סיבוכיות חישוב ואלגוריתמים של כפל

עבודת גמר לתואר שני בהוראת המדעים

תכנית ויצמן רוטשילד

מנחה: פרופ' אירית דינור

מגישה: עירית חמצני

1. מבוא

המושג "חישוב" קיים בתרבות האנושית כבר אלפי שנים. החישובים נעשו בהקשרים שונים החל בניהול חשבונות וכלה באסטרונומיה. כשמדברים על חישוב מתכוונים תהליך שבו מקבלים נתונים ומבצעים עליהם סדרה סופית של פעולות שבסופן מתקבלת התוצאה. בזמנים עבורו ה'מחשבים' היו האנשים - בעלי המקצוע שקיבלו נתונים , ידעו לבצע את תהליכי החישוב ולהפיק בסוף התהליך תוצאה עבור נתונים אלה. לאורך ההיסטוריה הומצאו מכונות חישוב מכניות שתכליתן הייתה לבצע חישובים אריתמטיים. מכונות אלו הקלו על התהליך, מנעו טעויות וקיצרו את זמן החישוב. אחת הדוגמאות למכונת חישוב היא מכונת החישוב שהגה פסקל. זו הייתה מכונת חישוב מכנית שנועדה לעזור לאביו של פסקל בעבודתו כגובה מיסים. קפלר, ליבניץ ואחרים שנדרשו לבצע חשובים מסובכים במסגרת עבודתם, תכננו גם הם מכונות חישוב שנועדו לבצע חישובים ספציפיים. באמצע המאה ב-19 הציע צארלס בבאג' מכונת חישוב מכנית שהכילה רעיון חדש. עד אותו יום המכונות היו מסוגלות לפעול על נתונים בדרך יחידה. דרך זו נקבעה ע"י המבנה הפיזי שלהן (החומרה). במידה ורצו שהמכונה תבצע פעולה אחרת היה צורך לשנות את המבנה שלה. באבג' הגה מכונה מתוחכמת יותר, אשר מקבלת הוראות פעולה ופועלת את פעולתה על הנתונים בהתאם להוראות אלה (מכונה שניתנת לתכנות). יש לציין כי בבאג' מעולם לא הצליח לבנות ולממש את המכונות שתכנן. יש האומרים כי הוא הקדים את זמנו ורואים בו אבי המחשב המודרני.

כידוע, מאז אותם ימים ובמיוחד מתחילת המאה העשרים התפתח תחום מדעי המחשב בצעדי ענק. תרומה משמעותית להתפתחות התחום נעשתה ע"י צבאות שונים. למשל, כבר במלחמת העולם הראשונה השתמשו הגרמנים והאמריקאים במכונות חישוב לירי ארטילרי ולהצפנה. במקביל להתפתחות היישומים השונים למנגנוני חישוב אוטומטיים, החל להתגבש בסיס התיאורטי לתחום מדעי המחשב במיוחד מהמחצית הראשונה של המאה ה-20. בתקופה זו החלו לעסוק באלגוריתמים ובחישוביות והמושג 'חישוב' הוגדר בצורה מדויקת. אלן טורינג שנחשב לאבי המחשב המודרני, הגיע מעיסוק תיאורטי בתורתו של גדל ומשפט אי השלמות ובבעיית ההכרעה שהציג הילברט. מתוך כך הוא הגה מודל מתמטי של מחשב שלימים נקרא 'מכונת טורינג'. מכונה זו מהווה את היסודות למחשב המודרני. מכונת טורינג מבצעת חישובים בדרך פשוטה מאד. בהינתן בעיה מסוימת, המכונה מקבלת כקלט את הנתונים של הבעיה וכן את הקידוד של המכונה כלומר את הדרך (האלגוריתם) לפתרון הבעיה. המכונה מתקדמת בסדרה סופית של צעדים (ע"פ קידוד המכונה) המופעלים על הקלט ובסוף התהליך מפיקה את התוצאה - פתרון הבעיה. המחשבים של ימינו מבוססים על רעיון זה, החומרה שלהם פשוטה (יחסית) ותוכנות שונות מאפשרות פתרונות לבעיות השונות מאד זו מזו.

