

Transparencias-del-tema-7-El-Met...



PruebaAlien



Algorítmica



2º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación Universidad de Granada



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.







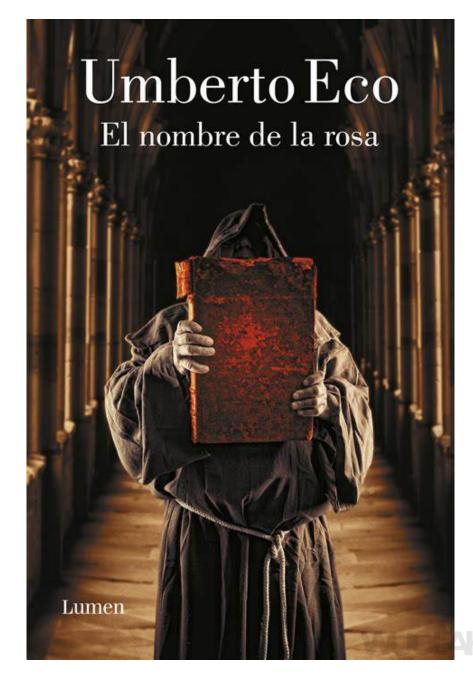


App Store

Algo conocido

Sólo hay una manera de encontrar la salida de un laberinto. Al llegar a cada nudo nuevo, o sea hasta el momento no visitado, se harán tres signos en el camino de llegada. Si se observan signos en alguno de los caminos del nudo, ello indicará que el mismo ya ha sido visitado, y entonces sólo se marcará un signo en el camino de llegada. Cuando todos los pasos de un nudo ya estén marcados, habrá que retroceder. Pero si todavía quedan uno o dos pasos sin marcar, se escogerá uno al azar, y se lo marcará con dos signos. Cuando se escoja un paso marcado con un solo signo, se marcarán dos más, para que ya tenga tres. Si al llegar a un nudo sólo se encuentran pasos marcados con tres signos, o sea, si no quedan pasos que aún falte marcar, ello indicará que ya se han recorrido todas las partes del laberinto.

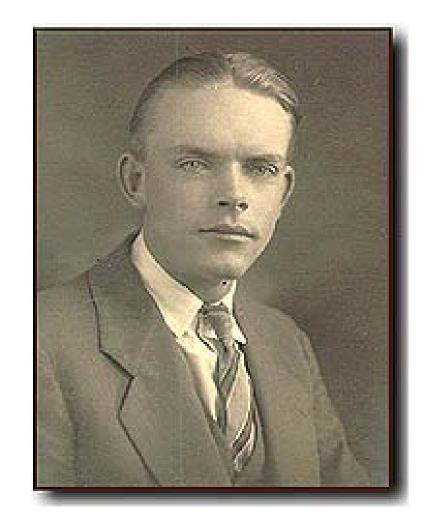
Umberto Eco, El Nombre de la Rosa (1980)





El método general

- El backtracking es una de las técnicas mas generales para la exploración de Grafos.
- Muchos problemas que buscan en un conjunto de soluciones o que buscan una solución optimal que satisfaga algunas restricciones, pueden resolverse con backtracking.
- El nombre backtrack lo acuñó D.H. Lehmer en los 50.
- R.J. Walker, dio una versión algorítmica (1960)
- Golomb y Baumert presentaron una descripción general del backtracking asociada a una gran variedad de aplicaciones.



Derrick H. Lehmer (1905-1991)



Backtracking

- Supongamos que tenemos que tomar una serie de decisiones entre una gran variedad de opciones donde,
 - □ No tenemos suficiente información como para saber que elegir
 - □ Cada decisión nos lleva a un nuevo conjunto de decisiones
 - □ Alguna sucesión de decisiones (pero posiblemente mas de una) pueden ser solución de nuestro problema
- El Backtracking es un método de búsqueda de esas sucesiones de decisiones que conduce a encontrar una que nos convenga





Descarga la APP de Wuolah.

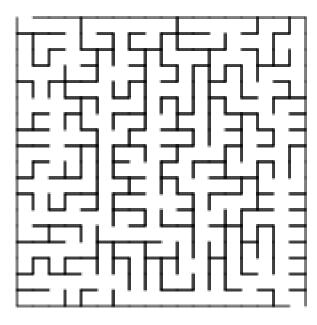
Ya disponible para el móvil y la tablet.





Ejemplo: Salir de un laberinto

- En un laberinto, encontrar el camino desde la entrada hasta el final
- En cada intersección hay que decidir entre varias alternativas:
 - Seguir recto
 - □ Girar a la izquierda
 - Girar a la derecha
- No tenemos suficiente información para decidir bien
- Cada elección nos lleva a otro conjunto de elecciones
- Una sucesión de elecciones pueden (o no) llevarnos a una solución
- Muchos recorridos de laberintos pueden resolverse con backtracking y también coloreos, puzzles ...







