י פרימיטיבי ו- $c \nmid n$ אזי $\sum_{j=0}^{n-1} (w^c)^j = 0$.

הערה: $2 \leq c$ (ניקח $c=1$).
הוכחה: $\sum_{j=0}^{n-1} (w^c)^j = \frac{(w^c)^n - 1}{w^c - 1} = \frac{1 - 1}{w^c - 1} = 0$

אם n הוא זוגי ו- w הוא שורש יחידה n -י פרימיטיבי אזי:
 1. w^2 הוא שורש יחידה $n/2$ פרימיטיבי.

2. $w^{n/2} = -1$
הערה: $2 \leq j \leq n/2 - 1 \Rightarrow w^{n/2+j} = -w^j$

הוכחה:

1. $1, w, \dots, w^{n-1}$ כולם שונים $\Leftarrow 1, w^2, \dots, w^{n-2}$ כולם שונים. כמו כן: $(w^2)^{n/2} = w^n = 1$
 2. $(w^{n/2})^2 = 1 \Leftarrow w^{n/2} = \pm 1 \Leftarrow w^{n/2} = -1$

DFT: נייצג את הפולינומים שנרצה להכפיל בשדה המרוכבים ונחשב: $F_n P$, כאשר:

כאשר w הוא שורש יחידה n -י פרימיטיבי.

$$F_n = \begin{pmatrix} 1 & 1 & \dots & 1 & \dots & 1 \\ 1 & w^1 & \dots & w^j & \dots & w^{n-1} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & w^i & \dots & w^{ij} & \dots & w^{i(n-1)} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & w^{n-1} & \dots & \dots & \dots & w^{(n-1)(n-1)} \end{pmatrix}$$

$$\begin{aligned} F_n P &= w^0 p_0 + w^1 p_1 + w^2 p_2 + \dots + w^{ij} p_j + \dots + w^{i(n-1)} p_{n-1}, \quad \forall i, 0 \leq i \leq n-1 \\ &= p_0 + p_1(w^i) + p_2(w^i)^2 + \dots + p_j(w^i)^j + \dots + p_{n-1}(w^i)^{n-1}, \quad \forall i, 0 \leq i \leq n-1 \\ &= p_0 + p_1 X + p_2 X^2 + \dots + p_{n-1} X^{n-1}, \quad X = w^0, w^1, \dots, w^{n-1} \text{ עבור } \end{aligned}$$

נשתמש ב-divide & conquer: נקטין את הפולינום ל-2 פולינומים קטנים בעלי דרגה $n/2$ (אם דרגת הפולינום היא לא חזקה של 2 אזי נשלים אותו לפולינום בעל דרגה שהיא חזקה של 2 ע"י הוספת מקדמי 0).

$$P(x) = P_{\text{even}}(x^2) + x P_{\text{odd}}(x^2)$$

לכן, ניתן לחשב את ערכי $P(x)$ עבור $x = 1, w, \dots, w^{n/2-1}$ בזמן קבוע לפי P_{odd} ו- P_{even} (שכן הם מדרגה $n/2$).

עבור $x = w^{n/2}, \dots, w^{n-1}$ נשים לב ש:

$$w^{n/2+j} = -w^j, \quad 0 \leq j \leq n/2 - 1$$

$$\Downarrow$$

$$(w^{n/2+j})^2 = w^j, \quad 0 \leq j \leq n/2 - 1$$

לכן, לחישוב בנקודות $y = w^{n/2}, \dots, w^{n-1}$ צריך לחשב:

$$P(y) = P_{\text{even}}(x^2) - xP_{\text{odd}}(x^2)$$

 כאשר $x = 1, \dots, w^{n/2-1}$.

סה"כ: $T(n) = O(n \log n) \leftarrow T(0) = 1, T(n) = 2T(n/2) + cn$.

כפל פולינומים:

$$\begin{aligned} p(x) &= p_0 + p_1x + p_2x^2 + \dots + p_{m-1}x^{m-1} \\ q(x) &= q_0 + q_1x + q_2x^2 + \dots + q_{m-1}x^{m-1} \end{aligned} \quad r(x) = p(x)q(x)$$

הרעיון: נחשב $2m-1$ נקודות של פולינום המכפלה ולפיהם נמצא את מקדמי הפולינום המגדירים אותו, כלומר:
 נחשב $r(x_i) = p(x_i)q(x_i)$ עבור $i = 0, \dots, 2m-2$.

נבחר את w להיות שורש היחידה ה- n -י ונחשב ערכים עבור $1, w, \dots, w^{2m-2}$ ע"י FFT והזמן יהיה: $O(m \log m)$.
 חישוב הכפל בנקודות $r(x_i) = p(x_i)q(x_i)$ ייקח $O(m)$.

