**תרגיל מעשי 2 – מבני נתונים**

**מגישים:**

tamirdennis – 208538702 –טמיר דניס

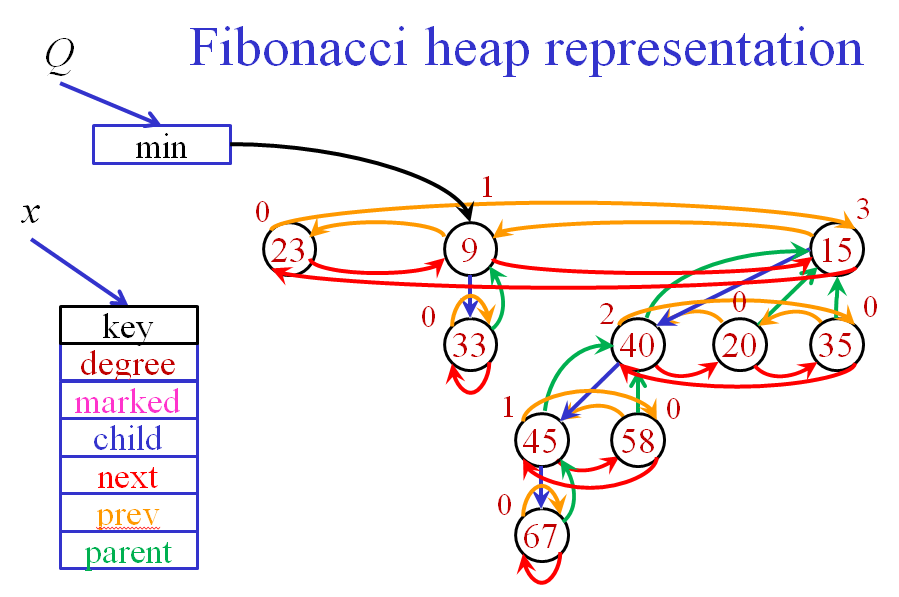
rotemtzaban – 31539406 – רתם צבן

**תיאור מחלקת FibonacciHeap ותיעוד על כל חבר במחלקה:**

המחלקה מממשת את מבני הנתונים הנקרא ערימת פיבונצ'י עבור מספרים שלמים אי שליליים. ערימה היא מבנה שמאפשר הכנסת איבר לפי מפתח אליה וקבלת הצומת המתאים לו, מחיקת צומת, מחיקת הצומת עם המפתח המינימלי בערימה, מציאת הצומת עם המפתח המינימלי בערימה, הורדת ערך למפתח המתאים לצומת מסויים בערימה ואיחוד שתי ערימות לערימה אחת. כחלק ממימוש המבנה המחלקה משתמשת במחלקת עזר HeapNode אשר עצם כזה יכיל את המפתח שהוכנס ופרמטרים נוספים שעוזרים למימוש המבנה.

כל HeapNode בעל השדות הבאים : מפתח הצומת – int key, דרגת הצומת – int degree (מספר הילדים שלו), מסומן או לא – Boolean marked , צומת ילד – HeapNode child, צומת הבא – HeapNode next, צומת לפני – HeapNode prev וצומת הורה – HeapNode parent. הבנאי מקבל מפתח ובונה צומת בודד עם המפתח ועם דרגה מאופסת ללא הורה או ילד ורשימה מקושרת של עצמו בלבד.

רשימה מקושרת של צומת כלשהי היא הרשימה המקושרת הנבנית מהמצביעים של צומת הבא וצומת לפני. ילדים של צומת הם כל הצמתים הנמצאים ברשימה המקושרת של צומת הילד וכל הילדים בעלי מצביע הורה לצומת ההורה המתאים.

