מבנ"ת- תרגיל מעשי 2.0

# חלק א- תיעוד פונקציות

### דגשים:

* עקרון מימוש כללי:
  + הערמה ממומשת על ידי רשימה מקושרת דו כיוונית (לא מעגלית) שמקשרת בין שורשי העצים בה:
    - מצביע ראש head מצביע לשורש השמאלי ביותר ברשימה.
    - מצביע זנב tail מצביע לשורש הימני ביותר ברשימה.
  + כל עץ בערמה ממומש באופן הבא:
    - לכל צומת עם ילדים יש מצביע לילד השמאלי ביותר בלבד.
    - לכל צומת שהיא לא שורש של עץ יש מצביע להורה שלה. לשורשים המצביע להורה הוא null.
    - צמתים אחים (שבאותה רמה) מקושרים זה לזה ברשימה מקושרת דו כיוונית (לא מעגלית). הבן השמאלי ביותר הוא ראש הרשימה, המצביע שלו לאח שמאלי הוא null. באופן דומה הבן הימני ביותר הוא זנב הרשימה והמצביע שלו לאח ימני הוא null.
* על מנת שהתיעוד יהיה יותר קריא לא נציין כאשר מתבצעת קריאה בפונקציה ל- setters ו-getters. הסיבוכיות של כל ה-setters וה-getters שמומשו בקוד היא .

## מחלקת FibonacciHeap

### שדות המחלקה

המחלקה כוללת את השדות הבאים:

* head- מצביע לראש הרשימה המקושרת של שורשי העצים בערמה.
* tail- מצביע לזנב הרשימה המקושרת של שורשי העצים בערמה.
* min- מצביע לצומת שהוא האיבר המינימלי בערמה.
* size- שומר את גודל הרשימה.
* treeCount- שומר את כמות העצים בערמה.
* markedCount- שומר את כמות הצמתים המסומנים הנוכחית.
* linkedCount- שומר את כמות פעולות החיבור שבוצעו מתחילת ריצת התוכנית.
* cutsCount- שומר את כמות פעולות החיתוך שבוצעו מתחילת ריצת התוכנית.

### isEmpty()

**תיאור**- הפונקציה מחזירה ערך TRUE אם ורק אם הערימה ריקה.

**מימוש**- אם המצביע head לראש רשימת השורשים בערמה הוא null מחזירה TRUE, אחרת FALSE.

**סיבוכיות**-

### insert(int key)

**תיאור**- הפונקציה יוצרת צומת מסוג HeapNode שמכיל את המפתח 𝑖 ומכניסה אותו לערימה. פעולה זו מחזירה את הצומת שנוצר שמכיל את המפתח i.

**מימוש**- הפונקציה:

* יוצרת צומת מסוג HeapNode עם המפתח key.
* אם הערמה ריקה- מעדכנת את זנב רשימת שורשי העצים בערמה להצביע לצומת זו.
* **מכניסה את הצומת לראש הרשימה המקושרת של שורשי עצי הערמה:**
  + מגדירה את ראש הרשימה הנוכחי להיות next של הצומת החדשה.
  + אם ראש הרשימה לא Null מגדירה את prev של head להצביע לצומת החדשה.
  + מגדירה את head להצביע לצומת החדשה.
* **עדכון שדה שורש-** מעדכנת את שדה isSource של הצומת להיות TRUE
* **עדכון גודל הערמה-** מעדכנת את גודל הערמה ב + 1
* **עדכון כמות העצים בערמה-** מעדכנת את כמות העצים ב + 1
* **עדכון מינימום-** משווה בין i לבין המפתח ששמור בשדה האיבר המינמלי ומעדכנת את מצביע המינימום לאיבר החדש במידה והוא קטן מהנוכחי.
* מחזירה את האיבר החדש שיצרה.

