Minimum spanning tree

- J. Erickson. Algorithms, Chap 7.5
- Cormen et al. Introduction to algorithms, Chps. 21,23
- J. Kleinberg, E.Tardos, Algorithm design, Chps. 4.4, 4.5
- S. Skiena, The algorithm design manual, Chp. 6.1.
- J. Edmond, How to think about algorithms, Chp. 16.2.3
- S.Skiena and M. Revilla, Programming challenges, Chp. 10.2

Minimum spanning tree

- Dado um grafo conexo, não dirigido, uma *spanning tree* é um subgrafo (que é uma árvore) que liga todos os vértices do grafo

- Na *Minimum Spanning Tree (MST)*, a soma de todos os pesos (dos arcos) é mínima

- O algoritmo de Kruskal cria uma MST juntando gradualmente várias sub-árvores, cada uma uma sub-solução óptima de MST

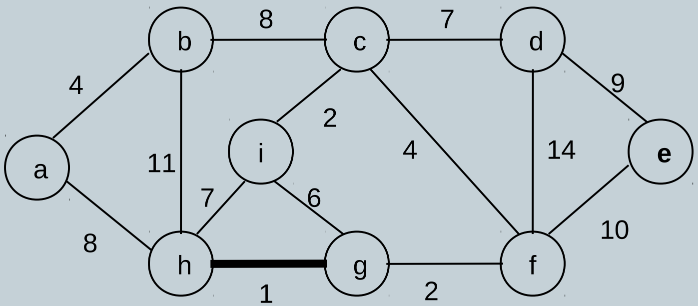
Kruskal

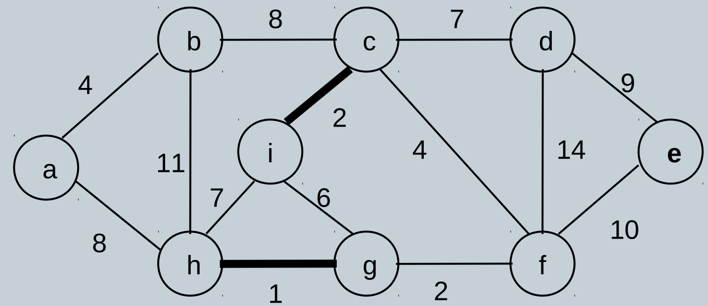
- Inicialmente, cada vértice é uma árvore com um elemento só
  - Make-Set(v)

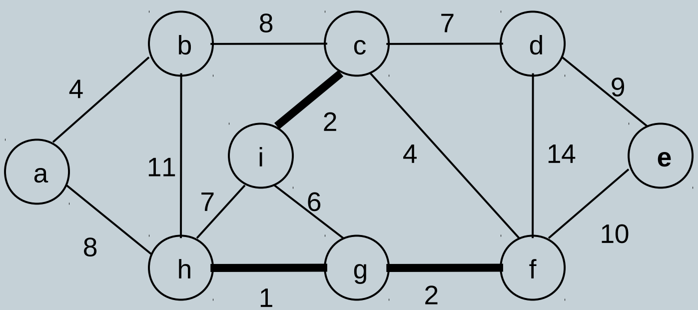
- Em cada iteração, o algoritmo junta duas sub-árvores através do arco livre com peso mais baixo
  - A = A + {(u,v)}
  - Union(u,v)

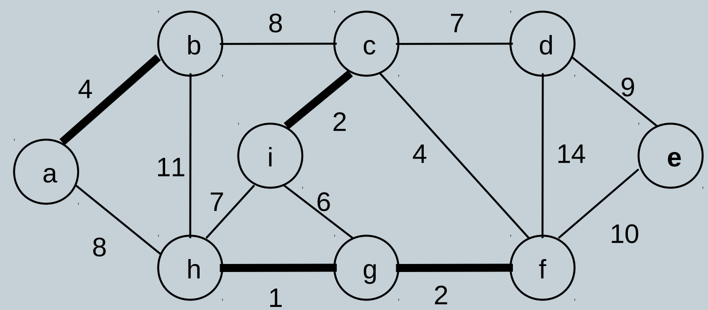
- A função Find-Set(u) devolve um identificador da árvore que contém o vértice *u*

