תורת החישוביות – תרגול מס' 11
סיבוכיות זכרון ומכונות אי דטרמיניסטיות

סיבוכיות זכרון

בהינתן מכונת טיורינג (חד סרטית) M, פונקציה $s:\mathbb{N}\to\mathbb{N}$ נקראת **חסם סיבוכיות זכרון** של M אם לכל x, M בריצתה על x אינה עוברת את תא מספר $s(|x|)$. אומרים של־M **סיבוכיות זכרון פולינומית** אם קיים פולינום $p(n)$ שהוא חסם סיבוכיות זכרון של M.

כשם שהגדרנו את P להיות מחלקת השפות הניתנות לקבלה על־ידי מכונת טיורינג עם חסם סיבוכיות זמן ריצה פולינומי, נגדיר כעת: $\{$קיימת מכונת טיורינג M בעלת סיבוכיות זכרון פולינומית כך ש־$L(M)=L\}$ =PSPACE.

האבחנה המעניינת הראשונה בקשר ל־PSPACE היא ש־$\mathrm{PSPACE}\subseteq\mathrm{R}$. ההוכחה מסתמכת על הטכניקה של ספירת קונפיגורציות שכבר הוצגה בעבר. אם $L\in\mathrm{PSPACE}$ ו־M מכונה המקבלת אותה בעלת חסם זכרון $p(n)$ אז נבנה מכונה M' שעל קלט w מריצה את M על w וסופרת את מספר צעדי M. אם M ענתה, M' עונה כמוה; ואחרת, אם M ביצעה יותר מדי צעדים, M' עוצרת ודוחה.

מספר הקונפיגורציות המקסימלי של M בריצתה על הקלט w הוא $|Q|\cdot p(|w|)\cdot|\Gamma|^{p(|w|)}$, ומספר זה ניתן לחישוב באמצעות M'. יותר מכך – ניתן לשמור מספר זה בזכרון **פולינומי** ב־$|w|$, שכן כדי לייצג את המספר k נדרשים רק $\lg k$ ביטים של זכרון, ולכן את מספר הקונפיגורציות המקסימלי ניתן לאחסן ב־

$$\lg\left(|Q|\cdot p(|w|)\cdot|\Gamma|^{p(|w|)}\right) \quad = \quad \lg|Q|+\lg p(|w|)+p(|w|)\lg|\Gamma| = O(p(|w|))$$

זכרון.

מכאן שהמכונה M' שבנינו לא רק מכריעה את $L(M)$, אלא גם היא עצמה פועלת בזכרון פולינומי. מכאן שלכל שפה $L\in\mathrm{PSPACE}$ ניתן להניח כי קיימת מכונה M בעלת סיבוכיות זכרון פולינומית שתמיד עוצרת.

האבחנה השניה היא כי $\mathrm{P}\subseteq\mathrm{PSPACE}$, מה שנובע מייד מכך שאם המכונה מבצעת רק k צעדים, היא לא יכולה להשתמש ביותר מאשר k תאי זכרון (כלומר, המכונה לא יכולה לבזבז יותר זכרון מאשר זמן) ולכן מגבלה על הזמן גוררת מיידית מגבלה על הזכרון, אך לא להיפך.

אי דטרמיניזם ו־PSPACE

כפי שראינו בעבר, NP היא מחלקת כל השפות שניתנות להכרעה על ידי מכונה **אי דטרמיניסטית** בעלת סיבוכיות זמן פולינומית. ציינו כבר כי מכונה דטרמיניסטית היא מקרה פרטי של מכונה אי־דטרמיניסטית, ולכן $\mathrm{P}\subseteq\mathrm{NP}$; בזמן פולינומי מכונה יכולה להשתמש רק במספר פולינומי של תאי־זכרון ולכן $\mathrm{P}\subseteq\mathrm{PSPACE}$. אבל מה היחס בין NP ו־PSPACE? נראה כעת כי $\mathrm{NP}\subseteq\mathrm{PSPACE}$.

תהי M מכונה אי דטרמיניסטית הפועלת בסיבוכיות **זמן** פולינומית, כלומר עומק עץ החישוב שלה על w הוא פולינומי ב־$|w|$. כמו כן, כל קונפיגורציה של ריצת M היא פולינומית ב־$|w|$בגודלה שכן כבר ראינו כי חסם על זמן הריצה גורר חסם על כמות הזכרון שהמכונה משתמשת בו.

נרצה להראות שאפשר לחפש מצב מקבל בעץ החישוב של M על w תוך שימוש בזיכרון פולינומי. למעשה, ראינו תעלול דומה בעבר – עבור מכונות שאינן מוגבלות חישובית, ראינו שאי דטרמיניזם לא מוסיף כוח חישוב ל־RE. שם, הנחנו ש־M אי־דטרמיניסטית מקבלת שפה L, ואז ביצענו חיפוש BFS דטרמיניסטי בעץ החישוב שלה. במקרה שלפנינו, חיפוש BFS לא מתאים, כי הוא עשוי לדרוש זיכרון אקספוננציאלי בעומק העץ, ולכן, ב־$|w|$.

לעומת זאת, ניתן לבצע סריקת DFS של עץ החישוב בחיפוש אחר קונפיגורציה מקבלת, ואם נמצאה כזו – עוצרים ומקבלים (ואחרת דוחים בסיום החיפוש). לצורך כך יש לזכור במחסנית את כל הקונפיגורציות במסלול שעברנו עד כה בעץ, אך מכיוון שיש מספר פולינומי של קונפיגורציות שכאלו, וייצוג של כל קונפיגורציה דורש זכרון פולינומי בלבד, האלגוריתם לא יחרוג מסיבוכיות הזכרון הפולינומית הנדרשת ממנו. נשים לב כי סיבוכיות **זמן הריצה** של האלגוריתם שלנו היא גרועה – אקספוננציאלית בעומק העץ (שהוא זמן הריצה של M).

משפט סביץ' (Savitch)

השאלה האם P=NP היא השאלה הפתוחה המרכזית במדעי המחשב התיאורטיים ואחת מהשאלות הפתוחות המרכזיות במתמטיקה בכלל (הקונצנזוס בקרב העוסקים בנושא הוא כי $\mathrm{P} \neq \mathrm{NP}$).

ניתן לשאול שאלה דומה גם עבור סיבוכיות זכרון. נסמן ב־NPSPACE את מחלקת השפות שיש מכונה אי דטרמיניסטית בעלת סיבוכיות זכרון פולינומית המקבלת אותן. האם PSPACE=NPSPACE? משפט סביץ' (Savitch) מראה כי התשובה לשאלה זו **חיובית**.

אם כן, תהא M אי דטרמיניסטית בעלת סיבוכיות זכרון פולינומית, $p(|w|)$, ואנו רוצים לבדוק האם בריצתה על מילה w יש לה מסלול חישוב מקבל. לרוע המזל, הפתרון הנאיבי של ביצוע DFS בעץ החישוב (מה שהוכיח כי $\mathrm{NP} \subseteq \mathrm{PSPACE}$) אינו עובד כאן, מהטעם הפשוט שעץ החישוב עשוי להיות בעל עומק אקספוננציאלי – עומק העץ הוא כזמן החישוב של המכונה, וכבר ראינו כי למכונה בעלת חסם סיבוכיות זכרון עשוי להיות זמן ריצה אקספוננציאלי (שכן מספר הקונפיגורציות האפשריות שלה לפני שהיא תתחיל לחזור על עצמה הוא אקספוננציאלי). היתרון שעלינו לנצל הוא בכך ש**גודל** כל קונפיגורציה הוא עדיין פולינומי.

השינוי הראשון שנבצע הוא להפסיק לדבר על עץ החישוב של המכונה ובמקום זאת נדבר על **גרף הקונפיגורצות** של M על w. זהו גרף מכוון שצמתיו הם הקונפיגורציות האפשריות בריצת M על w ויש קשת מהקונפיגורציה A לקונפיגורציה B אם המכונה עוברת מ־A ל־B בצעד בודד. הגרף דומה מאוד לעץ החישוב, כשההבדל היחיד הוא שהגרף הוא "חסר זכרון" – כל קונפיגורציה בו מיוצגת פעם אחת בדיוק, בלי תלות בדרכים השונות שבהן ניתן להגיע אליה.

לצורך פשטות נניח כי יש ל־M רק קונפיגורציה מקבלת יחידה – למשל, $[q_{acc}, 1, \mathtt{b}^*]$, כלומר זו שבה הסרט ריק והראש הקורא בתחילתו (תמיד ניתן לשנות את M כך שתקבל את אותה שפה אבל זו תהיה הקונפיגורציה המקבלת היחידה שלה). אם כן, הצטמצמנו לבעיית חיפוש בגרפים: אנו רוצים לדעת אם קיים מסלול מצומת s לצומת t (כאשר s היא הקונפיגורציה ההתחלתית של M ו־t הקונפיגורציה המקבלת) ואנו רוצים לעשות זאת באופן שיהיה חסכוני בזכרון.

אנו נראה כי קיים אלגוריתם אשר בהינתן גרף מכוון $G = (V, E)$ וצמתים $s, t \in V$ בודק האם קיים מסלול מ־s אל t ב־$\log^2 |V|$ זכרון. אם ניישם אלגוריתם זה לבעיה שלנו ינבע מכך שאנו מסוגלים לפתור את בעיית הישיגות בגרף הקונפיגורציות תוך שימוש בזכרון

$$O\left(\log^2\left(|Q_M| \cdot |\Gamma|^{p(|w|)} \cdot p(|w|)\right)\right) \quad = \quad O\left(\log^2(|Q_M|) + p^2(|w|) \cdot \log^2(|\Gamma|) + \log^2(p(|w|))\right) = O\left(p^2(|w|)\right)$$

וזה מסיים את ההוכחה כי זה מראה שניתן, תוך ניפוח ריבועי של סיבוכיות הזכרון (שמשאיר את סיבוכיות הזכרון פולינומית), לעבור ממכונה אי דטרמיניסטית למכונה דטרמיניסטית.

פתרון בעיית הישיגות בגרף

אם כן, ניתן לשכוח מתורת החישוביות; עברנו לבעיה באלגוריתמים על גרפים. ההבדל המהותי בינה לבין אלגוריתמים "רגילים" לחיפוש בגרף דוגמת DFS ו־BFS היא שאנו מעוניינים לחסוך ב**זכרון**, בעוד שבדרך כלל המטרה היא חסכון בזמן ריצה.

האלגוריתם לחיפוש בגרף הוא רקורסיבי; נניח כי הגרף G נתון לנו כ"קופסה שחורה" (אנחנו יודעים לייצר צמתים ב־V באופן סדרתי ובהינתן v, u אנו יודעים לבדוק האם $v \to u$ – נסמן זאת $E(u, v)$), ונגדיר פונקציה $\mathrm{Reach}(a, b, d)$ שמחזירה "כן" אם ורק אם קיים מסלול מאורך לכל היותר d מ־a אל b. פתרון בעיית הישיגות בגרף מתקבל מחישוב $\mathrm{Reach}(s, t, |V|)$.

הרעיון ב־Reach הוא פשוט: או שהמסלול מ־a אל b הוא טריוויאלי (מאורך 0 או 1), או שיש בהכרח "צומת ביניים" v שנמצא באמצע המסלול מ־a אל b. לכן כל מה שצריך לבדוק הוא האם, לכל $v \in V$, קיים מסלול מאורך $\frac{d}{2}$ מ־a אל v, ומסלול מאורך $\frac{d}{2}$ מ־v אל b. אם קיים כזה – סיימנו. אחרת, נבדוק את v הבא. זהו חישוב בזבזני מאוד מבחינת זמן, אך מכיוון שעבור ערכים שונים של v אנו יכולים למחזר את המקום שבו אנו משתמשים, זהו חישוב חסכוני מבחינת זכרון.

Reach(a, b, d):

1. if $a == b$ return True
2. if $d == 1$ return $E(a, b)$
3. for $v \in V$:
   (a) if Reach $(a, v, \lceil d/2 \rceil)$ and Reach $(v, b, \lfloor d/2 \rfloor)$ then return True
4. return False

סיבוכיות הזכרון של האלגוריתם בהרצתו עם $d = |V|$ היא כדלהלן. בשלבים 1 ו־2 אנו משתמשים בבירור ב־$O(\log|V|)$ זכרון (לא צריך יותר מכך כדי לייצג את צמתי הגרף ולבדוק קיום קשת ביניהם, כפי שראינו קודם). שלב 3 מתרחש $O(|V|)$ פעמים, אבל בכל פעם אנחנו יכולים להשתמש באותו הזכרון שהשתמשנו בו קודם, כך שקיום הלולאה חסר חשיבות עבורנו. באופן דומה, למרות שאנו קוראים פעמיים לרקורסיה, אפשר למחזר את כל הזכרון שנוצל בקריאה הראשונה לרקורסיה גם בקריאה השנייה לרקורסיה. לכן כל מה שעלינו לבדוק הוא את עומק עץ הרקורסיה; סיבוכיות הזכרון הכוללת שלנו היא עומק זה כפול סיבוכיות הזכרון של כל צומת – שכפי שראינו, היא $O(\log|V|)$ (הסיבוכיות של ייצוג a, b, v וביצוע השוואות ביניהם).

מכיוון שבכל איטרציה של האלגוריתם אנו מקטינים את d בחצי, ומתחילים עם $d = |V|$, הרי שעומק עץ הרקורסיה הוא $\log|V|$. לכן, סיבוכיות הרקורסיה הכוללת היא $O(\log|V| \cdot \log|V|) = O\left(\log^2|V|\right)$, כנדרש.