**Binomical heap**

**אור הירשמן 311146773 גיא יום טוב 305691057**

**\* Guy Yom-Tov --- guyyomtov --**

**\* Or Hirshman --- orhirshman --**

**המחלקה BinomialHeap:**

המחלקה מייצגת ערימה בינומית. הערימה מיוצגת ע"י רשימת שורשי העצים הבינומיים אשר מסודרים בסדר עולה מהעץ הבינומי עם הדרגה הקטנה ביותר לעץ הבינומי עם הדרגה הגבוהה ביותר. (למעשה, לא קיימת רשימה בפועל, אלא שרשרת של מצביעים) בנוסף, אנו בחרנו במימוש שאינו משתמש ברשימת מצביעים מעגלית.

**שדות :**

**private HeapNode firstNode**- מצביע לשורש העץ הבינומי בעל הדרגה הקטנה ביותר בערימה.

**private int size** – מספר הצמתים הכולל בערימה.

**private HeapNode minNode** – מצביע לצומת עם הערך (key) הקטן ביותר בערימה.

**פעולות:**

**קיימים 2 בנאים:** בנאי ריק, המאתחל ערימה חדשה, ובנאי שמקבל ערימה קיימת ואת מספר הצמתים בה. שניהם בסיבוכיות O(1).

**empty() .1** – סיבוכיות- O(1) מחזירה ערך true אם ורק אם הערימה ריקה. הפונקציה בודקת האם 0=size ומחזירה תשובה בהתאם.

**insert(i) .2** - הכנסת איבר שערכו i לערימה. אם הערימה ריקה נשנה את ערכיהם של firstNode ושל minNode ל HeapNode חדש בעל הערך i כ-key ונשנה את size ל1. במקרה והערימה איננה ריקה, ניצור ערימה בינומית חדשה המכילה צומת בודד עם הערך i כ-key ונבצע meld בין הערימה הקיימת לערימה החדשה וכך למעשה נחבר ביניהם וניצור ערימה חדשה המכילה את הערך i בנוסף. כיוון שבערימה יש לכל היותר logn עצים בינומיים ועבור כל עץ קיים מספר קבוע של פעולות, הסיבוכיות היא (.O(logn

**deleteMin() .3** - מחיקת איבר המינימום של הערימה מבלי להחזיר אותו. הפונקציה מניחה כי הערימה איננה ריקה. נמחק את הצומת המכיל את האיבר המינימאלי, כתוצאה מכך הערימה תתפרק לאוסף עצים בינומיים. כעת נמזג אוסף זה עם שאר הערימה בעזרת הפונקציה meld. (שתדאג למצוא את המינימום החדש ולעדכן את מספר הצמתים בעץ). כיוון שבערימה יש לכל היותר logn עצים בינומיים ועבור כל עץ קיים מספר קבוע של פעולות, הסיבוכיות היא (.O(logn

**findMin() .4** - הפונקציה מחזירה את ערכו של איבר המינימום בערימה. הפונקציה מניחה כי הערימה איננה ריקה. מחזירה את השדה key של השדה minNode. סיבוכיות.. O(1)

**meld(heap2) .5** - מיזוג הערימה עם ערימה נוספת heap2. בתחילה, מתקיימת קריאה לפונקציה merge (הפועלת בסיבוכיות O(logn) ). אשר מאחדת את הערימה הנוכחים יחדיו עם heap2 לערימה אחת המכילה את כל העצים הבינומיים משתי הערימות, בעוד שהם מסודרים בסדר עולה. כעת הערימה שלנו מצביעה על ערימה המכילה את כל העצים הבינומיים משתי הערימות (ייתכן כי ישנן דרגות המופיעות יותר מפעם אחת), לכן במקרים אלו, נבצע לינקים בין עצים. (לינק שקול לחיבור בינארי של דרגות העצים) ניתן לבצע זאת כיוון שעצים בעלי דרגות זהות הינם סמוכים. הפונקציה מעדכנת את צומת הminNode ואת size להיות סכום מספר הצמתים בערימות. כיוון שבשתי הערימות יש לכל היותר logn עצים בינומיים ועבור כל עץ קיים מספר קבוע של פעולות, הסיבוכיות היא (.O(logn

**size() .6** - הפונקציה מחזירה את מספר האיברים בערימה השווה לערכו של השדה size. סיבוכיות .O(1)

**minTreeRank() .7** **-** הפונקציה מחזירה את דרגת העץ הבינומי הקטן ביותר בערימה. הפונקציה מניחה שהערימה איננה ריקה. הפונקציה מחזירה את דרגת השדה firstNode אשר מייצג את העץ הבינומי עם הדרגה הקטנה ביותר. סיבוכיות.O(1)

