**מבנה נתונים - אלגוריתמים**

ספר בקורס:

Cormen T.H , Leiserson C.E, Rivest R.L, Stein C   
Introduction to algorithms , MIT Press  
<http://www2.eitan.ac.il/ds/index.asp>

ציון: 75% בחינה, 25% תרגילים

**עמ 4 - הגדרות ופעולות בסיסיות**

רשימות לינאריות : קבוצת/רשימת איברים עוקבים.

על רשימה לינארית יש מס פעולות שניתן לבצע.

ישנם כמה סוגי רשימות: מחסנית (lifo), תור(fifo) או תור דו צדדי(גם fifo וגם lifo אך לא ניתן לשלוף מהאמצע).

מימושים:

**מבנים**  
ישנם כמה אפשרויות להזנת רשימה במחשב:

|  |  |
| --- | --- |
| next | Value data info |

הקצאה סידרתית: השמה אחד אחרי השני (מערך חד מיימדי באורך גודל הרשימה)  
רשימה מקושרת: לכל איבר יש שתי חלקים (מצביע (הוא מצביע על האיבר הבא בתור) וערך) (struct בגודל של אורך הרשימה, כלומר בעל n קודקודים)

עמ 5 דוגמאות ל- Pseudo code: קוד בצורה ענייה (ללא טיפוס) אלא רק האלגוריתם  
 כל השורות של הקוד ממוספרות  
 פתיחה וסגירת קבוצת פקודות ע"י

if……  
 end if

while (...) loop ….  
end loop

שיעור 2:

דוגמא **לתור**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 13 | 12 | 11 |  |  |
| In -> | tail |  | head | Out-> |  |

Tail- מצביע על איבר אחרון

**תור מעגלי** – מוציאים מהראש (fifo)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | 13 | 12 | 11 |  |  |
| In -> | tail |  |  | head | Out-> |  |

Tail- מצביע על האיבר אחרי האחרון

חסרון: תמיד tail יצביע על תא ריק  
כלומר כאשר head = tail +1) mod (length Q)) התור מלא  
וכאשר head = tail אזי התור המעגלי ריק

כאשר אני מכניס איבר לתור מעגלי: אני מכניס אותו במקום ש-TAIL מצביע עליו ואז נקדם אותו באחד (אם הוא בסוף אז הוא יצביע על תחילת המערך (1))

כאשר אני מוציא איבר מתור מעגלי: הראש מתקדם באחד לכן כאשר אני מרוקן את התור head=tail

דוגמא נוספת לתור מעגלי:

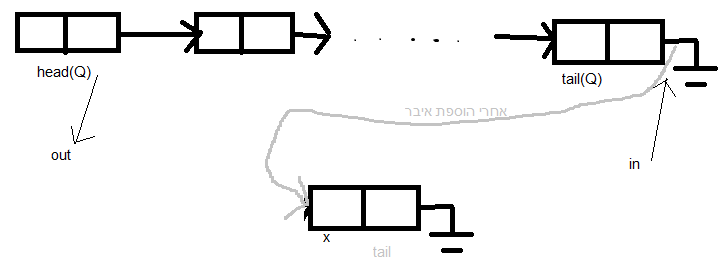
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 11 |  |  |  |  | 13 | 12 |
| head | Out-> |  | In -> | tail |  |  |

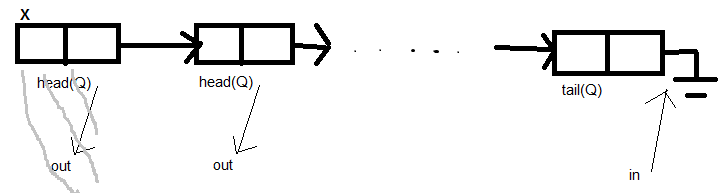
אחרי הוצאת איבר אחד:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | 13 | 12 |
|  |  |  | In -> | tail |  | Head (out->) |

**רשימה מקושרת לינארית**

הוספת איבר:



הוצאת איבר: נצטרך לבדוק הפעם אם הרשימה ריקה  


**רשימה מקושרת עם קשרים כפולים**

רשימה זו היא כמו רשימה מקושרת לינארית אך הפעם נוסף לנו הפוינטר לאיבר הוקדם ברשימה

חיסרון: צורך יותר זיכרון (חצי מהזיכרון שהיו צריכים לרשימה מקושרת מתווסף)

|  |  |  |
| --- | --- | --- |
| next | Value data info | pervious |

אם נרצה ליצור רשימה כזו אך מעגלית נצטרך ליצור איבר אחד שיהיה זקיף אשר בו התא המכיל value/data יכיל מידע (שאליו נתייחס כזקיף).

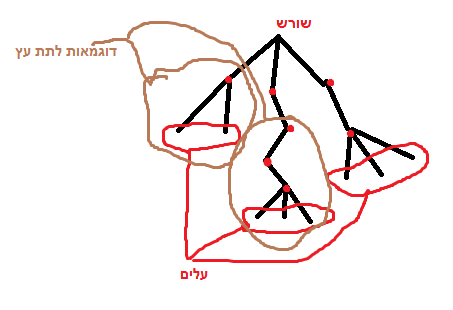
**סיבוכיות זמן ריצה**

טטה ᶱ= חסם מלעיל ומרע  
אומגה Ω= חסם מלרע  
O= חסם מלעיל

דוגמא: עמ' 5  
PUSH(S,x)   
ב-stack full יש 2 פקודות  
error פקודה 1  
ואם זה לא טעות אז 2 פקודות  
לך הסיבוכיות היא טטה של 4 .

עמ' 7 list-search  
מקסימום הפעמים שנבצע את הלולאה היא ש-K לא נמצא היא אורך הרשימה => O(n) n=אורך הרשימה  
מינימיום יהיה כאשר K יהיה הראשון => O(1)  
לכן הסיבוכיות זמן ריצה בדוגמא זו היא O(n)

שיעור 3

פרק 3: **עצים trees (עמוד 8 למטה)**

רמה: 0

1

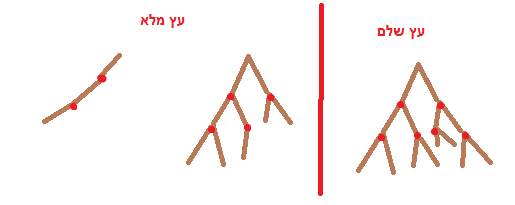
2

3

4

גובה העץ בדוגמא הוא h=4

הגדרות: עץ בינארי

עץ שלם = עץ שבו כל **העלים** הם בעלי אותו גובה. (יותר סימפטי לשם חיפוש וכו'...)  
עץ מלא = אם לכל **העלים** יש אותה רמה,ולכל צומת שאינו עלה יש שני בנים העץ הוא **עץ מלא**

\*\*(האיור לא נכון)

בעץ שלם h=Omega(log n)  
בעץ מלא h=O(n)

עץ בינארי הוא שורש בעל מקסימום בעל 2 תתי עצים בינארים או עת ריק.  
בעץ בינארי לכל אב יש מקסימום 2 בנים מינימום 2 בנים

