

מבני נתונים – פרויקט מספר 2 – ערמות מתקדמות

הקדמה

בתרגיל זה שני חלקים:

1. חלק מעשי: מימוש של ערימת פיבונאצ' עם מספר תוספות. עמודים 1-3 במסמך מתארים חלק זה.
2. חלק ניסוי-תאורטי: בהתבסס על המימוש מהחלק המעשי, נבצע מספר ניסויים עם ניתוח תאורטי נלווה. עמודים 4-7 מתאר חלק זה.

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

חלק מעשי

דרישות

בתרגיל זה יש לממש ערימת פיבונאצ' במימוש כפי שלמדנו בכיתה אבל עם מספר תוספות ושינויים. לכל איבר במבנה יש מפתח (key) שהוא מספר טבעי ומידע (info) שהוא מחרוזת. **המפתחות לא בהכרח ייחודיים, וכלל הערמה מתייחס כרגיל אך ורק למפתחות.** המימוש יהיה בשפת **Java 21** וצריך להיות מבוסס על קובץ השלד המופיע באתר הקורס.

הבנאי של המחלקה מקבל שני משתנים בוליאנים: `lazyMelds` ו-`lazyDecreaseKeys`. שני המשתנים האלו נשארים זהים לכל אורך חיי מבנה הנתונים ולא משנים את ערכם. שני המשתנים הבוליאנים האלו קובעים את ההבדלים בין ההתנהגות של המבנה לבין ערימת פיבונאצ', כפי שלמדנו בכיתה (כלומר למחלקה שתממשו ייתכנו ארבע התנהגויות שונות בהתאם לערכים האפשריים של שני המשתנים הנ"ל). ניתן להניח שאם מתבצעת פעולת `meld` בין שתי ערימות אז לשתייהן יש אותו ערך של `lazyMelds` ואותו ערך של `lazyDecreaseKeys`.

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

בפעולת `meld` משרשרים את רשימות העצים של שתי הערימות לרשימה אחת (כפי שלמדנו בכיתה בערימות פיבונאצ'). לאחר מכן, אם `lazyMelds=false` אז (בשונה מערימת פיבונאצ') יש לבצע successive linking על רשימת העצים. אחרת (`lazyMelds=true`) לא מבצעים successive linking, כמו בערימת פיבונאצ'.

בלי שום קשר לאמור לעיל, זכרו ש successive linking זו פעולה שאנו מבצעים בכל מקרה `deleteMin`, כמו בערימת פיבונאצ'.

- כאשר מבצעים פעולת `decreaseKey`: אם המפתח החדש של הצומת (לאחר ההקטנה) גדול או שווה למפתח של אביו אז אין צורך לבצע פעולות נוספות כי כלל הערימה לא מופר. אחרת יש לפעול כדלקמן:
- אם `lazyDecreaseKeys=true` אז פועלים כמו בערימת פיבונאצ': מבצעים cascading cut (כולל התחזוק והטיפול בסימונים של הצמתים, כפי שלמדנו בכיתה) ומוסיפים את הצומת שנותק לרשימת השורשים. את ההוספה הזו של תת העץ שנותק לרשימת השורשים יש לבצע באמצעות `meld` בין תת העץ שנותק לשאר הערימה. שימו לב שאת `meld` אנו מבצעים עבור כל צומת שנותק במסגרת `cascading cut`, כלומר אם יש מסלול ארוך של צמתים מסומנים שאותו ניתקנו אז יבוצעו הרבה פעולות `meld` במסגרת `cascading cut`. כאמור, אם `lazyMelds=false` אז בניגוד לערימת פיבונאצ', כל אחד מה `meld`-ים האלו יגרום לביצוע successive linking.
 - אחרת (`lazyDecreaseKeys=false`) נבצע `heapifyUp` כמו ב `decreaseKey` של ערימות בינומיות רגילות וערימות בינומיות עצלות.

