Articulações e Pontes

Sumário

1. Revisão de conceitos
2. Motivação
3. Definições
4. Exemplos
5. Algoritmo trivial
6. Tarjan
7. Execução
8. Implementação
9. Lista de questões

Revisão de conceitos

Revisão de conceitos

- Componente conexa:
  - Dado um grafo ***G*** não direcionado, uma componente conexa é um subgrafo máximo ***G'*** (um conjunto de vértices e arestas de ***G***) onde para qualquer par de vértices ***u***, ***v*** existe um caminho de ***u*** para ***v***;
- DFS:
  - Busca em profundidade, algoritmo de busca em grafos que visita vértices adjacentes recursivamente;
- Spanning subtree:
  - Subárvore gerada ao visitar os vértices durante a DFS, com as arestas utilizadas na descoberta dos vértices;

Motivação

Motivação do problema

- Dada uma rede de computadores conectados por cabos identificar possíveis pontos críticos (pontos que se em caso de falha iriam interromper a conexão e funcionamento do resto da rede)

Definições

Definições - Articulação

Dado um grafo ***G*** não direcionado

- Dado um grafo ***G*** conexo, um vértice ***v*** é uma articulação se ao ser removido (junto com as arestas incidentes ao vértice ***v***) o grafo ***G*** se torna desconexo;
- De forma mais genérica, um vértice ***v*** é uma articulação se ao ser removido o número de componentes conexas do grafo ***G*** aumenta;

Definições - Ponte

Dado um grafo ***G*** não direcionado

- Se ***G*** é conexo, uma aresta ***e*** é uma ponte se ao ser removida o grafo ***G*** se torna desconexo;
- De forma mais genérica, uma aresta ***e*** é uma ponte se ao ser removida o número de componentes conexas do grafo ***G*** aumenta;

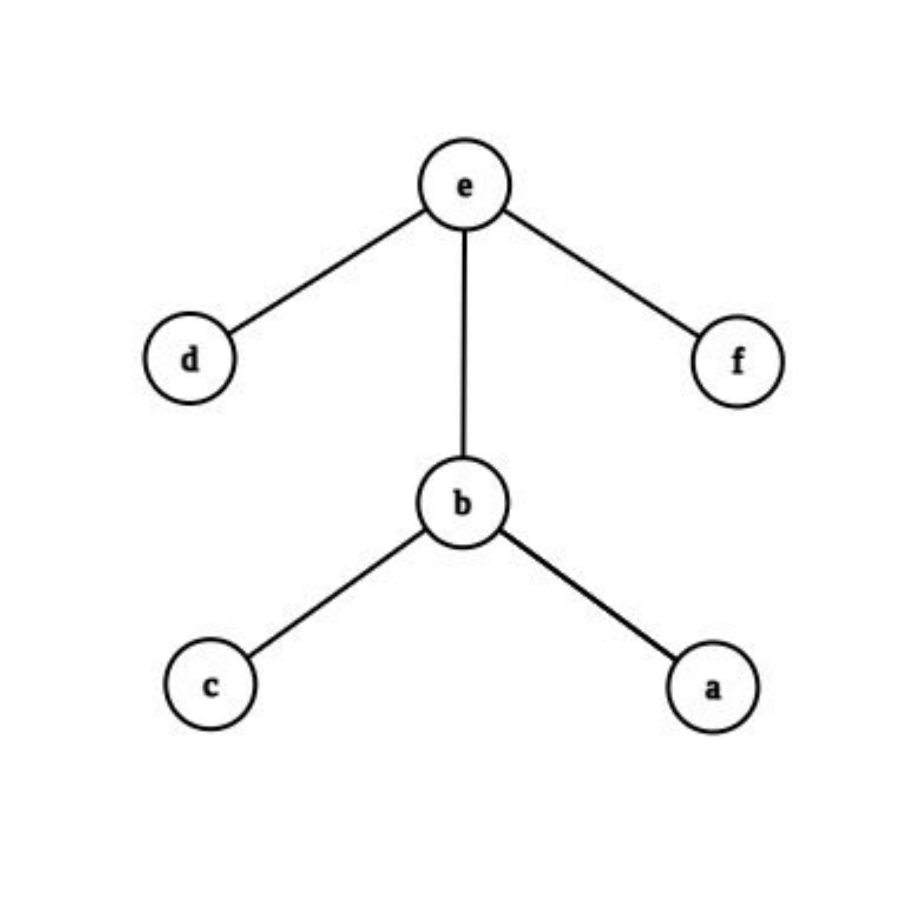
Definições - Grafo biconexo

Dado um grafo ***G*** não direcionado e conexo

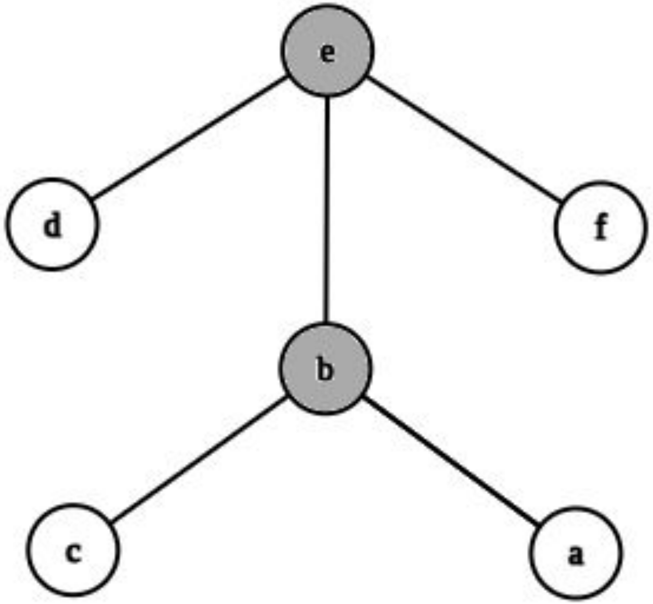
- Um grafo é dito biconexo se para qualquer par de vértice (u, v) existem dois caminhos entre u e v com conjuntos de vértices distintos (além de u e v); Em outras palavras, se o grafo não possui articulações/pontes, ele é biconexo;
- Em inglês: *edge-biconnected* ou *2-edge-connected*

