חלק א' – תיעוד:

$ModHash$: מחלקה המייצגת פונקציה מהמשפחה האוניברסלית של פונקציות מהצורה $((ax+b)mod\ p)mod\ m$

מתודה סטטית $GetFunc$ – מתודה סטטית שמגרילה פונקציית hash מהמשפחה האוניברסלית.
המתודה מקבלת את הערכים m ו-p, מגרילה פרמטר a בין 1 ל-p ומגרילה פרמטר b בין 0 ל-p.
קוראת לבנאי ומחזירה פונקציה עם אותם ערכי m,p,a,b.

מתודה $Hash$ - המתודה מקבלת מפתח ומחשבת את הפונקציה עליו על ידי (((a*key + b)%p)%m).

$OAHashTable$: מחלקה אבסטרקטית המממשת את הממשק של $IHashTable$ ומייצגת טבלת hash עם open addressing

שדה m – גודל הטבלה

שדה table – מערך $HashTableElements$ בגודל m.

מתודה Find – מממשת את המתודה Find בממשק IHashTable.
מקבלת מפתח לחיפוש ורצה על m הבדיקות הראשונות בסדרת הבדיקות שלו בעזרת המתודה האבסטרקטית Hash. בכל אינדקס שמתקבל בסדרת הבדיקות, אם התא באינדקס הזה הוא null, הפונקצייה קוטעת את סדרת הבדיקות ומחזירה null, האיבר לא נמצא בטבלה. אם התא מכיל איבר עם המפתח שמחפשים, המתודה תחזיר את האיבר שבתא. אם לאחר כל m הבדיקות לא נמצא איבר עם המפתח שחופש, מוחזר null.

מתודה Insert – מממשת את המתודה Insert בממשק IHashTable.
מקבלת איבר מסוג $HashTableElements$ ו ורצה על m הבדיקות הראשונות בסדרת הבדיקות של המפתח שלו בעזרת המתודה האבסטרקטית Hash. אם הגענו לתא בו איבר עם מפתח זהה, נזרוק חריג $KeyAlreadyExistsException$. אם הגענו תא null, נכניס בו את האיבר.
אם הגענו לתא "מחוק", נמשיך את סדרת הבדיקות עד שנגיע לתא null או עד סוף הסדרה, כדי לבדוק האם אין בהמשך סדרת הבדיקות איבר עם מפתח זהה. אם מצאנו איבר כזה בהמשך, נזרוק חריג $KeyAlreadyExistsException$, אחרת נכניס את האיבר בתא ה"מחוק" הראשון.
אם סיימנו את סדרת החיפושים מבלי שהכנסנו את האיבר או שנזרק חריג, נזרוק חריג מסוג $TableIsFullException$.

מתודה Delete – מממשת את המתודה Delete בממשק IHashTable.
מקבלת מפתח למחיקה ורצה על m הבדיקות הראשונות בסדרת הבדיקות שלו בעזרת המתודה האבסטרקטית Hash. בכל אינדקס שמתקבל בסדרת הבדיקות, אם התא באינדקס הזה הוא null, נזרוק חריג $KeyAlreadyExistsException$, האיבר לא נמצא בטבלה. אם התא מכיל איבר עם המפתח שמחפשים, המתודה תשים בתא הזה איבר "מחוק", אותו בחרנו להיות איבר עם מפתח וערך 1-. אם לאחר כל m הבדיקות לא נמצא איבר עם המפתח שחופש נזרוק חריג $KeyAlreadyExistsException$.

מתודה אבסטרקטית Hash – מקבלת מפתח ומספר איטרציה, מחזירה את הבדיקה שמתאימה לאיטרציה בסדרת הבדיקות של המפתח. זוהי מתודה אבסטרקטית ולכן המימוש נמצא במחלקות הבנות. מחלקות הבת של $OAHashTable$ ממשות את Hash עבור סוגי probing שונים.

$LPHashTable$: מחלקת בת של $OAHashTable$ המייצגת טבלה עם linear probing.

למחלקה איבר מסוג ModHash המיצג את הפונקציה h'.

מתודה Hash –מחזירה את הבדיקה ה-i בסדרת הבדיקות של המפתח k בעזרת החישוב :
$h(k,i) = (h'(k) + i) \ mod \ m$. אם הערך שמתקבל שלילי, נתקן אותו על ידי הוספת m.

