Data Structures – Project #2
Oded Neeman – odedneeman – 207110479
Roy Mayan – roymayan – 206483554

חלק מעשי

המחלקה HeapNode

תיאור: HeapNode מייצגת צומת בודד בערימת פיבונאצ'י של מספרים שלמים FibonacciHeap.

שדות:

שדה	תפקיד
key	מפתח הצומת
rank	דרגת הצומת – כמות הילדים הישירים שלו
mark	האם הצומת מסומן או לא
child	מצביע לילד
next	מצביע לצומת הבא
prev	מצביע לצומת הקודם
parent	מצביע להורה
original	מצביע ייעודי עבור פונקצית kMin (ראו בהמשך)

מתודות:

מתודה	תיאור	סיבוכיות
HeapNode(int key)	בנאי המחלקה. כל השדות מאותחלים לערכים דיפולטיים, פרק ל-next ו-prev שמצביעים ל-node החדש שכעת נוצר (זה המצב בברירת המחדל).	$O(1)$ מתבצעות השמות בזמן קבוע.
getX()	פונקציות אשר מחזירות את שדה X: כאשר X הוא .../key/rank/mark	$O(1)$
setX()	פונקציות אשר מבצעות השמה לשדה X: כאשר X הוא .../key/rank/mark הערה: setChild() מבצעת הגדלה של ה-rank ב-1.	$O(1)$
setNextPrev(HeapNode next)	מבצעת שיוך מצביעים דו-כיווני בין שני צמתים (עבור מצביעי next-prev).	$O(1)$ מתבצעות השמות בזמן קבוע.
setParentChild(HeapNode parent)	מבצעת שיוך מצביעים דו-כיווני בין שני צמתים (עבור מצביעי child-parent).	$O(1)$ מתבצעות השמות בזמן קבוע.
resetKeepChild()	מאפס (השמה ל-null) את כל שדות הצומת פרט למצביע הילד שלו.	$O(1)$ מתבצעות השמות בזמן קבוע.
isRoot()	מחזירה האם הצומת הנוכחי הוא שורש (בדיקת המצביע של ההורה – אם אין הורה מדובר בשורש).	$O(1)$
link(HeapNode other)	מחברת את הצומת הנוכחי לצומת other (מבצעת link). מתבצעת השוואת גדלים בין המפתחות של הצמתים, על מנת לבצע את החיבור תוך שמירת כלל הערימה. בחיבור אנו מתקנים מצביעים ונעזרים בפונקציות setNextPrev, setParentChild. כל link כזה גם מגדיל ב-1 את ה-counter הסטטי במחלקה FibonacciHeap של totalLinks.	$O(1)$
cut()	חותכת את הצומת הנוכחי מאביו. אם לצומת אין אחים (ילד יחידי של אביו) ננתק אותו מההורה שלו. אם יש לו אחים, נתקן את מצביע child של ההורה שלו במידת הצורך. כל cut מגדיל את ה-counter הסטטי המתאים במחלקה FibonacciHeap.	$O(1)$

המחלקה FibonacciHeap

תיאור: FibonacciHeap מייצגת ערימת פיבונאצ'י של מספרים שלמים.

שדות:

שדה	תפקיד
first	מצביע לאיבר הרשימה בערימה
last	מצביע לאיבר האחרון בערימה
nodesCount	כמות האיברים הכללית בערימה
treesCount	כמות העצים בערימה
markedCount	כמות הצמתים המסומנים בערימה
totalCuts	מספר החיתוכים שבוצעו עד כה (שדה סטטי)
totalLinks	מספר החיבורים שבוצעו עד כה (שדה סטטי)
GOLDEN	יחס הזהב (קבוע)

מתודות:

מתודה	תיאור	סיבוכיות
FibonacciHeap()	בנאי המחלקה. שדות המצביעים מאופסים ל-null ושדות הספירה ל-0.	$O(1)$ מתבצעות השמות בזמן קבוע.
getFirst()	מחזירה את הצומת (מסוג HeapNode) הראשון בערימה.	$O(1)$ יש מצביע על צומת זה.
getLast()	מחזירה את הצומת (מסוג HeapNode) האחרון בערימה.	$O(1)$ יש מצביע לראשון, והשדה prev שלו הוא האחרון.
isEmpty()	מחזירה TRUE אם ורק אם הרשימה ריקה.	$O(1)$ בדיקת האיבר הראשון בזמן קבוע.
insertNode(HeapNode node, boolean consolidate)	מתודה זו מבצעת לצומת הכנסה "עצלה": ממקמת אותה במקום הראשון בערימה, ובודקת אם יש לעדכן את המינימום של הערימה או לא. כמו כן שדות הספירה של מספר העצים ומספר הצמתים מועלים ב-1. עם קבלת ערך בוליאני true, לא נעלה את ספירת הצמתים, משום שנשתמש במתודה זו ב-consolidte, שבה אנו מכניסים צמתים אך לא באמת מעלים את ספירתם.	$O(1)$ הכנסה עצלה- הכנסת צומת למקום הראשון, על ידי שינוי מצביעים ועידכון שדות, כל אלה בכמות קבועה.
insert(int key)	יוצרת צומת מסוג HeapNode בעל מפתח key, מכניסה אותו לערימה, ומחזירה אותו. אנו יוצרים את הצומת בעזרת הבנאי של HeapNode ואז קוראים למתודה insertNode על הצומת ועל ערך false.	$O(1)$ יצירת הצומת בזמן קבוע ואז קריאה לפונקציה שרצה בזמן קבוע.
deleteMin()	מחיקת צומת שהמפתח שלו מינימלי מבין המפתחות בערימה. יש לנו מצביע על צומת זה ולכן הגישה אליו מיידית. אנו בודקים מקרי קצה כדי שלא נשנה מצביעים לדברים לא חוקיים, ומתקנים את המצביעים בהתאם, כך שהצומת המינימלי כבר לא יהיה בערימה. אנו מעדכנים את ספירת הצמתים (תמיד מורידים 1) והעצים (תלוי במספר הילדים של הצומת שמחקנו). בסוף הפונקציה אנו קוראים לconsolidate.	$O(n)$ מחיקת הצומת עצמו נעשית בזמן קבוע. על הסיבוכיות אחראית הקריאה ל-consolidate (ניתוח להלן).
consolidate()	גיבוש של העץ אחרי מחיקת מינימום. המתודה מבצעת פעולת to-buckets ואחריה from-buckets כפי שראינו בביתה: תחילה מאתחלים מערך (ה"דליים") של HeapNode, באורך לוג בבסיס יחס הזהב של מספר הצמתים, ועוד 1. זה מייצג את מספר הרמות השונות שיכולות להיות בערימה עם מספר האיברים שיש לנו. אנחנו מבצעים איטרציה על רמת השורשים בערימה, עוברים על העצים (כאשר השורש מתפקד כמצביע	$O(n)$ במקרה הגרוע יש לנו ערימה שהיא אוסף של n עצים שהם צומת בודד. המעבר על כולם לצורך ההכנסה לדליים ידרוש זמן $O(n)$ (לא יותר כי מספר החיבורים יהיה לוגריתמי). הפעולה מתבצעת באותו אופן שראינו בהרצאה ולכן הסיבוכיות Amortized היא אכן $\log(n)$.

