Instituto Tecnológico y de Estudios Superiores de Monterrey

Programación de estructuras de datos y algoritmos fundamentales (Gpo 602)

Act 5.2 - Actividad Integral sobre el uso de códigos hash

Adela Alejandra Solorio Alcázar A01637205
Luz Patricia Hernández Ramírez A01637277
Marcela Beatriz De La Rosa Barrios A01637239

28/11/2023

Act 5.2 - Actividad Integral sobre el uso de códigos hash

Se siguieron las indicaciones de la actividad al crear una clase de tabla Hash para el cual se abre el archivo de entrada "bitacora.txt" y se agregan todas las entradas en una estructura de tabla hash donde la clave es el número de puerto y los valores a guardar en la tabla hash son el total. Para esto, se investigó e implementó una función hash que evita tantas colisiones cómo fue posible, se implementó un método adecuado de manejo de colisiones (encadenamiento, direccionamiento abierto) y se imprimieron los 5 números de puerto con más accesos (ya obtenidos en la actividad 3.4) en la tabla hash, mismos que sí coinciden

Una función hash es una función matemática que convierte un dato ingresado a una cadena de caracteres de un tamaño específico (valor hash), es útil porque le asigna a cada entrada un valor hash único (si la función está bien implementada), lo que permite “esconder” el valor de entrada y poder acceder a el solo mediante el valor hash generado (Loo, s.f.).

Universal hashing es un método de la función hash que previene o reduce el número de colisiones hasta que la probabilidad de que suceda una es de 1/m, m siendo el tamaño de la hash table; como este método funciona es que crea una familia de funciones hash y selecciona una función random de ella, así obteniendo keys más variadas (Bargal, s.f.).

Código implementado:

- **Declaración de las clases y funciones con sus respectivas complejidades (Big O)**

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <queue>
#include <map>
#include <algorithm>
#include <math.h>
#include <list>
#include <random>
using namespace std;

struct Node {
  int port;  //key
  vector<string> ipAdd; //valor (IP
addresses)
  int count; // cant (number of accesses)
  Node *left, *right; //punteros a los hijos
  Node(int& p) : port(p), count(1),
left(nullptr), right(nullptr) {}
};

class HashTable {
  private:
    int capacity;
    vector<int> *table;
  public:
    HashTable(int V) {
      int size = getPrime(V);
      this->capacity = size;
      table = new vector<int>[capacity];
    }
    ~HashTable() {
      delete[] table;
      cout << "\nDestructor: HashTable
deleted. " << endl;
    }
    bool checkPrime(int); // O(sqrt(n))
```

```
    int getPrime(int); // O(sqrt(n))
    void insert(int); // O(1 + alpha)
    int hashFunction(int); // O(sqrt(n))
    void displayHash(); // O(n)
    void top5portsWIthMostAccesses(); // O(n)
    void top5portsWIthMostAccessesAndIPs();
// O(n)
};
```

- **Desarrollo de funciones**

```
bool HashTable::checkPrime(int n) {
  if (n == 1 || n == 0)
    return false;

  int sqr_root = sqrt(n);
  for (int i = 2; i <= sqr_root; i++)
    if (n % i == 0)
      return false;
  return true;
} // O(sqrt(n))

int HashTable::getPrime(int n) {
  if (n % 2 == 0)
    n++;
  while (!checkPrime(n))
    n += 2;
  return n;
} // O(sqrt(n))

int HashTable::hashFunction(int key) {
  int p = getPrime(capacity);
  int a = rand() % p;
  int b = rand() % p;

  return ((a * key + b) % p) % capacity;
} // O(sqrt(n))

void HashTable::insert(int key) {
  int index = hashFunction(key);
  table[index].push_back(key);
} // O(1 + alpha)

void HashTable::displayHash() {
  for (int i = 0; i < capacity; i++) {
    cout << i;
    for (auto x : table[i])
      cout << " --> " << x;
    cout << endl;
  }
} // O(n)

bool CompareCount(pair<int, Node*>& a,
pair<int, Node*>& b){ //Comparar el número de
accesos de cada puerto
  return a.second->count > b.second->count;
//ordenar de mayor a menor
}
void HashTable::top5portsWIthMostAccesses() {
  map<int, int> ports;
  for (int i = 0; i < capacity; i++) {
    for (auto x : table[i]) {
      ports[x]++;
    }
  }
  priority_queue<pair<int, int>> pq;
  for (auto x : ports) {
    pq.push(make_pair(x.second, x.first));
  }
  cout << "Top 5 ports with most accesses: "
<< endl;
  for (int i = 0; i < 5; i++) {
    cout << pq.top().second << " with " <<
pq.top().first << " accesses" << endl;
    pq.pop();
  }
} // O(n)

