BABEŞ-BOLYAI UNIVERSITY
Faculty of Computer Science and Mathematics

ARTIFICIAL INTELLIGENCE

Solving search problems

Uninformed search strategies

Topics

A. Short introduction in Artificial Intelligence (AI)

A. Solving search problems
   A. Definition of search problems
   B. Search strategies
      A. Uninformed search strategies
      B. Informed search strategies
      C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
      D. Adversarial search strategies

C. Intelligent systems
   A. Rule-based systems in certain environments
   B. Rule-based systems in uncertain environments (Bayes, Fuzzy)
   C. Learning systems
      A. Decision Trees
      B. Artificial Neural Networks
      C. Support Vector Machines
      - Evolutionary algorithms
   D. Hybrid systems

Content

- Problems
- Problem solving
  - Steps of problem solving
- Solving problem by search
  - Steps of solving problem by search
  - Search strategies

Useful information

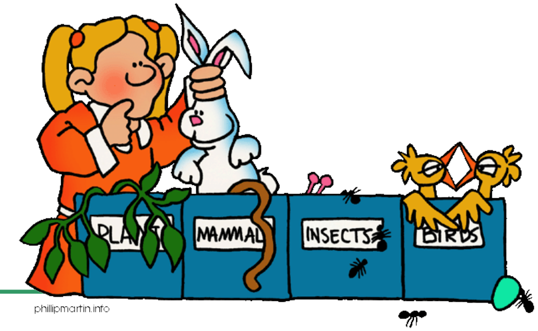
- Chapters I.1, I.2 şi II.3 of *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- Chapters 1 and 2 of *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Chapters 2.1 – 2.4 of

http://www-g.eng.cam.ac.uk/mmg/teaching/artificialintelligence/

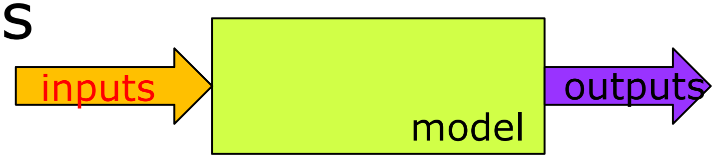
Problems

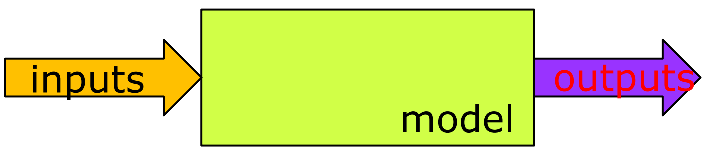
- Two problem types:
  - Solving in a deterministic manner
    - Computing the sinus of an angle or the square root of a number
  - Solving in a stochastic manner
    - Real-world problems → design of ABS
    - Involve the search of a solution → AI’s methods

Problems



- Tipology
  - Search/optimization problems
    - Planning, satellite's design
  - Modeling problems
    - Predictions, classifications
  - Simulation problems
    - Game theory



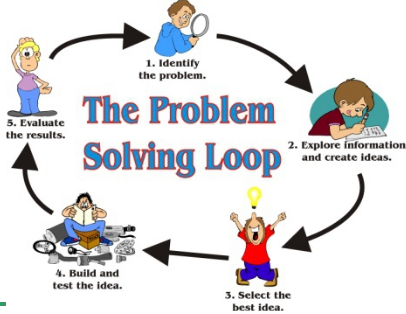


Problem solving

- Identification of a solution
  - In computer science (AI) → search process
  - In engineering and mathematics → optimisation process

- How?
  - Representation of (partial) solutions → points in the search space
  - Design of a search operators → map a potential solution into another one

Steps in problem solving

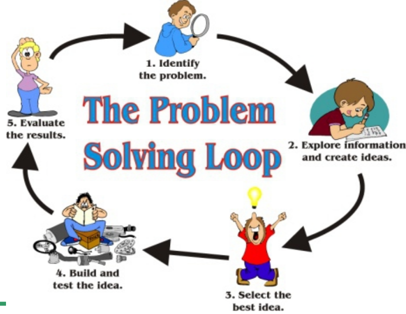


- Problem definition
- Problem analyses
- Selection of a solving technique
  - Search
  - Knowledge representation
  - Abstract methods

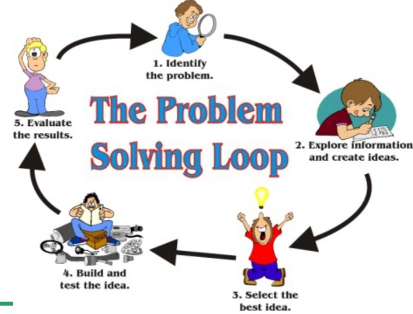
Solving problems by search

- Based on some objectives
- Composed by actions that accomplish the objectives
  - Each action changes a state of the problem
- More actions that map the initial state of problem into a final state

Steps in solving problems by search
Problem definition



- Problem definition involves:
  - A search space
    - All possible states
    - Representation
      - Explicit – construction of all possible states
      - Default – by using some data structures and some functions (operators)
  - One or more initial state
  - One or more final states
  - One or more paths
    - More successive states
  - A set of rules (actions)
    - Successor functions (operators) – next state after a given one
    - Cost functions that evaluate
      - How a state is mapped into another state
      - An entire path
    - Objective functions that check if a state is final or not



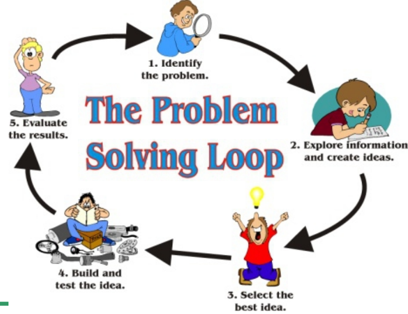
Steps in solving problems by search
Problem definition

- Examples
  - Puzzle game with 8 pieces
    - State’s space – different board configurations for a game with 8 pieces
    - Initial state – a random configuration
    - Final state – a configuration where all the pieces are sorted in a given manner
    - Rules -> white moves
      - conditions: move inside the table
      - Transformations: the white space is moved up, down, to left or to right
    - Solution - optimal sequence of white moves

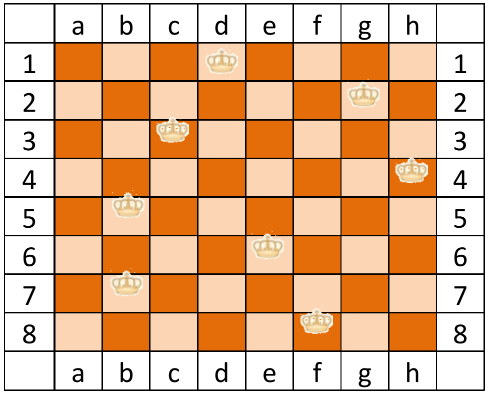
7	2	1
	5	6
3	8	4

1	2	3
4	5	6
7	8	

Steps in solving problems by search
Problem definition

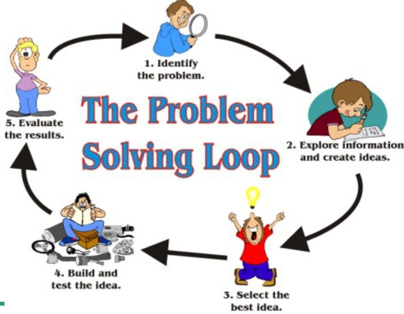


- Examples
  - Queen’s problem
    - State’s space – different board configurations for a game with n queens



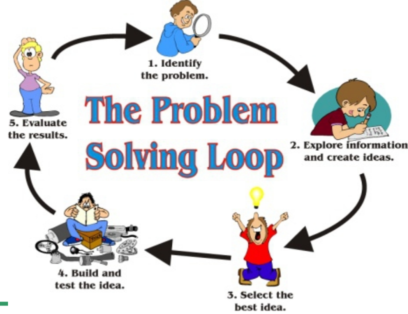
    - Initial state – a configuration without queens
    - Final state – a configuration n queens so that none of them can hit any other in one move
    - Rules -> put a queen on the table
      - conditions: the queen is not hit by any other queen
      - Transformations: put a new queen in a free cell of the table
    - Solution - optimal placement of queens

Steps in solving problems by search
Problem analyse



- The problem can be decomposed?
  - The sub-problems are independent or not?
- The possible state's space is predictable?
- We want a solution or an optimal solution?
- The solution is represented by a single state or by more successive states?
- We require some knowledge for limiting the search or for identifying the solution?
- The problem is conversational or solitary?
  - Human interaction is required for problem solving?

