מבני נתונים: סיכום הרצאות

22 ביוני 2025

הסיכום נכתב בזמן השיעור (ומבוסס על המצגת שהועלתה ללמדה) ולכן יתכן שנפלו בו טעויות, על אחריותכם.
© גיא יער־און.

חלק I

סיבוכיות

חסמים אסימפטוטיים:

1. $O()$: $f(n) \in O(g(n)) \iff \exists c > 0, n_0 \geq 0 : \forall n \geq n_0 : f(n) \leq c * g(n)$
2. $\Omega()$: $f(n) \in \Omega(g(n)) \iff \exists c > 0, n_0 \geq 0 : \forall n \geq n_0 : f(n) \geq c * g(n)$
3. $\Theta()$: $f(n) \in \Theta(g(n)) \iff \exists c_2 \geq c_1 > 0, n_0 \geq 0 : \forall n \geq n_0 : c_1 * g(n) \leq f(n) \leq c_2 * g(n)$
או ־ $\lim_{x\to\infty} \frac{f(n)}{g(n)} \in \mathbb{R}+$
4. $o()$: $f(n) \in o(g(n)) \iff \forall c > 0, \exists n_0 \geq 0 : \forall n \geq n_0 : f(n) \leq c * g(n)$
או $\lim_{x\to\infty} \frac{f(n)}{g(n)} = 0$
5. $\varpi()$: $f(n) \in \varpi(g(n)) \iff \forall c > 0, \exists n_0 \geq 0 : \forall n \geq n_0 : f(n) \geq c * g(n)$
או $\lim_{x\to\infty} \frac{f(n)}{g(n)} = \infty$

חשוב לזכור כי ־
$1 \leq logn \leq loglogn \leq logn \leq log^2 n \leq 2^{\sqrt{logn}} \leq \sqrt{n} \leq n \leq nlogn \leq n^2 \leq 2^n \leq n! \leq n^n$
* כדאי לזכור כי $logn! = log1 + log2 + ... + logn = \sum_{k=0}^{n} logk = \Theta(nlogn)$

שיטת האב:

עבור נוסחאות מהצורה $T(n) = aT(\frac{n}{b}) + f(n)$
תמיד נשווה את $n^{log_b a}$ כאשר המספר שיצא הוא החזקה של $log_b a$. נשווה אותו ל ־ $f(n)$ ־
אם $f(n) = \Theta(n^{log_b a})$ נקבל שזה יהיה $\Theta(f(n) * logn)$.
אם $f(n) = O(n^{log_b a - \varepsilon})$ כאשר $\varepsilon > 0$ נקבל כי $T(n) = \Theta(log_b a)$
אם $f(n) = \Omega(n^{log_b a + \varepsilon})$ כאשר $\varepsilon > 0$ נקבל כי $T(n) = \Theta(f(n))$ (כלומר אם $f(n)$ גדול מהאן בלוג, היא תנצח)

חלק II

מחסנית

מחסנית היא מבנה נתונים לינארי שדוגל בגישת $LIFO = last - in - first - out$ התומך בפעולות

הבאות -

1. $Push$- מכניסה איבר לראש המחסנית. (סיבוכיות הפעולה - $O(1)$)
2. Pop- מוציאה איבר מראש המחסנית. (סיבוכיות הפעולה - $O(1)$)
3. Top- מחזירה את האיבר שבראש המחסנית. (סיבוכיות הפעולה - $O(1)$)
4. $Multi-Pop$- מוציאה את כל האיברים מהמבנה. ראינו פעולה זאת בהרצאה כשניסינו לנתח - למרות שניתן לחשוב שהסיבוכיות היא $O(n^2)$ באופן מפתיע היא דווקא ב- $O(n)$.
5. $IsEmpty$.- מחזירה האם המחסנית ריקה או שלא. (סיבוכיות הפעולה - $O(1)$)
6. $Creat-Stack$: מחזיר מחסנית ריקה s. (סיבוכיות הפעולה - $O(1)$)

ניתן לממש מחסנית באמצעות מערך / רשימה מקושרת. בהרצאה ראינו עם מערך.

חסרון של מימוש עם מערך - גודל המחסנית מוגבל מראש,תתכן "גלישה" בניסיון להכניס איבר למחסנית מלאה.

חלק III

תור

מבנה נתונים לינארי הגודל בגישת $FIFO = FIRST - IN - FIRST - OUT$, מימוש ניתן באמצעות מחסנית/מערך. תומך בפעולות הבאות -

1. $-create-queue(Q)$ מחזיר תור ריק.
2. $-enqueue(x,Q)$מכניס איבר x לתוך Q.
3. $front(Q)$ - מחזיר את האיבר שבראש התור - התור עצמו לא משתנה.
4. $dequeue(Q)$- מוציא את האיבר שבראש התור.
5. $is-Empty$ - מודיע לנו אם התור ריק
6. $Size$- מחזיר את מס' האיברים בתור.

כל הפעולות הללו הן בסיבוכיות $O(1)$.

חלק IV

ניתוח לשיעורין

מה זה ניתוח לשיעורין? מדובר בטכניקה לניתוח זמן ריצה עבור **סדרת פעולות,** ניתוח זה יאפשר לנו לקבל חסם זמן ריצה נמוך יותר מזה שנראה כאשר אנו מניחים את ה$worst-case$ עבור כל פעולה. נניח כי עלות הפעולה הi הינה c_i, נרצה לחשב $T(n) = \sum_{i=1}^{n} c_i$ ואז עלות כל פעולה בממוצע תהיה $\frac{T(n)}{n}$.

כל ההדגמה בסיכום תעבוד עם מחסנית ועם פעולת $multy-pop$, כאשר תוציא את כל האיברים מהמחסנית. בהינתן k איברים במחסנית עלותה תהיה $O(k)$. מדובר בפעולה שנראית יקרה, היא לינארית במס' האיברים במחסנית. אך נשמע שאנחנו מחמירים מדי עם ניתוחה הכללי, נתבונן בניתוח שגוי - עבור סדרה של n פעולות, מה העלות ה$worstCase$ עבור כל הסדרה? נוכל לטעון שנבצע n פעולות מולטי-פופ, מה שיגרור עלות של $O(n^2)$. אבל זה כלל לא נכון ולא הדוק - בשביל שיהיה n איברים במחסנית, חייבות להיות n פעולות $push$ לפני. לכן, אם יש לנו n פעולות $push$ ואחכ פעולת מולטיפופ אחת, נשלם על הכנסתם n ועל הפעולה n סה"כ $2n$, כלומר באופן ממוצע כל פעולה עולה לנו 2. ולכן העלות הממוצעת לכל פעולה במקרה הגרוע היא $O(1)$ ולא $O(n)$. **כעת נציג מס' שיטות לחישוב העלות הממוצעת של סדרת פעולות:**

1. שיטת הצבירה

בשיטה זו נוכיח כי סדרה של n פעולות עולה $T(n)$ זמן לכל היותר. נתבונן בסדרה כללית של

פעולות־

$$S = \{\sigma_1, ...\sigma_n\}$$

כל פעולה $\sigma_i \in \{push, pop, multi-pop\}$, ולכן אם עלות הפעולה i הינה c_i, נרצה לחשב את $T(n) = \sum_{k=1}^{n} c_k$. נתבונן בכל המיקומים בסדרה בהם $\sigma_i = multi-pop$, נניח שמיקומים אלו הם $I = \{i_1, ...i_t\}$, נסמן את סדרת הפעולות ביניהם כ$S_{ij} = \{\sigma_{i,j-1}, ...\sigma_{i,j}\}$.
טענה: לכל $1 \leq j \leq t$, עלות הפעולה $nulti-pop$ הנמצאת במיקום i, j הינה $i_j - i_{j-1} - 1$ לכל היותר. יתרה מזאת, עלות הסדרה $S_{i,j}$ חסום ע"י $2(i_j - i_{j-1})$.
הוכחה לא פורמלית: לאחר הפעולה σ_{ij-1} המחסנית ריקה כיוון שזו הייתה פעולת מולטיפופ, כל שאר הפעולות בסדרה אינן פעולות מולטיפופ, לכן לכל פעולה אחרת היא $push$ או pop. במקרה הגרוע ביותר כל הפעולות יהיו $push$ ולכן כאשר נגיע לפעולה σ_{ij} יהיו $i_j - i_{j-1} - 1$ איברים במחסנית. לכן זו עלות פעולת המולטי פופ, ולכן העלות הכוללת הינה $2(i_j - i_{j-1} - 1)$, אכן חסום ע"י החסם המתואר למעלה.
כעת, לכל סדרה:

$$T(n) \leq \sum_{i=1}^{t} 2(i_j - i_{j-1}) + \sum_{j=i_t+1}^{n} c_j \leq 2(i_t - i_0) + n - i_t \leq 2(i_t - i_0) + 2(n - i_t) \leq 2n - 2i_0 = 2n$$

שכן $i_0 = 0$, סה"כ $T(n) \leq 2n$ ולכן העלות הממוצעת לפעולה הינה 2. פורמלית.

2. שיטת הבנק

בשיטה הזו אנחנו ניתן לכל פעולה עלות שונה ממה שהיא באמת, היא תקרא העלות לשיעורין. חלק מהפעולות יקבלו עלות גדולה יותר מעלותן האמיתית וחלק פחות. כלומר, חלק מהפעולות ישלמו על הפעולות האחרות. נשמור במעין בנק את העלויות שכבר שילמנו.
נסמן ב$\hat{c_i}$את העלות לשיעורין של הפעולה הi. נסמן בc_i את העלות האמיתית. נראה כי מתקיים:

$$\hat{c_i} = c_i + deposit - withdraw$$

כלומר העלות היא העלות האמיתית+ההפקדה לבנק, פחות עלות המשיכה. תמיד נשמור על העקרון ש"לא נכנסים למינוס" ולכן יתקיים

$$\sum_{i=1}^{n} \hat{c_i} \geq \sum_{i=1}^{n} c_i = T(n)$$

ננתח כעת כך את מולטיפופ:
עבור פעולת $push$: עלות הפעולה היא 1, בנוסף נכניס אחד לבנק ולכן:

$$\hat{c_i} = 1 + 1 - 0 = 2$$

עבור פעולת pop: עלות הפעולה היא 1, נמשוך יחידה אחת מהבנק (על מה ששילמנו מראש) ולא נפקיד ולכן

$$\hat{c}_i = 1 + 0 - 1 = 0$$

עבור פעולת $multiPop$**:** נניח שמס' האיברים במחסנית הינו k, עלות הפעולה היא אכן k ובבנק יש כרגע k יחידות אותם נמשוך ולכן

$$\hat{c}_i = k + 0 - k = 0$$

סה"כ העלות לשיעורין של פעולה מקסימלית היא 2, ולכן

$$T(n) \leq 2n$$

כלומר העלות הממוצעת לשיעורין של כל פעולה הינה 2.

3. שיטת הפוטנציאל

דומה לשיטת הבנק, רק פורמלי הרבה יותר: יהי D_0 המצב ההתחלתי, נסמן בc_i את העלות האמיתית של פעולה i. D_i הינו מצב המערכת לאחר הפעולה הi. פונקציית הפוטנציאל ϕ ממפה כל מצב של מבנה הנתונים D_i למס' ממשי שמציין את הפוטנציאל הממומש למבנה הנתונים. נגדיר את העלות לשיעורין של הפעולה הi כך:

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$$

נדרוש שתמיד יתקיים

$$\phi(D_n) \geq \phi(D_0)$$

ואז אכן

$$T(n) \leq \sum_{i=1}^{n} \hat{c}_i$$

דוגמאות:

ניתוח מונה בינארי:

נניח שיש לנו מונה עם k ביטים, הסופר בבינארית. נרצה לתמוך בפעולה אחת $increment$ שתעלה ב־1. נגדיר את עלות כל פעולה בצורה הבאה: עלות כל פעולה = מס' הביטים שהוחלפו. כמה עולה הפעולה? כמה עולה רצף כזה של n פעולות? ניתוח לא הדוק יגיד שנבצע במקרה הגרוע $O(k)$ החלפות בפעם אחת, ולכן סה"כ $O(nk)$ כאשר k מס הביטים. זה לא הדוק. לכן נבצע ניתוח לשיעורין ־ נגדיר את פונקציית הפוטנציאל להיות מס' האחדות במונה

$$\phi(D_0) = 0$$

וכן לכל $1 \le i \le n$ נדרוש $\phi(D_i) \ge 0$, אכן עונה לדרישות. נניח כי יש t אחדות רצופות מסוף המונה (מימין), במצב כזה עלות הפעולה הגדלה ב1 תהיה $t+1$ (שכן נשנה את t האחדות לאפסים, ואת האפס אחריהן לאחד), מצד שני יש כעת פחות $t-1$ אחדות בביטוי, ולכן:

$$\hat{c}_i = t + 1 - (t-1) = 2$$

ולכן רצף של n פעולות יעלה $2n$, וכל פעולה תהיה בממוצע 2.

ניתוח מערך דינאמי:

הבעיה: לפנינו מערך התומך בפעולת $insert$ - מכניס את האיבר למקום הפנוי הבא. נניח כי במערך יש n מקומות. נניח כי המערך מלא - כעת אם נתבקש להכניס איבר נוסף נבצע את הפעולות הבאות: נקצה מערך חדש בגודל $2n$, נעתיק אליו את n האיברים הישנים, ונוסיף את האיבר החדש במיקום $n+1$. הנחה: זמן הקצאה $O(1)$. נעיר כי ניתוח של n פעולות יכול להיות שגוי - במקרה הגרוע ביותר $insert$ עולה $O(n)$ ולכן n פעולות יעלה $O(n^2)$. האמנם? בשביל להגיע למצב שנבצע $O(n)$ צריך לעשות n פעולות שיעלו $O(1)$. לכן כל פעולה מתבצעת בזמן קבוע. ננתח:

נרצה למצוא פונקציית פוטנציאל וזה דבר די קשה. איך נחפש אותה? נרצה שככל שיש יותר איברים במערך כך הסיכוי להעתיק אותו יגדל, הפוטנציאל שיצטבר במערכת יהיה גדול יותר ולכן נרצה לשלם על כך מראש. נשתמש בשני משתנים, $size$ ישמור את אורך המערך וnum את מס' האיברים כרגע במערך. יש שני מקרים מעניינים: רגע לאחר שהעתקנו את כל האיברים מתקיים $size = 2num$, וכן כאשר המערך מלא מתקיים $size = num$. במקרה הראשון נרצה שפונקציית הפוטנציאל תהיה 0, ובשני שפונקציית הפוטנציאל תקבל את ערך $size$. מי תקיים את הדרוש?

$$\phi(D_i) = 2num - size$$

כיוון שתמיד המערך מלא בלפחות חצי מהאיברים אכן $\phi(D_i) \ge 0$. כעת ננתח לשיעורין.

מקרה ראשון. אם פעולה i לא גורמת להרחבת מבנה הנתונים. אזי $size_{i-1} = size_i$ וכן $c_i = 1$ וכן $num_i = num_{i-1} + 1$

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1}) = 1 + 2num_i - size_i - 2num_{i-1} + size_{i-1}$$

$$= 1 + 2(num_i - num_{i-1}) + size_{i-1} - size_i$$

$$= 1 + 2(1) + 0 = 3$$

כלומר במקרה בו אין הרחבה של מבני הנתונים מתקיים $\hat{c}_i = 3$.

מקרה שני. אם פעולה i כן גורמת להרחבת מבנה הנתונים. אזי $c_i = num_i = num_{i-1} = size_{i-1}$ וכן $size_i = 2size_{i-1}$ ולכן

$$\hat{c_i} = c_i + \phi(D_i) - \phi(D_{i-1}) = size_{i-1} + 2(num_i - num_{i-1}) + size_{i-1} - size_i$$

$$= 2(num_i - num_{i-1}) + 2size_{i-1} - size_i$$

$$= 2(0) + 0 = 0$$

סה"כ לכל פעולה $\hat{c_i} \leq 3$ ולכן $T(n) \leq 3n$ כלומר עלות ממוצעת לשיעורין לפעולה הינה 3.

חלק V

בעיה חשובה - מחסנית ותור מינימום

1. מחסנית מינימום - נרצה לתאר מחסנית מיוחדת, כלומר היא תומכת בכל הפעולות של המחסנית הרגילה עם טוויסט - היא מחזירה את האיבר המינימלי ביותר במחסנית (את ערכו) והיא לא מוציאה אותו מהמחסנית.

כיצד נעשה זאת?

נעזר במחסנית רגילה, פרט לכך שכל איבר במחסנית יהיה זוג (a, b) כך ש-a הוא הערך שהכניסו למחסנית, ו-b הוא הערך המינימלי במחסנית בעת הכנסת a. הדרך לחשב את b היא להסתכל על האיבר שהיה בראש המחסנית (a', b') - כלומר זו שכאשר a' הוא הערך שהכניסו אחרון למחסנית, ולהגדיר את $b = min\{a, b'\}$ בכל פעם כשמכניסים איבר למחסנית.

למימוש פעולת החזרת איבר המינימום פשוט נחזיר את הערך ששמור במיקום השני (b) בזוג האיברים שבראש המחסנית.

מה מיוחד כאן כל כך? נשים לב שכיוון שהפכנו את המחסנית להיות ממחסנית מספרים למחסנית של זוגות סדורים - לא יהיה קשה לשלוף את המינימום. ומה זה לא קשה?... $O(1)$!

2. תור מינימום - הרעיון זהה. נשתמש במחסנית מינימום + תור בשני מחסניות.

מהו תור בשני מחסניות?

יש לנו שתי מחסניות - מחסנית הכנסות ומחסנית הוצאות. אל מחסנית ההכנסות נכניס את כל האיברים החדשים, כשנרצה להוציא נעביר את כל האיברים במחסנית ההכנסות אל מחסנית ההוצאות - ואז למעשה יתהפך הסדר שלהם, נשלוף את האיבר הראשון בראש המחסנית , נוציא אותו, ואח"כ נחזיר את האיברים למחסנית ההכנסות.

נשים לב שכאשר נדחוף איבר למחסנית ההכנסות לכל איבר ניתן 4 שקלים. כניסה למחסנית ההכנסות, הוצאה ממנה לתוך מחסנית ההוצאות (2 שקלים) והוצאה ממחסנית ההוצאות. מדוע 4? להכניס זה 1, להוציא למחסנית ההוצאות זה 3+2 (כי $s1.push(s2.pop)$) ולאחר מכן זה עוד 1- הרביעי, להוציא אל מחסנית ההכנסות.

חלק VI

עצים ועצים בינאריים - הגדרות בסיסיות

עץ הוא מבנה נתונים היררכי.

1. **גרף -** גרף הוא זוג סדור $G=(V,E)$ כאשר V היא קבוצה סופית שאיבריה הם הקודקודים/הצמתים. וE היא קבוצה סופית של הקשתות בגרף. באופן מתמטי נוכל להגיד כי $E=\{(v,u)|v,u\in V\}$
2. **עץ חופשי -** גרף קשיר ללא מעגלים.
3. **עץ מושרש -** עץ שבו בחרנו את אחד הקודקודים להיות השורש.

* הגדרה רקורסיבית לעץ מושרש (היא ציינה שיכולים לשאול על זה במבחן - ולכן זה כאן) - צומת בודד הוא עץ מושרש. זהו גם שורש העץ.
אם r הוא צומת ו $T_1,...T_k$הם עצים מושרשים , אז המבנה הנוצר באופן הבא הוא עץ מושרש:
r הוא שורש של העץ החדש
השורשים של $T_1,...T_k$ מחוברים לr בקשתות.

4. **הורה -** "אבא", הוא הצומת שמחוברת מלמעלה בקשת עם הבן.
5. **אב קדמון -** למשל השורש הוא האב הקדמון של כל הצמתים בעץ.
6. **דרגה -** מספר הילדים של כל צומת
7. **עלה -** צומת ללא ילדים
8. **צומת פנימית -** צומת שאינה עלה
9. **מסלול -** סדרת צמתים שכל אחד הוא ההורה של הקודם.
10. **אורך המסלול = מספר הקשתות =** מספר הצמתים פחות אחד
11. **גובה העץ -** אורך המסלול הארוך ביותר משורש העץ לאחד העלים.
12. **עומק צומת -** אורך המסלול מהצומת לשורש העץ
13. **תת עץ מושרש בX -** בוחרים את אחד הקודקודים בעץ נגיד שהוא x. אזי העץ שx הוא שורשו ומכיל את כל צאציו של x הוא תת עץ מושרש.
14. **רמה של צומת -** מספר הקשתות שיש לעבור כדי להגיע משורש העץ עד לצומת המבוקש.
15. **עץ סדור -** יש משמעות לסדר הילדים. מי בימין ומי בשמאל.
16. **עץ בינארי:** עץ ריק, או לכל צומת יש 2,1,0 ילדים.
17. **עץ בינארי מלא -** לכל צומת פנימי יש בדיוק שני ילדים.
18. **עץ בינארי שלם -** עץ בינארי מלא בו כל העלים באותו העומק.

-

טענה: בעץ בינארי מלא עם mעלים יש $m-1$ קודקודיים פנימיים.

טענה: מס' הצמתים בעץ בינארי שלם מגובה h הוא $n=2^{h+1}-1$ ולכן גם $n+1=2^{h+1}\Longleftrightarrow$ $log(n+1)=h+1 \Longleftrightarrow h=log_2(n+1)-1$

טענה: בעץ בינארי שלם מתקיים $h=\Theta(logn)$

טענה: בעץ בינארי כלשהו מתקיים $h=O(n)$ **וכן** $h=\Omega(logn)$

עצים בינאריים וסריקות עצים בינאריים

כאשר נרצה לייצג במחשב עץ בינארי נוכל לעשות זאת עם מצביע לשורש, ומעין מילון מורחב כך שהוא מחזיק את הערך, מצביע לאבא, ושני מצביעים - אחד לבן הימני ואחד לשמאלי.

סריקת $In-Order$: עץ שמאלי - שורש - עץ ימני. נשים לב שסריקה זו בעץ בינארי מדפיסה את האיברים בסדר ממויין.

סריקת $Post-order$: עץ שמאלי - עץ ימני - שורש.

סריקת $Pre-order$: שורש - עץ שמאלי - עץ ימני.

סיבוכיות זמן של כל הסריקות היא $\Theta(n)$.

* גובה של עץ ריק הוא -1.

פסודו קוד כיצד לחשב גובה של עץ -

```
height(x):
if (x=null) then return -1
else
L=height(left(X))
R=height(right(x)
return max(L,R)+1;
```

סריקת BFS: סריקה לרוחב של העץ - נסרוק רמה רמה באמצעות תור.
פסודו של bfs:

```
bfs(x):
Q=creat_queue()
if (x!=null) then enqueue(X,Q)
while not isEmpty(Q):
x=dequeue(Q)
print(x)
if (left(X)!=null) then enqueue(left(x),Q)
if (right(X)!=null) then enqueue(right(x),Q)
```

מילון - $dictionary$:

המילון שומר אוסף איברים. לכל איבר יש מפתח ייחודי.על אוסף המפתחות מוגדר סדר לינארי. למשל - המספרים הטבעיים.

הכוונה בייחודי - לא תהיה חזרה פעמיים ברצף על key.

יש מספר פעולות על המילון:

1. אתחול - $creat-dictionary()$: יוצר מילון ריק.
2. הכנסת איבר - $insert(k,D)$- מכניס איבר למילון שמפתחו k.
3. מחיקת איבר - $delete(k,D)$- מסיר איבר מהמילון שמפתחו k.
4. חיפוש - $find(k,D)$- מחפש איבר במילון שמפתחו k, אם לא מצא יחזיר $null$.
5. עוקב - $successor(k,D)$- מחזיר מצביע לאיבר במילון שמפתחו הוא העוקב של k , ו$null$ אם אין כזה
6. קודם - $predecessor(k,D)$- מחזיר מצביע לאיבר במילון שמפתחו הוא הקודם של k , ו$null$ אם אין כזה
7. מינימום - $min(D)$- מחזיר את המפתח המינימלי במילון
8. מקסימום - $max(D)$- מחזיר את המפתח המקסימלי במילון

ניתן לממש מילון באמצעות רשימה מקושרת (ממוינת או שלא) - אך זה לא יעיל זה יהיה $O(n)$
לכן נממש באמצעות עץ חיפוש בינארי.

הגדרה עץ חיפוש בינארי: עבור כל צומת, כל הערכים משמאל קטנים מערך הצומת וערך הצומת קטן מכל אלו שנמצאים מימין.

חלק VII

עצי B, עצי 2-3, ועצי אדום שחור

שאלה: בהינתן סריקת $in-order$ **של העץ, האם ניתן לשחזר את מבנהו? לא!**

שאלה: בהינתן סריקת $pre-order$**של העץ, האם ניתן לשחזר את מבנהו? כן!**

אלגוריתם שפותר את זה:

* עבור רשימה באורך אפס - טריוואלי, עץ ריק. סיימנו.

*עבור רשימה באורך גדול-שווה 1: נקבע את השורש להיות הערך הראשון ברשימה.ואח"כ נחלק את הרשימה ל-2 חלקים כאשר אנו יודעים שמדובר בעץ חיפוש בינארי ולכן הערכים משמאל לשורש יהיו כל הקטנים יותר, ומימין כל הגדולים. ובאופן פורמלי יותר: אם זו הרשימה - $root,x_1,x_2,.....x_k,y_1,y_2,....y_m$ כך שכל האיקסים קטנים מכל הוואים, נחלק את הרשימה ל-2 ונפעיל את האלגוריתם באופן רקורסיבי על כל קבוצה עד שנגיע לרשימה באורך אפס.

עץ חיפוש מסדר גבוה:

נסתכל על עץ בו דרגת כל קודקוד היא עד m. בכל קודקוד מדרגה $k \leq m$ יש עד $k-1$ ערכים פנימיים וk מצביעים לאיברים הבאים: $x_1, ...x_k$. המצביע הראשון מצביע לתת עץ שכל איבריו קטנים מx_1וכך זה ממשיך עד שהמצביע האחרון מצביע לכל אלו שגדולים מx_k.

הרעיון דומה לעץ חיפוש בינארי - בכל קודקוד כשנחפש נצטרך לדעת לאן להמשיך, ולכן נבדוק אם הערך נמצא בקודקוד שאנו נמצאים בו. אם כן סיימנו, אם לא נמצא בין אלו שני ערכים בקודקוד הוא נמצא וכך נמשיך למצביע שלו. אם הגענו לעלה והוא לא בעלה - הוא בכלל לא בעץ.

m יהיה מס' קבוע ולכן הסיבוכיות של החיפוש בכל קודקוד תהיה $O(1)$. אורך החיפוש בעץ יהיה לינארי בגובה העץ!

* גובה העץ: אם כל הקודקודים מנוצלים ברמתם כלומר עד דרגה m, ברמה אפס יש את השורש, ברמה 1 יש m קודקודים, ברמה 2 יש m^2קודקודים... וסה"כ ברמה הi יש m^iקודקודים. בכל קודקוד כזה יש $m-1$ ערכים, מכאן שסה"כ מס הערכים בעץ הוא $(m-1)\sum_{i=0}^{h} m^i = m^{h+1}$.

כלומר לשמור על n ערכים נזדקק לגובה של $h = log_m n$

מה הבעיה בעץ הזה? או או... (דמיינו את אלעד)

יש מצב שהעץ לא מלא ולא מאוזן. לכן אנחנו הולכים להסתכל על עצי B.

עצי B:

בדומה לעצי חיפוש מסדר גבוה, הפעם נדרוש עוד קצת -

1. $m \geq 3$ + אי זוגי.
2. בעץ מסוג זה עם פרמטר m, דרגת כל קודקוד פנימי פרט לשורש תקיים : $\left[\frac{m}{2}\right] \leq degv \leq m$
3. דרגת השורש - לפחות 2, ולכל היותר m או!! שהשורש הוא עלה.
4. בכל קודקוד פרט לשורש יהיו עד $m-1$ ערכים פנימיים ולכל הפחות $\left[\frac{m}{2}\right] - 1$ערכים.
5. בכל הקודקודים הפנימיים, מספר המצביעים הוא מס' הערכים+1.
6. כל העלים בעץ נמצאים באותה רמה.

עכשיו ניגש לכיף האמיתי-

גובה העץ: נסמן $b = \left[\frac{m}{2}\right]$, זהו מס' הבנים המינימלי של כל קודקוד פנימי (פרט לשורש). ברמה של עומק 1 יש לפחות 2 קודקודים ($m \geq 3$). ברמה של עומק 2 לפחות $2b$ קודקודים... ברמה של עומק i יש $2b^{i-1}$ קודקודים.

בכל עלה - יש לפחות $b-1$ ערכים כמו כן $b-1 \geq \frac{b}{2}$ ולכן אם רמת העלים - h יש לפחות $2b^{h-1} * \frac{b}{2} = b^h$ ערכים בעלי העץ.

כלומר אם מאוחסנים בכל העץ n ערכים, בוודאות $n \geq b^h$. ומכאן $h \leq log_b n = log_{\left[\frac{m}{2}\right]} n = O(log_m n)$

כלומר - עץ B הוא עץ מאוזן.

(הגדרה - נאמר כי עץ הוא מאוזן אם גובה העץ הוא $O(logn)$)

מדוע עץ B הוא טוב? במסדי נתונים (נלמד בהמשך) שומרים כמויות גדולות מאוד של מידע על דיסק קשיח. אותו דיסק בנוי בבלוקים ונרצה לפנות לבלוקים ולא לערך בודד.. מכאן מוטיבציה - נגלה בשנים הבאות.

מימוש עצי B:

1. הכנסה:

אם יש מקום בעלה, נכניס אותו במיקום המתאים וסיימנו.

אחרת, יש יותר מדי ערכים בעלה אם נוסיף אליו. נכניס בכל זאת לעלה ונגרום לגלישה - כעת יש בדיוק m ערכים בעלה. נפצל את העלה כך: בדיוק $\frac{m-1}{2}$העלים מימין ו$\frac{m-1}{2}$העלים משמאל הקטנים יותר. את הערך האמצעי נפעפע כלפי מעלה - ונכניס לאבא של הקודקוד. התהליך זה ימשיך באופן רקורסיבי עד שנגיע למצב שאין כבר מה לפצל וההכנסה בוצעה בהצלחה.

סה"כ סיבוכיות ההכנסה היא $O(log_m n)$.

2. מחיקה:

כל מחיקה תתחיל בחיפוש הערך בעץ.

א. אם הערך שצריך למחוק נמצא בעלה - נמחק אותו,אם נוצר $underflow$ נסתכל על הערכים בקודקוד אח שלו (נניח הימני) ועל הערך שמפריד ביניהם ברמה מעל.אם ניתן ליצור מהם שני קודקודים - כלומר הם לפחות יחד m איברים אז "נאזן". (מומלץ לראות חלק זה במצגת תרגול כי מילים זה יפה אבל אם מישהו הבין מה כתבתי כאן שילך למזכירות לקבל תואר)

אך אם לא ניתן לאזן עם האח הימני כי סך הערכים קטן מm, נמזג.

ולסיכום -

אם העלה באנדרפלו, נסתכל על הערכים באח מימינו (אם מדובר באח הימני ביותר אז מימינו=האח השמאלי ביותר) ועל הערכים בעץ שביניהם, אם יש יותר מm ערכים אז נאזן ע"י הלוואת ערך מהאח. אם יש פחות מm ערכים, אז נמזג את העלים. כתוצאה מכך נמחק ערך בקודקוד האבא וכך נמשיך עד לתיקון ברקורסיה.

ב. לא עלה - דומה מאוד.

נחליף את הערך שנמחק עם העוקב שלו שהוא חייב להיות עלה ואז נמחק את התא שהחזיק את העוקב. כדי למצוא את העוקב נלך למצביע שמימין לאיבר , ונלך לערך השמאלי ביותר בתת העץ שמוצבע (שוב, הבנת? לך למזכירות, הדוגמה מאוד ברורה ואז זה לא נראה סינית).

לסיכום -

אם העלה הגיע $underflow$: נסתכל על הערכים באח שמימינו (אם האח הימני ביותר - נשתמש באח השמאלי) והערך שביניהם. אם יש יותר מm ערכים - נאזן ע"י "הלוואת" ערך מהאח. אם יש פחות מm ערכים, נמזג לעלה אחד. כתוצאה, נמחק ערך בקודקוד האבא, ודרוש תיקון ברקורסיה.

מחיקה היא גם כן בסיבוכיות של $O(log_m n)$.

שאלות חשובות:

* כמה פיצולים מתבצעים בהכנסת איבר לעץ B מסדר m? $O(log_m n)$

*כמה פיצולים מתבצעים בעת הכנסת סדרה של n איברים לעץ B ריק? $O(\frac{n}{m})$

עצי 2-3:

ובכן ניתן להסתכל על כך כמקרה פרטי של עץ חיפוש מסדר m. כאשר $m = 3$.

