מבני נתונים

תרגיל מעשי 2

FibonacciHeap

מגישים:

גיא שניידר 313119679 guyshnaider

עידו גזית 313197980 idogazit

תוכן עניינים

מחלקה HeapNode

<u>מטרת המחלקה:</u> מימוש צומת בודד השייך לערימה מסוג פיבונאצ'י שמכיל מפתח מסוג מספר שלם.

<u>שדות:</u>

שימוש	שם השדה
שומר את המפתח הייחודי של הצומת	`public int key`
שומר את הדרגה של הצומת (מספר הילדים של הצומת)	`public int rank`
שומר ערך בוליאני עבור האם הצומת מסומן או לא	`public boolean mark`
מצביע השומר הפניה לצומת האב	`public HeapNode parent`
מצביע השומר הפניה לצומת הבן הראשון	`public HeapNode child`
מצביע השומר הפניה לצומת הבא ("האח הבא")	`public HeapNode next`
מצביע השומר הפניה לצומת הקודם ("האח הקודם")	`public HeapNode prev`
מצביע השומר הפניה לצומת בערימה המקורית (לשימוש פונקציית kMin)	`public HeapNode pointer`

<u>מתודות:</u>

סיבוכיות	תיאור הפעולה	שם המתודה
$O(1)$	בנאי של המחלקה	HeapNode(int key)
	מחזירה את המפתח של הצומת	int getKey()
	מחזירה את הבן הראשון. הפעולה תחזיר null אם אין לצומת ילדים.	HeapNode getChild()
	מחזירה את ה"אח הבא". הפעולה תחזיר את הצומת עצמו אם אין לו אחים.	HeapNode getNext()
	מחזירה את ה"אח הקודם". הפעולה תחזיר את הצומת עצמו אם אין לו אחים.	HeapNode getPrev()
	מחזירה את האבא. הפעולה תחזיר null אם אין צומת אב.	HeapNode getParent()
	מחזירה את דרגת הצומת	int getRank()
	מחזירה true אם הצומת מסומנת, אחרת תחזיר false	boolean isMarked()
	מגדירה את המפתח להיות key	void setKey(int key)
	מגדירה את הדרגה להיות rank	void setRank(int rank)
	מגדירה את הבן הראשון להיות child	void setChild(HeapNode child)
	מגדירה את האבא להיות parent	void setParent(HeapNode parent)
	מגדירה את ה"אח הבא" להיות next	void setNext(HeapNode next)
	מגדירה את ה"אח הקודם" להיות prev	void setPrev(HeapNode prev)
	הפעולה מסמנת את הצומת, כלומר מגדירה mark=true	void mark()
	הפעולה מבטלת את סימון הצומת, כלומר mark=false	void unMark()
	מגדירה את מצביע pointer להיות הצומת שקיבל	void setPointer(HeapNode pointer)
	מחזירה את המצביע pointer	HeapNode getPointer()

מחלקה FibonacciHeap

מטרת המחלקה: מימוש ערימה מסוג פיבונאצ'י שמחזיקה מפתחות מסוג מספר שלם.

שדות:

שם השדה	שימוש
`public HeapNode first`	שומר מצביע לצומת הראשון בערימה
`public HeapNode min`	שומר מצביע לצומת בעל הערך המפתח בערימה
`public int size`	שומר את כמות האיברים בערימה
`public int treeCount`	שומר את כמות העצים בערימה
`public int markedCount`	שומר את כמות האיברים המסומנים בערימה
`public static int totalLinks`	מונה סטטי שסופר את כמות פעולות ה-link שבוצעו בריצת התוכנית
`public static int totalCuts`	מונה סטטי שסופר את כמות פעולות ה-cut שבוצעו בריצת התוכנית

מתודות:

	שם המתודה	תנאי קדם	תנאי סיום	תיאור הפעולה	פונקציות עזר	סיבוכיות WC
1	FibonnaciHeap()			בנאי של המחלקה		$O(1)$
2	boolean isEmpty()			בודקת אם הערימה ריקה		$O(1)$
3	HeapNode insert(int key)			הפעולה מקבל מפתח key ומכניס לערימה צומת חדש עם המפתח	HeapNode insertNode(HeapNode node)	$O(1)$
4	void deleteMin()			הפעולה מוחקת את הצומת בעל המפתח המינימלי בערימה	void connectNodes(HeapNode one, HeapNode two), void makeChildrenTrees(HeapNode x), void consolidate()	$O(n)$ Amortized: $O(\log n)$
5	void makeChildrenTrees(HeapNode x)	x הוא הצומת המינימלי בערימה שנמחק		הפעולה מוסיפה את הבנים של הצומת כעצים חדשים בערימה במקום בו הוא נמחק		$O(\log n)$
6	HeapNode link(HeapNode x, HeapNode y)	הדרגות של x ו-y שווה		הפעולה מקבלת שני צמתים ומגדירה את הצומת עם המפתח הקטן משניהם כאבא של הצומת השני הפעולה מחזירה את צומת האב	void linkChildToFather(HeapNode child, HeapNode father)	$O(1)$
7	void linkChildToFather(HeapNode child, HeapNode father)	המפתח של child גדול מהמפתח של father		הפעולה מקבלת שני צמתים ומגדירה את צומת האבא כהורה של צומת הבן	void connectNodes(HeapNode one,	$O(1)$

					HeapNode two)	
8	void consolidate()			הפעולה מסדרת מחדש את ערימת הפיבונאצי כערימה בינומית	HeapNode[] toBuckets(HeapNode x) HeapNode fromBuckets(HeapNode[] Buckets)	$O(n)$ Amortized: $O(\log n)$
9	HeapNode[] toBuckets(HeapNode x)	x הוא הצומת הראשון במערך		הפעולה מחברת את העצים השונים בערימה, ומחזירה מערך של צמתים כאשר בתא ה-i יש לכל היותר צומת אחת מדרגה i	HeapNode link(HeapNode x, HeapNode y) void connectNodes(HeapNode one, HeapNode two)	$O(n)$ Amortized: $O(\log n)$
10	HeapNode fromBuckets(HeapNode[] Buckets)			הפעולה מקבלת את המערך מ toBuckets() ומכניסה את הצמתים לערימה מחדש, מהקטן לגדול. הפעולה מחזירה מצביע לצומת הראשון בערימה	void connectNodes(HeapNode one, HeapNode two)	$O(\log n)$
11	HeapNode findMin()		הפעולה מחזירה null עבור ערימה ריקה	הפעולה מחזירה את הצומת המינימלי בערימה		$O(1)$
12	void meld(FibonnaciHeap heap2)			הפעולה מקבלת ערימת פיבונאצ'י נוספת ומבצעת מיזוג עם הערימה המקורית	void connectNodes(HeapNode one, HeapNode two)	$O(1)$
13	void connectNodes(HeapNode one, HeapNode two)	שני הצמתים אינם null		הפעולה מקבלת שני צמתים ומבצעת שרשור שלהם ע"י הגדרת המצביעים המתאימים		$O(1)$
14	int size()			הפעולה מחזירה את כמות הצמתים בערימה		$O(1)$
15	int[] countersRep()			הפעולה מחזירה מערך שבו התא ה-i מכיל את מספר העצים מדרגה i בערימה		$O(n)$
16	void delete(HeapNode x)	הצומת x קיים בערימה		הפעולה מקבלת צומת x ומבצעת מחיקה שלו מהערימה	void deleteMin(), void decreaseKey(HeapNode x, int delta)	$O(n)$ Amortized: $O(\log n)$
17	void decreaseKey(HeapNode x, int delta)	הצומת x קיים בערימה,		הפעולה מקבלת צומת x ומקטינה את המפתח שלו ב-delta	void cascadingCuts	$O(\log n)$ Amortized: $O(1)$

