WebAssembly

04/05/2020

Grupo 2
Paul Heinsohn
Sebastián Carreño

Profesor Jaime Navón

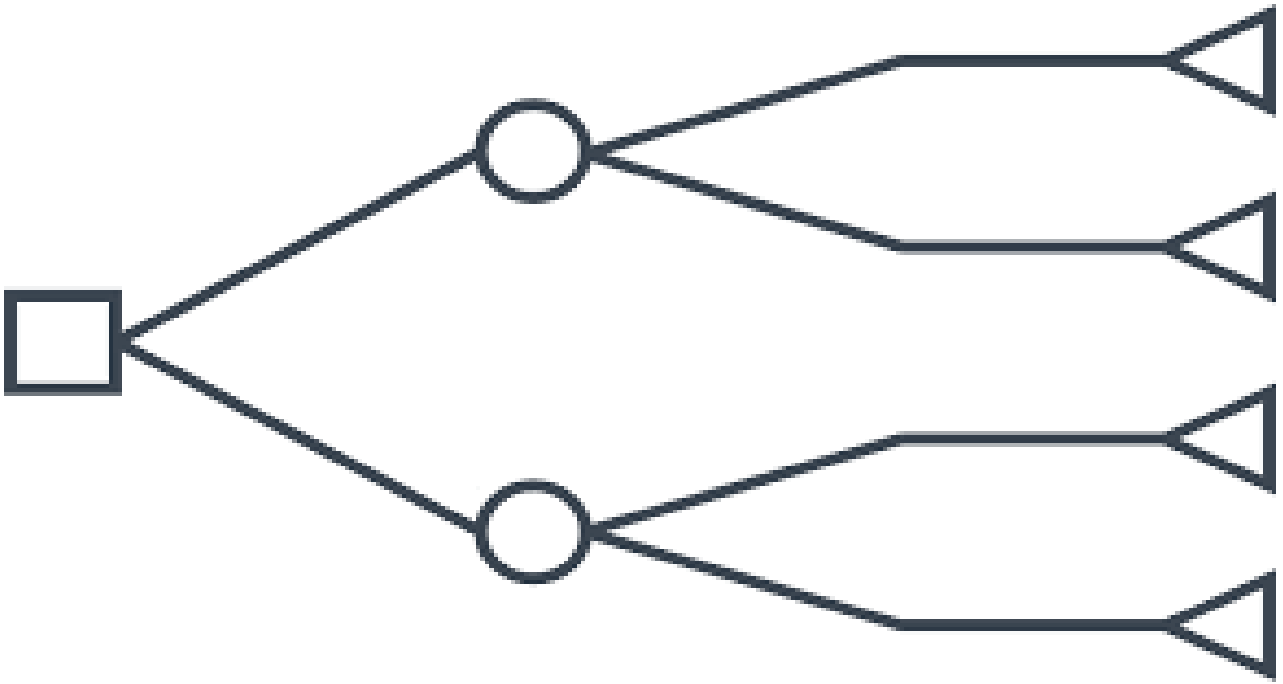
Problema Sugerido

- Encontrar un set de n números que no generen repetidos
- Usar n con los valores 5, 11, 23 y 47