**סיבוכיות-** כל הפעולות המתבצעות לוקחות זמן קבוע.

### deleteMin()

**תיאור**- מחיקת הצומת שהמפתח שלו מינימלי מבין המפתחות שבערימה (אין צורך להחזיר אותה).

**מימוש**- במידה וקיים איבר מינימלי בערמה, הפונקציה:

* שומרת מצביע לבן השמאלי ביותר של צומת המינימום בשם currChild.
* **אם קיים לפחות בן אחד לצומת המינימום** – מחברת את הרשימה המקושרת של הילדים לרשימת השורשים. בפועל, הופכת את הילדים של הצומת המינימלית להיות שורשי עצים בערמה:
  + מעדכנת את ה-prev של הבן השמאלי להיות האח השמאלי של הצומת המינימלית.
  + אם ה-prev הוא צומת – מעדכנת את ה-next שלה להצביע לבן השמאלי של הצומת המינימלית.
  + אם הצומת המינימלית היא גם ראש הרשימה המקושרת של השורשים – מעדכנת את הראש להצביע לבן השמאלי שלה.
  + מעדכנת את שדות הילדים של הצומת המינימלית כך שיגדירו אותם כשורשים:
    - את שדה ההורה משנה ל-null.
    - מדליקה את דגל השורש isRoot.
    - אם הבן הנוכחי היה מסומן – מוחקת את הסימון ומעדכנת את ספירת הצמתים המסומנים בעץ.
  + אם הצומת המינימלית היא גם זנב הרשימה המקושרת של השורשים – מעדכנת את הזנב להצביע לבן הימני שלה.
  + מעדכנת את ה-next של הבן הימני להצביע לאח השמאלי של הצומת המינימלית.
  + אם ה-next הוא צומת – מעדכנת את ה-prev שלה להצביע לבן הימני של הצומת המינימלית.
* **אם אין לצומת המינימום ילדים**:
  + אם יש לצומת המינימלית אח שמאלי (בפועל זה אומר שהיא לא ראש הרשימה המקושרת של שורשי העצים בערמה)- מעדכנת את next של האח השמאלי להצביע לאח הימני של הצומת המינימלית.

אחרת – מעדכנת את ראש הרשימה להצביע לאח הימני.

* + אם יש לצומת המינימלית אח ימני (בפועל זה אומר שהיא לא זנב הרשימה המקושרת של שורשי העצים בערמה)- מעדכנת את prev של האח הימני להצביע לאח השמאלי.

אחרת – מעדכנת את זנב הרשימה להצביע לאח השמאלי.

* **מנתקת את צומת המינימום מהערמה-** מאפסת את שדות המצביעים של הצומת המינימלית ל-null.
* קוראת ל- **consolidate**() שמבצעת את תהליך הקונסולידציה על הערמה. הפעולה לוקחת במקרה הגרוע .
* **עדכון גודל הערמה-** מעדכנת את גודל הערמה במינוס 1.
* **מעדכנת את המינימום החדש בערמה ואת כמות העצים החדשה:**
  + מאפסת את כמות העצים בערמה.
  + יוצרת מצביע למינימום הנוכחי שבהתחלה מצביע לראש הרשימה המקושרת של שורשי העצים.
  + יוצרת מצביע לראש ועוברת על שורשי העצים אחד אחד:
    - **עדכון מינימום**- משווה את המפתח במצביע המינימום הנוכחי למפתח הצומת הנוכחי ואם הוא גדול יותר מעדכנת את מצביע המינימום הנוכחי להצביע לצומת הנוכחית.
    - **עדכון כמות העצים בערמה-** מעדכנת את כמות העצים ב + 1 בכל עץ שעוברים אליו.
    - מקדמת את המצביע לשורש הנוכחי לאח הימני שלו.

מתבצעות לכל היותר איטרציות כאלה ככמות העצים בערמה לאחר קונסולידציה.

* + מעדכנת את מצביע המינימום להצביע למינימום הנוכחי. (נשים לב שאם הערמה כעת ריקה – הראש שלה null ולכן גם המינימום יתעדכן ל-null).