מכאן נפתחה הדרך לדיון תיאורטי ומעשי בחישוביות ולעיסוק בשאלה: 'האם בעיה מסוימת היא ברת חישוב', כלומר 'האם היא ניתנת לפתרון בעזרת מחשב או לא'. חשוב להדגיש כאן שהדיון הוא על הצורה הכללית של הבעיה ולא על מקרים פרטיים של הבעיה שלעיתים ניתנים לחישוב בניגוד לבעיה כללית. החוקרים מצאו שבעיות מסוימות הן לא ברות חישוב בצורה אינהרנטית, כלומר כל מחשב לא יצליח לפתור אותן בהינתן נתונים מסוימים. בהמשך לכך, בשנות השישים החלה להתפתח תורת הסיבוכיות שמרכזה השאלה עד כמה בעיה היא מסובכת ועד כמה אלגוריתם לבעיה מסובך או פותר אותה. תורה זו נמצאת במוקד עבודה זו.

תורת הסיבוכיות החישובית (Computational Complexity Theory) אם כן, היא ענף במדעי המחשב שבו ממיינים בעיות חישוביות על פי הקושי הפנימי שלהן תוך התייחסות והשוואת הבעיה למחלקות קושי של חישוב (Computational classes). בעיית חישוב היא בעיה שניתן

ומקובל לפתור אותה בעזרת מחשב, כלומר בעזרת מנגנון שמכיל צעדים מתמטיים (אלגוריתם). לבעיה מסוימת ייתכנו אלגוריתמים שונים שבעזרתם ניתן לפתור אותה. בעיה חישובית נחשבת קשה אם היא דורשת משאבים רבים בכל אלגוריתם שבו נבחר. התיאוריה מציגה מודלים מתמטיים של חישוב שמאפשרים ללמוד את הבעיות ולהעריך את כמות המשאבים החישוביים הנדרשת כדי לפתור אותם. שאלת הסיבוכיות היא שאלה של משאבים. שואלים על יעילות החישוב ומעריכים את כמות המשאבים הנדרשת לפתרון הבעיה (משאבי מחשב). מתוך המשאבים השונים הנדרשים לפעולת המחשב, המשאב העיקרי שנבחן הוא זמן ריצה. משאבים נוספים שעוסקים בהם הם כמות הזיכרון, מספר המעבדים (בדיון בעיבוד מקבילי) משאבי תקשורת ומספר שערים לוגיים. אחד התפקידים של התיאוריה של סיבוכיות חישובית הוא לקבוע את מגבלות של המחשב ולהסיק מה בר חישוב ומה לא. כחלק מהמחקר מתבוננים באלגוריתמים שונים לפתרון בעיה מסוימת ובודקים את יעילות החישוב של כל אחד מהם, ומנסים להעריך את כמות המשאבים הנדרשת כדי לפתור את הבעיה בעזרתם. הדיון באלגוריתמים לפתרון חשוב גם בדיון על החסם התחתון של סיבוכיות. השאלה היא האם מצאנו את האלגוריתם הכי פחות מסובך לבעיה או שיש פתרון פשוט יותר.

בעבודה זו אדון בנושא הסיבוכיות החישובית. הדיון בנושא נעשה תוך עיסוק בהערכת סיבוכיות של אלגוריתמים שונים לכפל של שני שלמים. בעבודה מודגם השימוש בכלים שונים למידול ולחישוב סיבוכיות.

2. מדידת סיבוכיות ושימוש בסימון אסימפטוטמי (asymptotic Notation)

כדי לדון בסיבוכיות של בעיה או של אלגוריתם יש להעריך את כמות המשאבים הנדרשת לפתרון. כמות המשאבים תלויה בגודל של הנתונים, ולכן הסיבוכיות תימדד ע"י בחינת האופן בו עולה כמות הפעולות הבסיסיות הנדרשת כאשר מגדילים את הinput. מגדירים פונקציה T(n) (n טבעי), ששווה למספר הפעולות הבסיסיות המכסימלי שהאלגוריתם דורש ל-Input באורך n. לעיתים הפונקציה T(n) תלויה בחלקים מסדר נמוך. השימוש בסימונים הבאים עוזר להתעלם מהפרטים ולהתרכז בתמונה הכללית.