$QPHashTable$: מחלקת בת של $OAHashTable$ המייצגת טבלה עם quadratic probing.

למחלקה איבר מסוג ModHash המיצג את הפונקציה h'.

מתודה Hash –מחזירה את הבדיקה ה-i בסדרת הבדיקות של המפתח k בעזרת החישוב :
$h(k,i) = (h'(k) + i^2) \ mod \ m$. אם הערך שמתקבל שלילי, נתקן אותו על ידי הוספת m.

$AQPHashTable$: מחלקת בת של $OAHashTable$ המייצגת טבלה עם alternating quadratic probing.

למחלקה איבר מסוג ModHash המיצג את הפונקציה h'.

מתודה Hash –מחזירה את הבדיקה ה-i בסדרת הבדיקות של המפתח k בעזרת החישוב :
$h(k,i) = \left(h'(k) + (-1)^i \cdot i^2\right) \ mod \ m$. אם הערך שמתקבל שלילי, נתקן אותו על ידי הוספת m.
במקום לכפול 1- במשך i פעמים, נבדוק אם i זוגי, אם כן נוסיף i^2, אחרת נחסר i^2.

$DoubleHashing$: מחלקת בת של $OAHashTable$ המייצגת טבלה עם double hashing.

למחלקה שני איברים מסוג ModHash המיצגים את הפונקציות h'_1, h'_2. את h'_1 נגדיר כרגיל להחזיר ערכים בין 0 ל-m. את h'_2 נגדיר להחזיר ערכים בין 0 ל-m-2. בהמשך, בחישוב ההאש, נוסיף ל-h'_2 1. כלומר היא תיתן ערכים בין 1 ל-m-1. בגלל ש-p ראשוני, נקבל ששתי הפונקציות זרות.

מתודה Hash –מחזירה את הבדיקה ה-i בסדרת הבדיקות של המפתח k בעזרת החישוב :
$h(k,i) = (h'_1(k) + i \cdot (h'_2(k) + 1)) mod \ m$. אם הערך שמתקבל שלילי, נתקן אותו על ידי הוספת m.

חלק ב' – ניסויים:

<u>שאלה 3:</u>

1. עבור המספר הראשוני $q = 6571$, גדלי הקבוצות הבאות הם:

$$|Q_1| = |\{i^2 \; mod \; q| \; 0 \le i < q\}| \;=\; 3286$$
$$|Q_2| = \left|\left\{(-1)^i \cdot i^2 \; mod \; q \middle| 0 \le i < q\right\}\right| \;=\; 6571$$

2. כאשר ביצענו את התהליך עם *QPHashTable*, נזרקו חריגים מסוג *TablesFullException* בכ-70% מהחזרות.
כאשר ביצענו את התהליך עם *AQPHashTabke*, לא נזרקו חריגים כלל.

נשים לב שעל פי סעיף 1, עבור *AQPHashTable* (כלומר Q2 בסעיף 1) יש בדיוק m איברים בסדרת הבדיקות, ולכן זוהי פרמוטציה. כלומר ניתן להגיע אל כל מקום בטבלה בעזרת סדרת בדיקות של כל מפתח. עבור כל מפתח שנרצה להכניס, כל עוד יש מקום פנוי בטבלה סדרת הבדיקות של המפתח תגיע לתא פנוי. לכן ייזרק חריג רק כאשר אכן כל m התאים בטבלה מאוכלסים ולא ניתן להכניס מפתח נוסף.

לעומת זאת, עבור *QPHashTable* (כלומר Q1 בסעיף 1), מספר התאים הנגישים בסדרת הבדיקות קטן ממש מ-m (למעשה, $\frac{m+1}{2}$), ולכן יתכנו תאים שאינם נגישים בעזרת סדרת בדיקות עבור מפתחות מסוימים. כלומר ייתכן מצב בו נרצה להכניס מפתח לטבלה שאינה מלאה, אך סדרת הבדיקות של אותו מפתח תגיע רק לתאים תפוסים, ולכן בסוף סדרת הבדיקות לא יימצא מקום פנוי להכנסת המפתח, וייזרק חריג *TablesFullException*.

