תרגיל 5- מבני נתונים 67109:

שם: דן קורנפלד

חלק 1:

גיבוב אוניברסלי:

שאלה 1:

- **הגדרה- גיבוב אוניברסלי**: נגדיר H משפחת פונקציות $h: U \to [m]$ נאמר כי H אוניברסלית אם עבור בחירה אקראית של h מ-H מתקיים כי **לכל** $x, y \in U$ $P(h(x) = h(y)) \leqslant \frac{1}{m}$
- נוכיח שמשפחת הפונקציות H אכן אוניברסלית: m גודל טבלת הגיבוב, $p = m$ מספר ראשוני, $k \in \mathbb{N}$, $k \in [0, p-1]$. יהי $a \in \{-(p-1),..,0\}$; $b, c \in \{0,..,p-1\}$, נגדיר $h_{a,b,c}(k) = \left(\left(ak^2 + bk + c\right) mod\ p\right)$, ונגדיר $H = \left\{h_{a,b,c} \middle| a \in \{-(p-1),..,0\}\ ;\ b, c \in \{0,..,p-1\}\right\}$.
- עבור $x, y \in U$ כך ש $x \neq y$. עבור $h_{a,b,c}$ כלשהו מ H:

$$P(h_{a,b,c}(x) = h_{a,b,c}(y)) = P\left(\left(ax^2 + bx + c = ay^2 + by + c\right) mod\ p\right) =$$

$$= P\left(\left(a\left(x^2 - y^2\right) + b(x - y) = 0\right) mod\ p\right) = P((a((x+y)(x-y)) + b(x-y) = 0) mod\ p) =$$

$$P((a(x+y)(x-y) = b(y-x)) mod\ p) = P((-a(x+y) = b) mod\ p)$$

- כלומר הסיכוי ש$-a(x+y) = b$ לאחר ביצוע פעולת $mod(p)$ היא 1 מתוך p אפשרויות, כלומר $\frac{1}{p}$
- ולכן, $P(h_{a,b,c}(x) = h_{a,b,c}(y)) = \frac{1}{p} \leqslant \frac{1}{m}$ כמבוקש

שאלה 2:

- נסמן $U = \{000, 001, ..., 999\}$. $m = 10$ גודל טבלת הגיבוב, ויהי $a \in \mathbb{N}$, $a \in \{1, 2,.., 9\}$. נסמן $h_a(x)$ הספרה הימנית ביותר של $a \cdot x$
- הפרכה: עבור $h_a(x)$ כלשהו, נבחר $x = 000, y = 100$, לכל a, $P(h_a(x) = h_a(y) = 0) > \frac{1}{10}$, מכיוון שהספרה הימנית תמיד תהיה 0
- כלומר אם ניקח $a = 1$, $P(h_1(x) = h_1(y)) = \frac{1}{9} > \frac{1}{10}$, כלומר מצאנו פונקציה h אחת שמקבלת מפתחות שונים ומחזירה ערך זהה, וההסתברות לבחור פונקציה כזו היא $\frac{1}{9}$, ולכן המשפחה H לא אוניברסלית.

תרגיל 5- מבני נתונים 67109:

שם: דן קורנפלד

חלק 2:

גיבוב K-אוניברסלי ו -$Open\ addresing$:

שאלה 1:

1) סעיף א:

הגדרה ל-1 אוניברסלית: H היא 1 אוניברסלית אם לכל $a \in \{0,..,m-1\}$ ולכל $x \in U$

$P(h(x) = a) \leqslant \frac{1}{m}$.

הגדרה ל-2 אוניברסלית: H היא 2 אוניברסלית אם לכל $a_1, a_2 \in \{0,..,m-1\}$ ולכל $x_1, x_2 \in U$

$P(h(x_1) = a_1 \wedge h(x_2) = a_2) \leqslant \frac{1}{m^2}$.