הגדרה:

עץ 2-3 הוא עץ סדור המקיים -

1. כל צומת מכיל איבר אחד או שניים
2. לצומת עם איבר אחד יש שני בנים, לצומת עם שני איברים יש 3 בנים.
3. כל העלים בעץ נמצאים באותה הרמה.
4. עץ חיפוש - מפתחות העץ מסודרים בדומה לעץ חיפוש בינארי

* טענה: אוסף עצי חיפוש 2-3 הם משפחה מאוזנת של עצים, כלומר גובה העץ הוא $O(logn)$.

מימוש - עם מילון:

1. חיפוש

מאוד דומה לעץ חיפוש בינארי. להמשיך ולחפש עד שמגיעים לעלה ואם לא בעלה - לא בעץ. להעזר בתכונת עץ חיפוש.

2. הכנסה

חפש מפתח k והוסף אותו כעלה. הוסף את האיבר החדש להורה שלו , למקום המתאים. אם נוצרה "גלישה" - תקן את העץ שיהיה עץ 2-3.

מה ההגדרה שלנו לגלישה? יותר מ2 איברים בקודקוד. איך נתקן?

נפצל את הקודקוד ל-2 קודקודים. התיקון יתבצע מהעלה כלפי מעלה לאורך מסלול החיפוש, התיקון יסתיים כאשר לאחר ההוספה אין גלישה או מגיעים לשורש ונפצל גם אותו.

כלומר - בהוספת איבר לצומת v -

אם v מכיל שני איברים - יום המזל שלך - סיימת.

אחרת, אם v מכיל שלושה איברים, פצל את תתי העצים שלו והוסף את האיבר האמצעי של v להורה שלו וחזור על התהליך עד שלא יהיה בעיות / תגיע לשורש.

חיפוש והכנסה - $O(logn)$.

3. מחיקה מהעץ

חפש מפתח k. אם האיבר עם מפתח k אינו עלה אז החלף אותו עם העוקב. הסר איבר מהעלה - אם נוצרה חמיקה תקן את העץ.

טיפול בחמיקה (צומת ריק) -

א. לצומת הריק יש אח צמוד עם שני איברים:

העבר איבר מההורה של v אל v.

העבר איבר מהאח של v להורה של v , גרור עם האיבר מהאח את התת עץ המתאים בכיוון המעבר.

פיניש.

ב. אחרת - אח צמוד עם איבר אחד.

העבר איבר מההורה של v אל v.

מזג את הצומת v עם צומת האח לצומת אחד עם 2 איברים ו3 בנים

אם נוצרה חמיקה בצומת ההורה של v - חזור על תהליך טיפול בחמיקה.

אחרת, פיניש.

חלק VIII

עצי AVL

ראשית נעיר (כי הופיע בעבר במבחנים) כי יש דבר הנקרא עץ פיבונאצ'י שבדומה למס' פיבונאצי מוגדר באופן הבא -

1. F_0הוא קודקוד בודד.
2. F_1הוא קודקוד וקודקוד בן המחובר אליו משמאל.
3. $\forall n \geq 2 : Fi$ הוא עץ עם קודקוד שורש, מצד שמאל מופיע F_{i-2}ומצד ימין מופיע העץ F_{i-1}

-♡- כי עץ פיבונאצ'י הוא עץ חיפוש מאוזן.

-

עץ AVL הוא עץ חיפוש מאוזן. כלומר גובהו הוא $H = O(logn)$. כאשר n הוא מספר הצמתים.

הגדרה: עץ AVL הוא עץ חיפוש בינארי המקיים כי ההפרש בין הגובה של תת העץ השמאלי לתת העץ הימני הוא $\{1, -1, 0\} \ni$. זה קורה עבור כל קודקוד! ערך זה נקרא "גורם האיזון".

נזכר כי הגובה של עץ ריק הוא -1.

טענה: משפחת עצי AVL היא משפחה מאוזנת של עצים.

הוכחה: נסמן ב - $min - node(h)$ את מס' הצמתים המינימלי בעץ AVL בגובה h. נשים לב כי -

$min - node(0) = 1$

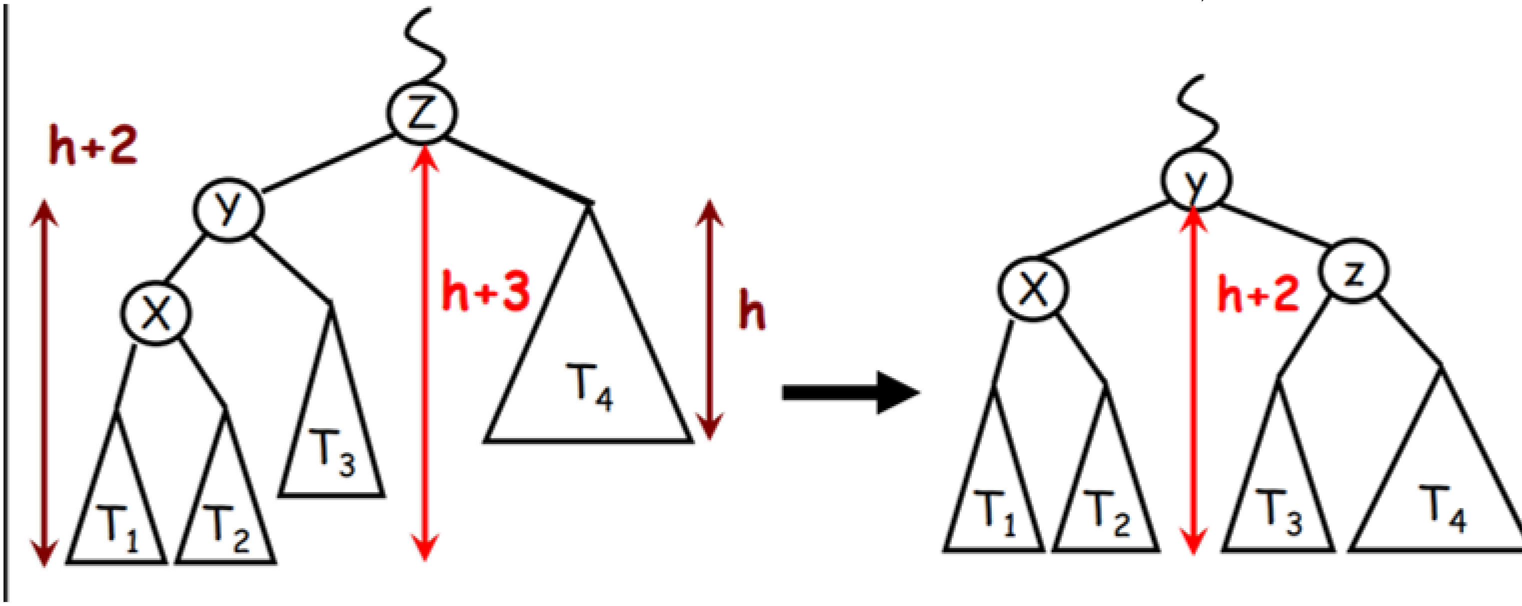
$min - node(1) = 2$

$min-node(h) = 1 + min-node(h-1) + min-node(h-2)$
אחרי הרבה מעברים....
$n \geq min-node(h) \geq 2^{\frac{h}{2}}$
$\Longleftrightarrow$
$logn \geq \frac{h}{2}$
ולכן אכן $h = O(logn)$
מש"ל.
-

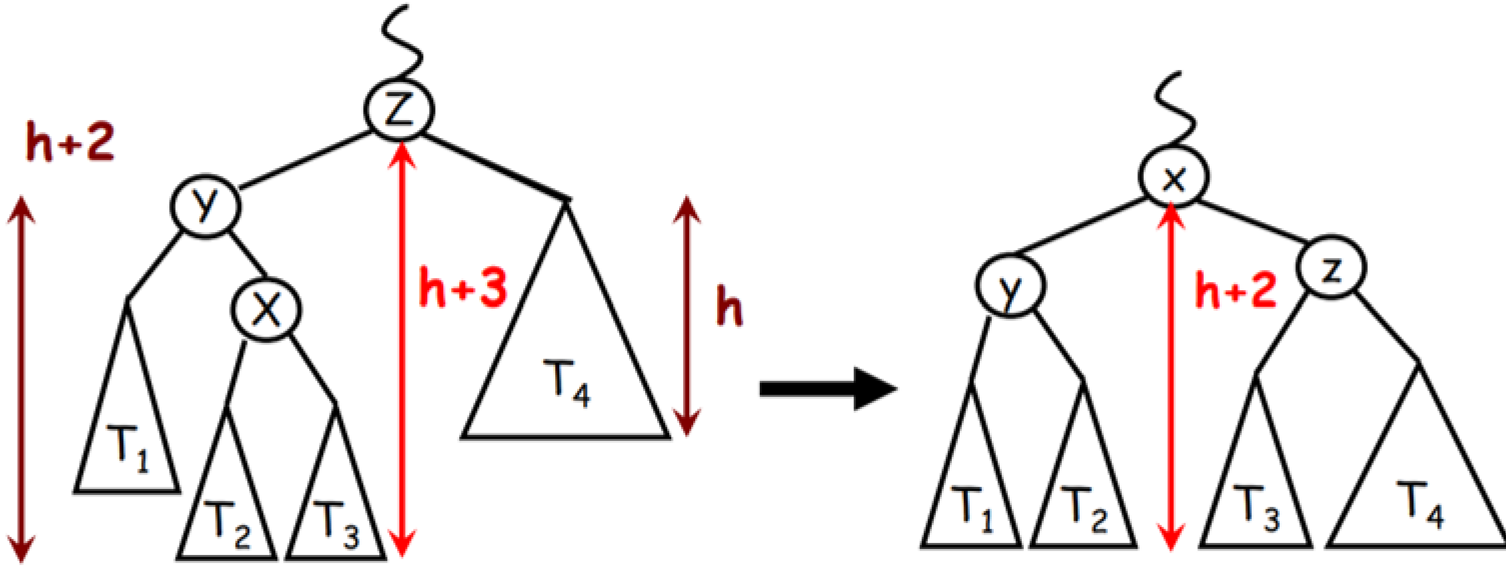
מימוש עץ AVL

1. הכנסה

נשים לב שמתכונת עץ החיפוש נרצה להכניס לעץ בדיוק במקום המתאים ולכן נחפש עבור כל צומת האם הערך שלנו גדול או קטן מהם עד שנגיע ל-$null$. כעת, מי אמר שהעץ נשאר AVL? נשים לב כי המקום היחיד למוקשים הוא במסלול ההכנסה, מהמקום שהכנסנו עד לשורש העץ! בשאר המקומות גורמי האיזון לא יכלו להשתנות. לכן נעלה למעלה מהקודקוד עד השורש
*הערה - יהי u קודקוד במסלול ההכנסה.אם גובה תת העץ של h לפני ולאחר ההכנסה נשאר זהה,אזי לכל אב קדמון של h גורם האיזון אינו משתנה! כלומר נשים לב כי אם נגיע במעבר כלפי מעלה לגורם איזון ששווה אפס - נסיים כי מעליו יובטח שגורמי האיזון הם בסדר ובטווח שלנו!
נשים לב כי אם איזון AVL מופר גובה העץ יגדל וכן האיזון יכול להיות מופר רק לב2 או -2.
כעת ננתח את ההכנסה -
יהי v קודקוד ולו תת עץ (ימני נניח) בגובה h. וכן תת עץ שמאלי בגובה $h+2$, נשים לב כי גורם האיזון של העץ הוא 2 - לא טוב. נרצה לארגנו מחדש כך שישאר עץ חיפוש בינארי ומקדם האיזון יקטן.
מכאן נסתכל על מושג שנקרא "גלגולים" - נגלגל את העץ כך שגורם האיזון יסתדר - (זה החלק הכבד ביותר בנושא הזה)
גלגול ראשון (נעזר בתמונה להמחשה) -



נשים לב שבעזרת תכונת עץ החיפוש נוכל לומר כי סדר איברי העץ הוא כדקלמן - $T_1, x, T_2, y, T_3, Z, T_4$.
ולכן אם נעביר את העץ למצב מימין, סדר איברי העץ לא ישתנה!
מה כן ישתנה? גורם האיזון שיהיה תקין!
2. גלגול שני -



גם כאן, נשים לב כי סדר איברי העץ הוא נשאר קבוע לאחר ה"גלגול". מה השתנה? גורם האיזון.

אלגוריתם לגלגול -

1. הכנס את הערך x כמו שמכניסים לעץ חיפוש בינארי. יהי v העלה שהוסף לעץ.
2. כל עוד $v \neq root$ כלומר אנחנו עולים מעלה במסלול אבל עד שנגיע לשורש בצע את ההבאים:

א.$v = parent(v)$

ב. שנה את פרמטר האיזון של v

ג.אם גורם האיזון הוא 0 - הכל מעל מאוזן, סיים.

ד.אם גורם האיזון משתנה ל2 או -2, בצע גלגול מתאים וסיים.

-

2. חיפוש בעץ AVL

ממש כמו בעץ חיפוש בינארי. אין מה לפרט.

3. הוצאה מעץ AVL

נתבונן באלגוריתם, רעיון דומה להכנסה:

1. הוצא את x כמו בעץ חיפוש בינארי.

2.תקן את גורמי האיזון ובצע גלגולים באופן הבא -

לכל צומת v לאורך המסלול החל מלמטה ועד השורש בצע-

א. שנה את פרמטר האיזון של v

ב. אם גורם האיזון משתנה ל2 או -2, בצע גלגול מתאים.

ג. אם גובה העץ ששורשו v לא השתנה וגורם האיזון תקין, המשך כלפי מעלה. כשתגיע לשורש - סיימת לתקן.

עצי AVL- תרגול: (חשוב מאוד!!!)

0.1 בעיית מציאת האיבר ה-K בגודלו

הבעיה: עליכם לתאר מבנה נתונים התומך בפעולות הכנסה, מחיקה, חיפוש ומציאת האיבר ה-k בגודלו. –♡– כל אחת מהפעולות צריכה להיות בעלות של $O(logn)$.

פתרון: נשתמש בעץ AVL ונוסיף שדה $size(v)$ לכל קודקוד שישמור את סה"כ האיברים שיש בתת העץ המושרש ב־v. למשל, בהינתן r־ שורש העץ, $size(r) = n$ ובהינתן שני בניו r_L, r_R יתקיים כי $size(r_L) = n_1$ וכי $size(r_R) = n_2$ כך ש $n = n_1 + n_2 + 1$.
נתבונן באלגוריתם הבא ־

$$Select(root, k):$$
$$If(k > root.size) Return Error$$
$$leftSize = root.left.size // null.size = 0$$
$$If(leftSize = k-1) Return root$$
$$elseif(leftSize > k-1) Return Select(root.left, k)$$
$$elseif(leftSize < k-1) Return Select(root.right, k - leftSize - 1)$$

מה קורה כאן? בהינתן כי k האיבר שנרצה למצוא, אם הוא גדול ממס' האיברים בעץ ־ שגיאה! הוא לא בעץ. אחרת, נגדיר את הסכום בשמאל להיות $leftSize$ ואז, אם מס' הערכים בשמאל בדיוק שווה ל $k-1$ (למה פחות אחד? כי התקדמנו מהשורש איבר קדימה) אז זה הערך ונחזיר אותו. אחרת, נפצל למקרים ־ אם מס' האיברים בעץ המושרש גדול מ $k-1$ נלך שמאלה כי הוא נמצא בשמאל, אחרת נלך ימינה. בסוף נמצא אותו..
אכן מדובר באלגוריתם שרץ בסיבוכיות זמן של $O(logn)$.
כעת נעבור לטפל בחיפוש ־ **בדיוק** כפי שמחפשים בעץ AVL ולכן גם כאן $O(logn)$.
*הכנסה/הוצאה: כמו בעץ AVL אך יש לעדכן את ערך המשתנה $size$ בכל פעם. מה נעשה? בכל פעם במסלול ההכנסה/הוצאה נוריד/נחסיר אחד מערך ה$size$. נותר לטפל כעת רק ברוטציות ־ שוב בהינתן רוטציה כלשהי שיש לעשות בעץ מס' הפעולות קבוע, נעדכן בהתאם את מס' האיברים בעץ וסה"כ גם כאן נקבל כי $O(logn)$.
כנדרש.

0.2 בעיית הרוטציות

יהיו T_1 וT_2שני עצים בינאריים עם אותם ערכים! צ"ל כי קיימת סדרה של רוטציות פשוטות באמצעותה ניתן לעבור מT_1לT_2. עליכם לתת חסם למס' הרוטציות בפתרון.
פתרון: נתבונן על קודקוד r שהוא ראש העץ של T_2והוא נמצא היכן שהוא בעץ T_1 (במקרה הגרוע הוא עלה).
רוטציות שומרות על תכונת עץ חיפוש בינרי ולכן כל הערכים בתת העץ השמאלי בשני העצים קטנים מהשורש וכל הערכים בתת העץ הימני גדולים מהשורש. כך נמשיך ברקורסיה לפעפע את כל האיברים כלפי מעלה.
מה הסיבוכיות? סה"כ לכל פעולה במקרה הגרוע היא עלה ולכן n פעפועים כלפי מעלה כשכל פעולה היא $O(1)$, במקרה הגרוע העצים שונים לגמרי ולכן נצטרך לעשות n פעפועים שונים ונקבל $O(n) * O(n) = O(n^2)$.

0.3 מערך ממויין $\Longleftarrow$ עץ AVL

בעיה: נתון מערך מספרים ממויין $A = [1, ..., n]$. עליכם להציע אלגוריתם הבונה עץ AVL עם אותם הערכים.
פתרון: 1. פתרון נאיבי ־ נכניס את האיברים לפי הסדר אל העץ, כל הכנסה תעלה $log(n)$ ויש סה"כ n הכנסות ולכן $O(nlogn)$ ־ לא טוב!! נרצה יותר טוב
2. פתרון טוב ־ נשתמש בעובדה שהמערך ממויין לשיפור זמן הריצה. נבנה את העץ לפי המערך, ולאחר מכן נוכיח שהתוצאה היא אכן עץ AVL.
נבחר את r להיות החציון ־ ונבנה את המערך באופן רקורסיבי על שני החצאים.
לדוגמה ־
המערך הנ"ל 1 2 3
נבחר את 2 החציון להיות האבא, ו1 ו־2 יהיו ילדיו.
נסמן ב־$h(n)$ את גובה העץ שמתקבל מn ערכים.
־

נטען כי $h(n)=\lceil log(n+1)\rceil-1$. **טענה זו יש להראות באינדוקציה:**
בסיס: $n=1$, $h(1)=\lceil log(2)\rceil-1=0$ ואכן בעץ עם קודקוד אחד גובהו 0.
צעד: נניח שנכון עבור n ונחלק למקרים.
א. n אי זוגי : אזי יש לנו חציון ואז בכל צד בעץ ישנם $\frac{n-1}{2}$
$h(n)=1+h(\frac{n-1}{2})=1+\lceil log((\frac{n-1}{2})+1)\rceil-1=\lceil log(\frac{n+1}{2})\rceil=\lceil log(n+1)\rceil-1$ כנדרש.
ב. אם n זוגי: אזי יש לנו שני עצים בגובה שונה. תת העץ הגדול (בה"כ T_1) מכיל $\frac{n}{2}$ערכים.
מכאן שגובה העץ $h(n)=1+h(\frac{n}{2})=1+\lceil log(\frac{n}{2}+1)\rceil-1=\lceil log((\frac{n+2}{2})\rceil=\lceil log(n+2)\rceil-1$
נשים לב שלכל מספר זוגי $x>2$ מתקיים: $\lceil logx\rceil=\lceil log(x-1)\rceil$
ולכן הוכחנו את הדרוש.

-

כעת נסתכל על קודקוד כלשהו בעץ המתקבל, עלינו להראות שהוא מקיים תכונת AVL. הקודקוד הזה עלה למקומו בהיותו חציון לתת מערך כלשהו.
*אם תת המערך היה אי זוגי $2k+1$ איברים אז בכל אחד מתתי העצים יש k ערכים ולכן הפרש הגבהים הוא בדיוק 0 - כדרוש.
* אם תת המערך היה זוגי $2k$ איברים אז בתתי העצים יש $k-1$ וk ערכים ולפי האינדוקציה שהראינו
$|h(k)-h(k-1)|=|\lceil log(k+1)\rceil-\lceil logk\rceil|\leq 1$
וסה"כ אכן הטענה נכונה כנדרש. הסיבוכיות כאן כמובן היא $O(n)$.

0.4 מיון באמצעות עץ AVL

הבעיה: נתון מערך $A=[1,....,n]$ של ערכים כלשהם.הצע אלגוריתם שמבצע מיון איברי A בעזרת עץ AVL.
הפתרון -
ניצור עץ AVL ריק.עבור כל $1\leq i\leq n$ נכניס את $A[i]$ לעץ, לבסוף נערוך סריקת $in-order$ שתדפיס אותו בסדר ממויין.(תרגיל 2 ש"ב)
זמן הריצה: אתחול העץ $n.O(1)$ הכנסות סה"כ שכל אחת עלתה $logn$ יקח סה"כ $O(nlogn)$. סריקת $in-order$ תעלה $O(n)$ ולכן סה"כ $O(nlogn)$.

0.5 מיזוג עצי AVL:

הבעיה: נתונים שני עצי AVL - T_1 וT_2.עם n_1 וn_2ערכים בהתאמה.**כך שאין ערך שמופיע בשני העצים.**
הצע אלגוריתם למיזוג שני העצים המוציא כפלט עץ AVL המכיל את כל הערכים.
פתרון:
1. ניצור מערך ממויין A_1של סריקת $in-order$ על T_1. סה"כ $O(n)$
2. ניצור מערך ממויין A_2של סריקת $in-order$ על T_2. סה"כ $O(n)$
3.נמזג את A_1 וA_2 למערך ממוין אחד $O(n)$
4. נבנה עץ AVL באמצעות השאלה ממקודם (עם חציון) - $O(n)$
ולכן סה"כ זמן הריצה יהיה $O(n)$!

חלק IX

ערימות מינימום ומקסימום

* ראשית לפני שניגש לנושא נעיר כי ישנה אפשרות לייצג עץ בינארי באמצעות מערך! כיצד? נשמור את הנתונים במערך רמה רמה. כלומר ראשית הקודקוד, ואח"כ כל מי שברמה השנייה משמאל לימין ישמרו (בדומה לאופן בו מתבצעת סריקת BFS) . ובאופן כללי - הבן השמאלי של צומת ברמה ה-i נמצאת בתא ה-$2i+1$

הבן הימני של צומת ברמה ה־i נמצאת בתא ה־$2i+2$
ישנם תאים שיהיו ריקים! וזה בסדר כי אין לכל צומת בהכרח ילדים או ילד אחד.
אז מדוע לא? הרי חישובי האינדקסים ולהבין כיצד העץ נראה היא פשוטה. התאים הריקים הם אלו הבעייתיים! בואו נדמיין את השרוך הגרוע ביותר מבחינתנו ־ השרוך הימני , יהיה מערך עם המון תאים ריקים!
נשים לב כי אם יש לנו מידע של n תאים, ידרש מקום של 2^n-1! זה סדר גודל אקספוננציאלי.
כלומר סה"כ זו לא שיטה יעילה (לרוב) לייצוג מערך.
מתי זה כן יעיל? בעץ שלם ־ בעץ שלם נזכר כי לכל צומת יש בדיוק 2 ילדים וכל העלים באותה רמה. בעץ כזה כל המידע אכן ינוצל ויהיה יעיל להשתמש במימוש זה!

הגדרה: עץ בינארי כמעט שלם ־ עץ בו כל הרמות מלאות, פרט *אולי* לרמה התחתונה בה קיים רצף ברמה התחתונה משמאל לימין. כלומר ־ פרט לרמה האחרונה מדובר בעץ שלם, וברמה התחתונה יש עלים שמגיעים משמאל שגורמים לו להיות לא שלם אבל התנאי הוא שהם מגיעים משמאל לימין ברצף. גם בעץ בינארי כמעט שלם כדאי ומומלץ לייצג באמצעות מערך! (בעץ כזה לא נקבל חורים בכלל במערך)
טענה: גובה העץ של עץ בינארי כמעט שלם יהיה $O(logn)$.

ערימת מינימום:

מבנה נתונים מופשט שיוגדר ע"י עץ בינארי כמעט שלם ויתמוך בפעולות הבאות:
א. $Creat-heap()$: אתחול, יוצר ערימה ריקה.
ב. $Insert(x,Q)$ ־ הכנסת איבר x לערימה.
ג. $min(Q)$־ מציאת האיבר עם המפתח הקטן ביותר
ד. $delete-min(Q)$ ־ הוצאת האיבר עם המפתח הקטן ביותר.
מדובר בתור עדיפויות, לכל אחד ניתן מס' והאיבר שיש לו עדיפות לצאת קודם לפני כולם הוא זה שקיבל את המס' הקטן ביותר (המחשה נחמדה ־ הקרנת סרט בקולנוע, כולם עומדים בתור ואז מגיע יהודה לוי ועוקף את כולם. למה? כי הוא קיבל כרטיס VIP (מס' נמוך יותר....))
*הדיון עבור תור מינימום זהה לתור מקסימום.

מימוש ־

ניתן וקל לממש את הערימה באמצעות עץ חיפוש כלשהו ($avl, B-trees$) וכו'.. כל הפעולות יהיו בעד $logn$ עם קבוע 1.44 קטן. האם אפשר לעשות זאת טוב יותר? כן!
א. נדרוש עץ בינארי כמעט שלם
ב. לכל צומת v נדרוש כי המפתחות של שני הבנים גדולים מהמפתח של v (כלומר, אנחנו לא מדברים על עץ חיפוש בינארי וזה נקודה חשובה).
כעת נשים לב שמהתכונות הללו יהיה קל למצוא את איבר המינימום ־ הוא יהיה בוודאות האיבר הראשון בעץ.

.1 $Min(Q)$

תמיד זה יהיה האיבר הראשון בעץ, ובאחסון באמצעות מערך נשים לב שהמימוש הפשוט יהיה $return-Q[0]$, כלומר אנחנו מדברים על סדר גודל של $O(1)$.

.2 $delete-min(Q)$

להחזירו כמו שאמרנו זה קל , אבל נרצה למחוק ונוצר חור במערך במיקום הראשון. נשים לב שתמיד(!!!!) בשביל לסגור את החור שנוצר כל מה שיש לעשות הוא לקחת את הערך שנמצא ברמה

האחרונה בצד הימני ביותר. ואז עדיין ישאר לנו עץ בינארי כמעט שלם, אך התכונה השנייה לא נשמרה ־ מה שנצטרך לעשות הוא לגלגל את העץ כלפי מטה עם הערך ששמנו בראש העץ, כך נמשיך באופן רקורסיבי עד שנקבל את התכונה השנייה. כמה זה יעלה לנו בסה"כ? $O(h) = O(logn)$ הקטנו גם כאן את הקבוע של $logn$ להיות בדיוק 1. (!!)

פסודו ־

$delete - min(Q):$
$Q[0] = Q[size(Q) - 1]$
$size(q) = size(q) - 1$
$heapify - down(Q, 0)$
$heapify - down(Q, i):$
$l = left(i)$
$r = right(i)$
$smallest = i$
$if(l < size(q) - and - q[l] < q[smallest])then - smellest = l$(כלומר אם יש לך בן שמאלי וגם הערך של הבן השמאלי קטן משלך שמור את האינדקס של הבן השמאלי
if(r<size(q) and q[r]<q[smallest])then smellest=r
if(smallest>i) then (כאן אנחנו שואלים האם בכלל האבא קטן משני בניו, ואם כן נפסיק כי הגענו לעץ טוב)
q[i]$\Longleftrightarrow Q[smallest]$
$heapify - down(Q, smallest)$

ובאופן כללי הגלגול יתבצע במילים כך: נשאל מי גדול יותר מהבנים ונלך לקודקוד שערכו נמוך ביותר. נגיע לצומת חדשה, כך נמשיך ונבחר עד שנקבל עץ בסדר.

3. הכנסת איבר ־ $Insert(q, x)$

את האיבר החדש נכניס תמיד במיקום $size$ ונגדיל את $size$ כמובן וכך אכן שמרנו על העץ עץ בינארי כמעט שלם. כעת צריך לבדוק את התכונה השנייה . כיצד? נבדוק האם הוא גדול מאבא שלו, אם גדול מעולה שמרנו על כל תכונות העץ. אחרת? צריך לבצע החלפה. אנחנו נמשיך לפעפע אותו עד כלפי מעלה עד שנעצור ונשים לב שגם כאן העלות תהיה $O(h) = O(logn)$ עם מקדם בדיוק 1!

פסודו ־

insert(Q,k):
Q[size(q)]=k
heapify-up(q,size(q))
size(q)++
heapify-up(A,i):
while i>0 and A[i]<A[parent(i) do A[i]$\Longleftrightarrow$ A[parent(i)], i=parent(i).

*נשים לב שבערך חצי מהערכים הם עלים, ולכן בערימת מינימום קשה לחפש את המקסימום ויקח בערך $\frac{m}{2}$ פעולות למציאתו בהינתן שיש m איברים.

מיון ערימה ־ $HeapSort$:

במשפט ־ הכנס את האיברים שברצונך למיין לערימה, צור ערימה במערך. בצע הוצאת מינימום והכנס אותו למקום אחרון במערך.

המוטיבציה היא שלהכניס n איברים יעלה $nlogn$ אך אם אנחנו יודעים את האיברים מראש זה יעלה פחות. אנחנו נראה כיצד בהינתן שאנחנו יודעים את כל האיברים מראש אנחנו יכולים ליצור מערך ערימה בזמן של $O(n)$.

* ראשית, נרצה ליצור ערימה. נמקם איברים במערך ונתחיל פעפוע מהעלים כלפי מעלה עד שנקבל ערימה. כמה זה יעלה? בוודאות פחות מ$nlogn$ כי נניח ויש n פעפועים כאלו וכל פעפוע עלה $logn$ ־ לא נעבור את החסם העליון הזה. לפי האלגוריתם אנחנו מתחילים לעבוד בערך מהאינדקס הפנימי האמצעי במערך (כי התחלנו מהקודקוד הפנימי האחרון במערך מימין), ומתחילים לנוע שמאלה בערכים בטיפול שלנו בעץ.

לצורך הדוגמה נניח כי יש 4 רמות בעץ - ונתחיל מהרמה האחרונה לפעפע. בערך יש שם $\frac{n}{2}$איברים ונבצע פעולה אחת עבור כל אחד, ברמה מעל יש $\frac{n}{4}$איברים ונבצע בערך 2 פעולות עבור כל אחד. כך עד האיבר ברמה הראשונה, עבורו נבצע בערך $logn$ פעולות. (מדוע 1 ואז 2 ואז 3...? זה כמה החלפות עשינו לקודקוד הנ"ל בשביל להגיע לרמה הזו בוורסט קייס). סה"כ נקבל את הסכום הבא -

בואו נהיה לארג'ים ונשים לב כי יש **לכל היותר** $\frac{n}{2}$איברים שעליהם מבצעים $heapUp$ בגובה 1. יש לכל היותר $\frac{n}{4}$איברים שעליהם מבצעים $HeapUp$ בגובה 2. כך ימשיך ונקבל....

$$O(\frac{n}{2}*1+\frac{n}{4}*2+\frac{n}{8}*3+.......) = n\sum_{h=1}^{logn}\frac{h}{2^h} < O(n\sum_{h=1}^{\infty}\frac{h}{2^h}) = O(n)$$

כלומר לבנות ערימה יקח $O(n)$ פעולות!!!!

כעת סה"כ סיבוכיות המיון ערימה תהיה כדקלמן -

* בניית ערימה $O(n)$

* הוצאת איברי המינימום אחד אחרי השני $O(nlogn)$

סה"כ $n + nlogn = O(nlogn)$

****גלעד יותר מרמז לעבור למבחן על תור מקסימום!**

**** כדאי לזכור שבניית ערימה היא טריקית והיא $O(n)$**

תרגול - ערימת מינימום ומקסימום

1. ערימת $min - max$:

הבעיה: הציעו מבנת התומך בפעולות: בנייה – בזמן $O(n)$. הכנסה – בזמן$O(logn)$. החזרת מינימום – בזמן $O(1)$. החזרת מקסימום – בזמן $O(1)$. הוצאת מינימום – בזמן $O(logn)$. הוצאת מקסימום – בזמן $O(logn)$.

פתרון: נבנה ערימת מינימום וערימת מקסימום. כל איבר יכנס לשתי הערימות - ויחזיק מצביעים דו כיוניים בין העותקים של האיבר בשני הרשימות. בעת הוצאת מינימום/מקסימום – הוצאה מהערימה התואמת קלה. הוצאה מהערימה השנייה תיעשה בעזרת שימוש במצביע.

