תיעוד למחלקת $BinomialHeap$:

תיאור:

המחלקת BinomialHeap ממשת ערימה בינומית

בנאים:

BinomialHeap():

מאתחל ערימה בינומית ריקה. זה לא לוקח שום פרמטרים ולא מאתחל שום משתני מופע.

BinomialHeap(int size, BinomialHeap.HeapNode last, BinomialHeap.HeapNode min):

זהו בנאי בעל פרמטרים של המחלקה BinomialHeap. הוא דורש שלושה פרמטרים:
size: מספר שלם המייצג את גודל הערימה.
last: מופע של המחלקה HeapNode, המייצג את הצומת האחרון בערימה.
min: מופע של מחלקת HeapNode, המייצג את הצומת המינימלי בערימה.
הבנאי מאתחל את משתני המופע size, last ו-min עם הערכים שסופקו.

תכונות:

size: מייצג את גודל הערימה

last: מייצג את השורש בעל הדרגה המקסימלית בערימה

min: מייצג את הצומת המינימלי בערימה

מתודות:

פונקציית size() : מחזירת את השדה size .

סיבוכיות הזמן: $O(1)$.

פונקציית empty(): מחזירה false אם $last$ הוא null ואחרת מחזירה true.

סיבוכיות הזמן: $O(1)$.

פונקציית recalculateSize() :

נתחיל מ-$last$ ונעבור על כל השורשים של העצים בערימה ונסכם את 2^{rank} של כל שורש, ומקבלים כמות האיברים בערימה.

סיבוכיות הזמן: כיוון שערימה בגודל n יש לכל היותר $\log n$ עצים, ולכן יש $\log n$ פעולת סכמה שכל אחת $O(1)$ ולכן בס"ה הפונקציה תעלה $O(\log n)$ זמן.

פונקציית link() : הפונקציה מקבלת שני צמתים ומחברת ביניהם.

משווים בין שני הערכים של הצמתים, וצומת בעל הערך היותר קטן נסים כבן של הצומת האחר בעזרת הפונקציה $setChild()$.

<u>פונקציות עזר:</u> setChild.

<u>סיבוכיות הזמן:</u> $O(1)$.

פונקציית insert(key, info) :

מקבלים מפתיח key ומידע $info$, מייצרים $item$ חדש עם הערך והמידע הנתון, ואז מייצרים צומת חדשה עם ה- $item$ הזה, כך שה- $next$ שלו הוא עצמו. מאתחלים ערימה בגודל 1, מכילה רק את הצומת שלנו ובסוף קוראים לפונקציה $meld$ עם העץ שלנו ועץ החדש.

<u>פונקציות עזר:</u> meld

<u>סיבוכיות הזמן:</u> אתחול צומת ועץ הוא $O(1)$ ועוד סיבוכיות הזמן של הפונקציה $meld$ שזה $O(\log n)$ ולכן מקבלים סיבוכיות הזמן של הפונקציה $O(\log n)$.

פונקציית findMin() : מחזירה את השדה min.

סיבוכיות הזמן: $O(1)$.

פונקציית SearchForMin() :

פונקציית עזר מקבלת עץ ומחפשת את האיברים בעל הערך המינימלי ומעדכנת את השדה min בהתאם.

נעבור על כל השורשים של תתי העצים בערימה, כיוון שאנו יודעים שהמינימום של העץ הוא השורש שלה אזי מספיק לבדוק רק את השורשים של העצים, נחפש בשורשים על השורש המינימלי ואזי הוא המינימום בערימה. נעדכן את השדה מינימום.

<u>סיבוכיות הזמן:</u> בעץ בעל n איברים, קיים לכל היותר $\log n$ עצים ולכן סיבוכיות הזמו היא $O(\log n)$

פונקציית deleteMin() :

עבור הערימה הבינומיאלית הנתונה, פונקציית $deleteMin$ מסירה הפריט המינימלי מהערימה. זה מתחיל בטיפול במקרים מיוחדים. לאחר מכן, הוא מסיר פיזית את הצומת המינימלי מהערימה ומתאים את מבנה הרשימה המקושרת המעגלי. אם לצומת המינימלי יש ילדים, העצים האלה נשלפים ומתמזגים לערימה בינומית חדשה. לבסוף, מתבצעת פעולת meld למיזוג הערימה החדשה עם הערימה הקיימת.

