Grafos

Universidade Federal de Mato Grosso do Sul
Câmpus de Ponta Porã - CPPP
Desafios de Programação

Grafos

Definição

Um grafo não direcionado (ou simplesmente grafo) é dado por:

- um conjunto V de vértices
- um conjunto E de arestas, onde cada aresta $e \in E$ relaciona (liga) dois vértices $u \in V$ e $v \in V$.

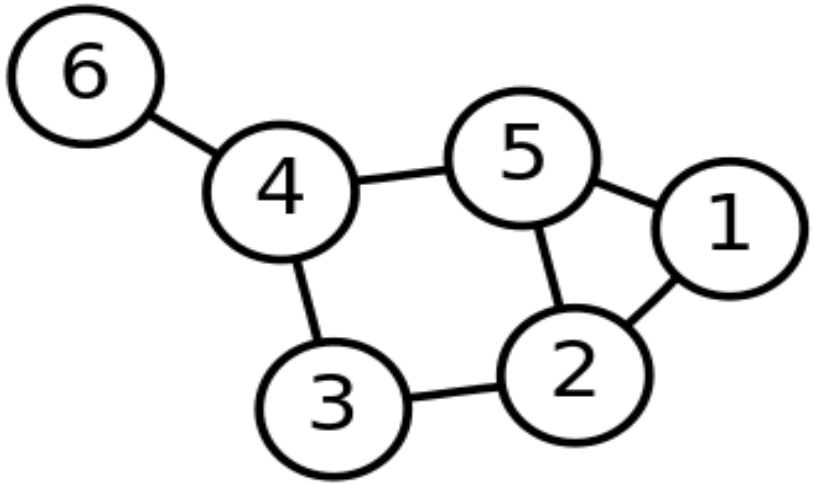


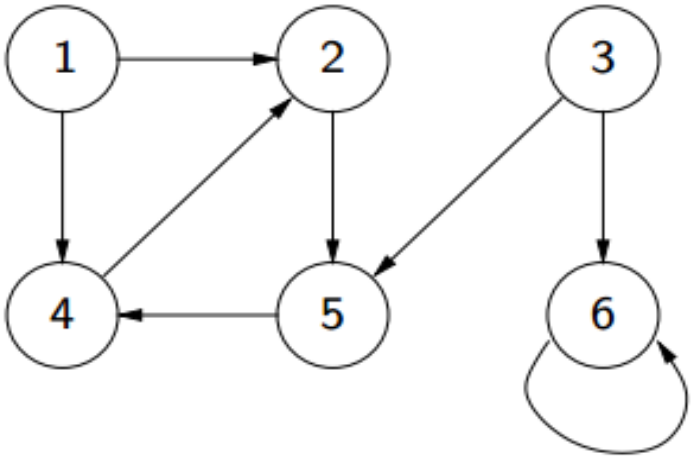
Figura 1 : Um grafo com 6 vértices e 7 arestas.

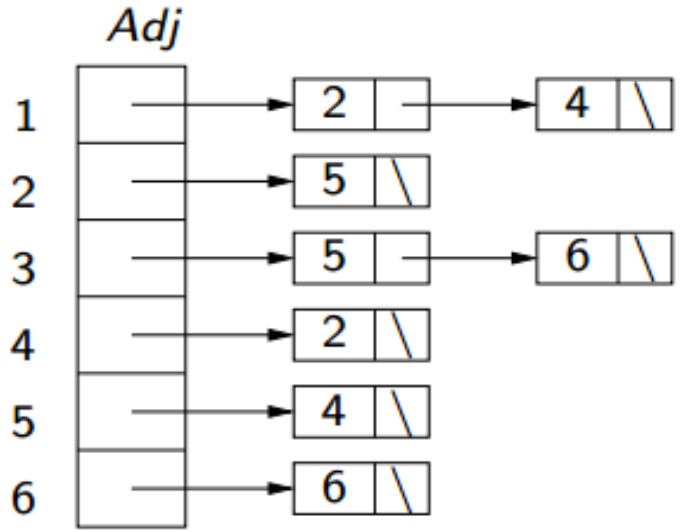
Representação de Grafos

Lista de Adjacência:

- Dado um grafo $G = (V, E)$, esta representação é tipicamente preferida pois é uma maneira compacta de representar grafos esparsos - aqueles onde $|E| << |V|^2$
- A representação por listas de adjacência consiste em um vetor Adj com $|V|$ listas de adjacência, uma para cada vértice $v \in V$.
- Para cada $u \in V$, Adj[u] contém ponteiros para todos os vértices v tal que $(u, v) \in E$. Ou seja, Adj[u] consiste de todos os vértices que são adjacentes a u

Lista de Adjacência





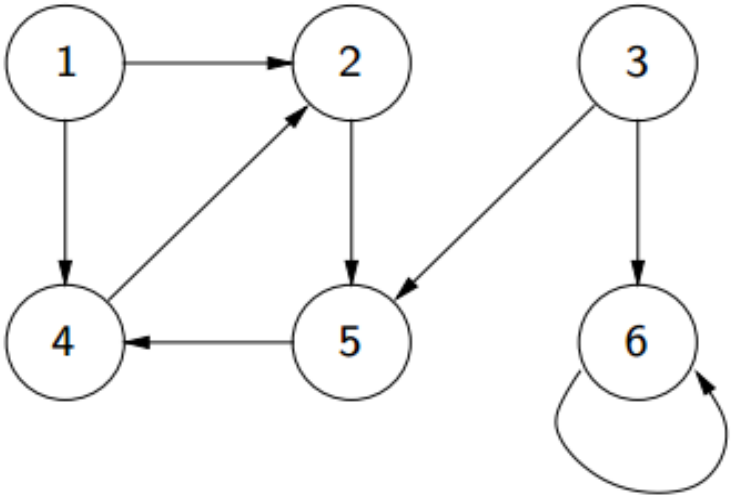
Matriz de Adjacência

Matriz de Adjacência:

- A representação por matriz de adjacência é preferida, entretanto, quando o grafo é denso, ou seja, quando $|E| \approx |V|^2$
- Para um grafo $G = (V, E)$, assumimos que os vértices são rotulados com números $1, 2, \ldots, |V|$.
- A representação consiste de uma matriz $A = (a_{ij})$ de dimensões $|V| \times |V|$, onde

$$a_{ij} = 1, \text{ se } (i, j) \in E$$
$$a_{ij} = 0, \text{ c.c}$$

Matriz de Adjacência



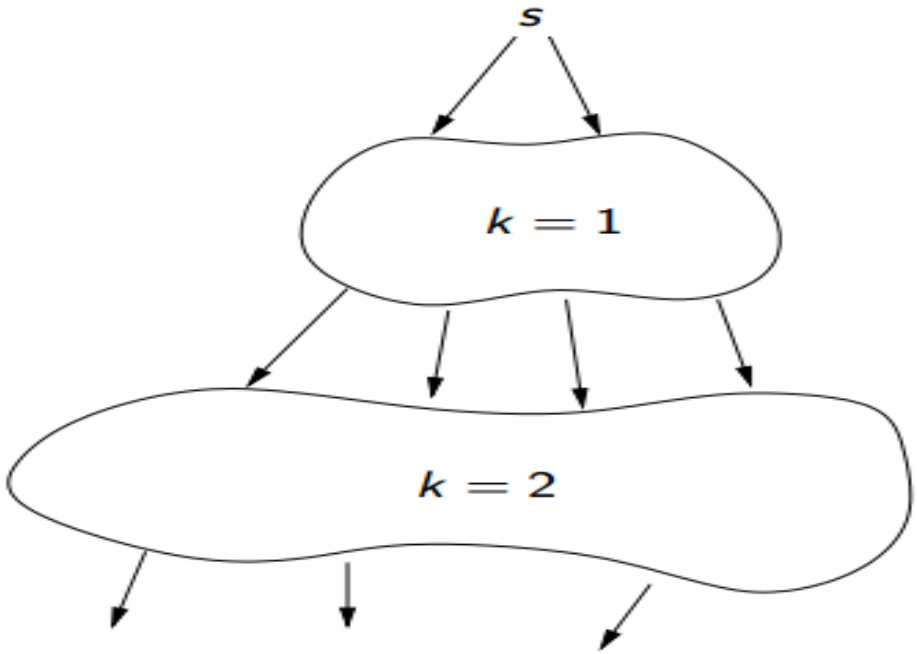
	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Algoritmos de Busca em Grafos

Busca em Largura:

- A busca em largura é um dos algoritmos mais simples para exploração de um grafo.
- Dados um grafo G = (V, E) e um vértice s, chamado de fonte, a busca em largura sistematicamente explora as arestas de G de maneira a visitar todos os vértices alcançáveis a partir de s.
- Esta busca é dita em largura porque ela expande a fronteira entre vértices conhecidos e desconhecidos de uma forma uniforme ao longo da fronteira.
- Ou seja, o algoritmo descobre todos os vértices com distância k de s antes de descobrir qualquer vértice de distância $k + 1$.

Busca em Largura



Algoritmo Busca em Largura

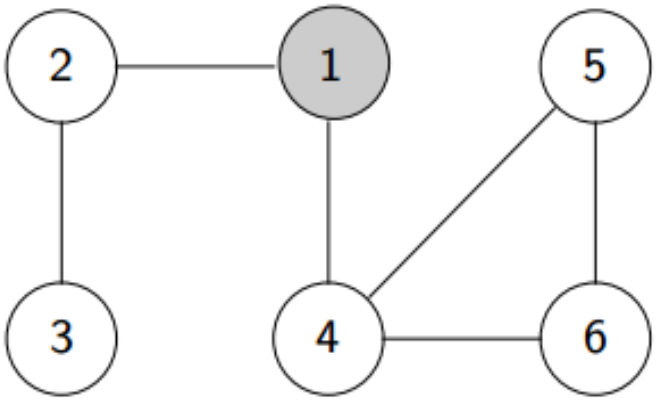
Para controlar a busca, o algoritmo pinta cada vértice na cor branca, cinza ou preta.

- Todos os vértices iniciam com a cor branca e podem, mais tarde, se tornar cinza e depois preta.
- Branca: não visitado
- Cinza: visitado
- Preta: visitado e seus nós adjacentes visitados

Algoritmo Busca em Largura

```
BFS(G, s)
1  para cada vértice u ← V[G] − {s}
2      cor[u] ← BRANCO
3      d[u] ← ∞
4      π[u] ← NULL
5  cor[s] ← CINZA
6  d[s] ← 0
7  π[s] ← NULL
8  Q ← nova Fila()
9  ENFILEIRA(Q, s)
10 enquanto !vazia(Q)
11     u ← DESENFILEIRA(Q)
12     para cada v ← Adj[u]
13         se cor[v] = BRANCO
14             cor[v] ← CINZA
15             d[v] = d[u] + 1
16             π[v] ← u
17             ENFILEIRA(Q, v)
18     cor[u] ← PRETO
```

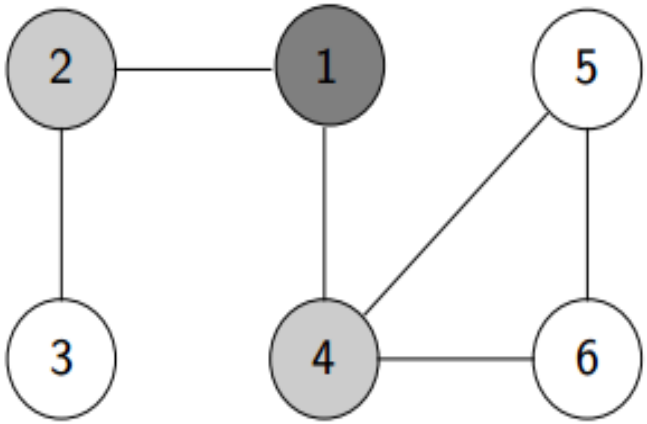
Exemplo



	1	2	3	4	5	6
d	0	∞	∞	∞	∞	∞
π	\	\	\	\	\	\
c	g	w	w	w	w	w

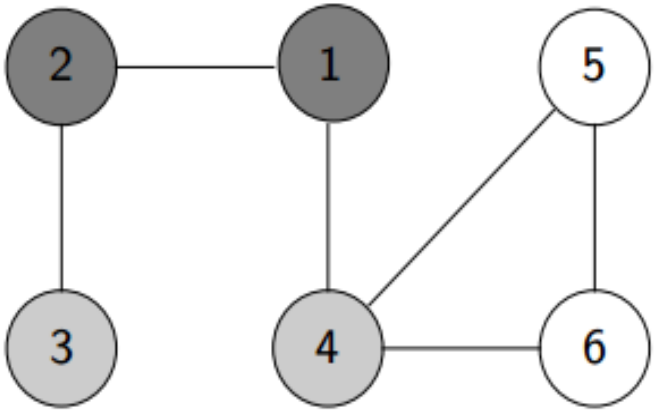
Q
1

Exemplo



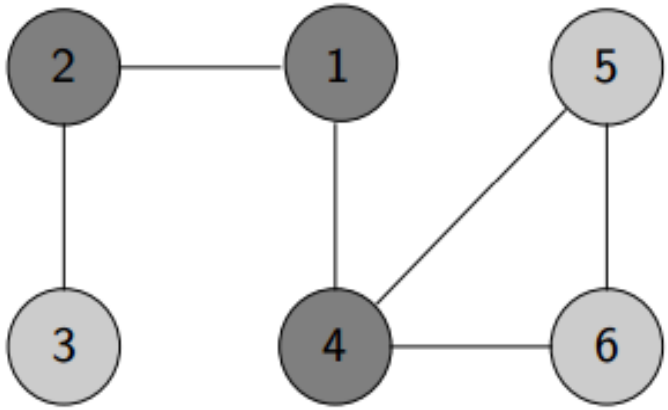
	1	2	3	4	5	6
d	0	1	∞	1	∞	∞
π	\	1	\	1	\	\
c	b	g	w	g	w	w
Q	2	4				

Exemplo



	1	2	3	4	5	6
d	0	1	2	1	∞	∞
π	0	1	2	1	\	\
c	b	b	g	g	w	w
Q	4	3				

Exemplo



	1	2	3	4	5	6
d	0	1	2	1	2	2
π	0	1	2	1	4	4
c	b	b	g	b	g	g
Q	3	5	6			

Implementação em C++

Busca em Largura

```
int n; //número de vértices
vector<int> distancia(n, INF); //INF representa infinito.
void dfs(vector< list<int> >& grafo, int u)
{
    distancia[u] = 0;
    queue<int> q;
    q.push(u);
    while(!q.empty())
    {
        int u = q.front();
        cout<<"vértice "<<u<<" foi visitado"<<endl;
        q.pop();
        for(list<int>::iterator it=grafo[u].begin();it!=grafo[u].end();it++)
        {
            if(distancia[*it]==INF)
            {
                distancia[*it] = distancia[u] + 1;
                q.push(*it);
            }
        }
    }
}
```

Busca em Profundidade

- A estratégia aqui é explorar o grafo em profundidade.
- Na busca em profundidade, as arestas são exploradas a partir do vértice mais recentemente visitado.
- Da mesma forma que a busca em largura, sempre que um vértice v é descoberto durante a busca na lista de adjacência de um outro vértice já visitado u, a DFS memoriza este evento ao definir o predecessor de v, $\pi[v]$ como u.

Busca em Profundidade

Os vértices do grafo são coloridos durante a busca.

- Branco: antes da busca.
- Cinza: quando o vértice for visitado.
- Preto: quando os vértices adjacentes foram visitados.

DFS marca cada vértice com um timestamp. Cada vértice tem dois timestamps.

d_v indica o instante em que v foi visitado (pintado com cinza).

