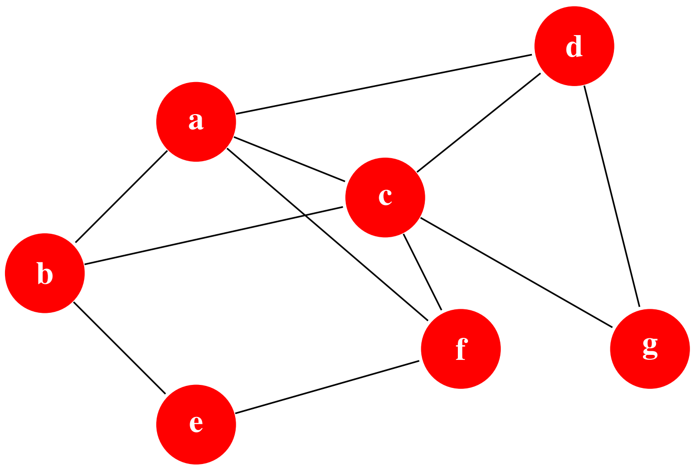
University of Augsburg, Institute of Computer Science | SS 2019
Prof. Dr. P. Fischer | 16. July 2019
J. Kastner, L. Rudenko | Solution 11

Analyzing Massive Data Sets

Exercise 1: Counting Triangles (homework)



a) Compute the number of triangles:

- **Nodes**: $V = \{a, b, c, d, e, f, g\}$
- **Degrees**: $deg(a) = 4, deg(b) = 3, deg(c) = 5, deg(d) = 3, deg(e) = 2,$ $deg(f) = 3, deg(g) = 2$
- **Adjacency**(v): $a(b, c, d, f)$, $b(a, c, e)$, $c(a, b, d, f, g)$, $d(a, c, g)$, $e(b, f)$, $f(a, c, e)$, $g(c, d)$
- **#Triangles**: 0

1. Node a: all neighbour-pairs in adjacency(a): bc, bd, bf, cd, cf, df
   - $deg(a) < deg(b)$ && $deg(a) < deg(c)$: **false**
     $deg(a) = 4, deg(b) = 3, deg(c) = 5$
   - $deg(a) < deg(b)$ && $deg(a) < deg(d)$: **false**
     $deg(a) = 4, deg(b) = 3, deg(d) = 3$
   - $deg(a) < deg(b)$ && $deg(a) < deg(f)$: **false**
     $deg(a) = 4, deg(b) = 3, deg(f) = 3$
   - $deg(a) < deg(c)$ && $deg(a) < deg(d)$: **false**
     $deg(a) = 4, deg(c) = 5, deg(d) = 3$
   - $deg(a) < deg(c)$ && $deg(a) < deg(f)$: **false**
     $deg(a) = 4, deg(c) = 5, deg(d) = 3$
   - $deg(a) < deg(d)$ && $deg(a) < deg(f)$: **false**
     $deg(a) = 4, deg(d) = 3, deg(f) = 3$

   #Triangles: 0

2. Node b: all neighbour-pairs in adjacency(b): ac, ae, ce
   - $deg(b) < deg(a)$ && $deg(b) < deg(c)$: **true**
     $deg(b) = 3, deg(a) = 4, deg(c) = 5$
     $ac \in E \Rightarrow$ #Triangles++
   - $deg(b) < deg(a)$ && $deg(b) < deg(e)$: **false**
     $deg(b) = 3, deg(a) = 4, deg(e) = 2$
   - $deg(b) < deg(c)$ && $deg(b) < deg(e)$: **false**
     $deg(b) = 3, deg(c) = 5, deg(e) = 2$

   #Triangles: 1
3. Node c: all neighbour-pairs in adjacency(c): $ab, ad, af, ag, bd, bf, bg, df, dg, fg$
   - $deg(c) < deg(a)$ && $deg(c) < deg(b)$: **false**
     $deg(c) = 5, deg(a) = 4, deg(b) = 3$
   - $deg(c) < deg(a)$ && $deg(c) < deg(d)$: **false**
     $deg(c) = 5, deg(a) = 4, deg(d) = 3$
   - $deg(c) < deg(a)$ && $deg(c) < degfb)$: **false**
     $deg(c) = 5, deg(a) = 4, deg(f) = 3$
   - $deg(c) < deg(a)$ && $deg(c) < deg(g)$: **false**
     $deg(c) = 5, deg(a) = 4, deg(b) = 2$
   - $deg(c) < deg(b)$ && $deg(c) < deg(d)$: **false**
     $deg(c) = 5, deg(b) = 3, deg(d) = 3$
   - $deg(c) < deg(b)$ && $deg(c) < deg(f)$: **false**
     $deg(c) = 5, deg(b) = 3, deg(f) = 3$
   - $deg(c) < deg(b)$ && $deg(c) < deg(g)$: **false**
     $deg(c) = 5, deg(b) = 3, deg(g) = 2$
   - $deg(c) < deg(d)$ && $deg(c) < deg(f)$: **false**
     $deg(c) = 5, deg(d) = 3, deg(f) = 3$
   - $deg(c) < deg(d)$ && $deg(c) < deg(g)$: **false**
     $deg(c) = 5, deg(d) = 3, deg(g) = 2$
   - $deg(c) < deg(f)$ && $deg(c) < deg(g)$: **false**
     $deg(c) = 5, deg(f) = 3, deg(g) = 2$

