Projeto e Análise de Algoritmos

Aplicações da Busca em Profundidade

Atilio G. Luiz

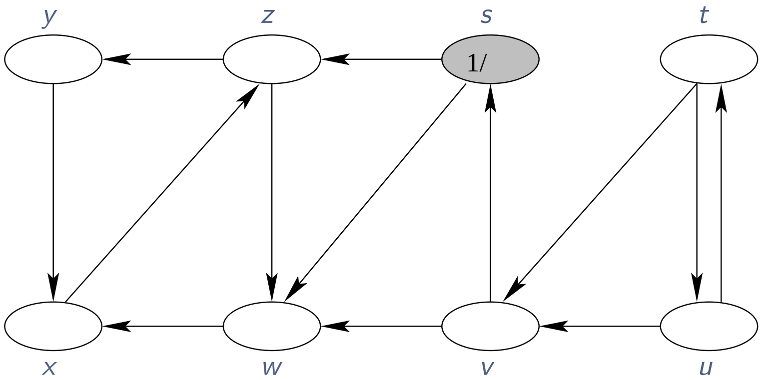
Primeiro Semestre de 2023

Ordenação Topológica

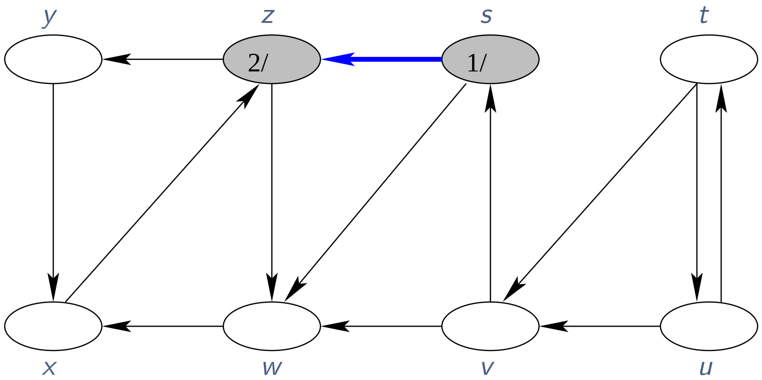
Antes de começar

Vamos rever o nosso exemplo de busca em profundidade em um grafo direcionado.