**binaryRep() .8** **-** הפונקציה מחזירה את הייצוג הבינארי של הערימה. כלומר, היא מחזירה מערך בוליאני המוגדר מאינדקס 0 עד הדרגה המקסימאלית שקיימת בעץ. עבור כל עץ שקיים בערימה, הערך באינדקס של הדרגה יהיה TRUE. הפונקציה ממירה את size מייצוג עשרוני לבינארי ולאחר מכן עוברת על כל התוים בייצוג הבינארי באופן הבא: עבור כל אינדקס המכיל את הערך 1 , יושם במערך הבוליאני באינדקס זה הערך true. במקרה בו הערימה ריקה, נחזיר מערך בוליאני עם תא אחד המכיל false. כיוון שבערימה יש לכל היותר logn עצים בינומיים ועבור כל עץ קיים מספר קבוע של פעולות, הסיבוכיות היא (.O(logn

**arrayToHeap(array) .9 –** הפונקציה בונה ומחזירה ערימה חדשה, הנוצרת ממערך מספרים. מתקבלת ערימה המכילה את איברי המערך בלבד. (אם הערימה הכילה קודם לכן איברים, הם ימחקו). ראשית, הפונקציה קוראת לפונקצית העזר becomeEmpty המרוקנת את הערימה מתוכנה. כעת עוברת בלולאה ומכניסה את איברי המערך לפי הסדר לתוך הערימה הריקה. ההכנסות נעשות בעזרת הפונקציה insert. הוכח בשיעור שזמן amortized של n הכנסות הינו לינארי במספר ההכנסות O(array.length()) -

**isValid() .10** –הפונקציה מחזירה true אם"ם הערימה הינה ערימה בינומית תקינה. הפונקציה עוברת על כל שורשי העריצה (העצים שניתן להגיע אליהם דרך firstNode ע"י שימוש בפונקציה getNextSibling) ובודקת עבור כל שורש האם הוא עץ בינומי והאם דרגתו שווה למספר הילדים שלו באמצעות שימוש בפונקציה checkTree. הפונקציה בודקת עבור כל דרגה של השורשים, אם היא מופיעה פעם אחת. הפונקציה מבצעת מספר קבוע של פעולות עבור כל צומת בעץ ולכן הסיבוכיות היא:. O(n)

**11. becomeEmpty()** -הפונקציה מוחקת את כל איברי הערימה ומעדכנת את השדות הרלוונטים כך שמתקבלת ערימה בינומית ריקה. סיבוכיות O(1).

**12.** **getfirstNode()** - הפונקציה מחזירה את השדה firstNode, שהוא שורש העץ הבינומי בעל הדרגה הקטנה ביותר בערימה. סיבוכיות O(1).

**13. merge(BinomialHeap heap2)** – פונקצית עזר המשמשת עבור הפונקציה meld. מחזירה ערימה חדשה המייצגת את איחוד הערימה הנוכחית, בנוסף לערימה heap2. מתקבלת ערימה המכילה את העצים הבינומיים משתי הערימות, בעודם מסודרים בסדר עולה מבחינת דרגות העצים. בערימה החדשה עלולות להופיע דרגות של עץ יותר מפעם אחת. כיוון שהעצים מסודרים בסדר עולה, אם קיימים שני עצים בעלי אותה הדרגה, הינם סמוכים אחד לשני. עובדה זו מאפשרת לפונקציה meld לפעול בקלות יחסית.. כיוון שבשתי הערימות יש לכל היותר logn עצים בינומיים ועבור כל עץ קיים מספר קבוע של פעולות, הסיבוכיות היא (.O(logn

**14. link(HeapNode hn1, HeapNode hn2)** - פונקצית עזר המשמשת עבור הפונקציה meld. הפונקציה בודקת האם דרגת hn1 גבוהה מדרגת hn2 וכמו כן בודקת ששתי הערימות אינן null. מבצעת לינק ביניהם כפי שתואר בשיעור (עבור המקרה ה1). סיבוכיותO(1) .

**15.** **countChildren(HeapNode node)** - פונקצית עזר המשמשת עבור הפונקציה checkTree. בודקת שהצומת node אינו null, וסופרת את בניו. סיבוכיות: בw.c, כל הצמתים הינם תחת עץ בינומי יחיד, במקרה זה דרגתו תהיה logn O(logn) .

**16.** **checkTree (HeapNode thisTree, int degree)** - פונקצית עזר המשמשת עבור הפונקציה isValid. הפונקציה בודקת האם העץ איננו null והאם דרגתו הינה הדרגה המתאימה. הפונקציה בודקת האם הוא עץ בינומי המקיים את כלל הערימה, והאם בניו עצים בינומיים בדרגות המתאימות. בw.c, כל הצמתים נמצאים בעץ אחד. ( n צמתים) ועבור כל צומת קיים מספר קבוע של פעולות O(n).

(הערה: n = size)

**תת המחלקה BinomialHeap.HeapNode:**

מחלקה אשר מייצגת צומת בערימה בינומית.

**שדות :**

**private int key** – המפתח של הצומת.

**private** **int degree** – דרגת הצומת (שקול למספר הילדים).

**private** **HeapNode** **child** – מצביע לילד של הצומת עם הדרגה המקסימאלית.

**private** **HeapNode** **next** – מצביע לאח עם הדרגה הבאה בתור.

**private** **HeapNode** **parent** – מצביע להורה של הצומת

**פעולות :**

**Constructor**

**Getters לשדות**

**Setters לשדות**

כל הפעולות של מחלקה זו הן בסיבוכיות O(1).

