CES-11

Algoritmos e Estruturas de Dados

Carlos Alberto Alonso Sanches
Juliana de Melo Bezerra

CES-11

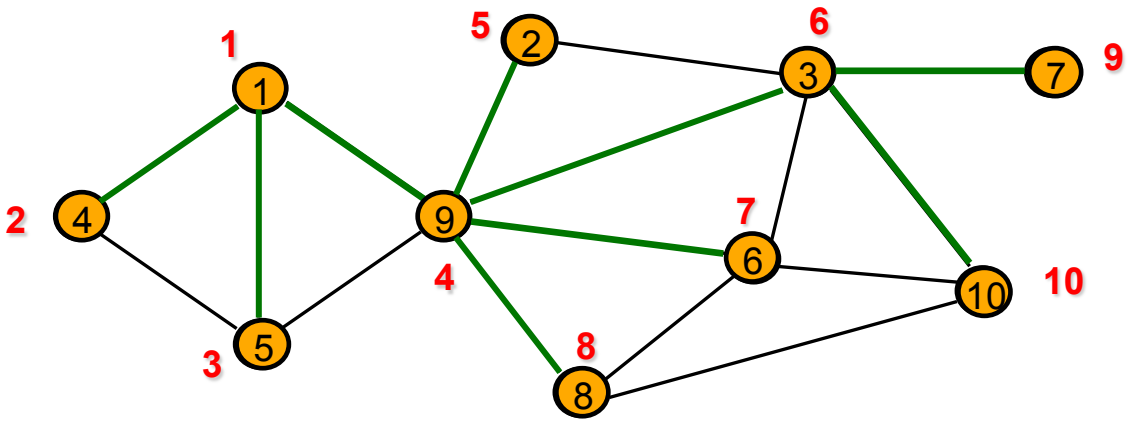
- Grafos
  - Conceitos gerais e representações
- Algoritmos em grafos
  - Exploração sistemática em largura
  - Caminhos mais curtos
  - Exploração sistemática em profundidade
  - Teste de aciclicidade
  - Ordenação topológica
  - Componentes fortemente conexos
  - Vértices e arestas de corte
  - Árvore geradora de custo mínimo

Exploração sistemática de um grafo

- Explorar um grafo é percorrê-lo completamente, visitando todos os vértices e as arestas.
- A ordem dessas visitas depende:
  - do vértice onde a exploração começa;
  - da ordem de armazenamento dos vértices e das arestas na estrutura de dados.
- Basicamente, há dois tipos de explorações: em largura e em profundidade.

Em largura (*breadth-first search*)

- Tática: enquanto for possível, examinar todos os vértices à mesma distância do vértice corrente; quando não for mais possível, aprofundar.
- Exemplo (supomos armazenamento em ordem crescente):



- Uma aplicação: a exploração em largura permite encontrar, por exemplo, as distâncias e os menores caminhos entre os vértices.

Exploração em largura

```
BFS(s) {
    desmarcar todos os vértices;
    int ce = 0;
    fila q;
    inicFila(q);
    marcar s;
    expl[s] = ++ce;
    enqueue(q,s);
    while (!isEmpty(q)) {
          curr = dequeue(q);
          // explorando curr
          for <curr,u> ∈ E {
              if u está desmarcado {
                 marcar u;
                 expl[u] = ++ce;
                 enqueue(q,u);
              }
          }
    }
}
```

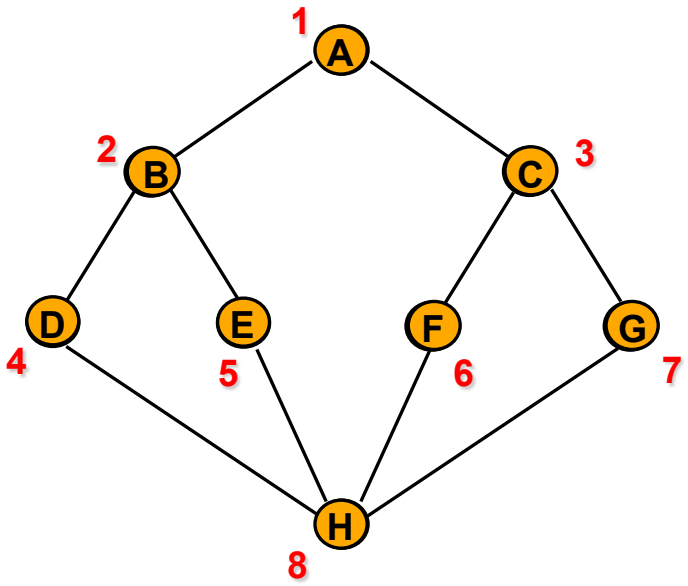
Código aplicável a grafos conexos e não orientados

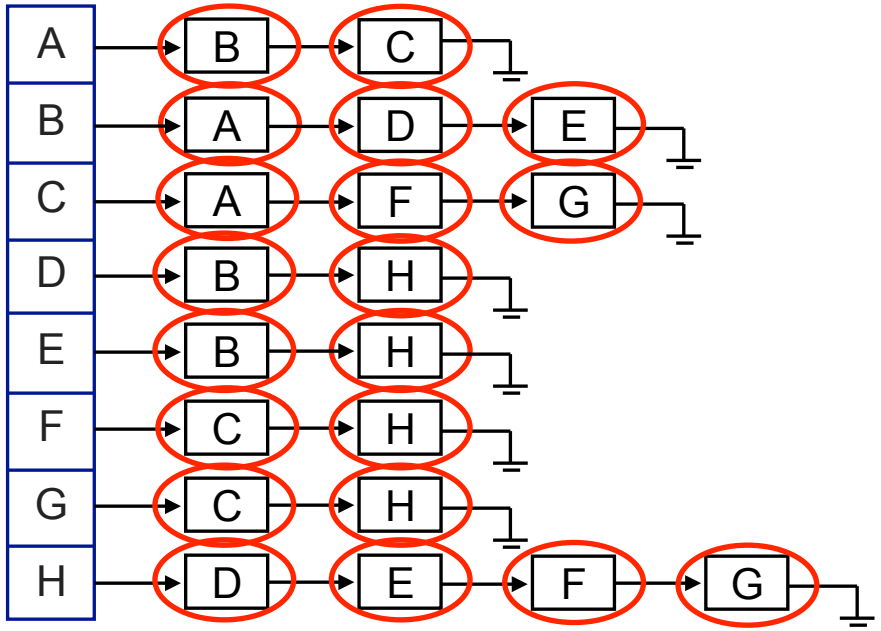
Cada vértice recebe um número de exploração (equivalente a marcá-lo)

