Basic Graph Algorithms

- Exploring Graphs
- Depth First Search
- DFS Forest - Start and Finish Time
- DAG, Topological Sorting
- Strongly Connected Components
- Breadth First Search
- Bipartite Graphs

IMDAD ULLAH KHAN

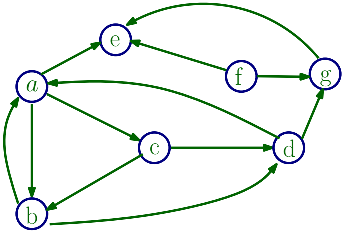
Reachability

Given a graph $G = (V, E)$

A vertex v is reachable form u, if there exists a path from u to v

- $R(u)$: the set of vertices reachable from u

$$R(u) \ : \ \{v : \exists \text{ a path from } u \text{ to } v\ \}$$



- g is reachable from a
- $g \in R(a)$
- $f \notin R(a)$
- $R(g) = \{e\}$
- $R(e) = \{\}$
- $R(c) = \{a, b, d, e, g\}$

Reachability

Given a graph $G = (V, E)$

A vertex v is reachable form u, if there exists a path from u to v

- $R(u)$: the set of vertices reachable from u

$$R(u) \; : \; \{ v : \exists \text{ a path from } u \text{ to } v \}$$

Fundamental graph exploration problems

- Given s and u, is $u \in R(s)$?
- Given s, find $R(s)$

A more constructive/algorithmic definition of reachability:

v is reachable form s, if v is a neighbor of s or v is reachable from a neighbor of s

$$R(s) \; = \; \{s\} \bigcup_{x \in N(s)} R(x)$$

Recursive Explore

Input: A graph $G = (V, E)$ and a node $s \in V$
Output: All nodes that are reachable from s, $R(s)$

- Encompasses the simpler question whether s is connected to v
- We give an algorithm for both directed and undirected graphs
- $R(s)$ is saved as a bit-vector $visited[1 \dots n]$ ▷ fixed order

$$visited[i] = 1 \iff v_i \in R(s)$$

- Populate $visited[\cdot]$ using $R(s) = \{s\} \bigcup_{x \in N(s)} R(x)$

Recursive Explore

Input: A graph $G = (V, E)$ and a node $s \in V$
Output: All nodes that are reachable from s, $R(s)$

- $R(s)$ is saved as a bit-vector $visited[1 \dots n]$ ▷ fixed order
- $visited[i] = 1 \iff v_i \in R(s)$
- Populate $visited[\cdot]$ using $R(s) \; = \; \{s\} \bigcup_{x \in N(s)} R(x)$

Algorithm Recursive algorithm to find $R(s)$

$visited[\cdot] \leftarrow$ ZEROS(n) ▷ initially no vertex is in $R(s)$
function EXPLORE(G,s)
  $visited[s] \leftarrow 1$
  for $x \in N(s)$ **do** ▷ a traversal of the adjacency list of s
    EXPLORE(G,x)

Recursion stopping condition?

Recursive Explore

Input: A graph $G = (V, E)$ and a node $s \in V$
Output: All nodes that are reachable from s, $R(s)$

- $R(s)$ is saved as a bit-vector $visited[1 \dots n]$ ▷ fixed order
- $visited[i] = 1 \iff v_i \in R(s)$
- Populate $visited[\cdot]$ using $R(s) = \{s\} \bigcup_{x \in N(s)} R(x)$

Algorithm Recursive algorithm to find $R(s)$

```
visited[·] ← ZEROS(n)                          ▷ initially no vertex is in R(s)
function EXPLORE(G,s)
    visited[s] ← 1
    for x ∈ N(s) do                            ▷ A traversal of the adjacency list of s
        if visited[x] = 0 then
            EXPLORE(G,x)
```

Iterative Explore

Input: A graph $G = (V, E)$ and a node $s \in V$
Output: All nodes that are reachable from s, $R(s)$

- Keep a *todolist* of nodes yet to be explored
- Initially no node is visited and *todolist* contains s
- Until the *todolist* is empty, remove a node from *todolist*, if not visited mark it visited and put all its neighbors in *todolist*

Algorithm EXPLORE(s) to find $R(s)$

```
visited ← ZEROS(n)
INSERT(todo, s)
while todo ≠ ∅ do
    u ← REMOVE(todo)
    visited[u] ← 1
    for x ∈ N(u) do                    ▷ A traversal of the adjacency list of u
        if visited[x] = 0 then
            INSERT(todo, x)
```

Correctness of EXPLORE(s)

If $u \in R(s)$, then $visited[u] = 1$

- Proof by induction on length of $s - u$ path
- If length of $s - u$ path is 0, then $u = s$ and $visited[s] = 1$
- If path is $s, v_1, \ldots, v_k, u$ ▷ length is $k + 1$
- By inductive hypothesis $visited[v_k] = 1$
- On calling EXPLORE(v_k), $visited[u]$ will become 1

If $u \notin R(s)$, then $visited[u] = 0$

- Since EXPLORE(v) is only called through some neighbor of v, $visited[u]$ will never become 1

Exploring the whole graph

- EXPLORE is still under-specified to analyze its runtime
- The EXPLORE procedure visits only the portion of the graph reachable from the given source s
- To examine the rest of the graph, we restart the procedure elsewhere, at some unvisited vertex
- What if we start EXPLORE from some already visited vertex?
- Notice that algorithm readily extends to directed graphs