<u>פונקציות עזר:</u> meld

<u>סובביות זמן:</u> לפעולת $deleteMin$ יש מורכבות זמן של $O(\log n)$ באשר n הוא מספר הצמתים בערימה, מכיוון שהיא כוללת מיזוג וסידור מחדש של עצים בינומיים.

פונקציית Meld() :

עבור הערימה הבינומיאלית הנתונה, פונקציית $meld$ משלבת שני ערימות בינומיות לערמה אחת. זה מתחיל בטיפול במקרים מיוחדים שבהם אחת הערימות או שתיהן ריקות. לאחר מכן, הוא קובע את הצומת המינימלי החדש על ידי השוואת הצמתים המינימליים של שתי הערימות. לאחר מכן, הוא יוצר מערכים לאחסון הצמתים של כל דרגה בשתי הערימות. הוא מבצע תוספת בינארית של העצים הבינומיים במערכים, ממזג עצים באותה דרגה ומעביר את כל העצים שנותרו. העצים המתקבלים מקושרים זה לזה ליצירת ערימה בינומית חדשה. לבסוף, פונקציית $meld$ מעדכנת את הגודל והצומת האחרון של הערימה בהתבסס על העצים הממוזגים.

<u>פונקציות עזר:</u> empty, getMaxDegree, recalculateSize, searchForMin

<u>סובביות זמן:</u> מורכבות הזמן של פעולת $meld$ היא $O(\log n)$, כאשר n הוא המספר הכולל של צמתים בשתי הערימות, שכן היא כוללת מיזוג וסידור מחדש של העצים הבינומיים.

פונקציית delete(item) :

מקבלים item ומוחקים אותו הצומת שלו.

בעזרת הפונקציה $decreasekey$ מעדכנים את ה-key של ה-$item$ להיות הערך המינימלי בערימה על ידי עדכנו להיות -1, כיוון שאנו יודעים ששאר הערכים הם חיובים אזי בהכרח הערך -1 הוא המינימלי.

נקבל שהצומת שאנו רוצים למחוק היא המינימום בערימה ולכן מספיק לקרוא לפונקציה $deleteMin$ והיא מוחקת את הצומת פיזית.

<u>פונקציות עזר:</u> deleteMin() ,decreaseKey()

<u>סיבוכיות הזמן:</u> סיבכיות הזמן של הפונקציה $decreasekey() = O(\log n)$ וסיבוכיות הזמן של $deleteMin = O(\log n)$ ולכן סיבוכיות הזמן היא $O(\log n)$

פונקציית numTrees() :

קוראים על הפונקציה $recalculateSize$ ובודקים האם הגודל של העץ הוא 0, אם כן אזי הערימה ריקה ומחזירים 0.

אחרת הערימה אינה ריקה, עוברים על ידי לולאת $while$ הצומת $last$ עד הצומת $last$ לו כולל ונסכם מספר תתי עצים קיבלנו ומחזירים את הערך שמקבלים.

<u>פונקציות עזר:</u> recalculateSize()

<u>סיבוכיות הזמן:</u> $recalculatesize$ היא $O(\log n)$ והמעבר בלולאת $while$ הוא $O(\log n)$ ולכן סיבוכיות הזמן של הפונקציה הוא $O(\log n)$.

פונקציית getMaxDegree():

מחזירים את ה-$rank$ של הצומת $last$. כיוון שבשדה $last$ שומרים את הצומת בעל הדרגה המקסימלית.

<u>סיבוכיות הזמן:</u> $O(1)$.

