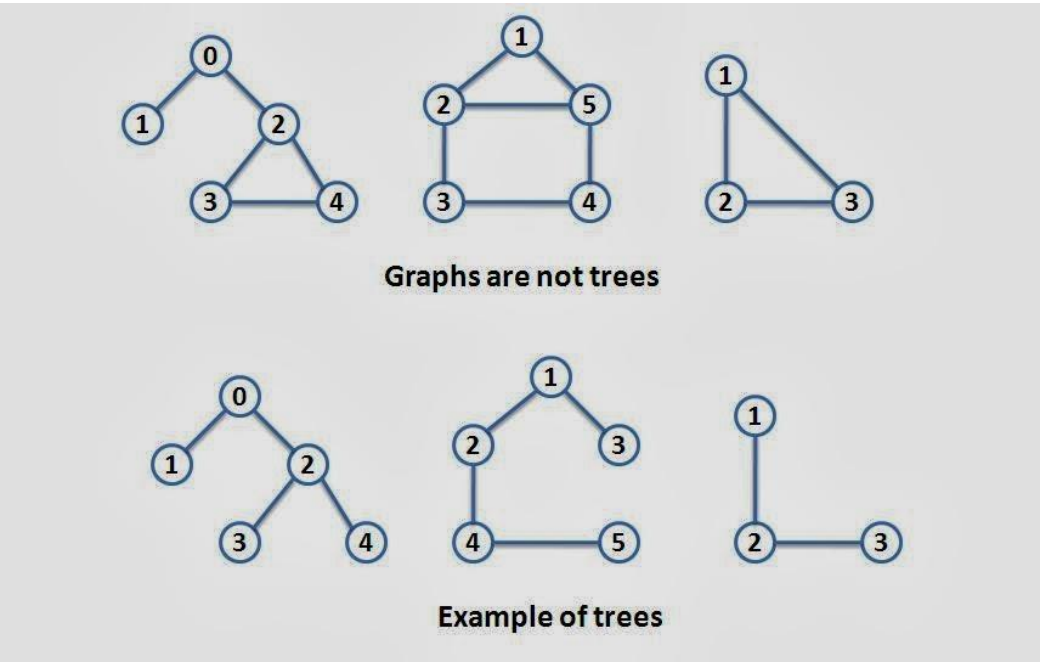
Minimum Spanning Tree

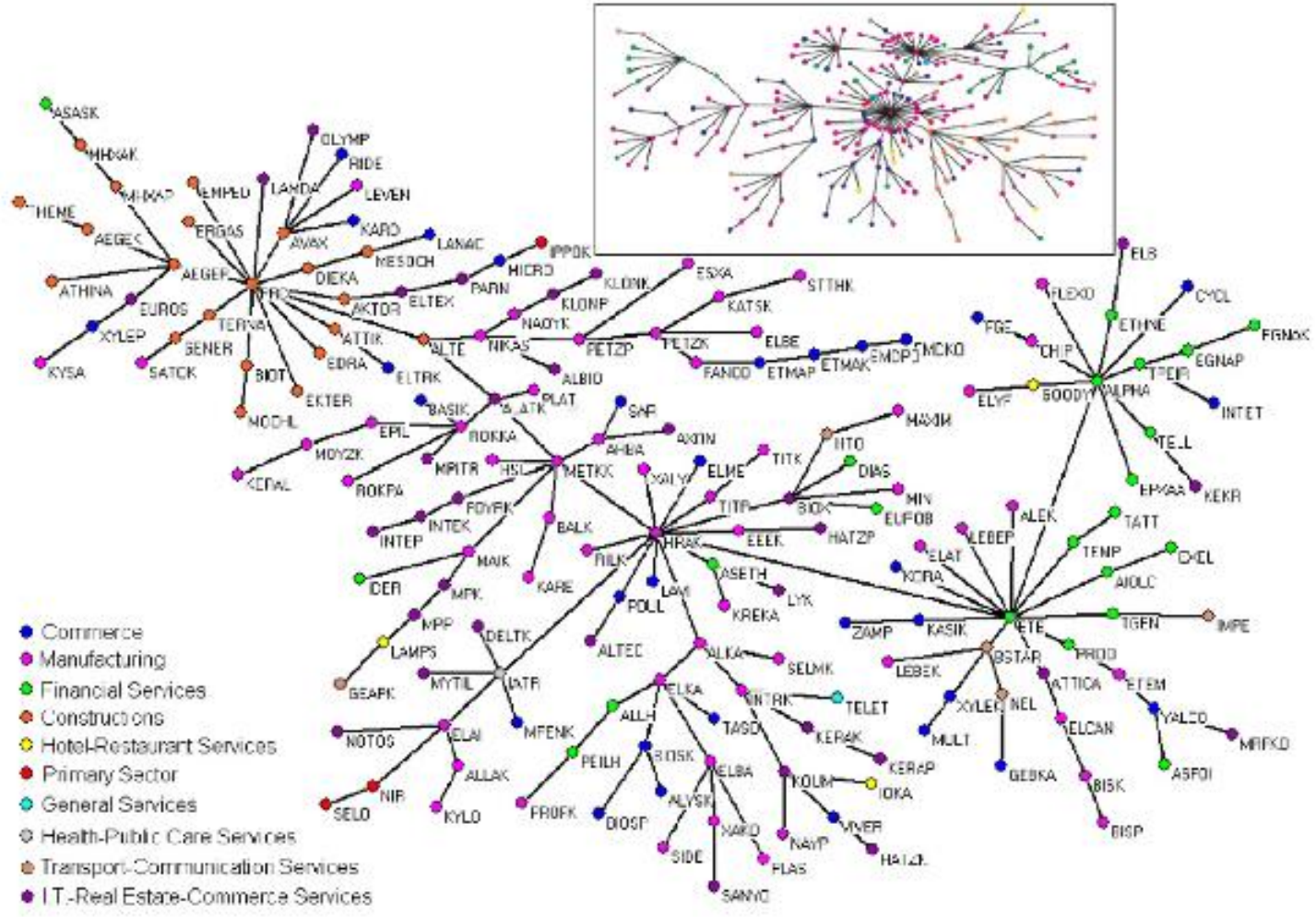
Computer Network Topology

