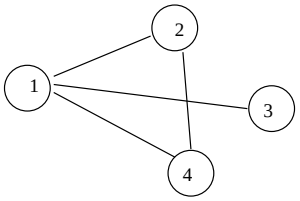
DFS

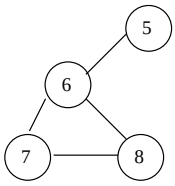
```
Void Graph::dfs(Vertex v)
{  v.visited=true;
   for each vertex w adjacent to v do
       if(!w.visited) dfs(w);
}

dftraversal
{  for every v in G do
       v.visited=false;
   for every v in G do
       if(!v.visited) dfs(v);
}
```

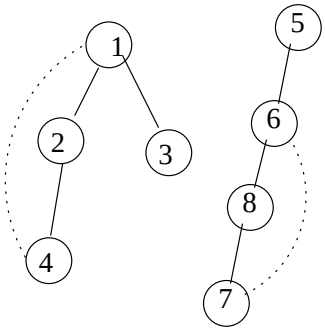
Time complexity $O(n+e)$, n:# of nodes; e:# of edges.

DFS





dfs(1)
dfs(2)
dfs(4)
dfs(3)
dfs(5)
dfs(6)
dfs(8)
dfs(7)

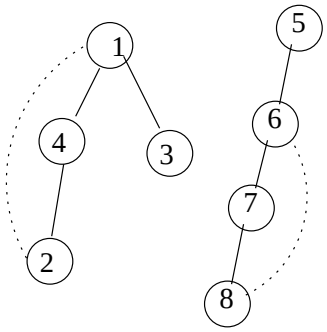


——— : tree edges

····· : back edges

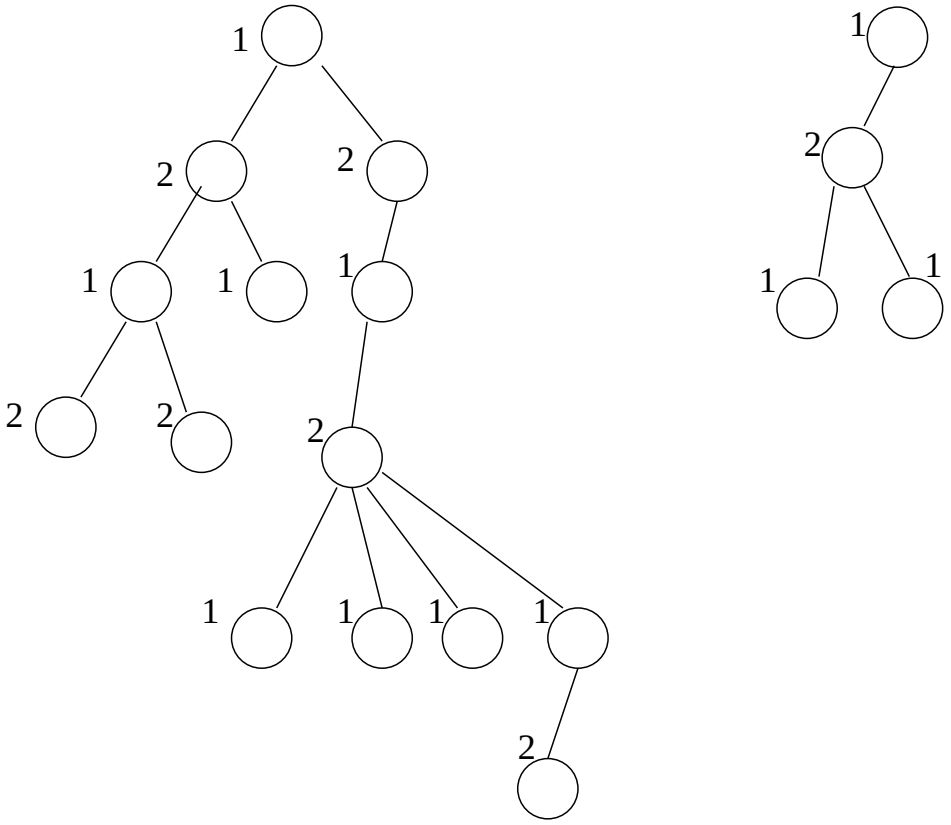
Since edges are not ordered dfs could be

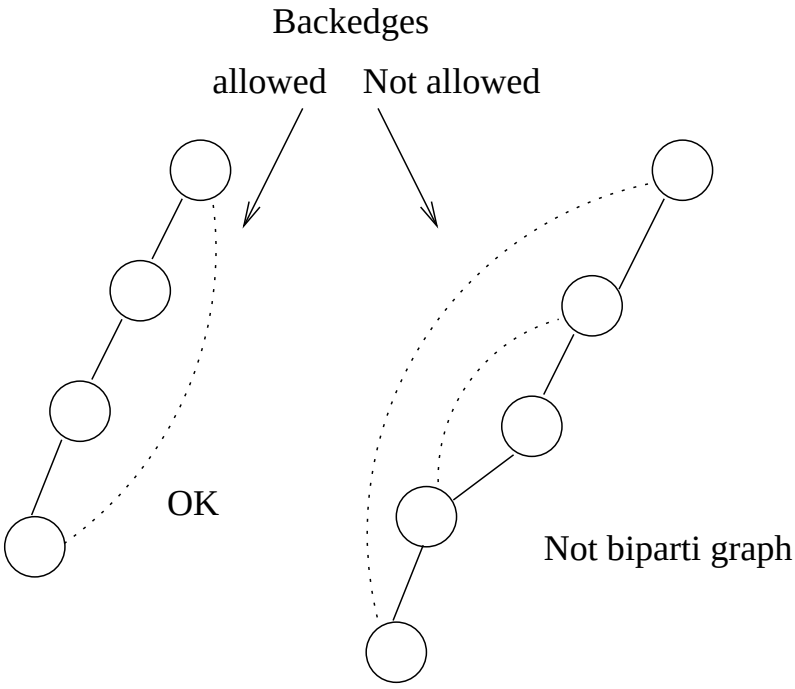
dfs(1)
dfs(4)
dfs(2)
dfs(3)
dfs(5)
dfs(6)
dfs(7)
dfs(8)



- Connectness: Is there a path between every pair of vertices? True iff only one call from dftraversal.
- Connected Components: Partition $G = (V, E)$ into $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$, ..., such that every G_i is connected. There are no edges in G joining a vertex in G_i to one in G_j for $i \neq j$. Solution: the nodes and edges visited during the ith dfs call from dftraversal form G_i.
- Testing if a graph is bipartite: Testing too see if the vertices in G can be partitioned into two sets S_1 and S_2 such that all the edges join a vertex in set S_1 to one in Set S_2.Next slide shows how dfs solves the problem.

Testing for bipartite graph property via dfs





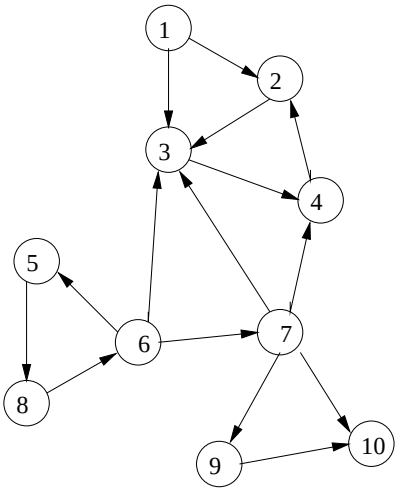
DFS directed graphs

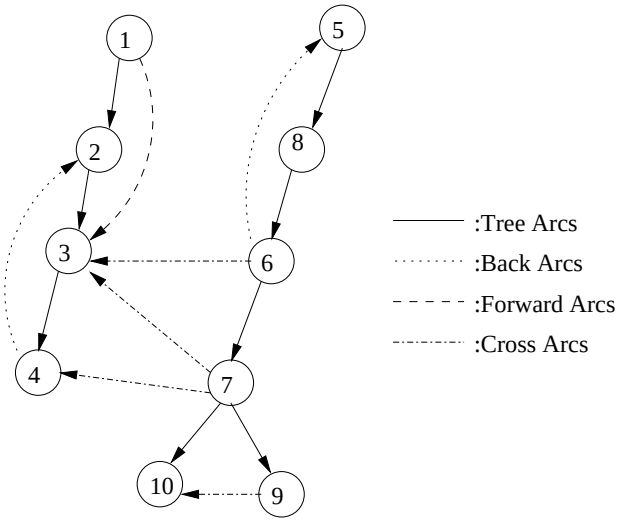
```
Void Graph::dfs(vertex v)
{  v.visited=true;
   for each vertex w adjacent from v do
       if(!w.visited) dfs(w);
}

dftraversal
{  for every vertex v in G do
       v.visited=false;
   for every vertex v in G do
       if(!v.visited) dfs(v);
}
```

Time complexity $O(n+e)$, n:# of nodes; e:# of edges.

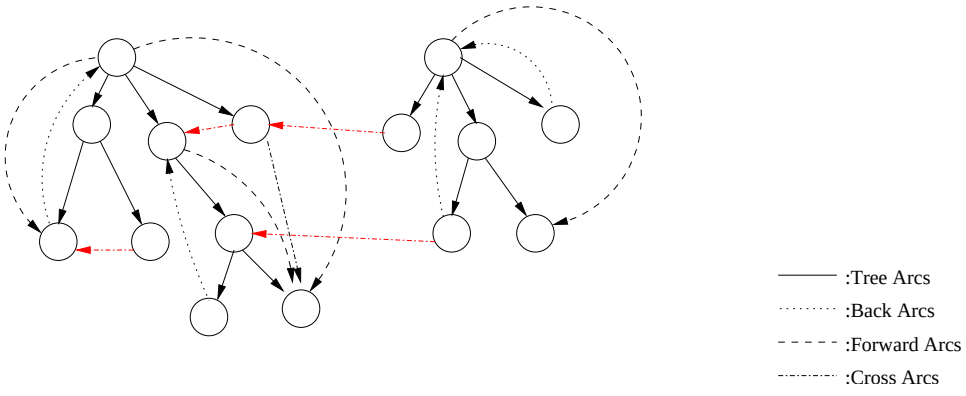
DFS


1
2
3
4
5
6
7
8
9
10


1
2
3
4
5
6
7
8
9
10
:Tree Arcs
:Back Arcs
:Forward Arcs
:Cross Arcs

DFS

- Tree Arcs: Edges in DF spanning forest
- Back Arcs: From a vertex to one of its ancestors in the spanning forest
- Forward Arcs:A non-spanning arc that goes from a vertex to a proper decendant.
- Cross Arcs: From a vertex to another vertex that is neither an ancestor nor a descendant



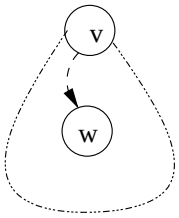
DFS directed graphs

```
Void Graph::dfs(vertex v)
{  v.dfnumber = count; // line not part of dfs
   count++;            // line not part of dfs
   v.visited=true;
   for each vertex w adjacent from v do
       if(!w.visited) dfs(w);}
//When !w.visited is true
//     then (v,w) is a tree edge
//v.dfnumber<w.dfnumber
//     then (v,w) is a forward edge
//else (v,w) is back or cross edge
dftraversal
{  count=1;
   for every vertex v in G do
       v.visited=false;
   for every vertex v in G do
       if(!v.visited) dfs(v);}
```

Time complexity $O(n+e)$, n:vertices; e:edges.

Identification of Arcs

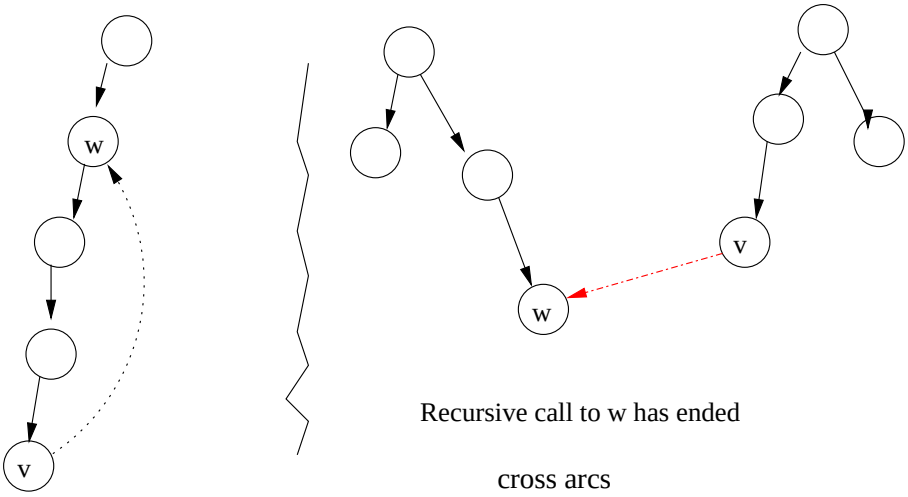
Forward Arcs



w.dfnumber>v.dfnumber

BACK ARCS AND CROSS ARCS

w.dfnumber<v.dfnumber



Recursive call to w has not ended

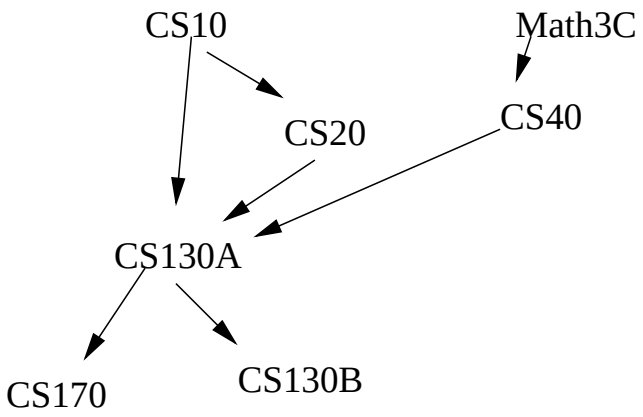
back arcs

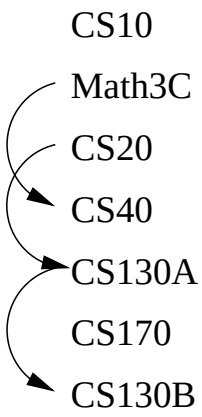
Recursive call to w has ended

cross arcs

Using a mark bit to identify the vertices that are active (in execution stack) one can distinguish between the two cases. $O(n+e)$ total time complexity.

Construct a total order consistent with partial order

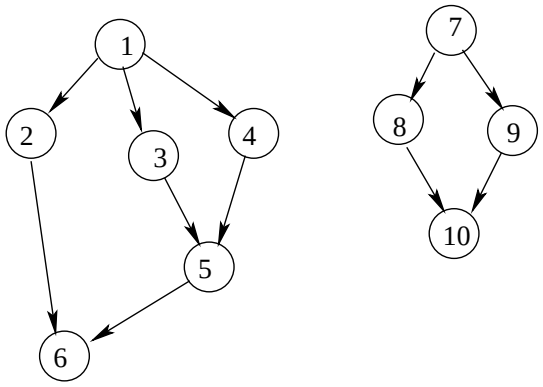




All edges directed from top to bottom.

Print partial order

```
Void Graph::dfs(vertex v)
{  ... // at the end add
   print(v);}
```



dfs(1)	Prints
dfs(2)	6
dfs(6)	2
dfs(3)	5
dfs(5)	3
dfs(4)	4
dfs(7)	1
dfs(8)	10
dfs(10)	8
dfs(9)	9
	7

Reverse the order for a total order consistent with partial order.