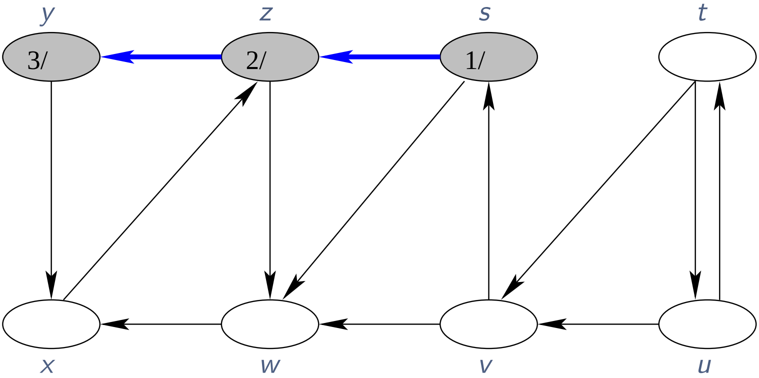
Exemplo de busca em profundidade



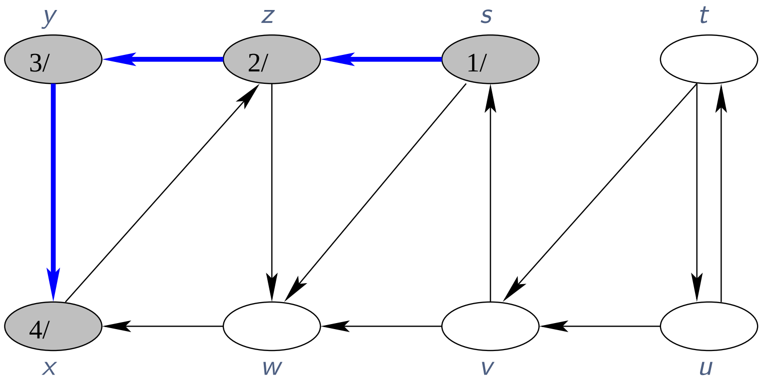
Exemplo de busca em profundidade



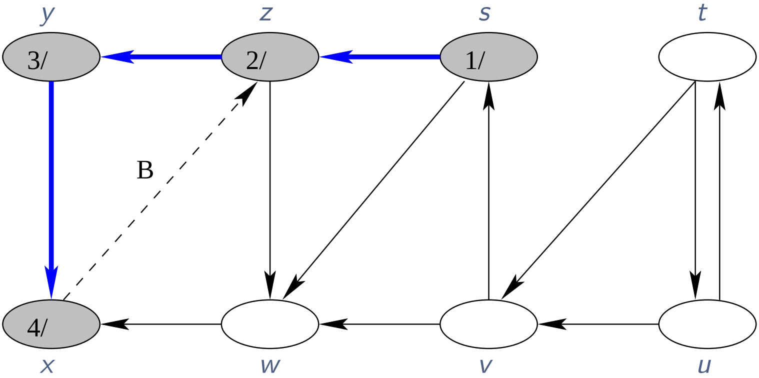
Exemplo de busca em profundidade



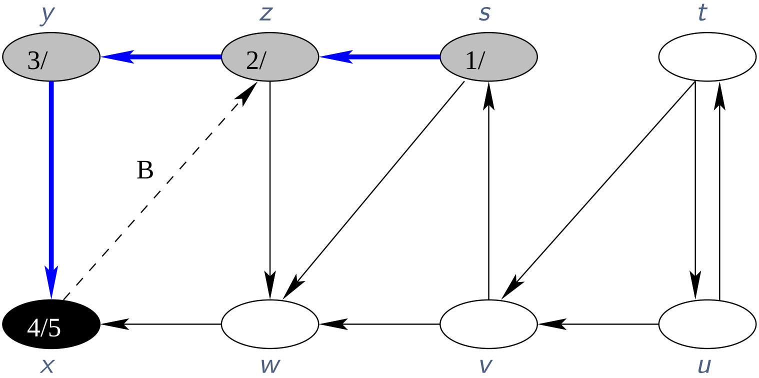
Exemplo de busca em profundidade



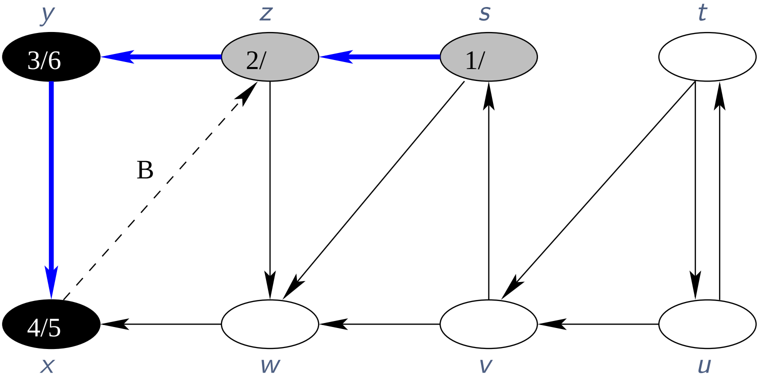
Exemplo de busca em profundidade



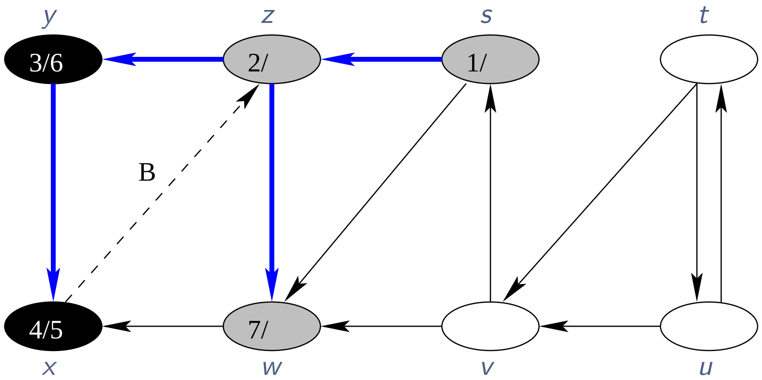
Exemplo de busca em profundidade



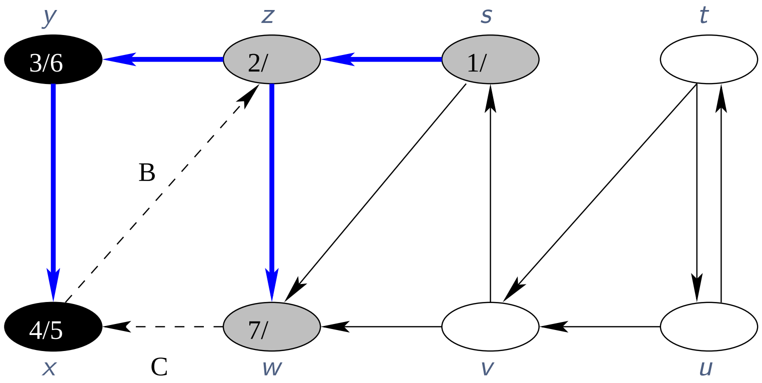
Exemplo de busca em profundidade



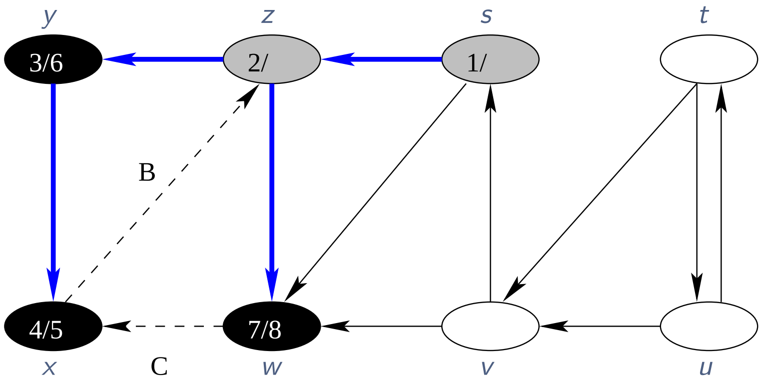
Exemplo de busca em profundidade



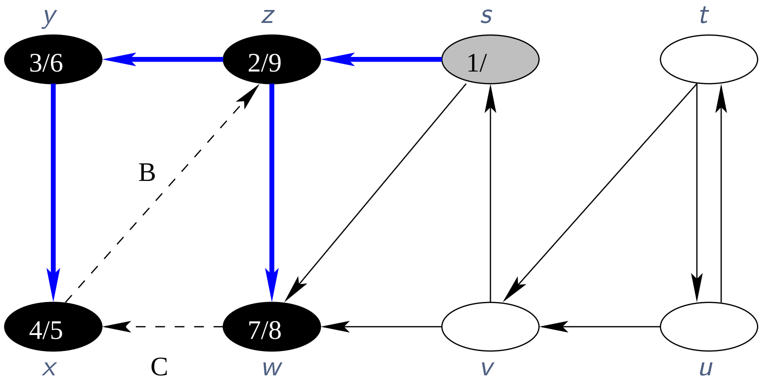
Exemplo de busca em profundidade



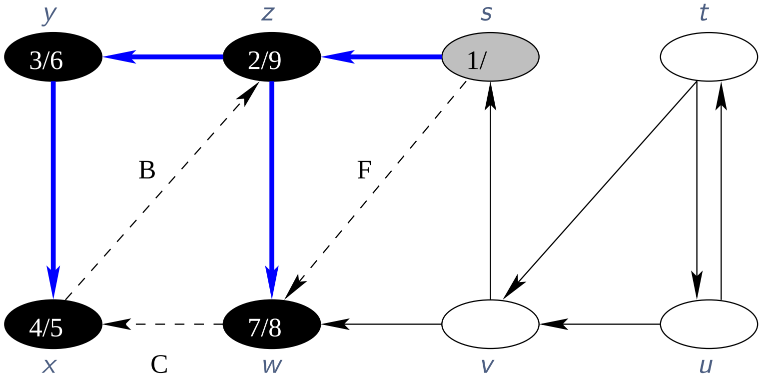
Exemplo de busca em profundidade



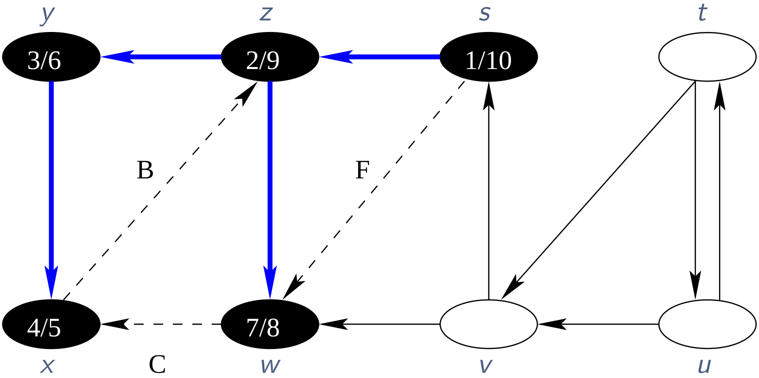