פונקציית decreaseKey(item, diff) :

עבור הערימה הבינומיאלית הנתונה, פונקציית $decreaseKey$ מקטינה את ערך המפתח של פריט שצוין בערימה בהפרש נתון $diff$. תחילה הוא מעדכן את המפתח של הפריט על ידי הפחתת ההפרש. לאחר מכן, הוא מתקן את מבנה הערימה על ידי השוואת המפתח המעודכן למפתח האב שלו. אם המפתח של האב גדול יותר, הצמתים מוחלפים ע"י מיתודת $swapNodes$ כדי לשמור על מאפיין סדר הערימה. תהליך זה ממשיך באופן איטרטיבי עד שהמפתח של ההורה קטן יותר או שהצומת מגיע לשורש. בנוסף, השדה min מתעדכן אם המפתח המופחת הופך למינימום החדש.

<u>פונקציות עזר:</u> swapNodes

<u>סבכיות זמן:</u> לפעולת $decreaseKey$ יש מורכבות זמן של $O(log\ n)$, כאשר n הוא מספר הצמתים בערימה, מכיוון שהיא כרוכה במעבר העץ מהצומת לשורש.

תיעוד למחלקת $HeapNode$:

תיאור:

המחלקה $HeapNode$ מייצגת צומת בערימה הבינומית, המכילה מידע כגון הפריט, הפניות להורה, ילד ואחים, והדרגה שלו.

בַּנַאִים:

HeapNode(heapItem item, HeapNode child, HeapNode next, HeapNode parent, int rank):

הבנאי של המחלקה $HeapNode$ מאתחל מופע חדש של צומת בערימה הבינומיאלית, לוקח פרמטרים עבור הפריט המשויך, הורה, ילד, אח דרגה. הוא מקצה את הערכים שסופקו למשתני המופע המתאימים, ויוצר צומת חדש עם המאפיינים שצוינו.

תכונות:

Item: שדה מטיפוס $HeapItem$ מייצג את הפריט המשויך לצומת שמאחסן את פתיח ומידע

next: מייצג את האח הבה לפי הדרגות בסדר עולה

parent: מייצג את ההורה של צומת זה בערימה

child: מייצג בנו בעל הדרגה המקסימלית

rank: מייצג את הדרגה

מתודות:

פונקציית setChild:

הפונקציה $setChild$ במחלקה $HeapNode$ משמשת להגדרת הצאצא השמאלי ביותר של צומת. זה דורש מעדכן את שדה ה-`ילד` של הצומת הנוכחי כדי להצביע על צומת הילד שצוין ומתפלת בעדקון שאר השדות כמו הורה, אח וכו.

סבכיות זמן: $O(1)$.

פונקציית getKey: מחזירה את המפתיח השייך לצומת הנוכחי. סבכיות זמן: $O(1)$.

פונקציית getInfo: מחזירה את המידע השמור השייך לצומת הנוכחי. סבכיות זמן: $O(1)$.

תיעוד למחלקת $HeapItem$:

תיאור:

המחלקה $HeapItem$ מייצגת פריט בערימה הבינומית, המכיל מידע כגון המפתח, נתונים משויכים והפניה ל-$HeapNode$ המקביל בערימה.

בַּנַאִים:

HeapItem(HeapNode node, int key, String info):

הבנאי מאתחל מופע חדש של פריט בערימה הבינומיאלית, לוקח פרמטרים עבור הצומת, המפתח והמידע המשויכים, ומקצה את הערכים שסופקו למשתני המופע המתאימים, ויוצר פריט חדש עם הצומת שצוין נכסים.

תכונות:

node: הפניה ל-$HeapNode$ המתאים בערימה הבינומית, המקשר את הפריט לצומת שלו בערימה.

key: מייצג את ערך המפתח המשויך לפריט, המשמש להשוואה ולקביעת סדר הערימה.

info: מאחסן מידע נוסף או נתונים הקשורים לפריט, תוך מתן כל פרט נוסף או הקשר הרלוונטי לפריט.