El método General

- Para aplicar el método backtracking, la solución deseada debe ser expresable como una n-tupla (x₁, ..., x_n) en la que x_i se elige de algún conjunto finito S_i.
- A menudo el problema a resolver trata de encontrar un vector que maximiza (o minimiza) una función criterio $P(x_1, ..., x_n)$.
- A veces, también se trata de encontrar todos los vectores que satisfagan P.

¿No se parece esto a las técnicas Greedy?



Un ejemplo

- Ordenar los enteros en A(1..n) es un problema cuya solución es expresable mediante una n-tupla en la que x_i es el índice en A del i-ésimo menor elemento.
- La función de criterio P es la desigualdad

$$A(x_i) \le A(x_{i+1}), 1 \le i \le n.$$

- El conjunto S_i es finito e incluye a todos los enteros entre 1 y n.
- La ordenación no es uno de los problemas que habitualmente se resuelven con backtracking





El método general

- Sea m_i el tamaño del conjunto S_i.
- Hay $m = m_1 \cdot m_2 \cdot ... \cdot m_n$ n-tuplas posibles candidatas a satisfacer la función P.
- El enfoque de la fuerza bruta propondría formar todas esas tuplas y evaluar cada una de ellas con P, escogiendo la que diera un mejor valor.
- Backtracking proporciona la misma solución pero en mucho menos de m intentos.





Top de tu grado.

RocioZabalo

Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.





El método general

- La idea básica es construir el mismo vector escogiendo una componente cada vez, y usando funciones de criterio modificadas $P_i(x_1, ..., x_n)$, que a veces se llaman funciones de acotación, para testear si el vector que se está formando tiene posibilidad de éxito.
- La principal ventaja de este método es que si a partir del vector parcial $(x_1, x_2, ..., x_i)$ se deduce que no se podrá construir una solución, entonces pueden ignorarse por completo $m_{i+1} \cdot ... \cdot m_n$ posibles test de vectores.





Diferencias con otras técnicas

- En los algoritmos greedy se construye la solución buscada, aprovechando la posibilidad de calcularla a trozos, pero con backtracking la elección de un sucesor en una etapa no implica su elección definitiva.
- En **Programación Dinámica**, la solución se construye por etapas a partir del Principio del Óptimo, y los resultados se almacenan para no tener que recalcular.
- Pero eso no es posible en Backtracking.





Tipos de restricciones en backtracking

- Muchos de los problemas que resolveremos usando backtracking requieren que todas las soluciones satisfagan un complejo conjunto de restricciones.
- En cualquier problema que consideremos, estas restricciones podrán dividirse en dos categorías:
 - explícitas e implícitas.





Tipos de restricciones en backtracking

Las **restricciones explícitas** son reglas que restringen cada x_i a tomar valores solo en un conjunto dado. Ejemplos comunes de restricciones explícitas son,

```
\begin{array}{ll} \square \, x_i \geq 0 & S_i = \{n^{\underline{os}} \text{ reales no negativos}\} \\ \square \, x_i = 0,1 & S_i = \{0,1\} \\ \square \, l_i \leq x_i \leq u_i & S_i = \{a: l_i \leq a \leq u_i\} \end{array}
```

- Las restricciones explícitas pueden depender o no del caso particular del problema a resolver.
- Las restricciones explícitas son condiciones frontera, es decir, vienen dadas por el propio planteamiento del problema.





Ver mis oportuni

Continúa donde le

Top de tu grado.

RocioZabalo

Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.





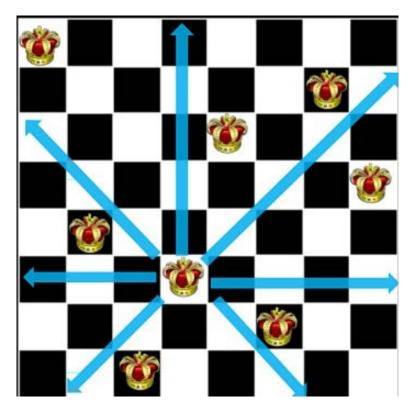
Tipos de restricciones en backtracking

- Todas las tuplas que satisfacen las restricciones explícitas definen un espacio de soluciones del caso que estamos resolviendo.
- Las **restricciones implícitas** determinan cual de las tuplas en ese espacio solución satisface la función de criterio, es decir, describen la forma en la que las x_i deben relacionarse entre si.
- Por tanto:
 - □ Las restricciones implícitas son las restricciones que impone el camino que define la solución.
- En definitiva, dan las tuplas del espacio solución que satisfacen la función de criterio.



El problema de las ocho reinas

- Un clásico problema combinatorio es el de colocar ocho reinas en una tablero de ajedrez de modo que no haya dos que se ataquen, es decir, que estén en la misma fila, columna o diagonal.
- Las filas y columnas se numeran del 1 al 8.
- Las reinas se numeran del 1 al 8.



