Artificial Intelligence 1

Prof. Dr. Frank Hopfgartner
Dr. Matthias Horbach

Institute for Web Science and Technologies (WeST)
University of Koblenz

Overview

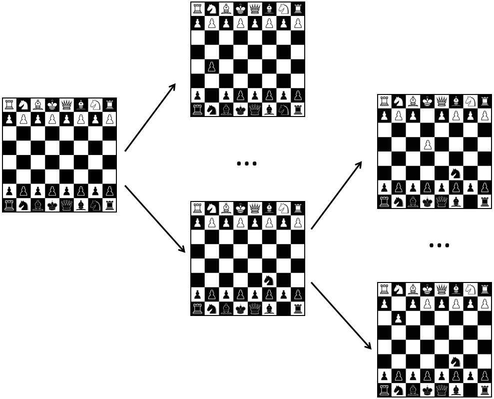
Search

- One of the core problems in AI is *general problem solving*:

  Given a problem specification, find a solution
- Direct algorithm design is often difficult
- Idea: Formalise problem as (graph) search problem
- Nodes are (partial) solutions, edges represent transition between (partial) solutions
- Challenges:
  - Graph is usually too big for traditional search algorithms such as Dijkstra ($\rightarrow$ heuristic search methods)
  - Graph is not explicitly given ($\rightarrow$ implicit graph search)

Example: Chess

- Nodes: Game configurations (what is the location of all pieces)?
- Edges: connects game configurations by legal moves
- Challenge: What will the opponent do in his turn?



More examples

- Games in general
  - Start node: initial state of the game
  - Edges: legal moves
  - Goal node: winning situation (e. g. checkmate)
  - Desired: node sequence to goal node
- Formal systems
  - Start node: statement that is to be proven true
  - Edges: deduction rules ($\{\phi, \neg\phi \vee \psi\} \longrightarrow \{\psi\}$)
  - Goal node: True statement
  - Desired: Proof
- Planning
  - Start node: initial configuration
  - Edges: legal actions
  - Goal node: goal configuration
  - Desired: action sequence

Formalisation

Definition

A *search problem* P is a tuple $P = (S, O, I, G)$ with

1. S is a set of states
2. $O \subseteq S \times S$ is the state transition relation
3. $I \in S$ is the initial state
4. $G \subseteq S$ is the set of goal states

Definition

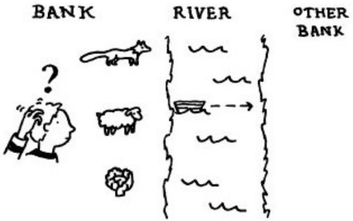
Let $P = (S, O, I, G)$ be a search problem. A *solution* π for P is a sequence $\pi = (S_1, \ldots, S_n)$ with

1. $S_1, \ldots, S_n \in S$, $S_1 = I$, $S_n \in G$
2. for all $i = 1, \ldots, n-1$, $(S_i, S_{i+1}) \in O$.

π is a an *optimal solution* if n is minimal.

Example: fox, sheep, cabbage 1/2

- A fox, a sheep, and a cabbage are on the left bank of a river
- The goal is to bring all three to the right bank of the river
- At no point in time is it allowed to leave either fox+sheep or sheep+cabbage alone on any bank
- The boat can carry only one additional object besides yourself



Example: fox, sheep, cabbage 2/2

Formalisation

- fox=f, sheep=s, cabbage=c
- States are tuples $(L, R, \circ)$ with $L, R \subseteq \{f, s, c\}$ and $\circ \in \{l, r\}$
- Examples:
  - $(\{f, s\}, \{c\}, l)$: fox and sheep are on the left bank, the cabbage is on the right bank, the boat (and you) is on the left bank.
  - $(\{\}, \{f, s, c\}, r)$: fox, sheep, and cabbage are on the right bank, the boat (and you) is on the right bank as well.
- Therefore $P_{fsk} = (S_{fsk}, O_{fsk}, (\{f, s, c\}, \{\}, l), \{(\{\}, \{f, s, c\}, l), (\{\}, \{f, s, c\}, r)\})$ and
  - $S_{fsk} = \{(L, R, \circ) \mid L, R \subseteq \{f, s, c\}, \circ \in \{l, r\}\}$
  - $O_{fsk} = \{((\{f, s, c\}, \{\}, l), (\{f, c\}, \{s\}, r)), \ldots\}$

Example: $(n^2 - 1)$-puzzle 1/2

- Consider n^2 tiles arranged in a square with pieces $1, \ldots, (n^2 - 1)$ and an empty field
- The goal is to sort the numbers by sliding tiles onto the empty field

1	2	3
8		4
7	6	5

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	

Example: $(n^2 - 1)$-puzzle 1/2

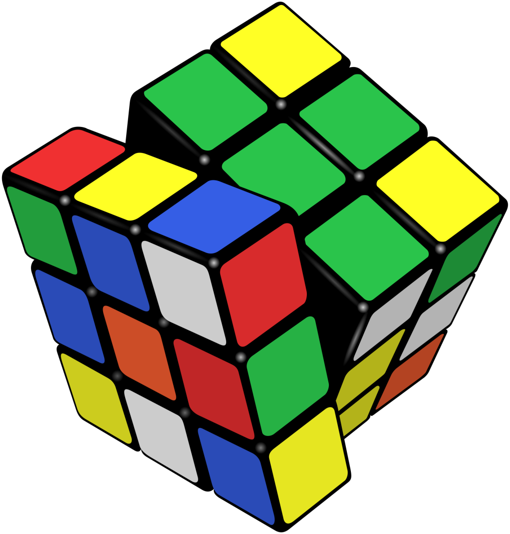
Formalisation

- States: 2-dimensional array `int[n][n]` with numbers $1, \ldots, (n^2 - 1)$ and 0 for "empty"
- Example: `[[1,2,3],[8,0,4],[7,6,5]]`

