**מבני נתונים – תרגיל מעשי 2**

**Fibonacci Heap**

**מגישים:** אופיר פפר – 203565833. שם משתמש במודל: ofirfeffer

אילור יפרח – 205828478. שם משתמש במודל: ilorifrach

**מחלקת HeapNode**

המחלקה מייצגת צומת בודד בעץ המקיים את חוקי הערימה.

המחלקה מממשת את הממשק Iterable ומאפשרת איטרציה על הבנים של הצומת.

**שדות המחלקה:**

Integer key - המפתח של הצומת. מספר שלם.

HeapNode parent - מצביע להורה של הצומת בעץ. Null במקרה שהצומת הוא השורש.

HeapNode right - מצביע לאח של הצומת מימין.

HeapNode left – מצביע לאח של הצומת משמאל.

HeapNode child – מצביע לזקיף של רשימת הבנים של הצומת.

boolean isMarked – משתנה בוליאני שערכו אמת אם אחד מבניו של הצומת נחתך.

int rank – דרגת הצומת.

**מתודות:**

1. **HeapNode(int key)** – בנאי המקבל מספר שלם כמפתח. מאתחל את המצביעים של האח הימני והשמאלי להצביע לצומת עצמו ומאתחל את child כזקיף.

**סיבוכיות**:O(1)

1. **HeapNode()** – בנאי ריק המאתחל צומת חדש עם מפתח null. משמש ליצירת זקיפים.

**סיבוכיות**: O(1)

1. **Integer getKey()** – מחזיר את המפתח של הצומת.

**סיבוכיות**: O(1)

1. **void appendSibling(HeapNode node)** – מוסיף צומת חדש לרשימת האחים של הצומת. בפרט, הצומת החדש יתווסף כאח של הצומת הנוכחי מימין. הרשימה היא רשימת מקושרת דו כיוונית עם זקיף לכן נדרשים 4 שינויי מצביעים בהתאם לאלגוריתם הנלמד בקורס.

**סיבוכיות**: O(1)

1. **void deleteSibling(HeapNode node)** - מוחק צומת מרשימת האחים. הרשימה היא רשימת מקושרת דו כיוונית עם זקיף ולכן הפעולה מתבצעת בעזרת שינוי 2 המצביעים של הצומת לאחים שלו בהתאם לאלגוריתם הנלמד בקורס. הנחת קדם: הצומת אכן קיים ברשימת האחים.

**סיבוכיות**: O(1)

1. **boolean isSentinel()** – פונקציה בוליאנית המחזירה אמת אם ורק אם הצומת הוא זקיף. הפונקציה בודקת אם מפתח הצומת הוא null, ואם כן אז הצומת הוא זקיף ולכן תחזיר אמת.

**סיבוכיות**: O(1)

1. **Iterator<HeapNode> iterator()** – פונקציה המחזירה Iterator עבור רשימת הבנים של הצומת. הפונקציה מאתחלת מופע חדש של המחלקה HeapNodeIterator ומחזירה אותו.

**סיבוכיות**: O(1)

**מחלקת HeapNodeIterator**

המחלקה מייצגת איטרטור הרץ על רשימה של Heap Nodes.

הנחת קדם: האיטרטור מתחיל את הריצה על רשימה כזו בזקיף – צומת שערך המפתח שלו הוא null. לכן יש לשלוח לבנאי המחלקה צומת שהוא זקיף.

**שדות המחלקה:**

HeapNode current – מצביע לצומת שיוחזר באיטרציה הבאה.

**מתודות:**

1. **HeapNodeIterator(HeapNode node)** – בנאי המקבל צומת ומאתחל את השדה current להיות אחיו הימני של הצומת. הנחת קדם: הצומת שנשלח לבנאי הוא זקיף.

**סיבוכיות**: O(1)

1. **boolean hasNext()** – פונקציה בוליאנית המחזירה אמת אם ורק אם נותרו צמתים ברשימה שעוד לא נעשתה עליהם איטרציה. הפונקציה בודקת אם המצביע לצומת בשדה current מצביע לצומת שהוא זקיף. אם current מצביע לזקיף הרי שסיימנו לעבור על כל הרשימה (כי הנחנו שהתחלנו בזקיף) ולכן תחזיר הפונקציה false. אחרת, תחזיר אמת.

**סיבוכיות**: O(1)

1. **HeapNode next()** – הפונקציה מחזירה את הצומת הבא ברשימה ומקדמת את current לאחיו הימני.

**סיבוכיות**: O(1)

**מחלקת FibonacciHeap**

מחלקה המייצגת ערימת פיבונאצ'י מעל המספרים השלמים האי-שליליים.

המחלקה מממשת את הממשק Iterable ומאפשרת איטרציה על שורשי העצים המוחזקים בערימה.

