**ממ"ן 16**

**שאלה 1**

**סעיף א'**

ההסתברות לבחור תא אחד מתוך M תאים היא - .

בכל בחירת תא שלנו ההסתברות זהה, ולכן, כדי ליצור שרשרת באורך 4, נצטרך להכניס את ארבעת המפתחות לאותו התא, כלומר ההסתברות שיקרה כך לתא מסויים היא .

אך ישנם M תאים שיכולים לקיים זאת, ולכן ההסתברות הכוללת היא:

**סעיף ב'**

נזכיר שבמיעון פתוח ב M תאים, אם אנחנו רוצים להכניס מפתח לתא תפוס, אז אין שרשרת לקשר אותו לתא, ובמקום זאת נחפש תא אחר במערך כך שנשים אותו שם.

נתאר את המצב שבו הכנסת המפתח K3 תדרוש שלוש בדיקות:

למעשה מצב זה קורה אם ורק אם המפתח מוכנס לתא תפוס, אשר מפנה אותו לתא אחר שתפוס, אשר מפנה אותו לתא אחר שריק, במקרה זה מתבצעים שלוש בדיקות הבודקות האם התא תפוס.

1. ההסתברות לבחור תא אחד מתוך M תאים היא -

2. ההסתברות לבחור תא , כך שהתא הראשון יפנה אותנו לתא הזה היא (מכיוון שיש 2 דרכים 'ליפול' על תא שכזה , אם נבחר את התא בעצמנו, או אם נבחר את התא הראשון שבחרנו והוא יפנה אותנו לתא הזה כי הוא תפוס).

3. ההסתברות לבחור את התא הראשון שוב, על מנת שיתבצעו 3 הבדיקות, זה בדיוק ההסתברות לבחור תא אחד מתוך M תאים והיא - .

כלומר, ההסתברות שיקרה מצב כזה לתא מסויים היא .

אך ישנם M תאים שיכולים לקיים זאת, ולכן ההסתברות הכוללת היא:

**סעיף ג'**

ע"פ משפט 11.6, במיעון פתוח, תוחלת מספר הבדיקות הנערכות בעת חיפוש כושל היא לכל היותר:

**שאלה 2**

שאלה 11-3 בעמוד 211 בספר.

**סעיף א'**

נשים לב כי מתרחש הדבר הבא:



כלומר, קל מאד לראות כי מתקיימת הפונקצית גיבוב:

ע"פ נוסחא 11.5, פונקצית הגיבוב לבדיקה ריבועית היא :

נשווה בין שני הביטויים הללו ונקבל:

נפתור את המשוואה:

וזה כמובן מתקיים כאשר:

וגם

ולכן, סכמה זאת היא מקרה פרטי של סכמת "הבדיקה הריבועית", ומתקיים:

**סעיף ב'**

על מנת להוכיח שהאלגוריתם בודק כל תא בטבלה, אנו בעצם צרכים להראות שהוא לא מגיע לאותו תא פעמיים - כי במצב זה יווצר מעגל והאלגוריתם לא יגיע לכל התאים.

נניח בשלילה שקיימים , כך שמתקיים:

כלומר:

נניח בלי הגבלת הכלליות ש , לכן קיים קבוע c כלשהו כך שמתקיים :

כלומר:

אך מכיוון ש i<j , אז קיים קבוע שלם כך שמתקיים , כלומר:

נזכור ש m הוא חזקה של 2, ולכן נסמן: ( כך ש x קבוע ).

כלומר מתקיים:

הגענו לכך ש הוא כפולה כלשהי של החזקה .

אם זוגי אז אי זוגי ולהפך, ולכן:

א) או ש a זוגי ואז הוא עצמו כפולה של .

ב) או ש זוגי ואז הוא עצמו כפולה של .

אפשרות א' לא הגיונית מכיוון ש בוודאי קטן מ m ( אחרת בסתירה ), ו-

, ובוודאי אם כך ש קטן גם מ 2m ואינו כפולה של .

אפשרות ב' לא הגיונית מכיוון ש:

ואם כך, הגענו לסתירה, כלומר הנחת השלילה שגויה, ולא קיימים כך שמתקיים:

ולכן לא יווצרו 'מעגלים' של חזרות על תאים, ובמקרה הגרוע, האלגוריתם יבדוק כל תא בטבלה.

**שאלה 3**

נפתח בהגדרת עץ חיפוש בינארי:

יהי x צומת בעץ חיפוש בינארי, ויהי y הבן השמאלי של x, ו z הבן הימני של x.

מתקיימים התנאים הבאים:

1. הערך של צומת y, וערך כל צומת בתת עץ של y קטנים או שווים לערך בצומת x.
2. הערך של צומת z, וערך כל צומת בתת עץ של z גדולים או שווים לערך בצומת x.

נראה כי תנאים אלו מתקיימים גם בשגרת המחיקה החדשה, בשגרה החדשה ישנם 3 שלבים למחיקת צומת z בעל שני בנים:

1. איתור העוקב y.
2. החלפה בין left[y] לבין left[z].
3. z אב לבן יחיד - מחיקת z כך שהבן הימני של z יכנס במקומו.

**\*הערה**

בשפה לא פורמאלית, נוכל להגיד כי איתור y, העוקב של z, זה בעצם ללכת לבן הימני של z, ומשם להמשיך ללכת לבנים השמאליים עד הסוף - הבן השמאלי האחרון הוא y.

ולכן - נוכל מיד להסיק כי הערך left[y] תמיד יהיה Nil.

נראה כי תנאי העץ חיפוש בינארי (השמורות) מתקיימים בכל שלב.

שלב א

בשלב זה רק נעשה איתור של העוקב של z, ולכן העץ אינו מושפע ונשאר עץ חיפוש בינארי.

שלב ב

מכיוון שהסברנו ש left[y] הוא בעצם NIL, אז אחרי שלב זה,ל- z אין בן שמאלי, ו y מקבל את כל התת עץ השמאלי של z.

כזכור, לפי ההגדרה, ערך הצומת y גדול שווה מצומת z (כי הוא העוקב שלו), וגם כל התת עץ השמאלי של z קטן שווה מ z (הגדרה). ולכן אחרי שינוי זה, עדיין נקבל כי y גדול שווה מכל התת עץ השמאלי שנוסף לו. הבעיה היחידה כרגע היא של- z ישנם ערכים בתת עץ הימני אשר קטנים ממנו , אך פרט זה לא רלוונטי כי שלב ג' מסדר אותו.

שלב ג

מוחקים את z ושמים את הבן הימני שלו (נקרא לו x) במקומו.

ערך x גדול שווה מערך z (בן ימני). מכיוון ש y הוא בשרשרת הבנים השמאליים של x אז גם ערך y קטן מערך x. ומכיוון שבסעיף קודם הראנו שהצומת y מקיימת את התנאים, אז כמובן שגם x גדול שווה מכל התת עץ השמאלי שלו, וקטן שווה מכל התת עץ הימני שלו.

ואם כך - הראנו כי לאורך כל הדרך נשמרת הגדרתו של עץ החיפוש בינארי, והמחיקה מוגדרת היטב.

**זמן ריצה**

זמן הריצה של שגרת המחיקה החדשה נקבע ע"פ השלב הראשון שלו - איתור העוקב של הצומת.

במקרה הגרוע - אורך הצומת הוא , ולכן שגרת החיפוש Tree-Successor ובהתאם שגרת המחיקה יפעלו ביעילות של:

**היתרונות**

בשיטה המתוארת בספר, אנו נאלצים לשמור נתונים על ערך הצומת y ובניו, ובשיטת המחיקה החדשה אין צורך כלל לשמור נתונים על מנת למחוק.

**חסרונות**

בשיטה המתוארת בספר, גובה העץ נשמר פחות או יותר, כי מבחינת מבנה העץ, רק הצומת של העוקב נמחק מהמבנה.

בשגרת המחיקה החדשה, מבנה העץ משתנה, והגובה שלו יכול לגדול בצורה משמעותית - מה שישפיע על שגרת החיפוש של העץ.

**שאלה 4**

**סעיף א'**

ניתן להבדיל בין החוטים לבין המצביעים לבנים האמיתיים בעזרת שדה האב (p).

יהי x צומת בעץ מחווט.

right

אם p[right[x]]=x:

אז right[x] הוא הבן הימני של x.

אחרת:

אז right[x] הוא חוט, כלומר מצביע לעוקב של x בסדר תוכי.

left

אם p[left[x]]=x:

אז left [x] הוא הבן השמאלי של x.

אחרת:

אז left[x] הוא חוט, כלומר מצביע לקודם של x בסדר תוכי.

**סעיף ב'**

נגדיר שגרות בשם IsWireFromRight(x) ו IsWireFromLeft(x) , שמקבלות צומת, ובודקות האם הצומת X מקושרת בחוט מצד ימין או מצד שמאל בהתאמה - כפי שהגדרנו בסעיף א'.

לשגרות הללו, נוסיף את התנאי שאם הבן הוא Nil ( מצב אשר מתייחס לבן השמאלי של האיבר הראשון בסדר תוכי , ולבן הימני של האיבר האחרון בסדר תוכי ), השגרות גם כן יחזירו True.

הכנסה

**WireTree-Insert(T,z)**

**1.** x <-- root[T]

**2.** while (Not-IsWireFromLeft(x) **and** key[z] ≥key[x]) **OR** (Not-IsWireFromRight(x) **and** key[z]<key[x]) **do**

**3.** if key[z] < key[x] then

**4.** x <-- left[x]

**5.** else

**6.** x <-- right[x]

**7.** p[z] <-- x

**8.** if root[T] = NIL then

**9.** root[T]<--z

**10.** else if key[z]<key[y] then

**11.** left[z]<--left[x]

**12.** right[z]<--x

**13.** left[x]<--z

**14.** else

**15.** right[z]<--right[x]

**16.** left[z]<--x

**17.** right[x]<--z

השינויים לעומת השגרת הכנסה המופיעה בספר:

בעוד שבספר תנאי העצירה של לולאת ה While (שורה 2) היה כאשר X מגיע לצומת שהיא NIL, כאן בעץ המחווט אין לנו מצביעים על NIL, ולכן תנאי העצירה יהיה כאשר X מגיע לצומת מחווט מצד ימין, ואנחנו מתכוונים להמשיך ימינה, **או** , כאשר X מגיע לצומת מחווט מצד שמאל, ואנחנו מתכוונים להמשיך שמאלה - במקרה כזה אנחנו מיד יודעים שהצומת X הוא האב לצומת Z החדשה.

עוד שינוי קל שעלינו לעשות הוא עניין החוטים, כמובן שאין זה מספיק רק להגדיר את Z כבן של X, אנו צרכים למתוח חוטים לבנים של Z כך שלא יהיו מצביעים ל NIL (שורות 10-17).

אם הצומת החדש Z שנכנס הוא בן שמאלי של X אז:

- X כמובן העוקב של Z בסדר תוכי, ולכן right[Z] יצביע בחוט ל X.

- left[Z] צריך להצביע על הקודם בסדר תוכי, וזה בדיוק מה שהצביע left[X] לפני ההכנסה של הצומת החדשה, ולכן נדאג קודם לקשר את left[Z] בחוט ל left[X].

אם הצומת החדש Z שנכנס הוא בן ימני של X אז:

- X כמובן הקודם של Z בסדר תוכי, ולכן left[Z] יצביע בחוט ל X.

- right[Z] צריך להצביע על העוקב בסדר תוכי, וזה בדיוק מה שהצביע right[X] לפני ההכנסה של הצומת החדשה, ולכן נדאג קודם לקשר את right[Z] בחוט ל right[X].

מעבר לשינויים המינורים שפורטו כאן, אין ההכנסה של הצומת החדשה Z משנה חוטים אחרים בעץ, וכמובן שהפעולות הנוספות שלנו נעשות ב O(1), ולכן כפי שמוכח בספר, גם שגרה זאת תעבוד ביעליות של:

מחיקה

**WireTree-Delete(T,z)**

**1.** if IsWireFromLeft(z) **OR** IsWireFromRight(z) then

**2.** y<--z

**3.** else

**4.** y<--Tree-Successor(z)

**5.** if Not-IsWireFromLeft(y) then

**6.** x<--left[y]

**7.** else if Not-IsWireFromRight(y) then

**8.** x<--right[y]

**9.** if x≠Nil then

**10.** p[x]<--p[y]

**11.** if p[y] = Nil then

**12.** root[T] <-- x

**13.** else if y=left[p[y]] then

**14.** if x=Nil then // leaf

**15.**  left[p[y]]<-- left[y]

**16.** else if x=left[y] then

**17.**  right[Tree-Predecessor(y)]<--right[y]

**18.** left[p[y]]<--x

**19.**  else

**20.**  left[Tree-Successor(y)]<--left[y]

**21.** left[p[y]]<--x

**22.** else

**23.** if x=Nil then // leaf

**24.**  right[p[y]]<-- right[y]

**25.** else if x=left[y] then

**26.**  right[Tree-Predecessor(y)]<--right[y]

**27.** right[p[y]]<--x

**28.**  else

**29.**  left[Tree-Successor(y)]<--left[y]

**30.** right[p[y]]<--x

**31.** if y≠z then

**32.** key[z]<--key[y]

**33.** copy y's satellite data into z

**34.** return y

השינויים לעומת השגרת המחיקה המופיעה בספר:

בעוד שבשגרה בספר אנו מבדילים בין המקרים 'עלה או אב לבן יחיד' לבין 'אב לשני בנים' ע"י הצבעה של הבנים ל NIL, בעץ מחווט אין לנו ערכי NIL (פרט לאיבר הראשון בסדר תוכי והאחרון בסדר תוכי), ולכן נשנה את כלל השוואות ל NIL לבדיקת חוטים (כפי שמופיע בשורות 1,5,7).

בשורה 9 נשמרת השוואה ל NIL כי היא מתייחסת למצב שבו ל Y אין בנים, ולכן X לא יאותחל לשום ערך ואכן ישאר כ NIL, בעצם גם בהמשך, כאשר נשאל האם x=Nil, המשמעות היא ש X לא אותחיל, ולכן זה שקול לשאלה " האם Y עלה ? ".

בעצם מלבד השינוי המינורי הנ"ל, נותר רק לדאוג לסדר את החוטים אחרי המחיקה, נסביר את שינויי החוטים בכמה מקרים:

אם הצומת Y שנמחק הוא עלה (שורות 14,23):

* אם Y הוא בן שמאלי - אז כל מה שנותר הוא לעדכן את המצביע left של האבא של Y.

כמובן שמצביע זה צריך לקבל את הקודם בסדר תוכי, וזה בדיוק מה שהיה left של Y, ולכן העדכון הדרוש הוא: left[p[y]]<--left[y] (שורה 15).

* אם Y הוא בן ימני - באופן שקול העדכון הנדרש הוא right[p[y]]<-- right[y] (שורה 24).

אם הצומת Y שנמחק הוא אב לבן יחיד (נקרא לבן X) (שורות 16,19,25,28):

* אם X הוא בן שמאלי של Y - אז הדבר היחידי שישתנה זאת ההצבעה של הקודם של Y בסדר תוכי, מכיוון שהקודם של Y הצביע בעצם על Y מצד ימין, אך עכשיו Y נמחק, ולכן ההצבעה שלו תשתנה לאותו מצביע ש Y הצביע מצד ימין קודם לכן, כלומר:

right[Tree-Predecessor(y)]<--right[y] (שורות 17,26).

* אם X הוא בן ימני של Y - באופן שקול, הדבר היחידי שישתנה זה ההצבעה של העוקב של Y בסדר תוכי, מכיוון שהעוקב של Y הצביע על Y מצד שמאל, אך עכשיו Y נמחק, ולכן ההצבעה שלו תשתנה לאותו מצביע ש Y הצביע מצד שמאל קודם לכן, כלומר:

left[Tree-Successor(y)]<--left[y] (שורות 20,29).

ושוב, הפעולות הנוספות שלנו (שינויים של החוטים בעץ ) נעשות ב O(1).

השגרה משתמשת בשגרת החיפוש Tree-Successorאשר רצה ביעילות של , ולכן, לפי הספר, סה"כ הסיבוכיות של שגרה זאת היא:

**סעיף ג'**

ההיתרון המשמעותי של השימוש בחוטים הוא מציאת העוקב והוקדם בסדר תוכי ביעילות של O(1) במקרים מסויימים ( לצמתים שהם לא אבות לשני בנים ), דבר שיכול לקצר זמני ריצה.

תמיר אביב,

305652000