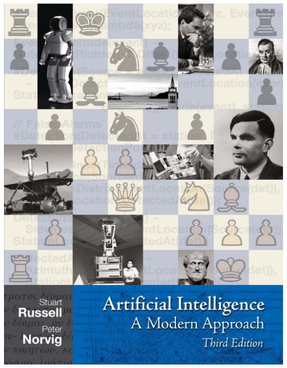
Uninformed state space search strategies
Michal Pěchouček, Milan Rollo

Department of Computer Science
Czech Technical University in Prague

http://cw.felk.cvut.cz/doku.php/courses/ae3b33kui/start

Recommended literature

:: Artificial Intelligence: A Modern Approach (Third Edition) by Stuart Russell and Peter Norvig, 2007 Prentice Hall.



http://aima.cs.berkeley.edu/

Initial remarks on Artificial Intelligence

Artificial intelligence as a:

- Approach to research and understanding of intelligence
- Approach to development of algorithms with characteristics of AI (which necessarily doesn't have to be human):
  - Decision support, optimization of decision trees, decision autonomy

Initial remarks on Artificial Intelligence

Artificial intelligence as a science about human reasoning and nature of human knowledge by modeling them by computers can be divided into following schools:

- symbolic functionalism – intelligence represented in symbols and mutual manipulations examples: knowledge systems, automated reasoning, planning
- connectionism – inspired by natural processes, emergency of intelligent behavior, high number of similar small connected and interacting units example: neural networks
- robotic functionalism (behavioralism)– based on assumption that combining high number of unintelligent processes (black boxes) can lead to intelligent behavior

  example: intelligent robotics
- hybrid and other approaches: multi-agent systems, genetic algorithms, artificial life, ...

Initial remarks on Artificial Intelligence

- **Strong AI** (Bretano): intelligence, which invokes mental states identical with mental states common to human understanding (extreme definition)

Initial remarks on Artificial Intelligence

- **Strong AI** (Bretano): intelligence, which invokes mental states identical with mental states common to human understanding (extreme definition)
- **Weak AI** (Turing): such an understanding of inputs so that it makes the system to react intelligently (as human)

Initial remarks on Artificial Intelligence

- **Strong AI** (Bretano): intelligence, which invokes mental states identical with mental states common to human understanding (extreme definition)
- **Weak AI** (Turing): such an understanding of inputs so that it makes the system to react intelligently (as human)
- **Middling AI** (Smith): the right type of behavior reacting to given precepts is done through an appropriate knowledge structures and reasoning machinery

Initial remarks on Artificial Intelligence

- **Strong AI** (Bretano): intelligence, which invokes mental states identical with mental states common to human understanding (extreme definition)
- **Weak AI** (Turing): such an understanding of inputs so that it makes the system to react intelligently (as human)
- **Middling AI** (Smith): the right type of behavior reacting to given precepts is done through an appropriate knowledge structures and reasoning machinery

- **Turing test** - test for a weak AI.

- **Turing machine** - example of abstract machine, which could be used to model any algorithm, computer program.

Initial remarks on Artificial Intelligence

Symbolic functionalism is based on modeling of two basic aspects of intelligent behavior:

- knowledge
- reasoning

Both can be modeled on various levels of details. **Strong methods** allow general reasoning models, while **weak methods** are specific.

:: Simplified task of AI according to the symbolic functionalism can thus be formulated as: how to represent the right knowledge and to to program such a reasoning mechanisms that will extend out knowledge base with new hypotheses.

Initial remarks on Artificial Intelligence

Having two extreme cases of AI according to the symbolic functionalism:

- **strong** – knowledge is represented by a predicate logic terms and symbols and reasoning model is represented by a deductive reasoning
- **weak** – knowledge is represented as a set of statements and reasoning is represented by a set of `if-then` clauses

In both cases is the set of newly acquired knowledge (either induced or hypothetic) huge and it is necessary to search and create it efficiently. Search strategy is thus a part of the reasoning model. Space of newly acquired knowledge is called **state space**. Durng the problem solving the state space is composed from partial solutions or auxiliary hypothesis. Some partial solutions are classified as goal states.

State Space

Problem solving, as one of aspects of intelligent reasoning, is considered as o problem of finding (from initial state s_0) of such a state s_n, which has features that we require – $\texttt{goal}(s_n)$. Such states we call goal states – s_{goal}. In some cases the problem solving is defined as a problem of finding a path from initial to goal state. In such a case we don't search space of nodes, but space of paths.

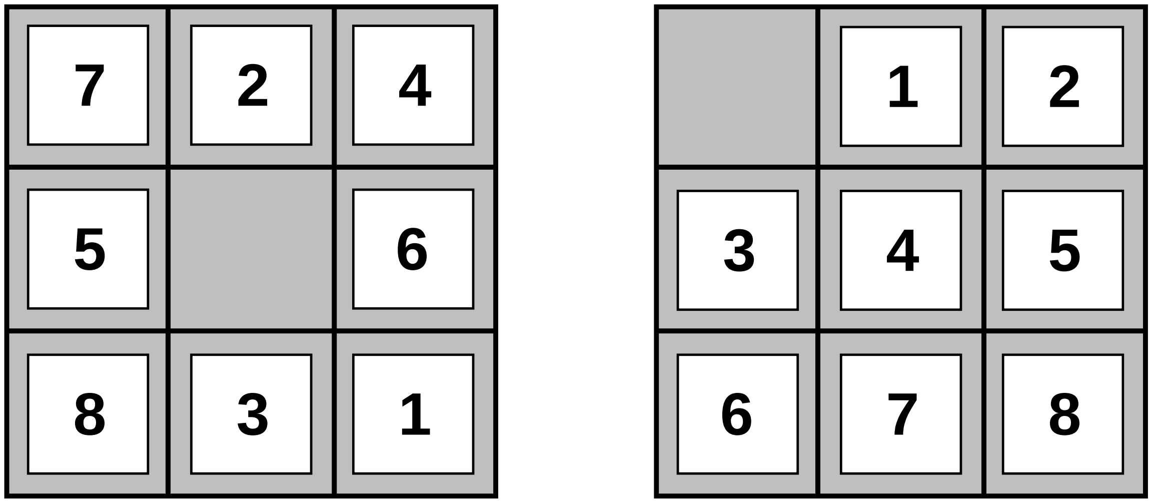
State space search problem is defined by

- initial state - s_0
- goal test – $\texttt{goal}(s_n)$
- successor function - a set of action-state pairs,
- path cost, evaluation of costs of applying actions

Examples: 8-queens problem, cryptarithmetics, chess, 8-puzzle, reasoning in math, natural language processing, planning and scheduling, robotic navigation

Situation becomes complicated in cases, when state space changes dynamically – e.g. in dynamic environment or playing games with adversarial opponent.

