שמות	ת"ז	שם משתמש
מוחמד דגש	314811290	daghash
נור נסר	318676418	noornasser

מבני נתונים – תרגיל מעשי 2 – תיעוד לכל פונקציה בקובץ השלד FibonacciHeap

המחלקה HeapNode:

כל עצם במחלקה הזו מהווה צומת אפשרית בערימת פיבונאצ'י, לכל עצם שייך למחלקה הזו ישנם 7 שדות והם:

Key: המפתח של הצומת, המפתחות הינם ייחודיים כלומר לכל צומת יש מפתח ששונה מהצמתים האחרים בעץ.

Degree: של הצומת מהווה את מספר הבנים של הצומת. (הדרגה)

Mark: של הצומת הוא יכול להיות 0 או 1 אשר מהווה את מספר הבנים שהיו לצומת וכרגע הם אינם, ואם צומת הוא שורש אז ה markשלו הוא תמיד 0.

Next: מהווה את האח הבא של הצומת הנוכחי ואם אין אז הוא מהווה את הצומת עצמו.

Prev: מהווה את האח הקודם של הצומת הנוכחי ואם אין אז הוא מהווה את הצומת עצמו.

Child: מהווה את הבן הראשון של הצומת הנוכחי.

Parent: מהווה את האבא של הצומת הנוכחי.

Helper: משתמשים בו רק בפונקציית kMin ,עוזר לנו לשמור האיברים הרלוונטיים.

המתודות של HeapNode:

getKey(): מחזירה את הערך של השדה key סיבוכיות זמן O(1)

Get_degree(): מחזירה את הערך של השדה degree סיבוכיות זמןO(1)

get_mark():מחזירה את הערך של השדה mark סיבוכיות זמן O(1)

get_next():מחזירה את הערך של השדה next סיבוכיות זמןO(1)

get_prev():מחזירה את הערך של השדה prev סיבוכיות זמן O(1)

get_child():מחזירה את הערך של השדה child סיבוכיות זמן O(1)

get_parent():מחזירה את הערך של השדה parent סיבוכיות זמן O(1)

set_key():הפונקציה מעדכנת את השדה key של הצומת הנוכחי, הסיבוכיות הינה O(1)

set_degree():הפונקציה מעדכנת את השדה rank של הצומת הנוכחי, הסיבוכיות הינה O(1)

set_mark():הפונקציה מעדכנת את השדה mark של הצומת הנוכחי הסיבוכיות הינה O(1)

set_next():הפונקציה מעדכנת את השדה next של הצומת הנוכחי, הסיבוכיות הינה O(1)

set_prev():הפונקציה מעדכנת את השדה prev של הצומת הנוכחי, הסיבוכיות הינה O(1)

set_child():הפונקציה מעדכנת את השדה child של הצומת הנוכחי, ומעדכנת את השדה Degree באופן המתאים, הסיבוכיות הינה O(1)

set_parent(): הפונקציה מעדכנת את השדה parent של הצומת הסיבוכיות הינה O(1).

המחלקה FibonacciHeap:

כל עצם במחלקה הזו מהווה ערימת פיבונאצ'י למחלקה הזו יש 5 שדות:

min: השדה הזה הוא מסוג HeapNode שמצביע על הצומת עם ה keyהכי קטן בערימה.

size: בשדה הזה שמור כמה צמתים יש בערימה.

trees: בשדה הזה שמור כמה עצים יש בערימה.

links: בשדה הזה שמור את מספר כל פעולות ה link שבוצעו מתחילת ריצת התוכנית.

cuts: בשדה הזה שמור את מספר כל פעולות ה cut שבוצעו מתחילת ריצת התוכנית.

marked: מספר הצמתים שמסומנים ב- mark מעודכן בכל חיתוך.