2. ערימת חציון:

הבעיה: הציעו מבנת התומך בפעולות: בנייה – בזמן$O(n)$בהינתן n איברים ואיבר החציון שלהם). הכנסה – בזמן$O(logn)$. החזרת חציון – בזמן $O(1)$. הוצאת חציון – בזמן $O(logn)$.

פתרון: כאשר במבנה יש n איברים נרצה לשמור שתי ערימות - אחת תשמור את $\lfloor\frac{n}{2}\rfloor$ האיברים הקטנים והשניה את $\lceil n/2\rceil$ האיברים הגדולים מהחציון. החציון הוא תמיד האיבר הקטן יותר בחצי השני (של הגדולים) - זו תהיה ערימת מינימום (המינימלי מביניהם זה החציון) והשנייה תהיה ערימת מקסימום - כלומר המקסימלי הוא זה שקטן מהחציון וישר אחריו.

תמיד יתקיים - $|H_2| - 1 \leq |H_1| \leq |H_2|$.

א. החזרת החציון - תמיד נחזיר את החציון ע"י החזרת המינימום בחצי השני - וזה אכן $O(1)$.

ב. הכנסת איבר - נשווה לחציון תמיד - אם גדול ממנו נכניס לערימה השנייה ואחרת לראשונה. אם הכנסת איבר תפר את הכלל שמצויין מעלה אודות גובה הערימות - נוציא את האיבר המקסימלי/מינימלי - זה שבראש הערימה שהופר הסדר בה, ונכניס אותה לערימה השנייה לפני הכנסת האיבר החדש, אח"כ נכניס את האיבר החדש לערימה המתאימה לו - מס קבוע של פעולות על ערימות בינארות ולכן $O(logn)$.

ג. הוצאת החציון - נוציא את המינימום מהחצי השני, אם החצי השני נהיה קטן מדי, נוציא את המקסימום מערימת החצי הראשון ונכניס אותו לערימת החצי השני. מס קבוע של פעולות על ערימות בינארות ולכן $O(logn)$.

3. איבר k בגודלו ממערכים ממוינים

הבעיה: נתונים m מערכים $A_1, ...A_m$ממוינים בסדר לא יורד. כמו כן נתון מס' טבעי k כך שk

בוודאות קטן־שווה ממס' הערכים הכולל בכל המערכים. מצאו את האיבר ה־k בגודלו בכל המערכים יחדיו. עליכם לעמוד בדרישות סיבוכיות של ־ $O(m+klogm)$.

פתרון: נבנה ערימת מינימום בגודל m. מי יכנס אליה? $A_1[0], A_2[0], ...A_m[0]$ ־ האיבר הראשון בכל מערך. כמו כן נזכור מאיזה מערך הגיע הנתון. כעת נחזור על התהליך הבא ־ עבור $i=1$ עד $i=k-1$ יתקיים ־

א. הוצא את האיבר המינימלי מהערימה.

ב. הכנס את האיבר הבא במערך של האיבר שיצא.

לבסוף ־ כאשר אנחנו בפעם ה־K־ית, החזר את המינימום בערימה.

סה"כ $O(m)$ לבניית הערימה, וכן K פעמים של הוצאת איבר ־ $logm$ כלומר $O(klogm)$ ולכן סהכ ־ $O(m+klogm)$.

4. מיזוג ערימות בינאריות

הבעיה: בהינתן שני ערימות H_1, H_2 ־ כך שגודל כל אחת היא n. עליך לממש את מיזוג שני ערימות ־

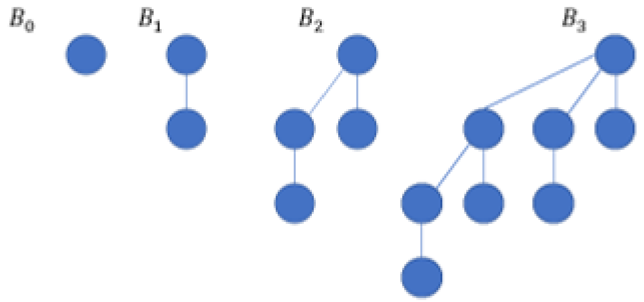
א. צור מערך חדש A בגודל $2n$. ־

ב. צור ערימה מהמערך החדש ־ $O(n)$

וסה"כ נקבל כי הסיבוכיות הינה $O(n)$.

עצים בינומיים

נגדיר ־ עץ בינומי מסדר k, כאשר B_kהוא עץ סדור. העץ B_0הוא שורש בודד. העץ B_kמקיים להלן ־ $B_k = B_{k-1} + B_{k-2}$ כאשר אחד מהם הוא הבן השמאלי של השורש של הראשון. כלומר ־



טענה: מס' הקודקודים בעץ B_kהוא 2^k.

הוכחה: נוכיח באינדוקציה על k.

בסיס ־ $2^0=1$ ואכן בB_0יש קודקוד אחד.

צעד: נניח נכונות עד B_{k-1}.לפי ההגדרה, B_kהוא שכפול פעמיים של B_{k-1}ולכן־

$$|B_k| = 2|B_{k-1}| = 2 * 2^{k-1} = 2^k$$

כנדרש.

טענה: גובה העץ B_k הוא k.

בסיס: טריוואלי, גובה 0 אכן באיבר בודד B_0.

צעד: נניח כי $h_{k-1}=k-1$ ונוכיח עבור H_k. לפי הגדרה ־ מתקיים כי תת העץ הגבוה ביותר בB_iהוא בכלל B_{i-1} ולכן אליו נוסיף את הקודקוד ונקבל ־

$$H_k = H_{k-1} + 1 = k - 1 + 1 = k$$

כנדרש.

טענה: יש בדיוק $\binom{k}{i}$ קודקודים בעומק i בעץ.
הוכחה: נוכיח באינדוקציה על עומק i.
בסיס: עבור עומק 0, כלומר האיבר הבודד אחד ־ מתקיים כי

$$\binom{k}{0} = \frac{k!}{(k-0)!0!} = 1$$

ואכן בעומק 0 יש איבר בודד שהוא הקודקוד של העץ.
צעד: נניח נכונות לעומק i. נרצה לבחון כמה קודקודים נוספו ־ כלומר להוכיח את הטענה עבור $i+1$.
נתבונן על עומק $i+1$. הוא מורכב מהקודקודים בעומק i בעץ הימני ומהקודקודים בעומק $i-1$ בעץ השמאלי, כלומר ־

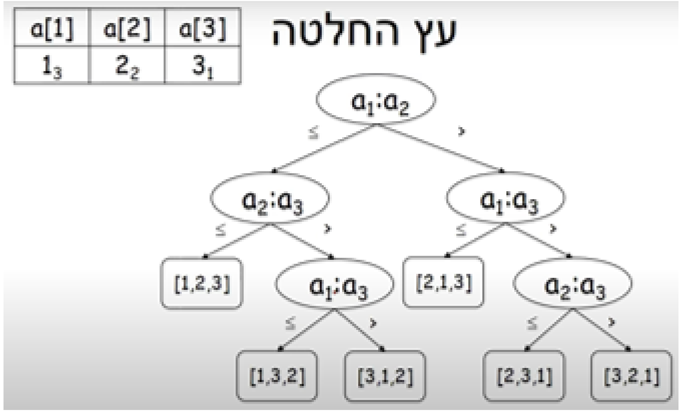
$$\binom{k}{i-1} + \binom{k}{i} = \binom{k}{i+1}$$

כשהמעבר האחרון הגיע כתוצאה מזהות פסקל ־ ראינו בבדידה.
טענה: הבנים של השורש הם $B_{k-1}, B_{k-2}, ...B_0$.

חלק X

טענה: כל אלגוריתם מיון מבוסס השוואות עורך לפחות $\Omega(nlogn)$ השוואות במקרה הגרוע.

(ולכן, מספיק להראות אלגוריתם שרץ באופן הרע ביותר $nlogn$ ומהטענה הזו הוא גם האופטימלי ביותר)
הוכחה: לכל מיון מבוסס השוואות ניתן להציג עץ החלטה המתאר את השאלות המתבצעות במהלך האלגוריתם. ננתח את עץ ההחלטה ונגיע לתובנה של המשפט. דוגמה לעץ כנ"ל ־



באופן זהה ניתן לכתוב עץ הכנסה לכל סוג מיון! הקודקודים הפנימיים בעץ מייצגים את השאלות, והעלים מייצגים את סידור האיברים. כלומר את התמורה של האיברים. נזכר כי בהינתן n איברים יש $n!$ תמורות. מסלול בעץ הוא ריצה מסויימת של האלגוריתם.
עץ החלטה הוא עץ בינארי (יש תמיד שתי אפשרויות) והוא עץ בינארי מלא!
מכאן שאם יש $n!$ עלים, ישנם בעץ מלא $n!-1$ צמתים פנימיים ולכן סה"כ נקבל כי יש $2n!-1$ צמתים בעץ ההחלטה.
גובה העץ ־ מס' ההשוואות שהאלגוריתם המיון מבצע במקרה הגרוע.
עץ ההחלטה לאלגוריתם המיון הטוב ביותר הינו העץ הנמוך ביותר האפשרי.

כיוון שבעץ ההחלטה יש $2n!-1$ צמתים, גובהו יהיה $\Omega(log(2n!-1))$
מכאן ־
$\Omega(log(2n!-1)) = \Omega(nlogn)$
כעת,
$n! = 1*2*3*...*n < n*n*n*n*....*n = n^n$
ולכן $log(n!) < log(n^n)$
ולכן $log(n!) < nlogn$
ומכאן ש $log(2n!-1) = \Theta(nlogn)$
במקרה הגרוע ביותר.
מש"ל.

חלק XI

הפרד ומשול ($Divide-and-conquer$)

אנחנו נדבר על שיטות שונות לחישוב פונקציית זמן ריצה של אלגוריתמים רקורסיביים.
הפרד ־ פצל את הבעיה לתתי בעיות זרות.
משול ־ פתור את תת הבעיות באופן רקורסיבי.
צרף ־ צרף את הפתרונות של התת־בעיות לפתרון הבעיה המקורית.
***לרוב נרצה לחלק את הבעיה ולנסות לכוון את הפתרון רק על רבע/שליש/חצי מהאיברים אם אכן יש אפשרות.**

--
*אין אלגוריתם מיידי שמעביר אותנו מנוסחה רקורסיבית לנוסחה סגורה. יש שיטות רבות לתרגם נוסחה מהצורה $T(n) = T(2n) + 3T(n^2) + 1$ וכו'...

שיטה ראשונה ־ עץ רקורסיה:

כפי שראינו במבוא והרבה בעבר. נפתח את העץ עד שנגיע לרמה סופית. נגדיר תמיד כי $T(1) = 1$ לרוב.
לצורך דוגמה נעבוד עם $T(n) = 2T(\frac{n}{2}) + 1$.
אם נפתח את העץ לפי רמות נקבל כך ־
1
1 1
1 1 1 1
....
וסה"כ בכל רמה h יהיה 2^h איברים
העץ יפתח כל עוד $\frac{n}{2^i} > 1 \iff n > 2^i \iff i = log_2 n$. כלומר, יהיו סה"כ $logn$ רמות, ברמה ה־$logn$ יהיה $2^{log_2 n} = n$ עלים, זה עץ שלם, הוא בפרט מלא, ולכן יש $n-1$ קודקודים פנימיים וסה"כ נקבל כי $T(n) = 2n-1 \in \Theta(n)$. (מבחינה מתמטית עלינו להוכיח באינדוקציה שאכן הכל כאן קורה, כלומר נוכיח באינדוקציה על n כי אכן זו הנוסחה).

שיטה שנייה ־ שיטת האיטרציה

זו שיטת ה"ניחוש", כלומר נציב הרבה פעמים עד שנקבל $pattern$ מסוויים, ואז נוכיח אותו באינדוקציה. בהמשך לנוסחת הנסיגה ־ $T(n) = 2T(\frac{n}{2}) + \Theta(n)$ נראה דוגמה־
$T(n) = 2T(\frac{n}{2}) + n$
$T(\frac{n}{2}) = 2T(\frac{n}{4}) + \frac{n}{2}$

$T(n)=2(2T(\frac{n}{4})+\frac{n}{2})+n=2^2T(\frac{n}{2^2})+2n$
סה"כ מתחילים לזהות דפוס - ולכן ברמה ה-i יהיה לנו - $T(n)=2^iT(\frac{n}{2^i})+ni$
ואם נקח $i=log_2n$ אכן $T(1)=1$ ואז נקבל כי - $T(n)=2^{log_2n}+nlogn=n+nlogn$
כלומר סה"כ $T(n)=nlogn+n$, עלינו להראות זאת בחסימה ע"י שני הצדדים, ונקבל כי $T(n)=\Theta(nlogn)$.

שיטה שלישית - נוסחת האב (שיטת המאסטר)

כפי שראינו הרבה, כמובן את הנוסחה לא נציין כאן. (מופיעה בתחילת הסיכום)
נתבונן בדוגמה -
$T(n)=T(\frac{2n}{3})+1$
$f(n)=1$ $b=1.5$ $a=1$
$n^{log_{1.5}1}=n^0=1$
מקרה 2 $\Longleftarrow$
ולכן סה"כ $T(n)=logn$

דוגמה ראשונה - אלגוריתם מיון מיזוג ($Merge-sort$)

הקלט: סדרה s של n איברים, והפלט יהיה סדרה s ממוינת בסדר עולה.
האלגוריתם: הפרד - פצל את s לשתי סדרות s_1,s_2 שבכל אחת $\frac{n}{2}$ איברים. משול: מיין את s_1 וs_2 באופן רקורסיבי. צרף - מזג את s_1 וs_2 הממוינות לסדרה אחת ממוינת.
פסודו -
$MergeSort(S):if(size(s)>1)partition(s)->(s1,s2)|MergeSort(s1)|mergeSort(s2)|s=Merge(s1,s2)$
נרצה לחשב את פונקציית זמן הריצה.
$T(n)=2T(\frac{n}{2})+\Theta(n)=2T(\frac{n}{2})+cn$
(כאשר c קבוע)
*כעת הניחוש שלנו הוא ש- $T(n)=\Theta(nlogn)$, טענה זו יש כמובן להראות באינדוקציה.

דוגמה נוספת - כפל מספרים (כפל מחרוזות)

הבעיה: נתונות שני מחרוזות בעלות n ביטים כל אחת. נרצה לכפול את המחרוזות. מה סיבוכיות הפעולה?
פתרון: מכפילים ביטים כמו שלמדנו בגן. יש n^2 מכפלות כאלו. נחשוב על הפתרון באמצעות רקורסיה. כלומר, במקום להכפיל מחרוזת באורך n נרצה לצמצם את הכפל למשהו כמו $\frac{n}{2}$. בהינתן מס' x נרצה לרשום אותו אחרת. נחלקו לשני חלקים ונקבל כי $X=x_1*2^{\frac{n}{2}}+x_2$. כאשר נכפיל ב-$2^{\frac{n}{2}}$ זה לא באמת עולה לי כי אנחנו רק מזיזים את הביטים. (יענו - בהינתן 1234. נוכל לרשום כי $1234=12*10^2+34$ - כלומר הכפל אכן לא באמת עולה)
כעת יש מס' נוסף - $Y=y_1*2^{\frac{n}{2}}+y_2$.
נראה כי אם נכפול נקבל -

$$xy=(x_1*2^{\frac{n}{2}}+x_2)(y_1*2^{\frac{n}{2}}+y_2)=x_1y_12^n+x_1y_22^{\frac{n}{2}}+x_2y_12^{\frac{n}{2}}+x_2y_2$$

מה קיבלנו כאן? נראה כי כל הפעולות של 2 בחזקת הן פעולות הזזת ביטים ולמעשה כאשר נכפיל שני מחרוזות צמצמנו את הבעיה שלנו ל-4 תתי בעיות בגודל $\frac{n}{2}$!
מכאן נקבל כי נוסחת הנסיגה היא -

$$T(n)=4T(\frac{n}{2})+O(n)=\Theta(n^2)$$

(נחסוך כמובן את ההוכחה - באינדוקציה)
זה לא טוב - מדוע? אפשר יותר טוב! זה בדיוק כמו הפתרון הנאיבי. בואו ננסה לצמצם את מס' הקריאות הרקורסיביות:
נגדיר -

$$A=x_1y_1,B=x_2y_2$$

נזכיר שנרצה לחשב את

$$C=(x_1+x_2)(y_1+y_2)=x_1y_1+x_1y_2+x_2y_1+x_2y_2$$

כעת נראה כי

$$xy=x_1y_12^n+x_1y_22^{\frac{n}{2}}+x_2y_12^{\frac{n}{2}}+x_2y_2=x_1y_12^n+2^{\frac{n}{2}}(x_1y_2+x_2y_1)+x_2y_2=x_1y_12^n+2^{\frac{n}{2}}(C-B-A)+x_2y_2$$

מכאן שקיבלנו נוסחה עם 3 איברים כעת (יש איזשהו קבוע באיבר האמצעי)! ולא 4 כמו קודם, נוכל לרשום -

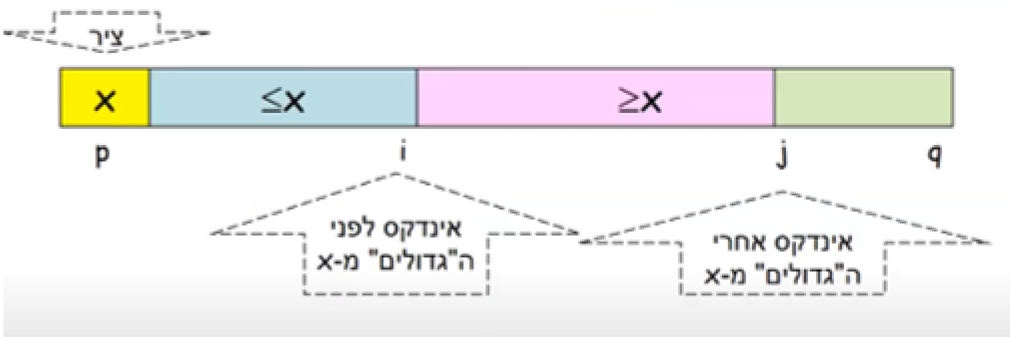
$$T(n)=3T(\frac{n}{2})+\Theta(n)$$

לפתור לפי מאסטר ולקבל כי $T(n)=\Theta(n^{log3})\approx\Theta(n^{1.58})$.

חלק XII
אלגוריתם *select*

המטרה - מציאת האיבר הi בגודלו.
הגדרה: יהי A מערך של n איברים שונים ויהי $1\leq i\leq n$. נאמר שהאיבר x הוא האיבר הi בגודלו במערך אם האיבר x גדול בדיוק מ$i-1$ איברים אחרים במערך A ($A[i]=x$).
*$i=1\Longleftarrow$ האיבר הקטן בגודלו
*$i=n\Longleftarrow$ האיבר הכי גדול.
$i=\lfloor\frac{n+1}{2}\rfloor$ או ערך עליון$\Longleftarrow$ האיבר הוא החציון במערך.
אלגוריתם נאיבי: מיין את כל איברי המערך והחזר את $A[i]$. עלות - $\Theta(nlogn)$.
האם אפשר לעשות זאת טוב יותר?
ניתן לעשות זאת ב - $O(n)$!
(נתחיל מפתרון שלא יעבוד)
נשתמש בעקרון ה*partition* - חלוקה. כלומר נקח איבר איבר ראשון שיהיה ה*pivot* - הציר שלפיו יתנהל המערך. כלומר מימינו כל מי שגדול ממנו ומשמאלו כל אלו שקטנים ממנו.
במפורט יותר - מתחילים את הריצה כאשר i ממוקם על האיבר הראשון וj על האיבר השני.מתחילים להשוות את ערך ערך j לערך i. כל עוד האיברים גדולים מ$a[i]$ נמשיך לקדם

את j. מתי נפסיק? כשנמצא ערך שקטן מi. מה נעשה לו? נקדם את i באחד, לאינדקס הבא, ואז כל שנצטרך לעשות הוא $A[i] \Leftrightarrow A[j]$, כך נמשיך עד שנקבל מצב שנוצר לנו מערך שבצד אחד ישנם כל האיברים הגדולים ומשמאל כל הקטנים.



דוגמה להחלפות הנ"ל -

כעת נוצר מצב שיש לנו $pivot$, אח"כ כל הקטנים ממנו ואח"כ כל הגדולים ממנו, את ה$pivot$ עצמו יש למקם גם כן, כל שנעשה הוא החלפה בין מיקום i למיקום 0 כלומר $A[0] \Leftrightarrow A[i]$.
כמה זה עלה לנו? בהינתן n איברים, מדובר ב $O(n)$.
(באופן זהה ניתן לעשות חלוקה עם $pivot$ רנדומלי, כאשר מה שנעשה הוא נבחר ערך רנדומלי ואז נעשה $A[0] \Longleftrightarrow A[randomIndex]$ ונמשיך כמו שעשינו כאן.

*** אלגורתם אקראי למציאת האיבר ה-i בגודלו: כאשר $q-p$ הוא אורך המערך.**

```
Rand-select(A,p,q,i)
if (p==q) then return A[p]
rand-partition(A,p,q)→ r
k=r-p+1
if(i=k) then return A[r]
if(i>k)
then return Rand-select(A,p,r-1,i)
else return rand-Select(A,r+1,q.i-k)
```

ניתוח זמן ריצה -

* במקרה הטוב - בדיוק n פעולות. נניח ורצינו את האיבר ה5 בגודלו ולאחר החלוקה הוא אכן ה5 בגודלו, אז סיימנו.
נניח ותמיד מדובר בחלוקה קבועה, למשל $\frac{9}{10}$ אזי $T(n) = T(\frac{9n}{10}) + n \Rightarrow \Theta(n)$
* במקרה הרע - n^2פעולות. מדוע? כל פעם בחלוקה הוא מחלק לנו לרוע מזלנו לא טוב - רק בפעם האחרונה נצליח למצוא את הk בגודלו שרצינו.
אבל אם זרקנו רק איבר אחד כל פעם , נקבל -

$T(n) = T(n-1) + \Theta(n) \Longrightarrow \Theta(n^2)$

זה לא טוב לנו!! גרוע אפילו מלמיין.

מה המסקנה שלנו? תמיד נרצה להעיף מהמערך חלק קבוע , כלומר נצמצם לפעם הבאה חלק שבר n־י קטן מ־1. נרצה $pivot$ שיזרוק שבר אן־י ונקבל סיבוכיות לינארית.

הפתרון שיעבוד:

```
Select(A,i,n)
  1. if n<5 return ith element of A
  2. Divide the n elements into groups of 5. Find the median of each 5-element group by sort.
  3. let B array of the medians of (2)
  4. x=select(B,⌈n/10⌉,⌈n/5⌉)
  5. partition around the pivot x. k=rank(x)
  6.if i=k return x
  7. if i<k then recursively SELECT the ith smallest element in the lower part
  else, then recursively SELECT the(i-k)th smallest element in the upper part
```

* ראשית נשים לב ש5־7 הם בדיוק כמו שעשינו קודם. חלק 1־4 מוצאים את ה$Pivot$. נעיר כי כל פעולה שנעשה על קבוצה בגודל 5 תהיה ב$O(1)$ ולכן מיון עבורה הוא מס' קבוע.

נחלק לקבוצות של 5, נמיין במס' קבוע ונמצא את החציון בכל חמישייה ־ הוא האיבר השלישי בכל חמישייה. כמה חציוניים כאלו יש? בערך $\frac{n}{5}$. נקח אותם ונפעיל ברקורסיה שוב ונמצא את החציון של החציונים. נקרא לו x. לאחר שסידרנו קיבלנו שיש כ־$\frac{n}{10}$ איברים שקטנים מהחציון x.

מה זה עוזר לנו? במקור כל האיברים סודרו בחמישיות, אם החציון שלהם קטן מאיקס גם כל השאר בחמישיות, ואם גדול מאיקס הם גדולים שווים מאיקס.

כעת בכל חמישייה יש 3 איברים שאינם רלוונטיים , ולכן סה"כ $\frac{1}{2} * \frac{3}{5} = \frac{3}{10}$אינם רלוונטיים עוד, כלומר מיפינו את המערך כדקלמן ־ יש לנו את הערך x , יש $\frac{7n}{10}$ שגדולים ממנו ו$\frac{3n}{10}$ שקטנים ממנו.

כעת נקבל את נוסחת הנסיגה להלן: יש לנו $\frac{n}{5}$ חמישיות וכל אחת מהן נמיין (מס' קבוע של איברים). נבחר כציר את חציון החציונים ונרצה למצוא את האיבר שהוא חציון החציונים ־ כלומר יש קריאה רקורסיבית לתת בעיה בגודל $\frac{n}{5}$. כמו כן ־ לפחות חצי מהקבוצה שיצרנו תהיה גדולה מחציון החציונים ־ דהיינו $\frac{n}{10}$ לפחות מהקבוצה גדולים ממנו, וכן סה"כ חציון החציונים יהיה גדול מ

$$\frac{n}{5} + \frac{n}{10} = \frac{3n}{10}$$

מהאיברים, ולכן צמצמנו את הבעיה לגודל חדש של $\frac{7n}{10}$.

נקבל כי ־

$$T(n) := \left\{ \begin{array}{cc} O(1) & n < 50 \\ T(\frac{n}{5}) + T(\frac{7n}{10}) & n \geq 50 \end{array} \right\}$$

איך נפתור? נוכיח באינדוקציה ־ $T(n) \leq cn$

בסיס: $n = 1$, נקבל כי הסיבוכיות היא $O(1)$ וזה תמיד נכון.

צעד : נעזר באינדוקציה שלמה ונניח נכונות לכל $\geq n$.

לפי הנחת האינדוקציה, $T(n) \leq \frac{1}{5}cn + \frac{7}{10}cn + n = \frac{9}{10}cn + n = cn - (\frac{1}{10}cn - n) \leq cn$

ולכן נרצה כי $-(\frac{1}{10}cn - n) \leq 0$

$\Longleftrightarrow$

$\frac{1}{10}c - 1 \geq 0$

$\Longleftrightarrow$

$c \geq 10$

ונקבל כי אכן כדרוש.

**** הערה - לא ניתן לעשות עם שלישיות. ניתן לעשות רק עם מס' אי זוגי בשביל לקבל חציון. כמו כן מיון שלישיה יתן לנו $O(nlogn)$ וזה לא טוב. ניתן לעשות שביעיות, תשיעיות וכו'.... אך הקבוע יגדל! ולכן חמישיות הכי יעיל.**

הפרד ומשול - בעיות מהתרגול

1. מערך עולה ויורד

נתון מערך $A=[1..n]$ כך שידוע שקיים $1 \le k \le n$ כך שהמערך "עולה" ב k התאים הראשונים, ויורד אחריהם. כלומר, לכל $1 \le i < k$ מתקיים $A[i] < A[i+1]$ ולכל $k \le i < n$ מתקיים $A[i] > A[i+1]$.

הצע אלגוריתם המוצא את k = האינדקס של איבר המקסימום.

פתרון: נראה כי k הוא האיבר היחידי שמקיים שהוא נמצא בין שני איברים שגדולים וקטנים ממנו. נקח את הערך האמצעי במערך. אם האינדקס האמצעי הוא k בדיוק נחזירו ב-$O(1)$. אם האינדקס שלנו מקיים $A[k-1] < A[k] < A[k+1]$ אזי אנחנו במגמת עלייה ולכן נפעיל רקורסיבית את הפתרון על החלק הימני של המערך. באופן דומה לחצי השמאלי. כלומר בכל פעם אנחנו נבדוק, אם אנחנו במגמת עלייה נפעיל רקורסיבית על ימין אם בירידה נפעיל רקורסיבית על שמאל ובסוף נסיים. (הרי המטרה למצוא את המקסימום ולכן נרצה להגיע תמיד לחלק הגדול יותר.

סיבוכיות -

$$T(n) = T(\frac{n}{2}) + O(1)$$

ולפי מאסטר זה $\Theta(logn)$

2. תת מערך כבד ביותר

קלט: מערך $A=[1..n]$ של n מספרים שונים.

פלט: תת-מערך $A=[i..j]$ שסכום איבריו גדול ככל הניתן. תת-מערך כזה ייקרא "כבד ביותר". ייתכן שיש כמה כאלו באותו הגודל.

פתרון: נשים לב שלפחות אחת משלוש האפשרויות נכונה:

1.קיים תת-מערך כבד ביותר שנמצא כולו בחצי הראשון של המערך.

2. קיים תת-מערך כבד ביותר שנמצא כולו בחצי השני של המערך.

3. קיים תת-מערך כבד ביותר שמתחיל בחצי הראשון ומסתיים בחצי השני.

לכן: נמצא תת-מערך כבד ביותר שמוכל בחצי הראשון נמצא תת-מערך כבד ביותר שמוכל בחצי השני נמצא תת-מערך כבד ביותר שמתחיל בחצי הראשון ומסתיים בחצי השני. לבסוף, נחזיר תת-מערך כבד ביותר מבין אלה שמצאנו (אם מצאנו יותר מאחד, נחזיר אחד מהם).

את שני תתי-המערכים הראשונים - נמצא ברקורסיה. כל פעם כמובן נשווה למקסימום.

איך מוצאים את תת המערך שמתחיל בחצי הראשון ונגמר השני? נמצא את הסיפא ואת הרישא בכל חלק (תחילית וסופית) ונחברם- זה יקח לנו $O(n)$.

מכאן נקבל שנוסחת הנסיגה הינה -

$$T(n) = 2T(\frac{n}{2}) + n \Longrightarrow \Theta(nlogn)$$

פתרון יותר טוב: נסמן את שני חלקי הקטע הימני ב-x ו-y. כל רישא של חצי היא אחת משתיים. כלומר כל רישא היא רישא של x או של כל החצי x ושאריות מy. נרחיב את הקריאות לרקורסיה שלנו כך שיחשבו גם: משקל כל המערך שהתקבל ברקורסיה (זה מס' קבוע של פעולות כי מתחילים

בסוף בחישוב האחורי ממערך בגודל 1), משקל הרישא הכבדה ביותר וכן משקל הסיפא הכבדה ביותר. משקל רישא כבד הוא המקסימום בין משקל רישא כבדה של החצי השמאלי+ משקל כל השמאליx ועוד המשך מהימני. כנ"ל על סיפא..... לכן נקבל את הנוסחה -

$$T(n) = 2T(\frac{n}{2}) + O(1) \Longrightarrow \Theta(n)$$

3. העלאה בחזקה

קלט: נתונים שני מספרים $a \in \mathbb{R}$ וכן $n \in \mathbb{N}$. **פלט:** a^n.
נאיבי: לולאה שמאותחלת ל-1 ומכפילים n פעמים ולכן $O(n)$.
נגדיר -

$$a^n = \left\{ \begin{array}{cc} 1 & n=0 \\ a^{n/2} * a^{n/2} & 2|n \\ a^{\frac{n-1}{2}} * a^{\frac{n-1}{2}} * a & otherwise \end{array} \right\}$$

מכאן שנקבל את הנוסחה הבאה:

$$T(n) = T(\frac{n}{2}) + O(1) \Longrightarrow O(logn)$$

לכאורה - לכל הפחות לא שיפרנו כלום כי אם נקבל 2^n נקבל שיש לעשות n פעולות הזזת ביטים ולכן זמן הכרחי הוא $\Omega(n)$ ולכן באסה לנו.

4. חישוב מס' פיבונאצי

קלט: מס' שלם n
פלט: החזרת המס' הn-י בסדרת פיבונאצ'י.
נאיבי: רקורסיה עם הנוסחה - באסה כי זה יתן לנו $O(2^n)$
פתרון טוב: נשים לב כי

$$\left(\begin{array}{c} f_{n+1} \\ f_n \end{array} \right) = \left(\begin{array}{cc} 1 & 1 \\ 1 & 0 \end{array} \right) \left(\begin{array}{c} f_n \\ f_{n-1} \end{array} \right)$$

מדובר בנוסחה רקורסיבית עם וקטור! נראה כי באופן כללי מתקיים

$$\left(\begin{array}{c} f_{n+1} \\ f_n \end{array} \right) = \left(\begin{array}{cc} 1 & 1 \\ 1 & 0 \end{array} \right)^i \left(\begin{array}{c} f_{n+1-i} \\ f_{n-i} \end{array} \right)$$

כלומר מתקיים כי

$$\left(\begin{array}{c} f_{n+1} \\ f_n \end{array} \right) = \left(\begin{array}{cc} 1 & 1 \\ 1 & 0 \end{array} \right)^n \left(\begin{array}{c} 1 \\ 0 \end{array} \right)$$

זה העלאה בחזקה - ניתן להעלות בחזקה ולקבל שזה $logn$ ואז יש לנו כפל נוסף של שתי מטריצות בגודל 2×2 ולכן מס' קבוע. מכאן שסה"כ מצאנו מס' פיבונאצ'י ב$O(logn)$

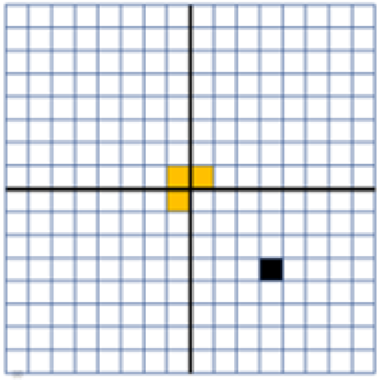
5. בעיית הריצוף

נתונים אריחים בצורת ר': ניתן לסובב כל אריח בכפולות של 90 מעלות. נתון לוח בגודל$n \times n$ כך ש$n = 2^k$ לk – מסוים. בלוח יש משבצת אחת "שרופה". הראו שניתן לרצף את כל הלוח, חוץ מהמשבצת השרופה, בעזרת אריחים כנ"ל.

