Blind Search Method

BLIND SEARCH STRATEGIES

- No information on the number of intermediate states or path cost to move from the current situation to the goal.
- Only if we are able to distinguish the target from the state or not.
- This search is also called uninformed search.

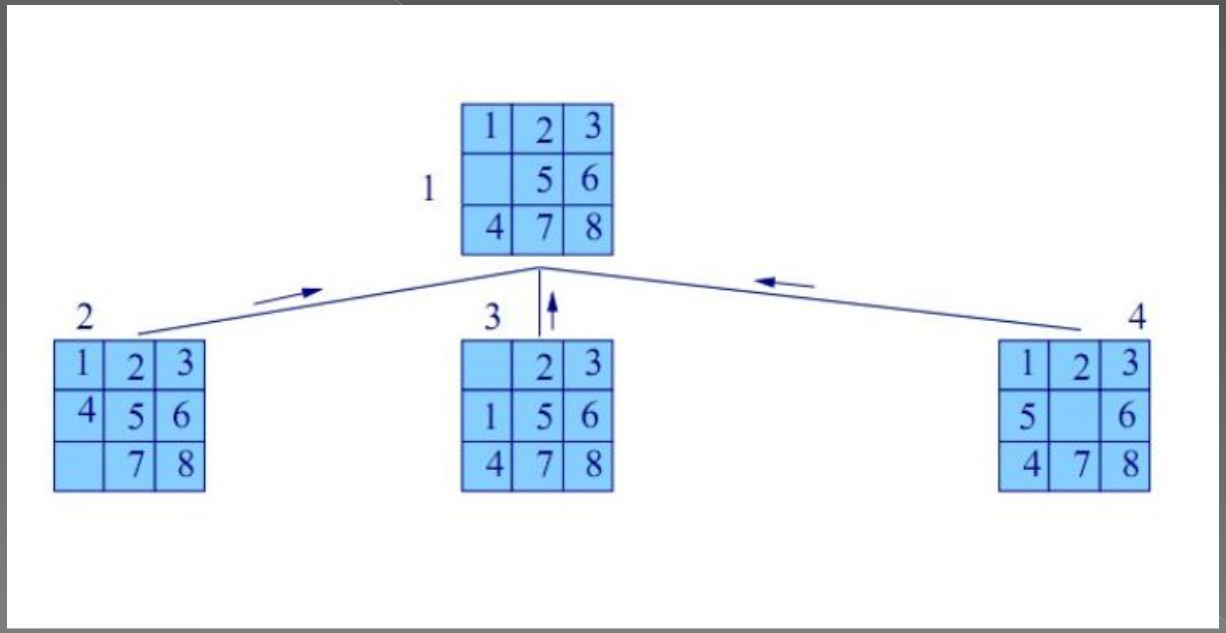
BLIND SEARCH METHODS

- Search preferential amplitude
- Preferred search depth
- Uniform cost search
- Search limited depth
- Iterative deepening search