Leonardo Batista - Dezembro 2017

Resumo sobre MST

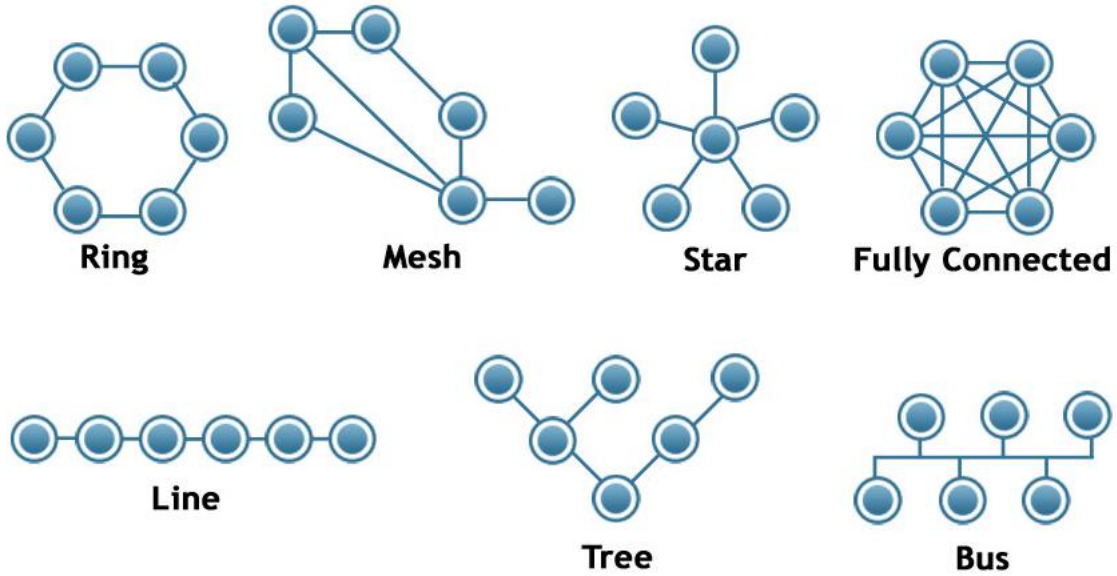
- Grafo --> Árvore
- G={V,E} -> T={V,V-1}

