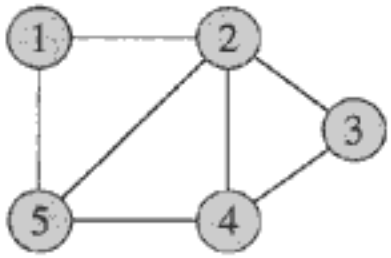
Capítulo 22: Algoritmos elementares de grafos

- Representações de grafos
- Busca em largura
- Busca em profundidade
- Ordenação topológica
- Componentes fortemente conectados

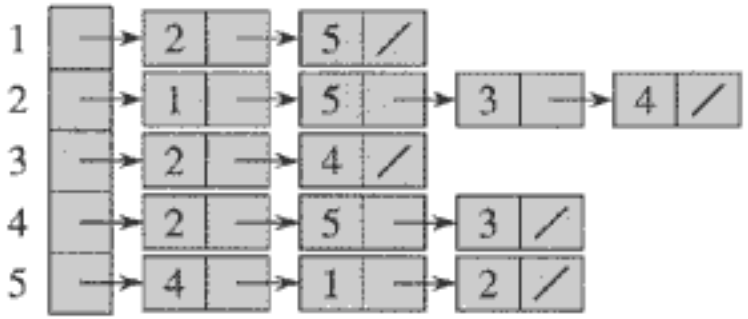
Representações de grafos

- Considere grafo G = (V, E)
- Dois tipos de estruturas de dados para representar G
  - Lista de adjacências
    - Adequada para representar grafo **esparso** (|E| é muito menor que $|V|^2$)
    - Quantidade de memória exigida: $\Theta(V+E)$
  - Matriz de adjacências
    - Adequada para representar grafo **denso** (|E| é próximo de $|V|^2$)
    - Também é mais eficiente para verificar se há aresta entre dois vértices
    - Quantidade de memória exigida: $\Theta(V^2)$

Representações de grafos



(a)



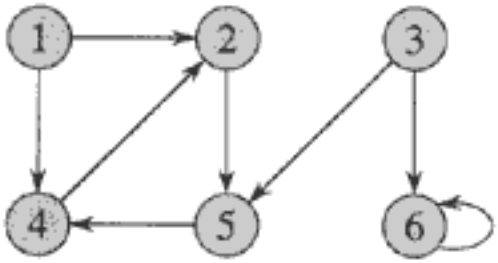
(b)

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

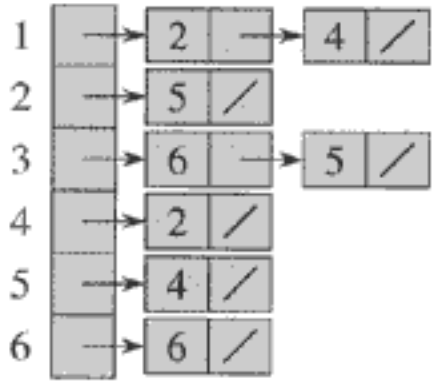
(c)

Figure 22.1 Two representations of an undirected graph. **(a)** An undirected graph G having five vertices and seven edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

Representações de grafos



(a)



(b)

	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

(c)

Figure 22.2 Two representations of a directed graph. **(a)** A directed graph G having six vertices and eight edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

Busca em largura

- BFS: algoritmo simples de pesquisa em grafo
  - Algoritmos de Dijkstra (caminhos mínimos) e Prim (AGM) usam idéias parecidas
- Características do algoritmo:
  - Entrada: G=(V, E) e vértice de origem s
  - Descobre cada vértice em G acessível a partir de s
  - Calcula distância (número de arestas) de s até demais vértices
  - Vértices com distância k de s são descobertos antes dos vértices com distância k+1
  - Produz “árvore primeiro na extensão” : raiz em s, caminho de s a demais vértices na árovre é mínimo

