מבני נתונים – תרגיל מעשי 2

מדידות

1. ניסוי 1:

m	Run-Time (in miliseconds)	totalLinks	totalCuts	Potential
1024	0.3748	1023	17	18
2048	0.6538	2047	19	20
4096	1.6624	4095	21	22

א. זמן הריצה האסימפטוטי של סדרת פעולות זו היא O(m):

- יש m פעולות הכנסה, כל אחת ב-O(1) – O(m).
- אח״כ פעולת deleteMin אחת. זה המקרה הגרוע ביותר של פעולה זו, בסיבוכיות O(m) (איחוד של m עצים בדרגה 0 לעץ אחד).
- לאחר מכן מתבצע decreaseKey על O(log m) צמתים. לאחר פעולת ה-deleteMin התקבל עץ בינומי, שלו מבנה קבוע. האינדקסים של הצמתים עליהם מתבצע decreaseKey מתאפיינים בכך שאלו צמתים ללא ילדים (עלים) ולכן הם גם נמצאים בצומת הימני ביותר בתת העץ שלהם. בגלל תכונה זו של מיקום הצמתים, כל פעולת decreaseKey אחרי הפעולה הזו בלולאה לא תגיע להורה שסומן, ולכן לא תגרור cascading-cuts ותתבצע ב-O(1).
- לבסוף, פעולת ה-decreaseKey האחרונה מבצעת cascading-cuts במשך logm-2 פעמים (כאורך הלולאה) משום שזה סך כל הצמתים המסומנים ומשום שמיקום הצומת (האינדקס ה-m-1) ממוקם באופן כזה שכל צומת שסומן עד כה נמצא במסלול בינו לבין השורש, וקריאה ל-decreaseKey תגרור cascading-cuts שיעבור דרך כולם. יש O(log m) צמתים מסומנים ולכן הסיבוכיות בפועל תהיה O(log m).

בסך הכול קיבלנו כי זמן הריצה הוא $O(m+m+\log m+\log m)=O(m)$

ב. פעולות LINK – מתבצעות O(m) פעולות link, כל אחת ב-O(1):
מתחילים מ-m עצים נפרדים של צומת אחד בדרגה 0, ובפעולת deleteMin מאחדים בין כולם לעץ בינומי שלם אחד. זה המקרה הגרוע ביותר עבור פעולה זו, ובמסגרתו מתבצע link במשך m-1 פעמים, בסיבוכיות O(m).
פעולות CUT – מתבצעות O(log m) פעולות cut, כל אחת ב-O(1):
פעולת decreaseKey מתבצעת על עץ בינומי מלא, ולפי הגדרת האינדקסים של הצמתים היא פועלת כל פעם על עלה כך שההורה שמסומן בכל פעולה בסדרה לא נקרא שוב בהמשך הסדרה. לכן, מתבצע חיתוך אחד בכל איטרציה של הלולאה, כלומר סה״כ logm-2 חיתוכים. לאחר מכן מתבצע decreaseKey נוסף שעובר דרך כל ההורים שסומנו בלולאה (כלומר מתבצע cascading-cuts שעובר דרך כולם). לכן, נוסף חיתוך אחד על חיתוך הצומת עצמו, ועוד חיתוך אחד על כל צומת מסומן. בסה״כ: $2(\log m-2)+1=O(\log m)$.

ג. פעולת ה-decreaseKey היקרה ביותר היא הפעולה האחרונה שמופעלת מחוץ ללולאה והיא עולה $\log m-2+1=\log m-1$.

זו הפעולה היקרה ביותר משום שזו הפעולה היחידה בה מתבצע cascading-cuts, בעוד בשאר הפעמים ה-decreaseKey מתבצע ב-O(1).

ניתן לראות שהתוצאות בטבלה תואמות לניתוח האסימפטוטי:
מספר הלינקים בכל הרצה הוא בדיוק m-1 ומספר החיתוכים הוא בדיוק 2(logm-2)+1, כצפוי.