   #Triangles: 1
4. Node d: all neighbour-pairs in adjacency(d): ac, ag, cg
   - $deg(d) < deg(a)$ && $deg(d) < deg(c)$: **true**
     $deg(d) = 3, deg(a) = 4, deg(c) = 5$
     $ac \in E \Rightarrow$ #Triangles++
   - $deg(d) < deg(a)$ && $deg(d) < deg(g)$: **false**
     $deg(d) = 3, deg(a) = 4, deg(g) = 2$
   - $deg(d) < deg(c)$ && $deg(d) < deg(g)$: **false**
     $deg(d) = 3, deg(c) = 5, deg(g) = 2$

   #Triangles: 2
5. Node e: all neighbour-pairs in adjacency(e): bf
   - $deg(e) < deg(b)$ && $deg(e) < deg(f)$: **true**
     $deg(e) = 2, deg(b) = 3, deg(f) = 3$
     $bf \notin E$

   #Triangles: 2

6. Node f: all neighbour-pairs in adjacency(f): ac, ae, ce
   - $deg(f) < deg(a)$ && $deg(f) < deg(c)$: **true**
     $deg(f) = 3, deg(a) = 4, deg(c) = 5$
     $ac \in E \Rightarrow$ #Triangles++
   - $deg(f) < deg(a)$ && $deg(f) < deg(e)$: **false**
     $deg(f) = 3, deg(a) = 4, deg(e) = 2$
   - $deg(f) < deg(c)$ && $deg(f) < deg(e)$: **false**
     $deg(f) = 3, deg(c) = 5, deg(e) = 2$

   #Triangles: 3
7. Node g: all neighbour-pairs in adjacency(g): cd
   - $deg(g) < deg(c)$ && $deg(g) < deg(d)$: **true**
     $deg(g) = 2, deg(b) = 3, deg(f) = 3$
     $cd \in E \Rightarrow$ #Triangles++

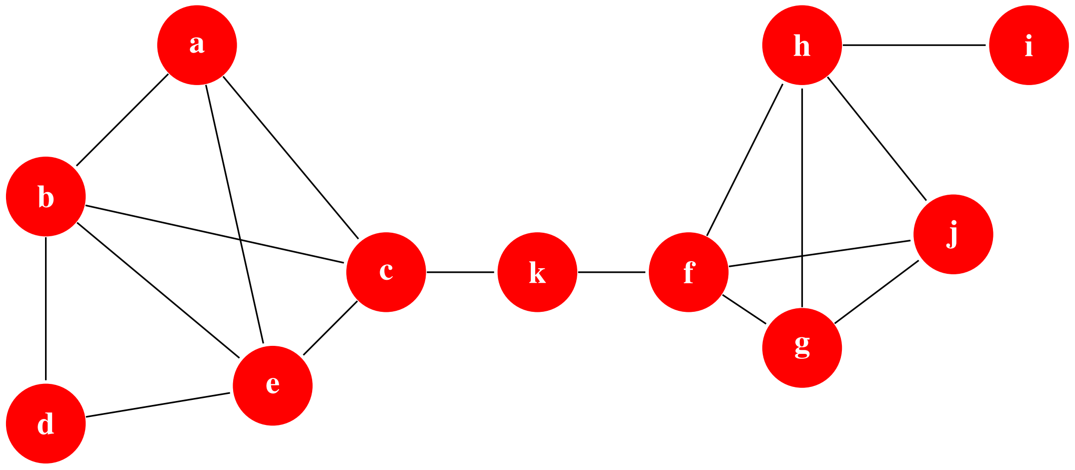
   #Triangles: 4

Graph G has 4 triangles: abc, acd, acf, cdg

b) Python code:

```
import itertools

# tests, whether vertex x is less than vertex y (x < y) in graph;
# < defines strict total order on vertices
def vertex_lt(x, y, graph):
  deg_x = len(graph[x]) # because all edges are bidirectional
  deg_y = len(graph[y])
  if deg_x < deg_y or deg_x == deg_y and x < y:
    return True
  return False

def count_triangles(graph):
  num_triangles = 0
  for v, adjacency_list in graph.items(): # v is vertex id
    #generator expression instead of list comprehension
    neighbors_of_greater_deg = (n for n in adjacency_list if vertex_lt(v, n, graph))
    for u, w in itertools.combinations(neighbors_of_greater_deg, 2):
      if w in graph[u]: # because all edges are bidirectional
        print('Found␣triangle:␣' + v + u + w)
        num_triangles += 1
  return num_triangles

graph = {'a': ['b','c','d','f'],
         'b': ['a','c','e'],
         'c': ['a','b','d','f','g'],
         'd': ['a','c','g'],
         'e': ['b','f'],
         'f': ['a','c','e'],
         'g': ['c','d']}

print(count_triangles(graph))
```