שימו לב שכאשר lazyDecreaseKeys=false ו lazyMelds=false אז המבנה מתנהג בצורה דומה מאוד לערימה בינומית רגילה, אם lazyDecreaseKeys=false ו lazyMelds=true אז הערימה מתנהגת כמו ערימה בינומית עצלה ואם lazyDecreaseKeys=true ו lazyMelds=true אז המבנה מתנהג כמו ערימת פיבונאצ'י. את המימוש כאשר lazyDecreaseKeys=true ו lazyMelds=false לא למדנו בכיתה.

שימו לב שבכל מקרה, כפי שלמדנו בכיתה, ללא תלות בערכם של lazyMelds ו lazyDecreaseKeys: יש לממש את delete באמצעות decreaseKey ו deleteMin. כמו כן insert ימומשו באמצעות meld.

הפעולות שיש לממש הן:

פְּעוּלָה	תִּיאוּר
Heap(lazyMelds, lazyDecreaseKeys)	הבנאי של המחלקה יקבל כפרמטר את הפרמטרים הבוליאניים lazyMelds ו lazyDecreaseKeys שתוארו לעיל.
insert(k, info)	הכנסת איבר בעל מפתח k לערמה עם מידע info. הפונקציה מחזירה מצביע לצומת בערימה שנוצר עבורו.
findMin()	הפונקציה מחזירה את איבר הערמה בעל המפתח המינימלי.
deleteMin()	מחיקת האיבר המינימלי מהערמה (אין צורך להחזיר).
decreaseKey(x, d)	הפונקציה מקבלת מצביע לאיבר הערמה x וטבעי d. היא מפחיתה את המפתח של x ב-d ומתקנת את הערמה בהתאם לערך של lazyDecreaseKeys, כפי שהוסבר לעיל.
delete(x)	הפונקציה מקבלת מצביע לאיבר הערמה x ומוחקת אותו מהמבנה.
meld(heap2)	הפונקציה ממזגת את הערמה עם ערמה נוספת heap2. ניתן להניח של heap2 יש ערכי lazyDecreaseKeys ו lazyMelds שזהים לערכים של המשתנים האלו באובייקט הנוכחי (this). יש לממש את הפעולה בהתאם לערך של lazyMelds כפי שתואר לעיל. לאחר הקריאה לפעולה הערמה heap2 אינה שמישה.
size()	הפונקציה מחזירה את מספר האיברים בערמה.
numTrees()	הפונקציה מחזירה את מספר העצים בערמה.
numMarkedNodes()	הפונקציה מחזירה את מספר הצמתים המסומנים בערימה. שימו לב שאם lazyDecreaseKeys=false אז לא ייתכן שבערימה יש צמתים מסומנים.
totalLinks()	הפונקציה מחזירה את סך החיבורים (links) המצטבר שבוצעו (חיבור שני עצים מאותה הדרגה) מרגע יצירת הערימה.
totalCuts()	הפונקציה מחזירה את סך החיתוכים (ניתוק קשת) המצטבר שבוצעו מרגע יצירת הערימה, כלומר את מספר הפעמים שבהם צומת התנתק מאביו (הוצא מרשימת הבנים שלו והוסף לרשימת השורשים) שבוצעו (במסגרת cuts cascading, כולל החיתוך הראשוני של הצומת x, אם קרה). שימו לב שניתוק של צומת מאביו יכול לקרות כאשר קוראים ל decreaseKey אבל גם כאשר קוראים ל delete. כמו כן, ניתוק יכול להתבצע רק אם lazyDecreaseKeys=true. כאשר אנו מבצעים deleteMin אנחנו מנתקים את הילדים של המינימום ומכניסים אותם לרשימת השורשים: במקרה הזה לא נוסיף את החיתוכים האלו למספר החיתוכים הכללי של הערימה.
totalHeapifyCosts()	הפונקציה מחזירה את סך העלויות של ביצוע פעולות heapifyUp שבוצעו מרגע יצירת הערימה. שימו לב שאנו מבצעים heapifyUp רק במקרה שבו lazyDecreaseKeys=false זה קורה כאשר קוראים ל decreaseKey או delete. אנו מגדירים את העלות של פעולת heapifyUp (בודדת) על צומת v להיות מספר הפעמים שהחלפנו בין v לבין אביו. במילים אחרות מדובר בהפרש בין הרמה של v בעץ לפני ביצוע הפעולה לרמה של v בעץ אחרי ביצוע הפעולה.

