Ojlerovi grafovi

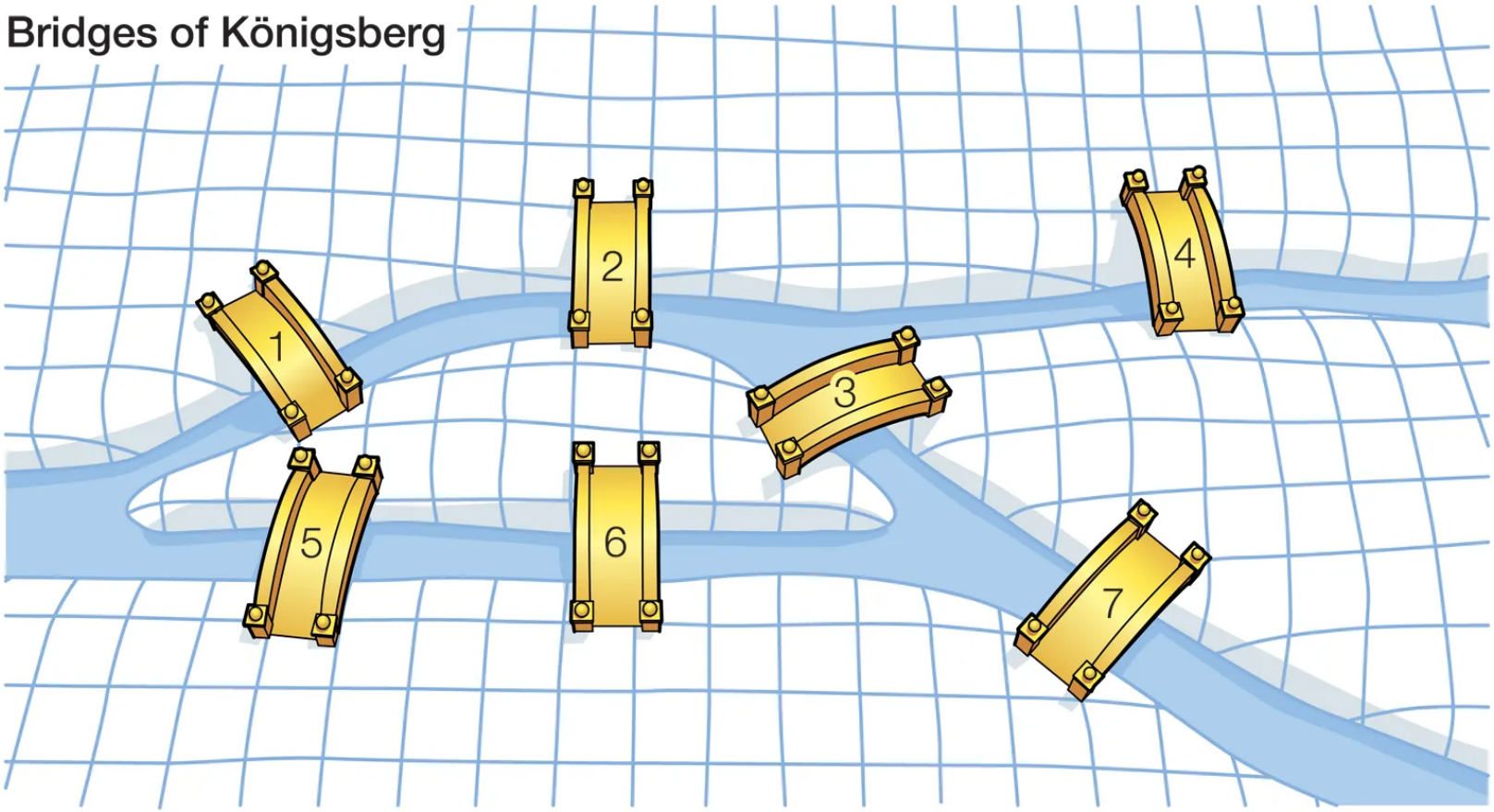
Bogdan Ljubinković, Miljan Jokić, Dalibor Nikolić, Lazar Jović, Anastazija Petrov, Marko Djordjević, Aleksa Nenadović i Meris Bilalović

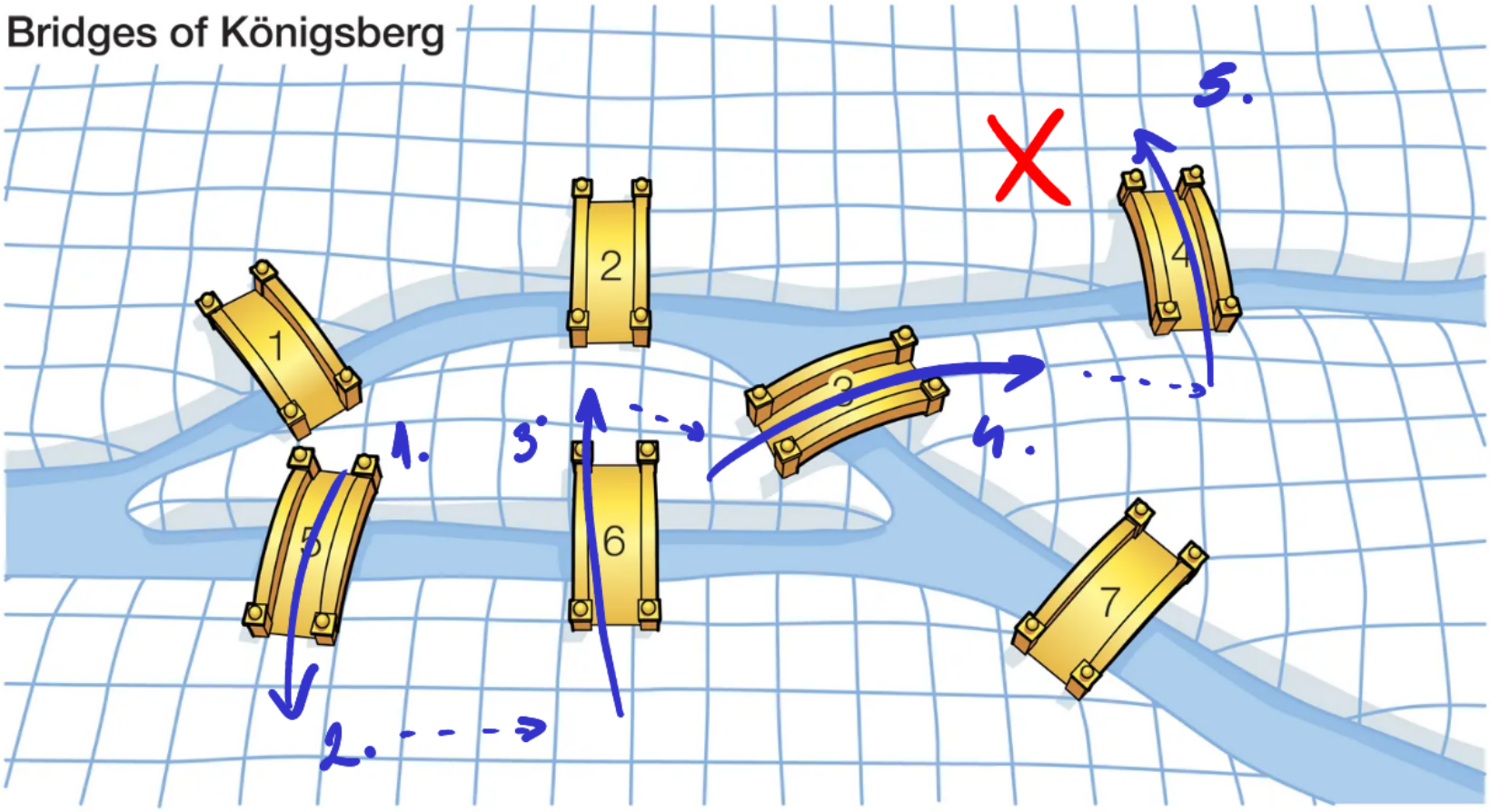
Oktobar 2024, FTN

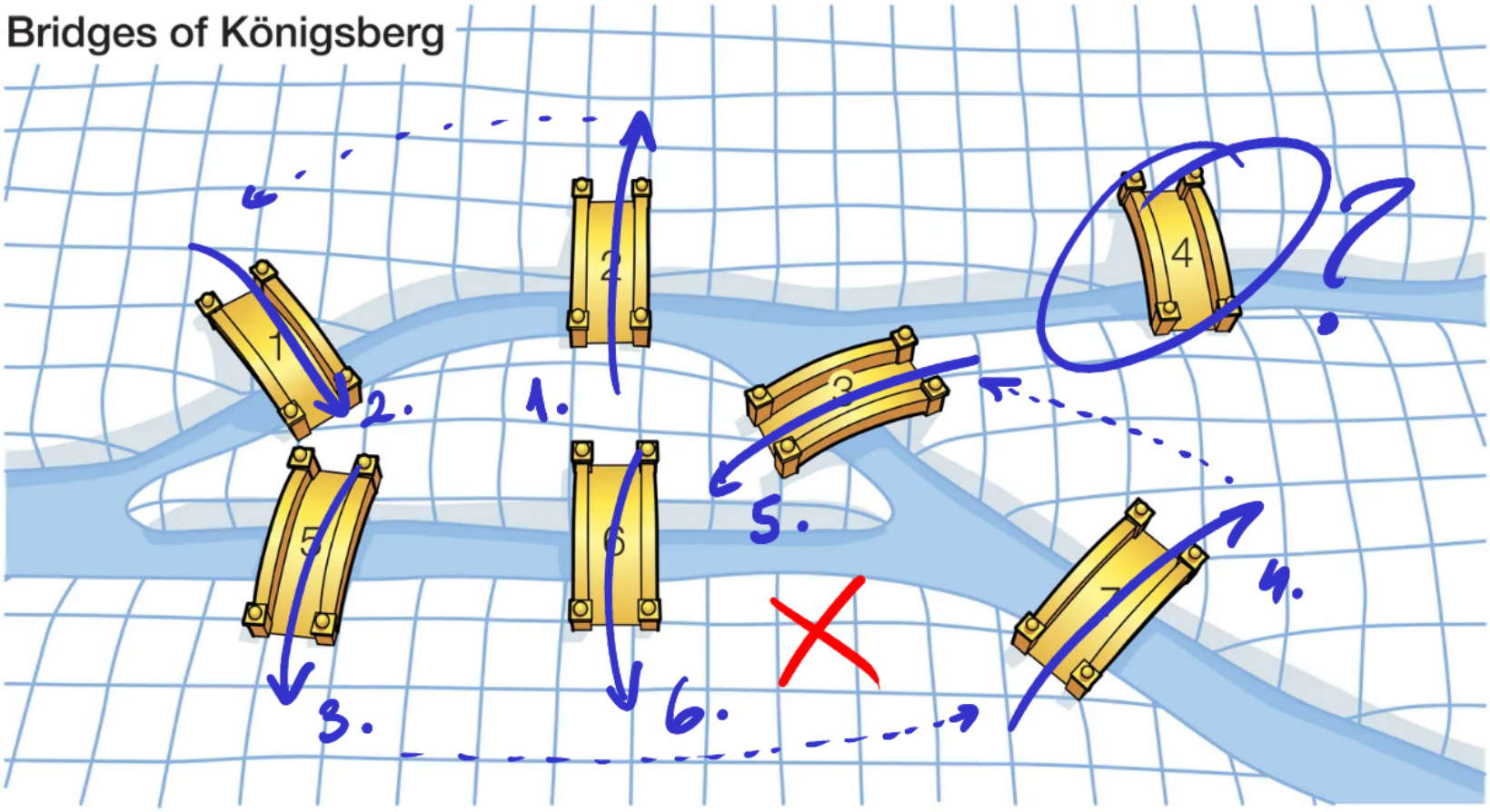
Sadržaj

Sedam mostova Kenigsberga (Kalinjingrada)

Problem je rešio 1735. godine švajcarski matematičar Leonard Ojler. Kenigsberg (danas Kalinjingrad u Rusiji) je bio grad u Pruskoj, kroz koji prolazi reka Pregolja. U jednom delu grada reka obilazi dva ostrva. Tokom izgradnje grada, izgradjeno je ukupno sedam mostova koja povezuju ostrva sa levom i desnom obalom reke i jedno sa drugim. Problem je bio odrediti šetnju po gradu, tako da se svaki most predje tačno jednom i da se vratimo odakle smo krenuli.

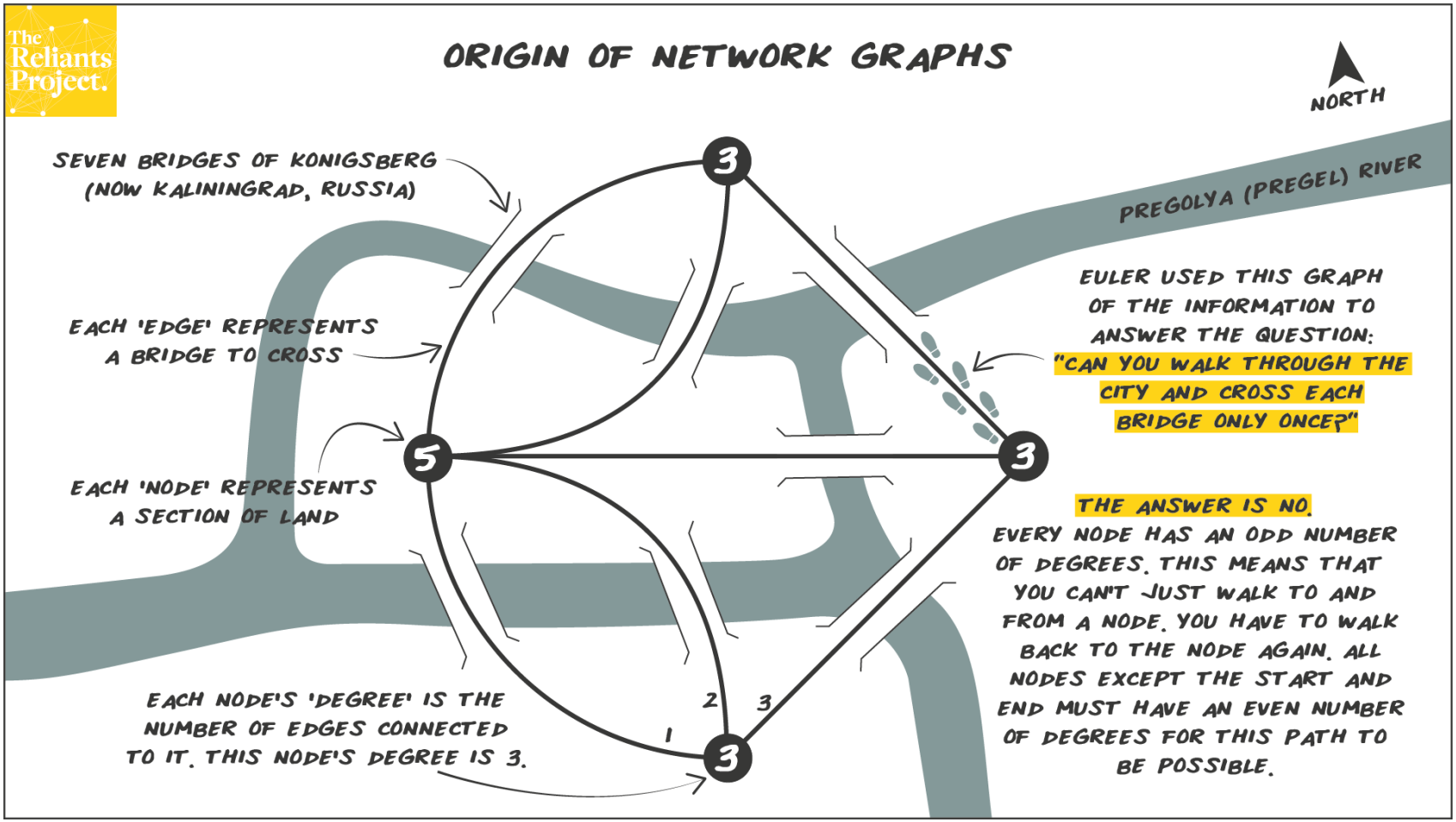



Bridges of Königsberg
1
2
3
4
5
6
7
1.
2.
3.
4.
5.


Bridges of Königsberg
1
2
3
4
5
6
1.
2.
3.
4.
5.
6.
7.

Ključni doprinos

Ključni doprinos u rešavanju ovog problema predstavlja grafovska reprezentacija datog problema. Ojler je problem dalje rešio razmatranjem parnosti broja grana koje izlaze iz pojedinačnih čvorova. Tako je konstatovao da je nemoguće napraviti traženu šetnju kroz grad.
U nastavku dajemo formalnu definiciju Ojlerovog grafa, koja u opštem slučaju važi i za grafove sa paralelnim granama.


The Reliants Project.
ORIGIN OF NETWORK GRAPHS
NORTH
SEVEN BRIDGES OF KONIGSBERG (NOW KALININGRAD, RUSSIA)
PREGOLYA (PREGEL) RIVER
EACH 'EDGE' REPRESENTS A BRIDGE TO CROSS
EULER USED THIS GRAPH OF THE INFORMATION TO ANSWER THE QUESTION: "CAN YOU WALK THROUGH THE CITY AND CROSS EACH BRIDGE ONLY ONCE?"
EACH 'NODE' REPRESENTS A SECTION OF LAND
THE ANSWER IS NO.
EVERY NODE HAS AN ODD NUMBER OF DEGREES. THIS MEANS THAT YOU CAN'T JUST WALK TO AND FROM A NODE. YOU HAVE TO WALK BACK TO THE NODE AGAIN. ALL NODES EXCEPT THE START AND END MUST HAVE AN EVEN NUMBER OF DEGREES FOR THIS PATH TO BE POSSIBLE.
EACH NODE'S 'DEGREE' IS THE NUMBER OF EDGES CONNECTED TO IT. THIS NODE'S DEGREE IS 3.
3
5
3
3
1
2
3

Ojerov put (staza) i Ojlerova tura

- **Definicija Ojerovog puta**: Neka je neusmeren multigraf G bez petlji (sme da sadrži paralelne grane). Tada za šetnju u grafu G kažemo da je Ojlerov put ako:
  - Obilazi sve čvorove grafa G.
  - Grane se ne ponavljaju.
  - Sve grane G su sadržane u posmatranoj šetnji.

- **Definicija Ojerove ture**: Za Ojlerov put kažemo da je Ojlerova tura ako su prvi i poslednji čvor tog "puta" isti.

Ojerov i polu Ojlerova graf

- **Definicija polu Ojerovog grafa**: Za neusmeren multigraf G, bez petlji, kažemo da je polu Ojlerov graf ako sadrži Ojlerov put:
- **Definicija Ojerovog grafa**: Za neusmeren multigraf G, bez petlji, kažemo da je Ojlerov graf ako sadrži Ojlerovu turu.
- Iz činjenica da je svaka Ojlerova tura ujedno i Ojlerov put možemo zaključiti da je svaki Ojlerov graf ujedno i polu Ojlerov graf.

Potreban i dovoljan uslov Ojlerovog grafa

- **Teorema**: Neka je graf G povezan neusmeren graf. Tada je G Ojlerov graf ako i samo ako je svaki čvor G parnog stepena.

- **Dokaz**: ($\Longrightarrow$) Uzmimo da je G Ojlerov graf. Kako je G Ojlerov graf postoji Ojlerova tura $v_1 e_1 v_2 e_2 ... u_n e_n u_{n+1}$. Neka se proizvoljan čvor v pojavljuje l puta ($l + 1$ ako je v_1) u Ojlerovoj turi. Stepen ovog čvora je tačno $2l$ jer za svako posećivanje čvora tokom ture mora postojati jedna ulazna i izlazna grana (sem u slučaju poslednjeg tj. prvog čvora), a kako je G neusmeren obe grane utiču na stepen čvora.

Potreban i dovoljan uslov Ojlerovog grafa

- **Teorema**: Neka je graf G povezan neusmeren graf. Tada je G Ojlerov graf ako i samo ako je svaki čvor G parnog stepena.

- **Dokaz**: ($\Longleftarrow$) Posmatrajmo najdužu stazu $v_1 e_1 v_2 e_2 \ldots u_n e_n u_{n+1}$ grafa G. Pretpostavimo da su 1. i poslednji čvor različiti. Kako je stepen čvora v_1 paran, a broj grana incidentnih sa v_1 koje su sadržane u stazi neparan mora postojati takva grana e koja nije sadržana u stazi čime bi ova staza mogla da se produži dodavanjem te grane.
  Pretpostavimo zatim da postoje grane ili čvorovi koji nisu sadržani u ovoj stazi. U oba slučaja, zbog povezanosti grafa, staza više ne bi bila najduža jer se može produžiti dodavanjem grane/čvora.

Potreban i dovoljan uslov polu Ojlerovog grafa

- **Teorema**: Neka je graf G povezan neusmeren graf koji nije Ojlerov. Tada je G polu Ojlerov graf ako i samo ako postoje tačno 2 čvora neparnog stepena.

- **Dokaz**: Ako predpostavimo da je graf polu Ojlerov onda postoji Ojlerov put u tom grafu. Svi čvorovi tog puta imaju paran broj incidentnih grana sem 1. i poslednjeg. Kako Ojlerov put obuhvata sve grane zasigurno znamo da ne postoji još neka grana izvan "puta" čime je stepen ta 2 čvora neparan. Posmatrajmo suprotan smer tj. povezan neusmeren graf koji nije Ojlerov i svi sem 2 čvora su parnog stepena. Dodavanjem grane nove grane e incidentne sa 2 čvora neparnog stepena kreiramo novi graf G' koji je Ojlerov po prethodnoj teoremi. Uklanjanjem grane e iz Ojlerove ture G' po definiciji dobijamo Ojlerov put koji je u potpunosti sadržan u grafu G čime je on polu Ojlerov graf.

Uvod u Priferov kod

Priferov kod je način kodiranja stabala sa n čvorova pomoću niza od $n-2$ brojeva. Svakom stablu odgovara jedinstveni Priferov kod, što omogućava efikasnu rekonstrukciju stabla.
Postupak generisanja koda uključuje:

- Pronalaženje lista sa najmanjim brojem i zapisivanje povezanog čvora u niz.
- Uklanjanje lista iz stabla.
- Ponavljanje dok ne ostanu samo dva čvora.

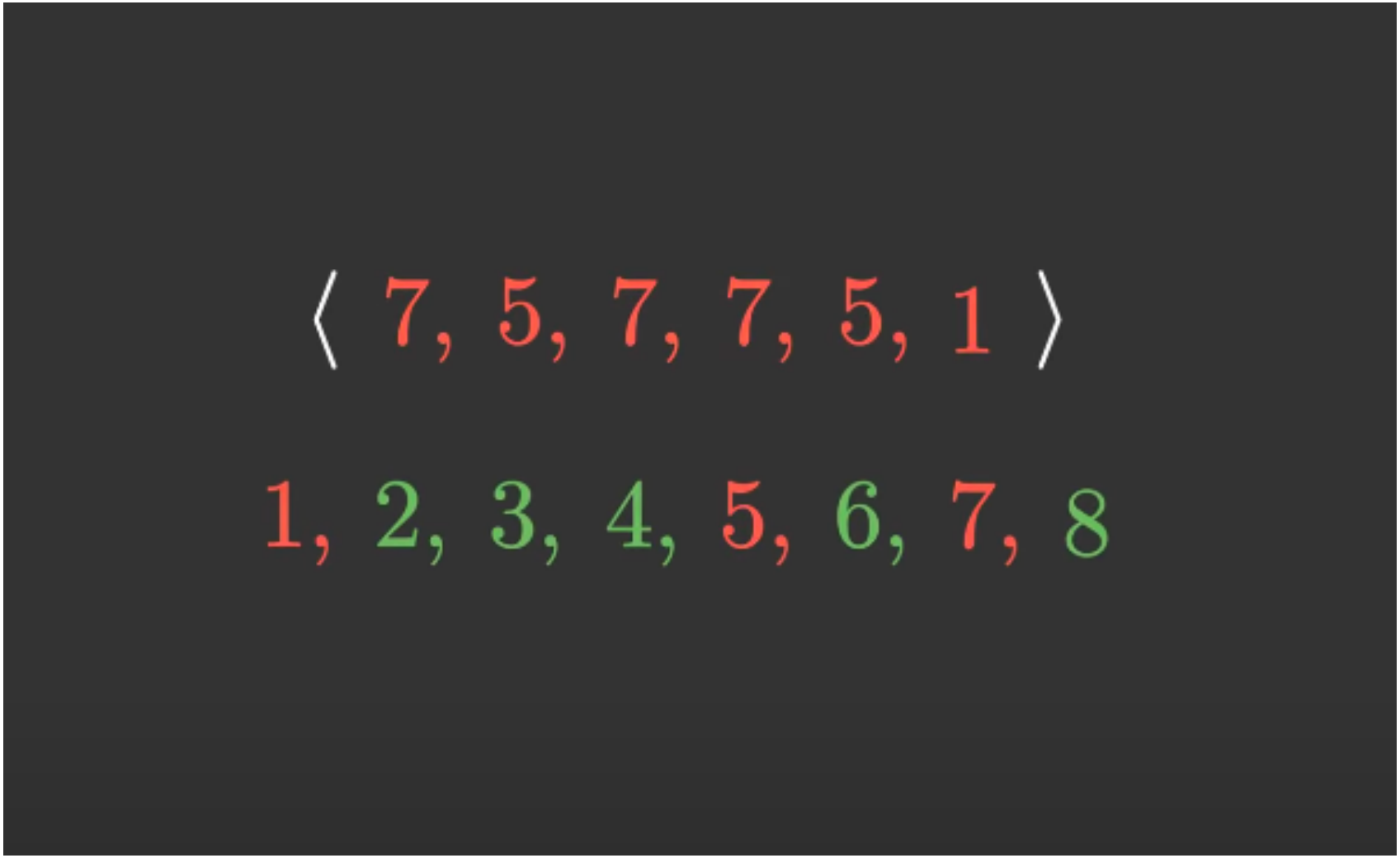


Figure: Primer Pruferovog koda, 2 - 3 - 4 - 6 i 8 su listovi

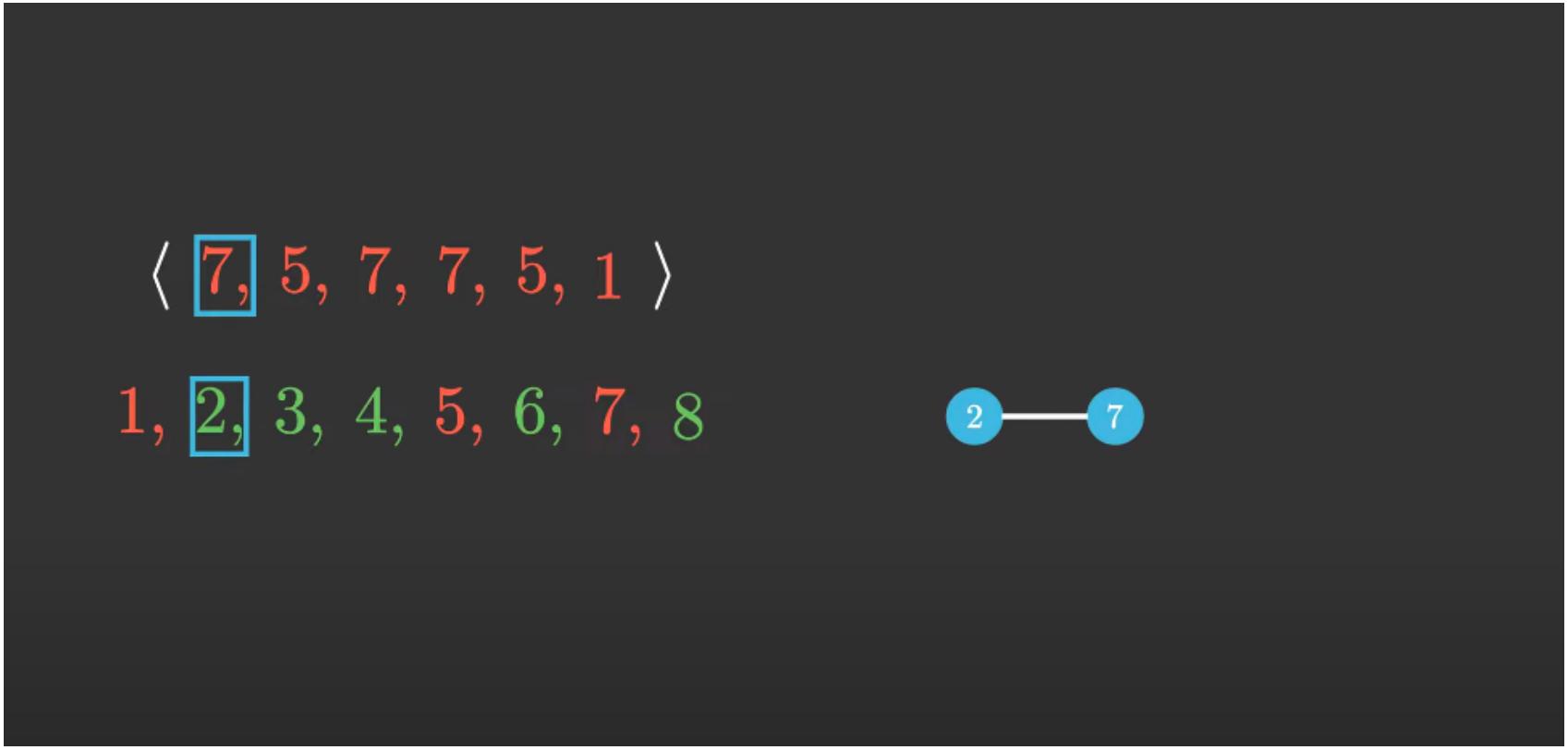


Figure: Pozicioniramo se na 7, iteriramo od 1 do 8 dok ne naidjemo na list. Povezujemo 2 i 7. Eliminišemo 2 i 7.

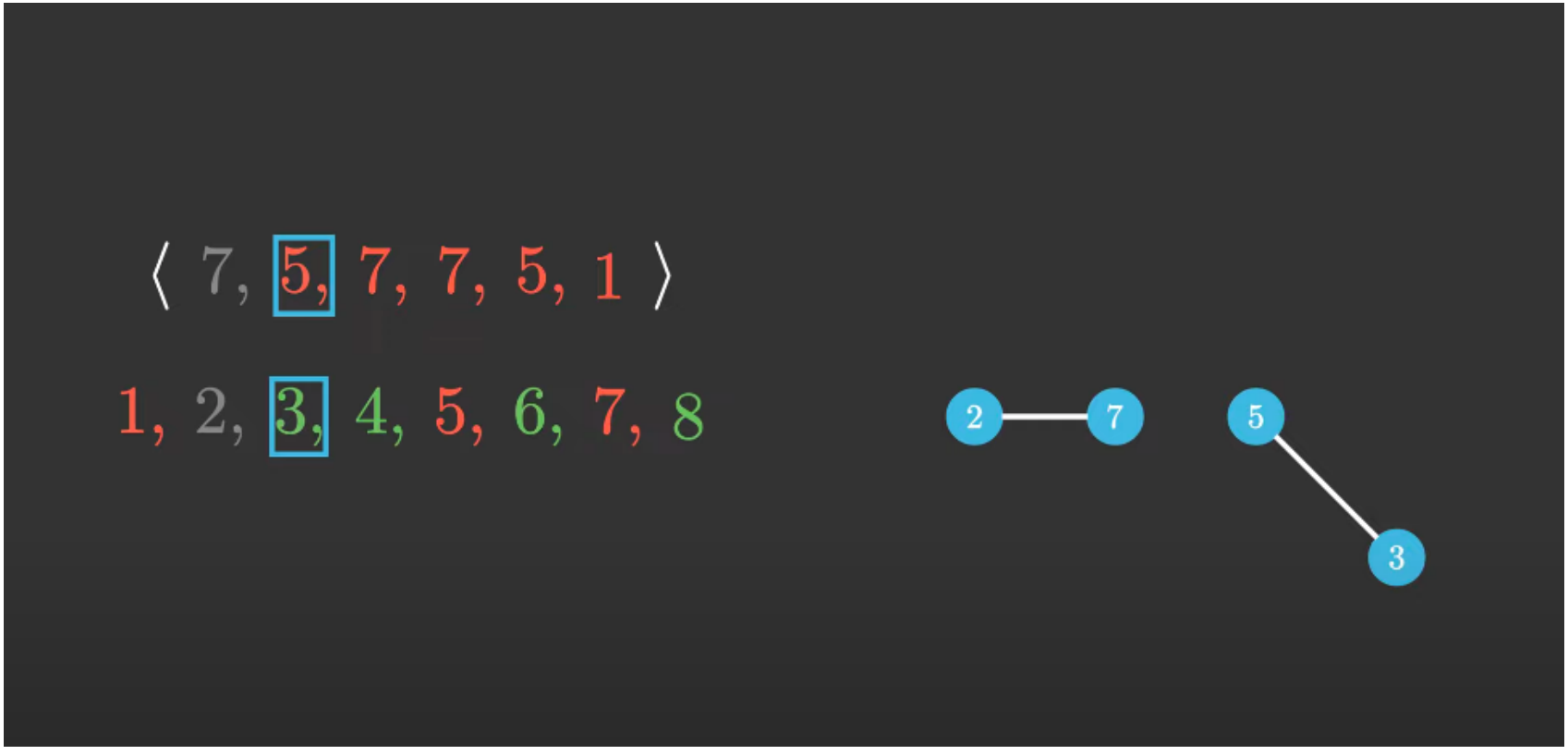


Figure: Prelazimo na 5, opet iteriramo od 1 do 8. Povezujemo 5 i list 3, eliminišemo oba.

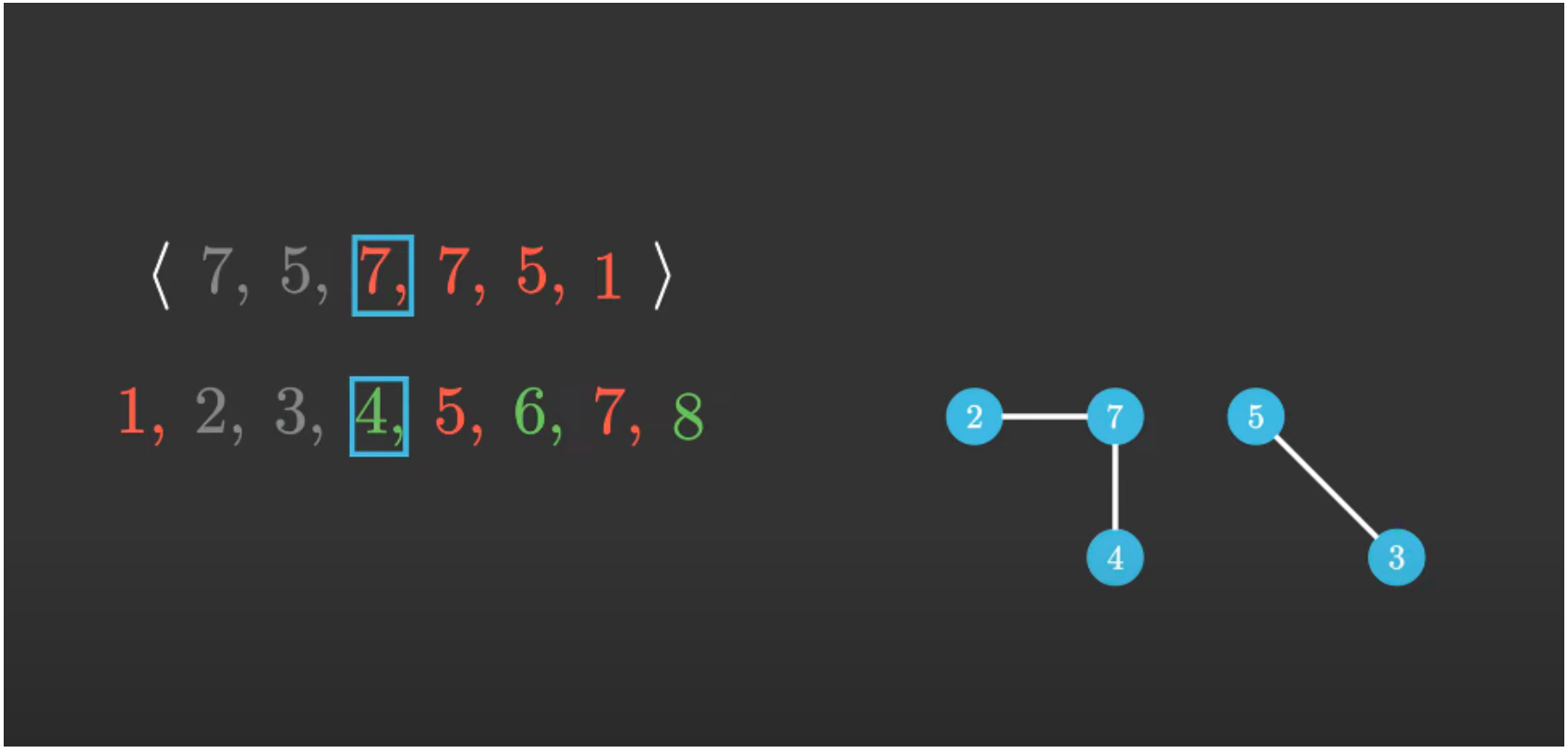


Figure: Prelazimo na 7, iteriramo od 1 do 8 kroz čvorove opet. Nailazimo na list 4, povežemo sa 7, eliminišemo oba.

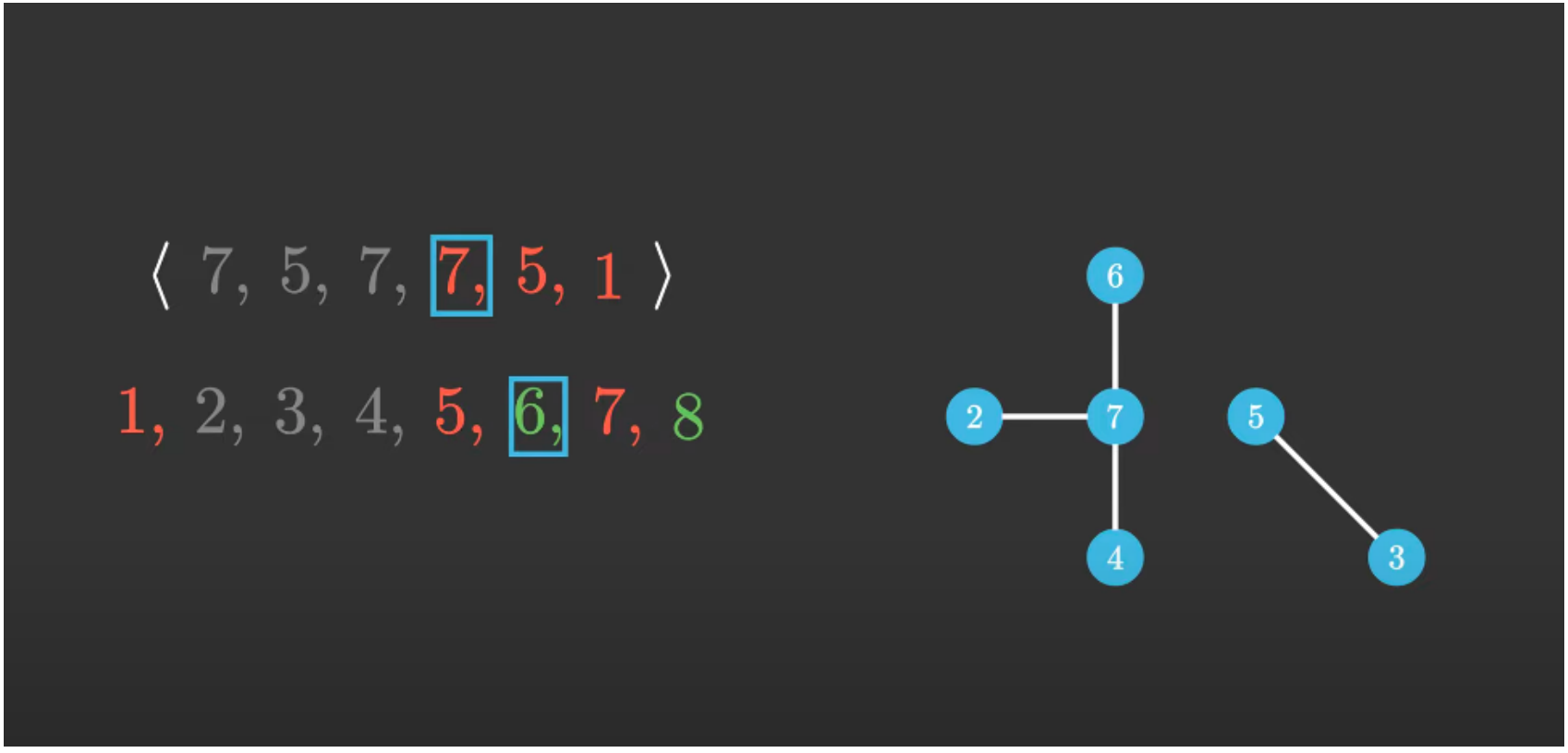


Figure: Prelazimo na 7, iteriramo od 1 do 8, nailazimo na list 6, povezujemo sa 7 i oba eliminišemo

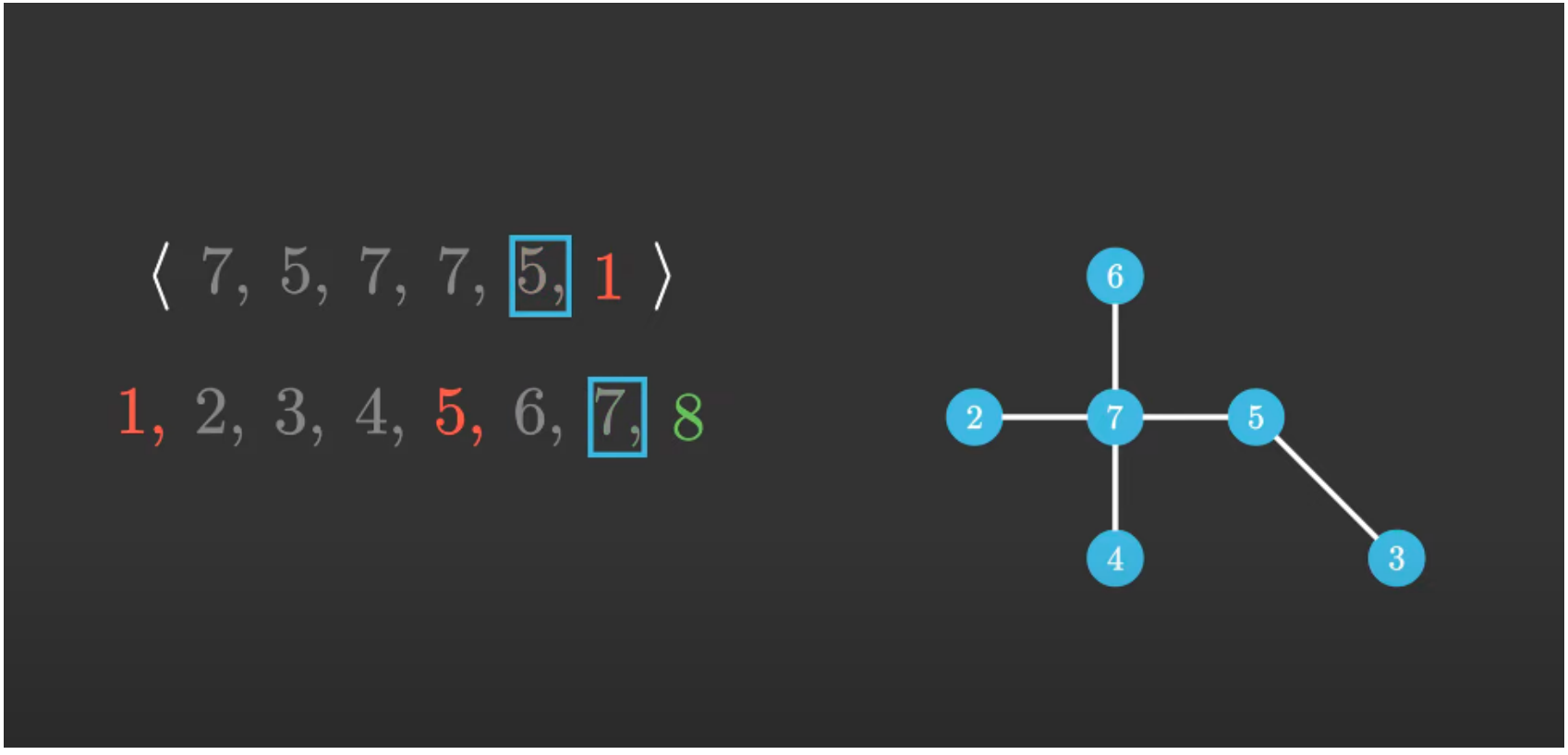


Figure: 7 je sada iz koda kompletno eliminisano! Sada ga tumačimo kao list medju čvorovima. Prelazimo na 5, iteriramo od 1 do 8, nailazimo na 7, povežemo ih i eliminišemo oba.

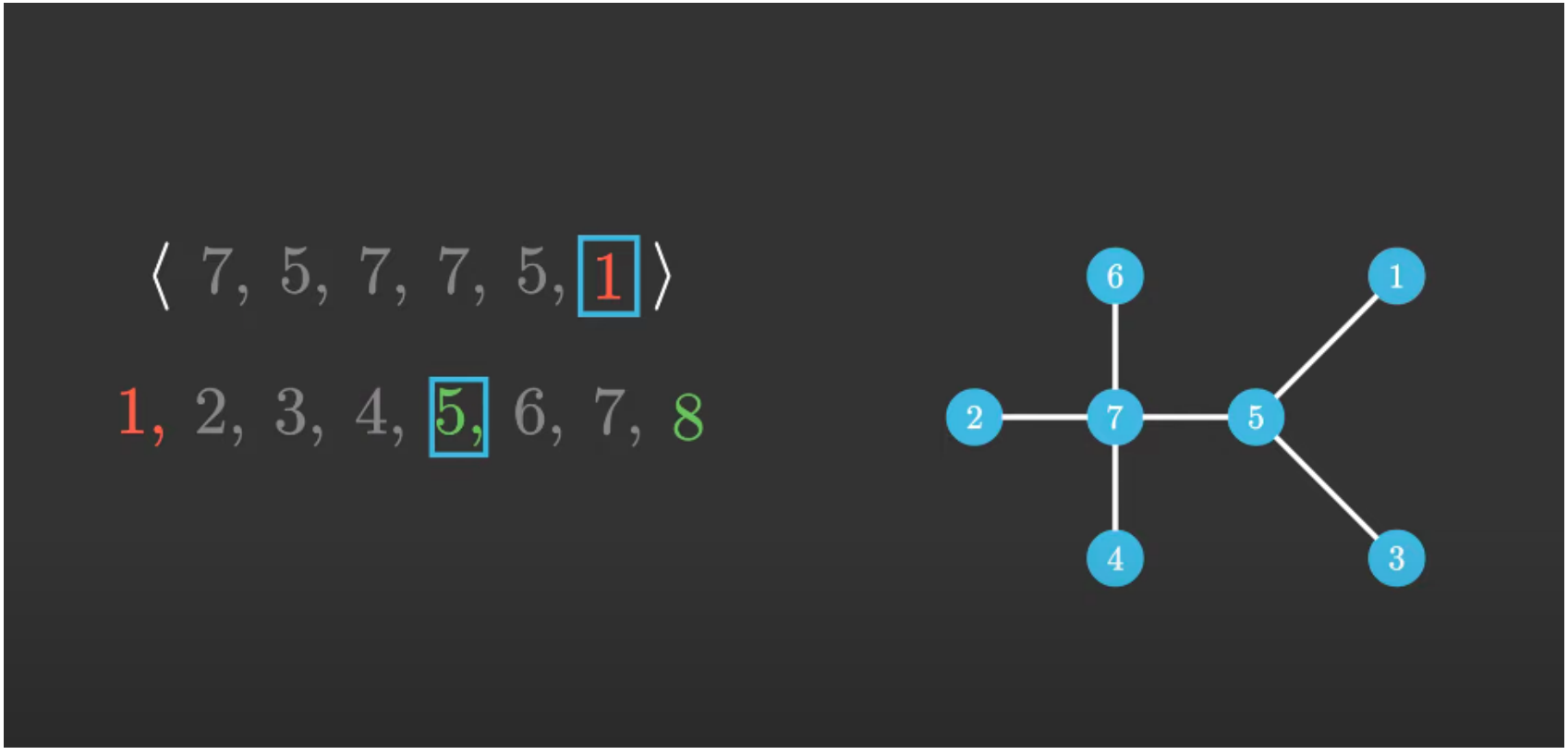


Figure: Sada je 5 kompletno eliminisano iz koda! Tumačimo ga medju čvorovima kao list. Prelazimo na 1, iteriramo od 1 do 8, nailazimo na 5, povezujemo ih i eliminišemo oba.

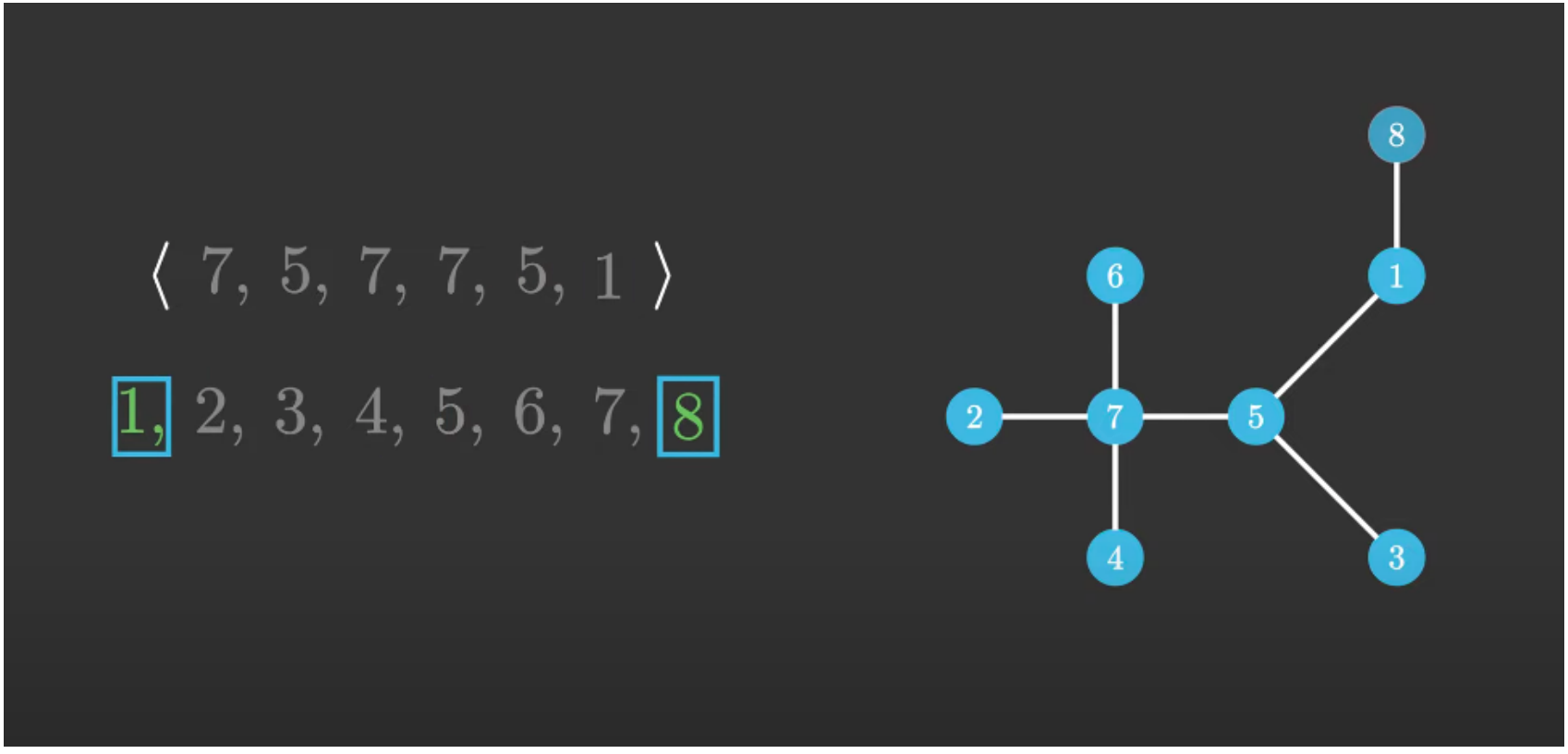


Figure: Na kraju nam ostaju 2 lista, samo ih povežemo.

Algoritam za odreivanje Ojlerove ture

Za odreivanje Ojlerove ture koristićemo **Flerijev algoritam**. Ovaj algoritam omogućava pronalaženje Ojlerove ture ili staze u grafovima kroz sekvencijalno uklanjanje grana i proveru mostova.

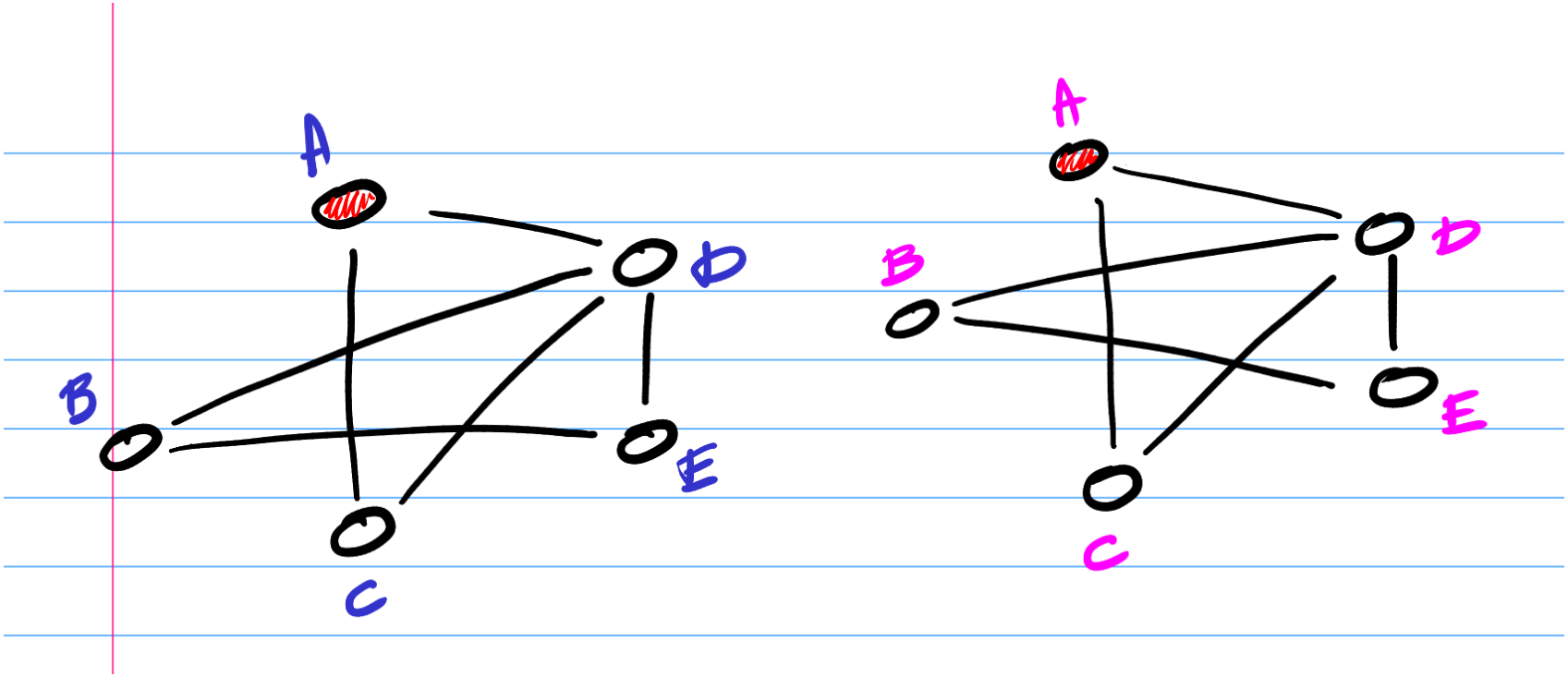


Figure: Sa leve strane je graf koji obilazimo, sa desne strane je duplikat nad kojim eksperimentišemo (nije bitan toliko). Biramo bilo koji čvor. Odabrali smo A...

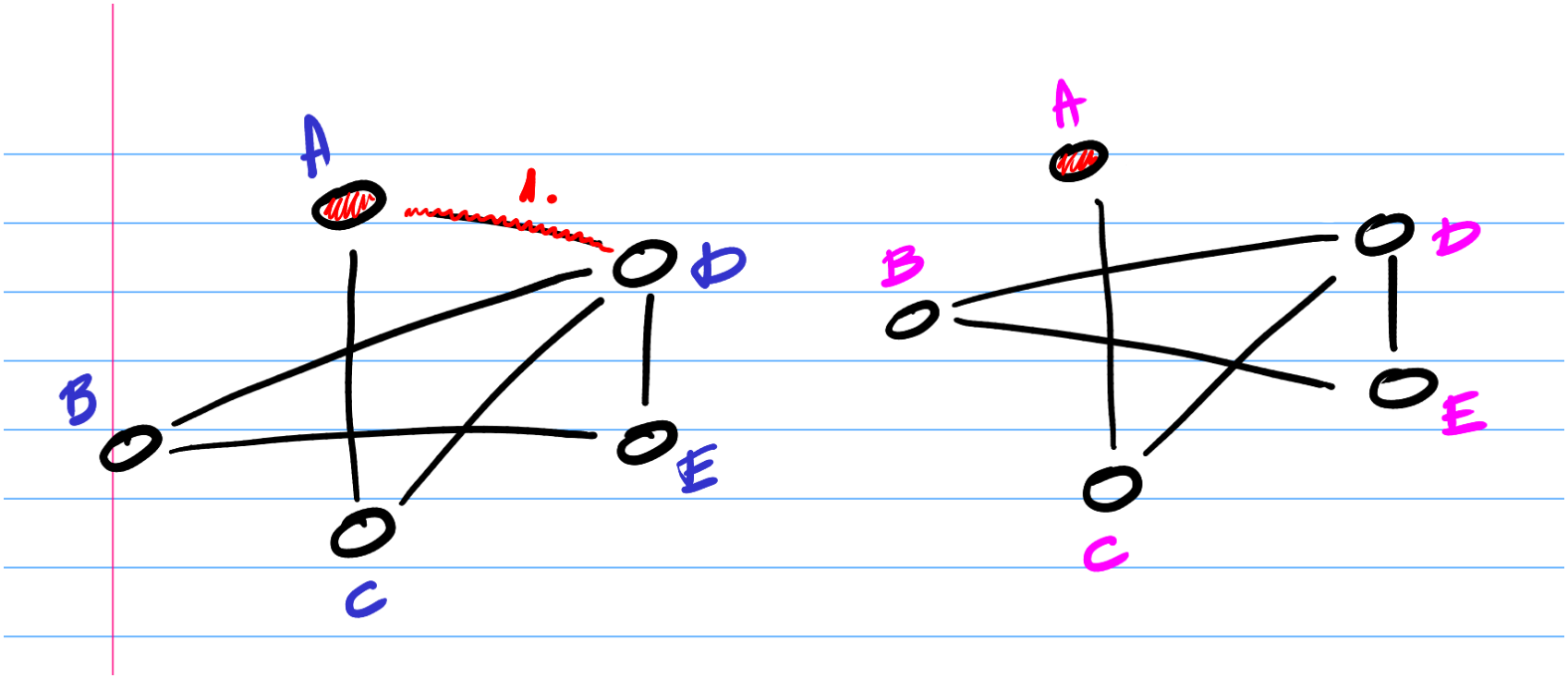


Figure: Biramo jednu granu koja izlazi iz čvora pod jednim uslovom. Kada se ta grana ukloni, ne sme podeliti graf na 2 dela. Odabrali smo crvenu granu sa leve strane, desno je uklanjamo.

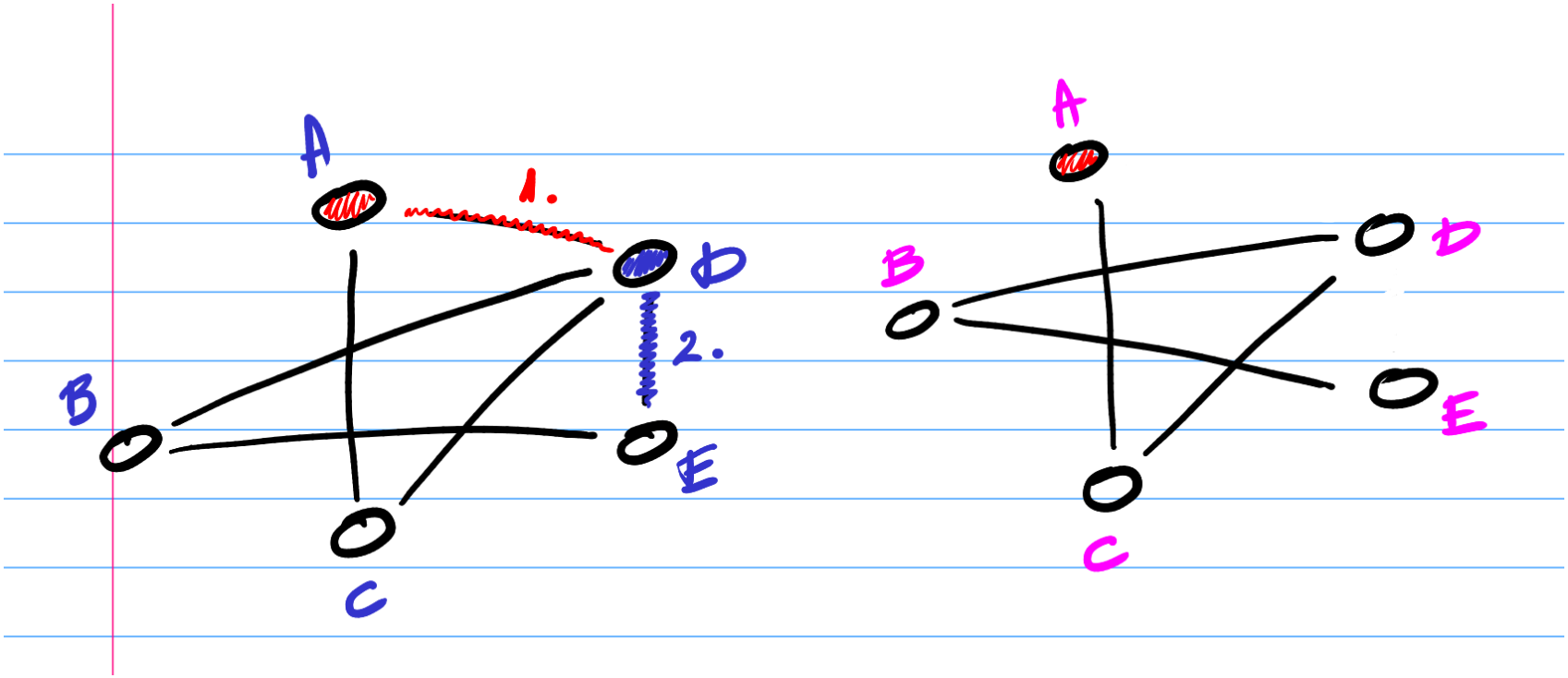


Figure: Eliminišemo jednu granu koja izlazi iz čvora D pod istim uslovom. Eliminišemo granu DE... Šta bi se desilo da smo odabrali granu DC?

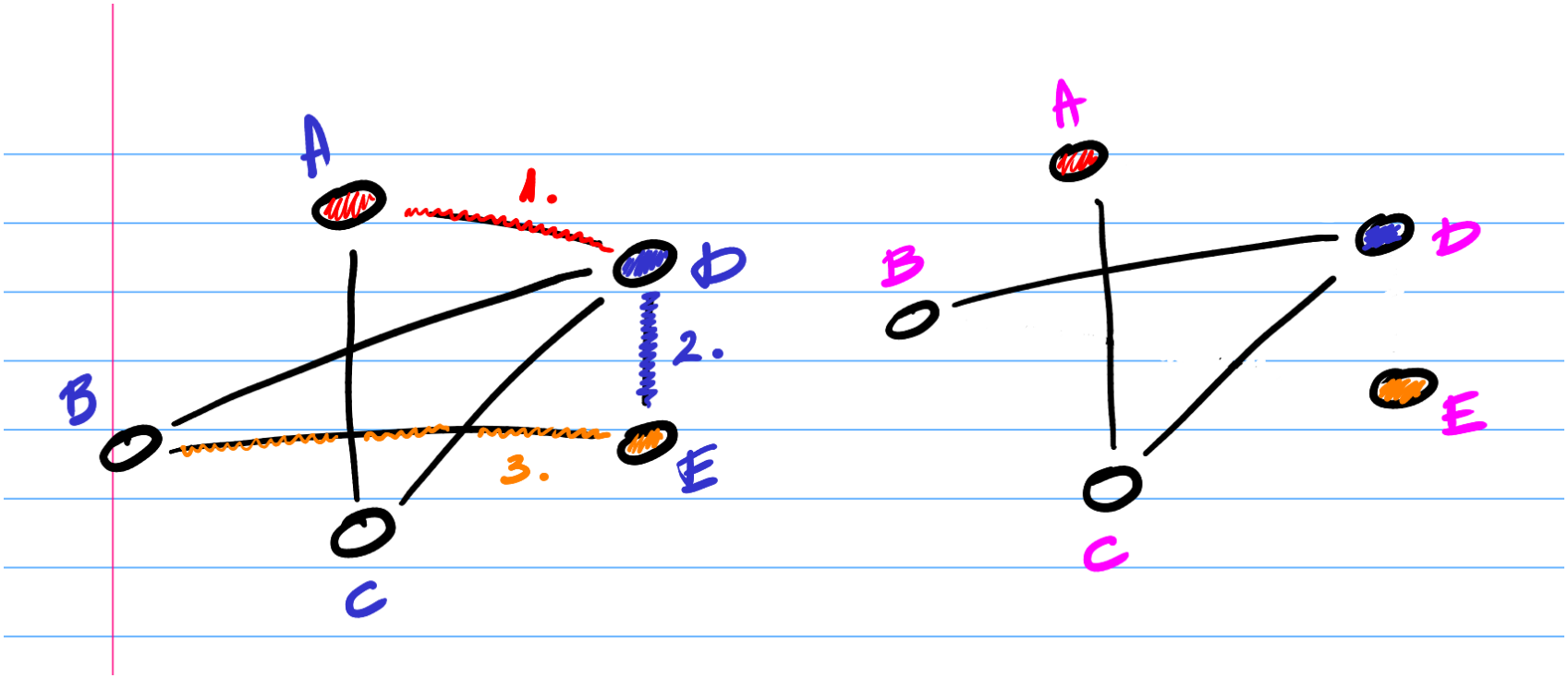
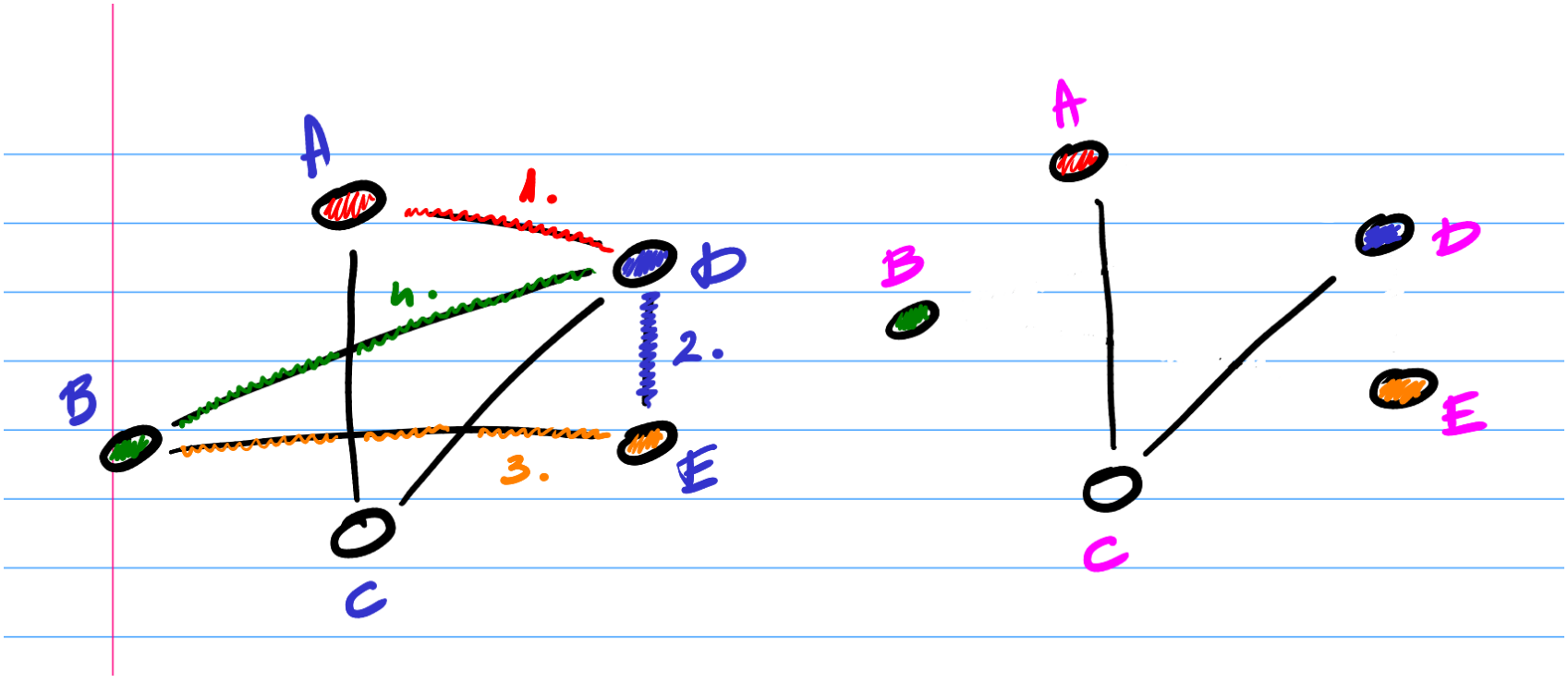
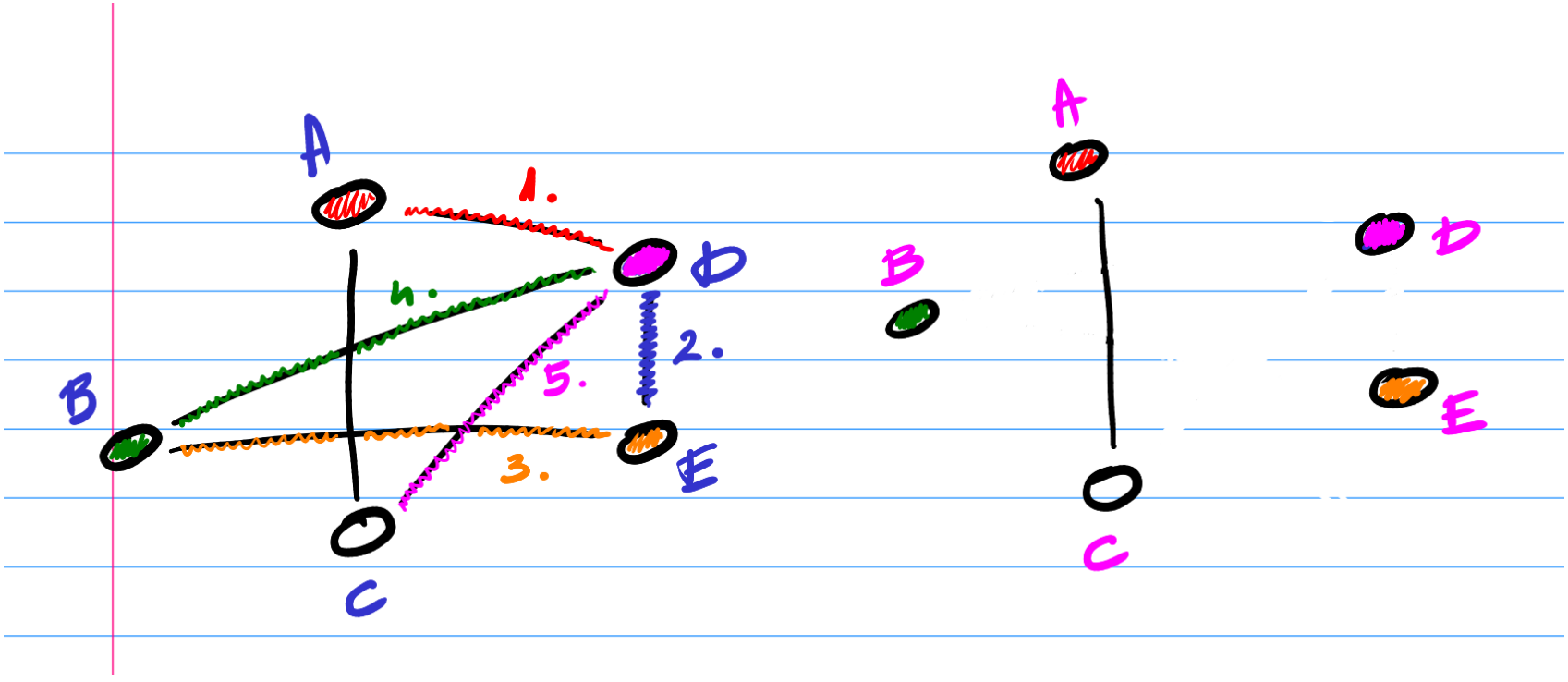


