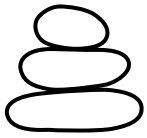
Binomial Heap



פרויקט שני בקורס מבני נתונים

עמוד 1 – פרטי הסטודנטיות

עמודים 2-11 – תיעוד המחלקות והמתודות

עמודים 12-17 – חלק ניסויי/תיאורטי

מגישות:	אליה אביטל	סתיו נקסון
תעודות זהות:	205854540	318814829
user name במודל:	elyaa	stavnacson

תיעוד המחלקות :

על מנת לממש ערמות בינומיות, מימשנו 3 מחלקות, מחלקה פנימית של הצמתים HeapNode, מחלקה פנימית HeapItem השומרת את המידע לכל צומת ומתחזקת מצביע אליו (ומצביע ממנו) ומחלקת האם המייצגת את מבנה הערימה BinomialHeap. כעת נפרט על תיעוד המחלקות, המתודות הנדרשות ומתודות העזר בהן השתמשנו. נציין מראש שבניתוחי הסיבוכיות אנו מניחות כי גישה לזיכרון, גישה לשדות של מופע, ופעולות אריתמטיות מתבצעות בזמן קבוע O(1).

תיעוד מימוש ערימה בינומית

תיעוד המחלקה HeapItem :

מטרת המחלקה היא שמירת המידע של הצומת כגון key, info בנפרד מהצומת בכדי שנוכל לבצע פעולת decrease key ביעילות ומבלי לפגוע במצביעים החיצוניים של המשתמש.

1. public HeapItem () :

מהות : הבנאי של מחלקת ה-items. לכל צומת שדות בהם אנו משמרים מידע רלוונטי. את המידע לגבי המפתחות והערכים של הצומת נרצה לשמור בנפרד, כאשר זהותו והצומת עצמו מצביעים זה על זה. זאת על מנת שנוכל לבצע החלפות בין items מבלי לשנות פיזית את המבנה.
אופן הפעולה : ברגע שמאותחל item נשמר הערך 0 במפתח שלו, הערך null והוא מצביע על צומת null.
סיבוכיות : O(1).

2. Public int getKey() :

מהות : המתודה מחזירה את המפתח של ה-item שעליו היא הופעלה. הקריאה תתבצע דרך צומת.
אופן הפעולה : המתודה נקראת על מופע של item מסוים ומחזירה את שדה ה- key שלו.
סיבוכיות : O(1).

3. Public String getInfo() :

מהות : המתודה מחזירה את המחרוזת (ה-value) של ה-item שעליו היא הופעלה. הקריאה תתבצע דרך צומת.
אופן הפעולה : המתודה נקראת על מופע של item מסוים ומחזירה את שדה ה- info שלו.
סיבוכיות : O(1).

4. Public HeapNode getNode() :

מהות : המתודה מחזירה את הצומת שעליו מצביע ה-item שעליו היא הופעלה.
אופן הפעולה : המתודה נקראת על מופע של item מסוים ומחזירה את שדה ה- Node שלו.
סיבוכיות : O(1).

5. **public void setKey(int key)** :

מהות : המתודה מעדכנת את המפתח של ה-item עליו הופעלה. הקריאה תתבצע דרך צומת.
אופן הפעולה : המתודה מקבלת מספר טבעי, נקראת על מופע של item מסוים, ומעדכנת את שדה ה-key שלו להיות המספר שקיבלה.
סיבוכיות : O(1).

6. **public void setInfo(String info)** :

מהות : המתודה מעדכנת את הערך של ה-item עליו הופעלה. הקריאה תתבצע דרך צומת.
אופן הפעולה : המתודה מקבלת מחרוזת, נקראת על מופע של item מסוים, ומעדכנת את שדה ה-info שלו להיות המחרוזת שקיבלה.
סיבוכיות : O(1).

7. **public void setNode(HeapNode node)** :

מהות : המתודה מעדכנת את המצביע לצומת של ה-item עליו הופעלה.
אופן הפעולה : המתודה מקבלת צומת, נקראת על מופע של item מסוים, ומעדכנת את שדה ה-node שלו להיות הצומת שקיבלה.
סיבוכיות : O(1).

תיעוד המחלקה HeapNode :

המחלקה מייצגת את הצמתים בערמה הבינומית. לכל צומת נתחזק שדות לשמירת מידע רלוונטי ובכך נוכל לבצע ביעילות פעולות על מבני הנתונים והמידע שהוא מחזיק עבורנו.

8. **public HeapNode () :**

מהות : הבנאי של מחלקת הצמתים. לכל צומת שדות בהם אנו משמרים מידע רלוונטי.
אופן הפעולה : ברגע שמאותחל צומת השדות parent,child,item מאותחלים ל-null, שדה ה-next של הצומת מאותחל לתחזק מצביע לצומת עצמו והדרגה, מספר הילדים, מאותחלת לאפס.
סיבוכיות : O(1).

9. **Public HeapItem getItem()** :

מהות : המתודה מחזירה את ה-item שעליו היא הופעלה, ששומר מידע נוסף לגבי הצומת.
אופן הפעולה : המתודה נקראת על מופע של item מסוים ומחזירה את שדה ה- item שלו.
סיבוכיות : O(1).

10. **Public HeapNode getChild()** :

מהות : המתודה מחזירה את הבן של הצומת עליו היא הופעלה.
אופן הפעולה : המתודה נקראת על מופע של item מסוים ומחזירה את שדה ה- child שלו.
סיבוכיות : O(1).

11. Public HeapNode getParent() :

מהות : המתודה מחזירה את ההורה של הצומת שעליו היא הופעלה.
אופן הפעולה : המתודה נקראת על מופע של item מסוים ומחזירה את שדה ה- parent שלו.
סיבוכיות : O(1).

12. Public HeapNode getNext() :

מהות : המתודה מחזירה את המצביע לצומת הבא, של הצומת שעליו היא הופעלה.
אופן הפעולה : המתודה נקראת על מופע של item מסוים ומחזירה את שדה ה- next שלו.
סיבוכיות : O(1).

13. Public int getRank() :

מהות : המתודה מחזירה את הדרגה של הצומת שעליו היא הופעלה.
אופן הפעולה : המתודה נקראת על מופע של item מסוים ומחזירה את שדה ה- rank שלו.
סיבוכיות : O(1).