הערה: כאשר מבצעים meld אז נדרש להוסיף את היסטוריית ה links , cuts ו heapifyCosts של heap2 לערימה שאליה מבצעים את meldn כי המשמעות של הפעולות totalLinks, totalCuts ו totalHeapifyCosts היא שמירה של תכונות של ההיסטוריה של הערימה. במeld אנו מאחדים שתי ערימות לערימה אחת (ולכן רוצים לשמור את התכונות של ההיסטוריה ה"מאוחדת" שלה).

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

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

הערות חשובות:

1. המימוש יבוצע על ידי מילוי קובץ השלד. מותר להחליף את תוכן הפונקציות הקיימות ולהוסיף פונקציות ושדות חדשים. אסור לשנות את חתימות הפונקציות הקיימות ואת שמות השדות הקיימים כדי לא לפגוע בטסטר. על כל הפונקציות/מחלקות להופיע בקובץ יחיד.
2. אין להשתמש באף מימוש ספרייה של מבנה נתונים.
3. עליכם לממש את כל הפעולות בסיבוכיות המיטבית.

סיבוכיות

יש לציין בקוד ולהסביר בקצרה במסמך התיעוד את סיבוכיות זמן הריצה במקרה הגרוע (האסימפטוטית, במונחי O הדוקים) של כל פונקציה שמכילה לולאות/רקורסיה, כתלות במספר האיברים בערימה n.

פלט

אין צורך בפלט למשתמש.

תיעוד

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

יש להגיש בנוסף לקוד גם מסמך תיעוד חיצוני. המסמך יכלול את תיאור תמציתי של המחלקה שמומשה, ואת תפקידו של כל חבר במחלקה. עבור כל פונקציה במחלקה יש להסביר מה היא עושה, כיצד היא פועלת ומהי סיבוכיות זמן הריצה שלה. בפרט, אם פונקציה קוראת לפונקציית עזר, יש להתייחס גם לפונקציית העזר בניתוח. עבור פונקציות שעולות זמן קבוע יספיק תיאור קצר ולא לפרט את ניתוח הסיבוכיות.

בדיקות

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

בקובץ שתגישו לא תהיה פונקציית **main** ולא יהיו הרצות קוד/הדפסות, דבר זה יפגע בטסטר שיבדוק לכם את התרגילים. אין צורך להגיש את הקוד הנוסף שכתבתם לחלק הניסויי.

חלק ניסויי/תאורטי

את הערימה שמתקבלת כאשר $\text{lazyDecreaseKeys}=\text{true}$ ו $\text{lazyMelds}=\text{false}$ (שאותה כאמור לא למדנו בכיתה) נכנה "ערימה בינומית עם ניתוקים". כאמור, בערימה הזו מבצעים successive linking בכל meld (לכן בפרט גם בכל insert) אבל גם לאחר כל ניתוק צומת במסגרת תהליך cascading cutn בפעולה decreaseKey. שימו לב ש successive linking זו פעולה שתמיד רצה בזמן $\Omega(\log n)$ כי בכל מקרה בסוף צריך לעבור על מערך ה"סלים" שאורכו לוגריתמי, לקחת מכל תא לא ריק את העץ ששמור בו ולצרף את העצים לרשימת שורשים אחת.

1. בצעו ניתוח לשיעורין לזמני הריצה של הפעולות insert, findMin, deleteMin, decreaseKey ו delete בערימה בינומית עם ניתוקים. הניחו שלכל אורך סדרת הפעולות מובטח שמספר האיברים בערימה בכל רגע הוא לכל היותר n .