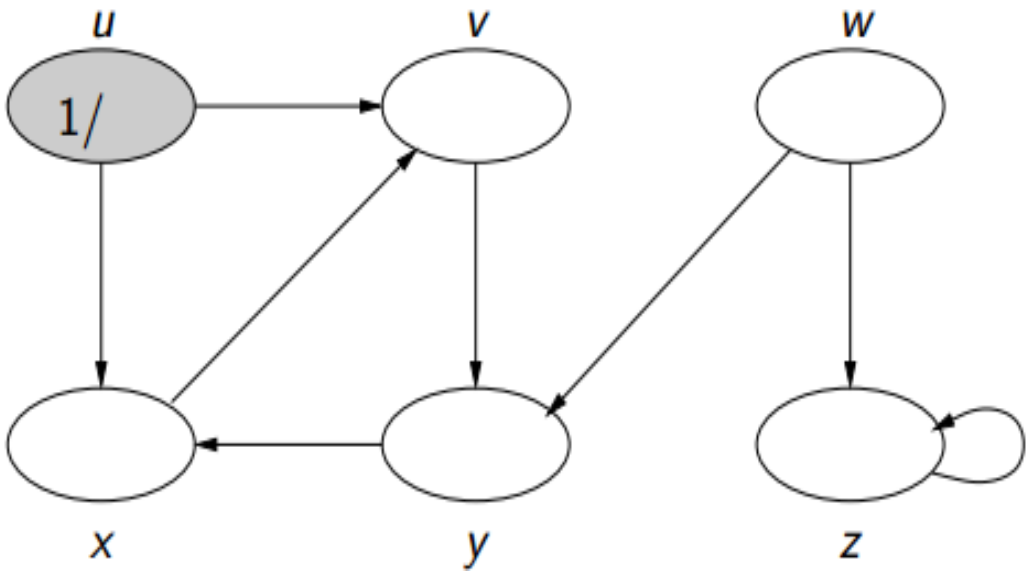
f_v indica o instante em que a busca pelos vértices na lista de adjacência de v foi completada (pintado de preto).

Algoritmo Busca em Profundidade

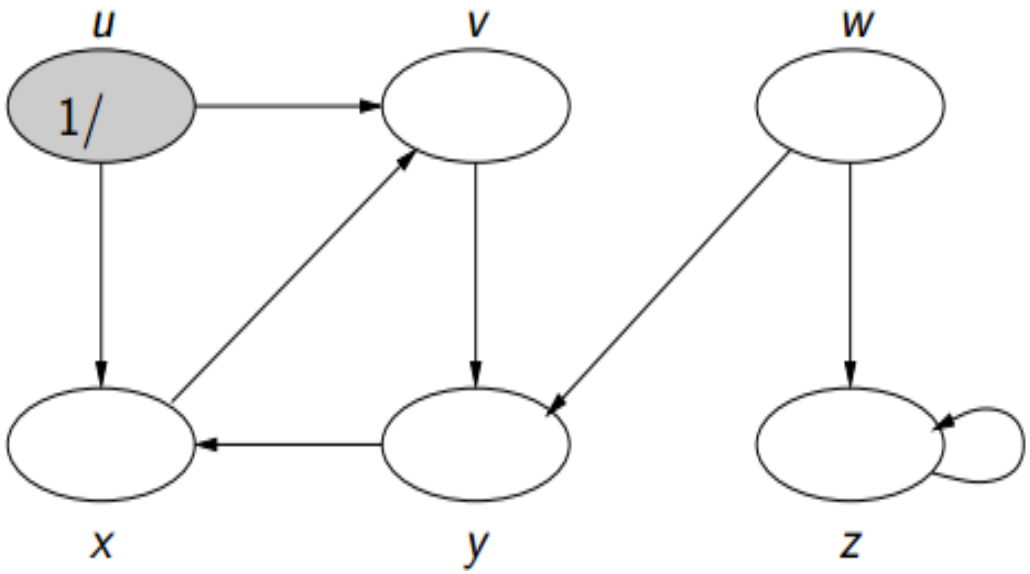
```
DFS(G)
for ∀u ∈ V[G] do
        cor[u] ← BRANCO
        π[u] ← NIL
tempo ← 0
for ∀u ∈ V[G] do
        if cor[u] = BRANCO then
                VisitaDFS(u)
```

```
VisitaDFS(u)
cor[u] ← CINZA
d[u] ← tempo ← tempo + 1
for ∀v ∈ Adj[u] do
        if cor[v] = BRANCO then
                π[v] ← u
                VisitaDFS(v)
cor[u] ← PRETO
F[u] ← tempo ← tempo + 1
```

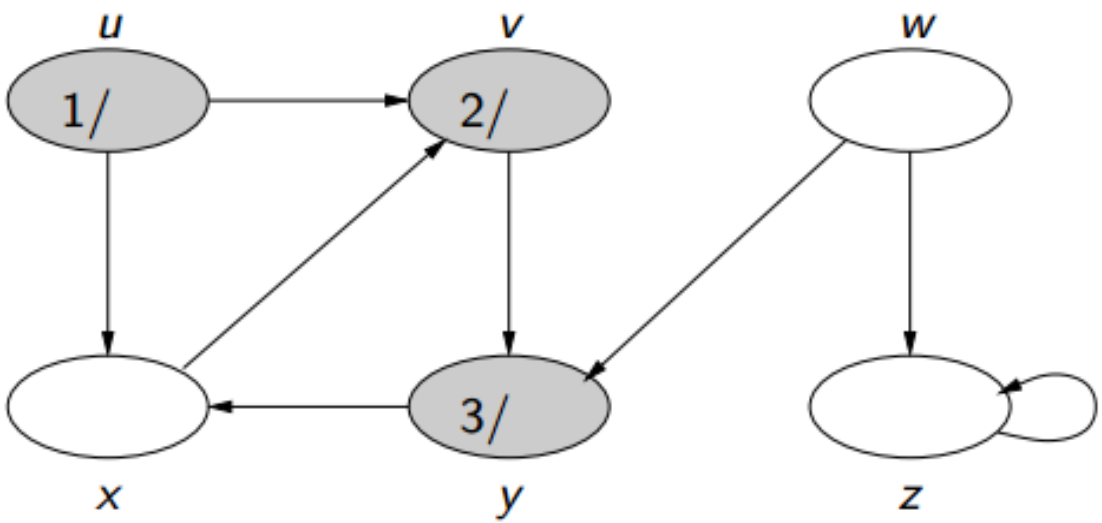
Exemplo Busca em Profundidade



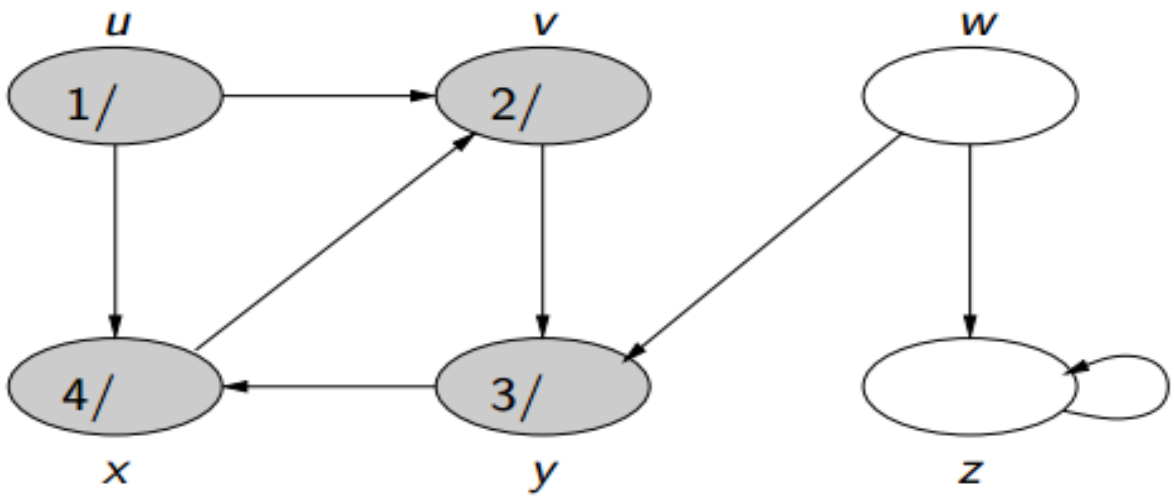
Exemplo Busca em Profundidade



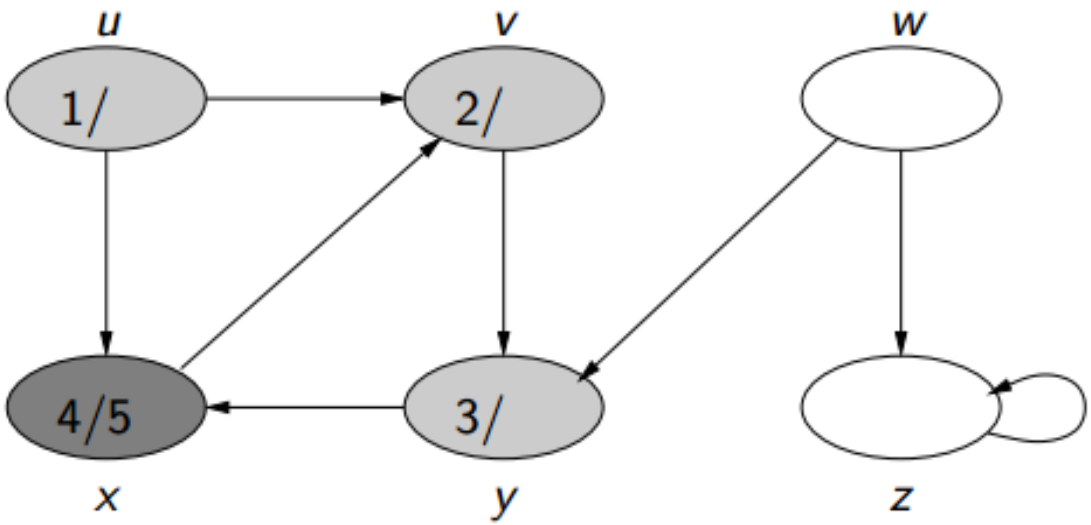
Exemplo Busca em Profundidade



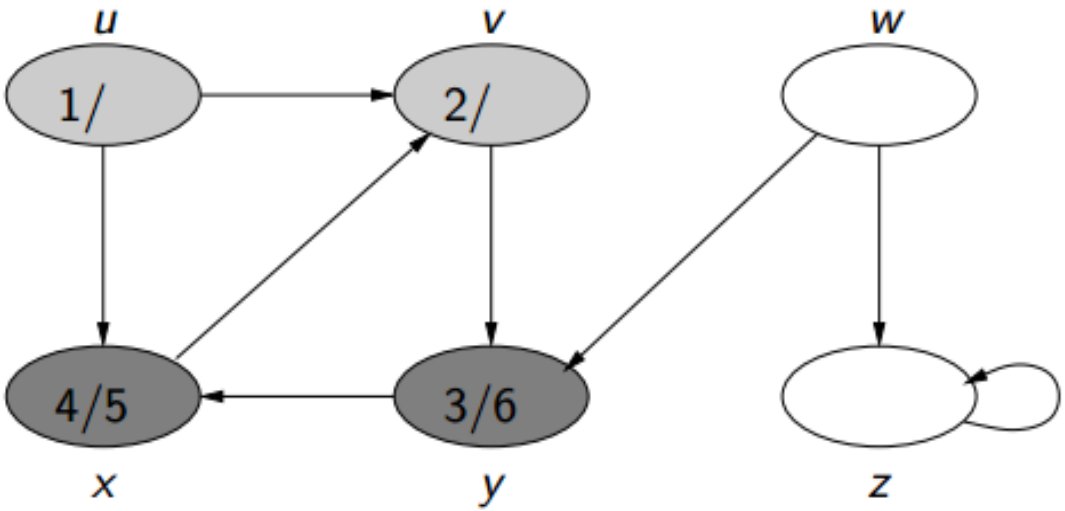
Exemplo Busca em Profundidade



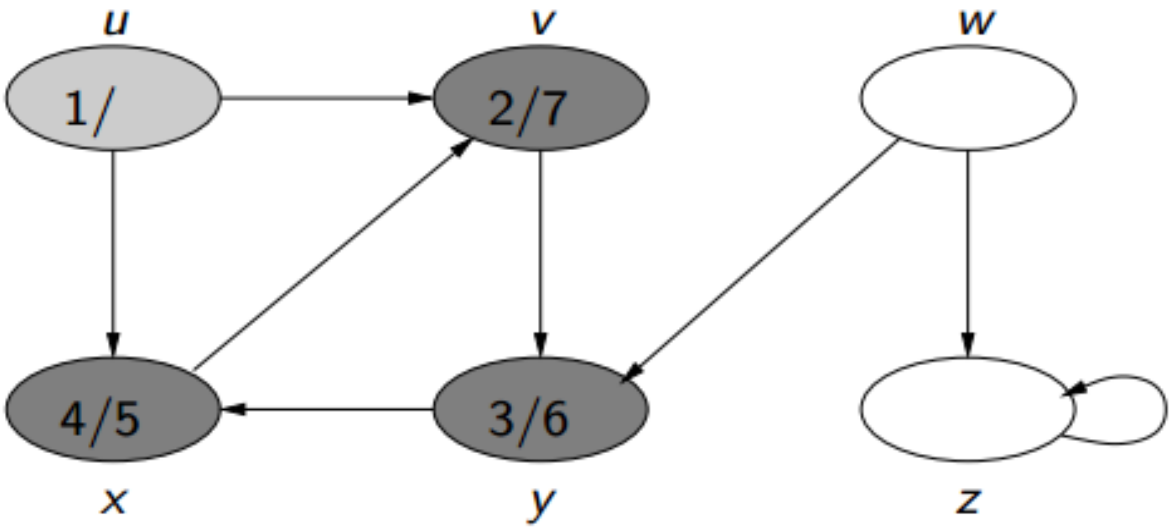
Exemplo Busca em Profundidade



