**שמות המגישים :**

**שם: דוד מנשר**

**שם משתמש: davidmensher**

**ת"ז: 212779920**

**תיעוד**

public static int *linkCounter* = 0; - שומר את מספר הלינקים  
public static int *cutCounter* = 0; - שומר את מספר החיתוכים  
public HeapNode minNode; - שומר את האיבר המינימלי  
private int Marked; - שומר את כמות האיברים המסומנים  
private int trees; - שומר את כמות העצים  
public HeapNode firstNode; - שומר את האיבר הראשון  
public int sizeField; - שומר את כמות האיברים

**isEmpty()**: הפונקציה מחזירה ערך TRUE אם ורק אם הערימה ריקה בזמן קבוע.

**setSize(int s)**: הפונקציה משנה את ערך הסייז בזמן קבוע.

**getFirstNode()**: הפונקציה מחזירה את האיבר הראשון בזמן קבוע.

**getFirstNode()**: הפונקציה מחזירה את הקודם לאיבר הראשון בזמן קבוע.

**setFirstNode()**: הפונקציה מגדירה איבר ראשון חדש בזמן קבוע.

**setMinNode()**: הפונקציה שמה איבר מינימלי חדש בזמן קבוע.

**insert(int i)**: הפונקציה יוצרת צומת מסוג HeapNode שמכיל את המפתח ומכניסה אותו לערימה. פעולה זו **מחזירה את הצומת שנוצר ושמכיל את המפתח .** על מנת לבצע את פעולות decreaseKey, delete בהמשך, יש להעביר אליהן צומת זה שמוחזר מפעולת ה insert. הפעולה עושה זאת בזמן קבוע בתור זה שהיא משתמשת בפעולת insertAfter.

**insertAfter(HeapNode i)**: הפונקציה שמה את הנוד בין האיבר הראשון והאיבר הקודם לו תוך כדי שהיא מעדכנת מצביעים ואת כמות העצים בזמן קבוע.

**deleteMin()**: מחיקת הצומת שהמפתח שלו מינימלי מבין המפתחות שבערימה (אין צורך להחזיר אותו). היא עושה זאת על ידי הקטנת size, בדיקת אם למה שאנחנו מוחקים אין ילדים, ואז מעלימים אותו בזמן קבוע על ידי חיבור מחדש של השכנים שלו – אחרת היא מקשרת בין כל הילדים שלו בצדדים תוך כדי שהיא מוחקת לכולם את שדה ההורה, ולבסוף היא מריצה consolidate כדי להשאיר את מספר העצים בלוג. זה לוקח O(log(n)) ammortized אבל O(n) Worst-Case.

**Consilidate()**: מחזירה ערימה חדשה עם לוג n עצים, כאשר n=size. לוקחת O(log(n)) ammortized אבל O(n) Worst-Case בגלל שתי הפעולות שהיא קוראת להן – ToBuckets וגם fromBuckets.

**toBuckets()**: מחזירה מערך בגודל לוג של n כאשר היא עושה succesive linking ועוברת על עצים אחד אחד. לוקחת O(log(n)) ammortized אבל O(n) Worst-Case.

**link(Heapnode Node1,HeapNode Node2)**: מחזירה נוד שמורכב מ2 הנודים שהיו בקלט. מעלה את הlinkcounter ב1, בודקת מה יותר גדול ומשנה את הפוינטרים. לוקחת זמן קבוע.

**fromBuckets()**: מחזירה ערימה שמורכבת מהאיברים במערך. לוקחת O(m) כאשר m זה מספר האיברים במערך – משמע O(log(n)).

**findMin()**: מחזיר את הצומת (מטיפוס HeapNode) שהמפתח שלו מינימלי מבין המפתחות שבערימה. בגלל שזה שדה, אז זה בזמן קבוע.

**meld(heap2):** מיזוג הערימה עם ערימה נוספת heap2. משנה את הפוינטרים בקצוות וגם משנה את כמות העצים והאיברים המסומנים. בכמות קבועה של זמן.

**size()**: הפונקציה מחזירה את מספר האיברים בערימה. יש משתנה אז זה בזמן קבוע.

**countersRep()**: הפונקציה מחזירה מערך מונים כך שבאינדקס שמור כמה עצים יש בערימה שהסדר שלהם הוא . כלומר, היא מחזירה מערך של integers, כך שלכל אינדקס בין עד הדרגה המקסימלית של עץ שקיימת בערימה, הערך שמוחזר במערך הוא מספר העצים שקיימים בערימה מסדר . הפונקציה עושה זאת על ידי מעבר על כל אחד מהעצים ובדיקת דרגתם. הסיבוכיות היא תלוייה במספר העצים והיא אז לוקחת O(log(n)) ammortized אבל O(n) Worst-Case.

**delete(HeapNode x)**: מחיקת הצומת x מהערימה. מבצעת decrease-key ואז delete-min ולכן עושה זאת ב O(log(n)) ammortized אבל O(n) Worst-Case.

