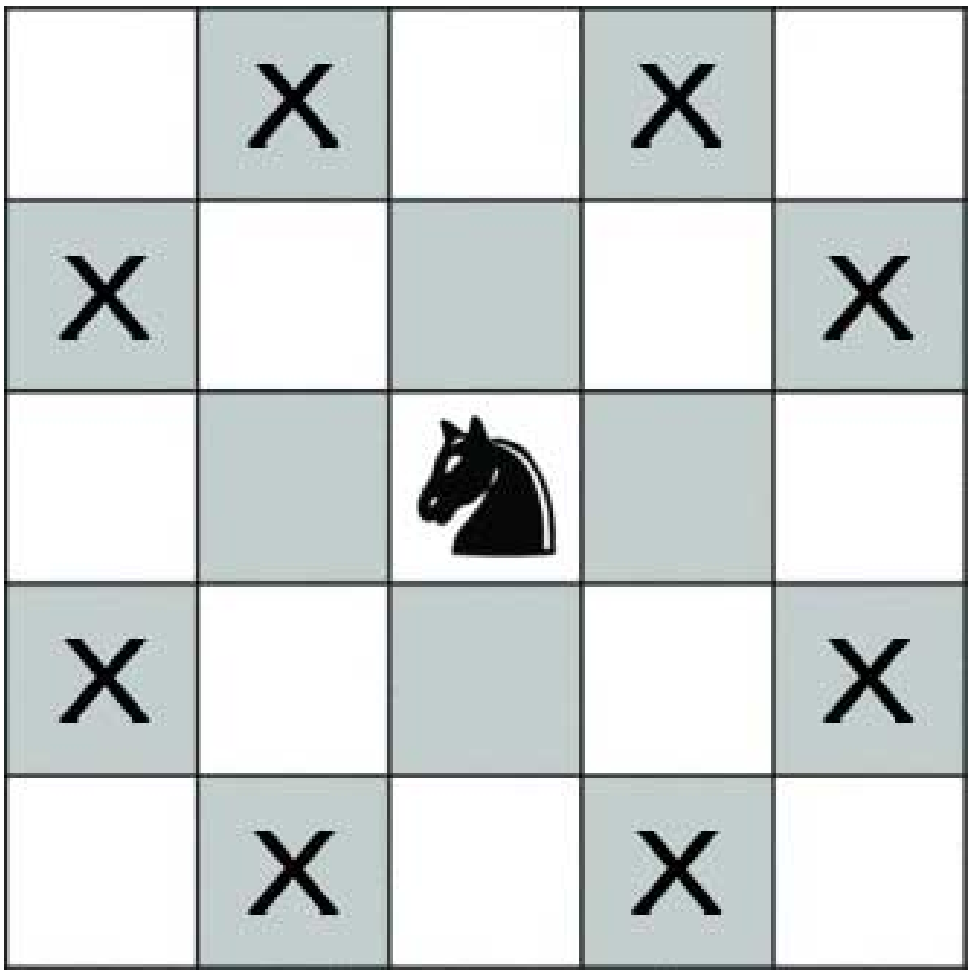
Recubrimiento de un grafo por vértices (Recorrido del caballo.)

practica 4: BÚSQUEDA



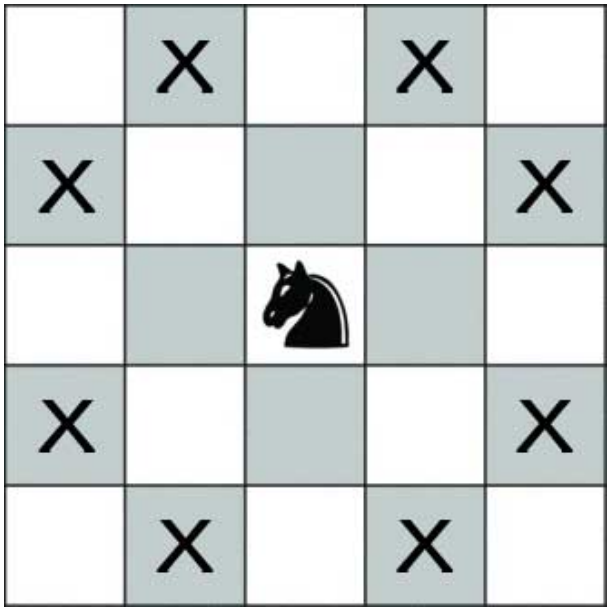
AUTORES:
Pablo Moreno Megías, Diego Lerena García, Manuel Vallejo Felipe, Ángel Díaz de la Torre, Francisco Navarro Morales, Marcel Kemp Muñoz y David Redondo Correa

Algorítmica [PRACTICAS]
Segundo curso del Grado de Ingeniería Informática.
Universidad de Granada.
curso 2016-2017.

