סיכום מתומצת למבחן במבני נתונים

21 ביוני 2025

מצרף כאן סיכום **מתומצת** למבחן, כלומר לא יהיה כאן פירוט לגבי מימוש המבנים יותר מדי - אלא בעיקר סיבוכיות ורעיון כללי
יש לי סיכום של 70 עמודים שנכתב תוך כדי ההרצאות - מי שמעוניין שישלח הודעה, נראה לי כבד מדי לקרוא 70 עמודים.
הסיכום מחולק לשניים - מבני נתונים, וטכניקות לחישוב סיבוכיות.
אם אתם מוצאים טעויות אשמח לעדכון (:
גיא יער-און

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 $logn! = log1 + log2 + ... + logn = \sum_{k=0}^{n} logk = \Theta(nlogn)$

מבני נתונים

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 ־ $O(1)$)

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 $O(1)$.

מבוא לעצים

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 m עלים יש $m-1$ קודקודיים פנימיים.

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

סריקות: 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.

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

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

עצי B

מדובר על עץ חיפוש מסדר גבוה (דרגת כל קודקוד היא עד m וכן כל קודקוד מדרגה $k\le m$ 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) כאשר נדרוש את התנאים הבאים:

1. $m\ge 3$ + אי זוגי.
2. בעץ מסוג זה עם פרמטר m, דרגת כל קודקוד פנימי פרט לשורש תקיים : $\left[\frac{m}{2}\right]\le degv\le m$
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.

טענה: עץ B הוא עץ מאוזן, כלומר $h=O(logn)$.

הכנסה, מחיקה וחיפוש מתבצעים ב$O(log_m n)$.

עץ $2-3$

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 $O(logn)$.

הכנסה, מחיקה וחיפוש מתבצעים ב$O(logn)$.

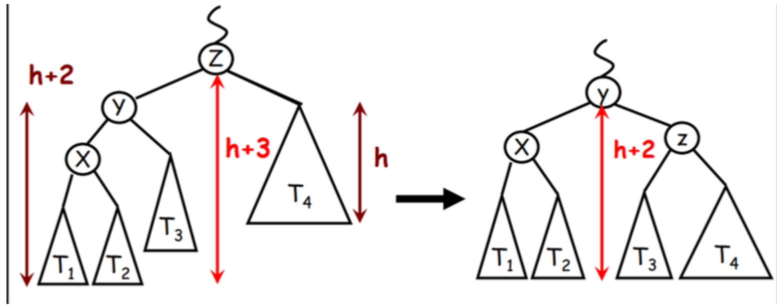
עץ AVL

הגדרה: עץ AVL הוא עץ חיפוש בינארי המקיים כי ההפרש בין הגובה של תת העץ השמאלי לתת העץ הימני הוא $\ni\{1,-1,0\}$. 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. כלומר, גובה העץ הינו $O(logn)$.

הכנסה לעץ: מתכונת עץ החיפוש, נרצה להכניס לעץ בדיוק במקום המתאים לכן נחפש עבור כל צומת האם הערך שלנו גדול או קטן עד שנכניס למיקום הנכון. מי אמר שלאחר ההכנסה העץ נשאר AVL? המקום היחיד למוקשים בגורמי האיזון הינו במסלול ההכנסה, מהמקום שהכנסנו עד לשורש העץ ובשאר המקומות גורמי האיזון לא יוכלו להשתנות. כמו כן, אם מקדם האיזון הופך לאפס אזי כל המקדמים מאליו ישמרו על תכונת AVL. אם האיזון מופר - גובה העץ יהיה ±2. כעת, כאשר קרתה הפרה אנחנו נבצע גלגול של העץ. למשל -



בעזרת תכונת החיפוש, נראה כי נוכל לסדר את איברי העץ בסדר שונה שישמור על האיזון הנדרש. **סה"כ הכנסה תהיה ב**$O(logn)$**.**

