תוכן עניינים

חלק מעשי

תיאור המחלקה שמומשה - FibonacciHeap

המחלקה יוצרת אוביקט של ערימת פיבונצ׳י.

קבועים (שדות מחלקה)

1. **total_links:** מספר פעולות החיבור (links) שבוצעו מתחילת ריצת התוכנית. פעולת חיבור היא הפעולה שמקבל שני עצים מאותו סדר (כלומר, לשורשי העץ אותו מספר ילדים) ומחברת אותם.
2. **total_cuts:** מספר פעולות החיתוך שבוצעו מתחילת ריצת התוכנית. פעולת חיתוך מתרחשת עקב decreaseKey, כאשר מנתקים תת עץ מאביו (cascading cuts).

שני השדות מאותחלים ל-0 ומתוחזקים בשירותי המחלקה.

שדות (שדות מופע)

1. Min : מצביע לצומת (שורש עץ) עם המפתח המינימלי.
2. First : מצביע לצומת (שורש עץ) הראשון, כלומר זה שהוכנס אחרון (על פי התנאים שהוגדרו במטלה).
3. total_Trees : מספר העצים בערימה.
4. total_Marks : מספר הצמתים המסומנים בערימה. (הבהרה : שורש של עץ לא מסומן לעולם. שני שדות אלה נדרשים על מנת לחשב את פונקצית הפוטנציאל ביעילות).
5. Size : מספר הצמתים בערימה.

בנאי

1. בנאי ריק : מאתחל את השדות First, Min ל-null, ואת שאר השדות ל-0.

מתודות שנדרשנו לממש

isEmpty()

מה היא עושה : מחזירה TRUE אם ורק אם הערימה ריקה.

כיצד היא פועלת :

המתודה בודקת האם הרשימה ריקה ע״י בדיקה אם השדה First המצביע לאיבר הראשון בערימה הינו null.

סיבוכיות זמן ריצה : פעולה בזמן קבוע O(1).

Insert(int key)

מה היא עושה : יוצרת צומת מסוג HeapNode שמכיל את המפתח key ומכניסה אותו לערימה. מחזירה את הצומת שנוצר (עם המפתח key).
האיבר החדש ייכנס לראש הערימה (משמאל לרשימת העצים).

כיצד היא פועלת :

תחילה המתודה יוצרת את הצומת החדש שנכניס לעץ ע״י בנאי של HeapNode. לאחר מכן בודקת האם העץ ריק : אם כן המתודה תגדיר את הצומת החדש בתור המינימום, ואם לא תבדוק האם הינו המינימום

(אם כן תעדכן) ותשתמש במתודת העזר InsertAsFirst. לאחר מכן תגדיל את השדות size ו-total_trees של העץ ב-1 ותקבע את הצמת החדש בתור First.

סיבוכיות זמן ריצה:

המתודה מבצעת פעולות בזמן קבוע וקוראת למתודה המבצעת פעולות בזמן קבוע (InsertAsFirst) ולכן סיבוכיות זמן הינה קבועה O(1).

deleteMin()

מה היא עושה: מוחקת את הצומת עם המפתח המינימלי בין המפתחות שבערימה.

הצאצאים של הצומת שנמחקה, יישארו עם אותו סדר יחסי ביניהם. בנוסף, הם גם יישארו באותו מיקום יחסי ברשימה, ולא יזוזו להתחלה.

כיצד היא פועלת:

א. ראשית בודקת שהעץ לא ריק. אם העץ לא ריק מבצעת את הפעולות הבאות.
ב. מורידה את ה-size באחד.
ג. שומרת את הצומת עם המפתח המינימלי כ-min ("צומת המינימום"). (נשים לב שהצומת הזה הוא תמיד שורש של עץ). שומרת את הדרגה שלו, כלומר את מספר הילדים שלו, כ-k.
ד. חלוקה למקרים:
   1. **אם בערימה יש עץ אחד (total trees =1):**
      a. **לצומת המינימום אין בנים (k=0):** מחיקת המינימום, תשנה את הערימה לערימה ריקה. לכן נשנה את השדות של הערימה ל-null ול-0 לפי הצורך.
      b. **לצומת המינימום יש בנים (else):** במקרה זה, כל הילדים של הצומת יהפכו להיות שורשי העצים בערימה. נעדכן את השדה First שיצביע לבן הראשון של צומת המינימום.
      נעבור על כל הבנים בלולאה, שרצה k פעמים כי זה מספר הילדים. בתוך הלולאה, נעדכן את ההורה של כל אחד מהם להיות null, וכן נעשה לו unmark (כי הוא הפך לשורש).
      (לא נשנה את השדה Min של הערימה כיון שבכך מטפלת הפונקציה consolidate שנקראת בסוף).
   2. **אם בערימה יש יותר מעץ אחד (else):**
      שומרים את השורש הבא אחרי צומת המינימום, ואת השורש הקודם ב-n ו-p בהתאמה.
      a. **אם לצומת המינימום אין בנים (k=0):** מחברים בין השורשים n ו-p (כך ש"ידלגו" מעל צומת המינימום).
      אם צומת המינימום היה First בערימה, מעדכנים את ה-First להיות n (השורש שבא אחריו).
      b. **לצומת המינימום יש בנים (else):** שומרים את הבן הראשון ואת הבן האחרון.
      עוברים בלולאה על כל הבנים, ומעדכנים אותם לפי הצורך, כאמור לעיל במקרה 1.
      מוסיפים את הילדים לרשימה המקושרת של שורשי העץ, על ידי שינוי המצביעים המתאימים, של n, p, הבן הראשון והבן האחרון.
      אם צומת המינימום היה First בערימה, מעדכנים את ה-First להיות הבן הראשון.
      בכל מקרה, לא נשנה את השדה Min של הערימה כיון שבכך מטפלת הפונקציה consolidate.

ה. קריאה לפונקציה consolidate.

סיבוכיות זמן ריצה:

1. פעולות בזמן קבוע.
2. לולאה שרצה k פעמים. k = דרגת צומת המינימום. בערימה עם n איברים חסומה ע"י O(log(n)).
3. קריאה לפונקציה consolidate: במקרה הגרוע O(n), amortized O(log(n)).

