מיה מרום mayamarom 209162544
קשת ברנדט keshetbrandt 207004078

קובץ תיעוד: תרגיל מעשי 2 – קורס מבני נתונים

<u>חלק מעשי – מימוש ערימת Fibonacci בשפת JAVA</u>:

את ההסברים על הפונקציות השונות וסיבוכיות זמן הריצה שלהן נפריד ל3 קטגוריות – הפונקציות שהתבקשנו לממש, פונקציות נוספות שאנחנו הוספנו למחלקה, פונקציות בזמן ריצה קבוע ושדות המחלקות.

1. פונקציות שהתבקשנו לממש:

- **isEmpty(): סיבוכיות זמן במקרה הגרוע O(1), אמורטייז O(1)**

הפונקציה מחזירה true אם הערמה הנוכחית ריקה. כלומר, אם הגודל שלה אפס. הגישה לשדה size נעשית בזמן קבוע ולכן – סיבוכיות הזמן היא O(1).

- **insert(int key): סיבוכיות זמן במקרה הגרוע O(1), אמורטייז O(1)**

הפונקציה מכניסה צומת חדש (מפתח שמתקבל כקלט) לערמה. היא מכניסה את הצומת כשורש חדש (השמאלי ביותר) בערמה ע"י עדכון מצביעי הprev,next של הצומת האחרון והראשון בערימה. אם הערימה הייתה ריקה, היא מכניסה את הצומת כך שיצביע על עצמו ומעדכנת את כל השדות בהתאם. אם המפתח החדש שהוכנס קטן מהמפתח המינימלי, מצביע המינימום מעודכן אליו. כמו כן, מצביע הfirst מעודכן אליו שכן הוא העץ החדש ביותר בערמה. סיבוכיות הזמן היא O(1) שכן מתבצעות מספר קבוע של עדכון שדות ומצביעים.

- **deleteMin(): סיבוכיות זמן ריצה במקרה הגרוע O(n), אמורטייז O(log n)**

הפונקציה מוחקת את האיבר המינימלי ביותר בערימה. ראשית הפונקציה בודקת האם הערימה ריקה או בגדול 1, אם כן היא מעדכנת את כל השדות לשדות ההתחלתיים של ערימה ריקה. אחרת, היא בודקת האם המינימום שמוחקים (אליו ניגש בעזרת המצביע מינימום של הערמה) הוא בעל ילדים. אם אין לו ילדים, ההסרה של השורש נעשית בעזרת עדכון מצביעי הprev,next של הצמתים השכנים שלו כך ש"ידלגו" עליו. אחרת, נהפוך כל אחד מילדיו לשורש בפני עצמו על ידי שינוי ההורה שלו לnull ועדכון מצביעי הprev,first של הילד הראשון והאחרון לצמתיו השכנים. כמו כן, היא הופכת את הסימון של השורשים החודשים לfalse במידה והיו מסומנים (שכן שורשים אינם מסומנים). לאחר מכן מתבצעת קריאה לsucLink (שגם מעדכנת את מצביע המינימום למינימום החדש) שסיבוכיות הזמן שלה היא O(n) כפי שהסברנו לעיל. כמובן שמתבצעים כל העדכונים ההכרחיים לשדות הערמה (גודל, מספר השורשים וכו'). המעבר על ילדי השורש מתבצעת בO(logn) שכן זוהי דרגתו המקסימלית של שורש כלשהו בערמה. ולכן סה"כ סיבוכיות הזמן WC היא O(n). כפי שהסברנו בפונקציה sucLink, הסיבוכיות אמורטייז שלה היא O(log n) ולכן גם כאן הסיבוכיות אמורטייז היא O(log n).

- **findMin(): סיבוכיות זמן ריצה במקרה הגרוע O(1), אמורטייז O(1)**

הפונקציה מחזירה את האיבר המינימלי בערימה. בגלל ששמרנו מצביע לאיבר המינימלי ותחזקנו אותו לאורך כל שאר הפונקציות והפעולות שמפעילים על הערימה, סיבוכיות זמן הריצה היא O(1).