חיפוש בעץ: כמו שמחפשים בעץ חיפוש בינארי, בכל פעם משווים את הערך הנוכחי ומתקדמים בהתאם למי שרוצים לחפש. **יקח** $O(logn)$**.**

הוצאה מהעץ: דומה להכנסה. נוציא את x מהעץ כמו בחיפוש בינארי, נתקן את גורמי האיזון ונבצע גלגולים: לכל צומת לאורך המסלול שהחל מהמקום שהוצאנו ועד לשורש נשנה את פרמטר האיזון, אם גורם האיזון משתנה ל±2 נבצע גלגול מתאים. כשנגיע לשורש סיימנו. **המחיקה מהעץ תעלה** $O(logn)$**.**

ערימת מינימום ומקסימום

הקדמה - **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). **גובה העץ של עץ בינארי כמעט שלם יהיה** $O(logn)$**.**

נדון בערימת מינימום. הדיון אודות ערימת מקסימום סימטרי

ערימה היא מבנה נתונים מופשט שיוגדר ע"י עץ בינארי כמעט שלם ויתמוך בפעולות הבאות שיפורטו מטה -

כיצד יראה המימוש? נדרוש עץ בינארי כמעט שלם, וכמו כן נדרוש שהמפתחות של שני הבנים של כל צומת יהיו גדולים מערך הצומת.

א. $Creat-heap()$: אתחול, יוצר ערימה ריקה. $O(1)$.

ב. $Insert(x,Q)$ - הכנסת איבר x לערימה. $O(logn)$. נכניס אותו תמיד ברמה התחתונה מימין, כעת שמרנו על עץ בינארי כמעט שלם. נבדוק בכל שלב האם גדול מאבא שלו. אם כן - טוב לנו, אחרת נפעפע כלפי מעלה ונבצע הכנסות. בשל הפעפוע שתלוי בגובה העץ נגיע ל$O(logn)$.

ג. $min(Q)$- מציאת האיבר עם המפתח הקטן ביותר $O(1)$. תמיד זה יהיה האיבר שבראש הערימה, לכן נחזירו.

ד. $delete-min(Q)$ - הוצאת האיבר עם המפתח הקטן ביותר. $O(logn)$. תמיד נמחק את האיבר שבראש, נחליפו עם האיבר שנמצא ברמה התחתונה מימין, כך שישאר לנו עץ בינארי כמעט שלם. מה שנותר הוא לבצע $heapify$ ולגלגל את הערך שכעת בראש הערימה עד שיגיע למקום המתאים ויקיים את התכונה השנייה של ערימה. סה"כ בשל הגלגול אכן נגיע ל$O(logn)$.

מדובר בתור עדיפויות, לכל אחד ניתן מס' והאיבר שיש לו עדיפות לצאת קודם לפני כולם הוא זה שקיבל את המס' הקטן ביותר

טענה שהוכחנו בהרצאה: בהינתן מערך, ליצור ממנו ערימה יעלה $O(n)$. (זה מפתיע כי ממבט ראשון נראה שזה יעלה $O(nlogn)$)

מיון ערימה: נבנה ערימה מהמערך $O(n)$, נבצע n הוצאות של איבר המינימום שיעלה $O(logn)$. סה"כ נקבל $n+nlogn=O(nlogn)$.

טבלאות גיבוב $(Hash)$

הערה - סליחה מראש על החפירה. כדאי לדעת בערך מה קורה. בפועל במבחן, ממה שאני ראיתי, צריך לדעת שניתן לחפש להכניס ולמחוק ב$O(1)$. אבל שיהיה כאן למקרה וממש יתקטננו איתנו.

אחד ממבני הנתונים השימושיים ביותר כיוון שהוא מאפשר חיפוש, הכנסה ומחיקה ב$O(1)$ **בתוחלת.** נשים לב שכעת אנחנו מדברים על המצב הממוצע, אותה תוחלת מהסתברות. איך נממש?

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

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.

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. איך נפתור?