Exercise 2: k-core (homework)



- First we compute degrees for each node: $deg(a) = 3, deg(b) = 4, deg(c) = 4, deg(d) = 2, deg(e) = 4, deg(k) = 2, deg(f) = 4, deg(g) = 3, deg(h) = 4, deg(i) = 1, deg(j) = 3$.
- Then we sort nodes into bins:

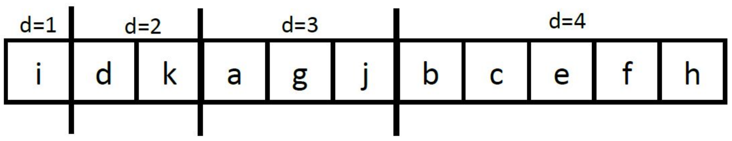


Abbildung 1: Sorting nodes into bins.

- Neighbours of each node will be visit:
  - node i visits its neighbour node h. Node h has higher degree (4), so its degree will be decreased to 3:

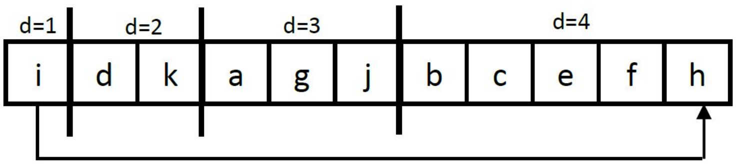


Abbildung 2: i visits its neighbour h.

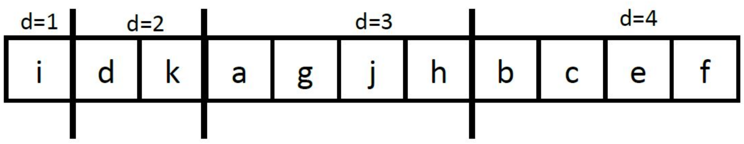


Abbildung 3: Nodes after decreasing of h-degree.

- node d visits its neighbour nodes b and e. Both nodes have higher degree (4), so their degrees will be decreased to 3:

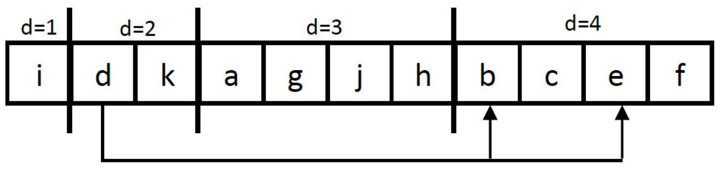


Abbildung 4: d visits its neighbours b and e.

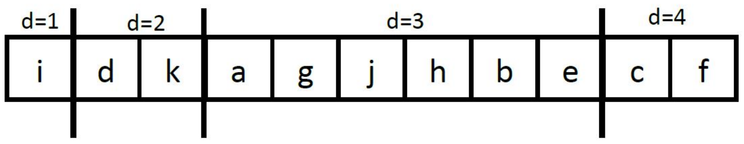


Abbildung 5: Nodes after decreasing of b- and e-degree.

- node k visits its neighbour nodes f and c. Both nodes have higher degree (4), so their degrees will be decreased to 3:

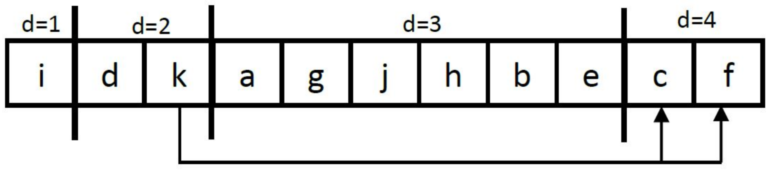


Abbildung 6: k visits its neighbours f and c.

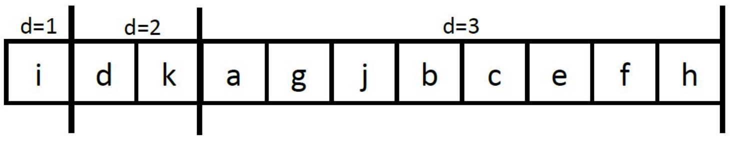


Abbildung 7: Nodes after decreasing of c- and f-degree.

- The next node, which has to visit its neighbours, is a with degree 3. But we don't have any nodes with higher degree. So we are ready now and can look for the 3-cores.

