ACH2024 –

Algoritmos e Estruturas de Dados II

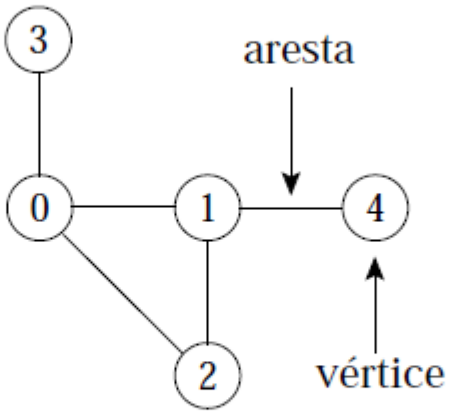
Prof. Helton Hideraldo Biscaro
heltonhb@usp.br

Parte 1 da Disciplina:

GRAFOS

Grafos

- **Grafo**: conjunto de vértices e arestas.
- **Vértice**: objeto simples que pode ter nome e outros atributos.
- **Aresta**: conexão entre dois vértices.



- Notação: $G = (V, A)$
  - G: grafo
  - V: conjunto de vértices
  - A: conjunto de arestas

Busca em Largura – Pseudocódigo Relembrando.

```
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 ← novaFila()
9 ENFILEIRA(Q, s)
```

Legenda:

cor[u]; *//indicativo de atingibilidade;*
π[u]; *//indica o vértice predecessor de u(pai);*
d[u]; *//indica a distância desde a origem d(s,u) - em arestas;*
Q; *//indica a fila (FIFO) – ponto chave do algoritmo.*

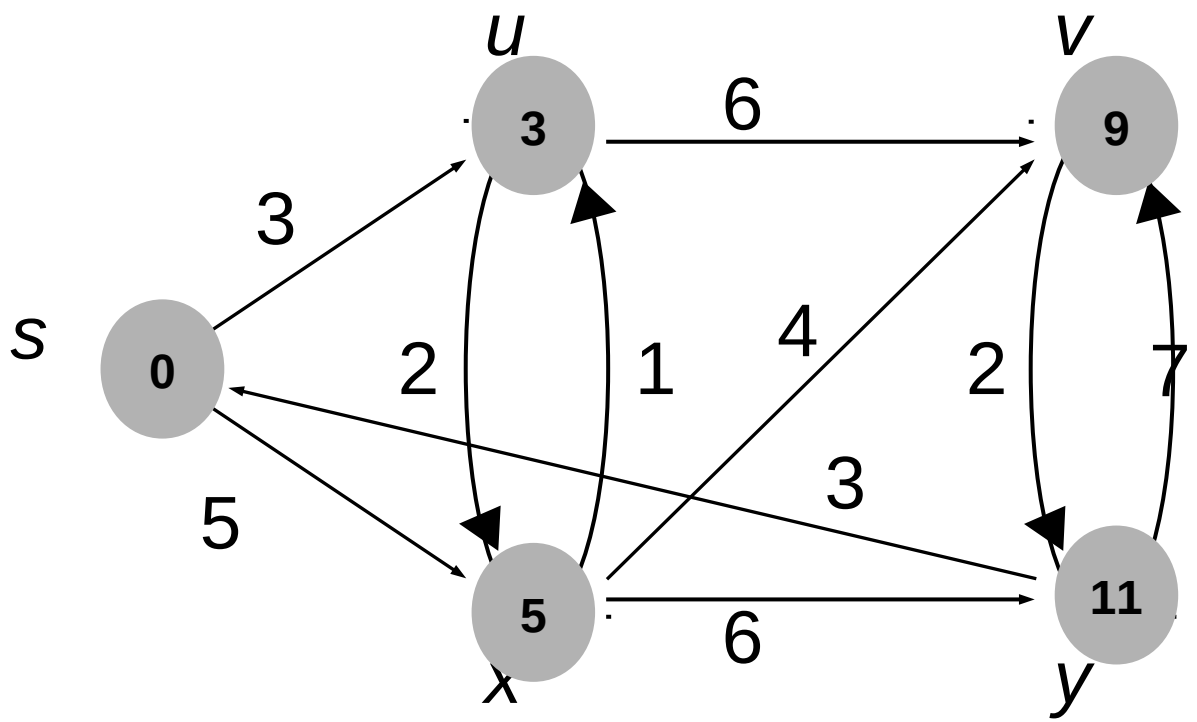
Busca em Largura – Pseudocódigo Relembrando.

```
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
```

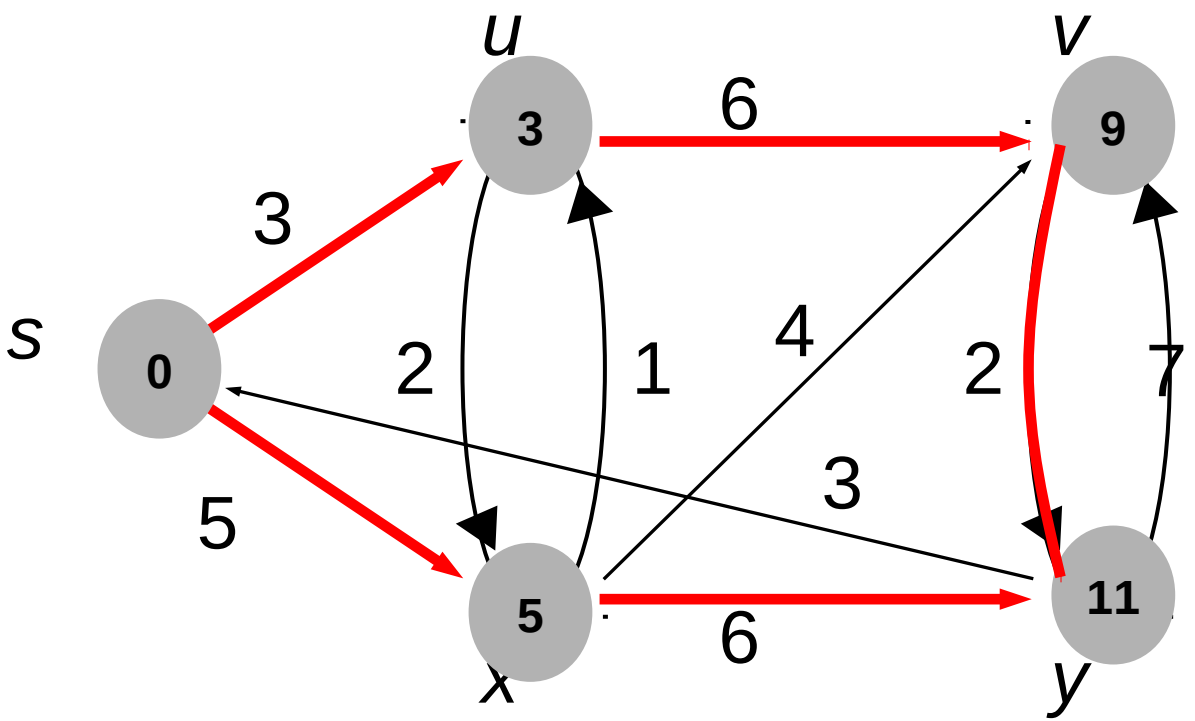
Caminho Mínimo

- A busca em largura obtém o caminho mais curto de u até v.
- O procedimento BFS constrói uma árvore de busca em largura que é armazenada na variável $\pi[\,]$
- Exercício: Escreva um código em C que imprime o caminho mais curto entre o vértice inicial e qualquer outro vértice do grafo:

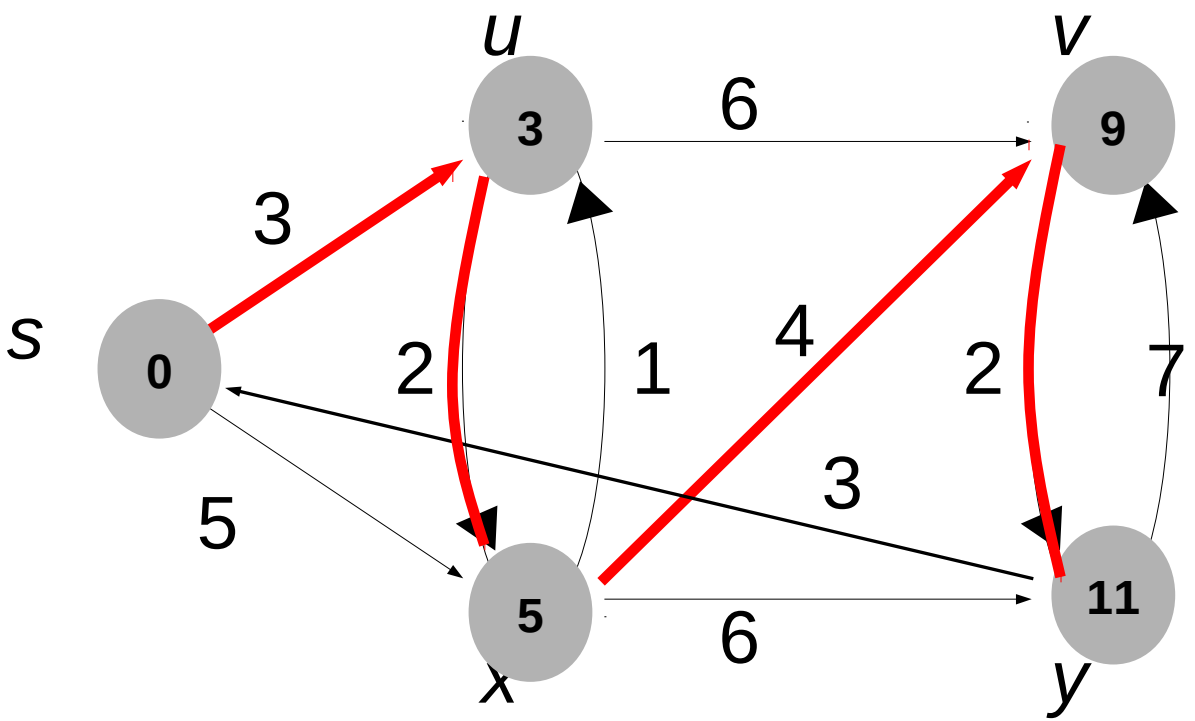
Caminho Mínimo- Pesos



Como Obter um caminho mínimo partindo de s para y?



Outra Possibilidade:



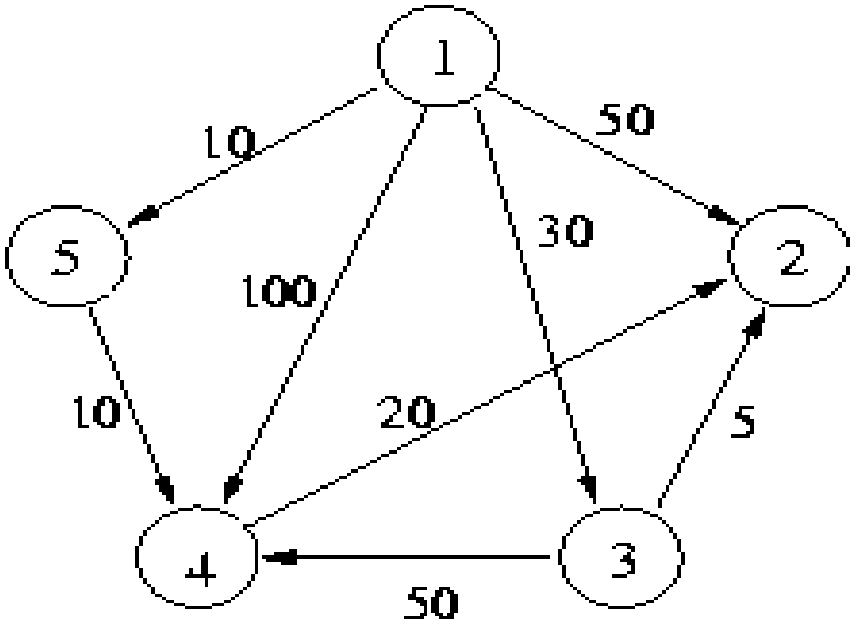
O algoritmo de Dijkstra

- Mantém um conjunto S de vértices cujos caminhos mais curtos até um vértice origem já são conhecidos.
- Produz uma árvore de caminhos mais curtos de um vértice origem s para todos os vértices que são alcançáveis a partir de s.
- Utiliza a técnica de **relaxamento**:
  - Para cada vértice $v \in V$ o atributo $p[v]$ é um limite superior do peso de um caminho mais curto do vértice origem s até v.
  - O vetor $p[v]$ contém uma estimativa de um caminho mais curto.
- O primeiro passo do algoritmo é inicializar os antecessores e as estimativas de caminhos mais curtos:

O algoritmo de Dijkstra – Pseudocódigo

função *Dijkstra*(G = *[1..n, 1..n]*: *grafo*): vetor[2..n]
    C := {2,3,...,n}
    Para *i* := 2 **até** *n*:
        p*[i]* := G*[1,i]*
    Repetir *n-2* vezes:
        v := Elemento de *C* que minimiza *D[v]*
        C := *C - {v}*
        Para cada elemento *w* de C:
            p[w] := min(p[w],p[v]+ G[v,w])
    Retornar *p*

Exemplo:



Passo	v	C	D
Início	-	{2,3,4,5}	[50,30,100,10]
1	5	{2,3,4}	[50,30,20,10]
2	4	{2,3}	[40,30,20,10]
3	3	{2}	[35,30,20,10]

O algoritmo de Dijkstra

- Somente temos o custo do caminho;
- Como Obter o caminho?

O algoritmo de Dijkstra

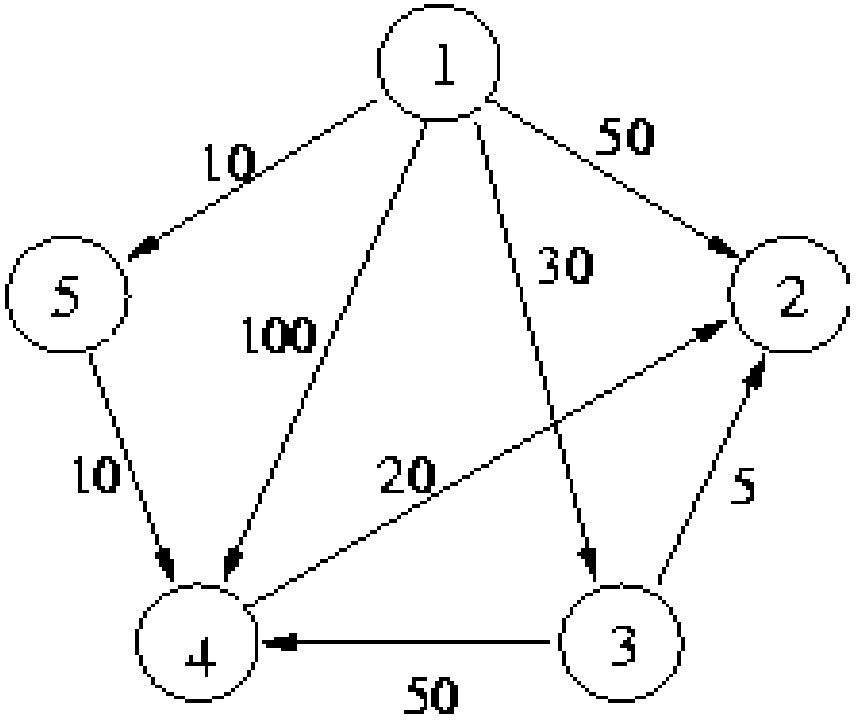
- Somente temos o custo do caminho;
- Como Obter o caminho?

