**קובץ תיעוד - FibonnaciHeap**

מגישים:

גיא דנקוביץ, 313246811, guydankovich

סידני פיינר, sidneyfeiner, 328871199

**FibonnaciHeap**

למחלקה השדות הבאים:

* HeapNode **first** – צומת שמייצג את תחילת הרשימה המקושרת
* HeapNode **last** – צומת שמייצג את סוף הרשימה המקושרת
* HeapNode **min** – צומת שמייצג את הצומת בעל המפתח המינימאלי
* Int **numOfTrees** – מספר המייצג את כמות העצים
* Int **numOfMarked** – מספר המייצג את כמות האיברים המסומנים בערימה
* Int **size** – מספר המייצג את כמות האיברים בערימה
* Static Int **totalLinks** – מספר המייצג את כמות הלינקים שבוצעו בערימה
* Static Int **totalCuts** – מספר המייצג את כמות החיתוכים בערימה

**תיאור הפונקציות:**

|  |  |  |
| --- | --- | --- |
| סיבוכיות זמן ריצה | תיאור | פונקציה |
| מבצעים כמות קבועה של פעולות לכן הסיבוכיות היא O(1) | הפונקציה מקבלת שני שורשים של עץ ומחברת ביניהם, מחזירה את השורש של העץ המאוחד | **Private HeapNode Link(HeapNode n1,HeapNode n2)** |
| סיבוכיות זמן הריצה היא במקרה הגרוע  O(N)  כיוון שעל כל עץ עוברים ומכניסים למקום המתאים, ומבצעים את הלינקים המתאימים(משתמשת בlink) כמו שנלמד בכיתה סיבוכיות אמורטייזד של פעולה זו היא O(logN) | מכניסה את כל העצים לתוך ה"דליים" המתאימים ועושה את הלינקים הנחוצים | **private void toBuckets (HeapNode [] cells)** |
| סיבוכיות זמן הריצה היא במקרה הגרוע  O(logN) ככמות הדליים | לוקחת את העצים מה"דליים" המתאימים ומרכיבה את הערימה מחדש בהתאם | **private void fromBuckets (HeapNode [] cells,Boolean unmarkRoots)** |
| כמו fromBuckets | פונקצית מעטפת שקוראת ל fromBuckets | **private void fromBuckets (HeapNode [] cells,Boolean unmarkRoots)** |
| סיבוכיות הזמן היא  במקרה הגרוע O(n)  אבל כמו שלמדנו בכיתה סיבוכיות אמורטייזד  O(logN) | מבצעת את כל תהליך הדליים, ולינקים נחוצים לאחר פעולות הdeleteMin | **Public int consolidate()** |
| הסיבוכיות היא בדיוק כמו שנלמדה בכיתה, במקרה הגרוע O(N)  סיבוכיות אמורטייזד  O(logN) | הפוקנציה מבצעת מחיקה של המינימום מהעץ, ולאחר מכן את תהליך הconsolidate ו successive linking | **Public int deleteMin()** |
| סיבוכיות זמן הריצה היא  **O(1)**  בדיקה של שדה | מחזירה האם הערימה ריקה | **Public boolean isEmpty()** |
| סיבוכיות זמן הריצה היא  **O(1)**  סיבוכיות של swapFirst | הפונקציה מכניסה את האיבר לערימה באמצעות lazy insert | **Public HeapNode insert(int k)** |
| סיבוכיות זמן הריצה היא  **O(1)**  סה"כ מתבצעת כמות קבועה של פעולות (שינוי פוינטרים) | הפונקציה מקבלת איבר ומכניסה אותו לתחילת הערימה | **Private void swapFirst(HeapNode node)** |
| כושה מספר קבוע של פעולות (שינוי ערכים) ולכן  **O(1)** | הפונקציה מקבלת שני צמתים בעץ ומחליפה את ערכיהם. | **Private void swapNode(IAVLNode node1,IAVLNode node2)** |
| סיבוכיות זמן הריצה היא  **O(1)**  סה"כ מתבצעת כמות קבועה של פעולות (שינוי פוינטרים) | הפונקציה מעדכנת את כל שדות הערימה לשדות של ערימה ריקה (ניקוי הערימה) | **Private void clear()** |
| סיבוכיות זמן הריצה היא כדרגה של האבא של הnode  O(rank(Node.parent) | מקבלת צומת ומנתקת אותו והאחים שלו מהאבא | **Private static void untieChildren(HeapNode node)** |
