Software Documentation

FIBONACCI - HEAP

המחלקה FibonacciHeap מממשת ערימת פיבונאצ'י. הערימה תומכת בפעולות הכנסה, מיזוג ומציאת האיבר המינימלי בערימה בסיבוכיות זמן ריצה wc: O(1).

הערימה תומכת במחיקת האיבר המינימלי בסיבוכיות זמן ריצה amortized: O(logn) כאשר n הוא מספר הצמתים בערימה.

בנוסף הפחתת ערך מפתח מבוצעת בסיבוכיות זמן ריצה amortized: O(1).

שדות המחלקה

min– מצביע לצומת המינימלי בערימה.

totalLinks– שדה סטטי המכיל את מספר הלינקים הכולל שבוצע מאז תחילת ריצת התוכנית.

totalCuts - שדה סטטי המכיל את מספר החיתוכים הכולל שבוצע מאז תחילת ריצת התוכנית.

size– מספר הצמתים בערימה.

numberOfTrees– מספר העצים בערימה.

numberOfMarked– מספר הצמתים המסומנים בערימה.

HEAP NODE

מחלקה שמייצגת צומת בערימה. כל צומת מכיל את השדות הבאים:

Key– מפתח הצומת

Rank- דרגת הצומת

Mark– סימון הצומת – מכיל 0 או 1.

Child– מצביע לילד השמאלי ביותר של הצומת

Next– מצביע לאיבר הבא ברשימת הצמתים (רשימת השורשים או רשימה של אחים).

Prev– מצביע לאיבר הקודם ברשימת הצמתים (רשימת השורשים או רשימה של אחים).

Parent– מצביע להורה של הצומת.

מתודות

```
Public boolean empty()
```

הפונקציה בודקת אם הערימה ריקה באופן הבא: אם המצביע לאיבר המינימלי מצביע לnull הרי שהערימה ריקה והפונקציה מחזירה true. אחרת הפונקציה תחזיר false.

סיבוכיות- הפונקציה מבצעת פעולה בודדת ולכן סיבוכיות זמן הריצה היא O(1).

```
Public HeapNode insert(int key)
```

תחילה ניצור צומת חדש עם המפתח key. הבנאי של המחלקה heapNode מאתחל את הצומת עם דרגה 0, שכן הוא ישורשר לרשימת הצמתים ולכן לאחר ההכנסה לא יהיו לו ילדים. בנוסף הבנאי מאתחל את כל מצביעי הצומת להיות null ואת ביט הסימון של הצומת להיות 0.

נחלק ל-2 מקרים:

1. הערימה ריקה: במקרה זה זהו הצומת היחידי בערימה, לכן נעדכן את מצביע המינימום להיות צומת זה. בנוסף נעדכן את מצביעי ה-prev וה- next של הצומת להצביע אל עצמו בכדי ליצור רשימה מקושרת דו כיוונית מעגלית.
2. אחרת, נכניס את הצומת אחרי האיבר המינימלי בעזרת מתודת העזר insertAfter.

נגדיל את מספר העצים במבנה ואת גודל המבנה.

לבסוף נבדוק האם האיבר החדש שהוכנס קטן מהמינימום של הערימה. אם כן נעדכן את מצביע המינימום להיות הצומת החדש.

מתודות עזר:insertAfter

סיבוכיות: לאורך כל הפעולה אנו מבצעים פעולות קבועות שאינן תלויות בגודל הערימה. בנוסף מתודה העזר insertAfter פועלת בסיבוכיות O(1) ולכן בסך הכל סיבוכיות המתודה היא גם כן O(1).

```
Public void deleteMin()
```

הפונקציה מוחקת את האיבר המינימלי מהערימה.

תחילה נעדכן את גודל הערימה. אם לאחר המחיקה הערימה ריקה נעדכן את המצביע של המינימום לnull ואת כמות העצים לאפס והפונקציה תסתיים.

אחרת, עלינו לשרשר את ילדיו של הצומת הנמחק (במידה וקיימים) אל רשימת השורשים.

