מספר קורס: 202.1.1031

מרצה: מיכל שמש

מתרגל: קרן

יוצר המסמך: דניאל שלמה

האגודה הסטודנטיאלית
אוניברסיטת בן-גוריון בנגב

סיכום הרצאות

מבני נתונים

2022

באדיבות מדור אקדמיה, אגודת הסטודנטים, אוניברסיטת בן גוריון.
www.bgu4u.co.il

02.06.22

Data structures - summary

– Runtime analysiss

Data structure	Search	Insertion	Delete	Succ/Pre	Max/Min	Height	Size	ADT
BST	θ (h) Iterative/recursive	θ (h)	θ (h)	θ (h) without fields	θ (h)	In worst case: $h = n - 1$ In best case: $h = \log(n)$	n	Dynamic set
AVL	θ (log n)	θ (log n)	θ (log n)	θ (log n)	θ (log n)	$h = \theta$ (log n) Proof: Fibonacci number	n	Dynamic set
B-tree	$\theta(t \log_t n) = \theta(\frac{t}{\log t} \log n)$	$\theta(t \log_t n)$	$\theta(t \log_t n)$	$\theta(t \log_t n)$	$\theta(t \log_t n)$	At most $\log_t \frac{n+1}{2}$	n	Dynamic set
Skip-list	θ (log n) expected	$\theta(\log n)$ expected	$\theta(1)$ expected	$\theta(1)$ worst case	$\theta(1)$ worst case	At most $\log n + 2$	$2n + O(\log n)$	Dynamic set
Hash (chaining)	$\theta(1+\alpha)$ expected $\theta(\log n / \log \log n)$ worst case	in $\theta(1)$ worst case if no rehashing ($\theta(n)$)	$\theta(1+\alpha)$ if no rehashing ($\theta(n)$) expected	Expected length $T[h(k)]$ $(k \epsilon U)$ *is at most* $1 + \frac{n}{m}$			n	Dictionary
Hash (Adressing)	$\theta(1)$ expected $\theta(n)$ *worst*	$\theta(1)$ expected without rehashing $\theta(n)$ *with rehashing*	$\theta(1)$ expected In both methods $\theta(n)$ *with rehashing*	-			n	Dictionary

02.06.22

Data structures - summary

Data structure	**Functions**								
heap	MaxHeapify	BulidMaxHeap	Maximum	ExtractMax	Increase key	Insertion		Heapsort	Type
	$\theta(\log n)$	$\theta(n)$	$\theta(1)$	$\theta(\log n)$	$\theta(\log n)$	$\theta(\log n)$		$O(n\log n)$	Priority queue
Quick -sort	Worst case	Best case	Expected time running			-			
	$\theta(n^2)$	$\theta(n\log n)$	$\theta(n\log n) \le r \quad \ge O(n\log n)$						
The selection problem	Randomized select – $\theta(n)$ $Select - O(n)$		-						
Linear sort	Sort array under assumptions								
Counting sort	$\theta(n)$								
Radix sort	$\theta(n)$								
Bucket sort	$\theta(n)$								

ארבעת המיונים הנפוצים לחשיבה עליהן: א. insertion sort ב.counting sort ג.Heapsort ד.Quicksort

המיונים לחשיבה עליהן בהתפלגות תקינה: א. Bucket sort. ב. Hash(רק אם מתקיימת הנחת "suha" / שימוש בפוני אוני). ג.Skip-list

המיונים לחשיבה עליהן כשיש מספר היבטים על נתונים של כל איבר: א.B-tree ב.Avl ג.BST ד. Hash (רק אם מתקיימת הנחת "suha" / שימוש בפוני אוני).

02.06.22

Data structures - summary

	Keys Type	Expected Run-Time	Worst Case Run-Time	Extra Space	In Place	Stable
Insertion Sort	Any	-	O(n^2)	O(1)	√	√
Merge Sort	Any	-	O($nlogn$)	O(n)	x	√
Heap Sort	Any	-	O($nlogn$)	O(1)	√	x
Quick Sort	Any	O($nlogn$)	O(n^2)	O($logn$) – Expected O(n) – Worst Case Can be reduced to O($logn$)	√	x
Counting Sort	Integers [0..k] או פונקציה שממפה לקטע מסוים	-	O(n+k)	O(n+k)	x	√
Radix Sort	d digits in base b	-	O($d(n$+$k)$)	Depends on the stable sort used	Depends on the stable sort used	√
Bucket Sort	[0,1) or [a,b)	O(n)	O(n^2)	O(n)	x	תלוי במיון של כל bucket

חסם עליון $f(n) = O(g(n))$	$\exists\, c > 0$ and $n_0 > 0$ such that: $0 \le f(n) \le cg(n) \;\; \forall\, n \ge n_0$	$\lim_{n\to\infty} sup \left\|\frac{f(n)}{g(n)}\right\| < \infty$
חסם תחתון $f(n) = \Omega(g(n))$	$\exists\, c > 0$ and $n_0 > 0$ such that: $0 \le cg(n) \le f(n) \;\; \forall\, n \ge n_0$	$\lim_{n\to\infty} \inf \left\|\frac{f(n)}{g(n)}\right\| > 0$
חסם הדוק $f(n) = \Theta(g(n))$	$\exists\, c_1, c_2 > 0$ and $n_0 > 0$ such that: $0 \le c_1 g(n) \le f(n) \le c_2 g(n) \; \forall\, n \ge n_0$	$0 < \lim_{n\to\infty} \inf \left\|\frac{f(n)}{g(n)}\right\| \le$ $\lim_{n\to\infty} sup \left\|\frac{f(n)}{g(n)}\right\| < \infty$

פתרון נוסחאות נסיגה

Master Theorem

- **נתון:** $T(n) = aT(\frac{n}{b}) + f(n)$
- **רעיון (לא פורמלי ולא מדוייק):**
נשווה את $f(n)$ עם $n^{\log_b a}$ ונראה "מי יותר דומיננטי"
- **כללים:**

1. אם $f(n) = O(n^{\log_b a-\epsilon})$ אז $T(n) = \theta(n^{\log_b a})$
2. אם $f(n) = \theta(n^{\log_b a})$ אז $T(n) = \theta(n^{\log_b a} \log n)$
3. אם $f(n) = \Omega(n^{\log_b a+\epsilon})$ וגם קיים $c < 1$ כך ש $af(\frac{n}{b}) < cf(n)$
לכל n מספיק גדול אז $T(n) = \theta(f(n))$

Data structures - summary

ADT

מבני נתונים מופשט – נתאים את המבני עפ״י צורך.

BST

1. גובה של העץ – הסתכלות מלמעלה לכיוון מטה (כמו בניין שאני עליו, מסתכל למטה) , לכיוון התחתית. כלומר, גובה של קודקוד הוא המסלול המקסימלי מהקודקוד לצאצא שלו. פורמלית : גובה העץ – העלה העמוק ביותר ועץ ריק גובהו -1.
2. עומק של עץ – הסתכלות מהתחתית כלפי השורש(כמו בניין שאני בתחתית שלו, מסתכל מעלה) ; עומק של העץ מס׳ הקשתות מהקודקוד לשורש.
3. סוגי עצים : **א**. עץ בינארי רגיל – יש לכל היותר שני ילדים. **ב**. עץ בינארי מלא- יש לכל קודקוד שני ילדים. **ג**. עץ בינארי מושלם – כל העלים באותו עומק. **ד**. עץ בינארי שלם – עץ בינארי מושלם שעליו מחוקים מצידו הימני והילך.
4. שליטה והבנה בחיפוש ה-successor וה-predecessor (היכן יכול להתמקם בעץ).
5. פעולת המחיקה – הבחנה בין שלושה מקרים : **א**. עלה – הסרה רגילה(שינוי המצביע של ההורה ל-null). **ב**. יש ילד אחד – הסרה וקישור בין ההורה לנכד. **ג**. יש שני ילדים – מציאת ה- successor, החלפה בין הערכים ומחיקה (לא לשכוח ״לחבר״ את תת העץ של ה-succ לאבא המקורי).

AVL

1. מוטיבציה – שינוי זמני הריצה של BST עפ״י כמות האיברים ולא עפ״י גובה העץ. כלומר $\theta(\log n)$ ולא $\theta(h)$.
2. עץ AVL מחזיק את כל השדות בדיוק כמו בעץ בינארי(key, left, right ,p) אך מחזיק בנוסף גם שדה גובה (h).
3. ההפרש בין תת העץ הימני לשמאלי בערך מוחלט הוא לכל היותר 1.
4. הכנסת קודקוד בעץ AVL יכולה ליצור חוסר איזון בעץ ועל כן נצטרך לאזן אותו, ישנם 4 מקרים.

02.06.22

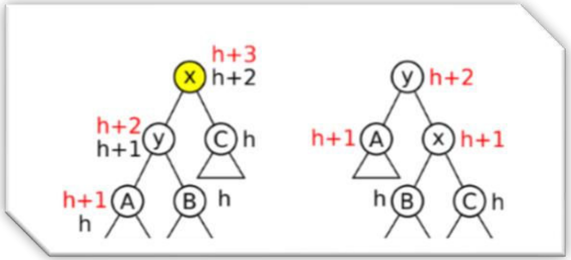
Data structures - summary

ראשית, לאחר ההכנסה נגדיר כ- x את הצומת הראשונה מן העלה כלפי מעלה בה מופר האיזון, ונתבונן בה. (מומלץ להתבונן בסרטוט תוך כדי קריאה).

a. Case Left-Left - הבן השמאלי של x (להלן יקרא t_l) גדול מן הבן הימני שלו(להלן יקרא t_r). ותת העץ השמאלי של t_l (להן יקרא t_{l_l}) גדול או שווה לתת העץ הימני שלו (להלן יקרא t_{l_r})

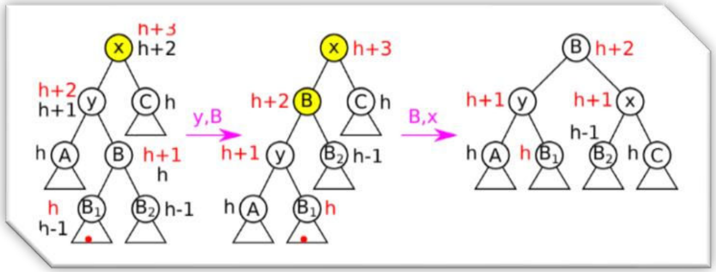
במקרה זה נבצע רוטציה ימנית – כלומר t_l יהיה השורש, תת העץ השמאלי שלו ישאר כמו שהוא , ותת העץ הימני שלו יהיה הקודקוד x עם כל תת העץ הימני בעץ המקורי.

בנוסף, תת העץ השמאלי של x יהיה הבן השמאלי של y בעץ המקורי.



b. Case Left-Right - הבן השמאלי של x (להלן יקרא t_l) גדול מן הבן הימני שלו(להלן יקרא t_r). ותת העץ הימני של t_l (להן יקרא t_{l_r}) גדול מתת העץ הימני שלו (להלן יקרא t_{l_l})

במקרה זה נבצע רוטציה כפולה – בדיוק כמו במקרה a אך עבור הצד הסימטרי (**יש להבחין כיצד לבצע את הרוטציה לפי ערכי הקודקודים גדול/קטן במקרה זה, הערכים שונים ממקרה a ולכן הרוטציה סימטרית עבור ערכים אלו)**



c. Case Right-Right - הבן הימני של x (להלן יקרא t_r) גדול מן הבן השמאלי שלו (להלן יקרא t_l). ותת העץ הימני של t_r (להלן יקרא t_{r_r}) גדול או שווה לתת העץ השמאלי שלו (להלן יקרא t_{r_l})

במקרה זה נבצע רוטציה ימנית – בדיוק כמו במקרה Left-Left.

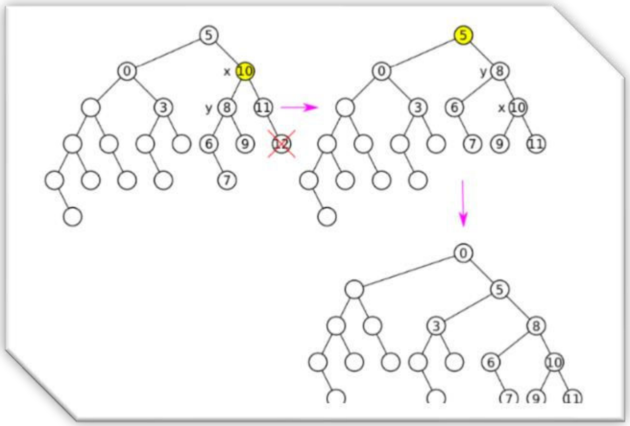
d. Case Right-Left - הבן הימני של x (להלן יקרא t_r) גדול מן הבן השמאלי שלו (להלן יקרא t_l). ותת העץ השמאלי של t_r (להלן יקרא t_{r_l}) גדול מתת העץ הימני שלו (להלן יקרא t_{l_l})

במקרה זה נבצע רוטציה כפולה – בדיוק כמו במקרה Left-Right.

