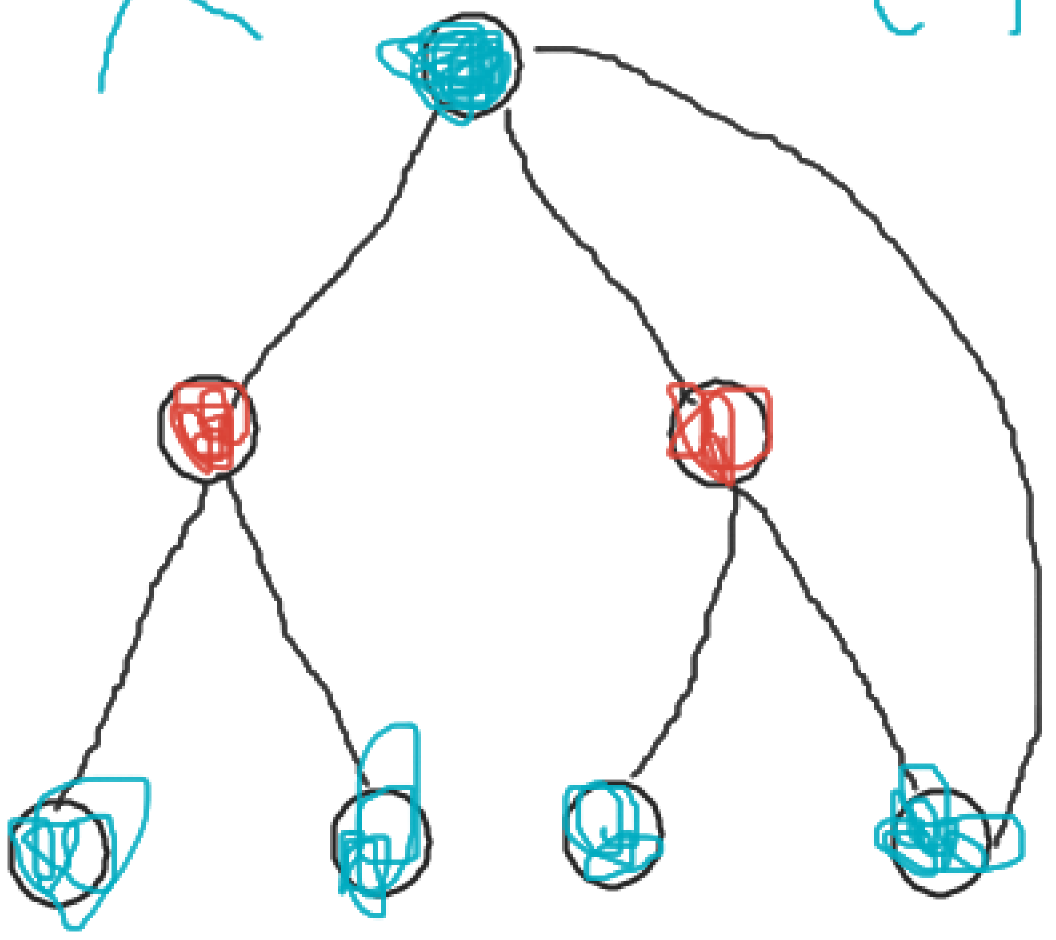
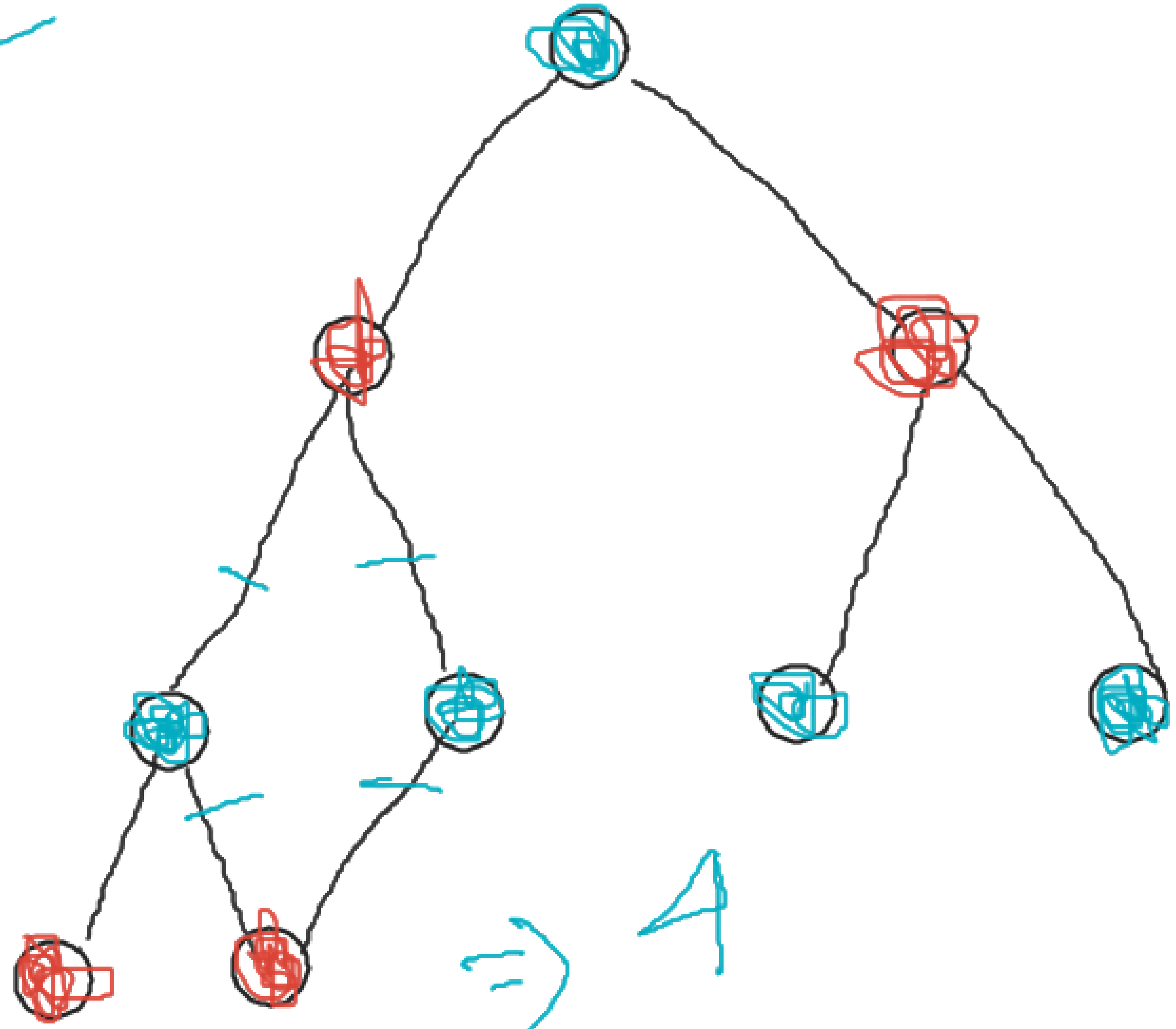
Bipartite Matching

Bicoloring

odd length cycle



⇒ 3



⇒ 4

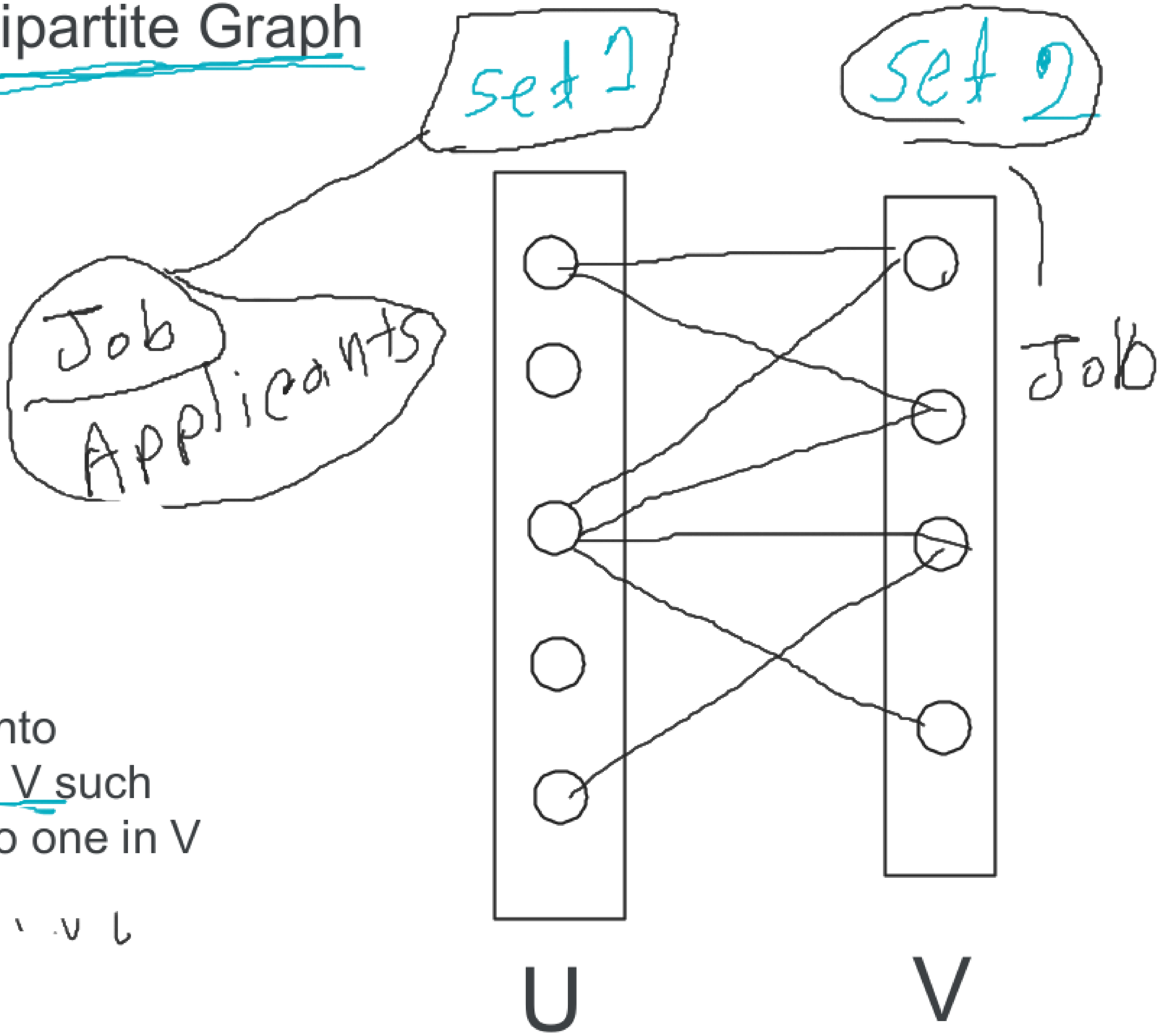
Bipartite

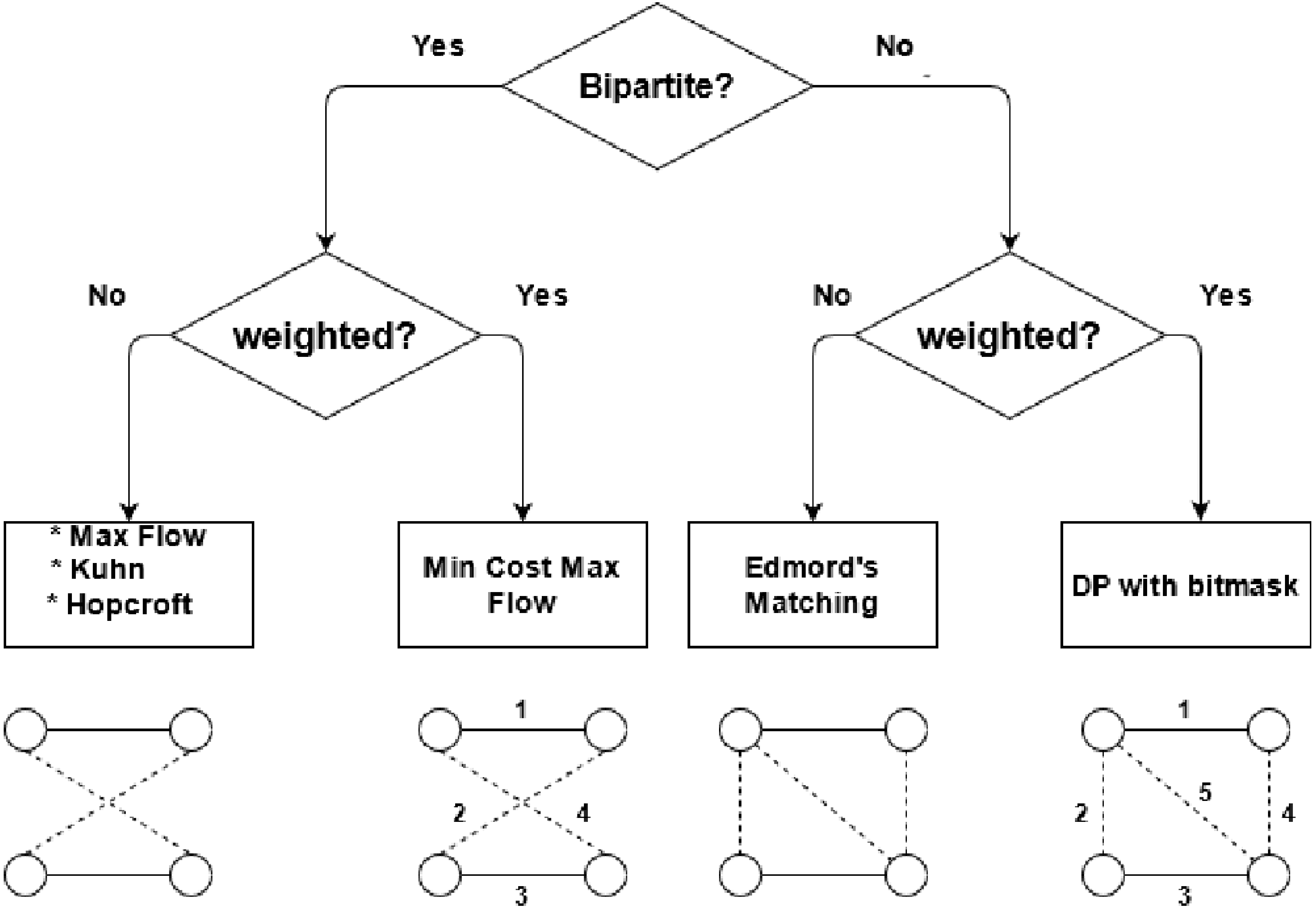
Bipartite Graph

#No odd length cycle

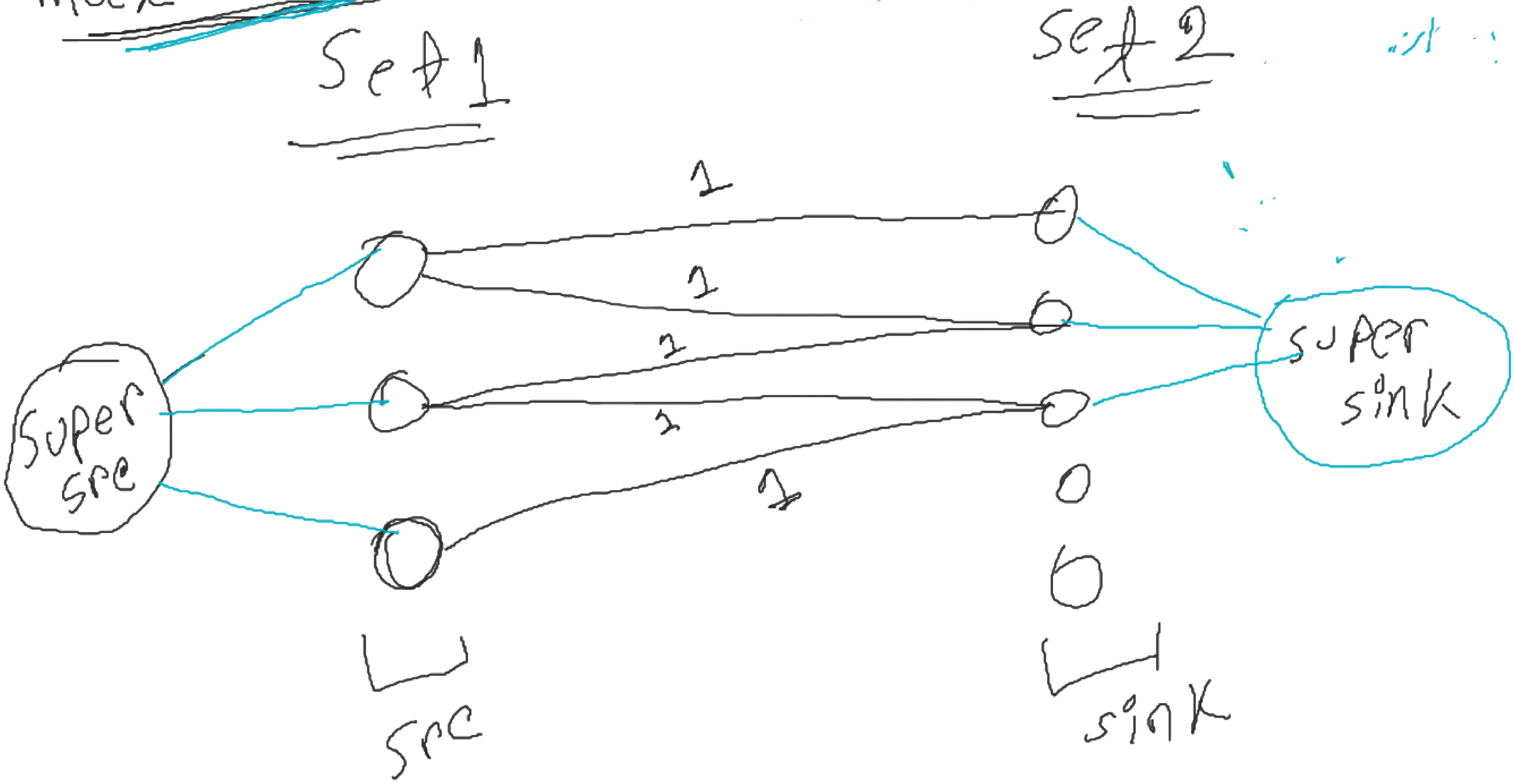
#Bicolorable

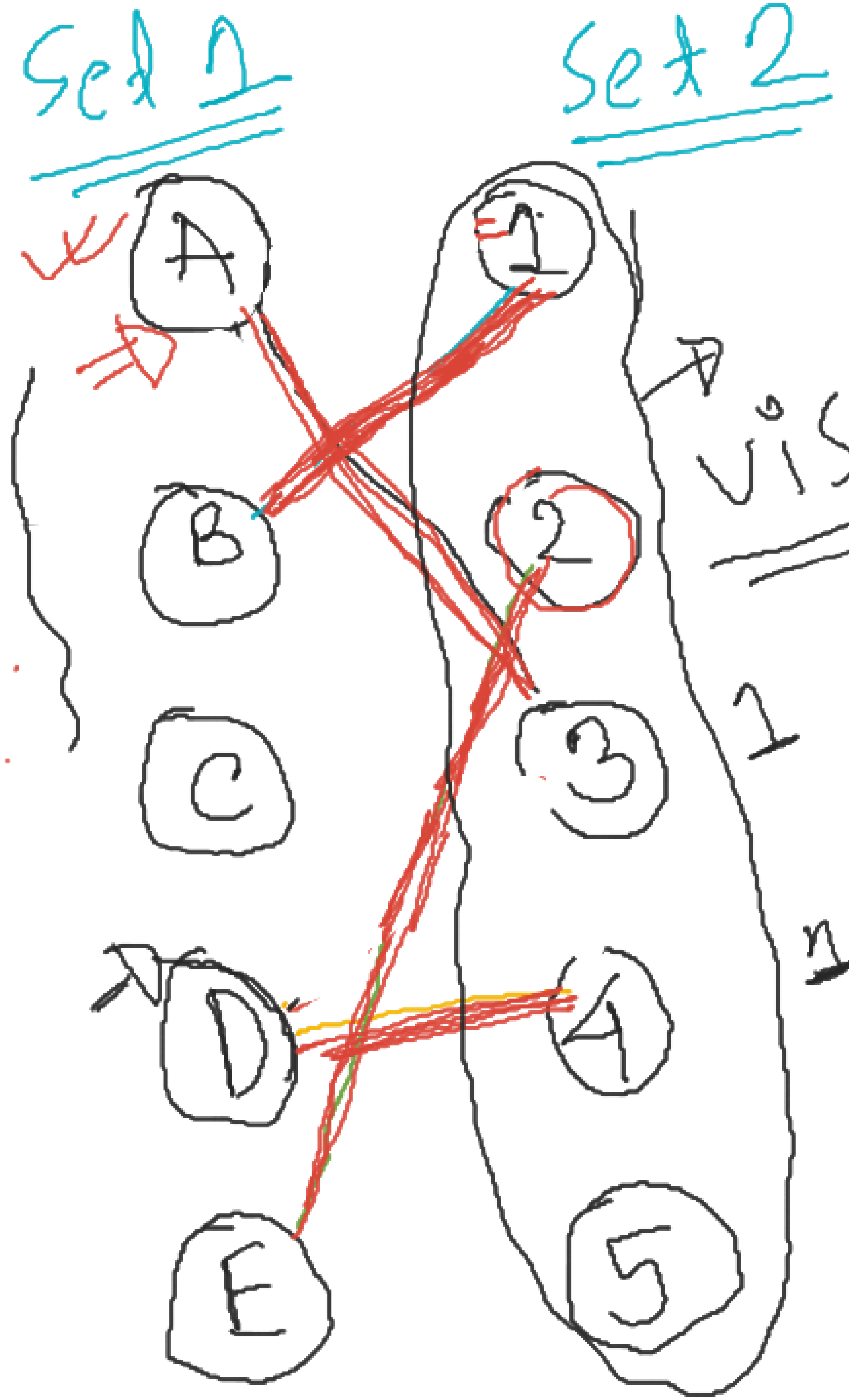
A graph whose vertices can be divided into two disjoint and independent sets U and V such that every edge connects a vertex in U to one in V




Bipartite?
Yes
No
weighted?
No
Yes
weighted?
No
Yes
* Max Flow
* Kuhn
* Hopcroft
Min Cost Max Flow
Edmord's Matching
DP with bitmask
1
2
4
3
1
5
2
4
3

Max flow





Red = Match

vis = Black

1 = prefer

D => 1 -> 5

A -> 3

B -> 1

D -> 4

Kuhn Algo

A -> 1, 2, 3, 4

B -> 1

D -> 3, 4

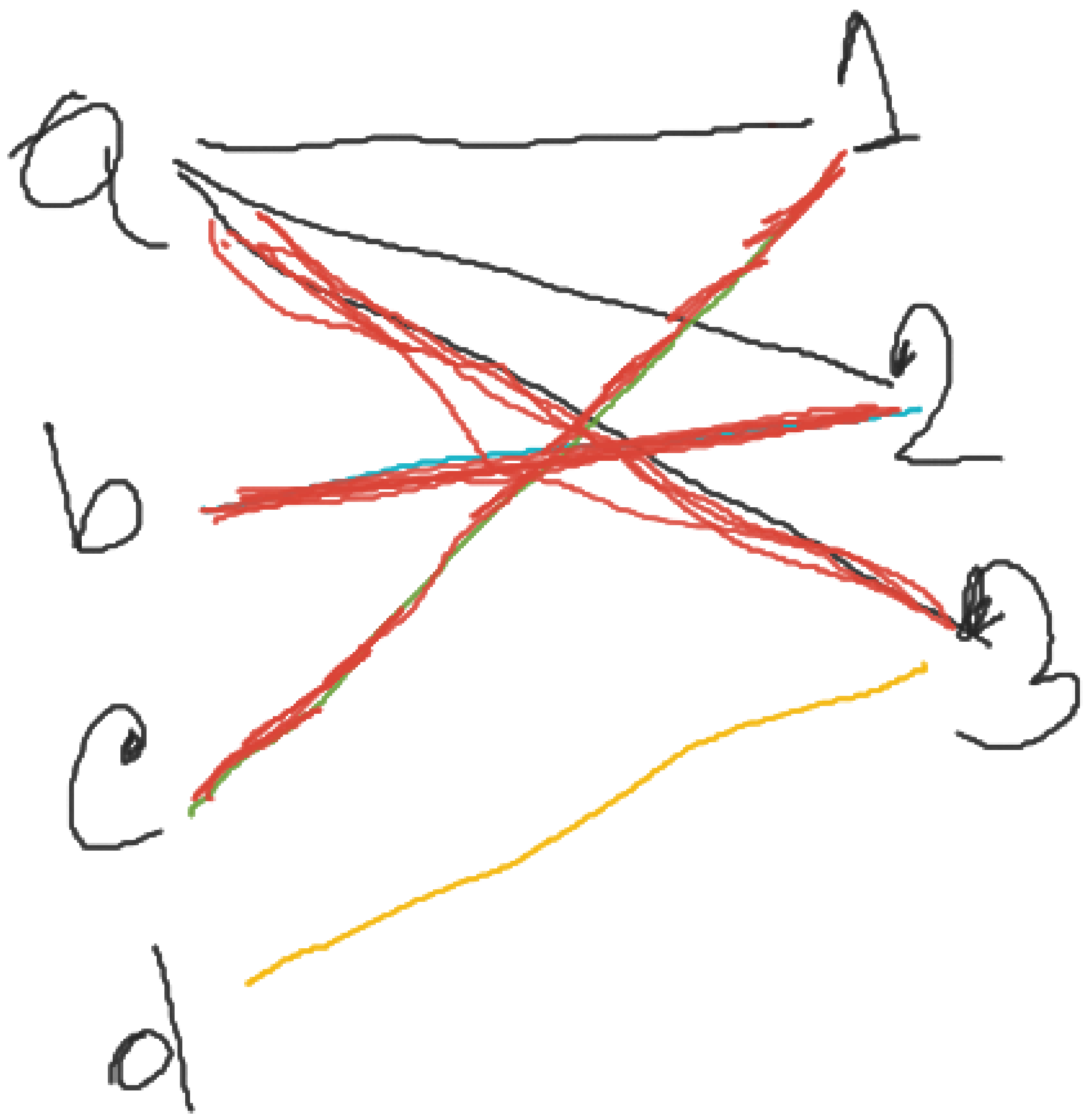
E -> 2

E -> 2

4 Gr

```
bool Kuhn(int u)
{
    for(int i = 0; i < edge[u].size(); i++)
    {
        int v = edge[u][i];
        if(vis[v])
            continue;
        vis[v] = 1;
        if(Right[v]==-1 || Kuhn(Right[v]))
        {
            Right[v] = u;
            Left[u] = v;
            return true;
        }
    }
    return false;
}
```

a -> 1 2 3
b -> 2
c -> 1
d -> 3



Max
Match
= 3

Time Complexity

VE

V => 500 => 1000

Kuhn use

```
bool Kuhn(int u)
{
    for(int i = 0; i < edge[u].size(); i++)
    {
        int v = edge[u][i];
        if(vis[v])
            continue;
        vis[v] = 1;
        if(Right[v]==-1 || Kuhn(Right[v]))
        {
            Right[v] = u;
            Left[u] = v;
            return true;
        }
    }
    return false;
}
```

V => 5000

hopcroft

```
int bpm(int node)
{
    memset(match,-1,sizeof m
    int cnt=0;
    //0 based index hole 0 to
    f(i,1,node)
    {
```

Hopcroft–Karp algorithm

BFS

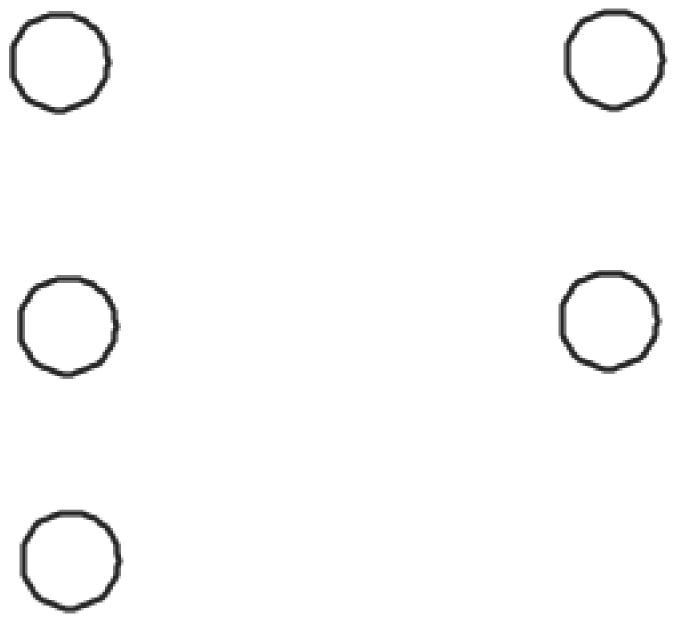
```
While(Augmenting Path)
{
    Update Matching;
}
```

DFS

Terms

#Free Vertex

#Matching and Not-Matching edges



#Augmenting Path

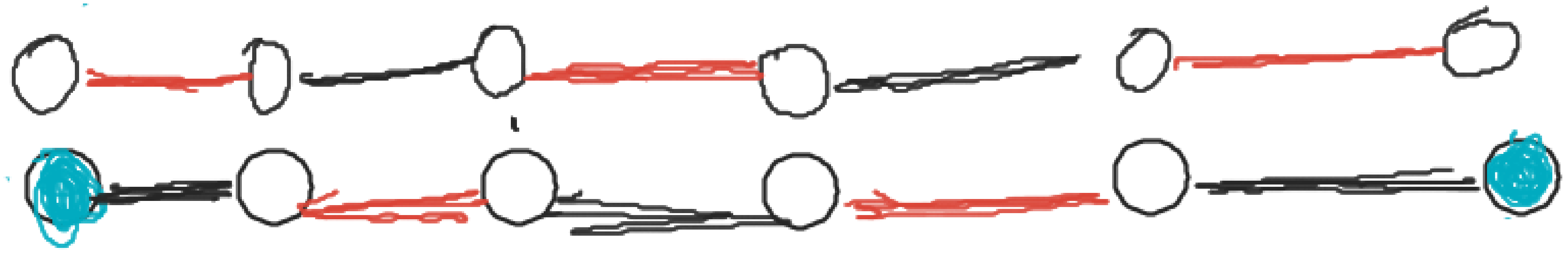
Starts and Ends at free vertex

Alternating Black and Red edges

Match += 1

Match => Red

Not match => Black



After update

Hopcroft-Karp(G):

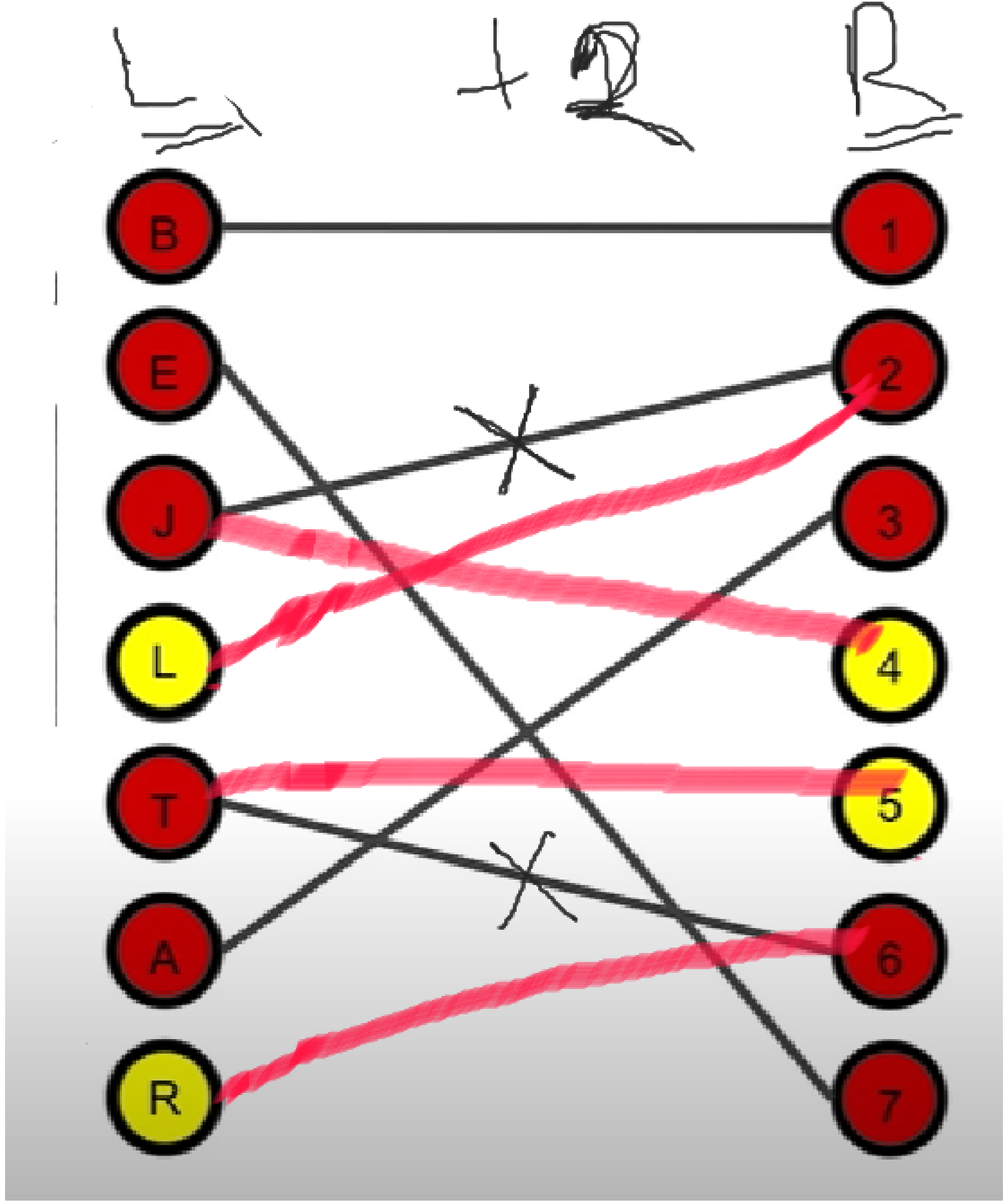
1. M = Ø
2. repeat
3. Use B.F.S. to build alternating level graph, rooted at unmatched vertices in Set A.
4. Augment current matching M with maximal set of vertex disjoint shortest-length paths (using D.F.S)
5. Until there are no more augmenting paths
6. return M

BFS

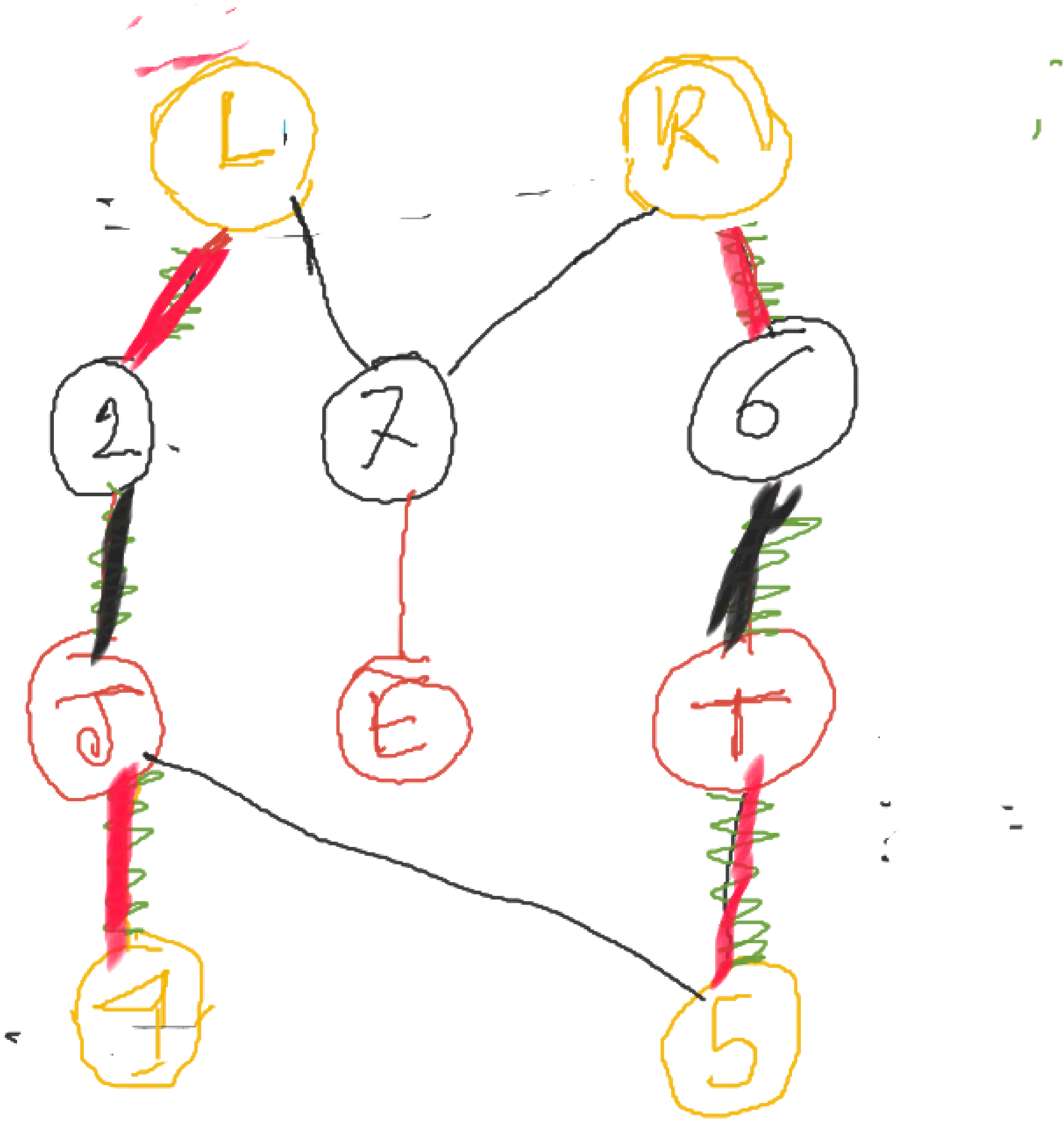
Yellow = free

Red = Match

B -> 1,4
E -> 7,3,6
J -> 2,4,5
L -> 2,7
T -> 7,6,5
A -> 3,6
R -> 6,7

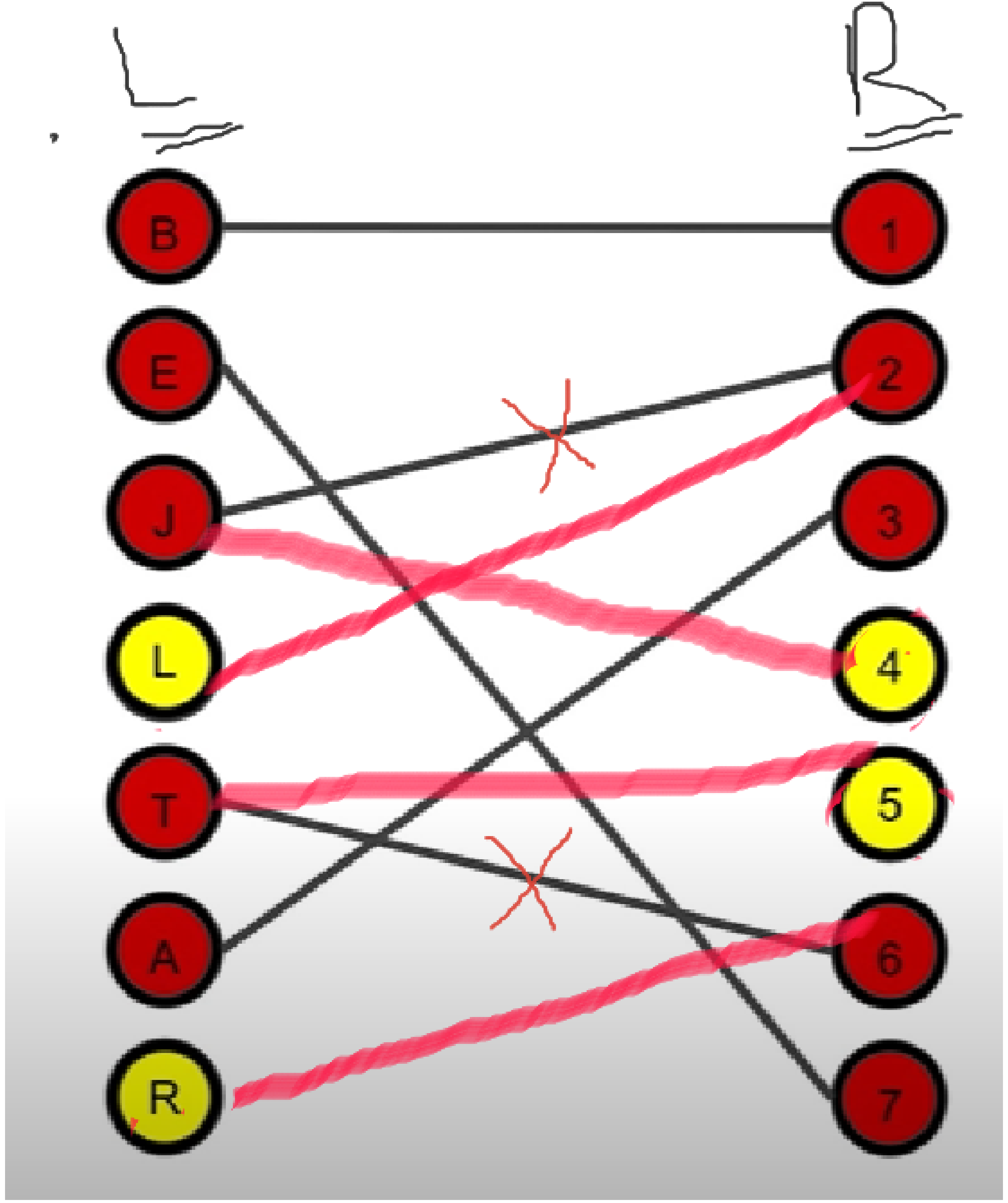


Current Matching



t = 2

+2



B -> 1,4
E -> 7,3,6
J -> 2,4,5
L -> 2,7
T -> 7,6,5
A -> 3,6
R -> 6,7

Current Matching

Time Complexity

Hopcroft-Karp(G):

1. M = Ø
2. repeat
3. Use B.F.S. to build alternating level graph, rooted at unmatched vertices in Set A.
4. Augment current matching M with maximal set of vertex disjoint shortest-length paths (using D.F.S)
5. Until there are no more augmenting paths
6. return M

O(1)

O(|E|)

O(√V)

Rough Idea

#Each phase increases the length of the shortest augmenting path by at least one

#After root(v) iterations , augment path lengths will be at least root(v)

(Total node = V) / (augment path length root(v)) = root(v)

#root(v) + root(v) = 2*root(v)