.1 $Chaining$**:** 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. 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 שa קטן יותר יש פחות סיכוי להתנגשות. תמיד יתקיים כי $E[search] = \Theta(1 + a)$, אסימפטוטית זה אומר $O(1)$.

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 - Universal Hashing .

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 $\frac{1}{m}$. **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 $P(x)=P(y)=\frac{1}{m}$.

סה"כ אם נסכם - נקבל שתוחלת חיפוש הוצאה והכנסה היא $O(1)$.

2. $Open\ Addressing$: 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.

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 כאשר נחפש - אם המפתח לא נמצא בטבלה אז $\Theta(\frac{1}{1-a})$, אם המפתח כן נמצא בטבלה אז $\Theta(\frac{1}{a}ln(\frac{1}{1-a}))$

מס' בדיקות להאם תא פנוי:

א. בדיקה לינארית - נבדוק האם תא תפוס, אם תפוס נלך לאחר אחריו וכן הלאה: $h(k,i)=(h'(k)+i)modm$. 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. 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 $c_2 \neq 0$. זה יתן סיבוכיות טובה יותר מהבדיקה הלינארית.

ג. דאבל האשינג: $h(k,i)=(h_1(k)+ih_2(k))modm$. כאשר h_1 נקראת פונקציית הבסיס וh_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 -

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

כאשר $c>0$ ונרצה שיהיה קטן.

גיבוב קוקייה

טבלת האש זה אחלה אבל אם יש יותר מדי התנגשויות זה נהיה בעייתי. נראה שיטה אחרת שתבטיח חיפוש ב$O(1)$ תמיד. במקרה זה נרצה להשתמש 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$T_2(h_2(x))$. **כיצד נממש?**

1. $Find$:

nruter $T_1(h_1(x))==x$ ro $T_2(h_2(x))==x$

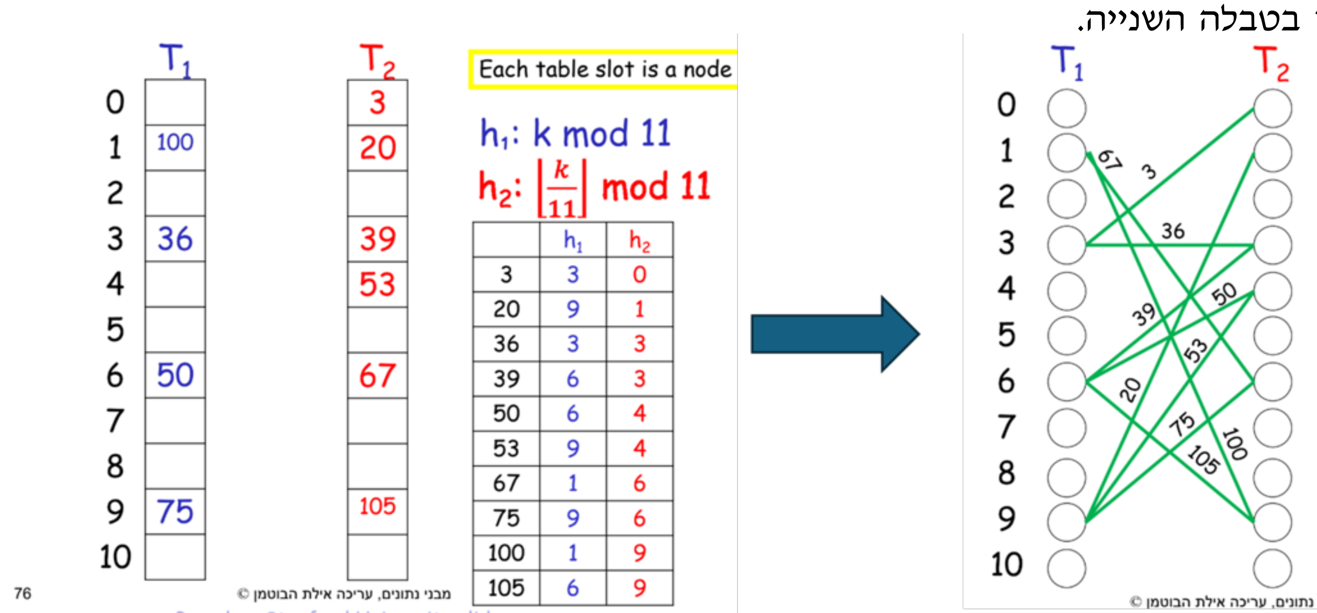
2. $Insert$: אנחנו נתחיל להכניס אל T_1 עד שנתחיל להתקל בהתנגשות. אם התנגשנו, נוציא את האיבר וננסה לחפש לו מקום בטבלה השנייה. כך נעבוד לסירוגין בין שתי הטבלאות. נרצה שיתקיים נעיר כי נרצה: $h_1, h_2 : U \to [m]$ כאשר $m = 4n$ (n= גודל הטבלה) כיוון שזה חשוב להוכחה שתראה לנו שאכן החיפוש ב$O(1)$ שכמובן נחסוך אותה כאן. ככל שהטבלה תהיה צפופה יותר ככה הסיכוי ללולאות גדול יותר, לכן אם $m = 4n$ יש יותר מקום פנוי יחסית. למה לא לבחור $m = 20n$? חבל על הזכרון...

