**מסמך תיעוד- ליהוא צור ויובל רמות**

**public static class HeapNode**

המחלקה מממשת צומת של ערמת פיבונאצ'י כמו שנלמד בכיתה. כל צומת שומר מפתח (מספר שלם) ומחרוזת של מידע, וגם שדות נוספים שעוזרים לבנות את הערמה לפי התכונות שנלמדו (פירוט בהמשך).

**שדות:**

public int key- המפתח של הצומת

protected int rank- הדרגה של הצומת, כלומר כמות הבנים הישירים שלו.

protected Boolean mark- שדה בוליאני ששומר האם הצומת מסומן או לא.

protected HeapNode kmin\_help- שדה עזר עבור הפונקציה kmin, ישומש רק בה ותפקידו להחזיק מצביע לצומת המתאים בערמה המקורית.

protected HeapNode child- מצביע לבנו השמאלי ביותר של הצומת (הראשון ברשימת הבנים).

protected HeapNode prev- מצביע לצומת הקודם ברשימת הבנים הנוכחית שהצומת שייך אליה (אם הוא היחיד, יצביע לעצמו באופן מעגלי).

protected HeapNode parent- מצביע לאביו הישיר של הצומת (null אם הצומת מהווה שורש באחד העצים בערמה).

protected HeapNode next- מצביע לצומת הבא ברשימת הבנים הנוכחית שהצומת שייך אליה (אם הוא היחיד, יצביע לעצמו באופן מעגלי).

**בנאים:**

public HeapNode(int key)- מאתחל את המפתח לפי הקלט, את mark ל false ואת הדרגה ל-0. שאר השדות (שדות שמחזיקים צמתים) מאותחלים לnull באופן אוטומטי (גם את הדרגה והmark יכולנו לאתחל אוטומטית, אך אין זה משנה לצורך המימוש).

**פונקציות:**

public int getKey()- הפונקציה מחזירה את שדה המפתח של הצומת, פעולה שלוקחת זמן של O(1) במקרה הגרוע

public int getRank()- הפונקציה מחזירה את שדה הדרגה של הצומת, פעולה שלוקחת זמן של O(1) במקרה הגרוע.

public void setRank(int rank)- הפונקציה מעדכנת את שדה הדרגה של הצומת, פעולה שלוקחת זמן של O(1) במקרה הגרוע.

public boolean isMark()- הפונקציה מחזירה את שדה הmark של הצומת, פעולה שלוקחת זמן O(1) במקרה הגרוע..

public void setMark(Boolean mark)- הפונקציה מעדכנת את שדה ה mark של הפונקציה, פעולה שלוקחת זמן O(1) במקרה הגרוע.

public HeapNode getChild()- הפונקציה מחזירה את שדה הchild של הצומת, פעולה שלוקחת זמן O(1) במקרה הגרוע.

public void setChild(HeapNode child)- הפונקציה מעדכנת את שדה ה child של הצומת, פעולה שלוקחת זמן O(1) במקרה הגרוע.

public HeapNode getPrev()- הפונקציה מחזירה את שדה הprev של הפונקציה, פעולה שלוקחת זמן O(1) במקרה הגרוע.

public void setPrev(HeapNode prev)- הפונקציה מעדכנת את שדה הprev של הפונקציה, פעולה שלוקחת זמן O(1) במקרה הגרוע.

public HeapNode getParent()- הפונקציה מחזירה את שדה הparent של הפונקציה, פעולה שלוקחת זמן O(1) במקרה הגרוע.

public void setParent(HeapNode parent)- הפונקציה מעדכנת את שדה הparent של הפונקציה, פעולה שלוקחת זמן O(1) במקרה הגרוע.

public HeapNode getNext()- הפונקציה מחזירה את שדה הnext של הצומת, פעולה שלוקחת זמן O(1) במקרה הגרוע וגם.

public void setNext(HeapNode next)- הפונקציה מעדכנת את שדה הnext של הצומת, פעולה שלוקחת זמן O(1) במקרה הגרוע.

public void setKey(int key)- הפונקציה מעדכנת את שדה המפתח של הצומת, פעולה שלוקחת זמן O(1) במקרה הגרוע.

public Boolean getMarked()- הפונקציה מחזירה את שדה הmark של הפונקציה, פעולה שלוקחת זמן O(1) במקרה הגרוע וגם.