Exploração em largura

- Durante a execução deste algoritmo, cada vértice pode ficar em três estados:
  - desmarcado (portanto, fora da fila): ainda não foi atingido;
  - marcado e na fila: atingido, mas não completamente explorado;
  - marcado e fora da fila: já explorado.
- Cada vértice entra na fila uma única vez, e cada aresta é visitada duas vezes. Portanto, sua complexidade de tempo é $O(n+m)$.

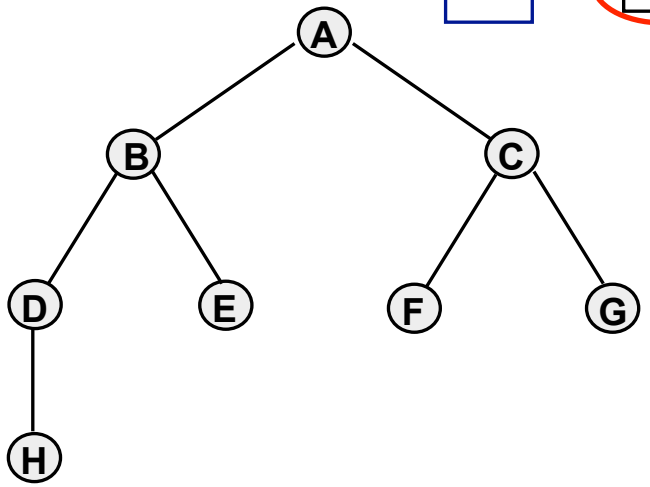
Exemplo





não visitado

visitado

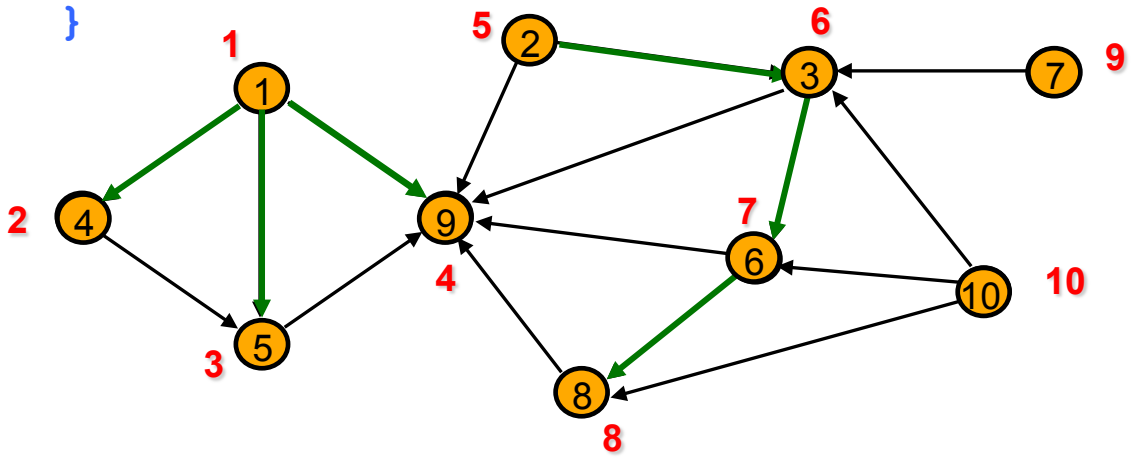


Árvore de exploração em largura

Exploração em largura (digrafos)

```
int ce;
fila q;

TravessiaBFS(s) {
    desmarcar todos os vértices;
    ce = 0;
    inicFila(q);
    BFS(s);
    for v ∈ V {
        if v está desmarcado
            BFS(v);
    }
}
```

Código adicional




```
BFS(v) {
    marcar v;
    expl[v] = ++ce;
    enqueue(q,v);
    while (!isEmpty(q)) {
        curr = dequeue(q);
        // explorando curr
        for <curr,u> ∈ E {
            if u está desmarcado {
                marcar u;
                expl[u] = ++ce;
                enqueue(q,u);
            }
        }
    }
}
```

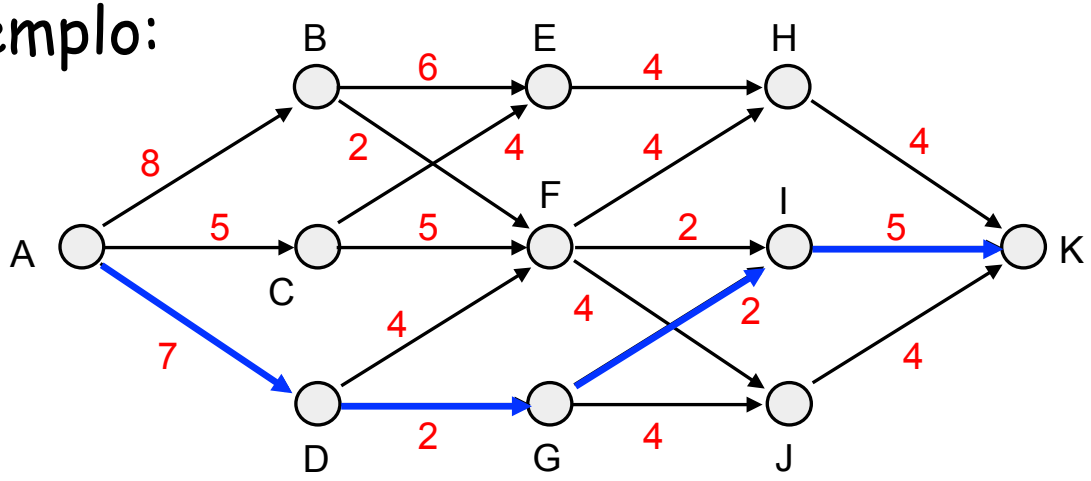
Solução também é válida para grafos desconexos