Big O notation: הגדרה של חסם עליון:

מסמנים:

$$f(n) = O(g(n))$$

משמעות הסימון היא שקיים $c > 0$ ממשי ומספר טבעי $n_0 > 0$ כך שמתקיים

$0 \le f(n) \le cg(n)$, עבור כל $n > n_0$

הערה: הסימון מציין קטן או שווה, הסימון הוא אסימטרי, ולא ניתן להחליף בין אגפי המשוואה.

דרך נוספת להגדרה: Set Definition

$$O\big(g(n)\big) = \{f(n) : \exists\ 0 < c \in R, 0 < n_0 \in N, : 0 \le f(n) \le cg(n)\ \forall\ n \ge n_0\}$$

כאן ברורה יותר הא-סימטריה.

מקובל להשתמש כמקרו : לדוגמא :

$$f(n) = n^3 + O(n^2)$$

או:

$$n^2 + O(n) = O(n^2)$$

באופן דומה מגדירים חסם תחתון, שמסומן ב- $\Omega(g(n))$.

חסם 'שווה' חיתוך בין חסם עליון לחסם תחתון ומסומן ב- Θ

$$\Theta\big(g(n)\big) = O(g(n)) \cap \Omega(g(n))$$

(חסמים עם >,< , קטן ממש, גדול ממש. מסומנים ב-ω, /o)

המטרה : למצוא חסם עליון לאלגוריתמים שונים. החסם העליון מציין את היעילות של האלגוריתם במקרה הגרוע ביותר , כלומר במקרה בו האלגוריתם מקבל נתונים הדורשים את מקסימום הזמן כדי לפתור אותו.

3. אלגוריתמים של כפל

3.1 בדיון בסיבוכיות אלגוריתמית נתבונן בסיבוכיות של אלגוריתמים שונים לכפל (מדובר על כפל של שני מספרים שלמים). כפל שימושי באפליקציות חישוביות שונות כמו קריפטוגרפיה ועיבוד אותות. כאשר כופלים שני מספרים בני מספר רב של ספרות יש השפעה רבה לבחירת האלגוריתם על זמן הריצה של התוכנית.

<u>כפל ע"י חיבור חוזר:</u>

דרך הפשוטה ביותר לביצוע כפל של שני שלמים היא ע"י ביצוע על ידי חיבור חוזר (Repetitive Addition). אם נתונים שני שלמים a,b כל אחד בעל n ספרות , בייצוג עשרוני. שני המספרים מיצגים כל אחד מספר שערכו בין 10^{n-1} ל-1 $- 10^n$.

אם מסכמים : x+x+x+.....+x , נקבל את התוצאה :x*y לאחר y-1 פעולות חיבור. אלגוריתם פשטני זה דורש y-1 פעולות חיבור.

לדוגמא:

$$123 * 456 = 123 + 123 + \cdots + 123$$

עבור כפל של שני מספרים בגודל $10^n - 1$ נקבל כ-10^n פעולות חיבור של שני מספרים בני n ספרות. לכן סיבוכיות האלגוריתם היא:

$$T(n) = O(n10^n)$$

כפל באלגוריתם בית ספר:

האלגוריתם הבא הוא האלגוריתם המוכר מימיי בית הספר היסודי ונקרא גם כפל ארוך. שיטת זו מבוססת על העובדה של ייצוג מספרים ושיטת הספירה שלנו היא על בסיס מיקום בבסיס 10. ולכן למשל: $a_2a_1a_0 = a_2 10^2 + a_1 10^1 + a_0 10^0$

באלגוריתם זה אנחנו מבצעים כפלים של כל אחת מהספרות של a בכל אחת מהספרות של b לדוגמא:

```
 * 123
   456
  ----
   738
  615
 492
 -----
 56088
```

בהשוואה לשיטת החיבור החוזר, בדוגמא זו נדרשות 9 פעולות כפל של חד ספרתי בחד ספרתי ועוד מספר קטן יותר של פעולות חיבור חד ספרתיות. לעומת 3*455 פעולות חיבור באלגוריתם שהוצע קודם לכן.
כאשר בונים פונקציה שמתארת את מספר הפעולות הבסיסיות בתלות בגודל המספרים מקבלים כי באלגוריתם זה יהיו לכל היותר $2n^2$ פעולות בסיסיות.
פונקציית הסיבוכיות היא: $\mathrm{T}(n) = O(n^2)$

