שם: איתמר בירן

ת״ז: 318534203

משתמש במודל: itamarbiran

שם: יונתן מקובסקי

ת״ז: 208935916

משתמש במודל: makovsky1

<u>**חלק א – מעשי:**</u>

<u>מחלקת **FibonacciHeap**</u>

<u>**שדות:**</u>

min (HeapNode) – מצביע לnode בעל המפתח המינימלי בערימה.

first (HeapNode) – מצביע לNode השמאלי ביותר מבין השורשים בערימה.

last (HeapNode) – מצביע לNode הימני ביותר מבין השורשים בערימה.

size (int) – מספר הצמתים בערימה.

totalCuts (static int) – מספר פעולות הcuts שביצענו בעת הפעלת הפונקציות השונות.

totalLink (static int) – מספר פעולות הlink שביצענו בעת הפעלת הפונקציות השונות.

Trees (int) – מספר העצים בערימה.

Marked (int) – כמה צמתים מסומנים (mark) יש לנו בערימה.

<u>**בנאים:**</u>

public fibonacciHeap() – בנאי של ערימה ריקה. מעדכן את השדות בהתאם.

Public fibonacciHeap (HeapNode node) – בנאי המקבל node. מתחזקים את השדות השונים של הערימה, בין היתר ע״י הnode וערכיו.

<u>**פעולות:**</u>

Public boolean isEmpty()

- מה עושה – בודקת אם המופע הנוכחי של המחלקה הוא ערימה ריקה
- כיצד פועלת – בודקת את ערך השדה size. אם הוא 0, כלומר שהמופע הנוכחי הינו ערימה ריקה, ועל כן נחזיר true. אחרת, נחזיר false.
- סיבוכיות זמן הריצה – O(1).
- קריאה לפונקציית עזר – אין.

public HeapNode insert(int key)

- מה עושה – מקבלת מפתח, ומכניסה לערימה Node חדש עם ערך המפתח שקיבלה.
- כיצד פועלת – יוצרת node חדש, ע״י קריאה לבנאי המקבל את key, ולאחר מכן מחזירה את הפעלת הפונקציה insertNode על הnode שיצרה.
- סיבוכיות זמן הריצה – O(1)
- קריאה לפונקציית עזר – insertNode.

Public HeapNode insertNode(HeapNode node)

- מה עושה – דואגת להכנסת node חדש לערימה.
- כיצד פועלת – בודקת אם הערימה ריקה – א. אם ריקה, תעדכן את השדה min להצביע על הNode החדש, וכן תעדכן את שאר השדות בהתאם. ב. אם לא ריקה – תשווה את המפתח של node לערך המפתח המינימלי בערימה, תעדכן את החmin במידת הצורך, וכן תעדכן את שאר השדות בהתאם.
  **הפעולה מכניסה את Node להיות השורש השמאלי ביותר בערימה.
- סיבוכיות זמן הריצה – O(1)
- קריאה לפונקציית עזר -אין .

Public void deleteMin()

- מה עושה – מוחקת את הצומת המינימלית מהרשימה והופכת את הערימה לערימה בינומית
- כיצד פועלת – מוחקת את הצומת על ידי שינוי המצביעים ואז שולחת לפונקציות עזר אשר בונות מחדש את העץ
- סיבוכיות זמן הריצה – amortized(O(logn))
- קריאה לפונקציית עזר - succesiveLinking

Private void successiveLinking()

- מה עושה – הופכת את הערימה לערימה בינומית אחרי מחיקה
- כיצד פועלת – מאתחלת מערך אשר גודלו נקבע לפי הערכה של הדרגה המקסימלית וקוראת לפונקציות toBuckets,fromBuckets
- סיבוכיות זמן הריצה – amortized(O(logn))
- קריאה לפונקציית עזר -toBuckets,fromBuckets

Private HeapNode link(HeapNode node1, HeapNode node2)