סה"כ במקרה הגרוע O(n), amortized O(log(n)).

findMin()

מה היא עושה: מוצאת את הצמת בעלת ערך המפתח המינימלי בערימה.

כיצד היא פועלת: מחזירה את השדה Min של הערימה.

סיבוכיות זמן ריצה: פעולה בזמן קבוע - O(1).

meld(FibonacciHeap heap2)

מה היא עושה: ממזגת את הערימה הנוכחית (this) עם ערימה נוספת, heap2.
משרשרים את היער של הערימה heap2 אחרי רשימת העצים של הערימה הנוכחית (this).

כיצד היא פועלת: המתודה תחילה בודקת אם heap2 ריק אם כן אין צורך לבצע כלום.

אם heap2 אינו ריק אז נבדור אם this ריק. אם כן אז נשנה את השדות מופע של this להיות אלו של heap2 . אם this גם לא ריק אז תחילה נשנה את השדות בהתאמה: נסכום size, total_Marks, total_Trees. אז נשווה בין המינימיום של הערימות ונקבע מינימיום חדש (השדה Min) להיות הקטן מביניהם. לאחר מכן נשרשר את שתי הרשימות המקושרת של השורשים דרך עדכון האיבר הראשון והאחרון בכל רשימת שורשים ומצביעי next,prev שלהם כך שנקשר את השורש הימני ביותר של this עם השמאלי ביותר של heap2 ואת הימנמי ביותר של heap2 עם השמאלי ביותר של this.

סיבוכיות זמן ריצה: פעולות בזמן קבוע O(1).

size()

מה היא עושה: מחזיר את כמות הצמתים בערימה.

כיצד היא פועלת: תחזירה את השדה size

סיבוכיות זמן ריצה: זמן קבוע O(1)

countersRep()

מה היא עושה: המתודה מחזירה מערך מונים כך שבאינדקס i מופיע מספר העצים בערימה שהינם מסדר i.

כיצד היא פועלת: תחילה המתודה יוצרת מערך בגודל 1+ Ceiling_log(this.size()) ואחר כך עוברת על כל השורשים ברשימת השורשים של העץ, כל שורש מדרגה i יוסיף 1 למקום ה-i במערך בנוסף נשמור int שיהיה הדרגה המקסימלית שנמדדה עד כה בעץ. לאחר מכן נעבור שוב על המערך כדי למצוא את הדרגה המקסימלית בו ולחתוך את החלקים הריקים שלו דרך כך שניצור מערך ריק באורך הדרגה המקסימלית ונעתיק את האיברים הראשונים המערך המקורי אליה.

סיבוכיות זמן ריצה:

משתמשים ב-Ceiling_log(n) וסיבוכיות זמן שלו הינה O(log(n))

תחילה מבצעים לולאה שתעבור על כל השורשים של העץ וכל איטרציה בה בזמן קבוע (במקרה הגרוע ביותר רשימת השורשים בעץ הינה האורך n) לולאה זו תוסיף סיבוכיות זמן של O(n).

לאחר מכן מבצעים מעבר על ערכי מערך שאורכו לא עולה על 1 + log(n) וכל איטרציה היא בזמן קבוע. לולאה זו תוסיף סיבוכיות זמן של O(log(n)).

סה"כ נקבל סיבוכיות זמן במקרה הגרוע ביותר O(n).

delete (HeapNode x)

מה היא עושה: מוחקת מהערימה את הצומת x.

כיצד היא פועלת: תחילה מבצעת decreaseKey ל-x לערך הקטן מהמינימום ואז קוראת לפונקציה deleteMin.

סיבוכיות זמן ריצה:

במקרה הגרוע: decreaseKey זה O(log(n)) ו-deleteMin זה O(n), לכן במקרה הגרוע זה O(n).

Amortized: decreaseKey זה O(1) ו-deleteMin זה O(log(n)), לכן O(log(n)).

decreaseKey(HeapNode x, int delta)

מה היא עושה: מפחיתה את ערכו של המפתח של הצומת x, ב-delta. $delta \geq 0$.

אם לאחר עדכון המפתח אין הפרה של תנאי הערימה בין הצומת לאביו, לא מתבצע חיתוך. אם התבצע חיתוך, תת-העץ שנחתך ייכנס לראש הרשימה (מצד שמאל).

כיצד היא פועלת:

1. מחסירה delta מערך המפתח של הצומת x.
2. אם ערך המפתח החדש קטן מהמפתח של צומת המינימום בערימה: מעדכנת את צומת המינימום של הערימה להיות x.
3. אם הצומת x אינו שורש (ההורה שלו אינו null) וגם המפתח החדש של x קטן מהמפתח של ההורה שלו:

א. ננתק את הצומת x מההורה שלו, באמצעות קריאה למתודת העזר cutter, עם הפרמטרים המתאימים.

ב. אם ההורה היה unmark: לא ניכנס ללולאת ה-while. נסמן את ההורה, וסיימנו.

ג. אם ההורה היה marked: במקרה זה, נצטרך לנתק את ההורה, מההורה שלו. כלומר צריך להמשיך "לעלות" למעלה בעץ, כל עוד ההורה marked. נעשה זאת בלולאת while, כל עוד יש הורה, וכל עוד ההורה הוא marked. אם זה מתקיים, ננתק את הבן מההורה באמצעות cutter. הלולאה תסיים לרוץ, או כאשר נגיע לצומת שאינו מסומן, או כאשר נגיע לשורש העץ. מחוץ ללולאה, נסמן את ההורה של הצומת האחרונה אותה חתכנו, באמצעות קריאה לפונקציה Mark (הפונקציה בודקת אם הצומת הוא שורש, ואם הוא שורש לא מסמנת אותו).

סיבוכיות זמן ריצה:

פעולות בזמן קבוע + לולאה של קריאות ל-cutter שפועל בזמן קבוע. מכיוון שהלולאה נעה במעלה עץ בערימה, אז מספר האיטרציות חסום ע"י הגובה המקסימלי של עץ בערימה, שהוא O(log(n)).