הפרכה:

נגדיר: $m = 2, U = \{x, y\}\ , H = \{h_1, h_2\}$.

$$h_1 :\ h_1(x) = 1,\ h_1(y) = 1\ ;\ h_2 : h_2(x) = 2,\ h_2(y) = 2$$

$$P(h(x) = 1) = \frac{1}{2} \leqslant \frac{1}{2} = \frac{1}{m},\ h\ is\ 1-univeral$$

אבל כאשר נרצה להוכיח שH היא לא 2 אוניברסלית:

ראינו בתרגול ש 2 אוניברסלי $\Leftarrow$ אוניברסלי. כלומר, לא אוניברסלי $\Leftarrow$ לא 2 אוניברסלי. לכן, אם נוכיח ש H לא אוניברסלית התנאי לא יתקיים. $P(h(x) = h(y)) = 1 > \frac{1}{m} = \frac{1}{2}$, לכן H לא אוניברסלית, ולכן היא גם לא 2 אוניברסלית, ולכן 1 אוניברסלי לא גורר של H היא 2 אוניברסלית.

2) סעיף ב:

הפרכה:

נגדיר: $m = 2, U = \{x, y\}\ , H = \{h\}$.

$h(x) = 1,\ h(y) = 2$. אכן H אוניברסלית: $P(h(x) = h(y)) = 0 < \frac{1}{2} = \frac{1}{m}$, אך H לא 2-אוניברסלית, מכיוון ש $P(h(x) = 1 \wedge h(y) = 2) = 1 > \frac{1}{m^2} = \frac{1}{4}$

שאלה 2:

	0	1	2	3	4	5	6	7	8	9	10	# collisions
Linear probing	77	54	57	23	74				63	21	76	11
Quadratic Probing	77	31	57		23			74	63	54	76	6
Double Hashing	77	23	57		54			74	63	21	76	4

ככל שהפעולה שאנחנו מבצעים על כל מספר יותר מורכבת, מספר ההתנגשויות קטן יותר. בפיזור לינארי יש רצפים של מספרים ולכן יש מספר התנגשויות הגבוה ביותר, ואילו בביצוע hash כפול יש הכי קצת התנגשויות.

תרגיל 5- מבני נתונים 67109:

שם: דן קורנפלד

חלק 3:

שימושים של $Hash - Tables$:

שאלה 1:

אלגוריתם עבור $PrintFrequencies(A)$:

1. ניצור $Hash\ Table$ שמוגדר כמערך, כך שכל תא יצביע לתת מערך שיכיל את הערך a מהמערך המקורי, ואת מספר המופעים (שיאותחלו בהמשך ל-1). בשלב הראשוני הטבלה תהיה ריקה.
2. נעבור על המערך A ונבדוק לכל $a \in A$ אם a כבר נמצא בטבלה, נוסיף 1 למספר המופעים שלו, ואם הוא לא נמצא הכנס תת מערך חדש ל-$Hash - Table$, ואתחל את מספר המופעים ל-1.
3. לאחר מכן, נעבור על ה-$Hash - Table$ מההתחלה לסוף ונדפיס את כל הצמדים (a, i_a). מותר לבצע זאת באופן שיטתי, מכיוון שאנחנו מגדירים מאחורי הקלעים את הטבלה להיות מערך, וידוע לנו שניתן לעבור על מערך באופן שיטתי, ולכן הפעולה הזו חוקית.

הסבר סיבוכיות זמן ריצה של האלגוריתם עבור $PrintFrequencies(A)$:

1. יצירת $Hash - Table$ ריק $O(1)$
2. השמת כל איבר מהמערך A לטבלת ה-$Hash - Table$ הוא $O(1)$ בתוחלת:
   (a) find עבור הבדיקה אם האיבר נמצא כבר במערך הוא $O(1)$ בתוחלת
   (b) הכנסת איבר למערך הוא $O(1)$ בתוחלת
   (c) עדכון מספר מופעים הוא $O(1)$

   לכן, מעבר על המערך A והשמת כל האיברים ב A יהיה בסיבוכיות זמן של $O(n)$.
3. מעבר על ה-$Hash - Table$ מההתחלה לסוף והדפסת כל הצמדים היא סיבוכיות זמן של $O(n)$.
4. לכן $O(1) + O(n) + O(n) = O(n)$ כמבוקש.

שאלה 2:

למדנו על $Hash - Table$ שמאפשר לבצע על משתנים (כמו מספרים) פעולת $Hash$, כך שערך החזרת הפעולה תהיה האינדקס שבו המשתנה יהיה בטבלה. פעולת ה $Hash$ בסיבוכיות זמן של $O(1)$.

