קורס : קודים לתיקון שגיאות ושימושיהם במדעי המחשב
מרצה : ד"ר קלים יפרמנקו
סמסטר סתיו תשפ"א
10/01/2021

הרצאה 22:

Local codes

מבוא :

קודים מקומיים, הם קודים לתיקון שגיאות המצוידים באלגוריתמים מקומיים. קודים אלה הם בעלי היסטוריה ארוכה של אינטראקציה עם תיאוריית הסיבוכיות, עם יישומים בולטים להוכחות אינטראקטיביות, הוכחות ניתנות לבדיקה באופן הסתברותי (PCP) , hardness amplification , אחזור מידע פרטי(private information retrieval) ותאוריית הסיבוכיות האלגברית .

מה היה לנו עד עכשיו?

$C : \{0,1\}^k \to \{0,1\}^n$ כך ש-C הוא קוד לתיקון שגיאות, והתרחיש שדיברנו עליו הוא כזה :

אליס שולחת הודעה m לבוב, אז ראשית היא מקודדת את ההודעה c(m) ולאחר מכן היא שולחת את הקידוד הנ"ל לבוב. קידוד זה מגיע אליו בתוספת רעש : $c(m) + e$. ואז בוב מפענח את ההודעה שקיבל.

היום נדבר על המצב שבו בוב לא מעוניין לקרוא את כל ההודעה שקיבל מאליס כי הוא מתעניין רק בביט אחד מתוך ההודעה שהוא מקבל מאליס, למשל אליס היא המרצה והיא מפרסמת ציונים, אז בוב מתעניין רק בציון שלו ואין לו עניין לדעת את כל הציונים. קיימים הרבה מקרים כאלו, שבהם מעניין אותנו רק דברים לוקליים.

איך עושים את זה?

נציג כמה סוגים של קודים לוקליים :

1. locally recoverable codes -קודים הניתנים לשחזור מקומי:

קוד על אלפבית סופי נקרא בר שחזור (קוד LRC) אם כל ביט בקידוד הוא פונקציה של מספר קטן (לכל היותר r) ביטים אחרים של מילת הקוד. באופן רשמי יותר, (n, k, r) קוד הניתן לשחזור מקומי (LRC). המשמעות שהקוד מפיק n ביטים מהמילה מתוך הודעה באורך k, ולכל ביט ממילת הקוד, קיימים לכל היותר r ביטים אחרים כך שניתן לשחזר מהם את ערך הביט הנ"ל.

אנחנו אומרים שלקוד יש מקום/סביבה (r, d) אם לקוד יש מרחק $\leq d$ וכל ביט בהודעה (במילת קוד) יכול להיות משוחזר מ- r ביטים אחרים.

הערה: קיימים קודים הניתנים לשחזור מקומי חלשים וחזקים : ב- weak local recoverability רק ביטים מההודעה ניתנים לשחזור מקומי על ידי r ביטים אחרים ממילת קוד. ב- strong local recoverability עבור כל ביט ממילת הקוד אנו יכולים לשחזר אותו באמצעות r ביטים אחרים ממילת קוד.

שימושים ב-locally recoverable codes :

דוגמה מספר 1 :

חברה גדולה מחזיקה cloud ענק והמון דיסקים שבהם נשמר מידע רב כלשהו. עם הזמן חלק מהדיסקים מתקלקלים או נשרפים כי יש בלאי. כלומר יש בעיה מקומית. נרצה להביא למצב שאם דיסק אחד התקלקל, עדין נוכל לקרוא כמה דיסקים אחרים, (כלומר קצת מידע, ולא את כל הדיסקים) ובאמצעותם לשחזר את מה שהיה כתוב בדיסק שהתקלקל. מכיוון שזו בעיה שצריך לפתור בתדירות גבוהה, צריך לעשות את זה מהר.

