LEC10 Depth-First Search

3.27 https://github.com/GUMI-21/MIT6.006_note

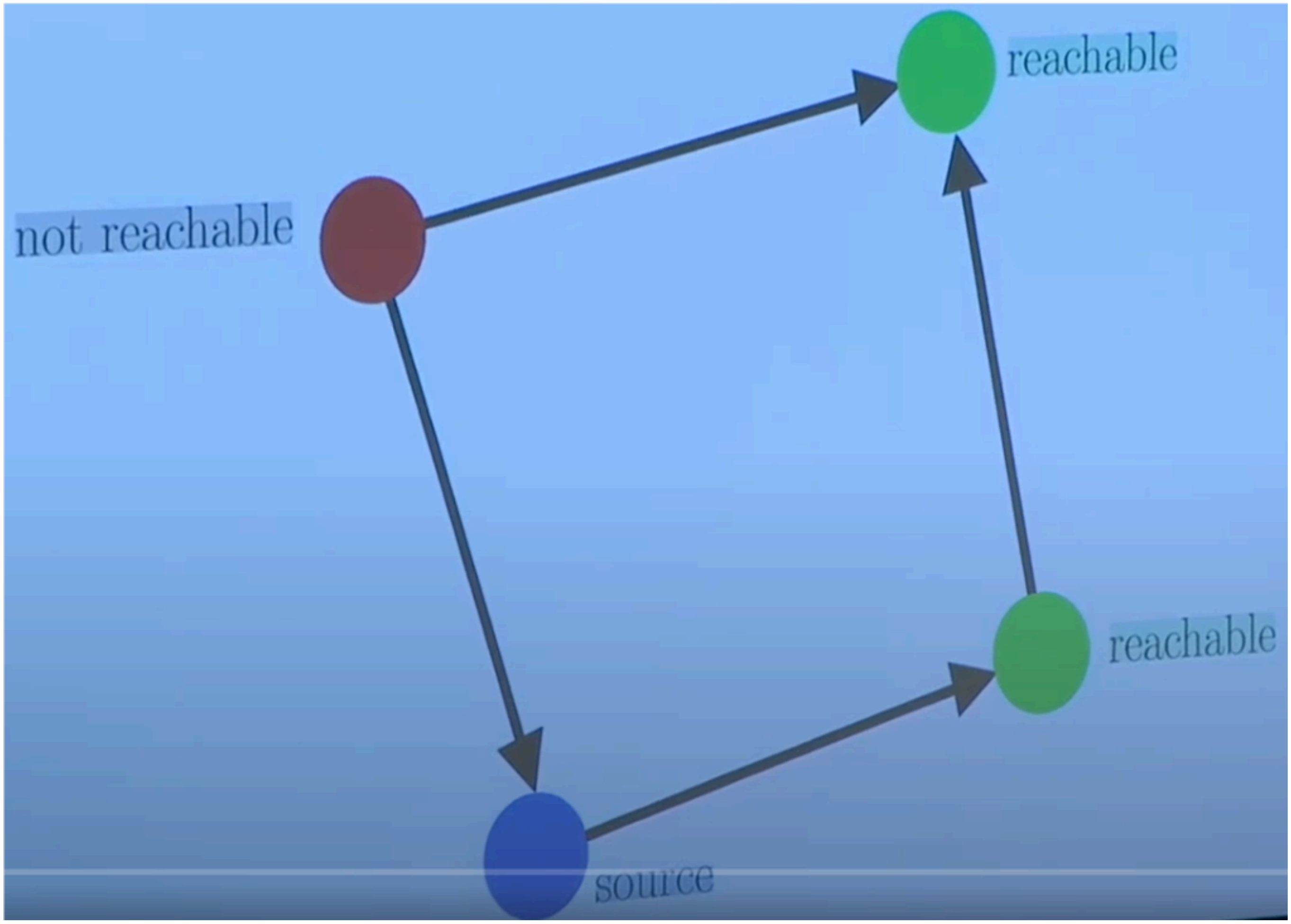
Previously

- Graph definitions
  directed/undirected, simple, neighbors, degree
- Graph representations
  Set mapping vertices to adjacency lists
- Paths
  simple paths(a path every vertix apear once), path length, distance, shortest path
- Graph Path Problems
  – Single Pair Reachability(G,s,t)
  – Single Source Reachability(G,s)
  – Single Pair Shortest Path(G,s,t)
  – Single Source Shortest Paths(G,s) (SSSP)
- BFS
  -algorithm that solves Single Source Shortest Paths
  -with appropriate data structures, runs in $O(|V| + |E|)$ time (linear in input size)

Depth-First Search (DFS)

New Problem

Signle Source Reachability.

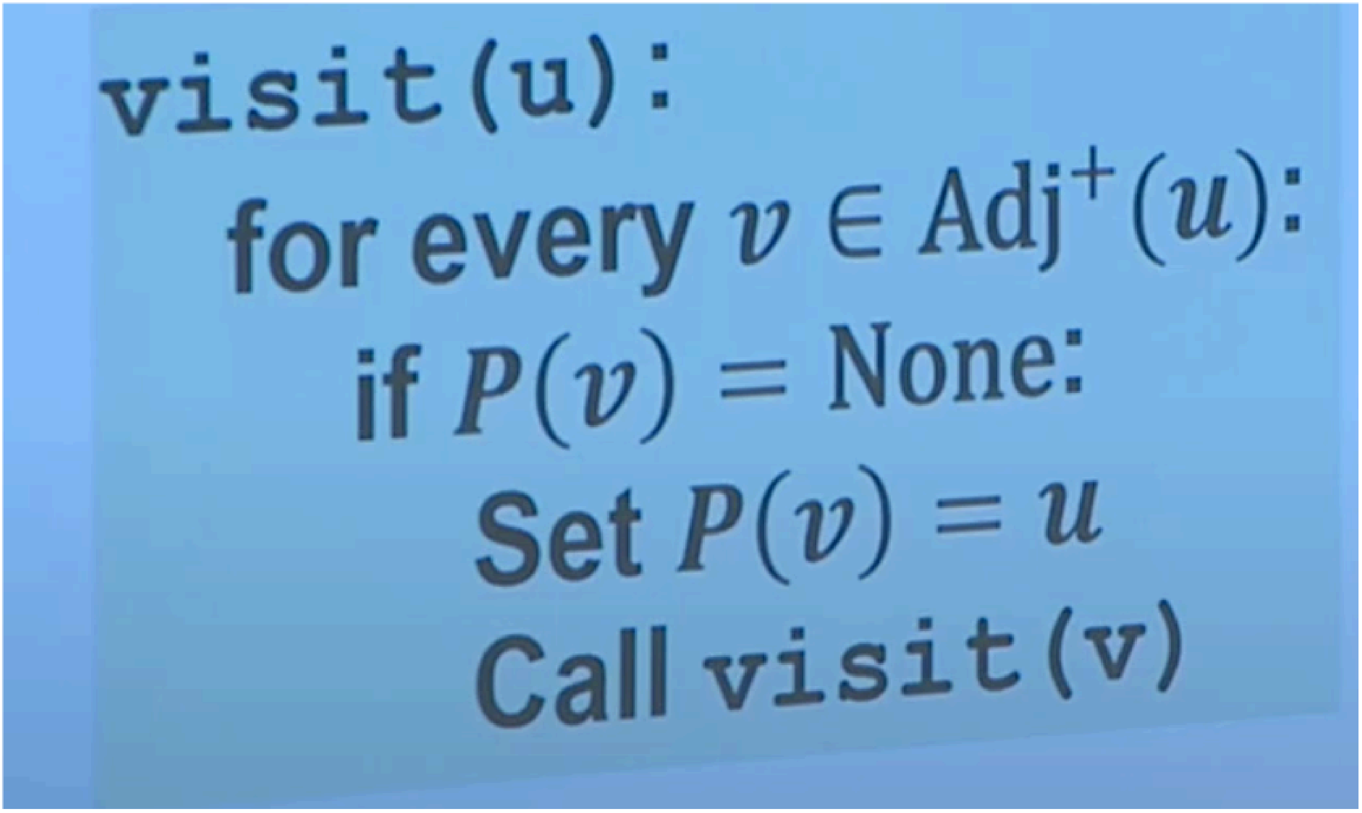


maintain a tree just like Parent[] in BFS(linear time), but I don't need my tree is the shortest path.

Alg

diff with BFS P: *Not level sets*

Set P(s) = None and then run visit(s)



When the recursion is unraveled-> the function will back trace, it's different form BFS, from source to depthest node then back call left nodes.

