Laboratorio Hash y conjuntos disyuntos

Presentado por:
Santiago Botero Garcia
Santiago Hurtado Martínez
Oscar Santiago Merino Suarez

Profesor: Sebastián Camilo Martínez Reyes

Escuela Colombiana de Ingeniería Julio Garavito

LinkedList

Descripción del Problema:
El problema que enfrentamos es implementar una tabla hash que maneje colisiones utilizando una estructura de datos de lista enlazada para resolver las colisiones. La tabla hash debe permitir la inserción, eliminación y búsqueda eficiente de elementos, manteniendo un tiempo de ejecución aceptable incluso en el peor de los casos.

Estrategia de Solución:
Para manejar las colisiones en la tabla hash, utilizaremos una estrategia de "encadenamiento", donde cada entrada de la tabla hash contendrá una lista enlazada de elementos que han colisionado en la misma posición hash. Cuando ocurra una colisión, simplemente agregaremos el nuevo elemento a la lista enlazada correspondiente.

Entrada:
Elementos a insertar en la tabla hash.
Elementos a buscar en la tabla hash.
Elementos a eliminar de la tabla hash.

Salida:
Resultados de las operaciones de búsqueda (encontrado/no encontrado).
Resultados de las operaciones de eliminación (éxito/error si el elemento no existe).
Información sobre el estado final de la tabla hash.

Ventajas:
Manejo eficiente de colisiones: Las listas enlazadas permiten manejar un gran número de colisiones sin afectar significativamente el rendimiento.
Las listas enlazadas pueden crecer y encogerse dinámicamente según sea necesario, sin necesidad de predefinir un tamaño fijo para la tabla hash.

Desventajas:
Mayor consumo de memoria: Cada entrada de la tabla hash debe contener un puntero a una lista enlazada, lo que puede aumentar el consumo de memoria en comparación con otras estrategias de manejo de colisiones.
En el peor caso, cuando todas las claves tienen la misma función hash y colisionan en la misma posición, la búsqueda, inserción y eliminación pueden tomar tiempo lineal en el número total de elementos en la lista enlazada correspondiente.

```
from sys import stdin
from random import randint
from time import time

class Node:
    def __init__(self, key, value):
        self.key = key
        self.value = value
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def insert(self, key, value):
        new_node = Node(key, value)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node

    def search(self, key):
        current = self.head
        while current:
            if current.key == key:
                return current.value
            current = current.next
        return None

    def remove(self, key):
        if not self.head:
            return

        if self.head.key == key:
            self.head = self.head.next
            return

        prev = self.head
        current = self.head.next
        while current:
            if current.key == key:
                prev.next = current.next
                return
            prev = current
            current = current.next

class HashTable:
    def __init__(self, size):
        self.size = size
        self.elements = [LinkedList() for _ in range(size)]
```

```
    def hash(self, key):
        return hash(key) % self.size

    def insert(self, key, value):
        index = self.hash(key)
        self.elements[index].insert(key, value)

    def search(self, key):
        index = self.hash(key)
        return self.elements[index].search(key)

    def remove(self, key):
        index = self.hash(key)
        self.elements[index].remove(key)

    def printElements(self):
        for index in range(len(self.elements)):
            print(index, ': ', end='')
            current = self.elements[index].head
            while current:
                print('(', current.key, ',', current.value, ')', end=' ->
')
                current = current.next
            print('None')
```

Casos de Prueba:
Se agregaron entradas a la tabla hash, cada una conteniendo información sobre un país destino, la fecha de exportación y la marca de café exportado, asociada a su respectivo precio:

```
def main():
    # Datos de ejemplo para representar la exportación de café desde
Colombia hacia otros países
    export_data = [
        (('Francia', '11-06-2021', 'Café Colombiano SAS'), 4500),
        (('Italia', '10-06-2021', 'Gustos Italianos SRL'), 5000),
        (('Estados Unidos', '09-06-2021', 'Java & Co.'), 5200),
        (('Alemania', '08-06-2021', 'Kaffee GmbH'), 4800),
        (('Reino Unido', '07-06-2021', 'British Beans Ltd.'), 4900),
        (('Japón', '06-06-2021', 'Tokyo Roasters Inc.'), 5100),
        (('Canadá', '05-06-2021', 'Maple Beans Co.'), 4700),
        (('Australia', '04-06-2021', 'Aussie Breweries Pty Ltd.'), 5300),
        (('Brasil', '03-06-2021', 'Café Brasileiro SA'), 4600),
        (('España', '02-06-2021', 'Café Español SL'), 5400)
    ]

    t1 = time()
    hashtable = HashTable(13)
    for e in export_data:
        hashtable.insert(e[0], e[1])
    hashtable.printElements()
```

```
    t2 = time()
    print('Searching for ', ('Francia', '11-06-2021', 'Café Colombiano
SAS'), ': ',
          hashtable.search(('Francia', '11-06-2021', 'Café Colombiano
SAS')))
    print(f"El tiempo de ejecución fue: {t2 - t1}s")
```

