<u>תיאור המחלקה</u>

<u>המחלקה FibonacciHeap מכילה את השדות הבאים :</u>

- min – מצביע לצומת עם המפתח המינימלי בערמה
- Size – מספר הצמתים בערמה
- numOfTrees – מספר העצים בערמה
- numOfMarked – מספר הצמתים המסומנים בערמה
- leftmost – מצביע לעץ השמאלי ביותר בערמה
- totalLinks – שדה סטטי שסוכם את מספר הlink שבוצעו מתחילת ריצת התוכנית
- totalCuts – שדה סטטי שסוכם את מספר הcut שבוצעו מתחילת ריצת התוכנית

<u>למחלקה FibonacciHeap יש מחלקה פנימית HeapNode לה השדות הבאים :</u>

- Key – מספר שמייצג את המפתח של הצומת
- Rank – דרגת הצומת, שווה גם למספר הילדים של הצומת
- Mark – משתנה בוליאני שמייצג האם הצומת מסומן או לא, כלומר האם הצומת איבד אחד מילדיו
- Child – מצביע לילד השמאלי ביותר של הצומת
- Right – מצביע לאח הימני של הצומת
- Left – מצביע לאח השמאלי של הצומת
- Parent – מצביע להורה של הצומת
- copyOfMe – משמש את הפונקציה Kmin במחלקה FibonacciHeap, מצביע לצומת עם אותו מפתח בערמה אחרת

המחלקה HeapNode תומכת בפעולות constructor, getKey.

פונקציות המחלקה FibonacciHeap

נסמן n – מספר הצמתים בערמה.

Constructor

הבנאי לא מקבל ארגומנטים, הוא אחראי לאתחל את שדות המחלקה לערכם הdefault.

לוקח O(1) זמן.

isEmpty

הפונקציה בודקת אם גודל העץ הוא 0, ולכן פועלת בסיבוכיות O(1) זמן.

Insert

בערמות פיבונאצ'י הוספת צומת שקולה לשרשור צומת בודד משמאל.

לכן הפונקציה משנה מספר קבוע של שדות ופועלת בO(1) זמן.

deleteMin

הפונקציה מבצעת את הפעולות הבאות:

1. קוראת לdeleteForReal שפועלת בO(logn).
2. מאתחלת מערך בגודל logn בO(1).
3. הפונקציה נכנסת ללולאה שפועלת מספר פעמים השווה למספר העצים בערמה ובכל איטרציה מבצעת קריאה לפונקציה toBucket, שפועלת במקרה הגרוע בO(log(n)).
4. הפונקציה מבצעת קריאה לfromBucket שפועלת בO(log(n)).

כפי שהוכחנו בכיתה, הamortized time של delete min הוא O(log(n)).

<u>הפונקציות להן קוראת deleteMin:</u>

deleteForReal

הפונקציה משנה מספר קבוע של מצביעים בO(1).

בנוסף, הפונקציה משנה עבור כל ילדי צומת מסוים את מצביע הparent שלהם.

כלומר, זמן הריצה של הפונקציה הוא O(logn), כאשר n הוא מספר הצמתים בעץ.

toBucket

הפונקציה מבצעת לולאה שרצה לכל היותר כאורך המערך – log(n) איטרציות. בכל איטרציה היא מבצעת link בין שני עצים באותה דרגה. הפונקציה link פועלת בO(1) זמן, ולכן הפונקציה toBucket פועלת בO(log(n)) זמן.

Link

הפונקציה מחברת בין שני עצים עם אותה הדרגה, משנה מספר קבוע של מצביעים ולכן פועלת בO(1) זמן.

fromBucket

הפונקציה עוברת על מערך באורך O(log(n)), ומבצעת שינוי במספר קבוע של מצביעים עבור כל תא לא ריק במערך. הפונקציה פועלת בO(log(n)) זמן.

findMin

בשדה min יש מצביע לצומת המינימלי בעץ, ולכן נחזיר אותו בO(1) זמן.

Meld

בערמות פיבונאצ'י meld של שתי ערמות שקול לשרשור מימין. לכן הפונקציה מעדכנת מספר קבוע של שדות ופועלת בO(1) זמן.

size

בשדה size שמור גודל העץ ולכן נחזיר אותו בO(1) זמן.

countersRep

בתחילה הפונקציה מאתחלת מערך מונים בגודל O(log(n)).

הפונקציה עוברת על השורשים של כל העצים השמורים בערמה, ומעדכנת את מערך המונים בהתאם לדרגת כל שורש.

במקרה הגרוע הערמה מורכבת מn עצים מדרגה 0, ולכן במקרה הגרוע הפונקציה תרוץ בO(n) זמן.

delete

הפונקציה מבצעת חישוב, וקריאה לפונקציות decreaseKey, deleteMin.

הפונקציה decreaseKey פועלת בO(1) זמן בamortized, הפונקציה deleteMin פועלת בO(log(n)) זמן בamortize ולכן הפונקציה delete תפעל בO(log(n)) זמן.

