שאלה 1

נחשב את ערך הגיבוב לכל מפתח:

|  |  |
| --- | --- |
| k | k mod 9 |
| 5 | 5 |
| 28 | 1 |
| 19 | 1 |
| 15 | 6 |
| 20 | 2 |
| 33 | 6 |
| 12 | 3 |
| 17 | 8 |
| 10 | 1 |

נכניס את המפתחות לטבלת הגיבוב לפי הסדר:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | |
| / | | / | | / | | / | | / | | ↓ | | / | | / | | / | |
|  |  |  |  |  |  |  |  |  |  | 5 | / |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | |
| / | | ↓ | | / | | / | | / | | ↓ | | / | | / | | / | |
|  |  | 28 | / |  |  |  |  |  |  | 5 | / |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | |
| / | | ↓ | | / | | / | | / | | ↓ | | / | | / | | / | |
|  |  | 19 | ↓ |  |  |  |  |  |  | 5 | / |  |  |  |  |  |  |
|  |  | 28 | / |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | |
| / | | ↓ | | / | | / | | / | | ↓ | | ↓ | | / | | / | |
|  |  | 19 | ↓ |  |  |  |  |  |  | 5 | / | 15 | / |  |  |  |  |
|  |  | 28 | / |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | |
| / | | ↓ | | ↓ | | / | | / | | ↓ | | ↓ | | / | | / | |
|  |  | 19 | ↓ | 20 | / |  |  |  |  | 5 | / | 15 | / |  |  |  |  |
|  |  | 28 | / |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | |
| / | | ↓ | | ↓ | | / | | / | | ↓ | | ↓ | | / | | / | |
|  |  | 19 | ↓ | 20 | / |  |  |  |  | 5 | / | 33 | ↓ |  |  |  |  |
|  |  | 28 | / |  |  |  |  |  |  |  |  | 15 | / |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | |
| / | | ↓ | | ↓ | | ↓ | | / | | ↓ | | ↓ | | / | | / | |
|  |  | 19 | ↓ | 20 | / | 12 | / |  |  | 5 | / | 33 | ↓ |  |  |  |  |
|  |  | 28 | / |  |  |  |  |  |  |  |  | 15 | / |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | |
| / | | ↓ | | ↓ | | ↓ | | / | | ↓ | | ↓ | | / | | ↓ | |
|  |  | 19 | ↓ | 20 | / | 12 | / |  |  | 5 | / | 33 | ↓ |  |  | 17 | / |
|  |  | 28 | / |  |  |  |  |  |  |  |  | 15 | / |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | |
| / | | ↓ | | ↓ | | ↓ | | / | | ↓ | | ↓ | | / | | ↓ | |
|  |  | 10 | ↓ | 20 | / | 12 | / |  |  | 5 | / | 33 | ↓ |  |  | 17 | / |
|  |  | 19 | ↓ |  |  |  |  |  |  |  |  | 15 | / |  |  |  |  |
|  |  | 28 | / |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

שאלה 2

א.

נחשב את האינדקסים שמחזיר האלגוריתם הנתון בשאלה עבור מפתח לפי הסדר:  
בניסיון הראשון , מחפשים בתא  
  
בניסיון השני , מחפשים בתא  
  
בניסיון השלישי , מחפשים בתא  
  
בניסיון הרביעי , מחפשים בתא

בניסיון ה־ (), , מחפשים בתא

זה מתאים בדיוק לסכמת הבדיקה הריבועית עבור , ,

ב.

**נתחיל בלנסח טענת עזר**.

טענה: לכל , הסדרה לכל   
היא תמורה של

נוכיח באינדוקציה.

עבור , הסדרה היא והטענה מתקיימת.

נניח שעבור מסוים, הסדרה לכל היא תמורה של   
נגדיר סדרה לכל   
(\*) בגלל שהם מוגדרים באופן זהה למעט ה־, או לכל

נתבונן בזוג אינדקסים כאשר .  
  