- מה עושה – מחברת שתי עצים אחד לשני
- כיצד פועלת – בודקת מי העץ בעל השורש המינימלי ומחברת ביניהם כך שהעץ בעל השורש המינימלי הוא האבא של העץ השני. כמו שראינו בכיתה
- סיבוכיות זמן הריצה – O(1)
- קריאה לפונקציית עזר -אין

Private void toBuckets(HeapNode [] bucket)

- מה עושה – עוברת על כל העצים בערימה והופכת אותם לכך שיהיה עץ אחד מכל דרגה לכל היותר ומכניסה אותם למערך
- כיצד פועלת –עוברת על דרגות העצים ומחברת עצים בעלי אותה דרגה על ידי שליחה לפונקציה link
- סיבוכיות זמן הריצה – O(T) כאשר T מספר העצים
- קריאה לפונקציית עזר –

- **Private void fromBuckets(HeapNode [] bucket)**
- מה עושה – מקבלת מערך בו יש עץ אחד מכל דרגה לכל היותר והופכת אותם לערימת פיבונאצ׳י

- כיצד פועלת – מקבלת מערך ואז מחברת בין העצים במערך כך שהעץ השמאלי ביותר הוא בדרגה הכי נמוכה.
- סיבוכיות זמן הריצה – O(logn)
- קריאה לפונקציית עזר -אין

Public HeapNode findMin()

- מה עושה – מחזירה את הNode בעל ערך המפתח המינימלי בערימה.
- כיצד פועלת – אם הערימה ריקה – מחזירה Null, אחרת, מחזירה את הNode עליו מצביע השדה min.
- סיבוכיות זמן הריצה – O(1).
- קריאה לפונקציית עזר – isEmpty.

Public void meld(FibonacciHeap heap2)

- מה עושה – מחברת את הערימה החדשה מימין לערימה המקורית, ולמעשה הופכת את שתי הערימות לערימה אחת.
- כיצד פועלת – אם הערימה החדשה היא ערימה ריקה, לא נבצע כלום. אחרת – נעדכן את הpointers בהתאם, כך שהערימה החדשה תתחבר מימין לערימה המקורית. כמו כן, נעדכן את השדות הרלוונטיים בהתאם (size,trees,marked,min).
- סיבוכיות זמן הריצה – O(1).
- קריאה לפונקציית עזר – isEmpty.

Public int size()

- מה עושה – מחזירה את מספר הצמתים בערימה.
- כיצד פועלת – מחזירה את ערך השדה size.
- סיבוכיות זמן הריצה – O(1).
- קריאה לפונקציית עזר – אין.

Public int[] countersRep()

- מה עושה – מחזירה רשימה, כך שבמקום הi, הערך יהיה מספר הצמתים בערימה כך שהשורש שלהם הוא מדרגה i.
- כיצד פועלת – אם הערימה ריקה – נחזיר מערך ריק. אחרת – נייצר רשימה מגודל דרגת השורש הגדולה ביותר + 1. נעבור על כל השורשים בערימה, ועבור שורש מדרגה i, נוסיף לערימה במקום הi + 1. כך נקבל בסוף את הרשימה הדרושה.
- סיבוכיות זמן הריצה – O(T). כאשר T מספר העצים בערימה.
- קריאה לפונקציית עזר – isEmpty.

Public void delete(HeapNode x)

- מה עושה – הפונקציה מקבל Node x, ומוחקת אותו מהעץ ע״י קריאה לפונקציה deleteMin.

- כיצד פועלת – הופכת את ערך המפתח של x להיות המפתח המינימלי בעץ, ואז מפעילה על x את הפעולה deleteMin.
- סיבוכיות זמן הריצה – amortized(O(logn))
- קריאה לפונקציית עזר – decreaseKey, deleteMin.

Public void decreaseKey(HeapNode x, int delta)

- מה עושה – מעדכנת את ערך המפתח כך שיהיה ערך המפתח פחות delta
- כיצד פועלת –מעדכנת את ערך המפתח ואז בודקת אם עדכון המפתח יגרום לכך שהערימה לא תקנית ואם כן אז שולחת לפונקצייה cascadingCuts
- סיבוכיות זמן הריצה – amortized(O(1))
- קריאה לפונקציית עזר -cascadingCuts

Public void cascadingCuts(HeapNode x)

- מה עושה – כמו שראינו בהרצאה מקבלת את המפתח שבוצע לו הפחתה של המפתח וחותכת אותו מהאבא ודואגת לסמן בהתאם את האבא או לחתוך אותו.
- כיצד פועלת – מתחילה במפתח שמקבלים וחותכת אם האבא מסומן גם אז תמשיך הלאה ותחתוך גם את האבא. הפונקציה תעצור או שנגיע לשורש או שהאבא לא מסומן ואז במקרה זה תסמן אותו ותסיים את הריצה. הפונקציה דואגת לסמן את הצמתים בהתאם למה שלמדנו בכיתה.
- סיבוכיות זמן הריצה – amortized(O(1))
- קריאה לפונקציית עזר -אין

Public int potential()

- מה עושה – מחזירה את פוטנציאל הערימה, כלומר את מספר העצים בערימה * פעמיים מספר הצמתים המסומנים (marked).
- כיצד פועלת – מחזירה את מכפלת ערכי השדות trees ו – 2*marked.
- סיבוכיות זמן הריצה – O(1).
- קריאה לפונקציית עזר - אין

Public int numberOfTrees()

- מה עושה – מחזירה את מספר הצמתים בערימה.
- כיצד פועלת – מחזירה את ערך השדה trees.
- סיבוכיות זמן הריצה – O(1).
- קריאה לפונקציית עזר – אין.

Public static int totalLinks()

- מה עושה – מחזירה כמה פעמים התבצעה הפונקציה link.
- כיצד פועלת – מחזירה את ערך השדה totalLinks.
- סיבוכיות זמן הריצה – O(1).
- קריאה לפונקציית עזר – אין.

Public static int totalCuts()

- מה עושה – מחזירה כמה פעמים התבצעה הפונקציה cut..
- כיצד פועלת – מחזירה את ערך השדה totalCuts.
- סיבוכיות זמן הריצה – O(1).
- קריאה לפונקציית עזר – אין.

Public static int[] kMin(fibonacciHeap H, int k)

- מה עושה – מחזירה מערך בו k האיברים המינימליים בערימה
- כיצד פועלת -מאתחלת מערך בגודל K וערימה חדשה בערמה החדשה תחילה מכניסה את המינימום לערימה החדשה ומכניסה את מפתח המינימום למערך ומוחקת אותו מהערימה החדשה ואז מכניסה את בניו ובודקת שנית מי המינימום מכניסה את המפתח למערך ואז מוחקת אותו ומכניסה את בניו וכך התהליך חוזר עד שהמערך מלא. לבסוף מחזירה את המערך.
- סיבוכיות זמן הריצה – O(degH*k)
- קריאה לפונקציית עזר –אין

מחלקת **HeapNode**

שדות:

Key (int) – מפתח הNode.

Rank (int) – דרגת הnode.

Mark (Boolean) – האם הnode הוא marked או לא.

Child (HeapNode) – מצביע לילד של הnode.

Next (HeapNode) – מצביע לnext.

Prev (HeapNode) – מצביע לprev.

Parent (HeapNode) – מצביע להורה.

Info (HeapNode) – מצביע אשר יבוא לידי ביטוי בפונקציה kMin.

בנאים:

Public HeapNode() – בנאי של צומת ריק, מעדכן את המפתח להיות ?, את הדרגה 1-, את mark להיות false, וכל שאר השדות האחרים להיות null.

Public HeapNode(int key) – בנאי של צומת המקבל מפתח. מעדכן את המפתח להיות key, את הדרגה 0, את mark להיות false, וכל שאר השדות האחרים להיות null.

פעולות:

פעולות בסיסיות להשגת השדות השונים של המחלקה, מימוש פשוט ע״י קריאה לשדה הרלוונטי של המופע this.

חלק ב׳ – ניתוח תיאורטי:

שאלה 1:

סעיף א׳:

טענה – זמן הריצה במונחי ״Big O״ של סדרת הפעולות כפונקציה של m הינו – O(m).

נביט בשלבים השונים:

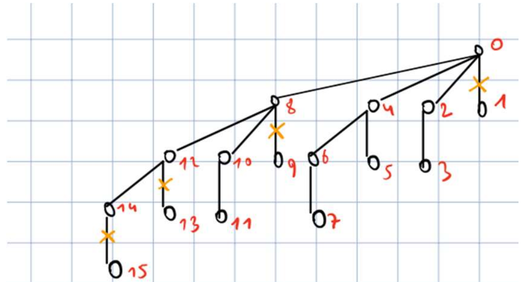
א. insert(k): for k=m-1, m-2, ..., 0, -1 : ראינו כי פעולת insert מסיבוכיות של O(1). אנחנו מבצעים m+1 הכנסות, לכן בסה״כ O(m+1) = O(m).

ב. deleteMin(): ניתוק המינימלי – מסיבוכיות O(1). לאחר מכן נבצע את פעולת successiveLinking על m עצים, כל עץ בגודל 1. מהגדרת successiveLinking, נקבל לאחר הריצה עץ מלא בגודל m – כלומר בעל (m-1) קשתות. במסגרת הפעולה, נבצע פעולות קבועות מסיבוכיות O(1). על כן נבצע בסה״כ m-1 חיבורים (כלומר יתבצעו m-1 קריאות לפעולה **link**), כלומר O(m-1) = O(m).

ג. for i=log2(m), ..., 2, 1: decreaseKey($m - 2^{i} + 1$, $m + 1$) : נסביר מדוע כל פעולת cut שנבצע במסגרת decreaseKey ***אינה תהיה פעולה סדרתית***, **כלומר האב של הצומת עליו נבצע את הdecreaseKey אינו ינותק**.
ראשית, נציין כי בכל פעולת decreaseKey, אכן תתבצע פעולת cut, שכן יופר המבנה התקין של העץ (כלומר ההורה של הצומת עליו נבצע את הdecreaseKey יכיל מפתח גדול יותר מבנו). שנית, עבור הערכים עליהם נבצע את הdecreaseKey, ניתן לראות כי הם תמיד יהיו הבנים של צומת אשר נמצאת על הענף השמאלי ביותר בעץ (כלומר הענף השמאלי לשורש). על כן האב של כל צומת כזו תמיד יהיה לא מסומן לפני החיתוך, ולאחר החיתוך יסומן, אולם לעולם לא תבוצע עליו פעולת החיתוך. על כן נקבל בסה״כ log(m) פעולות חיתוך (אשר יכללו גם חיבור של הצומת הנתון כעץ חדש בערימה), כאשר כל פעולה כזו היא מסיבוכיות של O(1), ובסה״כ O(log(m)) פעולות.

בסה״כ קיבלנו – סיבוכיות זמן הריצה הינה **O(m)** = O(m) + O(m) + O(log(m)) כנדרש.

