Busqueda y Resolución de Problemas.

- Espacio de Estados: Formalismo para representar el entorno.

  Ej: Puzle 8

7		8
6	1	2
5	4	3

objetivo →

1	2	3
8		4
7	6	5

- Estados

  matriz 3x3 → 9! = 362880 estados

  9! (factorial)

- Acciones

	0	1	2
0	3	1	8
1		2	7
2	4	5	0

	Acción	Precond.	Coste
Mover ↑	F>0	intercambiar((F,C),(F-1,C))	1
" ↓	F<2	intercambiar((F,C),(F+1,C))	1
" →	C<2	intercambiar(F,C),(F,C+1))	1
" ←	C>0	intercambiar((F,C),(F,C-1))	1

Casilla desplazada

Un espacio de estados viene definido por lo siguiente:

- Conjunto de Estados (E).
  - Todas las situaciones posibles del entorno.
- Conjunto de Acciones (A).
  - Permiten pasar de un estado a otro.

  precondición : E → {true, false}

  efectos : E → E

  Coste (opcional).

- Problema (instancia de problema).
  - Estado inicial →

1	2	
4	3	8
5	6	7

- Condición Objetivo (en caso de no tener un estado objetivo).

  → Imponer una condición para satisfacer mi problema.

  Ej: Suma de la fila 2 sea 7.

· Estrategias Solución

a) Busqueda en grafos abstractos

b) Representación vectorial

c) Representacion basadas en lenguajes lógicos.

a) Est. Búsqueda

- Irrevocables
  Exploran un único camino del grafo
- Tentativos
  Exploran de forma sistemática todos los caminos
  - Con Retroceso
    Guardan unicamente el camino actual
  - Con árbol
    Guardan todos los caminos interesantes generados
    - Amplitud
    - Dijkstra
    - A* → Heurístico

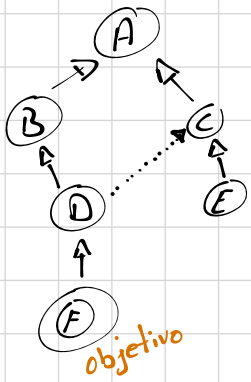
    } A ciegas

· Estructura de Datos

1) Árbol de Búsqueda
   - Con raíz en el nodo de salida.
2) "lista" de nodos abiertos
   - Nodos del árbol por donde puede continuar la búsqueda.

   ↳ Objetivos principales
   - Selección nodo abierto
   - Expansion de un nodo: Calcular y añadir Sus Sucesores al árbol

- Búsqueda en Amplitud
  1. Abiertos ≡ Cola
     Siempre selecciono los nodos más antiguos.
  2. Dos caminos al mismo nodo.
     Elección Arbitraria.
  3. Termina cuando se genera objetivo.



iter.	Abiertos
1	A
2	B, C
3	C, D
4	D, E
5	E, F

- Algoritmo de Dijkstra.
  - Algoritmo de Optimización
  - Cada arco (n, n') en el grafo tiene asociado un coste $c(n, n')$.
  - $\forall (n, n')\ c(n, n') \geq 0$
  - Caminos infinitos = costes infinitos

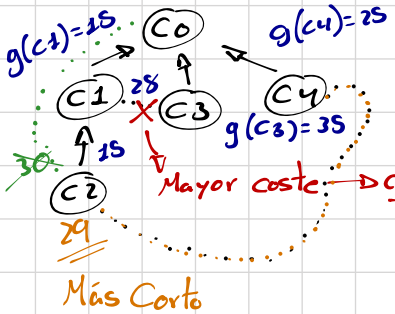
Coste de un camino → Suma de los costes de sus arcos.

$g(n)$ ≡ Coste del camino guardado en el árbol hasta n.

1. Abiertos ≡ Cola de prioridad (por orden creciente).
2. Dos caminos al mismo nodo.
   conservar el de menor $g(n)$.
3. Terminación: cuando se selecciona de Abiertos el objetivo.

	C0	C1	C2	C3	C4
C0		15		35	25
C1	15		15	28	
C2		15			4
C3	35	28			
C4	25		4		

Inicial: C0
Final: C3



iter	Abiertos [n(g(n))]
1	C0 (0)
2	C1(15), C4(25), C3(35)
3	C4(25), C2(30), C3(35)
4	C2(29), C3(35)

inserción en orden creciente

↳ Camino más corto
[C0 → C3]

$f(n) = g(n) + h(n)$

$h^*(n)$ = coste óptimo real de un camino desde n al objetivo.