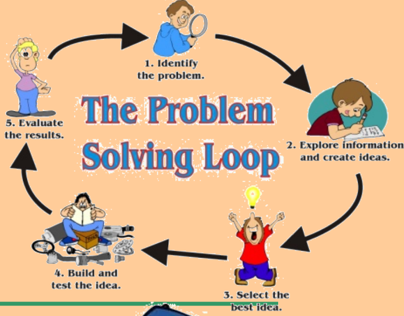
Steps in solving problems by search
Selection of a solving technique



- Solving by moving rules (and control strategy) in the search space until we find a path from the initial state to the final state

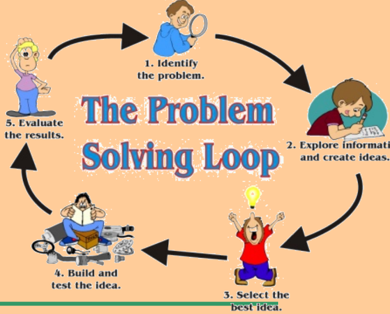
- Solving by search
  - Examination of all possible states in order to identify
    - A path from the initial state to the final state
    - An optimal state
  - The search space = all possible states and the operators that maps the states

Steps in solving problems by search
Selection of a solving technique



- Solving by search
  - More searching strategies → how we select one of them?
    - Computational complexity (temporal and spatial)
    - Completeness → the algorithms always ends and finds a solution (if it exists)
    - Optimality → the algorithms finds the optimal solution (the optimal cost of the path from the initial state to the final state)

Steps in solving problems by search
Selection of a solving technique

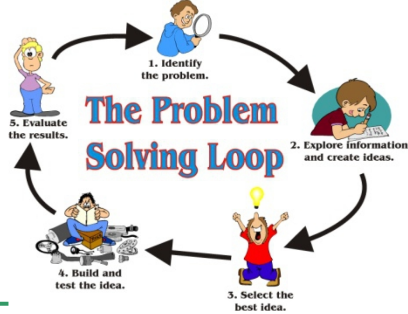


- Solving by search
  - More searching strategies → how we select one of them? → Computational complexity (temporal and spatial)
    - Strategy's performance depends on
      - Time for running
      - Memory for running

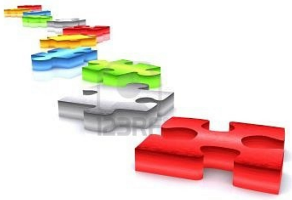
        Internal factors
      - Size of input data
      - Computer's performance
      - Compiler's quality

        External factors
    - Can be evaluated by complexity → computational efficiency
      - Spatial → required memory for solution identification
        - *S(n)* – memory used by the best algorithms *A* that solves a decision problem *f* with *n* input data
      - Temporal → required time for solution identification
        - T(n) – running time (number of steps) of the best algorithm A that solves a decision problem *f* with *n* input data

Steps in solving problems by search
Selection of a solving technique

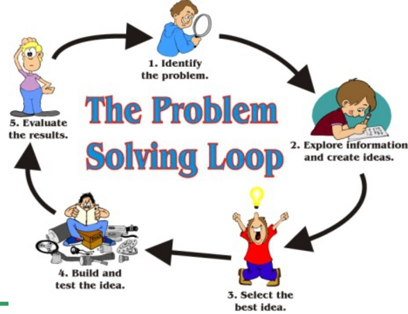


- Problem solving by search can be performed by:
  - Step by step construction of solution



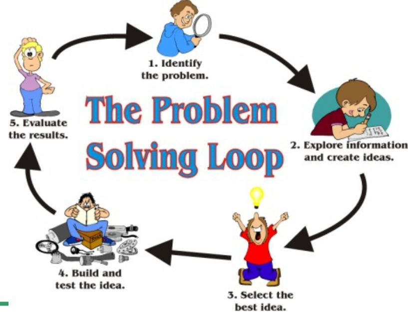
  - Optimal solution identification

Steps in solving problems by search
Selection of a solving technique



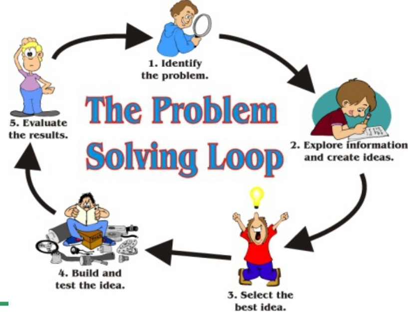
- Problem solving by search can be performed by:
  - Step by step construction of solution
    - Problem’s components
      - Initial state
      - Operators (successor functions)
      - Final state
      - Solution = a path (of optimal cost) from the initial state to the final state
    - Search space
      - All the states that can be obtained from the initial state (by using the operators)
      - A state = a component of solution
    - Example
      - Traveling Salesman Problem (TSP)
    - Algorithms
      - Main idea: start with a solution’s component and adding new components until a complete solution is obtained
      - Recurrent → until a condition is satisfied
      - The search’s history (path from initial state to the final state) is retained in LIFO/FIFO containers
    - Advantages
      - Do not require knowledge (intelligent information)

Steps in solving problems by search
Selection of a solving technique

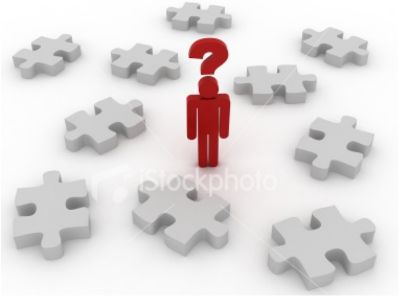


- Problem solving by search can be performed by:
  - Optimal solution identification
    - Problem’s components
      - Conditions (constraints) that must be satisfied by the solution
      - Evaluation function for a potential solution → optimum identification
    - Search space
      - All possible and complete solutions
      - State = a complete solution
    - Example
      - Queen’s problem
    - Algorithms
      - Main idea: start with a state that doesn’t respect some conditions and change it for eliminating these violations
      - Iterative → a single state is retained and the algorithm tries to improve it
      - The searches’ history is not retained
    - Advantages
      - Simple
      - Requires a small memory
      - Can find good solutions in (continuous) search spaces very large (where other algorithms can not be utilised)

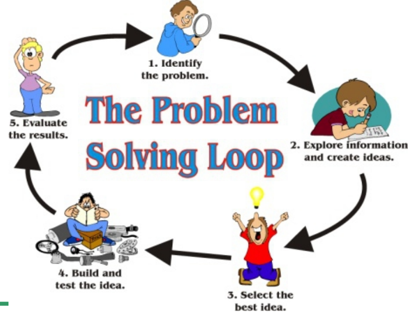
Steps in solving problems by search
Selection of a solving technique



- Solving problem by search involves:
  - Very complex algorithms (NP-complete problems)
  - Search in an exponential space

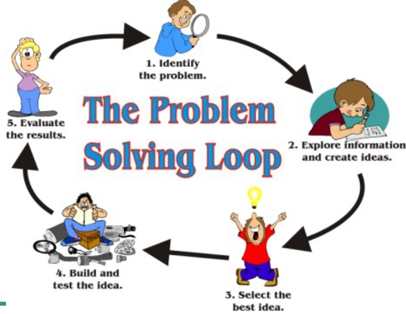


Steps in solving problems by search Selection of a solving technique



- Topology of search strategies:
  - Solution **generation**
    - **Constructive** search
      - Solution is identified step by step
      - Ex. TSP
    - **Perturbative** search
      - A possible solution is modified in order to obtain another possible solution
      - Ex. SAT - Propositional Satisfiability Problem
  - Search space **navigation**
    - **Systematic** search
      - The entire search space is visited
        - Solution identification (if it exists) → complete algorithms
    - **Local** search
      - Moving from a point of the search space into a neighbor point → incomplete algorithms
      - A state can be visited more times
  - **Certain** items of the search
    - **Deterministic** search
      - Algorithms that exactly identify the solution
    - **Stochastic** search
      - Algorithms that approximate the solution
  - Search space **exploration**
    - **Sequential** search
    - **Parallel** search