2. ניסוי 2:

M	Run-Time (in miliseconds)	totalLinks	totalCuts	Potential
1000	16.0723	1891	0	6
2000	21.4803	3889	0	6
3000	29.9269	5772	0	7

א. זמן הריצה האסימפטוטי של סדרת הפעולות הוא O(mlogm):
- m הכנסות ב-O(1)
- אח"כ מתבצעות $\frac{\mathrm{m}}{2}$ פעולות deleteMin. הפעולה הראשונה פועלת במקרה הגרוע ביותר של m עצים מדרגה 0, ולכן הסיבוכיות היא O(m). לאחר מכן מספר השורשים בערימה תמיד יהיה חסום ע"י log m ולכן כל deleteMin יתבצע ב-O(log m).

בסה"כ נקבל $O\left(\mathrm{m}+\mathrm{m}+(\frac{\mathrm{m}}{2}-1)\log \mathrm{m}\right)=O(\mathrm{m}\log \mathrm{m})$.

ב. פעולות LINK – מתבצעות $O(m\log m)$ פעולות link:
בפעם הראשונה יש O(m) לינקים כי כל הצמתים הם מדרגה 0.
לאחר מכן כל פעולת מחיקה חסומה ע"י O(logm) כי נוצרת ערימה בינומית תקנית (ומשום שאין פעולות decreaseKey, המבנה התקני של העצים הבינומיים נשמר).
יש בסך הכול $\frac{m}{2}$ מחיקות לכן קיבלנו O(mlogm).
פעולות CUTS – מתבצעות 0 פעולות cut:
אין אף קריאה ל-decrease key ולכן אין צורך בפעולות cut בכלל.

ג. משום שלא מתבצעים אף חיתוכים, הפוטנציאל של המבנה בסוף ריצת הפעולות שקול למספר העצים בערימה.
מספר העצים לאחר סיום הפעולות חסום ע"י logm ולכן גם הפוטנציאל הוא $O(\log m)$.

ניתן לראות שהתוצאות בטבלה תואמות לניתוח האסימפטוטי:
מספר החיתוכים הוא 0 בהתאם לציפיות. ראינו שמספר הלינקים גדל יחד עם גודל הקלט, וכן שהפוטנציאל מוגבל ב-logm.

<u>מסמך תיעוד</u>

<u>המחלקה FibonacciHeap</u>:

זו מחלקה המממשת ערימת פיבונאצ'י – ערימה של עצים המקיימים את הכלל: ערך כל צומת בעץ קטן מערכי ילדיו.

השורשים מקושרים זה לזה ברשימה מקושרת דו כיוונית, וכך גם כל צמתים אחים (ילדים של אותו הצומת). נשמר מצביע למינימום ולשורש השמאלי ביותר.

שדות המחלקה:

- static int CUTS: שדה סטטי המייצג את כמות החיתוכים שנעשו בתוכנית בסך הכול.
- static int LINKS: שדה סטטי המייצג את כמות הלינקים שנעשו בתוכנית בסך הכול.
- HeapNode min: מצביע לצומת בעל המפתח המינימלי בערימה.
- HeapNode first: מצביע לשורש הראשון ברשימה המקושרת, שהוא השורש השמאלי ביותר.
- int size: מספר המייצג את גודל הערימה – כמות הצמתים.
- int numOfRoots: מספר המייצג את כמות השורשים (כמות העצים) בערימה.
- int markedNodes: מספר המייצג את כמות הצמתים המסומנים בערימה.

בנאי המחלקה:

- FibonacciHeap(): יוצר ערימה ריקה, אתחול דיפולטי של השדות.

מתודות המחלקה:

- setMin(HeapNode newMin): מגדירה מינימום חדש עבור הערימה.
- getMin(): מחזירה את המינימום של המחלקה.
- setFirst(HeapNode newFirst): מגדירה שורש ראשון חדש עבור הערימה.
- getFirst(): מחזירה את השורש הראשון (השמאלי).
- setSize(int newSize): מגדירה את גודל הערימה.
- increaseSize(): מגדילה את גודל הערימה ב-1.
- decreaseSize(): מקטינה את גודל הערימה ב-1.
- getSize(): מחזירה את גודל הערימה.
- setNumberOfRoots(int newNum): מגדירה את כמות השורשים (העצים) בערימה.
- increaseNumOfRoots(): מגדילה את כמות השורשים ב-1.
- decreaseNumOfRoots(): מקטינה את כמות השורשים ב-1.
- getNumOfRoots(): מחזירה את מספר השורשים בערימה.
- SetMarkedRoots(int newMarked): מגדירה את כמות הצמתים המסומנים בערימה.
- increaseMarkedRoots(): מגדילה את כמות הצמתים המסומנים ב-1.
- decreaseMarkedRoots(): מקטינה את כמות הצמתים ב-1.
- getMarkedRoots(): מחזירה את מספר הצמתים המסומנים בערימה.