Exemplo de busca em profundidade



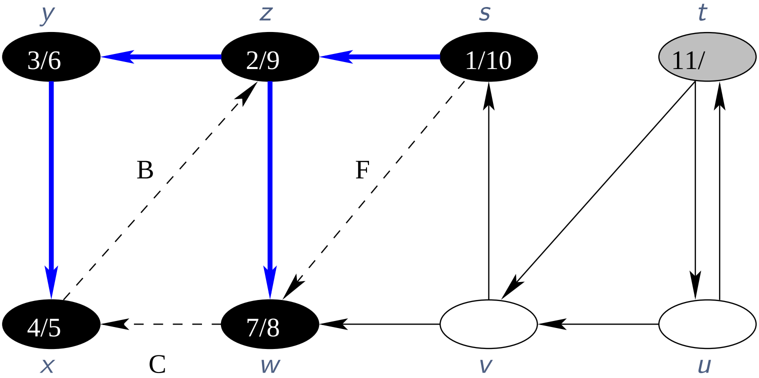
Exemplo de busca em profundidade



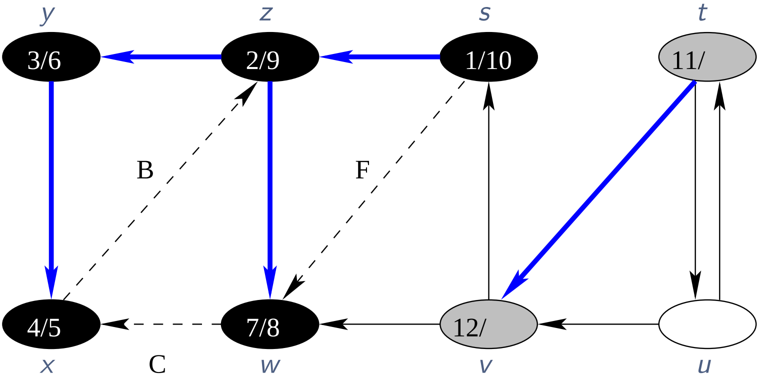
Exemplo de busca em profundidade



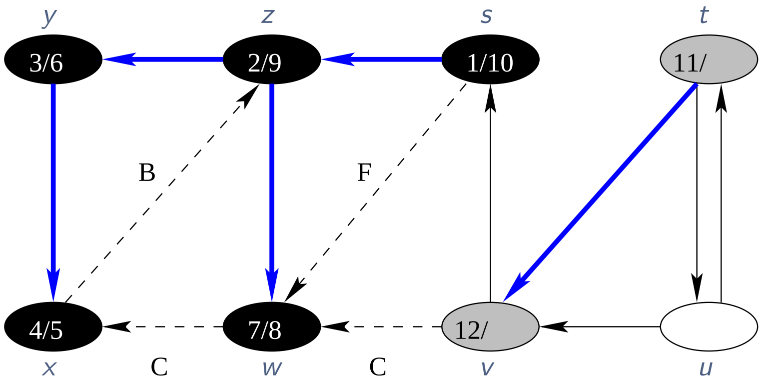
Exemplo de busca em profundidade



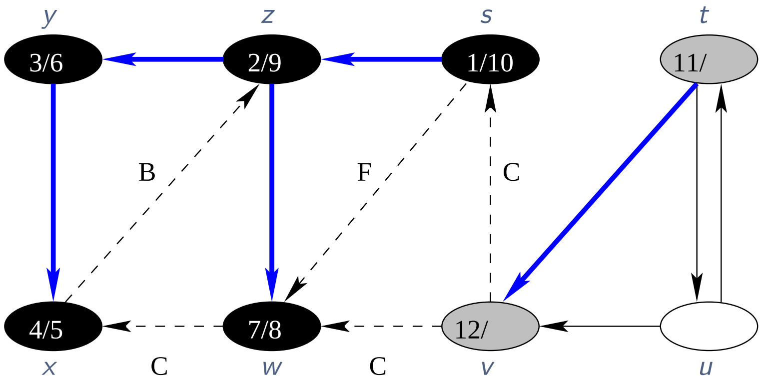
Exemplo de busca em profundidade



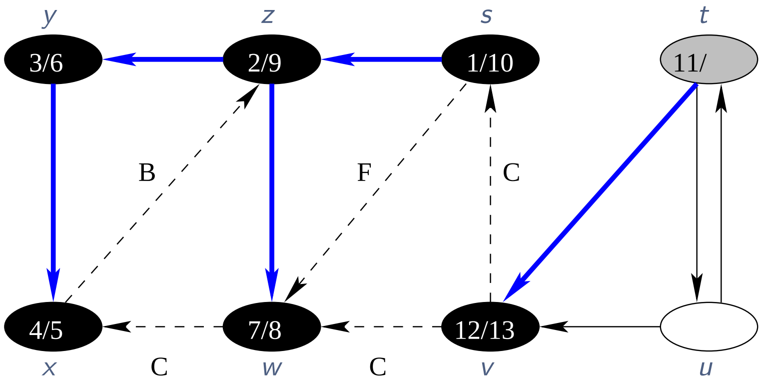
Exemplo de busca em profundidade



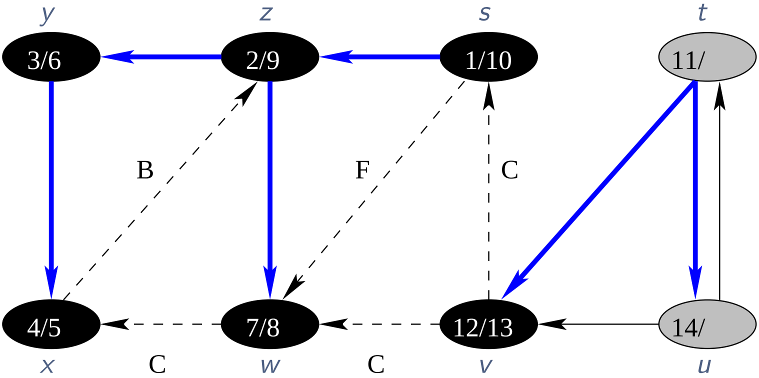
Exemplo de busca em profundidade



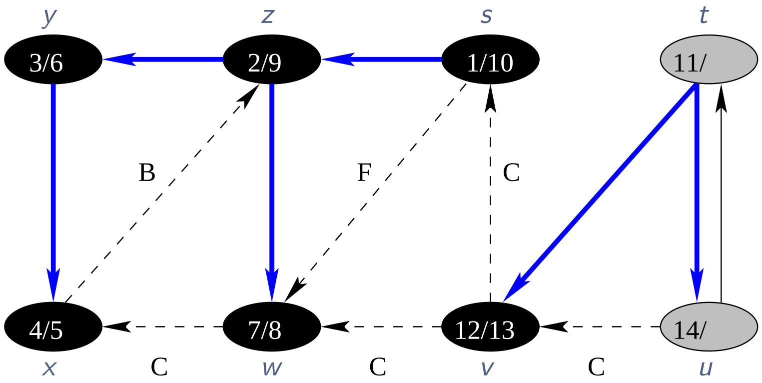
Exemplo de busca em profundidade



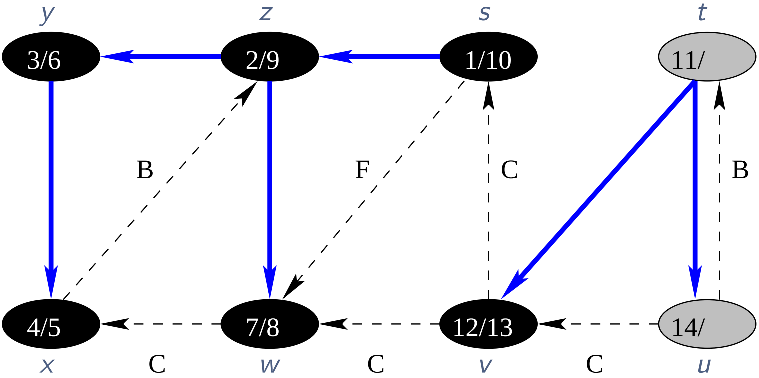
Exemplo de busca em profundidade



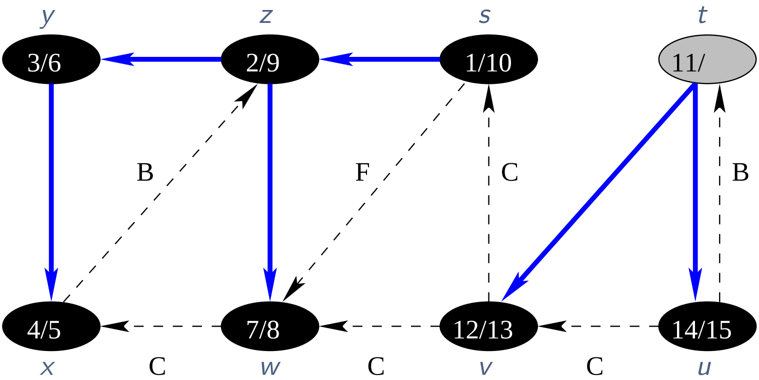
Exemplo de busca em profundidade



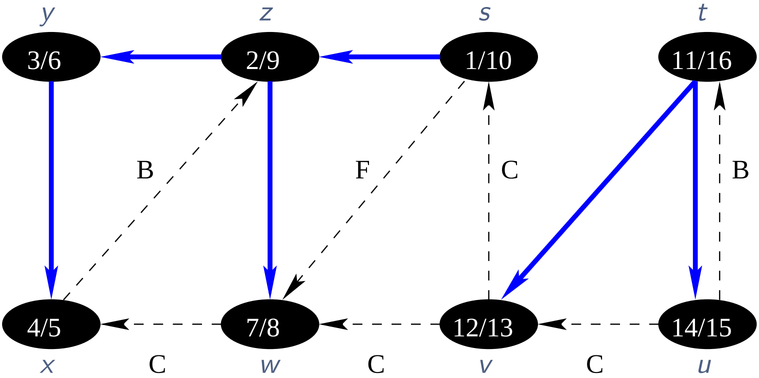
Exemplo de busca em profundidade