**** מצרף איור של עץ בגודל 2^4 להמחשת ההסבר.**



נבט בערכי: for: $i = \log_2(m), \ldots, 2, 1$

$i = \log(m)$ decreaseKey($\underset{1}{1}$, $m+1$)

$i = \log(m)-1$ decreaseKey($\underset{9}{\frac{m}{2}+1}$, $m+1$)

$i = \log(m)-2$ decreaseKey($\underset{13}{\frac{3}{4}m+1}$, $m+1$)

$i = \log(m)-3$ decreaseKey($\underset{15}{\frac{7}{8}m+1}$, $m+1$)

סיימנו, כנדרש. □

סעיף ב׳:

m	Run-Time (ms)	totalLinks	totalCuts	Potential
2^10	3	1023	10	29
2^15	15	32767	15	44
2^20	63	1048575	20	59
2^25	8938	33554431	25	74

סעיף ג׳:

מספר פעולות הlink: m-1. הסבר: ראה סעיף א׳.ב.

מספר פעולות הcut: log(m). הסבר: ראה סעיף א׳.ג.

הפוטנציאל: נזכר כי הנוסחה לחישוב הפוטנציאל הינה $potential = numOfTrees + 2 \cdot Marked$. לאחר סדרת הפעולות, נקבל:

NumOfTrees = log(m) + 1 – הסברנו בסעיף א׳, כי לעץ המקורי ייתווספו log(m) עצים בגודל 1, כמספר פעולות החיתוך (וכן נובע מהגדרת פעולת insert).

Marked = log(m) -1. כפי שראינו בסעיף א׳, אנו נסמן את כל הצמתים אשר נמצאים על הענף השמאלי ביותר בעץ שיתקבל, כלומר בסה״כ log(m) צמתים. אולם את השורש אנו לא מסמנים לעולם, ולכן בסה״כ log(m) – 1.

בסה״כ נקבל: $\boldsymbol{Potential = \log(m) + 1 + 2 * (\log(m) - 1) = 3\log(m) - 1}$.

סעיף ד׳:

מספר פעולות הlink: m-1. הסבר: נטען כי ההסבר מסעיף א׳.ב תקף – שכן השינוי באיברים עליהם נבצע decreaseKey חל רק לאחר ביצוע פעולת deleteMin, אשר במסגרתה מתבצעות פעולות הlink, על כן לא משפיע עליהן.

מספר פעולות הcut: 0. הסבר: בכל פעולת decreaseKey, הצומת עליה נבצע את הפעולה תהיה צומת על הענף השמאלי ביותר של העץ (מן השורש ומטה), ועל כן המבנה התקין של העץ יישמר. קיבלנו סדרת צמתים המקיימות יחס סדר תקין (כלומר הבן תמיד גדול מהאב, לפני ביצוע decreaseKey), ולאחר הפעלת decreaseKey עם **delta קבוע על כל אחד מן הצמתים**, בהכרח נקבל כי יחס הסדר יישמר. לכן, לא יתבצעו חיתוכים.

הפוטנציאל: נזכר כי הנוסחה לחישוב הפוטנציאל הינה $Potential = numOfTrees + 2 \cdot Marked$.

numOfTrees = 1. לאחר deletemin() הערימה תכיל עץ אחד, וכפי שטענו לעיל, לאחר decreaseKey לא ייתבצעו פעולות cut, לכן מספר העצים יישאר 1.

Marked = 0. מההסבר אודות מספר פעולות הcut, נקבל כי לא נסמן (mark) שום צומת, שכן פעולת הmark מתבצעת רק במסגרת הcut.

לאחר סדרת הפעולות, נקבל: $\boldsymbol{Potential = 1}$.

סעיף ה׳:

מספר פעולות הlink: 0. הסבר: link מתבצעת במסגרת deleteMin(), ואנו לא מבצעים אותה, אזי מספר הlink הוא 0.

מספר פעולות הcut: 0. הסבר: את פעולת הcut אנו מבצעים, רק כאשר פעולת הdecreaseKey מפרה את המבנה התקין של העץ. אולם, בסעיף הנוכחי, הערימה היא אוסף של עצים מגודל 1 בלבד, על כן פעולת decreaseKey על צומת, לא תפר את המבנה של העץ בו הוא נמצא, שכן עבור עץ בודד (בעל צומת 1) הוא תמיד יקיים מבנה תקין עם עצמו.

הפוטנציאל: נזכר כי הנוסחה לחישוב הפוטנציאל הינה $potential = numOfTrees + 2 \cdot Marked$. לאחר סדרת הפעולות, נקבל:

numOfTrees = m+1. הערימה מכילה עצים מגודל 1 עבור כל מפתח, ובסה״כ m+1 מפתחות.

Marked = 0. לא מבצעים פעולות cut, לכן בפרט לא נבצע אף פעולת mark.

לאחר סדרת הפעולות, נקבל: $Potential = m + 1$.

סעיף ו׳:

מספר פעולות הlink: מנימוק זהה לסעיפים לעיל (ג׳, ד׳) נקבל כי מספר פעולות הlink הוא **m-1**.

מספר פעולות הcut: עד להפעלת הפעולה שהתווספה לנו (כלומר לא כולל decreaseKey(m-2, m+1)) נקבל מניתוח זהה לסעיפים לעיל log(m) פעולות cut. כעת נביט בשלב שהתווסף לנו: ממבנה העץ, כפי שתיארנו לעיל, נקבל כי הצומת אשר תכיל את המפתח m-2 תהיה הצומת השמאלית והתחתונה ביותר בעץ (כלומר השמאלית ביותר בענף שיוצא שמאלה מן השורש). בסעיף ג׳ ראינו, כי לאחר הפעולה (3), כל הצמתים על הענף הנ״ל יהיו מסומנים. כלומר, כאשר נרצה לבצע decreaseKey על השמאלי ביותר (וכן קל לראות כי הפחתת ערכו ב(m+1) תפגע במבנה התקין של העץ) נאלץ לבצע סדרת פעולות cut על כל אבותיו בענף הנ״ל (שכן כולם מסומנים) מלבד השורש ** (מהגדרת התוכנית, לעולם לא יהיה מסומן). בענף הנ״ל יש **log(m)** צמתים, שכן בעץ בינומי מגודל m=2^k מהסדר הנ״ל (העץ בהכרח בינומי, שכן לאחר פעולת deleteMin הוא הופך לכזה), גובה העץ הינו k (ראינו בכיתה), וכן k=log(m). על כן נבצע log(m) – 1 פעולות cut (שכן על השורש לא נבצע, כפי שציינו ב**), ובסה״כ ביצענו log(m) + log(m) – 1 **פעולות cut, כלומר 2log(m) -1.**

הפוטנציאל: נזכר כי הנוסחה לחישוב הפוטנציאל הינה $potential = numOfTrees + 2 \cdot Marked$. לאחר סדרת הפעולות, נקבל:

numOfTrees = 2log(m). לפני הפעלת הdecreaseKey שהתווסף, היו לנו log(m) עצים (ראינו בסעיף ג׳, וכן לא כולל העץ ה״גדול״ עליו נבצע כעת את הdecreaseKey הנוסף).

ראינו כי כעת נבצע עוד log(m) – 1 פעולות cut, כלומר ייתווספו לנו Log(m) – 1 עצים, וכן נוסיף את העץ המקורי שנותר לנו. לכן בשלב זה התווספו לנו log(m) -1 + 1 = log(m) עצים.

ובסה״כ קיבלנו – numOfTrees = 2log(m).

Marked = 0. כל המסומנים שקיבלנו לאחר ביצוע פעולה (3) בסדרת הפעולות, ייהפכו לשורשים לאחר פעולת ביצוע decreaseKey שהתווספה, ושורש אינו מסומן, על כן נקבל כי כל המסומנים ייהפכו ללא מסומנים, ובסה״כ 0 מסומנים.

לאחר סדרת הפעולות, נקבל: $Potential = 2\log(m)$.

העלות היקרה ביותר של פעולת decreaseKey: מההסבר לעיל, פעולת הdecreaseKey שהתווספה לסעיף זה, תגרור log(m) – 1 פעולות cuts (לעומת כל שאר פעולות הdecreaseKey שהן מסיבוכיות O(1) כפי שתיארנו לעיל).

case	totalLinks	totalCuts	Potential	decreaseKey max cost
(c) original	m-1	Log(m)	3log(m) - 1	(skip)
(d) decKey(m-2^i) (skip)	m-1	0	1	(skip)
(e) remove line #2 (skip)	0	0	m+1	(skip)
(f) added line #4	m-1	2log(m) - 1	2log(m)	Log(m) - 1

שאלה 2:

סעיף א':

M	Run-Time(ms)	totalLinks	totalCuts	Potential
728	9	723	0	6
6560	22	6555	0	6
59,048	78	59,040	0	9
531,440	336	531,431	0	10
4,782,968	4732	4,782,955	0	14

סעיף ב':

נשים לב שאנו מבצעים m הכנסות כאשר הסיבוכיות של כל הכנסה היא O(1) לכן סה״כ סיבוכיות O(m) לאחר מכן אנו מבצעים כ $\frac{3m}{4}$ מחיקות כאשר סיבוכיות כל מחיקה היא *amortized(logm)* לכן נקבל שסיבוכיות המחיקות היא גדולה שווה ל:

$$O\left(\frac{3m}{4}\log\left(\frac{m}{4}\right)\right) = \theta(mlogm)$$

ולכן נקבל שסה״כ סיבוכיות הפעולות כתלות ב-*m* היא *O(mlogm)*

סעיף ג':

נשים לב תחילה שכמות פעולות ה-cut היא אפס, כי איננו מבצעים בסדרת פעולות זו decrease key כלל אלא רק מוחקים את המינימום כל פעם, פעולה אשר אינה מבצעת cut.

כמות פעולות ה-LINKS:

נשים לב שפעולת הלינק מתבצעת רק במחיקה הראשונה כיוון שלפני המחיקה הראשונה יש לנו רשימה מקושרת של צמתים כאשר היא ממויינת בסדר יורד. לכן במחיקה הראשונה נבצע חיבורים כך שנקבל ערימה בינומית תקינה כלומר עץ אחד מכל דרגה לכל היותר. בגלל אופן החיבור נקבל שהעצים ממויינים בסדר עולה, לכן כל פעם שנבצע מחיקות בהמשך לא יתבצעו עוד חיבורים כי כאשר נמחק את המינימום העצים שיווצרו יהיו מדרגה נמוכה יותר מהעצים שנמצאים כרגע בערימה ולכן פעולת הלינק תקרה רק במחיקה הראשונה.

נשים לב שכמות הלינקים במחיקה הראשונה היא הטור הבא $\sum_{i=1}^{\log m}\left\lfloor\frac{m}{2^i}\right\rfloor$ נשים לב שזה שווה לm פחות מספר הביטים שהם אחד בייצוג הבינארי של m . כלומר מספר הלינקים יהיה m פחות מספר הביטים שהם אחד בייצוג בייצוג הבינארי.

הפוטנציאל:

נשים לב שכאשר אנו מבצעים את המחיקה הראשונה נקבל ערימה בינומית תקינה וכיוון שהמינימום תמיד יהיה השורש של העץ הכי קטן כל פעם, בגלל אופן ההכנסה לערימה. לכן כל פעם נקבל ערימה בינומית תקינה אחרי כל מחיקה וכיוון שאנו לא מבצעים חיתוכים כלל אז הפוטנציאל הוא מספר העצים. נשים לב שכיוון שזו ערימה בינומית תקינה נקבל שמספר העצים יהיה כמספר הביטים אשר הם אחד בייצוג הבינארי של המספר אשר נקבל אחרי המחיקות כלומר סכום האחדות בייצוג הבינארי של:

$$\frac{m}{4} + 1$$