udp FACULTAD DE INGENIERÍA Y CIENCIAS

Laboratorio 4:

Super Hasheador-Inador

Maximiliano Sanders
Profesor: Nicolás Boettcher
Profesor Laboratorio: Victor Manriquez
Ayudantes: Nicolás Pino
Sebastián Campos

Introducción:

Para la presente actividad se nos solicita la creación de un algoritmo hash que pueda competir con sus pares SHA1, SHA256 y MD5, para esto se solicita la creación del software siguiendo las siguientes reglas:

- El número de caracteres final del texto procesado, debe ser fijo y no menor de 25 caracteres. Este número no debe variar, aunque el texto de entrada sea más largo.
- Se requiere que el programa pueda recibir como entrada, tanto un *string* o texto mediante ***STDIN***, cómo también mediante archivos con múltiples entradas separadas por el salto de línea.
- Cualquier ligero cambio al texto de entrada, debe cambiar el resultado final del algoritmo.
- El procesamiento del texto de entrada, debe ser rápido, no debe tomar mucho tiempo, independiente de la cantidad de texto de entrada. * ***Se recomienda el uso de operaciones matemáticas al texto de entrada, a fin de optimizar la velocidad de procesamiento del texto de entrada****
- Adicionalmente se requiere que posea a lo menos dos opciones:
    - una que procese el texto de entrada y calcule el Hash de estas entradas (sea por STDIN, cómo mediante un archivo)
    - Y otra opción que sólo calcule la entropía del texto de entrada. En este último caso, debe arrojar mediante ***STDOUT*** que entropía posee cada texto de entrada (El formato de la salida a STDOUT, debe mostrar tanto el texto analizado, cómo la entropía calculada separada por un delimitador que permita diferenciar los diferentes campos).

Además de lo anterior, se nos solicita una prueba de rendimiento comparados los algoritmos anteriormente mencionados con nuestro hash, haciendo pruebas con entradas de tamaño: 1,10,20 y 50. Finalmente un cálculo de la entropía de cada algoritmo mencionado y se realiza un análisis.

Desarrollo:

Código:

El código hecho responde al nombre “Super Hasheador-Inador” se basa en 4 pilares:

1. Input
2. Conversión a número
3. Funciones matemáticas
4. Unión y salida

La lógica que tiene el código es recibir el *string* de entrada, convertir esa entrada a un número, al cual realizar múltiples operaciones matemáticas con números primos, para finalmente realizar la unión de los resultados de estas operaciones y convertirlo a la salida en un hash con diccionario propio. En específico, se ocupan los números primos por su propiedad de tener buenas probabilidades para evitar colisiones a la hora de aplicar operaciones con ellos.
Desarrollamos:

- Input:
  El código pregunta que se desea hacer, si calcular el hash por lectura de archivo o por consola, o también tiene la opción de calcular la entropía del hash resultante con las mismas opciones anteriores.

```
Super Hasheador-Inador
Escoger lo que se desea hashear
(1) Por Archivo
(2) Por Input por consola
(3) En caso de calcular entropia
3
Calculadora de entropia
Ingrese opcion para calcular entropia del hash resultante:
(1) Archivo
(2) Input por consola
```

Menú del código

- Conversión a número:
  Una vez ingresado el *string* ya sea por consola o por una lectura de archivo, se procede a crear un arreglo del tamaño del *string* con los caracteres de este, para luego transformar cada carácter a número y sumarlo.

```
//suma del string
int n = input.length();
int arr[n];
for (int i = 0; i < n; i++){
    arr[i] = (int)input[i];
}
int sum = 0;
for (int i = 0; i < n; i++){
    sum += arr[i];
}
```

Conversión a número y suma

- Funciones matemáticas:
  Una vez hecha la suma de los valores de los caracteres, se continúa con las funciones matemáticas. Cada función recibe la suma anteriormente calculada, y en base a eso, se realizan las operaciones correspondientes. Antes, se establecen los siguientes números:

```
#define A 895932
#define B 909525
#define C 378632
#define D 420921
```

Variables definidas

Una vez establecidos estos valores, se procede con las funciones:

- Función Hash 1:

```
unsigned long long funcionHash1(int in){
    unsigned long long result = 0;
    result = (in * A)^ C;
    return result;
}
```

Función hash 1

Esta función recibe el número y multiplica éste por A (895.932), luego lo eleva a C (378.632), finalmente devuelve el resultado de esta operación.