**שדות המחלקה:**

HeapNode sentinel – זקיף לרשימת שורשי העצים של הערימה.

HeapNode min – מצביע לשורש מתוך רשימת השורשים אשר מחזיק את המפתח המינימלי.

int size – גודל הערימה.

int potential – פוטנציאל הערימה אשר ערכו מספר העצים + 2\*מספר הצמתים המסומנים.

**מתודות מופע:**

1. **FibonacciHeap()** – בנאי המאתחל ערימה חדשה. מאתחל את הגודל והפוטנציאל לאפס. מאתחל את המצביע min לnull. מתאחל את sentinel להיות צומת חדש עם מפתח null אשר מצביע לעצמו מימין ומשמאל.

**סיבוכיות**: O(1)

1. **HeapNode createSentinel()** – מתודה אשר מאתחלת צומת חדש עם מפתח null אשר מצביע לעצמו מימין ומשמאל ומחזירה אותו.

**סיבוכיות**: O(1)

1. **boolean empty()** – מתודה המחזירה true אם ורק אם העץ ריק. עושה זאת ע"י בדיקה אם גודל הערימה הוא 0.   
   **סיבוכיות:** O(1)
2. **HeapNode insert(int key)** –
3. **void deleteMin()** – מתודה אשר מוחקת את הצומת בעל המפתח המינימלי מתוך הערימה. אופן הפעולה: באמצעות המצביע לצומת המינימום עושים איטרציה על בניו של המינימום ומוסיפים אותם כשורשים ברשימת השורשים של הערימה.על כל שורש שנוסף לרשימת השורשים גדל שדה הפוטנציאל ב-1.לאחר מכן מוחקים את צומת המינימום מרשימת השורשים ומורידים 1 משדה הפוטנציאל. עתה, יש למצוא את המינימום החדש ונעשה זאת באמצעות מתודת consolidate שתבצע גם את תהליך תיקון הערימה.

**סיבוכיות worst case**: O(n)

**סיבוכיות amortized**: O(log n)

1. **void consolidate()** –
2. **HeapNode link(HeapNode y, HeapNode x)** – פונקציה המקבלת 2 שורשים של עצים (מתוך רשימת השורשים של הערימה) ומאחדת אותם לעץ אחד ע"י הוספת השורש עם המפתח הגדול יותר לרשימת הבנים של השורש השני. השורש שנתלה על גבי השורש השני יימחק מרשימת השורשים וכמו כן דרגת העץ המאוחד תגדל ב-1 מדרגתו הנוכחית. הפונקציה תחזיר את שורש העץ המאוחד.הנחת קדם: השורשים נמצאים ברשימת השורשים וכמו כן דרגתם זהה.

**סיבוכיות**: O(1)

1. **HeapNode findMin()** –
2. **void meld(FibonacciHeap heap2)** - מתודה המאחדת ערימת פיבונאצ'י עם הערימה הנוכחית. אופן פעולה: אם הערימה השניה ריקה – אין צורך לבצע דבר ונסיים ביצוע. אם הערימה הנוכחית ריקה, נעתיק את כל המצביעים מתוך הערימה השניה. אחרת, נעדכן את שדה המינימום להצביע על המינימלי מבין 2 שדות המינימום של הערימות. נעדכן את גודל הערימה להיות גודלה הנוכחי ועוד גודל הערימה השניה. נעדכן את פוטנציאל הערימה להיות פוטנציאל הערימה הנוכחית ועוד פוטנציאל הערימה השניה. לבסוף, נאחד את רשימות השורשים של הערימות. מכיוון שרשימות אלה הן רשימות מקושרות דו-כיווניות האיחוד נעשה בO(1) זמן ע"י שינוי מספר מצביעים.

**סיבוכיות**: O(1)

1. **int size()** –
2. **int[] countersRep()** – המתודה מחזירה מערך של מספרים שלמים ובו הערך באינדקס i הוא מספר העצים בערימה אשר דרגתם i. אופן פעולה: אתחול מערך שגודלו 5\*log(n) (שקול לחסם שנלמד בכיתה על דרגה מקסימלית של עץ בערימת פיבונאצ'י לאחר החלפת בסיס הlog לבסיס 10(. עתה תתבצע איטרציה על רשימת השורשים של הערימה – לכל שורש שמור שדה הדרגה ואז ערך המערך באינדקס הדרגה יעלה ב-1. לבסוף נחזיר את המערך.

**סיבוכיות** **worst case**: O(n) (לדוגמא, במקרה ונעשו רק פעולות insert יהיו n שורשים ברשימת השורשים).

**סיבוכיות best case**: O(log n)

סיבוכיות המתודה תשתפר באם יתבצעו קריאות לdeleteMin או delete אשר מתקנות את הערימה ע"י איחוד עצים בעלי דרגות זהות, אך היות וזה לא מובטח לא ניתן להניח סיבוכיות amortized טובה יותר מO(n).

