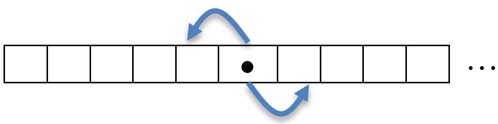
מודלים חישוביים בקורס (ובחיים)

[עד כה דיברנו על סיבוכיות של אלגוריתמים בעזרת הערכות כלליות ($\mathrm{O}(\cdots)$), ולא נכנסנו לדבר על איך מבצעים פעולות כמו חיבור מספרים או שימוש במצביעים לזיכרון.
ראינו בעבר [[בקורס *אוטומטים, שפות פורמליות וחישוביות*]] מודל חישוב בשם *מכונת טיורינג*, שאמנם שימושית עבור מטרות תאורטיות, אך היא די רחוקה מהמציאות מבחינת מימוש.
ע"מ שנוכל לדון בדברים מסוימים בהמשך, נציג כעת רעיון כללי של מודל נוסף, אשר יותר דומה לכלים הקיימים במציאות שלנו כיום.]

סיבוכיות (נדון בזאת החל משבוע הבא)	אלגוריתמים יעילים / מבני נתונים
• מודל: מכונת טיורינג • תכונות: ○ סרט אינסופי + מצב פנימי ○ קריאה / כתיבה במקום אחד בסרט ○ הזזת הראש ימינה / שמאלה:  • טוב ע"מ לענות על השאלה: "מה אפשר לענות בזמן פולינומי?"	• מודל: Word-RAM • תכונות: ○ רגיסטרים ○ קריאה מכתובת ("pointer") ○ פעולות אריתמטיות באמצעות רגיסטר מיוחד בזמן $\mathrm{O}(1)$ ○ רגיסטר חייב להיות מספיק גדול כדי להצביע לכל פריט בקלט (למשל קדקודים, קשתות) $\Leftarrow$ רגיסטר בגודל $\mathrm{O}(\log n)$ • טוב ע"מ לענות על השאלה: "מה אפשר לעשות בזמן לינארי?"

<u>הנחה</u>: פעולה אריתמטית על מספרים בעלי $\mathrm{O}(\log n)$ ביטים לוקחת זמן $\mathrm{O}(1)$.

$n =$ גודל הקלט (מס' ביטים)

[[הסבר: בד"כ במחשב יש רגיסטרים בגודל k סיביות, ופעולות אריתמטיות בין רגיסטרים מתבצעת בזמן קבוע. במילים אחרות, אם נתון לנו שהמידע שלנו מכיל מספרים שערכם הוא לכל היותר n, אז אם נסמן $k = \lceil \log_2 n \rceil$, ונניח שיש לנו מכונת חישוב עם רגיסטרים בגודל k סיביות, אז נוכל להניח שפעולה אריתמטית בין מספרים שכאלה מתבצעת בזמן קבוע.
לעומת זאת, אם יש לנו מעבד שגודל רגיסטר בו הוא k סיביות, ויש לנו ערך שגודלו $n = 2^k$ סיביות, אז ודאי שלא נוכל לעשות עליו פעולות אריתמטיות בזמן קבוע.]]

[להוסיף הסבר לגבי ההנחה]

בעיית התאמת המחרוזות

- <u>קלט</u>:
  - מחרוזת "ארוכה" $T[0,\ldots,n-1]$

- ○ מחרוזת "קצרה" $P[0,\ldots,k-1]$ $(k \le n)$

<u>הגדרה</u>: P מופיע ב- T עם הזזה s אם:

$$P[0,\ldots,k-1] = T[s,s+1,\ldots,s+k-1] =: T_s$$

- <u>פלט</u>: כל המיקומים s כך ש- $P = T_s$.

<u>דוגמה</u>:

$$T = 0110100\underline{1}000$$

$$P = 0100$$

<u>שימושים</u>:

- Ctrl+F [כלומר חיפוש בקובץ]
- מציאת תבנית ברצף של DNA

אלגוריתם נאיבי

- לכל $s = 0,\ldots,n-k$:
  - ○ נקרא את T_s ונוסיף את s לפלט אם $T_s = P$.

<u>זמן הריצה</u>:

- $n-k+1$ איטרציות בלולאה
- כל השאר לוקחת זמן k
- $\Leftarrow$ סה"כ $O(k(n-k+1)) = O(nk)$

אלגוריתם אקראי

נראה אלגוריתם אקראי בזמן $O(k+n)$.

זהו <u>חסום הדוק</u>: כי צריך זמן $k+n$ רק כדי לקרוא את הקלט.

תיאור פשוט לאלגוריתם Karp-Rabin

- נגריל מספר ראשוני $q < N = kn^2$
- נחשב את $b \leftarrow P \bmod q$
- לכל $s = 0,\ldots,n-k$:
  - ○ נחשב $a_s \leftarrow T_s \bmod q$
  - ○ אם $a_s = b$, נוסיף את s לפלט

[הערה: עד כה בקורס כשהראינו אלגוריתם הוכחנו שהוא נכון (=תמיד מחזיר תשובה נכונה) וניתחנו את זמן הריצה שלו. באלגוריתמים הסתברותיים אנו לא נראה שהאלגוריתם תמיד מחזיר תשובה נכונה, אלא שההסתברות שהאלגוריתם ייתן תשובה שגויה קטנה מספיק.
אבל **שימו לב**: לא מדובר על ההסתברות לתשובה שגויה עבור קלט רנדומלי, אלא **עבור כל קלט שהוא**, **תמיד** ההסתברות לשגיאה היא לכל היותר מה שאנו רושמים (למעשה אנו לא מגרילים קלט; הוא נתון לנו וזהו; אנו מגרילים ערכים הקשורים במציאת הפתרון).]

הסתברות להצלחה

- משפט: לכל קלט, ההסתברות שהאלגוריתם מחזיר תשובה שגוייה היא $\mathrm{O}\left(\frac{\log n}{n}\right)$.
- עובדה 1: יש לפחות $\frac{t}{\ln t}$ מס' ראשוניים קטנים מ-t (עבור t גדול דיו) [גמדים קטנים הוכיחו לנו את זה בלילה].
- עובדה 2: אם $w<2^t$ אזי יש פחות מ-t ראשוניים שמחלקים את w בלי שארית [ראינו את ההוכחה בשיעור שעבר].

הוכחה [[למשפט]]:
[הערה: אנחנו מתייחסים כאן לסדרה של ספרות בינאריות כמספר – למשל הסדרה 000110 היא המספר 6.]
לכל s כך ש-$P=T_s$, המיקום s יופיע בפלט [[כי אז גם $P \bmod q = T_s \bmod q$]].

הבעיה: אם יש s עבורו $P \neq T_s$ אבל $P \equiv T_s \bmod q$ [[זו אחת הדרכים לרשום ש-$P \bmod q = T_s \bmod q$]] (כלומר $a_s=b$) [[לדוגמה, $7\neq 4$ אבל $7 \bmod 3 = 1 = 4 \bmod 3$, לכן $7 \equiv 4 \bmod 3$]].

רוצים לחסום את ההסתברות שקיים s כזה (ראינו בשיעור שעבר שזה קורה עבור s כך ש-$0<w_s:=|P-T_s|$ מתחלק ב-q ללא שארית).

נשתמש בעובדה: q מחלק לפחות $0<w_s$ אחד $\Leftrightarrow$ q מחלק את $w:=\prod_{s:P\neq T_s} w_s$ [כלומר מס' ראשוני מחלק מכפלה אם"ם הוא מחלק לפחות אחד מהגורמים בה].

[[ע"מ לבדוק אם קיים w_s כזה, נבדוק אם q מחלק את מכפלת כל ה-w_s.
שימו לב שהאלגוריתם לא צריך לעשות את הבדיקה הזו; אנחנו עושים אותה בהוכחה, אבל האלגוריתם עצמו מבצע אך ורק את מה שכתבנו בתיאור.]]

P,T_s מספרים בעלי k ביטים $\Leftarrow$ $P,T_s<2^k$ $\Leftarrow$ $w_s=|P-T_s|<2^k$.
מספר ה-s-ים הבעייתיים $\geq$ מס' ה-s-ים בכלל $n\geq$ (מיקומים אפשריים בתוך T).
ניקח $w=\prod_{s:w_s\neq 0} w_s$, ונקבל:

$$w=\prod_{s:w_s\neq 0} w_s<\left(2^k\right)^n=2^{kn}$$

לכן לפי עובדה 2, יש $nk >$ מספרים ראשוניים שמחלקים את w.

לכן ההסתברות לשגיאה היא:

$$(\text{הסתברות לשגיאה}) = \frac{\text{מס' ה-}p\text{-ים הרעים} \le nk}{\text{מס' ה-}p\text{-ים שאני יכולה לבחור} > \frac{N}{\ln N}} \le \frac{nk\ln N}{N} = \frac{nk\left(\ln\left(kn^2\right)\right)}{kn^2} = \frac{\ln k + 2\ln n}{n} = \mathrm{O}\left(\frac{\log n}{n}\right)$$

שיפור ההסתברות לנכונות

ראינו אלגוריתם שטועה בהסתברות $\mathrm{O}\left(\frac{\log n}{n}\right)$.

ישנם שני דברים שניתן לעשות ע"מ לשפר את ההסתברות להצלחה (ע"י הקטנת ההסתברות לשגיאה):

1. לחזור l פעמים על האלגוריתם ולהחזיר רק s-ים שמתקבלים **בכל** הרצה. ההסתברות לטעות:

$$\mathrm{O}\left(\left(\frac{\log n}{n}\right)^l\right)$$

[<u>שימו לב</u>: שינוי זה מגדיל את זמן הריצה פי l.]

2. להגדיל את הפרמטר N.
למשל, אם $N = kn^{10} \Leftarrow$ ההסתברות לשגיאה היא:

$$\mathrm{O}\left(\frac{\log n}{n^9}\right)$$

<u>שימו לב</u>: אם $N \le poly(n)$]] כלומר אם N הוא לכל היותר פולינומיאלי ב-n [[אזי $\log N = \mathrm{O}(\log n)$.

ניתוח זמן ריצה

]] בכל אלגוריתם הסתברותי יש לענות על השאלות הבאות: [[

- <u>שאלה ראשונה</u>: באיזו הסתברות האלגוריתם נכון?
- <u>שאלה שנייה</u>: מה זמן הריצה?

]] ענינו כבר על הראשונה (ההסתברות להצלחה שווה ל-1 פחות ההסתברות לכישלון); כעת נתמקד בשאלה השנייה. [[

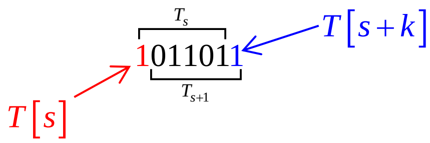
<u>מימוש נאיבי</u>:

- כל חישוב $a_s = T_s \bmod q$ לוקח זמן $\mathrm{O}(k)$ (לפי מס' הביטים ב-T_s).
- $\Leftarrow$ סה"כ $\mathrm{O}(nk)$ רק עבור הלולאה]] שזה לבד כבר גרוע כמו האלגוריתם הנאיבי [[.

<u>שיפור משמעותי</u>: זמן $O(k)$ לחישוב a_0, אבל זמן $O(1)$ לכל $0<s$.

איך?

<u>אבחנה [נראה בעזרת דוגמה]</u>:



לכן:

$$T_{s+1}=2T_s+T[s+k]-T[s]\cdot 2^k$$

בחשבון מודולרי [[= חישובים עם פעולות מודולו (mod)]] מתקיים:

$$(x+y)\bmod q=((x\bmod q)+(y\bmod q))\bmod q$$
$$(x\cdot y)\bmod q=((x\bmod q)\cdot(y\bmod q))\bmod q$$

אזי מהאבחנה:

$$a_{s+1}=T_{s+1}\bmod q=\left(2\underbrace{(T_s\bmod q)}_{=a_s}+T[s+k]-T[s]\cdot\underbrace{(2^k\bmod q)}_{=:c}\right)\bmod q$$

[הביטוי המסומן באדום הוא בדיוק a_s, ואותו כבר יש לנו מהצעד הקודם.

לחשב את $(2^k\bmod q)$ לוקח זמן $O(k)$, ליתר דיוק – אבל ערך זה לא תלוי ב- s, לכן בכל השלבים של החישוב הוא זהה. נקרא לערך זה $c:=(2^k\bmod q)$, ונחשב את c פעם אחת בתחילת הריצה וזהו.]

$\Leftarrow$ פעולות אריתמטיות על מס' $a_s,c,\dots$ בכלי $O(\log N)=O(\log n)$ ביטים $\Leftarrow$ חישוב a_{s+1} לוקח זמן $O(1)$ [לפי ההנחה שלנו על פעולות אריתמטיות של מספרים בגודל $O(\log n)$].

[נזכיר:

- T – n ביטים
- P – k ביטים
- $q<poly(n)$
- $\Leftarrow$ מס' ביטים ב- q הוא $\geq O(\log n)$

]

<u>סיכום זמני ריצה</u>:

- הגרלת מספר ראשוני q: זמן $O\left((\log n)^3\right)$ [לא נסביר כעת למה; סמכו עלינו לגבי זה בינתיים]
- חישוב $P \bmod q$: זמן $O(k)$
- חישוב $c = 2^k \bmod q$: זמן $O(k)$
- חישוב $a_0 = T_0 \bmod q$: זמן $O(k)$
- חישוב $a_1, a_2, \ldots, a_{n-k}$: זמן $O(n)$

- סה"כ $O(n+k)$