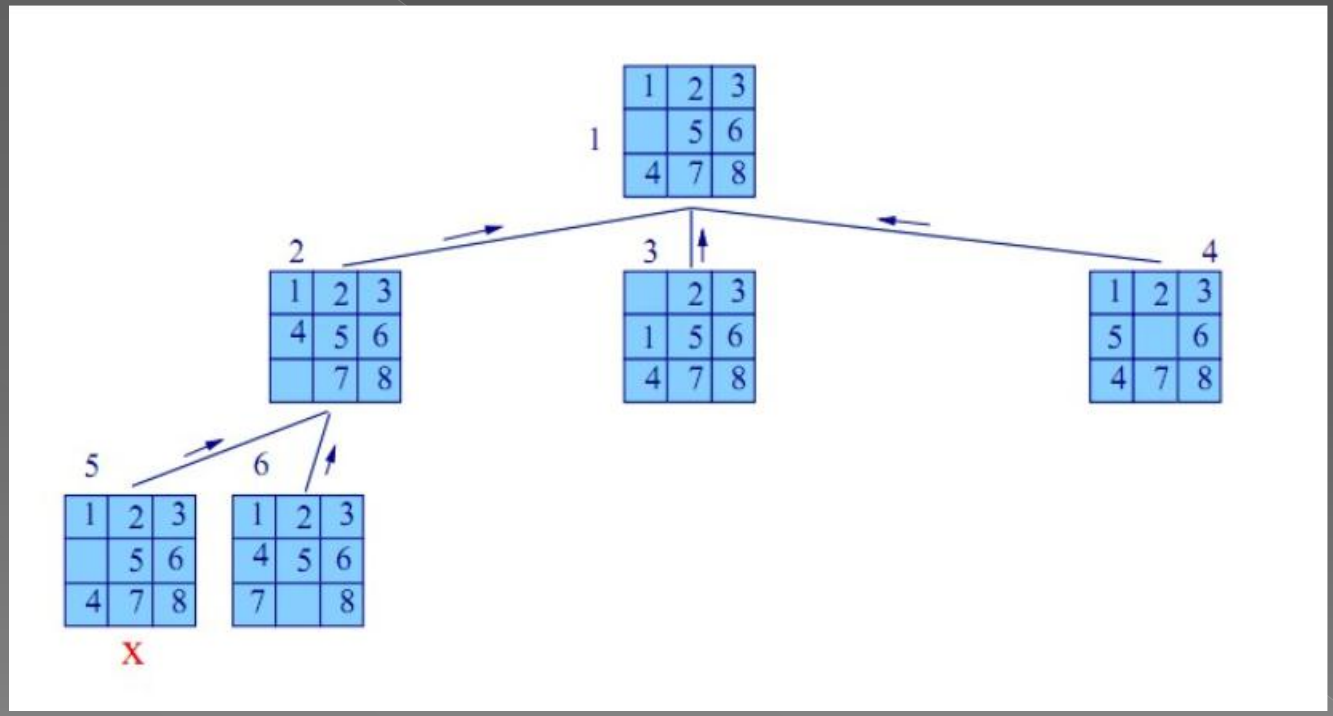
SEARCH RANGE

- The search in amplitude is to tour the tree by levels, ie, the parent goes first, then the children of this (from left to right), then the grandchildren ... and-so on until you find the desired item. For our algorithm we have defined the roles of father and High Definition High to return (if any), respectively, node-parent, left child and right child. These functions have been used in the main algorithm is iterative bpa, and consists of comparing the input elements until the list is vacÃa. First check if the parent is the found item and does the same with the children of this, then sends the list to evaluate all children of the left and right children and they all do the same.

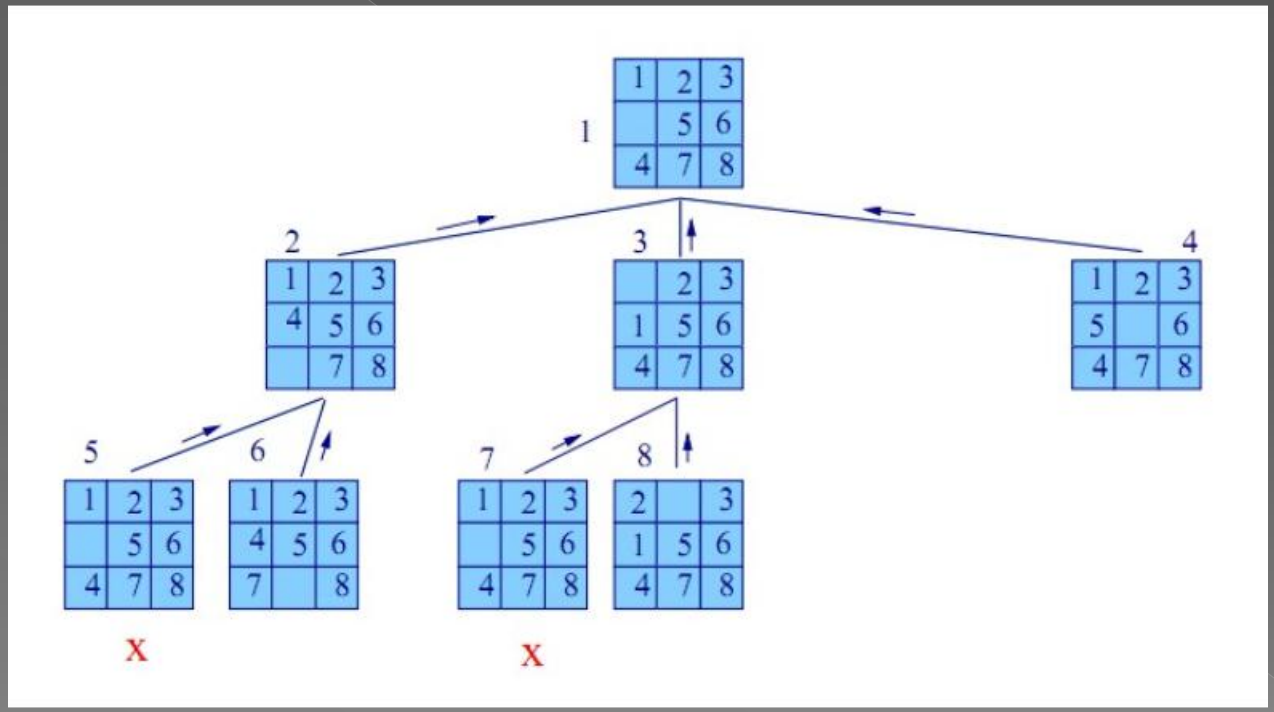
Breadth-first search (WIDTH)