Basta acrescentar mais um vetor Ant*[2..n]*, onde Ant*[v]* indica o vértice que precede *v* no caminho mais curto

O algoritmo de Dijkstra – Pseudocódigo

função *Dijkstra*(G = *[1..n, 1..n]*: *grafo*): vetor[2..n]
    C := {2,3,...,n}
    Para *i* := 2 **até** *n*:
        p*[i]* := *L[1,i]*
        Ant[i] := 1
    Repetir *n-2* vezes:
        v := Elemento de *C* que minimiza p*[v]*
        C := *C - {v}*
        Para cada elemento *w* de C:
            Se *p[w] > p[v]+ G[v,w]* ***então***
                p[w] := p[v]+ G[v,w]
                Ant[w] := v
    Retornar Ant

Exemplo:



Passo	v	C	p	Ant
Início	-	{2,3,4,5}	[50,30,100,10]	[1,1,1,1]
1	5	{2,3,4}	[50,30,20,10]	[1,1,5,1]
2	4	{2,3}	[40,30,20,10]	[4,1,5,1]
3	3	{2}	[35,30,20,10]	[3,1,5,1]

O algoritmo de Dijkstra – Pseudocódigo

Como selecionar o elemento w de forma eficiente?

Porque o algoritmo funciona?

- O algoritmo usa uma estratégia gulosa: sempre escolher o vértice mais leve (ou o mais perto) em $V - S$ para adicionar ao conjunto solução S,
- O algorimo de Dijkstra sempre obtém os caminhos mais curtos, pois cada vez que um vértice é adicionado ao conjunto S temos que $p[u] = \delta(Raiz, u)$.

O algoritmo de Dijkstra – Pseudocódigo

Exercício:

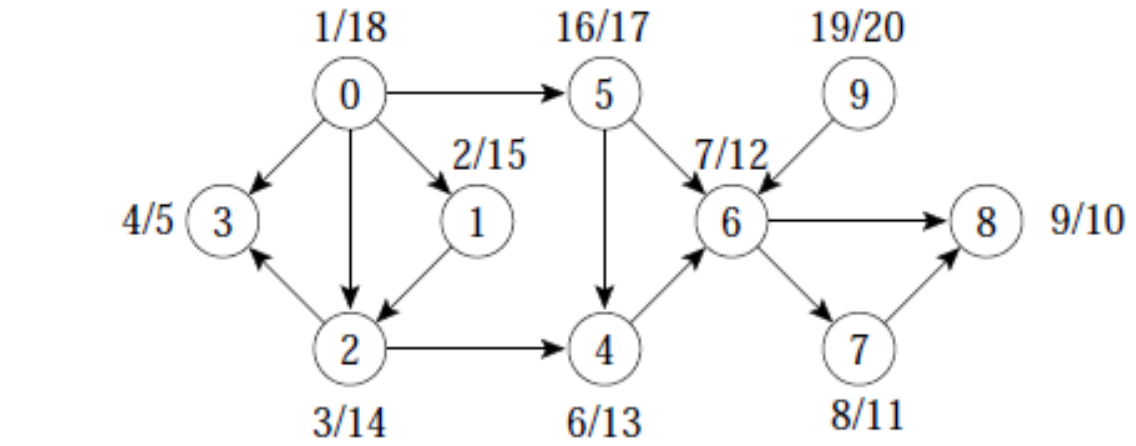
Traduzir o Pseudocódigo Anterior Para Linguagem C:

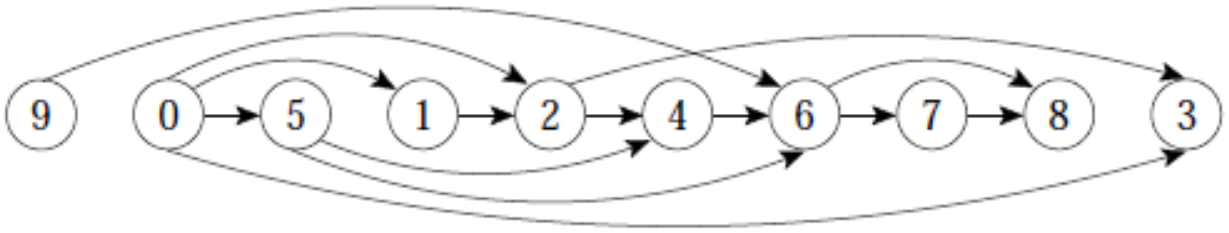
Ordenação Topológica

- Ordenação linear de todos os vértices, tal que se G contém uma aresta (u, v) então u aparece antes de v.
- Pode ser vista como uma ordenação de seus vértices ao longo de uma linha horizontal de tal forma que todas as arestas estão direcionadas da esquerda para a direita.
- Pode ser feita usando a busca em profundidade.

Ordenação Topológica

- Os grafos direcionados acíclicos são usados para indicar precedências entre eventos.
- Uma aresta direcionada (u, v) indica que a atividade u tem que ser realizada antes da atividade v.



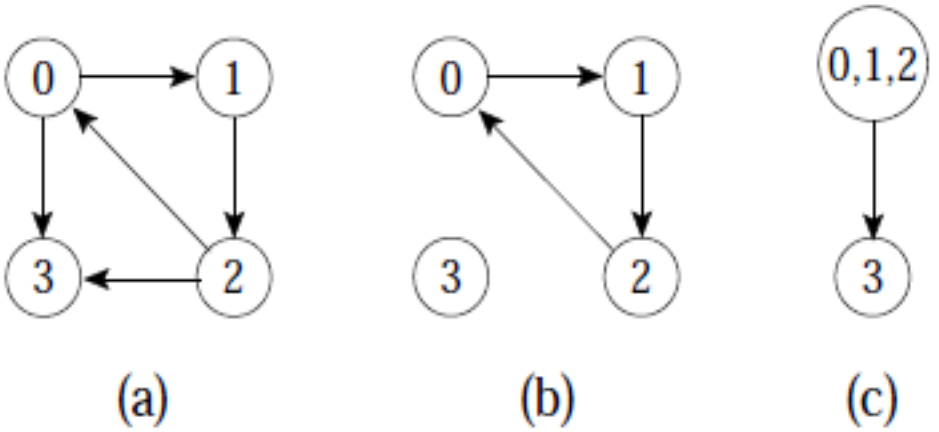


Ordenação Topológica

- Algoritmo para ordenar topologicamente um grafo direcionado acíclico $G = (V, A)$:
  1. Chama *BuscaEmProfundidade(G)* para obter os tempos de término $t[u]$ para cada vértice u.
  2. Ao término de cada vértice insira-o na frente de uma lista linear encadeada.
  3. Retorna a lista encadeada de vértices.
- A Custo $O(|V| + |A|)$, uma vez que a busca em profundidade tem complexidade de tempo $O(|V| + |A|)$ e o custo para inserir cada um dos $|V|$ vértices na frente da lista linear encadeada custa $O(1)$.

Componentes Fortemente Conectados

- Um componente fortemente conectado de $G = (V, A)$ é um conjunto maximal de vértices $C \subseteq V$ tal que para todo par de vértices u e v em C, u e v são mutuamente alcançáveis
- Podemos particionar V em conjuntos V_i, $1 \leq i \leq r$, tal que vértices u e v são equivalentes se e somente se existe um caminho de u a v e um caminho de v a u.



Componentes Fortemente Conectados – Algoritmo

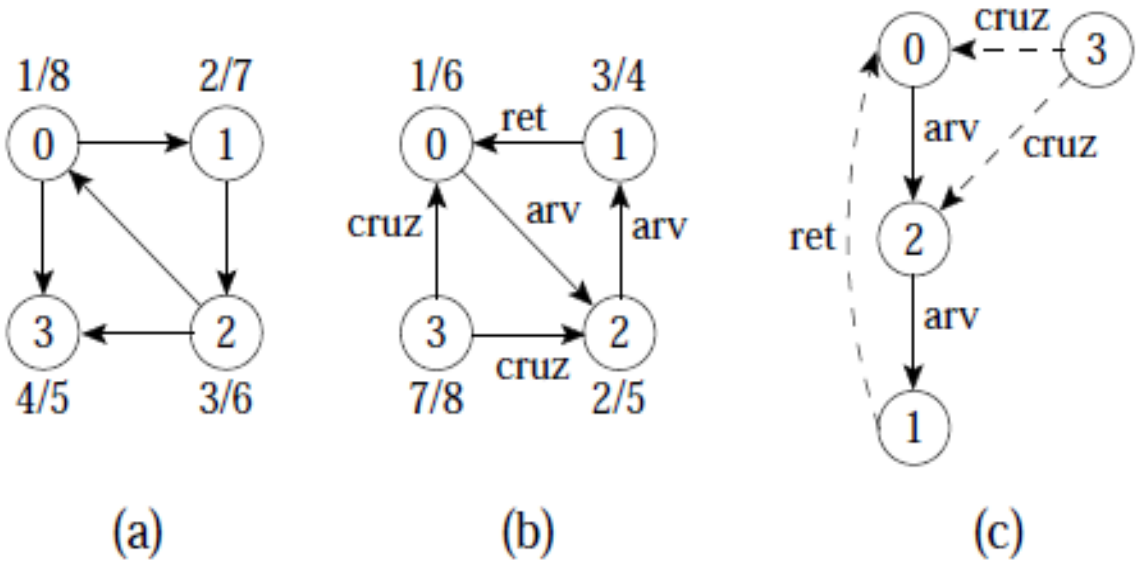
- Usa o **transposto** de G, definido $G^T = (V, A^T)$, onde $A^T = \{(u, v) : (v, u) \in A\}$, isto é, A^T consiste das arestas de G com suas direções invertidas.
- G e G^T possuem os mesmos componentes fortemente conectados, isto é, u e v são mutuamente alcançáveis a partir de cada um em G se e somente se u e v são mutuamente alcançáveis a partir de cada um em G^T.

Componentes Fortemente Conectados – Algoritmo

1. Chama *BuscaEmProfundidade*(G) para obter os tempos de término $t[u]$ para cada vértice u.
2. Obtem G^T.
3. Chama *BuscaEmProfundidade*(G^T), realizando a busca a partir do vértice de maior $t[u]$ obtido na linha 1. Inicie uma nova busca em profundidade a partir do vértice de maior $t[u]$ dentre os vértices restantes se houver.
4. Retorne os vértices de cada árvore da floresta obtida como um componente fortemente conectado separado.

Componentes Fortemente Conectados – Exemplo

- A parte (b) apresenta o resultado da busca em profundidade sobre o grafo transposto obtido, mostrando os tempos de término e a classificação das arestas.
- A busca em profundidade em G^T resulta na floresta de árvores mostrada na parte (c).



Problemas modelados por grafos

Problema do Carteiro Chines:

Problema discutido pelo matemático chinês Mei-Ku Kuan

Um carteiro deseja entregar suas cartas, percorrendo a menor distância possível e retornando ao ponto de partida

Ele deve passar por cada estrada pelo menos uma vez, mas deve evitar passar por muitas estradas mais de uma vez.