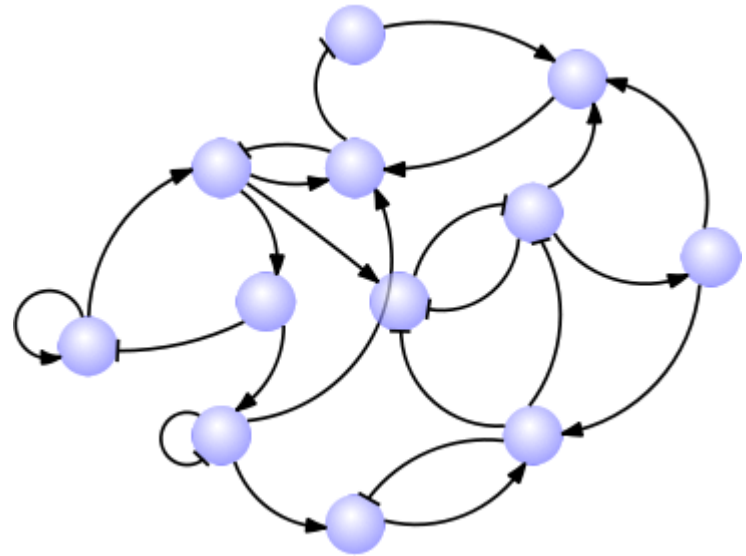
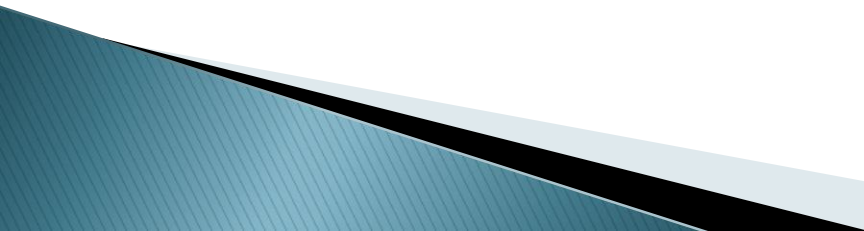


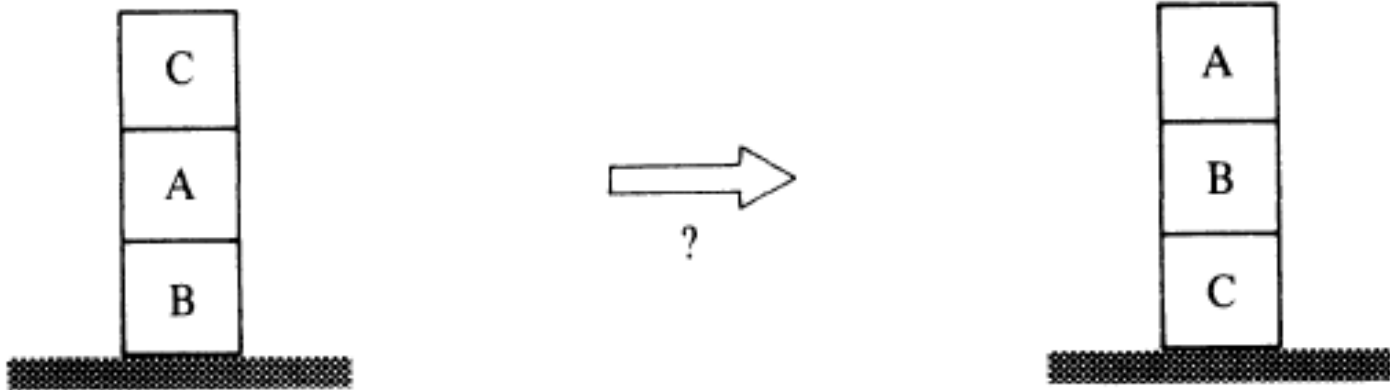
Solución de Problemas Mediante Búsqueda en un Espacio de Estados



