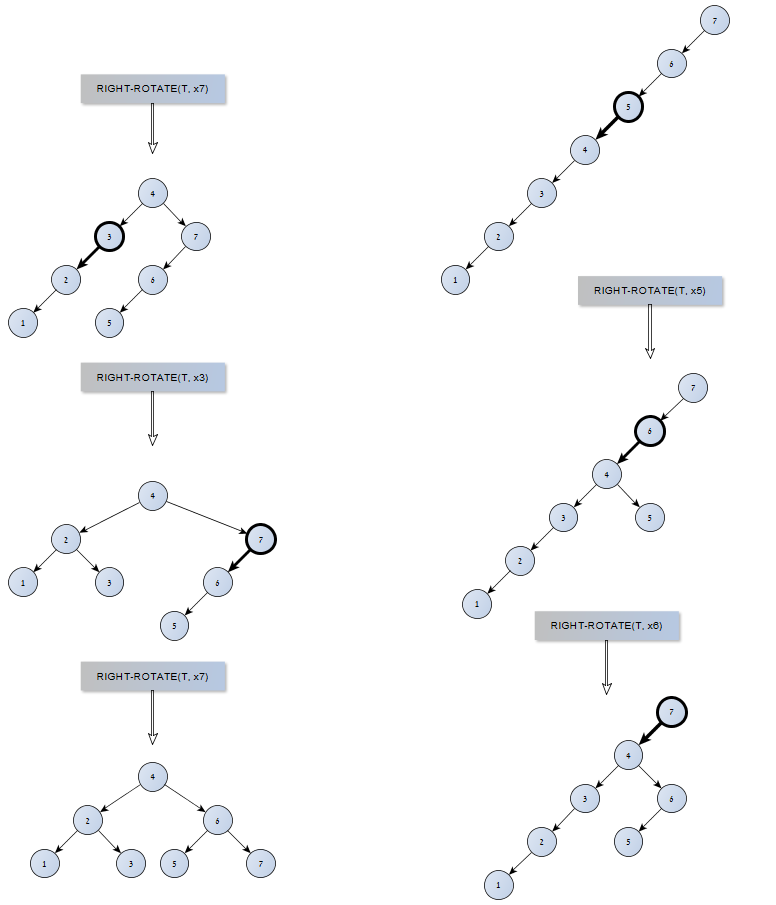
שאלה 1

א. יהי T עח"ב בצורת שרשרת שמאלית עם המפתחות 1,..,7, ויהיו מצביעים לצמתים שמכילים את המפתחות 1,…,7 בהתאמה. סדרת הפעולות הבאה תהפוך את T למאוזן:  
(1) RIGHT-ROTATE(T, x5)  
(2) RIGHT-ROTATE(T, x6)  
(3) RIGHT-ROTATE(T, x7)  
(4) RIGHT-ROTATE(T, x3)  
(5) RIGHT-ROTATE(T, x7)

הסבר: 4 צריך להיות בשורש העץ המאוזן כי הוא אמצע הסדרה אז יש מספר שווה של גדולים ממנו וקטנים ממנו. מפעילים כל פעם RIGHT-ROTATE על p[x4] (זה מקרב את הצומת 4 רמה אחת לשורש) עד שהוא נהיה שורש. כעת בתת העץ הימני שלו יהיו 5,6,7 ובתת העץ השמאלי שלו יהיו 1,2,3.  
בראש תת העץ השמאלי של השורש צריך להיות 2 ובראש תת העץ השמאלי של השורש צריך להיות 6 (אמצע הסדרות). נפעיל RIGHT-ROTATE על p[x2] ועל p[x6] ונקבל את העץ הרצוי.

התהליך מוצג גם בתרשים הבא:



ב. נתאר אלגוריתם המבצע את הפעולה על שרשרת באורך כאשר , כלשהוא.

TREE-BALANCE-NODE(T, topNode, n):  
 middleNode🡨topNode  
 for i🡨0 to   
 middleNode🡨left[middleNode]  
 for i🡨 0 to   
 RIGHT-ROTATE(T, p[middleNode])  
 if n > 3 then  
 TREE-BALANCE-NODE(T, left[middleNode], )  
 TREE-BALANCE-NODE(T, right[middleNode], )

הפעלה: TREE-BALANCE-NODE(T, root[T], n)

הסבר: השגרה מקבלת מצביע לראש של שרשרת שמאלית של מספרים עוקבים (יורדים) באורך n.  
היא מוצאת את הצומת האמצעי ע"י ירידה שמאלה פעמים, ולאחר מכן מעלה אותו לראש ע"י פעולות סיבוב ימני על אביו (כל פעולה מעלה אותו רמה אחת). כעת יש בראש צומת עם המספר האמצעי, בתת העץ השמאלי ובתת העץ השמאלי יש שרשראות שמאליות באורך ולכן מפעילים את השגרה באופן רקורסיבי על הבנים (אלא אם ואז ואין עוד מה לאזן).

שאלה 2

ניתן לבצע את הנדרש באמצעות האלגוריתם הבא:

FIX-RB-AFTER-SUBTREE-REMOVED(T, y ,z):  
 if y=root[T] then  
 root[T]🡨z  
 else if y=left[p[y]] then  
 left[p[y]]🡨z  
 else  
 right[p[y]]🡨z  
 p[z]🡨p[y]  
 color[z]🡨BLACK  
 RB-INSERT(T, y)

הסבר: (z הוא האח של הצומת x שהוסר, y הוא האבא שלהם)  
מסירים את y מהעץ ע"י קישור בין z לאביו של y (לפי שלושה מקרים: y שורש, y בן שמאלי, y בן ימני) וצובעים את y באדום. כעת יש לנו עץ אדום שחור (נוכיח בהמשך) שמכיל את הצמתים הרצויים למעט y. בשלב האחרון מוסיפים את y חזרה לעץ באמצעות שגרת הכנסה רגילה לעץ א"ש. מכאן שהעץ מכיל את הצמתים שרצינו וגם הוא עץ א"ש.

נוכיח שאם בוצע האלגוריתם עד לפני השורה האחרונה, T מקיים את תכונת האדום־שחור:  
1. ברור  
2. אם y אינו השורש, לא שינינו את השורש (אלא רק צמתים נמוכים ממנו) ולכן השורש נשאר שחור. אם y הוא השורש אז הוא מוחלף ב־z, ו־z נצבע שחור כך שעדיין השורש שחור.  
3. לא שינינו את הצבע של אף עלה NIL.  
4. חיברנו בן שונה רק ל־p[y] וצבענו אותו בשחור. אם p[y] שחור הטענה מתקיימת באופן ריק. אם p[y] אדום אז הבן השני שלו (זה ששונה מ־y) הוא שחור בגלל ש־p[y] קיים את תכונה 4 לפני ביצוע האלגוריתם, והבן z שלו (שחיברנו במהלך האלגוריתם) הוא שחור כי ככה צבענו אותו.  
5. כל מסלול פשוט לעלה שלא שייך לתת העץ המושרש ב־z לא השתנה, וכל מסלול פשוט לעלה שכן שייך לתת העץ המושרש ב־z מכיל אותו מספר של צמתים שחורים כי מחקנו בו צומת שחור, והפכנו צומת אחד מאדום לשחור. לכן מכל צומת, כל מסלול פשוט לעלה מכיל את אותו מספר של צמתים שחורים, בפרט זה אותו מספר שהיה לפני ביצוע האלגוריתם.

זמן ריצה: החלק הראשון של האלגוריתם, לפני השורה האחרונה, רץ בזמן קבוע. השורה האחרונה רצה בזמן כאשר הוא מספר האיברים שנותרו בעץ.

שאלה 3

מבנה הנתונים שלנו יהיה מורכב משני עצים א"ש:

- עץ הערכים. המפתחות בעץ זה הם המספרים שהוכנסו למבנה. בכל צומת יהיה גם מצביע לצומת המקביל בעץ .

- עץ הוותק. המפתחות בעץ זה יהיו מספרים עוקבים: בהכנסה הראשונה יכניסו 1, אח"כ 2 וכך הלאה, כך ש־m האיברים הוותיקים ביותר יהיו גם m האיברים בעלי המפתחות הנמוכים ביותר.  
צמתי העץ יחזיק את המידע הנוסף הבא:  
- שדה שערכו מספר הצמתים בתת העץ המושרש בצומת זה, לא כולל עלים nil (כלומר, זה עץ ערכי מיקום)  
- שדה שערכו המפתח של הצומת המקביל בעץ ועוד סכום הערך של שדה של הבנים של הצומת אם יש.  
- שדה מצביע לצומת המקביל בעץ .

