* Depth First Search :

- runs on directed/undirected graphs and ignores any weight
- we assume an underlying adjancy list representation
- exploring the graph within.

* Simple Implementation

DFS (G, startVertex).
    ResetGraph (G).
    DFSvertex (startVertex).
 	
DFSVertex (u)
    u.discovered = True
    for each v s.t u → v
        if(not v.discovered)
            DFSVertex (v)
	ResetGraph(G)
	    for v ∈ V
	        v. discovered = False

- recursive calls
- storing parents is less interesting here. In case of BFS path is always shortest possible but not incase of DFS.
- storing distance/length is also not interesting here due to same reason.
- Instead we use time stamps, finish time and discovery time to do some interesting tasks

* Implementation:

```
DFS(G, startVertex)
    ResetGraph(G)
    DFSVertex(startVertex)

ResetGraph(G)
    for v ∈ V
        v.π = nil
        v.discovery = -1
        v.finishing = -1
    time = 1

DFSVertex(u)
    u.discovery = time++
    for each v s.t u→v
        if(v.discovery < 0)
            v.π = u
            DFSVertex(v)
    u.finishing = time++
```

* One problem here is when we apply DFS from a vertex, there may be vertex which may remain undiscovered.

- if you want to explore the whole graph then.

```
DFS (G)
    ResetGraph (G)
    for u ∈ V
        if(u.discovery < 0)
            DFSVertex(U)
```

* This everything can be visualized in terms of balanced paranthesis. - where (→ signifies when a DFS call is made on a vertex and) → when the call exits

- A child will be nested in its parent

ex: A D B G G B F C C F D A
(((( )) (( ))))
(counting parantheses from the beginning will match discovery/finished times)

* Another approach to time stamps is common vertex colors - white (undiscovered), Grey (discovered, unfinished) and Black (finished)

* Edge classification:

(i) Tree Edge:
• parent to a child.
• go to an undiscovered vertex.

(ii) Back Edges:
- to an ancestor
- to a discovered but unfinished vertex
- every back edges makes a cycle, and removing back edges will remove all cycles. (also self loops are included).

(iii) Forward Edges:

- to a non-child descendant / not a direct child.
- to a finished vertex discovered after the current vertex

(iv) Cross Edges:

- any other edge can go to one branch to another or also from one tree to another tree.
- no ancestor - descendent relationship between the vertices it links.
- to a vertex finished before the current vertex's discovery.

* Undirected Graphs:
- no forward / cross Edges

* Analysis:
- we call DFS on every vertex (once) and explore each edge
- On a graph
- taking time $\Theta(|V|+|E|)$

- on a single vertex with $G' = (V', E')$ reachable from that vertex, takes time $\Theta(|V'| + |E'|)$ & all vertices are initialized.

* Deep Trees may cause Stack Overflow hence we may use our own stack implementation for DFS:

```
DFS (G)
    ResetGraph (G)
    S = new stack()
    for (u ∈ V)
        S.push(u);

    while(not S.isEmpty())
        x = S.pop ();
        if( x.isVertex ())
            ExploreVertex (x, S)
        else
            ExploreEdge (x, S)
```

```
ExploreVertex (u, S)
    if (u. discovery <0)
        u.discovery = time++
        S.push (u)
        for each v such that u → v
            S.push (u → v)
    else if (u.finishing < 0)
        u.finishing = time++

ExploreEdge (u→v, S)
    if (v.discovery < 0)
        (u→v). label = "tree Edge"
        v.π = u
        ExploreVertex (v, S)
    else if (v.finishing < 0)
        (u→v). lablel = "backEdge"
    else if (v.discovery > u.discovery)
        (u→v). label = "forwardEdge"
    else (u → v) . label = "cross Edge"
```