Como cada reina debe estar en una fila diferente, sin perdida de generalidad podemos suponer que la reina i se coloca en la fila i.

Todas las soluciones para este problema, pueden representarse como 8-tuplas $(x_1,...,x_n)$ en las que x_i es la columna en la que se coloca la reina i.



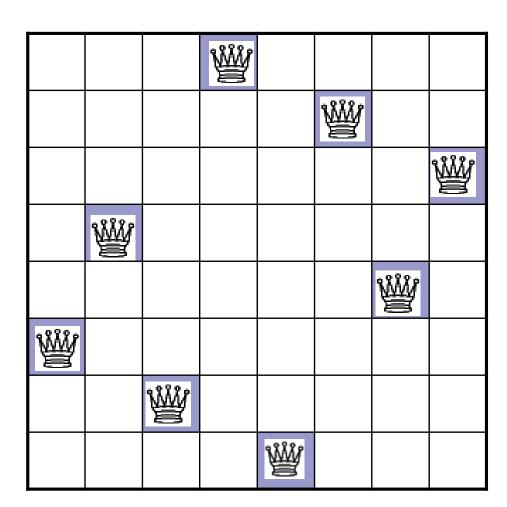


El problema de las ocho reinas

 Las restricciones explícitas las dan los conjuntos

$$S_i = \{1,2,3,4,5,6,7,8\}, 1 \le i \le n.$$

- Espacio solución con 8⁸ = 2²⁴ = 16M tuplas.
- Restricciones implícitas: ningún par de x_i puedan ser iguales (todas las reinas deben estar en columnas diferentes). Ningún par de reinas pueden estar en la misma diagonal.
- La primera de estas dos restricciones implica que todas las soluciones son permutaciones de (1,2,3,4,5,6,7,8).
- Esto lleva a reducir el tamaño del espacio solución de 8º tuplas a 8! = 40,320
- Una posible solución del problema es la (4,6,8,2,7,1,3,5).

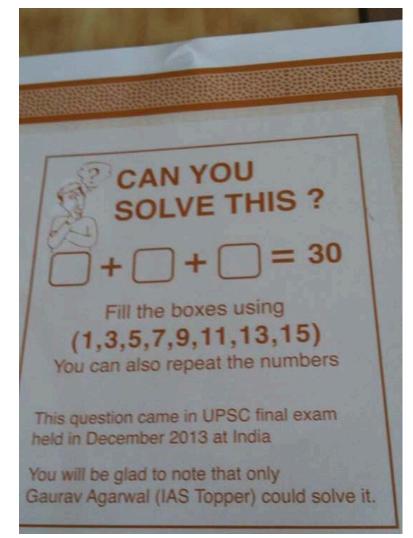






Problema de la suma de subconjuntos

- Dados n+1 números positivos: w_i, 1≤i≤n, y uno mas M
- Encontrar todos los subconjuntos de números w_i cuya suma valga M.
- Sea n = 4, M = 31 $(w_1, w_2, w_3, w_4) = (11, 13, 24, 7)$
- Los subconjuntos buscados son (11, 13, 7) y (24,7)
- No siempre es "tan" fácil







Ver mis oportuni

Continúa donde le

Top de tu grado.

RocioZabalo

Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.





Problema de la suma de subconjuntos

- Para representar la solución podríamos notar el vector solución con los índices de los correspondientes w_i.
- Las dos soluciones se describen por los vectores (1,2,4) y (3,4).
- Todas las soluciones son k-tuplas $(x_1, x_2,...,x_k)$, $1 \le k \le n$, y soluciones diferentes pueden tener tamaños de tupla diferentes.
- Restricciones explícitas: $x_i \in \{j: j \text{ es entero } y \text{ } 1 \leq j \leq n\}$
- Restricciones implícitas: que no haya dos iguales y que la suma de los correspondientes w_i sea M.
- Como, por ejemplo (1,2,4) y (1,4,2) representan el mismo subconjunto, otra restricción implícita que hay que imponer es que $x_i < x_{i+1}$, para $1 \le i < n$.





Problema de la suma de subconjuntos

- Puede haber diferentes formas de formular un problema de modo que todas las soluciones sean tuplas satisfaciendo algunas restricciones.
- Otra formulación del problema:
 - □ Cada subconjunto solución se representa por una n-tupla $(x_1,...,x_n)$ tal que $x_i \in \{0,1\}$, $1 \le i < n$, $y(x_i) = 0$ si w_i no se elige $y(x_i) = 1$ si se elige w_i .
 - \square Las soluciones del anterior caso son (1,1,0,1) y (0,0,1,1).
 - □ Esta formulación expresa todas las soluciones usando un tamaño de tupla fijo.
- Se puede comprobar que para estas dos formulaciones, el espacio solución consiste en ambos casos de 2⁴ tuplas distintas.





