

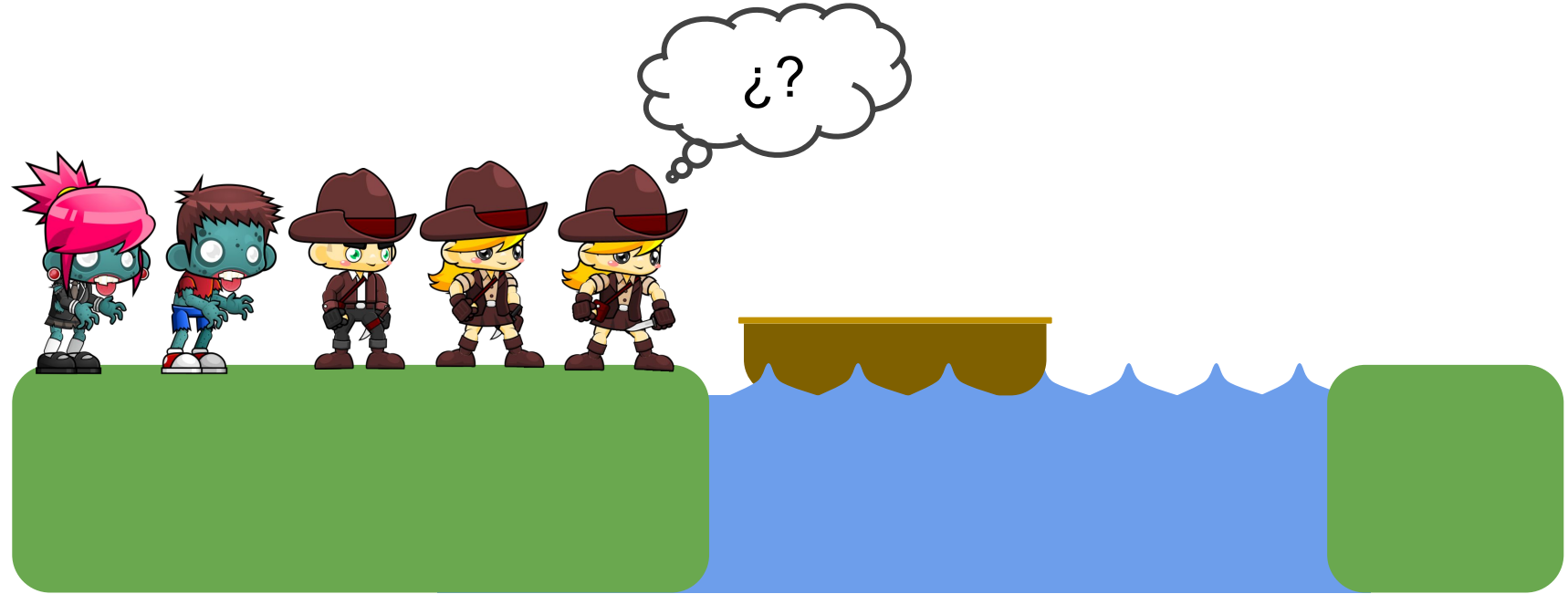
1. RESOLUCIÓN DE PROBLEMAS MEDIANTE BÚSQUEDA

IA 3.2 - Programación III

1° C - 2023

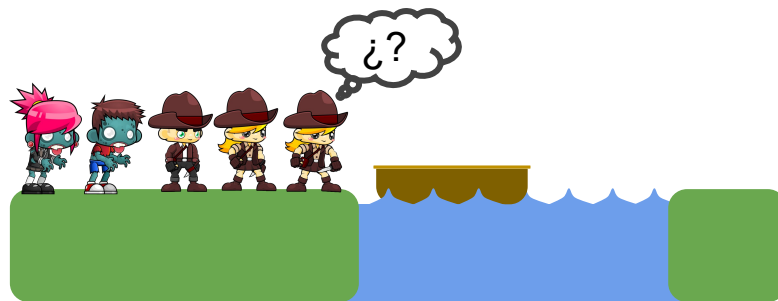
Lic. Mauro Lucci

Problema de cruce de río



Descripción

— — —



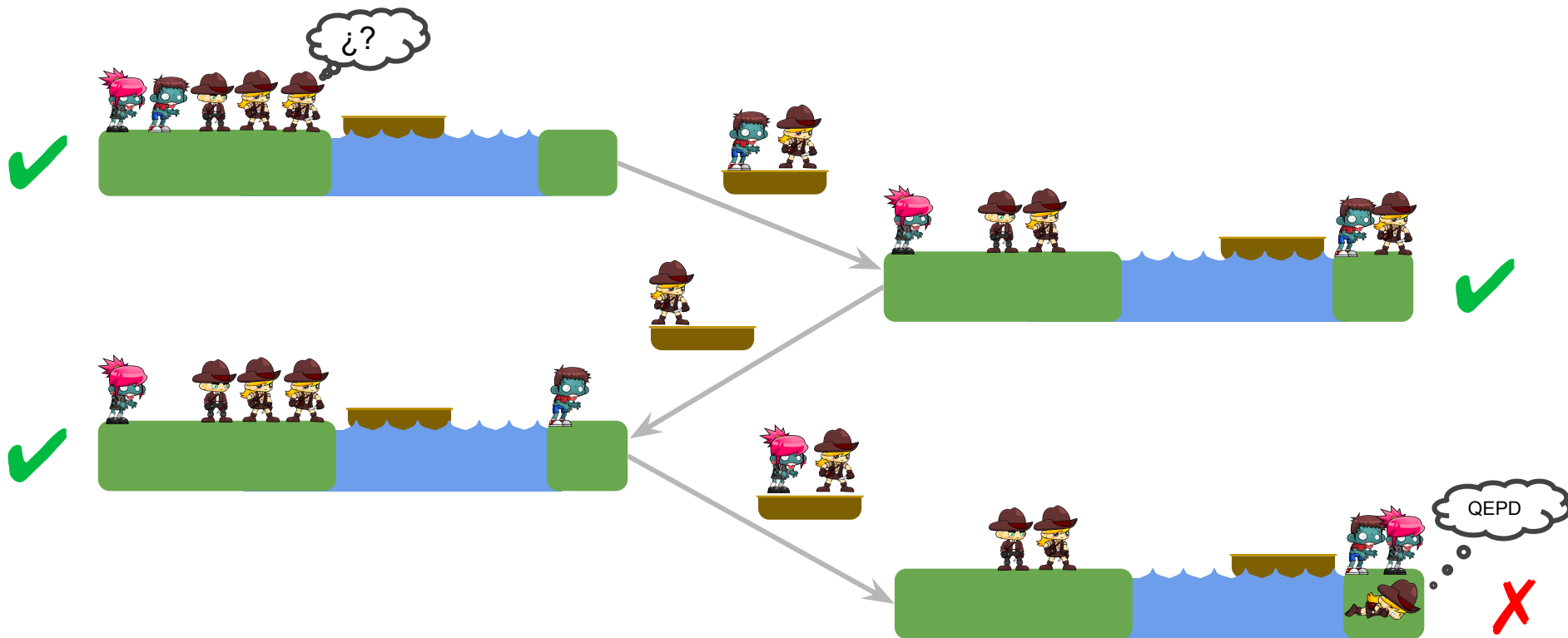
Objetivo. Todos deben llegar al otro lado sanos y salvos, en el menor número de cruces.

Reglas.

1. El río solo se cruza en bote.
2. La capacidad del bote es de 1 o 2 personas (humanas o zombies).
3. Al menos 1 persona (humana o zombie) debe estar en el bote para navegar.
4. Si en algún lado del río la cantidad de zombies supera a la de humanos (incluso por un momento), los zombies comerán a los humanos.



Ejemplo





Desafío

— — —

¿Es posible resolver el problema?

