

CERCA NO-INFORMADA
Resolució de problemes de
cerca i exploració d'alternatives

2. Resolució de problemes de cerca i exploració d'alternatives.

- Introducció.
- Cerca no-informada
- Concepte d'heurística
- Cerca informada
- Cerca local
- Cerca amb adversaris
- Cerca per a satisfacció de restriccions

Arbre semàntic: és una representació de coneixement que definim:

Lèxic: node (node pare, fill, arrel, fulla), arc o branca, nivell d'un arbre, profunditat, factor de ramificació, camí, cost d'una branca, funció heurística.

Estructural:

- Totes les branques uneixen un node pare amb un node fill.
- Tots els nodes fill només ho són d'un sol node pare.
- Els nodes que no són node pare són els nodes fulla.
- Existeix un únic node que no és node fill, és el node arrel.
- Les branques tenen un cost associat.
- El factor de ramificació correspon al número de fills que tenen els nodes.
- Un camí correspon a una seqüència de nodes units per branques des de l'arrel fins a una fulla.
- La profunditat d'un camí ve donada pel número de branques que té.
- La profunditat de l'arbre és la profunditat màxima de tots els camins.
- El cost d'un camí ve donat per la suma dels costos de totes les branques.

Semàntica: Depèn del problema Node= Estat d'un problema
Branca= Canvi d'estat

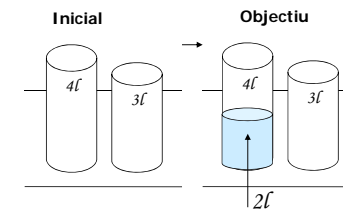
Procedimental:

- Proc. que genera tots els nodes fill d'un node donat (expandir).
- Proc. que recorren arbres amb una estratègia concreta.
- Proc. que gestionen llistes, piles, cues, llistes ordenades, etc.
- Proc. que eliminen camins amb cicles.
- Proc. que avaluen nodes.

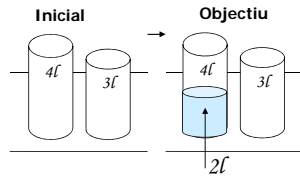
Exemple: el problema de les gerres d'aigua

Enunciat:

"Donades dues gerres d'aigua sense marques per poder fer mesures, sabem que una gerra es de 4 litres i l'altre de 3 litres. Tenim una aixeta per omplir les gerres d'aigua i una aigüera per llençar aigua que sobri. L'objectiu és definir la seqüència de passos que s'han de fer per aconseguir tenir 2 litres d'aigua a la gerra de 4 litres."



Exemple: el problema de les gerres d'aigua



• Estat:

$(G4, G3)$, on $G4$ i $G3$ són variables

Estat inicial: $(G4=0, G3=0)$

Estat final: $(G4=2, G3=0)$

• Canvis d'estat:

Emplenar(G4): $(G4=X, G3=Y) \rightarrow (G4=4, G3=Y)$

Emplenar(G3): $(G4=X, G3=Y) \rightarrow (G4=X, G3=3)$

Buidar(G4): $(G4=X, G3=Y) \rightarrow (G4=0, G3=Y)$

Buidar(G3): $(G4=X, G3=Y) \rightarrow (G4=X, G3=0)$

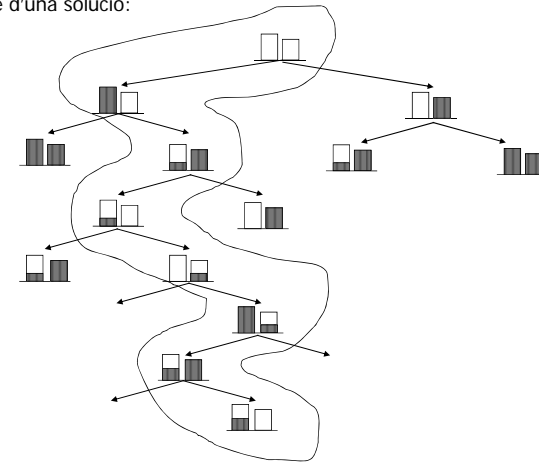
Traspasar(G4->G3): Si $(G4 \leq (3-G3)) \rightarrow (G4=0, G3=G3+G4)$

