

תיעוד הקוד

מחלקת HeapNode

מחלקה המייצג טיפוס מסוג צומת בערימה.

מכילה את השדות:

1. key – מספר שלם המייצג את מפתח הצומת.
2. rank – מספר שלם המייצג את מספר הבנים של הצומת.
3. mark – משתנה בוליאני שמייצג אם הצומת marked או לא.
4. child – מצביע מסוג HeapNode אל הבן (השמאלי) של הצומת.
5. next – מצביע מסוג HeapNode אל הצומת הבאה בערימה (אם מדובר בשורשים) או בעץ (אם מדובר באחים).
6. prev – מצביע מסוג HeapNode אל הצומת הקודמת בערימה (אם מדובר בשורשים) או בעץ (אם מדובר באחים).
7. parent – מצביע מסוג HeapNode אל האב.
8. pointer – מצביע מסוג HeapNode שמאותחל ל – null. בפונקציה kMin מקבל שימוש של להיות מצביע לאיבר המקורי בערימה.

פונקציות במחלקה:

1. **בנאים** – למחלקה יש שני בנאים.
 - I. `HeapNode(int key)` **public** – בונה צומת.
 - key- מכניס לשדה המפתח את מספר המפתח המתקבל כקלט לפונקציה זו.
 - מאתחל את שדות המצביעים next, prev להצביע אליו.
 - שאר השדות מאותחלים ל – null.
 - השדה mark מאותחל ל – false (כאשר צומת נוצר הוא לא מסומן).
 - II. `HeapNode(int key, HeapNode pointer)` **public**
מייצר צומת על ידי הבנאי הקודם, ומתאחל את שדה ה-pointer להיות הקלט של הבנאי.

2. **פונקציות get ו-set**

עבור כל שדה הוגדרו פונקציות get ו-set על מנת שנוכל לעדכן אותם ולקבל את ערכם במידת הצורך. פונקציות אלו פועלת ב- $O(1)$ משום שמדובר רק בעדכון תוכן של שדה או החזרה של ערך אליו יש לנו גישה ב- $O(1)$ (שדה).

3. `private HeapNode link(HeapNode curr)`

פונקציה לשימוש פנימי בתוך המחלקה – מופעלת כחלק מפונקציית Delete Min. הפונקציה הינה פונקציית מופע, כלומר פועלת על אובייקט מסוג HeapNode (לכן מופיעה תחת מחלקה זו).
בהינתן שני צמתים בעלי אותו rank, הפונקציה מחברת בין שני הצמתים לכדי עץ משותף בדרגה $rank + 1$.
הפונקציה מבצעת בדיקה : איזו צומת צריכה "להיתלות" על איזו, ומי השורש של העץ המשותף. בהתאם לכך מתבצעת החלפת מצביעים, בין היתר דרך קריאה לפונקציה connectNewChild שסיבוכיותה היא $O(1)$ כפי שיפורט בהמשך (זו פונקציה בתוך המחלקה FibonacciHeap לכן הסבר עליה יהיה בתיעוד של חלק זה).
הפונקציה מחזירה את העץ החדש שנוצר לאחר החיבור, על ידי החזרה של השורש. מאחר וכלל העבודה של הפונקציה מתבצעת על ידי החלפת מצביעים ב- $O(1)$, כלל הסיבוכיות של הפונקציה היא $O(1)$.

מחלקת FibonacciHeapמכילה את השדות:

1. first – מחזיק מצביע מסוג HeapNode לשורש שהראשון (השמאלי) בערימה.
2. min – מחזיק מצביע מסוג HeapNode לצומת בעל המפתח המינימלי בערימה.
3. n – מספר שלם המייצג את מספר הצמתים בערימה.
4. numRoots – מספר שלם המייצג את מספר השורשים בערימה.
5. numMarked – מספר שלם המייצג את מספר הצמתים בערימה שהינם מסומנים.
6. numLinks – משתנה סטטי שצובר את מספר link שבוצעו בתכנית.
7. numCuts – משתנה סטטי שצובר את מספר cutn שבוצעו בתכנית.

פונקציות המחלקה:**public** FibonacciHeap()

בנאי המחלקה. מאתחל את כלל השדות מסוג HeapNode להיות null, את המשתנים מסוג int להיות 0, והמשתנים הסטטיים אינם מאותחלים.

