Práctica 4: Algoritmos Backtracking y B&B

Álvaro Fernández García
Alexandra-Diana Dumitru
Ana María Romero Delgado
Germán Castro Montes
Marino Fajardo Torres

Índice

Descripción del problema

El ejercicio propuesto el siguiente:

> Nuestro objetivo es encontrar una solución en el juego solitario del **Continental**. Se colocan 32 piezas iguales en un tablero de 33 casillas donde las 'x' corresponden a las posiciones no válidas (Figura 1.). Solo se permiten movimientos de las piezas en horizontal y en vertical. Una pieza solo puede moverse saltando sobre otra y situándose en la siguiente casilla, que debe estar vacía. La pieza sobre la que se salta se retira del tablero.

Se consigue terminar con exito el solitario cuando queda una sola pieza 'o' en el tablero y se encuentra en la posición central de este (Figura 2.), que inicialmente estaba vacía.

X	X	O	O	O	X	X
X	X	O	O	O	X	X
O	O	O	O	O	O	O
O	O	O		O	O	O
O	O	O	O	O	O	O
X	X	O	O	O	X	X
X	X	O	O	O	X	X

X	X				X	X
X	X				X	X
			O			
X	X				X	X
X	X				X	X

Figura 1 y **Figura 2** respectivamente.

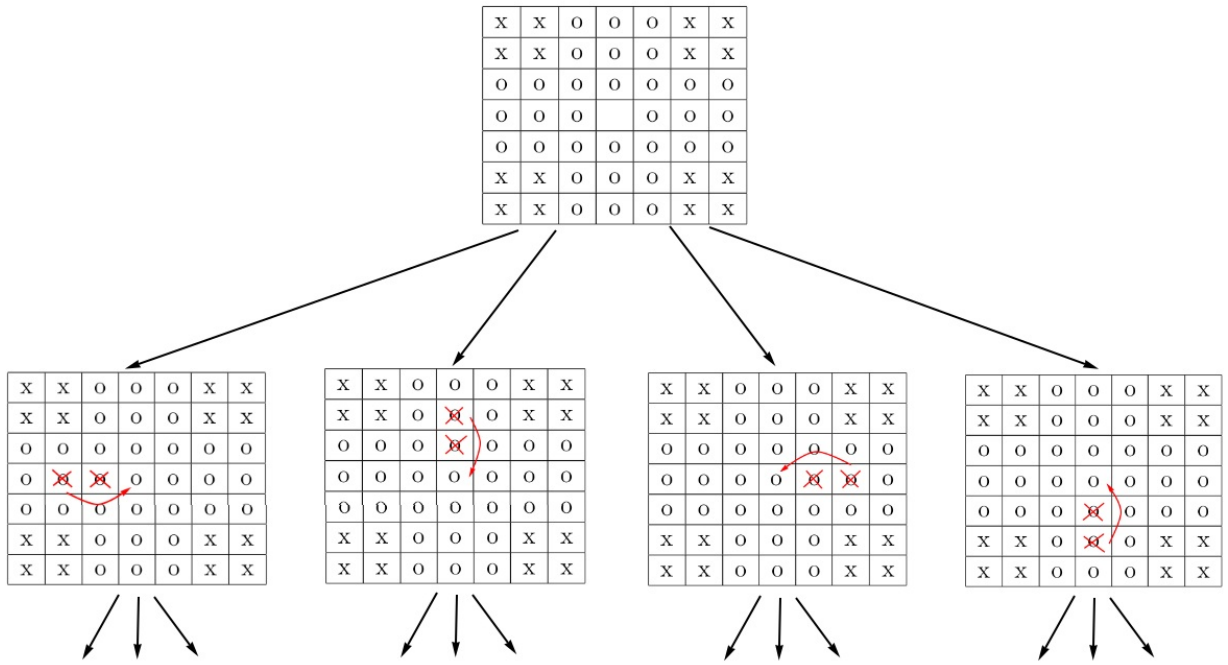
Algoritmo Backtracking

Se trata de un algoritmo de fuerza bruta. Como tal, pretende generar todas las posibles combinaciones del juego hasta dar con una que satisfaga el criterio final. Cada uno de los niveles del árbol contendrá todas las configuraciones de fichas resultantes de aplicar todos los movimientos factibles a partir de la configuración anterior.