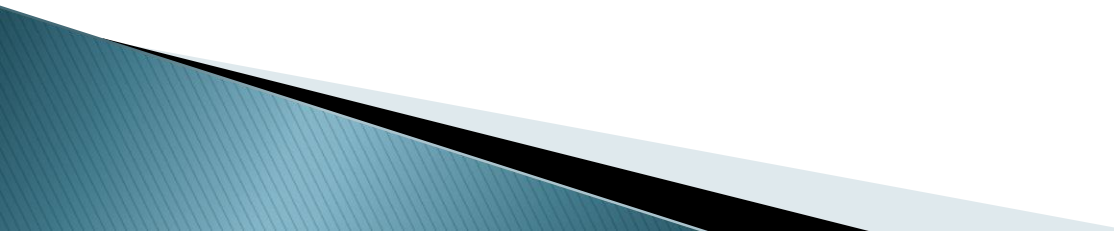
Espacio de Estados

- ▶ Una gran cantidad de problemas pueden ser representados por un espacio de estados.
 - ▶ Cada estado es una situación que puede presentarse.
 - ▶ El espacio de estados se representa con un grafo dirigido en el cual los nodos representan los posibles estados y los arcos representan las transiciones entre estados.
 - ▶ Las transiciones de una situación a otra dependen de un conjunto de reglas propias del problema.
- 

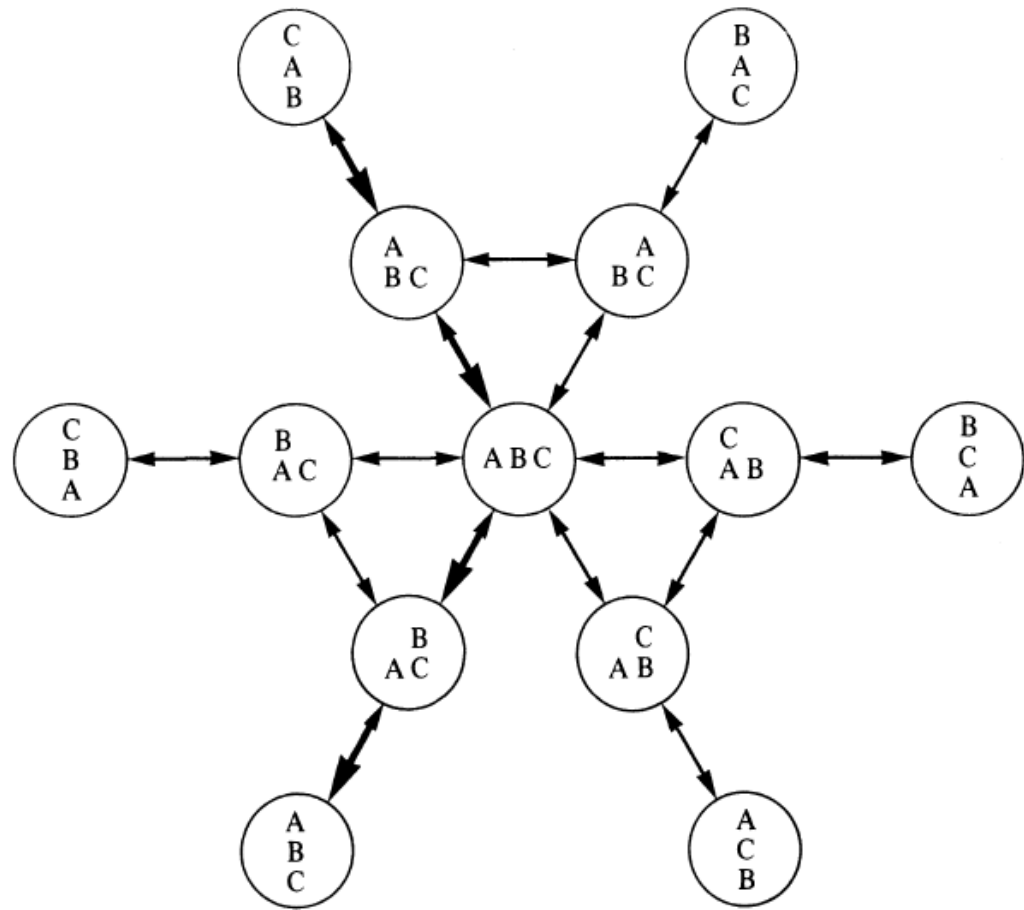
Ejemplo: Mundo de los bloques



Ejemplo: Mundo de los bloques (blocks world)

- ▶ Consiste en tres bloques sobre una mesa
 - ▶ Un bloque puede estar encima de otro
 - ▶ Solo se puede mover un bloque a la vez
 - ▶ Solo se puede mover un bloque cuando no tiene otro encima
 - ▶ Se puede poner un bloque sobre la mesa o sobre otro bloque
 - ▶ Existe una situación inicial y una situación final
- 

Espacio de Estados del Mundo de los Bloques



Representación del mundo de los bloques en PROLOG

$[[c,a,b], [], []] \rightarrow [[a,b,c], [], []]$
 $[[], [a,b,c], []]$
 $[[], [], [a,b,c]]$

Situación inicial

Objetivo (situación final)

Representación del mundo de los bloques en PROLOG

- ▶ Relación sucesor (s/2) que dada una situación del problema, regresa una nueva situación
- ▶ s(Situación1,Situación2).

```
s ([[H|T],B,C], [T, [H|B],C]) .  
s ([[H|T],B,C], [T,B, [H|C]]) .  
s ([A, [H|T],C], [[H|A],T,C]) .  
s ([A, [H|T],C], [A,T, [H|C]]) .  
s ([A,B, [H|T]], [[H|A],B,T]) .  
s ([A,B, [H|T]], [A, [H|B],T]) .
```

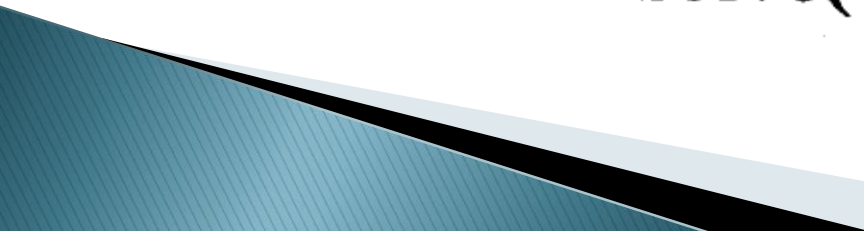
Representación del mundo de los bloques en PROLOG

Predicado para determinar si ya se
llegó a la solución

**goal(Situation) :-
member([a,b,c], Situation).**

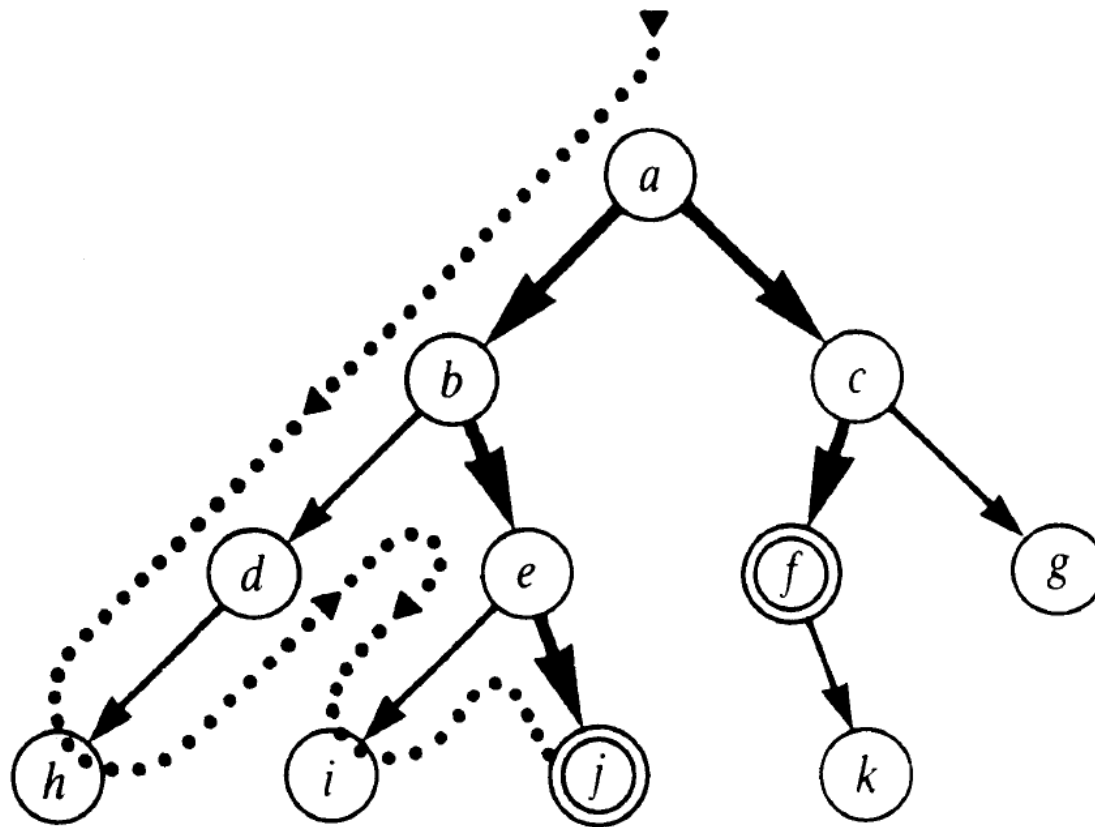
Predicado para encontrar
la solución

solve(Start, Solution)



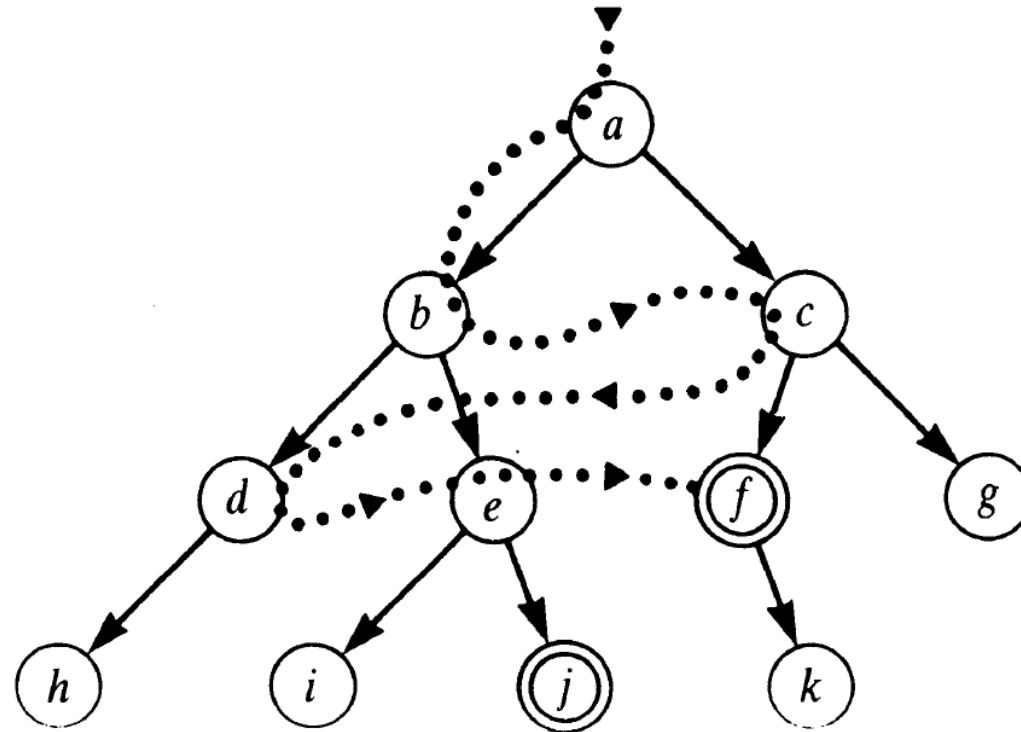
Búsquedas

- ▶ En profundidad (depth-first):



Búsquedas

- ▶ En anchura (breadth-first)



Búsqueda por profundidad

- ▶ Se inicia con el nodo raíz
- ▶ Si es una solución, se termina la búsqueda
- ▶ Si no, se expande el nodo (se obtienen sus hijos) y se realiza la búsqueda recursivamente por cada uno

Implementación de la búsqueda en profundidad en PROLOG

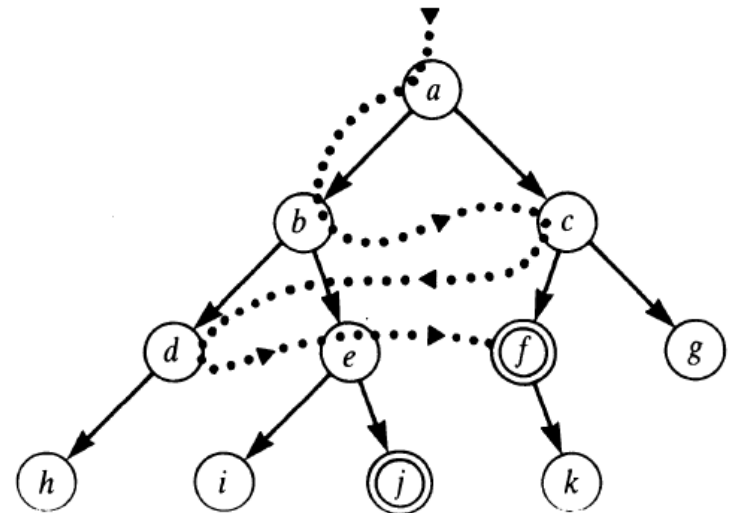
```
solve( Node, Solution) :-  
    depthfirst( [], Node, Solution).
```

```
depthfirst( Path, Node, [Node | Path] ) :-  
    goal( Node).
```

