

**Curso:** Algoritmos y Estructura de datos

**Informe del Proyecto (TF)**

**Sección:** CC31

**Grupo: Party Astronauts**

**Integrantes :**

**Marcelo Guerrero Iparraguirre u2020122589**

**Jorge Anco Galvez u20191b984**

**Profesor: Heider Sanchez**

**Índice**

Introducción

Descripción del caso de estudio planteado por el grupo

Diagrama de clases de entidades principales

Definición de TDA y estructuras de datos a usar o diseño de archivos a utilizar

Complejidad en notación detallada y Big O de los métodos de las estructuras de datos.

Conclusiones

Referencias

1. **Introducción**
2. **Descripción del caso de estudio**
3. **Diagrama de clases de entidades principales**
4. **Definición de TDA y estructuras de datos a usar p diseño de archivos a utilizar**
5. **Complejidad en notación detallada y Big O de los métodos de estructuras de datos**
6. **Conclusiones**
7. **Referencias**

1.Introducción

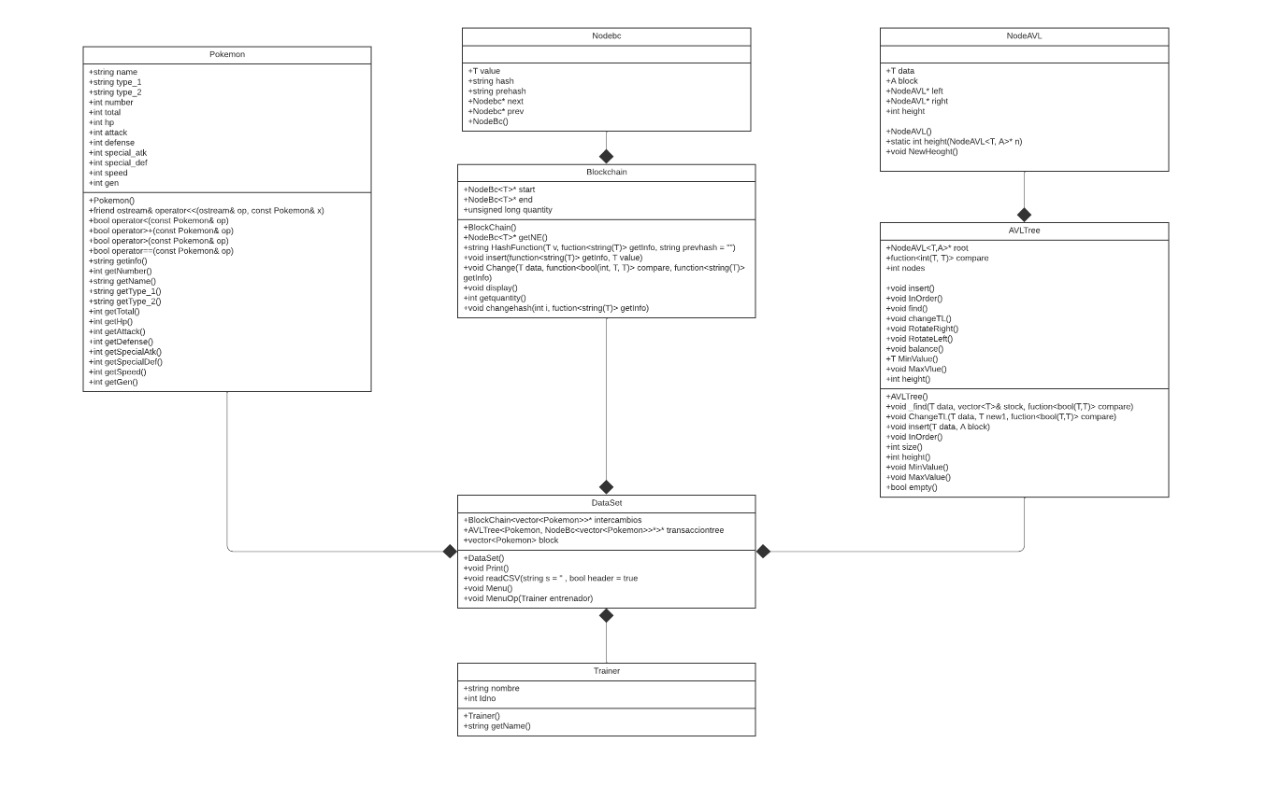
El blockchain es una estructura de datos que ha comenzado a ganar relevancia durante los últimos años debido a su uso en la elaboración de NFT’s. el objetivo de este proyecto es implementar un blockchain para almacenar diferentes tipos de transacciones.