מתודות של המחלקה FibonacciHeap:

פעולה	סוג	סיבוכיות	תיאור
isEmpty()	boolean	O(1)	בודקת אם המינימום הינו null. אחרת הערימה לא תהיה ריקה.
Insert(int key)	HeapNode	O(1)	אם הערימה הינה ריקה, תכניס הצמת ותסמנה כמינימום. אחרת נכניס את הצמת לשמאל האיבר המינימלי, תעדכן את המינימום אם יש צורך.
Consolidate()	void	O(Maxrank+#trees)	הפונקציה הזו מייצרת מערך נקרא לו A בגודל log(size) + 1 אז עוברת על העצים בערימה ובכל פעם מכניסה את העץ לאינדקס i במערך כאשר i הוא הדרגה של שורש העץ אם המקום הזה היה פנוי, אחרת היא מבצעת פעולת linkבין העץ שרוצים להכניס אותו לבין העץ באינדקס i במערך ומעדכנים את A[i] להיות null ומכניסים את העץ החדש ל- A[i+1] ואם היה המקום הזה היה תפוס עוברים על אותם סדר בדיקות עד שנעבור על כל העצים, אחר כך מעדכנים את המצביעים ה next-ו prev-בין העצים במערך, סיבוכיות זמן הריצה Amortizedהוא כגודל המערך שזה O(log(size)).
deleteMin()	void	O(log(n))	מעדכנת מספר העצים וגודל הערימה (כלומר מספר הצמתים). ומבחינה בין 4 מקרים: 1) כאשר למינימום יש גם בנים וגם

אחים, אז נדאג לעדכן המצביעים המתאימים, כך שהבנים והאחים שלו כולם אחים, והצומת המינימאלי לו נמצא בינהם. אחר כך נחפש בין האחים העדכניים את המינימיום המתאים לערימה נוכחית. זה עולה במצב הכי גרוע O(n) כי ככל היותר יש n עצים, ועלות amortized שווה ל O(log(n)), -כי אחרי התיקון הראשון לערימה, מספר העצים יהיה חסום ל log(n) - עצים. 2) כאשר יש למינימום אחים אבל אין לו בנים. מעדכנים המצביעים של האחים, כך שאי אפשר להגיע למינימום עכשיו. ואז מחפש המינימום החדש בין האחים של המינמום הקודם, ומעדכנים אותו. 3) כאשר יש לו בנים אבל אין לו אחים, אז מעדכנים אחד הבנים להיות המינימום החדש, ומעדכנים השדהparent של כל הבנים להיות.null 4) כאשר אין לו בנים ואין לו אחים, אז			

			המינימום של נערימה הוא null עכשיו. עכשיו נקרא לפונק ' consolidate() כדי לעדכן השורשים ולקבל ערימה בינומית. סיבוכיות הפונקציה: במקרה הכי גרוע O(n) כי עוברים על כל העצים, ובפעם הראשונה מספר העצים שווה למספר הצמתים. אבל, אחרי התיקון הראשון של החיבורים, הקבל ערימה מאוד קרובה לערימה בינומית, ולכן עלות amortized היא O(log(n)).
findMin()	HeapNode	O(1)	מחזירה את הצומת של הערימה שהמפתח שלה מינימלי, או ריק אם הערימה ריקה.
meld(FibonacciHeap heap2)	void	O(1)	אם שתי הערימות אינן ריקות, נמזג אותן בערימה שלנו, שורשי הערימה החדשה נשארות שורשים גם לאחר ההתמזגות, לבסוף אנו מעדכנים את המינימום במידת הצורך.
Size()	int	O(1)	מחזירה את מספר הצמתים בערימה.
countersRep()	Int[]	O(#RootList)	אנו יוצרים מערך בגודל הסדר המרבי של עץ, הערך i-th מכיל את מספר העצים בסדר i בערימה, עבור כל צומת ברשימת השורשים של הערימה, אנו מציעים שהדרגה שלו היא d, נגדיל ב-1 את הערך באינדקס ה-d במערך, ואז נחזיר את המערך.
delete(HeapNode x)	void	O(log(n))	אנו מקטין את המפתח שברצוננו למחוק בערך