En caso afirmativo, ¿cuántos cruces requiere la solución que encontró?

Problema y soluciones

Un **problema** de búsqueda se define por:

1. Estado inicial.
2. Acciones.
3. Modelo transicional.
4. Test objetivo.
5. Costo de camino.

Formulación. Proceso de decidir qué estados y acciones describen a un problema.

Abstracción. Proceso de remover de una representación detalles irrelevantes.

— — —

1. Estado inicial

Estado. Representación abstracta de los elementos y del entorno del problema, en un instante dado.

Estado inicial. Estado del problema en el instante inicial.



Ejemplo

— — —

Estado.

Cantidad en
lado izq.

Cantidad en
lado der.



Zombies (fila suma 2)






Humanos (fila suma 3)



Bote (fila suma 1)

Estado inicial.

Notación compacta:

(     ,)



Ejercicio

— — —

1. ¿Cuántos estados hay?
2. ¿Todos son *posibles*?



Respuesta

— — —

1. ¿Cuántos estados hay?

$$3 \times 4 \times 2 = 24.$$

2. ¿Todos son *posibles*?

No. Por ejemplo, los estados (    , ) y (,     ) .

2. Acciones

Las **acciones** modifican el estado actual.

Dado un estado **S**, la función **ACCIONES(S)** retorna el conjunto de acciones que se pueden ejecutar en S.



Ejemplo

— — —

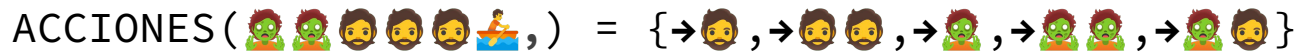
Acciones:

❑ Llevar 1 o 2 personas del lado izquierdo al lado derecho:

→👤, →👤👤, →🧟, →🧟🧟, →🧟👤.

❑ Llevar 1 o 2 personas del lado derecho al lado izquierdo:

←👤, ←👤👤, ←🧟, ←🧟🧟, ←🧟👤.














Ejemplo



$\text{ACCIONES}(\text{Man Man Man}, \text{Zombie Zombie Boat}) = \{\leftarrow \text{Zombie}, \leftarrow \text{Zombie Zombie}\}$

$\text{ACCIONES}(\text{Zombie Zombie Man}, \text{Man Man Boat}) = \{ \}$

3. Modelo transicional

El **modelo transicional** describe qué hace cada acción.

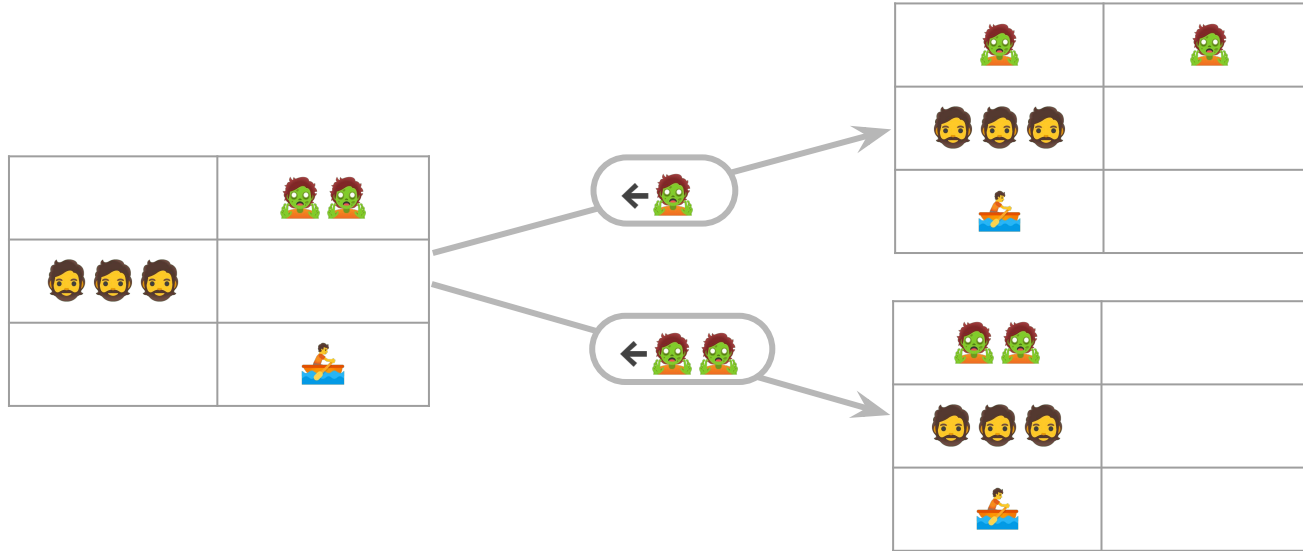
Dada una acción **A** y un estado **S**, la función **RESULTADO(S,A)** retorna el estado que resulta de aplicar A en S.

Si $\text{RESULTADO}(S,A) = S'$, entonces S' es un **sucesor** de S.



Ejemplo

— — —



RESULTADO((  ,  , , ) = (    , )

RESULTADO((  ,  , ,  ) = (     ,

Espacio de estados

— — —

El estado inicial, las acciones y el modelo transicional definen implícitamente el **espacio de estados** del problema, que es el conjunto de estados alcanzables desde el estado inicial mediante cualquier secuencia de acciones.

Cuando es finito, se representa con un **grafo dirigido**, donde los **nodos** son estados y los **arcos** son acciones.

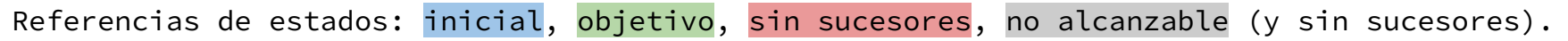
Un **camino** en el espacio de estados es una secuencia de estados conectados mediante una secuencia de acciones.



Ejercicio*

— — —

Dibujar el grafo dirigido que representa el espacio de estados del problema de cruce de río.



4. Test objetivo

El **test objetivo** determina si un estado dado es un estado objetivo.

También se puede especificar mediante una **propiedad** en lugar de una enumeración explícita de estados.

📖 **Ejemplo (Cruce de río).** Dado un estado S , ¿ $S \in \{(\text{👤👤👤👤👤🚣})\}$?

📖 **Ejemplo (Ajedrez).** ¿Hay jaque mate?

5. Costo de camino

— — —

El **costo de camino** es una función que asigna un costo numérico a cada camino. Asumimos que el costo de todo camino es igual a la suma de los costos individuales de cada acción del camino.

El **costo individual** de realizar una acción **A** que lleva de un estado **S** a un estado **S'** se denota **$c(S,A,S')$** , y lo asumimos no-negativo.

Ejemplo

$$c((\text{👤👤👤}, \text{🌿🌿🚣}), \leftarrow \text{🌿}, (\text{🌿👤👤👤🚣}, \text{🌿})) = 1$$

$$c((\text{👤👤👤}, \text{🌿🚣}), \leftarrow \text{🌿🌿}, (\text{🌿🌿👤👤👤🚣},)) = 1$$

...

$c(S,A,S') = 1$, para toda acción A que lleva de un estado S a un estado S.

Los costos individuales son unitarios, luego el costo de todo camino es la cantidad de acciones que efectúa.