		אי delta שלילי			(HeapNode x, HeapNode y)	
18	void cut(HeapNode x, HeapNode y)	צומת x קיים בעץ, צומת y הינו צומת האב של x		הפעולה מקבלת צומת x ואת צומת האב שלו y ומבצעת חיתוך הענף המקשר ביניהם בעץ	void connectNodes(HeapNode one, HeapNode two)	$O(1)$
19	void cascadingCuts(HeapNode x, HeapNode y)	צומת x קיים בעץ, צומת y הינו צומת האב של x	הפעולה נעצרת כאשר מגיעה לאב כלשהו שאינו מסומן	הפעולה מקבלת צומת x, וצומת האב שלו y חיתוכי של כל האבות המסומנים של x	void cut(HeapNode x, HeapNode y)	$O(\log n)$ Amortized: $O(1)$
20	int potential()			הפעולה מחזירה את הפוטנציאל הנוכחי, כלומר מספר העצים ועוד פעמיים מספר הצמתים המסומנים		$O(1)$
21	static int totalLinks()			פעולה סטטית אשר מחזירה את מספר פעולות ה-link שבוצעו בתוכנית		$O(1)$
22	static int totalCuts()			פעולה סטטית אשר מחזירה את מספר פעולות ה-cut שבוצעו בתוכנית		$O(1)$
23	static int[] kMin	H עץ בינומי, k<H.size()		פעולה סטטית אשר מקבלת ערימה H ומספר שלם k, ומחזירה מערך מספרים של k המפתחות הקטנים ביותר בערימה	void deleteMin(), HeapNode insertNode(HeapNode node)	$O(k(\log k + \deg(H)))$
24	HeapNode insertNode(HeapNode node)			הפעולה מקבלת צומת ומכניסה אותו לערימה		$O(1)$
25	HeapNode getFirst()		הפעולה מחזירה null עבור ערימה ריקה	הפעולה מחזירה את הצומת הראשון בערימה		$O(1)$

הערות:

1. מתודות שהמספר הסידורי שלהן מודגש בכחול הינן מתודות החובה שנדרשנו לממש, שאר המתודות הינן פונקציות עזר שתומכות במימוש.
2. מתודות שסיבוכיות זמן הריצה שלהן היא $O(1)$ הינן מתודות שמבצעות אך ורק מספר קבוע של פעולות.
3. מתודות שסיבוכיות זמן הריצה שלהן מסומנת בירוק :

insert	הפעולה משתמשת בפעולת insertNode ולכן הסיבוכיות שלה זהה לסיבוכיות פעולה זו. כלומר הסיבוכיות של הפעולה תהיה $\boldsymbol{O(1)}$.
/deleteMin	הפעולה מוחקת את הצומת המינימלי בערימה. הפעולה תכניס את ילדיו (אם יש כאלה) במקומו בעזרת makeChildrenTrees. לאחר מכן הפעולה תסדר מחדש את המערך בעזרת consolidate(). כתוצאה מכך סיבוכיות הפעולה תהיה תלויה בפעולות אלה, ובמקרה הגרוע

