תיעוד פרוייקט מעשי 2 בקורס מבני נתונים – מימוש ערימת פיבונאצ'י

לירון כהן, lironcohen3, 207481268
יובל מור, yuvalmor, 209011543

המחלקה FibonacciHeap

המחלקה מממשת ערימת פיבונאצ'י, ובה כל אחד מצמתי העץ מממש מופע ממחלקת HeapNode, אודותיה נפרט בהמשך.

שדות FibonacciHeap

- שדות סטטיים:
  - Double GOLDEN_RATIO – קבוע סטטי אשר ערכו הוא יחס הזהב (עבור חישובים).
  - Int totalLinks – שדה סטטי השומר את כמות פעולות ה-link שבוצעו בריצת התוכנית.
  - Int totalCuts – שדה סטטי השומר את כמות פעולות ה-cut שבוצעו בריצת התוכנית.
- שדות מופע:
  - HeapNode min – מצביע לצומת עם הערך המינימלי בערימה.
  - HeapNode first – מצביע לצומת הראשון בערימה (שורש העץ הכי שמאלי)
  - Int size – מספר הצמתים בערימה כולה.
  - Int treeNum – מספר העצים בערימה.
  - Int markedNum – מספר הצמתים המסומנים בערימה.

מתודות FibonacciHeap

FibonacciHeap()

המתודה בונה ערימה ריקה, כלומר מאתחלת את המינימום של הערימה והאיבר הראשון שלה להיות null. בנוסף, מאתחלת את שדות ה-size, treeNum ו-markedNum להיות 0.
סיבוכיות המתודה היא $O(1)$ כיוון שהיא קובעת ערכים של שדות.

isEmpty()

המתודה מחזירה אמת אם ערך השדה first של הערימה הוא null, ושקר אחרת.
סיבוכיות המתודה היא $O(1)$ כיוון שהיא בודקת ערך של שדה.

Insert(int key)

המתודה יוצרת צומת חדש עם המפתח הנתון ומכניסה אותו לתחילת הערימה (בתור האח הקודם לאיבר הראשון בערימה).

המתודה מעדכנת את מצביעי האחים הרלוונטיים ואת מצביע הצומת הראשון של הערימה.

בנוסף, המתודה מגדילה את ערכי השדות size ו-treeNum ב-1 ומעדכנת את ערך השדה min במידת הצורך. המתודה מחזירה את הצומת שנוצר.

סיבוכיות המתודה היא $O(1)$ כיוון שהיא מעדכנת שדות ומבצעת בדיקת תנאי פשוטה.

InsertKMin(HeapNode n) – מתודת עזר

המתודה זהה למתודה insert, אבל מקבלת צומת ומשתמשת בבנאי המתאים (נוצרה לשם פעולת kmin).

סיבוכיות המתודה היא $O(1)$ כיוון שהיא מעדכנת שדות ומבצעת בדיקת תנאי פשוטה.

InsertTree(HeapNode x) – מתודת עזר

המתודה מכניסה את הצומת שהתקבל לתחילת הערימה (בתור האח הקודם לאיבר הראשון בערימה).

המתודה מעדכנת את מצביעי האחים הרלוונטיים ואת מצביע הצומת הראשון של הערימה.

בנוסף, המתודה מגדילה את ערך השדה treeNum ב-1 ומעדכנת את ערך השדה min במידת הצורך.
המתודה מחזירה את הצומת.

סיבוכיות המתודה היא $O(1)$ כיוון שהיא מעדכנת שדות ומבצעת בדיקת תנאי פשוטה.

Link(HeapNode x, HeapNode y) – מתודת עזר

המתודה מקבלת שני צמתים (שהם שורשים של תתי עצים מאותו הסדר) ומקשרת ביניהם באמצעות השוואה בין המפתחות שלהם, כפי שנלמד בכיתה. המתודה מעדכנת את המצביעים הרלוונטיים ואת השדות treeNum ו-totalLinks.

המתודה מחזירה את הצומת העליון מבין השניים (כלומר בעל המפתח הנמוך יותר).

סיבוכיות המתודה היא $O(1)$ כיוון שהיא בודקת תנאים ומעדכנת שדות.

ClearHeap() – מתודת עזר

המתודה מנקה את ערכי השדות first, min ו-treeNum של הערימה (לצורך הכנסה ממתודת fromBuckets אודותיה יפורט בהמשך). סיבוכיות המתודה היא $O(1)$ כיוון שהיא מעדכנת שדות.