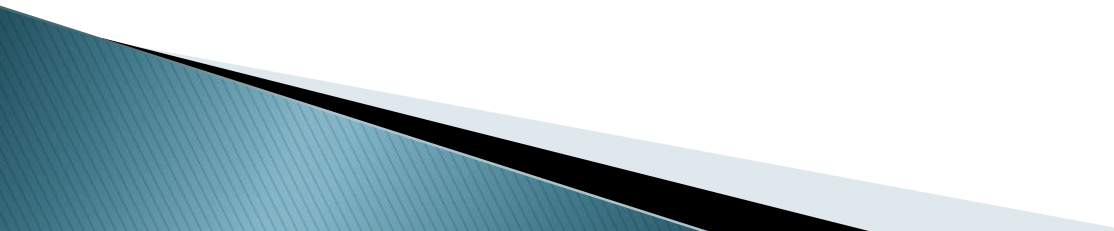
```
depthfirst( Path, Node, Sol) :-  
    s( Node, Node1),  
    not member( Node1, Path),  
    depthfirst( [Node | Path], Node1, Sol).
```

Búsqueda en Anchura

- ▶ Se procesan primero los nodos que están más cerca de la raíz
- ▶ Es más difícil de implementar que la búsqueda en profundidad pues hay que mantener un conjunto de soluciones (caminos) candidatas

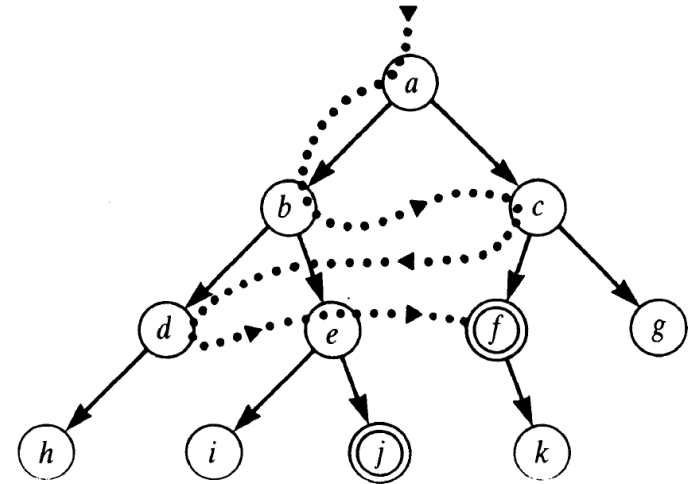


Implementación de Búsqueda en Anchura

- ▶ Se mantiene una lista de soluciones (caminos)
 - ▶ Si la cabeza de la primera solución es un nodo final, entonces se regresa ese camino
 - ▶ Si no, se elimina la primera solución de la lista, se generan todas las posibles extensiones de un paso de esta solución y se agregan al final de la lista de soluciones
 - ▶ Se ejecuta recursivamente la búsqueda con la nueva lista.
- 

Ejemplo

- ▶ Lista de soluciones:
- ▶ $[[a]]$ (se inicia con la raíz)
- ▶ $[[b,a],[c,a]]$ (se expande)
- ▶ $[[c,a],[d,b,a],[e,b,a]]$
- ▶ $[[d,b,a],[e,b,a], [f,c,a],[g,c,a]]$
- ▶ $[[e,b,a], [f,c,a],[g,c,a],[h,d,b,a]]$
- ▶ $[[f,c,a],[g,c,a],[h,d,b,a] [i,e,b,a], [j,e,b,a]]$ (Aquí acaba la búsqueda pues f es un nodo final)



Implementación de búsqueda en Anchura en Prolog

```
solve2(Start,Solution):-  
    breadthfirst([[Start]],Solution).  
breadthfirst([[Node|Path]|_],[Node|Path]):-  
    goal(Node).  
breadthfirst([[N|Path]|Paths],Solution):-  
    bagof([M,N|Path],(s(N,M),not(member(M,[N|Path]))),NewPaths),  
    concatena(Paths,NewPaths,Paths1),!,  
    breadthfirst(Paths1,Solution);  
    breadthfirst(Paths,Solution).
```