קלט: לוח בגודל $n \times n$ כך ש$n = 2^k$ עבור k טבעי. וכן מיקום של משבצת אחת שרופה.

פלט: מיקום אריחים חוקי שממלא את הלוח.

פתרון: נחלק את הלוח ל־4 חלקים שווים זרים, כל אחד בגודל $2^{i-1} \times 2^{i-1}$, בחלק אחד יש משבצת שרופה. בשביל שהלוחות הקטנים יהיו מופע של אותה בעיה, צריך שגם בכל אחת מהם תהיה משבצת שרופה. נשים אריח בנקודה המשותפת לשלושת הרבעים ונתייחס למשבצות של האריח הזה כשרופות. כלומר ־



כאן בתמונה פירקנו את הבעיה כך שאכן יש רבע עם אריח שרוף, שכן כל אחד מהצהובים ב"ר" נשרף כשמיקמנו. תמיד נבחר את הר שמשותפת לכל הרבעים, ונפתור רקורסיבית לכל אחד מהרבעים.

סיבוכיות $T(n) = 4T(\frac{n}{4}) + O(1) \Rightarrow \Theta(n)$

6. שאלה על $select$

נתונות n נקודות על ציר המספרים (לאו דווקא ממוינות), ומספר טבעי $k \leq n$. הצע אלגוריתם יעיל ככל הניתן המוצא את k הנקודות הקרובות (מרחק = הפרש בערך מוחלט) ביותר לחציון.

פתרון: ראשית, נמצא את החציון.

לכל איבר נחשב את מרחקו מהחציון.

נמצא במערך המרחקים את האיבר ה־k בגודלו ־ נסמנו d_k.

נחזיר את כל האיברים שמרחקם מהחציון הוא לכל היותר d_k

זמן ריצה $O(n)$.

חלק XIII

טבלאות גיבוב ($Hash$)

מוטביציה קלה: זהו אחד ממבני הנתונים השימושיים ביותר במדעי המחשב. אנחנו נתמוך בשלוש פעולות בלבד ־ מחיקה, הכנסה וחיפוש ונראה כי כל אחד מהם יקרה ב־$O(1)$. הבעיה? זה קורה רק במקרה הממוצע. לכן לא נוכל לעשות פעולות נוספות על מבנה נתונים זה.

יש לנו 200 סטודנטים עם תעודות זהות. יש ת"ז 65432 ונרצה למצוא אותה. איך ניגש אליה בקלות? אפשר מערך, אבל הוא יהיה בגודל היוניברס וחבל כי יש רק 200 סטודנטים.

ראשית נגדיר מושג שנקרא Map ־ **מדובר בפונקציה** $h : U \to \{0, ...m\}$ כאשר $U = Universe$ של המפתחות וm הוא גודל הטבלה. מה נרצה? נרצה בהינתן $k \in U$ להיות מסוגלים לגשת אל המפתח $h(k)$ בקלות. יש מספר בעיות ־

1. מה אם המפתחות שלנו לא רציפים? למשל $pointers$, תעודות זהות וכדומה.
2. מה קורה אם לא כל המפתחות משומשים? למשל ־ מס' תעודת הזהות של הסטודנטים בכיתה. יווצרו חורים באמצע.
3. מה אם הפונקציה שלנו לא $bijective$ ־ חד חד ערכית ועל? כלומר יש מצב שיש שני ערכים שהולכים לאותו המקום.

לשם כך נכיר את המושג Hash tables.

אם U גדולה יותר מ־m, יתקיים כי $h(k)$ היא לא חד חד ערכית ועל. כלומר, יהיו קיימים $k_1 \neq k_2$ כך ש־$h(k_1) = h(k_2)$. **למצב זה נקרא התנגשות.** כלומר ־ אם מספר המפתחות גדול ממס' הערכים שניתן לשבץ, בוודאות תהיה התנגשות.

דוגמה: בהינתן טבלת הגיבוב ההבאה כאשר $m = 10$־

$$H(k) = k - mod - 10$$

$$H(k) : U \to \{0, ...100\}$$

ואז יתקיים כי ־ $h(12) = 12mod10 = 2 = 22mod10 = h(22)$.

ראינו כי אכן $m > |U|$ ולכן נוצרה התנגשות.

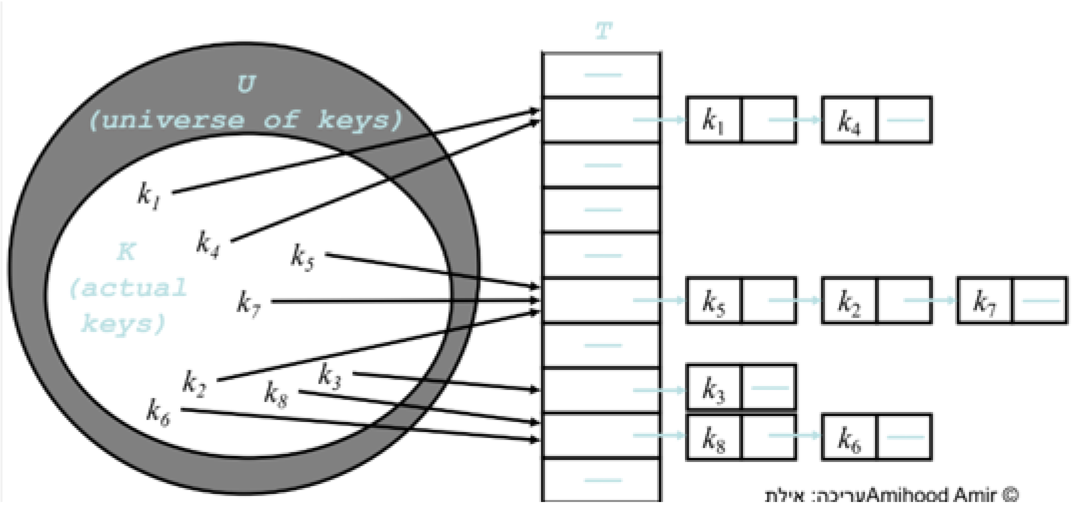
ראשית ־ איך נמצא פונקציית גיבוב?

הפונקציה הבאה נותנת חילוק אחיד יחסית ־ $h(k) = kmodm$. איך נבחר את m זו השאלה. מיהו ה־m הטוב? נראה כי אם המפתחות הם מספרים בבסיס b כלשהו ־ אם נבחר $m = b^p$ ניצור גיבוב אשר מתחשב רק ב$p = log_b m$ הספרות הפחות משמעותיות של המספר (אלו מימין), ואנחנו נרצה בדיוק ההפך ־ שהגיבוב יהיה תלוי בכמה שיותר מידע. לכן נבחר את m להיות מס' ראשוני שאיננו קרוב יותר מדי לחזקה של 2, כלומר שניצור גיבוב שמתחשב בכמה שיותר ספרות משמעותיות.

אוקי ־ אם יש התנגשות. או לחלופים ־ משתמש זדוני שולח מלא מפתחות עם אותו תא והוא רוצה להרוס לי את התוכנית. מה עושים?

1. פתרון ראשון ־ $Chaining$:

מדובר ב"שרשור". $chaining$ לוקחת את כל האלמנטים שמתאימים לאותו ערך ומכניסה אותם לתוך רשימה מקושרת.



פעולות שנוכל לעשות על הטבלה במצב התנגשות זה -

א. $Chined - Hash - Insert(T,x)$ - מכניס את הערך x לסוף הרשימה $T[h(key(x))]$.

ב. $Chined - Hash - Search(T,x)$ - מחפש אלמנט עם מפתח x ברשימה $T[h(k)]$.

ג. $Chined - Hash - Delete(T,x)$ - מוחק את האלמנט x מהרשימה $T[h(key(x))]$

מה באשר לזמן הריצה שלהם? במקרה הגרוע ביותר - חיפוש ומחיקה נוכל לעשות ב-$O(n)$, והכנסה ב-$O(1)$ כאשר נחזיק כמובן פוינטר לסוף הרשימה.

עם זאת, זה תלוי בשרשורים שיהיו. לכן נניח שיש גיבוב אחיד פשוט וכמו כן פונקציית הגיבוב שלנו h תמפה כל מפתח לחריץ כלשהו **בהסתברות אחידה ושווה.** לכן אנחנו ננתח מקרה ממוצע של זמן הפעולות.

ניתוח זמן ממוצע:

נניח כי n הוא מס' המפתחות, m הוא גודל טבלת האש, $\frac{n}{m}$ יהיה "גורם העומס" שלנו ונסמנו a. תמיד $a < 1$ וכן ככל ש-a קטן יותר יש פחות סיכוי להתנגשות.

כעת, בהינתן רשימה $T[j]$ אורכה יהיה n_j. מכאן שסכום כל אורכי הרשימות יהיה n (הגיוני) כלומר - $\sum_j n_j = n$.

הרי על פני הנייר, המקרה הגרוע ביותר הוא שכל האיברים התמפו לאותו המקום וכן האיבר לא ברשימה ולכן $O(n)$, אך זה לא יקרה בתוחלת. נראה כי התוחלת של n_j תהיה -

$$n_j = E[n_j] = \frac{n}{m} = a$$

מה זה אומר? עבור מפתח אחד - ההסתברות להגיע לתוך חריץ מסויים הוא $\frac{1}{m}$. **כעת שנחפש ערך בעץ (חדש - שאינו קיים)** נחשב את פונקציית הגיבוב $h(k)$ בזמן $O(1)$, נגיע אל רשימה מקושרת $T[h(k)]$ ששם המפתחות גם מפוזרים באופן אחיד, אורך הרשימה הנ"ל הוא n_j, זמן החיפוש בה הוא n_j וראינו מעלה שתוחלת זמן החיפוש בה הוא a. מכאן - זמן החיפוש של מפתח שאינו קיים בטבלה יהיה $\Theta(1+a)$. הסבר מפורט יותר, הסיכוי להגיע לתא ה-i היא $\frac{1}{m}$ ולכן נקבל כי

$$E[SEARCH] = n_1 * \frac{1}{m} + n_2 * \frac{1}{m} + ... + n_j * \frac{1}{m} = \sum_{i=1}^{j} \frac{n_i}{m} = \frac{1}{m} \sum_{i=1}^{j} n_i = \frac{n}{m} = a$$

נראה כי $\Theta(1+a)$ הוא מס' קבוע כי $a < 1$ ולכן אנחנו מבסוטים.

מדוע זה חשוב לחפש ערך שאינו קיים? למשל - כשנרצה להחזיק רשימת שמות משתמשים ונרצה לחפש האם השם הנ"ל כבר קיים במערכת או שלא לפני שנאשר ליוזר החדש להקרא בשם הזה - נרצה לסרוק את כל הטבלה ולבדוק האם אכן לא קיים שם המשתמש.

האם יש יתרון כלשהו בחיפוש ערך **שכבר קיים בטבלה?** מס הפעולות הצפוי תלוי במיקומו של המפתח בתוך הטבלה. אם הוספנו אותו בהתחלה יהיה קל יותר להגיע אליו ולהפך. לכן, בממוצע מס' האיברים שנבדוק בחיפוש הוא אורך הממוצע של הרשימה בזמן שהמפתח הוכנס פלוס אחד. ומאיפה בא האחד הזה? הוא ייצג את הנק' בה הוסיפו את האיבר החדש - נזכיר שלהכניס שווה $O(1)$. כלומר זה יותר טוב - נצטרך בממוצע לעבור רק על חצי מהטבלה (שוב, בהינתן איפה שהכניסו אותו).

באופן קצת יותר פורמלי - נניח שיש לנו מפתחות $k_1, ...k_n$ בסדר ההכנסה שלהם. כאשר נכניס את הערך k_i אורך הרשימה הצפוי יהיה $\frac{i-1}{m}$ (מדוע? בזמן הכנסת המפתח הi הוכנסו כבר $i-1$ מפתחות והסיכוי שהמפתח יגיע לאותו מקום בטבלה הוא $\frac{1}{m}$ ולכן הסיכוי ש$i-1$ יהיו במיקום הקודם שלהם הוא כדקלמן). לכן - **האורך הצפוי של חיפוש מוצלח הוא האורך הממוצע של הרשימות הללו -**

$$\frac{1}{n}\sum_{i=1}^{n}(1+\frac{i-1}{m}) = ... = 1+\frac{a}{2}-\frac{1}{2m}$$

זה כפי שראינו קודם בהנחה שלנו - נצטרך לעבור באופן ממוצע רק על חצי מהטבלה. מכאן שהמסקנה שלנו היא שזמן החיפוש הממוצע יהיה $\Theta(1+a)$.

***הערה - למען היעילות בדרך כלל בוחרים את** m **ב-**$O(n)$ **כלומר שיהיה פורפורציונלי לגודל הטבלה. כלומר בהינתן** $|U| = n$ **נרצה כי** $m = cn$ **כאשר** $c > 0$.

ולכן - כאשר נתכנן את הטבלה שגודלה יהיה פורפורציונלי למס' האיברם נקבל שכל הפעולות בטבלה מתבצעות בזמן קבוע בממוצע. כלומר - חיפוש, מחיקה והכנסה יתבצעו באופן ממוצע ב-$O(1)$**!**

לסיכום - תמיד מתקיים $E[search] = \Theta(1+a)$ **ולא משנה לנו אם האיבר כן נמצא בטבלה, או שלא.**

כעת נתבונן בבעיה - ומה אם כל המפתחות בטבלה ילכו לאותו הערך? אם כולם ימופו בטבלת הגיבוב לאותו הערך, החיפוש יהפוך להיות $O(n)$. איך נפתור את זה? הפתרון המקובל - להשתמש באקראיות בפונקציית הגיבוב. אבל אז תתעורר בעיה חדשה - איך נדע איזה ערך גיבוב מתאים לכל מפתח בזמן החיפוש?

הפתרון - $Hashing\ Universal$**:**

המטרה: שלא בכל פעם שנגבב מפתחות הם ילכו לאותו מפתח ויתנגשו (סטטיסטית). הביצוע: נשתמש במספר פונקציות גיבוב שונות ונבחר מהם באופן אקראי.

הגדרה: אוסף H של פונקציות המקיימות $f\in H, f: U \to \{0,..m\}$ הוא $Universal$ אם לכל זוג מפתחות $x,y\in U$ כאשר $x\neq y$ מס' הפונקציות בהם $h(x)=h(y)$ הוא לכל היותר $\frac{|H|}{m}$.

מדוע? נראה כי נקבל שההסתברות לבחור בפונקציה יהיה $\frac{\frac{|H|}{m}}{|H|} = \frac{1}{m}$, שזה כמובן טוב לנו.

כלומר - אם נבחר פונקציה $h\in H$ באופן אקראי הסיכוי שתהיה התנגשות בין x לy יהיה לכל היותר $\frac{1}{m}$.

מוטיבציה - גוגל בוחרת בעצמה פונקציות האש כאשר משתמש מחפש, יש המון כאלו רנדומיות. כך שלא יתכן שיבוא מישהו שינסה לדפוק לי את התוכנית ויצליח, כי ממפים את המידע בפונקציות האש שונות.

נבחר מס' ראשוני p כך ש $|U| < p$. נבחר שני מספרים, a ו-b שיהיו מ$\{1,...p-1\}$ ואז המשפחה תהיה

$$h_{a,b}(k) = ((ak + b) - modp)modm$$

יש לנו p^2פונקציות, כיוון שבחרנו סה"כ $p \times p$ אפשרויות.

משפט: נניח כי $h \in H$ נבחרה באקראיות, מס' ההתנגשויות הממוצע עם מפתח כלשהו הוא לכל היותר $a = \frac{n}{m}$.

כלומר גם במקרה ובו יריב זדוני ואכזר יבוא וינסה להרוס לנו את הטבלה ויכניס מלא נתונים זהים ־ במקסימום מס' ההתנגשויות יהיה a. והמסקנה שלנו.... ההסתברות ששתי מפתחות יתנגשו יהיה בסה"כ $P(x) = P(y) = \frac{1}{m}$.

המשמעות־ לקיחת פונקציה אקראית ממשפחת הפונקציות לא תשנה את סיכוי ההתנגשות של שתי מפתחות, כלומר הוא יהיה כמו בפונקציה אקראית אמיתית. ישנן שתי תכונות מעניינות ־

א. אם $m = n$ אזי תוחלת מס' ההתנגשויות תהיה קטנה מ$\frac{n}{2}$ ובהסתברות גבוהה מס' ההתנגשויות יהיה קטן מ־n.

ב. אם $m = n^2$ תוחלת מס' ההתנגשויות תהיה קטנה מחצי ובהסתברות גבוהה ־ **אין כלל התנגשויות.**

נספח. איך מוצאים פונקציית גיבוב אוניברסלי? (מהדרייב ־ 'סיכום יאיר תשפ"ד'):

נאיבי: נגדיר את $\mathcal{H}$ להיות קבוצת **כל** הפונקציות מU ל$[m]$.הבעיה $|\mathcal{H}| = m^u$. לכן, כדי לשמור איזו פונקציה בחרנו מהקבוצה נזדקק ל$\log m^u = u \log m$ וזה הרבה יותר מדי מקום

פתרון בר מימוש – משפחת הפונקציות המודולריות:

- נבחר מספר ראשוני $p \geq m$ כלשהו, $p \in \theta(m)$.
- בחר באקראי $a, b \in [p]$ כך ש$a \neq 0$.
- נסמן $h_{a,b}(x) = ((ax + b) \bmod p) \bmod m$ ו- $\mathcal{H} = \{h_{a,b}\}$ המשפחה האוניברסלית.
- הערה: מתקיים $|\mathcal{H}| = p(p-1) \in \theta(p^2) = \theta(m^2)$ ולכן, בשביל לשמור את הבחירה מספיק $\log m^2 = 2 \log m$ ביטים – תא אחד בזיכרון.

משפחה כמעט אוניברסלית

הגדרה: $\mathcal{H}$ נקראת כמעט אוניברסלית אם עבור כל זוג מפתחות **שונים** $x,y \in U$ מתקיים:

$$\Pr[h(x) = h(y)] \leq \frac{2}{m}$$

כלומר, בהשוואה בין שני איברים – הסיכוי ש"יתנגשו", הוא לכל היותר פי 2 מהסיכוי אם הפונקציה נבחרת באקראי מכלל הפונקציות האפשרויות.

אחת הדוגמאות היא משפחת פונקציות hash הכפליות:

פונקציה במשפחה, $h_a(x)$ תוגדר בצורה הבאה:

נניח ש: w מייצג את הגודל של מילה מסוימת במחשב (למשל 64 bits).

נגדיר טבלה שהגודל שלה הוא $m = 2^k$ כאשר $k < w$.

וכן מתקיימים התנאים:

1. $0 < a < 2^w$
2. a הוא אי-זוגי

נחשב את הפונקציה בצורה הבאה:

$h_a : U = [2^w] \to [2^k]$

$h_a(x) = \lfloor \frac{ax \bmod 2^w}{2^{w-k}} \rfloor$

פתרון שני ־ Open Addressing

נרצה לפתור את בעיית ההתנגשויות ללא *Chining*. למה? כי זה מבזבז מקום (הרי יש מערך שבתוכו מערכים־סיבוכיות לינארית ($O(n^2)$) ופוינטרים.

היכן נשים את המפתח שגורם להתנגשות? בתוך הטבלת גיבוב עצמה. כלומר ־ כאשר נזהה התנגשות, נשמור את האיבר המתנגש בחריץ ריק בטבלה. באיזה חריץ נבחר? אנחנו נכניס לטבלה מפתח, אם הוא מתנגש עם מפתח אחר נחפש לו מקום חלופי בטבלה עד שנמצא מקום פנוי ואם אין כזה־ נחזיר שגיאה.

נסתכל על הפונקציית "גישוש" הבאה -

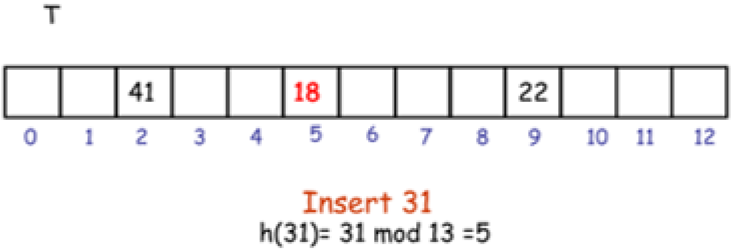
$$h : U \times \{0, ...m-1\} \to \{0, ...m-1\}$$

יש סדרה "מוגדרת מראש" של המיקומים האפשריים בטבלה. לכל מפתח יש רצף מוגדר ודטרמיניסטי של מיקומים - $h(k,0), ...h(k,m-1)$ כאשר $h(k,i)$ הוא המיקום שנבדק בנסיון ה-i עבור המפתח k. המפתח יכנס למיקום הראשון הפנוי מבין סדרת מיקומים זו. הפונקציה שתוארה מעלה מקבלת זוג סדור (מפתח k +מס' נסיונות ההכנסה הקודמים) ומחזירה את האינדקס החדש בו ישהה המפתח. אם נגיע למיקום האחרון ולא נמצא מפתח - נחזיר שגיאה כי כבר אין מקום בטבלה ויש $overFlow$.

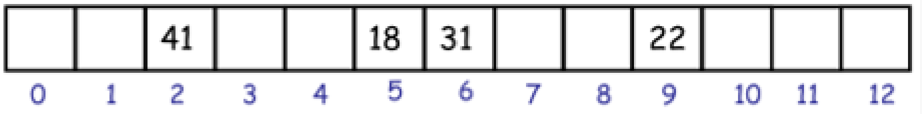
שיטה זו - גישוש לינארי - תמיד תנסה להכניס את המפתח בתא הבא הפנוי אחרי התא שבו ניסינו להכניס.

דוגמה -

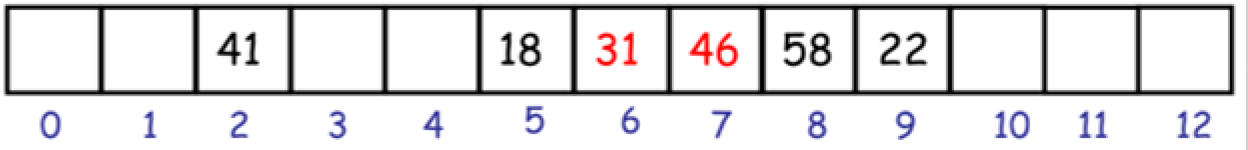
אנחנו עובדים עם פונקציית הגיבוב $H(k) = kmod13$: ננסה להכניס אל הטבלת גיבוב 13. עם זאת, הוא אמור לשהות באותו המיקום בו שוהה כעת 81. בעיה!



מה נעשה? נכניס בתא הבא -



כעת ננסה להכניס את 85. אך $58mod13 = 6$, מאוחסן שם ערך. ננסה את הערך הבא - גם ב7 מאוחסן ערך. אבל ב8 לא היה כלום - הכנסו לשם וסיימנו.



כאשר ננסה להכניס ערך במיקום 21 למשל ולא יהיה מיקום, נלך באופן מעגלי לתחילת הטבלה ונחפש שם - שגיאה תוחזר רק שלא יהיה מקום!

הערה: לא נוכל להכניס יותר מפתחות מגודל הטבלה ולכן $n \le m$ כלומר $a \le 1$ (נזכר כי m גודל טבלת הגיבוב וn הם כמה שמאוחסנים כעת)

חשוב לזכור כי מס' הגישושים שיהיו תלוי מאוד בפקטור העומס a. למשל כאשר $a = 0.5$ מס' הגישושים הצפוי יהיה 2 וכאשר $a = 0.9$ מס' הגישושים יהיה כ-01. למה אגב? כי 90% מהטבלה מלאה.

סיבוכיות:

* במקרה הגרוע - כפי שהערנו שניגשנו לנושא, $O(n)$.
* במקרה הממוצע -

א. חיפוש -

אם המפתח לא נמצא בטבלה אז $\Theta(\frac{1}{1-a})$ - ונחשוב על כך אם $a=\frac{1}{2}$ והטבלה חצי מלאה, אכן יקח באופן ממוצע 2 פעולות בשביל למצוא מקום ריק (הרי באופן הסתברותי, זה ריק-מלא-ריק-מלא...)
אם המפתח כן נמצא בטבלה אז $\Theta(\frac{1}{a}ln(\frac{1}{1-a}))$
-

בדיקות נפוצות - האם התא פנוי

א. בדיקה לינארית

כפי שהצגנו קודם, נבדוק האם תא תפוס. אם תפוס נלך לאחד אחריו. באופן פורמלי -

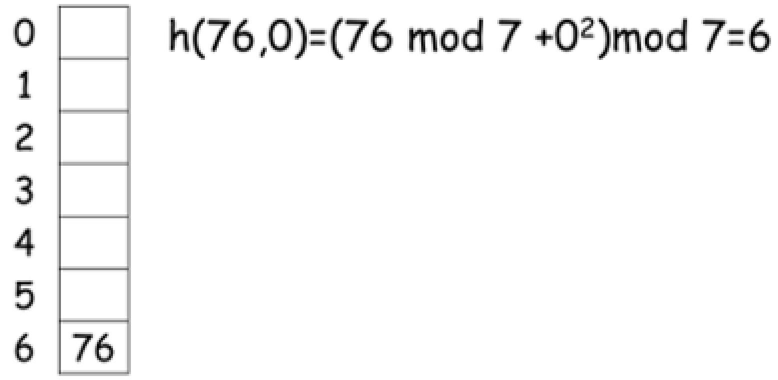
$$h(k,i)=(h'(k)+i)modm$$

כאשר $h'(k)$ פונקציית גיבוב רגילה.
יתרון: קל לממש.
חסרון: בעיית הצטברות ראשונית - בהינתן שלתא פנוי קודמים i תאים תפוסים - הסיכוי שיהיה התא הבא שיתמלא הוא $\frac{i+1}{m}$ ולא $\frac{1}{m}$ ואז יכולים להיווצר רצפים ארוכים של חורים. כלומר - כפי שראינו בדוגמה מעלה כאן בסיכום, יש רצף ארוך מצד אחד של מספרים ורצף ארוך של חורים והמודל ההסתברותי יעשה בעיות.
מכאן שלפי החסרון הזה בשיטה זו נקבל סיבוכיות גבוהה יותר בגלל ההצטברות הראשונית - במקרה הממוצע:
אם המפתח לא נמצא בטבלה אז $\Theta(\frac{1}{2}*(1+(\frac{1}{1-a})^2)$
אם המפתח כן נמצא בטבלה אז $\Theta(\frac{1}{2}*(1+(\frac{1}{1-a}))$

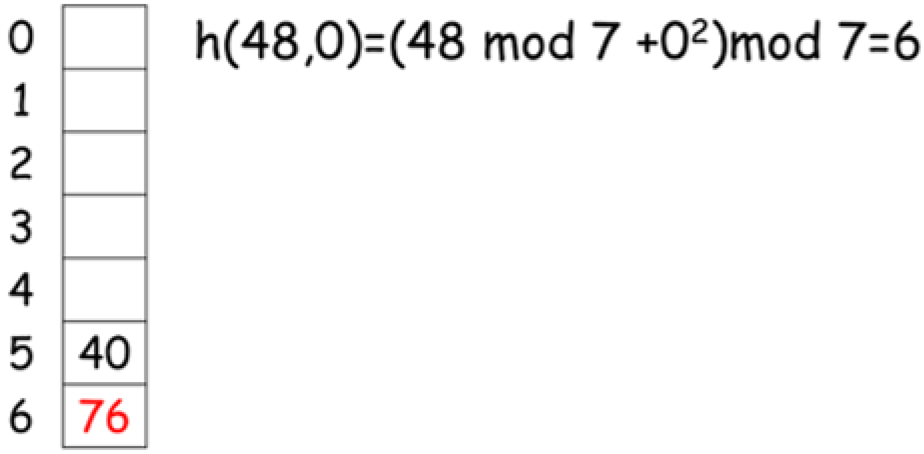
ב. בדיקה ריבועית

$$h(k,i)=(h'(k)+c_1i+c_2i^2)modm$$

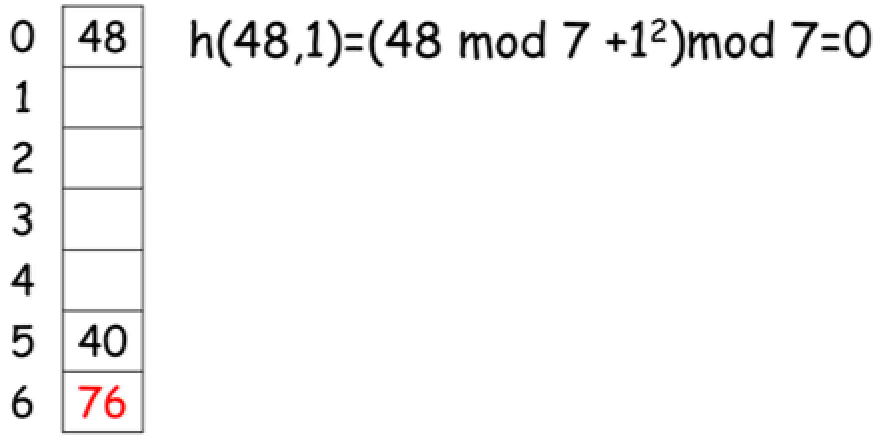
נדרוש כאן $c_2\neq 0$ וכמו קודם $h'(k)$ היא פונקציית גיבוב רגילה. זה יותר טוב מלינארי.
דוגמה - בהינתן הפונקציה הבאה: $h(k,i)=(kmod7+i^2)mod7$



כעת ננסה להכניס את 84 עם $i=0$:



בעיה ־ לכן נעבור לאינדקס $i = 1$:



ונראה כי הרעיון ברור. כל פעם מתחילים מאינדקס ומטפסים, עד שהטבלה מלאה.

3. Double Hashing:

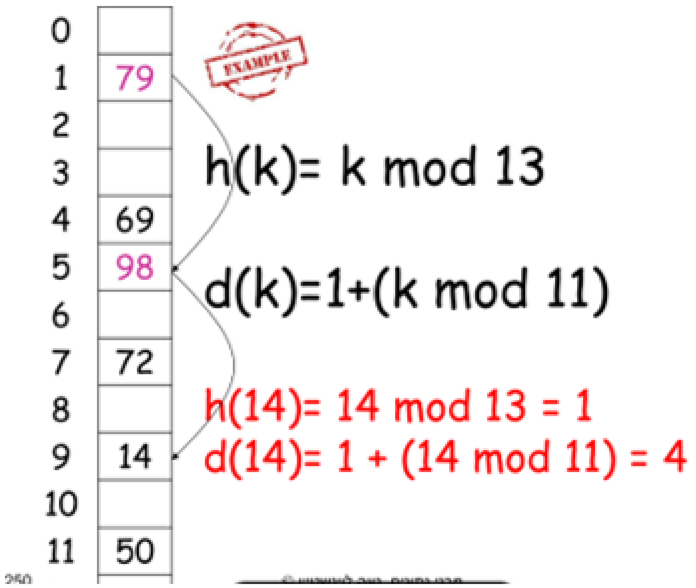
$$h(k,i) = (h_1(k) + ih_2(k))modm$$

כאשר h_1 נקראת פונקציית הבסיס ו־h_2 נקראת פונקציית הצעד.
נדרוש כי תמיד כל תא בטבלה ייבדק לאורך הדרך ולכן ־
א. $h_2(k) \neq 0$
ב. ל־$h_2(k)$ ול־m אין מחלקים משותפים הגדולים מ־1.
לכן ־ לרוב נקח m ראשוני ונגדיר כדקלמן ־

$$h_1(k) = kmodm, h_2(k) = (kmod(m - c)) + 1$$

כאשר $c > 0$ ונרצה שיהיה קטן.
דוגמה ־
נתבונן בפונקציות הבאות ־

$$h_1(k) = kmod13, h_2(k) = 1 + kmod11$$



ניתן לראות כאן בדוגמה את הנסיונות השונים עבור $i=0$ קיבלנו כי תא 1 תפוס, אח"כ $i=1$ קיבלנו כי תא 5 תפוס ורק כאשר הצבנו $i=2$ קיבלנו כאשר תא 9 פנוי ונשארנו בו.