נגדיר מבנה נתונים חדש ונקרא לו $Oldest\ Hash\ Table$, אשר יכיל 2 איברים: $Hash\ table$ בגודל $2n$ (נאמר בפורום ש-n הוא גודל הקלט שצריך להתייחס אליו, נרצה שגודל המערך יהיה קצת יותר גדול ממספר האיברים מכיוון שכאשר המערך כמעט מלא\ מלא הוא פחות משמעותי משמעותית) ונשתמש ב $Open\ addressing$ כדי למנוע התנגשויות, בשיטת $Double - Hashing$. ומבנה נתונים $Stack$ שנקרא לו oldest.

- ❖ עבור פונקציית $find(x)$- נרצה לבצע את פעולת ה-Hash (שמתבצעת ב-$O(1)$) ולבדוק אם הערך נמצא בטבלה, או לא. אם המיקום בטבלה ריק, האיבר לא נמצא בטבלה. הגישה לתא במערך בשיטת $Open\ addressing$ היא $O(1)$ בתוחלת.
- ❖ עבור פונקציית $insert(x)$- תחילה נבדוק באמצעות $find(x)$ אם הערך כבר נמצא, אם לא, נבצע את פעולת ה-$Hash$ עבור ה-x החדש ונכניס את האיבר החדש במקום המתאים, ונוסיף את האינדקס של האיבר שהכנסנו, ל-$Stack\ (oldest)$, כלומר ביצוע הפעולה ב $O(1)$ בתוחלת

  ★ - נשים לב ש$Double - Hashing$ יכול למפות את אותו התא ל2 איברים, לכן לפני ההכנסה נבדוק אם התא ריק, אם הוא לא ריק נבצע את פעולת ה- $Open\ addressing$, נוסיף 1 ל-i בפעולת ה-$Double - Hashing$, נקבל תא אחר ונחזור על הבדיקה אם התא ריק עד שנמצא תא ריק.
- ❖ עבור פונקציית $delete\_oldest()$- ניגש לראש ה-$Stack$, נבדוק אם היא לא ריקה, במידה והיא לא ריקה נוציא את האיבר האחרון מראש ה-$Stack$, ועם האיבר שהוצאנו ניגש לאינדקס בו אכסנו את האיבר האחרון ונמחק אותו מה $Hash - Table$. בהוצאת האיבר האחרון, האיבר שהוכנס לפניו יהפוך להיות האיבר האחרון, כך ניתן לבצע את פעולת המחיקה מספר פעמים ברצף. כלומר ביצוע הפעולה ב-$O(1)$

שאלה 3:

סעיף א:

נשתמש ב-$Hash - Table$ ונשתמש ב$Universal\ hashing$, כך שגודל הטבלה תהיה $2n$ (נרצה שגודל המערך יהיה קצת יותר גדול ממספר האיברים מכיוון שכאשר המערך כמעט מלא\ מלא הוא פחות משמעותי משמעותית), וכל איבר בטבלה יכיל את המפתח, ואת הערך שנרצה לשמור.

(לא ניתן להשתמש ב $Perfect\ Hashing$, מכיוון שהמפתחות לא ידועים מראש)

- ❖ עבור פונקציית $find(key)$- נשתמש ב-$Hash\ function$ עבור ה-key, נבדוק אם יש איבר באינדקס שאנחנו מקבלים, אם כן נחזיר את ה-$value$, אם אין ערך נחזיר $null$. מכיוון ש $Hash\ function$ פועלת בסיבוכיות זמן של $O(1)$ בתוחלת, וגישה לזיכרון גם ב$O(1)$, אכן הפונקציה בסיבוכיות זמן של $O(1)$ בתוחלת.
- ❖ עבור פונקציית $increment(key)$- בדומה ל-$find(key)$ נחפש את מיקום המפתח בטבלה באמצעות פונקציית ה-$hash$, לאחר מציאת התא, נעלה את הערך ב-1. פעולה זו גם כן בסיבוכיות זמן של $O(1)$ מכיוון שאנו משתמשים בפונקציית ה-$Hash$ שמחשבת את האינדקס בסיבוכיות זמן של $O(1)$, והגישה לתא המתאים הוא $O(1)$ בתוחלת.
- ❖ עבור פונקציית $set(key, new\_value)$- בדומה ל-$find(key)$ נחפש את מיקום המפתח בטבלה באמצעות פונקציית ה-$hash$, לאחר מציאת התא, נעדכן את הערך החדש בתא, לערך הרצוי. פעולה זו גם כן בסיבוכיות זמן של $O(1)$ מכיוון שאנו משתמשים בפונקציית ה-$Hash$ שמחשבת את האינדקס בסיבוכיות זמן של $O(1)$, כדי להגיע לתא המתאים הוא $O(1)$ בתוחלת.
- ❖ עבור בניית מבנה הנתונים- פעולת ההכנסה בודקת אם המפתח כבר נמצא במבנה הנתונים בסיבוכיות זמן של $O(1)$ בתוחלת, אם לא, מכניסה את הערכים הרצויים לטבלה. הכנסת n ערכים כמבוקש. $\underbrace{O(n)}_{\text{בתוחלת}} = n \cdot \underbrace{O(1)}_{\text{בתוחלת}}$