איך נעשה את זה? ראשית, נניח שכל דיסק ניתן לשחזר מ-r דיסקים אחרים, ולכן פשוט נאסוף את המידע ואחרי כל r דיסקים שמכילים מידע נוסיף דיסק $r + 1$ שישמור את תוצאת ה-XOR של r הדיסקים הקודמים (סוג של קוד Parity שלמדנו).

כך, במידה ודיסק אחד נהרס, נצטרך לקרוא את r הדיסקים האלה ונחשב את ה-XOR שלהם וכך נקבל בדיוק את הדיסק הזה שנהרס לנו ונצליח לשחזר אותו. כלומר, קוד זה יודע להתמודד עם נתונים שנפגעו ולשחזר אותם כהלכה.

הערה: באופן עקרוני ניתן לפתור את זה עם הקוד שלמדנו של ההעתקה, וכך אפילו נצטרך לקרוא רק דיסק אחד(ולא r דיסקים), אך הבעיה שהאחסון של זה יקר מאוד ולכן, בפועל לא נשתמש בזה כי זה לא מומלץ.

<u>דוגמה מספר 2:</u>

מתקפת סייבר על חברה גדולה אשר מחזיקה cloud ענק והמון דיסקים שבהם נשמר מידע רב כלשהו. המתקפה מביאה למצב שכמות רבה של דיסקים התקלקלו בבת אחת (ולא אחד) . מכיוון שמקרה זה הוא חריג ואינו קורה בתדירות גבוהה, <u>פה לא נתעקש על כך שזה יהיה מהיר</u>, אך עדין לא נרצה לקרוא את כל המידע (המידע הוא רב וזה לא ריאלי). אז למעשה, לשחזר את המידע במצב הזה אנחנו כבר יודעים לעשות: ניקח את המידע ואותו נקודד עם הקוד לתיקון שגיאות ולאחר מכן נבדוק מה המרחק של הקוד הזה. אם הוא d אז נוכל לשחזר כל עוד מספר הדיסקים שהתקלקלו לנו הוא לכל היותר $d-1$. אם נעשה את זה עם קוד Reed- Solomon אז נצטרך לקרוא את כל המקומות.

2. locally testable codes-קודים הניתנים לבדיקה מקומית:

קוד לבדיקה מקומית הוא סוג של קוד לתיקון שגיאות שעבורו ניתן לקבוע אם מחרוזת היא אכן מילת קוד על ידי קריאת מספר קטן (קבוע לרוב) של ביטים מהמחרוזת (ביטים אלו יבחרו באופן אקראי, וכך "היריב" לא יוכל לצפות באילו ביטים נבחר). כלומר, הכוונה ב-"בדיקה מקומית" היא שבאמצעות דגימה של מספר קטן מאוד (סופי) של ביטים ממילת הקוד ניתן יהיה לזהות בהסתברות טובה האם היא חוקית או לא. במצבים מסוימים, נרצה לדעת אם הנתונים פגומים מבלי לפענח את כל המידע, כך שנוכל לבצע פעולות מתאימות בתגובה. ולכן נרצה לדעת להפריד בין שתי אופציות(ע"מ להבחין מה נכון ומה לא): a זאת מילה קוד נכונה ו-b מילה רחוקה ממילת קוד נכונה. ליתר דיוק - אם היא נכונה, תמיד נזהה זאת ואם היא לא נכונה, אז ככל שהיא רחוקה מלהיות מילת קוד חוקית/נכונה ההסתברות שלנו לזהות זאת תגדל משמעותית (מובן מאליו שאם נפלה שגיאה בביט בודד במילת הקוד אין סיכוי לזהות זאת בהסתברות גדולה על ידי בדיקה של מספר סופי של ביטים - הרי חייבים לקלוע בדיוק לביט שהתקלקל כדי שיהיה בכלל סיכוי להבין שמשהו השתבש).

<u>שימושים ב-locally testable codes:</u>

משתמשים בהם בעיקר שרוצים לבנות מטבעות וירטואליים או שרוצים לבנות הוכחות, נרצה להיות מסוגלים לבדוק מהר אם מה שמוכיחים לנו זאת טענה נכונה או שמנסים לרמות אותנו.

