Cut Vertex & Cut Edge

Articulation Point / Cut Vertex

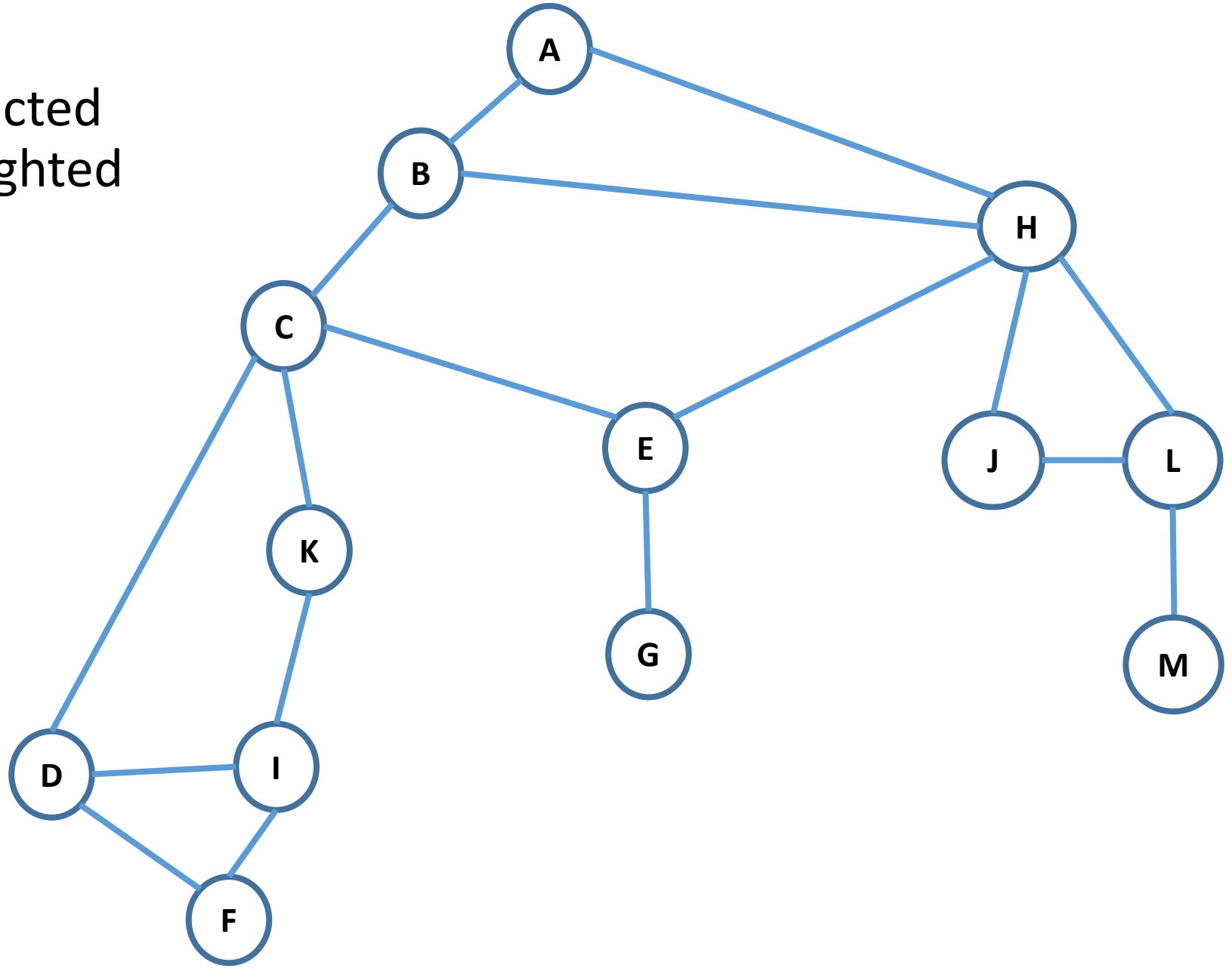
- A node in an undirected graph is an **articulation point or cut vertex** iff removing it disconnects the graph while creating two or more components.
- Articulation points represent vulnerabilities in a network – single points whose failure would split the network into 2 or more disconnected components

Cut Edge

- An edge in an undirected graph is a **cut edge** iff removing it disconnects the graph while creating exactly two components.
- Cut edges represent vulnerabilities in a network – single link failure would split the network into 2 disconnected components

Input:

Undirected
Unweighted
graph



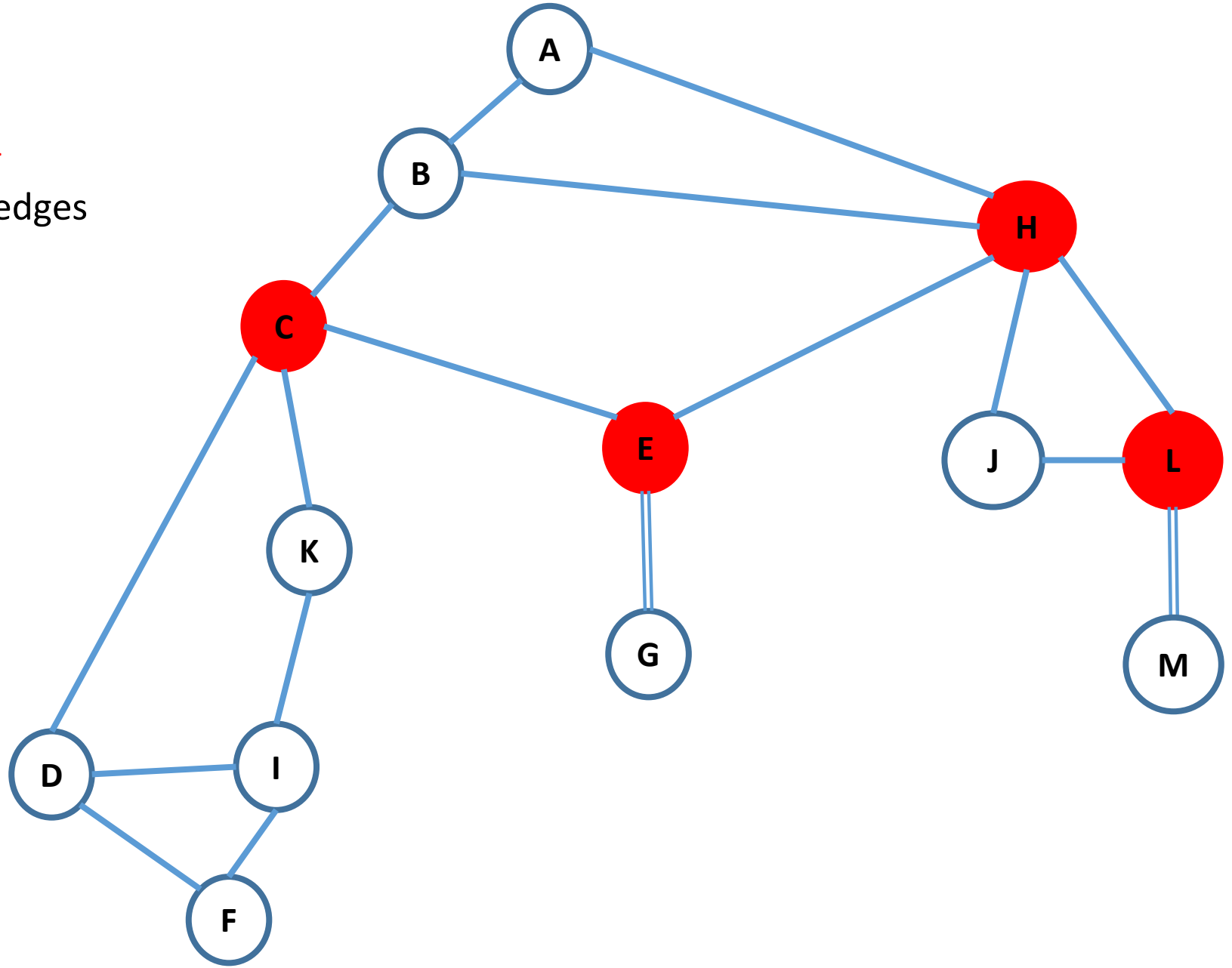
Output:

Set of cut vertices

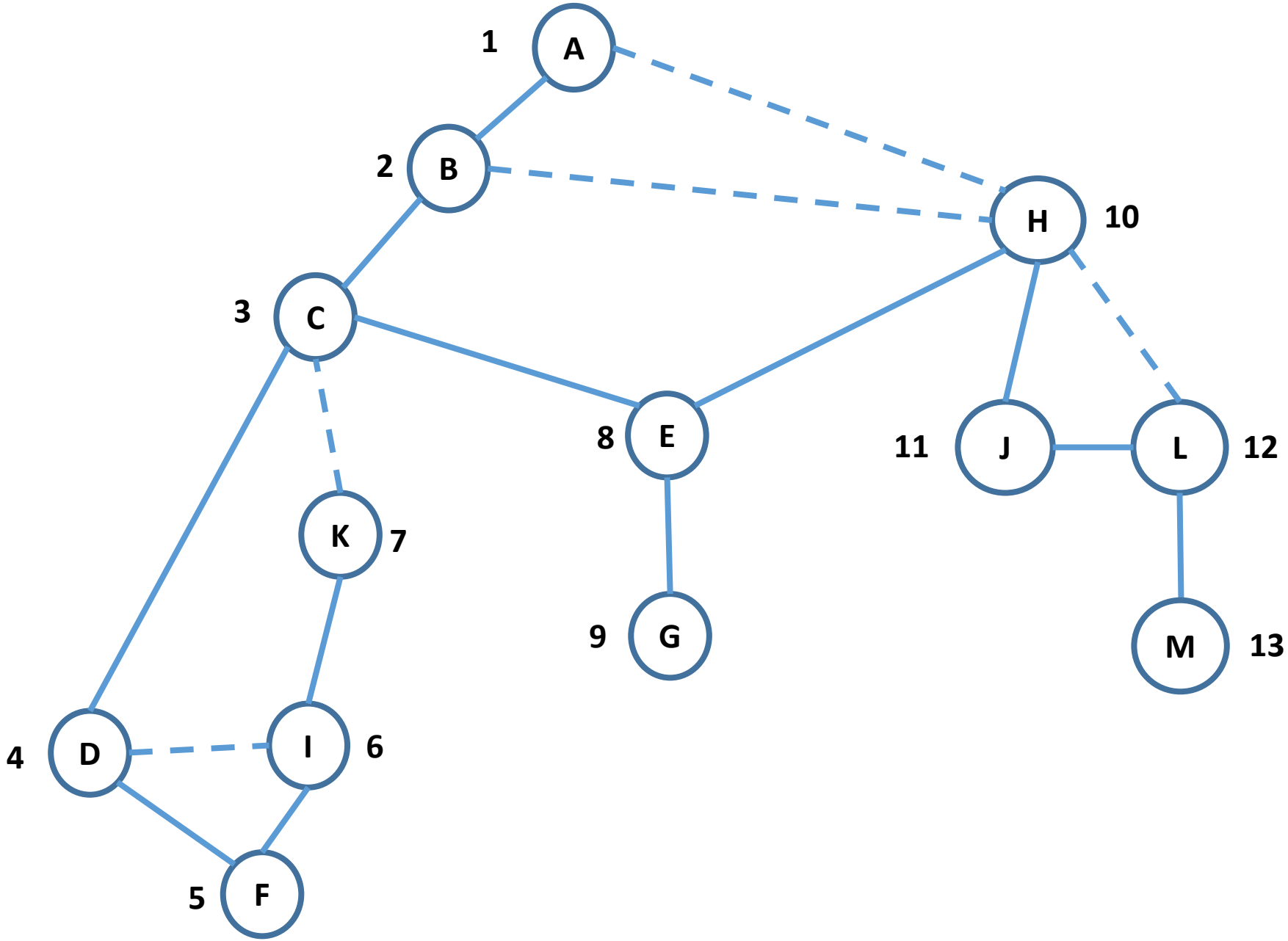
{C, E, H, L}

Set of cut edges

{EG, LM}



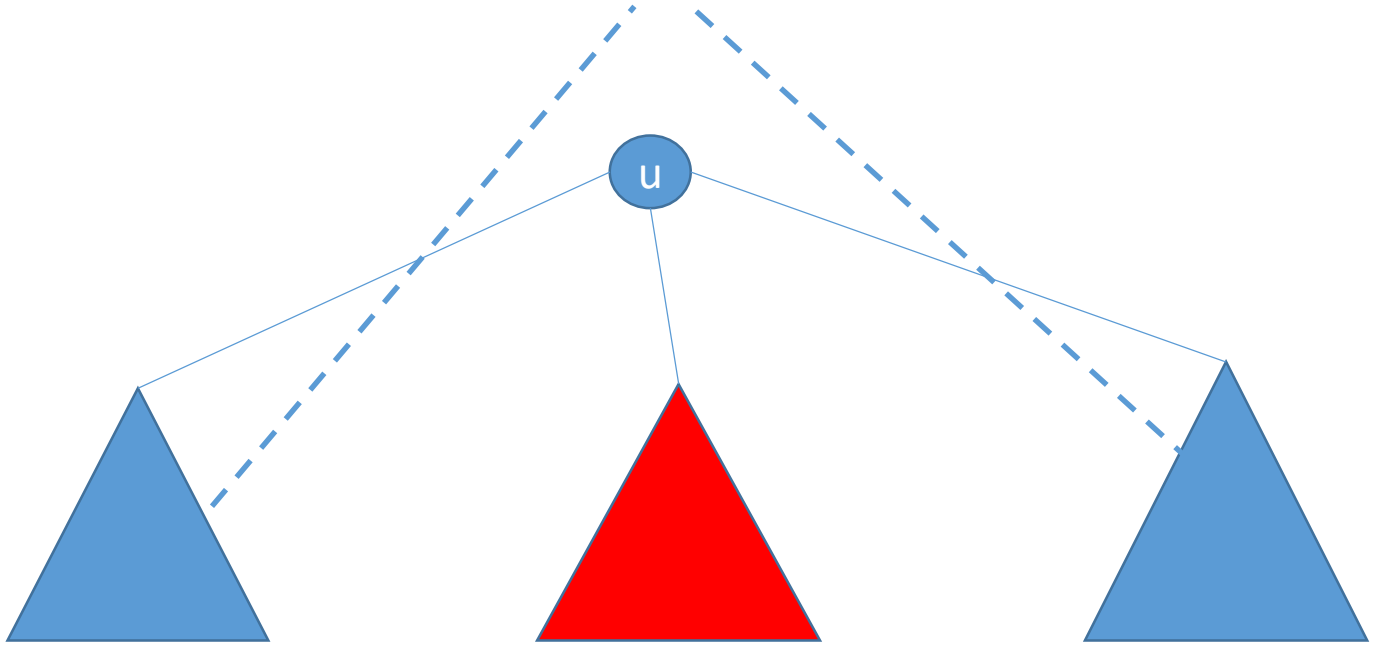
DFS:



Articulation Points from DFS Spanning Tree

- Root node in the DFS tree is an articulation point iff it has more than one child in the DFS tree.
- Leaf node in the DFS tree can not be an articulation point.
- A non-leaf and non-root node u is an articulation point iff no back edge goes above u from at least one sub-tree below some child of u

