# מבני נתונים - תרגיל מעשי 2

## מגישים :


נועה ארז

ת״ז : 322467044

שם משתמש : noaerez

גיא גולדרט

ת״ז : 318515996

שם משתמש : GuyGoldrat

## תיאור המחלקה האבסטרקטית OAHashTable

שדות:

1. HashTableElement טבלה המיוצגת באמצעות מערך בגודל m של עצמים מטיפוס HashTableElement.
2. m- ערכו של הפרמטר.

תיאור הפונקציות:

1. הפונקציה **find(key) :**
הפעולה מבצעת סדרת חיפוש בשיטת open addressing עבור המפתח key באמצעות שימוש במתודה האבסטרקטית hash.
אם key מופיע בתא מסוים בסדרת הבדיקות הפונקציה מחזירה את האיבר בתא. אחרת, אם אחד התאים ריק, המתודה מסיימת את סדרת הבדיקות ומחזיקה null. במידה והמפתח key אינו נמצא בסדרת החיפוש הפונקציה תחזיר null.

2. הפונקציה **insert(HashTableElement hte) :**
ראשית, הפונקציה בודקת באמצעות הפעולה find האם המפתח של האיבר the שהתקבל מופיע כבר בטבלה. במידה וכן, תזרוק שגיאת KeyAlreadyExistsException.
אחרת, נבצע סדרת בדיקות בשיטת open addressing עבור המפתח hte.key באמצעות שימוש במתודה האבסטרקטית hash. נכניס את האיבר החדש לתא הראשון הפנוי בסדרת הבדיקות ונסיים, כאשר תא פנוי הוא תא המכיל ערך null או אובייקט מטיפוס HashTableElement בעל מפתח 1- כיוון שהיה בו איבר אחר שנמחק. במידה ולא נמצא תא מתאים לאורך סדרת הבדיקות המתודה תזרוק שגיאת TableIsFullException.

3. הפונקציה **delete(key) :**
הפעולה מבצעת סדרת חיפוש עבור המפתח key בשיטת open addressing באמצעות שימוש במתודה האבסטרקטית hash.
במידה ואחד התאים ריק, נעצור את סדרת הבדיקות ונזרוק שגיאת KeyDoesntExistException. אחרת, כאשר נגיע לתא בטבלה שמפתחו שווה לkey נמחק אותו על ידי הכנסת אובייקט מטיפוס HashTableElement המכיל את הערכים 1- בתור key ו- value. במקרה והסתיימה סדרת הבדיקות ולא נמצא איבר בעל מפתח זהה בטבלה, נזרוק שגיאת KeyDoesntExistException .
משום שהמפתחות המוכנסים למבנה הם חיובים, באמצעות כך נוכל לזהות מהם התאים בהם הופיעו איברים שנמחקו.

תיאור המחלקה DoubleHashTable היורשת מהמחלקה האבסטרקטית :

על מנת לממש את פונקציית Hash בטבלה מסוג זה עלינו להגדיר שתי פונקציות hash אוניברסליות באופן בלתי תלוי זו בזו.
את הפונקציה הראשונה בחרנו באקראי מתוך משפחת הפונקציות האוניברסליות עם הפרמטרים p ו- m. כעת, על מנת שהפונקציה השנייה תחזיר ערכים בטווח 0 עד m-1 באופן בלתי תלוי לבחירת הפונקציה הראשונה נבחר שוב פונקציה באקראי מתוך המשפחה האוניברסליות, הפעם עם הפרמטרים p ו- m-1.

## שאלה 3

סעיף 1 :

$$Q_1 = \{i^2 \; mod \; q | \; 0 \le i < q\} = 3286$$

$$Q_2 = \{(-1)^i \cdot i^2 \; mod \; q | 0 \le i < q\} = \; 6571$$

סעיף 2 :

מתוך 100 הטבלאות שיצרנו בכ-70 מהם נזרק חריג מסוג TableIsFullException. לעומת זאת כאשר חזרנו על הפעולה עם $AQPHashTable$ לא נזרקו חריגים.
למעשה, מספר האיברים בקבוצה $Q_1$ הוא מספר האינדקסים השונים שנגיע אליהם במהלך Probing בטבלת quadratic probing. כלומר, במהלך ביצוע probing בטבלת $QPHashTable$ נעבור על מחצית מהתאים, לכן שיטת בדיקה זו אינה נותנת תמורה על התאים בטבלה.