Output:

```
   0 : None
   1 : ( ('Estados Unidos', '09-06-2021', 'Java & Co.') , 5200 ) -> None
   2 : None
1  3 : ( ('España', '02-06-2021', 'Café Español SL') , 5400 ) -> None
 2 4 : None
 3 5 : None
 4 6 : ( ('Alemania', '08-06-2021', 'Kaffee GmbH') , 4800 ) -> None
 5 7 : None
 6 8 : None
 7 9 : ( ('Francia', '11-06-2021', 'Café Colombiano SAS') , 4500 ) -> (
 8 ('Canadá', '05-06-2021', 'Maple Beans Co.') , 4700 ) -> None
 9 10 : None
10 11 : ( ('Australia', '04-06-2021', 'Aussie Breweries Pty Ltd.') , 5300
11 ) -> None
12 12 : ( ('Italia', '10-06-2021', 'Gustos Italianos SRL') , 5000 ) -> (
13 ('Reino Unido', '07-06-2021', 'British Beans Ltd.') , 4900 ) -> (
14 ('Japón', '06-06-2021', 'Tokyo Roasters Inc.') , 5100 ) -> (
15 ('Brasil', '03-06-2021', 'Café Brasileiro SA') , 4600 ) -> None
   Searching for  ('Francia', '11-06-2021', 'Café Colombiano SAS') :
   4500
   El tiempo de ejecución fue: 0.0009987354278564453s
```

Búsqueda binaria

Descripción del Problema:
El problema que enfrentamos es implementar una tabla hash que maneje colisiones utilizando un arreglo indexado con búsqueda binaria para resolver las colisiones. La tabla hash debe permitir la inserción, eliminación y búsqueda eficiente de elementos, manteniendo un tiempo de ejecución aceptable incluso en el peor de los casos.

Estrategia de Solución:
Para manejar las colisiones en la tabla hash, utilizaremos una estrategia de "re-hashing". Cuando ocurra una colisión, calcularemos una nueva posición utilizando una función de re-hash y luego insertaremos el elemento en la nueva posición. Utilizaremos un arreglo indexado para almacenar los elementos, y para resolver las colisiones en una misma posición, implementaremos una búsqueda binaria en el arreglo para encontrar la posición correcta de inserción.

Entrada:
Elementos a insertar en la tabla hash.
Elementos a buscar en la tabla hash.
Elementos a eliminar de la tabla hash.

Salida:
Resultados de las operaciones de búsqueda (encontrado/no encontrado).
Resultados de las operaciones de eliminación (éxito/error si el elemento no existe).
Información sobre el estado final de la tabla hash.
Ventajas y Desventajas:

Ventajas:
La búsqueda binaria tiene una complejidad de tiempo de O(log n), lo que garantiza un rendimiento eficiente incluso en el peor caso.

Utilizar un arreglo indexado puede consumir menos memoria que otras estructuras de datos, especialmente cuando el tamaño de la tabla hash es pequeño.

Desventajas:
La eficiencia de la búsqueda binaria depende del tamaño del arreglo.
Si el tamaño es demasiado grande, puede consumir mucha memoria; si es demasiado pequeño, puede haber muchas colisiones, lo que afectará el rendimiento.
Implementar la búsqueda binaria en un arreglo indexado puede ser más complejo que otras estrategias de manejo de colisiones.

```
from sys import stdin
from random import randint
from time import time

class HashTable:
    def __init__(self, size):
        self.size = size
        self.elements = [ [] for _ in range(size)]
```

```
    def hash(self, key):
        return hash(key) % self.size

    def insert(self, key, value):
        index = self.hash(key)
        entry = (key, value)
        self.elements[index].append(entry)
        self.elements[index].sort()

    def search(self, key):
        index = self.hash(key)
        entry_list = self.elements[index]
        left, right = 0, len(entry_list) - 1
        while left <= right:
            mid = (left + right) // 2
            if entry_list[mid][0] == key:
                return entry_list[mid][1]
            elif entry_list[mid][0] < key:
                left = mid + 1
            else:
                right = mid - 1
        return None

    def remove(self, key):
        index = self.hash(key)
        entry_list = self.elements[index]
        for i in range(len(entry_list)):
            if entry_list[i][0] == key:
                del entry_list[i]
                break

    def printElements(self):
        for index in range(len(self.elements)):
            print(index, ': ', self.elements[index])
```

