

מבנה נתונים – פרויקט מספר 2 – ערמות מתקדמות

הקדמה

בתרגיל זה שני חלקים:

1. חלק מעשי: שימוש של עירמת פיבונacci עם מספר תוספות. עמודים 1-3 במסמך מתארים חלק זה.
2. חלק ניסויי-תאורטי: בהתבסס על השימוש מהחלק המעשי, נבצע מספר ניסויים עם ניתוח תאורטי נלווה. עמודים 4-7 מתאר חלק זה.

שימו לב: בסוף המסמך (עמוד 7) ישן הוראות הגשה – הקפידו לפעול לפיהן. תאריך הגשה: 25/01/2026

חלק מעשי

דרישות

בתרגיל זה יש למשוך עירמת פיבונacci בימוש כפי שלמדו בכיתה אבל עם מספר תוספות ושינויים. לכל איבר במבנה יש מפתח (key) שהוא מספר טبוי ומידע (info) שהוא מחרוזת. **המפתחות לא בהכרח ייחודיים, וכל הערמה מתיחס כרגע אך ורק למפתחות** השימוש יהיה בשפת **Java 21** וצריך להיות מבוסס על קובץ השלד המופיע באתר הקורס.

הבנייה של המחלקה מקבל שני משתנים בוליאניים: lazyMelds ו lazyDecreaseKeys. שני המשתנים הללו נשארים זהים לכל אורך חי ממבנה הנתונים ולא משתנים את המערכת. שני המשתנים הבוליאניים האלו קובעים את ההבדלים בין ההתנהגות של המבנה לבין עירמת פיבונacci, כפי שלמדו בכיתה (כלומר למחלקה שתתמנשו יתכונו ארבע התנהגויות שונות בהתאם לערכי האפשרים של שני המשתנים הנ"ל). ניתן להניח שאם מתבצעת פעולה meld בין שתי עירמות אז לשתין יש אותו ערך של lazyMelds ואוטו ערך של lazyDecreaseKeys.

כעת נגדיר את ההשפעה של lazyMelds ו lazyDecreaseKeys על ההתנהגות של מבנה הנתונים.

בפעולה meld מושרים את רשימות העצים של שתי העירמות לרשותה אחת (כפי שלמדו בכיתה בערימות פיבונacci). לאחר מכן, אם lazyMelds=false אז (בשונה מערימת פיבונacci) יש לבצע successive linking על רשימת העצים. אחרת (lazyMelds=true) לא מבצעים successive linking, כמו בערימת פיבונacci.

בלי שום קשר כאמור לעיל, זכרו ש successive linking זו פעולה שאנו מבצעים בכל מקרה בוחן deleteMin, deleteMax, deleteMin, deleteMax, כמו בערימת פיבונacci.

כאשר מבצעים פעולה decreaseKey: אם המפתח החדש של הצומת (לאחר הקטנה) גדול או שווה למפתח של אביו אז אין צורך לבצע פעולות נוספות כי כל הערימה לא מופר. אחרת יש לפעול כדלקמן:

- אם lazyDecreaseKeys=true אז פועלם כמו בערימת פיבונacci: מבצעים cascading cut (כולל התחזוק והטיפול בסימונים של הצמתים, כפי שלמדו בכיתה) ומוסיפים את הצומת שנוטק לרשימת השורשים. את ההוספה הzo של תת העץ שנוטק לרשימת פיבונacci: מבצעים update כל צומת meld בין תת העץ שנוטק לשאר הערימה. שימו לב שעת meld אנו מבצעים update כל צומת שנוטק במסגרת cut, cascading cut, כלומר אם יש מסלול ארוך של צמתים מסוימים שאותו ניתן אז יבוצעו הרבה פעולות meld במסגרת cut. כאמור, אם lazyMelds=false אז
- בנויגוד לעירמת פיבונacci, כל אחד מהmeld-ים האלו יגרום לביצוע successive linking.
- אחרת (lazyDecreaseKeys=false) נבצע heapifyUp כמו בdecreaseKey heapifyUp של עירימות בינוימות רגילות וערימות בינוימות עצלות.

שימוש לב שכאשר `lazyDecreaseKeys=false` ו- `lazyMelds=false` אז המבנה מתנהג בצורה דומה מאוד לערימה ביןומית רגילה, אם `lazyDecreaseKeys=false` ו- `lazyMelds=true` אז הערימה מתנהגת כמו ערימה ביןומית עצלה ואם `lazyDecreaseKeys=true` ו- `lazyMelds=true` אז המבנה מתנהג כמו ערימת פיבונacci. את השימוש כאשר `lazyDecreaseKeys=true` ו- `lazyMelds=false` לא למדנו בכתיבה.