void
HashTable::top5portsWIthMostAccessesAndIPs()
{
  map<int, Node*> ports;
  for (int i = 0; i < capacity; i++) {
    for (auto x : table[i]) {
      if (ports.find(x) == ports.end()) {
        ports[x] = new Node(x);
      }
      ports[x]->count++;
    }
  }
  ifstream inputFile("bitacora.txt");
  if (!inputFile.is_open()) {
    cerr << "Error: No se pudo abrir el
archivo." << endl;
    return;
  }
  string mes, fecha, ipAddress, razon;
//variables para almacenar los datos del
archivo
  int dia, portNum; //variables para
almacenar los datos del archivo
```

```
  while (inputFile >> mes >> dia >> fecha >>
ipAddress) {
    getline(inputFile, razon);
    razon = razon.substr(1);
    portNum =
stoi(ipAddress.substr(ipAddress.find(":") +
1)); //obtener el puerto
    if (ports.find(portNum) == ports.end()) {
      ports[portNum] = new Node(portNum);
    }

ports[portNum]->ipAdd.push_back(ipAddress);
  }
  inputFile.close();
  vector<pair<int, Node*>>
topPorts(ports.begin(), ports.end());
  sort(topPorts.begin(), topPorts.end(),
```

```
&CompareCount);
  for (int i = 0; i < 5 && i <
topPorts.size(); ++i) {
    cout << topPorts[i].first << " with " <<
(topPorts[i].second->count) - 1<< " accesses"
<< endl;
    cout << "IP addresses: " << endl;
    for (int j = 0; j <
topPorts[i].second->ipAdd.size(); j++) {
      cout << topPorts[i].second->ipAdd[j] <<
endl;
    }
    cout << endl;
  }
}
```

- **Main**

```
int main(){
  //Abrir archivo
  ifstream inputFile("bitacora.txt");
  if (!inputFile.is_open()) {
    cerr << "Error: No se pudo abrir el archivo." << endl;
    return 1;
  }
  cout << "Hash Table "<<endl;
  HashTable hash(100);
  string mes, fecha, ipAddress, razon; //variables para almacenar los datos del archivo
  int dia, portNum; //variables para almacenar los datos del archivo
  while (inputFile >> mes >> dia >> fecha >> ipAddress) {
    getline(inputFile, razon);
    razon = razon.substr(1);
    portNum = stoi(ipAddress.substr(ipAddress.find(":") + 1)); //obtener el puerto
    hash.insert(portNum);
  }
  cout<<"-----------------------"<<endl;
  cout<<"Summarized:"<<endl;
  hash.top5portsWIthMostAccesses();
  cout<<"-----------------------"<<endl;
  cout << "Top 5 Ports with the Most Accesses (including their IPs):" << endl;
  hash.top5portsWIthMostAccessesAndIPs();
  cout<<"-----------------------"<<endl;
  inputFile.close();
  return 0;
};
```

Salida del código:

```
Hash Table
------------------------
Summarized:
Top 5 ports with most accesses:
6170 with 16 accesses
5525 with 15 accesses
6445 with 14 accesses
6195 with 14 accesses
5504 with 14 accesses
------------------------
```

```
------------------------
Top 5 Ports with the Most Accesses (including their IPs):
6170 with 16 accesses
IP addresses:
277.92.970.8:6170
230.33.53.59:6170
937.72.625.19:6170
90.28.654.26:6170
365.26.509.53:6170
306.98.945.11:6170
452.91.940.89:6170
682.90.524.94:6170
88.86.539.35:6170
917.56.870.54:6170
355.24.263.3:6170
26.90.994.43:6170
551.41.745.87:6170
432.25.264.17:6170
475.3.160.9:6170
158.89.136.98:6170
```

```
5525 with 15 accesses
IP addresses:
169.5.713.39:5525
773.69.326.5:5525
687.20.538.79:5525
77.40.631.83:5525
633.95.703.1:5525
26.20.623.71:5525
242.77.639.20:5525
107.10.407.39:5525
595.69.155.22:5525
490.42.559.96:5525
276.27.429.56:5525
91.64.321.45:5525
495.49.646.62:5525
535.88.300.1:5525
236.33.510.58:5525
```