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

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: נסמן בk את אורך המסלול או המעגל.

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 $c>9$ מתקיים

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

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

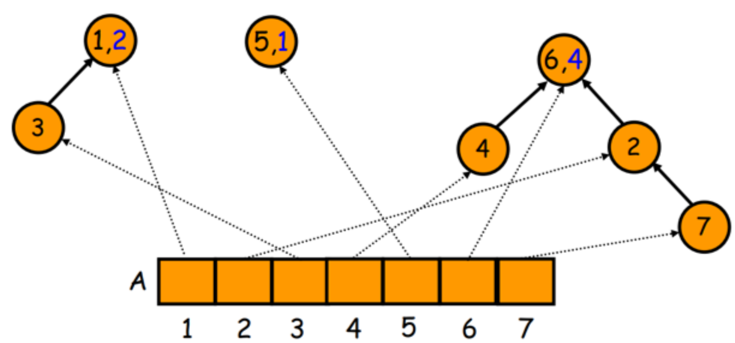
Union Find

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. $O(1)$.

כיצד נממש? 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 בקבוצה ולא אכפת לנו מסדר האיברים. אנחנו נשמור במערך פוינטר לכל איבר: כמו כן נשמור בסגול בראש העץ את מספר האיברים.



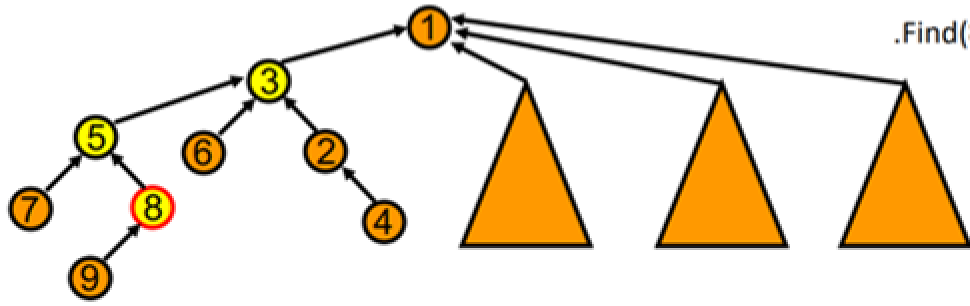
$Find$: נכנסים למערך, הוא מביא אותי לעץ הספציפי,ואז צריך לטפס אל השורש.

$Union$: נתלה את שורש העץ הקטן יותר בגובה תחת העץ הגדול. וכמובן נשנה את גודל הקבוצה.

