



Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

Clase 21: Backtracking

Rodrigo Toro Icarte (rntoro@uc.cl)

IIC1103 Introducción a la Programación - Sección 5

08 de Junio, 2015

Clases pasadas



Clases pasadas

Definición

Recursión es una estrategia para solucionar problemas llamando a una función dentro de si misma.

Ventajas:

- Códigos más cortos.
- Códigos más legibles.

Clases pasadas

```
1 def factorial(n):  
2     if(n <= 1):  
3         return 1  
4     return n*factorial(n-1)
```

Estructura solución recursiva:

- Firma: Nombre y parámetros.
- Casos bases.
- Llamados recursivos.
- Formar solución a partir de subsoluciones.

Clases pasadas

¿Cuándo usar recursión?

- 1) Cuando un problema se puede dividir en subproblemas idénticos, pero más pequeños.
- 2) Para explorar el espacio en un problema de búsqueda.

Backtracking

Idea: En forma **ordenada** y **exhaustiva** exploraremos todas las combinaciones posibles para solucionar un problema.

Backtracking

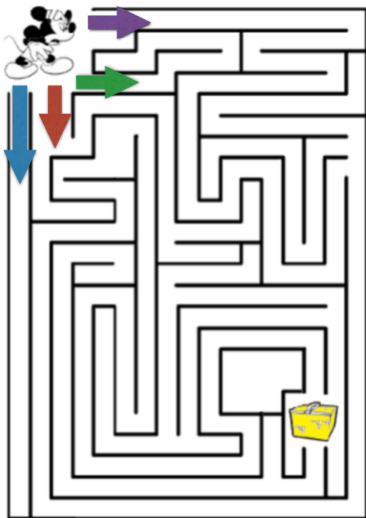
Idea: En forma **ordenada** y **exhaustiva** exploraremos todas las combinaciones posibles para solucionar un problema.

Ejemplo: Laberinto.

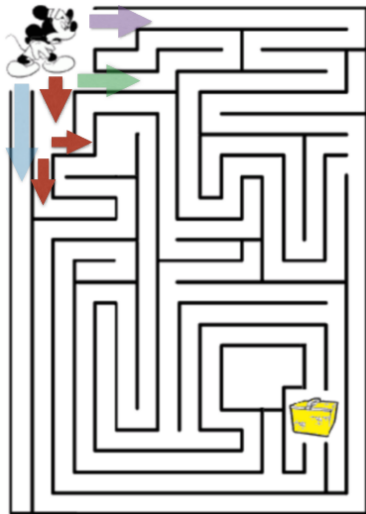
Backtracking



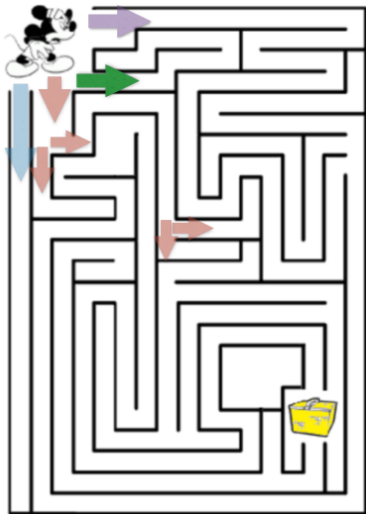
Backtracking



Backtracking



Backtracking



Backtracking



Backtracking

Ejemplo: Cree un programa que muestre todos los string de largo 3 que se pueden formar a partir de las letras "a", "b" y "c".

Backtracking

Ejemplo: Cree un programa que muestre todos los string de largo 3 que se pueden formar a partir de las letras "a", "b" y "c".

```
1 def permutaciones(s):
2     if(len(s) == 3):
3         print(s)
4         return
5     for c in ['a', 'b', 'c']:
6         permutaciones(s + c)
7
8 permutaciones("")
```

Backtracking

Ejemplo: Cree un programa que muestre todos los string de largo 3 que se pueden formar a partir de las letras "a", "b" y "c".

```
1 def permutaciones(s):
2     if(len(s) == 3):
3         print(s)
4         return
5     for c in ['a', 'b', 'c']:
6         permutaciones(s + c)
7
8 permutaciones("")
```

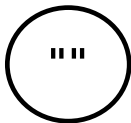
¿Cómo funciona esto?

Backtracking

... para ello analicemos las permutaciones de largo 2 de "a" y "b".

```
1 def permutaciones(s):
2     if(len(s) == 2):
3         print(s)
4         return
5     for c in ['a','b']:
6         permutaciones(s + c)
7
8 permutaciones("")
```

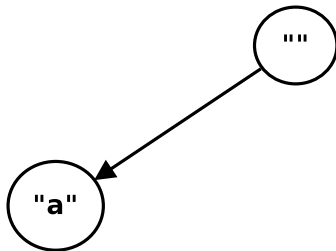

Backtracking



```
1 def permutaciones(s):
2     if(len(s) == 2):
3         print(s)
4         return
5     for c in ['a', 'b']:
6         permutaciones(s + c)
7
8 permutaciones("")
```

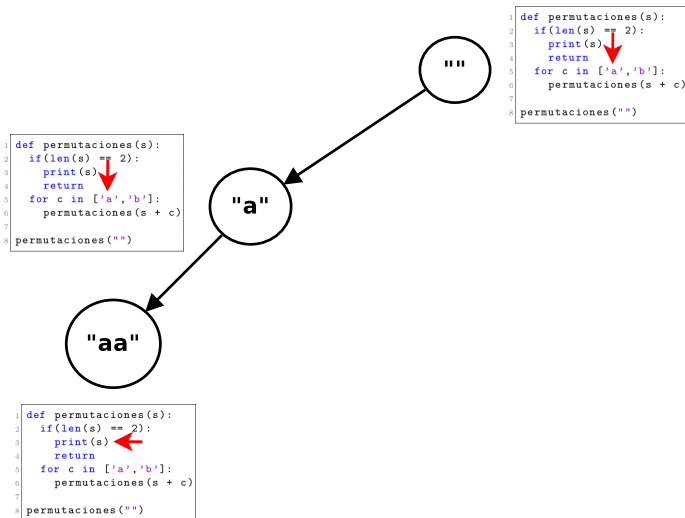
Backtracking

```
1 def permutaciones(s):  
2     if(len(s) == 2):  
3         print(s) ↓  
4         return  
5     for c in ['a', 'b']:  
6         permutaciones(s + c)  
7  
8 permutaciones("")
```



```
1 def permutaciones(s):  
2     if(len(s) == 2):  
3         print(s) ↓  
4         return  
5     for c in ['a', 'b']:  
6         permutaciones(s + c)  
7  
8 permutaciones("")
```

