גיליון רטוב 2 – חלק יבש

קורס : מבני נתונים 1
מספר קורס: 234218
תאריך הגשה : 17.06.2023
מגישים :
אברהם יבדייב ת.ז. 211904032
נדב לוי ת.ז. 322298795

מגישים : אברהם יבדייב ת.ז. 211904032
נדב לוי ת.ז. 322298795

<u>תיאור המבנה הכללי:</u>

בחרנו לנהל את המערכת באמצעות מבנה נתונים המכיל את הרכיבים הבאים :

טבלת ערבול עבור לקוחות החנות- לפי מספרי ה-Id של הלקוחות.

מבנה נתונים מסוג Union Find עבור התקליטים בחנות כאשר כל קבוצה מייצגת ערמת תקליטים- ממומש על ידי מערך של טיפוסים מסוג Record.

עץ AVL עבור חברי המועדון בחנות, ממוין לפי מספר ה-Id של חברי המועדון, כולל שדה נוסף שבאמצעותו אנו מחשבים את הפרס שניתן ללקוחות בהגרלות.

מערך של מספרים שלמים שמכיל בכל תא i במערך את מספר התקליטים בחנות מסוג זה.

מערך של מספרים שלמים שמכיל בכל תא i במערך את מספר התקליטים שנקנו מסוג i מתחילת החודש.

מספר שלם המכיל את מספר הלקוחות בחנות.

מספר שלם המכיל את מספר התקליטים בחנות.

מספר שלם המכיל את מספר חברי המועדון בחנות.

<u>סיבוכיות מקום:</u>

סיבוכיות המקום של מבנה הנתונים הינה $O(n+m)$ במקרה הגרוע, כאשר n הוא מספר הלקוחות בחנות ו-m הוא מספר התקליטים.

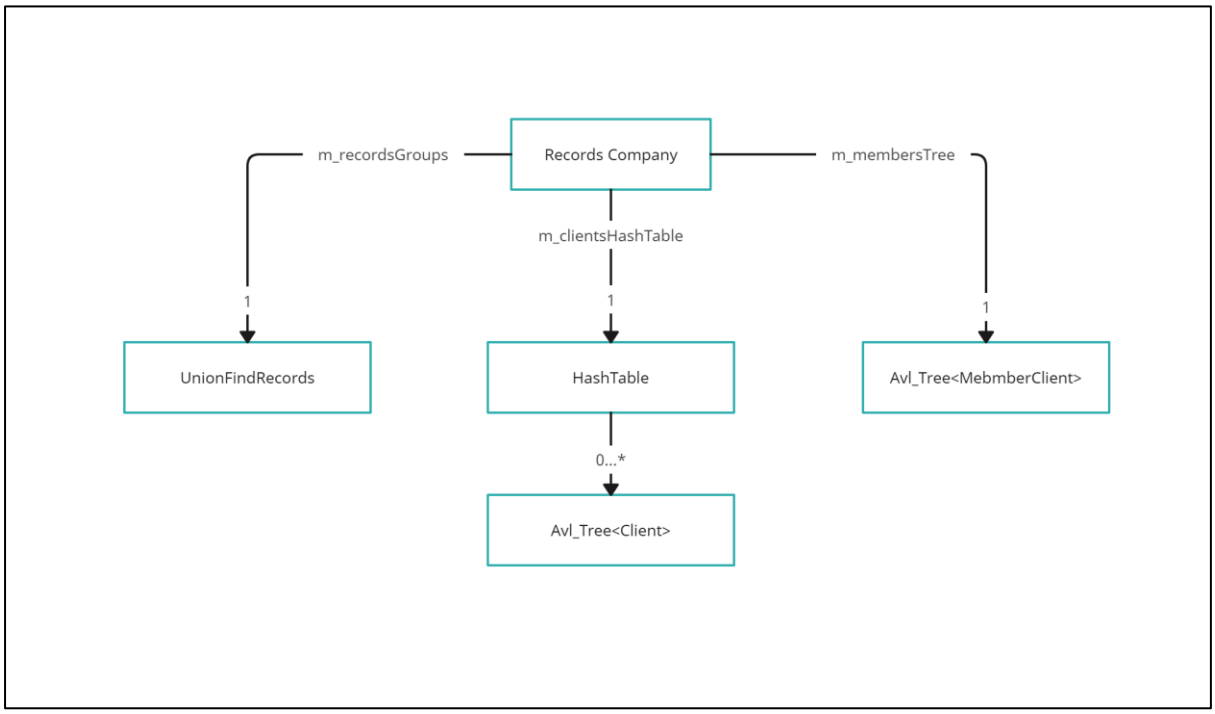
<u>הסבר</u>:

בכל רגע בזמן ריצת התוכנית גודל טבלת הערבול הוא $O(n)$ (נעשות הגדלה והקטנה של הטבלה בעת הצורך, מספר האיברים הכולל בתוך כל תאי הטבלה הוא n בכל רגע נתון), גודל המערך במבנה הנתונים מסוג Union Find הוא כמספר התקליטים בחנות בחודש הנוכחי (כך גם מערכי המספרים השלמים שבמערכת, $O(m)$), ומספר הצמתים בעץ ה-AVL הוא כמספר חברי המועדון- במקרה הגרוע n כאשר כל הלקוחות בחנות הינם חברי מועדון- $O(n)$.

לכן, סיבוכיות המקום הכוללת של המבנה הינה $O(n+m)$.

מגישים : אברהם יבדייב ת.ז. 211904032
נדב לוי ת.ז. 322298795

סקיצה להמחשה:



הוכחת נכונות המימוש:

סיבוכיות טבלת הערבול:

בחרנו לממש את טבלת הערבול כפי שנלמד בהרצאה, על ידי מערך בגודל דינמי שמשתנה בהתאם למספר הלקוחות בחנות בכל רגע נתון, כך שמספר הלקוחות יהיה בטווח שבין 25%-75% מגודל המערך (פרט למצב שבו המערך בגודלו ההתחלתי). בכך מובטח כי בכל שלב של ריצת התוכנית מתקיים $n = O(k)$ (וגם $k = O(n)$) כאשר n זהו מספר הלקוחות בחנות ו-k זהו גודל המערך. פונקציית הערבול בה אנו משתמשים היא mod(k) כאשר k הוא גודל המערך בעת השימוש בפונקציה. עבור פונקציה זו נקבל כי היא מקיימת את הנחת הפיזור האחיד הפשוט בממוצע על הקלט, וכל אחד מ-k איברי המערך נמצא בטווח שלה. לכן, כפי שנלמד בהרצאה, סיבוכיות זמן הריצה המשוערכת של הפעולות delete, insert היא $O(1)$ בממוצע על הקלט, וסיבוכיות זמן הריצה של הפעולה find היא $O(1)$ בממוצע על הקלט (לא משוערכת).

כדי לטפל בהתנגשויות אנו משתמשים בעצי AVL בכל תא במערך (בדומה ל-chain hashing) כך שבמקרה הגרוע, אם כל אחד מ-n הלקוחות בחנות נשלח לאותו תא במערך, זמן מציאת הלקוח במבנה יהיה $O(\log n)$: $O(1)$ פעולות עבור ההגעה לאינדקס המתאים במערך ו- $O(\log n)$ פעולות עבור ההגעה ללקוח המתאים בתוך התא.

סיבוכיות מבנה הנתונים מסוג Union Find:

בחרנו לממש את מבנה הנתונים Union Find כך שכל איבר במבנה מייצג תקליט מסוג מסוים וכל קבוצה במבנה מייצגת ערמת תקליטים (בתחילת חודש כל סוג של תקליטים נמצא בערמה נפרדת, כלומר כל איבר בקבוצה נפרדת). הנחה של ערמה אחת על ערמה שנייה מתבצעת על ידי פעולת union (איחוד של שתי ערמות לאחת) ואיתור הערמה שבה תקליט נמצא מתבצע על ידי פעולת find.

בנוסף, אנו מתחזקים במבנה שדות נוספים לכל קבוצה, בדומה לנעשה ב-"תרגיל הארגזים" מהתרגול, כך שבכל פעולת find ניתן למצוא את העמודה והגובה של כל תקליט בחנות מבלי צורך לעבור על כל איברי

הקבוצה בה הוא נמצא. בכך נוכל לאתר את עמודת התקליט ואת גובהו בערמה מבלי לפגוע בסיבוכיות זמן הריצה של הפעולה find.

