|  |  |
| --- | --- |
| UTN UNIVERSIDAD TECNOLÓGICA NACIONAL FACULTAD REGIONAL BUENOS AIRESINGENIERÍA EN SISTEMAS DE INFORMACIÓN“ALGORITMOS COMPLEJOS PARA ESTRUCTURAS DE DATOS AVANZADAS”Materia Electiva de 3er. Nivel  |  | | --- | | **Trabajo Monográfico - Tema: Backtracking**  **Ciclo Lectivo 2014** |   **Profesores:**   * *Ing. Sznajdleder, Pablo Augusto* * *Ing. Bruno, Oscar*   **Integrantes del grupo:**   * *Agrelo Brito, Carlos* * *Belfiore, Julieta* * *Codeiro, Pablo* * *Comparin, Leandro* * *Noziglia, Julián* |

Backtracking

Introducción

Backtracking es una estrategia para encontrar soluciones a problemas que satisfacen restricciones. Vamos a plantear un ejercicio como ejemplo. Proponemos al lector intentar resolver el problema con un algoritmo que le parezca acertado.

En un tablero de ajedrez de 8x8 casilleros, colocamos un caballo en una de sus esquinas. El objetivo es lograr que el caballo pise cada uno de los casilleros del tablero, marcándolos (de 1 a 64) sin volver a pisar en un casillero en el que ya estuvo, sin salirse del tablero y usando los movimientos permitidos en el ajedrez para el caballo.

Lo interesante de este ejercicio es que las condiciones del problema nos llevan indefectiblemente al uso de la estrategia de backtracking para su resolución. Primero, pensaremos en los datos que tenemos del problema y luego pensaremos como podemos usar esos datos de manera adecuada (haciendo uso de nuestra intuición).

El principal dato que debemos tener claro es el movimiento del caballo. En el juego de ajedrez, el caballo tiene ocho movimientos posibles (escribiremos los movimientos pensando en el tablero como una matriz de 8x8, y los movimientos serán de X casillas por X casillas[[1]](#footnote-1)):

* [2;1]
* [2;-1]
* [-2;1]
* [-2;-1]
* [1;2]
* [-1;2]
* [1;-2]
* [-1;-2]

Los otros datos que podemos tener en cuanta, son la dimensión del tablero (8x8), el casillero de inicio de movimientos [0;0] y el dato de que llegamos a resolver el problema, es decir, cuando llegamos a marcar el número 64 en los casilleros.

Primero, debemos pensar al movimiento del caballo como una función, que podemos armar independientemente del problema a resolver. Esta función la llamaremos "mueveCaballo" y la utilizaremos luego, en la resolución algorítmica. Una vez armada la función, podemos abstraernos del problema de mover la ficha y concentrarnos en el resto de la solución.

Sabemos que el ejercicio se termina cuando llegamos a 64 casilleros, entonces deberíamos hacer una condición While al inicio "mientras que la casilla marcada sea < a 64". Seguido, debemos considerar que el caballo comienza en la posición [0;0], esto sería inicializar una variable de posición del caballo, y marcar la casilla con un 1. Después, haríamos otro While "mientras que el movimiento caiga dentro de los límites del tablero y el casillero al que se mueva este vacío", y por último, llamaríamos a la función moverCaballo para que se mueva en una de los movimientos posibles (al azar) y marcaríamos la nueva casilla a donde el caballo se movió. Esto enunciado hasta ahora es la "jugada" base para resolver este algoritmo.

El problema es que pasa cuando no entra al último While, esto es, cuando la pieza sigue una cantidad de movimientos que lo deja en un camino cerrado: hay casillas todavía por marcar, pero no hay posibilidades donde moverse. Es ahí donde la estrategia de Backtracking tiene sentido. En este punto en que la pieza se estanca, el algoritmo debe volver sobre sus pasos, desmarcar la casilla anterior a la actual, y buscar otro movimiento, distinto al que probó anteriormente. Si llega a un camino cerrado nuevamente, volverá a la casilla anterior por otro camino y así sucesivamente, volverá todas las casillas que sean necesarias, hasta que encuentre el camino correcto que solucione el problema.

