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 בערימה 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 שדה המפתח להיות המפתח הנתון, מאתחל את המצביעים של האח הימני והשמאלי להצביע לצומת עצמו ומאתחל את child כזקיף.
   סיבוכיות: O(1)

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

3. **Integer getKey()** – מחזיר את המפתח של הצומת.
   סיבוכיות: O(1)

4. **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)

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

6. **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)

7. **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 השדהcurrent להיות אחיו הימני של הצומת. הנחת קדם: הצומת שנשלח לבנאי הוא זקיף.

סיבוכיות: O(1)

2. **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)

3. **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. מאתחל את sentinel להיות צומת חדש עם מפתח null אשר מצביע לעצמו מימין ומשמאל.
סיבוכיות: O(1)

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

3. **boolean empty()** – מתודה המחזירה true אם ורק אם העץ ריק. עושה זאת ע"י בדיקה אם גודל הערימה הוא 0.
סיבוכיות: O(1)

4. **HeapNode insert(int key)** – הפעולה מקבלת מפתח ויוצרת עצם חדש מטיפוס HeapNode שהמפתח שלו הוא הkey הנתון. כעת, משום שההכנסה עצלה, העצם החדש מייצג עץ בינומי מגודל 0. כלומר, נוסף לנו שורש חדש, אותו נרצה להוסיף לרשימת השורשים של הערימה, לכן נקרא למתודה sentinel.appengSibling(node).
בנוסף, יתכן שהמפתח החדש קטן מהמפתח של המינימום ששמור לנו כשדה במחלקה עד כה, לכן נעדכן את שדה המינימום במידת הצורך.
לבסוף, נגדיל את גודל הערימה ואת פוטנציאל הערימה ב-1 ונחזיר את הצומת החדש.
סיבוכיות: O(1)

5. **void deleteMin()** – מתודה אשר מוחקת את הצומת בעל המפתח המינימלי מתוך הערימה. אופן הפעולה: אם העץ ריק, אין צומת מינימום ונסיים הביצוע. אחרת, באמצעות המצביע לצומת המינימום נבצע איטרציה על בניו של המינימום ונוסיף אותם כשורשים ברשימת השורשים של הערימה ע"י קריאה למתודת appendSibling של הsentinel. לכל שורש שנוסף לרשימת השורשים נעדכן את מצביע הparent שלו לnull וכמו כן נגדיל שדה הפוטנציאל ב-1. לאחר מכן נמחק את צומת המינימום מרשימת השורשים ע"י פעולת deleteSibling של הsentinel ונוריד 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)

6. **void consolidate()** – מתודה שמטרתה לאחד עצים בעלי דרגה זהה עד אשר יהיה לכל היותר עץ אחד מכל דרגה. ראשית, נערוך בדיקה אם גודל העץ 0, אם כן, לא נעשה דבר ונחזור. אחרת, נבנה מערך שיכיל איברים מטיפוס HeapNode שגודלו 5*log(n) (שקול לחסם שנלמד בכיתה על דרגה מקסימלית של עץ בערימת פיבונאצ'י לאחר החלפת בסיס הlog לבסיס 10). האינדקסים במערך מייצגים את דרגות העצים הבינומיים המרכיבים את הערימה. עתה נבצע איטרציה על רשימת השורשים:
   - נתבונן באיבר הבא ברשימת השורשים, נסמנו ב- x.

- נניח דרגתו של העץ x היא d.
- כל עוד קיים עץ במקום ה d במערך, נבצע:
  i. את העץ במקום ה- d במערך נשים במשתנה y.
  ii. נעשה פעולת link בין x ל-y (פעולת ה link פועלת על 2 עצים באותו הסדר. הפעולה תתלה את y על x או להיפך ותחזיר את העץ החדש).
  iii. היות וכל פעולת link מפחיתה את מספר העצים ב-1, נפחית 1 מפוטנציאל העץ.
  iv. כעת נשים במקום d במערך null.
  v. משום שעשינו link בין שני עצים מדרגת d, קיבלנו כי x הוא עץ מדרגה d+1. נכניס את x למקום d+1 במערך החדש. (אם קיים איבר במערך במקום d+1, הלולאה תמשיך, וחוזר חלילה).