<u>דוגמה מספר 1:</u>

מטבעות וירטואליים – אדם כלשהו נותן הוכחה וטוען שהמטבע הזה הוא המטבע שלו. בשביל לבדוק את ההוכחה הזאת, לא נרצה לקרוא את ה-log של כל הטרנס אקסונים של כל המטבעות, אלא יספיק לנו לקרוא log של כמה כאלה וככה לדעת האם ההוכחה הזאת נכונה או לא.

<u>דוגמה מספר 2:</u>

משפט probabilistically checkable proof (PCP) . הרעיון של המשפט:

רוצים להוכיח טענה כלשהי, ההוכחה לטענה היא ארוכה מאוד ובנוסף, ניתן להציג אותה כשרשרת של גרירות. הבעיה בסוג הוכחה הנ"ל שמספיק שמישהו "ירמה" רק במקום אחד בהוכחה וזה יגרום לכך שכל ההוכחה תשתבש ותהפוך להיות לא נכונה. משפט PCP מיועד לסוג הוכחות כאלה והוא אומר שבשביל שההוכחה תהיה לא נכונה מישהו חייב "לרמות" בהרבה מקומות בהוכחה, כך נוכל לבדוק מהר מאוד את ההוכחה. איך? נקבל הוכחה, ואז כדי לבדוק אותה נבחר כמה מקומות ונקרא את המידע במקומות הללו, אם ההוכחה בהם נכונה אז רוב הסיכויים שההוכחה כולה נכונה, ואם לא אז ההוכחה בטוח לא נכונה.

3. locally decodable codes-קודים הניתנים לפענוח מקומי:

קוד לפענוח מקומי (LDC) הוא קוד לתיקון שגיאות המאפשר פענוח של ביט בודד מההודעה המקורית בסבירות גבוהה על ידי בחינה (או שאילתה) של מספר קטן של ביטים של מילת קוד שאולי מקולקלת. במילים אחרות, קודים אלו משתמשים במספר קטן של ביטים של מילת הקוד כדי לשחזר את המידע המקורי באופן הסתברותי. התדירות של השגיאות קובעת את הסבירות שהמפענח ישחזר נכון את הביט המקורי.

מאפיין זה יכול להיות שימושי, למשל, בהקשר שבו מידע מועבר בערוץ רועש, ורק תת-קבוצה קטנה של הנתונים נדרשת בזמן מסוים ואין צורך לפענח את ההודעה כולה בבת אחת.

אליס שולחת הודעה לבוב, ואת בוב מעניין רק ביט בודד מההודעה, הביט ה- i ,ולכן הוא לא מעוניין לקרוא את ההודעה כולה. נציין שבקודים רגילים, בוב יצטרך לקחת את כל ההודעה, לפענח אותה ורק אז יוכל לשחזר את הביט ה-i , אך עם קודים מקומיים אנחנו מעוניינים שבוב יקרא מספר קטן של ביטים ויהיה מסוגל לשחזר מתוכם את הביט ה-i .

שימו לב: עלולה להיות בעיה השיטה הזאת- אם בוב קורא למשל 3 ביטים, שאותם קבע מראש ולא בחר אותם באופן אקראי, ויש לנו רעש אדברסרילי. אז שלושת הביטים האלה בדיוק יכולים להיות משובשים, בגלל שהיה ידוע באילו ביטים הוא יבחר, ולכן המטרה היא לבחור את הביטים שנקרא באופן אקראי, ככה שגם אם יהי לנו רעש כזה, בסיכוי גבוהה נוכל לשחזר את הביט ה-i .

הערה: קודים לפענוח מקומי אינם תת קבוצה של קודים הניתנים לבדיקה מקומית, אם כי קיימת חפיפה מסוימת בין השניים.

<u>שימושים ב-locally decodable codes :</u>

שימוש רחב בבניית פרוטוקולים קריפטוגרפים, המאפשרים לנו לשמור על הפרטיות שלנו.