			המינימלי שיכול להיות ל- int, אז מחיקת אלמנט זה זהה. למחיקת האלמנט המינימלי של ה- heap, אז אנו קוראים לעזרה לפונקציה minDelete() .
decreaseKey(HeapNode x, int delta)	void	O(1)	אנו מקטינים את המפתח בערך של-delta, אם הוא מפר את תפקידי הערימה של פיבונאצ'י (מפתח הצומת קטן יותר ממפתח האב שלו), אנו קוראים לעזרה עבור הפונקציות cut() ו-cascading_cut(), ואז אנו מעדכנים את ה-min במידת הצורך.
Potential()	int	O(1)	פונקציה זו מחזירה את הפוטנציאל הנוכחי של הערימה, שהוא: פוטנציאל = #עצים + 2*#מסומנים
totalLinks()	Static int	O(1)	פונקציה סטטית זו מחזירה את המספר הכולל של פעולות קישור שנעשו במהלך זמן ריצה של התוכנית. פעולת קישור היא הפעולה שמקבלת כקלט שני עצים מאותה דרגה, ויוצר עץ בדרגה גדולה יותר באחד, על ידי תליית ה עץ שיש לו ערך גדול יותר בשורשו מתחת לעץ השני.
cascading_cut(HeapNode x)	void	O(log(n))	פונקצית עזר רקורסיבית שמנתקת תת-עץ מההורה שלו ומעדכנת את מספר המסומנים. אנחנו רוצים שמספר הצמתים בעץ מסדר k יהיה אקספוננציאלי ב- k. בדרך זו, אנו יכולים להחזיק רק מבחינה לוגריתמית עצים רבים בערימה מאוחדת.
totalCuts()	int	O(1)	פונקציה סטטית זו מחזירה את המספר הכולל של פעולות חיתוך שנעשו במהלך זמן ריצה של התוכנית. פעולת חיתוך היא הפעולה שמנתקת תת-עץ מההורה שלו (במהלך שיטות decreaseKey/delete).
kMin(FibonacciHeap H, int k)	Int[]	O(k*deg(H))	פונקציה סטטית זו מחזירה את k האלמנטים הקטנים ביותר בערימת פיבונאצ'י המכילה עץ בודד. זה ידוע

			שהשורש הוא האלמנט הקטן ביותר בערימה, אז על מנת לקבל את רכיבי ה--k 1 הקטנים הבאים בערימה, אנו מכניסים את ילדיו לערימה החדשה, כי אחד מהם הוא האלמנט השני הקטן ביותר בערימה שלנו. אחרי זה אנחנו מוצאים את זה ומכניסים את ילדיו (אם יש) לערימה החדשה (המועמדים החדשים לאלמנט הקטן הבא בערימה). לאחר מכן אנו מוחקים את ה-min של הערימה החדשה כדי שנוכל למצוא את אלמנט ה-min הבא. אנחנו עושים את זה k-1 פעמים.

חלק ניסויי/תיאורטי

<u>שאלה 1:</u>

א. בתוכנית מתקיימות m פעולות insert: פעולת הכנסה לערימה עולה זמן אסימפטוטי O(1) , כי אנחנו מוסיפים עוד עץ ב-rank שווה ל-0. וזאת פעולת הוספת עוד איבר לשורשים ועדכון מצביעים, ולכן לא תלויה במספר הצמתים או העצים בערימה.
לכן סך הכל פעולות insert עולות: $m * O(1) = O(m)$
פעולת מחיקה אחת: בגלל שזאת המחיקה הראשונה, היא עולה O(m) , כי צריכים לחבר כל שני צמתים יחד, עד שנקבל עץ אחד מ-rank של log(m) .
פעולות decrease-key: יש log(m)+1 פעולות כאלה, בגלל הבחירה לאלה צמתים להפעיל הפחתה, בגלל שרק בחלק קטן מהפעולות אנחנו עושים decrease-key ל-"אחים",
אז מתקיים: $\log(m) \leq cuts \leq 2\log(m)$. ויכולים לראות את זה בטבלה, כאשר לכל m הנחה זו מתקיימת. פעולת cut עולה O(1) (היא רק עדכון מצביעים). לכן סך הכל פעולות decrease-key עולות: $2\log(m) * O(1) = O(\log(m))$
אז בסך הכל סיבוכיות התוכנית היא $O(m) + O(m) + O(\log(m)) = O(m + \log(m))$.