עתה נבנה את רשימת השורשים מחדש. נאפס את הרשימה הקיימת. נבצע איטרציה על המערך – בכל אינדקס במערך אשר מכיל עץ (שאינו null) נוסיף עץ זה לרשימה. תוך כדי נחפש את המינימום החדש ע"י הסתכלות על שורשי העצים (ע"פ חוקיות הערימה מובטח שהמינימום יהיה שורש של אחד העצים).

סיבוכיות worst case: O(n)

סיבוכיות amortized: O(log n)

7. **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. הנחת קדם: השורשים נמצאים ברשימת השורשים וכמו כן דרגתם זהה. אופן הביצוע: אם המפתח של x גדול מהמפתח של y נקרא למתודה מחדש עם סדר צמתים הפוך (כלומר נקרא ל(link(x,y). עתה מובטח לנו שהמפתח של צומת x קטן יותר. נמחק את y מרשימת השורשים ע"י קריאה לdeleteSibling של הsentinel. נוסיף את y כבן של x ע"י קריאה לappendSibling ברשימת הבנים של x. נגדיל את דרגת x ב-1, נעדכן את שדה הparent של y להיות x. לבסוף נגדיל את השדה הסטטי totalLinks ב-1 ונחזיר את x.

סיבוכיות: O(1)

8. **HeapNode findMin()** – פונקציה אשר מחזירה מצביע לצומת המינימום של הערימה. אנו מתחזקים שדה min המחזיק את האיבר המינימלי בערימה. לכן, נחזיר את המצביע בשדה min.
סיבוכיות: O(1)

9. **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 ב(O(1 זמן ע"י שינוי מספר מצביעים.
סיבוכיות: O(1)

10. **int size()** – פונקציה המחזירה את גודל הערימה. נחזיר את שדה ה size המחזיק את גודל הערימה.
סיבוכיות: O(1)

11. **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 נגדיל את ערך המערך באינדקס הדרגה 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).

12. **void delete(HeapNode x)** – פונקציה אשר מוחקת מן הערימה צומת x המתקבל כקלט. הנחת קדם: הצומת קיים בערימה. נקטין את המפתח של האיבר x שקיבלנו כפרמטר למינוס אינסוף (במקרה שלנו מדובר בערימה בה מפתחות האיברים אי-שליליים, ולכן מינוס אינסוף מיוצג ע"י 1-). כעת האיבר x הוא המינימלי בערימה. נוכל להפעיל את המתודה deleteMin אשר תמחק את x מהערימה.

סיבוכיות worst case: O(n) (במקרה שבו התבצעו רק פעולות insert פעולת deleteMin תיקח O(n) זמן)

סיבוכיות amortized: O(log n) (סיבוכיות מקסימלית מתוך ניתוח הסיבוכיות של 2 המתודות אשר נעשה בהן שימוש בפונקציה זו)

13. **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 (זה נעשה באמצעות מתודת cut). נמשיך בתהליך של cascading cut (הרחבה בתיעוד המתודה cascadingCut). לבסוף, נבדוק אם ערך המפתח העדכני קטן מזה של הצומת השמור בשדה המינימום. אם כן, נעדכן את המצביע לצומת המינימון.

סיבוכיות worst case: O(n)

סיבוכיות amortized: O(1)

14. **void cut(HeapNode node, HeapNode parent)** – הפעולה מקבלת 2 צמתים מהערימה, כאשר node הוא אחד הבנים של parent. המטרה היא "לחתוך" את הצומת מרשימת הבנים של אביו ולהוסיף אותו כשורש ברשימת השורשים. ראשית נמחק את node מרשימת הצאצאים של parent ע"י פעולת deleteSibling ברשימת הבנים ונקטין את הדרגה של parent ב-1. לאחר מכן נוסיף את node לרשימת השורשים ע"י פעולת appendSibling ברשימה. node הוא שורש עתה ולכן נאתחל את שדה ה- parent שלו להיות null ונגדיל את פוטנציאל הערימה ב-1 (נוסף עץ חדש לרשימת השורשים). בנוסף, משום שעתה node הוא שורש, יש לבטל את ה"סימון" שלו אם קיים כזה (שדה isMarked של צומת). במידה ואכן היה מסומן לפני ביצוע החיתוך נבטל את הסימון ונקטין את שדה הפוטנציאל ב-2. חשוב - שדה הסימון של parent יתעדכן במתודה אחרת (cascadingCut).

