תיעוד הפרויקט המעשי ערימת פיבונאצ'י

Ruben Wolhandler rubenw 342674983
Daniel Malash danielmalash 208059113

מחלקת FibonacciHeap

שדות:

`int size` - גודל הערימה, כלומר כמות הצמתים שיש בה
`int totalMarked` - כמות הצמתים המסומנים בערימה
`static int totalCuts` - כמות הפעמים שביצענו את פעולת חיתוך
`static int totalLinks` - כמות הפעמים שביצענו פעולת איחוד
`private HeapNode min` - מצביע לצומת בעלת המפתח המינימלי בערימה
`private HeapNode first` - מצביע לצומת הראשון בערימה

בנאי:
`FibonacciHeap()` – יוצר ערימה ריקה (בגודל 0), המצביעים למינימום ולצומת הראשון מאותחלים לnull, וכמות הצמתים המסומנים היא 0.

מתודות:

`public boolean isEmpty()` – **פלט:** מחזירה אמת אם הערימה ריקה, ושקר אחרת. (בודקת האם גודל הערימה הוא 0).
סיבוכיות: O(1)

`public HeapNode insert(int key)` – יוצרת צומת חדשה עם המפתח key ומכניסה אותו לערימה. (גודל הערימה יעודכן). משתמשת ב`addToFirst(HeapNode insertNode)`
פלט: יוחזר הצומת שהוכנס לערימה
סיבוכיות: O(1)

`private void addToFirst(HeapNode insertNode)` – (מתודת עזר) מכניסה את הצומת insertNode לתחילת הערימה.
סיבוכיות: O(1)

`private int numRootNodes()` – **פלט:** תחזיר את מספר צמתי השורש של הערימה. (עוברת על צמתי השורש וסופרת כמה כאלה יש).
סיבוכיות: O(n)

`public void deleteMin()` – מוחקת את הצומת שהמפתח שלו הוא המינימלי בערימה.
המתודה מבצעת `consolidate()` וע"י כך בסוף הפעולה נקבל ערימה שמכילה עץ אחד בלבד מכל דרגה. ומכיוון שהסיבוכיות של consolidate היא O(n) נקבל
סיבוכיות: O(n)

`private void consolidate()` – הופכת את הערימה לערימה שמכילה עץ אחד בלבד מכל דרגה.
משתמשת במתודות עזר `toBuckets(HeapNode[] b)` ו- `fromBuckets(HeapNode[] b)`
תחילה נכניס את כל העצים למערך b במתודת עזר הראשונה, יתבצעו לינקים בין עצים בעלי אותה הדרגה, ואז במתודה השניה נוציא את העצים מהמערך ונכניס לערימה.
כפי שיוסבר בהמשך הסיבוכיות של מתודת העזר הראשונה היא O(n), ושל השנייה O(log(n)) ולכן
סיבוכיות: O(n)

`private void toBuckets(HeapNode[] b)` –מכניסה לתוך המערך b (כל מקום במערך מייצג את דרגות העץ הקיימות) את העצים בערימה, ומבצעת לינקים כך שכל שני עצים בעלי אותה הדרגה שנכנסו למקום אחד במערך, הופכים לעץ אחד בעל דרגה גדולה יותר.
בסוף הפעולה, המערך b מכיל עץ אחד לכל היותר בכל מקום. האיחודים מתבצעים בעזרת

`.link(HeapNode Parent, HeapNode Child)` אנו עוברים על כל העצים בערימה, לכן
סיבוכיות: O(n)

`private void fromBuckets(HeapNode[] b)` – המתודה מקבלת את המערך b לאחר שהתמלא בtoBuckets ובמעבר על המערך מוציאה את העצים שנוצרו, ומכניסה לערימה.
מכיוון שהמערך הוא בגודל שהוא O(log(n)) כי כך יצרנו אותו, שיהיה בגודל הדרגה המקסימלית בעץ, נקבל:
סיבוכיות: O(log(n))

