ALGORİTMA ANALİZİ DÖNEM PROJESİ
SOSYAL AĞ YAPILARI

Adı Soyadı: Büşra Medine GÜRAL

Öğrenci Numarası: 20011038

Mail: medine.gural@std.yildiz.edu.tr

Dersin Eğitmeni: M. Elif KARSLIGİL

Video Linki: -

1. Problem Tanımı

İlgili ödevde, graf şeklindeki bir ağ yapısında toplulukları tespit edebilmek amacıyla BFS algoritmasını da içeren bir algoritma tasarlanması istenmektedir. Öncelikle bir ara kesici değer hesabının yapılması, ardından bu değerlerden maksimum olanın ağdan kaldırılması gerekmektedir. Böylelikle ağ yapısındaki topluluklar tespit edilmiş olacaktır. Algoritmanın sonlanma koşulu ise iki şekilde olmaktadır: Ardışık k tane turdaki topluluk sayıları değişmediğinde algoritma sonlanmalıdır ya da toplulukların herhangi birinde minimum üye sayısı t olmalıdır.

2. Problem Çözümü

Problemin çözüm için ilk olarak BFS algoritmasının ana bileşenleri oluşturulmuştur. Bir kuyruk oluşturulup ilk değerleri ayarlandıktan sonra, başlangıç düğümünden her bir düğüme gidene kadar geçilen en kısa yoldaki düğümler, 'komşuluk' ve 'ziyaret' değerleriyle kontrol edilmiştir. 'Komşuluk' kontrolü matristeki değerin 1 olup olmadığı şeklinde kontrol edilirken, 'ziyaret' kontrolü, bir dizi tutularak 'true' şeklinde değiştirilen değerlerle yapılmıştır. Yol üstündeki düğümlerin kaydedilebilmesi için ise yine bir dizi (parent) kullanılmaktadır, her bir düğüme ulaşırken, ilgili düğümden önceki düğüm bu diziye kaydedilmiştir. Böylelikle geçilen yolların değerleri, parent dizisi sayesinde sürekli güncellenmektedir.

İkinci aşama olarak sonsuz bir döngü içerisinde belirlenen sonlanma koşulları gerçekleşene kadar BFS algoritması tekrarlı olarak çalışmaktadır. Öncelikle yol değerleri hesaplanmakta ve üzerinden en çok geçilen yol ('yol değerleri matrisi'ndeki maksimum değer) tespit edilmektedir. Sonrasında graf matrisinde, bu yol sıfırlanarak kaldırılmaktadır. Ardından tekrar bir BFS algoritmasıyla birlikte toplulukların özellikleri incelenmektedir. Her bir düğümden başlayarak daha önce ziyaret edilmediği koşulda, gidilebilecek maksimum yere kadar ulaşılmakta ve bu aşamada üzerinden geçilen düğümler ve sayıları kaydedilmektedir. Bu şekilde bulunan toplulukların sayıları ve topluluk üyeleri iki ayrı veri yapısında saklanmaktadır.

Son aşamada algoritmanın sonlanma koşulları kontrol edilmektedir. İlk olarak toplulukların ardışık k turda aynı kalıp kalmadığına bakılmaktadır. Bunun için bir önceki iterasyon ile gerçekleşen iterasyonun topluluk sayıları karşılaştırılmaktadır. Aynı olduğu takdirde kontrol değişkeni bir artırılarak girilen k değeri ile karşılaştırma yapılmakta ve koşul doğruysa topluluk sayıları ve üyeleri ekrana yazdırılarak algoritma sonlanmaktadır. İlk sonlanma koşulu gerçekleşmediğinde kontrol değişkeni sıfırlanarak ikinci sonlanma koşuluna bakılmaktadır. Bu koşul, üye sayılarını tutan diziyi kontrol ederek eşik değeri ile karşılaştırma yapmaktadır. Koşulun doğru olduğu durumda, yine topluluk bilgileri ekrana yazdırılarak algoritma sonlanmaktadır.

3. Karşılaşılan Sorunlar

Karşılaşılan ilk sorun bir kenar üzerinden kaç defa geçildiğini tutan matrisin güncellenmesiyle ilgiliydi. Çünkü her bir düğümün başlangıç olarak seçilip diğer düğümlere ulaşması aşamasında geçilen yolların artırılması yanlış değerlerle olmaktaydı. Bunu düzeltebilmek amacıyla 'void' olarak tanımlanan fonksiyonun dönüş değeri 'int**' yapılmıştır. Ardından maksimum değerli kenarlar kaldırıldıktan sonra parçalanmış graf üzerinde topluluk tespiti yapılırken eksik değişkenler yüzünden graf üzerinde doğru bir gezinme yapılamamaktaydı. Bunu çözebilmek için newCommCount gibi güncel değerleri tutan değişkenler ve currCommunity gibi diziler eklenmiştir.