Cada vez que el algoritmo realiza un movimiento comprueba si se ha alcanzado una solución, si es así el proceso finaliza, si no, se prepara el siguiente movimiento.

Para buscar el siguiente movimiento se seleccionan todas las fichas y todos los posibles saltos de las mismas (arriba, abajo, izquierda…). Si se llega a una configuración en la que no hay más alternativas y no se corresponde con una solución, se restaura a la combinación anterior y se procesa el siguiente movimiento factible.

Aquí se presenta como sería el árbol:



Para el algoritmo, se ha desarroyado una clase que cuenta con los siguientes atributos:

> **Tablero[7][7]**: Una matriz cuadrada de 7x7 de char.
> **Camino [31]**: Array que contiene los movimientos de la solución del solitario. Para ello se ha desarrollado una estructura llamada Movimiento que contiene la posición inicial, la posición final y la ficha comida.
> **MOVIMIENTOS**: constante que indica el número de movimientos que se deben realizar para llegar a la solución (*31 en nuestro caso*).
> **encontrado**: booleano que indica si se ha encontrado la solución.

A continuación tenemos los métodos necesarios para la realización del algoritmo backtracking.

> **Continental()**: Constructor, inicializa el tablero y pone encontrado a falso.
> **JuegoContinental()**: es la funcion que busca una solución al problema del Continental llamando al JuegoContinentalRec(k) con el nivel inicial a 0.
> **ImprimeSolucion()**: Imprime la secuencia de pasos para resolver el juego.
> **ImprimeSolucionPasos()**:Imprime la solución paso por paso mostrando el tablero.
> **MovFactible(int fil, int col, int mov)**: dada una ficha indicada por su fila y columna y un movimiento, comprueba si dicho movimiento es posible, es decir, que la posición a la que quiero mover la ficha esta vacía, que haya una ficha para comer en medio y que esté dentro de los límites del tablero
> **ActualizaTablero(int k)**: Aplica el movimiento k-ésimo al tablero.
> **RestauraTablero(int pos)**: Deja el tablero en un estado anterior:
> **ConstruyeMovimiento(int i, int j, int mov, int k)**: Crea el struct movimiento correspondiente a

> ese nivel.
> **ProcesaSolucion()**: Comprueba si una solución es válida o no, (*solo queda una ficha en el tablero y esta se encuentra en el centro*).
> **JuegoContinental_rec(int k)**: Explicada más adelante.
> **ImprimeTablero()**: imprime el tablero.

A continuación se desarrolla, en pseudocódigo, el algoritmo principal del backtracking: **JuegoContinental_rec(int k)**.

```
Require: Nivel (k)
 if ProcesaSolucion() then
    encontrado ← true
 else
    for i = 0 to 6 and not econtrado do
        for j = 0 to 6 and not econtrado do
            if en Tablero[i][j] hay una ficha then
                for m = 1 to 4 and not econtrado do
                    if MovFactible(i,j,m) then
                        ConstruyeMovimiento(i,j,m,k)
                        ActualizaTablero(k)
                        JuegoContinentalRecursivo(k+1)
                    end if
                end for
            end if
        end for
    end for
 end if
 if not encontrado then
    RestauraTablero(k-1)
 end if
```

Aquí se presenta el código del algoritmo:

ARCHIVO DE CABECERAS

```
#ifndef _CONTINENTAL_H
#define _CONTINENTAL_H

  #include<iostream>
  #include<vector>

  using namespace std;

  class Continental{
  public:
    //Constructor: Inicializa el tablero.
    Continental();

    //Busca una solución al problema del Continental.
    void JuegoContinental();

    //Imprime la secuencia de pasos para resolver el juego:
    //Precondición: Se ha encontrado una solución.
    void ImprimeSolucion();

    //Imprime la solución paso por paso mostrando el tablero:
```

```
    //Precondición: Se ha encontrado una solución.
    void ImprimeSolucionPasos();

  private:
    //Struct que representa un movimiento:
    struct Movimiento{
      pair<int,int> posOriginal;
      pair<int,int> posFinal;
      pair<int,int> fichaComida;
    };

    //Atributos privados del algoritmo:
    char Tablero[7][7];
    vector<Movimiento> Camino;
    static const int MOVIMIENTOS = 31;
    bool encontrado;

    //Funciones privadas:
    //Comprueba si un movimiento se puede realizar o no (es factible)
    bool MovFactible(int fil, int col, int mov);

    //Aplica un movimiento al tablero.
    void ActualizaTablero(int k);

    //Deja el tablero en un estado anterior:
    void RestauraTablero(int pos);

    //Crea el struct movimiento correspondiente a ese nivel:
    // 1 = izquierda, 2 = arriba, 3 = derecha, 4 = abajo.
    void ConstruyeMovimiento(int i, int j, int mov, int k);

    //Comprueba si una solución es válida o no:
    bool ProcesaSolucion();

    //Función recursiva para jugar al continental:
    //Método de backtracking:
    void JuegoContinental_rec(int k);

    //Imprime el Tablero:
    void ImprimeTablero();
  };

#endif
```