`private HeapNode link(HeapNode Parent, HeapNode Child)` – המתודה מחברת בין שני צמתים, כך שהצומת בעל המפתח הקטן יותר הוא האבא. (המתודה משנה את המצביעים המתאימים, ומעדכנת את השדות הרלוונטיים). **פלט:** מחזירה את צומת האב.
סיבוכיות: O(1)

`private HeapNode excludeNodeFromLink(HeapNode node)` – המתודה מנתקת את node מהאחים שלו (ומחברת בין האחים המתאימים בחזרה). **פלט:** יוחזר צומת האח הבא של node במידה וקיים.
סיבוכיות: O(1)

`public HeapNode findMin()` – **פלט:** מחזיר את הצומת (מטיפוס HeapNode) שהמפתח שלו מינימלי מבין המפתחות שבערימה.
סיבוכיות: O(1)

`public void meld(FibonacciHeap heap2)` – מיזוג הערימה עם ערימה נוספת heap2.
(שינוי המצביעים והשדות המתאימים עבור פעולת המיזוג)
סיבוכיות: O(1)

`public int size()` – המתודה מחזירה את מספר האיברים בערימה.
סיבוכיות: O(1)

`public int[] countersRep()` – המתודה מחזירה מערך מונים כך שבאינדקס i שמור כמה עצים יש בערימה שהסדר שלהם הוא i.
המתודה יוצרת מערך בגודל המתאים בעזרת `maxRank()`, עוברת על כל צמתי השורש בערימה ומוסיפה למקום המתאים במערך ועוד אחד בכל פעם. **פלט:** יוחזר מערך המונים הנוצר.
מעבר על צמתי השורשים יכול לקחת O(n) ועוד O(n) של מתודת העזר, נקבל
סיבוכיות: O(n)

`private HeapNode nodeMaxRank()` – מחזירה את הצומת עם הדרגה המקסימלית בערימה.
המתודה עוברת על צמתי השורש של הערימה, ומוצאת את הצומת בעל הדרגה המקסימלית.
סיבוכיות: O(n)

`private int maxRank()` – מחזירה את הדרגה המקסימלית בערימה.
המתודה נעזרת ב`nodeMaxRank()` שמחזירה את הצומת בעל הדרגה המקסימלית, ומחזירה את הדרגה של צומת זה.
סיבוכיות: O(n)

`public void delete(HeapNode node)` – מחיקת הצומת node מהערימה.
המתודה מורידה את ערך המפתח של הצומת node לערך מינימלי בעזרת `decreaseKey(HeapNode node, int delta)` ולאחר מכן מוחקת את המינימום.
בגלל השימוש בdecreaseKey נקבל במקרה הגרוע:
סיבוכיות: O(log(n))

`public void decreaseKey(HeapNode node, int delta)` – ערכו של המפתח של הצומת node יופחת בdelta.
המתודה משנה את ערך המפתח של node לערך המתאים, ולאחר מכן במידה וערך מפתח צומת

האב גדול יותר מהמפתח החדש של node, יבוצע `cut(node)` כלומר חיתוך הצומת מצומת האב והעברה שלו להיות צומת שורש. לאחר מכן יבוצע `cascadingCuts(parent)` על מנת לחתוך במעלה הערימה במידת הצורך (מותר שכל צומת יאבד רק בן אחד, אחרת הוא צריך להפוך לצומת שורש). בנוסף יעודכן המינימום במידת הצורך.
מכיוון ש`cascadingCuts` יכולה לקרוא לעצמה במעלה הערימה, במקרה הגרוע נקבל:
סיבוכיות: O(log(n))

`private void cut(HeapNode node)` – חיתוך הצומת node מהאב שלה.
המתודה חותכת את הצומת מהאב, ע"י שינוי המצביעים המתאימים ומכניסה אותו לתחילת הערימה בעזרת `addToFirst(node)`. בנוסף ישתנו השדות *totalCuts* ו- `totalMarked`
סיבוכיות: O(1)