ב.

m	Run-Time(ms)	totalLinks	totalCuts	Potential
2^{10}	1.357	1023	16	19
2^{15}	6.4844	32767	26	28
2^{20}	39.7751	1048575	36	39
2^{25}	8573.231	33554431	46	49

ג. פעולות link – ישנם O(m) פעולות Link מאחר ואנו קוראים לפעולת מחיקת הקטן ביותר כאשר הערימה יש בה m צמתים ב- rank=0, לכן אחרי המחיקה הזו, צריכים לחבר כל הצמתים ככל האפשר. ובגלל שכל העצים מאותו rank אז לחצי מהצמתים אנחנו לא עושים חיבור ולרבע מהם אנחנו עושים חיבור אחד וכו'... אז בסה"כ פעולות Link :

$$\sum_{i=1}^{\log(m)} \left(\frac{m}{2^i} * (i-1)\right) = 0 + \frac{m}{2} * 1 + \frac{m}{4} * 2 + \frac{m}{8} * 3 + \cdots + \frac{m}{m} * \log(m) = m$$

אז סה"כ יש O(m) פעולות Link. מה שמתאים לתוצאות שקיבלנו בסעיף ב' בטבלה.

פעולות cut – לפי ההסבר בסעיף א' קיבלנו כי $\log(m) \le cuts \le 2\log(m)$. אז לכל m מספר החיתוכים בוודאי קטן מ- 2*log(m).
ב- potential חסום מלמעלה כל ידי O(log(m)).

ד. המפתחות $m-2^i+1$ הם העצים מדרגה 0 בכל תת עץ מדרגה 1 עד log(m) בעץ הבינארי. לכן המפתחות $m-2^i$ הם השורשים של תתי העצים מדרגה 1 עד log(m) בעץ הבינארי. הפעולה תתבצע קודם כל על השורש של העץ ולאחר מכן נבצע DecreaseKey על הילדים של השורש ומכיוון שאנו מקטינים את הערך של השורש והילדים שלו באותה מידה אז הם יישארו גדולים מהשורש ולכן העץ לא ייחתך כלומר נדרש 0 פעולות חיתוך והפוטנציאל יהיה 1.

ה. בסעיף זה עדיין נבצע m+1 הכנסות אבל לא נמחק את המינימום (ללא ביצוע DeleteMin) ולכן נקבל ערימה שמורכבת מ-m+1 עצים מדרגה 0 ולא יהיו חיבורים בכלל. כשנבצע DecreaseKey על כל הצמתים שהם בעצם שורשים אז מבנה הערימה לא ישתנה והצומת לא יסומן אז לא יהיו בכלל פעולות חיתוך ולכן הפוטנציאל יהיה 1.

ו. העלות של פונ' DecreaseKey תלויה במספר החיתוכים, לכן עלות הפונקציה גדלה ככל שמספר החיתוכים עולה. מספר החיתוכים המקסימלי הינו log(m) כגובה העץ המקסימלי.

Case	totalLinks	totalCuts	Potential	DecreaseKey Max Cost
Original	m-1 = O(m)	Log(m)	3log(m)-1=O(log(m))	-
decKey($m-2^i$)	m-1 = O(m)	0	1	-
Remove line #2	0	0	1	-
Added line #4	m-1=O(m)	2*log(m)-1	2*log(m)	Logm-1=O(log(m))

שאלה 2:

א.

m	Run-Time(ms)	totalLinks	totalCuts	Potential
$3^6 - 1$	4	720	0	6
$3^8 - 1$	16	6550	0	6
$3^{10} - 1$	72	59037	0	9
$3^{12} - 1$	409	531427	0	10
$3^{14} - 1$	2493	4782952	0	14

ב. בסדרת פעולות אלו אנחנו מבצעים m פעולות insert שרצה בסיבוכיות זמן O(1) ולכן זמן ההכנסה עולה O(m). ואז אנו מבצעים $\frac{3m}{4}$ פעולות של DeleteMin בסיבוכיות O(log(m)) כל אחת ולכן זמן המחיקה עולה $\frac{3m}{4} * \log(m)$ אז סה"כ זמן ריצה אסימפטוטי הוא:

$$O(m) + O\left(\frac{3m}{4} * \log(m)\right) = O(m) + O\big(m * log(m)\big) = O(m * \log(m))$$

ג. פעולות cut – מספר ה-cuts שבוצעו תמיד 0 כי לא משתמשים ב-DecreaseKey.
פעולות Link – מספר פעולות חיבור הוא בערך O(mlog(m))
פוטנציאל – חסום מלמעלה על ידי O(log(m))