Fibonacci Heap Key Implementation

סטודנטים:
שלומי אפרגן – shlomia
גיא ביליצקי – guybilitski

המחלקה FibonacciHeap:
מממשת את מבנה ערימת פיבונאצ׳י.
תיעוד חברי המחלקה:
Min – מצביע אל האיבר המינימלי בערימה.
leftRoot – מצביע אל השורש השמאלי ביותר בערימה.
Size – מונה את מספר הצמתים בערימה.
numOfTrees – מונה את מספר העצים בערימה.
numOfMarks- מונה את מספר הצמתים המסומנים ב mark.
numOfLinks- משתנה סטטי המונה את מספר חיבורי העצים שבוצעו
numOfCuts – משתנה סטטי המונה את מספר פעולות ה cut שבוצעו.

תיעוד פונקציות המחלקה:

```
public boolean isEmpty()
```

מחזירה true אם ערך השמור בזיכרון הוא null, אחרת false. לכן $O(1)$

```
public HeapNode insert(int key)
```

יוצרת HeapNode חדש ואז קוראת לפוקנציה addHeapNode. הפונקציה addHeapNode (מופיעה בהמשך התיעוד) הינה בסיבוכיות $O(1)$ ולכן גם הפונקציה הנ"ל הינה $O(1)$

```
public void deleteMin()
```

פונקציה זו מוחקת את האיבר המינימלי ב $O(1)$. במידה ויש לצומת שנמחק בנים, נמחק את הקישור שלהם להורה שנמחק באמצעות הפעולה setParentAsNull - בסיבוכיות $O(logn)$ במקרה הגרוע שבו דרגת העץ הינה $logn$ (אינו משפיע אסימפטוטית בשל סיבוכיותה של הפעולה consolidate). כמו כן נוסיף אותם לרשימת שורשי הערימה. ההוספה לוקחת $O(1)$ בגלל שזוהי הוספת איברים לתחילת רשימה מקושרת דו כיוונית. כמו כן הפונקציה מבצעת consolidate. הפונקציה consolidate מופיעה בהמשך התיעוד וסיבוכיותה במקרה הגרוע $O(n)$ ובמקרה הממוצע$O(logn)$. ובשל כך זוהי גם הסיבוכיות של deleteMin.

```
public void consolidate()
```

הפונקציה יוצרת מערך basket שלתוכו נכנסים העצים שהדרגה שלהם זהה לזו של האינדקס במערך. בכך הפונקציה עוברת על שורשי הערימה, מוציאה אותם מה"רשימה המקושרת" של השורשים, ומכניסה אותם לbasket. במידה וכבר מופיע עץ במערך, הפונקציה תחבר בין העצים ב-$O(1)$ כיוון שהעץ שערך השורש שלו גדול יותר מתחבר לזה הקטן יותר בתור בן ימני. לאחר החיבור הפונקציה תרוץ על הbasket ותבצע חיבורים כאלו עד אשר תמצא מקום פנוי לשים את העץ שקיבלה. הראינו בכיתה באמצעות פונקציית פוטנציאל במספר העצים כי העלות הממוצעת לפעולה זו חסומה על ידי $2 * \log(n)$ ולכן בamortize עלות הפעולה הינה $O(logn)$. במקרה הגרוע, עלות הפעולה הינה $O(n)$.

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

הפונקציה מחברת בין שני זוגות איברים. בין האיבר הימני ביותר בערימה המקורית לזה השמאלי ביותר בערימה החדשה, וכן בין האיבר השמאלי ביותר בערימה המקורית, לימני ביותר בזו החדשה (להשלים את המעגליות של ה"רשימה המקושרת"). סה"כ מדובר במספר קבוע וידוע של פעולות ולכן עלות הפעולה הינה $O(1)$

```
public int size()
```

מחזירה את גודל הערימה ששמור לנו אליו מצביע. סיבוכיות $O(1)$.