Índice

1 Análisis del problema

Se tiene un tablero de Ajedrez y se sitúa un caballo solitario en cualquier casilla de este y se quiere, utilizando la reglas del juego, que el caballo recorra todas y cada una de las casillas del tablero sin repetir ninguna casilla visitada.



Si observamos la figura y enmueramos, definimos y ordenamos los movimientos del cabllao de ajedrez (1-8), tomando el primer movimiento como el superior más a la de la izquierda y siguiendo un sentido horario tenemos los 8 movimientos del cabllo de ajedrez desde una posición (xk,ym) de un tablero cuadrado hacia una posición (xk+1,ym+1), convirtiendo el propio tablero en un sistema de coordenadas cartesianas o un mapa:

posición 1: se le restan 2 posiciones a la x y se le suma una 1posicion a la y

posición 2: se le resta 1 a la x y se le suman 2 a la y

posición 3: se le suma 1 a la x y se le suman 2 a la y

posición 4: se le suman 2 a la x y se le suma 1 a la y

posición 5: se le suman 2 a la x y se le resta 1 a la y

posición 6: se le suma 1 a la x y se le restan 2 a la y

posición 7: se le resta 1 a la x y se le restan 2 a la y

posición 8: se le restan 2 a la x y se le resta 1 a la y

No hace falta decir que un movimiento es válido si y solo si el movimiento está dentro del tablero, pues entonces tenemos que tener cuidado en los bordes.

En casos de un tablero de dimensiones menores a 4x4 es posible que se dé el caso de que sea irresoluble el problema debido al propio movimiento del caballo.

2 Escoger una técnica (BackTracking o Branch and Bound) y justificar su elección frente a la otra metodología no seleccionada.

La técnica que hemos elegido para el problema del recorrido del caballo ha sido Backtracking

¿Por qué no Brach and Bound? Somos conscientes de que se puede resolver mucho más eficiente con Branch and Bound y que quizá utilizar branch and bound sea poco práctico teniendo en cuenta la cantidad de recorridos posibles para el caballo en un tablero 8x8. No obstante, es muy complicado establecer la bondad de un nodo o una estrategia de ramificación tan buena para que vaya podando y puede ser que vaya dejando soluciones que pueden ser factibles (si no lo hacemos con una estimación óptima). Un solución para ello sería algo parecido a un pathfinding realizado en las prácticas de IA. Si fusionamos esos dos conceptos obtendríamos un resultado preferible, pero encontrar un criterio que permita implementar una solución Branch and Bound es algo complejo que, probablemente, escape a los objetivos de esta asignatura.

Por ello, nos decantamos por Backtracking que aunque es menos eficiente y por consiguiente tarda más tiempo en determinar si hay solución o no, lo resuelve en tiempo razonable (que es el objetivo) y además es bastante más rápido de implementar y con tres parciales y dos finales a la vuelta de la esquina no era plan de ponerse a hacer el mongolo pensando una solución Branch and Bound.

3 Diseño de la solución, describiendo cada una de las componentes de la misma relacionándola con las componentes de un algoritmo BackTracking o Branch&Bound, según la técnica escogida.

3. Diseño de la solución

Decisiones de implementación

- El tablero se representará mediante una matriz
- Cada cuadro almacenará en que posición del viaje ha sido visitado
- Movimientos del caballo

Movimientos:

- Dada una posición inicial (u, v)
- Las posibles nuevas posiciones se obtienen:

  x = u + Movimientos [i].X
  y = v + Movimientos [i].Y

  • Si x ó y no están en el intervalo [1, n], el movimiento no es válido
  • Además es necesario que el cuadro (x, y) este a 0 (Ta[x, y] = 0)
  • No están disponibles los 8 movimientos posibles en todos los cuadros del tablero

Componentes BackTracking:

- Ensayar(int i, int x, int y, bool &exito,int tablero[MAX][MAX]) es la función donde se va a implementar el BackTracking, y que será usada recursivamente hasta obtener la solución.
- Dentro de la función tenemos:
  1. Un bool Q, que será la que nos indique si hemos resuelto el problema, y una variable k, que proporcionará el nº de movimientos intentados en la misma posición.
  2. Luego, está el bucle que no parará hasta que este la solución o se hayan recorrido 8 movimientos en la misma posición (k).
  3. Una vez dentro, aumentamos k, y comprobamos si puede realizarse un movimiento. Si se puede realizar, volvemos a llamar a la función recursiva, pero si no se encuentra la solución, se pone la posición del tablero a 0. Y así hasta encontrar una solución o que se agoten todas las opciones del tablero (8^8 en este caso).

