Ayudantía Backtracking

Backtracking

Berni Nazar, Paula Grune, Alex Infanta, Felipe Espinoza

MATERIAL DE APOYO

1. Cheatsheet C (notion resumen)
2. Ejercicios de práctica C
3. Cápsulas de semestres pasados

Dónde encuentro esto?

Links en ReadMe carpeta “Ayudantías” del repo

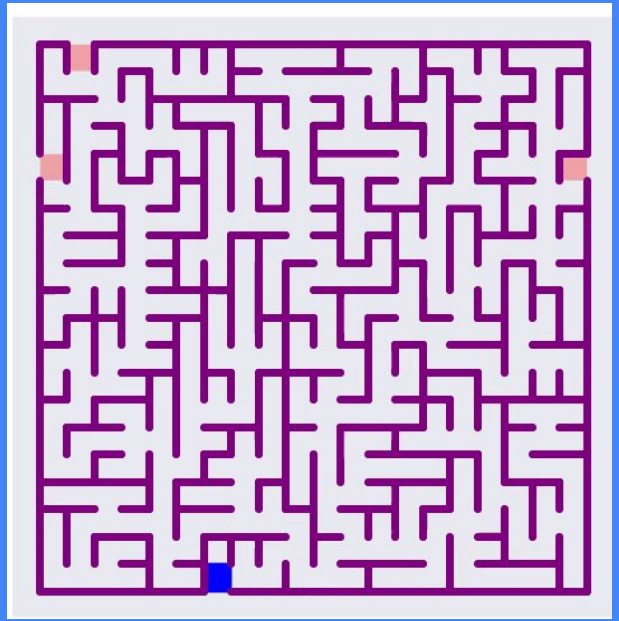
Motivación

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

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

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
  - **Dx**: dominio de **X**
- **R**: restricción
  - c/u para un subconjunto de **X**

```
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

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 = ∅, 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
```

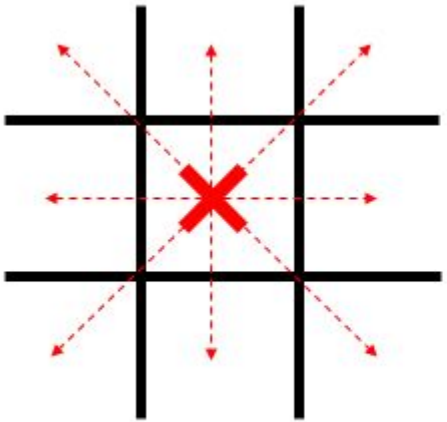
(*) Bajo la idea: Cada posible asignación genera un camino

Propagación

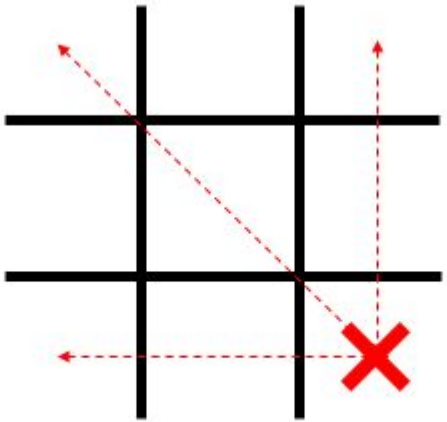
Cuando a una variable se le asigna un valor, se puede propagar esta información para luego poder **reducir el dominio** de valores **de otras variables**.

```
is solvable(X, D, R):
    if X = ∅, 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
```