סעיף ב:

נשתמש ב-$Hash - Table$ ונשתמש ב$Universal\ hashing$, כך שגודל הטבלה תהיה $2n$ (נרצה שגודל המערך יהיה קצת יותר גדול ממספר האיברים מכיוון שכאשר המערך כמעט מלא\ מלא הוא פחות משמעותי משמעותית), וכל איבר בטבלה יכיל את המפתח, ואת הערך שנרצה לשמור. כמו כן, בנוסף לטבלה, נשמור משתנה מספרי נוסף בשם to_add שמאותחל תחילה להיות 0. שמירת המשתנה מוסיפה שטח בגודל $O(1)$.
(לא ניתן להשתמש ב $Perfect\ Hashing$, מכיוון שהמפתחות לא ידועים מראש)

- עבור פונקציית $find(key)$- נשתמש ב-$Hash\ function$ עבור ה-key, נבדוק אם יש איבר באינדקס שאנחנו מקבלים, אם כן נחזיר את ה-$value + to\_add$, אם אין ערך נחזיר $null$. מכיוון ש $Hash\ function$ פועלת בסיבוכיות זמן של $O(1)$, ובפרט כדי להגיע לתא המתאים הוא $O(1)$ בתוחלת, וגישה לזיכרון גם ב$O(1)$, אכן הפונקציה בסיבוכיות זמן של $O(1)$ בתוחלת.
- עבור פונקציית $increment(key)$- בדומה ל-$find(key)$ נחפש את מיקום המפתח בטבלה באמצעות פונקציית ה-$hash$, לאחר מציאת התא, נעלה את הערך ב-1. פעולה זו גם כן בסיבוכיות זמן של $O(1)$ מכיוון שאנו משתמשים בפונקציית ה-$Hash$ שמחשבת את האינדקס בסיבוכיות זמן של $O(1)$ ובפרט כדי להגיע לתא המתאים הוא $O(1)$ בתוחלת.
- עבור פונקציית $set(key, new\_value)$- בדומה ל-$find(key)$ נחפש את מיקום המפתח בטבלה באמצעות פונקציית ה-$hash$, לאחר מציאת התא, נעדכן את הערך החדש בתא להיות $new\_value - to\_add$ לערך הרצוי. פעולה זו גם כן בסיבוכיות זמן של $O(1)$ מכיוון שאנו משתמשים בפונקציית ה-$Hash$ שמחשבת את האינדקס בסיבוכיות זמן של $O(1)$ בתוחלת.
- עבור בניית מבנה הנתונים- פעולת ההכנסה בודקת אם המפתח כבר נמצא במבנה הנתונים בסיבוכיות זמן של $O(1)$, אם לא, מכניסה את הערכים הרצויים לטבלה. הכנסת n ערכים $n \cdot \underbrace{O(1)}_{\text{בתוחלת}} = \underbrace{O(n)}_{\text{בתוחלת}}$ כמבוקש.
- עבור הפעולה $increment\_all()$- נעלה את הערך של המשתנה to_add ב-1.