The vertex u is a cut vertex

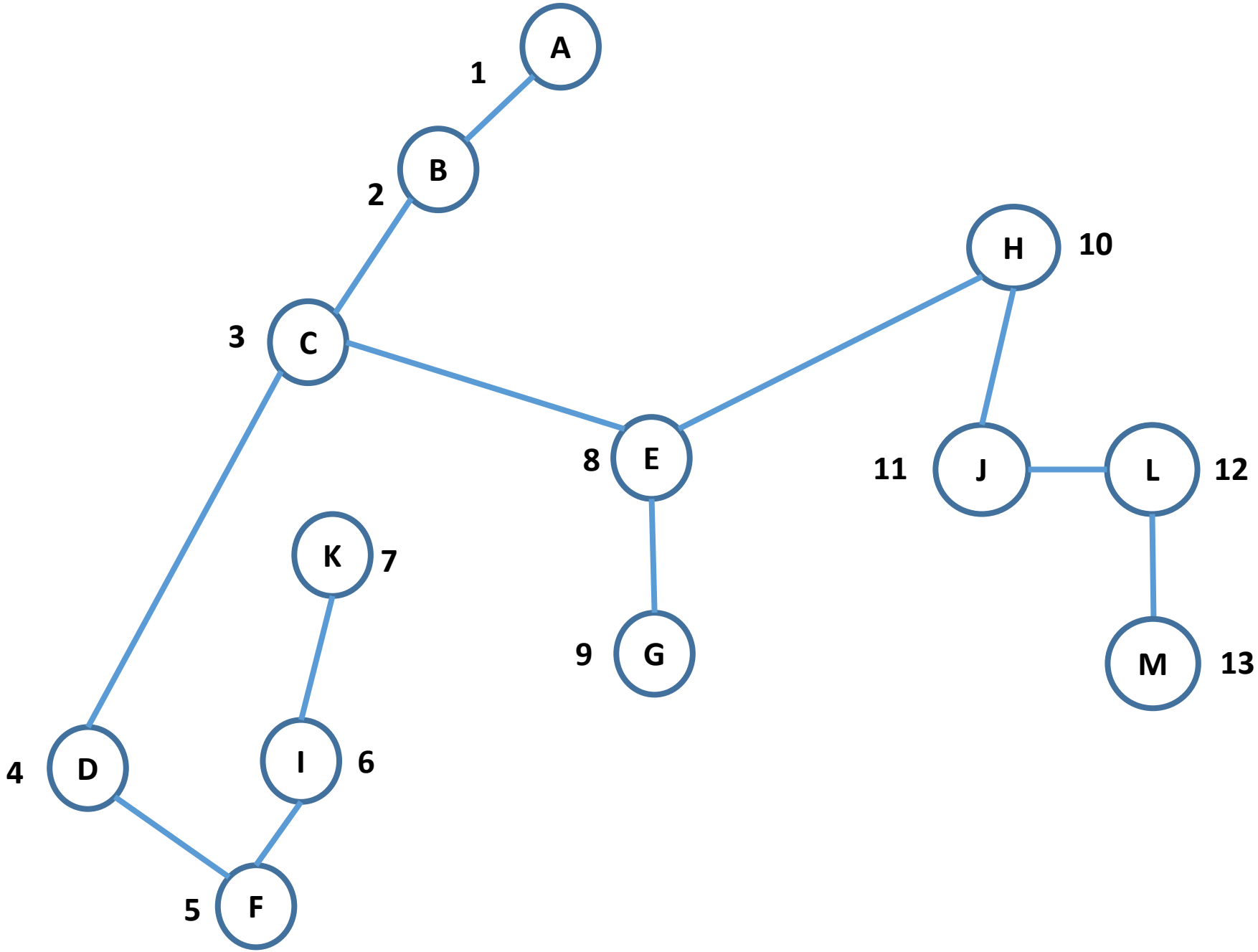


No back edge goes above

u from the red sub-tree

below some child of u

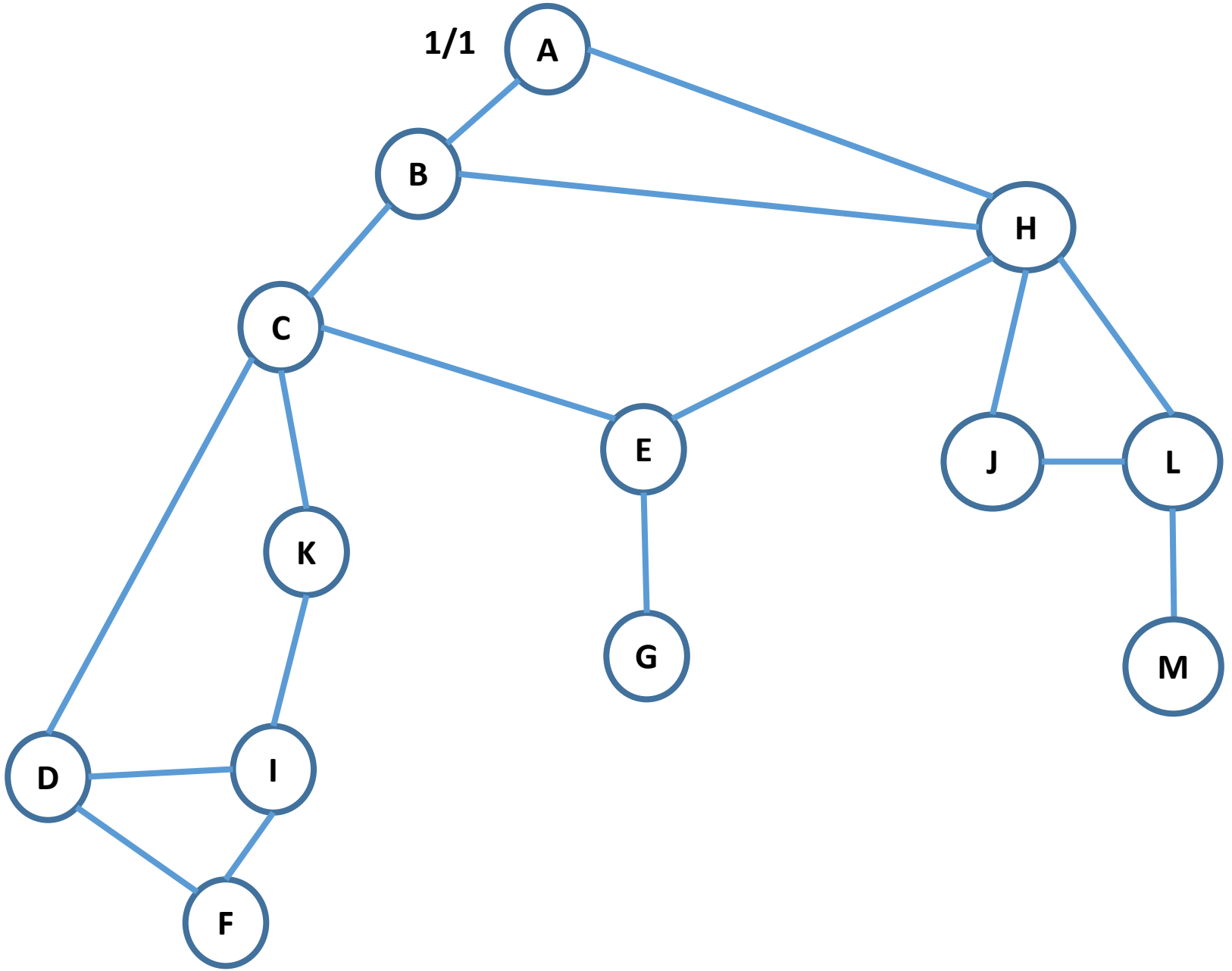
DFS:

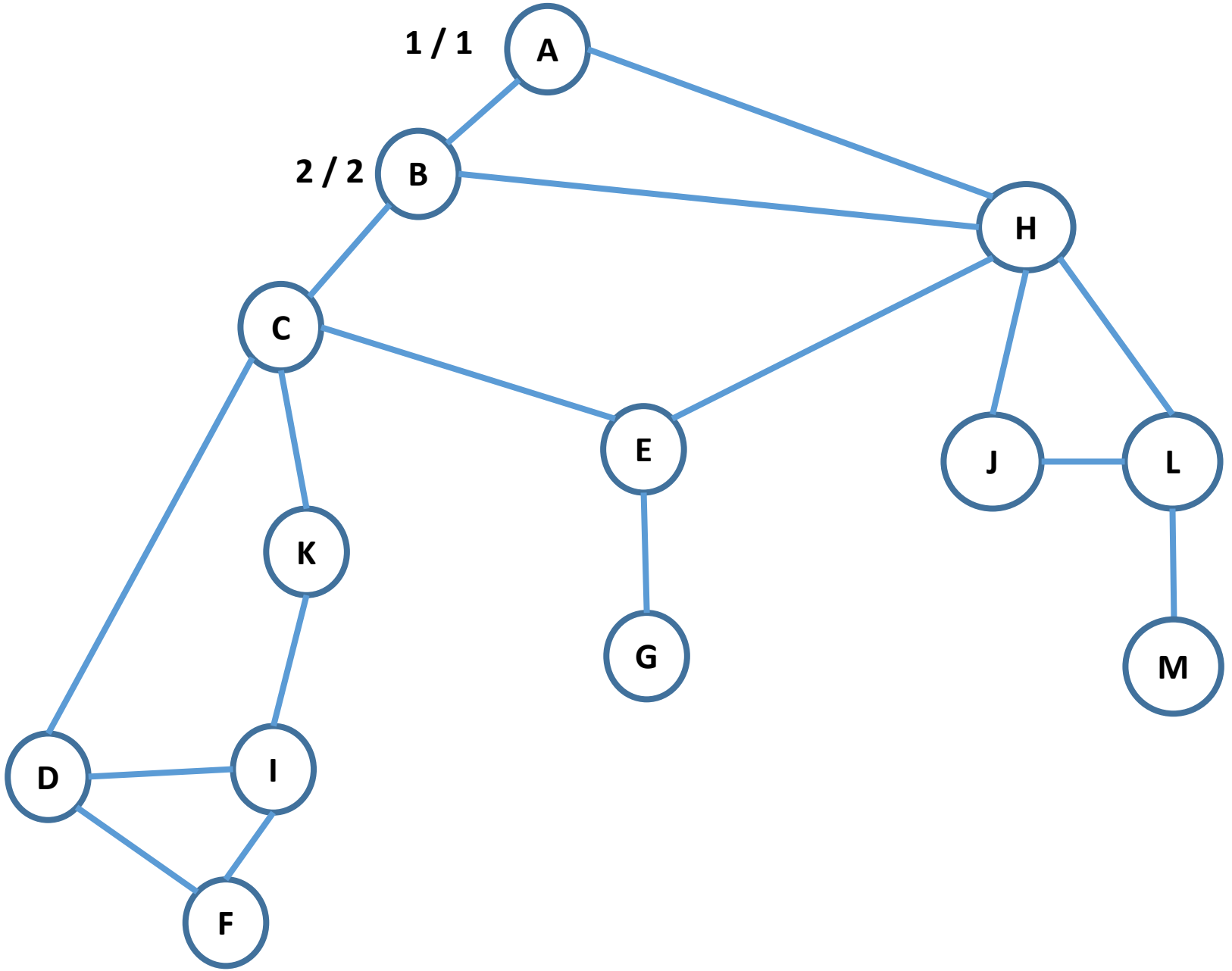


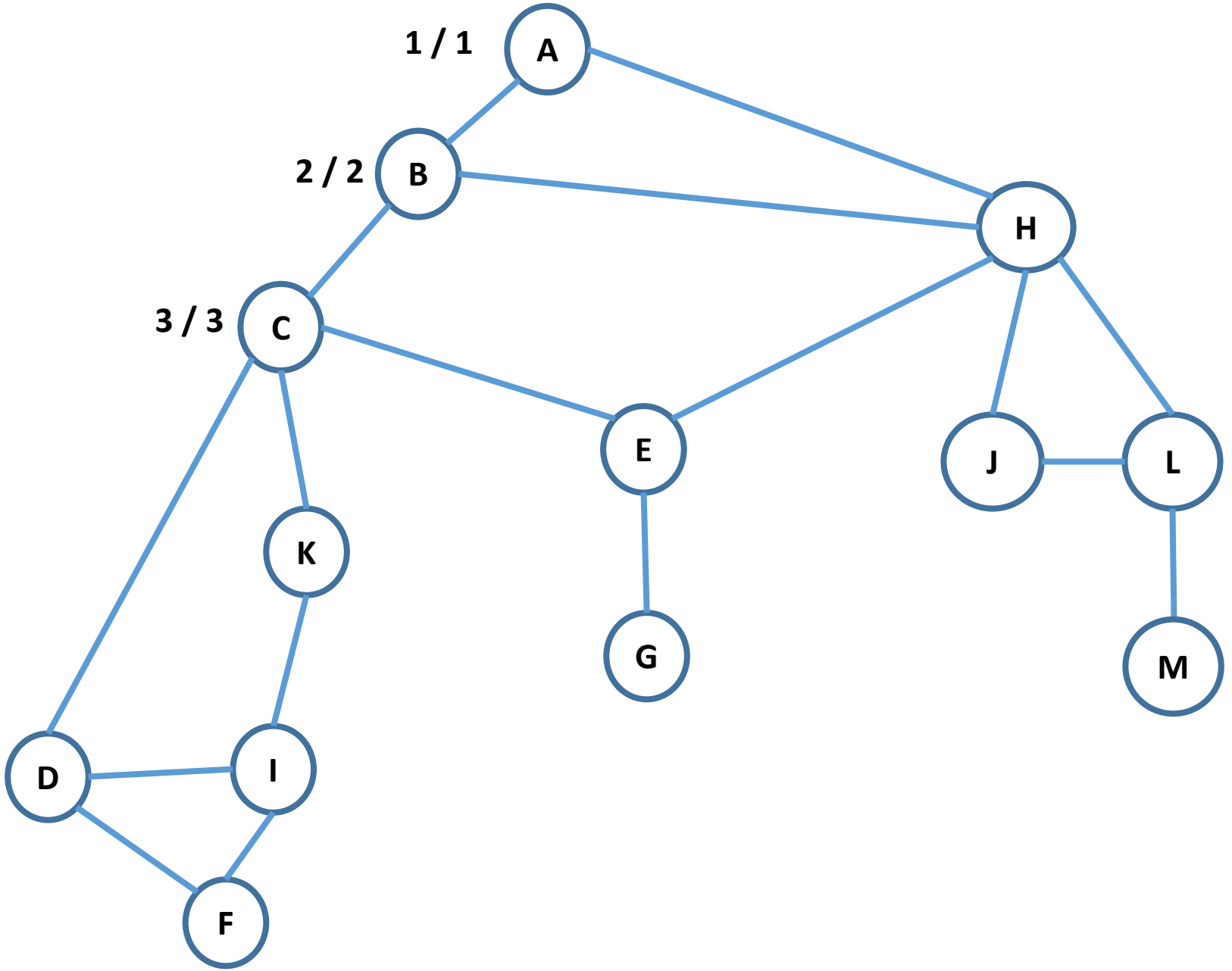
Articulation Points: the “low" function

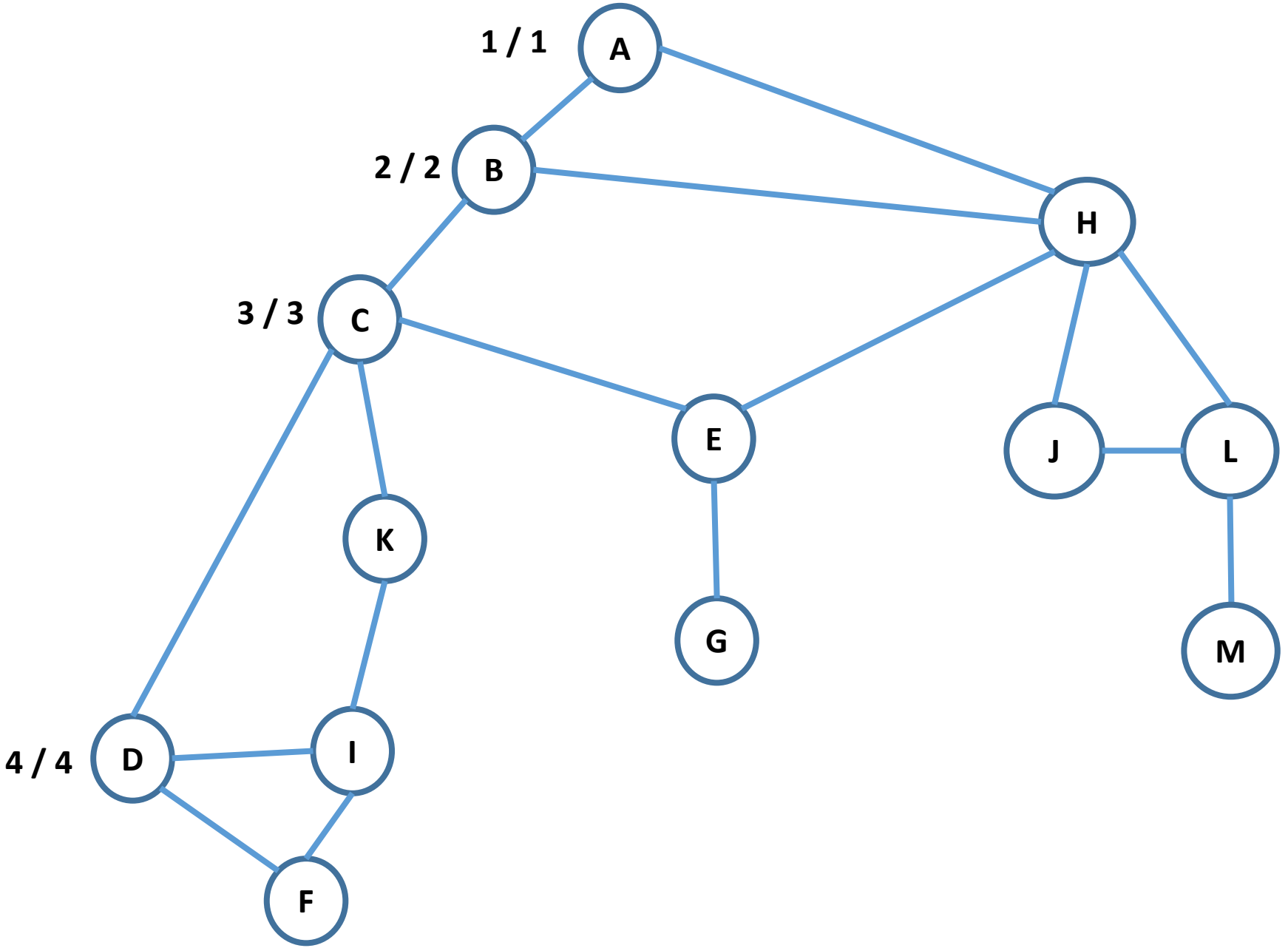
- low(u) is the minimum disc of any vertex that is either in the dfs subtree rooted at u (including u itself) or connected to a vertex above by a back edge from a subtree below some child of u.
- Key idea 1: if some child x of u has low(x) ≥ disc(u) then u is an articulation point.
- Key idea 2: low(u) = min ({low(w) | w a child of u } ∪ { disc (x) | {u, x} is a back edge from u })