ניתן לראות שבאלגוריתם הראשון יש גידול הוא אקספוננציאלי של מספר הפעולות, ואילו באלגוריתם השני הגידול הוא פולינומיאלי. יש הבדל גדול מאד בין שני האלגוריתמים. אפילו למספר בן 11 ספרות, מחשבון כיס שישתמש באלגוריתם בית ספר יהיה יעיל יותר ממחשב-על שיחשב את הכפל ע"י חיבור חוזר. השוואה זו מדגימה את העובדה שאם מסתכלים על המשאבים הנדרשים לחישוב של כפל של שני מספרים בני n ספרות אז עבור n מעל לסף (לא גבוה) לאלגוריתם הנבחר תהיה השפעה רבה יותר על המשאבים מאשר לטכנולוגיה שבה שמשמשת למימוש האלגוריתם.

אלגוריתם בית ספר הוא אלגוריתם שתואם למבנה העשרוני של מספרים. במהלך השנים נמצאו אלגוריתמים פחות אינטואיטיביים לכפל ועם זאת יעילים יותר. אחד מהם הוא אלגוריתם קרצובה שמתואר בסעיף הבא.

3.2 אלגוריתם קרצובה (Kartsuba Algorithm)

זהו אלגוריתם לכפל של שני שלמים שהתגלה ע"י Anatoly Karatsuba בשנת 1960, ופורסם ב-1962. אלגוריתם החדש חוסך מספר רב של פעולות כפל, ומקטין את הסיבוכיות בצורה משמעותית כפי שמוסבר בהמשך.

בהינתן שני מספרים x,y כל אחד באורך n ספרות , מחלקים כל אחד לשני מספרים באורך $\frac{n}{2}$ ספרות, ואז ניתן להציגם בצורה הבאה:

$$x = x_1 b^{\frac{n}{2}} + x_0$$

$$y = y_1 b^{\frac{n}{2}} + y_0$$

(b – הבסיס בו מוצגים המספרים)

מגדירים: $m = \frac{n}{2}$ ואז:

$$xy = (x_1 b^m + x_0)(y_1 b^m + y_0) =$$

$$= x_1 y_1 b^{2m} + (x_1 y_0 + x_0 y_1) b^m + x_0 y_0$$

התקבלו <u>ארבע</u> פעולות כפל של שני מספרים באורך $\frac{n}{2}$ שמחליפים את הכפל של שני מספרים באורך n. כפי שמוצג בהמשך, ניתן להסתפק ב-3 כפלים כאלה. בפתרון זה נוסף מספר קטן של פעולות חיבור וחיסור אשר סדר הגודל שלהן קטן יותר ולכן השפעתן על פונקציית הסיבוכיות ניתנת להזנחה.

מעבר לשלש פעולות כפל של שני מספרים באורך $\frac{n}{2}$:

מגדירים:

$(1)\ z_2 = x_1 y_1$

$(2)\ z_1 = x_1 y_0 + x_0 y_1$

$(3)\ z_0 = x_0 y_0$

באמצעות (1) ו- (3) וכפל נוסף, אפשר להגדיר את z_1 תוך שימוש בכפל אחד וב- z_0, z_2 שחושבו קודם:

$$z_1 = x_1 y_1 + x_1 y_0 + x_0 y_1 + x_0 y_0 - x_1 y_1 - x_0 y_0$$

$$z_1 = (x_1 + x_0)(y_1 + y_0) - z_2 - z_0$$

לאחר חישוב 3 מרכיבי הכפל מחברים אותם לאחר הזזה (shift) של z_1 ב- $\frac{n}{2}$ ספרות שמאלה הזזה של z_2 ב-n ספרות שמאלה.

פונקציית הסיבוכיות של אלגוריתם קרצובה:

$$(4)\quad T(n) = 3T\left(\frac{n}{2}\right) + O(n) = O\left(n^{\log_2 3}\right) + O(n) = O(n^{1.58496})$$

המשאבים הנדרשים לכפל שני מספרים בני n ספרות כוללים שלש פעולות כפל של שני מספרים בני n/2 ספרות. ובנוסף, מספר קטן יותר של פעולות חיבור וחיסור עם סיבוכיות מסדר n. הרעיון העומד בבסיס אלגוריתם קרצובה הוא לחלק את המספרים שוב את המספרים לשניים ולהשתמש בשלשה כפלים במקום ארבעה. חוזרים על הפעולה עד שמגיעים לשלב בו כבר לא כדאי להשתמש באלגוריתם. הסיבוכיות המתקבלת מוצגת במשוואה (4). יש שיפור יחסית לסיבוכיות של אלגוריתם בית ספר. בסעיף הבא אציג את דרך החישוב של תוצאה זו.