ToBuckets() – מתודת עזר

המתודה מכניסה את עצי הערימה למערך עצים (המיוצגים מחשבתית ע"י הדליים שהוצגו בכיתה), ומאחדת באמצעות מתודת העזר link את תתי העצים בעלי אותה דרגה.

המתודה מחזירה את מערך העצים כך שבכל דרגה יש לכל היותר עץ יחיד.

הראינו בהרצאה שסיבוכיות המתודה היא $O(n)$ WC וכן $O(\log(n))$ Amortized.

FromBuckets(HeapNode[] B) – מתודת עזר

המתודה מקבלת מערך עצים אשר נוצרו ע"י מתודת ToBuckets.

המתודה מנקה את שדות הערימה באמצעות מתודת העזר clearHeap, עוברת על המערך ומכניסה את העצים לערימה בסדר הנכון באמצעות מתודת העזר insertTree.

סיבוכיות המתודה היא $O(\log(n))$ כיוון שהיא עוברת על מערך בגודל $O(\log(n))$ וקוראת למתודות עזר הפועלות בסיבוכיות $O(1)$.

Consolidate() – מתודת עזר

המתודה קוראת למתודות העזר toBuckets ו-FromBuckets אחת אחרי השנייה.

סיבוכיות המתודה היא $O(n)$ WC וכן $O(\log(n))$ Amortized.

removeMin() – מתודת עזר

המתודה מוחקת את הצומת המינימלי בעץ.

אם לצומת המינימלי יש ילדים, מעדכנת את המצביעים הרלוונטיים כך שילדיו ייכנסו במקומו כשורשים של הערימה. בנוסף, המתודה מעדכנת את שדה ה-first במידת הצורך.

אם לצומת המינימלי אין ילדים, המתודה "מדלגת" (באמצעות שינוי מצביעים) על הצומת המינימלי בתוך רשימת האחים שלו ומעדכנת את שדה ה-first במידת הצורך.

המתודה מוחקת את הצומת המינימלי ושמה צומת מינימלי אחר במקומו (יעודכן לצומת הנכון לאחר קריאה למתודה updateMin) ומקטינה את שדה ה-size של הערימה ב-1.

סיבוכיות המתודה היא $O(1)$ מכיוון שהמתודה מבצעת שינויי מצביעים, בדיקות תנאי פשוטות וחישובים פשוטים.

meld()

המתודה מקבלת ערימת פיבונאצ'י ומאחדת את הערימה הנוכחית איתה (האיחוד נעשה "מימין").

המתודה מעדכנת את השדות min, size, markedNum ו-treeNum של הערימה.

בנוסף, המתודה מעדכנת את המצביעים הרלוונטיים.

סיבוכיות המתודה היא $O(1)$ כיוון שהיא מעדכנת ערכי שדות ומבצעת בדיקות תנאי פשוטות.

updateMin() – מתודת עזר

המתודה עוברת על שורשי הערימה ומעדכנת את ערך השדה min של הערימה לערך המינימלי מבין השורשים.

נציין שעקב תכונות הערימה הערך המינימלי יהיה חייב להימצא באחד השורשים.

סיבוכיות המתודה היא $O(\log(n))$ מכיוון שהמתודה מתבצעת לאחר קריאה למתודת העזר consolidate אשר מבטיחה קיום של לכל היותר עץ יחיד מכל דרגה, ובסה"כ עבור n איברים בערימה נקבל לכל היותר $\log(n)$ דרגות.

deleteMin()

המתודה קוראת למתודות העזר removeMin, meld, consolidate, updateMin בזו אחר זו.

סיבוכיות המתודה היא $O(n)$ WC וכן $O(\log(n))$ Amortized, כיוון שזוהי הסיבוכיות הגרועה ביותר של מתודות העזר שנקראות.

findMin()

המתודה מחזירה את ערך השדה min של הערימה.
סיבוכיות המתודה היא $O(1)$ כיוון שהיא מחזירה ערך של שדה.

size()

המתודה מחזירה את ערך השדה size של הערימה.
סיבוכיות המתודה היא $O(1)$ כיוון שהיא מחזירה ערך של שדה.

countersRep()

המתודה עוברת על שורשי הערימה ומעדכנת מערך מונים בהתאם לדרגות העצים בערימה.

סיבוכיות המתודה במקרה הגרוע היא $O(n)$ מכיוון שאם יתבצעו n פעולות insert ללא deleteMin, נקבל n שורשים בודדים (n עצים מדרגה 0) ונצטרך לעבור על כולם ולעדכן את המערך.

delete(HeapNode x)

המתודה מוחקת את הצומת מהעץ ע״י קריאה למתודה decreaseKey (עבור delta=∞) ולאחר מכן קריאה ל-deleteMin.