14. public void setItem(HeapItem item) :

מהות : המתודה מעדכנת את המצביע ל-item של הצומת עליו הופעלה.
אופן הפעולה : המתודה מקבלת אובייקט מטיפוס item, נקראת על מופע של צומת מסוים, ומעדכנת את שדה ה-item שלו להיות האובייקט שקיבלה.
סיבוכיות : O(1).

15. public void setChild(HeapNode child) :

מהות : המתודה מעדכנת את המצביע לבן של הצומת עליו הופעלה.
אופן הפעולה : המתודה מקבלת אובייקט מטיפוס Node, נקראת על מופע של צומת מסוים, ומעדכנת את שדה ה-child שלו להיות האובייקט שקיבלה.
סיבוכיות : O(1).

16. public void setParent(HeapNode parent) :

מהות : המתודה מעדכנת את המצביע להורה של הצומת עליו הופעלה.
אופן הפעולה : המתודה מקבלת אובייקט מטיפוס Node, נקראת על מופע של צומת מסוים, ומעדכנת את שדה ה-parent שלו להיות האובייקט שקיבלה.
סיבוכיות : O(1).

17. public void setNext(HeapNode next) :

מהות : המתודה מעדכנת את המצביע לצומת הבא של הצומת עליו הופעלה.
אופן הפעולה : המתודה מקבלת אובייקט מטיפוס Node, נקראת על מופע של צומת מסוים, ומעדכנת את שדה ה-next שלו להיות האובייקט שקיבלה.
סיבוכיות : O(1).

18. private void Swap(HeapNode node2) :

מהות : המתודה מבצעת החלפה בין שני צמתים מבלי לשנות פיזית את המבנה. ההחלפה שמתבצעת בפועל היא רק בין המצביעים שבין הצמתים ל-items. מטרת מתודה זו היא שכאשר נרצה לבצע פעולת decrease key, במקום לפעפע את הצומת מעלה רק נחליף items.
אופן הפעולה : המתודה מקבלת אובייקט מטיפוס Node, נקראת על מופע של צומת מסוים, ומחליפה את המצביעים שלהם ל-item אחד של השני, כמו כן מעדכנת את המצביעים של אותם items שיצביעו גם הם על הצמתים הנכונים.
סיבוכיות : O(1).

תיעוד המחלקה BinomialHeap :

המחלקה מייצגת את מבנה הערמה. היא מתחזקת שדות לשמירת מידע רלוונטי שעתיד לשמש עבור ביצוע פעולות בסיבוכיות מיטבית. כמו כן מכילה את המתודות לביצוע שינויים ופעולות אלו.

19. public BinomialHeap () :

מהות : הבנאי של מחלקת הערימה.
אופן הפעולה : ברגע שמאותחלת ערמה ריקה, שדה ה-size שלה מאותחל לאפס, שדה numOfTrees השומר את מספר העצים בערמה מאותחל לאפס, ושדות ה- min,last מאותחלים ל-null.
סיבוכיות : O(1).

20. Public int getNumOfTrees() :

מהות : המתודה מחזירה את מספר העצים בערמה בזמן הקריאה.
אופן הפעולה : המתודה נקראת על מופע של ערמה מסוימת ומחזירה את שדה ה- numOfTrees.
סיבוכיות : O(1).

21. Public int getSize() :

מהות : המתודה מחזירה את כמות האיברים בערמה.
אופן הפעולה : המתודה נקראת על מופע של ערמה מסוימת ומחזירה את שדה ה- size שלה.
סיבוכיות : O(1).

22. Public HeapNode getMin() :

מהות : המתודה מחזירה את הצומת בעל מפתח מינימלי בערמה. בחרנו לממש את המבנה כך שבהינתן 2 צמתים עם מפתח מינימלי זהה, המינימום בערמה תמיד יצביע על שורש של עץ.
אופן הפעולה : המתודה נקראת על מופע של ערמה מסוימת ומחזירה את שדה ה- min שלה.
סיבוכיות : O(1).

23. Public HeapNode getLast() :

מהות : המתודה מחזירה את הצומת בעל הדרגה המקסימלית בעץ, כלומר השורש של העץ הכי גדול.
אופן הפעולה : המתודה נקראת על מופע של ערמה מסוימת ומחזירה את שדה ה- last שלה.
סיבוכיות : O(1).

24. **public void setMin(HeapNode min)** :

מהות : המתודה מעדכנת את המצביע לצומת בעל המפתח המינימלי של הערמה עליה נקראה.
אופן הפעולה : המתודה מקבלת אובייקט מטיפוס Node, נקראת על מופע של ערמה מסוימת, ומעדכנת את שדה ה-min שלה להיות האובייקט שקיבלה.
סיבוכיות : O(1).

25. **public void setLast(HeapNode last)** :

מהות : המתודה מעדכנת את המצביע לצומת בעל הדרגה המקסימלית של הערמה עליה נקראה.
אופן הפעולה : המתודה מקבלת אובייקט מטיפוס Node, נקראת על מופע של ערמה מסוימת, ומעדכנת את שדה ה-last שלה להיות האובייקט שקיבלה.
סיבוכיות : O(1).

26. **public void setSize(int size)** :

מהות : המתודה מעדכנת את כמות האיברים בערמה עליה נקראה.
אופן הפעולה : המתודה מקבלת מספר טבעי, נקראת על מופע של ערמה מסוימת, ומעדכנת את שדה ה-size שלה להיות המספר החדש שקיבלה.
סיבוכיות : O(1).

27. **public void setNumOfTrees(int numOfTrees)** :

מהות : המתודה מעדכנת את כמות העצים בערמה עליה נקראה.
אופן הפעולה : המתודה מקבלת מספר טבעי, נקראת על מופע של ערמה מסוימת, ומעדכנת את שדה ה-numOfTrees שלה להיות המספר החדש שקיבלה.
סיבוכיות : O(1).

28. **public HeapItem findMin()** :

מהות : המתודה מחזירה את ה-item של הצומת בעל המפתח המינימלי בעץ.
אופן הפעולה : אם הערמה ריקה המתודה תחזיר null, אחרת תחזיר את ה-item של הצומת אליו מצביע השדה בו שמור המינימום בעזרת getMin() (ראה 22), getItem() (ראה 9).
סיבוכיות : O(1).

29. **public int size()** :

מהות : המתודה מחזירה את כמות האיברים בערמה.
אופן הפעולה : המתודה מחזירה את הערך ששמור בשדה size של הערמה באמצעות getSize() (ראה 21).
סיבוכיות : O(1).