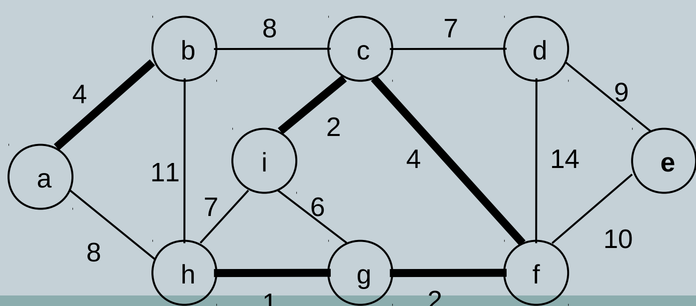
Kruskal


b
8
c
7
d
4
9
2
a
11
i
4
14
e
7
6
8
10
h
g
f
1
2

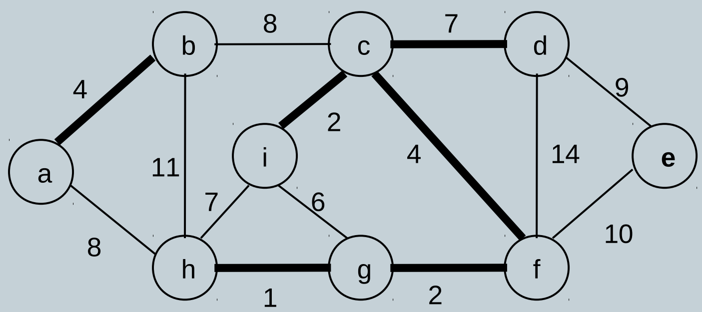

b
8
c
7
d
4
9
2
a
11
i
4
14
e
7
6
8
10
h
g
f
1
2

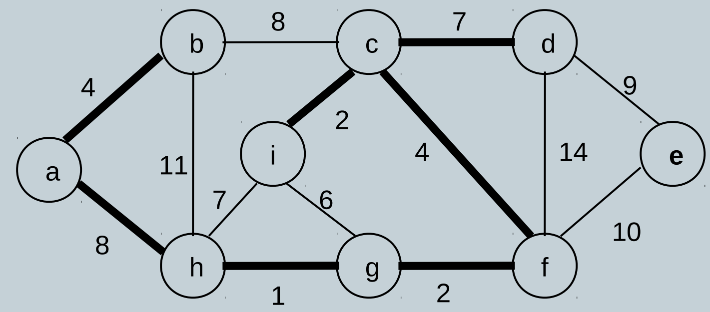

b
8
c
7
d
4
9
2
a
11
i
4
14
e
7
6
8
10
h
g
f
1
2

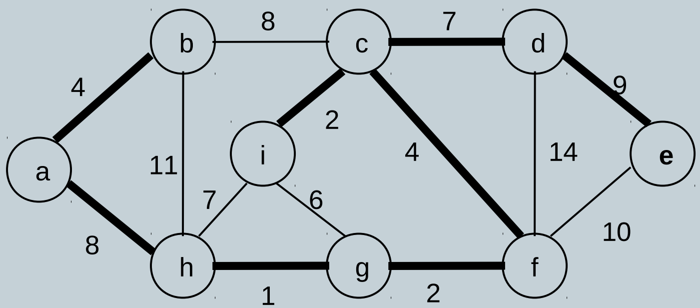

b
8
c
7
d
4
9
2
a
11
i
4
14
e
7
6
8
10
h
g
f
1
2

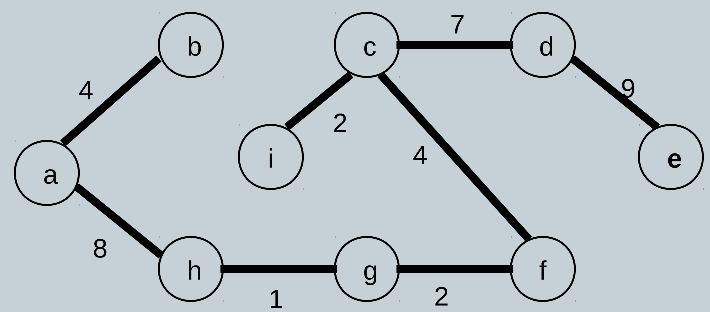

b
8
c
7
d
4
9
2
a
11
i
4
14
e
7
6
8
10
h
g
f
1
2

Kruskal


b
8
c
7
d
4
9
2
a
11
i
4
14
e
7
6
8
10
h
g
f
1
2


b
8
c
7
d
4
9
2
a
11
i
4
14
e
7
6
8
10
h
g
f
1
2


b
8
c
7
d
4
9
2
a
11
i
4
14
e
7
6
8
10
h
g
f
1
2


b
c
7
d
4
9
2
a
i
4
e
8
h
g
f
1
2

Kruskal


```
MST-Kruskal(G,w)

A = {}
for each vertex v in G
   Make-Set(v)
sort the edges of E into non-decreasing order by weight w

for each edge (u,v) in G, in non-decreasing order by
   weight
       if Find-Set(u)!=Find-Set(v)
            A = A + {(u,v)}
            Union(u,v)
return A
```

Kruskal

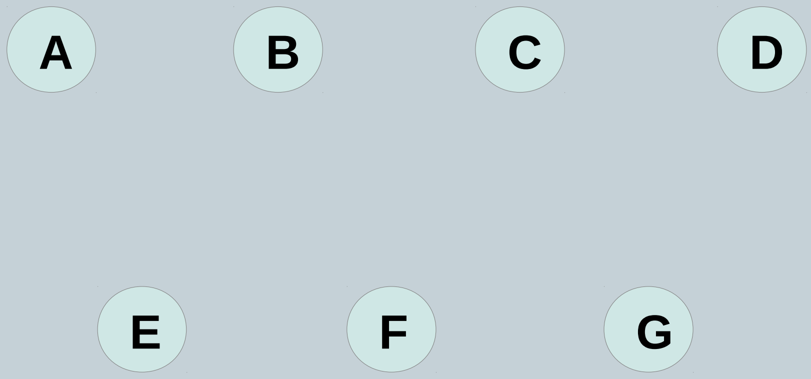
- Estrutura de dados “disjoint set”: mantém um conjunto de elementos particionados em sub-conjuntos disjuntos (sem elementos repetidos).

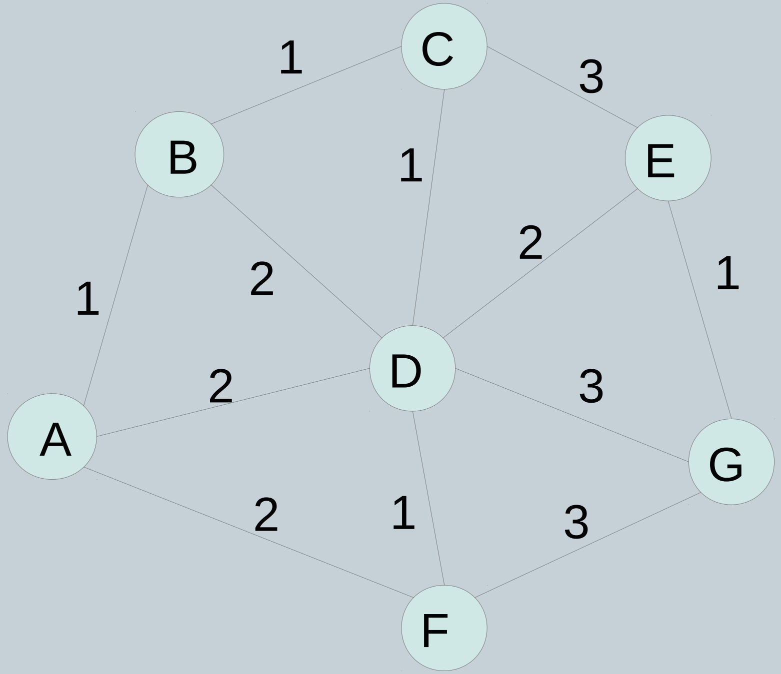
- Union-find efetua duas operações na estrutura de dados “disjoint set”:
  - Find: A que subconjunto um elemento pertence.
  - Union: Une dois subconjuntos num só subconjunto.

