**חלק תיאורטי**

**שמות המגישים:**

**עידו רוזינר (20961700) – idorosiner**

**תומר רודניצקי (206627820) rudnitzky**

**תיעוד FibonacciHeap.HeapNode**

כל node ייוצג על ידי:

int Key – המפתח של הצומת.

int rank – הדרגה של הצומת – מספר הצמתים שצומת זה הוא ההורה שלהם.

boolean mark – הסימן של הצומת – סימון הצומת ב-true במידה ואחד הילדים שלו נחתך (כמו שנלמד בשיעור).

HeapNode child – מצביע לילד של הצומת.

HeapNode next – מצביע לצומת הבא.

HeapNode prev – מצביע לצומת הקודם.

HeapNode parent – מצביע להורה של הצומת.

HeapNode pointerKmin – שדה זה משתנה רק בקריאה לפונקציה KMin(FibonacciHeap H, int k). השדה מצביע לצומת בעלת אותו מפתח בערימה המקורית.

להלן הפונקציות במחלקה:

|  |  |  |
| --- | --- | --- |
| **סיבוכיות זמן ריצה** | **תיאור הפונקציה** | **הפונקציה** |
| סיבוכיות זמן ריצה . | הבנאי של המחלקה. | public HeapNode(int key) |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את ה-key של הצומת. | public int getKey() |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את ה-rank של הצומת. | Public int getRank() |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את ה-mark של הצומת. | Public boolean getMark() |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את ה-child של הצומת. | Public HeapNode getChild() |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את ה-next של הצומת. | Public HeapNode getNext() |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את ה-prev של הצומת. | Public HeapNode getPrev() |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את ה-parent של הצומת. | Public HeapNode getParent() |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את ה-pointerKmin של הצומת. | Public HeapNode getpointerKmin() |
| סיבוכיות זמן ריצה . | הפונקציה מקבעת את ה-key של הצומת. | Public void setKey(int key) |
| סיבוכיות זמן ריצה . | הפונקציה מקבעת את ה-rank של הצומת. | Public void setRank(int rank) |
| סיבוכיות זמן ריצה . | הפונקציה מקבעת את ה-mark של הצומת. | Public void setMark(boolean mark) |
| סיבוכיות זמן ריצה . | הפונקציה מקבעת את ה-child של הצומת. | Public void setChild(HeapNode child) |
| סיבוכיות זמן ריצה . | הפונקציה מקבעת את ה-next של הצומת. | Public void setNext(HeapNode next) |
| סיבוכיות זמן ריצה . | הפונקציה מקבעת את ה-prev של הצומת. | Public void setPrev(HeapNode prev) |
| סיבוכיות זמן ריצה . | הפונקציה מקבעת את ה-parent של הצומת. | Public void setParent(HeapNode parent) |
| סיבוכיות זמן ריצה . | הפונקציה מקבעת את ה-pointerKmin של הצומת. | Public void setpointerKMin(HeapNode node) |

**תיעוד FibonacciHeap:**

כל FibonacciHeap (ערימת פיבונצ'י) תיוצג על ידי:

HeapNode min – מצביע לצומת בעלת המפתח המינימלי בערימה.

HeapNode first – מצביע לצומת הראשונה בערימה (הצומת הכי שמאלית/הצומת הכי חדשה).

Int size – מספר הצמתים בערימה.

Int numOfTrees – מספר העצים בערימה.

Int nonMarked – מספר הצמתים בערימה שה-mark שלהם הוא false.

Int highestDeg – הדרגה של העץ הכי גדול בערימה.

כמו כן, עבור כל הערימות פיבונצ'י שנוצרו במהלך ריצת התוכנית ישמרו הפרמטרים הסטטיים:

Static int cuts –מספר כל פעולות החיתוך שבוצעו מתחילת ריצת התוכנית. (תהליך זה יתבצע רק בקריאה ל-decreaseKey() כאשר מנתקים תת עץ מאביו)

Static int links – מספר כל פעולות החיבור שבוצעו מתחילת ריצת התוכנית. (תהליך זה יתבצע רק בקריאה ל-link(HeapNode x, HeapNode y))

