מגישים:

יותם חזן – 205566870, שם משתמש: yotamhazan
כפיר גרינברג – 313434037, שם משתמש: kfirgrinberg

חלק א – מימוש

מסמך תיעוד – OAHashTable

שדות המחלקה:

table – מערך המכיל אובייקטים מסוג HashTableElement. מערך זה מייצג עבורנו את טבלת ההאש (טיפוס מסוג HashTableElement [])

isDeleted – מערך המכיל מידע עבור כל צומת בטבלת ההאש האם נמחקה בעבר או לא (טיפוס מסוג boolean [])

mh – מייצג את משפחת פונקציות האש הרלוונטיות לטבלת ההאש (טיפוס מסוג ModHash, הרשאת גישה protected)

m – גודל טבלת ההאש (טיפוס מסוג int, הרשאת גישה protected)

בנאי המחלקה:

`OAHashTable(int m)`:

קיים למחלקה בנאי אחד המגדיר את טבלת האש לפי גודל קבוע מראש.

נוצרת טבלת האש בגודל m מתאים.

בנוסף, נוצר מערך המתוחזק על מנת לנטר עבור כל תא בטבלה האם **נמחק בעבר או לא**.

מתודות המחלקה:

`Find(long key)`:

מתודה המחפשת אובייקט בטבלת האש לפי מפתח.

סיבוכיות זמן ריצה: O(m)

המתודה מחזירה אובייקט מסוג HashTableElement במידה וקיים בטבלה.

אם לא, מחזירה null.

`Insert(HashTableElement hte) throws TableIsFullException,KeyAlreadyExistsException`:

מתודה המכניסה אובייקט מסוג HashTableElement לטבלת האש.

סיבוכיות זמן ריצה: O(m)

המתודה מנסה להכניס אובייקט לטבלה לפי פונקציית האש רלוונטית. כאשר נתקלה בתא הראשון הפנוי, האובייקט ייכנס לתא זה.

במידה והמפתח של האובייקט קיים כבר בטבלה (מזהה ייחודי) – ייזרק החריג: `KeyAlreadyExistsException`

כמו כן, במידה והטבלה מלאה, ייזרק החריג: `TableIsFullException`

`Delete(long key) throws KeyDoesntExistException`:

מתודה המוחקת אובייקט בטבלה (במידה וקיים) לפי המפתח שלו.

סיבוכיות זמן ריצה: O(m)

המתודה זורקת חריג אם האובייקט אינו קיים בטבלה: `KeyDoesntExistException`

אחרת, מחזירה את האיבר המתאים.

`abstract int Hash(long x, int i)`:

מתודה אבסטרקטית שמקבלת מפתח ואינדקס (מספרים שלמים).

מטרתה להחזיר מספר המותאם לפי פונקציית Hash הרלוונטית לטבלה.

מסמך תיעוד – IHashTable

<u>מתודות הממשק:</u>

מכיוון שזהו הממשק, המתודות אינן ממומשות ולכן נותרות ברמת החתימה **<u>בלבד</u>**.

```
Insert(HashTableElement hte) throws TableIsFullException,
                                   KeyAlreadyExistsException;
```

```
Delete(long key) throws KeyDoesntExistException
```

```
HashTableElement Find(long key)
```

כמו כן, קיימות 3 תת-מחלקות עבור **חריגים**:

1. `TableIsFullException extends Exception`
2. `KeyAlreadyExistsException extends Exception`
3. `KeyDoesntExistException extends Exception`

מסמך תיעוד – LPHashTable

מחלקה זו היא מחלקה היורשת מהמחלקה OAHashTable והיא אינה מרחיבה שדות מחלקה נוספים.

<u>בנאי המחלקה:</u>

`LPHashTable(int m, long p)`:

קיים למחלקה בנאי אחד המגדיר את טבלת האש לפי גודל קבוע מראש.

נוצרת טבלת האש בגודל m מתאים (על-ידי קריאה לבנאי של מחלקת האב).