- **meld(FibonacciHeap heap2): סיבוכיות זמן במקרה הגרוע O(1), אמורטייז O(1)**

הפונקציה מחברת בין 2 ערימות פיבונאצ'י. בגלל שבערימת פיבונאצ'י אין הגבלה על מספר העצים מאותה דרגה שיכולים להיות בערימה (ומתקנים את הערימה רק במחיקת איבר המינימום), ניתן פשוט לחבר את שתי הערימות בעזרת שינוי המצביעים ללא צורך בביצוע פעולה מורכבת יותר. יש לבדוק לאיזה מין הערימות יש איבר מינימלי קטן יותר ולשנות את המצביע המינימום בהתאם. כמו כן,

מחברים את ערימה 2 בסופה של ערימה 1 ולכן יש לשנות את המצביע לאיבר הראשון (first), את החיבור של האיבר האחרון בערימה 2 והאיבר הראשון בערימה 1, וכן את האיבר האחרון בערימה 1 והאיבר הראשון בערימה 2. לבסוף, נסכום את השדות של הערימות (roots, size, marks וכו'). כל הפעולות הן פעולות בעלות קבועה O(1), ולכן סיבוכיות הזמן קבועה גם היא.

- **size(): סיבוכיות זמן במקרה הגרוע O(1), אמורטייז O(1)**

הפונקציה מחזירה את גודל הערמה בעזרת שדה הsize שאנו מתחזקים. גישה אליו לוקחת זמן קבוע.

- **countersReps(): סיבוכיות זמן במקרה הגרוע O(n), אמורטייז O(log n)**

הפונקציה מחזירה מערך שמייצג עבור כל דרגה בערמה, כמה שורשים יש מהדרגה הזו. ראשית הפונקציה בודקת האם הערמה ריקה, אם כן היא מחזירה מערך ריק. אחרת היא עוברת על שורשי העצים בערימה ועבור כל שורש מעלה את הקאונטר של הדרגה שלו. כמו כן, שמרנו אינדקס bigI שמייצג את האינדקס של הדרגה הגדולה ביותר בעץ, כך שנוכל להחזיר את המערך בגודל הדרגה הגבוהה ביותר בפועל בעץ. סיבוכיות הזמן במקרה הגרוע היא O(n) שכן נעבור בלולאה על כל שורשי העץ ובמקרה הגרוע כל צמתי העץ יהיו שורשים. בהרצאה ראינו כי בממוצע יש בערימת פיבונאצ'י logn עצים ולכן סיבוכיות אמורטייז היא O(log n).

- **delete(HeapNode x): סיבוכיות זמן במקרה הגרוע O(n), אמורטייז O(log n)**

הפונקציה מוחקת את האיבר שמתקבל כקלט. היא עושה זאת בעזרת שתי הפונקציות decrease key, delete min שהסברנו לעיל. לכן, סיבוכיות הזמן שלה במקרה הגרוע היא O(n) וסיבוכיות האמורטייז שלה היא O(log n).

- **decreaseKey(HeapNode x, int delta): סיבוכיות זמן במקרה הגרוע O(n), אמורטייז O(1)**

הפונקציה מורידה את הערך של צומת x שמקבלת כקלט בערך delta החיובי שמתקבל. ראשית היא בודקת האם הdelta שהתקבלה כקלט היא הערך המקסימלי ביותר, אם כן היא הופכת את הערך של x להיות המינימלי ביותר והופכת את המצביע מינימום של הערמה להצביע על x. אחרת, היא מורידה את הערך של x בdelta ומעדכנת את המצביע מינימום רק אם הערך החדש של x נמוך מהערך של הצומת עליו המינימום מצביע. לאחר מכן, אם לx יש הורה והערך של ההורה שלו כעת גדול מהערך של x, היא מבצעת קריאה לcasCut שהסברנו לעיל. כל הפעולות מלבד ה cascading cuts מתבצעות בזמן קבוע. לכן, סיבוכיות הזמן במקרה הגרוע הינו כפי שהסברנו לעיל O(n), אמורטייז O(1).

- **nonMarked(): סיבוכיות זמן במקרה הגרוע O(1), אמורטייז O(1)**

הפונקציה מחזירה את מספר הצמתים הלא מסומנים שנמצאים כרגע בערמה. היא עושה זאת על ידי חיסור בין השדות size לבין numMarks שמייצגים את גודל הערמה ומספר המסומנים. גישה לשדות אלו נעשה בזמן קבוע ולכן סיבוכיות הזמן היא O(1).