(הופסת כפולה שלמה של )  
  
לפיכך, ההפרש בין ל־ הוא .  
בשילוב עם הטענה (\*) נסיק שאחד מבין שווה והשני שווה   
מכך ש כאשר מייצגים את כל התאים במערך, ו־ כאשר תמורה של (הנחת האינדוקציה), נובע ש־ כאשר מהווה תמורה של . כלומר הטענה נכונה עבור .

בזאת הוכחנו באינדוקציה שלכל , הסדרה לכל   
היא תמורה של .

**נחזור לטענה שיש לפתור בתרגיל**.

נסמן ב־ את פונקציית הגיבוב הנתונה בתרגיל לצורך עקביות עם המשוואה (11.5).  
המקרה הגרוע הוא כאשר מתבצעות בדיקות, ונבדקים התאים  
 עבור   
נתון ש־ הוא חזקה של לכן לפי טענת העזר הסדרה עבור היא תמורה של ומכאן שגם סדרת התאים שנבדקים היא תמורה של כלומר האלגוריתם בודק את כל התאים בטבלה.

שאלה 3

נגדיר (מלשון preorder) - הסדרה המתאימה לסריקה התחילית של עץ חיפוש בינרי בעל מפתחות שונים זה מזה .

טענה: לכל סדרה ועצים המקיימים מתקיים   
הוכחה:  
נוכיח באינדוקציה על אורך הסדרה .  
  
אם ריקה, ברור ש־ עץ ריק  
  
נניח שהטענה נכונה לכל סדרה באורך קטן מ־, והאורך של הוא .  
ערכי השורש של שווים:   
כל האיברים ב־ קטנים מ־ וכל האיברים ב־ גדולים מ־ (בגלל תכונת עץ החיפוש הבינרי והתנאי שהמפתחות שונים זה מזה)  
באותו אופן כל האיברים ב־ קטנים מ־ וכל האיברים ב־ גדולים מ־.  
נסמן ב־ את אינדקס האיבר הראשון הגדול מ־.  
 לכן לפי הנחת האינדוקציה   
 לכן לפי הנחת האינדוקציה   
תת העץ השמאלי, תת העץ הימני וערך השורש שווים, לכן   
מכאן שלכל סדרה שהיא תוצאה של סריקה תחילית על עח"ב בעל מפתחות שונים זה מזה מתאים עץ אחד בדיוק.

ניתן לשחזר את העץ תוך קריאת הסדרה משמאל לימין באמצעות אלגוריתם כגון הבא:

RESTORE-TREE(A)  
 S🡨init stack  
 T🡨init binary tree  
 if length[A]=0 then  
 return T  
 root[T]🡨init binary tree node  
 key[root[T]]🡨A[1]  
 PUSH(S, root[T])  
 for i🡨2 to length[A]  
 x🡨NIL  
 while (not STACK-EMPTY(S)) and A[i] > key[STACK-TOP(S)]  
 x🡨POP(S)  
 if x≠NIL then  
 right[x]🡨init binary tree node  
 key[right[x]]🡨A[i]  
 p[right[x]]🡨x  
 PUSH(S, right[x])  
 else   
 left[STACK-TOP(S)]🡨init binary tree node  
 key[left[STACK-TOP(S)]]🡨A[i]  
 p[left[STACK-TOP(S)]]🡨STACK-TOP(S)  
 PUSH(S, left[STACK-TOP(S))  
 return T

הסבר: האלגוריתם בונה את העץ באותו סדר שבו הוא נסרק כשנבנתה הסדרה. כשהוא מוצא בסדרה ערך קטן יותר מהקודם הוא מוסיף אותו כצומת משמאל ודוחף אותו למחסנית. כשהוא מוצא ערך גדול יותר הוא מטפס במעלה העץ ע"י שליפה מהמחסנית עד שהוא מגיע לצומת שערך אביו גדול מהערך בסדרה, ודוחף צומת חדש מימין.

האם זה נכון עבור סריקה סופית - בהחלט, ההוכחה דומה ו"סימטרית", שורש העץ יהיה האיבר האחרון בסדרה במקום הראשון, האינדקס המפריד בין תת העץ הימני והשמאלי יהיה של האיבר הראשון הקטן מ־ וכו'.

