

Depth-first search - DFS traversal

- Keep the following applications in mind:

1) Are all paths unique in a given G ?

2) Is G acyclic?

[Tarjan '72] 3) Find the strongly connected components in G ? I.e. a maximal $S \subseteq V$ s.t.

$\forall i, j \in S, i \rightarrow j \ \& \ j \rightarrow i$.

[Edmonds, Karp '72] 4) Maximum flow?

5) Matching?

(Hopcroft & Tarjan '73)

- DFS idea: Recursively explore a neighbor before moving to the next neighbor.

- Pseudocode:

DFS-graph ($G = (V, E)$) {

 for each $v \in V$ $visit[v] \leftarrow false$;

$count \leftarrow 1$;

 for each $v \in V$

 if ($visit[v] = false$) $DFS(v)$;

}

DFS(v) {

visit[v] ← true;

begin time → D[v] ← count ++;

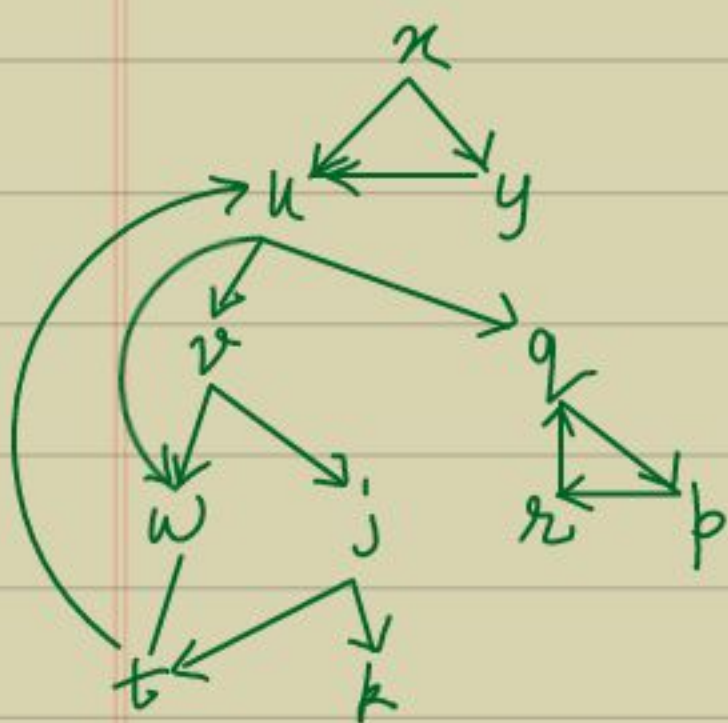
For each (v, w) ∈ E with visit[w] = false
do DFS(w);

end time → F[v] ← count ++;

}

▷ O(n+m) time complexity is easy to show.

- Example:



D[u] = 1

D[v] = 2

D[w] = 3

D[t] = 4

F[t] = 5

F[w] = 6

D[j] = 7

D[k] = 8

F[k] = 9

F[j] = 10

F[v] = 11

D[q] = 12

⋮

▷ Note the general relationship:

$D[u] < D[v]$

$< F[v] < F[u]$

(Interval enclosed)
for tree edge (u, v)

a forward edge or

- Lemma: (u, v) is an edge in the DFS-tree
iff $D(u) < D(v) < F(v) < F(u)$ & $(u, v) \in E$.

Pf: • (\Rightarrow) : is clear from the recursive nature of the DFS traversal.

• (\Leftarrow) : In the DFS algorithm either v is explored as a neighbor of u ; or, it is explored as a neighbor of w ; in the latter case (u, v) is called a forward edge.



□

Lemma: The other kinds of non-tree edges are:

(Parenthesis theorem)

• cross edge (u, v) : $(D[u], F[u])$ & $(D[v], F[v])$ are disjoint.

• backward edge (u, v) : $D[v] < D[u] < F[u] < F[v]$.

Egs.



Application 1: Unique-path testing

- Input: Directed graph $G = (V, E)$.

Output: Yes iff $\forall u, v \in V$, \exists at most one simple path from u to v .

- Easy algorithm:

For each $u \in V$ {

DFS(u);

if (DFS-tree has a forward
or a cross edge)

then output non-unique-path;

}

output unique-path;

▷ Backward edges in DFS(u) do not yield multiple simple paths from u to any v .

▷ Time complexity is $n \times O(mn)$.

Bringing it down to $O(n^2)$.

- Note that if $\exists u, v \in V$ s.t. in $\text{DFS}(u)$:
there are two backward edges from v
then G is non-unique-path!

Thus,

$\triangleright G$ is unique-path $\Rightarrow \forall u \in V, \text{DFS}(u)$
has no cross edge, no forward edge &
 $\forall v \in V$, at most one backward edge from v .

- This can be used to speed up the code:

$\text{DFS}(u)$ {

visit[u] \leftarrow true;

D[u] \leftarrow count++; // D[·], F[·] are initialized to -1.