30. public boolean empty() :

מהות : המתודה מחזירה ערך בוליאני (true או false) שמשמעותו האם הערמה ריקה מאיברים.
אופן הפעולה : המתודה מחזירה true אם בשדה size של הערמה שמור המספר 0, באמצעות getSize() (ראה 21).
סיבוכיות : O(1).

31. public int numTrees() :

מהות : המתודה מחזירה את מספר העצים בערמה.
אופן הפעולה : המתודה מחזירה את הערך שמתקבל מקריאה ל- getNumOfTrees() (ראה 20).
סיבוכיות : O(1).

32. Private HeapNode link(HeapNode heapNode1, HeapNode heapNode2) :

מהות : מתודת עזר הממזגת שני עצים בינומיים ומחזירה את השורש של העץ החדש שהתקבל.
אופן הפעולה : המתודה מקבלת שני צמתים שהינם שורשים של עצים בינומיים בעלי אותה דרגה, מסמנת את הצומת בעל המפתח הקטן (או שווה) ב-x והשני ב-y ומחברת את המצביעים הבאים; הופכת את ההורה של y להיות x, אם ל-x אין ילד, ה-next של y יהיה עצמו, ואחרת ה-next של y יהיה ה-next של הילד, וה-next של הילד יתעדכן להיות y. לבסוף נעדכן את y להיות הילד של x ונעלה את הדרגה שלו (המייצגת את מספר הבנים) ב-1. לבסוף נחזיר את x שהינו כעת השורש של העץ החדש שהתקבל.
סיבוכיות : O(1), מבצעים כמות קבועה של פעולות שינויי מצביעים.

33. Private HeapNode[ ] arrayOfTrees(int n) :

מהות : מתודת עזר שמחזירה מערך של צמתים כך שבכל אינדקס i במערך, קיים מצביע לשורש של העץ מדרגה i בערמה, אם קיים, ואחרת null. את הערך n נבחר בהתאם במתודה meld.
אופן הפעולה : תנאי קדם אותו אנו מניחים בכל קריאה למתודה זו הוא שהערמה אינו ריקה. המתודה נקראת על מופע של ערימה ומאתחלת מערך בגודל של הקלט שקיבלה. בלולאת do while שמבצעת איטרציה ראשונה על האיבר האחרון בערמה (ששמור בשדה last), שומרת את דרגת הצומת ואם המיקום במערך שמתאים לדרגה זו מצביע ל-null, תבצע השמה לצומת הנוכחי. כל עוד לא חזרנו לצומת last הלולאה תמשיך כאשר היא ניגשת בכל פעם לאיבר הבא (next). לבסוף תחזיר את המערך שהתקבל עם היציאה מהלולאה.
סיבוכיות : O(logn), מלבד פעולות שמתבצעות בזמן קבוע, עיקר העלות של מתודה זו הוא במעבר על השורשים של עצי הערמה. מאחר והערמה בינומית, הרוחב שלה, כלומר כמות העצים, היא לכל היותר לוגריתמית במספר הצמתים בערמה.

34. Private HeapNode findPrev(HeapNode node) :

מהות : מתודת עזר המשמשת למקרה פרטי פשוט במחיקת צומת מינימלי, כאשר הוא מדרגה אפס כלומר ללא בנים. מטרת המתודה היא בהינתן מצביע לצומת מסוים, למצוא את הצומת הקודם לו (מדרגה נמוכה יותר). נרצה להפעיל אותה על מנת למצוא את השורש החדש שיצביע ל-next של המינימום שנמחק. כפי שציינו ב-22, אופן המימוש הוא שהמינימום תמיד יצביע לשורש של עץ.

אופן הפעולה : המתודה נקראת על מופע של ערמה, מתחילה מהצומת אותו קיבלה כקלט, ומבצעת בלולאת while קפיצות לצומת הבא (באמצעות getNext(), 12) כל עוד הצומת הנוכחי אינו null וגם הצומת הבא אחריו אינו צומת הקלט ממנו התחלנו. כלומר, הרעיון הוא ללכת במעגל מהצומת הנוכחי עד שנגיע לצומת אחד לפניו. המתודה תחזיר את הצומת בו עצרנו, כאשר ייתכן כמובן שאם צומת הקלט הוא השורש של עץ יחיד בערימה, האיבר שיוחזר יהיה הוא עצמו.
סיבוכיות : O(logn) במקרה הגרוע. במקרה בו יש n (או n-1 במקרה שבו אנו עושים שימוש במתודה לאחר מחיקת מינימום) איברים בערמה, וקיים עץ מכל דרגה כלומר הרוחב הוא logn, נצטרך לעבור logn-1 עצים עד שנגיע לעץ שורשו הוא הקודם של צומת הקלט.

35. Private HeapNode findNewMin(HeapNode node) :

מהות : מתודת עזר אשר בהינתן צומת בערמה מחזירה את האיבר בעל המפתח המינימלי מבין הוא ואחיו (המחוברים במצביעי next). נשתמש במתודה זו כאשר נרצה למזג ערמות בינומיות ולעדכן מינימום חדש, וכאשר נרצה למחוק את המינימום מערמה מסוימת ולמצוא את המינימלי מילדיו.
אופן הפעולה : המתודה נקראת על מופע של ערמה, שומרת את הצומת אותו קיבלה כקלט בתור currMin, ואת ה-next שלו בתור curr. בלולאת while, כל עוד הצומת הנוכחי אינו צומת הקלט, נבדוק למי יש מפתח קטן יותר ונעדכן בהתאם את currMin לפני הקפיצה לצומת הבא.
אם בתום הלולאה המפתח של הצומת המינימלי שווה למפתח של צומת הקלט נרצה להחזיר את צומת הקלט. (זאת מתוך העדפה אישית שלנו להצביע למינימום בעל הדרגה הכי גבוהה אם יש כזה).
סיבוכיות : O(logn) במקרה הגרוע. במקרה בו הקריאה מתבצעת על שורש מעצי הערמה, מספר העצים ששווה למספר האחים שלו, הוא לכל היותר לוגריתמי במספר הצמתים בערמה. לכן, נצטרך לעבור על לכל היותר logn אחים למציאת המינימלי מבניהם.

36. Private int numberOfTreesInArray(HeapNode[ ] arrayOfTrees) :

מהות : מתודת עזר אשר בהינתן מערך המצביע על צמתים ובחלק מהמקומות ייתכן ומצביע על null, נרצה להחזיר את מספר האינדקסים שמצביעים אכן על צומת. כלומר מספר המקומות התפוסים במערך.
נשתמש במתודה זו בסיום מיזוג שתי ערמות עליו נפרט בהמשך. נקרא למתודה על מערך המתחזק מצביעים לשורשי העצים של הערמה שהתקבלה, כאשר המטרה היא לעדכן בסיום את מספר העצים בערמה בשדה numOfTrees.
אופן הפעולה : המתודה מאתחלת מונה, עוברת בלולאת for על המערך ובכל איטרציה כאשר המיקום הנוכחי במערך אינו שווה null, מעלה את המונה ב-1.
סיבוכיות : O(logn) במקרה הגרוע. ננתח את סיבוכיות המתודה בהקשר שבו היא מופיעה בקוד, כאשר המערך מתחזק מצביעים לעצי הערמה אליה ממזגים ערימה נוספת. הגודל של המערך נקבע במתודה הממזגת ערמות והוא הדרגה המקסימלית מבין הדרגות הקיימות בשתי הערמות + 1. לכן, אם במקרה הגרוע בשתי הערמות יש n איברים, מספר העצים בערימה בודדת הוא לכל היותר logn, וגם לאחר המיזוג גודל המערך יכול לגדול לכל היותר במיקום אחד (כמו בחיבור מספרים בינאריים).

37. Private void connectRoots(HeapNode[ ] treeArray) :

מהות : מתודת עזר אשר בהינתן מערך המצביע על צמתים ובחלק מהמקומות ייתכן ומצביע על null, עוברת על המערך ומעדכנת את שדה ה-next של כל אחד מהצמתים להיות הצומת הבא במערך אחריו שאינו null. נשתמש במתודה זו בסיום מיזוג שתי ערמות עליו נפרט בהמשך במטרה לחבר בין שורשי העצים בערמה החדשה שהתקבלה.

אופן הפעולה : ראשית נרצה לסרוק את המערך מתחילתו ולמצוא את האיבר הראשון שאינו null, נעשה זאת באמצעות לולאת while אשר עוברת על המערך כל עוד לא הגענו לסוף והאיבר הנוכחי בכל פעם הוא null, וכמספר האיטרציות מוסיפה 1 לאינדקס של איבר ראשון זה. אם יצאנו מהלולאה והגענו לאיבר האחרון, נסיים את ריצת המתודה. אחרת, נשמור את הצומת שבאינדקס שמצאנו בתור previous ונעבור שוב על המערך החל מהמיקום שמצאנו + 1, כאשר בכל איטרציה אם האיבר הבא במערך אינו null, נדאג ש-previous יצביע עליו. לבסוף, עבור האיבר האחרון במערך אשר אינו null, נדאג שיצביע לצומת שמצאנו בהתחלה.
סיבוכיות : O(logn) במקרה הגרוע. ננתח את סיבוכיות המתודה בהקשר שבו היא מופיעה בקוד, לאחר מיזוג 2 ערמות בינומיות לערמה שעליה התבצעה הקריאה. אם מיזגנו ערמות בעלות n צמתים כל אחת, נקבל שהמערך המצביע אל שורשי הערימה הממוזגת הוא לכל היותר logn+1 (בדומה לחיבור מספרים בינומיים, מספר הביטים יכול לגדול בביט אחד נוסף). לכן, מספר האיטרציות חסום על ידי logn.

38. Public void meld(BinomialHeap heap2) :

מהות : המתודה ממזגת את הערמה עליה מתבצעת הקריאה עם ערימה נוספת heap2.

אופן פעולה : נחלק למקרים –
1. אם שתי הערמות ריקות (בדיקה מתבצעת על ידי ()empty סעיף 30), נסיים את ההרצה.
2. אם הערמה עליה נקראה המתודה אינה ריקה אך הערמה השנייה ריקה נסיים את ההרצה.
3. אם הערמה עליה התבצעה הקריאה ריקה והערמה אותה ממזגים אינה ריקה, נעדכן את כל שדות הערמה this להצביע על תוכן השדות של הערמה השנייה, תוך שימוש במתודות set-ים שכתבנו, ונסיים את ההרצה.
4. המקרה המעניין – שתי הערמות אינן ריקות;
נעדכן את שדה ה-size של הערמה הנוכחית להיות סכום הגדלים של שתי הערמות.
נשמור את הדרגה המקסימלית מבין שדות ה-last של שתי הערמות ובאמצעות arrayOfTrees (ראה 33), ניצור 2 מערכים שיתחזקו מצביעים אל שורשי עצי הערמות. המערכים יאותחלו לגודל הדרגה המקסימלית + 2. זאת מפניי שראשית נרצה שלכל דרגה יהיה אינדקס מתאים במערך. אם נסתכל על העצים בערמות כעל ביטים בייצוג בינארי של מספר, שכל עץ מדרגה i הוא ביט דלוק במקום ה-i בספרה, וחיבור שני מספרים בינאריים יכול להאריך את המחרוזת בביט אחד לכל היותר. לכן, נרצה אפילו אינדקס נוסף למקרה ונצטרך אותו. נעבור בלולאה על מערך הערמה השנייה וכל איטרציה נבצע לפי המקרים הבאים :
- אם באינדקס i המערך מצביע ל-null, כלומר אין עץ מדרגה i, נמשיך לאיטרציה הבאה.
- אחרת, אם המערך במיקום ה-i מצביע על עץ, וגם בערמה הראשונה אין עץ מדרגה i, נדאג שמערך הערמה הראשונה במיקום ה-i, יצביע על עץ זה.
- אחרת, בשתי הערמות יש עץ מדרגה i ; נבצע link (ראה 32) ונשמור את שורש העץ שהתקבל במערך עזר carry. נשמור את דרגת העץ שהתקבל ונרוץ בלולאת while עד ש-carry יתרוקן, ונבצע קריאות ל-link שנגיע למיקום במערך הערמה הראשונה שאינו מצביע על null, כלומר לא קיים בה עץ מהדרגה שהתקבלה.
לבסוף, נבצע קריאה על מערך הערמה הראשונה למתודת העזר connectRoots (ראה 37), נעדכן את מספר העצים בערמה באמצעות מתודת העזר numberOfTreesInArray (ראה 36), ואת המינימום נעדכן באמצעות מתודת העזר findNewMin (ראה 35). מאחר ואנו יודעים שבאחת הערמות היה עץ מהדרגה המקסימלית, אנו יודעים שהאיבר האחרון של הערמה החדשה שהתקבלה יכול להימצא או באינדקס האחרון במערך, או באחד שלפניו. נבדוק ונעדכן את last בהתאם.

