Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2133 - Estructuras de Datos y Algoritmos

Informe Tarea 2

13 de junio de 2021

1º semestre 2021 - Profesores C. Gazali - Y. Eterovic

Vicente Espinosa - 17639247

Diseño función hashing

Pseudo-código Función:

```
Matriz* matriz_init(height, width)
{
    for(row = 0; row < height; row++)
    {
        for(col = 0; col < width; col++)
        {
            matriz[row][col][0] = random_number()
            matriz[row][col][1] = random_number()

        }
    }
}
```

Se usa el [0] para casillas de color blanco, y [1] para color negro

```
zobrist_hash(matrix, image, height, width)
{
    hash = 0
    for(row = 0; row < height; row++)
    {
        for(col = 0; col < width; col++)
        {
            if(image[row][col]== 127)
            {
                hash = hash XOR matrix[row][col][1]
            }
            else if(image[row][col]== 0)
            {
                hash = hash XOR matrix[row][col][0]
            }
        }
    }
    return hash
}



increment_hash(hash, matriz, new_column, new_row, new_dimention)
{
    for(col = 0; col < new_dimention; col++)
    {
        if(new_column[col] == 127)
        {
            hash = hash XOR matriz[new_dimention][col][1]
        }
        else if(new_column[col] == 0)
        {
            hash = hash XOR matriz[new_dimention][col][0]
        }
    }
    for(row = 0; row < new_dimention - 1; row++)
    {
        if(new_row[row] == 127)
        {
            hash = hash XOR matriz[row][new_dimention][1]
        }
```

```
        else if(new_row[row] == 0)
        {
            hash = hash XOR matriz[row][new_dimention][0]
        }
    }
}
```

Se usará **direccionamiento cerrado** simplemente para facilitar la complejidad de la solución, aunque probablemente sería más eficiente usar direccionamiento abierto. Dado lo anterior, no se necesitará tener una función de **resize**.

Como que todos los números de la matriz son generados de manera aleatoria y no tienen ningún tipo de relación entre ellos, no debería haber ninguna tendencia hacia algún número en particular. Luego, como el número que se usa para *mod* y calcular el index en que va cada hash es un numero primo, podemos asegurar la posibilidad de que el *mod* de dos hash distintos está minimizada. Por lo tanto, como se usa un numero primo, que es mayor a 2^{n^2}, pero que no es demasiado mayor, es la llave ideal para lograr una función de hash uniforme.

Link para referencia de este análisis

En caso de que suceda una colisión, dado que se usará direccionamiento cerrado, para la función de **probing** simplemente se revisará la casilla donde debería estar el elemento buscado, y si no está en ninguna de las ocurrencias, entonces se puede concluir que el elemento no está en la tabla.

El tamaño de la lista con los hashes será de el próximo numero primo superior a 2^{n^2}, al recibir una imagen original de tamaño nxn

Dado que se utilizará direccionamiento cerrado, se puede usar un **factor de carga** cercano a 1, por lo que se puede tener que $\frac{k}{2^{n^2}} \approx 1$, siendo k la cantidad de datos guardados.