מבני נתונים ואלגוריתמים – תרגול #10

טבלאות ערבול – *Hash tables*

מילון (Dictionary)- מבנה נתונים המאחסן אוסף של רשומות מהצורה (key, value). המפתחות שונים מרשומה לרשומה. מבנה הנתונים תומך בפעולות *Search, Insert, Remove*.

אם ידוע שהמפתחות הם מספרים שלמים בתחום *{0,…, m-1}*, אפשר להשתמש במערך בגודל *m* ולממש את כל הפעולות ב-*O(1)*.

<u>בעיה:</u> כאשר טווח המפתחות האפשריים גדול בהרבה ממספר המפתחות בהם משתמשים בפועל ואז יש בזבוז של מקום. לדוגמא – יש 10^9 מספרי זהות אפשריים, ופחות מ-10^7 תושבים בישראל.

פיתרון אפשרי – אפשר לממש מילון באמצעות עץ *AVL*, ונקבל שהפעולות לוקחות זמן של *O(logn)* כאשר *n* הוא מספר המפתחות בעץ.

פיתרון אחר – <u>טבלאות ערבול</u>

נגדיר טבלה בגודל *m* וניצור **פונקציית ערבול** שמחשבת אינדקס בטבלה מתוך המפתח עצמו :

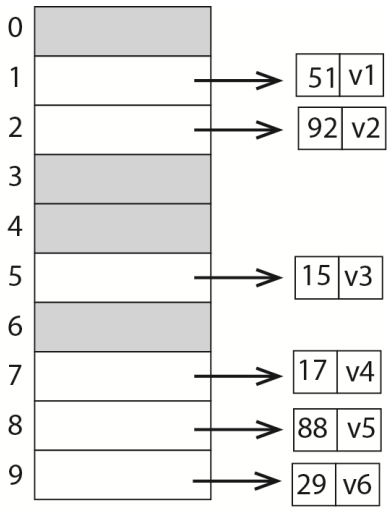
$$h: U \to \{0, \dots, m-1\}$$

כאשר *U* הוא תחום המפתחות האפשריים.

← נרצה שבממוצע הפעולות יתבצעו ב-*O(1)*

דוגמא: $h(k) = k \bmod 10$,*m=10*

קלטים:
51 → 51mod10 = 1
17 → 7
15 →5
92 →2
88 →8
29 →9

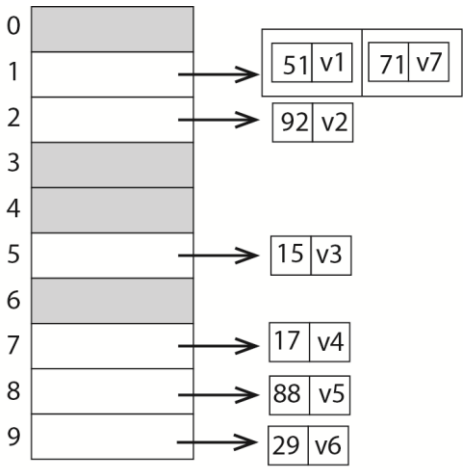


רואים ש-*m=10* לא טוב כי מסתכלים רק על הספרה האחורנה. גם 2 לא טוב.

התנגשות- כאשר $x_1 \neq x_2$ אך $h(x_1) = h(x_2)$.

פתרונות במקרה של התנגשות:

1) **שרשור *(Closed addressing)*** – יוצרים רשימה מקושרת מכל האיברים המעורבלים לאותו תא.



<u>מימוש פעולות:</u>

-*Insert*: נכניס את האיבר החדש לראש הרשימה המקושרת

-*Search*: מחפשים את האיבר ברשימה המקושרת

-*Delete*: מחפשים את האיבר ומוחקים אותו

<u>סיבוכיות הפעולות</u>: מניחים שפונקציית הערבול מפזרת את המפתחות <u>בצורה אחידה</u>. כלומר הסיכוי שאיבר מסוים יגובב לתא מסוים שווה לכל *m* התאים.