בנוסף, נוצרת פונקציית Hash על-פי הפרמטרים m,p ממשפחת הפונקציות האוניברסליות (המבוטאות במחלקת ModHash).

<u>מתודות המחלקה:</u>

`Hash(long x, int i)`:

מתודה המחשבת אינדקס **<u>חוקי</u>** בטבלה לפי פונקציית Hash ספציפית לסוג טבלת LPHashTable.

סיבוכיות זמן ריצה: O(1)

המתודה מחזירה אינדקס **<u>חוקי</u>** בטבלה.

מסמך תיעוד – QPHashTable

מחלקה זו היא מחלקה היורשת מהמחלקה OAHashTable והיא אינה מרחיבה שדות מחלקה נוספים.

<u>בנאי המחלקה:</u>

`QPHashTable(int m, long p)`:

קיים למחלקה בנאי אחד המגדיר את טבלת האש לפי גודל קבוע מראש.

נוצרת טבלת האש בגודל m מתאים (על-ידי קריאה לבנאי של מחלקת האב).

בנוסף, נוצרת פונקציית Hash על-פי הפרמטרים m,p ממשפחת הפונקציות האוניברסליות (המבוטאות במחלקת ModHash).

<u>מתודות המחלקה:</u>

`Hash(long x, int i)`:

מתודה המחשבת אינדקס **<u>חוקי</u>** בטבלה לפי פונקציית Hash ספציפית לסוג טבלת QPHashTable.

סיבוכיות זמן ריצה: O(1)

המתודה מחזירה אינדקס **<u>חוקי</u>** בטבלה.

מסמך תיעוד – AQPHashTable

מחלקה זו היא מחלקה היורשת מהמחלקה OAHashTable והיא אינה מרחיבה שדות מחלקה נוספים.

<u>בנאי המחלקה:</u>

`AQPHashTable(int m, long p)`:

קיים למחלקה בנאי אחד המגדיר את טבלת האש לפי גודל קבוע מראש.

נוצרת טבלת האש בגודל m מתאים (על-ידי קריאה לבנאי של מחלקת האב).

בנוסף, נוצרת פונקציית Hash על-פי הפרמטרים m,p ממשפחת הפונקציות האוניברסליות (המבוטאות במחלקת ModHash).

<u>מתודות המחלקה:</u>

`Hash(long x, int i)`:

מתודה המחשבת אינדקס **<u>חוקי</u>** בטבלה לפי מודל פונקציית Hash ספציפי לסוג טבלת AQPHashTable.

סיבוכיות זמן ריצה: O(1)

המתודה מחזירה אינדקס **<u>חוקי</u>** בטבלה.

מסמך תיעוד – DoubleHashTable

מחלקה זו היא מחלקה היורשת מהמחלקה OAHashTable והיא אינה מרחיבה שדות מחלקה נוספים.

<u>בנאי המחלקה:</u>

`DoubleHashTable(int m, long p)`:

קיים למחלקה בנאי אחד המגדיר את טבלת האש לפי גודל קבוע מראש.

נוצרת טבלת האש בגודל m מתאים (על-ידי קריאה לבנאי של מחלקת האב).

בנוסף, נוצרת פונקציית Hash על-פי הפרמטרים m,p ממשפחת הפונקציות האוניברסליות (המבוטאות במחלקת ModHash).

<u>מתודות המחלקה:</u>

`Hash(long x, int i)`:

מתודה המחשבת אינדקס בטבלה לפי מודל פונקציית Hash ספציפי לסוג טבלת QPHashTable.

סיבוכיות זמן ריצה: O(1)

המתודה מחזירה אינדקס **<u>חוקי</u>** בטבלה.

מסמך תיעוד – ModHash

שדות המחלקה:

m – גודל טבלת ההאש (טיפוס מסוג int)

p – מספר ראשוני (טיפוס מסוג long)

params – מערך המכיל מידע עבור שני מספרים שלמים המוגרלים בעבור כל פונקציה השייכת למשפחת הפונקציות האוניברסליות (טיפוס מסוג long [])

בנאי המחלקה:

`ModHash(int m, long p)`:

קיים למחלקה בנאי אחד המאתחל את שדות המחלקה m,p.