3.2 חישוב סיבוכיות/יעילות של אלגוריתם בשיטת Divide & Conquer (הפרד ומשול:)

הרעיון בבסיס שיטה נפוץ מאד ומופיע בתחומים שונים: בהינתן בעיה מסוימת, מחלקים אותה למספר בעיות קטנות יותר שיש להן את אותו מבנה. חוזרים על פעולת החלוקה עד אשר הבעיה מספיק קטנה וכדאי לפתור אותה בצורה ישירה. חישוב הסיבוכיות נעשה בצורה רקורסיבית וזאת ע"י חישוב הסיבוכיות של הבעיה הקטנה יותר. לבסוף מסכמים את הפתרונות לפתרון של הבעיה המקורית.

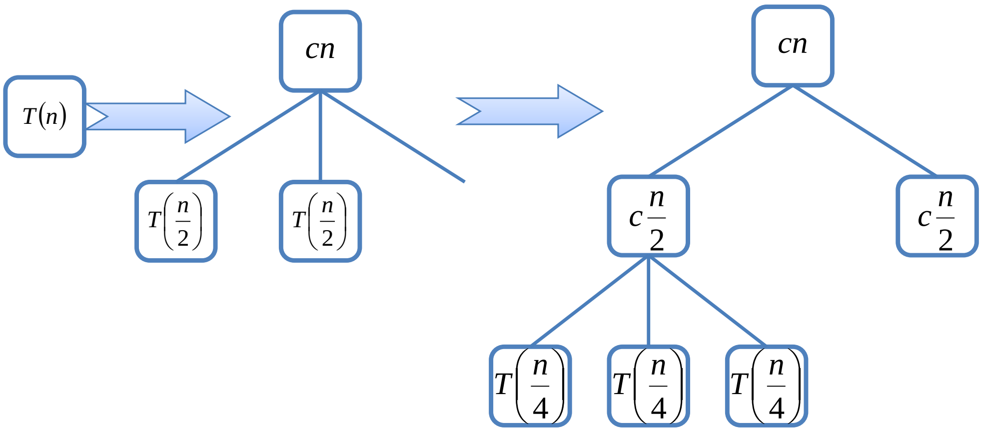
השלבים בפתרון בעיה בשיטת Divide & Conquer : 1. Divide: מחלקים את הבעיה למספר בעיות זהות לבעיה המקורית אך קטנות יותר בגודלן. 2. Conquer: פותרים את הבעיות הקטנות בצורה רקורסיבית, (שוב ושוב מחלקים את הבעיה לבעיות קטנות). אם הבעיה קטנה מספיק ,שאפשר לפתור אותה בצורה ישירה, פותרים. 3. Combine the solutions: משלבים את הפתרונות של כלל הבעיות כדי לקבל את הפתרון של הבעיה המקורית.