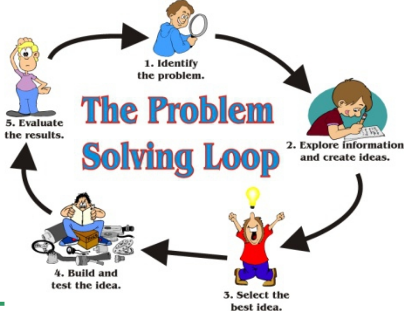
Steps in solving problems by search
Selection of a solving technique



Topology of search strategies:

- **Number of objectives**
  - **Single-objective** search
    - The solution must respect a single condition/constraint
  - **Multi-objective** search
    - The solution must respect more conditions/constraints
- **Number of solutions**
  - **single-modal search**
    - There is a single optimal solution
  - **multi-modal search**
    - There are more optimal solutions
- **Algorithm**
  - Search over a **finite number of steps**
  - **Iterative search**
    - The algorithms converge through the optimal solutions
  - **Heuristic search**
    - The algorithms provide an approximation of the solution
- Search **mechanism**
  - **traditional search**
  - **modern search**
- where the search takes **place**
  - **local search**
  - **global search**

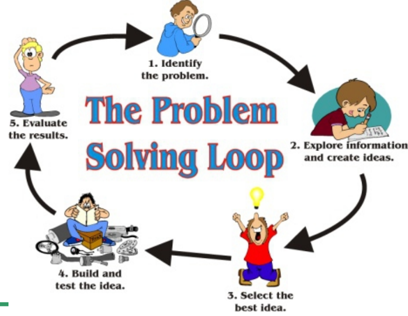
Steps in solving problems by search
Selection of a solving technique



Topology of search strategies:

- Type **(linearity) of constraints**
  - **Linear search**
  - **non-linear search**
    - Clasical (deterministic)
      - Direct – based on evaluation of the objective function
      - Indirect – based on derivates (I and/or II) of the objective function
    - Enumeration-based
      - How solution is identified
        - Uninformed – the solution is the final state
        - Informed – deals with an evaluation function for a possible solution
      - Search space type
        - Complete – the space is finite (if solution exists, then it can be found)
        - Incomplete – the space is infinite
    - Stochastic search
      - Based on random numbers
- **Agents** involves in search
  - Search by **a single agent** → without obstacle for achieving the objectives
  - **Adversarial search** → the opponent comes with some uncertainty

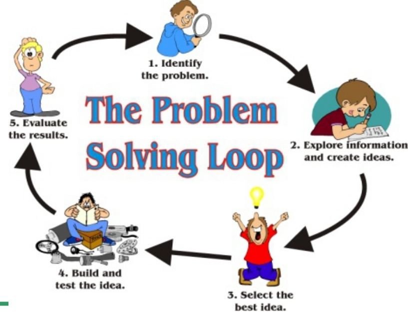
Steps in solving problems by search
Selection of a solving technique



Example

- Topology of search strategies
  - Solution generation
    - **Constructive search**
    - Perturbative search
  - Search space navigation
    - **Systematic search**
    - Local search
  - Certain items of the search
    - **Deterministic search**
    - Stochastic search
  - Search space exploration
    - **Sequential search**
    - Parallel search

Steps in solving problems by search
Selection of a solving technique

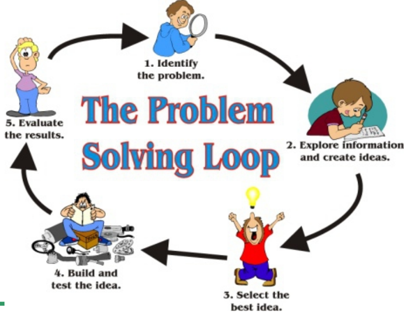


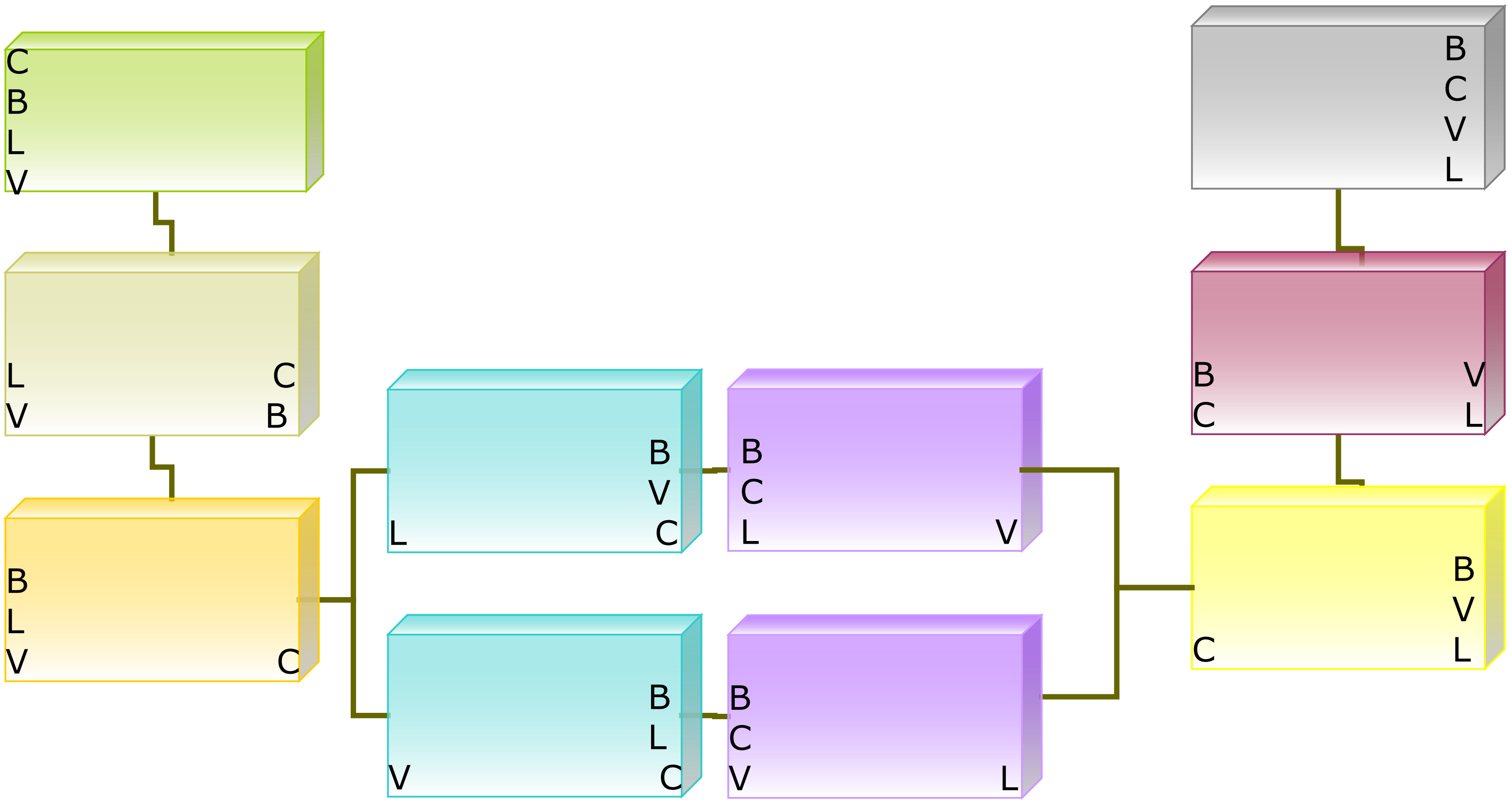
Example

- Constructive, global, deterministic, sequential search

- Problem"capra, varza şi lupul"
  - Input :
    - A goat, a cabbage and a wolf on a river-side
    - A boat with a boater
  - Output:
    - Move all the passengers on the other side of the river
    - Taking into account:
      - The boat has only 2 places
      - It is not possible to rest on the same side:
        - The goat and the cabbage
        - The wolf and the goat

