Definición de los TAD de las estructuras a utilizar

|  |
| --- |
| TAD Hash Table |
| Una tabla hash es una estructura de datos que almacena valores en un arreglo en forma de diccionario. Es decir, cuando se ingresa un objeto a la tabla hash, se hace mediante una llave y un valor, la llave se utiliza para determinar la posición que el objeto tendrá en el arreglo mediante una función hash. Dicha función hash es de la forma donde k es la clave y m un número entero. Es decir que la función hash se encarga de transformar las claves que recibe en números enteros, esto con el objetivo de darle a los objetos una posición en el arreglo. Lo que se guardará en dicha posición será el valor del objeto. |
| El invariante de una tabla hash es que para una llave k, siempre se obtiene el mismo valor del código hash, independientemente de cuántas veces se realice la operación. Para todo k se cumple que cuando la ingreso a la función hash obtengo siempre la misma posición.  Teniendo k una clave, h la función hash, i una posición en el arreglo y m el tamaño del arreglo: |
| Operaciones primitivas:   * HashTable: size 🡪 HashTable * hashFunction: key 🡪 int * add: key x value 🡪 boolean * search: key 🡪 value * delete: key 🡪 value * getSize: 🡪 int |

|  |
| --- |
| **HashTable(size)**  Crea una nueva tabla hash vacía con el tamaño indicado por parámetro.  {pre: }  {post: HashTable HashTable.size = size} |

|  |
| --- |
| **hashFunction (key)**  Retorna una posición en el arreglo.  {pre: HashTable.}  {post: } |

|  |
| --- |
| **add (key, value)**  con la clave ingresada por parámetro asigna una posición en el arreglo al valor que también se recibe por parámetro. La posición es determinada por la función hash.  {pre: HashTable.}  {post: } |

|  |
| --- |
| **search (key)**  con la clave ingresada por parámetro retorna el valor que se encuentra en la posición que dicha clave mapea en el arreglo.  {pre: HashTable. }  {post: } |

|  |
| --- |
| **delete (key)**  con la clave ingresada por parámetro retorna el valor que se encuentra en la posición que dicha clave mapea en el arreglo y elimina dicho valor del arreglo.  {pre: HashTable. }  {post: ,  } |

|  |
| --- |
| **getSize ()**  Retorna el tamaño del arreglo. No es necesariamente el número de elementos en la tabla, sino el espacio que se asigna en memoria para el arreglo.  {pre: HashTable. }  {post: size} |

|  |
| --- |
| TAD Priority Queue |
| Una ts un |
| El in |
|  |

|  |
| --- |
| TAD Stack |
| Una pila (stack) es una secuencia lineal de un número arbitrario de elementos. Las operaciones que permite son la inserción y eliminación solamente al tope de la misma. El tope es el último elemento agregado. La pila es una lista que sigue un patron LIFO (el último en llegar es el primero en salir). Una pila puede estar vacía o tener una secuencia de elementos.  Stack ={ Ø } |
| El invariante de una pila es que el acceso a los elementos de la pila está limitado a hacerse siempre desde el tope.  {inv: { eliminar: 🡪 n , | n es el último elemento en haberse agregado insertar: n | n se inserta al tope de la pila }} |
| Operaciones primitivas:   * isEmpty: 🡪 boolean * push: item 🡪 void * top: 🡪 item * pop: 🡪 item |

|  |
| --- |
| **isEmpty ()**  Indica si la pila está vacía.  {pre: TRUE }  {post: TRUE si la pila está vacía, FALSE en caso contrario} |

|  |
| --- |
| **push (item)**  Agrega el elemento a la pila.  {pre: TRUE }  {post: item es el nuevo tope de la pila} |

|  |
| --- |
| **top (item)**  Retorna mas no remueve el elemento al tope de la pila.  {pre: stack.isEmpty() debe ser FALSE }  {post: item, el tope de la pila} |

|  |
| --- |
| **pop ()**  Retorna y remueve el elemento al tope de la pila.  {pre: stack.isEmpty() debe ser FALSE }  {post: item, el tope de la pila stack} |

|  |
| --- |
| TAD Queue |
| Una ts un |
| El in |
|  |