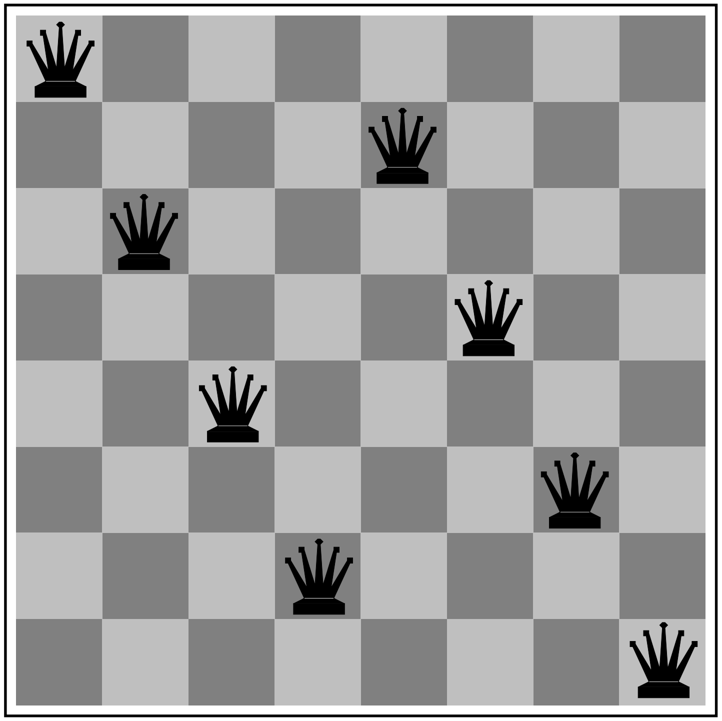
Example: The 8-puzzle



Start State

Goal State

Example: 8-queens

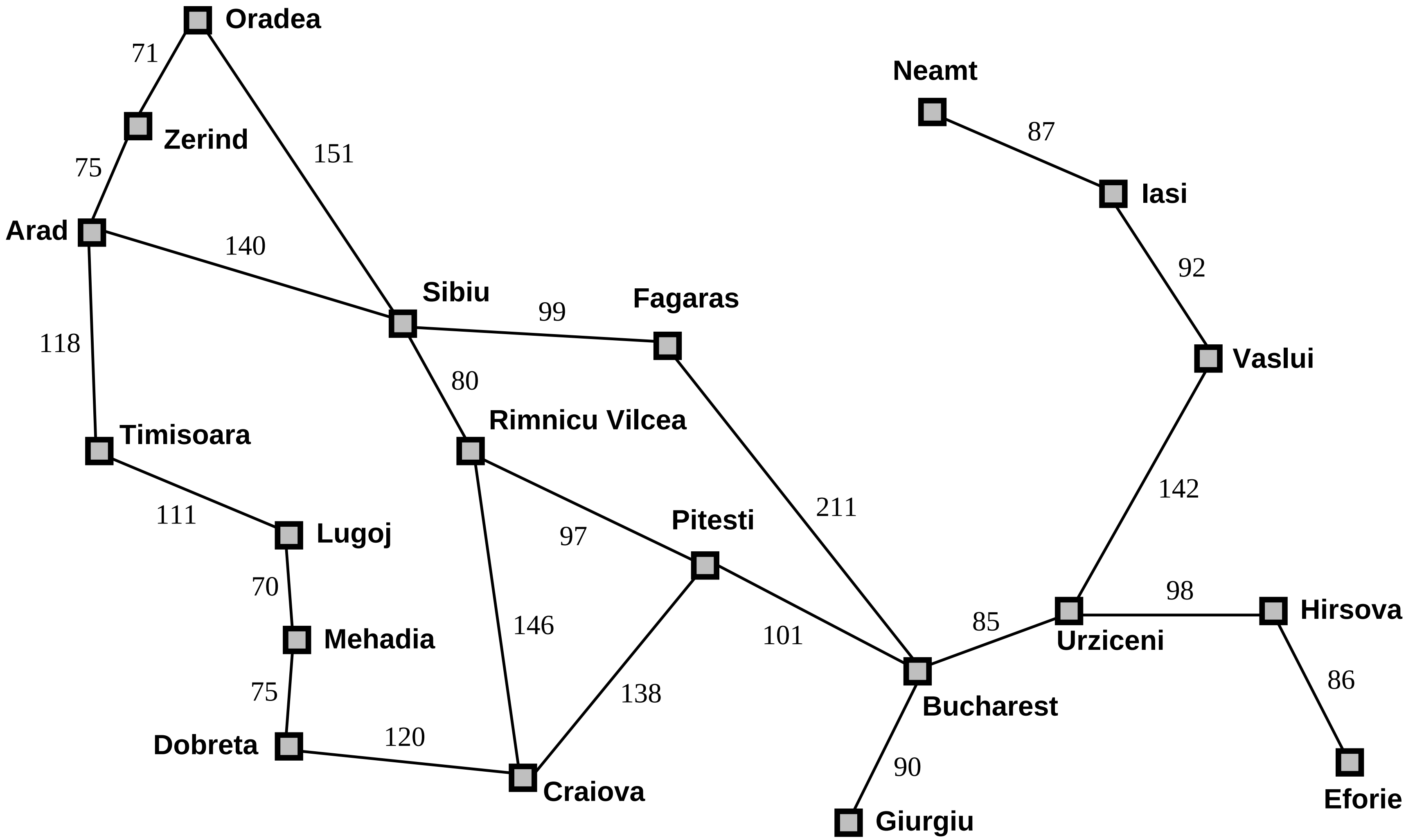


Partial solution of 8-queens problem

Example: Cryptarithmetic

```
    forty        solution:   19786        e.g. f=1, o=9, r=7, etc.
    + ten                    + 850
    + ten                    + 850
---------                ---------
    sixty                    21486
```

Example: Travelling salesman



SEARCH: State space search

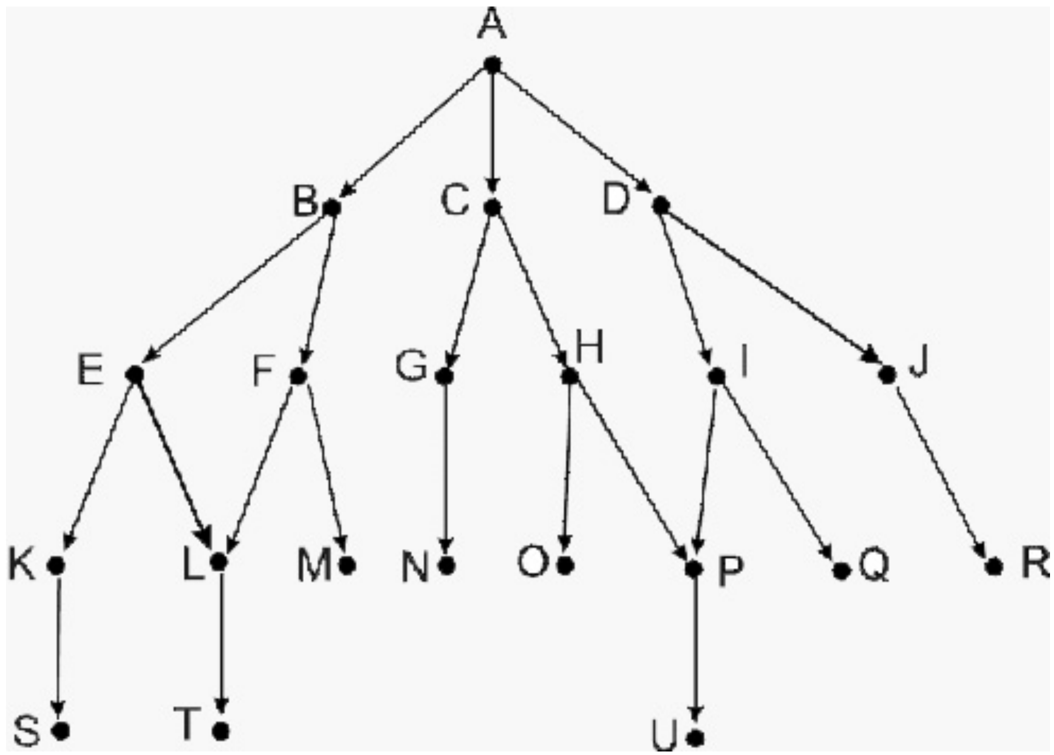
Similar to modeling of artificial intelligence during the state space search we face problems with