4 Esqueleto del algoritmo (pseudocódigo) que soluciona el problema, explicando cómo se ha incorporado cada una de las componentes de diseño de la técnica (BackTracking o Branch&Bound, según se aplique)

```
procedure Ensayar ( i      : Natural;
                    x, y   : Natural;
                    exito : Boolean) is
    k: Natural;
    Q: Boolean;
    u, v: Natural;   -- Nuevo movimiento
begin
   k := 0;
 loop
    k  := k + 1;
    Q := false;
    NuevoMov (u, v, k);
   if esValido (u, v) and T(u, v) = 0 then
      T(u, v) := i;
      if i < n**n then
          Ensayar (i+1, u, v, Q);
          if not Q then
             T (u, v) := 0;   --- Vuelta Atrás
          end if;
      else
          Q := true;   --- Completado
      end if;
   end if;
   exit when Q or (k = 8);
 end loop;
 exito := Q;
end Ensayar;
```

```
procedure ViajeDelCaballo is
    T      : Tablero;
     x, y : Natural ;
     exito: Boolean;
begin
    T := (others => 0);
     x := 4;  y := 4;   --- Posición inicial
     T(x, y) := 1;
     Ensayar (2, x, y, exito);
     if exito then
       ImprimirTablero (T);
     else
       Put_Line (“No se ha encontrado la solución”);
     end if;
end ViajeDelCaballo;
```

5 Explicación del funcionamiento del algoritmo, sobre un caso de ejemplo pequeño propuesto por los estudiantes.

El algoritmo comienza con el caballo colocado en la casilla que se le indique, a la que se le asigna el número 1. A continuación va calculando las casillas resultantes de efectuar cada uno de los 8 posibles movimientos, si el movimiento que calcula para un movimiento es válido (lleva a una casilla del tablero que aún no ha sido visitada) vuelve a llamar a la función y avanza. En caso de no encontrar ningún movimiento válido para una posición concreta, el algoritmo retrocede, marca un 0 en la última posición válida que se había calculado y sigue calculando sus hermanos. Para ello utiliza una matriz rellena a -1 (representa que la casilla no ha sido visitada) qie va rellenando con el turno o momento en el que el caballo se encuentra en cada casilla. Ejemplo de ejecución para un tablero 3x3:

```
Turno: 1 movimiento: 0
--------------------------------
 -1  -1  -1

 -1  -1  -1

 -1  -1   0

--------------------------------

Turno: 2 movimiento: 0
--------------------------------
 -1   1  -1

 -1  -1  -1

 -1  -1   0

--------------------------------

Turno: 2 movimiento: 1
--------------------------------
 -1   1  -1

 -1  -1  -1

 -1  -1   0

--------------------------------

Turno: 2 movimiento: 2
--------------------------------
 -1   1  -1

 -1  -1  -1

 -1  -1   0

--------------------------------
```

```
Turno: 3 movimiento: 0
--------------------------------
 -1   1  -1

 -1  -1  -1

  2  -1   0

--------------------------------

Turno: 3 movimiento: 1
--------------------------------
 -1   1  -1

 -1  -1  -1

  2  -1   0

--------------------------------

Turno: 3 movimiento: 2
--------------------------------
 -1   1  -1

 -1  -1  -1

  2  -1   0

--------------------------------

Turno: 3 movimiento: 3
--------------------------------
 -1   1  -1

 -1  -1  -1

  2  -1   0

--------------------------------
```

```
Turno: 3 movimiento: 4
--------------------------------
 -1   1  -1

 -1  -1  -1

  2  -1   0

--------------------------------

Turno: 3 movimiento: 5
--------------------------------
 -1   1  -1

 -1  -1  -1

  2  -1   0

--------------------------------

Turno: 4 movimiento: 0
--------------------------------
 -1   1  -1

 -1  -1   3

  2  -1   0

--------------------------------

Turno: 4 movimiento: 1
--------------------------------
 -1   1  -1

 -1  -1   3

  2  -1   0

--------------------------------
```

...

