**ערימת פיבונצ'י**

קארין בל פדר – carinebel - 203336425

תמי לביא – tamilavi - 203010574

תיאור המחלקה

## **HeapNode** מחלקת עזר פנימית:

מחלקה פנימית המייצגת איבר בערימה.

*משתני המחלקה הפנימית:*

* Key- מייצג את המפתח של האיבר
* Mark- מייצג האם האיבר כבר איבד ילד
* Parent- הפניה לאבא במידה ויש
* Children- רשימה מסוג ArrayList של ילדים בעץ
* Degree- הדרגה של הקודקוד – מספר הילדים שיש לו.

*מתודות המחלקה הפנימית:*

* Getters&setters למשתני המחלקה
* בנאי המקבל key
* addChild- הוספת heapNode כבן של הקודוקוד הנוכחי
* removeChild- הוצאת איבר מרשימת הילדים ועדכון ההורה שלו. במקרה הגרוע.
* isLegalKey- האם הקודקוד נמצא במיקום חוקי, כלומר האם המפתח שלו גדול/שווה מערך המפתח של אביו במידה וקיים.

## משתני עזר דינמיים:

* Roots- ArrayList של HeapNode המחזיק שורשים של כל עצי הפיבונצ'י בערימה.
* Size- מספר שלם המחזיק את גודל הערימה.
* Min-node- הפנייה לאיבר עם המפתח המינימלי בערימה (במידה וקיימים איברים).
* Marked- מספר הקודקודים בהם mark הוא חיובי, כלומר הם איבדו ילד ולא נותקו מאביהם.

## משתני עזר סטטיים:

* totalLinks: מספר הכולל של פעולות link שבוצעו עבור כל הערימות.
* totalCuts: מספר הכולל של פעולות cut שבוצעו עבור כל הערימות.

מימוש הפונקציות

הערה על ניתוחי זמן ריצה: בהתאם לאמירתו של שראל, נוכל להתייחס לפעולות מסויימות על רשימה כ- על אף שאינן ככה בפועל.

## פונקציות עזר:

|  |  |  |
| --- | --- | --- |
| **חתימת הפונקציה** | **תפקידה** | **זמן ריצה WC** |
| private void initializeHeap | אתחול הערימה, כולל אתחול כל משתני המחלקה (סטטיים ודינמיים). |  |
| private void link | מעבר על כל השורשים בערימה, ואיחוד כל שני שורשים בעלי גודל דומה. נחזור על התהליך עד שאין שני שורשים בעלי אותו גודל. נעזר ב- linkTwoTrees. |  |
| private HeapNode linkTwoTrees | איחוד שני עצים בעלי אותו מספר ילדים. נחזיר את השורש של העץ החדש (הקטן מביניהם) |  |
| private void findMinInroots | מציאת השורש בעל המפתח המינימלי. בפועל נקרא רק לאחר ביצוע פעולות link, לכן החסם על זמן הריצה. |  |
| private void detachNodeFromItsSiblings | הסרה של קודקוד מהקודקודים המקבילים לו. במידה והוא ילד של קודקוד – נסיר אותו מרשימת הילדים, אחרת הוא שורש ונסיר אותו מרשימת השורשים |  |
| private void addMinChildrenToRoots | הסרתו של min-node מרשימת השורשים (ולכן מהערימה) והוספת ילדיו (במידה וקיימים) כשורשים. נעזר ב- detachNodeFromItsSiblings. |  |
| private void cut | נפריד את קודקוד X מאביו ונוסיף אותו כשורש |  |
| private void cascadingCut | ביצוע cut במעלה העץ בו נמצא שורש X בצורה רקורסיבית, עד אשר נגיע לשורש, או להורה שעוד לא איבד ילד (כלומר mark=false). |  |
| public static void initializeLinks | איפוס totalLinks |  |
| public static int totalCuts | איפוס totalCuts |  |

