INF-253 Lenguajes de Programación
Tarea 2: C

12 de abril de 2022

1. Bienvenido a Po Inter City

Patode Hule, tras su arduo trabajo en la ciudad Pun Toycoma, decidió irse a otra ciudad, puesto que descubrió que no era el lugar donde la gustaría vivir. Entre las diversas ciudades que encontró en el camino, descubrió Po Inter City. Al entrar a la ciudad, se dio cuenta que podía mover los punteros como Patode quisiera, esto hizo que se obsesionara con los punteros y la memoria en C, decidiendo hacer su primer compresor de información. Él emocionado les pide ayuda a ustedes para realizar su idea, puesto que sabe que son buenos informáticos.

2. Compresión de números en bits

Dentro de la informática, a veces tener un conjunto de números guardado tal cual no es útil, por lo que se opta por otras representaciones que permitirían reducir el uso del espacio. En está tarea conocerán lo que es la compresión por bloques. El objetivo de está es que cuando tener un conjunto de números inicial, dividirlo en grupos de números y aplicar una compresión distinta a cada grupo, eligiendo siempre la mejor.

Para esto se presentará la información de la siguiente forma:

1. Métodos de compresión
2. Cómo dividir y elegir el mejor método de compresión
3. Operaciones aplicables al conjunto comprimido

2.1. Métodos de compresión

2.1.1. Representación en unario

Un número en unario corresponde una secuencia de 1's de largo el número en cuestión, por ejemplo 8 se representaría cómo 11111111. La representación en unario entonces corresponde escribir cada número del conjunto en unario, separándolo con un 0 entre cada número. Por ejemplo, el conjunto de números $\{8, 2, 8, 8, 2, 3\}$ se representaría de la siguiente forma 111111110110111111110111111110110111.

2.1.2. Representación elegida según frecuencia

Esta representación corresponde a elegir de que manera escribir cada número en unario, donde la manera que ocupe menos bits será otorgada al número de mayor frecuencia y la que ocupe más bits será asignada al número con menor frecuencia. Luego, se reescribe cada número según su nueva representación en unario separando cada número por un 0, generando igualmente una secuencia de bits que permitirá recordar cual es la representación de cada número. Para esto se usará una representación con el siguiente formato: el número en unario, seguido de un 0 y seguido por su representación; en caso que exista otro número se debe poner un 0 y luego seguir con la misma forma.

Por ejemplo, si se tiene el conjunto de números $\{8, 2, 8, 8, 2, 3\}$. Primero se calcula la frecuencia de cada número, obteniendo que: el 8 se encuentra tres veces, el 2 se encuentra dos veces y el 3 se encuentra una vez, por lo tanto al 8 se le asignará la codificación 1, al 2 la codificación 11 y al 3 la codificación 111, generando así la siguiente representación del conjunto de números original: 101101010110111. Y la secuencia de bits que permitirá recordar la representación de cada número será 111111110101101101110111, en esta secuencia primero se observa el número 8 con su representación, siguiéndole el 2 con su representación y terminando con el 3 con su representación.

2.1.3. Representación incremental

Esta representación tiene la característica de basarse en la cantidad de números diferentes dentro del conjunto, donde este le otorgará una representación que siempre termina en un 0 excepto por el último número, donde este último tendrá una secuencia de solamente 1's de largo la cantidad de números distintos dentro del conjunto menos 1. Además, la representación de cada número será otorgada según el primero que se encuentre de izquierda a derecha. Al igual que la representación anterior se requiere una segunda secuencia para poder recordar qué representación corresponde a cada número, esta sigue el siguiente formato: el número en unario, seguido de un 0 y seguido por su representación; en caso que exista otro número debe seguirlo con la misma forma. Por último se necesita una tercera secuencia que permita determinar cuantos números distintos tiene el conjunto, está estará en binario.

Por ejemplo, si se tiene el conjunto de números $\{8, 2, 8, 8, 2, 3\}$. Primero, se encuentra el número 8 por lo que se otorga la representación 0, el segundo número encontrado es 2 al cual se le otorga la representación 10 y finalmente el tercer número sería 3 el cual al ser el último número se le otorga la representación 11. Dado esto la representación sería 010001011. La secuencia que servirá para recordar la representación de cada número será 111111110011010111011. Y por último la cantidad de números distintos en binario sería 11.

2.2. Cómo dividir y elegir el mejor método de compresión

Para dividir el conjunto en grupos de números se asignará un entero g que indica de que tamaño será cada grupo, si la cantidad de números no es divisible por g, entonces el último grupo tendrá lo que sobre del conjunto. Al tener dividido en grupos el conjunto, se debe elegir la mejor representación; para esto cada representación tendrá asignada una función que determinará cuanto espacio utilizaría, finalmente se debe elegir la representación que utilice menos bits.

Las funciones para calcular la cantidad de bits son:

- **bits_unario**: se debe sumar cuanto usaría cada número del conjunto en unario, más los 0's para poder separar los números que correspondería a la cantidad de números -1. En el ejemplo utilizado en la sección 2.1.1 sería 31 bits.
- **bits_frec**: se debe sumar cuanto usaría cada número del conjunto con su representación, más los 0's para poder separar los números que correspondería a la cantidad de núme-

ros -1, además sumar la cantidad de 1's y 0's que tiene la secuencia para recordar la representación de cada número. En el ejemplo utilizado en la sección 2.1.2 sería 39 bits.

- **bits_inc**: se debe sumar cuanto usaría cada número del conjunto con su representación, además sumar la cantidad de 1's y 0's que tiene la secuencia para recordar la representación de cada número, agregando la cantidad de 1's. y 0's en el número binario par la cantidad de números distintos. En el ejemplo utilizado en la sección 2.1.3 sería 32 bits.

Además, cada representación consta de una función que entrega el conjunto de números comprimido:

- **comprimir_en_unario**: recibe el conjunto de números y retorna el conjunto comprimido en unario
- **comprimir_en_frec**: recibe el conjunto de números y retorna el conjunto comprimido en elegir según frecuencia
- **comprimir_en_inc**: recibe el conjunto de números y retorna el conjunto comprimido en incremental

2.3. Operaciones aplicables

Dado que tener el conjunto comprimido es útil poder realizar operaciones sobre él. Estas son las siguientes funciones a implementar:

- **comprimir**: recibe un conjunto de números y debe retornar el conjunto comprimido.
- **descomprimir_en_unario**: recibe un conjunto comprimido en unario y retorna el conjunto descomprimido
- **descomprimir_en_frec**: recibe un conjunto comprimido en elegir según frecuencia y retorna el conjunto descomprimido
- **descomprimir_en_inc**: recibe un conjunto comprimido en incremental y retorna el conjunto descomprimido
- **descomprimir**: recibe el conjunto total y retorna el conjunto descomprimido
- **donde_esta_unario**: recibe el conjunto comprimido en unario, un número e y un número i que corresponde a la i-ésima aparición de e dentro del conjunto. Debe retornar la posición de la i-ésima aparición del número e dentro del conjunto en unario. En caso que no exista la i-ésima aparición se debe retornar -1.
- **donde_esta_frec**: recibe el conjunto comprimido en elegir según frecuencia, un número e y un número i que corresponde a la i-ésima aparición de e dentro del conjunto. Debe retornar la posición de la i-ésima aparición del número e dentro del conjunto en unario. En caso que no exista la i-ésima aparición se debe retornar -1.
- **donde_esta_inc**: recibe el conjunto total comprimido en incremental, un número e y un número i que corresponde a la i-ésima aparición de e dentro del conjunto. Debe retornar la posición de la i-ésima aparición del número e dentro del conjunto en incremental. En caso que no exista la i-ésima aparación se debe retornar -1.
- **donde_esta**: recibe el conjunto total comprimido, un número e y un número i que corresponde a la i-ésima aparición de e dentro del conjunto. Debe retornar la posición de la i-ésima aparición del número e dentro del conjunto total comprimido. En caso que no exista la i-ésima aparición se debe retornar -1.

