|  |  |  |
| --- | --- | --- |
| TAD MaxPriorityQueue | | |
| Conjunto de elementos: [x1 << p1] [x2 << p2] … [xn << pn] | | |
| inv : ∀i, j, donde 1 ≤ i, j ≤ n, Si i ≠ j, entonces P\_i ≥ P\_j | | |
| **Operación** | **Entrada -> Salida** | **Tipo** |
| MaxPriorityQueue | -> MaxPriorityQueue | Constructora |
| insert | [x1 << p1] -> MaxPriorityQueue | Modificadora |
| eliminar | MaxPriorityQueue -> MaxPriorityQueue | Modificadora |
| isEmpty | MaxPriorityQueue -> boolean | Analizadora |
| search |  |  |

|  |
| --- |
| **Analisis de metodos MaxPriority** |
| Método: MaxPriorityQueue()  /\* Crea una nueva instancia de una Cola de prioridad vacía. \*/  {pre: true}  {post : MaxPriorityQueue= nil } |
| Método: insert(X elem, P prioridad)  /\* Inserta un elemento X con prioridad P a la cola de prioridades \*/  {pre: X ∉ MaxPriorityQueue}  {post: MaxPriorityQueue = [x1 << p1] [x2 << p2] … [xk << pk] [X<<P]... [xn<<pn]<-, pk >= P > pk+1 } |
| Método: search(X elem)  /\* Busca un elemento de la Cola de prioridad \*/  {pre: X ∈ HashTable}  {post: ∀ E, H, K : (search(H, K) = E) → (E ∈ H)} |
| Método: delete(K key)  /\* Elimina el primer elemento de la cola de prioridad \*/  {pre: HashTable != NIL}  post: MaxPriorityQueue = [x2 << p2] … [xn << pn]<-- |
| Método: isEmpty()  /\* Verifica si la Cola de prioridad está vacía. \*/  {pre: true}  post: Devuelve true si la Cola de prioridad no contiene ningún elemento, y false si contiene al menos un elemento. |

|  |  |  |
| --- | --- | --- |
| TAD HashTableChaining | | |
| Conjunto de Elementos:  HTC= { <key₁, value₁>, <key₂, value₂>, ..., <keyₙ, valueₙ> } | | |
| ∀ i,j ∈ [0, N-1], i ≠ j ⇒ (ki ≠ kj) ∨ (h(ki) ≠ h(kj)) ∨ (ki = kj ∧ nodes[h(ki)] contiene (ki, vi) ∧ nodes[h(kj)] contiene (kj, vj)) | | |
| **Operacion** | **Entrada -> Salida** | **Tipo** |
| HashTableChning | slotNumber -> HashtableChaining | Constructora |
| insert | (key, value) -> Stack | Modificadora |
| search | key -> value | Analizadora |
| delete | key -> Stack | Modificadora |
| size | Stack-> Stack.size | Analizadora |
| isEmpty | Stack -> boolean | Analizadora |
| keys | HashTable -> CK = { key₁, key₂, …., keyₙ} | Analizadora |
| values | HashTable -> CV = {value₁, value₂, ..., valueₙ} | Analizadora |

|  |
| --- |
| **Analisis de metodos HASHTABLE** |
| Método: HashTableChaining()  /\* Crea una nueva instancia de una Hashtable con encadenamiento (chaining) vacía. \*/  {pre: true}  {post : HashTableChaining = null [slotNumber]} |
| Método: insert(K key, V value)  /\* Inserta un par clave-valor en la Hashtable. \*/  {pre: keyN ∉ HashTable}  {post: HashTable = <key1, value1>, …<keyN, elem>} |
| Método: search(K key)  /\* Busca un valor asociado a una clave en la Hashtable. \*/  {pre: Key ∈ HashTable}  {post: ∀ E, H, K : (search(H, K) = E) → (E ∈ H)} |
| Método: delete(K key)  /\* Elimina un par clave-valor de la Hashtable. \*/  {pre: KeyN ∈ HashTable}  post: HashTable = {<Key1, value1>, …. } - <KeyN, ValueN> |
| Método: size()  /\* Devuelve el número de pares clave-valor almacenados en la Hashtable. \*/  pre: { true }  post: return {int} |
| Método: isEmpty()  /\* Verifica si la Hashtable está vacía. \*/  {pre: true}  post: Devuelve true si la Hashtable no contiene ningún par clave-valor (es decir, está vacía), y false si contiene al menos un par clave-valor. |
| Método: keys()  /\* Devuelve una colección de todas las claves almacenadas en la Hashtable. \*/  {pre: true}  post: Devuelve una colección que contiene todas las claves (K) presentes en la Hashtable. |
| Método: values()  /\* Devuelve una colección de todos los valores almacenados en la Hashtable. \*/  {pre: true}  post: Devuelve una colección que contiene todos los valores (V) presentes en la Hashtable. |

|  |  |  |
| --- | --- | --- |
| TAD Stack | | |
| Conjunto de elementos:  S= { <T₁>, <T₂>, ..., <Tₙ> } | | |
| inv : {true} | | |
| **Operación** | **Entrada -> Salida** | **Tipo** |
| Stack | NIL -> Stack | Constrcutora |
| top | Stack -> Stack.top | Analizadora |
| isEmpty | Stack -> boolean | Analizadora |
| pop | Stack -> Stack.pop | Modificadora |
| push | Elemento<T> -> Stack | Modificadora |

|  |
| --- |
| **Análisis de métodos STACK** |
| Método: Stack()  /\* Crea una nueva instancia de una pila vacía. \*/  {pre: true}  {post: Stack = < >} |
| Método: top()  /\* Devuelve el elemento en la cima de la pila sin eliminarlo. \*/  pre: true  post: { <Tₙ> } |
| Método: isEmpty()  /\* Verifica si la pila está vacía. \*/  {pre: true}  post: Devuelve true si la pila no contiene elementos (es decir, está vacía), y false si contiene al menos un elemento. |
| Método: pop()  /\* Elimina el elemento en la cima de la pila y lo devuelve. \*/  pre: true  post: {<Tₙ>}, Stack = { <T₁>, <T₂>, ..., <Tₙ(NEW)>} - <Tₙ(LAST)> |
| Método: push(T value)  /\* Agrega un elemento a la cima de la pila. \*/  {pre: true}  post: Después de la inserción, el conjunto T contiene el nuevo elemento value en la cima de la pila. |