Backtracking



Backtracking

```
1 def permutaciones(s):  
2     if(len(s) == 2):  
3         print(s)  
4         return  
5     for c in ['a', 'b']:  
6         permutaciones(s + c)  
7  
8 permutaciones("")
```

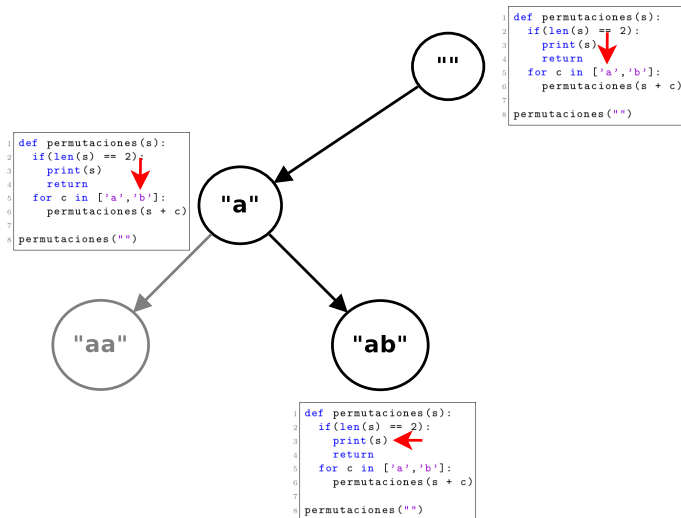
"aa"

"a"

""

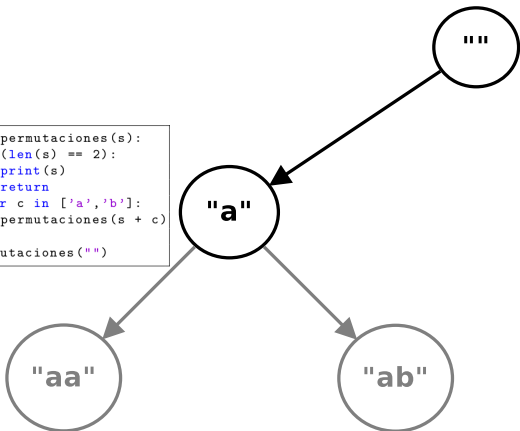

```
1 def permutaciones(s):  
2     if(len(s) == 2):  
3         print(s)  
4         return  
5     for c in ['a', 'b']:  
6         permutaciones(s + c)  
7  
8 permutaciones("")
```


Backtracking



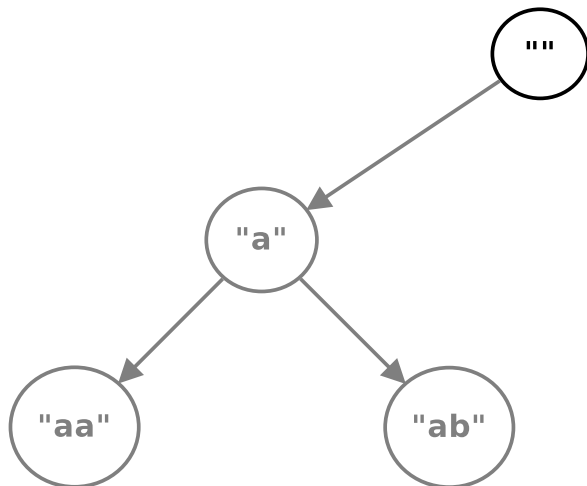
Backtracking

```
1 def permutaciones(s):  
2     if(len(s) == 2):  
3         print(s)  
4         return  
5     for c in ['a', 'b']:  
6         permutaciones(s + c)  
7  
8 permutaciones("")
```



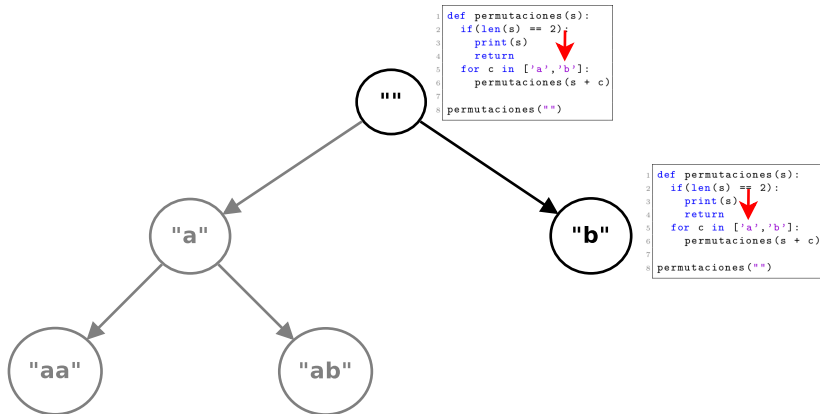
```
1 def permutaciones(s):  
2     if(len(s) == 2):  
3         print(s)   
4         return  
5     for c in ['a', 'b']:  
6         permutaciones(s + c)  
7  
8 permutaciones("")
```