CES-11

- Grafos
  - Conceitos gerais e representações
- Algoritmos em grafos
  - Exploração sistemática em largura
  - Caminhos mais curtos
  - Exploração sistemática em profundidade
  - Teste de aciclicidade
  - Ordenação topológica
  - Componentes fortemente conexos
  - Vértices e arestas de corte
  - Árvore geradora de custo mínimo

Caminhos mais curtos

- Um digrafo G=(V,E), onde V = {v_1, v_2, ..., v_n}, é chamado de <u>ponderado</u> se cada arco (v_i, v_j) ∈ E tem um custo c_{ij}.
- Distância entre dois vértices é a somatória dos custos dos arcos de um caminho que os une.
- Um problema clássico em grafos é encontrar o caminho mais curto ou a distância mínima entre dois vértices.
- Exemplo:

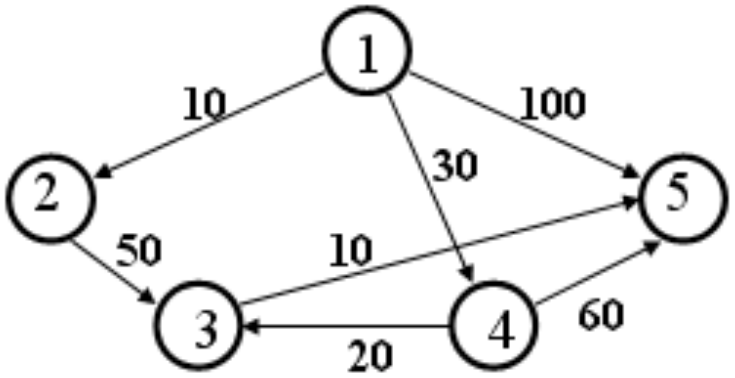


Distância mínima entre A e K:
7+2+2+5 = 16

Caminhos mais curtos

- O objetivo é encontrar, a partir de um vértice de origem, os caminhos mais curtos até os demais vértices do grafo.
- Caminhos de 1 a 3:
  - 1-2-3 (custo 60)
  - 1-4-3 (custo 50)
- Caminhos de 1 a 5:
  - 1-5 (custo 100)
  - 1-2-3-5 (custo 70)
  - 1-4-5 (custo 90)
  - 1-4-3-5 (custo 60)

- c_{ij} é o custo do vértice i até o vértice j
- Se não houver o arco (i,j), então $c_{ij} = +\infty$
- O algoritmo que apresentaremos também é válido para grafos não orientados

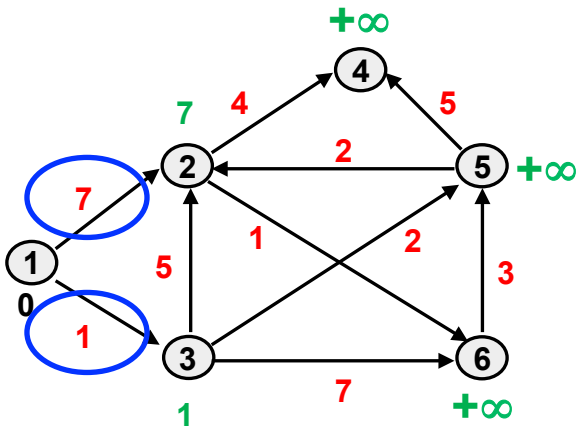


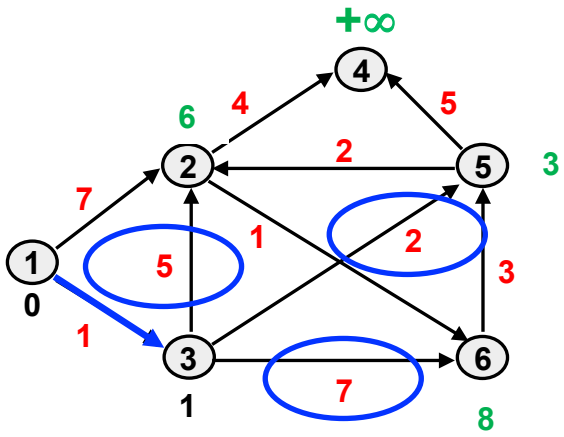
Grafos com arestas de mesmo peso

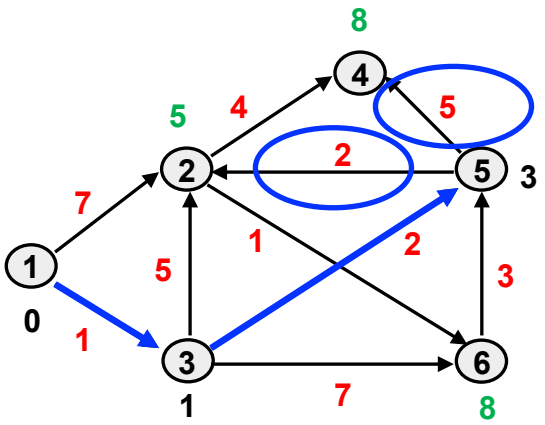
- Quando todas as arestas **têm pesos iguais**, basta uma simples alteração na exploração em largura.
- Afinal, os vértices vão sendo enfileirados seguindo a ordem da proximidade: portanto, basta incrementar a distância do vértice antecessor.
- O algoritmo de Dijkstra generaliza essa ideia.