3. מספר a נקרא שארית ריבועית מודולו m אם קיים מספר שלם x שמקיים $x^2 \equiv a \; (mod \; m)$.
נשים לב שעבור $QPHashTable$, המרחקים בין התאים בסדרת הבדיקות הם שאריות ריבועיות מודולו m כאשר x הוא $0...m$. ניעזר בטענות לגבי חשבון מודולרי.

<u>מתי $QPHashTable$ עשויה לזרוק חריג?</u>

עבור כל $m>2$ ראשוני, בדיוק $\frac{m-1}{2}$ מהשאריות הן שאריות ריבועיות ו- $\frac{m-1}{2}$ נוספות אינן, כאשר השאריות הן $\{1,...,m-1\}$ ו-0 אינו שארית.
לכן עבור m כזה גודל סדרת הבדיקות של מפתח ב-$QPHashTable$ הוא $\frac{m-1}{2}$ בלבד, ולכן עלול להיזרק חריג על אף שהטבלה אינה מלאה.

סדרת הבדיקות היא פרמוטציה של $\frac{m-1}{2}$ מספרים בין 1 ו-m. לכן כאשר בטבלה לכל היותר $\frac{m-1}{2}$ איברים, גדול סדרת הבדיקות של כל מפתח גדול ממספר התאים התפוסים, כלומר בהכרח נגיע לתא פנוי ולא ניתקל בבעיה. (נשתמש בעובדה זו בשאלה 4).

<u>למה $AQPHashTable$ לא זרק חריג?</u>

נראה שעבור כל $m>2$ ראשוני המקיים $m \equiv -1(mod \; 4)$, סדרת הבדיקות של *AQPHashTable* היא פרמוטציה, ולכן התופעה לא מתרחשת.
ניעזר בטענות נוספות:
1- אינו שארית ריבועית כאשר $m \equiv -1(mod \; 4)$
אם $x^2 \equiv y^2(mod \; m)$ אז $m = x + y$.

נניח בשלילה שסדרת הבדיקות אינה פרמוטציה. לכן קיימים $i \neq j$ כך ש- $0 \leq i,j \leq m-1$ ומתקיים $h(k,i)=h(k,j)$ ולכן $i^2 \equiv j^2 (mod\ m)$. ב-*AQPHashTable* יש חשיבות לזוגיות של האינדקס, לכן נפריד למקרים :

אם ל-*i,j* אותה זוגיות – אז לפי הטענה השנייה *i+j=m*. אבל ל-*i,j* אותה זוגיות, לכן הצד השמאלי זוגי, בעוד ש-*m* ראשוני ולכן אי-זוגי. סתירה.

אם הזוגיות שונה – אז $\left(\frac{i}{j}\right)^2 = -1\ (mod\ m)$, כלומר *-1* הוא שארית ריבועית, וזו סתירה לטענה הראשונה.

שאלה 4:

בשאלה הבאה המידות מתייחסות לזמן שלוקח להכניס את האיברים לטבלה בלבד, מבלי להתייחס לזמן שלוקח ליצור את הטבלאות ואת האיברים. יחידות זמן של שניות.

.1 $n = \left\lfloor \frac{m}{2} \right\rfloor$

Class	***Running Time (s)***
LPHashTable	*0.873*
QPHashTable	*0.901*
AQPHashTable	*0.943*
DoubleHashTable	*1.215*

זמן הריצה של שלוש הטבלאות הראשונות דומה, בעוד ש- *DoubleHashTable* רצה בזמן ארוך יותר.
זמן הריצה הארוך של *DoubleHashTable* נובע ראשית מכך שחישוב פונקציית ההאש ארוך יותר מהטבלאות האחרות. יש צורך בחישוב שתי פונקציות שונות. בנוסף, סדרת הבדיקות מתפלגת באופן אחיד על כל m התאים, לכן יתכנו גישות רבות ויקרות לזיכרון.

בשאר הטבלאות, חישוב פונקציית ההאש הוא זול יותר. נעשה שימוש בפונקציה אחת בלבד. בנוסף המרחק בין כל שתי בדיקות בסדרת הבדיקות מוגבל, ואף תא יחיד במקרה של *LPHashTable*, ולכן יש פחות גישות יקרות לזיכרון.