| סיבוכיות זמן הריצה היא  כסיבוכיות ניתוק הילדים כלומר הסיבוכיות של untieChildren. כמות הניתוקים היא לפי דרגת הצומת  O(rank(Node)) | הפונקציה מקבלת שורש של עץ שברצוננו למחוק | **Private void deleteRoot (HeapNode node)** |
| החזרת פוינטר O(1) | מחזירה את האיבר המינימלי ברשימה | **Public void findMin()** |
| **O(1)** מתבצע lazy meld – כמות קבועה של פעולות ועדכון פוינטרים | הפונקציה מאחדת (meld) בין הערימה המקורית לבין הערימה הנתונה | **Public void Meld(FibonnaciHeap heap2)** |
| **O(1)**  כמות קבועה של פעולות | פונקצית עזר שנועדה לחבר בין ערימה ריקה לערימה חדשה | **Private void heap1EmptyMeld(IAVLNode node)** |
| **O(1)**  החזרת שדה של הערימה | מחזירה את כמות האיברים בעץ | **Public int size ()** |
| הסיבוכיות היא כמעבר על כל עץ בערימה – O(numOfTrees) | מחזירה מערך שבמקום הi מופיעים כמות העצים בדרגה הi | **Public int[] countersRep(IAVLNode node)** |
| כסיבוכיות של decreaseKey ושל deleteMin סיבוכיות זמן ריצה במקרה הגרוע O(n) וסיבוכיות אמורטייזד O(logN) | מקבלת צומת ומוחקת אותה מהעץ ע"י הורדת המפתח לפחות מהמינימום ואז מחיקתו | **public void delete(HeapNode x)** |
| הפונקציה משתמשת בcascading cuts כדי לבצע מחיקות של צמתים לשורשים בכיתה, סיבוכיות זמן ריצה היא כתלות בגובה העץ, כפי שהוכח בכיתה הסיבוכיות במקרה הגרוע O(N) וסיבוכיות אמורטייזד: O(1) | מורידה את הערך של המפתח של x בדלתא | **Public void decreaseKey(HeapNode x,int delta)** |
| כפי שהוכח בכיתה הסיבוכיות במקרה הגרוע O(logN) וסיבוכיות אמורטייזד: O(1) | הפונקציה מבצעת את החיתוכים הנחוצים מעץ לאחר פעולת הdecreaseKey | **Private void cascadingCut(HeapNode node, HeapNode parent)** |
| מבצעת שינויי פוינטרים – כמות קבועה של פעולות  O(1) | הפונקציה מנתקת צומת מאבא שלו ומעדכנת את השדות המתאימים | **Private void cut(HeapNode node, HeapNode parent)** |
| מתבצעת רק חיבור וגישה לשדות:  **O(1)** | הפונקציה מחזירה את הפוטנציאל בערימה | **Public int potential()** |
| מתבצעת רק גישה לשדה:  **O(1)** | הפונקציה מחזירה את מספר הלינקים שנעשו עד כה בערימה. | **Public static int totalLinks()** |
| מתבצעת רק גישה לשדה:  **O(1)** | הפונקציה מחזירה את מספר החיתוכים שנעשו עד כה בערימה. | **Public static int totalCuts ()** |
| אנו מבצעים k פעמים delete min מערימת עזר, ועבור כל מחיקה הכנסה עצלה של לכל היותר degH ילדים. סה"כ הילדים בערימת העזר קטנה מ2^degH  ולכן כל מחיקה תתבצע בסיבוכיות אמורטייזד של :  נבצע k פעמים deleteMin ובכל שלב הכנסה של degH ולכן סה"כ הסיבוכיות חסומה ע"י:  כיוון שk חסום ע"י 2^degH נקבל שlogK חסום ע"י degH  ולכן הסיבוכיות שחושבה היא כמו הסיבוכיות המתבקשת בשאלה שהיא: | הפונקציה מחזירה מערך של k הצמתים הכי קטנים של ערימה בינומית | **Public static int[] kMin (FibonnaciHeap h,int k)** |
| מתבצעת רק גישה לשדה:  **O(1)** | מחזירה את כמות העצים בעץ | **public int getNumOfTrees ()** |
| מתבצעת רק גישה לשדה:  **O(1)** | מחזירה את העץ הראשון ברשימה | **public int getFirst ()** |