- **potential(): סיבוכיות זמן ריצה במקרה הגרוע O(1), אמורטייז O(1)**

הפונקציה מחזירה את הפוטנציאל של הערימה לפני הנוסחה שראינו בהרצאה (מספר השורשים ועוד פעמיים מספר הצמתים המסומנים). סיבוכיות הזמן היא O(1) כי דאגנו לתחזק את 2 השדות הרלוונטיים (מספר השורשים בערימה, מספר הצמתים המסומנים בערימה) לאורך כל הפונקציות שנמצאות במחלקה ולכן עבור פונקציה זו יש רק להחזיר את תוצאת המשוואה (חישוב בעלות O(1)).

- **totalLinks(): סיבוכיות זמן במקרה הגרוע O(1), אמורטייז O(1)**

הפונקציה מחזירה את מספר פעולות הlink (בהתאם לפונקציית שlinkו) שהסברנו לעיל, שנעשו בזמן הריצה הנוכחי. אנו מתחזקים שדה סטטי ששומר את הערך הנ"ל ולכן גישה אליו לוקחת זמן קבוע.

- **totalCuts(): סיבוכיות זמן ריצה במקרה הגרוע O(1), אמורטייז O(1)**

הפונקציה מחזירה את מספר פעולות ה cut שבוצעו בערימה (מהיצירה של הערימה ועד הקריאה לפונקציה). סיבוכיות הזמן היא O(1) משום שדאגנו לתחזק משתנה סטטי שסופר את מספר פעולות ה cut לאורך הריצה של התוכנית. פעולות cut היא פעולה של ניתוק צומת מאביו (בין אם זה נעשה בפונקציית cut שיצרנו או במקרה של מחיקת צומת והפיכת בניו לשורשים) לכן, הפונקציה מחזירה את הערך הסטטי הנוכחי, עלות קבועה של O(1).

- **kMin(FibonacciHeap H, int k): סיבוכיות זמן ריצה במקרה הגרוע O(k log n), אמורטייז O(k log n)**

הפונקציה מוצאת את k האיברים הקטנים ביותר בערימת פיבונאצ׳י (שמכילה עץ בינומי אחד) ומחזירה אותם במערך ממוין בסדר עולה. הפונקציה משתמשת בערימת מינימום כמבנה נתונים משני למציאת האיברים. הפונקציה מפעילה לולאת for בעלת k-1 איטרציות (האיבר הראשון שנכניס למערך הוא השורש של העץ שכן הוא המינימלי ולכן נותרו k-1 איברים להכניס למערך). לפי מבנה העץ, האופציות לאיבר המינימלי הבא בכל איטרציה הן אחד מהילדים של הצומת המינימלי הנוכחי או אחד מהאחים שלו. ולכן בכל איטרציה, נבדוק האם לצומת עליו אנחנו עומדים (נקרא minNode) יש ילדים או לא. במידה וכן, הפונקציה מוסיפה את כל הילדים לערימת המינימום (האחים של הצומת כבר נמצאים בערימה מאיטרציה קודמת). במידה ואין לו ילדים, כל המועמדים לאיבר המינימום כבר נמצאים בערימה. לאחר ההכנסות לערימת המינימום, נמצא את האיבר המינימלי בערימת העזר ונשמור את המצביע אליו בminNode (הוספנו שדה לכל אחד מהצמתים שנקרא pointer ומצביע על מיקום האיבר בערימה המקורית ולכן נוכל לגשת אליו). נוסיף אותו למערך ונמחק אותו מערימת העזר. לאחר מכן נחזור על כל התהליך באיטרציה נוספת של לולאת הfor (כאשר minNode הוא הבן של האיבר שהרגע הוספנו למערך). סיבוכיות הזמן היא O(k log n) עבור k איטרציות של לולאת הfor, בה בכל שלב נעבור על כל הבנים של הצומת בלולאת while (שבמקרה הגרוע deg(H)=log n כי זו הדרגה המקסימלית של שורש בעץ בינומי תקין בעל n צמתים).

2. פונקציות שאנחנו הוספנו למחלקה:

- **link(HeapNode root1, HeapNode root2): סיבוכיות זמן ריצה במקרה הגרוע O(1), אמורטייז O(1)**
הפונקציה מחברת שני עצים בינומיים לעץ בינומי אחד. ראשית, הפונקציה דואגת לכך שroot1 יהיה בעל המפתח הנמוך מבין שני השורשים שקיבלנו (לצורך נוחות ולמניעת שכפול קוד). לאחר מכן הפונקציה דואגת לסדר את המצביעים, הן של השורש של העץ המאוחד root1, והן את המצביעים של הבנים של root1 ובפרט הבן החדש root2. הפונקציה פועלת בסיבוכיות O(1) שכן היא מבצעת רק פעולות בעלות קבועה – שינוי מצביעים ועדכון השדות הרלוונטיים שהשתנו (rank עבור הצמתים, links ו-roots עבור הערימה).