- **cuantos_mas_grande_unario**: recibe el conjunto comprimido en unario y un número e. Debe retornar cuantos números son más grande que el número e dentro del conjunto respectivo.
- **cuantos_mas_grande_frec**: recibe el conjunto comprimido en elegir según frecuencia y un número e. Debe retornar cuantos números son más grande que el número e dentro del conjunto respectivo.
- **cuantos_mas_grande_inc**: recibe el conjunto total comprimido en incremental y un número e. Debe retornar cuantos números son más grande que el número e dentro del conjunto respectivo.
- **cuantos_mas_grande**: recibe el conjunto total comprimido y un número e. Debe retornar cuantos números son más grande que el número e dentro del conjunto respectivo.
- **mostrar_unario**: recibe el conjunto comprimido en unario. Debe mostrar por consola los bits de esta representación con el siguiente formato:

```
UNARIO:
bits
```

- **mostrar_frec**: recibe el conjunto comprimido en elegir según frecuencia. Debe mostrar por consola los bits de esta representación.

```
SEGUN FRECUENCIA:
bits
representaciones
```

- **mostrar_inc**: recibe el conjunto comprimido en incremental. Debe mostrar por consola los bits de esta representación.

```
INCREMENTAL:
bits
representaciones
n
```

- **mostrar**:recibe el conjunto total comprimido. Debe mostrar por consola los bits de cada grupo comprimido.
- **bits_total**: recibe el conjunto total comprimido. Debe retornar la cantidad de bits utilizados por ese conjunto comprimido.

2.4. Structs a utilizar

Para realizar esto se debe hacer uso de los siguientes structs:

Código 1: Struct de la representación en unario

```c
typedef struct Unario {
    // arreglo de '0' y '1'
    char* bits;
} Unario;
```

Código 2: Struct de la representación elegida según frecuencia

```c
typedef struct Frec {
    // arreglo de '0' y '1'
    char* bits;
    // arreglo para recordar que representacion tiene cada numero
    char* representaciones;
} Frec;
```

Código 3: Struct de la representación incremental

```
typedef struct Inc {
    // arreglo de '0' y '1'
    char* bits;
    // arreglo para recordar que representacion tiene cada numero
    char* representaciones;
    // cantidad de numeros distintos en el conjunto
    char* n;
} Inc;
```

Código 4: Struct de representación genérico

```
typedef struct Representacion {
    // puntero a la representacion creada
    void* representacion;

    // punteros a funciones
    int* (*su_descomprimir)(void*);
    int (*su_donde_esta)(void*, int, int);
    int (*su_cuanto_mas_grande)(void*, int, int);
    int (*su_bits)(void*);
    void (*su_imprimir)(void*)
} Representacion;
```

Código 5: Struct de representación genérico

```
typedef struct Conjunto_comprimido {
    // arreglo de representaciones de grupos de numeros dentro del conjunto
    void* representaciones;
} Conjunto_comprimido;
```

3. Funcionamiento del programa

Su programa debe primero pedir dos números c y g, que corresponden a la cantidad de números en el conjunto y la cantidad de números por grupo en el conjunto respectivamente, luego debe pedir el conjunto de números. Al realizar esto, su programa debe utilizar la función comprimir que debe retornar un puntero al struct del código 5.

Con el conjunto ya comprimido debe realizar un menú que permita preguntar por:

- Mostrar por pantalla el conjunto descomprimido
- Hacer uso de la función **donde_esta**, mostrando por consola el retorno entregado
- Hacer uso de la función **cuantos_mas_grande**, mostrando por consola el retorno entregado
- Hacer uso de la función **bits**, mostrando por consola el retorno entregado
- Hacer uso de la función **mostrar**
- Acceder a un grupo de números comprimido:
  - Hacer uso de su función **su_donde_esta**, mostrando por consola el retorna entregado
  - Hacer uso de su función **su_cuantos_mas_grande**, mostrando por consola el retorna entregado
  - Hacer uso de su función **su_bits**, mostrando por consola el retorna entregado
  - Hacer uso de su función **su_mostrar**

4. Ejemplo de compresión

Se ingresa $c = 30$ y $g = 10$, el conjunto de números corresponderá a $\{1, 1, 1, 1, 1, 1, 2, 3, 1, 1, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 1, 2, 3, 9, 16, 16, 16, 16, 16, 16\}$. Se divide el conjunto en grupos de 6, entregando los siguientes grupos $g_1 = \{1, 1, 1, 1, 1, 1, 2, 3, 1, 1\}$, $g_2 = \{8, 8, 8, 8, 8, 8, 8, 8, 8, 8\}$ y $g_3 = \{1, 2, 3, 9, 16, 16, 16, 16, 16, 16\}$.

- Para g_1: la representación en unario utilizaría 22 bits, la representación elegida según frecuencia utilizaría 39 bits y la representación incremental utilizaría 28 bits; por lo tanto se elige la representación en unario para g_1.
- Para g_2: la representación en unario utilizaría 89 bits, la representación elegida según frecuencia utilizaría 29 bits y la representación incremental utilizaría 21 bits; por lo tanto se elige la representación incremental para g_2.
- Para g_3: la representación en unario utilizaría 120 bits, la representación elegida según frecuencia utilizaría 76 bits y la representación incremental utilizaría 87 bits; por lo tanto se elige la elegida según frecuencia para g_2.

5. Resumen de las funciones a programar

Estas corresponden a las firmas de las funciones descritas en las secciones 2.2 y 2.3.

```
// 'n' representa la cantidad de numeros dentro del arreglo 'grupo'
Unario* comprimir_en_unario(int n, int* grupo);
Frec* comprimir_en_frec(int n, int* grupo);
Inc* comprimir_en_inc(int n, int* grupo);
// 'c' representa la cantidad de numeros dentro del arreglo 'numeros' y 'g'
    corresponde a la cantidad de numeros por 'grupo'
Conjunto_comprimido* comprimir(int c, int* numeros, int g);

int* descomprimir_en_unario(void* unario);
int* descomprimir_en_frec(void* frec);
int* descomprimir_en_inc(void* inc);
int* descomprimir(Conjunto_comprimido* conjunto_comprimido);

int donde_esta_unario(void* unario, int e, int i);
int donde_esta_frec(void* frec, int e, int i);
int donde_esta_inc(void* inc, int e, int i);
int donde_esta(Conjunto_comprimido* conjunto_comprimido, int e, int i);

int cuantos_mas_grande_unario(void* unario, int e);
int cuantos_mas_grande_frec(void* frec, int e);
int cuantos_mas_grande_inc(void* inc, int e);
int cuantos_mas_grande(Conjunto_comprimido* conjunto_comprimido, int e);

int bits_unario(void* unario);
int bits_frec(void* frec);
int bits_inc(void* inc);
int bits_total(Conjunto_comprimido* conjunto_comprimido);

void mostrar_unario(void* unario);
void mostrar_frec(void* frec);
void mostrar_inc(void* inc);
void mostrar(Conjunto_comprimido* conjunto_comprimido);
```

NOTA: Para las funciones **donde_esta**, **donde_esta_unario**, **donde_esta_frec**, **donde_esta_inc**, **cuantos_mas_grande**, **cuantos_mas_grande_unario**, **cuantos_mas_grande_frec** y **cuantos_mas_grande_inc** no se puede usar la función descomprimir.

6. Archivos a Entregar

Resumiendo lo anterior, los archivos mínimos a entregar son:

- README.txt
- Unario.c
- Unario.h
- Frec.c
- Frec.h
- Inc.c
- Inc.h
- main.c
- MAKEFILE

El archivo Unario.h debe tener todas las declaraciones de funciones que tengan relación con la representación en unario. El archivo Unario.c debe tener todas las implementaciones de las funciones que tengan relación con la representación en unario.

El archivo Frec.h debe tener todas las declaraciones de funciones que tengan relación con la representación en elegir según frecuencia. El archivo Frec.c debe tener todas las implementaciones de las funciones que tengan relación con la representación en elegir según frecuencia.

El archivo Inc.h debe tener todas las declaraciones de funciones que tengan relación con la representación en incremental. El archivo Inc.c debe tener todas las implementaciones de las funciones que tengan relación con la representación en incremental.

main.c debe contener el main del programa que haga todo lo pedido en la sección 3.

7. Sobre Entrega

- El código debe venir **indentado y ordenado**.
- Las funciones deberán ir comentadas, explicando clara y brevemente lo que realiza, los parámetros que recibe y los que devuelve (en caso de que devuelva algo). Se deja libertad al formato del comentario.
- Debe estar presente el archivo MAKEFILE para que se efectúe la revisión, este debe compilar **TODOS** los archivos.
- Se debe trabajar de forma individual.
- **La entrega debe realizarse en tar.gz y debe llevar el nombre: Tarea2LP_RolIntegrante.tar.gz**
- **El archivo README.txt debe contener nombre y rol del alumno e instrucciones detalladas para la compilación y utilización de su programa.**
- La entrega será vía aula y el plazo máximo de entrega es hasta el **29 de abril de 2022 a las 23:55** **hora aula**.
- Las copias serán evaluadas con nota 0 y se informarán a las respectivas autoridades.
- Si el makefile no está bien realizado, la tarea no se revisará.
- Se utilizará Valgrind para detectar los leak de memoria.

8. Calificación

Todos comienzan con nota máxima y se les irá descontando por cada punto omitido (el puntaje que aparece al lado es el máximo de puntos que se les descontará en caso de incumplimiento):

- Cada función tendrá su puntaje asignado y tendrá la siguiente rúbrica, teniendo f como puntos de función máxima:
  - No tiene la función programada o su código no tiene relación con lo descrito en la tarea (0 pts)
  - El código de la función es lógico con lo descrito, pero no se puede utilizar por algún otro motivo (MAX $f/4$ pts)
  - La función programada hace lo requerido en ciertos casos, pero no en su totalidad (MAX $f/2$ pts)
  - La función programada hace todo lo requerido y funciona en su totalidad (MAX f pts)
- Funciones:
  - **comprimir_en_unario** (6 pts)
  - **comprimir_en_frec** (6 pts)
  - **comprimir_en_inc** (6 pts)
  - **comprimir** (4 pts)
  - **descomprimir_en_unario** (6 pts)
  - **descomprimir_en_frec** (6 pts)
  - **descomprimir_en_inc** (6 pts)
  - **descomprimir** (4 pts)
  - **donde_esta_unario** (4 pts)
  - **donde_esta_frec** (4 pts)
  - **donde_esta_inc** (4 pts)
  - **donde_esta** (2 pts)
  - **cuantos_mas_grande_unario** (4 pts)
  - **cuantos_mas_grande_frec** (4 pts)
  - **cuantos_mas_grande_inc** (4 pts)
  - **cuantos_mas_grande** (2 pts)
  - **mostrar_unario** (3 pts)
  - **mostrar_frec** (3 pts)
  - **mostrar_inc** (3 pts)
  - **mostrar** (1 pts)
  - **bits_unario** (2 pts)
  - **bits_frec** (2 pts)
  - **bits_inc** (2 pts)
  - **bits_total** (1 pts)
- Main (11 pts)

- No permite pedir los números descritos en la sección 3, no se puede utilizar las funciones o se cae durante su ejecución (0 pts)
- Permite hacer hacer uso de la mitad de las funcionalidades pedidas (6 pts)
- Se ejecuta correctamente y se pueden utilizar todas las funcionalidades (11 pts)

- Descuentos
  - Código no ordenado (-10 puntos)
  - Código no compila (-100 puntos)
  - Warning (c/u -5 puntos)
  - Falta de comentarios (-30 puntos MAX)
  - Por cada día de atraso se descontarán 20 puntos (La primera hora de atraso serán solo 10 puntos).
  - Porcentaje de leak de memoria ((1-5) % -5 puntos (6- 50 %) -15 puntos (51-100 %) -25 puntos)

En caso de existir nota negativa esta será reemplazada por un 0.