Caso de Prueba:
Se agregaron entradas a la tabla hash, cada una conteniendo información sobre un país destino, la fecha de exportación y la marca de café exportado, asociada a su respectivo precio:

```
def main():
    new_data = [
        (('Francia', '11-06-2021', 'Café Parisien SAS'), 4500),
        (('Italia', '12-06-2021', 'Gusti Italiani SRL'), 5000),
        (('Estados Unidos', '13-06-2021', 'Java & Co.'), 5200),
        (('Alemania', '14-06-2021', 'Kaffee GmbH'), 4800),
        (('Reino Unido', '15-06-2021', 'British Beans Ltd.'), 4900),
        (('Japón', '16-06-2021', 'Tokyo Roasters Inc.'), 5100),
        (('Canadá', '17-06-2021', 'Maple Beans Co.'), 4700),
        (('Australia', '18-06-2021', 'Aussie Breweries Pty Ltd.'), 5300),
        (('Brasil', '19-06-2021', 'Café Brasileiro SA'), 4600),
        (('España', '20-06-2021', 'Café Español SL'), 5400)
```

```
    ]

    t1 = time()
    hashtable = HashTable(13)
    for e in new_data:
        hashtable.insert(e[0], e[1])
    hashtable.printElements()
    print('Searching for ', ('Francia', '11-06-2021', 'Café Parisien
SAS'), ': ', hashtable.search(('Francia', '11-06-2021', 'Café Parisien
SAS')))
    t2 = time()
    print("Tiempo de ejecución:", t2 - t1, "segundos")
```

Output:

```
   0 :  []
   1 :  [(('Alemania', '14-06-2021', 'Kaffee GmbH'), 4800)]
 1 2 :  []
  2 3 :  [(('Canadá', '17-06-2021', 'Maple Beans Co.'), 4700)]
  3 4 :  []
  4 5 :  [(('Australia', '18-06-2021', 'Aussie Breweries Pty Ltd.'),
  5 5300)]
  6 6 :  [(('Estados Unidos', '13-06-2021', 'Java & Co.'), 5200),
  7 (('Japón', '16-06-2021', 'Tokyo Roasters Inc.'), 5100)]
  8 7 :  [(('Francia', '11-06-2021', 'Café Parisien SAS'), 4500)]
  9 8 :  []
 10 9 :  [(('España', '20-06-2021', 'Café Español SL'), 5400), (('Reino
 11 Unido', '15-06-2021', 'British Beans Ltd.'), 4900)]
 12 10 :  []
 13 11 :  [(('Brasil', '19-06-2021', 'Café Brasileiro SA'), 4600),
 14 (('Italia', '12-06-2021', 'Gusti Italiani SRL'), 5000)]
 15 12 :  []
   Searching for  ('Francia', '11-06-2021', 'Café Parisien SAS') :  4500
   Tiempo de ejecución: 0.0 segundos
```

Búsqueda por profundidad

Descripción del Problema:
El problema es implementar una tabla hash que maneje colisiones utilizando una búsqueda por profundidad (DFS) para resolver las colisiones.
La tabla hash debe permitir la inserción, eliminación y búsqueda eficiente de elementos, manteniendo un tiempo de ejecución aceptable incluso en el peor de los casos.

Estrategia de Solución:
Para manejar las colisiones en la tabla hash, utilizaremos una estrategia de "búsqueda por profundidad (DFS)". Cuando ocurra una colisión, realizaremos una búsqueda por profundidad en una estructura de datos (como un árbol) asociada a la posición hash para encontrar una ubicación adecuada para insertar el nuevo elemento. Utilizaremos la DFS para explorar todas las posibles ubicaciones de inserción en la estructura de datos asociada a esa posición hash.

