Operations Research

Graph Search

Safia Kedad-Sidhoum

safia.kedad_sidhoum@cnam.fr

Cnam

2025-2026

Graph Search and its Applications

- Network accessibility, checking connectivity : In a physical network, such as a network of computers, it is sometimes useful to check if you can get anywhere from anywhere else. That is, for every choice of a point A and a point B, there should be a path in the network from the former to the latter.
- Connected components and Strongly connected components : How to compute the connected components (the "pieces") of a graph for undirected and directed graphs using graph search ?
- Shortest paths : the breadth-first search naturally allows the computations of shortest paths.

Graph Search

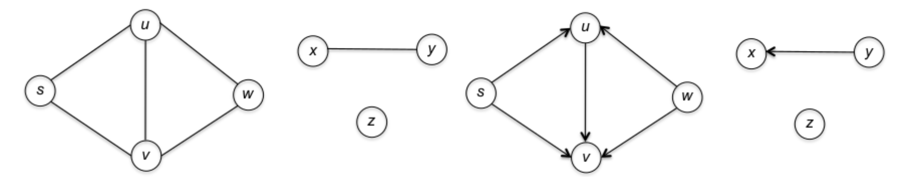
Input : An undirected or directed graph $G = (V, E)$, and a starting vertex $s \in V$.

Goal : Identify the vertices of V reachable from s in G.

A vertex v is "reachable," if there is a sequence of edges in G that travels from s to v. If G is a directed graph, all the path's edges should be traversed in the forward (outgoing) direction.

Graph Search

Example



(a) An undirected graph (b) A directed version

In (a), the set of vertices reachable from s is $\{s, u, v, w\}$. In (b), it is $\{s, u, v\}$.

Generic Search

Pseudo-code

Input: graph $G = (V, E)$ and a vertex $s \in V$.
Postcondition: a vertex is reachable from s if and only if it is marked as "explored."

---

mark s as explored, all other vertices as unexplored
while there is an edge $(v, w) \in E$ with v explored and w unexplored **do**
  choose some such edge (v, w) `// underspecified`
  mark w as explored

In the directed case, the edge (v, w) chosen in an iteration of the while loop should be directed from an explored vertex v to an unexplored vertex w.

Running time ?

Example

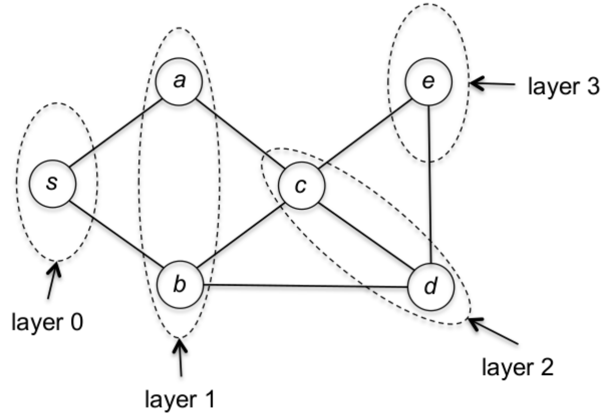
Generic search on directed and undirected graphs.

Tree search

Breadth-first search

General idea

Breadth-first search discovers vertices in layers.
The layer$-i$ vertices are the neighbors of the layer$-(i-1)$ vertices that do not appear in any earlier layer.



Breadth-first search

Pseudo-code

BFS

Input: graph $G = (V, E)$ in adjacency-list representation, and a vertex $s \in V$.
Postcondition: a vertex is reachable from s if and only if it is marked as "explored."

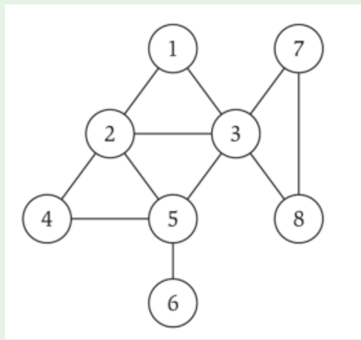
---

mark s as explored, all other vertices as unexplored
$Q :=$ a queue data structure, initialized with s
while Q is not empty **do**
  remove the vertex from the front of Q, call it v
  for each edge (v, w) in v's adjacency list **do**
    if w is unexplored **then**
      mark w as explored
      add w to the end of Q

Running time ?

Breadth-first search

Example



BFS Tree

Breadth-first search applications

Shortest paths distances

Notation : $dist(v, w)$: fewest number of edges in a path from v to w (or $+\infty$ if G contains no path from v to w.

Input : An undirected or directed graph $G = (V, E)$, and a starting vertex $s \in V$.

Output : $dist(s, v)$ for every vertex $v \in V$.

Shortest-path distances

Pseudo-code

Input: graph $G = (V, E)$ in adjacency-list representation, and a vertex $s \in V$.
Postcondition: for every vertex $v \in V$, the value $l(v)$ equals the true shortest-path distance $dist(s, v)$.

