

סיבוכיות

© ארזים

21 במרץ 2017

בשיעור שעבר דיברנו על פתרון מערכת משוואות לינאריות, וראינו שקילות לחישוב מטריצה הופכית, ושזה שקול לכפל מטריצות. ראינו אלגוריתם מהיר לכפל מטריצות - אלגוריתם שטראסן. לאחר מכן דיברנו על כפל פולינומים וכפל מספרים גדולים, וראינו את אלגוריתם קרצובה.

1 בעיית DFT (Discrete Fourier Transform)

יהי $\omega \in \mathbb{C}$ שורש יחידה פרימיטיבי מסדר n . אזי $1, \omega, \omega^2, \dots, \omega^{n-1}$ כולם שונים, וכן $\omega^n = 1$. למשל $\omega = e^{\frac{2\pi i}{n}}$. בהינתן פולינום

$$f(x) = \sum_{i=0}^{n-1} a_i x^i$$

ממעלה לכל היותר n , מטרתנו היא לחשב את הערכים $f(1), f(\omega), \dots, f(\omega^{n-1})$. לשם הפשטות נניח כי כפל של מספרים מרוכבים ניתן לבצע ביחידת זמן 1, ושאינן בעיות של אחסון מספרים בזיכרון.

הערה 1.1 אם נגדיר את המטריצה

$$V = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ 1 & \omega^2 & (\omega^2)^2 & \dots & (\omega^2)^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & (\omega^{n-1})^2 & \dots & (\omega^{n-1})^{n-1} \end{pmatrix}$$

כלומר $V_{i,j} = \omega^{ij}$, אזי

$$V \cdot \begin{pmatrix} a_0 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} f(1) \\ f(\omega) \\ \vdots \\ f(\omega^{n-1}) \end{pmatrix}$$

1.1 אלגוריתם Fast Fourier Transform (FFT) של Cooley-Tukey

רעיון שוב בשיטת הפרד ומשול.

$$\begin{aligned} f(x) &= \sum_{i=0}^{n-1} a_i x^i = \sum_{i=0}^{\frac{n}{2}-1} a_{2i} x^{2i} + \sum_{i=0}^{\frac{n}{2}-1} a_{2i+1} x^{2i+1} = \\ &= \sum_{i=0}^{\frac{n}{2}-1} a_{2i} x^{2i} + x \sum_{i=0}^{\frac{n}{2}-1} a_{2i+1} x^{2i} \end{aligned}$$

כעת נסמן

$$\begin{aligned} f_0(z) &= \sum_{i=0}^{\frac{n}{2}-1} a_{2i} z^i \\ f_1(z) &= \sum_{i=0}^{\frac{n}{2}-1} a_{2i+1} z^i \end{aligned}$$

וכעת נקבל

$$f(x) = f_0(x^2) + x f_1(x^2)$$

נשים לב כעת כי

$$(\omega^i)^2 = \omega^{2i} = \omega^{2i} \cdot 1 = \omega^{2i} \cdot \omega^n = \omega^{2i+n} = (\omega^{i+\frac{n}{2}})^2$$

מכאן נקבל כי

$$f_0((\omega^i)^2) = f_0((\omega^{i+\frac{n}{2}})^2)$$

ותוצאה זהה תקפה עבור f_1 .

האלגוריתם נחשב את $f_0(1), f_0(\omega^2), f_0((\omega^2)^2), \dots, f_0((\omega^{\frac{n}{2}-1})^2)$ כמו כן, נחשב גם את $f_1(1), f_1(\omega^2), f_1((\omega^2)^2), \dots, f_1((\omega^{\frac{n}{2}-1})^2)$. נשים לב כי ω^2 שורש יחידה פרימיטיבי מסדר $\frac{n}{2}$, כלומר עלינו לחשב DFT מסדר $\frac{n}{2}$ עבור f_0, f_1 . כעת, לכל $0 \leq i \leq \frac{n}{2} - 1$ נוציא

$$f(\omega^i) = f_0(\omega^{2i}) + \omega^i f_1(\omega^{2i})$$

ולכל $0 \leq j \leq \frac{n}{2}$, כאשר $i = j + \frac{n}{2}$, נסמן $\frac{n}{2} \leq i \leq n-1$ ונוציא

$$f(\omega^i) = f_0(\omega^{2j}) + \omega^i f_1(\omega^{2j})$$

סיבוכיות זמן ריצה:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$$T(n) = O(n \log n)$$