Public boolean isEmpty()

מחזירה אמת אם הערימה ריקה, שקר אחרת.
הפונקציה בודקת אם מספר הצמתים בערימה שווה ל-0 ולפיכך מחזיר את התוצאה. בדיקה שנערכת ב- $O(1)$ וזו גם סיבוכיות הפונקציה.

public HeapNode insert(**int** key)

הפונקציה מייצרת צומת חדשה עם מפתח key ומכניסה אותו לערימה. הפונקציה מחזירה את הצומת החדשה שהוכנסה.
מכיוון שערימת פיבונאצ'י זו ערימה עצלה, כל צומת חדשה שנכנסת הופכת לשורש נוסף בערימה. הפונקציה מעדכנת את שדה המינימום (במקרה הצורך), ומגדילה את מספר הצמתים בערימה. מתבצעת קריאה לפונקציה connectNewRoot שהסיבוכיות בה היא $O(1)$ כפי שיתואר בהמשך. לסיכום, בפונקציה מתבצעות החלפות מבציעים ולכן הסיבוכיות שלה $O(1)$ – דטרמיניסטי.

private void connectNewRoot(HeapNode newNode)

פונקציה פרטית לשימוש במחלקה.
הפונקציה מקבלת צומת ומצרפת אותה לרשימת השורשים בערימה. מכיוון שההכנסה היא לתחילת הרשימה, כל שורש נוסף כשורש הראשון בערימה, ושדה first מצביע אליו. הפונקציה מבצעת החלפת מצביעים ומגדילה את מספר השורשים בעץ. סיבוכיות: $O(1)$.

```
private void connectNewChild(HeapNode newChild, HeapNode oldChild)
```

פונקציה פרטית לשימוש במחלקה – כחלק מפונקציית link שתואר בהמשך.
הפונקציה מקבלת שני צמתים, והופכת אותם להיות אחים על ידי החלפת מצביעים.
סיבוכיות: $O(1)$.

```
public void deleteMin()
```

מוחקת את הצומת שהמפתח שלו מינימלי מבין המפתחות שבערימה, ולאחר מכן מבצעת link בין צמתים עם אותו ה-rank – כלומר מסדרת את הערימה להיות ערימה בינומית חוקית.
הפונקציה מבצעת ראשית מחיקה של הצומת וסידור מצביעים. לאחר מחיקת הצומת, הילדים של הצומת שנמחק הופכים להיות שורשים חדשים בערימה.
כלל הפעולות על המצביעים והשינויים מתבצעים ב- $O(1)$.
כעת הערימה אינה ערימה בינומית חוקית, שכן יש עצים מאותה דרגה, מכאן מתבצעת קריאה לפונקציה consolidate שמסדרת את העץ. כפי שיתואר בהמשך הפונקציה פועלת בסיבוכיות של $O(\log n)$. לסיכום, סיבוכיות הפעולה היא $O(\log n)$.

```
private void consolidate(HeapNode node)
```

הפונקציה מסדרת ערימה נתונה, לכדי ערימה בינומית חוקית. מתבצעת הכנסה של העצים בערימה לתוך קופסאות, link במקרה של עצים מאותה דרגה שנכנסו לאותה קופסה, ומייצרת ערימה חדשה על ידי איסוף כלל העצים שהתקבלו בקופסאות.
הפונקציה מייצרת קופסאות – $O(\log n)$ ובסופה מחברת בין השורשים של העצים בתוך הקופסאות $O(\log n)$.
תחילה, הפונקציה עוברת על כלל שורשי העצים בערימה (במקרה הגרוע n כאלה) ומכניסה אותם לקופסאות לפי הדרגה שלהם. בכל איטרציה כזו, מתבצעת בדיקה אם קיים כבר עץ בקופסה. אם כן, נבצע link בין שני העצים ב- $O(1)$ ונעביר אותם לקופסה הבאה וכן הלאה. בנוסף, מתבצע עדכון של שדה ה-min תוך כדי הריצה (מה שיעזור לנו להימנע מחיפוש min לאחר הפעולה – שיכול להיות יקר).
כפי שצוין קודם, לאחר שעברנו על כלל השורשים, נבצע איסוף חזרה לכדי ערימה בינומית חוקית ונחבר מצביעים בהתאמה.
כפי שראינו בכיתה, קיים מקרה גרוע בו הקונסולידציה מתבצעת על ערימה שבה מספר השורשים זהה למספר הצמתים (כל שורש הוא עץ) ומעבר על כלל השורשים והכנסה לקופסאות יהיה $O(n)$. מכיוון שזה מקרה יותר נדיר, כפי שראינו בכיתה ה-amortize cost של consolidate יהיה $O(\log n)$.

```
public HeapNode findMin()
```