Steps in solving problems by search
Selection of a solving technique





Search strategies – Basic elements

- Abstract data types (ADTs)
  - ADT list → linear structure
  - ADT tree → hierarchic structure
  - ADT graph → graph-based structure
- ADT
  - Domain and operations
  - Representation

Search strategies – Basic elements – ADT list

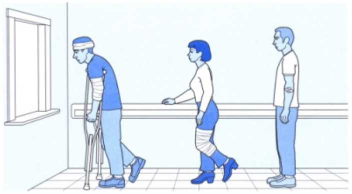
- Domain
  - $D = \{l \mid l = (el1, el2, \ldots)$, where el_i, $i=1,2,3\ldots$, are of type*TE* (*type of element*) and each element el_i, $i=1,2,3\ldots$, has a unique position in *l* of type *TP* (*Type of position*)}
- Operations
  - •Create(l)
  - •First(l)
  - •Last(l)
  - •Next(l,p)
  - •Prev(l,p)
  - •Valid(l,p)
  - •getElement(l,p)
  - •getPoz (l,e)
  - •Modify(l,p,e)
  - •AddFirst(l,e)
  - •AddToEnd(l,e)
  - •AddAfter(l,p,e)
  - •AddBefore(l,p,e)
  - •Eliminate(l,p)
  - •Search(l,e)
  - •IsEmpty(l)
  - •Dimension(l)
  - •Distroy(l)
  - •getIterator(l)
- Representation
  - Vector-based
  - Linked lists
- Special cases
  - **Stack – LIFO**
  - Queue – FIFO
  - Priority queue

Search strategies –
Basic elements – ADT list

- Domain
  - $D = \{l \mid l = (el1, el2, \ldots)$, where el_i, $i=1,2,3\ldots$, are of type*TE* (*type of element*) and each element el_i, $i=1,2,3\ldots$, has a unique position in *l* of type *TP* (*Type of position*)}
- Operations
  - •Create(l)
  - •First(l)
  - •Last(l)
  - •Next(l,p)
  - •Prev(l,p)
  - •Valid(l,p)
  - •getElement(l,p)
  - •getPoz (l,e)
  - •Modify(l,p,e)
  - •AddFirst(l,e)
  - •AddToEnd(l,e)
  - •AddAfter(l,p,e)
  - •AddBefore(l,p,e)
  - •Eliminate(l,p)
  - •Search(l,e)
  - •IsEmpty(l)
  - •Dimension(l)
  - •Distroy(l)
  - •getIterator(l)
- Representation
  - Vector-based
  - Linked lists
- Special cases
  - Stack – LIFO
  - **Queue – FIFO**
  - Priority queue

Search strategies – Basic elements – ADT list

- Domain
  - $D = \{l \mid l = (el1, el2, \ldots)$, where el_i, $i=1,2,3\ldots$, are of type*TE* (*type of element*) and each element el_i, $i=1,2,3\ldots$, has a unique position in *l* of type *TP* (*Type of position*)}
- Operations
  - •Create(l)
  - •First(l)
  - •Last(l)
  - •Next(l,p)
  - •Prev(l,p)
  - •Valid(l,p)
  - •getElement(l,p)
  - •getPoz (l,e)
  - •Modify(l,p,e)
  - •AddFirst(l,e)
  - •AddToEnd(l,e)
  - •AddAfter(l,p,e)
  - •AddBefore(l,p,e)
  - •Eliminate(l,p)
  - •Search(l,e)
  - •IsEmpty(l)
  - •Dimension(l)
  - •Distroy(l)
  - •getIterator(l)
- Representation
  - Vector-based
  - Linked lists
- Special cases
  - Stack – LIFO
  - Queue – FIFO
  - **Priority queue**



Search strategies – Basic elements – ADT Graph

- Domain – container with nodes and links among nodes
  - $D = \{node_1, node_2, \ldots, node_n, link_1, link_2, \ldots, link_m$, where $node_i$, $i=1,2,\ldots,n$ are nodes and $link_i$, $i=1,2,\ldots,m$ are edges between nodes$\}$

- Operations
  - create
  - createNode
  - traverse
  - getIterator
  - destroy

- Representation
  - List of edges
  - List of adjacency
  - Matrix of adjacency
  - Incident matrix

- Special cases
  - Un-oriented and oriented graphs
  - Simple and multiple graphs
  - Connex and non-connex graphs
  - Complete or non-complete graph
  - With or without cycles (a-cycle → forests, trees)

Search strategies – Basic elements – ADT Tree

- Domain – container with nodes and links between nodes
  - $D = \{node_1, node_2, \ldots, node_n, link_1, link_2, \ldots, link_m$, where $node_i$, $i=1,2,\ldots,n$ are nodes and $link_i$, $i=1,2,\ldots,m$ are edges between nodes without cycles}

- Operations
  - create
  - createLeaf
  - addSubtree
  - getInfoRoot
  - getSubtree
  - traverse
  - getIterator
  - destroy

- Representation
  - Vector-based
  - Linked lists of children

- Special cases
  - Binary trees (search trees)
  - N-ary trees

Search strategies – Basic elements – paths in graphs

- *path*
  - Unique nodes
- *trail*
  - Unique edges
- *walk*
  - Without restriction
- Close path
  - Initial node = final node
- circuit
  - A closed *trail*
- cycle
  - A closed *path*

Uninformed search strategies (USS)

- Characteristics
  - Are NOT based on problem specific information
  - Are general
  - Blind strategies
  - Brute force methods
- Topology
  - Order of node exploration:
    - USS in linear structures
      - Linear search
      - Binary search
    - USS in non-linear structures
      - Breadth-first search
        - Uniform cost search (branch and bound)
      - Depth first search
        - Limited depth first search
        - iterative deepening depth-first search
      - Bidirectional search

USS in linear structures Linear search

- **Theoretical aspects**
  - Checks each element of a list until the search one is found
  - The list of elements can be sorted
- **Example**
  - List = (2, 3, 1, ,7, 5)
  - Elem = 7
- **Algorithm**

```
bool LS(elem, list){
      found = false;
      i = 1;
      while ((!found) && (i <= list.length)){
            if (elem = list[i])
               found = true;
            else
               i++;
      }  //while
      return found;
}
```

USS in linear structures Linear search

- **Search analyse**
  - Time complexity
    - Best case: *elem = list[1] => O(1)*
    - Worst case: *elem* $\notin$ *list => T(n) = n +1 => O(n)*
    - Average case: *T(n) = (1 + 2 + … + n + (n+1))/(n+1) => O(n)*
  - Space complexity
    - *S(n) = n*
  - Completeness
    - yes
  - Optimality
    - yes

- **Advantages**
  - Simplicity, good time complexity for small structures
  - Containers can be un-sorted

- **Disadvantages**
  - Bad time complexity for large structures

- **Applications**
  - Search in real data bases

USS in linear structures Binary search

- **Theoretical aspects**
  - Identify an element in a sorted list
  - *Divide et Conquer* strategy

- **Example**
  - List = (2, 3, 5, 6, 8, 9, 13,16, 18), Elem = 6
    - List = (2, 3, 5, 6, 8, 9, 13,16, 18)
    - List = (2, 3, 5, 6)
    - List = (5, 6)
    - List = (6)

- **Algorithm**

```
bool BS(elem, list){
      found = false;
      left = 1;
      right = list.length;
      while((left < right) && (!found)){
             middle = left + (right - left)/2;
             if (element == list[middle])
                 found = true;
             else
                   if (element < list[middle])
                       right = middle - 1;
                   else
                       left = middle + 1;
      }  //while
      return found;
}
```