Entrada:
Elementos a insertar en la tabla hash.
Elementos a buscar en la tabla hash.
Elementos a eliminar de la tabla hash.

Salida:
Resultados de las operaciones de búsqueda (encontrado/no encontrado).
Resultados de las operaciones de eliminación (éxito/error si el elemento no existe).
Información sobre el estado final de la tabla hash.

Ventajas:
La búsqueda por profundidad garantiza que encontraremos una ubicación para insertar el nuevo elemento incluso en el peor de los casos.
La DFS puede adaptarse a diferentes estructuras de datos, lo que nos permite utilizarla con una variedad de estructuras de datos para manejar colisiones.

Desventajas:
La implementación de la DFS puede ser más compleja que otras estrategias de manejo de colisiones.
Dependiendo de la estructura de datos utilizada para la búsqueda por profundidad, puede haber un costo de rendimiento adicional en comparación con otras estrategias más simples.

```python
from sys import stdin
from random import randint
from time import time

WHITE = "white"
BLACK = "black"
GRAY = "gray"
import math

class HashTable:
    def __init__(self, size):
        self.elements = [{} for x in range(size)]  # Usamos diccionarios
para listas de adyacencias

    def getElements(self):
        return self.elements

    def printElements(self):
        for index in range(len(self.elements)):
            print(index, ': ', self.elements[index])

    def hash(self, key):
        return hash(key) % len(self.elements)

    def insert(self, key, value):
        index = self.hash(key)
        print('Inserting', value, 'with key', key, 'on index', index)
        # Insertamos en la lista de adyacencia correspondiente al índice
        self.elements[index][key] = value

    def search(self, key):
        index = self.hash(key)
        # Buscamos en la lista de adyacencia correspondiente al índice
        return self.elements[index].get(key, None)

    def update(self, key, value):
        index = self.hash(key)
        if key in self.elements[index]:
            self.elements[index][key] = value

    def delete(self, key):
        index = self.hash(key)
        if key in self.elements[index]:
            del self.elements[index][key]

    def dfs(self, index):
        def _dfs_visit(graph, vertex, visited):
            visited[vertex] = True
            print(f"Visited {vertex}: {graph[vertex]}")
            for neighbor in graph:
                if neighbor not in visited:
                    _dfs_visit(graph, neighbor, visited)
```

```
        # Obtenemos la lista de adyacencia del índice especificado
        graph = self.elements[index]
        visited = {}
        for vertex in graph:
            if vertex not in visited:
                _dfs_visit(graph, vertex, visited)
```

Caso de Prueba:

Se agregaron entradas a la tabla hash, cada una conteniendo información sobre un país destino, la fecha de exportación y la marca de café exportado, asociada a su respectivo precio retornando el tiempo de ejecución por cada consulta dfs:

```
def main():
    # Datos ficticios para representar la exportación de café desde
Colombia a otros países
    export_data = [
        (('Francia', '11-06-2021', 'Café Parisien SAS'), 4500),
        (('Italia', '12-06-2021', 'Gusti Italiani SRL'), 5000),
        (('Estados Unidos', '13-06-2021', 'Java & Co.'), 5200),
        (('Alemania', '14-06-2021', 'Kaffee GmbH'), 4800),
        (('Reino Unido', '15-06-2021', 'British Beans Ltd.'), 4900),
        (('Japón', '16-06-2021', 'Tokyo Roasters Inc.'), 5100),
        (('Canadá', '17-06-2021', 'Maple Beans Co.'), 4700),
        (('Australia', '18-06-2021', 'Aussie Breweries Pty Ltd.'), 5300),
        (('Brasil', '19-06-2021', 'Café Brasileiro SA'), 4600),
        (('España', '20-06-2021', 'Café Español SL'), 5400)
    ]

    t1 = time()
    hashtable = HashTable(13)
    for e in export_data:
        hashtable.insert(e[0], e[1])
    hashtable.printElements()
    print('Searching for ', ('Francia', '11-06-2021', 'Café Parisien
SAS'), ': ', hashtable.search(('Francia', '11-06-2021', 'Café Parisien
SAS')))
    print('Searching for ', ('Italia', '12-06-2021', 'Gusti Italiani
SRL'), ': ', hashtable.search(('Italia', '12-06-2021', 'Gusti Italiani
SRL')))
    print('Searching for ', ('Japón', '16-06-2021', 'Tokyo Roasters
Inc.'), ': ', hashtable.search(('Japón', '16-06-2021', 'Tokyo Roasters
Inc.')))
    hashtable.update(('Reino Unido', '15-06-2021', 'British Beans Ltd.'),
4950)
    print('Updated Search for ', ('Reino Unido', '15-06-2021', 'British
Beans Ltd.'), ': ', hashtable.search(('Reino Unido', '15-06-2021',
'British Beans Ltd.')))
    hashtable.delete(('Brasil', '19-06-2021', 'Café Brasileiro SA'))
    print('Post-delete Search for ', ('Brasil', '19-06-2021', 'Café
Brasileiro SA'), ': ', hashtable.search(('Brasil', '19-06-2021', 'Café
Brasileiro SA')))

    print("DFS on index 0:")
```

```
hashtable.dfs(0)
print("DFS on index 7:")
hashtable.dfs(7)

t2 = time()
print("El tiempo de ejecución fue", t2 - t1, "s")
```

