`FibonacciHeap`

מחלקה המייצגת את ערימת הפיבונאצ'י. שומרת מספר משתנים:

size: מסוג int , שומר את מספר הצמתים בערימה.

numTrees: מסוג int , שומר את מספר העצים בערימה.

numMarked: מסוג int , שומר את מספר הצמתים המסומנים בערימה.

start: מסוג HeapNode, שומר מצביע לצומת הראשון בערימה.

min: מסוג HeapNode, שומר מצביע לצומת המינימלי בערימה (בהכרח אחד משורשי העצים).

totalCuts: מסוג static int, מאחסן את מספר החיתוכים שנעשו במחלקה.

totalLinks: מסוג static int, מאחסן את מספר החיבורים שנעשו במחלקה.

`isEmpty()`

return type – boolean

סיבוכיות – O(1)

תיאור קצר – מחזיר true אם הערימה ריקה (שדה size הוא 0), אחרת false.

`insert(int key)`

return type – HeapNode

סיבוכיות – O(1)

תיאור קצר – מכניס לערימה צומת חדש עם מפתח key. יוכנס כצומת הראשונה בערימה. מחזיר את הצומת שנוצרה. נדאג לעדכן את כל המצביעים הדרושים ואת השדות size, numTrees, start, min.

`deleteMin()`

return type – none

סיבוכיות – WC: O(n), amortize: O(logn)

תיאור קצר – אם גודל הערימה הוא 1, נעדכן את כל שדות הערימה לכך שהיא ריקה. אחרת נאתחל מערך בגודל פעמיים logn, אם יש רק עץ אחד נבצע את תהליך consolidating על ילדיו, אחרת "נוסיף" לשורשים את ילדי המינימום (אם יש) ועליהם נבצע תהליך זה, תוך שימוש במתודה link.

`link(HeapNode x,HeapNode y)`

return type – HeapNode

סיבוכיות – O(1)

תיאור קצר – מקבל 2 צמתים שמייצגים שורשי עצים, ותולה שורש אחד על השני ומאחד את העצים, וכן מתחזק את השדות numTrees ו-totalLinks.

```
findMin()
```

return type – HeapNode

סיבוכיות – O(1)

תיאור קצר – מחזיר את הצומת המינימלי בערימה (שדה min).

```
meld (FibonacciHeap heap2)
```

return type – none

סיבוכיות – O(1)

תיאור קצר – מאחד שתי ערימות לערימה אחת (in place) ע"י איחוד שורשי העצים בשתי הערימות ועדכון בהתאם של שדות start, min, numMarked, numTrees ו-size.

```
size()
```

return type – int

סיבוכיות – O(1)

תיאור קצר – מחזיר את כמות הצמתים בערימה (שדה size).

```
countersRep()
```

return type – int[]

סיבוכיות – O(n)

תיאור קצר – מחזיר מערך בו הערך של התא ה-i הוא מספר העצים בערימה מדרגה i (דרגה של עץ הוא מספר הבנים של השורש). נעשה זאת בשני שלבים. ראשית נמצא את דרגת העץ הגבוהה ביותר. ניצור מערך בגודל דרגה זו. כעת נעבור על כל השורשים של העצים בערימה, ונוסיף 1 לתא המתאים במערך. במקרה הגרוע ביותר, יהיו לנו n עצים שונים בערימה בגודל n (אם כל עץ מדרגה 0) ולכן הסיבוכיות הינה O(n).

```
delete(HeapNode x)
```

return type – none

סיבוכיות – O(log n)

תיאור קצר – מוחק מהערימה את צומת x. נבצע זאת על ידי שראשית נוריד את המפתח של צומת x להיות המפתח של הצומת המינימלית פחות 1 באמצעות decreaseKey. כעת צומת x הינה הצומת המינימלית, ולכן אפשר לקרוא לdeleteMin.

```
decreaseKey(HeapNode x, int delta)
```

return type – none

סיבוכיות – O(1)

תיאור קצר – מעדכן את המפתח של x להיות x.key-delta. נעדכן את המבנה של העץ ושל הערימה באמצעות cut או cascadingCut כך שיקיימו את האינוריאנטות של עץ וערימה. נבצע cut אם האב של צומת x הוא צומת לא מסומן, והמפתח של x קטן משל האב שלו, ונהפוך את האב למסומן. נבצע cascadingCut אם האב של צומת x הוא צומת מסומן והמפתח של x קטן משל האב שלו. נעדכן את min בהתאם.

`potential()`

return type – int

סיבוכיות – O(1)

תיאור קצר – נחזיר את הפוטנציאל של הערימה כאשר נחשב אותה באמצעות הנוסחה $numTrees + 2 * numMarked$.

`cut(HeapNode x, HeapNode y)`

return type – none

סיבוכיות – O(1)

תיאור קצר – חותך את צומת x מאב y. נעדכן את המצביעים בהתאם ונגדיר את x להיות הצומת הראשון בערימה. נגדיל את numTrees ו totalCuts באחד. בנוסף, נדאג להפוך את x לצומת לא מסומן.

`cascadingCut(HeapNode x, HeapNode y)`

return type – none

סיבוכיות – O(1)

תיאור קצר – ראשית נבצע cut(x,y). כעת, אם y היה מסומן, נבצע cascadingCut(y,y.getParent()). אם y לא היה מסומן, נסמן אותו כעת, נגדיל את מספר המסומנים בערימה ב1, ונסיים.

`totalLinks()`

return type – int

סיבוכיות – O(1)

תיאור קצר – פונקציה סטטית המחזירה את מספר החיבורים שנעשו בסך הכל במחלקה. נחזיר את המשתנה totalLinks.