סיבוכיות : O(logn). נסמן ב-n את מספר הצמתים הכולל בשתי הערמות, כלומר מספר הצמתים בערמה הממוזגת. לכן, בכל ערמה יש לכל היותר logn בינומיים (חסם עליון לא בהכרח הדוק). במקרים 1,2 ו-3 מתבצעת כמות קבועה של עבודה. במקרה 4 כמות העבודה המתבצעת שקולה לכמות העבודה המתבצעת בחיבור 2 מספרים בינאריים על ידי חיבור המחרוזות הבינאריות שלהן, כאשר לכל אחת לכל היותר logn ביטים דלוקים. העבודה שמתבצעת בפועל מלבד לולאת ה-for החיצונית היא מספר הקריאות למתודה link אך סך העבודה שהיא מבצעת במקרה זה היא קבועה. הסיבוכיות של חיבור הספרות וגרירת ה-carry לאיטרציות נוספות חסום על ידי זמן ריצה לוגריתמי ב-n ממש כפי שראינו בחיבור מספריים בינאריים. לסיום, מתודות העזר בהן אנו משתמשים לעדכון שדות הערמה פועלות כל אחת בסיבוכיות לוגריתמית ולכן לא משנות אסימפטוטית את הסיבוכיות.

39. **Public HeapItem insert(int key, String info)** :

מהות : מתודה המכניסה איבר בעל מפתח טבעי key לערמה עם מידע info מטיפוס מחרוזת, ומחזירה את איבר הערמה שנוצר עבורו.
אופן הפעולה : המתודה מאחלת צומת ו-item ושומרת בו את הערכים שקיבלה. לאחר מכן מאתחלת ערמה בינומית חדשה ששדות ה-min,last שלה מצביעים על הצומת שאותחל וכמות האיברים/עצים בה היא אחת. לבסוף תמזג את הערמה הבינומית החדשה אל הערמה עליה נקראה באמצעות meld (ראה 38), ותחזיר את ה-item החדש שיצרה.
סיבוכיות : O(logn). אתחול האובייקטים ועדכון השדות מתבצעים בסיבוכיות קבועה והקריאה ל-meld מתבצעת בסיבוכיות לוגריתמית בגודל הערמה.

40. **Public void deleteMin()** :

מהות : מתודה המוחקת את האיבר המינימלי בערמה.
אופן הפעולה : נחלק למקרים :
1. אם הערמה ריקה, נסיים את ההרצה.
2. אם גודל הערמה הוא 1, נעדכן את השדות min,last ל-null, ומספר הצמתים/עצים ל-0.
3. אחרת, אם הצומת המסומן למחיקה הוא השורש של העץ היחידי בערמה, נאפס את שדות הערמה לערמה ריקה. אחרת, נמצא את השורש של העץ שהצביע עליו באמצעות המתודה findPrev (ראה 34). אם המינימום גם היה last נעדכן אותו להיות Prev, ונעדכן את ה-next של Prev להיות ה-next של הצומת המסומן. נוריד את מספר העצים של הערמה ב-1, נעדכן את המינימום החדש בעזרת findNewMin() (ראה 35), ונחסר מגודל הערמה 2 בחזקת הדרגה של הצומת המסומן – מספר הצמתים בעץ זה שכעת חתכנו.
אם הצומת המסומן היה צומת מדרגה 0, נסיים את ההרצה. אחרת, נאתחל ערמה בינומית חדשה, ששדה ה-last שלה יצביע על הבן של הצומת המסומן, כמות העצים יהיה הדרגה שלו, כמות הצמתים 2 בחזקת הדרגה פחות 1, ואת המינימלי מבין הילדים נמצא על ידי מתודת העזר findNewMin() (ראה 35). לסיום, ננתק את הילד מאביו, הצומת המסומן, ונמזג את הערמה החדשה אל הערמה הנוכחית בעזרת meld (ראה 38).
סיבוכיות : O(logn). אתחול, עדכון שדות ופעולות אריתמטיות מתבצעות ב-O(1). המתודה findPrev נקראת לכל היותר פעם אחת, findNewMin לכל היותר פעמיים, ומכיוון וסיבוכיות זמן הריצה שלהן חסומה על ידי O(logn), הן לא משפיעות אסימפטוטית על הסיבוכיות. עיקר העבודה מתבצעת בקריאה ל-meld שהעלות שלה היא O(logn).

41. Public void decreaseKey(HeapItem item, int diff) :

מהות : מתודה מקבלת מצביע לאיבר הערמה ומספר טבעי. היא מפחיתה את המפתח של item ב-diff ומתקנת את הערמה כפי שלמדנו בכיתה.
אופן הפעולה : המתודה מעדכנת את המפתח של ה-item שקיבלה. אם item לא משויך לצומת בערמה, כלומר שדה ה-node שלו מצביע ל-null, אז נסיים את ההרצה. נשמור את הצומת המקושר ל-item וכל עוד ההורה שלו אינו null נעלה מעלה לכיוון השורש. אם המפתח של הצומת הנוכחי קטן מהצומת של ההורה שלו – נבצע ביניהם swap (ראה 18) ונמשיך לאיטרציה הבאה. אחרת, נעצור ונצא מהלולאה (אין צורך לעלות עד השורש). השימוש ב-swap נעשה במטרה לא לפגוע במצביעים חיצוניים של המשתמש, לכן אם יש צורך להחליף בין שני צמתים בערמה, נשאיר אותם במקומם ונחליף בין איברי הערמה המצביעים עליהם ושומרים בעצם את המידע הנחוץ עליהם ועל יחס הסדר ביניהם. לבסוף, נבדוק אם יש צורך לעדכן את המצביע למינימום.
סיבוכיות : O(logn). במקרה הגרוע בו ה-item מצביע לאיבר הנמצא בתחתית העץ בעל הדרגה המקסימלית בערמה ו-diff גדול מספיק כך שנצטרך לפעפע עד השורש, יידרשו logn איטרציות. זאת מפניי שבערמה המכילה n איברים, העץ בעל עומק מקסימלי הוא לכל היותר logn.

42. Public void delete(HeapItem item) :

מהות : מתודה מקבלת מצביע לאיבר הערמה ומוחקת אותו מהמבנה.
אופן הפעולה : אם ה-item שקיבלה לא משויך לאף צומת במבנה, כלומר שדה ה-node שלו מצביע ל-null, אז נסיים את ההרצה. אחרת, אם הוא מצביע לאיבר המינימלי בערמה, נבצע קריאה ל-deleteMin() (ראה 40) ונסיים את ההרצה. אחרת, נבצע קריאה ל- decreaseKey(item,diff) (ראה 41), כך שהמפתח של הצומת פחות diff יהיה קטן מהמינימום ב-1. כעת, נבצע קריאה ל-deleteMin() ובכך נסיים.
סיבוכיות : O(logn). עלות הקריאה ל-decreaseKey היא לכל היותר O(logn) וכך גם הקריאה ל-deleteMin().

חלק ניסויי/תיאורטי

מספר ניסוי	מספר סידורי i	זמן ריצה (מילישניות)	מספר החיבורים הכולל	מספר העצים בסיום	סכום דרגות הצמתים שמחקנו
ניסוי ראשון	1	4	723	5	0
	2	2	2182	4	0
	3	5	6555	5	0
	4	17	19675	7	0
	5	43	59040	8	0
ניסוי שני	1	2	3298	5	2939
	2	2	11532	4	10443
	3	8	39804	5	36529
	4	25	135227	7	125393
	5	104	450905	8	421389
ניסוי שלישי	1	1	723	5	697
	2	2	2182	5	2156
	3	4	6555	5	6529
	4	15	19675	5	19649
	5	57	59040	5	59014

ניתוח זמן ריצה :

ניסוי 1: זמן הריצה $\Theta(n)$.

חסם עליון: מבצעים n פעולות הכנסה לתוך ערימה בינומית ריקה. הכנסה לערמה בינומית שקולה לפעולת increment של מספר בינארי, ולכן הניתוח יהיה דומה לזה שראינו בתרגול. כדי להראות חסם עליון על זמן הריצה, נראה שהסיבוכיות אמורטייזד של הכנסה בסדרה של n הכנסות בלבד היא $O(1)$. נראה זאת באמצעות שיטת הבנק. על כל הכנסה נגדיר שנקבל 2 מטבעות לבנק. באמצעות מטבע אחד נשלם על הפעולה עצמה, ונשאר עם מטבע נוסף שבעזרתו נשלם על לינק עתידי.

נראה את האינווריאנטה הבאה: בכל רגע נתון: מספר המטבעות בבנק = מספר העצים בערמה.

נוכיח באינדוקציה על מספר העצים T:

בסיס T = 0: לא ביצענו הכנסה, ולכן יש 0 מטבעות בבנק כנדרש.

נניח כי עבור T עצים בערמה יש בבנק T מטבעות ונוכיח עבור ערימה עם T+1 עצים.

נוציא את העץ המינימלי מהערמה. נניח שדרגתו k. אז נקבל כי בערמה יש T עצים, ונבחין כי זו עדיין ערימה תקינה שכן העצים מסודרים לפי דרגות וכל דרגה נותרת ייחודית, ולכן מהנחת האינדוקציה יש לנו בבנק T מטבעות. נבצע הכנסה לערמה מתוך העץ ששלפנו ונראה איך מספר המטבעות משתנה.

נוסיף צומת בודד. מאחר שהעץ ששלפנו בעל הדרגה k היה העץ מהדרגה המינימלית, זה אומר או שהוא היה מדרגה 0, כלומר הצומת היחיד בעץ הוא זה שכרגע נכניס, או שהוא היה מדרגה 1 ומעלה, וזה אומר שלא היה עץ בערמה מדרגה 0. בכל מקרה, המשמעות היא שבעקבות ההכנסה לא נצטרך לבצע אף לינק. כמות המטבעות במאזן בבנק תהיה T+1 (שכן על הכנסת הצומת קיבלנו למאזן עוד מטבע בסך הכול) ואכן כמות העצים במבנה היא T+1.

מכאן הוכחנו את השמורה. כלומר המאזן בבנק יהיה אי שלילי בכל רגע נתון ולכן הראנו שהכנסה לערמה בינומית כחלק מסדרת *n* הכנסות לוקחת amortized O(1).

ולכן חסם עליון על זמן הריצה הכולל יהיה:

$$cost\ (op_1, \dots, op_n) \leq \#insertions * O(1) = n * O(1) = O(n)$$

<u>חסם תחתון:</u> נוכל לחשב באופן ישיר באופן דומה למונה הבינארי. כל הכנסה זה כמו העלאה ב-1 של המספר. נרצה לספור כמה פעמים שינינו כל ביט במחצית הראשונה של סדרת ההכנסות שכן זה סך כל העבודה שהשקענו בביצוע *n* הפעולות.

את הביט הראשון שינינו *n* פעמים. את הביט השני שינינו $\left\lfloor \frac{n}{2} \right\rfloor$ פעמים, ואת הביט ה-*i* שינינו $\left\lfloor \frac{n}{i} \right\rfloor$ פעמים. אם נסכום על כל ההכנסות עד ל-*n/2* נקבל זמן ריצה של:

$$\sum_{i=1}^{\frac{n}{2}} \frac{n}{i} = n + \frac{n}{2} + \frac{n}{4} + \cdots + \frac{n}{\frac{n}{2}} = n + \frac{n}{2} + \frac{n}{4} + \cdots + 2 \geq n = \ \Omega(n)$$

ניסוי 2: זמן הריצה הוא $\Theta(n \log n)$.