**public class FibonacciHeap**

המחלקה מממשת ערמת פיבונאצ'י כפי שלמדנו בכיתה, מבנה נתונים אשר מאפשר מציאת מינימום וביצוע decreaseKey ביעילות. המימוש מתייחס לכמה הבהרות ספציפיות מטעמי אחידות המימוש בין הגשות של סטודנטים שונים, כפי שמתואר בדף ההנחיות.

**שדות סטטיים (לא תלויי מופע):**

protected static int links- שדה ששומר את כמות החיבורים שבוצעו במהלך קונסולודציה, במהלך ריצת התוכנית כולה.

protected static int cuts- שדה ששומר את כמות החיתוכים שבוצעו במהלך cascading cuts, במהלך ריצת התוכנית כולה.

**שדות מופע:**

protected HeapNode first- מצביע לצומת שהוא השורש הראשון בערמה (לאו דווקא הצומת המינימלי בערמה!).

protected HeapNode min- מצביע לצומת המינימלי בערמה. בפרט, הוא שורש, ולא בהכרח הfirst של הערמה.

protected int size- שדה ששומר את כמות הצמתים הכוללת בערמה (בכל העצים שיש בה יחדיו).

protected int trees- שדה ששומר את כמות העצים (כמות השורשים השונים) בערמה.

protected int marked- שדה ששומר את כמות הצמתים המסומנים בערמה כולה.

**בנאים:**

למחלקה יש בנאי ריק, אשר מאתחל את כל השדות באופן דיפולטי, שדות HeapNode מאותחלים לnull, שדות int מאותחלים לאפס.

**פונקציות:**

public Boolean isEmpty()- הפונקציה מחזירה true אם ורק אם הערמה ריקה. היא מבצעת זאת על ידי בדיקה האם השדה first מאותחל לnull – אם כן יוחזר true ואחרת false, פעולה שלוקחת זמן O(1) במקרה הגרוע.

protected void resetHeap()- פונקציית עזר אשר מאתחלת את שדות המופע בערמה – שדות הint מאותחלים ל-0 ושדות הצמתים מאותחלים לnull, פעולה שלוקחת זמן O(1) במקרה הגרוע מכיוון שאנו רק משנים שדות.

protected void connect2Nodes(HeapNode left, HeapNode right)- פונקציית עזר אשר מחברת בין 2 צמתים. הnext של Left נהיה right והprev של right נהיה left. נשים לב שהפעולה אינה סימטרית מבחינת שינויי השדות (אם נפעיל אותה עם 2 הצמתים בסדר הפוך, לכל אחד יתעדכן שדה שונה). בכל פעם שנשתמש בcononect2Nodes סדר ההכנסה של הקלט לפונקציה הוא לפי הסדר במשפט- הצומת הראשון הוא left והשני right. סיבוכיות הפעולה היא O(1) במקרה הגרוע כי רק שינינו שני שדות.

public HeapNode insert(int key)- הפונקציה בונה צומת חדש לפי מפתח הקלט ומכניסה אותו לראש רשימת העצים, כלומר הוא נהיה first החדש בסוף הפעולה. הפונקציה משתמשת בבנאי של הצומת כדי לבנות את הצומת, מוסיפה 1 לsize של העץ, משתמשת בפונקציית העזר insertNode אשר מבצעת את ההכנסה בפועל, ולאחר מכן מחזירה את הצומת החדש שהוכנס כפלט.