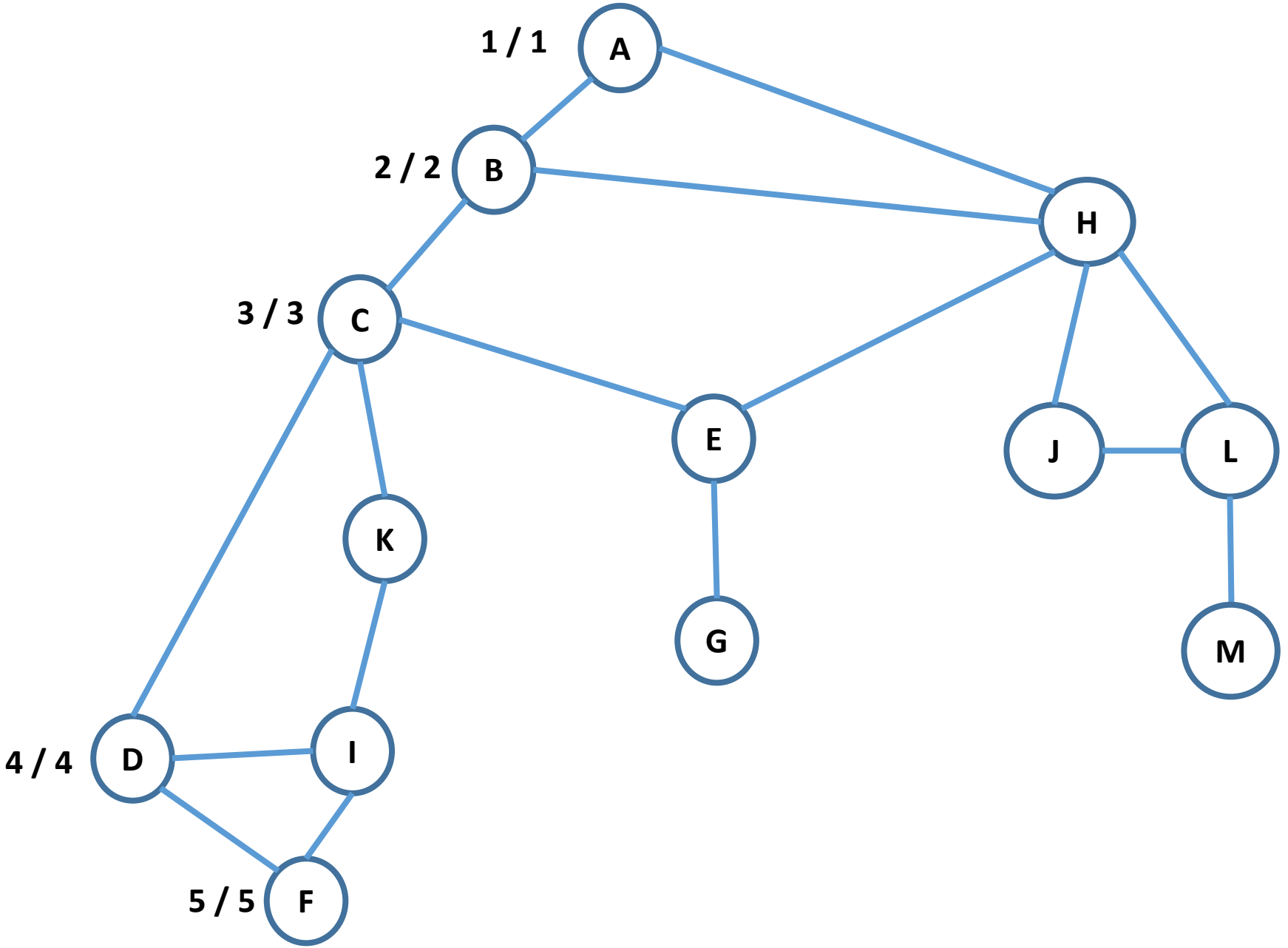
Tarjan’s Algorithm

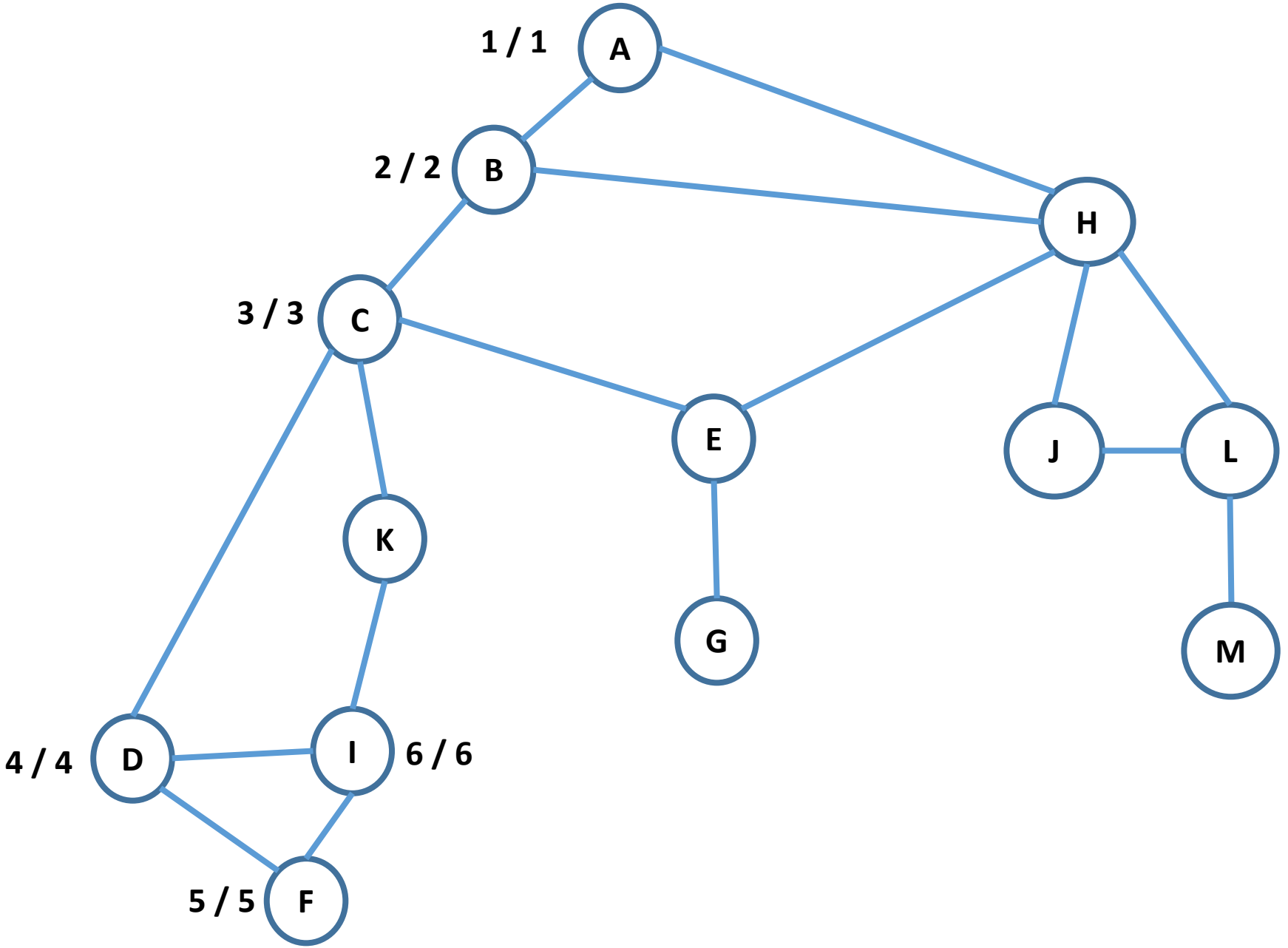

1/1
A
B
H
C
E
J
L
K
G
M
D
I
F

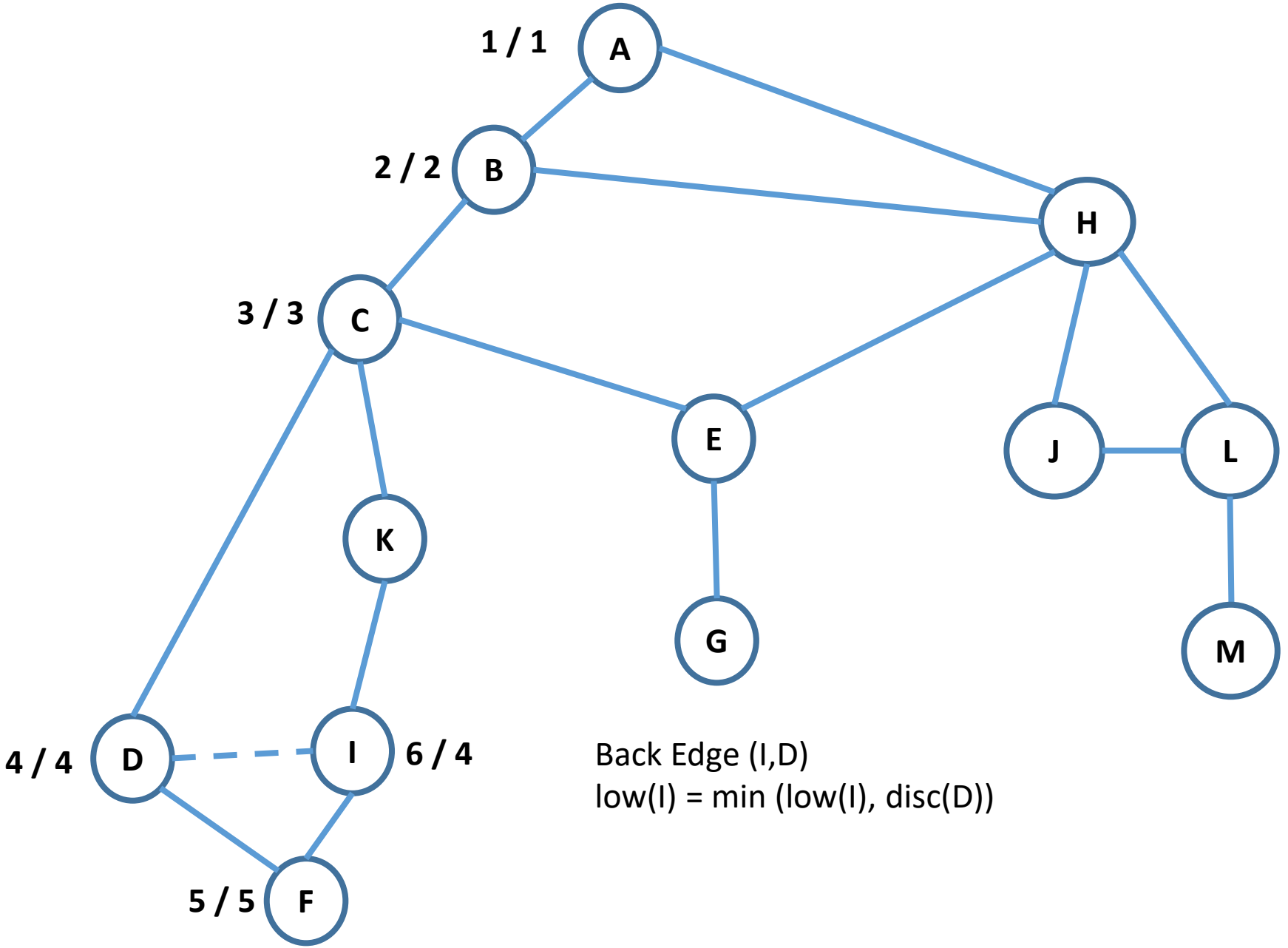

1 / 1
A
2 / 2
B
H
C
E
J
L
K
G
M
D
I
F


1 / 1
A
2 / 2
B
H
3 / 3
C
E
J
L
K
G
M
D
I
F

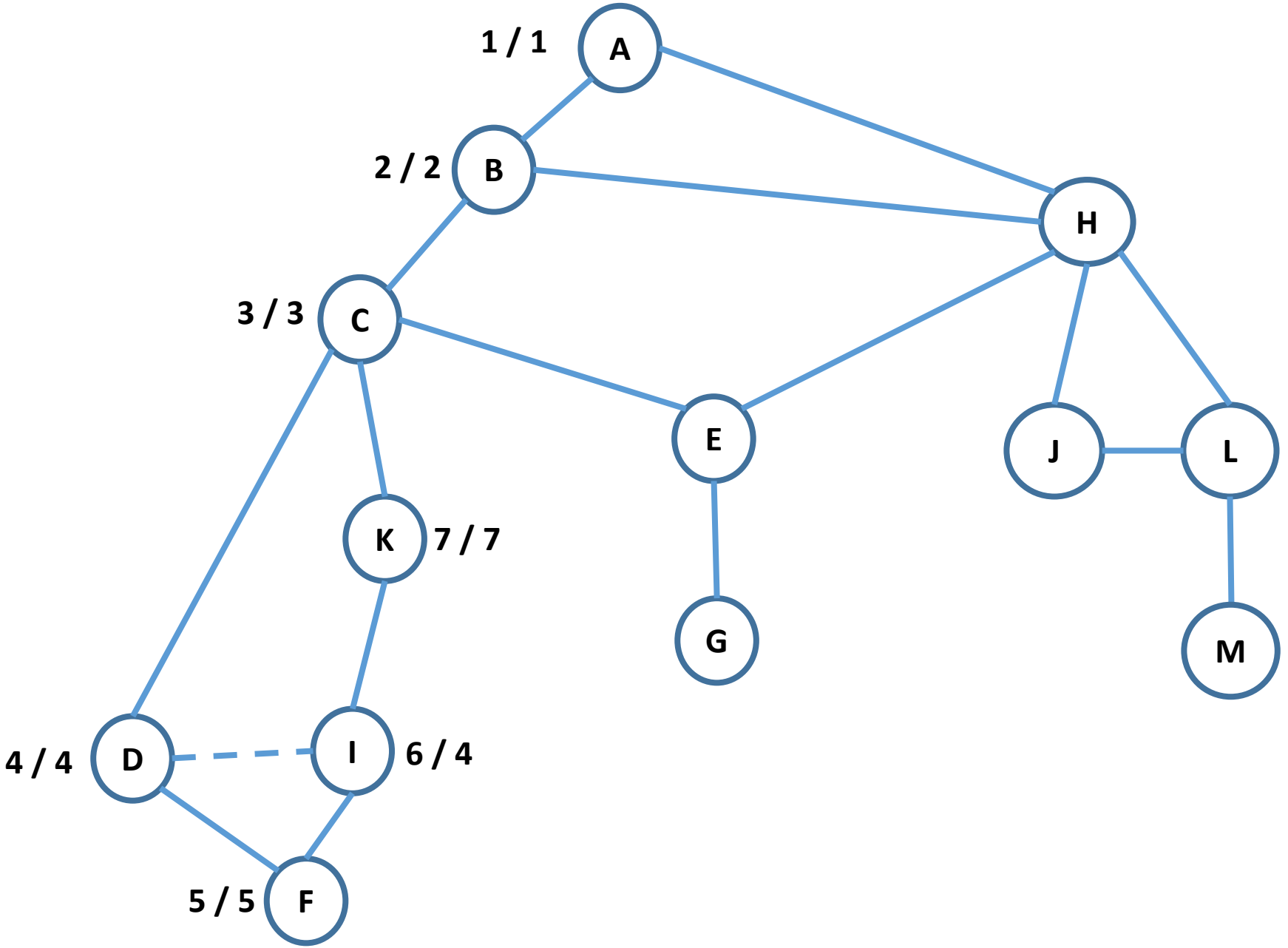

1 / 1
A
2 / 2
B
H
3 / 3
C
E
J
L
K
G
M
4 / 4
D
I
F

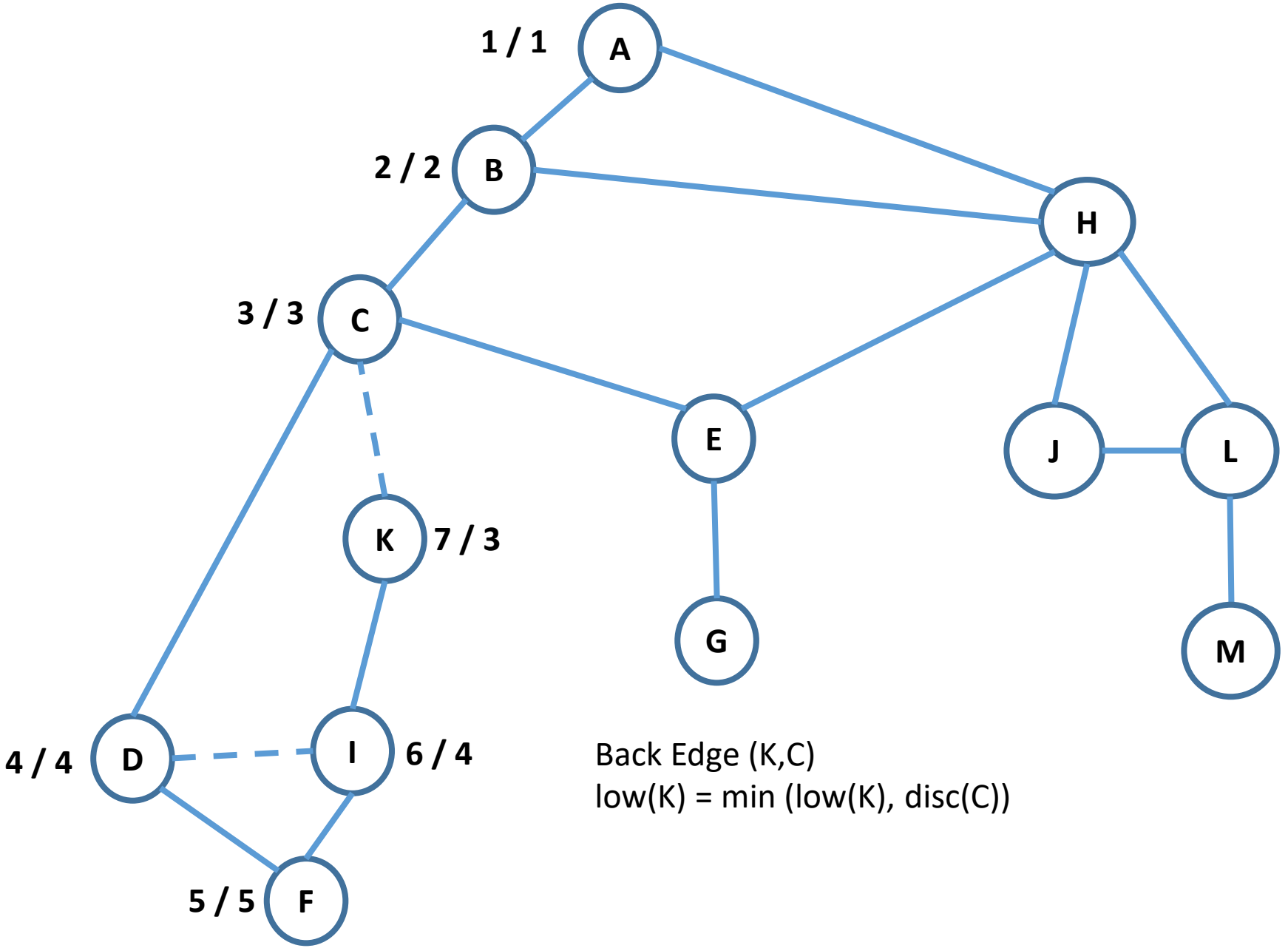

1 / 1
A
2 / 2
B
H
3 / 3
C
E
J