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

ייתכן ויש לנו n שורשים בערימה שניאלץ לעבור על כולם ולכן הסיבוכיות הינה $O(n)$. בכל מקרה ניתן להראות רצף של פעולות (אשר לא כוללות deleteMin למשל) שבהן העלות של פונקציה זו תהיה O(n) בכל מקרה.
הפונקציה עוברת על כל שורשי העץ ומכניסה אותם לרשימה שהאינדקס של הרשימה הינו דרגת העץ.

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

הפונקציה מוחקת את הצומת x. ראשית אם המפתח של x אינו המינימלי בעץ – מתבצעת קריאה לפונקציה actualDecreseKey() (מפורט בהמשך) שתוריד את ערך המפתח להיות מינימלי.

כעת – יימחק על ידי פעולת DeleteMin().

פעולת ה actualDecreaseKey היא זולה יותר אסימפטוטית מפעולת ה deleteMin (במקרה הממוצע והגרוע).

מאחר והפעולות מתבצעות באופן טורי – מתקיים כי עלות הסיבוכיות היא כשל deleteMin()– שהיא $O(\mathrm{n})$ במקרה הגרוע, וסיבוכיות $O(\log_2 n)$ במקרה הממוצע.

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

הפעולה קוראת לactualDecreaseKey על בסיס סימון – אם רוצים שהאיבר ימחק אזי הפעולה תקרא עם ערך אמת, ואם לא אז שקר. בפעולה זו נקראת הפונקציה עם סימון שקר. סיבוכיותה היא כסיבוכיות הפעולה שלה היא קוראת (מפורט בהמשך) $(W.C: O(\log_2 n),\ \ Amortized: O(1))$

```
public int potential()
```

מחזירה את מספר העצים ועוד פעמיים מספר הסימונים. יש לנו מצביעים לשני הערכים האלו ולכן הפעולה תעלה לנו $O(1)$

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

מחזירה ערך שיש לנו מצביע אליו. סיבוכיות $O(1)$

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

מחזירה ערך שיש לנו מצביע אליו. סיבוכיות $O(1)$

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

הפעולה מחזירה מערך עם k האיברים הקטנים ביותר בערימה על ידי הפעולות הבאות:

1. אתחול של ערימת פיבונאצ'י חדשה.
2. האיבר המינימלי כעת הוא השורש – לכן נכניס אותו למערך שיוחזר.
3. האיבר המינימלי הבא יהיה אחד מן הילדים של השורש – לכן נכניס את כולם לערימה.

4. ניקח את המינימלי מבין כולם בערימת הפיבונאצ׳י החדשה למערך המוחזר, ונמחק אותו מהערימה – העץ יסתדר תמיד כך שהמינימלי יהיה בשורש.
5. עבור הצומת שמחקנו זה עתה – נכניס את ילדיו מהערימה המקורית (מהווים מועמדים פוטנציאלים להיות המינימום הבא).
   - לכל צומת המוכנס לערימה החדשה קיים מצביע לצומת המקורי בערימה H – כך ההגעה חזרה לילדיו מתבצעת ב O(1).
6. נחזור על הפעולה עד שנעתיק את k האיברים המינימליים ביותר בערימה.
   - עלות הפעולה היא כנדרש –$O(K * \deg(H))$ – מבצעים K פעמים הכנסה של המינימום לערימה החדשה ב (1)o, שליפה של המינימום מהערימה החדשה ב O(1), והכנסת הבנים הפוטנציאלים שהם לכל היותר deg(H) מאחר ומספר הילדים המקסימלי של השורש בעץ תקין הוא לכל היותר deg(H) ובכל ירידה ברמה יש מספר שווה / קטן של בנים בהכרח.

```
private void disconnectFromList(HeapNode node)
```

הפונקציה מקבלת צומת מהרשימה המקורית ודואגת לנתק אותו מהרשימה. סה"כ זוהי מחיקה מ"רשימה מקושרת" ועדכון של מספר מצומצם של מצביעים ולכן סיבוכיותה O(1).

```
private HeapNode[] minOfNodes(HeapNode node1, HeapNode node2)
```

