Worksheet 33 - Group 1

Worksheet Group 1 Members

Marc Clinedinst: clinedim@onid.oregonstate.edu
Kelby Faessler: faesslek@onid.oregonstate.edu
James Fitzwater: fitzwatj@onid.oregonstate.edu
Tom Gariepy: gariepyt@onid.oregonstate.edu
Sean Reilly: reillys@onid.oregonstate.edu
Joseph Struth: struthj@onid.oregonstate.edu

Collaborators

Marc, Kelby, James, Tom, Sean, Joseph

Worksheet 33: Heaps and Priority Queues

In this assignment, we implement several different functions which control the behavior of a heap implemented on top of a dynamic array. More specifically, we implement the `_adjustHeap` and `addHeap` functions. Both of these functions are defined below and are accompanied by comments where needed.

```
/*
      The adjustHeap function adjusts the order of the values in the heap such that its
      essential ordering property is maintained.  This function is typically used after the
      root is removed from the heap.  It recursively "percolates down" the tree, adjusting
      elements s that they are in the correct order.  It begins by checking that the dynamic
      array is not null and that the value of max is less than the size of the array.  It then
      calculates the value of the left child and right child.  If the right child is less than
      max, this means that there are two children.  The index of the smallest of these two
      children is calculated.  If a swap is necessary, then it occurs, and the heap is adjusted
      again.  If the left child is less than max, this means there is one child.  If a swap is
```

```
	necessary, then it occurs, and the heap is adjusted again.
*/
void _adjustHeap(struct DynArr *da, int max, int position) {
	assert(da != 0);
	assert(max < sizeDynArr(da));

	int leftChild = (2 * position) + 1,
		rightChild = (2 * position) + 2,
		smallestChild;

	if (rightChild <= max) {
		smallestChild = indexSmallestDynArr(da, leftChild, rightChild);
		if (LT(da->data[smallestChild], da->data[position])) {
			swapDynArr(da, smallestChild, position);
			_adjustHeap(da, max, smallestChild);
		}
	} else if (leftChild <= max) {
		if (LT(da->data[leftChild], da->data[position])) {
			swapDynArr(da, leftChild, position);
			_adjustHeap(da, max, leftChild);
		}
	}
}

/*
	This function adds a value to a heap.  It first checks that the dynamic array is not null.
	It then adds the new value to the last position in the array and calculates the position
	of the new value.  It then navigates to the top of the tree by accessing each node's
	parent node, as long as swaps are necessary; along the way, it performs these swaps. Once
	a swap is no longer necessary, or the function has reached the root node, the function
	terminates.
```

```
*/
void addHeap(struct DynArr *da, TYPE value) {
	assert(da != 0);
	addDynArr(da, value);

	int position = sizeDynArr(da) - 1;
	int parent;

	while (position != 0) {
		parent = (position - 1) / 2;

		if (da->data[position] < da->data[parent]) {
			swapDynArr(da, position, parent);
			position = parent;
		} else {
			return;
		}
	}
}
```

Piazza Discussion

https://piazza.com/class/ib2kus4hsie528?cid=209