סיבוכיות המתודה היא WC $O(n)$ וכן Amortized $O(\log(n))$, כיוון שזוהי הסיבוכיות הגרועה ביותר של מתודות העזר שנקראות.

Cut(HeapNode x, HeapNode y) – מתודת עזר

המתודה מקבלת צומת x ואת ההורה שלה y, וחותכת את הקשת ביניהן כפי שראינו בכיתה.

המתודה מעדכנת את המצביעים הרלוונטיים, מכניסה את תת העץ ששורשו x לערימה ומעדכנת את השדות markedNum ו-totalCuts בהתאם.

סיבוכיות המתודה, כפי שראינו בכיתה היא $O(1)$.

cascadingCut(HeapNode x, HeapNode y) – מתודת עזר

המתודה חותכת את x מההורה שלו, y. אם צריך, פועלת באופן רקורסיבי עד אשר ההורה אינו מסומן.

המתודה קוראת למתודת העזר cut, ואם ההורה אינו null ואינו מסומן, מעדכנת אותו להיות מסומן ומסיימת.

אם ההורה מסומן, פועלת רקורסיבית על ההורה וההורה שלו.

סיבוכיות המתודה WC היא $O(\log(n))$ מכיוון שבמקרה הגרוע צריך לבצע כמות חיתוכים כעומק העץ, כלומר $O(\log(n))$ כאשר כל חיתוך נעשה בסיבוכיות $O(1)$.

decreaseKey(HeapNode x, int delta)

המתודה מקבלת צומת וערך delta, מקטינה את מפתח הצומת ב-delta ומבצעת חיתוכים במידת הצורך באמצעות מתודת העזר cascadingCut.

סיבוכיות המתודה במקרה הגרוע היא $O(\log(n))$ וכפי שראינו בכיתה, סיבוכיות ה-$amortized$ היא $O(1)$.

potential()

המתודה מחזירה את פוטנציאל הערימה. כלומר את סכום ערך השדה treeNum ופעמיים ערך השדה markedNum.

סיבוכיות המתודה היא $O(1)$ מכיוון שהיא מחזירה ערכים של שדות ומבצעת פעולה חשבונית פשוטה.

totalLinks()

המתודה מחזירה את מספר פעולות ה-$link$ שבוצעו מתחילת התוכנית.

המתודה מחזירה את ערך השדה הסטטי $totalLinks$ ולכן סיבוכיותה $O(1)$.

totalCuts()

המתודה מחזירה את מספר פעולות ה-cut שבוצעו מתחילת התוכנית.

המתודה מחזירה את ערך השדה הסטטי $totalCuts$ ולכן סיבוכיותה $O(1)$.

kMin(FibonacciHeap H, int k)

המתודה מקבלת ערימה בינומית ובה עץ בינומי יחיד ומספר k, ומחזירה מערך ממויין של k המפתחות הקטנים ביותר בערימה.

ראשית המתודה מאתחלת ערימת עזר ומערך שלמים.

המתודה ממלאת את מערך השלמים שיוחזר ע"י ביצוע הפעולות הבאות:

- אם לצומת המינימלי האחרון שנמצא יש ילדים, מכניסה את הילדים שלו לערימת העזר (תוך שימוש בבנאי המתאים ובמתודה insertKMin).
- המתודה קוראת למתודת findMin של ערימת העזר (וכך מוצאת את הצומת המינימלי בערימת העזר, שלו מפתח זהה לצומת המינימלי בערימה המקורית) ומכניסה את המפתח שלו למערך השלמים.
- המתודה מעדכנת את משתנה ההורה (כלומר המינימום הנוכחי) לצומת המקורי המתאים ממערך הצמתים (שנקרא באמצעות שדה של הצומת המתאים בערימת העזר).
- המתודה מבצעת deleteMin על ערימת העזר.

סיבוכיות המתודה היא $O(kDeg(H))$. המתודה מבצעת את הפעולות שהוצגו לעיל k פעמים (על מנת למלא את מערך השלמים שהוא בגודל k). בכל איטרציה, מוכנסים ילדי המינימום הנוכחי, שמספרם הוא לכל היותר Deg(H). בנוסף, מתבצעת פעולת deleteMin אשר סיבוכיותה $O(\log(n)) = O(\deg(H))$.

המחלקה HeapNode

המחלקה מממשת צומת בערימת פיבונאצ'י לפי הנלמד בכיתה.