.2 $n = \left\lfloor \frac{19m}{20} \right\rfloor$

Class	***Running Time (s)***
LPHashTable	*11.714*
QPHashTable	*Not tested*
AQPHashTable	*5.158*
DoubleHashTable	*7.317*

כפי שהראינו בשאלה 3, *QPHashTable* עלולה לזרוק חריג, ולכן לא נבצע עליה את הניסוי. בשאלה 3 הראינו שסדרת הניסויים של *QPHashTable* היא פרמוטציה של $\left\lceil \frac{m}{2} \right\rceil$ מספרים עד m. לכן בהרצה הראשונה מספר התאים התפוסים תמיד קטן מגודל סדרת הבדיקות האפשרית, כלומר בסוף כל סדרת בדיקות יימצא תא פנוי ולא ייזרק חריג.

כעת בהרצה השנייה, מספר התאים התפוסים גדול ממש מגודל סדרת הבדיקות, ולכן ייתכן שייזרק חריג למרות שהטבלה אינה מלאה.
AQPHashTable אינה מהווה בעיה, מפני ש-10,000,019 ראשוני ושקול ל-1mod4-, ובשאלה 3 סעיף 3 הראינו שזהו תנאי מספיק לכך ש-*AQPHashTable* לא תזרוק חריג.

ניתן לראות שכעת, זמן הריצה של *LPHashTable* ארוך משל שאר הטבלאות. כאשר האיברים גדל, ב- *LPHashTable* נוצרת בעיה של הצטברות משנית – רצפים זהים של סדרות בדיקות, שמתבטאים ברצפים של תאים תפוסים בזיכרון. אם הנחיתה הראשונה בסדרת בדיקות של איבר כלשהו היא בתא תפוס מתוך רצף תאים שכזה, נעבור על כל הרצף התפוס עד שנוסיף את האיבר בסופו. כך הרצפים ממשיכים לגדול. מהניסוי ניתן לראות שבעיה זו מתעצמת כאשר הטבלה מתמלאת.

הטבלה השנייה באורך זמן הריצה היא *DoubleHashTable*. הטבלה אמנם מטפלת לחלוטין בהצטברות משנית, אך הגישות היקרות לזיכרון גורמות לה להיות ארוכה יותר מאשר *AQPHashTable*.

AQPHashTable היא הטבלה שרצה בזמן המהיר ביותר. אמנם בניגוד ל-*DoubleHashTable* קיים חשש להצטברות, אך מפני שהמרחקים בסדרת החיפוש גדולים יותר מ-1, הטיפול בכך יותר טוב מאשר ב- *LPHashTable*.
היתרון של- *AQPHashTable* על פני *DoubleHashTable* הוא בזמני הגישה לזיכרון ובכך שחישוב פונקציית ההאש של *AQPHashTable* קצר יותר.

שאלה 5 :

איטרציות	*זמן ריצה*
3 ראשונות	*10.698*
3 אחרונות	*23.795*

זמן הריצה של 3 האיטרציות האחרונות ארוך יותר.
כאשר מוחקים תא, אנו מסמנים אותו בתור תא "מחוק", ולא שמים בו null. נעשה זאת כדי שבמהלך חיפוש מפתח לא נסיים את החיפוש כאשר נגיע לאיבר מחוק, אלא רק כאשר נגיע למפתח הרצוי או לתא ריק שטרם הוכנס אליו איבר, וכך לא נקטע את סדרת החיפושים שעשויה להימשך גם אחרי התא המחוק.

כלומר, כאשר מבצעים חיפוש, עוברים על כל התאים התפוסים, וגם המחוקים שנמצאים בסדרת החיפוש. למרות שאינם מייצגים עוד איברים במבנה הנתונים, תאים מחוקים משפיעים על זמן הריצה כמו תאים תפוסים. ככל שהוכנסו למבנה יותר איברים, בין אם הם נמחקו או לא, זמן הריצה מתארך.

בכל פעולת insert מבצעים חיפוש כדי לבדוק אם האיבר כבר קיים
במבנה. רץ עד שהוא מגיע לתא ריק (כי המפתחות לא קיימים בטבלה). ככל שמבצעים יותר מחיקות, יותר תאים בטבלה כבר לא ריקים ומסומנים במקום כ-"deleted". לכן זמן הריצה של find ארוך יותר באיטרציות המאוחרות.