**decreaseKey(HeapNode x, int )**: ערכו של המפתח של הצומת x יופחת ב . כלומר, מתבצע , וכמובן שיש לעדכן את מבנה הערימה בהתאם לשינוי זה (למשל לבצע Cascading Cuts במידת הצורך). הפעולה עושה את זה בזמן קבוע פלוס הזמן שלוקח לעשות CascadingCut – משמע, O(1) ammortized אבל O(n) Worst-Case.

**CascadingCut(HeapNode x,HeapNode y)**: הפעולה חותכת את x מy, מוסיפה אותו לעץ ומסמנת את y אם הוא לא כבר מסומן. אם הוא כן אז הוא עושה זאת רקורסיבית על y ואביו. O(1) ammortized אבל O(n) Worst-Case.

**Cut(HeapNode x,HeapNode y)**: הפעולה מוחקת את x מהוריו y בזמן קבוע.

**potential():** הפונקציה מחזירה את ערך הפוטנציאל הנוכחי של הערימה.

הפונטנציאל, כפי שהוגדר בשיעור, הינו

Potential = #trees + 2\*#marked. זה לוקח זמן קבוע.

**totalLinks():** פונקציה סטטית זו מחזירה את מספר כל פעולות הלינק שבוצעו מתחילת ריצת התוכנית. פעולת לינק הינה הפעולה שמקבלת שני עצים מאותו סדר ומחברת אותם. פעולה בזמן קבוע.

**totalCuts():**

פונקציה סטטית זו מחזירה את מספר כל פעולות הcut שבוצעו מתחילת ריצת התוכנית. פעולת cut קורת בזמן decreaseKey, כאשר מנתקים תת-עץ מהאבא שלו (כולל cascading cuts). פעולה בזמן קבוע.

**kMin(FibonacciHeap H, int k):** פונקציה סטטית זו מקבלת עץ בינומי (עם צמתים) ומספר חיובי ומחזירה מערך ממויין של הצמתים הקטנים ב-. על הפעולה לרוץ בסיבוכיות . אין לבצע שום שינוי בערימת הקלט (למשל לבצע delete-min). זה נעשה על ידי ערימת עזר שאנחנו מוסיפים לה את כל האיברים שחשודים להיות מינימלים, ומנצלים את היעילות שלה כדי להגיע לסיבוכיות הנדרשת.

**Seq1:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Potential | totalCuts | totalLinks | Run-Time | M |
| 19 | 6 | 1023 | 3.3 ms | 1024 |
| 22 | 7 | 2047 | 4.4 ms | 2048 |
| 25 | 8 | 4095 | 5.4 ms | 4096 |

1. זמן הריצה האסימפטוטי של סדרת הפעולות הנ"ל הינה O(m).

נסביר זאת:

המימוש של insert לוקח זמן קבוע, שכן בכל הכנסה פשוט משרשרים את הצומת לתחילת רשימת העצים של הערימה כפי שלמדנו בכיתה. לכן ההכנסות ייקחו לנו בסה"כ O(m) . ה-deletemin היחיד שלנו יבוצע למעשה בזמן -WC שלו, שזה O(m) שכן אנחנו מבצעים את ה-consolidate על רשימת עצים באורך m, וראינו בכיתה שזה לוקח זמן לינארי. בנוסף, רצף פעולות ה- decrease-key לוקח לנו O(log(m)), שכן אנחנו מבצעים log(m) פעולות, ובכיתה הוכחנו שבאמורטייזד בערימות פיבונאצ'י פעולת ה-decrease-key מתבצעת ב-O(1).

לכן אם נסכום את כל העלויות הללו, נקבל שאכן זמן הריצה האסימפטוטי של רצף הפעולות הנ"ל הוא O(m).

1. מספר פעולות ה-link הוא O(m). הסיבה לכך היא שפעולות link אנחנו מבצעים רק במסגרת פעולת ה- delete-min. פעולה זו מבוצעת כאן פעם אחת בלבד. ב-deletemin הנ"ל אנחנו בעצם מתחילים מערימה שהיא בצורת "רשימה מקושרת" ומסיימים בערימה בינומית רגילה. לכן מספר הלינקים שייתבצעו בתהליך זה (וזה גם מס' הלינקים הכולל, וכולם ייתבצעו בשלב ה-toBuckets) הוא O(m).

מספר פעולות ה-cut הוא O(log(m)) מכיוון שלפי ג, עלות כל פעולת Decreasekey היא O(1) וכוללת cut אחד, ולכן נקבל שהעלות הכוללת היא O(log(m)).

1. העלות המקסימלית של פעולת ה- decrease-key היא 1.

ההסבר לכך הוא שברצף הפעולות שלנו אנחנו מבצעים log(m) פעולות decrease-key על צמתים עם מפתחות מהצורה :

. על מנת שפעולת ה- decrease-key תהיה גדולה ממש מ-1, נצטרך שיהיו קיימים שני מפתחות שאנו עושים להם decreasekey שלהם יש הורה משותף (כי במקרה זה פעולת ה-decreasekey האחרונה שתתקיים תגרור שני חיתוכים).