1.2 שימוש של FFT כדי לחשב כפל פולינומים

בהינתן שני פולינומים

$$f(x) = \sum_{i=0}^{n-1} a_i x^i$$

$$g(x) = \sum_{i=0}^{n-1} b_i x^i$$

רוצים לחשב את

$$h(x) = f(x)g(x)$$

נחשב DFT מסדר $2n$ עבור f, g (נניח כי n חזקת 2, כמו שעשינו במובלע בכל האלגוריתם קודם). כעת, יש לנו את הערכים

$$f(1), g(1), f(\omega), g(\omega), \dots, f(\omega^{2n-1}), g(\omega^{2n-1})$$

כאשר ω שורש יחידה פרימיטיבי מסדר $2n$. כעת נחשב את

$$h(\omega^i) = f(\omega^i)g(\omega^i)$$

לכל $0 \leq i \leq 2n-1$. כעת, נבצע שוב FFT, לוקטור

$$\begin{pmatrix} h(1) \\ \vdots \\ h(\omega^{2n-1}) \end{pmatrix}$$

כלומר, נחשב את

$$\frac{1}{2n} (\omega^{-ij})_{i,j} \begin{pmatrix} h(1) \\ \vdots \\ h(\omega^{2n-1}) \end{pmatrix}$$

מדוע? אם נסמן

$$W(\omega) = (\omega^{ij})_{i,j}$$

כאשר ω שורש יחידה פרימיטיבי מסדר m , אזי

$$(W(\omega^{-1})W(\omega))_{i,j} = \sum_{k=0}^{m-1} \omega^{k(j-i)} = \begin{cases} m & j = i \\ 0 & j \neq i \end{cases}$$

נסביר את המקרה השני:

$$\sum_{k=0}^{m-1} \omega^{k(j-i)} = \frac{1 - \omega^{(j-i)m}}{1 - \omega^{j-i}} = \frac{0}{\underbrace{1 - \omega^{j-i}}_{\neq 0}} = 0$$

זמן ריצה של כפל פולינומים:

$$S(n) = 3T(2n) + O(n) = O(n \log n)$$

נכונות נשים לב שתחילה חישבנו את $h(1), h(\omega), \dots, h(\omega^{2n-1})$ שזה שקול בעצם לחישוב הבא: אם

$$h(x) = \sum_{i=0}^{2n-1} c_i x^i$$

אזי

$$W(\omega) \cdot \begin{pmatrix} c_0 \\ \vdots \\ c_{2n-1} \end{pmatrix} = \begin{pmatrix} h(1) \\ \vdots \\ h(\omega^{2n-1}) \end{pmatrix}$$

בשלב השני חישבנו את

$$\underbrace{\frac{1}{2n} W(\omega^{-1})}_{W(\omega)^{-1}} \begin{pmatrix} h(1) \\ \vdots \\ h(\omega^{2n-1}) \end{pmatrix} = \begin{pmatrix} c_0 \\ \vdots \\ c_{2n-1} \end{pmatrix}$$

1.3 כפל מספרים בני n ספרות

ניקח שני מספרים a, b ונביט בייצוג הבינארי שלהם:

$$a = \sum_{i=0}^{n-1} a_i 2^i$$

$$b = \sum_{i=0}^{n-1} b_i 2^i$$

נביט בפולינומים

$$f = \sum a_i x^i$$

$$g = \sum b_i x^i$$

ונכפול אותם. נקבל

$$\sum_{k=0}^{2n-2} x^k \underbrace{\left(\sum_{i=0}^k a_i b_{k-i} \right)}_{O(\log n) \text{ bits}}$$

אנחנו צטרך לסכום את כל המפרים הללו. בסך הכל נצטרך לחבר בערך $2n$ מספרים, שכל אחד מהם לוקח $O(\log n)$, כלומר סך הזמן הוא $O(n \log n)$.