Figure: Do samog kraja procesa nemamo više opcija, treba napomenuti samo da smo ovde sigurni da ima Ojlerova tura, neće uvek biti tako...


A
1.
D
4.
2.
B
3.
E
C
B
A
D
E
C


A
1.
D
4.
B
2.
5.
3.
E
C
A
B
D
E
C

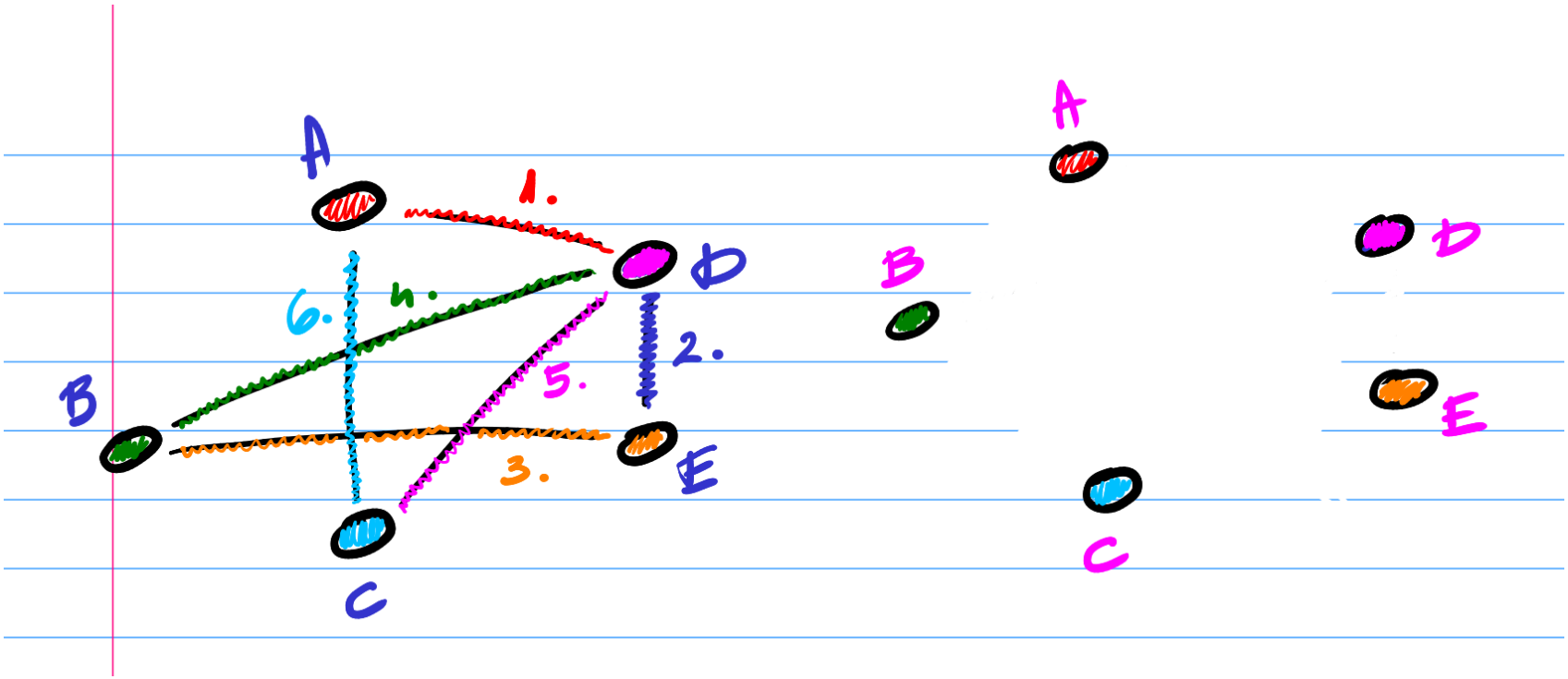


Figure: Kraj procesa. Našli smo turu A-D-E-B-D-C-A

```python
class Vertex():
    def __init__(self, value):
        self.value = value
        self.incidentEdges = []

    def __str__(self):
        return self.value

class Edge():
    def __init__(self, name, vertex1: Vertex, vertex2: Vertex):
        self.start = vertex1
        self.end = vertex2
        self.name = name

    def __str__(self):
        return self.name
```

Figure: Kod za pronalaženje Ojlerove ture, 1. deo

```python
class Graph():
    def __init__(self, vertices, edges):
        self.vertices = vertices
        self.edges = edges

    def add_edge(self, name, vertex1: Vertex, vertex2: Vertex):
        if not(vertex1 in self.vertices and vertex2 in self.vertices):
            return

        newEdge = Edge(name, vertex1, vertex2)
        self.edges.append(newEdge)
        vertex1.incidentEdges.append(newEdge)
        vertex2.incidentEdges.append(newEdge)

    def is_eulerian(self):
        for vertex in self.vertices:
            if len(vertex.incidentEdges) % 2 != 0:
                return False
        return True

    def find_euler_tour(self, startingVertex: Vertex):
        if not(self.is_eulerian()):
            return None

        visited = [startingVertex, startingVertex.incidentEdges[0]]
        currentVertex = startingVertex.incidentEdges[0].end if startingVertex.incidentEdges[0].start == startingVertex else startingVertex.incidentEdges[0].start
        while True:
            visited.append(currentVertex)
            nextEdge = None
            for edge in currentVertex.incidentEdges:
                if not(edge in visited):
                    nextEdge = edge
                    if not(edge.start in visited and edge.end in visited):
                        break
            if nextEdge == None:
                break
            visited.append(nextEdge)
            currentVertex = nextEdge.end if nextEdge.start == currentVertex else nextEdge.start

        return visited
```

Figure: Kod za pronalaženje Ojlerove ture, 2. deo