Data Structures – Practical Assignment 2

Gal Oren (302378633, orengal1), Katz Saar (318966249, saarkatz)

# Class documentation:

|  |
| --- |
| FibonacciHeap |

Static Fields:

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| **int** | *totalCuts* | Keeps the total number of cuts made by instances of FibonacciHeap. |
| **int** | *totalLinks* | Keeps the total number of links made by instances of FibonacciHeap. |

Static methods:

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Signature | Description | Time complexity |
| **int** | totalCuts() | Returns *totalCuts.* |  |
| **int** | totalLinks() | Returns *totalLinks.* |  |

Fields:

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| **int** | **numMarks** | The number of marked nodes in the heap. A node becomes only if one of its child was cut. |
| **int** | **numRoots** | The number of root in the heap. |
| **int** | **size** | The number of nodes in the heap. |
| HeapNode | **min** | A reference to the minimum of the heap. Root is null if the heap is empty. |

Public methods:

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Signature | Description | Time complexity |
| **boolean** | empty() | Returns if the heap is empty (), false otherwise. |  |
| HeapNode | insert(**int** key) | Inserts a new node with the key into the heap.  Returns a reference to the node. |  |
| **void** | deleteMin() | Deletes the minimum of the tree. One-pass successive linking is performed of the heap after a call to deleteMin. | Amortized |
| HeapNode | findMin() | Returns the minimum node of the heap. |  |
| **void** | meld(FibonacciHeap heap2) | Meld this heap with heap2. The resulting heap is contained in this instance and heap2 is emptied completely. |  |
| **int** | size() | Returns the size of the heap. |  |
| **int**[] | countersRep() | Return a counters array, where the value of the -th entry is the number of trees of order in the heap. |  |
| **void** | delete(HeapNode x) | Deletes the node from the heap. One-pass successive linking is then performed. If the node has a parent a cascading cut is performed between and it’s parent. While it’s true that the children of are disconnected from it, these cuts are not counted toward . | Amortized |
| **int** | potential() | Returns the potential of the tree. The potential is defined as the number of root plus twice the number of marked nodes. |  |

Private methods:

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Signature | Description | Time complexity |
| **static void** | cutWithoutCounting(HeapNode node x) | Cuts the node from is parent. If or its parent are null a NullPointerException is thrown. This is an internal function so this should never happen. |  |
| **void** | cut(HeapNode x) | Cuts the node from is parent using cutWithoutCounting and counts the cut toward |  |
| **void** | cascadingCutCounting(HeapNode x, **boolean** doCount) | Performs cascading cut between and its parent. If doCount is the cut is performed using cut, counting all the cuts.  Otherwise cutWithoutCounting is used, not counting any cut. |  |
| **void** | cascadingCut(HeapNode x) | Performs cascadingCutCounting with doCount == |  |
| HeapNode | link(HeapNode node1, HeapNode node2) | Links and . The node with the smaller key is linked to be the parent of the other and is then returned. The link is then counted towards . |  |
| **void** | onePassSuccessiveLinking(HeapNode first, HeapNode second) | Performs one-pass successive linking of the heap. | Amortized |
| **void** | meld() | Connects and in a double circular linked list fashion. |  |
| **void** | removeNodeFromList(HeapNode node) | Disconnects from its neighbors. |  |

|  |
| --- |
| HeapNode |

Fields:

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| String | **value** | The value contained by the node. Not used in this specific project. |
| **int** | **key** | The key of the node. |
| **int** | **rank** | The rank of the node (number of children of the node). |
| **boolean** | **mark** | Boolean value indicating whether the node is marked or not. |
| HeapNode | **child** | A reference to the child of the node. |
| HeapNode | **next** | A reference to the next neighbor of the node. |
| HeapNode | **prev** | A reference to the previous neighbor of the node. |
| HeapNode | **parent** | A reference to the parent of the node. |

Methods: Time complexity of all methods is

There is a getter and a setter for each of the fields of the node. The only public ones are the getters for , which is not used, and for .

# Benchmark measurements:

Sequence 1:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No. | No. items | Run-Time | totalLinks | totalCuts | Potential |
| 0 | 1000 | 0.001765 | 0 | 0 | 1000 |
| 1 | 1000 | 0.000086 | 0 | 0 | 1000 |
| 2 | 2000 | 0.000141 | 0 | 0 | 2000 |
| 3 | 3000 | 0.000304 | 0 | 0 | 3000 |

Sequence 2:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No. | No. items | Run-Time | totalLinks | totalCuts | Potential |
| 1 | 1000 | 0.004578 | 1460 | 0 | 8 |
| 2 | 2000 | 0.003145 | 4210 | 0 | 8 |
| 3 | 3000 | 0.002529 | 8554 | 0 | 8 |

באופן מוזר, הריצה של הערימה הראשונה בכל סדרה רצה לאט יותר בסדר גודל עבור סדרה 1 ופי 2 לערך עבור סדרה 2. יתכן שהריצה של העץ מהירה מכדי שהמדידה הזמן תהיה מדויקת. עבור סדרת ריצות עד איברים בקפיצות של ניתן להבחין בגידול בזמן הריצה. המצב הרצוי היה לבצע מספר רב של ריצות עבור כל כמות איברים ולמצע אולם לא ניתן לעשות זאת בצורה טובה כיוון שהערכים עבור ו- נספרים עבור כל הערימות מתחילת התוכנית.

סדרה 1:

* זמן ריצה אסימפטוטי: – כל פעולת מתבצעת ב-
* *סך פעולות : 0, לא מבצע פעולות כלל.*
* *סך פעולות : 0, לא מבצע פעולות כלל.*
* *פוטנציאל: , כיוון ש- רק מוסיף איברים מבלי לשנות את מבנה הרשימה, עבור סדרה של פעולות כל האיברים שנוספו יהיו שורשים ולכן הפוטנציאל יהיה מספר השורשים.*

*סדרה 2:*

* זמן ריצה אסימפטוטי: – פעולות מתבצעות ב- . לאחר מכן פעולות שמתבצעות ב-. בסך הכל נקבל .
* *סך פעולות : - לא מבצע פעולות כלל. לאחר כל פעולת מתבצע .*

*לפני ההפעלה הראשונה של ישנם שורשים בדרגה . לאחר מכן נמחק שורש אחד ומבוצע שמחבר את השורשים בזוגות ולכן מבוצעות לכל היותר פעולות . לאחר מכן מבוצע שוב ולאחריו עם כ- פעולות.* נשים לב כי לאחר מספר פעולות הערימה תהייה מצומצמת במיוחד ואז רוב פעולות ה- יבואו מהילדים של המינימום שנמחק שהתחברו עם שאר העצים הערימה. מספר הילדים של המינימום הוא לכל היותר

*בהמשכה לכך נקבל שמספר פעולות ה- הוא כ-*

* *סך פעולות : 0, ו- לא מבצעים פעולות כלל.*
* *פוטנציאל: – בסוף סדרת הפעולות נותרו הערימה איברים והערימה מצומצמת עד כדי הילדים של המינימום שנמחק בפעולה האחרונה. מספר הילדים המקסימלי של המינימום הוא ובעץ מצומצם יש לכל היותר שורשים שונים ולכן הפוטנציאל הסופי יהיה .*