MAIN OBSERVATION ABOUT BFS TREE

- → THERE IS NO NON-TREE EDGE FROM LEVEL i TO LEVEL $\geq i+2$

- → FOR EACH NON-TREE EDGE (u,v)
  $|\text{LEVEL}(u) - \text{LEVEL}(v)| \leq 1$

MAIN OBSERVATION ABOUT BFS TREE

→ THERE IS NO NON-TREE EDGE FROM LEVEL i TO LEVEL $\geq i+2$

→ FOR EACH NON-TREE EDGE (u,v)
$$|\text{LEVEL}(u) - \text{LEVEL}(v)| \leq 1$$

ASSUME THAT THE GRAPH IS DIRECTED

Q: IS THERE ANY CHANGE IN ALGO & OBSERVATIONS ABOUT BFS.

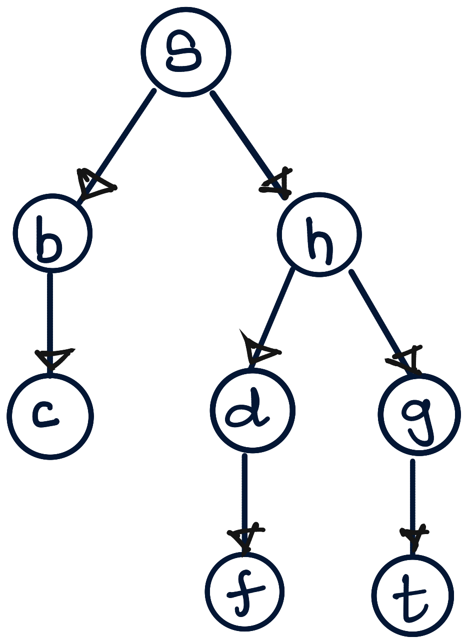
```
BFS(s)        (UNDIRECTED GRAPH)
{   for each v ∈ V
    {
        discovered[v] ← false;
    }
    discovered[s] ← true;
    Q.enqueue(s)
    while (Q is not empty)
    {   v ← Q.deque();
        for each neighbor w of v
        {   if (discovered[w] = false)
            {
                discovered[w] = true ;   Q.enque(w);
                add (v,w) to the BFS tree;
            }
        }
    }
}
```

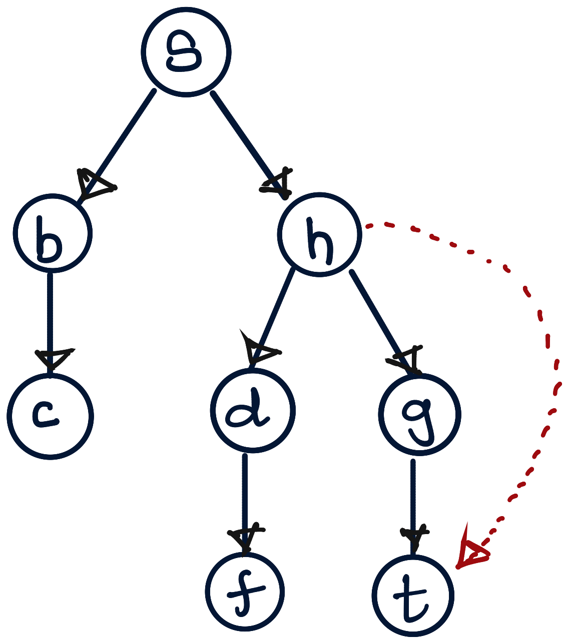
```
BFS(s)      (UNDIRECTED GRAPH)
{   for each v ∈ V
    {
        discovered[v] ← false;
    }
    discovered[s] ← true;

    Q.enqueue(s)
    while (Q is not empty)
    {   v ← Q.deque();
        for each neighbor w of v
        {   if (discovered[w] = false)
            {
                discovered[w] = true ;  Q.enque(w);
                add (v,w) to the BFS tree;
            }
        }
    }
}
```

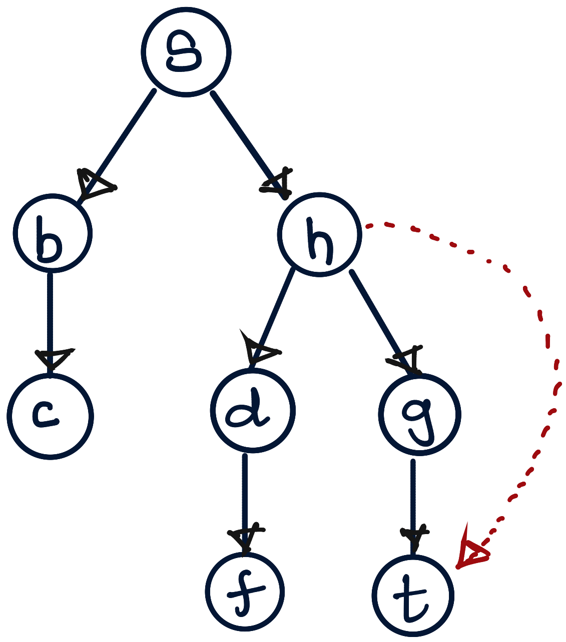
```
BFS(s)      (DIRECTED GRAPH)
{   for each v ∈ V
    {
        discovered[v] ← false;
    }
    discovered[s] ← true;

    Q.enqueue(s)
    while (Q is not empty)
    {   v ← Q.deque();
        for each neighbor w of v    (v → w edge)
        {   if (discovered[w] = false)
            {
                discovered[w] = true ;  Q.enque(w);
                add (v,w) to the BFS tree;
            }
        }
    }
}
```


s
b
h
c
d
g
f
t

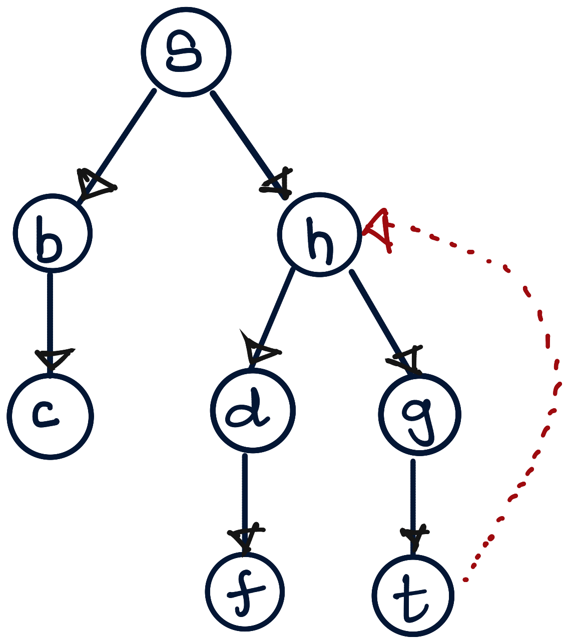


Q: CAN THERE BE A NON-TREE EDGE FROM h TO t ?

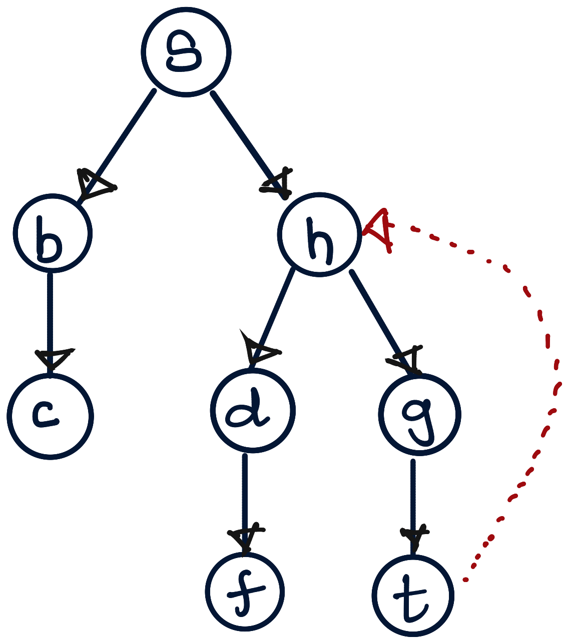


Q: CAN THERE BE A NON-TREE EDGE FROM h TO t?

A: NO

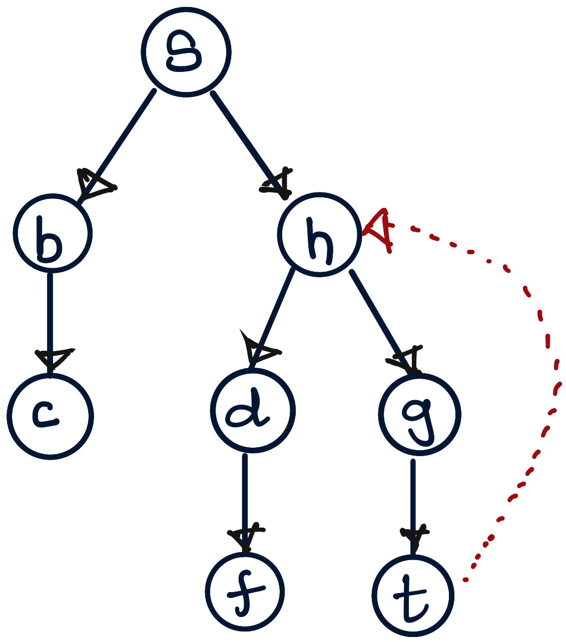


Q: CAN THERE BE A NON-TREE EDGE FROM t TO h ?



Q: CAN THERE BE A NON-TREE EDGE FROM t TO h ?