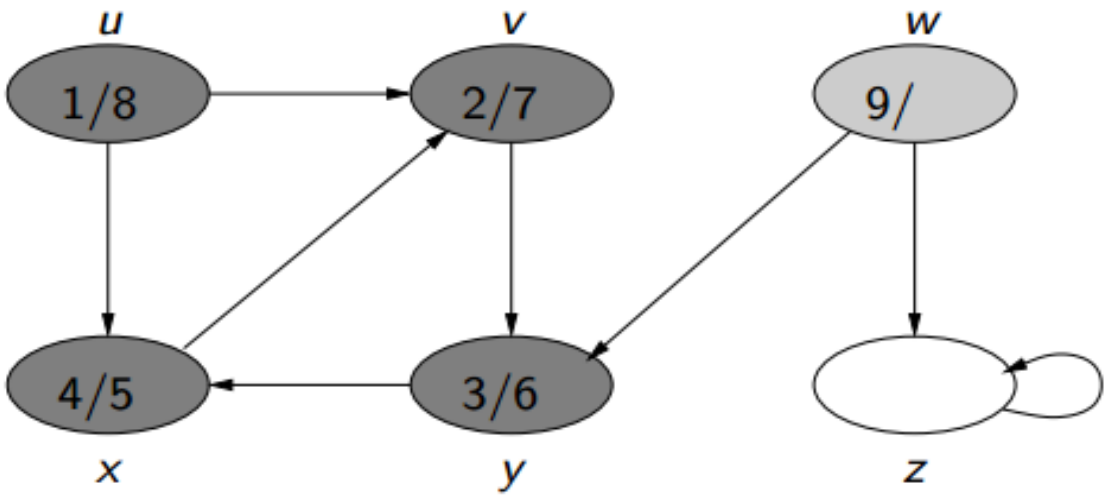
Exemplo Busca em Profundidade



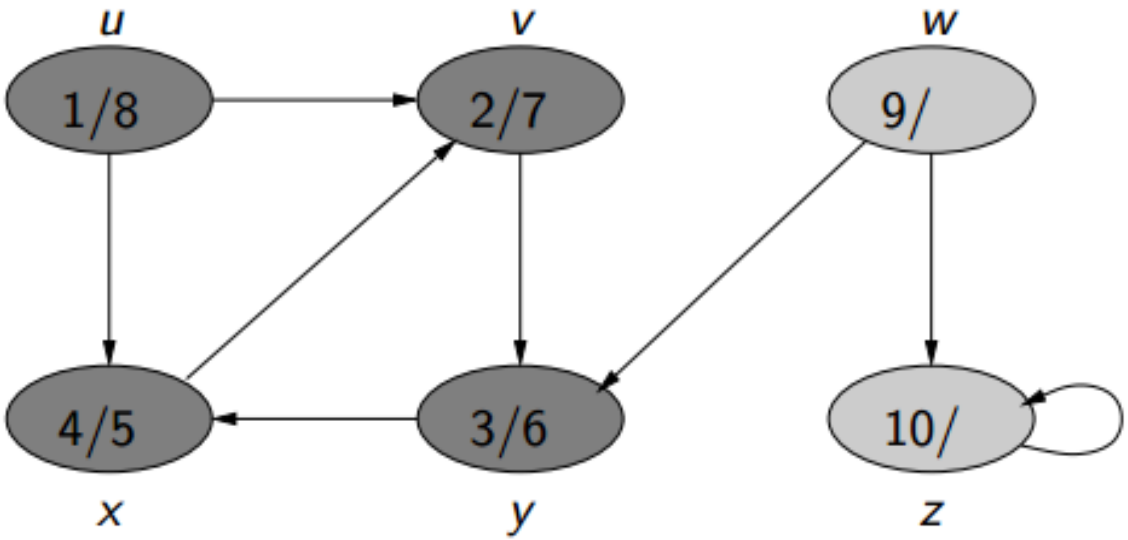
Exemplo Busca em Profundidade



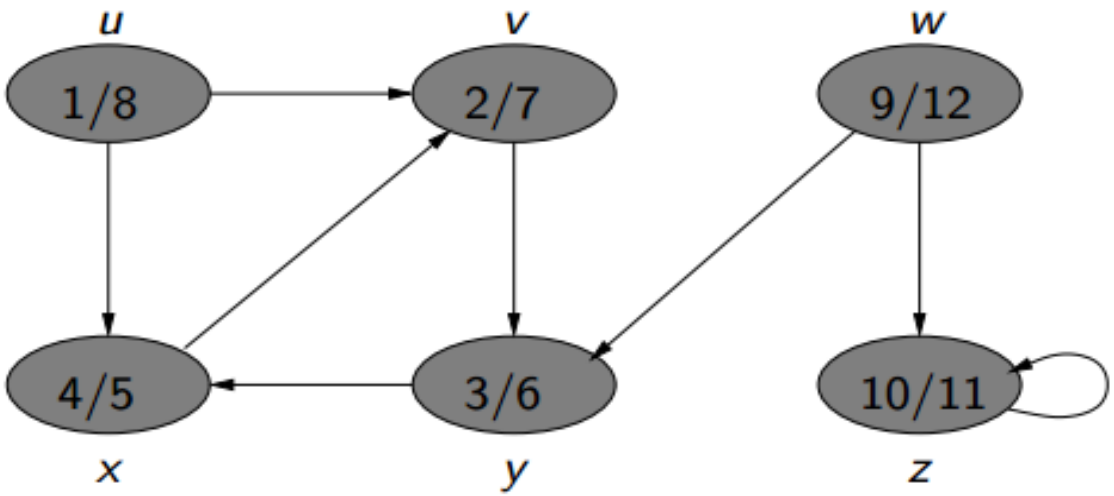
Exemplo Busca em Profundidade



Exemplo Busca em Profundidade



Exemplo Busca em Profundidade



Implementação em C++

Busca em Profundidade

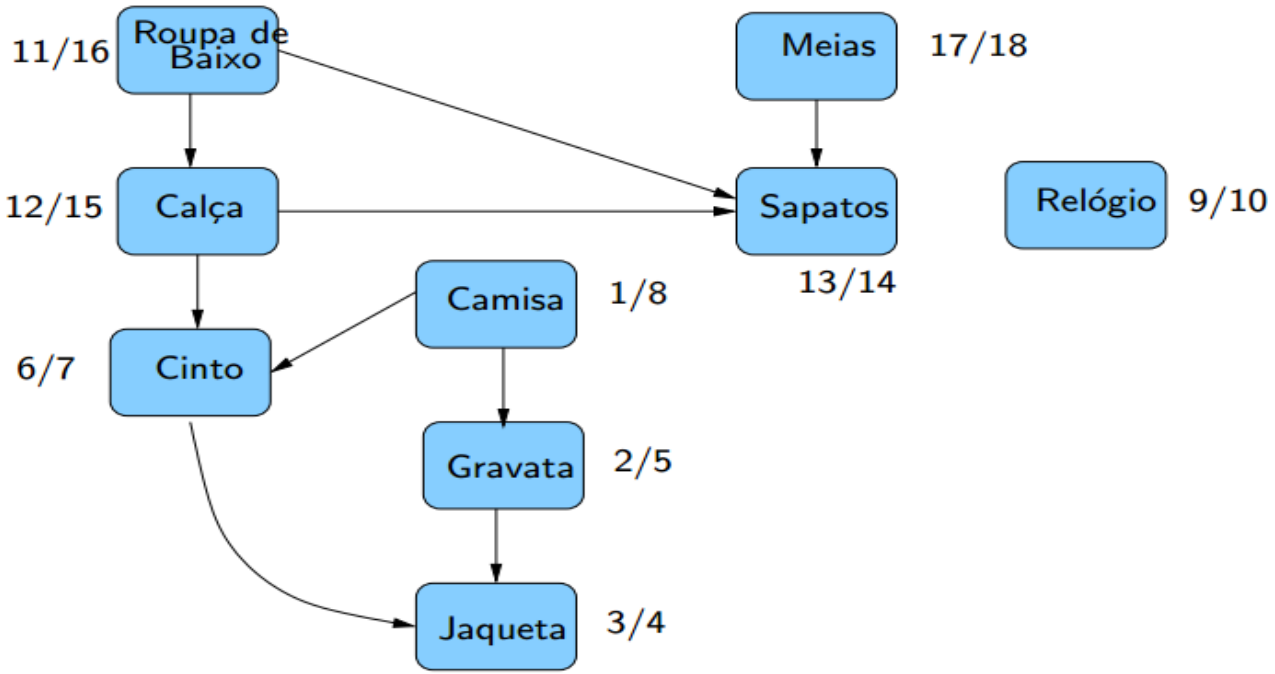
```
int n; //número de vértices
vector<bool> visitados(n, false);

void bfs(vector< list<int> >& grafo, int u) //u representa o vértice inicial
{
    cout<<"vértice "<<u<<" foi visitado"<<endl;
    visitados[u] = true;
    for(list<int>::iterator it=grafo[u].begin(); it!=grafo[u].end(); it++)
    {
        if(!visitados[*it])
            bfs(grafo, *it);
    }

}
```