- **state space representation** — implementation of actions (`expand` function), infinite loop prevention, ...
- **search strategies** — decision which action to apply as a first, estimations, heuristic functions

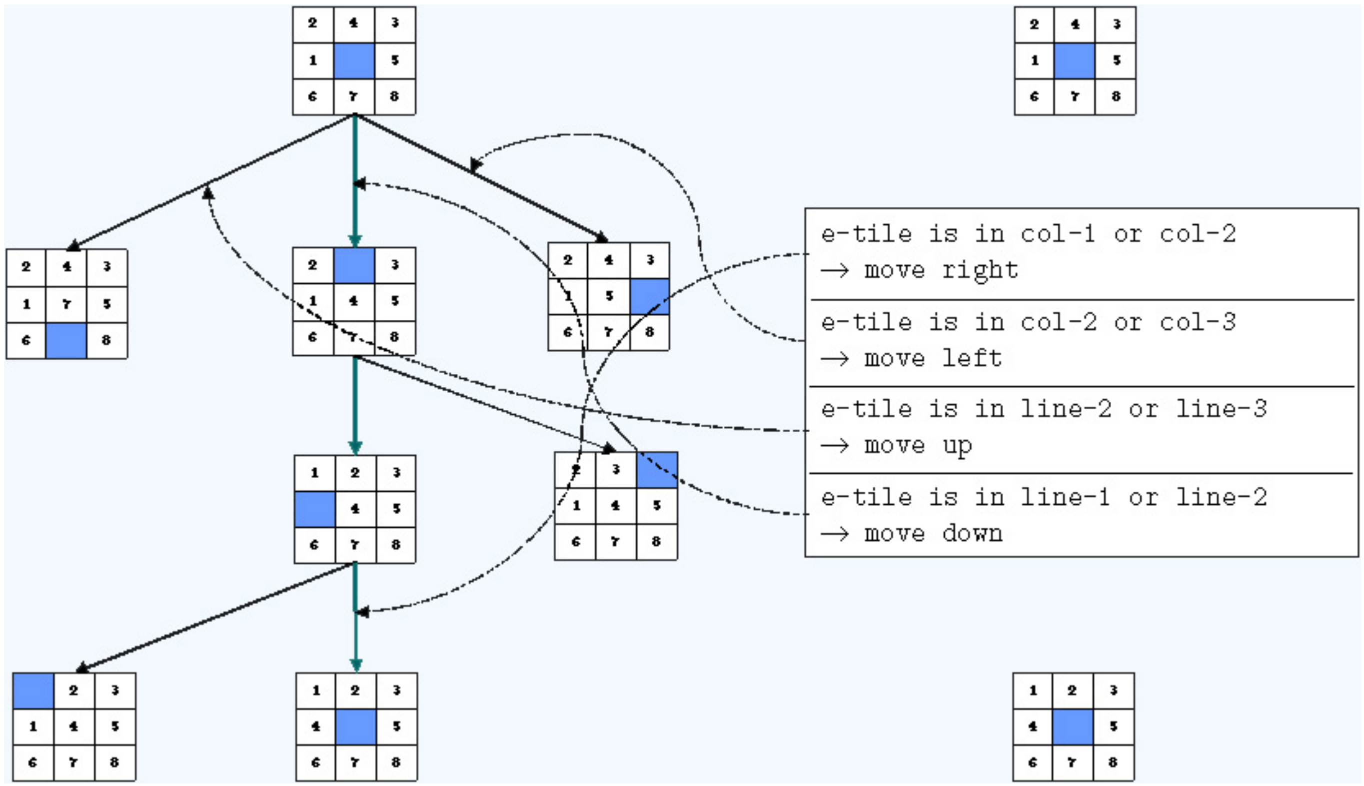
What are our requirements on successful algorithm?

- does it always find a solution if one exists, does if search whole state space? – **completeness**
- does it halt or will it run forever?
- does it always find a least-cost solution? – **optimality**
- what is a complexity of algorithm? – **time** and **space** complexity