	תהיה $\boldsymbol{O(n)}$, וכיוון ופעולות אלה מומשו כפי שנלמד בכיתה, סיבוכיות amortized של deleteMin היא $\boldsymbol{O(\log n)}$
makeChildrenTrees	הפעולה עוברת על כל הילדים של הצומת שנמחקה בdeleteMin ומבצעת על כל אחד מספר סופי של פעולות. ראינו בהרצאה שמספר הילדים של צומת $\leq 1.4\log(n)$ לכן סיבוכיות הפעולה תהיה $\boldsymbol{O(\log n)}$
link	הפעולה מבצעת מספר סופי של בדיקות, שינוי מצביעים ושינוי שדות, ולאחר מכן קוראת ל linkChildToFather לכן סיבוכיות הפעולה תהיה זהה לסיבוכיות פעולה זו ותהיה $\boldsymbol{O(1)}$
linkChildToFather	הפעולה מקבלת שני צמתים ומבצעת מספר סופי של עדכוני מצביעים ושינוי שדות לכן סיבוכיות הפעולה תהיה $\boldsymbol{O(1)}$
consolidate	הפעולה משתמשת בפעולות *toBuckets*ו *fromBuckets* לכן הסיבוכיות שלה תהיה תלויה בהן, כלומר במקרה הגרוע תהיה $\boldsymbol{O(n)}$ **,** כיוון והפעולות ממומשות כפי שראינו בהרצאה(שקף אחרון במצגת על ערימות פיבונאצי), נובע שסיבוכיות *amortized* היא $\boldsymbol{O(\log n)}$
toBuckets	הפעולה יוצרת מערך של צמתים באורך שווה ללוגריתם על פי יחס הזהב של n כיוון וראינו בהרצאה שהדרגה המקסימלית של עץ בערימה מגודל n חסום על ידי מספר זה לאחר מכן הפעולה עוברת על כל שורשי הערימה לאחר מחיקת הצומת המינימלי, ומכניסה אותם למערך לפי דרגות. אם בהכנסה לתא כבר קיים צומת מאותה דרגה מבצעים link בין הצמתים ומעבירים את הצומת לתא הבא במערך, כאשר כל פעם מבצעים link עד שמגיעים לתא ריק. סיבוכיות הפעולה במקרה הגרוע היא $\boldsymbol{O(n)}$ **.** נבחין כי המימוש הוא כפי שנלמד בכיתה ולכן הסיבוכיות amortized של הפעולה תהיה $\boldsymbol{O(\log n)}$
fromBuckets	הפעולה עוברת על המערך שאורכו שווה ללוגריתמם על פי יחס הזהב של n, לכן אורך המערך מקיים $\leq 1.4\log(n)$ כלומר סיבוכיות הפעולה תהיה $\boldsymbol{O(\log n)}$
countersRep	הפעולה עוברת על כל שורשי הערימה, ובמקרה הגרוע בו הערימה מכילה רק עצים מדרגה 0, תעבור על n שורשים לכן סיבוכיות הפעולה תהיה $\boldsymbol{O(n)}$
delete	הפעולה משתמשת בפעולות decreaseKey ו- deleteMin במקביל ולכן במקרה הגרוע הסיבוכיות תהיה כמו של deleteMin כלומר הסיבוכיות תהיה $\boldsymbol{O(n)}$. באופן דומה הסיבוכיות amortized תהיה $\boldsymbol{O(log n)}$.
decreaseKey	הפעולה במקרה הגרוע משתמשת בפעולה cascadingCuts ולכן הסיבוכיות שלה זהה לסיבוכיות פעולה זו, כלומר הסיבוכיות תהיה $\boldsymbol{O(\log n)}$. באופן דומה הסיבוכיות amortized תהיה $\boldsymbol{O(1)}$.
cut	הפעולה מקבלת שני צמתים ומבצעת מספר סופי של פעולות בדיקה והתאמת מצביעים עבור הצמתים והערימה הקיימת. ולכן הסיבוכיות של הפעולה תהיה $\boldsymbol{O(1)}$.
cascadingCuts	הפעולה במקרה הגרוע ביותר רצה לכל הגובה של עץ בינומי מגודל n איברים, ולכן תהיה חסומה ע"י עומק העץ, ראינו כי עומק עץ בינומי הוא כגודל הדרגה שלו, כלומר log(n) ולכן הפעולה חסומה ע"י $\boldsymbol{O(\log n)}$. נבחין כי הפעולה ממומשת כמו שהוצג בכיתה ולכן הוכח כי סיבוכיות amortized של פעולה זו היא $\boldsymbol{O(1)}$.
kMin	הפעולה יוצרת ערימה צדדית ומבצעת מספר קבוע של פעולות ולאחריהן הפעולה מבצעת k איטרציות ובתוך כל איטרציה מבצעת: - מוחקת צומת מהערימה הצדדית שבה יש לא יותר מ-k צמתים ולכן שלב זה חסום ע"י $O(\log k)$. - מכניסה לתוך הערימה הצדדית את כל הבנים של הצומת שזה עתה נמחק ע"י פעולות הכנסה בגודל קבוע. לצומת שזה עתה נמחק יש לכל היותר deg(H) ילדים (שכן זאת כמות הילדים של השורש בערימה H וידוע כי לשורש יש הכי הרבה ילדים בעץ בינומי H) ולכן שלב זה חסום ע"י $O(\deg(\mathrm{H}))$. ולכן סה"כ עבור כל הפעולה סיבוכיות הפעולה תהיה: $\boldsymbol{O(k(\log k+\deg(H)))}$
insertNode	הפעולה מקבלת צומת ומבצעת מספר סופי של פעולות בדיקה והתאמת מצביעים עבור הצומת והערימה הקיימת. ולכן הסיבוכיות של הפעולה תהיה $\boldsymbol{O(1)}$.