Espacios de soluciones

- Los algoritmos backtracking determinan las soluciones del problema buscando en el espacio de soluciones del caso considerado sistemáticamente.
- Esta búsqueda se facilita usando una organización en árbol para el espacio solución.
- Para un espacio solución dado, pueden caber muchas organizaciones en árbol.
- Los siguientes ejemplos examinan algunas de las formas posibles para estas organizaciones.





Ejemplo de espacio de soluciones

- La generalización del problema de las 8 reinas es el de las N reinas: Colocar N reinas en un tablero NxN de modo que no haya dos que se ataquen
- Ahora el espacio de soluciones consiste en las N! permutaciones de la N-tupla (1,2,...,N).
- La generalización nos sirve a efectos didácticos para poder hablar del problema de las 4 reinas
- La siguiente figura muestra una posible organización de las soluciones del problema de las 4 reinas en forma de árbol.
- A un árbol como ese se le llama **Árbol de** (búsqueda de soluciones) **Permutación.**





El subárbol de la izquierda contiene todas las soluciones con $x_1 = 1$ y $x_2 = 2$, etc. El espacio de soluciones esta definido por todos los caminos desde el nodo raíz a un nodo hoja. Hay 4! = 24 nodos hoja en la figura.



Suma de subconjuntos: árbol

- Vimos dos posibles formulaciones del espacio solución del problema de la suma de subconjuntos.
 - La primera corresponde a la formulación por el tamaño de la tupla
 - La segunda considera un tamaño de tupla fijo
- Con ambas formulaciones, tanto en este problema como en cualquier otro, el número de soluciones tiene que ser el mismo

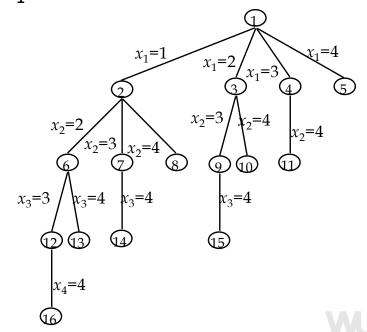




Suma de subconjuntos: árbol 1*

- Las aristas se etiquetan de modo que una desde el nivel de nodos i hasta el i+1 representa un valor para x_i.
- En cada nodo, el espacio solución se particiona en espacios subsolución.
- Los posibles caminos son (), que corresponde al camino vacío desde la raíz a si misma, (1), (12), (123), (1234), (124), (134), (2), (23), etc.

Así, el subarbol de la izquierda define todos los subconjuntos conteniendo w₁, el siguiente todos los que contienen w₂ pero no w₁, etc.



* (Horowitz-Sahni, 1978)



Suma de subconjuntos: árbol 2

- Una arista del nivel i al i+1 se etiqueta con el valor de x_i (0 o 1).
- Todos los caminos desde la raíz a las hojas definen el espacio solución.
- El subarbol de la izquierda define todos los subconjuntos conteniendo w₁, mientras que el de la derecha define todos los subconjuntos que no contienen w₁, etc.
- Consideramos el caso de n = 4
- Hay 2⁴ nodos hoja, que representan 16 posibles tuplas.



Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.



Terminología

- La organización en árbol del espacio solución se llama árbol de estados y en él cada nodo define un estado del problema.
- Todos los caminos desde la raíz a otros nodos definen el espacio de estados del problema.
- **Estados solución** son aquellos estados del problema S para los que el camino desde la raíz a S define una n-tupla en el espacio solución.
 - □ En el primero de los árboles anteriores, todos los nodos son estados solución, mientras que en el segundo solo los nodos hoja son estados solución.
- Estados respuesta son aquellos estados solución S para los que el camino desde la raíz hasta S define una tupla que es miembro del conjunto de soluciones (es decir satisfacen las restricciones implícitas) del problema.





Terminología

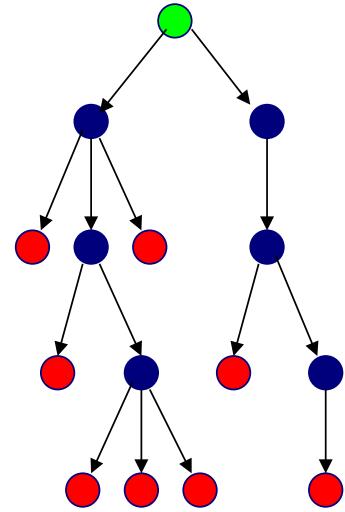
- Las organizaciones en árbol del problema de las 8 reinas se llaman **árboles estáticos**, porque son independientes del caso del problema que se vaya a resolver.
- En algunos problemas es ventajoso usar diferentes organizaciones de árbol para diferentes casos del problema.
- Entonces la organización en árbol se determina dinámicamente como el espacio solución que esta siendo buscado.
- Las organizaciones en árbol que son dependientes del caso concreto del problema a resolver se llaman árboles dinámicos.