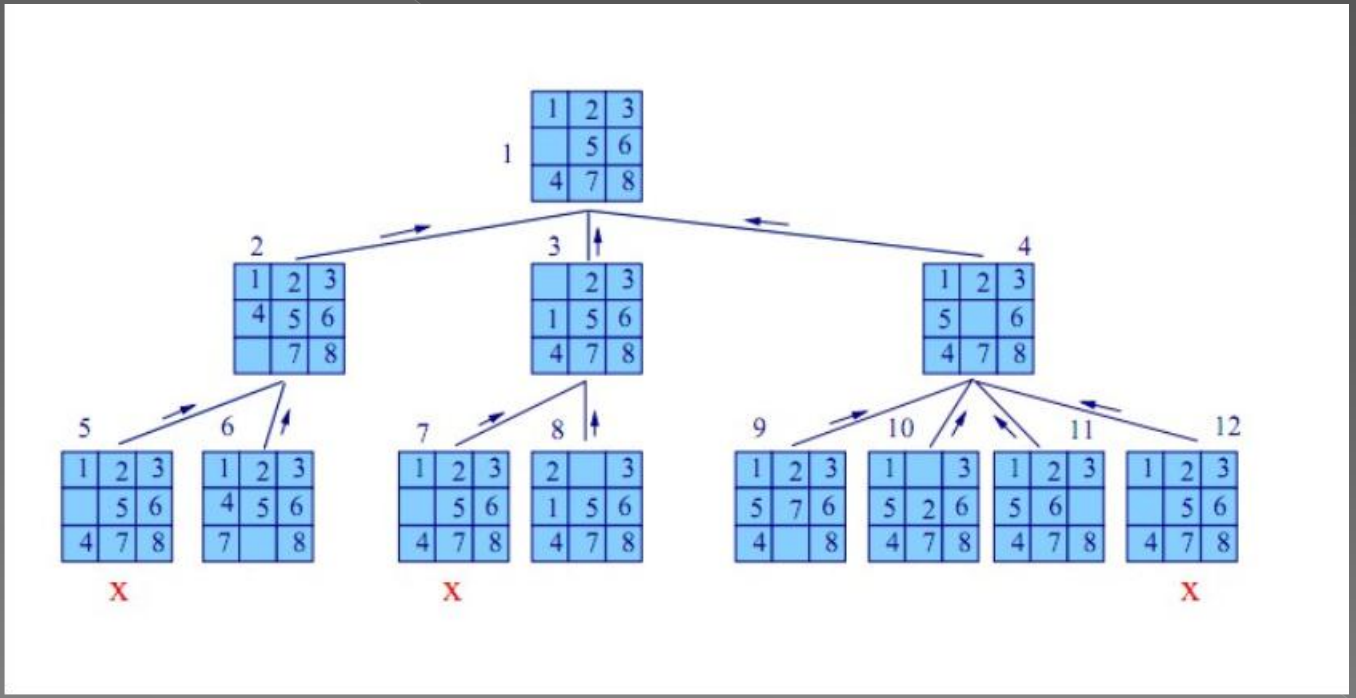
Breadth-first search



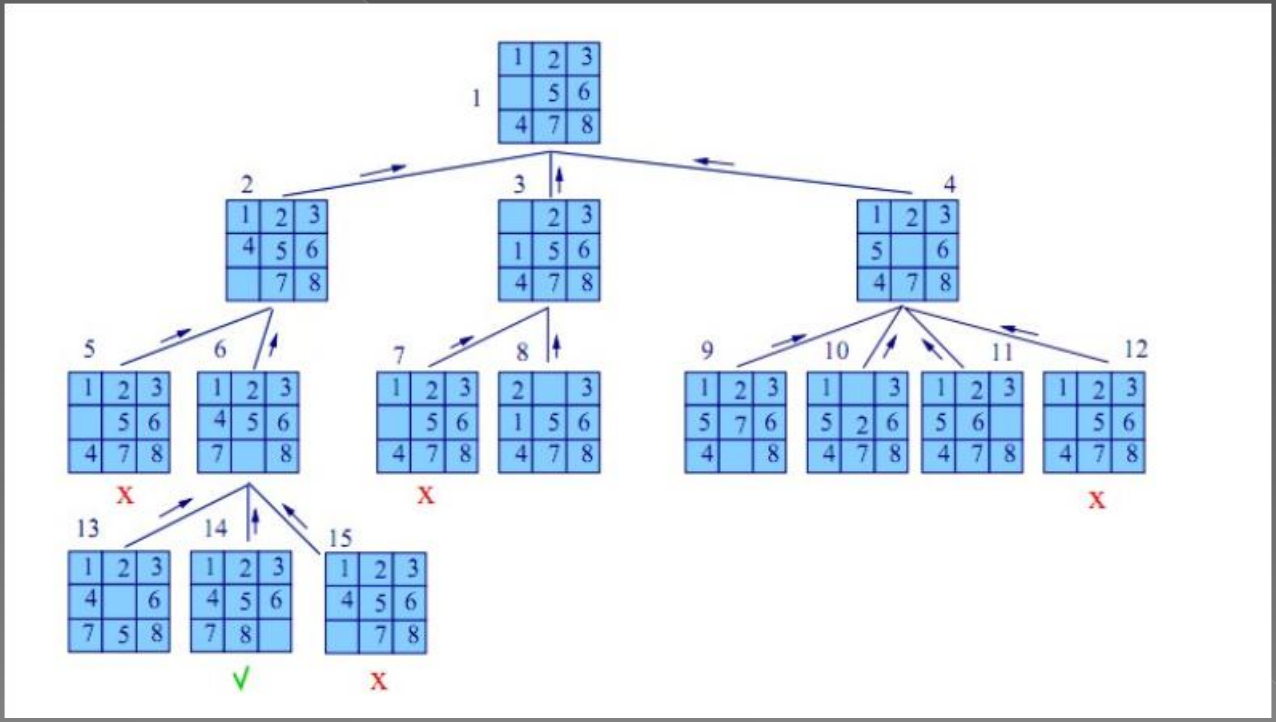
Breadth-first search



Breadth-first search



Breadth-first search



SEARCH RANGE ALGORITHM

- CL-USER 1> (defun bpa(elm lst)
- (cond ((null lst) nil)((equal nil(padre lst))nil)
- (t(if(equal elm(padre lst))t
- (if (equal elm (hi lst)) t
- (if (equal elm (hd lst)) t
- (or (bpa elm (second lst)) (bpa elm(third lst)))))))))
- BPA
- CL-USER 2> (defun padre(lst))
- (if (atom lst) nil
- (first lst)))
- PADRE
- CL-USER 3> (defun hd(lst)
- (if (equal nil (third lst)) nil
- (if (atom (third lst)) (third lst)
- (first (third lst)))))
- HD

SEARCH RANGE ALGORITHM

- CL-USER 4> (defun hi(lst)
- (if (equal nil (second lst)) nil
- (if (atom (second lst)) (second lst)
- (first (second lst)))))
- HI