**סיבוכיות-** במקרה הגרוע שנובע מפעולת הקונסולידציה במקרה שכל השורשים בערמה מדרגה 0 (n שורשים).

### consolidate()

**תיאור-** הפונקציה עושה קונסולידציה, מאחדת את שורשי העץ לכמות העצים המינימלית.

**מימוש**- הפונקציה:

* יוצרת מערך buckets בגודל .
* ***מבצעת את אלגוריתם successive linking****:*
  + *מגדירה מצביע לשורש הנוכחי בערמה שמתחיל מראש הרשימה המקושרת של השורשים.*
  + *עוברת על הרשימה המקושרת של שורשי העצים בערמה. כל עוד השורש הנוכחי לא null:*
    - *שומרת מצביע לאח הבא של הצומת הנוכחית בשם nextTree.*
    - *מנתקת את הצומת מהאח השמאלי שלה – שדה prev מצביע ל-null.*
    - *בודקת מה הדרגה של השורש הנוכחי.*
    - *אם הדרגה היא i היא מסתכלת על המקום ה-i במערך. כל עוד המקום ה-i במערך לא null:*
      * *משווה בין מפתח השורש במצביע השורש הנוכחי לבין מפתח השורש שבמקום ה-i במערך. מעדכנת את הגדול מבניהם להיות הבן השמאלי של הקטן יותר.*
      * *מעדכנת את שדה isRoot בהתאם.*
      * *מעדכנת את מצביע ההורה והילד בהתאם.*
      * *מעדכנת את הדרגה של השורש + 1.*
      * *מעדכנת את המצביע הנוכחי להורה.*
      * *משנה את המיקום הקודם במערך ל-null.*
      * ***מעדכנת את כמות החיבורים-*** *מוסיפה + 1 את* linkedCount.
    - *הלולאה מסתיימת כאשר המקום ה-i במערך ריק (null) ואז מתבצעת השמה של העץ המחובר לתא ה-i במערך. סך הכל יתבצעו לכל היותר איטרציות.*
    - *המצביע הנוכחי מתעדכן להצביע ל-nextTree- השורש הבא ברשימה המקושרת של שורשי הערמה.*
  + *חוזר חלילה עד שעברנו על כל שורשי הערמה. סך הכל יתבצעו לכל היותר איטרציות במקרה שבו כל שורשי הערמה מדרגה 0 (צמתים בודדים).*
* ***בונה מחדש את הרשימה המקושרת של שורשי העצים בערמה מתוך המערך שהתקבל:***
  + *מגדירה מצביע לשורש הנוכחי שמתחיל מהמקום ה-0 במערך, ומצביע לצומת הבאה.*
  + *עוברת באיטרציה על כל המקומות במערך. בפעם הראשונה שנתקלה בתא שמכיל שורש – מעדכנת את ראש הרשימה המקושרת של שורשי הערמה להצביע אליו. לאחר מכן, בכל פעם שנתקלת בצומת נוספת שאינה null במערך –מעדכנת את הצומת הנוכחית והצומת הבאה להיות אחים ומתקדמת עד שעברה על כל המערך.*
  + *מעדכנת את הזנב של הרשימה המקושרת להצביע לצומת האחרונה שחוברה לרשימה המקושרת של השורשים.*