המחלקה משתמשת במחלקה HeapNode שמכיל את השדות הבאים:

Int **key** – ערך המפתח של הצומת

Int **rank** – ערך הדרגה של הצומת

boolean **isMarked** – הגודל של כל צומת (כמות המפתחות הנמתאות תחתיו ועוד 1)

HeapNode **parent** – מצביע לאבא של הצומת

HeapNode **child** – מצביע לילד של הצומת

HeapNode **next** – מצביע לאח הבא של הצומת

HeapNode **prev** – מצביע לאח הקודם של הצומת

HeapNode **pointerToOriginalTree** – שדה שמומש רק עבור kmin בו קיים מצביע מהעתק של צומת לצומת שלו בעץ מקורי

הפונקציות היחידות שהוגדרו בתוך HeapNode הן getters וsetters של השדות

**מדידות:**

1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **m** | **Run-Time**  **(in miliseconds)** | **totalLinks** | **totalCuts** | **Potential** |
| 1024 | 1.58 | 1023 | 18 | 19 |
| 2048 | 3.27 | 2047 | 20 | 21 |
| 4096 | 15.35 | 4095 | 22 | 23 |

1. **insert** – O(1) m פעמים = O(m)

**deleteMin** – O(m) – מדובר במקרה הגרוע של deleteMin כי יש לנו ערימה מקושרת ארוכה

נותרנו עם עץ בינומי תקין בודד בגובה log(m)

הסקנו לפי הקוד והדפסות עזר את החיתוכים שמתבצעים בכל שלב:

**decreaseKey בלולאה:**

מבצעים decreaseKey לעלים על שרשרת של עץ שבכל פעם אבא שלהם לא מסומן – O(logm)

**decreaseKey אחרון:**

מבצעים decreaseKey לעלה שכל האבות שלו עד השורש מסומנים – מבצעים logm cuts – O(logm)

1. כפי שניתחנו בסעיף א' ראינו שביצענו cut אחד עבור כל decreaseKey בתוך הלולאה (לכן סה"כ cuts. עבור הcut האחרון עשינו חיתוכים מהעלה בתחתית העץ עד השורש (cascading cuts) כלומר עוד חיתוכים לכן בסה"כ:

כפי שניתחנו בסעיף א' הכנסנו תחילה 1025 עצים, ואז ביצענו פעולת deleteMin. כיוון שמספר העצים לאחר הdeleteMin הוא חזקה של 2, אז במהלך הconsolidation העץ שמתחיל בדלי הראשון, צריך להגיע לדלי האחרון. כדי להגיע לעץ בודד בדרגה בדיוק H נצטרך לבצע ליניקים. נוכיח באינדוקציה:

מקרה בסיס :

נרצה לקבל עץ בינומי מדרגה 1, אז על העץ הראשון, לא נבצע לינק, על העץ השני נבצע לינק. סה"כ לינקים הוא 1. לכן מקיים את הנדרש.

נניח עבור עצים עם degH = h-1. צעד האינדוקציה, כדי לקבל עץ בדרגה H נצטרך לקבל שני עצים מדרגה h-1 ולבצע לינק ביניהם. כדי לקבל עץ בודד מדרגה H-1, נצטרך לבצע כפול 2 כדי לקבל שני עצים כאלה ואז נבצע עוד לינק, סה"כ כמות הלינקים:

לכן סה"כ הלינקים עבור m שהיא חזקה של 2 נקבל עת בודד מדרגה log(m) וכמות הליניקים שיבוצעו תהיה

סה"כ כמות הליניקים היא:

1. ניתחנו בסעיף א' שכל הdecreaseKey בתוך הלולאה ימחקו מהאבות שלהם (שיהיו לא מסומנים) ויקרו בזמן O(1). פעולת decreaseKey האחרונה תגרור שרשרת של cascading cut בגובה העץ הגדול ביותר שנותר לאחר המחיקה (שהוא logm) פחות אחד, ולכן יתבצעו סה"כ logm-1 חיתוכים וזאת תהיה הפעולה היקרה ביותר. כיוון שזה קורה מעץ בגובה log(m) אז כמות החיתוכים תהיה

2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **M** | **Run-Time**  **(in miliseconds)** | **totalLinks** | **totalCuts** | **Potential** |
| 1000 | 21.14 | 1891 | 0 | 6 |
| 2000 | 4.07 | 3889 | 0 | 6 |
| 3000 | 3.47 | 5772 | 0 | 7 |

1. סיבוכיות זמן ריצה:

**Insert**: נבצע m פעמים lazy insert בO(1) ולכן סה"כ סיבוכיות: O(m)

**DeleteMin**: נבצע m/2 פעמים deleteMin. בפעם הראשונה deleteMin תקח O(n) כי מדובר במקרה הגרוע של deleteMin כשנצטרך לעשות consolidation על כל העצים שכמותם הוא ככמות של m. לאחר מכן נבצע deleteMin על עץ בינומי תקין בסיבוכיות של O(logm) וכך הלאה.

הסיבוכיות של deleteMin אם נבצע m/2 פעמים:

סה"כ נקבל:

1. בגלל שלא מבוצע decreaseKey אז כמות הcuts בסדרה תהיה 0 תמיד. ננתח את כמות הלינקים שמבוצעים במהלך סדרה זו:

פעולת deleteMin הראשונה תעלה ככמות העצים, שבהתחלה הם ככמות האיברים שהוכנסו – m.

לאחר פעולה זו נבצע deleteMin כאשר כמות העצים בערימה היא באזור logM, לכן על כל deleteMin לאחר מכןנבצע פעמים logM לינקים ולכן סה"כ:

1. פונקציות הפוטנציאל תהיה ככמות העצים לאחר סדרה זו, כי אין צמתים מסומנים כי לא בוצעו פעולות של decreaseKey. לאחר פעולת delete Min הערימה הופכת לערימה בינומית תקנית. בביצוע deleteMin האחרונה, בערימה יש בערך log(m/2) עצים לפני הפעולה ולכן כמות העצים לאחר deleteMin היא תהיהי O(log(m)) ולכן פונקצית הפוטנציאל שבמקרה זה תהיה בדיוק ככמות העצים תהיה **O(log(m)).**

ניתוח זה תואם את תוצאות הטבלה.