Approach

- Uso de un árbol de decisión con profundidad n + 1

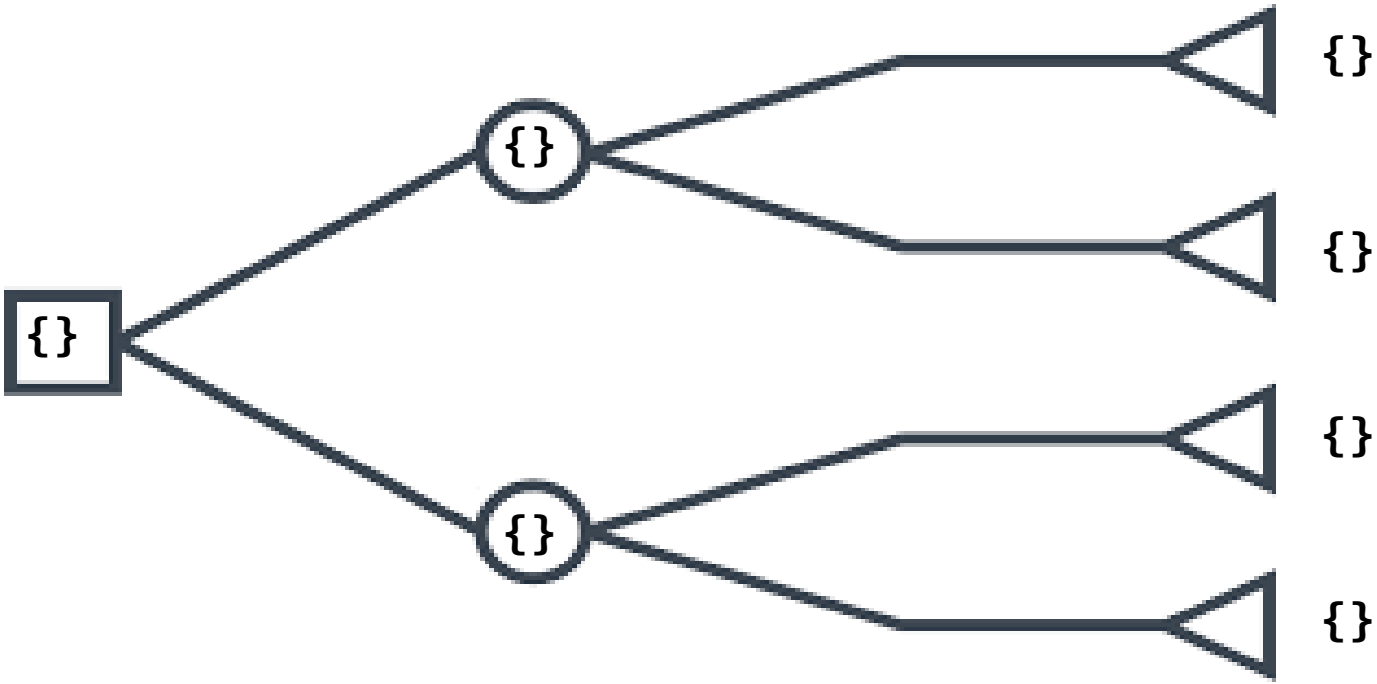
n = 2:



Approach

- Cada Nodo como un set de números, con nodo raíz vacío

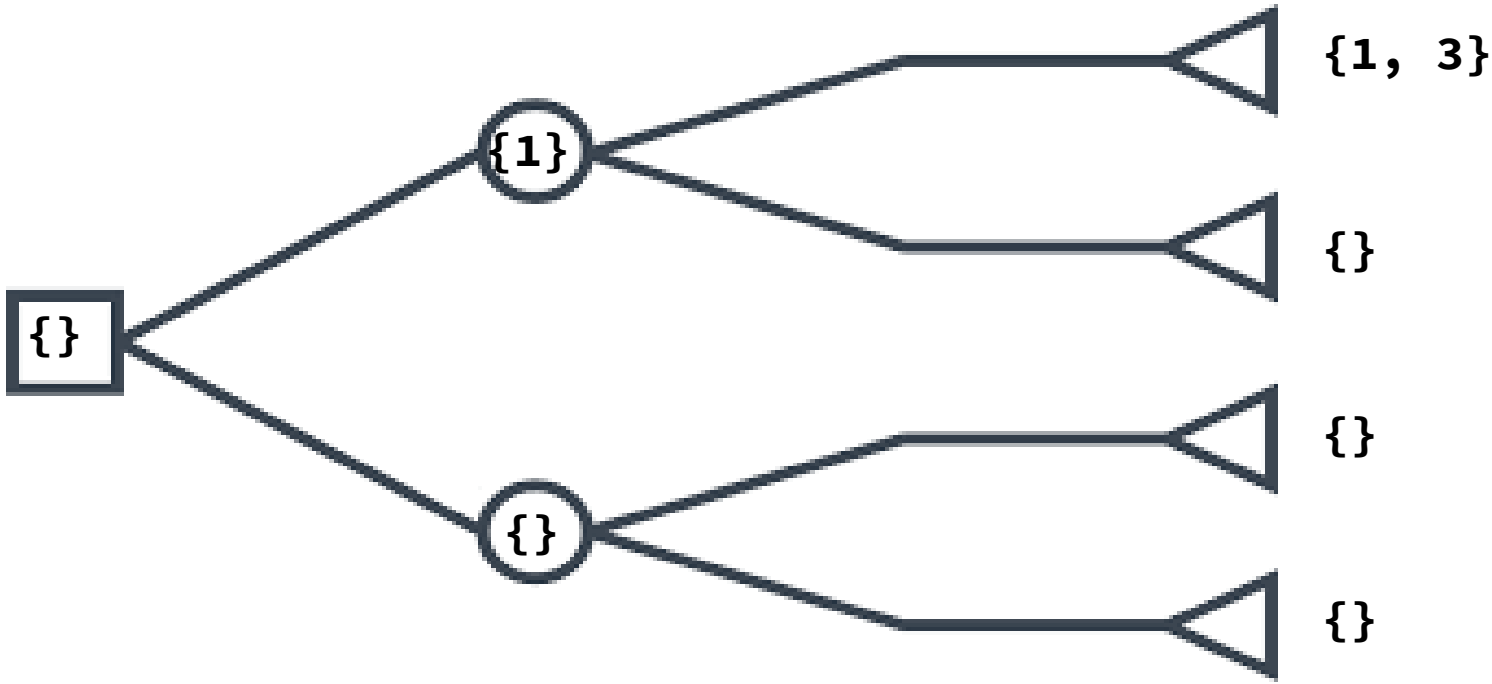
n = 2:



Approach

- Generar nodos hijos con DFS hasta llegar a n + 1

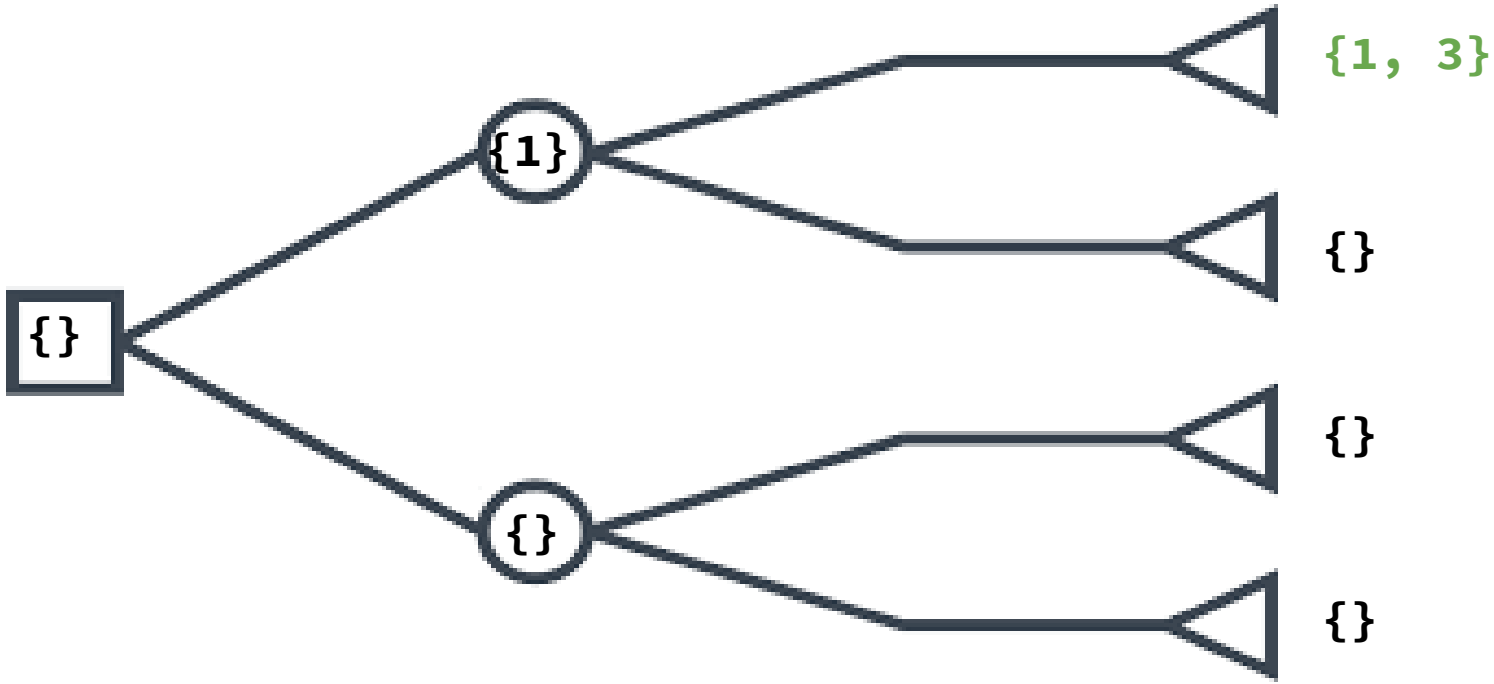
n = 2:



Approach

- Guardar nodo hoja como solución incumbente y continuar DFS

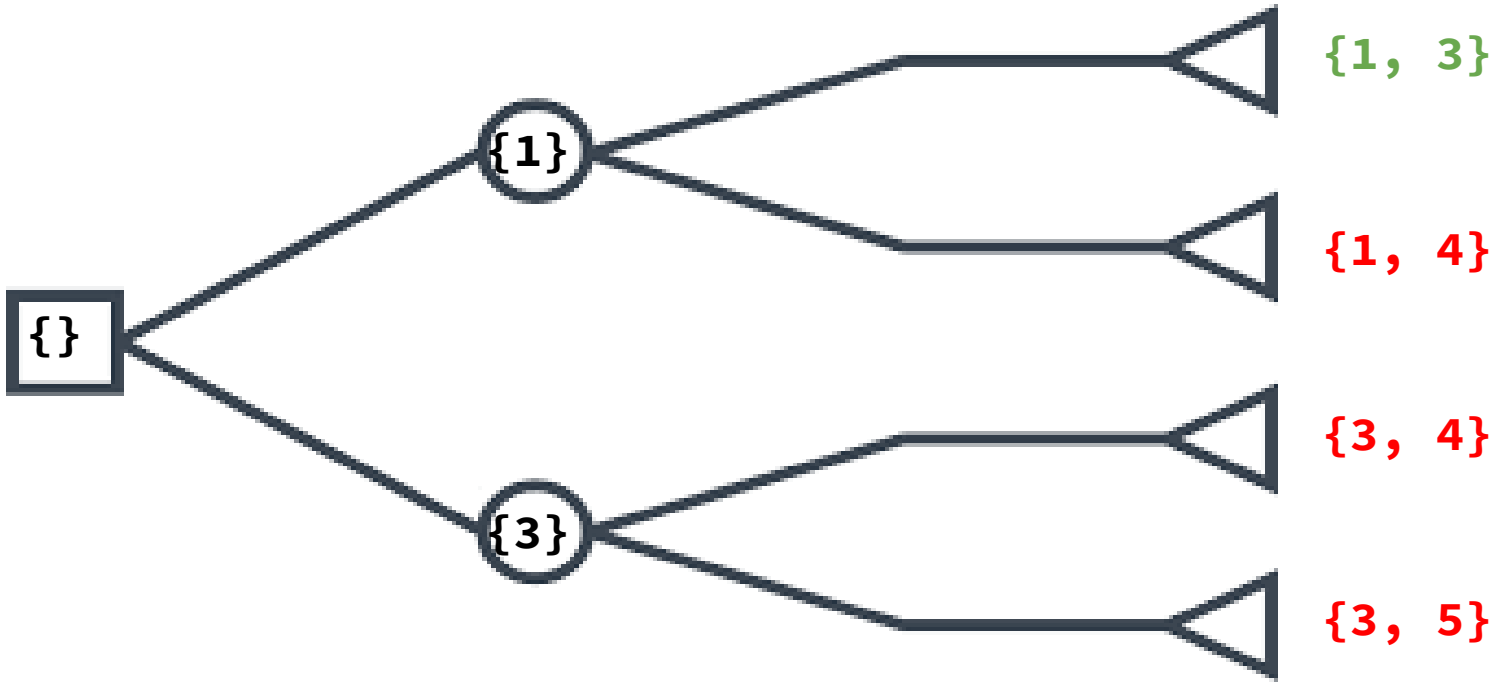
n = 2:



Approach

- Podar se supera el valor del máximo del incumbente

n = 2:



Approach

- Luego se generaría el hijo {4}, pero como 4 > max({1, 3}) entonces se poda y se da por finalizado el algoritmo.
- Incumbente actual se convierte en el óptimo

{1, 3}

Planificación

- Pasar approach a NodeJs
- Pasar de NodeJs a C
- Pasar de C a WebAssembly mediante Emscripten
- Comparar resultados

NodeJs

- Para cada n = [5, 11, 23, 47], obtenemos el incumbente del primer hijo del set vacío → {1}
- Expandir hijos de {} hasta que el hijo expandido sea mayor al incumbente

```
const optimizedSolution = (n_list) => {
  n_list.forEach(n => {
        result.incumbent = new Set([n * n]);
        optimizedRecursiveCall(n, new Set([1]));

        let i = 3;
        while (i < Math.max(...result.incumbent)) {optimizedRecursiveCall(n, new Set([i++]))};
  });
}
```

NodeJs

- Función recursiva

```
const optimizedRecursiveCall = (n, solution) => {
    if (solution.size === n) {
      if (Math.max(...solution) < Math.max(...result.incumbent)) {
        result.incumbent = solution;
      }
      return;
    }

    let num = nextValidNumber(solution, Math.max(...solution) + 1);
    while (num < Math.max(...result.incumbent)) {
        optimizedRecursiveCall(n, new Set([...solution, num]));
      num = nextValidNumber(solution, num + 1);
    }
    return;
}
```

¡Problema!

- Algoritmo en NodeJs muy lento para n muy grande
- Hipótesis: Algoritmo en C es mucho más rápido

Implementación C

- Réplica del código NodeJs, pero con uso de listas ligadas en lugar de Sets
- Misma complejidad sin uso de eliminación
  - Inserción O(1)
  - Búsqueda del máximo O(n), n tamaño de la lista ligada/Set

Emscripten

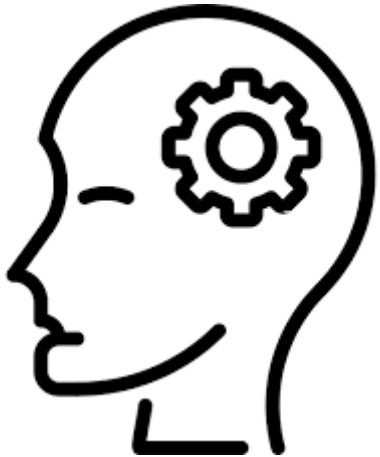
- Conversión del archivo .c a .wasm, .html y .js
- Archivo JS lee al archivo WASM
- Luego, se despliega en el HTML

¡Problema!

- ~~Hipótesis: Algoritmo en C es mucho más rápido~~
- Algoritmo en C igual o más lento que en NodeJs

¿Solución?