*לכל היותר יתבצעו בשלב זה איטרציות ככמות השורשים בעץ.*

***סיבוכיות-***  *במקרה הגרוע שנובע מהמקרה שבו כל השורשים בעץ הם צמתים מדרגה 0 (n שורשים).*

### findMin()

**תיאור-** החזרת הצומת (מטיפוס HeapNode ) שהמפתח שלו מינימלי מבין המפתחות שבערמה. מערימה ריקה יוחזר null.

**מימוש-** החזרת מצביע המינימום. אם הערמה ריקה הערך השמור בשדה הוא null.

**סיבוכיות-**

### getHead()

**תיאור-** הפונקציה מחזירה את האיבר הראשון ברשימה המקושרת של עצי הערמה.

**מימוש-** הפונקציה מחזירה את שדה head.

**סיבוכיות-**

### getTail()

**תיאור-** הפונקציה מחזירה את האיבר האחרון ברשימה המקושרת של עצי הערמה.

**מימוש-** הפונקציה מחזירה את שדה tail.

**סיבוכיות-**

### getTreeCount()

**תיאור-** הפונקציה מחזירה את מספר העצים בערמה.

**מימוש-** הפונקציה מחזירה את שדה treeCount.

**סיבוכיות-**

### getMarkedCount()

**תיאור-** הפונקציה מחזירה את מספר האיברים המסומנים בערמה.

**מימוש-** הפונקציה מחזירה את שדה markedCount.

**סיבוכיות-**

### meld(FibonacciHeap heap2)

**תיאור-** מיזוג ערימה נוספת heap2 עם הערימה הנוכחית.

**מימוש-** הפונקציה:

* אם הערמה הנוכחית ריקה- מעדכנת את ראש הרשימה המקושרת של השורשים בערמה להצביע לראש הרשימה המקושרת של השורשים של heap2 ואת min להצביע לאיבר המינימום של heap2.
* אחרת- מחברת את הזנב של הרשימה המקושרת של שורשי הערמה הנוכחית לראש הרשימה המקושרת של שורשי הערמה השנייה.
* אם heap2 לא ריקה:
  + מעדכנת את prev של ראש הרשימה המקושרת של heap2 להצביע לזנב של הרשימה המקושרת הנוכחית.
  + **עדכון מינימום-** משווה את המינימום של הערמה השנייה לערמה הנוכחית ומעדכנת את מצביע המינימום בהתאם.
* **עדכון זנב-** מעדכנת את הזנב של הרשימה המקושרת הנוכחית להיות הזנב של heap2.
* **עדכון גודל הערמה-** מעדכנת את גודל הערמה ב + גודל heap2.
* **עדכון כמות העצים בערמה-** מעדכנת את כמות העצים של הערמה ב + כמות העצים של heap2.
* **עדכון כמות המסומנים**- מעדכנת את כמות המסומנים של הערמה ב + כמות המסומנים ב-heap2.

**סיבוכיות**- כל הפעולות מבצעות בזמן קבוע.

### size()

**תיאור-** הפונקציה מחזירה את מספר האיברים בערמה.

**מימוש-** הפונקציה מחזירה את הערך השמור בשדה size של הערימה.

**סיבוכיות-**

### countersRep()

**תיאור-** הפונקציה מחזירה מערך מונים כך שבאינדקס 𝑖 מופיע מספר העצים בערימה שהסדר (rank) שלהם הוא i. כלומר, היא מחזירה מערך של integers כך שלכל אינדקס 𝑖 בין 0 עד הדרגה המקסימלית של עץ שקיימת בערימה, הערך שמוחזר במערך הוא מספר העצים שקיימים בערימה מסדר i. עבור ערימה ריקה יוחזר מערך ריק.

**מימוש-** הפונקציה:

* בודקת אם הערמה ריקה- אם כן מחזירה מערך ריק.
* בודקת מה הדרגה הכי גדולה במערך- מתחילה מהערך 0. עוברת על שורשי העצים ומעדכנת את הערך בהתאם לאם הוא גדול יותר מהדרגה המקסימלית הנוכחית.
* יוצרת מערך בגודל הדרגה המקסימלית+1 של int שמאותחלים כולם ב-0. כל אחד מהם יהיה counter לכמות העצים בדרגה ה-i.
* עוברת על שורשי העצים ומעדכנת +1 במקום ה-i כאשר i זו הדרגה של העץ הנוכחי. האיטרציה מתבצעת ככמות העצים בערמה.

**סיבוכיות**- במקרה הגרוע שבו בערמה יש n שורשים.

### delete(HeapNode x)

**תיאור-** מחיקת הצומת x מהערימה.

**מימוש-** אם x זו צומת אמיתית, הפונקציה:

* קוראת ל-decreaseKey על x עם d ששווה למפתח של המינימום + המפתח הנוכחי + 10 על מנת ש-x יהפוך לאיבר המינימלי בערמה. הפעולה מתבצעת ב-.
* קוראת ל-deleteMin – הפונקציה תמחק את x שכרגע הוא המפתח המינימלי, ותבצע קונסולידציה לעץ. הפעולה מתבצעת ב- במקרה הגרוע.