לצורך הייצוג של הערימה אפשר להגדיר שורש כצומת ללא הורה בערימה ועץ של שורש הוא כל הצמתים כולל השורש שהם הצאצאים של השורש (ילדים של ילדים וכן הלאה). הרשימה המקושרת בה שורש כלשהו נמצא היא רשימת השורשים של העצים ולכל אחד מהם אין צומת הורה. המבנה תמיד ישמור על כך שצומת הורה תהיה בעלת מפתח קטן יותר מצומת ילד ולכן השורשים של העצים בעלי המפתחות הקטנים ביותר בכל העץ. ניתן לראות ויזואלית כיצד מציגים את הערימה ואת השדות של כל צומת בדוגמא שבאיור הבא:

בנוסף רואים באיור את העצם Q מסוג FibonacciHeap עם מצביע לצומת min שהיא שורש של עץ מבין השורשים. המבנה ישמור שהמצביע הזה תמיד יהיה לשורש עם המפתח המינימלי מבין השורשים ולכן גם מבין כל הצמתים בעץ. לכל ערימת פיבונצ'י יש את השדות הבאים – מספר הצמתים – int size, הצומת עם המפתח המינימלי בערימה – HeapNode min, מספר הצמתים המסומנים – int numMarked ומספר השורשים בערימה – int numRoots. בנאי הערימה בונה ערימה ריקה כלומר מאפס את size, numMarked, numRoots ואת המצביע min מאתחל ל- null.

להלן הפונקציות שערימת פיבונצ'י מממשת יחד עם הסבר מפורט על מה כל פונקציה עושה, איך ומה סיבוכיות זמן הריצה שלה. בטבלה מצויינות מספר פונקציות עזר שבשימוש בחלק מהפונקציות שבהמשך מופיעה טבלה עם הסבר עליהן ועל סיבוכיותן.