Solución

— — —

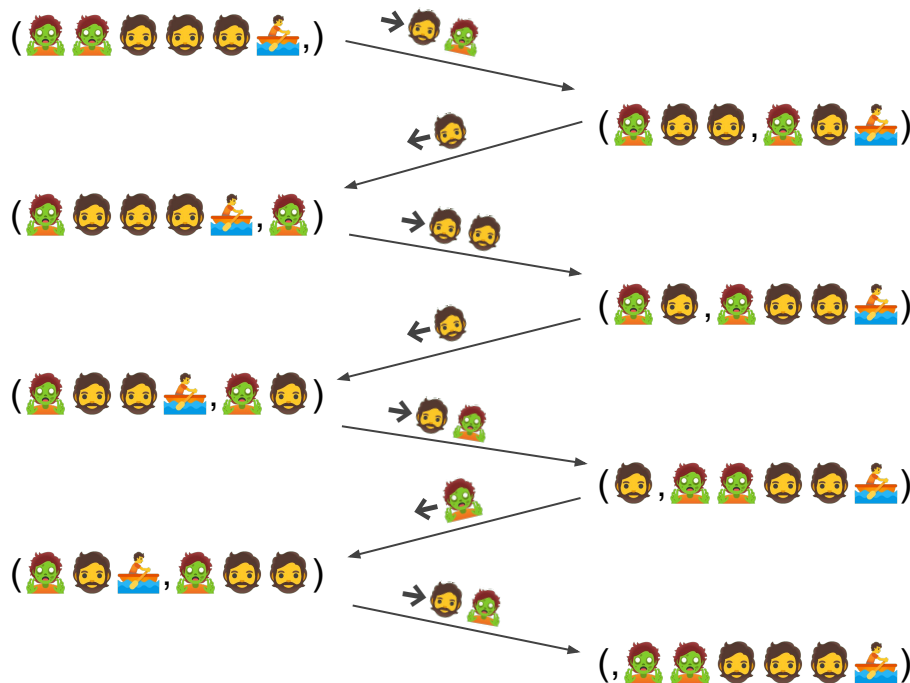
Dado un problema, una **solución** es un camino en el espacio de estados desde el estado inicial a un estado objetivo.

Una **solución óptima** tiene el menor costo de camino entre todas las soluciones.

El proceso de buscar una solución para un problema se llama **búsqueda**.

Ejemplo

Solución óptima de costo 7 (no es única):





Resumen

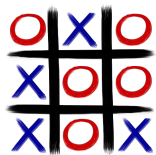
— — —

- ❑ Primero, identificar un objetivo y formular un problema bien definido.
- ❑ Un problema tiene 5 componentes: un estado inicial, un conjunto de acciones, un modelo transicional, una función de test objetivo y una función de costo de camino.
- ❑ El espacio de estados de un problema se representa con un grafo dirigido (cuando es finito). Los nodos representan los estados y los arcos representan las acciones.
- ❑ Una solución es un camino en el espacio de estados desde el estado inicial a un estado objetivo.

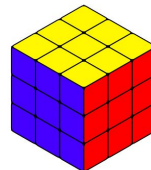


Próximamente

Formulado nuestro problema, lo siguiente es resolverlo. Problemas más interesantes suelen tener miles o millones de estados.



Problema	Estados
Ta-te-ti	$3^9 = 19683$ (765)
Cubo Rubik 2x2	$7! \times 3^6 = 3674160$
Cubo Rubik 3x3	4.3×10^{19}
Ajedrez 8x8	$\approx 10^{44}$



Buscar soluciones de manera eficiente en espacios de estados grandes o infinitos es una de las áreas más importantes de la IA. Veremos **algoritmos de búsqueda generales**, que pueden aplicarse para resolver cualquier problema de búsqueda.

Búsqueda de soluciones

1. Algoritmos de búsqueda en árboles.
2. Algoritmos de búsqueda en grafos.

— — —

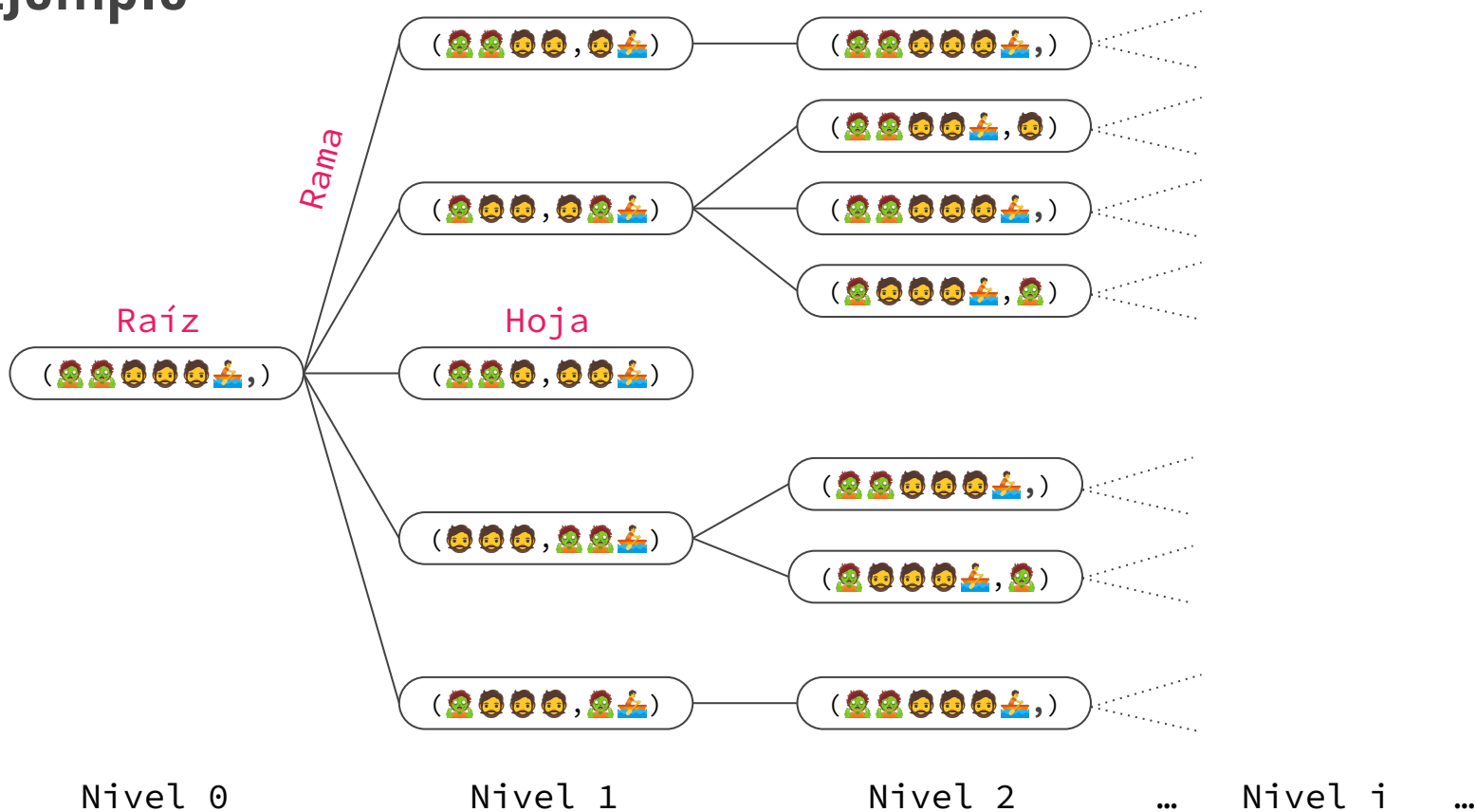
1. Árbol de búsqueda

Las posibles secuencias de acciones desde el estado inicial forman un **árbol de búsqueda**.

- La **raíz** contiene el estado inicial.
- Cada **nodo** contiene algún estado del espacio de estados.
- De cada nodo (**padre**) y por cada acción, hay una **rama** que lo conecta con un nodo (**hijo**) con el estado resultado.
- Una **hoja** es un nodo sin hijos (estados sin sucesores).