Output:

```
0 : None
1 : ( ('Estados Unidos', '09-06-2021', 'Java & Co.') , 5200 ) -> None
2 : None
3 : ( ('España', '02-06-2021', 'Café Español SL') , 5400 ) -> None
4 : None
5 : None
6 : ( ('Alemania', '08-06-2021', 'Kaffee GmbH') , 4800 ) -> None
7 : None
8 : None
9 : ( ('Francia', '11-06-2021', 'Café Colombiano SAS') , 4500 ) -> (
('Canadá', '05-06-2021', 'Maple Beans Co.') , 4700 ) -> None
10 : None
11 : ( ('Australia', '04-06-2021', 'Aussie Breweries Pty Ltd.') , 5300
) -> None
12 : ( ('Italia', '10-06-2021', 'Gustos Italianos SRL') , 5000 ) -> (
('Reino Unido', '07-06-2021', 'British Beans Ltd.') , 4900 ) -> (
('Japón', '06-06-2021', 'Tokyo Roasters Inc.') , 5100 ) -> (
('Brasil', '03-06-2021', 'Café Brasileiro SA') , 4600 ) -> None
Searching for  ('Francia', '11-06-2021', 'Café Colombiano SAS') :
4500
El tiempo de ejecución fue: 0.0009987354278564453s

0 :  []
1 :  [(('Alemania', '14-06-2021', 'Kaffee GmbH'), 4800)]
2 :  []
3 :  [(('Canadá', '17-06-2021', 'Maple Beans Co.'), 4700)]
4 :  []
5 :  [(('Australia', '18-06-2021', 'Aussie Breweries Pty Ltd.'),
5300)]
6 :  [(('Estados Unidos', '13-06-2021', 'Java & Co.'), 5200),
(('Japón', '16-06-2021', 'Tokyo Roasters Inc.'), 5100)]
7 :  [(('Francia', '11-06-2021', 'Café Parisien SAS'), 4500)]
8 :  []
9 :  [(('España', '20-06-2021', 'Café Español SL'), 5400), (('Reino
Unido', '15-06-2021', 'British Beans Ltd.'), 4900)]
10 :  []
11 :  [(('Brasil', '19-06-2021', 'Café Brasileiro SA'), 4600),
(('Italia', '12-06-2021', 'Gusti Italiani SRL'), 5000)]
12 :  []
Searching for  ('Francia', '11-06-2021', 'Café Parisien SAS') :  4500
Tiempo de ejecución: 0.0 segundos

