Divide y Venceras
Vuelta Atrás

PROGRAMACIÓN

Hugo Araya Carrasco

Concepto

La técnica de diseño de algoritmos llamada "divide y vencerás" (divide and conquer) consiste en descomponer el problema original en varios sub problemas más sencillos, para luego resolver éstos mediante un cálculo sencillo. Por último, se combinan los resultados de cada sub-problema para obtener la solución del problema original.

Algoritmo

El pseudocódigo sería:

```
funcion divide_y_venceras_1(problema){
    descomponer el problema en n subproblemas más pequeños;
    para i=1 hasta n hacer
        resolver el subproblema k;
    combinar las n soluciones;
}
```

Ejemplo

Un ejemplo de "divide y vencerás" es el quicksort. En el, se dividía el arreglo en dos sub-arreglos, para luego resolver cada uno por separado, y unirlos.

El tiempo necesario para ordenar un arreglo de elementos mediante el método de la burbuja es cuadrático: kN^2.

Si dividimos el arreglo en dos y ordenamos cada uno de ellos, el tiempo necesario para resolverlo es ahora: $k(N/2)^2+k(N/2)^2=(kN^2)/2$. El tiempo necesario para ordenarlo es la mitad, pero sigue siendo cuadrático.

Usando Recursividad

Un algoritmo del tipo:

```
funcion divide_y_venceras(problema){
   si el problema es trivial
      entonces resolver el problema;
   si no es trivial{
      descomponer el problema en n subproblemas más pequeños;
      para i=1 hasta n hacer
         divide_y_venceras(subproblema_k);
      combinar las n soluciones;
   }
}
```

Determinar el Umbral

Uno de los aspectos que hay que tener en cuenta en los algoritmos de divide y vencerás es dónde colocar el umbral, esto es, cuándo se considera que un subproblema es suficientemente pequeño como para no tener que dividirlo para resolverlo.

Normalmente esto es lo que hace que un algoritmo de divide y vencerás sea efectivo o no.

Por ejemplo, en el algoritmo de ordenación quicksort, cuando se tiene un array de longitud 3, es mejor ordenarlo utilizando otro algoritmo de ordenación (con 6 comparaciones se puede ordenar), ya que el quicksort debe dividirlo en dos sub-arrays y ordenar cada uno de ellos, para lo que utiliza más de 6 comparaciones.

Ejemplos

Problema de los puntos más cercanos

El problema es: "dado un conjunto de puntos P, hallar el par de puntos más cercanos". La distancia entre dos puntos *i* y *j* es :

$$\mathbf{sqrt[(xi-xj)^2+(yi-yj)^2].}$$

BackTracking (Vuelta Atrás)

Los algoritmos de vuelta atrás se utilizan para encontrar soluciones a un problema.

No siguen unas reglas para la búsqueda de la solución, simplemente una búsqueda sistemática, hay que probar todo lo posible hasta encontrar la solución o encontrar que no existe solución al problema.

Para conseguir este propósito, se separa la búsqueda en varias búsquedas parciales o subtareas. Asimismo, estas subtareas suelen incluir más subtareas, por lo que el tratamiento general de estos algoritmos es de naturaleza recursiva.

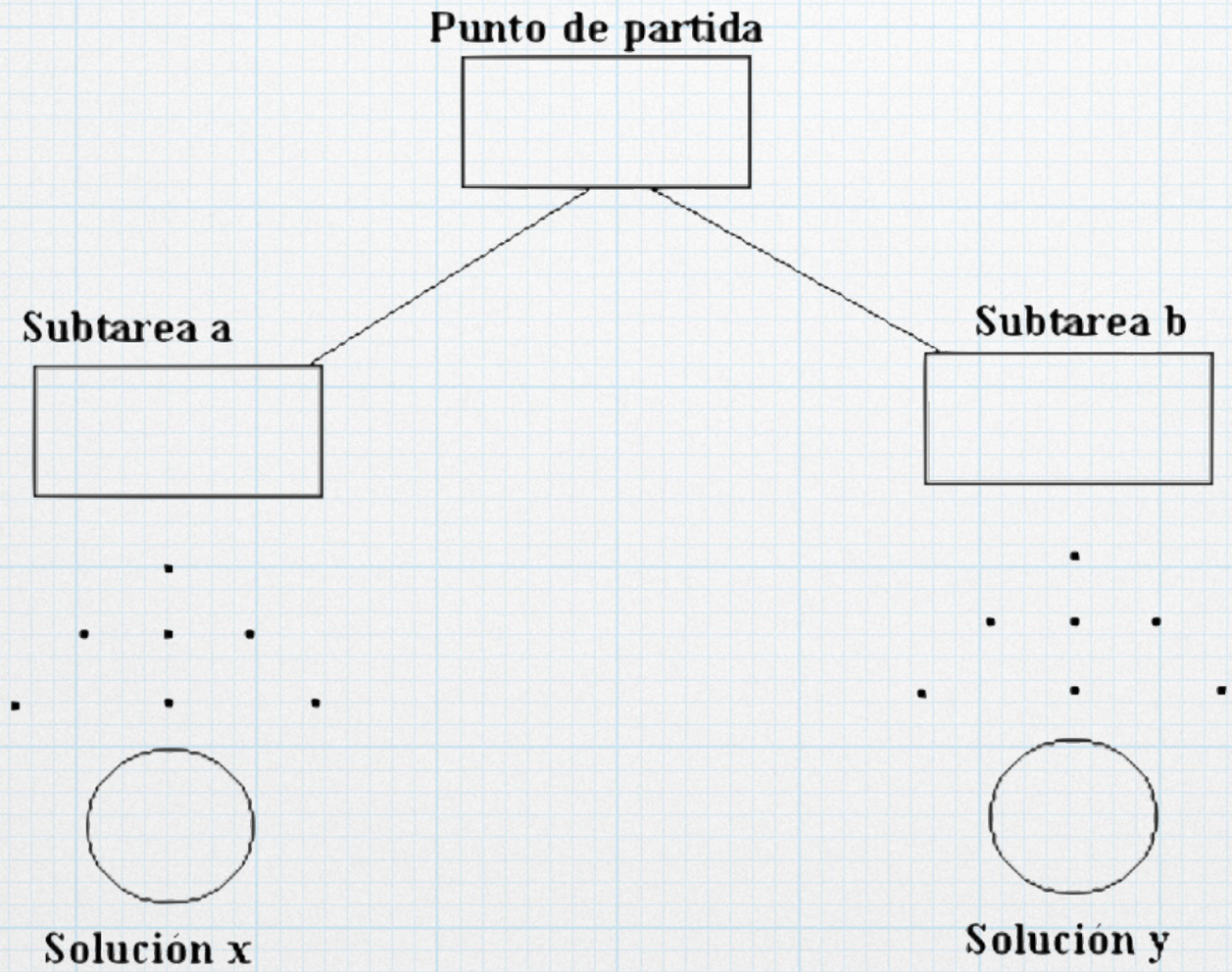
BackTracking

¿Por qué se llaman algoritmos de vuelta atrás?. Porque en el caso de no encontrar una solución en una subtarea se retrocede a la subtarea original y se prueba otra cosa distinta (una nueva subtarea distinta a las probadas anteriormente).

Puesto que a veces nos interesa conocer múltiples soluciones de un problema, estos algoritmos se pueden modificar fácilmente para obtener una única solución (si existe) o todas las soluciones posibles (si existe más de una) al problema dado.

Esquema

Gráficamente se puede ver así:



Esquema: Una solución

```
procedimiento ensayar (paso : TipoPaso)
    repetir
        seleccionar_candidato
            if aceptable then
                anotar_candidato
                if solucion_incompleta then
                    ensayar(paso_siguiente)
                if no acertado then
                    borrar_candidato
                else
                    anotar_solucion
                    Acertado <- cierto
    hasta que (acertado = cierto) o (candidatos_agotados)
fin procedimiento
```

Esquema Todas las Soluciones

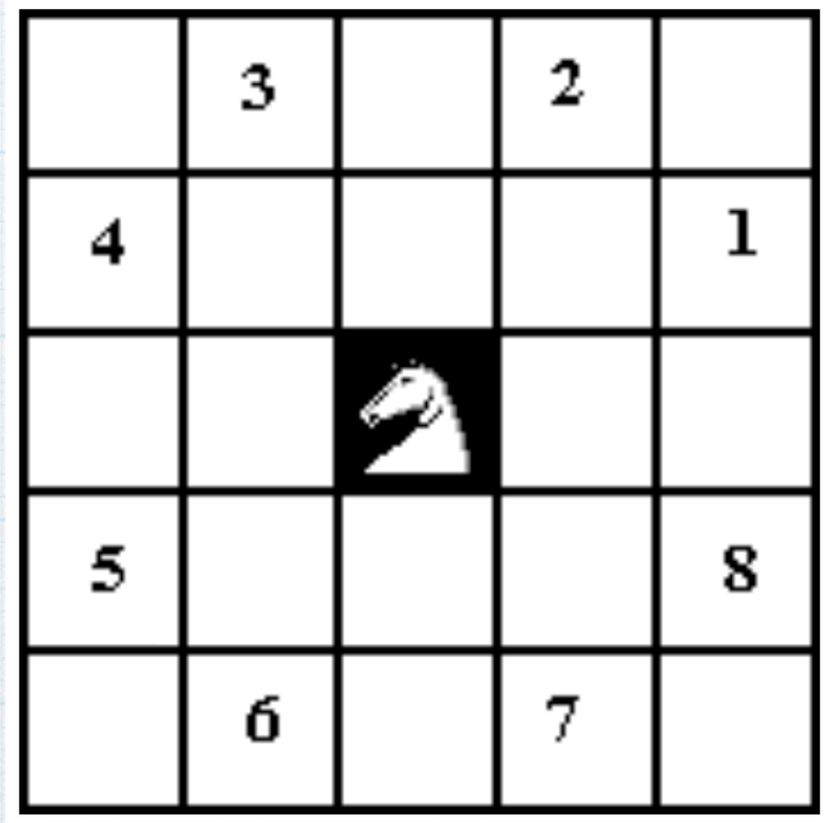
```
procedimiento ensayar (paso : TipoPaso)
        para cada candidato hacer
            seleccionar candidato
            if aceptable then
                anotar_candidato
                if solucion_incompleta then
                    ensayar(paso_siguiente)
                else
                    almacenar_solucion
                borrar_candidato
        hasta que candidatos_agotados
fin procedimiento
```

Salto del Caballo

Se dispone de un tablero rectangular, por ejemplo el tablero de ajedrez, y de un caballo, que se mueve según las reglas de este juego. El objetivo es encontrar una manera de recorrer todo el tablero partiendo de una casilla determinada, de tal forma que el caballo pase una sola vez por cada casilla. Una variante es obligar al caballo a volver a la posición de partida en el último movimiento. Para resolver el problema hay que realizar todos los movimientos posibles hasta que ya no se pueda avanzar, en cuyo caso hay que dar marcha atrás, o bien hasta que se cubra el tablero. Además, es necesario determinar la organización de los datos para implementar el algoritmo.

¿Cómo se mueve un caballo?. Para aquellos que no sepan jugar al ajedrez se muestra un gráfico con los ocho movimientos que puede realizar. Estos movimientos serán los ocho candidatos.

Salto del Caballo



Con las coordenadas en la que se encuentra el caballo y las ocho coordenadas relativas se determina el siguiente movimiento.

Eje_X = {2, 1, -1, -2, -2, -1, 1, 2}

Eje_Y = {1, 2, 2, 1, -1, -2, -2. -1}

Salto del Caballo

1	16	11	6	3
10	5	2	17	12
15	22	19	4	7
20	9	24	13	18
23	14	21	8	25

El tablero, del tamaño que sea, se representará mediante un array bidimensional de números enteros. Arriba se muestra una imagen con un tablero de tamaño 5x5 con todo el recorrido partiendo de la esquina superior izquierda.

Salto del Caballo

Cuando se encuentra una solución, una variable que se pasa por referencia es puesta a 1 (cierto). Puede hacerse una variable de alcance global y simplificar un poco el código, pero esto no siempre es recomendable.

Para codificar el programa, es necesario considerar algunos aspectos más, entre otras cosas no salirse de los límites del tablero y no pisar una casilla ya cubierta (selección del candidato). Se determina que hay solución cuando ya no hay más casillas que recorrer.