

Ayudantía Backtracking

Backtracking

Joaquín Peralta, Amelia González, Alexander Infanta, Elias Ayaach, Paula Grune

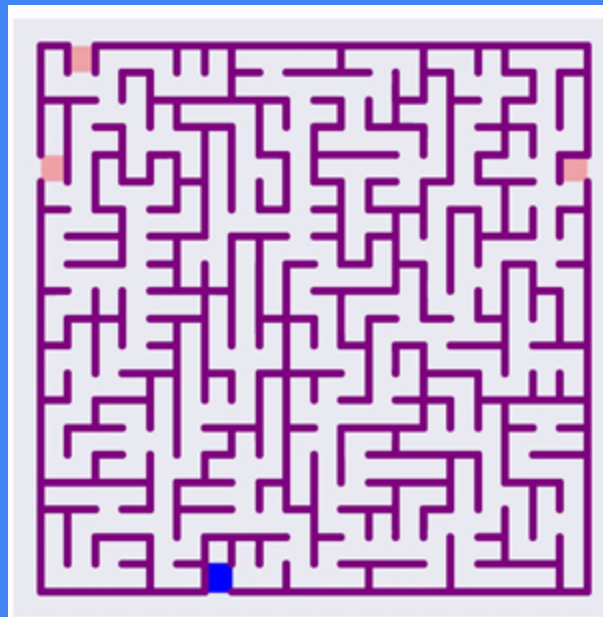
Motivación



Backtracking es como la vida: avanzas, la cagas, retrocedes y vuelves a empezar. Eso es aprender<3

Backtracking

6	2	3	7	1	8	9	4	5
4	5	9	2	3	6	1	8	7
8	7	1		9		3	2	6
9		4		7		2	5	1
7	1	8	9	5	2	6	3	4
2	6					7	9	8
5				8	7	4	1	2
1				6		8	7	3
3						5	6	9



💡 Backtracking es **igual o más rápido** que la fuerza bruta

¿Por qué Backtracking?

- Problemas de decisión: Búsqueda de una solución factible.
- Problemas de optimización: Búsqueda de la mejor solución.
- Problema de enumeración: Búsqueda de todas las soluciones posibles.

Backtracking

- **X**: variables
- **D**: dominios
 - **D_x**: dominio de **X**
- **R**: restricción
 - c/u para un subconjunto de **X**

is solvable(X, D, R):

if $X = \emptyset$, *return true*

$x \leftarrow$ alguna variable de X

for $v \in D_x$:

if $x = v$ viola R , *continue*

$x \leftarrow v$

if is solvable($X - \{x\}, D, R$):

return true

$x \leftarrow \emptyset$

return false

Backtracking

¿Se puede mejorar este algoritmo?

Hay tres mejoras posibles (las cuales veremos la próxima semana):

- Podas
- Propagación
- Heurísticas

Poda

Técnica para reducir el espacio de búsqueda, eliminando ramas del árbol de decisiones que no pueden conducir a una solución válida

En otras palabras, estamos **podando** parte del conjunto de caminos* a soluciones posibles.

Por Factibilidad, Dominio y Consistencia, etc.

is solvable(X, D, R):

if $X = \emptyset$, *return true*

$x \leftarrow$ alguna variable de X

for $v \in D_x$:

if $x = v$ **no es válida**, *continue*

$x \leftarrow v$

if is solvable($X - \{x\}, D, R$):

return true

$x \leftarrow \emptyset$

return false

(*) Bajo la idea: Cada posible asignación genera un camino

Propagación

Cuando a una variable se le asigna un valor, se puede propagar esta información para luego poder **reducir el dominio** de valores de **otras variables**.

is solvable(X, D, R):

if $X = \emptyset$, *return true*

$x \leftarrow$ alguna variable de X

for $v \in D_x$:

if $x = v$ viola R , *continue*

$x \leftarrow v$, propagar

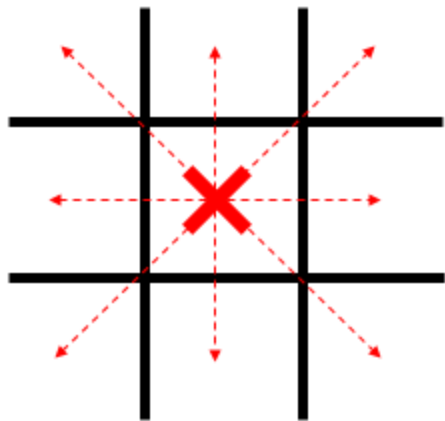
if is solvable($X - \{x\}, D, R$):

