

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE TECNOLOGIAS

DE LA INFORMACIÓN

Informe del método de la ingeniería

Tare integradora 2.

Algoritmos y estructuras de datos

Andres Cuellar Duvan Garcia Gustavo Villada

2020

## Fase 1: Entendimiento del problema.

Problema: El equipo VIP de simulación nos ha contratado para desarrollar un programa que permita realizar las operaciones CRUD (Create, Read, Update y Delete) sobre una base de datos. El programa debe encargarse además de la generación de los datos de las personas.

Requerimientos funcionales:

1. Realizar las 4 operaciones básicas CRUD sobre la base de datos.
   * Resumen: Los usuarios podrán realizar consultas y obtener la información de los distintos datos a través de las funcionalidades CRUD que son crear, leer, actualizar y eliminar.
   * Entradas: Parámetro de búsqueda a escogencia del usuario
   * Salida: Resultados estadísticos de una persona perteneciente a la base de datos.
2. Generación de los datos de las personas.
   * Resumen: El programa debe crear los datos de las personas de acuerdo con las condiciones establecidas.
   * Entradas: Dataset
   * Salida: Conjunto de datos asociados a las personas.
3. Guardar datos de los jugadores en formato CSV.
   * Resumen: Persistir la información asociada a las personas.
   * Entradas: Conjunto de datos
   * Salida: Archivo en formato CSV que contiene la información de las personas

## Fase 2: Búsqueda de información.

* Árbol estructura de datos

En ciencias de la computación y en informática, un árbol es un tipo abstracto de datos (TAD) ampliamente usado que imita la estructura jerárquica de un árbol, con un valor en la raíz y subárboles con un nodo padre, representado como un conjunto de nodos enlazados.

Una estructura de datos de árbol se puede definir de forma recursiva (localmente) como una colección de nodos (a partir de un nodo raíz), donde cada nodo es una estructura de datos con un valor, junto con una lista de referencias a los nodos (los hijos), con la condición de que ninguna referencia esté duplicada ni que ningún nodo apunte a la raíz.

Alternativamente, un árbol se puede definir de manera abstracta en su conjunto como un árbol ordenado, con un valor asignado a cada nodo. Ambas perspectivas son útiles: mientras que un árbol puede ser analizado matemáticamente, realmente es representado como una estructura de datos en la que se trabaja con cada nodo por separado (en lugar de como una lista de nodos y una lista de adyacencia entre nodos, como un grafo).