4. Karmaşıklık Analizi

Koddaki her bir fonksiyon için karmaşıklık incelenmiştir:

- printMatrix fonksiyonu, bir matrisi ekrana yazdırmak için tüm matrisi dolaştığından karmaşıklığı O(n^2)'dir.
- findMaxEdge fonksiyonu, bir matristeki maksimum değeri bulabilmek için tüm matrisi dolaştığından karmaşıklığı O(n^2)'dir.
- resetEdgeCounts fonksiyonu, bir matristeki değerleri sıfırlamak için tüm matrisi dolaştığından karmaşıklığı O(n^2)'dir.
- roadPassed fonksiyonu, bir matristeki değerleri yarıya düşürmek için tüm matrisi dolaştığından karmaşıklığı O(n^2)'dir.
- findEdgeCount fonksiyonu, iki düğüm arası en kısa yolları bulup bu yollardan geçilme sayılarını hesaplarken BFS algoritmasını uyguladığından karmaşıklığı O(V^2)'dir.
- findCommunities fonksiyonunda findEdgeCount fonksiyonu düğüm sayısı kadar çağrılmaktadır, karmaşıklığı V^3 olur. Ardından yine BFS işlemleri düğüm sayısı kadar gerçekleştirildiğinden karmaşıklık V^3 + V^3 olur. Topluluk üyelerinin yazdırılması için çağırılan iç içe üç adet for ile karmaşıklığa m^3 eklenir ancak m değeri

düğüm sayısından daha küçüktür. Tüm bu işlemler bitirme koşulu olan k tur kadar yapıldığından fonksiyonun son karmaşıklığı O(k*(V^3 + V^3 + m^3)) = O(k*V^3) olur.

Tüm bu fonksiyonlara bakıldığında en karmaşık olan fonksiyon findCommunities fonksiyonu olduğundan kodun total karmaşıklığı O(k*V^3) olmaktadır.

findCommunities
input:
graph - adjacency matrix representing the input graph
edgeCounts - matrix to store edge counts
distances - array to store distances
n - number of nodes in the graph
round - number of consecutive rounds for termination
t - minimum number of members in a community

```
function findCommunities(graph, edgeCounts, distances, n, round, t)
  newCommCount, oldCommCount = -1
  maxEdge, maxI, maxJ = 0, 0, 0
  sameRounds = 0
  queue = createArray(n)
  visited = createArray(n, false)
  currCommunity = createArray(n)
  commMembers = createArray(n)
  comms = createMatrix(n, n)
  while true do
    newCommCount = 0
    for j from 0 to n do
      edgeCounts = findEdgeCount(graph, j, n, distances, edgeCounts)
    end for
    edgeCounts = roadPassed(edgeCounts, n)
    findMaxEdge(n, edgeCounts, maxI, maxJ)
    graph[maxI][maxJ] = 0
    graph[maxJ][maxI] = 0
    edgeCounts = resetEdgeCounts(edgeCounts, n)
    for i from 0 to n do
      visited[i] = false
    end for
    for i from 0 to n do
      if not visited[i] then
        front, rear = -1, -1
        visited[i] = true
        queue[++rear] = i
        currMembers = 0
        while front != rear do
          current = queue[++front]
          currCommunity[currMembers] = current
          currMembers++
          for j from 0 to n do
            if graph[current][j] == 1 and not visited[j] then
              visited[j] = true
              queue[++rear] = j
            end if
          end for
        end while
        commMembers[newCommCount] = currMembers
        for j from 0 to currMembers do
          comms[newCommCount][j] = currCommunity[j]
        end for
```

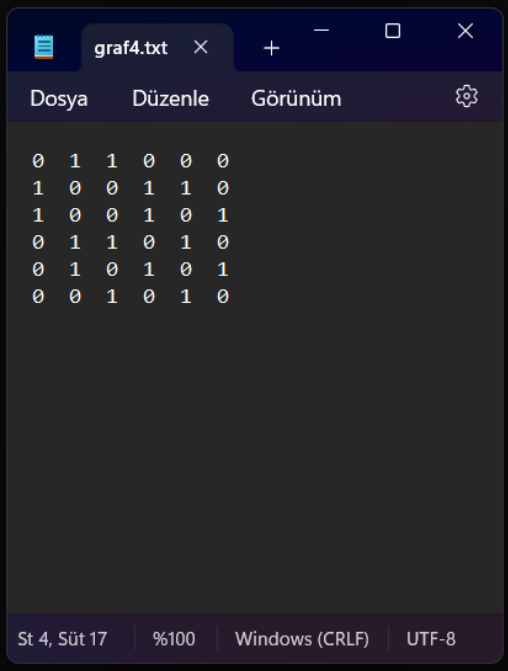
```
            newCommCount++
        end if
    end for
    if newCommCount == oldCommCount then
        sameRounds++
        if sameRounds == round then
            print("Community Details")
            return
        end if
    else
        sameRounds = 0
    end if
    for i from 0 to newCommCount do
        if commMembers[i] <= t then
            print("Community Details")
            return
        end if
    end for
    oldCommCount = newCommCount
  end while
end function

findEdgeCount
input:
  graph - adjacency matrix representing the input graph
  start - starting node for BFS
  n - number of nodes in the graph
  distances - array to store distances
  edgeCounts - matrix to store edge counts
function findEdgeCount(graph, start, n, distances, edgeCounts)
  visited = createArray(n, false)
  parent = createArray(MAX_NODES, -1)
  for i from 0 to n do
    visited[i] = false
  end for
  visited[start] = true
  parent[start] = -1
  distances[start] = 0
  queue = createArray(n)
  front, rear = -1, -1
  queue[++rear] = start
  while front != rear do
    current = queue[++front]
    for i from 0 to n do
      if graph[current][i] == 1 and not visited[i] then
        visited[i] = true
        queue[++rear] = i
        parent[i] = current
        temp = i
        while parent[temp] != -1 do
          if parent[temp] <= temp then
            edgeCounts[parent[temp]][temp]++
          else
            edgeCounts[temp][parent[temp]]++
          end if
          temp = parent[temp]
```

```
            end while
        end if
    end for
  end while
  for i from 0 to n do
    temp = i
    print("start + "->" + i ">>> Nodes passed: ")
    while temp != -1 do
      print(temp)
      temp = parent[temp]
    end while
  end for
  return edgeCounts
end function
```