סה"כ סיבוכיות זמן Worst-Case: O(log(n)).

כעת נראה חסם amortized: נשים לב כי סיבוכיות הזמן של הפעולה עצמה כאשר נבצע c חיתוכים הינו c+1 כלומר c פעולות חיתוך ע"י cutter בזמן קבוע ועוד שאר פעולות בזמן קבוע:

$$Real\ Time = c + 1$$

כעת נבחין כי מספר הצמתים המסומנים יורד ב-(c-1) כי נבצע חיתוך ראשון ואז שאר החיתוכים ינבעו מהורים מסומנים. בנוסף כל חיתוך ייצור עץ נוסף בערימה כלומר c עצים חדשים. מכאן ההפרש בפונקצית הפוטנציאל יהיה:

$$\Delta\phi = c - 2(c-1) = -2 - c$$

סה"כ נקבל:

$$amort\ time = Real\ Time + \Delta\phi = c + 1 - 2 - c = -1 = O(1)$$

ומכך נקבל חסם amortized של O(1).

potential()

מה היא עושה: מחזירה את ערך הפוטנציאל הנוכחי של הערימה. הפוטנציאל כפי שהוגדר בשיעור הוא: מספר העצים + פעמיים מספר הצמתים המסומנים.

כיצד היא פועלת: מחזירה את התוצאה של total_Trees + 2 * total_Marks.

סיבוכיות זמן ריצה: פעולה אריתמטית בודדת O(1).

totalLinks()

מה היא עושה: פונקציה סטטית, שמחזירה את מספר כל פעולות החיבור שבוצעו מתחילת ריצת התוכנית. פעולת חיבור היא הפעולה שמקבלת שני עצים מאותו סדר (rank) ומחברת אותם.

כיצד היא פועלת: מחזירה את השדה הסטטי total_Links.

סיבוכיות זמן ריצה: זמן קבוע O(1).

totalCuts()

מה היא עושה: פונקציה סטטית, שמחזירה את מספר כל פעולות החיתוך שבוצעו מתחילת ריצת התוכנית.

כיצד היא פועלת: מחזיר את השדה הסטטי total_Cuts.

סיבוכיות זמן ריצה: זמן קבוע O(1).

kMin(FibonacciHeap H, int k)

מה היא עושה: פונקציה סטטית. מקבלת ערימה H שהיא עץ (יער של עץ יחיד) שדרגתו deg(H), ומספר חיובי k<size(H). הפונקציה מחזירה מערך ממוין של k הצמתים הקטנים ב-H.

כיצד היא פועלת:

המתודה מקבלת ערימה של עץ אחד. תחילה יוצרת מערך ריק arr באורך k

לאחר מכן יוצרת ערימה ריקה ומכניסה צמת עם המפתח של השורש ומצביע שמצביע אליו (Xtrapointer). לאחר מכן מבצעת k פעמים את הפעולה הבאה:

תחילה מוצאת את המינימום של הערימה מכניסה אותו למערך arr ומסירה אותו דרך deleteMin. אז, דרך מצביע Xtrapointer מגיעה אל המיקום של הצמת בעץ המקורי (מיקום של הצמת בעל מפתח דומה) ואז מכניסה עבור כל בן של הצמת, צמת בעלת ערך מפתח דומה ומצביע אשר מצביע לבן לתוך הערימה.

לאחר שבוצעו k איטרציות ומילאנו את המערך arr נחזיר אותו.

סיבוכיות זמן ריצה:

תחילה נשים לב כי בכל איטרציה נוסיף לכל היותר deg(H) איברים מכיוון שלכל צמת בעץ H מספר הבנים שלה הינו הדרגה של הצמת שקטנה או שווה לדרגת העץ. ולכן ב-amortized כל פעולת deleteMin הינה בעלת חסם עליון = O(log(k deg(H)))

ומכיוון שנבצע k איטרציות נקבל חסם עליון $O(k\log(k \cdot deg(H)))$ כעת נניח כי k<H.size() אחרת לא נוכל לבצע את הפעולה מלכתחילה. בנוסף הדרגה של H גם חסומה ע״י הגודל של H. מכאן:

$$k\log(k \cdot deg(H)) \leq k\log(H.size^2) = 2\,k \cdot \log\,(H.size)$$

אך נשים לב כי בעץ בינומי $\deg(H) = \theta(\log\,(H.size))$ ולכן:

$$2\,k \cdot \log(H.size) = O(k \cdot \deg(H))$$

מתודות עזר

InsertAsFirst(HeapNode curr_first, HeapNode new_first)

מה היא עושה: הפונקציה מקבלת איבר ראשון של רשימה מקושרת דו-כיוונית (רשימת בנים של צמת או רשימת שורשים) כלשהיא ואיבר חדש ומכניסה אותו בין האיבר הראשון לאחרון. הנחת הקדם היא שהאיבר החדש אינו null.

כיצד היא פועלת: מקבלת שני צמתים. אם curr_first הוא null אז הרשימה המקושרת אליה מנסים להכניס את האיבר החדש ריקה, ונכניסו כאיבר בודד שצמביע לעצמו. אחרת נכניסו בין האיבר הראשון לאחרון דרך שינויי השדות next ו-prev.

סיבוכיות זמן ריצה: פעולות בזמן קבוע O(1).

link(HeapNode node1, HeapNode node2)

מה היא עושה: מחברת שני עצים מאותו הסדר (rank). השורש שכבר-לא-שורש יהפוך להיות הבן הראשון (השמאלי ביותר) של השורש-שנשאר-שורש. מחזירה את השורש של העץ החדש שנוצר.

מגדילה את מספר הלינקים שנעשו בתוכנית ב-1.

כיצד היא פועלת:

1. מקבלת שני צמתים שהם לא null, ושה-rank שלהם שווה.
2. מגדילה את מספר הלינקים שנעשו בתוכנית ב-1.
3. בודקת למי מצמתים מפתח יותר קטן, ומחזירה את תוצאת הפונקציה linkHelper שהופכת את הצומת עם המפתח הגדול יותר, לבן של הצומת עם המפתח הקטן יותר. הפונקציה linkHelper מחזירה את השורש של העץ החדש שנוצר.