`private void cascadingCuts(HeapNode node)` – מתודה רקורסיבית שתדאג לכך שכל צומת איבד לכל היותר ילד אחד, אחרת תהפוך את הצומת לצומת שורש.
במידה ו node אינו צומת שורש, תבדוק האם הוא "מסומן" כלומר האם כבר איבד ילד אחד קודם. אם אינו מסומן, תסמן אותו (מכיוון שהמתודה נקראת לאחר חיתוך). אם מסומן, תחתוך גם אותו מאביו (תבטל את הסימון מכיוון שעכשיו זה הוא צומת שורש), ולאחר מכן תקרא לעצמה עם צומת האב הבא. תתבצע לכל היותר כגובה הערימה ולכן במקרה הגרוע נקבל:
סיבוכיות: O(log(n))

`public int potential()` – מחזירה את ערך הפוטנציאל הנוכחי של הערימה.
המתודה מחשבת את הפוטנציאל כפי שהוגדר בכיתה, משתמשת ב`numRootNodes()` ולכן נקבל
סיבוכיות: O(n)

`public static int totalLinks()` – מחזירה את מספר כל פעולות הלינק שבוצעו מתחילת ריצת התוכנית. (כאשר פעולת לינק הינה הפעולה שמקבלת שני עצים מאותו סדר ומחברת אותם).
סיבוכיות: O(1)

`public static int totalCuts()` – מחזירה את מספר כל פעולות ה cut שבוצעו מתחילת ריצת התוכנית. פעולת cut קורת בזמן decreaseKey, כאשר מנתקים תת-עץ מהאבא שלו (כולל cascading cuts).
סיבוכיות: O(1)

`public static int[] kMin(FibonacciHeap H, int k)` – מקבלת עץ בינומי H (עם $2^{\deg(H)}$ צמתים) ומספר חיובי $k < size(H)$
פלט: מחזירה מערך ממויין של k הצמתים הקטנים ב-H
הסבר בפירוט: המתודה מאתחלת מערך בגודל k אותו תחזיר, וערימת עזר. נבצע הכנסה של הצומת המינימלי לערימת העזר. באיטרציה הראשונה יוכנס למערך הערך המינימלי. לאחר מכן בעזרת השדה kmin נבצע הכנסה של כל הילדים של הצומת המינימלי לערימת העזר, נמחק את המינימום (האב של הילדים שהוכנסו עתה) ונבצע חיפוש של הצומת הבא בעל המפתח המינימלי. הצומת הזה יכנס למערך, והילדים שלו יכנסו לערימת העזר שלנו. נמשיך ככה עד שהמערך מלא.
המתודה מבצעת לולאת while k פעמים ובתוכה נקראות: deleteMin() ומתבצעת בכל פעם הכנסה של ילדי המינימום הבא.

אכן אנחנו במקרה הגרוע ברמה הגרוע מכניסים deg(H) איברים (עבור k רמה) ולכן הכנסות עולות סה"כ במקרה הגרוע מכניסים כ-k*deg(H) איברים ולכן עלות הכנסות היא O(k*deg(H)) .

ואז מבצעים k פעמים deleteMin שעולה כל אחד O(log(k*deg(H))=O(log(k)+(deg(H))

ולכן סה"כ נקבל שסיבוכיות זמן היא:
O(k*deg(H)+O(k(log(k)+deg(h))=O(k(log(k)+2deg(h))=O(k*(log(k)+deg(H))

סיבוכיות: $O(k(\log k + \deg(H))$.

מחלקת HeapNode

שדות:

`public HeapNode prev` – מצביע לצומת הקודמת ברשימת הילדים (לאח של הצומת)
`public HeapNode next` – מצביע לצומת הבאה ברשימת הילדים (לאח של הצומת)
`private HeapNode parent` – מצביע לצומת האב של הצומת הנוכחית
`private HeapNode child` – מצביע לצומת הבן השמאלי של הצומת
`private int key` – המפתח של הצומת
`private boolean mark` – האם הצומת מסומן או לא (כלומר האם הוא איבד כבר ילד ראשון)
`public int rank` – הדרגה של הצומת
`private HeapNode kmin` – שדה עזר עבור המתודה kmin

בנאי:

`public HeapNode(int key)` – יוצר צומת עם מפתח key, הצומת יהיה לא מסומן בעל דרגה 0, מצביעי ההורה והילד הם null ומצביעי הצומת הבא והקודם יצביעו לצומת הנוכחי.