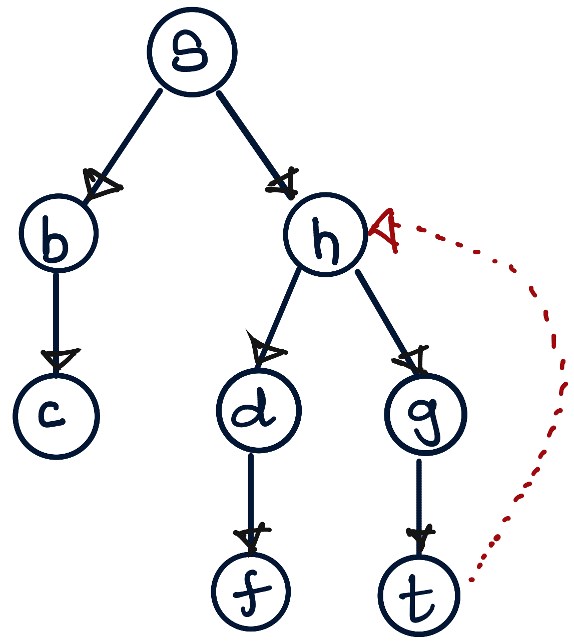
A : YES



Q: CAN THERE BE A NON-TREE EDGE FROM t TO h ?

A : YES

LEMMA : FOR ANY NON TREE EDGE $u \to v$
$\text{LEVEL}(v) - \text{LEVEL}(u) \leq 1$



Q: CAN THERE BE A NON-TREE EDGE FROM t TO h ?

A : YES

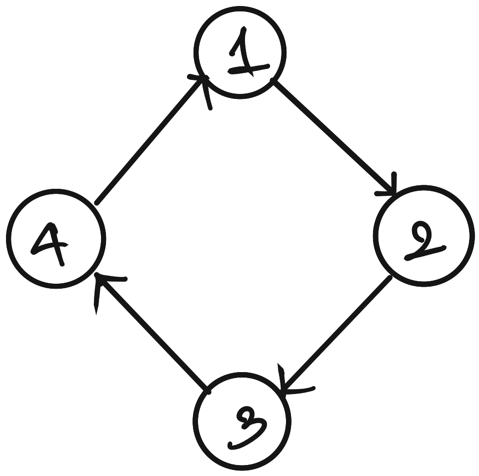
LEMMA : FOR ANY NON TREE EDGE $u \to v$
$\text{LEVEL}(v) - \text{LEVEL}(u) \le 1$

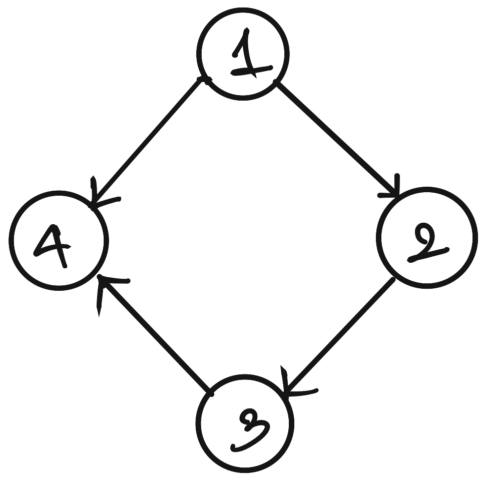
LEMMA : FOR EACH NON-TREE EDGE (u,v)
$|\text{LEVEL}(u) - \text{LEVEL}(v)| \le 1$

FOR UNDIRECTED GRAPHS.

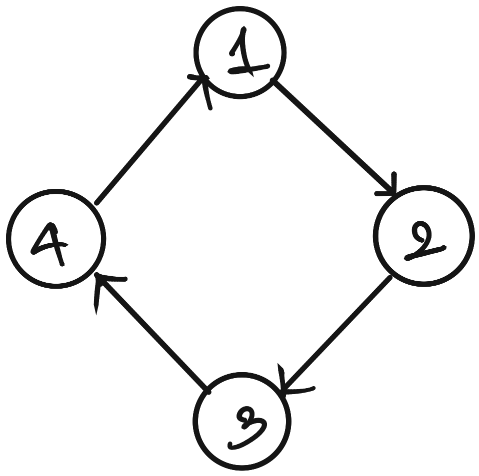
A DIRECTED GRAPH IS CALLED STRONGLY CONNECTED IF THERE IS A DIRECTED PATH BETWEEN ANY TWO NODES.

A DIRECTED GRAPH IS CALLED STRONGLY CONNECTED IF THERE IS A DIRECTED PATH BETWEEN ANY TWO NODES.

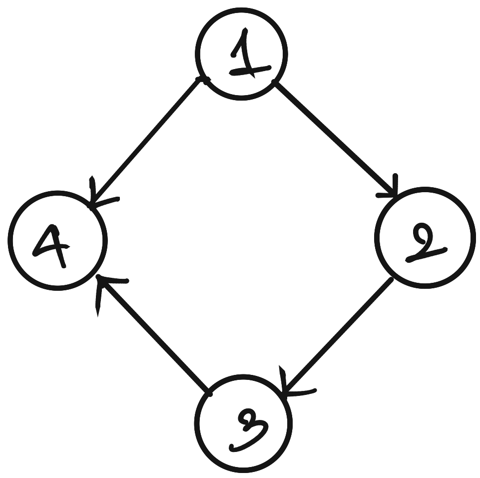




A DIRECTED GRAPH IS CALLED STRONGLY CONNECTED IF THERE IS A DIRECTED PATH BETWEEN ANY TWO NODES.

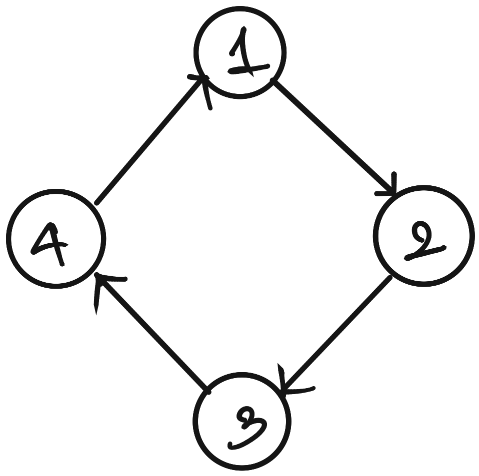


STRONGLY CONNECTED

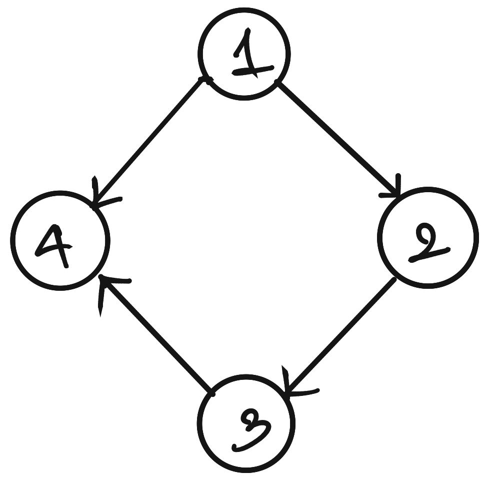


NOT STRONGLY CONNECTED

A DIRECTED GRAPH IS CALLED STRONGLY CONNECTED IF THERE IS A DIRECTED PATH BETWEEN ANY TWO NODES.



STRONGLY CONNECTED



NOT STRONGLY CONNECTED

Q: GIVEN A DIRECTED GRAPH, FIND IF IT IS STRONGLY CONNECTED OR NOT.

DEFINITION OF STRONGLY CONNECTED:

FOR EACH NODE $v \in V$, THERE IS A PATH FROM v TO EVERY OTHER VERTEX.

DEFINITION OF STRONGLY CONNECTED:

FOR EACH NODE $v \in V$, THERE IS A PATH FROM v TO EVERY OTHER VERTEX.

ALGO

FOR EACH $v \in V$
{
    FIND IF ALL OTHER VERTICES ARE
    REACHABLE FROM v
    IF NO
    {
        GRAPH IS NOT STRONGLY CONNECTED
        RETURN
    }
}
GRAPH IS STRONGLY CONNECTED

DEFINITION OF STRONGLY CONNECTED:

FOR EACH NODE $v \in V$, THERE IS A PATH FROM v TO EVERY OTHER VERTEX.

ALGO

```
FOR EACH v ∈ V
{    FIND IF ALL OTHER VERTICES ARE
     REACHABLE FROM v                  } HOW TO DO THIS PART
     IF NO
     {   GRAPH IS NOT STRONGLY CONNECTED
         RETURN
     }
}
GRAPH IS STRONGLY CONNECTED
```

DEFINITION OF STRONGLY CONNECTED:

FOR EACH NODE $v \in V$, THERE IS A PATH FROM v TO EVERY OTHER VERTEX.

ALGO

```
FOR EACH v ∈ V
{       DO BFS(v)

        ..

    IF BFSTREE(v) DOES NOT CONTAIN ALL VERTICES
    {   GRAPH IS NOT STRONGLY CONNECTED
        RETURN
    }
}
GRAPH IS STRONGLY CONNECTED
```

DEFINITION OF STRONGLY CONNECTED:

FOR EACH NODE $v \in V$, THERE IS A PATH FROM v TO EVERY OTHER VERTEX.

ALGO

FOR EACH $v \in V$
{
    DO BFS(v) ← TIME : $O(m+n)$

    IF BFSTREE(v) DOES NOT CONTAIN ALL VERTICES
    {
        GRAPH IS NOT STRONGLY CONNECTED
        RETURN
    }
}
GRAPH IS STRONGLY CONNECTED

DEFINITION OF STRONGLY CONNECTED:

FOR EACH NODE $v \in V$, THERE IS A PATH FROM v TO EVERY OTHER VERTEX.

ALGO

FOR EACH $v \in V$
{
    DO BFS(v) ← TIME : $O(m+n)$

    TIME: $O(n)$

    IF BFSTREE(v) DOES NOT CONTAIN ALL VERTICES
    {
        GRAPH IS NOT STRONGLY CONNECTED
        RETURN
    }
}
GRAPH IS STRONGLY CONNECTED

TOTAL TIME

DEFINITION OF STRONGLY CONNECTED:

FOR EACH NODE $v \in V$, THERE IS A PATH FROM v TO EVERY OTHER VERTEX.