נתבונן במימוש כאשר $\text{lazyDecreaseKeys}=\text{false}$ ו $\text{lazyMelds}=\text{false}$. המימוש הזה מתנהג כמו ערימה בינומית אבל הביצועים שלו לא זהים לערימה בינומית: בערימה בינומית נהוג לשמור את העצים ברשימת השורשים מסודרים בסדר הדרגות ואת meld מבצעים באופן דומה לחיבור בינארי של שני מספרים. לכן ייתכן שזמן הריצה של פעולת meld בודדת (או של פעולת insert בודדת) יהיה נמוך בהרבה מ \log . בפרט, בערימה בינומית פעולת insert מתבצעת באופן דומה לפעולת increment במונה בינארי וסדרה של n הכנסות לערימה בינומית ריקה תרוץ בזמן $O(n)$, בגלל אותם השיקולים של ניתוח אמורטייזד שדנו בהם בתרגול בנייתוח של המונה הבינארי. לעומת זאת במימוש במטלה כאשר $\text{lazyMelds}=\text{true}$ ו $\text{lazyDecreaseKeys}=\text{true}$ משתמשים ב successive linking (זו פעולה שהצגנו בקורס רק בשלב שבו דנו בערימות בינומיות עצלות) וכאמור, הפעולה הזו רצה תמיד בזמן $\Omega(\log n)$. על מנת לוודא שהביצועים של המימוש כאשר $\text{lazyDecreaseKeys}=\text{true}$ ו $\text{lazyMelds}=\text{true}$ יהיו זהים אסימפטוטית לערימה בינומית, ניתן לבצע עבודה אלטרנטיבית (במקום הקריאה ל successive linking). מכיוון שרצינו לחסוך מכם את הביצוע של העבודה הזו, הנחנו אתכם להשתמש ב successive linking במימוש של meld כאשר רוצים לממש ערימה בינארית. עם זאת, בנייתוח התיאורטי שתמלאו על ערימה בינומית בשאלות 2 ו 3 - **עליכם למלא את התשובות על ערימה בינומית כפי שלמדנו בכיתה**, אילו היינו ממשים אותה במטלה הזו כפי שצריך, ללא שימוש ב successive linking.

2. מלאו את הטבלה הבאה שבה זמני הריצה הם באמורטייזד (כפונקציה של n).

פעולה	ערימה בינומית	ערימה בינומית עצלה	ערימת פיבונאצ'י	ערימה בינומית עם ניתוקים
insert				
findMin				
deleteMin				
decreaseKey				
delete				

בחלק זה נערוך שלושה ניסויים המשתמשים בארבעת סוגי הערימות המתקדמות: ערימה בינומית, ערימה בינומית עצלה, ערימת פיבונאצ'י וערימה בינומית עם ניתוקים. נזכיר שוב שבמימוש שעשיתם אפשר "לקבל" כל אחת מהערימות האלו באמצעות מתן ערכים מתאימים ל lazyMelds ו lazyDecreaseKeys . עבור כל ניסוי נמדוד את זמן הריצה, מספר החיבורים, מספר החיתוכים, סה"כ העלויות של heapify Up, מספר העצים בסיום והעלות המקסימלית של פעולה בודדת. בכל הניסויים נגדיר $n = 464,646$. לצורך הניסויים, בנו מערך מצביעים המחזיק באינדקס i מצביע לאיבר בעל מפתח i .

- ניסוי ראשון:
 1. נכניס לערמה ריקה את האיברים $1, \dots, n$ בסדר אקראי.
 2. נמחק את המינימום.
- ניסוי שני:
 1. נכניס לערמה ריקה את האיברים $1, \dots, n$ בסדר אקראי.
 2. נמחק את המינימום.
 3. נמחק את המקסימום (בעזרת מצביע) עד שנישאר עם 46 איברים.
- ניסוי שלישי:
 1. נכניס לערמה ריקה את האיברים $1, \dots, n$ בסדר אקראי.
 2. נמחק את המינימום.
 3. נבצע $\lceil 0.1n \rceil$ פעמים הקטנת מפתח של המקסימום (בעזרת מצביע) ל0 (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 n . כמו בשאלה 2, עליכם למלא את התשובות על ערימה בינומית במימוש כפי שלמדנו בכיתה, אילו היינו מממשים אותה במטלה הזו כפי שצריך, ללא שימוש ב successive linking.