<u>פתרון רקורסיבי של פונקצית הסיבוכיות בעזרת עץ בינארי:</u>

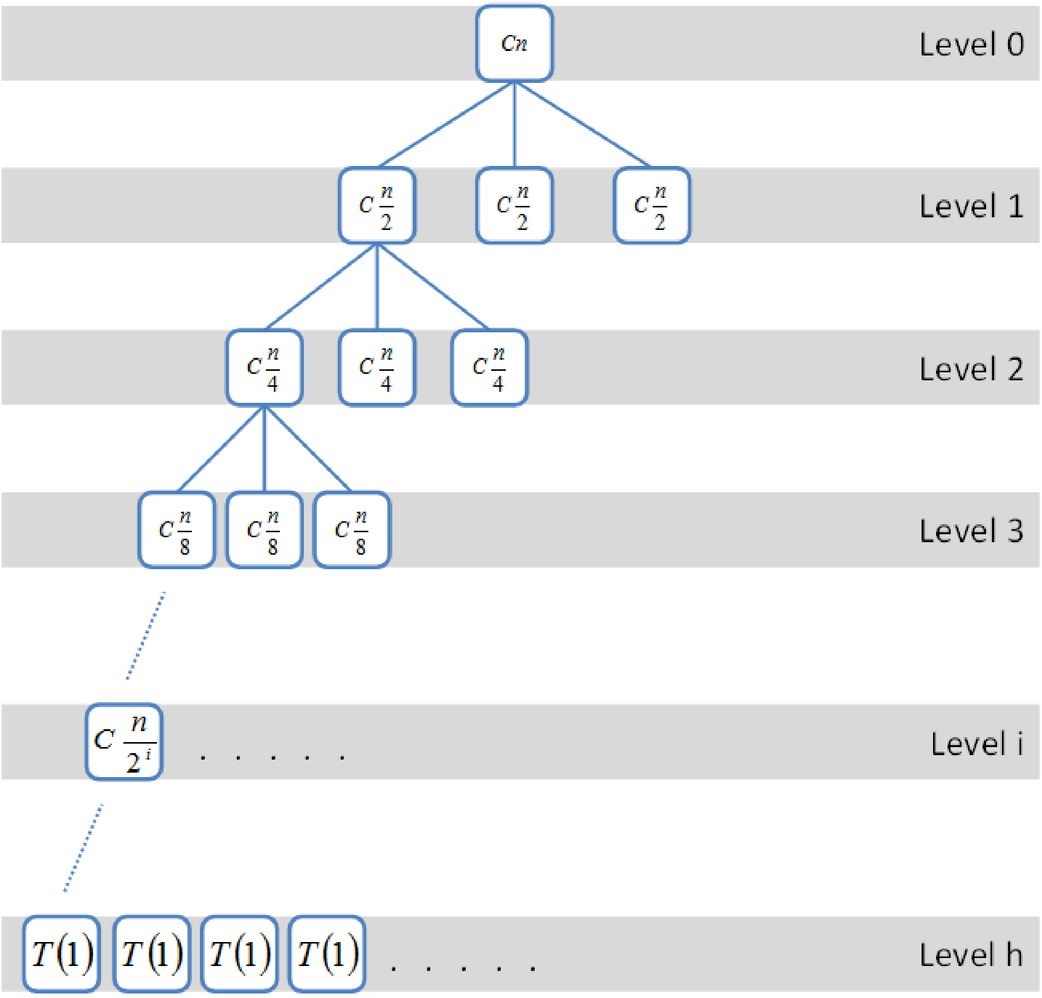
המשוואה (4) המוצגת בסעיף הקודם היא משוואה רקורסיבית ולכן כדי להעריך את היעילות של T(n), יש לחשב את T(n/2), וכדי לחשב את T(n/2) יש לחשב את T(n/4) וכן הלאה. התהליך נמשך עד אשר מגיעים למצב בו פונקציית היעילות אינה תלויה ב- n. וערכה קבוע. בשרטוט הבא מתואר עץ רקורסיה של פונקציית הסיבוכיות של אלגוריתם קרצובה (4).

שרטוט 1: תהליך פירוק המשוואה:

$$T(n)=3T\left(\frac{n}{2}\right)+O(n)=3T\left(\frac{n}{2}\right)+cn$$



שרטוט 2 : חישוב סיבוכיות :



$$T(n) = cn + \frac{3cn}{2} + \frac{9cn}{4} + \cdots . + 3^h T(1) = \sum_{i=0}^{h-1} \frac{3^i}{2^i} cn + 3^h T(1) =$$

$$T(n) = 3^{\log_2 n} T(1) + cn + cn \frac{\left(\frac{3}{2}\right)^{\log_2 n - 1} - 1}{\frac{3}{2} - 1}$$

$$= n^{\log_2 3} T(1) + cn + \frac{1.33cnn^{\log_2 3}}{n} - 2 =$$

$$O\left(n^{\log_2 3}\right) = O(n^{1.5849})$$

3.3 שימוש ב- FFT(Fast Furiere ransform) לביצוע כפל:

אלגוריתם נוסף שהוצג מספר שנים לאחר אלגוריתם קרצובה מבוסס התמרת פוריה הדיסקרטית.

