1. <u>תיעוד המחלקות:</u>

<u>FibonacciHeap</u>

שדות המחלקה:

א. private HeapNode min – מחזיק את המינימום בכל ערימה.
ב. private int size – מספר הצמתים בערימה.
ג. private static int totalLinks – כלל החיבורים של העצים שבוצעו מתחילת הריצה.
ד. private static int totalCuts – כלל החיתוכים שבוצעו מתחילת הריצה.
ה. private int numberOfTrees – מספר העצים בערימה.
ו. private int numberOfMarks – מספר הצמתים המסומנים בערמה.
ז. Private HeapNode first – מצביע לצומת שהוא השורש של העץ הראשון בערמה.

פירוט המתודות:

- public FibonacciHeap()
בנאי המחלקה, מאתחל את שדה ה- size, numberOfTrees ואת numberOfMarks ל-0.
השמה של int , סיבוכיות O(1).
- public int getNumberOfTrees()
המתודה מחזירה את הערך בשדה numberOfTrees, זה int, סיבוכיות O(1).
- public HeapNode getMin()
המתודה מחזירה מצביע לצומת שהוא עם המפתח המינימלי ושנמצא בשדה min, סיבוכיות O(1).
- public HeapNode getFirst()
המתודה מחזירה מצביע לצומת שנמצא בשדה first, סיבוכיות O(1)..
- public boolean isEmpty()
המתודה בודקת האם השדה first מצביע ל-null, אם כן – מחזירה true ואחרת,false .
סיבוכיות O(1).
- public HeapNode insert(int key)
המתודה קוראת ל(insertAllCases(key, null ,אשר מבצעת את ההכנסה עם המפתח הרצוי ומתבצע ב-O(1) . סה"כ בסיבוכיות O(1).
- public HeapNode insertAllCases(int key, HeapNode originalNode)
בעזרת הבנאי (HeapNode(key, originalNode, המתודה מייצרת את הצומת אותו נכניס לערמה עם המפתח המבוקש ומצביע לצומת המקורי(אם היה כזה, אם לא היה אז זה null).
המתודה מחברת את הצומת החדשה לפי המקרה מתאים – אם הערמה היתה ריקה לפני, או לא. לאחר מכן מתבצע עדכון של השדה min, בעזרת המתודה getKey אשר תחזיר את המפתח של המינימופ ובעזרתו נשווה למפתח של הצומת החדש. לבסוף השדה size והשדה numberOfTrees יגדלו ב-1, נחזיר את צומת החדשה. בסה"כ מספר קבוע של שינויי מצביעים ובדיקת ערכי int – סיבוכיות O(1).
- private HeapNode link(HeapNode x,HeapNode y)
המתודה ראשית תעלה את totalLinks ב-1. מזהה את הparent החדש ואת הchild החדש לפי השוואת המינימום שלהם . מבצע את החיבור בין הparent לchild לפי המקרים – אם ה parent הוא רק שורש ללא ילדים אז רק מחבר את הnext והprev של הchild לעצמו ואחרת תחבר את הצומת שבהתחלה ואת הצומת שבסוף. לאחר מכן, בכל מקרה, תחבר את השדות child וparent של הchild החדש והparent החדש בהתאם. תחזיר את הparent. בסה"כ מספר קבוע של שינויי מצביעים לכן סיבוכיות O(1).

- insertAfter(HeapNode first, HeapNode insertedRoot)

המתודה מכניסה את העץ החדש לערימה בסוף הערימה. סה"כ מספר קבוע של שינוי מצביעים לכן O(1).

- private int numOfBuckets()

המתודה מחשבת את הערך הלוגריתמי של מספר הצמתים בבסיס יחס הזהב. חישוב לכן סה"כ O(1).

- private HeapNode[] toBuckets()

המתודה תכניס את העצים למערך לפי הדרגה שלהם. ראשית תגדיר מערך בגודל numOfBuckets
כתלות בn. לאחר מכן לכל עץ בערמה, תכניס אותו למערך לפי הדרגה, ואם כבר יש שם עץ אז תחבר אותם ותכניס לתא הבא במערך. תחזיר את המערך. בסה"כ עוברת על כל השורשים אשר הם במקרה הגרוע O(n) לכן זו הסיבוכיות המקסימלית.

