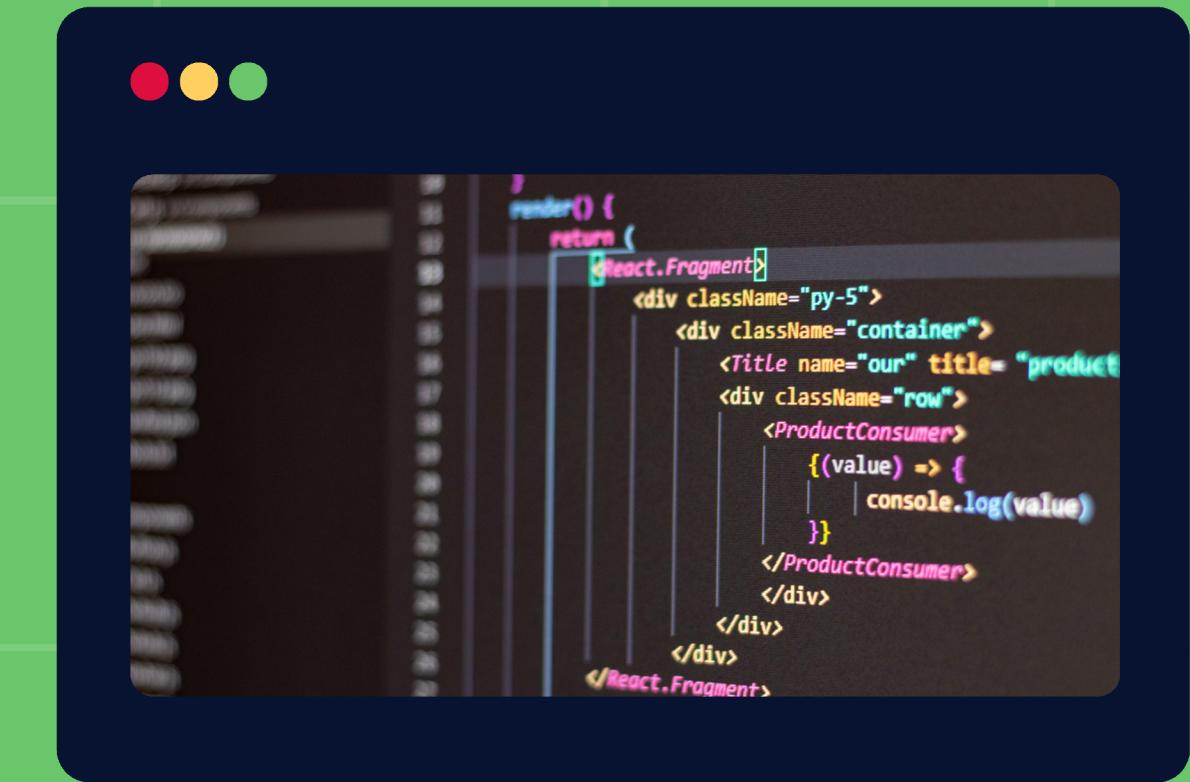


Recursion y Backtracking

Luis Rojas



```
render() {
  return (
    <React.Fragment>
      <div className="py-5">
        <div className="container">
          <Title name="our" title="product">
            <div className="row">
              <ProductConsumer>
                {(value) => (
                  <div>| console.log(value)</div>
                )}
              </ProductConsumer>
            </div>
          </div>
        </React.Fragment>
      )
    )
  }
}
```

Recursión



Se llama a sí misma para resolver un problema.

Se basa en dividir el problema en subproblemas más pequeños.



Estructura básica



Caso base: condición que detiene la recursión.



Llamada recursiva: la función se llama a sí misma con un subproblema más simple.