מדידות

1. סדרת פעולות ראשונה:

Insert(m), insert(m-1), …, insert(0)

delete-min()

decrease-key($m * (\sum_{k=1}^{i} 0.5^k) + 2$) for $i = 0, \ldots \log m - 2$

decrease-key(m-1)

m	Run-Time (in miliseconds)	totalLinks	totalCuts	Potential
1024	4.973	1023	18	19
2048	10.7675	2047	20	21
4096	14.3904	4095	22	23

א. הזמן הריצה האסימפטוטי של סדרת הפעולות

ראשית מבצעים $m + 1$ הכנסות לערימה, כאשר לפי תכונות ערימת פיבונאצי שמימשנו כל הכנסה מקיימת $O(1)$ פעולות, לכן עלות כל ההכנסות היא $O(m)$.

לאחר מכן נבצע delete-min(), כאשר אנחנו בWC , כיוון והערימה מכילה $m + 1$ עצים מדרגה 0. לכן נבצע $O(m)$ פעולות. לאחר מכן נקבל בערימה עץ בינומי יחיד המכיל את כל המפתחות חוץ מ 0 שנמחק.

לבסוף נבצע $\log m$ פעולות decrease key כאשר כל פעם מבצעים את הפעולה על המפתח $m * (\sum_{k=1}^{i} 0.5^k) + 2$. נשים לב כי המפתח ה-i בעצם יוצא

$$m * \left(\sum_{k=1}^{i} 0.5^k\right) + 2 = 2^{logm} * \left(\frac{0.5 - 0.5^{i+1}}{1 - 0.5}\right) + 2 = m - 2^{logm - i} + 2$$

כלומר נבצע את הפעולה על סדרת המפתחות $2, \frac{m}{2} + 2, \frac{3m}{4} + 2, \ldots, m - 2, m - 1$ (כאשר הפעולה על המפתח האחרון מתקבלת מהשורה האחרונה בסדרת הפעולות הנתונה)

כלומר נבצע $logm$ פעולות decrease-key, כל פעם על מפתח הנמצא בצומת שונה בערימה.

ראינו בהרצאה כי הסיבוכיות ammortized של פעולה בסדרת פעולות decrease-key הוא $O(1)$ לפעולה, לכן נבצע בסופו של דבר $O(\log m)$ פעולות .

כלומר סדרת הפעולות תקיים זמן ריצה אסימפטוטי כלהלן: $O(m + m + logm) = O(m)$.

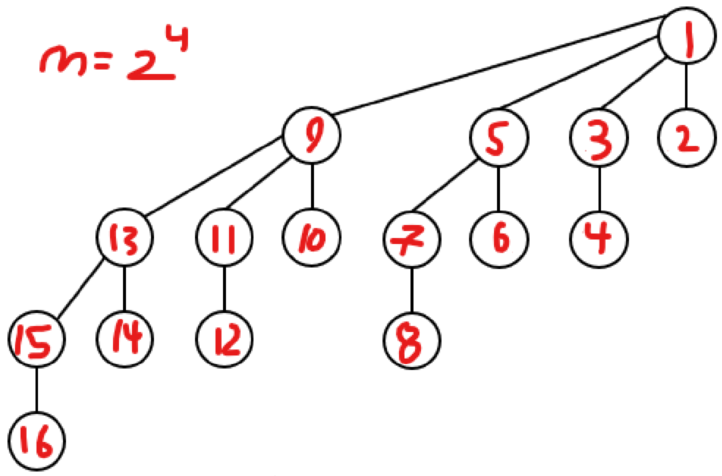
ב. מספר פעולות link וcut המבוצעות במהלך הסדרה

מספר פעולות הlink תלוי במספר הפעולות ש- delete-min() תעשה בסדרת הפעולות הנתונה. ראינו כי מספר הפעולות זה יהיה $O(m)$ בגלל שאנחנו נמצאים בWC. לכן מספר פעולות הlink יקיים $O(m)$