`totalCuts()`

return type – int

סיבוכיות – O(1)

תיאור קצר – פונקציה סטטית המחזירה את מספר החיתוכים שנעשו בסך הכל במחלקה. נחזיר את המשתנה totalCuts.

```
kMin(FibonacciHeap H, int k)
```

return type – int []

סיבוכיות – O(k(logk+deg(H))

תיאור קצר – מאתחל מערך בגודל k, בו האיבר הראשון זה שורש העץ ומכניס את כל ילדיו לערימת עזר. כעת עד שנגיע ל-k איברים במערך, אנחנו נמחק את האיבר המינימלי בערימת העזר ונכניס את כל ילדיו. כדי לדעת את מיקום הצומת שנמחק מערימת העזר בערימה המקורית, אנחנו נתחזק שדה original שיצביע למיקומו שם. הסיבוכיות היא כפי שרשום למעלה כי k פעמים אנחנו מפעילים את המתודה insertChildren כלומר k*deg(H) וכן k פעמים מחקנו את המינימלי מערימת העזר, שגודלה לכל היותר k*deg(H) ולכן הסיבוכיות של כל מחיקות המינימום היא klog(k*deg(H)) ולכן בסה"כ הסיבוכיות שפי שרשום למעלה.

```
insertChildren(HeapNode st, FibonacciHeap assistent)
```

return type – none

סיבוכיות – deg(H)

תיאור קצר – מכניסה את כל הילדים של צומת כלשהו לערימת העזר שיצרנו במתודה Kmin תוך כדי עדכון שדה original. הסיבוכיות היא כזו כי לכל צומת יש לכל היותר deg(H) ילדים.

HeapNode

מחלקה המייצגת צומת בערימה. שומרת מספר משתנים:

key: מסוג int, מייצג את המפתח של הצומת.

rank: מסוג int, מייצג את הדרגה של הצומת אותה נגדיר כמספר הילדים של הצומת.

marked: מסוג boolean, true אם הצומת מסומנת, אחרת false. נסמן צומת אם "חתכנו ממנה" בן שלה. צומת מדרגה 0 לא מסומנת.

child: מסוג HeapNode, מצביע לבן השמאלי ביותר של הצומת. אם לא קיים כזה, null.

next: מסוג HeapNode, מצביע לאח מימין באופן מעגלי.

prev: מסוג HeapNode, מצביע לאח משמאל באופן מעגלי.

parent: מסוג HeapNode, מצביע לאב, null אם לא קיים.

original: מסוג HeapNode, רלוונטי רק למתודה Kmin, שדה זה יצביע מצומת בערימת עזר לצומת הזהה בערימה המקורית.

נשים לב לפרט חשוב. ייתכן ואב של צומת מסויימת, לא יוגדר כ-child של צומת זו שכן child מצביע רק לבן השמאלי ביותר ואילו לצומת יש רק אב אחד.

```
HeapNode(int key)
```

return type – none

סיבוכיות – O(1)

תיאור קצר – בנאי של המחלקה. מגדיר את המפתח להיות key וכל שאר השדות מקבלים ערכים דיפולטיביים.

```
getRank()
```

return type – int

סיבוכיות – O(1)

תיאור קצר – מחזיר את הדרגה של הצומת (מספר הבנים של הצומת).

```
setRank(int rank)
```

return type – none

סיבוכיות – O(1)

תיאור קצר – מגדיר את הדרגה של הצומת להיות rank

```
isMarked()
```

return type – boolean

סיבוכיות – O(1)

תיאור קצר – מחזיר את ערך המשתנה marked.

```
setMarked(boolean marked)
```

return type – none

סיבוכיות – O(1)

תיאור קצר – מגדיר את הערך של השדה marked להיות הערך של marked שהועבר עם הקריאה למתודה.

```
getChild()
```

return type – HeapNode

סיבוכיות – O(1)

תיאור קצר – מחזיר את הערך שבשדה child.

```
setChild(HeapNode child)
```

return type – none

סיבוכיות – O(1)

תיאור קצר – מגדיר את השדה child להיות child שהועבר עם הקריאה למתודה.

```
getNext()
```

return type – HeapNode

סיבוכיות – O(1)

תיאור קצר – מחזיר את הערך שבשדה next.

```
setNext(HeapNode next)
```

return type – none

סיבוכיות – O(1)

תיאור קצר – מגדיר את השדה next להיות next שהועבר עם הקריאה למתודה.

```
getPrev()
```

return type – HeapNode

סיבוכיות – O(1)

תיאור קצר – מחזיר את הערך שבשדה prev.

```
setPrev(HeapNode prev)
```

return type – none

סיבוכיות – O(1)

תיאור קצר – מגדיר את השדה prev להיות prev שהועבר עם הקריאה למתודה.

```
getParent()
```

return type – HeapNode

סיבוכיות – O(1)

תיאור קצר – מחזיר את הערך שבשדה parent.

```
setParent(HeapNode parent)
```

return type – none

סיבוכיות – O(1)

תיאור קצר – מגדיר את השדה parent להיות parent שהועבר עם הקריאה למתודה.

```
setKey(int key)
```

return type – none

סיבוכיות – O(1)

תיאור קצר – מגדיר את השדה key להיות key שהועבר עם הקריאה למתודה.

```
getKey()
```

return type – int

סיבוכיות – O(1)

תיאור קצר – מחזיר את השדה key.

```
setOriginal(int key)
```

return type – none

סיבוכיות – O(1)

תיאור קצר – מגדיר את השדה original להיות הצומת שהועבר עם הקריאה למתודה.

```
getOriginal()
```

return type – HeapNode

סיבוכיות – O(1)

תיאור קצר – מחזיר את השדה original.