Exploring graphs

BFS - level by level exploration

compute distance (no. of edges)

parent information - paths

BFS tree - cross edges reveal cycles

Another strategy

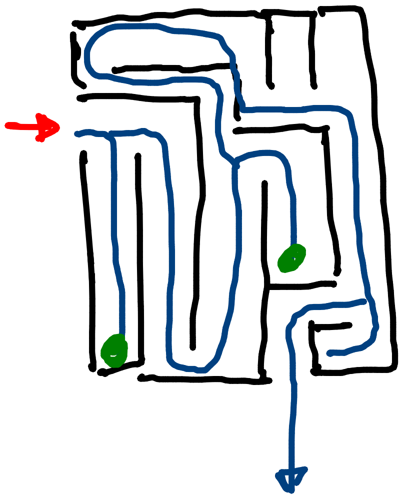
Depth First Search (DFS)

Searching a maze

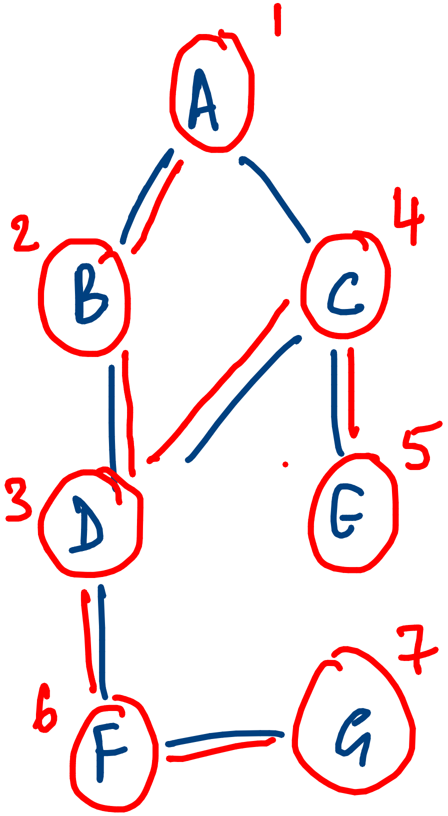
DFS:

Search neighbours depth first

Backtrack if you get stuck



```
dfsexplore(v)
    Mark[v] = 1
    for each edge (v,w)
        if  Mark[w]==0
            dfsexplore(w)
dfsexplore(v)
  Mark[v] = 1
  for each edge (v,w)
    if Mark[w] == 0
      dfsexplore(w)
```



d(A) → d(B) → d(D) → d(C) → d(E)

d(F) → d(G)

```
dfs(G)
    for each v, Mark[v]=0
    component = 0
    for each v
        if Mark[v]==0
            component = component+1
            dfsexplore(v, component)
```

```
dfsexplore(v,c)
    Mark[v] = c
    for each edge (v,w)
        if Mark[w]==0
            dfsexplore(w,c)
```

Like BFS

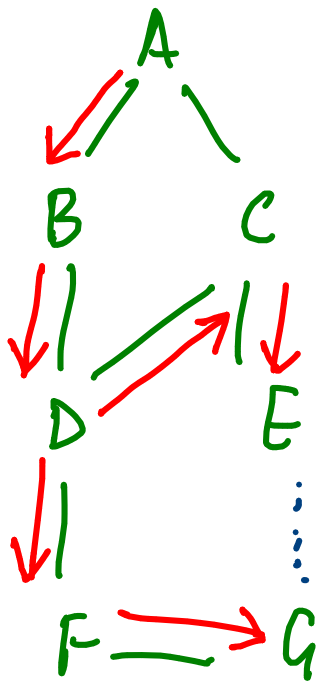
parent info

DFS Tree

Non tree edge C-A

"Back up the tree"

Back Edge



← ? E would have explored G

Can there be cross edges?

Non-tree edge, ends not related by ancestor

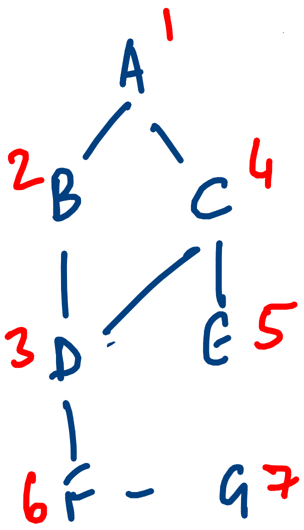
e.g. E,G Not possible

Check that, for undirected graphs:

BFS: Only cross edges, no back edges

DFS: Only back edges, no cross edges

The order of DFS visits can reveal a lot



```
dfsexplore(v)
    previsit(v)
    Mark[v]=1
    for each (v,w) ∈ E
        if Mark[w]==0
            dfsexplore(w)
```

