Tarea 3
Web Assembly

Julio Andrade
Gerardo Crot
Mª Josefa Espinoza

Tabla de contenidos

01
Algoritmo utilizado

Algoritmo

- Encontrado en internet en código en C++
- Lo pasamos a C para simplificar el traspaso (error que veremos más adelante)
- Vemos si existen 3 subconjuntos con un método recursivo.
    - Partimos con la suma dividida 3
    - Agregamos cada valor al subconjunto si se puede hasta llegar a 0 y guardamos el subconjunto al que va

src/partition/main.c

```
// codigo obtenido de: https://www.techiedelight.com/3-partition-problem-extended-print-all-partitions/
int isSubsetExists(uint32_t* S, int n, int a, int b, int c, int* arr){
    if(a==0 && b==0 && c==0){
        return 1;
    }
    if(n<0){
        return 0;
    }
    //case 1: the current item becomes part of the first subset
    int A = 0;
    if(a - S[n] >= 0){
        arr[n]=1;
        A = isSubsetExists(S, n-1, a-S[n], b, c, arr);
    }
    //case 2: the current item becomes part of the second subset
    int B = 0;
    if(!A && (b-S[n]>=0)){
        arr[n]=2;
        B = isSubsetExists(S, n-1, a, b-S[n], c, arr);
    }
    //case 3: the current item becomes part of the third subset
    int C = 0;
    if((!A && !B) && (c-S[n] >= 0)){
        arr[n]=3;
        C = isSubsetExists(S, n-1, a, b, c-S[n], arr);
    }
    //return true if we get a solution
    return A || B || C;
};
```

Algoritmo

- Si se cumple que hay subconjunto
  - Tomamos el arreglo con las posiciones y lo iteramos
  - Agregamos los valores a cada uno de los subconjuntos
  - Retornamos 1 si se pudo separar el conjunto o 0 en otro caso

src/partition/main.c

```
int partition(uint32_t* S, uint32_t n, uint32_t* subset1, uint32_t* subset2, uint32_t*
subset3){
    int sum = accumulate(S, n);
    int arr[n];
    int result = (n>=3) && !(sum%3) && isSubsetExists(S, n-1, (int) sum/3, (int) sum/3,(int)
sum/3, arr);
    Answer* answer = malloc(sizeof(Answer));
    answer->partition = result;
    answer->subset = calloc(3, sizeof(Subset*));
    if(result){
        for(int i=0; i<3; i++){
            answer->subset[i] = malloc(sizeof(Subset));
            for(int j =0; j<n; j++){
                if(arr[j]== i+1){
                    answer->subset[i]->len++;
                }
            }
            answer->subset[i]->subset = calloc(answer->subset[i]->len, sizeof(int));

            int k=0;
            for(int j =0; j<n; j++){
                if(arr[j]== i+1){
                    answer->subset[i]->subset[k] = S[j];
                    if (i == 0) {
                        subset1[k] = S[j];
                    }
                    if (i == 1) {
                        subset2[k] = S[j];
                    }
                    if (i == 2) {
                        subset3[k] = S[j];
                    }
                    k++;
                }
            }
        }
    }
    answer->msg = "3-partition of set is not posible\n";

    if (!answer -> partition){
        clean(answer);
        return 0;
    }
    else {
        clean(answer);
        return 1;
    }
}
```

02

Demo

03
Principales implementaciones

Uso de memoria

Usamos espacios de memoria para almacenar nuestros arreglos y así que se modifiquen dentro de la función.

Especificamos los tamaños y los agregamos al módulo con setValue con su valor.

index.js

```
const makeSubsetsPtr = (myModule, input) => {
  const subset1Ptr = myModule._malloc(4 * (input.length - 2));
  const subset2Ptr = myModule._malloc(4 * (input.length - 2));
  const subset3Ptr = myModule._malloc(4 * (input.length - 2));
  for (let i = 0; i < input.length - 2; i++) {
    myModule.setValue(subset1Ptr + i * 4, -1, "i32");
    myModule.setValue(subset2Ptr + i * 4, -1, "i32");
    myModule.setValue(subset3Ptr + i * 4, -1, "i32");
  }
  return [subset1Ptr, subset2Ptr, subset3Ptr];
};
```

index.js

```
const makePtrArray = (myModule, input) => {
  const arrayPtr = myModule._malloc(4 * input.length);
  for (let i = 0; i < input.length; i++) {
    myModule.setValue(arrayPtr + i * 4, parseInt(input[i]), "i32");
  }
  return arrayPtr;
};
```

Implementación con cwrap

`cwrap` permite vincular una función de C a los tipos de valores que se les pasará como input y los que recibiremos de output, y retorna una función reutilizable en JS.