את המבנה מימשנו באמצעות עצים הפוכים המיוצגים על ידי מערך (הדבר מתאפשר מכיוון שמספר התקליטים בכל חודש ידוע וקבוע מראש וזהו מספר האיברים במבנה). כפי שנלמד בהרצאה, באופן זה סיבוכיות זמן הריצה המשוערכת של הפעולות find, union במבנה היא $O(\log^* m)$.

<u>סיבוכיות העץ:</u>

העץ המכיל את חברי המועדון בחנות הינו עץ AVL שמומש בהתאם לנלמד בהרצאה כך שהפעולות עליו עומדות בדרישות הסיבוכיות הבאות (כאשר n הוא מספר הצמתים בעץ) : יצירת עץ- $O(1)$, הריסת עץ- $O(n)$, הכנסה והוצאה מהעץ- $O(\log n)$, איתור צומת לפי מידע נתון- $O(\log n)$, קבלת כל המידע שבעץ- $O(n)$, קבלת הערך המקסימלי בעץ- $O(\log n)$.

בנוסף, הוספנו לעץ שדה נוסף שבאמצעותו ניתן לחשב את הפרס לו זכאי כל אחד מחברי המועדון בכל חודש (בדומה ל-"תרגיל המחוסנים" מהתרגול). שדה זה מאפשר את קבלת סכום הפרס על ידי מעבר אחד על צמתי העץ- מהצומת המייצגת את חבר המועדון הרלוונטי ועד לשורש העץ- כך שסיבוכיות זמן הריצה של מציאת הפרס עבור חבר מועדון נתון היא $O(\log n)$. זוהי גם סיבוכיות זמן הריצה של הוספת פרס לחברי מועדון (מתבצע על ידי ירידה משורש העץ לחברי המועדון הרלוונטיים) והוספת חבר מועדון חדש לעץ.

<u>הסבר על הטיפוסים שבהם השתמשנו:</u>

<u>Client-</u> המחלקה מייצגת לקוח בחנות ושומרת עבור כל לקוח את המידע הבא : ה-Id של הלקוח, מספר הטלפון של הלקוח, האם הלקוח הינו חבר מועדון.

<u>MemberClient-</u> המחלקה מייצגת חבר מועדון בחנות ושומרת עבור כל חבר מועדון את המידע הבא : ה-Id של חבר המועדון כלקוח בחנות, החוב של חבר המועדון בחודש הנוכחי.

<u>Record-</u> הטיפוס מייצג תקליט בחנות ושומר עבור כל תקליט את המידע הרלוונטי עבורו כאיבר במבנה הנתונים Union Find (בכלל זאת גודל הקבוצה בה הוא נמצא וגובה הערמה בה הוא נמצא - רלוונטיים רק עבור תקליט שנמצא בשורש העץ, מציין לאב של האיבר בעץ הקבוצה, מציין לתקליט שנמצא בתחתית הערמה הרלוונטית- העמודה בה התקליט נמצא, ושדה נוסף באמצעותו נחשב את גובה התקליט בערמה).

מגישים : אברהם יבדייב ת.ז. 211904032
נדב לוי ת.ז. 322298795

<u>מימוש הפעולות המוגדרות על מבנה הנתונים:</u>

<u>בכל ההסברים להלן n הינו מספר הלקוחות בחנות, ו-m מספר התקליטים בחנות בחודש הנוכחי.</u>
<u>זיכרון ה"עזר" בו כל פעולה משתמשת הינו בסדר גודל של $O(1)$ אלא אם צוין אחרת.</u>

()RecordsCompany- הפונקציה מאתחלת את השדות של מבנה הנתונים בערך ברירת המחדל שלהם.
מספר הפעולות קבוע וידוע מראש ללא תלות בקלט ולכן סיבוכיות זמן הריצה של הפונקציה היא $O(1)$.

()~RecordsCompany - הפונקציה אחראית על שחרור מבנה הנתונים. על מנת לעשות זאת, הפונקציה תמחק ותשחרר את עץ חברי המועדון והאובייקטים שנמצאים בתוכו, את טבלת הערבול והאובייקטים שנמצאים בתוכה, את המערך במבנה שמכיל את מספר התקליטים בחנות מכל סוג ואת המערך במבנה שמכיל את מספר התקליטים שנקנו מכל סוג. נציין כי הפונקציה לא אחראית על שחרור מבנה הנתונים מסוג Union Find היות שפונקציית ההורס שלו היא האחראית על שחרור הזיכרון שהוא מקצה (עם זאת, הקריאה האוטומטית לפונקציית ההורס של מבנה זה מתבצעת בסיבוכיות זמן של $O(m)$).