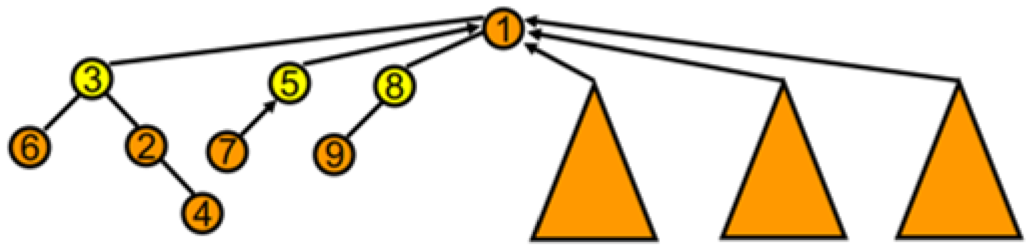
ניתוח סיבוכיות: זמן פעולת $Find$ במקרה הגרוע הוא $O(logn)$. פעולת $Union$ היא $O(1)$. נרצה לשפר - כיווץ מסלולים:

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

דוגמא: Find(8).



אחרי Find(8):



אנחנו למעשה בכל פעם נכווץ את המסלול. נדחוס את המסלול בכל פעם!

נגדיר: log^*n ־ מס' הפעמים שיש להוציא log מ־n כדי להגיע למס' קטן שווה מ־1. למשל ־

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

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

כעת, סיבוכיות פעולת $Find$ תהיה $O(log^*(n))$. (נקח זאת כמובן מאליו ללא הוכחה)

תכנון דינמי

מדובר על בעיות שממבט ראשוני נראות מפחידות, שהפתרון הנאיבי שלהן הוא אקספוננציאלי, אך לאחר שמזהים שיש קריאות חוזרות ושומרים אותן במערך או מטריצה (דינמי) מקבלים סיבוכיות לינארית או פולינומית.

תמיד נעבוד לפי השלבים הבאים:

א. נחשוב מהו הפתרון הנאיבי? לרוב נגיע לכך שצריך לעבור על כל האפשרויות ב־$\Omega(2^n)$, ולעיתים נוכל לראות שהפתרון הוא $O(2^n n)$.

ב. נפתח את האלגוריתם הרקורסיבי, שכולל נוסחה מתמטית עם תנאי עצירה. נסביר רעיונות את הנוסחה, ובמידת הצורך נוכיח אותה. גם כאן אם נריץ ברקורסיה נקבל פתרון אקספוננציאלי.

ג. נעבוד בתכנון דינמי, נחליט אם נשתמש במערך או מטריצה, נתאר את גודל המבנה ומה מכילה יחידה בודדת בתוכו. נסביר כיצד נמלא את המבנה והיכן יימצא הפתרון. במידת הצורך ־ נראה פסודו, ושחזור פתרון. תמיד ננתח את סיבוכיות הזמן שלרוב תהיה $O(n)$ או $O(n^2)$ או $O(n^3)$ ־ פולינומי, ואת המקום. ונציין האם ניתן לצמצם את השימוש בזכרון.

נתבונן בדוגמה הבאה:

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.

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: 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. 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. נשים לב כי ־

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}.

כלומר, נבדוק תמיד את התו האחרון. אם הוא שווה, נגדיל את אורך הסדרה באחד ונלך לבדוק את הקלט עבור $(i-1,j-1)$. אם הם שונים נרצה לבחון את המקסימום (שהרי רוצים תת סדרה ארוכה ביותר) מבין להוריד את התו האחרון בסדרה אחת, לבין השנייה.

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$[1,..j]$.