ALGO

FOR EACH $v \in V$
{
DO BFS(v) ← TIME : $O(m+n)$

TIME : $O(n)$

IF BFSTREE(v) DOES NOT CONTAIN ALL VERTICES
{
GRAPH IS NOT STRONGLY CONNECTED
RETURN
}
}
GRAPH IS STRONGLY CONNECTED

TOTAL TIME : $O(n(m+n))$

$= O(mn)$

DEFINITION OF STRONGLY CONNECTED:

FOR EACH NODE $v \in V$, THERE IS A PATH FROM v TO EVERY OTHER VERTEX.

ALGO

FOR EACH $v \in V$
{
DO BFS(v) ← TIME : $O(m+n)$

TIME: $O(n)$

IF BFSTREE(v) DOES NOT CONTAIN ALL VERTICES
{ GRAPH IS NOT STRONGLY CONNECTED
RETURN
}
}
GRAPH IS STRONGLY CONNECTED

TOTAL TIME : $O(n(m+n))$

$= O(mn)$

Q: CAN YOU DO BETTER?

IN OUR PREVIOUS ALGORITHM, WE PERFORMED n BFS. CAN THIS PROBLEM BE SOLVED IN LESS NUMBER OF BFS CALL.

IN OUR PREVIOUS ALGORITHM, WE PERFORMED n BFS. CAN THIS PROBLEM BE SOLVED IN LESS NUMBER OF BFS CALL.

OBSERVATION :

FIX ANY VERTEX v IN A STRONGLY CONNECTED GRAPH

1) THERE IS A DIRECTED PATH FROM ALL OTHER VERTICES TO v.
2) THERE IS A DIRECTED PATH FROM v TO ALL OTHER VERTICES.

IN OUR PREVIOUS ALGORITHM, WE PERFORMED n BFS. CAN THIS PROBLEM BE SOLVED IN LESS NUMBER OF BFS CALL.

OBSERVATION :

FIX ANY VERTEX v IN A STRONGLY CONNECTED GRAPH

A 1) THERE IS A DIRECTED PATH FROM ALL OTHER VERTICES TO v.

B 2) THERE IS A DIRECTED PATH FROM v TO ALL OTHER VERTICES.

IN OUR PREVIOUS ALGORITHM, WE PERFORMED n BFS. CAN THIS PROBLEM BE SOLVED IN LESS NUMBER OF BFS CALL.

OBSERVATION :

FIX ANY VERTEX v IN A STRONGLY CONNECTED GRAPH

A 1) THERE IS A DIRECTED PATH FROM ALL OTHER VERTICES TO v.

B 2) THERE IS A DIRECTED PATH FROM v TO ALL OTHER VERTICES.

IS THIS TRUE ?

IF A & B ARE TRUE IN A DIRECTED GRAPH, THEN THE GRAPH IS STRONGLY CONNECTED.

IN OUR PREVIOUS ALGORITHM, WE PERFORMED n BFS. CAN THIS PROBLEM BE SOLVED IN LESS NUMBER OF BFS CALL.

OBSERVATION :

FIX ANY VERTEX v IN A STRONGLY CONNECTED GRAPH

A 1) THERE IS A DIRECTED PATH FROM ALL OTHER VERTICES TO v.

B 2) THERE IS A DIRECTED PATH FROM v TO ALL OTHER VERTICES.

IS THIS TRUE ?

IF A & B ARE TRUE IN A DIRECTED GRAPH, THEN THE GRAPH IS STRONGLY CONNECTED.

TO SHOW THAT FOR ANY PAIR x ANY y, y IS REACHABLE FROM x $(x \rightsquigarrow y)$ AND x IS REACHABLE FROM y $(y \rightsquigarrow x)$

IN OUR PREVIOUS ALGORITHM, WE PERFORMED n BFS. CAN THIS PROBLEM BE SOLVED IN LESS NUMBER OF BFS CALL.

OBSERVATION :

FIX ANY VERTEX v IN A STRONGLY CONNECTED GRAPH

(A) 1) THERE IS A DIRECTED PATH FROM ALL OTHER VERTICES TO v.

(B) 2) THERE IS A DIRECTED PATH FROM v TO ALL OTHER VERTICES.

IS THIS TRUE ?

IF (A) & (B) ARE TRUE IN A DIRECTED GRAPH, THEN THE GRAPH IS STRONGLY CONNECTED.

TO SHOW THAT FOR ANY PAIR x ANY y, y IS REACHABLE FROM x ($x \rightsquigarrow y$) AND x IS REACHABLE FROM y ($y \rightsquigarrow x$)

$x \overset{A}{\rightsquigarrow} v \overset{B}{\rightsquigarrow} y$ $\qquad$ $y \overset{A}{\rightsquigarrow} v \overset{B}{\rightsquigarrow} x$

```
PICK ANY VERTEX v;
DO BFS(v)
IF ( BFS.TREE(v) DOES NOT CONTAIN ALL VERTICES)
{
    G IS NOT STRONGLY CONNECTED
    RETURN
}
```

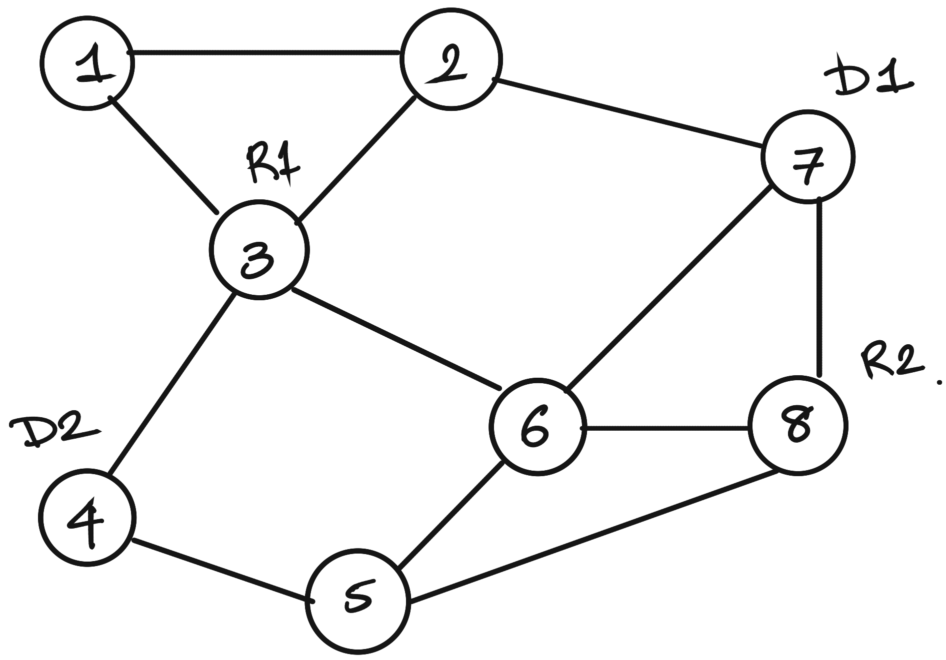
```
PICK ANY VERTEX v;
DO BFS(v)
IF ( BFS.TREE(v) DOES NOT CONTAIN ALL VERTICES)
{
    G IS NOT STRONGLY CONNECTED
    RETURN
}
REVERSE THE GRAPH
DO BFS(v)
IF ( BFS.TREE(v) DOES NOT CONTAIN ALL VERTICES)
{
    G IS NOT STRONGLY CONNECTED
    RETURN
}
G IS STRONGLY CONNECTED.
```

RUNNING TIME:

```
PICK ANY VERTEX v;
DO BFS(v)
IF ( BFS.TREE(v) DOES NOT CONTAIN ALL VERTICES)
{
    G IS NOT STRONGLY CONNECTED
    RETURN
}
REVERSE THE GRAPH
DO BFS(v)
IF ( BFS.TREE(v) DOES NOT CONTAIN ALL VERTICES)
{
    G IS NOT STRONGLY CONNECTED
    RETURN
}
G IS STRONGLY CONNECTED.
```

RUNNING TIME: $O(m+n)$

PROBLEM:

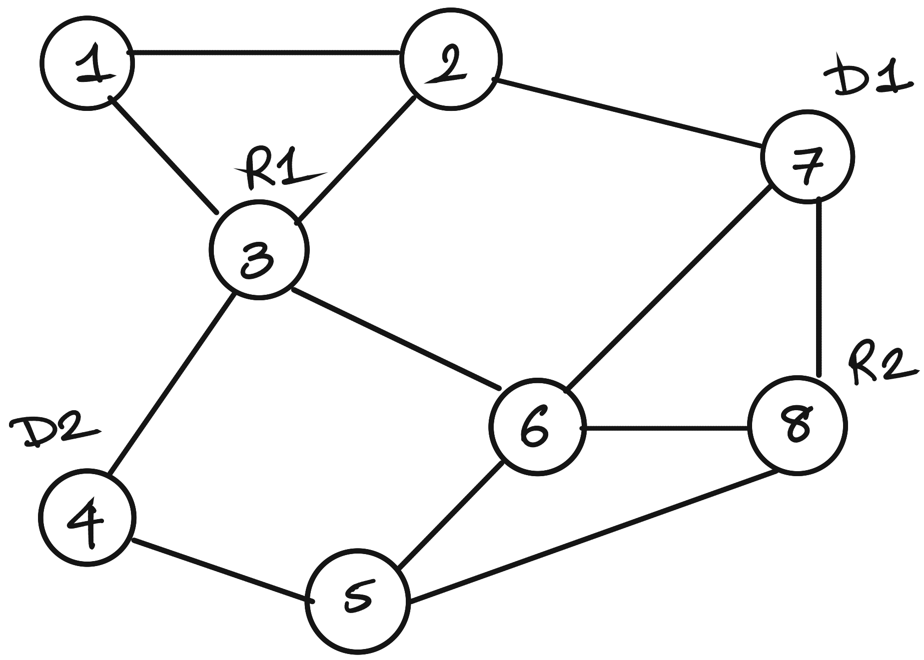


WE WANT TO MOVE FROM THE SOURCE TO DESTINATION FOR BOTH THE ROBOTS AS FOLLOWS

1) AT ONE TIME STEP, EXACTLY ONE ROBOT MOVES ACROSS AN EDGE
2) THE DISTANCE BETWEEN TWO ROBOTS $\geq K$

(K=2 HERE)

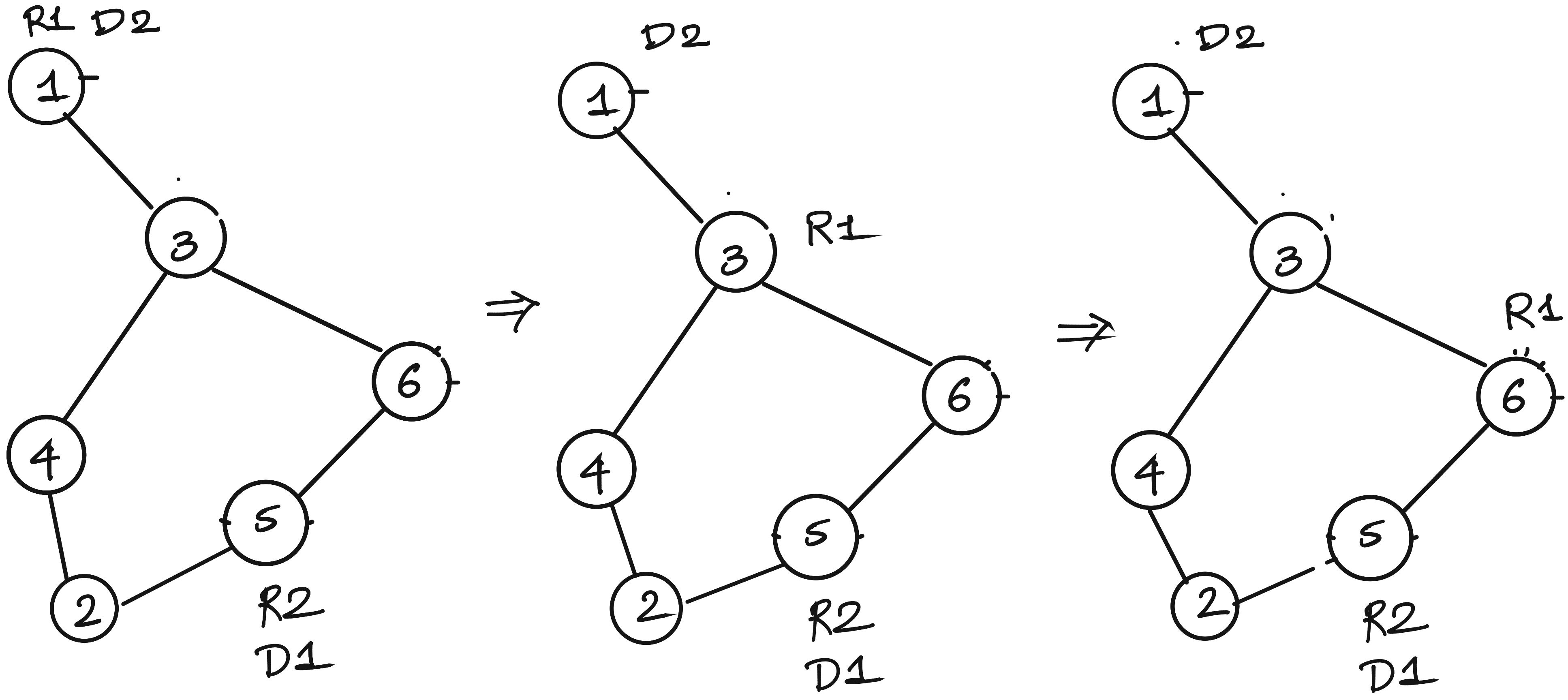
PROBLEM :



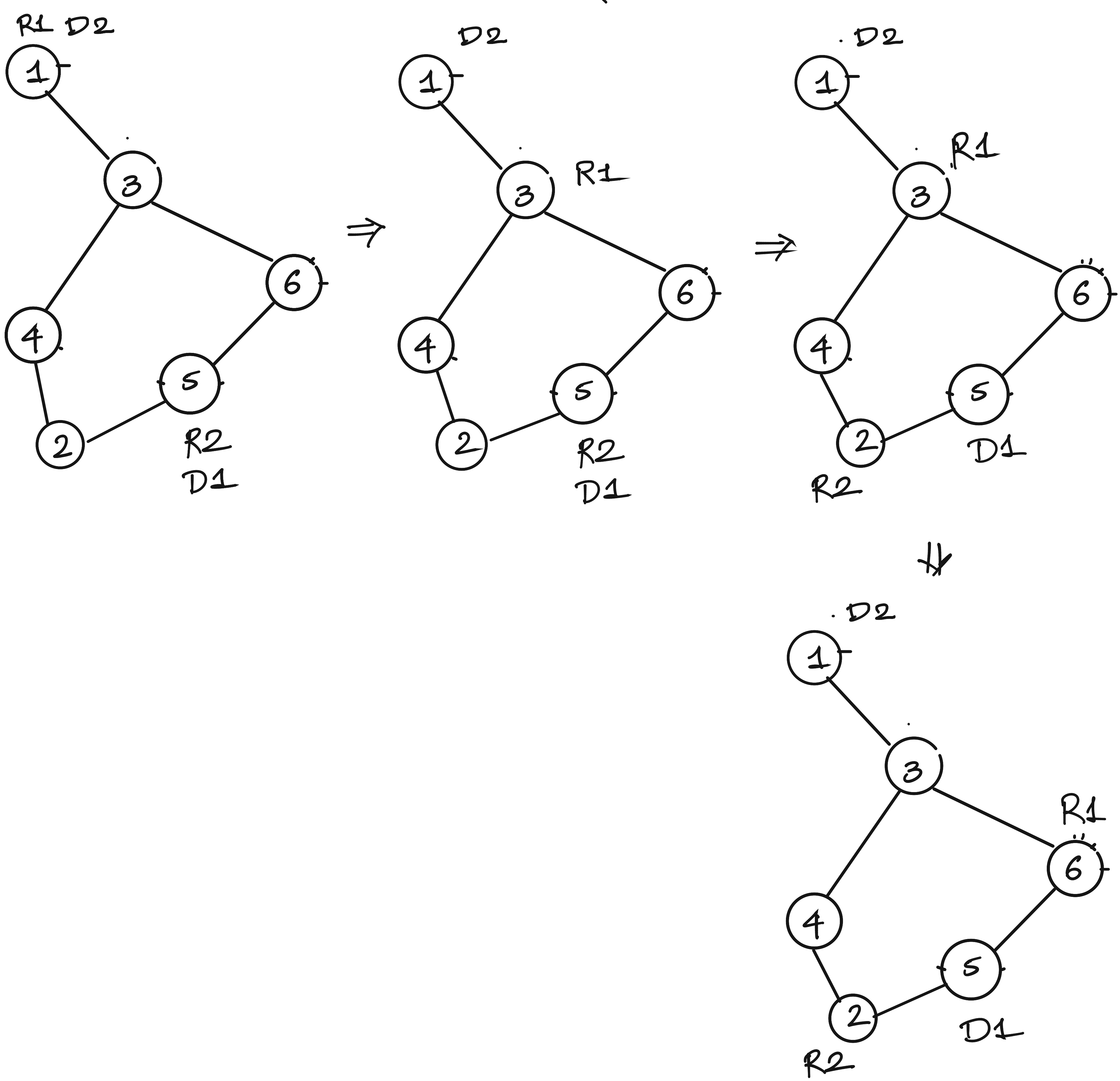
WE WANT TO MOVE FROM THE SOURCE TO DESTINATION FOR BOTH THE ROBOTS AS FOLLOWS

1) AT ONE TIME STEP, EXACTLY ONE ROBOT MOVES ACROSS AN EDGE
2) THE DISTANCE BETWEEN TWO ROBOTS $\geq K$ (K=2 HERE)

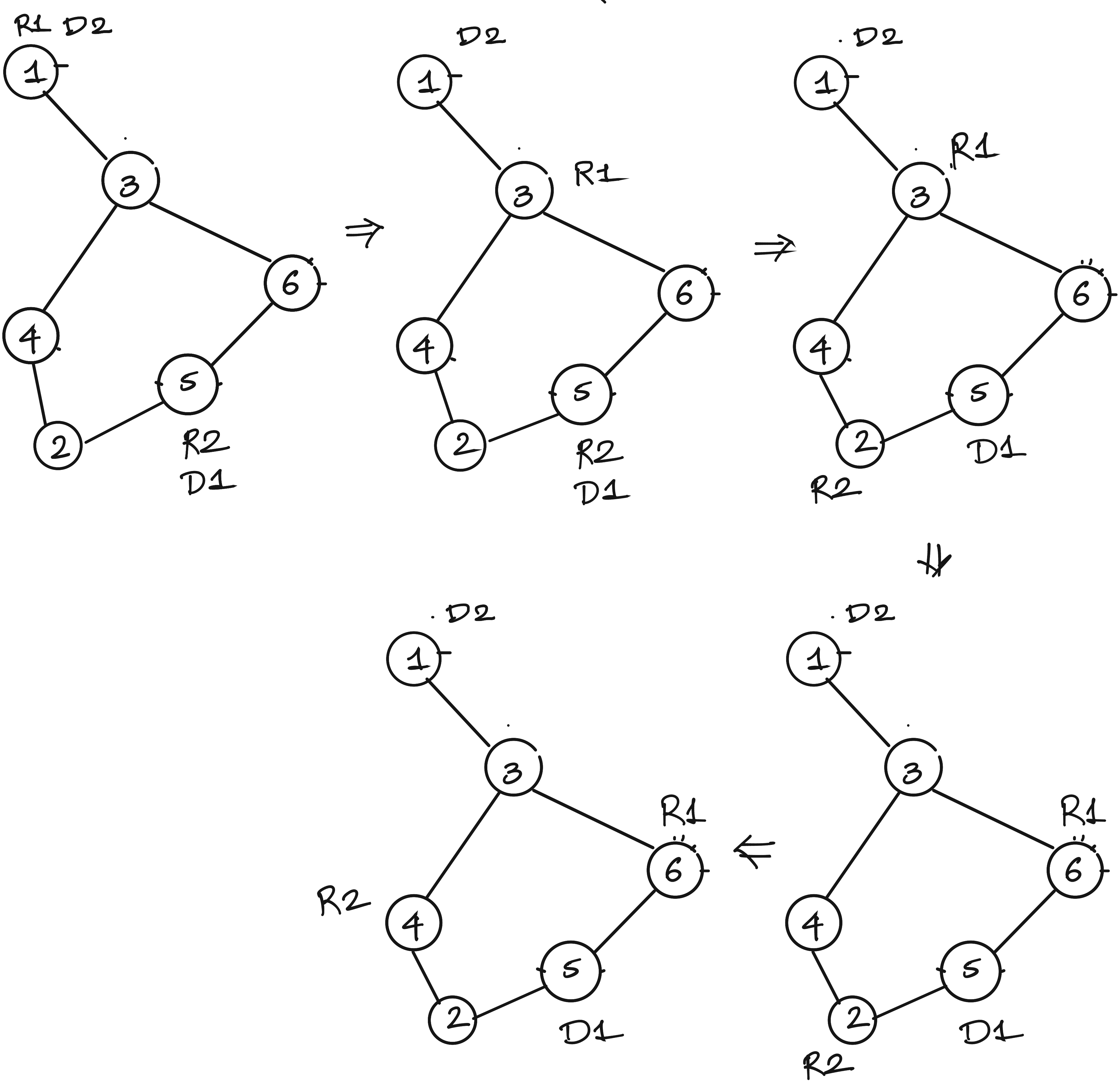
DESIGN AN ALGORITHM THAT FINDS SUCH A WALK.

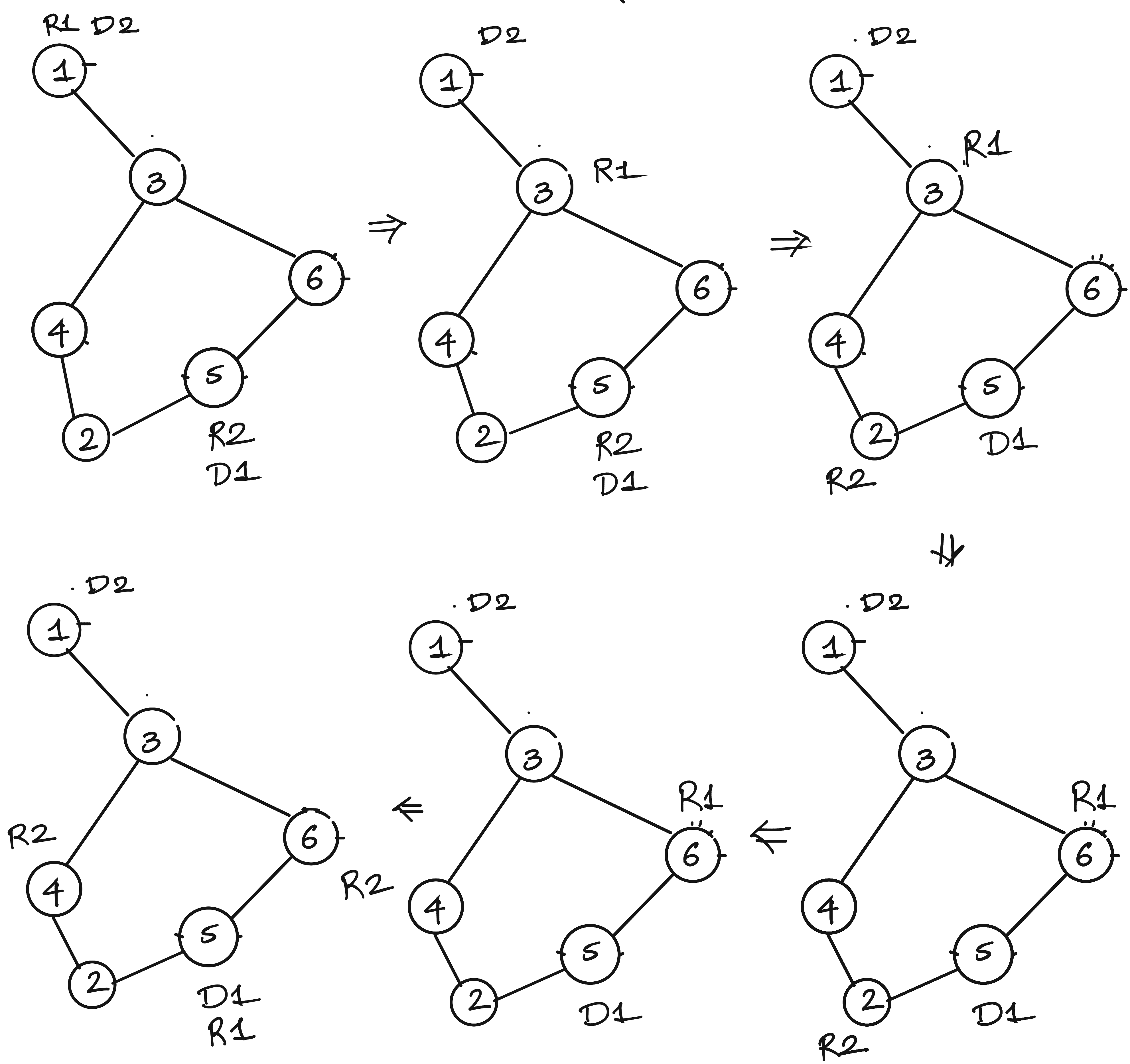


NOT CORRECT



R1 D2
1
3
6
4
5
2
R2
D1
D2
1
3
R1
6
4
5
2
R2
D1
D2
1
R1
3
6
4
5
2
D1
R2
D2
1
3
R1
6
4
5
2
D1
R2


R1 D2
R2
D1
D2
R1
R2
D1
D2
R1
R2
D1
D2
R1
R2
D1
D2
R2
R1
D1



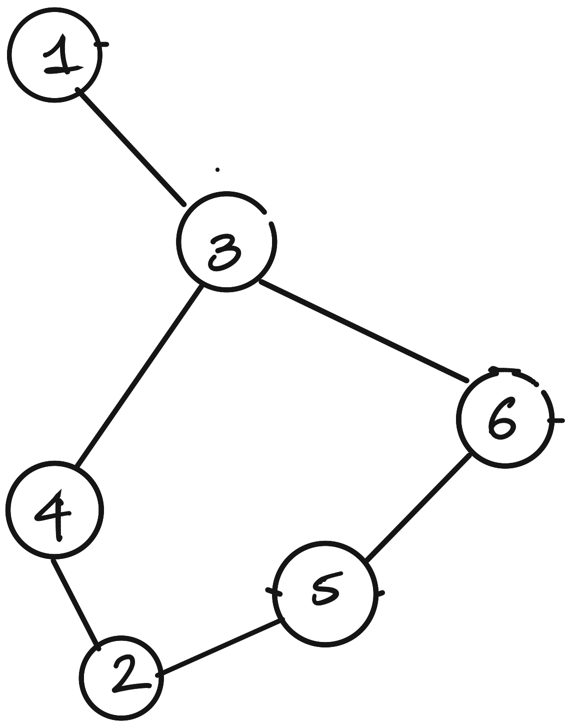
R1 D2
1
3
4
6
5
2
R2
D1
D2
1
3
R1
4
6
5
2
R2
D1
D2
1
R1
3
4
6
5
2
D1
R2
D2
1
3
6
R2
4
5
2
D1
R1
D2
1
3
R1
6
R2
4
5
2
D1
D2
1
3
R1
6
4
5
2
D1
R2

→ WE WANT TO DO BFS IN THIS GRAPH BUT THERE IS AN ADDED CONSTRAINT REGARDING THEIR DISTANCE

→ THIS CONSTRAINT DOES NOT ALLOW US TO DO BFS

→ WE WANT TO DO BFS IN THIS GRAPH BUT THERE IS AN ADDED CONSTRAINT REGARDING THEIR DISTANCE

→ THIS CONSTRAINT DOES NOT ALLOW US TO DO BFS



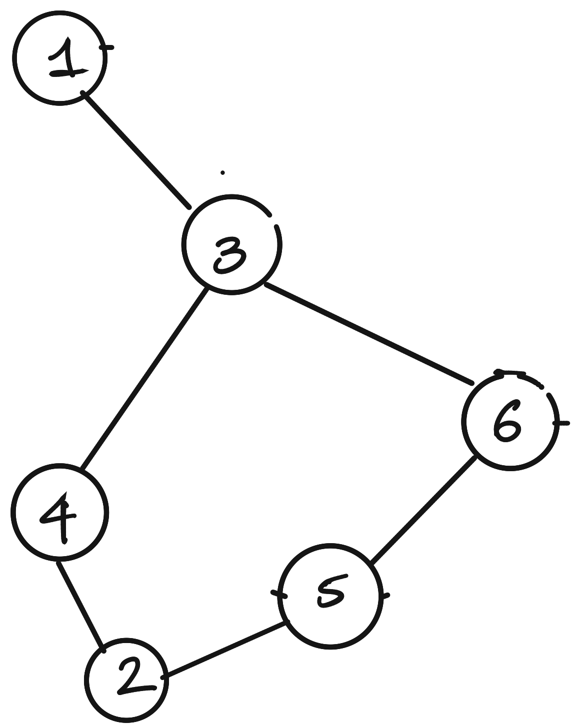
CONFIGURATION : (x, y)

CURRENT PLACE OF R1

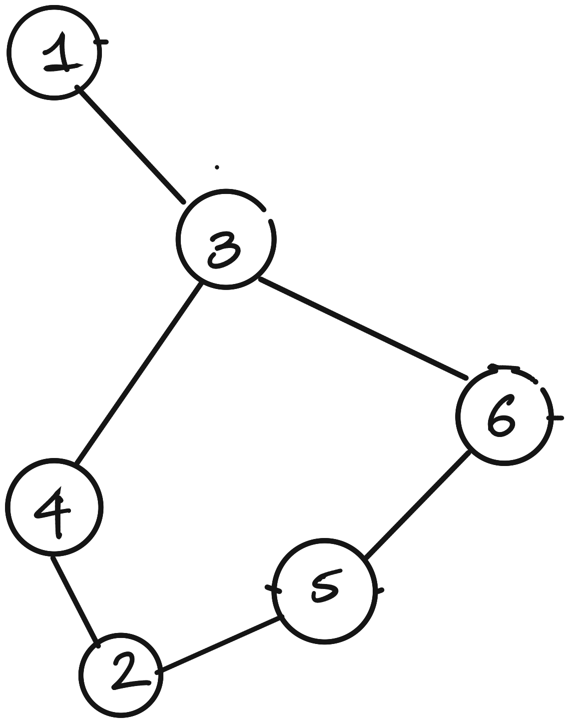
CURRENT PLACE OF R2

SOME CONFIGURATIONS ARE NOT ALLOWED

(3,4) (5,6), (5,5), (4,4) ......

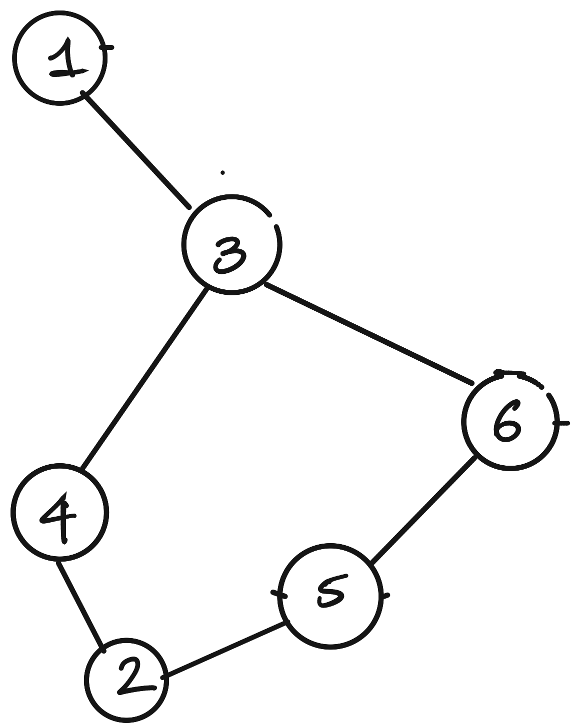


LET US FIND CONFIGURATIONS THAT ARE ALLOWED.



LET US FIND CONFIGURATIONS THAT ARE ALLOWED.

(1,6) (1,5) (1,2) (1,4)



LET US FIND CONFIGURATIONS THAT ARE ALLOWED.

(1,6) (1,5) (1,2) (1,4)

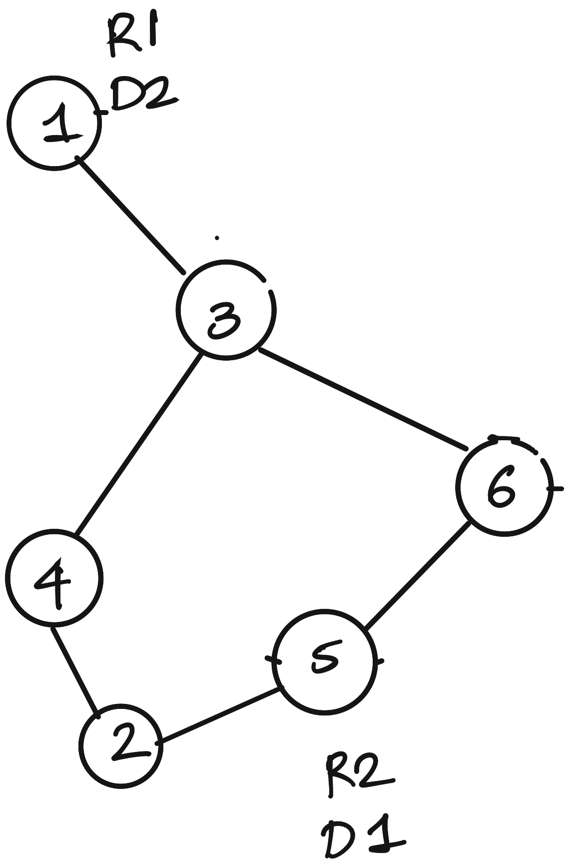
(3,5) (3,2)

(6,1) (6,4) (6,2)

(5,3) (5,4) (5,1)

(2,6) (2,3) (2,1)

(4,1) (4,5) (4,6)



LET US FIND CONFIGURATIONS THAT ARE ALLOWED.

INITIAL

(1,6) (1,5) (1,2) (1,4)

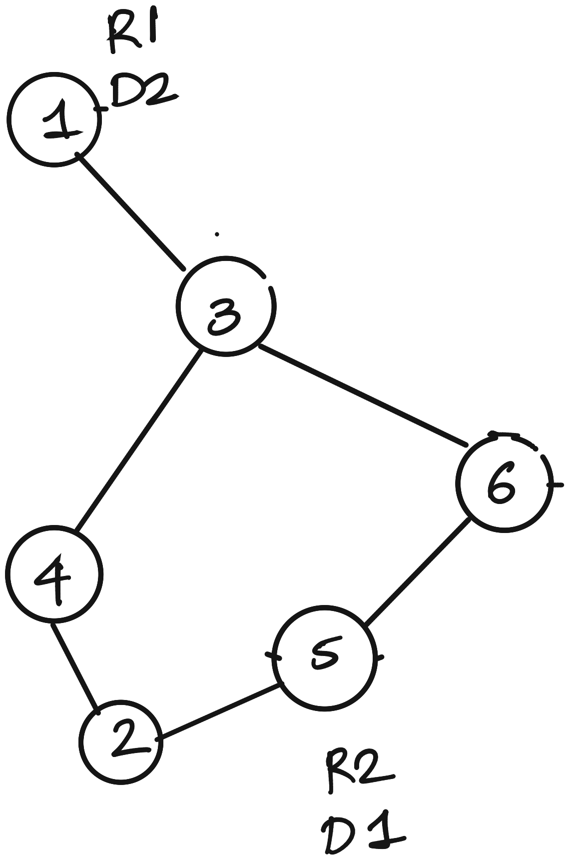
(3,5) (3,2)

(6,1) (6,4) (6,2)

(5,3) (5,4) (5,1) FINAL

(2,6) (2,3) (2,1)

(4,1) (4,5) (4,6)



LET US FIND CONFIGURATIONS THAT ARE ALLOWED.

INITIAL

$(1,6)$ $(1,5)$ — $(1,2)$ $(1,4)$

$(3,5)$ $(3,2)$

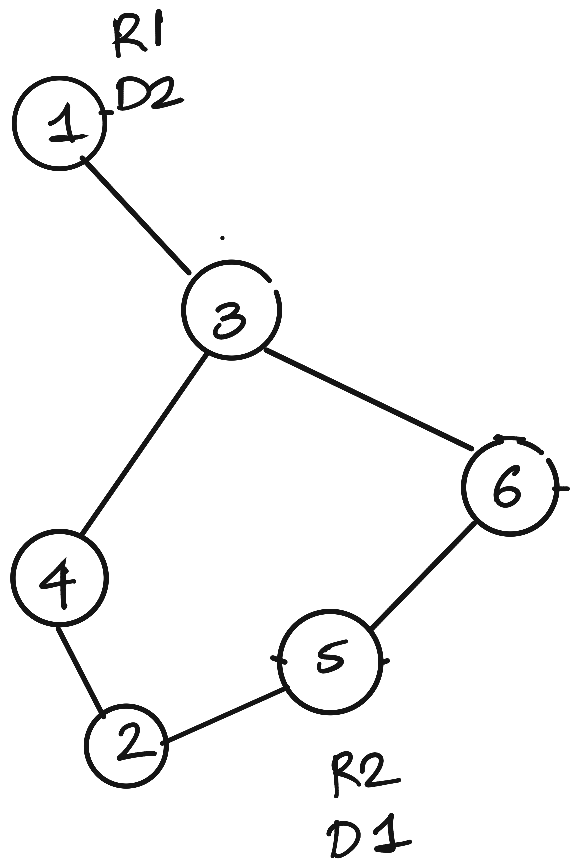
$(6,1)$ $(6,4)$ $(6,2)$

$(5,3)$ $(5,4)$ $(5,1)$ FINAL

$(2,6)$ $(2,3)$ $(2,1)$

$(4,1)$ $(4,5)$ $(4,6)$

CAN GO FROM $(1,5)$ AT $(1,2)$ SINCE $(5,2) \in G$.



LET US FIND CONFIGURATIONS THAT ARE ALLOWED.

INITIAL

(1,6) — (1,5) — (1,2) — (1,4)

(3,5) (3,2)

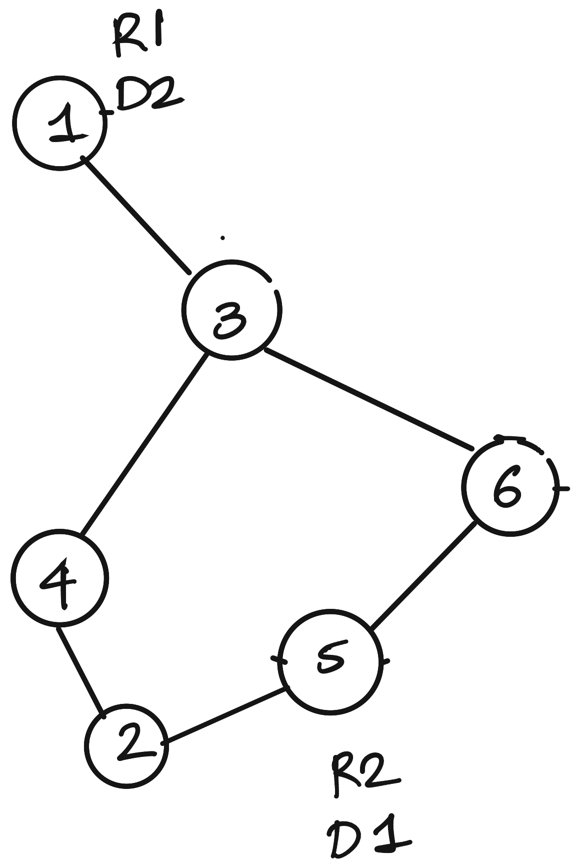
(6,1) (6,4) (6,2)

(5,3) (5,4) (5,1) FINAL

(2,6) (2,3) (2,1)

(4,1) (4,5) (4,6)

CAN GO FROM (1,5) AT (1,2) SINCE (5,2) $\in$ G.



LET US FIND CONFIGURATIONS THAT ARE ALLOWED.

INITIAL

(1,6) — (1,5) — (1,2) — (1,4)

(3,5) (3,2)

(6,1) (6,4) (6,2)

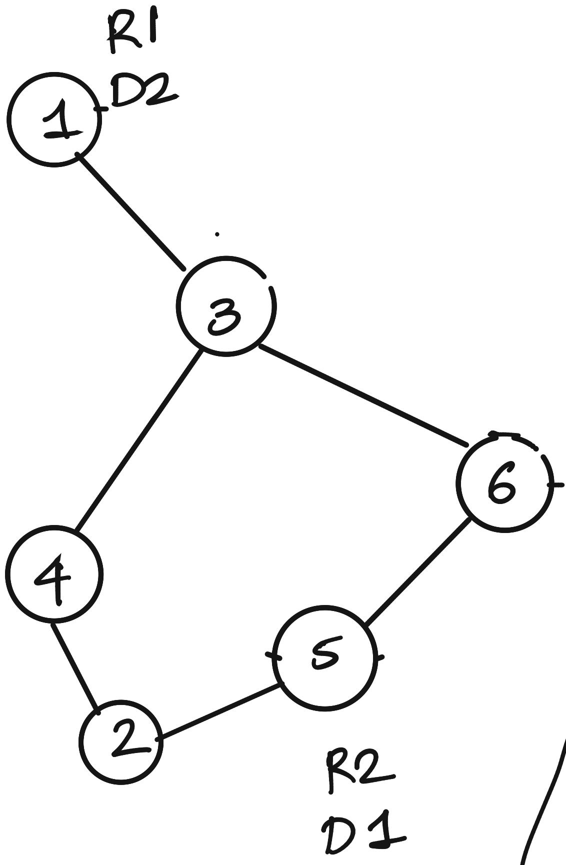
(5,3) (5,4) (5,1) FINAL

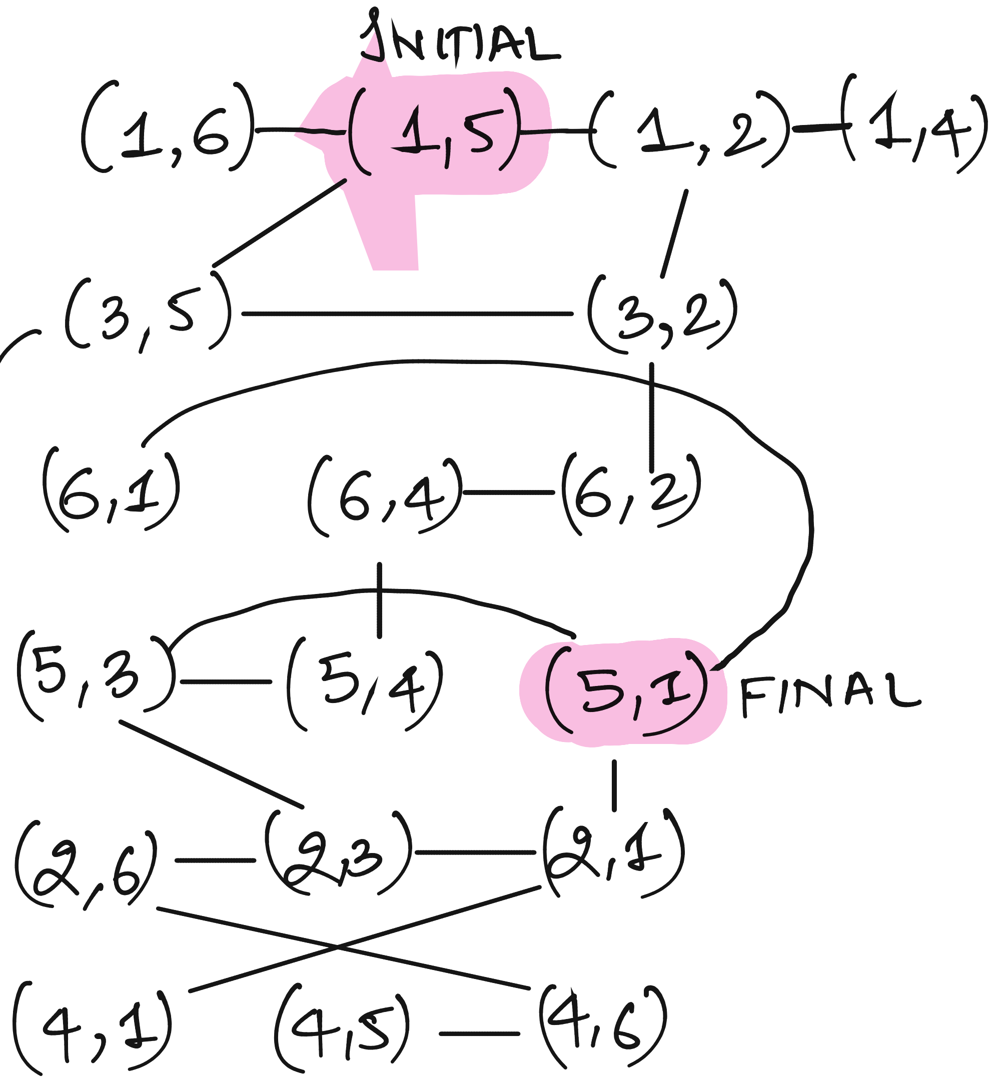
(2,6) (2,3) (2,1)

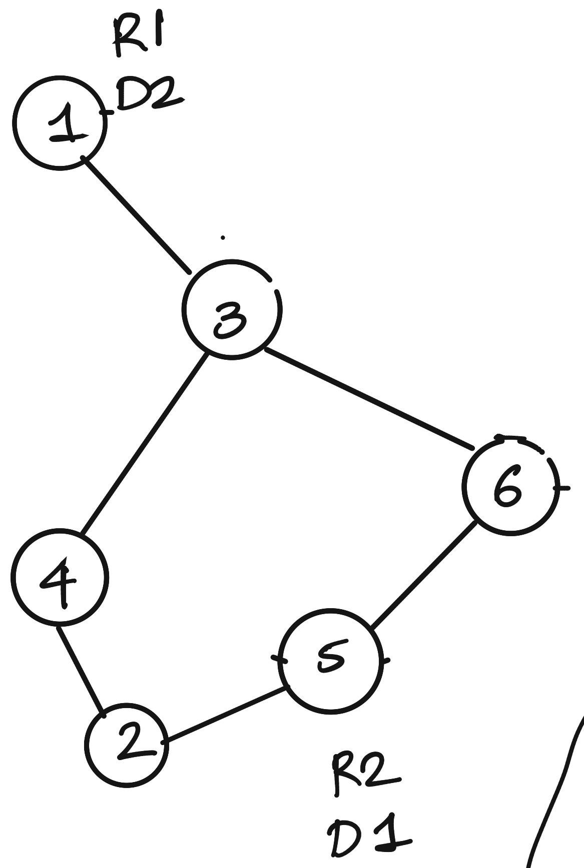
(4,1) (4,5) (4,6)

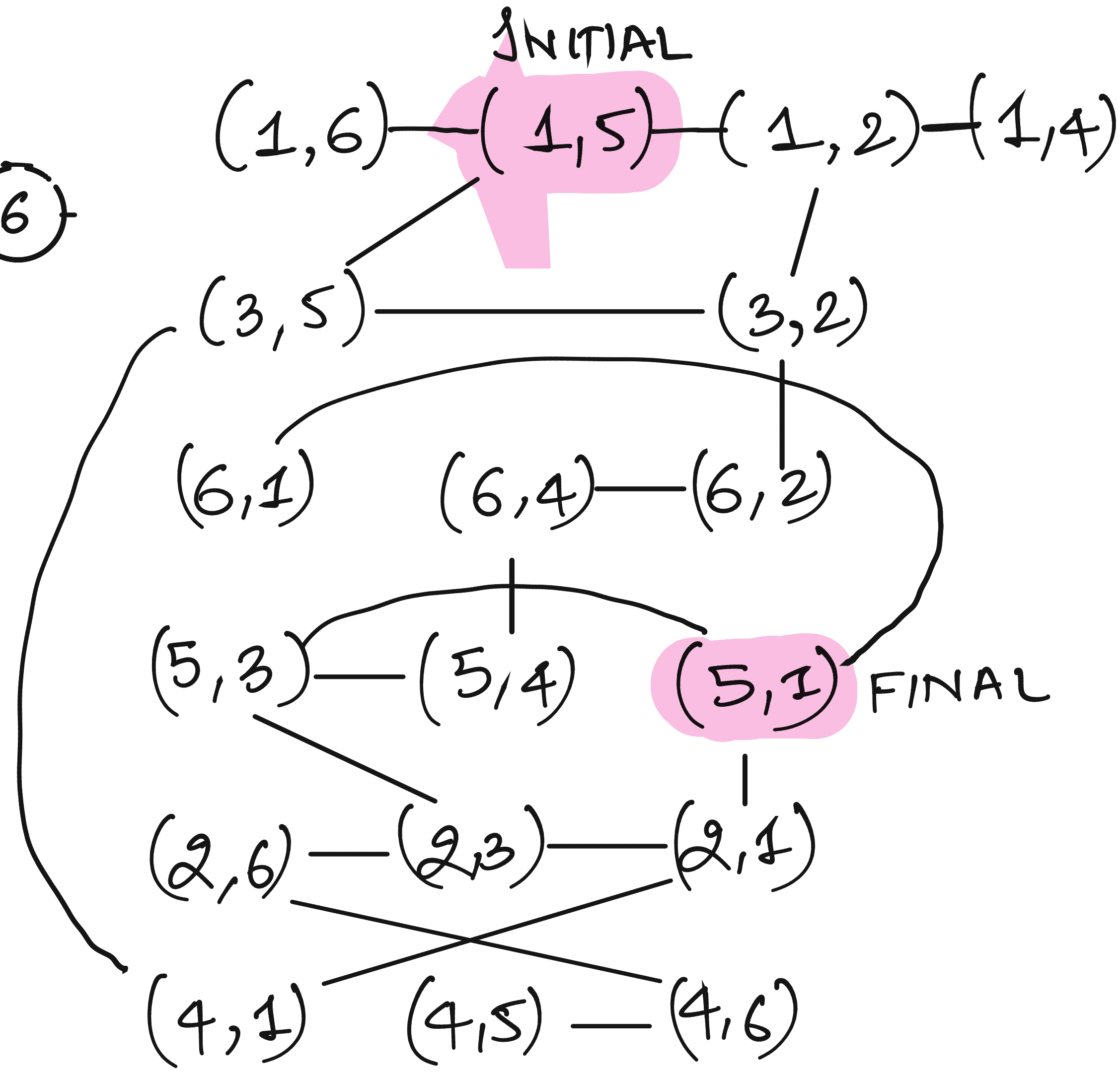
CAN GO FROM (1,5) AT (1,2) SINCE (5,2) $\in$ G.

NO EDGE BETWEEN (1,4) & (1,6)


R1
D2
1
3
6
4
5
2
R2
D1


INITIAL
(1,6)
(1,5)
(1,2)
(1,4)
(3,5)
(3,2)
(6,1)
(6,4)
(6,2)
(5,3)
(5,4)
(5,1)
FINAL
(2,6)
(2,3)
(2,1)
(4,1)
(4,5)
(4,6)





1) FIND ALLOWED CONFIGURATIONS

2) For each (x,y) IN OUR NEW GRAPH H,

THERE IS AN EDGE FROM $(x,y) - (x,y')$ if $(y,y') \in G$ & (x,y') IS ALLOWED

& THERE IS AN EDGE FROM $(x,y) - (x',y)$ if $(x,x') \in G$ & (x',y) IS ALLOWED

ONCE WE GET THIS NEW GRAPH H, WHAT SHOULD WE DO?

ONCE WE GET THIS NEW GRAPH H, WHAT SHOULD WE DO?

BFS IN OUR NEW GRAPH.

Q: WHAT IS THE RUNNING TIME OF THE WHOLE ALGORITHM.

ONCE WE GET THIS NEW GRAPH H, WHAT SHOULD WE DO?

BFS IN OUR NEW GRAPH.

Q: WHAT IS THE RUNNING TIME OF THE WHOLE ALGORITHM.

1) FINDING ALLOWED CONFIGURATIONS

ONCE WE GET THIS NEW GRAPH H, WHAT SHOULD WE DO?

BFS IN OUR NEW GRAPH.

Q: WHAT IS THE RUNNING TIME OF THE WHOLE ALGORITHM.

1) FINDING ALLOWED CONFIGURATIONS

```
FOREACH v ∈ G
{
    FOREACH u ∈ G
    {
        If d(u,v) In G ≥ K
            H ← H ∪ {u,v}
    }
}
```

ONCE WE GET THIS NEW GRAPH H, WHAT SHOULD WE DO?

BFS IN OUR NEW GRAPH.

Q: WHAT IS THE RUNNING TIME OF THE WHOLE ALGORITHM.

1) FINDING ALLOWED CONFIGURATIONS

```
FOREACH v ∈ G
{
    FOREACH u ∈ G
    {
        If d(u,v) In G ≥ K
            H ← H ∪ {u,v}
    }
}
```

HOW DO YOU FIND THIS DISTANCE?

ONCE WE GET THIS NEW GRAPH H, WHAT SHOULD WE DO?

BFS IN OUR NEW GRAPH.

Q: WHAT IS THE RUNNING TIME OF THE WHOLE ALGORITHM.

1) FINDING ALLOWED CONFIGURATIONS

```
FOREACH v ∈ G
{
    FOREACH u ∈ G
    {
        If d(u,v) In G ≥ K
            H ← H ∪ {u,v}
    }
}
```

HOW DO YOU FIND THIS DISTANCE?

DO BFS FROM EACH $v \in G$ AND FIND LEVEL OF EACH VERTEX IN BFS-TREE(v).

ONCE WE GET THIS NEW GRAPH H, WHAT SHOULD WE DO?

BFS IN OUR NEW GRAPH.

Q: WHAT IS THE RUNNING TIME OF THE WHOLE ALGORITHM.

1) FINDING ALLOWED CONFIGURATIONS

```
FOREACH v ∈ G
{
    FOREACH u ∈ G
    {
        If d(u,v) In G ≥ K
            H ← H ∪ {u,v}
    }
}
```

HOW DO YOU FIND THIS DISTANCE?

DO BFS FROM EACH $v \in G$ AND FIND LEVEL OF EACH VERTEX IN BFS-TREE(v).

$$O(n \cdot (m+n)) = O(mn + n^2)$$

2) ADDING EDGES TO H.

2) ADDING EDGES TO H.

```
FOREACH (u,v) in H
{
    FOR EACH EDGE (u,u') ADJACENT TO u
                               IN G
    {   If (u',v) in H
        {
            ADD EDGE (u,v) — (u',v) IN
               THE ADJACENCY LIST OF (u,v)
        }
    }
    FOR EACH EDGE (v,v') ADJACENT TO v
                        IN G
    {
        If (u,v') in H
        {
            ADD EDGE (u,v) — (u,v') IN
               THE ADJACENCY LIST OF (u,v)
        }
    }
}
```

2) ADDING EDGES TO H.

```
FOREACH (u,v) in H
{
    FOR EACH EDGE (u,u') ADJACENT TO u
                              IN G
    {   If (u',v) in H
        {
            ADD EDGE (u,v) — (u',v) IN
               THE ADJACENCY LIST OF (u,v)
        }
    }
    FOR EACH EDGE (v,v') ADJACENT TO v
                     IN G
    {
        If (u,v') in H
        {
            ADD EDGE (u,v) — (u,v') IN
              THE ADJACENCY LIST OF (u,v)
        }
    }
}
```

RUNNING TIME :

2) ADDING EDGES TO H.

```
FOREACH (u,v) in H
{
         FOR EACH EDGE (u, u') ADJACENT TO u
                                       IN G
         {      If (u',v) in H
                {
d_G(u)                ADD EDGE (u,v) — (u',v) IN
                          THE ADJACENCY LIST OF (u,v)
                }
         }
         FOR EACH EDGE (v,v') ADJACENT TO v
                              IN G
         {
d_G(v)         If (u,v') in H
               {
                 ADD EDGE (u,v) — (u,v') IN
                    THE ADJACENCY LIST OF (u,v)
               }
         }
}
```

RUNNING TIME :

2) ADDING EDGES TO H.

```
FOREACH (u,v) in H
{
            FOR EACH EDGE (u,u') ADJACENT TO u
                                          IN G
            {      If (u',v) in H       } log n
  d_G(u)           {
           O(1) {  ADD EDGE (u,v) — (u',v) IN
                       THE ADJACENCY LIST OF (u,v)
                   }
            }
            FOR EACH EDGE (v,v') ADJACENT TO v
                                IN G
            {
  d_G(v)           If (u,v') in H       } O(log n)
                   {
           O(1) {  ADD EDGE (u,v) — (u,v') IN
                       THE ADJACENCY LIST OF (u,v)
                   }
            }
}
```

RUNNING TIME :

2) ADDING EDGES TO H.

FOREACH (u, v) in H
{
$d_G(u)$
FOR EACH EDGE (u, u') ADJACENT TO u IN G
{ IF (u', v) in H } $\log n$
{
$O(1)$ { ADD EDGE $(u, v) - (u', v)$ IN THE ADJACENCY LIST OF (u, v)
}
}
$d_G(v)$
FOR EACH EDGE (v, v') ADJACENT TO v IN G
{
IF (u, v') in H } $O(\log n)$
{
$O(1)$ { ADD EDGE (u, v) — (u, v') IN THE ADJACENCY LIST OF (u, v)
}
}
}

RUNNING TIME : $\sum_{u \in G} \sum_{v \in G} \left(d_G(u) + d_G(v)\right) \log n$

RUNNING TIME : $\sum_{u \in G} \sum_{v \in G} \left(d_G(u) + d_G(v) \right) \log n$

$= \log n \sum_{u \in V} \left(n \, d_G(u) + 2m \right)$

RUNNING TIME : $\sum_{u \in G} \sum_{v \in G} \left(d_G(u) + d_G(v)\right) \log n$

$$= \log n \sum_{u \in V} \left(n\, d_G(u) + 2m\right)$$

$$= \log n \left(2mn + 2mn\right)$$

$$= 4mn \log n$$

$$= O(mn \log n).$$

LAST STEP: DO A BFS IN H FROM THE SOURCE CONFIGURATION TO THE DESTINATION CONFIGURATION.

\# VERTICES IN H :

\# EDGES IN H :

LAST STEP: DO A BFS IN H FROM THE SOURCE CONFIGURATION TO THE DESTINATION CONFIGURATION.

\# VERTICES IN H : $O(n^2)$
\# EDGES IN H : $O(mn)$

$\Rightarrow$ RUNNING TIME = $O(mn + n^2)$

MAKING BFS : $O(mn+n^2)$
ADDING VERTICES : $O(n^2 \log n)$
ADDING EDGES : $O(mn \log n)$
DOING BFS IN H : $O(mn+n^2)$

TOTAL RUNNING TIME : $O(mn\log n + n^2 \log n)$.