על מנת לשחרר את עץ חברי המועדון נבצע את הפעולות הבאות:
מחיקת האובייקטים שנמצאים בתוך העץ ומחיקת העץ עצמו (הזיכרון של החוליות עצמן שמהן מורכב העץ). על מנת לתמוך בפעולה הראשונה נקצה זיכרון נוסף עבור מערך שאיבריו מטיפוס *MemberClient וגודלו כמספר חברי המועדון בחנות. נייצא את כל האיברים שבתוך העץ לתוך המערך החדש באמצעות מעבר על כל איברי העץ (כרוך בסיבוכיות של $O(number\_of\_members)$ שזהו $O(n)$ במקרה הגרוע).
לאחר מכן נעבור איבר- איבר במערך ונשחרר אותו מהזיכרון. לבסוף נמחק את המערך החדש שיצרנו.
בסוף פונקציה זו יקרא באופן אוטומטי ההורס של העץ והוא זה שידאג לשחרר את הזיכרון של העץ על ידי מעבר מסוג postorder על כל חוליות העץ.

על מנת לשחרר את טבלת הערבול נשתמש בשיטה דומה- נקצה זיכרון נוסף עבור מערך שאיבריו מטיפוס *Client וגודלו כמספר הלקוחות בחנות. נייצא את כל האיברים שבתוך הטבלה לתוך המערך החדש באמצעות מעבר על כל איברי העץ שבכל תא במערך (כרוך בסיבוכיות של $O(n)$ כיוון שזהו מספר האיברים הכולל בעצים של כל תאי המערך). לאחר מכן נעבור איבר- איבר במערך ונשחרר אותו מהזיכרון. לבסוף נמחק את המערך החדש שיצרנו.

את שני המערכים הנוספים נשחרר ישירות באמצעות מספר קבוע של פעולות ולכן בסך הכול סיבוכיות זמן הריצה של פונקציה זו היא $O(n) + O(n) + O(m) = O(n+m)$.
זיכרון ה"עזר" בו הפעולה משתמשת הינו בסדר גודל של $O(n)$.

StatusType newMonth (int *records_stocks, int number_of_records) – הפונקציה מאפסת את התקליטים בחנות ואת סכום הכסף שכל חבר מועדון בחנות חייב.

כדי לעשות זאת, הפונקציה קוראת לפעולה המתאימה בעץ ה-AVL שאחראית על איפוס השדות שמגדירים את הפרס לו כל חבר מועדון זכאי (פעולה זו עוברת על כל הצמתים בעץ בסיבוכיות זמן ריצה של $O(n)$ במקרה הגרוע).

לאחר מכן, הפונקציה עוברת על כל האובייקטים בעץ (כלומר על כל חברי המועדון בחנות) באופן דומה למתואר בפעולת ההורס של המבנה (על ידי יצירת מערך מתאים, ייצוא האיברים מהעץ למערך ומעבר עליהם איבר-איבר) ומאפסת את השדה בו שמור החוב של כל חבר מועדון בחנות. כל זאת בסיבוכיות זמן ריצה של $O(n)$ במקרה הגרוע.

בשלב הבא הפונקציה ניגשת לאיפוס התקליטים בחנות. לשם כך היא קוראת לפעולת האתחול של מבנה הנתונים Union Find (זו כוללת מחיקה של קבוצות התקליטים מהחודש הקודם אם הן קיימות ויצירת "קבוצה" חדשה עם הערכים המתאימים לכל תקליט- מערך מאותחל של איברים מטיפוס Record, בסיבוכיות של $O(m)$).
כדי להשלים את הפעולה הפונקציה יוצרת מערכים חדשים עבור מספר התקליטים מכל סוג בחנות ומספר הרכישות מכל סוג של תקליטים (מוחקת את הישנים אם הם קיימים) ומאתחלת אותם בסיבוכיות זמן ריצה של $O(m)$.

לכן, סך הכול סיבוכיות זמן הריצה היא $O(n+m)$.
זיכרון ה"עזר" בו הפעולה משתמשת הינו בסדר גודל של $O(n+m)$.