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

שלב שלישי ־ תכנון דינאמי: נשתמש כמובן במטריצה בגודל $n\times n$. נתחיל למלא את המטריצה מההתחלה לסוף, נראה כי הנוסחה דורשת מאיתנו ברגע נתון לדעת מי מימיני ומי מעליי, לכן אם נמלא בעמודות משמאל לימין בכל רגע נתון יהיה לי את המידע הזה מה שיעזור לי למלא תא ב$O(1)$. לפי האלגוריתם את השורה והעמודה הראשונה נוכל למלא באפסים, ואת השאר בזמן ריצה של $O(1)$ פר תא. אנחנו משתמשים בn^2 תאים ולכן סיבוכיות הזמן והמקום תהיה $O(n^2)$. את סיבוכיות המקום ניתן לצמצם, אם בכל רגע נתון נשמור את העמודה הקודמת ואת העמודה הנוכחית, מה שיוביל לסיבוכיות מקום של $O(n)$. ניתן גם לשחזר ־ נקח טבלה נוספת של "חצים" כאשר חץ כזה $\nwarrow$ ייצג לנו לקיחה של תו, חץ $\leftarrow$ וחץ $\uparrow$ 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 $O(n^2)$.

אלגוריתמים 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:

* **למת הבחירה החמדנית** ־ קיים פתרון אופטימלי שמתחיל בבחירה החמדנית שהאלגוריתם בוחר.

***למת תת המבנה אופטימלי:** לאחר הבחירה החמדנית הראשונה, הבעיה מצטמצמת למציאת הפתרון אופטימלי לבעיה שמתיישבת עם הבחירה הראשונה. כלומר, בחירה חמדנית+ פתרון אופטימלי הוא יתן פתרון אופטימלי.

דוגמה:

מכונית מתוצרת אלגו־קאר נוסעת בהנעה חשמלית. כאשר הסוללה טעונה במלואה, המכונית יכולה לנסוע בדיוק m קילומטרים עד שהסוללה תתרוקן. אנו מתכננים לצאת לנסיעה על כביש

ארוך המתחיל בנקודה A ונגמר בנקודה B שפרוסות לאורכו n תחנות טעינה. התחנות ממוקמות במיקומים $p_1 < p_2 < ..... < p_n$ כאשר p_i הוא המרחק של תחתנת הטעינה ה־i מהנקודה A (וכן מתקיים $A < p_1$ וכן $B > p_n$) מובטח לנו כי לכל $1 \leq i \leq n-1$ מתקיים $p_{i+1} - p_i \leq m$. הצע אלגוריתם חמדני הממזער את מספר העצירות של הרכב לצרכי טעינה, והוכח את נכונותו.

פתרון: האינטואיציה תהיה פשוטה ־ בכל שלב נבחר את התחנה הרחוקה ביותר שניתן להגיע אליה בהתחשב במיכל הדלק הנוכחי. נאתחל רשימה ריקה $places$ ואת המיקום הנוכחי שלנו $spot = A$. בכל פעם נבדוק מיהי התחנה האחרונה והרחוקה ביותר p_i שניתן להגיע אליה מהמיקום הנוכחי. כלומר תחנה i כך ש $p_{i+1} - spot \geq m$ וכן $p_i - spot \leq m$, נכניס את p_i לרשימה של $places$ בה אנחנו נבקר ונעדכן את המיקום $spot$ להיות p_i. בסוף, כאשר $p_i = B$ נחזיר את $places$.

מדוע זה מבטיח פתרון?

למת הבחירה החמדנית ־ קיים פתרון אופטימלי לבעיית תחנות הטעינה , בו בוחרים את התחנה האחרונה ביותר שניתן להגיע אליה מA.

הוכחה ־ נסמן p_i את התחנה האחרונה ביותר שניתן להגיע אליה מA, נסתכל על פתרון אופטימלי OPT כלשהו. אם $p_i \in OPT$ הרי סיימנו, שכן הוא בפתרון האופטימלי. אחרת, נסתכל על התחנות הראשונות שיהיו קטנות מp_i בOPT. נסמנן $p_{j_1}, ...p_{j_r}$. בהכרח, לפי הגדרת p, לכל $1 \leq k \leq r$ מתקיים $p_{j_k} < p_i$. נגדיר OPT'. ע"י הוצאת כל p_{j_k}והוספת p_i. כלומר

$$OPT' = (OPT/\{p_{j_1}, ...p_{j_r}) \vee (p_i)$$

כעת נוכיח כי OPT' הינו פתרון חוקי ואופטימלי.