Exemplo de busca em profundidade

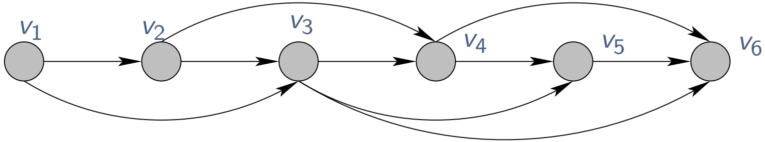


Exemplo de busca em profundidade



Ordenação Topológica

Uma ordenação topológica de um grafo direcionado G é uma ordenação dos vértices $v_1, v_2, \ldots, v_n$ tal que se (v_i, v_j) é uma aresta do grafo, então $i < j$.



Exemplo de aplicação

Representando dependências

- um grafo pode representar precedências entre tarefas
- queremos uma ordem que respeita as precedências

Exemplo de aplicação

Representando dependências

- um grafo pode representar precedências entre tarefas
- queremos uma ordem que respeita as precedências

Exemplo de aplicação

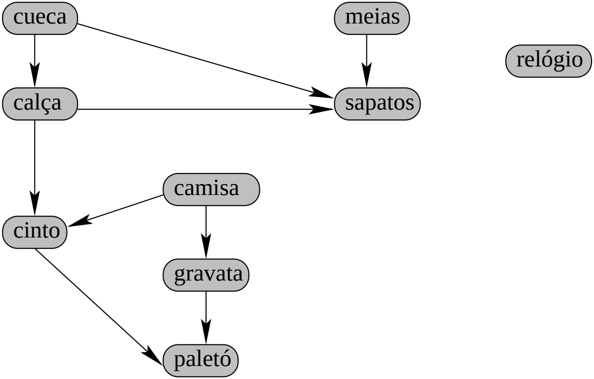
Representando dependências

- um grafo pode representar precedências entre tarefas