נראה שמצב כזה לא ייתכן-

מבניית העץ, נשים לב שכאשר מבצעים את פעולת ה-deletemin היחידה, בשלב ה-consolidate מכניסים את הצמתים לדליים לפי הסדר מהגדול לקטן. מאופן ריצת פעולת העזר toBuckets, עבור כל מפתח i זוגי, האב הישיר שלו יהיה עם המפתח i-1, שכן זהו האיבר היחיד שנמצא בדלי מדרגה 0 כאשר מכניסים את האיבר עם המפתח ה-i.

מהסבר זה ומכך שהמפתחות עליהם אנו מבצעים את פעולת ה- decreasekey הם כולם זוגיים, לכולם יהיה הורים עם מפתחות אי זוגיים **שונים**.

לכן, עלות פעולת ה- decreasekey המקסימלית הינה 1.

אכן ניתן לראות מתוצאות הטבלה שהמדידות מתיישבות עם כל הניתוחים האסימפטוטיים שביצענו.

**Seq2:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Potential | totalCuts | totalLinks | Run-Time | M |
| 6 | 0 | 991 | 5.5 ms | 1000 |
| 6 | 0 | 1990 | 6.6 ms | 2000 |
| 7 | 0 | 2990 | 7.8 ms | 3000 |

1. זמן הריצה האסימפטוטי של סדרת הפעולות הזו הוא O(mlog(m) ).

ההסבר לכך הוא ש-m ההכנסות הראשונות מבוצעות ב-O(m), שכן כל הכנסה מתבצעת ב- O(1)- מכניסים איבר לתחילת הרשימה ע"י כך שמשנים מספר סופי של מצביעים.

בנוסף לכך, זמן הריצה של m/2 המחיקות הוא O(mlog(m)). הביסה לכך היא שבכיתה הראינו שב- amortized, מחיקה מתבצעת ב- O(log(m) ) כאשר m זהו מספר האיברים שכרגע בערימה. לכן אם נסכום , כאשר מספר האיברים שנסכמים הוא , וזה יוצא לאחר חישובים אסימפטוטיים O(mlog(m)).

לכן בסה"כ, זמן הריצה האסימפטוטי של סדרת הפעולות הזו הוא O(mlog(m) ).

1. *תחילה, נשים לב שפעולות cut מתבצעות רק במסגרת פעולת ה-decrease-key, וכיוון שאנחנו לא משתמשים בפעולה זו בסדרה הנכחית, מספר ה-cut יהיה 0.*

*שנית , מספר פעולות ה-link שייתבצעו יהיה O(m). ההסבר לכך הוא שב- delete-min הראשון אנחנו מבצעים O(m) לינקים, שכן פעולת ה-consolidate במחיקה הזו מבצעת "link לכל צומת" (פרט למספר בודד של צמתים שהם השורשים) שכן בתחילת התהליך מתחילים מרשימה מקושרת, ובסופו אנחנו מקבלים ערימה בינומית רגילה.*

*שנית, נשים לב שבכל שאר פעולות ה- delete-min, איננו מבצעים כלל לינקים. הסיבה לכך היא שלאחר הלינק הראשון, האיבר המינימלי מהבנייה יהיה השורש של העץ הבינומי הכי קטן ברשימה הבינומית, ולכן כאשר נבצע delete min נמחק אותו ופעולת ה-consolidate לא תבצע דבר, שכן מהמחיקה אנחנו מקבלים תתי עצים בינומיים קטנים יותר, שלא היו ברשימה לפני כן. מסדר בניית העץ, זה ממשיך באותו האופן מהמחיקה השנייה עד למחיקה האחרונה, ולכן בסה"כ מספר הלינקים הוא O(m).*

1. *נזכר שערך הפוטנציאל הוא מספר העצים+2\*מספר הצמתים המסומנים.*

*הפוטנציאל של המבנה כפונקציה של m בסוף ריצת סדרת הפעולות הוא O(log(m)). נסביר זאת: תחילה, נשים לב שאין כלל צמתים שהם marked, שכן הפונקציונליות הזו נועדה לשימוש בפעולת ה-decrease-key, ובה אין לנו שימוש בסדרה הזו.*

*שנית, נשים לב שבסוף המחיקה האחרונה אנחנו נשארים עם m/2 איברים, וכן במחיקה האחרונה ביצענו consolidate, לכן בסופה נהרימה שלנו היא למעשה ערימה בינומית רגילה. הוכחנו שבערימה בינומית רגילה עם k איברים יש log(k) עצים, הווי אומר בערימה שלנו יהיו log(m/2) עצים, שזה O(log(m)).*

אכן ניתן לראות מתוצאות הטבלה שהמדידות מתיישבות עם כל הניתוחים האסימפטוטיים שביצענו.