Medida de Excentricidade

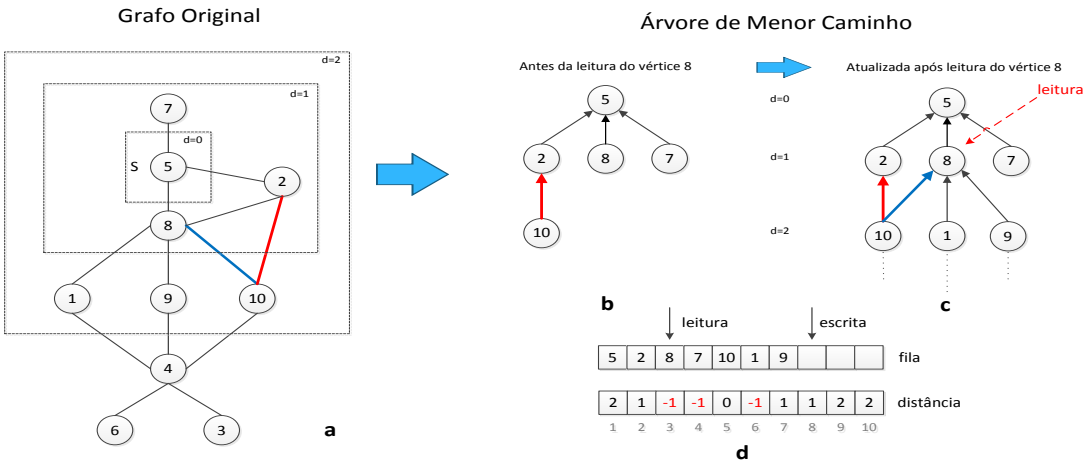
- **Centralidade**
  - Medida subjetiva e contextual
  - **Excentricidade**

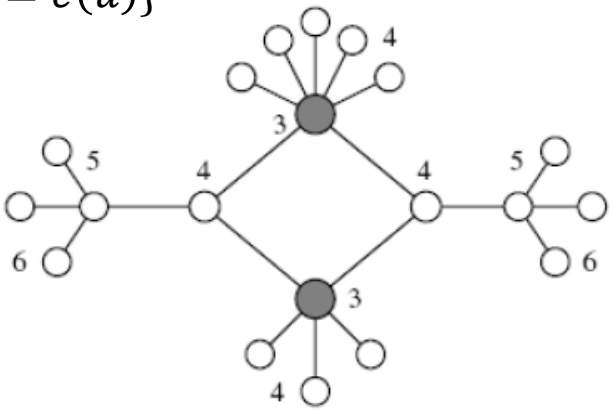
$$e(u) = \max\{d(u, v): v \in V\}$$

$$C_E(u) = \frac{1}{e(u)}$$

  - Centro do Grafo
    - $r(G) = \min\{e(u): u \in V\}$
    - $C(G) = \{u \in V: r(G) = e(u)\}$

- **BFS:**

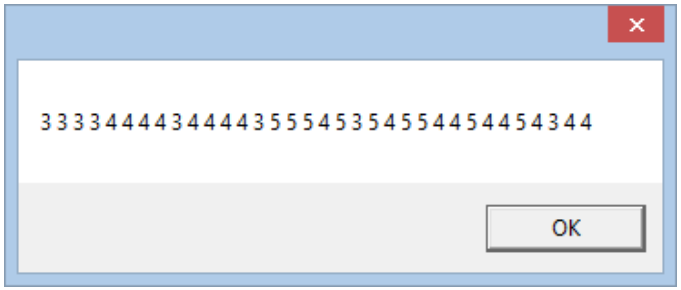




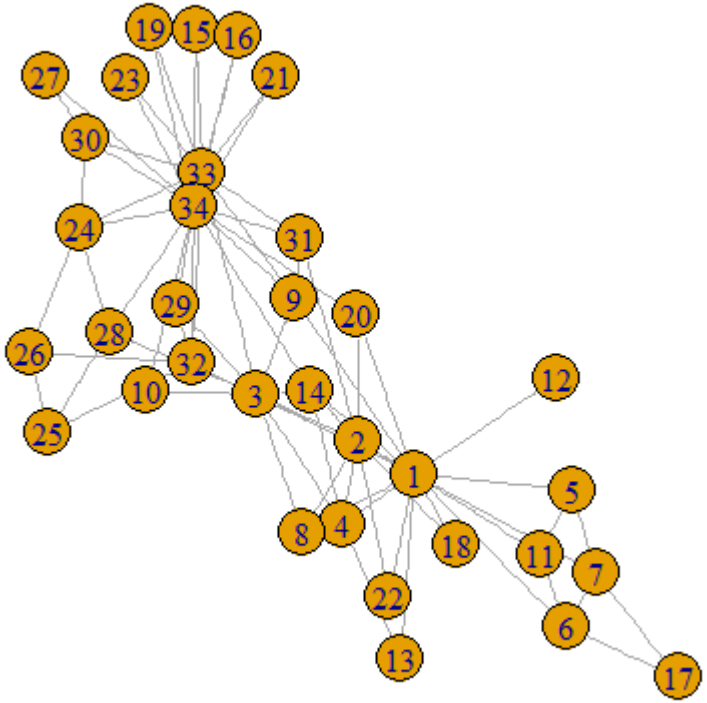
Medida de Excentricidade

- **Nossa implementação (C#):**

```
net = NetworkSamples.ZacharyKarate;
eccValues = Algorithms.Centrality.eccentricity(net);
```



- **Zachary Network:**



- **R + iGraph**

  - > library(igraph)
  - > karate <- graph.famous("Zachary")
  - > eccentricity(karate)
  - [1] 3 3 3 3 4 4 4 4 3 4 4 4 4 3 5 5 5 4 5 3 5 4 5 5 4 4 5 4 4 5 4 3 4 4

Medida de Excentricidade

```
static public int[] eccentricity(int[,] adjMatrix)
{
        int n = adjMatrix.GetLength(0);
        int[] eccent = new int[n];

        int[] distance;
        int[] queue;
        int pRead = 0;  //read pointer
        int pWrite = 0; //write pointer
        int readingNode = -1;

        // BFS Strategy // Breadth-First Search
        for (int startNode = 0; startNode<n; startNode++) // Each possible node as Start Point
        {
                // RESET
                distance = Enumerable.Repeat(-1, n).ToArray();
                queue = Enumerable.Repeat(-1, n).ToArray();
                pRead = pWrite = 0;

                queue[pWrite++] = startNode;
                distance[startNode] = 0; //Only start node at this level (d==0)

                // Walking through graph in BFS
                while (pRead<n)
                {
                        readingNode = queue[pRead++];

                        // Find Neighbors
                        for (int j = 0; j < n; j++)
                        {
                                if (adjMatrix[readingNode, j] == 1)
                                {
                                        if (!queue.Contains(j))
                                        {
                                                queue[pWrite++] = j;
                                                distance[j] = distance[readingNode]+1; // current d + 1
                                        }
                                }
                        }
                }

                // Max?
                eccent[startNode] = distance.Max();
        }

        return eccent;
}
```