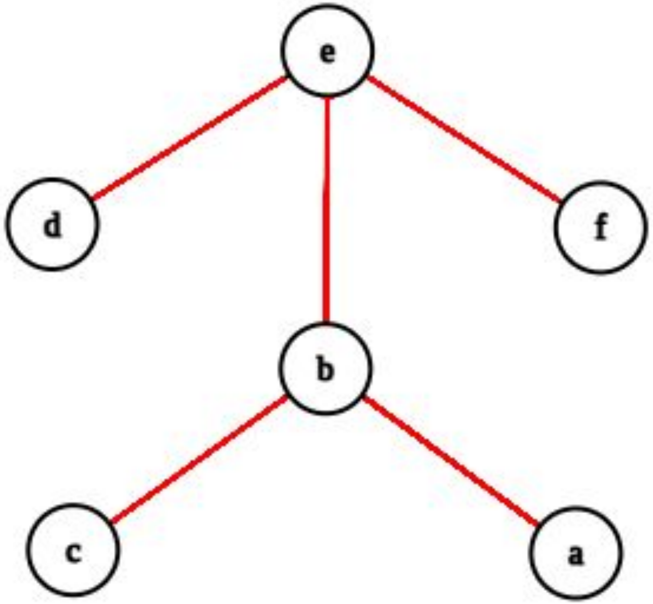
Exemplos

Exemplo 1

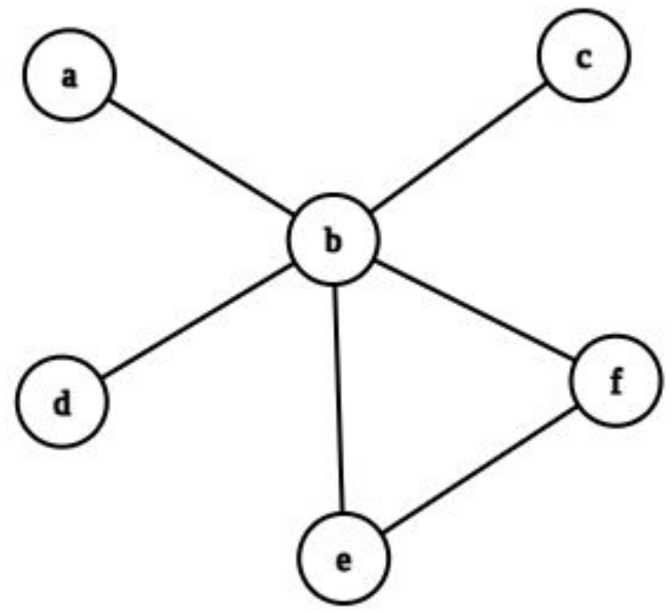


Exemplo 1

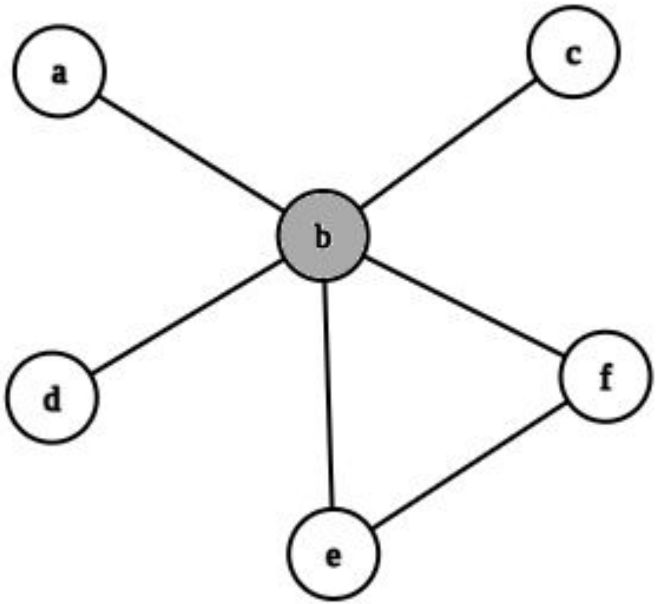


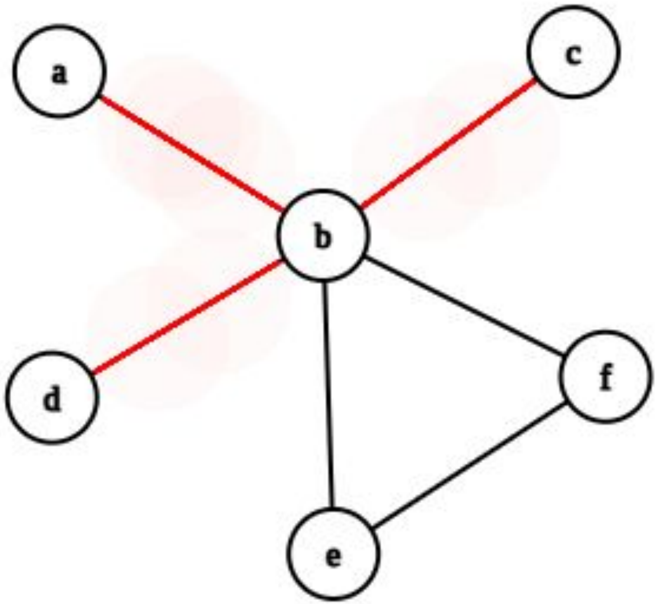


Exemplo 2



Exemplo 2





Algoritmo trivial

Algoritmo trivial - Articulações

1. Contar o número de componentes conexas iniciais
2. Para cada vértice ***v***
   a. Remover ***v*** do grafo
   b. Contar o número de componentes conexas
      i. Se número de componentes aumentou, então ***v*** é uma articulação
   c. Adicionar ***v*** novamente ao grafo

Algoritmo trivial - Articulações

1. Contar o número de componentes conexas iniciais // O(V+E)
2. Para cada vértice ***v*** // O(V)
    a. Remover ***v*** do grafo
    b. Contar o número de componentes conexas // O(V+E)
        i. Se número de componentes aumentou, então ***v*** é uma articulação
    c. Adicionar ***v*** novamente ao grafo

Complexidade final: $O(V^2+VE)$

Algoritmo trivial - Pontes

1. Contar o número de componentes conexas iniciais
2. Para cada aresta ***e***
    a. Remover ***e*** do grafo
    b. Contar o número de componentes conexas
        i. Se número de componentes aumentou, então ***e*** é uma ponte
    c. Adicionar ***e*** novamente ao grafo

Algoritmo trivial - Pontes

1. Contar o número de componentes conexas iniciais // O(V+E)
2. Para cada aresta **e** // O(E)
    a. Remover **e** do grafo
    b. Contar o número de componentes conexas // O(V+E)
        i. Se número de componentes aumentou, então **e** é uma ponte
    c. Adicionar **e** novamente ao grafo

Complexidade final: $O(VE+E^2)$

Algoritmo de Tarjan

Algoritmo de Tarjan

- Calcular para cada vértice durante a DFS:
  - *discovery(u)* - Valor da iteração da DFS da primeira visita ao vértice ***u***;
  - *low(u)* - Menor valor de *discovery* alcançável na *spanning subtree* da DFS (sem considerar back edges para o vértice pai de ***u***);
- Ao visitar um vértice ***u*** pela primeira vez, inicializaremos os valores de *discovery(u)* e *low(u)* com o valor da iteração atual;
- Após visitar cada vértice ***v*** adjacente não visitado, iremos atualizar o valor de *low(u)* com o menor valor entre *low(u)* e *low(v)*
- Caso um vértice ***v*** adjacente já tenha sido visitado e não seja o pai de ***u***, ou seja, ***(u, v)*** é uma back-edge, iremos atualizar o valor de *low(u)* com o menor valor entre *low(u)* e *low(v)* (mas não iremos visitar de novo o vértice ***v***!)

Algoritmo de Tarjan - Articulações

Dados dois vértices adjacentes, ***u*** e ***v***, se *low(v) >= discovery(u)*, então ***u*** é uma articulação;

- Se *low(v) < discovery(u)*, então é possível chegar em *u* a partir de *v* por outro caminho!
- **Caso *u* seja a raíz da DFS, ele só será articulação se possuir mais de um filho na *spanning subtree* da DFS!**

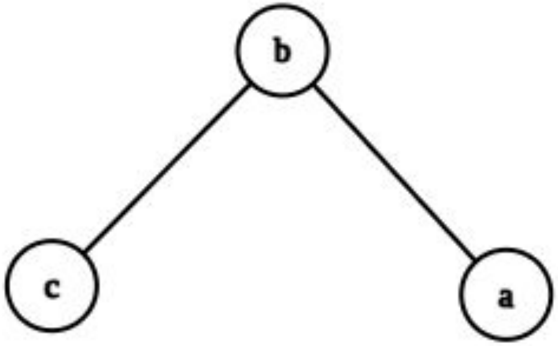
Algoritmo de Tarjan - Pontes

Dada uma aresta, ***(u, v)***, se *low(v) > discovery(u)*, então ***(u, v)*** é uma ponte

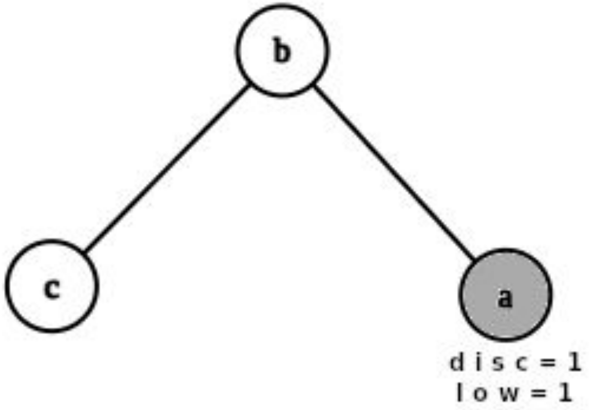
- Se *low(v) <= discovery(u)*, então é possível chegar em *u* a partir de *v* por outro caminho!

