מבני נתונים - פרויקט מספר 2 - ערימת פיבונאצ'י

מגישים:

שם: איתי צמח
שם משתמש: itaizemah
ת.ז: 209637453

שם: עודד כרמון
שם משתמש: odedcarmon
ת.ז: 208116517

תיעוד המחלקה:

המחלקה FibonacciHeap מממשת ערימת פיבונאצ'י המכילה מפתחות שלמים.
צמתי הערימה ממומשים על ידי המחלקה הפנימית HeapNode.

המחלקה FibonacciHeap

למחלקה 10 שדות:

- מספר שלם size המתאר את גודל הערימה, מאותחל כ-0.
- מצביעים לצמתים min,first,last המצביעים לצומת בעל המפתח הקטן ביותר, ולשורשי העצים השמאלי/ימני ביותר בהתאמה.
- מספרים שלמים numTrees, numMarked המתארים את כמות העצים בערימה וכמות הצמתים המסומנים בערימה, בהתאמה. משומשים לחישוב פוטנציאל העץ.
- מספרים שלמים סטטיים totalLinks, totalCuts המתארים את סה"כ כמות הקישורים והחיתוכים שהתבצעו על ידי כל המופעים של המחלקה עד כה, בהתאמה.
- קבועים סטטיים שלמים NEG_INFTY, POS_INFTY המתארים אינסוף חיובי ושלילי, לשימוש במחיקת איבר שרירותי (ראשית מקטינים את המפתח שלו למינוס אינסוף) וכאשר מחפשים את המינימום במהלך קונסולודיציה (משווים את המפתחות למועמד נוכחי למינימום שמאותחל לאינסוף).

מתודות המחלקה:

- int maxPossibleRank(): מחזיר חסם עליון הדוק אסימפטוטית לדרגה המקסימלית של עץ בערימה, בסיבוכיות $O(1)$, לשימוש ביצירת מערך הדליים בקונסולידציה ולחישוב ()countersRep.

- boolean isEmpty(): בודק בסיבוכיות $O(1)$ האם הערימה ריקה על ידי השוואת הגודל שלה ל-0.
- void addRoot(HeapNode newNode): מוסיפה צומת נתון כשורש בתחילת הערימה, מעדכנת את כל המצביעים הדרושים, מגדילה את שדה numTrees ומעדכנת את המינימום אם השורש החדש שהוספנו הוא מינימום חדש. פועלת בסיבוכיות $O(1)$.
- void removeNode(HeapNode node): אם קוראים לפונקציה זו עם קלט שאינו הצומת האחרון בערימה – הצומת מוסר מרשימת האחים שלו, בין אם הוא שורש או צומת פנימי. פועלת בסיבוכיות $O(1)$.
- HeapNode insert(int key): יוצרת צומת חדש עם המפתח הנתון ומכניסה אותו באופן עצל לתחילת הערימה באמצעות addRoot, מחזירה מצביעה לצומת שהכנסנו כדי שיהיה אפשר לגשת אליו בהמשך. פועלת בסיבוכיות $O(1)$.
- HeapNode bruteFindMin(): מחפשת באופן נאיבי את המינימום על ידי מעבר סדרתי על כל השורשים בערימה והשוואת המפתחות שלהם מול מועמד נוכחי למפתח מינימלי. משומשת לאחר קונסולידציה למציאת המינימום החדש. פועלת בסיבוכיות $O(\#\mathrm{oftrees}) = O(\mathrm{n})$.
- void deleteMin(): אם הערימה לא ריקה - מסירה את הצומת המינימלי מהערימה, אם היו לו ילדים מעלה אותם למעלה במקומו לרשימת השורשים. אם נשארו לאחר המחיקה צמתים בערימה מתבצעת קונסולידציה. פועלת בסיבוכיות $O(\mathrm{n})$ worst case.
- void consolidate(): מבצע קונסולידציה כפי שראינו בכיתה – מאתחל מערך של "דליים" אליו העצים בערימה מוכנסים לפי הדרגה שלהם (כל עץ מוסר ראשית מהערימה עצמה לפני שמועבר למערך), במקרה של התנגשות שני העצים מקושרים באמצעות link ומועברים לדלי המתאים עד שאין יותר התנגשות. לאחר שכל העצים הועברו לדליים, בונים מחדש את הערימה לפי המערך, מהעץ בעל הדרגה הנמוכה ביותר אל העץ בעל הדרגה הגבוהה ביותר. התהליך עד כה עלה $O(\mathrm{n})$, כעת ישנם $O(\log \mathrm{n})$ שורשים עליהם מבצעים bruteFindMin() למציאת המינימום החדש. סה"כ הסיבוכיות היא $O(\mathrm{n})$.
- void link(HeapNode node1, HeapNode node2): אם מקבל שורשים של של שני עצים בעלי אותה דרגה, תולה את העץ בעל המפתח הגדול יותר כבן חדש (שמאלי ביותר) של העץ בעל המפתח הקטן יותר. פועל בסיבוכיות $O(1)$.
- HeapNode findMin(): מחזיר את השדה min של הערימה בסיבוכיות $O(1)$.

- void meld(FibonacciHeap heap2): ממזג באופן עצל את הערימה הנוספת אל הערימה הנוכחית על ידי שרשור עצי הערימה הנוספת בסוף הערימה הנוכחית. פועל בסיבוכיות $O(1)$.
- int size(): מחזיר את השדה size של הערימה בסיבוכיות $O(1)$.
- int[] countersRep(): עובר סדרתית על שורשי הערימה ומעדכן מערך בגודל maxPossibleRank כך שיכיל לבסוף בתא ה-i את מספר העצים מדרגה i בערימה. פועל בסיבוכיות $O(\mathrm{n})$.
- void delete(HeapNode x): מוחק צומת מהערימה על ידי הקטנת המפתח שלו למינוס אינסוף ואז מחיקת המינימום. קורא לפונקציות decreaseKey ו-deleteMin שפועלות ב-$O(\mathrm{n})$ ולכן גם פונקציה זו בעלת סיבוכיות $O(\mathrm{n})$.
- void decreaseKey(HeapNode x, int delta): מקטין את המפתח של הצומת הנתון ב-delta ומעדכן את המינימום במקרה הצורך. אם כעת המפתח של הצומת קטן מהמפתח של ההורה שלו, הצומת נחתך באמצעות קריאה ל-cascadingCuts. הפונקציה פועלת בסיבוכיות $O(n)$.
- void cascadingCuts(HeapNode x, HeapNode y): מקבלת צומת x בעלת הורה y וחותכת את x מההורה שלו באמצעות cut. אם ההורה y לא היה מסומן, מסמנים אותו כעת, ואם היה מסומן, מבצעים גם עליו cascadingCuts.
- void cut(HeapNode x, HeapNode y): מקבלת צומת x בעלת הורה y, מסירה את x מהילדים של y באמצעות removeNode ומוסיפה אותו כשורש חדש בתחילת הערימה באמצעות addRoot. פועלת בסיבוכיות $O(1)$.
- int potential(): מחשבת את הפוטנציאל של העץ על ידי קריאת השדות numTrees, numMarked, פועלת בסיבוכיות $O(1)$.
- static int totalLinks(): מחזירה את השדה totalLinks בסיבוכיות $O(1)$.
- static int totalCuts(): מחזירה ת השדה totalCuts בסיבוכיות $O(1)$.
- static int[] kMin(FibonacciHeap H, int k): מחזירה את k האיברים הקטנים ביותר בעץ בינומי H, לפי האלגוריתם שראינו בתרגול על ערימות בינאריות – נתחזק ערימה (במקרה זה ערימת פיבונאצ'י) נוספת שתכיל את המועמדים הנוכחיים לאיבר הבא בגודלו. לאחר שמוצאים את האיבר המינימלי בערימת העזר ומוסיפים את המפתח שלו למערך שנחזיר, מוסיפים את ילדי האיבר הזה מהעץ אל הערימה ומוחקים אותו מהערימה על ידי מחיקת מינימום. k פעמים מכניסים לכל היותר deg H צמתים לערימה ומוחקים מינימום, לכן כיוון שעלות amortized של הכנסה ומחיקת מינימום הן $O(1)$ ו-
$O(log\, k + log\, deg\, H) = O(lo\, g\, k + degH)$ בהתאמה (כיוון שבערימה יש לכל היותר $O(k \cdot degH)$ צמתים) מתקיים שהסיבוכיות הכוללת היא

$O\big(k(\log k + degH)\big)$ כנדרש. על מנת לגשת חזרה לצומת המתאים בעץ מהצומת בערימה, הפונקציה משתמשת במחלקה HeapNodeWithInfo המרחיבה את HeapNode ומוסיפה לה מצביע שבו אנו מאכסנים עבור כל צומת בערימה את הצומת המתאים לו בעץ הבינומי.

<u>המחלקה HeapNode</u>:
למחלקה 7 שדות:

- מספר שלם key הקובע את מיקום הצומת בערימה.
- מספר שלם rank המתאר כמה ילדים יש לצומת הנוכחי.
- משתנה בוליאני mark שדלוק אם בפעם הבאה שאחד מילדי הצומת יחתך, גם הצומת עצמו יחתך.
- מצביע לצומת child – הילד השמאלי ביותר של הצומת
- מצביעים next, prev לאחים של הצומת אם הוא בן של צומת אחר או לשורשים האחרים אם הצומת הוא שורש בעצמו.
- מצביע parent לצומת שהוא ההורה של הצומת הזה, ו-null אם צומת זה הוא שורש.

מתודות המחלקה הן getters ו-setters אשר רק מחזירים או מעדכנים את שדות המחלקה פרט ל-setMark אשר בודק האם ה-mark שינה את ערכו ומעדכן את numMarked של העץ בהתאם.
בנוסף למחלקה יש בנאי המקבל key ויוצר צומת עם ה-key הנתון.

<u>המחלקה HeapNodeWithInfo</u>:
מחלקה זו מרחיבה את HeapNode ומוסיפה לה שדה נוסף של מצביע לצומת HeapNode אחר בשם info.
למחלקה יש בנאי המקבל מפתח וצומת ויוצר HeapNode באמצעות המפתח ומעדכן את info להיות הצומת הנתון.
מתודות המחלקה הן Getter ו-Setter ל-info.

מדידות:

חלק 1:

m	Run-Time (in miliseconds)	totalLinks	totalCuts	Potential
1024	1.731370	1023	18	19
2048	0.573594	2047	20	21
4096	1.105485	4095	22	23

א. זמן האמורטייזד של insert הוא $O(1)$, זמן האמורטייזד של delete-min הוא $O(log\, n)$, זמן האמורטייזד של decrease-key הוא $O(1)$. מתבצעות m פעולות insert, פעולת delete-min אחת, ו-$log\, m$ פעולות decrease-key. לכן זמן הריצה אסימפטוטי של סדרת פעולות זו כפונקציה של m הוא
$$O(m + log\, m + log\, m) = O(m)$$

ב. לפני קריאת ה-delete-min היחידה המתבצעת בסדרת הפעולות בערימה יש m עצים בגודל 1. כאשר מבצעים link כמות העצים קטנה ב-1. עבור m שהוא חזקה של 2, בסוף תהליך הקונסולודיציה נשאר עץ אחד (בינומי) כלומר התבצעו $m-1$ פעולות link לכן כמות פעולות ה-link הוא $O(m)$. מתבצעות $O(log\, m)$ פעולות decrease-key לכן כמות ה-cuts היא גם $O(log\, m)$ כיוון שכל decrease-key עשוי להיות אחראי ללכל היותר 2 cuts – פעם אחת עבור הצומת שכרגע מקטינים לה את המפתח, ופעם שנייה עבור ההורה של הצומת, אם הוא לא מסומן עדיין וכעת יהיה מסומן ועשוי להחתך בעתיד.

ג. פעולת ה-decrease-key היקרה ביותר האפשרית היא הפעולה האחרונה, במקרה שהיא מבצעת שרשרת של cascading-cuts שחותכת את כל הצמתים שסומנו בחיתוכים של פעולות ה-decrease-key הקודמות. כיוון שכל פעולת decrease-key יכולה להוביל לסימון של צומת אחד לכל היותר והתבצעו $O(log\, m)$ פעולות decrease-key זוהי גם העלות היקרה ביותר האפשרית.

תוצאות הניתוח התאורטי תואמות את הטבלה שקיבלנו – כמות ה-links עולה באופן לינארי ביחד עם m, וכמות ה-cuts עולה באופן לינארי כאשר m מוכפל במספר קבוע, כלומר עולה בקצב $O(log\, m)$.

חלק 2:

m	Run-Time (in miliseconds)	totalLinks	totalCuts	Potential
1000	5.098511	1891	0	6
2000	0.912362	3889	0	6
3000	1.413455	5772	0	7

א. זמן האמורטייזד של insert הוא $O(1)$, זמן האמורטייזד של delete-min הוא $O(log\, n)$. מתבצעות m פעולות insert, ו-$m/2$ פעולת delete-min. לכן זמן הריצה אסימפטוטי של סדרת פעולות זו כפונקציה של m הוא

$$O\left(m+\frac{m}{2}log\, m\right)=O(m\; log\; m)$$

ב. לפני קריאת delete-min הראשונה שמתבצעת בסדרת הפעולות בערימה יש m עצים בגודל 1. כאשר מבצעים link כמות העצים קטנה ב-1. עבור m שהוא חזקה של 2, בסוף תהליך הקונסולודיציה נשאר עץ אחד (בינומי) כלומר התבצעו $m-1$ פעולות link לכן כמות פעולות ה-link לאחר ה-delete-min הראשון היא $O(m)$. בכל שאר פעולות ה-delete-min יש בערימה $O(log\, m)$ עצים לכן סה"כ כמות הלינקים היא $O\left(m+\frac{m}{2}log\, m\right)=O(m\, log\, m)$. לא מתבצעות פעולות decrease-key לכן כמות ה-cuts היא 0.

ג. כיוון שהפוטנציאל שווה לכמות העצים ועוד פעמיים כמות הצמתים המסומנים, ואף צומת לא נהיה מסומן, הפוטנציאל יהיה שווה לכמות העצים. כיוון שמתבצעת לפחות פעולת delete-min אחת, כמות העצים היא לוג של כמות הצמתים הסופית כלומר $O(\log(m/2))=O(\log m)$.

תוצאות הניתוח התאורטי תואמות את הטבלה שקיבלנו – כמות ה-links עולה בקצב $m\log m$, כמות ה-cuts היא 0, והפוטנציאל עולה לאט מאוד, כצפוי לעלייה של $O(\log m)$.