מבני נתונים – תרגיל מעשי 2

מגישים: שי פוקס 313452252

Username: shayakivafux

נתן בלוך 316130707

Username: nathanbloch

תיעוד

המחלקה HeapNode

שדות המחלקה

- **int key –** המפתח של האיבר
- **int rank –** דרגת האיבר
- **boolean mark –** סימון של האיבר, האם הוא מסומן או לא.
  False מסמל אינו מסומן, ואילו true מסמן שאכן מסומן
- **HeapNode child –** האיבר שנמצא מתחת לאיבר הנוכחי
- **HeapNode next –** האיבר שנמצא מימין לאיבר הנוכחי
- **HeapNode prev –** האיבר שנמצא משמאל לאיבר הנוכחי
- **HeapNode parent –** האיבר שנמצא מעל לאיבר הנוכחי
- **HeapNode kminpointer** – נועד למתודת kmin כדי להחזיק מצביע לאיבר בערימה המקורית מוך ערימת המינימום שאנו מתחזקים במהלך המתודה.

בנאי המחלקה

- **HeapNode(int key) –** בנאי המקבל רק את המפתח של האיבר בלבד, ומאתחל את ה-Node.
  אתחול המפתח ל-key שניתן כפרמטר, אתחול דרגתו ל-0, וכן סימונו ל-false – כלומר שאינו מסומן.
  פועל בסיבוכיות זמן- O(1) כיוון שמדובר באתחול שדות בלבד.

מתודות המחלקה

- במחלקה יש **מתודות Setters/Getters** עבור השדות המחלקה. מאפשרות גישה לשדות האיבר ועריכתם של שדות המחלקה HeapNode.
  כל אחת מהן פועלת בסיבוכיות O(1).

המחלקה FibonacciHeap

שדות המחלקה

- **HeapNode startRoot** – מצביע אל השורש הראשון בערימה. מימוש המצביע דומה למימוש הנלמד בהרצאות, וזהו בעצם מצביע אל תחילת הרשימה.
- **HeapNode min** – מצביע אל האיבר המינימלי הערימה.
- **int size** – מספר המייצג את מספר האיברים הכולל בערימה.
- **int numOfRoot** – מספר המייצג את מספר השורשים בערימה. זה בעצם מייצג את מספר העצים בערימה.
- **int numOfMarked** – מספר המייצג את מספר האיברים(ה-Nodes) שהינם מסומנים כרגע.
- **static int totalCuts** – מספר המייצג את כמות הפעמים שביצענו cut. זהו משתנה סטטי, גלובלי, אשר מייצג את **סך כל** הפעמים שבוצע cut.
- **static int totalLinks** – מספר המייצג את כמות הפעמים שביצענו linking בין שני עצים. זהו משתנה סטטי, גלובלי, אשר מייצג את **סך כל** הפעמים שבוצע linking.

בנאי המחלקה

- **FibonacciHeap()** – בנאי ריק בו המאתחל את הערימה הריקה. פועל ב-O(1) כיוון שמדובר באתחול שדות המחלקה בלבד.

מתודות המחלקה

- **boolean isEmpty()** – הפונקציה מחזירה true אם ורק אם הערימה ריקה, משמע יש בה בסה"כ 0 איברים. אם בערימה לפחות איבר אחד אז יוחזר false. פועל ב-O(1) כיוון שמדובר בפעולת השוואה בודדת של שדה המתוחזק באופן קבוע.

- **HeapNode insert(int key)** – פונקציית הכנסת איבר חדש לערימה. הפונקציה מקבלת כפרמטר מפתח לאיבר החדש שיוכנס, בונה Node חדש עם המפתח key. הכנסת איבר חדש לעץ פיבונצ'י נעשית על ידי בניית עץ חדש בעל איבר אחד ויחיד שהוא ה-Node עם המפתח החדש. בהכנסה, בעצם פשוט מוסיפים את העץ החדש לערימה בכך שמוסיפים את השורש של העץ החדש לרשימה השורשים. פעולות אחרות אינן מתבצעות. הפונקציה מומשה בדומה למה שנלמד בהרצאות, והיא פועלת בזמן כולל של O(1). לאחר הכנסה של איבר חדש, האיבר הראשון בערימה יהיה האיבר החדש שהוכנס, וכן בכל הכנסה של איבר חדש לערימה מתחזקים את השדה min של הערימה, וכן מקדמים(ב-1) את מספר האיברים בעץ.
הנחת היסוד היא כי המפתח הניתן כפרמטר הינו ייחודי, כלומר לא קיים Node בערימה עם המפתח key.