שדות HeapNode

- Int key – המפתח של הצומת
- Int rank – דרגת הצומת (מספר הילדים של הצומת)
- Int mark – האם הצומת מסומן או לא.
- HeapNode firstChild – מצביע לבן השמאלי ביותר של הצומת
- HeapNode prevBro – מצביע לאח השמאלי של הצומת
- HeapNode nextBro – מצביע לאח הימני של הצומת
- HeapNode parent – מצביע להורה של הצומת
- HeapNode KMinNode – הצומת המקבילה, לשם מימוש פעולת kMin

מתודות HeapNode

נציין שכל מתודות המחלקה מבצעות בדיקות פשוטות, אחזורים והשמות לשדות ולכן מתבצעות ב-$O(1)$.

HeapNode(int key)

המתודה בונה עצם מסוג HeapNode ומאתחלת את המפתח שלו למפתח שהתקבל כארגומנט.

בנוסף, המתודה מגדירה את מצביעי האחים של הצומת כצומת עצמו.

getKey()

המתודה מחזירה את המפתח של הצומת.

setKey(int key) – מתודת עזר

המתודה מגדירה את שדה המפתח של הצומת כערך שהתקבל כארגומנט.

getRank() – מתודת עזר

המתודה מחזירה את הדרגה של הצומת (מספר הילדים שלו).

setRank(int rank) – מתודת עזר

המתודה מגדירה את שדה הדרגה של הצומת כערך שהתקבל כארגומנט.

getMark() – מתודת עזר

המתודה מחזירה את שדה הסימון של הצומת (ביט שערכו 1 אם הוא מסומן ו-0 אחרת).

setMark(int mark) – מתודת עזר

המתודה מגדירה את שדה הסימון של הצומת כערך שהתקבל כארגומנט.

getFirstChild() – מתודת עזר

המתודה מחזירה מצביע לבן השמאלי ביותר של הצומת.

setFirstChild(HeapNode child) – מתודת עזר

המתודה מגדירה את הבן השמאלי ביותר של הצומת כערך שהתקבל כארגומנט.

getPrevBro() – מתודת עזר

המתודה מחזירה מצביע לאח השמאלי של הצומת (באופן מעגלי).

setPrevBro(HeapNode prevBro) – מתודת עזר

המתודה מגדירה את האח השמאלי של הצומת כערך שהתקבל כארגומנט.

getNextBro() – מתודת עזר

המתודה מחזירה מצביע לאח הימני של הצומת (באופן מעגלי).

setNextBro(HeapNode nextBro) – מתודת עזר

המתודה מגדירה את האח הימני של הצומת כערך שהתקבל כארגומנט.

getParent() – מתודת עזר

המתודה מחזירה מצביע להורה של הצומת.

setParent(HeapNode parent) – מתודת עזר

המתודה מגדירה את ההורה של הצומת כערך שהתקבל כארגומנט.

חלק המדידות

שאלה 1

m	Run-Time (in milliseconds)	totalLinks	totalCuts	potential
1024	2	1023	19	20
2048	5	2047	21	22
4096	8	4095	23	24

סעיף א'

זמן הריצה האסימפטוטי של סדרת פעולות זו כפונקציה של m הוא $O(m)$.

ראשית, נבחן את המתודות שרצות ואת זמן הריצה האסימפטוטי שלהן:

- $m+1$ פעולות $insert$ – $m+1$ פעולות שלוקחות $O(1)$ זמן, כלומר זמן ריצה של $O(m)$.
- פעולת $delete$ min – במקרה זה אנו נמצאים במקרה הגרוע (רשימה מקושרת של צמתים בודדים) ולכן זמן הריצה הוא $O(m)$.
- $\log(m)+1$ פעולות $decrease\ key()$ - $\log(m)$ פעולות שלוקחות $O(\log(m))$ זמן, כלומר זמן ריצה של $O(\log^2(m))$.

ובסך הכל נקבל זמן ריצה אסימפטוטי של $O(m)$.

ניתן לראות מהטבלה שהכפלה פי 2 של m מובילה בקירוב להכפלה פי 2 של זמן הריצה, תופעה המאפיינית ריצה לינארית. נציין שמכיוון שמדובר במילישניות ועקב התלות בעומס על המחשב בו הרצנו את הקוד, יש לקחת את תוצאות זמן הריצה בעירבון מוגבל.