**סיבוכיות-**  שנובעת מהמקרה הגרוע של deleteMin.

### decreaseKey(HeapNode x, int d)

**תיאור-** ערכו של המפתח של הצומת x יופחת בערך 𝑑≥0

**מימוש-** אם x זו צומת אמיתית, הפונקציה:

* מפחיתה את המפתח של x ב-d.
* אם x הוא לא שורש - משווה את המפתח של x למפתח של ההורה שלו. אם המפתח של x קטן מהמפתח של ההורה שלו- קוראת לפונקציה הרקורסיבית cascadingCut על x וההורה שלו.
* אם x הוא שורש והמפתח שלו כעת קטן מהמפתח של הצומת המינימלית- מעדכנת את מצביע המינימום להצביע ל-x ומסיימת.

**סיבוכיות-**

### cascadingCut(HeapNode x, HeapNode y)

**תיאור-** הפונקציה מבצעת את פעולת cascading-cut.

**מימוש-** פונקציה רקורסיבית. הפונקציה:

* קוראת ל- cut(x,y) שרצה ב- על x וההורה שלו y.
* אם y הוא לא שורש
  + אם הסימון של y הוא false - מעדכנת את הסימון ל-true ומוסיפה 1 ל- markedCount
  + אחרת- קוראת רקורסיבית לעצמה על y וההורה של y

**סיבוכיות-** נובע ממספר הקריאות הרקורסיביות המקסימלי שמתרחש כאשר x הוא העלה העמוק ביותר בעץ וכל שושלת ההורים שלו מסומנים. מספר הקריאות לפונקציה יהיה לכן .

### cut(HeapNode x, HeapNode y)

**תיאור**- הפונקציה חותכת את הצומת x מההורה שלה y.

**מימוש-** הפונקציה:

* מוסיפה 1 ל- cutsCount.
* **מעדכנת את x להיות שורש**:
  + מוסיפה 1 ל- treeCount.
  + מאפסת את הסימון של x ומחסירה 1 מ- markedCount.
  + מדליקה את דגל השורש של x.
  + ומעדכנת את מצביע האב של x ל-null.
  + משווה את המפתח של x למפתח של איבר המינימום ואם הוא יותר קטן מעדכנת את המינימום להצביע ל-x.
* **מעדכנים את הילדים של y**:
  + אם היה x הבן היחיד של y – מעדכנת את מצביע הילד להצביע ל-null.
  + אחרת-
    - אם מצביע child של y הצביע על x- מעדכנת אותו להצביע לאח הימני של x.
    - אחרת- מחברת בין האח השמאלי לאח הימני של x ולהפך אם קיים אח ימני.
* מעדכנת את הדרגה של y להיות פחות 1 אחרי ש-x כבר לא הבן שלו
* מוסיפה את x לראש רשימת שורשי העצים בערמה:
  + מעדכנת את next של x להיות ראש הרשימה המקושרת של שורשי העצים בערמה, ואת prev של ראש הרשימה הנוכחי להצביע ל-x.
  + מעדכנת את האח השמאלי prev של x להיות null.
* מעדכנת את מצביע ראש הרשימה המקושרת של שורשי הערמה להצביע ל-x.

**סיבוכיות-**

### potential()

**תיאור-** הפונקציה מחזירה את ערך הפוטנציאל הנוכחי של הערימה. הפוטנציאל, הינו Potential = #trees + 2\*#marked

**מימוש-** הפונקציה מחשבת את הפוטנציאל באמצעות שדה כמות העצים ושדה מספר הצמתים המסומנים ומחזירה את הערך.

***סיבוכיות-***

### totalLinks()

**תיאור-** פונקציה סטטית זו מחזירה את מספר כל פעולות החיבור שבוצעו מתחילת ריצת התוכנית. פעולת חיבור היא הפעולה שמקבלת שני עצים מאותו סדר ומחברת אותם.

**מימוש-** הפונקציה מחזירה את הערך השמור בשדה הסטטי linkedCount.

**סיבוכיות-**

### totalCuts()

**תיאור-** פונקציה סטטית זו מחזירה את מספר כל פעולות החיתוך שבוצעו מתחילת ריצת התוכנית. פעולת חיתוך מתרחשת עקב decreaseKey כאשר מנתקים תת -עץ מאביו.