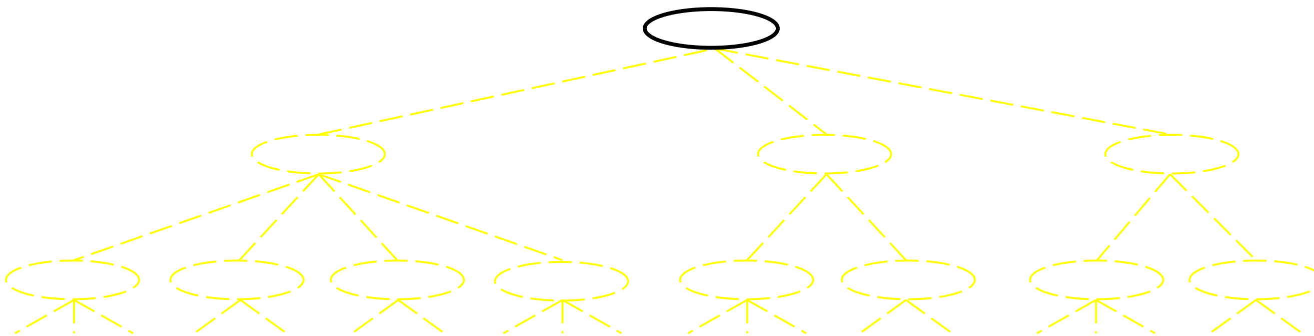
Prohledávání Stavového Prostoru



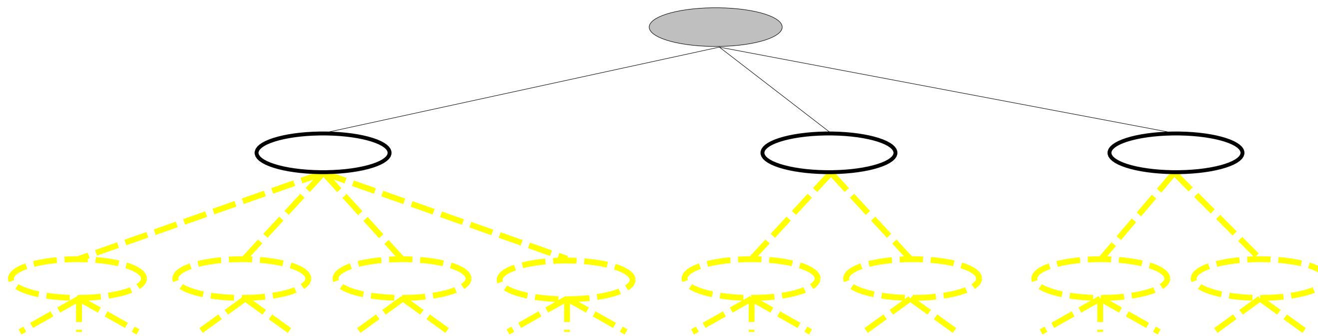
Prohledávání Stavového Prostoru



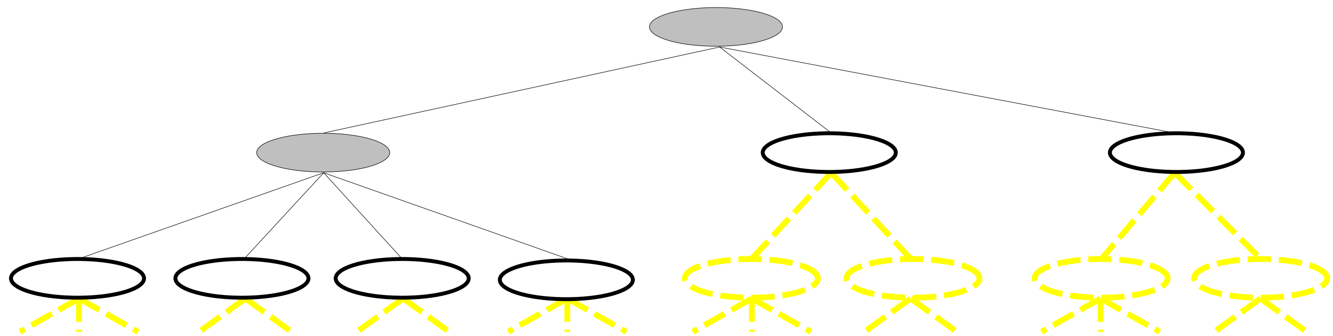
Prohledávání Stavového Prostoru



Prohledávání Stavového Prostoru



Prohledávání Stavového Prostoru



Strategies of uninformed state space search

forward chaining

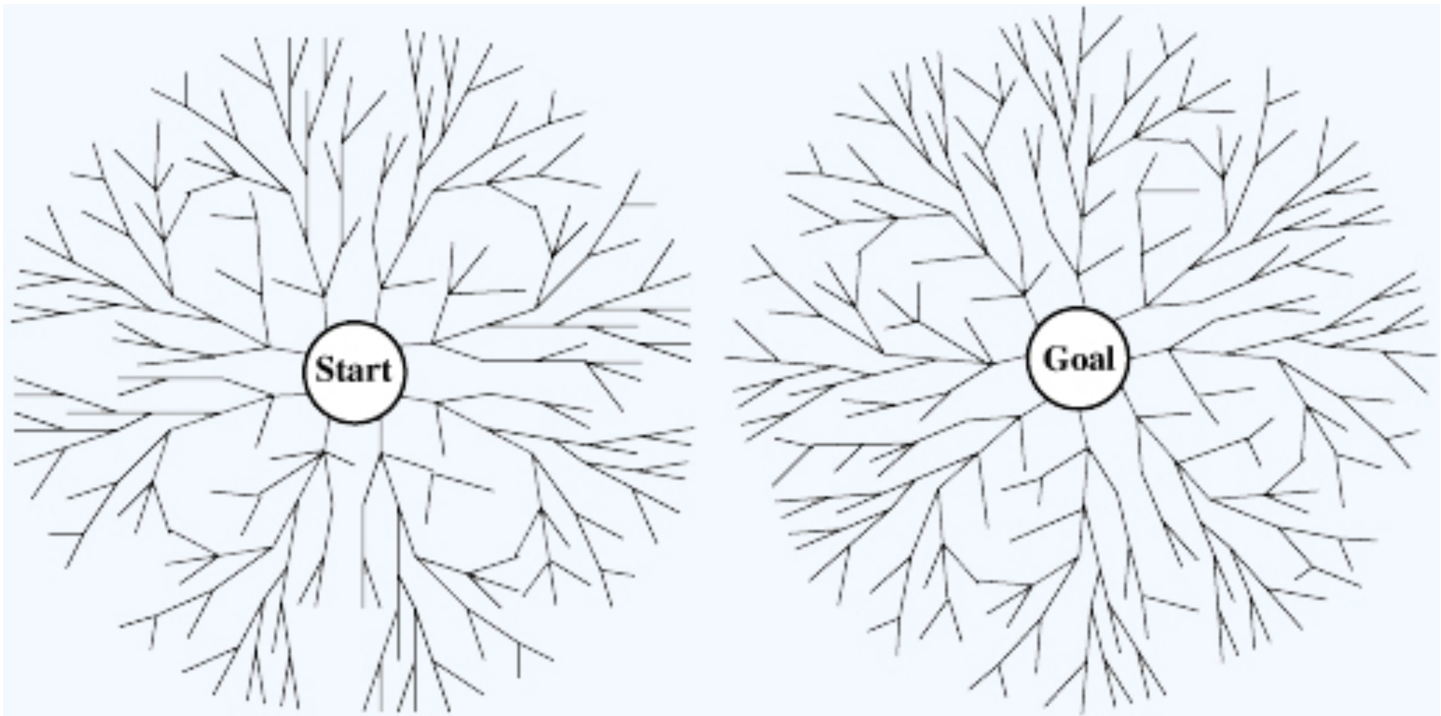
- searches the space from start to end,
- applies actions to find new states,
- process goes iteratively while solution if found

backward chaining

- searches the space from end to start,
- looks for actions, which generate current state
- conditions of those actions generate new goals
- process goes iteratively to state, which describes given problem

Strategies of uninformed state space search

Alternate strategies (Bidirectional search), searches the spaces from both sides



Strategies of uninformed state space search

Search strategies are also divided according to the order in which actions are applied

- **depth-first search** always applies actions on latest expanded node, in case of failure it applies *backtracking*
- **breadth-first search** first searches all states which have a same distance from initial state, before expanding on a new layer

During the state space search we work with:

1. dynamically generated state space in a form of oriented graph
2. s *data structures*: lists that are used for state space search:
   - `open list` – list of open states, used to control the state expansion
   - `closed list` – list of searched states, used to prevent infinite loops

Breadth First Search (BFS)

```
begin
    open := [Start]
    while (open <> []) do begin
        X := first(_open)
        open := open - [X]
        if X = GOAL then return(SUCCESS)
        else begin
                E := expand(X)
                open :=  open + E
             end
    end
    return(failure)
end.
```

mark <> means not equal,

operator – means remove of element(s) from list

operator + means add element(s) at the end of list

Remark - analysis of algorithm complexity

```
function SUM(seq)
    sum <= 0
    for i : 1 to length(seq)
        sum = sum + seq(i)
return sum
```

- how long will algorithm run? (simplification: algorithm run is equal to the number of operations)

Remark - analysis of algorithm complexity

```
function SUM(seq)
    sum <= 0
    for i : 1 to length(seq)
        sum = sum + seq(i)
return sum
```

- how long will algorithm run? (simplification: algorithm run is equal to the number of operations)
  - for `length(seq)` $= n$, $T(n) = 2n + 2$
  - for different n we can work with $T(n)_{avg}$ a $T(n)_{worst}$

Remark - analysis of algorithm complexity

```
function SUM(seq)
    sum <= 0
    for i : 1 to length(seq)
        sum = sum + seq(i)
return sum
```

- how long will algorithm run? (simplification: algorithm run is equal to the number of operations)
  - for `length(seq)` $= n$, $T(n) = 2n + 2$
  - for different n we can work with $T(n)_{avg}$ a $T(n)_{worst}$
- asymptotic analysis of algorithm?
  - $T(n) \approx O(f(n))$ `if` $T(n) \leq kf(n) + c \; \forall n$

Remark - analysis of algorithm complexity

- classification of problems:
  - P problems - (e.g.: $O(n^a)$,$O(\log n)$)
  - NP problems - nondeterministic P problems, deterministic exponential complexity on Turing machine
  - NP hard - hardest from NP class, i.e each problem from NP can be transformed into a solution of NP hard problem
  - NP-complete problems - class of NP problems, which are NP hard and in NP

Properties of BFS

Let b be *maximum branching factor* (highest number of edges from any node) of given tree, d - lowest tree depth, where solution can be found and m maximum tree depth - could be ∞.