## פונקציות מטרה:

|  |  |  |
| --- | --- | --- |
| **חתימת הפונקציה** | **תפקידה** | **זמן ריצה WC** |
| public boolean empty | מחזיר האם הערימה ריקה ע"י בדיקה האם יש שורשים |  |
| public HeapNode insert | מוסיף קודקוד חדש לערימה עם המפתח שהתקבל ע"י הוספת הקודקוד כשורש. |  |
| public HeapNode findMin | מחזיר את משתנה המחלקה min\_node |  |
| public void meld | ממזג בין שתי ערימות ע"י הוספת השורשים של העץ השני לעץ הנוכחי. בהתאם נעדכן את min\_node, size, marked. |  |
| public int size | מחזיר את משתנה המחלקה size המחזיק את גודל העץ המעודכן. |  |
| public int[] countersRep | ???? | ??? |
| public void arrayToHeap | מאתחל את הערימה (initializeHeap) ועושה insert לכל אחד מהאיברים במערך. |  |
| public void deleteMin | מחיקת האיבר המינימלי.(במידה וקיים).  משתמש ב- addMinChildrenToRoots (), link (), findMinInRoots (). |  |
| public void decreaseKey | עדכון המפתח של האיבר בהתאם לדלתא, ובמידת הצורך ביצוע cascadingCuts () על מנת לשמור על ערימת חוקית. |  |
| public void delete | מסיר איבר מהערימה ע"י ביצוע decreaseKey כך שיהיה המינמלי, ולאחר מכן ביצוע deleteMin. |  |
| public int potential | מחזיר את הפוטנציאל הנוכחי של העץ בעזרת marked ו-roots (ArrayList.size זה בסיבוכיות ) |  |
| public static int totalLinks | מחזיר את כמות ה-links שבוצעו סה"כ על ערימות. |  |
| public static int totalCuts | מחזיר את כמות ה-cuts שבוצעו סה"כ על ערימות. |  |

מדידות:

## צפי עבור מדידות insert:

למעשה עבור פעולות insert בלבד איננו מבצעים כלל פעולות link או cut. כל פעולת insert היא ולכן זמן הריצה של סדרת פעולות כזו היא .

כיוון שלא עשינו אף פעולת link או cut, כמות העצים היא ככמות האיברים, ואף איבר איננו מסומן. לפיכך הפוטנציאל שנצפה יהיה יהיה אף הוא .

## צפי עבור מדידות insert+delete:

כפי שניתחנו בסעיף הקודם, הסדרה של ה-insertים בלבד הינה -, ולכן זוהי הסיבוכיות של m הפעולות הראשונות. כמו כן נשים לב כי כיוון שאיננו עושים פעולות decrease-key, ולכן איננו מבצעים cuts כלל מה שמוביל לכך שאיננו מסמנים אף קודקוד. לכן הפוטנציאל בסיום רצף הפעולות תלוי בכמות השורשים בלבד.

כעת נסתכל על על הפקודות הבאות של delete-min-

בפקודה הראשונה נדרש לבצע link ל- m קודקודים שונים - . מעתה יש לנו שורשים (כפי שהוכחנו בכיתה), כשהדרגה הגבוהה ביותר היא . לאחר מכן עבור כל delete-min נוסף שמבוצע, ישנם שורשים, ו- ילדים ל-min\_node, לכן פעולת link מבוצעת על שורשים (מקוריים וחדשים).

לכן זמן ריצה של הפעולות- , ולכן זמן ריצה amortized לפי חסם זה הוא . כעת נשפר חסם זה:

כמות פעולות ה-link ים התאורטית היא . אולם איננו מבצעים הכנסות אקראיות, אלא בסדר יורד. לכן עבור אופן הפעולה של הערימה הספציפית נקבל מצב בו של האיברים נמצאים בעצים הכי גדולים לפי סדר יורד. כלומר, האיברים נמצאים בעץ הכי גדול, והאיבר המינימלי 1 נמצא בעץ הכי קטן. לפיכך כאשר נעשה פעולת delete-min, אנו מוחקים שורש עם מספר מינימלי של ילדים. כמו כן בשורש עבור הילדים של השורש המינימלי אין שניים מאותה דרגה זאת שיוון שלא ביצענו פעולות decrease-key/delete שעשויות להשפיע על מבנה העץ, ורק מפעולות link נקבל את המצב הזה.

לפיכך כשנבצע פעולת delete-min, לא נאלץ לבצע linkים נוספים מכיוון שתמיד נוסיף שורשים בעלי גודל ייחודי. לפיכך עבור סדרת הפעולות הזו כמות פעולות ה-link הינה , ולפיכך זמן הריצה הכולל של סדרת הפעולות הוא *, כלומר הזמן ריצה amortized לפעולה הוא .*

## תוצאות המדידות:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Potential** | **totalCuts** | **totalLinks** | **Time** | **Deleted?** | **Size** |
| 1000 | 0 | 0 | 0.204415 | No | 1000 |
| 7 | 0 | 991 | 2.553052 | Yes |
| 2000 | 0 | 0 | 0.381889 | No | 2000 |
| 8 | 0 | 1990 | 2.44614 | Yes |
| 3000 | 0 | 0 | 0.332709 | No | 3000 |
| 8 | 0 | 2990 | 2.827601 | Yes |