	לעץ), ועבור כל עץ: מאפסים את המצביעים לעץ הבא והקודם כי הם לא רלוונטיים, וכן הופכים את הסימון ל-0 (false) במקרה הצורך (כי הוא שורש כעת). לאחר מכן, אם במערך הדליים יש במקום המתאים לדרגה שלו עץ, אנחנו "מוציאים" את העץ השני מהמערך, מבצעים ביניהם link, ואת העץ שנוצר מכניסים למקום המתאים לדרגה החדשה. אם המקום החדש גם תפוס, ממשיכים באותו אופן. לאחר סיום האיטרציה אנחנו מאפסים את שדות הfirst והמינימום, ואת ספירת העצים- סדרת ההכנסות שנעשה תדאג לתחזק אותם בעזרת פונקציית insertNode. כעת עוברים על מערך הדליים מהסוף להתחלה (סדר דרגות יורד), ובכל תא - אם הוא לא ריק - מכניסים את העץ שבו לערימה (בעזרת insertNode על שורש העץ, עם ערך true).	
findMin()	החזרת הצומת (מטיפוס HeapNode) שהמפתח שלו מינימלי מבין המפתחות בערימה, מערימה ריקה יוחזר null.	$O(1)$ יש לנו מצביע לאיבר המינימלי
meld(FibonacciHeap heap2)	מיזוג ערימה נוספת heap2 עם הערימה הנוכחית. אם חושבים על ערימות כרשימה מקושרת של השורשים- אנחנו משרשרים את heap2 לסוף רשימת השורשים של הערימה שלנו. לשם כך דרוש לשנות 4 מצביעים לכל היותר, ולבחור את המינימום הקטן יותר מבין 2 הערימות להיות המינימום של הערימה המאוחדת. כמו כן, שדות הספירה יהיו סכום שני השדות של שתי הערימות.	$O(1)$ עדכון מספר קבוע של שדות ומצביעים.
size()	הפונקציה מחזירה את מספר האיברים בערימה.	$O(1)$ שדה שאנו דואגים לתחזק.
countersRep()	מחזירה מערך מונים כך שבאינדקס i מופיע מספר העצים בערימה שהסדר שלהם הוא i. עבור ערימה ריקה מוחזר מערך ריק. אנו מבצעים שתי איטרציות על רשימת השורשים. בראשונה אנו מוצאים מה הדרגה המקסימלית של עץ שיש, ולפיה מאתחלים את המערך בגודל הדרוש (+1). באיטרציה השנייה, אנחנו מוסיפים 1 לאיבר המתאים במערך לדרגת השורש עליו אנחנו עוברים.	$O(n)$ הסיבוכיות לינארית במספר השורשים. במקרה הגרוע, כל האיברים הם שורשים.
delete(HeapNode x)	מחיקת הצומת x מהרשימה. אנחנו מבצעים לצומת x decreaseKey כך שיהיה בוודאות המינימום החדש בערימה, ואז מוחקים את המינימום (כלומר את x).	$O(n)$ יש קריאה אחת לפונקציה decreaseKey ואחת ל-deleteMin. לכן הן קובעות את הסיבוכיות.
cascadingCuts(HeapNode x)	פונקציה שנקרא לה מתוך decreaseKey כדי לבצע את השינויים הנדרשים לאחר פעולה זו. השינויים: אנו חותכים את הצומת x (ומבטלים את הסימון שלה אם צריך), ואז כל עוד ההורה מסומן, חותכים גם אותו ומטפסים, עד שמגיעים להורה לא מסומן ואז עוצרים.	$O(n)$ בכל צומת שאנו בו אנו מבצעים פעולות בזמן קבוע- גם פעולת חיתוך נעשית בזמן קבוע. המקרה הגרוע ביותר, הוא המקרה בו נבצע חיתוכים ונמשיך להתקדם, עד שנגיע לשורש העץ ש-x נמצא בו. כלומר סיבוכיות כעומק העץ. ניתן לאחר שילוב של הכנסות ומחיקות מינימום לקבל עץ שהוא "שרשרת" כלומר בעומק לינארי ב-n- ראינו זאת בתרגול 9, שאלה 3. בתרגול הצמתים היו לא מסומנים, כדי שהם יהיו מסומנים לכל האורך, נדרש שינוי קטן ואפשרי (בכל פעם הוספת עוד שלב ביניים בו מוסיפים צמתים כך שהצומת החדש שיישאר בשרשרת לא יהיה שורש כשנמחק את הילד שלו, ולכן יהיה מסומן), וכך יתקבל המצב הגרוע.

