מסמך תיעוד וניסויים

שם מגיש: מתן סומך | ת"ז: 213120744

שם מגיש: שון סירוטה | ת"ז: 325027589

מסמך תיעוד

תיאור המחלקה המומשה

מימשנו את המחלקה Heap שמתארת אובייקט של ערימה.

הבנאי של הערמה מקבל שני פרמטרים בוליאנים: lazyMelds, lazyDecreaseKeys וממש לפיהם ערימה מהסוגים:

ערימה בינומית, ערימה בינומית עצלה, ערימת פיבונאצ'י ערימה בינומית עם ניתוקים.

בניגוד למימוש הרגיל של ערימה בינומית, בה פעולת Meld ממומשת באמצעות "סכום בינארי" של הערימות, פעולת הMeld מומשה על ידי successive linking.

בנוסף לתיאור ערמה כללית השתמשנו בשדות: מצביע למינימום שמאפשר לנו גישה לערימה, כמות הצמתים בערמה (size), עלות של heapifyCost, כמות צמתים מסומנים וכמות חיתוכים.

על מנת לתאר צומת בערמה השתמשנו בשדות: מצביע לילד, מצביע לאיבר הבא, מצביע לאיבר קודם, מצביע להורה, דרגה של צומת ושדה בוליאני שמעיד האם האיבר מסומן. את המפתח והערך של הצומת שמרנו במחלקה HeapItem כאשר הוא מופיע כשדה של צומת ומקושרת לצומת.

ניתוח פעולות מרכזיות

Heap(lazyMelds,DecreaseKeys)

סיבוכיות: $O(1)$

הבנאי של הערמה מקבל שני פרמטרים בוליאנים: lazyMelds, lazyDecreaseKeys וממש לפיהם ערימה מהסוגים:

ערימה בינומית, ערימה בינומית עצלה, ערימת פיבונאצ'י ערימה בינומית עם ניתוקים.

בבנאי אנחנו מאתחילים את השדות הבאים להיות 0:

size, numTrees, totalMarkedNodes, totalLinks, totalCuts, totalHeapifyCosts

Insert(k,info)

סיבוכיות: בהתאם למימוש של הפונקציה meld.

הכנסת איבר לערמה מתבצעת באמצעות הפונקציה meld ולכן הסיבוכיות שלה היא בהתאם לסיבוכיות של meld.

- אם lazyMelds = true אזי הסיבוכיות הינה $O(1)$
- אם lazyMelds = false אזי הסיבוכיות הינה $O(\log n)$ מכיוון שאנו מבצעים successive linking.

findMin()

סיבוכיות: $O(1)$

אנחנו שומרים מצביע למינימום.

deleteMin()

סיבוכיות:

	ערימה בינומית	ערימה בינומית עצלה	ערימת פיבונאצ'י	ערימה בינומית עם ניתוקים
WC	$O(\log n)$	$O(n)$	$O(n)$	$O(n)$
Amortized	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$

על מנת למחוק את איבר המינימום מהערמה אנחנו מבצעים את השלבים הבאים:

(1) יצירה ערמה של ילדיו של המינימום - יכול לקחת כדרגה של הצומת המינימלית.

(2) מחיקה של המינימום מהערמה - מבוצע על ידי החלפת מצביעים ולכן לוקח $O(1)$.

(3) חיבור של הערמה של הילדים לערמה שנשארה - מבוצע על ידי החלפת צביעים ולכן לוקח $O(1)$.

(4) עושים successive linking - יכול לקחת כמות הצמתים שיש בערמה. במקרה הגרוע ראינו כי בערמת פיבונאצי וערמה בינומית עצלה פעולה זו יכולה לקחת $O(n)$ אך בזמן לשיעורין עדיין ישאר $O(\log n)$.

cut(x,y)

סיבוכיות: .

אם lazymeld=True - $O(1)$, ואם lazymeld=False - $O(\log n)$.

פעולה זו מקבלת שני צמתים ומנתקת את צומת x מהצומת y ואז מוסיפה את x כשורש לערמה באמצעות meld.

