17.03.2024

Lilach Chay
322270513
lilachchay

Orian Marmelstein
207860487
orianm

# Binomial Heap

## תרגיל מעשי 2, קורס מבני נתונים

## תיעוד הפונקציות

### מחלקת BinomialHeap :

מחלקה הממשת ערימה בינומית. לכל ערמה יש שדה גודל המייצג את מספר הצמתים בערימה, שדה ששומר את מספר העצים בערימה, וכן שני מצביעים (מסוג HeapNode) - last שמצביע לאיבר האחרון בערימה ו- min שמצביע לאיבר המינימלי בערימה. כל צומת בערימה מיוצג על ידי 2 אובייקטים שמחוברים זה לזה – HeapNode ו- HeapItem , והם מיוצגים על ידי המחלקות הפנימיות הבאות:

### ↵ מחלקת HeapItem :

מחלקה פנימית של ערימה בינומית. מייצגת item של צומת בערימה ושומרת מידע על ערכי הצומת - שדה key שמחזיק את המפתח שהתקבל מהמשתמש ושדה info שמחזיק את המידע שהתקבל מהמשתמש. בנוסף מחזיקה מצביע ל- node של הצומת. נתאר את מתודות המחלקה:

- HeapItem(int key, String info) :
  הבנאי של המחלקה – מעדכן את שדות המפתח והמידע בהתאם, וכן יוצר איבר HeapNode חדש שמתאים ל- Item – ומחבר ביניהם.
  **סיבוכיות:** $O(1)$

### ↵ מחלקת HeapNode :

מחלקה פנימית של ערימה בינומית. מייצגת node של צומת בערימה ושומרת בעיקר מצביעים – כלומר שומרת מידע על מיקום הצומת בערימה. לכל node יש שדה item שמתאים לצומת, שדה child שמצביע על הילד בעל הדרגה הגבוהה ביותר של הצומת, שדה next שמצביע על ה"אח" של הצומת בעל הדרגה המינימלית שגבוהה מהדרגה של הצומת הנוכחי (אא"כ זהו ה"אח" האחרון, במקרה הזה הוא מצביע לאח הראשון), שדה parent שמצביע על ההורה של הצומת ושדה rank שמחזיק את דרגת הצומת (שגם שווה למספר הילדים של הצומת). נתאר את מתודות המחלקה:

- getKey() :
  מחזירה את ערך המפתח ששמור ב- HeapItem עליו מצביע הצומת הנוכחי.
  **סיבוכיות:** $O(1)$

- getMinRoot(HeapNode min):
  פונקציה סטטית של המחלקה. הפונקציה מקבלת צומת בעל המפתח המינימלי בערימה שאליה הוא שייך, ומחזירה את השורש של העץ בו הוא נמצא. הפונקציה בעצם מטפלת במקרה שבו ישנם 2 צמתים שונים בעלי אותו המפתח המינימלי, והצומת שעליו מצביע השדה min חובר בפעולת link להיות הבן של הצומת השני.

**סיבוכיות:** מספר האיטרציות בפונקציה הוא לכל היותר כגובה העץ, ז"א לכל היותר דרגת השורש. בכל איטרציה מתבצעת עבודה בסיבוכיות קבועה ולכן סה"כ קיבלנו $\boldsymbol{O(\log n)}$.

- Link(HeapNode node1, HeapNode node2) :
  פונקציה סטטית של המחלקה. הפונקציה מקבלת שני צמתים מאותה הדרגה, מחברת את הצומת בעל המפתח המקסימלי מבניהם בתור הילד של הצומת בעל המפתח המינימלי, ומחזירה את השורש (הצומת בעל המפתח המינימלי).
  **סיבוכיות:** $\boldsymbol{O(1)}$

- getBro() :
  הפונקציה מתחילה מהצומת this ועוברת בלולאה על אחיו (ע"י קריאה לשדה next) עד שמגיעה אל הצומת שמצביע על הצומת שממנו התחלנו ומחזירה אותו.
  **סיבוכיות:** מספר ה"אחים" של צומת (כולל הצומת עצמו) הוא כמספר הילדים של ההורה, ז"א ה-rank של ההורה. הדרגה של כל צומת היא לכל היותר log n ולכן כך גם מספר האיטרציות המקסימלי, ובכל איטרציה הסיבוכיות היא קבועה. סה"כ קיבלנו $\boldsymbol{O(\log n)}$.

כעת, נתאר את המתודות של המחלקה הראשית BinomialHeap:

**updateHeap(int size, HeapNode last, HeapNode min, int numberOfTrees) :**

הפונקציה מעדכנת את שדות הערימה הבינומית על פי הערכים החדשים שקיבלה.