- **void deleteMin()** – פונקצייה המוחקת את האיבר(ה-Node) בעל המפתח המינימלי מהערימה. כיוון שבכל עת, השדה min, המכיל מצביע אל ה- Node עם המפתח המינימלי, מתוחזק, אז הגישה לאיבר זה מתבצעת ב- O(1). ישנה קריאה לפונקציה deleteAndLink(HeapNode parent) בה מתבצע תהליך המחיקה של האיבר מהעץ בפועל. לאחר מחיקת האיבר, מתבצע תהליך consolidation בערימה ובו מבצעים פעולות linking של העצים בערימה כך שנגיע לערימה בינומית לאחר פעולה. בנוסף, בסיום המחיקה מעדכנים את מספר האיברים בערימה(שדה ה-size) להיות אחד פחות מהקודם, וכן מעדכנים את השדה min כך שיצביע אל האיבר עם המפתח המינימלי בעץ.
מימוש הפעולה הינו דומה לנלמד בהרצאות, ולכן זמן הריצה amortized הינו O(log n).

- **void deleteAndLink(HeapNode parent)** – זוהי פונקציית עזר בתהליך של מחיקת ה-Node בעל המפתח המינימלי מהעץ. פונקציה זו מבצעת בפועל את המחיקה של האיבר, ולאחר מכן מעבירה את ילדיו של האיבר הנמחק להיות שורשים בעץ, ובעצם הפונקציה מכינה את הערימה לתהליך ה-consolidation. הפונקציה מתחשבת בכל מקרי הקצה האפשריים ובפועלת בסיבוכיות זמן כוללת של לכל היותר O(log n).

- **HeapNode[] to_buckets()** – פונקצייה המממשת את האלגוריתם של הפיכת ערימת פיבונצ'י לערימה בינומית בעזרת שיטת הדליים. השלב הראשון הינו אתחול המערך באורך מתאים(לוגריתם של מספר האיברים, בבסיס של יחס זהב) כך שבעצם כל תא i במערך מייצג את הדלי עבור העצים מדרגה i. באלגוריתם נעבור על כל עץ בערימה הנתונה ונכניס לדלי המתאים בהתאם לדרגת העץ. כעת נחלק למקרים לפי מצב הדלי.
  - אם העץ הנוכחי הינו מדרגה k, ובתא ה-k(שקול לדלי המתאים לעצים מדרגות k) אין אף עץ, אז נשים את העץ הנוכחי בדלי.
  - במקרה בו הדלי כבר מלא, כלומר יש בו עץ מדרגה k שיושב בו, אז נבצע פעולת linking בין העצים כדי לקבל עץ מדרגה k+1. עבור העץ מדרגה k+1 שנוצר בפעולת ה-linking נרצה להכניס לתא הבא. נבצע תהליך דומה שבו ננסה להכניס את העץ הנוצר מדרגה k+1 אל התא ה-k+1 ואם לא נצליח אז נבצע פעולת linking וכך נמשיך עד שהעץ שקיבלנו הוכנס לדלי המתאים.
  - הפונקציה מחזירה מערך של HeapNode, כלומר מחזירה בעצם מערך המייצג את כל הדליים מהאלגוריתם. העצים הנמצאים במערך זה בעצם מייצגים את הערימה לאחר פעולות ה-linking הדרושות, ובעצם מדובר בערימה בינומית בסוף פעולה זו.
  - סיבוכיות זמן הריצה יכולה להיות ליניארית בגודל הקלט, במצב בו ישנם n עצים בערימה, וכל עץ מכיל איבר בודד. ולכן זמן הריצה ב- WC הינו O(n). כמו כן, ראינו בהרצאות כי זמן הריצה ה-amortized הינו O(log n) כיוון שפעולה זו הינה חלק מפעולת ה-delete-min והוכחנו זמן ריצה amortized עבורה.