5. בתהליך המחיקה, בדומה להכנסה יכול להתבצע מצב של חוסר איזון. גם במקרה זה נצטרך לבצע רוטציות. נתבונן בשני מקרים:

a. Case Left-Left - בדיוק כמו במקרה לעיל של ההכנסה. נבצע <u>רוטציה ימנית.</u>
במקרה זה, יכול להתבצע מצב שניצור איזון בתת העץ אך נגרום לאבא של תת העץ, עצם הפעולה, לחוסר איזון. במקרה זה נצטרך לבצע עוד רוטציה ואולי עוד עד כדי איזון סופי.

b. Case Left-Right - בדיוק כמו במקרה לעיל של ההכנסה. נבצע <u>רוטציה כפולה.</u>
במקרה זה, יכול להתבצע מצב שניצור איזון בתת העץ אך נגרום לאבא של תת העץ, עצם הפעולה, לחוסר איזון. במקרה זה נצטרך לבצע עוד רוטציה ואולי עוד עד כדי איזון סופי.



Data structures - summary

B-tree

B-tree – מוטיבציה למבני נתונים זה הוא הרצון לחסוך כמה שיותר קריאות מן הדיסק ע״י כך שנשמור יותר קודקודים ב- Blocks ולא נצטרך כל צעד לגשת ל-Ram.

<u>תיאור המבנה:</u> בכל קודקוד יהיו מספר שדות – נציין 3 שדות מרכזיים

א. שדה (int) השומר את מספר המפתחות בתוך הקודקוד x.

ב. שדה (int []) השומר מערך של pointers ל- satellite data של כל מפתח.

ג.שדה (int []) השומר מערך של pointers ל- children's של הקודקוד.

<u>מאפיינים:</u>

א. לכל קודקוד פנימי יש ״מספר המפתחות + 1 ״ ילדים.

ב. בכל קודקוד המערך המאחסן את המפתחות ממוין.

ג. לכל העלים בעץ יש עומק שווה.

ד. פרמטר t - נקרא הדרגה המינימלית אשר קובע את החסם העליון והתחתון של המפתחות שמאוחסנים בעץ, נבחין כי ב-

Root – החסם התחתון הוא לכל הפחות קודקוד 1, וחסם עליון $2t-1$.

Node - יש בין $t-1$ מפתחות, חסם תחתון, ל $2t-1$ מפתחות, חסם עליון.

הכנסה - נבצע את פעולת ההכנסה בדיוק כמו בעץ חיפוש.

המקרה הקל הוא כאשר העלה מכיל פחות מ- $2t-1$ קודקודים ולכן יש מקום לאובייקט להיכנס.

המקרה המורכב יותר כאשר האובייקט מגיע לעלה בו יש $2t-1$ קודקודים וזהו המקסימום, לכן נבצע הכנסה של הקודקוד ונעלה את קודקוד החציון לאבא שלו.

לאחר מכן, נחלק את הקודקודים לשני קבוצות בגודל $t-1$ כל אחד.

Data structures - summary

אחד מצידו הימני של החציון ואחד מצידו השמאלי.

בעיה נוספת שעלולה לקרות זה כאשר נבצע את האלגוריתם המתואר לעיל בו החציון עולה לקבוצת הקודקודים של האבא שלו – אם גם שם יש $2t-1$ נצטרך גם אותם לפצל.

וככה נוכל להמשיך עד שנגיע לשורש.

לכן, על מנת לפתור בעיה זו – כאשר נבצע הכנסה, בכל צומת בה נעבור ויהיו $2t-1$ קודקודים נפצל אותה טרם ההכנסה, מה שימנע את המצב בו נצטרך לטפס למעלה חזרה בעץ לטובת פיצול. דרך זו נקראת pass-1.

אפשרות נוספת נקראת pass - 2 בה אנו מבצעים פיצול רק בצומת המלאה הכי קרובה לקודקוד בו המפתח נכנס(כלומר, בעת ההכנסה נבצע שמירה של הקודקוד המלא האחרון אותו עברנו. כאשר נגיע לצומת בה נכניס, נחזור ונפצל את הקודקוד האחרון שנשמר).

מחיקה- נבצע פעולת מחיקה בדיוק כמו בעץ חיפוש.

המקרה הפשוט יותר הוא כאשר העלה מכיל יותר מ- $t-1$ קודקודים ולכן לאחר המחיקה העלה עדיין נשאר בטווח חוקי.

המקרה המורכב יותר הוא כאשר ישנם $t-1$ קודקודים בדיוק ולאחר המחיקה נוכל להגיע לטווח לא חוקי.

לכן טרם המחיקה נבצע את אחת משתי הפעולות הבאות - shifting or merging.

בפעולת ה- shifting אנו לוקחים איבר מן תת העץ הימני או השמאלי (משמאל ואז מימין, ורק במידה והוא מכיל לפחות t קודקודים) מעבירים אותו לאבא , והאבא מעביר את "האיבר המפריד" לתת העץ בו הולך להימחק הקודקוד.

במידה והפעולה הבאה לא מתאפשרת כיוון שיש לשני תתי העצים יש t-1 קודקודים אנו נבצע merging ונאחד את שני תתי העצים עם "האיבר המפריד".

ולכן, נקבל בסה"כ 2t-1 איברים.

נשים לב שיש הבדל בין פעולת המחיקה בין עלה לקודקוד פנימי.

בעלה אנו פשוט נסיר ואז נמשיך עפ"י האלגוריתם שתואר לעיל.

בקודקוד פנימי , אנו קודם נמחק את ה- succ נמשיך עפ"י האלגוריתם לעיל ואז נחליף אותו עם האיבר שהיה צריך להימחק.

באותו אופן, כמו שביצענו pass-1 בהכנסה נבצע כך גם במחיקה באופן סימטרי.

Data structures - summary

Probability

מרחב המנה - Ω זו קבוצה המכילה את סך כל האפשרויות עבור התרחשויות המקרה.

P(x) – נקראת פונקציית ההסתברות ; פונקציה ממרחב המנה Ω לטווח [0,1].

עבור מאורע $A \subseteq \Omega$ אז Pr[A] = סכום ההסתברויות של כל המאורעות P(x) כך ש-$x \in A$.

כלומר, $\mathbf{Pr[A] = \sum P(x) | x \in A}$

תוחלת הינו סכום הערכים הצפוי. כאשר שואלים אותנו מהו הערך הצפוי, נחשוב על תוחלת המקרה. ומכך נגדיר :

פונקציה רנדומלית - $X: \Omega \to R$

תוחלת עבור משתנה רנדומלי X - $\mathbf{E[X] = \sum k * Pr\,[X = k]}$ | $\mathbf{k} \in \Omega$

נשים לב, כי עבור אינדיקטור מקרי ערך ה-k הינו 0 או 1 ולכן התוחלת עבור אירוע שווה להסתברות של אותו מקרה.

כלומר , $\mathbf{E[X]} = \ldots\ldots = \mathbf{Pr\,[A]}$

הערה, לינאריות התוחלת - נשים לב כי סכום התוחלת שווה לתוחלת הסכום.

לסיכום הנושא, נשים לב שישנם שני התפלגויות – בינומית וגאומטרית.

התפלגות בינומית - זו התפלגות שמקבילה לשאלה עבור n זריקות והסתברות p מה הסיכוי שאקבל בסה״כ כל הזריקות ״head״?

Ω = all H\T strings of length n.

$P(y) = p^k * (1-p)^{n-k}$ where "k" in the number of "H" in y.

$X(y)$ = number of "H״s in y.

$E[x] = np$

Data structures - summary

התפלגות גיאומטרית – זו התפלגות המקבילה לשאלה מהו מספר הזריקות שאצטרך לזרוק עד שאקבל בפעם הראשונה "head" ?

לדוג' -

$$\Omega = \{H, TH, TTH, \dots\}$$

$$P(H) = \frac{1}{2}\,; P(TH) = \frac{1}{4}; P(TTH) = \frac{1}{8}\ \dots$$

$$X(y) = \text{length of } y.$$

$$E[X] = 2$$

Skip-list

מבני נתונים זה הינו ADT של dynamic set כך שלכל איבר יש key וגם satellite-data.

מבני זה מיוצג כרשימה מקושרת דו כיוונית ממוינת, ו"כבניין" לגובה כאשר לכל איבר יש הסתברות $\frac{1}{2}$ שגם ב"קומה" הבאה יהיה מקושר לאיבר אחר.

כל קודקוד יכיל את השדות הבאים : key, height, x.next, x.prev

גובה ה-sentinel הוא כגובהו של הקודקוד בעל "הקומה" המקסימלית – וזה מוגדר להיות גובהו של ה skip -list.

אלגוריתם החיפוש, הכנסה, ומחיקה דיי פשוטים ומופיעים במצגת (רק נדייק שהחיפוש מתחיל מ"הקומה" הגבוהה ביותר לכיוון מטה, וההכנסה ומחיקה הפוך).

כעת, נבצע הסבר על תוחלת המבני – גובהו וגודלו ונציין מספר טענות לגבי מבני- נתונים זה.

<u>טענה</u> : מספר הקודקודים במבני זה , לא כולל ה- sentinels הינו 2n.

הסבר : הסתברות של איבר להגיע ל- S_i הוא (התפלגות בינומית) $\frac{1}{2_i}$.

Data structures - summary

לכן התוחלת כפי שרשמנו לעיל $E[x] = np$ במקרה הנ״ל הינה $E[|S_i|] = \frac{n}{2_i}$.

לכן, סכום התוחלת ב- $\forall i$ S_i הינו $\sum E[|S_i|] = \sum \frac{n}{2_i} = n \sum \frac{1}{2_i} = 2n$

<u>טענה</u> : גובה העץ ב- list-skip הינו לכל היותר $\log n + 2$

הסבר : ההוכחה מעט ארוכה לפירוט, ניגע בקצרה בדרך ההוכחה.
נגדיר משתנה X_i אשר יהיה אינדיקטור רנדומלי – כלומר, 0 עבור S_i ריק ו-1 עבור S_i שאינו ריק.
לכן גובה המבני הינו $h = \sum X_i$ (הגובה – סכום כל ערכי ה-1) כיוון שאין אנו יודעים במדויק את h נשתמש בתוחלת.
נשים לב כי החסם עליון של התוחלת h הינו התוחלת של S_i שחישבנו לעיל.
נבצע פיתוח של תוחלת h , תוך הפרדת הסכום בנקודה $\log n$ ששם יש שינוי בערך החישוב, נבצע החלפה בחסם העליון ונחשב זאת.

<u>טענה</u> : מספר הקודקודים הכולל במבני skip-list הינו $2n + O(\log n)$.

הסבר : חישבנו לעיל שמספר הקודקודים ללא sentinels הינו 2n.
בנוסף, גובה ה skip-list המצופה הינו $\log n + 2$
לכן, בחישוב הסכום נקבל כי מספר הקודקודים הכולל הינו $2n + O(\log n)$

<u>טענה</u> : זמן הריצה עבור חיפוש/הכנסה הינו $2 \log n + 5$

הסבר : דרוש הרחבה

<u>טענה</u> : סיבוכיות זמן הריצה

הסבר : עבור מחיקה, הזמן המשוער הינו $\theta(1)$ כיוון שהגובה המצופה של קודקוד הוא 1 (כהתפלגות גיאומטרית) ולכן זהו גם סך השינויים שנצטרך לבצע.

Data structures - summary

Hash tables

מימוש מבני נתונים של מילון המאחסן קבוצה של אלמנטים (במערך) אשר לכל אחד יש key וpointer לdata.
פתרון נאיבי הוא להשתמש במבני נתונים זה בצורה כזו אשר יאחסן את הערך ה-k במקום ה- [k] במערך. לכן עבור קלטים גדולים נוכל לקבל טבלאות גיבוב מאוד גדולות.
הפתרון לכך הוא לבצע mapping לכל הערכים לפני הכניסה לטבלת גיבוב בתוך הטווח אותו אנו מגדירים, מה שיגרום להקטנת האחסון שנצטרך עבור ערכים גדולים.