**מימוש**- הפונקציה מחזירה את הערך השמור בשדה הסטטי cutsCount.

**סיבוכיות**-

### kMin(FibonacciHeap H, int k)

**תיאור-** פונקציה סטטית זו 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.

**מימוש-** הפונקציה:

* אם k=0 או שהערמה ריקה- מחזירה מערך ריק.
* אחרת- יוצרת ערמת עזר חדשה Hk שתכיל את האיברים החשודים להיות האיבר המינימלי הבא.
* יוצרת מערך בגודל k של HeapNodes.
* מחזיקה את המצביעים הבאים:
  + currMin- מצביע לצומת עם האיבר המינימלי הבא ב-H. זה האיבר הבא שצריך להכניס למערך k המינימליים.
  + minChild- מצביע לילד של currMin.
  + temp- מצביע לאיבר האחרון שהוכנס לערמת העזר.
* מוסיפה את מפתח המינימום של H לערמת העזר, ומעדכנת את שדה kMin של הצומת החדשה שהוסיפה להצביע לצומת המקבילה (עם אותו המפתח) ב-H.
* מבצעת באיטרציה k פעמים את הפעולות הבאות עד שהמערך מלא:
  + מעדכנת את מצביע currMin להצביע לשדה kMin של צומת המינימום הנוכחי בערמת העזר.
  + מוסיפה את המפתח של currMin למקום הבא במערך.
  + מפעילה על Hk את deleteMin. הערמה תהפוך לערמה בינומית תקינה באמצעות קונסולידציה.
  + מגדירה את minChild להצביע לילד של currMin.
  + מוסיפה את הילדים של currMin שכרגע נוסף למערך ל-Hk ומעדכנת את שדה kMin שלהם להצביע לצומת המקבילה בערמה המקורית. נניח ש-H זו ערמה בינומית תקינה ואז כמות הילדים של הצומת תהיה לכל היותר הדרגה של H מהתכונה שהדרגות של הילדים של צומת x הן כל הדרגות מ-0 עד deg(x)-1.
* מחזירה את המערך.

**סיבוכיות**- .

כל הפעולות עד הלולאה לוקחת זמן קבוע. הלולאה מתבצעת פעמים עד שהמערך מלא. נשים לב שבכל איטרציה מתווספים לערמה לכל היותר deg(H)-1 שורשים מהתכונה שכמות הילדים של צומת חסומה בדרגה של הצומת, והדרגה של הצומת היא לכל היותר הדרגה של הערמה. לכן גם הקונסולידציה חסומה ב- שזו כמות העצים שיש בעץ לכל היותר בקריאה אליה. הקונסולידציה תעדכן בכל איטרציה את כמות העצים להיות המינימלית כך שבסיומה כמות העצים בערמה תהיה חסומה ב-, כך שבאיטרציה הבאה הקונסולידציה תתבצע על לכל היותר פעולות successive linking. סה"כ הפונקציה מבצעת בכל איטרציה את פעולת deleteMin שתיקח כאמור ותבצע לכל היותר הכנסות של איברים לערמת העזר ולכן כל איטרציה מבצעת פעולות. סה"כ נקבל סיבוכיות של .

## מחלקת HeapNode

### שדות המחלקה

המחלקה כוללת את השדות הבאים:

* info- הערך השמור בצומת
* key- מפתח הצומת
* rank- הדרגה של הצומת, מספר הבנים שלה
* mark- דגל לסימון הצומת ב-cascading cut
* child- מצביע לילד של הצומת
* next- מצביע לאח של הצומת
* prev- מצביע לאח הקודם של הצומת
* parent- מצביע להורה של הצומת
* isRoot- דגל לסימון אם הצומת היא שורש
* kMin- מצביע לצומת המקבילה בערמה שרוצים להחזיר את מערך k האיברים המינימליים שלה. הצומת המקבילה זו הצומת בעלת אותו המפתח. בשימוש בפונקציית kMin של הערמה.