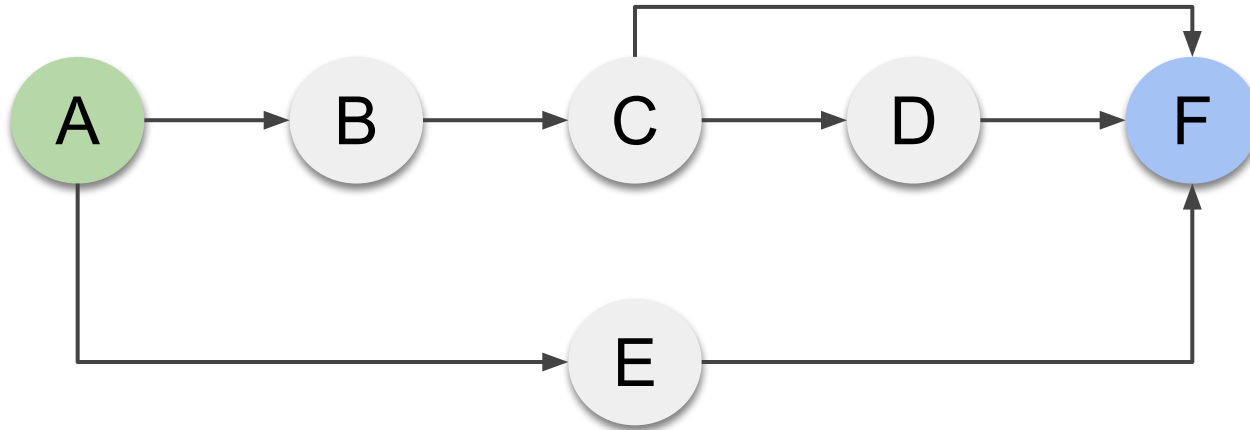
Ejemplo





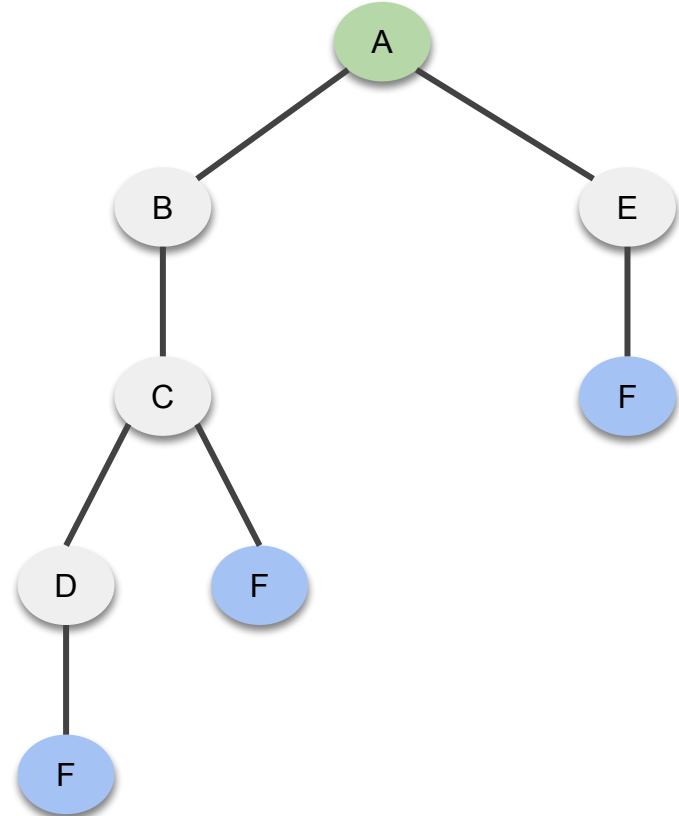
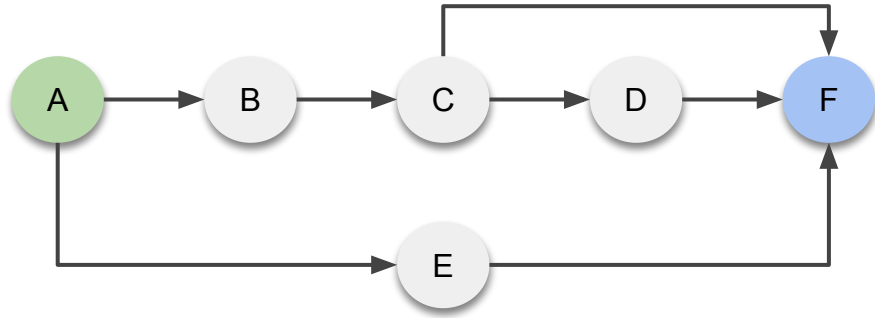
Ejercicio

Dar el árbol de búsqueda para el siguiente espacio de estados, donde A es el estado inicial y F el estado objetivo.





Respuesta



Búsqueda en árboles

— — —

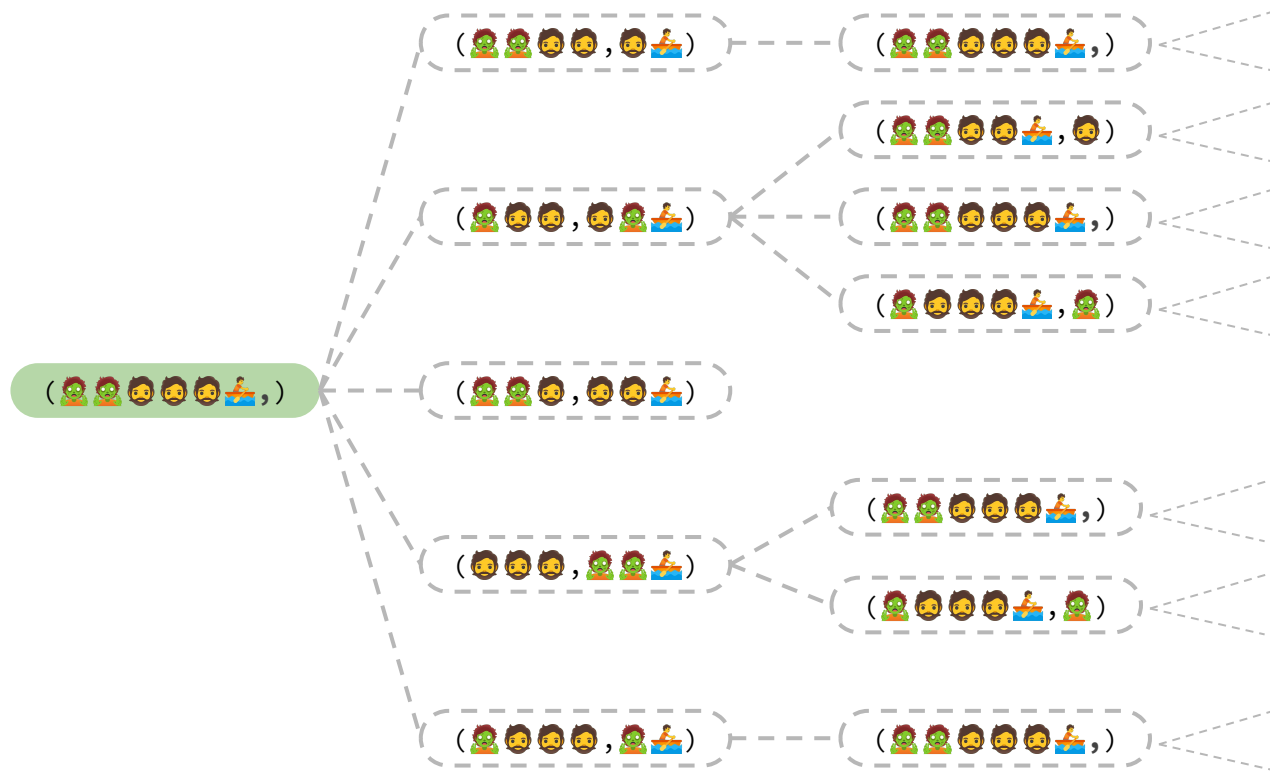
No siempre es necesario o posible generar el árbol completo para buscar una solución.