המיפוי הזה נעשה באמצעות hash function.
ישנן מספר שיטות להתמודד עם התנגשות בין ערכים הנכנסים לאותו תא :

א. Chaining
ב. Open addressing

Chaining – בשיטה הזו אנו נבצע מיפוי(mapping) לכל ערך ונכנסו למיקום בטבלה עפ״י ערך המיפוי. עבור ערכים זהים ניצור link-list (רשימה מקושרת) שתחזיק באיבר הראשון ברשימה את האיבר האחרון שנכנס.

ניתוח זמני הריצה עפ״י שיטה זו :

The worst case:

Search – $\theta(n)$

Insert – $\theta(1)$

Delete – $\theta(n)$ (Can reduce to $\theta(1)$ by using doubly linked list)

Data structures - summary

חישוב זמני הריצה עפ״י ה-worst case אינה הדרך הנכונה. לכן נעזר בהנחת ״הסואה״.

הנחת ה״סואה״ – (simple of uniform hashing) - לכל אובייקט יש סיכוי שווה להיכנס לתוך ה-slot, ללא תלות לשאר האיברים הקודמים.

הטענה נכונה כאשר המפתחות נבחרים רנדומלי ואז עבור חיפוש בטבלת הגיבוב זמן הריצה יהיה $\theta(1 + E[x])$

נסביר, $\theta(1)$ עבור ביצוע mapping והגעה למיקום במערך.

והתוחלת של x מושפעת מההסתברות של כל ערך להיכנס לטבלה בהסתברות $\frac{1}{m}$ עבור n פרמטרים ולכן נקבל $E[x] = n * \frac{1}{m}$

ולכן הזמן הצפוי הינו $\theta(1 + \alpha)$ כך ש- $\alpha = \left(\frac{n}{m}\right)$ נקרא מקדם העומס .

נרצה לשמור על מקדם העומס, α, בטווח של $\alpha \in [\frac{1}{4}, 1]$, כך נוכל לדאוג שזמן החיפוש יהיה $\theta(1)$ - זאת נעשה ע״י שימוש Rehashing.

פעולת ה-Rehashing עולה - $\theta(n + m) = \theta(n) \quad (\text{Assuming } n \in \Omega(m))$

<u>טענה</u>: הגודל המקסימלי של ה– bin לכל היותר $\theta(\log n / \log \log n)$ עם הסתברות של $1 - \frac{1}{n^{\theta(1)}}$ (לא הוכח בכיתה).

הערה חשובה: טרם הגעה לשאלה, אנו צריכים לבדוק האם מתקיימת הנחת "suha" כלומר, בהינתן טבלה בגודל m האם הסיכוי של כל קודקוד להיכנס לכל תא בטבלה זהה. כלומר $\frac{1}{m}$. במידה וכן, יש לציין זאת בתשובה לשאלה.

במידה ולא, הפתרון האלטרנטיבי הוא להשתמש בפונקציית גיבוב אוניברסלית מה שיאפשר לנו להשתמש להסתמך על הזמני ריצה אותם ניתחנו בכיתה.

<u>טענה</u>: יהי k איבר בעולם U. במימוש פונקציית גיבוב בשיטת chaining האורך המצופה המקסימלי של איבר במערך (כלומר לאחר המיפוי של k, במקום הנוכחי בטבלה) הוא $at\ most\ 1 + \frac{n}{m}$.

הסבר: הוכחה מלאה במצגת עמוד (19). פרטי ההוכחה - הגדרת משתנה אינדיקטור 1 עבור כך שהמיפוי שלו מגיע לקודקוד k , ו-0 לא.

Data structures - summary

נבצע חישוב של סכום התוחלות. חלוקה למקרים בין $k \notin S$ ואז התוחלת של X הינה ההסתברות של כל איבר כפול מספר האיברים. כלומר, $\frac{1}{m} * n$

ועבור מקרה ב׳ בו $k \in S$ ואז סכום התוחלת של X הינה 1 (עבור ההסתברות ש-k נמצא ב-S) וחישוב עבור שאר האיברים (n-1) כפול ההסתברות של כל איבר.

כלומר, $1 + (n-1) * \frac{1}{m}$.

כעת, נתבונן בשיטה ב׳ – Open addressing.

ל- Open addressing יש שתי שיטות למימוש : Linear probing או double hashing.

ב- Linear probing אנו מניחים כי אין satellite למפתחות אלא הערך עצמו מאוחסן במערך.

בשיטה זו אנו מכניסים את האיבר ע״י הmapping ובמידה ואין מקום אנו עוברים לאינדקס העוקב בטבלה עד שנמצא מקום פנוי.

שימוש בשיטה זו משפיעה עלינו בתהליך החיפוש והמחיקה.

בתהליך החיפוש אנו צריכים לבצע חיפוש לא רק במיקום של האיבר בטבלה אלא עד המקום הבא בטבלה שיהיה null (כיוון שישנה אפשרות שהאיבר נמצא לא במקומו עקב זאת שלא היה מקום בטבלה).

בתהליך המחיקה ישנן שתי אפשרויות :

א. לשים ערך מיוחד 'D' אשר יסמן כי הערך במיקום זה נמחק ובעת חיפוש נמשיך ונעבור אותו ״כאילו יש שם ערך״.
ב. מחיקת האיבר וכל האיברים אחריו עד הגעה למקום null , ואז לבצע reset לכל האיברים שנמחקו (פרט לערך שרצינו למחוק כמובן) ולמקם אותה בטבלה.
שימוש באפשרות זו מאפשר פינוי תאי זבל בטבלה אשר משפיע בקשר ישיר לחיפוש מהיר יותר ופחות ביצוע של rehashing.

כעת נתבונן בשיטה השנייה למימוש. בשיטה זו משתמשים בdouble hashing כלומר בשתי פונקציות hash.

בצורה פורמלית $h(k,i) = \left(h_1(k) + i * h_2(k)\right) mod\ m\ ;\ \ 0 \le i \le m-1$

ראשית משתמשים בפונקציית המיפוי הראשונה(בדיוק כמו בשיטה א׳ - h_1) במידה ויש מקום האיבר נכנס לערך המתאים במערך.

Data structures - summary

במידה ואין מקום, מבצעים מיפוי גם בפונקציה השנייה(h_2) (עבור ערך i=1, קונספטואלית עבור השורה לעיל ה- i=0) – במידה ויש מקום נכניס את הערך, במידה ואין נבצע בשנית מיפוי בפונקציה השנייה(h_2) אך עבור ערך i+1 וכן הלאה עד שיהיה מקום.

<u>הערה חשובה:</u> יש לשים לב שאין אפשרות לבצע החלפה בין h_1 לבין h_2 . כלומר שזו תהיה פונקציית הגיבוב וזו תהיה פונקציית הצעד. (הסבר מרחיב מצגת תרגול 7 שקופית 67)

בנוסף יש חשיבות ש- $\gcd(h_{step}(k), m) = 1$ לכל מפתח.(הסבר מרחיב במצגת לעיל).

הבנה חשובה. יש לחזור על זה.

Bloom filter

מבני נתונים זה נפוץ בשימוש לשמירת אתרים זדוניים. זהו אינו מבני נתונים דטרמיניסטי (נרחיב בהמשך).

ישנם מספר גדול של אתרים אותם היינו רוצים לשמור על מנת להזהיר את ה- client טרם הכניסה לאתר. אחסון של מספר אתרים גדול דורש זיכרון גדול.
הפתרון לכך הוא שימוש במבני נתונים זה. בשלב הראשון הוא יצירת מערך המאותחל לערכי 0 בכולו. בשלב הבא לכל כתובת url של אתר זדוני מבוצע mapping אשר משנה את ערך הביט מ-0 ל-1.
על כן, בגלישת לקוח לאתר מבוצע mapping לכתובת ה-url ובמידה ונמצא הביט 1 מוחזר לו הערך 'סכנה'.
על מנת להגביר את הדיוק משתמשים במספר פונקציות hash ווידוי של מספר ביטים (בהתאם למספר פונקציות ה-hash) אשר מגביר את הדיוק.
ישנו חסם בין מספר הפונקציות hash לדיוק – כיוון שכל שנגדיל את מספר הפונקציות יצטרכו יותר ביטים לאמת את סיכון האתר(מה שמקטין את השגיאה) אך מספר הביטים שיהפוך ל-1 במערך יגדל ועל כן גם השגיאה תגדל.
שימוש במבני נתונים זה, מאפשר שנחזיר ללקוח שגיאה עבור אתר תקין כלא תקין (false positive), אך לא ההפך ועל כן נקרא מבני דטרמיניסטי, החלטי.

<u>ניתוח ההסתברויות:</u>

שימוש בפונקציית hash אחת - ההסתברות שהביט יהיה 0 הינו $p = \left(1 - \frac{1}{m}\right)^n$; כאשר m זה גודל הטבלה ו-n מספר האיברים.
לכן, ההסתברות שהערך יופיע להיות 1 הינו $1 - p$.

Data structures - summary

עבור שימוש במספר פונקציית hash - שהביט יהיה 0 הינו $p = \left(1 - \frac{1}{m}\right)^{nt}$; כאשר m זה גודל הטבלה, t מספר הפונקציות ו-n מספר האיברים.
לכן, ההסתברות שהערך יופיע להיות 1 הינו $(1 - p)^t$.

הסבר על חישוב החסם מופיע במצגת בעמוד האחרון. יש לעבור על זה.

Heap

מבני נתונים זה מממש תור עדיפות אשר מאחסן קבוצה של איברים בעלי מפתחות, ותומך ב-3 השיטות הבאות:
Maximum – מחזיר את הערך המקסימלי (בדומה לערך peek).
Insert -מכניס את הערך הנכנס.
ExtractMax – מסיר ומחזיר את האיבר עם הערך הגדול ביותר.

מבני נתונים זה משמר שני דברים: א. בדומה לעץ בינארי שלם (עץ מלא שכל עליו באותו גובה , ורק עליו בצידו הימני חסרים).
ב. יחס המקסימליות/מינימליות בין האב לילדיו.
ניתן לממש מבני זה במערך אשר האיברים יאוחסנו עפ"י הlevels בעץ הבינארי השלם ויאפשר לנו ע"י שימוש בנוסחה גישה מהירה לאבא של הקודקוד ולבניו (הנוסחה מוצגת בפונקציית Navigation).

ישנם שני סוגי ערמות – ערמת מינימום וערמת מקסימום. העיקרון זהה לשתי הערימות – מיהו האיבר שנמצא כקודקוד השורש.
האבא בכל שלב תמיד גדול(בערמת מקסימום) משני בניו וכן הלאה. יחס המקסימליות.

כעת, ניגע בשיטות ובפונקציות של מבני נתונים זה.

Data structures - summary

Navigation – מכיוון שהמערך מסודר עפ״י רמות העץ בינארי ניתן להגיע להורה של כל קודקוד, לבן הימני ולבן השמאלי(במידה וקיים כמובן).

Parent - (ערך תחתון) $\frac{i-1}{2}$

Left - $2i+1$

Right - $2i+2$

MaxHeapify – פונקציה זו מקבלת index "i" ובודקת עבור תת העץ של קודקוד זה כי הוא מממש את עיקרון הערימה.
כלומר, שני הבנים קטנים מן האבא – במידה ולא, מחליף את האבא עם הילד בעל הערך המקסימלי. $\theta(\log n)$

BuildMaxHeap - פונקציה זו אחראית על בניית הערימה. היא קוראת לפונקציה MaxHeapify מקודקודי ההורים של העלים והילך (כלומר מ- $\frac{A.length}{2}$ down to 0) ובכל קריאה עוברת מהחלק התחתון של השורש כלפי מעלה בבדיקה שכל תת עץ עומד בתנאי הערימה.
בסוף תהליך זה הערימה מסודרת.