|  |  |
| --- | --- |
| **Explanation And Complexity:** | **Method Name:** |
| הפונקציה יוצרת צומת מסוג HeapNode שמכיל את המפתח i ומכניסה אותו לערימה. פעולה זו מחזירה את הצומת שנוצר ושמכיל את המפתח i.  הפונקציה מכניסה את הצומת לעץ כשורש ללא ילדים, כלומר מכניסה אותו לרשימת המקושרת של השורשים לאחר המינימום ומעדכנת את המצביעים הרלוונטים בהתאם. עם זאת הפונקציה מעלה באחד את numRoots ואת size של הערימה. במידה והמפתח של הצומת קטן משל המינימום או שהערימה ריקה הפונקציה מעדכנת את השדה min להיות הצומת שהוכנסה.  **סיבוכיות זמן הריצה של הפונקציה** – עדכון מספר קבוע של מצביעים ולכן O(1) במקרה הגרוע. | Intsert(int i) |
| מחיקת הצומת שהמפתח שלו מינימלי מבין המפתחות שבערימה. תוך כדי עדכון הצומת עם המפתח המינימלי בערימה.  בנוסף למחיקת הצומת שהאיבר שלו מינימלי הפונקציה מוסיפה את הילדים של הצומת הנמחק כשורשים של עצים לערימה (כל ילד עם התת עץ שלו) בעזרת פונקציית העזר cutInternal על כל ילד של הצומת הנמחק. לאחר מכן מבצעת consolidate על הערימה בעזרת פונקציית העזר consolidate שמופיעה בטבלה בהמשך.  **סיבוכיות זמן הריצה של הפונקציה** – cutInternal יחיד הוא O(1) פעולות, לכן על k ילדים של הצומת הנמחק הלולאה תיקח O(k) זמן. כמו שמפורט בטבלה בהמשך הפונקציה consolidate שמעדכנת את המינימום בנוסף להכל לוקחת O(log n) זמן amortized ו- O(n) זמן במקרה הגרוע. בכיתה ראינו שמספר הילדים של צומת מוגבל על ידי log n ולכן O(k) = O(log n) כלומר סך הכל deleteMin לוקחת במקרה הגרוע O(n) זמן ריצה ו - O(log n)זמן amortized. | deleteMin() |
| מחזירה את הצומת עם המפתח המינימלי בערימה בעזרת המצביע min.  **סיבוכיות זמן הריצה של הפונקציה** – בדיוק פעולה אחת שלוקחת O(1) במקרה הגרוע. | findMin() |
| הפונקציה ממזגת את הערימה עם ערימה נוספת heap2 על ידי שרשור שני רשימות השורשים. הפונקציה גם מעדכנת את min, size, numRoots ו- numMarked לפי הצורך.  **סיבוכיות זמן הריצה של הפונקציה** – עדכון מספר קבוע של מצביעים וערכים ולכן O(1) במקרה הגרוע. | Meld(heap2) |
| ערכו של המפתח של הצומת x יופחת ב Δ כלומר, מתבצע 𝑥. 𝑘𝑒𝑦 ← 𝑥. 𝑘𝑒𝑦 – Δ ולאחר מכן מבנה הערימה ישתנה כך שבסופו שוב תתקבל ערימת פיבונצ'י מינימום כבילה שבה לכל צומת שהוא לא שורש ההורה שלו בעל מפתח נמוך יותר ממנו.  **סיבוכיות זמן הריצה של הפונקציה** – אם x הוא לא שורש ולאחר הורדת ערך המפתח שלו הוא בעל מפתח קטן משל ההורה שלו משתמשים בשתי פעולות עזר – הראשונה cut על x וההורה שלו והשנייה cascadingCut על ההורה של x. בנוסף אם המפתח המעודכן קטן יותר מהמפתח של min מעדכנים את min להיות x.  **סיבוכיות זמן הריצה של הפונקציה** – Cut לוקח O(1) זמן ו- cascadingCut לוקח O(log n) זמן במקרה הגרוע ו- O(1) זמן amortized לכן סך הכל decreaseKey תתבצע ב – O(log n) זמן במקרה הגרוע וב- O(1) זמן amortized. | decreaseKey(HeapNode x,int ∆) |
| הפונקצייה משתמשת תחילה בפונקציה decreaseKey על מנת להוריד את ערך המפתח של x להיות מתחת למינימום הנוכחי ובכך הופכת את x להיות המינימום החדש ולאחר מכן מוחקת את המינימום החדש שהוא x בעזרת deleteMin. לכן סך הכל לוקחת O(n) זמן במקרה הגרוע ו- O(log n) זמן amortized בגלל deleteMin. | Delete(HeapNode x) |
| מחזירה True אם ורק אם הערימה ריקה. כלומר אם size = 0.    **סיבוכיות זמן הריצה של הפונקציה** – בדיוק פעולה אחת שלוקחת O(1) במקרה הגרוע. | empty() |
| מחזירה את מספר הצמתים בערימה. כלומר את size.  **סיבוכיות זמן הריצה של הפונקציה** – בדיוק פעולה אחת שלוקחת O(1) במקרה הגרוע. | Size() |
| הפונקציה מחזירה מערך מונים כך שבאינדקס i שמור כמה עצים יש בערימה שהסדר שלהם הוא i. כלומר, היא מחזירה מערך של integers כך שלכל אינדקס i בין 0 עד הדרגה המקסימלית של עץ שקיימת בערימה, הערך שמוחזר במערך הוא מספר העצים שקיימים בערימה מסדר i.  הפונקציה משתמשת בפונקציית עזר getMaxDegree כדי למצוא את הדרגה המקסימלית בערימה ויוצרת מערך בגודל מתאים, לאחר מכן עוברת על כל שורש פעם אחת כדי לעדכן את המערך בערכים מתאימים לכל דרגה.  (לא נדרשנו לנתח זמן ריצה) | countersRep() |
| הפונקציה מחזירה את ערך הפוטנציאל הנוכחי של הערימה כפי שהגדרנו בשיעור – שהוא מספר העצים ועוד פעמיים מספר הצמתים המסומנים.  **סיבוכיות זמן הריצה של הפונקציה** – בדיוק שורה אחת שלוקחת O(1) במקרה הגרוע. | potential() |
| פונקציה סטטית זו מחזירה את מספר כל פעולות ה cut שבוצעו מתחילת ריצת התוכנית. פעולת cut קורת בזמן decreaseKey כאשר מנתקים תת-עץ מההורה שלו. נספרות כל פעולות הניתוק שמתבצעות: גם את פעולת ה cut הישירה של ניתוק תת-העץ מההורה שלו, וגם את פעולות ה- cut שמבוצעות במהלך ה- cascadingCut. הפונרציה רק מחזירה את הערך הסטטי של המחלקה totalCuts.  **סיבוכיות זמן הריצה של הפונקציה** – בדיוק פעולה אחת שלוקחת O(1) במקרה הגרוע. | totalCuts() |
| פונקציה סטטית זו מחזירה את מספר כל פעולות הלינק שבוצעו מתחילת ריצת התוכנית. פעולת לינק הינה הפעולה שמקבלת שני עצים מאותו סדר ומחברת אותם (העץ שהמפתח בשורש שלו גדול יותר נתלה על העץ השני) כך שמתקבל עץ מסדר אחד גדול יותר. זהו גם מספר הפעמים שנקראת הפונקציה link בתוכנית. הפונקציה רק מחזירה את הערך הסטטי של המחלקה totalLinks.    **סיבוכיות זמן הריצה של הפונקציה** – בדיוק פעולה אחת שלוקחת O(1) במקרה הגרוע. | totalLinks() |