קיבלנו $F_{2m-1}R = V$, כאשר $V_i = r(w^i)$. אנחנו מעוניינים למצוא את R ולכן: $R = F_{2m-1}^{-1}V$.

טענה: F_n הפיכה ו- $F_n^{-1}[i,j] = 1/n w^{-ij}$, כלומר: השורה ה- i ב- F_n^{-1} היא השורה ה- $(n-i)$ ב- $1/n F_n$.
 לכן, ניתן לחשב את $F_n^{-1}V$ ב- $O(n \log n)$ ע"י FFT.

סה"כ: $O(n \log n)$ - חישוב $F_n V$
 $O(n)$ - חישוב $1/n F_n V$
 $O(n)$ - החלפת השורות בתוצאה

precision: לחישוב פתרון מדויק מספיק $O(\log n)$ ביטים. לכן אם מילה בזיכרון היא בעלת x ביטים, ניתן לחשב ב-FFT לפולינומים מדרגה 2^x .

• ניתן לבצע FFT מקבילי ב- $O(\log n)$.

ABF

אלגוריתם נאיבי למציאת מקומות התאמה בין טקסט באורך n לבין תבנית באורך m – $O(nm)$.

שיטת הדו-קרב (שיטת העדים):

הרעיון: דילול אורך המחרוזת שצריך לבדוק: בהינתן 2 מקומות בטקסט שרוצים לבדוק אם מהם מתחילה התאמה, ניקח נקודה שלישית שבעזרתה נוכל לבטל אחת מהנקודות ב- $O(1)$.

טבלת עדים: טבלת העדים היא **לתבנית**: עבור כל איבר בתבנית בודקים איזה תו בתבנית יוכל להכריע בדו-קרב ביחס לאיבר הראשון בתבנית, כלומר:

ניצור מערך W כך ש:

$$W[i] = \begin{cases} * & \text{אם לא ניתן לקבוע, לפי התבנית, מי "צודק"} \\ k & \text{כך ש-} P[k] \text{ הוא עד לביטול אחת הנקודות} \end{cases}$$

דוגמה:

P	A	A	B	B	A	A	B	B
---	---	---	---	---	---	---	---	---

W	*	3	3	4	*	7	7	8
---	---	---	---	---	---	---	---	---

הדו-קרב:

- אם נבדוק את כל הזוגות – $O(n^2)$.
- אם נבדוק רק את הזוגות שהמרחקים ביניהם קטן מ- m – $O(nm)$.

הרעיון:

מתחילים מ-2 הנקודות הראשונות בטקסט.

נבדוק תמיד את המקום האחרון הפוטנציאלי (כלומר: שטבלת העדים עבורו היא *) מול המקום הבא שעוד לא נבדק.

אם המיקום החדש "הפסיד" – נמשיך למיקום הבא.

אם המיקום האחרון הפוטנציאלי "הפסיד" – נבדוק את המיקום החדש מול המיקום הפוטנציאלי הקודם רק אם המרחק ביניהם קטן מ- m . אחרת, המקום החדש הופך להיות מקום פוטנציאלי חדש. בסוף נבדוק את כל המקומות הפוטנציאליים שיישארו.

פורמלית: דו-קרב בין i ל- j ($i < j$):

1. גש ל- $W[j - i + 1]$ (נותן את המיקום היחסי ביניהם).
2. אם רשום * – אין דו-קרב: הם מסכימים ביניהם ו- j הוא המיקום הפוטנציאלי החדש ו- $j + 1$ היא נק' הבדיקה הבאה.
3. אם רשום k : בודקים: אם $P[k] = T[i + k - 1]$ אז הורגים את j והמיקום הפוטנציאלי האחרון נשאר i . אחרת: הורגים את i ובודקים את j מול המיקום הפוטנציאלי הקודם.

דוגמה:

P	1	2	3	4	5	6
	A	B	C	A	B	C

T	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	A	B	C	C	B	C	A	B	C	A	B	C	B	C	A	B	C	B	B

W	*	2	3	*	5	6
---	---	---	---	---	---	---

מקומות פוטנציאליים בטקסט: 1, 4, 7, 14.