חוקי ־ לכל זוג נק' צמודות בפתרון שנמצאות לאחר p_i לא שינינו כלום וכן לפי ההגדרה $p_i - A \leq m$, והמרחק ביניהם תקין לכן חוקי. כמו כן, נבין בתחנה הראשונה בOPT' שמגיעה לאחר p_i. נקרא לה p_l , מתקיים $p_l - p_i \leq p_l - p_{j_r} \leq m$, לכן פתרון חוקי כמובן.

אופטימלי ־ קיימת לפחות p_j אחת, ומתקיים

$$|OPT'| = |(OPT/\{p_{j_1}, ...p_{j_r}) \vee (p_i)| \leq |OPT| - 1 + 1 = |OPT|$$

כלומר אכן גודל הפתרון קטן שווה מהאופטימלי, ולכן בהכרח אופטימלי בעצמו וחוקי.

תכונת תת המבנה האופטימלי: פתרון שמורכב מבחירה של התחנה האחרונה ביותר שניתן להגיע אליה בתוספת פתרון אופטימלי לבעיות הטעינה עם כל התחנות שנמצאות אחרי התחנה הנ"ל הוא פתרון אופטימלי.

נוכיח. כעת, נניח בשלילה שהפתרון של האלגוריתם A איננו אופטימלי. כלומר קיים B פתרון כך ש$|A| < |B|$. עקב הבחירה החמדנית בהכרח $p_i \in B$. היא הראשונה בדרך. בפרט $A/\{p_i\}$ וכן $B/\{p_i\}$ הם פתרונות לתחנות הטעינה עבור הערכים לאחר p_i.בפרט, $A/\{p_i\}$ פתרון אופטימלי. כלומר

$$|A/\{p_i\}| \leq |B/\{p_i\}| \Longrightarrow |A| - 1 \leq |B| - 1 \Longrightarrow |A| \leq |B|$$

בסתירה ל$|A| > |B|$. סה"כ אכן נקבל פתרון אופטימלי.

זמן הריצה של הבעיה יהיה לינארי שכן מדובר במעבר על הרשימה, לכן $O(n)$.

קוד הופמן:

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.

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.

מיונים

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. 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, ממיינים 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 שאינם מבוססי השוואות

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 01 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 0001 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$O(d*n)$.

טכניקות לחישוב סיבוכיות

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 $O(n)$.

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

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

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

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

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

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

תמיד נצטרך להגדיר פונקציית פוטנציאל בעצמנו. קשה לחשוב עליה. לכן כל עוד לא הכריחו אותנו - נעדיף לעבוד עם הבנק.

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.**
ניתן לפתח באמצעות: עץ רקורסיה, שיטת האב, שיטת איטרציה והוכחה באינדוקציה.
דוגמה 1. מיון מיזוג
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.

נוסחת הנסיגה - $T(n) = 2T(\frac{n}{2}) + \Theta(n) = 2T(\frac{n}{2}) + cn$, ננחש כי $T(n) = O(nlogn)$ ונוכיח זאת באינדוקציה.

דוגמה 2. כפל מספרים

נתונות שני מחרוזות בעלות n ביטים כל אחת. נרצה לכפול את המחרוזות. מה סיבוכיות הפעולה?

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. (כלומר - בהינתן 4321. 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 -

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

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 - $T(n) = 3T(\frac{n}{2}) + \Theta(n)$, נפתור לפי מאסטר ונקבל כי $T(n) = \Theta(n^{log3}) \approx \Theta(n^{1.58})$.