**סיבוכיות:** $\boldsymbol{O(1)}$

**Insert(int key, String info) :**

הפונקציה מכניסה לערימה צומת חדש עם המפתח key והמידע info. היא עושה זאת ע"י הכנסת הצומת החדש לערימה בתור עץ מדרגה 0, ואז משווה בין דרגת העץ הנוכחי לדרגת העץ שאחריו. אם הדרגות שוות הפונקציה שולחת את העצים לפונקציית העזר **link** של המחלקה HeapNode (מתוארת מעלה), ומשווה את דרגת העץ המאוחד החדש לדרגת העץ שאחריו, וחוזר חלילה. הפונקציה עוברת כך על העצים בלולאה עד שאין צורך בפעולות חיבור נוספות.

**סיבוכיות:** מספר העצים המקסימלי בערימה הוא $O(\log n)$ ולכן זהו מספר האיטרציות המקסימלי שהפונקציה מבצעת, כאשר בכל איטרציה מתבצעת עבודה בסיבוכיות קבועה. סה"כ $\boldsymbol{O(\log n)}$ .

**FindMin() :**

הפונקציה מחזירה את ה- HeapItem של הצומת המינימלי בערימה במידה והערימה אינה ריקה, אחרת מחזירה null.

**סיבוכיות:** $\boldsymbol{O(1)}$.

**deleteMin() :**

הפונקציה מוחקת את האיבר המינימלי בערימה, ומעדכנת את המינימום החדש. ראשית היא יוצרת ערימה חדשה מהילדים של הצומת המינימלי, ומנתקת את הצומת המינימלי וילדיו מהערימה. לאחר מכן מחפשת את השורש המינימלי החדש מבין השורשים שנותרו ומעדכנת את הערימה בהתאם. לבסוף היא קוראת לפונקציה meld המאחדת את הערימה הראשית עם ערימת הילדים.

נתאר את מתודות העזר:

- getBro():
  משמשת על מנת להשיג את הצומת הקודם – פירוט למעלה במחלקה HeapNode.
  **סיבוכיות:** $\boldsymbol{O(log\ n)}$.

- getNewMin(HeapNode node) :
  הפונקציה מקבלת מצביע לצומת, עוברת בלולאה על כל אחיו ומחזירה מצביע לצומת המינימלי ביניהם.
  **סיבוכיות:** מספר ה"אחים" של צומת (כולל הצומת עצמו) הוא כמספר הילדים של ההורה, ז"א ה-rank של ההורה. הדרגה של כל צומת היא לכל היותר log n ולכן כך גם מספר האיטרציות המקסימלי, ובכל איטרציה הסיבוכיות היא קבועה. סה"כ $\boldsymbol{O(\log n)}$.

- resetParent(HeapNode node) :
  פונקציה סטטית. הפונקציה מקבלת מצביע לצומת ועוברת בלולאה עליו ועל אחיו ע"מ לעדכן את שדה ההורה של כל אחד מהם ל- null (הופכת אותם לשורשים). משמשת ביצירת ערימת הילדים החדשה ללא השורש המינימלי.
  **סיבוכיות:** בדומה לפונקציות מעל יש מעבר על האחים של הצומת – סה"כ $\boldsymbol{O(\log n)}$.

**סיבוכיות:** במחיקת המינימום נקבל שזמן הריצה לאחר שימוש במתודות העזר יהיה סה"כ $\boldsymbol{O(\log n)}$.

**decreaseKey(HeapItem item, int diff) :**

הפונקציה מקבלת item של צומת בערימה ומספר שלם אי שלילי, מחסירה את המספר מערך המפתח של הצומת ומעדכנת את גובה הצומת ששונתה ואת הצומת המינימלי של הערימה בהתאם כך שערימת המינימום תישאר ערימה תקינה.

נתאר את מתודות העזר:

- heapifyUp(HeapItem pointer) :
  הפונקציה מקבלת item ו"מפעפעת" אותו במעלה הערימה כל עוד המפתח של ההורה של הצומת גדול מהמפתח של הצומת עצמו, או עד שהצומת הפך לשורש של הערימה.
  **סיבוכיות:** מספר האיטרציות של הפונקציה יהיה כעומק העץ - עומק עץ של ערימה בינומית שווה לדרגת השורש, והדרגה המקסימלית של שורש כנ"ל היא log n. בכל איטרציה מתבצעת עבודה קבועה ולכן סיבוכיות הזמן סה"כ תהיה $\boldsymbol{O(\log n)}$.