decreaseKey(HeapNode x, int delta)	ערכו של המפתח של הצומת x יופחת בערך delta>=0. אם הצומת שורש אז מפחיתים את המפתח ובודקים אם הוא המינימום החדש. אם הצומת פנימי: אם המפתח שלו עדיין גדול משל ההורה שלו, כלל הערימה נשמר ואפשר לסיים. אם כלל הערימה הופר- קוראים לcascadingCuts על הצומת x.	$O(n)$ כל הפעולות שהן לא הקריאה לcascadingCuts הן קבועות. לכן הסיבוכיות במקרה הרע היא הנגרמת מ-cascadingCuts.
nonMarked()	הפונקציה מחזירה את כמות האיברים שאינם marked בערימה. אנחנו מתחזקים שדה שסופר את מספר האיברים המסומנים, ואת מספר האיברים הכללי, לכן נחזיר את ההפרש ביניהם.	$O(1)$ שימוש בשני שדות.
potential()	הפונקציה מחזירה את ערך הפוטנציאל הנוכחי של הערימה הפוטנציאל, כפי שהוגדר בשיעור, הינו: Potential = #trees + 2*#marked אלו שני שדות שאנו שומרים.	$O(1)$ שימוש בשני שדות.
totalLinks()	פונקציה סטטית זו מחזירה את מספר כל פעולות החיבור שבוצעו מתחילת ריצת התוכנית. פעולת חיבור היא הפעולה שמקבלת שני עצים מאותו סדר ומחברת אותם. אנו נתחזק שדה סטטי של int, שבכל פעולת link נוסיף לו 1.	$O(1)$ גישה לשדה.
totalCuts()	פונקציה סטטית זו מחזיקה את מספר כל פעולות החיתוך שבוצעו מתחילת התכנית. גם לצורך זה, נתחזק שדה סטטי שיספור כל פעם בה אנו מבצעים חיתוך.	$O(1)$ גישה לשדה.
kMin(FibonacciHeap H, int k)	הפונקציה מקבלת ערימה H שהיא עץ בודד שדרגתו $\deg(H)$, ומספר חיובי $k < size(H)$. הפונקציה מחזירה מערך ממוין של k המפתחות הקטנים ב-H. האיבר המינימלי ב-H הוא כמובן השורש x, האיבר המינימלי אחריו הוא בהכרח אחד מהבנים שלו נניח y, והאיבר השלישי בגודלו הוא בהכרח אחד מהבנים האחרים של x או אחד מהבנים של y וכך הלאה. נאתחל ערימת עזר, שבה נחזיק בכל שלב באלגוריתם חתך של הערימה המקורית. בכל שלב נוציא את האיבר המינימלי מערימת העזר לתוך הפלט, ונוסיף את בניו (מהערימה המקורית H) אל ערימת העזר. *הערה: על מנת לגשת מערימת העזר לערימה המקורית H כדי להוסיף את הילדים החדשים אנחנו שומרים מצביע בכל node אל ה-node המקורי בערימה H.*	$O(k \cdot degH)$ כמות הילדים שאנו מוסיפים לערימת העזר בכל פעם, חסומה על ידי $degH$ שכן כמות הילדים של השורש היא הגדולה ביותר, ודרגה זו יורדת ככל שמתקדמים בעץ. כיוון שאנו מבצעים k איטרציות, ובכל פעם $O(degH)$ עבודה לכל היותר, סה"כ הסיבוכיות תהיה $O(k \cdot degH)$. נשים לב כי בנוסף להוספת הילדים, אנו גם מבצעים k פעמים deleteMin בערימת העזר. בכל איטרציה i יש לכל היותר $\log(i * degH)$ צמתים שעברו כבר consolidate קודם לכן, ובנוסף עוד $degH$ צמתים נוספים (מכאן הסיבוכיות העיקרית נובעת). כאשר נסכום על פני כל האיטרציות, סדרת הפעולות תרוץ בזמן: $O(k \cdot degH)$

חלק תיאורטי

שאלה 1

סעיף א:

ננתח את זמן הריצה של סדרת הפעולות:

- הכנסה: כל הכנסה בזמן קבוע, מבצעים $m+1$ הכנסות כאלו, סה"כ $O(m)$.
- מחיקה: פעולה אחת שתעלה לנו $O(m)$.
- decrease: כל פעולה ב-$O(1)$ כיוון שלא נמחק לאף צומת יותר מבן אחד (לכן לא נבצע cascading cuts אלא cut אחד). אנו מבצעים $logm$ פעולות כאלו, סה"כ $O(logm)$.

סה"כ סיבוכיות סדרת הפעולות היא $\boldsymbol{O(m)}$.

סעיף ב:

m	Run-Time (ms)	totalLinks	totalCuts	Potential
2^5	0.299	31	5	14
2^{10}	1.525	1023	10	29
2^{15}	13.886	32767	15	44
2^{20}	82.182	1048575	20	59

סעיפים ג-ו:

case	totalLinks	totalCuts	Potential	decreaseKey max cost
(c) original	$m-1$	$logm$	$3logm-1$	
(d) decKey	$m-1$	0	1	
(e) remove line #2	0	0	$m+1$	
(f) added line #4	$m-1$	$2logm-1$	$2logm$	$logm-1$

הסברים:

- totalLinks:
  - בכל המקרים, סדרת ההכנסות העצלות מביאה אותנו לערימה שמכילה $m+1$ צמתים יחידים. לאחר ביצוע deleteMin, נישאר עם m צמתים, ונבצע consolidate. בכל שלב אנו מחברים את העצים בזוגות, ומצמצמים את מספר העצים בחצי. להלן חישוב כמות החיבורים שנבצע:

$$\sum_{i=1}^{logm} \frac{m}{2^i} = m \sum_{i=1}^{logm} \frac{1}{2^i} = m \cdot \left(1 - \frac{1}{m}\right) = m-1$$

  - סה"כ נקבל $m-1$ כאלו.
  - נשים לב כי בגרסה e – כאשר לא מבצעים deleteMin לא נבצע כלל links.
- totalCuts:
  - בגרסה c – בכל איטרציה של decreaseKey אנו חותכים צומת אחד בלבד. כיוון שלכל צומת אנחנו חותכים לכל היותר בן אחד, לא נבצע cascading cuts.
  - בגרסה d – נבחין כי אנו מבצעים decreaseKey לבן הראשון (השמאלי ביותר) של השורש, ואז לבנו השמאלי, וכן הלאה בלולאה. כיוון שאנו מפחיתים את ערך המפתח ב-$m+1$ כל פעם (תחילה לאבא ואז לבנו) לא נצטרך לבצע חיתוכים. כלל הערימה נשמר לכל אורך סדרת הפעולות.
  - בגרסה e – במקרה זה, יש לנו אך ורק צמתים יחידים, ואין שום צורך לבצע חיתוכים.

- בגרסה f – בגרסה c גרמנו לשדרה השמאלית של העץ להיות כולה מסומנת (פרט לשורש ולעלה השמאלי ביותר). לכן, כאשר נבצע decreaseKey למפתח $m-2$, נגרום לשרשרת של חיתוכים (cascading cuts) להיכנס לפעולה. נוסף על $logm$ החיתוכים שבוצעו קודם (בגרסה c), כעת נבצע $logm-1$ חיתוכים נוספים, עבור כל צומת מסומן בשדרה השמאלית (כי השורש אינו מסומן). סה"כ נקבל $2logm-1$ חיתוכים.

- Potential (נסמנו p):
  - בגרסה c – תחילה יש לנו עץ אחד לאחר ההכנסות וdeleteMin ($p=1$). לאחר מכן בכל חיתוך אנו מסמנים את ההורה של הצומת שנחתך ($p+=2$), והצומת שנחתך יוצר עץ חדש ($p+=1$), סה"כ תוספת של 3 לפוטנציאל בכל חיתוך. נשים לב כי את השורש לא נסמן ולכן במקרה שלו הפוטנציאל יעלה רק ב-1 ולא ב-3. סה"כ נקבל:
  $$p=1+3logm-2=3logm-1$$
  - בגרסה d – אין חיתוכים. לכן נישאר עם הפוטנציאל ההתחלתי שהוא 1.
  - בגרסה e – אין חיתוכים לכן אין צמתים מסומנים, לכן הפוטנציאל הוא מספר העצים שהוא $m+1$.
  - בגרסה f – שרשרת החיתוכים שמתבצעת מורידה את סימון ההורים בשדרה השמאלית ($p-=2$), ויוצרת עץ חדש בכל חיתוך ($p+=1$). נשים לב כי גם כאן השורש לא לוקח חלק, ולכן בהינתן הפוטנציאל הקודם (גרסה c) נקבל:
  $$p=3logm-1-(logm-1)=2logm$$

- decreaseKey max cost:
  - כפי שתיארנו בגרסה f של הפוטנציאל, שרשרת החיתוכים הייתה באורך של $logm-1$ צמתים.

שאלה 2

סעיף א:

m	Run-Time (ms)	totalLinks	totalCuts	Potential
728	2.643	723	0	6
6560	6.008	6555	0	6
59048	32.135	59040	0	9
531440	161.236	531431	0	10
4782968	1370.725	4782955	0	14

סעיף ב:

ננתח את זמן הריצה של סדרת הפעולות:

- הכנסה: כל הכנסה בזמן קבוע, מבצעים $m+1$ הכנסות כאלו, סה"כ $O(m)$.
- מחיקת מינימום: כל מחיקת מינימום עולה $O(logm)$ באמורטייזד, ויש $\frac{3m}{4}$ כאלו, אז סה"כ $O(mlogm)$.

סה"כ סיבוכיות סדרת הפעולות היא $\boldsymbol{O(mlogm)}$.

סעיף ג:

- totalLinks:
  - נשים לב כי ההכנסה מתבצעת בסדר עולה, כיוון שאנחנו שומרים על סדר בהכנסה (צומת חדש משמאל לצמתים ותיקים יותר) נקבל את הצמתים החדשים ביותר משמאל. לאחר מכן, אנחנו מבצעים consolidate לפי סדר הרשימה, ולכן נטפל תחילה במפתחות הגדולים יותר (החדשים יותר) ומשם בסדר יורד עד שנגיע למפתחות הקטנים ביותר (הותיקים ביותר).
  - העצים הגדולים ביותר ברשימה הסופית הם העצים שנוצרו ראשונים (אחרת הם היו מתאחדים עם העצים הקטנים יותר). באופן זה, לאחר ביצוע deleteMin ראשון, נקבל יער של עצים שבו **העצים הקטנים יותר מכילים את האיברים הקטנים ביותר**.
  - כל פעולת deleteMin נוספת, תמחק איברים מהעץ הקטן ביותר. כתוצאה מכך ייווצרו עצים קטנים יותר, והם לא יתאחדו עם העצים הגדולים יותר (העצים לא מספיק גדולים עבור זה). כלומר – **לא יתרחשו יותר links משלב זה.**

  מספר החיבורים הסופי יהיה מספר החיבורים לאחר המחיקה הראשונה בערימה. בכל עץ מספר החיבורים הוא מספר הצמתים פחות 1 (כל צומת מחובר להורה שלו פרט לשורש). לכן מספר החיבורים הכללי הוא מספר הצמתים פחות מספר העצים.
  - מספר הצמתים: לאחר המחיקה הראשונה הוא **m**.
  - מספר העצים: מספר החזקות של 2 שמרכיבות את m, ניתן לחשוב על כך כ**כמות האחדות בייצוג הבינארי של m**.
- totalCuts: בבירור מספר החיתוכים הוא **0**, כיוון שבשום שלב לא ביצענו decreaseKey.
- Potential: כיוון שלא ביצענו חיתוכים, הפוטנציאל יהיה מספר העצים שנותרו לאחר כלל המחיקות. כפי שתיארנו קודם, מספר העצים הוא **כמות האחדות בייצוג הבינארי** של מספר הצמתים שנותרו: $m+1-\frac{3m}{4}=\boldsymbol{\frac{m}{4}+1}$.