Execução

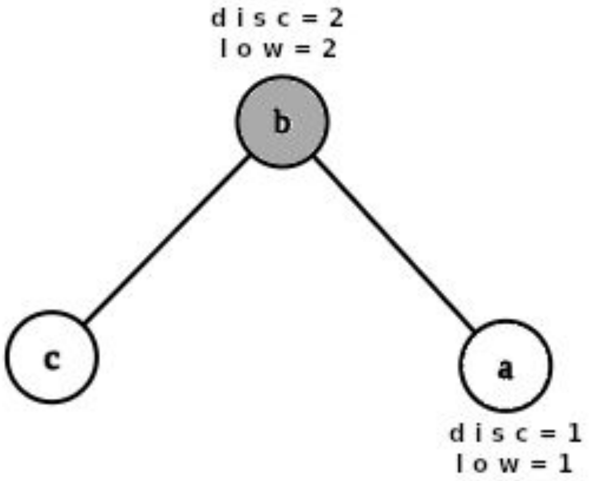
Execução - Exemplo 1



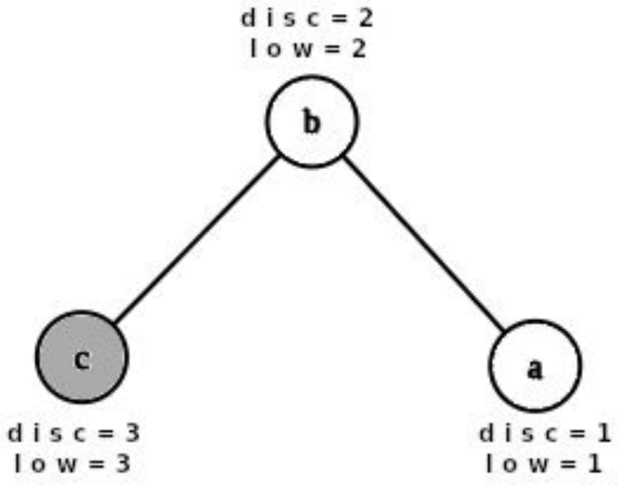
Execução - Exemplo 1



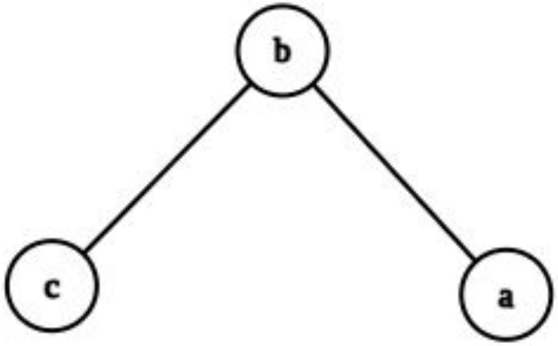
Execução - Exemplo 1



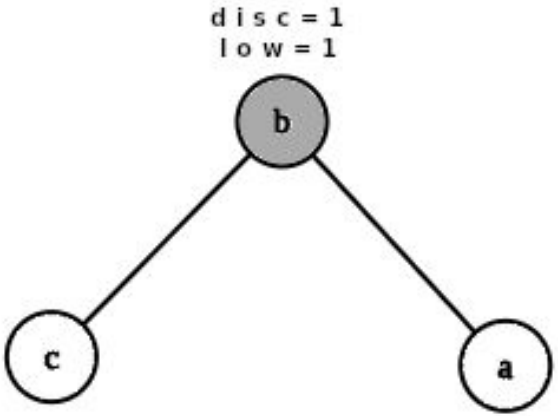
Execução - Exemplo 1



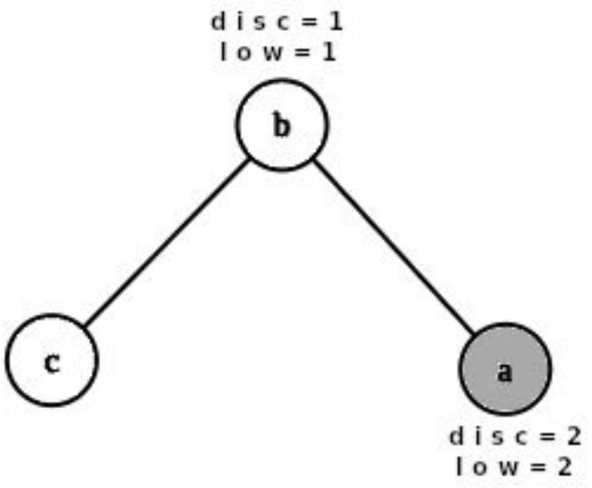
Execução - Exemplo 1



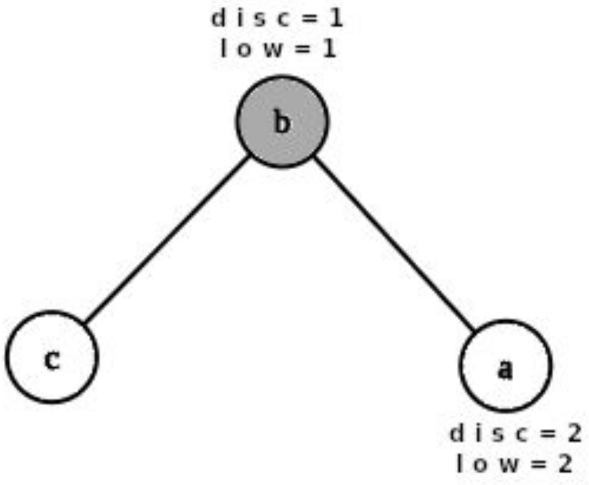
Execução - Exemplo 1



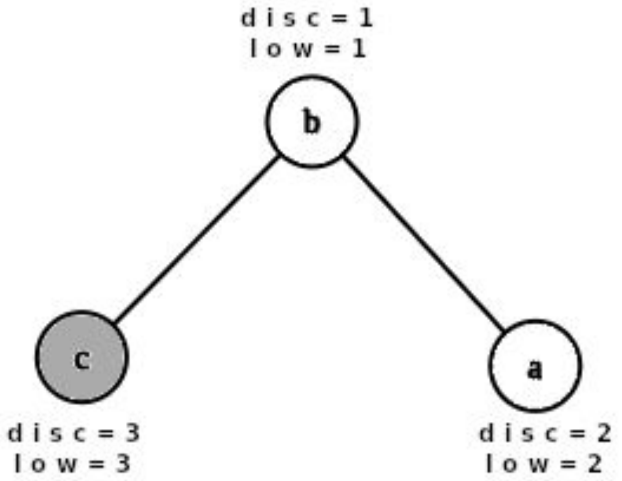
Execução - Exemplo 1



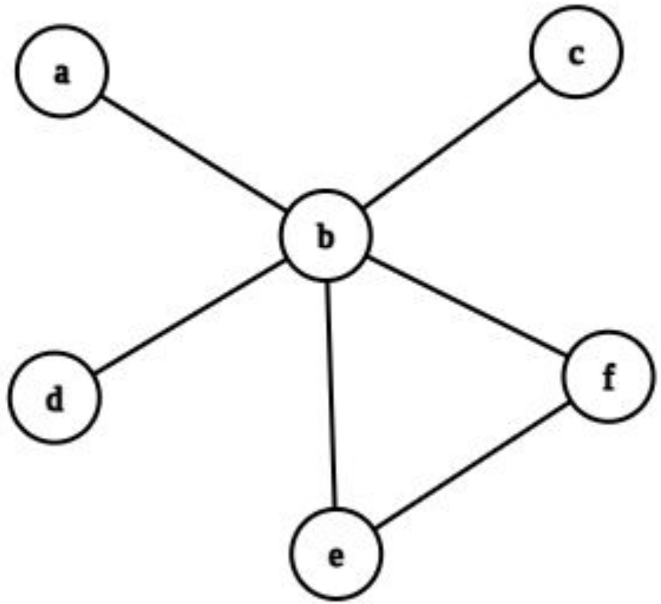
Execução - Exemplo 1



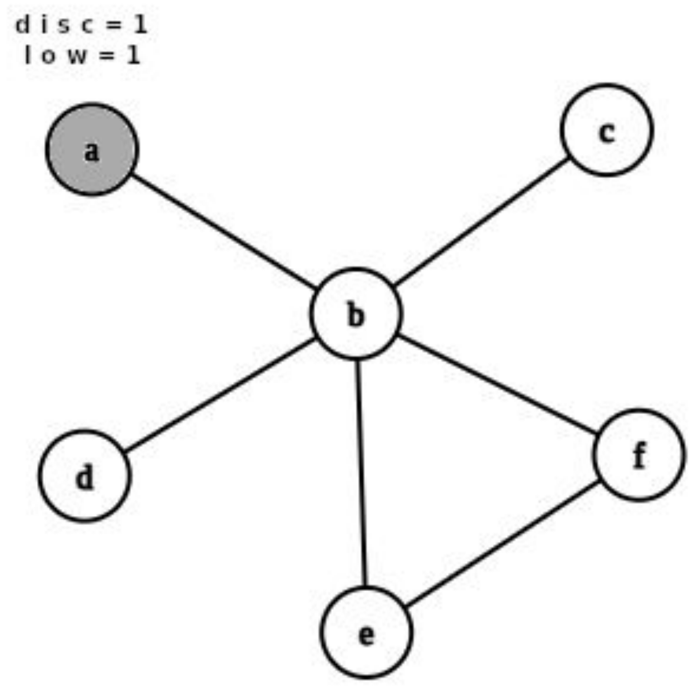
Execução - Exemplo 1



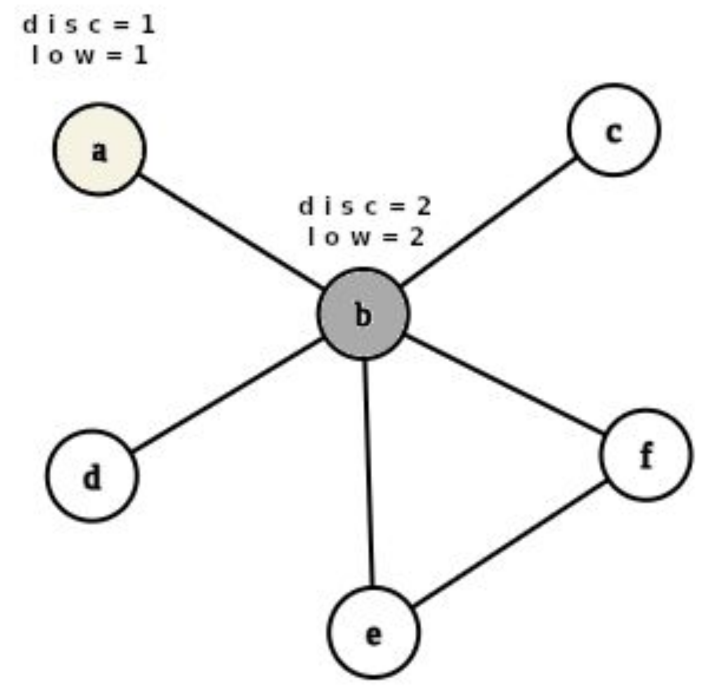
