**תיעוד + סיבוכיות זמן:**

**המחלקה FibonacciHeap :**

כללי:

לכל אובייקט במחלקה נתחזק את השדות הבאים:

* Size – מספר איברי הHeapNode במבנה הנתונים
* numTree- ספירת העצים תתוחזק ותישמר כשדה
* numMarked- בכל רגע נתון נדע את מספר הצמתים המסומנים
* Start- מצביע לעץ הראשון
* Tail- מצביע לעץ האחרון
* Min- HeapNode בעל מפתח מינימלי

לאורך כל הפונקציות שמשנות את מצב העץ נתחזק את כלל השדות הנ"ל.

בנוסף נחזיק את השדות הסטאטיים הבאים שיהווה מונים במהלך ריצת התכנית:

* Links- כמות החיבורים הכוללת שנעשתה
* Cuts- כמות הניתוקים הכולל

כמו כן עבור המחלקה **HeapNode**, לכל אובייקט נתחזק את השדות:

* Child
* Parent
* Prev, next
* Marked
* Origin- נועד בשביל kMin, לכל צומת שנבנה נחזיק מצביע לצומת ה"מקורי".
* Key
* rank

פונקציות:

isEmpty():

בודקת האם השדה this.start==null אם כן, תחזיר true, כלומר העץ ריק, אחרת false.

סיבוכיות: כל הבדיקות והפעולות בזמן קבוע ולכן O(1).

Insert:

3 פונקציות בעלות אותו השם עם חתימות שונות (העמסה):

1. הפונקציה מקבלת מפתח ויוצרת Node חדש מסוג HeapNode. לאחר מכן קוראת לinsert עם הצומת.
2. פונקציה שמשמשת לkMin- מקבלת מפתח וצומת מקורי ומעדכנת את שדה הorigin של הצומת החדשה לצומת המקורית, לאחר מכן תקרא לinsert עם הצומת החדשה.
3. **Insert ראשית**- מקבלת צומת. אם העץ ריק או שהמפתח קטן מהמינימום היא תעדכן את המינימום, ותקרא לadd at start. כמו כן, תעדכן את מספר הצמתים (size) ומספר העצים.

סיבוכיות: כל הבדיקות והפעולות בזמן קבוע ולכן O(1).

פונקציות עזר ל – Insert:

* Add at start

קוראת לaddat- עם הצומת ומעדכנת את השדה start לצומת.

* Add at tail

קוראת לaddat ומעדכנת את השדה tail לצומת.

* Addat

אם העץ ריק תקרא לinsertBefore רק עם הצומת (כך שinsert תחבר את הצומת לעצמה- רשימה מקושרת),ותעדכן את שדות הstart וtail. אחרת תקרא לinsertBefore עם הצומת והצומת ההתחלתית של העץ (this.start).

* insertBefore

הפונקצייה מקבלת 2 צמתים – הצומת שיש להכניס-newNode והצומת שלפניה נכניס- position.

אם זו אותה צומת- נחבר אותה לעצמה בעזרת setnext, setprev.

אחרת, נחבר את הצומת לprev של position ולposotion בקשר כפול.

סיבוכיות כוללת של פונק' העזר:

בהתאם להסברים, כלל הפעולות מתבצעות בזמן קבוע, ללא לולאות ואיטרציות שתלויות בקלט.

deleteMin:

אם העץ אינו ריק נפעל באופן הבא:

1. נקרא ל deleteMinFunc פונקציה זו מוחקת את המינימום ומכניסה את בניו לרשימת השורשים תוך התחשבות במקרה קצה.
2. נוריד את גודל העץ ב-1
3. נקרא לconsolidation הפונקציה מבצעת successive linking ומעדכנת את השדות המתאימים בהתאם

סיבוכיות:

deleteMinFunc:

פונקציה זו מוחקת את המינימום ומכניסה את בניו לרשימת השורשים תוך התחשבות במקרה קצה.

