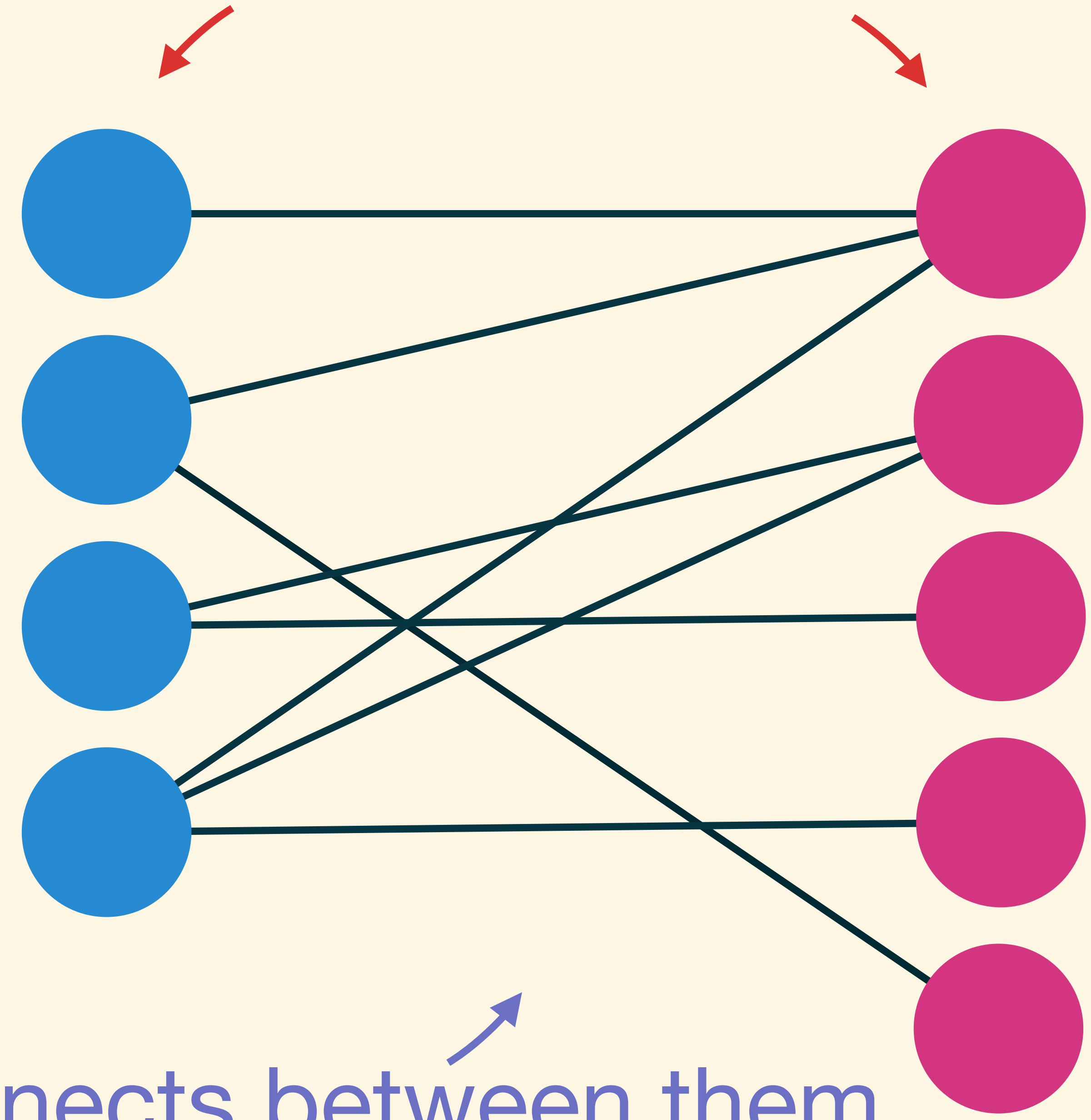
Bipartite Matching

郭至軒（KuoE0）

KuoE0.tw@gmail.com
KuoE0.ch

Bipartite Graph

two disjoint sets

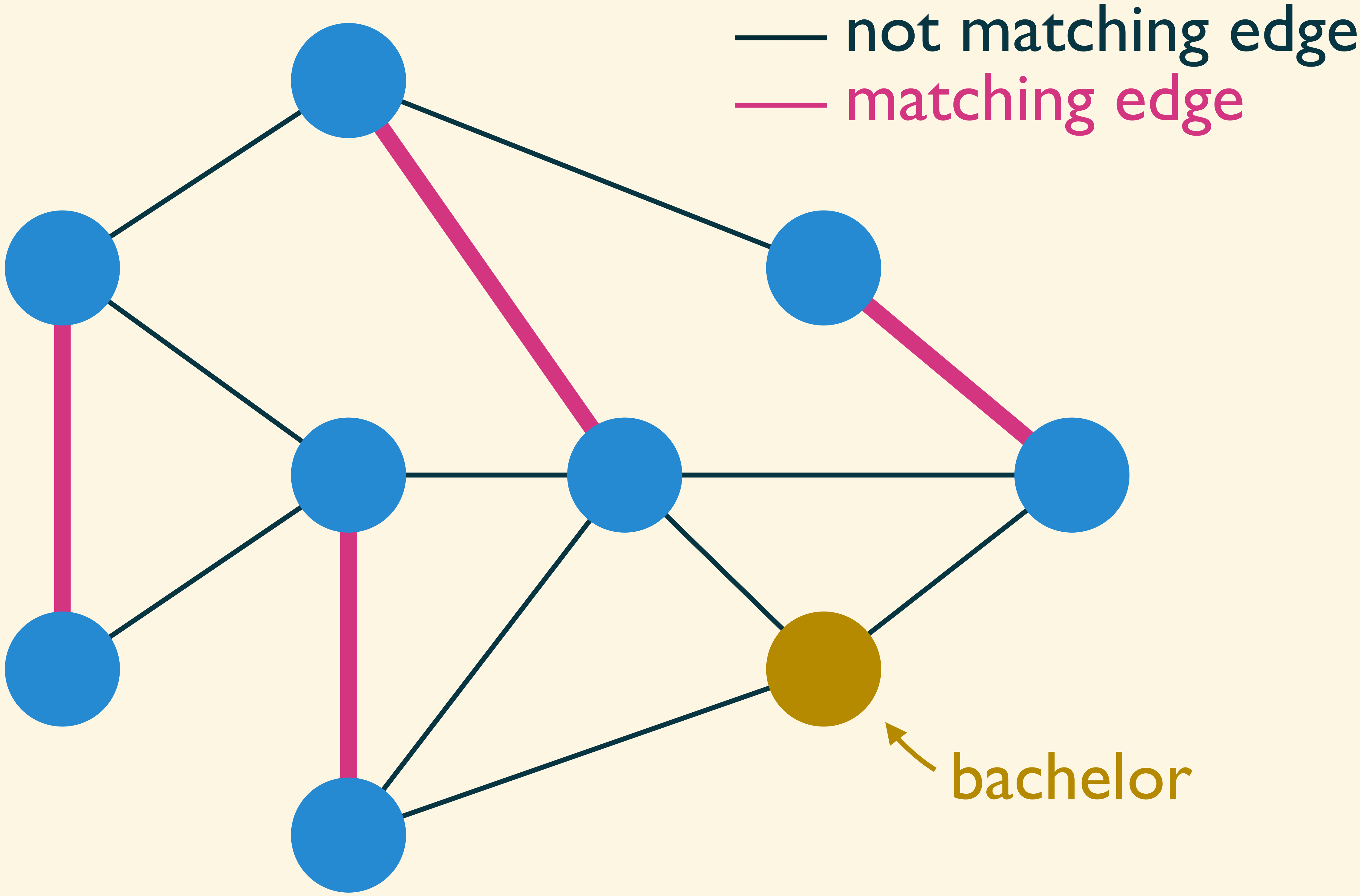


every edge connects between them

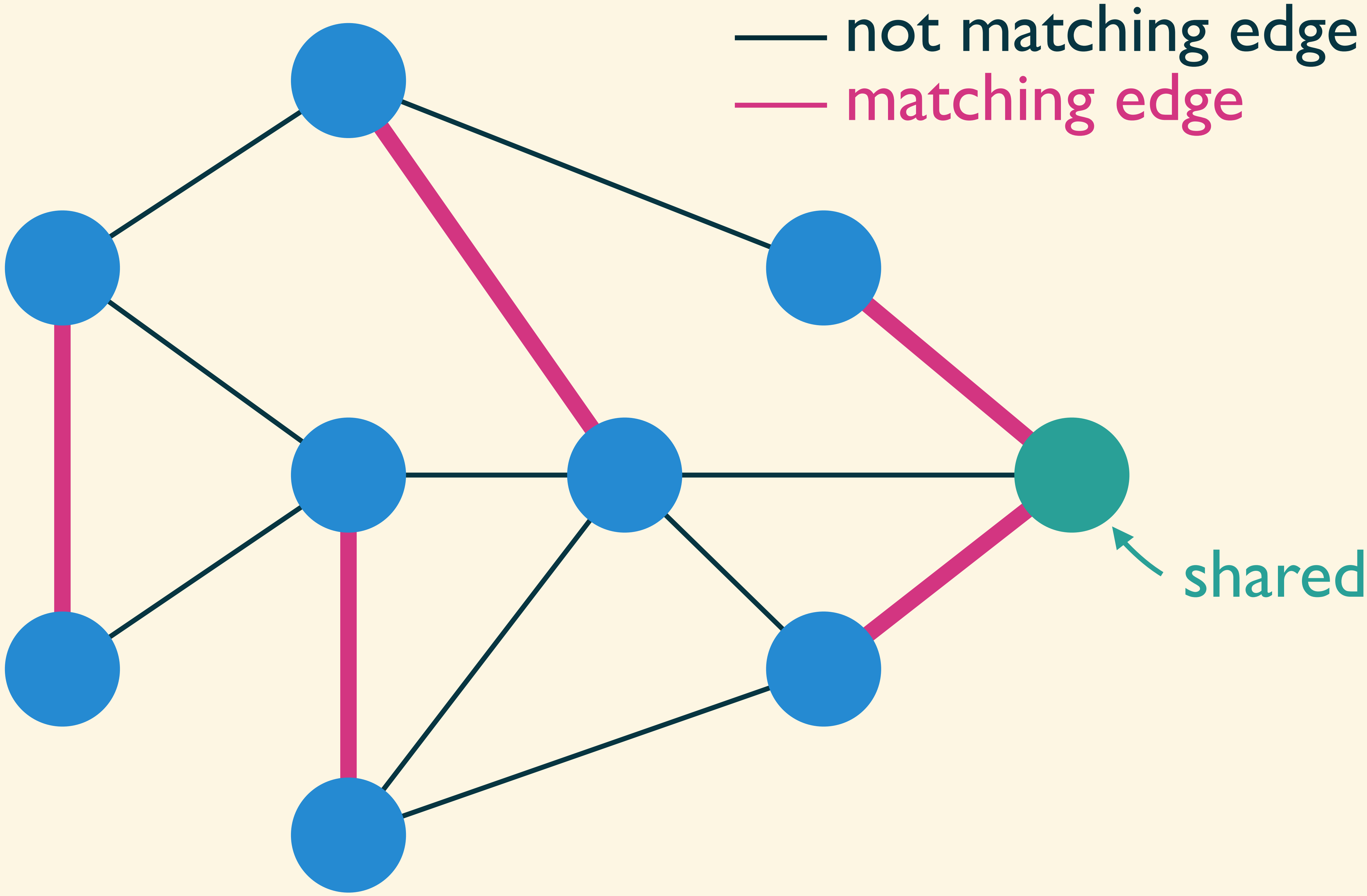
Matching

A subset of edges, **no two of which share an endpoint**.

Valid Matching

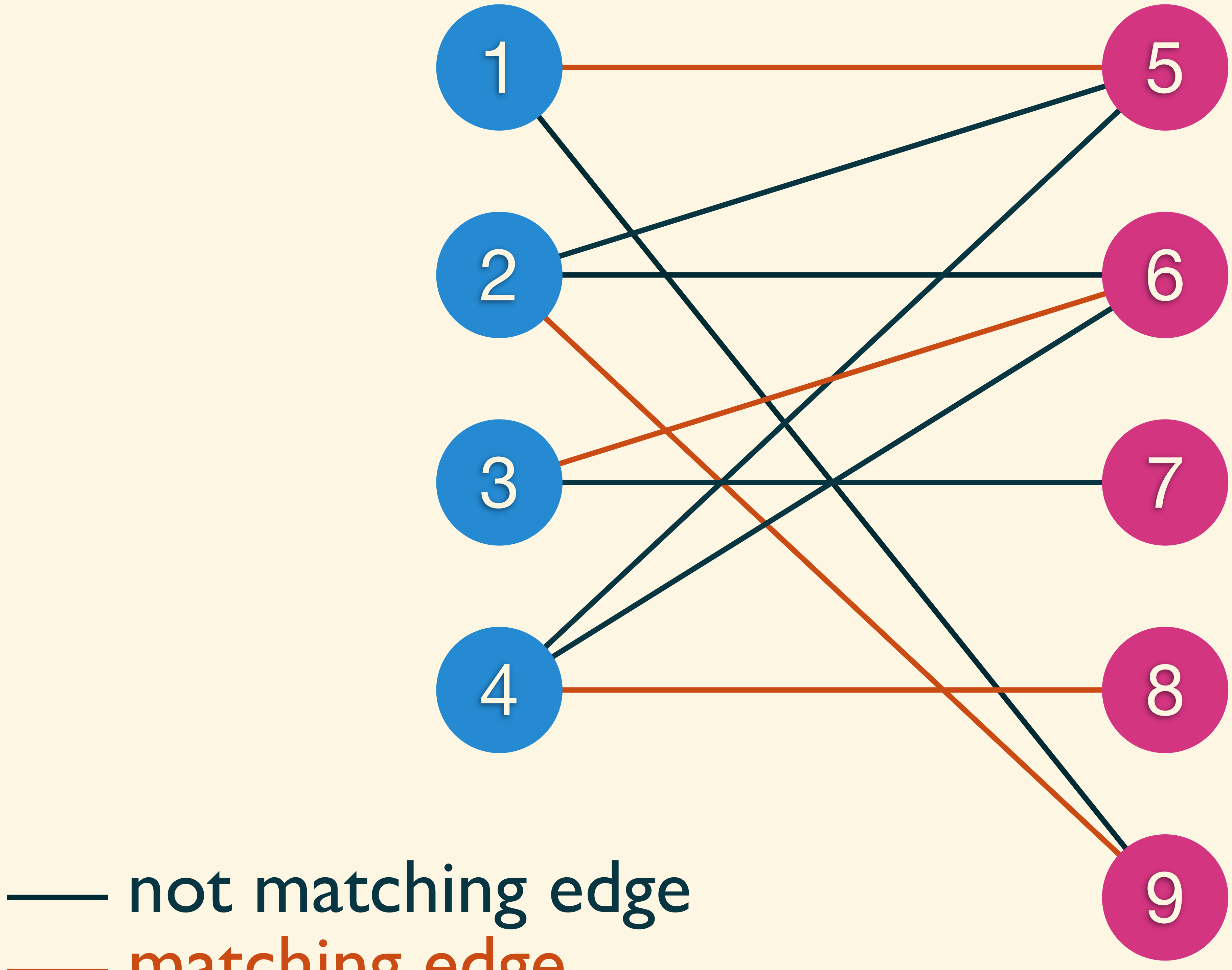


Invalid Matching



Bipartite Matching

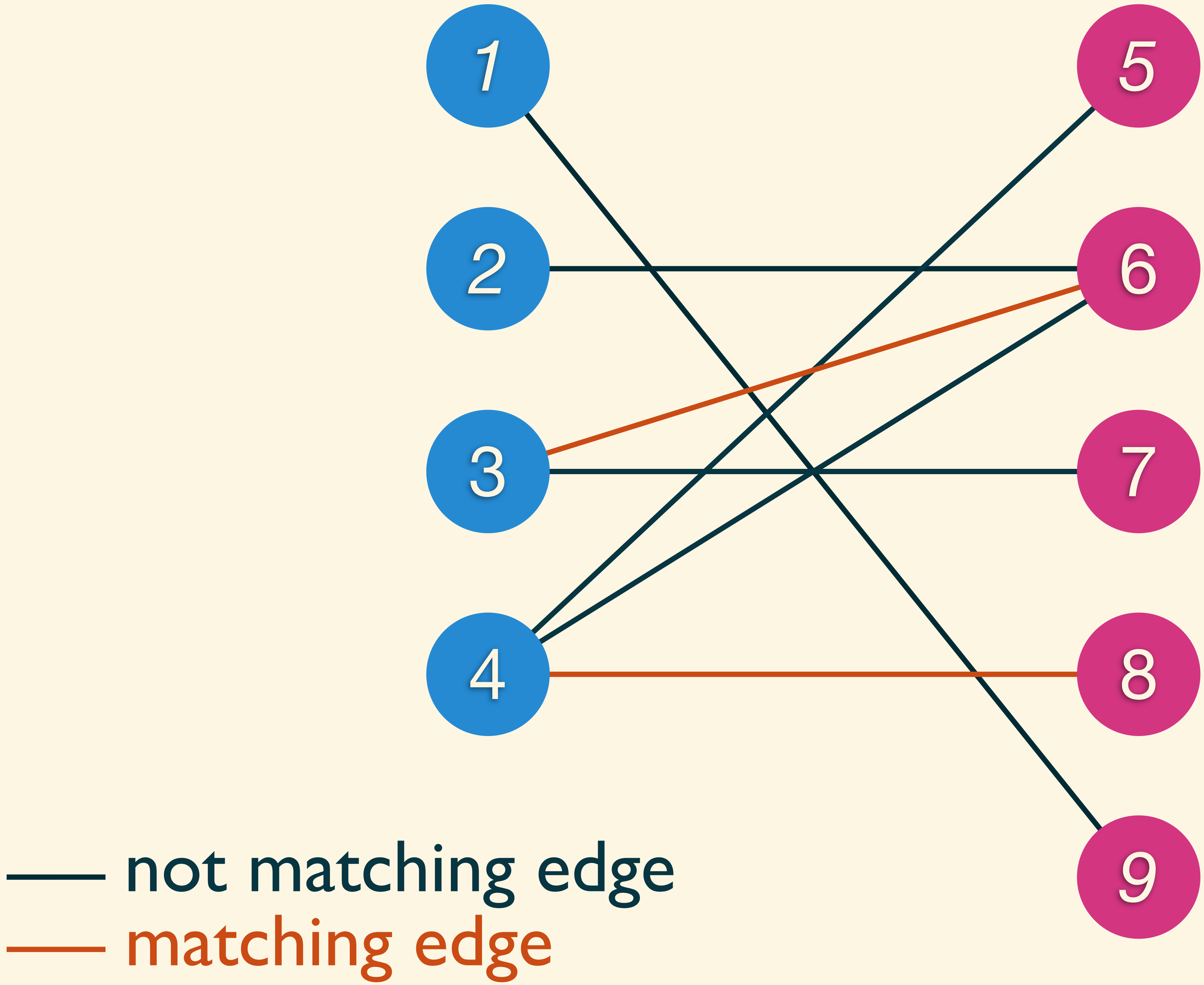
objective: **more matching**


1
2
3
4
5
6
7
8
9
not matching edge
matching edge

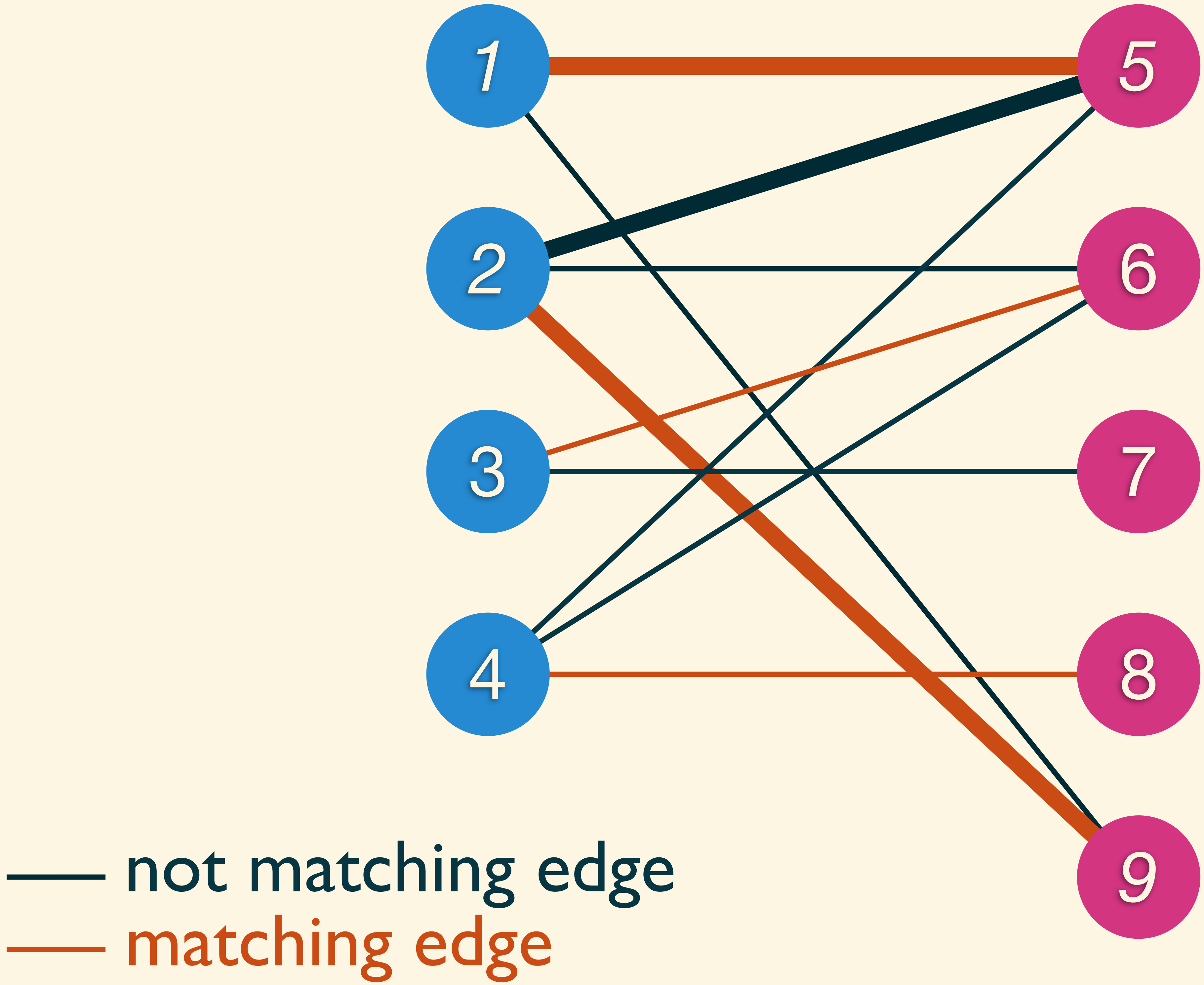
Alternating Path

A path in which the edges belong **alternatively to the matching and not to the matching**.

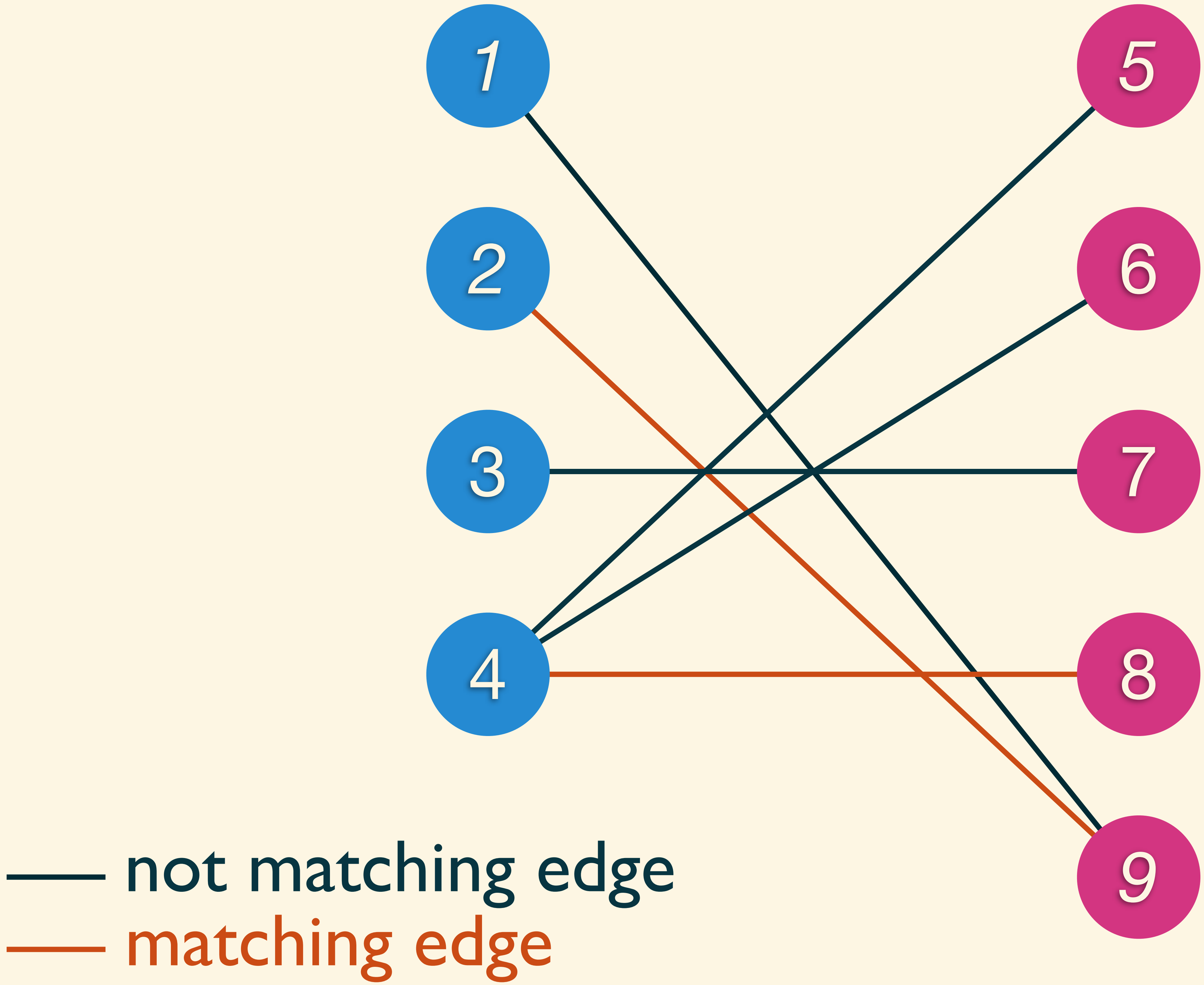
[1 - 5 - 2 - 9] is an alternating path.



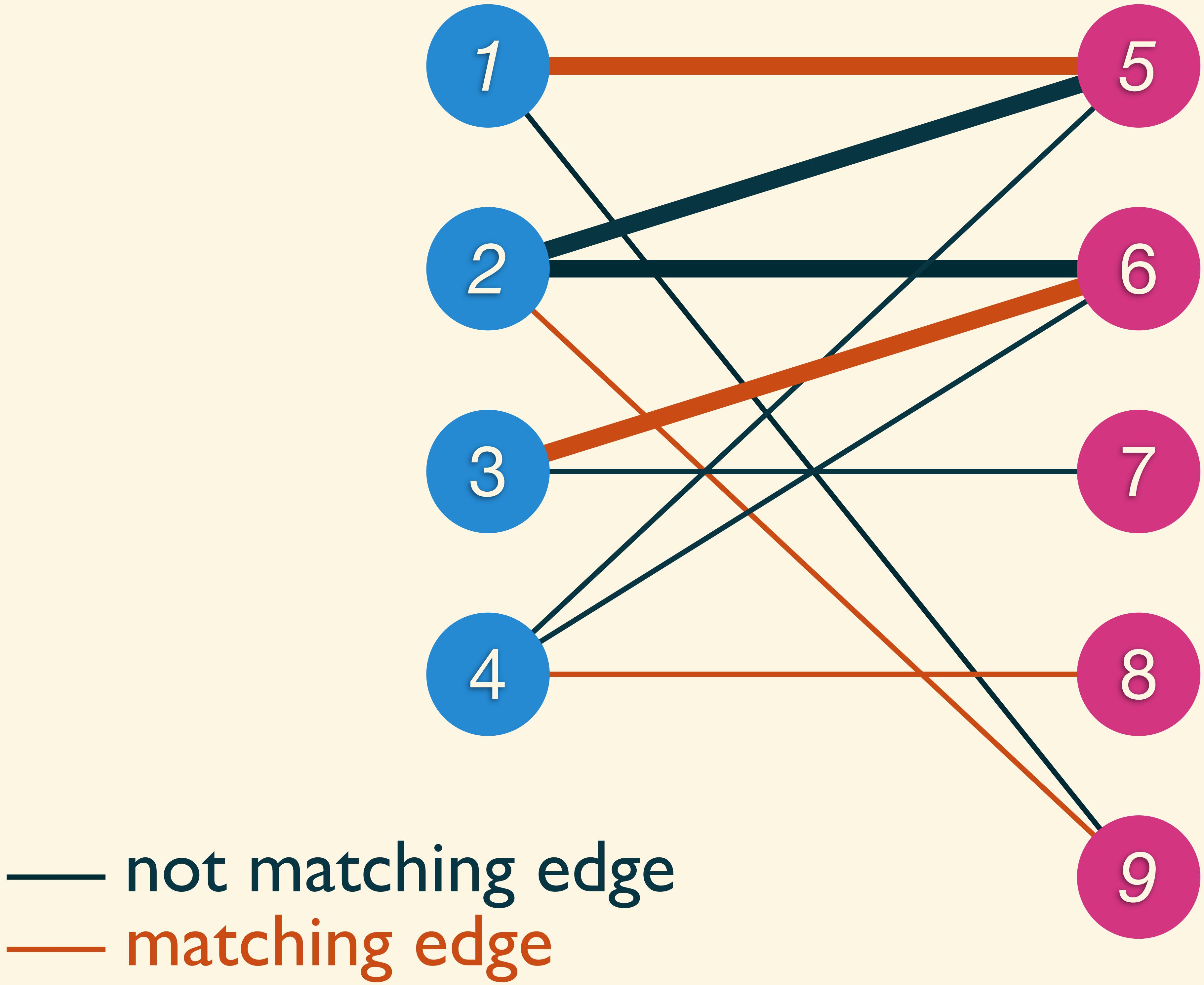
[1 - 5 - 2 - 9] is an alternating path.



[1 - 5 - 2 - 6 - 3] is **not** an alternating path.



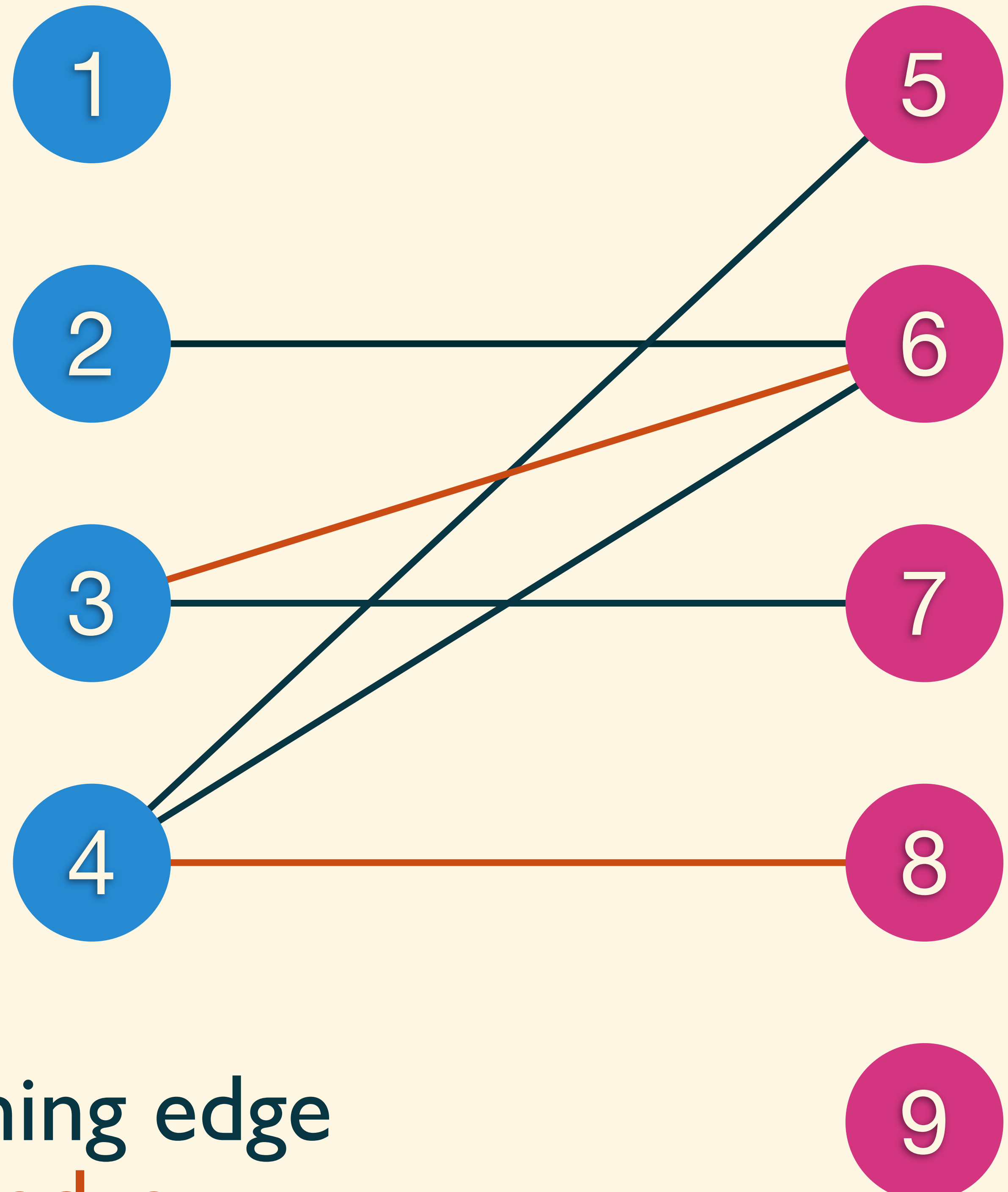
[1 - 5 - 2 - 6 - 3] is **not** an alternating path.



Alternating Cycle

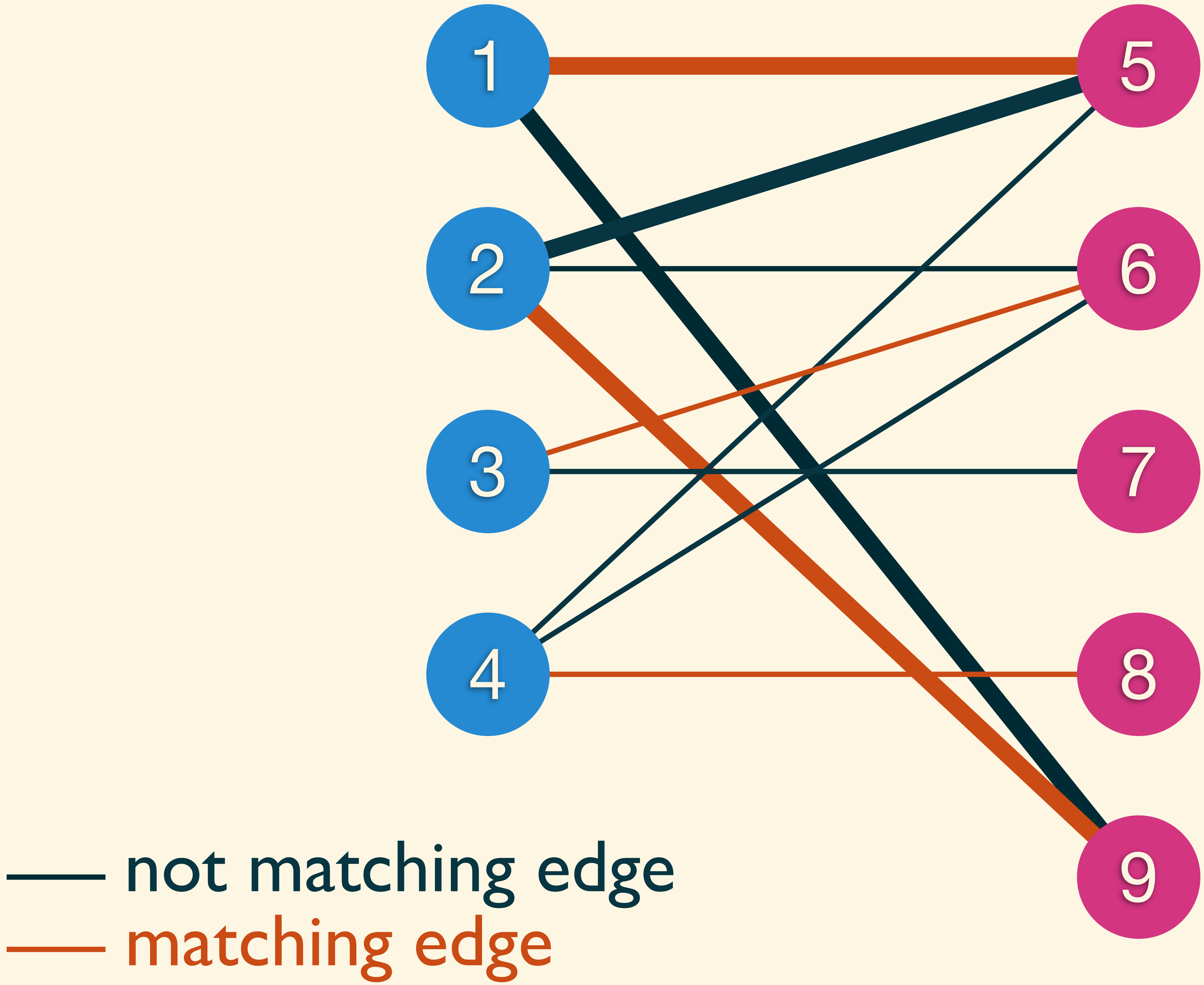
A cycle in which the edges belong **alternatively to the matching and not to the matching**.

[1 - 5 - 2 - 9 - 1] is an alternating cycle.

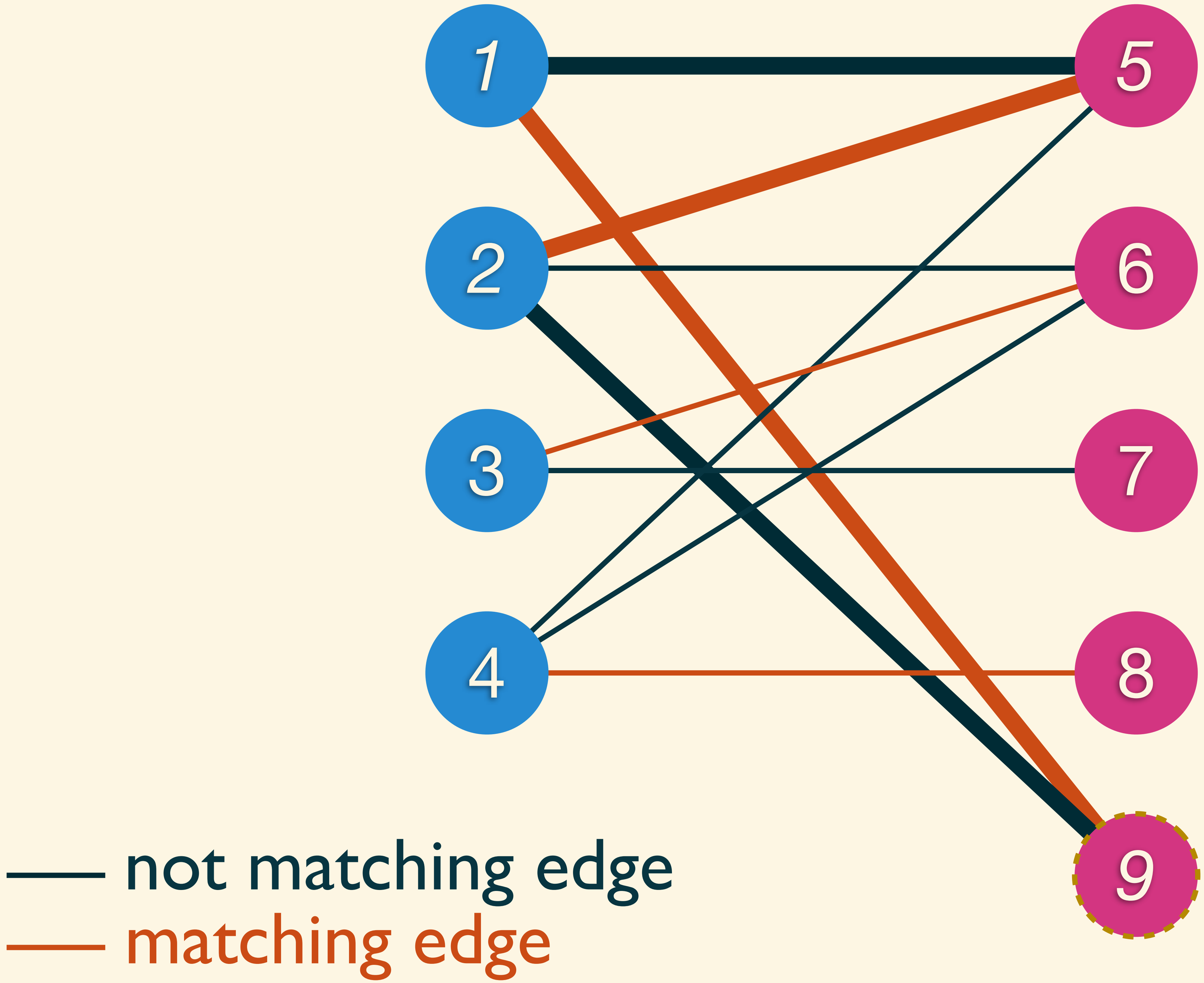


— not matching edge
— matching edge

[1 - 5 - 2 - 9 - 1] is an alternating cycle.



Reverse the alternating cycle, cardinality **won't change**.



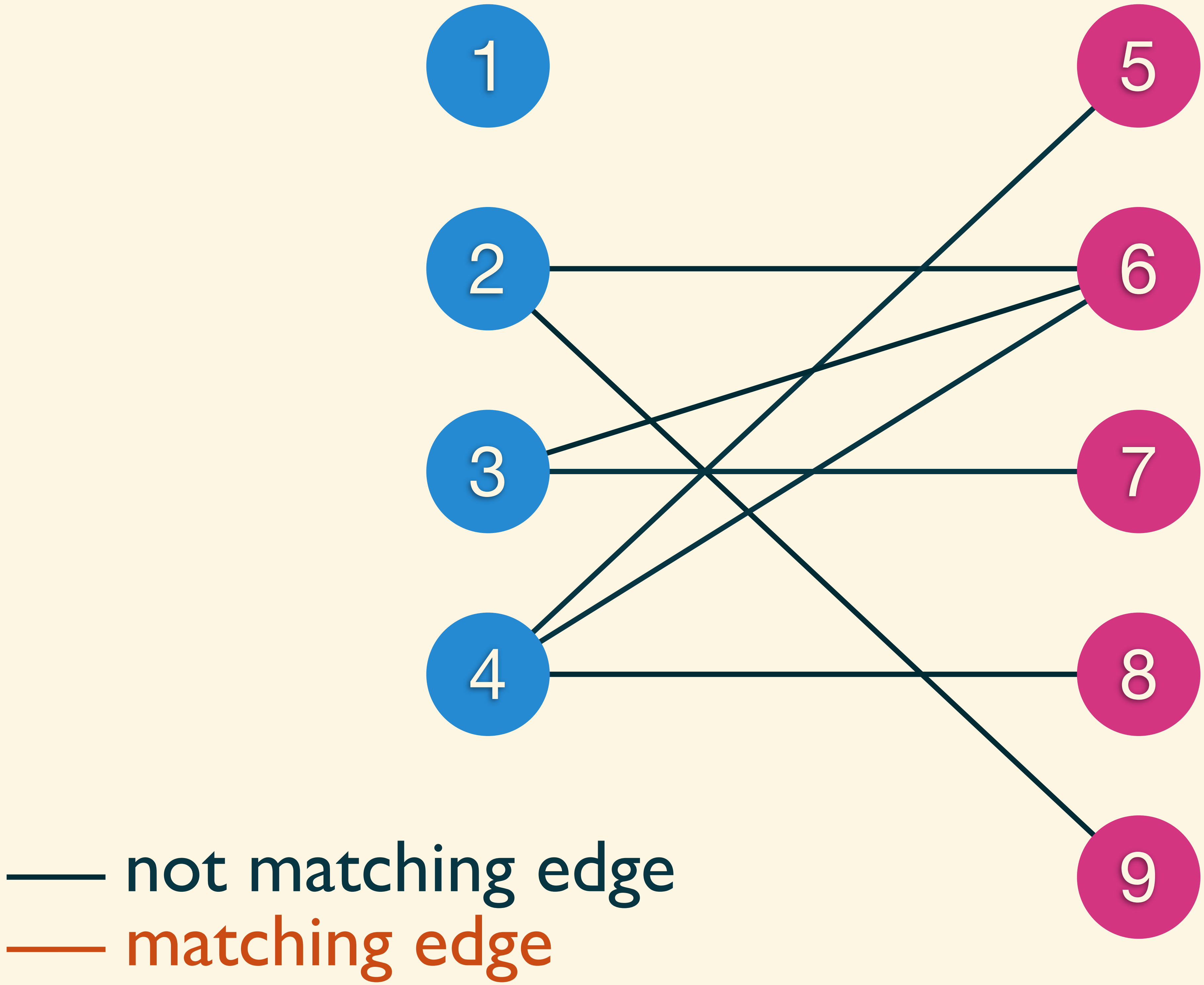
Maximum Matching

Augmenting Path Algorithm

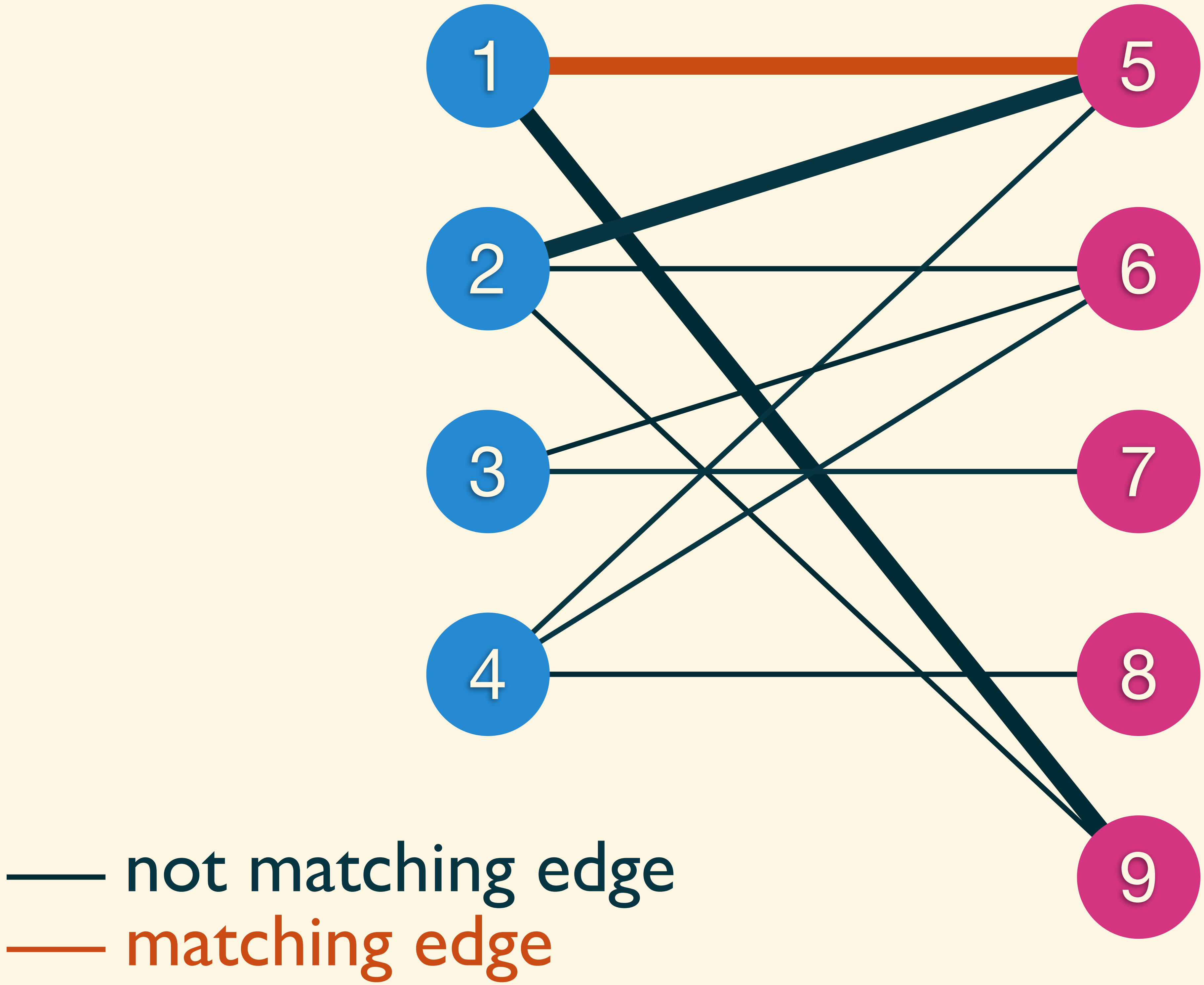
Augmenting Path

An **alternating path** that **starts from and ends on unmatched vertices**.

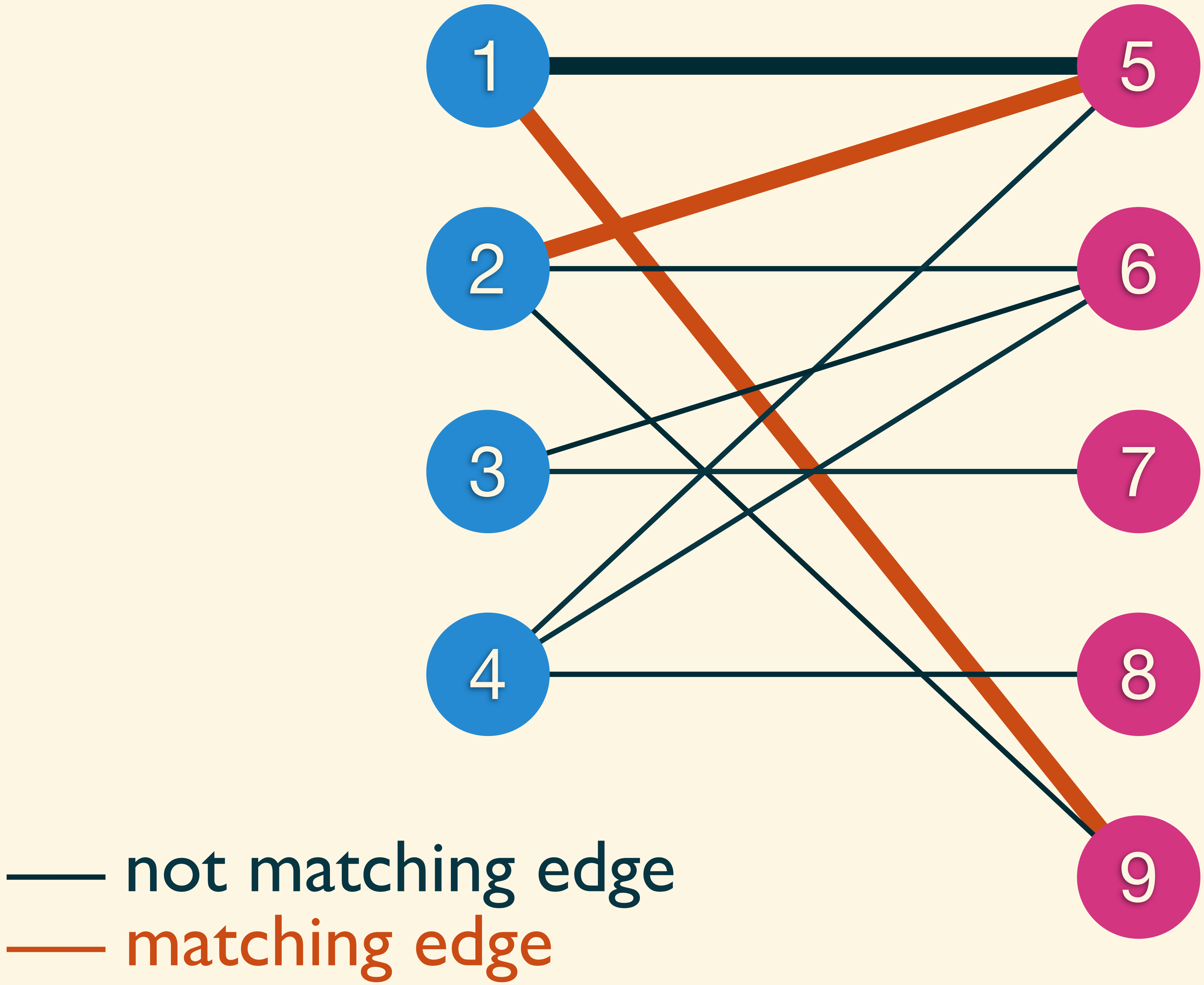
[2 - 5 - 1 - 9] is an augmenting path.



[2 - 5 - 1 - 9] is an augmenting path.



Reverse the augmenting path, cardinality **will increase**.



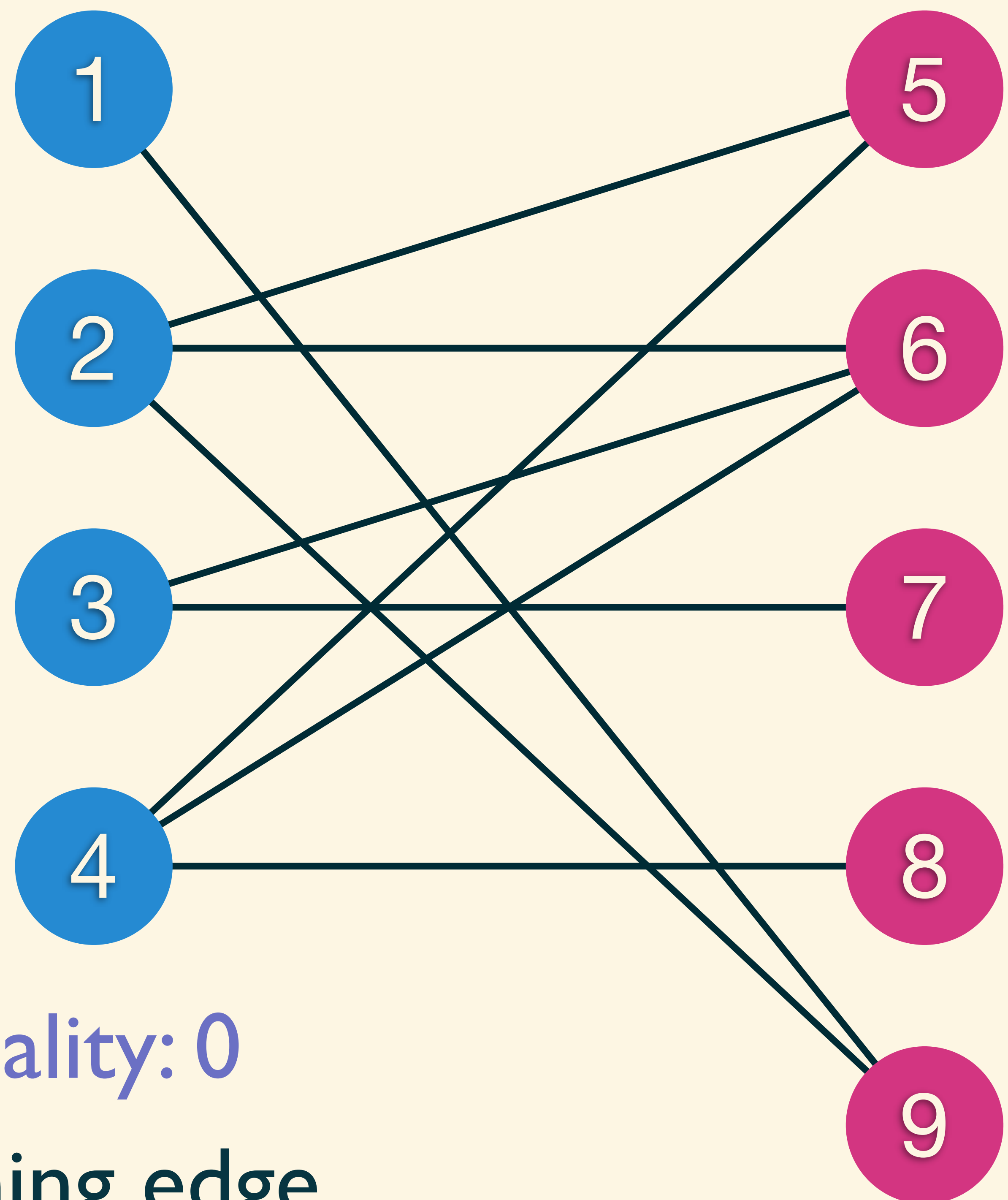
Augmenting Path

- The length of an augmenting path always is **odd**.
- Reversing an augmenting path will **increase the cardinality**.
- If there is **no augmenting path**, the **cardinality is maximum**.

Algorithm

1. Try to build augmenting paths from all vertices in one side.
2. Travel on graph.
3. If an augmenting path exists, reverse the augmenting path to increase cardinality.
4. If no augmenting path exists, ignore this vertex.
5. Repeat above step until there no augmenting path exists.

[1 - 5] is an augmenting path.

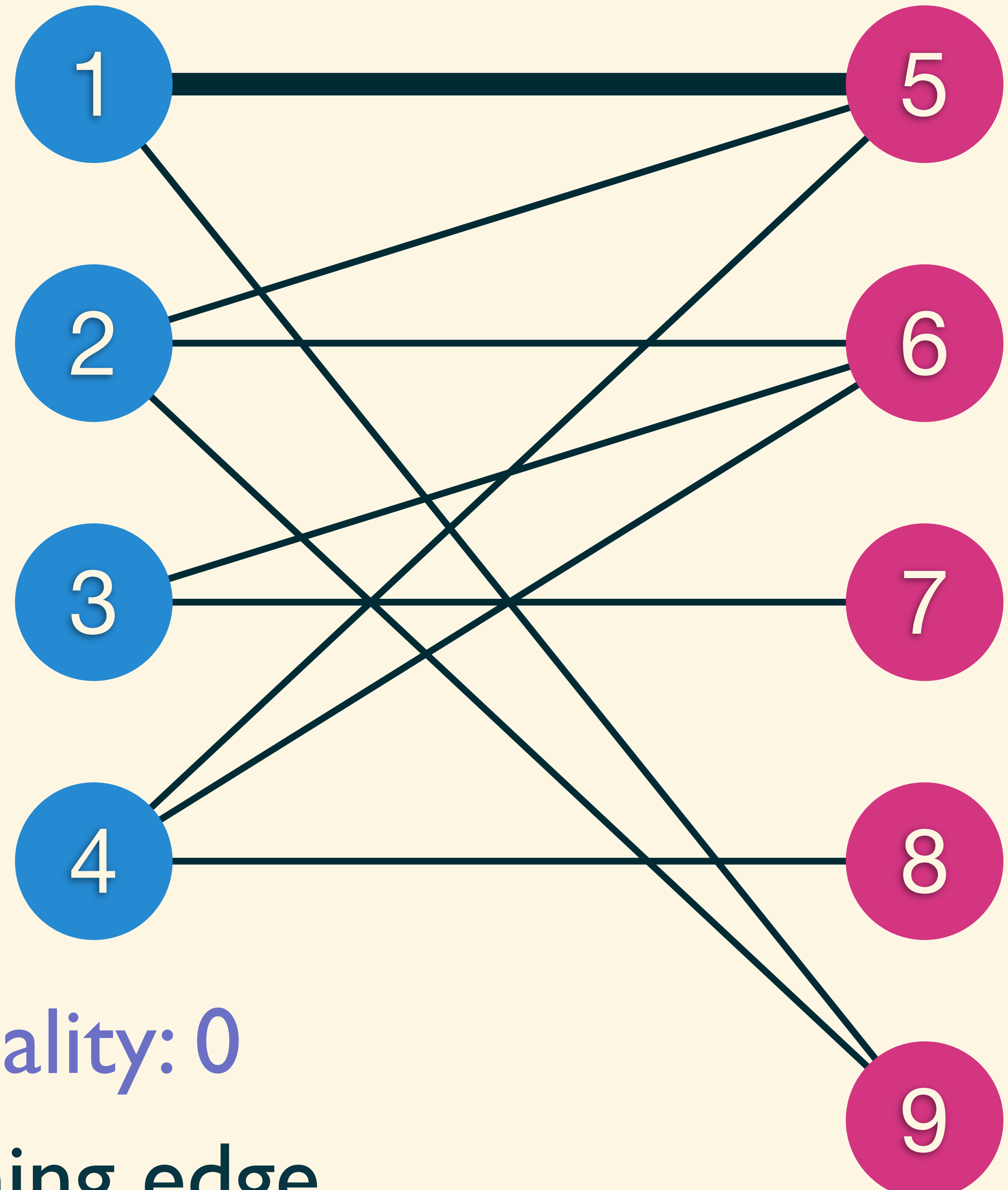


current cardinality: 0

— not matching edge

— matching edge

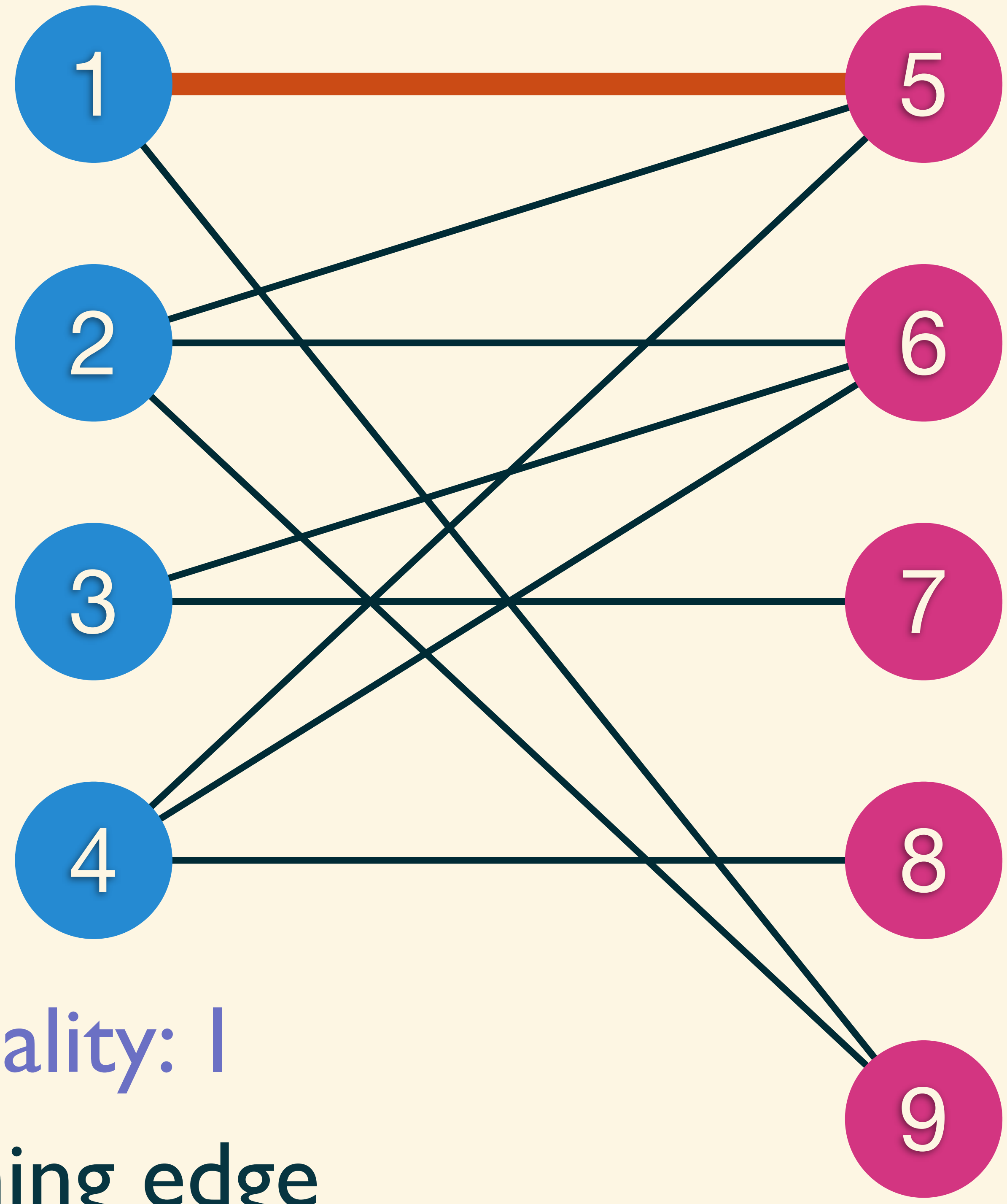
[1 - 5] is an augmenting path.



current cardinality: 0

— not matching edge
— matching edge

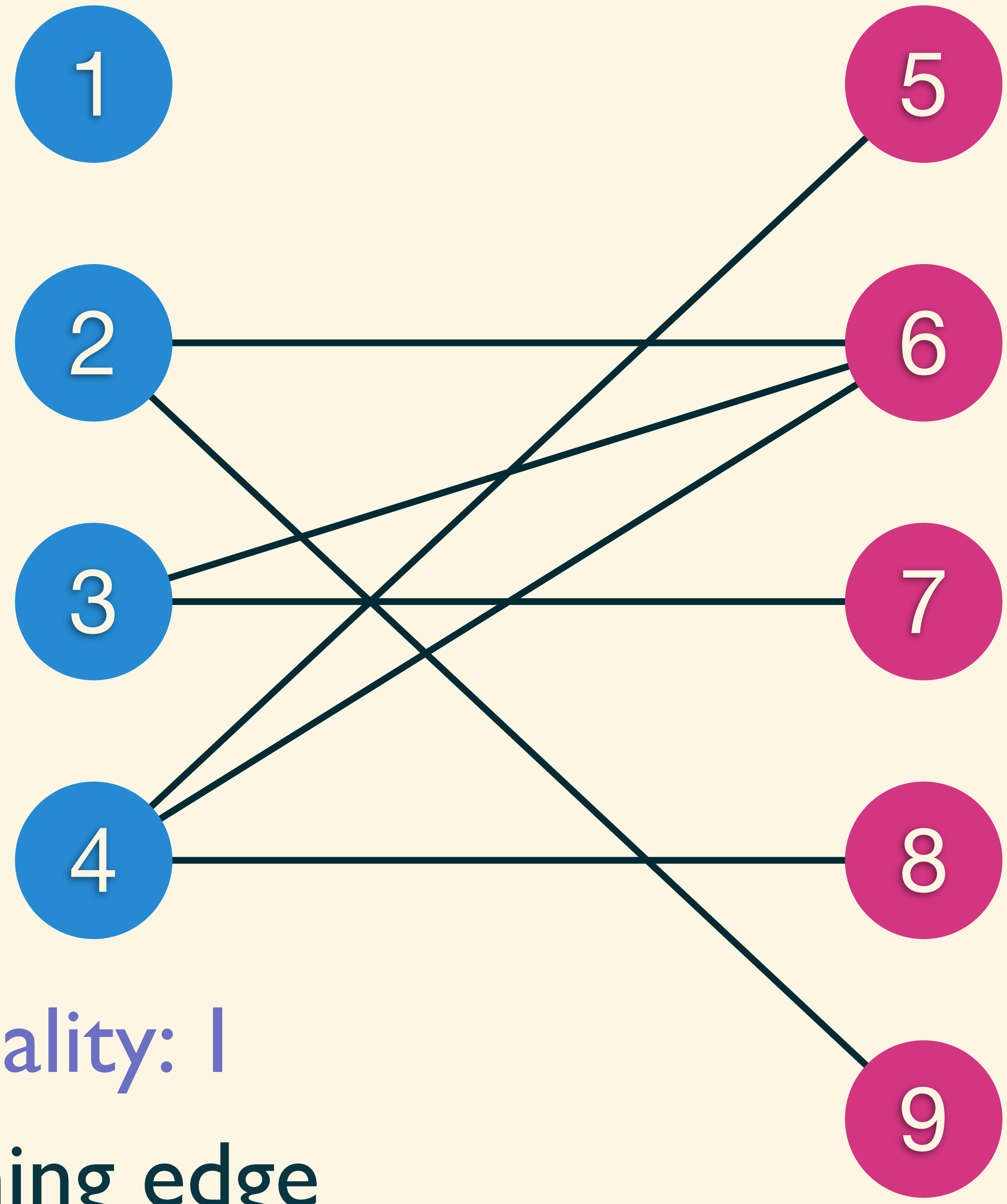
Reverse it!



current cardinality: 1

— not matching edge
— matching edge

[2 - 5 - 1- 9] is an augmenting path.

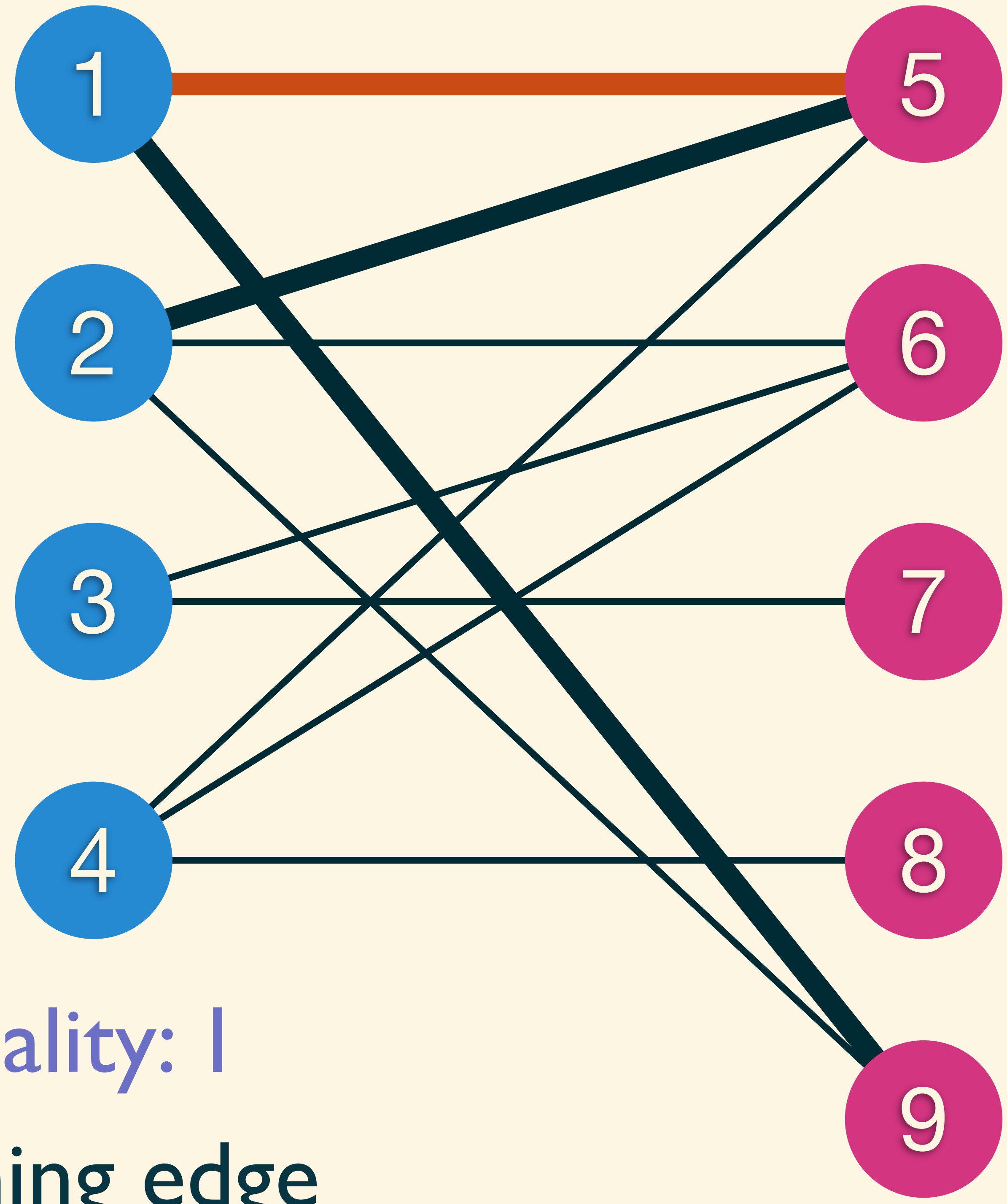


current cardinality: 1

— not matching edge

— matching edge

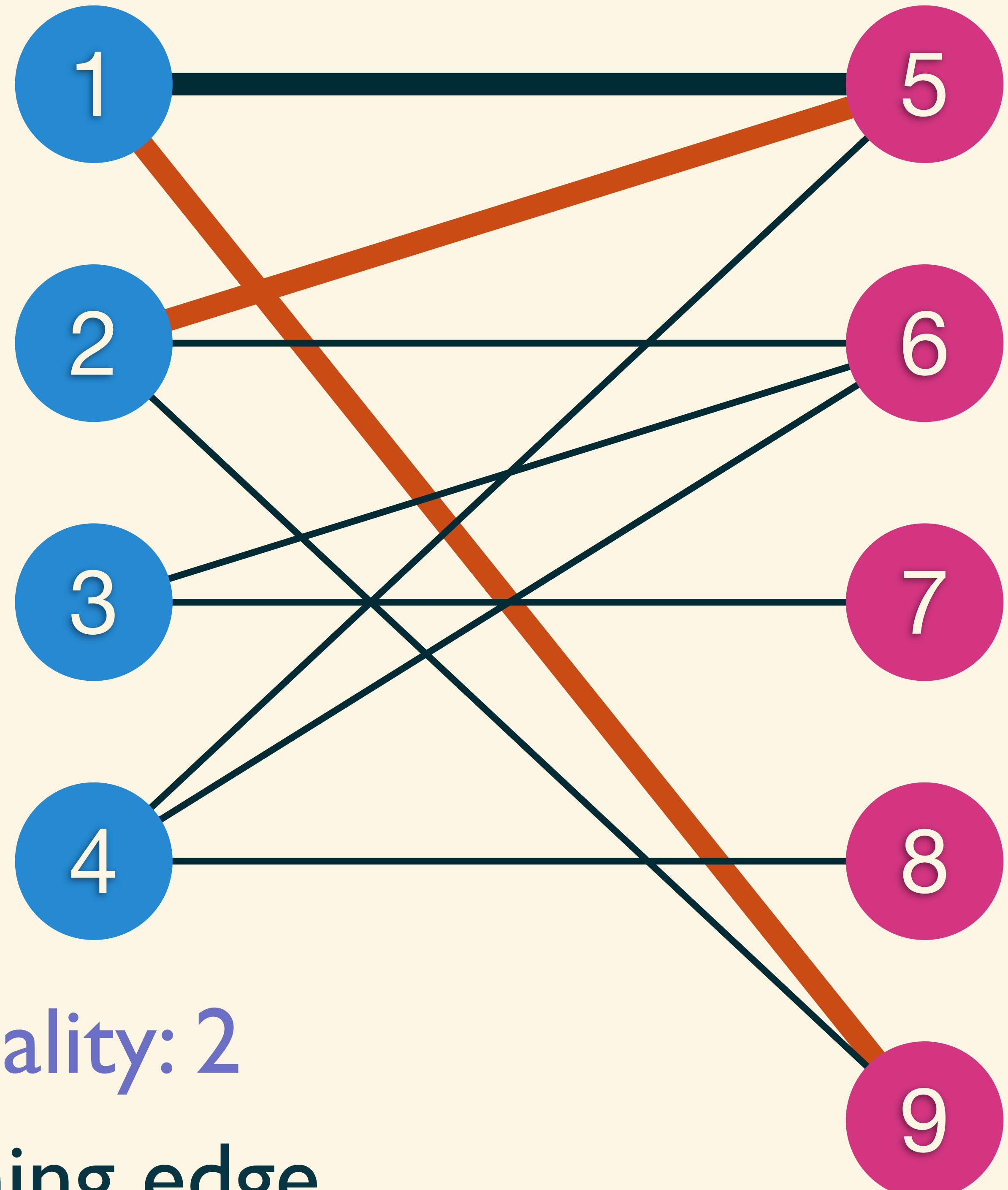
[2 - 5 - 1- 9] is an augmenting path.



current cardinality: 1

— not matching edge
— matching edge

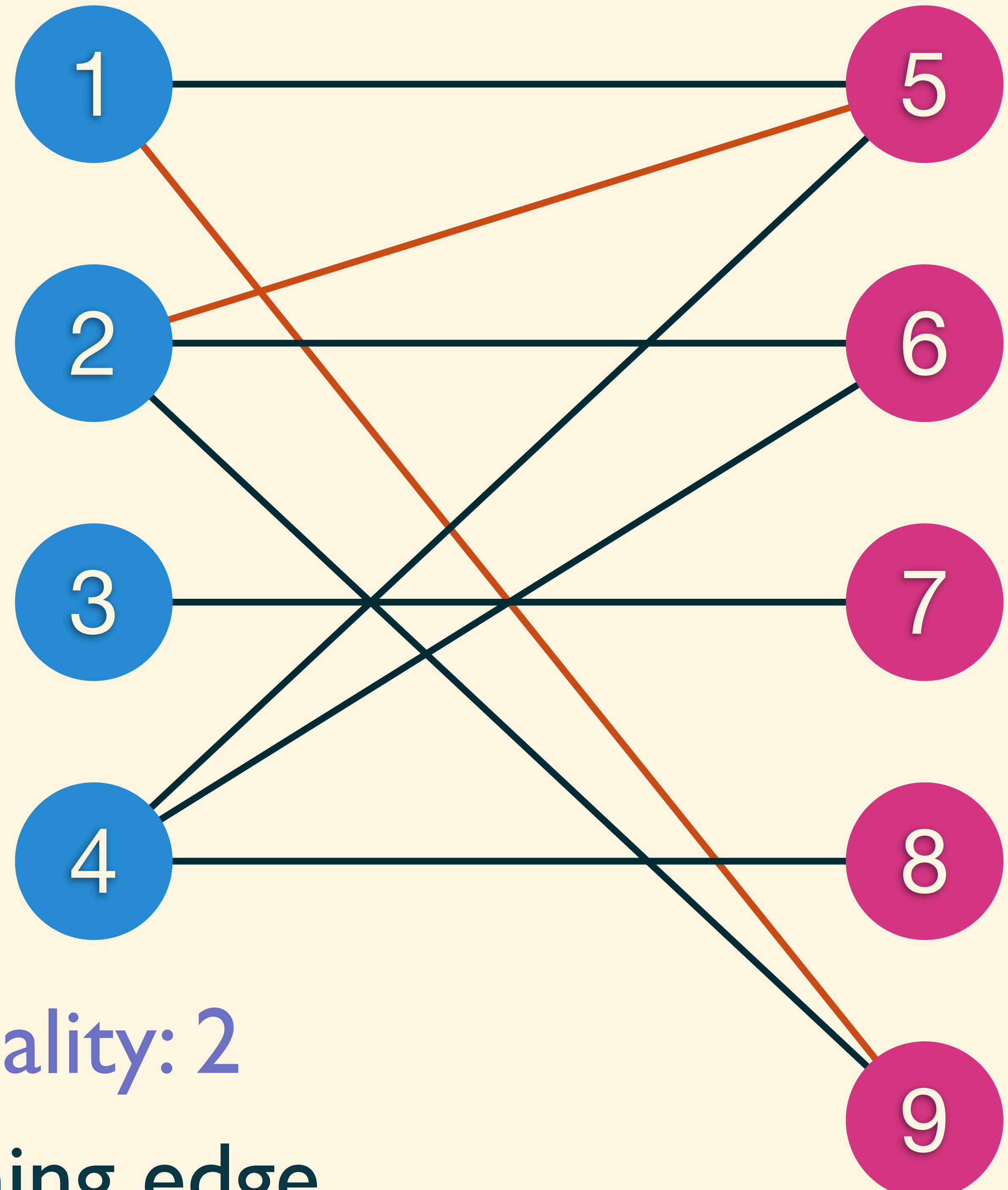
Reverse it!



current cardinality: 2

— not matching edge
— matching edge

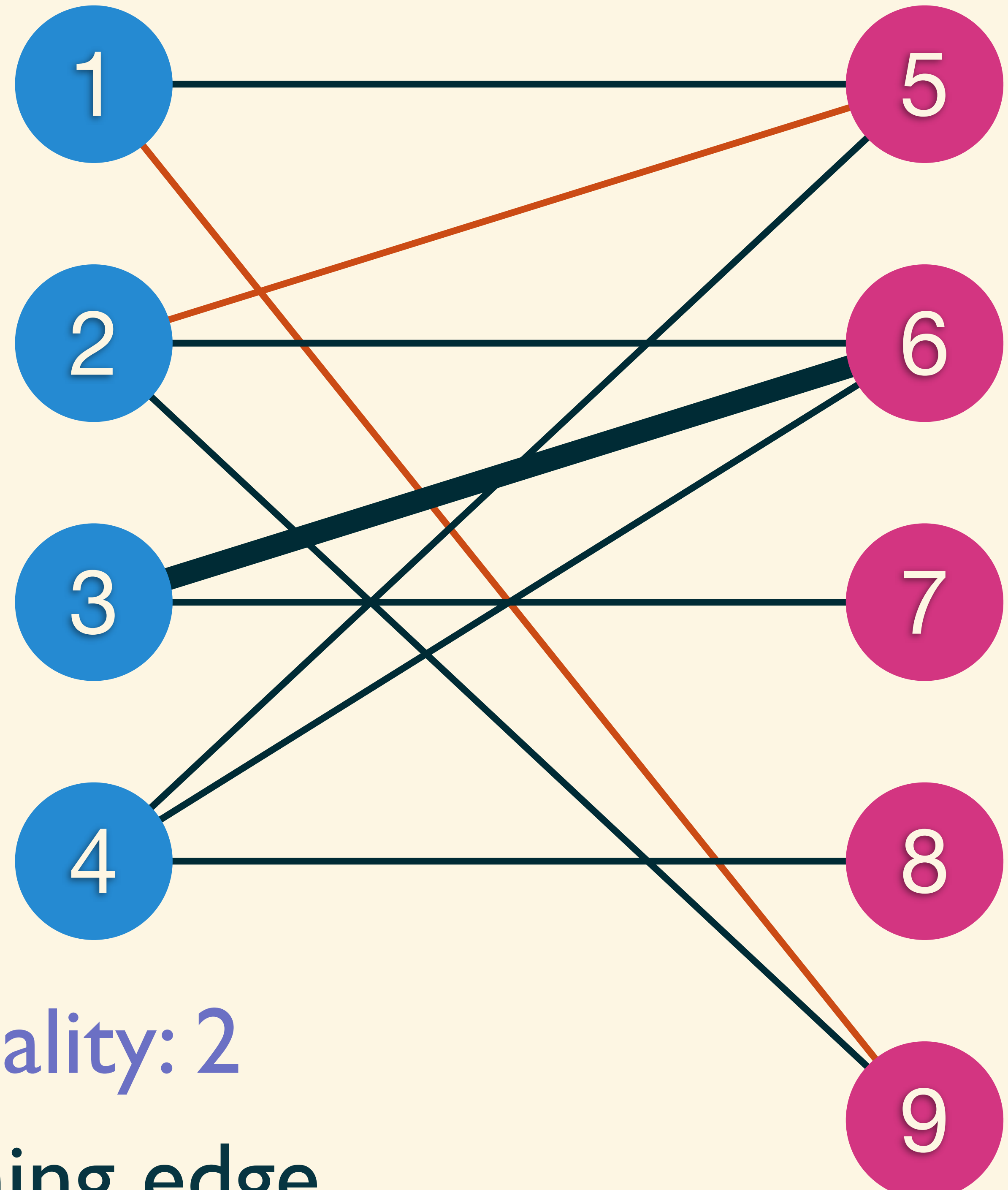
[3 - 6] is an augmenting path.



current cardinality: 2

— not matching edge
— matching edge

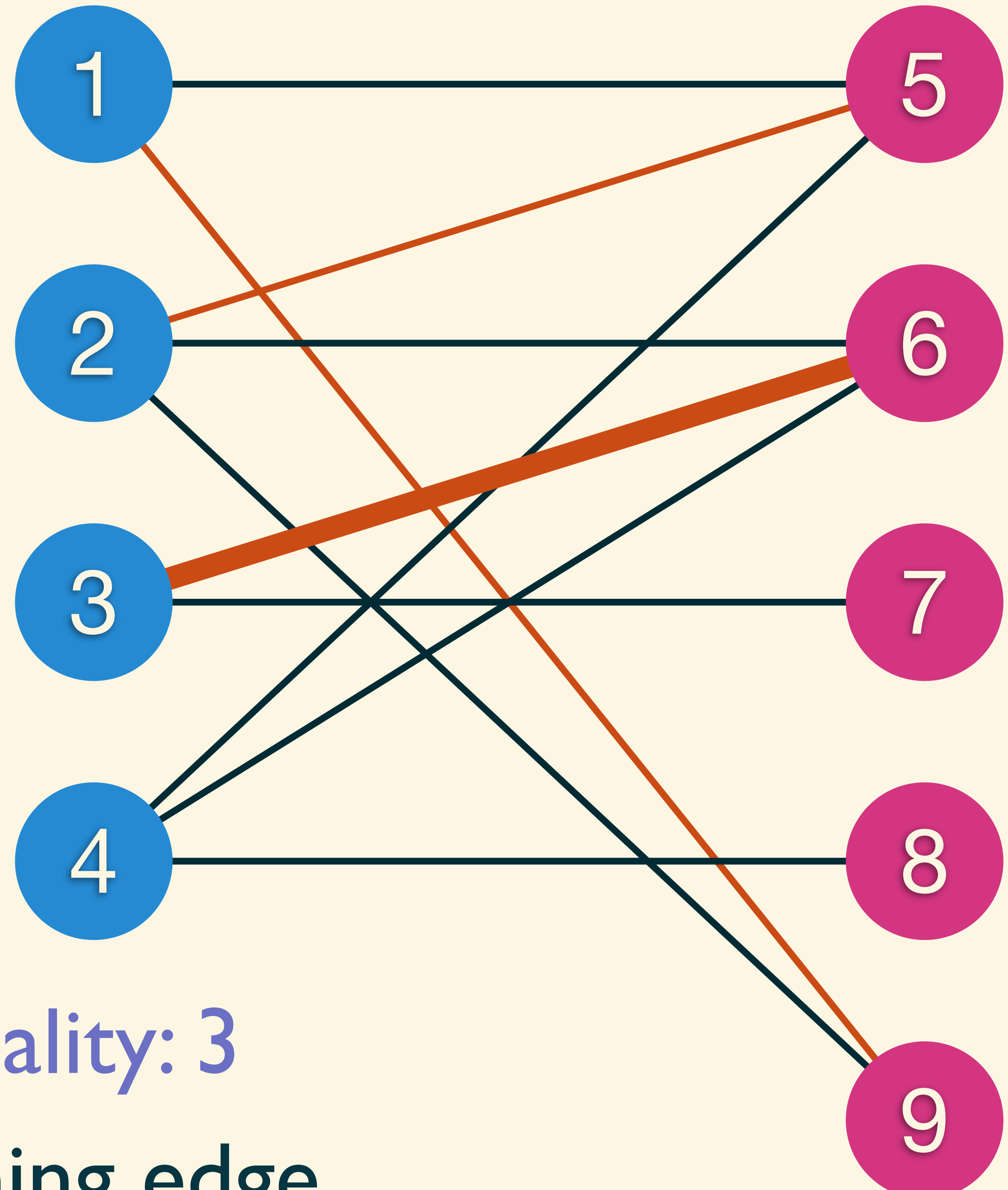
[3 - 6] is an augmenting path.



current cardinality: 2

— not matching edge
— matching edge

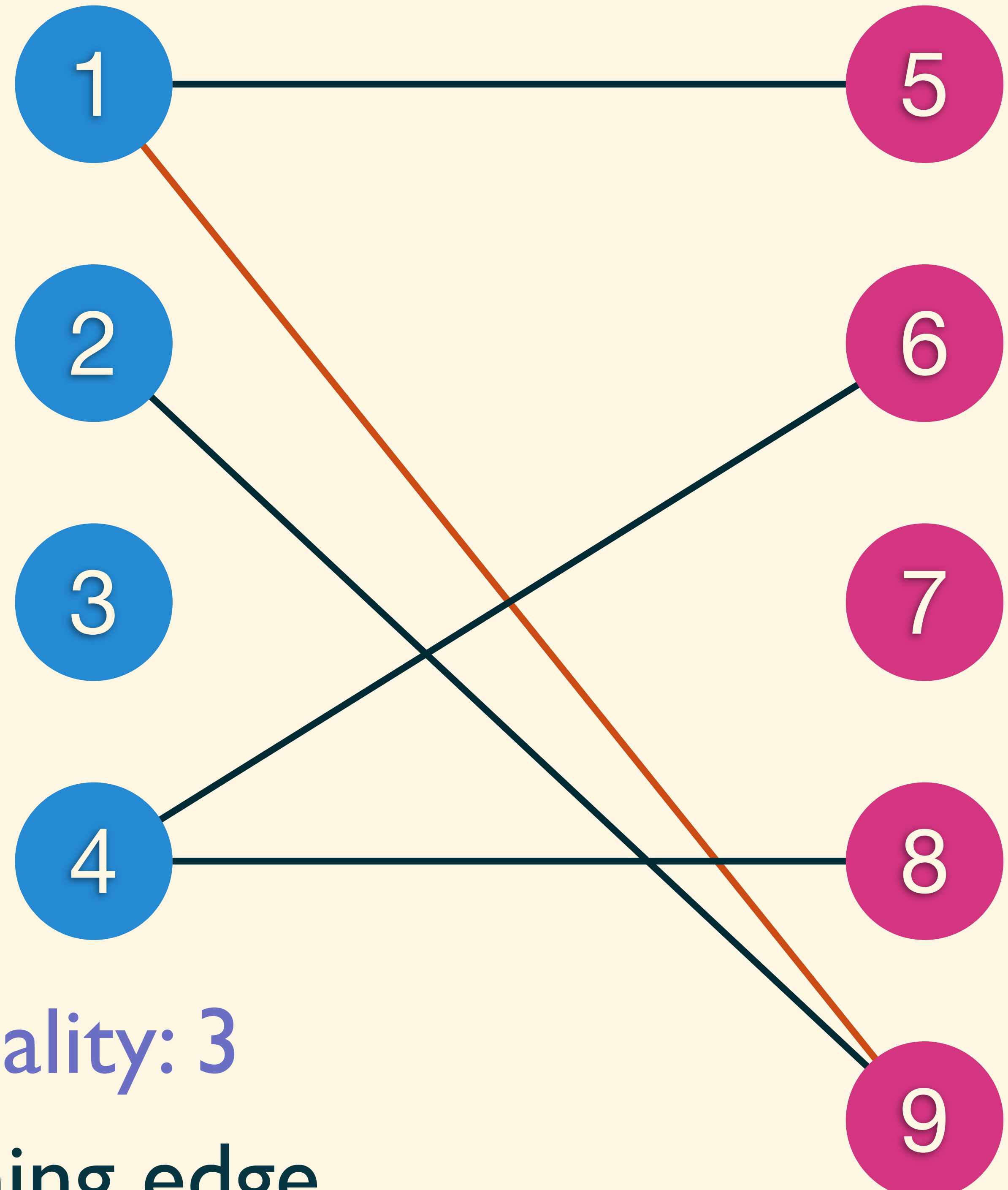
Reverse it!



current cardinality: 3

— not matching edge
— matching edge

[4 - 5 - 2 - 6 - 3 - 7] is an augmenting path.

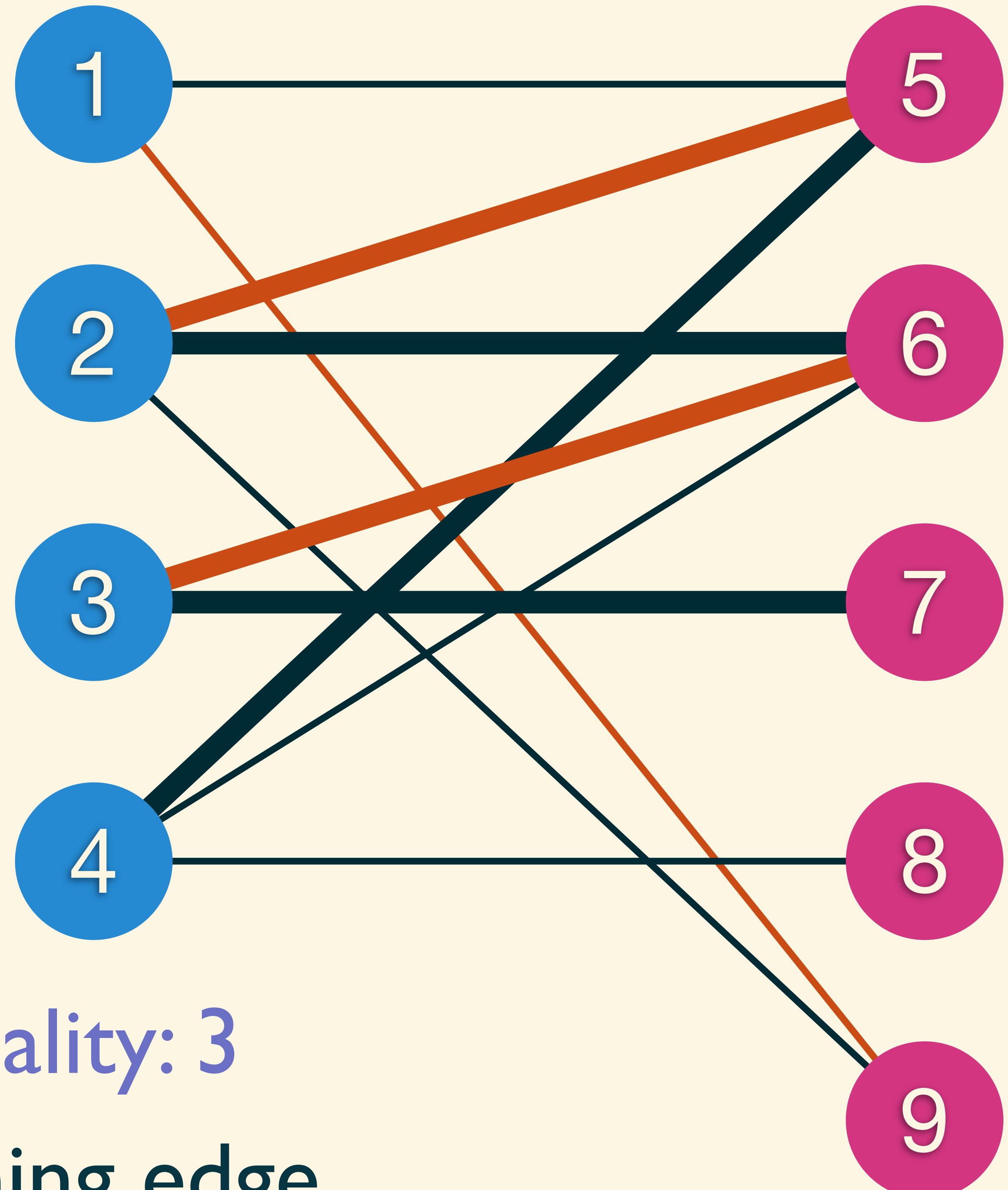


current cardinality: 3

— not matching edge

— matching edge

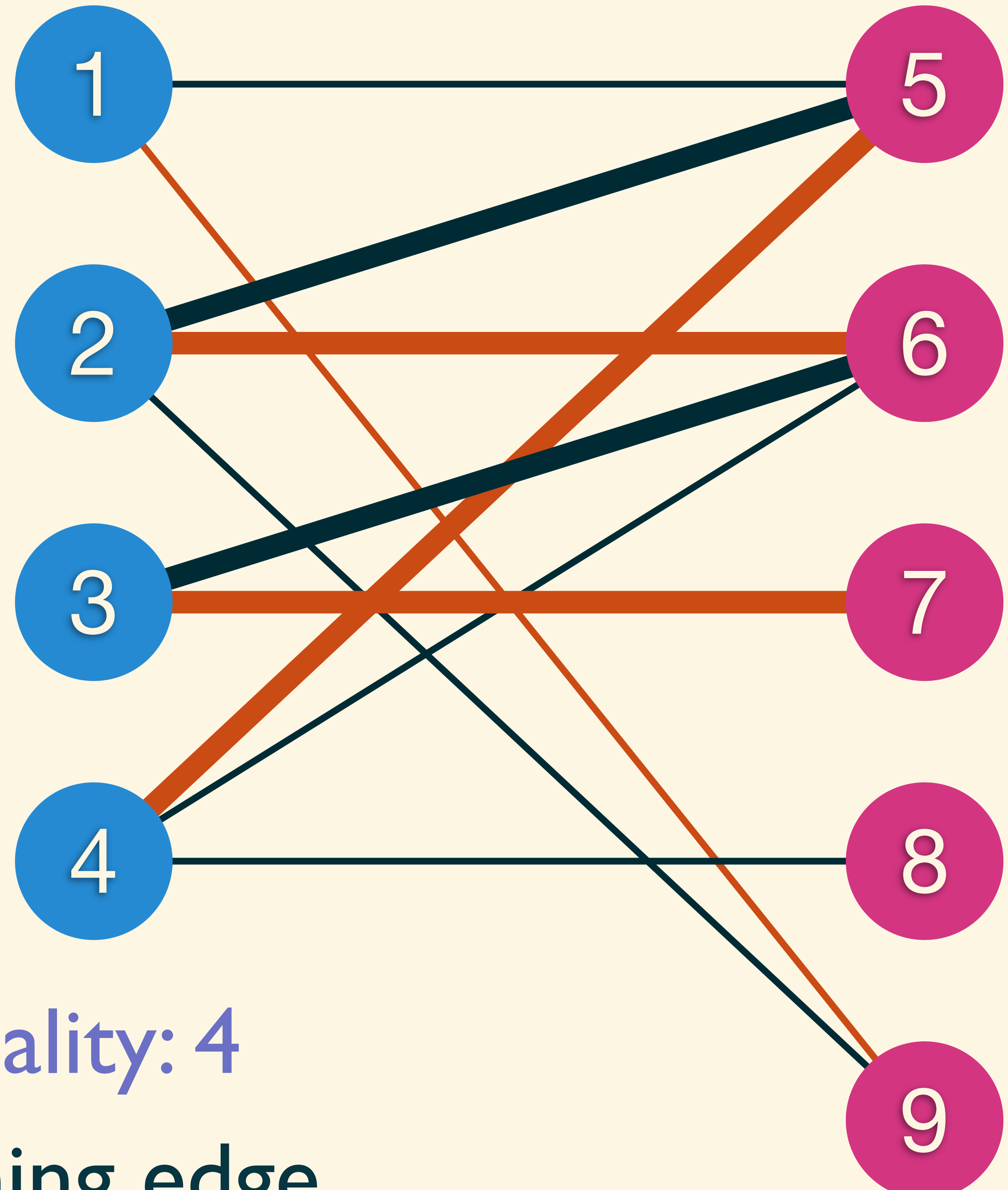
[4 - 5 - 2 - 6 - 3 - 7] is an augmenting path.



current cardinality: 3

— not matching edge
— matching edge

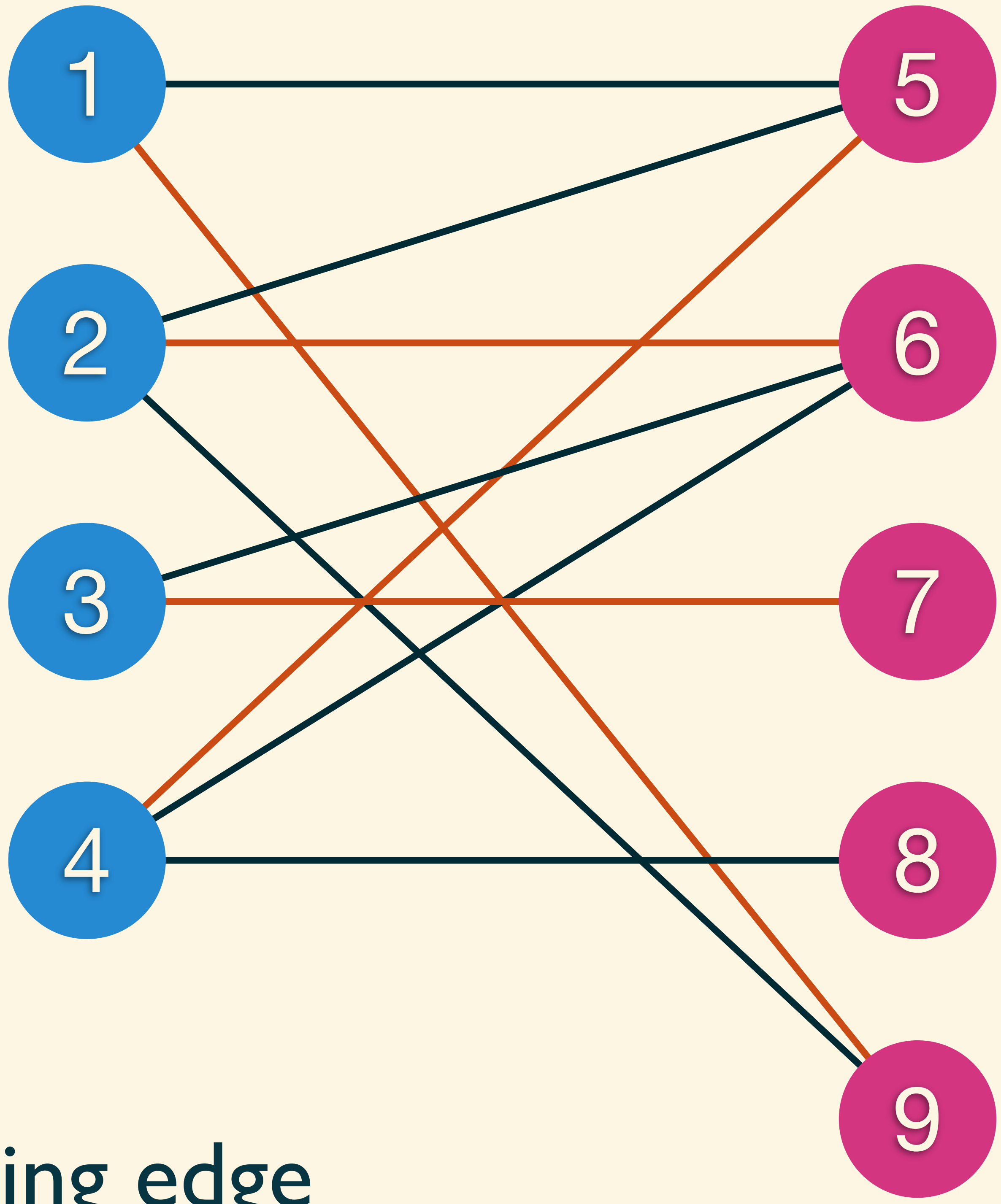
Reverse it!



current cardinality: 4

— not matching edge
— matching edge

Max cardinality is 4.



— not matching edge
— matching edge

Time Complexity

$$O(\mathbf{V} \times \mathbf{E})$$

V: number of vertices
E: number of edges

Source Code

```
// find an augmenting path
bool find_aug_path( int x ) {
    for ( int i = 0; i < ( int )vertex[ x ].size(); ++i ) {
        int next = vertex[ x ][ i ];
        // not in augmenting path
        if ( !visit[ next ] ) {
            // setup this vertex in augmenting path
            visit[ next ] = 1;
            /*
             * If this vertex is a unmatched vertex, reverse
augmenting path and return.
             * If this vector is a matched vertex, try to reverse
augmenting path and continue find an unmatched vertex.
             */
            if ( lnk2[ next ] == -1 || find_aug_path( lnk2[ next ] ) )
{
                lnk1[ x ] = next, lnk2[ next ] = x;
                return 1;
            }
        }
    }
    return 0;
}
```

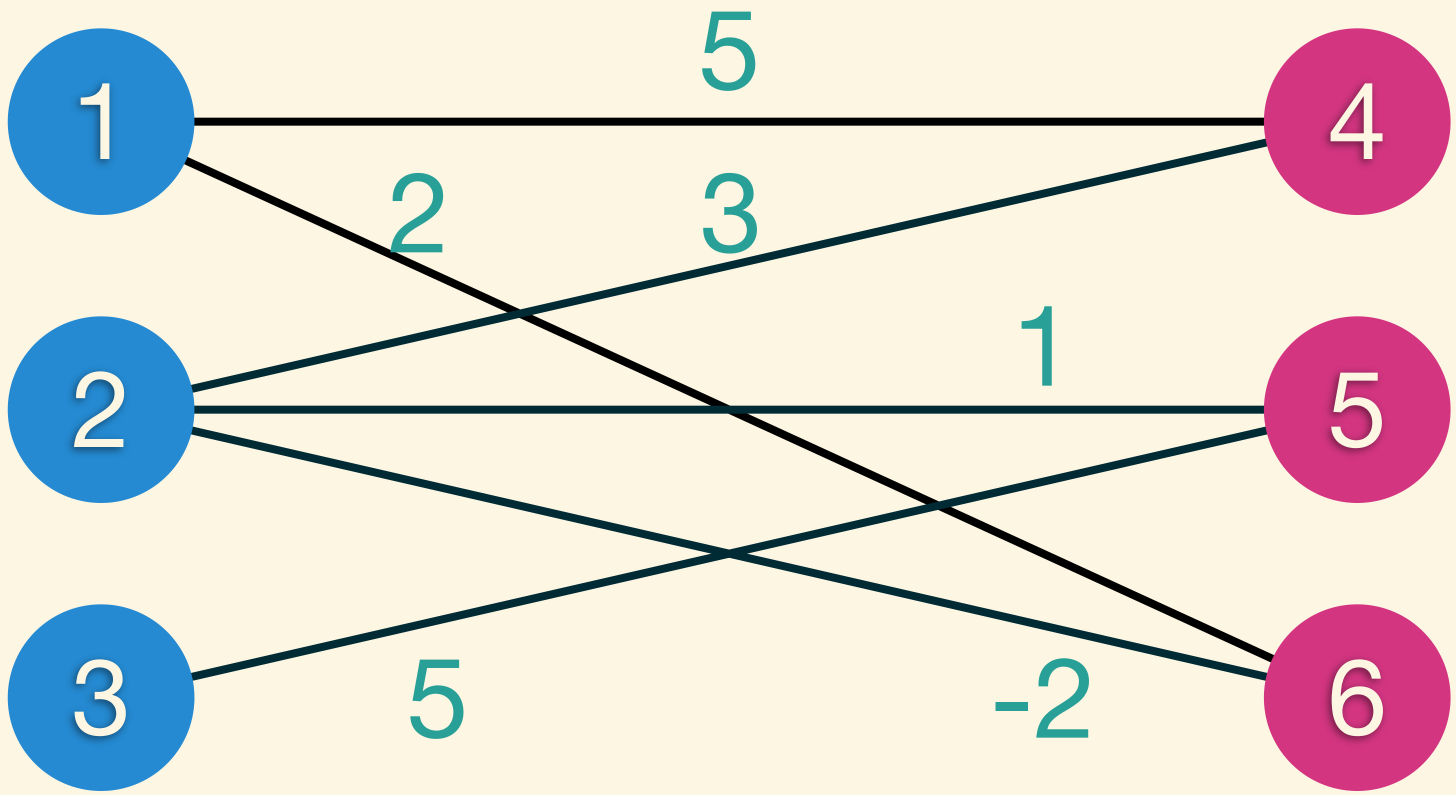
Practice Now

UVa 670 - The dog task

Maximum Weight Matching

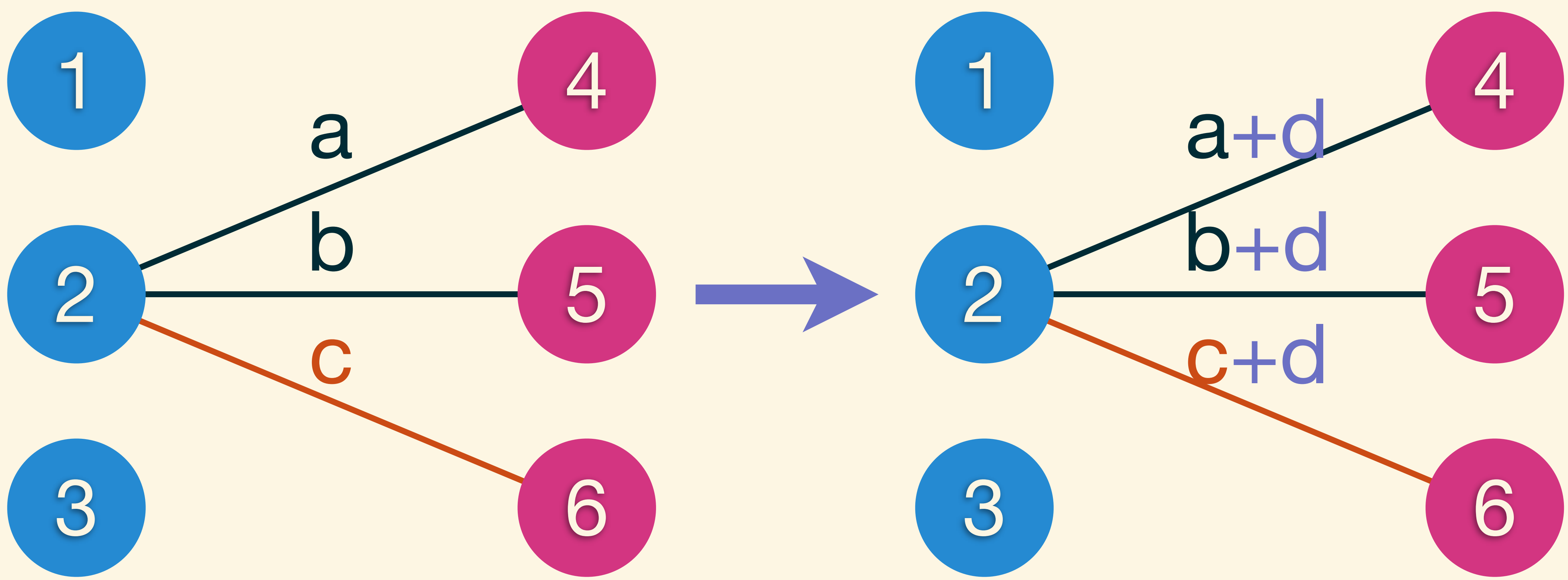
Hungarian Algorithm

(Kuhn-Munkres Algorithm)


1
2
3
4
5
6
5
2
3
1
5
-2

weight adjustment

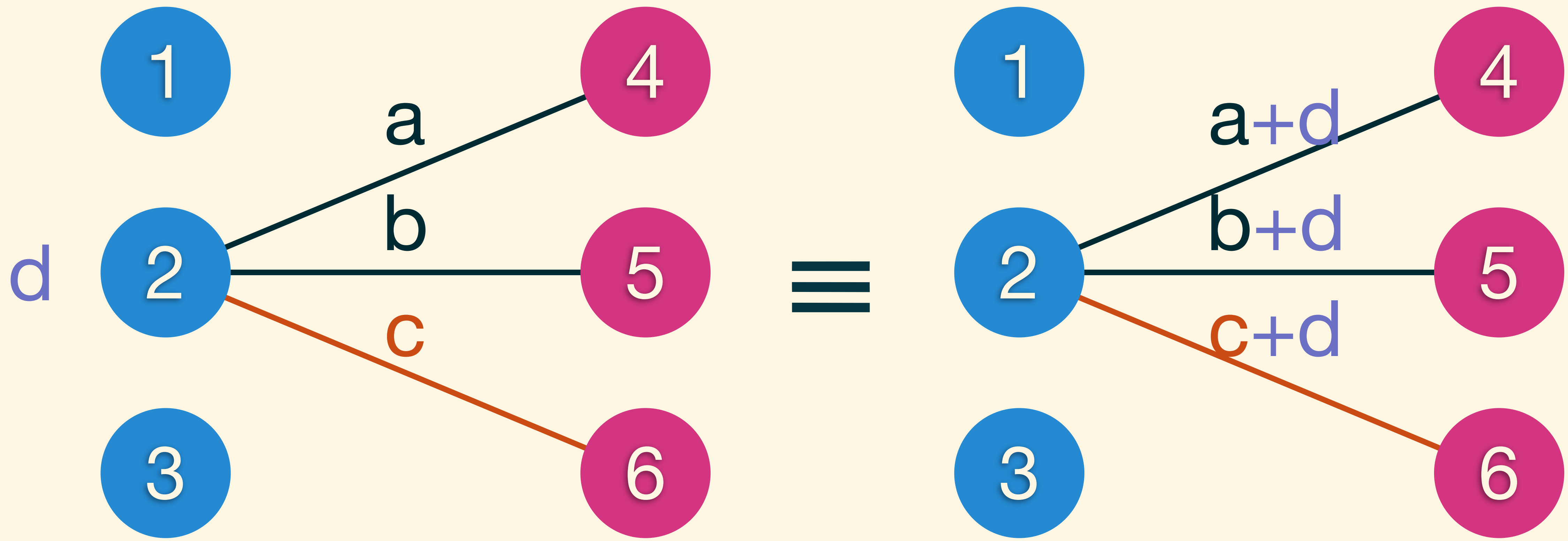
If add (subtract) some value on all edges connected with vertex X, the maximum matching won't be effected.



— matching edge — not matching edge

vertex labeling

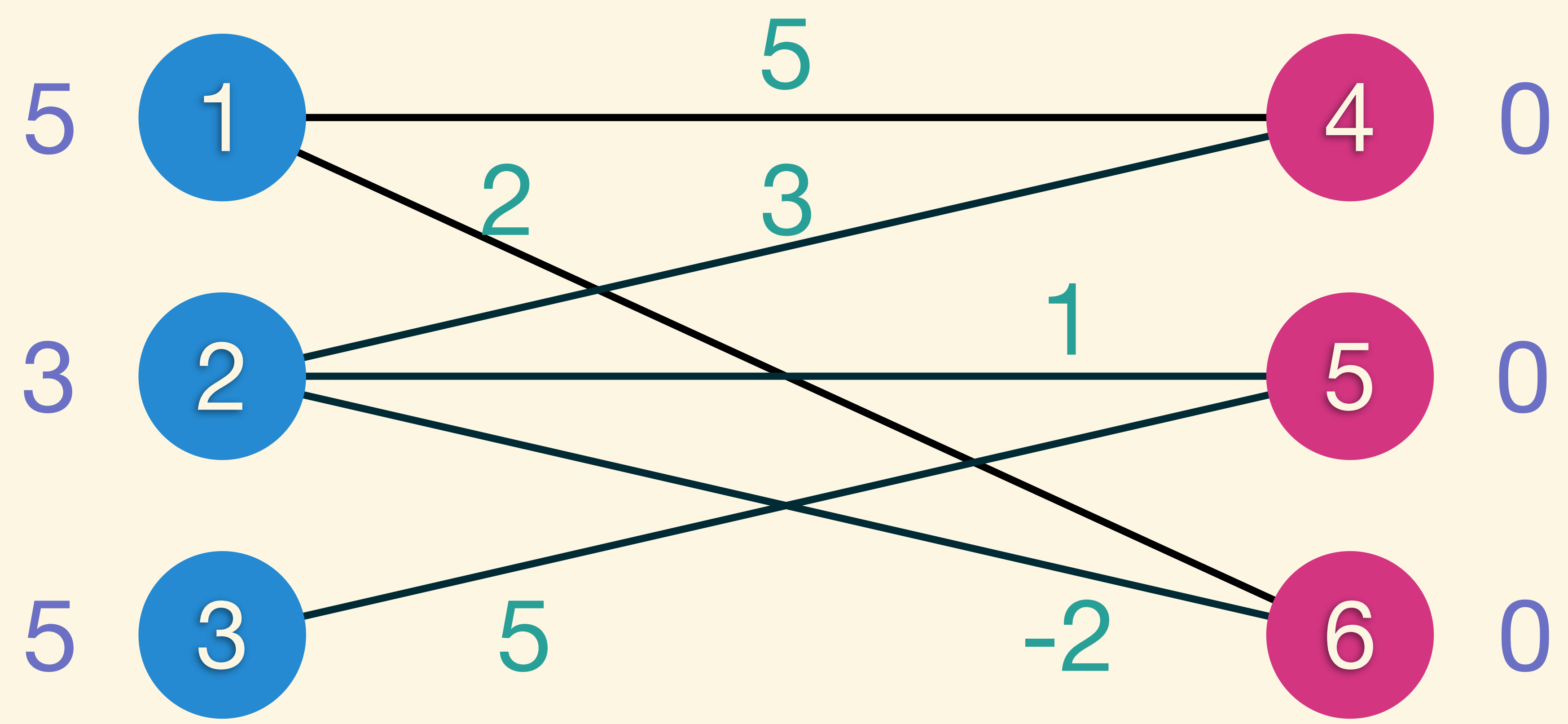
For convenient, add a variable on vertices to denote the value add (subtract) on edges connected with them.



— matching edge — not matching edge

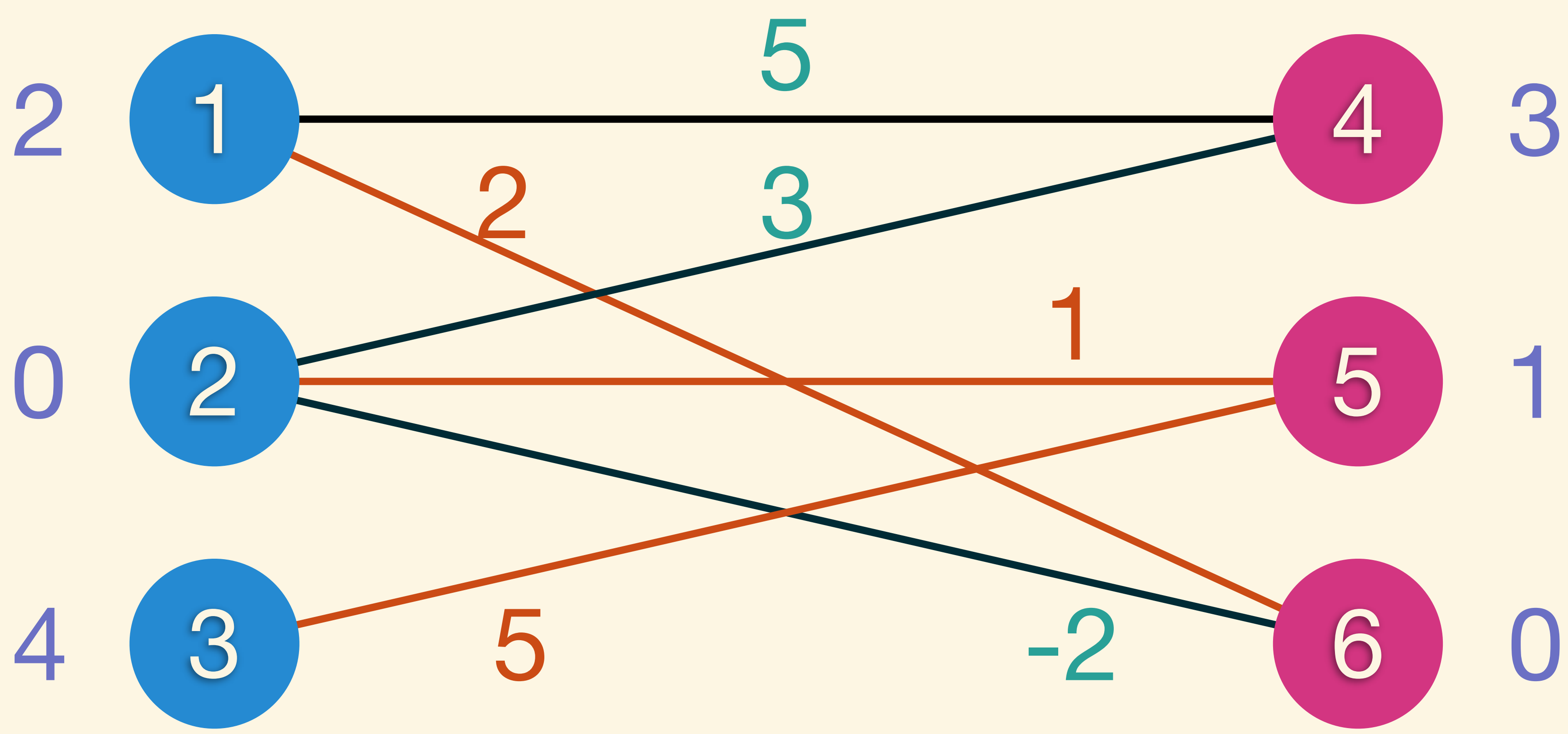
vertex labeling

`l(x)+l(y)≥w(x,y)`, for each edge(x,y)



minimize vertex labeling

Admissible Edge: `l(x)+l(y)=w(x,y)`



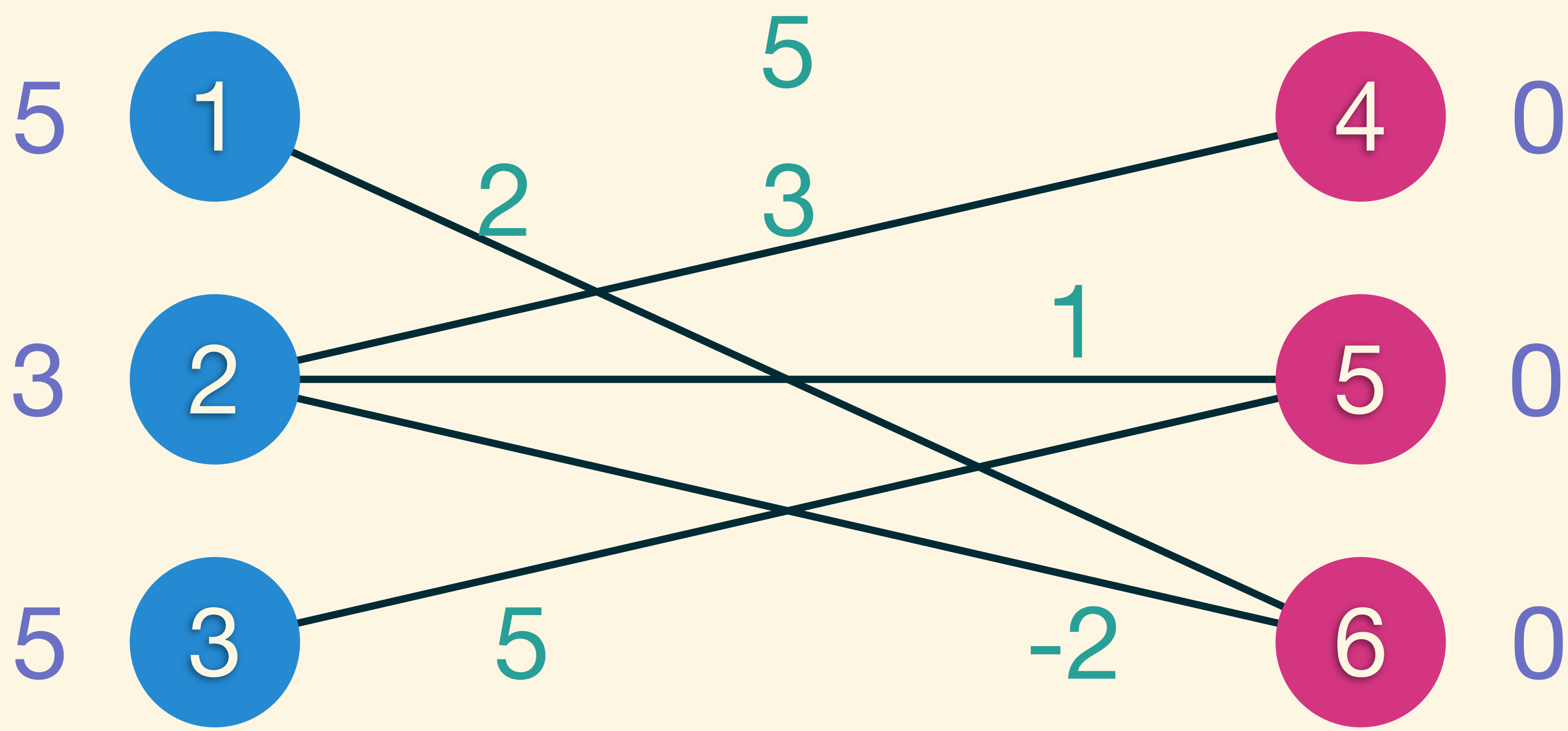
convert

maximum weight problem

to

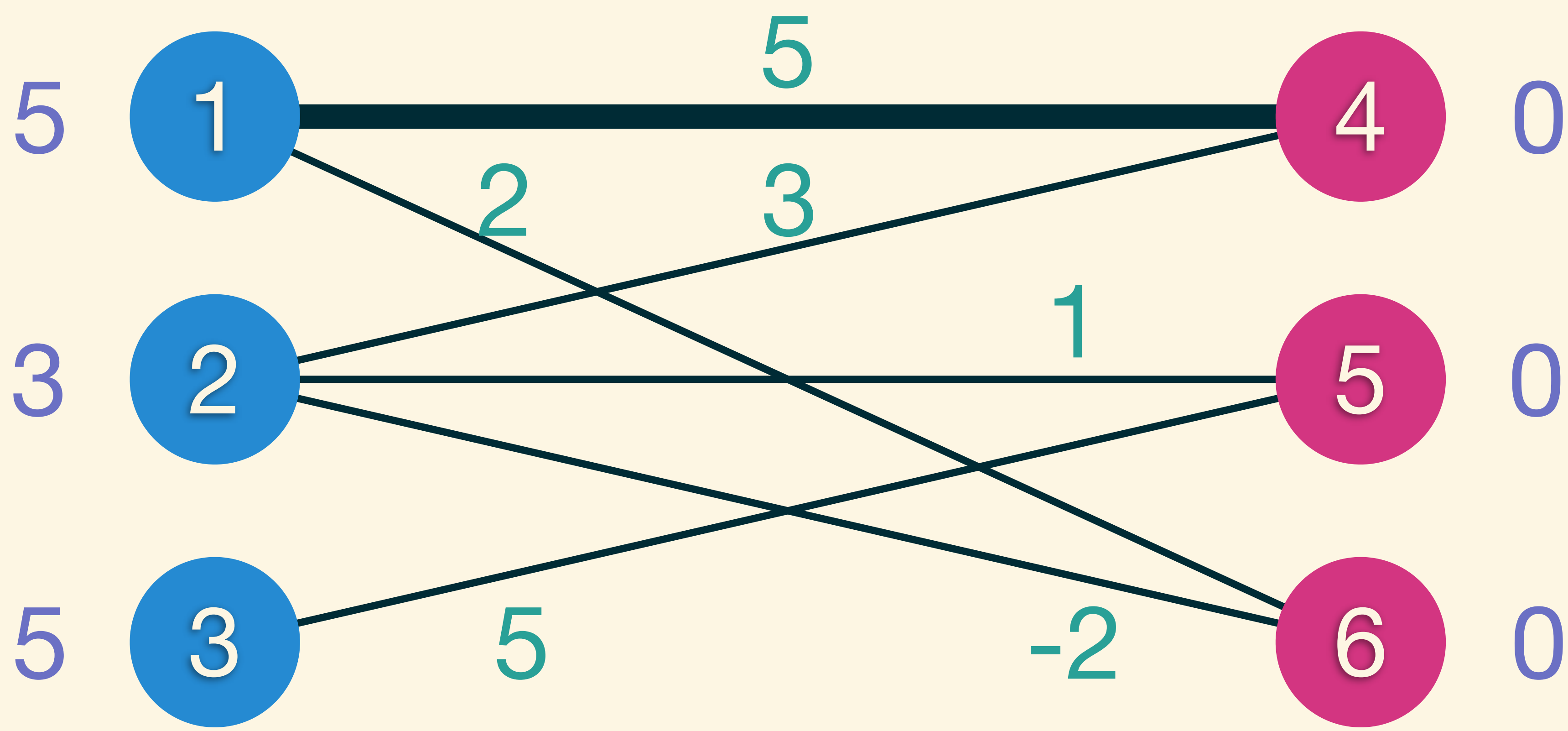
minimum vertex labeling problem

Augmenting Path with Admissible Edge



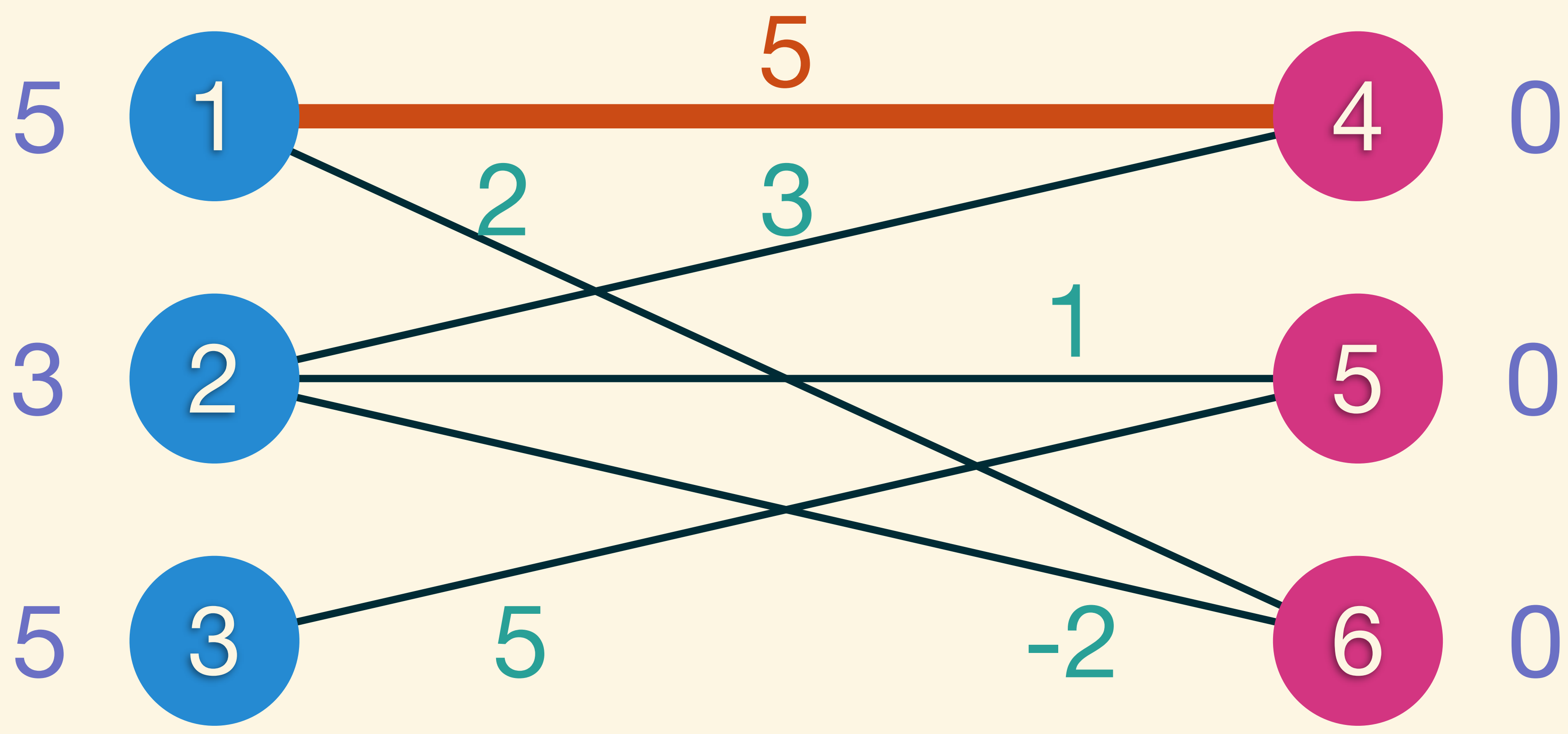
[1 - 4] is an augment path with admissible edges.

Augmenting Path with Admissible Edge



[1 - 4] is an augment path with admissible edges.

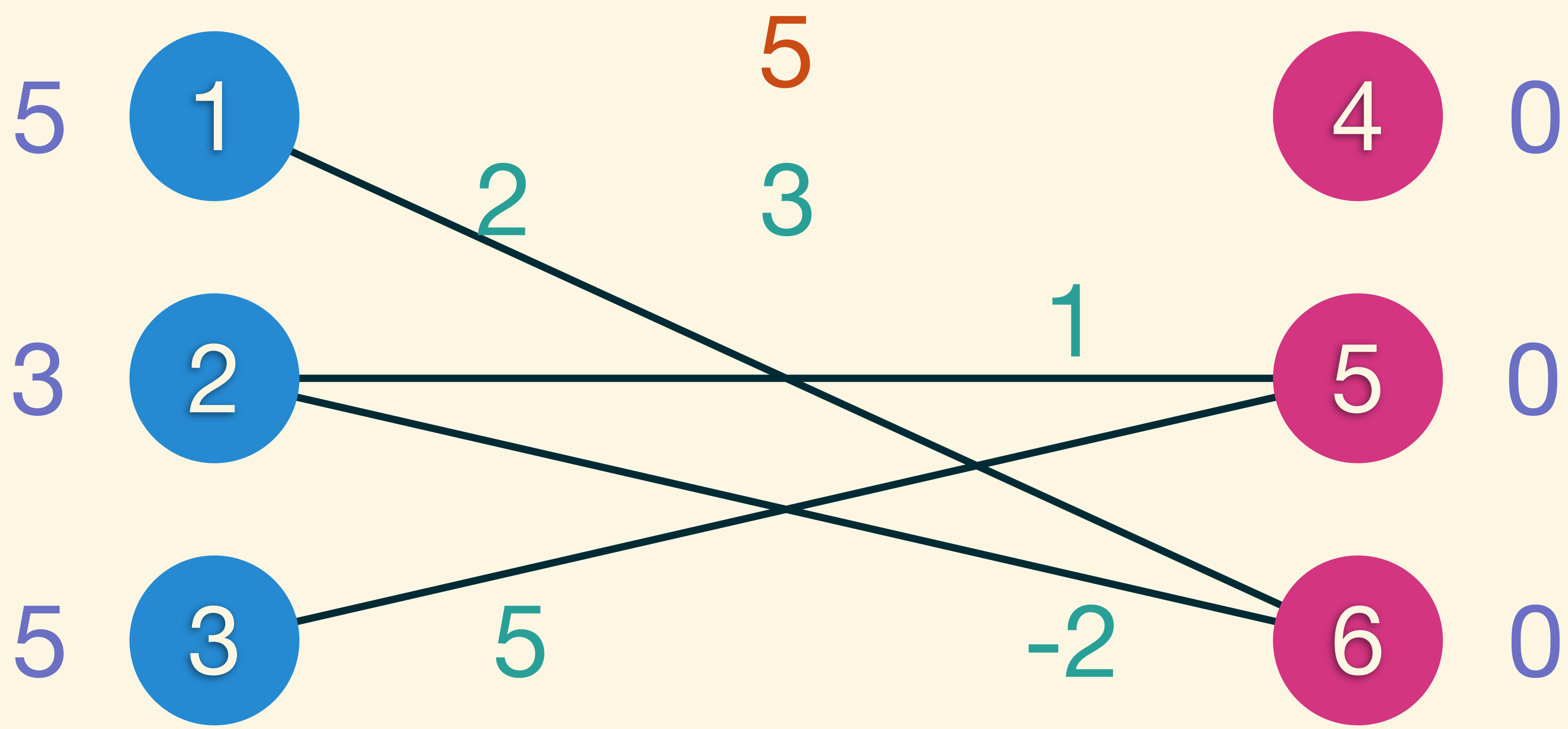
Augmenting Path with Admissible Edge



Reverse it!

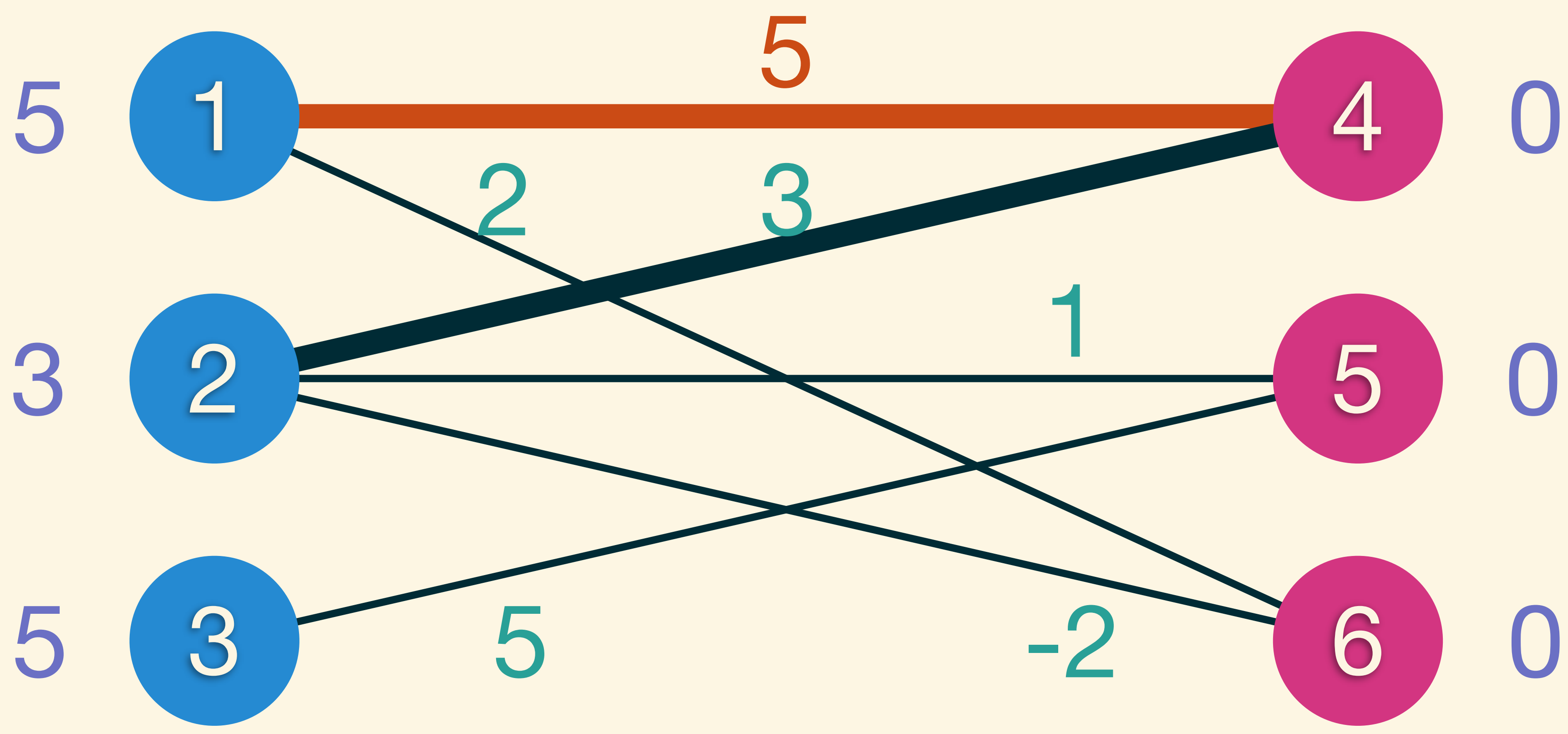
Current Weight = 5

Augmenting Path with Admissible Edge



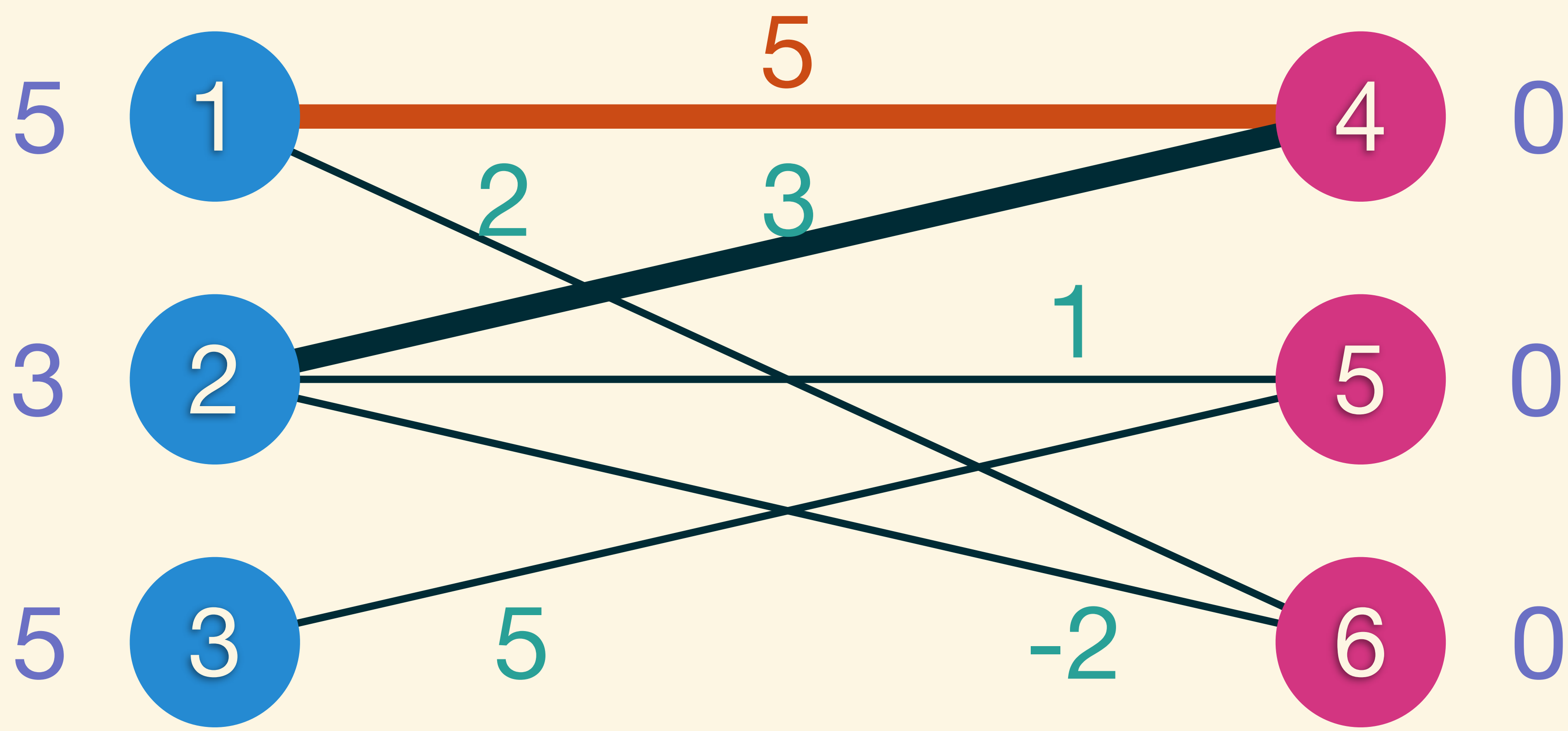
No augmenting path found start from vertex 2.

Augmenting Path with Admissible Edge



No augmenting path found start from vertex 2.

Adjust Vertex Labeling

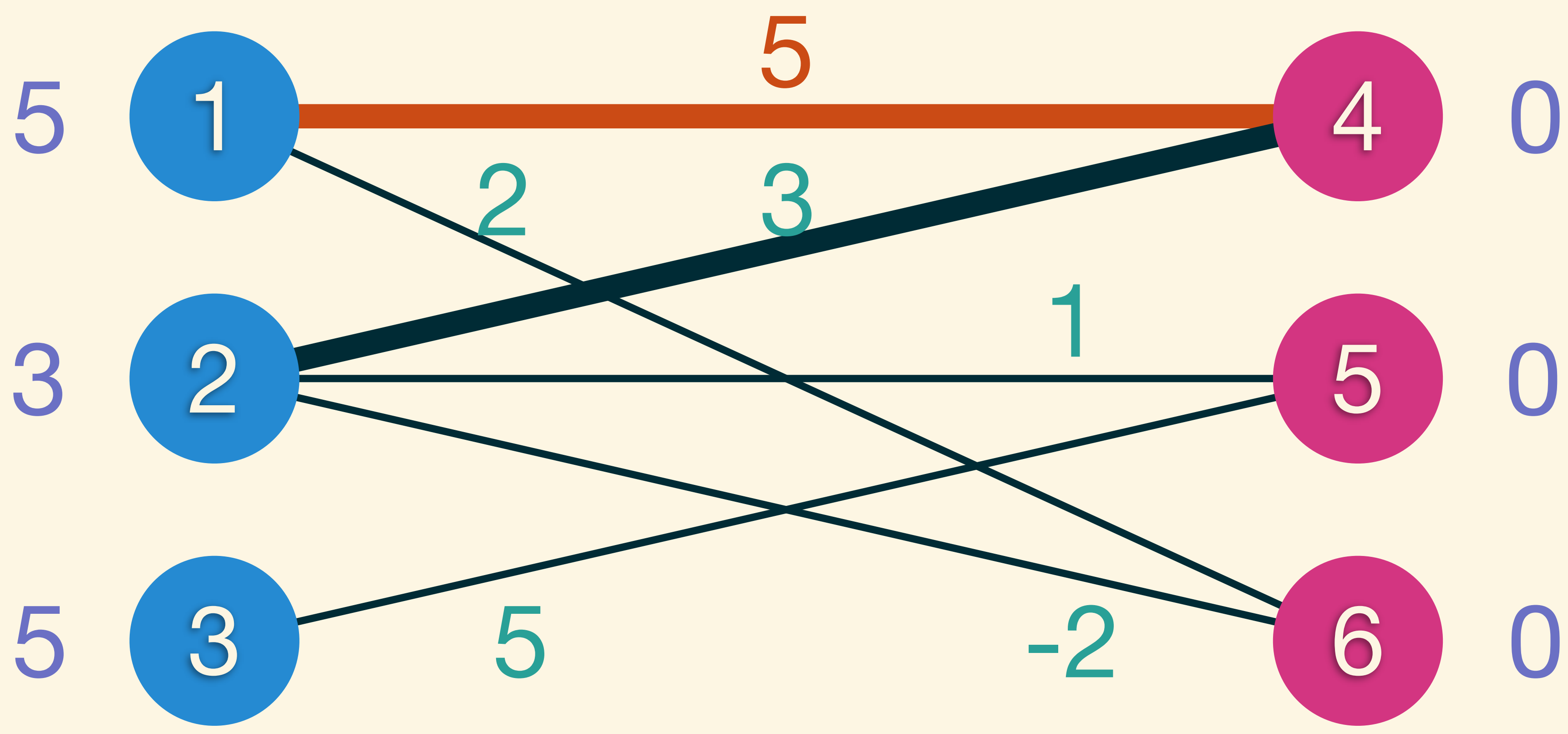


relax value d = `min(l(x)+l(y)-w(x,y))`

x: vertex in alternating path

y: vertex y not in alternating path

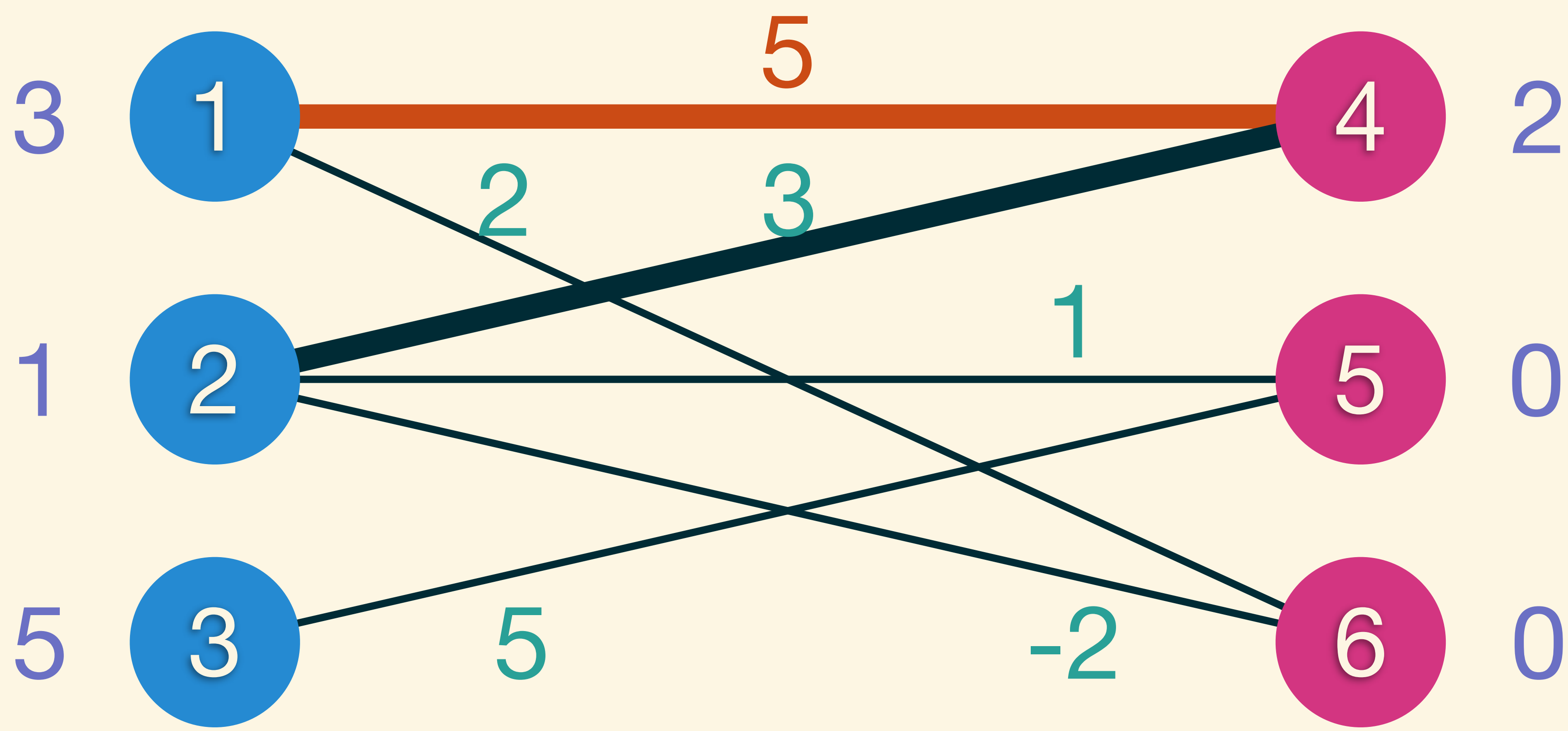
Adjust Vertex Labeling



R(1,6)=3
R(2,5)=2
R(2,6)=5

relax d = 2

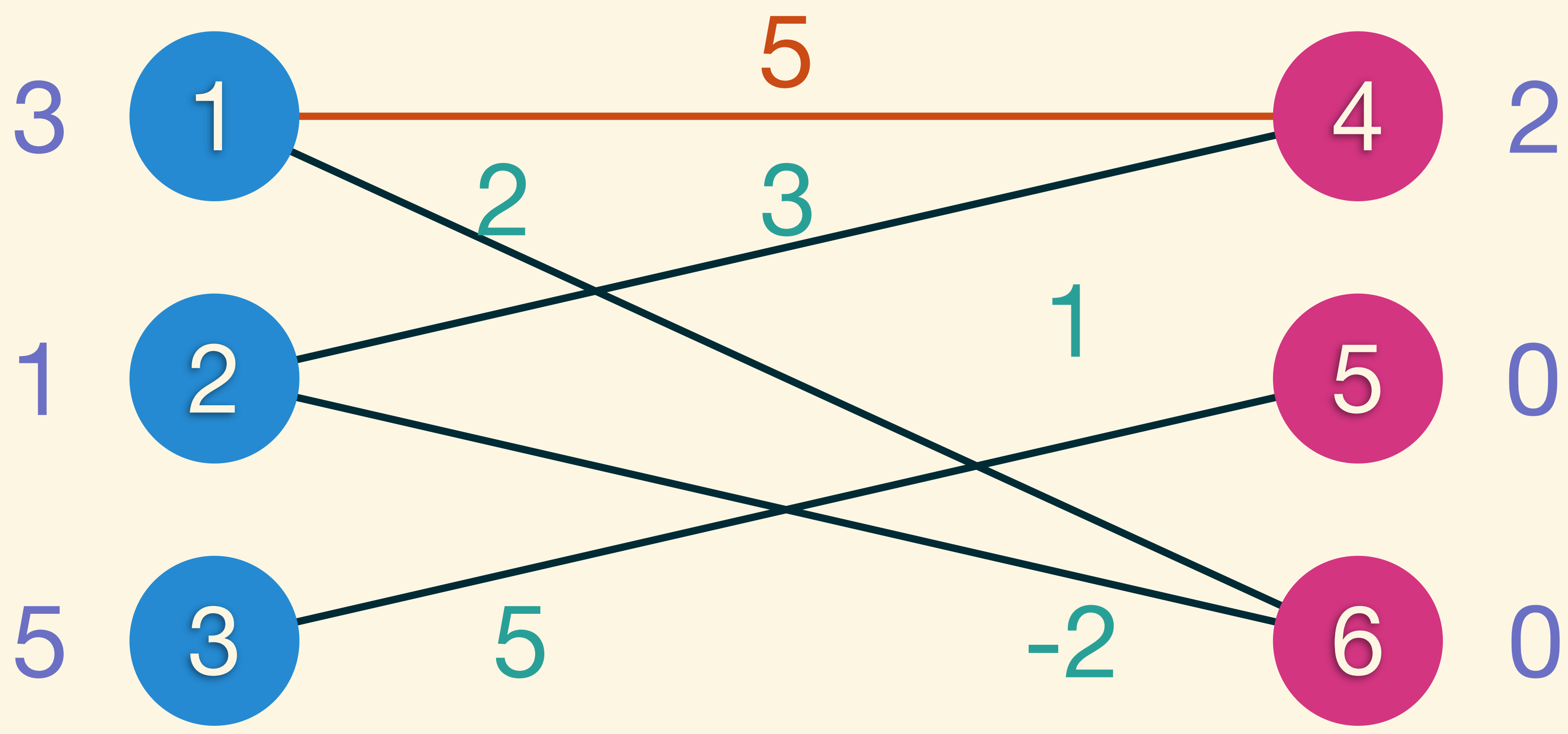
Adjust Vertex Labeling



`l(x)` subtract value d.

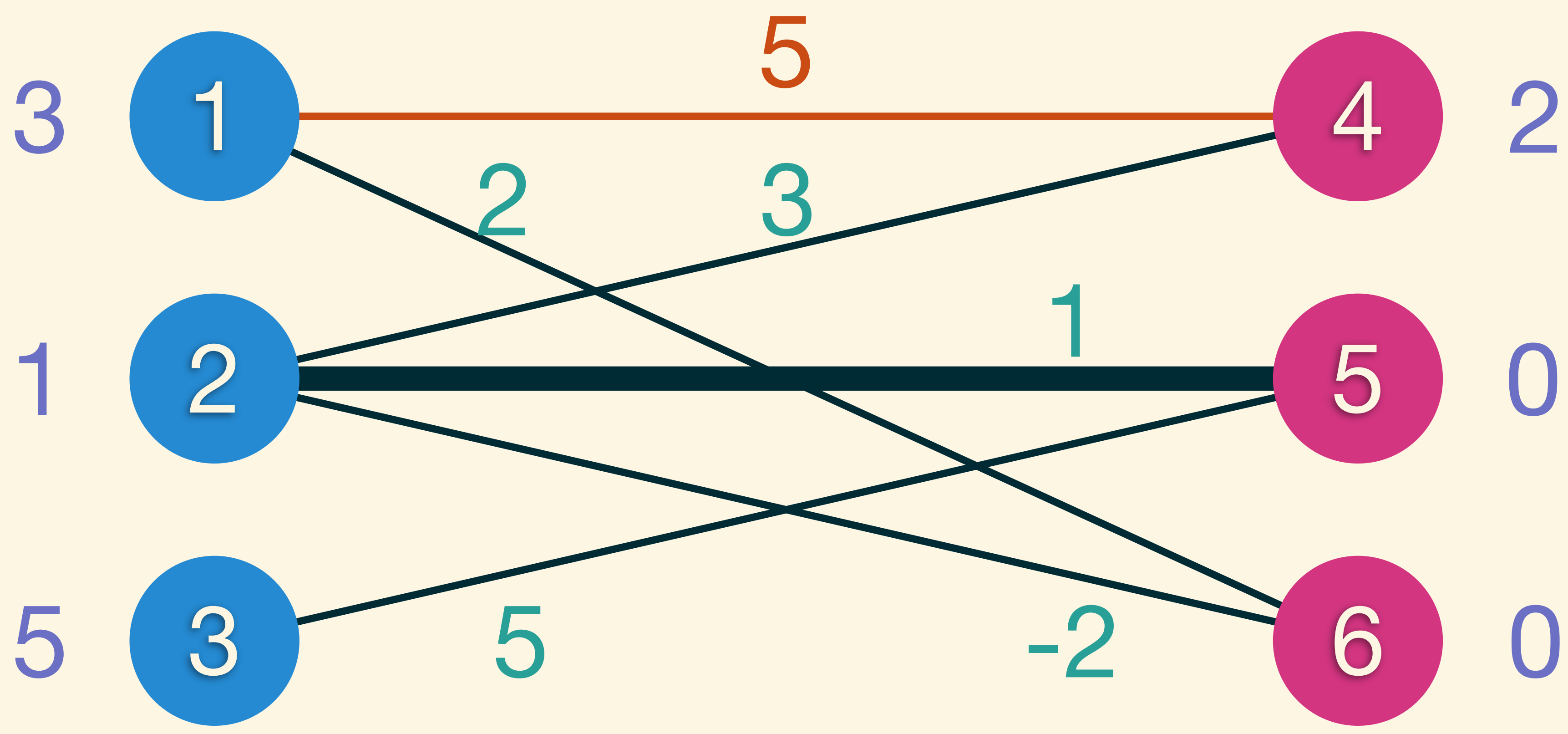
`l(y)` add value d.

Continue to Find Augmenting Path



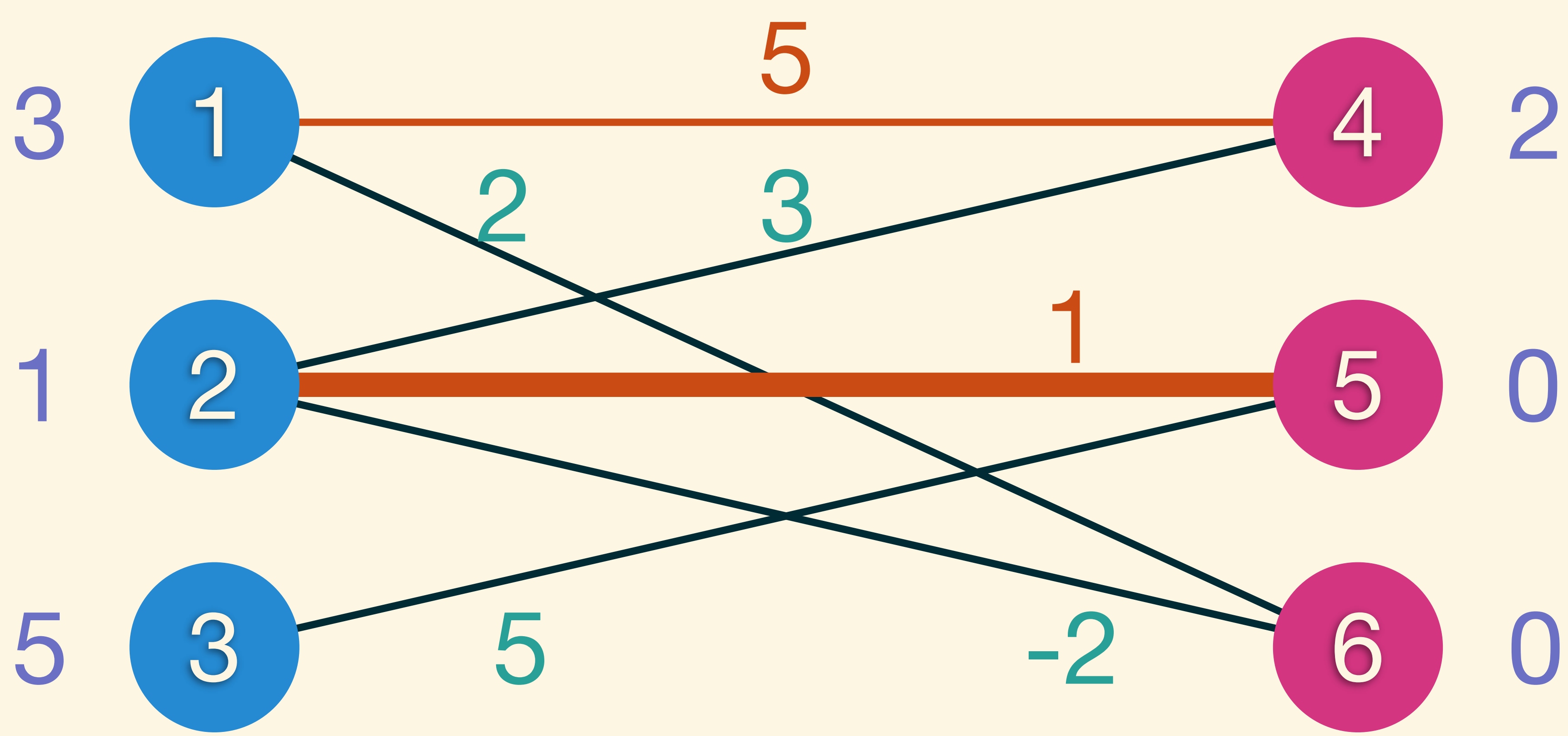
[2 - 5] is an augment path with admissible edges.

Continue to Find Augmenting Path



[2 - 5] is an augment path with admissible edges.

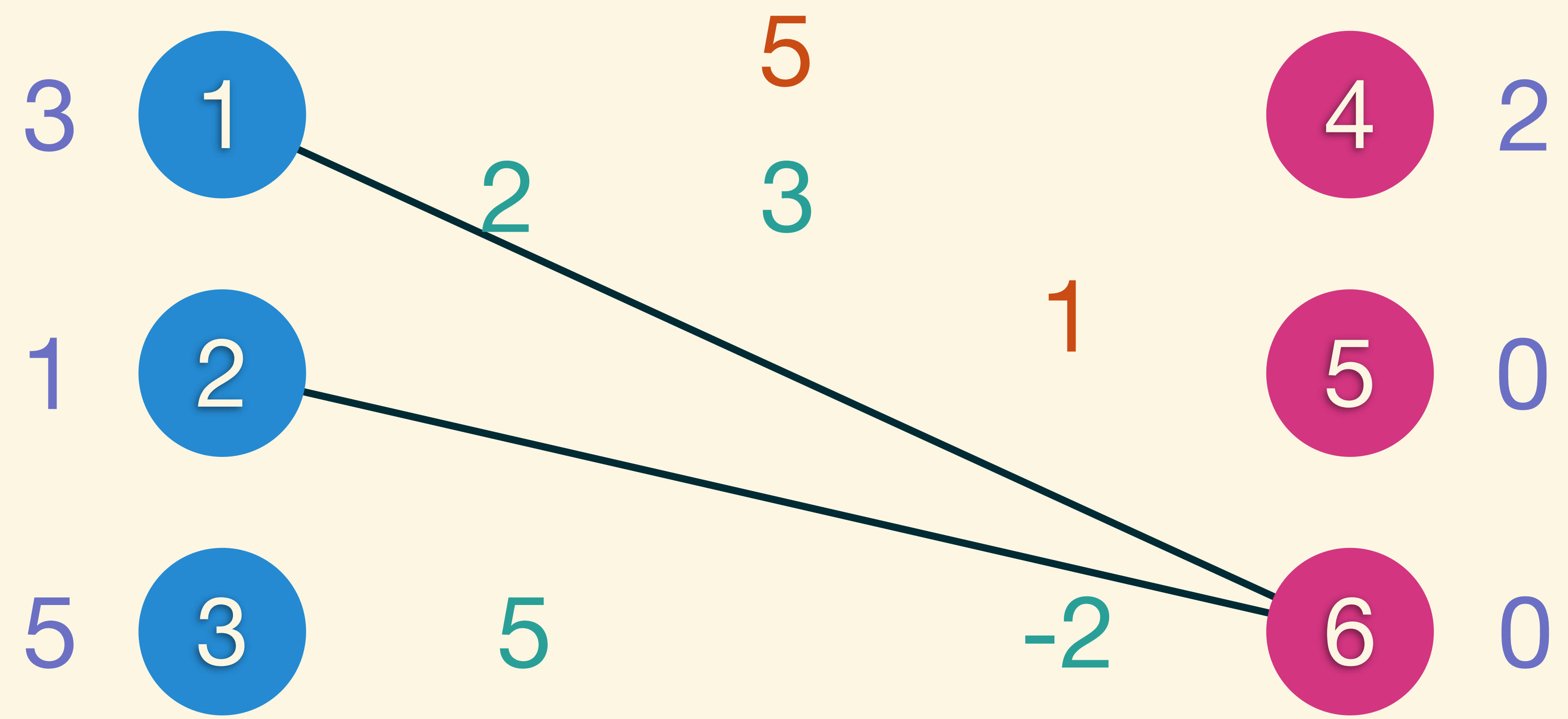
Continue to Find Augmenting Path



Reverse it!

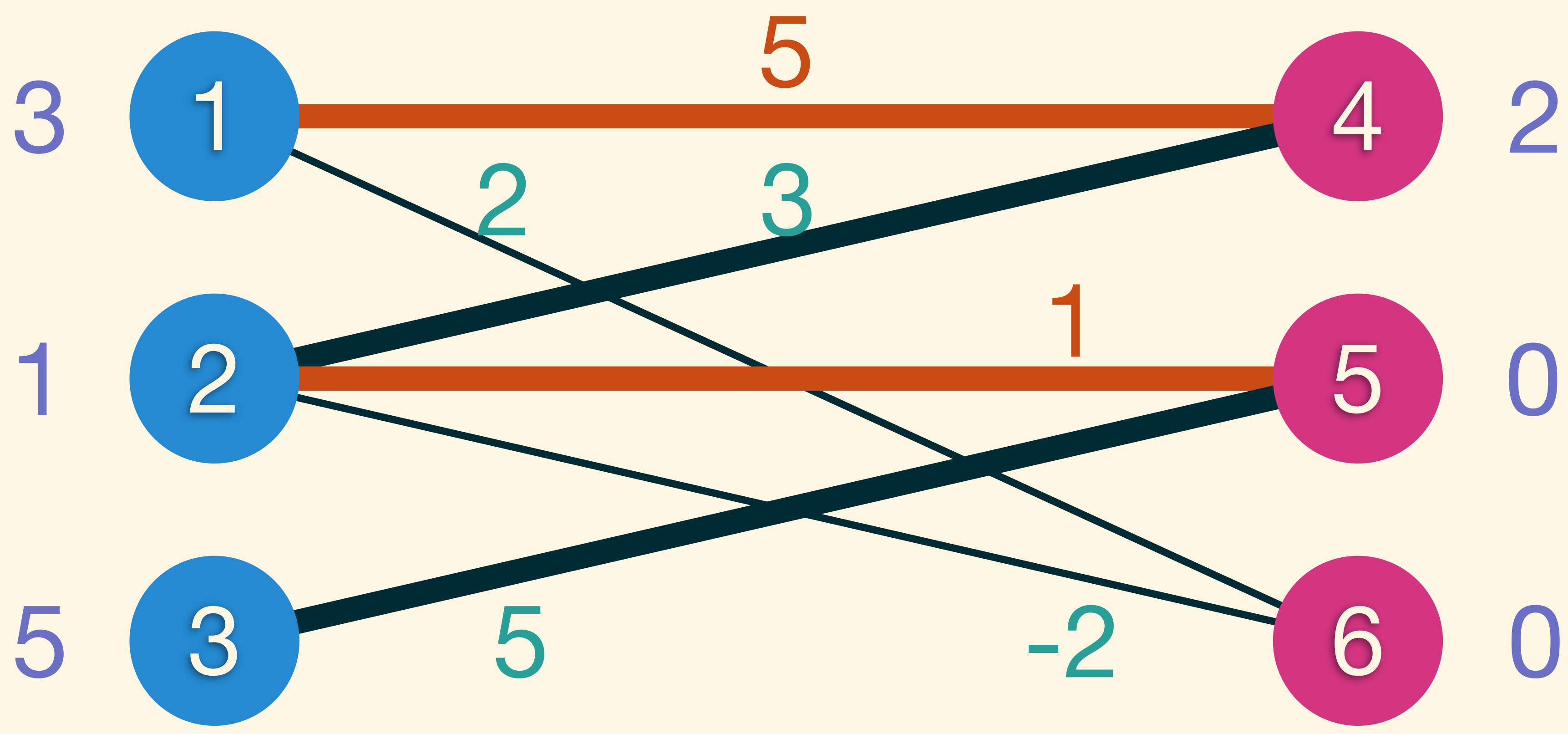
Current Weight = 6

Continue to Find Augmenting Path



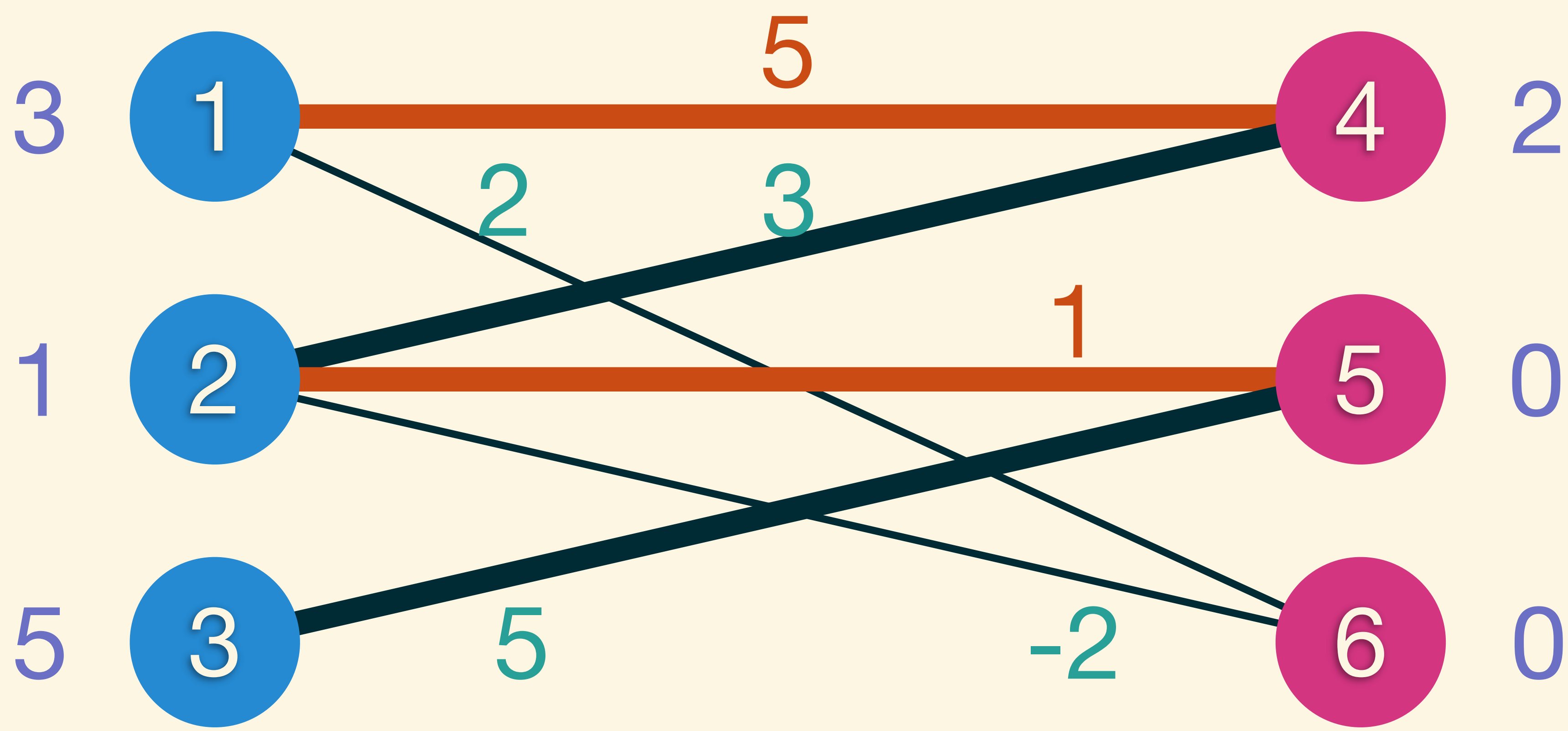
No augmenting path found start from vertex 3.

Continue to Find Augmenting Path



No augmenting path found start from vertex 3.

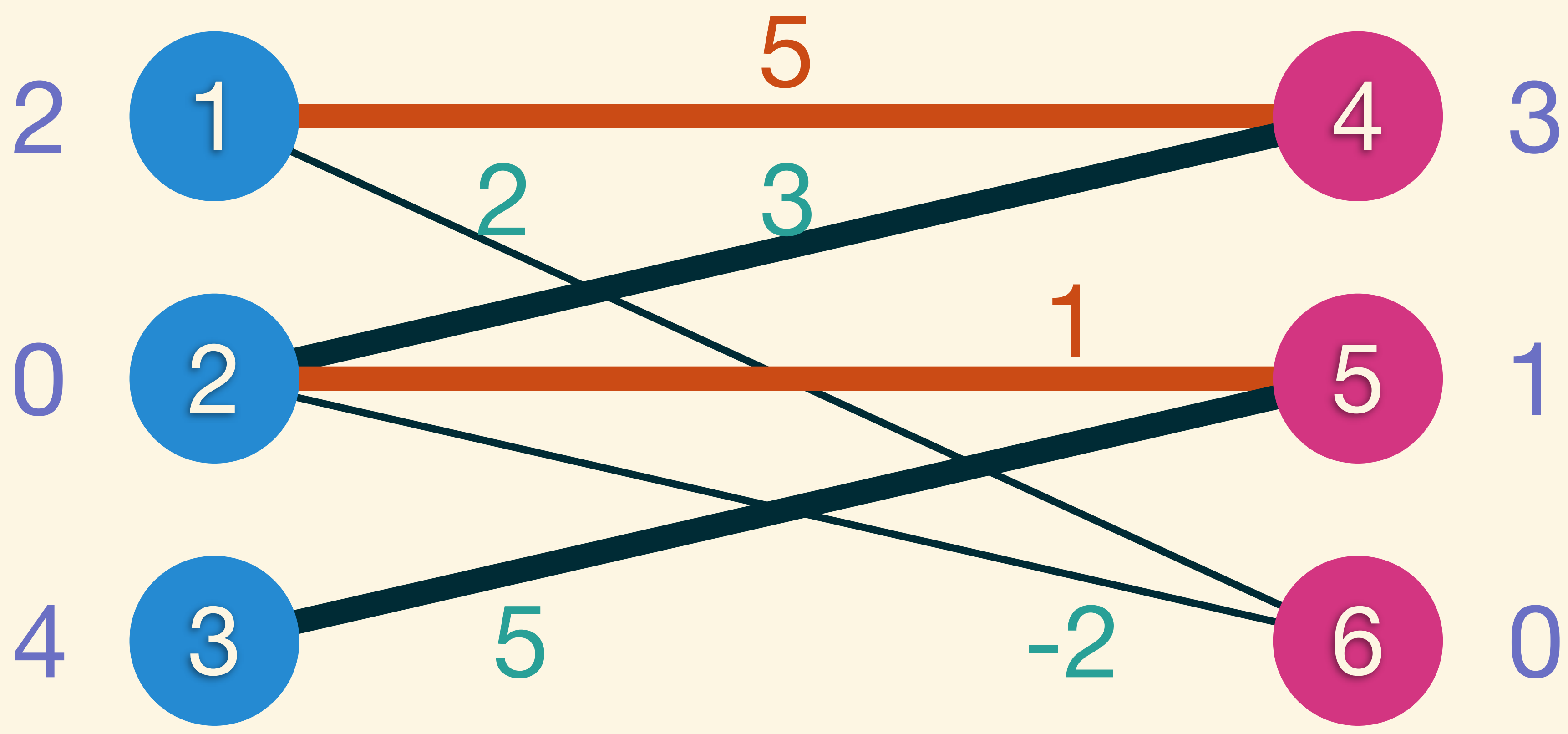
Adjust Vertex Labeling



R(1,6)=1
R(2,6)=3

→ relax d = 1

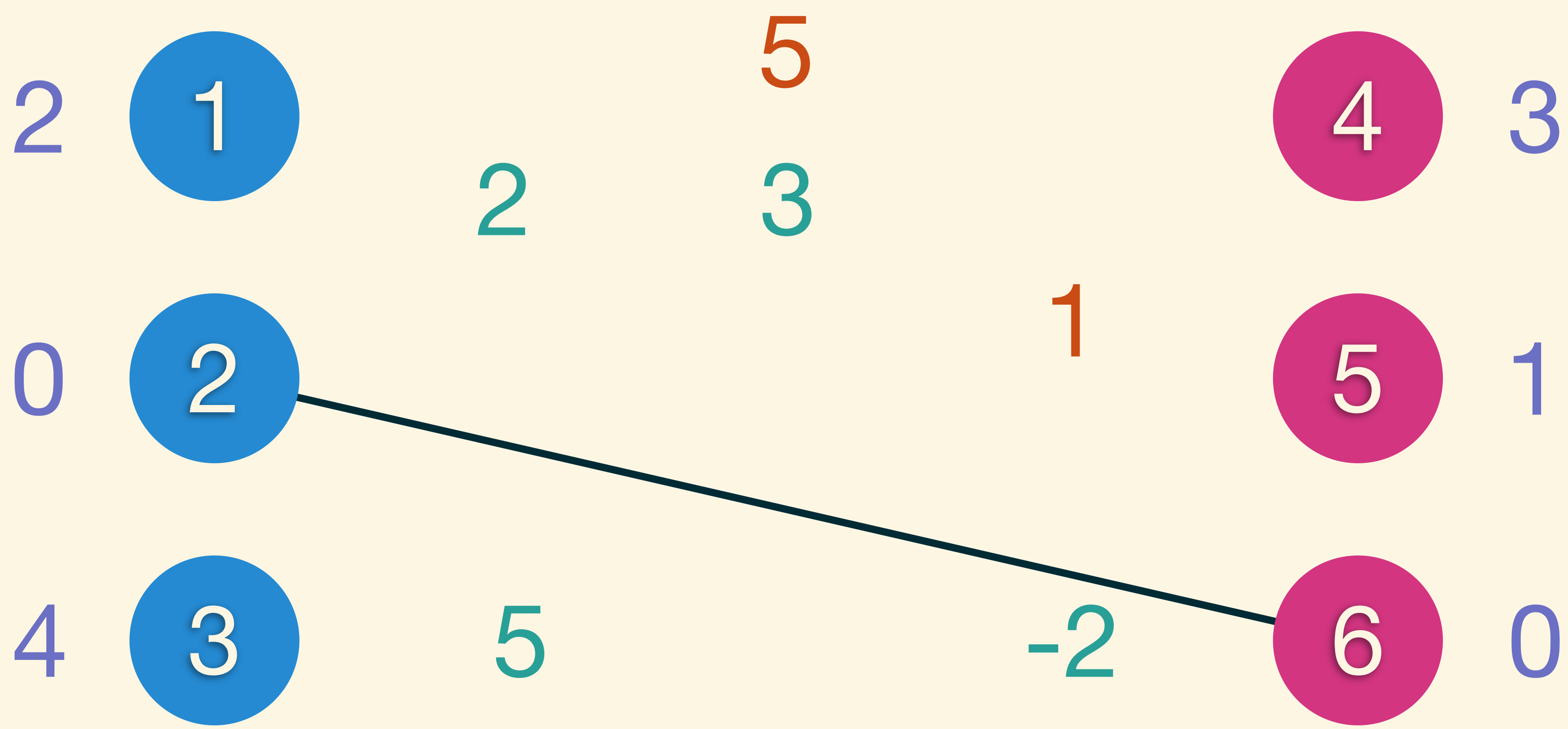
Adjust Vertex Labeling



`l(x)` subtract value d.

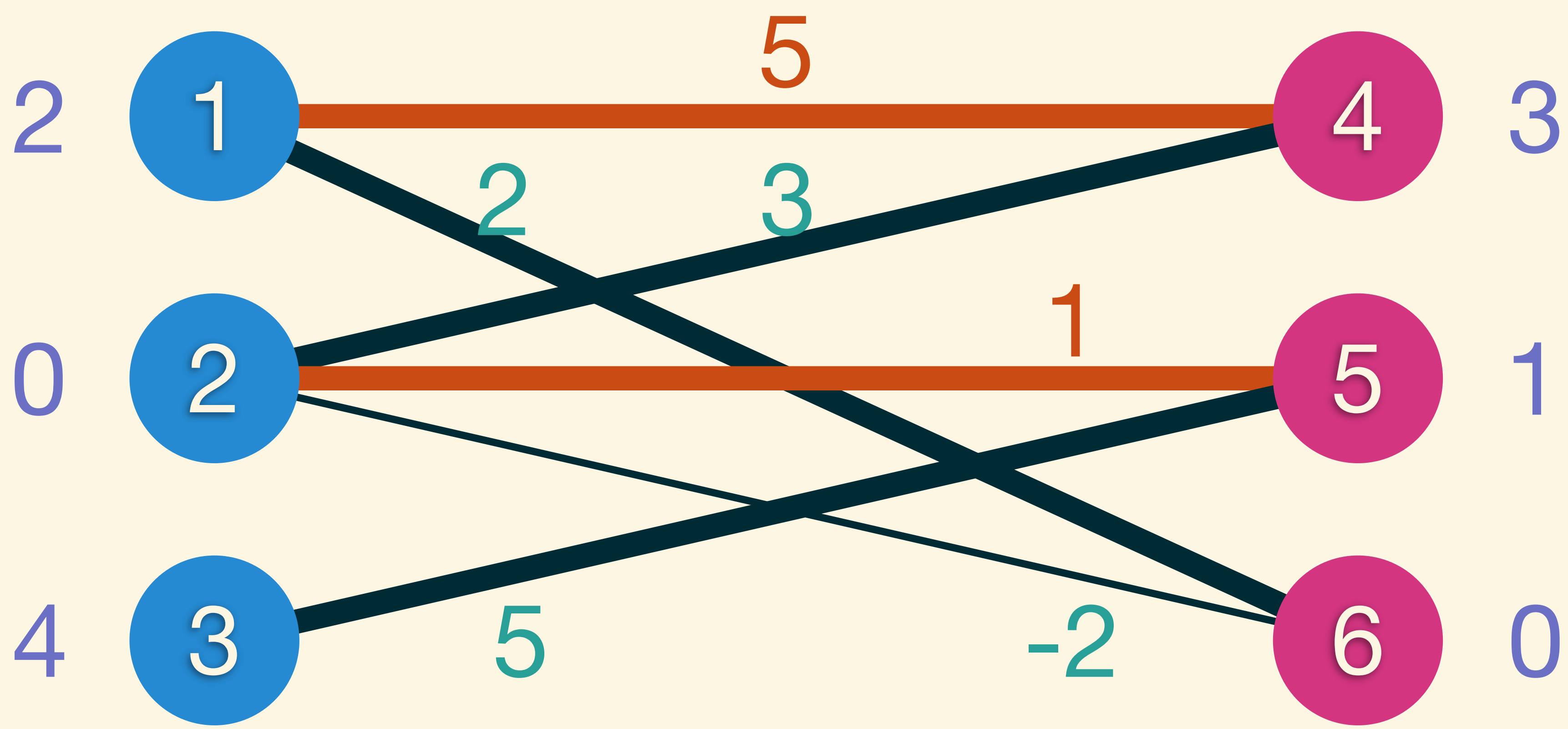
`l(y)` add value d.

Continue to Find Augmenting Path



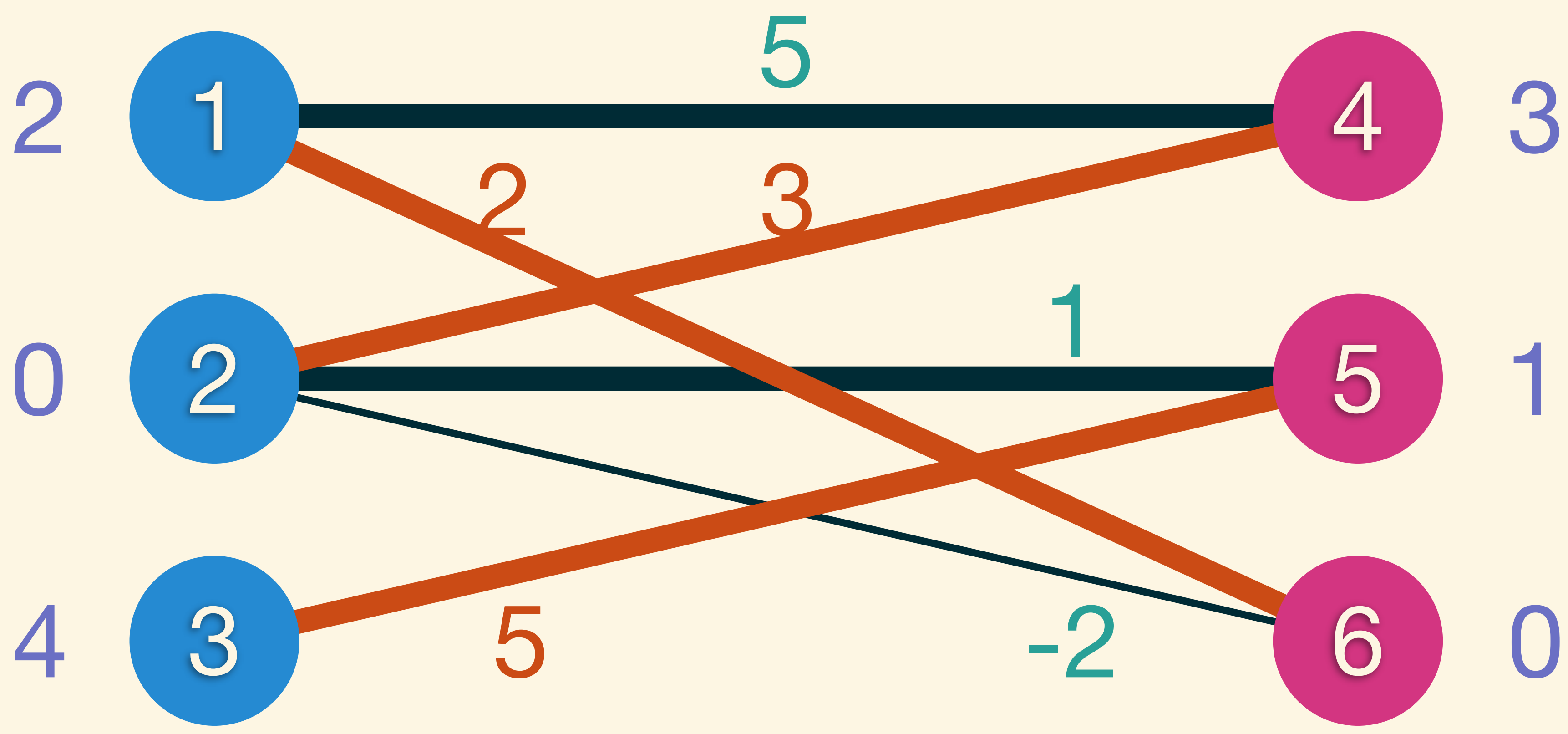
[3 - 5 - 2 - 4 - 1 - 6] is an augment path with admissible edges.

Continue to Find Augmenting Path



[3 - 5 - 2 - 4 - 1 - 6] is an augment path with admissible edges.

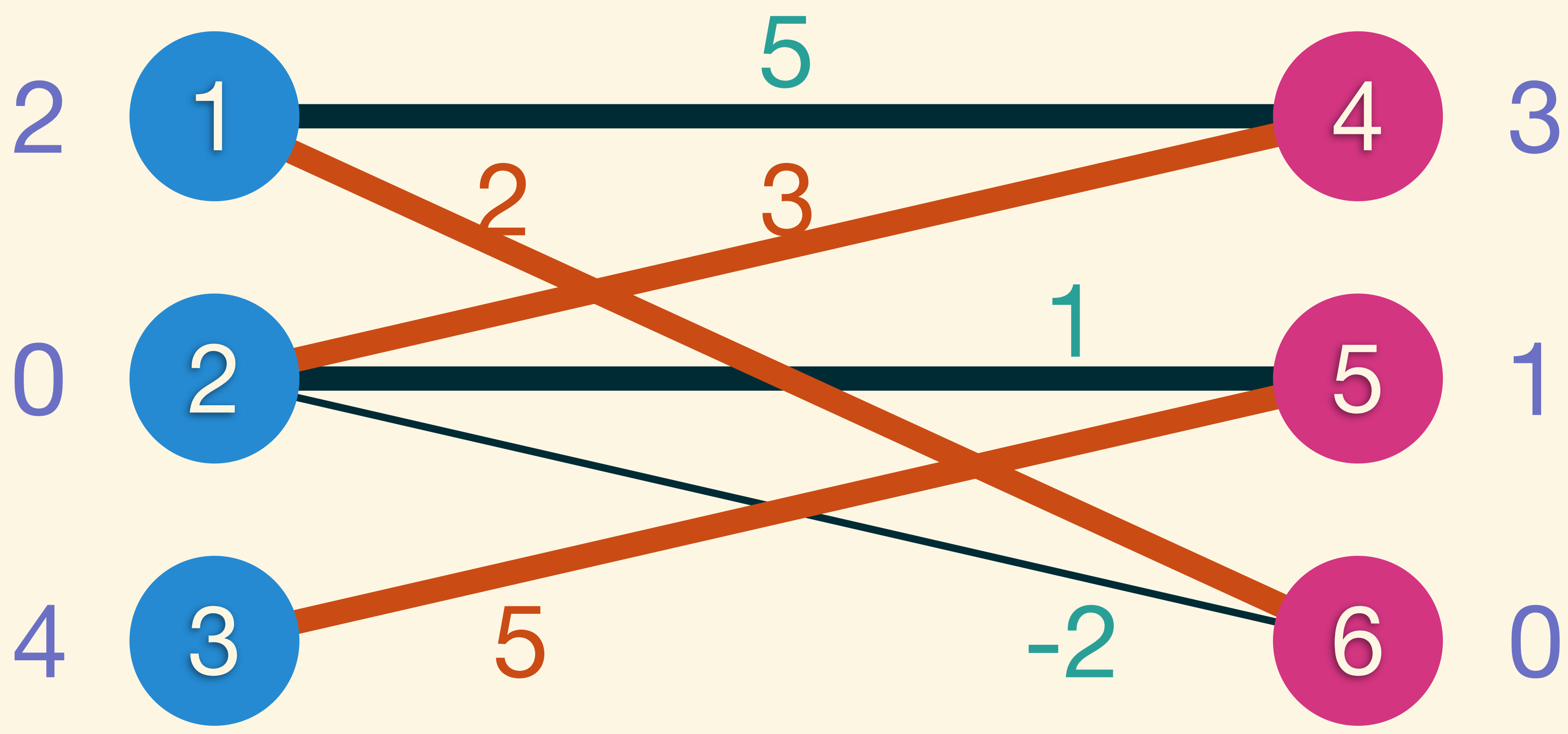
Continue to Find Augmenting Path



Reverse it!

Current Weight = 10

Maximum Weight Matching = 10



Algorithm

1. Initial vertex labeling to fit $l(x)+l(y) \geq w(x,y)$
2. Find **augmenting paths composed with admissible edges** from all vertices in one side.
3. If no augmenting path exists, **adjust vertex labeling until the augmenting path found**.
4. If augmenting path exists, continue to find next augmenting path.
5. Repeat above step until there no augmenting path exists.
6. Calculate the sum of weight on matching edges.

Practice Now

[POJ 2195 - Going Home]()

Problem List

UVa 670
UVa 753
UVa 10080
UVa 10092
UVa 10243
UVa 10418
UVa 10984
POJ 3565

Reference

- http://en.wikipedia.org/wiki/Bipartite_graph
- http://en.wikipedia.org/wiki/Matching_(graph_theory)
- http://www.flickr.com/photos/marceau_r/5244129689
- http://www.sanfilippo-chianti.it/offerta-svalentino.html
- http://www.csie.ntnu.edu.tw/~u91029/Matching.html

Thank You for Your Listening.