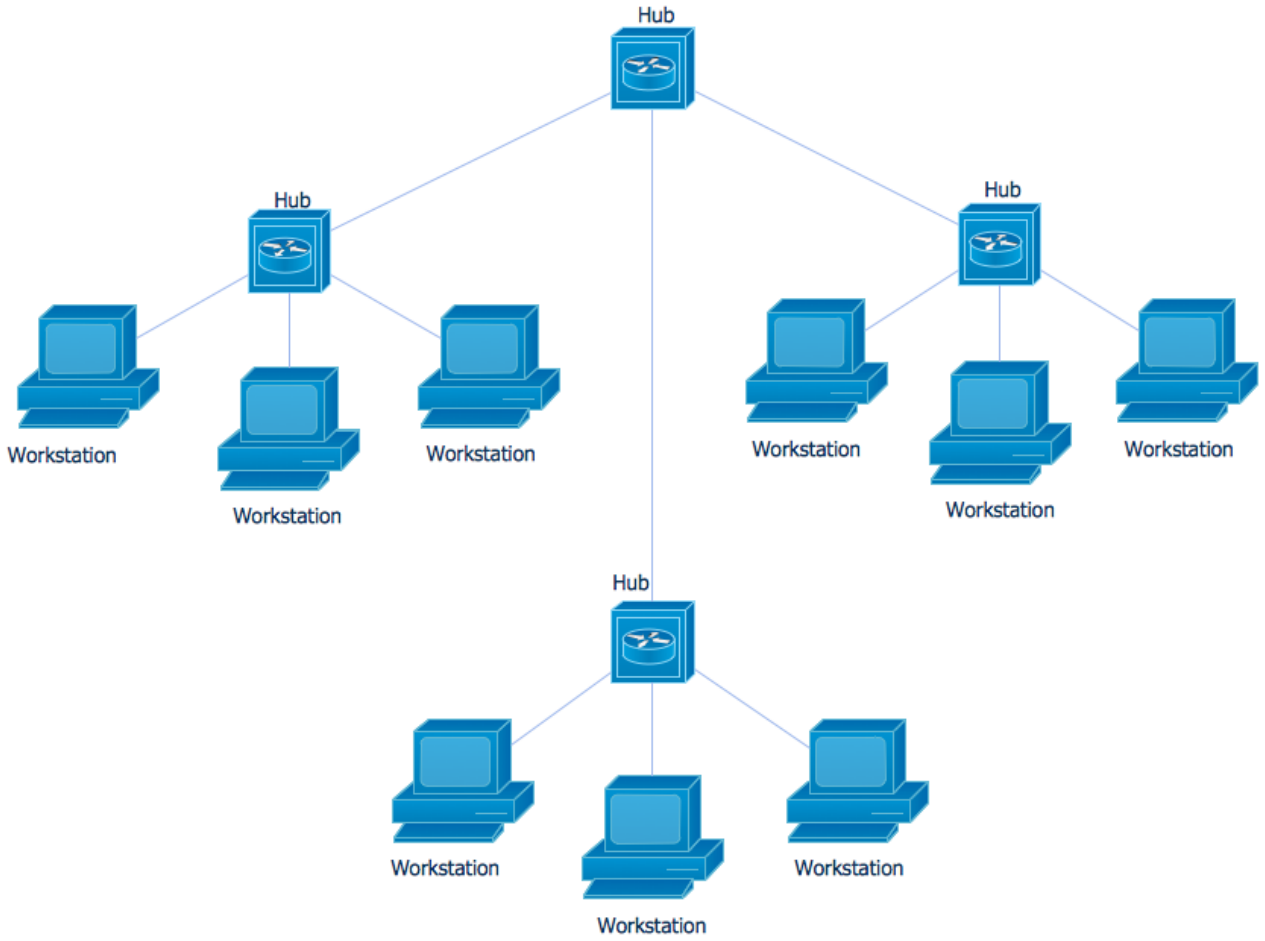




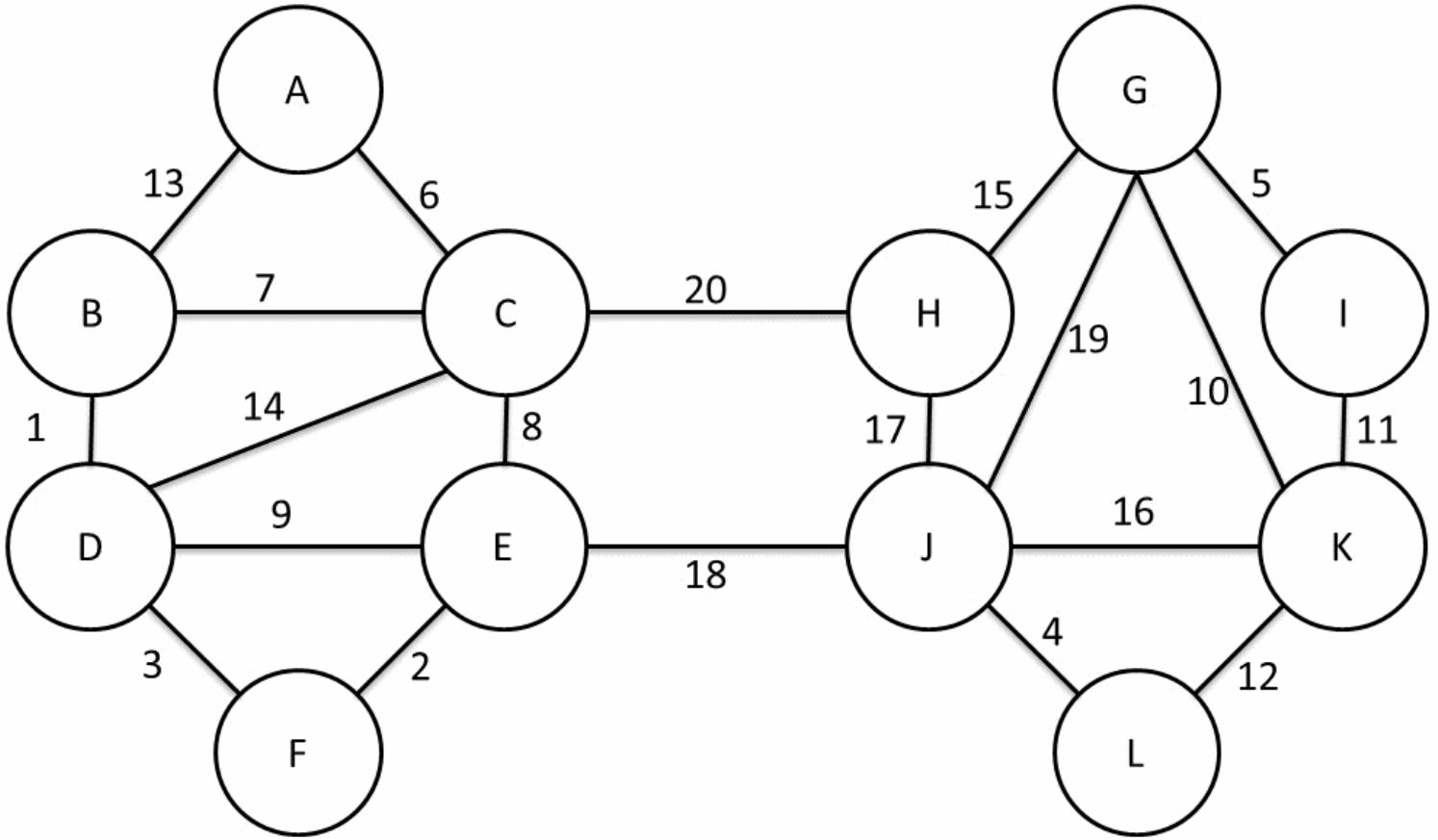
Topologia de Computadores

- *Multiple Spanning