Execução - Exemplo 2



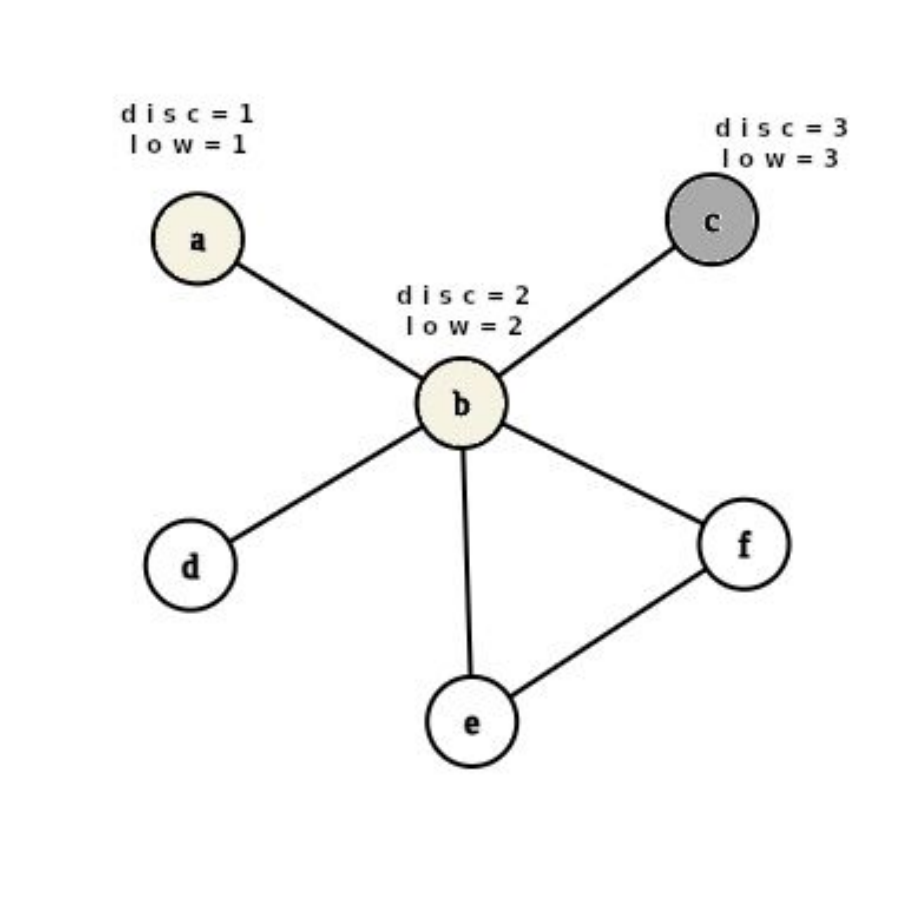
Execução - Exemplo 2



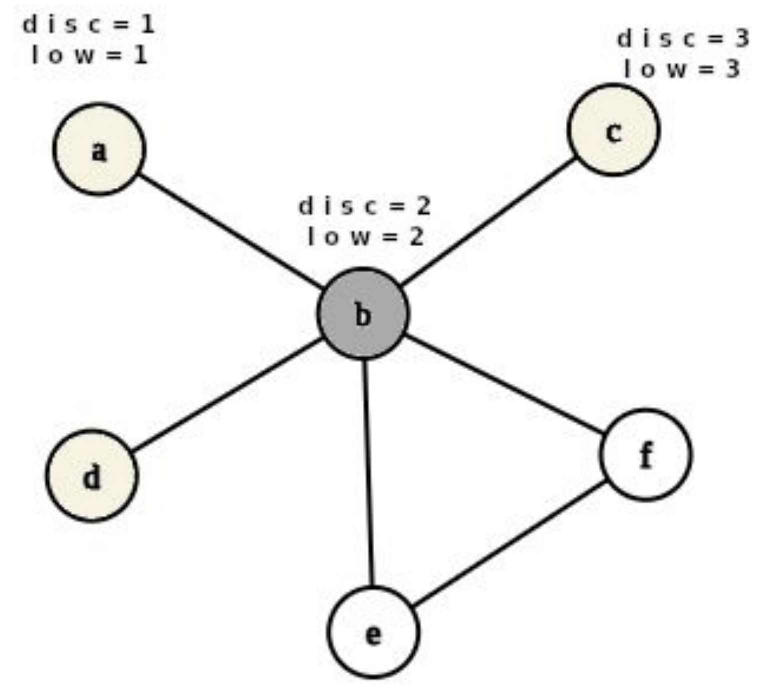
Execução - Exemplo 2



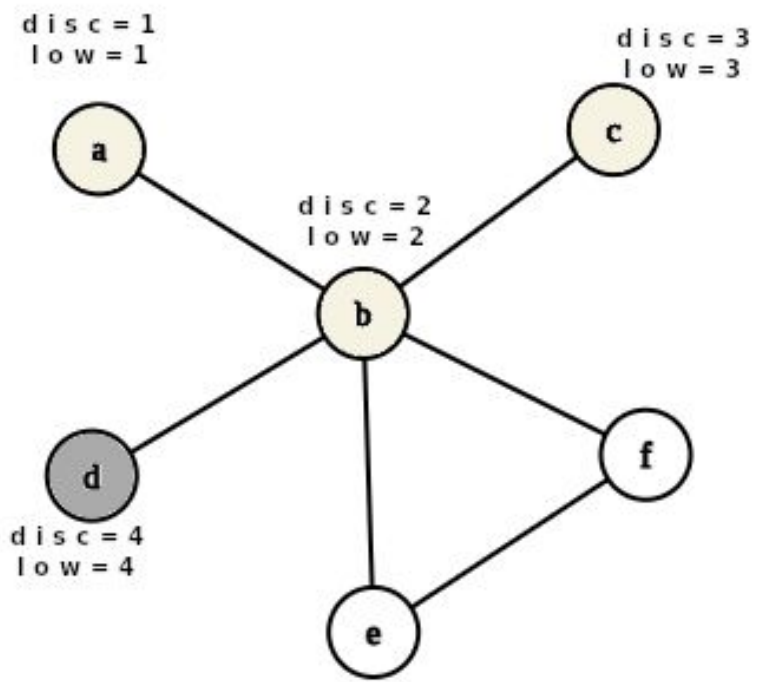
Execução - Exemplo 2



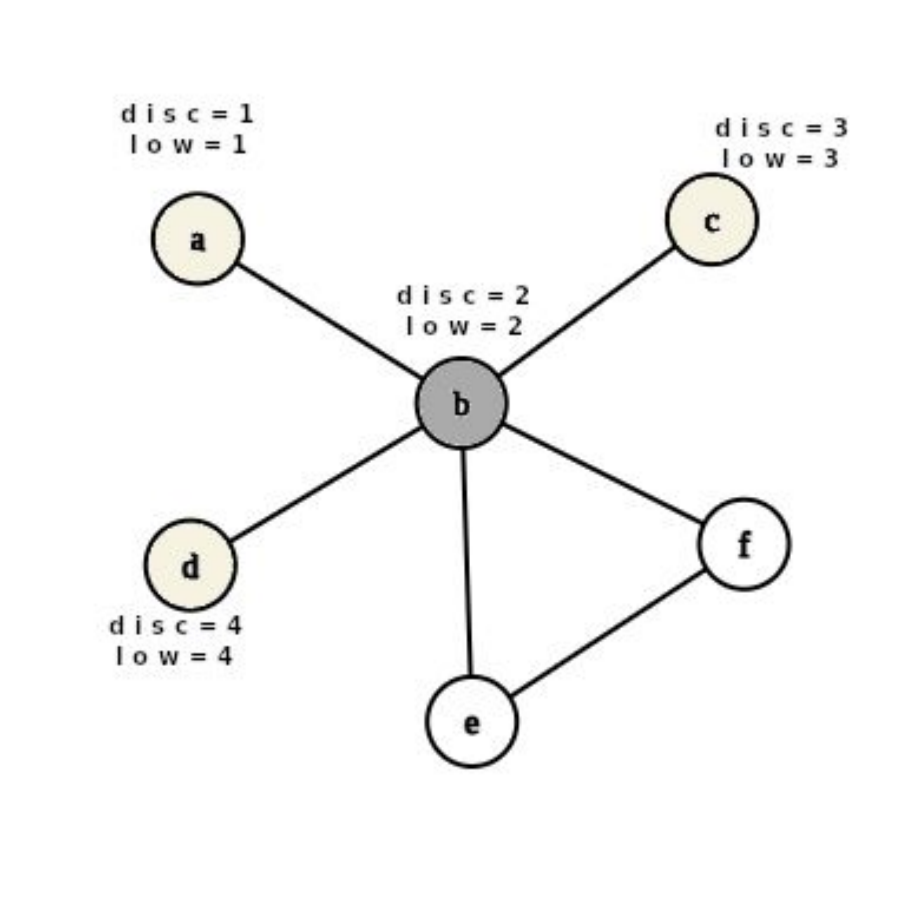
Execução - Exemplo 2



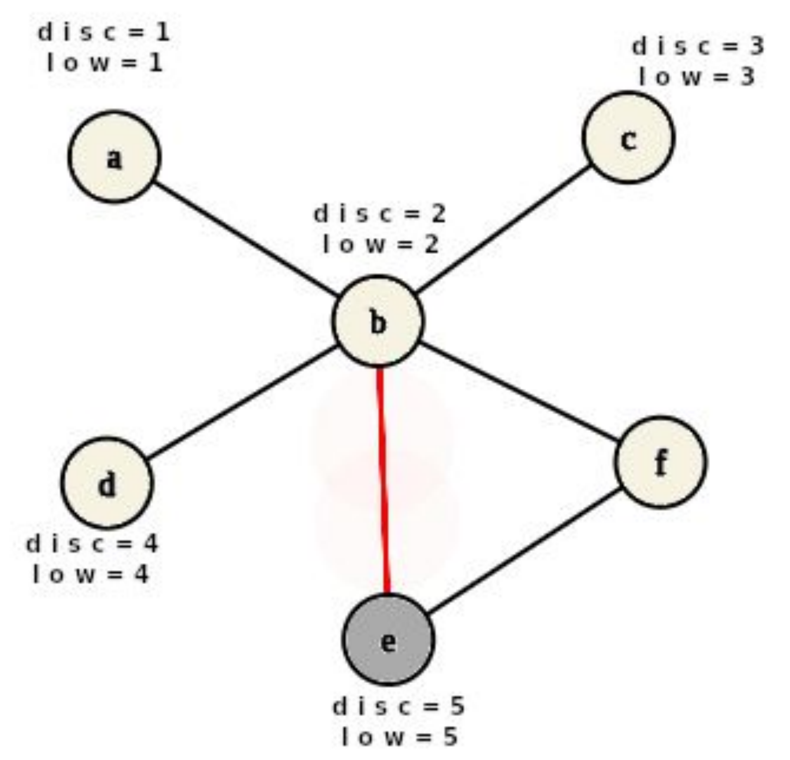
Execução - Exemplo 2



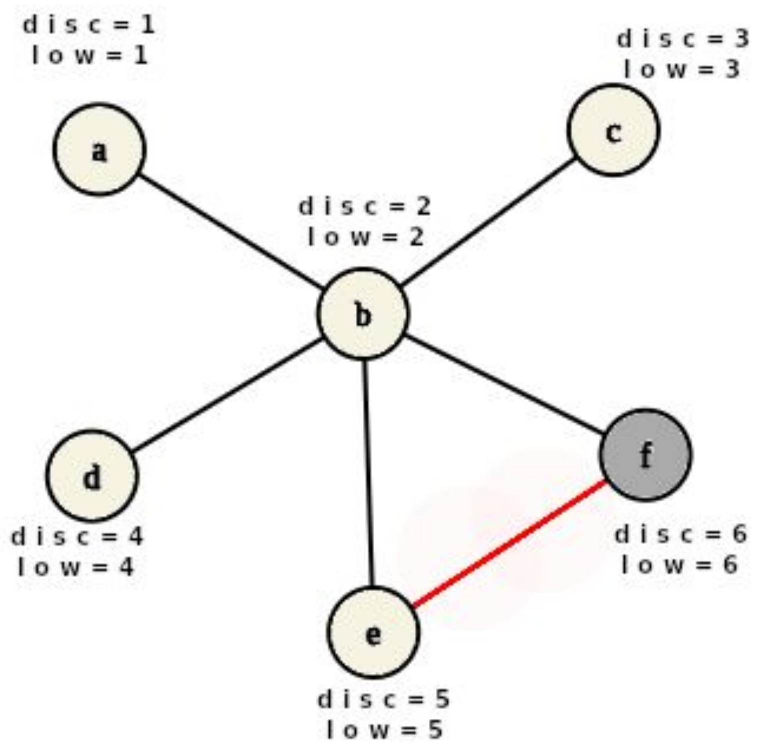
Execução - Exemplo 2



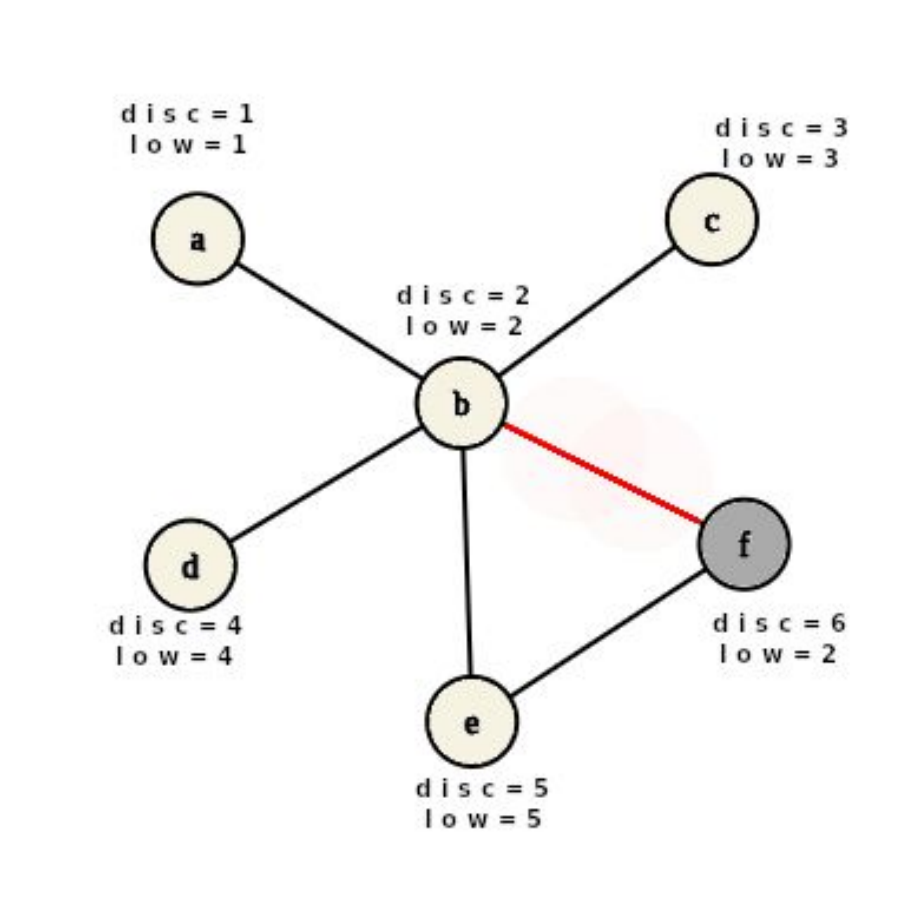
Execução - Exemplo 2



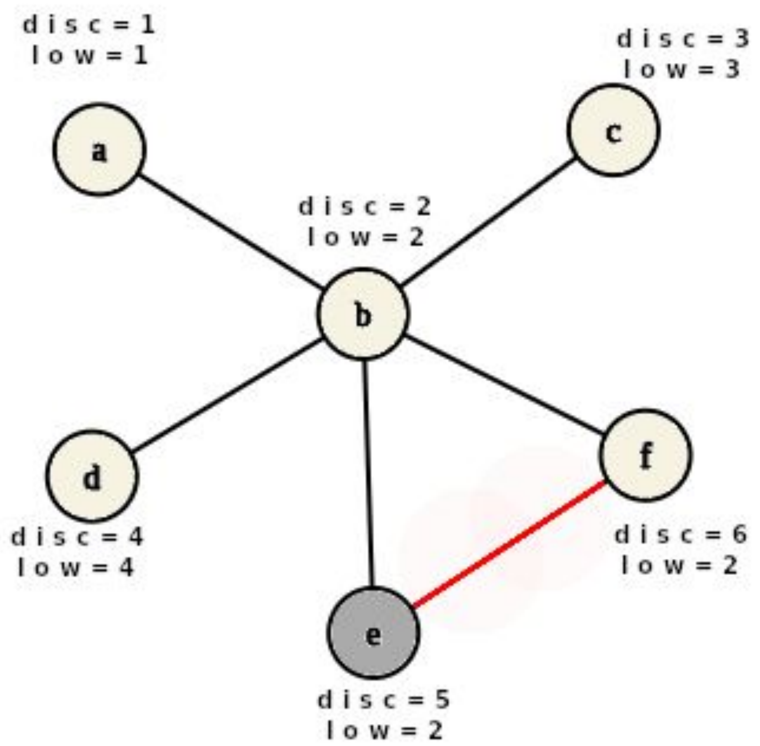
Execução - Exemplo 2



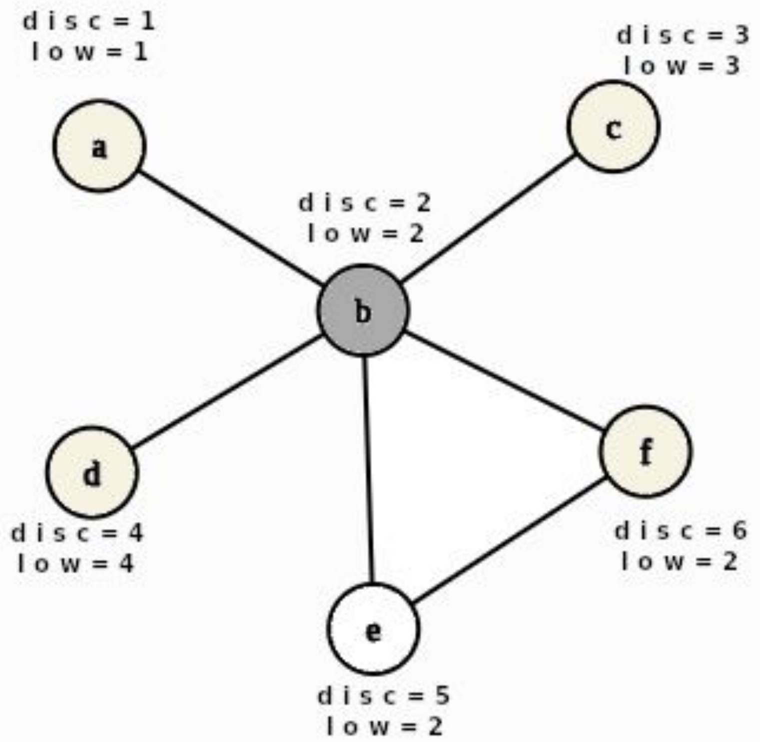
Execução - Exemplo 2