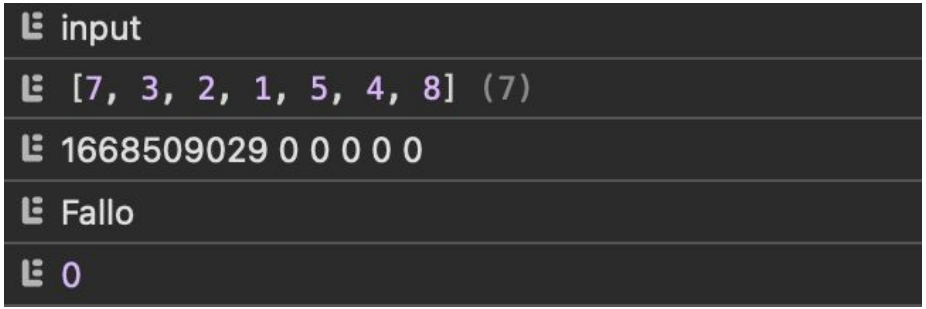
Permite enviar y recibir parámetros al módulo.

index.js

```
const [arrayPtr, subset1Ptr, subset2Ptr, subset3Ptr] = makePtrs(
      mymod,
      input
    );
    let partition = mymod.cwrap("partition", "number", [
      "number",
      "number",
      "number",
      "number",
      "number",
    ]);
    const wasmStartTime = performance.now();
    let result = partition(
      arrayPtr,
      input.length,
      subset1Ptr,
      subset2Ptr,
      subset3Ptr
    );
    const wasmEndTime = performance.now();
```

Implementación con cwrap

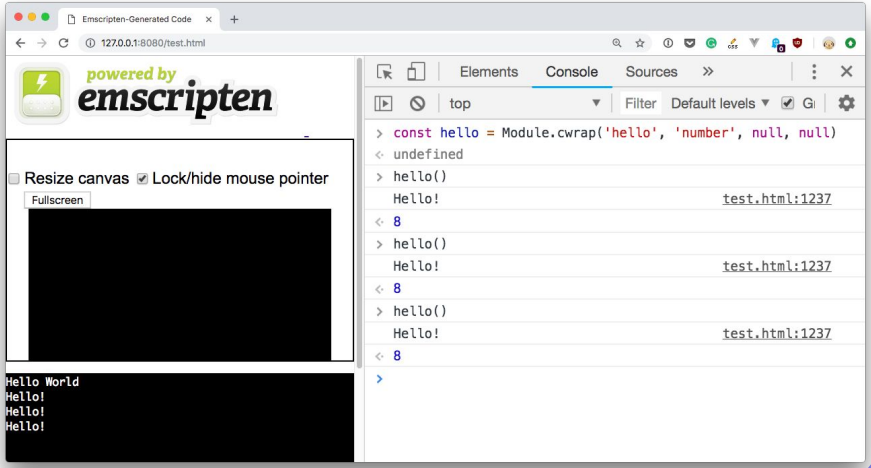
Inicialmente no podíamos usarlo sin esto, ya que no se enviaba nuestro input como queríamos.