כמו כן, יוצר מערך params ריק בעל שני תאים שיתעדכן עם הערכים המתאימים בעת קריאה לפונקציה GetFunc(int m, long p).

מתודות המחלקה:

`ModHash GetFunc(int m, long p)`:

מתודה סטטית היוצרת פונקציה חדשה השייכת למשפחת הפונקציות האוניברסליות.

סיבוכיות זמן ריצה: O(1)

מתודה סטטית המחזירה אובייקט מסוג ModHash לאחר שעודכנו כלל הפרמטרים המתאימים לפונקציה (בהתאם לשדות המחלקה).

`int Hash(long key)`:

מתודה המקבלת מפתח שעל-פיו תחושב פונקציית Hash.

סיבוכיות זמן ריצה: O(1)

מטרת המתודה היא להחזיר מספר שלם המחושב לפי פונקציית האש הרלוונטית לפרמטרי הפונקציה (פרמטרי אובייקט ModHash).

מסמך תיעוד – HashTableElement

שדות המחלקה:

key – מפתח האובייקט (טיפוס מסוג long)

value – ערך האובייקט (טיפוס מסוג long)

בנאי המחלקה:

`HashTableElement(long key, long value)`:

קיים למחלקה בנאי אחד המאתחל את שדות המחלקה.

מתודות המחלקה:

`long GetKey()`:

מתודה המחזירה את שדה המפתח של האובייקט.

סיבוכיות זמן ריצה: O(1)

`long GetValue()`:

מתודה המחזירה את שדה הערך של האובייקט.

סיבוכיות זמן ריצה: O(1)

חלק ב – ניסויים

3. בשאלה זו נשווה בין quadratic probing ל-alternating quadratic probing, ונבין את הקשר בין בחירת המקדמים של הביטוי הריבועי לבין התקינות של סדרת הבדיקה.

א. עבור המספר הראשוני $q = 6571$, חשבו אמפירית את גדלי הקבוצות:

$$Q_1 = \{i^2 \bmod q | 0 \le i < q\}$$
$$Q_2 = \{(-1)^i \cdot i^2 \bmod q | 0 \le i < q\}$$

<u>תשובה:</u>

$$\boxed{|Q_1| = 3286}$$

$$\boxed{|Q_2| = 6571}$$

ב. האם כל פעולות ההכנסה הושלמו בהצלחה, או שנזרקו חריגים? חזרו על התהליך הקודם עם AQPHashTable במקום QPHashTable. כיצד ניתן להסביר את השוני בין התוצאות?

<u>תשובה:</u>

<u>לא</u>, במחלקה QPHashTable נזרקו **חריגים** במהלך פעולות ההכנסה.

<u>החריג שנזרק הינו:</u> **TableIsFullException**

עבור המחלקה AQPHashTable כל פעולות ההכנסה הושלמו **<u>בהצלחה</u>**.

ההסבר לשוני בין התוצאות הינו על-פי עוצמת הקבוצות שחישבנו בסעיף א׳.

משמע, פונקציית האש של QPHashTable מבוססת על איברי הקבוצה Q_1.

פונקציית האש של AQPHashTable מבוססת על איברי הקבוצה Q_2.

עוצמת הקבוצה Q_1 הינה רק כמחצית מספר האיברים אותם אנו נדרשים להכניס לטבלה. ניתן להסביר זאת על-ידי כך, שכמות התאים אותם פונקציית האש של מחלקה QPHashTable יכולה למפות כל אובייקט <u>מוגבלת</u> למחצית גודל הטבלה (כעוצמת הקבוצה).