- **sucLink(): סיבוכיות זמן ריצה במקרה הגרוע O(n), אמורטייז O(log n)**
הפונקציה מתקנת את הערימה להיות ערימה תקינה. כלומר, עד עץ בינומי אחד בכל דרגה (כאשר הדרגה המקסימלית היא logn כאשר n מסמל את מספר הצמתים בכל העצים בערימה). הסיבוכיות במקרה הגרוע היא O(n) משום שהפונקציה עוברת על כל השורשים בערימה בלולאת for, ובמקרה הגרוע כל הצמתים הם שורשים בעלי דרגה 0. בכל איטרציה של לולאת הfor מתבצעת עבודה בעלות קבועה (פונקציית link גם היא בעלת סיבוכיות O(1), ובנוסף מתבצעת לולאת while (מקוננת בתוך הfor) שאחראית על איחודי עצים עד שנגיע לעץ בעל דרגה ייחודית (דרגה שלא קיימת בערימה). לולאת הwhile רצה בכל פעם מספר איטרציות משתנה (בהתאם למצב הסל) שאמנם יכול להגיע עד log n אך בוודאות לא יהיה כך בכל איטרציה. כפי שראינו בכיתה, לאחר איטרציה "יקרה" שכזו, בהכרח יהיו מספר איטרציות בה הלולאה תרוץ מספר קבוע של פעמים. לכן בסך הכל, מספר האיטרציות בהן לולאת הwhile רצה log n פעמים הינו סופי כך שסיבוכיות הזמן במקרה הגרוע נשארת O(n). בהתאם לפונקציית הפוטנציאל שראינו בכיתה, סיבוכיות האמורטייז היא O(log n).

- **cut(HeapNode x, HeapNode parent): סיבוכיות זמן במקרה הגרוע O(1), אמורטייז O(1)**
הפונקציה מבצעת את הניתוק בין צומת להורה שלו. היא עושה זאת על ידי שינוי המצביעים (הן של ההורה והן של הבן). כמו כן, הפונקציה דואגת לשנות את הדרגה של ההורה (ירדה ב-1) וכן לוודא שהצומת x, שהופך להיות שורש לא יהיה מסומן (שורשים אף פעם לא מסומנים). נעדכן את הקאונטר שסופר את מספר פעולות הcut שבוצעו בערימה. מספר השורשים עלה גם כן ולכן מעדכנים את השדה הזה של הערימה. את הצומת שניתקנו וכעת פך להיות עץ עצמאי נמקם ראשון בסדר העצים בערימה. כל המצביעים של next, prev מעודכנים בהתאם (אצל x ואצל parent). כפי שניתן לראות, הפונקציה מבצעת רק פעולות בעלות קבועה ואינה כוללת חישוב/מעבר שקשור למספר הצמתים בעץ ולכן סיבוכיות הזמן היא O(1).

- **casCut(HeapNode x, HeapNode parent) סיבוכיות זמן במקרה הגרוע O(n), אמורטייז O(1)**
הפונקציה מבצעת את שרשרת הניתוקים הנדרשת לאחר ניתוק של צומת מאביו. בכל שלב, היא בודקת האם צומת האב מסומנת או לא. במידה ולא, הפונקציה מסמנת אותה ומסיימת את פעולתה. במידה והצומת מסומנת, נדרש חיתוך נוסף והפונקציה מבצעת זו. הפעולות שנכללות בפונקציה זו כולן בעלות קבועה (כולל cut) ולכן סיבוכיות זמן הריצה נגזרת ממספר הקריאות הרקורסיביות של הפונקציה. במקרה הגרוע, ערימת הפיבונאצ׳י עלולה לכלול עץ שהוא בעצם שרשרת לינארית של צמתים באורך n. במקרה זה הפונקציה תיאלץ לטייל לאורך גובה השרשרת ובכל שלב לבצע cut. לכן, סיבוכיות הזמן היא O(n). עם זאת, זהו מקרה מאוד נדיר וחריג וכפי שלמדנו, סיבוכיות זמן הריצה amortized היא O(1) – וזאת על פי טענה שלמדנו בכיתה שלפיה כאשר נבצע d פעולות של decreseKey, בהן יתבצעו פעולות חיתוך שונות, סך כל הפעולות יהיה לכל היותר 2d פעולות.