<u>חסם עליון:</u> על אף שסדרה של כל n הכנסות (גם רנדומלית) תתבצע בזמן לינארי, נראה כי זה לא נכון עבור סדרה של n/2 פעולות של מחיקת המינימום. חסם עליון אינו מהווה בעיה. ראינו בביתה כי כל פעולת מחיקת המינימום לוקחת במקרה הגרוע O(log n). ולכן סדרה של n/2 מחיקות המינימום ייקחו לנו לכל היותר: $\frac{n}{2} \cdot O(\log n) = O(n \log n)$

<u>חסם תחתון:</u> נותר להראות שגם סדרת המחיקות מתבצעת בזמן $\Omega(n \log n)$. רוב העבודה המתבצעת בפעולת DeleteMin היא בקריאה לפונקציה meld. שאר העבודה זה מחיקה של שורש בזמן קבוע ושינוי מספר קבוע של מצביעים. בביתה ראינו כי הפעולה meld שקולה לחיבור של שני מספרים בינאריים. נבחין כי אנחנו לא יודעים היכן נמצא המינימום בכל מחיקה, שכן ההכנסה התבצעה רנדומלית. במקרה הגרוע, המקרה שקול לחיבור של שני מספרים בינאריים, מסדר גודל לכל היותר n. כפי שראינו במבוא מורחב הסיבוכיות לחישוב זה לוקחת זמן $\Theta(\log n)$. כעת אם נתבונן על המחצית

הראשונה של סדרת המחיקות, גודל הערימה עבור כל i יהיה n-i כי מחקנו i פעמים את המינימום. אז נשקיע בכל שלב $\Theta(\log n - i)$ עבודה על meld. נסכום ונקבל:

$$\sum_{i=1}^{\frac{n}{4}} \log(n-i) = \sum_{i=1}^{\frac{n}{4}} \log n - \sum_{i=1}^{\frac{n}{4}} \log i \geq \frac{n}{4}\log n - \frac{n}{4}\log 1 = \frac{n}{4}\log n = \Omega(n\log n)$$

ניסוי 3: זמן הריצה $\Theta(n\log n)$.

<u>חסם עליון</u> זמן הריצה של DeleteMin כפי שראינו בניתוח סיבוכיות ניסוי 2 הוא $\Theta(\log n)$ בגלל פעולת meld. אנו מבצעים מחיקות עד שנשארים עם מספר קבוע של צמתים (31 צמתים ליתר דיוק). כלומר אנחנו מסירים כמעט את כל הצמתים בניסוי. ולכן מספר הפעמים שמבצעים מחיקה הוא מסדר גודל $O(n)$. על כן גם כאן זמן הריצה יהיה $O(n\log n)$.

<u>חסם תחתון:</u> זמן הריצה יהיה לפחות $\Omega(n\log n)$. סדרת המחיקות תתבצע בזמן $\Omega(n\log n)$ על ידי ניתוח זהה כפי שעשינו בחסם התחתון לסיבוכיות הזמן של ניסוי 2. כלומר נוכל להתבונן על מחיקה רק של n/4 הצמתים הראשונים. כל מחיקה תיקח זמן לוגריתמי במספר הצמתים באותו שלב והסכום ייתן לנו את החסם התחתון שראינו קודם לכן בניסוי השני כאמור.

הקשר במשוואה בין מספר החיבורים הכולל, מספר העצים בסיום וסכום דרגות הצמתים שנמחקו:

נסמן את הסימונים הבאים:

#links – מספר החיבורים הכולל (מספר הפעמים שמבצעים Link בין שני עצים שווי דרגה).

#trees – מספר העצים בסיום.

#deletedRanks – סכום דרגות הצמתים שנמחקו.

#deletedNodes – מספר הצמתים שנמחקו

אם אנו מתחילים עם n צמתים נוכל לקבל את המשוואה הבאה:

$$\#trees = n - \#links + \#deletedRanks - \#deletedNodes$$

נבחין כי $BinomialHeap.size = n - \#deletedNodes$. נציב במשוואה:

$$\#trees = BinomialHeap\_size - \#links + \#deletedRanks$$

ההסבר למשוואה שהתקבלה:

בהינתן n צמתים שאנו רוצים להכניס לערמה, כל פעולת לינק מורידה את מספר העצים בערמה ב-1 (מכיוון ששני עצים מתאחדים לעץ אחד). למשל בהינתן שני עצים מדרגה k אז קריאה לפעולת link תהפוך אותם לעץ יחיד מדרגה k+1 ולכן אם היו t עצים בערמה, כעת יהיו t-1. כל פעולת מחיקה מעלה את מספר העצים במספר ששווה לדרגה של הצומת שנמחק (כי כל ילד של הצומת שנמחק הופך לעץ נפרד). למשל אם מחקנו שורש של עץ מדרגה k אז חשפנו את k בניו המהווים ערמות בינומיות שיתווספו לערמה.

כלומר בהינתן n הכנסות לערמה, מספר העצים יהיה מספר הצמתים פחות מספר הקריאות ל-link. עבור שני צמתים מדרגה 0 נבצע בהתחלה לינק אחד ונפחית את כמות העצים בערמה ב-1. לאחר מכן, אם יש לנו שני עצים מדרגה 1 לינק ביניהם שוב יפחית את מספר הצמתים ב-1. כך, לאחר

ההכנסות מספר העצים יהיה n-#links. למעשה, אפילו ניתן היה להראות כי מספר הלינקים שהתבצעו שווה למספר הקשתות הכולל של עצי הערימה.

כעת נראה איך מספר העצים משתנה בעת מחיקה. אם לינק, מפחית את כמות העצים בערמה, כל מחיקה מוסיפה בחזרה עצים אליה. ליתר דיוק, כאמור, כל מחיקה של מינימום בעץ מדרגה k תוסיף לערמה עוד k עצים, כמספר הבנים של הצומת. נבחין כי פעולת המחיקה תוסיף לנו עוד לינקים לאחר הקריאה ל-meld, שכן ייתכן שהעצים שחשפנו היו מדרגה שכבר הופיעה בערמה. כעת כל לינק שנבצע יפחית את מספר העצים בערמה ב-1 מאותו הסבר שהראינו קודם. נחשיב את מספר הלינקים של DeleteMin בתוך מספר הלינקים הכולל: #links. הדבר האחרון שנותר לעשות הוא להסיר את הצומת שנמחק מהחישוב במשוואה. זאת מאחר שמהרגע שנפטרנו מהמינימום הוא כבר לא חלק מסך הצמתים בערמה ולכן לא יתחלק בין כל העצים שהמשוואה מחשבת את מספרם. נישאר עם המשוואה:

#trees = n - #kinks #deltedRanks - #deletedNodes

כעת נוכל לראות שאכן המדידות שלנו מסתדרות עם המשוואה הזו:

מספר הניסוי	מספר סידורי i	מספר העצים - תוצאות הניסוי (מדידה בעזרת הפונקציה getNumTrees())	מספר העצים - תוצאות תיאורטיות (מדידה ידנית בעזרת המשוואה שמצאנו)
ניסוי ראשון	1	5	3^6-1-723+0-0 = 72-723 = 5
	2	4	3^7-1-2182+0-0 = 2186-2182 = 4
	3	5	3^8-1-6555+0-0 = 6560-6555 = 5
	4	7	3^9-1-19675+0-0 = 19682-19675 = 7
	5	8	3^10-1-59040+0-0 = 59048-59040 = 8
ניסוי שני	1	5	3^6-1-3298+2939-(3^6/1)/2 = 728-3298+2939-364 = 5
	2	4	3^7-1-11532+10443-(3^7-1)/2 = 4
	3	5	3^8-1-39804+36529- (3^8-1)/2 = 5
	4	7	3^9-1-135227+125393-(3^9-1)/2 = 7
	5	8	3^10-1-450905+421389-(3^10-1)/2 = 8
ניסוי שלישי	1	5	3^6-1-723+697-(3^6-1-2^5+1) = 31-723+697 = 5
	2	5	3^7-1-2182+2156-(3^7-1-2^5+1) = 31-2182+2156 = 5
	3	5	3^8-1-6555+6529-(3^8-1-2^5+1) = 31-6555+6529 = 5
	4	5	3^9-1-19675+19649-(3^9-1-2^5+1) = 31-19675+19549 = 5
	5	5	3^10-1-59040+59014-(3610-1-2^5+1)

סכום דרגות הצמתים שנמחקו בניסוי השלישי:

נעזר במשוואה שהגדרנו קודם לכן ונביע את סכום דרגות הצמתים שנמחקו בעזרת יתר המשתנים. כך נראה תלות ישירה של סכום דרגות הצמתים שנמחקו במספר הלינקים:

$$\#deletedRanks = \#trees + \#links - (n - \#deletedNodes)$$
$$= \#trees + \#links - \#\ nodes\ left\ in\ heap$$

לפי המדידות נבחין כי לכל i מספר העצים בסיום ומספר הצמתים בסיום נותרים קבועים: 5 ו-31 בהדרגה. אם נסביר מדוע זה קורה, נוכל להסיק כי סכום דרגות הצמתים שנמחקו תלוי באופן ישיר במספר החיבורים שנעשו. ואכן זה יסתדר עם המדידות כי עבור כל ניסוי מספר החיבורים שנעשו בערמה פחות סכום דרגות הצמתים שנמחקו יוצא קבוע ושווה למספר הצמתים בערמה פחות מספר העצים: 26 = 5-31. משם ארצה להגיע למסקנה נוספת: מספר הלינקים שקול בניסוי השלישי לסכום דרגות הצמתים שנמחקו. כתוצאה ממסקנה זו נוכל אף לדעת כי בניסוי השלישי, עבור כל i, סדרה של n הכנסות ולאחריה סדרה של n מחיקת מינימום תוביל בהכרח לעץ ריק, שכן נראה שמחיקה של עץ מדרגה k מוסיפה לסכום דרגות הצמתים שנמחקו k ובכל "מקזזת" את k הלינקים שיצרו את העץ על ידי ההכנסה.

ראשית, נבחין כי בכל פעולת insert יש רק לינקים, ובבירור סכום דרגות הצמתים שנמחקו לא משתנה במהלכה. בניסוי השלישי מאחר שאנו מכניסים את האיברים בסדר הפוך נקבל ערימה שבה האיברים הקטנים "יתנקזו" בתחילת הערימה. על כן, כדי למחוק את המינימום בכל שלב ניגש לעץ הראשון בערמה ונמחק ממנו את המינימום (השורש כמובן הוא המינימום מהתכונה של עץ בינומי שמפתח של הורה קטן שווה למפתח של ילד). נניח שדרגת העץ המינימלי היא m. המשמעות היא שלעץ זה יש m בנים בדרגות 0,1,...,m-1. כלומר מהרגע שמחקנו את השורש הוספנו לערמה עוד m עצים כל אחד מדרגה שונה. חשוב להבחין כי בוודאות לא נצטרך לבצע לינקים בשלב זה. הסיבה לכך טמונה בייחודיות של דרגות הערימה הבינומית. מאחר שהגישה למינימום בכל שלב תתבצע על האיבר הראשן בערמה, אנו יודעים בוודאות שכל העצים אחריו יהיו מדרגה גדולה יותר (כי אסור שניים מאותה דרגה, וסדר הדרגות של העצים הוא עולה). ולכן אם כעת דרגת העץ המינימלי היא m, כל העצים שאחרי הם מדרגה של לפחות m+1. כלומר מחיקת המינימום מוסיפה לי בכל שלב m לסכום הדרגות אך לא מוסיפה אף לא לינק אחד. ניתן להסיק כי על כל מחיקה של עץ מדרגה k "יתקזזו" לנו k לינקים שנעשו בהכנסה (בין השורש לכל אחד מ-k בניו) ולכן אם נעשה סדרה של n הכנסות בסדר הפוך שלאחריה סדרה של n מחיקות מינימום נקבל כי #deletedRank = #links. במילים אחרות, מספר הלינקים הוא בדיוק מספר המחיקות שנבצע במקרה זה.

אולם במקרה שלנו, עבור כל i בניסוי, אנו נשארים עם 31 צמתים ו-5 עצים. הסיבה לכך היא שמספר הקשתות בין צמתים בערמה שווה למספר הלינקים בערמה, ובכל עץ מדרגה k יש $2^k - 1$ קשתות. ולכן במקרה שלנו ישנם 31 צמתים (מה ששקול למספר הבינארי 11111) אשר מתפזרים ב-5 עצים, שדרגותיהם הן כפי שניתן לראות במספר הבינארי המייצג את הערימה בסיום הניסוי: 4,3,2,1,0. ולכן סכום הקשתות שלנו, כלומר מספר הלינקים שנשארו לנו "ללא פגע" לאחר מחיקת המינימום בניסוי השלישי הם:

$$\#links\ left\ in\ tree = \sum_{i=0}^{4} \#edges(B_i) = \sum_{i=1}^{4} 2^i - 1 = (2^0 - 1) + (2^1 - 1) + (2^2 - 1) + (2^3 - 1) + (2^4 - 1) = 0 + 1 + 3 + 7 + 15 = 26$$

ואכן כפי שראינו קודם לכן הקשר בניסוי השלישי בין סכום דרגות הצמתים שנמחקו ומספר הלינקים הוא קשר הלינארי הבא:

$$\#deletedRanks \ = \#trees + \#links - \#\ nodes\ left\ in\ heap \rightarrow$$
$$\#deletedRanks = \ 5 + \#links - 31 = \#links - 26$$