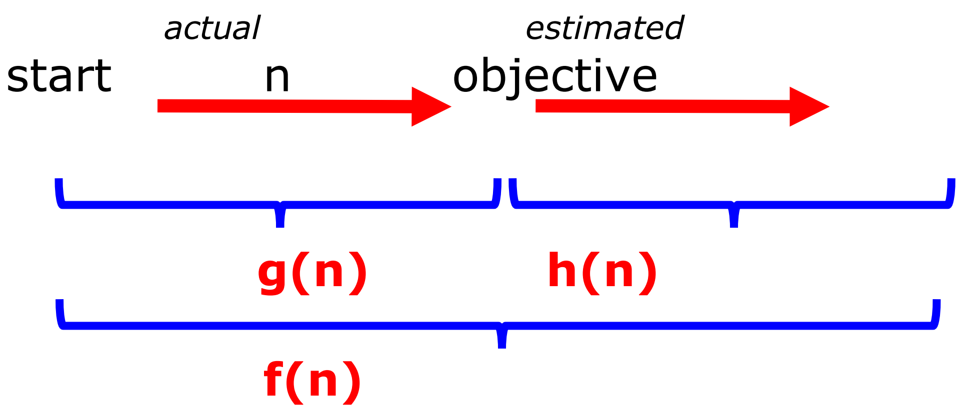
USS in linear structures
Binary search

- **Search analyse**
  - Time complexity $T(n) = 1$, for n = 1 and $T(n) = T(n/2) + 1$, otherwise
    Suppose that $n = 2k \Rightarrow k = \log_2 n$
    Suppose that $2k < n < 2k+1 \Rightarrow k < \log 2n < k + 1$
    $T(n) \quad = T(n/2) + 1$
    $T(n/2) \quad = T(n/22) + 1$
    …
    $T(n/2k-1) = T(n/2k) + 1$

    $T(n) = k + 1 = \log_2 n + 1$
  - Space complexity – $S(n) = n$
  - Completeness – yes
  - Optimality – yes
- **Advantages**
  - Low time complexity compare to linear search
- **Disadvantages**
  - Work with sorted vectors
- **Applications**
  - Guess a number game
  - Search in a phone book or in a dictionary

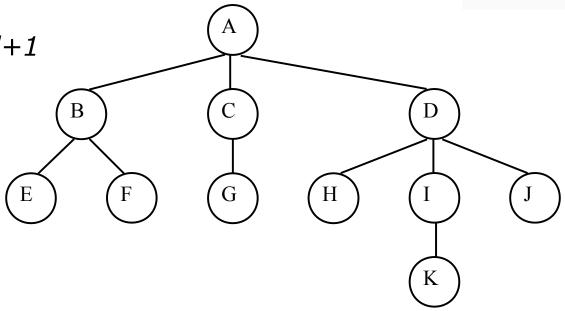
SS in tree-based structures

- Basic elements
  - *f(n)* – evaluation function for estimating the cost of a solution through node (state) *n*
  - *h(n)* – evaluation function for estimating the cost of a solution path from node (state) *n* to the final node (state)
  - *g(n)* – evaluation function for estimating the cost of a solution path from the initial node (state) to node (state) *n*
  - *f(n) = g(n) + h(n)*



USS in tree-based structures
Breadth-first search – BFS

- **Basic elements**
  - All the nodes of depth *d* are visited before all the nodes of depth *d+1*
  - All children of current node are added into a FIFO list (queue)
- **Examplu**



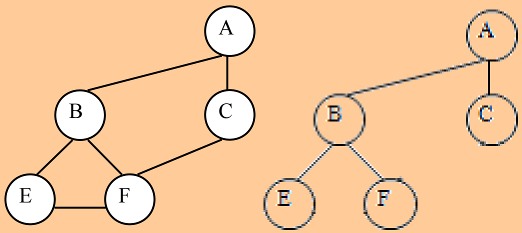
  - Visiting order: A, B, C, D, E, F, G, H, I, J, K
- **Algorithm**

```
bool BFS(elem, list){
        found = false;
        visited = Φ;
        toVisit = {start};       //FIFO list
        while((toVisit != Φ) && (!found)){
                node = pop(toVisit);
                visited = visited U {node};
                if (node == elem)
                    found = true;
                else{
                    aux = Φ;
                    for all (unvisited) children of node do{
                     aux = aux U {child};
                    }
                }
                toVisit = toVisit U aux;
        }  //while
        return found;
}
```

Vizitate deja	De vizitat
Φ	A
A	B, C, D
A, B	C, D, E, F
A, B, C	D, E, F, G
A, B, C, D	E, F, G, H, ,I, J
A, B, C, D, E	F, G, H, I, J
A, B, C, D, E, F	G, H, I, J
A, B, C, D, E, F, G	H, I, J
A, B, C, D, E, F, G, H	I, J
A, B, C, D, E, F, G, H, I	J, K
A, B, C, D, E, F, G, H, I, J	K
A, B, C, D, E, F, G, H, I, J, K	Φ

USS in tree-based structures
Breadth-first search – BFS

- **Search analyse:**
  - Time complexity:
    - *b* – ramification factor (number of children of a node)
    - *d* – length (depth) of solution
    - $T(n) = 1 + b + b^2 + \ldots + b^d \Rightarrow O(b^d)$
  - Space complexity
    - $S(n) = T(n)$
  - Completeness
    - If solution exists, then BFS finds it
  - Optimality
    - No



- **Advantages**
  - Finds the shortest path to the objective node (the shallowest solution)

- **Disadvantages**
  - Generate and retain a tree whose size exponentially increases (with depth of objective node)
  - Exponential time and space complexity
  - Russel&Norving experiment
  - Works only for small search spaces

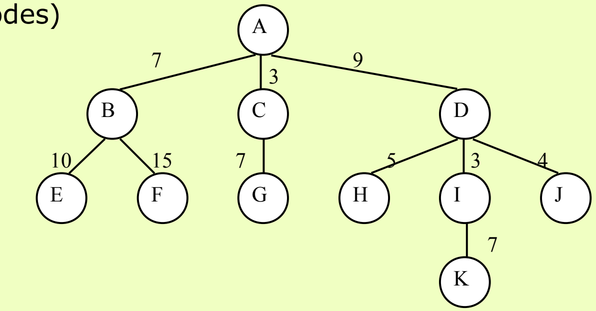
Vizitate deja	De vizitat
Φ	B
B	A, E, F
B, A	E, F, C
B, A, E	F, C
B, A, E, F	C
B, A, E, F, C	Φ

- **Applications**
  - Identification of connex components in a graph
  - Identification of the shortest path in a graph
  - Optimisation in transport networks → algorithm Ford-Fulkerson
  - Serialization/deseralisation of a binary tree (vs. serialization in a sorted manner) allows efficiently reconstructing of the tree
  - Collection copy (garbage collection) → algorithm Cheney

USS in tree-based structures
Uniform cost search – UCS

Basic elements

- BFS +special expand procedure (based on the cost of links between nodes)
- All the nodes of depth *d* are visited before all the nodes of depth *d+1*
- All children of current node are added into a FIFO ordered list
  - The nodes of minimum cost are firstly expanded
  - When a path to the final state is obtained, it becomes candidate to the optimal solution
- *Branch and bound* algorithm



Example

- Visiting order: A, C, B, D, G, E, F, I, H, J, K

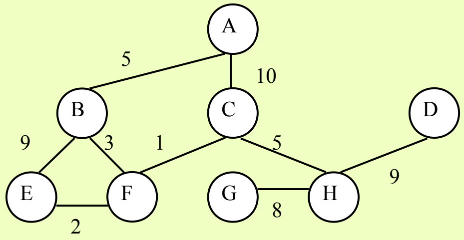
Algorithm

```
bool UCS(elem, list){
        found = false;
        visited = Φ;
        toVisit = {start};    //FIFO sorted list
        while((toVisit != Φ) && (!found)){
               node = pop(toVisit);
               visited = visited U {node};
               if (node== elem)
                   found = true;
               else
                   aux = Φ;
               for all (unvisited) children of node do{
                   aux = aux U {child};
               }  // for
               toVisit = toVisit U aux;
               TotalCostSort(toVisit);
        }  //while
        return found;
}
```

visited	**toVisit**
Φ	A
A	C(3), B(7), D(9)
A, C	B(7), D(9), G(3+7)
A, C, B	D(9), G(10), E(7+10), F(7+15)
A, C, B, D	G(10), I(9+3), J(9+4) ,H(9+5), E(17), F(22)
A, C, B, D, G	I(12), J(13) ,H(14), E(17), F(22)
A, C, B, D, G, I	J(13) ,H(14), E(17), F(22), **K(9+3+7)**
A, C, B, D, G, I, J	H(14), E(17), F(22), **K(19)**
A, C, B, D, G, I, J, H	E(17), F(22), **K(19)**
A, C, B, D, G, I, J, H, E	F(22), **K(19)**
A, C, B, D, G, I, J, H, E, F	K(19)
A, C, B, D, G, I, J, H, E, F, K	Φ