- private void fromBuckets(HeapNode[] B)

יוצרת מהעצים במערך ערמת פיבונאצ'י חדשה. נאתחל את השדה numberOfTrees ל-0 ואת הfirst לnull . תגדיר משתנה עזר x מסוג HeapNode אשר ייצג את המינימום ברשימה. בצורה איטרטיבית על מספר העצים המקסימלי ,לכל תא במערך שהוא לא null, נכניס את העץ הבא מימין לעצים הקיימים בערמה בעזרת insertAfter ונעדכן את המינימום ל-x ע"י השוואת המפתחות. בכל פעם שנכניס עץ נעדכן גם את השדה numberOfTrees. בסוף הריצה נגדיר את השדה min של הערמה להיות x . לכל תא במערך עושה מספר קבוע של פעולות והמערך באורך O(logn) לכן זו הסיבכויות.

- private void consolidate()

קוראת לtoBuckets שמחזירה מערך שבו - הצמתים מסודרים בעצים, והעצים מסודרים במערך לפי הדרגות – סיבכויות O(n). את המערך תשלח ל fromBuckets אשר מסדרת את העצים לפי הדרגה שלהם מהקטן לגדול – סיבוכיות O(logn). סה"כ סיבוכיות O(n).

- private void deleteMinNoChild()

מבצעת מחיקה בלבד ללא סידור לערמה בינומית של המינימום כאשר הוא הצומת היחיד בעץ, לפי המקרים – אם זה העץ היחיד בערמה אז נקבל בסוף עץ ריק, ואחרת, משרשרת את העצים שנותרים בערמה. בהתאם נקטין את השדה size ב-1. סה"כ מספר קבוע של שינוי מצביעים, O(1).

- private void deleteMinWithChild()

מבצעת מחיקה בלבד ללא סידור הערמה כאשר המינימום הוא לא הצומת היחיד בעץ., לפי המקרים – אם העץ הוא היחיד בערמה, אז נשרשר את השדה first לילד הראשון של המינימום וכך נקבל ערמה חדשה. אם יש יותר מעץ אחד בערמה אז נשרשר את הילדים של הצומת שהיתה הmin בין הprev לnext של הmin המקורי,זה יתבצע בלולאה ונעדכן בה את השדה parent של כל הילדים., בשלב זה אם היו ילדים אשר הם הופכים להיות שורשים והם היו marked, נשנה להם את הmark ל-0 ונעדכן את numberOfMarks. נקטין את size ב-1 בשני המקרים..במקרה הגרוע נצטרך לבצע לולאה על הילדים של הmin לכן זה יהיה בסיבוכיות O(logn).

- public void deleteMin()

מבצעת מחיקה של המינימום לפי המקרים – אם הערמה ריקה, נסיים. אם למינימום אין ילד – נבצע מחיקה בעזרת המתודה deleteMinNoChild. אחרת, בעזרת deleteMinWithChild. אם בסוף המחיקה העץ לא ריק, נקרא ל-consolidate . אחרת, נעדכן את numberOfTrees להיות אפס. בסה"כ נקרא ל-3 מתודות אשר הסיבוכיות שלהן היא – O(n),O(logn),O(logn) בהתאמה. לכן בסה"כ במקרה הגרוע הסיבכויות תהא O(n).

- public HeapNode findMin()

מחזירה את המצביע לצומת אשר נמצא בשדה min. בסה"כ O(1) .

- public void meld (FibonacciHeap heap2)

ראשית מעדכנת את המינימום להיות זה עם המפתח הקטן מבין שתי הערימות. לאחר מכן משרשרת את heap2 אחרי הערימה this. נעדכן את הגודל להיות סכום הצמתים של שתיהן

וכן את מספר העצים ואת מספר הmarked. בסה"כ מספר קבוע של שינויי מצביעים ואתחול מספרים לכן O(1).