**סיבוכיות:** הפונקציה קוראת לפונקציית העזר ושאר העבודה קבועה, לכן זמן הריצה יהיה $\boldsymbol{O(\log n)}$.

**Delete(HeapItem item):**

הפונקציה מקבלת HeapItem של צומת שנמצא בערימה ומוחקת אותו. היא עושה זאת על ידי הפחתת הערך של המפתח של הצומת כך שהוא יהיה הצומת המינימלי (על ידי קריאה ל- decreaseKey) ולאחר מכן מוחקת את המינימום ע"י קריאה ל – deleteMin.

**סיבוכיות:** סה"כ סיבוכיות הזמן תהיה סכום הסיבוכיות של הפונקציות הנקראות, כלומר $\boldsymbol{O(\log n)}$.

**Meld(BinomialHeap heap2):**

הפונקציה מקבלת ערימה בינומית נוספת, וממזגת את שתי הערימות לערימה בינומית חוקית אחת.
ראשית, הפונקציה קוראת למתודת העזר makeForest על שתי הערימות, על מנת לאחד אותן לערימה אחת לא חוקית בה נמצאים כל העצים, מסודרים בסדר דרגות עולה. נשים לב שייתכן שכעת ישנם 2 עצים מאותה דרגה בערימה.
בשלב השני הפונקציה עוברת על העצים בערימה, שולחת כל שני צמתים מאותה דרגה לפונקציה link ומחליפה אותם בצומת עם המפתח המינימלי ביניהם, כאשר הצומת השני הפך לילד של הצומת החדשה.
אם ישנם 3 צמתים מאותה דרגה עקב קריאות קודמות ל-link, נדלג כדי לאחד באיטרציה הבאה את שני העצים האחרונים מאותה הדרגה, על מנת לשמור על סדר דרגות עולה.
בסוף התהליך מתקבלת ערימה חוקית בה נמצאים הצמתים משתי הערימות הבינומיות.

נתאר את מתודות העזר:

- makeForest(BinomialHeap heap2):
  הפונקציה מייצרת ערימה בינומית חדשה, עוברת בלולאה על העצים ב-2 הערימות הבינומיות ומכניסה אותם לערימה החדשה לפי סדר הדרגות שלהם. לאחר מכן הפונקציה מעדכנת את הערימה הראשית this להיות הערימה המאוחדת.
  **סיבוכיות:** הפונקציה עוברת על 2 הערימות פעם אחת ולכן סה"כ $2O(\log n) = \boldsymbol{O(\log n)}$.

- addTree(HeapNode node):
  מתודת עזר של המתודה makeForest. הפונקציה מוסיפה לערימה this את הצומת node בתור העץ האחרון, וכן מעדכנת את גודל הערימה, המינימום ומספר העצים בהתאם.
  **סיבוכיות:** $\boldsymbol{O(\log n)}$.

- Link(HeapNode node1, HeapNode node2) :
  פונקציה של המחלקה HeapNode ולכן מפורטת למעלה. מחברת שני צמתים בצורה חוקית.
  **סיבוכיות:** $\boldsymbol{O(1)}$

- getMinRoot(HeapNode min):
  פונקציה של המחלקה *HeapNode* ולכן מפורטת למעלה. מוודאת שהמינימום הוא אכן שורש.
  **סיבוכיות:** $\boldsymbol{O(\log n)}$.

**סיבוכיות:** סה"כ נסכום את הסיבוכיות של *makeForest* וכן *getMinRoot* שנקראים רק פעם אחת, כל אחג בסיבוכיות לוגריתמית. בנוסף נסכום את כמות האיטרציות על העץ המשותף – כך שבכל איטרציה קוראים ל- *Link* שהוא בסיבוכיות קבועה. כמות האיטרציות היא לכל היותר ככמות העצים בערימה המאוחדת ולכן סה"כ $\boldsymbol{O(\log n)}$.

**Size():**

הפונקציה מחזירה את מספר הצמתים בערימה על ידי החזרת ערך השדה size של הערימה.

**סיבוכיות:** $O(1)$.

**Empty():**

הפונקציה מחזירה true אם הערימה ריקה (בודקת אם מספר הצמתים בערימה הוא אפס) ו-false אחרת.

**סיבוכיות:** $O(1)$.

**numTrees():**

הפונקציה מחזירה את מספר העצים בערימה על ידי החזרה של ערך השדה שהוספנו – numberOfTrees.