לפיכך, לאחר הכנסת כמחצית מאיברי הסדרה, מיפוי פונקציית האש יוביל לאינדקסים בטבלה המכילים אובייקטים **<u>לכל אורך סדרת החיפוש</u>**. ולכן, תיזרק שגיאה כי תיגמר סדרת החיפוש ללא מציאת מקום פנוי להכנסה.

אולם, בפועל המצב אינו כך וחצי מהטבלה עדיין ריקה.

החריג שייזרק הינו: **TableIsFullException**

ג. (בונוס) למדו על שאריות ריבועיות (quadratic residues) והסבירו את התופעה שבתרגיל זה. האם היא הייתה מתרחשת לכל ראשוני שהיינו בוחרים? מהו התנאי לקיום התופעה?

<u>תשובה:</u>

התופעה שמתקיימת בתרגיל זה הינה שעוצמת הקבוצה Q_1 היא בדיוק כמות האיברים המהווים את **השאריות הריבועיות** של המספר $q = 6571$ (**כולל 0**), כלומר:

$$|Q_1| = 3286$$

לעומת זאת, בקבוצה Q_2 שעוצמתה היא כמות כלל איברי הקבוצה, החישוב המתבצע הינו עבור האיברים המהווים את **השאריות הריבועיות** (כאשר **i זוגי** – כלומר, ביטוי חיובי) וגם עבור האיברים **שאינם השאריות הריבועיות** (כאשר **i אי-זוגי** – כלומר, ביטוי שלילי). ולכן:

$$|Q_2| = 6571$$

התנאי לקיום התופעה הינו:

קיום מספר ראשוני $\boxed{p > 2}$, אזי מתקיים: יש בדיוק $\frac{p-1}{2}$ שאריות ריבועיות ו- $\frac{p-1}{2}$ שאריות שאינן ריבועיות.

כלומר, התופעה מתרחשת עבור כל **מספר ראשוני גדול מ- 2**.

4. **א.** בשאלה זו נשווה בין המימושים השונים ל-open addressing. תעדו את זמן הריצה של כל סעיף בטבלה של אותו הסעיף. עבור כל אחד מהסעיפים, הוסיפו הסבר מילולי להבדלים בזמני הריצה בין סוגי הטבלאות. בשאלה זו לא אמורים להיזרק חריגים.

תשובה:

Class	*Running Time (sec)*
LPHashTable	*1.8383*
QPHashTable	*1.7524*
AQPHashTable	*2.1792*
DoubleHashTable	*1.9505*

ההבדלים בין זמני הריצה בין סוגי הטבלאות:

ישנם שני פרמטרים עליהם אנו מתבססים כדי לציין את ההבדלים בזמני הריצה:

1. **סיבוכיות פונקציית *Hash* מסוימת**
2. **פיזור יעיל יותר של אובייקטים בטבלה (לפי הגדרת הדגימה)**

אם כן, כאשר אנו ניגשים לאמוד את ההבדלים בין זמני הריצה, ראשית נתבונן בפונקציית *Hash* המיוצגת בכל טבלה.

מכיוון שבסעיף זה אנו נדרשים להכניס כמות איברים השווה **כמחצית מגודל הטבלה**, אנו מצפים כי זמני הריצה של הטבלאות: ***LPHashTable***, ***QPHashTable*** יהיו דומים באופן יחסי.

הסיבה לכך היא שההכנסה מתבצעת לטבלה ריקה וכאשר **גודל הטבלה** גדול ביחס לינארי **מכמות האיברים** אותה אנו נדרשים להכניס (במקרה זה, פי 2), אנו מצפים כי כל הכנסה תתרחש בסיבוכיות זמן *amortized - O(1)*.

לאחר מכן, נותר לנתח את סוגי הטבלאות הבאות: ***AQPHashTable***, ***DoubleHashTable***.

הכנסה לטבלה מסוג ***DoubleHashTable*** תגרום לעלייה בזמני הריצה.

זאת כיוון שאמנם הדגימה היא לינארית, אך עבור כל מפתח אותו אנו רוצים להכניס לטבלה יש לבצע קריאה לשתי פונקציות *Hash* שונות.