USS in tree-based structures
Uniform cost search – UCS

- **Complexity analyses**
  - Time complexity
    - *b* – ramification factor
    - *d* - length (depth) of solution
    - $T(n) = 1 + b + b^2 + \ldots + b^d \Rightarrow O(b^d)$
  - Space complexity
    - $S(n) = T(n)$
  - Completness
    - yes – if solutions exists, then UCS finds it
  - Optimality
    - Yes

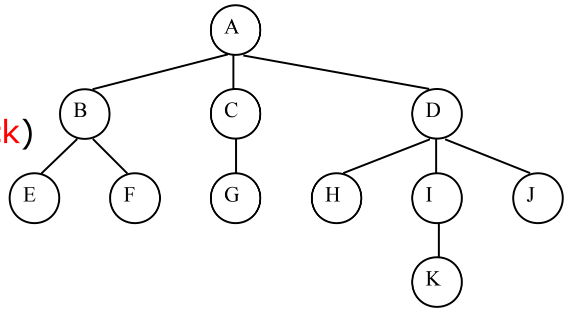


- **Advantages**
  - Finding the minimum cost path to the objective node
- **Disadvantages**
  - Exponential time and space complexity
- **Applications**
  - Shortest path → Dijkstra algorithm

Vizitate deja	De vizitat
Φ	A(0)
A(0)	B(5), C(10)
A(0), B(5)	F(8), C(10), E(14)
A(0), B(5), F(8)	C(9), E(10)
A(0), B(5), F(8), C(9)	E(10), H(14)
A(0), B(5), F(8), C(9), E(10)	H(14)

USS in tree-based structures depth-first search – DFS

- **Basic elements**
  - Expand a child and depth search until
    - The final node is reached or
    - The node is a leaf
  - Coming back in the most recent node that must be explored
  - All the children of the current node are added in a LIFO list (stack)



- **Examplue**
  - Visiting order: A, B, E, F, C, G, D, H, I, K, J

- **Algorithm**

```
bool DFS(elem, list){
      found = false;
      visited = Φ;
      toVisit = {start};    //LIFO list
      while((toVisit != Φ) && (!found)){
            node = pop(toVisit);
            visited = visited U {node};
            if (node== elem)
               found = true;
            else{
                  aux = Φ;
                  for all (unvisited) children of node do{
                     aux = aux U {child};
                  }
                  toVisit = aux U toVisit;
            }
      }  //while
      return found;
}
```

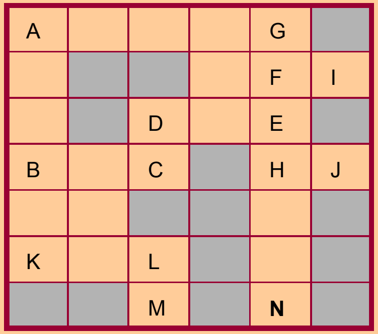
Vizitate deja	De vizitat
Φ	A
A	B, C, D
A, B	E, F, C, D
A, B, E	F, C, D
A, B, E, F	C, D
A, B, E, F, C	G, D
A, B, E, F, C, G	D
A, B, E, F, C, G, D	H, I, J
A, B, E, F, C, G, D, H	I, J
A, B, E, F, C, G, D, H, I	K, J
A, B, E, F, C, G, D, H, I, K	J
A, B, E, F, C, G, D, H, I, K, J	Φ

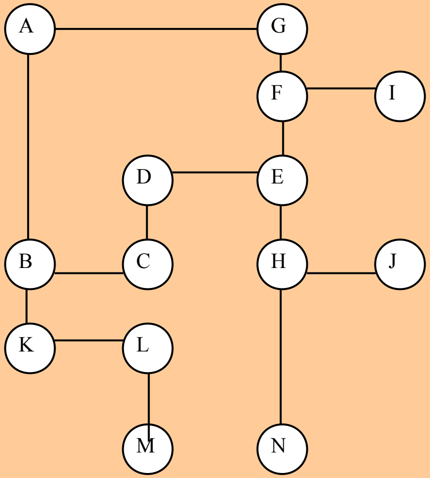
USS in tree-based structures
depth-first search – DFS

- **Complexity analyse**
  - Time complexity
    - b - ramification factor
    - d^{max} – maximal length (depth) of explored tree
    - $T(n) = 1 + b + b^2 + \ldots + b^{dmax} \Rightarrow O(b^{dmax})$
  - Space complexity
    - $S(n) = b * d_{max}$
  - Completeness
    - No → the algorithm does not end for infinite paths (there is no sufficient memory for all the nodes that are visited already)
  - Optimality
    - No → depth search can find a longer path than the optimal one

- **Advantages**
  - Finding the shortest path with minimal resources (recursive version)

- **Disadvantages**
  - Dead paths
    - Infinite cycles
    - Longer solution than the optimal one
- **Applications**
  - Maze problem
  - Identification of connex components
  - Topological sorting
  - Testing the graph planarity





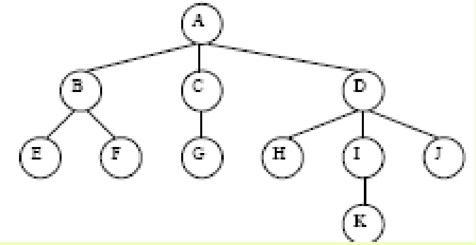
USS in tree-based structures
depth-first search – DFS

```
bool DFS_edges(elem, list){
   discovered = Φ;
   back = Φ;
   toDiscover = Φ;  //LIFO
   for (all neighbours of start) do
        toDiscover = toDiscover U {(start, neighbour)}
   found = false;
   visited = {start};
   while((toDiscover != Φ) && (!found)){
        edge = pop(toDiscover);
        if (edge.out !є visited){
              discovered = discovered U {edge};
              visited = visited U {edge.out}
              if (edge.out == end)
                     found = true;
              else{
                     aux = Φ;
                     for all neighbours of edge.out  do{
                     aux = aux U {(edge.out, neighbour)};
              }
              toDiscover = aux U toDiscover;
        }
        else
              back = back U {edge}
   }  //while
   return found;
}
```

Muchia	Muchii vizitate deja	Muchii de vizitat	înapoi	Noduri vizitate
	Φ	AB, AF	Φ	A
AB	AB	BC, BK, AF	Φ	A, B
BC	AB, BC	CD, BK, AF	Φ	A, B, C
CD	AB. BC, CD	DE, BK, AF	Φ	A, B, C, D
DE	AB, BC, CD, DE	EF, EH, BK, AF	Φ	A, B, C, D, E
EF	AB, BC, CD, DE, EF	FI, FG, EH, BK, AF	Φ	A, B, C, D, E, F
FI	AB, BC, CD, DE, EF, FI	FG, EH, BK, AF	Φ	A, B, C, D, E, F, I
FG	AB, BC, CD, DE, EF, FI, FG	GA, EH, BK, AF	Φ	A, B, C, D, E, F, I, G
GA	AB, BC, CD, DE, EF, FI, FG	EH, BK, AF	GA	A, B, C, D, E, F, I, G
EH	AB, BC, CD, DE, EF, FI, FG	HJ, HN, BK, AF	GA	A, B, C, D, E, F, I, G, H
HJ	AB, BC, CD, DE, EF, FI, FG, HJ	HN, BK, AF	GA	A, B, C, D, E, F, I, G, H, J
HN	AB, BC, CD, DE, EF, FI, FG, HI, HN	BK, AF	GA	A, B, C, D, E, F, I, G, H, N