ניתוח זמן הריצה – BuildMaxHeap מבצעת לולאה $\frac{n}{2}$ פעמים. בכל איטרציה קוראת לפונקציה MaxHeapify שעלותה הוא $O(\log n)$ לכן סה״כ עלות בניית הינו $O(n \log n)$. <u>שים לב – זהו חישוב ראשוני, חישוב מדויק מתואר בהמשך!</u>

בניתוח זמן מדויק יותר נשים לב כי בקריאות האיטרציות הראשונות פעולות ה- MaxHeapify מבצעות פעולות ב-$\theta(1)$ לכן בחישוב מדויק זה – זמן הריצה של בניית הערימה הינו $\theta(n)$. (חישוב זמן ריצה זה מתבסס על כך שמספר הקודקודים בגובה h הוא לכל היותר $\frac{n}{2^{h+1}}$ ערך עליון ; חישוב מלא במצגת)

Maximum – החזרת קודקוד העץ. זמן הריצה הינו $\theta(1)$.

Data structures - summary

ExtractMax – פעולה זו מבצעת הוצאה ומחיקת ה-Max מן העץ. הפונקציה מחליפה את קודקוד השורש עם האיבר האחרון במערך(העלה הימני ביותר) ואז מבצעת קריאה לפונקציה MaxHeapify אשר מבצעת סידור לתת העץ הקיים, כלומר לכלל העץ. $\theta(\log n)$

Increaekey – מקבלת כערך את מיקום הקודקוד במערך ואת והערך לשינוי.
הפונקציה מחליפה את הערך ומבצעת לולאה של בדיקות האם הקודקוד קטן מן האבא – כשזה קורה, מתבצעת עצירה בלולאה.
זמן הריצה הינו $\theta(\log n)$.

Insertion – הכנסת ערך חדש. תיאור - הכנסת ערך "אינסוף" כביכול לסוף המערך, ואז ביצוע של Increasekey עבור הקודקוד "אינסוף" לשינוי הערך שקיבלנו להכנסה.
תזכורת , הקריאה לפונקציה Increasekey בסוף השינוי מבצעת גם בדיקה כי הערימה חזרה להיות כנדרש.
זמן ריצה זה הינו $\theta(\log n)$.

Heapsort – פונקציה זה ממיינת את המערך למערך ממוין.
פונקציה זו רצה מן האיבר האחרון עד הראשון ובכל קריאה מחלצת את המקסימום ע"י קריאה לפונקציה ExtractMax ושומרת את הערך במקום ה-i.
בצורה זו כל פעם אנו מוציאים את המקסימום מהערמה ושומרים אותו מהסוף לתחילת המערך.
זמן הריצה הינו $\theta(n \log n)$ הסבר - סה"כ $n-1$ איטרציות ועבור כל איטרציה קריאה לExctractMax שעלותה $\theta(\log n)$

Compression algorithms

מוטיבציית הבעיה – במידה ויש לנו טקסט המכיל מספר רב של מילים, כמה ביטים/מקום נצטרך כדי לקודד אותו?
כדי לפתור בעיה זו אחת מן השיטות היא שימוש בקוד תחיליות. קוד תחיליות הינו קוד אשר מקודד כל תו ומבטיח כי התחילית של כל תו שונה אחת מן של השנייה מה שמונע בעיות בdecoding.
העלות של קוד זה הוא החישוב מספר המילים כפול מספר הביטים שהוקצו לכל אות. (ללא חישוב הקידוד).

Data structures - summary

קוד תחיליות אופטימלי זהו קוד שמאפשר עלות מינימלית.
דרך ייצוג של קוד תחיליות – העלים מייצגים את האותיות, אות לכל עלה. והמסלול מן השורש לעלה מיוצג ע״י ביט 0 או 1 (בהתאם לפנייה שמאלה או ימינה).
הערך, המספר של כל עלה מייצג את סך כמות האותיות שנמצאים בתת העץ שלו.

מימוש אפשרי לקוד תחיליות הוא קוד האופמן.

תיאור האלגוריתם :

בשלב הראשון אנו מבצעים ספירה של כלל האותיות והתדירויות שלהם בטקסט.
לבסוף, יוצרים קודקודים אשר כל קודקוד מכיל את האות (למשל ״a״) ואת התדירות שלה בטקסט(למשל ״1475״).

בשלב השני, אנו ניקח כל פעם את שני הקודקודים/ תת העץ המינימלי ונחבר אותם. כמובן שלא נשכח לעדכן את השורש בסכום הערכים.
בצורה זו נמשיך עד אשר נגיע לעץ שלם, כלומר לעלה אחד. כך שלא קיימים שני עצים נפרדים.
בנייה של קוד תחיליות זה נוכל לעשות ע״י שימוש בערמה (heap) הוצאת המינימום פעמיים, סכימה והכנסה.
בצורה זו, אנו מייצגים כל אות במספר ביטים כך שכל שרמת השכיחות גבוהה יותר כך מספר הביטים המייצגים אותה נמוך יותר וזהו הקידוד עבור הטקסט.

- הערה, לא נרחיב בסיכום זה אך יש לשים לב לגבי הלמה – קיים קוד תחיליות אופטימליות בו שתי האותיות הנפוצות ביותר הן אחיות בעץ.
  הוכחה והסבר מצגת 11 עמוד 17.

דרך נוספת של קידוד טקסט נקרא Lempel-Ziv. בשונה מדרך הקידוד הרגילה, קידוד זה עובד על זיהוי חזרות של בלוקים בטקסט.
אלגוריתם זה עובר בצורה שיטתית מתחילת הטקסט ועד סופו שכל בלוק מכיל את הקוד התחיליות המקסימלי שהיה עד כה בטקסט ועוד ערך אחד.

הרחבת ההסבר : נניח שיש לנו מילה המורכבת מ-10 אותיות. אנו מתחילים מ- בלוק ריק ולכן האיבר הראשון יהיה הבלוק הראשון עם איבר אחד.
כעת נסתכל על האיבר השני – אם הוא מכיל את אותה אות כמו האות הקודמת עפ״י התיאור לעיל נוסיף עוד אות אחת (כלומר בלוק המכיל את האות הראשונה והבלוק הקודם שהיה) ואלו יישמרו כבלוק מספר 2.

Data structures - summary

במידה ולא, וזה אות חדשה, ניצור בלוק חדש ונוסיף בלוק זה לבלוקים הקיימים.
החלוקה של העץ לבלוקים מכונה trie.

בסיום חלוקת האותיות לבלוקים נמחק את העץ וכל בלוק נקודד לזוג מספרים (i,j) כך ש-i זהו האבא של הבלוק (כלומר מה ערך הסטרינג אותו אנו מעתיקים מן האבא) ו-c זו האות שמצטרפת לסטרינג של האבא.

אנו תיארנו לעיל עבור עץ, אך ישנן מספר אפשרויות לאחסון הבלוקים :
דרך אחת זה שכל קודקוד מיוצג ע״י חוליה של רשימה מקושרת (linked list) ומצביעים לילדים(כל ילד מיוצג ע״י חוליה ומצביעים לילדים וכן הלאה). זמן הריצה הינו $\theta(\propto n)$ כך ש-$\propto$ גודל האלפבית ו-n זהו אורך הסטרינג.
דרך נוספת, כל קודקוד מיוצג ע״י טבלת hash והמצביעים לילדים מאוחסנים בתוך כל טבלה. זמן הריצה המצופה הינו $\theta(n)$
עוד דרך לאחסון, שימוש בטבלת hash גלובלית אשר תאחסן זוג ((j,c),i) – בו j בלוק האבא ו-c התו האחרון ב-i.

קידוד- דרוש הסבר

Data structures - summary

Quicksort

בדומה ל-sort merge שהשתמשה ב- Divide & Conquer כלומר חלוקה לתתי מערך , מיון ברקורסיה ומיזוג, גם מבני נתונים זה משתמש בזה.

מבני נתונים זה מבצע חלוקה של המערך ע״י שימוש בפונקציה Partition – לוקח איבר במערך, נסמנו q, עושה חלוקה לשני תתי מערך, הגדולים ממנו וקטנים ממנו והאיבר q מאופסן ב- A[q] ונקרא פיבה.
בסוף תהליך זה האיבר במקום ה- A[q] ממוקם באינדקס הנכון. נבצע זאת גם על תתי המערכים(הקטנים והגדולים ממנו) ולבסוף נקבל מערך ממוין.

זמן הריצה – עבור <u>המקרה הגרוע ביותר</u> זמן הריצה הינו $\theta(n^2)$. מקרה זה מתרחש כאשר המערך ממוין ולכן עבור כל קריאה, נסמנה בגדול המערך n, המערכים הבאים שנקבל הינם 0 ו- n-1. (כלומר הקריאות הינם n ואז n-1 ואז n-2 וכו׳ ...)
קריאה לפונקציה Partition עולה $\theta(n)$ סה״כ יש לבצע n קריאות ולכן מקבלים $\theta(n^2)$.

<u>המקרה הטוב ביותר</u> הינו כך שכל קריאה מחלקת את שני תתי המערך לגדלים שווים. (עבור קריאה בעלת מערך בגודל n , נקבל שני מערכים בגדלים $\frac{n}{2}$)
במקרה זה, גודל המערך קטן בחצי כל קריאה ולכן נדרשות רק $\log_2 n$ קריאות.
כל קריאה לפונקציה Partition זה $\theta(n)$ ולכן סה״כ נקבל זמן ריצה $\theta(\mathrm{n} \log_2 n)$

<u>המקרה המצופה</u> – חלוקה לקודקודים אדומים וירוקים (חישוב מלא במצגת) סה״כ זמן הריצה הינו $\theta(n \log n)$ דרוש הרחבה

נשים לב, שבמהלך כל החישוב הנחנו כי המערך נבחר באופן אקראי ועל כן גם המפתחות.
בהינתן מערך שאינו נבחר באופן אקראי, נגריל איבר אקראי מן המערך, הוא ישמש כפיבה, ולכן נחליף אותו עם האיבר האחרון ואז נבצע את ה-partition.

Data structures - summary

The selection sort

בבעיית הבחירה אנו מקבלים מערך A בגודל n ועם אינדקס i והמטרה להחזיר את האיבר במיקום ה-i הקטן ביותר במערך.

פונקציה ראשונה נקראת Randomized – select פונקציה זו מגרילה פיבה, (ומכניסה את הפיבה לאינדקס הנכון במערך, נסמן מיקום זה ב-k).
לאחר מכן, בודקת האם מספר האיברים הקטנים ממנו שווה לאינדקס i (אם כן, מחזירה את הערך במקום ה- A[k] (במיקום הפיבה) במידה וקטן מבצעת קריאה ברקורסיה בשנית עבור תת המערך השמאלי ובצורה סימטרית אם זה גדול עבור תת המערך הימני.
הזמן הצפוי לפונקציה זו הינו $\theta(n)$.

הפונקציה השנייה הינה פונקציית ה- *select* , פונקציה זו בדומה לפונקציה הראשונה אך ההבדל היחיד הוא השימוש בפונקציית העזר *ChoosePivot* הבוחר פיבה בצורה נכונה.
פונקציית העזר *Choosepivot* מחלקת את המערך באופן שלא עולה על 3:7 וזאת ע״י חלוקת המערך לקבוצות בעלות חמישה איברים ומציאת החציון עבור כל קבוצה.
לאחר מציאת החציונים, הכנסת כלל החציונים למערך ממוין והוצאת החציון. דרוש הרחבה
סה״כ פעולה זו לוקחת $\theta(n)$.

Sorting algorithm

כעת נדבר על שלושה סוגי מיונים ליניאריים בתנאים מוגדרים עבור הקלט.
לפני כן, נגדיר שלושה מושגים:

א. *Comparison based* – מיון הנעשה ע״י ביצוע השוואה.
ב. *In place* – שמירת מקום נוסף עבור איברים, מערכים וכו׳.
ג. *Stable* – שמירה על הסדר היחסי עבור מפתחות עם ערכים זהים במהלך המיון.

Data structures - summary

ניתן לייצג מיון הנעשה ע״י ביצוע השוואות כעץ החלטות. (למשל עבור כל בדיקה האם $a_1 > a_2$ נבצע ירידה ימינה, וההפך ירידה שמאלה).

גובה העץ הינו החסם התחתון של סיבוכיות האלגוריתם(כי בעץ נעשות רק השוואות אבל יש עוד פעולות רקע שנצטרך לעשות).

חסם זה הינו $\Omega(n \log n)$.

הוכחה מלאה במצגת. נקודות ההוכחה- מספר העלים הינו $n!$, מספר הפרמוטציות לגדלי האיברים. בנוסף אנו יודעים כי מספר העלים תלוי ב-h.

כלומר, $2^h > number_{leaves} = n!$. נבודד את h ונקבל כי $h > \Omega(n \log n)$.

(מוכח במצגת גם כי מספר העלים בעץ מגובה h הוא לכל היותר 2^h).

כעת, נדבר על שלושה סוגי מיון – i. Counting sort ii. Radix sort iii. Bucket sort

נשים לב, כי אלגוריתמי מיון אלה אינם מבוססי השוואה.

<u>i. Counting sort</u>

ההנחה עבור אלגוריתם מיון זה כי הקלט שנקבל, מערך A, נמצא בטווח בין $\{0...k\}$.

שלב א׳ – נבנה מערך עזר B ונאתחל את כל השדות ב-0.

שלב ב – נעבור על מערך A , ובכל תא שנעבור נוסיף למערך עזר B +1 באותו אינדקס של הערך. (למשל כאשר נבקר במערך A בתא 8 והוא יכיל את הערך 2 – נוסיף ל-B באינדקס 2, +1).

שלב ג- נעבור על מערך עזר B מאינדקס 1 ונשנה כל ערך בתא כסכום האיבר לפניו וערכו הנוכחי.(כלומר ע״י כך נוכל להחזיק באינדקס של המיקום הנכון של האיבר בסידור המערך A).

שלב ד – השלב האחרון, נעבור שוב על מערך A מהסוף להתחלה ובכל תא שנעבור נסתכל על מערך העזר B באינדקס של הערך שקיבלנו מ-A נמקם את האיבר בערך של המקום ב-B ונחסיר ממנו 1.

נשים לב, כי מיון זה הינו מיון יציב – כיוון שעבור ערכי k זהים, המיקום שלהם יהיה באותו סדר במערך(הראשון, השני וכן הלאה).

כל זאת, כיוון שאנו מבצעים את ההכנסה מהסוף.

Data structures - summary

Radix sort .ii

ההנחה עבור אלגוריתם מיון זה כי המספרים שנקבל בעלי d ספרות.
אנו נבצע סידור של האיברים עפ״י ערך הספרה האחרונה, ואז עפ״י ערך הספרה אחת לפני האחרונה עד שנגיע לסידור עפ״י הספרה הראשונה.
נשים לב, כי זהו מיון יציב וכי אנו שומרים על הסדר היחסי בכל שלב.

ניתוח זמן הריצה הצפוי הינו $\theta(d(n+k))$ כך ש-n מספר האיברים, d מספר הספרות ו- k ערך הבסיס(בסיס 10 למשל, עשרוני).
נשים לב שלערך הבסיס יש השפעה. מצד אחד, ככל שערך הבסיס יקטן כך d יקטן ונצטרך לבצע פחות איטרציות.
מצד שני, ערך ה-k המייצג את אורך הספרה יגדל ונצטרך לבצע בדיקה יותר ארוכה עבור כל ערך.
לכן, זמן הריצה עבור n איברים ו-b ביטים יהיה $\theta(\left(\frac{b}{r}\right)(n+2^r))$.
במצגת מצאו את ה-r עבור n מספיק גדול זמן החיפוש יהיה האופטימלי. יש לחזור על זה.

Bucket sort .iii

ההנחה עבור אלגוריתם מיון זה כי כל ערכי המספרים קטנים מ-1. (במידה וערכי המספרים גדולים מ-1 נוכל לחלק אותה בערך העשרוני של האיבר הגדול ביותר).
נבנה מערך עזר B המכיל את המספרים מ-0 ועד 9.
במיון זה אנו נבצע הכנסה של כל ערך עפ״י הערך התחתון. למשל, עבור 0.75 – ערך זה יכנס לתא 7 במערך עזר B.
בהנחה כי כל האיברים נבחרו רנדומלית בטווח בין 0 ל-1 הזמן הצפוי הינו $\theta(n)$. חישוב של התוחלת עבור אינדיקטור מקרי מופיע במצגת.

02.06.22

Data structures - summary

Amortized analysis

ניתוח פחת – זהו ניתוח לשיעורין.

ביצוע חישוב של החסם העליון עבור המקרה הגרוע ביותר כחישוב של **סדרה של פעולות** עבור מקרה זה.

3 שיטות לחישוב:

א. שיטת הספירה הזהירה -הגדרת t פעולות. חישוב הכולל של הפעולות בצורה מוקפדת, וחלוקה ב-t.

ב. שיטת החיובים האסימונים – לכל פעולה יש עלות וחיוב. חיוב הוא הסכום אנו "משלמים" בעת ביצוע הפעולה.
לכל פעולה עלות המתארת בדיוק כמה לקחה.
אנו מגדירים את החיוב עפ"י "ניתוח העתידי של הפעולות העלולות לקרות" (צריך להראות שכאשר מתבצעת פעולה בודדת יש מספיק כסף (מתוך החיוב) או מתוך הקרדיט לשלם עבורה).
כל סכום שנשאר מפעולה נשמור בקרדיט.

ג. שיטת הפוטנציאל – בשיטה זו אנחנו מגדירים מהו הפוטנציאל של מבנה הנתונים לאחר כל פעולה.

נגדיר c_i העלות של הפעולה ה-i במבנה.

נגדיר D_i מבנה הנתונים אחר הפעולה ה-i.

נגדיר ϕ את פונקציית הפוטנציאל. לכן ניתוח השיעורים הוא $\bar{c} = c_i + \phi(D_i) - \phi(D_{i-1})$ כלומר – החיוב שווה לפוטי לאחר הפעולה ה-i פחות לפני הפעולה ה-i ועוד העלות.

$BB[\alpha]\ tree$ ——

עץ בינארי רגיל אשר משמר יחס α בין בניו. (כך ש- $\alpha \in (0,1)$)

כלומר, עבור כל קודקוד v בעץ, גודל תת העץ השמאלי(מספר הקודקודים) גדול מ- $\alpha * v$ וכנ"ל עבור תת העץ הימני. (ניתוח פחת במצגת. יש לעיין).

Data structures - summary

Graphs

ישנן מס׳ דרכים לייצג קשתות בגרפים, השניים העיקריות הן כמטריצת ייצוג וכרשימה מקושרת.

חשובה ההבנה בהבדל בין השיטות: מטריצה – גישה ב- $\theta(1)$ (לאחר אתחול של המטריצה) אך מקום של $\theta(n^2)$, וברשימה מקושרת הגישה בזמן של $\theta(number\ of\ edges\ to\ vertecies\ u)$ והמקום ב- $\theta(V+E)$.

כעת נראה מספר אלגוריתמי חיפוש:

אלגוריתם BFS – אלגוריתם חיפוש לרוחב.

אלגוריתם זה מקבל גרף, נסמנו ב-G, וקודקוד, נסמנו ב-s, ובודק את המרחק של כל קודקוד בעץ מ-s.

הסבר מימוש האלגוריתם ב- high level: נגדיר שלושה צבעים לכל קודקוד.

צבע לבן עבור קודקוד שטרם התגלה. צבע אפור עבור קודקוד שהגענו אליו אך טרם סיימנו לעבור על שכניו ושחור עבור קודקוד שסיימנו איתו.

נגיע לקודקוד u נסמנו באפור, נכניס את השכנים שלו (לבנים בלבד) למחסנית ונשנה את מרחקם להיות 1+ ביחס לקודקוד שגילה אותם.

נצבע את הקודקוד u בשחור ונמשיך באותו אופן לעבור על שכניו.

זמן הריצה עבור אלגוריתם זה הינו $\theta(|E|+|V|)$.

בזמן ביצוע האלגוריתם עדכנו כל קודקוד להיות מי שמצא אותו כאבא שלו.

לכן, על מנת למצוא את המסלול מקודקוד s לקודקוד u נוכל לבצע BFS ואז לעבור בצורה רקורסיבית מהאבא של s עד שנגיע לאבא כערך של u.

נגדיר עץ BFS להיות עץ פורס אשר אין בו מעגלים (מבחינה הבנית – כל הקשתות אשר גילו קודקוד והגיעו אליו נסמנם).

אלגוריתם DFS – אלגוריתם חיפוש לאורך.

העיקרון המרכזי של אלגוריתם זה הוא קבלת קודקוד s בגרף וכל צעד להתקדם לעבר השכן של הקודקוד הבא שלא בוקר וכן הלאה, ואז לבצע באקטרקינג.

בסוף התהליך נבצע בשנית על כל שאר הקודקודים שלא בוקרו בכלל בגרף.

כמובן שעבור אלגוריתם חיפוש זה נעדיף לממש את הקשתות כרשימה מקושרת. זמן הסיבוכיות יהיה $\theta(V+E)$.

נגדיר יער DFS להיות כקבוצות של תתי העצים המחוברים בקשתות מכוונות לעבר הקודקודים אותם גילו.

Data structures - summary

כעת נציין מספר משפטים בהן עסקנו בהרצאה -

א. משפט הסוגריים – נגדיר אינטרוול של קודקוד s כזוג סדור (זמן ההתחלה של סריקת הקודקוד s, וזמן הסיום של קודקוד s).

v צאצא של קודקוד u ביער DFS אמ״מ האינטרוול של u מוכל באינטרוול של v.

במידה ואין ביניהם יחס של הכלה אז בהכרח הם זרים(כי לא יכול להיות ביניהם חיתוך) ועל כן הם נמצאים בעצים שונים.

ב. משפט המסלול הלבן – ביער חיפוש DFS נאמר כי v צאצא של u אמ״מ בזמן החיפוש של קודקוד u היה קיים מסלול לבן מ-u אל v. (יש להכיר את ההוכחה, שימוש במשפט הסוגריים). צד שני לבדוק

סיווג קשתות הגרף ל- 4 סוגים :

קשתות העץ – אלו הקשתות הרגילות שמופיעות במהלך החיפוש והגילוי ביער DFS.

קשתות אחורה – כל הקשתות (u, v) כך ש- v אב קדמון של u.

קשתות קדימה – כל הקשתות (u, v) כך ש- v צאצא של u, ו-v אינו ילד של u .

קשתות חוצות – שאר הקשתות.

ניתן לסווג את הקשת (u, v) ע״י צבע הקודקוד u במהלך הבדיקה -

א. u היה לבן – קשת העץ.

ב. u היה אפור – קשת אחורה.

ג. u היה שחור – קשת קדימה/ קשת חוצה.

שימוש ראשון ונפוץ : בדיקת קיום מעגל בגרף.

א. גרף G חסר מעגלים אמ״מ היער DFS לא מכיל קשתות אחורה.

ב. מציאת מיון טיפולוגי (קיים רק עבור DAG, הסבר למטה).

גרף DAG הינו גרף מכוון חסר מעגלים. מיון טופולגי הוא מיון לינארי כך שכל קשת (u, v) , u מופיעה לפני v.

Data structures - summary

ניתן להשתמש באלגוריתם ה-DFS לטובת מיון טופולוגי.

נבחר קודקוד אקראי ונבצע עליו את האלגוריתם ובכל שלב בו הקודקוד נהיה שחור נמקם אותו בראש הרשימה.

Disjoint sets

זהו ADT לתחזוק קבוצות זרות.

מבנה הנתונים, C , מכיל קבוצה של $Disjoint$ כך שלכל $s \in C$ קיים נציג לקבוצה.

מבנה הנתונים נתמך ביצירת קבוצה, איחוד ומציאת נציג.

דוג לשימוש במבני זה הוא חישוב רכיבי הקשירות של גרף לא מכוון (או יצירת קבוצות של רכיבי קשירות).

ראינו מספר מימושים לדוג' זו :

א. מימוש ראשון – רשימה מקושרת.

כל קבוצה מיוצגת ונשמרת ע"י רשימה מקושרת. ההבדל היחיד מרשימה מקושרת סטנדרטית זה שיש חוליה נוספת "ראשית" וכל חוליה ברשימה מצביעה עליה.

חוליה "ראשית" זו מחזיקה שתי שדות – שדה ראשון המצביע על הראש(המייצג) ושדה השני המצביע על האיבר האחרון.

במימוש זה עלות הריצה עבור – ()Makeset ו Findset הינו $\theta(1)$ ועבור האיחוד במקרה הגרוע ביותר הינו $\theta(n)$.

ראינו כי גם עבור ניתוח פחות נקבל כי זמן הריצה הינו $\theta(n)$.

מה שמוביל אותנו לחשוב על מימוש נוסף.

ב. מימוש שני – רשימה מקושרת ומיזוג לפי גודל. הרשימה הקטנה תתמזג לרשימה הגדולה מה שיוביל "לחיסכון" בשינוי המצביעים וההפניות באיחוד.

אולם, גם עבור מימוש זה המקרה הגרוע ביותר הינו $\theta(n)$ עבור שתי קבוצות אשר כל אחת בגודל n/2.

אלם בחישוב מדויק יותר נקבל שעבור רצף פעולות m , כאשר u הינו מספר פעולות האיחוד זמן הריצה הינו $O(m + u \log u)$.

ג. מימוש שלישי- כל רשימה מיוצגת כ"עץ".

כל קודקוד בעץ מחזיק פוינטר לעבר אבא שלו, והשורש מחזיק אל עצמו.

Data structures - summary

גם כאן האיחוד יכול לקחת $\theta(n)$ כיוון שעבור חיפוש המייצג כאשר האיבר הוא העלה האחרון וגובה העץ הינו $\theta(n)$ זה יהיה זמן האיחוד.

ניתן לפתור בעיה זו בשתי שיטות:

א. איחוד הקבוצות לפי גובה. נאחד את העץ בעל הגובה הנמוך יותר לעץ עם הגובה הגבוה יותר. לאחר מספר תובנות נסיק כי זמן הריצה הינו $O(\log n)$.

ב. כיווץ מסלולים בכל פעולה של *Findset(x)* – שהרי אנו עוברים בכל מקרה במסלול מהקודקוד לעבר השורש אז נעדכן את המצביעים של כל אלו לשורש ובכך נוכל לכווץ את גובה העץ.

עבור שיטה זו נחזיק שדה *rank* (ולא *height*, כי אין אנו יודעים מה גובהו אחר הכיווץ) ונבצע איחוד מהעץ שדרגתו נמוכה יותר לדרגתו של העץ הגבוהה יותר.

נשים לב, כי אם נאחד את המימושים כלומר, נבצע גם כיווץ וגם מיזוג עפ"י ה- rank נקבל כי זמן עבור איחוד הינו $O(\log n)$ ועבור m פעולות זמן הריצה הינו $O(m\alpha(n))$ כך ש- α בסדר גודל קבוע.

MST – Minimum spanning tree

הגדרת עץ פורש מינימלי – גרף לא מכוון G אשר לכל קשת יש משקל. אנו שואלים מה המסלול שעובר בכל קודקודי הגרף בעל המשקל הנמוך ביותר.

שלושה משפטים עיקריים שליוו אותנו במהלך הרצאה זו:

א. גרף לא מכוון בעל *n-1* קודקודים $\Leftrightarrow$ הגרף הינו עץ (קשיר) $\Leftrightarrow$ קיים מסלול בין כל שני קודקודים בגרף.

ב. לכל גרף G אשר מכיל בדיוק מעגל 1, לכל קשת e אשר נמצאת במעגל כאשר נוריד אותה מהגרף ונוסיף צלע חדשה נקבל מעגל.

ג. יהי A תת גרף של MST, הוספת קשת קלה (הגדרה בהמשך) ל-A משאירה אותו בטוח (כלומר מוכל ב-MST).

יהי A תת גרף של MST.

נגדיר "קשת בטוחה" כקשת כאשר נוסיף אותה ל-A, תת גרף A עדיין יהיה מוכל ב- MST.

Data structures - summary

יהי A חיתוך של קודקודי G לשתי קבוצות כך שכל קבוצה מכבדת את A. כלומר, אין קשתות אשר נחתכות בחתך זה.
קשת קלה חוצה מוגדרת כקשת עם משקל מינימלי מכל הקשתות אשר חוצה את החתך.

אלגוריתם הכללי של GenericMST - נגדיר קבוצה A כקבוצה ריקה וכל פעם נוסיף קשת בטוחה. נבצע פעולה זו n-1 פעמים.
בתהליך זה אנו מבטיחים כי A יהיה קשיר (הוספת n-1 צלעות), גרף ממשקל מינימלי (כיוון שאנו מוסיפים בכל פעם קשת בטוחה) ולכן נקבל MST.

אלגוריתם קרוסקל – נאתחל קבוצה A כקבוצה ריקה. נמיין את הקשתות עפ״י המשקל שלהם ונוסיף אותם כל פעם ל-A עפ״י יחס סדר המשקל שלהם ובתנאי שאינם יוצרים מעגל.
אפשרות למימוש אלגוריתם זה - שימוש ב disjoint ויצירת קבוצה לכל איבר. מיון הקשתות עפ״י המשקל שלהם ומעבר על קשתות.
אם קודקודי הקשת אינם תחת אותה קבוצה נבצע איחוד של אותן קבוצות ונוסיף קשת זו לחלק מהעץ המינימלי הפורש.
זמן הריצה הינו $O(E\log V)$, ובמידה הקשתות ממוינות זמן הריצה הינו $O(E\alpha(V))$.

אלגוריתם פרימס – נאתחל קבוצה A כקבוצה ריקה. נתחיל מקודקוד a ונתבונן בכל שכניו. נוסיף את ערך הקשת בעלת המשקל הנמוך ביותר.
לאחר מכן, נעבור לקודקוד זה, ונבדוק עבור שכניו מה הערך המינימלי (ואינה יוצרת מעגל).
אפשרות למימוש אלגוריתם זה – נבנה תור עדיפות אשר מכיל את כל הקודקודים בגרף. כל קודקוד יכיל זוג סדור בערך דיפולטיבי של ∞ ו null. נאתחל קודקוד a לערך 0.
כל עוד תור זה אינו ריק, כלומר קיימים קודקודים נבצע עבור כל קודקוד u כך:
נחלץ את המינימום שלו מבין כל ערימה (ניתן לבצע זאת ע״י ערימת מינימום) – נעבור על כל שכניו וניקח את הערך המינימלי ונוסיף אותה כקשת בגרף.
נבצע תהליך איטרטיבי זה על מעבר כל הקודקודים.
זמן הריצה הינו $O(E\log v)$.

02.06.22

Data structures - summary

תרגולים קרן ברגר – שיטות ודרכי פתרון

תרגול 1 - מבוא

<u>תרגיל 1 :</u> שאלה על מציאת איבר מקסימלי ואיבר שני בגודלו.

על פניו בחישוב ראשוני מציאת האיבר המקסימלי יהיה בזמן ריצה של $n-1$ ועבור השני בגודלו $n-2$ וכן הלאה במידה ויהיו עוד איברים שנרצה למצוא את גודלם.

פתרון יעיל יותר – בניית עץ תחרות שהמנצח בכל פעם הוא בעל הערך הגדול יותר.

לכן, עבור מציאת האיבר המקסימלי הראשון נצטרך לבצע גם $n-1$ השוואות אך עבור האיבר המקסימלי הבא נוכל לבצע רק $\log_2 n$ השוואות (נצטרך לעבור בכל רמה בעץ, כגובהו, וזה מספיק לדעת כדי למצוא את האיבר השני. ע״י כך שנעבור בכל המסלול של ה-max ולהשוואות את האיברים ״שהתחרו״ בו)

<u>תרגיל 2 :</u> אין תובנה מרכזית, אלא מקרה של חישוב ריצה.

מקרה ריצה בו יש n איברים ויש איטרציה על כל האיברים כך שבכל איטרציה מאתחלים משתנה בערך k להיות 1.

לאחר מכן, בכל לולאה פנימית יש הכפלה של הערך k ב-2 כל עוד קטן מ-i. זמן הריצה עבור מקרה זה הינו $\theta(\log_2 n!)$ לראות שמבין למה.

<u>תרגיל 3 :</u> מקרה ריצה בו אנו מחשבים למשל ערך וחזקה עבור קלט מסוים.

המקרה הטריוויאלי הוא קריאה לפונקציה פעמיים כאשר יש את ערך ההכפלה ביניהם. נשים לב שזמן ריצה זה $n-1$

לעומת זאת, ייעול של האלגוריתם זה להבחין כי המעבר בקריאות זהה ולכן לבצע רק קריאה אחת ולהכפיל אותה בעצמה. וזהו זמן ריצה של $\log n$.

<u>תרגיל 4 :</u> מציאת איבר מינימום ומקסימום.

לקיחת המערך וחלוקתו לזוגות – עבור כל זוג השוואה מי יותר גדול ומיון למערכים $A_1 and\ A_2$.

לבסוף, מיון המינימום מ-A_1 ומיון המקסימום מ-A_2.

02.06.22

Data structures - summary

תרגול 2 – זמני ריצה

הערה: טריק שימושי - פיצול של $\log(n!)$ לסיגמא של סכום ה- logים.

הערה: טריק שימושי - $2^{\log n-1} * (\log n) < (\log n - 1)(\log n) = !(\log n)$ עבור $k \geq 4$.

תרגול 3 - ADT

<u>תרגיל 1:</u> מימוש תור ע"י שימוש ב-2 מחסניות. שיטת המימוש ע"י כך שכל פעם שנכניס איבר נעביר את כל האיברים למחסנית השנייה, נכניס את האיבר, ואז נחזיר את שאר האיברים. (פעולת ההכנסה יקרה, אך ההוצאה זולה. ניתן לשפר את יעילות ההכנסה אך זה יבוא על חשבון ההוצאה).

<u>תרגיל 2:</u> מטריצה בוליאנית – והחזרה ב θ קבוע פרמטרים שהוגדרו בשאלה. (הכיוון חשיבה בשאלה זו – זה החזקת שדות נוספים ומערך המונה את מספר האחדות ואפסים, ולא לבצע חישוב כל פעם מחדש).

<u>תרגיל 3:</u> תרגיל חשוב. קונספט דומה לתרגיל 2. לקחים עיקריים:

א. כאשר מבקשים להחזיר את האיבר האחרון שנכנס יש לחשוב לא רק על מימוש של $stack$ אלא גם על מימוש של רשימה דו כיוונית.

ב. כאשר מבקשים החזרה ב- $\theta(t)$ כך ש-t הינו קבוע ביחס לאיברים שנכנסו לפני איבר מסוים האינטואיציה על רשימה דו כיוונית גבוה יותר מאשר המימוש של $stack$, כיוון שבמימוש מחסנית הגישה לאיברים שנכנסו לפני תהיה יותר מסורבלת ותלויה דווקא באיברים שיכנסו ולא שנכנסו.

תרגיל 4: **תרגיל חשוב** מערך חכם – ב-$\theta(1)$ לכל הפעולות במערך.
הסבר - שימוש בשדות נוספים ושלושה מערכים אשר כל מערך מייצג סדר הכנסה/זמן הכנסה וכו'.
שלושה מערכים – מערך A, אשר שומר את כלל האיברים שהוכנסו, מערך B מייצג את זמן ההכנסה פייר איבר, ומערך C השומר רק את האיברים שנכנסו באתחול האחרון.
בכל פעם שאנו רוצים לקרוא ממערך A אנו נסתכל במערך B באינדקס של A ונבדוק עבור ערך זה האם הוא גדול מן השדה top – במידה וכן נבדוק האם הוא תואם לזמן ההכנסה ב-c. אם תואם – נחזיר את הערך במקורי ב-A ואם לא נחזיר את ערך האתחול האחרון שהיה.

בשאלות בהם נתבקש להשתמש בשליפה, אתחול ובנייה ב- $\theta(1)$ללא אילוץ על הזיכרון – יש לחשוב על מערך זה!

02.06.22

Data structures - summary

תרגול 4 - עצי BST

<u>תרגיל 1</u> : טענה מתרגיל זה כי משתי סריקות שונות של עץ T, כאשר אחת מהן היא Inorder ניתן לשחזר בצורה וודאית את העץ.

<u>תרגיל 2</u> : טענה והוכחה באינ׳ על היחס בין מספר עלים לדרגת הקודקודים. נשים לב, שעבור דרגה 1 בעץ בינארי אין הוספה של עלה לעץ , אך קודקוד עם דרגה 2 יש הוספה, עקב פיצול נוסף, של עלה לעץ.

<u>תרגיל 3</u> : הדפסת המסלול המקסימלי בעץ. לקח מרכזי משאלה זו – לא לשכוח שרקורסיה זה עוד כלי המאפשר לנו להשיג תוצאה.
בשאלה זו, פירוק הבעיה פיזית וניסיון בידיים להרכיב את הפתרון היה נותן לנו אינדיקציה מעולה שמדובר ברקורסיה(כי היינו שמים לב, שאנו רוצים לבדוק כל זוג קודקודים **מלמטה למעלה** ולהחזיר את הערך המקסימלי).