1. **void delete(HeapNode x)** –
2. **void decreaseKey(HeapNode x, int delta)** – המתודה מקבלת צומת בערימה (הנחת קדם: הצומת קיים בערימה) ומספר. המתודה תפחית מספר זה מן המפתח של הצומת תוך שמירה על חוקיות הערימה. אופן הפעולה: תחילה נעדכן את ערך המפתח לערך העדכני. אם לא הופרה חוקיות הערימה, כלומר המפתח המעודכן גדול מזה של אביו, נסיים את הביצוע. אחרת, ננתק את הצומת (שהוא אולי שורש של תת-עץ) מאביו על ידי מחיקתו מרשימת הבנים ונוסיף אותו כשורש ברשימת השורשים של הערימה. נמשיך בתהליך של cascading cut – נסתכל על השדה isMarked בצומת האב; אם ערך שדה זה הוא false הרי שלא נותק ממנו בן בעבר ונסיים התהליך בסימון צומת האב ע"י השמת ערך true בשדה זה. אם ערך שדה זה הוא true, הרי שכבר נותק ממנו אחד הבנים בעבר ולכן נחתוך גם אותו מאביו ונמשיך רקורסיבית בתהליך הcascading cut.

**סיבוכיות** **worst case**: O(n)

**סיבוכיות amortized**: O(1)

1. **void cut(HeapNode x, HeapNode y)** –
2. **void cascadingCut(HeapNode y)** – מתודה אשר מקבלת צומת שנחתך ממנה אחד מבניה בתהליך cut. נסתכל על השדה isMarked של הצומת; אם ערך שדה זה הוא false הרי שזו הפעם הראשונה שנחתך לצומת זו בן ונסיים התהליך בסימון הצומת ע"י השמת ערך true בשדה זה. אם ערך שדה זה הוא true, הרי שנותק ממנו בן נוסף בעבר ולכן נחתוך גם את צומת זה מאביו ע"י שימוש בפעולה cut ונמשיך רקורסיבית בתהליך הcascading cut עם ההורה של צומת זה. התהליך יסתיים כאשר נגיע לצומת שלא היה מסומן לפני כן או בהגעה לשורש.

**סיבוכיות** **worst case**: O(n)

**סיבוכיות amortized**: O(1)

1. **int potential()** – פונקציה המחזירה את שדה הפוטנציאל שערכו מתוחזק ע"י שאר פעולות הערימה. ערך שדה הפוטנציאל הוא מספר העצים + 2\*מספר הצמתים המסומנים**.   
   סיבוכיות**: O(1)

**מתודות סטטיות:**

1. **int totalLinks()** – המתודה מחזירה את הערך השמור במשתנה הסטטי totalLinks. ערכו של משתנה זה הוא מספר הlinkים שבוצעו בין עצים מדרגה זהה לאורך ריצת התוכנית כולה (סכום הlinks בכל הערימות שנוצרו בהרצה).

**סיבוכיות**: O(1)

1. **int totalCuts()** - המתודה מחזירה את הערך השמור במשתנה הסטטי totalCuts. ערכו של משתנה זה הוא מספר הcuts שבוצעו לאורך ריצת התוכנית כולה (סכום הcuts בכל הערימות שנוצרו בהרצה).

**סיבוכיות**: O(1)

class FibonacciHeap

1. public FibonacciHeap()
2. public boolean empty()
3. public HeapNode insert(int key)
4. public void deleteMin()
5. private void consolidate()
6. private HeapNode link(HeapNode y, HeapNode x)
7. public HeapNode findMin()
8. public void meld(FibonacciHeap heap2)
9. public int size()
10. public int[] countersRep()
11. public void delete(HeapNode x)
12. public void decreaseKey(HeapNode x, int delta)
13. private void cut(HeapNode x, HeapNode y)
14. private void cascadingCut(HeapNode y)
15. public int potential()
16. public int countMarked(HeapNode node)
17. public static int totalLinks()
18. public static int totalCuts()

HeapNode {

Integer key;

HeapNode parent;

HeapNode right;

HeapNode left;

HeapNode child;

boolean isMarked;

int rank;

1. public HeapNode(int key)
2. private HeapNode()
3. public void appendSibling(HeapNode node)
4. public void deleteSibling(HeapNode node)
5. public boolean isSentinel()

class HeapNodeIterator implements Iterator<HeapNode> {

HeapNode current;

1. public HeapNodeIterator(HeapNode node)
2. public boolean hasNext()
3. public HeapNode next()