Finding SCC in a directed graph

Algorithm for finding SCCs

1) Run DFS on $G^T$, the graph with all edges reversed, and order vertices reversely by their post numbers
2) Run DFS on $G$, considering vertices in the above order. Whenever explore terminates, set $SSC_{-}count = SSC_{-}count + 1$, and restart explore(v) at the vertex with highest post value among the left vertices.

Correctness Proof uses

Property 1: If explore is started at $v$ it will terminate when all vertices reachable from $v$ have been discovered. Furthermore, $pre(v)$ is the lowest, and $post(v)$ the highest among all discovered vertices

Property 3: If we order all SCCs by their largest post number, then all edges point forward
Example 1

Order by post(·) in $G^T$

\[
\text{DA} \rightarrow \text{AE} \rightarrow \text{FC} \rightarrow \text{EB}
\]

SCC, largest post, edges in $G$

DFS in $G$:
Example 2:

\[ 6 \]

\[ 6^T \quad \text{SCC-DAG of } 6^T \]
Ordered by post in GT

BFS in G

Path in Graphs
Goal: Calculate distances

Def: \[\text{dist}(u, v) = \text{length of shortest path between } u \text{ and } v\]

distances from S
How to find distances from a source $s$?

**Single Source Shortest Path Algorithm**

**BFS**

- Start from source $s$
- Find its neighbors $\rightarrow$ vertices at distance 1
- Give set of vertices at distance 1, find yet to be seen vertices at distance $1 + 1$

**BFS ($G, s$)**

**Input:** $G = (V, E)$, $s \in V$

**Output:** For all vertices reachable from $u$,
\[
\text{dist}(u) = \text{dist}(s, u)
\]
\[
\forall u \in V : \text{dist}(u) = \infty
\]
\[
\text{dist}(s) = 0
\]
\[
Q = [s] \quad (\text{queue contains } s)
\]

While $Q \neq \emptyset$

- $u = \text{extract}(Q)$
- For all edges $(u, v)$
  - if $\text{dist}(v) = \infty$
    - $\text{dist}(v) = \text{dist}(u) + 1$
    - inject $(0, v)$
Example

\[ \begin{array}{c}
2 \\
E \\
D \\
S \\
A \\
1 \\
2 \\
F \\
P \\
C \\
\end{array} \]

\[ u \quad 2 \quad v \]

- \([S]\) \quad \text{dist} \ 0: \ S
- \([A, C, D]\) \quad \text{dist} \ 1: \ A, C, D
- \([C, D, B]\) \quad \text{dist} \ 2: \ B
- \([D, B]\)
- \([B, E, F]\) \quad \text{dist} \ 2: \ E, F
- \([E, F]\)
- \([F]\)
- \(\emptyset\)

Claim: This gives \(\text{dist}(v) = \text{dist}(s, v)\)

**Proof:** By induction

Assume at time \(d\), we have

- \(\text{dist}(v) = \text{dist}(s, v)\) \ if \(d \leq d\)
- \(\text{dist}(v) = \infty\) \ otherwise

\(Q = \{v \in V : \text{dist}(s, v) = d\}\)

\[ \begin{array}{c}
S \\
\end{array} \]

Claim: Running Time = \(O(|V| + |E|)\)
Dijkstra's Algorithm

For many problems, edges have length:

- **Street Networks**, time it take an infection to infect neighbors in contact network

Formal Setting

Graph $G = (V, E)$

edge lengths $\ell(u,v)$ for $(u,v) \in E$

source $s$

Goal: $\forall w \in V$, Find

$$d(s,w) = \min \sum_{(u,v) \in \omega} \ell(u,v)$$

where the minimum goes over all paths $\omega$ from $s$ to $w$

Example:

![Graph Diagram]

Try to solve systematically, finding nearest vertices first

1. $v_1 = S \quad d_1 = 0$
2. $v_2 = C \quad d_2 = 2$
3. $v_3 = B \quad d_3 = 3$
4. $v_4 = D \quad d_4 = 5$
5. $v_5 = E \quad d_5 = 6$
More systematically

\[ K = \text{known nodes} = \{v_1, \ldots, v_K\} \]
\[ U = \text{unknown nodes} \]
Want to keep \( d(s, v_A) \) as small as possible

