Depth-first search

Alfons Juan
Jorge Civera

Departament de Sistemes
Informàtics i Computació

Learning objectives

- To analyze depth-first search.
- To describe depth-first search in its *backtracking* variant.
- To apply iterative deepening search.

Contents

1 Introduction 3

2 Depth-first search 4

3 Backtracking 6

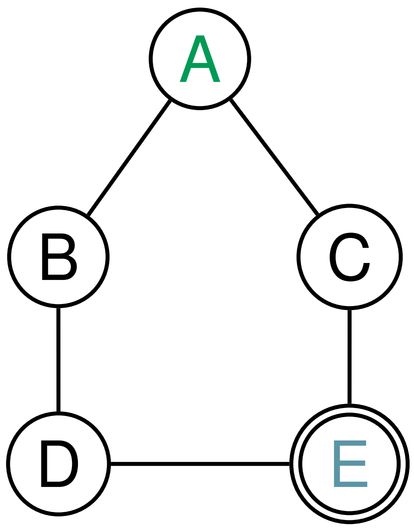
4 Iterative deepening search 8

5 Conclusions 9

1 Introduction

Depth-first search (DFS) enumerates paths (from the source node) until finding a solution (target node) by prioritizing the deepest (longest) paths and limiting the maximum depth (only finite paths):

State graph



Search (limited)

A
AB
AC
ABA
ABD
ABAB
ABAC
ABDB
ABDE sol.!

Note: closed nodes are not stored for efficiency.

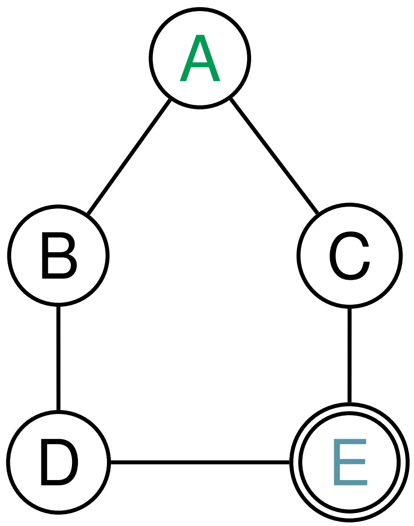
2 Depth-first search [1, 2]

```
DFS(G, s', m)              // Depth-first search with maximum depth of m
  O = InitStack(s')                       // Open: search frontier-stack
  while not EmptyStack(O):
    s = Pop(O)                    // selection LIFO (Last in, first out)
    if Goal(s) return s                               // solution found!
    if Depth(s) < m:                          // maximum depth not reached
      forall (s, n) ∈ Adjacents(G, s):        // generation: n child of s
        Push(O, n)                                // n added to the stack
  return NULL                                        // no solution found
```

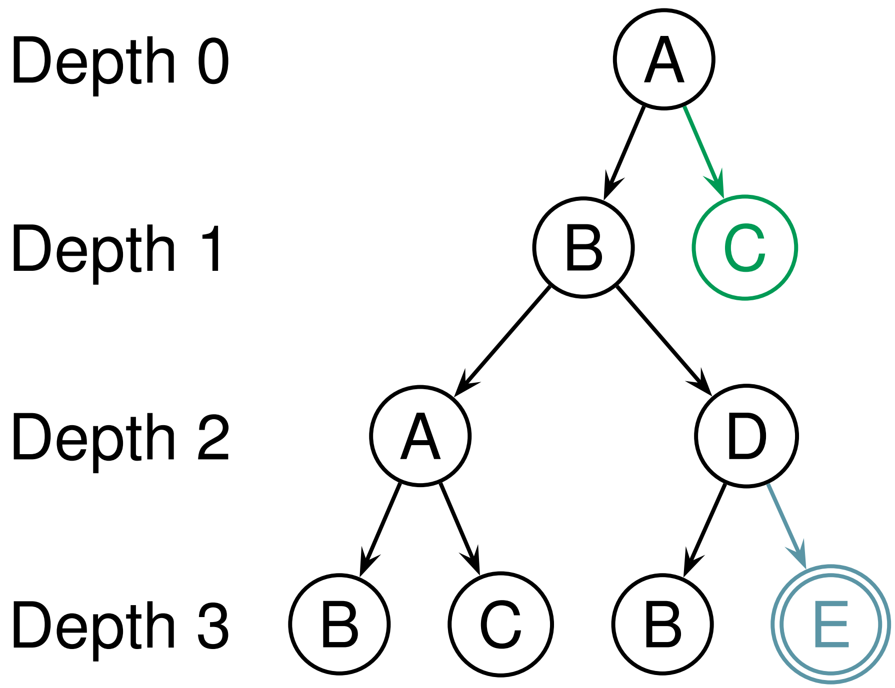

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Depth-first search tree ($m = 3$)

State graph



Search tree



It. 8: ABABCDBEC E sol.!