BackEdge[u] \leftarrow 0;

For each $(u, v) \in E$ {

if (visit[v] = false) DFS(v);

.....(contd.)

\therefore forward/cross edge

```

else if (F[v] > 0) output non-unique-path;
else {
    BackEdge[v]++;
    if (BackEdge[v] ≥ 2)  $\therefore$  Two backward edges
        output non-unique-path;
    } //end else
} //end for
F[u] ← count++;
} //end DFS(u)

```

Lemma: The algo. for DFS(u) above, takes $O(n)$ time.

Pf:

- The edges that are traversed are either DFS(u) tree edges, or ≤ 1 forward edge, or ≤ 1 cross edge, or ≤ 2 backward edges.

- This gives us a count of $n-1+1+n = O(n)$ edges actually traversed. \square

Theorem: Doing this for all u , we get an $O(n^2)$ time algo. to decide unique-path.

Application 2: Topological ordering in dag

- Can a backward edge appear in DFS-Graph (G) , if G is a dag?

Theorem: Testing whether G is a dag is doable in $O(n+m)$ time.

- Let $G = (V, E)$ be a dag & $u \in V$.

Let v, w be vertices in $\text{DFS}(u)$.

- If (v, w) is a DFS-tree edge then $F[v] > F[w]$.
- If (v, w) is a forward edge then $F[v] > F[w]$.
- If (v, w) is a cross edge then $F[v] > F[w]$.

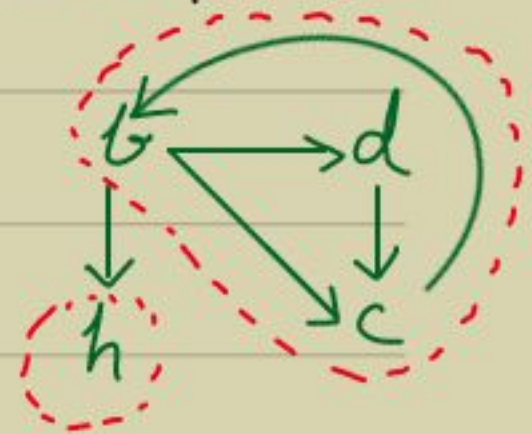
Theorem: Decreasing order of finish time $F[\cdot]$ gives us a topological ordering on a dag G in $O(n+m)$ time.

Application 3: Compute strongly connected comp.

- Defn: Let $G=(V,E)$ be a directed graph. Vertices $u, v \in V$ are strongly connected if $u \rightarrow v$ & $v \rightarrow u$.

A maximal subset of vertices in V , with every pair strongly connected, is called a strongly connected component.
(SCC)

▷ Using repeated DFS we can find the strongly connected components in $n \times O(m+n)$ time.



Pf: (Exercise) \square

- Qn: Can this be improved?

- Yes, in $O(m+n)$ time.

We'll discuss Kosaraju (1978)'s idea.

- Defn: The root of a scc^S is the vertex which is visited first in DFS.

$\Rightarrow \triangleright$ Root of a scc^S is the vertex that finishes last (among S).

- This gives an idea to spot the root of some scc :

Do DFS on G , order V in the increasing order of $F[\cdot]$ & pick the last vertex u .

$\Rightarrow u$ is the root of a scc in G , say S .

- Qn: How do we find S ?

Let S' be another scc & $v' \in S'$.

Lemma: $\forall u' \in S, v' \in S', (v', u') \notin E$.

Pf:

• Let v be the root of S' (Contd.)

• Let (v', u) be an edge.

• Case 1: [DFS visits v' first] Clearly, u will be a descendant of v' in the DFS tree.

$\Rightarrow u$ will be a descendant of $v \Rightarrow F[u] < F[v]$
which is a contradiction!

• Case 2: [DFS visits u' first]

Since v' is unreachable from S , first the exploration of u' will finish & then the DFS of S' can begin.

$\Rightarrow F[u] < F[v]$.

\Rightarrow We get a contradiction in all cases. \square

..... (contd.)

..... (contd.)

- This gives us the idea to reverse the edges in G & consider $G^r := (V, E^r)$:

▷ S^r is a SCC in G^r & there is no edge going out of S^r .

▷ DFS on u in G^r gives us S^r !

Final algorithm for SCC

- Do DFS on G & store V in the decreasing order of $F[\cdot]$.
- Compute G^r by reversing the edges.
- Initialize $\forall v \in V, \begin{cases} \text{SCC-found}[v] \leftarrow \text{false}; \\ \text{SCC-num}[v] \leftarrow -1; \end{cases}$
- $i \leftarrow 1;$
- For each $v \in V$ {
 if ($\text{SCC-found}[v] = \text{false}$) {
 Do DFS(v) in G^r ;
 Let the DFS-tree have vertices A ;
 For each $x \in A$ {
 $\text{SCC-found}[x] \leftarrow \text{true};$
 $\text{SCC-num}[x] \leftarrow i;$
 remove x from G^r ;
 }
 $i++;$
 }
}

Theorem: SCC is computable in $O(m+n)$ time.

Pf: (Exercise)