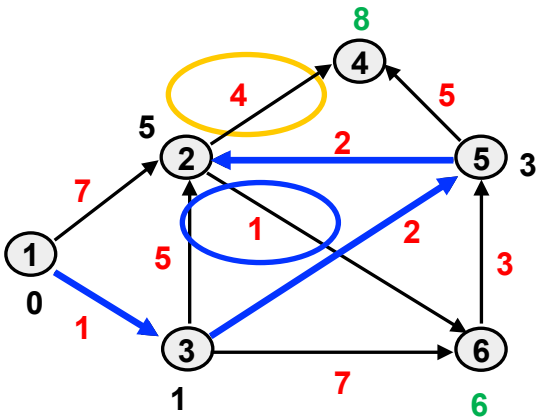
```
BFSMinCam(r) {
    queue q; inicFila(q);
    int ce = 0;
    for v ∈ V - {r} {
        d[v] = +∞;
        expl[v] = 0;
    }
    expl[r] = ++ce;
    d[r] = 0;
    enqueue(q,r);
    while (!isEmpty(q)) {
        u = dequeue(q);
        for <u,v> ∈ E {
            if (expl[v] == 0) {
                expl[v] = ++ce;
                d[v] = d[u] + 1;
                enqueue(q,v);
            }
        }
    }
}
```

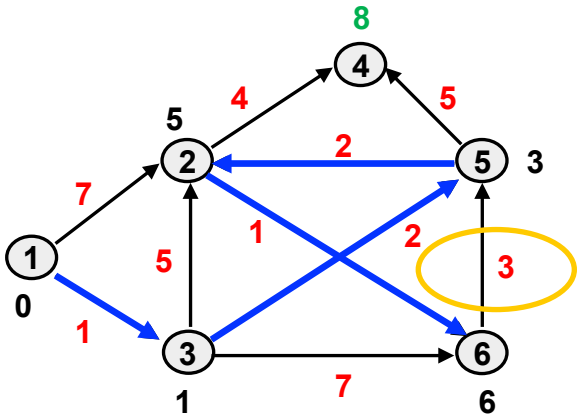
Exemplo do algoritmo de Dijkstra

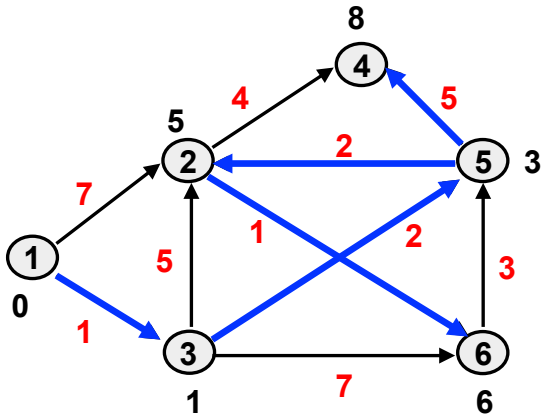










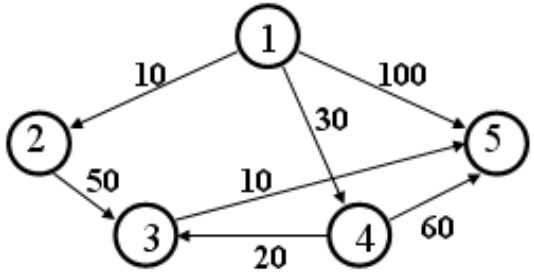


Algoritmo de Dijkstra (1959)

```
Dijkstra(u) {
    for v ∈ V - {u}
        d[v] = +∞;
    d[u] = 0;
    S ← V;      \\ S contém vértices sem distância definida
    while S ≠ ∅ {
        selecionar j tal que d[j] == min_{i∈S}{d[i]};
        S ← S - {j};   \\ j passa a ter distância definida
        for <j,w> ∈ E, onde w ∈ S
            if (d[w] > d[j] + c_jw) {
                d[w] = d[j] + c_jw;
                pred[w] = j;
            }
}
```

Predecessor: permite encontrar os vértices presentes nesse caminho

Passo a passo



Calcularemos as distâncias a partir do vértice 1

j	S	d[1]	d[2]	d[3]	d[4]	d[5]
-	1,2,3,4,5	0	+∞	+∞	+∞	+∞
1	2,3,4,5		0+10=10 (1-2)		0+30=30 (1-4)	0+100=100 (1-5)
2	3,4,5			10+50=60 (1-2-3)		
4	3,5			30+20=50 (1-4-3)		30+60=90 (1-4-5)
3	5					50+10=60 (1-4-3-5)
5	∅					

Complexidade de tempo

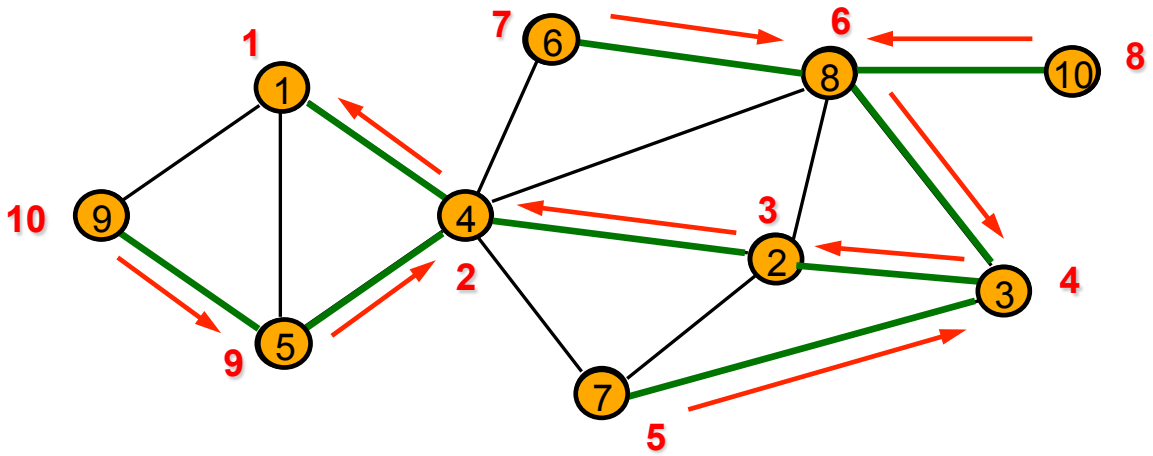
- *Grosso modo*, o algoritmo de Dijkstra gasta tempo $O(n^2+m) = O(n^2)$.
- No entanto é possível melhorar essa complexidade se o conjunto S for implementado com um *heap* de mínimo.
- Nesse caso, o tempo passaria a ser $O((n+m).\log n)$:
  - O *heap* possuirá inicialmente n elementos.
  - No total, são realizadas n extrações de mínimo e m modificações de valor (será preciso manter um vetor auxiliar que armazena a posição corrente que cada vértice ocupa no *heap*).

