
WebAssembly

Experiencias con Rust y C++

Grupo 4

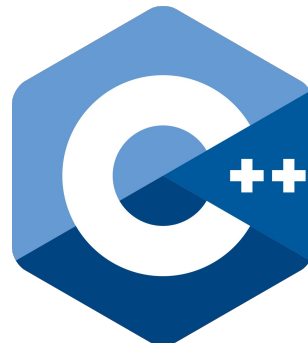
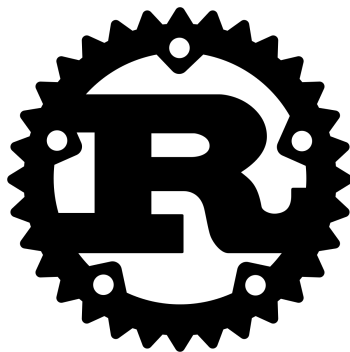
Natalia Barra Marchant

Luis Chodiman Herrera

Mauricio Ortiz Angel

Descripción del problema

Experimentar con WebAssembly y la integración de los lenguajes Rust y C/C++



Experiencia con Rust

En un grafo dado, encontrar un camino hamiltoniano en caso de existir.

- **wasm-pack** para generar el template y conectar código js con código Rust.
 - Código Rust resuelve el problema utilizando *backtracking*.
 - Matriz de adyacencia ingresa como *input* a través de JavaScript y el *output* lo entrega Rust.
-

```
pub fn search_hamiltonian(dimension: i32, matrix: String) -> Vec<i32>{  
    let mut graph = create_matrix(matrix, dimension);  
    let result = hamiltonian::run(&mut graph);  
    return result  
}
```

```
pub fn run(graph: &mut Vec<Vec<bool>>) -> Vec<i32> {  
    let out: Vec<i32> = ham_cycle(graph);  
    return out;  
}
```

```
fn ham_cycle(graph: &mut Vec<Vec<bool>>) -> Vec<i32> {  
    let mut path = vec![-1; graph.len()];  
  
    path[0] = 0;  
  
    if solve(graph, &mut path, 1) == false {  
        path[0] = -1;  
    }  
  
    return path;  
}
```




Crear “arreglo” para
definir recorrido

```
pub fn create_matrix(edges: String, vertices: i32) -> Vec<Vec<bool>> {  
    let mut graph: Vec<Vec<bool>> = vec![vec![]; vertices as usize];  
  
    for i in 0..vertices {  
        for j in 0..vertices {  
            graph[i as usize].push(false);  
        }  
    }  
  
    for c in edges.split("\n") {  
        let nodes = c.split_whitespace().collect::<Vec<&str>>();  
        if nodes.len() > 1 {  
            let a = nodes[0].parse::<usize>().unwrap();  
            let b = nodes[1].parse::<usize>().unwrap();  
            graph[a][b] = true;  
            graph[b][a] = true;  
        }  
    }  
  
    return graph;  
}
```

Inicializar matriz
vacía

Configurar
matriz de
adyacencia



```
fn solve(graph: &mut Vec<Vec<bool>>, path: &mut Vec<i32>, pos: usize) -> bool {
    // Base case: we reached all nodes
    if pos == graph.len() {
        // If there is connection between last node of path and first node
        if graph[path[pos - 1] as usize][path[0] as usize] == true {
            return true;
        } else {
            return false;
        }
    }

    // Traverse all nodes, except the start node
    for v in 1..graph.len() {
        // If there is connection with last node added and this node is not already in the path
        if is_safe(v as i32, graph, path, pos) {
            path[pos] = v as i32;

            // Explore further in this branch
            if solve(graph, path, pos + 1) {
                return true;
            }
            // Else, backtrack
            path[pos] = -1;
        }
    }

    // No solution found
    return false;
}
```

```
solveButton.addEventListener("click", _ => {  
  const nodes = parseInt(document.getElementById("nodes").value, 10);  
  const edges = document.getElementById("edges").value;  
  
  const result = module.search_hamiltonian(nodes, edges);  
  let output = document.getElementById("result");  
  
  let array = [];  
  if(result[0] !== -1) {  
    for (let i = 0; i < result.length-1; i++) {  
      array.push({source: result[i], target: result[i+1]});  
    }  
  
    let s = result.reduce((a, b) => a+' -> '+b);  
    output.innerHTML = s+' -> '+result[0];  
  } else {  
    output.innerHTML = "No tiene camino hamiltoneano";  
  }  
})
```