### getKey()

**תיאור**- הפונקציה מחזירה את מפתח הצומת.

**מימוש**- הפונקציה מחזירה את שדה key של הצומת.

**סיבוכיות**-

### setKey(int k)

**תיאור**- הפונקציה מגדירה את מפתח הצומת להיות שווה ל-k.

**מימוש**- הפונקציה מעדכנת את שדה key להיות שווה ל-k.

**סיבוכיות**-

### getValue()

**תיאור**- הפונקציה מחזירה את ערך הצומת.

**מימוש**- הפונקציה מחזירה את שדה info של הצומת.

**סיבוכיות**-

### getRank()

**תיאור**- הפונקציה מחזירה את הדרגה של הצומת.

**מימוש**- הפונקציה מחזירה את שדה rank של הצומת.

**סיבוכיות**-

### setRank(int r)

**תיאור**- הפונקציה מגזירה את דרגת הצומת להיות שווה ל-r.

**מימוש**- הפונקציה מעדכנת את שדה rank של הצומת להיות שווה ל-r.

**סיבוכיות**-

### getMarked()

**תיאור**- הפונקציה מחזירה את הסימון הנוכחי של הצומת.

**מימוש**- הפונקציה מחזירה את שדה mark של הצומת.

**סיבוכיות**-

### setMark(boolean b)

**תיאור**- הפונקציה מגדירה את הסימון של הצומת להיות b.

**מימוש**- הפונקציה מעדכנת את שדה mark להיות שווה ל-b.

**סיבוכיות**-

### getChild()

**תיאור**- הפונקציה מחזירה את הילד של הצומת.

**מימוש**- הפונקציה מחזירה את הצומת השמורה במצביע child של הצומת.

**סיבוכיות**-

### setChild(HeapNode x)

**תיאור**- הפונקציה מגדירה את x להיות הילד של הצומת.

**מימוש**- הפונקציה מעדכנת את שדה child להצביע ל-x.

**סיבוכיות**-

### getNext()

**תיאור**- הפונקציה מחזירה את האח הבא של הצומת.

**מימוש**- הפונקציה מחזירה את שדה next של הצומת.

**סיבוכיות**-

### setNext(HeapNode x)

**תיאור**- הפונקציה מגדירה את x להיות אח של הצומת.

**מימוש**- הפונקציה מעדכנת את מצביע next להצביע ל-x.

**סיבוכיות**-

### hasNext()

**תיאור**- הפונקציה מחזירה true אם האיבר הבא שונה מ-null.

**מימוש**- הפונקציה מחזירה את תוצאת this.next != null.

**סיבוכיות**-

### getPrev()

**תיאור**- הפונקציה מחזירה את האח הקודם של הצומת.

**מימוש**- הפונקציה מחזירה את שדה prev של הצומת.

**סיבוכיות**-

### setPrev(HeapNode x)

**תיאור**- הפונקציה מגדירה את x להיות האח הקודם של הצומת.

**מימוש**- הפונקציה מעדכנת את שדה prev להצביע ל-x.

**סיבוכיות**-

### getParent()

**תיאור**- הפונקציה מחזירה את ההורה של הצומת.

**מימוש**- הפונקציה מחזירה את שדה parent של הצומת.

**סיבוכיות**-

### setParent(HeapNode p)

**תיאור**- הפונקציה מגדירה את ההורה של הצומת להיות p.

**מימוש**- הפונקציה מעדכנת את המצביע בשדה parent של הצומת להצביע ל- p.

**סיבוכיות**-

### getIsRoot()

**תיאור**- הפונקציה מחזירה האם הצומת היא שורש או לא.

**מימוש**- הפונקציה מחזירה את שדה isRoot של הצומת.

**סיבוכיות**-

### setIsRoot(boolean b)

**תיאור**- הפונקציה מגדירה את הצומת כשורש או שלא לפי b.

**מימוש**- הפונקציה מעדכנת את שדה isRoot להיות שווה ל-b.

**סיבוכיות**-

### getkMin()

**תיאור**- הפונקציה מחזירה את המצביע לצומת המקבילה בעלת אותו המפתח בערמה המקורית ממנה רוצים להחזיר מערך של k האברים המינימליים. בשימוש בפונקציית kMin של מחלקת הערמה.

**מימוש**- הפונקציה מחזירה את שדה kMin של הצומת.

**סיבוכיות**-

### setkMin(HeapNode node)

**תיאור**- הפונקציה מגדירה את המצביע kMin להצביע ל-node. node זו הצומת המקבילה בעלת אותו המפתח בערמה המקורית ממנה רוצים להחזיר מערך של k האברים המינימליים. בשימוש בפונקציית kMin של מחלקת הערמה.