Backtracking

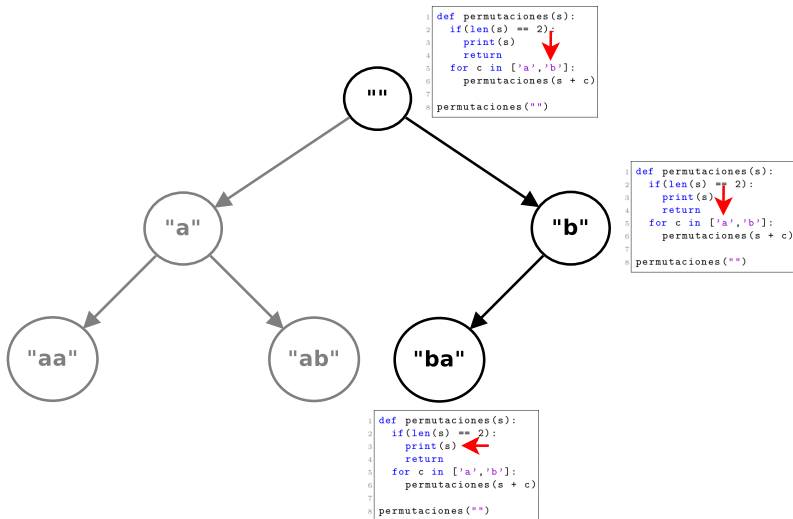


```
1 def permutaciones(s):  
2     if(len(s) == 2):  
3         print(s)      ↓  
4         return  
5     for c in ['a','b']:  
6         permutaciones(s + c)  
7  
8 permutaciones("")
```

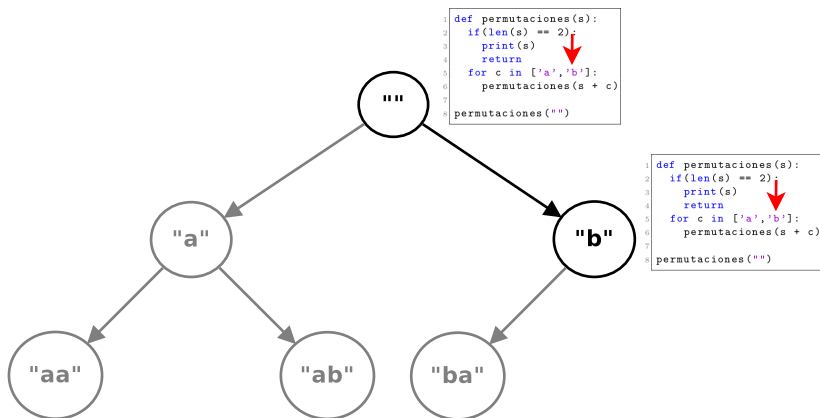
Backtracking



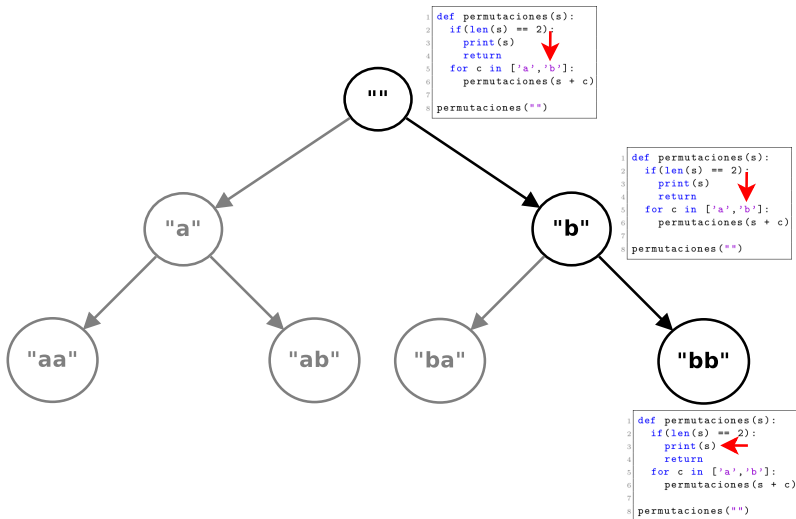
Backtracking



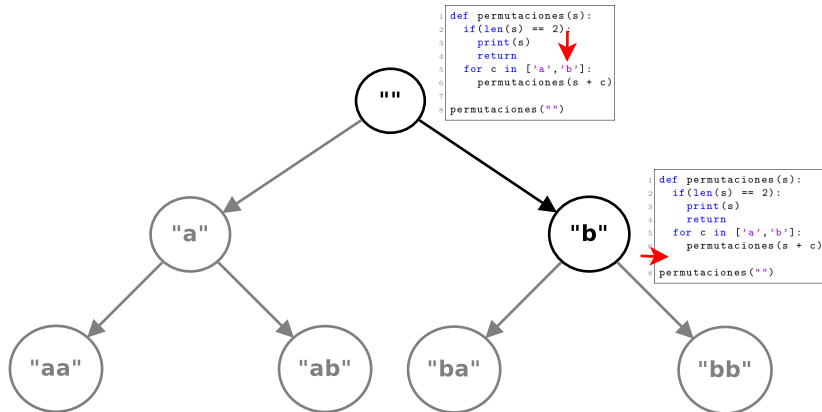
Backtracking



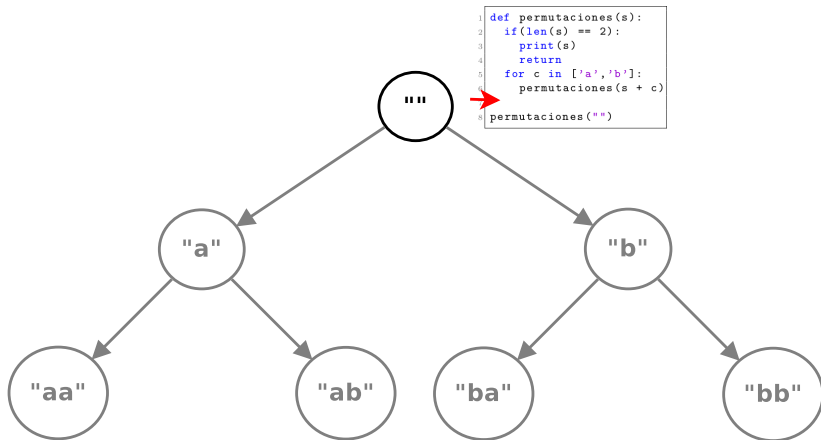
Backtracking



Backtracking



Backtracking



Backtracking

¿Cuáles son los elementos principales de una solución mediante backtracking?