L
K
G
M
4 / 4
D
I
5 / 5
F


1 / 1
A
2 / 2
B
H
3 / 3
C
E
J
L
K
G
M
4 / 4
D
I
6 / 6
5 / 5
F

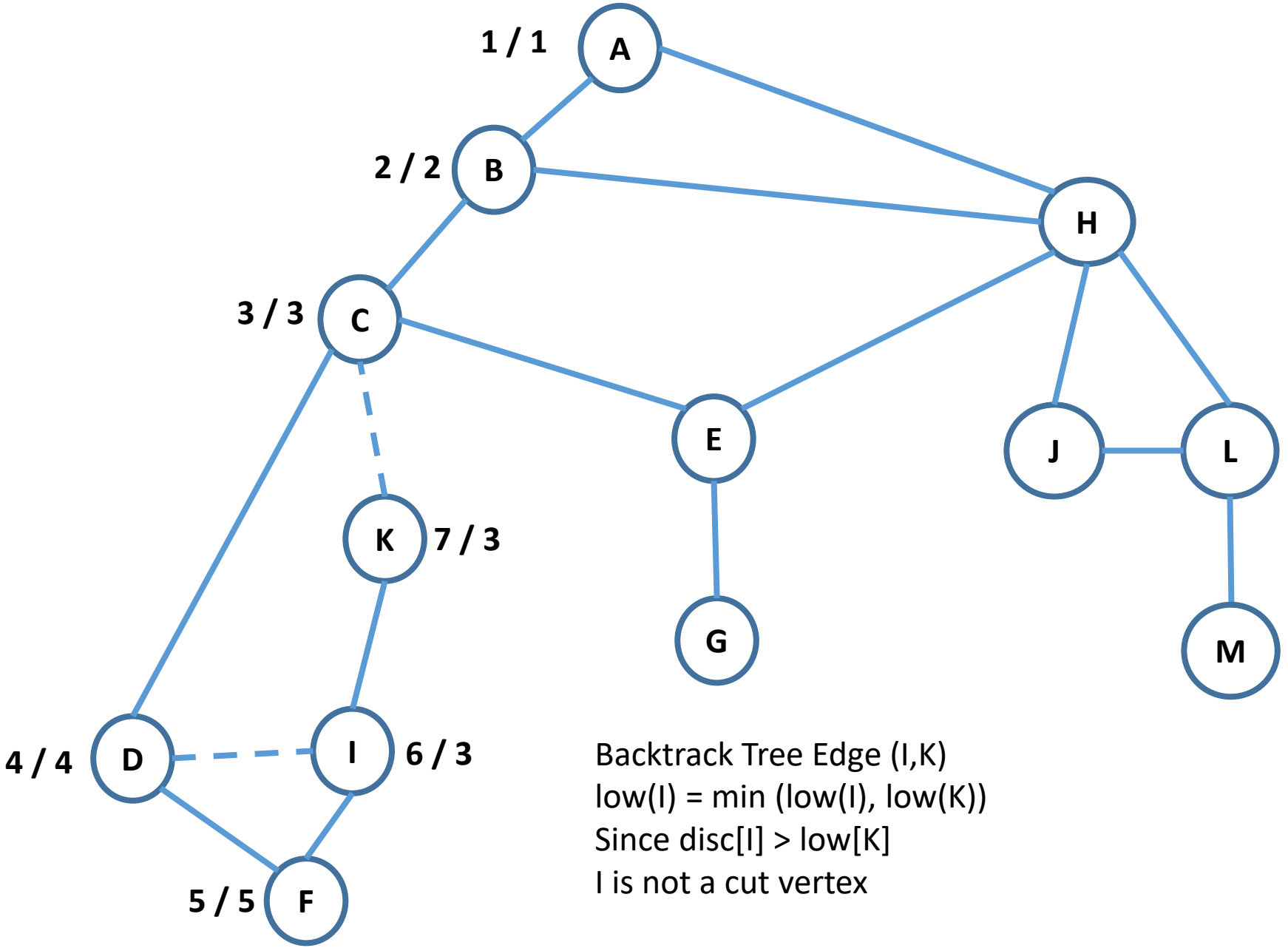


Back Edge (I,D)
low(I) = min (low(I), disc(D))

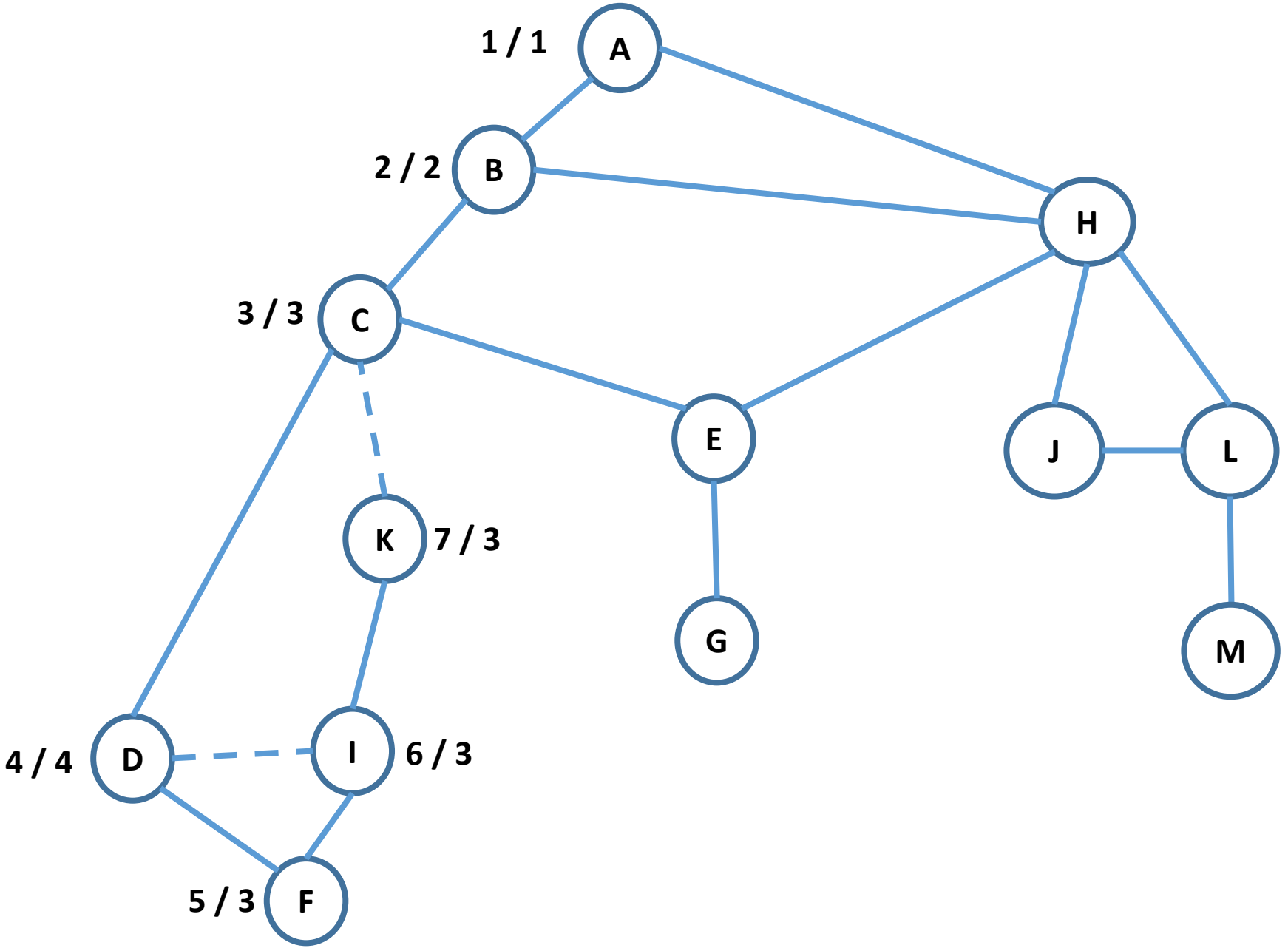

1 / 1
A
2 / 2
B
H
3 / 3
C
E
J
L
K
7 / 7
G
M
4 / 4
D
I
6 / 4
5 / 5
F

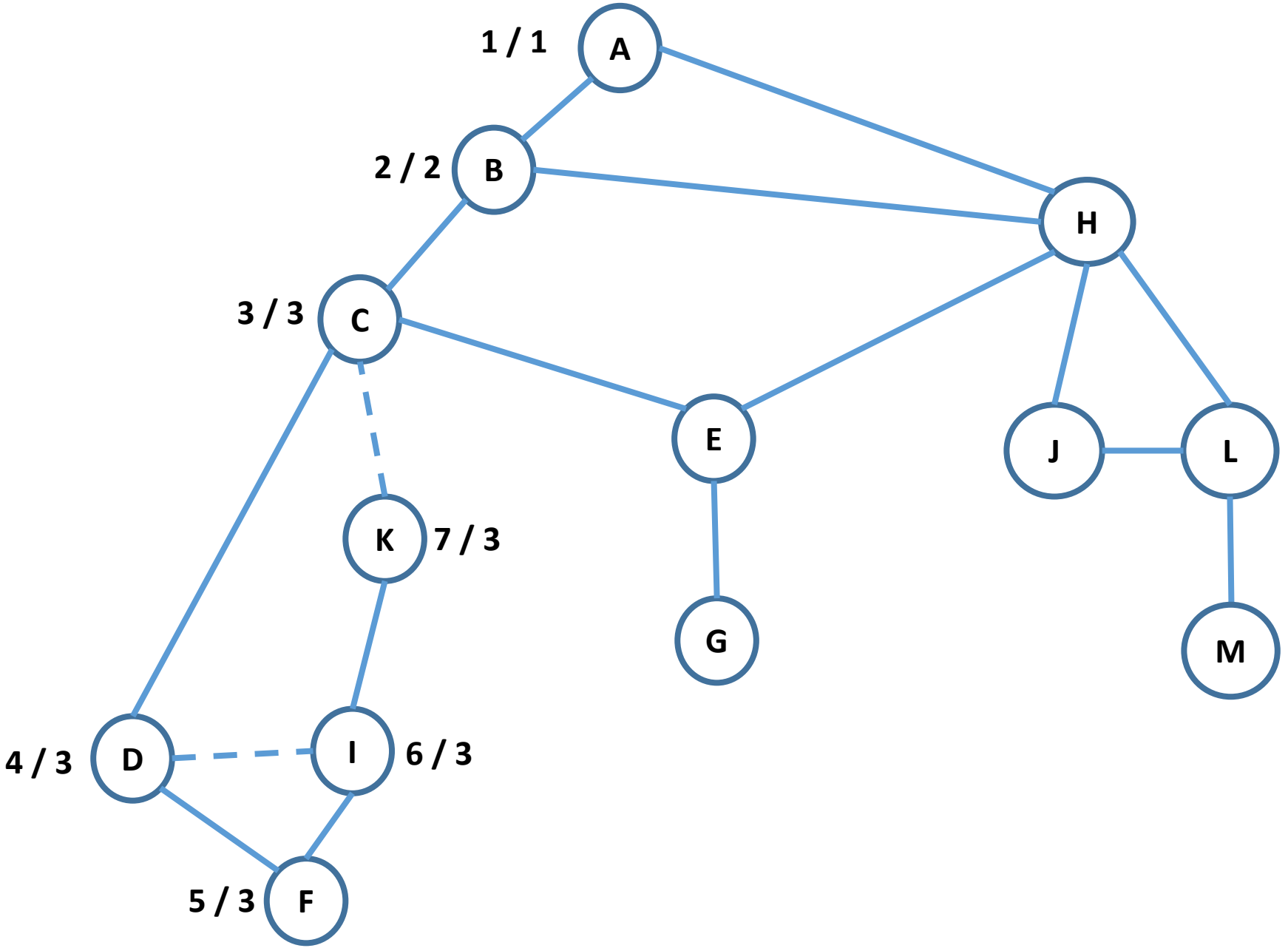


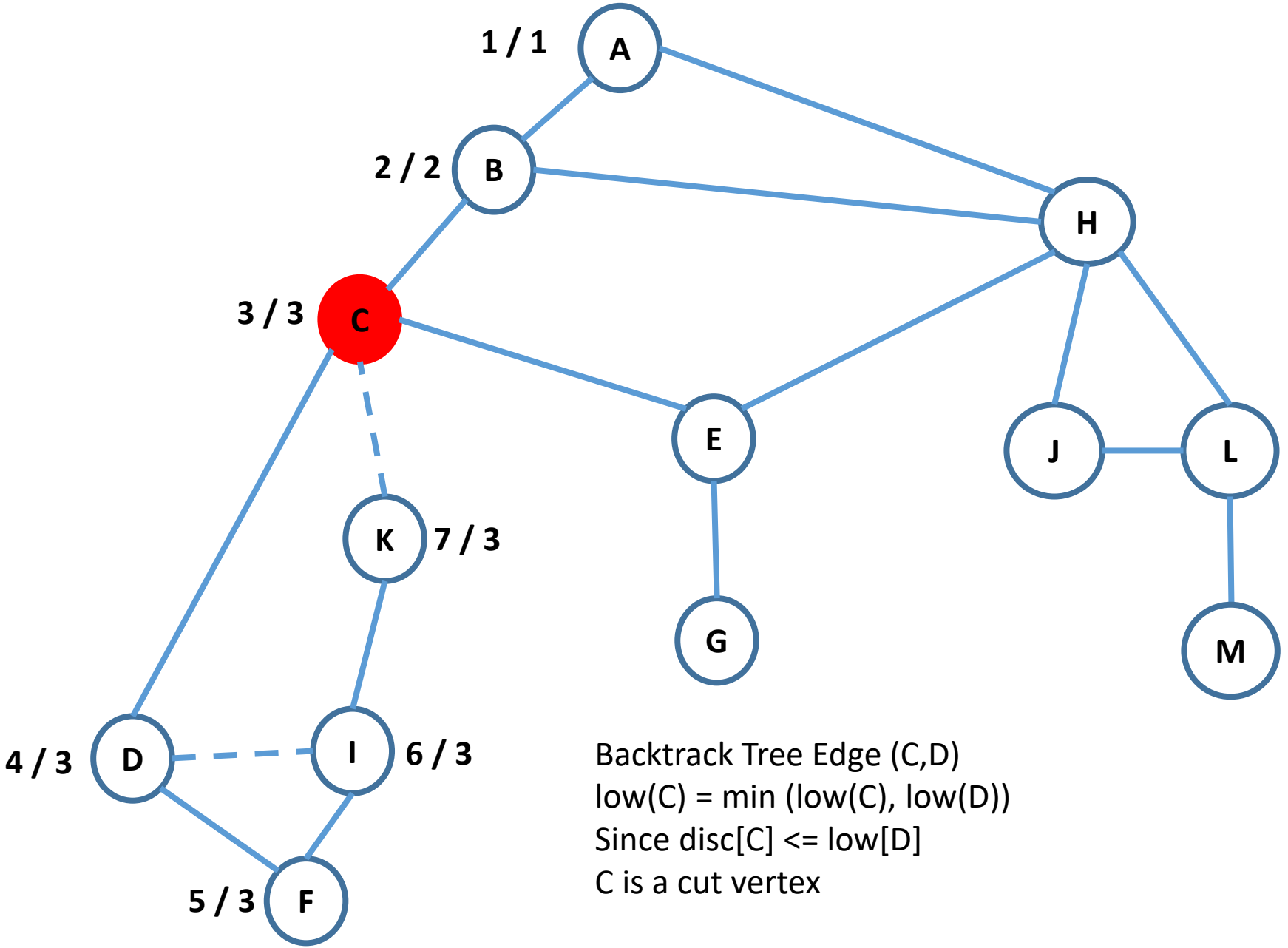
Back Edge (K,C)
low(K) = min (low(K), disc(C))