**מימוש**- הפונקציה מעדכנת את המצביע בשדה kMin של הצומת להצביע ל- node.

**סיבוכיות**-

# חלק ב- ניסויי תיאורטי

## שאלה 1

### סעיף א'

ננתח את סיבוכיות זמן הריצה של סדרת הפעולות הנתונה. ראשית יש לנו פעולות הכנסה, כאשר עלות כל פעולה כזו היא . לאחר מכן אנו מבצעים פעולת מחיקת האיבר המינימלי. הפעולה עצמה נעשית בסיבוכיות אך היא גוררת אחריה successive linking של m האיברים הנותרים. לכן, העלות הכוללת של הפעולה שהיא בעצם מקרה ה-worst case שנלמד היא . בסופו של דבר אנו מבצעים פעולות decreaseKey. נתעכב על מבנה העץ ועל הצמתים המסוימים אשר אנו מבצעים להם הורדה בדרגה. אנו טוענים כי מתבצעת פעולת cut בודדה ולא פעולות cascading-cut בעת הקריאה להורדת מפתח על צמתים אלו.

לאחר שהכנסנו לעץ את כל הצמתים אנו מבצעים פעולת מחיקת מינימום אשר משאירה את הערימה עם בדיוק m צמתים, שזו חזקה של 2 בהתאם להגדרת התרגיל. הוצאת המינימום במקרה זה תהפוך את הערימה שלנו לערימה בינומית בהתאם לטענה שראינו בהרצאה. העץ מלא בשלב בו אנו מתחילים לבצע את פעולות הקטנת המפתח. פעולות אלו אכן שוברות את תנאי הערימה כיוון שכל המפתחות בעץ שווים/קטנים מ-m ולכן כן ניאלץ לבצע פעולות חיתוך. בהרצאה הגדרנו באופן רקורסיבי עץ בינומי מדרגה k כשורש של עץ בינומי מדרגה התלוי על שורש של עץ בינומי אחר מדרגה . בפועל אנו נעבור בלולאה על כל האיברים שהם שורש תת העץ הגדול ביותר של תת העץ הקודם לפי הצורה בה אנו בוחרים את הצמתים שנוריד להם מפתח, כך שבעצם פעולות הקטנת המפתח יצרו לנו תתי עצים בגודל של צומת אחד (כי הם העוקבים של השורש) שאותם נחתוך ונוסיף לערימה המקורית*. לכן בסה"כ נראה שכל פעולת הקטנת מפתח גוררת cut יחיד ולא cascading-cut ברצף של פעולות חיתוך בעלות של כל אחת.*

*בסך הכל סיבוכיות רצף הפעולות תהיה*

### סעיף ב'

הטבלה שהתקבלה:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Potential** | **totalCuts** | **totalLinks** | **Run-Time (ms)** | **m** |
| 29 | 10 | 1023 | 2 |  |
| 44 | 15 | 32767 | 11 |  |
| 59 | 20 | 1048575 | 73 |  |
| 74 | 25 | 33554431 | 14591 |  |

### סעיף ג'

פעולות הlink מתרחשות בשלה ה- successive linking כאשר האלגוריתם מחבר בכל פעם שני תתי עץ באותה הדרגה. בשלב ההכנסה, כל העצים שנחבר יהיו מדרגה 0 וכמות העצים השונים תהא שהוא מספר טבעי (2 בחזקת מספר טבעי), אפשר להסתכל על החתך הרוחבי של האלגוריתם שבפועל יחבר כל זוג מהצורה . בשלב הבא, לאחר מחיקת האיבר המינימלי האלגוריתם יחבר בין עצים בדרגה 1, כך שהשורשים שיתחברו הם וכך הלאה במהלך רמות. בסה"כ יהיו בכל שלב כזה חיבורים, כלומר:

וזה מספר פעולות ה-link שיתבצעו.

מספר פעולות ה-cut כפי שהסברנו בסעיף א' בעצם מתבצע פעמים.

**פוטנציאל**

### סעיף ד'

### סעיף ה'

## שאלה 2