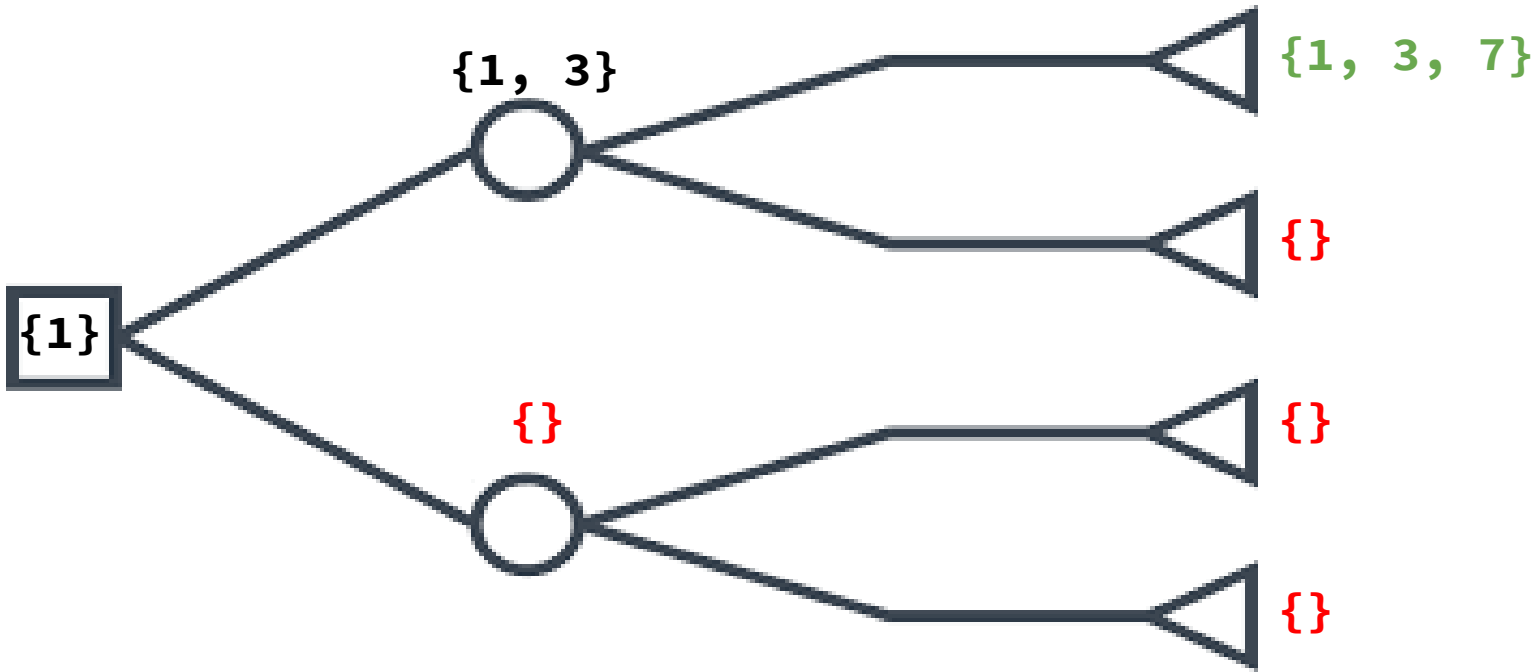
- Algoritmo muy lento
- Poco tiempo



Solución "Naive"

- Nodo padre es {1}, y algoritmo se detiene cuando llega a profundidad n

n = 3:



DEMo Soluciones

Conclusiones

- C no siempre es más rápido que NodeJs

Conclusiones

- C no siempre es más rápido que NodeJs
- WebAssembly no fue útil para nuestra solución

Conclusiones

- C no siempre es más rápido que NodeJs
- WebAssembly no fue útil para nuestra solución
- Interés en otras soluciones al problema propuesto