הפונקציה מחזירה את הצומת בערימה שהמפתח שלה הוא מינימלי. מכיוון שמדובר בשדה של המחלקה, סיבוכיות $O(1)$.

```
public void meld (FibonacciHeap heap2)
```

הפונקציה מבצעת מיזוג בין שתי ערימות. מכיוון שערימת פיבונאצ'י היא ערימה עצלה, ה-meld מתבצע בצורה עצלה ובזמן קבוע. בהינתן שתי ערימות, נבצע חיבור בין השורשים הקיימים בערימות לכדי רשימת שורשים אחת. נעדכן את המינימום על ידי בדיקה מי המינימלי מבין שני הערכים בערימות. כלל הפעולות מתבצעות ב- $O(1)$ וזו גם סיבוכיות הפונקציה.

```
public int size()
```

הפונקציה מחזירה את מספר הצמתים בערימה. מכיוון שמדובר בשדה של המחלקה, הסיבוכיות היא $O(1)$.

```
public int[] countersRep()
```

הפונקציה מחזירה מערך מונים כך שבאינדקס i שמור כמה עצים יש בערימה שהסדר שלהם הוא i . כלומר, היא מחזירה מערך של integers, כך שלכל אינדקס i בין 0 עד הדרגה המקסימלית של עץ שקיימת בערימה, הערך שמוחזר במערך הוא מספר העצים שקיימים בערימה מסדר i . הפונקציה עוברת על כלל השורשים בערימה, ומעדכנת את המערך בהתאם ל-rank שלהם. סיבוכיות במקרה הגרוע (כשיש n שורשים) היא $O(n)$. כמו בפונקציות אחרות, amortized cost יהיה $O(\log n)$.

```
public void delete(HeapNode x)
```

הפונקציה מוחקת את האיבר x על ידי הפעלת הפונקציות decreaseKey (עם דלתא שבהכרח יצור ל- x מפתח שקטן מהמינימלי האפשרי על מנת שהוא יהפוך למינימלי בעץ) ואחריה DeleteMin. decreaseKey מתבצעת ב- $O(1)$ אמורטיזד, ו-DeleteMin ב- $O(\log n)$. לכן סה"כ סיבוכיות פעולה זו היא $O(\log n)$.

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

פונקציה זו מפחיתה δ מערכו של המפתח של האיבר x . אם לאחר ההפחתה x קטן מאביו, אז מתבצעת קריאה לפונקציה cut אשר חותכת את x מהעץ. אם לאחר חיתוך זה יש צורך בחיתוך חלק מאבותיו של x גם כן, אז מתבצעת קריאה לפונקציה cascadingCuts. מכיוון שפונקציה זו פועלת בפרויקט זה על ערימת פיבונאצ'י, סיבוכיותה היא $O(1)$ אמורטיזד.

```
private void cut(HeapNode x, HeapNode parent)
```

פונקציה זו חותכת את האיבר x מהעץ אליו השתייך. סיבוכיות פעולה זו היא $O(1)$ משום שהיא רק קוראת לפונקציות get ו-set מהמחלקה HeapNode אשר פועלות ב- $O(1)$, ומבצעת תנאים וחישובים ב- $O(1)$.

```
private void cascadingCuts(HeapNode y1, HeapNode y2)
```

במתודה זו חותכים את $y1$ מהעץ אליו השתייך ומכניסים אותו לערימה כעץ חדש ונפרד אשר $y1$ הוא שורשו.