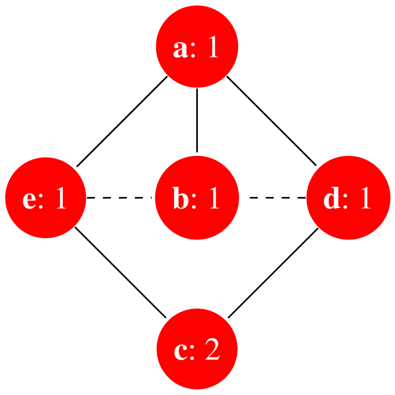
We want to have 3-core and all belonging components. All nodes from 3-degree bin and higher belong to the 3-core. The belonging components have to be connected, so we take the node a from the 3-degree bin and look for all neighbours of a in the current bin or in the bins with higher degrees. These are the nodes b, c and e. Next we will check the neighbours of these new nodes. If they have the neighbours that do not yet belong to the k-core, they are included. Then we check the neighbours of the neighbours and so on.

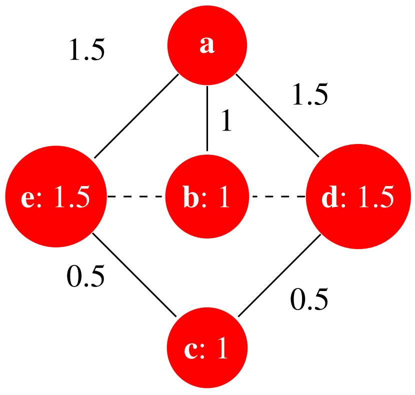
This way we get 3-core $abcefghj$ and two components $abce$ and $fghj$.

Exercise 3: Girvan-Newman Algorithm (homework)

a) In order to compute the betweenness we first have to start a breadth first search (BFS) for each node. We determine to each node the number of shortest paths starting at node a (always on the left-hand side). On the right-hand side we compute the values for each path. In the nodes on the right-hand side are intermediary results which are used for computation of the next edge weights.

For node a:

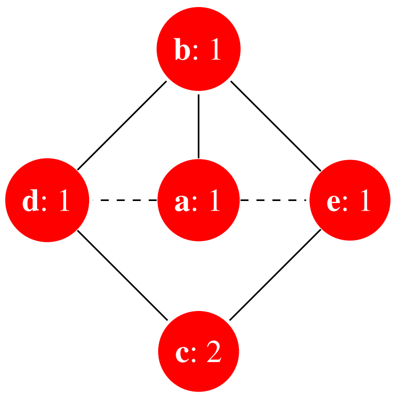


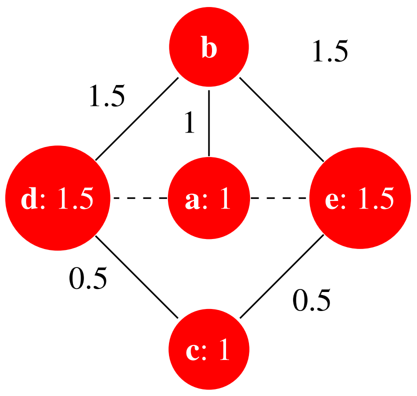


a has only one shortest path starting at a. b, e and d also have only one shortest path starting at node a. But if we consider c, it can be reached from e and d, so we add the numbers of shortest paths from e and d to get the number of shortest paths for node c, which is 2.

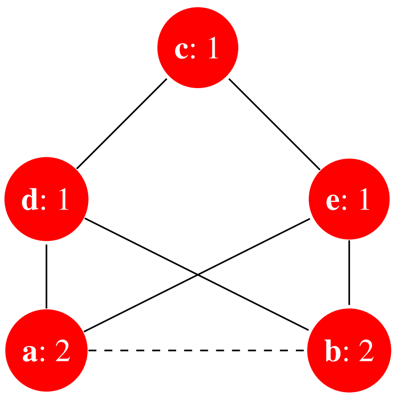
Now we give each leaf a value of 1. So c and b receive a value of 1. Each leaf and later each node gives their values to the paths. If there is only one path the whole value is received, while there are more than 2 paths we have to consider the nodes which are reached with the paths. The values are split based on the parent values considering the number of shortest paths (left-hand side). c can be reached from the nodes e and d. So the path $e - c$ receives one half and the path $d - c$ the other half. If we go on, node d receives all values from incoming paths plus 1, which is 1.5 in this case.

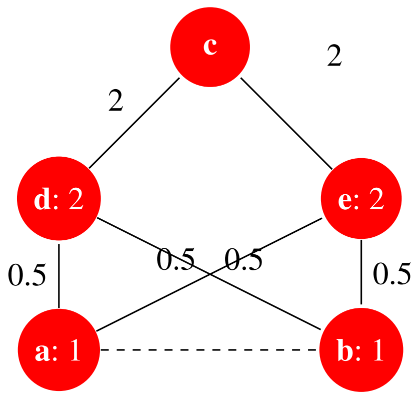
For node b:



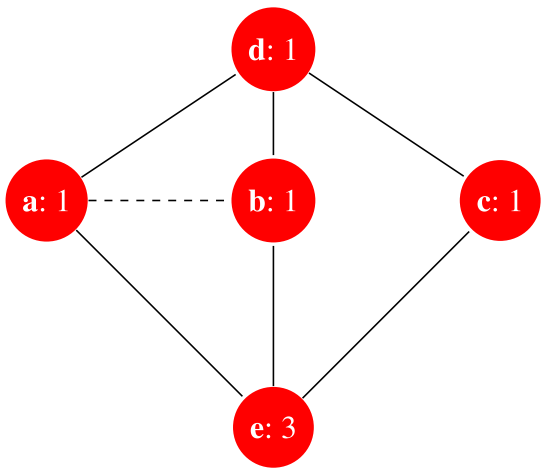


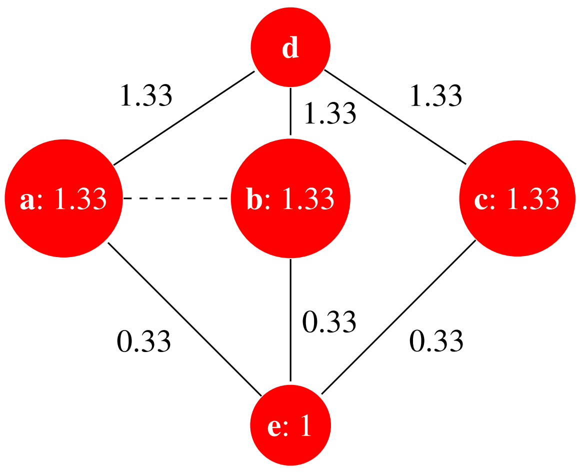
For node c:



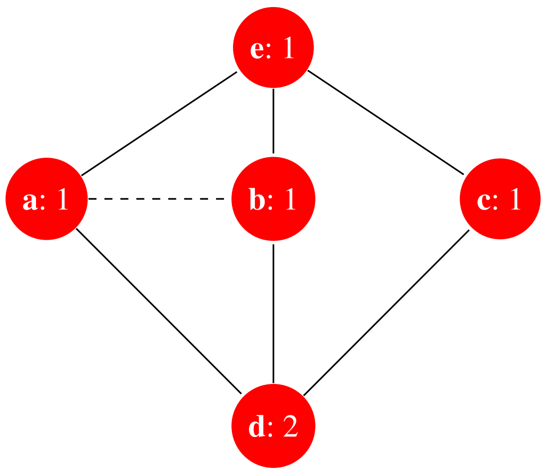


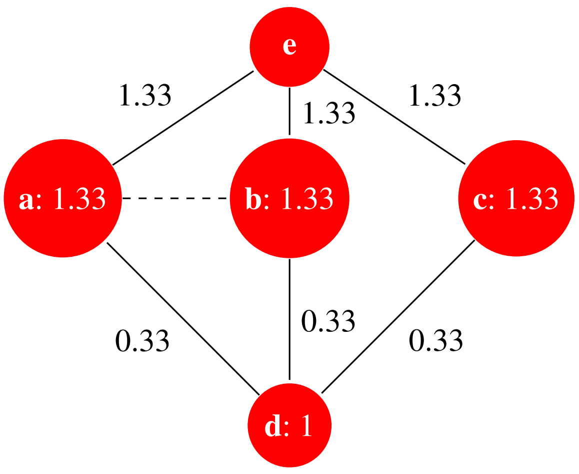
For node d:





For node e:



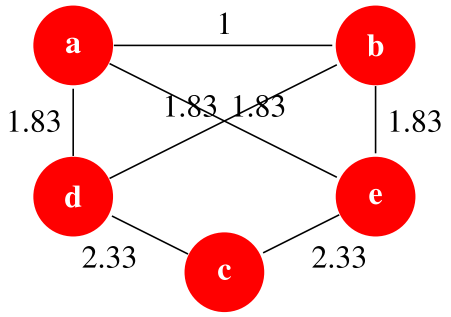


Now we have to sum up the values of the betweenness for all paths over the graphs for each node and divide the summed up value through 2.

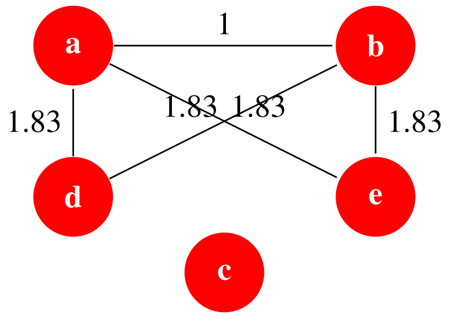
- $a - b$: $\frac{1+1+0+0+0}{2} = 1$
- $a - e$: $\frac{1.5+0+0.5+0.33+1.33}{2} = 1.83$
- $a - d$: $\frac{1.5+0+0.5+1.33+0.33}{2} = 1.83$
- $b - d$: $\frac{0+1.5+0.5+1.33+0.33}{2} = 1.83$
- $b - e$: $\frac{0+1.5+0.5+0.33+1.33}{2} = 1.83$
- $c - d$: $\frac{0.5+0.5+2+1.33+0.33}{2} = 2.33$
- $c - e$: $\frac{0.5+0.5+2+0.33+1.33}{2} = 2.33$

Now we can proceed with the **Girvan-Newman algorithm**. First we remove all paths with a betweenness-value of 2.33. After that we remove all paths with a betweenness-value of 1.83. The next step is to remove all paths with a betweenness-value of 1.