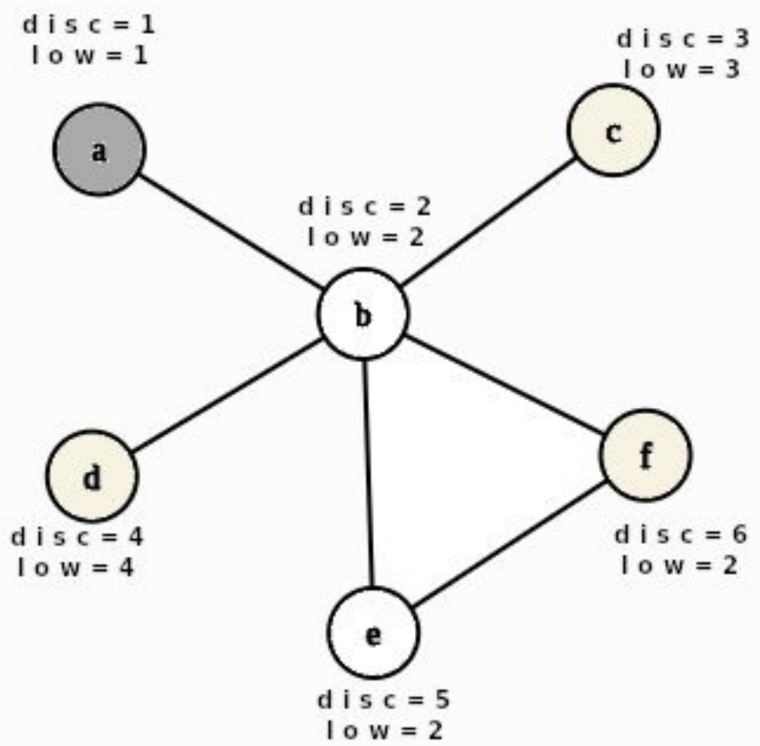
Execução - Exemplo 2



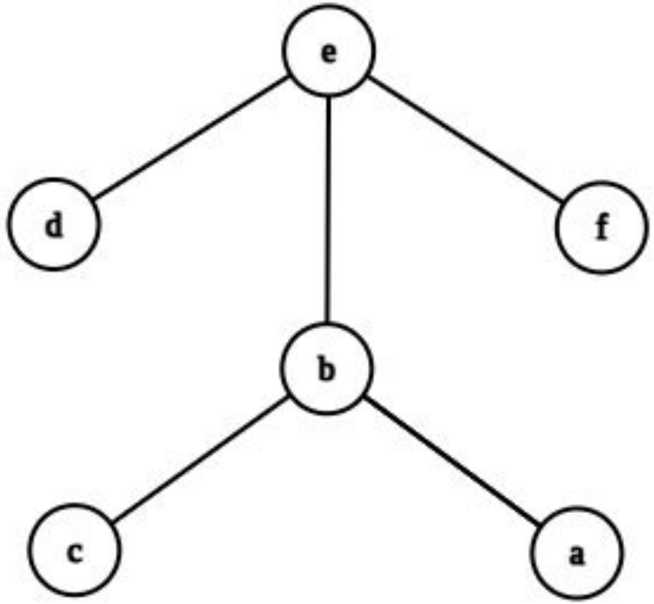
Execução - Exemplo 2



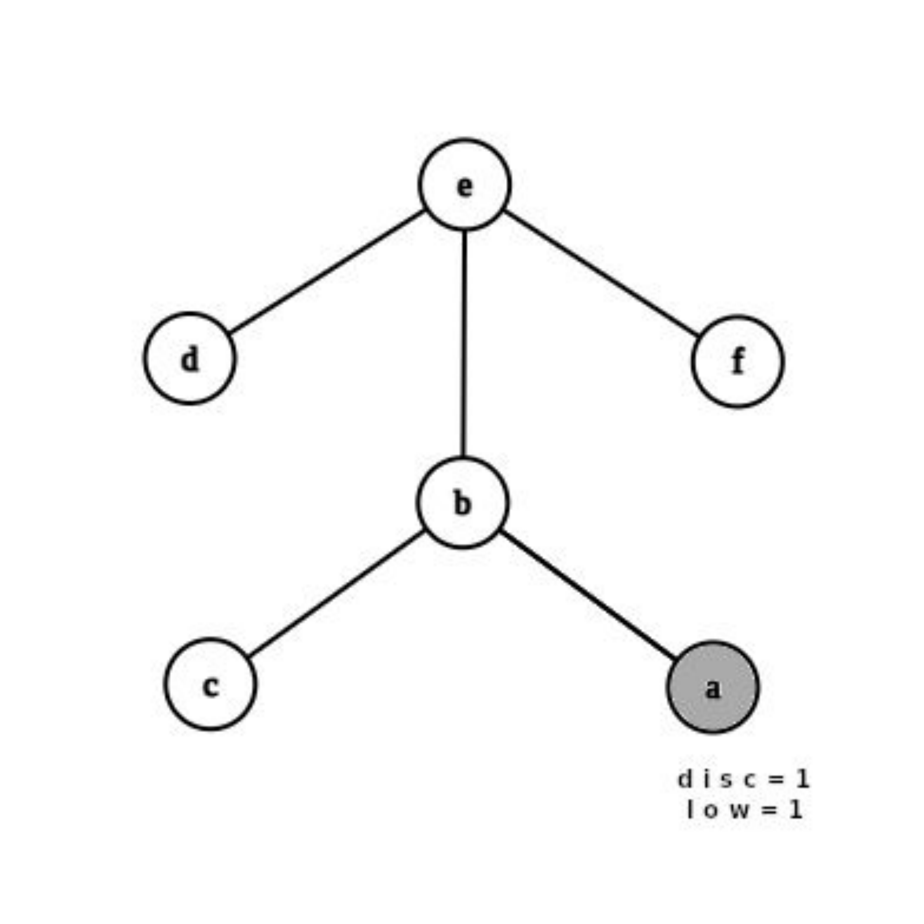
Execução - Exemplo 2



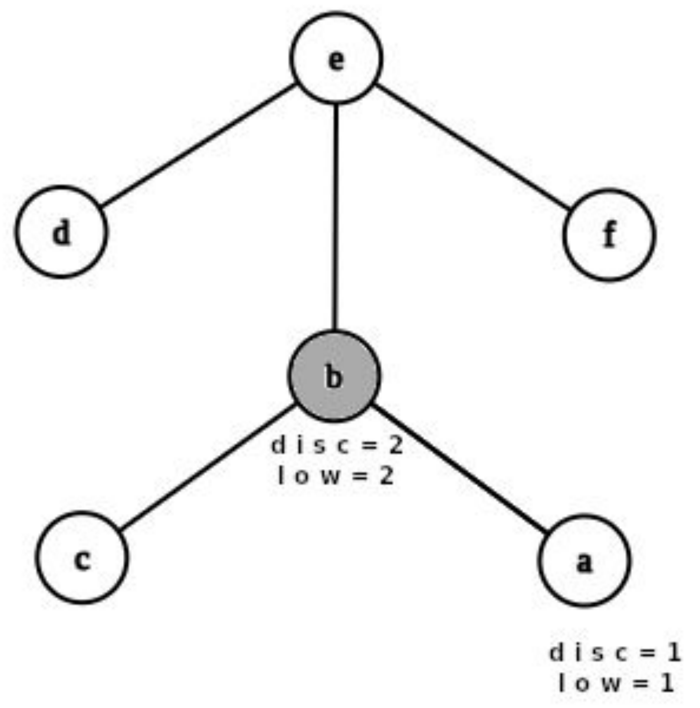
Execução - Exemplo 3



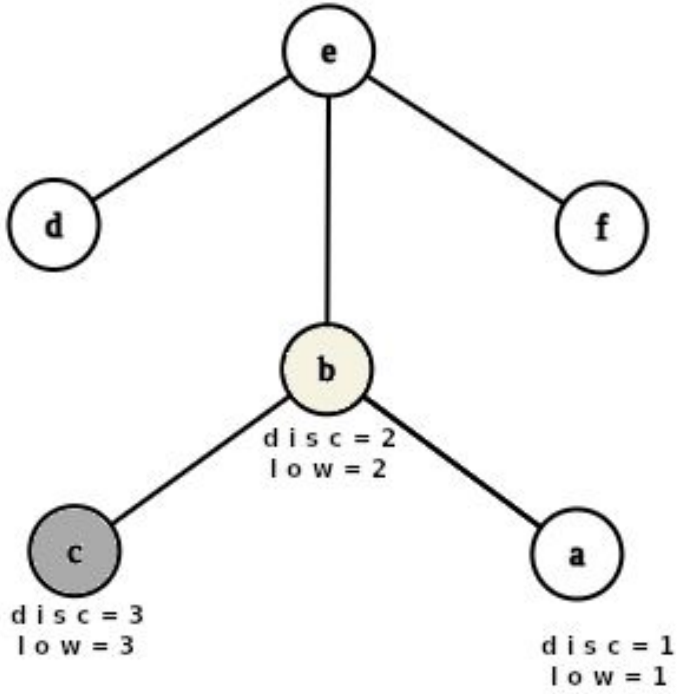
Execução - Exemplo 3



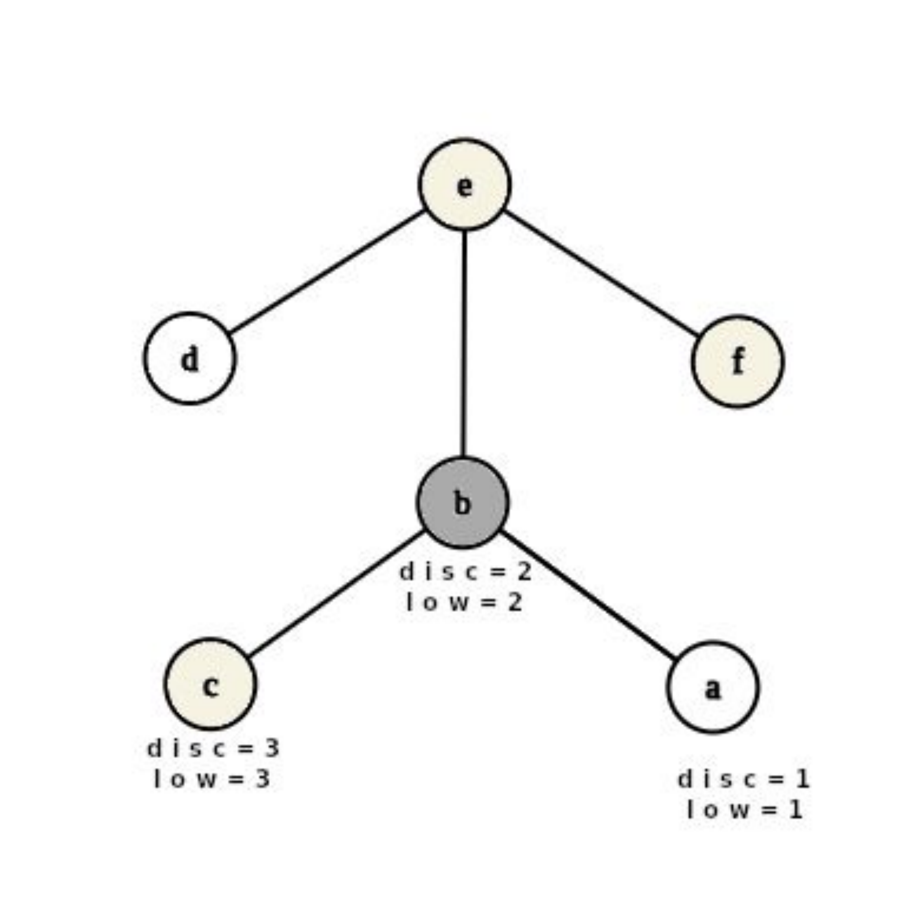
Execução - Exemplo 3



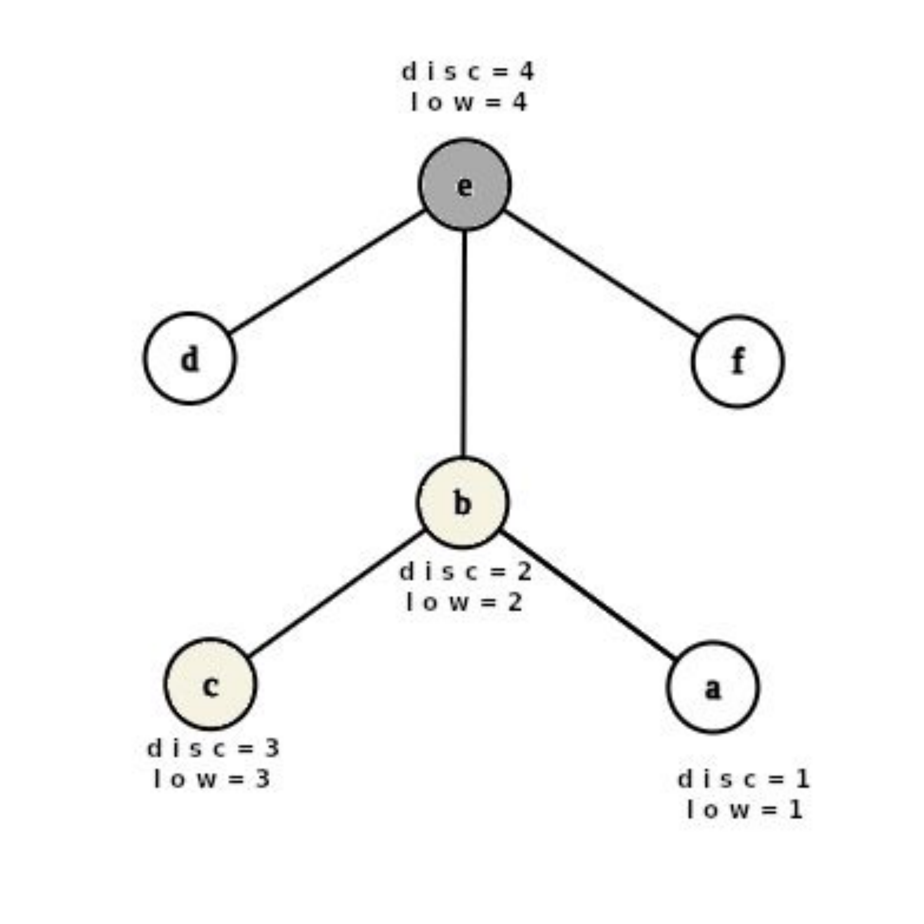
Execução - Exemplo 3



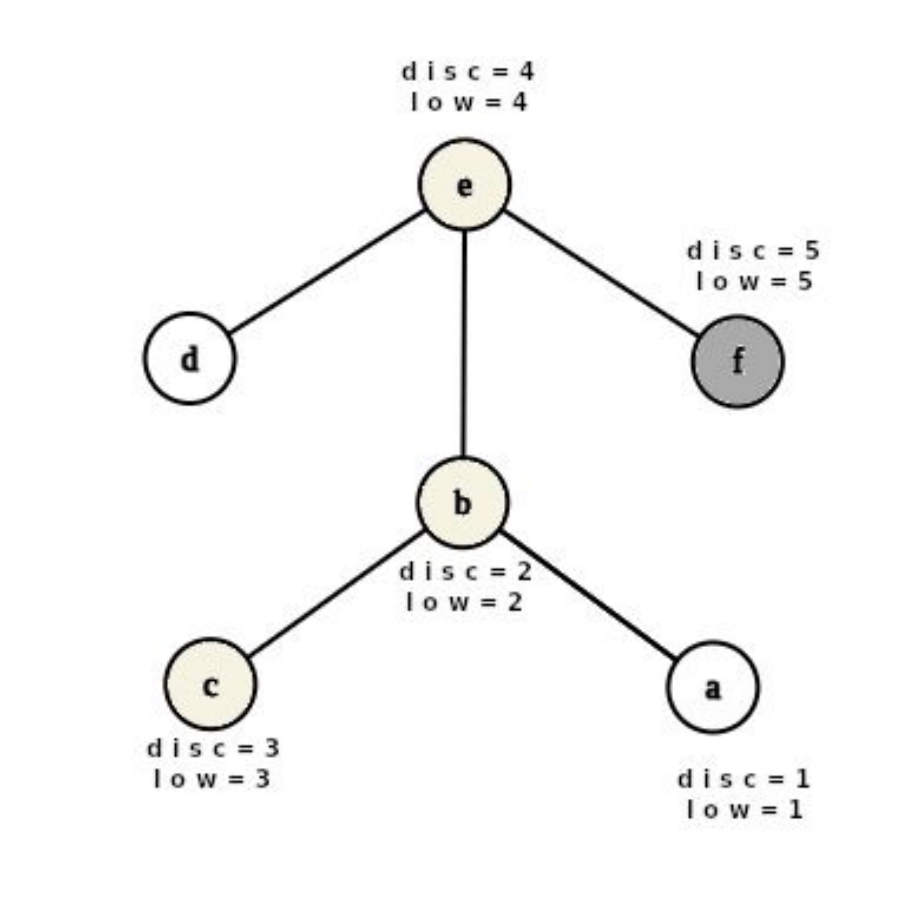
Execução - Exemplo 3



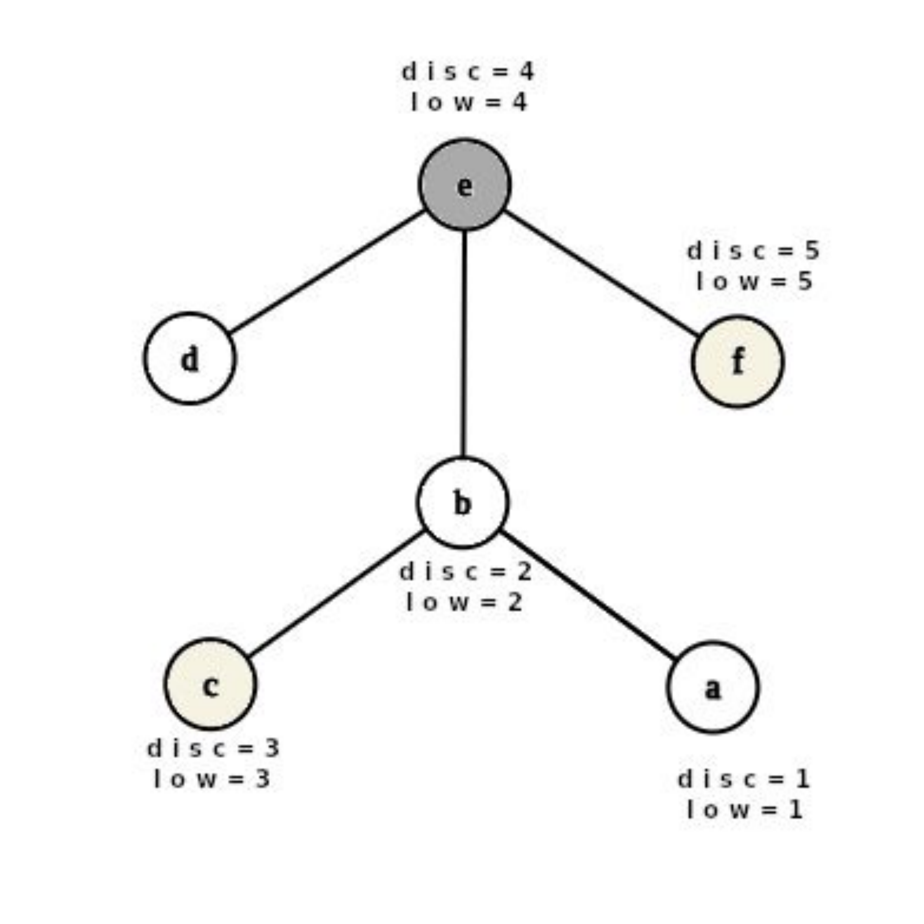
Execução - Exemplo 3



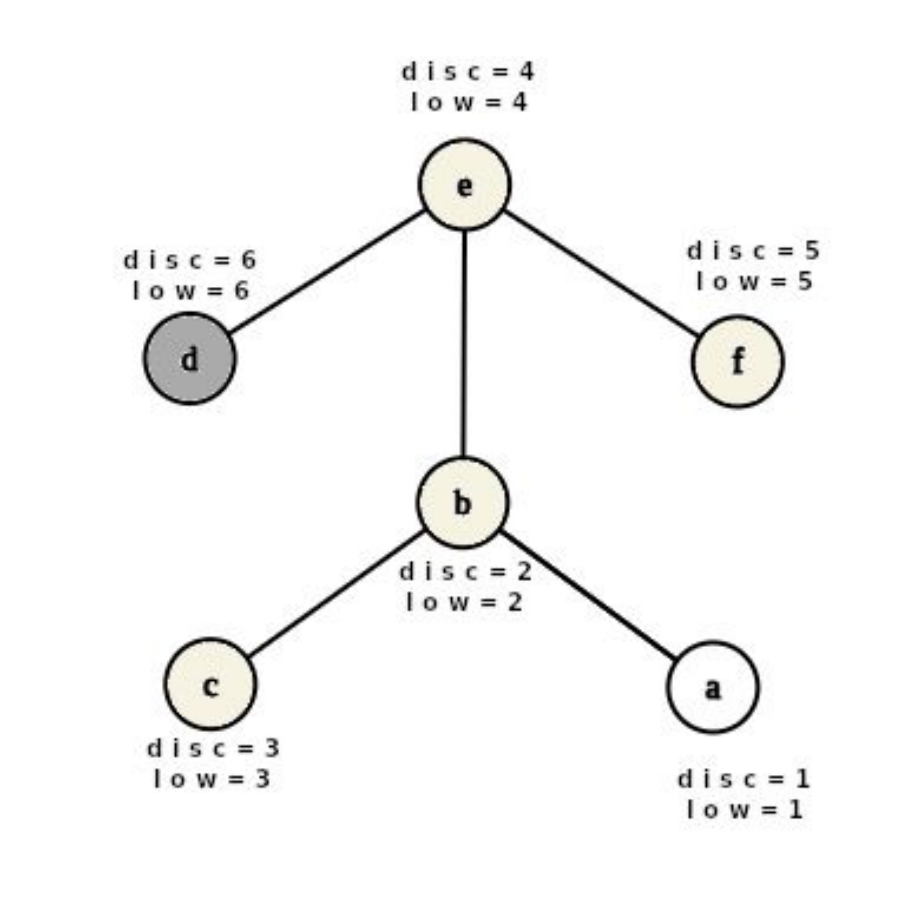
Execução - Exemplo 3