- Existe um ciclo se, para nós a e b, Find(a) = Find(b)

Kruskal

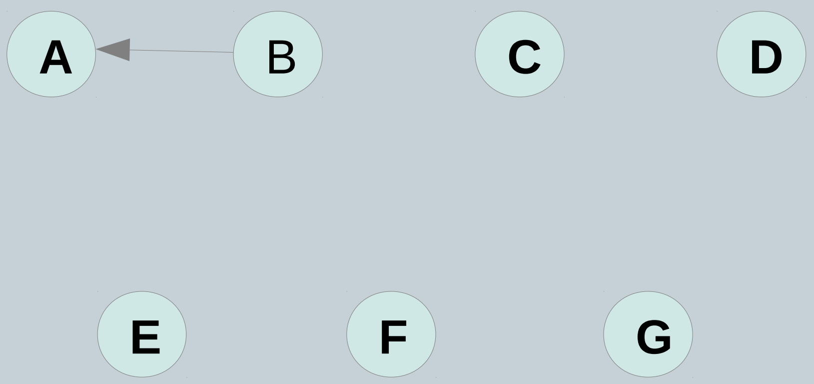
- Make set: Each node is an element of singeton set and is the root.

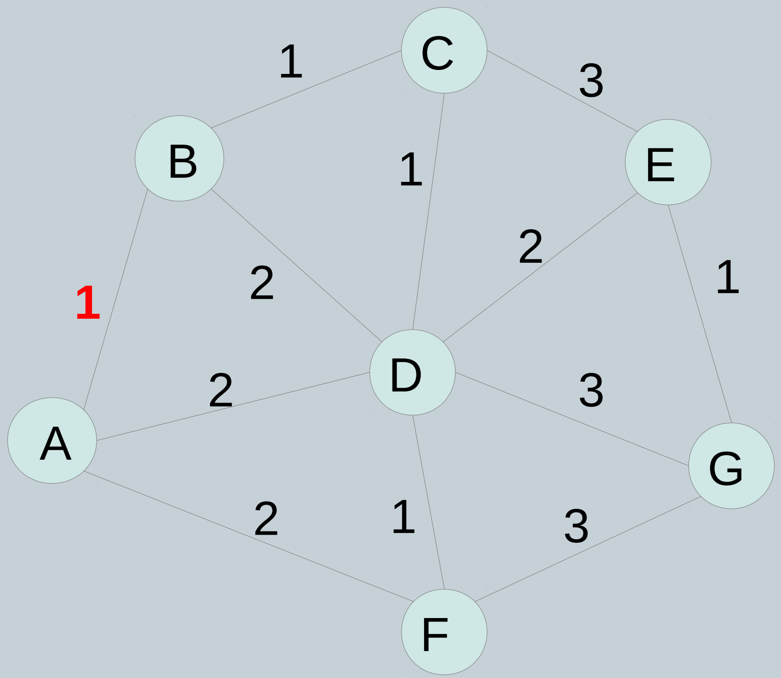




Kruskal

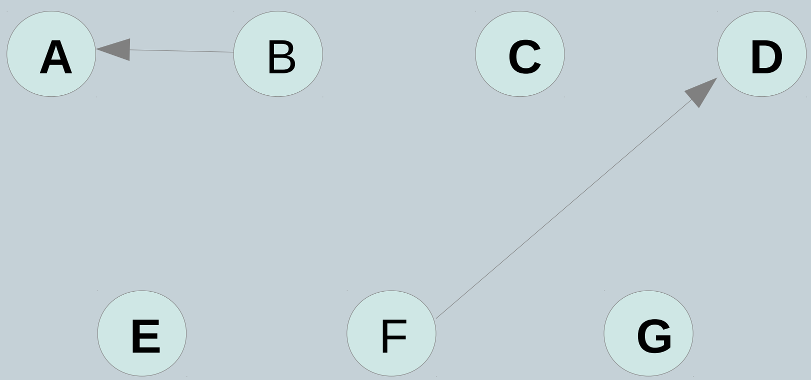
- Merge sets that are connected by an edge:connect the roots

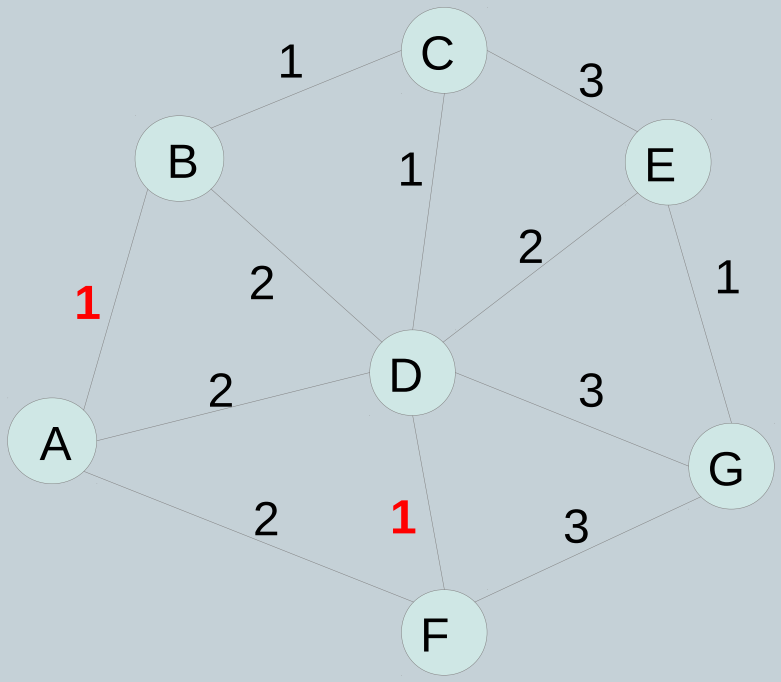




Kruskal

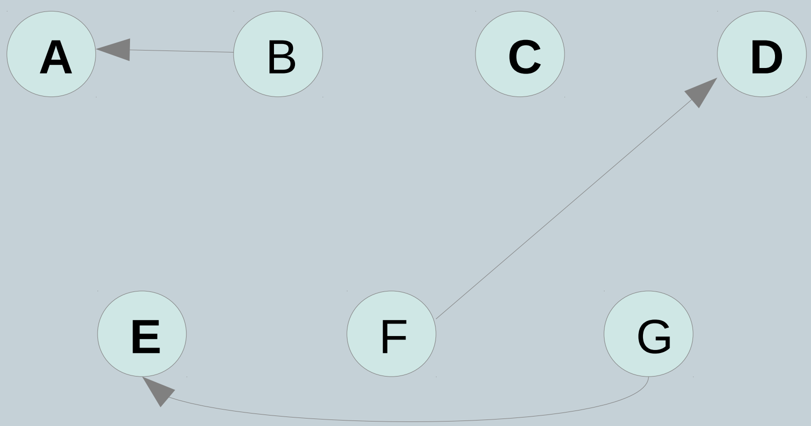
- Merge sets that are connected by an edge:connect the roots

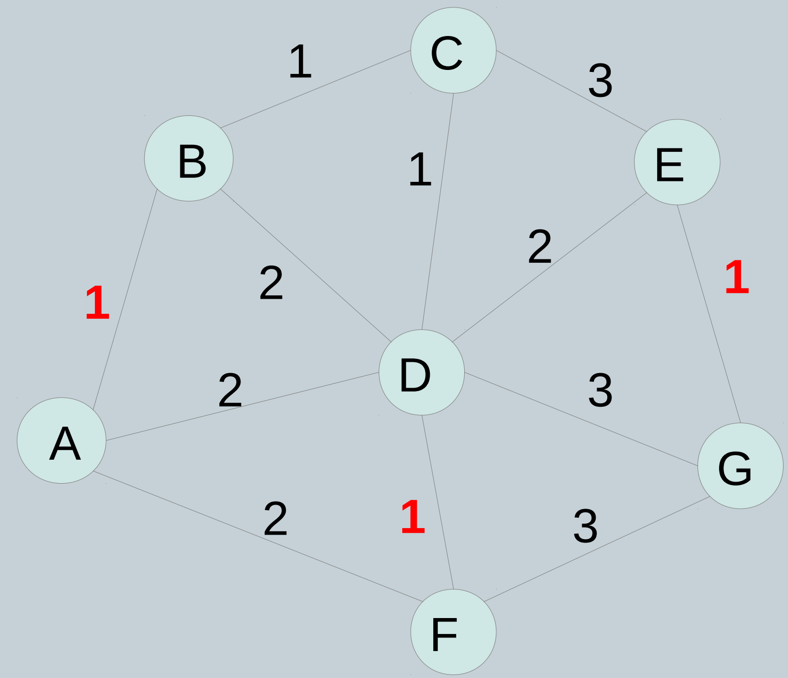




Kruskal

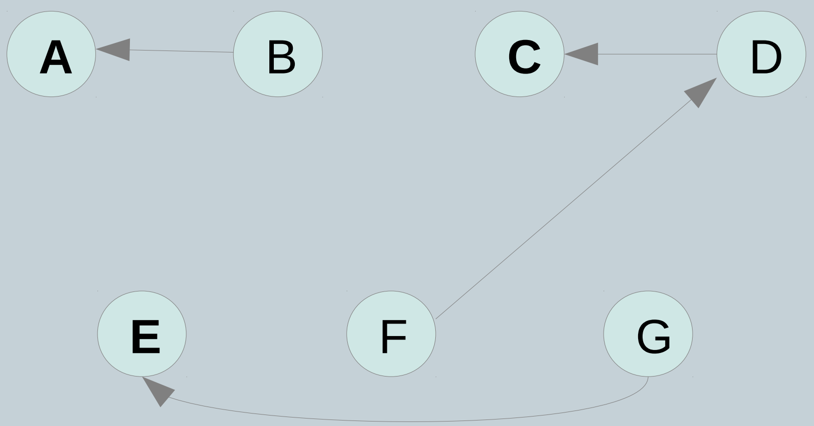
- Merge sets that are connected by an edge: connect the roots

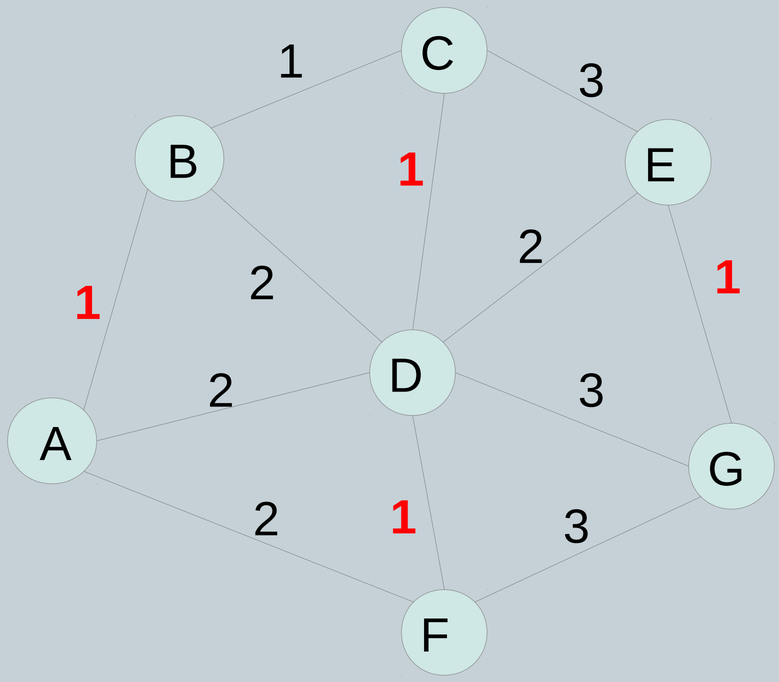




Kruskal

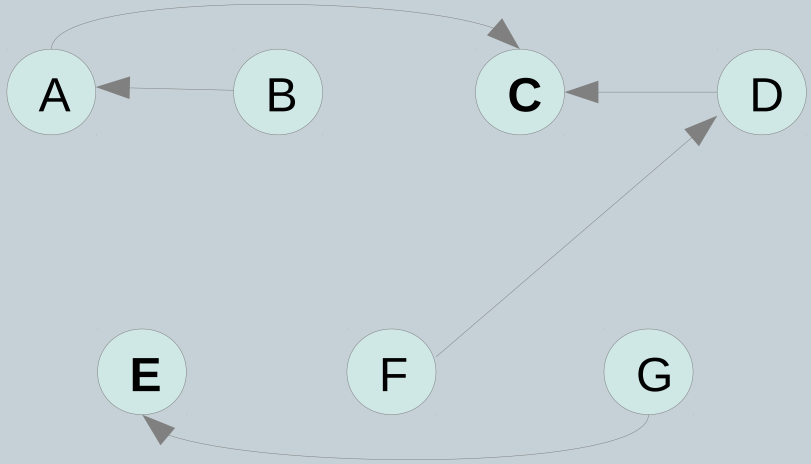
- Merge sets that are connected by an edge: connect the roots

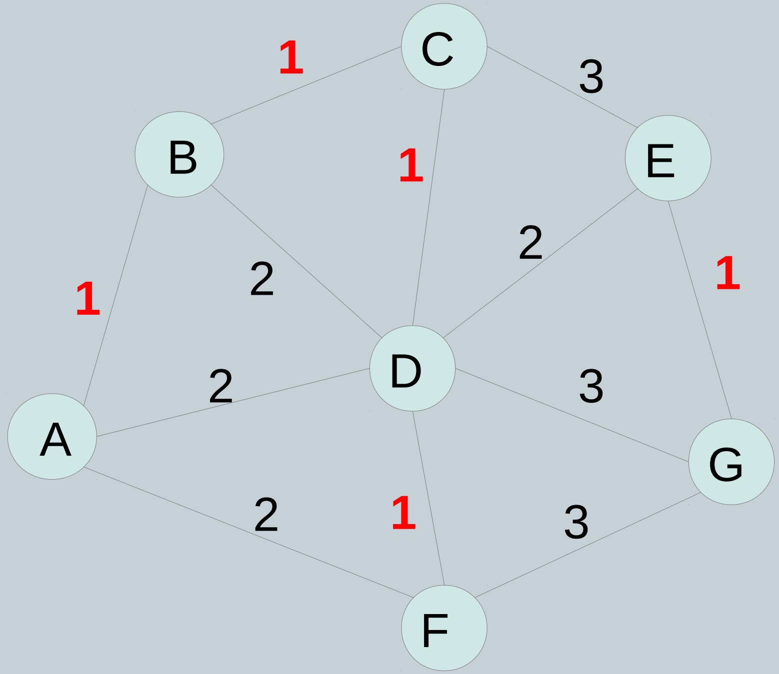




Kruskal

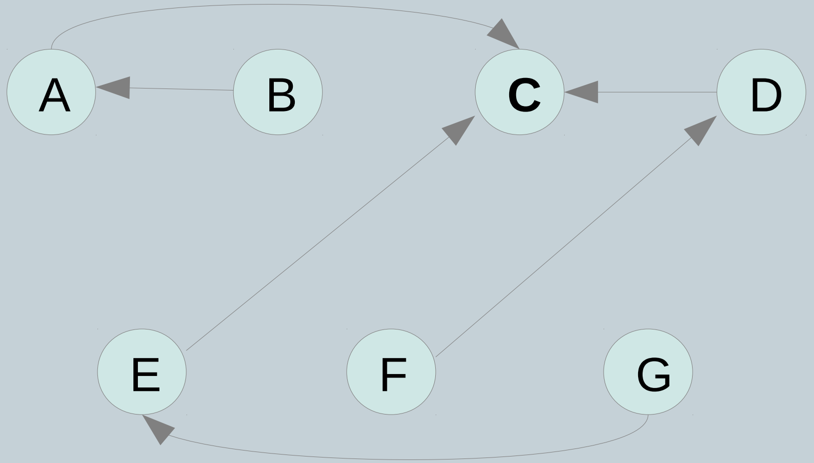
- Merge sets that are connected by an edge: connect the roots

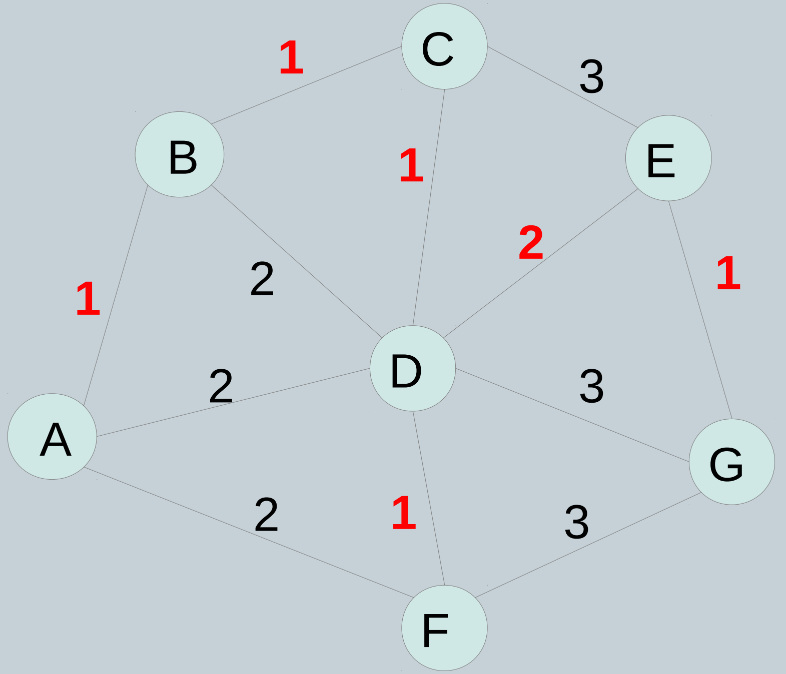




Kruskal

- Merge sets that are connected by an edge: connect the roots





Kruskal

- Sort |E| edges takes O(|E| log |V|) time.
  (Note that O(log|E|) = O(log|V|))

- Make-set takes O(|V|) time

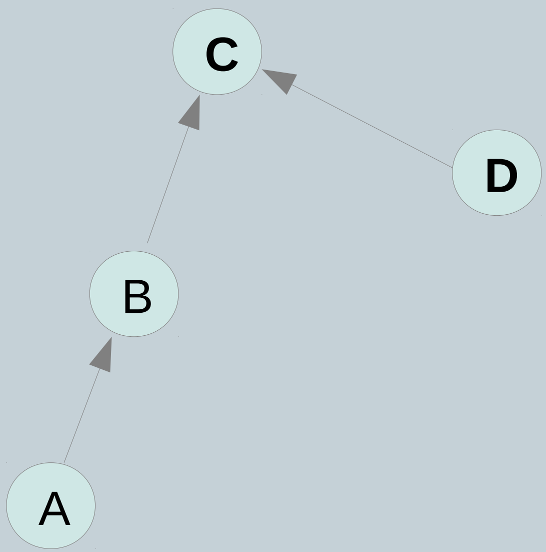
- Find-set takes O(|V|) time

- Union takes O(1) time

Then Kruskal takes O(|E||V|) time. But it can be improved..

Kruskal

- Improvement on the time of Find-set:



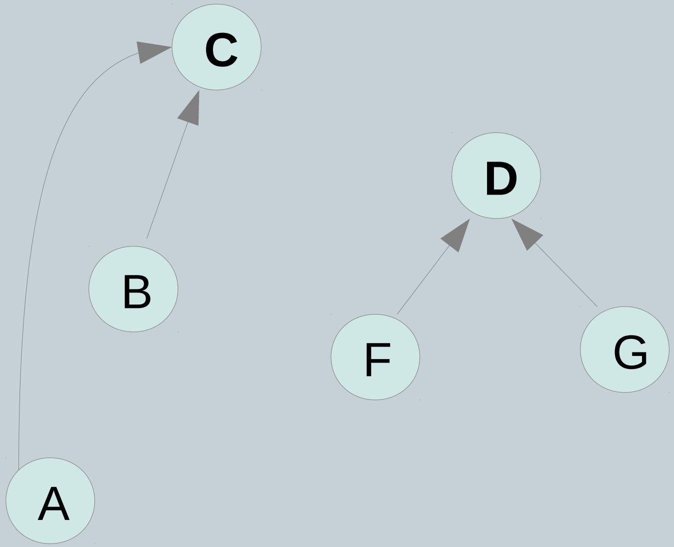
Connect the pointer of the small tree to the pointer of large tree to keep the final tree with reduced height.

The maximum height is log(|V|)

Then, Kruskal takes O(|E| log |V|) time.

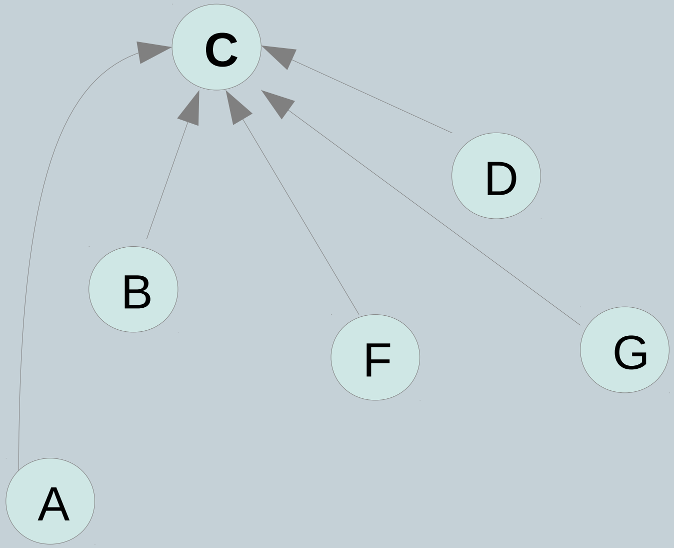
Kruskal

- Further improvement on the time of Find-set: Path compression



Kruskal

- Further improvement on the time of Find-set: Path compression



Connect all descendents to the root of the new tree

This takes O(log n) but if repeated p times, it takes O(p log* n)

Then, Kruskal takes O(|E| log* |V|) time.

Kruskal

- log* n is the iterated logarithm

$$\log^* n := \begin{cases} 0 & \text{if } n \le 1; \\ 1 + \log^*(\log n) & \text{if } n > 1 \end{cases}$$

Example: log*(2^{65536}) = 5

Union-Find


```
void link(int a, int b){

    if(rank[a]>rank[b])
        set[b]=a;
    else{
        set[a]=b;
        if(rank[a]==rank[b])
            rank[b]++;
    }
}

int find(int a){

    if(set[a]!=a)
        set[a]=find(set[a]);
    return set[a];
}

void union_find(int a, int b){

    link(find(a), find(b));
}
```

Inicialização:

```
…
for(i=0; i<N; i++){
    set[i]=i;
    rank[i]=0;
}
```

Articulation Points

- Seja G um grafo conexo não dirigido

- Um ponto de articulação *(articulation point)* de G é um vértice cuja remoção torna G um grafo desconexo

Articulation Points

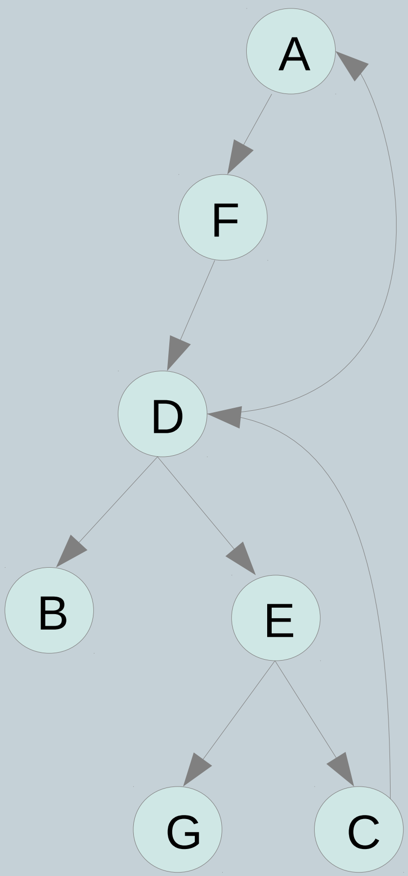
- Seja G um grafo conexo não dirigido

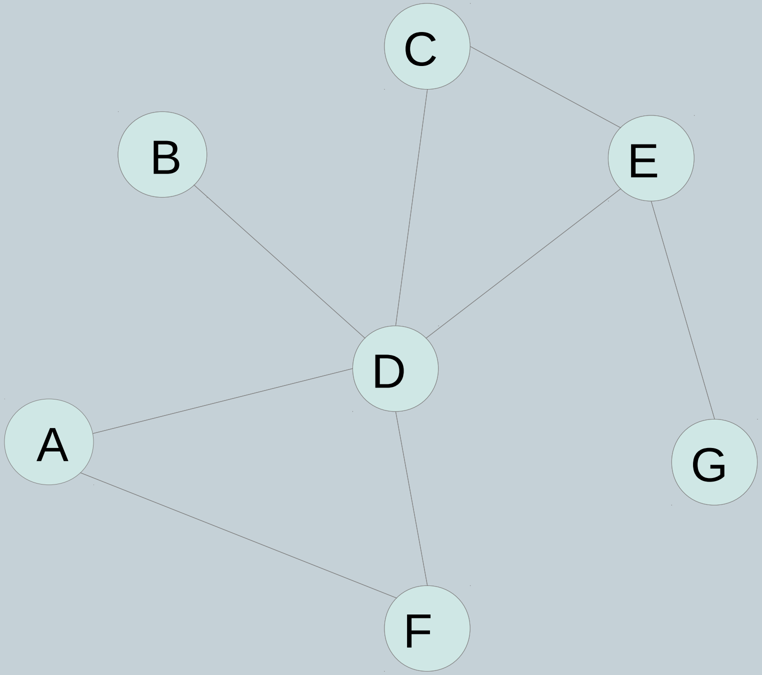
- Um ponto de articulação *(articulation point)* de G é um vértice cuja remoção torna G um grafo desconexo

- Implementação naïve: Para cada nó i, retirar esse nó do grafo, correr DFS e verificar conexidade. Tem complexidade $O(|V|\ (|V| + |E|))$.

Articulation Points

- Seja $\mathbf{G_t}$ a árvore DFS de um grafo





Articulation Points

- Seja $\mathbf{G_t}$ a árvore DFS de um grafo

- A raíz, **r**, de $\mathbf{G_t}$ é um ponto de articulação se e só se
  - **r** tem pelo menos dois filhos

- Um vértice **v** (não raíz) de $\mathbf{G_t}$ é um ponto de articulação se e só se:
  - **v** tem um filho **w** em $\mathbf{G_t}$ tal que não existe nenhuma ligação (back edge) entre **w** (ou descendentes) e um antecessor de **v**

No exemplo anterior, nós D e E são pontos de articulação.

Articulation Points

- Definição
  - Sendo dfs[v] o índice da travessia dfs no vértice **v**

$$low[v] = min\begin{Bmatrix} dfs[v] \\ dfs[x_i] \\ low[w_i] \end{Bmatrix}$$

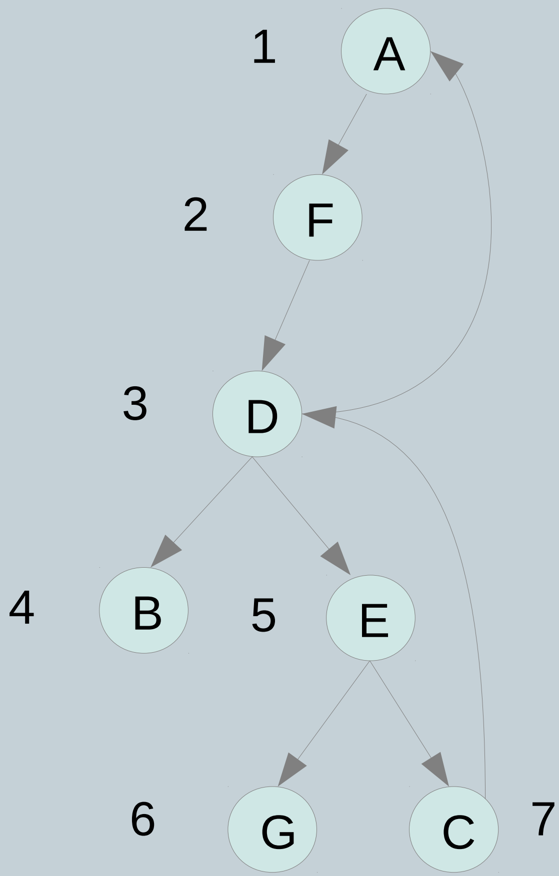
  - com $\mathbf{x_i}$ um vértice antecessor de **v** com uma ligação a **v** (com back edge) e $\mathbf{w_i}$ os filhos de **v;**
  - **low[v]** é o menor índice de travessia dfs de um nó que pode ser alcançado por **v** (por back edge).

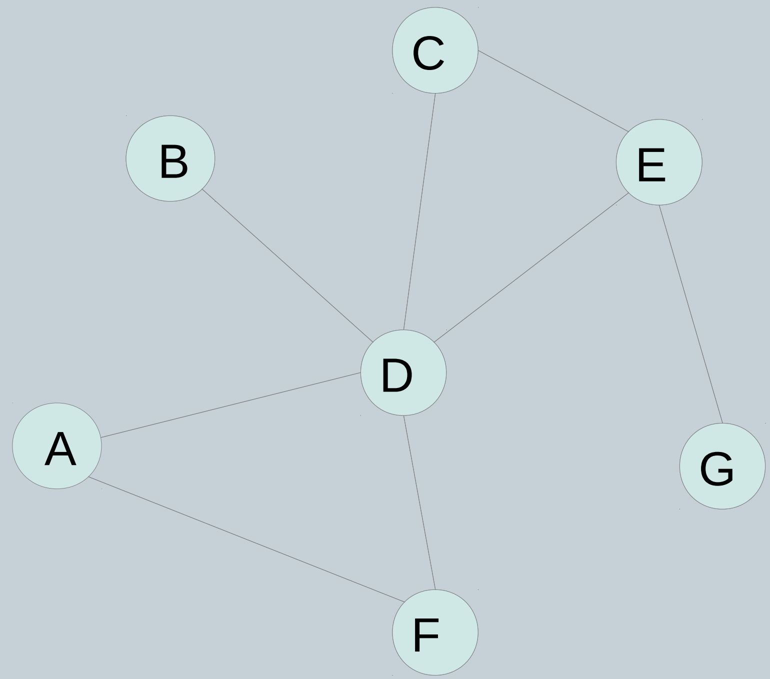
- Um vértice **v** (não raíz) é um ponto de articulação se e só se tiver um filho **w** tal que:

$$low[w] >= dfs[v]$$

Articulation Points

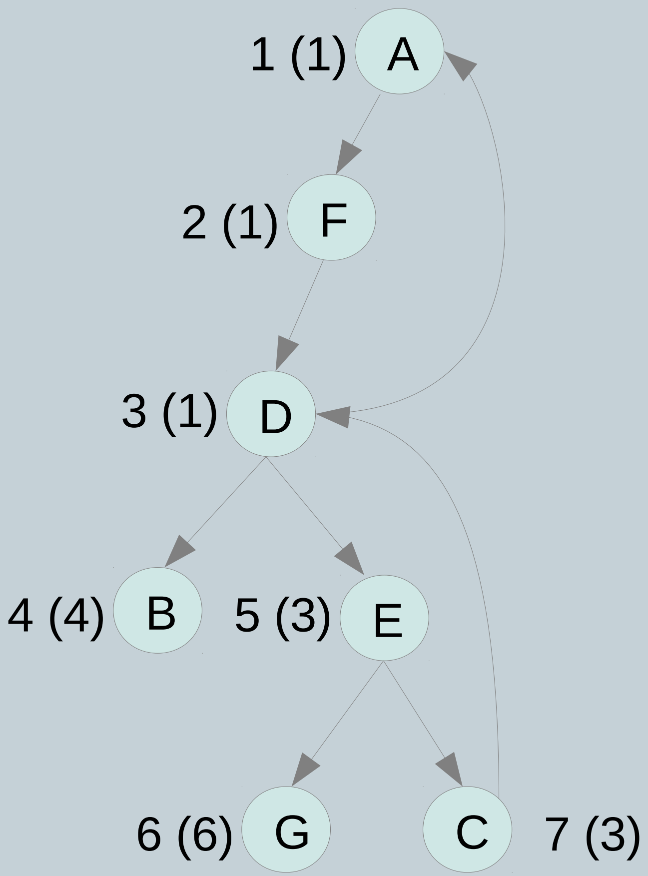
- Calcular dfs(i)

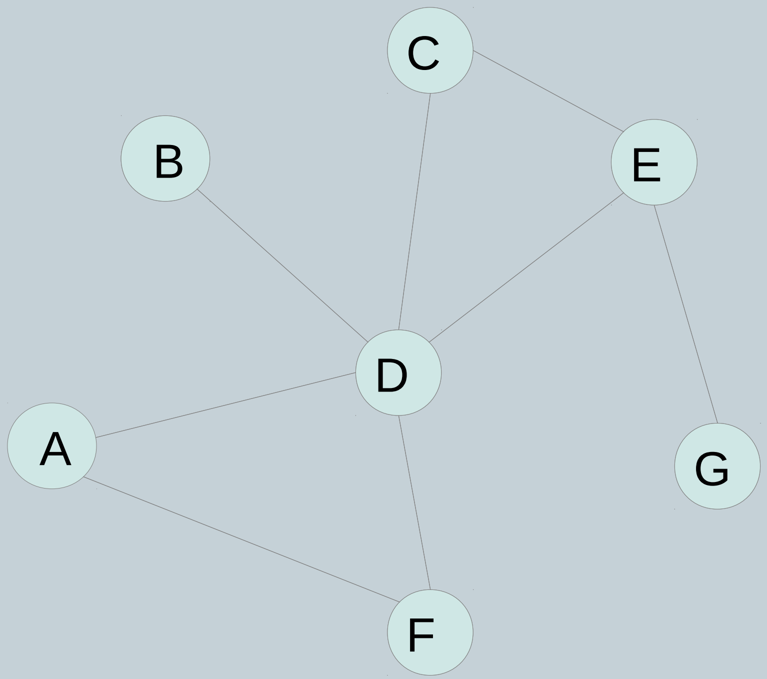




Articulation Points

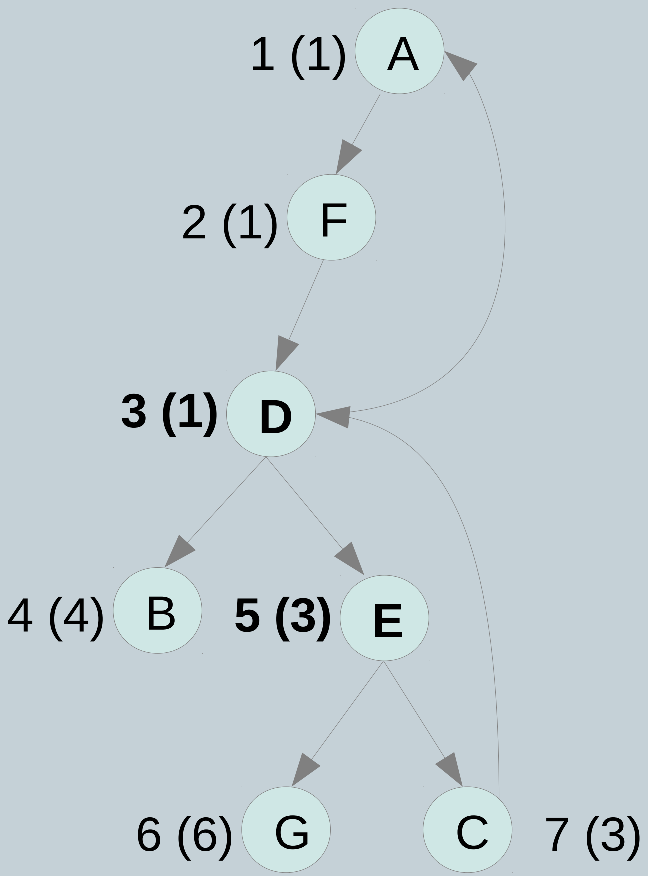
- Calcular low(i)

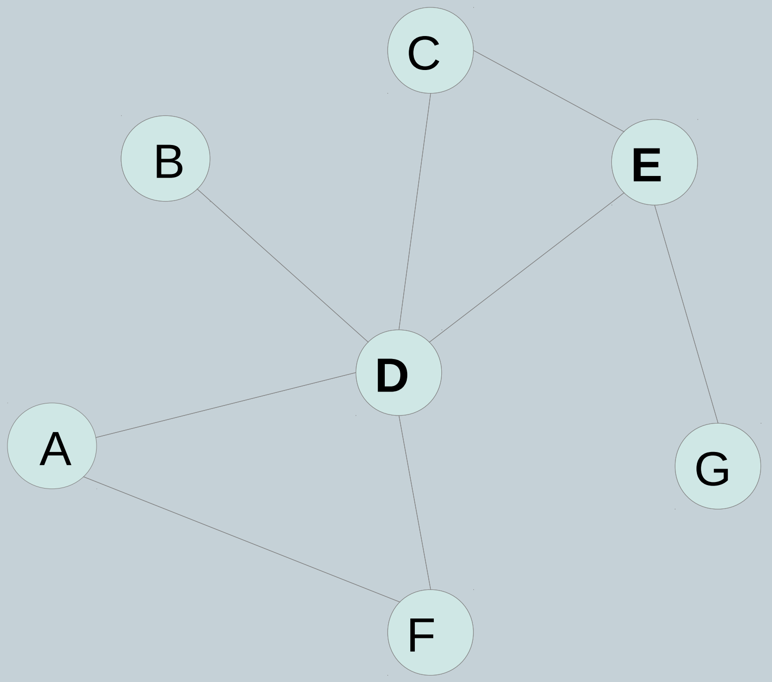




Articulation Points

- Pontos de articulação





Articulation Points


```
procedure Art(v)
low[v]=dfs[v]=time=time+1;
for each (v,w)  in E do
    if (dfs[w]=-1) then
          Art(w)
          low[v] = min{low[v], low[w]}
          if(dfs[v]=1) and (dfs[w]!=2) then
                      “v is an articulation point”
          if(dfs[v]!=1) and (low[w]>=dfs[v])then
                      “v is an articulation point”
    else if (w is not parent of v)
          low[v] = min{low[v], dfs[w]}
```