הערה : נשים לב, שבתהליך המחיקה עבור קודקוד שיש לו שני בנים אנו מבצעים חיפוש עבור ה-$succ$, מוחקים אותו ומבצעים החלפה עם הערך הנמחק.
ההנחה שלנו, והיא לגמרי נכונה אך יש לשים לב שאנו מבינים אותה היא שאין ל$succ$ בין שמאלי ולכן לאחר המחיקה לא צריך לדאוג לתת העץ השמאלי שלו.

<u>תרגיל 4</u> : קבלת קלט אינדקס i ומציאת מספר המפתחות הגדולים ממנו. שאלה זו חזרה במספר וארציות בקורס, פתרון לבעיה זו זה שימוש בשדה $size$ עבור כל תת עץ.
נבצע ירידה מהשורש לעלים נבצע בדיקה האם הערך i גדול או קטן מן המפתח בקודקוד. אם הוא קטן, ניתן להוסיף את סכום כל תת העץ הימני(ויש לנו את השדה הזה) והמשך חיפוש בתת העץ השמאלי.
ואם הוא גדול, אז ממשיכים חיפוש כרגיל עד שנגיע לאותו קודקוד.

<u>תרגיל 5</u> : נתונים שני עצי חיפוש בינאריים ונתון כי כל ערכי T_1 גדולים ממש מ- T_2 ויש לבצע איחוד של שניהם ב- $\theta(\min(h_1,h_2))$.
פתרון לשאלה זו הוא חלוקה למקרים עבור $h_1 > h_2$ ועבור $h_1 < h_2$.
תמיד נתבונן בתת העץ הנמוך היותר וניקח את הקודקוד המינימלי/המקסימלי (תלוי האם העץ שמתמזג אליו קטן מכל ערכיו או גדול). נסיר אותו ונשים אותו כשורש.
בשלב הבא נחבר אליו כתת העץ הימני את העץ הגדול יותר וכתת העץ השמאלי את העץ הקטן יותר. כך הצלחנו לבצע ב- $\theta(\min(h_1,h_2))$ איחוד של שני העצים.

Data structures - summary

<u>תרגיל 6 :</u> בדיקת האם עץ הוא עץ BST או לא.
פתרון שגוי נפוץ עבור שאלה זו הוא לבצע בדיקה בצורה רקורסיבית כי כל בן שמאלי קטן יותר מערך הקודקוד הנוכחי וכי הבן הימני גדול יותר.
זו טעות כיוון שיכול להיות שבכל שלב הטענה הזו מתקיימת אך כעץ אין שמירה על ההגדרה.
פתרון נכון לשאלה זו זה שמירה של הגבולות העליונים והתחתונים של הערכים המורשים בכל תת העץ ובדיקה עבור טווח זה.
עוד פתרון לשאלה זו, הוא ביצוע הליכת $Inorder$ ובדיקה שהיא ממוינת בסדר עולה. זמן הריצה עבור שני הפתרונות הנ״ל הם $\theta(n)$

תרגול 5 – עצי AVL

שאלות 1-2 הינן שאלות שהוכחו באינדוקציה על הגדרה שמפורטת במצגת.
<u>שאלה 1 :</u> מתוארות פעולות ADT וצריכים להציע מבנה נתונים נכון עבור פעולות אלו.
דגשים משאלה זו – גם כאן בדומה לשאלות לעיל השתמשו בשדה לכל קודקוד הנקרא $size$ אלא שכאן מבני השאלה היה טיפה אחר והיה צריך לבצע הוספה של $size$ בקודקוד בכל פניה ימינה בעץ.

<u>שאלה 2 :</u> שאלה 4 במצגת הינה שאלה ארוכה ומורכבת. מומלץ לעבור עליה דרך המצגת.
דגשים משאלה זו – שימוש בשני עצי AVL כך שאחד מחזיק את ערכי הקודקודים והשני את ערכי ההכנסה עם פוינטרים ביניהם. בדומה לשאר השאלות, שימוש ושילוב של למבני נתונים.

תרגול 6– B-trees and probability

תרגיל 1 : נתוני שני עצי B , T_1T_2 בנוסף נתון כי הערכים בעץ T_1 קטנים ממש מכל הערכים בעץ T_2. אלגוריתם לאיחוד-
פתרון – עבור המקרה ששניהם אותו גובה ניצור ״שורש מדומה״ שיהיה בין ערכי המינימום בעץ למקסימום בעץ השני ואת שני תתי העצים נחבר אליו.
במידה וגבהי העצים אינם זהים גם כאן ניצור ״ערך מדומה״ , נסמנו ב-k, אשר יהיה הערך בין המיני׳ למקסי׳. נרד בתת העץ T_1 עד אשר נגיע לצומת כגובה העץ T_2 ועוד אחד. במהלך כל ירידה במידה ויש צומת מלאה בדרך נבצע פיצול.
נוסיף את הערך k לצומת אליה הגענו ואז נבצע הכנסה של העץ T_2 , לבסוף נבצע מחיקה של הערך k.
דגש מרכזי משאלה זו- ניתן ליצור ערכים שישמשו אותנו לטובת הבעיה הנתונה ולא לשכוח להיעזר בפונקציות העץ(כמו במקרה הנ״ל שאנו מסתמכים שלאחר מחיקת ערך k האיזון יתבצע לבד).

02.06.22

Data structures - summary

הערה: זכירת הנוסחה של גובה העץ ב- *Btree* יכולה להקל בחישובי גבהים עבור ערכי מפתחות.

הערה: דוגמאות למחיקה והכנסה בעץ *Btree* מופיעות במצגת. בנוסף, דוגמאות להסתברות ותוחלת. אין דגשים מיוחדים – תרגילים פשוטים.

תרגול 7 Hash-table

תרגיל 1: מערך של מספרים ממשיים A וערך כלשהו X ,למצוא האם קיימים שני ערכים במערך שסכומם הוא X.

פתרון – מיון המערך ב- $O(n\log n)$ ואז חיפוש עם שני מצביעים(בדומה לפתרון מתרגול 1 שאלה 5).

במידה ומבקשים מאיתנו ב- $O(n)$ זמן צפוי טרם הפתרון במידה ורוצים להשתמש בטבלאות גיבוב יש לבדוק -

האם הנחת ה-Suha מתקיימת או שצריך להשתמש בפונקציות גיבוב אוניברסליות. חשוב, יש לציין זאת!

בפתרון לשאלה זו – ניקח פונקציית גיבוב אוניברסלית ונבצע מיפוי לכל ערכי המערך.

לאחר מכן, עבור כל ערך במערך נבדוק האם X-A[i] נמצא בטבלת גיבוב.

ע״י שימוש בפונקציית גיבוב תוחלת זמן הגיבוב הינו $O(1)$ ללא תלות בהתפלגות הערכים. ולכן תוחלת זמן הריצה של האלגוריתם הינה $O(n)$.

תרגיל 2: נתון מבני נתונים ושואלים עבור מימוש למבני נתונים זה.

הפתרון המלא במצגת – העיקרון החשוב עבור שאלה זו זה האפשרות לממש שני מבני נתונים שונים(במקרה הנ״ל טבלת גיבוב וגם רשימת דילוגים) כך שכל אחד מהם יענה על צורך אחר במני הנתונים המבוקש.

תרגיל 3: שתי קבוצות אחת בגודל m והשנייה בגודל n כך ש- $m \le n$.

בסעיף א׳ מציאת אלגוריתם דטרמיניסטי בזמן ריצה יעיל במקרה הגרוע ביותר – עדיף למיין את המערך הקטן יותר, ואז עבור כל איבר במערך הגדול לחפש אותו בתוך המערך הקטן (ונחזיק counter שיעלה באחד , ולבסוף נבדוק האם הוא שווה בגודלו לגודל של המערך הקטן).

כך נקבל זמן ריצה של $O(mlogm + nlogm) = O(n\ logm)$.

צריך לחשוב על זה כך שבכל מקרה יהיה חיבור של m ו-n עבור כפל של log x ולכן אנו צריכים לשאוף ש-x יהיה קטן כאפשר ולכן המיון יהיה עבור המערך הקטן.

מיון של המערך הגדול וחיפוש בו היה נותן לנו $O(n\log n)$ – וזהו זמן ריצה גרוע יותר!

בסעיף ב׳ מציאת אלגוריתם דטרמיניסטי בזמן ריצה צפוי יותר טוב – אתחול טבלת גיבוב בגודל m , הכנסת הערכים של T (בגודל n) לטבלה ולאחר מכן מבצעים m

Data structures - summary

חיפושים לכל היותר בתוך הטבלה. לכן נקבל זמן ריצה $O(m+n)=O(n)$ (אין חשיבות לאתחול עבור n או m)

הערה: כאשר מדברים על זמן חיפוש צפוי יש לחשוב על רשימת דילוגים או טבלת גיבוב.

תרגיל 4: השימוש בשאלה זו בבלום פילטר הוא ממש נכון כיוון שעל מנת לבדוק האם $S \subseteq T$ כאשר אנו יוצרים בלום פילטר עבור S נצטרך שכלל האיברים ב- $S\ /\ T$, נסמנם ב-x, יקבלו false-positive שההסתברות לכך היא $(1-p)^{xk}$.

תרגול 8 Heaps Huffman code

<u>תרגיל 1</u>: הוצאת איבר מהאינדקס i במערך.

דגש לפתרון - נשים לב שבמחיקת איבר יש לבצע גם *Minheapfy* וגם *decreaseKey* (עבור הפונק׳ הראשונה, בערימת מינימום לדוג׳, לוודא שכל תת העץ מתחת גדול מן הערך שהוחלף, כי הרי שהערך שהוחלף הגיע מהעלה. עבור הפונק׳ השנייה לוודא שהוא גדול מאביו, כי יכול להיות שהוא עבר לתת עץ אחר).

<u>תרגיל 2</u>: בהינתן ערמת מקסימום H , סיבוכיות הזמן למציאת C האיברים הגדולים בH.

פתרון: הפתרון הנאיבי הוא לבצע C פעמים את הפעולה ExtractMax ולכן נקבל סה״כ $O(C\log n)$.

פתרון יעיל יותר עבור C קבוע ולא קבוע.

C קבוע - הוא מציאת האיבר ה-c בגודלו ועל כן שאר האיברים שגדולים ממנו הם c-1 ומכך נוכל האיברים הגדולים ביותר לוקח $O(C^2)=O(1)$

C תלוי ב-n - לכן נשתמש בערימה נוספת אשר תכיל איברים מהערימה המקורית.

כל איבר שיכנס לערימה החדשה מחזיק גם מצביעים לילדים המקוריים שלו

נכניס לערימה החדשה את השורש של הערימה המקורית.

לאחר מכן נבצע C פעמים Extract-Max() על הערימה החדשה ובכל פעם שמוציאים איבר, נכניס לערימה החדשה את הבנים שלו מהערימה המקורית.

בכל שלב בערימה החדשה יש את האיברים הפוטנציאלים להיות האיבר הבא בגודלו.

בערימה החדשה יש פחות איברים (לכל היותר C) לכן נקבל כי הזמן הוא $O(C\log C)$. פתרון חשוב לזכור. העקרון המנחה הוא ניצול התהליך של Extract-Max() לאיטרציה הבאה.

Data structures - summary

<u>תרגיל 3 :</u> מציאת האיבר ה-i הקטן ביותר במערך מגודל n.

יש שני פתרונות : הראשון(פחות יעיל) – למיין את המערך בזמן ריצה של $O(n\log n)$ ואז לחלץ את האיבר הראשון סה"כ נקבל $O(n\log n)$

הפתרון השני(היותר יעיל) – לבנות ערימה מהמערך בזמן ריצה של $O(n)$.

נבצע *i Extractmax()* פעמים סה"כ בזמן ריצה של $O(i\log n)$ ואם נשתמש בשאלה קודמת נוכל להקטין את הבעיה ל -$O(n+i\log i)$

הדגש המרכזי מהשאלה הזאת – אין צורך לבצע מיון או סידור של מערך תמיד. למשל כמו במקרה הנ"ל שימוש בערימה עדיף כיוון שזמן הריצה שלה בבנייה יותר נמוך ממיון $(n\ logn)$ $vs\ n//$ מאפשר לנו הוצאה מהירה.