להלן פונקציות העזר שהיו בשימוש יחד עם הסבר מפורט על מה כל פונקציה עושה, איך ומה סיבוכיות זמן הריצה שלה:

|  |  |
| --- | --- |
| 1. יצירת currentRoots שמכיל את השורשים של הערימה בעזרת פונקציית העזר createRootList. 2. אתחול מערך הנקרא resultRoots בגודל של דרגת השורש המקסימלי האפשרי בעץ בעזרת פונקציית העזר getMaxPossibleRootDegree. 3. מעבר על כל צומת במערך currentRoots וביצוע מספר לינקים איתו בעזרת פונקציית העזר link עד שהלינק יביא לעץ בדרגה שעדיין לא נמצאת במערך resultRoots. ולבסוף השמה של העץ הסופי ב- resultRoots ב- index שהוא דרגת העץ. 4. מעבר על מערך resultRoots והוספת כל צומת בו לרשימת השורשים של הערימה בעזרת פונקציית העזר addNodeToRootList, תוך כדי עדכון המינימום של הערימה במידת הצורך.   **סיבוכיות זמן הריצה של הפונקציה** – נסמן את מספר העצים (או מספר השורשים) ההתחלתי לפני ביצוע consolidate ב- T0 לכן יצירת currentRoots לוקח T0 זמן. גודל המערך resultRoots הוא O(log n) לכן צעדים 2 ו- 4 לוקחים O(log n) זמן. בנוסף ראינו שמספר הלינקים המקסימלי האפשרי הוא גם T0 בהרצאה לכן צעד 3 לוקח גם T0 זמן. סך הכל מכיוון שמספר העצים המקסימלי האפשרי הוא n בערימה במקרה הגרוע הפונקציה תיקח O(n) = 2 T0+2log n זמן. אמנם אם בוחרים פונקציית פוטנציאל שהיא פעמיים מספר העצים בערימה מקבלים ש-  Amortized cost = 2 T0+2log n + 2 T1-2 T0 + 2k – 2 = O(log n)  T1 מספר העצים אחרי הפעולה, k דרגת המינימום שנמחק.  T0 התבטל ויודעים שמספר הילדים מוגבל על ידי O(log n) וקיבלנו שהפונקציה מתבצעת ב- O(log n) זמן amortized. | Consolidate() |
| מעבר על כל השורשים בערימה על מנת למצוא את השורש בעל הדרגה המקסימלית והחזרתו.  **סיבוכיות זמן הריצה של הפונקציה** –בערימת פיבונ'צי אם מבצעים סדרה של n הכנסות בלבד מתקבלים n שורשים עם דרגה 0 ולכן במקרה הגרוע הפונקציה לוקחת O(n) זמן. | getMaxDegree() |
| מעלה את המשתנה הסטטי של המחלקה totalCuts ב-1 וקוראת ל- cutInternal עם node ו – parent.  **סיבוכיות זמן הריצה של הפונקציה** – cutInternal היא O(1) לכן סך הכל O(1) זמן במקרה הגרוע. | cut(HeapNode node, HeapNode parent) |
| הפונקציה מקבלת צומת עם ההורה שלו בהנחה שהם לא null ומוחקת את node מרשימת הילדים של parent. הפונקציה מורידה את הדרגה של parent ב-1 ולבסוף מוסיפה את node לרשימת השורשים של הערימה בעזרת פונקצית העזר addNodeToRootList.  **סיבוכיות זמן הריצה של הפונקציה** – שינוי מספר קבוע של משתנים וערכים + קריאה אחת ל- addNodeToRootList שלוקחת O(1) זמן כלומר סך הכל הפונקציה לוקחת O(1) זמן במקרה הגרוע. | cutInternal(HeapNode node, HeapNode parent) |
| הפונקציה בצורה רקורסיבית חותכת צמתים בעזרת cut כל עוד הצומת מסומן. היא עושה את הפעולה כלפי מעלה בעץ מילד להורה וכאשר מגיעה לצומת לא מסומן היא מסמנת אותו ומסיימת את הקריאה הרקורסיבית.  **סיבוכיות זמן הריצה של הפונקציה** - | cascadingCut(HeapNode node) |
| מחברת שני שורשים מהערימה x ו- y בהנחה שהמפתח של x קטן יותר מהמפתח של y כך ש-y יהיה ילד חדש של x. בנוסף מעלה את המשתנה הסטטי totalLinks ואת דרגת x ב-1.  **סיבוכיות זמן הריצה של הפונקציה** – שינוי מספר קבוע של מצביעים וערכים ולכן לוקחת O(1) זמן במקרה הגרוע. | link(HeapNode x, HeapNode y) |
| הוספת צומת מהערימה לרשימת השורשים של הערימה. אם אין מינימום הפונקציה מעדכנת את המינימום להיות הצומת. לבסוף מאפסת את הסימון של הצומת ל- false ומגדילה את המשתנה הסטטי numRoots ב-1.  **סיבוכיות זמן הריצה של הפונקציה** - שינוי מספר קבוע של מצביעים וערכים ולכן לוקחת O(1) זמן במקרה הגרוע. | addNodeToRootList(HeapNode node) |
| הפונקציה מחשבת ומחזירה בעזרת מספר הצמתים בערימה ובעזרת פעולות מתמטיות את הדרגה המקסימלית שיכולה להיות לצומת בערימה. לפי החסם העליון לדרגה שלמדנו בכיתה שהוא - logφ*n כאשר* φ יחס הזהב הינו בערך 1.618 לכן עיגלנו למעלה וקבענו שהדרגה המקסימלית שהפונקציה תחזיר הינה log1.6*n* עם עיגול למעלה לשלמים שגדול מהערך המקסימלי האפשרי.  **סיבוכיות זמן הריצה של הפונקציה** – מספר קבוע של פעולות כלומר O(1) זמן במקרה הגרוע. | getMaxPossibleRootDegree() |

