

# Análisis y Algoritmos

Luis Alberto Pineda Chavez  
Universidad de Artes Digitales

Guadalajara, Jalisco

Email: idv17c.lpineda@uartesdigitales.edu.mx

Profesor: Efraín Padilla

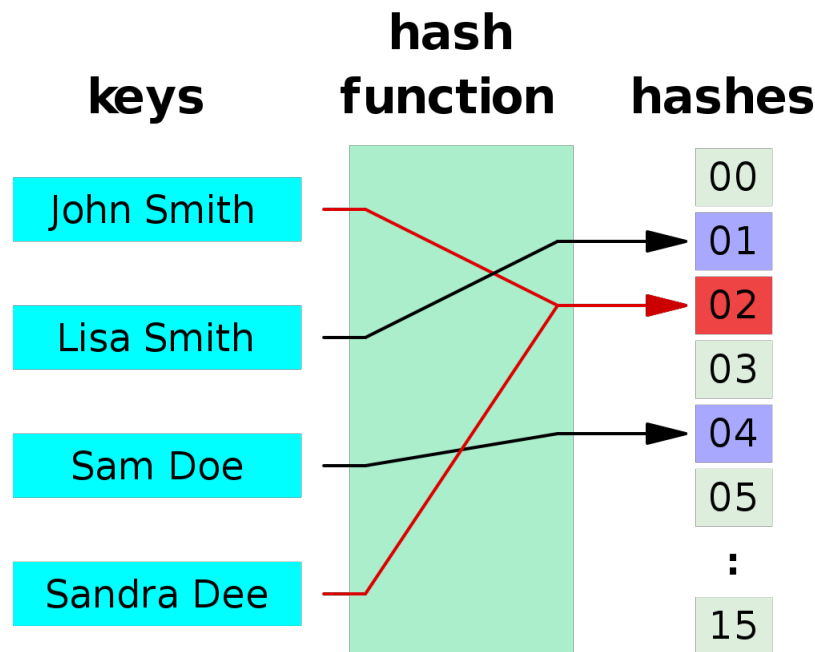
Mayo 23, 2019

## 1) Tablas Hash

Las tablas hash son estructuras de datos que actúan 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 búsqueda sigue siendo muy bueno sin importar el tamaño.



En el mejor de los casos, cada key tendrá un solo valor en su contenedor, esto hará que la búsqueda en la estructura completa sea 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:

```

Microsoft Visual Studio
Cuantos datos quieres evaluar?
1
18.3924][271.178]
907.225]
786.859][392.446]
178.48][978.07][22.8773]
680.905][98.5035]
647.727]
962.147]
828.166][422.571][757.361]
1
191.056][119.064]
674.083]
78.3819][317.093]
704.727]
996.164]
664][530.385][945.90][925.582]
668.803][85.009][932.803]
48.844]
1.27729]
365.691][394.100][930.505]
232.534]
42.5636]
293.178]
981.005][870.608]
142.235]
171.460]
724.297]
207.819]
772.352][7.94910]
962.015][719.007][891.205]
138.824]
778.475]
705.303]
674.142][627.346][834.95]
135.009][254.21]
448.663][538.657]
479.896]
212.929]
1
1
VUAD\Análisis de algoritmos\Orden\vd4\Debug\Orden.exe (proceso 14248) se cerró con el código 0.
Para que la consola se cierre automáticamente cuando se detiene la depuración, habilite la opción Herramientas>Opciones>Depuración>Cerrar automáticamente la consola cuando se detiene la depuración.
Presione cualquier tecla para cerrar esta ventana.

Microsoft Visual Studio
Cuantos datos quieres evaluar?
1
870.608]
191.056][119.064]
962.015]
668.803]
422.571]
223.526][371.460]
828.166][538.657]
705.303]
530.385]
174.142][674.683][22.8773]
12.5836]
448.663][834.95]
664]
704.727]
772.352][724.297]
335.803][825.743][881.205]
1.27729]
932.803]
647.727][85.009]
757.011]
786.859][996.364][392.446]
293.178][945.90]
178.48][98.5035]
289.919][265.693][627.346]
142.235]
719.007]
82.534][548.844]
16.3819][254.21][930.505][640.905][978.07]
981.005]
894.100]
271.178]
907.225]
530.824]
479.896]
1
18.3924]
1.94016][925.582]
778.475]
317.093]
1
VUAD\Análisis de algoritmos\Orden\vd4\Debug\Orden.exe (proceso 10884) se cerró con el código 0.
Para que la consola se cierre automáticamente cuando se detiene la depuración, habilite la opción Herramientas>Opciones>Depuración>Cerrar automáticamente la consola cuando se detiene la depuración.
Presione cualquier tecla para cerrar esta ventana.

Microsoft Visual Studio
Cuantos datos quieres evaluar?
1
91.27729]
757.361][271.178]
42.5636]
704.727]
705.303]
962.015][422.571][530.634][98.5035]
786.859]
963.747]
448.663]
670.000][978.07]
479.896][317.093][371.460]
7.94910][757.352]
178.48][548.844]
265.691][907.225]
1
135.009][119.064]
538.657]
232.534][930.505]
212.929]
891.205][945.90]
992.446][932.803]
664][647.727]
191.056]
448.663][772.352][394.100]
719.007]
85.009]
828.166]
289.919]
627.346]
668.803]
142.235]
724.297][22.8773][778.475]
16.3819][630.385]
293.178]
1396.164][834.95]
254.21][18.3924]
981.005]
674.142][674.683]
1
VUAD\Análisis de algoritmos\Orden\vd4\Debug\Orden.exe (proceso 1180) se cerró con el código 0.
Para que la consola se cierre automáticamente cuando se detiene la depuración, habilite la opción Herramientas>Opciones>Depuración>Cerrar automáticamente la consola cuando se detiene la depuración.
Presione cualquier tecla para cerrar esta ventana.

```

## 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 };
    m_currentPrime = 0;
    m_hashA = 0;
    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.