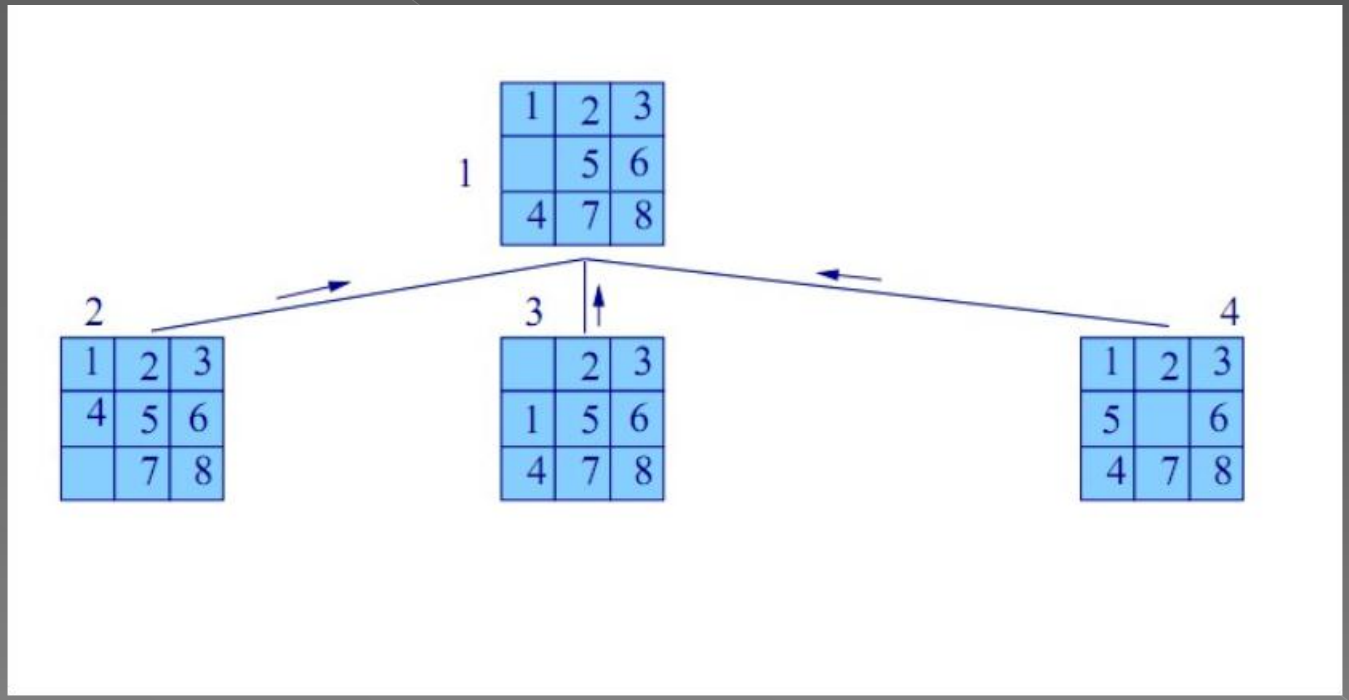
- CL-USER 5> bpa ‘e’(a(b c d)(e f))
- T

- CL-USER 6> bpa ‘d’(a(b c d)(e f))
- T

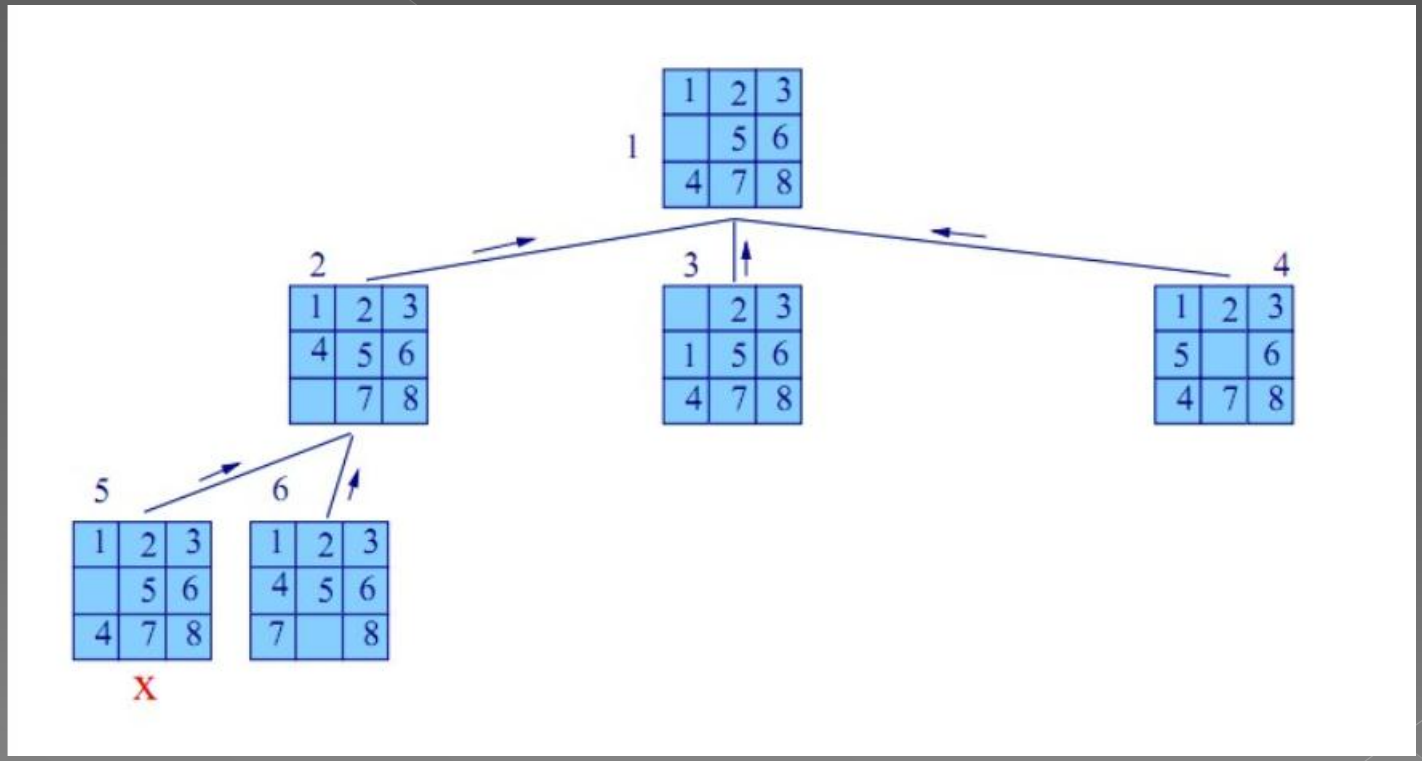
SEARCH IN DEPTH

- The search depth is trava © s of the nodes of a tree as follows: Father - Son left (recursively until the Ã ° last) â € "right child, ie, first see if the" father "is the desired item, then see your "left child" and Ã ° last his â € œhijo right ", recursively for each of your children is telling the girl ± os to trava © s of all nia ± I left and then right until you find the desired item. in our algorithm, when a tree is expressed in the list, all the nodes is n entering pre-order (left Der Father-HH), then sa ³ I have to go from element to element "list" (tree) enters ³ to find the item. we define a main function "Search", which takes two parameters, the first element to looking and looking seconds for the Ã tree where the topic. then goes to the function n "Busq_Prof" that takes three parameters, the first element to search, the second the first (Ã volume or list) of trees and finally the rest of the tree, and recursively loop of search for the article.

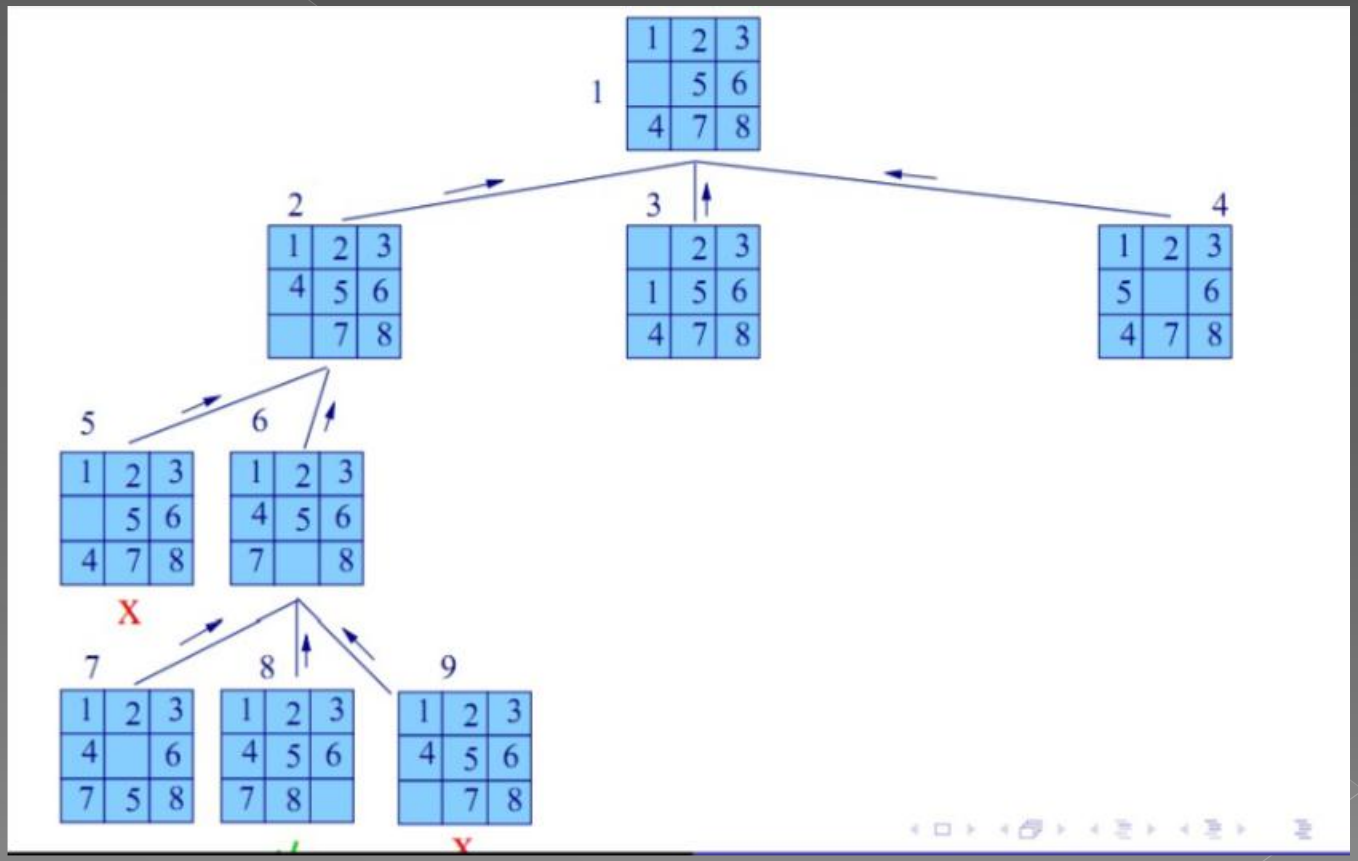
SEARCH IN DEPTH



SEARCH IN DEPTH



SEARCH IN DEPTH



DEEP SEARCH ALGORITHM

- CL-USER 1> (defun Busq_Porf(n P R)
- (if (atom P) (if (null P) (if (null R) 100 (Busq_Prof n (first R)(rest R)))
- (if (equal n P) 1 (if (null R) 100 (+ 1 (Busq_Prof n (first R)(rest R))))))
- (Busq_Prof n (first P)(if (null(rest P)) R (cons (rest P) R)))))
- BUSQ_PROF

- CL-USER 2> (defun Buscar(x A)
- (setq Canti_Nod (Busq_Prof x (first A)(rest A)))
- (if (> Canti_Nod 100)(format t «No se encontro el elemento ~d» x)
- (format t «Se recorrio ~d nodos para habllar el elemento ~d» Canti_Nod x)))
- BUSCAR

- CL-USER 3> Buscar '4'(1(2 4 5)(3 6))
- Se recorrió 3 nodos para hallar el elemento 4

- CL-USER 4> Buscar '10'(1(2 4 5)(3 6))
- No se encontró el elemento 10