Quality: incomplete and suboptimal.

Complexity: $O(b^m)$ time and $O(bm)$ space.

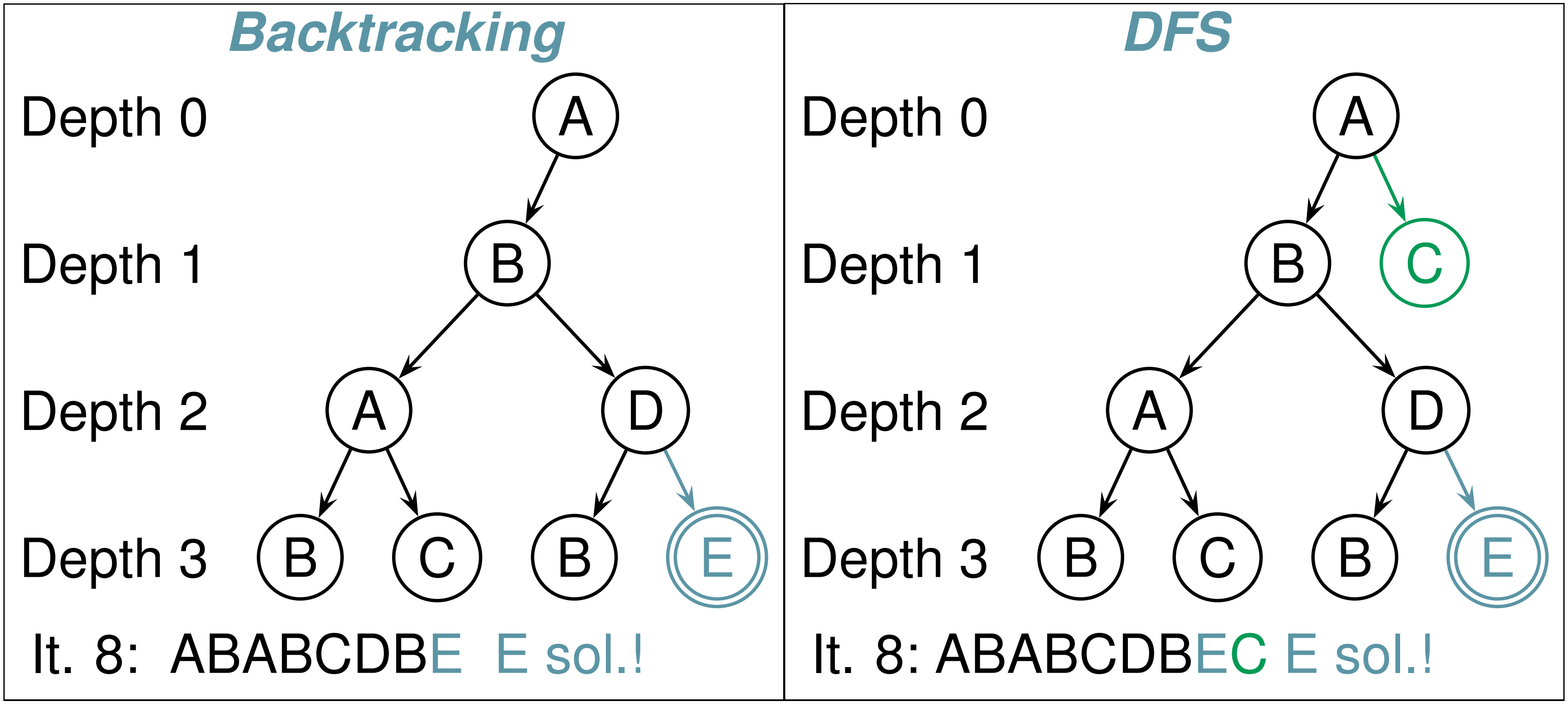
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

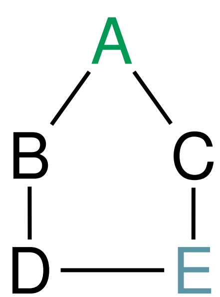
3 Backtracking

DFS (recursive) variant with individual child generation:

```
BT(G, s, m)                                  // Backtracking with maximum depth of m
  if Goal(s) return s                        // solution found!
  if m = 0 return NULL                       // maximum depth reached
  n = FirstAdjacent(G, s)                    // generation: n first child of s
  while n ≠ NULL:
    r = BT(G, n, m − 1)                      // current child result
    if r ≠ NULL: return r                    // if r is solution, stop
    n = NextAdjacent(G, s, n)                // generation: n next child of s
  return NULL                                // no solution found
```

Backtracking search tree ($m = 3$)





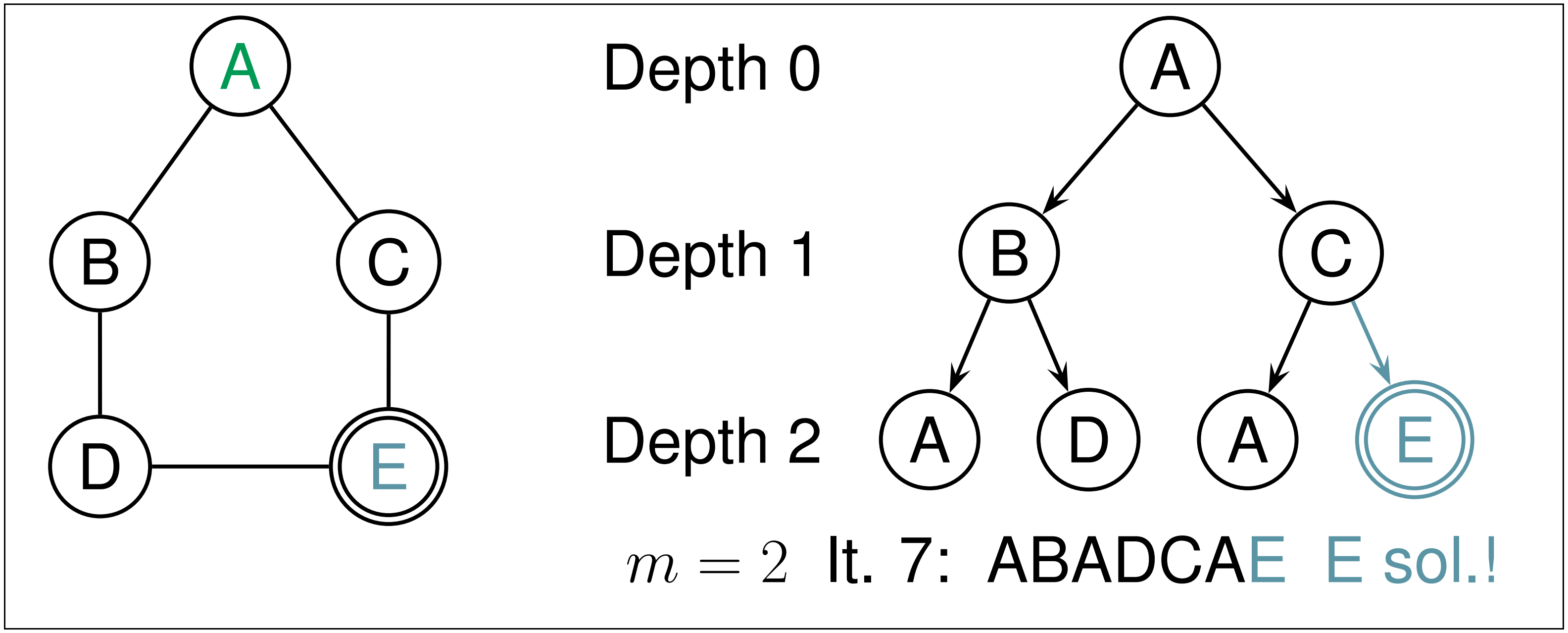
Quality: incomplete and suboptimal.

Time complexity: $O(b^m)$.

Space complexity: $O(m)$, better than DFS $O(bm)$.

4 Iterative deepening search [3]

IDS(G, s) // ***Iterative deepening search***
  for $m = 0, 1, 2, \ldots$: **if** $(r = \text{DFS}(G, s, m)) \neq$ NULL: **return** r



Quality: complete and optimal with actions of equal cost.

Complexity: $O(b^d)$ time and $O(bd)$ space.

5 Conclusions

Topics covered:

- Depth-first search iterative version.
- The *backtracking* DFS variant.
- A well-known DFS extension called iterative deepening search.

Highlights:

- Incomplete and suboptimal.
- Reasonable space complexity, specially with *backtracking*.
- It may be a good approach to search for deep solutions, specially when the path length (cost) is not very relevant.
- The iterative deepening search is complete, optimal for equal cost edges, and has a reasonable space complexity; it is thus generally preferable to BFS.

References

[1] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, third edition, 2010.

[2] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 2018.

[3] R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 1985.