---

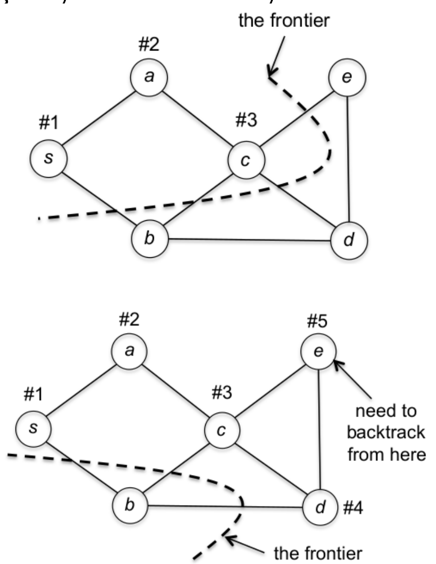
mark s as explored, all other vertices as unexplored
$l(s) := 0$, $l(v) := +\infty$ for every $v \neq s$
$Q :=$ a queue data structure, initialized with s
while Q is not empty **do**
    remove the vertex from the front of Q, call it v
    for each edge (v, w) in v's adjacency list **do**
        if w is unexplored **then**
            mark w as explored
            $l(w) := l(v) + 1$
            add w to the end of Q

Running time ?

Depth-first search

General idea and example

Depth-first search always exploring from the most recently discovered vertex and backtracking only when necessary.



Depth-first search

Pseudo-code 1/2

DFS **(Iterative Version)**

Input: graph $G = (V, E)$ in adjacency-list representation, and a vertex $s \in V$.
Postcondition: a vertex is reachable from s if and only if it is marked as "explored."

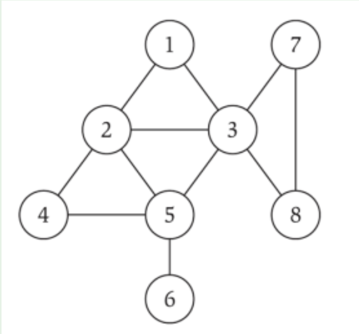
---

mark all vertices as unexplored
$S :=$ a stack data structure, initialized with s
while S is not empty **do**
    remove ("pop") the vertex v from the front of S
    if v is unexplored **then**
        mark v as explored
        for each edge (v, w) in v's adjacency list **do**
            add ("push") w to the front of S if w is unexplored

Running time ?

Depth-first search

Example



DFS Tree

Depth-first search

Pseudo-code 2/2

DFS (Recursive Version)

Input: graph $G = (V, E)$ in adjacency-list representation, and a vertex $s \in V$.
Postcondition: a vertex is reachable from s if and only if it is marked as "explored."

```
// all vertices unexplored before outer call
mark s as explored
for each edge (s, v) in s's adjacency list do
    if v is unexplored then
        DFS (G, v)
```


Depth-first search

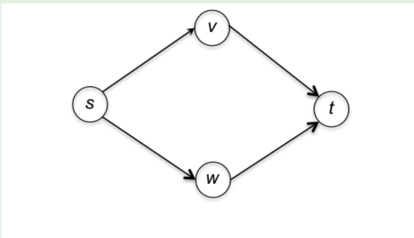
Application

Depth-first search can be used to compute a topological ordering of a directed acyclic graph.

Let $G = (V, E)$ be a directed graph. A topological ordering of G is an assignment $f(v)$ of every vertex $v \in V$ to a different number such that : for every $(v, w) \in E, f(v) < f(w)$.

Example (Quizz)

How many different topological orderings does the following graph have ?



Topological ordering

Properties

Every directed acyclic graph (DAG) has at least one topological ordering.

Every directed acyclic graph has at least one source vertex.

Problem

Input : A directed acyclic graph $G = (V, E)$.

Output : A topological ordering of the vertices of G.

Topological ordering

Pseudo-code 1/2

`TopoSort`

Input: directed acyclic graph $G = (V, E)$ in adjacency-list representation.
Postcondition: the f-values of vertices constitute a topological ordering of G.

---

mark all vertices as unexplored
$curLabel := |V|$ `// keeps track of ordering`
for every $v \in V$ **do**
  if v is unexplored **then** `// in a prior DFS`
    `DFS-Topo` (G, v)

Topological ordering

Pseudo-code 2/2

`DFS-Topo`

Input: graph $G = (V, E)$ in adjacency-list representation, and a vertex $s \in V$.
Postcondition: every vertex reachable from s is marked as "explored" and has an assigned f-value.

---

mark s as explored
for each edge (s, v) in s's outgoing adjacency list **do**
  if v is unexplored **then**
    `DFS-Topo` (G, v)
$f(s) := curLabel$ `// s's position in ordering`
$curLabel := curLabel - 1$ `// work right-to-left`

Topological ordering

Example