5. Ekran Görüntüleri

1.örnek

```
Enter the number of nodes: 6

For how many rounds should communities be terminated when they remain the same? Enter: 4

What is the minimum number of members in communities? Enter: 3
[0->0] >>> Nodes passed: 0
[0->1] >>> Nodes passed: 1 0
[0->2] >>> Nodes passed: 2 0
[0->3] >>> Nodes passed: 3 1 0
[0->4] >>> Nodes passed: 4 1 0
[0->5] >>> Nodes passed: 5 2 0

[1->0] >>> Nodes passed: 0 1
[1->1] >>> Nodes passed: 1
[1->2] >>> Nodes passed: 2 0 1
[1->3] >>> Nodes passed: 3 1
[1->4] >>> Nodes passed: 4 1
[1->5] >>> Nodes passed: 5 4 1

[2->0] >>> Nodes passed: 0 2
[2->1] >>> Nodes passed: 1 0 2
[2->2] >>> Nodes passed: 2
[2->3] >>> Nodes passed: 3 2
[2->4] >>> Nodes passed: 4 3 2
[2->5] >>> Nodes passed: 5 2

[3->0] >>> Nodes passed: 0 1 3
[3->1] >>> Nodes passed: 1 3
[3->2] >>> Nodes passed: 2 3
[3->3] >>> Nodes passed: 3
[3->4] >>> Nodes passed: 4 3
[3->5] >>> Nodes passed: 5 2 3

[4->0] >>> Nodes passed: 0 1 4
[4->1] >>> Nodes passed: 1 4
[4->2] >>> Nodes passed: 2 3 4
[4->3] >>> Nodes passed: 3 4
[4->4] >>> Nodes passed: 4
[4->5] >>> Nodes passed: 5 4
```



```
[5->0] >>> Nodes passed: 0 2 5
[5->1] >>> Nodes passed: 1 4 5
[5->2] >>> Nodes passed: 2 5
[5->3] >>> Nodes passed: 3 2 5
[5->4] >>> Nodes passed: 4 5
[5->5] >>> Nodes passed: 5

The road [0-1] was passed 4 times.
The road [0-2] was passed 3 times.
The road [1-3] was passed 2 times.
The road [1-4] was passed 3 times.
The road [2-3] was passed 3 times.
The road [2-5] was passed 3 times.
The road [3-4] was passed 2 times.
The road [4-5] was passed 2 times.
Path [0-1] is deleted

Graph:

0       0       1       0       0       0
0       0       0       1       1       0
1       0       0       1       0       1
0       1       1       0       1       0
0       1       0       1       0       1
0       0       1       0       1       0

Edges:

0       4       3       0       0       0
0       0       0       2       3       0
0       0       0       3       0       3
0       0       0       0       2       0
0       0       0       0       0       2
0       0       0       0       0       0
[0->0] >>> Nodes passed: 0
[0->1] >>> Nodes passed: 1 3 2 0
[0->2] >>> Nodes passed: 2 0
[0->3] >>> Nodes passed: 3 2 0
[0->4] >>> Nodes passed: 4 3 2 0
[0->5] >>> Nodes passed: 5 2 0
```

```
[1->0] >>> Nodes passed: 0 2 3 1
[1->1] >>> Nodes passed: 1
[1->2] >>> Nodes passed: 2 3 1
[1->3] >>> Nodes passed: 3 1
[1->4] >>> Nodes passed: 4 1
[1->5] >>> Nodes passed: 5 4 1

[2->0] >>> Nodes passed: 0 2
[2->1] >>> Nodes passed: 1 3 2
[2->2] >>> Nodes passed: 2
[2->3] >>> Nodes passed: 3 2
[2->4] >>> Nodes passed: 4 3 2
[2->5] >>> Nodes passed: 5 2

[3->0] >>> Nodes passed: 0 2 3
[3->1] >>> Nodes passed: 1 3
[3->2] >>> Nodes passed: 2 3
[3->3] >>> Nodes passed: 3
[3->4] >>> Nodes passed: 4 3
[3->5] >>> Nodes passed: 5 2 3

[4->0] >>> Nodes passed: 0 2 3 4
[4->1] >>> Nodes passed: 1 4
[4->2] >>> Nodes passed: 2 3 4
[4->3] >>> Nodes passed: 3 4
[4->4] >>> Nodes passed: 4
[4->5] >>> Nodes passed: 5 4

[5->0] >>> Nodes passed: 0 2 5
[5->1] >>> Nodes passed: 1 4 5
[5->2] >>> Nodes passed: 2 5
[5->3] >>> Nodes passed: 3 2 5
[5->4] >>> Nodes passed: 4 5
[5->5] >>> Nodes passed: 5

The road [0-2] was passed 5 times.
The road [1-3] was passed 3 times.
The road [1-4] was passed 2 times.
The road [2-3] was passed 7 times.
The road [2-5] was passed 3 times.
```