```

```
Inserting 4500 with key ('Francia', '11-06-2021', 'Café Parisien SAS')
on index 7
Inserting 5000 with key ('Italia', '12-06-2021', 'Gusti Italiani SRL')
on index 11
Inserting 5200 with key ('Estados Unidos', '13-06-2021', 'Java & Co.')
on index 6
Inserting 4800 with key ('Alemania', '14-06-2021', 'Kaffee GmbH') on
index 1
Inserting 4900 with key ('Reino Unido', '15-06-2021', 'British Beans
Ltd.') on index 9
Inserting 5100 with key ('Japón', '16-06-2021', 'Tokyo Roasters Inc.')
on index 6
Inserting 4700 with key ('Canadá', '17-06-2021', 'Maple Beans Co.') on
index 3
Inserting 5300 with key ('Australia', '18-06-2021', 'Aussie Breweries
Pty Ltd.') on index 5
Inserting 4600 with key ('Brasil', '19-06-2021', 'Café Brasileiro SA')
on index 11
Inserting 5400 with key ('España', '20-06-2021', 'Café Español SL') on
index 9
0 :  {}
1 :  {('Alemania', '14-06-2021', 'Kaffee GmbH'): 4800}
2 :  {}
3 :  {('Canadá', '17-06-2021', 'Maple Beans Co.'): 4700}
4 :  {}
5 :  {('Australia', '18-06-2021', 'Aussie Breweries Pty Ltd.'): 5300}
6 :  {('Estados Unidos', '13-06-2021', 'Java & Co.'): 5200, ('Japón',
'16-06-2021', 'Tokyo Roasters Inc.'): 5100}
7 :  {('Francia', '11-06-2021', 'Café Parisien SAS'): 4500}
8 :  {}
9 :  {('Reino Unido', '15-06-2021', 'British Beans Ltd.'): 4900,
('España', '20-06-2021', 'Café Español SL'): 5400}
10 :  {}
11 :  {('Italia', '12-06-2021', 'Gusti Italiani SRL'): 5000,
('Brasil', '19-06-2021', 'Café Brasileiro SA'): 4600}
12 :  {}
Searching for  ('Francia', '11-06-2021', 'Café Parisien SAS') :  4500
Searching for  ('Italia', '12-06-2021', 'Gusti Italiani SRL') :  5000
Searching for  ('Japón', '16-06-2021', 'Tokyo Roasters Inc.') :  5100
Updated Search for  ('Reino Unido', '15-06-2021', 'British Beans
Ltd.') :  4950
Post-delete Search for  ('Brasil', '19-06-2021', 'Café Brasileiro SA')
:  None
DFS on index 0:
DFS on index 7:
Visited ('Francia', '11-06-2021', 'Café Parisien SAS'): 4500
El tiempo de ejecución fue 0.00099945068359375 s

Inserting 4500 with key ('Francia', '11-06-2021', 'Café Parisien SAS')
on index 7
Inserting 5000 with key ('Italia', '12-06-2021', 'Gusti Italiani SRL')
on index 11
```

```
Inserting 5200 with key ('Estados Unidos', '13-06-2021', 'Java & Co.')
on index 6
Inserting 4800 with key ('Alemania', '14-06-2021', 'Kaffee GmbH') on
index 1
Inserting 4900 with key ('Reino Unido', '15-06-2021', 'British Beans
Ltd.') on index 9
Inserting 5100 with key ('Japón', '16-06-2021', 'Tokyo Roasters Inc.')
on index 6
Inserting 4700 with key ('Canadá', '17-06-2021', 'Maple Beans Co.') on
index 3
Inserting 5300 with key ('Australia', '18-06-2021', 'Aussie Breweries
Pty Ltd.') on index 5
Inserting 4600 with key ('Brasil', '19-06-2021', 'Café Brasileiro SA')
on index 11
Inserting 5400 with key ('España', '20-06-2021', 'Café Español SL') on
index 9
0 :  {}
1 :  {('Alemania', '14-06-2021', 'Kaffee GmbH'): 4800}
2 :  {}
3 :  {('Canadá', '17-06-2021', 'Maple Beans Co.'): 4700}
4 :  {}
5 :  {('Australia', '18-06-2021', 'Aussie Breweries Pty Ltd.'): 5300}
6 :  {('Estados Unidos', '13-06-2021', 'Java & Co.'): 5200, ('Japón',
'16-06-2021', 'Tokyo Roasters Inc.'): 5100}
7 :  {('Francia', '11-06-2021', 'Café Parisien SAS'): 4500}
8 :  {}
9 :  {('Reino Unido', '15-06-2021', 'British Beans Ltd.'): 4900,
('España', '20-06-2021', 'Café Español SL'): 5400}
10 :  {}
11 :  {('Italia', '12-06-2021', 'Gusti Italiani SRL'): 5000,
('Brasil', '19-06-2021', 'Café Brasileiro SA'): 4600}
12 :  {}
Searching for  ('Francia', '11-06-2021', 'Café Parisien SAS') :  4500
Searching for  ('Italia', '12-06-2021', 'Gusti Italiani SRL') :  5000
Searching for  ('Japón', '16-06-2021', 'Tokyo Roasters Inc.') :  5100
Updated Search for  ('Reino Unido', '15-06-2021', 'British Beans
Ltd.') :  4950
Post-delete Search for  ('Brasil', '19-06-2021', 'Café Brasileiro SA')
:  None
DFS on index 0:
DFS on index 7:
Visited ('Francia', '11-06-2021', 'Café Parisien SAS'): 4500
El tiempo de ejecución fue 0.0010004043579101562 s
```