Implementación con cwrap

Recibe 3 argumentos:

- Nombre de la función
- El tipo del retorno de la función
- Arreglo de tipos de parámetros a recibir



Resultados

Se muestran los resultados del código en Web Assembly utilizando jquery

index.js

```
if (result === 1) {
      let subsets = getSubsets(
        mymod,
        subset1Ptr,
        subset2Ptr,
        subset3Ptr,
        input
      );
      showPartition(subsets[0], 0);
      showPartition(subsets[1], 1);
      showPartition(subsets[2], 2);
    } else {
      showError();
    }
```

index.js

```
const showPartition = (subset, i) => {
  let result = $("#results");
  let subsetNumbers = "";
  for (let i = 0; i < subset.length; i++) {
    subsetNumbers += `<p class='h4 p-2'>${subset[i]}</p>`;
  }
  let partitionDiv = `<div id='partition${i}' class='d-flex flex-
wrap p-5 justify-content-center'>${subsetNumbers}</div>`;
  result.append(partitionDiv);
};
```

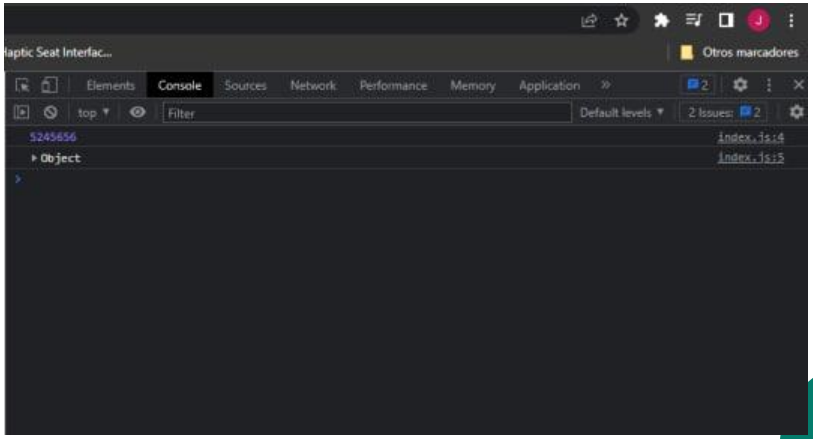
04
Problemas encontrados

Obtención de resultados

Inicialmente el código no nos entregaba los datos de un struct

Nos entregaba un espacio de memoria y el objeto del Módulo

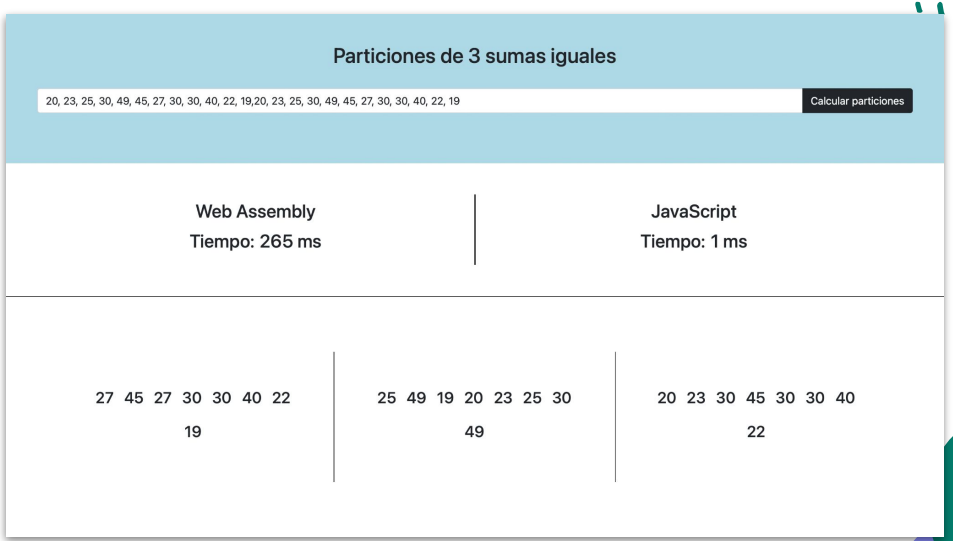
Decidimos que nos entregue un entero y que modifique arreglos iniciales en el código en C



Problemas con los tiempos

Finalmente, pese a usar el optimizador O3, el programa en Web Assembly sigue siendo más lento para casos grandes

Se lo asociamos a problemas de memoria



Dudas o comentarios

Tarea 3
Web Assembly

Julio Andrade
Gerardo Crot
Mª Josefa Espinoza