- isEmpty(): מתודה בוליאנית שבודקת אם הערימה ריקה. סיבוכיות O(1) – בדיקה פשוטה.
- insert(int key): מתודה שמקבלת מפתח, יוצרת צומת חדשה ומכניסה אותו לערימה משמאל (העץ הראשון) בתור שורש של עץ מדרגה 0. הסיבוכיות היא O(1) משום שהמימוש הוא עצלני – רק עדכון של שדות ומצביעים.
- deleteMin(): מוחקת את הצומת עם המפתח המינימלי. לאחר מכן קוראת ל-consolidate המאחדת את העצים כך שיש לכל היותר עץ אחד מכל דרגה, ומוצאת את הצומת המינימלי החדש מבין העצים החדשים.
  סיבוכיות: במתודה יש מעבר על רשימת הילדים של המינימום והפיכתם לשורשים בעזרת insertRoots, מעבר זה מתבצע ב-O(log n), משום שדרגת המינימום היא לכל היותר log n. לאחר מכן נקראת המתודה consolidate.
  ב-worst case נקבל סיבוכיות O(log n + n) = O(n) משום שהסיבוכיות worst case של consolidate היא O(n).
- findMin() – מחזירה את הצומת בעל המפתח המינימלי בערימה. סיבוכיות O(1) – הצומת המינימלי הוא שדה במחלקה.
- meld(FibonacciHeap heap2): מתודה שממזגת את הערימה עם ערימה נוספת הנתונה כקלט. המתודה עושה שימוש ב-insertRoots כדי להכניס את רשימת השורשים של heap2 מימין לרשימת השורשים של הערימה המקורית. סיבוכיות המתודה: O(1) – מיזוג הערימות נעשה בצורה עצלנית בדומה ל-insert. רק שינוי של מצביעים והחלפת המינימום במקרה הצורך.
- size(): מחזירה את גודל הערימה (כמות הצמתים בערימות). סיבוכיות O(1) – גודל הערימה נשמר כשדה במחלקה.
- countersRep(): מתודה שמחזירה מערך של דרגות, כך שבכל תא במערך מופיע מספר העצים בדרגה זו בערימה. סיבוכיות O(n) במקרה הגרוע ביותר. המתודה עוברת על כל העצים בערימה, כך שבמקרה הכי גרוע תעבור על n עצים (אם יש n עצים מדרגה 0 בערימה).
- delete(HeapNode x): המתודה מקבלת צומת x ומוחקת אותו מהערימה. היא נעזרת במתודה decreaseKey והופכת את x לצומת המינימלי בערימה, ואז מפעילה deleteMin.
  סיבוכיות WC היא O(n)+O(n)=O(n) משום שסיבוכיות ה-WC הן של decreaseKey והן של deleteMin היא O(n).
- decreaseKey(HeapNode x, int delta): המתודה מקבלת צומת x ומספר שלם delta, ומקטינה את המפתח של x ב-delta. אם הצומת המעודכן מפר את תכונת הערימה, כלומר הוא קטן יותר מההורה שלו, המתודה חותכת את הצומת מההורה והופכת אותו לשורש נוסף ע״י קריאה למתודה makeRoot ב-O(1). לאחר מכן, אם יש צורך (אם ההורה הוא צומת מסומן), המתודה מבצעת cascading-cuts ע״י קריאה למתודה cascade.
  סיבוכיות WC היא O(n) משום שהמתודה cascade פועלת בסיבוכיות זו במקרה הגרוע ביותר.
- potential(): המתודה מחשבת ומחזירה את פונקציית הפוטנציאל של הערימה לפי הנוסחא:

$$\Phi = \#trees + 2 \cdot \#markedNodes$$

מספר העצים ומספר הצמתים המסומנים שמורים כשדות בערימה, ולכן זהו חישוב פשוט בסיבוכיות $O(1)$.

