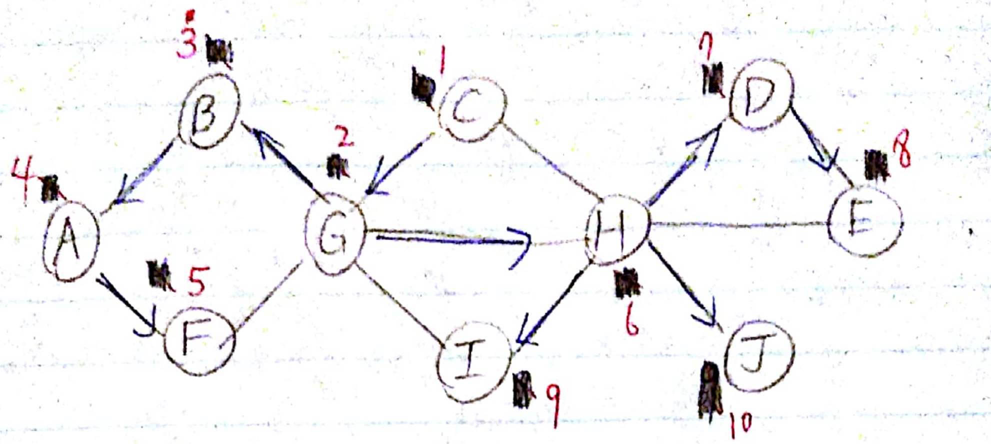
CSE 417

HW2 _ Yao-Chung Liang _ 1826630 _ yliang2@uw.edu

1.



(a) tree edges:

[CG], [GB], [BA], [AF], [GH], [HD], [DE], [HJ], [HI]

(b) back edges:

[GF], [CH], [HE], [GI]

(c)(d)

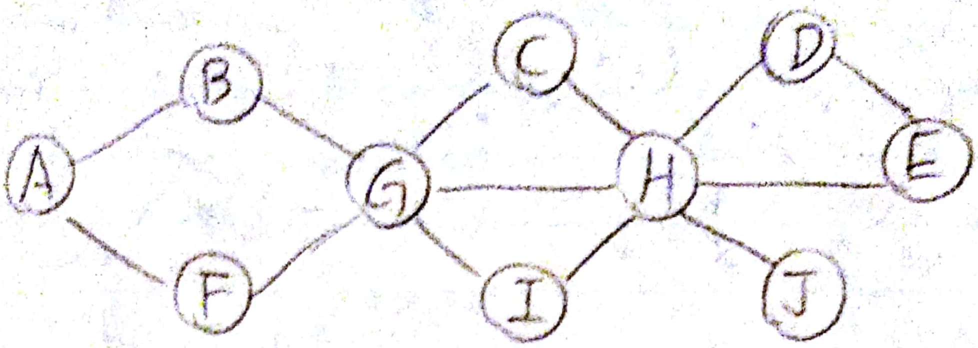
vertex	dfs #	Low
A	4	2
B	3	2
C	1	1
D	7	6
E	8	6
F	5	2
G	2	1
H	6	1
I	9	1
J	10	6

(e) articulation points:

G, H

(f) [CG] → [GB] → [BA] → [AF] → [FG] → [GH] → [HD] → [DE] → [EH] → [HI] → [IG] → [HJ] → [HC]

2.



Biconnected components:

① [A B] [B G] [G F] [F A]

② [C G] [G H] [C H] [G I] [H I]

③ [H D] [D E] [E H]

④ [H J]