הערה: בהמשך השיעור נחזור לעסוק בקודים מסוג זה.

4. self correctable codes-קודים הניתנים לתיקון מקומי:

דיברנו על כך, שכאשר נרצה לשחזר איזשהו ביט מהר אז אנחנו מתמודדים עם זה שרק דיסק אחד נהרס. כעת נעסוק במצב בו למשל מישהו אדברסילי, שיבש לנו כמות קבועה של דיסקים, והביא להרס של כל הרצף הזה של r הדיסקים ובשאר לא נגע, אז כבר לא נוכל לשחזר. אך עדין נרצה במקרה כזה לבחור לקרוא כמה מוקמות באופן אקראי ומתוכם לשחזר את המידע.

קודים אלו, עובדים בצורה דומה ל-locally decodable codes , רק שבמקרה זה תחשבו שיש חלק מסוים של הודעות שהן משובשות ומעניין אותנו לשחזר ביט אחד מהקוד. כלומר, ההבדל ביניהם הוא שב- (LDC) אנחנו רוצים לשחזר את אחד מתוך k הביטים של ההודעה ופה נרצה לשחזר אחד מ-n ביטים של מילת הקוד, באופן לוקלי ובצורה מהירה.

הערה: סוג זה של קוד נחשב למושג חזק יותר מ- (LDC), כי כשיש לנו קוד לינארי בדרך הזאת נוכל תמיד לשמור את ההודעה כחלק מהקוד, בביטים הראשונים שלו. ואז אם נרצה לשחזר ביט כלשהו פשוט נעשה self correction לחלק הזה של ההודעה (להתחלה של הקוד).

נתעמק כעת ב-locally decodable codes :

<u>הגדרה:</u>

קוד $C \subseteq \sum^n$ כך ש: $C: F_1^k \to F_2^n$ נקרא locally decodable אם לכל $0 \le i \le k$ קיים אלגוריתם פענוח אקראי D_i כך ש-D_i קורא רק q מקומות ממילת הקוד ומתקיים: $d(c(m), w) \le \delta \cdot n$ אז:

$P_r\,(D_i(w) = m_i) = 1 - \varepsilon$. וזה נקרא: $(q, \delta, \varepsilon) - \text{LDC}$

הערה: שאילתה הכוונה, איזה מקום במילת קוד הוא מבקש. q הוא הפרמטר החשוב, וערכו הוא כמה שאילתות הקוד הזה עושה לתוך מילת הקוד המשובשת.

שימו לב: $D_i(w)$ אמור לשחזר לנו את הביט ה-i של המילה המקודדת.

<u>הסבר:</u>

אלגוריתם הפענוח לא מקבל את מילת הקוד W, אבל הוא יכול לבחור איזשהם q אינדקסים: $i_1, i_2, \ldots, i_q$ וכך לקבל את: $w[i_1], w[i_2], \ldots, w[i_q]$. כלומר אלגוריתם הפענוח לא יכול לקרוא את כל מילת הקוד, אז הוא קורא איזשהם q מקומות מתוך המילה(לא בהכרח ברצף) ובעזרתם הוא מצליח לשחזר ביט אחד מסוים מההודעה שמעניין אותנו.

שימו לב: ביטים ממילת קוד הם אינם ביטים מהודעה.

מושג: אלגוריתם פענוח D_i נקרא חלק אם כאשר מסתכלים על כל שאילתה שלו i_j אז היא מתפלגת באופן אחיד (כלומר, היא יכולה לקבל את אחד הערכים מ-1 עד n באותו סיכוי).

- אם D_i אלגוריתם פענוח אז הוא מפענח נכון מילות לא משבשות(זה תמיד). אך אם D_i אלגוריתם פענוח חלק, ונניח שקיימות $\delta \cdot n$ שגיאות אז: $\mathrm{P_r}\left(D_i(w) \neq m_i\right) \leq q \cdot \delta$.