נחלק ל-3 מקרים:

1. הצומת הנמחק הוא השורש היחידי (ובהכרח יש לו ילדים כי לא נכנסנו למקרה בו גודל הערימה הוא 1). נעבור על ילדיו של הצומת הנמחק ונכבה צמתים שמסומנים (שכן הם הופכים להיות שורשים). לאחר מכן נבצע succesiveLinking על ילדיו.

2. לצומת הנמחק אין ילדים, במקרה זה נמחק את הצומת מרשימת השורשים ונקשר את הרשימה בלעדיו ע"י עדכון מצביעים. לאחר מכן נבצע succesiveLinking על רשימת השורשים המעודכנת.

3. לצומת הנמחק יש ילדים, במקרה זה תחילה נכבה את הסימון של ילדיו במידה וקיימים צמתים מסומנים. לאחר מכן נשרשר את ילדיו אל רשימת השורשים ע"י שינויי מצביעיונבצע succesiveLinking על רשימת השורשים המעודכנת.

פונקציות עזר-succesiveLinking, checkForMarked

סיבוכיות-נבצע כמות קבועה של פעולות שאינה תלויה בכמות האיברים n (עדכון מצביעים ושדות). לאחר מכן נקרא לפונקציות העזרcheckForMarked ו-succesiveLinking (בהתאם למקרה הנתון). הפונקציות פועלות בסיבוכיות O(n)ו- O(logn) במקרה הגרוע ולכן הפונקציה תרוץ בסיבוכיות: O(n+c+logn)=O(n).

```
Private void checkForMarked(HeapNode node)
```

הפונקציה מקבלת מצביע לרשימת צמתים ועוברת על הרשימה. בכל פעם שמגיעים לצומת מסומן הפונקציה מכבה את הסימון ומורידה את מספר הצמתים המסומנים בערימה ב-1.

<u>סיבוכיות:</u> אנו קוראים לפונקציה זו רק מתוך הפונקציה deleteMin על רשימת ילדיו של הצומת הנמחק. דרגת כל צומת חסומה ע"י O(logn) ולכן סיבוכיות זמן הריצה של הפונקציה היא O(logn).

```
Private void succesiveLinking (HeapNode x, HeapNode itemToDelete)
```

הפונקציה מקבלת צומת x כאשר x שורש ברשימת השורשים ואת הצומת הנמחק itemToDelete. הפונקציה מאחדת עצים בעלי אותה דרגה לעץ אחד שדרגתו גדולה באחד מדרגתם המקורית.

הפונקציה עוברת על רשימת העצים החל מהשורש x אותו קיבלה, תוך כדי המעבר ננתק את המצביע להורה itemToDelete במידה וקיים. עבור כל שורש נשים את העץ שלו בתא המתאים ברשימת הדליים לפי דרגתו, אם התא ריק לא יבוצע דבר. אחרת, נבצע Link לשני העצים בתא ונחזור על התהליך עד שנגיע לתא ריק.נשים לב שלאחר התהליך יוותרו לכל היותר logn עצים. לבסוף ניצור את הערימה מחדש מרשימת הדליים בעזרת הפונקציה arrayToHeap.

<u>פונקציות עזר</u>link-, arrayToHeap.

<u>סיבוכיות-</u> במקרה הגרוע כל עץ בערימה הוא מדרגה אפס, כלומר כל האיברים בערימה הם שורשים. במקרה זה נצטרך לעבור על n איברים ולבצע לכל היותר n לינקים (כי כל link מוריד את כמות העצים באחד ולכן לא יתכן שנבצע יותר מ- n לינקים).כמו כן טיפול ב"דליים" ופונקציית העזר arrayToHeap פועלות בסיבוכיות O(logn), נבצע עדכוני מצביעים ולינקים בזמן קבוע ובסה"כ הפונקציה תפעל בסיבוכיות O(n).

O(n+clogn+c)=O(n)

```
Private void arrayToHeap(HeapNode[] buckets)
```

הפונקציה מקבלת רשימה ועוברת עליה תוך כדי בדיקה אם התא ריק, במידה ולא נשרשר את העץ בתא לרשימת השורשים בעזרת הפונקציה insertAfter. תוך כדי המעבר נמצא את השורש המינימלי ונעדכן את המצביע בהתאם. כמו כן נעדכן את כמות העצים.

<u>פונקציות עזר-</u>insertAfter.

<u>סיבוכיות-</u>insertAfter פועלת בסיבוכיות זמן O(1). כמו כן, אורך המערך שמקבלת הפונקציה הוא לכל היותר באורך O(logn) (כי בוצע succesiveLinking). בכל איטרציה מתבצעת כמות קבועה של פעולות ולכן סיבוכיות זמן הריצה במקרה הגרוע הוא בסה"כ O(logn).

```
Private void insertAfter(HeapNode currentNode, HeapNode afterNode)
```

הפונקציה מקבלת צומת בשם currentNode וצומת בשם afterNodeומכניסה את הצומת afterNode אחרי הצומת currentNode. הפונקציה מבצעת כמות פעולות קבועה של שינויי מצביעים ולכן פועלת <u>בסיבוכיות</u>O(1).

```
Private HeapNode link(HeapNode smallestNode, HeapNode nextNode)
```

פעולה זו מחברת 2 צמתים כך שהאיבר המינימלי הוא ההורה של האיבר השני. תחילה נבדוק מי משני הצמתים הוא המינימלי. אם smallestNode איננו המינימלי נחליף מצביעים כך ש-smallestNode יהיה הצומת המינימלי.

<u>נחלק ל2 מקרים:</u>

1. אם דרגת smallestNode היא אפס (נשים לב ש-2 צמתים נשלחים למתודה זו אמ"מ הדרגה שלהם שווה – לכן גם הדרגה של nextNode היא אפס): זהו לינק פשוט שמעדכן את מצביע ה-child של smallestNode ואת מצביע ה-parent של nextNode בהתאם.
2. אחרת, נשרשר את nextNode אל רשימת הבנים של smallestNode ונעדכן את מצביע ההורה של nextNode.

<u>מתודות עזר:</u>insertAfter

לבסוף נגדיל את דרגת הצומת smallestNode ב-1.

<u>סיבוכיות:</u> אנו מבצעים פעולות קבועות (עדכון מצביעים) ללא קשר לגודל הערימה. לכן סיבוכיות המתודה היא O(1).

```
Public HeapNode findMin()
```

המתודה מחזירה את הצומת שהמפתח שלו הוא המפתח המינימלי בערימה.

נשים לב שמתכונת סדר הערימה מתחייב שצומת זה הוא שורש.

<u>סיבוכיות</u>- O(1).

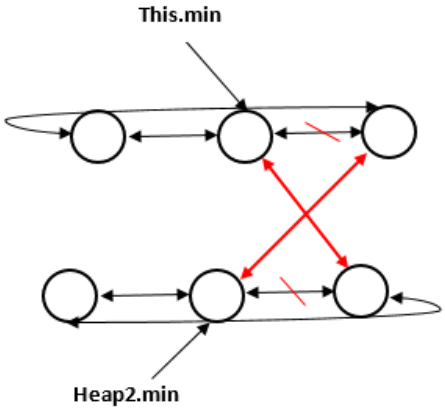
```
Public void meld (FibonacciHeap heap2)
```

במתודה זו אנו מחברים 2 ערימות.

<u>ישנם 3 מקרים אפשריים:</u>

1. אם heap2 היא ערימה ריקה אין צורך לעדכן דבר ולכן נחזור.
2. אם הערימה הנוכחית ריקה: נעדכן את מצביע המינימום להיות המינימום של heap2.
3. 2 הערימות לא ריקות. במצב זה עלינו לשרשר את רשימת השורשים של 2 הערימות.
בכדי להימנע ממעגל פנימי נבצע זאת בצורה הבאה:
נסמן את הצומת המינימלי של הערימה השניה ב- x.
נחבר את הצומת המינימלי של הערימה הנוכחית עם האיבר שנמצא אחרי x (נחבר על ידי עדכון המצביעים prev,next של 2 הצמתים האלו)
נחבר את הצומת שנמצא אחרי האיבר המינימלי בערימה אל x (על ידי עדכון המצביעים prev,next של 2 הצמתים האלו).
בכך נימנע ממעגל ונשרשר את 2 הרשימות יחדיו. שרשור זה מטפל במקרי קצה גם עבור רשימה בעלת איבר בודד.

<u>הרעיון:</u>



לבסוף נבדוק האם הצומת המינימלי ב-heap2 קטן מהצומת המינימלי בערימה הנוכחית. אם כן נעדכן את המצביע. בנוסף נעדכן את גודל הערימה, מספר העצים ומספר המסומנים בהתאם.

<u>סיבוכיות-</u> אנו מבצעים פעולות קבועות במתודה זו ללא קשר למספר האיברים הערימות. לכן סיבוכיות המתודה היא O(1).

```
Public int size()
```

המתודה מחזירה את השדה size של המחלקה. עדכון השדה מתבצע במתודות ההכנסה והמחיקה.

סיבוכיות: O(1).

```
Public int[] countersRep()
```

הפונקציה מחזירה מערך כאשר המיקום ה-iבמערך מכיל את כמות העצים בערימה שדרגתם i.

אם הערימה ריקה נחזיר מערך ריק. אחרת, נעבור על העצים החל מהעץ שהשורש שלו בעל המפתח המינימלי ועבור כל עץ דרגתו היא המיקום שניגש אליו ברשימה ונעלה את הערך במיקום זה באחד.

<u>סיבוכיות-</u> נעבור על כל העצים בערימה, כלומר במקרה הגרוע כל איבר הוא עץ (למשל ביצענו רק הכנסות) ולכן סיבוכיות זמן הריצה במקרה הגרוע היא O(n) כאשר n הוא כמות האיברים בערימה.

```
Public void delete(HeapNode x)
```

הפונקציה מקבלת צומת x, מבצעת decreaseKey לערך מינוס אינסוף ואז מבצעת deleteMin כאשר האיבר המינימלי יהיה x ולכן הוא ימחק.

<u>פונקציות עזר-</u>deceaseKey, deleteMin.

<u>סיבוכיות-</u> decreaseKey פועלת במקרה הגרוע בסיבוכיות O(logn) וdeleteMin בסיבוכיות O(n) במקרה הגרוע. לכן בסה"כ נקבל: O(logn+n)=O(n).

```
Public void decreaseKey(HeapNode x, int delta)
```

הפונקציה מקבלת צומת x ומפחיתה מהמפתח שלו את הdelta שקיבלה גם כן.

<u>נחלק למקרים:</u>

1. אם x הוא שורש או שהמפתח של x גדול מהמפתח של ההורה שלו (כלומר סדר הערימה נשמר) אין צורך לבצע cut. נבצע בדיקה האם הצומת בעל המפתח המינימלי השתנה, נעדכן במידת הצורך והפונקציה תסתיים. (הבדיקה של המינימום נעשית על שני המצבים אך השינוי אפשרי למעשה רק כאשר x שורש בגלל כלל סדר הערימה).

2. מגיעים למצב זה כאשר X לא שורש וגם סדר הערימה הופר. במקרה זה נחתוך את x מהעץ ע"י שימוש בפונקציית העזר cut. לאחר מכן נבצע cascading cuts ע"י לולאה שמבצעת חיתוכים כל עוד ההורה מסומן. ברגע שההורה לא מסומן נצא מהלולאה ואם מדובר בצומת שאינו שורש נשנה את הסימון ל-1. כמו כן, נעלה את כמות המסומנים ב-1. לבסוף נבדוק אם המפתח החדש של x קטן מהמפתח המינימלי ונעדכן את המצביע למינימום אם נדרש. (נציין כי במהלך שאר החיתוכים האפשריים המינימום לא יכול להשתנות בגלל סדר הערימה).