הפונקציה מקבלת 2 צמתים ומחזירה מערך של 2 צמתים כאשר באינדקס 0 נמצא המינימלי מביניהם, ובאינדקס 1 נמצא המקסימלי מביניהם. פעולה קבועה ולכן בעלות O(1).

```
private void addHeapNode(HeapNode newNode)
```

הפונקציה מקבלת צומת חדש ודואגת להכניס אותו למקום השמאלי ביותר ב"רשימה המקושרת" של שורשי הערימה. סה"כ מדובר בהכנסה לרשימה מקושרת ומספר פעולות עדכון ולכן סיבוכיותה O(1).

```
private void concatenateRoots(HeapNode root1, HeapNode root2)
```

הפונקציה מקבלת שני צמתים מאותה דרגה כאשר root2 קטן יותר מroot1. הפונקציה דואגת שבסוף ריצתה root2 יהיה עץ שבנו השמאלי הוא root1. הפעולות שמבוצעות הן מצומצמות תוך חלוקה למקרים האם יש בנים ופעולות בהתאם לזה. סה"כ סיבוכיותה O(1).

```
private void actualDecreaseKey(HeapNode x, int delta, boolean toBeDeleted)
```

הפעולה מבצעת הורדת ערך של איבר כשם שלמדנו בכיתה. היא תפחית את ערך האיבר לערך הדרוש ותקרא לcascadingCut. במידה והערך אינו מיועד למחיקה, היא גם תבצע את העדכון למינימום החדש במידת הצורך. סיבוכיות הפעולה היא כשל סיבוכיות פעולה ה cascadingCut שמנותחת בהמשך.
$(W.C: O(\log_2 n),\ Amortized: O(1))$

```
private void cascadingCut(HeapNode x, HeapNode y)
```

הפעולה מבצעת את המחיקות על בסיס הסימונים כשם שלמדנו בכיתה. ייתכן והערך שנדרשנו להפחית מערכו יהיה בתחתית עץ בעל דרגה גבוהה מאד ולכן במקרה הגרוע סיבוכיותה תהיה $O(logn)$. במקרה הממוצע, הראינו בכיתה באמצעות פונקציית פוטנציאל על מספר העצים ועוד פעמיים מספר הצמתים המסומנים, כי הסיבוכיות המתקבלת הינה $O(1)$.

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

כפי שלמדנו בכיתה – חותכת את הצומת x מההורה שלו y ומעבירה אותו להיות שורש. פעולה קבועה שעולה O(1).

```
private void setParentsAsNull(HeapNode h)
```

הפונקציה מקבלת צומת ודואגת שההורה שלו ושל כל אחיו יהיה null. סיבוכיות הפעולה היא כמספר אחיו של הצומת. נשים לב כי במידה ומדובר בבנים של אחד השורשים בערימה, סיבוכיות הפעולה היא כדרגת העץ. לפיכך סיבוכיות הפעולה הינה O(k) כאשר k הוא מספר האחים של h.

המחלקה HeapNode:

חברי המחלקה:

Key – מפתח הצומת .

Size – סך מספר הצמתים בתתי העצים של הצומת הנוכחי (כולל הוא עצמו).

Rank – מספר הילדים של הצומת (ברמה אחת מתחת בלבד).

Mark – ערך בוליאני לאם הצומת מסומן או לא (לאחר חיתוך אחד מילדיו כנלמד בכיתה).

Child – מצביע לבן הישיר השמאלי ביותר של הצומת.

Prev – מצביע לצומת שמשמאלו של הצומת הנוכחי (בצורה מעגלית).

Next – מצביע לצומת שמימינו של הצומת הנוכחי (בצורה מעגלית).

Parent - מצביע להורה של הצומת, אם הוא שורש – יהיה null.

Pointer – משמש עבור פעולת ה kMin – כאשר מכניסים צומת לערימה החדשה – נשמור לו מצביע לצומת המקורי בערימה המקורית לגישה יעילה וזולה.

הפונקציה getKey – מחזירה את מפתח הצומת.

הפונקציה getChild – מחזירה את מה ש child מצביע אליו.

הפונקציה getNext – מחזירה את מה ש Next מצביע אליו.