3. פונקציות get/set ונוספות (סיבוכיות זמן קבועה O(1)):

- **getKey()**: הפונקציה מחזירה את המפתח של הצומת עליו הופעלה הפונקציה.
- **getMarked()**: הפונקציה מחזירה האם הצומת שעליו הופעלה הפונקציה מסומן (יחזיר true) או לא.
- **getChild()**: הפונקציה מחזירה את הצומת שהיא הבן השמאלי ביותר של הצומת עליו הופעלה הפונקציה או null אם אין לצומת ילדים.
- **getParent()**: הפונקציה מחזירה את הצומת שהיא ההורה של הצומת עליו הופעלה הפונקציה או null אם הוא שורש.
- **getPrev()**: הפונקציה מחזירה את הצומת הקודם (הצומת מצד שמאל) של הצומת עליו הופעלה הפונקציה או את הצומת עצמו אם אין לצומת זה אחים.
- **getNext()**: הפונקציה מחזירה את הצומת הבא (הצומת מצד ימין) של הצומת עליו הופעלה הפונקציה או את הצומת עצמו אם אין לצומת זה אחים.
- **getRank()**: הפונקציה מחזירה את הדרגה (מספר הבנים) של הצומת עליו הופעלה הפונקציה.
- **setKey(int key)**: הפונקציה משנה את המפתח של הצומת להיות key.
- **setMarked(Boolean marked)**: הפונקציה משנה את שדה marked של הצומת להיות לפי הקלט.
- **setPrev(HeapNode prev)**: הפונקציה קובעת את הצומת הקודם של הצומת עליו הופעלה הפונקציה להיות prev (כלומר, הצומת prev יהיה משמאל לצומת this).
- **setNext(HeapNode next)**: הפונקציה קובעת את הצומת הבא של הצומת עליו הופעלה הפונקציה להיות next (כלומר, הצומת next יהיה מימין לצומת this).
- **setParent(HeapNode parent)**: הפונקציה תקבע את הצומת parent להיות ההורה של הצומת עליו הופעלה הפונקציה.
- **setChild(HeapNode child)**: הפונקציה תקבע את הצומת child להיות הבן השמאלי ביותר של הצומת עליו הופעלה הפונקציה.
- **setRank(int rank)**: הפונקציה תשנה את הדרגה של הצומת עליו הופעלה הפונקציה להיות rank.
- **getFirst()**: הפונקציה תחזיר את העץ האחרון שנוסף לערימה (העץ השמאלי ביותר).
- **משתנים ושדות שיצרנו במחלקת FibonacciHeap:**
  - first – מצביע על הצומת (השורש) האחרון שהוספנו לערמה.
  - Min – מצביע על הצומת המינימלי ביותר בערמה.
  - Roots – שומר את מספר השורשים שקיימים בערמה.
  - Size – שומר את מספר הצמתים הכולל שנמצאים בערמה.
  - numMarks – שומר את מספר הצמתים המסומנים בערמה.
  - Links – משתנה סטטי ששומר את מספר פעולות הלינק שנעשו בערימה בזמן הריצה הנוכחי.
  - Cuts – משתנה סטטי ששומר את מספר החיתוכים שנעשו (ניתוק צומת מאביו) בערימה בזמן הריצה הנוכחי.
- **משתנים ושדות שיצרנו במחלקת HeapNode:**
  - Marked – שדה בוליאני השומר אם הצומת מסומן או לא.
  - Child – מצביע לבן השמאלי ביותר של הצומת (null אם אין לו בנים).

- Parent - מצביע לאבא של הצומת (null אם אין לו אבא, כלומר אם הוא שורש).
- Prev – מצביע לצומת הקודם של הצומת (מצביע על עצמו במידה ואין צומת קודם).
- Next - מצביע לצומת העוקב של הצומת (מצביע על עצמו במידה ואין צומת עוקב).
- Rank – שומר את דרגת הצומת, כלומר את מספר הבנים שלו.
- Pointer – שדה עזר שמשמש אותנו לטובת הפונקציה kMin. מחוץ לפונקציה זו הוא מאותחל להיות null ואין לו שימוש. בפונקציה kMin הוא משמש אותנו כמצביע לעצמו מערימת המינימום לערימת H המקורית.

mayamarom 209162544 מיה מרום
keshetbrandt 207004078 קשת ברנדט

חלק ניסויי-תיאורטי:

שאלה מס׳ 1:

א. נוכיח כי זמן הריצה האסימפטוטי של סדרת הפעולות כפונקציה של m הוא $O(m)$.

