Outline and Reading

- Definitions (§6.1)
  - Subgraph
  - Connectivity
  - Spanning trees and forests
- Depth-first search (§6.3.1)
  - Algorithm
  - Example
  - Properties
  - Analysis
- Applications of DFS (§6.5)
  - Path finding
  - Cycle finding

Subgraphs

- A subgraph $S$ of a graph $G$ is a graph such that
  - The edges of $S$ are a subset of the edges of $G$
  - The vertices of $S$ are a subset of the vertices of $G$
- A spanning subgraph of $G$ is a subgraph that contains all the vertices of $G$

Connectivity

- A graph is connected if there is a path between every pair of vertices
- A connected component of a graph $G$ is a maximal connected subgraph of $G$

Trees and Forests

- A (free) tree is an undirected graph $T$ such that
  - $T$ is connected
  - $T$ has no cycles
This definition of tree is different from the one of a rooted tree
- A forest is an undirected graph without cycles
- The connected components of a forest are trees

Spanning Trees and Forests

- A spanning tree of a connected graph is a spanning subgraph that is a tree
- A spanning tree is not unique unless the graph is a tree
- Spanning trees have applications to the design of communication networks
- A spanning forest of a graph is a spanning subgraph that is a forest
Depth-First Search

- Depth-first search (DFS) is a general technique for traversing a graph.
- A DFS traversal of a graph G visits all the vertices and edges of G.
- Determines whether G is connected.
- Computes the connected components of G.
- Computes a spanning forest of G.

DFS on a graph with \( n \) vertices and \( m \) edges takes \( O(n + m) \) time.

DFS can be further extended to solve other graph problems:

- Find and report a path between two given vertices.
- Find a cycle in the graph.

Depth-first search is to graphs what Euler tour is to binary trees.

DFS Algorithm

The algorithm uses a mechanism for setting and getting "labels" of vertices and edges.

Algorithm DFS(G, v)

Input: graph G and a start vertex v of G
Output: labeling of the edges of G as discovery edges and back edges

1. setLabel(v, VISITED)
2. for all e in G.incidentEdges(v)
   a. if getLabel(e) = UNEXPLORED
      i. w ← opposite(v, e)
      ii. if getLabel(w) = UNEXPLORED
         A. setLabel(e, DISCOVERY)
         B. DFS(G, w)
      iii. else
         A. setLabel(e, BACK)

Example

- unexplored vertex
- visited vertex
- unexplored edge
- discovery edge
- back edge

Example (cont.)

DFS and Maze Traversal

- The DFS algorithm is similar to a classic strategy for exploring a maze:
  a. We mark each intersection, corner and dead end (vertex) visited.
  b. We mark each corridor (edge) traversed.
  c. We keep track of the path back to the entrance (start vertex) by means of a rope (recursion stack).

Properties of DFS

Property 1

\( DFS(G, v) \) visits all the vertices and edges in the connected component of v.

Property 2

The discovery edges labeled by \( DFS(G, v) \) form a spanning tree of the connected component of v.
Analysis of DFS

- Setting/getting a vertex/edge label takes $O(1)$ time
- Each vertex is labeled twice
  - once as UNEXPLORED
  - once as VISITED
- Each edge is labeled twice
  - once as UNEXPLORED
  - once as DISCOVERY or BACK
- Method incidentEdges is called once for each vertex
- DFS runs in $O(n + m)$ time provided the graph is represented by the adjacency list structure
  - Recall that $\sum\deg(v) = 2m$

Path Finding

- We can specialize the DFS algorithm to find a path between two given vertices $u$ and $z$ using the template method pattern
- We call $\text{DFS}(G, u)$ with $u$ as the start vertex
- We use a stack $S$ to keep track of the path between the start vertex and the current vertex
- As soon as destination vertex $z$ is encountered, we return the path as the contents of the stack

Algorithm $\text{pathDFS}(G, v, z)$

1. $\text{setLabel}(v, \text{VISITED})$
2. $S.push(v)$
3. if $v = z$ return $S.elements()$
4. for all $e \in G.\text{incidentEdges}(v)$
5.   if $\text{getLabel}(e) = \text{UNEXPLORED}$
6.     $w \leftarrow \text{opposite}(v, e)$
7.     $S.push(e)$
8.     if $\text{getLabel}(w) = \text{UNEXPLORED}$
9.       $\text{setLabel}(e, \text{DISCOVERY})$
10.      $\text{pathDFS}(G, w, z)$
11.     $S.pop()$
12.   else
13.     $T \leftarrow \text{new empty stack}$
14.     repeat
15.       $o \leftarrow S.pop()$
16.       $T.push(o)$
17.     until $o = w$
18.     return $T.elements()$
19. $S.pop()$

Cycle Finding

- We can specialize the DFS algorithm to find a simple cycle using the template method pattern
- We use a stack $S$ to keep track of the path between the start vertex and the current vertex
- As soon as a back edge $(v, w)$ is encountered, we return the cycle as the portion of the stack from the top to vertex $w$