return true

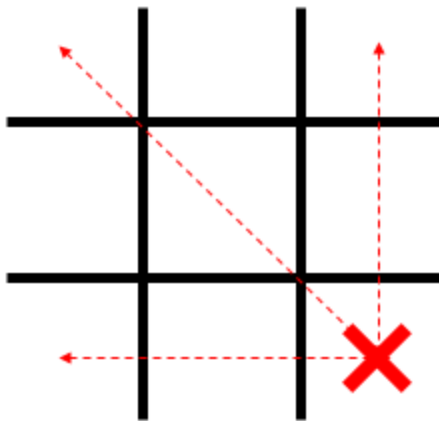
$x \leftarrow \emptyset$, propagar

return false

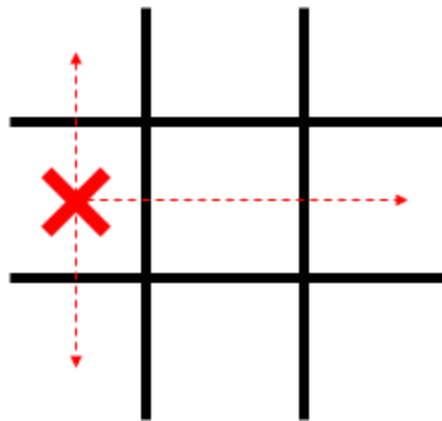
Heurística



4 ways to win the game



3 ways to win the game

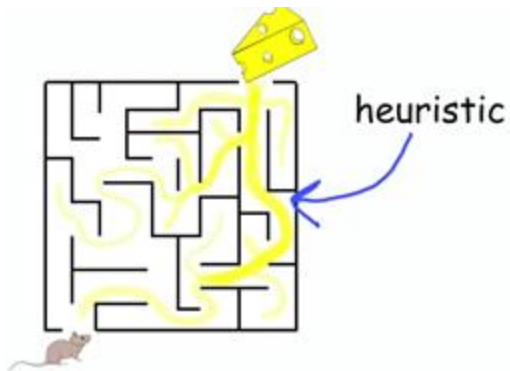


2 ways to win the game

<https://qiao.github.io/PathFinding.js/visual/>

Heurística

Una heurística es una aproximación al mejor criterio para abordar un problema.



is solvable(X, D, R):

if $X = \emptyset$, *return true*

$x \leftarrow$ la mejor variable de X

for $v \in D_x$, de mejor a peor:

if $x = v$ viola R , *continue*

$x \leftarrow v$

if is solvable($X - \{x\}, D, R$):

return true

$x \leftarrow \emptyset$

return false

} “lo mejor” respecto a lo que mi heurística determina

Heurística

Una heurística es una aproximación al mejor criterio para abordar un problema.

una mala heurística nos puede dejar así:



no solutions?

is solvable(X, D, R):

if $X = \emptyset$, *return true*

$x \leftarrow$ la mejor variable de X

for $v \in D_x$, de mejor a peor:

if $x = v$ viola R , *continue*

$x \leftarrow v$

if is solvable($X - \{x\}, D, R$):

return true

$x \leftarrow \emptyset$

return false

} “lo mejor” respecto a lo que mi heurística determina

Problema 1

Registros Académicos de una nueva universidad necesita un sistema para asignar salas a los cursos luego de la toma de ramos. Considere que $S = \{s_1, \dots, s_n\}$ y $C = \{c_1, \dots, c_m\}$ son los conjuntos de salas y cursos, respectivamente. Luego, para una sala s_i su capacidad es $a(s_i)$ y para un curso c_j , su cantidad de inscritos es $\ell(c_j)$, su módulo horario es $1 \leq t(c_j) \leq 9$ y su día de la semana es $1 \leq d(c_j) \leq 5$. Una asignación es válida si cada sala tiene a lo más un curso en todo momento, todo curso c_j es asignado a alguna sala s_i en su horario y $0,8 \leq \ell(c_j)/a(s_i) \leq 1$, debido a que los estudiantes no quieren estar en salas demasiado grandes para su curso.

- (a) Supongamos que todos los cursos tienen un solo módulo horario a la semana.
 - (i) [1 pto.] Defina variables, dominios y restricciones para el problema de asignar salas a cursos.
 - (ii) [3 ptos.] Proponga el pseudocódigo de un algoritmo de backtracking que retorne una asignación cursos válida o -1 si no existe.

(a) Supongamos que todos los cursos tienen un solo módulo horario a la semana.

(i) [1 pto.] Defina variables, dominios y restricciones para el problema de asignar salas a cursos.

Representaremos la asignación indicando en qué combinación sala-módulo-día se asigna un curso o si se deja vacío. Para esto, consideramos las variables

$$X = \{x_{t,d,i} \mid 1 \leq t \leq 9, 1 \leq d \leq 5, 1 \leq i \leq n\}$$

donde la variable $x_{t,d,i}$ indica qué curso se asigna en el módulo t del día d a la sala s_i . El dominio de estas variables es $\{0, 1, \dots, m\}$, i.e. el indicador del curso que se asigna, incluyendo 0 para el caso en que la sala se deja vacía en ese momento.

Las restricciones son (1) si $x_{t,d,i} = j$, entonces debe cumplirse que $d(c_j) = d$ y $t(c_j) = t$, (2) si $x_{t,d,i} = j$, entonces $0,8 \leq \ell(c_j)/a(s_i) \leq 1$. La restricción de cantidad de cursos por sala en un momento dado está implícita en que las variables toman un solo valor del rango de ids de cursos.

*No hay una única solución válida

- (ii) [3 ptos.] Proponga el pseudocódigo de un algoritmo de backtracking que retorne una asignación cursos válida o -1 si no existe.

input : Arreglo multidimensional M , módulo t , día d , sala i
output: ¿Es posible asignar cursos a partir de la celda indicada?

Backtrack(M, t, d, i):

```
1  if  $t, d, i$  fuera del rango :  
2      return true  
3  for  $j \in \{0, 1, \dots, m\}$  :  
4      if Check( $M, t, d, i, j$ ) :  
5           $M[t][d][i] \leftarrow j$   
6           $t', d', i' \leftarrow$  siguiente celda de  $M$   
7          if Backtrack( $M, t', d', i'$ ) :  
8              return true  
9  return false
```

input : Arreglo multidimensional M , módulo t , día d , sala i , curso j
output: ¿Se respetan las restricciones al asignar j en t, d, i ?

Check(M, t, d, i, j):

```
1  if  $0,8 > \ell(c_j)/a(s_i)$  OR  $\ell(c_j)/a(s_i) > 1$  :  
2      return false  
3  if  $t(c_j) \neq t$  OR  $d(c_j) \neq d$  :  
4      return false  
5  return true
```

output: Asignación o -1

Solver():

```
1   $M \leftarrow$  arreglo multidimensional de  $9 \times 5 \times n$  de ceros  
2  if Backtrack( $M, 1, 1, 1$ ) :  
3      return  $M$   
4  return  $-1$ 
```

[1 pto.] Explique cómo modificar su solución de (a) si cada curso c_j tiene una prioridad diferente $p(c_j)$ y se exige que primero se asignen los cursos con mayor prioridad. No requiere mostrar pseudocódigo.

Solución.

Al momento de iterar sobre los valores de j posibles, este **for** debe recorrer los valores de j según la prioridad. Basta con ordenar los índices de acuerdo a $p(c_j)$.

Problema 2

Considere el siguiente acertijo lógico: En cinco casas alineadas de izquierda a derecha, cada una de un color diferente, viven 5 personas de diferentes nacionalidades, cada una de las cuales prefiere un deporte diferente, una bebida diferente y una mascota diferente. Además, están dados los siguientes hechos:

- La persona inglesa vive en la casa roja.
- La persona española tiene el perro.
- La persona argentina vive en la primera casa de la izquierda.
- Se prefiere el fútbol en la casa amarilla.
- La persona que prefiere el basketball vive en la casa junto a la persona que tiene el gato.
- La persona argentina vive junto a la casa azul.
- La persona que prefiere el tenis bebe jugo de naranja.
- La persona chilena bebe té.
- La persona japonesa prefiere el judo.
- Se prefiere el fútbol en la casa junto a la casa donde se tiene el caballo.
- Se bebe café en la casa verde.
- La casa verde está inmediatamente a la derecha (a su derecha) de la casa blanca.

- a) (1 pt.) Modele este problema como un CSP (Problema de Satisfacción de Restricciones), para esto defina cuáles son las variables, dominios y restricciones del problema.

a) (1 pt.) Modele este problema como un CSP (Problema de Satisfacción de Restricciones), para esto defina cuáles son las variables, dominios y restricciones del problema.

■ **Variables:**

- $\text{Color}[i]$, $1 \leq i \leq 5$: color de la casa i .
- $\text{Nacionalidad}[i]$, $1 \leq i \leq 5$: nacionalidad de la persona que vive en la casa i .
- $\text{Deporte}[i]$, $1 \leq i \leq 5$: deporte preferido por la persona que vive en la casa i .
- $\text{Bebida}[i]$, $1 \leq i \leq 5$: bebida preferida por la persona que vive en la casa i .
- $\text{Mascota}[i]$, $1 \leq i \leq 5$: mascota preferida por la persona que vive en la casa i .

