Análisis y Algoritmos

Luis Alberto Pineda Chavez
Universidad de Artes Digitales

Guadalajara, Jalisco

Email: idv17c.lpineda@uartesdigitales.edu.mx

Profesor: Efraín Padilla

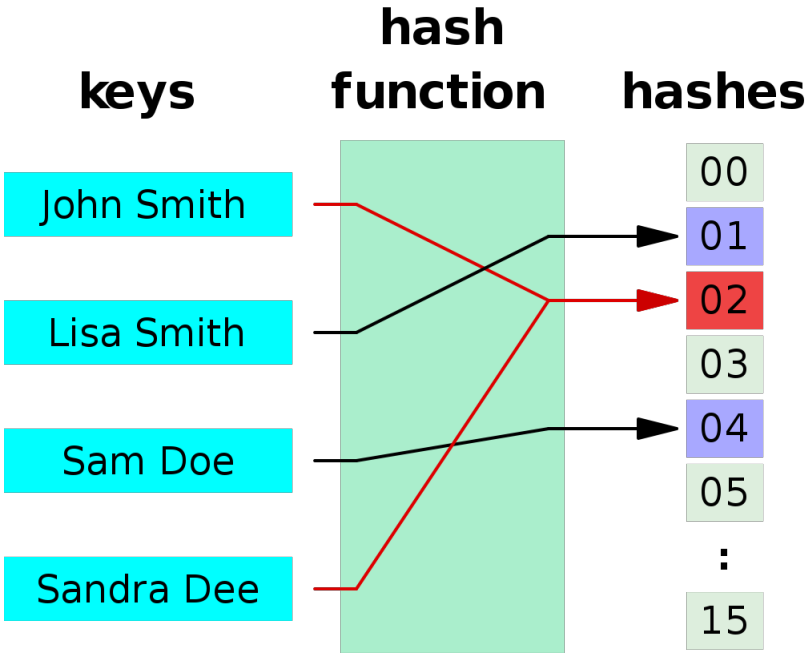
Mayo 23, 2019

1) Tablas Hash

Las tablas hash son estructuras de datos que actuan como contenedores asosiativos, cuentan con una serie de contenedores internos, cada uno de ellos asociado a una clave o ”key” única, que sirve como forma de acceso al mismo. Todos los datos que se encuentran en dicho contenedor son llamados valores o ”values”.