לבסוף, נגדיל את השדה הסטטי שסופר את כמות הcuts הכוללת ב-1.

סיבוכיות worst case: O(1)

15. **void cascadingCut(HeapNode y)** – מתודה אשר מקבלת צומת שנחתך ממנו אחד מבניו בתהליך cut. תחילה נבדוק אם צומת זה הוא שורש ע"י הסתכלות על שדה הparent שלו. אם אכן שורש, נסיים הביצוע. אחרת, 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 וכמו כן נגדיל את פוטנציאל העץ ב-2. 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)

16. **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)

17. **Iterator<HeapNode> iterator()** – פעולה המחזירה iterator על רשימת השורשים של הערימה. נאתחל איטרטור חדש ונחזיר אותו

סיבוכיות: O(1)

<u>מתודות סטטיות:</u>

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)

2. **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)

מדידות

Sequence 1

m	Run-Time	totalLinks	totalCuts	Potential
1000	1	0	0	1000
2000	2	0	0	2000
3000	2	0	0	3000

זמן ריצה אסימפטוטי: זמן הריצה האסימפטוטי של סדרת הפעולות הוא O(m), משום שמתבצעות m פעולות insert שזמן הריצה שלהן הוא O(1).

מספר הlinks: ערימת פיבונאצ'י מבצעת lazy insertion, כלומר לא נעשה consolidate בפעולת insert ולכן בסדרת פעולות זו כלל לא התבצעו פעולות link.

מספר הcuts: מכיוון שאין קריאות לdecreaseKey ברור שלא התבצעו חיתוכים ולכן תמיד totalCuts יהיה אפס.

פוטנציאל: פוטנציאל הערימה בסוף סדרת הפעולות הוא m = O(m), משום שמתבצעות m פעולות insert שכל אחת בתורה מוסיפה עץ לרשימת השורשים וכמובן שאף אחד מהצמתים בערימה אינו מסומן, אזי בסוף סדרת הפעולות יהיו בדיוק m עצים בערימה.

Sequence 2

m	Run-Time	totalLinks	totalCuts	Potential
1000	4	1891	0	6
2000	5	3889	0	6
3000	6	5772	0	7

<u>זמן ריצה אסימפטוטי</u>: מתבצעות m פעולות insert בסיבוכיות זמן O(1) ולאחריהן m/2 פעולות deleteMin בסיבוכיות זמן ממוצעת (אמורטייזד) O(log m). לכן זמן הריצה האסימפטוטי הוא:

$$m * O(1) + \frac{m}{2} * O(\log m) = O(m) + O(mlog\ m) = \boldsymbol{O(mlog\ m)}$$

<u>מספר הlinks</u>: בקריאה הראשונה לdeleteMin נעשות O(m) פעולות link משום שהערימה היא למעשה רשימה מקושרת אחת עם m איברים. שאר פעולות הdeleteMin עושות O(1) פעולות link, כי לאחר הסרת המינימום הערימה ברובה מתוקנת. לכן סך כל פעולות הlink הוא:

$$1 * \mathrm{O}(m) + \left(\frac{m}{2} - 1\right) * O(1) = O(m) + O(m) = \boldsymbol{O(m)}$$

<u>מספר הcuts</u>: מכיוון שאין קריאות לdecreaseKey ברור שלא התבצעו חיתוכים ולכן תמיד totalCuts יהיה אפס.

<u>פוטנציאל</u>: מכיוון שהפעולה האחרונה שמתבצעת היא פעולת deleteMin הרי שמתבצע תהליך של consolidate בסיום סדרת הפעולות. ברור גם כי אין צמתים מסומנים כי לא עשינו כלל חיתוכים ולכן פוטנציאל הערימה בסיום הביצוע הוא בדיוק מספר הביטים ה"דולקים" בייצוג הבינארי של גודל הערימה, כלומר פוטנציאל הערימה הוא פונקציה של מספר הביטים הנדרשים בייצוג הבינארי של גודל הערימה: O(log(m/2)) = O(log m)