לפיכך, זמן העלייה הכולל של ההכנסות לטבלה זו יעלה ביחס לשתי הטבלאות הראשונות (אמנם לא עלייה דרסטית).

לבסוף, נותרנו עם טבלה מסוג ***AQPHashTable***.

מבחינת פונקציית ה- *Hash*, הדגימה של פונקציה זו דומה לדגימה של פונקציית *Hash* של טבלת ***QPHashTable***. ישנו הבדל קריטי מבחינת החישוב הראשוני של אינדקס ההכנסה, מכיוון שיתכן כי עבור חלק מהאינדקסים האי-זוגיים במהלך סדרת החיפוש, האינדקס המתקבל הינו שלילי.

לפיכך, הפונקציה תצטרך לבצע עדכון לאינדקס כדי להפכו לאינדקס חיובי בטווחי הטבלה (כלומר, אינדקס חוקי).

לכן, סיבוכיות זמן הריצה של ההכנסה לסוג טבלה זה תעלה ביחס לאחרות (עקב עדכון חישוב אינדקס).

ולסיכום, אנו מצפים כי דירוג זמני ההכנסה בין הטבלאות השונות יהיה:

1. *Quadratic Probing Hash Table*
2. *Linear Probing Hash Table*
3. *Double Hash Table*
4. *Alternating Quadratic Probing Hash Table*

ללא הבדל משמעותי בין מקומות 1-2

ב. חזרו על הסעיף הקודם, אבל כש-$n = \left\lfloor \frac{19m}{20} \right\rfloor$. אין לבצע סעיף זה עבור QPHashTable (נמקו מדוע). האם ההבדל בביצועים לעומת הסעיף הקודם שונה בהתאם לסוג הטבלה? נמקו.

Class	*Running Time (sec)*
LPHashTable	*5.379*
AQPHashTable	*7.3125*
DoubleHashTable	*4.3898*

תשובה:

ראשית, **אין** לבצע סעיף זה עבור QPHashTable מכיוון שאנו מצפים לזריקת חריג מסוג: **TableIsFullException**. הסיבה לכך דומה לחריגה שנזרקה בשאלה 3, סעיף ב.

עוצמת הקבוצה שנגזרת מהגדרת פונקציית Hash של המחלקה QPHashTable קטנה ממספר האיברים אותם אנו נדרשים להכניס לטבלה בסעיף זה. ניתן להסביר זאת על-ידי כך, שכמות התאים אותם פונקציית Hash יכולה למפות כל אובייקט מוגבלת.

לפיכך, לאחר הכנסת חלק מאיברי הסדרה, מיפוי פונקציית האש יוביל לאינדקסים בטבלה המכילים אובייקטים קיימים **לכל אורך סדרת החיפוש**. ולכן, תיזרק שגיאה כי תיגמר סדרת החיפוש ללא מציאת מקום פנוי להכנסה.

אולם, בפועל המצב אינו כך וישנם תאים ריקים בטבלה.

כן, ישנו הבדל בביצועים בין סעיף זה לבין הסעיף הקודם.

ההבדל בין זמני כל פונקציה בין סעיף זה לסעיף הקודם נובע עקב גדילת מספר האיברים אותם אנו נדרשים להכניס לטבלה.

במצב זה, כאשר כמות האיברים אותה אנו נדרשים להכניס לטבלה קרובה יחסית לגודל הטבלה עצמה, בא לידי ביטוי ההבדל באופן הדגימה של כל פונקציה.

שיטת **Double Hashing** הינה **המהירה ביותר** בתרחיש זה. הסיבה לכך היא עקב העובדה שהפיזור הינו רנדומלי ביחס לטבלה (על-פי מודל הדגימה של פונקציית Hash) ובנוסף, סיבוכיות הזמן של כל הכנסה היא **לינארית** – זאת כיוון שהקפיצות בין כל איבר לאיבר העוקב לו בסדרת החיפוש הינן **בקבוע**.

