# Problema Canibal y Misioneros

Tags

```
#include <iostream>
#include <vector>
using namespace std;

// Definición del estado del problema
struct Estado {
    int canibalesIzq;
    int misionerosIzq;
    int canibalesDer;
    int misionerosDer;
    bool boteIzq;  // true si el bote está en la izquierda, f
alse si está en la derecha
};

// Función para imprimir el estado actual
void imprimirEstado(const Estado &estado) {
    cout << "Izquierda - C: " << estado.canibalesIzq << " M:
" << estado.misionerosIzq
         << " | Derecha - C: " << estado.canibalesDer << " M:
" << estado.misionerosDer
         << " | Bote en " << (estado.boteIzq ? "izquierda" :
"derecha") << endl;
}

// Función para verificar si un estado es seguro (caníbales n
o superan a misioneros en ninguna orilla)
bool esSeguro(const Estado &estado) {
    // No puede haber más caníbales que misioneros en ninguna
orilla, a menos que no haya misioneros
    if (estado.misionerosIzq > 0 && estado.canibalesIzq > est
```

```
ado.misionerosIzq)
        return false;
    if (estado.misionerosDer > 0 && estado.canibalesDer > est
ado.misionerosDer)
        return false;
    return true;
}

// Función para verificar si hemos alcanzado el estado final
(todos en la derecha)
bool esEstadoFinal(const Estado &estado) {
    return estado.canibalesIzq == 0 && estado.misionerosIzq =
= 0 &&
           estado.canibalesDer == 3 && estado.misionerosDer =
= 3;
}

// Función de backtracking para resolver el problema de caníb
ales y misioneros
bool resolverProblema(vector<Estado> &solucion, Estado estado
Actual) {
    // Agregamos el estado actual a la solución
    solucion.push_back(estadoActual);

    // Si es el estado final, hemos resuelto el problema
    if (esEstadoFinal(estadoActual)) {
        return true;
    }

    // Generar posibles movimientos
    vector<Estado> siguientesEstados;

    if (estadoActual.boteIzq) {
        // El bote está en la izquierda, debemos mover gente
a la derecha
        // Intentar mover un caníbal
```

```
        if (estadoActual.canibalesIzq > 0) {
            Estado nuevoEstado = estadoActual;
            nuevoEstado.canibalesIzq--;
            nuevoEstado.canibalesDer++;
            nuevoEstado.boteIzq = false;
            if (esSeguro(nuevoEstado)) siguientesEstados.push
_back(nuevoEstado);
        }
        // Intentar mover un misionero
        if (estadoActual.misionerosIzq > 0) {
            Estado nuevoEstado = estadoActual;
            nuevoEstado.misionerosIzq--;
            nuevoEstado.misionerosDer++;
            nuevoEstado.boteIzq = false;
            if (esSeguro(nuevoEstado)) siguientesEstados.push
_back(nuevoEstado);
        }
        // Intentar mover dos caníbales
        if (estadoActual.canibalesIzq > 1) {
            Estado nuevoEstado = estadoActual;
            nuevoEstado.canibalesIzq -= 2;
            nuevoEstado.canibalesDer += 2;
            nuevoEstado.boteIzq = false;
            if (esSeguro(nuevoEstado)) siguientesEstados.push
_back(nuevoEstado);
        }
        // Intentar mover dos misioneros
        if (estadoActual.misionerosIzq > 1) {
            Estado nuevoEstado = estadoActual;
            nuevoEstado.misionerosIzq -= 2;
            nuevoEstado.misionerosDer += 2;
            nuevoEstado.boteIzq = false;
            if (esSeguro(nuevoEstado)) siguientesEstados.push
_back(nuevoEstado);
        }
        // Intentar mover un caníbal y un misionero
```

```
        if (estadoActual.canibalesIzq > 0 && estadoActual.mis
ionerosIzq > 0) {
            Estado nuevoEstado = estadoActual;
            nuevoEstado.canibalesIzq--;
            nuevoEstado.misionerosIzq--;
            nuevoEstado.canibalesDer++;
            nuevoEstado.misionerosDer++;
            nuevoEstado.boteIzq = false;
            if (esSeguro(nuevoEstado)) siguientesEstados.push
_back(nuevoEstado);
        }
    } else {