Tree Protocol* (IEEE 802.1s)
- IP, MAC, Gateway, Name





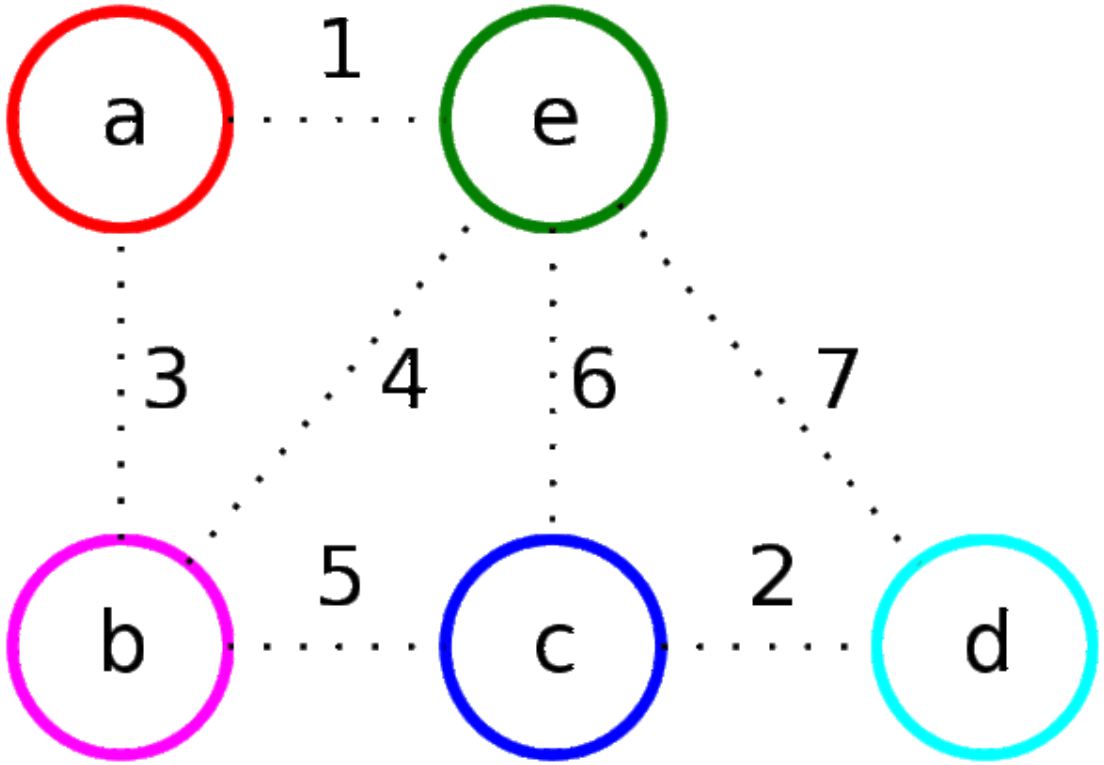
Algoritmo de Borůvka's (1926)