- totalLinks(): זו מתודה סטטית שמחזירה את מספר פעולות ה-cut שבוצעו לאורך כל התוכנית. מספר החיתוכים שמור כשדה סטטי של המחלקה ולכן הסיבוכיות היא $O(1)$.
- totalCuts(): זו מתודה סטטית שמחזירה את מספר פעולות ה-link שבוצעו לאורך כל התוכנית. מספר ה-links שמור כשדה סטטי של המחלקה ולכן הסיבוכיות היא $O(1)$.
- kMin(FibonacciHeap H, int k): זו מתודה סטטית שמקבלת עץ בינומי המיוצג כערימת פיבונאצ׳י ומספר שלם k, ומחזירה מערך המכיל את k המפתחות הקטנים ביותר בעץ.

  המתודה פועלת באופן הבא:

  1) יצירת ערימת פיבונאצ׳י ריקה שתשמש כערימת עזר – $O(1)$.
  2) הכנסת המפתח של הצומת המינימלי, שהוא שורש העץ, לאינדקס 0 במערך – $O(1)$.
  3) הכנסת העתקים של כל הילדים של צומת זה לערימת העזר ע״י קריאה ל-insertLevel על המצביע ששמור בשדה child של הצומת המינימלי. מתודת העזר פועלת בסיבוכיות $O(degH)$.
  4) הפעלת deleteMin על ערימת העזר והכנסת המפתח המינימלי לאינדקס הבא בתור במערך. הסיבוכיות היא log על גודל הערימה כמו שראינו בכיתה.
  5) כעת חוזרים על סעיפים 3-4 עבור הצומת שהוסר מערימת העזר ב-deleteMin. ניתן לגשת לצומת המקורי בעץ הבינומי באמצעות השדה nodePointer אצל כל העתק צומת בערימת העזר.

  המתודה עוצרת כאשר המערך מתמלא, כלומר נמצאו k המפתחות המינימליים.

סיבוכיות: לכל אחד מ-k הצמתים המינימליים מבצעים insertLevel בסיבוכיות $O(degH)$, ואז deleteMin בסיבוכיות log על גודל הערימה הנוכחי.

לכל $1 \leq i \leq k$ נסמן n_i – גודל ערימת העזר בפעולת ה-deleteMin ה-i ונסמן d_i – מספר הצמתים שנוספו לערימה בפעולת ה-insertLevel ה-i. מכאן:

$$n_i = \left(\sum_{j=1}^{i} d_j\right) - (i-1)$$

כלומר, גודל הערימה בשלב נתון הוא מספר הצמתים שנוספו לה עד כה, פחות מספר הצמתים שנמחקו ע״י deleteMin.

משום שאלו צמתים של עץ בינומי, לכל אחד יש לכל היותר degH ילדים. לכן ניתן לומר כי $d_i \leq degH$ לכל i. נקבל:

$$n_i \leq \sum_{j=1}^{i} d_j \leq \sum_{j=1}^{i} degH$$

הביטוי שהתקבל יהיה המקסימלי עבור $i = k$. מכאן שגודל הערימה המקסימלי הוא:

$$\sum_{j=1}^{k} degH = k \cdot degH$$

לכן פעולת deleteMin על הערימה הגדולה ביותר תהיה בסיבוכיות:

$$O(\log(\mathrm{k}\cdot \mathrm{degH})) = O(\log \mathrm{k} + \log(\mathrm{degH}))$$

נקבל שהסיבוכיות הכוללת של המתודה kMin היא:

$$\mathrm{O}\left(\sum_{\mathrm{i}=1}^{\mathrm{k}} \mathrm{degH} + \log \mathrm{n_i}\right) \le \mathrm{O}\left(\sum_{\mathrm{i}=1}^{\mathrm{k}} \mathrm{degH} + \log \mathrm{k} + \log(\mathrm{degH})\right)$$

$$= \mathrm{O}(\mathrm{k}(\mathrm{degH} + \log \mathrm{k} + \log(\mathrm{degH})) = \mathrm{O}(\mathrm{k}(\log \mathrm{k} + \mathrm{degH}))$$

כדרוש.