טענה: חסם תחתון למיון

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

הוכחה (לא הכי פורמלית): לכל מיון מבוסס השוואות ניתן להציג עץ החלטות שייצג את השאלות שהתבצעו לאורך האלגוריתם. לכל סוג מיון יש עץ כזה. אזי, יהי מיון S שמיוצג ע"י עץ החלטה T. הקודקודים הפנימיים של העץ מייצגים את השאלות שהתבצעו, ועלי T את סידור האיברים. בהינתן $|T| = n$, יש $n!$ פרמוטציות לסידור האיברים. בכל שלב יש 2 אפשרויות בלבד לכן עץ הכנסה הוא עץ בינארי, ותמיד כל העלים יופיעו באותה רמה. סה"כ מדובר בעץ מלא. בעץ מלא, מס' הקודקודים הפנימיים הוא $m - 1$ כאשר m הוא מס' העלים. ממה שראינו, זה גורר $n! - 1$ קודקודים פנימיים. סה"כ בעץ יש $2n! - 1$ קודקודים, מכאן שגובה העץ יהיה $\Omega(log(2n! - 1)) = \Omega(log(n!)) = \Omega(nlogn)$, כנדרש.

אלגוריתם סלקט

המטרה, תהיה מציאת האיבר הi בגודלו. אפשר לעשות זאת ב$O(n)$!
נשתמש בעקרון החלוקה. נקח איבר ראשון שיהיה $pivot$ כציר לפיו יתנהל המערך. כלומר, מימינו כל מי שגדול ממנו ומשמאל מי שקטן. תמיד נרצה "להעיף" מהמערך חלק קבוע. כלומר, לצמצם לפעם הבאה חלק שבר n־י. נרצה $pivot$ שיעשה זאת. האלגוריתם יעבור כך:
א. אם $n < 5$ אזי תחזיר את האיברים של A ותמיים ב$O(1)$.
ב. תחלק את המערך לקבוצות של 5. תמצא את החציון של כל אחת מהקבוצות האלו ע"י מיון של $O(5) = O(1)$ בכל קבוצה. כמה חציונים כאלו יש? $\frac{n}{5}$.
ג. נקרא למערך החציונים B. נבצע סלקט על $select(B, \lceil\frac{n}{10}\rceil, \lceil\frac{n}{5}\rceil)$.
כעת אנחנו מוצאים את חציון החציונים, נקרא לו x. לאחר שסידרנו מצאנו שיש כ $\frac{1}{2} * \frac{n}{5} = \frac{n}{10}$ איברים שקטנים מהחציון x. מדוע זה מקדם אותנו? במקור האיברים סודרו בחמישיות, אם החציון שלהם קטן מאיקס גם כל השאר בתוך החמישיות קטן מאיקס (אלו שמשמאל לחציון). בכל חמישייה, יש 3 איברים שלא רלוונטים עוד, סה"כ $\frac{1}{2} * \frac{3}{5}$ מהאיברים, שזה $\frac{3}{10}n$ מהאיברים, לא רלוונטים עוד. כעת יש $\frac{7n}{10}$ איברים שגדולים מחציון החציונים ו$\frac{3n}{10}$ שקטנים מחציון החציונים.
ד. נבצע חלוקה לפי x. נסמן $k = rank(x)$. אם $i = k$ אזי נחזיר את x. אם $i < k$ נבצע רקורסיבית את הקריאה על i האיברים הקטנים בחלק הקטן יותר. אחרת, כלומר $i > k$ אזי נבצע רקורסיבית את הקריאה על $i - k$ האיברים הקטנים בחלק הגדול.

נקבל כי $T(n) := \left\{ \begin{array}{cc} O(1) & n < 50 \\ T(\frac{n}{5}) + T(\frac{7n}{10}) & n \geq 50 \end{array} \right\}$, קל להוכיח באינדוקציה ש$T(n) = O(n)$.

הערה ־ לא ניתן לבצע עם שלשות במקום חמישיות. אפשר רק מס' אי זוגי בשביל לקבל חציון. מיון שלשה יוביל ל$O(nlogn)$. ניתן לעשות שביעיות, תשיעיות וכו'. נשים לב שהקבוע יגדל.