Hash con conjuntos disyuntos

Descripción del Problema:
El problema es implementar una tabla hash que maneje colisiones utilizando conjuntos disyuntos para resolver las colisiones.
La tabla hash debe permitir la inserción, eliminación y búsqueda eficiente de elementos, manteniendo un tiempo de ejecución aceptable incluso en el peor de los casos.

Estrategia de Solución:
Utilizaremos una tabla hash con un tamaño predefinido.
Cada entrada de la tabla hash contendrá una lista de tuplas (clave, valor).
Cuando ocurra una colisión, utilizaremos la estructura de conjuntos disyuntos para unir los conjuntos correspondientes a las posiciones de las colisiones.
Esto permitirá mantener conjuntos de claves relacionadas entre sí, incluso si colisionan en la misma posición de la tabla hash.
Para buscar un elemento, calcularemos su posición hash y luego realizaremos una búsqueda binaria en la lista correspondiente. Si la búsqueda binaria no encuentra la clave en la lista, intentaremos encontrarla en el conjunto disyunto correspondiente.
Para insertar un elemento, primero calcularemos su posición hash y luego lo agregaremos a la lista correspondiente. Si ya existe en la lista, lo reemplazaremos. Luego, uniremos el conjunto disyunto de la posición hash con el conjunto disyunto de la posición original del elemento (si es diferente).
Para eliminar un elemento, primero lo buscaremos en la lista correspondiente. Si lo encontramos, lo eliminaremos de la lista. Luego, actualizaremos los conjuntos disyuntos si es necesario.

Entrada:
Elementos a insertar en la tabla hash.
Elementos a buscar en la tabla hash.
Elementos a eliminar de la tabla hash.

Salida:
Resultados de las operaciones de búsqueda (encontrado/no encontrado).
Resultados de las operaciones de eliminación (éxito/error si el elemento no existe).
Información sobre el estado final de la tabla hash y los conjuntos disyuntos.

Ventajas:
La utilización de conjuntos disyuntos garantiza que, incluso en el peor de los casos, donde todas las claves colisionan en la misma posición de la tabla hash, podremos encontrar y unir los conjuntos disyuntos de manera eficiente.

La implementación con conjuntos disyuntos permite manejar eficazmente colisiones y relaciones entre claves que colisionan en la misma posición.

La complejidad de tiempo de las operaciones como búsqueda, inserción y unión de conjuntos disyuntos suele ser aceptable, especialmente cuando se utilizan estructuras de datos eficientes.

Desventajas:
La implementación de la estructura de conjuntos disyuntos y su integración con la tabla hash puede ser más compleja en comparación con otras estrategias de manejo de colisiones.

Dependiendo de la implementación de los conjuntos disyuntos, puede haber un consumo adicional de memoria para almacenar la información de los conjuntos y sus relaciones.