סיבוכיות הפעולה היא במקרה הגרוע O(1). הפונקציה מבצעת פעולות בזמן קבוע חוץ מהקריאה לפונקציית העזר, אך גם פונקציית העזר פועלת בזמן קבוע (הסבר בהמשך) ולכן סה"כ O(1) במקרה הגרוע.

protected void insertNode()- פונקציית עזר אשר מבצעת הכנסה בפועל של הצומת לערמה. הפונקציה מוסיפה 1 לשדה trees של הערמה. לאחר מכן בודקת אם הערמה ריקה בעזרת isEmpty. אם הערמה ריקה, שדה המינימום של הערמה משתנה מnull לצומת החדש, מחברים את הצומת לעצמו באופן מעגלי בעזרת connect2Nodes ואז first עובר להצביע לצומת שלנו ומסתיימת פעולת הפונקציה. אחרת, הערמה לא הייתה ריקה, ואם המפתח של הצומת החדש קטן מהמפתח של המינימום הנוכחי בערמה, מעדכנים את המינימום להיות הצומת החדש. לאחר מכן, מחברים בין הצומת החדש לבין first בעזרת connect2Nodes ולאחר מכן מחברים בין הprev של first לבין הצומת בעזרת connect2Nodes. לאחר מכן בכל מקרה נשנה את first להצביע לצומת החדש ונסיים את הפעולה.

סיבוכיות הפעולה היא O(1) במקרה הגרוע, מכיוון שמתבצעות רק מספר חסום לכל קלט של פעולות גישה ושינוי שדות ושל קריאות לפונקציה connect2Nodes שפועלת בO(1).

public void deleteMin()- הפונקציה מתחילה בהורדת השדה size ב-1. לאחר מכן, אם size החדש הוא -1 או 0, קוראים לפונקציית העזר resetHeap כדי לאתחל השדות ומסתיימת פעולת הפונקציה. אחרת, הערמה אחרי המחיקה לא תהיה ריקה. אם first=min :

אם לצומת min יש ילדים, first עובר להצביע על הבן הראשון (השמאלי) של min.

אחרת, אין לmin ילדים וfirst עובר להצביע על הnext של min.

כעת, אם לmin אין ילדים, מחברים בעזרת connect2Nodes בין הprev של min לבין הnext של min (כדי לדלג עליו ובכך להסיר אותו מהערמה).

אחרת, לmin יש ילדים. מסמנים בnode את הבן השמאלי של min ורצים בלולאת do while עד שnode חוזר להיות הבן השמאלי של min. בגוף הלולאה נבדוק אם הצומת הנוכחי מסומן, אם כן נוריד את שדה marked ב1 ונשנה את שדה הmark שלו לfalse. נשנה בכל מקרה את שדה ההורה של הצומת לnull ואז נעבור לצומת הבא על ידי next עבור האיטרציה הבאה של הלולאה.

כעת, לאחר הלולאה, אם הnext של min הוא עצמו, כלומר הוא השורש היחיד, נעביר את first להצביע על בנו הראשון (השמאלי).

אחרת, נבצע connect2Nodes בין הprev של child לבין הnext של min ולאחר מכן נבצע connect2Nodes בין הprev של min לבין child.

לבסוף, בכל מקרה בו הערמה לא הייתה ריקה אחרי ההורדה, נקרא לפונקציה consolidateעל הערמה, ונסיים את פעולת הפונקציה.

הסיבוכיות היא O(log(n)) באמורטייז ו O(n) במקרה הגרוע. מלבד בדיקה ושינוי שדות וקריאות לconnect2Nodes אשר כולן יחד מתבצעות בזמן קבוע, הפעולות היקרות יותר בפונקציה הן לולאת ה do while ופעולת הקונסולידציה. הלולאה עולה O(log(n)) במקרה הגרוע, כי למינימום יש לכל היותר O(log(n)) בנים כמו שראינו בהרצאה (לכל היותר קבוע כפול log(n) בנים לכל היותר, ומה שעזר לנו לשמור על זה הוא מנגנון הcascading cuts, אשר לא מאפשר לצומת לאבד יותר מבן אחד מבלי להפוך לשורש חדש בעצמו בעץ וכך אנו שומרים פחות או יותר על תכונת הlog(n) ילדים לכל שורש כמו בעץ בינומי מושלם). על כל בן אנו מבצעים פעולות בזמן קבוע O(1) במהלך הלולאה. פעולת הקונסולידציה עולה O(log(n)) באמורטייז ו O(n) במקרה הגרוע (הסבר בהמשך) ולכן הסיבוכיות של הלולאה נבלעת בזה בכל מקרה, ונקבל שהסיבוכיות של deleteMin היא O(log(n)) באמורטייז וO(n) במקרה הגרוע.

protected void consolidate()- הפעולה מבצעת קונסולידציה על הערמה כמו שראינו בהרצאה, ובסופה הערמה היא ערמת פיבונאצ'י המכילה ערמות עם עץ אחד לכל היותר מכל דרגה. ראשית, מאתחלים מערך בשם ranks של צמתים בגודל הערך התחתון של (פעמיים לוג מספר הצמתים ועוד 2). משנים את שדה הnext של first.prev לnull ובכך הורסים את מעגליות הרשימה, אך אין זה משנה מכיוון שבסוף הפעולה אנו בונים את הערימה מחדש. כעת מתחילים לרוץ בלולאה מfirst ועד שמגיעים לnull. קודם כל שומרים מצביע לnext של הצומת הנוכחי בלולאה ושומרים את הדרגה שלו במשתנה k.

רצים בלולאה פנימית: כל עוד ranks[k] אינו null, ובגוף הלולאה הפנימית קוראים לפונקציה link עם הצומת הנוכחי ועם ranks[k] אשר מחברת בין שני העצים ששני הצמתים הללו שורשים שלהם. לאחר מכן משנים את ranks[k] ל null כי חיברנו שני עצים בדרגה הזו ורוקנו את המקום הזה במערך. לאחר מכן נקדם את k ב1 ונסיים את גוף הלולאה הפנימית.

אחריה, נשים את הצומת הנוכחי בתוך ranks[k] ולאחר מכן הצומת הנוכחי משתנה לnext שלו ששמרנו בהתחלה, לצורך האיטרציה הבאה של הלולאה החיצונית. כעת מסתיימת גם הלולאה החיצונית.

אחריה, נרוקן את הערמה ונבנה אותה מחדש: נשים את first להצביע לnull ונאפס את trees. נרוץ בלולאת for על האינדקסים של ranks מהסוף להתחלה, ובכל איטרציה , אם יש עץ במערך באינדקס הנוכחי נשתמש בפונקציית העזר insertNode כדי להכניס אותו לערמה. לאחר סיום הלולאה קיבלנו את הערמה החדשה שבנינו מהעצים שנוצרו או נשארו אחרי הקונסולידציה.

סיבוכיות הפעולה היא O(log(n)) באמורטייז ו O(n) במקרה הגרוע.

המקרה הגרוע מתרחש בין היתר בפעולת קונסולידציה אשר מתרחשת אחרי סדרה של O(n) הכנסות רצופות לערמה ריקה. הסיבה לכך היא שבInsert כל צומת נכנס לעץ בתור עץ בינומי מדרגה 0, ובשאר הפונקציות אנחנו לא מקטינים כלל את כמות העצים בשום מצב, ולכן בהכרח בקריאה הזו יש O(n) עצים בהתחלה עליהם אנו מבצעים את הקונסולידציה (לא יכולים כמובן להיות יותר מכיוון שn היא כמות הצמתים בעץ ולכן לכל היותר יש n שורשים). במקרה זה הקונסולידציה אכן עולה O(n) כמו שראינו בהרצאה, מכיוון ואנו עוברים בלולאה החיצונית על n שורשים ומבצעים סך הכל בכל השלבים ביחד n-1 לינקים (כל אחד עולה O(1) ):

לכן במקרה הגרוע העלות הכוללת היא אכן O(n) מכיוון שסך הכל ביצענו n לינקים בO(1) וגם n פעמים פעולות נוספות בלולאה בO(1).

עם זאת, כמו שראינו בכיתה, באמורטייז הסיבוכיות היא O(log(n)). זה נובע מכך שכאמור, לאחר הקונסולידציה, נקבל ערמה המכילה לכל היותר עץ אחד מכל דרגה, ולכן דרושות פעולות רבות עד שנגיע למצב "רע" כמו המתואר למעלה, או קרוב לכך. כמו שאמרנו בכיתה, מכיוון שכל קונסולידציה מתקנת את העץ למצב “טוב” כזה, נקבל באמורטייז O(log(n)) .

נסביר זאת פורמלית: כפי שראינו בהרצאה, עלות הפעולה היא כאשר כמות הלינקים במהלך הטיפול בעץ הi (מוסיפים 1 עבור ההכנסה עצמה). נסמן: k -דרגת המינימום שנמחק , L -מספר הלינקים שבוצעו סך הכל, - מספר העצים לפני הפעולה. מתקיים:

כאשר המעבר השני נכון מכיוון שמספר הלינקים שבוצעו לא יכול להיות יותר ממספר העצים הכולל (אחרי הוספת הבנים של המינימום) כי בכל שלב אם יתבצע לינק, יורד מספר העצים ב-1 כפי שאמרנו בהרצאה. המעבר השלישי נכון מכיוון שk חסום על ידי 2log(n) כפי שראינו בהרצאה.

נסתכל על פונקציית הפוטנציאל כפי שהוגדרה בהרצאה (ובתרגיל הנוכחי). נסמן ב- מספר העצים אחרי הפעולה כאשר בפועל זה (בערך) log(n). נשים לב שמספר המסומנים רק יכול לקטון, כי אנחנו לא מבצעים חיתוכים אלא רק הופכים צמתים שעלולים להיות מסומנים, לשורשים, ולכן הפוטנציאל יכול רק אולי לרדת במובן הmark ולכן אפשר להתעלם מהם.

נעיר כי שאר הפעולות בפונקציה נבלעות בסיבוכיות הזו : בניית הערמה עולה תמיד כגודל המערך ranks ולכן תמיד O(log(n)) (בפנים הלולאה מבצעים insertNode שהיא O(1)), ושאר הפעולות הן פעולות בזמן קבוע.

protected HeapNode link(HeapNode parent, HeapNode child)- הפונקציה מחברת בין שורשי 2 עצים בערמה לבניית עץ גדול יותר. ראשית, הפונקציה מגדילה ב1 את links. אם המפתח של child קטן משל parent, נבצע swap בין המצביעים שלהם. כעת, מובטח לנו שהמפתח של parent קטן יותר ולכן נוכל לחבר לו את child כבן ולשמור על הכללים. ההורה של child נהיה parent,וchild נהיה לא מסומן. כעת, אם לparent אין ילדים, השדה child שלו נהיה הצומת child, ונבצע connect2Nodes בין child לעצמו ליצירת רשימה מעגלית. אחרת, לparent כבר היו ילדים קודם, נבצע connect2Nodes בין הבן האחרון (הימני) של parent לבין child ולאחר מכן connect2Nodes בין child לבין הבן הראשון (השמאלי) של parent, ולבסוף נשנה את השדה child של parent להיות הצומת child. בסוף הפונקציה, בלי קשר אם היו לparent בנים קודם או לא, נגדיל את הדרגה של parent ב1 מכיוון שהוספנו לו בן, ולבסוף נחזיר את parent (שורש העץ החדש שנבנה).

הסיבוכיות היא O(1) במקרה הגרוע, מכיוון שביצענו מספר חסום ולא תלוי בקלט של פעולות על שדות ופעולות connect2Nodes כאשר כל הפעולות הללו הן בO(1).

public HeapNode findMin()- הפונקציה מחזירה את שדה המינימום של הערמה, הסיבוכיות היא O(1) במקרה הגרוע.

public void meld(FibonacciHeap heap2)- הפונקציה מחברת בין 2 ערמות פיבונאצ'י (מאחדת בין ערמת הקלט לערמה עליה מופעלת הפונקציה). אם ערמת הקלט ריקה, לא עושים דבר. אחרת, אם הערמה הנוכחית ריקה, מאתחלים את השדות שלה (first,marked,size,trees,min) להיות בדיוק השדות של ערמת הקלט, ולאחר מכן מסתיימת פעולת הפונקציה. אחרת, שתי הערמות אינן ריקות. מחברים בעזרת connect2Nodes בין השורש האחרון בערמה הנוכחית לשורש הראשון בערמת הקלט, ומחברים באותו אופן בין השורש האחרון בערמת הקלט לשורש הראשון בערמה הנוכחית (סה"כ גם חיברנו את ההצבעות וגם שימרנו את המעגליות של רשימת השורשים). לאחר מכן, מגדילים את הsize של הערמה הנוכחית בsize של ערמת הקלט, מגדילים את trees שלה בtrees של ערמת הקלט, וכנ"ל לגבי marked. לבסוף, המינימום של הערמה הנוכחית נבחר להיות השורש עם המפתח המינימלי בין המינימום המקורי של הערמה הנוכחית לבין המינימום של ערמת הקלט.

סיבוכיות הפעולה היא O(1) במקרה הגרוע (כפי שראינו בהרצאה), מכיוון שאנו מבצעים כמות חסומה לכל קלט של שינויי שדות ושימושים בפונקציה connect2Nodes שפועלת בO(1).

public int size()- הפונקציה מחזירה את כמות הצמתים בערמה באמצעות גישה ישירה לשדה size, ולכן הסיבוכיות היא O(1) במקרה הגרוע.

public int[] countersRep()- הפונקציה מחזירה מערך של שלמים כך שבכל אינדקס i במערך נמצאת כמות העצים מסדר i הקיימים בערמה (במקרה שהערמה ריקה יוחזר מערך ריק). אם הערמה ריקה, כאמור נחזיר מערך ריק. אחרת, נעבור על רשימת השורשים של הערמה באופן סדרתי בלולאה (עד שנחזור חזרה לשורש הראשון) כדי להשיג את הדרגה המקסימלית בערמה. לאחר מכן, נאתחל את מערך הפלט כמערך ריק בגודל הדרגה שמצאנו פלוס 1, כדי שנוכל לשמור counters עבור דרגות בין 0 לבין המקסימום כולל. כעת, נעבור שוב בלולאה על שורשי הערמה (שוב, מהשורש הראשון ועד שנחזור אליו חזרה) ובכל איטרציה נחלץ את דרגת השורש מהשדה rank שלו ונוסיף 1 לערך השמור באינדקס הדרגה הזו במערך הפלט. בסיום הלולאה, נקבל את המערך המבוקש, ונחזיר אותו.

סיבוכיות הפעולה היא O(n) במקרה הגרוע. יש לנו 2 לולאות אשר עוברות על כל השורשים ומבצעות בכל איטרציה עבודה בO(1), ולכן הסיבוכיות תלויה באופן לינארי בכמות השורשים בערמה. לכן המקרה הגרוע יתרחש כאשר כמות השורשים היא מקסימלית, שזה כאשר כל הצמתים בערמה הם שורשים, מצב שמתקבל לדוגמא כאשר נקרא לפונקציה לאחר ביצוע סדרת פעולות המכילה רק n הכנסות על ערמה ריקה (ללא פעולות נוספות). במצב כזה יש בערמה n שורשים (וגם n צמתים) ולכן כל אחת מ2 הלולאות תיקח O(n) זמן, ולכן כאמור סיבוכיות הפעולה תהיה O(n).

public void delete(HeapNode x)- הפונקציה מוחקת את הצומת x מהערמה(מקבלים הצבעה ישירה אליו בקלט). אם הצומת x אינו צומת המינימום של הערמה (בדיקה מול השדה), נבצע decreaseKey לx בz כאשר z=x.getKey()-this.min.getKey()+1. נשים לב שz חיובי ולכן עומד בדרישת הקלט עבור decreaseKey. לאחר מכן (בלי קשר אם x היה המינימום או לא), x כעת נהיה המינימום ונבצע deleteMin על הערמה בכדי למחוק אותו.