בעת הכנסה לטבלה עלינו לעבור על פרמוטציה של תאי הטבלה עד שנגיע לתא ריק אליו נכניס את האיבר החדש. אולם, בטבלת quadratic probing נבדוק רק מחצית מתאי הטבלה, לכן במידה וכבר הוכנסו $\frac{m}{2}$ איברים במיקומים אותם בדקנו, ייזרק חריג למרות שבפועל קיימים מקומות פנויים בטבלה.

לעומת זאת, בטבלת $AQPHashTable$ לא נזרקו חריגים מפני שכפי שניתן לראות הקבוצה $Q_2$ מכילה m איברים שונים, כלומר סדרת הבדיקות מחזירה תמורה על תאי הטבלה.
על כן, כאשר נבצע probing בטבלה נצטרך לעבור על כל m התאים על מנת לקבוע שהטבלה מלאה. לכן, במידה ונותרו תאים ריקים בטבלה תתבצע הכנסה אליהם ולא יוחזר חריג.

סעיף 3 :

מספר a נקרא שארית ריבועית מודולו p אם קיים פתרון שלם למשוואה הבאה $x^2 \equiv a(mod \; p)$ . לכל מספר ראשוני p>2 קיימות בדיוק $\frac{p-1}{2}$ שאריות ריבועיות (לא כולל השארית 0) ו- $\frac{p-1}{2}$ שאריות שאינן ריבועיות.

המספר הראשוני p שנבחר בשאלה זו מקיים $p = 3(mod 4)$ . לכל מספר ראשוני המקיים תכונה זו מתקיים פיזור אנטי סימטרי של השאריות הריבועיות. כלומר, אם $n(mod \; p)$ הוא שארית ריבועיות מודולו p אז $-n(mod \; p)$ הוא אינו שארית ריבועית.

לכן, במקרה זה נקבל מיפוי של האיברים בטווח $0 \le i < p$ ל- $2\frac{p-1}{2} + 1 = p$ ערכים שונים.
מתכונה זו נובע כי עבור טבלת AQP וראשוני p = 6571 המקיים את התכונה המתוארת, נקבל מיפוי לp תאים שונים בטבלה. בטבלת QP נקבל מיפוי ל $\frac{p-1}{2}$ ערכים שהם למעשה השאריות הריבועיות מודולו p , וזאת בהתאם לתוצאות המדידה.

לעומת זאת, אם p ראשוני מקיים $p = 1(mod \; 4)$ מתקיים פיזור סימטרי של השאריות הריבועיות, כלומר בהינתן ש$n(mod \; p)$ הוא שארית ריבועית מודולו p אז גם $-n(mod \; p)$ הוא שארית ריבועית כזו.

לכן, עבור p כמתואר, טבלת AQP אינה תמפה את האיברים ל p ערכים שונים אלא רק ל$\frac{p-1}{2}$ , כלומר למחצית מתאי הטבלה. במקרה זה, טבלת AQP תניב פיזור זהה לפיזור של טבלת QP.

## שאלה 4

סעיף 1 :

| Class | Running Time |
|---|---|
| LPHashTable | 0.522 seconds |
| QPHashTable | 0.524 seconds |
| AQPHashTable | 0.565 seconds |
| DoubleHashTable | 0.624 seconds |

בשאלה זו מדדנו את הזמן הדרוש להכנסת האיברים לטבלה בלבד (ללא הזמן שנדרש להגרלת המפתחות). ניתן לראות כי ארבעת סוגי הטבלאות QP LP ו-AQP פועלות בזמני ריצה כמעט זהים וזאת משום שהטבלה רק חצי מלאה.

בדומה לניתוח מההרצאה, היחס בין מספר האיברים בטבלה לבין גודלה הוא $\frac{1}{2}$ ,

כלומר $\frac{1}{2} = \frac{n}{m} = \alpha$, ואכן ראינו שעבור $\alpha \leq 0.6$ ההבדל בזמני הריצה בין הטבלאות השונות הוא זניח. ההבדל נעשה משמעותי כאשר הטבלה קרובה למלאה ומספר ההתנגשויות גדל.

לעומת זאת, DoubleHashTable פעל בזמן הארוך ביותר, זאת מכיוון שפעולות החישוב אותן הוא מבצע מסובכות יותר מבשאר סוגי הטבלאות, וכאשר אין הרבה התנגשות בטבלה העובדה ששיטת בדיקה זו נותנת תמורה שמתפלגת אחיד לא באה לידי ביטוי.