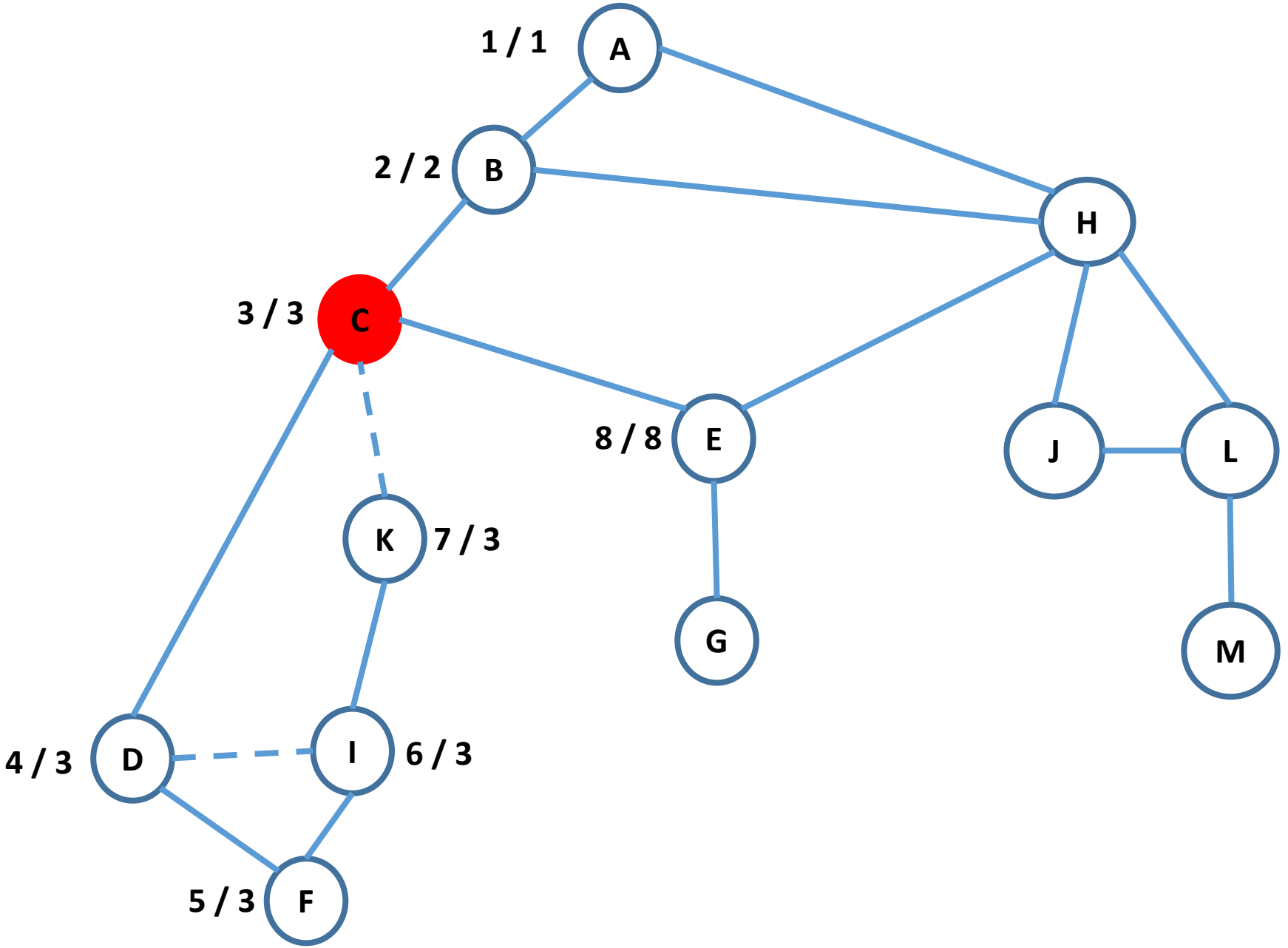
Backtrack Tree Edge (I,K)
low(I) = min (low(I), low(K))
Since disc[I] > low[K]
I is not a cut vertex

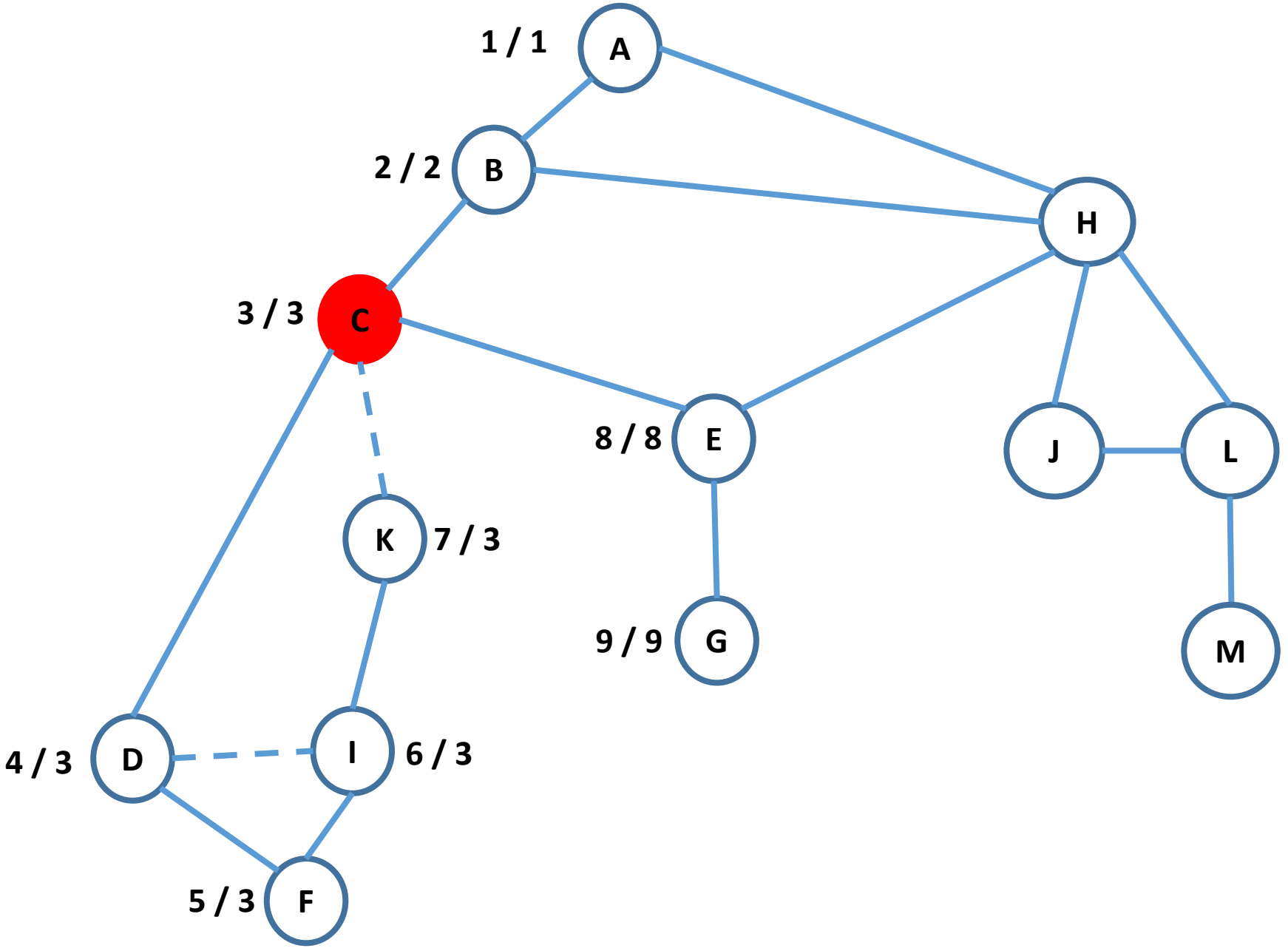

1 / 1
A
2 / 2
B
H
3 / 3
C
E
J
L
K
7 / 3
G
M
4 / 4
D
I
6 / 3
5 / 3
F

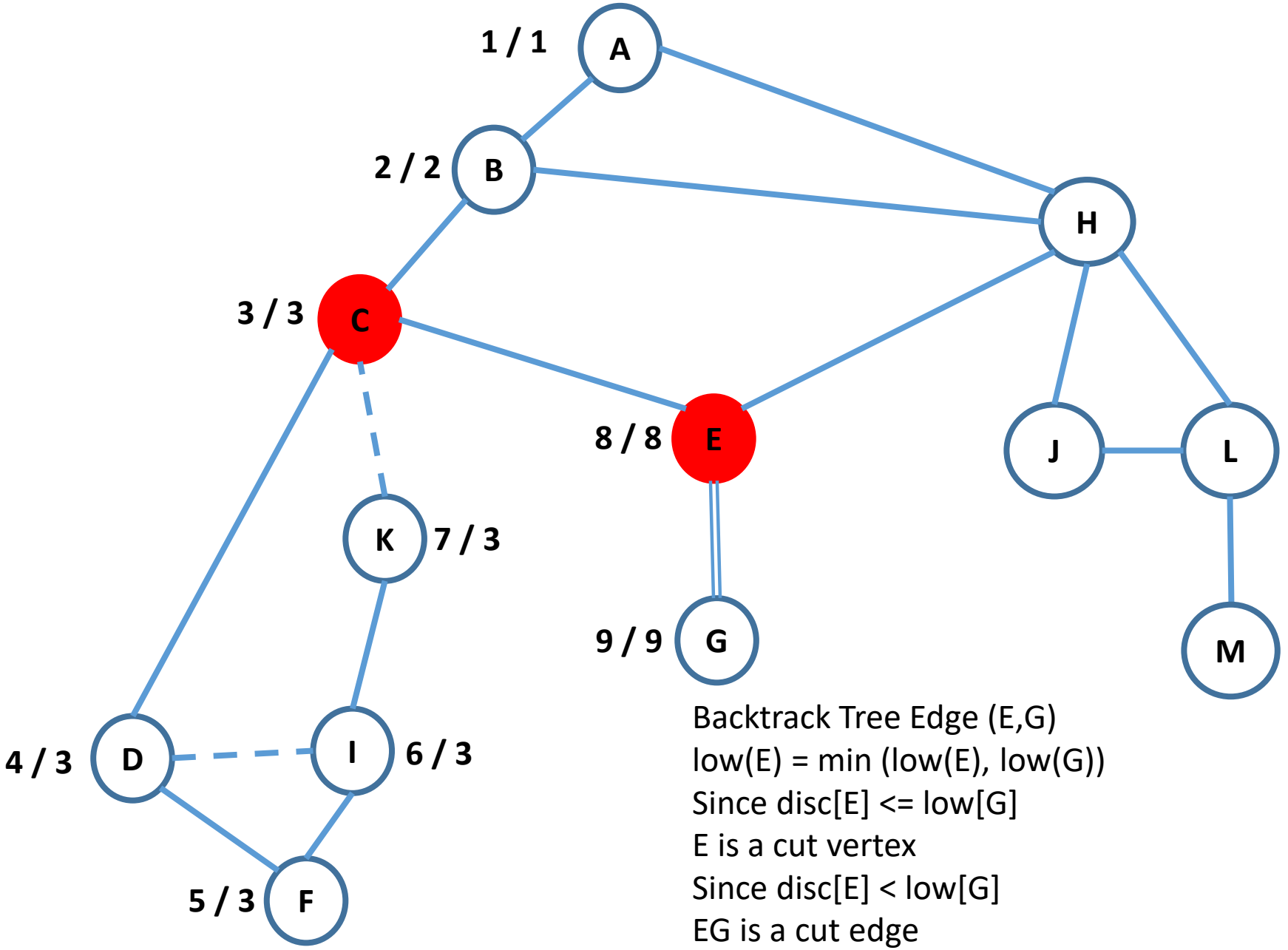

1 / 1
A
2 / 2
B
H
3 / 3
C
E
J
L
K
7 / 3
G
M
4 / 3
D
I
6 / 3
5 / 3
F



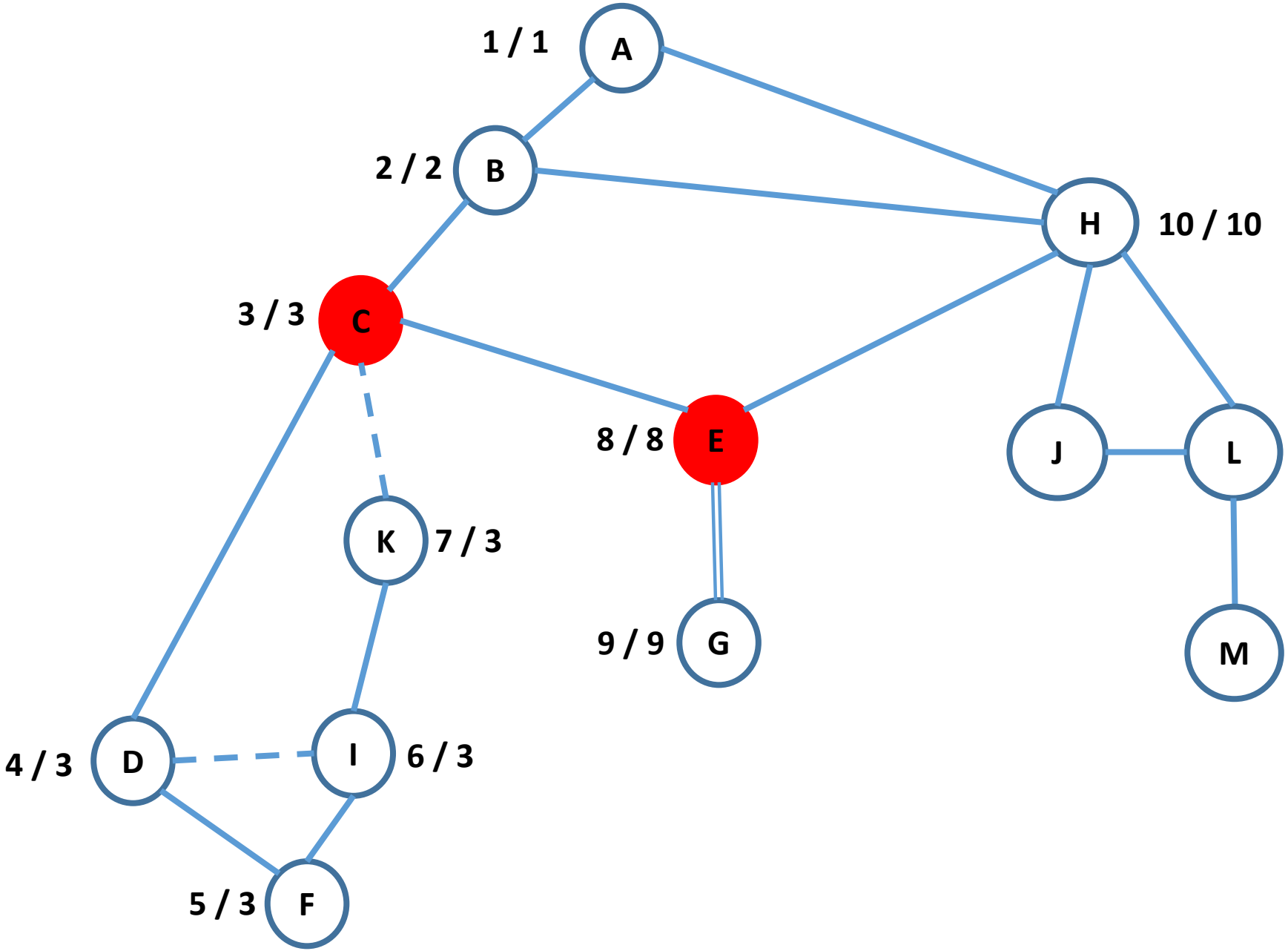
Backtrack Tree Edge (C,D)
low(C) = min (low(C), low(D))
Since disc[C] <= low[D]
C is a cut vertex