La idea es mantener un árbol parcial y expandirlo si los nodos generados hasta el momento no contienen un estado objetivo.


1. Comenzamos por la raíz.
2. ¿El nodo actual contiene un estado objetivo?
3. En caso afirmativo, retornamos la solución.
4. De lo contrario, **expandimos** el nodo actual y **generamos** sus hijos.
5. Elegimos un nodo de la **frontera** (nodos generados sin expandir) y volvemos al paso 2.




Ejemplo



Referencias:

 Nodo no generado

● Nodo de frontera

 Nodo expandido



Algoritmo general de búsqueda en árboles

— — —

Todos los algoritmos de búsqueda en árboles comparten esta estructura básica:

```
1 function TREE-SEARCH(problema) return solución o fallo
2   generar la raíz con el estado inicial
3   inicializar la frontera con la raíz
4   do
5       if la frontera es vacía then return fallo
6       elegir un nodo y removerlo de la frontera
7       if el nodo contiene un estado objetivo then return solución
8       expandir el nodo y agregar los nodos generados a la frontera
```



Algoritmo general de búsqueda en árboles

— — —
Todos los algoritmos de búsqueda en árboles comparten esta estructura básica:

```
1 function TREE-SEARCH(problema) return solución o fallo
2   generar la raíz con el estado inicial
3   inicializar la frontera con la raíz
4   do
5       if la frontera es vacía then return fallo
6       elegir un nodo y removerlo de la frontera
7       if el nodo contiene un estado objetivo then return solución
8       expandir el nodo y agregar los hijos generados a la frontera
```

El criterio de elección del próximo nodo a expandir da lugar a diferentes **estrategias de búsqueda**.

Estructura del nodo

Un nodo n se puede representar con una estructura con 4 componentes:

- **$n.estado$** estado que representa n .
- **$n.padre$** el nodo del árbol que generó a n .
- **$n.acción$** la acción que se aplicó en el padre para generar a n .
- **$n.costo$** costo del camino de la raíz a n .
- **$n.profundidad$** (opcional) nivel del árbol en el que se encuentra n .

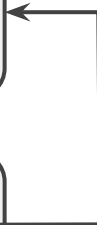


Ejemplo

— — —

estado = (🧟🧟🧔🧔🧔🧑🚶,)
padre = None
acción = None
costo = 0

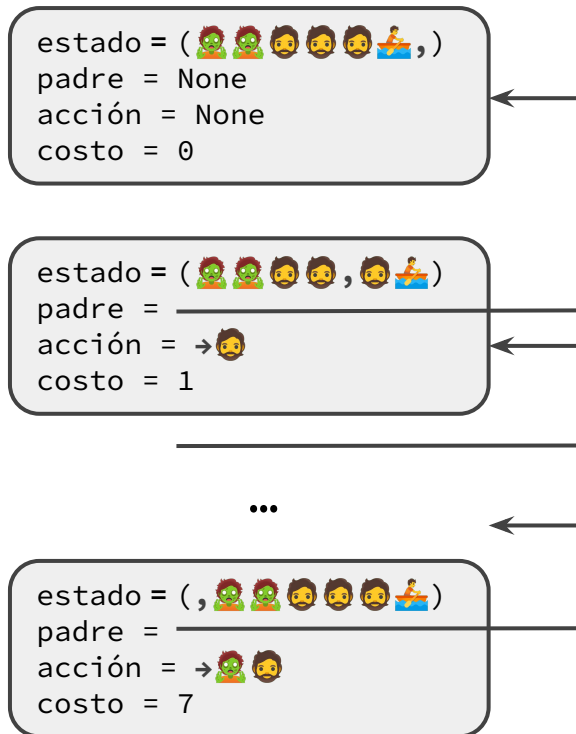
estado = (🧟🧟🧔🧔,🧑🚶)
padre =
acción = → 🧔
costo = 1





Ejemplo

— — —



Una **solución** se puede obtener invirtiendo el camino que comienza en un nodo que contiene un estado objetivo y escala por los padres hasta llegar a la raíz.



Camino cíclicos

El árbol de búsqueda de un problema puede contener **estados repetidos**, generados por **camino cíclicos** en el espacio de estados.

Esto incluye todos los problemas donde las acciones son **reversibles**.

Para estos problemas, el árbol de búsqueda es **infinito**. Es decir, **un problema con un espacio de estados finitos puede tener un árbol de búsqueda infinito y el algoritmo general de búsqueda podría no terminar.**



Ejemplo

— — —



se genera el siguiente árbol de búsqueda parcial:

(🧟🧟👤👤👤🚣,)



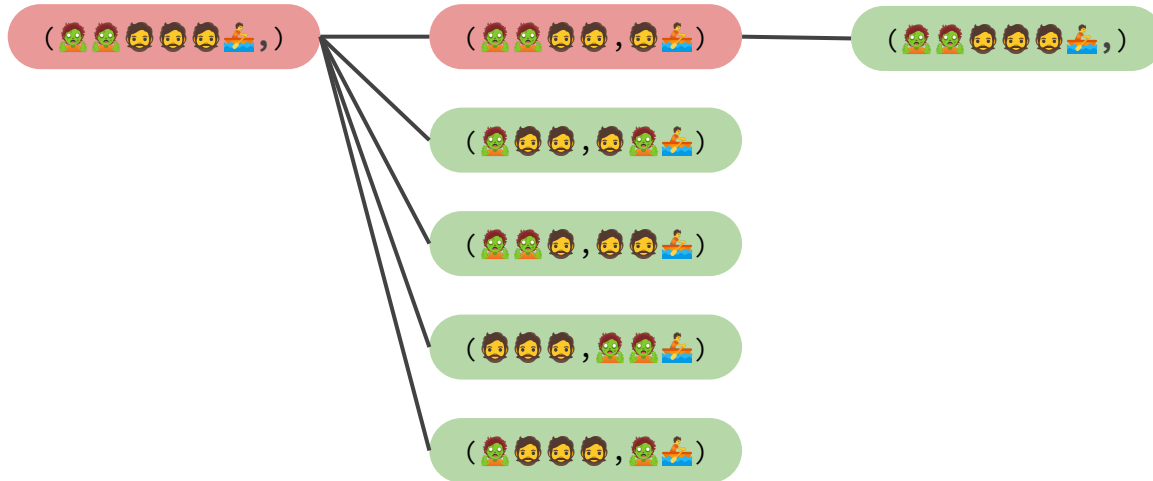
Al seguir el camino (🧟🧟👤👤👤🚣,) $\xrightleftharpoons[\text{👤}]{\text{👤}}$ (🧟🧟👤👤,👤🚣)

The diagram illustrates the concept of permutations. It starts with a sequence of 7 emojis in a pink box: 🧑🏿, 🧑🏿, 🧑🏿, 🧑🏿, 🧑🏿, 🧑🏿, and 🏊. This sequence branches into five green boxes, each containing a different permutation of the same 7 emojis, showing that the order of elements can change while the elements themselves remain the same.