אם לאחר חיתוך זה יש צורך בחיתוך כמה מאבותיו של $y1$, פונקציה זו נקראת בצורה רקורסיבית. בכל קריאה סיבוכיות המתודה היא $O(1)$, אך בשל פוטנציאל הקריאה הרקורסיבית, בסה"כ סיבוכיות פעולה זו היא $O(\log n)$ במקרה הגרוע בו חותכים איברים מכל שלב בעץ, אחד אחרי השני. משום שגובה עץ בערימת פיבונאצ'י חסום על ידי $\log n$, סיבוכיות המתודה תהיה $O(\log n)$.

```
public int potential()
```

הפונקציה מחזירה את ערך הפוטנציאל הנוכחי של הערימה. הפוטנציאל, כפי שהוגדר בשיעור, הינו $Potential = \#trees + 2 * \#marked$. מכיוון שהחישוב מתבצע על שדות שהגישה אליהם היא $O(1)$, סיבוכיות הפעולה היא $O(1)$.

```
public static int totalLinks()
```

פונקציה זו מחזירה את מספר הלינקים שבוצעו מתחילת העבודה על הערימה. סיבוכיותה $O(1)$ משום שהיא רק מחזירה את ערכו של המשתנה הסטטי `numLinks` של המחלקה `FibonacciHeap`.

```
public static int totalCuts()
```

פונקציה זו מחזירה את מספר החיתוכים שבוצעו מתחילת העבודה על הערימה. סיבוכיותה $O(1)$ משום שהיא רק מחזירה את ערכו של המשתנה הסטטי `numCuts` של המחלקה `FibonacciHeap`.

```
public static int[] kMin(FibonacciHeap H, int k)
```

פונקציה זו מחזירה מערך של k המפתחות של k האיברים המינימליים בערימה H . בשלב הראשון הפונקציה מכניסה לערימת עזר את האיבר המינימלי ב- H , ולאחר מכן מבצעת `deleteMin` ומוחקת אותו. בשלב הבא הפונקציה מכניסה לערימת העזר את כל בניו של האיבר המינימלי שנמחק באיטרציה הקודמת, וזאת משום שהאיבר המינימלי הבא בהכרח נמצא ביניהם (לפי תכונת עצים בינומיים). כעת נבצע `deleteMin` פעם נוספת ולאחר מכן באופן דומה לקודם, נכניס את כל בניו של האיבר שמחקנו באיטרציה הקודמת. כעת הצומת המינימלי יהיה בהכרח בין האחים של הצומת שנמחק או בין בניו. כך נמשיך עד שנבצע k פעמים `deleteMin` מערימת העזר. סה"כ סיבוכיות הפעולה תהיה:

הכנסות: בכל שלב מספר הבנים של האיבר שנמחק באיטרציה הקודמת חסום על ידי $\deg H$. הכנסה בערימת פיבונאצ'י היא מסיבוכיות של $O(1)$, מבצעים k הכנסות, ולכן סה"כ סיבוכיות כל ההכנסות לערימת העזר:

$$k \cdot \deg H \text{ insertions} \rightarrow O(k \cdot \deg H)$$

מחיקות: מספר האיברים בערימת העזר חסום על ידי $kdegH$ וזאת משום שבכל הכנסה הכנסנו אליה לכל היותר $degH$ איברים, וביצענו k הכנסות. כל מחיקה מערימת פיבונאצ'י היא מסיבוכיות $\log(\#num_nodes_in_heap)$ אמורטייזד, ולכן סה"כ סיבוכיות k המחיקות שנבצע חסומה על ידי:

$$k \text{ deletions} \rightarrow O(k \cdot \log(k \cdot degH))$$

על כן בסה"כ סיבוכיות כלל הפונקציה תהיה:

$$\begin{aligned} O(kdegH) + O(k \log(kdegH)) &= O(kdegH) + O(k(\log k + \log degH)) \\ &= O(kdegH) + O(k \log k + k \log degH) = \mathbf{O(k(\log k + degH))} \end{aligned}$$

```
public static void insertChildren(FibonacciHeap tmpHeap, HeapNode parent)
```

פונקציה זו מכניסה את כל ילדיו של `parent` ל-`tmpHeap`.

נעשה שימוש בפונקציה זו רק על ידי המתודה `kMin`, כאשר שם מספר הבנים של כל צומת הוא לכל היותר $degH$. על כן, סיבוכיות פונקציה זו היא $O(degH)$, כמספר המקסימלי של בנים שיכולים להיות ל-`parent` המתקבל כקלט. באופן כללי סיבוכיות פונקציה זו היא $O(\#parent's_children)$.