משפטים: עבור עץ בינארי -> עמוד 9

**שורש** - קודקוד ללא הורים

**עלה** - קודקוד ללא ילדים

**עומק/גובה העץ** - אורך המסלול הארוך ביותר בעץ (כשסופרים קשתות), נסמן זאת באות D.

**עומק קודקוד i** - אורך המסלול מהשורש לקודקוד ה- i, נסמן (d(i.

**הורה** - לכל קודקוד i (מלבד השורש) קיים הורה - (parent(i (עבור השורש הפעולה תחזיר לנו null).

**בנים** - לכל קודקוד i שאינו עלה קיים בן שמאלי - (left(i, וקיימים קודקודים שאינם עלים שיש להם גם בן ימני - (right(i.

**עץ שלם** - עץ שבו כל העלים הם בעלי אותו גובה.

**עץ מלא** הינו עץ בו כל הקודקודים פרט לעלים הם מדרגה 2.

**עץ כמעט שלם** - עץ שבו כל הרמות מלאות חוץ מאולי הרמה התחתונה שבה החל ממקום מסוים יתכן שאין יותר קודקודים.

**גובה העץ** - אורך המסלול מהעלה הנמוך ביותר לשורש.

**עמודים 9-10**  
שכבות המשתנים בזיכרון:

|  |
| --- |
| קבועים, קוד, משתנים סטטיים |
| משתנים אוטומטיים משתנים של פונקציות |
| אזור התאים החופשיים משתנים של allocate וכו' ולאחר מכן הם משתחררים |
| משתנים דינמיים |

מעשית בזיכרון אין באמת שכבות, אך זוהי דרך נוחה לראות את כל המשתנים כיצד הם נשמרים (ב-stack ו-heap)

מבנה של משתנה של ב"עץ חיפוש בינארי":

|  |  |  |
| --- | --- | --- |
|  | parent |  |
| right | ערך | left |

כאשר left- יהיה בעל ערך קטן יותר מהערך  
right- יהיה בעל ערך גדול יותר מהערך

דוגמא-   
מערך של מספרים: 2, 14, 13, 13, 0 ,4  
לכן העץ חיפוש בינארי של מערך זה הוא:

4  
 13 0  
14 13 2

אם נזין כעת את נתוני העץ לתוך מערך חדש כך שנתחיל קודם להזין את התת עץ השמאלי ולאחר מכן את השורש ואז את התת עץ הימני נקבל מערך ממויין

0 2 4 13 13 14

הקשר בין n (מס הקודקודים) ל-h (גובה העץ(:

אם העץ הוא עץ שלם: n=2h+2h-1+…+21+20= 2h+1-1  
log2n≤O(h)

h≥Ω(log2n)  
h≤O(n)

**עמוד 11**

משפט עזר: אם לקודקוד מסוים x יש שני בנים, אזי לעוקב שלו y אין בן שמאלי (או שכל הבנים של y

הם גם כן שווים ל- y.

משפט עזר: אם תת-העץ הימני של קודקוד x הוא ריק, וקיים y = succ(x), אזי y הוא האב הקדמון הקטן ביותר של x כך שהבן השמאלי שלו גם כן אב-קדמון של x.

**עמוד 10**  
 פונקצייה זו מראה כיצד מוצאים את המספר העוקב של מספר כלשהו בעץ.TREE-SUCCESSOR (x)   
אם יש בן ימני נרד אליו ומשם נלך לבן השמאלי שלו (נכד שמאלי) אם אין אז הבן הוא ההבא.  
 אם לא אז נעלה עד שלא יהיה לאן לעלות שמאלה ו האחד הוא האב הימני הראשון

16

18 14

20 17 15 6

7 3

13 4 2

9

דוגמא:  
succ(16)=17  
succ(13)=14  
succ(6)=7

**עמוד 11**

משפט עזר: אם לקודקוד מסוים x יש שני בנים, אזי לעוקב שלו y אין בן שמאלי (או שכל הבנים של y הם גם כן שווים ל- y.

TREE-DELETE (T, z)

מטרה: לבטל קודקוד X

16

18 14

20 17 15 6

7 3

13 4 2

מקרה א- אין בנים ל-Z   
  
  
  
  
  
  
  
מקרה ב- יש ל-Z בן אחד

16

18 14

20 17 15 6

7 3

13 4 2

מקרה ג- יש ל-Z 2 בנים   
נאתר את ההבא.  
ודוחפים אותו במקום Z  
(לא יתכן שלאיבר הבא יהיו 2 בנים – משפט נ"ל)

16

18 14

20 17 15 6

7 3

13 4 2

2.5

X

Y=successor

\*\*\*שאלה יפה למבחן: מצא את הpredessesor(הקודם)

**עמוד 15**  
הגדרה: עץ AVL (Adelson-Velskii, Landis) הוא עץ חיפוש בינארי שבו כל קודקוד מקיים:  
| h[[1]](#footnote-1)( left-subtree ) - h( right-subtree ) | <= 1

דוגמא לעץ לא מאוזן (לא עץ AVL)

החזקה הזו היא הגבוה של העץ אם ניקח את העלה הימני כקודקוד ואז כל פעם נעלה ונתייחס למספר הבא ונחשב את הגובה מהקודקוד החדש  
עץ מאוזן יהיה עץ בו הגבהים האלו יהיו 0, 1- ו/או 1+

8-2

10-1 70

11-1 90

120

משפט: אם נכניס n אברים בצורה אקראית לתוך עץ בינארי (ריק מלתחילה), אזי המסלול הממוצע מהשורש לעלה הוא: O(log n) (בעצם: (log n) ).

נסמן ב- P(i) מספר הקודקודים הממוצע מהשורש לקודקוד כלשהו:  
נניח כי יש n קודקודים לעץ. P(0)=0 , P(1)=1   
נניח כי השורש (האיבר הראשון שהוזרם) נקרא a,   
נניח כי i איברים שיוזרמו קטנים מ-i , ולכן n-i-1 איברים גדולים מ-a  
כלומר יראה כך:

אזי: P(n)=1/n =1+O(log n)

ולכן כיוון שידוע כי זה גם =Ω(log n) אזי נכון לומר=Ө(log n)

לאחר שמכניסים איבר חדש insert לעץ בינארי מאוזן ורוצים לשמור על האיזון שלו ישנם כמה

**מקרים**:  
הגדרה LL:

הקודקוד החדש Y נכנס בתת עץ השמאלי (L) של התת עץ השמאלי (L) של השורש A.  
A לא בהכרח השורש.   
הגדרה מדויקת: A האב הקדמון הקרוב ביותר של Y שעבורו גורם האיזון נהפך ל-I2.

הגדרה: RR

הקודקוד החדש Y נכנס בתת עץ הימני (R) של התת עץ הימני (R) של השורש A.  
A לא בהכרח השורש.   
הגדרה מדויקת: A האב הקדמון הקרוב ביותר של Y שגורם האיזון שלו נהפך ל-I2.

הגדרה: LR

הקודקוד החדש Y נכנס בתת עץ הימני (R) של התת עץ השמאלי (L) של השורש A.

הגדרה: RL

הקודקוד החדש Y נכנס בתת עץ השמאלי (L) של התת עץ הימני (R) של השורש A.  
  
RR:

10

11 8

12 10.5 9 7

8

10 7

11 9

8

10 7

11 9

12 10.5

8

10 7

7.5 5

8

10 7

7.5 5

5.5 4

7

8 5

10 7.5 5.5 4

LL:

LR:

8

10 4

6 3

7 5

8

10 4

6 3

8

10 6

7 4

5 3

6

8 4

10 7 5 3

RL:

10

12 8

14 11 9 4

8

10 4

12 9  
  
 14 11

8

12 4

14 10

8

10 4

14 10  11 9

נסמן ב- f(h) את המספר המינימלי של עץ AVL (עץ מאוזן) בגובה h  
f(0)=1 f(1)=2

f(h)=f(h-1) + f(h-2) +1

f(h)=2\*f(h-1)+1

טענה: f(h)>= ᶲh ᶲ=(1+(5)0.5=1.6

נוכיח באינדוקציה:

H=0 : F(0)=1 =ᶲ1

H=1: F(1)=2>ᶲ2

נניח כי האי-שיויון מתקיים עבור h<=k-1: f(k)>= ᶲk-1

f(k) )=f(k-1) + f(k-2) +1>=ᶲk-1+ᶲk-2+1=ᶲk-2(1+ᶲ)+1=ᶲk+1>=ᶲK

n=f(h)>= ᶲh

logn1>=h\*log ᶲ

h<=(1/logᶲ)log n=1.4\*log n

=> h=O(log n)

**חיפוש**- עמוד 15

**בד"כ נתונה קבוצה S אנו מחפשים איבר X הנמצא בתת קבוצה Tהמוכלת ב-S**

דוגמא:

חיפוש שם בספר טלפונים:   
S= כל המחרוזות באורך >= 20: =2025|S|  
T= מספר השמות בספר טלפונים >= 106 (סתם ככה)  
  
מקרה ממוצע (סיבוכיות): ממוצע של כל מקרה (i) נסכום אותם ונחלק בכל המקרים (n)

(1/n)\*∑i=(1/n)\*(n\*(n-1)/2)=O(n)

**חיפוש לינארי:** O(n)

נעבור על כל הרשימה מתחילה ועד סופה

**חיפוש בינארי:** O(log n)

כאשר הרשימה ממוינת על מנת לחפש איבר בה כל פעם נחתוך באמצע הרשימה ונבדוק אם הוא שווה קטן או גדול (אם שונה נחתוך את התת מערך/ווקטור המתאים וכן הלאה)

**חיפוש אינטרפולציה:**O(log(log(n)) הסיבוכיות יותר מורכבת באמת כיוון שצריך לחשב את j

הנחות: נתונים ערכים נומריים:x0<x1<……<xn

נתון Z שיש לחפש אותו ב-T

ערכי xi גדלים בצורה פחות או יותר אחידה ברוח (x0,xn)

מטרת החיפוש: מציאת אינדקס j כך שמתקיים: xj<=Z<=xj+1

Z-X0=[(XN-X0)/n-0](j-0)

J=[(Z-X0)/(Xn-X0)]\*n

אלגוריתם:

1) עבור Z נתון הגדר J=[(Z-X0)/(Xn-X0)]\*n

2) אם Z=Xj גמרנו

3) חאם לא, נבחר רווח שמאלי (X0,Xj) או ימני (Xj+1,Xn)

4) נחזור לנק 1 (יש דוגמא מאחורי עמ/ 15)

Z

1 2 …….. j j+1

(0,x0)

(n,xn)

## HASHING ערבול גיבוב

**עמוד 16**

**הגדרה:**

נתון יקום של **S** פריטים ובתוכו קבוצה (מצומצמת יותר) של **Z** פריטים.  
איך להגדיר פונקציה כדי למפות כל פריט למקומות בתוך טבלה (מערך).

דוגמא: ת.ז - קיימים 109 אפשרויות לכל ספרה- בישראל 7\*109

בפעם הראשונה שזה התעורר זה בקומפילציה, עבור ההצהרות, כלומר כיצד לארגן את ההצהרות עבור הקומפילציה. כיוון שהקומפילציה בנויה מטבלה של סימבולים.

לכל ההצהרות יש טבלה שבה מוכל המידע על ההצהרה הזאת, וכל סימבל מצביע אחד על השני.  
דוגמא:

Const int n=10;  
int x,y;  
int z[20];

Double u;  
….  
u=x+y;

הכי פחות יעיל:  
כלומר תחילה יצרו את כל המשתנים (הגדרתם) ברשימה מקושרת ולכל כל פעם שהמחשב מגיע למשתנה הוא רץ ומחפש ברשימה מקושרת את המשתנה. ולכן עבור קומפילציה (שאמורה לעבוד מאוד מהר) חיפוש ברשימה מקושרת לוקח יותר מידי זמן)

במקרה של משתנים יש S=26\*3631 ולעומת זאת (בממוצע) T=104

מבנה יעיל יותר עבור t קטן: (עמוד 16)  
כאשר T קטן מאוד ניתן ליצור מערך של פוינטרים לstruct של כל משתנה ויהיה לנו פונקציית key(x) אשר תקשר את המשתנה x לאינדקס של התא במערך של הפוינטרים המכיל את המצביע לstruct של x.

**T[n]**

|  |
| --- |
| **S**  Struct  עבור x |
| slot 2  **Key(x)** |
| **t**  x |
| **n פריטים** |
| Struct  עבור משתנה אחר |
|  |

מבנה עבור t גדול: - פונקציית hashing  
ניצור פונקצית key אשר תקשר את המשתנה בשם מסוים לכתובת מסוימת ע"פ הערך ההסקי של השם שלו. נצביע למערך בגודל m עבור t בעל n פריטים גודל המערך (m) יהיה ע"פ היחס הבאα=(n/m) :. כך שזמן החיפוש יהיה לכל היותר 3 שניות לדוגמא

**S**

|  |
| --- |
| 0 |
| 1 |
| 2 |
|  |
|  |
|  |
|  |
| m-1 |

יכול מאוד להיות שפונקצית hashing של שתי משתנים יוביל אותנו לאותו כתובת, מה שיקרה במקרה שנגיע לאותו כתובת כאשר יש כבר משתנה(x) שהגיע לשם ניצור רשימה מקושרת של ה-struct הזה כפי שניתן לראות בדוגמא הנ"ל (כאשר Y נולד אחרי X)

**t**

**T[]**

**Hashing(key(y))**

Struct   
עבור x

Struct   
עבור y

**n פריטים**

Key(x)

Key(y)

**Hashing(key(x))**

**פונקציות hashing:**   
עבור כל אפשרות של hashing קיימות 3 פונקציות לשיטת hashing בעמוד 16:   
(chained-hash-\*\*\*\*\*(T,x)

1. נבחר את המפתח k=key(x)   
   ואז נחשב k2 וניקח את 5 הספרות האמצעיות של k2.  
   מתברר ששיטה זו אינה טובה כיוון שההתפלגות במקרה של פונ' זו אינה מספיק טובה.
2. חילוק:   
   k=key(x)  
   hashing(k)=k mod m  
   יש להיזהר לא לבחור m=2i, כיוון שאם בוחרים שמות בנאלים למשתנים כגון (x1, y1,z1) כל המשתנים לאחר פעולת mod יתכנסו לאותו כתובת.
3. כפל:   
   hashing(k)=[m\*(k\*A mod 1)]  
   נבחר ערך עבור A (לדוגמא בחירה טובה היא: 50.5-1)/2)  
   נבחר ערך עבור m (לדוגמא: m=2p כאשר p הוא ראשוני)  
    לדוגמא: עשינו Key(x) וקבלנו k=7 ונקבע כי A=0.81 m=10  
    h(7)=[10\*(7\*0.81 mod 1)]=[10\*95.67 mod 1)]=[10\*0.67)]=[6.7]=6

הערכה של האלגוריתם:  
הנחה:  
 key(x) , O(1)   
O(1) , h(key(x)  
הכנסת פריט O(1)  
הסרת פריט O(1)  
חיפוש פריט-   
במקרה הגרוע ביותר:  
כל הפריטים מתאימים לאותו slot ולכן יש רשימה מקושרת באורך n. O(1+n) (+1 כיוון שיש את זמן חישוב h(key(x)))  
במקרה הממוצע:  
נסמן את אורך הרשימה במקושרת הנמצאת ב slot T[j] ב-nj  
הערך הממוצע של nj הוא =(n/m)α  
 מקרה ראשון- חיפוש נכשל: יש לעבור על כל הרשימה המקושרת בslot- שאורכו הממוצע הוא α,   
 לכן סדר האלגוריתם של החיפוש הוא O(1+α)  
 מקרה שני- חיפוש מצליח: נניח כי הפריט שאנו מחפשים הוכנס במקום ה-i. אחריו הוכנסו n-i   
 פריטים. לכן ב-slot של הפריט הנדון יש לפניו בממוצע (n-i)/m פריטים. (m הוא גודל   
 כל המערך T לכן בשביל לקבל ממוצע נחלק ב-m)  
 לכן זמן החיפוש הממוצע יהיה:

=

אם n גדול מאוד  
=> O(1+ 0.5α)=O(1+α)=O(1)

דוגמא: נניח כי n=1000 ואנחנו מוכנים לקבל זמן חיפוש של 3. מה ערכו של m(כמות ה-slots)?

אנו יודעים כי =(n/m)α ולכן כל שנותר הוא להציב

2+0.5α-α/(2n)=2+0.5\*1000/m-(1000/m)/(2\*1000)=3 => m=499.5

**בשרשור Hashing**

כלומר לאחר שעשינו hashing קבלנו לאותו תא (key) מספר משתנים לכן כעת נצטרך לבצע שרשור לכל hashing בו תתבצע "התנגשות".

(עמוד 16) ראה 3 הפונקציות האחרונות המופיעות בעמוד.

**כיצד לבחור את פונקצית ה-hashing על מנת שלא יהיו יותר מידי "התנגשויות"?**

ישנם m ערכים אפשריים-

K=key(x)

L=Key(y)

כלומר אנו נרצה לדעת מהי ההסתברות שh(k)=h(l) :

מס הזוגות (h(k),h(l)) : m2

מס הזוגות (h(k),h(l)) שעבורן h(k)=h(l) =m

לכן ההסתברות לקבלת h(k)=h(l) היא m/m2=1/m

זו היא פונקציית hash מאוד מאוד יעילה:

**Ha,b(k)=[(a\*k+b)mod p] mod m**

**דוגמא:**

a€{1,2,…,30)

b€{0,1,2,…30}

מס פונ' ה-hash הוא: 30\*31

נבחר: k=20 a=3 b=2

H3,2=[(3\*20+2)mod31]mod5=0

H24,29=…=3

**משפט:** נניח שבחרנו פונקציות hash מתוך אוסף אוניברסאלי אם מפתח k לא נמצא עדיין בטבלה או ה-slot שאליו יגיע הוא בממוצע α=n/m.   
אם K כבר נמצא בטבלה האורך הממוצע של ה-slot שלו הוא α+1.

## Perfect Hashing

**הגדרה:**

שיטת ערבול נקראת מושלמת אם מס הגישות לזיכרון הנדרשות כדי לבצע חיפוש הוא O(1).

**דוגמא:** Perfect hashing

בה נבצע פעמיים hashing ובכך נגיע ב-O(1) לכל תא

נניח שבחרנו פונ' hash אוניברסאלית:  
p=101, a=3, b=42, m=9

H3,42(22)=7  
H3,42(37)=7  
H3,42(40)=7  
…..

כעת לאחר שכולם הגיעו לאותו תא נבצע hashing חדש עם אינדקסים נוספים ונגיע לתא חדש  
P=101, a=23, b=88, m=9

H23,88(22)=8  
H23,88(37)=3  
H23,88(40)=0

כלומר כעת צריך לדעת כיצד לבנות את טבלת המערך (הזיכרון/מבנה הנתונים):

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
| 7 |
|  |

8 ...... 3 2 1 0 b a m

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 22 |  |  | 37 |  |  | 40 | 83 | 23 | 9 |

## Open hashing

**Linear probing:**

k=key(x)

H(k,i)=h, (k)+i i=0,1,2,3,…..

כלומר אם המקום h מלא אז ניקח את המקום (k)+i

**Quadratic probing:**

h(k,i)=h, (k)+c1\*i+ c2\*i i=0,1,2,3,…..

**Double probing/hashing:**

h(k,i)=[h1(k)+i\*h2(k)] mod m i=0,1,2,3,…..

את h1 לדוגמא ניתן לחשב בכל צורה  
אם התא אליו הגענו נבצע Hash מחדש עם i מקודם באחד

**דוגמא:**

h1(k)=k mod 13

h2(k)=1+(k mod 11)

m=13

|  |  |
| --- | --- |
| 0 |  |
| 1 | 79 |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 | 98 |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 | 14 |
| 10 |  |
| 11 |  |
| 12 |  |

זהו המצב הנתון וכעת נרצה להוסיף k=14

i=0: h(14,0)=h1(14)+ 0\*h2(14)=1

אבל תא מס 1 כבר מלא לכן נעבור ל-i+1=1

i=1: h(14,1)=h1(14)+1\*h2(14)=5

אבל תא מס 5 כבר מלא לכן נעבור ל-i+1=1

i=2: h(14,2)=h1(14)+2\*h2(14)=9

לכן נכניס בסוף את k=14 לתא מספר 9

מיונים:\*(להשלים מהדפים)

Heap sort

1. H- הגובה של התת עץ ולא הBF שלו [↑](#footnote-ref-1)