**מבני נתונים – פרויקט מעשי מספר 2 - ערמה d-ארית**

|  |  |
| --- | --- |
| שם: רן ערמוני  שם משתמש: ranarmoni  ת.ז - 204027312 | שם: שחף גורן  שם משתמש: shahafgoren  ת.ז - 203269931 |

**מחלקות-**

DHeap\_Item – מומשה מראש.

DHeap – מחלקה המממשת ערימה D-ארית של אובייקטים מסוג DHeap\_Item.

**המחלקה DHeap:**

**שדות –**

|  |  |
| --- | --- |
| כמות האיברים בערימה | **private** **int** size |
| כמות האיברים המקסימלית בערימה | **private** **int** max\_size |
| דרגת הערימה | **private** **int** d |
| המערך בו מוחזקת הערימה | **private** DHeap\_Item[] array |

**מתודות-**

אם לא צוין אחרת – n מייצג את כמות האיברים בערימה (שווה לערך השדה size).

חלק מניתוחי הסיבוכיות מתבססים על ההנחה שהוכחה בכיתה שגובה העץ הוא o(logn).

|  |  |  |
| --- | --- | --- |
| חתימה | סיבוכיות | הסבר |
| DHeap(**int** m\_d, **int** m\_size) | O(m\_size) | בנאי לערימה ריקה מדרגה m\_d וגודל מקסימלי m\_size. הבנאי מאתחל אובייקט ערימה עם הפרמטרים הנתונים, מאתחל מערך בגודל m\_size של אובייקטי ערימה וממלא אותו ב-null. הסיבוכיות של המתודה נובעת מאתחול הערימה ומכתיבת הערכים כ-null. |
| **public** **int** getSize() | O(1) | מחזיר את ערך השדה size |
| **public** **int** arrayToHeap(DHeap\_Item[] array1) | O(max\_size) | תחילה נצטרך להפוך את כל הערכים במערך שלנו ל-null – כלומר לבצע פעולה בזמן קבוע על max\_size ערכים – o(max\_size). לאחר מכן נעתיק את כל הערכים שאינם null ממערך הקלט למערך של הערימה – o(array1.length). כעת נבצע את האלגוריתם שנלמד בתרגול להפיכת מערך לערימה די-ארית, ע"י ביצוע heapify-down לכל האובייקטים החל מהשכבה השניה מלמטה בעץ (אותה אנחנו מוצאים בזמן קבוע – האיבר הראשון בשכבה זו הוא האבא של האיבר האחרון במערך) עד לשורש העץ. כפי שהוכח בתרגול – זה יקח o(n) כאשר n הוא מספר האיברים שאינם null במערך הקלט. סכום שלושת השלבים הוא o(max\_size) בגלל שmax-size חוסם מלמעלה את שני הגדלים האחרים – את גודל מערך הקלט, ולכן בוודאי את כמות האיברים הלא-ריקים בו. המתודה סוכמת בזמן הריצה את מספר ההשוואות שבוצעו בכל פעולת heapify-down ומחזירה מספר זה. |
| **public** **boolean** isHeap() | O(n) | המתודה רצה בלולאה על כל האיברים (למעט השורש) ובודקת האם כלל הערימה מתקיים (כלומר הם גדולים או שווים להורה שלהם, שנמצא בעזרת מתודת העזר getItemParent שרצה בזמן קבוע).  מתקיימת בדיקה שלוקחת זמן קבוע על כל איבר בערימה – כלומר o(1)\*o(n)=o(n) |
| **public** **static** **int** parent(**int** i, **int** d) | 0(1) | בחרנו למספר את איברי הערימה מ-0 (ולא מ-1 כמו בשיעורים), לכן הנוסחא לאיתור הורה של צומת היא . המתודה מקבלת אינדקס i ומחזירה את האינדקס של ההורה שלו לפי נוסחא זו. |
| **public** **static** **int** child (**int** i, **int** k, **int** d) | 0(1) | באופן דומה, הנוסחא מחזירה את האינדקס של הילד הk בעזרת הנוסחה – i\*d+k. |
|  |  |  |
| **public** **int** Insert(DHeap\_Item item) | O(logn) | האיבר מוכנס במקום האחרון ברשימה, ואז מבוצע עליו heapify-up וערך ההחזרה של heapify-up מוחזר. Heapify-up על איבר בשכבה התחתונה ביותר פועל ב- O(logn), לכן זוהי גם הסיבוכיות של insert (ההכנסה לסוף הרשימה לוקחת זמן קבוע). |
| **public** **int** Delete\_Min() | O(dlogn) | קוראת למתודה Delete(DHeap\_Item item) על שורש המינימום. הסיבוכיות נובעת מהסיבוכיות של מתודת המחיקה (ראו בהמשך) כי איתור המינימום (בעזרת Get\_Min()) לוקח זמן קבוע. |
| **public** DHeap\_Item Get\_Min() | O(1) | מחזירה את המינימום של הערימה ע"י החזרת האיבר במקום ה-0 במערך. |
| **public** **int** Decrease\_Key(DHeap\_Item item, **int** delta) | O(logn) | מחשבת את הערך החדש שהמפתח צריך לקבל, מחליפה את המפתח הקיים בחדש (בעזרת המתודות המתאימות. שתי הפעולות לוקחות זמן קבוע) ואז קוראת ל-heapifyup, שרצה ב-o(logn) (ראו סעיף). |