Decrease key

הפונקציה מקטינה את ערך המפתח של צומת ומבצעת לכל היותר קריאה אחת לcascadingCut.

כפי שהוכחנו בכיתה, פונקציית cascadingCut פועלת בO(1) זמן בamortized ולכן decrease key גם היא תפעל בO(1) זמן בamortized.

cascadingCut

הפונקציה עוברת על מסלול אחד מצומת מסוים לשורש של העץ שהוא נמצא בו ומבצעת את פעולת הcut לכל צומת מסומן במסלול זה, עד אשר מגיעה לצומת לא מסומן.

כלומר הפונקציה cascadingCut ב O(num of cuts) זמן, כאשר כל חיתוך עולה O(1) זמן כמפורט בפונקציה cut.

נוכיח שבamortized time מספר החיתוכים הוא O(1) זמן –

לשם ההוכחה נגדיר -

Φ = num of trees + 2*num of marked nodes

c = num of cuts in one call to cascadingCut

$$\text{Amort(cascadingCut)} = \text{actual time} + \text{change in potential} = \underbrace{c}_{\text{Actual time}} + \underbrace{c}_{\text{change in num of trees}} + \underbrace{2(1-(c-1))}_{\text{change in num of marked nodes}} = 4 = O(1)$$

ולכן הפונקציה cascadingCut פועלת בO(1) זמן בamortized.

Cut

הפונקציה מבצעת חיתוך של צומת מהעץ בו הוא נמצא. לשם כך היא משנה מספר קבוע של מצביעים ולכן פועלת בO(1) זמן.

Potential

לערמה יש שדות עם מספר העצים ומספר הצמתים המסומנים. הפונקציה סוכמת אותם ולכן לוקחת O(1) זמן.

totalLinks

למחלקה שדה סטטי המחזיק את מספר הlink שבוצעו, ולכן הפונקציה מחזירה את הערך השמור בשדה בO(1) זמן.

totalCuts

למחלקה שדה סטטי המחזיק את מספר הcuts שבוצעו, ולכן הפונקציה מחזירה את הערך השמור בשדה בO(1) זמן.

kMin

1. הפונקציה יוצרת מערך בגודל K ומכניסה אליו את השורש, שהוא האיבר המינימלי בעץ.
2. הפונקציה מאתחלת ערימה חדשה, HeapOfSmallest, בO(1).
3. לאחר מכן, הפונקציה קוראת לfindMinimalChild על השורש, שפועלת בO(numOfChild)=O(degH), ומכניסה את הצומת שחזר מהפונקציה לHeapOfSmallest בO(1).

4. הפונקציה נכנסת ללולאה שרצה k-1 פעמים ומבצעת את הפעולות הבאות על HeapOfSmallest:
   - מציאת המינימום בO(1) זמן.
   - הכנסת מפתח המינימום שמצאנו למערך באינדקס המתאים בO(1) זמן.
   - קריאה לfindMinimalChild שפועלת בO(numOfChild), שחסום ע"י O(degH).
   - קריאה לfindMinimalBrother שפועלת בO(numOfBrothers), שחסום ע"י O(degH).
   - הכנסת לכל היותר שני צמתים לO(1).
   - מחיקת המינימום. נשים לב כי בערימה לכל היותר K איברים בכל רגע נתון, שכן בכל איטרציה היא גדלה לכל היותר באיבר אחד. כלומר מחיקת המינימום חסומה בזמן אמורטייזד ע"י O(logk).

   בסה"כ הלולאה פועלת באמורטייזד בזמן k*O(degH + logk)=O(k(logk + degH))

ולכן הפונקציה פועלת באמורטייזד בO(1)+O(degH)+O(k(logk + degH)) = O(k(logk + degH)) כנדרש.

findMinimalChild

הפונקציה עוברת על כל ילדי צומת מסוים ומחזירה את הצומת עם המפתח המינימלי ביניהם. לכן פועלת בO(numOfChild).

findMinimalBrother

הפונקציה עוברת על כל אחיו של צומת מסוים ומחזירה את הצומת עם המפתח המינימלי מבין הצמתים עם מפתח שגדול ממפתח הצומת, לכן פועלת בO(numOfBrother).

<u>מדידות</u>

<u>ניסוי מספר 1 –</u>

m	Run-Time (in miliseconds)	totalLinks	totalCuts	Potential
1024	0.1941	1023	18	19
2048	1.1624	2047	20	21
4096	2.3664	4095	22	23

א. ניתן לראות שככל שm גדל, כך גם זמן הריצה גדל.

בתחילה נבצע הכנסה של m איברים כל הכנסה דורשת O(1) זמן ובסה"כ ידרוש O(m) זמן.

לאחר מכן נבצע את פעולת הdelete min שתדרוש O(m) זמן בעקבות פעולות הlink.