מתודות עזר:

- link(HeapNode root1, HeapNode root2): מתודת עזר ל-deleteMin: מחברת שני עצים מאותה דרגה k לעץ אחד מדרגה k+1. השורש של העץ החדש הוא השורש בעל המפתח הנמוך יותר מבין השורשים הנתונים, והשורש השני מצטרף בתור הילד הראשון של השורש הראשון.
  סיבוכיות המתודה: O(1) – שינויים פשוטים של מצביעי שדות הצמתים הרלוונטיים.
- clear(): מתודת עזר שהופכת את הערימה לערימה ריקה. סיבוכיות O(1) – שינויים בשדות הערימה.
- consolidate(): מתודת עזר ל-deleteMin: המתודה מחברת את כל העצים באותה דרגה, כך שלבסוף יש לכל היותר עץ אחד בכל דרגה. לאחר מכן המתודה מחזירה את העצים החדשים לערימה לפי סדר הדרגות, מהנמוכה לגבוהה.
  סיבוכיות: המתודה מבצעת פעולות חישוב ב-O(1) וקוראת למתודות העזר toBuckets ו-fromBuckets. לכן סיבוכיות Worst case היא O(n) – toBuckets ב-O(n) ו-fromBuckets ב-O(log n).
- toBuckets(HeapNode[] rankTrees): מתודת עזר ל-consolidate, שמקבלת מהמתודה consolidate מערך ריק של צמתים ומכניסה לכל תא במערך עץ בדרגה של התא. אם התא כבר מלא, המתודה קוראת ל-link, מחברת בין העצים ומכניסה את העץ החדש לתא המתאים במערך. פעולה זו נמשכת עד שיש לכל היותר עץ אחד בכל דרגה.
  סיבוכיות: המתודה עוברת על רשימת השורשים ולכן הסיבוכיות היא כמספר השורשים בערימה. במקרה הגרוע ביותר יש n עצים בדרגה 0, כלומר n שורשים. ולכן סיבוכיות worst case היא O(n).
- fromBuckets(HeapNode[] rankTrees): מתודת עזר ל-consolidate, שמקבלת ממנה מערך של עצים המסודרים לפי דרגה (לאחר שהתבצעה פעולת toBuckets). המתודה מכניסה לערימה את כל העצים לפי סדר הדרגות.
  סיבוכיות: המתודה עוברת על מערך העצים החדש. משום שיש לכל היותר עץ אחד מכל דרגה והוכחנו בכיתה כי כמות העצים מוגבלת ע״י O(log n), סיבוכיות worst case היא O(log n).
- insertRoots(HeapNode prev, HeapNode next, HeapNode firstNode): מתודת עזר המשמשת להכנסת רשימה מקושרת של צמתים אחים שמתחילה ב-firstNode לתוך רשימת השורשים. הצמתים החדשים מוכנסים בין prev ל-next. סיבוכיות המתודה היא O(1) – שינוי של המצביעים הרלוונטיים.