- Función Hash 2:

```
unsigned long long funcionHash2(int in){
    unsigned long long result = 0;
    result = (in * B)* D;
    return result;
}
```

Función hash 2

Esta función recibe el número y lo multiplica por B(909.525) para luego multiplicarlo por D(420.921), finalmente devuelve el resultado de esta operación.

- Función Hash 3:

```
unsigned long long funcionHash3(int in){
    srand(in);
    unsigned long long number = (rand() % 6320430) + 227832;
    return number;
}
```

Función hash 3

Esta función recibe el número y lo establece como semilla para la función srand, para que luego se encuentre un número aleatorio entre 227.832 y 6.320.430, finalmente devuelve el resultado de esta operación.

- Función Hash 4:

```
unsigned long long funcionHash4(int in){
    unsigned long long result = 0;
    time_t now = time(0);
    tm *ltm = localtime(&now);
    int year = 1900 + ltm->tm_year;
    int month = 1 + ltm->tm_mon;
    int day = ltm->tm_mday;
    result = ((in)*month)*year;
    return result;
}
```

Función hash 4

Esta función recibe el número y realiza las operaciones con la fecha actual, en específico multiplica el número con el mes y luego lo multiplica por el año. Finalmente devuelve el resultado de esta operación.

- Función Hash 5:

```
unsigned long long funcionHash5(int in){
    unsigned long long result = 0;
    time_t now = time(0);
    tm *ltm = localtime(&now);
    srand(in);
    int day = ltm->tm_mday;
    unsigned long long number = (rand() % 4053946) + 2098960 ;
    result = (in * number) + (A+B);
    return result;
}
```

Función hash 5

Esta función recibe el número, lo establece como semilla para el srand(), luego encuentra un número entre 2.098.960 y 4.053.946. Una vez obtenido el número aleatorio, se multiplica el número recibido por el número aleatorio y se le suma a esto el resultado de la suma de A más B. Finalmente se devuelve el resultado de esta operación.

- Función Hash 6:

```
unsigned long long funcionHash6(int in){
    unsigned long long result;
    result = (in * D) - C;
    result = (result^A)+B;
    return result;
}
```

Función hash 6

Esta función recibe el número y lo multiplica por por D(420921) y le resta C(378632), luego lo eleva a A(895932) y al resultado de esto le suma B(909525). Finalmente se devuelve el resultado de esta operación.

- Unión:
  Una vez realizada todas las operaciones, se transforman en *string* y se almacenan en variables. Se unen en una variable de la siguiente forma:
  - función hash 1
  - función hash 3
  - la suma del *string*
  - función hash 4
  - función hash 2
  - función hash 5
  - función hash 6

```
//operaciones matematicas locas
unsigned long long op1 = funcionHash1(sum);
unsigned long long op2 = funcionHash2(sum);
unsigned long long op3 = funcionHash3(sum);
unsigned long long op4 = funcionHash4(sum);
unsigned long long op5 = funcionHash5(sum);
unsigned long long op6 = funcionHash6(sum);

//union y comprobacion de tamaño
string hasheado = to_string(op1) + to_string(op3) +
                  to_string(sum) + to_string(op4) +
                  to_string(op2) + to_string(op5) +
                  to_string(op6);
```

Unión

Una vez unidas todas las operaciones y la suma, se comprueba el largo del *string* resultante, si es mayor a 50, se corta con los primeros 50 caracteres.

```
if (hasheado.length() > 50){
    hasheado = hasheado.substr(0,50);
}
```