<u>פונקציות עזר-</u>cut

<u>סיבוכיות-</u>סיבוכיות זמן הריצה של פונקציית העזר היא O(1). במקרה הגרוע נבצע decreaseKey לצומת הנמוך ביותר בעץ ולולאת הcascading cuts תתבצע עד השורש. במקרה זה סיבוכיות זמן הריצה היא כגובה העץ. גובה העץ בערימת פיבונאצ'י לא חסום ולכן במקרה הגרוע נקבל חסם בסדר גודל של n. לכן סיבוכיות זמן ריצה(w.cO(n.

```
Private void cut(HeapNode x, HeapNode y)
```

הפונקציה מקבלת שני צמתים x ו-y ומבצעת חיתוך של x מההורה שלו y.

אם x היה מסומן מפחיתים אחד מכמות המסומנים ומשנים את סימונו לאפס (שורשים אינם מסומנים). מפחיתים את הדרגה של y ב-1. בודקים אם x ילד יחיד ומשנים מצביעים בהתאם. לבסוף נכניס את x לרשימת העצים בעזרת הפונקציה insertAfter ונעלה את כמות העצים ב-1.

<u>פונקציות עזר</u>insertAfter-

<u>סיבוכיות-</u> פונקציית העזר insertAfter בעלת סיבוכיות זמן ריצה O(1) וcut מבצעת כמות פעולות קבועה גם כן, לכן סיבוכיות זמן הריצה בסה"כ היא O(1).

```
Public int potential()
```

פונקצייה זו מבצעת כמות פעולות קבועה, היא מחזירה את כמות העצים בזמן הקריאה לפונקציה ועוד פעמיים כמות הצמתים המסומנים.

<u>סיבוכיות-</u> O(1).

```
Public static int totalLinks()
```

המתודה מחזירה את השדה הסטטי `totalLinks`.

סיבוכיות: O(1).

```
Public static int totalCuts()
```

המתודה מחזירה את השדה הסטטי `totalCuts`.

סיבוכיות: O(1).

מדידות

Sequence 1

m	Run-Time (in miliseconds)	totalLinks	TotalCuts	Potential

1000	3.569978	0	0	**1000**
2000	3.971601	0	0	**2000**
3000	4.306286	0	0	**3000**

מהו זמן הריצה האסימפטוטי של סדרת פעולה זו כפונקציה של m?

כל פעולת הכנסה מתבצעת בזמן קבוע במקרה הגרוע, כלומר O(1). אנו מבצעים m הכנסות ולכן סיבוכיות זמן הריצה האסימפטוטית של הסדרה היא: O(m).

כמה פעולות link וכמה פעולות cut מבוצעות, כפונקציה של m, במהלך סדרה זו של פעולות?

מבוצעות אפס פעולות מכיוון שאנו מבצעים הכנסות בלבד ופונקציית הinsert אינה מבצעת cut או link.

מה הפוטנציאל של המבנה כפונקציה של m בסוף ריצת סדרת פעולות זו?

הפוטנציאל של המבנה:

potential=#trees+2#marked

בסדרה של הכנסות בלבד מספר המסומנים הוא 0 ומספר העצים שווה למספר ההכנסות (כלומר m) ולכן הפוטנציאל של המבנה בסיום פעולות ההכנסה הוא O(m).

ניתן לראות כי התוצאות להן ציפינו אכן מופיעות בטבלה באותו האופן. כיוון שהכנסנו מעט איברים והזמנים יחסית קטנים זמן הריצה לא נראה לינארי. עבור כמות יותר גדולה של איברים זמן הריצה אכן לינארי.