**מדידות:**

על השאלות הבאות יש לענות ללא הסתמכות על הרצת התוכנה:

1. **תארו סדרה של m פעולות על המבנה שמימשתם, כך שעבור כל פעולה מבין m-1 הפעולות הראשונות, זמן amortized לפעולה קבוע, ואילו עבור הפעולה האחרונה בסדרה זמן הריצה הוא (log m).**

* עבור m-1 הפעולות הראשונות, נכניס איברים בסדר עולה, בכל פעם נכניס איבר, הוא יעשה MELD עם האיברים הקיימים .הסדר העולה יאפשר זמן אמורטייזד של פעולה כל פעולה ב O(1)
* הפעולה האחרונה תהיה מחיקת האיבר המינימלי, מכיוון שהכנסנו אותו ראשון, הוא יהיה ה NODE של השורש, ומחיקתו תרגום לפירוק של הערמה לlogm עצים , ונצטרך לעשות להם meld. סה"כ (log m).

1. **מה זמן הריצה הכולל מבחינה אסיפמטוטית של *m* הפעולות כפונקציה של *m*?**

* נשים לב ש WC של פעולת הכנסה יהיה O(log(m)) כפי שנלמד בשיעור. גם הפעולה האחרונה באותו WC, ולכן סה"כ זמן אסימפטוטי לm פעולות יהיה O(m\*log(m))

1. **עבור *m* = 1,000, 2,000, 3,000 מה יהיה הייצוג הבינארי של הערימה בסדרה של *m* פעולות כפי שתיארתם (i) אחרי הפעולה ה-(m-1) (ii) אחרי הפעולה ה-m?**

* עבור m=1000

1. אחרי הפעולה ה-(m-1) – 1111100111 (999)
2. אחרי הפעולה ה-m- 1111100110 (998)

* עבור m=2000

1. אחרי הפעולה ה-(m-1) – 11111001111 (1999)
2. אחרי הפעולה ה-m- 11111001110 (1998)

* עבור m=3000

1. אחרי הפעולה ה-(m-1) – 101110110111 (2999)
2. אחרי הפעולה ה-m- 101110110110 (2998)

**על השאלה הבאה יש לענות עם הסתמכות על הרצת התוכנה:**

1. **הריצו את סדרת הפעולות שתיארתם בסעיף 1. סיפרו את מספר פעולות linking (כלומר, תליה של עץ בינומי אחד על השני) ב-(m-1) הפעולות הראשונות (עבור שלושת ערכי m). הציגו את התוצאות שקיבלתם (מספר ה-linking) והסבירו אותם.**

* עבור m=1000

1. עבור (m-1) פעולות – 991 links
2. עבור הפעולה ה-m- 9 links

* עבור m=2000

1. עבור (m-1) פעולות – 1990 links
2. עבור הפעולה ה-m- 10 links

* עבור m=3000

1. עבור (m-1) פעולות – 2990 links
2. עבור הפעולה ה-m- 11 links

ניתן לראות שספירת מספר הלינקים תואם לסדרת הפעולות שנתנו בסעיף הקודם. עבור הm-1 פעולות ראשונות יש לנו מספר לינקים שכאשר מחלקים אותו במספר הפעולות יוצא O(1) פעולה, כיוון שזמן הפעולה פה תלוי במספר הלינקים שנעשים. ואילו הפעולה האחרונה תמיד לוקחת כ- log m לינקים, כלומר פעולה אחת שב O(logm) .

1. **הריצו את סדרת הפעולות שתיארתם בסעיף 1 עבור שלושת ערכי m. וודאו שהתוצאות של (i) ו-(ii) הם כפי שצפיתם. רשמו במפורש מה החזירה פונקציית binaryRep אחרי פעולה m-1 ואחרי פעולה m, והשוו לחישוב התיאורטי שביצעתם.**

* עבור m=1000

1. אחרי הפעולה ה-(m-1) –

[true, true, true, true, true, false, false, true, true, true]

1. אחרי הפעולה ה-m-

[true, true, true, true, true, false, false, true, true, false]

* עבור m=2000

1. אחרי הפעולה ה-(m-1) –

[true, true, true, true, true, false, false, true, true, true, true]

1. אחרי הפעולה ה-m-

[true, true, true, true, true, false, false, true, true, true, false]

* עבור m=3000

1. אחרי הפעולה ה-(m-1) –

[true, false, true, true, true, false, true, true, false, true, true, true]

1. אחרי הפעולה ה-m-

[true, false, true, true, true, false, true, true, false, true, true, false]

ניתן לראות שהתשובות תואמות לחישוב התאורטי, וזה כמובן בשל העובדה שהפונקציה **binaryRep** מחזירה ייצוג בינארי של גודל הערמה, וזה נגזר מכמות האיברים, בדיוק כמו החישוב התאורטי שלנו מקודם.