Conceito de Floresta

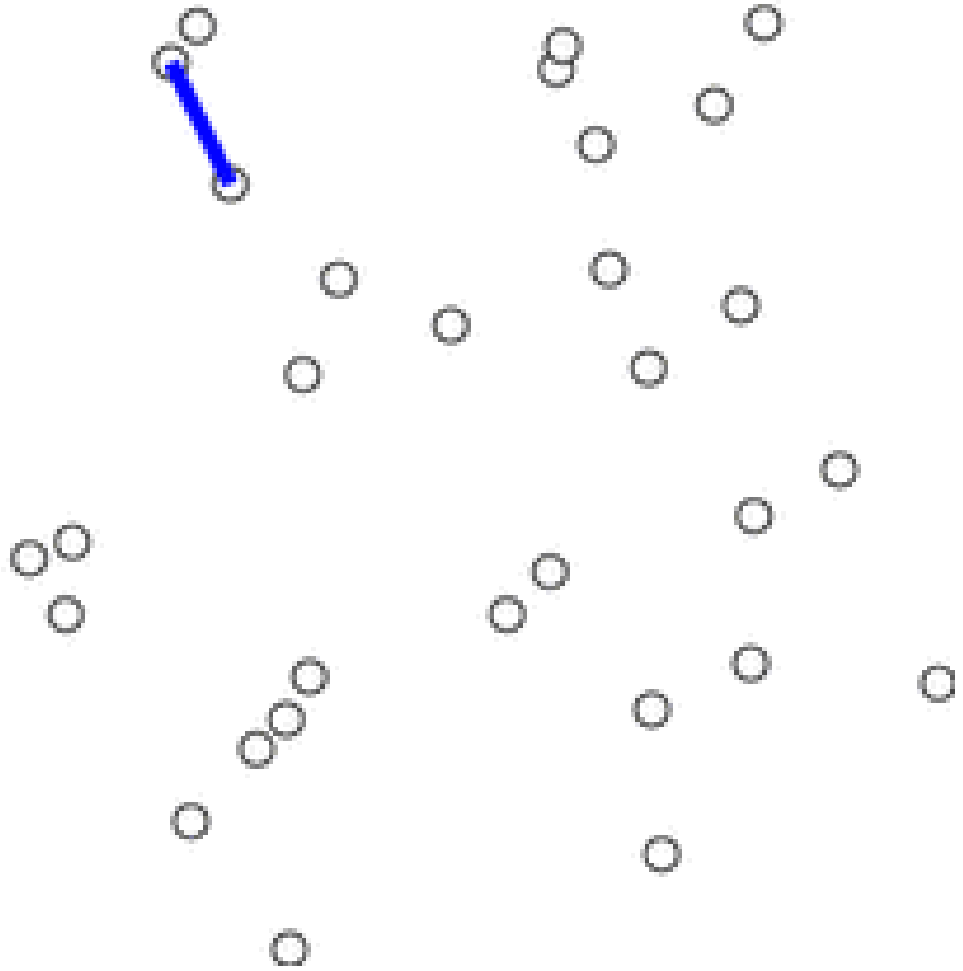
Algoritmo de Kruskal's (1956)

Edge	ab	ae	bc	be	cd	ed	ec
Weight	3	1	5	4	2	7	6



Conceito de menor custo

Algoritmo de Prim’s (1957)

