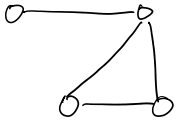
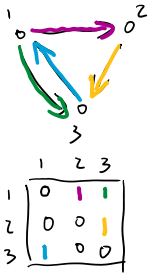
A Graph is a collection of vertices and edges
nodes arcs



$G = (V, E)$

On computer we typically use

- "adjacency lists"
  every vertex has a list of neighbors
  adjacency[v] array that contains lists
  - uses $O(|E|)$ space

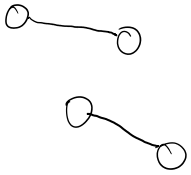


- "adjacency matrix"
  table that for every pair of vertices says if there is an edge connecting them or not
  - uses $O(|V|^2)$ space

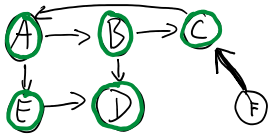
"simple graph" graph without duplicate edges

$$\frac{|V|}{2} \leq |E| \leq |V|^2$$

↑ otherwise we have vertices that do not connect to anything "isolated vertices"



DFS Depth First Search




```
DFS(v)
  visited[v] = true      v.visited = true
  for (v,u) ∈ E  // for every out-neighbor u of v
    if not visited[u]
      DFS(u)
```

```
for v ∈ V
  if not visited[v]
    DFS(v)
```

```
DFS(A)
└> DFS(B)
   ├> DFS(C)
   └> DFS(D)
└> DFS(E)
```

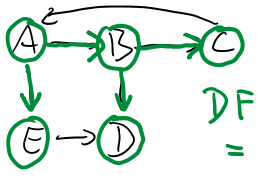
after calling DFS(v) then visited contains all vertices that can be reached from v.

time complexity: $O(|E|)$ because every vertex and edge is visited at most once.

$O(|V|+|E|)$

DFS-tree ← set of edges used to visit each vertex for the first time

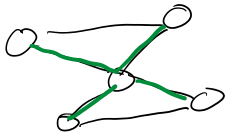
```
T = {}
DFS(v)
  visited[v] = true
  for (v,u) ∈ E
    if not visited[u]
      T = T ∪ {(v,u)}
```



DFS-tree = green edges

```
for (v,u) ∈ E
    if not visited[u]
        T = T ∪ {(v,u)}
        DFS(u)
```

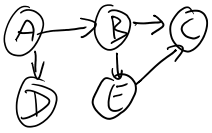
Def: For undirected graph
a spanning tree is
- a tree (no cycles)
- connects entire graph



Q: Find cycles
Q: Topological order

```
post counter = 0
pre counter = 0
DFS(v)
    visited[v] = true
    pre counter += 1
    pre order[v] = pre counter
    for (v,u) ∈ E
        if not visited[u]
            DFS(u)
    post counter += 1
    post order[v] = post counter
```

Def: Topological Order of graph G
Ordering of the vertices
such that all edges go left to right

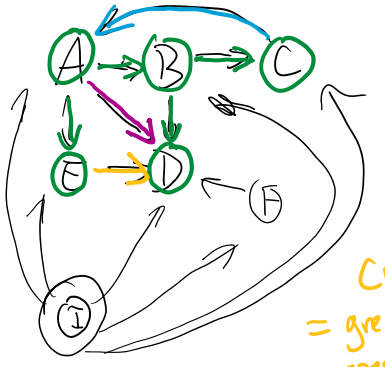


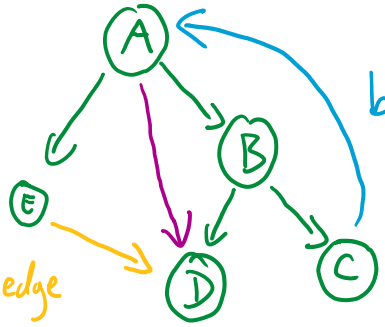
A → D B → E → C

∃ cycle ⇒ ∄ order
∄ cycle ⇒ ∃ order

Preorder = order in which DFS visits vertices
Postorder = order in which DFS is "done" with vertices

Post order = reverse of topological order





back edge
= green path goes up
= preorder(A) < preorder(C)
and postorder(C) < postorder(A)

Crossing edge
= green path goes up and down
= postorder(E) > postorder(D)
preorder(E) > preorder(D)

forward edge
= green path goes down
= postorder(D) < postorder(A)
preorder(D) > preorder(A)

preorder(E) > preorder(D) = postorder(D) < postorder(A)

preorder(D) > preorder(A)

(u, v) { forward : pre u < pre v post u > post v
back : pre u > pre v post u < post v
crossing : pre u > pre v post u > post v

DFS(v)

queue.add(v)

while queue not empty

v = queue.pop()

if visited[v]

continue

visited[v] = true

for (v, u) ∈ E

queue.add(u)

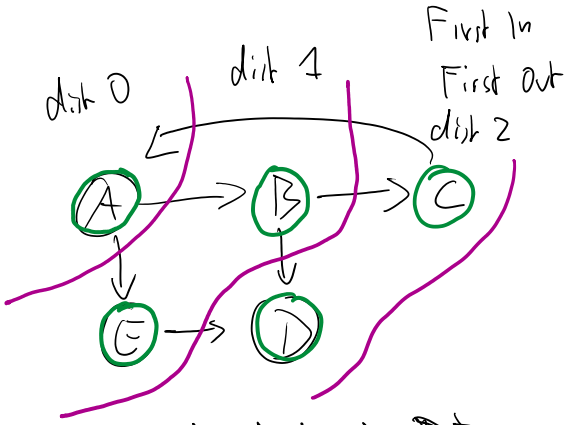
queue is FILO (stack) DFS

First In Last Out

If we use a FIFO then this is BFS

First In

First out



Queu: A B E C D

DFS BFS O(|E|)

BFS visits in order of distance