מסמך תיעוד ומדידות

המחלקה IHashTable - מנשק

פונקציות

```
public void Insert(HashTableElement hte) throws TableIsFullException,
KeyAlreadyExistsException
```

פונקציה המכניסה איבר לטבלת ה-hash

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

פונקציה המוחקת איבר מטבלת ה-hash

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

פונקציה המחפשת איבר בטבלת ה-hash לפי המפתח

מחלקות פנימיות

```
public class TableIsFullException extends Exception
```

חריג המודיע על טבלה מלאה

```
public class KeyAlreadyExistsException extends Exception
```

חריג המודיע על מפתח שכבר נמצא בטבלה

```
public class KeyDoesntExistException extends Exception
```

חריג המודיע על מפתח שאינו קיים בטבלה

המחלקה HashTableElement

שדות

```
private long key
private long value
```

פונקציות

```
public HashTableElement(long key, long value)
```

בנאי של איבר חדשה מסוג HashTableElement

```
public long GetKey()
```

הפונקציה מחזירה את המפתח של המופע

```
public long GetValue()
```

הפונקציה מחזירה את הערך של המופע

המחלקה OAHashTable – מחלקה אבסטרקטית

שדות

```
private static final HashTableElement DELETED = new HashTableElement(0,0)
```

אלמנט סטטי וסופי בטבלה המציין תא שנמחק ממנו איבר.

```
protected HashTableElement [] table
```

מערך המאחסן את איברי הטבלה.

פונקציות

```
public OAHashTable(int m)
```

בנאי של אובייקט חדש מסוג OAHashTable, מאתחל מערך בגודל m.

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

מימוש פונקציית החיפוש שהוגדרה במנשק. הפוקנציה מקבלת מפתח ומחזירה את האיבר בטבלה עם המפתח הנתון. מחזירה null אם המפתח לא נמצא בטבלה.

```
public void Insert(HashTableElement hte) throws
TableIsFullException,KeyAlreadyExistsException
```

מימוש פונקציית ההכנסה שהוגדרה במנשק. הפונקציה מקבלת איבר HashTableElement המכיל מפתח וערך ומכניסה לטבלה את האיבר במקום הפנוי המתאים. זורקת חריג אם הטבלה מלאה או אם המפתח כבר קיים בטבלה.

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

מימוש של פונקציית המחיקה שהוגדרה במנשק. הפונקציה מקבלת מפתח, מחפשת את מיקומו בטבלה ומוחקת אותו. האיבר מוחלף באיבר Deleted. זורקת חריג אם המפתח לא נמצא בטבלה.

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

פונקציה אבסטרקטית שמחשבת את האיבר ה-i בסדרת הבדיקה של איבר בעל מפתח x.

המחלקה ModHash

שדות

```
long a
long b
int m
long p
```

פונקציות

```
public ModHash(long a, long b, int m, long p)
```

בנאי אשר יוצר אובייקט ModHash חדש בעל שדות a,b,p,m

```
public static ModHash GetFunc(int m, long p)
```

הפונקציה קוראת לבנאי של ModHash עם ערכי a,b אקראיים ומחזירה אובייקט חדש של פונקציית האש.

```
public int Hash(long key)
```

הפונקציה מקבלת מפתח ומחזירה את פתרון המשוואה ((ax + b) mod p) mod m בהתאם לערכי המופע

```
public int DHash(long key)
```

הפונקציה מחשבת את ערך ה-hash עבור h2 - פונקציית הצעד בשיטת ה-DoubleHashing. במקרה זה, תוצאת החישוב בהכרח תהיה מספר שזר ל-m.

```
public static long abGenrator(long leftLimit, long rightLimit)
```

הפונקציה מגרילה באופן אקראי ערכי a ו-b בטווח חוקי, אשר יכללו בבניית פונקציה ממשפחת הפונקציות המודולריות האוניברסלית.

המחלקה LPHashTable

שדות

```
ModHash hashFunc
```

פונקציות

```
public LPHashTable(int m, long p)
```

בנאי אשר יוצר אובייקט LPHashTable חדש בהתאם לערכי m ו-p שהתקבלו עם פונקציה ממשפחת הפונקציות המודולריות האוניברסלית.

```
public int Hash(long x, int i)
```

הפונקציה מקבלת מפתח ומספר איטרציה i ומחזירה אינדקס בטבלה - פתרון המשוואה (h(x) + i) mod m כאשר h(x) הוא הערך המוחזר מקריאה לפונקציית ה-hash.

המחלקה QPHashTable

שדות

```
ModHash hashFunc
```

פונקציות

```
public QPHashTable(int m, long p)
```

בנאי אשר יוצר אובייקט QPHashTable חדש בהתאם לערכי m ו-p שהתקבלו עם פונקציה ממשפחת הפונקציות המודולריות האוניברסלית.

```
public int Hash(long x, int i)
```

הפונקציה מקבלת מפתח ומספר איטרציה i ומחזירה אינדקס בטבלה - פתרון המשוואה (h(x) + i^2) mod m כאשר h(x) הוא הערך המוחזר מקריאה לפונקציית ה-hash.

המחלקה AQPHashTable

שדות

```
ModHash hashFunc
```

פונקציות

```
public AQPHashTable(int m, long p)
```

בנאי אשר יוצר אובייקט AQPHashTable חדש בהתאם לערכי m ו-p שהתקבלו עם פונקציה ממשפחת הפונקציות המודולריות האוניברסלית.

```
public int Hash(long x, int i)
```

הפונקציה מקבלת מפתח ומספר איטרציה i ומחזירה אינדקס בטבלה - פתרון המשוואה (h(x) + (-1)^i * i^2) mod m כאשר h(x) הוא הערך המוחזר מקריאה לפונקציית ה-hash.

המחלקה DoubleHashTable

שדות

```
ModHash hashFunc1
ModHash hashFunc2
```

פונקציות

```
public DoubleHashTable(int m, long p)
```

בנאי אשר יוצר אובייקט DoubleHashTable חדש בהתאם לערכי m ו-p שהתקבלו עם פונקציה ממשפחת הפונקציות המודולריות האוניברסלית.

```
public int Hash(long x, int i)
```

הפונקציה מקבלת מפתח ומספר איטרציה i ומחזירה אינדקס בטבלה - פתרון המשוואה (h1(x) + i * h2(x)) mod m כאשר h1(x) ו-h2(x) הם הערכים המוחזרים מקריאה לפונקציות ה-hash של hashFunc1 ו-hashFunc2 בהתאמה.

מדידות

שאלה 3

סעיף א

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

סעיף ב

עבור QPHashTable נזרק חריג שמציין כי הטבלה מלאה טרם הכנסת כל האיברים לטבלה.

עבור AQPHashTable סיימנו לבצע את כל סדרת ההכנסות.

לשיטת ה-Quadric Probing חיסרון בכך שסדרת הבדיקות שהיא מבצעת על איבר מסוים לשם הכנסתו היא לא מלאה. כלומר, סדרת הבדיקות לא מחסה את כל הטבלה ולא נעבור על כך הטבלה בזמן הבדיקה (ודוגמה לכך ראינו בכמות האיברים השונים שקיבלנו בקבוצה Q1). לכן נזרק החריג שציין כי הטבלה מלאה על אף שקיימים בה תאים פנויים.

לעומת זאת, שיטת ה-Alternating Quadric Probing יכולה לבצע סדרת בדיקות שעוברת על כל התאים בטבלה בהכנסת איבר עבור טבלה שגודלה הוא מספר ראשוני המקיים שארית 3 בחלוקה ל-4. בשיטה זו נקבל את קבוצת כל השאריות האפשריות של אותו מספר ראשוני שבאמצעותן נעבור על כל הטבלה ונבצע סדרת בדיקות מלאה. תכונה זו של Alternating Quadric Probing הודגמה בסעיף א' שם ראינו כי קיבלנו את כל שאריות החלוקה האפשריות עבור המספר הראשוני q=6571. לכן, בשיטה זו עבור טבלה בגודל q נקבל זריקת חריג רק לאחר שכל תאי הטבלה אכן תפוסים.

סעיף ג - בונוס

התופעה בשאלה נובעת משאריות ריבועיות. כאשר סימנו של האיבר המושווה מתחלף וגם מספר התאים בטבלה p הוא ראשוני שחופף לתוצאה 3 mod4 (השארית שווה ל-3, כמו במקרה ש-6571 mod4 = 3), נקבל שהסדרה המתקבלת על-ידי חישוב $((-1)^i \cdot i^2) \bmod m$ כך ש-$0 \le i < p$ מהווה פרמוטציה של הקבוצה $\{n \mid 0 \le n < p\}$. לכן, כל עוד קיים תא פנוי בטבלה הוא יימצא בשיטת Alternating Quadric Probing.

שאלה 4

סעיף א

Class	*Running Time (nanoseconds)*
LPHashTable	1445805600
QPHashTable	3227518433
AQPHashTable	3176650667
DoubleHashTable	1650336867

ניתן לראות שטבלאות ה-Hash בשיטות ה-Linear Probing ו-Double Hashing היו המהירות ביותר (בערך פי 2). ניתן להסביר ממצאים אלו בכך ששיטת ה-Linear Probing מתאפיינת בקבועים נמוכים ביחס לאחרות בעוד ש-Double Hashing מתאפיינת בסדרות חיפוש מגוונות יותר ובכך מקטינות את כמות הבדיקות שצריך לבצע.

שיטת ה-Linear Probing מתאפיינת בקבועים נמוכים. אמנם היא סובלת מבעיית ההצטברות הראשונית (יכולים להיווצר רצפים ארוכים של תאים תפוסים שיגדילו את הסיכוי להיכנס לתא מסוים), אך חישוב המיקום של האיבר הבא בסדרת ההכנסות הוא קל ומהיר (חישוב השארית העוקבת). בנוסף, יש מעט מאד cache misses ביחס לשיטות האחרות וגם מעט פניות לזיכרון (כי סדרת החיפוש של איבר יושבת באופן רציף בזיכרון בתאים סמוכים של המערך).

בשיטת ה-Double Hashing חישוב הצעד הבא יקר ביחס לשאר השיטות, אך בניגוד אליהן סדרת הבדיקה לא תלויה רק במיקום הראשוני אלא גם במפתח (על-פיו מחושבת גם פונקציית הצעד ולא רק פונקציית הבסיס). שיטה זו מקטינה את בעיות ההצטברות הראשונית והמשנית וכך כמות הפניות לתאים בזיכרון שיש לבצע בעת הכנסה של שני איברים שונים היא נמוכה יותר בתוחלת ביחס לשאר השיטות.

שאר השיטות שמתבססות על Quadric Probing לא זוכות לקבועים נמוכים כמו של Linear Probing (החישוב יותר מסובך מאשר הוספה של 1 ויש לבצע קפיצות ממיקום אחד בזיכרון לאחר). כמו כן, סדרת הבדיקות שלהן נקבעת על-פי מיקום ההכנסה הראשוני (בניגוד ל-Double Hashing). כלומר, אם לשני מפתחות שונים יתקבל אותו הערך בפונקציית ה-hash אז לשניהם תהיה בדיוק אותה סדרת בדיקה (בעיית ההצטברות המשנית).

סעיף ב

Class	*Running Time (nanoseconds)*
LPHashTable	6098114633
AQPHashTable	11178840133
DoubleHashTable	4536235000

לא נבצע סעיף זה עבור QPHashTable שכן בשיטה זו סדרת הבדיקות היא לא מלאה. לכן, כאשר נכניס הרבה איברים לטבלה סביר שיזרק חריג המודיע כי הטבלה מלאה. בדומה לסעיף א', Alternating Quadric Probing הייתה לשיטה הכי איטית מאותן הסיבות שכבר ציינו ומכך שבעיית ההצטברות המשנית נעשית חמורה יותר כאשר הטבלה מתמלאת – יש יותר תאים תפוסים ויש סיכוי רב יותר ששני מפתחות יקבלו את אותה התוצאה בפונקציית ה-hash ויבצעו את אותה סדרת בדיקות טרם הכנסתם.

יחד עם זאת, בבדיקה זו ניכר יתרון לשיטת ה-Double Hashing שהייתה לשיטה המהירה ביותר. ההבדל נבע מכך שבסעיף א' שמרנו על $\alpha \leq 0.5$. במצב זה הפיזור בטבלה בשיטת ה-Linear Probing יחסית אחיד ולא סביר לפגוש רצפים ארוכים מידי של תאים תפוסים שיעטו את תהליך ההכנסה. בסעיף ב' הטבלה כבר כמעט מלאה ולכן ה-Load Factor גדול יותר. במצב זה יש בטבלה כבר הרבה איברים וסביר שפונקציית ה-hash תוביל אותנו לתא תפוס שלאחריו רצף של תאים תפוסים בזיכרון שנצטרך לעבור עליהם עד שנגיע לתא פנוי להכנסה.

שאלה 5

Iteration	*Running Time (nanoseconds)*
First 3 iterations	19144828300
First 3 iterations	61546302100

ניתן לראות כי זמן הריצה של ממוצע האיטרציות האחרונות גדל בערך פי 3 בהשוואה לממוצע האיטרציות הראשונות. הבדל זה נובע מכך שעבדנו על אותה הטבלה בכל האיטרציות. ככל שהתקדמנו במספר האיטרציות כך נוספו איברים חדשים מסוג deleted שתופסים מקום במערך. איברים אילו נבדלים מתאים ריקים בטבלה (NULL) שלא נמחקו מהם איברים בעבר. בניגוד לתא ריק המהווה פעולה טרמינלית עבור חיפוש האיבר לשם הכנסה או מחיקה, תא המאוחסן באיבר מסוג Deleted לא בהכרח מציין כי סדרת הפניות (probing) לאותו איבר הגיעה לסיומה. כך, ככל שנתקדם במספר האיטרציות שנבצע, יתווספו עוד איברי Deleted חדשים לטבלה, ומכאן שנצטרך לבצע סדרת ארוכה יותר של probings על-מנת להיות בטוחים כי האיבר שאנחנו רוצים להכניס לטבלה לא הוכנס בעבר ורק אז להכניסו למקום המתאים (שיכול להיות אחד מה-probing הקודמים). לכן, בין איטרציה אחת לשנייה כמות הפניות שיש לבצע עולה, ומכאן שזמן החישוב עבור סדרת הפעולות שיש לבצע התארך.