![Imagen que contiene Diagrama

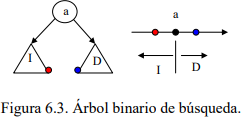
Descripción generada automáticamente](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAYABgAAD/4REiRXhpZgAATU0AKgAAAAgABAE7AAIAAAAdAAAISodpAAQAAAABAAAIaJydAAEAAAA6AAAQ4OocAAcAAAgMAAAAPgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFdpbHNvbiBGcmVkeSBSb21hbiBSb2RyaWd1ZXoAAAAFkAMAAgAAABQAABC2kAQAAgAAABQAABDKkpEAAgAAAAM1MAAAkpIAAgAAAAM1MAAA6hwABwAACAwAAAiqAAAAABzqAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMjAxNjowNDoxMiAwOTo1MToyOQAyMDE2OjA0OjEyIDA5OjUxOjI5AAAAVwBpAGwAcwBvAG4AIABGAHIAZQBkAHkAIABSAG8AbQBhAG4AIABSAG8AZAByAGkAZwB1AGUAegAAAP/bAEMABwUFBgUEBwYFBggHBwgKEQsKCQkKFQ8QDBEYFRoZGBUYFxseJyEbHSUdFxgiLiIlKCkrLCsaIC8zLyoyJyorKv/bAEMBBwgICgkKFAsLFCocGBwqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKv/AABEIAWAB5gMBIgACEQEDEQH/xAAcAAEAAwADAQEAAAAAAAAAAAAABAUGAQIDBwj/xABJEAABBAIABAQCCAIHBAcJAAABAAIDBAURBhIhMRNBUWFxgQcUFSIyQlKRI6EkM2JygpLBFiVD0QgmNFNjorE1RHODssLS4fD/xAAXAQEBAQEAAAAAAAAAAAAAAAAAAQID/8QAKBEBAAIABAUEAgMAAAAAAAAAAAERIVHR8BJBYZGxAjGh4SJxQoHB/9oADAMBAAIRAxEAPwD9IoiICIiAiIgIiICIiAiIgIiICIiAiIgIiICIiAiIgIiICIiAiIgIiICKtwWeocR4917EvfLVEz4myujLRIWHlJbvu3YPUdDpWSAiIgIiICIiAiIgIiICIiAiIgIiICIiAiIgIiICIiAiIgIiICIiAiIgIiICIiAiIgIiICIiAiIgIiICIiAiIgIiICIiAiIgIiIPmtPNcS3+KuK5bHElTF4Hh+5yua6i2R5Z4DXnbyRpo3vsSfXSz2J47z0fFGKqnK5bL4zLVrJFjKYWOnG5zIjI18Og1zgddnA9COq+lN4JxXg8RwzGeaLiOQvuse8aG4hGQzQBA5Wjvs7VPU+i+vHfxlvJcR5vJvxUUkNNliSEMjjfGYyCGxjmPKfxHr0Hl0WKngrnUd6x+Woq7nOfODH2+IeJm/RhwlfwGeFTL5erBXqYuvj4CyeZw2X9W/cY1vU6GgB7r7Dj4rUGMrRZCwLNtkTWzTtYGCR4H3nBo7bOzpYgfRHj4LGInx/EWeoy4egMfVdDLAeWPzdp8LgHO6Aka2AAtzUgdVpwwPnlsujYGGabXPIQNcztADZ79AB7LrMxN1nvejnETUXlver2REWWhERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEQkAbPQBfGIfpS4m4m+lr7F4Tx/j8NQzfV7V3wCR0/E4PB0OvZB9efa8Rzo6L4JpYpWtmaZP6tpPXevPXYFSVFx+Nq4ut4FKIRsJLnddlxPcknqT7lSkBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREHWWWOCF8sz2xxsBc5zjoNA8yVQMy+VzRJwFeKvT/LduNcfE92RjRI9yR8Fxl435viGDDkf0CuwWLnTpId/cj7+2z8loWtDWhrQAANADyQUf2PnDMJTxNYHTRhFWDwyfX8HN/NeT8pmsJ97N14r1Pm+9bpMLXRD1fGSdgeZafktEhAIII2D3QeMUte/TbJC9k9eZnRzTtr2lc1qsFOERVYY4Y29mxtDQP2VDRZ9gcUHHMJFDItdLWj10ikb+No6dAd7+PZaNAREQEREBERAREQEREBERAREQEWbn+kHhWtnJMNPmYG5GOUwvq8ri9rg0P7AdtEde3lva6Yn6R+Fs3d+qY/JP8cwunYyepNB4jG9XOZ4jG84A6/d2pcVY06LGQ/S5wNPyeFn4i2SMyRPMEoZLrW2scW6e/qPuAl2+mlPj+kHhmXh2TOR5FzqMU/1aRwqy+IyXeuQxcvOHdR05fNUaRFmIfpH4Umw1rKNyzWVqcrYZxLBJHKyR34WGJzQ/Z8hy9fJWHD/FOI4ninfhbMk31d4ZMyWvJC+MkbALJGtcOnsgt0REBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQUOE8M8T8QkO3L40IeCew8Ia+HRXyx8mep0+OHywOdJTsMbXt2Gj+FFMD90F3mSOntoLYICIiCk4g2L+GMYBd9dAPXry8p3/AKK7WROYp5T6QIq5seHFjo3CPmJDZ5ndCAezuUDt32tcgIiICIiAiIgIiICIuplYH8he3n1vl31166QdkVfRy8eTdKKkFgNjBAkmiMbXEHWhvr/LSUo8nLFMMq+uzn6RtqhwLB7uJ6n4AIJ7nBo24gDYHU+ZXKhYzFQ4uuYoZJ5uZ3M59iZ0rnH4uJKmoPnGKpZrGWPpKyGLxrxkp7ZkxzpoS0WC2szk5SdczefY9N7WKp1MlluLOF77X8XZWeKvb+0ZspRkhgrSvrOHIxvI0NPN06bHYbJX3xFnh/Hh6RHaKWJqb6zPd8ixGCyUXDX0TRSYu0yTHzh1tjq7ga39GkG5Br7n3iB1110o2UbxLhrfFE2OhylCld4miNq5SpmWZtQ12B8kLS0833gAXAO119F9mRanH1T6s9Y0ZiKiIy+9X5+gwcEuS4luZKtxs7HyWcfNRyr6kjrjJI2yfxg0s2WNPQjl2A4dAvpP0aZDiC/Hlhm7Fq/QhnY3G5C7j/qU9lnLt3NHpvQO6B3KN9VuEVia3+tFnHEREUBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBEUDK5mriY2+OXSTSdIa8Y3JKfQD/XsEEqzZhp1n2LUrIYYxtz3nQAWec6/xXoQOmx2I395+uWa0PQebGn9z7L0q4e3l54r/ABKGgsPNDj2O5o4T5Fx/O737DyWiA0NDoEENuJotxZxzasYqFvIYuXoQqmGtm8DqKny5XHt/BHK/lniHoHdnD03o+60SIKBvFEzoS53DmYbIDrwjHFs/PxNa+a6yszuc3DIwYek7o8tk555B6Ajoz5bPupj+I6EfFcXDz5QL0tY2WM9WA6KtUFceH8WcSzGupxuqsH3WEdj679fdVvhZnh07gL8vjWj+qed2Ivg4/jHsevutGiCHjctTy1fxaMweAdPYRp7D6OaeoKmKpyXD1e7YFyrI+jfaPu2YDpx9nDs4exUSLPW8VI2vxRC2ME6ZfgafBf8A3h3Yfj090GhRcMe2RgfG4Oa4bDmnYIXKAizHHEXF1rEGrwS+lXsygh9q047iH9ka7+5WT+i36PuLcCy1JxxxFZuPM3PDXguPdH7l3be/Tsg+kS5WhDfjoyW4W25BtkHOOcj15e+vddPr1p2U+rMx0n1dv47L3gN/wjqT/JSvq8PjmfwmeKRrn5Rza+K9EECKpfGVdZnyLnVtEMqsiaG/Eu1zE/PS71MTRo2JbFWsyOaY7kkA+874lTEQEREBERAREQdJpWQQvlkOmMaXOPsF0pW4r9GC3X5jFYjbIwuGiWuGx08uhUfLWq1eqyO4x0jLUrK4Y0bJLzr9uqmRxtiiZHG0NYxoa0DyAQdkREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBF1e9kUbnyODGNG3OcdABZx+QvcSyvr4YyU8c08smQI06X1EQ9P7X7eqCVkc+7687F4SMW8iBt+/6uuD5vPr/ZHX4L1xOBZRmdcuSuu5GQaksyDqB+lo/K32Cl47GVcVUFelEGMHUnuXH1J8ypaAiIgIiIPj9/gzPWP+ktVz9fJQCpFTEkkR3zCLqzl126uHf2X2BVDa9H/bR9kWT9fOPbGa+ugi8QkP/wA2wrdAREQF1kjZNG6OVjXscNFrhsFdkQZ1+Du4d5m4ZmaIt7fj5yTER58h/If5eym4ziCtkJTWlY+neYPv1Zxp3xHk4e4VqoOTw9PLQhlyLbmnccjTyvjPq1w6goJyLNizl+Hjq+H5THN6CxG3+PEP7Y/MPcdfirylfq5Gq2xSnZNE7s5p7ex9D7IJCLh72xsL5HBrWjZc46ACo38V1ZJOTF1LmT07RfVi+4PfmcQCPhtBeoqJ/E7q7x9dwuTgi1szeE17W/HlcT+wVpRyNTJV/Hozsnj7EtPY+hHcH2KCSiIgIiICIiCvmtVps7Dj5K5kljiNlshALY9ENHzO+nwKsFX4y7FkJrk0dfwzDMa4lJ34gaAdj22SPkrBAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQF52LENStJYtSNihiaXPe86DQPNeizmTj+3eJosVK0mhSjbZsg/hlkJPIw+w0XEe7UCLJZjPjxMRG3HUTvltWWc0ko/U1nYDzHN+wUj7DyRhO+Ibnja6SCOPQP93WldgBoAA0B2ARBnpLmbwbfEyLW5Sk0ffmgj5ZmD1LB0d8tKwsZ7G1sXHkH2mGCbXhFh2ZSezWjzPsrFZKDEVcRxwwfV2OrXI3y1tjfgTA7eG+QDtg/HaCSzHXeI5RPnGurUGu5oceD1f6OlPn/d7LRsY2NgZG0Na0aAA0AuUQEREBERAREQUjW0/9v5HiSX699mNBj5RyeH4rtHffe9/JXajDH1RlTkhF/SzCIDJzH8AcXa1vXcnrrakoCIiAiIgIiICpL3Do+suvYWc4+8ernMG45fZ7ex+PdXahZmZ9fCXJYjyyNhdyn0OuhQZvGi/xhI85qNkOPpyGIxQvJZckb3f/AHAewWvjjZDG2OJjWMaNBrRoBQsHVFLA0oBrbYW8xA7uI2T8ySp6AqHLYh9WR+XwbRFejHNJE3o2y0d2uHr6FXyIIuMyEOVxsF2vsMmbvld3afNp9wdhSlR8PMFW/l6TT9yO14rG/pDxvQ+YP7q8QEREBeNyyKlKaw4cwiYXa9dDsvZQMrcmq/VWV4BM+xYZE4OB01hP3nfIbQe2OldPja88kAryTRNkfEPyOI2QpKIgIiICIiAiIgIiICIiAiIgIiICIiAiIgIiICIiAiIgIiICz+BcDxPxMHc/iNtxA83bX1ePWvb/AF2tAs7kZDhOJosm8kUbsba1k+UUgP3Hn2Oy0n2ag0SIDsbHZEBZ7ifxftThzwAeb7THPr/u/Ck38t8q0KzlWX7d4sNuLmNHGNdDE/8ALNK7XMR6hoAG/XaDQyyxwxukme2NjRsuedAfNVuN4lwuYuz08VlKtuxX14scMgcW/suOIeGsXxTjTRzVfx4D2AeWkH2IWK4c+gjg7hzKWrkNea0JxpsU8hLYx7a6n5oPpSKkl4N4fmx0FGXGROq13OdFGXO00u7nv5qS7h7FPioRupMLMc4PqDZ/hEa0R19h3QWSps/xbg+FjVGevspm25za7XMc4yuaNkANB2evQdz2G1crFcWULdn6SeBrMFSaavVnuOnlZGXMhDoNNLiOjdnoN+aLCyh4/wCGJ+HLGdZlo24+rL4Mz5Y3xvjk2ByGNzQ8O6jTdbOxpc0ePeGchib+SgysbK2OG7hsRvgfX6bHMyQBw35dOvltfOMzhMtBxZmc03EXbNKjxTTyD4Y4HF1iFtUMc+Nuv4ha470N/hPmFG4spZHi/OZTijCYfIvxVWLHsfBNUfDLkfBs+LJyRPAc4NadDY6nYG1IxiOtf1hE33muSZ7vGYrti2OM+k6jmPpBmoUrbG4WvhnXp5rNaSu6N4lA2fEDTy8p3vWvdXeJ+kThXNyTsoZZnNBAbL/rEMkG4R3kb4jW8zB+pux7r5lxjWyP0g8QZqfh3EZVtd/DRrxTXKMlZliRtlkhib4gHUtBHXW/h1VnxDPDx7l8TJX4dzf2disfc+02PoSV5C2WHw/q0YeBzv3103Y+6Oqv8Yzx8+r63S4cXbxH3u29wXH3DnEuQ+pYa9JNOYjMxslSaISMBALmOewBw6jq0nutGvlf0eXM1HxTDjKGQzWV4ahou5n5vFGrLTkaWhkYkLGGUkb30Otd19UVmI5MxM80a5kaWOa11+3BWDzppmkDOY+g33XjkM3QxcUcl2csZLrkLY3P3/lBUmepWtcv1qvFNyHbfEYHaPtteoaBrQHTt07KKhXcq2m2Itq2rPi9W+BHza+OyNLjIXrtZrfqOOdbc5u9eIGBp9Cp6IPi3D3FX0yZLi25Ws8O0oaEc5AfbHI2Nu+wc3q7ovrt2tNdws9aQtE0sLm7aOgcR5b91MRBVcNXPrnD9UuO5oWCGYHuHs+6dj5b+atVmsnWuYDJy5rGAS0pRzXqnYkj/iM99dx5qyxnEeKy8LH07kZLmh3hyHkeP8J6oLNcOcGtLnHQA2SfJR7ORpU4/EtW4YWer5AFQWLdni0mnjWzVsS7pPdILHSjf4IwevXzcg8+Hs1XOQty23eCclZc6o9zdNkY37oG/XYPRaxQrOIo28YMfPXY6s1oa1mvwgdteipxbv8ADEhjybnXMT0EdsDckHtIPMf2v3QaVF0iljnhbLC9skbxtrmnYIXdAUD6zadxB9WbEBTZXL3SEdTIXDQB9Nb/AJKbJIyGN0kr2sY0bc5x0APiqvh7JuzFOe4JY5a77D21nRkEGMaHcd+ocgtkREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAVbn71GjiJTkmCaOUeG2vrbp3Hsxo8yf/2pV+/XxlKS3dkEcUY6n1PkB6knppU+Kx0+Svtzmaj5Jdf0Sq7qKzT5/wB8+Z+SCBh6fFGGotIMN+F5Lm0pH8r67T2Y2T82h6q1+3Lv4Tgrniexby/5ldIgzklTO5z+HkCzFUXD78MEnPNJ6gv7AH26r3ZZOEzdbGmJseOsx8tVzW6Ecje7CfcdR8CrxQczjG5fFy1S/wAOQ/eilA6xvHVrh8CgnIqvAZN2RoFtkBl2s7wbMf6Xjz+B7j2KtEBERAREQEREBERARecpka5rmuYI27MgLSSR7HfT9iutS3BerNsVJWyxO7OaUHsiIgIiICIuksrIIXyyuDWMaXOcfIBBR8Qk5K5UwUTtCc+La13ELT2+Z0FZWcPjrmvrNOGQgaBLOuviq7hqI2zZzc7SJL7v4Qd3ZC3o0fPv8wr5BUu4YxHgyMjpRMc9pAeBst9xvzXnwxblkxrqVwk26D/q8pd3cB+F3zGvntXSz+S1iOJKuTBIr3NVbPoHfkcfn0+aDQLhzWvaWvAc0jRBHdcogzk2Mu4Cw63gB41M7M2OJ6f3oz5H27FW+MytXL1fHpv3o6fG4afGfRw8ipipcpgTNabkcTOaWQZ+do+5MP0vHmPfuEFb9KNOa/8ARfnq9Vr3TPpvDGx/iJ15KN9EvBs3A/0e0sXcldJZduaYE7EbnflHwV7i8427M6jkofqeQYPvQPOxIP1MP5grhAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQF52bMNSrJYsyNjhiaXPe49GgLu97Y2Oe9wa1o2SToALMQtl4uvtsygswdd24IyNG28b++4foHkPPv6IO9GpNxHkY8vkmOjpQndGo783/ivH6vQeQ91pUA0NDoEQEREBERBncw04TMRZuBv9HmLYb4Hk3s2T5HofY+y0LXBzQ5p2CNgjzXWeGOzXkgnYHxyNLXtPYg9wqPh2d9CxNw/ckc+ao0PryPOzLCT9078yOx+CC/REQEREBERAREQFVWXV+Hq7p61Nxhlm5p/C/Jvu7X/JWqd+6Dhrg5oc3qCNgrlZniDiiHg2Z9/iO3FDhpS1kchGjE/9PuCr3HZGrlsbBfx0zZ6thgfFI3s4HzQSUREBZ/iWaS5NUwVbfPefudw/JC3q4+2+wV+5zWMLnkNa0bJPkFQ8OsfetW83OP8AtTvDrA/lhaen7nr+yC9jjbFG2ONoaxgDWtHYALsiICiZSgzKYuxTl6CVhaD+k+R/dS0QVPDd+S9iWstu3cquMFn++3z+Y0fmrZZ+x/ufiyKyNNq5QCGY/plH4D8+o/ZaBAREQQMthqmZrtjtNIfG7mimjPK+Jw7FpVZBl7WFnFPiNwMJPLDkANMf6B/6Xe/YrRLys1obdd8FqJksUg05jxsEIPUHY2OoRZoxXuFXF1YSXsOBsw/ilrD+z+pvt3V7RvVslTjtUZmzQyDbXtP/APaQSEREBERAREQEREBERAREQEREBERAREQEREBEXUvaHaLm79NoOydu6LN37cvEN+TD4yRzKkJ5b9ph/eJh/V6nyHug85ZX8XXpKkHM3C138s8o6fW3jR5Gn9A8z5notMxjY42sjaGtaNAAaAC6Va0NOrHXqxtihiaGsY0aAAXqgIiICIiAiIgKl4jpTvgiyONbu9Rdzxt/7xn52fMdvcBXSII9C7DkaENus7mjlaHD29lIWcra4e4jNPRbQyb3SwH8sU3dzPbfcfNaNAREQEREBERAREQUHGXBuK454ffic3G50LjzNcw6cxw7EKD9HXCl7gnh12CtXBdq15Sakx6OEZ68pHstavOxPHVryTzvDI42lznHyAQd3ODWlziAB1JJ7KlucZcP0SBYycQ2dAsDnj92gqJXpT8UuZfyhlhx3evSDuXxB5Pk9d+QWgrU61KIRU68UDB2bGwNH8kGYy3EePzlatisPkGOlyT+TY21wjHV50dHstTBDHWrxwQtDY42hjWjyAGgouQwuOyjdXakcjvyya09vwcOoVVVtW+H8pDjclK+zRsu5att/VzHf928/wDoUGjREQEREEDN40ZbDz1ez3Dmjd+l46tP7rpw/kn5XCwzzt8Oy3cdhmtcsjejv+Y9iFYve2Nhe9wa1o2ST0AWOp3bj+ILs/DFP61St6dJNM7w4myjpzN83bHQ69Ag2SKh/wCtgY1xGIc7f3mAyAa9jrv8VzBxG6G0ypnaT8dLIeWOQuDopD7OHY+xQXqIiAqG5hLFG47I8OubFM47mqO6RWP/AMXe6vkQVmHzlfLse1rH17UR1NVmGnxn4eY9CFZqry2DhyTo7EUjqt6HrFZj/E32PqPYqtZxTJjZHUc/Vey+3+pFdvOLY9WDyPqD2QaZFQC1xNcLX1qNKlET1FqRzn69dN6A+xQ3OJqRc+1j6d6IOGhTlLXhvmSHd/gEF+igYvM08vG81HkSRHUsMg5ZIj6Ob5KegIiICIiAiIgIiICIiAiIgKNkMhXxdGS3cfyxRjrobJPYADzJPTSkrP2ohl+M4a0w5q2Khba5T2dM8ua3/K1rjr+2D5IOsVDK5wixlbEuPqu6spV3aeR/4j/9AvccHYMQOi+pEtdve5pNnffrzbV2iDLZPhvJVcfM3hfKTV3Obr6vM7nbrz5CerXa7Hr1U/hSag/CMgx1d1T6u4xzVn/jjk/NzepJ67897V0s/kWjFcWY+/C0hmScadkNHQuDS+N5+Aa5u/ceiDQIiICIiAiIgIiICIq/OZF2Lw81mNvPN0ZE39T3HTR8Nn9kFXxVbiuV34enE+zknASRCLX9HcPwyOPYAH91xHj+LbNSL6xm6dCZo++IKnjB3zcQrPB4luKo6efEtTHxLEzvxSPPc/D0CskGf8LifHxB31mrl+X8TXReA93w1sfup+JzdbLNexgfDZh6TVpRp8Z9x6e6sVQ8SU3QMbm6DQ25SHM/Q/rYvzMPr02R7hBfIvOvOy1ViniO2SMDm/AheiAiIgIiICouKYzdho4z/h3LLWyjfeNv3iP5BXqo+I5BTnxd9/8AVw2gyR2/wteNbP8AJBdgBrQ1o0ANADyXKIgKt4hotyGBtQkbeGGSMg6Ie3qCPmFZKDmrbaOEuWHkDkidrfqRofzQd8Taddw9SzJ0fJC1zwPJ2uv81LUHCQSVsFSimHLI2FvOPRxGyP3U5AREQZzJNdn879kB7m0arRJd5TrxCfwx/DoSfgtDHGyKNscTGsY0aa1o0AFS8MsHiZeUuLnyZB+yfIBrQB8P+avEBeFynXv1JK1yJssMg05rgvdEFDgJpqN2xgrkhkdWaJKsjjt0kJ6dfUtPT5hXypL7Qzi7EytA55GTROdrry8vNr9wFdoCIiCPkLsWNx09yweWKBhe4+wVXw/jJAHZfKN5slbHMebr4DD2jb6aHf1O104ye5uGrxBnOyxfrQSg9uR8rWuP7FX6AiIgz/EOLfHI3OYpvJkag5nBvQWYx3jd69Ox8irmjchyGPr3KrueGxG2WN3q0jYXuRtpB7FUXBxP+zvJ4gkbFbsxscP0id4A+Q6fJBeoiICIiAiIgIiICIiAiIgKkoStZxlmK7j/ABHQ15wP7BDm/wDqxyu1Q8Qw2adqtnMfC+eWoDHYgZ3lgcRzaHm5pHMB/eHmgvkUejerZKnHapTNmhkG2vadqQgKk4kkaJMRBrck+QYGADtytc4n9mn91cyyxwROlme1kbBtznHQAWexMjuIM2c0WObQrNdDRDhrxdn78uvQ6Ab7bPmg0aIiAiIgIiICIiAs7xa97bHDzWkhkmXjbIB5t8KU9fmAtEqviKlNew0gqf8AaYXNmh93NO9fMbHzQWiKHicnDlsbFbrno8ac092OHdp9wVMQF1la18T2yfgLSHb9F2VLxNfdXxv1Kp969e/gQMb3G+hd8ANlB58DyTS8EYt9nYlMP3tj3KvlHoVG0cfBVj/DDGGb9dBSEBERAREQFGyFGDJ4+anabzRTN5XD09/l3UlEGexeWkxsrMRn3hk7fu17LujLLR2O/J3qFoQd9l4W6Va/XMFyBk8R7te3YVNLwjFyhtHLZbHxjsytZGv/ADNcUF7NPFXidLPI2ONo2XPOgFmhJLxbk4jGxzMJVk5y9w19bkHYAfpB6781Li4Roc0bshNbyb4/wuuzc/7tADT8wrxrQxoa0BrQNAAdkHKIiAiIgz2O1iOK79KQBkWSd9arnr95+tPHx6NOvitCoGXxMOXqCKVzo5GO54ZmHTonjs4KpbxDcw9iOjxBVfI5wPh26redsgHclo6t9/JBpUVA3jfh987oI77ZLDe8LWOLx8tKJJl8hxDc+zsdG/GQuYXyzWRyTGPetsZ3HXXU+qCVSnGX4usWYTzVcbGa4cOzpnaLtfAdP8S0CjY+hBjKMVSozlijGhvqSfMk+ZKkoCIiCs4iozZDA2oKjuSxy88LiN6e3q0/uF7YfJxZfFQ3IRy8409h7xvHRzT7g7Cmqht4u7jr8mRwHI4zHms0pDpsp/U0/ldr5FBfIqD/AGxx9eJzstHYxjmHTxaiIHyI6Ee4QcYULUbXYeOxk3POmfVoiW/Nx6Ae5QWGayceJxM1qTbnAcsbG9XPeejWgeZJXXAY92LwVWrLy+M1pfMW9jI4lzyP8TioVHFXb2QiynEBYJIetalGdsgP6ifzO18h5K+QEREBERAREQEREBERAREQEREFHPw2IrEtrBWnYyxKS6RrGB0Ujj3c5nr7gjv5rtycSgcolxzv7ZDwfjrSukQUA4cnvyCTiHIOvMadirGzw4f8Q7u+fT2V81rWNDWANaBoADoFyiAiIgIiICIiAiIgIi4c4NaXOOgBsk+SDJ5WC5Q4qgHDBjbZtNdNbrynULmjpzHXVriem/PSmycT2KcY+0cDk2yb0RVgNgfHbNrnhppvS3M5IOt1/LBv8sLejf36u+av0GfGfyN+EfZGEtNc7pz32+A1nuQfvH5BRuF6z35bIz5p7ZszBJ4T3j8LYj1byDyBC1Kz+ZH2VnqWab0ik1UtdenKT9x3yd0+YQaBERAREQEREBERAREQEREBERAREQFQs/pXHsrmybFKkGOj8g6R2wf2aQr5UXDo8fIZm86MNMtvwmu/WxjRo/u5yC9Wf4jH2fbo5xgA+qv8Owdd4X9HfIHTvktAvG3Wju05q0w2yVhY4fFB7A77IqXhe1JLiTUsuJs4+Q1Zd9zy/hPzaQf3V0gIiICIiAiIgh0ctSyU1qGpO2SWnL4M8YP3o3aB0R8CCpi+M1OG+KsF/wBJGbKxOEWAzWw8h22yFkW9OHk7bTor7MgIiICIiAiIgIiICIiAiIgIiICIiAiIgIiICIiAiIgIiICo+KZXyUIsVXJE+Sf4GwerY+73f5em/UhXiz+L/wB68TXcoesNTdOt6bB/iOHz6f4QgvK8DK1aOCIaZG0NaPYL0REBRshSjyOOnpz/AIJmFpI7j0I9weqkogp+Gbs1rEiG7oXKbjXnAP5m9N/AjqrhZ62RhuLYLW+WtlAIJewAmA+4fiRsfJaFAREQEREBERAREQEREBERAREQeVmYVqks7ujY2F5+Q2qvhKIR8L03hxf44dPzHuedxcP5ELniyeWDhi4azQ+aRojjaTrZcdaVpXgbWqxQR9GRMDG/ADSD0REQZ+cfZPGcM4BEGVZ4Eh66bKwEsPoNjmHudLQKr4ix7sjhZo4Ty2I9TQO/TI07af3CkYnINyuJr3WDl8ZgLm/od2c35HY+SCYiIgIiICIiCNkXzxY6eWlEyWzGwuiY/sXa6Bd6dkXKMFlrHME0bZA1405uxvRHqvZQqTr/ANdususb4Aka6tI3XVhaNtI3vYIPX3CCaiIgIiICIiAiIgIiICIiAiIgIiICIiAiIgIiICIiAiIgrOIcg/HYaV9frZlIhrj1kd0B+Xf4Ar2w+OjxOIrUYtkQsDSSdknzOz7qsd/vfjIN/FWxDNn0M7x/9rf/AKitAgIiICIiCBm8d9q4eeqDyyOHNE/9Lx1af3C4wWR+1MPDO8FswHJMw92vHRwPzVgs9A37H4wlh2RVyrfFjGujZm/iHzGj8doNCiIgIiICIiAiIgIiICIiAiIgoeIiye/hqDiQZ7glAHmIwXn5dFfKjfz2OOom/dMNSk5x9Q9zgB/5dq8QEREBZ/DkYziK/hyQI5v6bWb7E6kAHoHaP+JaBUHFIfTjqZmBu34+UOlA/NC7o8e/Q7A9QEF+i4a5r2BzCHNcNgjzC5QEREBERAVdhs7Qz0VuTGyOkbUtyU5uZhbqWM6cOvfr5qxXwE07mP4E444ox2dytS5jM/dlrQQWiyvts43zxjo/m3o82/bSR7zeX+xqtTNVzmvidH35Qzlabc03Emb+nOgNgRch6xh3KXb1rue29r5W5+T4nx/GfEE/EeUxdnB2Z4KFepaMUEAhia8OfH2k5iSTz76HQ0vBnFU9XiBuYzWWylfH3+DftS3WinJFeUljeaFh6Nd16e5T95X8TMeCIv23jET5fZ0Xwqna4xx3FVOhiW5rHfbOJuGqzPZcXi+ZjGujk5duERBcNjeuvbotD9Hlt9TiiDF553FtHOyUnPkq5i79Zq2i0t53xODnN2Cew5Rp3ZWsd9dGbwvfLV9UREUUREQEREBERAREQEREBERAREQEREBEVdmMuzFxRtZE6xbsO5K9dn4pHf6AeZ8kFiomRyEePxti2774hjLuRvdx8gPcnoqlvD9vKRl3EeQmdz/+6UpXQxs671zN09x99gH0XtLwhhpYgzwZ2a0Q6O3K137h2z80Hrw3j34/Ds+snmtWHGxYds9XvOz8uvZWyzMzMvw1zWY7E2XxjeskMjQbEI9Wka5wB5Eb6dytBUtQ3akVmrIJIZWhzHDzBQeyIiAiIgKp4koSXcQ59Ubt1XCxX0OvO3rofEbHzVsqbK5idtwYzDRNnyD28zi8/wAOBv6nkfyHmgnYzIRZPFV7sR0yZgdo+R8x8ipQcHfhIPwKzlPgurHH/vG3btvLy8tbM6GNpPfTGEdPjtSJeEcY4h9V1unK0EMkr2pBy+/KSWn5goLxFnG38jw/YjhzkouUJHckd9rOV0ZPYSgdOv6hoewWjB2NhAREQEREBERAREQERdZJBFE+R3RrGlx+AQUeB5bOdzl5vNszsrbIP/Db5e331fKm4Uje3h+KWbRksSPmcR58zjr/AMulcoCIiAuk0LLEEkMo2yRpa4exXdEFHwtM9lCbFzuJnxspgO+5j7sP+XQ36tKvFnr4+yuLqd9rNQ5Bv1Oy4aGnDrG4+vXbf8S0KAiIgIozr8GphDI2eWEEuhjcC/4aXTFzXbNMTZKs2rK4kthD+Ysb5Bx7c3rrfxQTFRv4MwMmDyWHfQ3Qys0k9yHxpP4r5DzPO+bY2fIED0V4iDK5X6NuGMzkZbl2lNz2AwWoobcsUVoMGm+LG1wbJrX5gfQ7VbBwdfzfGl7KcU0MfBizinYiChBO6bxYnScxe88jQ3oAA0b169FvETffDwMdB9FXCUMgllpWrcza8lVstvIWJnNhe3lMYLnnTddAPLy6qfguBcJw9kTfpMtzW/C8Bk127NZdFHsHkYZHHlbsDt30tEitz7pUCIiiiIiAiIgIiICIiAiIgIiICIiAiIgLO4Jn2nnMlmZtu5JnU6oc0jw2RnlcR/eeCd+Y16LRKl4Ve04iSMDTobU0Txr8weQUF0iIgLP4iP7K4jv4pg1VlaLddvkzmJD2j/ECfba0Co5JGycdQxMd9+GiXyAejn6bv5tKC8REQEREHhetsoY+xbl/BBE6R3wA2q3hek6viW2rGnW7x+sTya/EXdQPgBoBe/EQceGcjygkiu92mjZOhvSkYyVs2JqSR/hdCwj9gglIiIPK1Vhu1Ja1lgfFK0se0+YKqeFppvs2WjbLnS0JnV+d3d7R+F3zCu1S4Nwly2ZmjILDZEfT9TW6KC6REQEREBERARFCzGRbisPZuv0fCYS0Hzd2A/dBFyuebStR0KVd97IytLmV4+zR+p7uzW+5+SivxOfyUE0eTy0FWKZnKYKcGywEaI8Rx6/5QpXD2IOMouls/fyFs+LblJ2S8/lB/SOwH/NW6DOwYjOYenDBichXtQV4xGytbiLCQBofxGk66D9JU3E5xuQkfVtV5KN+Ibkqykb1+ppHRzfcfPStVT8R419uiLVI+HkKf8WvIB12O7T6gjoQguEUXGXmZPF17kbS0TMDuU92nzB+B2FKQFFyORrYqm6zdk5I29B02XHyAHmT6KUs7Qi+3OJLGSsferY+QwU4yOnia+/J7nroenVBHtVc7xTTMc0UOGpOc1zBM3xbB0dg6BDWHoCOpPqAp8mJzTY3Grn3eKR0+sVWvYD8AWn+au0QZ4Z+3irMdfiavHDHK7kjvQEmFx8g7fVhPlvp5b2rHOy5KPB2XYKuyxfLCIGPeGt5j2JJ8h3Uu1Vgu1ZK1qJssMrS17HDYcCqXhqSepNdwduQyPoOaYJHElz4H75Nn1BDm/AD1QYj6L/ohv8AB3Et3iXiHNNyOUvMeJGRsPI0ucCTzHuemuy+rIiAiIgIiICIiAiIgIiICIiAiIgIiICIiAiIgIiICIiAs3Zl/wBm+IpLcoIxeSLfFk8q8/YE+jXDXX1HXutIuk0MdiF8M8bZIpGlr2PGw4HuCEHYEOaC07B7ELlZ9nD97GOH+z2U8GvzbNO5GZo2j0Y7Yc35lwHkF6vHFEkbmMdiq7iOku5JdH15NN3/AJkFjkcjVxVGS3elEcTB383HyAHmT2AVdw7Vsu+s5XIxmK1feHeE49YYwNMZ8ddT7krmpw4PrbLuZtyZO3GdxmRobFCfVkY6D4kk+6ukBERAREQcOaHNLXDYI0QfNZvBzfYN13D155DOZz8fK8nUkZO+TZPVzf8A06rSqLkcbUytQ1r8IljJ2OpBafIgjqD7hBKRUMeMz2Ody0MtFcrAabFfiPiD/wCa09R8Wk+67vh4lst5TZx1Eb6viY+d2vbfKAf3QeuezP2XWbFWjNnIWPuVqzfxPd6n0aO5K9sJjfsnExVnO8SXq+aT9bz1JXni+H6uMnfaLpLd6RvLJcsEOkcPQaADR7AAK0QEREBERAREQFS8Uf8As6uHfgNuIP8ATXN5q6UDOY77Vwtmm13K+Rn3HA604dR1+KCeirMDlhlca18mmWoT4VmLzjkHcf6j2Ks0BEVVxDlPs7HFkAMlyyfBrRN7vef9B3JQRuDfCHDbBXO4hYnDNHf/ABXb6/HavlDxNAYzEVqYdzGJgDnfqd3cfmSSpiDh2+U6766Kj4MMR4XhMLi4GafmJOzzeM/f89q9WexcpxHEFvEWPuw2XmzSdrod/jZ8Qevvs+iDQoiICoiXj6QGBnL4ZxrzJ1683iM5f5FyupZWQQvlmeGRsBc5zjoADzVHw445KxbzxBEVzlZV35wt3p3+Ikn4cpQX6IiAiIgIiICIiAiIgIiICIiAiIgIiICIiAiIgIiICIiAiIgIiICIiAiIgIiICIiAiIgIiICIiAiIgIiICIiCkyXD7n5I5bC2BRyRaGSOLeaOw0dmyN2N68nDqP5LiPM5WB3h5LAWNjvNTkZLGfh1D/3arxEFFJmcvYf4WMwE7SR/X3ZWRRj5Al5/y/NeuLwTq952Syln69knNLRLy8rIWn8sbfIepOyfXyVwiAiIgKHlMXXy1PwLQcNHmjkYdPicOzmnyIUxEGeZZ4ixDPDt0hm4W6DZ6jmRzEer2PIb09Wu6/pC9pOILXJ/RuH8nPJ+gtjj18S54Cu0QZ12IyWdkaeInRQUmu5hjqzy4Sf/ABX6Gx/ZA16krQgBrQGgAAaAHkuUQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERAREQEREBERB/9k=)

* Árbol binario

Cada nodo puede tener: un hijo izquierdo, o un hijo derecho, o ambos o sin hijos. A lo más cada nodo puede tener dos hijos. Un árbol binario está formado por un nodo raíz y un subárbol izquierdo I y un subárbol derecho D. Donde I y D son árboles binarios. Los subárboles se suelen representar gráficamente como triángulos.

* Árbol binario de búsqueda

Para cada nodo de un árbol binario de búsqueda debe cumplirse la propiedad: Las claves de los nodos del subárbol izquierdo deben ser menores que la clave de la raíz. Las claves de los nodos del subárbol derecho deben ser mayores que la clave de la raíz.



Esta definición no acepta elementos con claves duplicadas.

Se indican en el diagrama de la Figura 6.3, el descendiente del subárbol izquierdo con mayor clave y el descendiente del subárbol derecho con menor valor de clave; los cuales son el antecesor y sucesor de la raíz. El siguiente árbol no es binario de búsqueda, ya que el nodo con clave 2, ubicado en el subárbol derecho de la raíz, tiene clave menor que ésta.

Tomado de: www2.elo.utfsm.cl/~lsb/elo320/clases/c6.pdf

* Árbol AVL

Un árbol AVL es un árbol binario de búsqueda que cumple con la condición de que la diferencia entre las alturas de los subárboles de cada uno de sus nodos es, como mucho 1. La denominación de árbol AVL viene dada por los creadores de tal estructura (Adelson-Velskii y Landis). Recordamos que un árbol binario de búsqueda es un árbol binario en el cual cada nodo cumple con que todos los nodos de su subárbol izquierdo son menores que la raíz y todos los nodos del subárbol derecho son mayores que la raíz. Recordamos también que el tiempo de las

operaciones sobre un árbol binario de búsqueda son O(log n) promedio, pero el peor caso es O(n), donde n es el número de elementos. La propiedad de equilibrio que debe cumplir un árbol para ser AVL asegura que la profundidad del árbol sea O(log(n)), por lo que las operaciones sobre estas estructuras no deberán recorrer mucho para hallar el elemento deseado. Como se verá, el tiempo de ejecución de las operaciones sobre estos árboles es, a lo sumo O(log(n)) en el peor caso, donde n es la cantidad de elementos del árbol. Sin embargo, y como era de esperarse, esta misma propiedad de equilibrio de los árboles AVL implica una dificultad a la hora de insertar o eliminar elementos: estas operaciones pueden no conservar dicha propiedad.

* Árbol Rojinegro
  + Árbol binario estricto (los nodos nulos se tienen en cuenta en la definición de las operaciones  todo nodo hoja es nulo)
* Cada nodo tiene estado rojo o negro
* Nodos hoja (nulos) son negros
* La raíz es negra (esta condición se impone para simplificar algunas operaciones) Se cumplen las condiciones:

1. Un nodo rojo tiene dos hijos negros.
2. Todo camino de la raíz a cualquier hoja pasa por el mismo número de nodos negros.

Tomado de: [https://www.infor.uva.es/~cvaca/asigs/doceda/rojonegro.pdf](https://www.infor.uva.es/%7Ecvaca/asigs/doceda/rojonegro.pdf)

## Fase 3: Búsqueda de soluciones creativas.

Existen numerosas estructuras de datos que se encargan de almacenar información de una manera particular, haciendo que cada una de ellas tenga diferentes características que pueden adaptarse de mejor manera a cierto tipo de problemas dependiendo de las condiciones y necesidades que proporcione el mismo.

Teniendo en cuenta lo anterior, se procederá a hacer uso de una técnica conocida como lluvia de ideas para enlistar algunas de las opciones que a primera vista son factibles para resolver de manera pertinente el problema abordado:

**Alternativa 1: Tabla Hash**

Esta es una estructura de datos que se caracteriza porque funciones como buscar o insertar son notablemente rápidos, haciendo que sea una excelente alternativa para solucionar algunos aspectos del problema ya que se requiere un nivel de eficiencia aceptable en los algoritmos y estructuras usadas en la solución del problema.

**Alternativa 2: Montículos**

**“**La estructura de datos conocida como montículo es un arreglo de objetos que podemos ver como un árbol binario casi completo. Cada nodo del árbol corresponde a un elemento del arreglo” (Thomas H. Cormen, 2009)**.** Esta estructura de datos también podría incluirse en el grupo de candidatos a suplir la funcionalidad de almacenar los precios de las divisas, aunque dada la gran cantidad de datos a almacenar, puede resultar ineficiente el uso de esta estructura.

**Alternativa 3: Árbol Binario**

“Un árbol binario es un conjunto finito de elementos, el cual está vacío o dividido en tres subconjuntos separados: El primer subconjunto contiene un elemento único llamado raíz del árbol. El segundo subconjunto es en sí mismo un árbol binario y se le conoce como subárbol izquierdo del árbol original. El tercer subconjunto es también un árbol binario y se le conoce como subárbol derecho del árbol original“ (Serrano, 2013). Este tipo de estructura es una fuerte candidata debido a que sus operaciones de búsqueda, eliminación e inserción son notablemente rápidas. La desventaja de esta estructura de datos es que no es auto-balanceada y ello puede traer problemas en la eficiencia de las operaciones elementales.

**Alternativa 4: Árbol n-ario**

“Un árbol n-ario es una estructura recursiva, en la cual cada elemento tiene un número cualquiera de árboles n-arios asociados. Estos árboles corresponden a la generalización de un árbol binario. La diferencia radica en que esta estructura puede manejar múltiples subárboles asociados a cada elemento, y no solamente 2, como en el caso de los árboles binarios.” (Blanco, 2017). Esta estructura de datos es poco viable para el problema que se desea resolver, debido a que esta estructura no permite mantener un criterio de orden determinado y esa es la principal condición del software a crear.

**Alternativa 5: Arboles Rojinegros.**

“Un árbol rojo-negro es un árbol de búsqueda binario con un bit extra de almacenamiento por nodo: Su color, que puede ser rojo o negro” (Thomas H. Cormen, 2009). Esta es una estructura de datos idéntica a un árbol binario pero con ciertas características adicionales que le permiten tener la facultad de auto-balancearse por sí solos. Esta es una alternativa muy viable debido a que puede soportar las dos principales condiciones del problema en cuestión: Un criterio especifico de orden y una notable eficiencia en las operaciones de los elementos de la estructura.

**Alternativa 6: Arboles AVL.**

“Un árbol AVL es un árbol binario de búsqueda con altura equilibrada: Para cada nodo x, las alturas de los subárboles izquierdos y derechos de x difieren como máximo 1.” (Thomas H. Cormen, 2009). Esta también resulta una alternativa a tener en cuenta de cara a la resolución del problema en cuestión debido a su condición de auto-balanceo, permitiendo que la estructura se mantenga equilibrada y que sea posible mantener una notoria eficiencia en las operaciones principales que manipulan los elementos almacenados.

**Fase 4: Transición de la formulación de ideas a los diseños preliminares**

A continuación, se evaluarán individualmente cada una de las alternativas planteadas en el inciso anterior y se procederá a descartar las propuestas menos factibles teniendo en cuenta los requerimientos del problema abordado, es decir, sus funcionalidades y condiciones de eficiencia. Para ello se procederá a aplicar la técnica de revisión selectiva que consiste en describir los aspectos relevantes de la información abordada dividiéndolos en dos secciones conocidas como ventajas y desventajas:

|  |  |  |
| --- | --- | --- |
| **Alternativa** | **Ventajas** | **Desventajas** |
| 1. **Tabla Hash** | Los algoritmos que manipulan este tipo de estructura son muy eficientes dado que tienen una complejidad temporal O(1). | El proceso de redimensionamiento de la tabla resulta una maniobra costosa de implementar. Además, no es posible establecer un criterio de ordenamiento de los elementos almacenados. |
| 1. **Montículo** | Los objetos guardados pueden ser organizados de manera ascendente o descendente. La ordenación se realiza con una complejidad temporal relativamente óptima. | La complejidad temporal puede llegar a ser ineficiente si se trata de un número grande de elementos o nodos, hecho que hace imposible la implementación de esta estructura en el problema en cuestión. |
| 1. **Árbol Binario** | Su funcionamiento puede emparejarse directamente con situaciones de la vida real. La información puede guardarse de manera ordenada. Puede redimensionarse sin mayor problema. | Si un árbol no está equilibrado, sus funcionalidades pueden no ser eficientes. No es posible usarlo para datos que no requieran un orden. Además no cumple con la propiedad de auto-balanceo. |
| 1. **Árbol n-ario** | Su funcionamiento puede emparejarse directamente con situaciones de la vida real. Puede redimensionarse sin mayor problema. | No es posible guardar información de manera ordenada, aunque si de manera jerárquica. |
| 1. **Árbol Rojinegro** | Su principal ventaja es que puede auto-balancearse, además de que es posible establecer un criterio de orden entre los elementos. | Las operaciones de eliminación e inserción suelen ser difíciles de entender y por ende de implementar en código. |
| 1. **Árbol AVL** | Su principal ventaja es que puede auto-balancearse, además de que es posible establecer un criterio de orden entre los elementos. | Las operaciones de eliminación e inserción suelen ser difíciles de entender y por ende de implementar. |

Teniendo en cuenta el análisis anterior, resulta necesario descartar las estructuras de datos cuyo funcionamiento no proporciona una solución óptima al problema planteado. Este es el caso del montículo debido a que la complejidad temporal de sus algoritmos es elevada comparada con otras estructuras, además de que este problema puede tornarse más grave al tener la necesidad de manipularse una gran cantidad de datos.

**Fase 5: Selección y evaluación de la mejor solución**

A continuación, se expondrán los criterios mediante los cuales se elegirán las estructuras de datos a utilizar en la resolución del problema en cuestión:

**Criterio 1:** Complejidad temporal de los algoritmos que manipulan la estructura de datos.

|  |  |
| --- | --- |
| **Complejidad Temporal** | **Puntuación** |
| O (logn) | 4 |
| O (n+k) | 2 |
| O (1) | 6 |

**Criterio 2:** Criterio de ordenamiento entre sus elementos.

|  |  |
| --- | --- |
| **Orden entre sus elementos** | **Puntuación** |
| Si | 10 |
| No | 5 |

**Criterio 3:** Propiedad de auto-balanceo de la estructura de datos.

|  |  |
| --- | --- |
| **Propiedad de auto-balanceado** | **Puntuación** |
| Auto-balanceado | 10 |
| No auto-balanceado | 2 |

En el siguiente recuadro se mostrarán las diferentes alternativas que, luego de evaluarlas con los criterios descritos, se elegirán las que cumplan con las condiciones establecidas con mayor rigurosidad.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Criterio 1** | **Criterio 2** | **Criterio 3** | **Total** |
| Alternativa 1. Tabla Hash | 6 | 5 | 2 | 13 |
| Alternativa 2. Árbol Binario | 4 | 10 | 2 | 16 |
| Alternativa 3: Árbol n-ario | 4 | 5 | 2 | 11 |
| Alternativa 4. Árbol rojinegro | 4 | 10 | 10 | 24 |
| Alternativa 5. Árbol AVL | 4 | 10 | 10 | 24 |

**Selección definitiva:** Teniendo en cuenta el método de selección utilizado anteriormente, se llegó a la conclusión de que las estructuras que deberán ser usadas en la creación del software son las siguientes: Árbol ARN y árbol AVL. La principal razón por la que se eligieron estas estructuras es porque su complejidad temporal es baja y además porque es posible mantener un criterio de ordenamiento entre sus elementos

# Fase 6: Elaboración de informes y especificaciones AD ABB

|  |
| --- |
| **TAD ABB**  *<K extends Comparable, V>* |

|  |
| --- |
|  |
| **Invariante**  Sea x un nodo del árbol. Si y es un nodo en el subarbol izquierdo de x, entonces key[y] key[x]. Si y es un nodo en el subarbol derecho de x, entonces key[y] key[x] |



|  |
| --- |
| **Operaciones primitivas**   * insertar ABB ABB * eliminar ABB  ABB * rotarIzquierda ABB ABB * rotarDerecha ABB ABB * mínimo ABB Nodo * máximo ABB Nodo * sucesor ABB Nodo * transplant ABB ABB |

|  |
| --- |
| **Insertar** |
| “Agregar nuevo nodo al árbol binario de búsqueda garantizando la propiedad de orden” |
| {pre: } |
| {post: se agrega el nuevo nodo al árbol} |

|  |
| --- |
| **Eliminar** |
| “Elimina el nodo cuya clave es indicada y garantiza la propiedad de orden en el árbol” |
| {pre: El árbol debe de estar creado y al menos su nodo-raíz debe existir} |
| {post: se elimina el nodo indicado} |

|  |
| --- |
| **RotarIzquierda** |
| “rotar hacía la izquierda el árbol cambiando su estructura sin alterar su orden” |
| {pre: el nodo que se desea rotar debe tener hijo izquierdo} |
| {post: se rota el árbol hacía la izquierda y éste conserva su orden} |

|  |
| --- |
| **RotarDerecha** |
| “rotar hacía la derecha el árbol cambiando su estructura sin alterar su orden” |
| {pre: el nodo que se desea rotar debe tener hijo derecho} |
| {post: se rota el árbol hacía la derecha y éste conserva su orden} |

|  |
| --- |
| **Mínimo** |
| “mínimo valor del ABB” |
| {pre: existe al menos un nodo en el árbol} |
| {post: se obtiene el valor mínimo del árbol} |

|  |
| --- |
| **Máximo** |
| “máximo valor del ABB” |
| {pre: existe por lo menos un nodo} |
| {post: se obtiene el valor máximo del árbol} |

|  |
| --- |
| **Sucesor** |
| “Da el sucesor de un nodo del árbol (Dado un nodo ‘x’ donde key[x]=k, el sucesor de ‘x’ es el nodo ‘y’ tal que key[y] es la llave más pequeña, mayor que key[x])” |
| {pre: el árbol está creado y para encontrar el sucesor deben existir como mínimo dos nodos que cumplen con la relación descrita en la restricción} |
| {post: se obtiene el sucesor del elemento señalado} |

**TAD Rojinegro**

|  |
| --- |
| **TAD Árbol Rojo-Negro**  *<K extends Comparable, V>* |
|  |
| **Invariante**   * Es un ABB * Cada nodo tiene un campo extra que almacena su color * La raíz es negra * Todo nodo es rojo y negro |



|  |
| --- |
| **Operaciones primitivas**   * insertar ARN ARN * eliminar ARN  ARN * rotarIzquierda ARN ARN * rotarDerecha ARN ARN * mínimo ARN Nodo * máximo ARN Nodo * sucesor ARN Nodo * transplant ARN ARN |

**TAD AVL**

|  |
| --- |
| **TAD AVL**  *<K extends Comparable, V>* |
|  |
| **Invariante**   * Es un ABB * Cada nodo cuenta con un factor de balanceo balanceFactor = height(left subtree) - height(right subtree) |



|  |
| --- |
| **Operaciones primitivas**   * insertar AVL AVL * eliminar AVL  AVL * rotarIzquierda AVL AVL * rotarDerecha AVL AVL * mínimo AVL Nodo * máximo AVL Nodo * sucesor AVL Nodo * transplant AVL AVL |

**Diseño de las pruebas.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo de la prueba:** obtener la key del nodo más pequeño del arbol | | | | |
| **Clase** | Método | Escenario | **Entradas** | **Resultado** |
| AVLTree | getMin() | setup1() | El valor de la key que puede variar el criterio por el cual este construido el arbol | La key del Nodo avl que fue agregado dentro del arbol |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo de la prueba:** obtener la key del nodo más grande del arbol | | | | |
| **Clase** | Método | Escenario | **Entradas** | **Resultado** |
| AVLTree | getMax() | Setup2() | El valor de la key que puede variar el criterio por el cual este construido el arbol | La key del Nodo avl que fue agregado dentro del arbol |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo de la prueba:** obtener el nodo predecesor del indicado | | | | |
| **Clase** | Método | Escenario | **Entradas** | **Resultado** |
| AVLTree | getPredecessor() | Setup3() | El valor de la key que puede variar el criterio por el cual este construido el arbol | La key del Nodo avl que es predecesor del indicado |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo de la prueba:** obtener el nodo sucesor del indicado | | | | |
| **Clase** | Método | Escenario | **Entradas** | **Resultado** |
| AVLTree | getSucessor() | Setup4() | El valor de la key que puede variar el criterio por el cual este construido el arbol | La key del Nodo avl que es sucesor del indicado |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo de la prueba:** obtener el peso del arbol | | | | |
| **Clase** | Método | Escenario | **Entradas** | **Resultado** |
| AVLTree | getWeight() | Setup5() |  | El peso del árbol , ósea el número de nodos agregados |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo de la prueba:** obtener el nodo por medio de una key | | | | |
| **Clase** | Método | Escenario | **Entradas** | **Resultado** |
| AVLTree | Search() | Setup5() | El valor de la key que puede variar el criterio por el cual este construido el arbol | El Nodo avl que es busacado por su propia key |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo de la prueba:** se inserta o añade un nodo al arbol | | | | |
| **Clase** | Método | Escenario | **Entradas** | **Resultado** |
| AVLTree | Insert() | Setup5() | El valor de la key que puede variar el criterio por el cual este construido el árbol, el valor del nodo | Se añade un nodo dentro del árbol avl |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo de la prueba:** se borra o elimina un nodo del arbol | | | | |
| **Clase** | Método | Escenario | **Entradas** | **Resultado** |
| AVLTree | Delete () | Setup5() | El valor de la key que puede variar el criterio por el cual este construido el árbol | Se rotorna el nodo que ha sido eliminado del arbol |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo de la prueba:** Añadir un nodo nuevo dentro del árbol ABB | | | | |
| **Clase** | Método | Escenario | **Entradas** | **Resultado** |
| ABBTree | add() | setup1() | El valor y la key del nodo que se va a agregar al arbol | Se agrega un nodo dentro del árbol ABB |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo de la prueba:** Remover un nodo nuevo dentro del árbol ABB | | | | |
| **Clase** | Método | Escenario | **Entradas** | **Resultado** |
| ABBTree | remove() | Setup2() | El valor y la key del nodo que se va a eliminar del arbol | Se elimina un nodo dentro del árbol ABB |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo de la prueba:** Buscar un nodo nuevo dentro del árbol ABB | | | | |
| **Clase** | Método | Escenario | **Entradas** | **Resultado** |
| ABBTree | Search() | Setup3() | El valor de la key del nodo que se va a buscar en el arbol | Se retorna el nodo buscado dentro del árbol ABB |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo de la prueba:** Añadir un nodo nuevo dentro del Trie | | | | |
| **Clase** | Método | Escenario | **Entradas** | **Resultado** |
| Trie | add() | setup1() | El valor del nodo que se va a agregar dentro del Trie | Se agrega un nodo dentro del Trie |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo de la prueba:** Buscar un nodo nuevo dentro del Trie | | | | |
| **Clase** | Método | Escenario | **Entradas** | **Resultado** |
| Trie | Search() | Setup2() | El valor del nodo que se va a buscar dentro del Trie | Se retorna un valor de verdad sobre dentro del Trie o no |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo de la prueba:** Completar una palabra con el Trie | | | | |
| **Clase** | Método | Escenario | **Entradas** | **Resultado** |
| Trie | completeSentence() | Setup3() | El valor del nodo o uno similar del que se quiere encontrar dentro del Trie | Se retorna un listado de los posibles palabras que se buscan |