- **Definición 1:**

Un heurístico $h(n)$ es optimista (admisible) si:

$\forall n,\ h(n) \leq h^*(n)$

Propiedad 1

- Costes positivos: $\forall (n,n') \rightarrow c(n,n') \geq 0$
- $h(n)$ admisible: entonces A^* es admisible en grafos finitos.
  ↳ si hay solución, encuentra la óptima.

Si además el coste de caminos infinitos es infinitos (ej: $\forall (n,n') \rightarrow c(n,n') \geq \varepsilon > 0$), entonces A^* es admisible es grafos infinitos.

cualquier cantidad

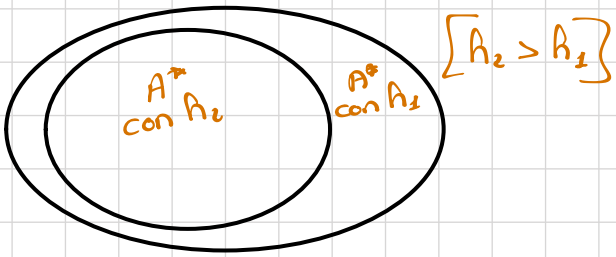
- **Definición 2:**

Dados 2 heurísticos optimistas h_1, h_2 para un mismo problema, decimos que h_2 está más informado que h_1 si:

$\forall n \notin$ objetivo $\rightarrow h_2(n) > h_1(n)$

Propiedad 2:

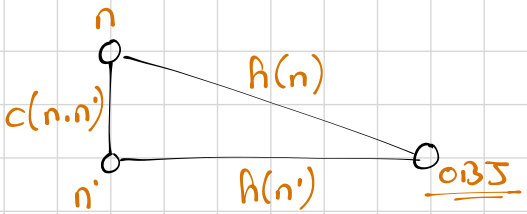
Si h_2 está más informado que h_1, entonces A^* con h_1 expande al menos los mismos nodos que A^* con h_2.



- **Definición 3:**

Un heurístico $h(n)$ cumple la propiedad monótona.

Si para todo arco del grafo (n,n') se cumple:



$\forall \text{arco}(n,n') \rightarrow h(n) \leq c(n,n') + h(n')$

Si se cumple:

$$g(n_1) \leq g(n_2) \leq g(n_3) \leq g(n_4) \leq \ldots . g(n_i) \leq \ldots$$

Propiedad 3:

- Si h(n) es monótono, entonces cuando A* selecciona un nodo n, ha encontrado el camino óptimo hasta él (nunca se volverá a abrir).

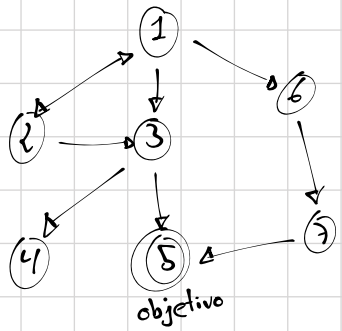
- Si h(n) es monótono, es admisible.
  Si es admisible, PUEDE ser o no monótono.

Nota Curiosa: Si h(n) es monótono, A* es el mejor algoritmo de los de su clase (expande menos nodos que ningún otro).

- Backtrack
  - Propiedades
    1- Completitud: Si hay solución, la encuentra.
    2- Admisibilidad: Si hay solución, encuentra la óptima.
    3- Consumo de Memoria (nodos almacenados) y tiempo (iteraciones ≡ nodos examinados) sobre problemas tipo.

- Backtrack Acotado (e, p)
  - Limito la profundidad del árbol para recorrerlo.
    Si: p > max

- BID (backtrack iterative deepening)



prof. max = 0
1 (prof. máx) = Si)
'fracaso'
prof. max = 1
1
1,2 (No solución)
1 ↩ retroceso
1,3 (No solución)
1 ↩ retroceso
1,6 (No solución)
1 ↩ retroceso (prof. máx? Si)
'fracaso'

prof. max = 2
1
1,2
1,2,1
1,2
1,2,3
1,2
1
1,3
1,3,4
1,3
1,3,5 → Solución

- Propiedades → Si completo, sirve para árboles infinitos
  1) Completo: Si
  2) Admisible: En general no.
     ↳ de un nodo a otro, la profundidad puede ser la misma para todos, pero los costes pueden ser diferentes.

Peor Caso
- Memoria: d+1
- Iteraciones:
  - nº profundizaciones: d+1

$O(b^d)$

Árbol con factor ramificación b sola, prof. d