הסבר: האלגוריתם שלנו קורא q מקומות, שכל אחד מהם מתפלג באופן אחיד. נרצה לדעת מה הסיכוי ש i_j שנבחר, ״נופל״ במקום לא נכון, כלומר משובש: נזכיר שיש $\delta \cdot n$ מקומות משובשים מתוך n מקומות ולכן הסיכוי ״ליפול״ על מקום אחד כזה הוא: $\frac{\delta \cdot n}{n} = \delta$, ומכאן נובע שהסיכוי ״ליפול״ על q מקומות כאלה זה לכל היותר $q \cdot \delta$. בנוסף, אם כל המוקמות לא משובשים אז פשוט נחזיר תשובה נכונה, כי הנחנו שכאשר האלגוריתם הפענוח שלנו קורא מילה נכונה/לא משובשת אז הוא מחזיר תשובה נכונה. ולכן:

בשביל לבנות קוד פיענוח מקומי צריך לבנות אלגוריתם פענוח D_i לכל i כך ש:

1. השאילתות מתפלגות בצורה אחידה
2. D_i מחזיר תשובה נכונה על מילת קוד לא משובשת

למעשה, ראינו כבר קוד כזה ולא הוכחנו שהוא כזה, מדובר בקוד Hadamard.

<u>תזכורת:</u> קוד Hadamard הוא קוד לתיקון שגיאות על שם Jacques Hadamard המשמש לזיהוי ותיקון שגיאות בעת העברת הודעות בערוצים רועשים מאוד או לא אמינים. קוד זה הוא דוגמה לקוד לינארי באורך 2^m מעל הא״ב הבינארי.

קוד Hadamard הוא גם דוגמה לקוד פשוט הניתן לפענוח מקומי, הממפה מחרוזת באורך k למילת קוד באורך 2^k כלומר: $C: \{0,1\}^k \to \{0,1\}^{2^k}$. איך קורה המיפוי?

מקבלים הודעה m ושולחים את כל הפונקציונליים הלינאריים של ההודעה הנ״ל:
$$m \to \left(\langle m, x\rangle\right)_{x \in \{0,1\}^k}$$

<u>טענה:</u> קוד Hadamard הוא $\mathrm{LDC} - (2, \delta, 2 \cdot \delta)$

<u>משמעות הטענה:</u> הקוד מפיק n=2 ביטים מהמילה מתוך הודעה באורך $k = \delta$, ולכל ביט ממילת הקוד, קיימים לכל היותר $r = 2 \cdot \delta$ ביטים אחרים כך שניתן לשחזר מהם את ערך הביט הנ״ל.

במילים אחרות, הוא יכול לבחור 2 מקומות y, x ולקבל את $\langle m,x\rangle$ ואת $\langle m,y\rangle$ ובאמצעותם לחשב את m_i.
אז איזה y, x נבחר כדי לקבל את m_i?

הערה: במקרה בו לא היה חשוב לנו ש- y, x (שתי השאילתות) יתפלגו באופן אחיד, והיינו צריכים לבחור שאילתה אחת ,אשר ממנה נקבל $\langle m,x\rangle = \sum m_j \cdot x_j$ והיינו רוצים למצוא את m_i אז נשים לב שעבור וקטור שכולו אפסים ובמקום ה-i יש 1 כך נקבל את y , ונוכל לדעת מהו m_i.

נרצה ש- y, x יתפלגו באופן אחיד ולכן, אלגוריתם הפענוח (אשר מפענח את הביט הi של ההודעה) יהיה:

- x נבחר באופן אקראי ואחיד ו- $y = x + e_i$. הכוונה היא להפוך את הביט ה-i ולבקש את הערך של הקוד במקומות האלה..
- [*] לאחר מכן נקבל בחזרה את מילות הקוד: $\langle m,x\rangle$, $\langle m,x+e_i\rangle$ כלומר אנחנו מקבלים את הסכום: $\langle m,x\rangle = \sum m_j \cdot x_j$ ובנוסף $\langle m,x+e_i\rangle = \sum m_j \cdot x_j + m_i$ שזה אותו הסכום רק שבמקום ה-i נוסיף את m_i . וכעת, כדי לקבל את m_i נחסיר בין שתי המשוואות הללו שקיבלנו ונמצא את $w[x] - w[y]$.