Busca em largura

- Vértices são pintados de branco, cinza ou preto durante busca
  1. Inicialmente todos os vértices são brancos: ainda não descobertos
  2. Vértice descoberto (primeira vez que é encontrado) durante busca é pintado de cinza: adjacência ainda não foi totalmente explorada
  3. Vértice com adjacência totalmente explorada é pintado de preto
- cor[u]: guarda a cor atual do vértice
- d[u]: guarda distância de s a u
- π[u]: guarda predecessor de u (quem descobriu u)

```
BFS(G, s)
 1  for each vertex u ∈ V[G] − {s}
 2       do color[u] ← WHITE
 3          d[u] ← ∞
 4          π[u] ← NIL
 5  color[s] ← GRAY
 6  d[s] ← 0
 7  π[s] ← NIL
 8  Q ← ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11       do u ← DEQUEUE(Q)
12          for each v ∈ Adj[u]
13               do if color[v] = WHITE
14                    then color[v] ← GRAY
15                         d[v] ← d[u] + 1
16                         π[v] ← u
17                         ENQUEUE(Q, v)
18          color[u] ← BLACK
```

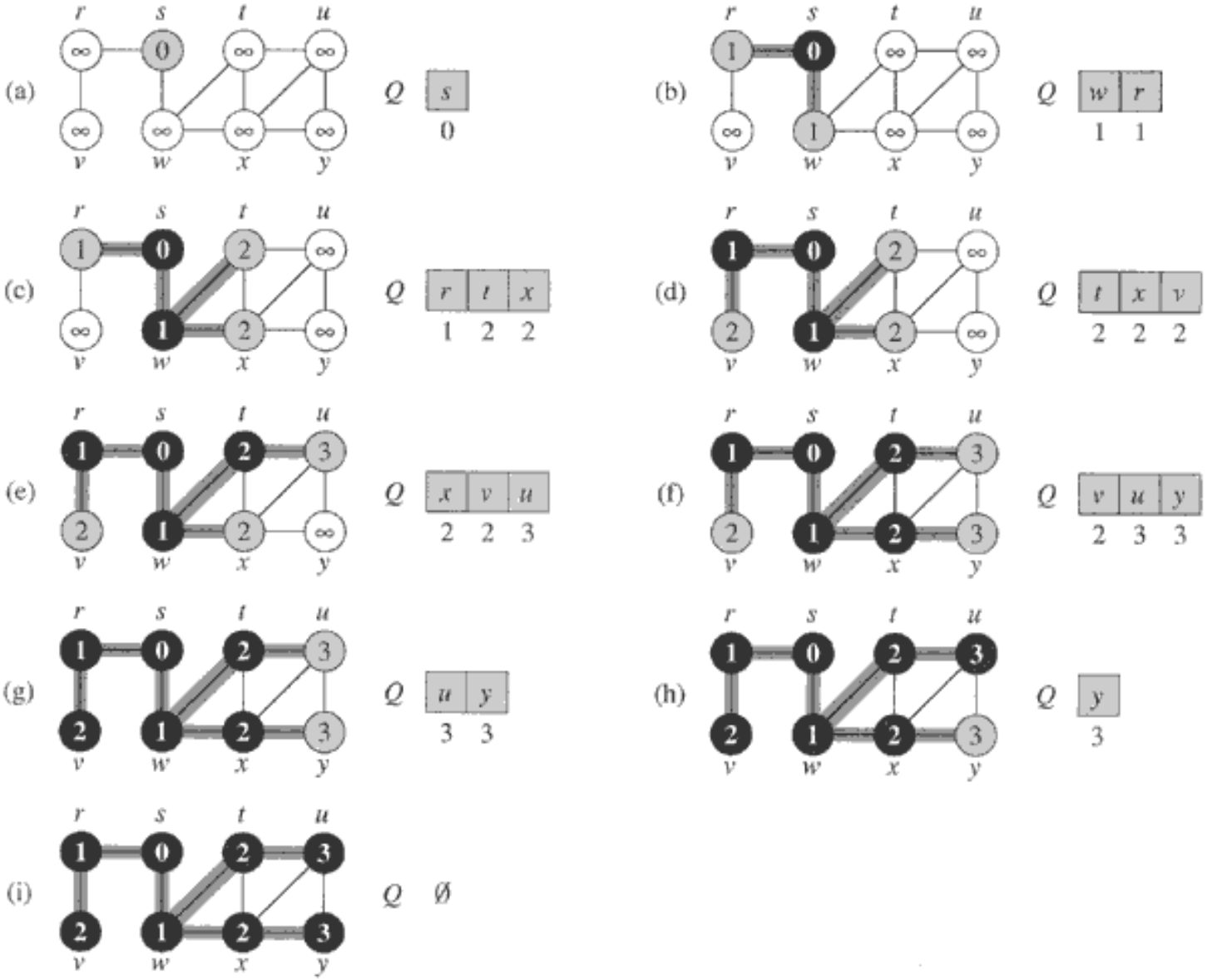


Figure 22.3 The operation of BFS on an undirected graph. Tree edges are shown shaded as they are produced by BFS. Within each vertex u is shown $d[u]$. The queue Q is shown at the beginning of each iteration of the **while** loop of lines 10–18. Vertex distances are shown next to vertices in the queue.

Busca em largura

- Análise
  - Linhas 1 a 4: $\Theta(V)$
  - Linhas 5 a 9: $\Theta(1)$
  - Loop while das linhas 10 a 18:
    - **Observação importante**: Cada vértice é colocado no máximo uma vez na fila (teste linha 13)
    - Linhas 10 e 11: $\Theta(V)$
    - Linhas 12 a 17: a adjacência de cada vértice é examinada no máximo uma vez. O tamanho de todas as listas de adjacências é $\Theta(E)$, resultando no custo $\Theta(E)$
    - Linhas 18: $\Theta(V)$
  - Custo total: $\Theta(V+E)$
    - Linear no tamanho da lista de adjacências

Árvores primeiro na extensão

- Após BFS ser executado
  - Procedimento PRINT-PATH imprime menor caminho de s até v

```