Este problema es un antiguo problema matemático y muchos han intentado resolverlo. Se han encontrado numerosas formas de resolver este problema con este algoritmo, ósea, que existen muchos caminos posibles. Este ejercicio en un ejemplo particular de un problema más general: la ruta Hamiltoniana [[2]](#footnote-2)en la teoria de grafos.

Backtracking se traduce en español como "Vuelta atrás", ya que justamente el algoritmo se basa en esto: probara de una forma, y si no se llega al resultado esperado, volverá sobre sus pasos y probará otra forma, y así sucesivamente hasta encontrar la correcta. En conclusión, es un algoritmo de prueba y error.

Esquema del algoritmo

El siguiente esquema muestra un algoritmo base de las estrategia Backtracking. Luego podemos modificar el algoritmo para encontrar: a) una solucion cualquiera; b) todas las posibles soluciones; c) la mejor de todas las soluciones

**Funcion Backtracking** (Etapa i) **devuelve:** boolean

**Inicio**

Éxito = falso**;**

*IniciarOpciones*(i, GrupoOpciones o)**;**

**Repetir**

*SeleccionarnuevaOpcion*(o, Opcion n)**;**

**Si** (Aceptable(n)) **entonces**

*AnotarOpcion*(i, n)**;**

**Si** *SolucionCompleta*(i) **entonces**

Éxito = verdadero**;**

**Sino**

Éxito = Backtracking(i+1)**;**

**Si** Éxito = false **entonces**

*cancelamosAnotacion*(i, n)**;**

**finsi;**

**Finsi;**

**Finsi;**

**Hasta** (éxito = verdadero) **o** (*NoQuedanOpciones*(o))**;**

**Retorna** Éxito**;**

**Fin;**

Concepto

Hablando más técnicamente del algoritmo, en su forma básica, la idea de backtracking se asemeja a un recorrido en profundidad dentro de un grafo dirigido. Como el objetivo es encontrar soluciones para algún problema, se van construyendo soluciones parciales a medida que progresa el recorrido y estas soluciones limitan las regiones en las que se puede encontrar una solución completa. El recorrido tiene éxito si, procediendo de esta forma, se puede definir por completo una solución. En este caso el algoritmo puede, o bien detenerse (si lo único que se necesita es una solución del problema) o bien seguir buscando soluciones alternativas (si deseamos examinarlas todas). Por otra parte, el recorrido no tiene éxito si en alguna etapa la solución parcial construida hasta el momento no se puede completar. En tal caso, el recorrido vuelve atrás exactamente igual que en un recorrido en profundidad, eliminando sobre la marcha los elementos que se hubieran añadido en cada fase. Cuando vuelve a un nodo que tiene uno o más vecinos sin explorar, prosigue el recorrido de una solución.

Donde se puede aplicar

Para que este tipo de estrategia pueda ser aplicada, los problemas deben cumplir ciertas condiciones. Estos deben satisfacer un determinado tipo de restricciones (en el ejemplo, fueron los límites del tablero, la cantidad de casilleros, el movimiento del caballo, la casilla de inicio), donde el orden de los elementos de la solución no importan. Consiste en un conjunto de variables, y a cada una se le debe asignar un valor sujeto a las restricciones del problema (en el ejemplo, numerar las celdas de la matriz del 1 al 64). La técnica va creando todas las posibles combinaciones de elementos y su principal virtud es que en la mayoría de las implementaciones se puede evitar combinaciones estableciendo funciones de acotación (que serán explicadas más adelante) reduciendo el tiempo de ejecución.

