הרצאה 6 – יום ראשון 10.8

סיבוכיות

בחלק 1 של הקורס – חישוביות, השאלה הייתה רק "האם אפשר לחשב את זה", ובחינתנו כל המודלים שמסוגלים לחשב את אותו דבר הם שקולים. בחלק 2 של הקורס – סיבוכיות – השאלה היא: איזה משאבים האלגוריתם שלנו דורש? זמן, מקום זיכרון?

סיבוכיות זמן: זמן הריצה, כפונקציה של אורך הקלט: $t := t(n)$, כאשר $n := |x|$. אם x הוא מספר בייצוג בינארי, אז $n = O(\log_2 x)$.

נשקול את המקרה הגרוע או המקרה הממוצע.

נצטרך לשאול לפי מה מודדים זמן? כי זמן בפועל תלוי במחשב שרצים עליו.

המודל החישובי שמקובל לעבוד איתו הוא מכונת טיורינג. *Church-Turing* אומר לנו שזה

אחנו עובדים עם ההנחה שאורך הקלט ידוע לנו.

במ"ט, נמדוד את הזמן לפי מספר הצעדים שהמכונה מבצעת בהינתן קלט x.

אורך הקלט נמדד לפי **מספר הביטים** שדרוש כדי לקודד את הקלט. לכן נוח להתייחס לקלט בתור מחרוזת בינארית.

יש מודלים חישוביים שלא שקולים ל-*DTM* או *NTM*:

מודל אקראי – *Randomized TM*. יש אלמנט של הסתברות בריצה. יש להם מחלקות זמן ריצה:

- *RP – Randomized Polynomial*. שפה L תהיה ב-*RP* אם קיים לה *RTM* כלשהו M שמקיים:
  - אם $x \in L$, אז M מקבל בהסתברות לפחות $1/2$.
  - אם $x \notin L$, אז M מקבל בהסתברות 0 (תמיד דוחה).
- *coRP*. כמו *RP* אבל הפוך:
  - אם $x \notin L$, אז M מקבל בהסתברות לכל היותר $1/2$.
  - אם $x \in L$, אז M מקבל בהסתברות 1.
- *ZPP – Zero Probabilistic Polynomial*. זמן הריצה פולינומי בתוחלת, ואין טעויות. מתקיים: $ZPP = RP \cap coRP$.
- *BPP – Bounded-error Probabilistic Polynomial*. הסתברות לכל היותר חצי לטעות בשני המקרים.

באופן כללי מתקיים:

$$\mathrm{ZPP} = \mathrm{RP} \cap \mathrm{coRP},\ \ \mathrm{RP,coRP} \subseteq \mathrm{BPP} \subseteq \mathrm{P}^{\mathrm{NP}}$$

מודל אורקל – *Oracle TM*. מודל שיש בו אלמנט של קופסה שחורה שיכול לתת פתרון לבעיה בזמן מיידי.

זה מ"ט רגיל בתוספת סרט שאילתות, ואורקל A כלשהו. בכל שלב בריצה, ה-*TM* יכול לכתוב מחרוזת x על סרט השאילתה, ולקבל תשובה מיידית (בצעד אחד) האם $x \in A$.

הסימון:

$$\mathrm{P}^{\mathrm{A}}, \mathrm{NP}^{\mathrm{A}}, \mathrm{coNP}^{\mathrm{A}} \dots$$

משמעותו "מחלקה X עם אורקל עבור השפה A". אז לדוגמה, NP^{SAT} זה מחלקת השפות שמוכרעות ע"י *TM* אי-דטרמיניסטי אם יש אורקל לבעיית *SAT* (ספיקות של נוסחאות בוליאניות). מכיוון ש-*SAT* היא *NPC*, מתקיים בעצם ש- $NP^{SAT} = NP$.

מעגלים בוליאניים – *Circuits*. מודל שמזכיר מעגלים לוגיים במחשב. כל קלט הוא בגודל קבוע n, יש שערים לוגיים *AND, OR, NOT, XOR* וכו, יש חיבורים (בכיוון אחד) בין שערים, ויש פלט. הפלט לא חייב להיות בגודל n. לדוגמה, לפעמים הוא יהיה ביט יחיד – 0 או 1, לקבל או לדחות. כל מעגל בעצם מחשב פונקציה בוליאנית: $f: \{0,1\}^n \to \{0,1\}^n$.

הסיבוכיות נמדדת ע"י **גודל** (מספר השערים) **ועומק** (המסלול הארוך ביותר מהקלט לפלט). מחלקות סיבוכיות:

- *P/poly* – שפות שמוכרעות ע"י מעגלים בגודל פולינומי בגודל הקלט.
- *NC (Nick's Class)* – שפות שמוכרעות ע"י מעגלים בעומק פולי-לוגריתמי (*polylogarithmic*) – כלומר $O\left(\log^k n\right)$.
- *AC* – כמו *NC* אבל מותר שערי *AND, OR* עם כניסות מרובות.

הוכחה אינטראקטיבית – *Interactive Proof*. מודל שבו החישוב מתואר ע"י דיאלוג בין שני משתתפים:

- ה"מוכיח" (*Prover*) הוא בעל כוח בלתי מוגבל. הוא רוצה לשכנע את הצד השני שטענה כלשהי נכונה.
- ה"מוודא" (*Verifier*) הוא בעל כוח מוגבל. הוא לא מאמין למוכיח.

המוודא שואל את המוכיח שאלות לגבי הטענה, והמוכיח עונה. המודל צריך לקיים:

אם הטענה נכונה, אז המוודא משתכנע בהסתברות לפחות $2/3$. אם הטענה לא נכונה, אז ההסתברות שהמוודא ישתכנע שהיא נכונה היא לכל היותר $1/3$.

מבחינת סיבוכיות זמן ומקום, נחלק את הבעיות למחלקות – לפי האלגוריתם הכי טוב שקיים עבורם. באופן כללי, נתעלם מקבועים וביטויים קטנים.

זמן פולינומי

נאמר שפונקציה $f(n)$ היא פולינומית אם $f(n) = O(n^C)$ עבור קבוע כלשהו C.

אז פונקציה $f(n)$ היא **לא** פולינומית אם $n^C = o\big(f(n)\big)$ עבור קבוע כלשהו C.

נאמר ש- $f(n) = O\big(g(n)\big)$ אם קיימים קבועים k, n_0 כך שלכל $n > n_0$, מתקיים $f(n) \le k \cdot g(n)$. הפונקציה g היא חסם אסימפטוטי עליון של f.

נאמר ש- $f(n) = o\big(g(n)\big)$ אם:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

דוגמה

כמה זמן דרוש כדי לבדוק אם קלט שייך לשפה: $L_1 \coloneqq \{0^{n/2}1^{n/2} : n \in \mathbb{N}\}$, עם *1-tape DTM*?

ניזכר באלגוריתם שראינו בהרצאה 1:

האלגוריתם מסמן את ה-0 הראשון שלא מסומן, ואז מחפש את ה-1 הראשון שלא מסומן, עד שמגיעים לסוף (או שרואים משהו בלתי צפוי).

עבור קלט באורך *n*, נבצע לכל היותר $n/2$ מעברים כי בכל מעבר מסמנים שני תווים או שדוחים.

כל מעבר הוא *n* צעדים, ובמקרה הגרוע (מבחינת זמן) נעבור על כל הקלט. סה"כ $n \cdot (n/2) = n^2/2$.

זמן ריצה דטרמיניסטי

יהי *DTM* כלשהו *M*, ופונקציה $t\colon \mathbb{N} \to \mathbb{N}$. נאמר שזמן הריצה של *M* הוא $t(n)$ אם לכל קלט באורך *n*, מספר צעדי הריצה של *M* הוא לכל היותר $t(n)$.

המחלקה $\mathrm{DTIME}\big(t(n)\big)$ מכילה את כל השפות שמוכרעות ע"י *DTM* (כולל מכונות עם מספר סרטים) בזמן $O\big(t(n)\big)$.

$$\mathrm{DTIME}\big(t(n)\big) \coloneqq \big\{L(M) : M \text{ is a DTM with runtime } O\big(t(n)\big)\big\}$$

אז בדוגמה הקודמת, $L_1 \in \mathrm{DTIME}(n^2)$.

המשך דוגמה

אלגוריתם מהיר יותר עבור $L_1 \coloneqq \{0^{n/2}1^{n/2} : n \in \mathbb{N}\}$:

1. נעבור על הקלט ונבדוק שאין 0 אחרי 1. — $n+1$ צעדים.
2. נרוץ עד שנדחה או נקבל: — $\log_2 n$ חזרות.
   a. נעבור על הסרט, ונוודא שמספר האפסים ואחדות שלא מסומנים הוא זוגי. — $n+1$ צעדים.
   b. נעבור על הסרט, ונסמן כל 0 שני וכל 1 שני. — $n+1$ צעדים.
   c. נעבור על הסרט, אם הכל מסומן, נקבל. — $n+1$ צעדים.

סה"כ, $O(n \cdot \log n)$.

כל פעם, אנחנו בעצם מורידים חצי מהאפסים וחצי מהאחדות. אם היה מספר שווה של אפסים ואחדות, אז הם תמיד יהיו שווים ותמיד נקבל מספר זוגי.

אם הם שונים, אז אחד קטן מהשני ויגיע ל-1 לפני השני. בשלב הזה, אם יש (בה"כ) אפס יחיד ויש מספר זוגי של אחדות, אז נדחה. אם במקרה יש גם מספר אי זוגי של אחדות (חייב להיות 3 או יותר) אז בגלל שיש רק 1 יחיד, נסמן אותו, ונסמן חצי (מעוגל כלפי מעלה) מהאחדות, אז יישארו מספר אי זוגי של אחדות.

אלגוריתם עוד יותר מהיר, עם שני סרטים:

1. נעבור על הסרט עד ה-1 הראשון. נעתיק את כל האחדות לסרט 2. אם רואים 0, נדחה. $n+1$ צעדים.
2. נחזור ל-0 האחרון בסרט 1. נלך אחורה על שני הסרטים. אם נגיע להתחלה באותו צעד, נקבל. אחרת, נדחה. $n+3$ צעדים.

סה"כ $O(n)$.

המחלקה P

שפה תהיה שייכת למחלקה P אם קיים DTM בזמן ריצה פולינומי שמכריע אותה.

$$L \in P \Leftrightarrow \exists c \in \mathbb{N}: \exists M \in \mathrm{DTIME}\big(O(n^c)\big): L = L(M)$$

כלומר המחלקה P מוגדרת:

$$P \coloneqq \bigcup_{c=0}^{\infty} \mathrm{DTIME}\big(O(n^c)\big)$$

למה אנחנו רוצים זמן פולינומי? סיבה ראשונה, כי הוא הרבה יותר קצר מזמן אקספוננציאלי:

$$\lim_{n\to\infty} \frac{c^n}{n^c} = \infty, \qquad poly(n) = o(\exp(n))$$

	Size=n	n	n^2	n^3	n^5	2^n	3^n
0	10	.00001 sec	.0001 sec	.001 sec	.1 sec	.001 sec	.059 sec
1	20	.00002 sec	.0004 sec	.008 sec	3.2 sec	1.0 sec	58 min
2	30	.00003 sec	.0009 sec	.027 sec	24.3 sec	17.9 min	6.5 years
3	40	.00004 sec	.0016 sec	.064 sec	1.7 min	12.7 days	3855 cents.
4	50	.00005 sec	.0025 sec	.125 sec	5.2 min	35.7 years	2*10^8 cents.
5	60	.00006 sec	.0036 sec	.216 sec	13.0 min	366 centuries	1.3*10^13 cents.

סיבה שנייה, כי אפשר לעשות רדוקציה בין כל המודלים הדטרמיניסטיים בזמן פולינומי. וכי $poly(n) \cdot poly(n) = poly(n)$.

סמלוץ של DTM עם k סרטים על DTM עם סרט אחד דורש זמן $O(n^2)$.

$$DFA \xrightarrow{O(1)} 1\text{-}tape\ DTM \xrightarrow{O(n^2)} k\text{-}tape\ DTM$$

מכונת "גישה אקראית" – *RAM (Random Access Machine)*

שקול בכוח למ"ט – כל מה שאפשר לחשב על אחד אפשר לחשב על השני. יש להם:

- *register* (אוגר) בגודל סופי, שכל תא יכול להחזיק מספר שלם כלשהו. בהתחלה ה-*register* מחזיק את הקלט בתא הראשון.
- זיכרון – *memory*. בהתחלה הוא ריק, ואפשר להגדיל אותו תוך כדי הקלט. הזיכרון מוגדר כפונקציה: $M: D \to \mathbb{Z}$, כאשר D היא קבוצה סופית של מספרים שלמים שהם הכתובות זיכרון הקיימות.
- תכנית – *program*. סדרת פקודות δ.
- מצב – *state*. כל מצב מתואר ע"י: $(R[0], R[1], \dots R[k], c, M)$. המצב הנוכחי של ה-*register*, מיקום הפקודה הנוכחית c, והזיכרון.

המצב ההתחלתי הוא $(x, 0, \dots, 0, 1, \emptyset)$, כאשר x זה הקלט.

- לפעמים אומרים "אוגר" (ביחיד) ואז יש בו הרבה תאים שכל אחד יכול להחזיק מספר. לפעמים אומרים "אוגרים" (ברבים) ואז כל אוגר יכול להחזיק מספר. זה רק עניין סמנטי.
- נקודה חשובה: האוגרים לא מוסיפים כוח חישובי. באותה מידה אפשר להשתמש רק בזיכרון (פשוט להגדיר כמה מספרים בזיכרון שהם בפועל אוגרים). זה רק בשביל הנוחות של ההסברה (שיהיה דומה יותר למחשב במציאות) וכדי שיהיה אפשר לגשת אליהם בזמן $O(1)$ במקום דרך אוסף פקודות של גישה לזיכרון.

נוכל להגדיר את ה-*RAM* כך: $\mathcal{R} \coloneqq (k, \delta)$, כאשר: $k \in \mathbb{N}$ זה מספר האוגרים, ו- δ זו תכנית סופית – אוסף פקודות:

- טעינה – *load*. $R[i] \leftarrow M\big[R[j]\big]$. טעינה מהזיכרון (במיקום ששמור באוגר) לאוגר.
- שמירה – *store*. $M[R[i]] \leftarrow R[j]$. שמירה של משהו מאוגר בזיכרון (במיקום ששמור באוגר).
- חיסור – *subtract*. $R[i] \leftarrow R[i] - R[j]$. מחסירים מאוגר i את המספר ששמור באוגר j.
- הזזה ימינה – *shift right*. $R[i] \leftarrow R[i] \text{ div } 2$. חלוקה ב-2 ועיגול לכיוון 0.
- טעינת 1 – *load constant 1*. $R[i] \leftarrow 1$. טעינת המספר 1 לתוך אוגר.
- קפיצה בתנאי – *conditional jump*. $\text{if } R[i] \geq 0, \text{ go-to } L$. אם הערך באוגר i גדול או שווה 0, לעבור לפקודה כלשהי L.
- קבלה – *accept*.
- דחייה – *reject*.

כאשר $i, j \in [k]$ הם כתובות, ו- $L \in [|\delta|]$ זו פקודה.

עם אוסף הפקודות הזה, אפשר בזמן קבוע לבצע: טעינת הקבוע 0, חיבור, קפיצה בלי תנאי, קפיצה אם שווה / לא שווה / קטן / קטן שווה / גדול, חישוב מודולו 2, הזזה שמאלה. וכך אפשר לחשב כל חישוב שאפשר לחשב באופן תיאורטי. באופן כללי, אנחנו רוצים כמה שיותר כלים (כדי שנוכל לעשות כמה שיותר דברים) אבל במודלים חישוביים, אנחנו גם רוצים שהם יהיו פשוטים כדי שנוכל להוכיח עליהם דברים.

שפה של *RAM*

שפה של RAM כלשהו $\mathcal{R}$ מוגדרת להיות אוסף הקידודים ש-$\mathcal{R}$ מקבל. לדוגמה: תהי $y := (y_1, \dots y_n) \in \Sigma^*$, כאשר $|\Sigma| = m$. כל $\sigma \in \Sigma$ יקבל מספר בין 0 ל- $m-1$, כלומר כל $y_i \in \{0,1,\dots,m-1\}$. הקידוד של y יהיה שרשור של $y_1, \dots y_n$: $y := y_1 m^n + y_2 m^{n-1} + \cdots + y_{n-1} m + y_n$.

סמלוץ של *RAM* על *1-tape-DTM*

1. נקבל את הקידוד של x, ואחריו k פעמים #.
2. האוגרים שיש בהם ערך שונה מאפס (יש כמות סופית) מיוצגים ע"י רצף של זוגות – [כתובת, ערך] כאשר אחרי כל זוג יש #.
3. כל פקודה מיוצגת ע"י אוסף מצבים.
   a. לדוגמה, $R[2] \leftarrow R[2] \text{ div } 2$. המ"ט הולך לתא 2, קורא את הערך v, וכותב $v \text{ div } 2$.
   b. עוד דוגמה, $R[i] \leftarrow M\big[R[j]\big]$. המ"ט הולך לתא j, קורא את הערך, הולך לתא i, וכותב אותו.
   c. כל פקודה אפשר לסמלץ באופן דומה.
4. הסיבוכיות של סמלוץ t צעדים של *RAM* היא $O(t^3 \cdot (n+t)^2)$.

סמלוץ של *1-tape-DTM* על *RAM*

נצטרך רק שני אוגרים. $R[1]$ שומר את תוכן הסרט משמאל לראש, $R[2]$ שומר את תוכן הסרט מימין לראש (שמור בסדר הפוך – *big endian* – כדי שתהיה גישה נוחה יותר ל-*LSB*). כדי לסמלץ צעד של ה-*TM*, נקרא את ה-*LSB* של $R[2]$. נכתוב במקומו מה שהפקודה של ה-*TM* אומרת. נסמלץ את התזוזה של הראש – נעביר את ה-*LSB* מ-$R[1]$ ל-$R[2]$ (תזוזה שמאלה) או מ-$R[2]$ ל-$R[1]$ (תזוזה ימינה). סמלוץ של t צעדים דורש זמן $O(t)$.

בסה"כ, המודלים שקולים והזמן למעבר ביניהם הוא פולינומי. כלומר כשנגדיר מחלקות (כמו P וכו') זה לא משנה באיזה מודל משתמשים.

בחזרה למחלקה *P*

ניזכר שאמרנו שהמחלקה P היא קבוצת השפות שיש עבורן *DTM* שמכריע אותן. בפועל, זה לא משנה אם זה *DTM* או *RAM*, כי המודלים שקולים. המחלקה P הפכה להיות מחלקת כל הבעיות שאפשר לפתור ע"י מחשב בזמן פולינומי.

מסלול בגרף מכוון – שייך ל-*P*

$$\text{PATH} := \{\langle G, s, t\rangle : G \textit{ is a directed graph}, \quad \{s,t\} \subseteq V(G), \quad \textit{there exists a path } s \rightsquigarrow t \textit{ in } G\}$$

נוכיח: נבנה אלגוריתם דטרמיניסטי פולינומי.

1. נסמן את קודקוד ההתחלה s.
2. נבצע כל עוד מסמנים קודקוד:
   a. לכל צלע (a,b), אם a מסומנת ו-b לא, נסמן את b.
3. אם t מסומנת נקבל, אחרת נדחה.

סיבוכיות: עבור גרף עם n קודקודים ו-m צלעות, גודל הקלט לינארי ב-m. במקרה הגרוע נעבור על כל הצלעות, שזה $O(n^2)$.

מספרים זרים – *co-primes*

$$co\text{-}primes \coloneqq \{\langle x, y\rangle : \gcd(x, y) = 1\}$$

נשתמש באלגוריתם האוקלידי: עבור קלט x, y:

1. נבצע עד ש- $y = 0$:
   a. נקבע $x \coloneqq x \bmod y$.
   b. נחליף את $x \leftrightarrow y$.
2. אם $x = 1$ נחזיר 1, אחרת 0.

הסיבוכיות של הלולאה היא לוגריתמית בערך הקלט, שזה פולינומי בגודל הקלט. החישוב של *mod* הוא פולינומי.

מחלקת *NP*

Nondeterministic Polynomial Time – מחלקת השפות שיש אלגוריתם שמכריע אותן, אם מאפשרים אי-דטרמיניזם. פורמלית:

יהי *NTM* כלשהו $\mathcal{M}$, ופונקציה $t: \mathbb{N} \to \mathbb{N}$. מחלקת השפות $\mathrm{NTIME}\big(t(n)\big)$ היא מחלקת כל השפות שמוכרעות ע"י *NTM* כלשהו שרץ בזמן $O\big(t(n)\big)$.

$$\mathrm{NTIME}\big(t(n)\big) \coloneqq \big\{L(\mathcal{M}): NTM\ \mathcal{M}\ has\ a\ runtime\ O\big(t(n)\big)\big\}$$

$$\mathrm{NP} \coloneqq \bigcup_{c=0}^{\infty} \mathrm{NTIME}\big(O(n^c)\big)$$

עץ הריצה של *NTM* על קלט *x* – אם יש מסלול שמגיע למצב מקבל, נמצא אותו. אם אין אף מסלול כזה (כולם דוחים או לא עוצרים) אז לא נקבל. תיאורטית אפשר למדל את זה על מחשב, אבל זה דורש מעקב אחרי כל הקונפיגורציות וזה אקספוננציאלי (וכנראה חורג מהיכולות של כל מחשב אמיתי, עבור קלט ארוך).

<u>טענה</u>:

$$\mathrm{NTIME}\big(f(n)\big) \subseteq \mathrm{DTIME}\big(2^{O(f(n))}\big)$$

<u>הוכחה</u>: יהי *k* המספר המקסימלי של אפשרויות שיש ל-*NTM* בצעד כלשהו. אז אחרי *m* צעדים, יש לכל היותר k^m קונפיגורציות.

אם ה-*NTM* מקבל את *L* תוך $O\big(f(n)\big)$ צעדים, אז ה-*DTM* השקול מקבל את *L* תוך $k^{O(f(n))}$ צעדים.

עבור $k \geq 2$, קיים קבוע $c \geq 1$ כלשהו כך ש:

$$k^{O(f(n))} = 2^{c \cdot O(f(n))} = 2^{O(f(n))}$$

כלומר הזמן של *NTM* הוא **לפחות** לוגריתמי בזמן של ה-*DTM* שמסמלץ אותו.

הגדרה שקולה ל-*NP* – "עדים"

נחשוב על *NTM* בתור *DTM* שיש לו סרט נוסף – "עד". הקלט הוא (x, w), כאשר *w* הוא ה"עד". ואז מתקיים:

- אם $x \in L$, אז קיים *w* כך ש- $\mathcal{M}(x, w) = 1$.
- אם $x \notin L$, אז לכל *w* מתקיים $\mathcal{M}(x, w) = 0$.

העד הוא הוראות שאומרות ל-*DTM* איזה בחירה לבצע בכל צעד. אז במקום עץ, מחשבים רק מסלול אחד.

אלגוריתם וידוא

מוודא עבור שפה *L* הוא אלגוריתם (נגיד *DTM*) כלשהו *V* כאשר:

$$L \coloneqq \{x \mid V\ accepts\ (x, w)\ for\ some\ string\ w\} = \{x \mid \exists w :\ V(x, w) = 1\}$$

אם *V* רץ בזמן פולינומי ב- $|x|$, אז *L* ניתנת לווידוא בזמן פולינומי. הגדרה שקולה ל-*NP* – מחלקת השפות שניתנות לווידוא בזמן פולינומי.

הוכחת השקילות:

כיוון ראשון – $\mathrm{NTM} \Longrightarrow \mathrm{Verifier}$. בהינתן *NTM*, אנחנו יודעים כבר לבנות *DTM* שקול. ל-*DTM* הזה יש את רצף הבחירות ה"נכון". הרצף הזה הוא העד. האורך של העד הוא פולינומי בגודל הקלט, וזה מספר הצעדים של ה-*DTM*.

כיוון שני – $\text{NTM} \Longrightarrow \text{Verifier}$. בהינתן שפה L שיש לה אלגוריתם וידוא V, נבנה את $\mathcal{M}$, NTM ש"מנחש" את העד הנכון, w, באורך לכל היותר $|x|^k$. ונריץ את $V(x,w)$. אם V מקבל אז נקבל, אחרת נדחה. $\mathcal{M}$ הוא NTM פולינומי שמכריע את L.

דוגמאות לשפות NP

מסלול המילטוני בגרף מכוון:

$$\text{HAM-PATH} \coloneqq \{\langle G,s,t\rangle \mid there\ is\ a\ hamiltonian\ path\ from\ s\ to\ t\ in\ G\}$$

כדי להוכיח שהשפה ב-NP, נבנה NTM שמכריע את השפה. עבור קלט $\langle G,s,t\rangle$, כאשר $|V(G)|=m, |E(G)|\le m^2$:

1. נסמן את s.
2. נבצע:
   a. באופן אי-דטרמיניסטי, נבחר צלע (a,b) מהקודקוד האחרון a שמסומן לקודקוד b שלא מסומן, ונסמן את b. אם אין צלע כזו, נדחה.
   b. אם $b=t$, נצא מהלולאה.
3. נעבור על הקודקודים. אם כולם מסומנים נקבל, אחרת נדחה.

מספר הצעדים הוא פולינומי במספר הקודקודים.

קליקה:

$$\text{CLIQUE} \coloneqq \{\langle G,k\rangle \mid there\ is\ a\ k\text{-}clique\ in\ G\} \in \text{NP}$$

נבנה מוודא V: בהינתן $\langle\langle G,k\rangle,w\rangle$, כאשר w זה קבוצת קודקודים שהם אמורים להיות הקליקה. נבדוק אם w תת גרף של G על k קודקודים. אם לא, נדחה. אחרת, נעבור על כל הזוגות של קודקודים ב-w. אם באחד הזוגות אין צלע – נדחה. אחרת, נקבל.

Subset – Sum:

$$\text{Subset-Sum} \coloneqq \{\langle S,t\rangle \mid S\subseteq \mathbb{N},\qquad \exists S'\subseteq S : \Sigma S = t\}$$

נבנה מוודא: $\langle\langle S,t\rangle,w\rangle$, נבדוק האם $w\subseteq S$ והאם $\Sigma w = t$. אם כן, נקבל. אחרת, נדחה.

מחלקת $coNP$

נשים לב ששפה המשלימה של שפה ב-NP לא בהכרח ב-NP בעצמה. לדוגמה שפת $HAM\text{-}PATH$ – לא ידוע לנו על אלגוריתם פולינומי, אפילו עם אי-דטרמיניזם. זה לא אומר שהשפה בהכרח לא ב-NP. רק שאנחנו לא יודעים. המחלקה $coNP$ היא מחלקת השפות שהמשלימה שלהן ב-NP.

$$L\in \text{NP} \Leftrightarrow \bar{L}\in \text{coNP}$$

אז נכון לעכשיו, יכול להיות ש- $\text{NP}\ne\text{coNP}$. זו הדעה הרווחת, אבל זה לא מוכח. אם $\text{P}=\text{NP}$ אז $\text{NP}=\text{coNP}$, אבל ההיפך לא בהכרח נכון.

אנחנו כן יודעים ש- $\text{NP}\cap\text{coNP}\ne\emptyset$. לדוגמה, $\text{FACTOR}\in\text{NP}\cap\text{coNP}$:

$$\text{FACTOR} \coloneqq \{\langle n,k\rangle \mid n\ has\ a\ nontrival\ divisor < k\}$$

כלומר, הזוגות n,k כך שקיים מספר ראשוני $p<k,\ \ p\notin\{1,n\}$ שמחלק את n ללא שארית.

- השפה ב-NP כי יש אלגוריתם וידוא (עד – המספר p, הבדיקה האם הוא ראשוני ומחלק את n היא פולינומית).
- היא גם ב-$coNP$ כי השפה המשלימה ב-NP: שפת הזוגות האלה כך שכל המחלקים הראשוניים של n גדולים או שווים ל-k. העד – פירוק לראשוניים של n. (לכל n יש פירוק יחיד לראשוניים). הבדיקה שכל המספרים בפירוק ראשוניים וגדולים מ-k היא פולינומית.

P מול NP

השאלה הגדולה – האם $\text{P}=\text{NP}$. הדעה הרווחת היא שלא. אם כן, אז $\text{P}=\text{NP}=\text{coNP}$, וזה יהרוס, לדוגמה, את רוב שיטות ההצפנה שלנו. (רק בתיאוריה. כי בפועל, זמן פולינומי של n^{20000} גרוע יותר מזמן אקספוננציאלי של 2^n עד בערך 350,000. וגם אם מוכח שבעיה ב-P, זה לא עוזר אם לא מצאנו אלגוריתם פרקטי). אם באמת $\text{P}\ne\text{NP}$, אז זה אומר שיש בעיות שניתנות לווידוא אך לא לפתירה בזמן סביר.