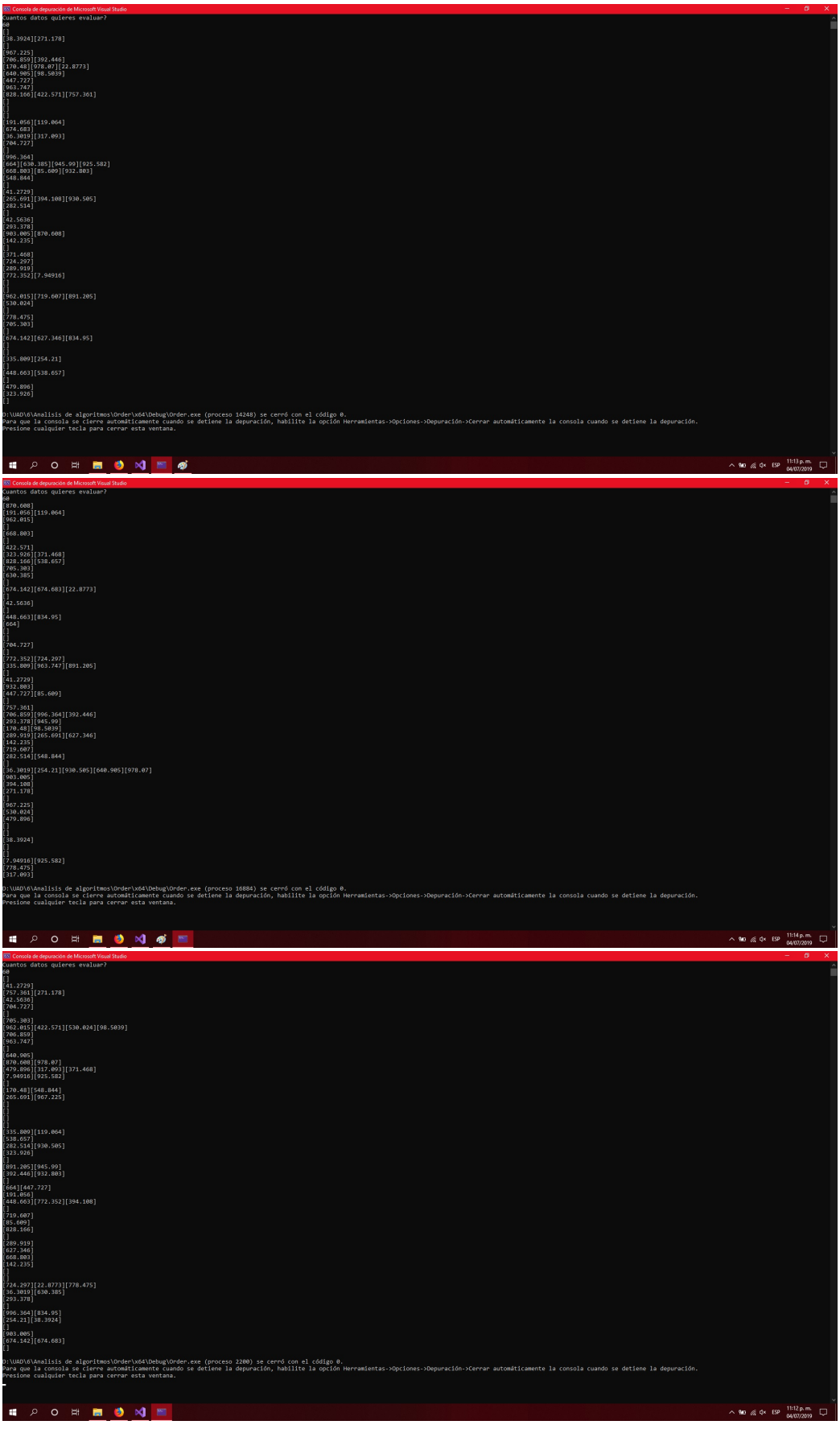
El objetivo

El uso de estas tablas es muy práctico, ya que se pueden almacenar grandes cantidades de información en ellos pero el tiempo de busqueda siquie siendo muy bueno sim importar el tamaño.



En el mejor de los casos, cada key tendrá un solo valor en su contenedor, esto hará que la busqueda en la estructura completa dea de complejidad O(1)

Estos son los resultados de una sencilla implementación de tablas Hash con tres funciones hash distintas que procesan datos numericos de punto flotante:



Implementación:

```
Hash::Hash()
{
        srand(time(nullptr));
        m_primeNumbers = new int[20]{ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
        37, 41, 43, 47, 53, 59, 61, 67, 71 };
        int m_currentPrime = 0;
        int m_hashA = 0;
        int m_hashB = 0;
}

Hash::~Hash()
{
        if (m_primeNumbers != nullptr)
        {
                delete[] m_primeNumbers;
                m_primeNumbers = nullptr;
        }
}

std::vector<std::vector<float>>* Hash::Get()
{
        return &m_table;
}

int Hash::GetRandomPrime()
{
        return m_currentPrime = m_primeNumbers[std::rand() % 19];
}

void Hash::Initialize(int size)
{
        m_table.resize((int) (size - (size * 0.1f)));
        m_currentPrime = m_primeNumbers[std::rand() % 19];

        m_hashA = std::rand() % m_table.size();
        m_hashB = 1 + std::rand() % ((m_table.size() <= 1 ? 2 : m_table.size()) - 1);
}

void Hash::Multiplication(float value)
{
        m_table[((int) value * UINT32_C(2654435761)) % m_table.size()].push_back(value);
}

void Hash::Division(float value)
{
        float first = (int) value % (m_table.size() / 2), second = std::floor((int)
        value % m_table.size());

        m_table[(int)(first + second) % m_table.size()].push_back(value);
}

void Hash::Universal(float value)
{
        int key = m_currentPrime + (value * m_hashA) + (value * m_hashB);
        m_table[key % m_table.size()].push_back(value);
}
```

REFERENCIAS

[1] Cormen, T., Leiserson, C., Rivest, R. and Stein, C. (2009). Introduction to algorithms. Cambridge (England): Mit Press.