        // El bote está en la derecha, debemos mover gente a
la izquierda
        // Intentar mover un caníbal
        if (estadoActual.canibalesDer > 0) {
            Estado nuevoEstado = estadoActual;
            nuevoEstado.canibalesDer--;
            nuevoEstado.canibalesIzq++;
            nuevoEstado.boteIzq = true;
            if (esSeguro(nuevoEstado)) siguientesEstados.push
_back(nuevoEstado);
        }
        // Intentar mover un misionero
        if (estadoActual.misionerosDer > 0) {
            Estado nuevoEstado = estadoActual;
            nuevoEstado.misionerosDer--;
            nuevoEstado.misionerosIzq++;
            nuevoEstado.boteIzq = true;
            if (esSeguro(nuevoEstado)) siguientesEstados.push
_back(nuevoEstado);
        }
        // Intentar mover dos caníbales
        if (estadoActual.canibalesDer > 1) {
            Estado nuevoEstado = estadoActual;
            nuevoEstado.canibalesDer -= 2;
```

```
            nuevoEstado.canibalesIzq += 2;
            nuevoEstado.boteIzq = true;
            if (esSeguro(nuevoEstado)) siguientesEstados.push
_back(nuevoEstado);
        }
        // Intentar mover dos misioneros
        if (estadoActual.misionerosDer > 1) {
            Estado nuevoEstado = estadoActual;
            nuevoEstado.misionerosDer -= 2;
            nuevoEstado.misionerosIzq += 2;
            nuevoEstado.boteIzq = true;
            if (esSeguro(nuevoEstado)) siguientesEstados.push
_back(nuevoEstado);
        }
        // Intentar mover un caníbal y un misionero
        if (estadoActual.canibalesDer > 0 && estadoActual.mis
ionerosDer > 0) {
            Estado nuevoEstado = estadoActual;
            nuevoEstado.canibalesDer--;
            nuevoEstado.misionerosDer--;
            nuevoEstado.canibalesIzq++;
            nuevoEstado.misionerosIzq++;
            nuevoEstado.boteIzq = true;
            if (esSeguro(nuevoEstado)) siguientesEstados.push
_back(nuevoEstado);
        }
    }

    // Probar cada uno de los siguientes estados recursivamen
te
    for (auto &nuevoEstado : siguientesEstados) {
        if (resolverProblema(solucion, nuevoEstado)) {
            return true;  // Encontramos una solución
        }
    }
```

```
    // Si no hay solución en este camino, retrocedemos
    solucion.pop_back();
    return false;
}

int main() {
    // Estado inicial: 3 caníbales, 3 misioneros en la izquie
rda, y el bote en la izquierda
    Estado estadoInicial = {3, 3, 0, 0, true};
    vector<Estado> solucion;

    if (resolverProblema(solucion, estadoInicial)) {
        for (const auto &estado : solucion) {
            imprimirEstado(estado);
        }
    } else {
        cout << "No hay solución." << endl;
    }

    return 0;
}
```

## Explicación de los componentes principales:

1. **Estructura** `Estado`: Representa la cantidad de caníbales y misioneros en ambas orillas y la posición del bote.
2. `imprimirEstado()`: Muestra el estado actual para ayudar a visualizar el progreso.
3. `esSeguro()`: Verifica que no haya más caníbales que misioneros en ninguna de las orillas, evitando que los misioneros sean devorados.
4. `esEstadoFinal()`: Verifica si todos los caníbales y misioneros han llegado a la orilla derecha.
5. `resolverProblema()`: Implementa el algoritmo de backtracking para probar todos los posibles movimientos entre orillas, mientras verifica que cada estado sea seguro.

## Cómo funciona:

1. El código inicializa el estado con 3 caníbales y 3 misioneros en la orilla izquierda, y el bote también en la orilla izquierda.
2. Utiliza backtracking para probar todos los movimientos posibles, asegurándose de que el estado sea seguro.
3. Imprime los pasos que llevan a la solución si se encuentra una, o indica que no hay solución.

Este código resuelve el problema de los caníbales y misioneros y muestra los pasos para llevar a todos los misioneros y caníbales al otro lado del río de forma segura.