מספר פעולות ה cut תלוי במספר הפעולות ש-decrease-key יבצע, וראינו כי מדובר במספר המקיים $O(logm)$, לכן גם מספר פעולות cut יהיה $O(logm)$

ג. עלות פעולת decrease-key היקרה ביותר

כתוצאה מסדר הפעולות נקבל בערימה עץ בינומי יחיד מעומק $logm$ מהצורה הבאה:



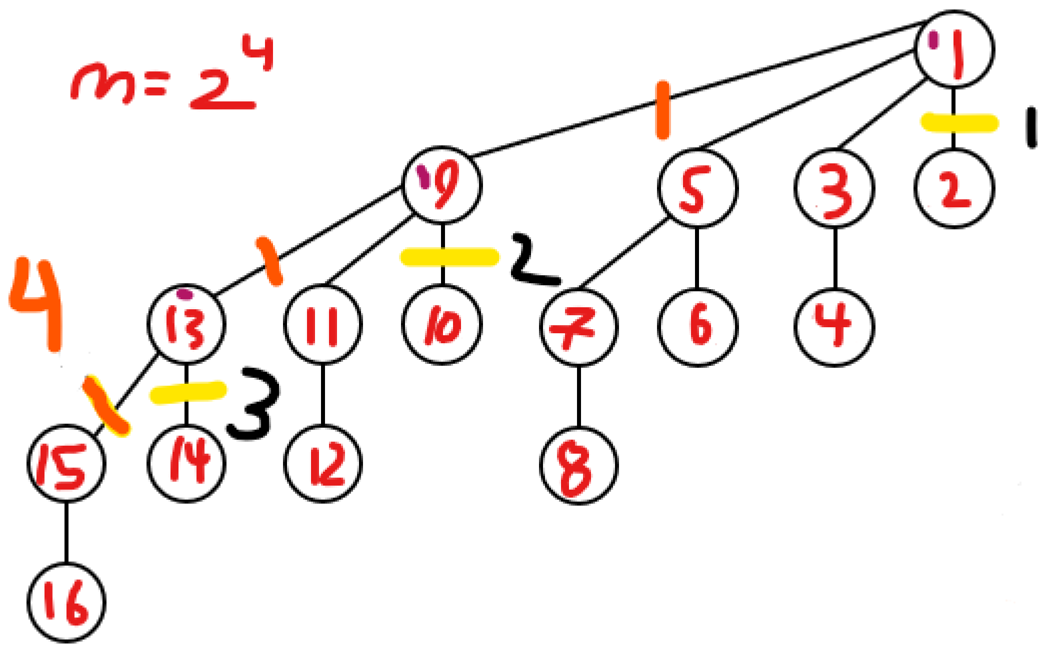
בפעולה הראשונה נבצע חיתוך של הצומת 2, שהיא בן ישיר של שורש העץ.

בפעולה השנייה נחתוך את הבן הישיר של הצומת מהדרגה השנייה הכי גבוה בעץ שהוא הבן השמאלי הישיר של השורש, ונסמן את הבן השמאלי של השורש.

כלומר במחיקה ה-i נמחק את הבן הישיר של צומת מעומק i, שהוא בן ישיר של הצומת מעומק i-1.

בפעולה האחרונה נבצע חיתוך של הצומת בעומק i שהוא הבן השני במספר של הצומת מעומק i-1 שחותכים בסדרת הפעולות, והפעולה האחרונה תגרום לכך שנבצע cascading cuts על כל הצמתים שהם בנים שמאליים של ההורה שלהם לאורך כל העץ.

נבצע בפעולה זאת $O(\text{logm})$ חיתוכים, כתלות בעומק העץ. כפי שניתן לראות בדוגמה:



כאשר המספרים השחורים מייצגים את מספר החיתוכים המתבצעים בסדרה של decreaseKey, והמספר הכתום הוא מספר הפעולה האחרונה שמתבצעת בשורה decrease-key(m-1) בסדרת הפעולות, והנקודה הסגולה אומרת כי הצומת מסומנת.

בעקבות הפעולה האחרונה מבצעים cascading cuts עד לשורש העץ, שמיוצגים לפי החיתוכים הכתומים בעץ.

