Ben-Gurion University of the Negev

Solution

שאלה 1:
נתון לנו מערך בגודל n לא ממויין ועלינו למצוא את החציון.

נתאר את האלגוריתם ולאחר מכן נראה זמן ריצה.

תיאור האלגוריתם:
נתחזק טווח נאתחל את הערך התחתון של הטווח באיבר המינימלי במערך והערך העליון במקסימלי.
כל עוד לא מצאנו את החציון נעבור על המערך החל מהאיבר הראשון עד שנמצא איבר שנמצא בטווח (כיוון שהוא לא ממויין ואין לנו אף מידע על האיברים בו זה שווה ערך להגרלה).

נבדוק על האיבר הזה כמה איברים קטנים ממנו וכמה גדולים ממנו.

אם מספר האיברים שגדולים ממנו גדול מ$\frac{n}{2}$ נעדכן את הערך המינימלי של הטווח להיות האיבר הזה.

אם מספר האיברים שקטנים ממנו גדול מ$\frac{n}{2}$ נעדכן את הערך המקסימלי של הטווח להיות האיבר הזה.

<u>אבחנה:</u> מתכונת החציון נשים לב שבמערך ממוין אי זוגי החציון ממקום במקום ה - $\frac{n}{2}$ ואילו במערך זוגי החציון ממוקם במקום ה - $\lfloor\frac{n}{2}\rfloor$ (ערך תחתון). מכך נסיק שבאלגוריתם שלנו:
במקרה שהמערך אי זוגי: כאשר נגיע לאיבר במערך שמס' האברים הגדולים ממנו שווה למספר האברים הקטנים ממנו נדע בוודאות שהוא החציון.
במקרה שהמערך זוגי: כאשר נגיע לאיבר במערך שמס' האברים הגדולים ממנו גדול ב – 1 ממס' האברים הקטנים ממנו נדע בוודאות שהוא החציון.

נשים לב להבחנה חשובה:

בכל הגרלה של מספר, (במקרה שלנו מעבר החל מהאיבר הראשון, אבל זה שקול), אנחנו **מצפים** "לזרוק" לפחות $\frac{1}{8}$ מהמערך, כלומר לצמצם את הטווח שלנו.

נסביר זאת ע"י חלוקה של הטווח לרבעים, נגיד שהטווח ה"טוב" שלנו הוא החצי האמצעי והטווח "הרע" הוא הרבעים בקצה. קיים סיכוי של $\frac{1}{2}$ בהגרלת מספר לפגוע בחצי הטוב וקיים סיכוי זהה לפגוע ברבעים בקצוות. נניח שאילו פגענו באחד הרבעים בקצה במקרה הגרוע צמצמנו את הטווח שלנו רק באיבר אחד אבל אילו פגענו בחצי הטוב צמצמנו לכל הפחות את הטווח בשלושת רבעי (כי "זרקנו" את הרבע בקצה המתאים). ולכן אנחנו מצפים בכל איטרציה לזרוק:

$$E[X] \geq \frac{1}{2}\cdot\frac{1}{4}+\frac{1}{2}\cdot\frac{1}{n} \geq \frac{1}{8}$$

כל איטרציה מצמצמים את הטווח לפחות בשמינית ולכן מכיוון שזה *fraction* קל להראות שמספר הפעמים שדרושות כדי לצמצם את הטווח לאיבר אחד הוא:$O(\log n)$

חישוב זמן ריצה:
אתחול טווח, מציאת מינימום ומקסימום במערך: *O(n)*.
מתבצעת לולאה כל עוד לא מצאנו את החציון, לפי החישוב הנ"ל הלולאה הזו תתבצע בזמן צפוי של: $O(\log n)$
בכל כניסה ללולאה נמצא איבר שנמצא בטווח (מעבר על המערך): *O(n)*
מציאת מספר האיברים הקטנים ממנו והגדולים ממנו (מעבר על המערך): *O(n)*
האלגוריתם הסתברותי ולכן הזמן הצפוי בסך הכל הוא:

$$O(n\log n)$$

סיבוכיות זכרון:

אנו מחזיקים מספר קבוע של שדות ולכן כמות הזכרון הנדרשת היא $O(1)$.

שאלה 2:

נתאר את האלגוריתם ולאחר מכן נחשב את זמן הריצה:
תיאור האלגוריתם:

באופן כללי ננקוט בשאלה זו באסטרטגיה של אופטימיזציה, כלומר נמצא תחילה טווח תקין כלשהו ולאחר מכן ננסה לצמצם אותו ככל הניתן.
ניצור שתי ערמות, ערמת מינימום וערמת מקסימום, בגדול k (כמספר המערכים). נאתחל שני משתני int a ו – b האחד (a) ייצג את הערך ההתחלתי של הטווח והשני (b) את הערך הסופי של הטווח.
נאתחל k מצביעים לכל אחד מהתאים הראשונים של כל אחד מהמערכים ונכניס את האבר הראשון של כל אחד מהמערכים לכל אחת מההערמות שיצרנו. כעת, נפעל באופן הבא:
לכל אחד מהמערכים נבצע לולאת while שכל עוד הערך בתא הבא קטן או שווה מערך המקסימום **וגם** המערך עדיין לא נגמר, נמחק את האיבר הקודם לו משתי הערמות ונכניס אותו (ונקדם את המצביע). נשים לב שבביצוע פעולה זו אנו רק מקרבים את הערך של כל אחד מהמערכים לערך המקסימום ולכן בהכרח מצמצמים את הטווח. בסיום התהליך של לולאת ה while לכל אחד מהמערכים נקבל טווח נציגות שמתחיל בשורש של ערמת המינימום ומסתיים בשורש של ערמת המקסימום נעדכן את a ו – b שאתחלנו להיות השורש של ערימת המינימום והשורש של ערימת המקסימום בהתאמה.
נשים לב שבחלק הראשון של האלגוריתם הגענו לטווח המצומצם ביותר האפשרי עם האברים הראשונים של כל אחד מהמערכים כפי שתואר לעיל, זאת אומרת שבטווח שנמצא בחלק הראשון של האלגוריתם, מעבר לתא הבא של אחד מהמערכים תגרום בהכרח להגדלת הטווח, מכך ניתן להסיק שהטווח הבא הקצר מהטווח שמצאנו חייב להתחיל מהאבר המקסימלי בטווח הראשון שמצאנו. כעת, נבצע הכנסות של כל האברים הבאים של כל אחד מהמערכים שהמצביע שלהם הוא לא האבר המקסימום בטווח הראשון שמצאנו (שהמצביע שלהם למעשה לא מצביע ל b) לערמות תוך כדי מחיקת האברים הקודמים (שקטנים ממש מ b). נחזור על לולאת ה while מהחלק הראשון של האלגוריתם לכל אחד מהמערכים עד ששוב נגיע לטווח ונעצר. נבדוק אם הטווח שהתקבל קטן יותר מהטווח הנוכחי, אם כן, נחליף ביניהם ונמשיך באלגוריתם בדיוק באותו האופן, אם לא, לא נחליף בין הטווחים ונמשיך לבצע את הפעולות שתוארו עד שנגיע לאיבר האחרון בכל המערכים.

ניתוח זמן ריצה:
באלגוריתם אנו עוברים על כלל האברים במערך מכניסים ומוחקים אותם (לפי הצורך) לשתי ערימות בגודל K. במקרה הגרוע אנו נמחק כל אבר שנכניס לערמה. מתיאור האלגוריתם, אבר שנמחק לא יוכנס שנית לערמה. כמו כן, הגישות לאבר המינימום והמקסימום שמתבצעות לאורך כל האלגוריתם הוא קבוע מכיוון שמדובר בערמות. לכן, בסה"כ מתבצעות לכל היותר שתי הכנסות לכל אבר ושתי מחיקות (הכנסה ומחיקה לכל אחת מהערמות). זמן הכנסה ומחיקה בערמה הוא log(n) , לכן מכיוון שסך הכל יש לנו n אברים וגודל כל ערמה הוא k נבצע: $4n * \log(k) + cn$ פעולות, (הוספנו את cn כי לכל היותר נגש סדר גודל של n פעמים לשורש וגישה יחידה היא O(1) בערמות). ולכן נקבל שזמן הריצה הוא : $4n * \log(k) + cn = O(nlog(k))$ כנדרש.

שאלה 3:

סעיף א':

נתחזק עץ avl שיעזור לנו במהלך האלגוריתם, נסמנו tree.

שלב א':

כעת נעבור על אברי הקבוצה, ולכל איבר x נבצע tree.search(x), כלומר נבדוק האם אותו איבר נמצא בעץ.

אם התשובה חיובית, כלומר האיבר נמצא בעץ, לא נבצע כלום.

אחרת אם התשובה שלילית, כלומר האיבר לא נמצא בעץ, נבצע tree.inset(x), כלומר נכניס את האיבר לעץ.

הערה: אם יש לנו את הערך z/2 ואנחנו מבצעים עליו חיפוש בעץ ומוצאים איבר ששווה לו שכבר נמצא בעץ באלגוריתם יחזיר תשובה חיובית. כיוון ש: $\frac{z}{2}+\frac{z}{2}=z$

שלב ב':

כעת נעבור על העץ in-order. נשים לב להבחנה הבאה: אם a+b=z אז a=z-b כלומר לכל איבר a שנשלוף מהעץ בסריקה, נבצע חיפוש לאיבר z-a, אם מצאנו איבר כזה סיימנו אחרת נמשיך בסריקה.

זמן ריצה:

עוברים על איברי הקבוצה ולכל איבר מבצעים חיפוש בעץ avl ובמידה ולא נמצא בעץ נבצע הכנסה, פעולות אלה יקחו לנו יקח לנו $O(\log\log n)$, נראה זאת:

אנו מבצעים חיפוש תחילה בעץ ריק לאחר מכן על עץ בגודל אחד וככה הלאה עד לגודל העץ שהוא כאמור סדר גודל של $\log n$ ולכן בסהכ:

$$\sum_{i=0}^{\boldsymbol{log\, n}} \log i = \log((\log n)!) = O(\log\log n)$$

אותו חישוב מתבצע גם להכנסה. ופעולות אלה מתבצעות n פעמים ולכן זמן הריצה של שלב זה הוא: $O(n\log\log n)$

כעת על כל אחד מאיברי העץ נבצע חיפוש לזוג שישלים אותו לסכום z, סך הכל יש $\mathrm{O}(\log n)$ איברים כאלה וכל חיפוש יעלה לנו $\mathrm{O}(\log\log n)$ ולכן בסך הכל זמן הריצה של שלב ב' הוא: $\mathrm{O}(\log n\log\log n)$

ולכן בסך הכל זמן הריצה הכולל של האלגוריתם הוא: $O(n\log\log n)$**.**

סעיף ב':

במקרה בו z<n ניצור מערך A של אינטג'רים באורך z+1 ונאתחל את כל הערכים בו להיות 0 . נשים לב שבכדי שסכום שני מספרים מהקבוצה יהיה שווה ל z שני המספרים הללו חייבים להיות קטנים/ שווים מ z כל אחד. נעבור על כל אברי הקבוצה S . עבור כל מספר i ב - S שקטן או שווה ל z נוסיף אחד בתא ה A[i] במערך שיצרנו. לאחר שעברנו על כל האיברים בקבוצה S נבצע את התהליך הבא:

ניצור משתנה בוליאני b ונאתחל אותו להיות false. כמו כן, נאתחל משתנה אינטג'ר להיות בעל הערך z.נרוץ בלולאת for על כל המערך A שיצרנו מהתא האחרון ועד התא הראשון כל עוד b false או עד שהגענו לתא הראשון בלי ש b השתנה.
לכל תא החל מהתא האחרון במערך נעשה את הפעולות הבאות:
נבדוק האם הערך של התא גדול מ – 0. אם כן, נבצע z-A[i] וניגש לתא המשלים של הערך של i ל – z , זה יהיה התא ה – A[z-i] ונבדוק אם ערכו גדול מ – 0. אם כן אזי נוריד את ערכו של האינדקס של התא מהמשתנה שאתחלנו בערכו של z (והוחסר ממנו הערך של התא הראשון שנבדק ועל כן לאחר ההחסרה הזו ערכו יהיה 0). ונעדכן את b להיות true , אם לא נעבור לתא הבא.
(**דגש:** העקרון עליו הסתמכנו באלגוריתם הזה הוא אותו עקרון לפיו פועל Counting Sort).

חישוב זמן ריצה של האלגוריתם:
נשים לב שבמקרה הגרוע z<n במספר קטן של איברים (או יכול להיות אפילו שווה ל n) ולכן את רוב אברי הקבוצה נצטרך לשים במערך. במקרה הגרוע, אנו מאתחלים מערך באורך n+1 ושמים את הערך 0 בכל אחד מהתאים, זה יקח זמן של $2n+2=O(n)$. לאחר מכן, עוברים שוב על כל המערך A שיצרנו ובכל ריצה בלולאה במקרה הגרוע מחסירים פעמיים ערך מ – z , סך הכל מספר קבוע של פעולות n פעמים ולכן זמן הריצה כאן גם כן יהיה $O(n)$, אם נחבר את זמני הריצה נקבל זמן ריצה לינארי שתלוי ב – n ולכן זמן הריצה הכולל של האלגוריתם יהיה $O(n)$.

שאלה 4:

יהי גרף G=(V,E). ניצור גרף חדש G'=(V',E') כך שלכל קודקוד מהגרף המקורי v נוסיף עוד "כפיל" v' ולכל קשת $(v1,v2)$ בגרף המקורי, נחסיר אותה ונוסיף את הקשתות (v'1,v2) ו- (v1,v'2). כעת יצרנו גרף דו"צ שמורכב מקודקודי הגרף הראשון בצד אחד והקודקודים "הכפילים" בצד השני. כעת נשים לב שאילו היה קיים מסלול זוגי בין x ל-y בגרף המקורי, עדיין אותו מסלול יהיה קיים באורך זהה בגרף החדש, אולם לעומת זאת אילו היו אך ורק מסלולים מאורך אי זוגי, אזי לא יהיה מסלול כעת בין xל-y. נראה את נכונות הטענה:
אי קיום מסלול אי זוגי: זהו גרף דו צדדי ולכן לא קיים מסלול באורך אי זוגי מקודקוד x בצד אחד לקודקוד y באותו צד.

קיום מסלול זוגי מינימלי, נניח בשלילה שקיים מסלול מינימלי אחר, אז זו סתירה למינימליות האורך של המסלול וזו סתירה.

לכן האלגוריתם הוא:

ניצור קבוצת קודקודים חדשה, לכל קודקוד ניצור "כפיל": O(V)

ניצור קבוצת קשתות חדשה, לכל קשת, נסיר אותה ונוסיף שתיים במקומה: O(E)

נריץ BFS החל מקודקוד s כלשהו שנקבל כקלט: O(V+E)

כעת בכל קודקוד בגרף המקורי אורך המסלול הוא או אורך המסלול הזוגי המינימלי מקודקוד s או אינסוף.

לכן זמן ריצת האלגוריתם בסך הכל הוא: $O(V+E)$

שאלה 5:

סעיף א:

נבצע סריקת DFS על העץ, כאשר אנו ממספרים את זמן הכניסה וזמן היציאה של כל קודקוד. בנוסף נתחזק משתנה שיספור את זמן היציאה האחרון נקרא לו lastTime, משתנה זה יתעדכן במהלך האלגוריתם DFS בכל פעם כאשר אנו יוצאים מקודקוד מסויים ולכן בסוף הריצה המשתנה הזה יחזיק את הקודקוד האחרון שממנו יצאנו.

הבחנה: אם בגרף קיימים שורשים, הקודקוד בעל זמן היציאה האחרון (כלומר הגדול ביותר) הוא בהכרח שורש.

נסביר את ההבחנה: נניח בשלילה שקיימים שורשים בגרף והקודקוד x בעל זמן היציאה הגדול ביותר אינו שורש. אם הגענו לקודקוד x זאת אומרת שעדיין לא עברנו בשורש מכיוון שאילו כן היינו עוברים בשורש x היה חלק מתת העץ שלו וזו היתה סתירה לכך שזמן היציאה מx הוא מקסימלי. אבל מכיוון ש-x גם אינו שורש זו סתירה לכך שקיימים שורשים בגרף.

כעת נעבור על כל קודקודי הרף עד שנמצא את הקודקוד נסמנו ב- x שזמן היציאה ממנו היה זמן היציאה ששמרנו במשתנה lastTime. כעת אם קיימים שורשים בגרף, x הוא בהכרח אחד מהם.

נאתחל מערך בינארי כגודל מספר הקודקודים בגרף באפסים. כעת נבצע DFS החל מקודקוד x ולכל קודקוד שאנו מגיעים נעדכן את המיקום במערך הבינארי במקום שלו להיות 1. (לדוגמא אם כעת אנחנו נמצאים בקודקוד מספר 3, נעדכן את arr[3] להיות 1). בסוף ה DFS נעבור על ערכי המערך החל מאינדקס מספר 1 ועד הסוף ונחזיר שx הוא שורש אם"ם כל הערכים במערך הם 1 (כלומר ביקרנו בכל קודקוד בגרף כאשר התחלנו מקודקוד x).

ניתוח זמן ריצה:

סריקת DFS ראשונה כדי למצוא את הקודקוד עם זמן היציאה האחרון: O(E+V).

סריקת DFS שניה שמתחילה מהקודקוד החשוד להיות שורש: O(E+V).

מעבר על המערך הבינארי שמייצג אם הגענו לקודקוד $1 \leq i \leq n$: O(V)

ולכן בסך הכל זמן הריצה הוא: $O(V+E)$.

סעיף ב:

כדי למצוא את כל השורשים בגרף G נצטרך קודם כל למצוא שורש כלשהו אחד. נריץ את האלגוריתם שמצאנו בסעיף א' כדי למצוא שורש. אם מצאנו קודקוד כלשהו נסמנו בx אחרת נחזיר שאין שורשים בכלל. כעת נשים לב שאם קיימים שורשים נוספים כלשהם בהכרח קיים מסלול בינם ובין השורש x, מכיוון שאילולא היה להם מסלול לx זהו היה קודקוד שהם לא מגיעים אליו ולכן הם לא שורשים. בנוסף נשים לב שאילו קיים מסלול בין קודקוד y כלשהו לקודקוד x שמצאנו כשורש אזי y בוודאי שורש מכיוון שדרך x ניתן להגיע לכל הקודקודים ואם לy קיים מסלול לx אזי גם לy קיים מסלול לכל אחד מקודקודי הגרף. כעת נותר לנו למצוא את כל הקודקודים שיש להם מסלול לקודקוד x. נשתמש בשיטה הבאה: נעבור על כל קשתות הגרף **ונחליף** את כיווני הקשתות, כעת יצרנו גרף חדש וכל המסלולים שבגרף הישן נכנסו לx כעת יוצאים מx. ולכן נותר לנו לעשות סריקת BFS החל מהקודקוד x וכל קודקוד שנגיע אליו הוא בהכרח שורש בגרף המקורי מכיוון שבגרף המקורי קיים מסלול ממנו לx כיוון שבגרף החדש לאחר היפוך הקשתות קיים מסלול מx לאותו קודקוד.

ניתוח זמן ריצה:

הרצת האלגוריתם מסעיף א': O(E+V)

מעבר על כל קשתות הגרף: O(E)

סריקת BFS החל מהקודקוד שמצאנו דרך האלגוריתם של סעיף א': O(E+V)

ולכן בסך הכל זמן הריצה הוא: $O(V+E)$.

שאלה 6:

עבור כל קלט שהוא אינו חזקה כלשהי של 2, נניח כי הוא דורש 3 פעולות במקום 1. לאחר ההפעלה של האלגוריתם על הקלט הזה נשאר לנו יתרה של 2 נוספים, ה-2 האלה יעזרו לנו לפצות במקרה הממוצע על הפעלת האלגוריתם על איבר שהוא חזקה של 2:

לכל הכנסה של איבר x כך ש: $x = 2^k$ $k\epsilon\mathbb{N}$:

קיימים $2^k - 2^{k-1}$ איברים בהם היתרה שצברנו על כל איבר היא 2, ולכן חסכנו סך הכל:

$2 \cdot (2^k - 2^{k-1})$ פעולות. נפשט את הביטוי ובסך הכל חסכנו 2^k פעולות, בדיוק מספר הפעולות הדרושות להפעלת האלגוריתם על האיבר x. ולכן מספר הפעולות הממוצע הוא לכל היותר 3 לכל איבר. ולכן יעילות האלגוריתם היא: $O(1)$.