El ide de desarrollo para dicho proyecto fue principalmente Visual Studio Code, además se utilizó replit y Clion. el lenguaje de programación utilizado para el proyecto fue c++ y los conocimientos y conceptos utilizados provienen del curso “Algoritmos y Estructuras de Datos”, asi como programacion 2 y estructuras discretas

2.Descripción del caso de estudio

El caso de estudio de este proyecto final consiste en desarrollar un programa simulando el funcionamiento de un “PC” del videojuego Pokémon al momento de almacenar pokemones en dicho pc o retirarlos hacia el equipo pokémon del usuario.

Utilizando un dataset de la página web Kaggel y haciendo uso de diferentes estructuras como árboles binarios y listas doblemente enlazadas para lograr el desarrollo de las diferentes funciones requeridas en el proyecto. Todas las estructuras de datos se han implementado de manera genérica.

3. Diagrama de clases de Entidades Principales

4. Definición de TDA y estructuras de datos

Según Garcia Cano y Solano Galvez(2017)Un tipo de dato abstracto es un conjunto de datos u objetos creado de manera personalizada por el programador para un fin específico.Dichos datos se pueden manipular de manera similar a los tipos de datos que están predefinidos dentro del lenguaje de programación, encapsulando la información que se requiera.

Lista Doblemente Enlazada

Como señala Gunawardena(2009) una lista doblemente enlazada es una lista que contiene una conexión con sus nodos siguientes y anteriores.No como la lista simplemente enlazada, la lista doblemente enlazada permite recorrer la lista de 2 maneras

Árbol Binario.

Según Parlante(2001) Un árbol binario está hecho de nodos,donde cada nodo contiene un puntero derecha y un puntero izquierda y un elemento de datos.Además se tiene la raíz que es el punto mas alto del árbol.Los punteros derecha es izquierda recursivamente apuntan a subárboles más pequeños a cada uno de sus lados.

Archivos

Un objeto archivo, como su nombre lo indica es un objeto contenido en la computadora,dicho objeto puede contener información la cual puede ser manipulada por el sistema operativo o un programa. Los archivos tiene un nombre y un sufijo con el cual se identifica que tipo de archivo es(.csv y/o .txt y/o .bi) Para el proyecto se está utilizando un archivo .csv llamado “Pokémon” del cual se extraen los datos de diferentes pokemones y sus estadísticas respectivas

5.Complejidad en notación detallada y Big O de los métodos de estructuras de datos