אנו מבצעים $m+1$ פעולות insert, שמבוצעות בצורה עצלה (כלומר, כל צומת מתווסף כעץ בעל צומת יחיד לשאר השורשים בערימה). רק לאחר שאנו מבצעים delete-min, אנו מבצעים successive-linking ויוצרים עץ בינומי (במקרה הזה, זה יהיה עץ בינומי מלא לאור העובדה ש-m הוא חזקה של 2). פעולה זו הינה בסיבוכיות $O(m)$. לאחר מכן, אנו מבצעים log m פעולות decrease-key. בכל פעולה כזו, אנו מבצעים decrease-key לצומת שהמפתח שלו הוא עלה, ולכן אין צורך לבצע cascading-cuts.

נוכיח זאת: בפעולת decrease-key אנו מבצעים successive-linking. העץ הראשון הוא 0, לאחר מכן 1, וכן הלאה, והאחרון הוא m-1 (כולם עצים שמכילים צומת בודד). בכל איטרציה, נאחד זוג צמתים עוקבים, ולכן הצומת בעל המפתח האי-זוגי (שגדול מהזוגי) הוא העלה. בנוסף, מהגדרת עץ פיבונאצ׳י (ובינומי), עץ מדרגה k+1 משמעו שלשורש יש k ילדים בדרגות 0,...,k. מכאן ניתן להבים כי לכל צומת שאינו עלה, יש בן אחד בלבד שהוא עלה. כאשר אנו מבצעים cut לעלה, אנו מורידים לכל היותר בן אחד מההורה שלו, ולכן אין צורך בביצוע cascading-cuts, אלא רק בסימון הצומת. לסיכום, ביצענו $\log m$ פעולות cut בסיבוכיות $O(1)$, ולכן בסה״כ קיבלנו בלולאה השנייה $O(\log m)$, והסיבוכיות הכוללת היא $O(m)$.

ב.

m	Run-Time (ms)	totalLinks	totalCuts	Potential
2^5	1	31	5	14
2^{10}	2	1,023	10	29
2^{15}	13	32,767	15	44
2^{20}	128	1,048,575	20	59

ג. **פעולות link**: לאחר פעולות ה-insert, נותרנו עם בדיוק m עצים מדרגה 0, כלומר חזקה של 2, ולכן לאחר שנבצע successive-linking, יתקבל עץ בינומי יחיד מדרגה log m. בכל פעולת link אנו מקטינים את מספר העצים בערימה ב-1. כלומר, אם לאחר פעולות successive-linking נותרנו עם עץ יחיד (שבמקור היו m), ניתן להבין כי בוצעו m-1 פעולות link. לאחר מכן, מתבצעות log m פעולות decrease-key, שעבורן אנו לא מבצעים successive-linking ולכן לא מתבצעות פעולות link נוספות.

פעולות cut: מתבצעות בקוד log m פעולות decrease-key. כפי שהוכחנו בסעיף א׳, כל פעולה כזו מורכבת מפעולת cut יחידה, ולכן בסך הכל אנו מבצעים log m פעולות cut.

פוטנציאל הערימה: ניזכר כי פונקציית הפוטנציאל היא: Potential = #trees + 2 × #marked.

מס׳ העצים בערימה הוא log m + 1, זאת משום שלאחר successive-linking נותרנו עם עץ 1, ולאחר מכן נוספו עוד log m עצים מדרגה בודדת (לאחר סדרת פעולות cut). מס׳ הצמתים המסומנים

הוא $\log m - 1$, מפני שביצענו log m פעולות decrease-key, אך אחת הייתה על צומת שההורה שלו הוא השורש, ולכן לא סומן. בסה״כ : $\text{Potential} = \log m + 1 + 2 \times (\log m - 1) = 3\log m - 1$.