הערה: אם נדע מהו ה-y אנחנו עדין לא נדע כלום על ה-e_i. כי y יכול לקבל כל ערך ב- $\{0,1\}^k$ באופן אחיד .

שימו לב: אם יש לנו $\delta \cdot 2^k$ מקומות משובשים אז הסיכוי ש-y, x לא יהיו משובשים הוא לפחות $1 - 2\cdot\delta$

<u>לסיכום אלגוריתם הפענוח:</u> אלגוריתם D_i בוחר באופן אקראי $x \in \{0,1\}^k$ ו- $y = x + e_i$. שואל את $w[x]$ ואת $w[y]$ ומחזיר: $w[x] - w[y]$.

ההוכחה לטענה שהקוד הזה הוא קוד פיענוח לוקלי היא טריוויאלית מכיוון ש y, x שניהם מתפלגים באופן אחיד ואם לוקחים מילה אקראית לחלוטין והפוכים ביט אחד בה אז עדין מקבלים משהו אקראי לחלוטין וב-[*] הוכחנו שזה מחזיר דבר נכון.

הערה: אחת הבעיות המרכזיות בבנייה כזאת, היא שאורך הקוד הוא 2^k (אקספוננציאלי) כלומר ארוך מדיי.

<u>דוגמה נוספת ל-LDC - private information retrieval :</u>

יש משתמש, user בשם u ויש לו איזשהו מספר i ובנוסף ישנם כמה שרתים שלכולם יש גישה ל-data base.
נעסוק כעת בבעיה בה המשתמש שלנו לא רוצה שידעו מהו ה-i הזה שהוא מחזיק, שהוא מידע פרטי שלו.
המטרה שלו היא לגשת בדרך כלשהי ל-DB ולקבל משם את ה- i הזה, מבלי לגלות את ה-i הזה לאף אחד מהשרתים. נצא מנקודת הנחה שהשרתים שלנו לא מתקשרים אחד עם השני והמטרה שלנו היא לעשות את זה עם כמה שפחות חיבורים.

במילים אחרות, אנחנו רוצים לדעת מהו ה-$D[i]$ מבלי לגלות את i .

הפתרון הטריוויאלי זה לבקש את כל ה-DB ולסנן את המקום ה-i. אבל אנחנו יכולים להשתמש גם ב- locally decodable code כדי לשחזר את המקום ה-i הזה. איך נעשה זאת?

המשתמש יכול להסתכל על אלגוריתם הפענוח D_i, אלגוריתם D_i שלו ניגש למקום ה-x ולמקום ה-y , אז הוא יכול לבקש את s_1 לשלוח את $c(D[x])$ ומ-s_2 לשלוח את $c(D[y])$. למה זה מועיל?

נזכיר ש-y, x מתפלגים באופן אחיד, כלומר כל אחד מהשרתים מקבל אוסף של ביטים אקראיים ומתוכם הוא לא יכול להסיק מהם ה-y, x . אבל המשתמש מתוך שני הביטים האלה: $c(D[x])$ ו-$c(D[y])$ יכול להסיק מהו ה-$D[i]$ ע״י החישוב: $c(D[x]) - c(D[y])$.

קורס: קודים לתיקון שגיאות ושימושיהם במדעי המחשב
מרצה: ד"ר קלים יפרמנקו
סמסטר סתיו תשפ"א
10/01/2021

הערה: אם היינו רוצים להשתמש פה בקוד Hadamard אז הייתה בעיה, כי בשביל לשלוח את x היינו צריכים k ביטים ואז למעשה לא עשינו פה כלום, כי זה בעצם כמו שאחד השרתים ישלח לנו את אותם k ביטים מתוך ה-DB ואנחנו פשוט נשחזר אותו מתוכם.