Traspasar(G4->G3): Si $(G4 > (3-G3)) \rightarrow (G4=G4-(3-G3), G3=3)$

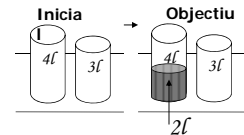
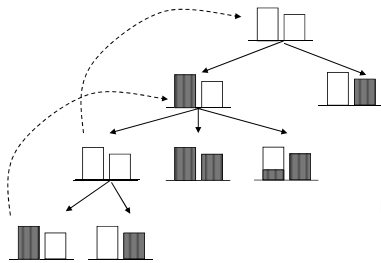
Traspasar(G3->G4): Si $(G3 \leq (4-G4)) \rightarrow (G4=G4+G3, G3=0)$

Traspasar(G3->G4): Si $(G3 > (4-G4)) \rightarrow (G4=4, G3=G3-(4-G4))$

• Exemple d'una solució:



EVITAR CAMINS AMB CICLES



EliminarCicles(L): comprova per a cada camí de la llista L, que el primer node del camí no hi és a la cua del mateix camí.

En llenguatge Lisp, la funció EliminarCicles:

`(DEFUN EliminarCicles (L) (REMOVE-IF (LAMBDA (c) (MEMBER (CAR c) (CDR c))) L))`

Procediments que fan una cerca: cal considerar dos punts essencials

- Generació del fills d'un node donat (depèn del problema)
- Estratègia o ordre en la generació

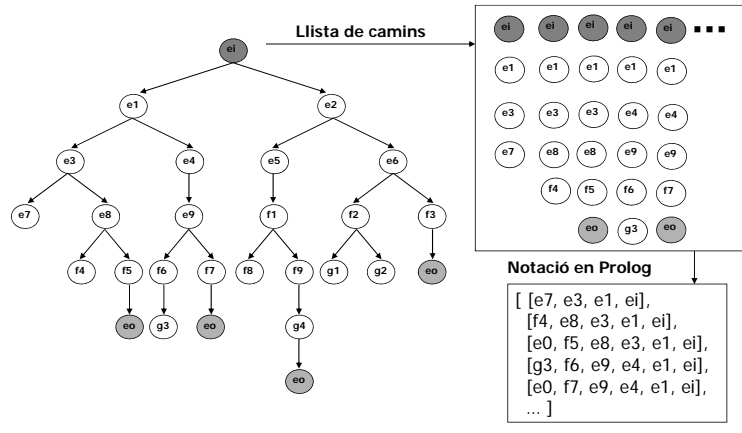
Consideració prèvia: Evitar estat repetits.

CERCA NO INFORMADA

- **Cerca en profunditat:** expandeix primer els camins més profunds.
- **Cerca en profunditat limitada:** expandeix primer els camins més profunds fins a un límit prefixat.
- **Cerca en amplada:** expandeix primer els camins menys profunds.
- **Cerca de cost uniforme:** expandeix primer els camins amb menys cost acumulat.

Algorisme de Cerca

Idea: Mantenir tots els camins d'un arbre en una llista de camins.



Funció CERCA (NodeArrel, NodeObjectiu)

1. Llista = [[NodeArrel]];
 2. **Fins que** (Cap(Cap(Llista))=NodeObjectiu O bé (Llista=NIL) **fer**
 - a) C=Cap(Llista);
 - b) E=Expandir(C);
 - c) E=EliminarCicles(E);
 - d) Llista=Insertar(E,Cua(Llista));
 3. **Fins que**;
 4. **Si** (Llista<>NIL) **Retornar**(Cap(Llista));
 5. **Sinó Retornar**("No existeix Solucio");
- Funcio**

Cap(L): Retorna el primer element de la llista L

Expandir(C): Retorna la llista dels camins resultants d'expandir el node de més profunditat del camí C.

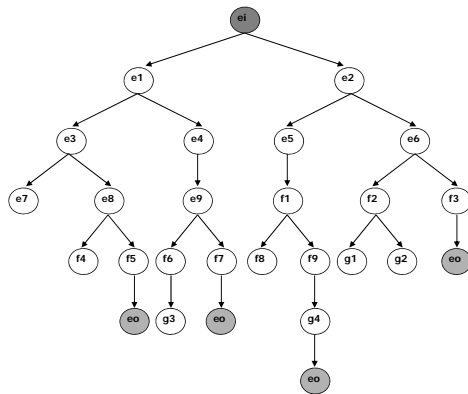
EliminarCicles(E): Retorna els camins de la llista E que no tenen cicles.

Insertar(X,L): Retorna la llista L, després d'insertar-hi la llista d'elements, X.