שימוש לב שבכל מקרה, כפי שלמדנו בכתיבה, ללא תלות בערך של `lazyMelds` ו- `lazyDecreaseKeys`: יש למש את `delete` באמצעות `insert`, `deleteMin` ו- `decreaseKey`. כמו כן `insert`, `deleteMin`, `decreaseKey` ימומשו באמצעות `meld`.

הפעולות שיש למש הן:

פעולה	תיאור
<code>Heap(lazyMelds, lazyDecreaseKeys)</code>	הבנייה של המחלקה יקבל כפרמטר את הפרמטרים הבוליאניים <code>lazyMelds</code> ו- <code>lazyDecreaseKeys</code> שתוארו לעיל.
<code>insert(k, info)</code>	הכנסת איבר בעל מפתח <code>k</code> לערימה עם מידע <code>info</code> . הפונקציה מחזירה מצביע לצומת בערימה שנוצר עבורה.
<code>findMin()</code>	הפונקציה מחזירה את איבר הערימה בעל המפתח המינימלי.
<code>deleteMin()</code>	מחיקת האיבר המינימלי מהערימה (אנו צריך להחזיר).
<code>decreaseKey(x, d)</code>	הפונקציה מקבלת מצביע לאיבר הערימה <code>x</code> וטבעי <code>d</code> . היא מפחיתה את המפתח של <code>x</code> ב- <code>d</code> ומתקנת את הערימה בהתאם לערך של <code>lazyDecreaseKeys</code> , כפי שהסבירנו לעיל.
<code>delete(x)</code>	הפונקציה מקבלת מצביע לאיבר הערימה <code>x</code> ומוחקת אותו מהמבנה.
<code>meld(heap2)</code>	הפונקציה מזוגת את הערימה עם ערימה נוספת <code>heap2</code> . ניתן להניח ש <code>heap2</code> יש ערכי <code>lazyMelds</code> ו- <code>lazyDecreaseKeys</code> שעוזים לערכים של המשתנים הללו באובייקט הנוכחי (<code>this</code>). ש למש את הפעולה בהתאם לערך של <code>lazyMelds</code> בפונקציה <code>meld</code> לאחר הקוריאה לפונקציה <code>heap2</code> אינה שימושה.
<code>size()</code>	הפונקציה מחזירה את מספר האיברים בערימה.
<code>numTrees()</code>	הפונקציה מחזירה את מספר הצמתים המסומנים בערימה.
<code>numMarkedNodes()</code>	הפונקציה מחזירה את מספר הצמתים המסומנים בערימה. שימוש ב- <code>lazyDecreaseKeys=false</code> או לא יתכן שבערימה יש צמתים מסומנים.
<code>totalLinks()</code>	הפונקציה מחזירה את סך החיבורים (<code>links</code>) המצביעו שכזו (חיבור שני עצים מאותו הדרגה) מרגע יצירת הערימה.
<code>totalCuts()</code>	הפונקציה מחזירה את סך החיתוכים (ניתוק קשת) המצביעו מרגע יצירת הערימה, כולל את מספר הפעמים שבהם צומת התנתק מבאיו (הוצאת מרשימה הבנים שלו והוסף לרשימה השורשים) שבוצעו (במסגרת <code>cascading cuts</code>), כולל החיתוך הראשון של הצומת א, אם קרה. שימוש ב- <code>lazyDecreaseKeys=true</code> או לא יתכן שצומת מבאיו יכול לקורות כאשר קוראים ל- <code>decreaseKey</code> . כאמור גם כאשר קוראים ל- <code>delete</code> . כמו כן, ניתן יכול להבצע ורק אם <code>lazyDecreaseKeys=true</code> . כאשר אנו מבצעים ניתוק קשת הזרוקים את הלידים של המינימום ומכניםים אותם לרשימה השורשים: במקרה זה לא נוסיף את החיתוכים הללו למספר החיתוכים הכללי של הערימה.
<code>totalHeapifyCosts()</code>	הפונקציה מחזירה את סך העליות של ביצוע פעולות <code>heapifyUp</code> שבוצעו מרגע יצירת הערימה. שימוש לב שאנו מבצעים את <code>heapifyUp</code> זה קורה כאשר קוראים ל- <code>decreaseKey</code> או ל- <code>delete</code> . אנו מגדירים את העליות של פעולות <code>heapifyUp</code> (בז'ד) על צומת <code>v</code> להיות מספר הפעמים שהחלה פנו בין <code>v</code> לבין אביו. במקרים אחרים מזובר בהפרש בין הרמה של <code>v</code> לבין ביצוע הפעולה לרמה של אביו. במקרה אחר ביצוע הפעלה.