```
6445 with 14 accesses
IP addresses:
192.30.301.39:6445
18.30.488.95:6445
676.63.322.12:6445
392.72.754.52:6445
826.14.26.96:6445
126.46.766.74:6445
975.69.685.87:6445
96.50.625.4:6445
718.71.804.69:6445
448.94.586.67:6445
546.72.947.17:6445
842.57.282.73:6445
179.77.589.78:6445
203.69.591.7:6445
```

```
4784 with 14 accesses
IP addresses:
729.36.924.12:4784
541.90.966.28:4784
528.47.9.45:4784
477.58.924.91:4784
196.89.257.33:4784
639.2.417.45:4784
683.64.971.34:4784
78.83.621.72:4784
375.3.675.78:4784
836.41.82.90:4784
56.70.271.75:4784
565.43.106.52:4784
23.94.658.27:4784
16.59.125.95:4784
```

```
5365 with 14 accesses
IP addresses:
554.19.736.32:5365
531.61.953.7:5365
915.31.984.5:5365
898.97.97.56:5365
501.57.909.79:5365
617.82.382.31:5365
939.29.895.59:5365
1.74.592.32:5365
834.46.353.68:5365
806.48.860.67:5365
810.37.716.52:5365
667.87.502.73:5365
163.3.148.38:5365
691.95.234.77:5365

------------------------

Destructor: HashTable deleted.
```

Reflexión individual:

Marcela de la Rosa: La investigación y el código para obtener los top 5 puertos más visitados a partir del archivo "bitacora.txt" reflejan un conocimiento sólido para el diseño eficiente de estructuras de datos, mismo que se puede observar a partir de la implementación de una HashTable con atención a la función hash y el manejo de colisiones, para la cual la explicación de la función introduce el universal hashing que ayuda a minimizar colisiones. El código resultante, que imprime los 5 puertos con más accesos, demuestra la utilidad práctica de la tabla hash para el análisis de grandes conjuntos de datos.

Adela Solorio: La importancia de usar la función hash y el método de resolución de colisiones correctos es muy grande porque mientras más se eviten colisiones en la tabla hash, más funcional es, pues es más fácil la búsqueda y acceso a los datos en ella, también, esto aumenta la eficiencia de recursos de almacenamiento y el rendimiento,

pues entre menos colisiones, más pequeña puede ser la tabla y más rápidas las consultas. El resolver una situación de este tipo utilizando tablas hash es conveniente porque estas proporcionan seguridad, consistencia y velocidad en el almacenamiento y procesamiento de los datos, lo cual es necesario cuando se trata de registros como los de esta actividad.

Luz Patricia Hernández: Explorar la implementación de hash tables en este proyecto me brindó una valiosa perspectiva sobre la agilidad y eficiencia en la búsqueda y recuperación de datos, especialmente al tratar con información vinculada a dominios. Al construir una tabla hash donde la clave es el dominio y el valor es un resumen integral que abarca el número de accesos, las conexiones y las IPs asociadas, las hash tables son una herramienta excelente, pues facilitan la asociación entre claves y valores, gracias a la función de hash que asigna la clave a una ubicación específica en la tabla. Este enfoque permite un acceso casi instantáneo a los datos, independientemente del tamaño de la base, ya que se reduce el conjunto de posibles ubicaciones a una sola.
Aunque las hash tables son eficientes, a comparación de otros algoritmos que vimos, su rendimiento puede variar por lo que algoritmos de búsqueda y ordenamiento como el árbol binario de búsqueda o el algoritmo de ordenamiento rápido (quicksort) podrían complementarlas también .Finalmente, este método no solo facilita la recuperación de datos, sino que también permite presentar la información de manera clara y estructurada, como el número de accesos, las conexiones únicas y las IPs asociadas al dominio.

Referencias

Bargal, S. (s.f.). Universal Hashing. *Boston University*. Recuperado de https://www.cs.bu.edu/faculty/homer/537/talks/SarahAdelBargal_UniversalHashingnotes.pdf

Loo, A. (s.f.). Hash function. *Corporate Finance Institute.* Recuperado de https://corporatefinanceinstitute.com/resources/cryptocurrency/hash-function/