האם זה נכון עבור סריקה תוכית - לא. להלן שני עח"בים בעלי מבנה שונה שלשניהם מתאימה הסדרה לסריקה תוכית:

האם זה נכון עבור סריקה תחילית אם קיימים שני מפתחות זהים - לא. להלן שני עח"בים בעלי מבנה שונה שלשניהם מתאימה הסדרה לסריקה תחילית:

שאלה 4

א

ניתן לבדוק אם הוא מצביע רגיל או חוט כך: אם אז זהו מצביע רגיל לבן, אחרת זהו חוט.

הוכחה:

טענת עזר: לכל צומת בעץ רגיל (לא מחווט), TREE-PREDECESSOR. נימוק: ע"פ האלגוריתם TREE-PREDECESSOR (בהנחה שהוא פועל באופן סימטרי ל־TREE-SUCCESSOR), הצומת המוחזר הוא אב קדמון של או שייך לתת העץ השמאלי של או NIL. בפרט הוא אינו .

אם אז ברור ש־ הוא אכן חוט כי אם הוא היה מצביע לבן רגיל אז היה מתקיים .  
אם אז הוא האבא של (כי לא עשינו שינויים במצביעים אלא רק ב־). כלומר שווה לבן השמאלי או לבן הימני של בעץ המקורי. כיוון שהשינוי היחיד שיכל לקרות ל־ הוא הצבה של TREE-PREDECESSOR אם לא היה ל־ בן שמאלי, וכיוון ש־TREE-PREDECESSOR אינו הבן הימני (טענת עזר), חייב להיות הבן השמאלי המקורי כלומר מצביע רגיל.

ניתן לבדוק אם הוא מצביע רגיל או חוט כך: אם אז זהו מצביע רגיל לבן, אחרת זהו חוט.

הוכחה (סימטרית להוכחה הקודמת):

טענת עזר: לכל צומת בעץ רגיל (לא מחווט), TREE-SUCCESSOR. נימוק: ע"פ האלגוריתם TREE-SUCCESSOR, הצומת המוחזר הוא אב קדמון של או שייך לתת העץ הימני של או NIL. בפרט הוא אינו .

אם אז ברור ש־ הוא אכן חוט כי אם הוא היה מצביע לבן רגיל אז היה מתקיים .

אם אז הוא האבא של (כי לא עשינו שינויים במצביעים אלא רק ב־). כלומר שווה לבן השמאלי או לבן הימני של בעץ המקורי. כיוון שהשינוי היחיד שיכל לקרות ל־ הוא הצבה של TREE-SUCCESSOR אם לא היה ל־ בן ימני, וכיוון ש־TREE-SUCCESSOR אינו הבן השמאלי (טענת עזר), חייב להיות הבן השמאלי המקורי כלומר מצביע רגיל.

ב

הכנסה:

THREADED-TREE-INSERT(T, z):  
 y🡨NIL  
 x🡨root[T]  
 while x≠NIL **AND p[x]=y** do  
 y🡨x  
 if key[z]<key[x] then  
 x🡨left[x]  
 else  
 x🡨right[x]  
 p[z]🡨y  
 if y=NIL then  
 root[T]🡨z  
 else  
 if key[z]<key[y] then  
 **left[z]🡨left[y]**  
 left[y]🡨z  
 **right[z]🡨y**  
 else  
 **right[z]🡨right[y]**  
 right[y]🡨z  
 **left[z]🡨y**

הסבר כללי: האלגוריתם זהה ל־TREE-INSERT שבספר עם **תופסות מודגשות**. התוספות מטפלות בזיהוי חוטים, ובשימור החוטים במהלך ההכנסה.  
בשלב הראשון מוצאים את המקום להכניס. הצומת החדש תמיד ייכנס כעלה במקום ריק בעץ. ריק יכול להיות מצביע left/right שהוא NIL, או מצביע left/right שהוא חוט (במקרה של עץ ריק, הצומת ייכנס כשורש).  
אם מכניסים צומת לעץ ריק, אין יהיו SUCCESSOR ו־PREDECESSOR לכן שדות left ו־right שלו יישארו NIL וזה תקין.  
אם מכניסים צומת כבן שמאלי אז ה־right שלו יהיה האבא שלו (הצומת שאליו מכניסים), וזה מתלכד עם הערך שהשגרה TREE-SUCCESSOR היתה מחזירה עליו אם העץ היה נטול חוטים. ה־left שלו יהיה ה־left הישן של אבא שלו (ה־left החדש של אבא שלו יהיה כמובן הצומת החדש עצמו).  
באופן סימטרי, אם מכניסים צומת כבן ימני אז ה־left שלו יהיה האבא שלו (הצומת שאליו מכניסים), וזה מתלכד עם הערך שהשגרה TREE-PREDECESSOR היתה מחזירה עליו אם העץ היה נטול חוטים. ה־right שלו יהיה ה־right הישן של אבא שלו (ה־right החדש של אבא שלו יהיה כמובן הצומת החדש עצמו).

נכונות:  
מספר הפעמים שלולאת ה־while מתבצעת הוא לכל היותר כגובה העץ כי x מתחיל בשורש ומתקדם רמה אחת כלפי מטה בכל איטרציה עד שהוא נהיה NIL או חוט, מכאן שהאלגוריתם מסתיים.  
נוכיח שאם T היה עח"ב עם חוטים, אז לאחר ביצוע THREADED-TREE-INSERT נוסף לו הצומת z הוא נשאר עח"ב עם חוטים.

הלולאה באלגוריתם מקיימת את הטענה הבאה: בתחילת כל איטרציה, p[x]=y ובנוסף, לכל שני צמתים a,b שונים מ־NIL ש־y עבר דרכם במהלך האלגוריתם, אם b=left[a] אז key[z]<key[a] ואם b=right[a] אז key[z]>=key[a].

יהיו w, x שני צמתים ב־T אחרי ביצוע האלגוריתם כך ש־x שייך לתת העץ השמאלי של w.  
אם x≠z אז key[x]≤key[w] כי העץ קיים את תכונת עץ החיפוש הבינרי לפני ביצוע האלגוריתם, וכל השינוי שלנו היה הוספת עלה חדש.  
אם x=z זה אומר ש־w נמצא ע"ג המסלול שהאלגוריתם בדק. x שייך לתת העץ השמאלי של w כלומר התבצעה פניה שמאלה ולכן key[x]=key[z]<=key[w].

יהיו w, y שני צמתים ב־T אחרי ביצוע האלגוריתם כך ש־y שייך לתת העץ הימני של w.  
אם y≠z אז key[w]<=key[y] כי העץ קיים את תכונת עץ החיפוש הבינרי לפני יצוע האלגוריתם, וכל השינוי שלנו היה הוספת עלה חדש.  
אם y=z זה אומר ש־w נמצא ע"ג המסלול שהאלגוריתם בדק. Y שייך לתת העץ הימני של w כלומר התבצעה פניה ימינה לכן key[w]<=key[z]=key[y].

משתי ההוכחות האחרונות נובע שהעץ לאחר השינוי מקיים את תכונת עץ החיפוש הבינרי.

אם מכניסים את z לעץ ריק, left, right יישארו ריקים וזה גם הערכים שהיו מחזירות עליו השגרות TREE-PREDECESSOR, TREE-SUCCESSOR.

אם מכניסים את z כתת עץ שמאלי, ברור מתוך המימוש של TREE-SUCCESSOR (על עח"ב בלי חוטים) שהיא היתה מחזירה p[z] כי ל־z אין בן ימני, ו־z הוא בן שמאלי של אביו ולא ימני. לכן ההשמה ל־right[z] נכונה.  
כמו כן ל־p[z] לא היה בן שמאלי לפני ההכנסה, לכן הלולאה בשגרה TREE-PREDECESSOR על p[z] היתה "עולה ימינה" בעץ כל עוד ניתן ומחזירה את הצומת הראשון שהפניה אליו היא שמאלה (או NIL). ל־z אין בן שמאלי לכן והוא עצמו בן שמאלי ולכן ה־PREDECESSOR שלו הוא בדיוק הערך הישן של left[p[z]] ולכן ההשמה ל־left[z] נכונה.

אם מכניסים את z כתת עץ ימני, אז גם במקרה זה הערכים של left[z], right[z] יהיו נכונים מנימוקים סימטריים.

מסקנה: האלגוריתם קובע את החוטים עבור הצומת החדש z כנדרש.  
האם החוטים של שאר הצמתים נשארים תקינים? ה־PREDECESSOR של z כש־z בן שמאלי הוא צומת שיש לו בן ימני (או NIL), לכן חוט ימני לא רלוונטי לגביו. ה־PREDECESSOR של z כש־z בן ימני הוא אביו וגם כאן לא רלוונטי חוט שמאלי. באופן סימטרי ל־SUCCESSOR של z אין חוט שמאלי (יש לו בן אמיתי או NIL). אין שינוי בשאר ה־PREDECESSOR וה־SUCCESSOR שלא שאר האיברים, ומכאן שאין צורך לשנות את החוטים שלהם והם נשארים תקינים.

זמן ריצה: כאשר h - גובה העץ. h הוא מספר הפעמים שמתבצעת הלולאה במקרה הגרוע. שאר הפעולות מתבצעות בזמן קבוע.

מחיקה:

נגדיר שתי שגרות עזר:

THREADED-TREE-PREDECESSOR(x):  
 if left[x]≠NIL and p[left[x]]=x then  
 x🡨left[x]  
 while right[x]≠NIL and p[right[x]]=x  
 x🡨right[x]  
 return x  
 y🡨p[x]  
 while y≠NIL and x=left[y]  
 x🡨y  
 y🡨p[y]  
 return y

THREADED-TREE-SUCCESSOR(x):  
 if right[x]≠NIL and p[right[x]]=x then  
 x🡨right[x]  
 while left[x]≠NIL and p[left[x]]=x  
 x🡨left[x]  
 return x  
 y🡨p[x]  
 while y≠NIL and x=right[y]  
 x🡨y  
 y🡨p[y]  
 return y

הסבר שגרות עזר:  
מטרת השגרות להביא את הצומת הקודם (THREADED-TREE-PREDECESSOR) או העוקב (THREADED-TREE-SUCCESSOR) בעח"ב מחווט. השגרות מבוססות על השגרות בספר עבור עח"ב רגיל (רק TREE-SUCCESSOR נתונה, את PREDECESSOR הסקתי ע"י סימטריה) עם תיקונים עבור התחשבות בחוטים והכנסה של TREE-MINIMUM/TREE-MAXIMUM לתוך הקוד.

הסבר נכונות THREADED-TREE-SUCCESSOR: אם לצומת x יש בן ימני (אמיתי) אז העוקב שלו הצומת השמאלי ביותר בתת העץ הימני שלו. במקרה כזה, התנאי בשורה הראשונה יתקיים, ולמשתנה x יוצב הבן הימני של x. לאחר מכן כל עוד יש בן שמאלי אמיתי ל־x, x יקודם לשם, ולאחר מכן יוחזר. יוצא שהחזרנו את הצומת השמאלי ביותר בתת העץ הימני.  
אם לצומת x אין בן ימני (אמיתי) אז העוקב שלו הוא האב הקדמון הנמוך ביותר של x אשר הבן השמאלי שלו הוא גם קדמון של x (אם אין כזה, אז אין עוקב ל־x). במקרה כזה התנאי בשורה הראשונה לא יתבצע, במקום זה תתבצע לולאת ה־while שבה בתחילת כל איטרציה y=p[x] (בגלל זה לא צריך לבדוק חוטים - אם y=p[x] וגם x=right[y] אז בעצם x הוא בן אמיתי). הלולאה תתקדם במעלה העץ כל עוד הצומת הקודם הוא בן שמאלי, ותיפסק כאשר עברנו את הצומת (y=NIL) או הצומת הקודם הוא בן ימני. בסיום הלולאה הערך של y הוא בדיוק העוקב של x וזה מה שיוחזר.  
נכונות THREADED-TREE-PREDECESSOR מוסברת באופן סימטרי.

זמן ריצה: כל אחת מהשגרות רצה בזמן כאשר גובה העץ. זמן זה מתקבל כאשר צריך לעבור על כל הרמות מעלה שנמצא בתחתית עד השורש.

עכשיו נגדיר את השגרה המבוקשת:

THREADED-TREE-DELETE(T, z):  
 zprev🡨THREADED-TREE-PREDECESSOR(z)  
 zsucc🡨THREADED-TREE-SUCCESSOR(z)  
 if (left[z]≠NIL and p[left[z]]=z) and (right[z]≠NIL and p[right[z]]=z) then  
 ▷ z has two children  
 if p[zsucc]≠z and right[zsucc]≠NIL p[right[zsucc]]=zsucc then  
 p[right[zsucc]]🡨p[zsucc]  
 left[p[zsucc]]🡨right[zsucc]  
 left[zsucc]🡨left[z]  
 p[left[zsucc]]🡨zsucc  
 if zsucc≠right[z] then  
 right[zsucc]🡨right[z]  
 p[right[z]]🡨zsucc  
 p[zsucc]🡨p[z]  
 if p[zsucc]=NIL  
 root[T]🡨zsucc  
 else if left[p[zsucc]]=z then  
 left[p[zsucc]]🡨zsucc  
 else  
 right[p[zsucc]🡨zsucc  
 right[zprev]🡨zsucc  
 else if right[z]≠NIL and p[right[z]]=z then  
 ▷ z has only right child  
 p[right[z]]🡨p[z]  
 if p[z]≠NIL and left[p[z]]=z then ▷ z is left child  
 left[p[z]]🡨right[z]  
 else if p[z]≠NIL and right[p[z]]=z then ▷ z is right child  
 right[p[z]]🡨right[z]  
 else ▷ z is root  
 root[T]🡨right[z]  
 left[zsucc]🡨zprev  
 else if left[z]≠NIL and p[left[z]]=z then  
 ▷ z has only left child  
 p[left[z]]🡨p[z]  
 if p[z]≠NIL and right[p[z]]=z then ▷ z is right child  
 right[p[z]]🡨left[z]  
 if p[z]≠NIL and left[p[z]]=z then ▷ z is left child  
 left[p[z]]🡨left[z]  
 else ▷ z is root  
 root[T]🡨left[z]  
 right[zprev]🡨zsucc  
 else  
 ▷ z has no children  
 if p[z]≠NIL and left[p[z]]=z then ▷ z is left child  
 left[p[z]]🡨zprev  
 if p[z]≠NIL and right[p[z]]=z then ▷ z is right child  
 right[p[z]]🡨zsucc  
 else ▷ z is root  
 root[T]🡨NIL  
 return z

הסבר כללי:  
האלגוריתם מטפל במחיקה בהתאם לאחד מ־4 מקרים שונים:  
1. ל־z יש שני בנים  
2. ל־z יש רק בן ימני  
3. ל־z יש רק בן שמאלי  
4. ל־z אין בנים  
במקרה של שני בנים הוא מטפל ע"י עדכון של אביו ובניו של z להצביע לעוקב של z ולהפך (בשונה מהאלגוריתם בספר שמעתיק את המפתח והמידע הנלווה של העוקב לתוך z ומחזיר את העוקב. האלגוריתם שלי תמיד מחזיר/"מוחק" את z).  
בשאר המקרים הוא מטפל בהסרת z ע"י הפניית האב של z לבנו (אם יש), והפניית בנו (אם יש) לאביו. בנוסף, בכל המקרים האלגוריתם מתקן את החוטים אם צריך.

הסבר נכונות:  
האלגוריתם פותח בחישוב ושמירת הקודם והעוקב של z במשתנים zsucc ו־zprev בהתאמה.  
לאחר מכן הוא מבצע סדרת פעולות ע"פ מצב הבנים של z. נראה עבור כל מצב כזה מדוע סדרת הפעולות מסירה את z ושומרת על תכונת עץ החיפוש הבינרי.

1. ל־z יש שני בנים  
העוקב של z יחליף את z: כל ההצבעות שהיו ל־z ישונו כך שיצביעו לעוקב, וכל המצביעים של z יועתקו לעוקב. כיוון של־z יש בן ימני, העוקב שלו הוא TREE-MINIMUM(right[z]). נטפל באופן שונה בשני מקרים: העוקב הוא הבן הימני, העוקב אינו הבן הימני.  
אם העוקב אינו הבן הימני, ולעוקב יש בן ימני, נחבר את הבן הימני לכבן השמאלי של האבא של העוקב (במקום העוקב). כמובן שפעולה זאת לא מפרה את תכונת עח"ב. הצומת השמאלי ביותר בתת העץ הימני של העוקב מצביע שמאלה לעוקב (חוט), והצומת הימני ביותר בתת העץ הימני של העוקב מצביע ימינה לאביו של העוקב. הצבעות אלה יישארו נכונות גם לאחר ההזזות בעץ ולכן האלגוריתם לא משנה אותן.  
לאחר מכן מעדכנים את המצביע השמאלי של העוקב ואת המצביע לאבא של בנו השמאלי של z.  
אם העוקב אינו הבן הימני של z, מעדכנים את המצביע הימני של העוקב ואת המצביע לאבא של בנו הימני של z. (אם העוקב הוא כן הבן הימני אז אין לעוקב בן שמאלי, וצריך לשמר את בנו הימני).  
לאחר מכן מעדכנים את המצביע ל־z באבא של z אם יש כזה, או את המצביע לשורש אם אין כזה, שיצביע לעוקב.  
לבסוף, מתוקן החוט הימני בקודם של z להיות העוקב של z. (מכך שיש ל־z בן שמאלי, בטוח שיש לו קודם, ובטוח שהקודם הוא הצומת הימני ביותר בתת העץ השמאלי של z. מכאן שהמצביע הימני בקודם של z הוא חוט ל־z שיש לעדכן)

2. ל־z יש רק בן ימני  
הבן הימני של z יקח את מקומו של z. מעדכנים את המצביע לאבא של הבן הימני של z להיות האבא של z.  
לאחר מכן, אם z הוא בן שמאלי, מעדכנים את המצביע השמאלי של אביו להיות בנו של z, אם זה הוא בן ימני, מעדכנים את המצביע הימני של אביו להיות בנו של z, ואם אין לו אב אז מעדכנים את השורש להיות בנו של z.  
הקודם של z, אם קיים, לא נמצא בתת העץ השמאלי של z (כי ל־z אין בן שמאלי), לפיכך הוא אחד מהאבות הקדמונים של z ויש לו בן ימני. לפיכך המצביע הימני שלו אינו חוט, ואינו צריך עדכון.  
העוקב של z (שבטוח קיים) שייך לתת העץ הימני של z ואין לו בן שמאלי, לכן יש לו חוט ל־z. מעדכנים את החוט להצביע על הקודם של z.

3. ל־z יש רק בן שמאלי  
סימטרי לחלוטין למקרה 2

4. ל־z אין בנים  
אם z בן שמאלי, מעדכנים את המצביע השמאלי של אביו להיות חוט לקודם של z. לקודם של z יש בן ימני לכן אין צורך לעדכן את המצביע הימני שלו.  
אם z בן ימני, מעדכנים את המצביע הימני של אביו להיות חוט לעוקב של z. לעוקב של z יש בן שמאלי לכן אין צורך לעדכן את המצביע השמאלי שלו.  
אם ל־z אין אבא, הוא למעשה הצומת היחיד בעץ, והעדכון היחיד שצריך הוא לעדכן את השורש ל־NIL.

זמן ריצה: זמן הריצה של האלגוריתם במקרה הגרוע הוא כאשר גובה העץ בגלל הקריאות ל־THREADED-TREE-PREDECESSOR ו־THREADED-TREE-SUCCESSOR. שאר הפעולות מתבצעות בזמן קבוע.

ג.

היתרון העיקרי הוא שיפור זמן הריצה של סריקה של העץ לפי סדר המפתחות. עם שימוש בחוטים, ניתן למצוא את העוקב כשיש חוט ימני (והקודם כשיש חוט שמאלי) בזמן קבוע. במקרים שיש בן אמיתי עדיין צריך למצוא את העוקב דרך חיפוש הצומת השמאלי ביותר בתת העץ הימני (הימני ביותר בתת העץ השמאלי).