PRINT-PATH(G, s, v)
1  if v = s
2     then print s
3     else if π[v] = NIL
4             then print "no path from" s "to" v "exists"
5             else PRINT-PATH(G, s, π[v])
6                  print v
```

Busca em profundidade

- DFS: algoritmo de busca utilizado por outros algoritmos importantes (ex.: ordenação topológica, CFC)
- Características do algoritmo
  - Entrada: G=(V, E)
  - Realiza busca "mais profunda" sempre que possível
    1. Arestas são exploradas a partir de vértice v mais recentemente descoberto
    2. Quando todas arestas de v são exploradas, busca retorna para terminar a exploração de demais arestas do vértice que descobriu v
  - Busca é feita a partir de diversas origens (diferente do BFS)
    - DFS gera floresta "primeiro na profundidade"
      - Composta por diversas árvores primeiro na profunidade

Busca em profundidade

- Similaridades entre DFS e BFS
  - Vértices também são pintados de branco, cinza ou preto
  - É calculado o predecessor de cada vértice
- Diferenças DFS e BFS
  - BFS faz busca a partir de uma origem: DFS faz a partir de várias
  - BFS explora todos os vértices mais próximos primeiro: DFS não
  - DFS não calcula distância a partir de uma origem
  - BFS usa fila para descoberta e DFS uma pilha (recursão)
  - DFS registra tempo de descoberta d[u] e término f[u] de cada vértice
    - d[u]: tempo em que u foi descoberto
    - f[u]: tempo em que foi terminada a exploração da adjacência de u (u é pintado de preto)

```
DFS(G)
1  for each vertex u ∈ V[G]
2       do color[u] ← WHITE
3          π[u] ← NIL
4  time ← 0
5  for each vertex u ∈ V[G]
6       do if color[u] = WHITE
7          then DFS-VISIT(u)
```

```
DFS-VISIT(u)
1  color[u] ← GRAY          ▷ White vertex u has just been discovered.
2  time ← time + 1
3  d[u] ← time
4  for each v ∈ Adj[u]      ▷ Explore edge (u, v).
5       do if color[v] = WHITE
6          then π[v] ← u
7               DFS-VISIT(v)
8  color[u] ← BLACK         ▷ Blacken u; it is finished.
9  f[u] ← time ← time + 1
```

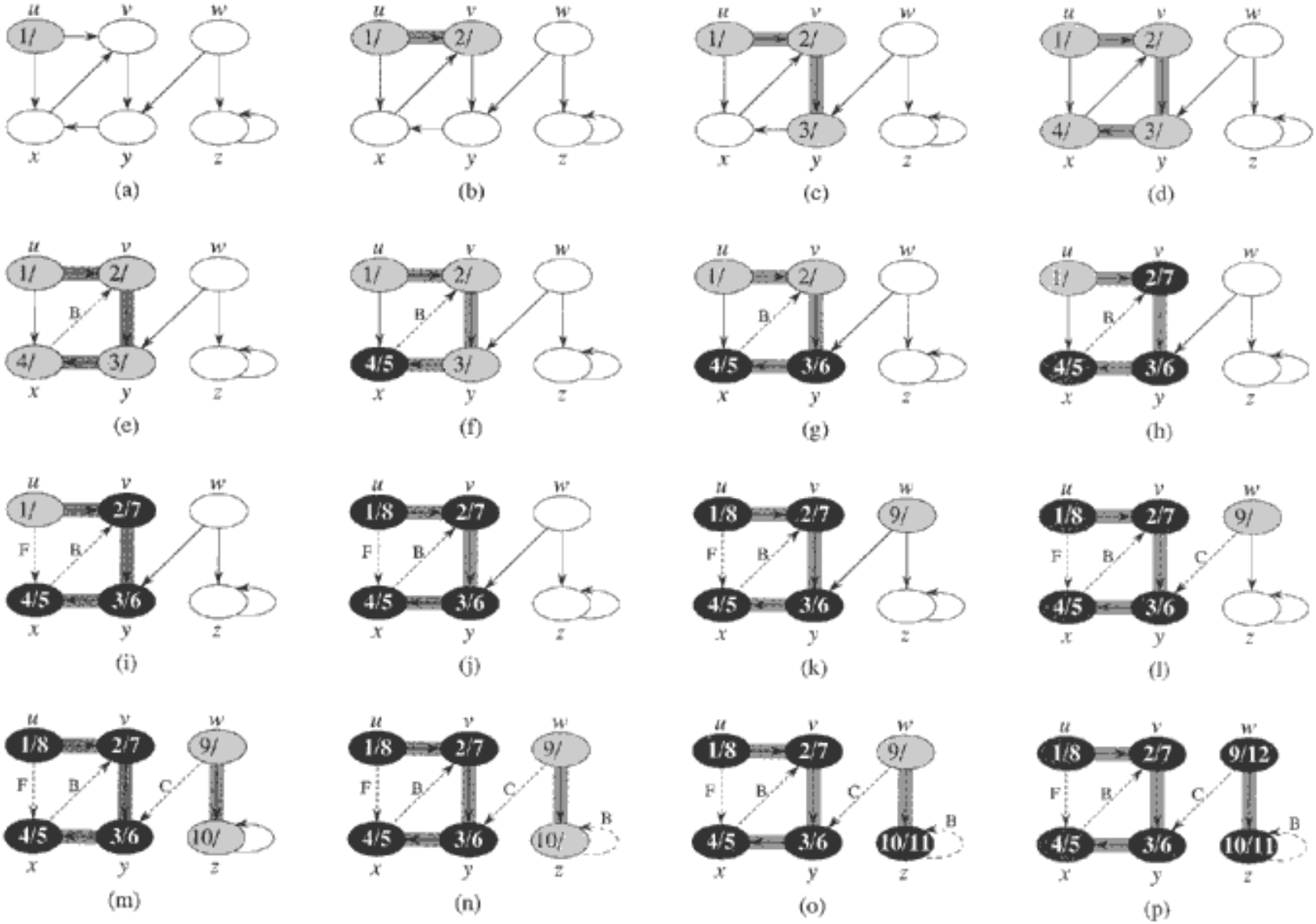


Figure 22.4 The progress of the depth-first-search algorithm DFS on a directed graph. As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise). Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges. Vertices are timestamped by discovery time/finishing time.

Busca em profundidade

- Análise
  - DFS
    - Linhas 1 a 3: $\Theta(V)$
    - Linha 4: $\Theta(1)$
    - Linhas 5 e 6: $\Theta(V)$
    - Linha 7: chamada só é feita se vértice for branco
      - No máximo $\Theta(V)$ chamadas a partir de DFS. Nesse caso…
  - DFS-Visit é chamado apenas uma vez para cada vértice
    - Linhas 1 a 3: $\Theta(V)$ custo total
    - Linhas 4 a 7: $\Theta(E)$ custo total
    - Linhas 8 e 9: $\Theta(V)$ custo total
  - Custo total de DFS: $\Theta(V+E)$

Busca em profundidade

- **Propriedade dos parênteses**: em qualquer busca em profundidade em G=(V,E), para quaisquer dois vértices u e v, exatamente uma das três condições é válida:
  1. Os intervalos [d[u],f[u]] e [d[v],f[v]] são completamente disjuntos
     - Nem u e nem v é descendente um do outro na floresta gerada
  2. O intervalo [d[u],f[u]] está inteiramente dentro de [d[v],f[v]]
     - u é descendente de v em uma árvore primeiro na profundidade
  3. O intervalo [d[v],f[v]] está inteiramente dentro de [d[u],f[u]]
     - v é descendente de u em uma árvore primeiro na profundidade

Busca em profundidade

- Classificação de arestas: dada a floresta primeiro na profunidade, há quatro tipos de arestas (u, v)
  - Aresta da árvore: v foi descoberto pela exploração de u
  - Aresta de retorno: (u, v) conecta um vértice u a um ancestral v em uma árvore
  - Aresta diretas: (u, v) não pertence à árvore e conecta um vértice u a um descendente v em uma árvore
  - Arestas cruzadas: demais arestas. Podem estar entre vértices da mesma árvore (desde que um não seja ancestral do outro) ou entre vértices de árvores diferentes

Busca em profundidade

- DFS pode ser modificado para classificar arestas
  - Quando analisa aresta (u, v), se v é:
    - Branco: (u, v) é aresta da árvore
    - Cinza: (u, v) é aresta de retorno
    - Preto: (u, v) é aresta direta ou cruzada

Ordenação topológica

- Ordenação topológica de GAO
  - Ordenação linear dos vértices onde, se há aresta (u, v), então u aparece antes de v na ordenação
  - Ex. aplicação: verificar pré-requisistos de disciplinas na matrícula
  - Complexidade: $\Theta(V+E)$

TOPOLOGICAL-SORT(G)

1 call DFS(G) to compute finishing times $f[v]$ for each vertex v
2 as each vertex is finished, insert it onto the front of a linked list
3 **return** the linked list of vertices