- **HeapNode from_buckets(HeapNode[] buckets)** – פונקצייה המקבלת את מערך הדליים שנוצר בפונקציה to_buckets ובונה ממערך זה מחדש את הערימה. התא ה-i במערך מכיל את העץ מדרגה i אם קיים כזה לאחר תהליך ה-consolidation. הפונקציה עוברת על כל המערך, ומחברת את העצים אחד לשני על מנת ליצור ערימה מהם. לאחר תהליך ה-consolidation נקבל ערימה בינומית, כלומר לכל דרגה יש לכל היותר עץ אחד מדרגה זו. זמן הריצה הינו $O(\log_{\phi} n)$ כאורך מערך הדליים שיצרנו, שכן נצטרך לעבור על כל תא במערך ולאסוף את העצים מכל התאים על מנת לחבר אותם לכדי ערימה אחת.
הפונקציה זו מחזירה HeapNode שהוא יהיה האיבר ההתחלתי של הערימה – start Root חדש לערימה.

- **void consolidate()** – זו פונקציית ה- consolidation הראשית. הפונקציה מופעלת לאחר פעולות מחיקה מהעץ, כאשר דרוש תהליך של איזון הערימה. הפונקציה קוראת למתודות to_buckets ולאחר מכן למתודה from_buckets שכפי שהוסבר קודם לכן, בהן מתבצעת תהליך ה-consolidation בשיטת הדליים. בסיום המתודות הללו מתוחזקים כל השדות הדרושים – min,startRoot אשר אותם מתחזקים לאחר השינויים בעץ. זמן הריצה הוא כזמן הריצה של המתודות הללו – amortized O(log n).

- **static HeapNode link(HeapNode node1, HeapNode node2)** – פונקצייה סטטית שמקבלת שני עצים מדרגה זהה – ועושה ביניהם פעולת linking. ההנחה לגבי הפונקציה היא כי המפתחות בעץ node1 קטנים מהמפתחות בעץ node2, כך שמעבירים לפונקציה זו את העצים בסדר הנכון לפי הנחה זו. הפונקציה מחזירה את השורש של העץ הנוצר. הפונקציה מטפלת בכל מקרה הקצה האפשריים, וכן מעדכנת את השדה totalLinks שסופר את מספר פעולות ה- linking שבוצעו, וכן מעדכת את הדרגה של העץ החדש. סה"כ זמן הריצה הינו קבוע - O(1). מתבצעות בפונקציה זו פעולות קבועות בלבד בעיקר באשר לשינויי מצביעים ותיחזוק שדות.

- **HeapNode findMin()** – פונקציה המחזירה איבר(HeapNode) שמצביע אל האיבר עם המפתח המינימלי בעץ. הפעולה נעשית ב-O(1) זמן קבוע שכן מתוחזק בכל שלב שדה זה, על ידי ביצוע פעולות בכל שאר הפעולות על הערימה.

- **void meld(FibonacciHeap heap2)** – פונקציית ה-meld, איחוד של שתי ערימות פיבונצ'י לכדי ערימה אחת. כיוון שאין הגבלות על העצים בערימת פיבונצ'י וכיוון שמותר שיהיו כמה עצים מאותה הדרגה באותה הערימה, ביצוע ה-meld מתבצע כפי שלמדנו על ערימה בינומית עצלה. באופן פשוט – מחברים את השורשים של הערימה הנתונה heap2 לסוף הערימה הנוכחית(this) על ידי שינויי מצביעים, וזה מסיים את פעולת ה-meld על ערימות פיבונצ'י. אין צורך בביצוע פעולות נוספות של consolidation למשל שכן עץ פיבונצ'י אכן מאפשר כפילויות של עצים מדרגות שוות בערימה אחת. מספר השורשים החדש הינו סכום של מספר השורשים הקודם בתוספת מספר השורשים של הערימה heap2. המינימום של הערימה החדשה שנוצרת – הינו המינימום בין שני המינימומים של ערימות הקלט. מספר האיברים הכולל בערימה הינו סכום האיברים בשתי הערימות.
סיבוכיות זמן הריצה של פעולה זו הוא O(1) – זמן קבוע לכל פעולת meld.

