Jorge Del Barco Garza A01284234

Análisis de Complejidad y Reflexiones Finales Actividad 5.2

```cpp
int main(int argc, char const *argv[])
{
  int opcion = 0;
  bool cargar = false;
  unordered_map<string, IpAddress*> logger;

  do{
    cout << endl;
    cout << "MENU " << endl;
    cout << "1. Cargar bitacora" << endl;
    cout << "2. Consultar IP" << endl;
    cout << "3. Salir" << endl;
    cout << "Selecciona un numero: ";
    cin >> opcion;
    if(opcion != 1 && opcion != 2 && opcion != 3){
      continue;
    }

    switch(opcion){
      case 1:
        {
          fstream in("bitacoraACT5_2.txt");
          string readLog = "";
          string key = "";

          while(getline(in, readLog)){
            key = getIpAddress(readLog);
            if (logger.find(key) == logger.end()){
              logger.insert({key, new IpAddress(readLog)});
            } else {
              logger[key]->increasePingCounter();
            }
          }

          cout << "¡La bitacora ha sido cargada!" << endl;
          cargar = true;
          continue;

          break;
        }
      case 2:
        {
          if(cargar == true){
            string ipconsultar;
            cout << "Ingresa la IP que quieres consultar:\n";
            cout << "(Ejemplo: 224.182.134.50, 155.66.242.110)
            \n";
            cin >> ipconsultar;
            cout << endl;
            cout << "RESULTADOS CONSULTA:" << endl;
            getIpInfo(logger, ipconsultar);
          }else{
            cout <<"¡Necesitas cargar la bitacora primero!" <<
            endl;
            continue;
          }
        break;
        }
      case 3:
        cout << "¡Salida del sistema exitosa!" << endl;
        return 0;
        break;
    }
  }while(opcion != 3);
}
```

Análisis de complejidad

En esta actividad se trabajó con una hash table, específicamente un diccionario, llamado *unordered_map* en la STL de C++. La complejidad computacional de esta estructura de datos es de O(1) para los métodos de inserción, búsqueda y eliminación de datos. Esto se debe gracias a la hash function que se encarga de darnos acceso al índice en donde se encuentra el valor que guarda una llave única, que en este caso fue una IP. ¿Cómo esta función hash nos otorga una complejidad O(1)? Esto se debe a que la función hash devuelve un índice vacío en la hash table después de haber realizado un algoritmo que lo conforman operaciones aritméticas simples. Estas operaciones son suma, resta, división, multiplicación y el cálculo del módulo de algún valor. El acceso a un índice de la hash table es como utilizar el operador “[]” para acceder a los elementos de un arreglo.

Aunque teóricamente, el hash table dentro de lo que cabe puede ser O(n). En este caso se toma en cuenta el menor número de colisiones de hash. Una colisión se da cuando el algoritmo de la función hash devuelve un índice a un lugar de la hash table que ya está ocupado. Por lo que encontrar otro lugar vació disponible para guardar un arreglo puede hacer que debamos recorrer todos los espacios de la hash table de manera secuencial.