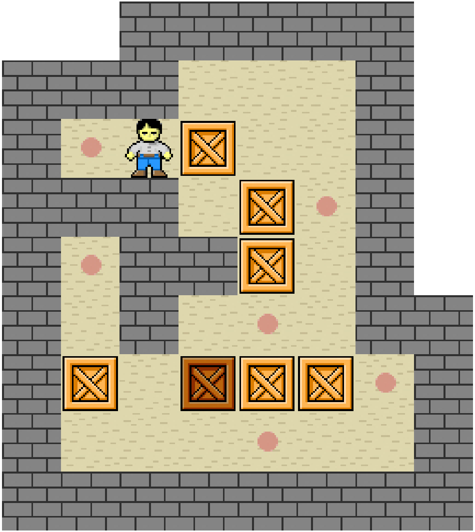
1	2	3
8		4
7	6	5

- Therefore $P_3 = (S_3, O_3, I, \{[[1,2,3],[8,0,4],[7,6,5]]\})$ with
  - $S_3 = \{[[x_1, x_2, x_3], [x_4, x_5, x_6], [x_7, x_8, x_9]] \mid \{x_1, \ldots, x_9\} = \{0, \ldots, 8\}\}$
  - I is random
  - $O_3 = \{($`[[3,4,8],[1,2,7],[5,0,6]]`, `[[3,4,8],[1,2,7],[0,5,6]]`$), \ldots\}$

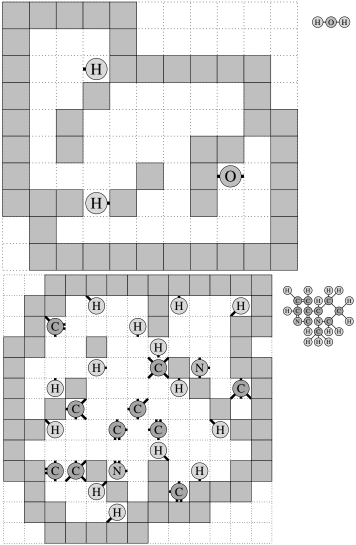
More examples: Rubik's Cube



More examples: Sokoban



More examples: Atomix



The problem of defining states

- Finding the "right" definition of a state is crucial and influences the problem complexity
- Recall fox-sheep-cabbage: $(L, R, \circ)$ with $L, R \subseteq \{f, s, k\}$ and $\circ \in \{l, r\}$
- Better: $(L, \circ)$ with $L \subseteq \{f, s, k\}$ and $\circ \in \{l, r\}$ (everything that is not on the left bank, is automatically on the right bank)
- Advantage: state space is significantly smaller, search may be more efficient
- A good state definition is often more important than selecting the right search algorithm

Overview

Uninformed search

- Given a search problem $P = (S, O, I, G)$ with initial state I
- Oftentimes the state space S is not given explicitly, but will be (partially) constructed during search starting from I (implicit graph search)
- At the beginning we only know I and its direct successors
- At the time we look at a successor N of I, we can construct its direct successors
- Assumption for uninformed search: we have no further information besides the state transition relation
- So we have no idea which "direction" to explore during search

Implicit graph search 1/2

- Search graph is usually not explicitly given
- is being (partially) constructed only during search

Example: 8-puzzle

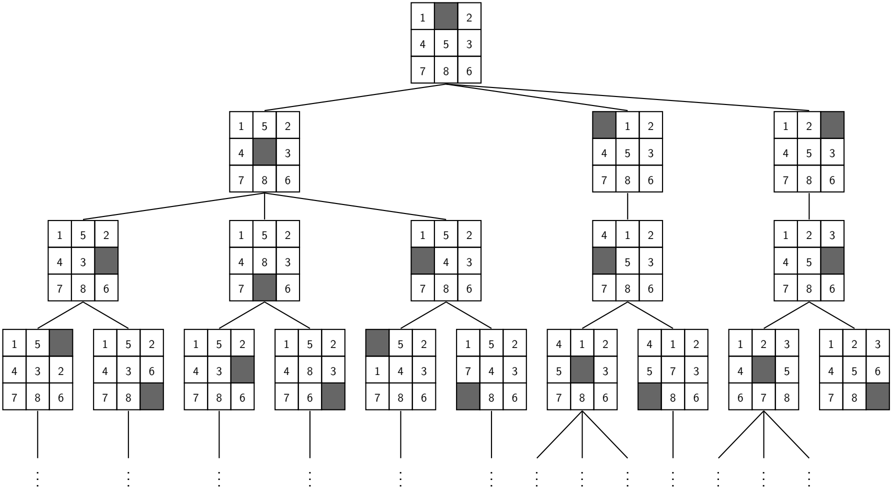
1		2
4	5	3
7	8	6

Initial state

1	2	3
4	5	6
7	8	

Goal state

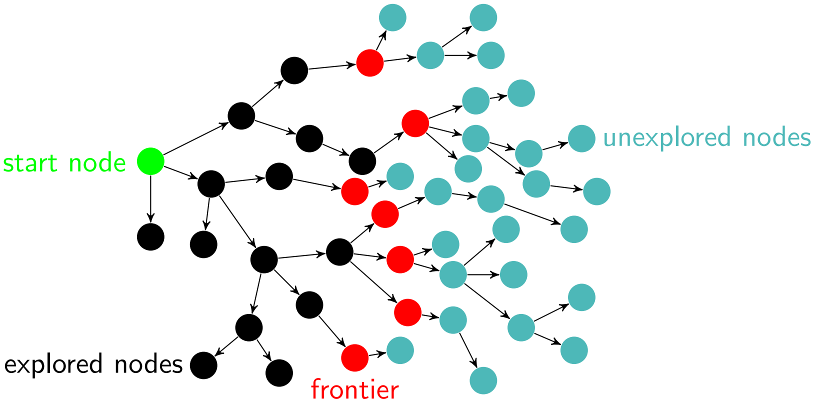
Implicit graph search 2/2



Graph search 1/2

- General schema: try out paths incrementally starting from the initial state
- At any time we maintain a set of *frontier nodes*; these form the boundary between the explored part of the graph and the unexplored one.
- The longer the algorithm runs the further we push the boundary, until we reach a goal node
- The *search strategy* is defined by the way the next node is selected:
  - breadth-first search
  - depth-first search
  - iterative depth-first search
  - bidirectional search

Graph search 2/2



Evaluation of search strategies

When do we regard a search strategy as "good"?