Al seguir el camino (🧟🧟👤👤👤🏊,) $\xrightleftharpoons[\text{👤}]{\text{👤}}$ (🧟🧟👤👤,👤🏊)

se genera el siguiente árbol de búsqueda parcial:



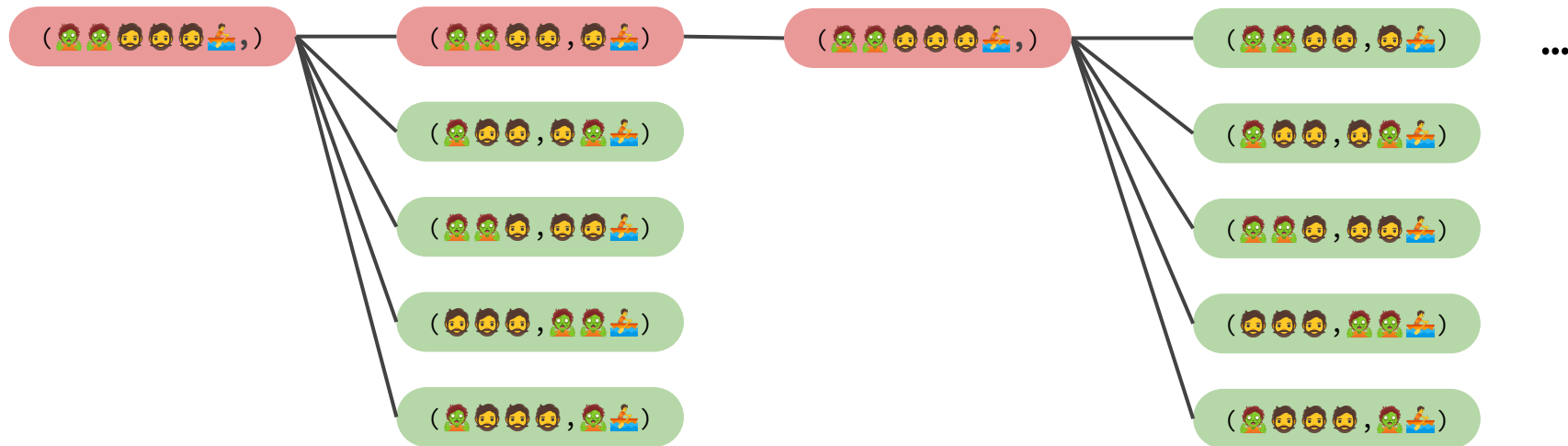


Ejemplo

— — —



se genera el siguiente árbol de búsqueda parcial:





Detección de caminos cíclicos

— — —

Antes de agregar un nuevo nodo a la frontera, controlamos que no repita estados en el camino de la raíz a su padre.

Este control es fácil de agregar, aprovechando que cada nodo almacena a su padre.

Algoritmo general de búsqueda en árboles con detección de ciclos

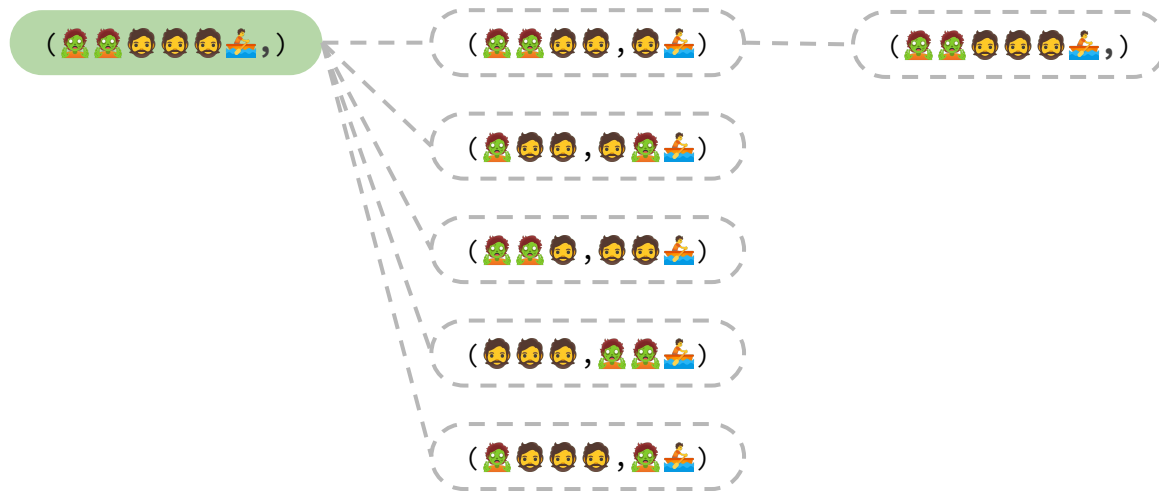
— — —

```
1 function TREE-SEARCH(problema) return solución o fallo
2   generar la raíz con el estado inicial
3   inicializar la frontera con la raíz
4   do
5     if la frontera es vacía then return fallo
6     elegir un nodo y removerlo de la frontera
7     if el nodo contiene un estado objetivo then return solución
8     for all hijo generado al expandir el nodo
9       if el hijo NO genera un ciclo then
10        agregar el hijo a la frontera
```



Ejemplo

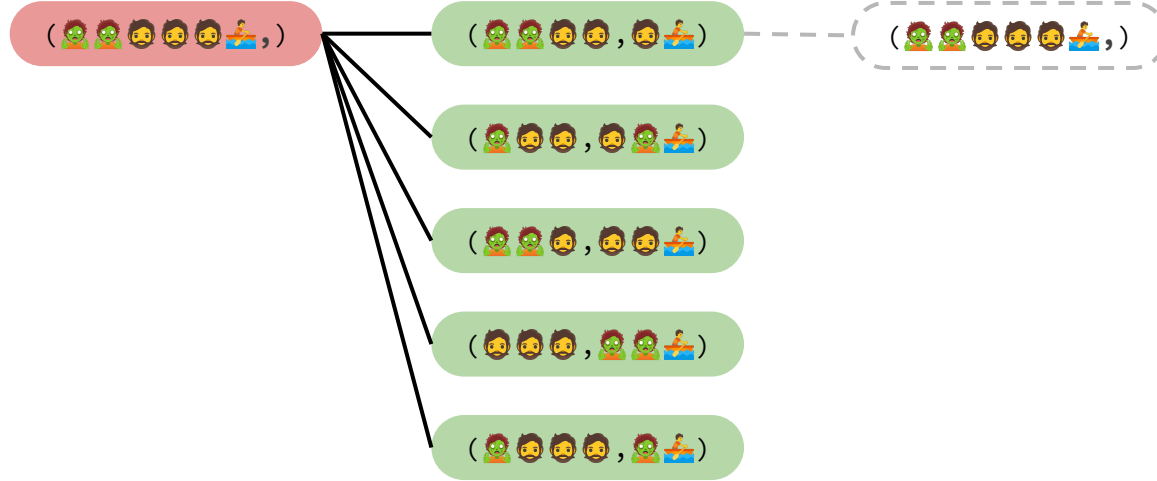
— — —





Ejemplo

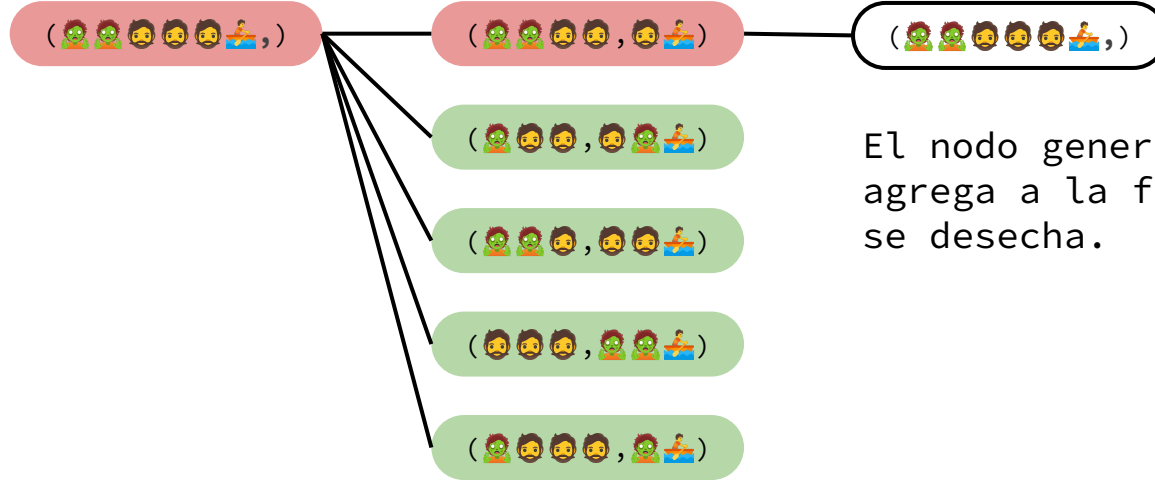
— — —





Ejemplo

— — —



El nodo generado no se
agrega a la frontera,
se desecha.