USS in tree-based structures depth-limited search – DLS

- **Basic elements**
  - DFS + maximal depth that limits the search (d_{lim})
  - Solved the completeness problems of DFS
- **Example**
  - $d_{lim} = 2$
  - Visiting order: A, B, E, F, C, G, D, H, I, J



- **Algorithm**

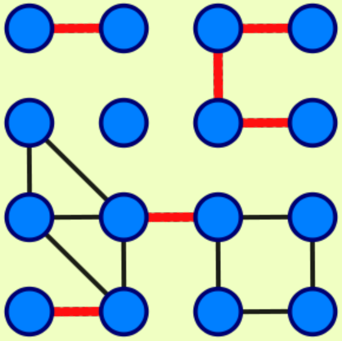
```
bool DLS(elem, list, dlim){
       found = false;
       visited = Φ;
       toVisit = {start}; //LIFO list
       while((toVisit != Φ) && (!found)){
              node = pop(toVisit);
              visited = visited U {node};
              if (node.depth <= dlim){
                     if (node == elem)
                        found = true;
                     else{
                        aux = Φ;
                        for all (unvisited) children of node do{
                         aux = aux U {child};
                        }
                        toVisit = aux U toVisit;
                     }//if found
              }//if dlim
       } //while
       return found;
}
```

Vizitate deja	De vizitat
Φ	A
A	B, C, D
A, B	E, F, C, D
A, B, E	F, C, D
A, B, E, F	C, D
A, B, E, F, C	G, D
A, B, E, F, C, G	D
A, B, E, F, C, G, D	H, I, J
A, B, E, F, C, G, D, H	I, J
A, B, E, F, C, G, D, H, I	J
A, B, E, F, C, G, D, H, I, K, J	Φ

USS in tree-based structures depth-limited search – DLS

- **Complexity analyse**
  - Time complexity:
    - *b* – ramification factor
    - d^{lim} – limit of length (depth) allowed for the explored tree
    - $T(n) = 1 + b + b^2 + \ldots + b^{dlim} \Rightarrow O(b^{dlim})$
  - Space complexity
    - $S(n) = b * d_{lim}$
  - Completeness
    - Yes, but ⇔ $d_{lim} > d$, where d = length (path) of optimal solution
  - Optimality
    - No → DLS can find a longer path than the optimal one

- **Advantages**
  - Solves the completeness problems of DFS

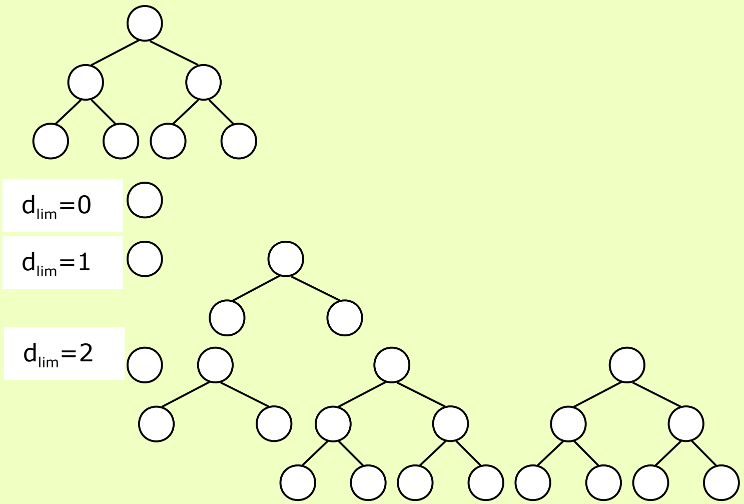


- **Disadvantages**
  - How to choose a good limit d_{lim}?

- **Applications**
  - Identification of bridges in a graph

USS in tree-based structures
iterative deepening depth search – IDDS

- **Basic elements**
  - U DLS(d_{lim}), where d_{lim} = 1, 2, 3, ..., d_{max}
  - Solves the identification of the optimal limit d_{lim} from DLS
  - Usually, it works when:
    - The search space is large
    - The length (depth) of solution is known
- **Example**



- **Algorithm**

```
bool IDS(elem, list){
      found = false;
      dlim = 0;
      while ((!found) && (dlim < dmax)){
            found = DLS(elem, list, dlim);
            dlim++;
      }
      return found;
}
```

USS in tree-based structures
iterative deepening depth search – IDDS

- **Complexity analysis**
  - Time complexity:
    - b^{dmax} nodes at depth d_{max} are expanded once => 1 * b^{dmax}
    - b^{dmax-1} nodes at depth d_{max}-1 are expanded twice => 2 * (b^{dmax-1})
    - ...
    - b nodes at depth 1are expanded d_{max} times => d_{max} * b^1
    - 1 node (the root) at depth 0 is expanded d_{max}+1 times => (d_{max}+1)*b^0

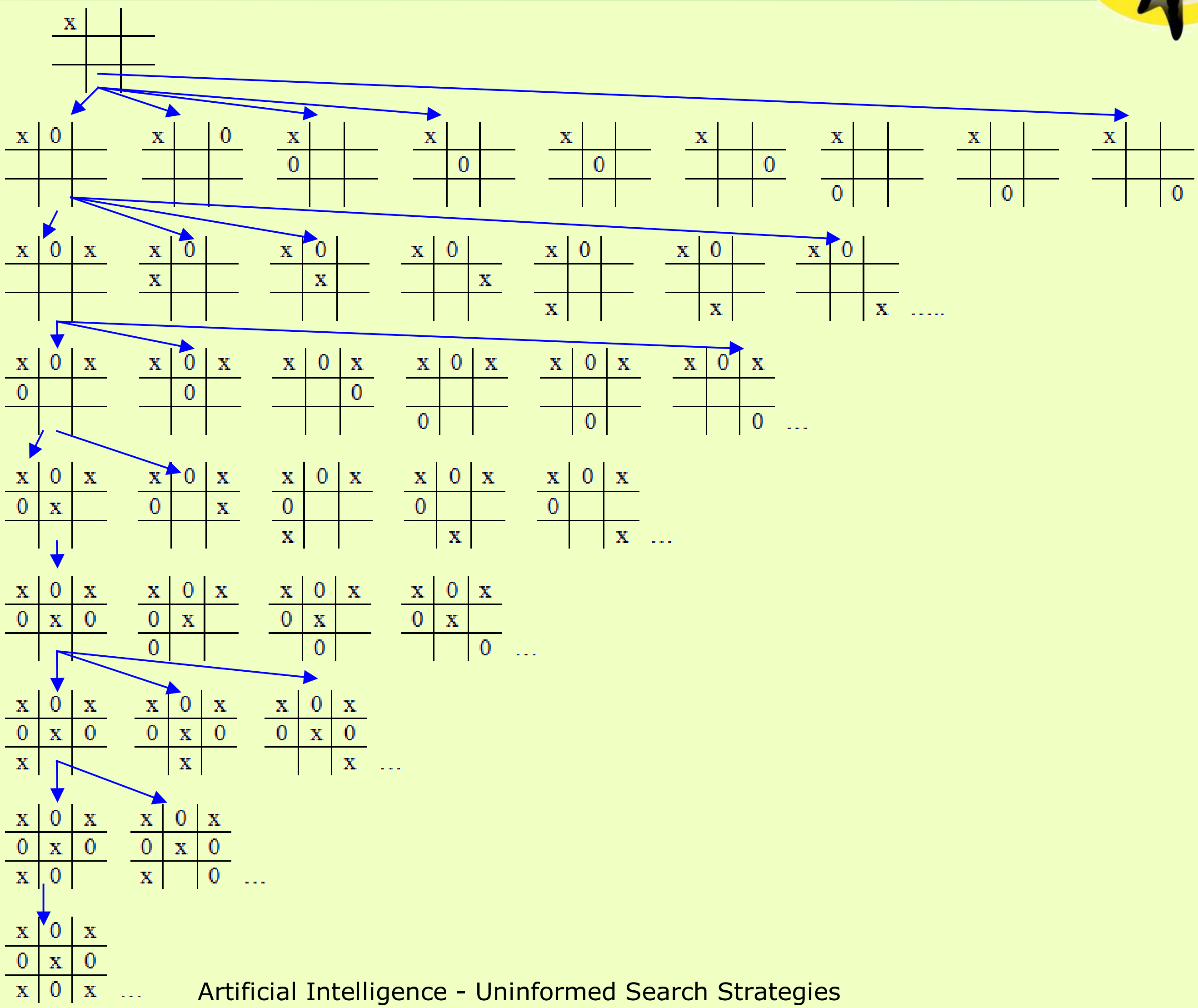
    $$T(n) = \sum_{i=0}^{d_{\max}} (i+1) b^{d_{\max}-1} \Rightarrow O(b^{d_{\max}})$$