Properties of BFS

Let b be *maximum branching factor* (highest number of edges from any node) of given tree, d - lowest tree depth, where solution can be found and m maximum tree depth - could be ∞.

- **complete** ?

Properties of BFS

Let b be *maximum branching factor* (highest number of edges from any node) of given tree, d - lowest tree depth, where solution can be found and m maximum tree depth - could be ∞.

- **complete**: YES (if b is finite)

Properties of BFS

Let b be *maximum branching factor* (highest number of edges from any node) of given tree, d - lowest tree depth, where solution can be found and m maximum tree depth - could be ∞.

- **complete**: YES (if b is finite)
- **time** ?

Properties of BFS

Let b be *maximum branching factor* (highest number of edges from any node) of given tree, d - lowest tree depth, where solution can be found and m maximum tree depth - could be ∞.

- **complete**: YES (if b is finite)
- **time**: $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$ – according to algorithm given on slide 21, we count number of expanded nodes (maximum number of nodes in open list), valid only if $m > d$ (else $O(b^d)$).

Properties of BFS

Let b be *maximum branching factor* (highest number of edges from any node) of given tree, d - lowest tree depth, where solution can be found and m maximum tree depth - could be ∞.

- **complete**: YES (if b is finite)
- **time**: $1 + b + b^2 + b^3 + ... + b^d = O(b^d)$ – could be implemented in case, if we test expanded node for solution immediately after the expansion (see algorithm on following slide) – we assume this algorithm while studying algorithm complexity in the rest of the lecture.

Properties of BFS

```
begin
     if Start = GOAL then return(SUCCESS)
    while (_open <> []) do begin
        X := first(open)
        open := open - [X]
        else begin
                    E := expand(X)
                    if for any Y in E: Y = GOAL then return(SUCCESS)
                        else open := open + E
                end
    end
    return(failure)
end.
```

Properties of BFS

Let b be *maximum branching factor* (highest number of edges from any node) of given tree, d - lowest tree depth, where solution can be found and m maximum tree depth - could be ∞.

- **complete**: YES (if b is finite)
- **time**: $1 + b + b^2 + b^3 + ... + b^d = O(b^d)$
- **space** ?

Properties of BFS

Let b be *maximum branching factor* (highest number of edges from any node) of given tree, d - lowest tree depth, where solution can be found and m maximum tree depth - could be ∞.

- **complete**: YES (if b is finite)
- **time**: $1 + b + b^2 + b^3 + ... + b^d = O(b^d)$
- **space**: $O(b^d)$

Properties of BFS

Let b be *maximum branching factor* (highest number of edges from any node) of given tree, d - lowest tree depth, where solution can be found and m maximum tree depth - could be ∞.

- **complete**: YES (if b is finite)
- **time**: $1 + b + b^2 + b^3 + ... + b^d = O(b^d)$
- **space**: $O(b^d)$
- **optimal** ?

Properties of BFS

Let b be *maximum branching factor* (highest number of edges from any node) of given tree, d - lowest tree depth, where solution can be found and m maximum tree depth - could be ∞.

- **complete**: YES (if b is finite)
- **time**: $1 + b + b^2 + b^3 + ... + b^d = O(b^d)$
- **space**: $O(b^d)$
- **optimal**: yes, if we optimize depth

Properties of BFS

Let b be *maximum branching factor* (highest number of edges from any node) of given tree, d - lowest tree depth, where solution can be found and m maximum tree depth - could be ∞.

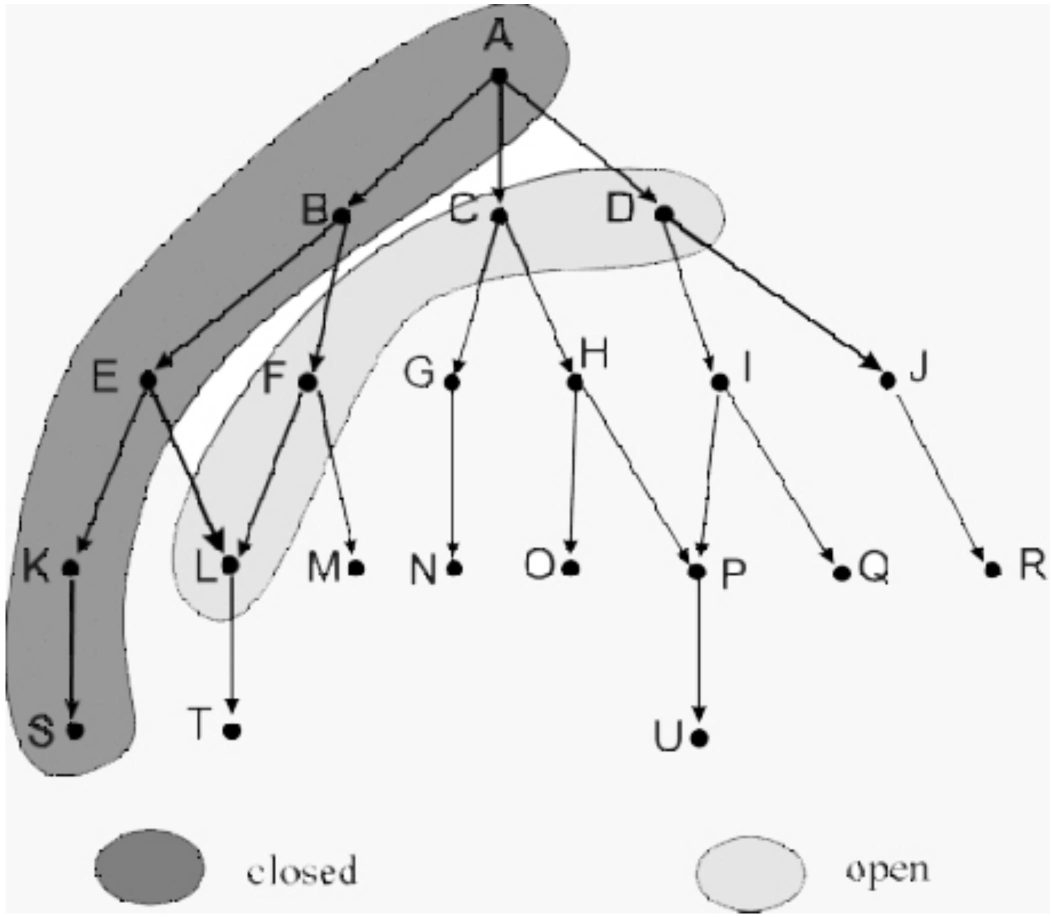
- **complete**: YES (if b is finite)
- **time**: $1 + b + b^2 + b^3 + ... + b^d = O(b^d)$
- **space**: $O(b^d)$
- **optimal**: yes, if we optimize depth