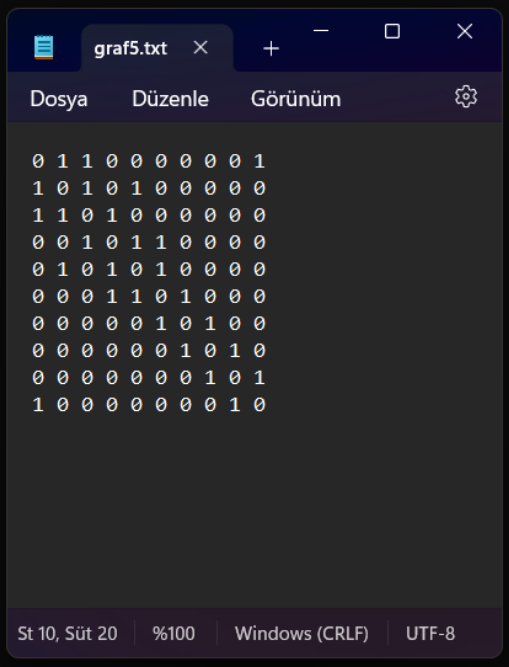
```
The road [3-4] was passed 3 times.
The road [4-5] was passed 2 times.
Path [2-3] is deleted

Graph:

0       0       1       0       0       0
0       0       0       1       1       0
1       0       0       0       0       1
0       1       0       0       1       0
0       1       0       1       0       1
0       0       1       0       1       0

Edges:

0       0       5       0       0       0
0       0       0       3       2       0
0       0       0       7       0       3
0       0       0       0       3       0
0       0       0       0       0       2
0       0       0       0       0       0
[0->0] >>> Nodes passed: 0
[0->1] >>> Nodes passed: 1 4 5 2 0
[0->2] >>> Nodes passed: 2 0
[0->3] >>> Nodes passed: 3 4 5 2 0
[0->4] >>> Nodes passed: 4 5 2 0
[0->5] >>> Nodes passed: 5 2 0

[1->0] >>> Nodes passed: 0 2 5 4 1
[1->1] >>> Nodes passed: 1
[1->2] >>> Nodes passed: 2 5 4 1
[1->3] >>> Nodes passed: 3 1
[1->4] >>> Nodes passed: 4 1
[1->5] >>> Nodes passed: 5 4 1

[2->0] >>> Nodes passed: 0 2
[2->1] >>> Nodes passed: 1 4 5 2
[2->2] >>> Nodes passed: 2
[2->3] >>> Nodes passed: 3 4 5 2
[2->4] >>> Nodes passed: 4 5 2
```

```
[2->5] >>> Nodes passed: 5 2

[3->0] >>> Nodes passed: 0 2 5 4 3
[3->1] >>> Nodes passed: 1 3
[3->2] >>> Nodes passed: 2 5 4 3
[3->3] >>> Nodes passed: 3
[3->4] >>> Nodes passed: 4 3
[3->5] >>> Nodes passed: 5 4 3

[4->0] >>> Nodes passed: 0 2 5 4
[4->1] >>> Nodes passed: 1 4
[4->2] >>> Nodes passed: 2 5 4
[4->3] >>> Nodes passed: 3 4
[4->4] >>> Nodes passed: 4
[4->5] >>> Nodes passed: 5 4

[5->0] >>> Nodes passed: 0 2 5
[5->1] >>> Nodes passed: 1 4 5
[5->2] >>> Nodes passed: 2 5
[5->3] >>> Nodes passed: 3 4 5
[5->4] >>> Nodes passed: 4 5
[5->5] >>> Nodes passed: 5

The road [0-2] was passed 5 times.
The road [1-3] was passed 1 times.
The road [1-4] was passed 4 times.
The road [2-5] was passed 8 times.
The road [3-4] was passed 4 times.
The road [4-5] was passed 9 times.
Path [4-5] is deleted

Graph:

0       0       1       0       0       0
0       0       0       1       1       0
1       0       0       0       0       1
0       1       0       0       1       0
0       1       0       1       0       0
0       0       1       0       0       0
```

```
Edges:

0       0       5       0       0       0
0       0       0       1       4       0
0       0       0       0       0       8
0       0       0       0       4       0
0       0       0       0       0       9
0       0       0       0       0       0

The algorithm is terminated because the number of members in one of the communities is at least 3.
Number of Communities: 2
Community Number -> 0
Members -> 0 2 5

Community Number -> 1
Members -> 1 3 4
```

2. örnek

```
Enter the number of nodes: 10

For how many rounds should communities be terminated when they remain the same? Enter: 12

What is the minimum number of members in communities? Enter: 2
[0->0] >>> Nodes passed: 0
[0->1] >>> Nodes passed: 1 0
[0->2] >>> Nodes passed: 2 0
[0->3] >>> Nodes passed: 3 2 0
[0->4] >>> Nodes passed: 4 1 0
[0->5] >>> Nodes passed: 5 4 1 0
[0->6] >>> Nodes passed: 6 5 4 1 0
[0->7] >>> Nodes passed: 7 8 9 0
[0->8] >>> Nodes passed: 8 9 0
[0->9] >>> Nodes passed: 9 0

[1->0] >>> Nodes passed: 0 1
[1->1] >>> Nodes passed: 1
[1->2] >>> Nodes passed: 2 1
[1->3] >>> Nodes passed: 3 2 1
[1->4] >>> Nodes passed: 4 1
[1->5] >>> Nodes passed: 5 4 1
[1->6] >>> Nodes passed: 6 5 4 1
[1->7] >>> Nodes passed: 7 8 9 0 1
[1->8] >>> Nodes passed: 8 9 0 1
[1->9] >>> Nodes passed: 9 0 1

[2->0] >>> Nodes passed: 0 2
[2->1] >>> Nodes passed: 1 2
[2->2] >>> Nodes passed: 2
[2->3] >>> Nodes passed: 3 2
[2->4] >>> Nodes passed: 4 1 2
[2->5] >>> Nodes passed: 5 3 2
[2->6] >>> Nodes passed: 6 5 3 2
[2->7] >>> Nodes passed: 7 8 9 0 2
[2->8] >>> Nodes passed: 8 9 0 2
[2->9] >>> Nodes passed: 9 0 2

[3->0] >>> Nodes passed: 0 2 3
[3->1] >>> Nodes passed: 1 2 3
```



```
[3->2] >>> Nodes passed: 2 3
[3->3] >>> Nodes passed: 3
[3->4] >>> Nodes passed: 4 3
[3->5] >>> Nodes passed: 5 3
[3->6] >>> Nodes passed: 6 5 3
[3->7] >>> Nodes passed: 7 6 5 3
[3->8] >>> Nodes passed: 8 9 0 2 3
[3->9] >>> Nodes passed: 9 0 2 3

[4->0] >>> Nodes passed: 0 1 4
[4->1] >>> Nodes passed: 1 4
[4->2] >>> Nodes passed: 2 1 4
[4->3] >>> Nodes passed: 3 4
[4->4] >>> Nodes passed: 4
[4->5] >>> Nodes passed: 5 4
[4->6] >>> Nodes passed: 6 5 4
[4->7] >>> Nodes passed: 7 6 5 4
[4->8] >>> Nodes passed: 8 9 0 1 4
[4->9] >>> Nodes passed: 9 0 1 4

[5->0] >>> Nodes passed: 0 2 3 5
[5->1] >>> Nodes passed: 1 4 5
[5->2] >>> Nodes passed: 2 3 5
[5->3] >>> Nodes passed: 3 5
[5->4] >>> Nodes passed: 4 5
[5->5] >>> Nodes passed: 5
[5->6] >>> Nodes passed: 6 5
[5->7] >>> Nodes passed: 7 6 5
[5->8] >>> Nodes passed: 8 7 6 5
[5->9] >>> Nodes passed: 9 0 2 3 5

[6->0] >>> Nodes passed: 0 2 3 5 6
[6->1] >>> Nodes passed: 1 4 5 6
[6->2] >>> Nodes passed: 2 3 5 6
[6->3] >>> Nodes passed: 3 5 6
[6->4] >>> Nodes passed: 4 5 6
[6->5] >>> Nodes passed: 5 6
[6->6] >>> Nodes passed: 6
[6->7] >>> Nodes passed: 7 6
[6->8] >>> Nodes passed: 8 7 6
```

```
[6->9] >>> Nodes passed: 9 8 7 6

[7->0] >>> Nodes passed: 0 9 8 7
[7->1] >>> Nodes passed: 1 4 5 6 7
[7->2] >>> Nodes passed: 2 3 5 6 7
[7->3] >>> Nodes passed: 3 5 6 7
[7->4] >>> Nodes passed: 4 5 6 7
[7->5] >>> Nodes passed: 5 6 7
[7->6] >>> Nodes passed: 6 7
[7->7] >>> Nodes passed: 7
[7->8] >>> Nodes passed: 8 7
[7->9] >>> Nodes passed: 9 8 7

[8->0] >>> Nodes passed: 0 9 8
[8->1] >>> Nodes passed: 1 0 9 8
[8->2] >>> Nodes passed: 2 0 9 8
[8->3] >>> Nodes passed: 3 5 6 7 8
[8->4] >>> Nodes passed: 4 5 6 7 8
[8->5] >>> Nodes passed: 5 6 7 8
[8->6] >>> Nodes passed: 6 7 8
[8->7] >>> Nodes passed: 7 8
[8->8] >>> Nodes passed: 8
[8->9] >>> Nodes passed: 9 8

[9->0] >>> Nodes passed: 0 9
[9->1] >>> Nodes passed: 1 0 9
[9->2] >>> Nodes passed: 2 0 9
[9->3] >>> Nodes passed: 3 2 0 9
[9->4] >>> Nodes passed: 4 1 0 9
[9->5] >>> Nodes passed: 5 4 1 0 9
[9->6] >>> Nodes passed: 6 7 8 9
[9->7] >>> Nodes passed: 7 8 9
[9->8] >>> Nodes passed: 8 9
[9->9] >>> Nodes passed: 9

The road [0-1] was passed 7 times.
The road [0-2] was passed 7 times.
The road [0-9] was passed 12 times.
The road [1-2] was passed 3 times.
The road [1-4] was passed 8 times.
```

```
The road [2-3] was passed 8 times.
The road [3-4] was passed 1 times.
The road [3-5] was passed 7 times.
The road [4-5] was passed 7 times.
The road [5-6] was passed 12 times.
The road [6-7] was passed 9 times.
The road [7-8] was passed 8 times.
The road [8-9] was passed 9 times.
Path [0-9] is deleted

Graph:

0       1       1       0       0       0       0       0       0       0
1       0       1       0       1       0       0       0       0       0
1       1       0       1       0       0       0       0       0       0
0       0       1       0       1       1       0       0       0       0
0       1       0       1       0       1       0       0       0       0
0       0       0       1       1       0       1       0       0       0
0       0       0       0       0       1       0       1       0       0
0       0       0       0       0       0       1       0       1       0
0       0       0       0       0       0       0       1       0       1
0       0       0       0       0       0       0       0       1       0

Edges:

0       7       7       0       0       0       0       0       0       12
0       0       3       0       8       0       0       0       0       0
0       0       0       8       0       0       0       0       0       0
0       0       0       0       1       7       0       0       0       0
0       0       0       0       0       7       0       0       0       0
0       0       0       0       0       0       12      0       0       0
0       0       0       0       0       0       0       9       0       0
0       0       0       0       0       0       0       0       8       0
0       0       0       0       0       0       0       0       0       9
0       0       0       0       0       0       0       0       0       0
[0->0] >>> Nodes passed: 0
[0->1] >>> Nodes passed: 1 0
[0->2] >>> Nodes passed: 2 0
[0->3] >>> Nodes passed: 3 2 0
[0->4] >>> Nodes passed: 4 1 0
```

```
[0->5] >>> Nodes passed: 5 4 1 0
[0->6] >>> Nodes passed: 6 5 4 1 0
[0->7] >>> Nodes passed: 7 6 5 4 1 0
[0->8] >>> Nodes passed: 8 7 6 5 4 1 0
[0->9] >>> Nodes passed: 9 8 7 6 5 4 1 0

[1->0] >>> Nodes passed: 0 1
[1->1] >>> Nodes passed: 1
[1->2] >>> Nodes passed: 2 1
[1->3] >>> Nodes passed: 3 2 1
[1->4] >>> Nodes passed: 4 1
[1->5] >>> Nodes passed: 5 4 1
[1->6] >>> Nodes passed: 6 5 4 1
[1->7] >>> Nodes passed: 7 6 5 4 1
[1->8] >>> Nodes passed: 8 7 6 5 4 1
[1->9] >>> Nodes passed: 9 8 7 6 5 4 1

[2->0] >>> Nodes passed: 0 2
[2->1] >>> Nodes passed: 1 2
[2->2] >>> Nodes passed: 2
[2->3] >>> Nodes passed: 3 2
[2->4] >>> Nodes passed: 4 1 2
[2->5] >>> Nodes passed: 5 3 2
[2->6] >>> Nodes passed: 6 5 3 2
[2->7] >>> Nodes passed: 7 6 5 3 2
[2->8] >>> Nodes passed: 8 7 6 5 3 2
[2->9] >>> Nodes passed: 9 8 7 6 5 3 2

[3->0] >>> Nodes passed: 0 2 3
[3->1] >>> Nodes passed: 1 2 3
[3->2] >>> Nodes passed: 2 3
[3->3] >>> Nodes passed: 3
[3->4] >>> Nodes passed: 4 3
[3->5] >>> Nodes passed: 5 3
[3->6] >>> Nodes passed: 6 5 3
[3->7] >>> Nodes passed: 7 6 5 3
[3->8] >>> Nodes passed: 8 7 6 5 3
[3->9] >>> Nodes passed: 9 8 7 6 5 3

[4->0] >>> Nodes passed: 0 1 4
```

```
[4->1] >>> Nodes passed: 1 4
[4->2] >>> Nodes passed: 2 1 4
[4->3] >>> Nodes passed: 3 4
[4->4] >>> Nodes passed: 4
[4->5] >>> Nodes passed: 5 4
[4->6] >>> Nodes passed: 6 5 4
[4->7] >>> Nodes passed: 7 6 5 4
[4->8] >>> Nodes passed: 8 7 6 5 4
[4->9] >>> Nodes passed: 9 8 7 6 5 4

[5->0] >>> Nodes passed: 0 2 3 5
[5->1] >>> Nodes passed: 1 4 5
[5->2] >>> Nodes passed: 2 3 5
[5->3] >>> Nodes passed: 3 5
[5->4] >>> Nodes passed: 4 5
[5->5] >>> Nodes passed: 5
[5->6] >>> Nodes passed: 6 5
[5->7] >>> Nodes passed: 7 6 5
[5->8] >>> Nodes passed: 8 7 6 5
[5->9] >>> Nodes passed: 9 8 7 6 5

[6->0] >>> Nodes passed: 0 2 3 5 6
[6->1] >>> Nodes passed: 1 4 5 6
[6->2] >>> Nodes passed: 2 3 5 6
[6->3] >>> Nodes passed: 3 5 6
[6->4] >>> Nodes passed: 4 5 6
[6->5] >>> Nodes passed: 5 6
[6->6] >>> Nodes passed: 6
[6->7] >>> Nodes passed: 7 6
[6->8] >>> Nodes passed: 8 7 6
[6->9] >>> Nodes passed: 9 8 7 6

[7->0] >>> Nodes passed: 0 2 3 5 6 7
[7->1] >>> Nodes passed: 1 4 5 6 7
[7->2] >>> Nodes passed: 2 3 5 6 7
[7->3] >>> Nodes passed: 3 5 6 7
[7->4] >>> Nodes passed: 4 5 6 7
[7->5] >>> Nodes passed: 5 6 7
[7->6] >>> Nodes passed: 6 7
[7->7] >>> Nodes passed: 7
```

```
[7->8] >>> Nodes passed: 8 7
[7->9] >>> Nodes passed: 9 8 7

[8->0] >>> Nodes passed: 0 2 3 5 6 7 8
[8->1] >>> Nodes passed: 1 4 5 6 7 8
[8->2] >>> Nodes passed: 2 3 5 6 7 8
[8->3] >>> Nodes passed: 3 5 6 7 8
[8->4] >>> Nodes passed: 4 5 6 7 8
[8->5] >>> Nodes passed: 5 6 7 8
[8->6] >>> Nodes passed: 6 7 8
[8->7] >>> Nodes passed: 7 8
[8->8] >>> Nodes passed: 8
[8->9] >>> Nodes passed: 9 8

[9->0] >>> Nodes passed: 0 2 3 5 6 7 8 9
[9->1] >>> Nodes passed: 1 4 5 6 7 8 9
[9->2] >>> Nodes passed: 2 3 5 6 7 8 9
[9->3] >>> Nodes passed: 3 5 6 7 8 9
[9->4] >>> Nodes passed: 4 5 6 7 8 9
[9->5] >>> Nodes passed: 5 6 7 8 9
[9->6] >>> Nodes passed: 6 7 8 9
[9->7] >>> Nodes passed: 7 8 9
[9->8] >>> Nodes passed: 8 9
[9->9] >>> Nodes passed: 9

The road [0-1] was passed 4 times.
The road [0-2] was passed 4 times.
The road [1-2] was passed 3 times.
The road [1-4] was passed 10 times.
The road [2-3] was passed 10 times.
The road [3-4] was passed 1 times.
The road [3-5] was passed 12 times.
The road [4-5] was passed 12 times.
The road [5-6] was passed 24 times.
The road [6-7] was passed 21 times.
The road [7-8] was passed 16 times.
The road [8-9] was passed 9 times.
Path [5-6] is deleted
```

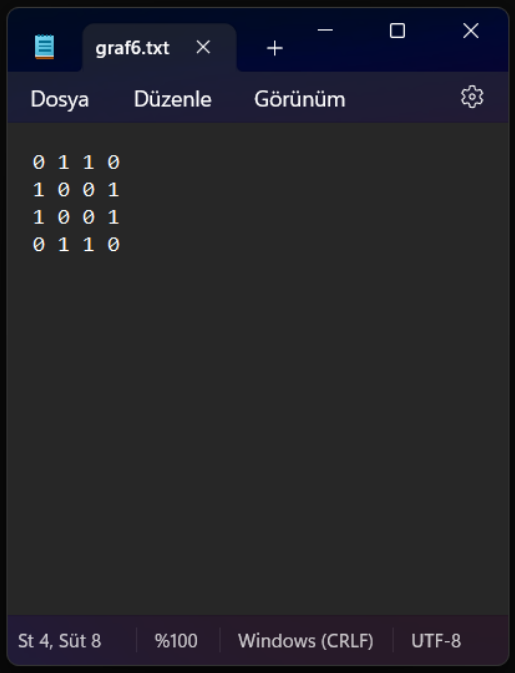
```
Graph:

0       1       1       0       0       0       0       0       0       0
1       0       1       0       1       0       0       0       0       0
1       1       0       1       0       0       0       0       0       0
0       0       1       0       1       1       0       0       0       0
0       1       0       1       0       1       0       0       0       0
0       0       0       1       1       0       0       0       0       0
0       0       0       0       0       0       0       1       0       0
0       0       0       0       0       0       1       0       1       0
0       0       0       0       0       0       0       1       0       1
0       0       0       0       0       0       0       0       1       0

Edges:

0       4       4       0       0       0       0       0       0       0
0       0       3       0       10      0       0       0       0       0
0       0       0       10      0       0       0       0       0       0
0       0       0       0       1       12      0       0       0       0
0       0       0       0       0       12      0       0       0       0
0       0       0       0       0       0       24      0       0       0
0       0       0       0       0       0       0       21      0       0
0       0       0       0       0       0       0       0       16      0
0       0       0       0       0       0       0       0       0       9
0       0       0       0       0       0       0       0       0       0
[0->0] >>> Nodes passed: 0
[0->1] >>> Nodes passed: 1 0
[0->2] >>> Nodes passed: 2 0
[0->3] >>> Nodes passed: 3 2 0
[0->4] >>> Nodes passed: 4 1 0
[0->5] >>> Nodes passed: 5 4 1 0
[0->6] >>> Nodes passed: 6
[0->7] >>> Nodes passed: 7
[0->8] >>> Nodes passed: 8
[0->9] >>> Nodes passed: 9

[1->0] >>> Nodes passed: 0 1
[1->1] >>> Nodes passed: 1
[1->2] >>> Nodes passed: 2 1
[1->3] >>> Nodes passed: 3 2 1
```

Uzun bir örnek olduğundan 3, 4 ve 5. iterasyonlar kırpılmıştır. Oluşan topluluklar:

```
Graph:

0       1       1       0       0       0       0       0       0       0
1       0       1       0       0       0       0       0       0       0
1       1       0       0       0       0       0       0       0       0
0       0       0       0       1       1       0       0       0       0
0       0       0       1       0       1       0       0       0       0
0       0       0       1       1       0       0       0       0       0
0       0       0       0       0       0       0       1       0       0
0       0       0       0       0       0       1       0       0       0
0       0       0       0       0       0       0       0       0       1
0       0       0       0       0       0       0       0       1       0

Edges:

0       1       1       0       0       0       0       0       0       0
0       0       1       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0
0       0       0       0       1       1       0       0       0       0
0       0       0       0       0       1       0       0       0       0
0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       3       0       0
0       0       0       0       0       0       0       0       4       0
0       0       0       0       0       0       0       0       0       3
0       0       0       0       0       0       0       0       0       0

The algorithm is terminated because the number of members in one of the communities is at least 2.
Number of Communities: 4
Community Number -> 0
Members -> 0 1 2

Community Number -> 1
Members -> 3 4 5

Community Number -> 2
Members -> 6 7

Community Number -> 3
Members -> 8 9
```

3. örnek

```
Enter the number of nodes: 4

For how many rounds should communities be terminated when they remain the same? Enter: 1

What is the minimum number of members in communities? Enter: 1
[0->0] >>> Nodes passed: 0
[0->1] >>> Nodes passed: 1 0
[0->2] >>> Nodes passed: 2 0
[0->3] >>> Nodes passed: 3 1 0

[1->0] >>> Nodes passed: 0 1
[1->1] >>> Nodes passed: 1
[1->2] >>> Nodes passed: 2 0 1
[1->3] >>> Nodes passed: 3 1

[2->0] >>> Nodes passed: 0 2
[2->1] >>> Nodes passed: 1 0 2
[2->2] >>> Nodes passed: 2
[2->3] >>> Nodes passed: 3 2

[3->0] >>> Nodes passed: 0 1 3
[3->1] >>> Nodes passed: 1 3
[3->2] >>> Nodes passed: 2 3
[3->3] >>> Nodes passed: 3

The road [0-1] was passed 3 times.
The road [0-2] was passed 2 times.
The road [1-3] was passed 2 times.
The road [2-3] was passed 1 times.
Path [0-1] is deleted

Graph:

0       0       1       0
0       0       0       1
1       0       0       1
0       1       1       0
```



```
Edges:

0       0       3       0
0       0       0       3
0       0       0       4
0       0       0       0
[0->0] >>> Nodes passed: 0
[0->1] >>> Nodes passed: 1
[0->2] >>> Nodes passed: 2 0
[0->3] >>> Nodes passed: 3

[1->0] >>> Nodes passed: 0
[1->1] >>> Nodes passed: 1
[1->2] >>> Nodes passed: 2
[1->3] >>> Nodes passed: 3 1

[2->0] >>> Nodes passed: 0 2
[2->1] >>> Nodes passed: 1
[2->2] >>> Nodes passed: 2
[2->3] >>> Nodes passed: 3

[3->0] >>> Nodes passed: 0
[3->1] >>> Nodes passed: 1 3
[3->2] >>> Nodes passed: 2
[3->3] >>> Nodes passed: 3

The road [0-2] was passed 1 times.
The road [1-3] was passed 1 times.
Path [0-2] is deleted

Graph:

0       0       0       0
0       0       0       1
0       0       0       0
0       1       0       0
```

```
Edges:

0       3       2       0
0       0       0       2
0       0       0       1
0       0       0       0
[0->0] >>> Nodes passed: 0
[0->1] >>> Nodes passed: 1 3 2 0
[0->2] >>> Nodes passed: 2 0
[0->3] >>> Nodes passed: 3 2 0

[1->0] >>> Nodes passed: 0 2 3 1
[1->1] >>> Nodes passed: 1
[1->2] >>> Nodes passed: 2 3 1
[1->3] >>> Nodes passed: 3 1

[2->0] >>> Nodes passed: 0 2
[2->1] >>> Nodes passed: 1 3 2
[2->2] >>> Nodes passed: 2
[2->3] >>> Nodes passed: 3 2

[3->0] >>> Nodes passed: 0 2 3
[3->1] >>> Nodes passed: 1 3
[3->2] >>> Nodes passed: 2 3
[3->3] >>> Nodes passed: 3

The road [0-2] was passed 3 times.
The road [1-3] was passed 3 times.
The road [2-3] was passed 4 times.
Path [2-3] is deleted

Graph:

0       0       1       0
0       0       0       1
1       0       0       0
0       1       0       0
```

```
Edges:

0       0       1       0
0       0       0       1
0       0       0       0
0       0       0       0

The algorithm is terminated because the number of members in one of the communities is at least 1.
Number of Communities: 3
Community Number -> 0
Members -> 0

Community Number -> 1
Members -> 1 3

Community Number -> 2
Members -> 2
```