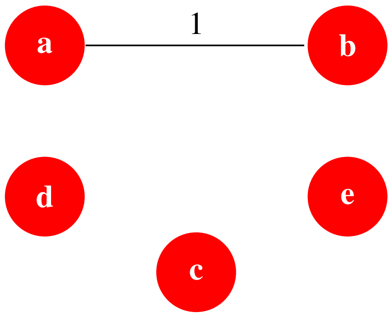
0) Graph with all paths.



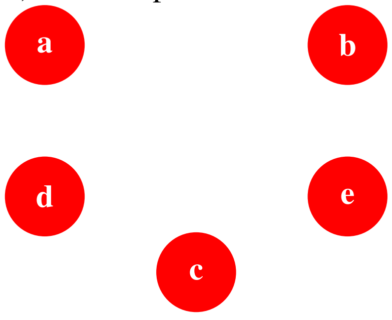
1) Remove paths with a betweenness of 2.33.



2) Remove paths with a betweenness of 1.83



3) Remove paths with a betweenness of 1

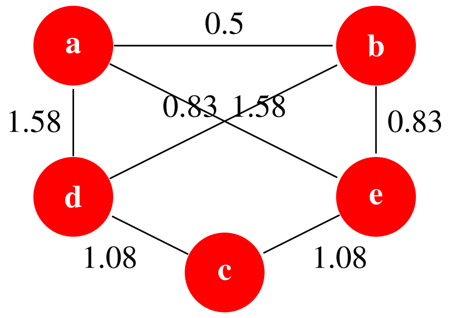


b) If we now consider only the betweenness for node a, d and e we receive:

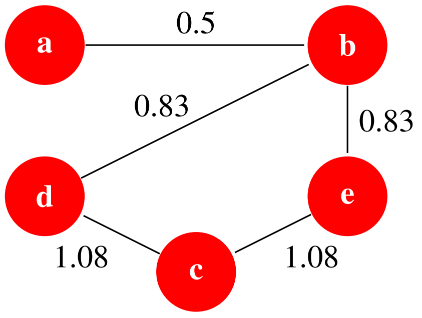
- $a-b$: $\frac{1+0+0}{2}=0.5$
- $a-e$: $\frac{1.5+0.33+1.33}{2}=1.58$
- $a-d$: $\frac{1.5+1.33+0.33}{2}=1.58$
- $b-d$: $\frac{0+1.33+0.33}{2}=0.83$
- $b-e$: $\frac{0+0.33+1.33}{2}=0.83$
- $c-d$: $\frac{0.5+1.33+0.33}{2}=1.08$
- $c-e$: $\frac{0.5+0.33+1.33}{2}=1.08$

Now we can proceed with the Girvan-Newman Algorithm. First we remove all paths with a betweenness-value of 1.58. After that we remove all paths with a betweenness-value of 1.08. The next step is to remove all paths with a betweenness-value 0.83 and last we remove the edge with the weight of 0.5.

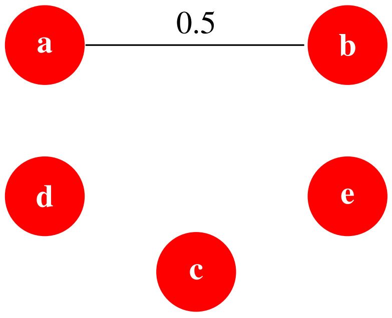
0) Graph with all paths.



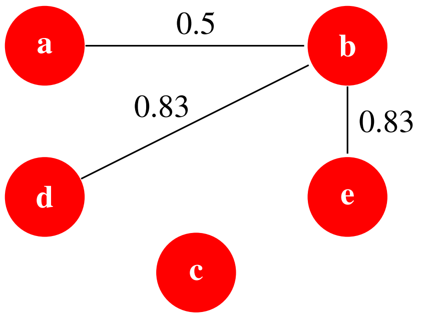
1) Remove paths with a betweenness of 1.58.



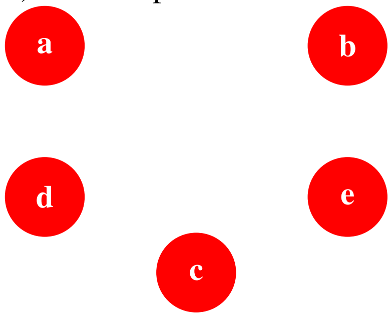
3) Remove paths with a betweenness of 0.83