מקדם עומס: $\alpha = \frac{n}{m}$ (*n*- מספר המפתחות בטבלה). מקדם העומס מתאר את המספר הממוצע של איברים שמאוחסנים בכל תא (כלומר ברשימה המקושרת של התא).

← זמן חיפוש **כושל** בממוצע: *O(1+α)*.

הוכחה: בהנחה שהערבול אחיד ופשוט, ההסתברות שמפתח *k* יגובב לתא מסוים שווה לכל *m* התאים. לכן הזמן הממוצע לחיפוש כושל הוא סריקה של הרשימה המקושרת המתאימה עד סופה. האורך הממצוע של רשימה הוא מקדם העומס $\alpha = \frac{n}{m}$. לכן תוחלת מספר האיברים הנבדקים היא α. לכן הזמן הנדרש לחיפוש כושל (כולל חישוב *h(k)*) הוא *O(1+α)*.

← זמן חיפוש **מצליח** בממוצע: *O(1+α)*.

הוכחה: נניח שמחפשים איבר *x*. מספר האיברים שנסרקים ברשימה המקשורת הוא *1* ועוד מספר האיברים שלפני *x* (אלו הם האיברים שהתווספו אחרי *x*). לכן נחפש כמה איברים נוספו אחרי *x*, בממוצע.

נגדיר עבור *i=1,2,…,n* את x_i להיות האיבר ה-*i* שמוכנס לטבלה, ו-$k_i = key[x_i]$.

נגדיר עבור כל זוג *i, j* (*i≠j*) משתנה אינדיקטור X_{ij} שמקבל ערך 1 אם $h(k_i) = h(k_j)$ ו-0 אחרת. מכיוון שמדובר בערבול פשוט ואחיד הסיכוי להתנגשות הזוג *i,j* הוא:

$$P(X_{ij} = 1) = P\left(h(k_i) = h(k_j)\right) = \frac{1}{m} \;\rightarrow\; E[X_{ij}] = \frac{1}{m}$$

תוחלת מספר האיברים הנסרקים במהלך חיפוש מוצלח היא:

$$E\left[\frac{1}{n}\sum_{i=1}^{n}\left(1 + \sum_{j=i+1}^{n} X_{ij}\right)\right]$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left(1 + \sum_{j=i+1}^{n} E[X_{ij}]\right)$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left(1 + \sum_{j=i+1}^{n} \frac{1}{m}\right)$$

$$= 1 + \frac{1}{nm}\sum_{i=1}^{n}(n-i)$$

$$= 1 + \frac{1}{nm}\left(\sum_{i=1}^{n} n - \sum_{i=1}^{n} i\right)$$

$$= 1 + \frac{1}{nm}\left(n^2 - \frac{n(n+1)}{2}\right)$$

$$= 1 + \frac{n-1}{2m}$$

$$= 1 + \frac{\alpha}{2} - \frac{\alpha}{2m}$$

נוסיף את זמן חישוב פונקציית הערבול ונקבל:= $\Theta\left(2 + \frac{\alpha}{2} - \frac{\alpha}{2m}\right) = \boldsymbol{\Theta(1 + \alpha)}$.

המשמעות: אם מספר התאים הוא לפחות ביחס ישר למספר האיברים בטבלה אזי *n=O(m)* ומכאן נקבל *α = n/m = O(m)/m = O(1)*. לכן חיפוש אורך בזמן קבוע.