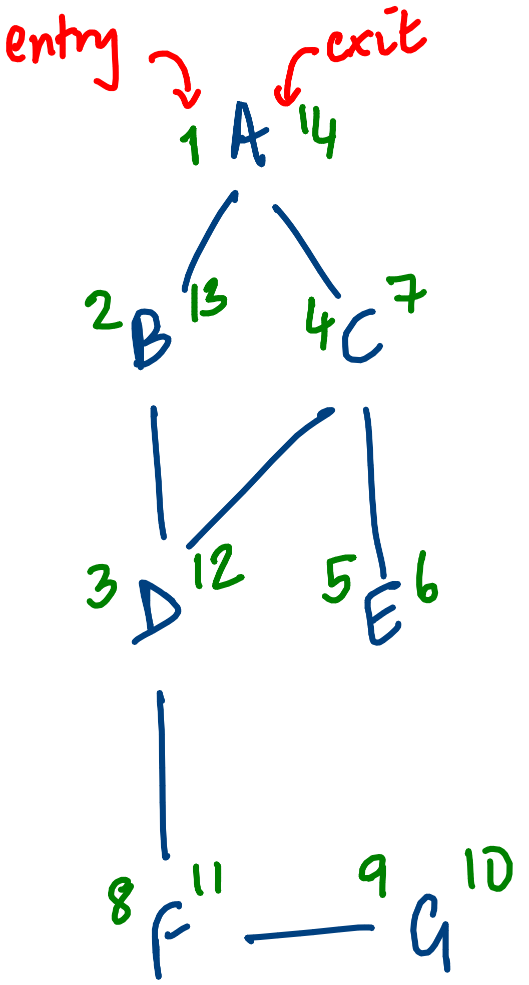
previsit(v) → { count = count+1
visitorder[v] = count }

count = 0 initially

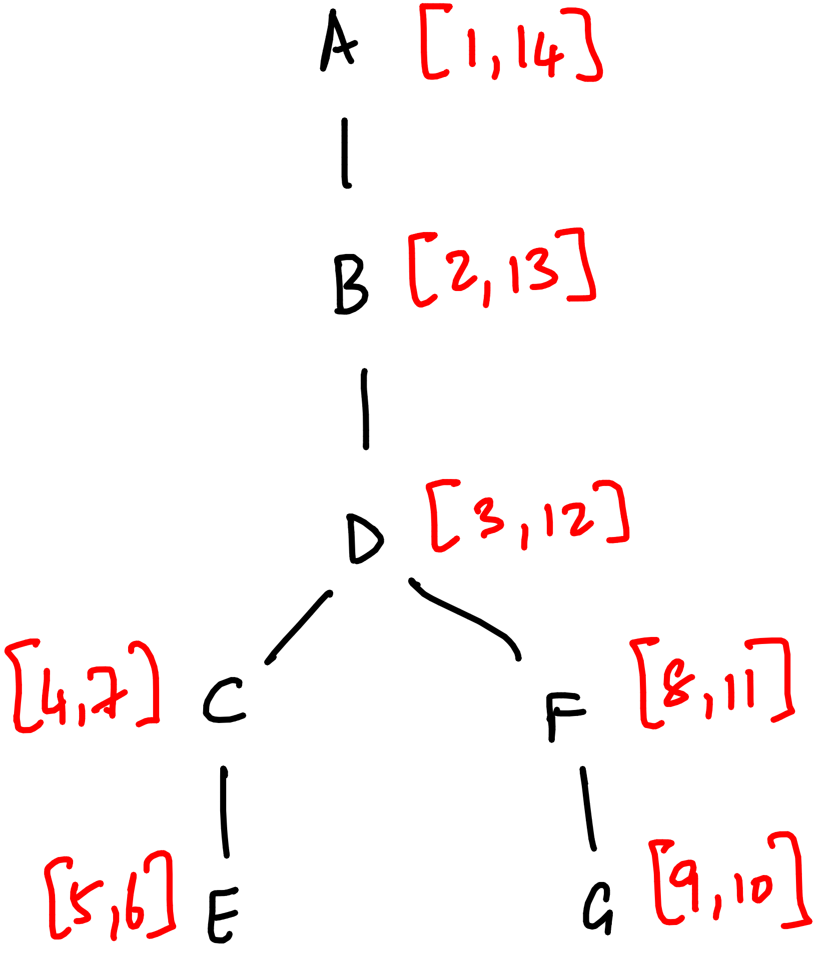
Also record the "time" we finish each vertex

component no.