Proof

Clain: DFS visits all reachable v $\in$ V & correctly sets P(v).
Use Induction on k: distence to S(*Source*)
Base case: K=0 => S=0, ok
Induction step: Consider a vertex v with $distence(s, v) = k + 1$
Take u in V prev. on shortest path => $distence(s, u) = k$.
DFS consider v in Adj+(u),

1. P(v) != None 2.P(v) = None .done

Runtime

O(|E|) + O(|V|) *parent array*

A example of DFS path is not the shortest path

```
    A
   / \
  B   C
   \   |
```

```
D-E
```

if A->B->D->E then recure-> Parent[c] = A
DFS will lose edge of CE, But BFS loses edge DE.

Graph Connectivity

An undirected graph is connected if there is a path connecting every pair of vertices.
In a directed graph, vertex u may be reachable from v, but v may not be reachable from u.

Full-BFS and Full-DFS

Full-DFS to solve connectivity

-for v \in V: if v is unvisited: {DFS(v)}

- runtime
  O(V+E), linear time.

DAGS and Topological Ordering

Directed Acyclic Graph (DAG)

Directed graph that contains no directed cycle.
example: *A tree*.

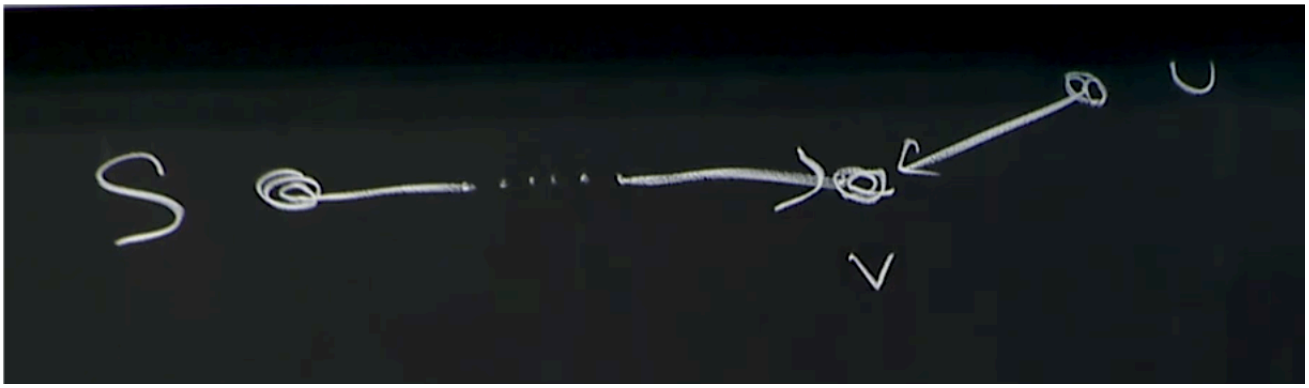
Topological order

A Topological Order of a graph G = (V, E) is an ordering f on the vertices such that: every edge (u, v) ∈ E satisfies f(u) < f(v).

- f: **the time of DFS finished processing the node. After recursion back to the node.**
  Means u has to appear before v.
  not unique
  -> *If there is a directed edge (u→v)(u \to v)(u→v), then vertex u must appear before vertex v in the ordering.*

Finishing order

Order in which a Full-DFS finishes visiting each vertex.
G is DAG => reverse of finishing order is a Topologocal order

- Proof
  (u,v) \in E, want: u is ordered before v.
  Two cases:
  1.u visited before v. means $visit(v)\ must\ be\ called\ before\ visit(u)$
  2.v visited before u.
  

  DAG means no path from v to u. => u cannot be reached from v.
  visit(v) completes without seeing u.

Cycle Detection

Full-DFS will find a topological order if a graph G = (V, E) is acyclic.

- Given a directed graph, does exist a cycle in DG?
  if only if
- *Alg*
  if G has a Cycle
  then => *Full DFS will traverse an edge v to some ancestor of v.*
  Proof: Take cycle (V0,V1,...,Vk,V0)
  let v0 is first visited by DFS, => visit Vk => see (Vk,Vc) \in E

Application of Topological order

Task Scheduling.
Circuit Dependency Analysis.
Expression Evaluation.