<u>תרגיל:</u>

נניח שאני משתמשים בפונקציית ערבול אקראית *h* לערבול *n* מפתחות שונים זה מזה לתוך טבלה *T* באורך *m*. מהי תוחלת מספר ההתנגשויות? כלומר מהי התוחלת של עוצמת הקבוצה:

$$?\{(k,l): h \neq l \; and \; h(k) = h(l)\}$$

<u>פיתרון:</u>

נגדיר עבור כל זוג *k,l* (*k≠l*) משתנה אינדיקטור X_{kl} שמקבל ערך 1 אם $h(k) = h(l)$ ו-0 אחרת. מכיוון שמדובר בערבול פשוט ואחיד הסיכוי להתנגשות הזוג *k,l* הוא:

$$P(X_{kl} = 1) = P\big(h(k) = h(l)\big) = \frac{1}{m} \;\rightarrow\; E[X_{kl}] = \frac{1}{m}$$

כעת נגדיר מ"מ נוסף *Y* להיות מספר כל ההתנגשויות, כלומר $Y = \sum_{k \neq l} X_{kl}$. נחשב את *E[Y]*:

$$E[Y] = E\left[\sum_{k\neq l} X_{kl}\right] = \sum_{k\neq l} E[X_{kl}] = \binom{n}{2}\frac{1}{m} = \frac{n(n-1)}{2}\cdot\frac{1}{m} = \frac{n(n-1)}{2m}$$

2) *Rehash* (*Open addressing*):

הרעיון – כל האיברים נמצאים בטבלה עצמה. כאשר מכניסים איבר חדש, בודקים את התאים בזה אחר זה עד שמוצאים מקום ריק. באותו אופן על מנת לחפש איבר, בוחנים בשיטתיות תאים בטבלה עד שמוצאים את האיבר המבוקש או שמגיעים למסקנה שהוא לא נמצא.

במקום לחפש איבר בטבלה לפי הסדר הקבוע *0,1,2,…,m* (שייקח *O(n)*), ניצור *i* פונקציות גיבוב חוזר *(Rehash)* $h_i(x)$, כך שאם $h(x_1) = h(x_2)$ ו-x_1 כבר נמצא בטבלה, נלך ל-$h_1(x_2)$, ואז ל- $h_2(x_2)$ וכו', עד שנמצא מקום פנוי.

דוגמאות לפונקציות *Rehash* בהניתן פוקנציית ערבול *h(x)*:

<u>*Double hashing*</u>: $h_i(x) = \big(h(x) + i\cdot r(x)\big) mod\ m$ (*r(x)* היא פונקציית צעד)

<u>*Linear probing*</u>: $h_i(x) = (h(x) + i)\ mod\ m$

<u>*Quadratic probing*</u>: $h_i(x) = (h(x) + i^2)\ mod\ m$

← זמן חיפוש **מצליח/כושל** בממוצע : $O(\frac{1}{1-\alpha})$ (ללא הוכחה).

ערבול קוקיה *(Cuckoo)*: יש 2 פוקנציות ערבול h_1,h_2 שמקיימות $h_1(x) \neq h_2(x)$.

אם $h_1(x) = h_1(y)$ ו-$h_2(x) = h_2(y)$ אז $x = y$.

הרעיון: בודקים אם $h_1(x)$ פנוי. אם תפוס נבדוק את$h_2(x)$. אם גם תפוס, נוציא את האיבר שנמצא ב-$h_2(x)$ (נקרא לו *y*) ובמקומו נשים את *x*. באותה שיטה נחפש מקום ל-*y*.

ערבול אוניברסלי

לכל בחירה של פוקנציית ערבול קיימת סדרה גרועה של מפתחות כך שתיווצר רשימה באורך מקסימלי. הפיתרון: בזמן יצירת טבלת ערבול, לבחור באקראי פונקציית ערבול מתוך קבוצה פונקציות שהוגדרה מראש. נרצה שקבוצת הפונקציות תהיה כזו, שעבור כל סדרת מפתחות, בחירה אקראית של אחת הפונקציות תיצור פיזור טוב.

הגדרה: תהי H קבוצת פונקציות ערבול $H: U \to \{0,\dots,m-1\}$. הקבוצה H נקראת אוניברסלית אם לכל זוג מפתחות שונים $x,y \in U$, מספר הפונקציות עבורן $h(x) = h(y)$ הוא $\frac{|H|}{m}$.

← ההסתברות שבבחירה אקראית של h , x יתנגש עם *y* היא $p = \frac{1}{|H|}\left(\frac{|H|}{m}\right) = \frac{1}{m}$.

<u>תרגיל:</u>

הציעו דרך להקצאה ושחרור מקומות עבור איברים בתוך טבלת הערבול עצמה (כשאר משתמשים בשיטת השרשור). כלומר, במקרה של התנגשויות, לא ניצור תאים חדשים לרשימות המקושרות, אלא נשתמש במקומות הפנויים של הטבלה. הפעולות צריכות להתבצע בזמן קבוע.

<u>פיתרון:</u>

- ניצור טבלה T.
- נשתמש בשדה נוסף כדגל המציין אם התא פנוי או לא.
- ניצור רשימה דו-מקושרת של כל התאים הפנויים.
- כל תא יחזיק 2 פוינטרים:
  תא פנוי - לרשימת הפנויים.
  תא תפוס - לצורך הרשימה המקושרת של כל הערכים שמעורבלים לאותו ערך.

פעולות:

<u>הוספה:</u>

1) מקום j פנוי – מחק מרשימת הפנויים והכנס ערך.
2) מקום j תפוס – בודקים אם הערך x שב-j מקיים את h(x)=j.
   אם כן – ניקח את המקום בראש רשימת הפנויים (ונמחק אותו משם), נשים בו את הערך ונשרשר לרשימה של תא j.
   אם לא – (הוא שייך לרשימה של תא אחר) – נעביר את הערך ה'ישן' לתא אחר* ובמקומו נשים את הערך ה'חדש' ב-j.

   * נשים אותו במקום פנוי, ונעדכן את הפוינטרים ברשימה המקושרת אליה הוא שייך.

<u>מחיקה:</u>

נסמן ב-j את המיקום שצריך למחוק, ו-x ערך הערבול.

1) אם x שייך ל-j ו-j לא מצביע על כלום – מחק והכנס לראש רשימת הפנויים.
2) אם x שייך ל-j, ויש עוד איברים מקושרים – העבר את הראשון ברשימה ל-j ושחרר את המקום הישן.
3) אם x לא שייך ל-j – מחק מהרשימה המקושרת הרלוונטית והוסף את המקום לרשימת הפנויים.

<u>חיפוש:</u> רגיל.

סיבוכיות: טיפול ברשימת הפנויים – O(1). שאר הפעולות – כמו בטבלה רגילה O(1+α).

<u>תרגיל:</u>

נתונה טבלת *hash* בגודל *m=11* ו-*2* פונקציות *hash*:

h1(x)= ((value of the last letter of x) mod m

h2(x)= (sum of the values of the first and last letters of x) mod (m-1) +1

כאשר הערך של אות הוא מיקומה באלפבית (*a=1, b=2,…*)

	bread	milk	apple	pasta	water	salad	steak	rice	chips	chocolate
h1	4	0	5	1	7	4	0	5	8	5
h2	7	5	7	8	2	4	1	4	3	9

א) ציירו את טבלת ה *hash* – לאחר הכנסת כל אחת מהמילים בכל אחת מהשיטות הבאות:

a. Chaining with h1 as your hash function

b. Linear probing with h1 as your hash function

c. Rehashing with h1 as your first hash function, and h2 as a step function

ב) למה אי אפשר להשתמש ב-*h1* כפונקצית צעד?

ג) למה אי אפשר להשתמש ב-*h2* כפונקציה ראשונה?

<u>פיתרון:</u>

א)

	a	b	c
0	milk, steak	Milk	Milk
1	pasta	Pasta	Pasta
2		Steak	Steak
3			chips
4	bread, salad	Bread	Bread
5	apple, rice, chocolate	Apple	Apple
6		Salad	
7	water	Water	Water
8	chips	Rice	Salad
9		Chips	Rice
10		chocolate	chocolate

ב) כי *h1* יכולה להחזיר 0, ואז לא נבדוק מקום אחר כאשר נעשה *rehash*.

ג) כי *h2* יכולה "לפספס" ערכים מסוים בטבלה, למשל 0.