- Completeness: if there is a goal node, does the strategy always find it?
- Runtime complexity: how many nodes do we need to process to find a goal node?
- Space complexity: how many nodes are in `Frontier` at any given time?
- Optimality: do we always find an optimal solution (=minimal distance to start node)?

Algorithm schema

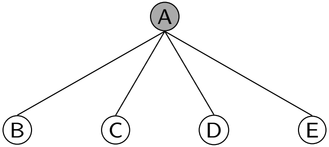
```
Algorithm graph_search
Input:
   Start note I,
   Successor state function Succ(.),
   Goal test function isGoal(.)
Output: FAIL/SUCCESS

Explored = {};
Frontier = {I};
while(true){
   if(Frontier == {}) return FAIL;
   n = remove node from Frontier;
   add n to Explored;
   if isGoal(n) return SUCCESS;
   for all n' in Succ(n)
        if n' not in Explored or Frontier
        add n' to Frontier;
}
```

- Frontier implemented as queue $\longrightarrow$ breadth-first search
- Frontier implemented as stack $\longrightarrow$ depth-first search

Example breadth-first search

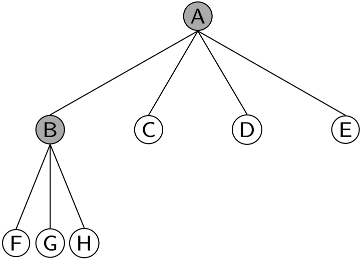
Explored	Frontier
-	A

Example breadth-first search



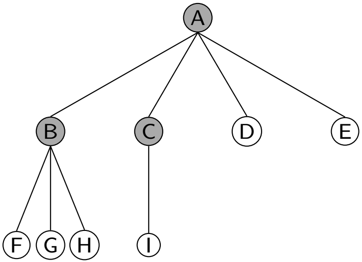
Explored	Frontier
-	A
A	B,C,D,E

Example breadth-first search



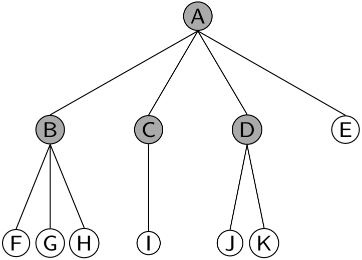
Explored	Frontier
-	A
A	B,C,D,E
A,B	C,D,E,F,G,H

Example breadth-first search



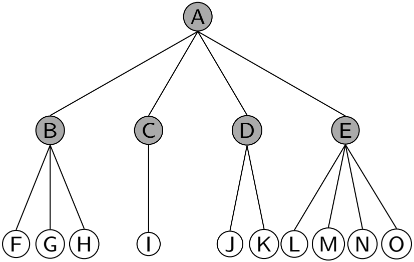
Explored	Frontier
-	A
A	B,C,D,E
A,B	C,D,E,F,G,H
A,B,C	D,E,F,G,H,I

Example breadth-first search



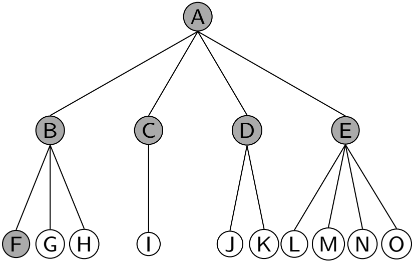
Explored	Frontier
-	A
A	B,C,D,E
A,B	C,D,E,F,G,H
A,B,C	D,E,F,G,H,I
A,B,C,D	E,F,G,H,I,J,K

Example breadth-first search



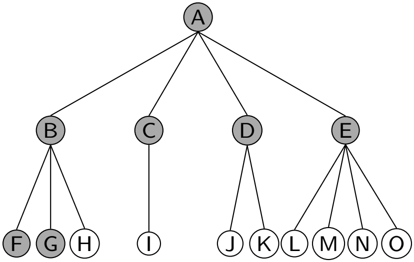
Explored	Frontier
-	A
A	B,C,D,E
A,B	C,D,E,F,G,H
A,B,C	D,E,F,G,H,I
A,B,C,D	E,F,G,H,I,J,K
A,B,C,D,E	F,G,H,I,J,K,L,M,N,O

Example breadth-first search



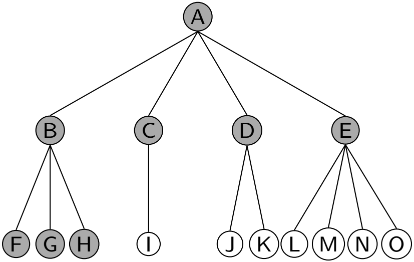
Explored	Frontier
-	A
A	B,C,D,E
A,B	C,D,E,F,G,H
A,B,C	D,E,F,G,H,I
A,B,C,D	E,F,G,H,I,J,K
A,B,C,D,E	F,G,H,I,J,K,L,M,N,O
A,B,C,D,E,F	G,H,I,J,K,L,M,N,O

Example breadth-first search



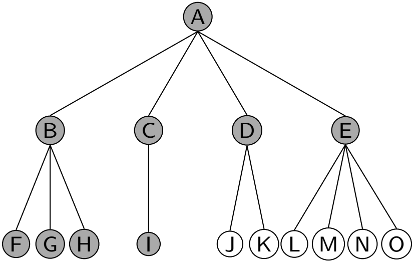
Explored	Frontier
-	A
A	B,C,D,E
A,B	C,D,E,F,G,H
A,B,C	D,E,F,G,H,I
A,B,C,D	E,F,G,H,I,J,K
A,B,C,D,E	F,G,H,I,J,K,L,M,N,O
A,B,C,D,E,F	G,H,I,J,K,L,M,N,O
A,B,C,D,E,F,G	H,I,J,K,L,M,N,O

Example breadth-first search



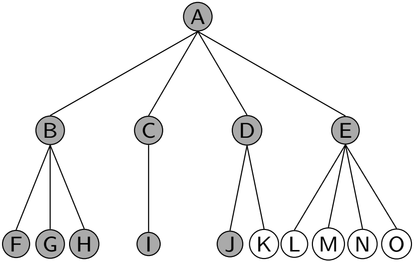
Explored	Frontier
-	A
A	B,C,D,E
A,B	C,D,E,F,G,H
A,B,C	D,E,F,G,H,I
A,B,C,D	E,F,G,H,I,J,K
A,B,C,D,E	F,G,H,I,J,K,L,M,N,O
A,B,C,D,E,F	G,H,I,J,K,L,M,N,O
A,B,C,D,E,F,G	H,I,J,K,L,M,N,O
A,B,C,D,E,F,G,H	I,J,K,L,M,N,O

Example breadth-first search



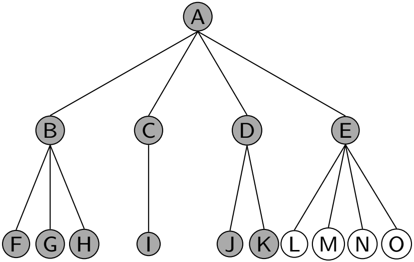
Explored	Frontier
-	A
A	B,C,D,E
A,B	C,D,E,F,G,H
A,B,C	D,E,F,G,H,I
A,B,C,D	E,F,G,H,I,J,K
A,B,C,D,E	F,G,H,I,J,K,L,M,N,O
A,B,C,D,E,F	G,H,I,J,K,L,M,N,O
A,B,C,D,E,F,G	H,I,J,K,L,M,N,O
A,B,C,D,E,F,G,H	I,J,K,L,M,N,O
A,B,C,D,E,F,G,H,I	J,K,L,M,N,O

Example breadth-first search



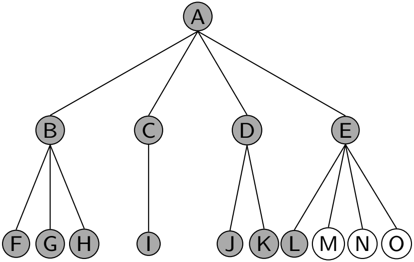
Explored	Frontier
-	A
A	B,C,D,E
A,B	C,D,E,F,G,H
A,B,C	D,E,F,G,H,I
A,B,C,D	E,F,G,H,I,J,K
A,B,C,D,E	F,G,H,I,J,K,L,M,N,O
A,B,C,D,E,F	G,H,I,J,K,L,M,N,O
A,B,C,D,E,F,G	H,I,J,K,L,M,N,O
A,B,C,D,E,F,G,H	I,J,K,L,M,N,O
A,B,C,D,E,F,G,H,I	J,K,L,M,N,O
A,B,C,D,E,F,G,H,I,J	K,L,M,N,O

Example breadth-first search



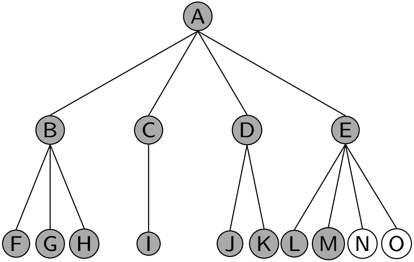
Explored	Frontier
-	A
A	B,C,D,E
A,B	C,D,E,F,G,H
A,B,C	D,E,F,G,H,I
A,B,C,D	E,F,G,H,I,J,K
A,B,C,D,E	F,G,H,I,J,K,L,M,N,O
A,B,C,D,E,F	G,H,I,J,K,L,M,N,O
A,B,C,D,E,F,G	H,I,J,K,L,M,N,O
A,B,C,D,E,F,G,H	I,J,K,L,M,N,O
A,B,C,D,E,F,G,H,I	J,K,L,M,N,O
A,B,C,D,E,F,G,H,I,J	K,L,M,N,O
A,B,C,D,E,F,G,H,I,J,K	L,M,N,O

Example breadth-first search



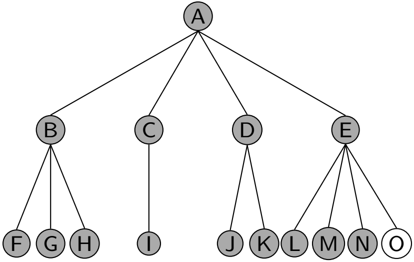
Explored	Frontier
-	A
A	B,C,D,E
A,B	C,D,E,F,G,H
A,B,C	D,E,F,G,H,I
A,B,C,D	E,F,G,H,I,J,K
A,B,C,D,E	F,G,H,I,J,K,L,M,N,O
A,B,C,D,E,F	G,H,I,J,K,L,M,N,O
A,B,C,D,E,F,G	H,I,J,K,L,M,N,O
A,B,C,D,E,F,G,H	I,J,K,L,M,N,O
A,B,C,D,E,F,G,H,I	J,K,L,M,N,O
A,B,C,D,E,F,G,H,I,J	K,L,M,N,O
A,B,C,D,E,F,G,H,I,J,K	L,M,N,O
A,B,C,D,E,F,G,H,I,J,K,L	M,N,O

Example breadth-first search



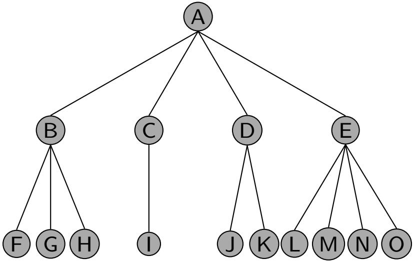
Explored	Frontier
-	A
A	B,C,D,E
A,B	C,D,E,F,G,H
A,B,C	D,E,F,G,H,I
A,B,C,D	E,F,G,H,I,J,K
A,B,C,D,E	F,G,H,I,J,K,L,M,N,O
A,B,C,D,E,F	G,H,I,J,K,L,M,N,O
A,B,C,D,E,F,G	H,I,J,K,L,M,N,O
A,B,C,D,E,F,G,H	I,J,K,L,M,N,O
A,B,C,D,E,F,G,H,I	J,K,L,M,N,O
A,B,C,D,E,F,G,H,I,J	K,L,M,N,O
A,B,C,D,E,F,G,H,I,J,K	L,M,N,O
A,B,C,D,E,F,G,H,I,J,K,L	M,N,O
A,B,C,D,E,F,G,H,I,J,K,L,M	N,O

Example breadth-first search



Explored	Frontier
-	A
A	B,C,D,E
A,B	C,D,E,F,G,H
A,B,C	D,E,F,G,H,I
A,B,C,D	E,F,G,H,I,J,K
A,B,C,D,E	F,G,H,I,J,K,L,M,N,O
A,B,C,D,E,F	G,H,I,J,K,L,M,N,O
A,B,C,D,E,F,G	H,I,J,K,L,M,N,O
A,B,C,D,E,F,G,H	I,J,K,L,M,N,O
A,B,C,D,E,F,G,H,I	J,K,L,M,N,O
A,B,C,D,E,F,G,H,I,J	K,L,M,N,O
A,B,C,D,E,F,G,H,I,J,K	L,M,N,O
A,B,C,D,E,F,G,H,I,J,K,L	M,N,O
A,B,C,D,E,F,G,H,I,J,K,L,M	N,O
A,B,C,D,E,F,G,H,I,J,K,L,M,N	O

Example breadth-first search



Explored	Frontier
-	A
A	B,C,D,E
A,B	C,D,E,F,G,H
A,B,C	D,E,F,G,H,I
A,B,C,D	E,F,G,H,I,J,K
A,B,C,D,E	F,G,H,I,J,K,L,M,N,O
A,B,C,D,E,F	G,H,I,J,K,L,M,N,O
A,B,C,D,E,F,G	H,I,J,K,L,M,N,O
A,B,C,D,E,F,G,H	I,J,K,L,M,N,O
A,B,C,D,E,F,G,H,I	J,K,L,M,N,O
A,B,C,D,E,F,G,H,I,J	K,L,M,N,O
A,B,C,D,E,F,G,H,I,J,K	L,M,N,O
A,B,C,D,E,F,G,H,I,J,K,L	M,N,O
A,B,C,D,E,F,G,H,I,J,K,L,M	N,O
A,B,C,D,E,F,G,H,I,J,K,L,M,N	O
A,B,C,D,E,F,G,H,I,J,K,L,M,N,O	-

Analysis breadth-first search

Let n be the maximal number of successors of any one node,
e the distance to the goal node that is found first, and
d the maximal search depth
(=maximal distance of a childless node)

Theorem

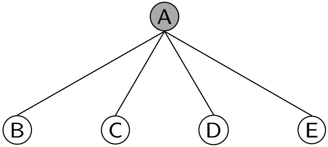
Breadth-first search is complete, has runtime complexity of $O(n^{e+1})$, space complexity of $O(n^{e+1})$ and is optimal.

Example depth-first search

Ⓐ

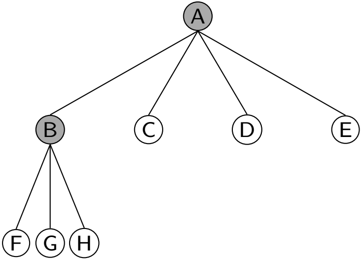
Explored	Frontier
-	A

Example depth-first search



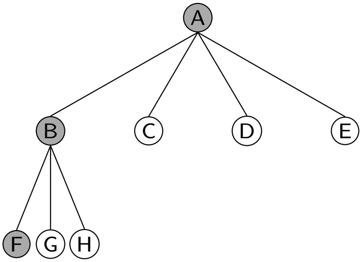
Explored	Frontier
-	A
A	B,C,D,E

Example depth-first search



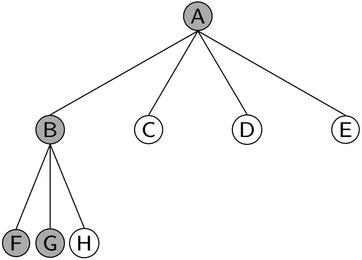
Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E

Example depth-first search



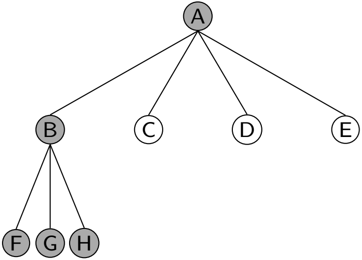
Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E

Example depth-first search



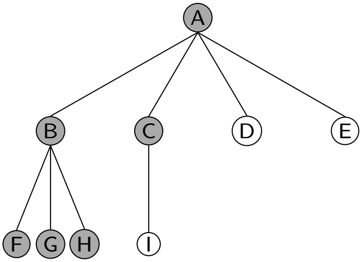
Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E

Example depth-first search



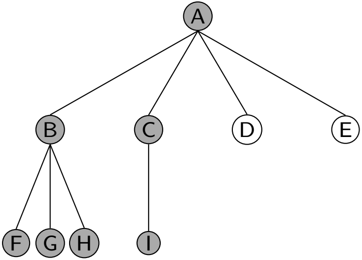
Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E

Example depth-first search



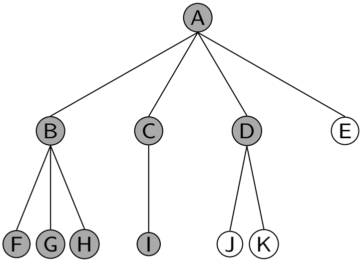
Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E
A,B,F,G,H,C	I,D,E

Example depth-first search



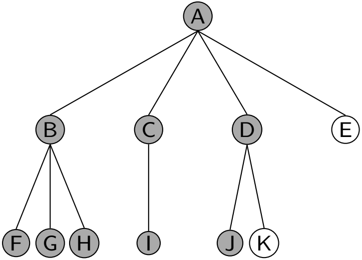
Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E
A,B,F,G,H,C	I,D,E
A,B,F,G,H,C,I	D,E

Example depth-first search



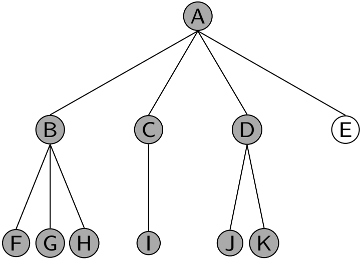
Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E
A,B,F,G,H,C	I,D,E
A,B,F,G,H,C,I,D	J,K,E

Example depth-first search



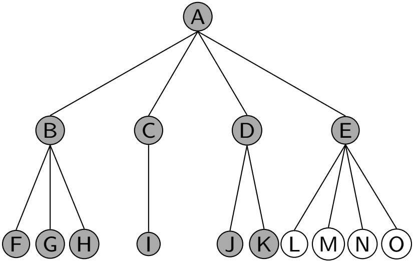
Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E
A,B,F,G,H,C	I,D,E
A,B,F,G,H,C,I,D,J	K,E

Example depth-first search



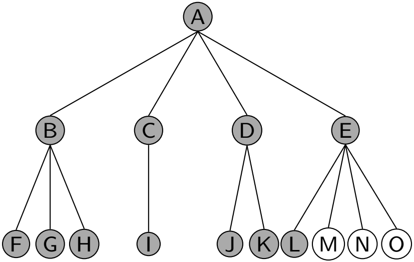
Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E
A,B,F,G,H,C	I,D,E
A,B,F,G,H,C,I,D,J	K,E
A,B,F,G,H,C,I,D,J,K	E

Example depth-first search



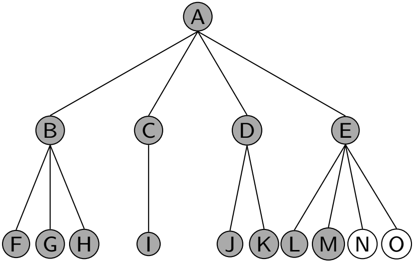
Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E
A,B,F,G,H,C	I,D,E
A,B,F,G,H,C,I,D,J	K,E
A,B,F,G,H,C,I,D,J,K,E	L,M,N,O

Example depth-first search



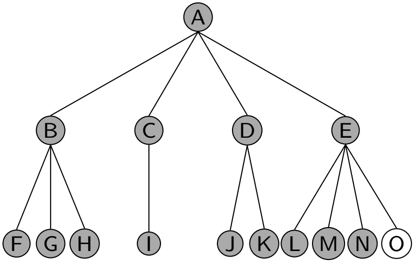
Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E
A,B,F,G,H,C	I,D,E
A,B,F,G,H,C,I,D,J	K,E
A,B,F,G,H,C,I,D,J,K,E	L,M,N,O
A,B,F,G,H,C,I,D,J,K,E,L	M,N,O

Example depth-first search



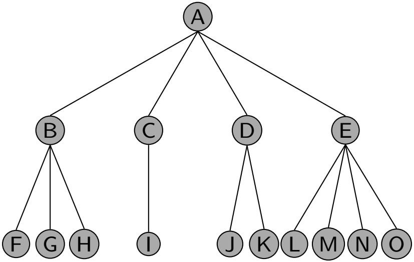
Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E
A,B,F,G,H,C	I,D,E
A,B,F,G,H,C,I,D,J	K,E
A,B,F,G,H,C,I,D,J,K,E	L,M,N,O
A,B,F,G,H,C,I,D,J,K,E,L	M,N,O
A,B,F,G,H,C,I,D,J,K,E,L,M	N,O

Example depth-first search



Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E
A,B,F,G,H,C	I,D,E
A,B,F,G,H,C,I,D,J	K,E
A,B,F,G,H,C,I,D,J,K,E	L,M,N,O
A,B,F,G,H,C,I,D,J,K,E,L	M,N,O
A,B,F,G,H,C,I,D,J,K,E,L,M	N,O
A,B,F,G,H,C,I,D,J,K,E,L,M,N	O

Example depth-first search



Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E
A,B,F,G,H,C	I,D,E
A,B,F,G,H,C,I,D,J	K,E
A,B,F,G,H,C,I,D,J,K,E	L,M,N,O
A,B,F,G,H,C,I,D,J,K,E,L	M,N,O
A,B,F,G,H,C,I,D,J,K,E,L,M	N,O
A,B,F,G,H,C,I,D,J,K,E,L,M,N,O	-

Analysis depth-first search

Let n be the maximal number of successors of any one node,
e the distance to the goal node that is found first, and
d the maximal search depth
(=maximal distance of a childless node)

Theorem

Depth-first search is complete (for finite d), has runtime complexity of $O(n^d)$, space complexity of $O(dn)$, and is not optimal.

What happens if d is infinite (and what is an example of such a problem)?

Why is depth-first search not optimal?

Breadth-first search in Prolog

bfs.pl:

```
edge(a,b). edge(a,c). edge(a,d). edge(a,e). edge(b,f).
edge(b,g). edge(b,h). edge(c,i). edge(d,j). edge(d,k).
edge(e,l). edge(e,m). edge(e,n). edge(e,o).

bfs(Goals, [X|_], _) :- member(X, Goals), !, write(X).
bfs(Goals, [X|RestFrontier], Explored) :-
   member(X, Explored),
   !,
   bfs(Goals, RestFrontier, Explored).
bfs(Goals, [X|RestFrontier], Explored) :-
   write(X),
   succ(X, L),
   append(RestFrontier, L, Frontier),
   bfs(Goals, Frontier, [X|Explored]).

succ(X, L) :- findall(Y, edge(X,Y), L).
```

```
?- bfs([k],[a],[]).
abcdefghijk
```

Depth-first search in Prolog

dfs.pl:

```
edge(a,b). edge(a,c). edge(a,d). edge(a,e). edge(b,f).
edge(b,g). edge(b,h). edge(c,i). edge(d,j). edge(d,k).
edge(e,l). edge(e,m). edge(e,n). edge(e,o).

dfs(Goals, [X|_], _) :- member(X, Goals), !, write(X).
dfs(Goals, [X|RestFrontier], Explored) :-
   member(X, Explored),
   !,
   dfs(Goals, RestFrontier, Explored).
dfs(Goals, [X|RestFrontier], Explored) :-
   write(X),
   succ(X, L),
   append(L, RestFrontier, Frontier),
   dfs(Goals, Frontier, [X|Explored]).

succ(X, L) :- findall(Y, edge(X,Y), L).
```

```
?- dfs([k],[a],[]).
abfghcidjk
```

Iterative depth-first search 1/2

- Advantage of depth-first search (disadvantage of breadth-first search): we have to store few (many) nodes in memory
- Advantage of breadth-first search (disadvantage of depth-first search): we only explore paths of sufficient lengths (it may be that we take many detours even if a goal node is quite close)

Idee: Combine advantages in *iterative depth-first search*

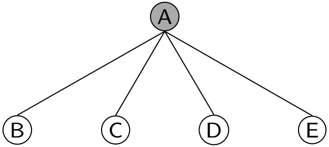
1. Do depth-first search only up to a certain depth T (initially 1)
2. If we find a goal node, we terminate
3. Otherwise increment T by 1 and go to step 1.

Example iterative depth-first search

$T = 1$

Explored	Frontier
-	A

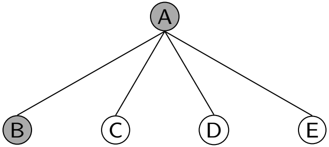
Example iterative depth-first search



$T = 1$

Explored	Frontier
-	A
A	B,C,D,E

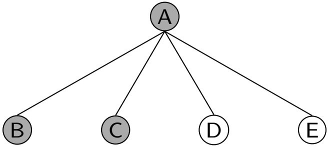
Example iterative depth-first search



$T = 1$

Explored	Frontier
-	A
A	B,C,D,E
A,B	C,D,E

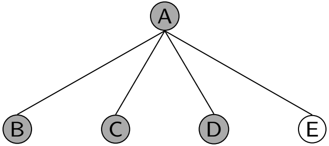
Example iterative depth-first search



$T = 1$

Explored	Frontier
-	A
A	B,C,D,E
A,B	C,D,E
A,B,C	D,E

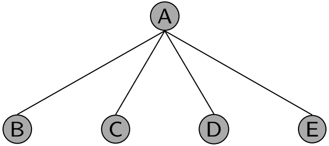
Example iterative depth-first search



$T = 1$

Explored	Frontier
-	A
A	B,C,D,E
A,B	C,D,E
A,B,C	D,E
A,B,C,D	E

Example iterative depth-first search



$T = 1$

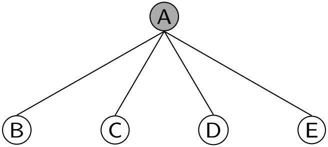
Explored	Frontier
-	A
A	B,C,D,E
A,B	C,D,E
A,B,C	D,E
A,B,C,D,E	-

Example iterative depth-first search

$T = 2$

Explored	Frontier
-	A

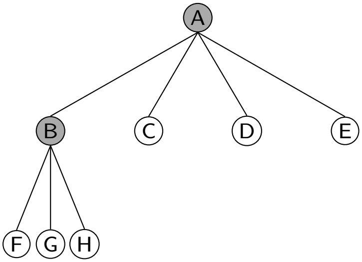
Example iterative depth-first search



$T = 2$

Explored	Frontier
-	A
A	B,C,D,E

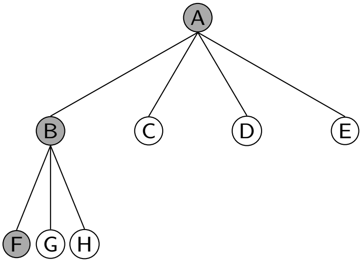
Example iterative depth-first search



$T = 2$

Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E

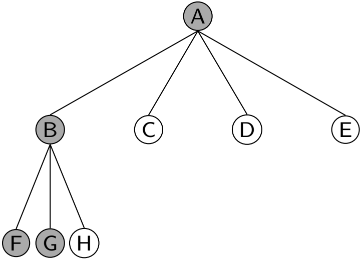
Example iterative depth-first search



$T = 2$

Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E

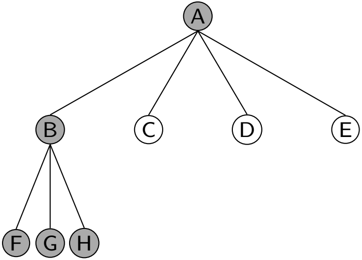
Example iterative depth-first search



$T = 2$

Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E

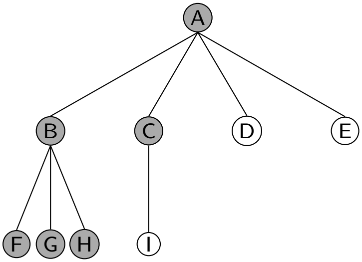
Example iterative depth-first search



$T = 2$

Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E

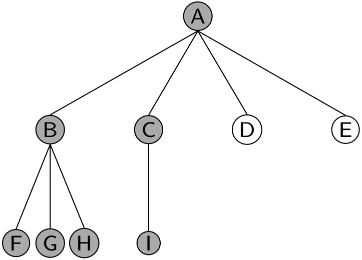
Example iterative depth-first search



$T = 2$

Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E
A,B,F,G,H,C	I,D,E

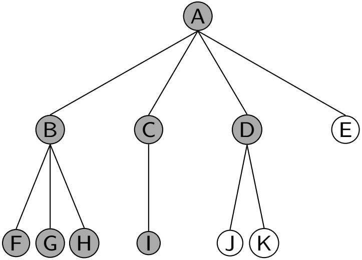
Example iterative depth-first search



$T = 2$

Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E
A,B,F,G,H,C	I,D,E
A,B,F,G,H,C,I	D,E

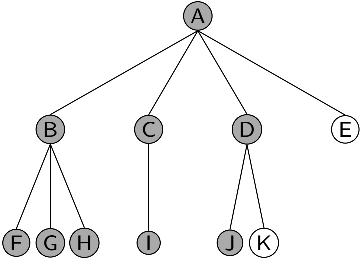
Example iterative depth-first search



$T = 2$

Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E
A,B,F,G,H,C	I,D,E
A,B,F,G,H,C,I,D	J,K,E

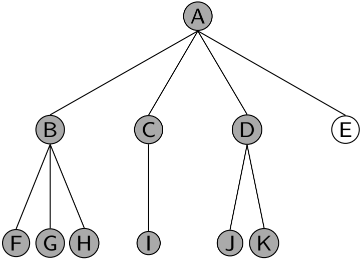
Example iterative depth-first search



$T = 2$

Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E
A,B,F,G,H,C	I,D,E
A,B,F,G,H,C,I,D,J	K,E