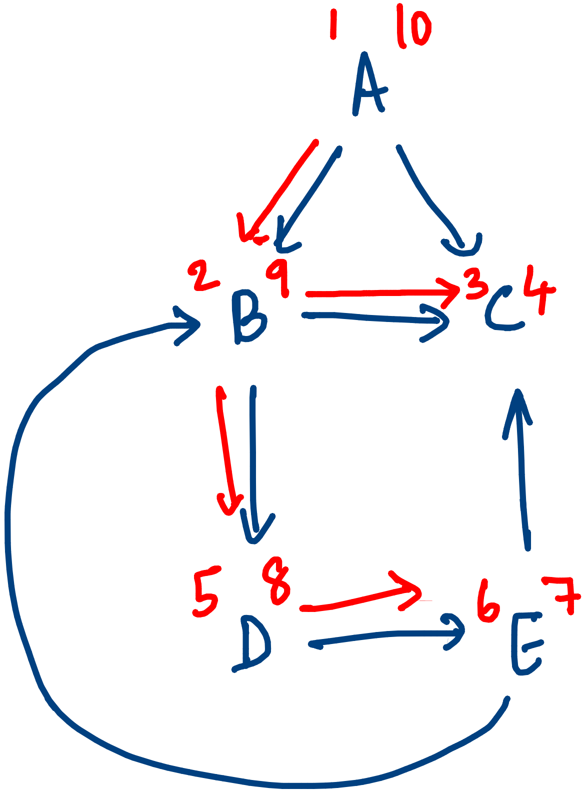
```
dfsexplore(v, c)
    previsit(v)          →    count = count + 1
                              entry[v] = count
    Mark[v] = c
    for each (v,w) ∈ E
        if Mark[w] == 0
            dfsexplore(w,c)
    postvisit(v)         →    count = cont + 1
                              exit [v] = count
```

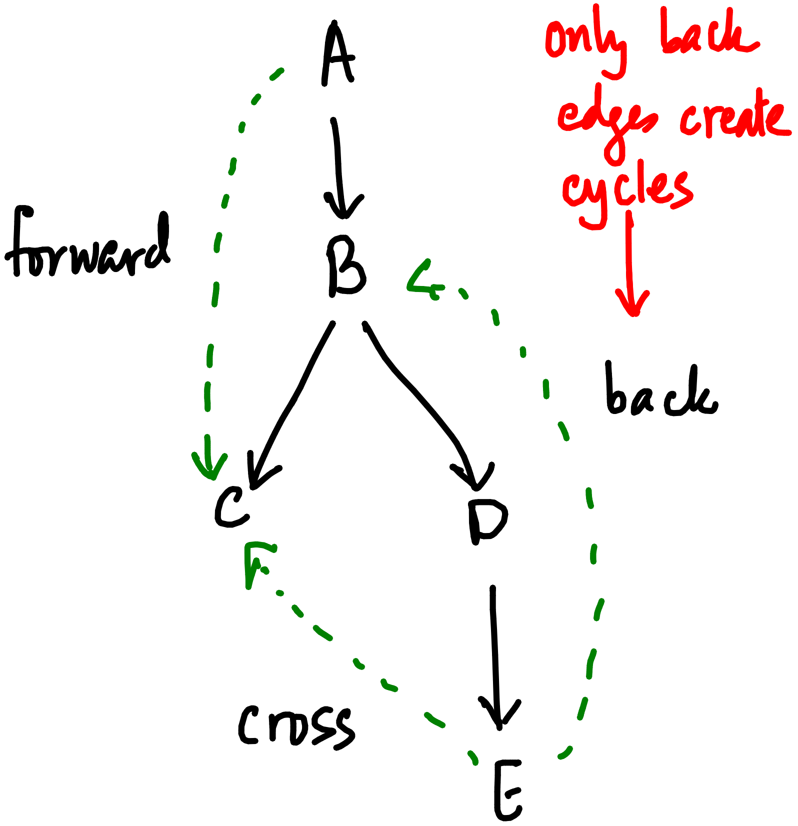


Ancestor : Intervals nested

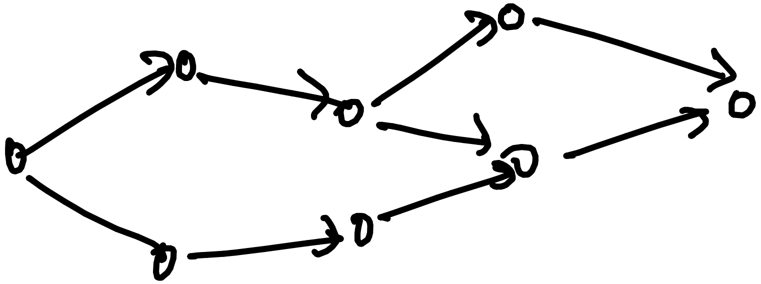


Directed Graphs





Directed Acyclic Graphs



Extremely useful model for dependencies

e.g. Courses & prerequisities

Typical questions

- Find a "legal" sequence to complete all courses
- If tasks can be done in parallel, min time to complete

Question !

"legal" order of vertices

When we enumerate v, all w s.t $w \to v$ are already enumerated

$\Rightarrow$ implies any u s.t. $u \to \dots \to v$ has also been enumerated

Enumeration = total order

When is an enumeration legal ?

-- - . i --- j - --

For any $(i,j) \in E$, i appears befre j

respects E = partial order

Start the enumeration?

First vertex cannot have incoming edges

indegree = 0

Does such a vertex always exist?

If not, walk back

After n steps - cycle!

Eliminate this vertex (& outgoing edges)

- still a DAG directed acyclic

"TOPOLOGICAL SORT OF A DAG"