Recopilar input

Mostrar resultados

Bonus: Interfaz gráfica con D3.js

```
updateButton.addEventListener("click", _ => {
  d3.selectAll("svg > *").remove();
  const nodes = parseInt(document.getElementById("nodes").value, 10);
  let nodes_data = [...Array(nodes).keys()].map((x) => { return {id: x.toString()} });

  const edges = document.getElementById("edges").value;
  let edges_data = edges.split("\n")
    .map(x => x.split(" "))
    .filter(x => x.length > 1)
    .map((a) => { return {source: a[0], target: a[1]}});
```

Estructurar
información


```
var simulation = d3.forceSimulation().nodes(nodes_data);  
simulation  
  .force("charge_force", d3.forceManyBody())  
  .force("center_force", d3.forceCenter(width / 2, height / 2));
```

Inicializar
ambiente de
simulación

```
var node = svg.append("g")  
  .attr("class", "nodes")  
  .selectAll("g")  
  .data(nodes_data)  
  .enter().append("g")
```

Agregar nodos a simulación

```
var link_force = d3.forceLink(edges_data).id(function(d) { return d.id; }).distance(100)  
simulation.force("links", link_force)
```

```
var link = svg.append("g")  
  .attr("class", "links")  
  .selectAll("line")  
  .data(edges_data)  
  .enter()  
  .append("line")  
  .attr("stroke-width", 2);
```

Definir fuerza en
enlaces

Agregar enlaces
a simulación

```
simulation.on("tick", tickActions );

function tickActions() {
  node
    .attr("transform", function(d) {
      return "translate(" +
        Math.max(7, Math.min(width - 7, d.x))
        + ", "
        + Math.max(7, Math.min(height - 7, d.y)) + ")";
    })

  link
    .attr("x1", function(d) { return d.source.x; })
    .attr("y1", function(d) { return d.source.y; })
    .attr("x2", function(d) { return d.target.x; })
    .attr("y2", function(d) { return d.target.y; });
}
```

Comportamiento de la
simulación

```
var drag_handler = d3.drag()  
  .on("start", drag_start)  
  .on("drag", drag_drag)  
  .on("end", drag_end);
```

} Arrastrar nodos

```
function drag_start(d) {  
  if (!d3.event.active) simulation.alphaTarget(0.3).restart();  
  d.fx = d.x;  
  d.fy = d.y;  
}
```

```
function drag_drag(d) {  
  d.fx = d3.event.x;  
  d.fy = d3.event.y;  
}
```

```
function drag_end(d) {  
  if (!d3.event.active) simulation.alphaTarget(0);  
  d.fx = null;  
  d.fy = null;  
}
```

} Comportamiento

Experiencia con C/C++

Aplicación web para codificar texto libre. En algunos casos, permite decodificar

- Librería Emscripten para exportar funciones
 - Código C++ para Vigenere y Cifrado de Cesar
 - Librería en C para Código Morse
 - Archivo cipher.cpp exporta llamadas a funciones (intermediario)
 - Archivo cipher.js realiza llamada a funciones
-

Archivo principal - cipher.cpp

```
#ifdef __cplusplus
extern "C" {
#endif

char* EMSCRIPTEN_KEEPALIVE caesar(char* text, int n, int s) {
    for (int i = 0; i < n; i++) {
        if (isupper(text[i])) text[i] = char(int(text[i] + s - 65) % 26 + 65);
        else if (isspace(text[i])) continue;
        else text[i] = char(int(text[i] + s - 97) % 26 + 97);
    }
    return text;
}

char* EMSCRIPTEN_KEEPALIVE morse_encode(char* text) {
    return encoder(text);
}

char* EMSCRIPTEN_KEEPALIVE vigenere_encoder(char* text, char* key) {
    return vigenere_encode(text, key);
}

char* EMSCRIPTEN_KEEPALIVE vigenere_decoder(char* text, char* key) {
    return vigenere_decode(text, key);
}

#ifdef __cplusplus
}
#endif
```

Para poder compilar

Función escrita directamente
en el archivo cipher .cpp

Función importada desde
librería en C 'morse-code'

Funciones importadas desde
archivo C++ vigenere.cpp

Archivo js - cipher.js

```
if (cipher == 'morse') {  
    types = ['string'] ;  
    args = [input];  
}
```



```
let result = Module.ccall(  
    name, // name of C function  
    'string', // return type  
    types, // argument types  
    args // arguments  
);
```