	ערימה בינומית	ערימה בינומית עצלה	ערימת פיבונאצ'י	ערימה בינומית עם ניתוקים
ניסוי 1				
ניסוי 2				
ניסוי 3				

- 4.** עבור כל ניסוי מבין שלושת הניסויים שהוגדרו:
- הריצו אותו על כל אחת מארבעת הערימות המתקדמות.
 - יש למלא בטבלה את הממוצע על פני 20 ניסויים זהים. לשם כך עבור כל ניסוי יש להגריל 20 פרמוטציות אקראיות של האיברים $1, \dots, n$ ועבור כל פרמוטציה שהוגרלה יש לבצע את הניסוי על כל אחד מארבעת סוגי הערימות עם הפרמוטציה שהוגרלה.
 - בשורה "עלות מקסימלית לפעולה" יש לרשום את העלות המקסימלית של פעולה בודדת שבוצעה בסדרה (הכנסה / מחיקת מינימום / מחיקה / הקטנת מפתח). אנו מגדירים את העלות של פעולה בודדת להיות הסכום של מספר החיבורים, מספר החיתוכים ועלויות heapify Up שבוצעו בפעולה.

ניסוי 1:

ערימה בינומית	ערימה בינומית עצלה	ערימת פיבונאצ'י	ערימה בינומית עם ניתוקים	
				זמן ריצה (מילישניות)
				גודל הערימה בסיום
				מספר העצים בסיום
				מספר חיבורים
				מספר חיתוכים
				סה"כ עלויות של heapify up
				עלות מקסימלית לפעולה

ניסוי 2:

ערימה בינומית	ערימה בינומית עצלה	ערימת פיבונאצ'י	ערימה בינומית עם ניתוקים	
				זמן ריצה (מילישניות)
				גודל הערימה בסיום
				מספר העצים בסיום
				מספר חיבורים
				מספר חיתוכים
				סה"כ עלויות של heapify up
				עלות מקסימלית לפעולה

ערימה בינומית עם ניתוקים	ערימת פיבונאצ'י	ערימה בינומית עצלה	ערימה בינומית	
				זמן ריצה (מילישניות)
				גודל הערימה בסיום
				מספר העצים בסיום
				מספר חיבורים
				מספר חיתוכים
				סה"כ עלויות של heapify up
				עלות מקסימלית לפעולה

5.

- א. האם יש הבדלים בביצועים של הניסויים בין הערימות השונות?
 ב. האם התוצאות שקיבלתם בסעיף 4 מאששות 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 תתבצע באופן אלקטרוני באתר הקורס במודל.
 הגשת התרגיל היא בזוגות בלבד!
 כל זוג יבחר נציג/ה ויעלה רק תחת שם המשתמש של הנציג/ה את קבצי התרגיל (תחת קובץ **zip**) למודל.

על ההגשה לכלול שלושה קבצים:

1. קובץ המקור (הרחבה של קובץ השלד שניתן) 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 לחילופין אימיילים אוניברסיטאיים) של המגישים.
3. מסמך בפורמט **pdf** שמכיל קודם את התיעוד החיצוני ולאחר מכן את המענה לחלק הניסויי/תיאורטי.

שמות קובץ התיעוד וקובץ הzip צריכים לכלול את שמות המשתמש האוניברסיטאיים של הזוג המגיש לפי הפורמט **Heap_username1_username2.pdf/zip**, בתוכן הקבצים יש לציין את שמות המשתמש, תעודות הזהות ושמות המגישים (בכותרת המסמך ובשורת הערה 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.

בהצלחה!