**How to find** \( v_{A+1} \)

\( v_{A+1} \) must lie on shortest path \( \omega \)

\[ S \quad \overset{u}{\longrightarrow} \quad v_{A+1} \]

- \( u \) must be in \( K \)
- path from \( s \) to \( u \) must be a shortest path

\[ \Rightarrow \ell(\omega) = d(s, u) + \ell(u, v_{A+1}) \]

\[ d(s, v_{A+1}) = \]

\[ = \min_{u \in K} d(s, u) + \ell(u, v_{A+1}) \]

\[ = \min_{v \in K} \min_{u \in K} (d(s, u) + \ell(u, v)) \]

\[ \Rightarrow v_{A+1} \text{ minimizes } \min_{u \in K} \{d(s, u) + \ell(u, v)\} \]

**Dijkstra** does this inductively, keeping an array \( \text{dist}[v], \forall v \in V \) such that

\[ \text{dist}[v] = \begin{cases} 
  d(s, v) & v \in K \\
  \min_{u \in K} (\text{dist}(u) + \ell(u, v)) & \text{otherwise}
\end{cases} \]

so we can inductively choose \( v_{A+1} \) such that it minimizes \( \text{dist}[u] \) over \( u = V \setminus K \)
Question: After adding \( v_{k+1} \) to \( K \), how do we update \( \text{dist}[w] \)?

Answer: We want to maintain

\[
\text{dist}[v] = \begin{cases} 
  d(s, v) & \text{for } v \in K \\
  \min_{u \in K} \left( \text{dist}(u) + \ell(u, v) \right) & \text{for } v \notin K
\end{cases}
\]

\( K \cup \{ v_{k+1} \} \)

new shortest path in \( K \) via \( v_{k+1} \)

\( \text{dist}[w] = \min \{ \text{dist}[w], \ \text{dist}(v_{k+1}) + \ell(v_{k+1}, w) \} \)

update \( (v_{k+1}, w) \)

\text{Dijkstra}(s, \ell, s)

\text{dist}[s] = 0

\forall v \neq s \quad \text{dist}[v] = \infty

U = V

while \( U \neq \emptyset \)

choose \( u \in U \) n.t.h. \( \text{dist}[u] \) is minimal

remove \( u \) from \( U \)

\forall \text{edges } (u, v) \in E

\text{dist}[v] = \min \{ \text{dist}[v], \ \text{dist}[u] + \ell(u, v) \}

Q: How do we implement this?
Priority Queue
contains a set of
(element, key) pair
vertices dist

Operations
• Insert (elem, key)
• Decrease (elem, key)
• Delete Min \( \leftarrow \) removes elem with lowest key

\text{dijkstra}(6, 2, s)
\text{dist}[s] = 0
\forall v \neq s \quad \text{dist}[v] = \infty
U = V \quad \forall u \quad \text{insert}(u, \text{dist}[u])
while U \neq \emptyset
\quad \text{choose} \ u \in U \ \text{s.t.} \ \text{dist}[u] \ \text{is minimal}
\quad \text{remove} \ u \ \text{from} \ U \quad u = \text{Delete Min}
\forall \text{ edges } (u, v) \in E
\quad \text{olist}[v] = \min \{ \text{dist}[v], \text{dist}[u] + l(u, v) \}

\text{Decrease} \ \text{Key} (v, \text{olist}[v])

Running Time
\|V\| \text{ inserts } \leq \text{ deletes}
\|E\| \text{ decrease}
<table>
<thead>
<tr>
<th>Implementation</th>
<th>delete(\min)</th>
<th>insert / decrease(\text{key})</th>
<th>(V_x) delete(\min) + ((1M+1E) \times \text{insert})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>(O(n))</td>
<td>(O(1))</td>
<td>(O(n^2))</td>
</tr>
<tr>
<td>Binary heap</td>
<td>(O(\log n))</td>
<td>(O(\log n))</td>
<td>(O((n+m) \log n))</td>
</tr>
<tr>
<td>d-ary heap</td>
<td>(O\left(\frac{\log n}{\log d}\right))</td>
<td>(O\left(\frac{\log n}{\log d}\right))</td>
<td>(O(nd + m \frac{\log n}{\log d}))</td>
</tr>
<tr>
<td>Fibonacci heap</td>
<td>(O(\log n))</td>
<td>(O(1))</td>
<td>(O(n \log n + m))</td>
</tr>
</tbody>
</table>