- queremos uma ordem que respeita as precedências

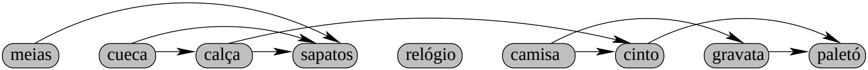
Exemplo de aplicação

Representando dependências

- um grafo pode representar precedências entre tarefas
- queremos uma ordem que respeita as precedências



Exemplo de ordenação topológica



Condições de existência

Todo grafo direcionado possui ordenação topológica?

- não, um ciclo direcionado não possui
- nem outro grafo que contém um ciclo

Um grafo direcionado é acíclico se não contiver um ciclo direcionado.

Condições de existência

Todo grafo direcionado possui ordenação topológica?

- não, um ciclo direcionado não possui
- nem outro grafo que contém um ciclo

Um grafo direcionado é acíclico se não contiver um ciclo direcionado.

Condições de existência

Todo grafo direcionado possui ordenação topológica?

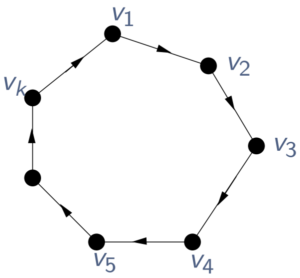
- não, um ciclo direcionado não possui
- nem outro grafo que contém um ciclo

Um grafo direcionado é acíclico se não contiver um ciclo direcionado.

Condições de existência

Todo grafo direcionado possui ordenação topológica?

- não, um ciclo direcionado não possui
- nem outro grafo que contém um ciclo

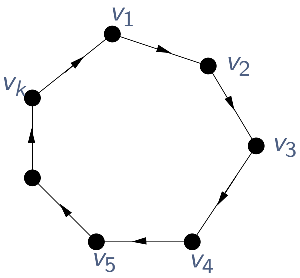


Um grafo direcionado é acíclico se não contiver um ciclo direcionado.

Condições de existência

Todo grafo direcionado possui ordenação topológica?

- não, um ciclo direcionado não possui
- nem outro grafo que contém um ciclo



Um grafo direcionado é acíclico se não contiver um ciclo direcionado.

Condições de existência

Teorema

Um grafo direcionado é acíclico se e somente se possui uma ordenação topológica.