סעיף ב'

במהלך סדרת הפעולות מתבצעות $O(m)$ פעולות $link$. פעולות אלו מתבצעות בזמן הקריאה הבודדת לפעולת $delete$ min, ובזמן זה הערימה היא רשימה מקושרת של m צמתים בודדים, ולכן בתהליך ה-$consolidate$, יתבצעו $O(m)$ קישורים על מנת לקשרם לעץ בינומי.
ניתן לראות בטבלה כי אכן מדובר ב-$O(m)$ פעולות $link$.

במהלך סדרת הפעולות מתבצעות $O(\log(m))$ פעולות cut. פעולות אלו מתבצעות בזמן קריאה לפעולת $decrease\ key$. מלמה 1 שראינו בהרצאה, $\log(m)$ פעולות ה-$decrease\ key$ הראשונות שביצענו יבצעו לכל היותר $2\log(m)$ פעולות cut. לכן, יתבצעו $O(\log(m))$ פעולות cut.
ניתן לראות בטבלה כי אכן מדובר ב-$O(\log(m))$ פעולות cut.

סעיף ג'

עלות פעולת $decrease\ key$ תלויה באופן לינארי במספר פעולות ה-cut שמתבצעות תחת $cascading\ cuts$. במקרה הגרוע, נבצע $decrease\ key$ לעלה וכל הוריו עד השורש יהיו מסומנים. לכן, נבצע מספר פעולות cut כעומק העץ, כלומר $O(\log(m))$. מכאן, עלות פעולת ה-$decrease\ key$ היקרה ביותר תהיה $O(\log(m))$.

שאלה 2

m	Run-Time (in milliseconds)	totalLinks	totalCuts	potential
1000	10	1891	0	6
2000	14	3889	0	6
3000	17	5772	0	7

סעיף א'

זמן הריצה האסימפטוטי של סדרת פעולות זו כפונקציה של m הוא $O(m\log(m))$.

נבחן את המתודות שרצות ואת זמן הריצה האסימפטוטי שלהן:

- m פעולות $m - insert$ פעולות שלוקחות $O(1)$ זמן, כלומר זמן ריצה של $O(m)$.
- $\frac{m}{2}$ פעולות $-delete\ min$
  - פעולת $delete\ min$ הראשונה לוקחת $O(m)$
  - $\frac{m}{2} - 1$ הפעולות האחרות חסומות ע"י $O(\log(m))$ כל אחת (מכיוון שלאחר הפעלת הפעולה הראשונה מספר העצים בערימה חסום ע"י $O(\log(m))$). כלומר, זמן הריצה של חלק זה הוא $O(mlog(m))$.

ובסך הכל נקבל זמן ריצה אסימפטוטי של $O(mlog(m))$.

ניתן לראות שהטבלה תואמת בקירוב את ההערכה, תחת העירבון המוגבל שצוין לעיל.

סעיף ב'

במהלך סדרת הפעולות מתבצעות $O(mlog(m))$ פעולות $link$. פעולות אלו מתבצעות בזמן הקריאות לפעולת $delete$ min.

בקריאה הראשונה, הערימה היא רשימה מקושרת של m צמתים בודדים, ולכן בתהליך ה-$consolidate$, יתבצעו $O(m)$ קישורים על מנת לקשרם לעץ בינומי.

עבור הקריאות הבאות, ראינו בהרצאה שכמות פעולות ה-$link$ (סומן כ-L_i) בכל קריאה חסום ע"י $O(\log(m))$ ולכן בסך הכל עבור קריאות אלו נקבל חסם של $O(mlog(m))$.

בסך הכל מתבצעות $O(m) + O\big(mlog(m)\big) = O(mlog(m))$ פעולות $link$.

ניתן לראות בטבלה כי אכן מדובר בכ-$O(mlog(m))$ פעולות $link$.

במהלך סדרת הפעולות מתבצעות 0 פעולות cut, זאת מכיוון שלא מתבצעת קריאה לפעולה $decrease\ key$.

<u>סעיף ג'</u>

ראשית, מכיוון שלא מתבצעות פעולות cut (ולכן לא מתבצעים סימונים בצמתי העץ), הפוטנציאל של המבנה הוא מספר העצים במבנה.

לאחר הרצת סדרת הפעולות, בערימה ישנם $\frac{m}{2}$ צמתים (כיוון שהוכנסו m צמתים ופעולת $delete\ min$ נקראה $\frac{m}{2}$ פעמים). ראינו בהרצאה שמספר העצים במבנה הוא לכל היותר אורך הייצוג הבינארי של מספר הצמתים בערימה. אורך הייצוג הבינארי לוגריתמי ביחס למספר ולכן נקבל שמספר העצים הוא $O\left(\log\left(\frac{m}{2}\right)\right)$ כלומר $O(\log(m))$.