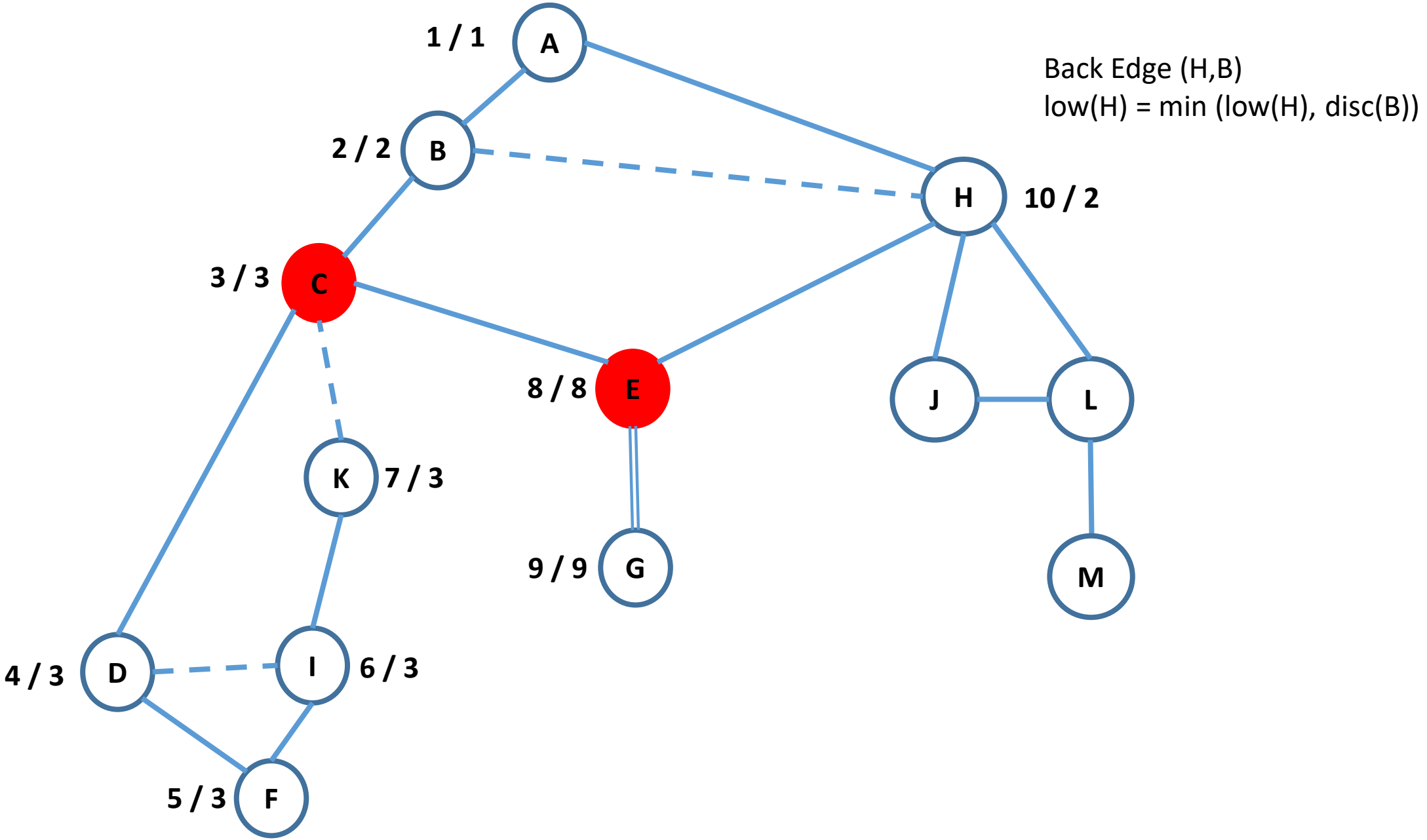

1 / 1
A
2 / 2
B
H
3 / 3
C
8 / 8
E
J
L
K
7 / 3
G
M
4 / 3
D
I
6 / 3
5 / 3
F


1 / 1
A
2 / 2
B
H
3 / 3
C
8 / 8
E
J
L
K
7 / 3
9 / 9
G
M
4 / 3
D
I
6 / 3
5 / 3
F

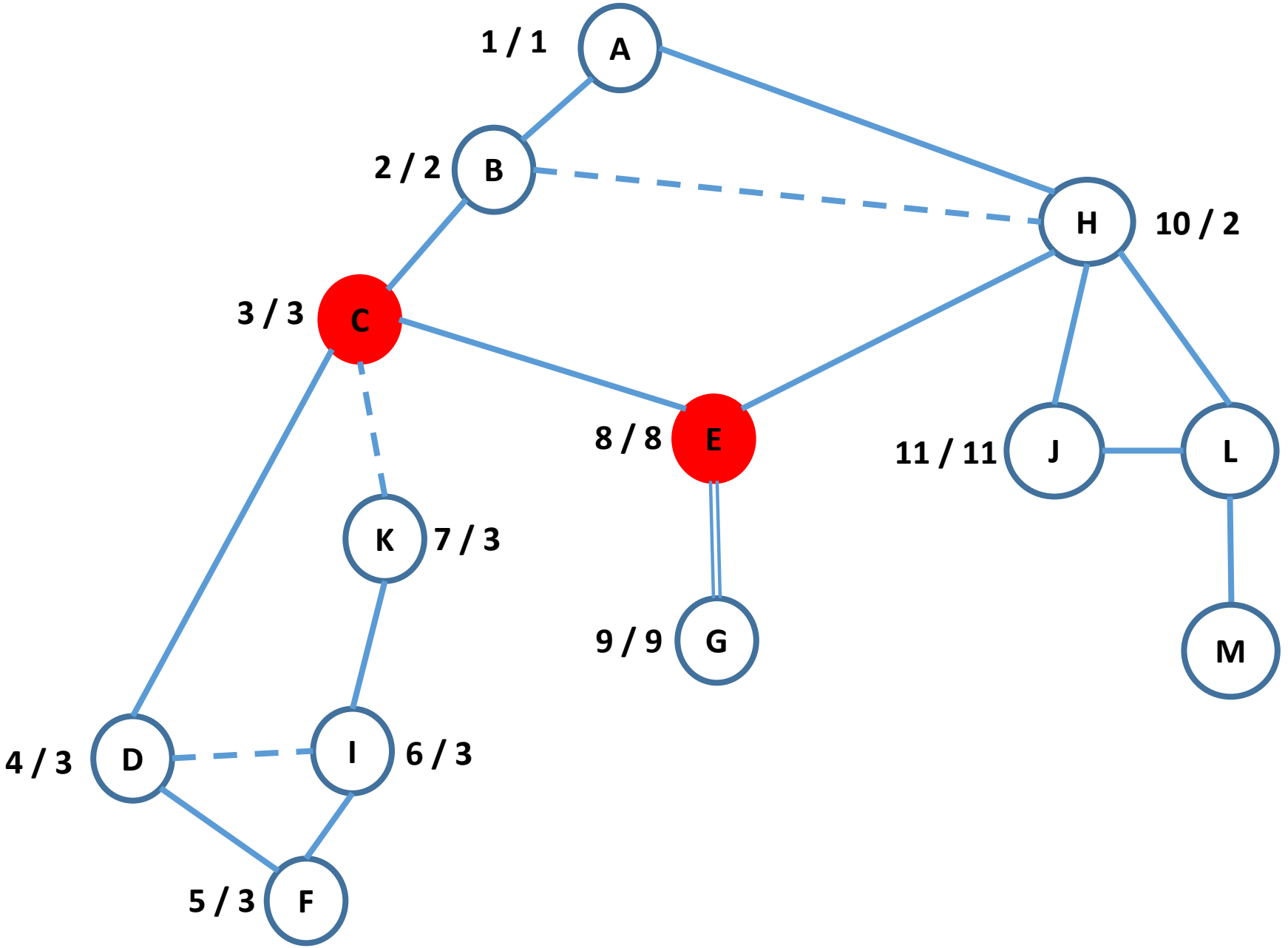


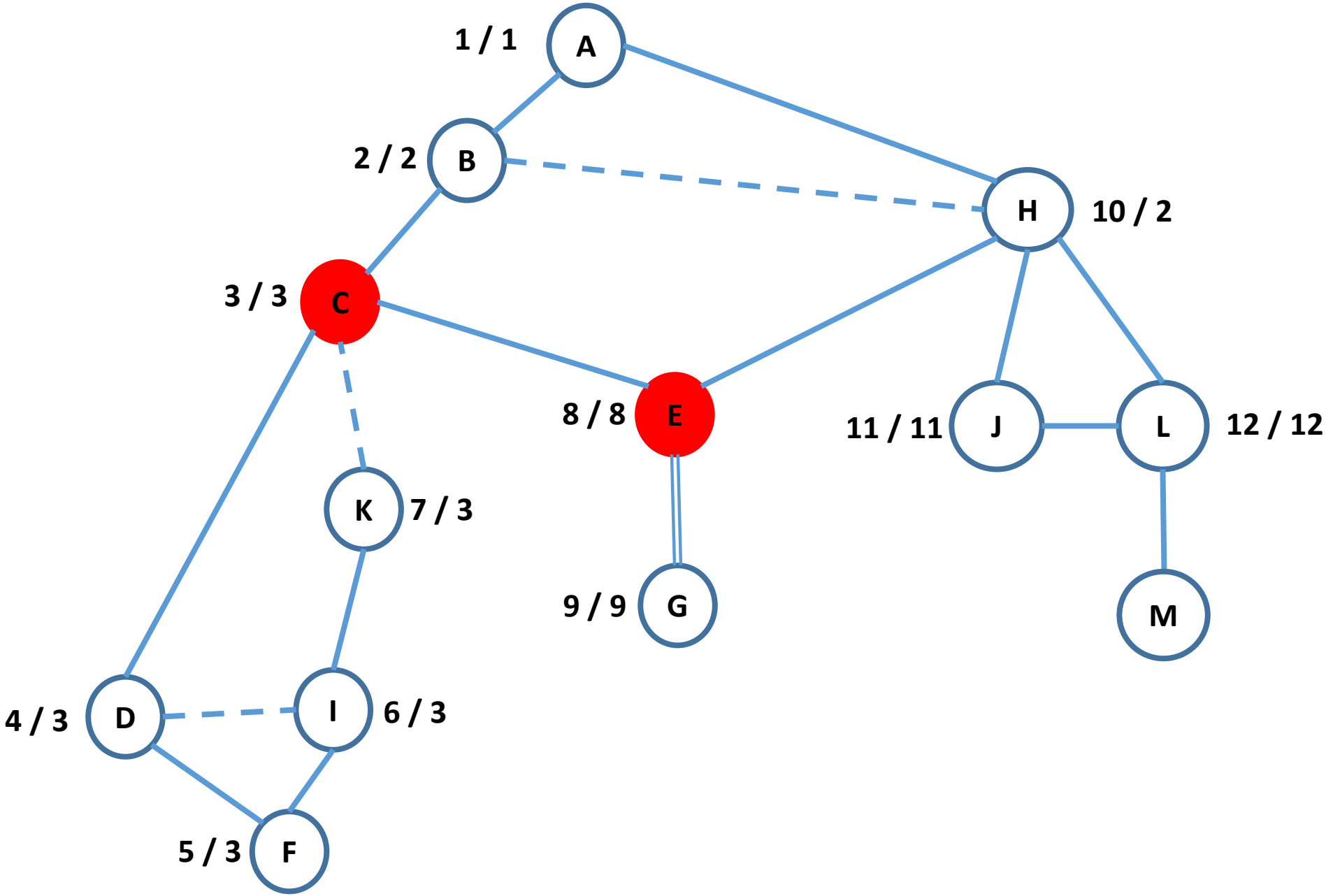
Backtrack Tree Edge (E,G)
low(E) = min (low(E), low(G))
Since disc[E] <= low[G]
E is a cut vertex
Since disc[E] < low[G]
EG is a cut edge