Generación de estados de un problema

Cuando se ha concebido un árbol de estados para algún problema, podemos resolver este problema generando sistemáticamente sus estados, determinando cuales de estos son estados solución, y finalmente determinando que estados solución son estados respuesta.





405416_arts_esceniq

Top de tu grado.

RocioZabalo

Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.





Generación de estados de un problema

- Hay dos formas diferentes de generar los estados del problema.
- Las dos comienzan con el nodo raíz y generan otros nodos.
- Un nodo que ha sido generado, pero para el que no se han generado aun todos sus nodos hijos, se llama un nodo vivo.
- El nodo vivo cuyos hijos están siendo generados en ese momento se llama un **E-nodo** (nodo en expansión).
- Un **nodo explorado** es un nodo generado que o no se va a expandir o que todos sus hijos ya han sido generados.
- En ambos métodos de generar estados del problema tendremos una lista de nodos vivos.





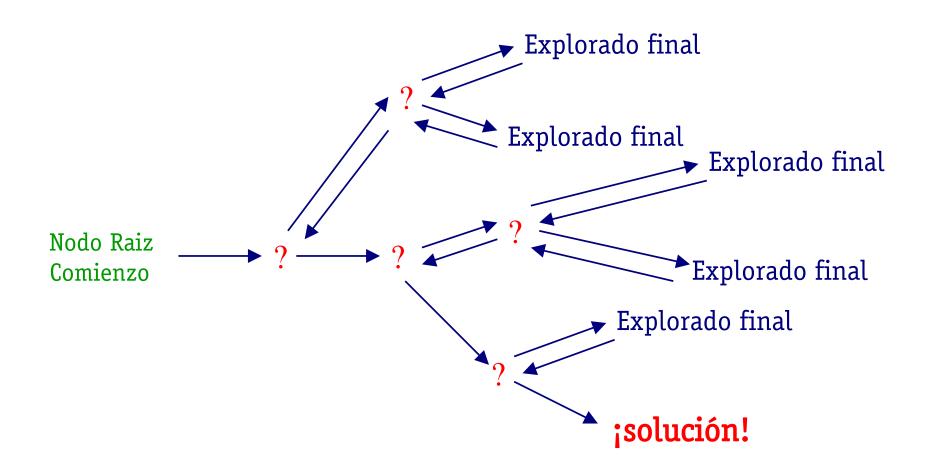
Generación de estados de un problema

- En el primer método, tan pronto como un nuevo hijo C, del E-nodo en curso R, ha sido generado, este hijo se convierte en un nuevo E-nodo.
- R se convertirá de nuevo en E-nodo cuando el subarbol C haya sido explorado completamente.
- Esto corresponde a una generación primero en profundidad de los estados del problema.
- Adicionalmente se usan funciones de acotación para eliminar nodos vivos sin tener que generar todos sus nodos hijos.
- Esto habrá de hacerse cuidadosamente para que a la conclusión del proceso al menos se haya generado un nodo respuesta, o se hayan generado todos los nodos respuesta si el problema requiriera encontrar todas las soluciones.
- A esta forma de generación (exploración) se le llama Backtracking





Como procede el Backtracking







Generación de estados de un problema

- En el segundo método, el E-nodo permanece como E-nodo hasta que se hace nodo explorado
- Como en el otro método, también se usan funciones de acotación para detener la exploración en un sub-árbol.
- El método se adapta muy bien a la resolución de problemas de optimización combinatoria (espacio de soluciones discreto): Problema de la Mochila, soluciones enteras, etc.
- En este método, la construcción de las funciones de acotación es (casi) mas importante que los mecanismos de exploración en si mismos.
- A esta segunda forma de generación (exploración) se le llama Branch and Bound (Ramificación y Poda).





Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.





Algoritmo Backtracking

- Podemos presentar una formulación general, aunque precisa, del proceso de backtracking.
- Supondremos que hay que encontrar todos los nodos respuesta, y no solo uno.
- Sea (x₁, x₂,..., x_i) un camino desde la raíz hasta un nodo en el árbol de estados.
- Sea $T(x_1, x_2,..., x_i)$ el conjunto de todos los posibles valores de x_{i+1} tales $(x_1, x_2,..., x_{i+1})$ es también un camino hacia un estado del problema.
- Suponemos la existencia de funciones de acotación B_{i+1} (expresadas como predicados) tales que B_{i+1} ($x_1, x_2, ..., x_{i+1}$) es falsa para un camino ($x_1, x_2, ..., x_{i+1}$) desde el nodo raíz hasta un estado del problema si el camino no puede extenderse para alcanzar un nodo respuesta.
- Así los candidatos para la posición i+1 del vector solución X(1..n) son aquellos valores que son generados por T y satisfacen B_{i+1} .





Algoritmo Backtracking

■ El Procedimiento Backtrack, representa el esquema general backtracking haciendo uso de T y B_{i+1} .