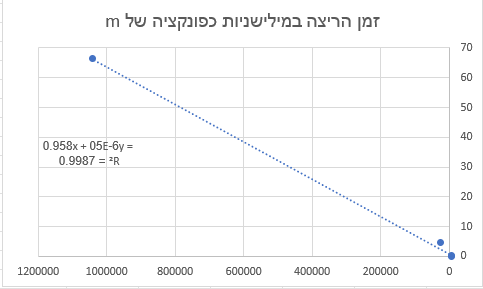
להלן הפונקציות במחלקה:

|  |  |  |
| --- | --- | --- |
| **סיבוכיות זמן ריצה** | **תיאור הפונקציה** | **הפונקציה** |
| סיבוכיות זמן ריצה . | הבנאי של המחלקה. | public FibonacciHeap() |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את ה-min של הערימה. | Public HeapNode getMin() |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את ה-first של הערימה. | Public HeapNode getFirst() |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את ה-size של הערימה. | Public int getSize() |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את ה-numOfTrees של הערימה. | Public int getNumOfTrees() |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את ה-nonmarked של הערימה. | Public int getNonMarked() |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את ה-highestDeg של הערימה. | Public int getHighestDeg() |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את cuts. | Public static int getCuts() |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את ה-links. | Public static int getLinks() |
|  |  |  |
| סיבוכיות זמן ריצה . | הפונקציה מקבעת את ה- min של הערימה. | Public void setMin(HeapNode min) |
| סיבוכיות זמן ריצה . | הפונקציה מקבעת את ה- first של הערימה. | Public void setFirst(HeapNode first) |
| סיבוכיות זמן ריצה . | הפונקציה מקבעת את ה- size של הערימה. | Public void setSize(int size) |
| סיבוכיות זמן ריצה . | הפונקציה מקבעת את ה- numOfTrees של הערימה. | Public void setNumOfTrees(int numOfTrees) |
| סיבוכיות זמן ריצה . | הפונקציה מקבעת את ה-nonMarked של הערימה. | Public void setNonMarked(int nonMarked) |
| סיבוכיות זמן ריצה . | הפונקציה מקבעת את ה-highestDeg של הערימה. | Public void setHighestDeg(int highestDeg) |
| סיבוכיות זמן ריצה . | מעלה את הערך של cuts ב-1. | Public void incCuts() |
| סיבוכיות זמן ריצה . | מעלה את הערך של links ב-1. | Public void incLinks() |
|  |  |  |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה true אם הערימה ריקה ואחרת false. | Public Boolean isEmpty() |
| הפונקציה רק משנה מצביעים ולכן סיבוכיות זמן הריצה שלה הוא . | הפונקציה יוצרת צומת שמפתחו הוא key, מכניסה את הצומת לערימה ומחזירה אותו. | Public HeapNode insert(int key) |
| הפונקציה קוראת ל-removeMin() שרצה בסיבוכיות ולאחר מכן קוראת לפונקציה consolidate() שרצה בסיבוכיות ב-W.C ו- באמורטייזד. לכן הפונקציה deleteMin() רצה בסיבוכיות ב-W.C ו- באמורטייזד. | הפונקציה מוחקת את הצומת בעלת המפתח המינימלי מהערימה. | Public void deleteMin() |
| רצה על כל הילדים של הצומת המינימלית ושנה את המצביעים שלהם – סיבוכיות . כל שאר הפעולות בפונקציה הן שינוי מצביעים ולכן סיבוכיות זמן הריצה של הפונקציה הוא . | הפונקציה מבצעת את עדכון המצביעים כך שלא יהיה ניתן להגיע לצומת המינימלית דרך הערימה. | private void removeMin() |
| הפונקציה רצה על כל השורשים של העצים בערימה – סיבוכיות ב-W.C ו- באמורטייזד, וקוראת לפעולות  link(HeapNode x, HeapNode y) ו- meldNode(HeapNode node) שרצות ב-. כל שאר הפעולות מתבצעות ב- ולכן סיבוכיות זמן הריצה של הפונקציה consolidate() הוא ב-W.C ו- באמורטייזד. | הפונקציה מאחדת את כל העצים בערימה כנלמד בשיעור (הגרסה שמבצעת את התהליך במלואו). | Private void consolidate() |