כעת נראה בנייה של קודים לתיקון שגיאות שהם הרבה יותר קצרים. יש בנייה של locally decodable code כך ש: $n = 2^{2^{\sqrt{\log k}}}$.

אם נבנה איזשהו קוד locally decodable כאשר $n = \text{poly(k)}$ זה יהיה מצוין, כי זה אומר שיכולנו לבנות private information retrieval כך שהמשתמש היה שולח $O(\log n)$ ביטים, ומקבל בחזרה $\log n$ ביטים וכך מצליח לשחזר את ה-i.

שימו לב: בכל אופן חייב לשלוח $\log n$ ביטים כי אפילו במקרה שלא היה אכפת למשתמש שידעו את i, נצטרך $\log n$ ביטים כדי לשלוח את i.

פה אנחנו עושים בבנייה משהו באמצע, כי $\log k \ll 2^{\sqrt{\log k}} \ll k$. <u>אז בקצרה איך הבניה עובדת?</u>

1. S – matching vectors
2. איך בעזרת ה- S – matching vectors לבנות locally decodable קוד
3. אלגוריתם פענוח

<u>הגדרה</u>: קבוצה $\{u_i\}_{i=1}^{k} \subseteq Z_m^h$ נקראת S תואם עבור $s \subseteq \{m\}$ אם מתקיים:

1. לכל i: $\langle u_i, u_i \rangle = 0$ הכוונה: $\sum_{i=1}^{m} u_i[j]^2 \% m$
2. $\langle u_i, u_j \rangle \in S$
3. $0 \notin S$

<u>למה</u>: קיימים S – matching vectors, $\{u_i\}_{i=1}^{k} \subseteq Z_m^h$ עבור $m = 6$ ו- $|S| = 3$ נניח, $S = \{1, 3, 4\}$ כך ש:

$$h \leq 2^{\tilde{O}(\sqrt{\log n})}$$

הערה: הלמה הזאת מפתיעה כי אם היינו בוחרים לכל m שהוא מספר ראשוני אז אפשר להוכיח ש: $h \geq poly(n)$. פעם לאורך תקופה ארוכה האמינו ש- m הוא מכפלה של שני מישורים, ואז Grolmusz (וזאת הבנייה שלו) למעשה בנה את ה-S – matching vectors, ככה שאנחנו מקבלים משהו הרבה יותר טוב מפולונימלי ב-n.

<u>בנייה של הקוד</u>:

אנחנו בוחרים $\gamma \in F_q$ כך ש: γ יוצר מסדר m (הכוונה: $\gamma^m = 1$) וגם עבור $1 \leq j \leq m-1$ מתקיים: $\gamma^j \neq 1$. ונקבע ש- $\{u_i\}_{i=1}^{k}$ יהיה ה-S – matching vector כמו בלמה.

<u>איך הבנייה הזאת תעבוד?</u>

הקוד שלנו: $C: F^k \to F^{m^h}$ ואיך הוא יעבוד?

$$C(m_1, \dots, m_k) = \left(\sum m_i \gamma^{\langle u_i, x \rangle}\right)_{x \in Z_6^h}$$

הסבר: יש m^h קורדינאטות. לכל קורדינאטה נתאים איזשהו מקום ב- Z_m^h, כלומר עבור כל קורדינאטה x נחשב את הסכום: $m_i \gamma^{\langle u_i, x \rangle}$.