Recursive Functions

```
int recursion ( x )  
{  
    if ( x==0 )  
        return;  
    recursion ( x-1 );  
}
```

Base case
Function being called again by itself

Ejemplo Factorial

$$n! = n \times (n-1)!$$

```
int factorial(int n) {  
    if (n == 0) return 1;      // caso base  
    return n * factorial(n-1); // llamada recursiva  
}
```

Backtracking

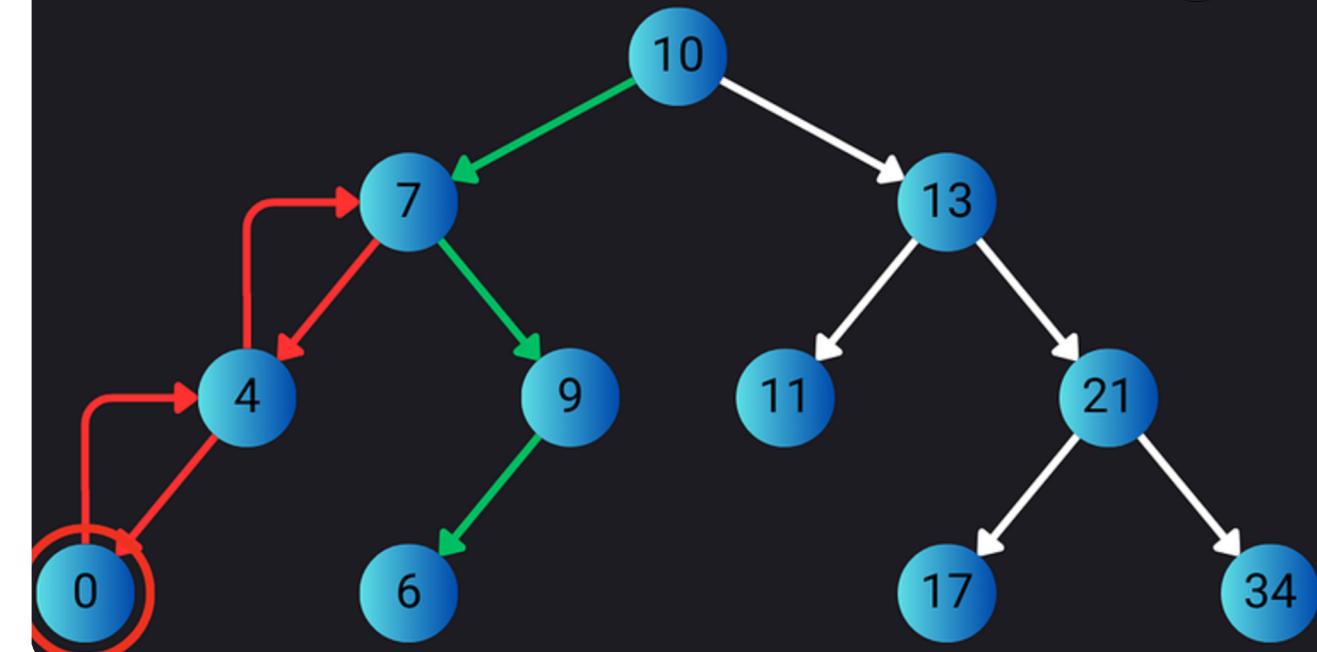


Es una técnica de diseño de algoritmos para resolver problemas de búsqueda y optimización.