Scanned by CamScanner

3. modification of articulation-point algorithm

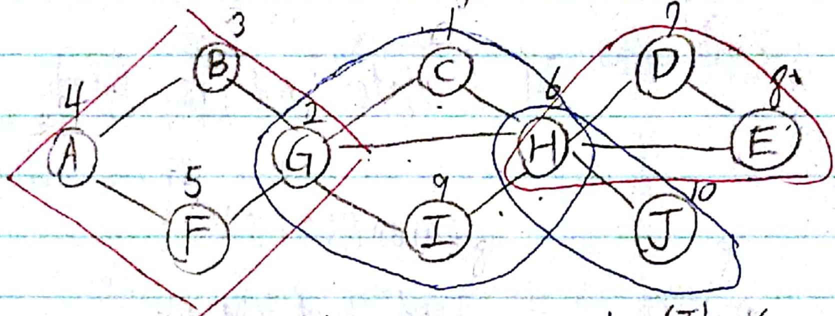
```
def Find_Art ( Vertex V) :
  { Visited (V) is true
    Low (V) = counter ++
    dfs# (V) = counter ++
    for each W adjacent to V
    {
      If (W is not visited) :                // tree edge
      { V is W's parent
        store (V, W) edges in edges_tracker
        Find_Art (W)
        if ( Low (W) ≥ dfs# (V) ) :
        {
           V is an articulation point
        }
        Low (V) = min { Low (V) , Low(W)}
      }
      else
      {
        If (W is not V's parent) :              // Back edge
        {
          Low (V) = min { Low (V), dfs# (W)}
          If ( all W adjacent to V are visted)
          {
            pop out edge_trackers till we reach back
            the node with dfs# == Low(V)
            then connect these nodes and become a biconnected graph
          }
        }
      }
    }
  }
```

3、

the modification is when it encounters a situation that (W adjacent to V is not V's parent) and (all W adjacent to V is visited)

then start to pop out the tree edges I stored before, till it reaches the node with dfs # == Low (V) then connect these nodes together become a biconnected graph

Simulation :



Low(F) = 2
Low(A) = 2
Low(B) = 2
dfs(G) = Low(F) = 2
⇓
become first biconnected graph

Low(E) = 6
Low(D) = 6
dfs(H) = Low(E) = 6
⇓
second biconnected graph

Low(J) = 6
dfs(H) = Low(J) = 6
⇓
third biconnected graph

Low(I) = 1
Low(H) = 1
Low(G) = 1
dfs(C) = Low(I) = 1
⇓
Fourth biconnected graph

edge_tracker

CG
GB
BA
AF
~~FG~~ ← back edge
⇓
pop AF BA GB ⇒ biconnected

CG
GH
HC
HD
DE
~~EH~~ ← back edge
⇓
pop DE HD
⇓
biconnected

CG
GH
HC
HI
HJ
⇓
pop HJ
⇓
biconnected

CG
GH
HC
HI
IG
⇓
backedge
⇓
pop CG GH HC HI IG ⇒ biconnected

CSE417_HW2_Yao-Chung Liang_1826630_yliang2@uw.edu

4.

Results:

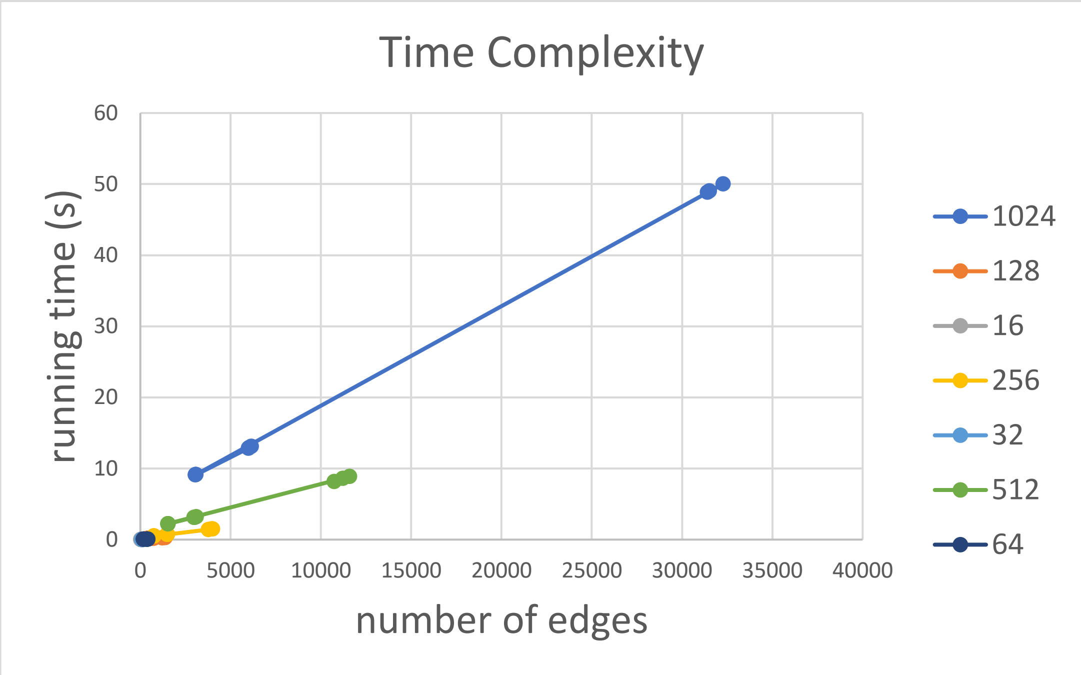


Fig.1 Time complexity according to different number of edges.