CES-11

- Grafos
  - Conceitos gerais e representações
- Algoritmos em grafos
  - Exploração sistemática em largura
  - Caminhos mais curtos
  - Exploração sistemática em profundidade
  - Teste de aciclicidade
  - Ordenação topológica
  - Componentes fortemente conexos
  - Vértices e arestas de corte
  - Árvore geradora de custo mínimo

Em profundidade (*depth-first search*)

- Tática: enquanto for possível, aprofundar-se no grafo; quando não for mais possível, recuar um nível.
- Exemplo (supomos armazenamento em ordem crescente):



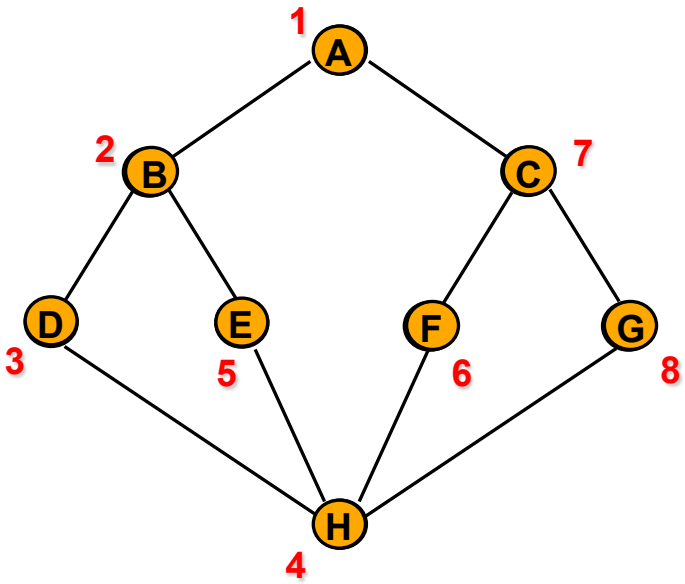
- A exploração em profundidade possibilita respostas a várias questões. Por exemplo: ordenação topológica; se o grafo é acíclico ou conexo; quais são os seus componentes conexos e os eventuais vértices e arestas de corte.

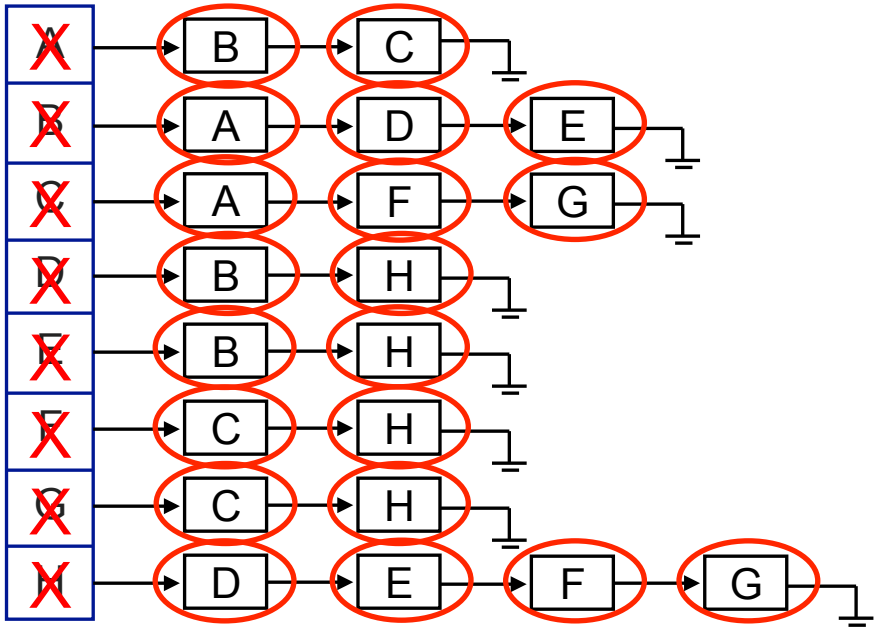
Versão recursiva

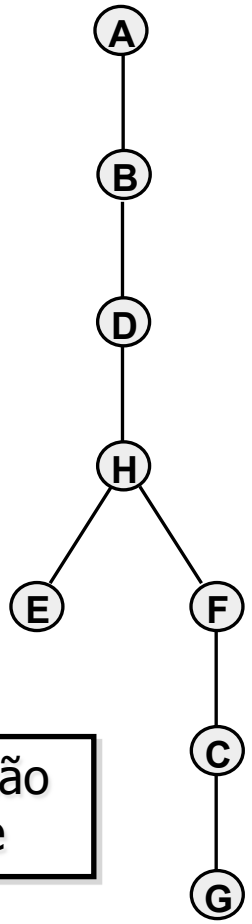
```
int ce = 0;
desmarcar todos os vértices;

DFS(s) {
     marcar s;
     expl[s] = ++ce;
     // explorando s
     for <s,v> ∈ E {
          if v está desmarcado
               DFS(v);
     }
}
```

- Analogamente à exploração em largura, a complexidade de tempo também é O(n+m).
- É possível escrever esse algoritmo em versão iterativa, com o uso de uma pilha explícita.

Exemplo





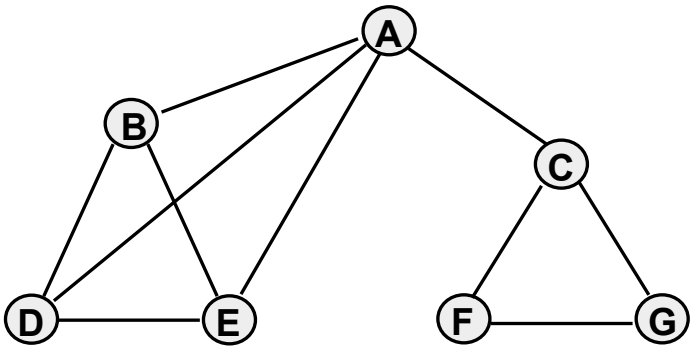


não visitado

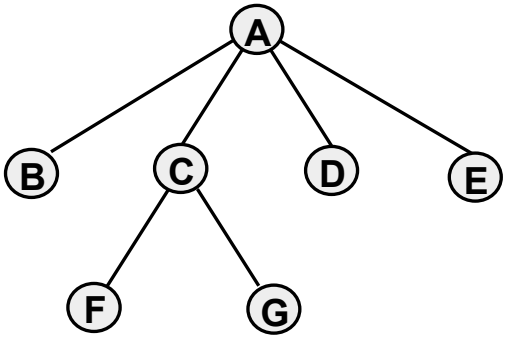
visitado

Árvore de exploração em profundidade

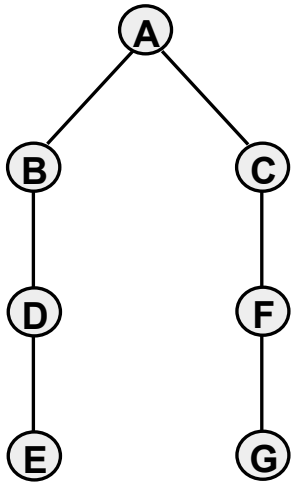
Comparação: largura x profundidade



Árvore de exploração em largura



Árvore de exploração em profundidade



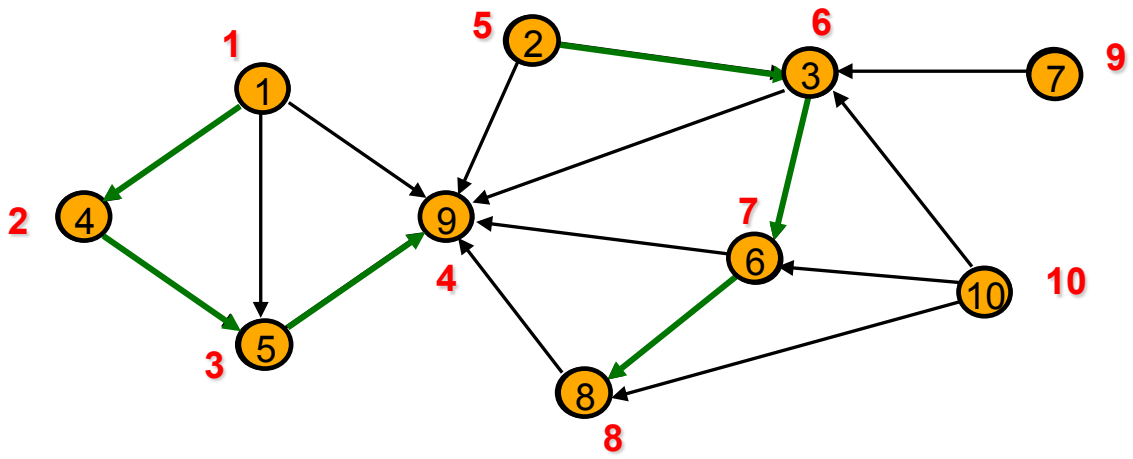
Há problemas em grafos que podem ser resolvidos através de ambas explorações

Exploração em profundidade (digrafos)

```
int ce;

TravessiaDFS(s) {
    desmarcar todos os vértices;
    ce = 0;
    DFS(s);
    for v ∈ V {
        if v está desmarcado
            DFS(v);
    }
}
```

Código adicional

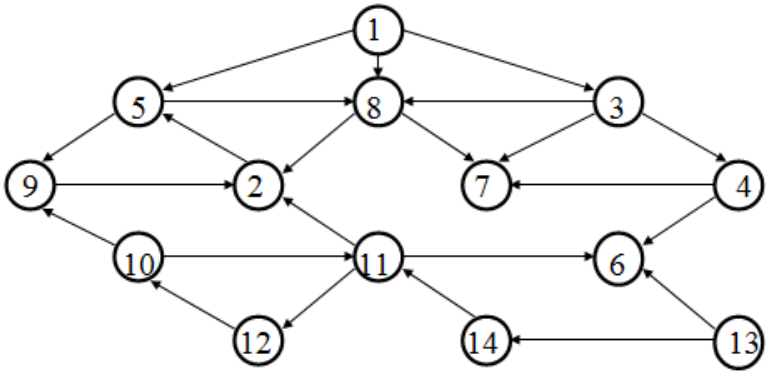
```
DFS(v) {
    marcar v;
    expl[v] = ++ce;
    // explorando v
    for <v,u> ∈ E {
        if u está desmarcado
            DFS(u);
    }
}
```

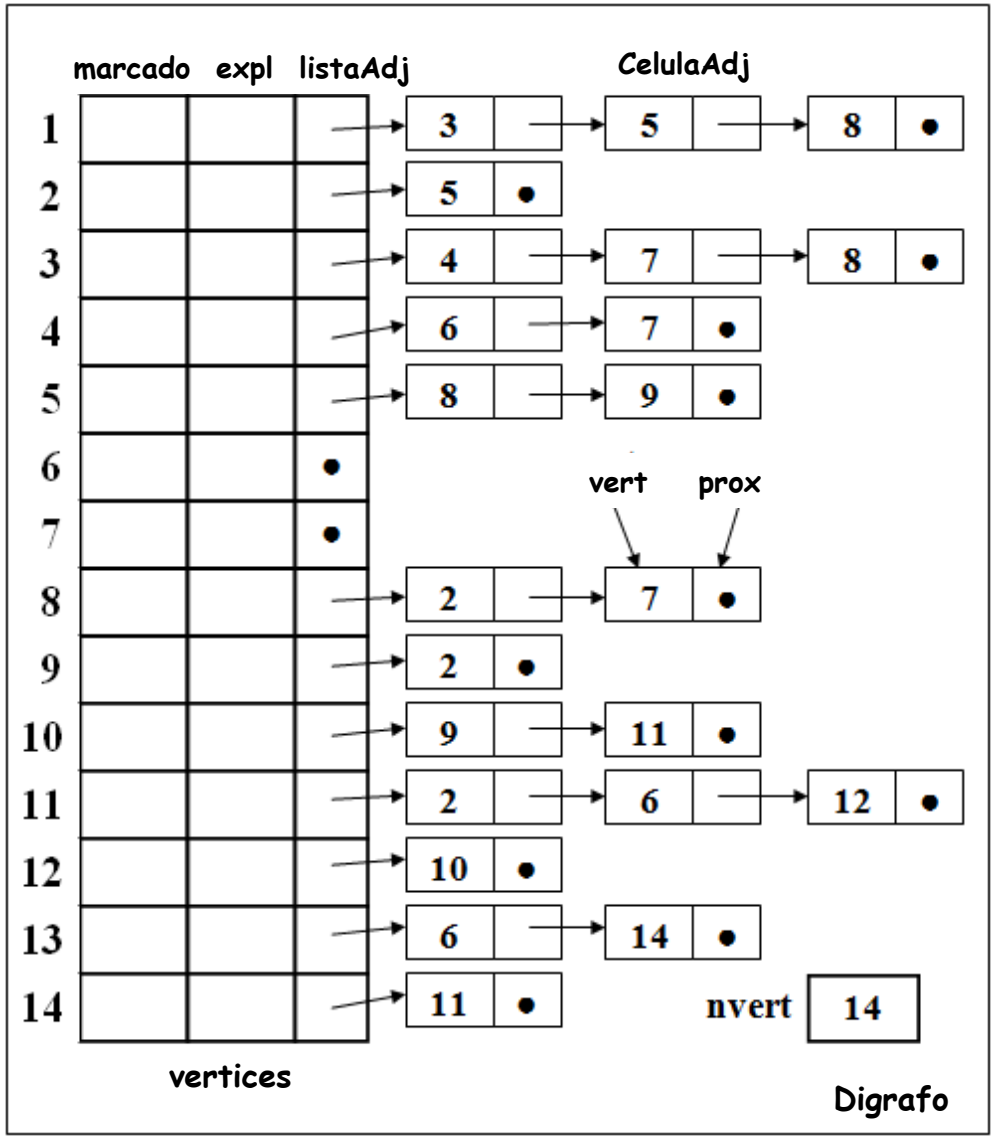


Solução também é válida para grafos desconexos

Exemplo de implementação

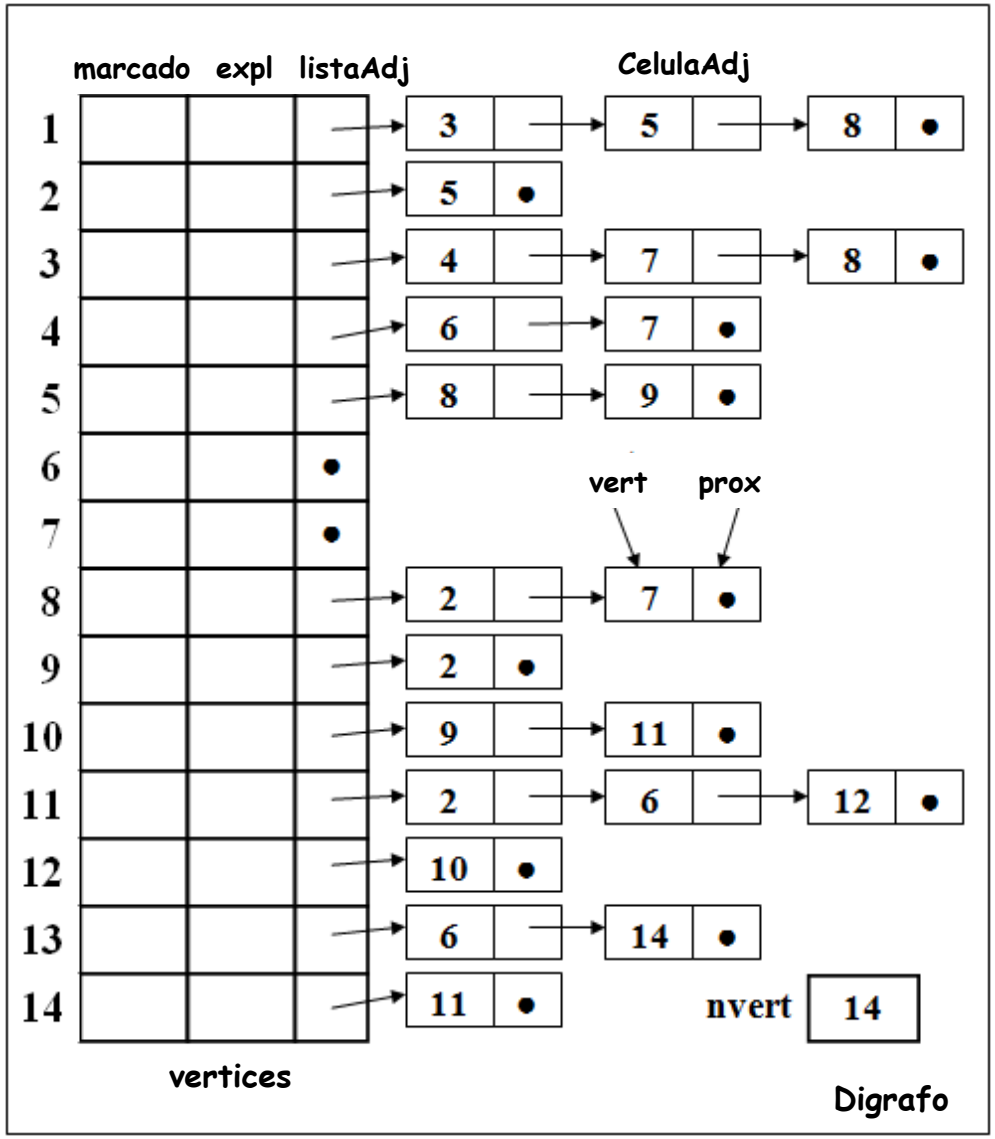
- Podemos implementar ambas explorações guardando a numeração das visitas na própria estrutura de dados.
- Exemplo:





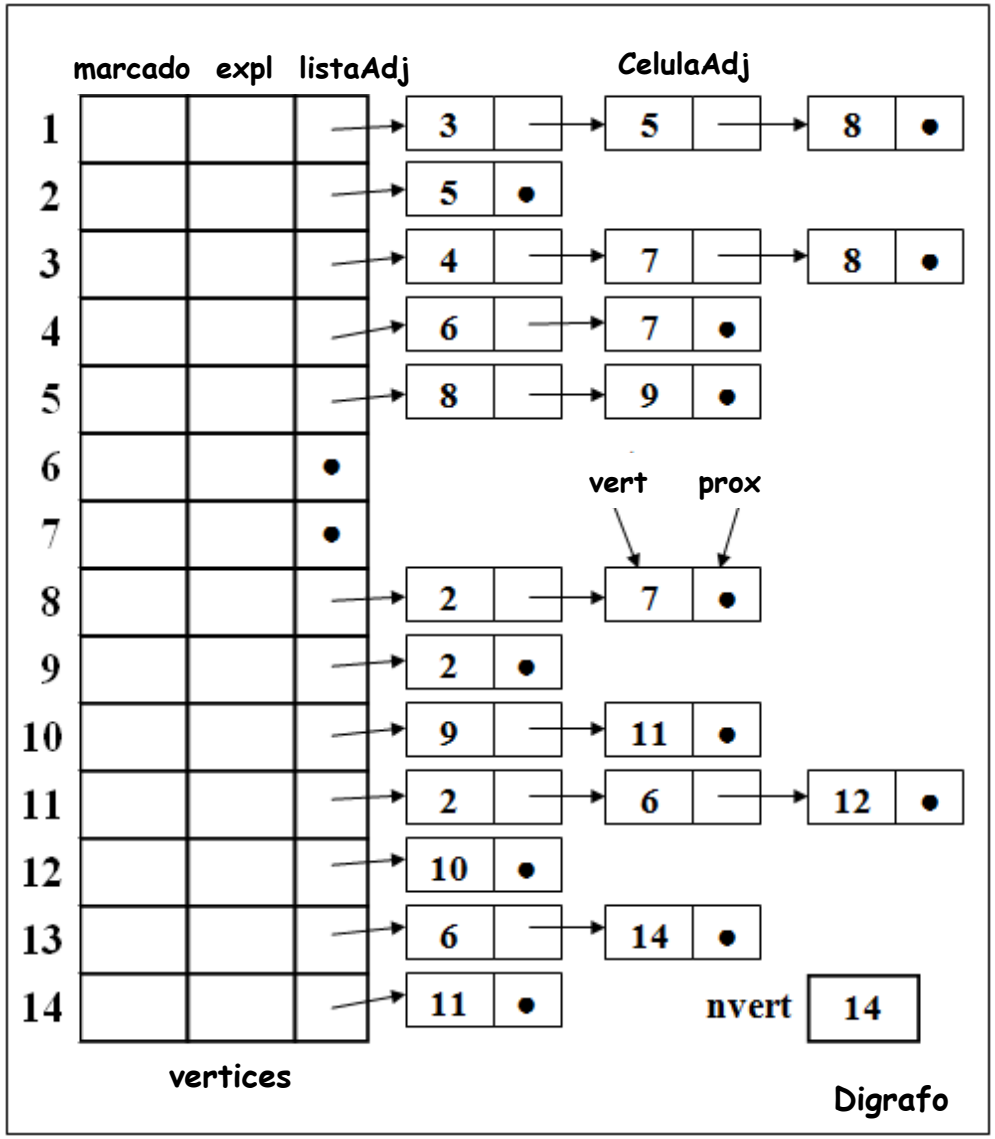
Exemplo de implementação

```
typedef int vertice;

struct CelulaAdj {
   vertice vert;
   CelulaAdj *prox;
};

struct CelulaVert {
   bool marcado;
   int expl;
   CelulaAdj *listaAdj;
};

struct Digrafo {
   int nvert;
   CelulaVert *vertices;
};

Digrafo G;
int ce;
```



Implementação (em profundidade)

```
TravessiaDFS(vertice s)
{
  vertice v;

  for(v=1; v<=G.nvert; v++)
    G.vertices[v].marcado = false;

  ce = 0;
  DFS(s);

  for(v=1; v<=G.nvert; v++) {
    if(!G.vertices[v].marcado)
      DFS(v);
  }
}
```



Implementação (em profundidade)

```
DFS(vertice s) {
  CelulaAdj *p;

  G.vertices[s].marcado = true;
  G.vertices[s].expl = ++ce;

  p = G.vertices[s].listaAdj;

  while (p!=NULL) {
    if(!G.vertices[p->vert].marcado)
      DFS(p->vert);
    p = p->prox;
  }
}
```