Ordenação Topológica

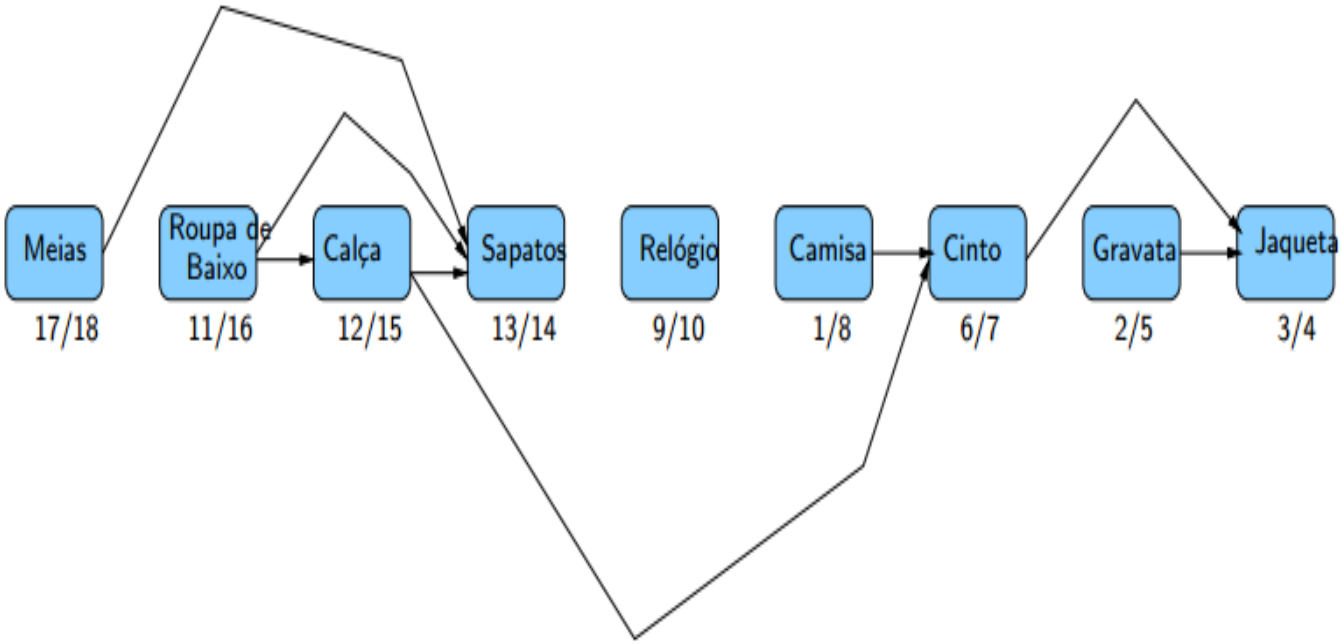
Mostraremos como busca em profundidade pode ser empregada para encontrar uma ordenação topológica de um grafo direcionado acíclico $G = (V, E)$.

- Uma ordenação topológica $(u_1, \ldots, u_n)$ dos vértices de G é uma ordenação linear tal que se $(u_i, u_j) \in E$, então u_i precede u_j na ordenação, ou seja, $i < j$.
- Ordenação topológica pode ser vista como um arranjo dos vértices na horizontal, tal que as arestas vão da esquerda para a direita.

Exemplo


11/16
Roupa de Baixo
Meias
17/18
12/15
Calça
Sapatos
13/14
Relógio
9/10
Camisa
1/8
6/7
Cinto
Gravata
2/5
Jaqueta
3/4

Exemplo


Meias
Roupa de Baixo
Calça
Sapatos
Relógio
Camisa
Cinto
Gravata
Jaqueta
17/18
11/16
12/15
13/14
9/10
1/8
6/7
2/5
3/4

Algoritmo

Algoritmo **Topological-Sort(G):**

Chame DFS(G) para computar $f[v]$ para cada vértice v. Conforme cada vértice terminar, insira ele na frente de uma lista encadeada.
Retorne a lista encadeada com os vértices.

Fazer:

Uva - 124 - Following Orders