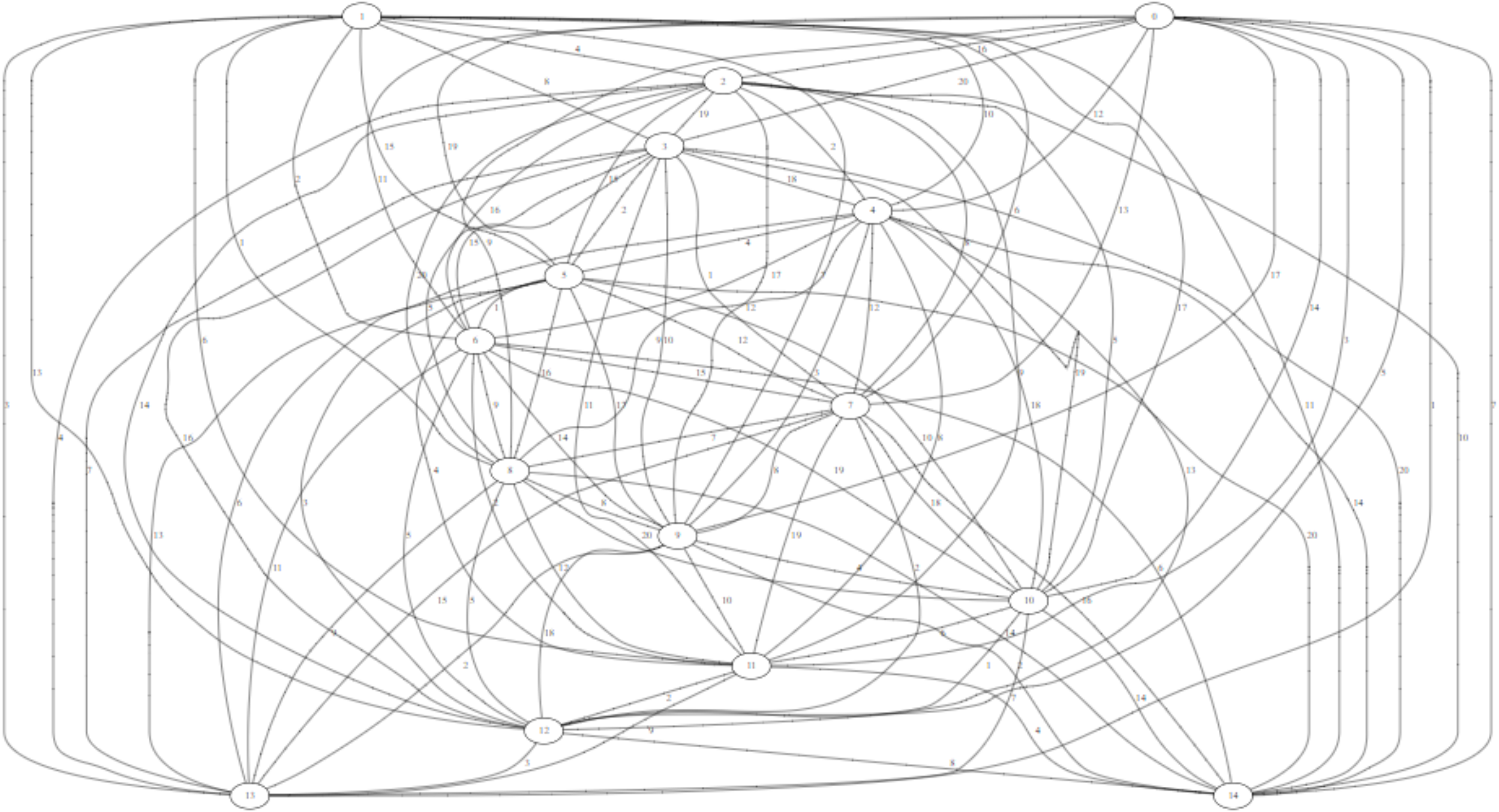


Conceito de centralização

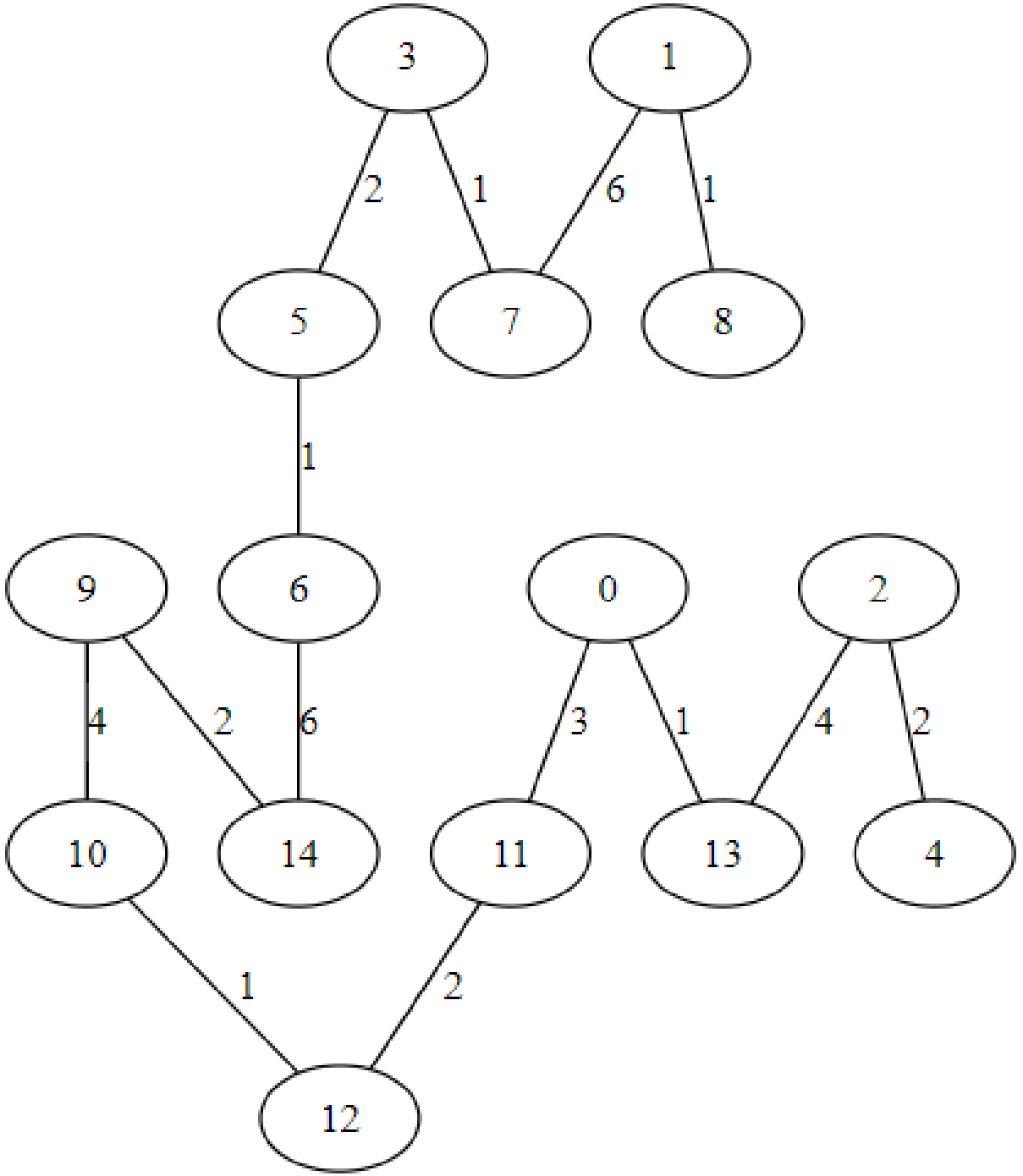
```c
void prims(grafo_t *g, int v){
    int i, j;
    //aresta_t **matriz_adj;

    if (g == NULL){
        perror("prims");
        exit(EXIT_FAILURE);
    }

    if (v > g->n_vertices){
        perror("prims vertices");
        exit(EXIT_FAILURE);
    }

    int k=0;
    while(k != g->n_vertices-1){
        uint8_t w, id, menor = 0xFF;
        for (i=0; i < g->n_vertices; i++){

            w = g->matriz_adj[v][i].weight;

            #ifdef DEBUG
                printf("v:%d, i:%d\n", v, i);
                printf("Peso: %d; Flag: %d\n", w, g->matriz_adj[v][i].flag);
            #endif // DEBUG

            if ((adjacente(g,v,i)) && (!g->matriz_adj[v][i].flag) && (!g->vertices[i].flag)){ //
                if(w <= menor){// condição de definição do menor
                    menor = w;
                    id = i; // vertice adjacente com o menor peso na aresta
                }
                if (menor == w){
                    /*nothing*/
                }else{
                    //printf("Removido: (%d, %d)\n", v, i);
                    rem_adjacencia(g, v, i); // remove a adjacencia atual por que o peso é maior
                }
            }
        }
                    //printf("Removido: (%d, %d)\n", v, i);
                    rem_adjacencia(g, v, i); // remove a adjacencia atual por que o
                }
            }
        }

        g->matriz_adj[v][id].flag = TRUE;
        g->matriz_adj[id][v].flag = TRUE;
        g->vertices[v].flag = TRUE;
        g->vertices[id].flag = TRUE;
        v = id; // próximo vertice
        k++;
    }

    for (i=0; i < g->n_vertices; i++){
        for (j=0; j < g->n_vertices; j++){