**סיבוכיות:** $O(1)$.

## חלק תיאורטי – ניסויים

### שאלה 1 – תוצאות:

**ניסוי ראשון:**

| מספר סידורי i | זמן ריצה (מילישניות) | מספר החיבורים הכולל | מספר העצים בסיום | סכום דרגות הצמתים שמחקנו |
|---|---|---|---|---|
| 1 | 8 | 723 | 5 | 0 |
| 2 | 1 | 2182 | 4 | 0 |
| 3 | 3 | 6555 | 5 | 0 |
| 4 | 10 | 19675 | 7 | 0 |
| 5 | 26 | 59040 | 8 | 0 |
| 6 | 56 | 177134 | 12 | 0 |

**ניסוי שני:**

| מספר סידורי i | זמן ריצה (מילישניות) | מספר החיבורים הכולל | מספר העצים בסיום | סכום דרגות הצמתים שמחקנו |
|---|---|---|---|---|
| 1 | 5 | 3482 | 5 | 3123 |
| 2 | 2 | 12148 | 4 | 11059 |
| 3 | 15 | 41813 | 5 | 38538 |
| 4 | 30 | 141053 | 7 | 131219 |
| 5 | 76 | 469121 | 8 | 439605 |
| 6 | 187 | 1551788 | 12 | 1463227 |

**ניסוי שלישי:**

| מספר סידורי i | זמן ריצה (מילישניות) | מספר החיבורים הכולל | מספר העצים בסיום | סכום דרגות הצמתים שמחקנו |
|---|---|---|---|---|
| 1 | 5 | 723 | 5 | 697 |
| 2 | 2 | 2182 | 5 | 2156 |
| 3 | 4 | 6555 | 5 | 6529 |
| 4 | 19 | 19675 | 5 | 19649 |
| 5 | 69 | 59040 | 5 | 59014 |
| 6 | 87 | 177134 | 5 | 177108 |

## שאלה 2 – ניתוח התוצאות:

**ניסוי ראשון:**

הוכחנו בכיתה שסיבוכיות זמן הריצה של insert היא $O(\log n)$ במקרה הגרוע. בניסוי זה אנחנו מכניסים $n$ צמתים בזה אחר זה, ולכן הסיבוכיות במקרה הגרוע תהיה $O(n\log n)$.

נראה שניתן למצוא חסם הדוק יותר – נשים לב שהכנסת צומת לערימה בינומית שקול להוספת אחד למונה בינארי כפי שראינו בתרגול, שכן כאשר מספר הצמתים בעץ הוא זוגי - לא קיים בערימה עץ של צומת בודדת, ולכן הוספת צומת בודדת לערימה שקולה להוספת אחד למונה בינארי כאשר הביט הראשון בו הוא 0, וכל חיבור בין עצים (link) שקול להפיכת 1 במונה הבינארי ל-0, ולכן כמות העבודה בפועל היא כמות הפעמים שחיברנו עצים מאותה הדרגה. כמות הפעמים שחיברנו עצים מדרגה 0 היא $\frac{n}{2}$ (כל פעם שמספר הצמתים בעץ הפך לזוגי), עצים מדרגה 1 חיברנו $\frac{n}{4}$ פעמים וכו', ז"א הסיבוכיות בפועל היא $\frac{n}{2}+\frac{n}{4}+\cdots=\boldsymbol{O(n)}$.

נשים לב שמתקיימת המשוואה: $\boldsymbol{n=numOfTress+countOfLink}$ , כאשר $n$ הוא מספר הצמתים שהוכנסו. משוואה זו הגיונית משום שכל צומת שהכנסנו לערימה נכנס תחילה כעץ מדרגה אפס, ז"א ללא פעולות link מספר העצים היה גם הוא בדיוק $n$, ובכל פעולת לינק אחת (שהוסיפה אחד לצד הימני של המשוואה) "איבדנו" עץ אחד (והחסרנו אחד מהצד הימני של המשוואה), ולכן המשוואה נשארת מאוזנת.

**ניסוי שני:**

למדנו בכיתה שסיבוכיות הזמן של הפעולות insert ו- deleteMin היא $O(\log n)$. אנחנו מבצעים סדרה של $\frac{3n}{2}$ פעולות כנ"ל, ולכן חסם עליון יהיה $O(n\log n)$. נוכיח שחסם תחתון הוא $\Omega(\mathrm{n}\log n)$ בעזרת רדוקציה:

למדנו בכיתה שלא ניתן למיין מערך באורך $O(n)$ בסיבוכיות $o(n\log n)$.

נניח שניתן לבצע סדרה של $n$ הכנסות ו-$\frac{n}{2}$ מחיקות מינימום ב-$o(\mathrm{n}\log n)$, אזי ניתן יהיה להכניס לערימה את איברי מערך באורך $\frac{n}{2}$ ולאחר מכן להכניס לערימה $\frac{n}{2}$ איברים נוספים שכולם גדולים מהאיבר המקסימלי במערך (סה"כ $n$ הכנסות), ואז לבצע deleteMin $\frac{n}{2}$ פעמים, כאשר בכל פעם נכניס למערך חדש את האיבר שנמחק, ובכך נמיין מערך באורך $O(n)$ בסיבוכיות $o(n\log n)$ – וקיבלנו סתירה! לכן, סיבוכיות הזמן של סדרת הפעולות הנ"ל היא $\boldsymbol{\Theta(\mathrm{n}\log n)}$.

נשים לב שמתקיים:

$$\boldsymbol{n - numberOfDeleteMin + sumOfDeletedRanks = numberOf\ tress + countOfLink}$$

(נשים לב שכאשר מציבים את $sumOfDeletedRanks$ וגם את $numberOfDeleteMin$ להיות 0 מתקבלת המשוואה מהניסוי הראשון)

משוואה זו הגיונית משום שאם נמחק צומת מדרגה k, על פי המימוש של deleteMin אנחנו בעצם מוסיפים לערימה את עצי הילדים של אותה הצומת, שהם מדרגות $0, 1, \dots k-1$, (סה"כ k עצים) ולכן צד ימין של המשוואה יגדל ב – k-1 (הוספנו k עצים אך איבדנו עץ אחד מדרגה k), וייתכן שיבוצעו לינקים, אך נשים לב שעל כל link שיבוצע נגדיל את countOfLink ב-1 ונוריד אחד מ-numberOfTrees, כלומר סה"כ קיבלנו שצד ימין יגדל ב-k-1.

בצד שמאל, אנחנו נוסיף אחד ל-numberOfDeleteMin ונוסיף k ל- sumOfDeletedRanks, ולכן גם צד שמאל גדל ב – k-1 - כלומר המשוואה נשארת מאוזנת.

**<u>ניסוי שלישי:</u>**

למדנו בביתה שסיבוכיות הזמן של הפעולות insert ו- deleteMin היא $O(\log n)$. אנחנו מבצעים סדרה של $O(n)$ פעולות כנ"ל, ולכן סיבוכיות זמן הריצה תהיה $\boldsymbol{O(n \log n)}$.

נשים לב שכפי שלמדנו בביתה, המפתחות שנכנסו ראשונים נמצאים בעצים הימניים יותר (בעלי הדרגות הגבוהות יותר), ולכן כאן הצמתים בעלי המפתחות הקטנים (שנכנסו בסוף) נמצאים בעצים השמאליים, ובפרט המפתח המינימלי שייך לצומת שהיא השורש של העץ השמאלי ביותר. לכן, לאחר כל פעולת deleteMin לא מתבצע אף לינק (לא קיימים בערימה עצים בעלי דרגות ששוות לדרגות ילדי המינימום, אחרת הם היו שמאליים ממנו בסתירה להסבר).

בניסוי הזה מתקיים:

$$\boldsymbol{26 = sumOfDeletedRanks - countOfLink}$$

בכל קריאה ללינק אנחנו יוצרים "קשת" בין 2 צמתים, ז"א ש- countOfLink בסוף התהליך יהיה מספר הקשתות שנוצרו בין צמתים. כפי שהסברנו קודם, עקב סדר הכנסת הצמתים, לאחר פעולת deleteMin לא מתבצע אף לינק - ולכן כל הקשתות נוצרו רק בהכנסת צמתים חדשים.

בעת מחיקת צומת, אנחנו "מנתקים" את כל הקשתות שהיו לה עם ילדיה, סה"כ קשתות כדרגת הצומת, ולכן sumOfDeletedRanks הוא מספר הקשתות שניתקנו בכל פעולת מחיקה.

נקבל סה"כ שההפרש בין countOfLinks לבין sumOfDeletedRanks הוא מספר הקשתות שנותרו בין צמתים.

בסוף סדרת הפעולות אנחנו נשארים עם 31 צמתים שמחולקים ל-5 עצים (מדרגות 0, 1, 2, 3 ו-4), כאשר כל עץ הוא רכיב קשירות חסר מעגלים (עץ). מספר הקשתות בכל עץ שווה למספר הצמתים בו פחות אחד, ולכן מספר הקשתות בכל הערימה הוא:

$$2^0 - 1 + 2^1 - 1 + 2^2 - 1 + 2^3 - 1 + 2^4 - 1 = 26$$

ואכן, קיבלנו שסה"כ מתקיימת המשוואה: $26 = sumOfDeletedRanks - countOfLink$.