DFS

→ each time new vertex explored,

Immediately explore neighbours

→ start at 'source vertex' (i) (mark it visited)

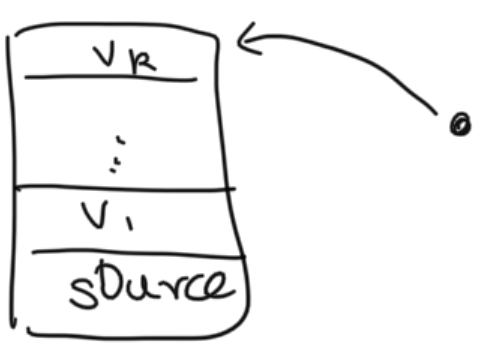
→ Visit first unvisited nbhr of (i), call it (j)

→ suspend exploration of (i) and explore (j) instead

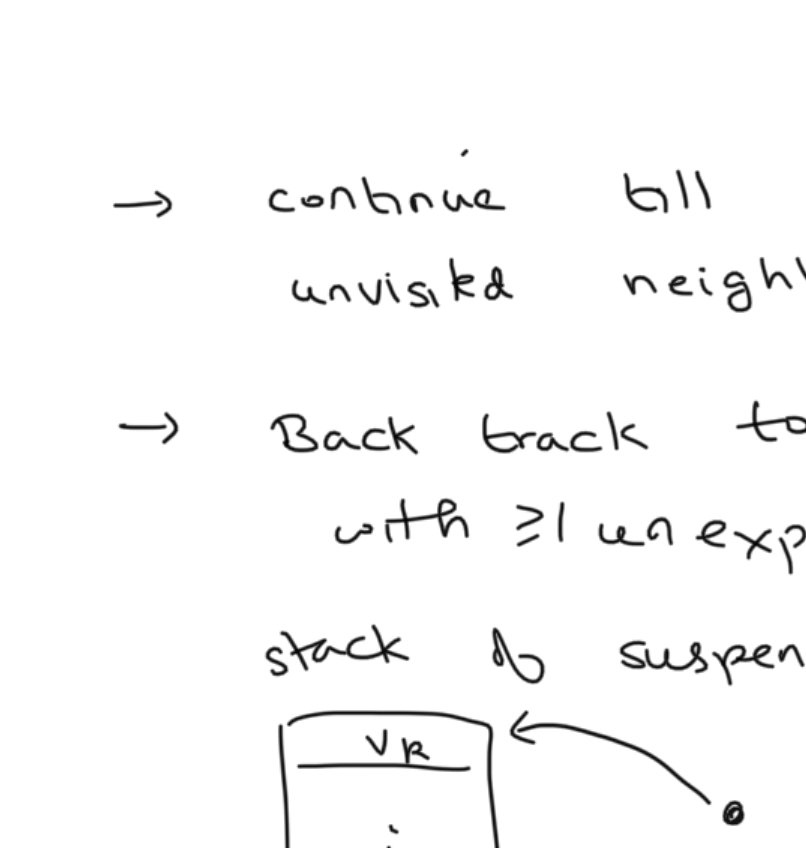
→ continue till you reach vertex with no unvisited neighbours

→ Back track to nearest suspended vertex with ≥1 unexplored nbhrs → ...

stack of suspended vertices..



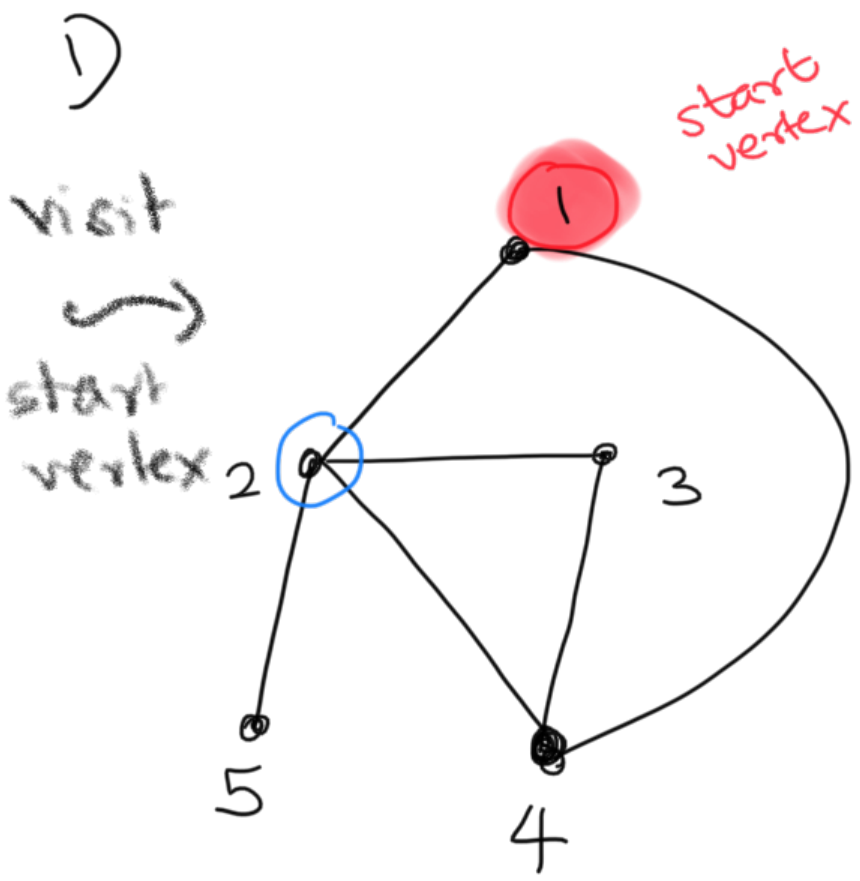
Dry run



1	2	3	4	5	
F	F	F	F	F	visited

∅ stack

1)



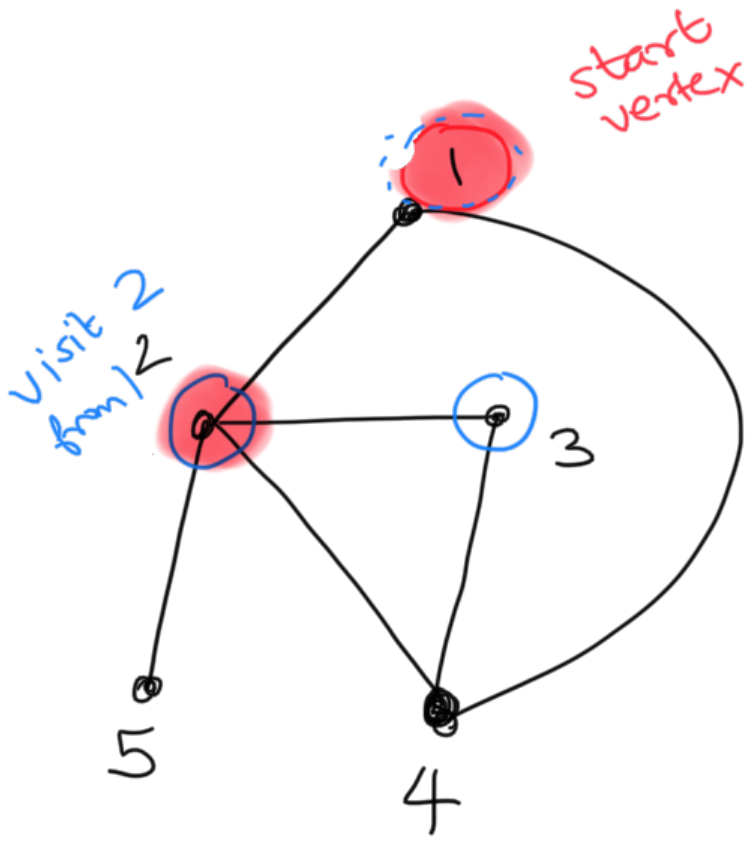
1	2	3	4	5	
T	F	F	F	F	visited

stack

1

suspend exploration of 1

2)



parent (2) = 1

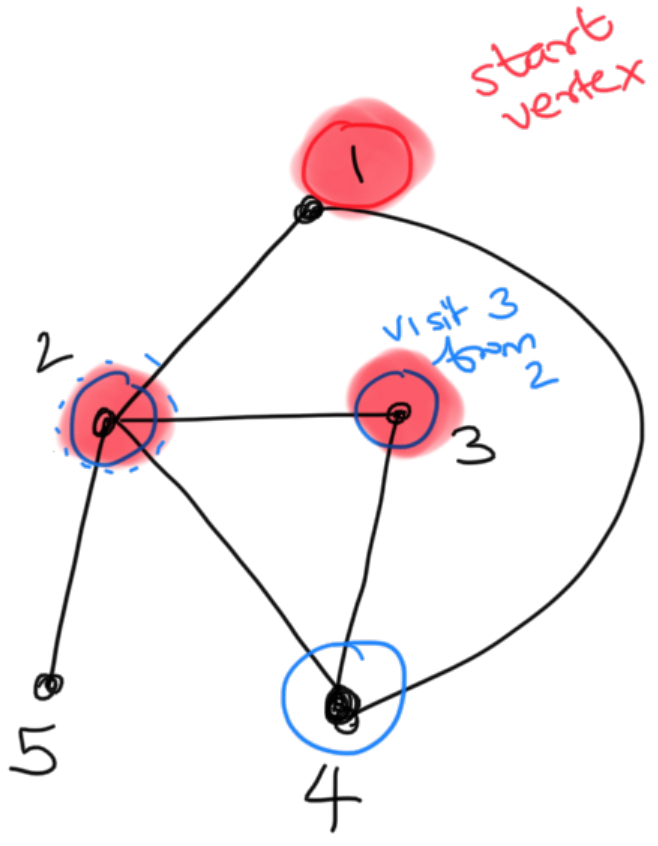
1	2	3	4	5	
T	T	F	F	F	visited

stack

2
1

suspend exploration of 2

3)



parent (3) = 2

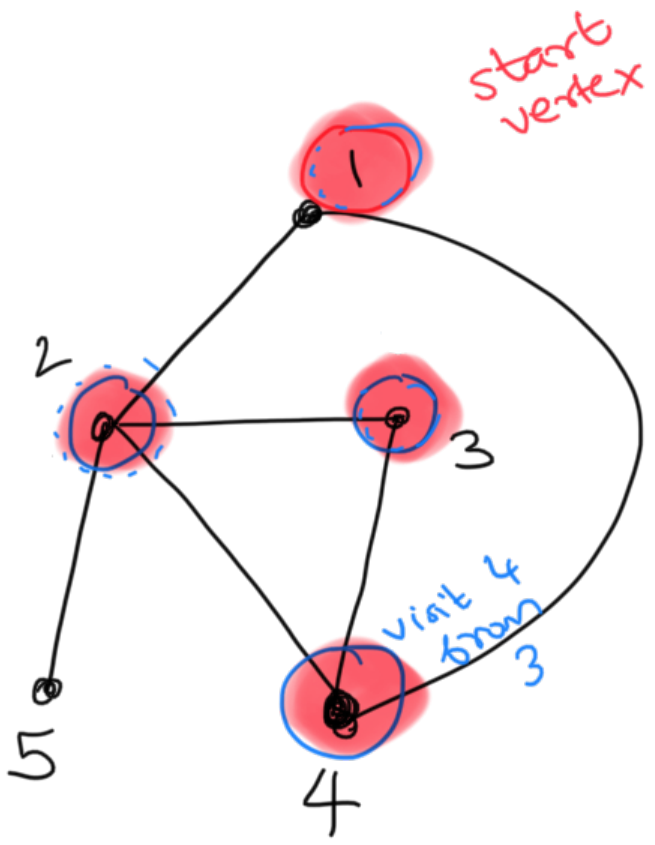
1	2	3	4	5	
T	T	T	F	F	visited

stack

3
2
1

suspend exploration of 3

4)



parent(4) = 3

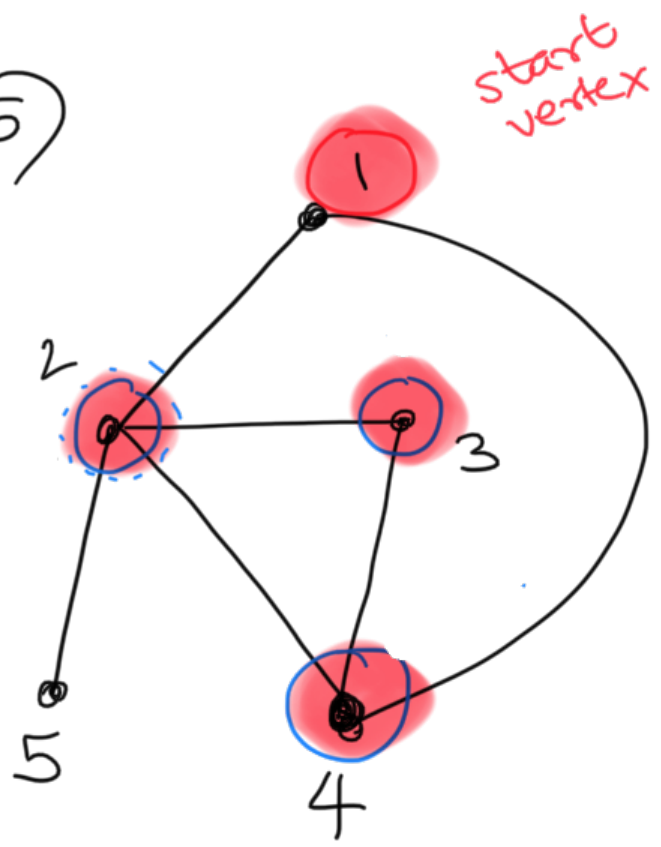
1	2	3	4	5
T	T	T	T	F

visited

stack
3
2
1

4 is a dead end.
All nbhrs explored

5)



1	2	3	4	5
T	T	T	T	F

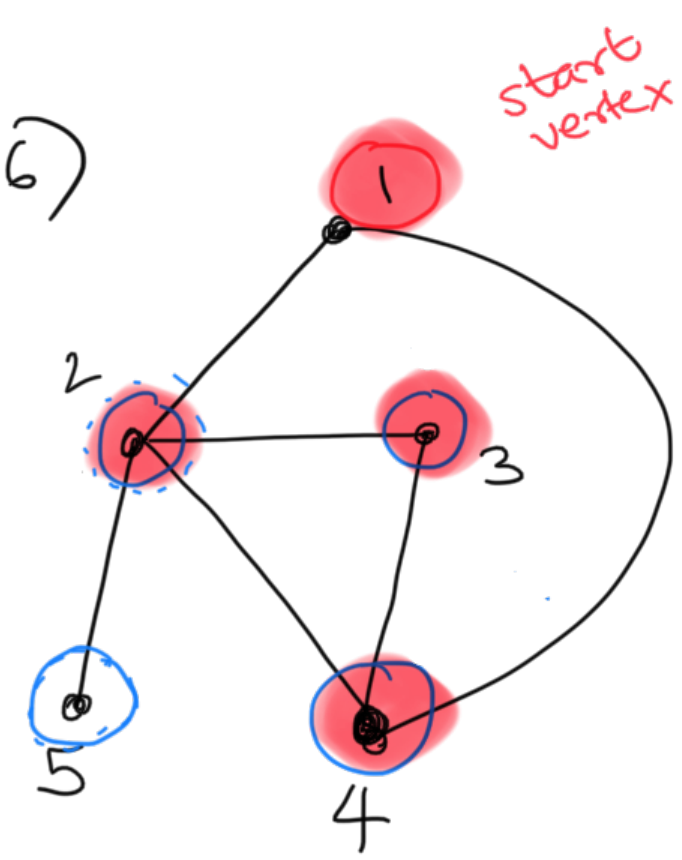
visited

3 ↶ pop 3 from stack to explore

stack
2
1

3 is a dead end!

6)



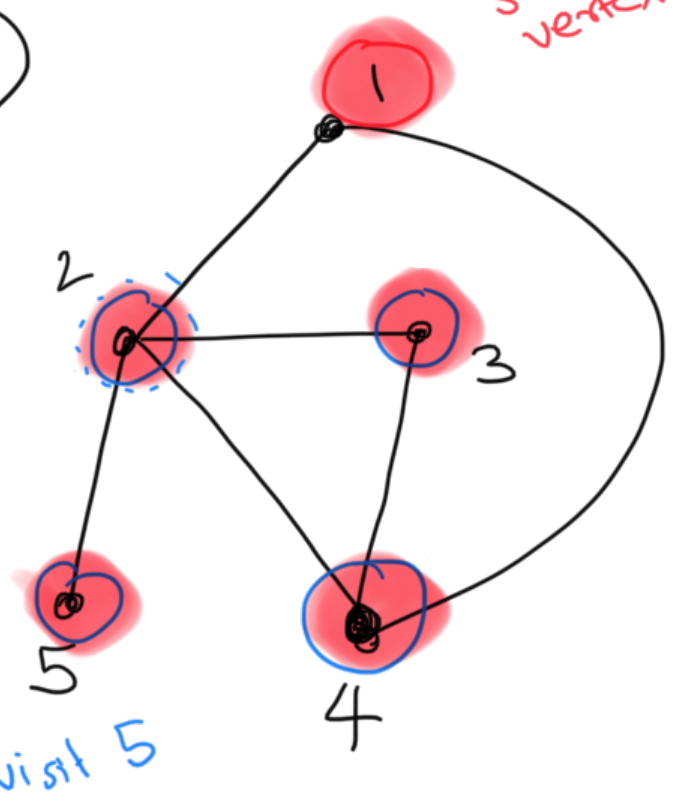
1	2	3	4	5
T	T	T	T	F

visited

2 ← pop 2 from stack to explore

stack
1

7)



Visit 5 from 2

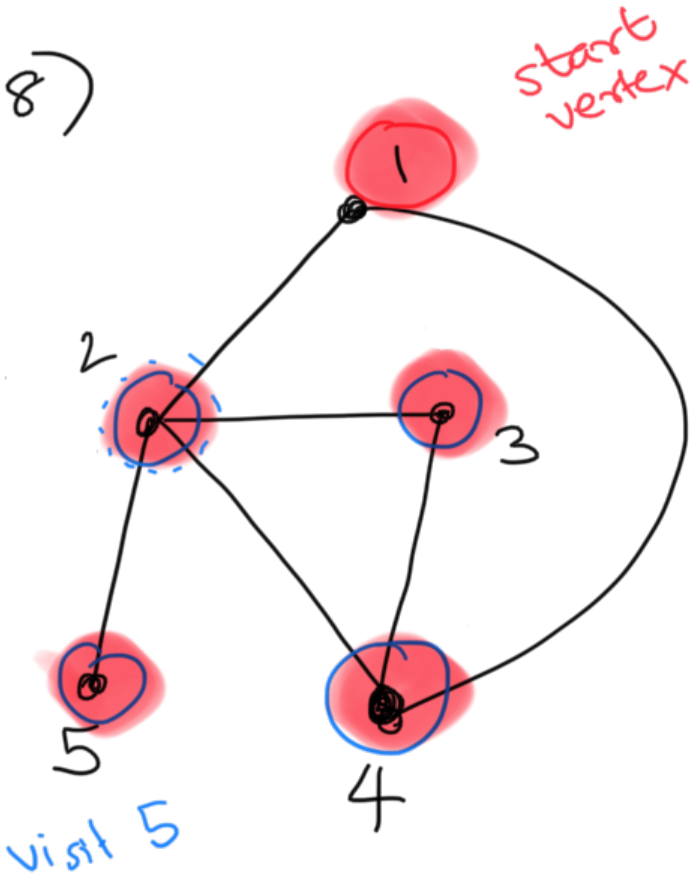
parent(5) = 2

1	2	3	4	5
T	T	T	T	T

visited

stack

8)



Visit 5 from 2

1	2	3	4	5
T	T	T	T	T

visited

pop 1 from stack

↑

∅

1 is a dead end

All nbrs explored -

Terminate.

DFS path recovery

source $\overset{?}{\rightsquigarrow}$ target ?

→ If visited [target] = T, ∃ path bet source, target

target → parent(target), parent(parent(target))

. — . , source

→ if visited [target] = F, ∄ path bet source and target

Implement using recursion ← (implicit stack)

visited : | F | F | . . . | F |

parent : | None | . . . | None |

```
def DFS ( start vertex) :
    visited [ start ] = T
    for w neighbour of start :
        if visited [w] = F ,
            parent (w) = start
            DFS (w)
```