סיבוכיות זמן ריצה: פעולות בזמן קבוע + קריאה לפונקציה linkHelper שפועלת בסיבוכיות זמן O(1), סה"כ O(1).

linkHelper(HeapNode dad, HeapNode kid)

מה היא עושה: מקבלת שני צמתים שהם לא null, עם rank שווה, שהראשון ביניהם הוא עם מפתח קטן יותר. מוסיפה את הצומת השני כבן של הצומת הראשון. מחזירה את השורש של העץ החדש שנוצר.

כיצד היא פועלת:

1. מקבלת שני צמתים, אחד מיועד להיות אב והשני בן.
2. נגדיל את ה-rank של parent ב-1.
3. נכניס את הבן כראשון ברשימת הבנים של האב דרך InsertAsFirst.
4. נעדכן את השדה parent של kid להיות dad, ואת השדה child של dad להיות kid.
5. נחזיר את ההורה.

סיבוכיות זמן ריצה: כל הפעולות הן פעולות בזמן קבוע, לכן O(1).

cutter(HeapNode child, HeapNode parent)

מה היא עושה: חותכת תת עץ מההורה שלו, והופכת אותו לעץ חדש.
הצומת שנחתך מאביו ייכנס לראש הערימה (מצד שמאל).
בנוסף, אם להורה היתה רשימת בנים, סדר הבנים לא משתנה, פרט לבן שנחתך.

כיצד היא פועלת: תחילה תבדוק את דרגת ה-parent :

אם הינה 1 אז כדי לנתק את בן מן האב נשנה את השדה child ל-null. אחרת נחבר בין הקודם של child והבא אחריו ובכך מעשית "נמחק אותו מרשימת הבנים" אלא אם היה הבן הראשון ואז נשים את ההבא אחריו להיות הבן של parent דרך שינוי שדה child.

לאחר מכן נוריד את השדה rank של parent ב-1 נשנה את השדה parent של child ל-null. נבצע פעולת unmark על child (אסור שורשים מסומנים) נעלה את total_trees ו-total_Cuts ב-1 נכניס את child

לרשימת השורשים דרך InsertAsFirst ונקבע את השדה First של הערימה להיות child כדי לשים אותו כראשון משמאל.

סיבוכיות זמן ריצה: פעולות בזמן קבוע + קריאה ל-unmark ‏(O(1)) + קריאה ל-InsertAsFirst ‏(O(1)) סה״כ O(1).

Ceiling_log(int n)

מה היא עושה: מחשבת חסם עליון לדרגה בעץ בעל n איברים.

כיצד היא פועלת:

חסם עליון הינו: $\log_{\varphi} n \approx 1.44 \log_2 n \leq 1.5 \log_2 n$

לכן נחשב לוגריתם על בסיס 2 של n ונכפיל ב-1.5 וניקח ערך שלם תחתון.

שומרת שני int ראשון i שישמור את מספר החזקה של ה-int השני k. נבצע את האיטרציה הבאה כל עוד k<n נכפיל את k בשתיים ונעלה את i באחד – כך i ימשיך להיות החזקה ב-2 של k) לערך הראשון שאינו קטן מ-n נחזיר את i וכך נקבל ערך תקרה ללוגריתם של n.

כעת נכפיל את i ב-3 ונחלק ב-2 כדי לקבל חסם עליון לדרגה מקסימלית בעץ .

סיבוכיות זמן ריצה : פעולות בזמן קבוע + לולאה בה נבצע log(m)+1 איטרציות לכל היותר וכל איטרציה בזמן קבוע. על כן סיבוכיות זמן ריצה הינה O(log n).

consolidate()

מה היא עושה: אחרי deleteMin, עוברת על כל העצים ויוצרת מהערימה, ערימה בינומית לא עצלה. מבוצע במלואו ולא one-pass. רשימת העצים הסופית, החדשה, מסודרת לפי דרגות בסדר <u>עולה</u>, כלומר העץ השמאלי ביותר הוא בעל הדרגה הכי נמוכה.

כיצד היא פועלת:

אם העץ ריק – לא עושה כלום.

אם העץ לא ריק: נחלק את המתודה לשני חלקים:

א. הכנסה של עצים למערך העזר.

ב. הוצאה של העצים מהמערך, ומציאת המינימום החדש.

בחלק א׳, אנחנו לא מעדכנים את מספר העצים, למרות שפעולות link משנות את המספר, כיוון שהמספר יתעדכן בחלק ב׳.

<u>חלק א׳ – הכנסה של עצים למערך</u>

1. ניצור מערך עזר של צמתים. אורך המערך הוא $\lceil 1.5 \cdot \log(n) \rceil + 1$ (נחשב את $\lceil \log(n) \rceil$ ע״י קריאה לפונקציה ceiling log על מספר הצמתים בערימה, שמסומן כאן ב-n).
2. נקרא לשורש של העץ הראשון, ש-First מצביע אליו, current.
3. נכניס אותו למערך, באינדקס שהוא ה-rank שלו.
4. נקדם את current להיות current.next ונתחיל לרוץ בלולאה. הלולאה רצה כל עוד לא חזרנו לשורש של העץ הראשון.
5. בתוך הלולאה:
   א. שומרים את current כ-loop_node. מקדמים את current להיות current.next. כעת יש לנו שורש, loop_node, והשורש שבא אחריו, current. אנחנו רוצים להכניס למערך את loop_node.
   ב. שומרים את הדרגה של loop_node במשתנה k.

ג. **אם האינדקס ה-k במערך ריק (null):** לא נכנסים ללולאת ה-while הפנימית. מכניסים את loop_node למערך באינדקס ה-k וממשיכים לצומת הבאה.
ד. **אם האינדקס ה-k "תפוס" (לא null):** נכנסים ללולאה הפנימית, ורצים בה כל עוד האינדקס שאליו אנחנו רוצים להכניס את העץ תפוס.
   1. אם התא הזה "תפוס", זה אומר שיש שם עץ בדרגה של האינדקס, כלומר בדרגה k. לכן נעשה link לעץ שנמצא שם, ולעץ ש-loop_node הוא השורש שלו.
   2. הפונקציה link מחזירה את השורש של העץ החדש שנוצר. נדרוס את loop_node, ונשמור בו את תוצאת הפונקציה.
   3. נרוקן את התא ה-k כלומר נשים בו null.
   4. נעדכן את k להיות הדרגה של השורש של העץ החדש שנוצר (כלומר של loop_node).

<u>חלק ב' – הוצאה מהמערך, ומציאת המינימום החדש</u>

1. נאתחל משתנה i להיות 0. הוא יסמן את האינדקס שלנו במערך.
2. נאתחל משתנה שישמור את מספר העצים (במבנה הערימה החדש שנוצר), ונאתחל אותו ל-0.
3. נאתחל שלושה משתנים ל-null: minimal שבו נשמור את צומת המינימום החדש, new_first שבו נשמור את הצומת הראשון החדש, ו-previous, שתפקידו לעזור לנו לקשר בין השורשים בערימה החדשה.
4. נרוץ בלולאה על המערך, כלומר כל עוד i קטן ממש מגודל המערך.
   a. אם התא במערך שהגענו אליו אינו ריק, כלומר יש בו שורש של עץ, נעשה:
      i. נגדיל את מספר העצים ב-1.
      ii. אם המשתנה שבו אנחנו שומרים את הצומת הראשון החדש הוא null (כלומר עוד לא שמנו בו שורש), נשים בו את הצומת שבתא. זה קורה פעם אחת, רק בתא הלא-ריק הראשון שאנחנו פוגשים. שם נמצא העץ מהדרגה הנמוכה ביותר, וזה העץ שאנחנו רוצים שיהיה ראשון בערימה החדשה.
      iii. אם המשתנה של הצומת המינימלי החדש הוא null, נשמור בו את הצומת שבתא. זה יקרה בתא הלא-ריק הראשון שנפגוש. אחרת, כלומר החל מהתא הראשון, נבדוק אם המפתח של הצומת שבתא, קטן מהמפתח של הצומת המועמד להיות מינימלי. אם כן, נעדכן את הצומת המועמד להיות זה שבתא.
      iv. אם המשתנה previous הוא null, נשים בו את הצומת החדש. כשהוא לא null, כלומר אחרי ששמנו בו את העץ הראשון שמצאנו, נעשה:
      בשדה next שלו נשים את הצומת שבתא.
      בשדה prev של הצומת שבתא, נשים אותו.
5. בסיום הלולאה, previous מסמן את העץ שהיה בתא הימני ביותר במערך. כלומר העץ עם הדרגה הגבוהה ביותר בערימה, ונרצה שהוא יהיה הכי ימני בערימה כלומר האחרון. נחבר אותו לצומת הראשון החדש: ה-next שלו יצביע לצומת הראשון. ה-prev של הצומת הראשון יצביע אליו.
6. נעדכן את שדות הערימה – Min, First ו-total_Trees בהתאם למשתנים שעדכנו. (מספר הצמתים בערימה לא השתנה. גם מספר הצמתים המסומנים לא השתנה: במהלך פעולות link רק הפכנו צמתים שהיו שורשים, ללא-שורשים. צמתים שהם שורשים לא היו מסומנים מלכתחילה).

<u>סיבוכיות זמן ריצה במקרה הגרוע</u>: O(n)

הזמן שלוקח הטיפול בעץ ה-i הוא מספר הלינקים שהעץ עובר + 1. (בלינקים אנחנו כוללים גם מעבר לתא הבא, פעולה בזמן קבוע שלא משנה אסימפטוטית). נסמן את מספר הלינקים שהעץ ה-i עובר ב- L_i.

נסמן את מספר העצים <u>לפני שקראנו ל-deleteMin</u> ב-T. אחרי deleteMin מספר העצים הוא T+k-1, כאשר k הוא הדרגה של צומת המינימום שמחקנו, כלומר מספר הילדים שלו. כלומר, הורדנו עץ אחד, והוספנו k עצים חדשים במקומו.

אז הזמן הכולל שלוקח לפעולה, כלומר מעבר על כל העצים, הוא: $\sum_{i=1}^{T+k-1}(L_i+1)$.

נסמן ב-L את מספר הלינקים הכולל שמתבצע.

נקבל:

$$\sum_{i=1}^{T+k-1}(L_i+1)=\sum_{i=1}^{T+k-1}L_i+\sum_{i=1}^{T+k-1}1=L+T+k-1$$

מספר הלינקים חסום על ידי מספר העצים. כל לינק מוריד את מספר העצים באחד, אז לא ניתן לעשות יותר לינקים ממספר העצים. לכן

$$L+T+k-1\leq 2(T+k-1)$$

k, הדרגה של הצומת, בערימה עם n איברים חסומה על ידי log(n). אז:

$$2(T+k-1)\leq 2(T+\log(n))$$

במקרה הגרוע, מספר העצים הוא n ואז נקבל שהסיבוכיות היא:

$$O(T+\log(n))=O(n+\log(n))=O(n)$$

<u>כעת נראה סיבוכיות זמן אמורטייזד O(log(n)) דרך פונקציית הפוטנציאל</u>:

כבר הראנו כי מספר הפעולות שנבצע הינו מספר הלינקים שנבצע (נחליף את $T+k-1$ בסימון T_0:

$$Real\ Time=O(T_0+\log(n))$$

נעשה scale down ל - $T_0+\log(n)$:

$$Real\ Time=T_0+\log(n)$$

כעת נסמן את מספר העצים לאחר ה-consolidation ב-T_1 ומכיוון שמכל דרגה יש לכל היותר עץ בודד אז מספר העצים חסום ע״י הדרגה המקסימלית ולכן $T_1\leq\log(n)+1$

$$\Delta\phi=T_0-T_1$$

ואז נקבל :

$$amort\ time=\ Real\ Time+\Delta\phi=T_0+\log(n)-T_0+T_1=\ T_1+\log{(n)}=O(\log{(n)})$$

ולכן סיבוכיות זמן אמורטייזד הינה $O(\log{(n)})$

<u>Mark(HeapNode node)</u>

מה היא עושה: מסמנת שורש שנחתך לו בן פעם אחת.

הבהרה: שורשי העצים לעולם אינם מסומנים (כי אי אפשר לתלוש שורשים).

כיצד היא פועלת: תחילה בודקת אם הצומת אינו שורש ואינו מסומן. אם מתקיימים תנאים אלו נגדיל את השדה total_marks ב-1 ונשנה את שדה mark של node להיות true.

סיבוכיות זמן ריצה: פעולות בזמן קבוע O(1).

<u>unmark(HeapNode node)</u>

מה היא עושה: מורידה סימון לשורש.

כיצד היא פועלת: תחילה בודקת אם הצומת מסומן. אם מתקיים תנאי זה נקטין את השדה total_marks ב-1 ונשנה את שדה mark של node להיות false.

סיבוכיות זמן ריצה: פעולות בזמן קבוע O(1).

<u>המחלקה HeapNode</u>

שדות מופע:

1. key
2. rank - מספר הילדים הישירים של הצומת
3. mark – שדה בוליאני, מקבל true אם הצומת מסומן ו-false אם לא
4. child
5. next
6. prev
7. parent
8. Xtra_pointer – מצביע למיקום של הצומת בערימה אחרת, שדה בשביל המימוש של הפונקציה kMin

בנאי:

הבנאי מקבל מספר שלם, כמפתח של הצומת. הדרגה מאותחלת ל-0, mark מאותחל ל-false כי הצומת לא מסומן, הילד וההורה מאותחלים ל-null. prev, next מצביעים לעצמו.

מתודות:

כל הפונקציות הן פונקציות שנותנות גישה לשדות האוביקט.

<u>חלק ניסויי/תאורטי</u>

<u>שאלה 1</u>

<u>סעיף א'</u>

זמן הריצה האסימפטוטי הוא O(m).

הסבר:

1. הכנסה של m+1 איברים: הכנסה עולה O(1), סה״כ O(m+1) = O(m).
2. פעם אחת deleteMin: המחיקה עצמה עולה O(1), לאחר המחיקה נעשה consolidate. כיוון שלאחר המחיקה יש m איברים, זה יעלה O(m).
3. log(m) פעמים decreaseKey: הפעולה decreaseKey עולה amortized O(1) ולכן סה״כ כל הפעולות יעלו O(log(m)).

<u>סעיף ב'</u>

m	Run-Time(ms)	totalLinks	totalCuts	Potential
2^{10}	1.8771	1023	10	29
2^{15}	12.5739	32767	15	44
2^{20}	108.1351	1048575	20	59
2^{25}	11242.2803	33554431	25	74

<u>סעיף ג'</u>

<u>פעולות link:</u> יתבצעו m-1 פעולות לינק.

הסבר: בתחילה הכנסנו m+1 איברים לעץ, ואז מחקנו את המינימום. נותרו m איברים. m הוא חזקה שלמה של 2. לכן, בפעולת ה-consolidate לאחר מחיקת המינימום, נוצר עץ בינומי מלא יחיד, עם m איברים. בעץ עם m צמתים יש m-1 קשתות.

כיוון שהתחלנו מ-m צמתים שכל אחד מהם הוא עץ בפני עצמו, כל קשת בעץ החדש שנוצר לנו, נוצרה על ידי פעולת link. לכן, אם יש לנו m-1 קשתות, עשינו m-1 פעולות link.

<u>פעולות cut:</u> log(m) פעולות cut.

הסבר: כל אחד מהצמתים שעליהם מבוצע decreaseKey הוא בן של הורה אחר. ולכן כל פעם עושים cut יחיד. אנחנו אף פעם לא עושים cut ליותר מילד אחד של אותו הורה.

יש בסה״כ log(m) פעולות decreaseKey ולכן זה יהיה מספר פעולות ה-cut.

נוכיח כי לכל האיברים שנבחרו הורים שונים:

למה: הוכחה באינדוקציה שכל הבנים של השורש, המפתחות שלהם הם חזקות שלמות של 2:

מקרה בסיס של עץ בעל שורש בודד שהוא 0 זה טריוויאלי (אין בנים)

כעת נניח כי עבור k-1 אם נכניס את האיברים $2^{k-1}-1$,...,-1,0 לערימה ונמחק מינימום (-1) נקבל עץ בינומי מדרגה k-1 שכל בניו הינם החזקות של 2 מ-0 עד 2^{k-2}.

כעת נכניס את האיברים $2^{k}-1$,...,-1,0 לערימה ונמחק מינימום (-1) נקבל עץ בינומי מדרגה k שהינו שני עצים בינומים מדרגה k-1 ששורש אחד נשען על השני. מכיוון שהכנסנו איברים לפי הסדר נקבל עץ ראשון שיכיל את האיברים 0 עד $2^{k-1}-1$ ועץ שני שיכיל את האיברים 2^{k-1} עד $2^{k}-1$.

מכיוון ש-0 קטן מכל איבר בעץ השני יהיה השורש של העץ בדרגה k. ויחובר אליו בתור בן השורש של העץ השני שהינו האיבר המינימלי בו : 2^{k-1}.

כעת נבחיין כי הבנים של שורש העץ 0 הינם השורשים שלו בעץ מדרגה k-1 שהיתה לפניכן (מהנחת אינדוקציה $1,2,\ldots,2^{k-2}$) ו- 2^{k-1}. ולבסוף קיבלנו כי בניו הינם $1,2,\ldots,2^{k-1}$. והוכחנו את הלמה.

כעת נבחין כי אם ננסה לרדת לאורך הבנים המקסימלים מהשורש נקבל :

מהלמה הבן המקסימלי של שורש עץ בינומים המכיל איברים $0,1,\ldots,2^{\mathrm{k}}-1$ הינו 2^{k-1} והוא עצמו שורש של עץ המכיל איברים 2^{k-1} עד $2^{k}-1$. ולכן ניתן להקבילו לעץ המכיל איברים $0,1,\ldots,2^{\mathrm{k}-1}-1$ ומכאן בנו המקסימלי יהיה $2^{k-1}+2^{k-2}$. באופן אינדוקטיבי ניתן לראות כי האיבר ה-t כאשר נרד דרך הבנים המקסימלים יהיה :

$$2^{k-1}+\cdots+2^{k-t}=2^{k-t}(1+2+\cdots+2^{t-1})=2^{k-t}\cdot(2^{t}-1)=2^{k}-2^{k-t}$$

ובנו בעל הערך המימלי של האיבר ה-t הינו $2^{k}-2^{k-t}+1$ ולכן לאיברים $\{2^{k}-2^{k-t}+1 \mid 1\le t\le k\}$ יש הורים שונים.

<u>הפוטנציאל:</u>

מספר העצים : log(m)+1.

הסבר : לפני פעולות ה-decreseKey היה לנו עץ יחיד. עשינו log(m) פעולות decreaeKey, וכל אחת מהן יצרה עץ יחיד נוסף. לכן לאחר כל הפעולות הללו יש לנו log(m)+1 עצים.

מספר הצמתים המסומנים : log(m)-1.

הסבר : עשינו log(m) פעולות cut, כלומר יצרנו log(m) עצים חדשים. בכל יצירה של עץ חדש, ניתקנו צומת מאביו, וסימנו את האבא כ-marked. עשינו זאת לכל צומת מלבד הצומת שהוא הבן של השורש, כי את השורש לעולם לא מסמנים כ-marked. לכן מספר הצמתים המסומנים היא log(m)-1.

הפוטנציאל :

$$\begin{aligned}\phi&=(total\ trees)+2(total\ marks)=\log(m)+1+2(\log(m)-1)\\&=\log(m)+1+2\log(m)-2=3\log(m)-1\end{aligned}$$

<u>סעיף ד' :</u>

<u>פעולות link:</u> יתבצעו m-1 פעולות לינק.

הסבר : מאותו הנימוק של סעיפים קודמים.

<u>פעולות cut:</u> 0.

הסבר : אנחנו מבצעים decreseKey להורה, לאחר מכן לילד שלו, לאחר מכן לילד שלו וכך הלאה (מהלמה והסבר לאחר מכן) . בכל פעם ה-decreaseKey מוריד את אותו המספר, ולכן המבנה של הערימה (ובפרט של העץ) נשמר, ואין צורך לעשות cut כי לכל צמת שערכו מורד להורה שלה הורד אותו ערך ולכן אינה קטנה ממנו.

<u>הפוטנציאל:</u>

מספר העצים : לא עשינו שום cut אז מספר העצים נשאר 1.

מספר הצמתים המסומנים : 0, כי לא עשינו פעולות cut.

$$\phi=(total\ trees)+2(total\ marks)=1+0=1$$

סעיף ה'

פעולות link: 0. פעולות links מתבצעות רק בזמן שעושים consolidate. עושים consolidate רק כשקוראים לפונקציה deleteMin. ולכן אם לא קראנו ל-deleteMin, לא תתבצע אף פעולת link.

פעולות cut: 0. כיון שלא בוצעה אף פעולת link, אין אף קשת בין צומת הורה לצומת ילד, ולכן לא תהיה אף פעולת cut.

הפוטנציאל:

מספר העצים : m+1, כי הכנסנו m+1 איברים שונים, לא מחקנו אף איבר, ולא ביצענו שום פעולת link.

מספר הצמתים המסומנים : 0, כי לא עשינו פעולות cut.

$$\phi = (total\ trees) + 2(total\ marks) = m + 1 + 0 = m + 1$$

סעיף ו'

פעולות link: יתבצעו m-1 פעולות לינק.

הסבר : מאותו הנימוק של סעיפים קודמים.

פעולות cut: $2\log(m) - 1$.

חוץ מהפעולה שהתווספה, עשינו log(m) פעולות cut. מאותו הנימוק של סעיף ג'.

הפעולה הנוספת, הוסיפה לנו log(m)-1 פעולות cut. הסבר : בגלל ש-log(m) פעולות ה-decreaseKey יצרו לנו בעץ שרשרת של צמתים שהם מסומנים, מהשורש ועד לצומת שהמפתח שלו m-2. כעת, נקטין את המפתח לערך 3- , שקטן מהמפתח שלו מן הסתם, כי כל המפתחות שנשארו בעץ הם מספרים טבעיים. לכן, צריך לנתק אותו מההורה שלו. כאמור כל הצמתים בדרך לשורש מסומנים, לכן נצטרך לנתק את כולם. כלומר בסה"כ log(m)-1 פעולות cut.

העלות היקרה ביותר של פעולת decreaseKey = מספר החיתוכים הגדול ביותר: זה יקרה ב-decreseKey האחרון, כלומר הפעולה שנוספה. היא עולה log(m)-1 פעולות cut, כפי שהוסבר לעיל.

הפוטנציאל:

מספר העצים : 2log(m).

לפני הוספת הפעולה היו לנו log(m)+1 עצים (סעיף ג'). הפעולה מוסיפה לנו log(m)-1 פעולות cut, כל פעולה כזו מוסיפה עץ. אז מספר העצים הוא : log(m)+1 + (log(m) – 1) = 2log(m).

מספר הצמתים המסומנים : 0. בפעולה שהוספנו, באמצעות cascading cuts, ניתקנו את כל הצמתים המסומנים והפכנו אותם לשורשים, ולכן הם הפכו ללא-מסומנים.

$$\phi = (total\ trees) + 2(total\ marks) = 2\log(m) + 0 = 2\log(m)$$

סיכום סעיפים ג'-ו'

case	totalLinks	totalCuts	Potential	decresaseKey max cost
(c) original	$m-1$	$\log(m)$	$3\log(m)-1$	-
(d) decKey(m-2^i)	$m-1$	0	1	-
(e) remove line #2	0	0	$m+1$	-
(f) added line #4	$m-1$	$2\log(m)-1$	$2\log(m)$	$\log(m)-1$

שאלה 2

סעיף א'

i	m	Run-time(ms)	totalLinks	totalCuts	Potential
6	728	4.5056	723	0	6
8	6560	4.0264	6555	0	6
10	59048	17.3731	59040	0	9
12	531431	100.8808	531431	0	10
14	4782968	2252.7128	4782955	0	14

סעיף ב'

זמן הריצה האסימפטוטי: O(mlog(m)).