הניתוק של הצומת x מהצומת y נעשה על ידי שינוי מצביעים ולכן העלות של הפעולה היא כעלות של פעולת הmeld. נשים לב שאם lazymeld=True אזי הפעולה נעשה ב$O(1)$ ואם lazymeld=False אזי אנחנו נבצע successive linking ולכן העלות תהיה $O(\log n)$ מכיוון שכאשר lazymeld=False אזי אנחנו שומרים על האינווריאנטה בה כמות השורשים חסומה על ידי $\log n$.

cascadingCut(y)

סיבוכיות:

- אם Lazymeld=false: $O(\log^2 n)$

  מבנה הערימה מוגבל ולכן העומק לכל היותר $O(\log n)$. בכל קריאה רקרוסיבית אנו קוראים לcut שקוראת לmeld ומפעילה successive linking ולכן יכול לקחת $O(\log^2 n)$.

- אם Lazymeld=true אזי מבנה הערימה אינו מוגבל ולכן העומק לכל היותר $O(n)$ ולכן זו סיבוכיות הפעולה.

פעולה זו מבצעת cascadingCut כפי שראינו בכיתה. אנחנו מתחילים מצומת y וכל עוד ההורה של y הוא צומת מסומן אז מנתקים את y מהערמה וממשיכים באופן רקורסבי להורה של y.

decreaseKey(x,d)

- אם Lazymeld=falseLazyDecreaseKey=true, אז הסיבוכיות זמן במקרה הגרוע הינה: $O(\log^2 n)$:

  במקרה הגרוע סיבוכיות זמן הינה כמות החיתוכים $\times$ זמן ריצה WC של פעולת meld. במקרה הנ"ל, מכיוון שאנו מבצעים consolidate אז מבני הערימה נשאר מוגבל ולכן כמות החיתוכים חסומה על ידי $O(\log n)$ ומשך זמן ריצה WC של meld הינו $O(\log n)$.

- אם Lazymeld=trueLazyDecreaseKey=true, אז הסיבוכיות זמן במקרה הגרוע הינה: $O(n)$:

  סיבוכיות הזמן היא פשוט כמות החיתוכים. מכיוון שמבנה הערימה אינו מוגבל, כמות החיתוכים חסומה על ידי $O(n)$.

- אם LazyDecreaseKey=false, Lazymeld=false אז הסיבוכיות זמן במקרה הגרוע הינה: $O(logn)$:

  במקרה זה אנו מבצעים פעולת HeapifyUp ומבנה הערימה מוגבל ולכן הגובה חסום על ידי $O(\log n)$

- אם LazyDecreaseKey=false, Lazymeld=true אז הסיבוכיות זמן במקרה הגרוע הינה: $O(n)$:

  במקרה זה אנו מבצעים פעולת HeapifyUp ומבנה הערימה אינו מוגבל ולכן הגובה חסום על ידי $O(n)$

delete(x)

סיבוכיות:

	ערימה בינומית	ערימה בינומית עצלה	ערימת פיבונאצ'י	ערימה בינומית עם ניתוקים
WC	$O(\log n)$	$O(n)$	$O(n)$	$O(\log^2 n)$
Amortized	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$

אנו מבצעים את הפעולה delete באמצעות decreseKey ו deleteMin ולכן הסיבוכיות בהתאם לסיבוכיות של פעולות אלה.

meld(heap2)

סיבוכיות:

- lazyMelds=false - טבלת סיבוכיות מופיעה תחת הניתוח של consolidate (ראה למטה)
- lazyMelds=True - $O(1)$

הפעולה ממומשת על ידי concat (שרשור הרשימות) ובמידה וְ lazyMelds=false אזי מבצעים גם consolidate.

פעולת הconcat לוקחת זמן קבוע ולכן הסיבוכיות בהתאם לפעולה consolidate.

removeNode()

סיבוכיות: $O(1)$

פעולה זו מוחקת מסירה מהערמה את השורש של המינימום ומחיזרה ערמה חדשה רק עם המינימום. פעולה זו היא פעולת עזר לפונקציה consolidate. הסרת המינימום נעשה על ידי שינוי מצביעים ולכן הסיבוכיות של פעולה זו קבועה.

concat(H)

סיבוכיות: $O(1)$

פעולה זו מקבלת ערמה נוספת ומשרשת את שתי הערמות לערמה אחת. השרשור נעשה על ידי שינוי מצביעים ולאחר מכן השוואת המינימום לקביעת המינימום של הערמה המשורשת. שינוי מבצביעים והשוואת מינימוסמים לוקחים זמן קבוע ולכן סיבוכיות הפעולה הינה $O(1)$.