ד. אם בשורה 3# נבצע decreaseKey על המפתחות $m - 2^i$ (בלי ה-״1+״), בכל פעולה נבצע decrease-key לבן הימני ביותר (הגדול ביותר) ברמה $\log m - i$. נוכיח זאת באינדוקציה על הדרגה.
<u>בסיס האינדוקציה:</u> עבור צומת מדרגה 0, מתקיים באופן טריוויאלי.
<u>צעד האינדוקציה:</u> נניח שהטענה נכונה עבור כל דרגה שקטנה מ-k. נוכיח את נכונות הטענה עבור בדרגה ה-k. לשורש בעל דרגה k-1 , קיימים k-1 בנים. נזכיר כי הכנסנו את הצמתים בסדר יורד (1, ... ,k-1), ולכן הבן השמאלי ביותר (1) הוא שורש של עץ מדרגה k-1, וההורה שלו הוא שורש של עץ שמכיל 2^{k-1} איברים, ולכן הוא האיבר ה- $2^k = m - 2^i$.
ראשית, נבצע decrease-key לשורש. לאחר מכן, נבצע decrease-key לבן הגדול ביותר שלו, אך מכיוון שכבר ביצענו decrease-key להורה שלו ואנו מקטינים אותו באותה delta לא יבוצע cut (ההפרש בין המפתחות נותר זהה, ולפני פעולת decrease-key ההורה היה קטן מהבן ולכן זה נותר גם לאחר השינוי).
פעולות link : לאחר סדרת פעולות ה-insert, נותרנו עם בדיוק m עצים מדרגה 0 (כלומר חזקה של 2).
לכן לאחר ביצוע successive-linking נקבל עץ בינומי אחד מדרגה log m. בכל פעולת link מספר השורשים בערימה קטן ב-1. מכאן, לאחר ביצוע successive-linking מספר העצים בערימה קטן מ-m עצים לעץ בודד. לאור כל, ניתן להסיק כי בוצעו m-1 פעולות link. לבסוף, אנו מבצעים log m פעולות decrease-key, שעבורם לא נבצע cut, ולכן לא יבוצעו פעולות link נוספות.
פעולות cut : כפי שתיארנו, לא נבצע פעולות cut.
פוטנציאל הערימה : יש עץ אחד בודד בערימה, מפני שלאחר successive-linking נותרנו עם עץ 1 ולא ביצענו פעולות cut. מכאן שאין לנו צמתים מסומנים.לכן $\text{Potential} = 1$.

ה. אם נמחק את שורה 2# (לא נבצע delete-min), לא נבצע successive-linking ובפרט לא נבצע שום פעולת link. לכן, בכל פעולת decrease-key, בסך-הכל נחסיר m+1 מכל מפתח של צומת, ולא יהיה צורך בביצוע פעולות cut (ההפרש בין המפתחות של הורה ובן נותר זהה).
פעולות link : לא נבצע פעולות link.
פעולות cut : לא נבצע פעולות cut.
פוטנציאל הערימה : מס׳ העצים בערימה הוא m+1, זאת משום שביצענו successive-linking ולא מחקנו את המינימום. אין צמתים מסומנים כי לא בוצעו שום פעולות cut. בסה״כ $\text{Potential} = m + 1 + 2 \times 0 = m + 1$.

ו. לאחר הוספת השורה ״decreaseKey(m-2,m+1)״ וביצוע כלל הפעולות המתוארות בסעיף ג׳, נותר בערימה עץ פיבונאצ׳י יחיד בעל שרשרת של צמתים ימניים מסומנים (מפני שמחקנו את הבן השמאלי של כל צומת ימני). כעת, בפעולת decreaseKey(m-2,m+1), אנו מבצעים עדכון למפתח של הבן של השורש בעל דרגה $(\log m - 1)$ ומנתקים בניהם. לאחר מכן, מכיוון שאביו מסומן- ננתק את אביו.
כך נבצע עוד $\log m - 2$ פעמים עד שנגיע לשורש העץ.

פעולות link: נבצע link בדומה לסעיף ג׳.

פעולות cut: לאחר שביצענו את הפעולות בסעיף ג׳, נבצע decreaseKey לעלה הגדול ביותר בעץ וננתק אותו. כפי שתיארנו, ההורה שלו מסומן, ולכן ננתק גם אותו. נבצע תהליך זה עד לשורש (אותו לא ננתק או נסמן), ולכן נבצע בסה״כ עוד $\log \mathrm{m}-1$ פעולות cut.