Procedimiento BACKTRACK(n)

```
{Todas las soluciones se generan en X(1..n) y se imprimen tan pronto como se encuentran. T(X(1),...,X(k-1)) da todos los posibles valores de X(k) dado que habíamos escogido X(1),..., X(k-1). El predicado B_k(X(1),...,X(k)) determina los elementos X(k) que satisfacen las restricciones implícitas}
```

```
Begin k = 1 While k > 0 do If queda algún X(k) no probado tal que X(k) \in T(X(1),..., X(k-1)) \text{ and } B_k (X(1),..., X(k)) = \text{true} Then if (X(1),..., X(k)) es un camino hacia un nodo respuesta Then print (X(1),..., X(k)) k = k+1
```

WUOLAH

else k = k-1



Algoritmo Backtracking recursivo

■ El siguiente algoritmo, Rbacktrack, presenta una formulación recursiva del método, ya que como backtracking básicamente es un recorrido en postorden, es natural describirlo así,

Procedimiento RBACTRACK(K)

```
{Se supone que los primeros k-1 valores X(1),...,X(k-1) del vector solución X(1..n) han sido asignados}

Begin for cada X(k) tal que X(k) \in T(X(1),...,X(k-1)) and B(X(1),...,X(k)) = true do If (X(1),...,X(k)) es un camino hacia un nodo respuesta Then print (X(1),...,X(k)) RBACKTRACK(K+1)
```





Eficiencia de backtracking

- La eficiencia de los algoritmos backtracking depende básicamente de cuatro factores:
 - \square el tiempo necesario para generar el siguiente X(k),
 - \square el número de X(k) que satisfagan las restricciones explícitas,
 - □ el tiempo para las funciones de acotación Bi, y
 - \square el número de X(k) que satisfagan las B_i para todo i.
- Las funciones de acotación se entienden buenas si reducen considerablemente el número de nodos que generan.
- Las buenas funciones de acotación consumen mucho tiempo en evaluaciones, por lo que hay que buscar un equilibrio entre el tiempo global de computación, y la reducción del número de nodos generados.





Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.





Eficiencia de backtracking

- De los 4 factores que determinan el tiempo requerido por un algoritmo backtracking, solo la cuarta, el número de nodos generados, varía de un caso a otro.
- Un algoritmo backtracking en un caso podría generar solo O(n) nodos, mientras que en otro (relativamente parecido) podría generar casi todos los nodos del árbol de espacio de estados.
- Si el número de nodos en el espacio solución es 2ⁿ o n!, el tiempo del peor caso para el algoritmo backtracking será generalmente O(p(n)2ⁿ) u O(q(n)n!) respectivamente, con p y q polinomios en n.
- La importancia del backtracking reside en su capacidad para resolver casos con grandes valores de n en muy poco tiempo.
- La dificultad está en predecir la conducta del algoritmo backtrack en el caso que deseemos resolver.





Solución para las 8 reinas

- Generalizamos el problema para considerar un tablero nxn y encontrar todas las formas de colocar n reinas que no se ataquen.
- Podemos tomar (x₁,...,x_n) representando una solución si x_i es la columna de la i-esima fila en la que la reina i esta colocada.
- Los x_i's serán todos distintos ya que no puede haber dos reinas en la misma columna.
- ¿Como comprobar que dos reinas no estén en la misma diagonal?

			(1,4)				
(2,1)		(2,3)					
(2,1)		(2,3)					
	(3,2)						
(4,1)		(4,3)					
			(5,4)				(5,8)
(6,1)				(6,5)		(6,7)	
	(7,2)				(7,6)		
		(8,3)		(8,5)		(8,7)	





Solución para las 8 reinas

- Si las casillas del tablero se numeran como una matriz A(1..n,1..n), cada elemento en la misma diagonal que vaya de la parte superior izquierda a la inferior derecha, tiene el mismo valor "fila-columna".
- También, cualquier elemento en la misma diagonal que vaya de la parte superior derecha a la inferior izquierda, tiene el mismo valor "fila+columna".
- Si dos reinas están colocadas en las posiciones (i,j) y (k,l), estarán en la misma diagonal solo si,

$$i - j = k - l \circ i + j = k + l$$

La primera ecuación implica que

$$j - l = i - k$$

La segunda que

$$j - l = k - i$$

Así, dos reinas están en la misma diagonal si y solo si

$$|j-l| = |i-k|$$





Solución para las 8 reinas

■ El procedimiento COLOCA(k) devuelve verdad si la k-ésima reina puede colocarse en el valor actual de X(k). Testea si X(k) es distinto de todos los valores previos X(1),...,X(k-1), y si hay alguna otra reina en la misma diagonal. Su tiempo de ejecución de O(k-1).

```
Procedimiento COLOCA(K)
```

```
{X es un array cuyos k primeros valores han sido ya asignados. ABS(r) da el valor absoluto de r}
```

```
Begin
```

```
For i:=1 to k do
    If X(i) = X(k) or ABS(X(i)-X(k)) = ABS(i-k)
    Then return (false)
    Return (true)
end
```





Ver mis oportuni

Continúa donde le

405416_arts_esceniq

Top de tu grado.