consolidate()

סיבוכיות:

	ערימה בינומית	ערימה בינומית עצלה	ערימת פיבונאצ'י	ערימה בינומית עם ניתוקים
WC	$O(\log n)$	$O(n)$	$O(n)$	$O(n)$
Amortized	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$

(1) תחילה אנחנו מאתחילים מערך עזר D. אנו יודעים כי הדרגה המקסימלית של צומת בערימת פיבונאצי עם n צמתים חסומה על ידי $D(n) \leq \lfloor \log_{\Phi} n \rfloor$ ולכן מערך בגודל $\log_{\Phi} n + 1$ מספיק לנו להבטיח שלא נחרוג מגבולות המערך. נשים לב שחסם זה נכון גם עבור הערמות.

(2) אנחנו עוברים על שורשי הערמה ובכל שלב עוברים על השורש x, מסירים אותו מהערמה בעזרת המתודה removeNode (שלוקחת זמן קבוע) ושמים אותו במערך D לפי גודלו דרגתו. כאשר x הוא עם דרגה d כך שכבר יש צומת בתא זה במערך אז אנחנו מבצעים linking וממשיכים לבדוק במיקום החדש במערך עד שמוצאים תא ריק.

עלות שלב זה הינה $O(T_0 + D(n))$ כאשר T_0 הינו מספר השורשים לפני האיחוד.

(3) בשלב זה אנחנו מקבלים שהערמה שהתחלנו איתה ריקה ובידנו מערך D כך שבכל תא לא ריק במערך יש שורש מדרגה ייחודית. אזי אנחנו עוברים על המערך ומאחדים את כלל השורשים לערמה אחת. במהלך המעבר, מעדכנים את מצביע ה-min של הערמה.

עלות שלב זה הינה כגודל המערך כלומר $O(D(n))$.

נשים לב שבכל אחת מהערמות מלב הערימה הבינומית יכולים להיות לנו $O(n)$ שורשים בשלב 2 שנצטרך לעבור עליהם ולכן הWC של כלל הערמות מלבד הערימה הבינומית הינו $O(n)$. ערימה בינומית אוכפת סדר בכל רגע ולכן בכל רגע אין לנו יותר מ$O(\log n)$ שורשים בערמה.

heapifyUp(x)

סיבוכיות:

- אם Lazymeld=false אזי מבנה הערימה מוגבל ולכן העומק מוגבל ולכן הגובה חסום על ידי $O(\log n)$, וזו סיבוכיות הפעולה.
- אם Lazymeld=true אזי מבנה הערימה אינו מוגבל ולכן העומק חסום על ידי $O(n)$, וזו סיבוכיות הפעולה.

המתודה מקבלת מצביע לצומת x ומבצעת עליו heapifyUp, כלומר היא בודקת את ערך המפתח שלו ומבעבעת אותו למעלה עד למקום המתאים לו. הפעפוע למעלה נעשה באמצעות החלפת מציעים של הHeapItem ולכן זו עלות זו קבועה אך כמות הפעפוע חסומה על ידי עומק של הצומת x.

פעולות "שמירת נתונים"

את המתודות:

size(), numTrees(), totalMarkedNodes(), totalLinks(), totalCuts(), totalHeapifyCosts()

ערכים אלה מופיעים כשדות של הערמה. תחילה אנחנו מעדכנים את כל הערכים להיות 0 ובמהלך הפעולות השונות מעדכנים ערכים אלה בהתאם. כך שאנחנו מחזירים את כל הפעולות האלה ב$O(1)$.

הערה. הוספנו נספח שמאתר כיצד אנחנו שומרים על שדות אלה במהלך ביצוע הפעולות השונות.

מסמך ניסוי

סעיף 1

נסמן בT_0 את כמות השורשים בערמה לפני ביצוע פעולה.

נסמן בT_1 את כמות השורשים בערמה לאחר ביצוע פעולה.

נסמן בL את כמות הLinkים

נסמן בm את כמות הצמתים המסומנים.

נסמן בm' את כמות הצמתים המסומנים לאחר הפעולה.

נשים לב כי מכיוון LazyMeld=True אנחנו שומרים על כך שכמות השורשים חסומה על ידי $\log n$ לאחר כל פעולה.

נגדיר פונקציית פוטנציאל:

$$\Phi = (\log(n) + 1) \cdot \#(\text{number of mark nodes})$$

<u>ניתוח Insert:</u>

בפעולת Insert אנו מבצעים meld ולכן מתבצע consolidate.

נשים לב כי מתקיים $L \leq T_0$ שהרי כמות הlinkים היא לכל היותר ככמות השורשים. כמות הצמתים המסומנים אינה משתנה לאורך הפעולה.

אזי:

$$\widehat{c} = c + \Phi_{After} - \Phi_{before} = (L + T_0 + 1) + (2T_1 + (\log n + 1)m) - ((\log n + 1)m) \leq$$

$$T_0 + T_0 + 1 + 2\log n + \log n \cdot m + m - \log n \cdot m - m = 4\log n + 1 \leq 5\log n$$

ולכן פעולת Insert לוקחת זמן של $O(\log n)$ לשיעורין.

<u>ניתוח decreaseKey:</u>

נסמן בc את כמות החיתוכים שבוצעו. אזי:

נסמן בm' את כמות הצמתים המסומנים לאחר הפעולה. אזי $m' = m - c$ או $m' = m - c + 1$ חיתוכים ובכל מקרה $m' \leq m - c + 1$.

אחרי כל חיתוך אנחנו מבצעים meld ולכן העלות הינה $c + T_0 + L + (c-1)\log n$ שהרי הmeld הראשון יקח לנו ככמות העצים וכמות הlinkים שביצענו ולאחר מכן נקבל ערמה "תקינה" וכל meld יקח לנו $\log n$. כמו בפעולת insert נשים לב שמתקיים $L \leq T_0$.

אזי:

$$\widehat{c} = c + \Phi_{After} - \Phi_{before} = (c + T_0 + L + (c-1)\log n) + ((\log n + 1) \cdot m') - ((\log n + 1) \cdot m) \leq$$

$$(c + T_0 + L + c\log n - \log n) + (\log n + 1)(m - c + 1) - ((\log n + 1) \cdot m) \leq$$

$$(c + T_0 + T_0 + c\log n - \log n) + (m\log n - c\log n + \log n + m - c + 1) - (m\log n + m) =$$

$$2\log n + 1 \leq 3\log n$$

ולכן פעולת decreaseKey לוקחת זמן של $O(\log n)$ לשיעורין.

ניתוח deleteMin:

בפעולה זו אנחנו מוחקים את המינימום, מוסיפים את ילדיו כשורשים לעץ ואז עושים פעולת consolidate.

אזי:

$$\widehat{c} = c + \Phi_{After} - \Phi_{before} = (T_0 + \log n - 1 + L) + (\log n + 1)(m) - ((\log n + 1)m)$$

נשים לב כי $L \leq T_0 + \log n - 1$ שהרי כמות הלינקים יכולה להיות לכל היותר ככמות השורשים. אזי:

$$\leq T_0 + \log n - 1 + L \leq \log n + \log n - 1 + 2\log n \leq 5\log n$$

ולכן קיבלנו כי העלות לשיעורין של פעולה זו הינה $O(\log n)$.

ניתוח findMin:

אנחנו שומרים מצביע למינימום ולכן העלות היא $O(1)$.

ניתוח delete:

אנו מבצעים את הפעולה delete על ידי decreaseKey ו deleteMin ולכן העלות לשיעורין שלו מקסימלית מבינהם, כלומר $O(\log n)$.

לסיכום:

פעולה	עלות לשיעורין
Insert	$O(\log n)$
decreaseKey	$O(\log n)$
deleteMin	$O(\log n)$
findMin	$O(1)$
delete	$O(\log n)$

סעיף 2

פעולה	ערימה בינומית	ערימה בינומית עצלה	ערימת פיבונאצ'י	ערימה בינומית עם ניתוקים
Insert	$O(1)$	$O(1)$	$O(1)$	$O(\log n)$
decreaseKey	$O(\log n)$	$O(\log n)$	$O(1)$	$O(\log n)$
deleteMin	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
findMin	$O(1)$	$O(1)$	$O(1)$	$O(1)$
delete	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$

סעיף 3

	ערימה בינומית	ערימה בינומית עצלה	ערימת פיבונאצ'י	ערימה בינומית עם ניתוקים
ניסוי 1	$O(n)$	$O(n)$	$O(n)$	$O(n \log n)$
ניסוי 2	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
ניסוי 3	$O(n \log n)$	$O(n \log n)$	$O(n)$	$O(n \log n)$