Sequence 2

m	Run-Time (in miliseconds)	totalLinks	TotalCuts	Potential

1000	10.360969	2203	0	**6**
2000	10.421213	4395	0	**6**
3000	11.672490	6589	0	**7**

<u>מהו זמן הריצה האסימפטוטי של סדרת פעולה זו כפונקציה של m?</u>

עבור m הכנסות ביצענו O(m) פעולות. לאחר מכן ביצענו m/2 מחיקות ברצף. המחיקה הראשונה מתבצעת בסיבוכיות O(m). לאחר המחיקה הראשונה הערימה מכילה O(logm) עצים. במחיקה השניה אנו מוסיפים את הילדים של הצומת הנמחק לרשימת השורשים ועל רשימה זו מבצעים את פעולת ה- `succesiveLinking`. מכיוון שדרגת הצמתים חסומה ע"י O(logm) נקבל שרשימת השורשים לאחר הוספת ילדיו של הצומת הנמחק מכילה
O(logm) + O(logm) = O(logm) שורשים (מספר השורשים לפני הוספת הילדים + הוספת הילדים של הצומת הנמחק). לכן במקרה זה המחיקה תתבצע בסיבוכיות זמן ריצה O(logm).
עובדה זו נכונה עבור כל מחיקה שנבצע החל מהאיבר השני ועד m/2. לכן סיבוכיות זמן הריצה עבור המחיקות היא:

$$O\left(m+\sum_{i=2}^{\frac{m}{2}}\log(m)\right)=O\big(\mathrm{mlog(m)}\big)$$

בסך הכל עבור ההכנסות והמחיקות נקבל:

O(m) + O(mlogm) = O(mlogm)

<u>כמה פעולות link וכמה פעולות cut מבוצעות, כפונקציה של m, במהלך סדרה זו של פעולות?</u>

<u>מספר פעולות cuts:</u>
לא מבוצעות כלל פעולות cut כיוון שפונציית הinsert וגם הפונקציה deleteMin לא מבצעות כלל cuts.
<u>מספר פעולות לינק:</u>
מבוצעות m פעולות הכנסה אשר לא מבצעות link ולכן יש לנו m עצים עליהם נבצע deleteMin בפעם הראשונה. מספר הלינקים בכל פעולת מחיקה חסום ע"י מספר העצים (שכן כל פעולת לינק מורידה את מספר העצים ב-1). בהתאם להסבר בסעיף הקודם, במחיקה הראשונה יש לנו m עצים ולכן מספר הלינקים חסום ב-m. במחיקה השניה יש לנו O(logm) עצים ולכן מספר הלינקים חסום ב-O(logm). עבור סדרת המחיקות נקבל שמספר הלינקים חסום ע"י:

$$O(m)+O\left(\sum_{i=2}^{\frac{m}{2}}\log(m)\right)=O(m)+O\big(mlog(m)\big)=\ O\big(\mathrm{mlog(m)}\big)$$

נשים לב שבפועל מספר פעולות הלינק תלוי גם במימוש המחלקה. אם מתחזקים את הצומת המינימלי להיות גם בעל הדרגה המינימלית, נקבל שבסדרת פעולות המחיקה נבצע פחות לינקים מ-O(mlogm). זאת מכיוון שהצומת המינימלי בכל מחיקה יהיה גם בעל דרגה מינימלית, ולכן

כשנצרף את ילדיו לרשימת השורשים נבצע פחות לינקים מאשר אם נחבר O(logm) צמתים לרשימה לפי המימוש הכללי.

<u>מה הפוטנציאל של המבנה כפונקציה של m בסוף ריצת סדרת פעולות זו?</u>

הפוטנציאל של המבנה:

potential=#trees+2#marked

מספר המסומנים הוא אפס כי לא מבוצע decreaseKey לכן פונקציית הפוטנציאל היא מספר העצים בלבד לאחר ביצוע סדרת הפעולות. לאחר סדרת ההכנסות יש לנו m עצים ונעבור על כולם בdeleteMin הראשון. לאחר פעולת המחיקה בערימה שמכילה m צמתים יש לכל היותר log(m+1) עצים. לאחר המחיקה האחרונה יש בערימה m/2 איברים ולכן מספר העצים שנותרו הוא log(m/2+1) ובסה"כ O(logm).