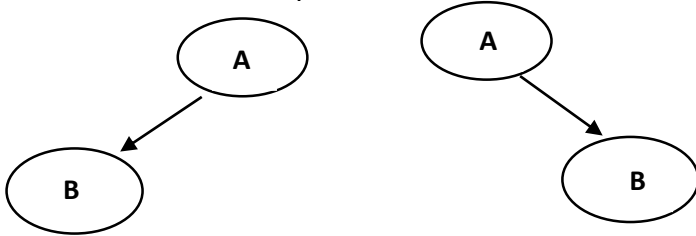


תרגיל 3-פתרון

1. נחלק למקרים:

א. אם מדובר בעץ לא חיפוש בינארי לא ניתן לקבוע: לדוגמה עבור שני עצים הנ"ל נקבל את אותו המעבר:



ב. עבור עץ חיפוש בינארי נראה אלגוריתם. עבור רשימה באורך 0: טרויאלי.

עבור רשימה ארוכה מ-1, נקבע את השורש להיות בערך הראשון ברשימה. נחלק את הרשימה לשני חלקים: חלק ראשון- האיברים שקטנים מהשורש, חלק שני- האיברים שגדולים מהשורש ונפעל ברקורסיה על כל חלק.

2. בזמן הוצאה והכנסה יש לשמור על תכונת הערימה. לשם כך נשתמש באלגוריתם ההופך עץ בינארי כמעט שלם אשר עבורו מופרת תכונת הערימה רק בשורש בחזרה לערימה. העץ מורכב משורש v המצביע לשתי ערימות.

נציע אלגוריתם המתקן את הערימה (ערימת מקסימום) sift-down (v):

1. אם V עלה או גדול משני בניו -סיים.

2. אחרת, החלף את v עם הבן בעל מפתח המקסמלי u והמשך עם הערימה ששורשה u .

קעת נציע אלגוריתם הבונה ערימה ב $O(n)$:

:MakeHeap($T, x_1 \dots x_n$)

עבור על כל צמתי העץ הפנימיים כך שנעבור על הבנים לפני הוריהם (איזה מעבר למדתם שמתאים)

עבור כל צומת בצע sift-down(v)

נוכיח נכונות: לאור סדר בחירת הצמתים בזמן שהאלגוריתם מבצע sift-down(v) הצומת v כבר מצביע

על עצים המקיימים את תכונת הערימה. לפיכך לאחר ביצוע sift-down נוצרת ערימה.

הסבר זה נכון עבור כל צומת, כולל השורש, ולכן בסיום התהליך נקבל ערימה תקינה.

זמן ריצה: תחילת נסתכל על המשפט הבא: עבור עץ בינארי שלם בגובה h הכולל $n = -1 + 2^{h+1}$

צמתים, סכום הגבהים קטן מ- n . (ההוכחה מושארת לקורא).

נשים לב שזמן הריצה של MakeHeap נתון ע"י סכום הגבהים של כל הצמתים: $O(\sum_v h(v))$

כדי להדפיס את כל K הערכים בעץ חיפוש בינארי הגדולים מ- X וקטנים מ- Y , נפעל ע"פ האלגוריתם הבא:

נחפש את האיבר X בעץ, בשיטת $inorder$ (אנו בוחרים דווקא בשיטת מעבר זו, כי המטרה היא לא להדפיס אף איבר הקטן ממנו, ולכן: בפעם הראשונה בה נגיע לאיבר X , אנו יודעים ש"דילגנו" על כל הערכים הקטנים ממנו ועדיין לא "נגענו" באף ערך הגדול ממנו).

כשנמצא את X , נמשיך לעבור על העץ בדרך של $inorder$, אך הפעם, לא רק "ניגע" בקודקוד, אלא ממש נדפיס אותו (בשיטת מעבר מסוג $inorder$, כאמור, אנו דואגים לכך שההדפסה תהיה ממוינת, מאחר ובעץ חיפוש בינארי תמיד מתקיים שהבן השמאלי קטן מהאבא, והאבא קטן מהבן הימני. זהו אכן סדר ההדפסה של $inorder$ - בן שמאלי, אבא, בן ימני, עבור כל תת-עץ בעץ).

אנו "ניכנס" לפקודת ההדפסה אם הערך בקודקוד הנוכחי גדול מ- X וקטן מ- Y . כלומר: רק כאשר אנו בטווח הערכים הרצוי (בין X ל- Y), וכל עוד לא הגענו ל- Y , נמשיך ונדפיס את הערכים. בצורה זו, כל הערכים יודפסו בצורה ממוינת.

יש לשים לב כי אנו לא עוברים על כל הקודקודים בעץ. למשל, כאשר הקודקוד הנוכחי קטן מ- X -- אנו כלל לא פונים לבן השמאלי שלו, מאחר ואנו יודעים בוודאות שלא יהיו שם ערכים בין X ל- Y , מאחר וכל הערכים שם קטנים ממש מ- X . באופן סימטרי, לא נקרא לבן הימני של קודקוד מסוים, אם הערך שלו גדול מ- Y . בצורה זו אנו בעצם "חוסמים" את העץ שלנו לטווח הערכים הרצוי. רק כאשר נמצא את X (או את העוקב ל- X) נתחיל להדפיס את הערכים בשיטת $inorder$ וכשנגיע ל- Y (או לקודם המייד שלו) - הדפסה זו תסתיים, כי הקריאות הרקורסיביות יבואו על סיומן.

בעצם, כל שעלינו לעשות הוא להוסיף לפונקציה $inorder$ את תנאי ההדפסה הבאים:

BetweenXY(node *, int X, int Y)

```

if (v==NULL) then
    return;
if (v->value > X) then // חסימת העץ" משמאל אם ערכיו קטנים מהגבול התחתון
    BetweenXY(v->left); // קריאה רקורסיבית לתת העץ השמאלי
if (v->value > X && v->value < Y) then // הדפסת הערך רק אם הוא בטווח
    Print v->value;
if (v->value < Y) then // חסימת העץ" מימין אם ערכיו גדולים מהגבול העליון
    BetweenXY(v->right); // קריאה רקורסיבית לתת העץ הימני
    
```

סיבוכיות האלגוריתם:

סיבוכיות זמן: הסיבוכיות של $inorder$ היא $O(n)$ (מעבר על כל הקודקודים פעם אחת בלבד), אך אנו לא עוברים על כל הקודקודים, אלא, תחילה אנו מחפשים את X ובכך מתחילים את הריצה. סיבוכיות החיפוש של קודקוד כלשהו בעץ היא $O(\log n)$, כגובה העץ.

בנוסף, מרגע שמצאנו את הקודקוד הרלוונטי - אנו עוברים על K הערכים ומדפיסים אותם לפי סדר $inorder$, בזה אחר זה, דבר המאפשרת הדפסה ממוינת. פעולה זו נעשית בסיבוכיות $O(K)$ כי אנו עוברים על K קודקודים שעלינו להדפיס אותם. ולכן: **סיבוכיות אלגוריתם זה היא $O(K + \log n)$.**

סיבוכיות מקום: אנו "מאחסנים" עץ בעל n קודקודים ולכן: **סיבוכיות המקום היא: $O(n)$.**

7. הרעיון: נעבור על הרשימה ונכניס את $n/2$ איברים הראשונים לתור ואת $n/2$ איברים האחרונים למחסנית. לאחר מכן נרוקן את התור ואת המחסנית תוך כדי ההדפסה.

פסאודו-קוד:

1. *Create Q // create queue*

2. *Create S // Create stack*

3. $x = head(L)$

4. *while* $i \leq n/2$

Q.enqueue(x)

$X = next(x)$

$i += 1$

5. *while* $j \leq n$

S.push(x)

$X = next(x)$

$j += 1$

6. *while* $Q.isempty == False$ and $S.isempty == False$:

Print(Q.dequeue)

Print(S.pop)

*Print(" * ")*

עבור סדרה של מספרים – עלינו למצוא את תת-הסדרה המונוטונית העולה בעלת האורך המקסימאלי. בתחילה, נבנה את העץ בתור עץ בינארי לכל דבר (ע"י הכנסה רגילה – מה שגדול מהקודקוד – נצרך לבן הימני ומה שקטן ממנו – לבן השמאלי). תוך כדי בנייה נדאג להכניס יחד עם הערך המספרי של הקודקוד, גם index רץ של מיקומו בקלט.

כעת, משהעץ בנוי, "נלך" עד לבן השמאלי ביותר, ונעבור בדרך בכל הבנים השמאליים שבדרך מהשורש ועד אליו (כלומר: השורש, בנו השמאלי ביותר, בנו השמאלי ביותר שלו וכן הלאה), בכל פעם נכניס אחד מהם למחסנית (S1), כאשר היא ריקה. אלו יהיו התחלות כל תתי הסדרות שנמצא בעץ.

בכל שלב, נעבור על העץ בצורת Inorder ונכניס את האיבר למחסנית הסדרה (S2), רק אם האינדקס שלו גדול מהאינדקס של הערך הנוכחי (אין טעם לבדוק האם גם הערך המספרי גדול מהערך המספרי של הצומת הנוכחי, מאחר ואנו עוברים על העץ בשיטת Inorder, בה אנו דואגים לעבור על העץ באופן ממוין).

נמשיך לעבור בצורה זו על כל העץ, עד שנגיע בכל פעם לערך המקסימאלי בתת הסדרה, עבודה, אין ערכים גדולים יותר בעלי אינדקסים גדולים יותר. אולם, יכול להיות שיש איברים בעלי אינדקסים גדולים יותר שערכם המספרי קטן מזה שמצאנו, אך עדיין גדול מערכו של האיבר הקודם לו במחסנית, ולכן, נשמור את תת הסדרה הזו ואת אורכה (max=) ונוציא את האיבר האחרון במחסנית (S2) ונמשיך לעבור על העץ בצורת Inorder. עתה, ערכים אחרים ייכנסו למחסנית (S2), כי האינדקס אליו אנו משווים, קטן יותר מזה של האיבר שהוצא מהמחסנית (S2) קודם לכן, ולכן: נמצא אפשרויות נוספות לתתי-סדרות אופציונאליות ונשנה את ערך max בהתאם.

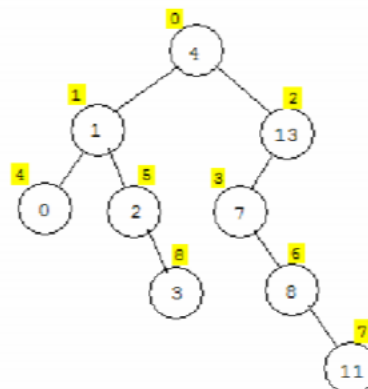
כאשר המחסנית (S2) תתרוקן – נכניס את האיבר הבא מ"השורש" השמאלי של העץ הנמצא במחסנית S1. בסופו של דבר – נדפיס את כל תתי הסדרות בעלות האורך המקסימאלי.

דוגמה הממחישה אלגוריתם זה:

(הדוגמה שניתנה בתרגיל): נתונה הסדרה { 4, 1, 13, 7, 0, 2, 8, 11, 3 }.

נמצא את כל תתי-הסדרות המונוטונית העולות ע"פ האלגוריתם שלעיל.

נבנה את העץ. לאחר הבנייה, העץ נראה כך:



(המספרים המסומנים בצהוב הם האינדקסים ואילו המספרים הנמצאים בקודקוד הם הערכים ממש).

דוגמה: מכניסים את 4, 1 ו-0 למחסנית S1, מאחר ובכל פעם פונים לבן השמאלי ביותר ומפסיקים ב-0 כי זה השורש.

נכניס את 0 למחסנית S2. נעבור על העץ בצורת Inorder ונבדוק בכל פעם אם האינדקס של האיבר הבא גדול ממנו. אם כן – נכניס אותו למחסנית. לאחר שהכנסנו את 0, נעבור ל-1, האינדקס של 1 (1) לא גדול מזה של 0 (4) ולכן: לא נכניס אותו למחסנית. לעומת-זאת, האינדקס של 2 (5) כן גדול מהאינדקס של 0 (4) ולכן: כן נכניס את 2 למחסנית. באופן דומה, נכניס גם את 3. נמשיך לעבור על העץ בצורה זו ולראות, שעד שנגיע ל-11 (הקודקוד האחרון בשיטת מעבר זו – לא נמצא ערך גדול מזה של 3, ועל כן – מצאנו תת-סדרה ראשונה באורך 3). נשמור את תת הסדרה {0, 2, 3} ואת אורכה (3) ונעדכן את $\max=3$. נוציא את 3 מהמחסנית S2 ונמשיך בהכנסת האיברים בשיטת Inorder, כאשר בכל פעם נבדוק עתה אם האינדקס שלהם גדול מהאינדקס של האיבר העליון במחסנית S2. עבור כל האיברים הבאים בסדר זה – 4, 13 ו-7 – האינדקס שלהם לא גדול מזה של 2 (5) ולכן, לא נכניס אותם למחסנית, אך כאשר נגיע לאיבר 8 – האינדקס שלו (6) דווקא כן גדול מזה של 2 (5) ולכן: נכניס אותו למחסנית. האינדקס של 11 (7) גדול מזה של 8 (7), שהוא האיבר האחרון במחסנית S2, ולכן: נכניס גם אותו.

הגענו לקודקוד האחרון בשיטת מעבר Inorder ולכן: מצאנו תת סדרה נוספת. נראה כי אורכה (4) גדול יותר מזו של תת הסדרה הקודמת (3) ועל-כן, נעדכן את ערך $\max=4$ ונשמור תת סדרה נוספת זו {0, 2, 8, 11} ואת אורכה (4). בצורה זו נמשיך להוציא את איברי המחסנית S2 בזה אחר זה ולמצוא תתי-סדרות נוספות. כאשר נוציא את 0, המחסנית S2 תתרוקן ולכן: נכניס את האיבר הבא במחסנית S1 (1) ונחזור על התהליך המתואר לעיל שוב. לבסוף, נעבור על כל תתי-הסדרות השמורות ונדפיס רק את אלו שאורכן שווה לאורך המקסימאלי (4).