- **int size()** – פונקציה זו מחזירה את ה-size של הערימה. ה-size של הערימה מוגדר כמספר האיברים ברשימה. הפונקציה פועלת ב-O(1) כיוון שבכל שלב של השדה size מתוחזק בכל פעולה על גבי הערימה.

- **int[] countersRep()** – פונקציה המחזירה מערך המייצג את דרגות העצים שיש בערימה. התא ה-i במערך מייצג את מספר העצים מדרגה i שנמצאים בערימה. במידה והערימה ריקה, אז נחזיר מערך מגודל 0. במתודה אין הגבלה על סיבוכיות זמן הריצה לפי הנחיות התרגיל. בתחילה, הפונקציה עוברת על כל העצים בערימה ומחפשת מהי הדרגה הגבוהה ביותר בעץ. נסמן ב-max. לאחר מכן, נאתחל מערך מגודל max+1. גודל זה יספיק על מנת לתאר את כל דרגות העץ הקיימות בערימה שכן חיפשנו ראשית את הדרגה המקסימלית ועל כן זה מתאפשר. לאחר מכן, נעבור על כל עץ בערימה ונבדוק את דרגתו ולפיה נעדכן את מונה התא המתאים ב-1. המערך המתקבל הינו מערך כך שהמספר שנמצא בתא ה-i מייצג את מספר העצים בערימה שהינם מדרגה i. נחזיר בסוף הפעולה את המערך המתקבל.
סיבוכיות זמן הריצה במקרה הגרוע יכולה להיות O(n), במצב בו ישנם n עצים בערימה, וכן n איברים בה, כך שכל עץ בערימה מכיל איבר בודד. מעבר על כל העצים על מנת למצוא את הדרגה המקסימלית יקח במקרה זה - O(n), אתחול המערך גם כן - O(n), ומעבר על העצים פעם נוספת על מנת לבנות את המערך – O(n), וסה"כ קיבלנו כי במקרה הגרוע(WC) ייתכן כי זמן הריצה יהיה ליניארי במספר האיברים בערימה – O(n).

- **void delete(HeapNode x) –** פונקציית המחיקה של ה-Node x המתקבל כפרמטר מהערימה. מחיקה מתבצעת על ידי פעולת decrease-key של האיבר x כך שיהיה האיבר בעל המפתח המינימלי בעץ. לאחר מכן, בעזרת הפונקציה delete-min מוחקים את האיבר עם המפתח המינימלי בעץ, שבמקרה זה הינו x. **סיבוכיות זמן הריצה הינו amortized O(log n) כפי שנלמד בכיתה.**

- **void decreaseKey(HeapNode x, int delta) –** פונקצייה המקבלת איבר בערימה – HeapNode x, וערך מספרי delta ומורידה את הערך של המפתח של x ב-delta. אם חוקיות הערימה נשמר מבחינת האינווריאנטה שלה – אז שום דבר אינו נעשה. האינווריאנטה של הערימה היא שכל מפתח גדול מהמפתח שנמצא מעליו, וקטן מהמפתח שנמצא מתחתיו. במקרה בו x הינו שורש של עץ, האינווריאנטה נשמרת בכל מקרה. במצב בו פעולת ה-decreaseKey גרמה לכך שחוקיות הערימה לא נשמרה, אז נבצע סדרה של cascading-cuts לפי הנלמד בהרצאות, שלאחריהן הערימה תהיה ערימת פיבונצ'י חוקית. **סיבוכיות זמן הריצה הינה amortized O(1).**