Backtracking

¿Cuáles son los elementos principales de una solución mediante backtracking?

- Nodos o estados (S):
 - Definen el estado actual de la búsqueda.
 - Son los parámetros de la función.

Backtracking

¿Cuáles son los elementos principales de una solución mediante backtracking?

- Nodos o estados (S):
 - Definen el estado actual de la búsqueda.
 - Son los parámetros de la función.
- Estado inicial (s_0):
 - Es el estado en que parte la búsqueda.

Backtracking

¿Cuáles son los elementos principales de una solución mediante backtracking?

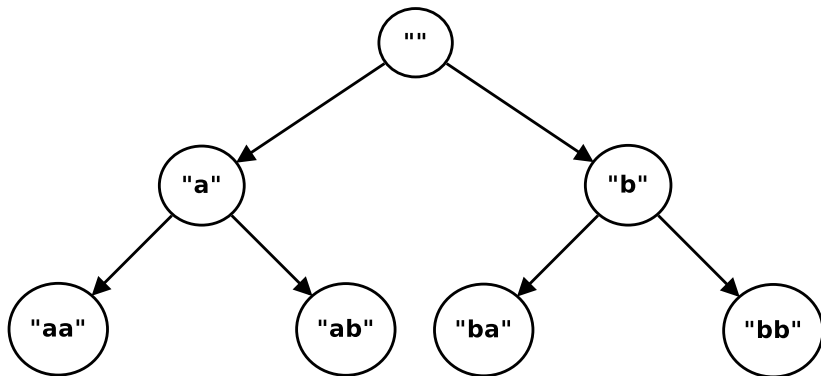
- Nodos o estados (S):
 - Definen el estado actual de la búsqueda.
 - Son los parámetros de la función.
- Estado inicial (s_0):
 - Es el estado en que parte la búsqueda.
- Aristas o acciones (A):
 - Llevan de un estado al siguiente.
 - Son los parámetros del llamado recursivo.

Backtracking

¿Cuáles son los elementos principales de una solución mediante backtracking?

- Nodos o estados (S):
 - Definen el estado actual de la búsqueda.
 - Son los parámetros de la función.
- Estado inicial (s_0):
 - Es el estado en que parte la búsqueda.
- Aristas o acciones (A):
 - Llevan de un estado al siguiente.
 - Son los parámetros del llamado recursivo.
- Caso base o estados objetivos(G):
 - Estados en los que retorno sin hacer otros llamados recursivos.

Backtracking



S: Strings de largo menor o igual a 2 formados por "a" y "b".

*s*₀: String de largo 0 ("")

A: Agregar una "a" o una "b".

G: Strings de largo 2.

Backtracking

Estructura general solución recursiva:

```
1 def backtracking(s):
2     # Caso base: Verifico si 's' está en G
3     if(es_objetivo(s)):
4         return ... # Retorno algún valor de interés
5
6     # Recorro estados sucesores de s
7     for h in s.sucesores:
8         # Aseguro no haber revisado 'h' aún
9         if(no_ha_sido_explorado(h)):
10            # Realizo llamado recursivo
11            p = backtracking(h)
12            ... # de ser necesario, uso p
13
14    ... # podría ser necesario retornar 'algo' luego de
        revisar los sucesores
```

Ejemplos

Problema misioneros y caníbales.

Ejemplos

Problema misioneros y caníbales.

Descripción:

- En la rivera de un río hay 3 caníbales, 3 misioneros y un bote.
- Todos deben cruzar el río en forma segura.
- El bote puede trasladar a lo más 2 personas.
- En ningún momento pueden haber más caníbales que misioneros en un lado del río.

Ejemplos

Problema misioneros y caníbales.

Descripción:

- En la riberas de un río hay 3 caníbales, 3 misioneros y un bote.
- Todos deben cruzar el río en forma segura.
- El bote puede trasladar a lo más 2 personas.
- En ningún momento pueden haber más caníbales que misioneros en un lado del río.

Programe un función que encuentre la solución del problema.

Ejemplos

Problema misioneros y caníbales.

Descripción:

- En la rivera de un río hay 3 caníbales, 3 misioneros y un bote.
- Todos deben cruzar el río en forma segura.
- El bote puede trasladar a lo más 2 personas.
- En ningún momento pueden haber más caníbales que misioneros en un lado del río.

Programa una función que encuentre la solución del problema.

$\text{¿}\langle S, A, s_0, G \rangle\text{?}$

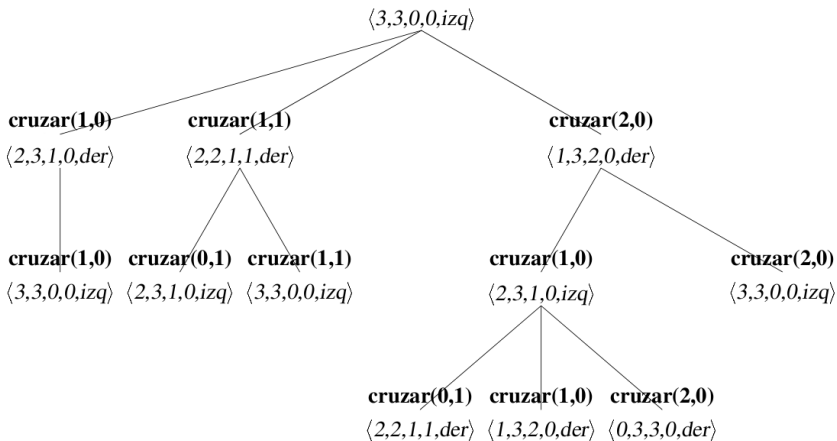
Ejemplos

Elementos importantes:

- S : $(C_i, M_i, C_d, M_d, \text{bote})$.
- A : Cruzar una o dos persona al otro lado del río.
- s_0 : $(3, 3, 0, 0, \text{izq})$
- G : $\{(0, 0, 3, 3, \text{der})\}$

Ejemplos

Backtracking



Ejemplos

IMPORTANTE

En nuestro algoritmo de backtracking debemos, de alguna forma, asegurar que no volvamos a revisar el mismo estado (de lo contrario podemos quedar en un loop infinito).