הסבר:

1. ביצענו m+1 פעולות הכנסה, שעולה כל אחת O(1), סה"כ O(m).
2. עשינו 3m/4 פעולות deleteMin, שכל אחת מהן עולה O(log(m)) ב-amortized (ניתן לחשב כך כי ביצענו רצף של פעולות deleteMin) סה"כ O(m(log(m)).

סעיף ג'

פעולות link: m-onesBin(m) , כאשר הפעולה onesBin(m) מוגדרת להיות - מספר האחדות בייצוג הבינארי של המספר m.

בפעולת ה-deleteMin הראשונה:

יש לנו m+1 צמתים שהוכנסו בסדר עולה (מהקטן לגדול). נמחק את האיבר המינימלי ואז נעשה consolidate ל-m האיברים שנותרו, שנמצאים במבנה של רשימה מקושרת.

נבחין כי על פי ההוראות במטלה, האיברים אשר מוכנסים אחרונים לערימה, מוכנסים להיות ראשונים מצד שמאל. האיברים הוכנסו בסדר עולה, ולכן האיבר הגדול ביותר הוא הראשון, וככל שנלך ימינה תמיד המפתחות יהיו קטנים יותר. האיבר הקטן ביותר הוא האיבר האחרון.

לכן, ה-consolidate מבוצע על האיברים בסדר יורד. במהלך ה-consolidate, בכל רגע במערך העזר שלנו, האיברים מסודרים בסדר עולה משמאל לימין. איברים שנמצאים משמאל, הם איברים שהכנסנו מאוחר יותר למערך. (והרי הכנסנו את האיברים בסדר יורד).

לאחר ה-consolidate, האיברים מסודרים לפי סדר עולה של המפתחות. כלומר, העץ השמאלי ביותר הוא בעל המפתחות הקטנים ביותר. לכל עץ, ערכי כל המפתחות בעץ, קטנים מערכי כל המפתחות של כל העצים שנמצאים מימינו. עבור כל הורה, גם הבנים שלו מסודרים בסדר עולה.

מספר הלינקים שבוצעו בפעולה זו:

בעצם בפעולה הזו לקחנו m איברים וחיברנו אותם לערימה בינומית (בשלב הזה זו בדיוק ערימה בינומית על פי ההגדרה).
אם היינו מחברים את m האיברים לעץ, אז היינו מקבלים עץ עם m-1 קשתות. כל קשת נוצרת על ידי link ולכן היינו אומרים שנעשו m-1 פעולות link.
אבל, לא חיברנו את כל האיברים לעץ, אלא לערימה בינומית (כי m הוא לא חזקה שלמה של 2). הקשתות ה"חסרות" לנו לעץ, הן הקשתות שבין השורשים של העצים בערימה. מספר הקשתות האלה הוא מספר השורשים בעץ פחות אחד. מספר השורשים בעץ הוא מספר האחדות בייצוג הבינארי של m. כלומר onesBin(m)-1.
אז בסה"כ נקבל שיש m-1 לינקים, פחות הקשתות החסרות לנו, שמספרן onesBin(m)-1. כלומר:

m-1-(onesBin(m)-1) = m-onesBin(m).

(הערה: הנוסחה הזאת נכונה גם לשאלה 1. שם m היה חזקה שלמה של 2, ולכן onesBin(m)=1. ולכן לאחר פעולת deleteMin אחת היינו מקבלים שמספר הלינקים הוא m-1).

פעולות ה-deleteMin האחרות:

כעת, מההסבר לעיל, צומת המינימום הוא תמיד שורש עם הדרגה הנמוכה ביותר בערימה. ולכן בכל פעם שנמחק אותו, או שאין לו בנים, או שיש לו בנים שהם בדרגה נמוכה מהם ונוסיף אותם לערימה.
כיון שהוא עם הדרגה הנמוכה ביותר, אין שורשים עם דרגה נמוכה ממנו, אז בפרט הבנים שלו, שהם עם דרגה נמוכה ממנו, הם היחידים עם דרגה זו בערימה. מבין הבנים של הצומת שהסרנו, הבן מהדרגה הנמוכה ביותר הוא בעל המפתח הנמוך ביותר, וזאת מההסבר לעיל. ולכן, גם המינימום הבא, הוא השורש של העץ בעל הדרגה הנמוכה ביותר.
לכן בכל פעולת deleteMin, המבנה של הערימה הבינומית יישמר ואז במהלך ה- consolidate **לא יבוצעו פעולות link נוספות.**

לסיכום, מספר פעולות ה-link הוא מספר פעולות הlink שבוצעו ב-deleteMin הראשון:

.m-onesBin(m)

<u>פעולות cut:</u> 0. פעולות cut מתבצעות כאשר אנחנו עושים decrearKey בלבד (או עושים delete, לא של המינימום, שקוראת ל-decreaseKey). כיוון שכאן לא עשינו אף פעולת decreaseKey, לא נעשתה אף פעולת cut.

<u>הפוטנציאל:</u>

מספר העצים: $onesBin\left(\left\lceil\frac{m}{4}\right\rceil\right)$, כי כמות האיברים בעץ לאחר הורדת $\left\lfloor\frac{3m}{4}\right\rfloor$ הינה $\left\lceil\frac{m}{4}+1\right\rceil$. מכיוון שהפעולה האחרונה שביצענו הינה deleteMin אז הערימה הינה בעלת מבנה של ערימה בינומית ומספר העצים בה הינו $onesBin\left(\left\lceil\frac{m}{4}+1\right\rceil\right)$.

מספר הצמתים המסומנים: 0, כי לא ביצענו פעולות decreaseKey ולכן גם לא ביצענו פעולות cut.

$$\phi = (total\ trees) + 2(total\ marks) = onesBin\left(\left\lceil\frac{m}{4}+1\right\rceil\right) + 0 = onesBin\left(\left\lceil\frac{m}{4}+1\right\rceil\right)$$