| הפונקציה רק משנה מצביעים ולכן סיבוכיות זמן הריצה שלה הוא . | הפונקציה מאחדת את שני העצים שבהם x ו-y הם השורשים של העצים (כנלמד בשיעור) ומחזירה את השורש של העץ הנוצר מהמיזוג. | Private HeapNode link(HeapNode x, HeapNode y) |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את ה-min של הערימה. | Public HeapNode findMin() |
| הפונקציה רק משנה מצביעים ולכן סיבוכיות זמן הריצה שלה הוא . | הפונקציה ממזגת את הערימה עם heap2 בצורת meld עצל (כפי שנלמד בשיעור). הערימה החדשה תיכנס מימין לערימה המקורית. | Public void meld(FibonacciHeap heap2) |
| הפונקציה רק משנה מצביעים ולכן סיבוכיות זמן הריצה שלה הוא . | הפונקציה ממזגת את הצומת (שיכולה להיות גם שורש של עץ) לערימה כמו בפונקציה הנ"ל. | Private void meldNode(HeapNode node) |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את ה-size של הערימה. | Public int size() |
| תחילה הפונקציה יוצרת מערך בגודל highestDeg שהוא לכל היותר . לאחר מכן הפונקציה רצה על כל השורשים של העצים ברשימה – סיבוכיות ב-W.C. סה"כ נקבל כי הסיבוכיות של הפונקציה היא . | הפונקציה מחזירה מערך מונים כך שבאינדקס מופיע מספר העצים בערימה שהסדר (rank) שלהם הוא . כלומר, היא מחזירה מערך של integers ,כך שלכל אינדקס בין 0 עד הדרגה המקסימלית של עץ שקיימת בערימה, הערך שמוחזר במערך הוא מספר העצים שקיימים בערימה מסדר . עבור ערימה ריקה יוחזר מערך ריק. | Public int[] countersRep() |
| הפונקציה בהכרח קוראת ל-deleteMin() שרצה ב- ב-W.C ו- באמורטייזד. כמו כן אם האיבר המינימלי אינו x, הפונקציה קוראת ל- decreaseKey(HeapNode x, int delta) שרצה בסיבוכיות בסיבוכיות ב-W.C ו- אמורטייזד. סה"כ נקבל כי הפונקציה רצה בסיבוכיות ב-W.C ו- באמורטייזד. | הפונקציה מוחקת את הצומת x מהערימה. | Delete(HeapNode x) |
| הפונקציה קוראת ל-cascadeCut(HeapNode x) שרצה בסיבוכיות ב-W.C ו- אמורטייזד. כל שאר הפעולות קורות ב-. לכן הפונקציה רצה בסיבוכיות ב-W.C ו- אמורטייזד. | הפונקציה מפחיתה את ערכו של המפתח של הצומת x בערך delta. כלומר, מתבצע . | Public void decreaseKey(HeapNode x, int delta) |
| הפונקציה פועלת בדומה לפונקציה שנלמדה בשיעור ולכן זמן הריצה שלה ב-W.C הוא וזמן האמורטייזד שלה הוא . | הפונקציה מבצעת cascadeCut כמו שנלמד בשיעור. | Private void cascadeCut(HeapNode x) |
| הפונקציה רק משנה מצביעים ולכן סיבוכיות זמן הריצה שלה הוא . | הפונקציה חותכת את הצומת x מהעץ של אביו ומכניסה את התת עץ ש-x הוא שורשו לראש הערימה (משמאל). לבסוף הפונקציה מחזירה את ההורה של x (לפני ביצוע החתך). | Private HeapNode cutAndInsertHeap(HeapNode x) |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את כמות האיברים שאינם marked בערימה. | Public int nonMarked() |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את ערך הפוטנציאל הנוכחי של הערימה. | Public int potential() |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את מספר כל פעולות החיבור שבוצעו מתחילת ריצת התוכנית. | Public static int totalLinks() |
| סיבוכיות זמן ריצה . | הפונקציה מחזירה את מספר כל פעולות החיתוך שבוצעו מתחילת ריצת התוכנית. | Public static int totalCuts() |
| בשיעורי בית 4 שאלה 3 הוכחנו כי סיבוכיות זמן הריצה של הפונקציה הוא .  נשים לב כי . לכן, מכיוון ש- הוא העץ היחיד בערימה, מתקיים:  .  מכאן נקבל כי סיבוכיות זמן הריצה של הפונקציה הוא . | פונקציה סטטית זו מקבלת ערימה H שהיא עץ שדרגתו , ומספר חיובי . הפונקציה מחזירה מערך ממוין של הצמתים הקטנים ב-. | Public static int[] kMin(FibonacciHeap H, int k) |

1.א. להלן תרשים המתאר את תוצאות הניסוי בצורה גראפית. הוספנו קו מגמה, מדד

R^2 ומשוואה המתארת בקירוב את הקו. מכל אלה המסקנה היא כי הביטוי האסימפטומטי

הוא O(m).