Ejemplos

IMPORTANTE

En nuestro algoritmo de backtracking debemos, de alguna forma, asegurar que no volvamos a revisar el mismo estado (de lo contrario podemos quedar en un loop infinito).

Opciones:

- Pensar una forma de recorrer las acciones tal que nunca se revise el mismo estado dos veces.

Ejemplos

IMPORTANTE

En nuestro algoritmo de backtracking debemos, de alguna forma, asegurar que no volvamos a revisar el mismo estado (de lo contrario podemos quedar en un loop infinito).

Opciones:

- Pensar una forma de recorrer las acciones tal que nunca se revise el mismo estado dos veces.
- Recordar estados revisados en una lista `close`.

Ejemplos

```
1 def backtracking(s,close):
2     close.append(s)
3     # Caso base: Verifico si 's' está en G
4     if(es_objetivo(s)):
5         return ... # Retorno algún valor de interés
6
7     # Recorro estados sucesores de s
8     for h in s.sucesores:
9         # Aseguro no haber revisado 'h' aún
10        if(h not in close)::
11            # Realizo llamado recursivo
12            p = backtracking(h,close)
13            ... # de ser necesario, uso p
14
15    ... # podría ser necesario retornar 'algo' luego de
        revisar los sucesores
```

Ejemplos

Misioneros y caníbales:

Ejemplos

Misioneros y caníbales:

```
38 l = resolver([3,3,0,0,'izq'], [])
39 print(' c m c m bote')
40 for m in l:
41     print(m)
```

Demo.

Ejemplos

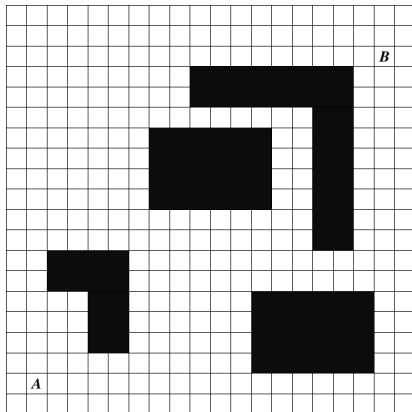
```
19 def resolver(s, close):
20     # Casos bases
21     if(s == [0,0,3,3,'der']):
22         return [s]
23     # Estado inválido
24     if(0 < s[1] < s[0] or 0 < s[3] < s[2]):
25         return []
26     # Agrego estado a close
27     close.append(s)
28     # Realizo cambios de estado
29     for m in sucesores(s):
30         # Si no está en close reviso este estado
31         if(m not in close):
32             r = resolver(m, close)
33             if(len(r) > 0):
34                 return [s] + r
35     return []
```

Ejemplos

```
1 def sucesores(s):
2     mov_izq = [[-2,0,2,0], [0,-2,0,2],
3                [-1,0,1,0], [0,-1,0,1],
4                [-1,-1,1,1]]
5     d = 1; pos = 'der'
6     if(s[4] == 'der'):
7         d = -1; pos = 'izq'
8     movimientos = []
9     for m in mov_izq:
10        m_n = []
11        for i in range(len(m)):
12            m_n.append(s[i]+d*m[i])
13        if(min(m_n) < 0):
14            continue
15        m_n.append(pos)
16        movimientos.append(m_n)
17    return movimientos
```

Ejemplos

Laberinto



Ejemplos

Laberinto:

```
1  XXXXXXXXXXXXXXXXXXXXXXXX
2  X   O                       X X X
3  XXXXXXXXXXXXXXXX X X X X
4  X X                       X X X   X
5  X  XXXXXXXX   X  XXXXXX X
6  X X           X X           X X
7  X X XXXXX   X  XXXXXX X
8  X X           X                       X
9  X  xg  x  XXXXXXXXXXXXXXXX X
10 X     x                       X
11 XXXXXXXXXXXXXXXXXXXXXXXX
```

"/maze_3.txt"

Ejemplos

Reglas:

- Se debe leer el laberinto desde un archivo.
- Se tiene un agente y un objetivo en el laberinto.
- Se debe llevar al agente hasta el objetivo.
- Al finalizar, se muestra el camino que siguió el agente.

Ejemplos

Reglas:

- Se debe leer el laberinto desde un archivo.
- Se tiene un agente y un objetivo en el laberinto.
- Se debe llevar al agente hasta el objetivo.
- Al finalizar, se muestra el camino que siguió el agente.

Elementos importantes:

- S : Posiciones válidas de **A**.
- A : Mover agente un paso en alguna dirección.
- s_0 : Posición inicial del agente.
- G : Estado con agente en posición **B**.

Ejemplos

```
1 class laberinto:
2     # Constructor, lee laberinto y localiza al agente
3     def __init__(self, path):
4         self.leer_laberinto(path)
5         self.encontrar_agente()
6     # Muestra el tablero
7     def mostrar(self):
8         for fila in self.lab:
9             print(''.join(fila))
10    # Lee el laberinto desde el archivo
11    def leer_laberinto(self, path):
12        archivo = open(path)
13        self.lab = []
14        for linea in archivo:
15            lab_linea = []
16            for c in linea.rstrip('\n'):
17                lab_linea.append(c)
18            self.lab.append(lab_linea)
19        archivo.close()
```

Ejemplos

```
20 # Encuentra la posición inicial del agente
21 def encontrar_agente(self):
22     for i in range(len(self.lab)):
23         for j in range(len(self.lab[i])):
24             if self.lab[i][j] == 'g':
25                 self.pos = (i,j)
26                 return
27 # Método para resolver el laberinto
28 def resolver(self):
29     self.mover(self.pos, [])
```


Ejemplos

```
32 def mover(self, pos, close):
33     i, j = pos; close.append(pos)
34     # Caso base, llego al objetivo
35     if self.lab[i][j] == 'o':
36         return True
37     # Me muevo un paso en alguna dirección
38     for di in [-1,0,1]:
39         for dj in [-1,0,1]:
40             if(di == dj == 0): continue
41             # Sólo entro si la posición no es un muro
42             if(self.lab[i+di][j+dj] != 'x'):
43                 # Genero nuevo estado
44                 pos_nueva = (i+di, j+dj)
45                 # Si no está en close, me muevo
46                 if(pos_nueva not in close and
47                    self.mover(pos_nueva, close)):
48                     self.lab[i][j] = '.'
49                     return True
50     return False
```

Ejemplos

Sudoku:

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

Ejemplos

Elementos importantes:

- S : Tablero con números del 0 al 9 que cumple reglas del sudoku.
- A : Agregar un dígito a una casilla vacía.
- s_0 : Tablero inicial leído desde el archivo.
- G : Estado sin casillas vacías.

Ejemplos

Elementos importantes:

- S : Tablero con números del 0 al 9 que cumple reglas del sudoku.
- A : Agregar un dígito a una casilla vacía.
- s_0 : Tablero inicial leído desde el archivo.
- G : Estado sin casillas vacías.

... resolverlo queda como ejercicio propuesto.

Ejemplos

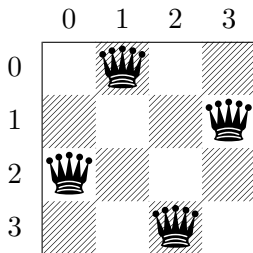
8-Reinas

Es sabido que se pueden poner 4 reinas sobre un tablero de 4×4 tal que ninguna reina está atacando a otra.

Ejemplos

8-Reinas

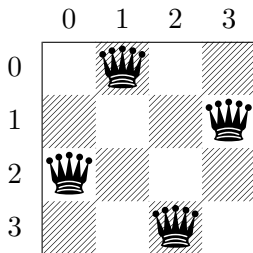
Es sabido que se pueden poner 4 reinas sobre un tablero de 4×4 tal que ninguna reina está atacando a otra.



Ejemplos

8-Reinas

Es sabido que se pueden poner 4 reinas sobre un tablero de 4×4 tal que ninguna reina está atacando a otra.



¿Es posible poner 8 reinas sobre un tablero de 8×8 ?

Ejemplos

Estrategia:

- S : Lista con reinas tal que no se atacan unas a otras.
- A : Agregar una reina.
- s_0 : Lista vacía.
- G : Lista con 8 reinas.

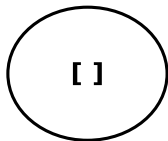
Ejemplos

Estrategia:

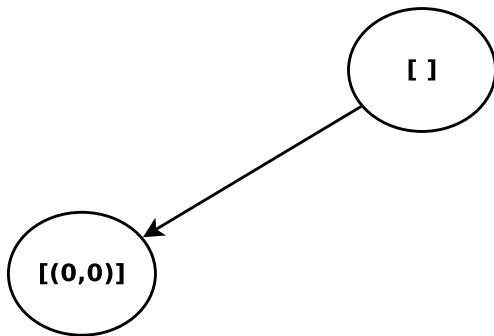
- S : Lista con reinas tal que no se atacan unas a otras.
- A : Agregar una reina.
- s_0 : Lista vacía.
- G : Lista con 8 reinas.

Ejemplo: Consideremos el caso de poner 2 reinas en un tablero de 2×2 .

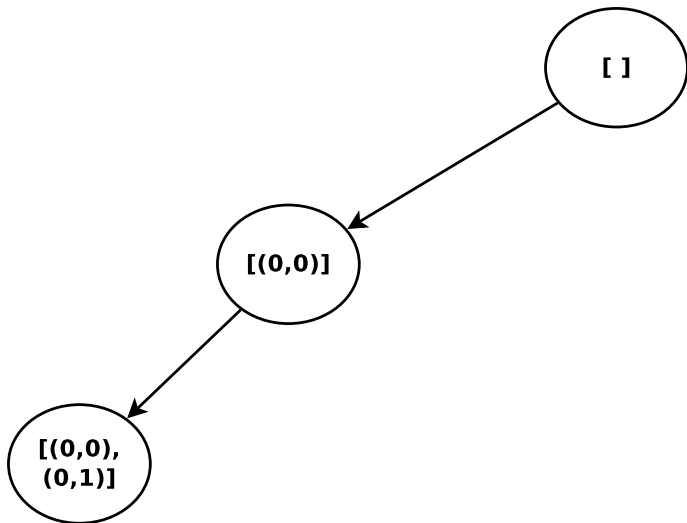
Ejemplos



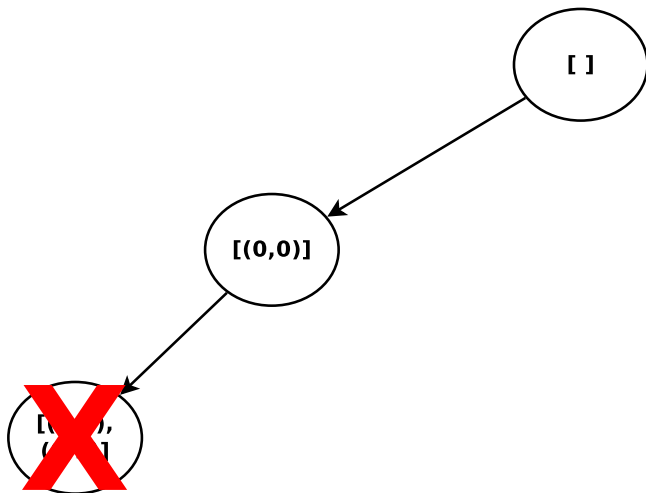
Ejemplos



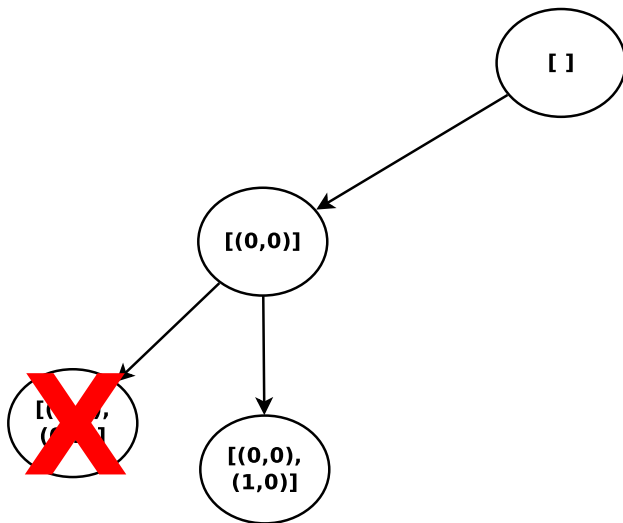
Ejemplos



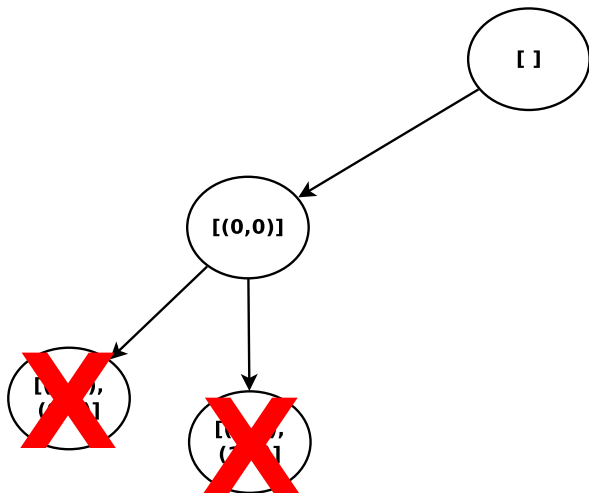
Ejemplos



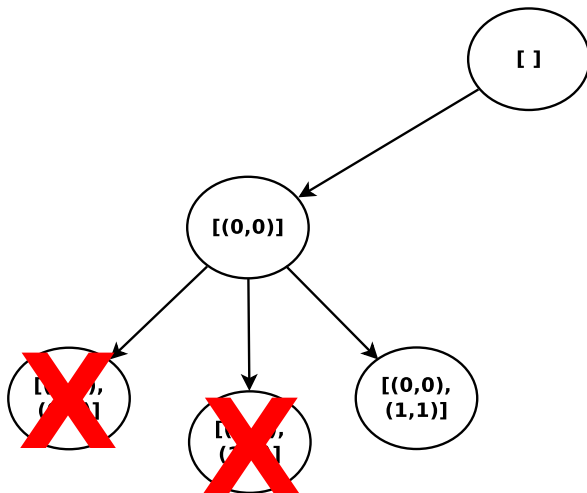
Ejemplos



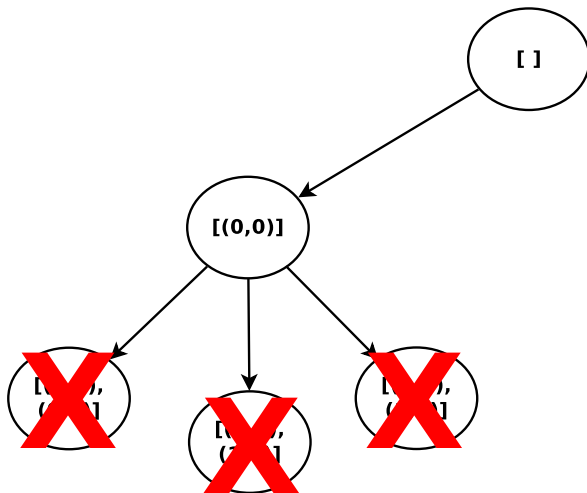
Ejemplos



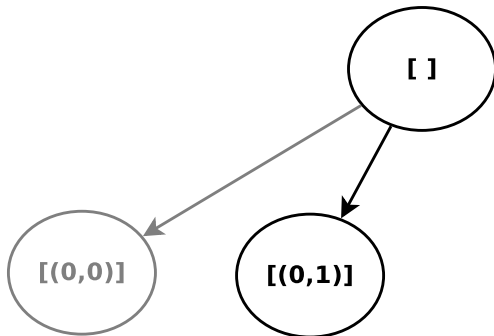
Ejemplos



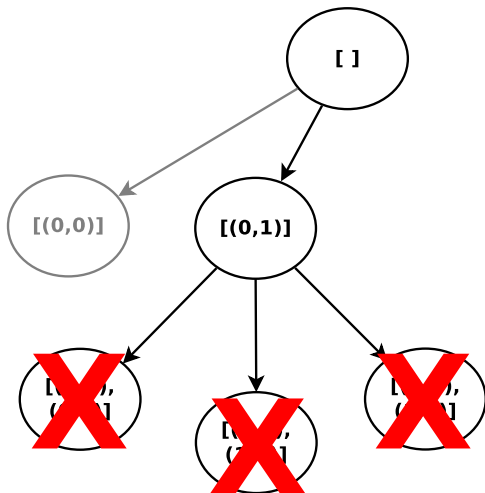
Ejemplos



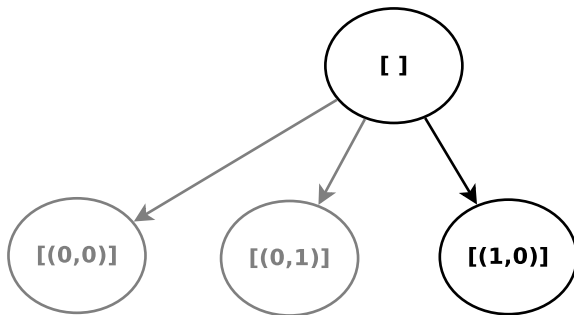
Ejemplos



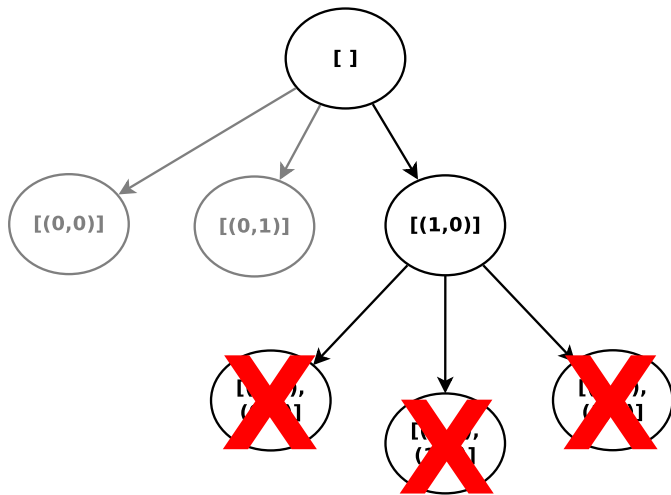
Ejemplos



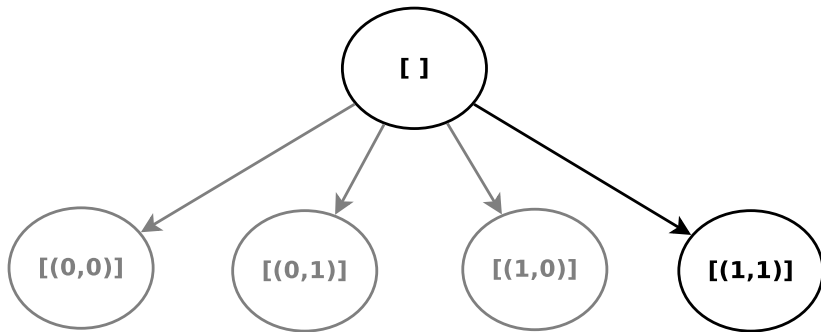
Ejemplos



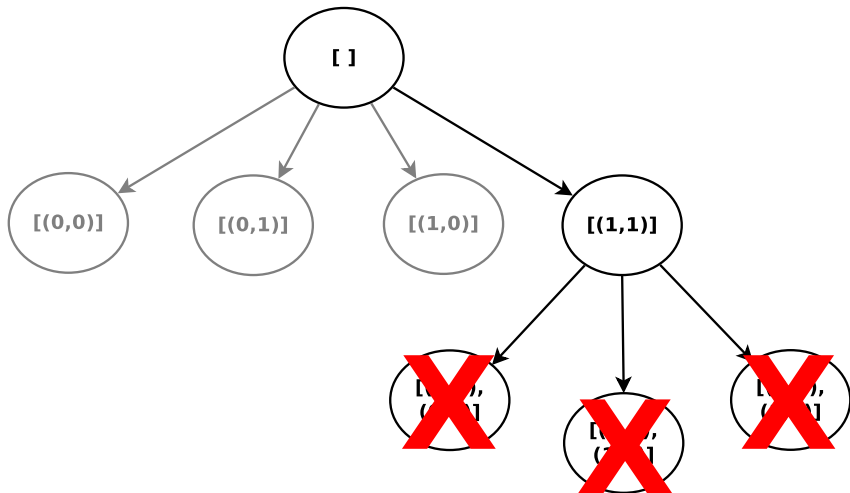
Ejemplos



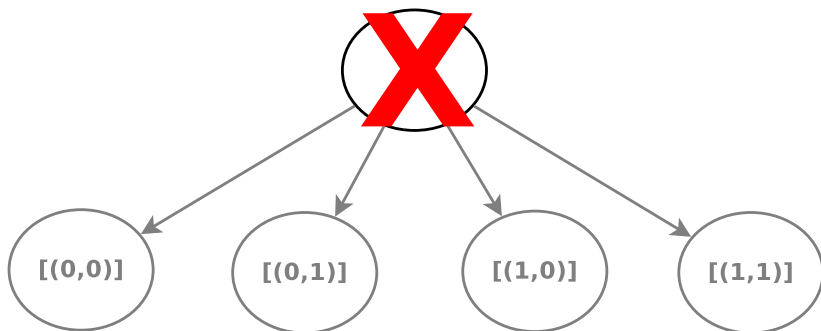
Ejemplos



Ejemplos



Ejemplos



Ejemplos

Programemos este ejemplo.

Ejemplos

Programemos este ejemplo.

¿Cómo codificamos las reinas?

Ejemplos

Programemos este ejemplo.

¿Cómo codificamos las reinas?

Creemos la clase `Reina` con su posición en el tablero y su constructor.

Ejemplos

Programemos este ejemplo.

¿Cómo codificamos las reinas?

Creemos la clase `Reina` con su posición en el tablero y su constructor.

Agreguemos el método `esta_atacando(self,r)` tal que recibe una reina `r` y retorna `True` ssi la reina `self` está atacando a `r`.

Ejemplos

Implementa la función `resolver(...)` que retorna `True` ssi es posible posicionar `n` reinas en un tablero de `n × n`.

Ejemplos