La operación de unión de conjuntos disyuntos puede tener un costo adicional en términos de tiempo y recursos computacionales, especialmente si los conjuntos son grandes.
Esto puede afectar el rendimiento general de la tabla hash en operaciones que involucran colisiones frecuentes.

```
from sys import stdin
from random import randint
from time import time

class DisjointSets:

    def __init__(self, A):
        self.sets = [set([x]) for x in A]

    def getSets(self):
        return self.sets

    def findSet(self, x):
        for si in self.sets:
            if x in si:
                return si
        return None

    def makeSet(self, x):
        if self.findSet(x) is None:
            self.sets.append(set([x]))
            return self.sets[len(self.sets)-1]
        return self.findSet(x)

    def union(self, x, y):
        s1 = self.findSet(x)
        s2 = self.findSet(y)

        if s1 is None:
            s1 = self.makeSet(x)
        if s2 is None:
            s2 = self.makeSet(y)
        if s1 != s2:
            s3 = s1.union(s2)
            self.sets.remove(s1)
            self.sets.remove(s2)
            self.sets.append(s3)

    def connectedComponents(self, Arcs):
```

```
        for e in Arcs:
            self.union(e[0], e[1])
            print('After processing arc', e, self.sets)
        result = []
        for si in self.sets:
            if len(si) > 1:
                result.append(si)
        return result


class HashTable:
    def __init__(self, size):
        self.size = size
        self.elements = [ [] for _ in range(size)]
        self.disjoint_sets = DisjointSets([x for x in range(size)])

    def hash(self, key):
        return hash(key) % self.size

    def insert(self, key, value):
        index = self.hash(key)
        entry = (key, value)
        self.elements[index].append(entry)
        self.elements[index].sort()
        # Realiza una unión en los conjuntos disjuntos
        self.disjoint_sets.union(index, value)

    def search(self, key):
        index = self.hash(key)
        entry_list = self.elements[index]
        left, right = 0, len(entry_list) - 1
        while left <= right:
            mid = (left + right) // 2
            if entry_list[mid][0] == key:
                return entry_list[mid][1]
            elif entry_list[mid][0] < key:
                left = mid + 1
            else:
                right = mid - 1
        return None

    def remove(self, key):
        index = self.hash(key)
        entry_list = self.elements[index]
        for i in range(len(entry_list)):
            if entry_list[i][0] == key:
                del entry_list[i]
                break

    def printElements(self):
        for index in range(len(self.elements)):
            print(index, ': ', self.elements[index])
```

Caso de Prueba:
Se agregaron entradas a la tabla hash, cada una conteniendo información sobre un país destino, la fecha de exportación y la marca de café exportado, asociada a su respectivo precio:

```
def main():
    export_data = [
        (('Francia', '11-06-2021', 'Café Parisien SAS'), 4500),
        (('Italia', '12-06-2021', 'Gusti Italiani SRL'), 5000),
        (('Estados Unidos', '13-06-2021', 'Java & Co.'), 5200),
        (('Alemania', '14-06-2021', 'Kaffee GmbH'), 4800),
        (('Reino Unido', '15-06-2021', 'British Beans Ltd.'), 4900),
        (('Japón', '16-06-2021', 'Tokyo Roasters Inc.'), 5100),
        (('Canadá', '17-06-2021', 'Maple Beans Co.'), 4700),
        (('Australia', '18-06-2021', 'Aussie Breweries Pty Ltd.'), 5300),
        (('Brasil', '19-06-2021', 'Café Brasileiro SA'), 4600),
        (('España', '20-06-2021', 'Café Español SL'), 5400)
    ]

    t1 = time()
    hashtable = HashTable(13)
    for e in export_data:
        hashtable.insert(e[0], e[1])
    hashtable.printElements()
    print('Searching for ', ('Francia', '11-06-2021', 'Café Parisien
SAS'), ': ', hashtable.search(('Francia', '11-06-2021', 'Café Parisien
SAS')))
    t2 = time()
    print("Tiempo de ejecución:", t2 - t1, "segundos")
    print('Conjuntos disjuntos:', hashtable.disjoint_sets.getSets())
```

Output:

```
0 :  [(('Italia', '12-06-2021', 'Gusti Italiani SRL'), 5000)]
1 :  [(('Australia', '18-06-2021', 'Aussie Breweries Pty Ltd.'),
5300), (('Canadá', '17-06-2021', 'Maple Beans Co.'), 4700)]
2 :  []
3 :  [(('Japón', '16-06-2021', 'Tokyo Roasters Inc.'), 5100)]
4 :  [(('Brasil', '19-06-2021', 'Café Brasileiro SA'), 4600)]
5 :  []
6 :  [(('Francia', '11-06-2021', 'Café Parisien SAS'), 4500)]
7 :  []
8 :  [(('España', '20-06-2021', 'Café Español SL'), 5400)]
9 :  [(('Estados Unidos', '13-06-2021', 'Java & Co.'), 5200)]
10 :  []
11 :  []
12 :  [(('Alemania', '14-06-2021', 'Kaffee GmbH'), 4800), (('Reino
Unido', '15-06-2021', 'British Beans Ltd.'), 4900)]
Searching for  ('Francia', '11-06-2021', 'Café Parisien SAS') :  4500
Tiempo de ejecución: 0.0 segundos
```

Conjuntos disjuntos: [{2}, {5}, {7}, {10}, {11}, {4500, 6}, {0, 5000}, {5200, 9}, {4800, 12, 4900}, {3, 5100}, {1, 5300, 4700}, {4600, 4}, {8, 5400}]