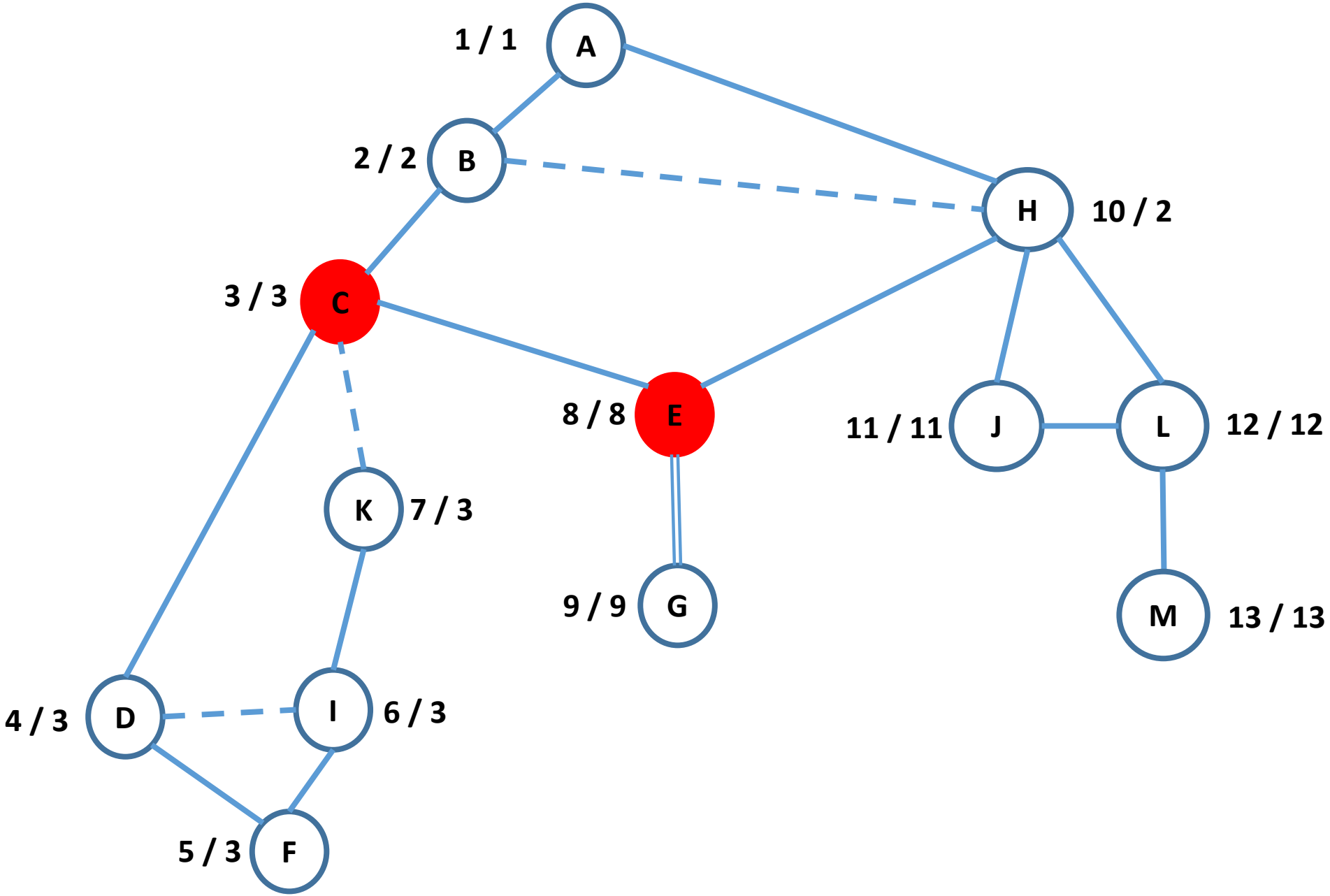

1 / 1
A
2 / 2
B
H
10 / 10
3 / 3
C
8 / 8
E
J
L
K
7 / 3
9 / 9
G
M
4 / 3
D
I
6 / 3
5 / 3
F

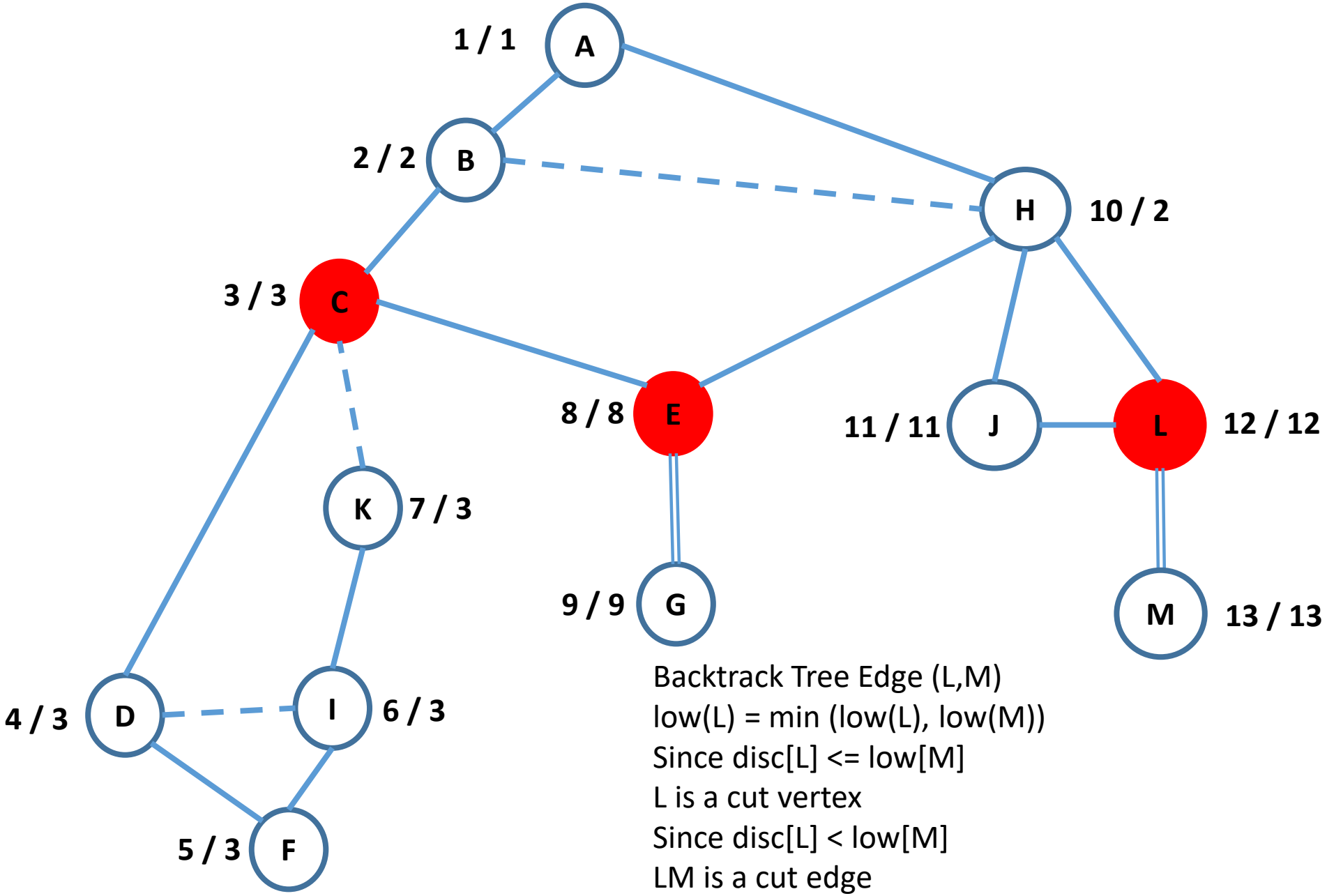


Back Edge (H,B)
low(H) = min (low(H), disc(B))