Implementa la función `resolver(...)` que retorna `True` ssi es posible posicionar `n` reinas en un tablero de `n × n`.

- ¿Qué parámetros debería recibir?

Ejemplos

Implementa la función `resolver(...)` que retorna `True` ssi es posible posicionar `n` reinas en un tablero de `n × n`.

- ¿Qué parámetros debería recibir?
- ¿Cuál debería ser el valor inicial de `reinas`?

Ejemplos

Implementa la función `resolver(...)` que retorna `True` ssi es posible posicionar `n` reinas en un tablero de `n × n`.

- ¿Qué parámetros debería recibir?
- ¿Cuál debería ser el valor inicial de `reinas`?
- ¿Cuál es el caso base?

Ejemplos

Implementa la función `resolver(...)` que retorna `True` ssi es posible posicionar `n` reinas en un tablero de `n × n`.

- ¿Qué parámetros debería recibir?
- ¿Cuál debería ser el valor inicial de `reinas`?
- ¿Cuál es el caso base?
- ¿Cómo debería ser la llamada recursiva?

Ejemplos

Implementa la función `resolver(...)` que retorna `True` ssi es posible posicionar `n` reinas en un tablero de `n × n`.

- ¿Qué parámetros debería recibir?
- ¿Cuál debería ser el valor inicial de `reinas`?
- ¿Cuál es el caso base?
- ¿Cómo debería ser la llamada recursiva?

¡Resuelve el problema!

Ejercicios

- 1) Modifica tu código de las n reinas para que la función `resolver(...)` retorne la lista con las posiciones de las reina en el tablero resuelto (`None` si no es posible).
- 2) Programa un algoritmo recursivo que resuelva el Sudoku difícil de tu Tarea 1.