| **public** **int** Delete(DHeap\_Item item) | O(dlogn) | המתודה מחליפה בין האיבר האחרון במערך לאיבר אותו רוצים למחוק, מסירה את האחרון, ואז קוראת ל-heapifydowm על האיבר שהוחלף עם האיבר שמחקנו שרצה ב-o(dlogn). המתודה מחזירה את כמות ההשוואות שביצעה heapify-down. |
| **public** **void** remove(DHeap\_Item item) | O(1) | פונקציית עזר לפונקציית המחיקה. מסירה את האיבר שקיבלה כפרמטר, ומעדכנת את שדה המיקום באותו איבר להיות 1-. רצה בזמן קבוע. |
| **public** **static** **int** DHeapSort(**int**[] array1, **int** d) | O(nlogn) | תחילה הפונקציה יוצרת ערימה חדשה מדרגה d וגודל מקסימלי בהתאם לגודל מערך הקלט. לאחר מכן נבנה מערך בו כל מספר ממערך הקלט מוחלף באיבר ערימה בעל אותו מפתח (convertIntArrayToItemsArray), ואז המערך הנ"ל הופך לערימה בעזרת arrayToHeap ומוכנס לערימה שיצרנו. לאחר מכן מבוצעות בלולאה n פעולות החזרת המינימום, הכנסתו למערך הפלט, ואז מחיקתו (זמן קבוע לכל פעולה, בעזרת הפונ' הרלוונטיות). תוך כדי הריצה נסכמות פעולות ההשוואה שמבוצעות בעת ההפיכה לערימה ובעת מחיקת המינימום, ולבסוף נתון זה מוחזר.  ראו אנליזה מפורטת לסיבוכיות בפרק ניתוח המדידות. |
| **public** **static** DHeap\_Item[] convertIntArrayToItemsArray(**int**[] intArray) | O(n) | המתודה מאתחלת מערך פלט של איברי ערימה באורך מערך הקלט, ואז רצה בלולאה על מערך הקלט ועבור כל איבר מוסיפה למערך הפלט איבר ערימה חדש עם מפתח מתאים. המתודה עוברת על כל איברי מערך הקלט ועל כל אחד מהן מבצעת פעולות בזמן קבוע, לכן בסה"כ זמן הריצה של המתודה הוא o(n). |
| **public** **int** heapifyUp(DHeap\_Item item) | O(logn) | המתודה מתחילה באיבר הקלט ורצה בלולאה במעלה העץ כל עוד האיבר בקלט לא מקיים את כלל הערימה (נבדק ע"י השוואה יחידה בין האיבר לאביו). בכל איטרציה כזו האיבר מוחלף עם אביו. המתודה מבצעת לכל היותר איטרציה אחת בכל רמה בעץ, ובכל איטרציה מבצעת פעולות בזמן קבוע, לכן זמן הריצה o(logn). לבסוף מוחזרות מספר ההשוואות. |
| **public** **int** heapifyDown(DHeap\_Item item) { | O(dlogn) | המתודה מתחילה באיבר הקלט ורצה בלולאה במורד העץ כל עוד האיבר בקלט לא מקיים את כלל הערימה (נבדק ע"י השוואה בין האיבר לבנו המינימאלי ע"י קריאה ל-getMinSon שרצה בזמן o(d) ומחזירה את הבן המינימאלי ואת מספר ההשוואות שביצעה). בכל איטרציה כזו האיבר מוחלף עם בנו המינימאלי. המתודה מבצעת לכל היותר איטרציה אחת בכל רמה בעץ, ובכל איטרציה מבצעת פעולת השוואה ב-o(d) ופעולות נוספות בזמן קבוע, לכן זמן הריצה o(dlogn). לבסוף מוחזרות מספר ההשוואות שנסכמו לאורך הריצה. |
| switchItems(DHeap\_Item item1, DHeap\_Item item2) | O(1) | פונקציית עזר שמחליפה מקומות בין האיברים בקלט, תוך כדי עדכון שדות המיקום שלהם. רצה בזמן קבוע. |
| setPosition(DHeap\_Item item, **int** newPos) | O(1) | פונקציית עזר שמכניסה איבר למקום נתון במערך ומעדכנת את שדה המיקום שלו בהתאם. רצה בזמן קבוע. |
| **public** Object[] getMinSon(DHeap\_Item dad) | O(d) | הפונקציה עוברת בלולאה על ילדיו של הצומת ובודקת מי המינימאלי ע"י השוואה סדרתית. מתבצעת איטרציה אחת לכל בן, ובכל איטרציה מספר פעולות קבוע. יש לכל היותר n בנים, לכן המתודה רצה ב-o(n). תוך כדי הריצה נספרות פעולות ההשוואה, ולבסוף מוחזר מערך עם שני פריטים – מספר הפעולות שבוצעו, והבן המינימאלי. |
| **public** DHeap\_Item getItemChild(DHeap\_Item item, **int** k) { | O(1) | המתודה מאתרת את האינדקס של הבן ה-k של איבר הקלט בעזרת הפונקציה child ואז מחזירה את האיבר המתאים. רצה בזמן קבוע. |
| **public** DHeap\_Item getItemParent(DHeap\_Item item) | O(1) | המתודה מאתרת את האינדקס של האב של איבר הקלט בעזרת הפונקציה parent ואז מחזירה את האיבר המתאים. רצה בזמן קבוע. |