סעיף 4

בחרנו להשאיר את ערכים אינם מעוגלים מכיוון שערכים אשינם מעוגלים יכולים גם להראות לנו מתי ערך מסויים נשאר קבוע בין הריצות השונות (לדוגמה גודל הערמה בסוף) למתי ערך מסויים תלוי באופן הכנסת האיברים לערמה ובהוצאתם מהערמה. לדוגמה בניסוי השני בערימת בינומית עם ניתוקים קיבלנו שכמות העצים הממוצעת הייתה 2.9, דבר שמעיד שהייתה קריאה בה מספר העצים בסיום היה 2 ואינו קבוע 3.

ניסוי 1

	ערימה בינומית	ערימה בינומית עצלה	ערימת פיבונאצ'י	ערימה בינומית עם ניתוקים
זמן ריצה (מילישניות)	338.6	204.2	183.95	341.05
גודל הערמה בסיום	464645	464645	464645	464645
מספר העצים בסיום	9	9	9	9
מספר חיבורים	464653.6	464636	464636	464653.45
מספר חיתוכים	0	0	0	0
סך עלויות heapify up	0	0	0	0
עלות מקסימלית לפעולה	18	464636	464636	18

ניסוי 2

	ערימה בינומית	ערימה בינומית עצלה	ערימת פיבונאצ'י	ערימה בינומית עם ניתוקים
זמן ריצה (מילישניות)	1949.85	2145.45	569.7	995.8
גודל הערמה בסיום	46	46	46	46
מספר העצים בסיום	4	4	3.35	2.9
מספר חיבורים	7552542.85	7552355.5	748826.25	1408273.9
מספר חיתוכים	0	0	748783.6	748766
סך עלויות heapify up	4149491.6	4149366.35	0	0
עלות מקסימלית לפעולה	35.5	464636	464636	39.6

ניסוי 3

	ערימה בינומית	ערימה בינומית עצלה	ערימת פיבונאצ'י	ערימה בינומית עם ניתוקים
זמן ריצה (מילישניות)	368.55	150.1	149.8	360.5
גודל הערמה בסיום	464644	464644	464644	464644
מספר העצים בסיום	8	8	11.7	10.55
מספר חיבורים	464670.5	464653.35	511576.75	511602.75
מספר חיתוכים	0	0	46944.45	46951.75
סך עלויות heapify up	116856.7	116927.2	0	0
עלות מקסימלית לפעולה	18	464636	464636	18

סעיף 5

סעיף א

כן כפי שניתן לראות יש הבדלים בביצועים של הניסויים בין הערימות, ויש גם דמיון, לפי המכנה המשותף בין שתי ערימות.

למשל, בין ערימה בינומית עצלה וערימת פיבונאצ'י המשותף הוא – lazyMeld, מה שגורם לעלות הפעולה המקסימלית להיות זהה (בפעולת ה-deleteMin לאחר ההכנסות). באופן דומה כך גם הערימה הבינומית (במימוש שלה כפי שביצענו כאן ולא כפי שלמדנו בכיתה) והערימה הבינומית עם הניתוקים זהות, עקב ה-successive linking שמבוצע בכל הכנסה. בנוסף, ניתן לראות כי עלויות ה-heapify up, יכולות להיות גבוהות מאוד ואכן עדיף להשתמש successive linking (רואים זאת בעיקר בניסוי השני, אך גם בשלישי). ניתן לראות לפי העלות המקסימלית לפעולה את ההבדל בין העלות בניתוח לשיעורין לבין הworst case, בכל שבערימת פיבונאצ'י ובערימה הבינומית העצלה, עלות הפעולה המקסימית היא הגבוהה ביותר (ה- deleteMin לאחר ההכנסות), אך בסך הכל, כמות הפעולות (לפחות בערימת פיבונאצ'י, בלי ה-heapify up, שעלול להיות יקר) נמוכה הרבה יותר מהאחרות. כלומר, מדי פעם משלמים עלות גבוהה הרבה יותר, אך זה קורה רק במקרים "נדירים".

סעיף ב

התוצאות שקיבלנו מאששות את מה שענינו בסעיף 3.