StatusType addCostumer (int c_id, int phone) – הפונקציה יוצרת אובייקט מטיפוס Client עם הערכים המתאימים (סיבוכיות זמן ריצה $O(1)$), מוודאת שאינו נמצא במערכת ($O(1)$ ממוצע על הקלט), מכניסה אותו לטבלת הערבול במקרה שכל הערכים תקינים ($O(1)$ משוערך בממוצע על הקלט- הכנסה לטבלת ערבול כמפורט לעיל) ומעדכנת את מספר הלקוחות בחנות ($O(1)$). סך הכול סיבוכיות זמן הריצה היא $O(1)$ משוערך בממוצע על הקלט.

Output_t<int> getPhone (int c_id) – על מנת לגשת ללקוח הרלוונטי בחנות, הפונקציה יוצרת ראשית לקוח עם הערכים המתאימים ($O(1)$), בודקת אם הוא נמצא במערכת ומאתרת אותו במקרה שכן ($O(1)$ בממוצע על הקלט- גישה לאיבר בטבלת ערבול), ומחזירה את מספר הטלפון שלו במקרה שכל הערכים תקינים.
סך הכול סיבוכיות זמן הריצה היא $O(1)$ בממוצע על הקלט.

StatusType makeMember (int c_id) - על מנת לגשת ללקוח הרלוונטי בחנות, הפונקציה יוצרת ראשית לקוח עם הערכים המתאימים ($O(1)$), בודקת אם הוא נמצא במערכת ומאתרת אותו במקרה שכן ($O(\log n)$ במקרה הגרוע- גישה לאיבר בטבלת ערבול כפי שפורט לעיל), ומעדכנת את השדה הרלוונטי בו כך שיצביע על כך שהוא חבר מועדון ($O(1)$).

בנוסף, הפונקציה יוצרת עצם מטיפוס של חבר מועדון עם הערכים הרלוונטיים ומוסיפה אותו לעץ חברי המועדון לאחר שווידאה שלא היה חבר מועדון קודם לכן ($O(\log n)$ במקרה הגרוע).

לבסוף היא מעדכנת את מספר חברי המועדון בחנות ($O(1)$).
סך הכול סיבוכיות זמן הריצה היא $O(\log n)$ במקרה הגרוע.

Output_t<bool> isMember (int c_id) – על מנת לגשת ללקוח הרלוונטי בחנות, הפונקציה יוצרת ראשית לקוח עם הערכים המתאימים ($O(1)$), בודקת אם הוא נמצא במערכת ומאתרת אותו במקרה שכן ($O(1)$ בממוצע על הקלט- גישה לאיבר בטבלת ערבול), ומחזירה האם הוא חבר מועדון במקרה שכל הערכים תקינים (אנו מתחזקים שדה ששומר את נתון זה ולכן ניתן לגשת למידע זה בסיבוכיות של $O(1)$).
סך הכול סיבוכיות זמן הריצה היא $O(1)$ בממוצע על הקלט.

StatusType buyRecord (int c_id, int r_id) – על מנת לבצע רכישה של לקוח, הפונקציה תבדוק את ערכי ההכנסה ואת תקינותם ($O(1)$), לאחר מכן הפונקציה בודקת אם הלקוח נמצא במערכת ומאתרת אותו במקרה שכן ($O(1)$ בממוצע על הקלט- גישה לאיבר בטבלת ערבול ו- $O(\log n)$ במקרה הגרוע).
במידה והלקוח הוא חבר מועדון הפונקציה גם מוצאת את הלקוח בתוך עץ ה- Avl שמחזיק את חברי המועדון- $O(\log n)$ (כי אנו שומרים את החובות של חברי המועדון בשדה בחוליות הללו), ולאחר מכן מעדכנת את החוב המעודכן שלו ואת כמות התקליטים שנקנו מסוג זה. סך הכול סיבוכיות זמן הריצה היא $O(\log n)$.

StatusType addPrize (int c_id1, int c_id2, double amount) – תחילה, נבדוק את ערכי הכניסה ותקינותם ($O(1)$). כאמור, אנו מתחזקים עץ avl עם שמורה – כמות הכסף בפרסים שקיבל כל חבר מועדון. ולכן במידה וצריך לחלק פרסים לכל השחקנים בעלי id בין c_id1 לבין c_id2 אז אנחנו נעבור

מגישים : אברהם יבדייב ת.ז. 211904032
נדב לוי ת.ז. 322298795

בעץ ונוסיף לכל הצמתים עם c_id < c_id2 את הפרס –amount בשדה extra של כל חולייה בעץ - פעולה זו תיקח לנו $O(log\ n)$ עד אשר נגיע לסוף העץ או לצומת עם id = c_id2. לאחר מכן, באופן אנלוגי – נעבור על העץ ונחסיר מכל הצמתים עם id>=c_id1
את הפרס – כלומר נחסיר מהשדה extra את הערך amount, פעולה זו כרוכה גם כן בסיבוכיות של $O(log\ n)$. ובסה"כ פעולה זו כרוכה בסיבוכיות של $O(log\ n)$ כנדרש.

(int c_id) Output_t<double> getExpenses – על מנת לגשת למידע של הלקוח הרלוונטי בחנות, הפונקציה יוצרת ראשית לקוח עם הערכים המתאימים ($O(1)$), בודקת אם הוא נמצא במערכת ומאתרת אותו במקרה שכן ($O(1)$ בממוצע על הקלט- גישה לאיבר בטבלת ערבול ו- $O(\log n)$ במקרה הגרוע), ברגע שהיא קיבלה אותו היא בודקת אם הוא חבר מועדון $O(1)$ במידה ולא היא מסיימת את התוכנית כי אז המערכת אינה שומרת ומתחזקת את החובות של הלקוח הזה. במידה והלקוח הוא חבר מועדון אז הפונקציה גם מוצאת את הלקוח בתוך עץ ה- Avl שמחזיק את חברי המועדון $O(\log n)$ – ומחזירה את החוב שלו פחות כמות הפרסים שהוא קיבל. סך הכול סיבוכיות זמן הריצה היא $O(\log n)$ במקרה הגרוע.

(int r_id1, int r_id2) StatusType putOnTop – תחילה, הפונקציה מוודאת את תקינותם של ערכי הכניסה ומוודאת שאכן יש תקליטים עם אינדקסים כאלו ($O(1)$), במידה וכן, הפונקציה בודקת אם שני התקליטים כבר נמצאים בערמה משותפת – היא עושה זאת באמצעות שתי קריאות לפונקציה find שמאפיינת מבנה מסוג unionFind. אנחנו גם מצמצמים מסלולים בכל קריאה לפונקציה find ולכן הסיבוכיות המשוערכת של שתי פעולות אלו היא $O(\log^* m)$ כאשר m זה מספר התקליטים במבנה.
במידה והתקליטים אינם נמצאים בערמה משותפת אזי הפונקציה גם מאחדת בין הערמות בסיבוכיות של $O(1)$. נציין כי בכל שלב אנו גם מתחזקים את השדות של הגבהים של כל תקליט ומעדכנים אותם במידת הצורך (כרוך בסיבוכיות של $O(\log^* m)$ היות שמדובר במספר קבוע של פעולות find על העץ). סה"כ סיבוכיות זמן הריצה של פונקציה זו היא $O(\log^* m)$ כנדרש.

(int r_id, int *column, int *hight) StatusType getPlace – תחילה, הפונקציה מוודאת את תקינותם של ערכי הכניסה ומוודאת שאכן יש תקליט עם אינדקס כזה ($O(1)$), במידה והקלטים תקינים, הפונקציה מחפשת את השורש של הערמה בו נמצא התקליט עם r_id, היא מחשבת בדרך ממנו אל השורש את הגובה של התקליט באמצעות סכימת המסלולים – כרוך בסיבוכיות של $O(\log^* m)$.
על מנת לקבל את העמודה בה נמצאת הערמה של התקליט הנוכחי אנחנו מוצאים את השורש של הערמה בה נמצא התקליט וניגשים אל השדה column שלו, שגם אותו אנו מתחזקים.
נציין למען הסדר הטוב כי אמנם אנחנו מדברים על הערמות בתור עצים עם המונח של השורש – אך זה רק לצורכי נוחות ובהירות ההסבר, ולמעשה כפי שציינו הערמות ממומשות ע"י מערכים.
סה"כ סיבוכיות זמן הריצה של פונקציה זו היא $O(\log^* m)$ כנדרש.