בשיטת **Linear Probing**, לאחר יצירת אינדקס התחלתי לטבלת החיפוש, הקפיצות הינן קבועות בין כל איבר לאיבר העוקב לו בסדרת החיפוש. לפיכך, אנו מצפים כי במהלך הכנסה של מספר איברים רב, ייווצרו התנגשויות עקב הסמיכות בין האיברים בטבלה.

לעומת שתי השיטות לעיל, שיטת **Alternative Quadratic Probing** הינה **האיטית ביותר וגם הכי פחות יעילה**. הסיבה לכך נובעת ממודל הדגימה בסוג טבלה זה.

ראשית, הדגימה מתבצעת לפי קפיצה **ריבועית** בין כל איבר בסדרת החיפוש (לעומת קפיצה לינארית בשתי השיטות האחרות). כביכול אנו מצפים לפיזור יעיל יותר של פונקציית Hash.

אולם, עקב החישוב המורכב יותר של דגימה ריבועית, אנו מצפים כי חישוב פר אינדקס מסוים בסדרת החיפוש יהיה ארוך יותר מבחינת סיבוכיות הזמן.

כמו כן, עקב שינויי הסימן בין כל דגימה לדגימה העוקבת לה (חיובי/שלילי), חישוב האינדקס נהיה מסובך יותר וישנו צורך תדיר לעדכן את האינדקס כך שיהווה אינדקס חוקי בטבלת Hash.

לפיכך, ההבדל בזמנים **משמעותי** – זאת עקב סיבוכיות זמן גבוהה יותר של פונקציית Hash.

5. בשאלה זו נחקור את השפעת מחיקת איברים ב-open addressing על סיבוכיות הזמן של פעולות על הטבלה. השוו את זמן ביצוע 3 האיטרציות הראשונות לזמן ביצוע 3 האיטרציות האחרונות. האם קיים הבדל? אם כן, הסבירו מדוע.

תשובה:

Iterations	*Running Time (sec)*
First 3 iterations	*13.727*
Last 3 iterations	*56.964*

קיים הבדל בתוצאות 3 האיטרציות הראשונות ל- 3 האיטרציות האחרונות.

הסיבה להבדל נובעת משתי סיבות עיקריות:

1. **זמני האיטרציה הראשונה לעומת זמני שאר האיטרציות**
2. **עליית כמות האיברים המחוקים בטבלה**

1. באיטרציה הראשונה, הכנסת האיברים הינה לטבלה ריקה (**ללא** איברים מחוקים). ואילו בשאר האיטרציות ההכנסה של כל איבר לטבלה מושפעת מכמות האיברים המחוקים בכל נקודת זמן של הכנסה. זאת כיוון שעל-פי האלגוריתם של פונקציית מחיקה, יידרש מעבר ממושך יותר בסדרת החיפוש כאשר נתקלים בתא שבו היה איבר מחוק.

   לפיכך, **בוודאות** יידרש זמן ארוך יותר ל-3 האיטרציות האחרונות.

2. כמו כן, קיימת **עלייה** בזמני כל איטרציה (כלומר, סדרת זמני 6 האיטרציות הינה סדרה מונוטונית עולה). הסיבה לכך נובעת מעליית כמות האיברים "המחוקים" בטבלה (במימוש שלנו: isDeleted[i] = true).

עקב העובדה כי סדרת ההכנסות היא של איברים המוגרלים רנדומלית, בכל איטרציה ייתכן כי ימופו חלק מהאיברים לתאים מחוקים וחלק לתאים ריקים (״null טהור״).

כלומר, ככל שמספר האיטרציות יגדל, באופן ישיר גם כמות האיברים המחוקים בטבלה תגדל – עד אשר כל האיברים בטבלה ימופו באיזושהי איטרציה N ולכן, מאותה איטרציה ואילך (n>N), זמני הריצה אמורים להישאר דומים פחות או יותר.