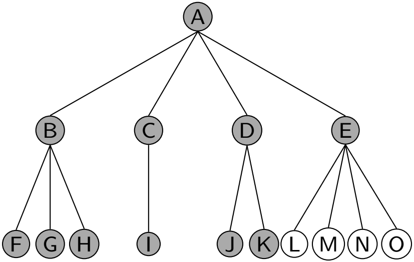
Example iterative depth-first search



$T = 2$

Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E
A,B,F,G,H,C	I,D,E
A,B,F,G,H,C,I,D,J	K,E
A,B,F,G,H,C,I,D,J,K	E

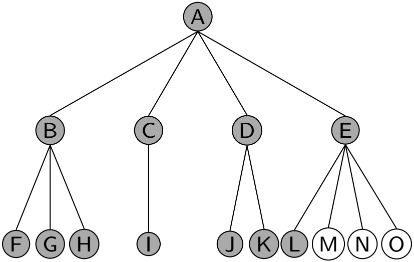
Example iterative depth-first search



$T = 2$

Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E
A,B,F,G,H,C	I,D,E
A,B,F,G,H,C,I,D,J	K,E
A,B,F,G,H,C,I,D,J,K,E	L,M,N,O

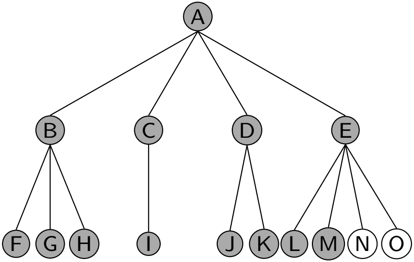
Example iterative depth-first search



$T = 2$

Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E
A,B,F,G,H,C	I,D,E
A,B,F,G,H,C,I,D,J	K,E
A,B,F,G,H,C,I,D,J,K,E	L,M,N,O
A,B,F,G,H,C,I,D,J,K,E,L	M,N,O

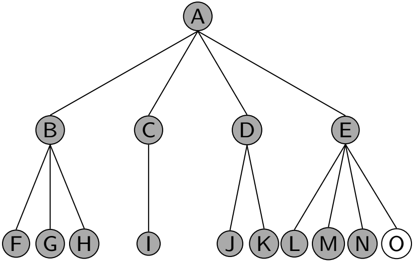
Example iterative depth-first search



$T = 2$

Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E
A,B,F,G,H,C	I,D,E
A,B,F,G,H,C,I,D,J	K,E
A,B,F,G,H,C,I,D,J,K,E	L,M,N,O
A,B,F,G,H,C,I,D,J,K,E,L	M,N,O
A,B,F,G,H,C,I,D,J,K,E,L,M	N,O

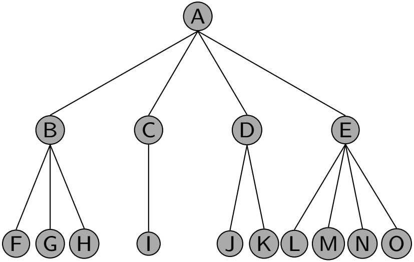
Example iterative depth-first search



$T = 2$

Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E
A,B,F,G,H,C	I,D,E
A,B,F,G,H,C,I,D,J	K,E
A,B,F,G,H,C,I,D,J,K,E	L,M,N,O
A,B,F,G,H,C,I,D,J,K,E,L	M,N,O
A,B,F,G,H,C,I,D,J,K,E,L,M	N,O
A,B,F,G,H,C,I,D,J,K,E,L,M,N	O

Example iterative depth-first search



$T = 2$

Explored	Frontier
-	A
A	B,C,D,E
A,B	F,G,H,C,D,E
A,B,F	G,H,C,D,E
A,B,F,G	H,C,D,E
A,B,F,G,H	C,D,E
A,B,F,G,H,C	I,D,E
A,B,F,G,H,C,I,D,J	K,E
A,B,F,G,H,C,I,D,J,K,E	L,M,N,O
A,B,F,G,H,C,I,D,J,K,E,L	M,N,O
A,B,F,G,H,C,I,D,J,K,E,L,M	N,O
A,B,F,G,H,C,I,D,J,K,E,L,M,N,O	-

Iterative depth-first search 2/2

Let n be the maximal number of successors of any one node,
e the distance to the goal node that is found first, and
d the maximal search depth
(=maximal distance of a childless node)

Theorem

Iterative depth-first search is complete, has runtime complexity $O(n^{e+1})$ (like breadth-first search), space complexity $O(dn)$ (like depth-first search) and is optimal.

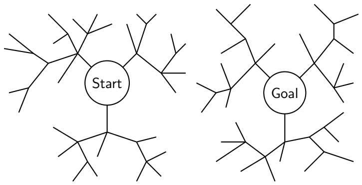
Disadvantage of depth-first search: All nodes that have been explored at depth T will be re-explored at depth $T+1$. However, asymptotically this has no influence on the runtime (this is a constant factor in the O notation).

Bidirectional search 1/2

Idea: Search from the start and the goal node in parallel (goal node should be uniquely determined).

- Forward search from the start node (breadth-first search)
- Backward search from the goal node (breadth-first search)

Whenever a node is added to `Frontier` in either search, check whether this node is already in `Frontier` of the other search. If this is the case we found a path to the goal using that node.



Bidirectional search 2/2

Let n be the maximal number of successors of any one node,
e the distance to the goal node that is found first, and
d the maximal search depth
(=maximal distance of a childless node)

Theorem

Bidirectional search is complete, has runtime complexity $O(n^{(e+1)/2})$, *space complexity* $O(n^e)$ *and is optimal.*

Problem for bidirectional search: determining the predecessor of a node (for backward search) may be non-trivial.

Chapter 3.1: Uninformed Search

Summary

Chapter 3.1: Summary

- Uninformed Search: we have no further information besides the state transition relation
- General strategy: extend paths incrementally starting from the start node
- Search strategies
  - Depth-first search
  - Breadth-first search
  - Iterative depth-first search
  - Bidirectional search