            g->matriz_adj[i][j].adj = FALSE;
            if(g->matriz_adj[i][j].flag){
                g->matriz_adj[i][j].adj = TRUE;
            }
            //printf("Flag[%d][%d]: %d\n", i, j, g->matriz_adj[i][j].flag);
        }
    }
}

void kruskal(grafo_t *g){
    if (g == NULL){
```

```
graph {
    0 -- 2 [label = 16];
    0 -- 3 [label = 20];
    0 -- 4 [label = 12];
    0 -- 5 [label = 19];
    0 -- 6 [label = 11];
    0 -- 7 [label = 13];
    0 -- 8 [label = 9];
    0 -- 9 [label = 17];
    0 -- 10 [label = 14];
    0 -- 11 [label = 3];
    0 -- 12 [label = 5];
    0 -- 13 [label = 1];
    0 -- 14 [label = 7];
    1 -- 2 [label = 4];
    1 -- 3 [label = 8];
    1 -- 4 [label = 10];
    1 -- 5 [label = 15];
    1 -- 6 [label = 2];
    1 -- 7 [label = 6];
    1 -- 8 [label = 1];
    1 -- 9 [label = 7];
    1 -- 10 [label = 17];
    1 -- 11 [label = 6];
    1 -- 12 [label = 13];
    1 -- 13 [label = 3];
    1 -- 14 [label = 11];
    2 -- 3 [label = 19];
    2 -- 4 [label = 2];
    2 -- 5 [label = 15];
    2 -- 6 [label = 16];
    2 -- 7 [label = 8];
    2 -- 8 [label = 20];
    2 -- 9 [label = 12];
    2 -- 10 [label = 5];
    2 -- 11 [label = 9];
    2 -- 12 [label = 14];
    2 -- 13 [label = 4];
    2 -- 14 [label = 10];
    3 -- 4 [label = 18];
    3 -- 5 [label = 2];
    3 -- 6 [label = 15];
    3 -- 7 [label = 1];
    3 -- 8 [label = 5];
    3 -- 9 [label = 10];
    3 -- 10 [label = 19];
    3 -- 11 [label = 11];
    3 -- 12 [label = 16];
    3 -- 13 [label = 7];
    3 -- 14 [label = 20];
    4 -- 5 [label = 4];
    4 -- 6 [label = 17];
    4 -- 7 [label = 12];
    4 -- 8 [label = 9];
    4 -- 9 [label = 3];
    4 -- 10 [label = 18];
    4 -- 11 [label = 8];
    4 -- 12 [label = 13];
    4 -- 13 [label = 6];
    4 -- 14 [label = 14];
    5 -- 6 [label = 1];