הערה: כאשר `meld` או נדרש להוסיף את היסטוריית ה- `links`, `cuts` וה- `links` של `heap2` לערימה שאליה מבצעים את `meld` כי המשמעות של הפעולות `insert`, `deleteMin`, `decreaseKey` ו- `totalLinks`, `totalCuts` היא שמירה של תוכנות של ההיסטוריה של הערימה. ב- `meld` אנו מחדדים שתי ערימות אחת (ולכן רצים לשמור את התוכנות של ההיסטוריה "מאוחדת" שלה).

לצורך מימוש פעולות אלו, ניעזר במחלקה **HeapNode** המופיעה בקובץ. המחלקה **HeapNode** המייצגת צומת בערימה מכילה את השדות הבאים:

- key – המפתח ששמור בצומת זה.
- info – המידע ששמור בצומת זה.
- child – בן כלשהו של צומת זה.
- next – האח הבא של צומת זה.
- prev – האח הקודם של צומת זה.
- parent – ההורה של צומת זה בערימה.
- rank – דרגת הצומת (מספר הבנים).

הערות חשובות:

1. המימוש יבוצע על ידי מיילוי קובץ השלד. מותר להחליף את תוכן הפונקציות המקוריות ולהוסיף פונקציות חדשות חדשות. אסור לשנות את חתימות הפונקציות המקוריות ואת שמות השדות המקוריים כדי לא לפגוע בפשטם. על כל הפונקציות/מחלקות להופיע בקובץ יחד.
2. אין להשתמש באך מימוש ספרייה של מבנה נתוניים.
3. עלייכם למשם את כל הפעולות בסיבוכיות המיטבית.

סיכום

יש לציין בקוד ולהסביר בקצרה במסמך התיעוד את סיבוכיות זמן הריצה במקרה הגורע (האיסימפטוטית, במונחי O הדוקים) של כל פונקציה שמכליה לולות/רקעישה, כתלות במספר האיברים בערימה ח.

פלט

אין צורך בפלט למשתמש.

תיעוד

בנוסף לבדיקות אוטומטיות של הקוד שיוגש, קובץ המקור יבדק גם באופן ידני. חשוב להקפיד על תיעוד לכל פונקציה וכמות סבירה של העורות. הקוד צריך להיות קרייא, בפרט הקפידו על בחירת שמות משתנים ועל אורך השורות.

יש להגיש בנוסף לקוד גם מסמך תיעוד חיצוני. המסמר יכול את תיאור תמציתי של המחלקה שמוסמה, ואת תפקידו של כל חבר במחלקה. עבור כל פונקציה במחלקה יש להסביר מה היא עשוה, כיצד היא פועלת ומהי סיבוכיות זמן הריצה שלה. בפרט, אם פונקציה קוראת לפונקציית עוז, יש להתייחס גם לפונקציית העזר בניתו. עבור פונקציות שעולות זמן קבוע יספק תיאור קצר ולא לפרט את ניתוח הסיבוכיות.

בדיקות

התרגילים ייבדקו באמצעות תוכנת טسطור שקוראת לפונקציות המפורטוות מעלה בתרחישים שונים, ומוגדרת את נכונות התוצאות. מומלץ מאוד למשם אוסף בדיקות עבור המימוש, לא בשביל ההגשה, אלא כדי לבדוק שהקוד לא רק רץ, אלא גם נכון!

בקובץ שתגישיו לא תהיה פונקציית **main** ולא יהיה הרצות קוד/הdfsות, דבר זה יפגע בטسطור שיבדק לכם את התרגילים. אין צורך להציג את הקוד הנוסף שכתבתם לחلك הנסיוני.

חלק ניסויי/תאורטי

את הירימה שמתקבלת כאשר `lazyDecreaseKeys=true` ו- `lazyMelds=false` (שאותה כאמור לא מדובר ברכבת) נenna "ירימה ביןומית עם ניתוקים". כאמור, הירימה הזו מבצעים successive linking בכל meld (לכן בפרט גם בכל `insert`) אבל גם לאחר כל ניתוק צומת במסגרת תחילת cascading cut. פועלה של `decreaseKey` successive linking זו פועלה **תמיד** רצה בזמן $(a \log(a))$ כי בכל מקרה בסוף צריכה לעבור על מערך ה"סלים" שאורכו לוגריתמי, לקחת מכל תא לא ריק את העץ ששומר בו ולצף את העצים לרשימת שורשים אחת.

1. בצעו ניתוח לשיעוריןLazyDecreaseKeys=false lazyMelds=false היריצה של הפעולות insert, findMin, deleteMin, decreaseKey בערימה ביןומית עם ניתוקים. הניחו שלכל אורך סדרת הפעולות מובטח שמספר האיברים בערימה בכל רגע הוא לכל היותר a .

נחבון בימוש כאשר lazyDecreaseKeys=false lazyMelds=false. השימוש הזה מתנהג כמו ירימה ביןומית אבל הביצועים שלו לא זהים לירימה ביןומית: בערימה ביןומית נהוג לשמר את העצים ברשימה של השורשים מסודרים בסדר הדרגות ואת meld מבצעים באופן דומה לחיבור ביןאי של שני מספרים. لكن ייתכן שזמן הריצה של meld פעולה insert בודדת (או של פעולה `insert` בודדת) יהיה נמוך בהרבה מה `log(a)`. בפרט, בערימה ביןומית פעולה insert ממבצעת באופן דומה לפעולה `increment` במוניה ביןאי וסדרה של a הכנסות לעירמה ביןומית ריקה תרעוץ בזמן (a^2) , בגלל אותן השיקולים של ניתוח אמורטייז שdone בהם בתרגול בניתוח של המונה הבינאי. לעומת זאת בימוש במלטה כאשר `useLazyMelds=true` ו lazyDecreaseKeys=true מושתמשים בsuccessive linking (זו פעולה שהציגנו בקורס רק בשלב שבו דנו בערימות ביןומיות עצמאיות) וכאמור, הפעולה הזו רצה **תמיד** בזמן $(a \log(a))$. על מנת לוודא שהביצועים של השימוש כאשר `useLazyMelds=true` ו lazyDecreaseKeys=true יייזו אסימפטוטית לעירמה ביןומית, ניתן לבצע עבודה אלטרנטטיבית (במקום היראה לsuccessive linking). מכיוון שרצינו לחסוך מכם את הביצוע של העבודה הזו, הנחנו אתכם להשתמש בוגם של meld כאשר רוצים למשערמה ביןומית בשאלות 2 ו- 3 - **עליכם למלא את התשובות על ירימה ביןומית כפי שלמדו בכיתה**, אילו הינו ממשים אותה במלטה הזו כפי שצריך, ללא שימוש בsuccessive linking.

2. מלאו את הטבלה הבאה שבה זמני הריצה הם אמורטייז (כפונקציה של a).

ירימה ביןומית עם ניתוקים	ירימת פיבונacci	ירימה ביןומית עצלה	ירימת פיבונacci	ירימה ביןומית	פעולה
					insert
					findMin
					deleteMin
					decreaseKey
					delete

בחלק זה נערך שלושה ניסויים המשתמשים בארכבעת סוג הירימות המתקדמות: ירימה ביןומית, ירימה ביןומית עצלה, ירימת פיבונacci וירימה ביןומית עם ניתוקים. נזכיר שוב שבשימוש שעשיותם אפשר "לקבל" כל אחת מהירימות הללו באמצעות מתן ערכים מתאימים ל- `lazyDecreaseKeys` ו- `lazyMelds`. עבור כל ניסוי נמדד את זמן הריצה, מספר החיבורים, מספר החיתוכים, סה"כ העלות של Up,heapify, מספר העצים בסיום והעלות המksamליות של פעולה בודדת. בכל הניסויים נגידר $n = 464,646$. לצורך הניסויים, בנו מערך מצביים המחזק באינדקס i מצביים לאיבר בעל מפתח i .

- ניסוי ראשון:
 1. נכנס לערמה ריקה את האיברים $a, \dots, 1$ בסדר אקראי.
 2. נמחק את המינימום.
- ניסוי שני:
 1. נכנס לערמה ריקה את האיברים $a, \dots, 1$ בסדר אקראי.
 2. נמחק את המינימום.
 3. נמחק את המקסימום (בעזרת מצביע) עד שנישאר עם 46 איברים.
- ניסוי שלישי:
 1. נכנס לערמה ריקה את האיברים $a, \dots, 1$ בסדר אקראי.
 2. נמחק את המינימום.
 3. נבצע [a_1] פעמים הקטנת מפתח של המקסימום (בעזרת מצביע) ל-0 (כלומר עד שנישאר עם 90% איברים שהמפתח שלהם חיובי וכל שאר האיברים הם עם מפתח 0).
 4. נבצע מחיקת מינימום שוב.