Demonstração

- se G tem uma ordenação topológica, então ele é acíclico
- em seguida, vamos mostrar a recíproca

Condições de existência

Teorema

Um grafo direcionado é acíclico se e somente se possui uma ordenação topológica.

Demonstração

- se G tem uma ordenação topológica, então ele é acíclico
- em seguida, vamos mostrar a recíproca

Condições de existência

Teorema

Um grafo direcionado é acíclico se e somente se possui uma ordenação topológica.

Demonstração

- se G tem uma ordenação topológica, então ele é acíclico
- em seguida, vamos mostrar a recíproca

Condições de existência

Teorema

Um grafo direcionado é acíclico se e somente se possui uma ordenação topológica.

Demonstração

- se G tem uma ordenação topológica, então ele é acíclico
- em seguida, vamos mostrar a recíproca

Um lema auxiliar

- Uma fonte é um vértice com grau de entrada zero.
- Um sorvedouro é um vértice com grau de saída zero.

Lema

Todo grafo direcionado acíclico G com pelo menos um vértice possui uma fonte e um sorvedouro.

Demonstração

- tome um caminho mais longo P no grafo G que vai de s até t
- observe que s é uma fonte e t é um sorvedouro

Um lema auxiliar

- Uma fonte é um vértice com grau de entrada zero.
- Um sorvedouro é um vértice com grau de saída zero.

Lema

Todo grafo direcionado acíclico G com pelo menos um vértice possui uma fonte e um sorvedouro.

Demonstração

- tome um caminho mais longo P no grafo G que vai de s até t
- observe que s é uma fonte e t é um sorvedouro

Um lema auxiliar

- Uma fonte é um vértice com grau de entrada zero.
- Um sorvedouro é um vértice com grau de saída zero.

Lema

Todo grafo direcionado acíclico G com pelo menos um vértice possui uma fonte e um sorvedouro.

Demonstração

- tome um caminho mais longo P no grafo G que vai de s até t
- observe que s é uma fonte e t é um sorvedouro

Um lema auxiliar

- Uma fonte é um vértice com grau de entrada zero.
- Um sorvedouro é um vértice com grau de saída zero.

Lema

Todo grafo direcionado acíclico G com pelo menos um vértice possui uma fonte e um sorvedouro.

Demonstração

- tome um caminho mais longo P no grafo G que vai de s até t
- observe que s é uma fonte e t é um sorvedouro

Um lema auxiliar

- Uma fonte é um vértice com grau de entrada zero.
- Um sorvedouro é um vértice com grau de saída zero.

Lema

Todo grafo direcionado acíclico G com pelo menos um vértice possui uma fonte e um sorvedouro.

Demonstração

- tome um caminho mais longo P no grafo G que vai de s até t
- observe que s é uma fonte e t é um sorvedouro

Um lema auxiliar

- Uma fonte é um vértice com grau de entrada zero.
- Um sorvedouro é um vértice com grau de saída zero.

Lema

Todo grafo direcionado acíclico G com pelo menos um vértice possui uma fonte e um sorvedouro.

Demonstração

- tome um caminho mais longo P no grafo G que vai de s até t
- observe que s é uma fonte e t é um sorvedouro

Prova do teorema

Agora podemos terminar a demonstração

- considere um grafo direcionado acíclico $G = (V, E)$
- afirmamos que G possui uma ordenação topológica
- vamos mostrar por indução em $|V|$
- se $|V| = 1$, então a afirmação é clara

Considere um grafo com pelo menos dois vértices

- pelo lema anterior, G possui uma fonte v_1
- pela hipótese de indução, o grafo $G - v_1$ possui uma ordenação topológica $v_2, \dots, v_n$
- logo $v_1, v_2, \dots, v_n$ é uma ordenação topológica de G

Prova do teorema

Agora podemos terminar a demonstração

- considere um grafo direcionado acíclico $G = (V, E)$
- afirmamos que G possui uma ordenação topológica
- vamos mostrar por indução em $|V|$
- se $|V| = 1$, então a afirmação é clara

Considere um grafo com pelo menos dois vértices

- pelo lema anterior, G possui uma fonte v_1
- pela hipótese de indução, o grafo $G - v_1$ possui uma ordenação topológica $v_2, \dots, v_n$
- logo $v_1, v_2, \dots, v_n$ é uma ordenação topológica de G

Prova do teorema

Agora podemos terminar a demonstração

- considere um grafo direcionado acíclico $G = (V, E)$
- afirmamos que G possui uma ordenação topológica
- vamos mostrar por indução em $|V|$
- se $|V| = 1$, então a afirmação é clara