מספר הlink שווה לO(m) שכן מספר הlink שווה לm כפול סכום הסדרה $\sum_{i=1}^{\log(m)} \frac{1}{2^i}$ – כל שני צמתים בודדים מתאחדים לעץ מדרגה 1, כל 2 עצים מדרגה 1 הפוכים לעץ מדרגה 2 וכו'.

סדרה זו מתכנסת ל1, ולכן מספר הlink שווה לO(m).

לאחר מכן נבצע את פעולות הdecrease key. כל פעולה כזו תדרוש O(1) זמן, ובסה"כ מתרחשים O(log(m)) cuts. פעולת הdecrease key תלויה במספר הcut המתבצעים – נסביר כי מספר הcut הוא O(log(m)). זאת מכיוון שביצענו decrease key כל פעם לילד של צומת שעוד לא חתכנו לו ילד. לכן לא נוצר במהלך התהליך מפל חיתוכים.

כלומר, מספר הcut הוא כגובה העץ – log(m). כל חיתוך דורש O(1) זמן, ולכן הפעולה דורשת O(log(m)) זמן.

החיתוך האחרון ידרוש גם הוא O(log(m)) זמן שכן בפעולות decrease key הקודמות גרמנו לכל הצמתים במסלול השמאלי ביותר של העץ להיות marked. כשנבצע את פעולת הdecrease key האחרונה, יתבצע מפל חיתוכים שיתחיל בעלה ויימשך עד השורש, כלומר יתבצעו פעולות שדורשות O(1) זמן כמספר הפעמים ששקול לגובה העץ – סה"כ O(log(m)) זמן.

בסה"כ נקבל O(m+log(m))=O(m) זמן.

ב. כמתואר בסעיף א' מספר הlink שווה לO(m) שכן מספר הlink שווה לm כפול סכום הסדרה $\sum_{i=1}^{\log(m)} \frac{1}{2^i}$ – כל שני צמתים בודדים מתאחדים לעץ מדרגה 1, כל 2 עצים מדרגה 1 הפוכים לעץ מדרגה 2 וכו'.

סדרה זו מתכנסת ל1, ולכן מספר הlink שווה לO(m).

מספר הcut גם הוא O(log(m)) כמתואר בסעיף א'.

בסה"כ נקבל סיבוכיות O(m).

ג. נשים לב כי לאחר מחיקת המינימום, יש בערמה 2^m-1 צמתים. כלומר, קיים עץ מכל דרגה מ0 ועד log(m).

לכן, העץ הגבוה ביותר הוא מגובה O(log(m)). במקרה הגרוע ביותר כל הצמתים במסלול הארוך ביותר בו מסומנים, ופעולת decreaseKey היקרה ביותר תבצע cascadingCuts לאורך כל המסלול הנ"ל, כלומר יתבצעו O(log(m)) חיתוכים.

לכן פעולת decreaseKey היקרה ביותר תפעל בO(log(m)).

ניתן לראות כי הניתוח התיאורטי תואם את תוצאות המדידות.

<u>ניסוי מספר 2 –</u>

M	Run-Time (in miliseconds)	totalLinks	totalCuts	Potential
1000	2.708	1891	0	6
2000	3.309	3889	0	6
3000	3.6805	5772	0	7

א. זמן הריצה האסימפטוטי של סדרת פעולות זו מורכב מ3 מרכיבים :

- הכנסה של m איברים לערמה. הכנסה דורשת O(1) זמן, מכניסים m איברים ולכן ידרוש O(m) זמן.
- הdeleteMin הראשון ידרוש O(m) זמן מכיוון שיש m עצים בערמה.
- שאר פעולות הdeleteMin יפעלו כגובה העץ, כלומר O(log(m)) זמן. ישנן O(m) פעולות כאלו.

בסה"כ נקבל O(mlog(m)) זמן.

ב. מספר פעולות הcut – 0, כיוון שלא מבוצעת קריאה לפונקציה decreaseKey.

מספר פעולות הlink – O(mlogm), שכן עלות פעולת הdeleteMin מושפעת ממספר פעולות הlink.

בסה"כ נקבל שהפעולה דרשה O(mlogm) זמן.

ג. **פוטנציאל המבנה כפונקציה של m הוא O(log(m)).**

נשים לב שבניסוי זה לא ביצענו קריאה לפונקציה decreaseKey. לכן הפוטנציאל שווה למספר העצים בערמה, שכן מספר הצמתים המסומנים הוא 0.

בנוסף, גודלו של כל עץ בערמה הוא חזקה של 2 לכן מספר העצים שווה למספר הביטים הדלוקים בייצוג הבינארי של גודל הערמה, שחסום ע"י O(log(m)).

אכן, ניתן לראות כי בייצוג הבינארי של 500,1000,1500 (מספר הצמתים בסוף ריצת הניסוי עבור כל m) יש 6,6,7 ביטים דלוקים בהתאמה. כלומר, הניתוח התיאורטי תואם את המדידות.