■ **Dominios:**

- $\text{Color} = \{\text{rojo}, \text{amarillo}, \text{azul}, \text{verde}, \text{blanco}\}$.
- $\text{Nacionalidad} = \{\text{chilena}, \text{peruana}, \text{inglesa}, \text{española}, \text{argentina}\}$.
- $\text{Deporte} = \{\text{fútbol}, \text{tenis}, \text{vóleibol}, \text{básquetbol}, \text{judo}\}$.
- $\text{Bebida} = \{\text{té}, \text{jugo de naranja}, \text{café}, \text{leche}, \text{agua}\}$.
- $\text{Mascota} = \{\text{perro}, \text{gato}, \text{conejo}, \text{caballo}, \text{cebra}\}$.

■ **Restricciones:** Las restricciones están dadas por los hechos definidos para el problema. Por ejemplo: “La persona inglesa vive en la casa roja”.

- b) (2 pts.) Proponga el pseudocódigo de la función `solveCSP()` para resolver el acertijo, indicando los parámetros adecuados para realizar la llamada inicial y obtener la solución al acertijo. Asuma que dispone de una función `verificaCSP()` adecuada para verificar las restricciones del problema.

- b) (2 pts.) Proponga el pseudocódigo de la función `solveCSP()` para resolver el acertijo, indicando los parámetros adecuados para realizar la llamada inicial y obtener la solución al acertijo. Asuma que dispone de una función `verificaCSP()` adecuada para verificar las restricciones del problema.

input : mascota buscada, bebida buscada

output: id de la casa donde está la mascota, id de la casa donde está la bebida

```
solveCSP(mascota, bebida):
```

```
0   for i = 0, 24 Casa[i] ← null
```

```
1   if Acertijo(Casa, 0):           // resuelve el acertijo, si tiene solución es true
```

```
2       for i = 0, 24               // recorre todas las variables
```

```
3           if Casa[i] = mascota
```

```
4               m ← (i + 1) / 5 // el ID de la casa correspondiente
```

```
5           if Casa[i] = bebida
```

```
6               b ← (i + 1) / 4 // el ID de la casa correspondiente
```

```
7       return b, m                // retorna los ID de la casa de la bebida y la mascota
```

```
8   return -1, -1                  // no había solución
```


- b) (2 pts.) Proponga el pseudocódigo de la función `solveCSP()` para resolver el acertijo, indicando los parámetros adecuados para realizar la llamada inicial y obtener la solución al acertijo. Asuma que dispone de una función `verificaCSP()` adecuada para verificar las restricciones del problema.

```
input : Arreglo Casa[0...24], // todas las variables de forma consecutiva en el arreglo Casa
        índice 0 i 25
output: true si hay solución, en Casa están los valores de cada variable

Acertijo(Casa, i):
1   if i = 25: return true
2   for v = Dominio(Casa[i])
3       if verificaCSP(Casa, i, v): // asume que tiene acceso a los hechos (restricciones)
4           Casa[i] ← v
5           if Acertijo(Casa, i + 1):
6               return true
7       Casa[i] ← null
8   return false
```

Un tablero de *kakuro square* es una grilla de $N \times M$ donde cada posición tiene una celda que puede ser rellenada por un número natural distinto de 0. Además, cada fila y columna tiene un número natural que indica el valor que debe tener la suma de los números correspondientes a la fila o columna. P.ej.:

	23	15	12
23			
17			
10			



	23	15	12
23	9	8	6
17	8	4	5
10	6	3	1

El problema de rellenar la grilla cumpliendo con las restricciones se puede resolver utilizando la estrategia de backtracking.

- Identifica las variables del problema y sus respectivos dominios.
- Explica con tus palabras cómo se podría revisar en tiempo $O(1)$ si la asignación de una variable rompe una restricción.

Respuesta:

a) Para resolver con backtracking este problema se debe asignar a cada celda un número, por lo que las variables son las celdas [1pto].

En teoría cada celda puede tener cualquier número natural, pero ya que no puede haber un dominio infinito para resolverlo con backtracking es necesario acotar los dominios. Un dominio acotado correcto puede ser los números del 1 al $\text{MAX}(\text{restriction})$. Ya que se sabe que nunca un número puede ser mayor al valor de su restricción [1pto].

b) Se puede mantener un contador por cada restricción del problema que cuente la suma actual de cada fila o columna. De esta manera se inmediatamente si rompo la restricción cuando el contador supera la restricción [2pts].