Kosaraju's Two-Pass Algorithm

1 Algorithm

The strongly connected components (SCC's) of a directed graph G are its maximal strongly connected subgraphs, where a strongly connected subgraph is one where there is a path from each vertex to every other vertex.

```
def kosaraju(G):
   S = stack()
   G_rev = reverse(G)   #Reverse all the edges in G

   #Pass 1
   for v in G_rev:
      if v unexplored:
         explore(v)
         DFS_pass1(G_rev, v, S)

   #Pass 2
   while not empty(S):
      v = S.pop()
      if v explored: #Flipped after pass 1
         unexplore(v)
         SCC rooted at S = DFS_pass2(G, v)

def DFS_pass1(G, v, stack):
   for (v, w) in G:
      if w unexplored:
         explore(w)
         DFS_pass1(G, w)
   stack.push(v)

def DFS_pass2(G, v):
   nodes = []
   for (v, w) in G:
      if w explored:
         unexplore(w)
         nodes += w
         DFS_pass2(G, w)
   return nodes
```

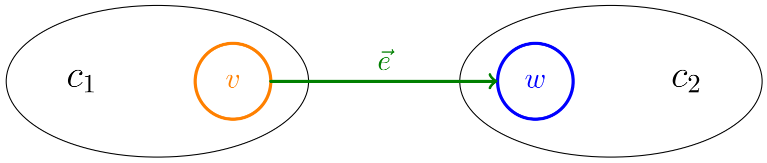
Essentially, the algorithm sorts the vertices in the graph by DFS "finishing time" and then iterates backwards through them to conduct one more DFS to uncover the SCC's. The second pass only explores "sink" nodes of a respective SCC.

Kosaraju's Two-Pass Algorithm

2 Analysis/Correctness

The SCC's of G themselves introduce a "meta-DAG", in that if they are collapsed into nodes and the paths between then are maintained, the resulting graph is also a DAG.

Key Lemma Consider 2 adjacent SCC's in G:

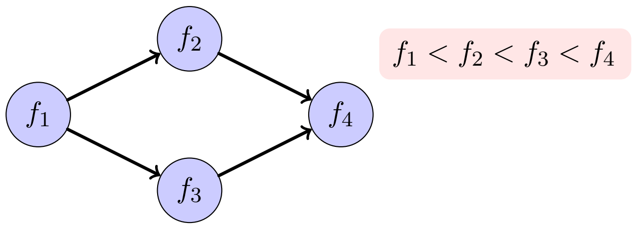


If $f(v)$ is the position on the stack of node v, with higher values corresponding to positions towards the top of the stack, then:

$$\max_{v \in c_1} f(v) < \max_{w \in c_2} f(w) \tag{1}$$

Proof of Key Lemma In the first pass of the algorithm, all of c_1 is explored before c_2 because the meta-graph is acyclic. That is, if the first v chosen in the looped DFS is inside c_2, the algorithm will trace paths all the way to a "sink" vertex in c_1 and push the rest of its nodes onto the stack before backtracking to c_2.

Corollary From the key lemma, we deduce that the maximum f-value (top of the stack) must lie in the sink SCC in the meta-graph.



Correctness Intuition By the above corollary, the second pass of the algorithm begins in a sink SCC, c^*. The first DFS uncovers c^*, and the rest recurses on $G - c^*$.

Runtime Because the algorithm is really just 2 DFS loops, it runs in $O(m + n)$ time with some slow constants.