מתודות:

`public HeapNode getParent()` – מחזירה את שדה ההורה.

`public void setParent(HeapNode parent)` – הופכת את parent להיות האב של הצומת הנוכחית.

`public HeapNode getChild()` – מחזירה את שדה הילד הראשון של הצומת.

`public int getKey()` – מחזירה את שדה המפתח.

`public void setKey(int key)` – הופכת את key להיות המפתח של הצומת הנוכחית.

Sequence 1:

m	Run-Time (in miliseconds)	totalLinks	totalCuts	Potential
1024	0.648188	1023	18	19
2048	1.087266	2047	20	21
4096	2.117142	4095	22	23

א. Insert עולה O(1) ולכן סדרה של m הכנסות עולות O(m)
Delete-min עולה מספר העצים + log(m) , אחרי m הכנסות יש לנו m עצים ולכן עלות deletemin עולה O(m)
Decreasekey עולה ב-amortized O(1) פעולות ולכן אם מבצעים סדרה של log(m)-1 decreasekey
אז עלות של סדרה זו היא O(log(m))
ולכן סה"כ סיבוכיות של סדרה זו היא $O(m) + O(m) + O(\log(m)) = O(m)$

ב. מבצעים deletemin פעם אחד כשיש לנו m עצים מדרגה 0 (אין לשום שורש ילדים)
ולכן עושים successice link כך (בערך):

m/2: links of tree of rank 0
m/4: links of tree of rank 1
m/8: links of tree of rank 2
.
.
.
1:link of rank log(m) (approximation)

$$number\ of\ links \cong m * \sum_{i=1}^{\log(m)} \frac{1}{2^i} < m * \sum_{i=1}^{infini} \frac{1}{2^i} = O(m)$$

ראינו בכיתה כי Amort(decreasekey)= number of cuts <= 2

ולכן עבור סדרה של log(m) decreaseKey נקבל שעשינו number of cuts = O(2log(m))=O(log(m))

ג. DecreaseKey היקרה היא עומק העץ (כאשר נעשה decreaseKey של עלה ונצטרך לעשות cascadingCuts עד השורש) ולכן במקרה הגרוע נקבל ש decreaseKey עולה log(m)

Sequence 2

M	Run-Time (in miliseconds)	totalLinks	totalCuts	Potential
1000	3.232392	1891	0	6
2000	1.538373	3889	0	6
3000	2.001277	5772	0	7

א. ראינו קודם שסדרה insert עולה O(m)
deleteMin בפעם ראשונה עולה O(m) ואז כל פעם O(log(m))
ולכן סדרה של delete min עולה O(m/2)(log(m)=O(mlog(m))

ב. לא נצטרך לעשות cut כי בdeletemin לא עושים את פעולה זו
פעולות link : עושים m/2 פעולות deleteMin כאשר בפעם ראשונה עושים O(m) לינקים
כמו שראינו במדידות 1
ואז אחרי זה יש לנו לכל היותר log(m) עצים בערימה וראינו בכיתה שבמקרה זה עושים
O(log(m)) לינקים וכן הלאה (אותו דבר m/2 פעמים עד כדי קבוע)
ולכן סה"כ נקבל שמספר הלינקים הוא O(m)+O(m*log(m))=O(m*log(m))

ג. Potential= #tree + 2#marked בגלל שלא עשינו שום cuts נקבל כי potential = #tree
ואחרי deleteMin ראינו כי יש לכל היותר log(n+1) עצים כאשר n הוא מספר האיברים
אחרי m/2 פעולות deletemin נשאר m/2 איברים ולכן potential<=log(m/2+1) =
O(log(m))

(בדקנו שזה אכן תואם לתוצאות שלנו)