R10 Depth-First Search

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

- BFS: discovers vertices reachable from a queried vertex s level-by-levl oautward from s.
- DFS: discovers vertices reachable from a queried vertex s by searching undiscovered vertices as deep as possible before exploring other branched.
  DFS tree will note represent shortest paths in an unweighted graph.
  For index-labeled adjacency lists DFS python code:

```
def dfs(Adj, s, parent = None, order = None): # s: start vertex
    if Parent is None:
        parent = [None for v in Adj]       # O(V)
        parent[s] = s
        order = []
    for v in adj[s]:                       # O(Adj[s]) loop over neighbors
        if parent[v] is None:
            parent[v] = s
            dfs(Adj, v, parent, order)     # Recursive Call
    order.append(s)                        # the finishing order
    return parent, order
```

- *runtime Analysis*
  dfs called on each vertex at most once.
  the amount of work done by each recursive search from vertex v is proportional to the *out-degree deg(v) of v.* => O(|E|)
  and the *parent array* returned has length|V|, depth-first search runs in O(|V| +|E|) time.

Full Graph Exploration

Full-BFS or Full-DFS
explore each connected component in the graph by performing a search from each vertex in the graph that has not yet been dis covered by the search.
a Full-DFS code

```
def full_dfs(Adj):
    parent = [None for v in Adj] # O(V)
    order = []
    for v in range(len(Adj)): # O(V) loop over vertices.
        if parent[v] is None:
            parent[v] = v # the search start of dfs this time.
```

```
            dfs(Adj, v, parent, order) # DFS from v (BFS can also be used)
    return parent, order
```

dfs is often used to refer to both a method to search a graph from a specific vertex, and as amethod to search an entire.

DFS Edge Classification

- tree edge: parent[v] = u
- back edge: u is v's descendant but edge is not part of the DFS tree
- forward edge: v is u's descendant but edge is not part of the DFS tree
- cross edge: neither
  consider a graph edge from vertex u to v, We call the edge a *tree edge* if the edge is part of the DFS tree (i.e. parent[v] = u). Otherwise, the edge from u to v is not a tree edge and is either a *back edge, forward edge, or cross edge* depending respectively on whether: u is a descendant of v, v is a descendant of u, or neither are descendants of each other, in the DFS tree.

How identify back edges computationally?

While performing a depth-first search, keep track of the set of ancestors of each vertex in the DFS tree during the search (in a direct access array or a hash table). When processing neighbor v of s in dfs(Adj, s), if v is an ancestor of s, then (s, v) is a back edge, and certifies a cycle in the graph.

Topological Sort

If the graph is acyclic, the order returned by dfs (or graph search) is the reverse of a topological sort order.

Excise

- giving a diploma to student a before student b whenever a oversees b in any student organization
- => check if there is a cycle in graph of students.
  While performing the DFS, keep track of the ancestors of each vertex in the DFS tree, and evaluate if each new edge processed is a back edge.
  If a back edge is found from vertex u to v, follow parent pointers back to v from u to obtain a directed cycle in the graph to prove to the principal that no such order exists.

Otherwise, if no cycle is found, the graph is acyclic and the order returned by DFS is the reverse of a topological sort, which may then be returned to the principal.