|  |
| --- |
| template <class T, class A>  class AVlTree {      NodeAVL<T, A>\* root;      function<int(T, T)> compare;      int nodes = 0;  private:      void insert(NodeAVL<T, A>\*& thnode, T element, A ablock) {          if (thnode == nullptr) //2+...          {              thnode = new NodeAVL<T, A>(element, ablock);    //2              nodes++;    //2          }          else if (element < thnode->data) //2+...              return insert(thnode->left, element, ablock);   //2          else if (element >= thnode->data) //2+...              return insert(thnode->right, element, ablock);  //2          balance(thnode);    //1      }        void InOrder(NodeAVL<T, A>\* thnode) {          if (thnode == nullptr) return;  //1+1          InOrder(thnode->left);  //1          cout << thnode->block << " ";   //3          InOrder(thnode->right); //1      }        void find(NodeAVL<T, A>\* thnode, T data, vector<T>& stock, function<bool(T, T)> compare) {          if (thnode == nullptr) return;  //1+1          find(thnode->left, data, stock, compare);   //1          if (compare(thnode->data, data))    //2+..              stock.push\_back(thnode->data);  //1          find(thnode->right, data, stock, compare);  //1      }        void changeTL(NodeAVL<T, A>\* thnode, T data, T new1, function<bool(T, T)> compare) {          if (thnode == nullptr) return;  //1+1          changeTL(thnode->left, data, new1, compare);    //1          if (compare(thnode->data, data))    //1+...              thnode->data = new1;    //2          changeTL(thnode->right, data, new1, compare);   //1      }          void RotateRight(NodeAVL<T, A>\*& thnode) {          NodeAVL<T, A>\* temp = thnode->left; //3          thnode->left = temp->right; //3          thnode->NewHeight();    //2          temp->right = thnode;   //2          thnode = temp;  //1          thnode->NewHeight();    //2      }        void RotateLeft(NodeAVL<T, A>\*& thnode) {          NodeAVL<T, A>\* temp = thnode->right;    //3          thnode->right = temp->left; //3          thnode->NewHeight();    //2          temp->left = thnode;    //2          thnode = temp;  //1          thnode->NewHeight();    //2      }        void balance(NodeAVL<T, A>\*& thnode) {          int balance\_fact = NodeAVL<T, A>::heighti(thnode->left) - NodeAVL<T, A>::heighti(thnode->right);    //4          if (balance\_fact > 1) {     //1+...              int right\_left\_h = NodeAVL<T, A>::heighti(thnode->right->left); //3              int right\_right\_h = NodeAVL<T, A>::heighti(thnode->right->right);   //3              if (right\_left\_h > right\_right\_h) {     //1+...                  RotateRight(thnode->right); //1              }               RotateLeft(thnode);    //1          }          if (balance\_fact < -1) {    //2 + ...              int left\_left\_h = NodeAVL<T, A>::heighti(thnode->left->left);   //3              int left\_right\_h = NodeAVL<T, A>::heighti(thnode->left->right); //3              if (left\_left\_h < left\_right\_h) {   //1+...                  RotateLeft(thnode); //1              }              RotateRight(thnode); //1          }      }      T MinValue(NodeAVL<T, A>\* thnode)      {          while (thnode->left != nullptr) //2+n(...)          {              thnode = thnode->left;  //2          }          return thnode->data;    //2      }      void MaxVlue(NodeAVL<T, A>\* thnode)      {          if (thnode->right == nullptr) //2+...          {              cout << thnode->data;   //2          }          else          {              MaxVlue(thnode->right); //1          }      }      int height(NodeAVL<T,A> \*thnode){          if(!thnode) return -1;  //1+2          else{              int tleft = height(thnode->left);   //3              int tright = height(thnode->right); //3              if (tleft > tright) //1+...                  return tleft + 1;   //2              else                  return tright + 1;  //2          }      }    public:      AVlTree(function<int(T, T)> compare) : compare(compare) {          root  = nullptr;    //1      }      void \_find(T data, vector<T>& stock, function<bool(T, T)> compare) {          find(root, data, stock, compare);   //1      }      void ChangeTL(T data, T new1, function<bool(T, T)> compare) {          changeTL(this->root, data, new1, compare);  //1      }      void insert(T data, A block) {          insert(this->root, data, block);    //1      }      void InOrder() {          InOrder(this->root);    //1      }      int size() {          return nodes;   //2      }      int height() {          return height(this->root);  //2      }      void MinValue()      {          MinValue(this->node);   //1      }      void MaxValue()      {          MaxValue(this->root);   //1      }      bool empty() {          if (root == nullptr)    //2+..              return true;    //1      }    };  //Big O: O(n) |

