Breadth-first search

Alfons Juan
Jorge Civera

Departament de Sistemes
Informàtics i Computació

Learning objectives

- To describe breadth-first search.
- To build the breadth-first search tree.
- To analyze the quality and complexity of breadth-first search.

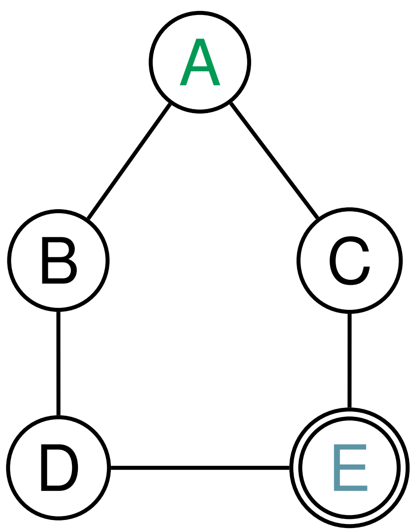
Contents

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

1 Introduction

Breadth-first search (BFS) enumerates paths (from the source node) until finding a solution (target node) by prioritizing those paths with minimum length and avoiding cycles (without repeating nodes).

State graph



Search

A

AB
AC

ABD
ACE sol.!

2 Breadth-first search [1, 2, 3, 4]

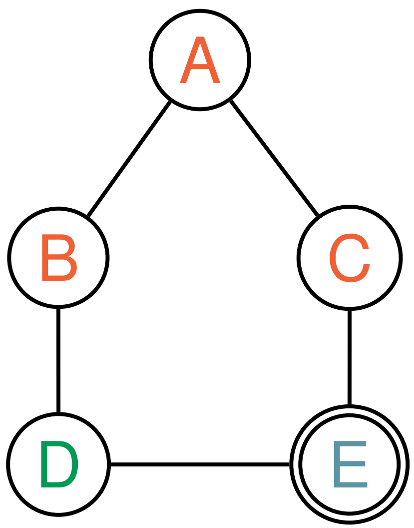
```
BFS(G, s')                    // Breadth-first search; G graph and s' initial node
  O = InitQueue(s')                         // Open: search frontier-queue
  C = ∅                                     // Closed: set of explored nodes
  while not EmptyQueue(O):
    s = Unqueue(O)                          // FIFO (First in, first out) selection
    C = C ∪ {s}                             // s is now explored
    forall (s, n) ∈ Adjacents(G, s):        // generation: n child of s
      if n ∉ C ∪ O:                         // n not discovered until now
        if Goal(n) return n                 // solution found!
        Append(O, n)                        // n added to the queue
  return NULL                               // no solution found
```


UNIVERSITAT POLITÈCNICA DE VALÈNCIA

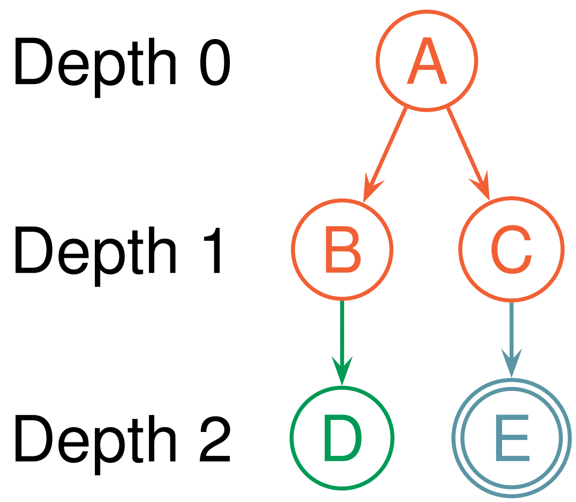
3 The breadth-first search tree

BFS produces a ***search tree*** rooted at the source node and ***depth*** d equal to the length of “the” (a) shortest path to a solution:

State graph



Search tree



It. 3: ABCD E sol.!

Note: ties solved by alphabetic order (“1st the leftmost”).

4 Completeness, optimality and complexity

- ***Completeness:*** Yes, a solution is always found (if any exists).
- ***Optimality:*** Yes, for actions of equal positive cost.
- ***Complexity:***
  - $G = (V, E)$ *explicit:* $O(|V| + |E|)$ time and space.
  - G *implicit,* with ***branching factor*** b and up to ***depth*** d***:***

    Worst case (full tree): $O(b^d)$ time and space.

    $Goal$ checked after selection: $O(b^{d+1})$ time and space.

5 Conclusions

Topics covered:

- Breadth-first search algorithm.
- Breadth-first search tree.
- Breadth-first search quality and complexity.

Highlights about BFS:

- Complete and optimal for edges of equal cost.
- Excessive space complexity, specially with deep solutions.
- It might be a good option for sparse graphs (with few edges), shallow solutions and edges of equal cost.

References

[1] E. Moore. The shortest path through a maze. In *Proc. of the Int. Symposium on the Theory of Switching, Part II*, pages 285–292. Harvard University Press, 1959.

[2] C. Y. Lee. An algorithm for path connections and its applications. *IRE Trans. on Electronic Computers*, EC-10, 1961.

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

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

bfs.py

```python
#!/usr/bin/env python3
from queue import Queue
G={'A':['B','C'],'B':['A','D'],'C':['A','E'],
    'D':['B','E'],'E':['C','D']}
def bfs(G,s,t):
	if s==t: return [s]
	O=Queue(); O.put((s,[s])) # Open queue
	OCs=set(); OCs.add(s)      # Open and closed set
	while O:
		s,path=O.get()
		for n in G[s]:
			if n not in OCs:
				if n==t: return path+[n]
				O.put((n,path+[n]))
				OCs.add(n)
print(bfs(G,'A','E'))
```

bfs.py.out

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