ראשית נעיר כי הניתוח בסעיף 3 לגבי ערימה בינומית הוא לפי מה שלמדנו בכיתה ולכן, לפחות בניסוי הראשון (שם אין פעולת decreaseKey), הערימה הבינומית והערימה הבינומית עם הניתוקים דומות מאוד. לכן גם אין קשר אמיתי בין התוצאות שקיבלנו עבור הערימה הבינומית לבין הניתוח בסעיף 3.

בניסוי הראשון צפינו שהערימה הבינומית העצלה וערימת פיבונאצ'י יהיו מהירות יותר (ואכן שתי הערימות זהות בניסוי הזה באופן יחסית מאחר שלא מבוצע בהן decreaseKey). כך שבמקרה זה החיזוי אכן התממש. נעיר כי מבחינת כמות חיבורים, הייתה כמות דומה מאוד, אך הזמן היה שונה (מאחר שבערימה הבינומית עם ובלי הניתוקים עוברים על כל מערך הסלים בכל הכנסה, גם אם אין חיבורים...). בנוסף ניתן לראות כפי שהערנו כי הפעולה היקרה היא זהה עבור ערימת פיבונאצ'י והבינומית העצלה, וזאת מפני שברוב הפעולות "משלמים" קצת עד שמגיעה הפעולה היקרה, בניגוד לשתי הערימות האחרות, שם משנים קצת בכל פעולה (כמות חיבורים קטנה בכל successive linking), אבל מבחינת זמן ריצה זה לוקח יותר.

בניסוי השני, החיזוי היה שזמן הריצה יהיה דומה – $(O(n\log n))$. ניתן לראות שהזמנים אינם דומים כלל, וניתן להסביר את זה על ידי הקבועים. הקבועים עבור ערימת פיבונאצ'י והערימה הבינומית עם הניתוקים נמוכים מהערימות האחרות, בעיקר עקב ביצוע ה-cascading cuts, שחוסך את עלות ה-heapify up, שכאן היא מאוד גדולה (כי מחקנו את המינימום – ה-heapify up הוא מקסימלי). לכן גם זמן הריצה הרבה יותר מהיר עבור שתי הערימות עם הcascading cuts, אם כי ערימת פיבונאצ'י אפילו מהירה יותר מהערימה הבינומית עם הניתוקים עקב ה-lazyMeld(פעולת ה-decreaseKey זולה הרבה יותר בערימת פיבונאצ'י, ולכן גם המחיקות...). לכן, למרות שיש פער בזמני הריצה, הוא לא סותר את החיזוי התיאורטי, אלא מלמד אותנו שהקבועים בניתוח חשובים לניתוח זמן הריצה בפועל (הניתוח האימפטוטי מעיד על קצת גידול זמן הריצה, ולא על זמן הריצה בפועל).

בניסוי השלישי, חזינו כי ערימת פיבונאצ'י תרוץ בזמן הכי מהיר, וזה אכן קרה, אך בפער קטן מאוד מהערימה הבינומית העצלה. אפשר להסביר זאת בכך שההבדל בין שתי הערימות הוא במחיקה, ובעוד שבערימה הבינומית העצלה משלמים בו במקום, בערימת פיבונאצ'י דוחים את התשלום לסוף, ומשלמים הרבה בסוף (ב-deleteMin האחרון עוברים על מערך גדול יותר ומבצעים יותר חיבורים), כך שיוצא שזמן הריצה דומה בין שתי הערימות בסופו של דבר. לעומת זאת, בערימה הבינומית עם הניתוקים ובלי הניתוקים זמן הריצה היה ארוך יותר, כאשר בערימה עם הניתוקים ביצענו הרבה פעמים successive linking) כשגם אם חיברנו מעט, זה עולה אומגה גדולה של $\log n$ כפי שציינו), ובזו בלי הניתוקים, במקום ה-successive linking ב- decreaseKey ביצענו heapify up, מה שגרם לזמנים להיות דומים.

סעיף ג

כפי שציינו, היו הבדלים בעלות הפעולה המקסימלית, שנובעים מהתכונות של כל ערימה :

הערימות ש"מסדרות מיד" – הערימה הבינומית עם ובלי הניתוקים, והערימות שדוחות לסוף – הערימה הבנומית העצלה וכמובן שערימת פיבונאצ'י. בערימות אלה הפעולות המקסימליות היו זהות, כיוון שהן בדיוק מבוצעות ב-deleteMin שאחרי כל ההכנסות. לעומת זאת הערימה הבינומית עם ובלי הניתוקים מבצעות successive linking בכל פעם (decreaseKey ,deleteMin), ולכן בכל פעם הערימה יחסית מסודרת, כלומר צריך לבצע פחות שינויים, אך משלמים בזמן ריצה גבוה.