7CR

RocioZabalo

ponywaszas

Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.





Solución para las 8 reinas

Procedimiento NREINAS(N)

{Usando backtracking este procedimiento imprime todos los posibles emplazamientos de n reinas en un tablero nxn sin que se ataquen}

Begin

X(1) := 0, k := 1

While k > 0 do

X(k) := X(k) + 1

While $X(k) \le n$ and not COLOCA (k) do

X(k) := X(k) + 1

If $X(k) \leq n$

Then if k = n

Then print (X)

Else k := k + 1; X(k) := 0

Else k := k - 1

end

{k es la fila actual}
{hacer para todas las filas}
{mover a la siguiente columna}
{puede moverse esta reina?}

{Se encontró una posición}

{Es una solución completa?}

{Ir a la siguiente fila}

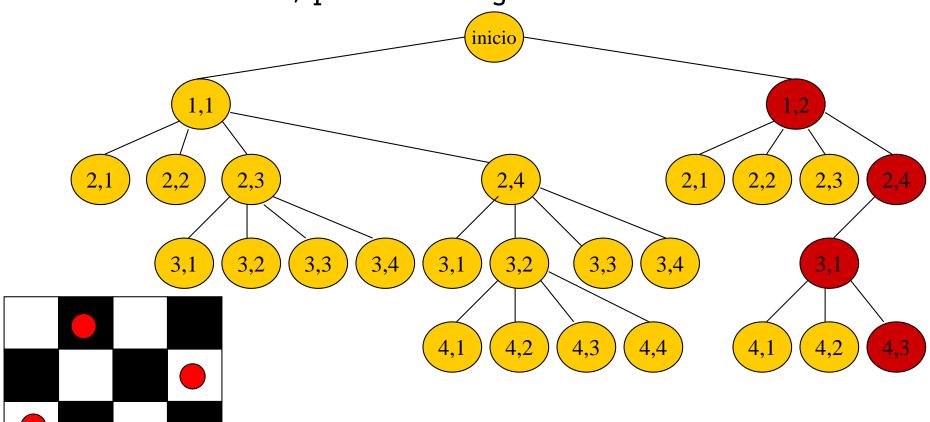
{Backtrack}





Ejemplo... para el caso de 4-reinas

La aplicación de Bactracking en este caso, comenzando colocando la reina 1 en la fila 1, produce el siguiente árbol



Se comprobaron 27 nodos... Sin backtracking se hubieran comprobado 155 nodos



Solución para la suma de subconjuntos

- Tenemos n números positivos distintos (usualmente llamados pesos) y queremos encontrar todas las combinaciones de estos números que sumen M.
- Los anteriores ejemplos nos 4 mostraron como podríamos formular este problema usando tamaños de las tuplas fijos o variables.
- Consideraremos una solución backtracking usando la estrategia del tamaño fijo de las tuplas.
- En este caso el elemento X(i) del vector solución es uno o cero, dependiendo de si el peso W(i) esta incluido o no.





Solución para la suma de subconjuntos

- Generación de los hijos de cualquier nodo en el árbol:
- Para un nodo en el nivel i, el hijo de la izquierda corresponde a X(i) = 1, y el de la derecha a X(i) = 0.
- Una posible elección de funciones de acotación es $B_k(X(1), ..., X(k)) = true si y solo si,$

$$\sum_{1...k} W(i)X(i) + \sum_{k+1...n} W(i) \ge M$$

Claramente X(1), ..., X(k) no pueden conducir a un nodo respuesta si no se verifica esta condición.





18 DES

Ver mis oportuni

Continúa donde le

Top de tu grado.

RocioZabalo

Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.





Solución para la suma de subconjuntos

- Las funciones de acotación pueden fortalecerse si suponemos los W(i)'s en orden creciente.
- En este caso, X(1),...,X(k) no pueden llevar a un nodo respuesta si

$$\sum_{1...k} W(i)X(i) + W(k+1) > M$$

■ Por tanto las funciones de acotación que usaremos serán las definidas de la siguiente forma: $B_k(X(1), ..., X(k))$ es true si y solo si

$$\sum_{1..k} W(i)X(i) + \sum_{k+1..n} W(i) \ge M$$

У

$$\sum_{1..k} W(i)X(i) + W(k+1) \leq M$$





Solución para la suma de subconjuntos

- Ya que nuestro algoritmo no hará uso de B_n , no necesitamos preocuparnos por la posible aparición de W(n+1) en esta función.
- Aunque hasta aquí hemos especificado todo lo que es necesario para usar cualquiera de los esquemas Backtracking, resultaría un algoritmo mas simple si diseñamos a la medida del problema que estemos tratando cualquiera de esos esquemas.
- Esta simplificación resulta de la comprobación de que si X(k) = 1, entonces