**מדידות:**

**Sequence 1:**

Insert(m), insert(m-1), …, insert(1)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **M** | **Run-Time**  **(in miliseconds)** | **totalLinks** | **totalCuts** | **Potential** |
| **1000** | **2.24** | **0** | **0** | **1000** |
| **2000** | **3.93** | **0** | **0** | **2000** |
| **3000** | **4.34** | **0** | **0** | **3000** |

זמן הריצה האסימפטוטי של סדרת פעולות כזאת כפונקציה של m היא לינארית ב-m משום שפעולת insert היא O(1) זמן במקרה הגרוע, כלומר m פעולות כאלו יקחו O(m) זמן במקרה הגרוע.

מתבצעות 0 פעולות link ו-0 פעולות cut לכל m בסדרת הכנסות כי רק פעולת deleteMin יכולה לבצע link ורק פעולות delete או decreaseKey יכולות לבצע cut.

הפוטנציאל של המבנה כפונקציה של m הוא בדיוק m בסוף ריצת סדרת פעולות זו מכיוון שהפוטנציאל הוא מספר העצים ועוד מספר הצמתים המסומנים. אך בסדרת הכנסות מספר העצים הוא כמספר ההכנסות משום שלא בוצע שום deleteMin שמבצע links בין עצים, לכן