|  |  |
| --- | --- |
| **TAD** | Stack |
| **Grafico** | push  pop  peek  3  S =  5  1  9 |
| **Invariantes** | * Sólo se puede agregar un nuevo elemento al final de la pila * Sólo se puede eliminar un elemento al final de la pila |
| **Operaciones** | * push S x Book -> S |
| * pop S -> Book |
| * peek S -> Book |
| * size S -> S |

|  |  |
| --- | --- |
| **TAD** | HashTable  Table |
| **Grafico** | 0  1  2  3  4  5  6 |
| **Invariantes** | * Insertar un elemento en la tabla debe costar en complejidad temporal O(1) * Buscar un elemento en la tabla debe costar en complejidad temporal O(1) * Borrar un elemento en la tabla debe costar en complejidad temporal O(1) |
| **Operaciones** | * insert T x K -> Ninguna |
| * search K -> Elemento |
| * delete K -> Elemento |
| * reHashing Ninguna -> Ninguna |
| * calculateChargeFactor Ninguna -> Ninguna |
| * funtionHash K x i -> Double |
| * codeHash K -> Integer |
| * funtionOne Integer -> Integer |
| * funtionTwo Integer -> Integer |

|  |  |
| --- | --- |
| **TAD** | Queue |
| **Grafico** | Front  Back  **Q=**  Dequeue  Enqueue |
| **Invariantes** | * El primer elemento en entrar es el primero en salir * Solo se pueden ingresar elementos en la ultima posicion |
| **Operaciones** | * Enqueue Q x Book -> Q |
| * Back Q -> Book |
| * Dequeue Q -> Book |
| * Front Q -> Book |

|  |  |
| --- | --- |
| **TAD** | Priority Queue Min Heap |
| **Grafico** | 1  2  3  4  5  **P=** |
| **Invariantes** | * El elemento que está de padre debe ser menor que sus hijos. * El árbol debe de ser semicompleto. |
| **Operaciones** | * ExtractMin P -> P |
| * insert P X E -> P |
| * isEmpty P -> Boolean |
| * Minimum P -> E |