Project Report

Lokesh V

Yash Mehan

Tarjan and Jen Schmidt Algorithms

Jens Schmidt Algorithm

This algorithm is for finding the lobes in the ear decomposition in the graph. We compute the DFS tree of the graph. Thus we have essentially the same graph, but with some sense of directionality in the edges: the tree edges are all directed towards the root. Non tree edges play a pivotal role in this algorithm, because the ear decomposition starts out from C_1 being a cycle. The next lobe, C_2 (if it exists) has its two endpoints on C_1. In general, C_i has its endpoints on $C_1 \dots C_{i-1}$. Since evey lobes are build bottom up (i.e. C_1 to C_n) (where n is the number of backedges), the backedges, because of whom the cycles are shown in the graph become important.

Pseudocode

```
input(graph)
do ear decomposition of graph
print out the lobes
if there is more than once cycle
  print "more than one bi connected components"
```

After getting a DFS tree, we walk in the backedge from the earliest visited node. we keep walking until we reach back to the node, and mark all vertices as visited on the way. This is a cycle, and is the first lobe C_1. The next lobe starts from the backedge out of the second earliest visited node, and keep walking until we reach a visited vertex. Repeat this process for each backedge. If we find a cycle other than the first, this implies there is another biConnected component present in the graph. If C_1 is the only cycle, then the graph is one entire biCC.

Run times

	V+E	all ear _print	bicon or not?	
5k_full	12502500	3.163	2.019	
8k_full	32004000	9.52	5.512	
10k_full	50005000	15.505	8.848	
cti	65072	0.042	0.024	
delauneaY	131042	0.065	0.054	
mycielskia	16740391	4.573	2.888	
dictionary	141690	0.43	0.032	
erdos	13197	0.2	0.019	
fe_4	339361	0.032	0.028	
fe_body	208821	0.053	0.052	

Tarjan Algorithm

Case: Root R

- If `R` has more than 1 children Then `R` is a articulation point

Case: Non - Root vertex

- For each non root vertex `v` find the `low(v)`

`low(v):`

1. `low(v) = d(v)`
2. `low(v) = min(low(v),(low(w))` for all `w` which is a child of `v`
3. `low(v) = min(low(v),(d(x))` for all `x` where there exists a backedge from `v` to `x`

Articulation point condition

1. for a given root vertex `v` if one of children's `low(w)` is greater than or equal to `d(v)` then `v` is an articulation point
2. Condition: `d(v)<=low(w)` for atleast one child `w` than `v` is a articulation point,

3. In other words `d(v)<= max(low(w))` then `v` is a articulation point.

How to compute this?

1. First do a DFS find the rooted tree with its discovery times
2. Bottom up do the `low(v)` calculation
3. Now for each node check the articulation point condition.

Finding Bridges from this?

1. Bridges have at least one end-point as an articulation point.
2. For a edge vw(v->parent,w->child)
   => If `d(v)<low(w)` then `vw` is a bridge

finding connected components

- Lets take a `u`,`v`,`w` ,such that `u` is the child of `v` and `v` is child of `w`
- if `v` is articulation point due to `u` , then `uv` and `vw` belong to different components
- Else both belong to same biconnected components

V+E	time
12502500	1.588
32004000	4.52
50005000	7.936
65072	0.034
131042	0.047
16740391	2.304
141690	0.041
13197	0.016
339361	0.028
208821	0.071