כלים מתמטיים שמהוים בסיס לאלגוריתם FFT לכפל:

תיאורית האינטרפולציה של פולינומים:
בהינתן n נקודות במישור:

$$S = (x_0, y_0), (x_1, y_1), \ldots . . (x_{n-1}, y_{n-1})$$

כך שכל ה- x_i שונים

$$\{\exists !\ p(x_i) | p(x_i) = y_i\ \forall i\}$$

(כלומר : קיים פולינום יחיד $P(x)$ שעבורו מתקיים $p(x_i) = y_i$ לכל i)
מכאן שאם נתון פולינום ונחשב את ערכו ב-n נקודות שונות, אוסף הנקודות הוא ייצוג של הפולינום.

שורשי יחידה פרימיטיביים:
המספר ω הוא שורש יחידה n–י פרימיטיבי (primitive nth root of unity), אם עבור $n \geq 2$ הוא מקיים: 1. $\omega^n = 1$, ω הוא שורש n-י של 1. 2. המספרים $1, \omega, \omega^2, \ldots . , \omega^{n-1}$ שונים זה מזה.
מכאן שרוב שורשי היחידה הפרימיטיביים יהיו מספרים מרוכבים.
למספרים אלו מספר תכונות ייחודיות ואתייחס לשתי תכונות שימושיות לאלגוריתם FFT: רפלקטיביות ורדוקטיביות.

-רפלקטיביות (*The reflective property*) :
אם ω הוא שורש יחידה *n*-י פרימיטיבי, ו-$n \geq 2$ אז מתקיים:

$$\omega^{k+\frac{n}{2}} = -\omega^k$$

-רדוקטיביות - צמצום (*The reduction property*):
אם ω הוא שורש יחידה <u>*2n*-י</u> פרימיטיבי אז ω^2 הוא שורש יחידה <u>*n*-י</u> פרימיטיבי.
הוכחה:
2n החזקות של *ω* : $(1, \omega, \omega^2, \ldots, \omega^{2n-1})$ הן שונות אחת מהשנייה (*distinct*).
ולכן גם *n* החזקות של ω^2: $(1, \omega^2, (\omega^2)^2, \ldots, (\omega^2)^{n-1})$ שונות אחת מהשנייה.
משמעות תכונה זו היא שאם יש לנו *2n* שורשי יחידה עבור פולינום מדרגה *2n-1*, אז חצי משורשי היחידה הם גם שורשי יחידה עבור פולינום מדרגה *n-1*.

<u>התמרת פוריה דיסקרטית (The Discrete Fourier Transform)</u>: התמרת פוריה הדיסקרטית של פולינום מסדר n-1 היא ערכי הפולינום ב-n שורשי היחידה

<u>התמרת פוריה הפוכה דיסקרטית (The Inverse Discrete Fourier Transform)</u>: בהינתן וקטור $(y_0, y_1, \ldots, y_{n-1})$ שמייצג את ערכי הפולינום ב-n שורשי היחידה $(1, \omega, \omega^2, \ldots, \omega^{n-1})$, התמרה זו מחשבת את ערכי המקדמים של חזקות הפולינום בדרך הבאה:

$$a_i = \sum_{j=0}^{n-1} \frac{y_i \omega^{-ij}}{n}$$

אלגוריתם Cooley-Tuckey ל-FFT:

בשלב הראשון מחלקים את הפולינום, לשני פולינומים בדרגה שהיא חצי מהדרגה של הפולינום המקורי בדרך הבאה:

נתון פולינום: $p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$

מפרקים אותו לשני פולינומים מדרגה $\frac{n}{2} - 1$ בדרך הבאה :

$$p_{even}(x) = a_0 + a_2x + a_4x^2 + \cdots + a_{n-2}x^{\frac{n}{2}-1}$$

$$p_{odd}(x) = a_1 + a_3x + a_5x^2 + \cdots + a_{n-1}x^{\frac{n}{2}-1}$$

ואז:

$$p(x) = p_{even}(x^2) + xp_{odd}(x^2)$$

יש לחשב את p(x) עבור כל אחד מ- n שורשי היחידה הפרימיטיביים מסדר n $1, \omega, \omega^2, \ldots., \omega^{n-1}$. לחלופין, ניתן להשתמש באלגוריתם הנ"ל לחשב את ערכי p_{odd} ו- p_{even} עבור החזקות של ω^2: $(1, \omega^2, (\omega^2)^2, \ldots, (\omega^2)^{n-1})$. הסיבוכיות תהיה = $O\left(\frac{n^2}{2}\right)$ $O(n^2)$. אך אם נשתמש שוב בשיטת הפרד ומשול ונפעיל את האלגוריתם בצורה רקורסיבית על כל אחד מהפולינומים p_{odd} : ו- p_{even}, סיבוכיות החישוב תרד מאד.

