Minimum Heap Algorithms

Kaminski / cs3310

Storage 1) array of nodes - a node contains just the Data, but no LChPtr or RChPtr
2) N - that is, locations [0] to [N-1] contain valid data, anything in [N] or beyond is garbage

Formula `node[i]`'s LChild is at: **`2i+1`**, its RChild is at: **`2i+2`**, its Parent is at: **`trunc((i-1)/2)`**

Create - 2 Algorithms:
#1 Start a Baby Heap, then keep using Insert algorithm
#2 Special Create (this is more efficient & useful if array's already filled with data & you need to heapify it)

Special Create

```
for i = "subscript of last node which has a child" downto 0
    call WalkDown(i)
```

NOTE: the last node in the heap is `item[N-1]`, so its parent is in location: `trunc(((N-1)-1)/2)`

WalkDown (IN: StartFrom)

```
I = StartFrom
SmCh = SubOfSmCh(i)
while ((2i+1) <= (N-1)) AND (item[i] > item[SmCh])
{   swap item[i] with item[SmCh]
    i = SmCh
    SmCh = SubOfSmCh(i)
}
```

NOTE: the SWAP approach is more intuitive, but the SHIFT approach is more efficient

Insert (IN: NewItem)

```
location[N] = NewItem
increment N
call WalkUp(N-1)
```

NOTE: who should check `IsHeapFull`, the `Insert` algorithm itself or the caller of `Insert`?

WalkUp (IN: StartFrom)

```
i = StartFrom
while (i > 0) AND (item[i] < item[parent(i)])
{   swap item[i] with item[parent(i)]
    i = parent(i)
}
```

NOTE: SWAP is more intuitive, but SHIFT approach is more efficient

Delete (OUT: MinItem)

```
MinItem = item[0]
put item[N-1] in item[0]
decrement N
call WalkDown(0)
```

NOTE: who should check `IsHeapEmpty`, the `Delete` algorithm itself or the caller of `Delete`?

SubOfSmCh (i) function

```
if ((2i+2) > (N-1)) OR (item[2i+1] <= item[2i+2])
    return 2i+1
else
    return 2i+2
```

IsHeapFull `check if N == MaxN`
IsHeapEmpty `check if N == 0`

EmptyOutHeap

```
while NOT IsHeapEmpty
    call Delete
```