סיבוכיות הפעולה היא O(log(n)) באמורטייז וO(n) במקרה הגרוע, כמו הסיבוכיות של deleteMin. הסיבה לכך היא שהסיבוכיות של decreaseKey כפי שנראה בהמשך היא O(n) במקרה הגרוע וO(1) באמורטייז, ולכן גם אם יתרחש decreaseKey, הסיבוכיות תיבלע בזו של deleteMin אשר יתרחש בכל מקרה.

public void decreaseKey(HeapNode x, int delta)- הפונקציה מקטינה את המפתח של הצומת x בדלתא (כאשר דלתא חיובית) ומשתמשת בcuts וcascading cuts כפי שלמדנו בהרצאה כדי לייעל את הפעולות בערמה באמורטייז. ראשית, נשנה את שדה המפתח של x להיות הערך הקודם פחות דלתא. אם כעת המפתח של איקס נהיה קטן יותר מהמפתח של המינימום, x נהיה המינימום החדש בערמה. אם לx אין הורה או אם יש לו הורה עם מפתח שעדיין קטן מהמפתח החדש של x, נסיים את פעולת הפונקציה ((return;. אחרת, נקרא לפונקציה cascadingCuts עם x ועם ההורה שלו כקלט ונסיים את פעולת הפונקציה.

הסיבוכיות תהיה O(1) באמורטייז וO(n) במקרה הגרוע, מכיוון שאנו מבצעים רק פעולות בזמן קבוע למעט הקריאה לcascadingCuts, אשר נראה כי הסיבוכיות שלה היא כאמור O(1) באמורטייז וO(n) במקרה הגרוע.

protected void cascasdingCuts(HeapNode child, HeapNode parent)- הפונקציה חותכת ומסמנת צמתים לפי מנגנון הcascading cuts אשר למדנו בהרצאה. אם child מסומן (true בשדה mark), נקטין את השדה marked ב1 ונסיר את הסימון מchild (נשנה לfalse). נעשה זאת מכיוון שchild יהפוך כעת לשורש חדש בערמה ויש לנו כלל מההרצאה ששורשים אף פעם אינם מסומנים. לאחר מכן, נקרא לפונקציה cut עם child ועם parent בסדר הזה כדי לבצע את החיתוך בפועל. אחר כך נבצע insertNode על child כדי להכניס אותו לערמה כשורש חדש. כעת, אם לצומת parent יש הורה, אז אם parent מסומן נקרא ברקורסיה לcascadingCuts עם parent ועם ההורה, ואחרת, נסמן את parent ונגדיל ב1 את השדה marked.

סיבוכיות הפעולה היא O(n) במקרה הגרוע ו O(1) באמורטייז. הפונקציה מבצעת רק פעולות בזמן קבוע, קריאות לinsertNode שראינו שמתבצע בO(1) תמיד, וקריאות לcut אשר נראה בהמשך כי תמיד פועלת בO(1), אך הסיבוכיות נובעת מן הרקורסיה שעלולה להתרחש בכל קריאה אם יש לנו שרשרת מסומנת. במקרה הגרוע, השרשרת המסומנת תהיה באורך של n ולכן הקריאה לפונקציה תגרור רקורסיה n פעמים, ובכל איטרציה נבצע O(1) עבודה, ולכן הסיבוכיות במקרה הגרוע היא O(n). עם זאת, כפי שראינו בהרצאה, מנגנון הcascading cuts נועד בשביל שבאמורטייז, כלומר במקרה הממוצע, עומק הרקורסיה יהיה O(1) מכיוון שכל צומת לא יכול לאבד יותר מבן אחד לפני שייחתך בעצמו ויהפוך לשורש חדש כחלק מן התהליך. לכן במקרה הממוצע נקבל שהסיבוכיות תהיה O(1).

protected void cut(HeapNode child, HeapNode parent)- הפונקציה מבצעת את החיתוך בין שני הצמתים. ראשית, נוסיף 1 לשדה הסטטי cuts. נשנה את ההורה של child לnull כדי לנתק אותו מההורה. אם הnext של child הוא עצמו, כלומר הוא היה היחיד ברשימת הבנים של parent, נשנה את שדה הchild של parent לnull מכיוון שכעת לא נותרו לו עוד ילדים. אחרת, יש ילדים נוספים לparent. נשנה את שדה child של parent להיות הnext של צומת הקלט child, רק אם child הוא הבן הישיר של parent, ונחבר בעזרת connect2Nodes בין הprev של child לבין הnext של child, ובכך בעצם הסרנו את child מהרשימה. בסוף נוריד ב1 את הדרגה של parent בכל מקרה, מכיוון שהוא איבד בן.

סיבוכיות הפעולה היא O(1) במקרה הגרוע, מכיוון שאנו מבצעים רק מספר חסום של פעולות על שדות לכל קלט ולכל היותר קריאה אחת לconnect2Nodes אשר פועלת תמיד בO(1).

public int potential()- הפונקציה מחשבת ומחזירה את הפוטנציאל של הערמה כפי שהוגדר בהנחיות כלומר הסכום של פעמיים מספר העצים בערמה וכמות הצמתים המסומנים. החישוב מתבצע בעזרת גישה ישירה לשדות trees ו marked ולאחר מכן מוחזרת התוצאה.

סיבוכיות הפעולה היא O(1) במקרה הגרוע מכיוון שמבצעים חישוב אריתמטי בודד ו2 גישות לשדות, פעולות שכולן בזמן קבוע תמיד.

public static int totalLinks()- הפונקציה מחזירה את ערך השדה הסטטי links, בסיבוכיות O(1).

public static int totaCuts()- הפונקציה מחזירה את ערך השדה הסטטי cuts, בסיבוכיות O(1).

public static int[] kMin(FibonacciHeap H, int k)- הפונקציה מקבלת ערמה המכילה עץ בינומי יחיד ומספר שלם k ,ומחזירה מערך של k האיברים הכי קטנים בערמה. ראשית, אם הערמה ריקה, נחזיר מערך ריק. אחרת, נאתחל מערך פלט בגודל k וערמת עזר ריקה. נכניס לערמת העזר בעזרת insert צומת עם מפתח זהה למפתח המינימלי בערמה, ועבור הצומת החדש שיצרנו נשנה את שדה הkmin\_help שלו להצביע על המינימום של הערמה המקורית. כעת, נעבור בלולאת for k פעמים , ובכל איטרציה i (כאשר i ירוץ מ0 לk-1 כולל) ניקח את המינימום בערמת העזר, ונסמן אותו node, נכניס את המפתח שלו למערך הפלט באינדקס i ,ואחר כך נמחק את המינימום מערמת העזר בעזרת deleteMin. כעת, ניגש לצומת מהערמה המקורית שיושב בשדה kmin\_help של node, ונסמן את הבן שלו child. אם child אינו null, נרוץ בלולאה על כל הבנים הללו, ובכל איטרציה נכניס לרשימת העזר בעזרת insert צומת חדש עם המפתח של הבן הנוכחי שרצים עליו , וכך ששדה הkmin\_help של הצומת החדש יצביע על הילד שלפיו הוא נוצר מהערמה המקורית. בסוף לולאת הfor, בניית מערך הפלט הושלמה ונחזיר אותו.

סיבוכיות הפעולה היא ונסמן כאשר h דרגת העץ היחיד שבערמה, כלומר h=deg(H). מכיוון שהערמה מכילה רק עץ בינומי יחיד, לכל צומת בעץ מתקיים שדרגתו קטנה שווה מh, שהיא הדרגה של השורש (מתכונות עץ בינומי שראינו, לשורש יש הכי הרבה ילדים). בכל איטרציה i נשים לב שבערמת העזר כמות הצמתים חסומה מלמעלה על ידי . באיטרציה שלפני כן הוכנסו h איברים לכל היותר (הבנים של הצומת שנשלף מערמת העזר באיטרציה הקודמת ויש לכל היותר h כאלה) ולפני ההכנסה שלהם בוצע deleteMin, ולכן כלל האיברים למעט הם נמצאים בלכל היותר עצים. לכן כאשר נעשה deleteMin בערמה הנוכחית ולכן עלותו תהיה :

*המעבר הלפני אחרון נכון כי* והמעבר האחרון נכון כי כי בעץ בינומי יש צמתים (ראינו בכיתה). לאחר מכן מכניסים O(h) בנים לערמת העזר וכל הכנסה עולה O(1) ולכן העלות של הלולאה (הכוללת) היא O(h).

מכיוון שיש k איטרציות כנ"ל נקבל שעלות הפעולה היא .