3. מלאו את הטבלה הבאה. בכל תא יש לרשום את זמן הריצה האסימפטוטי של הניסוי המתאים על סוג הערימה המתאים כפונקציה של a . כמו בשאלה 2, עליכם למלא את התשובות על ערימה ביןומית במימוש כדי שלמדו בכיתה, אילו היו שימושים אותה במללה הזו כפי שציריך, ללא שימוש בlinking.

ערימה ביןומית עם ניטוקים	ערימה פיבונאצ'י	ערימה ביןומית עצלה	ערימה ביןומית	
				ניסוי 1
				ניסוי 2
				ניסוי 3

- 4.** עבור כל ניסוי מבין שלושת הניסויים שהוגדרו:
- הריצו אותו על כל אחת מרבעת הערימות המתקדמות.
 - יש למלא בטבלה את הממוצע על פני 20 ניסויים זהים. לשם כך עבור כל ניסוי יש להגריל 20 פרמטריזציות אקראיות של האיברים $a, \dots, 1$ ועבור כל פרמטריזיה שהוגלה יש לבצע את הניסוי על כל אחד מרבעת סוגי הערימות עם הפרמטריזיה שהוגלה.
 - בשורה "עלות מקסימלית לפועלה" יש לרשום את העלות המקסימלית של פועלה בחודת שבוצעה בסדרה (הכנסה / מחיקת מינימום / מחיקה / הקטנת מפתח). אנו מגדירים את העלות של פועלה בודדת להיות הסכום של מספר החיבורים, מספר החיתוכים ועלויות `heapify` שבוצעו בפועלה.

ניסוי 1:

ערימה ביןומית עם ניתוקים	ערימת פיבונacci'	ערימה ביןומית עצלה	ערימה ביןומית	
				זמן ריצה (מילישניות)
				גודל הערימה בסיום
				מספר העצים בסיום
				מספר חיבורים
				מספר חיתוכים
				סה"כ עליות של up heapify
				עלות מקסימלית לפעולה

ניסוי 2:

ערימה ביןומית עם ניתוקים	ערימת פיבונacci'	ערימה ביןומית עצלה	ערימה ביןומית	
				זמן ריצה (מילישניות)
				גודל הערימה בסיום
				מספר העצים בסיום
				מספר חיבורים
				מספר חיתוכים
				סה"כ עליות של up heapify
				עלות מקסימלית לפעולה

ערימה ביןומית עם ניתוקים	ערימת פיבונacci'	ערימה ביןומית עצלה	ערימה ביןומית	
				זמן ריצה (מילישניות)
				גודל הערימה בסיום
				מספר העצים בסיום
				מספר חיבורים
				מספר חיתוכים
				סה"כ עליות של up heapify
				עלות מקסימלית לפעולה

.5

- א. האם יש הבדלים בביטויים של הניסויים בין הערים השונות?
 ב. האם התוצאות שקיבלתם בסעיף 4 מאשיות או מפריכות את מה שעניתם בסעיף 3?
 ג. האם בניסויים היו הבדלים בין העלות המקסימלית של פעולה לבין העלות הממוצעת של פעולה?

הוראות הגשה

הגשת התרגיל תבוצע באופן אלקטרוני באתר הקורס במודול.

הגשת התרגיל היא בזוגות בלבד!

כל זוג יבחר נציג/ה ויעלה בדף תחת שם המשתמש של הנציג/ה את קבצי התרגיל (תחת קובץ **zip**) למודול.

על ההגשה לכלול שלושה קבצים:

1. קובץ המקור (הרחה של קובץ השלד שניית) תחת השם **Heap.java**.
2. קובץ טקסט **info.txt** המכיל את פרטי הזוג: שמות מלאים בעברית, שמות מלאים באנגלית ומספרי ת"ז. יש לרשום גם שמות משתמש (או ליחלופין אימיילים אוניברסיטאיים) של המתושים.
3. מסמך בפורמט **pdf** שמכיל קודם את התיעוד החיצוני ולאחר מכן מכון את המענה לחלק הניסויי/תיאורטי.

שמות קובץ התיעוד וקובץ הקזז צריכים לכלול את שמות המשתמש האוניברסיטאיים של הזוג המתיש לפי הפורמט **zip/zip**, בתוכן הקבצים יש לציין את שמות המשתמש, תעודות זהירות ושמות המתושים (בכותרת המסמך ובשורת הערא בקובץ המקור).

הגשת שיעורי הבית באיחור - באישור מראש בלבד. הגשה באיחור ללא אישור תגרור הורדת נקודות מהציון.
 הגשת התרגיל היא חובה לשם קבלת ציון בקורס.

בצלחה!