En conclusión, podemos aplicar esta estrategia para los siguientes tipos de problemas:

1. Problemas de Decisión: Búsqueda de las soluciones que satisfacen ciertas restricciones.
2. Problemas de Optimización: Búsqueda de la mejor solución en base a a una función objetivo.

Implementación

En general se suele implementar esta solución con un procedimiento recursivo.  Así, en cada llamada al procedimiento se toma una variable y se le asignan todos los valores posibles, llamando a su vez al procedimiento para cada uno de los nuevos estados.

Aplicaciones

La técnica de backtracking es usada en muchos ámbitos de la programación, por ejemplo, para el cálculo de expresiones regulares, para tareas de reconocimiento de texto y de sintaxis de lenguajes regulares (compiladores), para la implementación de algunos lenguajes (como Prolog o Planner), para algoritmos de inteligencia artificial, entre otros usos.

Eficiencia y complejidad algorítmica

Este tipo de estrategia no siempre es eficiente, ya que, como ya dijimos, se basa en el método prueba y error, y en muchas ocasiones hay que recorrer todo el árbol o gran parte de él. También puede resultar ineficiente desde el lado de la recursividad por la memoria que gasta durante las diferentes llamadas recursivas.

Lo que debemos tener en cuenta a la hora de pensar en la eficiencia es:

* El número de nodos que se visitan para conseguir la solución
* El trabajo realizado en cada nodo (el coste de la función de ver si es aceptable hasta el momento)
* El costo general: en su peor caso será exponencial

Otros métodos

Se puede modificar esta estrategia para mejorar su eficacia y complejidad algorítmica. Por ejemplo, como las variables se pueden procesar en cualquier orden, generalmente es más eficiente intentar ser lo más restrictivo posible con las primeras (esto es, las primeras con menores valores posibles). Este proceso poda el [árbol de búsqueda](http://es.wikipedia.org/wiki/%C3%81rbol_(estructura_de_datos)) antes de que se tome la decisión y se llame a la subrutina recursiva.

Cuando se elige qué valor se va a asignar, muchas implementaciones hacen un examen hacia delante (Forward Checking), para ver qué valor restringirá el menor número posible de valores, de forma que se anticipa en preservar una posible solución y hace que la solución encontrada no tenga restricciones destacadas.

Buscando mejorar el backtracking, nacen nuevos métodos que derivan de este. Uno de los más usados, es el *Ramificación y Poda*. Este método es una modificación del backtracking, pero con la diferencia de que cada solución tiene asociado un costo y la solución que se busca es una de mínimo costo llamada óptima. Además de ramificar una solución padre, se trata de eliminar de consideración aquellos hijos cuyos descendientes tienen un costo que supera al óptimo buscado acotar el costo de los descendientes del hijo. La forma de acotar depende de cada problema, y bien hecha puede reducir eficientemente el tiempo de búsqueda de la solución óptima al "podar" los subarboles de descendientes costosos.

Conclusión

La estrategia de backtracking es útil para resolver numerosa cantidad y variedad de problemas, pero, en general, es muy costoso (en cuanto a complejidad algorítmica). Se pueden aplicar ciertas modificaciones para mejorar su eficiencia, e incluso existen métodos similares que utilizan la lógica de backtracking para funcionar, pero hay que tener cuidado porque cuando se agregan modificaciones se suman también las complejidades algorítmicas de esas modificaciones. Igualmente, como puede verse en este trabajo, las aplicaciones de este algoritmo son muchas, y esto es porque siempre llega a encontrar una solución (si existe).

1. El signo '-' significa, retroceder en los casilleros [↑](#footnote-ref-1)
2. Un camino hamiltoniano es un camino que pasa por cada vértice exactamente una vez. Un grafo que contiene un camino hamiltoniano se denomina un ciclo hamiltoniano si es un [ciclo](http://es.wikipedia.org/wiki/Grafo_ciclo) que pasa por cada vértice exactamente una vez (excepto el vértice del que parte y al cual llega). [↑](#footnote-ref-2)