סיבוכיות:

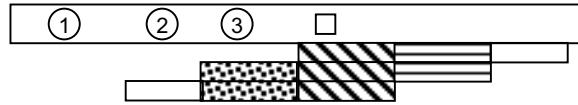
- כל עוד מתקדמים קדימה (כלומר, המיקום החדש מתבטל) – $O(n)$.
- אם מיקום ישן מתבטל אזי צריך לחזור אחורה. נחייב את האלמנט שבטל בחזרה אחורנית. כיוון שמיקום יכול להתבטל רק פעם אחת – $O(n)$.
- בסוף אנחנו נשארים עם מקומות פוטנציאליים (*). כיוון שבדו-קרב שבו יש * ממשיכים עם המיקום הפוטנציאלי החדש והוא לא יכול להתקדם יותר מ- n פעמים $\leftarrow$ אין יותר מ- n *.

נכונות: ① ② ③ □ – 3, 2, 1 – מקומות פוטנציאליים. □ – מיקום חדש.

1. אם □ מתבטל – אין בעיה.
2. אם (3) מתבטל – בודקים את □ מול (2).

3. אם $\square$ מסכים עם (3) (כלומר היה * בטבלת העדים) אזי לא צריך לבדוק אותו מול (2), כלומר: אם (2) ו-(3) מסכימים ו-(3) $\square$ מסכימים $\leftarrow$ (2) מסכימים.
למה זה נכון? הסכמה פירושה שהם מסכימים על החלק המשותף (החופף) שלהם: הסכמה זו היא **טרנזיטיביות**.

החפיפה של (2) ו- $\square$ היא תת-קבוצה של החפיפה של (3) ו- $\square$ וכל מקום שמשותף ל-(3) ול- $\square$ משותף גם ל-(2) ול- $\square$.



תוצאה: קיבלנו שכל המועמדים שלנו מסכימים זה עם זה, כלומר: אם התבנית מתחילה ב-2 מועמדים, אזי הם מסכימים על החלק המשותף שלהם.

בדיקת המועמדים:

- נעבור מועמד מועמד ובדוק. **הבעיה:** במקרה הגרוע זה יהיה $O(nm)$ (למשל: עבור $P = AAA...A$ כל הטקסט יהיה מועמדים).

פתרון – שיטת הגל:

כיוון שכל 2 מועמדים מסכימים ביניהם על החלק המשותף, אזי ניתן לבדוק את החלק המשותף מול רק מועמד אחד שכולל אותו, שכן שניהם מצפים לאותם תווים שם. השיטה:
1. עוברים על הטקסט ובכל מקום שבו מתחיל מועמד חדש מתחילים למספר מ-1 עד אורך התבנית.

דוגמה: (המקומות המודגשים הם המועמדים) $|P| = 6$

1	2	3	4	1	2	3	4	1	2	3	4	5	6	
---	---	---	---	---	---	---	---	---	---	---	---	---	---	--

- נעבור כעת על הטקסט ונשווה כל אלמנט שכתוב בו מספר i עם $P[i]$ ונכתוב:
נכתוב y אם התו שבטקסט (לא המספר) $P[i] =$ (כלומר: מה שכתוב בטקסט הוא נכון וזה נכון לכולם).
נכתוב n אחרת (כלומר: מה שכתוב בטקסט הוא לא נכון והוא לא נכון לכולם).

דוגמה: (באדום – מועמדים לא טובים, שכן ב-6 המקומות שלהם יש n . בכחול – מועמד טוב) $|P| = 6$

y	y	y	y	y	y	y	y	y	n	y	n	y	y	
---	---	---	---	---	---	---	---	---	---	---	---	---	---	--

- כדי לגלות את המקומות הלא-טובים עושים **גל** מה- n ים שמאלה. כל "n" הגל מתאפס ומתחיל מחדש.
- הערה:** עבור תמונה דו-מימדית $(n \times n, m \times m)$ הרעיון דומה: משתמשים ב-2 טבלאות עדים ובגל דו-מימדי. הסיבוכיות היא $O(n^2 \log m)$.
- הערה:** ABF לא עובד עם $\emptyset$ שכן הוא מסתמך על טרנזיטיביות.

סיכום תוצאות עבור Pattern Matching:

- התאמת תבניות ללא $\emptyset$:**
יש אלגוריתם **לזיהוי מדויק** (ללא ספירת שגיאות) ב- $O(n)$ (שזו הסיבוכיות האופטימלית, שכן חייבים לעבור על הטקסט) – ABF.

- התאמת תבניות עם $\emptyset$:**
 - עבור א"ב חסום קיים אלגוריתם **לזיהוי מדויק** בזמן $O(n \log m \log \sigma)$ – קידוד התווים + mismatch.
 - עבור א"ב $\{0, 1\}$ קיים אלגוריתם **לספירת שגיאות** בזמן $O(n \log m)$.
 - עבור א"ב חסום קיים אלגוריתם **לספירת שגיאות** בזמן $O(n|\pi| \log m)$.
 - עבור א"ב לא חסום קיים אלגוריתם **לזיהוי מדויק** בזמן $O(n \log m \log \sigma)$.
 - עבור א"ב לא חסום קיים אלגוריתם **לספירת שגיאות (אי-התאמות)** בזמן $O(n \sqrt{m} \log m)$ – kosaraju.

Less Than Matching:

עבור א"ב חסום: לכל תו בתבנית נבצע מכפלה בשביל לבדוק את מספר השגיאות שנגרם לכל מקום בו נציב את התבנית על הטקסט. החישוב יהיה כדלהלן:

$$\chi_{\leq \sigma}(a) = \begin{cases} 0 & , a \geq \sigma \vee a = \emptyset \\ 1 & , a < \sigma \end{cases} \quad \chi_{\sigma}(a) = \begin{cases} 1 & , \sigma = a \\ 0 & , \sigma \neq a \vee \sigma = \emptyset \end{cases}$$

$$V \leftarrow \sum_{\sigma \in \pi} V_{\sigma} \quad \text{נחשב } V_{\sigma} = \chi_{\leq \sigma}(T) \cdot \chi_{\sigma}(P^R) \quad \text{ובסוף נסכום:}$$

סה"כ:

עבור א"ב חסום נקבל: $O(n|\pi| \log m) = O(n \log m)$.

Levenshtein Erros (Edit Distance)

רוצים למצוא עבור כל מיקום בטקסט את המס' המינימלי של שגיאות (מסוג: הכנסה, מחיקה ו-mismatch) שעברו P מתאים ל-T במיקום שמתחילים ב-k (כלומר: אנחנו מחפשים את המס' המינימלי של שגיאה באות, הכנסה או מחיקה או מחיקה שנוצרת לבצע כדי להגיע להתאמה של התבנית לטקסט). כיוון שאנחנו מחפשים מיקומים k בהם נגמרת התבנית זה שקול לכך שההתאמה מתחילה במיקום שה-reverse של הטקסט נגמר ומתאים ל-reverse של התבנית.

- אלגוריתם נאיבי: $O(n^3)$ – עבור כל מקום בטקסט יש 3 אפשרויות לבדיקה.

הרעיון: תכנות דינמי:

נסמן ב- $ED(i, j)$ את המס' המינימלי של פעולות שצריך לעשות כדי ש- $p_1 p_2 \dots p_i$ יתאים לטקסט המסתיים ב- t_j .

1. אם $p_{i+1} = t_{j+1}$ אזי $ED(i+1, j+1) = ED(i, j)$.

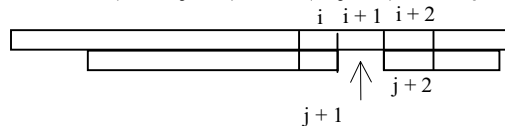
2. אחרת:

א. אם השגיאה היא מסוג mismatch, אזי: $ED(i+1, j+1) = ED(i, j) + 1$.

ב. אם השגיאה היא מסוג הכנסה: $ED(i+1, j+1) = ED(i+1, j) + 1$.

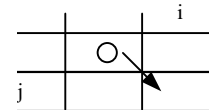


ג. אם השגיאה היא מסוג מחיקה: $ED(i+1, j+1) = ED(i, j+1) + 1$.

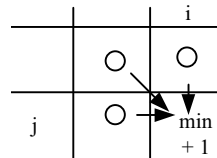


מטריצת התכנות הדינמי:

מיקום (i, j) = מס' הפעולות המינימלי שצריך לבצע כדי ש- $p_1 p_2 \dots p_i$ יתאים למחרוזת המסתיימת במיקום j בטקסט.



אם $p_i = t_j$:



אחרת:

אתחול המטריצה:

המחרוזת באורך 0 שנגמרת במקום ה-i.

	0	1	...	n
0	0	0	...	0
1	1			
.	.			
.	.			
m	m			

לפני שהתחיל הטקסט התאמנו i איברים ← צריך להוריד i איברים כדי שתהיה התאמה.

קצת נמלא את המטריצה שורה שורה משמאל לימין ומלמעלה למטה.

סה"כ: $O(nm)$.

Galil's Open Problem

הבעיה: בהינתן טקסט, תבנית ומס' k רוצים למצוא את כל המקומות ב- T בהם יש לכל היותר k mismatches.

ניתן לפתור את זה ב- $O(nk)$ ע"י suffix trees & LCA.

הרעיון:

בהינתן מחרוזת S באורך n . נעבד את S בזמן $O(n)$ באופן שיאפשר תשובות בזמן $O(1)$ לשאלות מהסוג הבא:

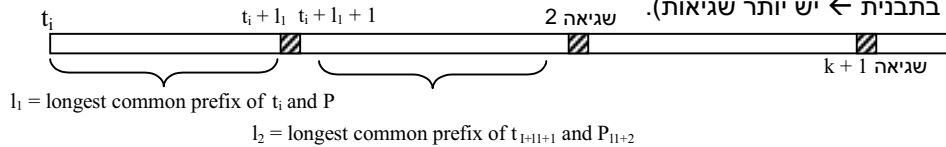
נתון: i ו- j אינדקסים ב- S , $0 \leq i, j \leq n$

שאלתה: האורך של הרישא המשותפת הגדולה ביותר של $S_i S_{i+1} \dots S_n$ ושל $S_j S_{j+1} \dots S_n$.

בעזרת זה ניתן לפתור את הבעיה של מציאת הופעות של התבנית בטקסט:

נבנה מחרוזת של הטקסט והתבנית יחד בזמן $O(n + m)$. כעת נשאל את השאלות עבור האינדקסים הבאים: $(1, n+1)$, $(2, n+1)$, $\dots$, כאשר נקבל תשובה $m \leftarrow$ יש הופעה של התבנית. אחרת - n .

באמצעות השאלתה $(i, n+1)$ ניתן למצוא מאיזה מקום מתחילה השגיאה. נניח קיבלנו l_1 , ז"א: יש l_1 מקומות שווים ולכן נשאל כעת עבור $(i + l_1 + 1, n + l_1 + 2)$ ונבדוק כמה מקומות שווים וכן הלאה. אם אנחנו מחפשים k שגיאות אזי צריך לקפוץ לכל היותר $k + 1$ פעמים **ולסיים** לעבור על כל התבנית (שכן אם יש עוד בתבנית $\leftarrow$ יש יותר שגיאות).



סה"כ: $O(k)$ קפיצות, כל קפיצה ב- $O(1)$, n מקומות $\leftarrow O(nk)$.

הפתרון של Galil's Open Problem עבור Edit Distance

הרעיון: נעשה תכנות דינמי על המטריצה D של התכנות הדינמי.

נשים לב שמעניין אותנו כל האלכסונים שחותכים את השורה האחרונה במטריצה (כלומר, שהתבנית נגמרת בטקסט). יש n אלכסונים כאלה ולכל אלכסון נרצה למצוא ב- $O(1)$ את השורה התחתונה ביותר אליה מגיע האלכסון עם אותו מספר. ע"י שאלתה של $i, n+1$ (כאשר $i = \text{תחילת האלכסון}$) ניתן למצוא את הגבול של האלכסון.

נשים לב גם שהאלכסון משפיע על האלכסון שמעליו ועל האלכסון שמתחתיו. נתקדם על האלכסונים עד אשר המס' יהיה גדול מ- k או עד השורה האחרונה.

מימוש:

נגדיר את אלכסון d להיות האלכסון בו $j - i = d$ ב- $D[i, j]$, כלומר:

מעניינים אותנו האלכסונים מ- $-m$ עד $m - n$ (הם האלכסונים שמגיעים עד השורה האחרונה) - סה"כ n אלכסונים.

נבנה מטריצה L שעבורה:

$L[d, e] = \text{השורה הגדולה ביותר אליה מגיע אלכסון } d \text{ עם מס' שגיאות } e$.

כל אלכסון $L_{d,k}$ שהמס' הרשום בו הוא m הוא אלכסון טוב (שכן הגענו ל- k שגיאות וסיימנו את התבנית). אם רשום שם מס' קטן מ- $m \leftarrow$ יש כבר k שגיאות ועוד לא הגענו לסוף $\leftarrow$ האלכסון אינו טוב.

		1	2		
	0	1	...	m	
-1	0				
-2	1				
	.				
	.				
	.				
	m				

דוגמה:

	0	1	2	3	4
-4	-	-	-	-	4
-3	-	-	-	4	
-2	-	-	4		
-1	-	4			
0	2	3	4		
1	0	4			
2	0	4			

L

דוגמה:

		1	2					
	0							
-1								
-2								
-3								
-4								

D

אם נרצה למצוא עבור $k = 1$ אזי האלכסונים המתאימים הם $-1, 1, 2$.

האלגוריתם:

אתחול:

	-1	0	1	2	...	k-1	k
-k						k-1	
-(k-1)					.		
.				.			
.				.			
.				.			
-2			1				
-1		0					
0	-1						
1	-1						
.	.						
.	.						
.	.						
(n - m)	-1						

כעת ממלאים את השורות משמאל לימין
ומלמעלה למטה, כך שהערך של $L_{d,e}$ יהיה
כך ש-
 $r = \max(L_{d,e-1} + 1, L_{d+1,e-1} + 1, L_{d-1,e-1})$

הערך הזה ימשיך באלכסון כל עוד התבנית
מתאימה לטקסט, כלומר:

$$p_{r+1} = t_{r+d+1}$$

$$p_{r+2} = t_{r+d+2}$$

.

.

.

לבסוף, כל שורה של L שבה מגיעים ל- m היא מופע טוב של התבנית.

כל מילוי של ערך בטבלה ניתן לעשות ע"י suffix trees ו-LCA ב- $O(1)$.
סה"כ למלא טבלה בגודל $2nk - O(nk)$.

התאמת מחרוזות באמצעות אוטומט סופי

זהו אלגוריתם לזיהוי התאמות (לא לספירת שגיאות). הוא לא מאפשר $\emptyset$ (בגלל טרנזיטיביות).

האוטומט:

נבנה בעיבוד מקדים אוטומט לתבנית שעליו נוכל להריץ את הטקסט.

הרעיון: מצב q אומר שראינו $P[1..q]$. נרצה לעבור ממצב q למצב q' כך ש- q' הוא המקסימלי המקיים ש- $P[1..q']$ הוא רישא של התבנית עצמה וסיפא של $P[1..q]$.

דוגמה: $P = ababca$. אני במצב q_4 (כלומר: ראיתי עד כה: $abab$). נניח שאני רואה עכשיו a . לאן צריך לעבור? כיוון שראיתי עכשיו a אזי סה"כ ראיתי $ababa$ (*). הרישא הגדולה ביותר של התבנית שהיא סיפא של מה שראיתי עד עכשיו (*) זה aba ולכן אני אחזור למצב q_3 .

ברגע שמגיעים ל- $q_m \leftarrow$ התבנית מתאימה ל- m התווים הקודמים $\leftarrow$ יש התאמה.

סה"כ:

סריקת הטקסט בעזרת האוטומט (מעבר אחד על האוטומט עם הטקסט) – $O(n)$.
 בניית האוטומט (בכל מצב יש התייחסות לכל תו אפשרי): $O(m|\Sigma|)$.
 סה"כ: $O(n + m|\Sigma|)$.

KMP – Knuth, Morris, Pratt

הרעיון: שמירת אינפורמציה שנצברו מהשוואות קודמות ע"י חישוב פונקציית $Prefix$ עבור התבנית, כך שניתן יהיה בעת בדיקת הטקסט לפסול מקומות לבדיקה.
 פונקציית ה- $Prefix$ אומרת: **בכמה ניתן להשתמש מההשוואה הקודמת אם יש אי-התאמה.**

הגדרת פונקציית הרישא: פונקציית הרישא מוגדרת להיות הרישא המקסימלית של התבנית $P[0..j]$ שהיא גם הסיפא של $P[1..j]$.

j	0	1	2	3	4	5
$P[j]$	a	b	a	b	a	c
$\Pi[j]$	0	0	1	2	3	0

דוגמה:

ניתן לראות שאם יש כישלון ב- (4) אזי ה- a, b במקומות 2 ו-3 זהים לאלו שבמקומות 0 ו-1.

$f \leftarrow \text{KMPPrefixFunction}(P)$ {build failure function}

האלגוריתם:

$i \leftarrow 0$

$j \leftarrow 0$

while $i < n$ do

if $P[j] = T[i]$ then

if $j = m - 1$ then

return $i - m - 1$ {a match}

$i \leftarrow i + 1$

$j \leftarrow j + 1$

else if $j > 0$ then {no match, but we have advanced}

$j \leftarrow f(j-1)$ {j indexes just after matching prefix in P}

else

$i \leftarrow i + 1$

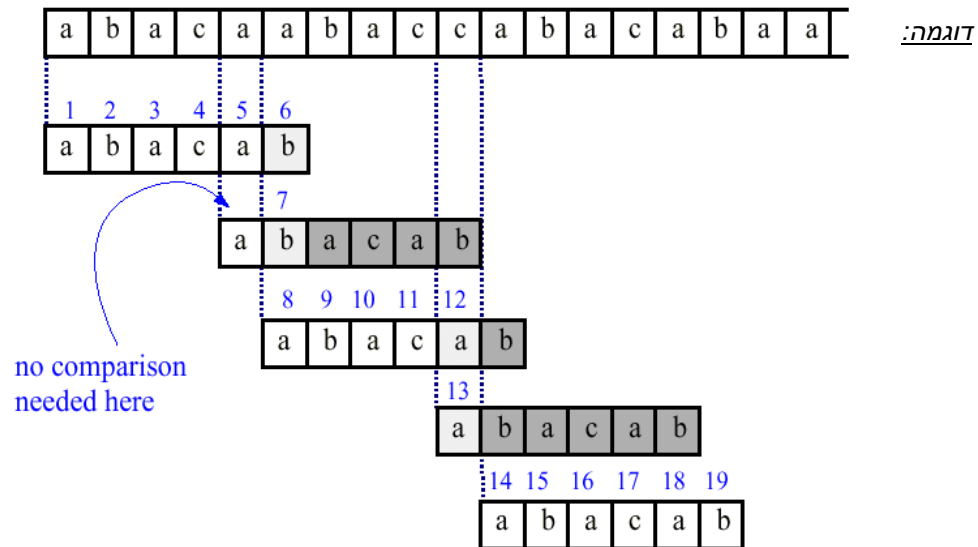
return "There is no substring of T matching P "

סיבוכיות:

מציאת ההתאמות – $O(n)$.

חישוב ה- $Prefix$ – $O(m)$.

סה"כ: $O(n + m)$.



Rabin-Karp

הרעיון: נתרגם כל תו לספרה ואת רצף הספרות של התבנית שיוצר מספר נחפש בטקסט שגם הוא מתורגם לספרות. כך מציאת התאמה לא נעשית תו-תו אלא עובר כמה תווים יחד, כשכל השוואה היא $O(1)$.

האלגוריתם:

1. כל אות בא"ב עוברת למספר בבסיס $d = |\Sigma|$.
2. ניצור טקסט ותבנית חדשים לפי המספרים.
3. נשווה את המספר בתבנית מול המספרים בטקסט: עוברים ממספר אחד לבא אחריו ע"י הורדת הספרה המובילה והוספת הספרה האחרונה.

$$P = P[m] + dP[m-1] + d^2P[m-2] + \dots + d^{m-1}P[1]$$

באופן כללי: והקפיצה ממספר שמתחיל ב- t_s לבא אחריו בטקסט שמתחיל ב- t_{s+1} נעשית כך:

$$t_{s+1} = (t_s - d^{m-1} \cdot T[s]) \cdot d + T[s+m+1]$$

דוגמה: $020 \rightarrow 203$ ($d = 10$): $(020 - 10^2 \cdot 0) \cdot 10 + 3$

סיבוכיות:

- הפיכת התבנית והטקסט: $O(n + m)$.
- בחירת מספר באורך m : $O(m)$.
- השוואת מספר בגודל m : $O(1)$.
- מעבר למספר הבא בטקסט: $O(1)$.
- סה"כ: $O(n + m)$.

בעיה: אם אורך התבנית (m) גדול מאוד אזי השוואות מספרים גדולים אינה פעולה אלמנטרית.

פתרון: נבחר מספר ראשוני q ונחשב כל פעם את P ואת t_s מודולו q (לרוב נבחר q כך ש- dq ייכנס למילת מחשב אחת).

בעיה: אם $t_s \neq p \bmod q$ נפסול את המקום. אולם אם $t_s = p \bmod q$ זה לא אומר ש- $t_s = p$. לכן עבור מקומות כאלו נצרך לבדוק התאמה ממש של התבנית על אותו מקום בטקסט (Brute-Force).

סיבוכיות הזמן: $O((n - m + 1)m)$.

תוחלת הזמן של האלגוריתם (אם נבחר q מספיק גדול והעובדה שאין מופעים רבים של התבנית בטקסט): $O(n + m)$.

גרפים

BFS – חיפוש לרוחב (משתמשים ב-FIFO – תור) – $O(V + E)$.
הרעיון: מוסיפים קודם את כל הילדים **באותה רמה** ואח"כ ממשיכים לרמה הבאה.
יעיל למציאת מרחקים.

DFS – חיפוש לעומק (משתמשים ב-LIFO – מחסנית) – $O(V + E)$.
הרעיון: לא גומרים שכבה-שכבה אלא "משפחה-משפחה" – ממשיכים עם כל הילדים עד שגומרים ילד-ילד.
יעיל למציאת מעגלים, מציאת רכיבים קשירים, מיון טופולוגי ובדיקה האם גרף הוא DAG.

סיווג הקשתות:

DFS בגרף לא מכוון:

קשת בעץ.

קשת אחורה.

DFS בגרף מכוון:

קשת בעץ.

קשת אחורה.

קשת קדימה.

קשת הצידה (לרוחב).

טענה: גרף לא מכוון הוא ללא מעגלים אם"ם אין קשתות אחורה ב-DFS כלשהו שלו.
משפט: קיים בגרף G מעגל אם"ם לכל DFS של G יש קשת אחורה.

רכיב קשיר: תת-גרף קשיר מקסימלי, כלומר: קבוצה מקסימלית של קודקודים שבין כל זוג יש מסלול.
רכיב קשיר חזק: רכיב קשיר בגרף מכוון.

- הערה: כל הקודקודים צריכים להיות באחד הרכיבים הקשירים (חזק), אבל לא כל הקשתות צריכות להיות בתוך הרכיבים הקשירים.

רכיבים דו-קשירים בגרף לא מכוון:

נק' חיתוך: קודקוד הוא **נקודת חיתוך** אם הוצאתו מהגרף הקשיר הופכת את הגרף לגרף לא קשיר.
הגדרה שקולה: קודקוד v הוא נקודת חיתוך אם קיימים קודקודים שונים v, w כך ש-v נמצא בכל מסלול מ-x ל-w.

גרף דו-קשיר: גרף שאין בו אף נקודת חיתוך.

הגדרה שקולה (1): גרף שאם נוציא ממנו קודקוד כלשהו עדיין יישאר תת-גרף קשיר.

טענה: אם הגדרה (1) מתקיימת אזי לכל מסלול פשוט באורך גדול/שווה ל-2 $v_0, v_1, \dots, v_k$ קיים מעגל שמכיל את הקשתות v_0v_1 ואת $v_{k-1}v_k$.

רכיב דו-קשיר: תת-גרף דו-קשיר מקסימלי, כלומר: קבוצה מקסימלית של קודקודים שמהווה גרף דו-קשיר (כלומר: תת-גרף קשיר שהוצאת קודקוד כלשהו ממנו עדיין משאירה תת-גרף קשיר).
הגדרה שקולה: רכיב דו-קשיר הוא תת-גרף שבו לכל זוג קשתות e, e' מתקיים:
1. $e' = e$ או 2. קיים מעגל משותף פשוט המכיל את e ואת e' .

היחס הנ"ל הוא **יחס שקילות**:

1. רפלקסיבי: $B(e, e)$.

2. סימטרי: $B(e, e') \rightarrow B(e', e)$.

3. טרנזיטיבי: $B(e, e') \text{ and } B(e', e'') \rightarrow B(e, e'')$.

← רכיב דו-קשיר בגרף הוא מחלקת שקילות (עם הקודקודים שלו) תחת יחס השקילות הנ"ל.

זיהוי נקודות חיתוך בגרף ע"י DFS – $O(V + E)$

- שורש ה-DFS הוא נקודת חיתוך אם יש לו יותר מבן אחד.
- עלה בעץ ה-DFS הוא אף פעם לא נקודת חיתוך.
- צומת פנימי v הוא נקודת חיתוך אם"ם **קיים תת-עץ** (בעץ ה-DFS) ש-v הוא שורשו, כך שכל הקשתות אחורה היוצאות מתת-עץ על v או על צאצא של v (הגדרה שקולה: קיים תת-עץ ש-v הוא השורש שלו כך שאין ל-v קשתות אחורה מתת-עץ זה).

מציאת כל הרכיבים הדו-קשירים: נמצא את כל נקודות החיתוך: נקודות החיתוך מגדירות את רכיבי הדו-קשירות.

טענות:

1. $A = (V_A, E_A), B = (V_B, E_B)$ רכיבים קשירים בגרף לא-מכוון אזי:
 א. $V_A \cap V_B = \emptyset$ – אחרת ניתן היה לאחד את הרכיבים דרך הקודקוד המשותף $\leftarrow$ סתירה למקסימליות של A ושל B.
 ב. $E_A \cap E_B = \emptyset$ – אם יש קשת משותף אזי בהכרח יש קודקוד משותף ולפי 1 זה לא מתקיים.
2. $A = (V_A, E_A), B = (V_B, E_B)$ רכיבים דו-קשירים בגרף לא-מכוון אזי:
 א. ייתכן ש- $V_A \cap V_B \neq \emptyset$
 ב. $E_A \cap E_B = \emptyset$ – אם יש קשת משותף אזי בהכרח יש מעגל משותף לכולם (לפי הטרנזיטיביות) וזה מאחד את הרכיבים $\leftarrow$ סתירה למקסימליות.
 ג. ייתכן ש- $A \cap B = \emptyset$
3. $A = (V_A, E_A), B = (V_B, E_B)$ רכיבים קשירים חזק בגרף מכוון אזי:
 א. $V_A \cap V_B = \emptyset$
 ב. $E_A \cap E_B = \emptyset$