  - Space complexity
    - $S(n) = b * d_{max}$
  - Completness
    - yes
  - Optimality
    - yes

- **Advantages**
  - Requires linear memory
  - The goal state is obtained by a minimal path
  - Faster than BFS and DFS
- **Disadvantages**
  - Requires to know the solution depth

- **Applications**
  - Tic tac toe game

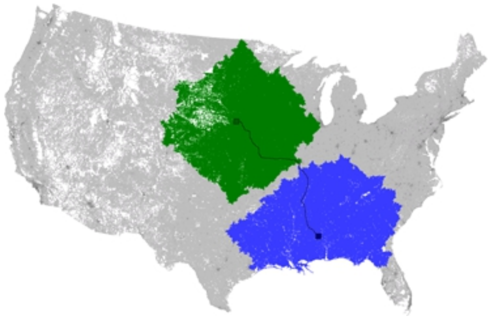
USS in tree-based structures
iterative deepening depth search – IDDS



USS in tree-based structures bi-directional search – BDS

- **Basic elements**
  - 2 parallel search strategies
    - *forward*: from root to leaves
    - *backward*: from leaves to root

    that end when they meet
  - any SS can be used in a direction
  - Requires establishing:
    - the parents and the children of each node
    - the meeting point
- **Example**



- **Algorithm**
  - Depend on the SS used

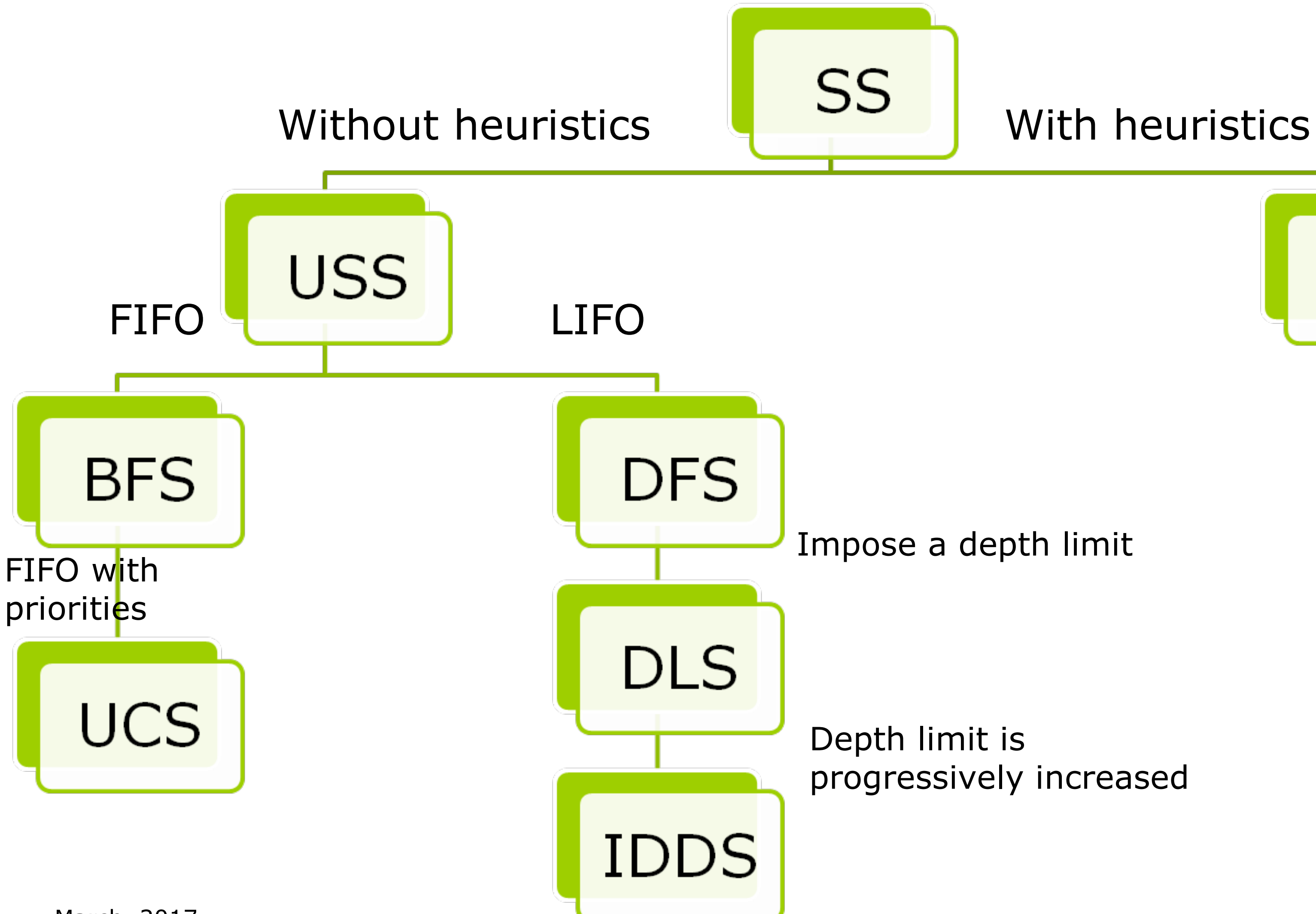
USS in tree-based structures
bi-directional search – BDS

- **Complexity analyse**
  - Time complexity
    - b – ramification factor
    - d – solution length (depth)
    - $O(b^{d/2}) + O(b^{d/2}) \Rightarrow O(b^{d/2})$
  - Space complexity
    - $S(n) = T(n)$
  - Completeness
    - yes
  - Optimality
    - yes

- **Advantages**
  - Good time and space complexity
- **Disadvantages**
  - Each state must be reversed
    - From had to tail
    - From tail to head
  - Difficult to implement
  - Identification of parents and children for all the nodes
  - The final state must be known

- **Applications**
  - Partitioning problem
  - Shortest path

USS in tree-based structures



USS in tree-based structures

Comparison of performances

SS	Time complexity	Space complexity	Completeness	Optimality
BFS	$O(b^d)$	$O(b^d)$	Yes	Yes
UCS	$O(b^d)$	$O(b^d)$	Yes	Yes
DFS	$O(b^{dmax})$	$O(b*d_{max})$	No	No
DLS	$O(b^{dlim})$	$O(b*d_{lim})$	Yes, if $d_{lim} > d$	No
IDS	$O(b^d)$	$O(b*d)$	Da	Yes
BDS	$O(b^{d/2})$	$O(b^{d/2})$	Yes	Yes

Next lecture

A. Short introduction in Artificial Intelligence (AI)

A. **Solving search problems**
   A. **Definition of search problems**
   B. **Search strategies**
      A. Uninformed search strategies
      B. **Informed search strategies**
      C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
      D. Adversarial search strategies

C. Intelligent systems
   A. Rule-based systems in certain environments
   B. Rule-based systems in uncertain environments (Bayes, Fuzzy)
   C. Learning systems
      A. Decision Trees
      B. Artificial Neural Networks
      C. Support Vector Machines
      - Evolutionary algorithms
   D. Hybrid systems

Next lecture – Useful information

- Chapter II.4 of *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- Chapters 3 and 4 of *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Chapter 2.5 of http://www-g.eng.cam.ac.uk/mmg/teaching/artificialintelligence/

References

- Presented information have been inspired from different bibliographic sources, but also from past AI lectures taught by:
  - PhD. Prof. Laura Diosan – www.cs.ubbcluj.ro/~lauras
  - PhD. Prof. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop