**Strongly Connected Components**

---

Observation 1:

If we start DFS at a sink "vertex" then we can identify the entire sink component.

Q: how to identify a vertex in the sink?

Observation 2:

Any sink in $G$ is a source in $G^R$.

$G^R$ is the reverse graph.

Observation 3:

We can identify source by decreasing order of DFS.
we can identify source by decreasing order of DFS post numbers.

1. \( \text{post}(v) > \text{all reachable node} \)
   
   \[ \text{post}(c) > \text{post}(c') \]

2. what if we choose \( x \in C' \)
   
   \[ \text{post}(v) > \text{post}(x) \]
   
   \[ \text{post}(c) > \text{post}(c') \]

---

See Algorithm: \((G)\)

\[ O(E+V) \]

1. Create \( G^k \)

\[ O(E+V) \]

2. Use DFS post numbers on \( G^k \)

\[ O(V) \]

[For all] \( [\text{pick the highest unvisited } \text{post}(v)] \)

\[ O(V+E) \]

[use DFS in G to mark all reachable vertices]

output this subset as \( \text{src} \)
\[ O(V+E) \] output this substep as a set

- \( V \log V \) time to sort!
- \( E = O(|V|) \) then graph is sparse

\[ O(V+E) = O(|V|) \]

**BFS: Breadth First Search**

Shortest path from a given vertex \( u \).

\[ d(u, x) \quad \forall x \text{ in } G \]

- length of shortest path from \( u \) to \( x \)
- \# of hops

Generalize to cost based

\[ d(u, x) = \infty \quad \text{if } u \text{ cannot reach } x. \]
Single Source BFS (u):

\[
\text{for all } x \in V \\
\quad d(u, x) = \infty \\
\quad d(u, u) = 0
\]

Insert u in queue Q

While Q not empty

\[
\nu = \text{dequeue}(Q)
\]

\[
\text{for all neighbors } x \in \text{u}
\]

\[
\text{if } d(u, x) = \infty \\
\quad d(u, x) = \text{dist}(u, u) + 1
\]

Extra edge over all vertices

Add x to the Q.

Induction to prove correctness.

Time for len(Q)

Total time \(O(V + E)\)
Dijkstra's Algorithm

Given a weighted graph

\( f(v, u) \) is the weight on edge \((v, u)\)

All weights are positive

Algorithm:

1. Start at \( A \)
2. Set distance to \( A \) as 0, and distance to all other vertices as \( \infty \)
3. Select the node \( x \) with the smallest distance from \( A \)
4. If \( \text{distance}(x) = \infty \), stop.
5. For each neighbor \( u \) of \( x \), calculate new distance to \( u \)
6. Update the distance to \( u \) if the new distance is smaller than the current distance
7. Repeat steps 3-6

Example:

\( Q = \{ A, B, C, D, E \} \)

Choose the children.

Every time we pick the node with the shortest distance.

From \( A \):

\[ Q = \{ A \} \]

But we cannot just arbitrarily choose the children.
we will maintain a priority queue

1. pick the minimum cost node
2. change the distances.

Dijkstra (U)

\[ \forall x \in V, \quad d(u, x) = \infty \quad \text{if} \quad d(x) = \infty \]
\[ d(u) = 0 \]

Insert u in PQ. \( \leftarrow \) Priority Queue

while PQ not empty.

\[ u = \min (PQ) \leftarrow \]
for all \( x \in \text{neighbours}(u) \):

\[ d(x) = \min \{ d(x), d(u) + l(u, x) \} \]

Choose weight of \( x \) in PQ

PQ: Priority Queue

Array PQ

\[ \begin{bmatrix}
1 & 2 & 3 & \cdots & n \\
\hline
d(1) & d(2) & \cdots & d(n)
\end{bmatrix} \]

Array of vertex and their weights

\[ \mathcal{O}(N) \] to extract \( \min \)\

Changing the weight: \( \mathcal{O}(1) \)
Charging the washer: \( O(1) \)

While loop is called \( O(1V) \) time

there are \( |V| \) calls for \( \text{extract min PQ} \)

\( O(1V) \)

\( O(1V^2) \)

Total cost of charging weight is \( O(1E) \)

\( O(1V^2 + 1E) \)

\( O(1V^2 + 1E) \) \( \rightarrow \) good for dense graphs.

Very bad for sparse graphs.

---

Priority Queue based on binary trees

\( \text{extract min} : O(\log |V|) \)

\( \text{decrease key} : O(\log |V|) \)

\(|V| \) calls to \( \text{extract min} \)

\(|E| \) calls to \( \text{decrease weight} \)

\( O\left(\left(\frac{|V| + |E|}{\log |V|}\right)\right) \)

\( \frac{|V|}{\log |V|} \) factor

---
vertex \( x \) has child \( X \)

1. root is always the least/min of all node
2. any node is smaller than all children

Extract min:
1. return 3 as the min
2. after the tree:
   a) pick the last node (rightmost leaf)

worse case:
\( O(\log_2 |V|) \) \{

decrease key:
swap upwards until we reach the common level

worse case:
\( O(\log_2 |V|) \)

create the pq:

Idea 1: sort all vertices by weight & insert into pq.

1 2 3 4 5 6 7 8 9 10

\( o(|V| \log |V|) \)
\[ 9 7 1 0 3 1 5 4 2 6 8 \]

\[ \text{(111)} \]

- For all node trickle up.
- \( O(111) \) time

1. From each node trickle up
2. Any node that is swapped, trickle down

\[ 2 \log |V| = O(\log |V|) \]