ניתוח המדידות:

מדידות:

* **עבור d=2**
  + מס' השוואות עבור m=1000: 16846
  + מס' השוואות עבור m=10,000: 235319
  + מס' השוואות עבור m=100,000: 3018433
* **עבור d=3**
  + מס' השוואות עבור m=1000: 16419
  + מס' השוואות עבור m=10,000: 226562
  + מס' השוואות עבור m=100,000: 2896582
* **עבור 4d=**
  + מס' השוואות עבור m=1000: 17610
  + מס' השוואות עבור m=10,000: 242803
  + מס' השוואות עבור m=100,000: 3076609

ניתוח אסימפטוטי:

בתחילה אנו מבצעים arrayToHeap: במהלך הפעולה אנחנו מבצעים O(n) פעולות heapify down.  
במהלך פעולת heapify down מבצעים d השוואות (d-1 ע"מ למצוא את הבן המינימלי ועוד השוואה עם האבא).  
כל פעולה עשוייה להמשיך בworst case עד לשורש כלומר בין 1 ל (גובה העץ).

מס' האיברים ברמה האחרונה < n

מס' האיברים ברמה הלפני אחרונה >

*לאחר מכן אנו מבצעים n פעולות delete-min כאשר כל פעולה עשויה בworst-case לקחת (h פעולות heapify down).*

וקיבלנו:

נראה כי החסם הדוק:

נבנה מערך סדרתי מ1 עד n כך שנקבל ערימה מלאה לגמרי

בפעולת הheapify down נבצע O(n) פעולות

לאחר מכן נבצע delete-min:

ברמה האחרונה יש איברים כאשר כלומר לפחות איברים

נבצע מס' מחיקות השווה לכמות האיברים ברמה האחרונה

על כל מחיקה נעביר איבר מהתחתית לראש הערימה ונבצע heapify down אשר ימשך לפחות עד הרמה הלפני אחרונה

כלומר נקבל : פעולות

כלומר חסם הדוק:

מדידות:

* **עבור d=2**
  + מס' השוואות עבור x=1: 100000
  + מס' השוואות עבור x=100: 152831
  + מס' השוואות עבור x=1000: 303431
* **עבור d=3**
  + מס' השוואות עבור x=1: 100000
  + מס' השוואות עבור x=100: 130869
  + מס' השוואות עבור x=1000: 213115
* **עבור 4d=**
  + מס' השוואות עבור x=1: 100000
  + מס' השוואות עבור x=100: 123001
  + מס' השוואות עבור x=1000: 181228

במהלך decrease-key אנחנו מבצעים heapify up במהלכו מתבצעת השוואה אחת לכל רמה כאשר הפעולה עשויה לעלות את כל גובה העץ

כלומר: לאחר n decrease-key נבצע לכל היותר nh השוואות.

ובסה"כ

נראה כי חסם זה הדוק:

נבנה ערימה ובכל שלב נפחית את המפתח של האיבר האחרון בערימה כך שהוא קטן מהאיבר המינימלי ונקבל שביצענו nh פעולות השוואה

ונקבל סה"כ לפחות:

כלומר ניתן להסיק כי decrease key רצה ב