
Strong connectivity in digraphs

Definition. Two vertices v and w are **strongly connected** if they are mutually reachable: that is, if there is a directed path from v to w and a directed path from w to v . A digraph is strongly connected if all its vertices are strongly connected to one another.

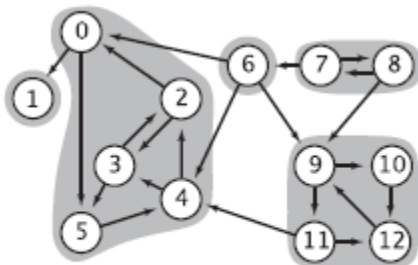
Strong connectivity in digraphs is an equivalence relation on the set of vertices, as it has the following properties:

Reflexive : Every vertex v is strongly connected to itself.

Symmetric : If v is strongly connected to w , then w is strongly connected to v .

Transitive : If v is strongly connected to w and w is strongly connected to x , then v is also strongly connected to x .

Equivalence classes form **strongly connected components**.



A digraph and its strong components

API

```
public class SCC
    SCC(Digraph G)           preprocessing constructor
    boolean stronglyConnected(int v, int w) are v and w strongly connected?
    int count()              number of strong components
    int id(int v)            component identifier for v
                             (between 0 and count()-1)

    API for strong components
```

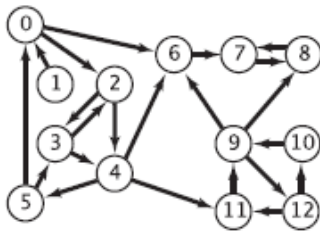
Kosaraju's algorithm

Run standard DFS on G and put vertices in stack as visited.

Use DFS to compute the reverse postorder of its G 's reverse.

Pop element from the stack and use DFS on G , marking visited vertices. This gives strongly connected components.

DFS in reverse digraph (ReversePost)

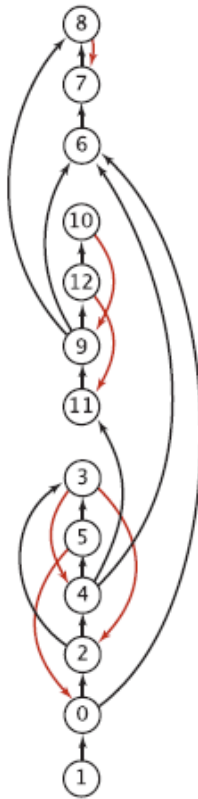


check unmarked vertices in the order

0 1 2 3 4 5 6 7 8 9 10 11 12

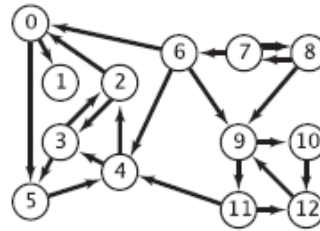
```

dfs(0)
  dfs(6)
    dfs(7)
      dfs(8)
        check 7
        8 done
      7 done
    6 done
  dfs(2)
    dfs(4)
      dfs(11)
        dfs(9)
          dfs(12)
            check 11
            dfs(10)
              check 9
              10 done
            12 done
          check 8
          check 6
          9 done
        11 done
        check 6
      dfs(5)
        dfs(3)
          check 4
          check 2
          3 done
          check 0
          5 done
        4 done
        check 3
      2 done
    0 done
  dfs(1)
    check 0
    1 done
    check 2
    check 3
    check 4
    check 5
    check 6
    check 7
    check 8
    check 9
    check 10
    check 11
    check 12
  
```



reverse
postorder
for use
in second
dfs()
(read up)

DFS in original digraph

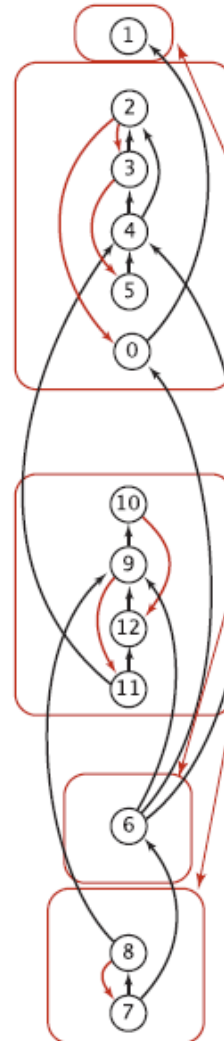


check unmarked vertices in the order

1 0 2 4 5 3 11 9 12 10 6 7 8

```

dfs(1)
  1 done
dfs(0)
  dfs(5)
    dfs(4)
      dfs(3)
        check 5
        dfs(2)
          check 0
          check 3
          2 done
          3 done
          check 2
          4 done
        5 done
        check 1
      0 done
    check 2
    check 4
    check 5
    check 3
  dfs(11)
    check 4
    dfs(12)
      dfs(9)
        check 11
        dfs(10)
          check 12
          10 done
        9 done
      12 done
    11 done
    check 9
    check 12
    check 10
  dfs(6)
    check 9
    check 4
    check 0
    6 done
  dfs(7)
    check 6
    dfs(8)
      check 7
      check 9
      8 done
    7 done
    check 8
  
```



strong
components

Kosaraju's algorithm for finding strong components in digraphs

Theorem: Kosaraju's algorithm uses preprocessing time and space proportional to $V+E$ to support constant-time strong connectivity queries in a digraph.