1.4 אבסטרקציה

הקלט של DFT הוא ווקטור

$$\begin{pmatrix} a_0 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

ואנחנו כופלים אותו במטריצה $W(\omega)$. נשים לב שבכל שלב האלגוריתם מחשב פונקציה לינארית של הקלט.

ניתן לתאר את האלגוריתם באופן הבא: נכתוב במטריצה את מקדמי הצירוף הלינארי שחושב בכל שלב בחישוב. n השורות הראשונות יהיו מהצורה $(\delta_{i,j})_j$, כאשר

$$\delta_{i,j} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

כעת, כל שורה מעתה והלאה תהיה צירוף לינארי של שתיים מהשורות הקודמות. ניזכר כי במהלך האלגוריתם חישבנו את

$$f_0(\omega^{2^i}) + \omega^i f_1(\omega^{2^i})$$

כאשר $f_0(\omega^{2i}), f_1(\omega^{2i})$ חושבו קודם. כלומר חישבנו צירוף לינארי של הפונקציות הלינאריות (בקלט) $f_0(\omega^{2i}), f_1(\omega^{2i})$ עם המקדמים $1, \omega^i$. מודל החישוב הוא כזה שבו בכל שלב מחשבים צירוף לינארי של שתי שורות קודמות במטריצה. בסוף החישוב רוצים שאותן n פונקציות לינאריות בקלט שרצינו לחשב יופיעו בשורות כלשהן של המטריצה. המטריצה שמתארת את החישוב מתחילה במטריצה I_n , ולאחר מכן, בשורה t מופיע הצירוף הלינארי שחושב בשלב t .

טענה 1.2 כל אלגוריתם לינארי לחישוב DFT, המשתמש במקדמים עם ערך מוחלט לכל היותר 1, חייב לבצע $\Omega(n \log n)$ פעולות.

הוכחה: נגדיר פונקציית התקדמות. בהתחלה, הערך שלה יהיה 1, ובסוף הערך שלה יהיה $n^{\frac{n}{2}}$. בכל שלב הערך שלה גדל לכל היותר פי 2. מכאן ינבע כי יש לפחות

$$\log(n^{\frac{n}{2}}) = \frac{n}{2} \log n$$

פעולות.

בהנתן מטריצה A ממימד $m \times n$, כאשר $m \geq n$, נגדיר

$$\phi(A) = \max_{\substack{B \text{ is a square} \\ \text{minor of } A \text{ of} \\ \text{dimension } n}} |\det(B)|$$

נטען כי $\phi(I) = 1$, וכן כי $\phi(W(\omega)) = n^{\frac{n}{2}}$. מדוע? כי

$$\overline{W} \cdot W = n \cdot \text{Id}$$

משום שלקחנו $|\omega| = 1$, אזי $\omega^{-1} = \overline{\omega}$. לכן

$$|\det(\overline{W} \cdot W)| = n^n$$

$$|\det(W)|^2 = n^n$$

$$|\det(W)| = n^{\frac{n}{2}}$$

בתחילת האלגוריתם ערך הפונקציה הוא 1. במהלך החישוב, מתישהו נחשב את כל השורות של W , ולכן W קיימת כמינור של המטריצה בסוף, כלומר ערך הפונקציה הוא לפחות $n^{\frac{n}{2}}$.

נרצה להראות כעת כי ההתקדמות בכל שלב חסומה. נסמן את המטריצה שהתקבלה אחרי k שלבים בתור A_k . במטריצה A_{k+1} הוספנו שורה מהצורה $\alpha R_i + \beta R_j$, כאשר $|\alpha|, |\beta| \leq 1$. אם השורה הזו לא מופיעה בתת המטריצה של A_{k+1} בעלת דטרמיננטה מקסימלית, אזי בבירור $\phi(A_{k+1}) = \phi(A_k)$. לכן נניח שהיא כן מופיעה בה. נניח שיחד

איתה הופיעו השורות $R_{i_1}, \dots, R_{i_{n-1}}$ אזי

$$\begin{aligned} \phi(A_{k+1}) &= \left| \det \begin{pmatrix} R_{i_1} \\ \vdots \\ R_{i_{n-1}} \\ \alpha R_i + \beta R_j \end{pmatrix} \right| = \left| \alpha \det \begin{pmatrix} R_{i_1} \\ \vdots \\ R_{i_{n-1}} \\ R_i \end{pmatrix} + \beta \det \begin{pmatrix} R_{i_1} \\ \vdots \\ R_{i_{n-1}} \\ R_j \end{pmatrix} \right| \leq \\ &\leq |\alpha| \left| \det \begin{pmatrix} R_{i_1} \\ \vdots \\ R_{i_{n-1}} \\ R_i \end{pmatrix} \right| + |\beta| \left| \det \begin{pmatrix} R_{i_1} \\ \vdots \\ R_{i_{n-1}} \\ R_j \end{pmatrix} \right| \leq \\ &\leq \left| \det \begin{pmatrix} R_{i_1} \\ \vdots \\ R_{i_{n-1}} \\ R_i \end{pmatrix} \right| + \left| \det \begin{pmatrix} R_{i_1} \\ \vdots \\ R_{i_{n-1}} \\ R_j \end{pmatrix} \right| \leq \phi(A_k) + \phi(A_k) = 2\phi(A_k) \end{aligned}$$

square minor of A_k of dimension n
square minor of A_k of dimension n

לכן התלחנו לעשות מה שרצינו. נשים לב שאם היינו מניחים $|\alpha|, |\beta| < c$, היינו מקבלים את אותה תוצאה, בשינוי קבוע כלשהו. ■

2 קושי חישובי

נרצה לראות איך קושי חישובי יכול לעזור לנו - בפרטת להכין פרוטוקול הצפנה. בהינתן קלט x , מחושבת הצפנה $f(x)$, שבעזרת מפתח אפשר לפענח אותה ולקבל חזרה את x . נקודת המוצא של כל פרוטוקול קריפטוגרפי היא שיש פונקציה שנורא קשה לחשב. במקרה שתיארנו, זו f^{-1} .

2.1 הצפנות מפתח פומבי

מישהו, למשל Ebay, מפרסם מפתח פומבי e , ומפתח פרטי d . המצפין מחשב פונרציה הידועה לכולם, $Enc(x, e)$ - כלומר מצפין את x . המפענח (Ebay) מחשב פונקציה

$$Dec(Enc(x, e), d) = x$$

ההנחה המובלעת היא שללא ידיעת d , קשה לחשב את x מתוך e , $Enc(x, e)$. בעצם, ההנחה היא שמידעת e קשה לחשב את d .

2.1.1 פרוטוקול RSA

בוחרים שני מספרים ראשוניים גדולים p, q (בני n ספרות). מחשבים $N = p \cdot q$. כמו כן, בוחרים מספר e הזר למכפלה $(p-1)(q-1)$ - כלומר מספר e שלא חולק אף גורם עם

המכפלה. מחשבים d כך שמתקיים

$$e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$$

המפתח הפומבי הוא (N, e) , והמפתח הפרטי הוא d .
 Enc : בהנתן מספר $1 \leq x \leq N-1$ שזר לאותו N , מחשבים

$$Enc(x, (N, e)) = x^e \pmod{N}$$

Dec : בהנתן $y = x^e \pmod{N}$ נחשב את

$$Dec(y, d) = y^d \pmod{N}$$

הפענוח עובד שכן

$$\begin{aligned} y^d \pmod{N} &= x^{ed} \pmod{N} = x^{1+k(p-1)(q-1)} \pmod{N} \\ &= x \cdot (x^k)^{(p-1)(q-1)} \pmod{N} = x \pmod{N} \end{aligned}$$

השוויון האחרון מתקיים שכן קבוצת המספרים הזרים למספר N היא חבורה כפלית בגודל $(p-1)(q-1)$. לכן יש לנו איבר בחזקת סדר החבורה, שראינו בשיעורי הבית שזה האיבר הנייטרלי - 1.

הערה 2.1 אם p, q ידועים, אז אפשר לחשב את d . מדוע? משפט השאריות הסיני אומר שבהנתן השאריות של מספר מודולו p ומודולו q אפשר לחשב את השארית שלו מודולו pq . נראה איך, ואיך עושים את זה, בשבוע הבא, וגם נראה את הפרוטוקול של רבין, שמסתמך על קושי אחר - ואף שקול לו.