EECS3101 notes:: DFS examples

Jerry Wu

2023-03-21

BFS's evil twin

Pushing children (child abuse)

For BFS in a tree, we try to traverse all the nodes in a layer before moving to the next layer. However for DFS, we're interested in moving down all the layers first, then moving to the next set of nodes beside it. The pseudocode for DFS in a tree is exactly the same as BFS, however, we're using a **stack** this time rather than a queue (push/pop instead of enqueue/dequeue).

```
NOT_QUITE_DFS(root):
    S = Stack()
    Push(Q, root)
    while Q not empty:
        x = Pop(Q)
        print(x)
        for each child c of x:
            Push(Q,c)
```

Just like in BFS, we want to void visiting vertices in the graph that have already been visited. We can remember which vertex introduces us to a new vertex π_v. However, we have to remember these values as well::

- Clock variable:: Incremented whenever the colour of a vertex is changed
- $\forall v \in V$, remember two timestamps::
  - d_v is the discovery time when the vertex is first encountered (changes to grey)
  - f_v is the finishing time when all vertices neighbours have been visited (changes to black)

Full pseudocode is available on slide 23.

Runtime of DFS

The runtime of DFS is exactly identical to BFS ($O(|V| + |E|)$)

Applications

DFS allows us to detect cycles within a graph. We want to determine the descendant/ancestor relationship in the DFS forest.

"Overlapping parethensis are ILLEGAL"-Larry YL Zhang 2023

- First, we can trace back the π_v pointers (red edges) starting from y to see if we can arrive at u. The worst case for this is if the tree is a linked list ($O(|V|)$). We can forget about this!

A better way is that we can say for sure that $d_v < f_v \forall v \in V$, and we can visualize starting and ending time as an interval $[d_v, f_v]$. For parenthesis, the starting and ending times are either all contained within each other's intervals, or are completely disjoint. This is how we know that bracket matching is valid. So from this propisition, we can say that a node is a descendant of an ancestor if and only if **the descendant's waiting time interval is contained within its ancestor's waiting time interval**.

Classifying edges

We have 4 types of edges in DFS::

- Tree edge:: An edge in the DFS forest
- Back edge:: A non tree edge pointing from a vertex to its ancestor in the DFS forest
- Forward edge:: A non tree edge pointing from a vertex to its descendant in the DFS forest
- Cross edge:: Every other edge

We can efficientyy check edge types because we can look at the waiting time intervals. For example, if $u = [1, 8] \supset v = [2, 7]$, then $u \to v$ is a tree edge i.e. $\pi_v = u$. If not a tree edge, then we need to determine whether it is a forward or back edge. We do this by comparing the waiting time intervals once again. If $[u_1, v_1] \supset [u_2, v_2]$, then $u_1 \to v_1$ is a forward edge, else back edge. So from here, we can say that **a graph is cyclic if and only if DFS yields a back edge!**

Proof

- The if case:: let an edge be (u, v). By definition of back edge, v is an ancestor of u in the DFS tree. Then $\exists$ a path from v to u along with the back edge. BOOM! Cycle.
- The only if case:: Invoke the white path theorem (full proof on slide 45).

DFS properties in undirected graphs

- In an undirected graph, $\forall \epsilon \in E$, ϵ is always either a tree edge or back edge. It is impossible to have forward edges or a cross edge.

Why do we care about cycles?

"Don't tell people about prerequisite waivers"-Larry YL Zhang 2023

Say that the chair of the EECS department wants to introduce a cycle in the prerequesite tree for courses in the department. Then all the courses in the cycle can't be taken by anyone because there's no place to start! This is known as "circular dependency" or a deadlock in operating systems. Another realistic example is requiring experience to get a job, but needing a job to get experience (too relatable!).

More applications

- Topological sort
  - Do DFS
  - Order vertices accordinv to their finishing times f_v
- Strongly connected components
  - Subgraphs where any pair of vertices can reach each other in a graph (eg. mutual friends)