Execução - Exemplo 3



Execução - Exemplo 3



Implementação

```
void tarjan(int u, int root){
    // incrementa iteração da DFS e inicializa discovery e low
    discovery[u] = low[u] = dfsIteration++;
    int children = 0;
    for (int v: adj[u]){
        // Se v não foi descoberto/visitado
        if(discovery[v] == -1){
            // Marca u como pai de v e visita v
            parent[v] = u;
            children++;
            tarjan(v, root);

            // Verifica se u é uma articulação
            if (low[v] >= discovery[u] && u != root){
                is_articulation[u] = true;
            }

            // Verifica se (u, v) é uma ponte
            if (low[v] > discovery[u]){
                bridges.emplace_back(min(u, v), max(u, v));
            }

            // Atualiza low(u)
            low[u] = min(low[u], low[v]);
        }
        // Se (u, v) for uma back-edge mas v não é pai de u, atualiza low(u)
        else if (v != parent[u]){
            low[u] = min(low[u], discovery[v]);
        }
    }

    // Se u for a raíz da DFS e tiver mais de um filho na spanning tree, ele é uma articulação
    if (u == root && children > 1){
        is_articulation[u] = true;
    }
}
```

Implementação

Declarações:

```
// Declarações
int dfsIteration = 0, MAXN=12345;
vector<pair<int, int>> bridges;
vector<bool> is_articulation;
vector<vector<int>> adj(MAXN, vector<int>());
vector<int> low(MAXN, -1), discovery(MAXN, -1), parent(MAXN, 0);
```

Loop principal:

```
for (int i = 0; i <= num_v; i++){
    if (discovery[i] == -1){
        tarjan(i, i);
    }
}
```

Questões

Questões

1. https://www.spoj.com/problems/SUBMERGE/
2. https://br.spoj.com/problems/MANUT/
3. https://www.spoj.com/problems/EC_P/
4. https://judge.beecrowd.com/pt/problems/view/1790
5. https://codeforces.com/contest/178/problem/B2
6. https://codeforces.com/contest/118/problem/E