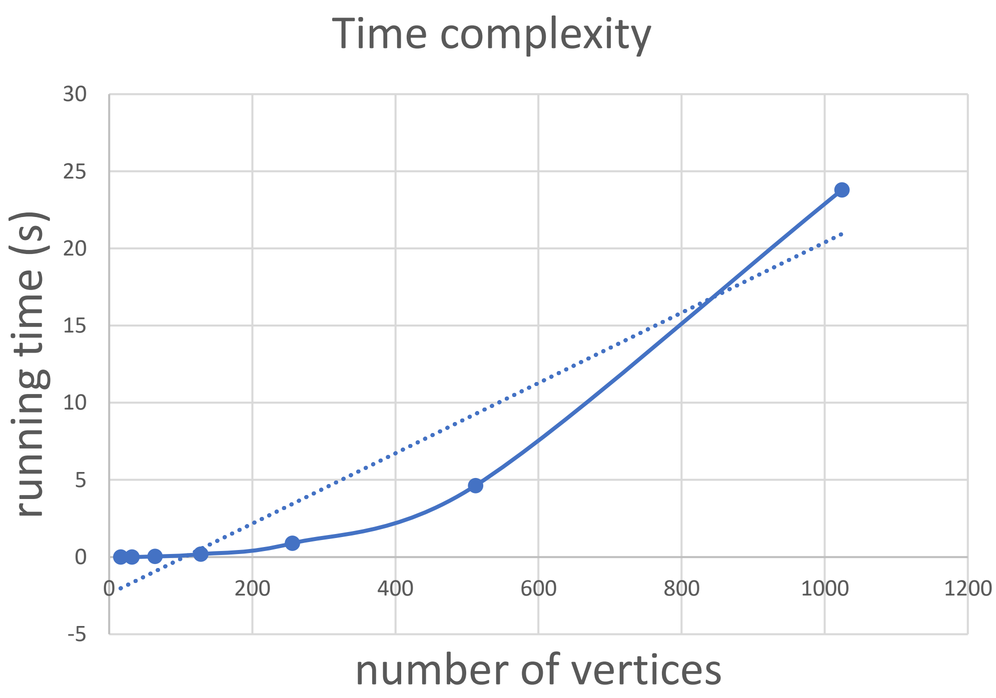


Fig.2 Time complexity. Running time is the average time spent on certain number of vertex with different number of edges.

Analysis:

From Fig.1 I got from different size of input file and ran on my computer, I found the dependence between edges and running time is about linear time complexity. And the interesting thing is that even though there are the same number of edges with different number of vertices, the running time would also increase a little bit by the increasing number of vertices. And after I analyzed the complexity by summing up the number of vertices (V) and edges (E), I found that the complexity is also follow the O(V+E).

And it seems like number of vertices have more impact on the growing running time. And the vertices are more informative is because each vertex can generate n edges if there are n vertices in the graph, but an edge can only connect two vertices together. And from Fig.1, we can see that as the number of vertices increase, the slope become even steep. Compared to the impact of vertex, the edges are less important, for they are dependent on vertices.

After I analyze my algorithm, I found there is still some discrepancies. Like from my analysis of my algorithm, there should be only linear time complexity, however from the graph I observed it's not really O(n) but maybe $O(n^{1.2})$ or more. It may seem like no big deal, but when dealing real problems, the nodes would be billions, thus I still need to try to lower the order. And also, I find I did too much data preprocess in my code, I should do it in an easier way and should not waste a lot of memory to do these stuffs.

On the other hand, the results are varied from each other's hardwares, so it's more meaningful to use big O-analysis to determine which algorithm is more efficient.

My processor is Intel(R) Core(TM) i7-8750H CPU @2.20 GHz, memory system is RAM@ 15.6Gb.