IMPLEMENTACIÓN

```
Continental::Continental(){
  Camino.resize(MOVIMIENTOS);
  for(unsigned i = 0; i<7; ++i)
    for(unsigned j = 0; j<7; ++j){
      if((i == 0 || i == 1 || i == 5 || i == 6) && (j == 0 || j == 1 || j == 5 || j == 6))
        Tablero[i][j] = 'X';
      else
        Tablero[i][j] = 'O';
    }
  Tablero[3][3] = '-';
  encontrado = false;
}
```

```
bool Continental::MovFactible(int fil, int col, int mov){
  switch (mov) {
    case 1: //izquierda
      if(col-2 > -1) //No estas fuera de la matriz
        if(Tablero[fil][col-2] != 'X') //pos valida.
          if(Tablero[fil][col-2] == '-' && Tablero[fil][col-1] == 'O')
              return true;
      break;
    case 2: //arriba
      if(fil-2 > -1) //No estas fuera de la matriz
        if(Tablero[fil-2][col] != 'X') //pos valida.
          if(Tablero[fil-2][col] == '-' && Tablero[fil-1][col] == 'O')
              return true;
      break;
    case 3: //derecha
      if(col+2 < 7) //No estas fuera de la matriz
        if(Tablero[fil][col+2] != 'X') //pos valida.
          if(Tablero[fil][col+2] == '-' && Tablero[fil][col+1] == 'O')
              return true;
      break;
    case 4: //abajo
      if(fil+2 < 7) //No estas fuera de la matriz
        if(Tablero[fil+2][col] != 'X') //pos valida.
          if(Tablero[fil+2][col] == '-' && Tablero[fil+1][col] == 'O')
              return true;
      break;
  }
  return false;
}


void Continental::ActualizaTablero(int k){
  Tablero[Camino[k].posOriginal.first][Camino[k].posOriginal.second] = '-';
  Tablero[Camino[k].posFinal.first][Camino[k].posFinal.second] = 'O';
  Tablero[Camino[k].fichaComida.first][Camino[k].fichaComida.second] = '-';
}


void Continental::RestauraTablero(int pos){
  Tablero[Camino[pos].posOriginal.first][Camino[pos].posOriginal.second] = 'O';
  Tablero[Camino[pos].posFinal.first][Camino[pos].posFinal.second] = '-';
  Tablero[Camino[pos].fichaComida.first][Camino[pos].fichaComida.second] = 'O';
}


void Continental::ConstruyeMovimiento(int i, int j, int mov, int k){
  Camino[k].posOriginal.first = i;
  Camino[k].posOriginal.second = j;

  switch (mov) {
    case 1:
      Camino[k].posFinal.first = i;
      Camino[k].posFinal.second = j-2;
      Camino[k].fichaComida.first = i;
      Camino[k].fichaComida.second = j-1;
      break;
    case 2:
      Camino[k].posFinal.first = i-2;
```

```
      Camino[k].posFinal.second = j;
      Camino[k].fichaComida.first = i-1;
      Camino[k].fichaComida.second = j;
      break;
    case 3:
      Camino[k].posFinal.first = i;
      Camino[k].posFinal.second = j+2;
      Camino[k].fichaComida.first = i;
      Camino[k].fichaComida.second = j+1;
      break;
    case 4:
      Camino[k].posFinal.first = i+2;
      Camino[k].posFinal.second = j;
      Camino[k].fichaComida.first = i+1;
      Camino[k].fichaComida.second = j;
      break;
  }
}


bool Continental::ProcesaSolucion(){
  int nfichas = 0;
  for(int i = 0; i<7; ++i)
    for(int j = 0; j<7; ++j)
      if(Tablero[i][j] == 'O')
        nfichas++;

  if(nfichas != 1)
    return false;

  for(int i = 0; i<7; ++i)
    for(int j = 0; j<7; ++j)
      if(Tablero[i][j] == 'O')
        if(!(i == 3 && j == 3))
          return false;

  return true;
}


void Continental::ImprimeSolucion(){
  for(unsigned i = 0; i < Camino.size(); ++i)
    cout << "[<" << Camino[i].posOriginal.first << "," << Camino[i].posOriginal.second << "><" <<
 Camino[i].posFinal.first << "," << Camino[i].posFinal.second << ">]";
}


void Continental::ImprimeSolucionPasos(){
    Continental aux;
    aux.Camino = this->Camino;
    cout << endl << "Situación inicial: " << endl << endl;
    aux.ImprimeTablero();
    cin.get();
    cout << endl;
    for(unsigned i = 0; i < aux.Camino.size(); ++i){
      cout << "Paso " << i+1 << " de " << MOVIMIENTOS << " : ";
      cout << "[<" << aux.Camino[i].posOriginal.first << "," << aux.Camino[i].posOriginal.second
<< "><" << aux.Camino[i].posFinal.first << "," << aux.Camino[i].posFinal.second << ">]" << endl;
      cout << endl;
      aux.ActualizaTablero(i);
```

```
      aux.ImprimeTablero();
      cout << endl << "Pulsa enter para ver el siguiente paso..." << endl;
      cin.get();
    }
}


void Continental::ImprimeTablero(){
  for(int i = 0; i<7; ++i){
    for(int j = 0; j<7; ++j){
      cout << "[";
      if(Tablero[i][j] == 'X')
        cout << "\e[1;31mX\e[0m";
      else if(Tablero[i][j] == 'O')
        cout << "\e[1;34mO\e[0m";
      else
        cout << " ";
      cout << "]";
    }
    cout << endl;
  }
}


void Continental::JuegoContinental_rec(int k){
  if(ProcesaSolucion())
    encontrado = true;
  else{
    //Buscar la siguiente ficha:
    for(int i = 0; i<7 && !encontrado; ++i)
      for(int j = 0; j<7 && !encontrado; ++j)
        if(Tablero[i][j] == 'O')
          //Siguiente movimiento para esa ficha:
          for(int m = 1; m <= 4 && !encontrado; ++m)
            if(MovFactible(i,j,m)){
              ConstruyeMovimiento(i,j,m,k);
              ActualizaTablero(k);
              JuegoContinental_rec(k+1);
            }
  }
  if(!encontrado)
    RestauraTablero(k-1);

}

void Continental::JuegoContinental(){
  JuegoContinental_rec(0);
}
```

MAIN

```
int main(){
  Continental con;
  high_resolution_clock::time_point tantes, tdespues;
  duration<double> duracion;
  char res;

  cout << "Calculando una solución..." << endl;
```

```
  tantes = high_resolution_clock::now();
  con.JuegoContinental();
  tdespues = high_resolution_clock::now();

  duracion = duration_cast<duration<double>>(tdespues - tantes);
  cout << "Solución encontrada. Tiempo transcurrido: " << duracion.count() << " segundos" << endl
;

  cout << "Desea ver la solución paso por paso? (s/n): ";
  cin >> res;
  res = tolower(res);
  while(res != 's' && res != 'n')
    cin >> res;

  if(res == 's'){
    con.ImprimeSolucionPasos();
    cout << endl;
  }
  else{
    cout << "Los pasos son:" << endl;
    con.ImprimeSolucion();
    cout << endl;
  }
}
```

Para finalizar, contamos en último lugar con un ejemplo (resumido) de la ejecución del mismo: