Universidade de Pernambuco - Campus Garanhuns
Licenciatura em Computação

Algoritmos e Estruturas de Dados

Algoritmos em Grafos

Prof. Emanoel Barreiros

Busca em profundidade

- A busca em profundidade, do inglês *depth-first search),* é um algoritmo para caminhar no grafo
- A estratégia é buscar o mais profundo no grafo sempre que possível
- As arestas são exploradas a partir do vértice v mais recentemente descoberto que ainda possui arestas não exploradas saindo dele

Busca em profundidade

- Quando todas as arestas adjacentes a *v* tiverem sido exploradas a busca anda para trás para explorar vértices que saem do vértice do qual *v* foi descoberto
- O algoritmo é a base para muitos outros algoritmos importantes, tais como verificação de grafos acíclicos, ordenação topológica e componentes fortemente conectados

Busca em profundidade

- Para acompanhar o progresso do algoritmo cada vértice é colorido de branco, cinza ou preto
- Todos os vértices são inicializados como brancos
- Quando um vértice é *descoberto* pela primeira vez ele torna-se cinza, e é tornado preto quando sua lista de adjacentes tenha sido completamente examinada

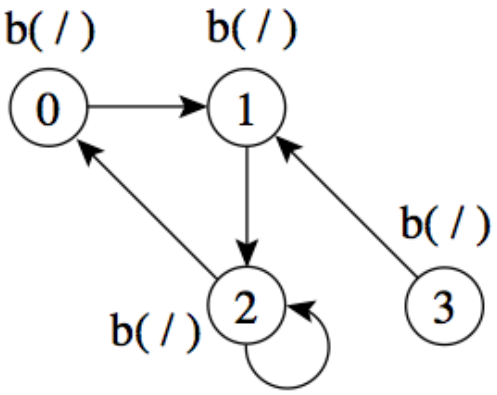
Busca em profundidade

- d[v]: tempo de descoberta
- t[v]: tempo de término do exame da lista de adjacentes de v
- Estes registros são inteiros entre 1 e 2|V| pois existe um evento de descoberta e um evento de término para cada um dos |V| vértices

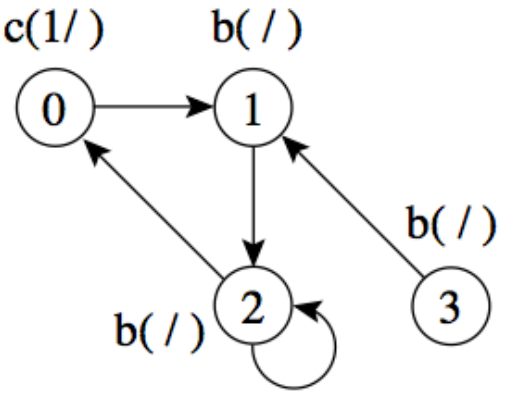
Algoritmo da busca em profundidade

- Entender algoritmo no código!

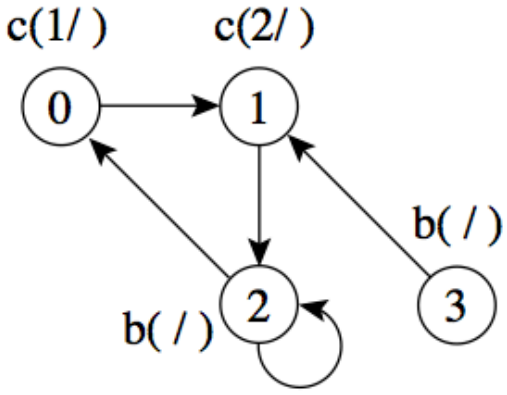
Exemplo



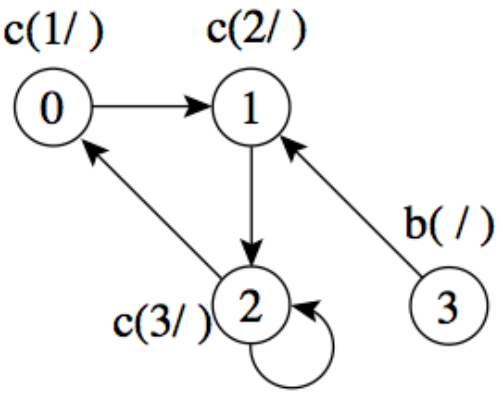
(a)



(b)

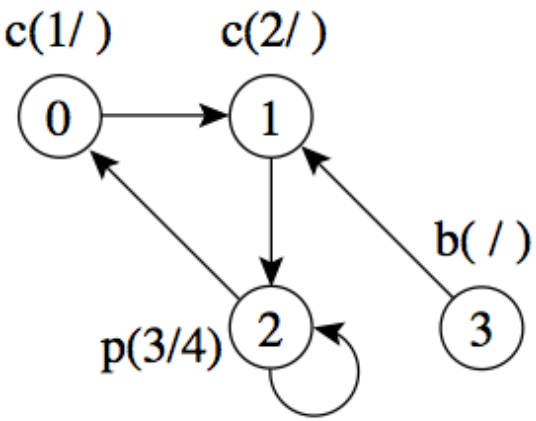


(c)

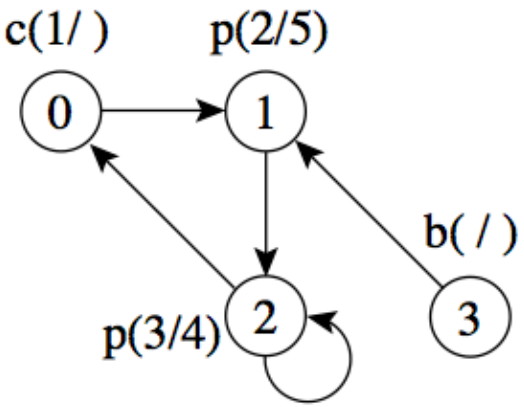


(d)

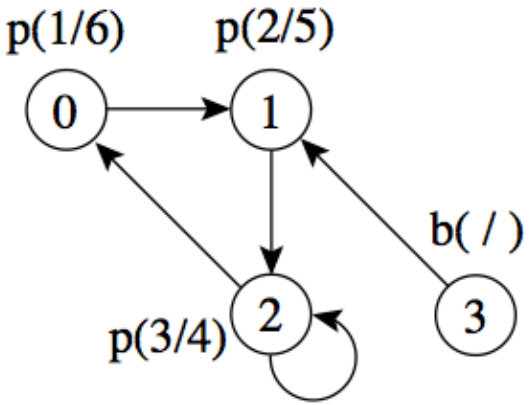
Exemplo (cont.)



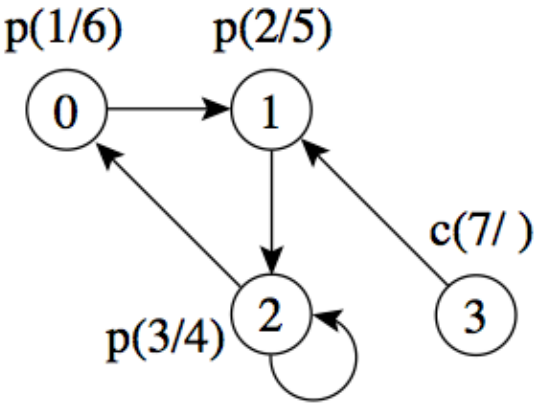
(e)



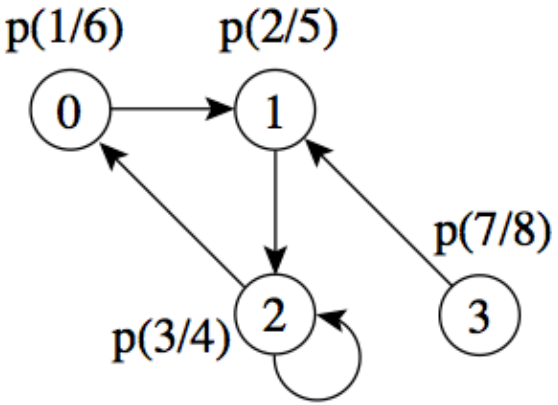
(f)



(g)



(h)



(i)

Classificação de arestas

- **Arestas de árvore**: são arestas de uma árvore de busca em profundidade. A aresta (u, v) é uma aresta de árvore se v foi descoberto pela primeira vez ao percorrer a aresta (u, v)
- **Arestas de retorno**: conectam um vértice u com um antecessor v em uma árvore de busca em profundidade (inclui *self-loops)*

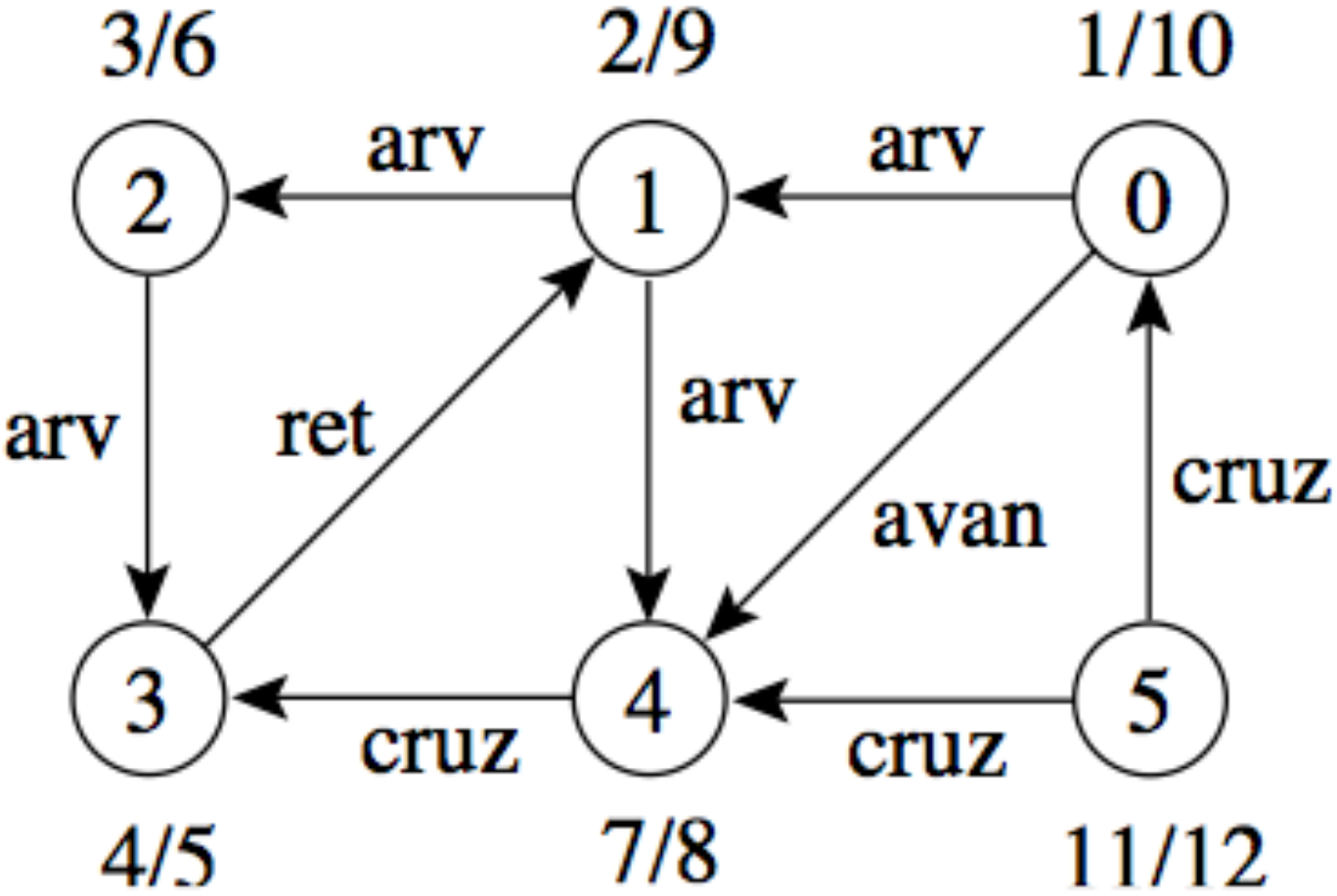
Classificação de arestas

- **Arestas de avanço**: não pertencem à árvore de busca em profundidade mas conectam um vértice a um descendente que pertence à árvore de busca em profundidade
- **Arestas de cruzamento**: podem conectar vértices na mesma árvore de busca em profundidade, ou em duas árvores diferentes

Classificando pela cor dos vértices

- Se o próximo vértice que eu encontrar for:
  - Branco, indica uma aresta de árvore.
  - Cinza, indica uma aresta de retorno.
  - Preto, indica uma aresta de avanço quando u é descoberto antes de v ou uma aresta de cruzamento caso contrário

Exemplo


3/6
2/9
1/10
2
arv
1
arv
0
arv
ret
arv
avan
cruz
3
4
5
cruz
cruz
4/5
7/8
11/12

Verificando se um grafo é acíclico

- A busca em profundidade pode ser usada para verificar se um grafo é acíclico ou contém um ou mais ciclos
- Se uma aresta de retorno é encontrada durante a busca em profundidade em G, então o grafo tem ciclo
- Um grafo direcionado G é acíclico se e somente se a busca em profundidade em G não apresentar arestas de retorno

Busca em largura

- Expande a fronteira entre vértices descobertos e não descobertos uniformemente através da largura da fronteira
- O algoritmo descobre todos os vértices a uma distância k do vértice origem antes de descobrir qualquer vértice a uma distância k + 1

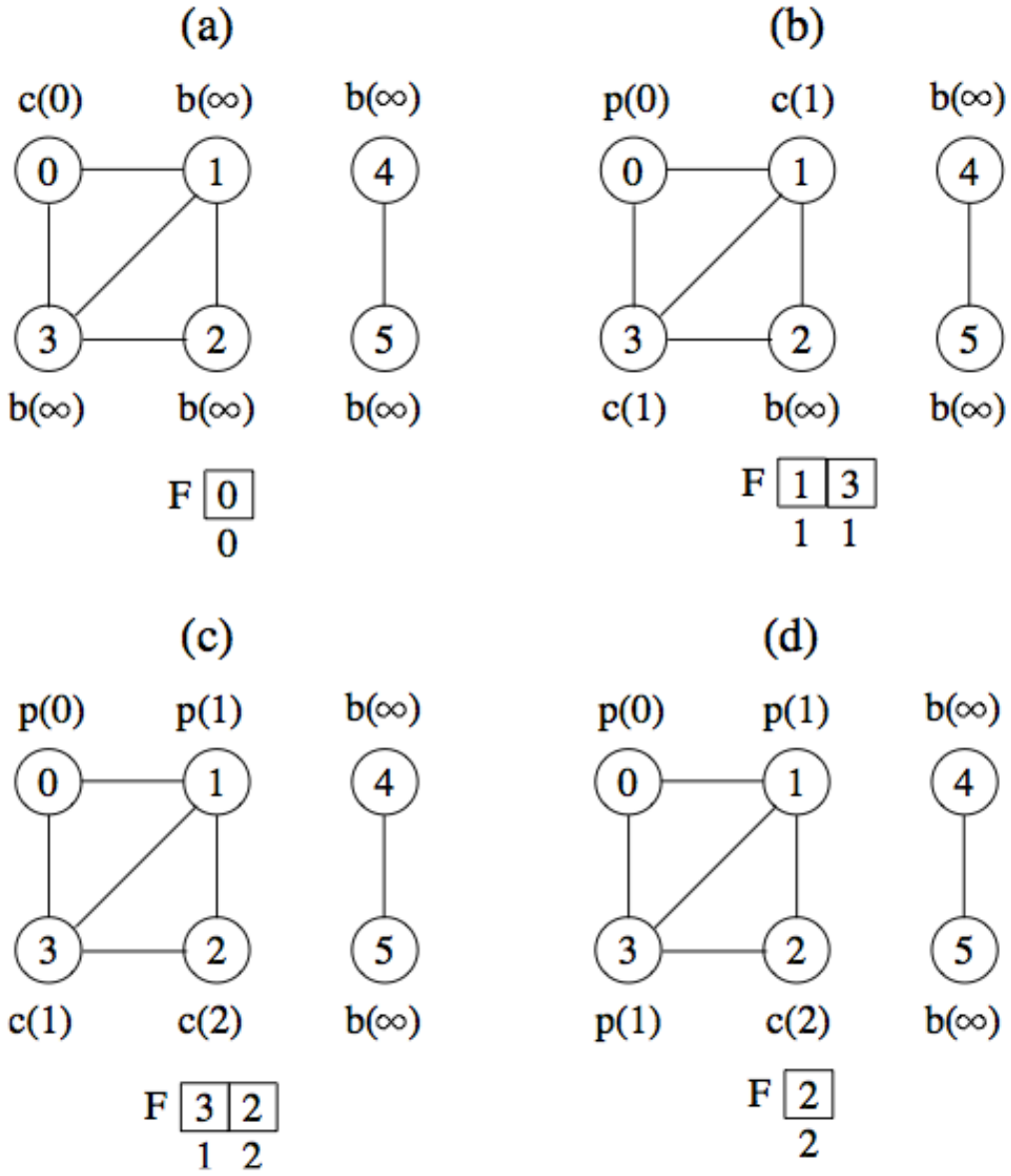
Busca em largura

- Cada vértice é colorido de branco, cinza ou preto
- Todos os vértices são inicializados branco
- Quando um vértice é descoberto pela primeira vez ele torna-se cinza
- Vértices cinza e preto já foram descobertos, mas são distinguidos para assegurar que a busca ocorra em largura

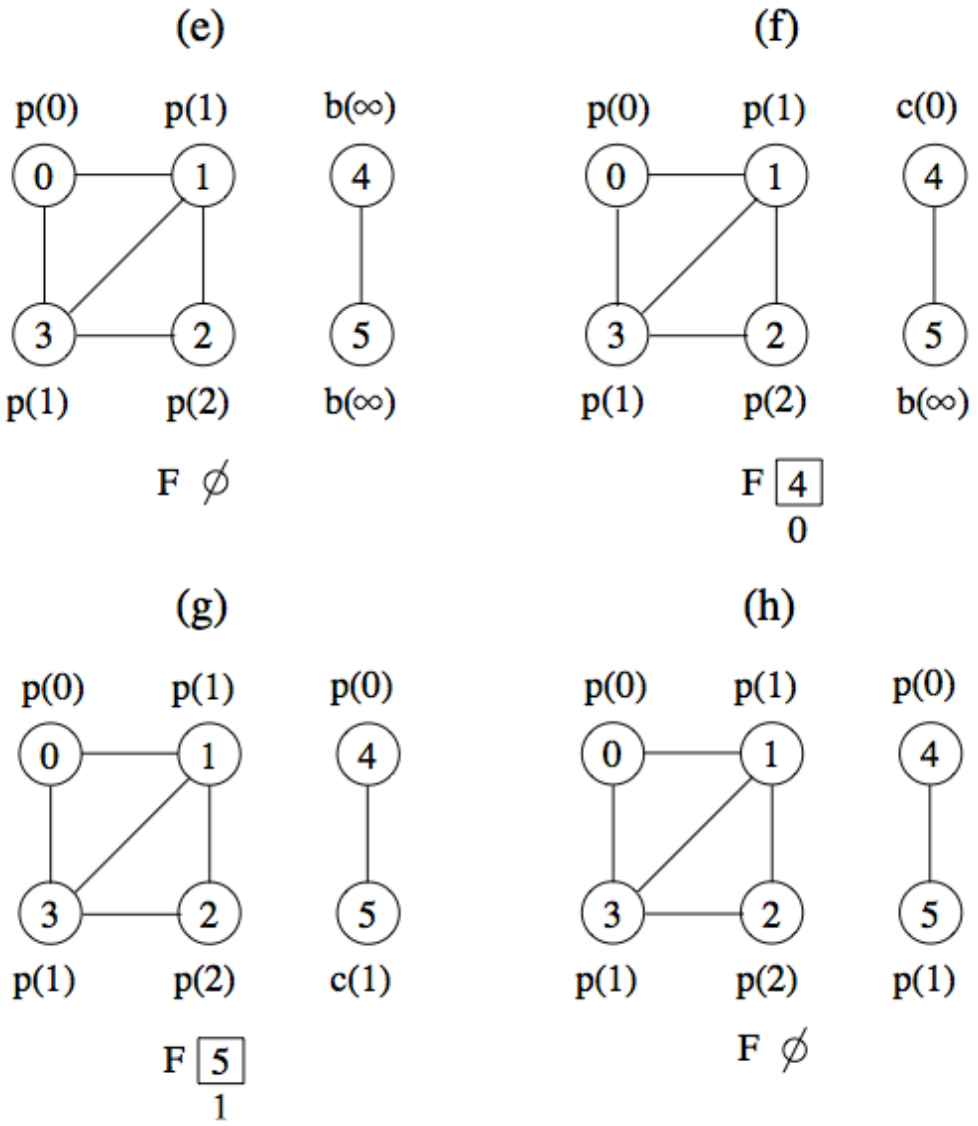
Busca em largura

- Se$(u,v) \in A$ e o vértice u é preto, então o vértice v tem que ser cinza ou preto
- Vértices cinza podem ter alguns vértices adjacentes brancos, e eles representam a fronteira entre vértices descobertos e não descobertos
- Entender o algoritmo no código!

Exemplo


(a)
c(0)
b(∞)
b(∞)
0
1
4
3
2
5
b(∞)
b(∞)
b(∞)
F 0
0
(b)
p(0)
c(1)
b(∞)
0
1
4
3
2
5
c(1)
b(∞)
b(∞)
F 1 3
1 1
(c)
p(0)
p(1)
b(∞)
0
1
4
3
2
5
c(1)
c(2)
b(∞)
F 3 2
1 2
(d)
p(0)
p(1)
b(∞)
0
1
4
3
2
5
p(1)
c(2)
b(∞)
F 2
2

Exemplo


(e)
p(0)
p(1)
b(∞)
0
1
4
3
2
5
p(1)
p(2)
b(∞)
F ∅
(f)
p(0)
p(1)
c(0)
0
1
4
3
2
5
p(1)
p(2)
b(∞)
F 4
0
(g)
p(0)
p(1)
p(0)
0
1
4
3
2
5
p(1)
p(2)
c(1)
F 5
1
(h)
p(0)
p(1)
p(0)
0
1
4
3
2
5
p(1)
p(2)
p(1)
F ∅

Caminhos mais curtos

- A busca em largura obtém o **caminho mais curto** de u até v
- O procedimento *VisitaBfs* contrói uma árvore de busca em largura que é armazenada na variável antecessor

Imprimindo o menor caminho

- O programa abaixo imprime os vértices do caminho mais curto entre o vértice origem e outro vértice qualquer do grafo, a partir do vetor antecessor. obtido na busca em largura

```
public void imprimeCaminho (int origem, int v) {
  if (origem == v) System.out.println (origem);
  else if (this.antecessor[v] == -1)
    System.out.println ("Nao existe caminho de " + origem + " ate " + v);
  else {
    imprimeCaminho (origem, this.antecessor[v]);
    System.out.println (v);
  }
}
```