6 Enunciado de un problema o caso real donde se pueda utilizar la técnica seleccionada para solucionarlo (distinto de los comentados en clase).

El algoritmo sirve para resolver el famoso problema del movimiento del caballo en el juego del ajedrez. El esquema que utiliza se puede extrapolar para resolver otros problemas de búsqueda pero estaríamos hablando del esquema branch and bound típico, puesto que este no tiene nada de especial.

7 Cálculo del orden de eficiencia teórico del algoritmo.

El algoritmo prueba todos los caminos posibles hasta que encuentra uno válido. Suponiendo que tuviera que probar todos los caminos posibles: Como en total el caballo tiene que visitar $MAX \times MAX$ casillas (sea $MAX,\ n$, n^2 casillas)...

Si suponemos que de cada nodo surgen siempre 5 nodos válidos (para establecer una aproximación, podría darse que un nodo tuviera 8 hijos válidos o incluso 0 hijos válidos) y dado que el nivel del árbol de búsqueda debe ser n^2 (un nivel por cada posición posible del tablero), como de cada nodo suponemos que salen 5 nodos, de la raíz se obtienen 5 nodos, de cada uno de esos nodos se obtienen otros 5 (en total 25) y de cada uno de esos 25, otros 5... es decir, que en total se evaluarían 5^x nodos siendo x el nivel del nodo, como $x = n^2$, el número de nodos a evaluar sería $5^{n^2} = 5^{2n}$ que, para un tablero 8x8 sería un total de $5^16 = 152587890625$ nodos, si el número de nodos válidos fuera mayor, por ejemplo 8 (que sería una buena cota superior porque es imposible que todos los nodos den siempre 8 hijos válidos) se obtendría un total de $2{,}814749767 \times 1014$ nodos.

En conclusión, suponiendo que cada nodo se evalúa en $O(1)$, la eficiencia teórica del algoritmo sería

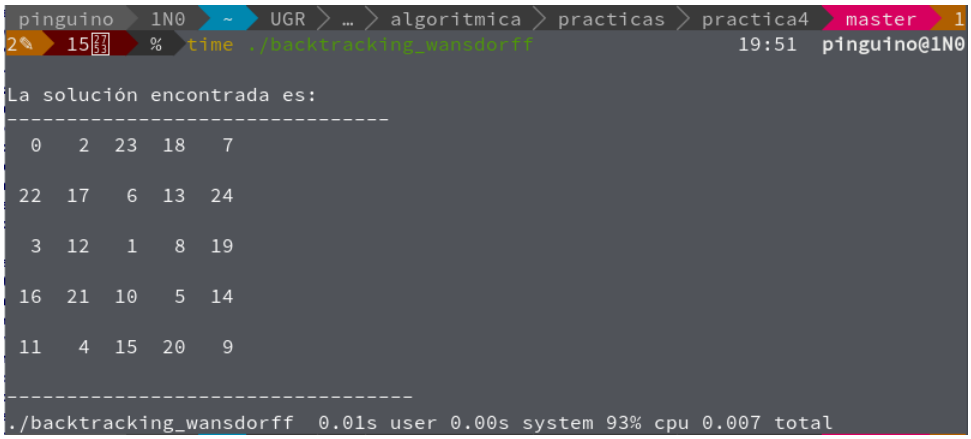
$O(k^{2n})$ donde k es el número medio de hijos válidos por nodo (al principio y dependiendo del tamaño del tablero podrá tomar valores próximos a 8 y en niveles profundos del árbol tomará valores cercanos a 0, por lo que una buena aproximación sería 4 o 5).

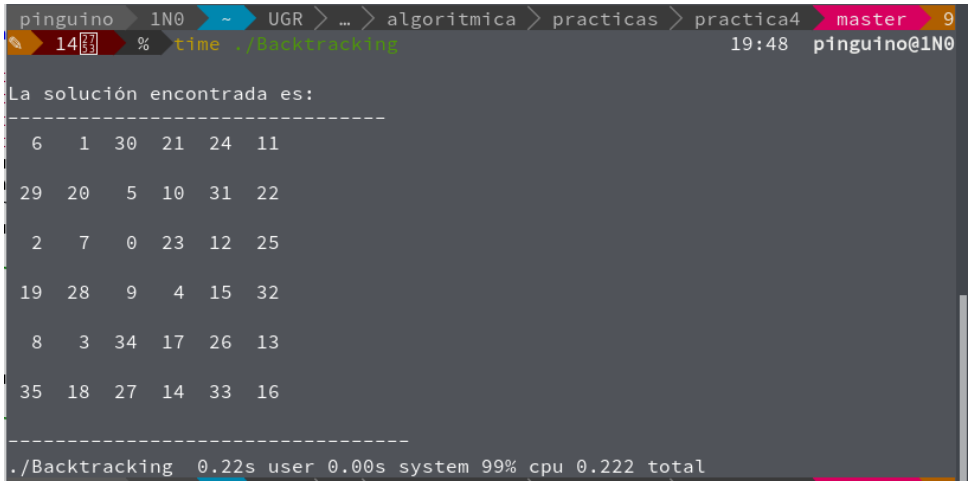
8 8. Instrucciones sobre cómo compilar y ejecutar el código de la práctica.