Ordenação topológica

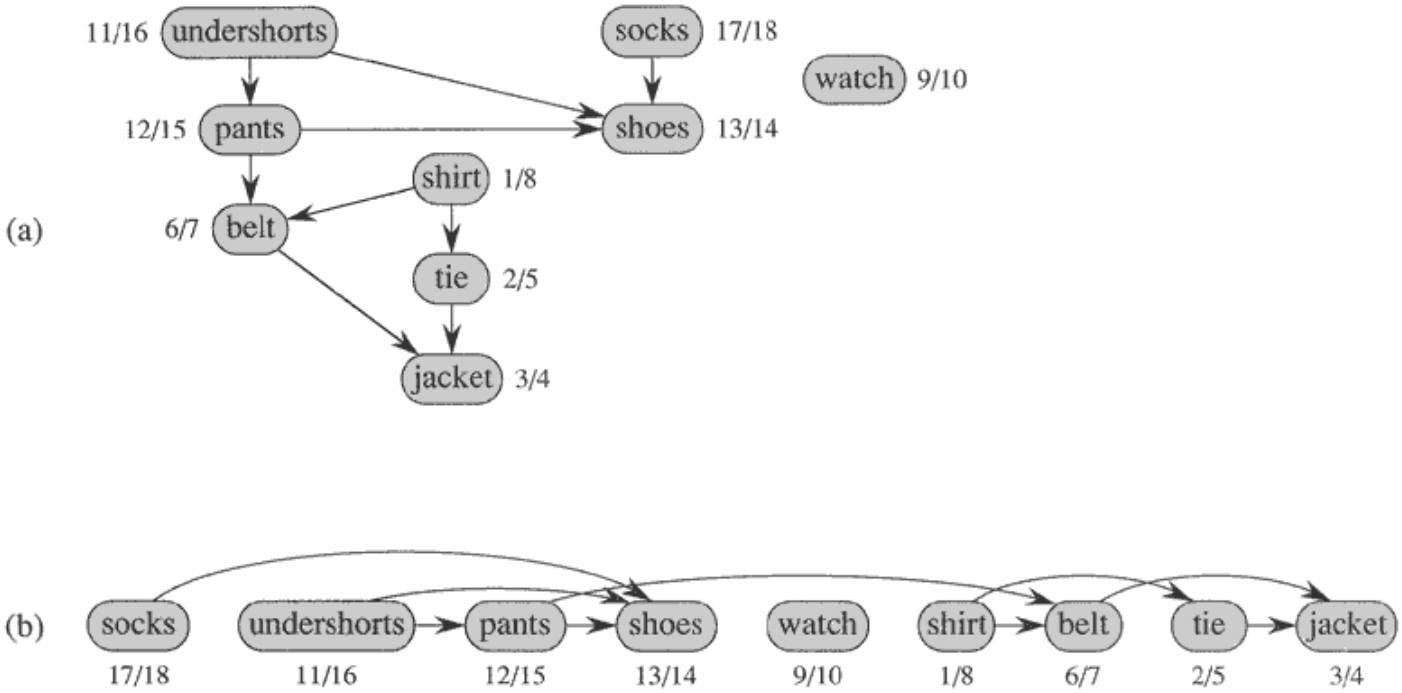


Figure 22.7 **(a)** Professor Bumstead topologically sorts his clothing when getting dressed. Each directed edge (u, v) means that garment u must be put on before garment v. The discovery and finishing times from a depth-first search are shown next to each vertex. **(b)** The same graph shown topologically sorted. Its vertices are arranged from left to right in order of decreasing finishing time. Note that all directed edges go from left to right.

Componentes fortemente conectados

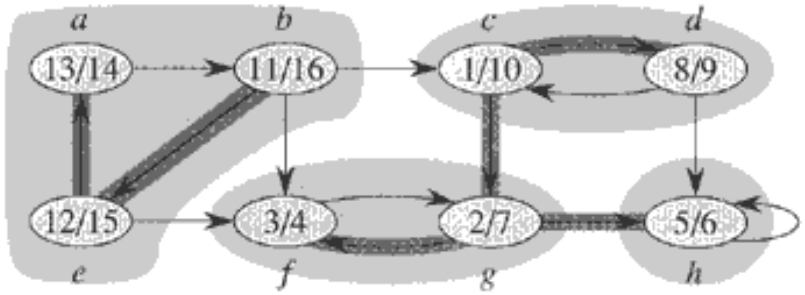
- Diversos algoritmos que trabalham com grafos orientados realizam a decomposição de um grafo em seus CFCs
  - Aplicações em compiladores, mineração de dados, etc.
- Complexidade do algoritmo CFC: $\Theta(V+E)$

STRONGLY-CONNECTED-COMPONENTS(G)

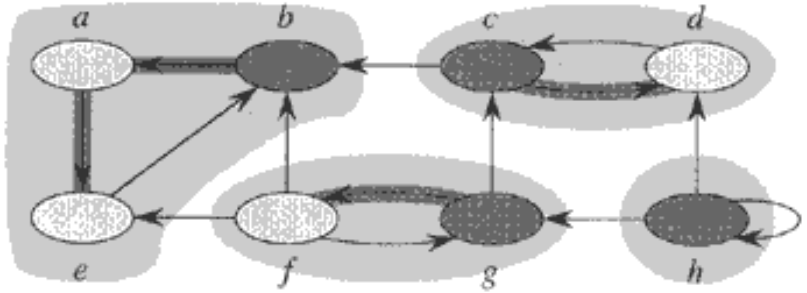
1 call DFS(G) to compute finishing times $f[u]$ for each vertex u
2 compute G^T
3 call DFS(G^T), but in the main loop of DFS, consider the vertices in order of decreasing $f[u]$ (as computed in line 1)
4 output the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component

Componentes fortemente conectados

(a)



(b)



(c)