מתודות:

פונקציית swapNodes(item):

נעדכן את הערך של ה-$item$ להיות $item.key - diff$

כיוון שאנו שנינו את הערך, צריך לבדוק אם הערימה עכשיו תקינה ואם לו לעדכן אותה.

אם הצומת הוא שורש, אזי אין מה לתקן וסיימנו.

אם הצומת אינו שורש, על ידי לולאת while נבדוק האם הערך של ה $item$ הוא יותר גדול מהערך של האב שלנו:

- אם כן, סיממנו.
- אחרית, ערך צומת הבן יותר קטנה מערך צומת האב ולכן נקרא על פונקציה $swapNodes(item1, item2)$ ואז שני הצמתים מחליפים מקומות.

מעדכנים את המינימום לפי ההתאם. אם הערך החדש הוא הערך המינימלי אזי השדה min עכשיו מצביע על $item.node$ (הצומת של ה-$item$)

<u>סיבוכיות הזמן:</u> לולאת ה-$while$ רצה $O(\log n)$ הרצות ושאר הפעולות ב-$O(1)$ ולכן סיבוכיות הזמן של הפונקציה היא $O(\log n)$.

חלק ב: חלק ניסויי/תיאורטי

פתרון שאלה 1

	מספר סידורי i	זמן ריצה (מילישניות)	מספר החיבורים הכולל	מספר העצים בסיום	סכום דרגות הצמתים שמחקנו
ניסוי ראשון	**1**	2	723	5	/
	2	3	2182	4	/
	3	6	6555	5	/
	4	16	19675	7	/
	5	64	59040	8	/
ניסוי שני	**1**	4	3282	5	2923
	2	8	11470	4	40381
	3	39	39869	5	36594
	4	83	135247	7	125413
	5	122	450682	8	421166
ניסוי שלישי	**1**	1	860	5	834
	2	2	3544	5	3518
	3	6	6825	5	6799
	4	29	30581	5	30555
	5	74	68873	5	68847

פתרון שאלה 2

<u>ניתוח עבור ניסוי **ראשון**:</u>

בניסוי זה מבצעים n פעולות הכנסה לערימה כאשר מתחילים עם ערימה ריקה. זה שקול לבעיית $increment$ שראינו בתרגול לכן הניתוח זהה ונקבל עלות זמן של $O(n)$

עכשוי נתאר הכשר בין מספר החיבורים לבן מספר העצים לבן n: נבחין ראשית כי בעץ בינותי בעל t צמתים יש בדיוק $t-1$ קשתות לכן בבנית עץ זה צריכים לבציע $t-1$ לינקים בדיוק. כעת עבור ערימה בינומית כללית בעלת n צמתים ו- d עצים מתקיים:

$$(\text{מספר החיבורים}) = n - d$$

ניתוח עבור ניסוי **שני**:

סדר ההכנסה לא משפיע לע ניתוח הסבביות של ההכנסות, וכל מחיקה עולה לכל היותר $O(\log n)$ זמן לכן בסה"כ: $O\left(\log n + \frac{n}{2}\log n\right) = O(n \log n)$

במקרה של הכנסה אקראית אי אפשר לדעת קלום לגבי הכשר בין מספר החיבורים לבן מספר העצים.

ניתוח עבור ניסוי **שלישי**:

ניתוח הסובביות זהה לסעיף קודם: $O(n \log n)$

נבחין כי האיבר המינימלי בערימה חייב להיות השורש של העץ העל הדרגה המנימלית. יותר על כך, בכל צומת האיבר המינימרי בתת עץ שלה (לא כולל צומת עצמה) הוא הילד העל הדרגה המינימלית (כל זה נובע מהעובדה שהכנסנו בסדר יורד). לכן כאשר לבצעין $deleteMin$ אנחנו בפועל מפרקים את העץ הקטן לעצים קטנים יותר ולכן לא נצתרך לינקים חדשים ובנוסף לכן נשארים עם תת העייה שקולה בגודל $n - 1$. לכן לעולם לא נבציע עוד לינקים. אזי המשוואה דומה לניתוח עבור הניסוי הראשון.