Space is the biggest problem - you can easily generate more than 100MB/sec

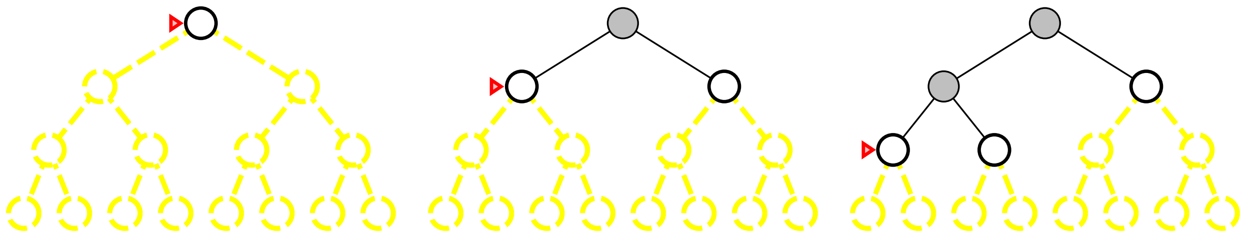
Prohledávání do šířky – Breadth First Search (BFS)

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

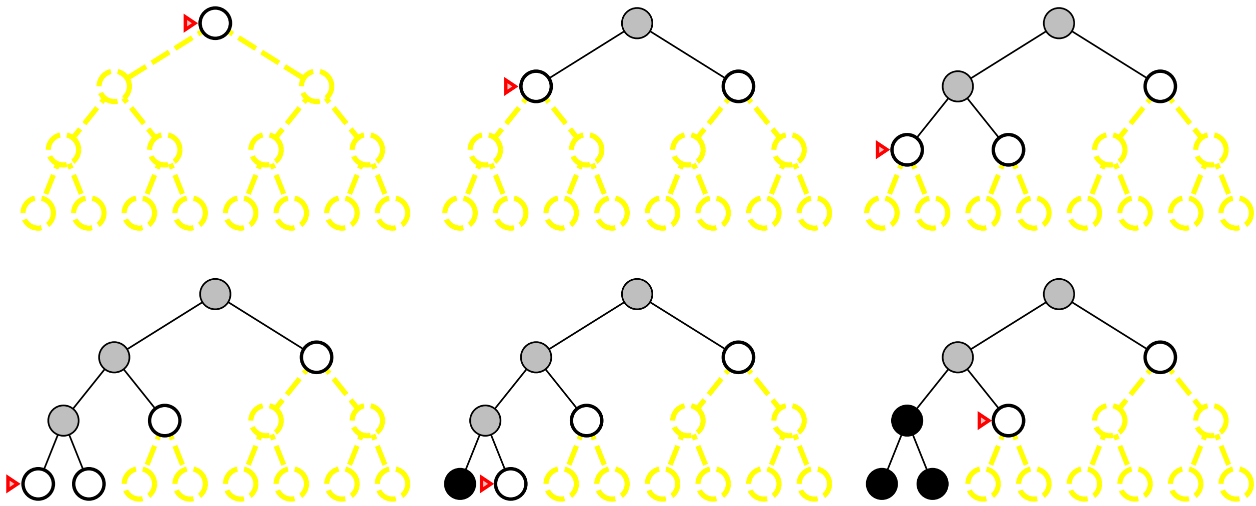
Prohledávání do hloubky – Depth First Search (DFS)



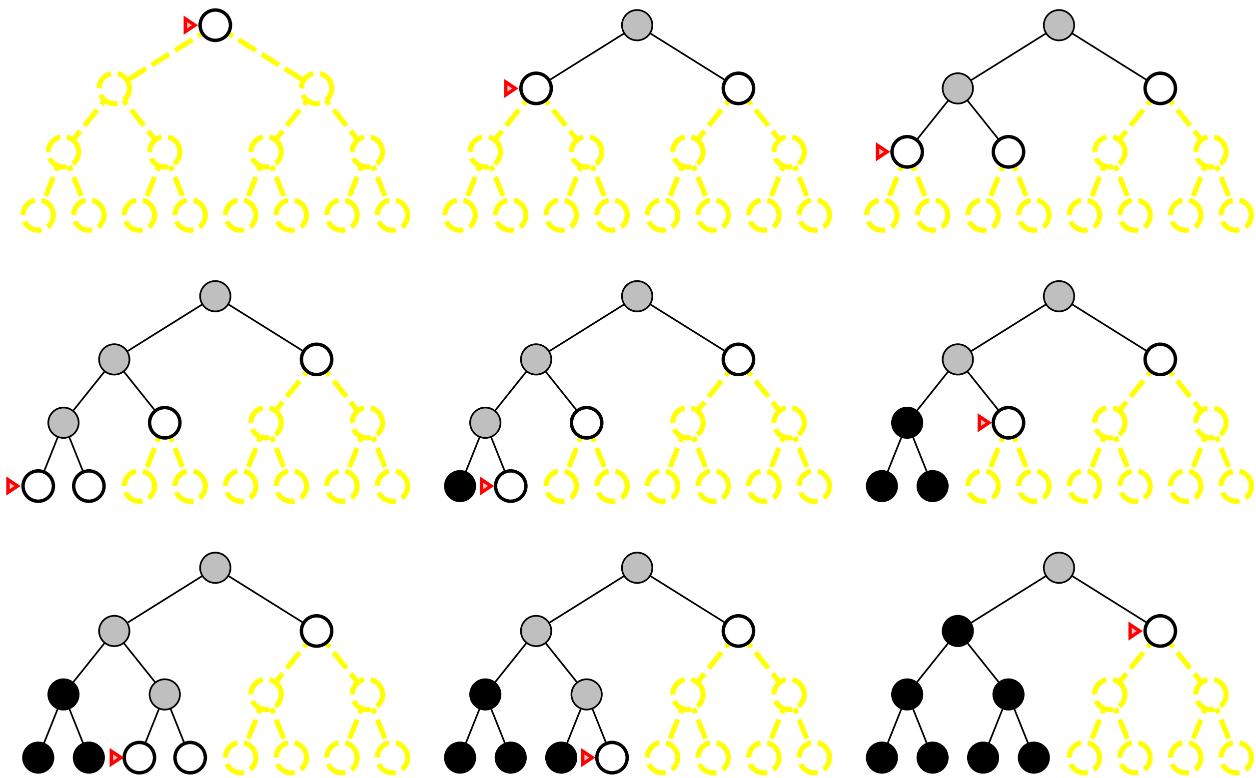
Prohledávání do hloubky – Depth First Search (DFS)