- updateMin(HeapNode x) : המתודה מקבלת שורש x קיים של עץ בערימה, ומעדכנת את המצביע לצומת המינימום אם x קטן מהמינימום הנוכחי. המתודה מבצעת בדיקה פשוטה ולכן הסיבוכיות היא O(1).
- makeRoot(HeapNode x) : מתודת עזר ל-decreaseKey, שמקבלת צומת x שאינו שורש והופכת אותו לשורש. במסגרת זאת היא מעדכנת את השדות של האחים וההורה שלו ומצרפת אותו לרשימת השורשים. לשם כך היא משנה מספר קבוע של מצביעים ולכן פועלת בסיבוכיות O(1).
- cascade(HeapNode x) : מתודת עזר רקורסיבית ל-decreaseKey שמממשת את פעולת cascading-cuts. היא מקבלת את ההורה של הצומת שבוצע לו decreaseKey ומבצעת חיתוכים ע״י makeRoot במסלול בין הצומת לשורש כל עוד הצומת מסומן. היא עוצרת כשהיא מגיעה לצומת לא מסומן, ואם זה לא השורש, היא מסמנת אותו. במקרה הגרוע המתודה יכולה לעבור על כל המסלול מהצומת עד לשורש, ולכן הסיבוכיות היא פונקציה של גובה העץ. ראינו בכיתה שאפשר לבנות ערימת פיבונאצ׳י בצורת שרוך, ולכן במקרה הגרוע גובה העץ יהיה n ובהתאם הסיבוכיות WC תהיה O(n).
- markNode(HeapNode x) : המתודה מסמנת את הצומת x ומעדכנת בהתאם את שדה markedNodes. אלו שתי פעולות עדכון ב-O(1) לכן הסיבוכיות היא O(1).
- unmarkNode(HeapNode x) : המתודה מסירה את הסימון מהצומת x ומעדכנת בהתאם את שדה markedNodes. אלו שתי פעולות עדכון ב-O(1) לכן הסיבוכיות היא O(1).
- insertLevel(FibonacciHeap C, HeapNode x) : זו מתודת עזר ל-kMin. המתודה מקבלת צומת x שנמצא בעץ בינומי H כלשהו, ומכניסה לתוך ערימת פיבונאצ׳י C העתקים של הצומת (צומת פיבונאצ׳י חדש עם אותו מפתח) ואת כל האחים שיש לו בעץ הבינומי. לכל העתק שמוכנס יש מצביע לצומת הבינומי המקורי והוא מאוחסן בשדה nodePointer של ההעתק. לפי תכונת עצים בינומיים, מספר הילדים המקסימלי של כל צומת בעץ הוא degH, ולכן מספר האחים המקסימלי שיש ל-x הוא O(degH). המתודה מבצעת איטרציה על כל האחים ולכן הסיבוכיות היא O(degH).

<u>המחלקה HeapNode</u> :

מחלקה המייצגת צומת בערימת פיבונאצ׳י. כל המתודות במחלקה ממומשות באמצעות בדיקות פשוטות ושינויי מצביעים בסיבוכיות O(1).

שדות המחלקה :

- int key – המפתח של הצומת.
- int rank – הדרגה של הצומת, שקולה למספר הילדים של הצומת.
- boolean marked – האם הצומת מסומן.
- HeapNode child – מצביע לילד הראשון (השמאלי ביותר) של הצומת.
- HeapNode parent – מצביע להורה של הצומת.
- HeapNode prev – מצביע לצומת הקודם ברשימה (הצומת שנמצא משמאל).
- HeapNode next – מצביע לצומת הבא ברשימה (הצומת שנמצא מימין).

- HeapNode nodePointer – מצביע לצומת בעץ בינומי עם אותו מפתח. זה שדה שמאותחל ל-NULL בכל הצמתים ונעשה בו שימוש רק במסגרת המתודה kMin.

בנאי המחלקה:

- HeapNode(int key) – יוצר צומת עם מפתח key המאותחל כשורש יחיד לעץ בדרגה 0.

מתודות המחלקה:

- getPointer(): מחזירה את הצומת הבינומי עליו מצביע הצומת הנוכחי.
- setPointer(HeapNode x): מגדירה צומת בינומי חדש לצומת הנוכחי.
- getKey(): מחזירה את המפתח של הצומת.
- setKey(int newKey): מגדירה מפתח חדש עבור הצומת.
- getRank(): מחזירה את הדרגה של הצומת.
- setRank(int newRank): מגדירה דרגה חדשה עבור הצומת.
- increaseRank(): מגדילה את דרגת הצומת ב-1.
- decreaseRank(): מקטינה את דרגת הצומת ב-1.
- isMarked(): מחזירה true אמ"ם הצומת מסומן, false אחרת.
- getParent(): מחזירה את ההורה של הצומת.
- setParent(HeapNode x): מגדירה הורה חדש עבור הצומת.
- getChild(): מחזירה את הילד הראשון של הצומת.
- setChild(HeapNode x): מגדירה ילד ראשון חדש עבור הצומת.
- getNext(): מחזירה את הצומת הבא.
- setNext(HeapNode x): מגדירה צומת חדש בתור הצומת הבא.
- getPrev(): מחזירה את הצומת הקודם.
- setPrev(HeapNode x): מגדירה צומת חדש בתור הצומת הקודם.
- isOnlyChild(): מחזירה true אמ"ם לצומת אין אחים, false אחרת.
- mark(): הופכת את הצומת למסומן.
- unmark(): הופכת את הצומת ללא מסומן.
- isRoot(): מחזירה true אם הצומת הוא שורש, false אחרת.