- public int size()

מחזיר את הערך בשדה size. סה"כ החזרה של int, O(1).

- public int[] countersRep()

נגדיר מערך מונים בגודל הערך שnumOfBuckets מחזירה, בצורה איטרטיבית על העצים בערמה נסכום כמה עצים יש מכל דרגה. ייתכן שהמערך גדול מדיי ויכיל בסוף אפסים לכן נעתיק את האיברים הרלוונטיים למערך חדש בגודל המדויק של העץ עם הדרגה המקסימלית, נחזיר את המערך המעודכן. בסה"כ שתי לולאות נפרדות על גודל המערך שהוא O(logn) וכן מספר קבוע של שינויי מצביעים – בסה"כ O(logn).

- public void delete(HeapNode x)

בהנתן צומת x , אם הוא המינימום, נקרא לdeleteMin , אחרת נקרא לdecreseKey עם הצומת x ועם דלתא -1 + (x.getKey() - this.min.getKey()) ולאחר מכן נבצע deleteMin. בסה"כ במקרה הגרוע O(n) לפי הסיבוכיות של deleteMin.

- private void insertForCut(HeapNode node)

מבצעת הכנסה ייעודית למתודה cut , מכניסה את העץ לפני שאר העצים בערימה. סה"כ מספר קבוע של שינויי מצביעים לכן O(1).

- private void cut(HeapNode childToRoot, HeapNode parentOfChild)

נגדיל את השדה totalCuts ב-1. אם childToRoot היה מסומן, נאתחל את mark שלו ל-0. נקטין את הדרגה של parentOfChild ב-1. אם childToRoot הוא הchild היחיד של parentOfChild אז נאתחל את ה child של parentOfChild לnull. אחרת, נשרשר את הילדים שנשארים וכן אם מחקנו את הchild של parentOfChild אז נגדיר לו את הchild הבא. לאחר כל אלה בכל מקרה נבצע insertForCut עם childToRoot. בסה"כ מספר קבוע של שינויי מצביעים, מתבצע ב(O(1.

- public void cascadingCut(HeapNode x, HeapNode y)

תחילה נקרא ל cut(x,y) שתנתק את x מ y ותוסיף את x לרשימת השורשים. לאחר מכן בודקים אם y היה מסומן או לא. אם לא – נסמן אותו ונעלה את מספר המוסמנים ב-1. אם כן באופן רקורסיבי נקרא ל cascadingCut(y,y.getParent()) . סיבוכיות – כל פעולת cut היא O(1) ומבצעים לכל היותר n חיתוכים שכן גובה העץ הוא לכל היותר n ולכן בסה"כ O(n) .

- public void decreaseKey(HeapNode x, int delta)

תחילה נעדכן את המפתח של x להיות המפתח שלו פחות הדלתא, ואז נבדוק האם ל x יש אבא וגם המפתח שלו קטן מהמפתח של אבא של, נקרא לcascadingCut(x, x.parent) . ואז בודקים אם המפתח החדש של x קטן מהמינימום – אם כן נעדכן את x להיות המינימום. סה"כ סיבוכיות O(n).

- public int potential()

הפונקציה מחזירה את ערך הפוטנציאל של הערימה , כלומר מספר העצים בערימה ועוד פעמיים מספר הצמתים המסומנים. סיבוכיות O(1) .

- public static int totalLinks()

פונקציה סטטית אשר מחזירה את מספר פעולות ה link שבוצעו. סיבוכיות O(1) .

- public static int totalCuts()

פונקציה סטטית אשר מחזירה את מספר פעולות ה cut שבוצעו. סיבוכיות O(1) .

- public static int[] kMin(FibonacciHeap H, int k)

נאתחל מערך ריק arr של int שיהיה מערך התוצאה. נאתחל ערימת פיבונאצ'י עזר minheap .נכניס את הצומת המינימלי בערימה H לערימת העזר באמצעות קריאה ל פונקציה minHeap.insertAllCases(H.min.getKey(), H.min) , כאשר יש מצביע מהצומת בערימת העזר לצומת בערימה H .לאחר מכן במשך K פעמים – נשמור את הצומת המינמלי בערימת העזר. נמחק אותו מערימת העזר באמצעות קריאה לפונקציה minHeap.deleteMin() נכניס את הילדים של הצומת שאליו הוא מצביע בערימה המקורית

לערימת העזר באמצעות קריאה לפונקציה insertAllCases לכל אחד מן הילדים שלו. כאשר לכל אחד מן הצמתים יש מצביע אליו בערימה המקורית לבסוף נוסיף את המפתח שלו למערך התוצאה.

<u>סיבוכיות</u> - מכיוון שהערימה H היא ערימה בינומית לכל אחד מן הצמתים בערימה יש לכל היותר degH ילדים. ראינו בהרצאה כי סיבוכיות deleteMin היא כמספר העצים שהיו לפני מחיקת הצומת ועוד אלו שנוספו כתוצאה מהמחיקה לפני ביצוע ה consolidation . בנוסף לאחר כל מחיקה אנחנו מקבלים כי מספר העצים בערימה הוא (מספר האיברים בערימה log) . לכל היותר מספר האיברים האיברים בערימת העזר הוא k*degH לכן , לאחר המחיקה מספר העצים הוא לכל היותר log(k*degH) , בכל שלב מוסיפים לכל היותר degH איברים לערימת העזר לכן לפני כל מחיקה מספר העצים בערימת העזר הוא לכל היותר log(k*degH)+degH , לאחר המחיקה נוספים עוד לכל היותר log(k*degh) עצים לערימת העזר ולכן בסה"כ מבצעים בכל שלב O(log(k*degH)+degH) ובסה"כ נקבל כי הסיבוכיות היא :

$$O\big(k(\log(k*degH)+degH)\big)=O\big(k(logk+\log(degH)+degH)\big)=O(k(logk+degH))$$

-

<u>HeapNode</u>

<u>שדות המחלקה :</u>

א. private int key – מפתח הצומת.
ב. private int rank – הדרגה של כל צומת (מספר הילדים שלו).
ג. private int mark – סימון הצומת – 1 אם איבד ילד, 0 אם לא איבד.
ד. private HeapNode child – מצביע לבן של הצומת.
ה. private HeapNode parent – מצביע לאבא של הצומת.
ו. private HeapNode next – מצביע לצומת הבא שאליו מחובר הצומת.
ז. private HeapNode prev – מצביע לצומת הקודם שאליו מחובר הצומת.
ח. private HeapNode originalNode – מצביע ל צומת HeapNode נוסף.

<u>פירוט המתודות :</u>

- public HeapNode(int key) – בנאי המחלקה. מאתחל את השדות key להיות ערך ה key, mark , rank להיות 0 , originalNode להיות null . סיבוכיות – O(1) .
- public HeapNode(int key, HeapNode originalNode) – בנאי נוסף של המחלקה .את השדות mark, rank, key מאתחל באותו האופן , את שדה originalNode מאתחל להיות ה originalNode אשר מקבל כקלט. סיבוכיות O(1) .
- public int getKey() – מחזיר את המפתח של הצומת. סיבוכיות O(1).
- public void setKey(int newKey) – מעדכן את המפתח של הצומת להיות הערך שנתון בקלט. סיבוכיות O(1).
- public HeapNode getPrev() – מחזיר את ה prev של הצומת . סיבוכיות O(1).

- public HeapNode getNext() – מחזיר את הצומת שהוא ה next . סיבוכיות O(1) .
- public void setNext(HeapNode newNext) – מעדכן את שדה ה next של הצומת להיות הצומת הנתון בקלט. סיבוכיות O(1) .
- public void setPrev(HeapNode newPrev) – מעדכן את שדה ה prev להיות הצומת הנתון בקלט הפונקציה. סיבוכיות O(1) .
- public int getRank() – הפונקציה מחזירה את מספר הילדים של הצומת. סיבוכיות O(1) .
- public void setRank(int newRank) – הפונקציה מעדכנת את שדה ה rank של הצומת להיות הערך הנתון בקלט הפונקציה. סיבוכיות O(1) .
- public int getMark() – הפונקציה מחזירה את ה mark של הצומת. סיבוכיות O(1) .
- public void setMark(int newMark) – הפונקציה מעדכנת את שדה ה mark של הצומת להיות הערך הנתון בקלט. סיבוכיות O(1) .
- public HeapNode getChild() – הפונקציה מחזירה את הצומת ששדה ה child מצביע אליו. סיבוכיות O(1) .
- public void setChild(HeapNode newChild) – הפונקציה מעדכנת את שדה ה child של הצומת להיות הצומת הנתון בקלט הפונקציה. – סיבוכיות O(1) .
- public HeapNode getParent() – הפונקציה מחזירה את הצומת שהוא ה parent של הצומת. סיבוכיות O(1) .
- public void setParent(HeapNode newParent) – הפונקציה מעדכנת את שדה ה parent של הצומת להיות הצומת הנתון בקלט. סיבוכיות O(1).

מדידות

ניסוי 1:

M	Run-Time (in miliseconds)	totalLinks	totalCuts	Potential
1024	0.715	1023	18	19
2048	1.0288	2047	20	21
4096	1.9462	4095	22	23

א. אחלק את התוכנית לפעולות:

- מבצעים m+1 פעמים insert, כל אחד עולה O(1) לכן בסה"כ זה O(m).
- מבצעים פעם אחת deleteMin, בגלל שקיבלנו m עצים שכל אחד מהם בעל צומת אחת, זה הworst case של פעולה זו וכפי שלמדנו זה מתבצע בO(m).
- נתחיל לולאה שרצה logm-2 פעמים ובכל פעם נבצע decreaseKey.

לפני ניתוח הסיבוכיות של הלולאה נסביר על מבנה העץ שקיבלנו –
זה עץ בינומי מלא מדרגה logm בדיוק. לפי הגדרתו בנוי משני עצים בינומיים שתלויים אחד על השני(זה עם השורש הגדול יותר תלוי על זה עם השורש הקטן יותר) בדרגה logm-1 כל אחד. בגלל שהכנסנו את האיברים לפי הסדר מהקטן לגדול, ובגלל פעולת הconsolidation נקבל שהעץ הנתלה הוא זה שמכיל את כל האיברים שגדולים מ$\frac{m}{2}$ והעץ שנתלים עליו זה כל האיברים שקטנים מ$\frac{m}{2}$. כך ניתן לחלק את העץ הגדול לשני תתי עצים. בכל פעם נתמקד רק בעץ הנתלה- כלומר זה עם האיברים הגדולים יותר.
נתבונן בחישוב המפתח עליו מתבצע הdecreseKey –
ראשית נחשב את ערך הסכום : $\sum_{k=1}^{i} 0.5^i$, כאשר $i = 0, \dots \log m - 2$. נסמן את ערך הסכום – w. נקבל בכל פעם שw הוא– $0, \frac{1}{2}, \frac{3}{4}, \frac{15}{16}, \frac{31}{32} \dots$. לכן שבכל שלב אנחנו מתמקדים בעץ שגודלו קטן פי 2 מהעץ הקודם ובו כל האיברים גדולים ממש מ$w*m$.

בשלב האחרון – נוסיף למספר זה 2 - אם נוסיף 1 נקבל את השורש של תת העץ עם האיברים שגדולים מ- *w*m* (הוא הכי קטן בקב' איברים זו, לכן הוא השורש). אם נוסיף עוד 1, נקבל את האיבר המינימלי שגדול משורש זה.
סה"כ קיבלנו את האיבר עם המפתח *w*m+2*. אם נבצע בכל פעם *decreseKey* עם מפתח זה, יתבצע *mark* בכל פעם רק לשורש תת העץ הרלוונטי . בפרט שורשים אלה הם ילדים אחד של השני, ואנחנו מסמנים אותם מהקטן לגדול. לכן בכל שלב בלולאה מתבצע *cut* אחד. פעולת *cut* בודדת תעלה *O(1)* וזה מתבצע *logm-1* פעמים, לכן סה"כ הסיבוכיות היא *O(logm)*.

- מבצעים decreseKey לצומת עם המפתח m-1 . צומת זה הוא הילד של הצומת האחרון שקיבל mark=1. לכן כאשר מבצעים לו cut, נהיה חייבים לחתוך גם את האב שהוא מסומן. בגלל שבלולאה סימנו את כל האבות של צומת זה, ברגע שעשינו cut לאב אחד, נצטרך לבצע cut לכל צמתים אלה עד אחד לפני השורש, סה"כ עוד logm-1 פעולות cut. כלומר שלב זה עלה O(logm).

סה"כ סדרת הפעולות ארכה O(m).

ב. סה"כ פעולות link יתבצעו m-1 פעמים, בכל שלב נבצע link לכל שני עצים עם אותה דרגה, אז במקרה זה כל פעם מספר העצים שיש לחבר יקטן פי 2, ונקבל סכום של סדרה הנדסית : $\sum_{i=1}^{logm}\frac{m}{2^i} = -m\left(\frac{1}{m}-1\right) = m-1$ כפי שציפינו. כלומר O(m).
סה"כ פעולות cut לפי ההסבר בסעיף קודם הוא 2(logm -1). כלומר O(logm).

ג. הפעולה decreseKey(m-1) שמתבצעת מחוץ ללולאה היא היקרה ביותר ואורכת O(logm) , הסברנו למעלה.

ניתן לראות שהתוצאות בטבלה בהלימה עם הציפייה התיאורטית לכל סעיף. מספר פעולות ה linkהוא בדיוק m-1 . מספר החיתוכים הוא 2logm -2 , זמן הריצה הוא כקבוע כפול גודל הקלט.

ניסוי 2

M	Run-Time (in miliseconds)	totalLinks	totalCuts	Potential
1000	2.8299	1891	0	6
2000	1.9261	3889	0	6
3000	2.0226	5772	0	7

א. להלן הסבר:
- מבצעים m פעמים insert, כל אחד לוקח O(1) לכן בסה"כ זה O(m).
- עבור המחיקות – ה*deleteMin* הראשון יעלה *O(m)* כי כל העצים הם צומת בלבד, ולאחר מכן כל מחיקה תעלה *O(logm)* כי מספר העצים הוא לכל היותר *logm* ולפני ביצוע פעולת *consolidation* מוסיפים עוד לכל היותר *logm* עצים. זה מתבצע $\frac{m}{2}$ פעמים, לכן סה"כ הסיבוכיות *O(mlogm)*.

סה"כ סדרת הפעולות ארכה *O(mlogm)* פעולות.

ב. ב*deleteMin* הראשון יתבצעו $\frac{M}{2}$ פעולות *link* ולאחר מכן בכל מחיקה(סה"כ עוד $\frac{m}{2}-1$ מחיקות) נבצע עוד *O(logm)* פעולות שכן יש לכל היותר *logm* עצים ונוסיף לאחר מחיקה עוד *logm* עצים לכל היותר עליהם נבצע *consolidation*. סה"כ *O(mlogm)*.
Cut לא מתבצע כלל בסדרה זו של פעולות.

ג. הפוטנציאל מחושב ע"י מספר העצים ועוד מספר המסומנים כפול 2. נשים לב שבגלל שאין פעולות *cut* אז גם אין מסומנים כלל לכן נספור כמה עצים יש. נשים לב שלאורך כל הריצה

יש לכל היותר *logm* עצים והורדנו חצי מהאיברים במהלך הפעולות לכן לכל היותר יהיה שינוי במספר העצים ב-1. סה"כ *O(logm)*.

נשים לב כי התוצאות אכן תואמות את התשובות התיאורטיות , שכן הפוטנציאל הוא *log* של מספר האיברים ומספר פעולות ה *link* הוק כקבוע כפול *mlogm* . בנוסף זמן הריצה הוא כקבוע כפול *mlogm* .