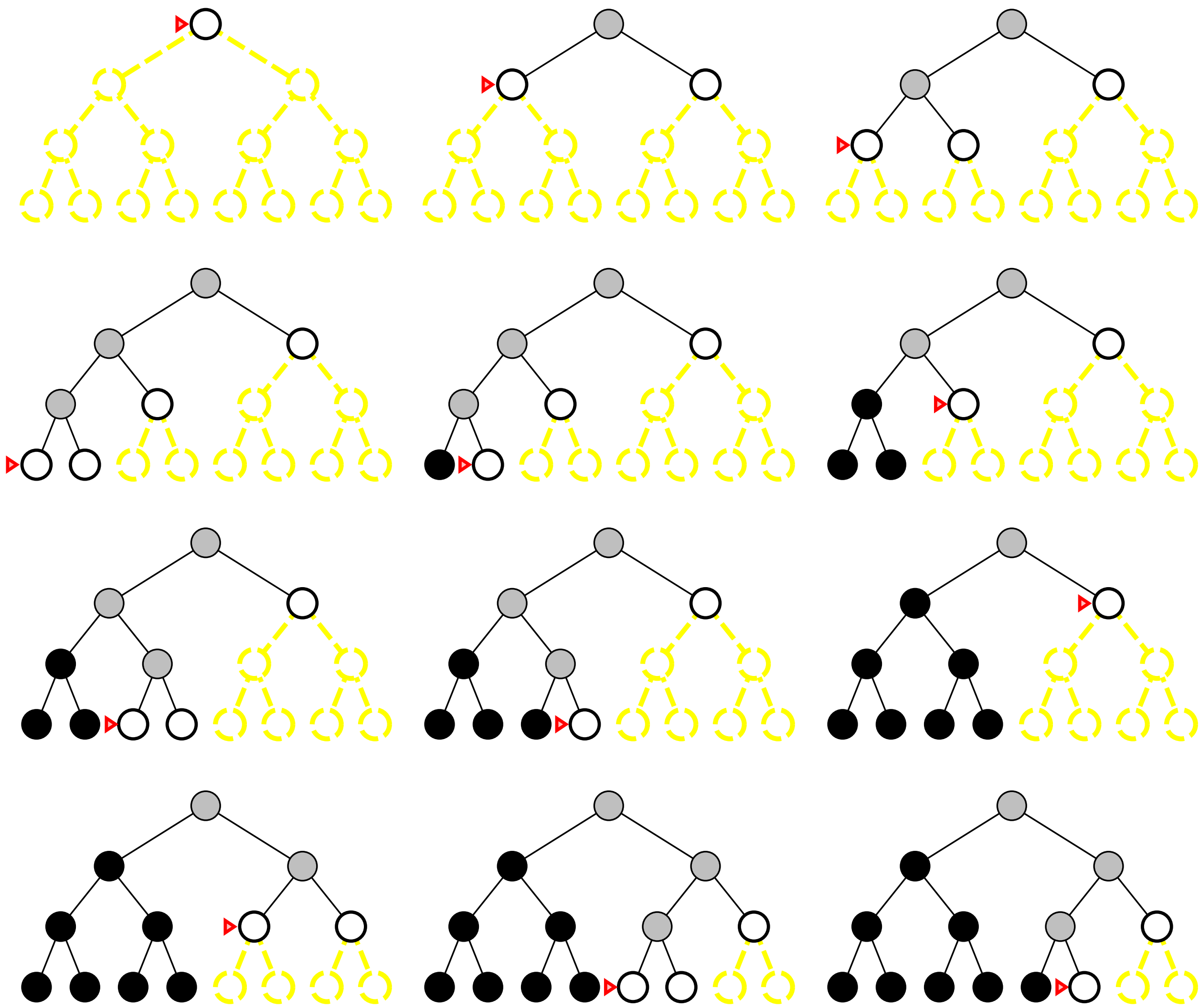
Prohledávání do hloubky – Depth First Search (DFS)



Prohledávání do hloubky – Depth First Search (DFS)



Prohledávání do hloubky – Depth First Search (DFS)



Properties of DFS

Algorithm, which doesn't care about loops:

```
begin
    open := [Start]
    while (open <> []) do begin
        X := first(open)
        open := open - [X]
        if X = GOAL then return(SUCCESS)
        else begin
                E := expand(X)
                open := E + open
             end
    end
    return(failure)
end.
```

Properties of DFS

Algorithm, which prevents infinite looping using a `closed` list:

```
begin
    open := [Start], closed := []
    while (open <> []) do begin
        X := first(open)
        closed := closed + [X], open := open - [X]
        if X = GOAL then return(SUCCESS)
        else begin
                E := expand(X)
                E := E - closed
                open := E + open
             end
    end
    return(failure)
end.
```

Properties of DFS

Let b be *maximum branching factor* (highest number of edges from any node) of given tree, d - lowest tree depth, where solution can be found and m maximum tree depth - could be ∞.

Properties of DFS

Let b be *maximum branching factor* (highest number of edges from any node) of given tree, d - lowest tree depth, where solution can be found and m maximum tree depth - could be ∞.

- **complete** ?

Properties of DFS

Let b be *maximum branching factor* (highest number of edges from any node) of given tree, d - lowest tree depth, where solution can be found and m maximum tree depth - could be ∞.

- **complete**: NO (even if b is bounded, due to the possible existence of loops)

Properties of DFS

Let b be *maximum branching factor* (highest number of edges from any node) of given tree, d - lowest tree depth, where solution can be found and m maximum tree depth - could be ∞.

- **complete**: NO (even if b is bounded, due to the possible existence of loops)
- **time** ?

Properties of DFS

Let b be *maximum branching factor* (highest number of edges from any node) of given tree, d - lowest tree depth, where solution can be found and m maximum tree depth - could be ∞.

- **complete**: NO (even if b is bounded, due to the possible existence of loops)
- **time**: b^m – i.e. exponentially by m, problems, if m is much larger than d.

Properties of DFS

Let b be *maximum branching factor* (highest number of edges from any node) of given tree, d - lowest tree depth, where solution can be found and m maximum tree depth - could be ∞.

- **complete**: NO (even if b is bounded, due to the possible existence of loops)
- **time**: b^m – i.e. exponentially by m, problems, if m is much larger than d.
- **space** ?

Properties of DFS

Let b be *maximum branching factor* (highest number of edges from any node) of given tree, d - lowest tree depth, where solution can be found and m maximum tree depth - could be ∞.

- **complete**: NO (even if b is bounded, due to the possible existence of loops)
- **time**: b^m – i.e. exponentially by m, problems, if m is much larger than d.
- **space**: $O(bm)$

Properties of DFS

Let b be *maximum branching factor* (highest number of edges from any node) of given tree, d - lowest tree depth, where solution can be found and m maximum tree depth - could be ∞.

- **complete**: NO (even if b is bounded, due to the possible existence of loops)
- **time**: b^m – i.e. exponentially by m, problems, if m is much larger than d.
- **space**: $O(bm)$
- **optimal** ?

Properties of DFS

Let b be *maximum branching factor* (highest number of edges from any node) of given tree, d - lowest tree depth, where solution can be found and m maximum tree depth - could be ∞.

- **complete**: NO (even if b is bounded, due to the possible existence of loops)
- **time**: b^m – i.e. exponentially by m, problems, if m is much larger than d.
- **space**: $O(bm)$
- **optimal**: no

Alternate strategies

DL-DFS (Depth-limited) search:
depth-first search with depth limit l

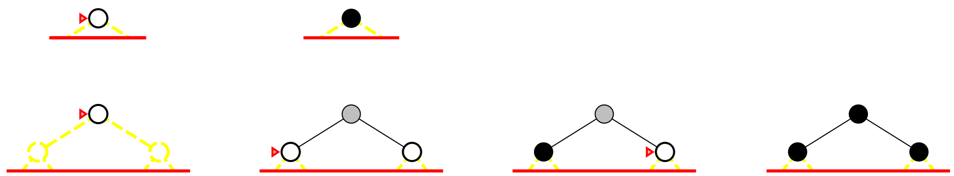
ID-DFS (Iterative deepening) search:
depth-first search with iteratively increasing depth limit l

Algorithm:

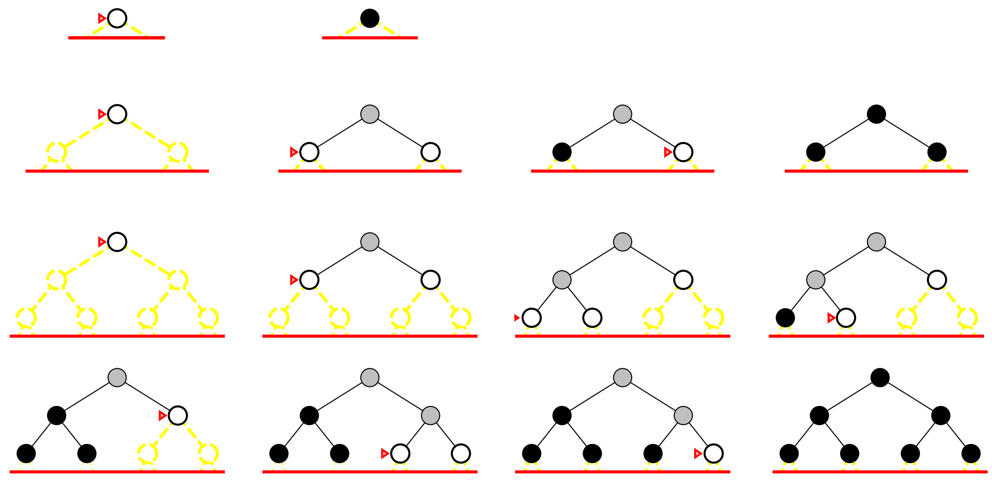
1. $l = 1$
2. `do DL-DFS with depth` l
3. `if solution found end`
   `else` $l = l + 1$ a goto 2

Algoritmus IDDFS prohledávání

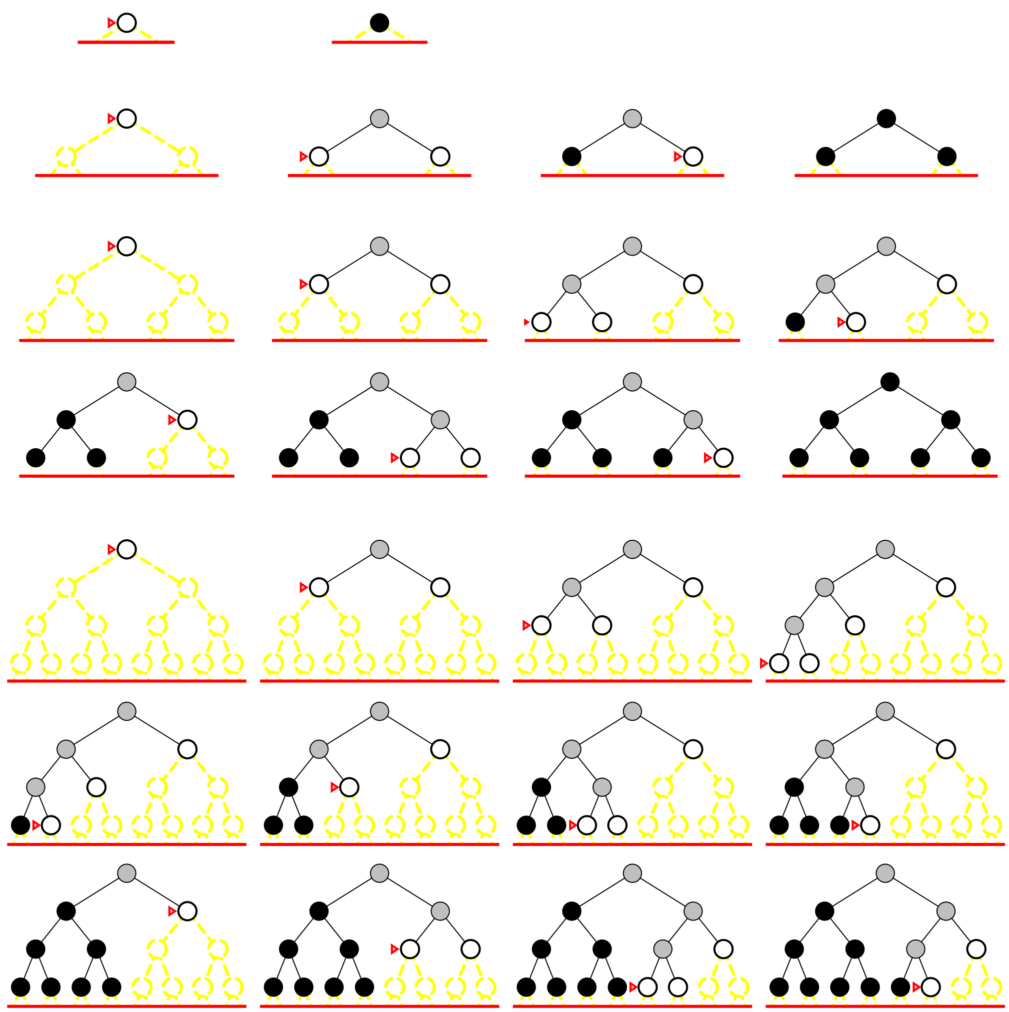
Algoritmus IDDFS prohledávání



Algoritmus IDDFS prohledávání



Algoritmus IDDFS prohledávání



Iterative deepening search

- **complete** ?

Iterative deepening search

- **complete**: YES (if b is bounded)

Iterative deepening search

- **complete**: YES (if b is bounded)
- **time** ?

Iterative deepening search

- **complete**: YES (if b is bounded)
- **time**: $d + 1 + (d)b + (d-1)b^2 + (d-2)b^3 + ... + b^d < db^d = O(b^d)$
  if we assume thet each search is carried out by algorithm with complexity $O(b^l)$

Iterative deepening search

- **complete**: YES (if b is bounded)
- **time**: $d + 1 + (d)b + (d-1)b^2 + (d-2)b^3 + ... + b^d < db^d = O(b^d)$
- **space** ?

Iterative deepening search

- **complete**: YES (if b is bounded)
- **time**: $d + 1 + (d)b + (d-1)b^2 + (d-2)b^3 + ... + b^d < db^d = O(b^d)$
- **space**: $O(bd)$

Iterative deepening search

- **complete**: YES (if b is bounded)
- **time**: $d + 1 + (d)b + (d-1)b^2 + (d-2)b^3 + ... + b^d < db^d = O(b^d)$
- **space**: $O(bd)$
- **optimal** ?

Iterative deepening search

- **complete**: YES (if b is bounded)
- **time**: $d + 1 + (d)b + (d-1)b^2 + (d-2)b^3 + ... + b^d < db^d = O(b^d)$
- **space**: $O(bd)$
- **optimal**: yes, if we optimize depth

Iterative deepening search

- **complete**: YES (if b is bounded)
- **time**: $d + 1 + (d)b + (d-1)b^2 + (d-2)b^3 + ... + b^d < db^d = O(b^d)$
- **space**: $O(bd)$
- **optimal**: yes, if we optimize depth

:: **comparison:** for $b = 10$ and $d = 5$ in a worst case:

Iterative deepening search

- **complete**: YES (if b is bounded)
- **time**: $d + 1 + (d)b + (d-1)b^2 + (d-2)b^3 + ... + b^d < db^d = O(b^d)$
- **space**: $O(bd)$
- **optimal**: yes, if we optimize depth

:: **comparison:** for $b = 10$ and $d = 5$ is the number of expanded nodes in a worst case:

- $N(\text{id-dfs}) = 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456$
- $N(\text{bfs}) = 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111$

ID-DFS expands just 11% nodes more, which pays off due to the huge memory savings.

Comparison of strategies

Criterion/algorithm	BFS	DFS	DL-DFS	ID-DFS	BiDir
time	b^d	b^m	b^l	b^d	$b^{\frac{d}{2}}$
space	b^d	bm	bl	bd	$b^{\frac{d}{2}}$
optimality	yes	no	no	yes	yes
completeness	yes	no	yes (for $l \geq d$)	yes	yes

where b is branching factor, d is a depth of shallowest solution, m is maximum tree depth, l is depth limit.