הערה לשיטה זו בנושא מחיקה (רלוונטית לכל אפשרות 2 לזיהוי התנגשויות): כאשר נמחק איבר בטעות נוכל לחשוב שמפתח מסוים לא נמצא בטבלה. מדוע? כי רצף הגישוש שלנו ייעצר שנגיע לתא ריק. למרות שיתכן מאוד שיש מפתחות אחרים שמופו אחרי התא שמחקנו. לכן אנחנו נסמן תאים מחוקים באמצעות $flag$ כלשהו שלישי - חוץ מ"פנוי" ו"תפוס" - כך שאם יש לנו בזמן חיפוש תא ריק - החיפוש יעצר בו, אך אם נתקל במפתח מחוק החיפוש יימשך. רק שנמצא את המפתח או שמגיעים לתא ריק (שלא סומן כמחוק) - החיפוש ייעצר.

הערות בכללי להאשינג:

א. שתי השיטות מבוססות על מודלים הסתברותיים שהוכיחו באופן מתמטי שאכן מדובר במס' קבוע של פעולות.

ב. בשיטה השנייה אין סדר לוגי בו האיברים מסודרים, בניגוד לשיטה הראשונה. יתרון השנייה הוא שאינה מבזבזת מקום.

ג. בשתי השיטות צריך לשמור על פקטור עומס a סביר בשביל להבטיח ביצועים טובים.

סיכום מתומצת על האשינג

בעיית $2-SUM$

נתונה קבוצה S של n מספרים שונים, פונקציית גיבוב מושלמת $h:S\to\mathbb{Z}_n$ ומספר k , תאר אלגוריתם הבודק האם קיימים שני איברים שונים $x_i,x_j\in S$ כך ש$x_i+x_j=k$.

פתרון: נכניס את איברי S לטבלה באמצעות הפונקציה המושלמת. נעבור על כל האיברים, נחשב עבור כל איבר X_iאת הערך $k-x_i$ ונחפש בטבלת הגיבוב האם התוצאה קיימת בטבלה (חיפוש $O(1)$), אם כן יופי אם לא נתקדם. ככה נעבור על כל האיברים ויעלה לנו $O(n)$.

איך מוצאים פונקציה h מושלמת?

האם ניתן לקבוע את h כחלק קבוע כלשהו במבנה? לא! היריב תמיד יוכל לבחור לנו קבוצה רעה של מפתחות ולהרוס לנו. כי לפי עקרון שובך היונים חייבים להיות לפחות $\frac{|U|}{m}$ איברים שממופים לאותו הערך.

לכן נבחר את הפונקציה h באקראי בזמן הריצה.

גיבוב אוניברסלי: נסמן בH את האוסף הסופי של פונקציות הגיבוב הממפות את המרחב וכפי שראינו הסיכוי שעבור כל $x,y \in U$ יתקיים $h(x) = h(y)$ הוא $P < \frac{1}{m}$. כלומר לא ניתן למנוע התנגשויות לחלוטין אך ההסתברות להתנגשות מאוד קטנה.
נגדיר H היא קבוצת כל הפונקציות מU אל m . הבעיה היא ש $|H| = m^{|U|}$, כדי לשמור איזו פונקציה בחרנו מהקבוצה נצטרך $log(m^u) = ulogm$ מקום רק בשביל לשמור את אינדקס הפונקציה! זה יותר מדי מקום עבורנו וזה לא בא בחשבון.
מה נעשה? נקח מס' ראשוני גדול $p \geq m$, נבחר באקראי $a, b \in [p]$ (מהטווח של p–כלומר מ1 עד p) וכן $a \neq 0$.
נסמן:

$$H_{a,b}(x) = ((ax + b)modp)modm$$

ועכשיו בהצלחה ליריב הזדוני....
כעת $H_{a,b}$ היא משפחה אוניברסלית ומתקיים כי $|H| = p(p-1) = \Theta(p^2) = \Theta(m^2)$, זה כבר סדר גודל טוב עבורנו!

התנגשויות

נסמן $a = \frac{n}{m}$ והוא מספר האיברים הממוצע בטבלה. נראה כי אם $a > 1$ בהכרח יהיו התנגשויות וכן אם $a \leq 1$ ייתכן שלא יהיו התנגשויות וככל שa יהיה קטן יותר כך יהיו פחות התנגשויות.
1. שיטת השרשור: כל איברים שמתנגשים לאותו תא מוכנסים לרשימה מקושרת אחד אחרי השני. זמן הכנסה: $O(1)$, זמן חיפוש/הוצאה : תלוי בעומס בתא ולכן $O(1+a)$. בעיות: גישה לא רצופה בזכרון + יש תאים בהם זמן החיפוש גדול יותר. זמן חיפוש בתוך רשימה הוא תלוי באורך הרשימה ־ לא עפים על זה.
2. מיעון פתוח, יענו $open - addressing$: אם איבר רוצה ללכת לתא שכבר יש שם מישהו, נשלח אותו לתא אחר.
* כל האיברים יושבים בתוך טבלה ספציפית. אפשרי רק כאשר $a \leq 1$.
***אופציה אחת ־** נעבור על כל התאים $h(x,i)$ עד שנמצא את התא המתאים או תא ריק. במחיקת האיבר עם המפתח x ־ נחפש את האיבר אך **לא נמחק אותו** אלא נסמן * שנמחק מהתא ־ אחרת מה יקרה? אם ננסה לחפש בטבלת הגיבוב שלי איבר שקיים נעצור בתא שנמחק כי למה שנמשיך לחפש אחרי תא ריק? לכן נמשיך בחיפוש כשנראה *, כשנרצה להכניס את x_3 במקרה, אז נוכל להכניס לתא * ולמחוק את *. תוחלת זמן החיפוש וההכנסה הוא $O(\frac{1}{1-a})$ ותוחלת זמן חיפוש לאיבר שכבר בטבלה הוא $O(\frac{1}{a}ln(\frac{1}{1-a}))$.
* שוב, הנחת הגיבוב האחיד אינה מעשית. במקרה זה נניח שיש לנו פונקציה אחת ממשפחה אקראית.
־ דגימה לינארית: אם תפוס, לך לתא אחריו, אם הוא תפוס? לך לאחד אחריו וכך תמשיך.
־ דאבל האשינג: כמו שמתואר כאן בסיכום מעלה.
לכמה תאים אפשר להגיע בצורה זו?
נסמן $d = gcd(m, h_2(x))$. ־ המחלק הגדול ביותר של שני המספרים. החיפוש יעבור על m/d תאים. דרישה: לכל x, יתקיים $h_2(x)$ זר לm. כלומר $gcd(m, h_2(x)) = d = 1$ כלומר נדרוש שהמחלק הכי גדול יהיה 1, זה יקרה אם m הוא ראשוני.

גיבוב מושלם

כמו שאמרנו ־ אין דבר כזה במציאות אך נזרום עם זה שזה מושלם יחסית בהסתברות טובה.
מבנה נתונים לבעיית המילון התומך בגישה לאיבר בזמן $O(1)$ במקרה הגרוע נקרא "טבלת גיבוב מושלמת" (כלומר־ **אם המספרים ידועים לנו מראש, יש לנו רשימה קבועה של כל הסטודנטים**

מראש! **ולא נרצה להוסיף / להוציא סטודנטים**) אזי ניתן לעשות גיבוב מושלם. נבחר פונקצייה ממשפחה אוניברסלית H_m ונקווה לטוב - אם יהיו התנגשויות נתחיל מחדש (נבחר פונקציה אחרת). מה ההסתברות שנתחיל מחדש? אם נבחר $m = 2n^2$ נקבל כי הסיכוי לבנות טבלה ללא התנגשויות הוא לפחות $\frac{1}{2}$ ולכן, תוחלת הזמן הדרושה עד להצלחה היא

$$n\sum_{i=0}^{\infty}\frac{1}{2^i} \leq 2n = O(n)$$

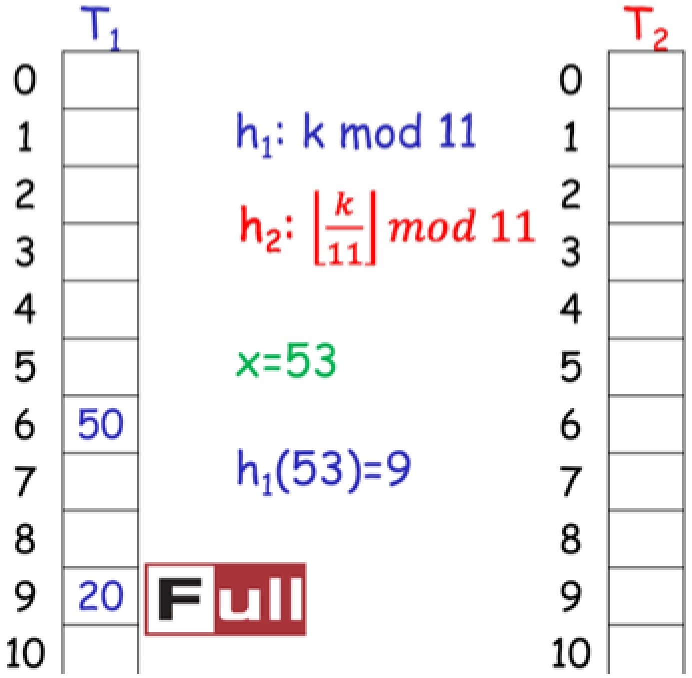
מה הבעיה? $2n^2$ זו טבלה גדולה מדי!
השיפור יהיה אלגוריתם FKS:
טבלה ראשית בגודל $m = n$. התנגשויות נפתרות ע"י מבנה עזר נוסף (כמו בשיטת השרשור) - טבלת גיבוב. גודל טבלת הגיבוב בכל "סל" הוא ריבועי ביחס למס' האיברים באותו סל. זמן הגישה לאיבר יהיה $O(1)$ וכן זמן הבנייה הוא $O(n)$ כי מילוי הטבלה הראשית לוקח $O(n)$ וכל טבלה של סל לוקחת זמן ליניארי במס' האיברים בסל בתוחלת. המקום הכולל שצפוי להתפס יהיה גם כן ליניארי!

חלק XIV
גיבוב קוקייה

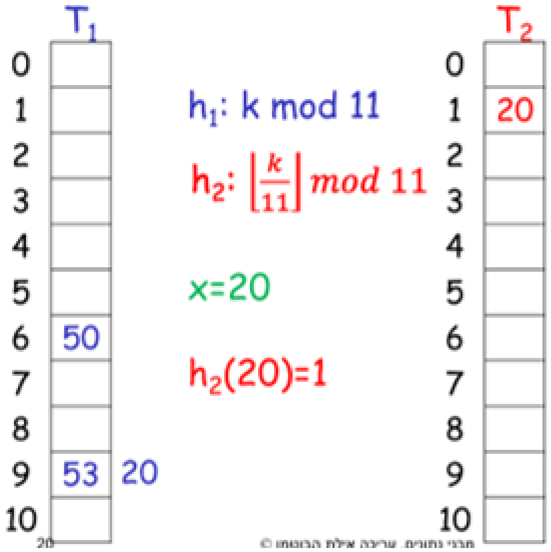
כעת נשתמש בשתי פונקציות האש ולא באחת. שתיהן יהיו בגודל זהה, ופונקציית ההאש תספק אינדקס לכל אחת מהן. כלומר, בהינתן שתי פונקציות האש T_1 וT_2מפתח x ישמר ב$T_1(h_1(x))$ או ב$T_2(h_2(x))$.
מימוש:
1. $Find$:

return $T_1(h_1(x)) == x$ or $T_2(h_2(x)) == x$

2. $Insert$: נראה כי ההכנסה לגיבוב קוקייה תהיה מעניינת. אנחנו נתחיל להכניס אל T_1 עד שנתחיל להתקל בהתנגשות. למשל, בדוגמה הבאה:



נראה כי ננסה להכניס 35 ונרצה ללכת אל אינדקס 9. אך הוא תפוס! מה נעשה? נכניס את 35 אל תא 9 בT_1 ונוציא את 02! עבור 02 נחפש מקום בטבלה השנייה:



כך בעצם נמשיך להתחלף בין שתי הטבלאות. כלומר, נניח וינסו כעת להכניס ערך אל טבלה 1, אין מקום לכן נחליף ונוציא את הערך. כעת ננסה להכניס את הערך אל הטבלה השנייה, גם שם אין מקום! נוציא משם את הערך וכעת אותו ננסה להכניס לטבלה הראשונה. וכך לסירוגין.
נעיר כי נרצה: $h_1, h_2 : U \to [m]$ כאשר $m = 4n$ (n= גודל הטבלה). נתבונן בפסודו -

```
function insert(x) is
  if lookup(x) then
    return
  end if
  while Max-Loop ≤ then
    if T1[h1(x)] = ⊥ then
      T1[h1(x)] := x
      return
    end if
    x ↔ T1[h1(x)]
    if T2[h2(x)] = ⊥ then
      T2[h2(x)] := x
    return
    end if
    x ↔ T2[h2(x)]
  repeat
  rehash()
  insert(x)
end function
```

נראה כי ייתכן ונגיע ללופ בו לא נצליח להכניס ערך לעץ, לשם כך בקוד יש הגבלה $max - loop$ כלשהי על מס' האיטרציות שניתן לעשות. אם נעבור אותו אז נחליף טבלאות גיבוב, נעתיק אליהם מחדש את האיברים וכן ניצור שתי פונקציות חדשות.

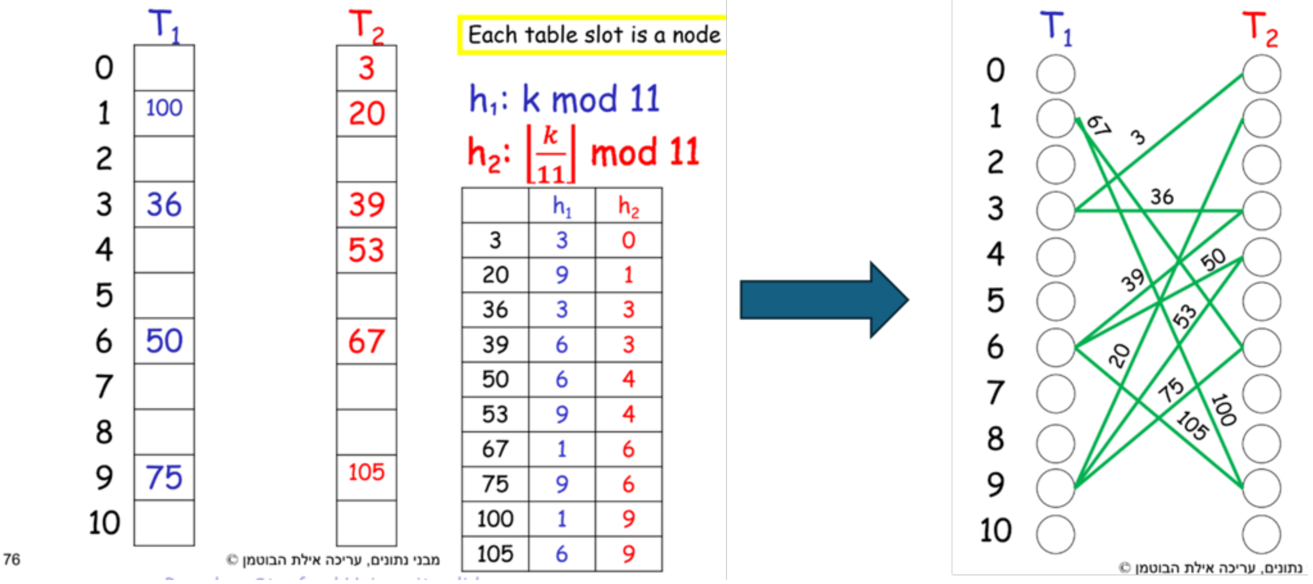
גרף קוקייה

גרף קוקייה הוא ייצוג מתמטי של טבלת גיבוב קוקייה באמצעות גרף. כל טבלת גיבוב קוקייה ניתנת להמרה לגרף כזה, וניתוח התכונות של הגרף מאפשר לנו להבין את ההתנהגות של טבלת הגיבוב. כל איבר שבטבלאות הופך ל-$node$.

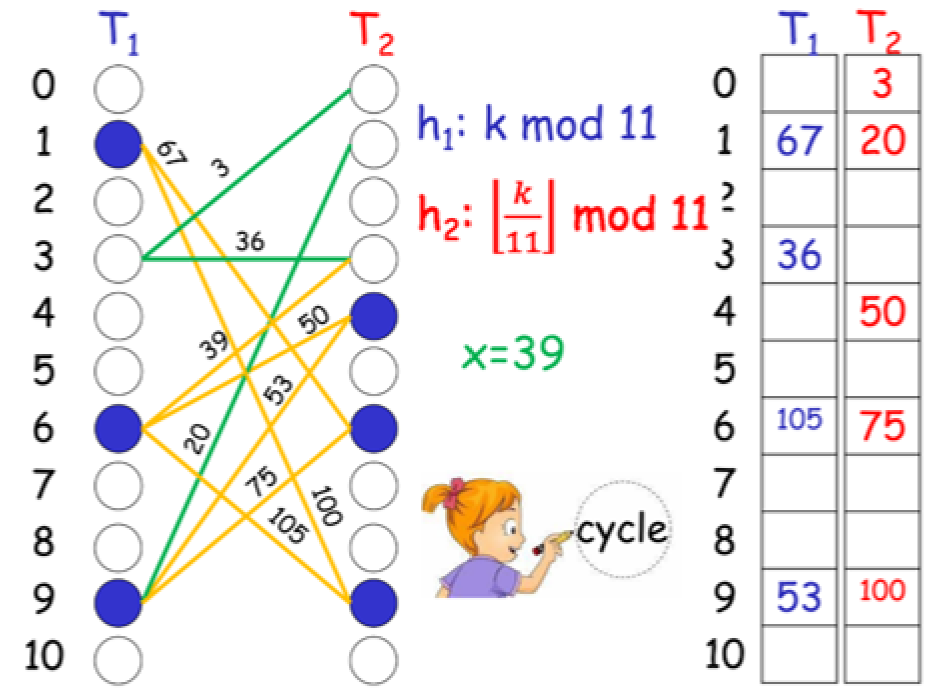
איך יוצרים גרף קוקייה מטבלת גיבוב קוקייה?

צמתים: צד אחד (L) מייצג את כל המיקומים האפשריים בטבלה הראשונה T_1 הצד השני (R) מייצג את כל המיקומים האפשריים בטבלה השנייה T_2

קשתות: עבור כל ערך x שאנחנו רוצים להכניס לטבלה, יוצרים קשת בין המיקום $h_1(x)$ בטבלה T_1 למיקום $h_2(x)$ בטבלה T_2 כלומר, כל ערך x יוצר קשת מהמיקום שלו בטבלה הראשונה למיקום שלו בטבלה השנייה.



נתבונן בדוגמה. כל קשת בגרף מקשרת בין מיקום בטבלה הראשונה משמאל לשנייה מימין. כל מפתח יוצר קשת אחת בגרף. הקשת מחברת בין המיקום שפונקציית הגיבוב הראשונה מחשבת לבין המיקום של השנייה.



תהליך ההכנסה מתואר בגרף כמסלול. כשדוחקים מפתח ממיקום אחד לאחר זה כמו לעבור על הקשתות במסלול. **אם נוצר מסלול פשוט בגרף** - ההכנסה תסתיים בהצלחה כשנגיע למיקום פנוי! **אם נוצר מסלול עם חזרות (מעגל)** אנחנו עלולים להכנס למצב של לולאה אנסופית.

רכיב קשיר - חלק בגרף בו מכל צומת ניתן להגיע לצומת אחרת ע"י הליכה על מסלול פשוט. יכולים להיות כמה רכיבי קשירות.

מחזור - מסלול בגרף שמתחיל בצומת ומסתיים בצומת בלי לעבור על אותה קשת פעמיים.

טענה: תנאי הכרחי ומספיק להצלחת הכנסה של מפתח לטבלת גיבוב קוקייה הוא שהרכיב הקשיר של הגרף שמכיל את המפתח, יכיל לכל היותר רכיב קשירות אחד.

סיבוכיות סוגי ההכנסות

k - אורך המסלול/ מעגל.

1. במסלול פשוט או מעגל יחיד, אם $k < clogn$ אזי זמן ההכנסה הוא $O(k)$
2. בכל מקרה, בשלב הראשון האלגוריתם ישקיע זמן של $cnlogn$ למציאת מקום פנוי , וישקיע זמן לבנייה מחדש של המבנה (במידת הצורך).

נגדיר $T(n)$ זמן ההכנסה של איבר חדש למבנה עם $n-1$ איברים. נשים לב כי $T(n)$ משתנה מקרי. אם נחשב, אחרי חישובים ארוכים מאוד נגלה כי לכל $c > 9$ מתקים

$$E[T(n)] \leq O(1) * O(\frac{n}{n-3}) = O(1)$$

כלומר סה"כ סיבוכיות ההכנסה במקרה הממוצע גם כאן - תהיה $O(1)$. גם אם יווצרו מעגלים ונתקן - בוורסט קייס עדיין בתוחלת יהיה לנו $O(1)$.

חלק XV

מבני נתונים לינאריים (מחסנית, תור, רשימה מקושרת ומערך) - בעיות מאתגרות

מערך: תאים רצופים בזכרון. גישה למיקום נתון בזמן קבוע. חיפוש במערך כללי $O(n)$ ובמערך ממוין $O(logn)$. גודל סטטי - לא ניתן להרחיב או להקטין.

תרגיל - חיפוש בינארי

נתון מערך בינארי ממוין $A[1..n]$ ומספר x המופיע במערך. נסמן ב-k את האינדקס (המינימלי) כך ש$A[k] = x$. הצע אלגוריתם המוצא את האינדקס k בזמן:

1. $O(logn)$
2. $O(mink, logn)$
3. $O(logk)$

פתרון:

1. טריוואלי. נתחיל באמצע של המערך וכל פעם נלך לחצי אחר. סה"כ ממש כמו בעץ - חיפוש בינארי של $O(logn)$.
2. גם כן כמעט טריוואלי - נרוץ עד לאינדקס הראשון זה $O(k)$, איך עושים ב$O(logn)$ זה כמו ב-1. אנחנו **נרוץ במקביל עם שני פוינטרים ונעצור ברגע שנמצא** k. כיוון שנרוץ פעמיים זה יעלה לנו:

$$2min\{k, logn\} = O(min\{k, logn\})$$

3. נניח שהמערך הממויין הוא בגודל די גדול (אנסופי), נחפש את התא הראשון שבו מיקום התא הוא חזקה של 2 וכן הערך בו הוא לפחות x. יקח לנו $logk$ פעולות להגיע לשם. טענה - תוך $[logk]$ צעדים נשאר את מערך שגודלו לכל היותר $2(k-1)$. החיפוש ייעצר בחזקה כלשהי i של 2:

$$2^{i-1} < k \leq 2^i$$

$$2^i = 2 * 2^{i-1} < 2k$$

כמה קפיצות עשינו? i קפיצות, וכן

$$i = log2^i < log2k = O(logk)$$

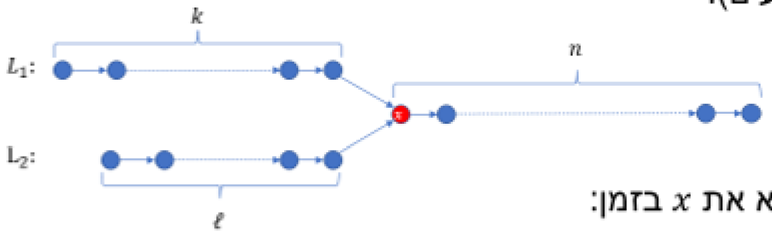
סה"כ חסמנו את גודל המערך שנרצה לחפש בו את האיבר הראשון לk איברים, זה עלה לנו $O(logk)$, כמו כן כעת נחפש בתוך התחום החדש הזה זה יעלה לנו גם $O(logk)$ ולכן סה"כ

$$2logk = O(logk)$$

רשימה מקושרת: כל איבר כולל תוכן ומצביע לאיבר הבא. ניתן לסרוק את הרשימה בזמן לינארי. ניתן להוסיף או להוריד איברים בקלות. גישה יקרה לאיברים באמצע הרשימה.

תרגיל - רשימות מתמזגות

- נתונות רשימות מקושרות באורך שונה (k,ℓ וn אינם ידועים):



- מצא את x בזמן:
  1) $O(k+\ell+n)$.
  2) $O(d^2)$ כאשר $d = \max\{k,\ell\}$.
  3) $O(d)$.

יש נקודה x בו הן מתאחדות. נרצה למצוא את הנקודה הזו.

פתרון:

1. בתחילה נמצא את האורך של כל רשימה. רשימה L_1 היא באורך $k+n$, רשימה L_2 היא באורך $\ell+n$. בה"כ נניח $k \geq \ell$, ההפרש באורך של הרשימות הוא $k-\ell$. נתחיל מללכת על $k-\ell$ האיברים הראשונים בL_1, ואז נגיע למצב ששתי הרשימות הן זהות באורכן. כעת נלך שוב על ℓ האיברים הבאים בשתי הרשימות ונגיע לאחר ℓ צעדים אל האיבר x.

2. בה"כ שוב $k \geq \ell$, לכן $d = k$. נרצה לעשות זאת ב$O(k^2)$. נבצע איטרציות במקביל, כך שבכל איטרציה i נשווה את הקודקוד הi שבL_1, עם כל אחד מהקודקודים הראשונים בL_2, עד שנמצא שוויון איבר ומצאנו את x, סה"כ

$$1 + 2 + ... + d = O(d^2)$$

3. בה"כ שוב $k \geq \ell$, לכן $d = k$. שוב נבצע איטרציות כך שבאיטרציה הi נשווה את הקודקוד הi של L_1עם כל אחד מi הקודקודים הראשונים של L_2, עד שנמצא שוויון. נעשה זאת אמ"מ i הוא חזקה של 2 , כלומר נקפוץ מאינדקס 1, לאינדקס 2, לאינדקס 4... טענה: בדרך הזו נמצא קודקוד y שנמצא מימין לx ומרחקו יהיה לכל היותר k ממנו.
סה"כ

$$1 + 2 + 4 + ... + 2^{logk} = O(k)$$

ואז כשנרצה למצוא את הקודקוד x, נלך עד k צעדים אחורה, וסה"כ יעלה לנו

$$k + k = 2k = O(k) = O(d)$$

דומה לפתרון שראינו קודם לכן, בבעיה הקודמת.

תרגיל - זיהוי מעגל ברשימה מקושרת

נתונה רשימה מקושרת חד־כיוונית בעלת n איברים. ייתכן שהקודקוד האחרון חובר לקודוקוד קיים ויצר מעגל. נרצה אלגוריתם המזהה אם יש מעגל, ואם יש מוצא את הקודקוד הראשון במעגל.
פתרון:
1. נוכל לסמן כל איבר שהיינו בו, ולכן כשנגיע לראשונה לאיבר שכבר סימנו קיבלנו מעגל. סה"כ זה יעלה לנו $O(n)$, אך השתמשנו גם ב$O(n)$ מקום כאשר שמרנו מידע בביטים או דגל אם היינו במקום.
2. פתרון של סיבוכיות $O(1)$ מקום וסיבוכיות זמן $O(n^2)$: נסמן, האיבר הi ברשימה יהיה האיבר שהגענו אליו לאחר i צעדים. הבחנה: אנו ברשימה מעגלית ולכן קיים איבר עם שני אינדקסים שונים. דהיינו קיימים $i_1 < i_2$ כך ש$A_{i_2} = A_{i_1}$, כלומר קיים איבר i שהוא גם האיבר ה$n+1$, בכל שלב נרוץ עד איבר j, עבור כל j נחפש האם קיים i מתאים כך שיווצר שוויון. סה"כ זה יעלה לנו

$$1 + 2 + ... + n = O(n^2)$$

3. באופן דומה, ננחש רק לערכי j כאשר הם חזקה של 2 ונעצר בj הראשון שיקיים $j = 2^{logn+1}$ ונקבל

$$1 + 2 + 4 + 8 + ... = O(n)$$

נשים לב שבפתרונות לא מצאנו את המעגל עצמו אלא האם קיים מעגל. איך נמצא את נקודת ההתחלה? כמו בשאלה עם הרשימות המתמזגות, כאשר רצינו למצוא את x.

תרגיל - אלגוריתם הצב והארנב - פלויד

המשך לתרגיל הקודם. נרצה למצוא את נקודת ההתחלה בדרך אחרת. אם הרשימות מאוד ארוכות כך שאי אפשר לשמור בזכרון את המס' n. נדמיין צב וארנב שמתחילים ללכת על הרשימה, בכל צעד, הצב יתקדם קודקוד אחד ואילו הארנב יתקדם 2 צעדים. אם אין מעגל - הארנב יגיע לקצה הרשימה. אחרת, בשלב כלשהו שניהם יהיו בתוך המעגל, ואז הארנב חייב לעקוף את הצב. לבסוף הם ייפגשו. מה זמן הריצה?

עד הרגע שהצב בתוך המעגל, $O(n)$ צעדים.

מהרגע שהצב במעגל, עד שהארנב יעקוף אותו: בכל סיבוב עד המפגש מס' הצעדים שהארנב צריך לבצע על מנת להגיע לצב קטן ב-1 בדיוק, לכן לאחר $O(n)$ צעדים נוספים השניים יפגשו.

לכן סה"כ $O(n)$ עד הפגישה.

איך מוצאים ככה את נקודת המפגש?

אלגוריתם: אתחל שני צבים – אחד מתחילת הרשימה, ואחד מנקודת המפגש. בכל סיבוב שני הצבים יתקדמו בצעד אחד. (הצב הראשון מתחילת הרשימה והשני ינוע בתוך המעגל),ברגע שהצבים נפגשים – דווח על הקודקוד כראשון במעגל. כלומר, נק' המפגש הוא המקום בו המעגל מתחיל. פלויד הוכיח שזה אכן עובד **תמיד.**

הוכחה: נסמן בX את מספר הצעדים שהצב ביצע בשלב הראשון עד נקודת המפגש עם הארנב. בזמן זה, הארנב ביצע בדיוק$2X$ צעדים. לכן, קודקוד המפגש, הוא קודקוד עם מספר X וגם עם מספר $2X$. אחרי X צעדים בשלב 2: צב א' יהיה במרחק X צעדים מההתחלה. צב ב' יהיה במרחק X צעדים מנקודת המפגש, שהם $2X$ צעדים מההתחלה. מכיוון שקודקוד המפגש משלב 1 הוא במרחק X מההתחלה וגם במרחק $2X$, שני הצבים יהיו בשלב זה על אותו קודקוד.

תרגיל - בעיית $2SUM$

נתונה קבוצה של מספרים $S = x_1, x_2, \dots x_n$. הצע מבנה נתונים התומך בפעולות הבאות: בנייה. שאילתת-סכום(y) – מחזירה האם קיים זוג איברים $x_i, x_j \in S$ כך ש$x_i + x_j = y$ואת האינדקסים.

פתרון: בנייה – נשמור במערך ממוין $A[1..n]$.

שאילתא בזמן $O(n)$. נחפש זוג אינדקסים מתאים. נסמן $\ell < h$. נסתכל על הסכום של $A[1]+A[n]$. אם שווה ל$y = A[1] + A[n]$ סיימנו ו$(\ell, h) = (1, n)$. אם $A[1] + A[n] < y$ אז$\ell >1$. אם $A[1] + A[n] > y$ אז $h < n$. לכן ניתן באופן רקורסיבי לפתור את הבעיה למערך שקטן בתא אחד, כך נתקדם אל המערך ונוריד מהקצוות במידת הצורך ובסוף נמצא לאחר n פעולות את שני המספרים.

חלק XVI

בעיית $Union - Find$

איחוד קבוצות זרות. נתון עולם של איברים U כך ש $|U| = n$ ונרצה לתמוך בפעולות הבאות:

1. $MakeSet(i)$- יוצר קבוצה חדשה בעלת איבר בודד i ומחזיר אותה. $O(1)$
2. $Find(i)$ - מחזיר את הקבוצה לה שייך האיבר i. $O(log^*n)$
3. $Union(p,q)$- מקבל שני פוינטרים לשתי קבוצות, יוצר קבוצה חדשה שמכילה את האיברים משני הקבוצות ומחזיר אותה. יש לשים לב כי הקבוצות ישארו זרות זו לזו שכן אנחנו מאחדים את הקבוצות שנקבל ומוחקים אותן מיד. לא יתכנו שני קבוצות עם אותו איבר. $O(1)$

דוגמה לשימוש: נתונים n ערים $\{1, ...n\}$. בתחילה כל הערים מנותקות. מדי פעם סוללים כביש בין שני ערים. יש לאפשר את הפעולות הבאות - $addRoad(x,y)$ - הוסף כביש ישיר בין שנ הערים, $checkConnectivity(x,y)$- בדוק אם הערים מחוברות במסלול כלשהו. פתרון: לכל עיר יהיה $makeSet$. כאשר עושים $addRoad$ נעשה $find$ לשני הערים x,y ואז נעשה $union$ לשני המצביעים ויצרנו כביש. כאשר נרצה לעשות $connectivity$ נעשה $findx, findy$ ונבדוק האם הפוינטר זהה - ואם כן יש מסלול בניהם אחרת לא.

איך נבנה את מבנה הנתונים?

1. **מימוש נאיבי עם מערך:** נשתמש במערך A. בתא $A[i]$ נשמור את שם הקבוצה אליה שייך האיבר $A[i]$.

$Find$ יהיה פשוט לגשת ל$A[i]$ ויחזיר את השם ב$O(1)$

נשתמש במשתנה $counter$ בשביל לתת שמות לקבוצות

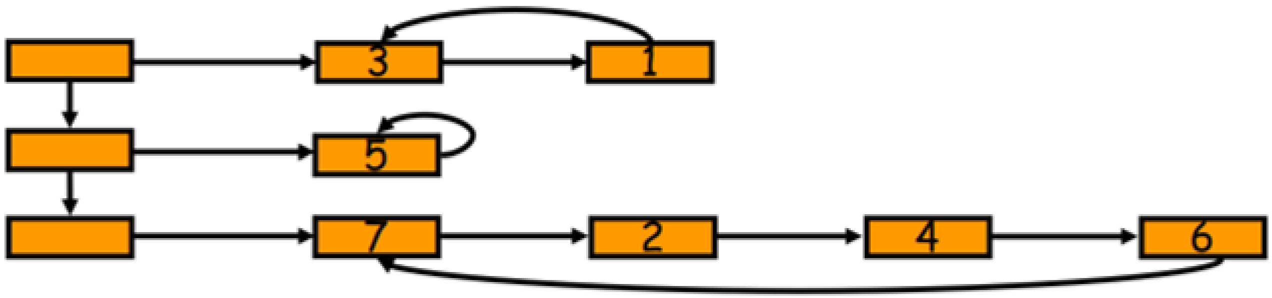
$makeSet$: $A[i] = counter++$ ב$O(1)$

$Union$: נעדכן את $counter++$, נעבור על כל איברי המערך ואם $A[i] == A[q]$ או $A[i] == A[p]$ אזי $A[i] = counter$. נראה שפעולה זו תעלה לנו $O(n)$.

לא טוב..... יותר מדי יקר.

הערה: הנחנו ש$n = |U|$, לכן יכולנו ליצור מערך סטטי בהתחלה. אם זה לא המצב כמובן שזה לא יעבוד.

2. **מימוש נאיבי בעזרת רשימות מקושרות מעגליות:** נייצג כל קבוצה כרשימה מעגלית. נחזיק רשימה מקושרת של מצביעים אל הרשימות המעגליות. דוגמה:



$Find(i)$ - ממומשת ע"י מעבר על כל הרשימות עד שמוצאים זה יעלה $O(n)$!!!

$MakeSet(i)$- הוספת רשימה עם איבר בודד ב$O(1)$

$Union(p, q)$- אנחנו מחזיקים בפוינטר לסוף הרשימה ולכן נשנה את הפוינטרים שבמקום שאחד יצביע להתחלה של הראשון, יצביע לסוף של השני. זה כמובן $O(1)$.

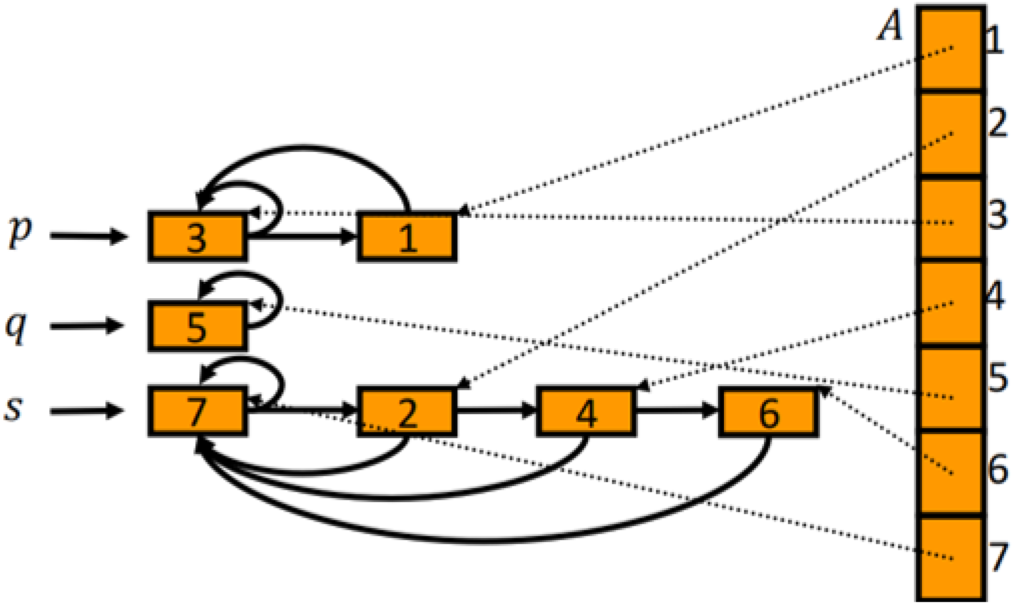
גם זה לא טוב לנו. יותר מדי יקר למצוא.

3. **פתרון יותר טוב:**

נשלב את שני הרעיונות הקודמים. נרצה $MakeSet$ ו$Find$ ב$O(1)$ ו$Union$ ב$O(logn)$ בתוחלת.

תזכורת. אם m פעולות מתבצעות בתוך זמן M , אזי הזמן המשוערך לכל פעולה הוא $\frac{M}{m}$.

נממש עם רשימות ומערך מיפוי. נייצג כל קבוצה כרשימה. בכל איבר נשמור פוינטר לראש הרשימה וכן כמובן פוינטר לאיבר הבא ברשימה. בנוסף לזה נחזיק מערך מיפוי איברים $A[i]$ ובו מצביע לאיבר הi. קבוצה מיוצגת ע"י מצביע לראש הרשימה המתאימה.



$MakeSet$: ניצור רשימה חדשה ונצביע עליה מ-$A[i]$. נחזיר מצביע לסוף ותחילת הרשימה.כמובן $O(1)$

$Find$: מתוך $A[i]$ נגיע אל הרשימה בה הוא מוחזק, ונלך מהאיבר אל הפוינטר (הרי הוא מחזיק פוינטר לתחילת הרשימה), ונחזיר אותו. כמובן $O(1)$.

$Union$: נאחד את הרשימות המוצבעות ע"י p, q לרשימה אחת ונעדכן את כל המצביעים לראש הרשימה החדשה. נחזיר את ראש הרשימה החדשה.

בצער רב זה $O(n)$ **- האמנם??** כאשר מאחדים שתי קבוצות יש לשנות באחת הקבוצות את כל המצביעים לראש החדש. לכן כמובן עדיף לעשות זאת בקבוצה הקטנה מבין השניים. נשפר את המימוש - נוסיף בראש הרשימה $counter$ שייצג את גודל הקבוצה. כעת שניגש לאחד נלך כמובן לזה שגודל הקבוצה שלו קטן יותר.

נשים לב שבתחילה הקבוצות מכילות איבר אחד. בפעם הראשונה שנעדכן מצביע זה קבוצה מגודל אחד, בפעם השנייה מגודל שניים, בפעם השלישית קבוצה שכבר לפחות בגודל 4.... כמה נאחד? אינטואיטיבית $logn$. ופורמלית -

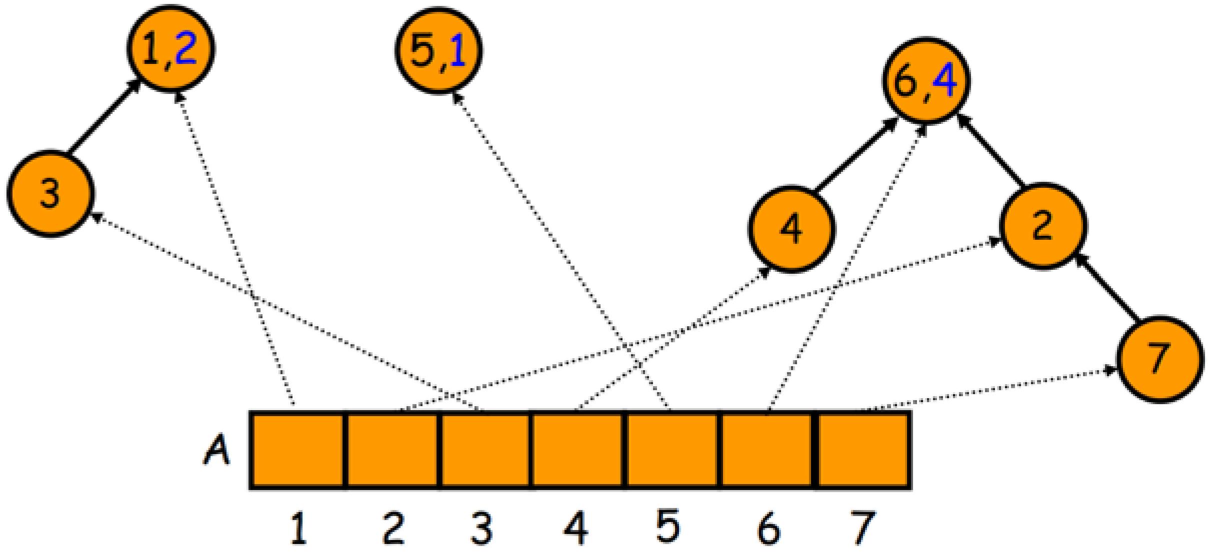
טענה: אם בכל איחוד מוסיפים את הקטנה לגדולה, אז בכל סדרה של m פעולות שמכילה לפחות n פעולות $makeSet$ כל איבר x משנה את ראש הקבוצה אליה הוא שייך לכל היותר $logn$ פעמים. כלומר לאחר $logn$ מעברים הקבוצה אליה x שייך תהיה בגודל n.

לכן לפי ניתוח לשיעורין - ממוצע מס' הפעולות ל-$Union$ יהיה $O(logn)$.

4. עוד יותר טוב: עצים הפוכים - $Up-Trees$

לכל קבוצה ניצור עץ הפוך (הבנים מצביעים להורים), ובו צומת לכל איבר בקבוצה. שורש העץ יכיל גם את מספר איברי הקבוצה. בנוסף נחזיק מערך גישה לאיברים. בביצוע פעולת $Union$ אנחנו נתלה את שורש העץ הקטן יותר מתחת לשורש העץ הגדול יותר ונעדכן את גודל הקבוצה המאוחדת.

נשים לב שאנחנו ב**קבוצה** ולא אכפת לנו מסדר האיברים ולכן השימוש ברשימה היה די מיותר. אנחנו נשמור במערך פוינטר לכל איבר: כמו כן נשמור בסגול בראש העץ את מספר האיברים.

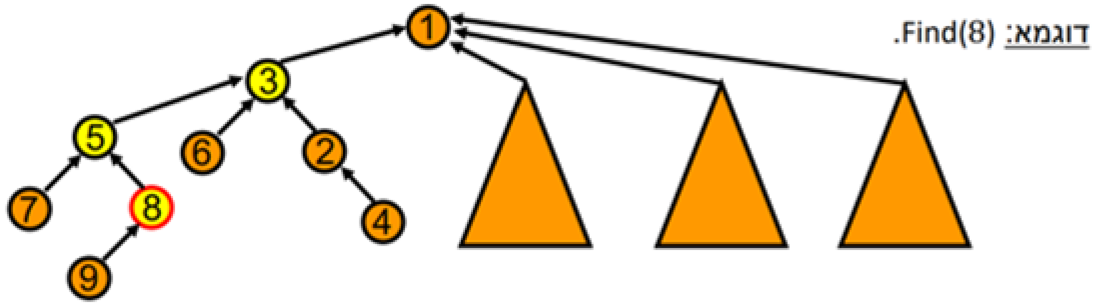


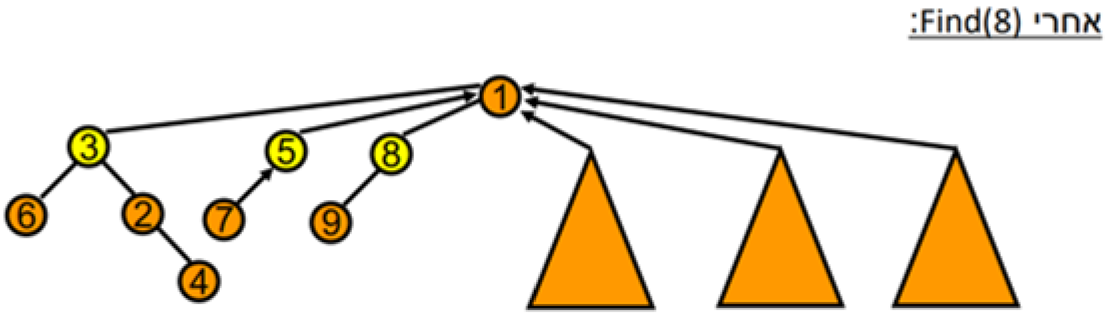
$Find$: נכנסים למערך, הוא מביא אותי לעץ הספציפי,ואז צריך לטפס אל השורש.
$Union$: נתלה את שורש העץ הקטן יותר בגובה תחת העץ הגדול. וכמובן נשנה את גודל הקבוצה.

ניתוח סיבוכיות:

זמן פעולת $Find$ במקרה הגרוע הוא $O(logn)$. פעולת $Union$ היא $O(1)$. נרצה לשפר - כיווץ מסלולים:

בזמן ביצוע $Find(i)$, נעדכן את שדה ההורה של כל הצמתים במסלול מi ועד השורש, כך שיצביעו ישירות לשורש. כך -





אנחנו למעשה בכל פעם נכווץ את המסלול. נדחוס את המסלול בכל פעם!
נגדיר: $log*n$ - מס' הפעמים שיש להוציא log מn כדי להגיע למס' קטן שווה מ-1. למשל -

$$log^*(2^{2^{2^{2^{2}}}}) = 5$$

וכן

$$log^*(n) = O(1)$$

כעת, סיבוכיות פעולת $Find$ תהיה $O(log^*(n))$. **(לא נוכיח ולא נדון מדוע)**

חלק XVII
תכנון דינאמי

סדר הפעולה בתכנון דינמי

א. אלגוריתם נאיבי.
ב. אלגוריתם רקורסיבי שכולל:
1. נוסחה רקורסיבית.
2. תנאי עצירה.
3. הסבר רעיוני של הנוסחה ומשמעות כל אינדקס ומשתנה בנוסחא.
4. הוכחת נכונות של הנוסחה הרקורסיבית.

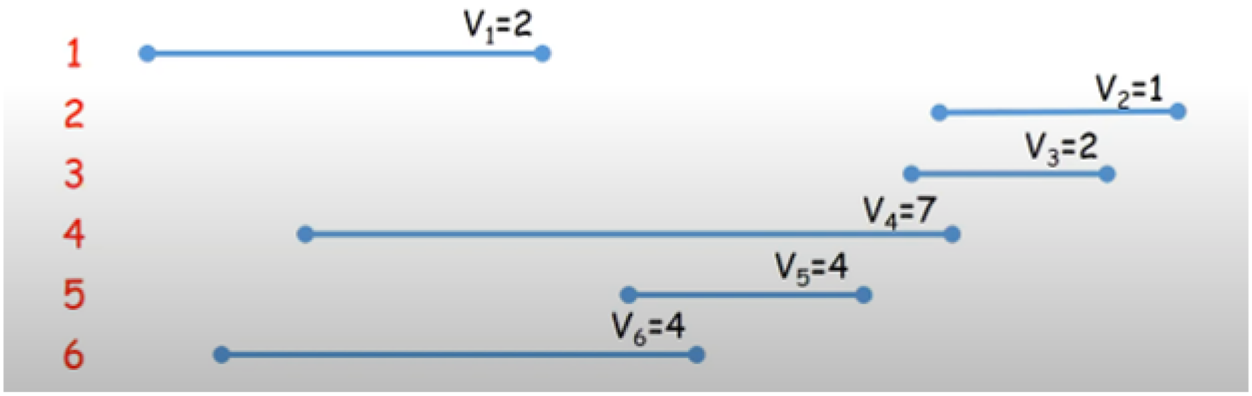
ג. אלגוריתם תכנות דינאמי שכולל -
1. תאור מבנה הנתונים (מערך, עץ, טבלה)
2. גודל המבנה
3. מה מכילה יחידה בודדת בתוך המבנה.
4. אופן מילוי המבנה
   - i. תנאי עצירה
   - ii. איך משתמשים בנוסחה הרקורסיבית
   - iii. סדר מילוי היחידות (למעלה למטה, שמאל לימין וכדומה)
5. היכן במבנה נמצא הפתרון בעיה
6. פסודו קוד לאלגוריתם
7. ניתוח סיבוכיות זמן ומקום - ולציין אם ניתן לצמצם שימוש בזכרון (איך ולכמה)
8. שחזור פתרון - אם רלוונטי.

מהו תכנון דינאמי? תיאור כללי ולא פורמלי -

לפיתוח אלגוריתם המבוסס על תכנון דינאמי, דרוש אוסף תתי בעיות הנגזרות מן הבעיה המקורית ומקיימות את התכונות הבסיסיות הבאות:
א. מס' תתי הבעיות הוא פולינומיאלי (אצלנו $n+1$)
ב. הפתרון לבעיה המקורית אפשר לחשב בקלות מהפתרונות לתת הבעיות.
ג. יש סדר של תתי הבעיות מן "הקטנה ביותר" לגדולה ביותר ויש נוסחת נסיגה שקל לחשב.
* נעיר כי נפתור בעיות שניתן לפתור בצורה רקורסיבית - במקרים בהם שיש קריאות רבות עם קלט זהה. במקום לחשב מחדש כל פעם את הקריאות נחשב את הפונקציה על כל קלט פעם אחת בלבד. ישנן שתי שיטות מרכזיות לחישוב הפונקציה - אחת היא מלמעלה למטה והשנייה מלמטה למעלה.

בעיה ראשונה - בעיית תזמון מקטעי ממושקלים

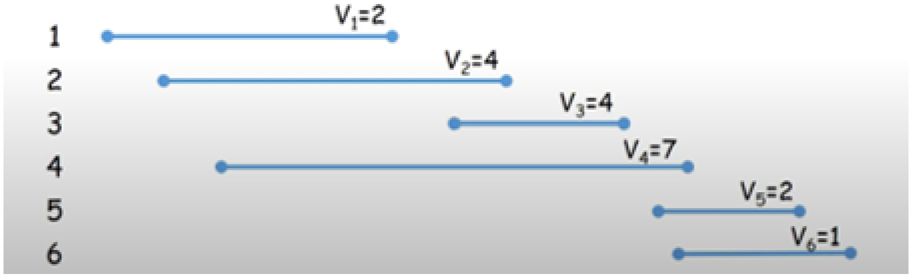
הבעיה: נתונות אוסף של n בקשות הממוספרות $1,...n$. לכל בעיה i יש מועד התחלה s_iויש מועד סיום f_i. לכל בקשה i יש משקל התחלה v_i, כמופיע כאן -



למשל: עבור העבודה V_2 נניח שהתחילה לעבוד ב$16:00$ ותמשך עד $17:30$, סה"כ שעה וחצי זמן העבודה כפול משקל העבודה(באלפים) -

$$1.5 * 1000 = 1,500$$

בעל המפעל מבין שאינו יכול לקבל את כל העבודות - מדוע? כי יש חפיפה בזמנים וניתן להפעיל את פס הייצור עבור עבודה אחת כל פעם.
אנחנו נרצה לדעת כיצד ניתן למקסם את הרווח שלנו במפעל - כלומר למצוא את המצב האופטימלי עבורנו. אנחנו נרצה לקבל מקטעים זרים זה לזה ולדעת כיצד לבחור אותם באופן המקסימלי הרווחי.
למשל - בדוגמה מעלה נוכל לבחור את עבודה 2 ועבודה 5.
מטרה: מוצאים תת קבוצה $S \subseteq \{1,...n\}$ של מקטעים שכולם זרים זה לזה וסכום ערכיהם $\sum_{i \in S} v_i$ הוא מקסימלי. (לאו דווקא אומר שהרבה עבודות=הרבה רווחים)
פתרון נאיבי - יש לנו 2^n תתי קבוצות, נגדיר את המקסימום להיות אפס ונתחיל לסרוק סכום עבורS_{max} כל אחת מהקבוצות, נסרוק את כולן ונשווה כל פעם לS_{max} ונקבל סה"כ $\Theta(2^n)$ - אקספוננציאלי. השם ישמור - נרצה הרבה יותר טוב.
פתרון: נרצה להגדיר את הבעיה בצורה רקורסיבית. נמיין את האיברים לפי מקטעי הסיום שלהם - כמו כאן (נשים לב שאלו אותם העבודות כמו למעלה, רק במיון לפי סדר שעת הסיום שלהם)



כמו כן נשנה את מספורם מחדש - כלומר נמספרם מחדש את V_i לפי $f_1 \leq f_2 \leq .. \leq f_n$, נאמר שבקשה i קודמת לבקשה j אם $i < j$.
נגדיר $p(j)$ עבור מקטע j - אינדקס i הגדול ביותר כך ש$j > i$ וכן המקטעים i וj זרים. כמו כן $p(j) = 0$ אם אין אינדקס i לפניו.
למשל אצלנו בדוגמה - $P(v_2) = 0$ מדוע? היחיד שמסתיים לפני v_2 הוא v_1 אך הם אינם זרים.
כמו כן - $p(v_3) = v_1$ - כי הוא מסתיים לפני והם זרים.
כל ה"הכנות" האלו יעלו לנו $nlogn$ כי מיון לוקח $nlogn$ וכן אח"כ עברנו על כולם באופן לינארי $O(n)$ שחישבנו את $p(j)$ **כלומר עד כאן** - $O(nlogn + n) = O(nlogn)$.
כעת המטרה שלנו היא להעזר בכך למצוא את הפתרון האופטימלי - כלומר לא נמצא את הקבוצה עצמה אלא את הסכום.
תובנה טריוואלית - המקטע האחרון n יכול להיות שייך ל-A או יכול להיות שלא שייך.

תובנה טריוואלית נוספת - אם $n \in A$ אזי אף אחד מהמקטע עם אינקס שבין $p(n)$ לn (ושונה מהם) לא יכול להשתייך ל־A. כלומר - הערך $p(n)$ אומר לנו מי המקטע הכי גדול שיכול להיות ביחד עם n בקבוצה.
תובנה נוספת - אם $n \in A$ אזי A כולל בהכרח פתרון אופטימלי של הבעיה המבוססת על הבקשות $\{1, ...p(n)\}$. (מההסבר מעלה - אם הוא כבר בפנים, בשביל שלא תהיה חפיפה בוודאות אפשרי עד אינדקס $p(n)\}$.
מכאן שניתן להגיע לנוסחה הרקורסיבית - או שהאיבר ה־n נמצא או שלא. ננסח זאת כך -

$$opt = \left\{ \begin{array}{cc} n-in & v + opt(1, ...p(n)) \\ n-No & opt(1, ...n-1) \end{array} \right\}$$

מכאן שהנוסחה תתעדף את המקסימום מבין האפשרויות, כלומר

$$opt(j) = max\{(v + opt(1, ..p(n)), opt(1, ..n-1)\}$$

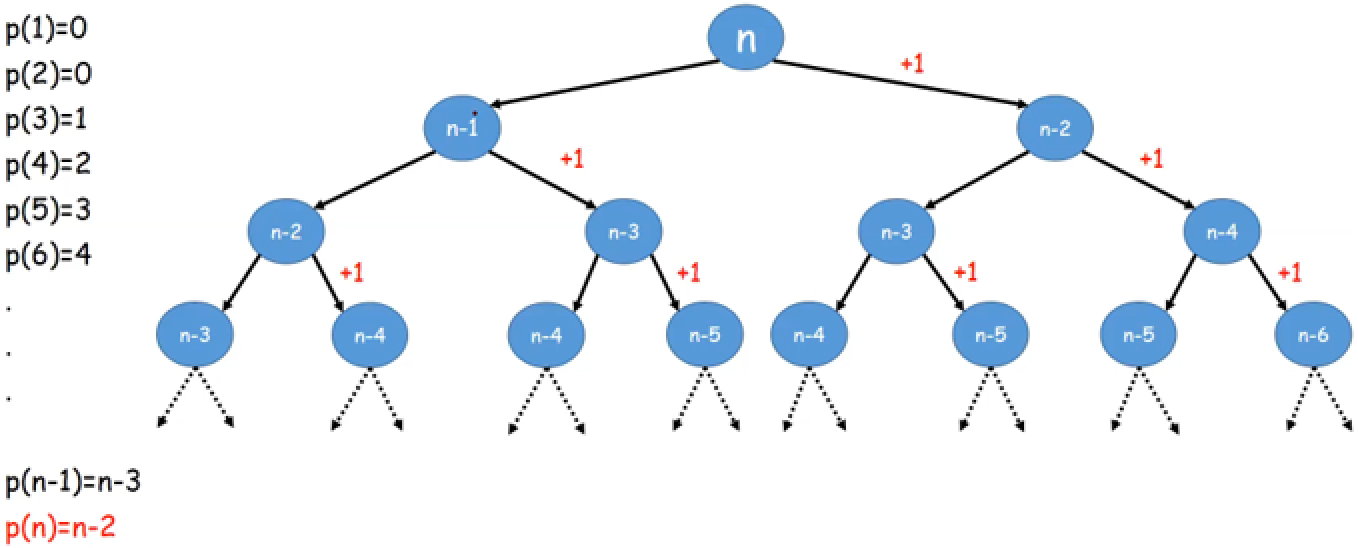
מכאן שהפתרון האופטימלי A_jעבור $\{1, ...j\}$ הוא
א. $j \in A_j \Longleftarrow OPT(j) = v_j + OPT(p(j))$
ב. $j \notin A_j \Longleftarrow OPT(j) = OPT(j-1)$
ובאופן כללי - $OPT(j) = max\{v_j + OPT(p(j)), OPT(j-1)\}$ יפתור את הבעיה. (כמובן שכמו כל רקורסיה, נוסיף תנאי אם $j = 0$ נחזיר 0).
זהו המרכיב המכריע הראשון עליו מבוסס תכנון דינאמי - נוסחה שמבטאת את הפתרון האופטימלי (או ערכו) במונחי הפתרונות האופטימליים לתת בעיות קטנות יותר, כלומר - המכריע הראשון הוא חישוב נוסחת נסיגה למציאת פתרון אופטימלי.
כעת - נחשב את זמן הפתרון. נכתוב את נוסחת זמן הריצה שלה.

$$T(n) := \left\{ \begin{array}{cc} 1 & n = 0 \\ 1 + T(p(n)) + T(n-1) & else \end{array} \right\}$$

מזל שלמדנו הפרד ומשול - ננסה לפתור את הנוסחה הנ"ל. בהינתן קלט בגודל n מה אפשר להגיד על זמן הריצה? נתבונן על דוגמה (בה כל קטע חופף לזה שהיה שניים לפניו וכל הערכים שווים 1) -

עץ הרקורסיה

```
Compute-Opt(n)
    if j = 0 then return 0
    return max(v_n + Compute-Opt(n-2) , Compute-Opt(n-1))
```



עץ זה מזכיר לנו את העץ לחישוב פיבונאצ'י, זה ערך קצת יותר כללי - הוא אמנם ספציפי אך כבר עובדים עם n כללי. נראה כי העץ מתואר כך:

$$opt(n) = max\{v_n + opt(n-2), opt(n-1)\}$$

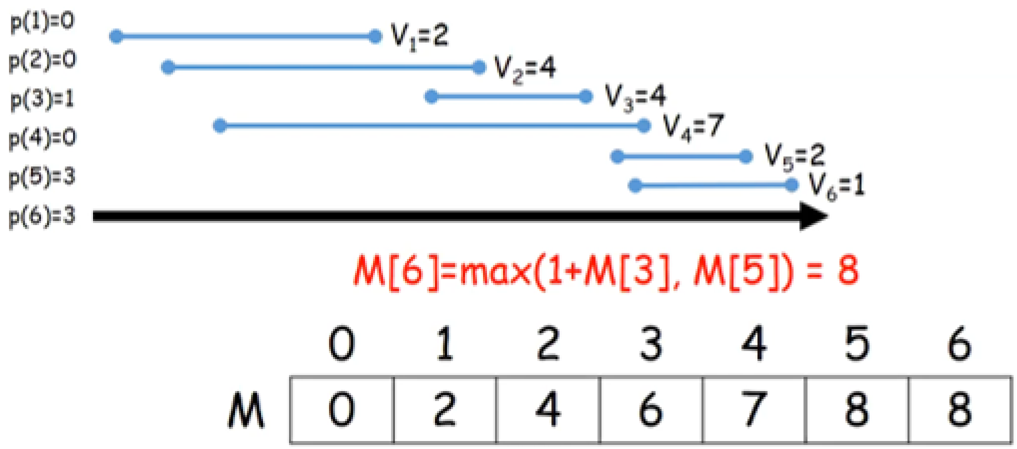
קיבלנו עץ רקורסיה שמתאר את חישוב האיבר ה־n־י בסדרת פיבונאצ'י. הוא עץ מלא. מה גובה העץ? נתבונן על אורך המסלול הקצר ביותר - המסלול הימני ביותר שהולך תמיד ימינה גודלו יהיה $\frac{n}{2}$. זה סדר גודל של מינימום - שכן יש מסלול גדול יותר אבל בהינתן $h = \frac{n}{2}$ יש לנו $2^{\frac{n}{2}+1}$ עלים $\Leftarrow$ זה לא טוב! אם עוד לפני המצב הגרוע קיבלנו אקספוננציאלי מה נקבל במצב הגרוע? אבל לא עבדנו לחינם. כשנסתכל על העץ נראה שיש הרבה קריאות שחוזרות על עצמן! כמה קריאות רקורסיביות יש = כמה קודקודים עם ערכים שונים בצמתי העץ קיימים. וכמה כאלו קיימים? יש $n+1$ קריאות שונות - $f(0), ...f(n)$ $\Leftarrow$ איך נחסוך כאן? נשמור במערך בגודל $n+1$ את הערכים הקודמים

f(0)	f(1)	f(2)	f(3)	f(4)	f(5)
1	1	2	3	5	8

נשים לב שבאופן כללי בבעיה שלנו - כל תא במערך מסתמך על שני תאים קודמים ולכן לא רק עבור המקרה הכללי עם פיבונאצ'י ולכן קיבלנו רעיון אחר לפתרון הבעיה -
העקרון המכריע השני של הפתרון בשיטת התכנון הדינאמי: האלגוריתם הרקורסיבי פותר למעשה רק $n+1$ תת בעיות. כיצד נחסוך במס' הקריאות?
נשמור את הנתונים בתוך מערך כמו בפסודו הנ"ל -

M[0]=0
for j=1,...n
m[j]=$max\{v_j+M[p(j)],m[j-1]\}$

במקום לחשב ברקורסיה – המערך ישמור את התוצאה. כלומר – אין רקורסיה כבר (יש!!). המערך פותר את הבעיה כי אכן מסתמכים על דברים שכבר חישבנו וירדנו מהזמן האקספוננציאלי. כך זה נראה –



מה העלות שלנו? כפי שראינו כאן מדובר בעלות לינארית – אם נזכור טוב בהתחלה מיינו את המערך של סדר הפעולות וזה עלה לנו $nlogn$ ולכן סה"כ ירדנו בפתרון הבעיה מפתרון אקספוננציאלי ל$O(nlogn)$!

הערה: אם כבר היינו מקבלים מערך ממויין הפתרון האופטימלי היה $O(n)$ וזהו הפתרון האופטימלי שכן אי אפשר בפחות – במינימום, אני חייב לקרוא את כל המידע.

נשים לב – מצאנו את מחיר הקבוצה המקסימלית. לא מצאנו את הקבוצה המקסימלית עצמה!

*הרחבה נאיבית של הפתרון – נוסיף מערך נוסף S כך ש$S[i]$ יכיל קבוצת מקטעים אופטימלית מערך $\{1,...n\}$ וגודל המערך הנ"ל הוא דו ממדי $O(n^2)$ – כלומר נשמור עבור כל אפשרות במערך M את הדרך אליה הגענו למספר. במקרה הגרוע זה יעלה לנו אכן $O(n^2)$

*ניתן לעשות זאת באופן יפה יותר – לאחר שמצאנו את המחיר וגילינו אותו, כעת נעשה רוורס ברקורסיה ונחזור אחורה וננסה להבין את הערכים שהרכיבו לנו את הסכום – נראה שזה יעלה לנו $O(n)$. נראה פסודו –

if $v_j+M[p(j)]\geq M[j-1]$then output j togther with the result of find-solution(p(j))
else
output the result of Find-Soultion

זה בדיוק ברקורסיה, היא תעלה $O(n)$ כי הפעם הולכים בדיוק על מסלול אחד.

זהו! הפכנו בעיה מזמן $O(2^n)$ לזמן של $O(n)$ כאשר ממויין או $O(nlogn)$ כשלא – זה מדהים. מדוע פירטנו כאן כל כך הרבה בדרך? זה בדיוק תכנון דינמי. זו הייתה דוגמה קלאסית שממחישה זאת.

הבעיה השניה – כפל מטריצות

נזכר שבהינתן שתי מטריצות A ו־B בשביל לכפול אותן נדרוש כי $A\in\mathbb{F}^{n\times m}$ וכי $B\in\mathbb{F}^{m\times k}$ ואז נקבל כי $AB\in\mathbb{F}^{n\times k}$

אלגוריתם נאיבי: כפי שד"ר עדי בן צבי לימדה אותנו – $(AB)_{ij}=\sum_{k=1}^{n}A_{ik}B_{kj}$ – כמה עולה לכפול שתי מטריצות בשיטה זו? $O(m\times n\times k)$ – טו מאץ'.

נזכר כי כפל מטריצות היא פעולה אסוציאטיבית – $A(BC)=(AB)C$.
בהינתן שלוש מטריצות נראה כי מהיר יותר לחשב את $A(BC)$

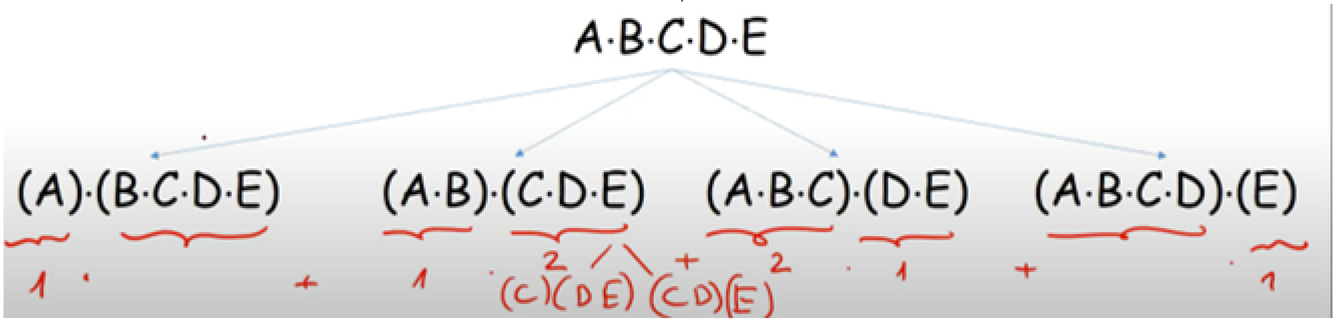
*נעיר כי המטרה שלנו היא בעיית אופטימום - איזה בעיית הצבת סוגריים תתן לנו את מינימום המכפלות. לא נרצה ממש לחשב את הכפל עצמו.

קלט: סדרה של n מטריצות: $A_1,....A_n$

ממדי המטריצה A_iהם $p_{i-1}\times P_i$לכל $1\le i\le n$.

פלט: הצבת סוגריים מלאה במכפלה $A_1*...*A_n$באופן שימזער את מס' המכפלות הסקלריות המבוצעות בעת חישוב כפל מטריצות.

אלגוריתם נאיבי - בדיקת כל הצבות הסוגריים האפשריות. כלומר האלגוריתם הנאיבי עובר על כל התוצאות האפשריות להצבת סוגריים בהינתן סדרת מטריצות.



כמה אפשרויות יש כאלו? נראה כי לפי הדוגמה מעלה, בהינתן $p(5)$ מימין לשמאל - יש אפשרות אחת ($p(1)$) ועוד $p(4)$ ותמיד יהיה המשלים ל-5. למשל בדוגמה השנייה מימין זה 2 ו-3. כלומר נוכל לכתוב שמס' האפשרויות להצבת סוגריים-

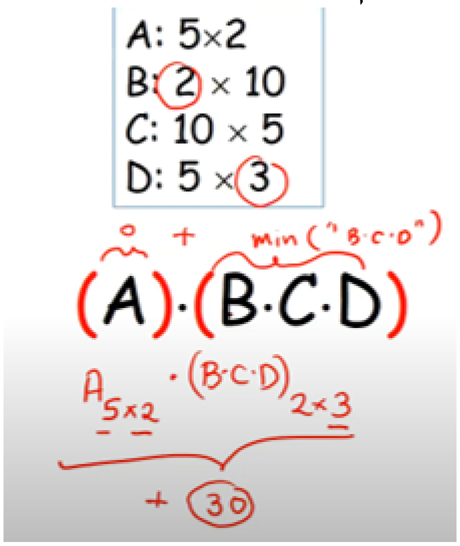
$$p(n)=\left\{\begin{array}{cc} 1 & n=1 \\ \sum_{k=1}^{n-1}p(k)p(n-k) & n\ge 2 \end{array}\right\}$$

נרצה להעבירה מנוסחה רקורסיבית לנוסחה סגורה! אנחנו רוצים להבין באופן נאיבי כמה אפשרויות יש לנו. הפתרון לנוסחת הנסיגה הוא- מס' קטלן C_n (מי היה מאמין שזה שימושי), כמו כן - אנשים חשובים עמלו והשקיעו וגילו כי $C_n=\Omega(\frac{4^n}{n^{1.5}})$ ולכן מס' ההצבות הסוגריים האפשריות הוא מס' אקספוננציאלי - האלגוריתם הנאיבי **לא סביר בעליל**.

כעת - לפי עקרון תכנון דינאמי - לאחר שנכשל הרעיון הנאיבי - ניגש לפתרון בהפרד ומשול

אפיון המבנה של הצבת סוגריים אופטימלית: מה מינימום המכפלות בהינתן n מטריצות.

נתבונן בדוגמה הבאה (לטובת אינטואציה)- נראה כי במינימום יתבצעו 3*2*5 מכפלות בהתאם לגודל המטריצה הגדולה, כעת כשנפרק את הבעיה לתת בעיה - נקבל 0 מכפלות במטריצה A ועוד האופטימום - מס' המכפלות המינימלי בתת הבעיה של BCD שבוודאות תהיה קטנה מ-30. מכאן אנחנו מקבלים מוטביציה לנוסחת נסיגה.



נראה כי המינימום להכפיל את ארבעת המטריצות הוא הכמות שעולה להכפיל את A ועוד הכמות שעולה להכפיל את BCD ועוד הכמות שעולה להכפיל את שני החיבורים. מדוע טענה זו נכונה? נב"ש כי יש חלק בו לא לקחנו את המינימום - אזי לקחנו אחד גדול ממנו בוודאות בסתירה לכך שאנחנו מחפשים פתרון אופטימלי לבעיה. מכאן שאם נסתכל על הדוגמה שלנו מלמעלה , בהינתן מטריצות A, B, C, D ישנן 3 אפשרויות מכפלה -

$$A(BCD),(AB)(CD),(ABC)(D)$$

ולכן הכמות המינימלית למכפלות תהיה המינימום מבין כל השלושה. לפי אותו תהליך שתואר מעלה.
המידע שמעניין אותנו הוא גדלי המטריצה בצורה רציפה. כלומר נרצה לשמור במערך P את גדלי המטריצה - בצורה הזו: $\{......\} =< p_0, ...p_n >$ כאשר נמספר את המטריצות מ-1 עד n וכן הערך הראשון p_0יתאר את מס' השורות במטריצה הראשונה.
דוגמה: בהינתן $< p_0, p_1, p_2, p_3, p_4 >=< 5, 2, 10, 5, 3 >$, **הסימון** $m(1,4)$ **ייצג לנו את המס' המינימלי של מכפלות עבור המטריצות 1 עד 4.**
*הערה - אם אנחנו במטריצות למשל 1-4 אז אנחנו עובדים עם אינדקסים $0, 1, 4$ בחישוב הסכום המינימלי למכפלת שני המטריצות ובאופן כללי בהינתן $m(k, g)$ נעבוד עם אינדקסים $k-1, k, g$

$$min(1,4) = m(1,1) + m(2,4) + p_0 * p_1 * p_4 = m(1,1) + m(2,4) + 5 * 2 * 3$$

זה ערך אפשרי אחד עבור מינימום מס' המכפלות. צריך לחשב סה"כ את כל האפשרויות וביניהם להחזיר את המינימום.
מכאן שבאופן כללי נוכל לתאר פורמלית את מה שאמרנו, באמצעות ההגדרה הרקורסיבית להלן:
נסמן $A_{i..j} = A_i * ...A_j$
הצבת סוגריים אופטימפלית במכפלה $A_{1...n}$מפצלת את המכפלה בין A_kלבין A_{k+1}עבור $k \in \mathbb{N}$ כלשהו. מכאן ש-

$$A_{1..n} = (A_{1..k})(A_{k+1...n})$$

הצבת סוגריים בתת הסדרה $A_{i..k}$חייבת להיות אופטימלית עבור תת סדרה זו, אחרת נקבל הצבת סוגרים אופטימלית יותר בסתירה.
תכונת תת המבנה האופטימלי מתקיימת - בדיוק לפי הפרד ומשול.
נגדיר - $M[i, j]$- המס' המינימלי של המכפלות הסקלריות הנדרשות לחישוב המטריצה $A_{i...j}$, כאשר כל הנתונים הללו נשמרים בM שהיא מטריצה דו ממדית. כאשר המשבצת שבאמת נרצה להחזיר בסוף היא $M[1, n]$ שבתוכה תהיה התשובה לכל השאלה שלנו - מה מס' המכפלות המינימליות? נראה כעת כי -

$$M[i,j] = \left\{ \begin{array}{cc} 0 & i = j \\ min_{1 \le k \le j}\{M[i,k] + M[k+1,j] + p_{i-1}p_k p_j & i < j \end{array} \right\}$$

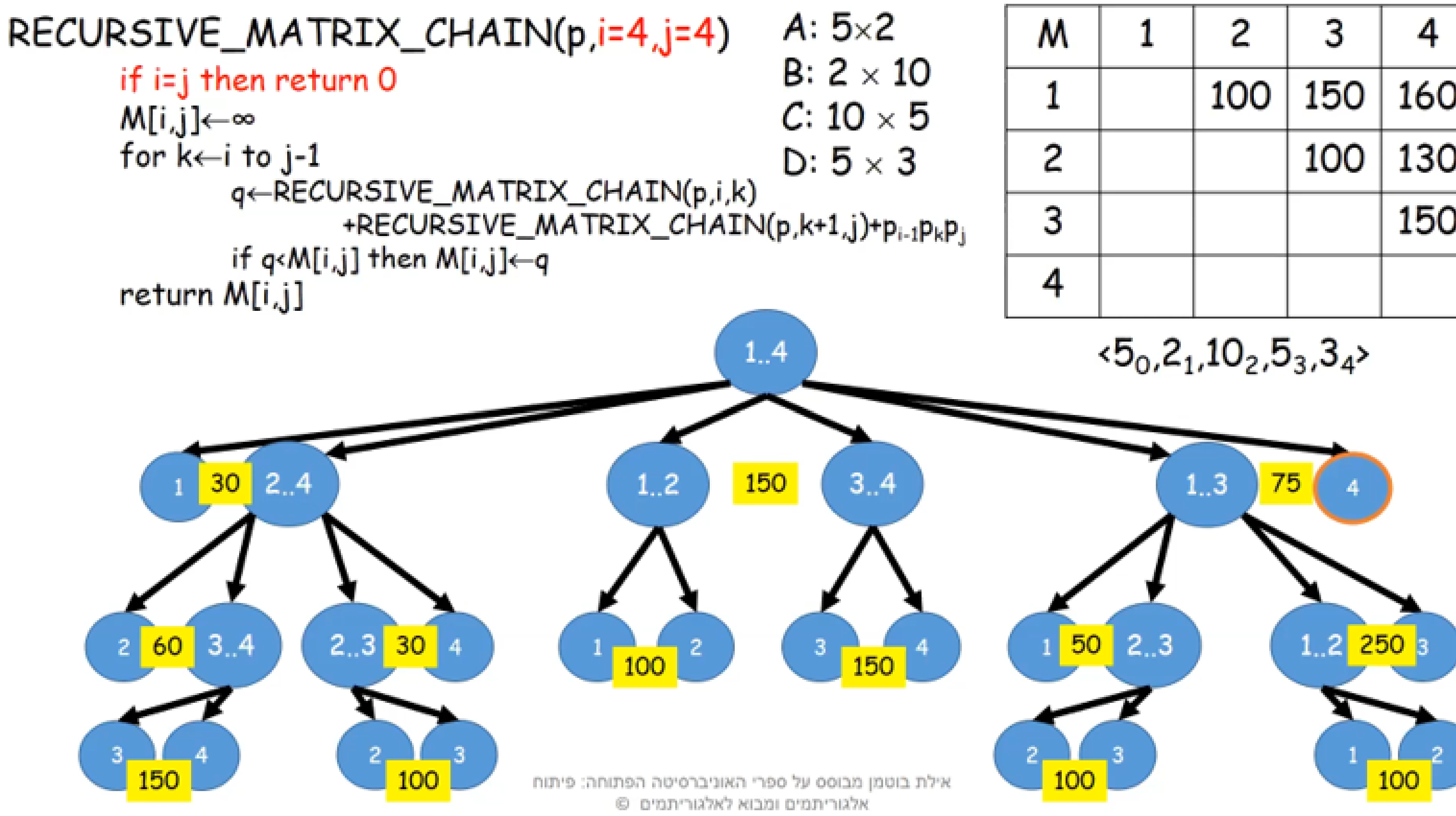
(מלא אינדקסים! פחד אלוהים! - בקטנה. זה בול כמו שראינו למעלה באינטואיציה רק פורמלי הרבה יותר ופורמלי=נקודות במבחן אז אנחנו אוהבים פורמלי)
נראה פסודו קוד - כאשר i, j הם הגדלים של המטריצה הכוללת וp הוא מערך שכולל את הגדלים כפי שתואר מעלה.

```
RECURSIVE_METRIX_CHAIN(p,i,j)
```

if i=j return 0
M[i,j]=max_Value
for k=i to j-1
$q = recursive - metrix - chain(p,i,k) + recursive - metrix - chain(p,k+1,j) + p_{i-1}p_kp_j$
if q<M[i,j] then M[i,j]=q
return M[i,j];

נראה כי k למעשה הוא האפשרויות בהם אנחנו רצים ובודקים מיהו המינימום. כאשר אכן k יכול להיות בין i לבין j - כמו בנוסחה מעלה. הערה נוספת היא שנכנסים בהתחלה אל ערך $M[i,j]$ מאוד גדול ואז נתחיל לבדוק האם יש קטנים ממנו.

דוגמת הרצה (מומלץ לעבור על מה שיש במצגת) -



מה העלות של הפתרון הרקורסיבי?

ראינו בדוגמה מעלה שקיבלנו עבור 4 מטריצות 160 במקרה האופטימלי, האם זה אומר לי משהו? לא. עלינו לראות דוגמה של n מטריצות. נחשב את נוסחת זמן הריצה ונקבל -

$$T(n) \geq \sum_{k=1}^{n-1}(T(k)+T(n-k)+1)+1 = \sum_{k=1}^{n-1}(T(k)+T(n-k))+n-1+1 = \sum_{k=1}^{n-1}(T(k)+T(n-k))+n = 2\sum_{k=1}^{n-1}(T(k))+n$$

טענה. $T(n) = \Omega(2^n)$ - הוכחה לבד באינדוקציה כמובן..... לא קשה.

כמו תמיד בתכנון דינאמי - כשנכשלנו וקיבלנו פתרון אקספוננציאלי גם בהפרד ומשול - נשים לב שיש קריאות שחוזרות על עצמן וננסה להעזר בזה בשביל להוריד את זמן הריצה לפולינומיאלי.

נצטרך לחזור לטבלה מלמעלה M - נסתכל עליה כמטריצה משולשית עליונה. בכל האיברים שמתחת לאלכסון הם לא רלוונטים לנו - נניח אפסים. בקריאות שמעל האלכסון נשים ערכים - כמו קודם. נרצה להשתמש בטבלה הדו ממדית ולהשתמש בנתונים שכבר חישבנו. ננראה כי -

$$A_{1...n} = (A_{1..k})(A_{k+1....n})$$

$$A_{i...i} + A_{i...j} \wedge i < j$$

$$\binom{n}{2} + n = \frac{n(n-1)}{2} + n = \frac{n^2+n}{2} = \Theta(n^2)$$

מה חישבנו כאן? נראה כי באידקסים מעל האלכסון יש בדיוק $\binom{n}{2}$ כאלו ־ וכן יש את האינדקסים על האלכסון ־ בדיוק n. מדוע בחרנו $\binom{n}{2}$? אין חשיבות לסדר ואין חזרה של זוגות.

מה עשינו כאן? רצינו למצוא **אינטואציה לפתרון הבעיה שלנו** ־ כלומר אם נצליח לשמור את הקריאות שלנו ולהעזר בהן ־ נגיע לזמן ריצה פולינומי של n^2. כלומר סה"כ הקריאות המקסימליות שנעשה יהיה cn^2 על המטריצה.

איך נמלא את המטריצה M? בדומה לבעיה הקודמת שראינו (כאן מעלה בסיכום) ־ המטריצה M שומרת לנו את כמות המכפלות ולא את סדר הסוגריים. נשמור כבר תוך כדי בצורה הבאה ־ נשים לב שנוכל לשמור מידע זה באמצעות מספר אחד ־ הרי כל כפל n מטריצות מחולק ל־2 ע"י מחיצה (סוגר) ולכן המספר 1 למשל, יעיד שהסוגר עבר כאן ־ $(A_1)(A_2A_3.....)$. כלומר נמלא במטריצה שני נתונים ־ 1. הערך המינימלי (כמו קודם) 2. היכן מאוחסן הסוגר. את המידע אודות היכן מאוחסן הסוגר ־ נשמור במטריצה S.

M	1	2	3	4
1				
2				
3				
4				

S	1	2	3	4
1				
2				
3				
4				

ברור כי באלכסון הראשי של המטריצה M תמיד יאוחסנו אפסים ־ מההגדרה מעלה: $i = j \Longrightarrow A_{ii} = 0$.

נראה כי תמיד בנסיון לחשב ערך תא במטריצה נהיה תלויים בשני ערכים קודמים ־ למשל

$$M[1,2] = M[1,1] + M[2,2] + P_0P_1P_2$$

**הערה ־ בניגוד לפתרון של הפרד ומשול, כאן באופן ברור אנחנו משתמשים בנתוני עבר ואילו קודם לכן אנחנו חישבנו אותם כל פעם מחדש.

תוך כדי שנמלא את M גם S תתעדכן בהתאם. נשים לב שבסוף שנסיים קל יהיה לנו להבין היכן להניח את הסוגריים ־ בהינתן הטבלה הבאה:

M	1	2	3	4
1	0	100	150	160
2		0	100	130
3			0	150
4				0

S	1	2	3	4
1		1	1	1
2			2	3
3				3
4				

נראה כי הערך המינימלי שלנו למכפלות יהיה 160, נלך לאותה משבצת במטריצה S ונקבל שמיקום הסוגר הראשון יהיה בערך 1, אח"כ נקבל את המצב הבא -

$$A_1(A_2A_3A_4)$$

ונרצה להבין היכן עכשיו הסוגריים נמצאות. זו תת בעיה של מיקום 2 עד 4 ולכן נלך לאותו מיקום בטבלה S ונראה כי מאוחסן שם הערך 3, כלומר המצב יראה כך -

$$A_1(A_2A_3)A_4$$

וכן נגיע כעת ל2 ו־3 וברור שאין לאן להמשיך ולכן זה הוא המצב האופטימלי לפתרון הבעיה.
כלומר בזכות S קל לשחזר כעת את המצב של הסוגריים במצב האופטימלי.
מה סיבוכיות זמן הריצה? מילאנו סדר גודל של n^2 משבצות, עבור כל אחת מהפעולות הלכנו למלא משבצת בטבלה S (n פעולות - שורה כפול עמודה) ולכן סה"כ סיבוכיות זמן הריצה שלנו היא $O(n^3)$

פסודו קוד לפתרון הבעיה - ללא רקורסיה - מלא אינדקסים וקצת מסריח:

```
MATRIX-CHAIN-ORDER(p)
n ← length[p] - 1
for i ← 1 to n
    do m[i, i] ← 0
for l ← 2 to n
    do for i ← 1 to n - l + 1
           do j ← i + l - 1
              m[i, j] ← ∞
              for k ← i to j - 1
                  do q ← m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j
                     if q < m[i, j]
                        then m[i, j] ← q
                             s[i, j] ← k
return m and s
```

אילת בוטמן מבוסס על ספרי האוניברסיטה הפתוחה: פיתוח

מה סיבוכיות זמן הריצה? מילאנו סדר גודל של n^2 משבצות, עבור כל אחת מהפעולות הלכנו למלא משבצת בטבלה S (n פעולות - שורה כפול עמודה) ולכן סה"כ סיבוכיות זמן הריצה שלנו היא $O(n^3)$.

וסיימנו - פתרנו כבר בעיה שניה של תכנון דינאמי. ירדנו מזמן ריצה פסיכי של $\Theta(\frac{4^n}{n^{1.5}})$ לזמן ריצה של $\Theta(2^n)$ לזמן ריצה פולינומי של - $O(n^3)$.

בעיה שלישית - תת סדרה משותפת הארוכה ביותר

הגדרה: בהינתן שתי סדרות X ו־Y נאמר שסדרה Z היא תת סדרה משותפת של X ו־Y אם Z היא תת סדרה של X **וגם** תת סדרה של Y.

למשל בהינתן $X = 1234$ ובהינתן $Y = 2345$ אז $z = 234$ היא תת סדרה משותפת שלהן (וגם המקסימלית במקרה זה, שהרי $z = 2, 3, 4, 23, 34$ הם תתי סדרות משותפות)

*יתכן שיש יותר מתת סדרה משותפת ארוכה ביותר אחת.

קלט: שתי סדרות $X =< x_1, ...x_n > Y =< y_1, ...y_n >$

פלט: תת סדרה משותפת ארוכה ביותר של X ו־Y.

שלב ראשון - פתרון נאיבי

כרגיל - נרצה למצוא פתרון נאיבי טריוואלי שנראה שיכשל -

צור את כל תתי הסדרות האפשריות של X , בדוק לכל סדרה שיצרנו האם היא תת סדרה של Y, מתוך תת הסדרות של X שהן תתי הסדרות של Y נמצא את הגדולה ביותר. (כלומר נאיבי לגמרי - נעבור על הכל ונבדוק אם זה הכי גדול...). מה יעילות פתרון כזה? לX יש 2^n תתי סדרות ולכן סה"כ הסיבוכיות תהיה $O(2^n)$

שלב שני - נמצא פתרון רקורסיבי באמצעות הפרד ומשול

כאן נרצה למצוא תכונה כלשהי שתעזור לנו לגשת באופן רקורסיבי לפתרון. את אותה תכונה נדרש להוכיח.

טענה: בהינתן שתי סדרות $X =< x_1, ...x_{n-1}, A >, Y =< y_1, ...y_{m-1}, A >$ יתקיים כי $Z =< .....A >$ (כלומר אם שתי סדרות נגמרות ב־A גם Z תסתיים בה).

הוכחה: נב"ש שלא מסתיים ב־A,אז נוסיף את A בסוף ואז זה בסתירה לכך שZ היא תת הסדרה הארוכה ביותר.
האם תמיד שתי סדרות יגמרו באותו תו? הלוואי אבל לא.
או ששתי הסדרות מסתיימות באותו התו ־ או שלא. (ראינו את זה כבר בדוגמה 1 של תכנות דינאמי..)
כעת, בהינתן שתי מחרוזות $X =< x_1, ...x_{n-1}, A >, Y =< y_1, ...y_{m-1}, B >$ ל־Z יש שלוש אפשרויות: Z מסתיים ב־A, Z מסתיים ב־B או שZ לא מסתיים באף אחת מהם.
קיבלנו הגדרה רקורסיבית.
הגדרה: בהינתן סדרה $X =< x_1, ...x_m >$ נגדיר את הרישא $(prefix)$ של הi של X עבור $0 \leq i \leq m$ כ־$X_i =< x_1, ...x_i >$.

משפט תת מבנה אופטימלי של תמ"א (תת מחרוזת ארוכה) ־ למשפט זה תפקידנו להגיע בזמן מבחן ולנסח פורמלית ־

יהיו $X =< x_1, ...x_m >, Y =< y_1, ...y_n >$ ותהי $Z =< z_1, ...z_k >$ תמ"א של X ו־Y.
א. אם $X_m = Y_n$ אזי $Z_k = x_m = y_m$ וכן Z_{k-1} היא תמ"א של X_{m-1} ו־Y_{n-1}.
ב. אם $X_m \neq Y_n$ אזי אם $z_k \neq x_m$ אזי Z היא תמ"א של X_{m-1} ו־Y.
ג. אם $X_m \neq Y_n$ אזי אם $z_k \neq y_n$ אזי Z היא תמ"א של X ו־Y_{n-1}.

הוכחה: לבד ־ זה לא קשה ודי טריוואלי. ההוכחה תהיה עם נב"ש.
כמו שראינו כבר במקרים קודמים, הנוסחה הרקורסיבית תחשב לנו את כמות האיברים במחרוזת המקסימלית ולא נדע את האיברים עצמם ־ זה ייעשה בהמשך.
נקבל את נוסחת הנסיגה הבאה ־ נגדיר $f(i, j)$ כפונקציה שמחזירה את אורך תת הסדרה הארוכה ביותר כאשר במחרוזת הראשונה אנחנו ב$[1, ..i]$ ובשנייה ב$[1, ..j]$.

$$T(x_i, y_j) = F(i, j) = \left\{ \begin{array}{cc} 0 & i = 0 \lor j = 0 \\ F(i-1, j-1) + 1 & x_i = y_j \\ max\{F(i-1, j), F(i, j-1)\} & otherwise \end{array} \right\}$$

ניתן לכתוב כתיבה של האלגוריתם עצמו ־ במבחן אין צורך כאן כי זה ממש טריוואלי.

מה סיבוכיות זמן הריצה?
נשים לב שבהינתן מקרה בו $X = Y$ ממש, אזי סיבוכיות האלגוריתם שלנו יהיה תמיד $O(n)$! אך זה המקרה הטוב שלנו.
ומה אם $X \neq Y$? נקבל עץ שלם וכמה עלים יש בעץ בינארי שלם? נקבל $\Omega(2^m)$ כלומר ־ מבלי לחשב את נוסחת הנסיגה הפסיכית הזו, במקרה הגרוע נקבל פתרון אקספוננציאלי. וזה מספיק בשביל לפסול.
ניתן לראות שבאותו עץ שלם נקבל מלא חזרות... מה שמוביל לשלב השלישי

שלב שלישי ־ מציאת פתרון שאינו רקורסיבי שמשתמש בעובדה שיש הרבה חזרות ־ תכנון דינאמי

ושוב ־ כמו בפעמים הקודמות נמצא עצמנו מגיעים לטבלה. נשים לב שאם נשמור את הנתונים הקיימים לא נצטרך לעבוד קשה ונגיע לסיבוכיות זמן ריצה של $O(m * n)$. נקח טבלה בגודל $n + 1$ על $n + 1$ ובמקום לעבוד ברקורסיה מהסוף להתחלה ־ נלך על מההתחלה אל הסוף. נמלא עמודה עמודה (או שורה שורה, לבחירתנו).
נראה שישר לפי האלגוריתם נוכל למלא חלק גדול מהלוח באפסים ־ את כל המקומות בהם i או jשווים לאפס. משם נראה שאת אותה "מטריצה דו ממדית" ניתן לחשב בקלות , כמו בדוגמה כאן ־

$$LCS(7,6) = \begin{cases} 0 & if\ i=0\ or\ j=0 \\ LCS(i-1,j-1)+1 & if\ i,j>0\ and\ x_i=y_j \\ max(LCS(7,6),LCS(6,6)) & if\ i,j>0\ and\ A\neq B \end{cases}$$

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	·1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

מכאן עברנו על מטריצה בגודל $k = max\{n,m\}$ של $k \times k$ ולכן סיבוכיות זמן הריצה היא $O(n^2)$.
נזכר כי קיבלנו את אורך הסדרה - אך לא את הסדרה עצמה. באופן דומה שוב, נקח טבלה נוספת של "חצים" כאשר חץ כזה ↖ ייצג לנו לקיחה של תו, חץ ← וחץ ↑ יסמנו תזוזה בתוך הטבלה, וכך נוכל לשחזר מאינדקס $[k,k]$ עד להתחלה את המחרוזת שאכן לקחנו (כמובן שנשתמש בפועל במספרים למשל $0,1,2$ שייצגו את החצים במקום הרעיון המופשט של חצים). נראה כי תוספת השחזור עלתה לנו בדיוק $n+n=2n$ צעדי הליכה על האלכסון ולכן הסיבוכיות שלנו הייתה ועודנה $O(n^2)$
וכמובן - השלב האחרון בתכנון דינמי הוא גם לצרף פסודו:

```
LCS-LENGTH(X, Y)
m ← length[X]
n ← length[Y]
for i ← 1 to m
    do c[i, 0] ← 0
for j ← 0 to n
    do c[0, j] ← 0
for i ← 1 to m
    do for j ← 1 to n
        do if x_i = y_j
            then c[i, j] ← c[i − 1, j − 1] + 1
                b[i, j] ← "↖"
            else if c[i − 1, j] ≥ c[i, j − 1]
                then c[i, j] ← c[i − 1, j]
                    b[i, j] ← "↑"
                else c[i, j] ← c[i, j − 1]
                    b[i, j] ← "←"
return c and b
```

*הערה - ניתן לחסוך סיבוכיות מקום ולוותר על המערך הנוסף, ולהחליט שמשתמשים במערך המקורי לתזוזה לשחזור המחרוזת כאשר נראה שבהינתן שני מספרים מעל ובצד של A_{ii} שווים יבחר

בו ואחרת יבחר המינימלי מבינהם - ולא נצטרך טבלה נוספת.

בעיה רביעית - סדרת פיבונאצ'י

את סדרת פיבונאצ'י ניתן לכתוב באמצעות נוסחת הנסיגה הבאה:

$$f(i) := \left\{ \begin{array}{cc} 1 & i=0 \\ 1 & i=1 \\ f(i-1)+f(i-2) & i\geq 2 \end{array} \right\}$$

אם ננסה לחשב את עץ הרקורסיה של בעיית פיבונאצ'י, נראה כי $f(i)=\Theta(2^n)$. זה לא פתרון פולינומי - וחבל, כי בתכלס מבצעים הרבה קריאות שחוזרות על עצמן! לא חבל לחשב כל פעם מחדש?

בתכנון דינמי נראה כי נוכל להשתמש בזכרון על מנת להמנע משימוש בקריאות חוזרות. נתבונן בפתרון הבא -

```
Fibonacci(n)
F[0] = 1
F[1] = 1
for i=2 to n
F[i] = F[i-1] + F[i-2]
return F[n]
```

יש סה"כ n קריאות רקורסיביות, לכן נשמור במערך בכל פעם את $f(i)$ במיקום $A[i]$. מה סיבוכיות הפתרון? $O(n)$ זמן! עוברים סה"כ על n איברים. ובמקום? $O(n)$ גם כן כיוון שניצלנו מערך באופן מלא. נראה כי ניתן לחסוך גם במקום - הכיצד? נוכל לשמור שני משתנים שייצגו בכל פעם את הקודם ואת הקודם קודם, נסמנם y ו־z, ובכל פעם נעדכן אותם. נקבל שסיבוכיות מקום תהיה $O(1)$.

בעיה חמישית - תת סדרה בלתי תלויה בעלת ערך מקסימלי

הגדרה: בהינתן סדרה $(a_1,....a_n)$ ערך הסדרה הינו $\sum_{i=1}^{n}a_i$.

הגדרה: בהינתן סדרה $(a_1,....a_n)$, תת סדרה בלתי תלויה של הסדרה A' שתוגדר כך - $A'=(a_{i1},....a_{ik})$ היא כזו שתקיים:

1. $1\leq i_1\leq ...\leq i_k\leq n$
2. לכל $1\leq j\leq k-1$ מתקיים $i_j<i_{j+1}-1$, דהיינו בתת הסדרה אין שני איברים שהיו רצופים בסדרה המקורית.

נרצה למצוא בהינתן מערך A את תת הסדרה הבלתי תלויה עם ערך מקסימלי. כלומר, גם אין שני איברים רצופים וגם ערכה מקסימלי. ערכי הסדרה לאו דווקא מספרים שלמים וחיוביים.

אלגוריתם נאיבי: נאתחל משתנה $max=-\infty$ ולכל סדרה A' של A נבדוק אם A' הינה סדרה בלתי תלויה, אם A' היא סדרה בלתי תלויה נחשב את ערך הסדרה A', אם הערך גדול מmax נציב את הערך במקס, וכן נגדיר $B=A'$, נחזיר את (B,max). כמה תתי סדרות יש? 2^n לפחות, אח"כ נעבור כל פעם עוד n פעמים. סה"כ פתרון אקספוננציאלי של $\Theta(n*2^n)$. גרוע.

אלגוריתם הפרד ומשול:

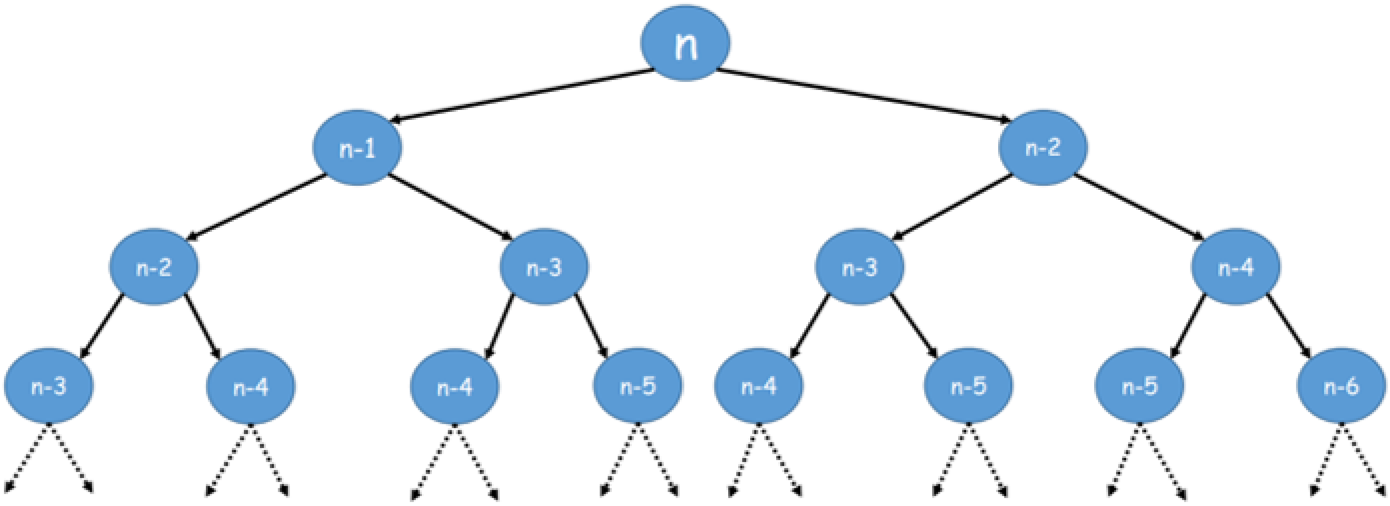
תובנה טריויאלית: תהי $A^*=(a_{i1},...a_{ik})$תת סדרה בלתי תלויה עם ערך מקסימלי של A. המספר האחרון עם אינדקס n יכול להיות שייך לA^* או לא להיות שייך לו. כלומר, $i_k=n$ או $i_k<n$.

תובנה שנייה: אם $a_n\notin A^*$ אזי A^* הוא הפתרון האופטימלי למופע של הבעיה המבוססת על הסדרה $(a_1,...a_{n-1})$ - כלומר הקטנו את הבעיה באחד

תובנה שלישית: אם $a_n \in A^*$, אזי $A^*/\{a_n\}$ הוא הפתרון האופטימלי למופע של הבעיה המבוססת על הסדרה $\{a_1, ...a_{n-2}\}$ - כלומר הקטנו את הבעיה בשתיים
מהתובנות הללו נוכל להגיע לנוסחת הנסיגה הבאה: כאשר f מייצג את אורך הסדרה הכי גדולה וכן i הוא אורך המערך.

$$f(i) := \left\{ \begin{array}{cc} 0 & i = 0 \\ max\{0, a_1\} & i = 1 \\ max\{f(i-1), a_i + f(i-2) & i \geq 2 \end{array} \right\}$$

תובנה רביעית: ערך פתרון אופטימלי למופע של הבעיה המבוססת על הסדרה $(a_1, ...a_n)$ הוא גדול או שווה לערך פתרון אופטימלי למופע של הבעיה המבוססת על הסדרה $(a_1, ...a_{n-1})$.
מה יעילות הנוסחה הרקורסיבית? אם נפתח את נוסחת הנסיגה, נקבל את עץ הרקורסיה הזה. הוא בדיוק כמו עץ פיבונאצ'י!



מכאן שקיבלנו סיבוכיות לנוסחת הנסיגה של $\Theta(2^n)$- לא פתרון פולינומי! מה עשינו לא טוב? יש מלא קריאות שחוזרות על עצמן! נתבונן באלגוריתם הבא -

```
MAX_INDEPENDENT_ARRAY(A[1..n])
F[0] = 0
F[1] = max(0,A[1])
for i=2 to n
F[i] = max(F[i-1], A[i]+F[i-2])
return F[n]
```

מה עשינו כאן? שמרנו את הקריאות שחוזרות על עצמן באמצעות מערך. שמרנו מקום במערך ולכן סיבוכיות מקום הינה $O(n)$, וכן גם סיבוכיות זמן הריצה הינה $O(n)$! סה"כ ירדנו לפתרון פולינומי, כנדרש. נראה שנוכל כמו בפיבונאצ'י להוריד גם את סיבוכיות המקום ובמקום לשמור במערך לשמור X Y שישמרו לנו את המידע.
נשים לב כי אמנם יכולנו לקבל את אורך הסדרה המקסימלי, אך לא את תת הסדרה באורך מקסימלי עצמה: לכן נרצה לשחזר את הפתרון שלנו. כיצד? נעשה $backtracking$ אחורה מהרגע שסיימנו, כלומר בכל פעם נשווה עם איבר שנצטרך להוסיף לסדרה או שלא.

בעיה שישית - תת סדרה עולה ארוכה ביותר

כעת, נרצה בהינתן סדרה למצוא את תת הסדרה שלה שתהיה עולה הארוכה ביותר. אם ניגש לפתרון הנאיבי, ישנן בהינתן n איברים 2^n תתי סדרות. נרצה לעבור על כל תתי הסדרות, ולבדוק האם הן סדרות עולות. זה יעלה $O(n)$ לכל סדרה, וכן נרצה בסוף גם לחשב את המקסימום מבין הסדרות העולות. סה"כ יעלה לנו $O(n2^n)$, אקספוננציאלי ולא יעיל בעליל.

כעת ניגש לפתרון הרקורסיבי, נראה שאם נגדיר פונקציה שמוצאת את תת הסדרה המקסימלית $f(i)$ באינדקסים $i-1$, זה לא יעזור לנו, שכן אנחנו נדרשים בכל רגע נתון לשתי אפשרויות: אם x_i לא בסדרה, אוקיי, אכן נוכל להשתמש בערך $f(i-1)$, אבל מה אם הוא כן בסדרה? צריך לדעת מי היה האיבר שלפניו. לכן נגדיר את הפונקציה הבאה: $f(i)$ תחזיר את תת הסדרה המקסימלית שמסתיימת באינדקס i. כעת, קל יותר לגשת להגדרה הרקורסיבית. נגדיר את הפונקציה

$$f(i) = 1 + max\{(f(j)|1 \leq j \leq i \wedge x_j \leq x_i) \vee 0)$$

נראה כי כעת הפונקציה מוגדרת היטב, כיוון שx_i הוא האיבר האחרון בסדרה, כעת כל שנותר הוא להשוות את כל האינדקסים שלפניו אליו ולבדוק אם קטנים ממנו, שהרי $f(j)$ מחזיק את תת הסדרה הארוכה ביותר עד לאינדקס j. כמובן שיתקיים $1 \leq i \leq n$, נשמור את הנתונים בתוך מערך בגודל n, נתחיל למלא אותו משמאל לימין (כמו מילוי מערך רגיל), נראה כי מילוי כל תא יקח לנו $O(n)$ בשל הצורך לעבור במקרה הגרוע על $n-1$ אינדקסים, וכן יש n תאים ולכן סיבוכיות הזמן תהיה $O(n^2)$ והמקום $O(n)$, כעת כל שנותר הוא להצביע על הערך המקסימלי - הפתרון יהיה $max_{1\leq i\leq n}(f(i))$- כלומר למצוא את הפתרון נצטרך לעבור על כל התאים, כמובן זה לא ישפיע על הסיבוכיות האסימפטוטית בזמן הריצה. נראה כי לא נוכל להקטין את השימוש במקום, שכן ממילא עלינו בסוף לעבור על כל המערך שיצרנו ולחשב את המקסימום.

בעיה שביעית - בעיית הסטודנטים

הבעיה: ישנה רשימה של n סטודנטים, המסודרים ע"פ סדר לקסיגוגרפי של שמות משפחה (לא ניתן לשנות את הסדר). בנוסף, נתון מידע על חברויות בין זוגות סטודטנים (כלומר, לכל זוג סטודנטים, נתון האם הם חברים האחד של השני או שאינם חברים). ניתן לחלק את הסטודנטים ע"י העברת קו מפריד בין שני סטודנטים ברשימה (לא ניתן "לערבב" סטודנטים מאיזורים שונים של הרשימה). מעוניינים לחלק את הסטודנטים ל-k קבוצות כך שבכל קבוצה יש לפחות 2 סטודנטים. המטרה היא למצוא את (מספר החברויות ב) חלוקה חוקית שמביאה לסך חברויות מכובדות גדול ביותר בתוך קבוצות הלימוד. כלומר, לכל קבוצה בודקים כמה חברויות מוגדרות בה, וסוכמים על פני כל הקבוצות.

הפתרון: האלגוריתם הנאיבי יהיה לעבור על סך הקבוצות $\binom{n-1}{k-1}$, לבדוק את כולם, לסנן חלוקות לא הגיוניות (אם יש פחות מ-2 חברים) ולחשב את הקבוצה עם מס' החברים הכי גדול. סה"כ זה $\Omega(\binom{n-1}{k-1})$. נגש לרקורסיבי. נוכל להקטין את הקלט משני הכיוונים - מס' הסטודנטים ומס' הקבוצות. כל חלוקה של n סטודנטים לk קבוצות נוכל לתאר כבחירה של גודל הקבוצה האחרונה (נסמן כs) וחלוקה של שאר הסטודנטים $(n-s)$ הראשונים ל$k-1$ קבוצות בצורה אופטימלית. לכן נגדיר $f(i,j)$ מס' החברויות הגדול ביותר שניתן ליצור בחלוקת i הסטודנטים הראשונים לj קבוצות. נראה כי

$$f(i,j) = max_{2\leq 2\leq i-2(j-1)} f(i-s, j-1) + R(i-s+1, i)\}$$

כאשר $R(x,y)$ מייצג את מס החברויות שיש בקבוצה שמכילה סטודנטים מאינדקס x עד לאינדקס y.

ניצור מטריצה $n\times k$, נוכל למלא כל תא בה לפי ההגדרה ב$O(n)$ אם נמלא לפי עמודות, ואז הסיבוכיות זמן תהיה $O(n^2k)$, באשר למקום $O(nk)$, נוכל לצמצם את סיבוכיות זמן המקום אם בכל זמן נתון נשמור שני עמודות אחרונות, ואז סיבוכיות המקום תהיה $O(k)$.

כעת יש לבצע חישוב מקדים בשביל לחשב את R בזמן $O(1)$. נניח שהקלט הוא כדקלמן: $A[i,j] = \left\{\begin{array}{ll} 1 & i-is-friend-of-j \\ 0 & o.w \end{array}\right\}$, כלומר הקלט אודות החברויות הינו נתון במטריצה משולשית עליונה, כלומר כל תא $A[i,j]$ מכיל את מס' החברויות של סטודנט i עם סטודנט j. נחשב

מטריצה $B[i,j]$ שכל $B[i,j]$ יכיל את מס' החברויות של סטודנט j עם סטודנטים מ$[i,....j]$. בדיוק כפי שהנוסחה דורשת.

$$B[i,j] = \sum_{t=i}^{j} A[t,j] = A[i,j] + \sum_{t=i+1}^{j} A[t,j] = A[i,j] + B[i+1,j]$$

כלומר ניתן לחשב את המטריצה B ע"י חישוב לפי עמודות בצורה יעילה, כך שכל תא יסתמך על קודמו ויעלה $O(1)$. לבסוף נרצה לחשב את $R[i,j]$

$$R[i,j] = \sum_{t=i}^{j} B[i,t] = B[i,j] + \sum_{t=i+1}^{j} B[i,t] = B[i,j] + R[i,j-1]$$

כלומר את הערכים כאן נוכל לחשב שורה שורה מהאלכסון הראשי ימינה, סה"כ העיבוד המקדים יעלה $O(n^2)$. **בבעיה זו השתמשנו בעיבוד מקדים בשביל לפתור אותה.**

חלק XVIII

מבוא לאלגוריתמים חמדניים

אלגוריתם חמדן הוא אלגוריתם שמוצא פתרון אופטימלי לבעיה ע"י כך שבכל שלב נבחרת האפשרות שנראית הטובה ביותר באותו הרגע.

מתי אלגוריתם חמדן פותר את הבעיה ומתי לא? הוא פותר בהינתן התכונות הבאות:

* קיים פתרון אופטימלי שמתחיל בבחירה החמדנית שהאלגוריתם בוחר.

* תת מבנה אופטימלי: לאחר הבחירה החמדנית הראשונה, הבעיה מצטמצמת למציאת הפתרון אופטימלי לבעיה שמתיישבת עם הבחירה הראשונה.

הערה: לא לכל בעיה האלגוריתם החמדן נותן את הפתרון האופטימלי , אך ישנן בעיות שעבורן הוא מתאים.

למשל: בהינתן קבוצת מטבעות וסכום עודף, מצא את קבוצת המטבעות המינימלית שזהו סכומה. כלומר מס' מטבעות מינימלי. תמיד נוכל לבחור את האפשרות הכי טובה ברגע נתון. למשל בהינתן סכום 36 ומטבעות $1,5,10,20$ נרצה לקחת ממש את המטבעות 20 בהתחלה אחכ 10 אחכ 5 ואחכ 1. כל פעם המקסימלי שניתן לקחת.

"חמדן" - מקסימום מקומי

בעיית התרמיל

קלט: $X = \begin{pmatrix} x_1 \\ . \\ . \\ . \\ x_n \end{pmatrix}$. כאשר לכל $1 \leq i \leq n$ מתקיים $x_i \in \mathbb{N}$. וכן מס' טבעי s.

פלט: וקטור $B = \begin{pmatrix} b_1 \\ . \\ . \\ . \\ b_n \end{pmatrix}$ בינארי, כלומר $b_i \in \{0,1\}$. כך ש$S = \sum b_i x_i$. כלומר נרצה למצוא תת קבוצה של איברים שסכומה S.

פתרון: ראינו את הבעיה בתרגול אך הגענו לפתרון פולינומי לא לינארי. כעת נניח הנחה סמויה ונראה שנרד ללינארי.

פתרון נאיבי: לכל וקטור בינארי B בדוק אם $S=\sum b_i x_i$. אם כן החזר אמת, החזר שקר. כמה וקטורים בינאריים יש? לפחות $\Omega(2^n)$.

פתרון חמדני: נסתכל על אלגוריתם שלא יעבוד: נמיין את סדרת המשקולות X בסדר יורד, נסמן את הסדרה שהתקבלה Y ונציב $t=s$. לכל i בין 1 לn נבצע: אם $t-y_i$ גדול שווה אפס, אזי נקבע $b_i=1$ וכן $t=t-y_i$. אחרת, $b_i=0$. אם $t=0$ נחזיר אמת, אחרת שקר. מדוע הוא לא פותר את הבעיה? כי אם ניקח תמיד את המספר הראשון שנראה הוא יכול להכשל למרות שכן קיים פתרון אחר. בעיית התרמיל היא NP (ניתן לוודא האם אכן מדובר בזמן פולינומי).

הגדרה - סדרת סכומים עולה: סדרה X תקרא סדרת סכומים עולה אם $x_i > \sum_{j=1}^{i-1} x_j$ (כלומר איבר גדול מסכום כל קודמיו)

טענה: קיים אלגוריתם חמדן לבעיה.

הוכחה: **נשתמש בהנחה סמויה כי אכן הסדרה X היא סדרת סכומים עולה.** מיין את סדרת המשקולות X בסדר יורד, נסמן את הסדרה שהתקבלה Y ונציב $t=s$. לכל i בין 1 לn נבצע: אם $t-y_i$ גדול שווה אפס, אזי נקבע $b_i=1$ וכן $t=t-y_i$. אחרת, $b_i=0$. אם $t=0$ נחזיר אמת, אחרת שקר. מדוע זה עובד? זה אותו פתרון מקודם. הפעם הייתה לנו הנחה סמויה, שעזרה לפתרון. מה עלות הפתרון? בגלל המיון - $\Theta(nlogn)$.

חלק XIX

$?P=NP$

מחלקת הסיבוכיות P: מחלקת כל הבעיות שניתנות לפתרון דטרמיניסטי (לא רנדומי) בזמן ריצה פולינומי.

מחלקת הסיבוכיות NP: מחלקת הבעיות שפתרונותיהן ניתנות לאימות בזמן ריצה פולינומי.

האם $N=NP$? לא ידוע. יש פרס של מליון דולר למי שיצליח לפתור אותה. לא ידוע האם $P=NP$. ההנחה הרווחת $P\neq NP$.

NP **קשה:** מחלקת הבעיות שקשות לפחות כמו הבעיות הקשות ביותר בNP.

חלק XX

קוד הופמן - אלגוריתם חמדן

מדובר בקוד לדחיסת נתונים. דחיסת נתונים: יש הודעה גלויה שנרצה לדחוס. יש מידע, נקודד אותו להודעה דחוסה שתשתמש בפחות זכרון, נרצה שיהיה מפענח שבהינתן הודעה דחוסה נפתח אותה ונקבל את ההודעה. ההודעה תקרא M, הדחוסה C וההודעה הפתוחה לאחר המפענח M'. בדחיסה לא הפסדית מתקיים $M=M'$. בדחייה הפסדית מתקיים $M\neq M'$.

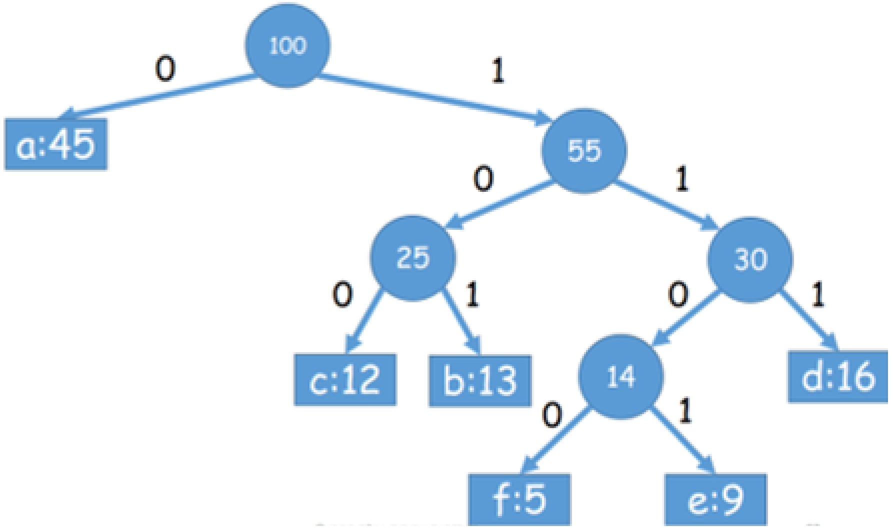
שיעור הדחיסה יהיה $\frac{|M|}{|C|}$ וכן $|X|$ מסמן את מס' הביטים במחרוזת X. הערה - אנחנו לא בעולם אסימפטוטי. אנחנו ממש רוצים להתייחס לקבועים. מדוע נרצה לדחוס? להקטין שטח אחסון, להקטין זמן תקשורת בהעברת מידע ולחסוך זכרון.

אלגוריתם דחיסה חמדני שימושי מאוד:

*קוד באורך קבוע: למשל, נתון - קובץ שמכיל $100,000$ תאים. נתונה גם טבלת שכיחויות של כל אחד מהתווים שיופיע. כמה אותיות ניתן לייצג עם ביט 1? 2^1. כמה אותיות עם 3 ביטים? $2^3=8$. עם k ביטים? 2^k.

נסמן בn את כמות האותיות. $2^k=n$. כלומר $k=logn$. הרעיון בקוד הופמן - לייצג אות עם שכיחות גבוהה בכמה שפחות ביטים. למשל אם האות a מופיעה $50,000$ פעמים והאות c מופיעה $20,000$, נעדיף שהאות a תיוצג עם כמה שפחות ביטים - למשל 01 ואילו c תהיה 010 למשל. אם נשתמש בשיטה זו מס' הביטים לייצוג המחרוזת ירד. נדרוש כי אף מילת קוד לא תהיה תחילית של מילת קוד אחרת - בשביל לפענח נכונה את הטקסט.

ייצוג קוד באמצעות עץ בינארי:
* כל העלים יהיו באותו גובה, בתוך כל $node$ יהיה סכום השכיחויות לכל תת עץ. מקרה נאיבי - כל מס' ייוצג באמצעות 3 ביטים. בעלים יהיו התווים שלנו כולל שכיחויותיהם. למשל - בעלה x_1 יופיע $a-20$ ובעלה x_2 שכן שלו יהיה $b, 52$. אזי באבא של העלים X יהיה כתוב 72.
אחרת - לא נאיבי: ככל שאתה יותר נפוץ תהיה גבוה יותר בעץ.



טענה: עץ בינארי שאינו מלא (כמו בדוגמה עם ה3 סיביות) לא יכול לייצג קוד תחיליות אופטימלי.
נוסחה לחישוב כמות סיביות לקידוד קובץ:
נסמן $f(c)$ שכיחויות של איבר c בקובץ.
$d_T(c)$ עומק העלה c בעץ = אורך מילת הקוד של c.
הנוסחה למס' הסיביות תהיה

$$B(T) = \sum_{c \in C} f(c) d_T(c)$$

$B(T)$ נקרא העלות של העץ T. מסקנה - בעץ הקוד האופטימלי יש n עלים שהם כמות התווים בטקסט ו $n-1$ קודקודים פנימיים.
קוד הופמן : האלגוריתם בונה את העץ T המייצג את הקוד האופטימלי מלמטה למעלה. מתחיל עם קבוצה של $|C|$ עלים, מבצע $|C|-1$ פעולות מיזוג ליצירת העץ הסופי. (יצירת הקודקודים הפנימיים).
נחשב שכיחות של כל ביט. יצור תור קדימיות לערכי שכחויות. נסדר ת כל הביטים בתור עלים ונתחיל למזג צמתים על מנת לבנות את העץ: 1. נוציא את שתי השכיחויות המינמליות בתור 2. עבור שתי שכיחויות המינימום ניצור צומת חדש, כאשר נגדיר את הקטנה להיות בן שמאלי והגדולה להיות בן ימני. 3. נוסיף את צומת האב – כלומר סכום השכיחויות שלהם לתור הקדימויות 4. נחזור על הלולאה לעיל עד שלא נותרו יותר ערכים בעץ 5. נסמן כל צלע ימנית ב 1 וכל שמאלית ב 0.
איך ניגשים לשאלה כזו? בהינתן טבלת שכיחויות ואותיות -
א. מסדרים את כל האותיות וממינים אותם בהתאם לשכיחויות שהם מופיעים $O(nlogn)$
ב. שמים את כל האותיות בעץ בעלים ובונים אותו מלמטה למעלה, בכל פעם מחברים את שני הערכים עם השכחויות הקטנות ביותר. כאשר מחברים, השכיחות החדשה היא סכום השכיחויות. לבסוף מגיעים לשורש עץ. הבנייה תעלה $O(n)$
ג. עוברים על העץ מהראש לעלים. בכל פניה שמאלה בעץ שמים 0 ובכל פנייה ימינה בעץ שמים 1. $O(n)$
ד. כעת, עבור כל אות אנחנו מתחילים לחשב מלמטה עד למעלה (לשורש) ומחשבים את הקידוד שלה כאשר הקידוד הוא הביטים שלאורך המסלול מהעלה לשורש.

ה. קיבלנו קידוד אופטימלי סה"כ עלה $O(nlogn)$.
נשים לב שמדובר באלגוריתם חמדני. מדוע חמדני? בכל פעם בחרנו בתת בעיה שלקחה את שני הערכים של השכיחות הקטנים ביותר וכך בנינו את העץ. זו הייתה בחירה חמדנית שכן מי אמר שזה יוביל לתת בעיה של הפתרון המקורי? ובכן הוכחנו זאת בהרצאה (לא נוכיח כאן) אך מדובר באלגוריתם חמדני.

חלק XXI

מיונים

נמיין את האיברים לפי מפתח מסויים, ונגדיר תמיד כאשר ממיינים מערך את הסדר בין כל שני איברים עם מפתח שונה כיחס הסדר בין המפתחות. אם יש שני איברים עם מפתח זהה - מבחינתנו אין דרישה מי מהם יהיה קודם אלא רק שיהיו סמוכים זה לזה.

-

מיון הכנסה ($Insertion-sort$)
באיטרציה ה־i מוודאים שהרישא $[1,..i]$ ממוינת. תחילה מוצאים את מקומו של האיבר ה־i ביחס ל־$i-1$ האיברים הראשונים, ואז מכניסים אותו במיקום זה ומזיזים את האיברים שאחריו מיקום אחד ימינה.
מויקיפדיה: בכל איטרציה (מעבר על המערך) נלקח איבר ממערך הקלט, ומוכנס למקומו הנכון בתוך ה"תת־מערך" הממוין שנבנה, במהלך המיון, בחלק השמאלי של המערך. לאחר השלב הראשון כולל האזור הממוין במערך את שני האיברים הראשונים, לאחר מכן את שלושת האיברים הראשונים וכן הלאה. כך עד לסיומו של מערך הקלט. המיון נעשה במקום, כלומר ללא צורך בזיכרון נוסף, פרט למערך עצמו ולתא עזר בודד.
תמיד נקח את האיבר השני ונשווה אליו את כל האיברים שלפניו, אח"כ נעשה זאת עם השלישי וכן הלאה.... כלומר בכל פעם יש החלפה (אם יש צורך) ואז סריקה מהתחלה עד האיבר ה־i לוידוא שאכן המערך ממויין.
סיבוכיות זמן הריצה - $O(n^2)$

-

מיון בועות ($Bubble-sort$)
עוברים על המערך מתחילתו לסופו, כל פעם שרואים איברים סמוכים כך שהראשון גדול מהשני מחליפים ביניהם. המיון יסתיים כאשר יהיה מעבר כלשהו שלא התבצע בו שום שינוי (נבדוק זאת עם $flag$). אם נרצה לעשות פסודו - נעשה דאבל פור כאשר הפור הראשון ירוץ עד n, והשני עד $n-i$ כאשר אם נראה מצב שבו הערכים צריכים להתחלף נעשה $swap$.
סיבוכיות זמן הריצה - $O(n^2)$

-

מיון בחירה ($Selection-sort$)
באיטרציה הi דואגים כי i האיברים הקטנים במערך יהיו ממוינים בתחילתו. בכל איטרציה מוצאים את האיבר המינימלי מבין האיברים שטרם מוינו ומביאים אותו למקומו.
סיבוכיות זמן הריצה - $O(n^2)$

-

מיון מהיר ($Quick-sort$)
בכל שלב בוחרים איבר ציר כלשהו (או הראשון או באקראי) ומסדרים את המערך כך שהאיברים הקטנים מהציר יהיו משמאלו והגדולים ממנו מימינו. נמיין באופן רקורסיבי את האיברים. **ראינו בפרק "הפרד ומשול"** כי סיבוכיות זמן הריצה בתוחלת - $O(nlogn)$.

-

מיון מיזוג ($Merge-sort$)
מחלקים את המערך לשני חצאים, ממיניים כל חצי ולבסוף ממזגים את שני החצאים הממוינים למערך אחד ממוין. כמובן שנרוץ עד למערך בגודל 1.
סיבוכיות זמן הריצה - $O(nlogn)$ אך נזכור כי יש לנו גם סיבוכיות מקום!

-

מיון AVL

כפי שראינו בעבר - מכניסים את כל האיברים לעץ AVL - זה עולה לי $nlogn$, מדפיסים את כל האיברים בעץ ב$in-order$- הוכחנו בתרגיל הבית שזה אכן מדפיס בצורה ממוינת ב-$O(n)$

סיבוכיות זמן הריצה - $O(nlogn)$

-

מיון ערימה ($Heap-sort$)

כפי שראינו, ניצור ערימה $O(n)$ - ואז בכל פעם נוציא את איבר המינימום מהערימה ונסדר את העץ - $nlogn$. ראינו זאת. היתרון - אין צורך במקום.

סיבוכיות זמן הריצה - $O(nlogn)$

מיונים מיוחדים

בהנחה שידוע לנו מידע נוסף על הקלט ולא רק ביסוס השוואות בין שני איברים, ניתן לכתוב אלגוריתמים שירוצו ב$o(nlogn)$ זמן, כלומר בפחות מהחסם התחתון שראינו למיון מבוסס השוואות.

1. מיון מניה - $Count-sort$**:** נתון מערך A בגודל n כך שכל הערכים בA הם מספרים שלמים בתחום $[0,R]$. כל מספר הוא מזהה וצמוד לו מידע נוסף, לשני עותקים שונים של אותו מזהה יכולים להיות מצורפים מידעים שונים. למשל- $[(3,"alice"),(1,"bob"),(3,"charlie"),(2,"dave")]$ כאן 3 מופיע פעמים אבל עם מידע שונה. האלגוריתם יפעל כך -

צור מערך חדש C בגודל R כך שהתא i שמש לספירת מס' המופעים של הערך i במערך A. כדי לדעת היכן לשים את האיברים נחשב את סכומי הרישות של C, כלומר התא הi ישמור את מס' הערכים בA שהם קטנים שווים מi. (כלומר - נעבור על מערך C ולפיו ניתן לדעת כיצד ניתן למקם את האיברים החדשים במערך הממויין. אנחנו יודעים כי יש למשל פעם 1, פעם שניים ופעמיים שלוש. לכן שלוש יופיע בשני האינדקסים האחרונים). כעת במערך C ישמר המידע הבא: כמה איברים קטנים ממני. אם קודם C היה המערך: $1,1,2$. עכשיו הוא יהפוך ל$1,2,4$. (סוכמים כמה קטנים ממני עד כה כל פעם). כעת נעבור חזרה על האיברים מהסוף להתחלה ונוכל למיין באמצעות המידע הזה ששמור לאן צריך לשלוח את האיבר (לפי כמה קטנים ממנו) ונעדכן את המידע תוך כדי.

לסיכום: שלושה שלבים. ראשון הוא מניית מס' מופעים של כל איבר $O(n)$. אח"כ חישוב סכומי הרישות (מיקומי האחרונים מכל סוג) $O(R)$. אח"כ העתקת האיברים למערך החדש עם השימוש במידע ממערך העזר לפי כמה קטנים ממנו $O(n)$. סה"כ $O(n+R)$ זמן וכן גם מקום. הערה - בהינתן מספרים מהסכומים $[0,cn]$ עבור $c>0$ ניתן למיין ב$O(n)$ זמן.

2. מיון בסיס - $Radix-Sort$**:** נתונה קבוצה של מספרים S באורך n מתוך $\{0,1,...R^d-1\}$. למשל - מיון מספרים בבסיס 10 עד מליון. הרעיון - מיון לפי הספרות של המספר. הבחנה - ניתן למיין את המספרים לפי הספרה השמאלית ביותר MSB ולקבל מיון גס. אך נרצה לשפר למיון מדוייק. הרעיון הוא כזה: לכל i בין 1 לd מיין את המערך A במיון יציב לפי הספרה הi (כלומר, כאשר d הוא האורך הכי גדול של מספר בתחום, למשל 1000 אז $d=4$, בצע מעבר על הספרה הi של המספר ומיין לפיה. כאשר אתה מתחיל מערך האחדות תמיד, וממשיך למיין לפי ערך המאות, אלפים וכו'. אכן האלגוריתם הזה ממיין (ההוכחה לא כאן). זמן הריצה: לכל אחת מהעמודות נבצע מיון מניה, בזמן $O(n+R)$, יש d עמודות ולכן סה"כ $d(n+R)$, כלומר $O(d(n+R))$. מקום ידרש $O(n+R)$. עבור $R=n$ נקבל כי המיון הוא ב$O(d*n)$.