סעיף 2 :

| Class | Running Time |
|---|---|
| LPHashTable | 3.973 seconds |
| AQPHashTable | 2.240 seconds |
| DoubleHashTable | 2.823 seconds |

טבלת QPHashTable אינה מחזירה פרמוטציה של סדרת הבדיקות ולמעשה תבדוק רק כמחצית מתאי הטבלה. לכן, כאשר ננסה להכניס $\frac{19}{20}m$ איברים לטבלה סביר כי סדרת בדיקות מסוימת של מחצית מאיברי הטבלה תהיה מלאה וכאשר נבצע probing בטבלת QP על סדרה זו, נקבל שהטבלה מלאה ואין מקום פנוי להכנסת האיבר, לכן תיזרק שגיאת TableIsFullException.

בניסוי זה ניתן לראות שLP הוא בעל זמני הריצה הגרועים ביותר מפני שכפי למדנו בשיעור לאט לאט ייווצרו שרשראות רציפות של איברים בטבלה. בשיטה זו, הסיכוי להכנסה לתא מסוים אינו $\frac{1}{m}$ אלא תלוי כעת בגודל הבלוק שלפניו ובהכנסות קודמות. לכן, כאשר נכניס איבר עבורו חישוב פונקציית ה Hash מחזירה תא שנמצא בתוך רצף כזה נצטרך להגיע לסופו, דבר הלוקח זמן רב. כלומר, כאשר הטבלה קרובה למלאה ההבדל בזמני הריצה בין הבדיקה הליניארית לשאר הבדיקות כבר אינו זניח.

מתוצאות הניסוי ניתן לראות כי DoubleHash פעל בזמני ריצה טובים יותר מLP (למרות שבניסוי הקודם היה איטי יותר) וזאת מכיוון שDoubleHash מבצע סדרת בדיקות המהווה תמורה שמתפלגת אחיד על איברי הטבלה. בניסוי זה הטבלה כמעט מלאה לכן ישנן יותר התנגשויות בעת הכנסה ומשום כך יתרון זה בא לידי ביטוי לעומת שיטות הבדיקה האחרות. מנגד, DoubleHash מצריך גישות IO רבות לזכרון וחישובים מסובכים יותר משיטות probing אחרות מאחר ושתי פונקציות Hash צריכות להיות מחושבות.

עם זאת, על אף שDoubleHash נותן תמורה שמתפלגת אחיד יותר מAQPHashTable ,
מתוצאות הניסוי נובע כי ל AQP זמני ריצה טובים יותר כאשר הטבלה קרובה למלאה.
ניתן להסביר זאת על ידי כך ש AQP מבצע פחות חישובים בביצוע הprobing ונדרשות פחות גישות לזכרון, לכן פועל בזמן מהיר יותר.

## **שאלה 5**

| Iterations | Running Time |
|---|---|
| First 3 iterations | 5.4 seconds |
| Last 3 iterations | 14.8 seconds |

ניתן לראות כי בשלושת האיטרציות האחרונות זמן הריצה גדול פי שלוש מאשר בשלושת האיטרציות הראשונות. זאת, על אף שבכל איטרציה התחלנו מטבלה ריקה וביצענו הכנסה של אותה כמות איברים.

ההבדל בין זמני הריצה, נובע מכך שבעת מחיקה אנו מסמנים בסימון מיוחד תאים בהם היה איבר שנמחק. כך, כאשר נגיע לתא מסומן במהלך סדרת הבדיקות של פעולת Find בתהליך ה probing, נתייחס אל התא כתפוס ונמשיך הלאה משום שייתכן והאיבר נמצא בהמשך הטבלה.

לאחר האיטרציה הראשונה יישארו בטבלה $\frac{m}{2}$ תאים המסומנים בסימון זה, ובכל איטרציה נוספת יתווספו בין $\frac{m}{2}$ ל 0 תאים כאלו לטבלה. בתחילת כל פעולת Insert נבצע Find למפתח כדי לוודא שלא נכניס את אותו מפתח פעמיים. כעת, ישנן פעולת Find שיקח להם זמן רב יותר מאחר וכל אחת תצטרך לעבור על תאים נוספים המסומנים בסימון מיוחד אשר נוצרו באיטרציות הקודמות.

בשלושת האיטרציות האחרונות הרבה מתאי הטבלה כבר מסומנים כתאים שנמחקו בעבר, למרות שבפועל אין בהם איברים מסדרת ההכנסות הנוכחית. משום שבסדרת הבדיקות יתבצע מעבר על תאי הטבלה עד שיימצא תא ריק ולא מסומן, זמן הריצה של כל אחת מהפעולות יגדל ומכאן נובע ההבדל בזמני הריצה.