נספח

שמירה על ערכים במבנה נתונים

להלן פירוט של אופן שמירה של הערכים הבאים :

size, numTrees, totalMarkedNodes, totalLinks, totalCuts, totalHeapifyCosts

אנחנו שומרים ערכים אלה מבלי לפגוע בסיבוכיות של שאר הפעולות.

שמירת הערךsize

- כאשר מבצעים insert בעת יצירת ערמת הסינגלטון שלנו נעדכן את הsize שלו להיות 1 על מנת שבמהלך פעולת הconcat נעלה את כמות הצמתים ב1.
- בסוף פעולת הdeleteMin נקטין את כמות הצמתים ב1.
- נשים לב שפעולת הdelete קוראת למתודה deleteMin ולכן אין צורך להוריד גם שם את גודל המערך.
- במהלך ביצוע concat נאחד את כמות הצמתים של שתי הערמות.

שמירת הערךnumTrees

- בפעולת concat מאחדים את כמות השורשים של שני העצים
- בפעולת insert כאשר יוצרים את הערמה עם הצומת הבודדת מעדכנים את כמות השורשים של הערמה זו להיות 1 - כך מובטח לנו שבמהלך הmeld כאשר מבצעים concat אכן כמות השורשים תגדל ב1.
- במהלך פעולת deleteMin אנחנו מוחקים את המינימום ומוסיפים לעץ את כל הילדים של המינימום. אזי לפני קריאה לפעולת consolidate נעדכן את כמות השורשים להיות :

- אם יש רק שורש אחד אז נעדכן את numOfTrees להיות MinchildNumOfTrees שזה פשוט ה$rank$ שלה מינימום.

- אם יש יותר משורש אחד אז נעדכן את numOfTrees להיות numOfTrees+MinchildNumOfTrees-1.

- כאשר מבצעים linking במהלך הconsolidate אז נוריד את כמות השורשים ב1. בכל מקרה כאשר אנחנו בונים בחזרה את הערמה נספור את כמות השורשים (שזה כמות התאים שאינם ריקים במערך).
- כאשר אנחנו מבצעים cut אז אנחנו יוצרים ערמה מהצומת שאותו מנתקים ואז עושים meld, נעדכן בערמה זו את כמות השורשים להיות 1 כדי שבעת ביצוע concat נוסיף 1 לכמות השורישם במערך.

שמירת הערךtotalMarkdNodes

- כאשר אנחנו מבצעים cut נוריד את כמות הצמתים המסומנים ב1.
- כאשר אנחנו מבצעים cascadingCut ומגיעים לצומת שאינו מסומן ולפני השורש נסמן אותו ונעלה את הכמות הצמתים המסומנים ב1.

שמירת הערךtotalLinks

- אנו מבצעים linking רק במהלך המתודה cosolidate ולכן לאחר ביצוע פעולת link נעלה את כמות הlinkים ב1.
- במהלך הפעולה concat נחבר את כמות הlinkים של שתי הערמות על מנת לשמור את ההיסטוריה של כמות הlinkים.

הערה. מבחינת design ייתכן שהיה יותר נכון לעדכן את כמות הlinkים בתחילת המתודה link אך מתודה זו נמצאת תחת המחלקה HeapNode ולכן בחרנו לעקוב אחר כמות הlinkים בדרך עקיפה.

שמירת הערךtotalCuts

- במתודה cut נעלה את כמות החיתוכים ב1.
- במהלך הפעולה concat נחבר את כמות החיתוכים של שתי הערמות על מנת לשמור את ההיסטוריה של כמות החיתוכים.

שמירת הערךtotalHeapifyCosts

- במתודה heapify, הגדרנו משתנה count שסופר את כמות הheapifyUp שביצענו והמתודה heapify מחזירה את המונה הזה. אזי, בעת קריאה למתודה heapify (שקוראת בdecreaseKey כדי לבצע heapifyUp נעשה :

```
this.heapifyCosts += heapifyUp(x.node);
```

וכך נשמור בכל קריאה לheapifyUp אנחנו מעדכנים את המונה הכללי.