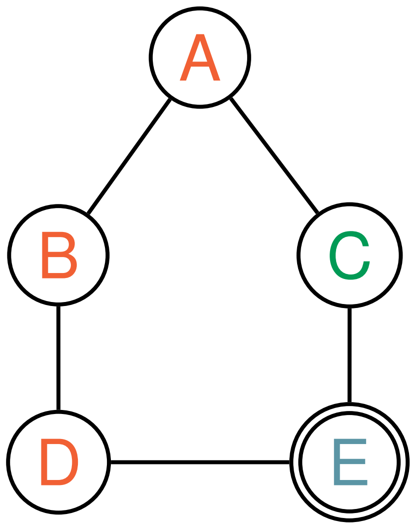
Depth-first search (graph search) [1, 2]

Graph search: keeps track of explored nodes in a set C.

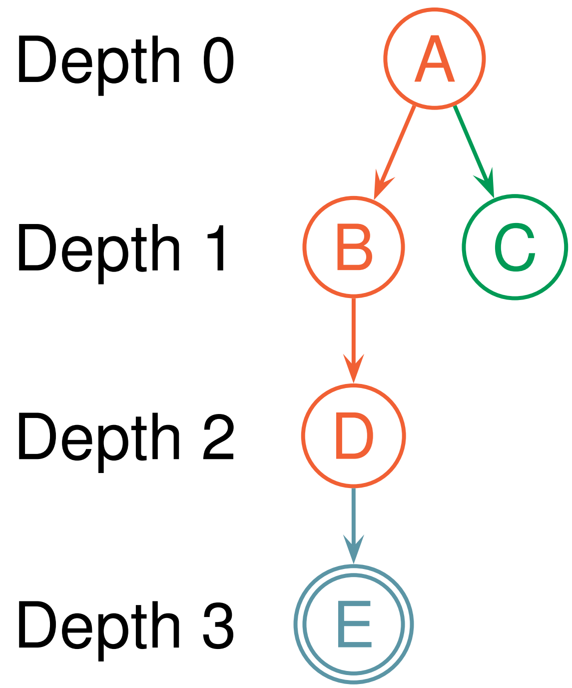
```
DFS(G, s')                    // Depth-first search; G graph and s initial node
  O = IniStack(s')            // Open: search frontier-stack
  C = ∅                       // Closed: set of explored nodes
  while not EmptyStack(O):
    s = Pop(O)                // selection LIFO (Last in, first out)
    if Goal(s) return n       // solucion found!
    C = C ∪ {s}               // s already explored
    forall (s, n) ∈ Adjacents(G, s):   // generacion: n child of s
      if n ∉ C ∪ O:           // n not found unit now
        Push(O, n)            // n is added to the stack
  return NULL                 // no solution found
```

Depth-first search (on a graph): search tree

State graph



Search tree



It. 4: ABDEC E sol.!

Properties: complete, suboptimal, $O(|V|+|E|)$ time and space.

In general, worse than breadth-first search!

dfs.py

```
#!/usr/bin/env python3
G={'A':['B','C'],'B':['A','D'],'C':['A','E'],
    'D':['B','E'],'E':['C','D']}
def dfs(G,s,m,t):
    if s==t: return [s]
    if m==0: return None
    for n in G[s]:
        path=dfs(G,n,m-1,t)
        if path!=None: return [s]+path
print(dfs(G,'A',3,'E'))
```

dfs.py.out

```
['A', 'B', 'D', 'E']
```

dfsi.py

```
#!/usr/bin/env python3
from collections import deque
G={'A':['B','C'],'B':['A','D'],'C':['A','E'],
    'D':['B','E'],'E':['C','D']}
def dfsi(G,s,m,t):
    O=deque(); O.append((s,[s]))
    while O:
        s,path=O.pop()
        if s==t: return path
        if len(path)<=m:
            for n in list(reversed(G[s])):
                O.append((n,path+[n]))
print(dfsi(G,'A',3,'E'))
```

dfsi.py.out

```
['A', 'B', 'D', 'E']
```

bt.py

```python
#!/usr/bin/env python3
G={'A':['B','C'],'B':['A','D'],'C':['A','E'],
	'D':['B','E'],'E':['C','D']}
def bt(G,s,m,t):
	if s==t: return [s]
	if m==0: return None
	for i in [0,1]:
		n=G[s][i]; path=bt(G,n,m-1,t)
		if path!=None: return [s]+path
print(bt(G,'A',3,'E'))
```

bt.py.out

```
['A', 'B', 'D', 'E']
```

pi.py

```python
#!/usr/bin/env python3
G={'A':['B','C'],'B':['A','D'],'C':['A','E'],
	'D':['B','E'],'E':['C','D']}
def dfs(G,s,m,t):
	if s==t: return [s]
	if m==0: return None
	for n in G[s]:
		path=dfs(G,n,m-1,t)
		if path!=None: return [s]+path
def pi(G,s,t):
	for m in range(len(G)):
		path=dfs(G,s,m,t)
		if path!=None: return path
print(pi(G,'A','E'))
```

pi.py.out

```
['A', 'C', 'E']
```