Basics of AI and Machine Learning

State-Space Search: Depth-first Search

Jendrik Seipp

Linköping University

Depth-first Search

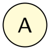
Depth-first Search

Depth-first search (DFS) expands nodes in opposite order of generation (LIFO).

$\rightsquigarrow$ deepest node expanded first

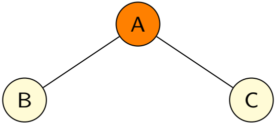
$\rightsquigarrow$ open list implemented as stack

Depth-first Search: Example



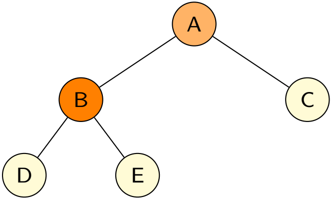
open: A

Depth-first Search: Example



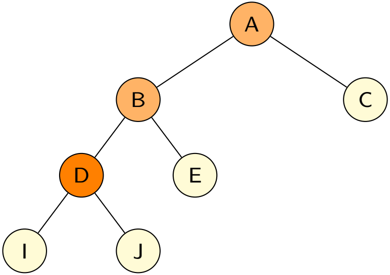
open: C, B

Depth-first Search: Example



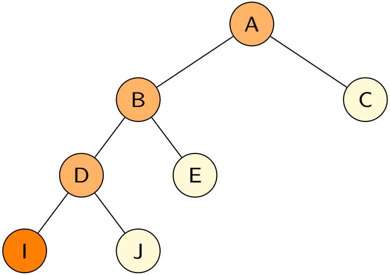
open: C, E, D

Depth-first Search: Example



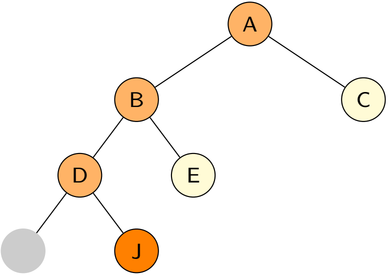
open: C, E, J, I

Depth-first Search: Example



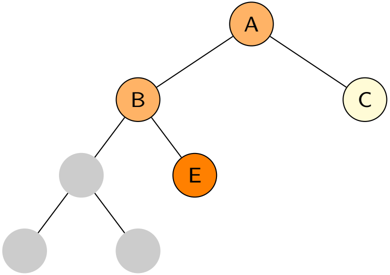
open: C, E, J

Depth-first Search: Example



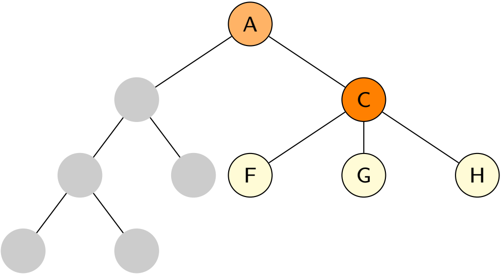
open: C, E

Depth-first Search: Example



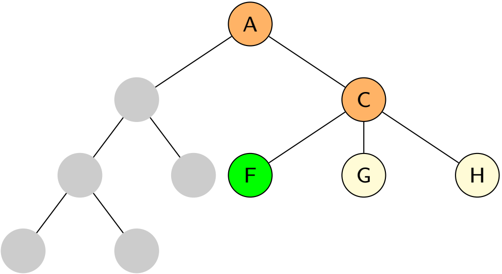
open: C

Depth-first Search: Example



open: H, G, F

Depth-first Search: Example



⇝ solution found!

Depth-first Search: Some Properties

- almost always implemented as a tree search (we will see why)
- not complete, not semi-complete, not optimal: cycles, does not explore in layers
- complete for acyclic state spaces,
  e.g., if state space directed tree

Depth-first Search (Non-recursive Version)

depth-first search (non-recursive version):

Depth-first Search (Non-recursive Version)

open := **new** Stack
open.push_back(make_root_node())
while not *open*.is_empty():
    n := *open*.pop_back()
    if is_goal(n.state):
        return extract_path(n)
    for each $\langle a, s' \rangle \in$ succ(n.state):
        n' := make_node(n, a, s')
        open.push_back(n')
return unsolvable

Non-recursive Depth-first Search: Discussion

discussion:

- there isn't much wrong with this pseudo-code
  (as long as we ensure to release nodes that are no longer required when using programming languages without garbage collection)
- however, depth-first search as a recursive algorithm is simpler and more efficient

$\rightsquigarrow$ CPU stack as implicit open list

$\rightsquigarrow$ no search node data structure needed

Depth-first Search (Recursive Version)

function depth_first_search(s)

```
if is_goal(s):
    return ⟨⟩
for each ⟨a, s'⟩ ∈ succ(s):
    solution := depth_first_search(s')
    if solution ≠ none:
        solution.push_front(a)
        return solution
return none
```

main function:

Depth-first Search (Recursive Version)

```
return depth_first_search(init())
```

Summary

Summary

depth-first search: expand nodes in LIFO order

- usually as a tree search
- easy to implement recursively
- very memory-efficient

Comparison of Blind Search Algorithms

completeness, optimality, time and space complexity

	search algorithm				
criterion	breadth-first	uniform cost	depth-first	depth-bounded	iterative deepening
complete?	yes*	yes	no	no	semi
optimal?	yes**	yes	no	no	yes**
time	$O(b^d)$	$O(b^{\lfloor c^*/\varepsilon \rfloor + 1})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$
space	$O(b^d)$	$O(b^{\lfloor c^*/\varepsilon \rfloor + 1})$	$O(bm)$	$O(b\ell)$	$O(bd)$

$b \geq 2$	branching factor
d	minimal solution depth
m	maximal search depth
ℓ	depth bound
c^*	optimal solution cost
$\varepsilon > 0$	minimal action cost

remarks:

* for BFS-Tree: semi-complete

** only with uniform action costs