סעיף ג:

נשתמש ב-$Hash-Table$ ונשתמש ב$Universal\ hashing$, כך שגודל הטבלה תהיה $2n$ (נרצה שגודל המערך יהיה קצת יותר גדול ממספר האיברים מכיוון שכאשר המערך כמעט מלא\ מלא הוא פחות משמעותי משמעותית), וכל איבר בטבלה יכיל את המפתח, ואת הערך שנרצה לשמור. כמו כן, בנוסף לטבלה, נשמור במבנה נתונים הכללי מספר משתנים נוספים

(לא ניתן להשתמש ב $Perfect\ Hashing$, מכיוון שהמפתחות לא ידועים מראש)

1. new_value- משתנה זה הוא משתנה מספרי שבהתחלה מאותחל להיות $null$, משתנה ישמור את הערך החדש לאחר זימון הפונקציה
2. $master\_version$- משתנה מספרי שמתחיל בערך הדיפולטיבי 0, בכל שינוי כללי של ה-new_value, נעלה את ערכו ב-1 כך כל ה-$local\_version$ שיהיו לכל תא בטבלה ידעו למי להקשיב (למשתנה הכללי, או למשתנה הלוקלי)

בנוסף, נשמור בכל תא (בנוסף למפתח ולערך) 2 ערכים נוספים:

1. ptr_new_value- מצביע לערך של new_value
2. $version$- משתנה מספרי המאותחל להיות תחילה ל-0, כך בריצת הפונקציה האלגוריתם ידע אם להתייחס לערך הלוקלי, או לערך הגלובלי

- עבור פונקציית $find(key)$- נשתמש ב-$Hash\ function$ עבור ה-key, נבדוק אם יש איבר באינדקס שאנחנו מקבלים, אם כן נבדוק אם $version = master\_version$ אם כן, נחזיר את הערך בתא, אם לא, נחזיר את הערך הגלובלי. אם לא נמצא את התא, נחזיר $null$. מכיוון ש $Hash\ function$ פועלת בסיבוכיות זמן של $O(1)$,ובפרט כדי להגיע לתא המתאים הוא $O(1)$ בתוחלת, וגישה לזיכרון גם ב $O(1)$, אכן הפונקציה בסיבוכיות זמן של $O(1)$ בתוחלת.
- עבור פונקציית $increment(key)$- בדומה ל-$find(key)$ נחפש את מיקום המפתח בטבלה באמצעות פונקציית ה-$hash$, לאחר מציאת התא, נבדוק אם $version = master\_version$ אם כן, נעלה את הערך ב-1, אם לא נגדיר את הערך המקומי להיות הערך של הערך הגלובלי+1, ונעדכן את ה-$version$, להיות $master\_version + 1$ פעולה זו גם כן בסיבוכיות זמן של $O(1)$ מכיוון שאנו משתמשים בפונקציית ה-$Hash$ שמחשבת את האינדקס בסיבוכיות זמן של $O(1)$.
- עבור פונקציית $set(key, new\_value)$- בדומה ל-$find(key)$ נחפש את מיקום המפתח בטבלה באמצעות פונקציית ה-$hash$, לאחר מציאת התא, נעדכן את הערך החדש בתא להיות new_value לערך הרצוי, ונעדכן את ה$version$ להיות $master\_version$. פעולה זו גם כן בסיבוכיות זמן של $O(1)$ מכיוון שאנו משתמשים בפונקציית ה-$Hash$ שמחשבת את האינדקס בסיבוכיות זמן של $O(1)$.
- עבור בניית מבנה הנתונים- פעולת ההכנסה בודקת אם המפתח כבר נמצא במבנה הנתונים בסיבוכיות זמן של $O(1)$, אם לא, מכניסה את הערכים הרצויים לטבלה. הכנסת n ערכים $n \cdot \underbrace{O(1)}_{\text{בתוחלת}} = \underbrace{O(n)}_{\text{בתוחלת}}$ כמבוקש.

הסבר: בתהליך זה, אנו בודקים בכל פעם את מספר הגרסה של התא הפנימי מול המשתנה הגלובלי. כך בכל תא ותא ידוע: אם הערך הגרסה שווה לגרסה הכללית, התא מעודכן ויש להתייחס לערך התא, אחרת יש להתייחס לערך התא הכללי. חשוב לציין שמוקצה עוד $O(n)$ מקום באלגוריתם זה, כדי לשמור את הערכים הפנימים שנשתמשים בכל תא (הפוינטר למצביע הכללי, ומספר הגרסה).