מבחינת ניתוח סיבוכיות: ניתן להוכיח את הממצאים עם ניתוח סיבוכיות פשוט:

מבצעים m הכנסות לעץ. סיבוכיות כל הכנסה היא O(1). שאר הפעולות מסיבוכיות לוגריתמית(decrease key- logm) לכל היותר ולכן לא ישפיעו על הניתוח. נקבל O(m) כפי שציפינו.

1.ב. להלן הטבלה:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Potential | TotalCuts | TotalLinks | Run-Time(ms) | m |
| 14 | 5 | 31 | 0.0256 | 32 |
| 29 | 10 | 1023 | 0.3328 | 1024 |
| 44 | 15 | 32767 | 4.6853 | 32768 |
| 59 | 20 | 1048575 | 66.5648 | 1048576 |

1.ג-ו. להלן הטבלה הסופית:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| decreaseKey max cost | Potential | TotalCuts | TotalLinks | case |
| - | 3logm-1 | logm | m-1 | ג' |
| - | 1 | 0 | m-1 | ד' |
| - | m+1 | 0 | 0 | ה' |
| Logm -1 | 2logm | 2logm-1 | m-1 | ו' |

נסביר:

ג': פעולות link- מתבצעות m-1 פעולות כאלה, מכיוון שלאחר מחיקת המינימום נותרים m = 2^k צמתים. נוכיח כעת באינדוקציה כי במהלך ה consolidate אם m חזקה של 2 אז נקבל באמת עץ מלא ועליו נבצע m-1 פעולות link:

בסיס: ערימה בגודל 2 תהיה בנויה מ2 עצים בגודל 0 עלו נבצע לינק 1 ונקבל עץ אחד

צעד: נניח נכונות עבור m1 = 2^k ונוכיח עבור m2 = 2^(k+1) = 2\*2^k = 2m1. מהנחת האינדוקציה נקבל 2 עצים מגודל m ואז נחברם לעץ אחד. מההנחה נקבל כי 2(m1-1)+1 = 2m1-1 = m2-1 כנדרש.

פעולות cut: מתבצעות logm פעולות cut(כמספר פעולות decrease key) שכן אחרי כל פעולות decrease key ערך הצומת יהפוך לשלילי ולכן בהכרח למינימלי מה ולכן נדרש לעשות לו cut. צמתים אלה הם עלים ולכן בכל פעולה כזו נחתוך רק בן אחד – כלומר עבור כל פעולת decrease key נבצע בדיוק פעולת cut 1.

הפוטנציאל: מספר הצמתים המסומנים - הוא מספר החיתוכים פחות אחד(לעולם לא נסמן את השורש)- לכן logm – 1. מספר העצים - לאחר פעולת delete min יווצר עץ אחד ואז בכל צומת נחתכת יווצר עץ נוסף ולכןו יווצרו log(m) + 1 עצים. נציב בנוסחה הנתונה לחישוב הפוטנציאל ונקבל 3logm-1.

ד': פעולות link- פעולות deacrese key לא משפיעה על מספר פעולות link ולכן גם כאן m-1.

פעולות cut- לא נבצע cut כלל – 0 (cut מתבצעת אחרי הפרות ואין)

פוטנציאל- 0 פעולות cut יגרור שלא יהיו צמתים מסומנים, כלומר 0 צמתים מסומנים. מספר העצים הוא 1 ולכן מהצבה בנוסחה נקבל פשוט 1.

ה': פעולות link- פעולות שמתרחשות אחרי delete min שלא נעשית ולכן נקבל 0.

פעולות cut- פעולות decrease key נעשית רק על השורשים שלא נחתכים ולכן נקבל גם כאן 0.

פוטנציאל- מספר העצים – m+1. אין צמתים מסומנים כי שורש לא יכול להיות מסומן. מהצבה בנוסחה נקבל m+ 1.

ו': פעולות link- פעולות deacrease key לא עלולה להוביל לשינוי במספר פעולות link ולכןו דם כאן יהיו m-1 פעולות link.

פעולות cut- נחתך בן 1 מכל צומת ב"חלק" השמאלי של העץ ולכן כל הצמתים שם יוסמנו. ולכן מספר פעולות cut יהיה כאורך החלק. כעת גובה העץ הוא logm-1 והוא שווה לגודל החלק ולכן יתווספו logm-1 פעולות cut. בתוספת לחיתוכים מdecrease key נקבל סך הכל 2logm -1.

פוטנציאל- בכל פעולת cut מספר הצמתים המסומנים יורד ב1 ואילו מספר העצים עולה ב1. בסופו של דבר כל הצמתים המסומנים יהפכו לשורשים(נותרנו עם 0 צמתים מסומנים) ולכן הפוטנציאל יהיה פשוט מספר העצים שקיבלנו בסעיף ג' בתוספת מספר העצים החדשים שנוצרים, כלומר 2logm.

2.א. להלן הטבלה:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Potential | TotalCuts | TotalLinks | Run-Time(ms) | m |
| 6 | 0 | 723 | 3.2464 | 728 |
| 6 | 0 | 6555 | 8.3465 | 6560 |
| 9 | 0 | 590 | 36.4029 | 59048 |
| 10 | 0 | 531431 | 277.3543 | 531440 |
| 14 | 0 | 4782955 | 1743.3553 | 4782968 |

ב. כל הכנסה היא O(1) ולכן סה"כ O(m). נספור את מספר פעולות הlink. נסמן ב S את סכום הספרות של m לפי הבסיס הבינארי( במילים אחרות מספר הספרות שהן 1). נתובנן במצב אחרי המחיקה הראשונה- נקבל ערימה בעלת m עצים עם צומת יחיד. נבצע פעולות link ובסופן יהיה בערימה עץ לכל היותר עץ בינומי אחד דרגה k כש k<=logm>=0. נבחין כי רק במידה במקום הi בייצוג הבינארי יהיה 1 אז יהיה בערימה עץ בינומי מדרגה i. מכאן נסיק כי מספר העצים בערימה שווה למספר הספרות שהן 1 שהוא פשוט S. כאמור, ככל שדרגת העץ גדולה יותר כך הוא מחזיק יותר צמתים (2^deg(tree)). כמו כן, מהגדרת העץ אם נתבונן בצומת x הבנים של הבן הגדול ביותר שלו יהיו גדולים מהבנים של הבן הבא אחריו וכן הלאה. גם לאחר delete min(המינימום הוא השורש הראשון שהוכנס) ה"כלל" ישמר. מכאן נסיק שבמחיקה הבאה אחרי המחיקה הראשונה(ובמחיקות הבאות בכלל) הצומת שיימחק הוא השורש של העץ הקטן בערימה. פעולת המחיקה בהכרח תקטין את דרגות העצים שיווצרו(או תעלים את העץ כליל אם מדובר בצומת בודד). כלומר, לא יווצר עץ עם אותה דרגה מה שמונע את הצורך בפעולות link. מסקנה- פעולות הlink יתרחשו רק בdelete min הראשונה. ולכן מספרן (נמסנו ב- y) יהיה:

Y= (num of tree before link ) - (num of trees after link) = m – S

מכאן נסיק כי הסיבוכיות תהיה O(m) (כל פעולת link היא O(1)). בהרצאה למדנו שלאחר consolidate יהיו O(logm) עצים. O(m\*3/4) = O(m) פעולות מחיקה. וכלן סהכ עבור delete min נקבל O(mlogm) מה שישפיע על הסיבכוית הכוללת. כלומר סה"כ הסיבוכיות תהיה O(mlogm).

ג.פעולות :link הסברנו בסעיף ב כי נבצע סה"כ m-S פעולות link.

פעולות cut: deacrese key עלול לגרום לפעולות cut ואנו לא מבצעים את הפעולה לכן מספר פעולות cut יהיה 0.

הפוטנציאל: בסוף הריצה. הערימה תכיל עצים בינומיים השונים בגודלם. מספר הצמתים יהיה m+1-(3/4)\*m = (1/4)\*m +1. ולכן כפי שהראנו קודם סכום הספרות בייצוג הבינארי יהיה (1/4)\*m +1 (רק איפה שבייצוג הבינארי יש את הספרה 1 עץ בדרגה של מיקום הספרה במחרוזת הבינארית). איו חיתוכים ולכן אין צמתים מסומנים ולכן מהצבה בנוסחה נקבל פשוט את מספר מספר העצים.