נחלק למקרים:

* מספר העצים הוא 1- אם אין ילדים- העץ יהיה ריק ולכן נעדכן את ההתחלה והסוף לnull. אחרת, נגדיר את הילד השמאלי להיות הstart והprev שלו לtail.
* אחרת, אם האיבר בstart נגדיר את start להיות הnext שלו, ובאותו אופן לגבי tail. כעת נבדוק לגבי ילדיו, אם אינם קיימים רק נחבר את הprev עם הnext שלו, אחרת, נקרא לputFamily עם הprev, next והילד השמאלי.

*סיבוכיות: O(1), כל הפעולות נעשות בזמן קבוע.*

*putFamily:*

*פונקציה שנועדה, בעת מחיקת השורש, לחבר את הילדים לprev ולnext של השורש.*

*הפונקציה מקבלת 3 HeapNode- prev,next,child ומעדכנת את החיבור של ה"משפחה" במקום הצומת שנמחק.*

*כמו כן, הפונקציה תעדכן את שדות הstart והprev במידת הצורך.*

*סיבוכיות: O(1), כל הפעולות נעשות בזמן קבוע.*

*Consolidation:*

*מטרתה לבצע successive linking ולמצוא את המינימום החדש.*

*הפונקציה בונה מערך של צמתים בגודל 2\*log(this.size). כפי שראינו בהרצאה, לכל הצמתים בעץ יש rank שאינו עולה על 1.4404 log2n, ולכן לכל עץ יהיה מקום ברשימה ע"פ הrank שלו.*

*כעת נעבור על הרשימה המקושרת החל מthis.start ועד הtail, נבצע לכל השורשים- makeUnmarked ונאתחל את הparent שלהם לnull.*

*לבסוף נקרא לlinkAndPut- פונקציה שתשים את הצמתים במערך לפי הrank, ובמידת הצורך- אם יש 2 צמתים באותה משבצת, תבצע להם link ותעבירם למשבצת הבאה.*

*כעת, לאחר שסיימנו ליצור את המערך, נקרא לfixListOfRoots עם המערך כך שנאתחל את העץ מחדש עם העצים החדשים שחיברנו.*

סיבוכיות:

*makeUnmarked:*

*מקבלת צומת ומעדכנת את השדה marked שלה לfalse. כמו כן, הפונקציה בודקת אם השדה היה true ואם כן מורידה את numMarked ב1.*

*סיבוכיות: O(1), כל הפעולות נעשות בזמן קבוע.*

*linkAndPut:*

*פונקציה רקורסיבית.*

*הפונקציה מקבלת את המערך basket שיצרנו בconsolidation וצומת חדש שיש להכניס למערך במיקום במערך השווה לrank של הצומת.*

*כעת, אם המקום מלא- היא תקרא לlink על הצומת במערך והצומת שלנו, תהפוך את המקום במערך לnull ותקרא לlinkandput עם הצומת לאחר חיבור העצים.*

*אם המקום ריק, פשוט תמקם את הצומת במערך במקום זה.*

*סיבוכיות: O(1), כל הפעולות נעשות בזמן קבוע.*

*Link:*

*חיבור 2 עצים עם שורשים בעלי אותו הrank.*

*תחילה הפונקציה תעדכן את שדה הlinks לפלוס אחד והnumTrees למינוס אחד.*

*לאחר מכן תגדיר את node1 להיות הצומת עם המפתח הקטן מבין השניים.*

*כעת, אם הrank גדול מאפס יש לחבר את node2 לילדים של node1, אחרת יש לחבר את node2 לעצמה ולכן נקרא לinsertBefore שתבצע זאת.*

*לבסוף נבצע את החיבורים בין node1 לnode2 כך שnode1 יהיה האבא של node2, נגדיל את הrank של node1 ב1 ונחזיר את שורש העץ החדש- node1.*

*סיבוכיות: O(1), כל הפעולות נעשות בזמן קבוע.*

*fixListOfRoots:*

*לאחר ששמנו את כל העצים במערך וביצענו את החיבורים הנדרשים, נותרנו עם לכל היותר עץ אחד מכל rank.*