El código es un .cpp, se compila con g++ sin ningún parámetro especial necesario. Si se desea modificar la casilla inicial o el tamaño del tablero basta con modificar los valores de las variables x e y en el main o de la constante global MAX (que representa el número de filas y de columnas del tablero).

El código por defecto se ejecuta en un tablero 7x7 porque la solución es inmediata. En un tablero 8x8 tarda unas horas en encontrar la respuesta, pero la encuentra.

Algunas ejecuciones con distintos tamaños:

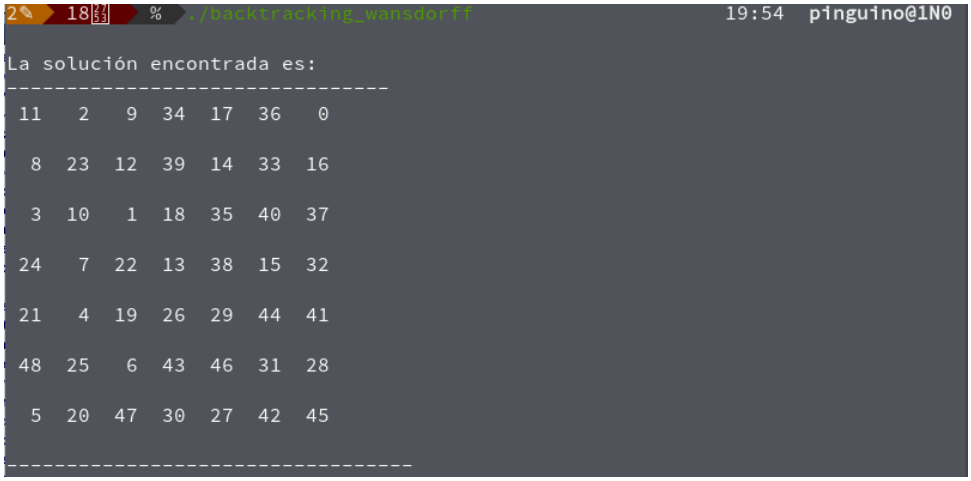




```
pinguino 1N0 ~ UGR … algoritmica practicas practica4 master 1
1✎ 14 % time ./Backtracking                                19:48  pinguino@1N0

La solución encontrada es:
--------------------------------
 10   1   8  23  16  45  48

  7  24  11  46  13  42  15

  2   9   0  17  22  47  44

 25   6  27  12  43  14  41

 28   3  18  21  38  35  32

 19  26   5  30  33  40  37

  4  29  20  39  36  31  34

--------------------------------
./Backtracking  0.81s user 0.00s system 99% cpu 0.807 total
```

```
2✎ 18 % ./backtracking_wansdorff                           19:53  pinguino@1N0

La solución encontrada es:
--------------------------------
  0   2  23  18   7

 22  17   6  13  24

  3  12   1   8  19

 16  21  10   5  14

 11   4  15  20   9

--------------------------------
```

```
pinguino 1N0 ~ UGR … algoritmica practicas practica4 master 1
2✎ 15 % ./backtracking_wansdorff                           19:52  pinguino@1N0

La solución encontrada es:
--------------------------------
 35   2   9  16  29  26

  8  15   6  27  10  17

  3  34   1  30  25  28

 14   7  22   5  18  11

 33   4  13  20  31  24

  0  21  32  23  12  19

--------------------------------
```


2✎ 18 % ./backtracking_wansdorff 19:54 pinguino@1N0
La solución encontrada es:
11 2 9 34 17 36 0
8 23 12 39 14 33 16
3 10 1 18 35 40 37
24 7 22 13 38 15 32
21 4 19 26 29 44 41
48 25 6 43 46 31 28
5 20 47 30 27 42 45