

Tarjan's Rivalry HV3

Q1) Given: Undirected graph $\rightarrow G$
Edge $\rightarrow e$

Goal: Find if G has a cycle containing e
in linear time

* Depth First Search (DFS) *

↳ Visits all vertices and marks them as not visited

↳ Visits all vertices again and if vertex v is not visited, explore the vertex, checking all adjacent points reachable from v .

Step 1: Run DFS normally from v to v
If v isn't visited, but e is, e ~~is~~ is contained in a cycle.

Step 2: If step one was false, run DFS in reverse, moving from v to v .
If v isn't visited, but e is, e is contained in a cycle. If not, e is not contained in a cycle.

Runtime: $O(|V| + |E|)$

The runtime of DFS is $O(|V| + |E|)$ and we perform DFS twice.

$O(|V|)$ checks all vertices in the visited set for step 1.

$O(|E|)$ handles the explore portion.

Justification: This works, because we are checking for back edges which are necessary to create a cycle.

Directional step -> \leftarrow if the edge has
back edges

front/back edges

This is mostly true, but we will see that it's
not always true

Because if back edges exist, you might
include them all as a legitimate
second choice in taking direction

Not so much because if all of them
is a back edge, then it's
already included in back

The next point we can make is that

if there is no back edge, then it's
not a back edge

& thus it's always a legitimate
second choice

• If there is no back edge

• Then it's a valid second choice

• And a back edge, then it's

obviously a back edge, so it's not

legitimate as a second choice

• Because it's already included in back

(Q2) Given: graph of the city $\rightarrow G = (V, E)$

V is the intersections

E is one way roads

a) Since the streets are one way we have a directed graph. The claim is that there is a way to legally drive from any intersection to another. This falls under the relation for strongly connected component. Two nodes v and v' of a directed graph are connected if there is a path from v to v' and v' to v . This can be checked in linear time by using DFS (Depth First Search)

b) Using the same directed graph, taking the townhall as s , we can perform DFS again for an intersection (vertex) v and repeat for all vertices. The property will hold true if s cannot reach a vertex with a different connected component. If the vertex has a different connected component, it cannot make it back to s .

Runtime: $O(|V| + |E|)$ \rightarrow DFS is run for both situations.

Justification: Directed graphs are connected if there is a path from v to v' and from v' to v .

Tarjan's Rollback HW3

(Q3)

Given: Graph of roads $\rightarrow G = (V, E)$

$V \rightarrow$ Cities

$E \rightarrow$ Roads

$|e| \rightarrow$ length of Road E

$E' \rightarrow$ list of city pairs

Goal: find the road that creates maximum decrease in driving time

Dijkstra

↳ Takes input of graph, vertices and edge lengths

↳ finds shortest path from starting point to every vertex by attempting every path and exiting once the distance becomes greater.

Step 1: Run Dijkstras using s as starting point

Step 2: Run Dijkstras using t as starting point.

*Now we have all possible distances between s and t *

Step 3: Cycle through E' and choose the path that creates the shortest distance between s and t

Runtime: $O(|E| + |E| \log |V|)$

$O(|E|) \rightarrow$ Searching E'

$O(|E| \log |V|) \rightarrow$ Dijkstras

Dijkstra's

Justification: We run twice for the undirected graph to precompute the shortest distances from s and t so we can search for the closest pairs of cities from E' to s and t .

What makes this problem easier is "locality".
if two points are close then there is a
shorter time of computation of x-axis with
respect to one point than others.

Two points will have minimum distance

from point to point and it will

reduce possibility of being far away from each other

at least

using 2D points and 2D distance check and
reduced search radius for each point

so that it will be faster

(N) $\log(2) \cdot n^2 \log(n)$ time

and n^2 space

and n^2 time

Torad Ralola HW3

Q4) Given: Weighted graph $\rightarrow G = (V, E)$
positive weights $\rightarrow l_i$ for $i \in E$

Goal: Decide if input $e = (v, v) \in E$ with
weight (e is part of an MST of G)
in linear time.

Step 1: Remove all weighted edges having a
greater weight than l_e

Step 2: With the new graph, run DFS
(Depth First Search) to look for a
cycle.

* Cycles break tree properties, so
if a cycle exists, it cannot be a
tree *

Step 3: If no cycle is found: v cannot be
reached from v with max weight
being l_e . There is a minimum
spanning tree containing e .

Runtime: $O(|E| + |V|)$

$O(|E|) \rightarrow$ Traversing all edges to remove
those with higher weight than
 l_e .

$O(|E| + |V|) - 1$ DFS

Justification: Since we do not have to find the MST; just the existence of one, we only have to prove that a cycle does not exist to uphold the properties of trees being acyclic. This means we can use DFS to check for cycles.