TADs for Integrative Task 1

Samuel Navia Quiceno (A00405006)
Simón García (A00371828)
Juan Camilo Criollo Cifuentes(A00402515)

TAD Hash Table
Hash Table = {e_1, e_2, …, e_m} where every element e_x is composed by (K_x, V_x) for a Hash Table with keys Type K, and values type V with a size m - 1.

{inv: HashTable size = m - 1}
{inv: All keys K must be unique.}
{inv: Every e_x must be accessed through hash(K_x)}

Primitive Operations: ● createHashTable: -> HashTable ● Hash: K -> int ● Add: e -> HashTable ● Search: K -> V ● Delete: K -> HashTable ● getValues: -> e

createHashTable() - Constructor
“Creates an empty Hash Table of type (K , V)”
{pre: True}
{post: HashTable = {(e_1), (e_2), …, (e_m)}, Every e = null}

Hash(K) - Analyzer
“Enter a key K and return the index in which it will be located in the Hash Table.”
{pre: K is an integer K >= 0}
{post: hash(K key)= $m\ *\ (kA\ \%\ mod\ 1)$ where hash(k) belongs to HashTable[h(k)]}

add(e) - Modifier
“Add a new element e (composed of value V and key K) in HashTable[h(K)]”
{pre: K and V must be the same type of object as HashTable(K, V)}

{pre: K must be a unique key}

{post: HashTable[hash(K)] = e}

search(K) - Analyzer

“Search for an e using its key K and return the value associated with it.”

{pre: K must match the type of K of the HashTable}

{post: V}
{post: null, K was not found. }

Delete(K) - Modifier

“Delete e_x from the HashTable using its key K_x.”

{pre: K_x must match the type of K of the HashTable}

{post: HashTable[h(K_x)]}
{post:KeyNotFoundException, K_x couldn't be found}

getValues() - Analyzer

“Returns every value e saved inside the HashTable”

{pre: True}

{post: {e_1, e_2, …, e_m}}

TAD Stack

Stack= {a_1, a_2, …,a_m} size m - 1 and a_m is the last added node type T.

{inv: The only visible element is a_m}
{inv: a_x can only be added at a_m}
{inv: a_m is the only node that can be removed}
{inv: order a_1, a_2, …, a_m must not be changed}

Primitive Operations:

- createStack: -> Stack
- isEmpty: -> boolean

- push: a -> Stack
- Top: -> a
- Pop: -> Stack

createStack() - Constructor

“Creates an empty Stack of type T”

{pre: True}

{post: Stack ={a_1}, where a_1 is null}

isEmpty() - Analyzer

“Determines if the stack is empty.”

{pre: True}

{post: true if a_m == null}
{post: false if a_m != null}

push(a_m) - Modifier

“Add node a_m to the end of the stack.”

{pre: a_m must match the type T of the Stack}

{post: Stack = {a_1, a_2, …,a_{m-1}, a_m}}

top() - Analyzer

“Returns the value of the last node in the stack (a_m)”

{pre: True}

{post: a_m}
{post: StackException, if stack is empty}

pop() - Modifier

“Deletes the last element in the stack (a_m)”

{pre: True}

{post: Stack = {a_1, a_2, …, a_{m-1}}, where a_{m-1} becomes the new a_m afterwards}
{post: StackException, if stack is empty}

TAD Queue

Queue = {a_1, a_2, …, a_m} of size m-1 and the last added node type T is a_m.

{inv: a_m must be added at the end of the stack}
{inv: a_1 must be the first deleted element}
{inv: order a_1, a_2, …, a_m must not be changed}

Primitive Operations:

- createQueue: -> Queue
- isEmpty: -> boolean
- enqueue: a -> Queue
- dequeue: -> a
- front: -> a

createQueue() - Constructor

“Creates an empty Queue of type T”

{pre: True}

{post: Queue ={a_1}, where a_1 is null}

isEmpty() - Analyzer

“Determines if the queue is empty.”

{pre: True}

{post: true if a_1 == null}
{post: false if a_1 != null}

enqueue(a_m) - Modifier

“Add node a_m to the end of the queue”

{pre: a_m must be of the same type T as the Queue.}

{post: Queue = {a_1, a_2, …,a_{m-1}, a_m}}

dequeue() - Modifier

“Retrieves and then deletes the first element in the queue(a_1)”

{pre: True}

{post: returns a_1 and Queue = {a_2, a_3, …, a_m}, where a_2 becomes the new a_1 afterwards}
{post: QueueException, if queue is empty}

front()

“Returns the value of the first node in the queue (a_1)”

{pre: True}

{post: a}
{post: QueueException, if the queue is empty.}

TAD Priority Queue

PriorityQueue = {a_1, a_2, …, a_m} where each element a_i and a_j have an assigned priorities p_i and p_j respectively so that if p_i >= p_j then i <= j and so { ..., a_i , a_j , …}.

{inv:The element a_m must be added while maintaining the order of priority.}
{inv: a_1 must be the first element removed from the queue, where a_1 has the highest priority and the first added in its priority group.}
{inv: The order of elements with the same priority (if any) must be preserved as they were added.}

Primitive Operations:

- createPriorityQueue: -> PriorityQueue
- isEmpty: -> boolean
- enqueue: a -> PriorityQueue
- dequeue: -> a
- front: -> a

createPriorityQueue() - Constructor

“Creates an empty Priority Queue of type T.”
{pre: True}
{post: PriorityQueue = {a_1} where a_1 = null}

isEmpty() - Analyzer
“Determines if the priority queue is empty.”
{pre: True}
{post: true if a_1 == null}
{post: false if a_1 != null}

enqueue(a_x) - Modifier
“Add an item a_x with a given priority to the queue.”
{pre: item must match the type T}
{post: PriorityQueue = {a_1, a_2, …, a_x, … a_m}, where a_x is added maintaining the priority order and after any element a_y with the same priority but added before.}

dequeue() - Modifier
“Retrieves and then deletes the first element in the priority queue (a_1)”
{pre: True}
{post: returns a_1 (it had the highest priority and was the first of its priority added) and PriorityQueue is updated accordingly to keep the priority order.} {post: PriorityQueueException, if PriorityQueue is empty.}

front() - Analyzer
“Returns the value of the item with the first added item with the highest priority (a_1)”
{pre: True}

{post: a}
{post: PriorityQueueException, if PriorityQueue is empty}