*כעת, הפונקציה תעדכן את העץ כך שתעבור על מערך הbasket.*

*תחילה נאתחל את הערימה כך ששדות הstart וtail הם null.*

*כאשר ניתקל באיבר הראשון במערך שאינו null נעדכן את הstart להצביע על הצומת הזו.*

*ומכאן והלאה נחבר את כל הצמתים במערך (שאינם null), לפי הסדר ברשימה מקושרת דו כיוונית החל מהstart.*

*לבסוף נעדכן את tail להיות הצומת האחרון.*

*במהלך המעבר על הצמתים נחפש את הצומת עם המפתח המינימלי ונעדכן את המינימום להצביע אליו.*

סיבוכיות:

*findMin:*

*אם העץ ריק, תחזיר null אחרת תחזיר את this.min – שדה אשר מתוחזק בעת פעולות על הערימה.*

*סיבוכיות: O(1), כל הפעולות נעשות בזמן קבוע.*

*Meld:*

*מקבלת ערימה נוספת וממזגת עם הערימה הקיימת.*

*אם הערימה הקיימת ריקה, תעעדכן את שדות הstart והtail לשדות אלה בערימה החדשה.*

*אחרת, תקרא לputFamily עם הtail,start,heap2.getstart כך שנחבר לרשימה המקושרת של הערימה הנוכחית את החדשה בסופה.*

*נעדכן את הsize, numtree, nummarked לחיבור השדות של שתי הערימות.*

*סיבוכיות: O(1), כל הפעולות נעשות בזמן קבוע.*

*Size:*

*מחזירה את this.size – מספר הצמתים בערימה.*

*סיבוכיות: O(1), כל הפעולות נעשות בזמן קבוע.*

*Delete:*

*נגדיר את diff להיות ההפרש בין המפתח של הצומת למפתח של המינימום ונבצע decreaseKey לצומת עם 1+diff. כך, הצומת יהיה בעל המפתח המינימלי וכשנקרא לdeletemin נמחק אותו.*

*סיבוכיות:*

*decreaseKey:*

*תחילה הפונקציה תעדכן את המפתח של הצומת לערך החדש, ע"י setKey.*

*אם המפתח החדש קטן מהמינימום, נעדכן את המינימום אליו.*

*כמו כן, אם הצומת אינה שורש והמפתח החדש שלה קטן מאביה אז נקרא לcascadingCut.*

*סיבוכיות:*

*cascadingCut:*

*פונקציה רקורסיבית.*

*נבצע cut בין הצומת לאביו.*

*כעת, נבדוק האם האב מסומן. אם לא, נסמנו ונעלה את מונה הnumMarked.*

*אם כן, נקרא רקורסיבית לcascadingCut עם האב ואביו.*

*Cut:*

*ביצוע החיתוך עצמו בין הצומת לאב.*

*נעדכן את שדה הparent של הצומת לnull ונקרא לmakeUnmarked.*

*נוריד את הrank של האב של הצומת ב1.*

*כעת נחלק למקרים לפי מספר הילדים של האב:*

*אם זה הבן היחיד, נקבע את הילד של האב לnull.*

*אחרת אם הchild של האב זה הצומת נעדכן לnext שלו.*

*בנוסף, נחבר את האחים של הnode אחד לשני ברשימה המקושרת (נמחק את הnode משם).*

*נקרא לaddAtStart עם הצומת ונעלה את מספר העצים והcuts.*

*סיבוכיות: O(1), כל הפעולות נעשות בזמן קבוע.*

*Potential:*

*נחזיר את numTree+2\*numMarked.*

*שדות של הערימה אותם תחזקנו לאורך הפונקציות השונות.*

*סיבוכיות: O(1), כל הפעולות נעשות בזמן קבוע.*

*totalLinks:*

*פונקצייה סטטית שתחזיר את כלל החיבורים שבוצעו.*

*לשם כך אתחלנו שדה סטטי ובכל פעולת לינק אנו מעלים את ערכה ב1.*

*סיבוכיות: O(1), כל הפעולות נעשות בזמן קבוע.*

*totalCuts:*

*בדומה, כל פעולות החיתוך שהתבצעו.*

*אתחלנו שדה סטטי ובכל פעולת חיתוך נעלה את ערכה ב1.*

*סיבוכיות: O(1), כל הפעולות נעשות בזמן קבוע.*

*countersRep:*

*אם המבנה נתונים ריק נחזיר מערך ריק*

*אחרת נעבור פעמיים על השורשים בערימה, בפעם הראשונה נבדוק מה הדרגה המקסימלית של העץ בערמה. לאחר מכן נאתחל מערך מונים בגודל זה + 1. בפעם השנייה נעבור שוב על העצים בערמה כאשר בכל עץ נעלה באחד את במיקום של דרגתו את התא במערך המונים ב-1.*

*סיבוכיות: בכל קריאה לפונק' אנחנו עוברים בדיוק פעמיים על רשימת העצים בערימה ומס' העצים במקרה הגרוע הוא כמס' האיברים בערימה ולכן הסיבוכיות במקרה הגרוע היא O(n).*

*Kmin:*

*נאתחל מערך בגודל K – O(k)*

*לאחר מכן נאתחל ערימת עזר ונכניס אליה את המינימום של הערימה H עם מצביע לצומת המקורי בערימה H בעזרת הבנאי הפנימי שהגדרנו במחלקה. לאחר מכן נבצע K איטרציות באופן הבא:*

*בכל איטרציה נקרא לפונק' removeAndPut על ערימת העזר שלנו – פונקציה זאת בכל קריאה מכניסה את הבנים המקוריים של המינימום בערימת העזר לערימת העזר (הבנים מהערימה H), מוחקת את המינימום ומחזירה את המינימום. את המינימום שחזר נציב במערך במקום המתאים.*

*סיבוכיות:*

*~~טענה: מס' העצים בערימת העזר בכל תחילת איטרציה של removeAndPut הינו O(log(deg(H)) לכל היותר:~~*

*~~• בהנחיות התרגיל נכתב כי ניתן להניח שדרגת השורש היא הגבוהה ביותר בערימה H ולכן בכל איטרציה אנו מוסיפים לכל היותר deg(H) צמתים.~~*

*~~• נשים לב שאנו לא מבצעים פעולות decreaseKey \ deleteNode ולכן ערימת העזר מתנהגת כערימה בינומית עצלה.~~*

*~~כעת נוכיח את הטענה באינדוקציה על מס' האיטרציות:~~*

*~~בסיס:~~*

*~~האיטרציה מס' 1: בתחילת האיטרציה זו רק המינימום המקורי בערימה ולכן הטענה מתקיימת.~~*

*~~האיטרציה מס' 2: אנו מוסיפים את המינימום בערימה H ולאחר מכן מוסיפים את בניו, יש לו deg(H) בנים ולכן טרם פעולת ה- successive linkingיש בערימה deg(H) צמתים. ולכן לאחר פעולה זו יש לכל היותר O(log(deg(H)) עצים (כי אנו מקבלים ערימה בינומית תקינה).~~*

*~~צעד: נניח עבור i, נוכיח עבור i+1, מהנחת האינדוקציה נובע כי בתחילת האיטרציה ה-i יש לכל היותר בערימת העזר O(log(deg(H)) עצים. במהלך האיטרציה ה- i+1 נוסיף את בניו של המינימום הנוכחי שמספרם לכל היותר deg(H) ונמחק את המינימום לכן נקבל כי טרם פעולת ה- successive linking יש בערימה O(deg(H)+~~*

*removeAndPut:*

*פונקצייץ עזר ל-Kmin ולכן פועלת רק ואך על ערימות עזר שלכל צומת בהן יש מצביע לצומת המקורי.*

*בכל קריאה לפונקציה מתבצע הכנסה של הבנים המקוריים של המינימום בערימת העזר לערימת העזר (הבנים מהערימה H), מוחקת את המינימום ומחזירה את המינימום.*

*סיבוכיות:*

*ראה פירוט בפונק' Kmin.*