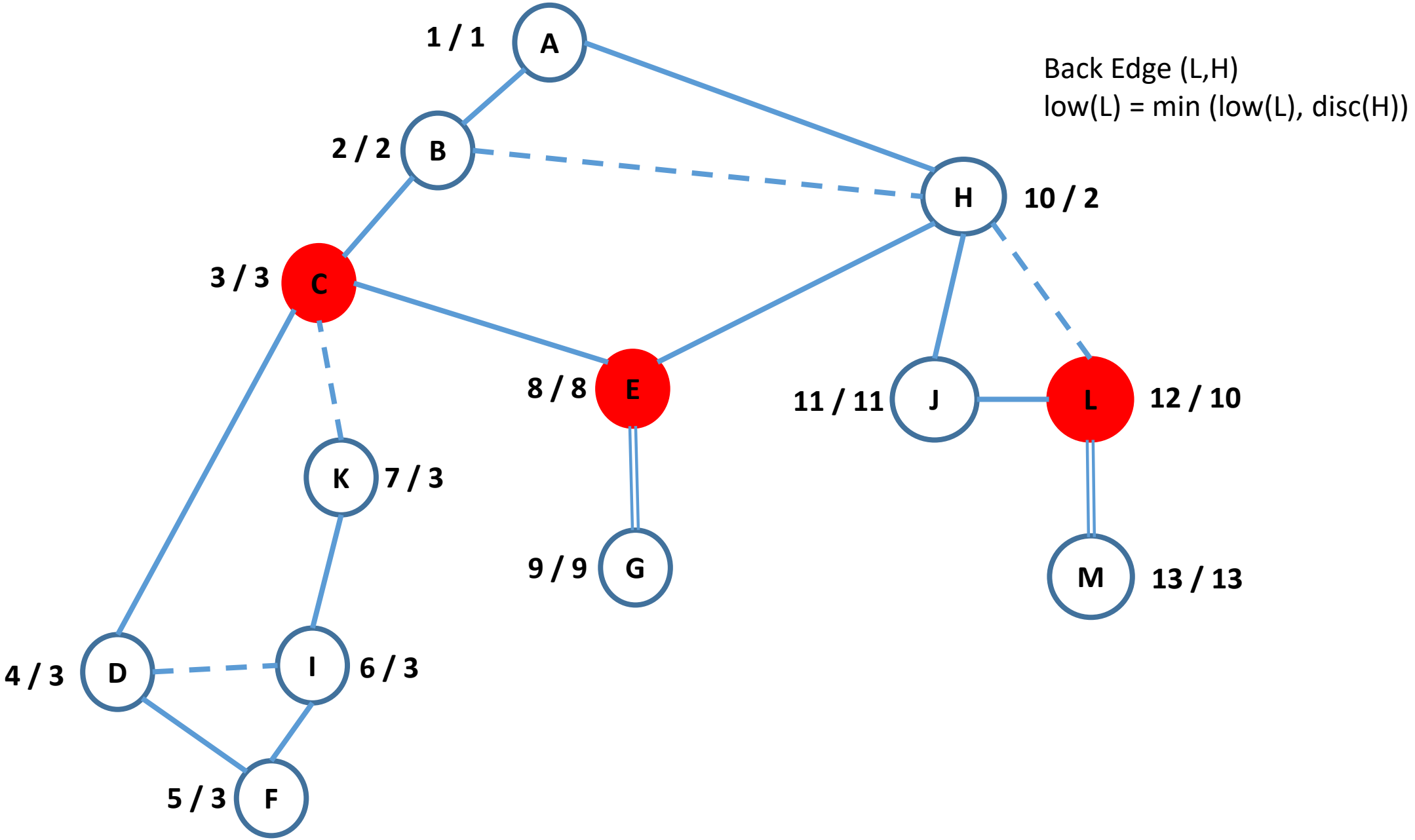

1 / 1
A
2 / 2
B
H
10 / 2
3 / 3
C
8 / 8
E
11 / 11
J
L
K
7 / 3
9 / 9
G
M
4 / 3
D
I
6 / 3
5 / 3
F

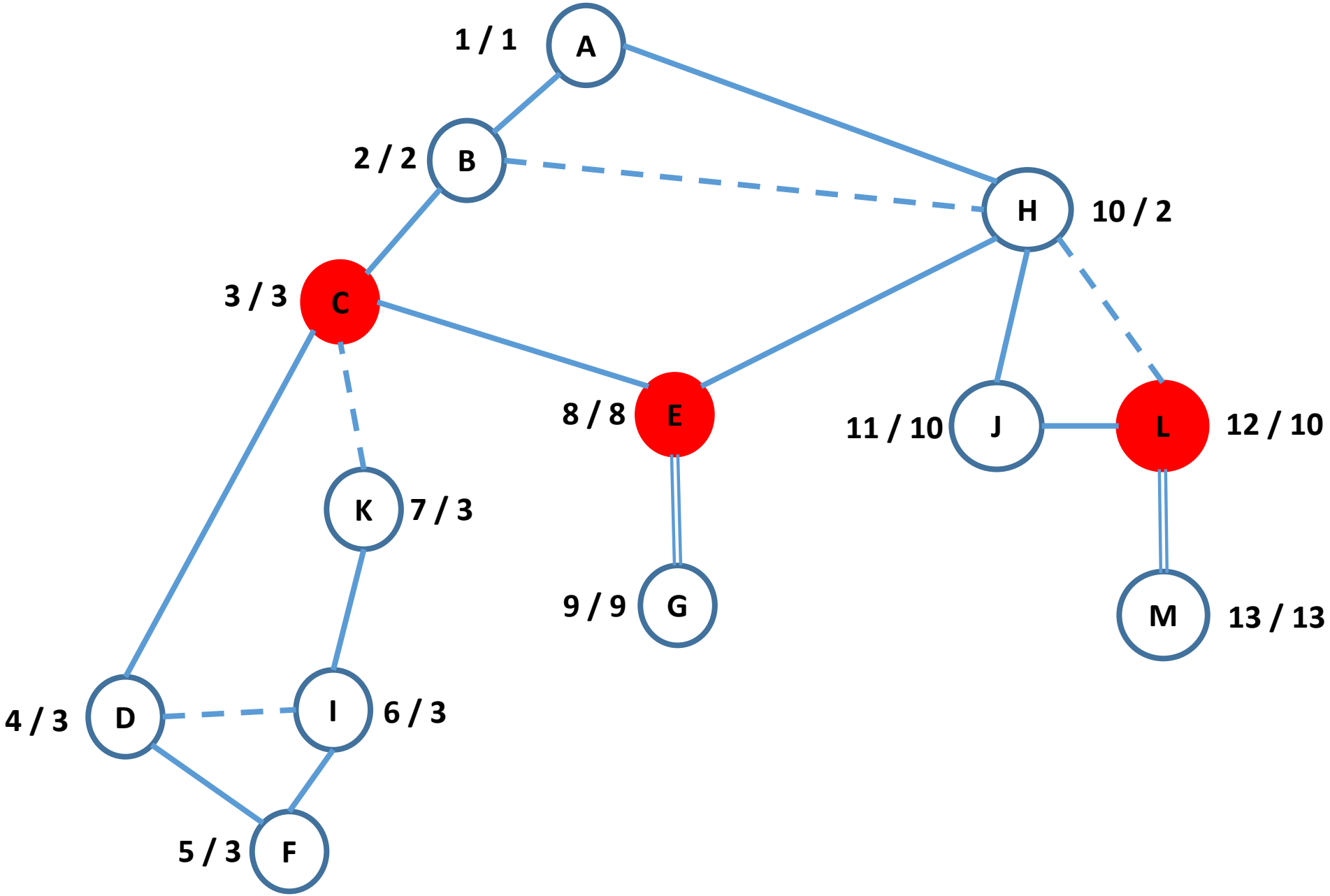

1 / 1
A
2 / 2
B
H
10 / 2
3 / 3
C
8 / 8
E
11 / 11
J
L
12 / 12
K
7 / 3
9 / 9
G
M
4 / 3
D
I
6 / 3
5 / 3
F

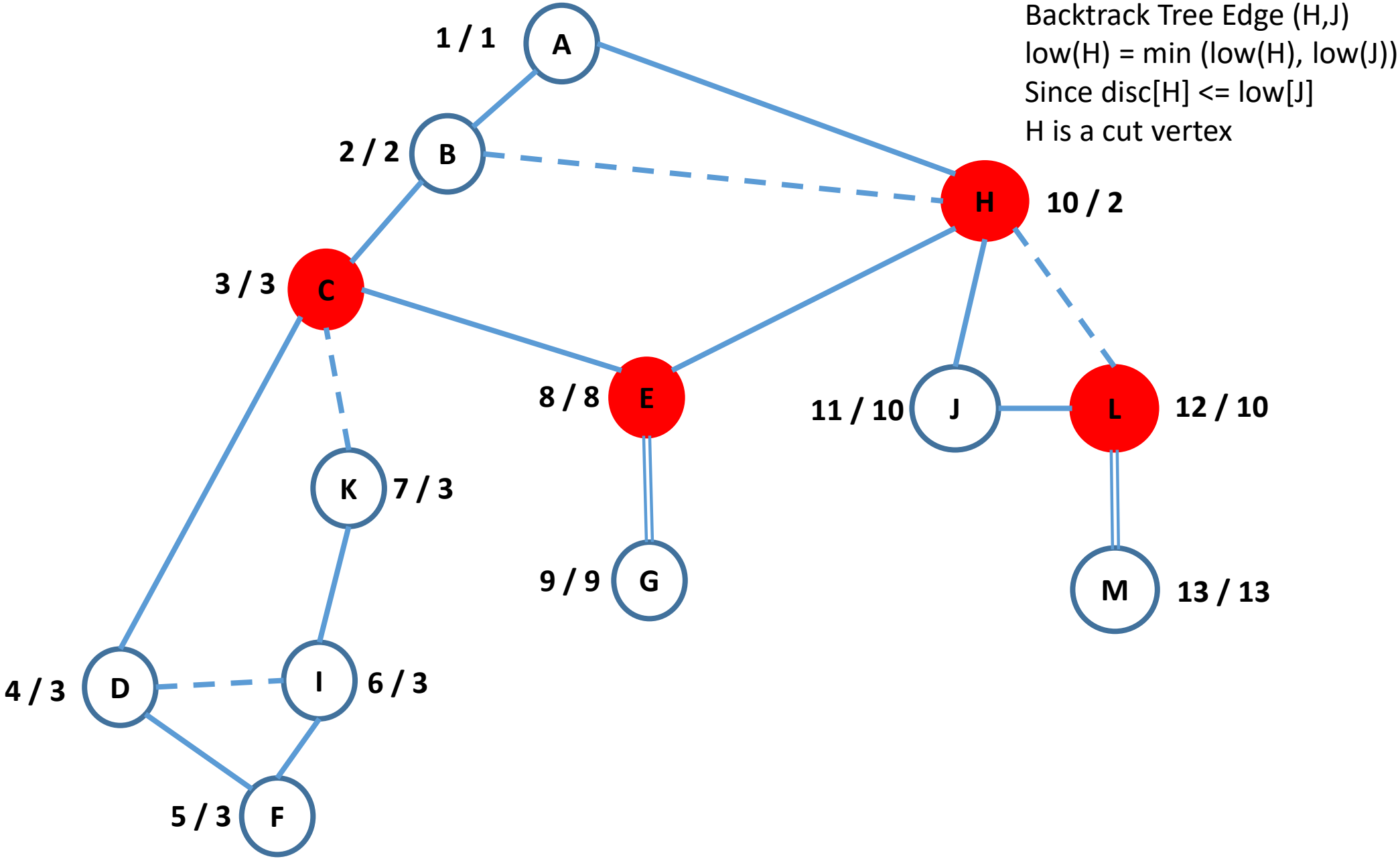

1 / 1
A
2 / 2
B
H
10 / 2
3 / 3
C
8 / 8
E
11 / 11
J
L
12 / 12
K
7 / 3
9 / 9
G
M
13 / 13
4 / 3
D
I
6 / 3
5 / 3
F

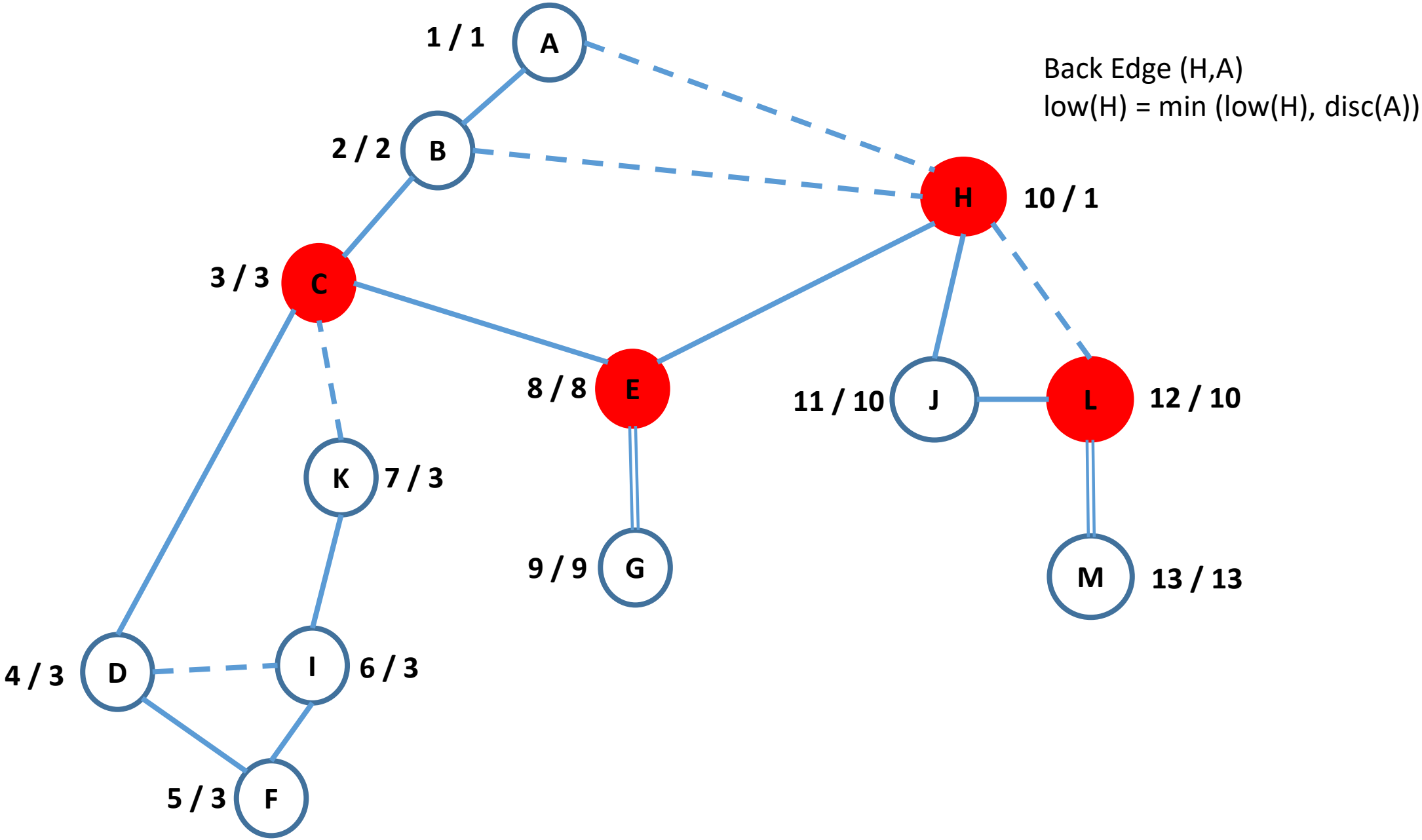


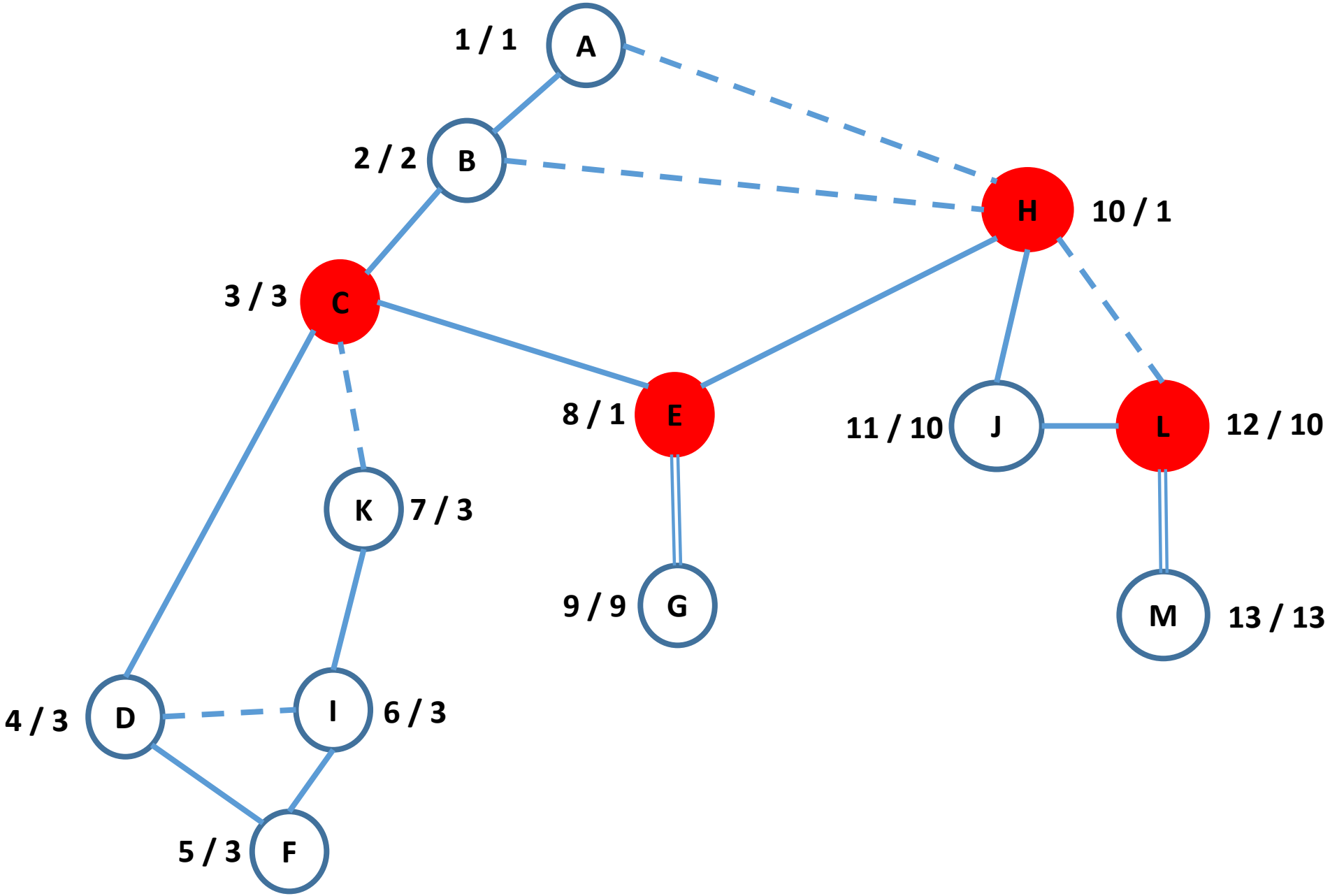
Backtrack Tree Edge (L,M)
low(L) = min (low(L), low(M))
Since disc[L] <= low[M]
L is a cut vertex
Since disc[L] < low[M]
LM is a cut edge

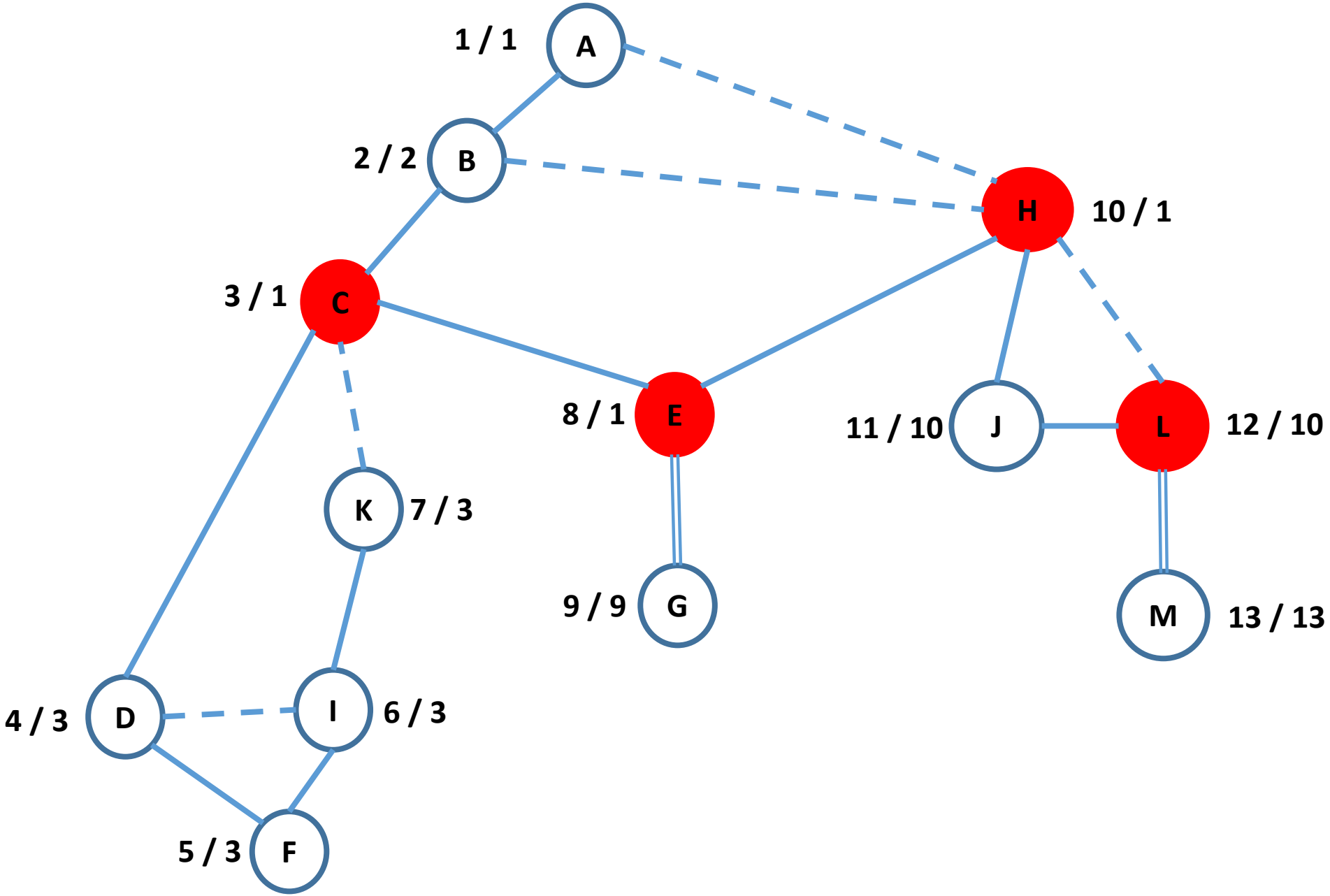

1 / 1
A
2 / 2
B
H
10 / 2
3 / 3
C
8 / 8
E
11 / 11
J
L
12 / 10
K
7 / 3
9 / 9
G
M
13 / 13
4 / 3
D
I
6 / 3
5 / 3
F
Back Edge (L,H)
low(L) = min (low(L), disc(H))

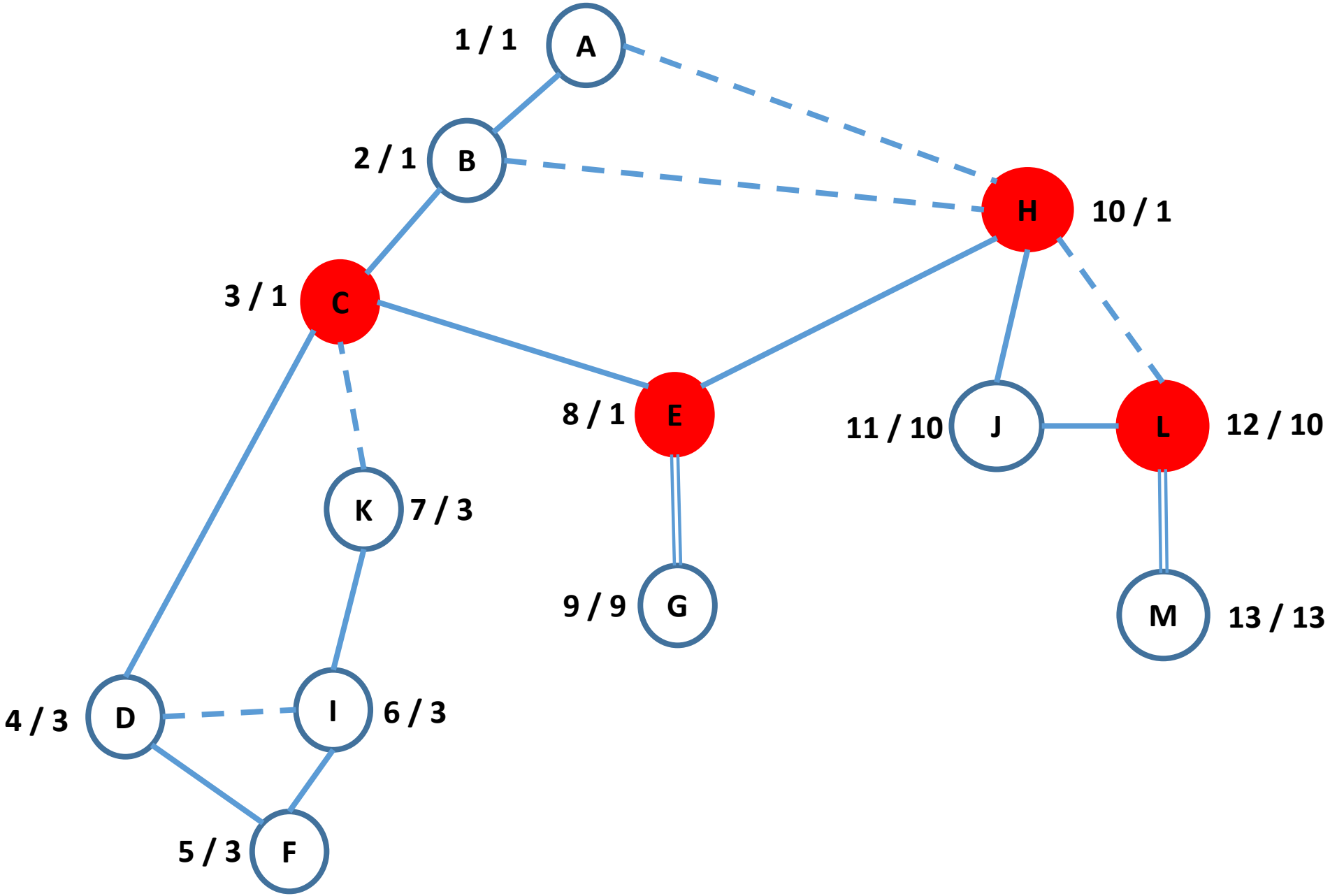

1 / 1
A
2 / 2
B
H
10 / 2
3 / 3
C
8 / 8
E
11 / 10
J
L
12 / 10
K
7 / 3
9 / 9
G
M
13 / 13
4 / 3
D
I
6 / 3
5 / 3
F


Backtrack Tree Edge (H,J)
low(H) = min (low(H), low(J))
Since disc[H] <= low[J]
H is a cut vertex
1 / 1
A
2 / 2
B
3 / 3
C
H
10 / 2
8 / 8
E
11 / 10
J
L
12 / 10
K
7 / 3
9 / 9
G
M
13 / 13
4 / 3
D
I
6 / 3
5 / 3
F


1 / 1
A
2 / 2
B
3 / 3
C
H
10 / 1
Back Edge (H,A)
low(H) = min (low(H), disc(A))
8 / 8
E
11 / 10
J
L
12 / 10
K
7 / 3
9 / 9
G
M
13 / 13
4 / 3
D
I
6 / 3
5 / 3
F


1 / 1
A
2 / 2
B
H
10 / 1
3 / 3
C
8 / 1
E
11 / 10
J
L
12 / 10
K
7 / 3
9 / 9
G
M
13 / 13
4 / 3
D
I
6 / 3
5 / 3
F


1 / 1
A
2 / 2
B
H
10 / 1
3 / 1
C
8 / 1
E
11 / 10
J
L
12 / 10
K
7 / 3
9 / 9
G
M
13 / 13
4 / 3
D
I
6 / 3
5 / 3
F


1 / 1
A
2 / 1
B
H
10 / 1
3 / 1
C
8 / 1
E
11 / 10
J
L
12 / 10
K
7 / 3
9 / 9
G
M
13 / 13
4 / 3
D
I
6 / 3
5 / 3
F

Significance of Low Value

- Low value of a vertex u gives the minimum discovery time of a vertex v above it in the DFS tree that can be reached from any vertex in u’s subtree through some back edge.