Caminos redundantes

-- --

El árbol de búsqueda de un problema también tiene **estados repetidos** cuando hay **caminos redundantes** en el espacio de estados.

Existen siempre que haya más de una forma de llegar de un estado a otro.

El algoritmo de búsqueda en árboles no detecta caminos redundantes, incluso en su versión con detección de ciclos.



Ejemplo

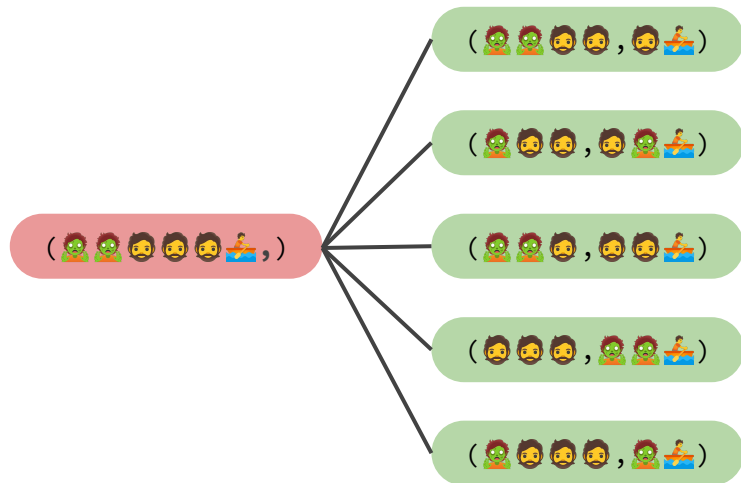
— — —

(🧑🏿🧑🏿🧑🏿🧑🏿🧑🏿🧑🏿🧑🏿,)