    5 -- 7 [label = 12];
    5 -- 8 [label = 16];
    5 -- 9 [label = 17];
    5 -- 10 [label = 10];
    5 -- 11 [label = 4];
    5 -- 12 [label = 3];
    5 -- 13 [label = 13];
    5 -- 14 [label = 20];
    6 -- 7 [label = 15];
    6 -- 8 [label = 9];
    6 -- 9 [label = 14];
    6 -- 10 [label = 19];
    6 -- 11 [label = 2];
    6 -- 12 [label = 5];
    6 -- 13 [label = 11];
    6 -- 14 [label = 6];
    7 -- 8 [label = 7];
    7 -- 9 [label = 8];
    7 -- 10 [label = 18];
    7 -- 11 [label = 19];
    7 -- 12 [label = 2];
    7 -- 13 [label = 15];
    7 -- 14 [label = 16];
    8 -- 9 [label = 8];
    8 -- 10 [label = 20];
    8 -- 11 [label = 12];
    8 -- 12 [label = 5];
    8 -- 13 [label = 9];
    8 -- 14 [label = 14];
    9 -- 10 [label = 4];
    9 -- 11 [label = 10];
    9 -- 12 [label = 18];
    9 -- 13 [label = 2];
    9 -- 14 [label = 2];
    10 -- 11 [label = 6];
    10 -- 12 [label = 1];
    10 -- 13 [label = 7];
    10 -- 14 [label = 14];
    11 -- 12 [label = 2];
    11 -- 13 [label = 9];
    11 -- 14 [label = 4];
    12 -- 13 [label = 3];
    12 -- 14 [label = 8];
    }
```

G={15,103}


1
0
2
3
4
5
6
7
8
9
10
11
12
13
14

```
graph {
        0 -- 11 [label = 3];
        0 -- 13 [label = 1];
        1 -- 7 [label = 6];
        1 -- 8 [label = 1];
        2 -- 4 [label = 2];
        2 -- 13 [label = 4];
        3 -- 5 [label = 2];
        3 -- 7 [label = 1];
        5 -- 6 [label = 1];
        6 -- 14 [label = 6];
        9 -- 10 [label = 4];
        9 -- 14 [label = 2];
        10 -- 12 [label = 1];
        11 -- 12 [label = 2];
}
```



T={15,14}

Complexidade dos Algoritmos

Matrix Adjacência: O(|V|^2)

Lista encadeada + árvore binária: O(|E|Log|V|)

Referências

http://www.webgraphviz.com/

http://www.geeksforgeeks.org

https://en.wikipedia.org