2) Remove paths with a betweenness of 1.08

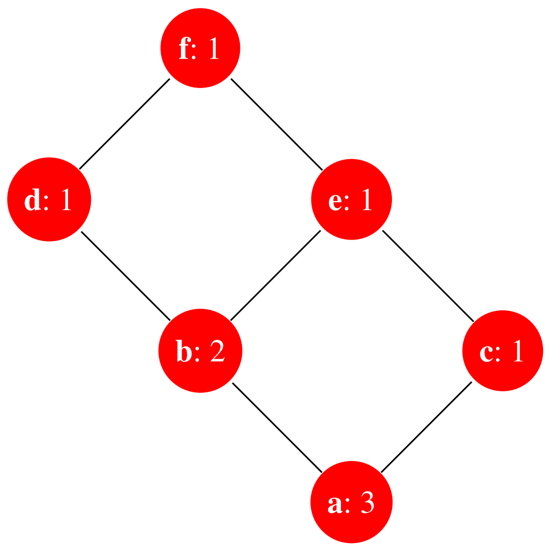


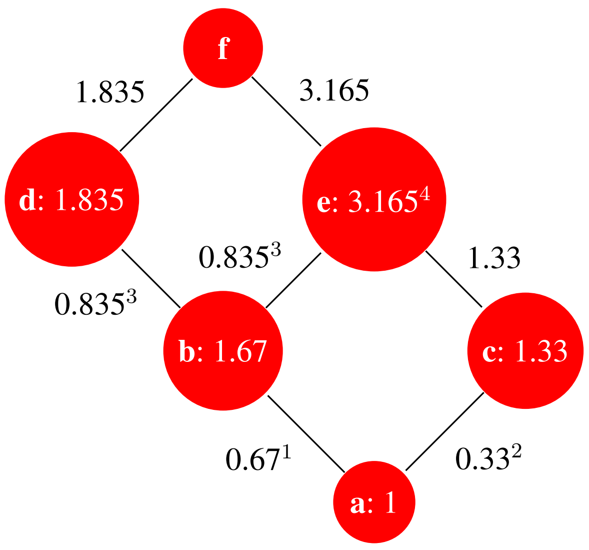
4) Remove paths with a betweenness of 0.5



Compared to subtask a) Girvan Newman needs one more step for removing the edges with the highest betweennesses. But all in all the order of the clusters the nodes are in is the same comapred to subtask a).

An additional example for edge weights calculation:





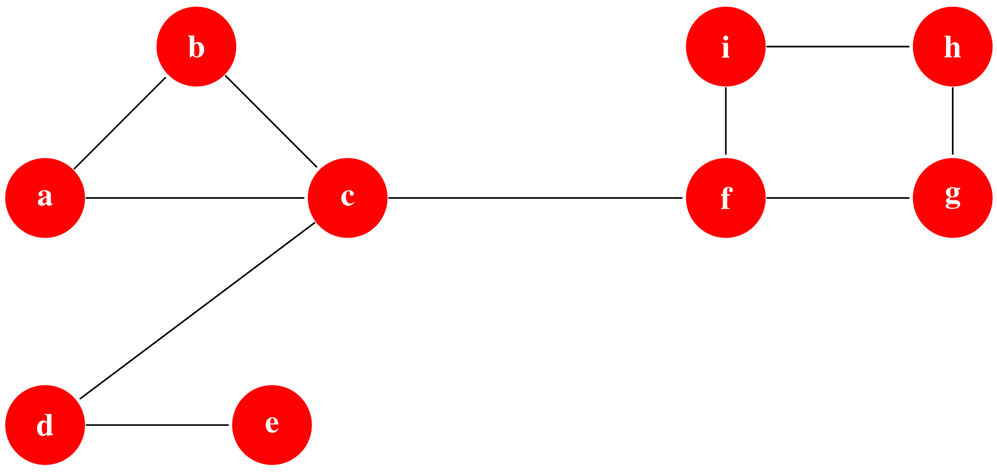
$1 : 1 * 2/3 =$ **0.67**, because the are two shortest paths through b
$2 : 1 * 1/3 =$ **0.33**, because the are one shortest path through c
$3 : 1.67 * 0.5 =$ **0.835**
$4 : 0.835 + 1.33 + 1 =$ **3.165**

Important!

1 will be added to the result of previous edge **before** the weight for the next edge can be determined: $(0.67 + 1) * 0.5 = 0.835$ (**not** $0.67 * 0.5 + 1$ as it was discussed in some exercises!)

Exercise 4: Modularity (homework)

Given is the following graph G:



$Q(G, S) = \frac{1}{2m} \sum_{s \in S} \sum_{i \in s} \sum_{j \in s} (A_{ij} - \frac{k_i k_j}{2m})$, $m = 10$

$\Delta_{ij} = A_{ij} - \frac{k_i k_j}{2m}$

- $s_1 : \{a, b, c\}$, $s_2 : \{d, e\}$, $s_3 : \{i, h, f, g\}$
  s_1 :
  - $\Delta_{ab} = 1 - \frac{2*2}{2*10} = \frac{4}{5} = 0.8$
  - $\Delta_{bc} = 1 - \frac{2*4}{2*10} = \frac{6}{10} = 0.6$
  - $\Delta_{ac} = 1 - \frac{2*4}{2*10} = \frac{6}{10} = 0.6$

  Each edge should be considered **twice**: ab and ba, bc and cb and so on. The **loops** (link from a node to itself) must also be taken into account:
  - $\Delta_{aa} = 0 - \frac{2*2}{2*10} = -0.2$
  - $\Delta_{bb} = 0 - \frac{2*2}{2*10} = -0.2$
  - $\Delta_{cc} = 0 - \frac{4*4}{2*10} = -0.8$

  $\Rightarrow \sum = 2 * (0.8 + 0.6 + 0.6) - (0.2 + 0.2 + 0.8) = 4 - 1.2 = 2.8$
  s_2 :
  - $\Delta_{de} = 1 - \frac{2*1}{2*10} = \frac{9}{10} = 0.9$
  - $\Delta_{dd} = 0 - \frac{2*2}{2*10} = -0.2$
  - $\Delta_{ee} = 0 - \frac{1*1}{2*10} = -0.05$

  $\Rightarrow \sum = 2 * 0.9 - (0.2 + 0.05) = 1.8 - 0.25 = 1.55$
  s_3 :
  - $\Delta_{ih} = 1 - \frac{2*2}{2*10} = \frac{4}{5} = 0.8$
  - $\Delta_{if} = 1 - \frac{2*3}{2*10} = \frac{7}{10} = 0.7$
  - $\Delta_{ig} = 0 - \frac{2*2}{2*10} = -\frac{1}{5} = -0.2$
  - $\Delta_{hf} = 0 - \frac{2*3}{2*10} = -\frac{3}{10} = -0.3$
  - $\Delta_{hg} = 1 - \frac{2*2}{2*10} = \frac{4}{5} = 0.8$
  - $\Delta_{fg} = 1 - \frac{3*2}{2*10} = \frac{7}{10} = 0.7$
  - $\Delta_{ii} = 0 - \frac{2*2}{2*10} = -0.2$
  - $\Delta_{ff} = 0 - \frac{3*3}{2*10} = -0.45$
  - $\Delta_{hh} = 0 - \frac{2*2}{2*10} = -0.2$
  - $\Delta_{gg} = 0 - \frac{2*2}{2*10} = -0.2$

  $\Rightarrow \sum = 2 * (0.8 + 0.7 - 0.2 - 0.3 + 0.8 + 0.7) - (0.2 + 0.45 + 0.2 + 0.2) = 5 - 1.05 = 3.95$
  $Q(G, S) = \frac{1}{2*10} * (2.8 + 1.55 + 3.95) =$ **0.415**
- $s'_1 : \{a, b, c, d, e\}$, $s'_2 : \{i, h, f, g\}$ s'_1 :
  - $\Delta_{ab} = 1 - \frac{2*2}{2*10} = \frac{4}{5} = 0.8$
  - $\Delta_{ac} = 1 - \frac{2*4}{2*10} = \frac{1}{5} = 0.6$
  - $\Delta_{ad} = 0 - \frac{2*2}{2*10} = -\frac{1}{5} = -0.2$
  - $\Delta_{ae} = 0 - \frac{2*1}{2*10} = -\frac{1}{10} = -0.1$
  - $\Delta_{bc} = 1 - \frac{2*4}{2*10} = \frac{7}{10} = 0.6$
  - $\Delta_{bd} = 0 - \frac{2*2}{2*10} = -\frac{1}{5} = -0.2$
  - $\Delta_{be} = 0 - \frac{2*1}{2*10} = -\frac{1}{10} = -0.1$

- $\Delta_{cd} = 1 - \frac{4*2}{2*10} = \frac{6}{10} = 0.6$
- $\Delta_{ce} = 0 - \frac{4*1}{2*10} = -\frac{1}{5} = -0.2$
- $\Delta_{de} = 1 - \frac{2*1}{2*10} = \frac{9}{10} = 0.9$
- $\Delta_{aa} = 0 - \frac{2*2}{2*10} = -0.2$
- $\Delta_{bb} = 0 - \frac{2*2}{2*10} = -0.2$
- $\Delta_{cc} = 0 - \frac{4*4}{2*10} = -0.8$
- $\Delta_{dd} = 0 - \frac{2*2}{2*10} = -0.2$
- $\Delta_{ee} = 0 - \frac{1*1}{2*10} = -0.05$

$\Rightarrow \sum = 2 * (0.8 + 0.6 - 0.2 - 0.1 + 0.6 - 0.2 - 0.1 + 0.6 - 0.2 + 0.9) - (0.2 + 0.2 + 0.8 + 0.2 + 0.05) = 5.4 - 1.45 = 3.95$

s'_2 : as before
$\Rightarrow \sum = 3.95$

$Q(G, S) = \frac{1}{2*10} * (3.95 + 3.95) =$ **0.395**

On the basis of the higher modularity we decide to partition the graph G as follows:
$s_1 : \{a, b, c\}$, $s_2 : \{d, e\}$, $s_3 : \{i, h, f, g\}$