Exemple de l'evolució de l'algorisme CERCA:

Suposem un problema que requereix construir el següent arbre sense cicles:

ei: Estat Inicial
eo: Estat objectiu



Funció CERCA_profunditat (NodeArrel, NodeObjectiu)

1. Llista = [[NodeArrel]];
 2. **Fins que** (Cap(Cap(Llista))=NodeObjectiu O bé (Llista=NIL) **fer**
 - a) C=Cap(Llista);
 - b) E=Expandir(C);
 - c) E=EliminarCicles(E);
 - d) Llista=Insertar_davant(E,Cua(Llista));
 3. **Fins que**;
 4. **Si** (Llista<>NIL) **Retornar**(Cap(Llista));
 5. **Sinó Retornar**("No existeix Solucio");
- Funcio**

Funció CERCA_amplada (NodeArrel, NodeObjectiu)

1. Llista = [[NodeArrel]];
 2. **Fins que** (Cap(Cap(Llista))=NodeObjectiu O bé (Llista=NIL) **fer**
 - a) C=Cap(Llista);
 - b) E=Expandir(C);
 - c) E=EliminarCicles(E);
 - d) Llista=Insertar_darrera(E,Cua(Llista));
 3. **Fins que**;
 4. **Si** (Llista<>NIL) **Retornar**(Cap(Llista));
 5. **Sinó Retornar**("No existeix Solucio");
- Funcio**

Funció CERCA_cost_uniforme (NodeArrel, NodeObjectiu)

1. Llista=[[NodeArrel]];
 2. Fins que (Cap(Cap(Llista))=NodeObjectiu O bé (Llista=NIL) fer
 - a) C=Cap(Llista);
 - b) E=Expandir(C);
 - c) E=EliminarCicles(E);
 - d) Llista=Inserció_ordenada_cost(E,Cua(Llista));
 3. Ffinsque;
 4. Si (Llista<>NIL) Retornar(Cap(Llista));
 5. Sinó Retornar("No existeix Solucio");
- Ffuncio

Inserció_ordenada_cost(E,L): Inserta els elements de la llista E a la llista L, la inserció es fa de manera que els camins quedin ordenats de menor a major cost.

Funció CERCA_prof_limitada (CamiActual, NodeObjectiu,prof_max)

1. Si (Cap(CamiActual)=NodeObjectiu) llavors Retornar(CamiActual);
2. Sinó Si (prof_max=0) llavors Retornar("Prof. màxima"); Fsi
3. Sinó
 - a) E=Expandir(C);
 - b) E=EliminarCicles(E);
 - c) Per a cada (cami c de E) fer
 - i. Resultat=**CERCA_prof_limitada**(c,NodeObjectiu,prof_max-1)
 - ii. Si (Resultat<>"Prof. màxima") llavors Retornar(Resultat); Fsi
 - d) Fper
4. Fsi

Ffuncio

	Gestió de la llista de camins
CERCA	Llista
CERCA EN AMPLADA	Cua
CERCA EN PROFUNDITAT	Pila
CERCA DE COST UNIFORME	Llista ordenada per cost de camí
CERCA EN PROFUNDITAT LIMITADA	(Sense llista) Solució recursiva

Anàlisi d'un algorisme de cerca:

Completesa: un algorisme és complet si garanteix que si existeix la solució sempre la troba.

Optimalitat: un algorisme és òptim si garanteix que la solució que troba és òptima en algun sentit.

Complexitat en temps: ve donada pel número de nodes que l'algorisme necessita obrir per trobar la solució.

Complexitat en espai: ve donada pel número de nodes que l'algorisme necessita tenir alhora a memòria .

Anàlisi dels algorismes de cerca:

Crítteri	Cerca en amplada	Cerca en profunditat	Cerca de cost uniforme	Cerca en profunditat limitada
Complet	Si (si b finita)	Si (si eliminem cicles)	Si (si b finita i costos positius)	No
Òptim	Si (si òptim=més curt)	No	Si	No
Temps	$O(b^{d+1})$	$O(b^m)$	$O(b^m)$	$O(b^k)$
Espai	$O(b^{d+1})$	$O(bm)$	$O(b^m)$	$O(bk)$

b: factor de ramificació.
d: profunditat de la solució menys profunda.
m: profunditat màxima de l'arbre.
k: límit imposat a la profunditat a la cerca en profunditat limitada