הערה: ניתן לחשב את השדות size, sum של צומת ע"פ ערכי שדות אלה על שני בניו, לכן זמן הביצוע של פעולות ההכנסה והמחיקה נשאר (משפט 14.1).

פעולות:

INSERT(S, k): יוצרים צומת חדש של העץ עם המפתח k וצומת חדש של העץ עם שדה sum=k. מעדכנים את המצביעים ההדדיים בין הצמתים ומכניסים כל אחד לעץ המתאים. בהכנסה לעץ מבצעים שינויים: ראשית, לפני ביצוע RB-INSERT-FIXUP מחשבים ומעדכנים את הערכים size, sum בצומת החדש ובכל אחד מהצמתים שמעליו. שנית, אחרי ביצוע LEFT/RIGHT-ROTATE, מחשבים מחדש את size, sum. כל אחד מהחישובים האלה לוקח זמן קבוע ומתבצע לכל היותר פעמים, ושגרות ההכנסה לוקחות פעמים לכן הזמן הריצה במקרה הגרוע הוא .

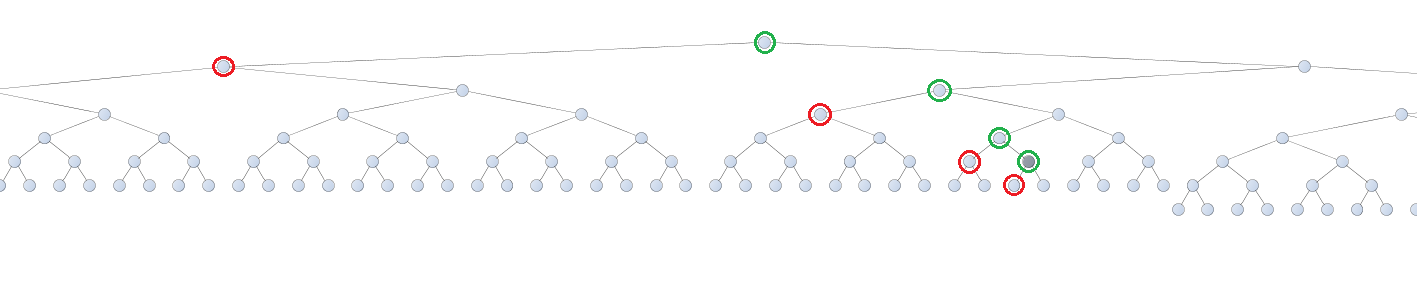
DELETE-MIN(S): מוצאים את האיבר המינימלי בעץ ואת המקביל לו בעץ ומוחקים את שניהם. לאחר המחיקה בעץ מעדכנים את השדות size, sum של הצמתים שמעליו עד לשורש (שינוי ב־RB-DELETE וב־LEFT/RIGHT-ROTATE). זמן ריצה

DELETE-OLD(S): מוצאים את האיבר המינימלי בעץ ואת המקביל לו בעץ ומוחקים את שניהם. עדכונים לשדות כמו שמופרט ב־DELETE-MIN. זמן ריצה

SUM-OLD(S, m): מוצאים את האיבר ה־m הקטן ביותר בעץ - זהו האיבר ה־m הוותיק ביותר. ניתן לבצע את זה בזמן כי זהו עץ ערכי מיקום). לאחר מכן מסכמים את ערכי האיבר ה־m והוותיקים ממנו ע"י אלגוריתם כזה: (מעבירים לו מצביע לאיבר ואת )

SUM-UP-OP(T, x):  
 sum🡨0  
 y🡨right[x]  
 while x≠nil[T] do  
 if right[x]=y then  
 sum🡨sum+key[otherTree[[x]]]  
 if left[x]≠nil[T] then  
 sum🡨sum+sum[left[x]]  
 y🡨x  
 x🡨p[x]  
 return sum

הסבר: כדי למצוא את הסכום של כל האיברים שקטנים מהערך המקביל ל־x, נתבונן במסלול מצומת x לשורש. בכל צומת במסלול, אם בוצעה פניה שמאלה, נוסיף לסכום את המפתח של התא המקביל בעץ , וכן את ערך השדה sum של הבן השמאלי שלו, אם יש.  
הדגמה:

אם רוצים את סכום האיברים הקטנים מהאיבר האפור הכהה, יש לסכם את המפתחות של הצמתים המקבילים לצמתים הירוקים מהעץ וכן את השדות sum של הצמתים האדומים.  
זמן ריצה: מציאת האיבר ה־m הקטן ביותר - . השגרה לסיכום האיברים הקטנים ממנו במבנה עולה ממנו עד לשורש לכן גם היא רצה ב־. לכן זמן הריצה הוא

שאלה 4

מבנה הנתונים שלנו יהיה מורכב משני עצים א"ש:

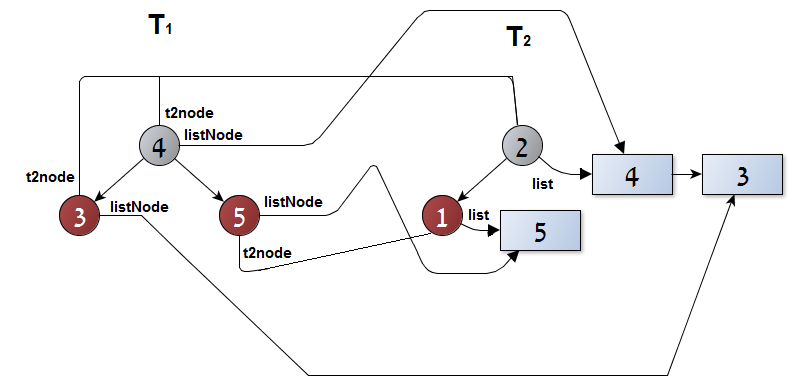
עץ המכיל את המפתחות k היחודיים במבנה. בנוסף בכל צומת יהיה שדה בשם listNode שיהיה מצביע לרשימה דו מקושרת ושדה t2node שיהיה מצביע לצומת המתאים בעץ .

עץ המכיל את הכמויות היחודיות של המפתחות במבנה. המפתח של כל צומת יהיה הכמות, ובנוסף שדה list יכיל מצביע לרשימה דו מקושרת שערכיה - המפתחות במבנה בעלי כמות זו.

בנוסף, המבנה ישמור את המידע הבא:

- שדה item\_num שיכיל את מספר המפתחות במבנה (יאותחל ל־0 ביצירת המבנה)  
- שדה mode שיכיל מצביע לצומת המקסימלי בעץ   
- שדה last\_freq שיכיל את השכיחות שחושבה בעת ההכנסה האחרונה.

דוגמה: אם למבנה הוכנסו המפתחות 4 ואז 4 ואז 5 ואז 3 ואז 3, העצים ייראו כך:



פעולות: ( מספר המפתחות היחודיים - צמתים בעץ )

INSERT(S, k): מגדילים את item\_num ב־1.  
מבצעים SEARCH(T, k) בעץ .  
אם מוצאים צומת x עם המפתח k, מציבים last\_freq🡨(key[t2node[x]]+1)/item\_num. מסירים את הצומת listNode מהרשימה (אם הרשימה התרוקנה, מסירים אותה ואת הצומת t2node כולו). לאחר מכן מכניסים בעץ צומת חדש (או ערך חדש ברשימה בצומת קיים) עם כמות גדולה ב1 ממה שהסרנו.  
אם לא, יוצרים צומת עם מפתח k, מקשרים לצומת בעל ערך 1 בעץ (יוצרים כזה אם צריך).  
מכניסים לשדה mode את המקסימום בעץ .  
זמן ריצה (זמן של SEARCH/DELETE/INSERT) הערה: מספר הצמתים בעץ קטן או שווה למספר הצמתים בעץ .

DELETE(S, k): מקטינים את item\_num ב־1. מחפשים בעץ את הצומת שמפתחו k. אם המפתח המתאים בעץ הוא 1, מוחקים את הצומת לחלוטין (ואת הצומת ברשימה בעץ )., אם לא, מסירים את הצומת מהרשימה בעץ ומכניסים מחדש עם ערך נמוך ב־1.  
מכניסים לשדה mode את המקסימום בעץ .  
זמן ריצה (זמן של SEARCH/DELETE)

INCREASE(S, k, Δ): מוחקים מהעצים ומכניסים מחדש עם הכמות k+Δ.  
מכניסים לשדה mode את המקסימום בעץ .  
זמן ריצה (SEARCH/DELETE/INSERT)

LAST-FREQ(S): מחזירים את last\_freq. זמן ריצה

MODE(S): מחזירים את האיבר הראשון ברשימה בצומת שהשדה mode מצביע אליו. זמן ריצה .