Algorithms

DFS Algorithm

: start

dfs (graph, source)

curVertex <- graph[source]

marked[source] = true

for adjVertex in curVertex.adj()

if not marked[source]

dfs(graph, adjVertex)

: end

Runtime

DFS marks all the vertices connected to a given proportional to the sum of their degrees.

BFS Algorithm

:start

bfs(graph, source)

queue // stores vertices

marked[source] =true;

queue.add(source)

while (queue.isEmpty())

v <- queue.pop()

for adjVertex in v

if !marked[adjVertex]

queue.add(adjVertex)

:end

Runtime

BFS takes time proportional to VE in the worst case.

Theorem

For any vertex v reachable from s, BFS computes a shortest path from s to v (no path from s to v has fewer edges).

Algorithm connected components

:start

connectedComponents(graph)

count = 0 // stores the number of connected components

source = 0

for i =0, i<graph.V(), i++

if !marked[i]

dfs(graph, i)

count++

:end

Runtime:

Proposition C. DFS uses preprocessing time and space proportional to VE to support constant-time connectivity queries in a graph.

Cycle Detection Algorithm

: start

global hasCycle = false

dfs (graph, source)

curVertex <- graph[source]

marked[source] = true

for adjVertex in curVertex.adj()

if not marked[source]

dfs(graph, adjVertex)

else if (adjVertext != source) hasCycle = true

: end

Two Color Algorithm

: start

global color[]

global isTwoColorable = true

dfs (graph, source)

curVertex <- graph[source]

marked[source] = true

for adjVertex in curVertex.adj()

if not marked[source]

color[adjVertex] = !color[source]

dfs(graph, adjVertex)

else if (color[adjVertex] = color[source]) isTwoColorable = false

: end