פוטנציאל הערימה: מס׳ העצים הוא $2\log \mathrm{m}$, מפני שלאחר סעיף ג׳, ביצענו עוד $\log \mathrm{m}-1$ פעולות cut ולכן נוספו עוד $\log m-1$ עצים לערימה. אין לנו צמתים מסומנים, מפני שעשינו cut לכל הצמתים המסומנים. בסה״כ: $\text{Potential} = 2\log m$.

העלות היקרה ביותר של פעולת decreaseKey היא $O(\log m)$.כפי שתיארנו, נבצע decreaseKey לבן של השורש בעל דרגה $(\log m-1)$. הוא בהכרח אינו שורש, ולכן יש צורך בביצוע cascading-cuts. כלל ההורים במסלול מהצומת הנ״ל אל השורש מסומנים, ולכן נקרא רקורסיבית לפונקציה עוד m-1 פעמים (עד לשורש). ובסה״כ נבצע $\log m$ קריאות לרקורסיה, ובכל קריאה נבצע $O(1)$ עבודה.

case	totalLinks	totalCuts	Potential	decKey max cost
original	m-1	log(m)	3log(m) - 1	
decKey(m-2^i)	m-1	0	1	
remove line #2	0	0	m+1	
add line #4	m-1	2log(m) - 1	2log(m)	log(m)

שאלה מס' 2:

א.

	m	Run-Time (ms)	totalLinks	totalCuts	Potential
i=6	728	7	723	0	6
i=8	6,560	15	6,555	0	6
i=10	59,048	75	59,040	0	9
i=12	531,440	383	531,431	0	10
i=14	4,782,968	2,867	4,782,955	0	14

ב. ראשית, אנו מבצעים m+1 פעולות insert אשר מתבצעות ב-$O(1)$ עבודה, בסה"כ $O(m)$. לאחר מכן אנו מבצעים $3m/4$ פעולות *delete-min*. לאחר המחיקה הראשונה, אנחנו נותרים עם m צמתים ונבצע *successive-linking*.נשים לב כי m מתחלק ב-4, ולכן לא ייתכן שנישאר עם עצים מדרגה 0 או 1 (אחרת יישארו לנו עצים נוספים מדרגות אלה ונאחד אותם). מכאן, נסיק כי לאחר ביצוע successive-linking הצומת עם הדרגה המינימלית הוא מדרגה 2. כלומר, נישאר עם לכל הפחות עץ אחד ולכל היותר *logm-2* עצים. לאור העובדה שהכנסנו תחילה את המפתחות הקטנים, המפתחות הגדולים יהיו כעת בראש הערימה. זאת אומרת שכל המפתחות בעץ i שדרגתו קטנה מעץ j, יהיו קטנים מהמפתחות בעץ j. לפיכך, כאשר נבצע *delete-min* נוסף, נמחק את השורש של העץ בעל הדרגה הקטנה ביותר. לכן, במחיקה שלאחר מכן, נמחק את הצומת הקטן ביותר- שהוא עלה ולא נבצע successive-linking . נמשיך כך, ולכן לא נבצע successive-linking עד שנמחק $\frac{3}{4}$ מהצמתים, ולכן זמן הריצה הוא בסה"כ: $O(m) + O(\log m) + 0.75O(m) = O(m)$

ג. **פעולות link**: ניתן להגדיר את מספר פעולות הlink בעזרת הבעת המספר m בביטים (נסמן ב-x). מספר פעולות הlink אם כך הוא m-x. כמות הlink המקסימלית שנבצע היא m-1 וזאת משום שמלכתחילה היו לנו רק m-1 עצים בערימה. x-1 הוא מספר המסמל כמה עצים לא חוברו בסוף תהליך הsuccessive-linking. בעצם, כמות הביטים ה"דולקים" היא ככמות עצי הפיבונאצ'י בערימה. כמות עצים זו מעידה על מספר פעולות הlink שלא בוצעו ולכן אנו מחסרים אותם מהכמות המקסימלית. לכן m-x = m-1-x+1 = m-1-(x-1).

פעולות cut: אנו לא מבצעים פעולות decrease-key ולכן לא נבצע פעולות cut.

הפוטנציאל: ניזכר כי פונקציית הפוטנציאל היא: $\text{Potential} = \#\text{trees} + 2 \times \#\text{marked}$.

מפני שאנו לא מבצעים פעולות cut, אין צמתים מסומנים ומתקיים כי: $\text{Potential} = \#\text{trees}$, ולכן

$$\text{Potential} = \text{t1} - \text{links} = \text{m} + 1 - \text{links}$$