$$\sum_{1...k} W(i)X(i) + \sum_{k+1...n} W(i) > M$$





Esquema de algoritmo recursivo

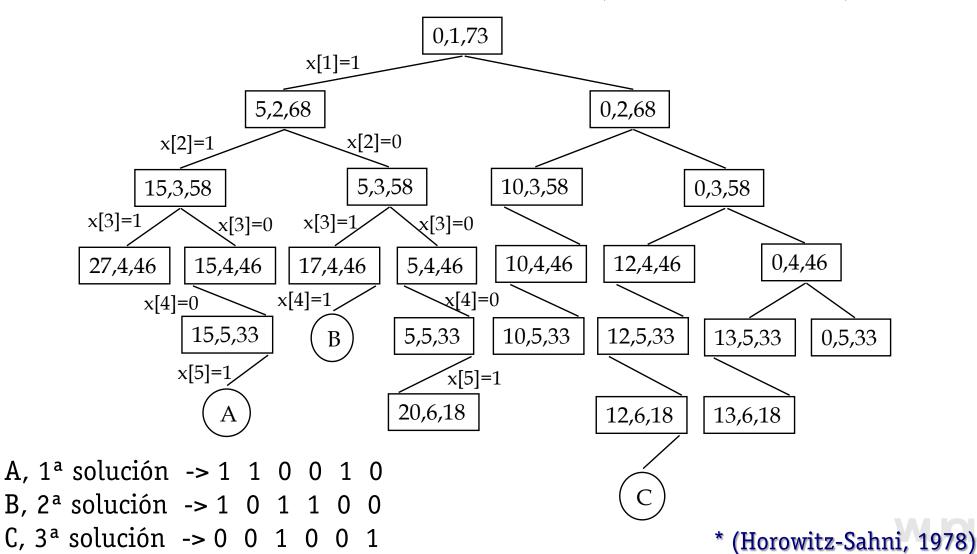
```
Procedimiento SUMASUB (s,k,r)
    {Los valores de X(j), 1 \le j < k, ya han sido determinados. s = \sum_{j=k-1}^{n} W(j)X(j) y r
   =\sum_{k=n} W(j). Los W(j) están en orden creciente. Se supone que W(1) \leq M y que
   \sum_{i=1}^{n} W(i) \geq M
Begin
{Generación del hijo izquierdo. Nótese que s+W(k) \le M ya que B_{k-1} = true}
          X(k) = 1
          If s + W(k) = M
{4}
{5}
          Then For i = 1 to k print X(j)
          Else
{7}
                   If s + W(k) + W(k+1) \leq M
                   Then SUMASUB(s + W(k), k+1, r-W(k))
{Generación del hijo derecho y evaluación de B k}
           If s + r - W(k) \ge M and s + W(k+1) \le M
           Then X(k) = 0
           SUMASUB(S, K+1, R-w(K))
end
```





Ejemplo*

Como trabaja SUMASUB para el caso en que: W = (5, 10, 12, 13, 15, 18) y M = 30.





Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.





Ciclos hamiltonianos

- Sea G = (V, E) un grafo conexo con n vértices. Un ciclo Hamiltoniano es un camino circular a lo largo de los n vértices de G que visita cada vértice de G una vez y vuelve al vértice de partida, que naturalmente se visita dos veces.
- Estamos interesados en construir un algoritmo backtracking que determine todos los ciclos Hamiltonianos de G, que puede ser dirigido o no.
- El vector backtracking solución $(x_1, ..., x_n)$ se define de modo que x_i represente el i-ésimo vértice visitado en el ciclo propuesto.





Ciclos hamiltonianos

- Todo lo que se necesita es determinar como calcular el conjunto de posibles vértices para x_k si ya hemos elegido $x_1, ..., x_{k-1}$.
- Si k = 1, entonces X(1) puede ser cualquiera de los n vértices.
- Para evitar imprimir el mismo ciclo n veces, exigimos que X(1) = 1.
- Si 1 < k < n, entonces X(k) puede ser cualquier vértice v que sea distinto de X(1), X(2),...,X(k-1) que este conectado por una arista a X(k-1).
- X(n) solo puede ser el único vértice restante y debe estar conectado a X(n-1) y a X(1).



Ciclos hamiltonianos

```
Algoritmo Hamiltoniano (k)
while
    x[k] = SiguienteValor(k)
    if (x[k] = 0) then return
    if (k = N) then print solucion
    else Hamiltoniano (k+1)
endWhile
```

Utilizando este algoritmo podemos particularizar el esquema backtracking recursivo que vimos para encontrar todos los ciclos hamiltonianos

```
Algoritmo SiguienteValor (k)
while

value = (x[k]+1) mod (N+1)

if (value = 0) then return value

if (G[x[k-1],value])

for j = 1 to k-1 if x[j] = value then break

if (j=k) and (k < N or k = N and G[x[N],x[1]])

then return value
endWhile
```

