Ayudantía Backtracking

Backtracking

Motivación

Motivación

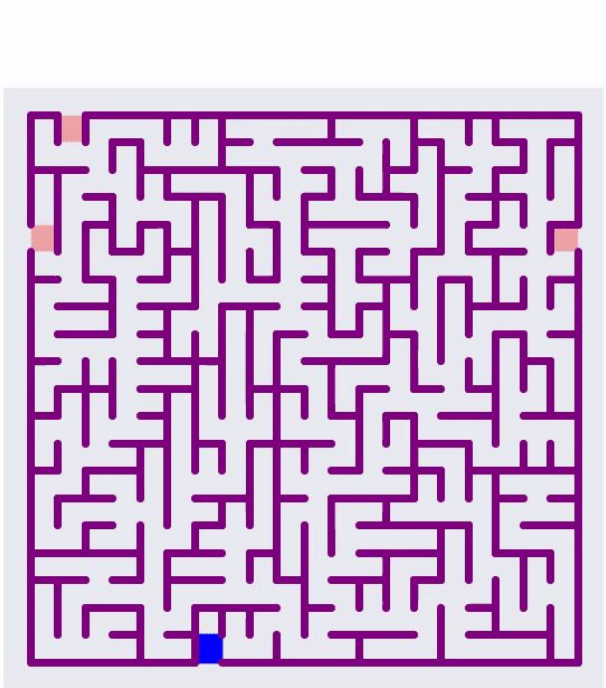
https://qiao.github.io/PathFinding.js/visual/

Recomendación

Película el efecto mariposa

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



¿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

```
is solvable(X, D, R):
    if X = ∅, return true
    x ← alguna variable de X
    for v ∈ D_x:
        if x = v viola R, continue
        x ← v
        if is solvable(X − {x}, D, R):
            return true
        x ← ∅
    return false
```

Backtracking

¿Se puede mejorar este algoritmo?

Hay tres mejoras posibles:

- Podas
- Propagación
- Heurísticas

Poda

Se deducen restricciones a partir de las restricciones o asignaciones anteriores que pueden ser agregadas al problema.

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

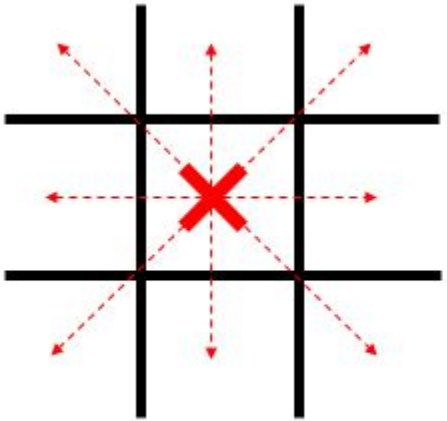
```
is solvable(X, D, R):
    if X = ∅, return true
    x ← alguna variable de X
    for v ∈ D_x:
        if x = v no es válida, continue
        x ← v
        if is solvable(X − {x}, D, R):
            return true
        x ← ∅
    return false
```

Propagación

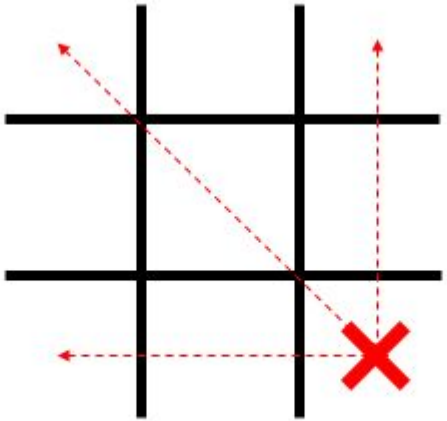
Cuando a una variable se le asigna un valor, se puede propagar esta información para luego poder reducir el dominio de valores de otras variables.

```
is solvable(X, D, R):
    if X = ∅, return true
    x ← alguna variable de X
    for v ∈ D_x:
        if x = v viola R, continue
        x ← v, propagar
        if is solvable(X − {x}, D, R):
            return true
        x ← ∅ , propagar
    return false
```

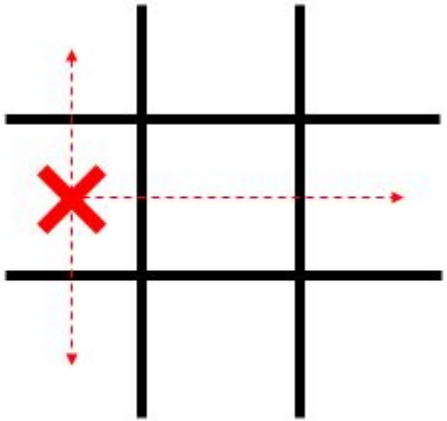
Heurística



4 ways to win the game



3 ways to win the game



2 ways to win the game

Heurística

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

```
is solvable(X, D, R):
    if X = ∅, return true
    x ← la mejor variable de X
    for v ∈ D_x, de mejor a peor:
        if x = v viola R, continue
        x ← v
        if is solvable(X − {x}, D, R):
            return true
        x ← ∅
    return false
```

Enunciado

Para asegurar la conectividad del trasporte en el extremo sur del país existen tramos en los cuales se utilizan barcazas para llevar vehículos (autos particulares y camiones) entre dos puntos que no tienen conectividad por tierra. La capacidad de la barcaza se define en función de los metros lineales de vehículos que puede acomodar (4 filas de vehículos de máximo 15 metros cada fila son 60 metros lineales de capacidad máxima) y el peso máximo total que puede transportar (por ejemplo 240.000 kilos de carga). Así una barcaza B se define como

(`B.n_filas`, `B.m_por_fila`, `B.max_carga`).

Los vehículos V que están a la espera de transporte están en una fila y tienen determinado su largo y peso (`V.largo`, `V.peso`) expresados en metros y kilogramos.

(a) [1 pto.] Identifique las Variables, Dominios y Restricciones del problema.

(b) [3 ptos.] Diseñe un algoritmo para definir qué vehículos de la fila transportar de modo de maximizar la cantidad de vehículos sin superar la capacidad de la barcaza (en metros lineales totales y la carga máxima de la misma). **No considere** la capacidad de cada fila de la barcaza, sino la **capacidad total**.

(c) [2 ptos.] Modifique su algoritmo anterior para que entregue en qué fila de la barcaza va cada vehículo a transportar, al maximizar la cantidad de vehículos sin superar la capacidad de la barcaza.