בנוסף בגלל תכונת הרדוקטיביות והרפלקטיביות (reduction & reflective properties) יש צורך לחשב זאת רק עבור מחצית מערכי ω. ואז להשתמש בערכים שחושבו עבור p_{odd} : ו- p_{even} כדי לחשב גם את $p(\omega^k)$ וגם את $p(\omega^{\frac{n}{2}-k})$

$$p(\omega^k) = p_{even}(\omega^{2k}) + \omega^k(p_{even}(\omega^{2k})$$

$$p\left(\omega^{\frac{n}{2}-k}\right) = p_{even}(\omega^{2k}) - \omega^k(p_{even}(\omega^{2k}))$$

שימוש באלגוריתם לחישוב כפל של פולינומים:

נתונים שני מספרים עשרוניים כל אחד בן n ספרות

1. מניחים $n = 2^k$. אם לא מתקיים מרפדים באפסים את עד אשר מספר הספרות יהיה חזקה של 2.
2. מציגים את הספרות של כל מספר כווקטור מקדמים של פולינום (עבודה בבסיס 10)
3. מפעילים על כל אחד מהווקטורים את אלגוריתם cooley Thukey ל- FFT ומוצאים עבורו את ערכי הפולינום בכל שורשי היחידה הפרימיטיביים שלו. עבודה בסדר 2n. בגלל ההפעלה הרקורסיבית וחישוב שני שורשים בסבב אחד.
4. כופלים את הערכים שהתקבלו עבור כל אחד משורשי היחידה הפרימיטיביים מסדר 2n. ומקבלים וקטור באורך 2n, שמייצג את פולינום התוצאה.
5. מבצעים Inverse FFT בעזרת אלגוריתם Cooley-Thukey כדי לחזור להצגה של מקדמים.

הסיבוכיות :

$$T(n) = O(n\log(n)^2)$$

4. סיכום

בעבודה זו הצגתי הצצה לעולם הסיבוכיות החישובית, ניתן מבוא ורקע היסטורי לחישוביות ולמדעי המחשב. בהמשך, כדי להדגים את רעיון סיבוכיות החישוב נדונו אלגוריתמים שונים לביצוע כפל של שני מספרים שלמים , תוך התייחסות לסיבוכיות של כל אחד מהם. בדיון זה מוצגים הבדלים בסיבוכיות של אלגוריתמים שמבצעים למעשה את אותו חישוב. במשך השנים נמצאו אלגוריתמים טובים אף יותר מבחינת סיבוכיות. נשאלת השאלה האם קיים אלגוריתם טוב יותר מהאלגוריתם האחרון שנמצא, ומהי הסיבוכיות הטובה ביותר שאפשר להגיע אליה עבור כפל של שני שלמים. זו שאלה של חסם תחתון. ההנחה הרווחת היא שהחסם התחתון לסיבוכיות כפל היא $T(n) = O(nlogn)$. ואם זאת טרם למצאה ההוכחה להנחה זו. שאלת החסם התחתון לסיבוכיות של אלגוריתמי הכפל היא אחת השאלות הפתוחות של מדעי המחשב.

מקורות:

Arora, S., & Barak, B. Computational Complexity: A Modern Approach. (2009)' Prinston University Press

ויקיפדיה – ערכים: סיבוכיות, מדעי המחשב,Karatsoba Algorithm, multiplication Algorithms

יום הולדת 100 לאלן טורינג http://www.gadial.net/2012/06/23/alan_turing_is_100/

קורס בנושא אלגוריתמים
https://www.youtube.com/watch?v=JPyuH4qXLZ0&list=PL81A705FB7F988E7C&index=1

קורס MIT 5.046J/18410J

http://people.cs.uchicago.edu/~laci/HANDOUTS/karatsuba.pdf