- **void cut(HeapNode x, HeapNode y) –** פונקציית החותכת את ה-Node x מ-y, כאשר ההנחה היא כי y הינו ה-parent של האיבר x. הפונקציה בעצם לוקחת את תת-העץ של האיבר x ומוסיפה אותו כעץ בערימה, כלומר כעת x יהפוך לשורש של עץ בערימה עצמה. לאחר הפעולה, האיבר הראשון בערימה יהיה העץ שנחתך, כלומר הערימה תצביע לעץ שבו x השורש. כמו כן, לאחר פעולה זו x יהפוך להיות unmarked שכן הוא שורש. ולכן אם הוא היה מסומן קודם לכן(לפי שנחתך מהעץ המקורי), אז נוריד הערך של השדה numOfMarked ב-1, שכן כעת בערימה יש איבר אחד פחות שמסומן. כמו כן, כיוון שהוספנו עץ חדש לערימה, נעדכן ב-1 את השדה numOfRoots שמונה את מספר העצים בערימה.
בכל שימוש בפונקציה זו – מגדילים את ערך השדה הסטטי totalCuts ב-1, שכן שדה סטטי זה סופר את כמות הפעמים הכוללת שבהן ביצענו cut.
סיבוכיות זמן הריצה הינה O(1) – זמן קבוע לכל פעולת cut, שכן מבצעים מספר סופי של פעולות שכל אחת מהן הינה פשוטה ולוקחת זמן קבוע.

- **void cascading_cuts(HeapNode x, HeapNode y) –** פונקציה זו מתבצעת לאחר פעולת decrease-key שגרמה לכך שהערימה אינה חוקית – כלומר שהאינווריאנטה של הערימה אינה מתקיימת. פעולה זו תבצע ראשית cut של האיבר x מה- parent שלו – y, כיוון שהם אלו שגרמו לבעיה של שבירת האינווריאנטה של הערימה. לאחר מכן, יתבצעו סדרה של פעולות cut כל עוד התנאים הבאים מתקיימים:
  - y אינו השורש של העץ.
  - ה- parent של y הינו מסומן – כלומר marked.

    - ❖ במקרה בו תנאים אלו מתקיימים נבצע באופן רקורסיבי קריאה לפונקציה cut(y,y.getParent) – כך שגם y יחתך מה- parent שלו.

הפונקציה תפסיק את סדרת ה-cascading cuts כאשר נגיע לצומת שאינו מסומן, ובמצב זה נסמן אותו, או כאשר נגיע לשורש של העץ, ובמצב זה לא נמשיך את האלגוריתם כי בהכרח השורש אינו מסומן.

בסה"כ הפונקציה תבצע סדרת פעולות cut על גבי המסלול מהאיבר ה- x עד לשורש העץ, וסדרת פעולות זו תיגמר כאשר נגיע לאיבר שאינו מסומן(unmarked) שאותו נסמן לשימוש עתידי(אלא אם זהו השורש).

סיבוכיות זמן הריצה היא amortized O(1) כפי שראינו בהרצאות.

- **int potential() –** פונקציה המחזירה את הפוטנציאל של הערימה. הפוטנציאל מחושב באופן הבא.
$$Potential = \#trees + 2\#marks$$
סיבוכיות זמן הריצה היא O(1) שכן הערימה מתחזקת את השדות הנדרשים על מנת לחשב פוטנציאל.

- **static int totalLinks() –** פונקציה סטטית המחזירה ערך מספרי המייצג את המספר הכולל של פעולות link שהתבצעו עד כה בכל ריצת התוכנית.
סיבוכיות זמן הריצה היא O(1), מכיוון שהערימה מתחזקת מונה סטטי לפרמטר זה.

- **static int totalCuts() –** פונקציה סטטית המחזירה ערך מספרי המייצג את המספר הכולל של פעולות cut שהתבצעו עד כה בכל ריצת התוכנית.
סיבוכיות זמן הריצה היא O(1), מכיוון שהערימה מתחזקת מונה סטטי לפרמטר זה.

- **Void inserttominheap(HeapNode x)-** פונקציה זו היא פונקציית עזר לפונקציית k-min. תפקידה להכניס את הילדים של צומת לתוך ערימת העזר בה אנו משתמשים ב-k-min שיפורט בהמשך. X הינו הבן של הצומת אשר את ילדיו נרצה להכניס לערימה.
סיבוכיות זמן הריצה של הפונקציה היא $O(x.parent.rank)$ כלומר כמספר הילדים שיש לצומת אשר את ילדיה אנו מכניסים לערימת העזר שב-w.c דרגתה כדרגת העץ הבינומי H , כלומר $O(Deg(H))$.

- **heapNode insert(int key, HeapNode x)-** פונקציה זו היא פונקציית עזר לפונקציית k-min. פועלת בדיוק באותו אופן כמו insert שתוארה מעלה רק שבפונקציה זו בפרמטר השני שהוא heapNode ישמש כמצביע לעץ הבינומי המקורי לאיבר שהוכנס זה עתה לערימת המינימום כפי שמתבצע במתודת k-min (יוסבר בהמשך) , x יעדכן את השדה kminpointer שנועד בדיוק לפעולה זו. סיבוכיות פונקציה זו זהה לפונקציית insert המקורית כיוון שעדכון השדה kminpointer עולה O(1).

- **static int[] kmin(FibonacciHeap h,int k)-** פונקציה זו היא פונקציה סטטית המחזירה מערך עם k המפתחות הקטנים ביותר בעץ בינומי הניתן כקלט. נאתחל מערך בגודל k וערימת פיבונצ'י שתשמש כערמת עזר. נכניס לערימה את האיבר המינימאלי בעץ הבינומי, לאחר מכן נבצע k פעמים:
נכניס את האיבר המינימאלי בערימת העזר למערך (באיטרציה ה-i נכניס את האיבר למערך במקום ה-i כאשר $0 \leq i < k$). לאחר מכן נבצע delete-min על ערימת העזר כעת אם לאיבר שמחקנו מערימת העזר יש ילדים בעץ הבינומי המקורי(שומרים פוינטר לעץ המקורי) נכניס אותם לערימת המינימום .
ניתוח סיבוכיות:
הלולאה חוזרת k פעמים כאשר בכל איטרציה יכולים להכניס לערימת העזר לכל היותר $Deg(H)$ איברים שהם הילדים של האיבר המינימאלי (בערימת העזר) בעץ הבינומי המקורי. על כן בערימת העזר יהיו לכל היותר $O\left(k \cdot Deg(H)\right)$. על פעולת delete-min על ערימה בגודל כזה תעלה
$O(\log\left(k \cdot Deg(H)\right) = O(\log(k) + \log\left(Deg(H)\right))$ (לאחר שבאיטרציה הקודמת בוצע גם כן delete-min ולכן מספר השורשים של עצים בערימה הינו $O(\log\left(k \cdot Deg(H)\right)$).עבור לכל היותר $Deg(H)$ פעולות insert באיטרציה שכאמור עולה $O(1)$ בערמת פיבונצ'י מתקיים כי הוספת האיברים לערמת העזר תעלה בכל איטרציה לכל היותר $O(Deg(H))$.
על כן סיבוכיות זמן הריצה ב-*w.c* הינה:

$$k \cdot \left(O\left(Deg(H)\right) + O\left(\log(k) + \log\left(Deg(H)\right)\right)\right) = \boldsymbol{O}\left(\boldsymbol{k}\left(\boldsymbol{logk} + \boldsymbol{Deg}(\boldsymbol{H})\right)\right)$$

מדידות

חלק א - Sequence 1

m	Run-Time (in miliseconds)	totalLinks	totalCuts	Potential
1024	1.642	1023	18	19
2048	3.601	2047	20	21
4096	5.891	4095	22	23

סעיף 1.