האם התוצאות שקיבלנו תואמות את הציפיות?

כפי שניתן לראות, $\#links = m - 1$ לכל m, לכן אכן מספר פעולות link מקיים $O(m)$.

בנוסף $\#cuts \approx 2logm$ לכל m לכן מספר פעולות cut באמת מקיים $O(logm)$.

לכן תוצאות המדידות תואמות את הציפיות שלנו.

__2. סדרת פעולות שנייה:__

Insert(m), insert(m-1), …, insert(1)

deleteMin(), deleteMin (), …, deleteMin() (run delete min m/2 times).

M	Run-Time (in miliseconds)	totalLinks	totalCuts	Potential
1000	23.456599999999998	2882	0	6
2000	27.3962	6773	0	6
3000	31.403299999999998	10038	0	7

א. הזמן הריצה האסימפטוטי של סדרת הפעולות

ראשית מבצעים m פעולות insert, כאשר כל פעולה מבצעת O(1) פעולות. לכן נבצע O(m) פעולות בשלב זה

לאחר מכן נבצע $\frac{m}{2}$ פעולות delete-min(), כאשר כל פעם נמחק את המפתח הכי קטן בערימה, כלומר סדרת המפתחות $1,2,3, \ldots, \frac{m}{2}$. ראינו כי מספר הפעולות הממוצע לפעולה יהיה $\mathrm{O(logn)}$ לסדרה של פעולות deletemin על ערימה עם n איברים, לכן כל פעולה תבצע O(logm) פעולות בממוצע, נכפיל במספר הפעמים שמבצעים את הפעולה ונקבל כי יתבצעו $O(\frac{m}{2} logm)$ פעולות.

לכן בסופו של דבר נקבל כי סדרת הפעולות תבצע מספר פעולות אסימפטוטי כלהלן $O(m + mlogm) = O(mlogm)$

ב. מספר פעולת link וcut המבוצעות במהלך הסדרה

מספר פעולות link צריך להיות תלוי במספר פעולות ש deletemin מבצעת. לכן כפי שראינו יתבצעו $O(mlogm)$ פעולות link.

מספר הפעולות cut תלוי במספר הפעולות שdecreasekey מבצעת בסדרת הפעולות. כיוון ולא נקרא לdecreasekey בשום שלב בסדרה, יתבצעו 0 פעולות cut.

ג. פוטנציאל המבנה בסוף סדרת הפעולות

ראשית נשים לב כי כיוון ולא נבצע פעולות decreaseKey, מספר הצמתים המסומנים יהיה 0 ולכן הפוטנציאל תלוי אך ורק במספר העצים.

נשים לב כי אחרי כל deleteMin נבצע סידור מחדש של הערימה כערימה בינומית לא עצלה.

בנוסף נראה כי לא נבצע delete או decreaseKey ולא נשנה את העצים בערימה בצורה שתתן לנו עצים שאינם בדיוק בינומים(כלומר לא נחתוך עצים באמצע כפי שמימוש של decreaseKey מבצע).

לכן כפי שראינו במבנה של ערימה בינומית לכל מספר איברים n יש דרך חד חד ערכית לקבוע את מספר העצים שיהיה בערימה ומספר זה יהיה logn.

לכן כיוון ובסוף סדרת הפעולות יש לנו $\frac{m}{2}$ צמתים בערימה, יהיו לנו $O(log\frac{m}{2})$ עצים כלומר הפוטנציאל יקיים $O(logm)$.

האם התוצאות שקיבלנו תואמות את הציפיות?

כפי שניתן לראות $\#cuts = 0$, כפי שנובע מכך שלא מבצעים פעולות decreaseKey.

בנוסף, $\#links \approx \frac{m}{4} * logm$ לכל m לכן פעולות link אכן מקיימות $O(mlogm)$.

ולבסוף נשים לב כי $log500 \approx 9, log1000 \approx 10, log1500 \approx 10.5$ ובנוסף מתקיים לכל m כי $potential \approx \frac{1}{2} log\frac{m}{2}$ לכן הפוטנציאל מקיים $O(logm)$.

כלומר תוצאות המדידות עמדו בתוצאה שלנו.