Ejemplo

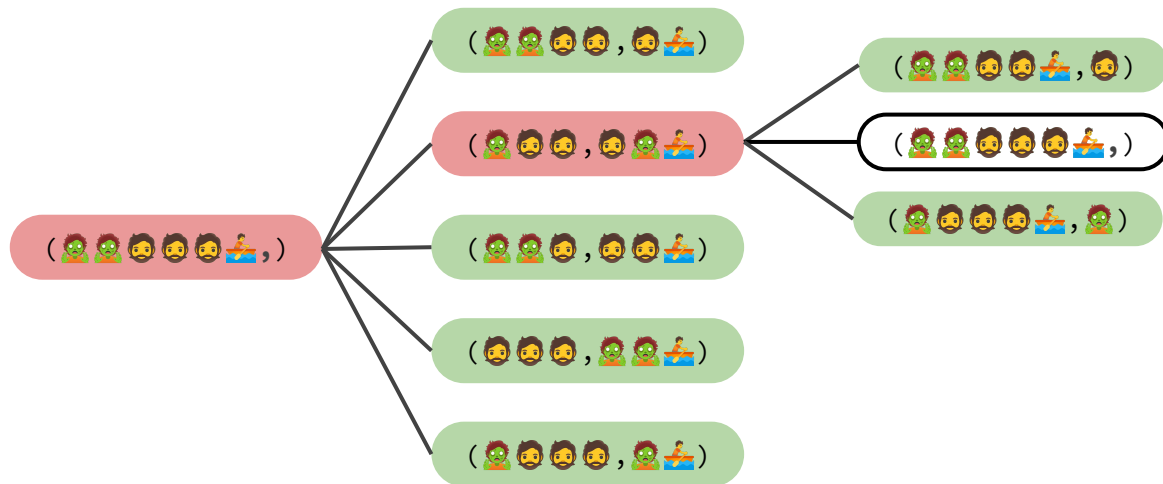
— — —





Ejemplo

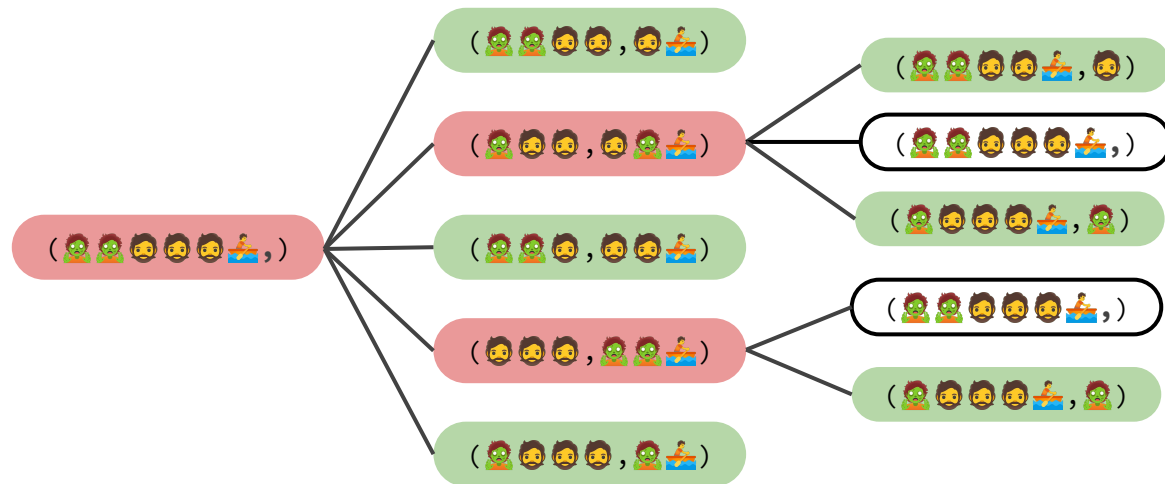
— — —





Ejemplo

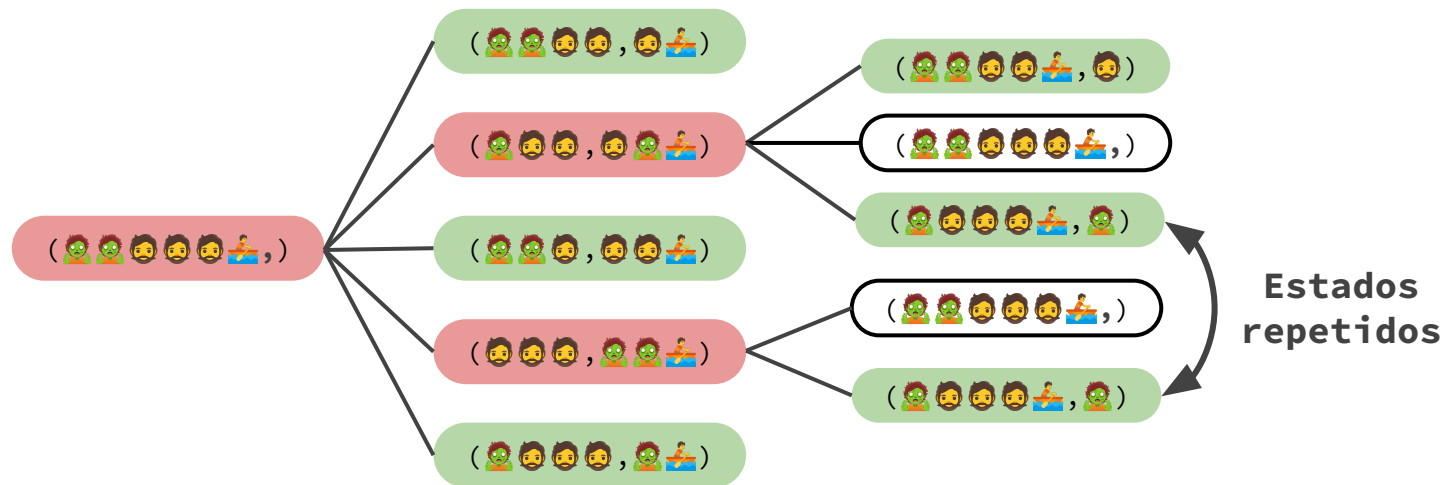
— — —





Ejemplo

— — —





Caminos redundantes

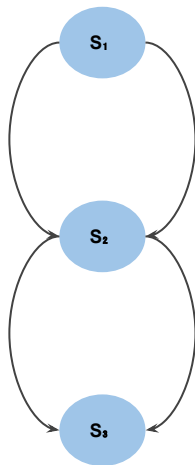
-- --

La **proliferación** de estados repetidos en el árbol de búsqueda puede ser una complicación. **No detectar caminos redundantes puede hacer que problemas tratables, con pocos estados, se vuelvan intratables.**

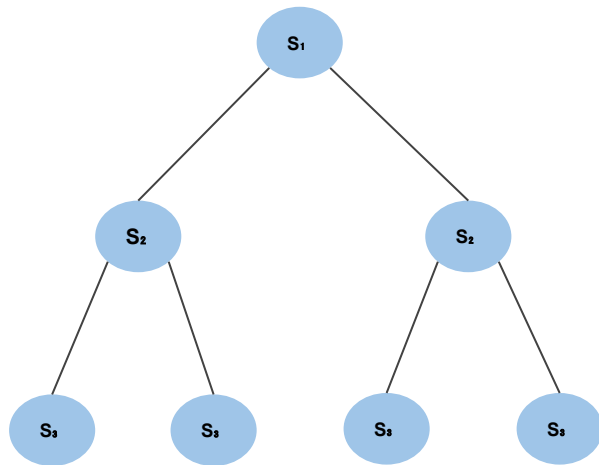
Explosión combinatoria

— — —

Considerar un espacio de estados donde para cada estado S_i hay dos acciones al mismo estado S_{i+1} .



La cantidad de estados a d pasos o menos de S_1 es $d + 1$.

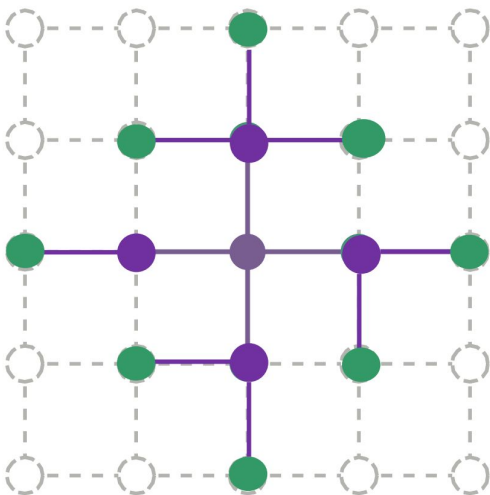


La cantidad de nodos de un árbol de búsqueda con d niveles es $2^{d+1} - 1$.

Explosión combinatoria

— — —

Considerar un espacio de estados con forma de grilla rectangular.



La cantidad de estados a d pasos o menos del estado inicial es:

$$2d^2 + 2d + 1$$

La cantidad de nodos de un árbol de búsqueda con d niveles (sin detectar ciclos) es:

$$1 + 4^1 + \dots + 4^d = (1 - 4^{d+1}) / (1 - 4)$$



Explosión combinatoria

— — —

Para el árbol con
 $2^{d+1} - 1$ nodos:

d	Estados	Nodos	Tiempo
10	11	2047	0,001 seg
20	21	2.097.151	2,1 seg
30	31	$2,5 \times 10^9$	36 min
40	41	$2,2 \times 10^{12}$	25 días
50	51	$2,3 \times 10^{15}$	71 años

Para el árbol con
 $(1 - 4^{d+1}) / (1-4)$ nodos:

d	Estados	Nodos	Tiempo
10	221	1.398.101	1,4 seg
15	481	$1,43 \times 10^9$	24 min
20	841	$1,46 \times 10^{12}$	17 días
25	1301	$1,5 \times 10^{15}$	48 años

1M de nodos generados por segundo



Detección de caminos redundantes

— — —

La forma de evitar caminos redundantes es recordar los estados **alcanzados** en el pasado, es decir, los estados de los nodos ya **expandidos** y de la **frontera**.

“Los algoritmos que olvidan su historia están condenados a repetirla”.

Algoritmo de búsqueda en grafos

El algoritmo de búsqueda en árboles que recuerda los estados alcanzados para no repetirlos se conoce como **algoritmo de búsqueda en grafos**.

El árbol de búsqueda que se genera tiene a lo sumo una copia de cada estado, luego se lo pueden pensar como una búsqueda dentro del propio **grafo** de espacio de estados.

Algoritmo de búsqueda en grafos

Cuando se genera un nodo con un estado ya alcanzado significa que se encontró un camino redundante.

¿Qué hacemos? **La respuesta no es sencilla ya que se pueden omitir soluciones óptimas.**

Veremos un algoritmo general de búsqueda en grafos que siempre desecha el camino redundante recién descubierto. Más adelante veremos otras alternativas.



Algoritmo general de búsqueda en grafos

— — —

```
1 function GRAPH-SEARCH(problema) return solución o fallo
2   generar la raíz con el estado inicial
3   inicializar la frontera con la raíz
4   inicializar un conjunto de estados alcanzados con el estado inicial
5   do
6     if la frontera es vacía then return fallo
7     elegir un nodo y removerlo de la frontera
8     if el nodo contiene un estado objetivo then return solución
9     for all hijo generado al expandir el nodo
10      if el estado del hijo no fue alcanzado then
11        agregar el estado del hijo a los alcanzados
12        agregar el hijo a la frontera
```

Criterios de evaluación para algoritmos de búsqueda

1. **Compleitud.**

¿Encuentra al menos una solución?

2. **Optimalidad.**

¿Encuentra la solución óptima?

3. **Tiempo.**

¿Cuántos nodos expande?

4. **Memoria.**

¿Cuánta memoria usa?

— — —



Resumen

- ❑ El algoritmo general de búsqueda en árboles puede no terminar, incluso si el espacio de estados es finito pero existen caminos cíclicos o redundantes.
- ❑ Evitar caminos cíclicos requiere controlar estados repetidos en el camino de la raíz al nodo actual.
- ❑ Los caminos redundantes pueden hacer que problemas tratables se vuelvan intratables, debido a la explosión combinatoria de nodos repetidos en el árbol de búsqueda.



Resumen

— — —

- ❑ Evitar caminos redundantes requiere aumentar el algoritmo de búsqueda con el conjunto de estados alcanzados.
- ❑ El algoritmo general de búsqueda en grafos evita seguir caminos cíclicos y redundantes, pero su consumo de memoria es mayor.
- ❑ Los algoritmos de búsqueda se comparan considerando 4 factores: completitud, optimalidad, tiempo y memoria.



Próximamente

— — —

Existen diferentes estrategias de búsqueda, es decir, el criterio para elegir el próximo nodo de la frontera a expandir.

Estas estrategias se dividen en dos grandes familias:

- **No-informadas**: sólo tienen acceso a la definición del problema.
- **Informadas**: incorporan una función heurística para guiar la búsqueda.