זמן הריצה האסימפטוטי הינו O(m). ראשית מבצעים סדרה של m+1 הכנסות של איברים, הכנסת איבר מתבצעת ב-O(1) ולכן סיבוכיות זמן הריצה היא O(m+1) = O(m).
לאחר מכן מבצעים delete-min יחיד שעלותו תהיה O(m) כיוון שלפני המחיקה היו m+1 עצים מדרגה 0, ופעולה זו נעבור על כל העצים ונבצע פעולות linking עד אשר יתחברו לעץ בינומי אחד גדול(בגודל m, זה אפשרי כיוון ש-m הינו חזקה של 2). כעת נבצע סדרה של O(log m) פעולות decrease-key. בסדרה זו נבצע את ה-decrease-key באופן הבא: באיטרציה הראשונה נפנה לבן עם הדרגה המינימלית שלו, שזהו בעצם שורש של עץ מדרגה 0, קיים כזה כיוון שעץ בינומי מדרגה k בנוי מ-k ילדים שהם שורשים של עצים מדרגות 0,...,k-1, ונבצע עליו decrease-key שיגרום לכך שיחתך מהעץ, ויסמן את שורש העץ כולו. באיטרציה הבא נפנה לבן עם הדרגה המקסימלית של השורש, וצומת זה מהווה שורש של עץ בינומי מדרגה הקטנה ב-1 משורש העץ, log m – 1, ובאותו אופן נלך לבן עם הדרגה המינימלית שלו ונבצע עליו decrease-key, ונסמן את שורש של עץ זה. נמשיך באותו אופן ונטייל על המסלול שהוא כגובה העץ, כאשר בכל פעם נחתוך את הילד עם הדרגה המינימלית ונסמן את שורש תת-העץ. בפעולה decrease-key(m-1), נחתוך את האיבר עם המפתח m-1 מאבא שלו, כאשר אבא שלו הינו כבר מסומן שכן איבד כבר בן אחר, ולפיכך תתבצע סדרת פעולות cascading-cuts מאיבר זה עד לשורש העץ כולו, שכן כל איבר במסלול הינו מסומן לפי מה שהוסבר. במסלול כלפי מטה ביצענו O(log m) פעולות cut, שעלותן O(1) כל אחת, וגם במסלול כלפי מעלה ביצענו סדרה של O(log m) פעולות cut, כגובה העץ. סה"כ נבצע O(log m) + O(log m) = O(log m) פעולות cut ברצף פעולות זה, ולכן סה"כ סיבוכיות זמן הריצה הכוללת הינה
O(m) + O(log m) = O(m)

סעיף 2.

פעולות Link – התבצעו O(m) פעולות linkings בסה"כ. כל הפעולות התבצעו במהלך פעולת ה-delete-min הראשונית בה m עצים מדרגות 0 התחברו לכדי עץ בינומי מדרגה log m. הסבר מפורט על תהליך ה-linking בסעיף 1.

פעולות cut – התבצעו O(log m) פעולות cut בסה"כ. בתחילת התבצעו logm -1 פעולות cut בתהליך הירידה מטה בעץ, ולאחר מכן התבצעו עוד logm -1 פעולות cut, כך שבסה"כ 2logm – 2, ולכן O(log m) פעולות cut. הסבר מפורט בסעיף 1.

סעיף 3.

כפי שהוסבר לגבי רצף פעולות זה, בפעולה decrease-key(m-1) התבצעו O(log m) פעולות cut. זו הייתה פעולת ה-decrease-key היקרה ביותר, כיוון שרצף הפעולות הקודם לה גרם לכך שכל הצמתים במסלול ממנה כלפי השורשים יהיו מסומנים(marked), ולפיכך בעת פעולת cut נוספת במהלך ה-decrease-key האחרון, תתבצע סדרת פעולות cascading-cut מצומת זה עד לשורש העץ. לפיכך, בפעולה זו יתבצעו O(log m) פעולות cut, כגובה העץ, וכן כל פעולה היא בזמן קבוע O(1) ולכן בסה"כ פעולה זו של decrease-key(m-1) הינה פעולת
ה-decrease-key היקרה ביותר, והיא לקחה O(log m). שאר פעולות ה-decrease-key התבצעו ב-O(1) כיוון שלא גררו לאחריהן סדרת פעולות cascading-cuts. התוצאות תואמות את הטבלה שקיבלנו ולפיהן מספר פעולות ה-cut הכולל הוא 2logm – 2 ולפיכך O(log m) במונחי זמן ריצה אסימפטוטי.

חלק ב - Sequence 2

M	Run-Time (in miliseconds)	totalLinks	totalCuts	Potential
1000	1.932	1891	0	6
2000	2.886	3889	0	6
3000	4.411	5772	0	7

סעיף 1.

זמן הריצה האסימפטוטי הכולל של סדרת פעולות זו הינו $O(mlogm)$. ראשית סדרת m הכנסות תעלה O(m), שכן הכנסה לערימת פיבונצ'י מתבצע בדרך עצלה של O(1), ולכן זמן הריצה הינו כאורך הקלט – O(m). לאחר סדרת הכנסות זו נקבל ערימה של m עצים שכל אחד מדרגה 0. במחיקה הראשונה בסדרת המחיקות תעלה O(m), ולאחריה נקבל ערימה בינומית שכל פעולת delete-min על גבי תיצור ערימה בינומית חדשה עם איבר אחד פחות. פעולת delete-min בערימה בינומית בעלת n איברים תקח O(log n), על כן, נקבל את הסיבוכיות הבאה:
זמן הריצה האסימפטוטי הכולל של סדרת פעולות זו הינו $O(mlogm)$. ראשית סדרת m הכנסות תעלה O(m), שכן הכנסה לערימת פיבונצ'י מתבצע בדרך עצלה של O(1), ולכן זמן הריצה הינו כאורך הקלט – O(m). לאחר סדרת הכנסות זו נקבל ערימה של m עצים שכל אחד מדרגה 0. במחיקה הראשונה בסדרת המחיקות תעלה O(m), ולאחריה נקבל ערימה בינומית שכל פעולת delete-min על גבי תיצור ערימה בינומית חדשה עם איבר אחד פחות. פעולת delete-min בערימה בינומית בעלת n איברים תקח O(log n), על כן, נקבל את הסיבוכיות הבאה:

זמן הריצה האסימפטוטי הכולל של סדרת פעולות זו הינו $O(mlogm)$. ראשית סדרת m הכנסות תעלה O(m), שכן הכנסה לערימת פיבונצ'י מתבצע בדרך עצלה של O(1), ולכן זמן הריצה הינו כאורך הקלט – O(m). לאחר סדרת הכנסות זו נקבל ערימה של m עצים שכל אחד מדרגה 0. במחיקה הראשונה בסדרת המחיקות תעלה O(m), ולאחריה נקבל ערימה בינומית שכל פעולת delete-min על גבי תיצור ערימה בינומית חדשה עם איבר אחד פחות. פעולת delete-min בערימה בינומית בעלת n איברים תקח O(log n), על כן, נקבל את הסיבוכיות הבאה:

$$O(m) + O(m) + \sum_{i=2}^{m/2} O(logm) = O(m) + \left(\frac{m}{2} - 1\right) \cdot O(logm) = O(m) + O(mlogm) = \boldsymbol{O(mlogm)}$$

סעיף 2.

פעולות cuts – מספר פעולות ה-cuts הינו 0, שכן לא התבצעו בכלל decrease-keys ולפיכך אין גם פעולות cut.
פעולות links – מספר פעולות ה-linking הינו O(mlogm) אסימפטוטית. בתחילה בפעולת ה-delete-min הראשונה נבצע O(m) פעולות linking על מנת להביא את העץ לערימה בינומית. בשאר $\frac{m}{2}-1$ פעולות ה-delete-min נבצע O(log m) פעולות linking כיוון שבכל שלב זו ערימה בינומית. ולכן בסה"כ נבצע O(m) + O(mlogm) = O(mlogm) פעולות linking.

סעיף 3.

לאחר כל סדרת פעולות זו, נישאר עם $\frac{m}{2}$ איברים בערימה בינומית תקינה. ולכן מספר העצים בערימה זו יהיה O($log(\frac{m}{2})$) =O(log m). כמו כן, מספר האיברים המסומנים בסוף סדרת הפעולות יהיה 0, מכיוון שלא ביצענו אף לא פעולות decrease-key אחת ולפיכך לא התבצעו פעולות cut במהלך סדרת הפעולות, ולכן סימוני ה-marked של האיברים אינם השתנו, ולכן אין איברים מסומנים בערימה בסיום התהליך. לכן מספר האיברים המסומנים הינו 0.
פונקציית הפוטנציאל הוגדרה באופן הבא

$$\text{Potential} = \#trees + 2 \cdot \#marks$$

ולפיכך ערך פונקציית הפונטציאל בסוף סדרת פעולת זו הינה

$$Potential = O(log(\frac{m}{2})) + 2 \cdot 0 = O(log\, m)$$

התוצאות שהתקבלו אכן תואמת לתוצאות שהתקבלו במדידות.