Comprobación del largo

- Salida:
  Una vez que tenemos el *string* resultante, se procede a la salida del hash. Para esto se establece el siguiente diccionario:

```
//Diccionario propio para el hash
string Diccionario[62] = {
    "0","1","2","3","4","5","6","7","8","9",
    "A","B","C","D","E","F","G","H","I","J",
    "K","L","M","N","O","P","Q","R","S","T",
    "U","V","W","X","Y","Z",
    "a","b","c","d","e","f","g","h","i","j",
    "k","l","m","n","o","p","q","r","s","t",
    "u","v","w","x","y","z"
};
```

Diccionario

Luego se procede a crear sub arreglos del *string* resultante, de tamaño 2:

```
//Convertir en arreglos de tamaño 2
string div[25];
int c = 0;
for (int i = 0; i < 25; i++){
    div[i] = hasheado.substr(c,2);
    c = c + 2;
}
```

subarreglos de tamaño 2 en string

Después se crea un arreglo de enteros, donde se transforman los números de *string* a números enteros.

```
int w[25];
for (int i = 0; i < 25; i++){
    w[i] = stoi(div[i]);
}
```

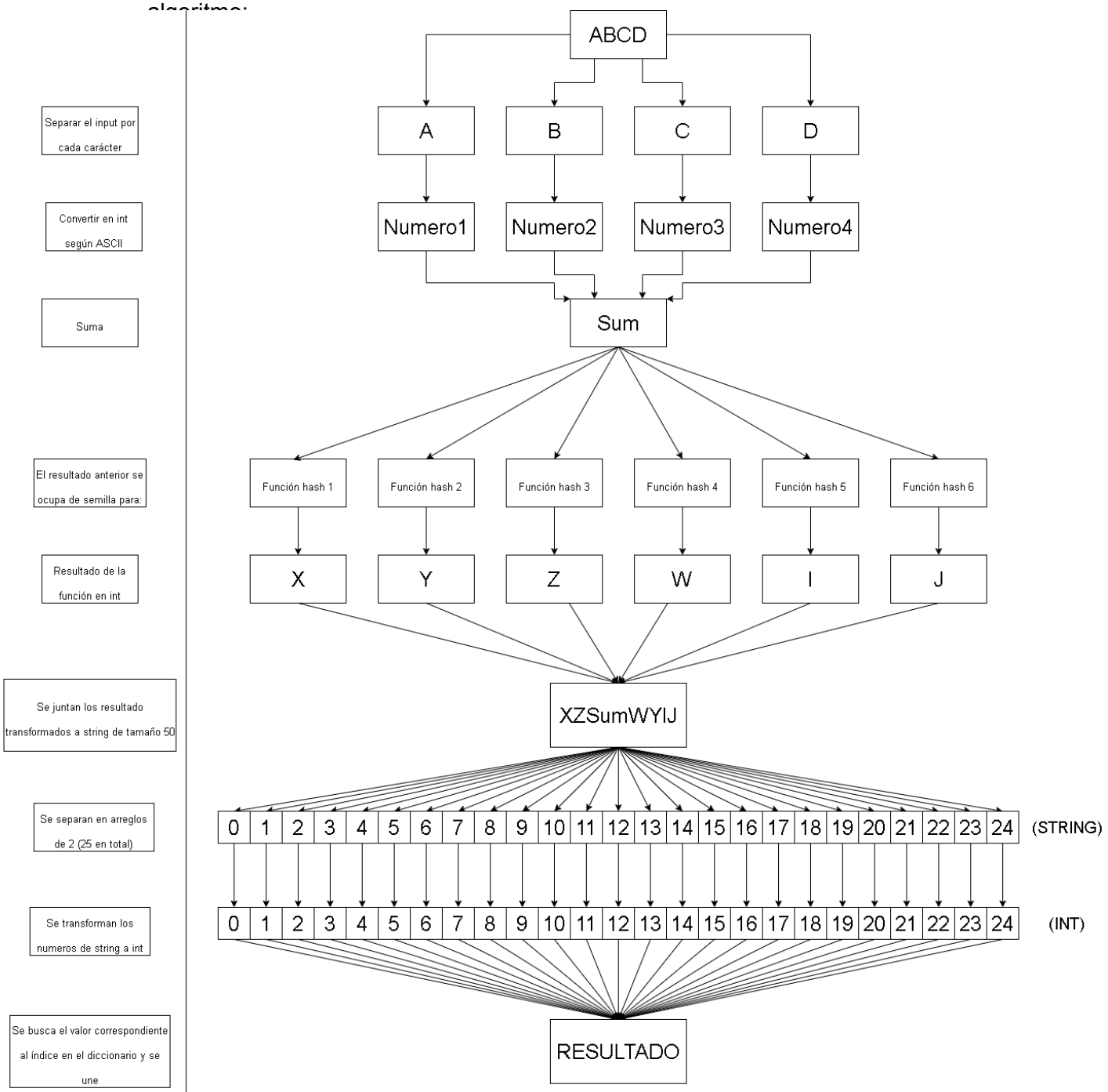
subarreglos de tamaño 2 convertido a int

Finalmente, para el resultado final, se establece una variable de tipo *string* . Para rellenar esta variable, se busca en el diccionario dependiendo del valor que toma el entero del arreglo anteriormente convertido. Como cabe la posibilidad de que el número del arreglo sea mayor al tamaño del diccionario (62), se establece la condición que si el valor presente en el arreglo es mayor a 62, se le resta a este el tamaño del diccionario para así dejar el valor dentro del rango de este. Para finalizar, se envía el resultado.

```
//Buscar el valor segun el diccionario
string result = "";
for (int i = 0; i < 25; i++){
    int index = w[i];
    if (index > 62){
        index = index - 63;
    }
    result = result + Diccionario[index];
}
return result;
```

Unión de cada valor correspondiente al diccionario

A continuación se muestra un diagrama para el mostrar el funcionamiento del algoritmo:



Prueba de rendimiento:

Para la prueba de rendimiento se establecen los siguientes archivos de entrada:

- Para una entrada: se ingresa por consola la primera entrada del archivo rockyou.txt
- Para 10 entradas: datos10.txt (10 primeras líneas del archivo rockyou.txt)
- Para 20 entradas :datos20.txt (20 primeras líneas del archivo rockyou.txt)
- Para 50 entradas: datos50.txt (50 primeras líneas del archivo rockyou.txt)

Los resultados del rendimiento de cada algoritmo se muestran en la siguiente tabla:

	1 entrada	10 entradas	20 entradas	50 entradas
MD5	0,000035	0,000058	0,000102	0,0001440
SHA1	0,000044	0,000175	0,000063	0,000066
SHA256	0,0001219	0,000057	0,000155	0,000127
SuperHasheador	0.0000006920	0.0000007310	0.0000009870	0.0000011430

Prueba de entropía:

Para calcular la entropía, ocupamos la función correspondiente:

$$E = L * log_2(W)$$

Con:

L = largo de la cadena

W = base utilizada

Los resultados se presentan en la siguiente tabla:

	Largo	Base	Entropía
MD5	32	16	128
SHA1	40	16	160
SHA256	64	16	256
SuperHasheador	25	62	148,8549078

Análisis de las pruebas:

Dados los resultados de las pruebas anteriores podemos identificar:

A. El hash creado es mucho más rápido que el resto de competidores, esto se debe a que los otros algoritmos poseen vueltas o repeticiones de procesos, mientras que nuestro código no posee repeticiones de ningún tipo, además cabe destacar que la entropía calculada es mayor que MD5, pero menor que SHA1 Y SHA256. Esto puede dar pie a que nuestro algoritmo sea bueno para la comprobación de estructura de archivos donde no se requiere tanta entropía pero sí velocidad de procesamiento.