|  |
| --- |
| template <class T>  class BlockChain {  private:      NodeBc<T>\* start;   //2      NodeBc<T>\* end;     //2      unsigned long quantity; //2  public:      BlockChain() {          start = end = nullptr;  //2          quantity = 0;   //1      }      NodeBc<T>\* getNE() {          return end; //1      }      string Hashfunction(T v, function<string(T)> getInfo, string prevhash = "") {         srand(time(NULL));   //2          int seed = rand()%100;  //4          int factor = rand()%6 + 2;  //4          string data = getInfo(v);   //3          string hash = "", prevh = "";   //3          for (int i = 0; i < data.length()/factor; ++i) {    //1+n(1+2+....)              hash = hash + data[i];  //2          }          long long hog = stoll(hash);    //4          if (prevhash != "") {   //1+...              for (int i = 0; i < prevhash.length() / factor; ++i) {  //1+n(1+2+...)                  prevh = prevh + prevhash[i];    //2              }              long long phog = stol(prevh);   //4              phog = phog + hog;  //2          }          hog = hog \* seed;   //2          return to\_string(hog) + to\_string(data.length());   //5      }      void insert(function<string(T)> getInfo, T value) {          NodeBc<T>\* new1 = new NodeBc<T>(value); //3          if (quantity == 0) {    //1+...              start = end = new1; //2              new1->prevhash = "No hash"; //2              new1->hash = Hashfunction(value, getInfo);  //2          }          else {              end->next = new1;   //2              new1->prev = end;   //2              end = new1; //1              new1->prevhash = new1->prev->hash;  //2              new1->hash = Hashfunction(value, getInfo, new1->prevhash);  //2          }          ++quantity; //2          }        void ChangeT(T data, function<bool(int, T, T)> compare, function<string(T)> getInfo) {          NodeBc<T>\* aux = start; //2          int j = 0;  //2          while (aux != nullptr) {    //1+n(....)              for (int i = 0; i < aux->value.size(); ++i) {   //1+n(1+2+...)                  if (compare(i, aux->value, data)) { //1+...                      aux->value.at(i) = data.at(1);  //4                      changehash(j, getInfo); //1                  }              }              aux = aux->next;    //2              ++j;    //2          }      }      void display() {          NodeBc<T>\* aux = start; //2          int j = 1;  //2          while (aux != nullptr) {              cout << "Block " << j << "\n";  //3              cout << "->Previous Hash: " << aux->prevhash << "\n";   //3              cout << "->Hash: " << aux->hash << "\n";    //3              cout << "->Transacciones: " << "\n";    //2              for (int i = 0; i < aux->value.size(); ++i) {   //1+n(1+2+...)                  cout << aux->value.at(i) << "\n";   //3              }              cout << "\n";   //1              aux = aux->next;    //2              j++;    //2          }      }      int getquantity() {          return quantity;    //1      }      void changehash(int i, function<string(T)> getInfo) {          NodeBc<T>\* aux = start; //2          NodeBc<T>\* aux2;    //2          int j = 0;  //2          while (aux != nullptr) {    //1+...              if (j == i) {   //1+...                  aux->hash = aux->hash + "2";    //4                  aux2 = aux; //1                  for (int c = j; c < quantity - j - 1; ++c) {    //1+n(1+2+...)                      aux2->next->prevhash = aux2->hash;  //4                      aux2->next->hash = Hashfunction(aux2->value, getInfo, aux2->hash);  //4                      aux2 = aux2->next;  //2                  }              }              aux = aux->next;    //2              j++;    //2          }      }  };  //Big O: O(n^2) |

6.Conclusiones

En conclusión el proyecto nos ha ayudado a entender de manera correcta el uso del blockchain en diferentes ámbitos.

Otra conclusión es que durante el desarrollo del proyecto el grupo ha aprendido a usar mejor las herramientas de github y organizarse para optimizar el tiempo otorgado para utilizar el trabajo.

Al momento de desarrollar el trabajo e implementar los diferentes tipos de estructuras el grupo aprendió a distinguir de mejor manera las diferentes situaciones en las que se deben implementar los diferentes tipos de estructuras

Como conclusión final, el grupo aprendió diferentes herramientas de desarrollo que serán de gran utilidad durante el transcurso de la carrera y la vida como un profesional. Además de adquirir nuevos conocimientos en el ámbito de las estructuras de datos, lectura y escritura de archivos, trabajo en equipo, etc.

7. Bibliografía

García, E,. & Solano, J. (2017) *Guía práctica de estudio 03. Tipo de dato abstracto.* Recuperado de : <http://odin.fi-b.unam.mx/salac/practicasEDAA/eda1_p3.pdf>

Gunawardena,A.(2009) Lecture 11 Doubly Linked Lists & Array of Linked List. Recuperado de <https://www.cs.cmu.edu/~guna/15-123S11/Lectures/Lecture11.pdf>

Parlante,E.(2001) Binary Trees. Recuperado :<http://cslibrary.stanford.edu/110/BinaryTrees.html>