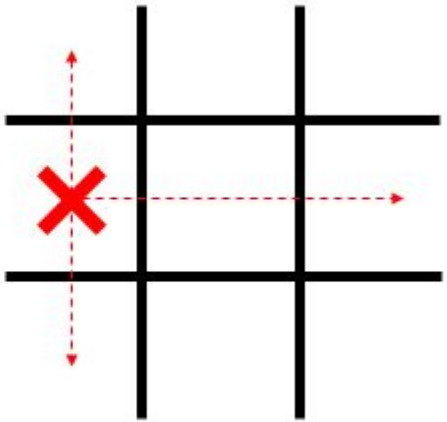
Heurística



4 ways to win the game



3 ways to win the game

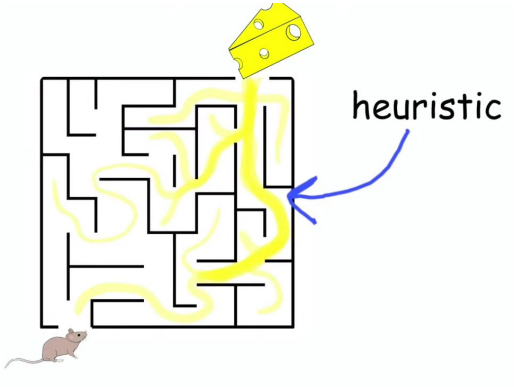


2 ways to win the game

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

Heurística

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



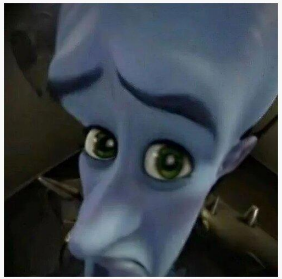

```
is solvable(X, D, R):
    if X = ∅, return true
    x ← la mejor variable de X
    for v ∈ D_x, de mejor a peor:
        if x = v viola R, continue
        x ← v
        if is solvable(X − {x}, D, R):
            return true
        x ← ∅
    return false
```

} "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 = ∅, 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
```

} "lo mejor" respecto a lo que mi heurística determina

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.

Solución

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

Recordemos:

1. ¿Qué debemos modificar? (variables)

   Filas de la barcaza! (incluir largo y peso).

1. ¿Sobre qué iteramos para encontrar esta configuración? (dominios)

   Vehículos disponibles (y sus atributos).

1. Al colocar un vehículo, qué **NO** debe pasar? (restricciones)

   Pasarse de la carga de la barcaza o el largo máx. de la fila.

```
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
```

(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**.

```
IsSolvable(X, D, R, M):
    if X.met_utilizados ≤ met_barcaza && X.p_utilizado ≤ p_max:
        if X.n_vehiculos ≥ M.n_vehiculos:
            M = X
        return True

    for v in D:
        agregar v a X
        if IsSolvable(X, D\{v}, R, M):
            marcar X como solución
        quitar vehículo de X
    return False
```

Caso Base

(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.

```
IsSolvable(X, D, R, M, F):
    if F > 4 return False
    if X.metros_utilizados[F] ≤ B.m_por_fila && X.p_utilizado ≤ p_max:
        if X.n_vehiculos ≥ M.n_vehiculos:
            M = X
        return True

    for v in D:
        agregar v a X en la fila F
        if IsSolvable(X, D\{v}, R, M, F):
            marcar X como solución
        else if IsSolvable(X, D/{v}, R, M, F + 1):
            marcar X como solución
        quitar vehículo de X
    return False
```

Enunciado 2

Registros Académicos de una nueva universidad necesita un sistema para asignar salas a los cursos luego de la toma de ramos. Considere que $S = \{s_1, \ldots, s_n\}$ y $C = \{c_1, \ldots, c_m\}$ 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 $\{0, 1, \ldots, m\}$, 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 ∈ {0, 1, ..., m} :
4        if Check(M, t, d, i, j) :
5            M[t][d][i] ← j
6            t', d', i' ← 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 > ℓ(c_j)/a(s_i) OR ℓ(c_j)/a(s_i) > 1 :
2        return false
3    if t(c_j) ≠ t OR d(c_j) ≠ d :
4        return false
5    return true
```

output: Asignación o -1

```
Solver():
1    M ← arreglo multidimensional de 9 × 5 × n de ceros
2    if Backtrack(M, 1, 1, 1) :
3        return M
4    return −1
```

(b) [1 pto.] Explique cómo modificar su solución de (a) para encontrar una asignación válida si es que los cursos pueden tener más de 1 módulo horario a la semana. No requiere mostrar pseudocódigo.

Solución.

Si consideramos que la sala debe ser la misma, existen dos opciones: (1) agregar una restricción adicional para verificar en `Check` o (2) propagar obligando que cualquier otra aparición del mismo curso solo pueda tomar un valor de sala.

Si consideramos que no hay restricción de sala, se puede extender el mismo algoritmo propuesto de manera que en lugar de m cursos, esta cantidad se aumenta viendo cada curso-módulo-día como un curso diferente. Luego, el algoritmo propuesto resuelve este problema sin necesitar más cambios.

(c) [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 $p(c_j)$.