<u>תרגיל 4 :</u> נתונות שתי ערימות $H_1 and\ H_2$ בגבהים $T_1 and\ T_2$ כך ש-כל האיברים בערימה 1 גדולים ממש מהאיברים בערימה השנייה. בניית ערימה ב- $O(n_2)$

העיקרון המנחה בתרגיל זה : חלוקה לגבהים. אם $n_2 > n_1$ בניית ערימה חדשה מאיחוד של שניהם ונקבל סה"כ $O(n_1+n_2)=O(n_2)$

אם $n_2 \le n_1$ נצרף את איברי המערך H_2 לסוף המערך של H_1 ובחישוב עפ"י נוסחת מציאת האבא (מוסבר במצגת) נוכל לראות שכל האיברים שיכנסו יהיו עלים ובכך שימרנו את תכונת הערימה ואת זמן הריצה המבוקש $O(n_2)$. דגש חשוב -שליטה במערך, בנוסחת הילדים והאבא.

<u>תרגיל 5:</u> מיון k מערכים ממוינים בזמן ריצה $O(n\log k)$.

פתרון- אותו עקרון מנחה כמו בשאלות הראשונות. נבנה ערימה עם k האיברים הראשונים מכל מערך. סה"כ זמן ריצה של $O(k)$.

לאחר מכן, נבצע הוצאה של האיבר המינימלי מן הערימה, והכנסה של האיבר הבא מאותו מערך בו היה האיבר המינימלי. בסה"כ נקבל $O(n\log k)$.

הערה : נשים לב, כי עבור קוד האופמן שהייצוג שלו יהיה עפ"י סדרת פיבונאצי נקבל עץ אחד ארוך.
התופעה הזאת קוראת כיוון שסכום כל הערכים של n האיברים בסדרת פיבונצי קטנים ממש מהערך במקום *n+2*.
ולכן, כל פעם שניקח את שני הערכים הקטנים נצטרך לבצע שימוש בערך שכבר חישבנו (כלומר, שיש לו כבר "גובה").

02.06.22

Data structures - summary

תרגול 9 – *Lempel ziv & Merge sort*

<u>תרגיל 1 :</u> עבור אלגוריתם - כיווץ זה הטקסט הארוך והקצר ביותר שייצר n זוגות.

אורך הטקסט עבור הטקסט הארוך ביותר - $\theta(n^2)$ (העץ יראה כמו ענף אחד ארוך שכל בלוק מכיל את הבלוק לפניו עם אות אחת נוספת).

עבור הטקסט הקצר ביותר - $O(n\log n)$ (לעץ בכל רמה יהיה מספר בנים בגודל של מספר האלפבית).

לכן, עבור הרמה הראשונה יהיה לו אות אחת סה״כ $27 * 1$ ועבור הרמה השנייה שתי אותיות סה״כ $27^2 * 2$ וכן הלאה.

החישוב המלא מופיע במצגת, צריך לעבור עליו נקבל את הזמן שרשמנו לעיל.

הערה : אלגוריתם מיון *Merge-sort* במקרה הגרוע ביותר הינו $O(n^2)$ ועבור מערך אקראי נקבל $O(n\log n)$.

אז מה המוטיבציה להשתמש בו ולא להשתמש במבני שמבטיח תמיד זמן ריצה של $O(n\log n)$ כמו merge-sort ?

א. ההסתברות ל- $O(n\log n)$ קרובה ל-1 בmerge sort.
ב. הקבועים של אלגוריתם זה טובים משל merge sort.
ג. Quicksort אלגוריתם מיון In-place ולכן דורש פחות זיכרון.

<u>תרגיל 2 :</u> דרך להפוך מיון מבוסס השוואות לא יציב למבנה יציב הוא הגדרת עוד מפתח שיוגדר לפי האינדקס שלו.

בדרך זו נשווה בין הערכים ונמיין לפי הגודל שלהם. במידה והם שווים בערכם, נמיין לפי האינדקס שלהם.

<u>תרגיל 3 :</u> איך ניתן ב- $O(n)$ לבדוק האם קיים מספר שמופיע יותר מ- $\frac{n}{2}$ (ערך עליון)?

לקח עיקרי- ההבנה שאם קיים איבר כזה אז הוא בהכרח מופיע בחציון. נשתמש בפונקציית *select* שתאפשר לנו למצוא את החציון ואז נספור את כמות החזרות שלו במערך. סה״כ נקבל $O(n)$.

<u>תרגיל 4</u> : מיון מערך בזמן $O(n)$ כאשר כלל המפתחות בערכים בין 1 ל-k.

בשאלה זו התבקשנו להשתמש באלגוריתם מיון in-place . הפתרון הוא להשתמש באפשרות מיון ע״י שנפעיל *Partition* כל פעם על פיבה i כך ש- i רץ מ-1 ועד k.

הפעולה *Partition* עולה $O(n)$ ואותה אנו מבצעים k פעמים כך ש-k קבוע ולכן נקבל כי הזמן ריצה הינו $O(nk) = O(n)$

Data structures - summary

<u>הערה חשובה:</u> נשים לב כי זמן ריצה צפוי של select ב-

RandomizedPartition הינו $\theta(n)$ צפוי.

ובאמצעות בחירה חכמה של *choosePivot* נקבל $\theta(n)$ במקרה הגרוע ביותר.

<u>תרגיל 4</u> : הוכחה על מיון מערך בצורה רקורסיבית. אלגוריתם המיון הינו 2/3 על תחילת המערך ואז מיון 2/3 על סופו של המערך ואז שוב על תחילתו.
הפתרון הוכחה באינדוקציה. יש להסתכל ולהבין לעומק.

תרגול 10 - מיוניים בזמן לינארי

<u>שאלת חימום</u> : מציאת אבות קדמוניים משותפים.
עקרון השאלה הוא ביצוע חישוב מקדים (פעם אחת שייקח זמן ריצה ארוך) אך יהיה שימושי לפעמים נוספות ורבות.
נרצה להשתמש בשימוש מקדים שיש לנו חזרתיות רבה על data מסוים. (במקרה הנ״ל שמירה במטריצה את הערכים, ובכך כל פעם שנדרוש נוכל לראות במערך עבור $\theta(1)$.

<u>שאלה 1</u> : מערך בעל n מספרים בקלט מסוים – ע״י ביצוע מקדים חיפוש האיברים בטווח [a,b] ב- $\theta(1)$.
שימוש ב- counting sort ושמירת מערך העזר c.
הדגש המרכזי – בהינתן מערך בטווח מסוים יש לחשוב על שימוש בחיפוש לינארי זה.

<u>שאלה 2</u> : מיון מילים לפי סדר לקסיקוגרפי בזמן $O(n)$.
הדגש המרכזי – ריפוד ביטים ואותיות ושימוש ב-Radix וביצוע רדוקציה לבעייה. (הקבלת כל אות למספר).

<u>הערה</u> : ביצוע מיון דליים מוגדר להיות בין [0,1) ניתן להשתמש בנוסחה הבאה בטווח $[a, b)$. $d = \frac{b-a}{n}$

Data structures - summary

שאלה 3 : בהינתן מערך בגודל n של מספרים שלמים בטווח $[0, n^3]$. אלגוריתם למיון יעיל.

הדגש בשאלה זו שניתן להבחין שמיון השוואות במקרה זה יקח לנו $O(n\log n)$ בניגוד לcounting sort אשר יקח לנו $O(n^3)$ כיוון שה-k אינו קבוע.

שימוש ב- Radix עבור k=n ו d=4 נוכל לייצג כל מספר בבסיס של $\log_{10} x$ ובכך לבצע לכל מספר ריפוד לאורכו של המספר המקסימלי ולבצע השוואות בין המספרים עבור כל חלק (סה״כ ישנם 4 חלקים) ולכן נקבל $O(4 * 2n) = O(n)$.

שאלה 4 : נתונות m קבוצות $S_1, S_2 \dots S_m$ כל אחת מהקבוצות מכילה מספריים שלמים בטווח $[1, n]$ ומתקיים כי $m = O(n)$ סכום כל הקבוצות הוא n.

אלגורתים למיון הקבוצות בזמן ומקום של $O(n)$.

שלושה רעיונות :

א. מיון כל קבוצה בנפרד - $O(n\log n)$.

ב. מיון כל קבוצה בנפרד ע״י counting sort ייקח לנו $O(n^2)$

ג. לצרף לכל מערך שדה אשר מייחד אותו. לבצע איחוד של כל המערכים, מיון ע״י counting sort ופיצול. סה״כ נקבל $O(n)$.

הסיבה ששימוש בפונקציה זו נקבל $O(n)$ בניגוד לסעיף ב׳ אשר גם משתמש בפונקציה הנ״ל הוא שאנו חוסכים את המערך c.

הסבר – ניתוח הזמן בשיטה זו הינו $O(n + k)$ כך ש-k הינו גודל המערך של c.

כאשר אנו מבצעים את הפעולה הזו עבור m מערכים כך ש-$m \in O(n)$ נקבל זמן ריצה של $\theta(n^2)$ ועבור איחוד המערכים וביצוע המיון נוכל לחסוך את השימוש במערך m, c פעמים ולהשתמש בגודלו רק פעם אחת.

הערה: שימוש במיון דליים מתאפשר רק כאשר הנתונים מתפלגים באופן אחיד. על כן, רק כאשר יש נתון בשאלה המעיד זאת יש לחשוב על שיטה זו.

תרגול 11 – ניתוח לשיעורין וגרפים

חזרה ודוגמאות על ניתוח לשיעורין שראינו בכיתה.

דגשים רלבנטיים מהתרגול:

תרגיל 2 : מימוש תור בעזרת מספר מחסניות בזמן ריצה של $\theta(n)$. המקרה הנאיבי בו כל האיברים במחסנית A נעביר את כל האיברים למחסנית B ואז נשלוף את האיבר הראשון, ונחזיר את שאר האיברים למחסנית A. שיטה זו משפיעה על פעולת ההוצאות להיות ב-$\theta(n^2)$ [ע״י חישוב סדרה חשבונית נגיע לכך].

02.06.22

Data structures - summary

הדרך היותר טובה היא שימוש גם בשתי מחסניות אך לא לבצע את המעבר של כלל האיברים בכל הוצאה אלא בתלות למספר האיברים במחסנית העזר (הסבר מרחיב במצגת).

במצגת מנותחת בעיה בשלושת הדרכים (פוטנציאל, הספירה הזהירה והאסימונים). מומלץ לראות.

תרגיל 1 בגרפים : גרף לא מכוון ולהציע אלגוריתם ליצירת גרף מכוון חסר מעגלים.

הדגש המרכזי בשאלה זו הוא להכניס לראש את השיטה של מספור קודקודים והגדרת קשת לפי מי גדול/קטן יותר.

תרגיל 2 : שאלה חשובה. ניקח שתי קבוצות A ו-B ונבדוק מה המסלול הקצר ביותר מ-A ל-B עפ״י שני ערכי קודקודים מ-A ו-B ונראה מהו המסלול הקצר ביותר שלהם.

הפתרון – להוסיף שני קודקודים, אחד בכל צד, ולחבר קשתות לכל קודקודי A ועבור הקודקוד השני לכל קודקודי B.

לאחר מכן, להריץ BFS ולגלות את המרחק בין שתי הקודקודים ולהחסיר את ערך המרחק ב-2.

הדגש המרכזי מתרגיל זה הוא לחשוב על הוספת קודקודים לגרף כדי לבצע רדוקציה לבעיה שלה אנו יודעים כבר מהו הפתרון.

תרגול 12 – DFS and topological sort

תרגיל 1- הדגש היחיד שניתן לקחת משאלה זו היא ההבנה שבמטריצת ייצוג שכנויות ניתן להיעזר גם בשורות וגם בעמודות בו זמנית על מנת לבצע בדיקה על קשתות הגרף (למשל **"super-sink"**) בזמן של $O(|v|)$.

תרגיל 2- הדגש בתרגיל זה, זו ההבנה שגרף לא מכוון חסר מעגלים בעל n-1 קודקודים מכיל מעגל ולכן זמן הריצה עבור DFS או BFS יהיה בזמן שלו $O(|V|+|E|)=O(V)$.

תרגיל 3- מידול בעיה לגרפים (ניתן לזהות זאת למשל ע״י המשפט בשאלה ״חסר מעגלים״) ביצוע מיון טופולוגי , והתעסקות עם דרגת הכניסה והיציאה של הקודקודים לקבלת מידע על קשתות בגרף.

תרגיל 4- ביצוע מיון טופולוגי וסידור במערך למשל, הוספת קשתות כמו בגרף ושכלול הסכום המקסימלי המצטבר של כל קודקוד (דרך חשובה להוסיף דארגז הכלים).