קורס: קודים לתיקון שגיאות ושימושיהם במדעי המחשב
מרצה: ד״ר קלים יפרמנקו
סמסטר סתיו תשפ״א
10/01/2021

<u>שאלה- מה האורך של הקוד כפונקציה של k ?</u>

נשים לב שהאורך של הקוד שלנו הוא 6^h . בנוסף, אנחנו יודעים לפי הלמה ש- $h \le 2^{\tilde{O}(\sqrt{\log n})}$ ולכן האורך של הקוד שלנו : $6^h \le 6^{2^{O(\sqrt{\log n})}}$. עכשיו נסתכל על פולינום מדרגה 4:

$$p(x) = \prod_{i \in S}(x - \gamma^i) = (x - \gamma^1)\cdot(x - \gamma^3)\cdot(x - \gamma^4) = \sum_{i=0}^{3} a_i \cdot x^i$$

נשים לב: שלכל $i \in S$ אז: $p(\gamma^i) = 0$

<u>אלגוריתם פענוח במקרה שלנו:</u>

1. בחר את x באופן אקראי: $x \in Z_6^h$,
2. עכשיו תדגום 4 מקומות: $x\%6\,,\ (x+u_i)\%6\,,\ (x+2\cdot u_i)\%6\,,(\,x+3\cdot u_i)\%6$
3. חשב את: $c_i = a_0\cdot w[x] + a_1\cdot w[x+u_i] + a_2\cdot w[x+2\cdot u_i] + a_3\cdot w[x+3\cdot u_i]$
4. תחזיר את: $m_i = c_i \cdot \gamma^{-\langle u_i,x\rangle}\cdot p(1)^{-1}$

שימו לב: מכיוון שאנחנו בוחרים את איקס באופן אקראי אז גם כל אחד מהמקומות שבחרנו: $x\,,\ x+u_i\,,\ x+2\cdot u_i\,, x+3\cdot u_i$, מתפלג באופן אחיד כלומר כל השאילתות מתפלגות באופן אחיד.

נשאר להוכיח: שאם במקום W נציב את מילת הקוד הנכונה אז נקבל את התוצאה הנכונה, כלומר אם במקום ה-W נציב את מילת הקוד מה יהיה c_i?

$$c_i =$$

$$a_0\cdot\sum_{j=0}^{k} m_j\cdot\gamma^{\langle u_j,x\rangle} + a_1\cdot\sum_{j=0}^{k} m_j\cdot\gamma^{\langle u_j,x+u_i\rangle} + a_2\cdot\sum_{j=0}^{k} m_j\cdot\gamma^{\langle u_j,x+2\cdot u_i\rangle} + a_3\cdot\sum_{j=0}^{k} m_j\cdot\gamma^{\langle u_j,x+3\cdot u_i\rangle}$$

$$= \sum_{j=0}^{k} m_j\cdot\left(a_0\cdot\gamma^{\langle u_j,x\rangle} + a_1\cdot\gamma^{\langle u_j,x+u_i\rangle} + a_2\cdot\gamma^{\langle u_j,x+2\cdot u_i\rangle} + a_3\cdot\gamma^{\langle u_j,x+3\cdot u_i\rangle}\right) =$$

$$= \sum_{j=0}^{k} m_j\cdot\gamma^{\langle u_j,x\rangle}\left(a_0 + a_1\cdot\left(\gamma^{\langle u_j,u_i\rangle}\right)^1 + a_2\cdot\left(\gamma^{\langle u_j,u_i\rangle}\right)^2 + a_3\cdot\left(\gamma^{\langle u_j,u_i\rangle}\right)^3\right) =$$

$$=* \sum_{j=0}^{k} m_j\cdot\gamma^{\langle u_j,x\rangle}\left(p(\gamma^{\langle u_j,u_i\rangle})\right) =** m_i\cdot\gamma^{\langle u_i,x\rangle}\left(p(\gamma^{\langle u_i,u_i\rangle})\right)$$

*תזכורת לתכונה שיש לפולינום שבחרנו: שלכל $i \in S$ אז: $p(\gamma^i) = 0$ אז בעצם כאשר $i \neq j$ מתקיים ש: $\langle u_j,\ u_i\rangle \in S$

** ולכן זה מה שנשאר לנו מכל הסכום הזה כי לכל $i \neq j$ הביטוי הזה $p(\gamma^{\langle u_j,u_i\rangle})$ שווה לאפס ולכן נשארים רק אם המקרה ש $i = j$