Consiste en explorar todas las posibles soluciones,



Tree Backtracking



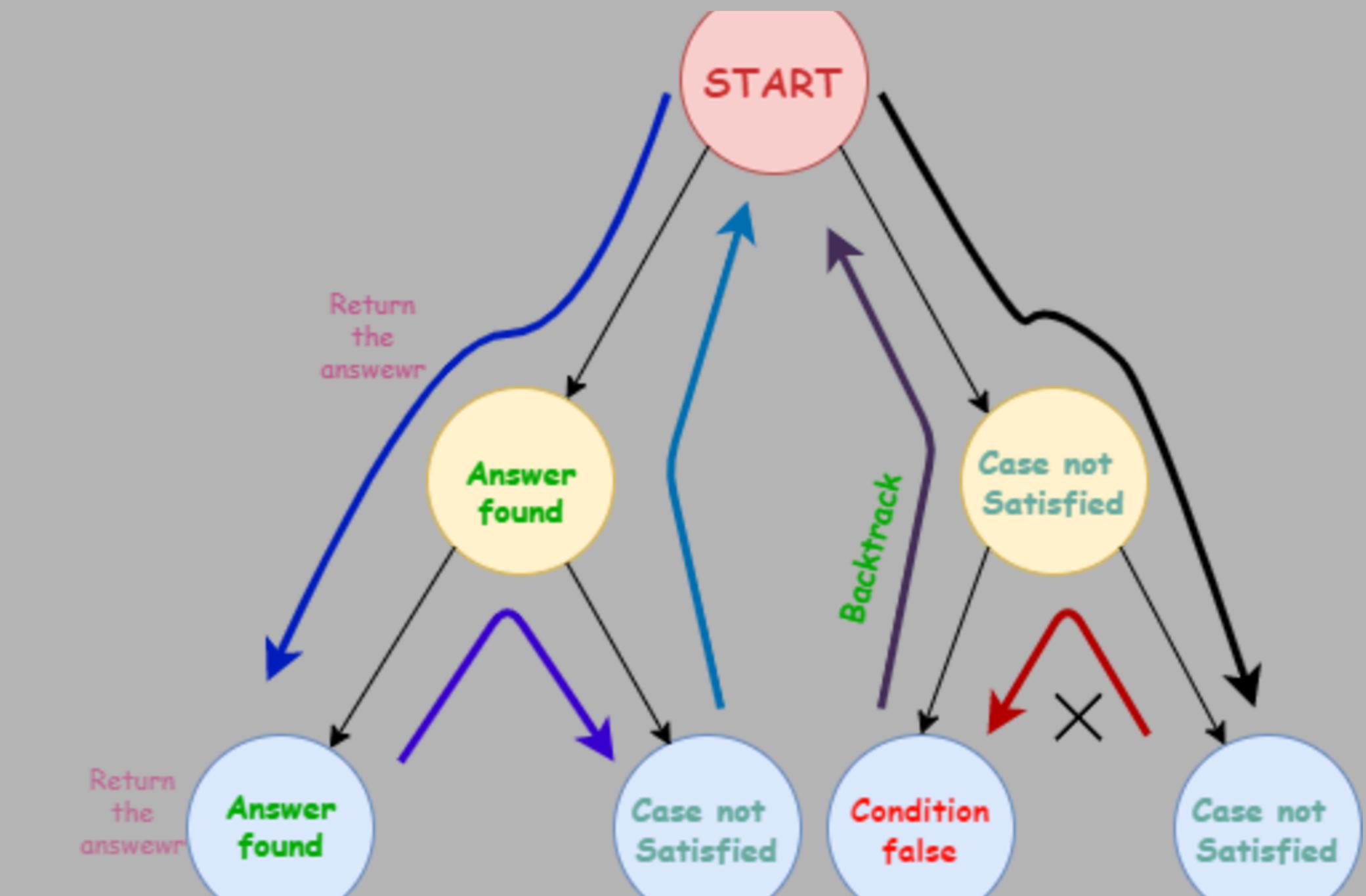
Idea principal



Construir una solución de forma incremental.



1. Verificar en cada paso si es válida.
2. Si no es válida → retroceder y probar otra opción.
3. Si es válida y completa → guardar la solución.



Ejemplo N reinas N=4



Colocar 4 reinas en un tablero de ajedrez 4×4 de forma que ninguna se ataque entre sí.



```
int N = 4;
int tablero[10]; // tablero[i] = columna donde está la
reina en la fila i

bool esValido(int fila, int col) {
    for (int i = 0; i < fila; i++) {
        // misma columna
        if (tablero[i] == col) return false;
        // misma diagonal
        if (abs(tablero[i] - col) == abs(i - fila)) return
false;
    }
    return true;
}

void resolver(int fila) {
    if (fila == N) { // caso base: todas las reinas colocadas
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++)
                cout << (tablero[i] == j ? "Q " : ". ");
            cout << "\n";
        }
        cout << "\n";
        return;
    }
    for (int col = 0; col < N; col++) {
        if (esValido(fila, col)) {
            tablero[fila] = col; // colocar reina
            resolver(fila + 1); // llamada recursiva
            // backtracking: no hace falta deshacer, se sobreescribe
        }
    }
}
```