Considere um grafo com pelo menos dois vértices

- pelo lema anterior, G possui uma fonte v_1
- pela hipótese de indução, o grafo $G - v_1$ possui uma ordenação topológica $v_2, \ldots, v_n$
- logo $v_1, v_2, \ldots, v_n$ é uma ordenação topológica de G

Prova do teorema

Agora podemos terminar a demonstração

- considere um grafo direcionado acíclico $G = (V, E)$
- afirmamos que G possui uma ordenação topológica
- vamos mostrar por indução em $|V|$
- se $|V| = 1$, então a afirmação é clara

Considere um grafo com pelo menos dois vértices

- pelo lema anterior, G possui uma fonte v_1
- pela hipótese de indução, o grafo $G - v_1$ possui uma ordenação topológica $v_2, \ldots, v_n$
- logo $v_1, v_2, \ldots, v_n$ é uma ordenação topológica de G

Prova do teorema

Agora podemos terminar a demonstração

- considere um grafo direcionado acíclico $G = (V, E)$
- afirmamos que G possui uma ordenação topológica
- vamos mostrar por indução em $|V|$
- se $|V| = 1$, então a afirmação é clara

Considere um grafo com pelo menos dois vértices

- pelo lema anterior, G possui uma fonte v_1
- pela hipótese de indução, o grafo $G - v_1$ possui uma ordenação topológica $v_2, \dots, v_n$
- logo $v_1, v_2, \dots, v_n$ é uma ordenação topológica de G

Prova do teorema

Agora podemos terminar a demonstração

- considere um grafo direcionado acíclico $G = (V, E)$
- afirmamos que G possui uma ordenação topológica
- vamos mostrar por indução em $|V|$
- se $|V| = 1$, então a afirmação é clara

Considere um grafo com pelo menos dois vértices

- pelo lema anterior, G possui uma fonte v_1
- pela hipótese de indução, o grafo $G - v_1$ possui uma ordenação topológica $v_2, \ldots, v_n$
- logo $v_1, v_2, \ldots, v_n$ é uma ordenação topológica de G

Prova do teorema

Agora podemos terminar a demonstração

- considere um grafo direcionado acíclico $G = (V, E)$
- afirmamos que G possui uma ordenação topológica
- vamos mostrar por indução em $|V|$
- se $|V| = 1$, então a afirmação é clara

Considere um grafo com pelo menos dois vértices

- pelo lema anterior, G possui uma fonte v_1
- pela hipótese de indução, o grafo $G - v_1$ possui uma ordenação topológica $v_2, \ldots, v_n$
- logo $v_1, v_2, \ldots, v_n$ é uma ordenação topológica de G

Prova do teorema

Agora podemos terminar a demonstração

- considere um grafo direcionado acíclico $G = (V, E)$
- afirmamos que G possui uma ordenação topológica
- vamos mostrar por indução em $|V|$
- se $|V| = 1$, então a afirmação é clara

Considere um grafo com pelo menos dois vértices

- pelo lema anterior, G possui uma fonte v_1
- pela hipótese de indução, o grafo $G - v_1$ possui uma ordenação topológica $v_2, \ldots, v_n$
- logo $v_1, v_2, \ldots, v_n$ é uma ordenação topológica de G

Prova do teorema

Agora podemos terminar a demonstração

- considere um grafo direcionado acíclico $G = (V, E)$
- afirmamos que G possui uma ordenação topológica
- vamos mostrar por indução em $|V|$
- se $|V| = 1$, então a afirmação é clara

Considere um grafo com pelo menos dois vértices

- pelo lema anterior, G possui uma fonte v_1
- pela hipótese de indução, o grafo $G - v_1$ possui uma ordenação topológica $v_2, \ldots, v_n$
- logo $v_1, v_2, \ldots, v_n$ é uma ordenação topológica de G

Encontrando uma ordenação topológica

A demonstração anterior é construtiva

- é baseada em exibir uma ordenação topológica
- ela sugere um algoritmo recursivo ou iterativo

Algoritmo para ordenação topológica

1. encontre uma fonte v_1 de G
2. recursivamente ou iterativamente, obtenha uma ordenação $v_2, \ldots, v_n$ de $G - v_1$
3. devolva $v_1, v_2, \ldots, v_n$

- pode-se implementar esse algoritmo em tempo $O(V + E)$ (exercício 22.4-5 CLRS)

Encontrando uma ordenação topológica

A demonstração anterior é construtiva

- é baseada em exibir uma ordenação topológica
- ela sugere um algoritmo recursivo ou iterativo

Algoritmo para ordenação topológica

1. encontre uma fonte v_1 de G
2. recursivamente ou iterativamente, obtenha uma ordenação $v_2, \ldots, v_n$ de $G - v_1$
3. devolva $v_1, v_2, \ldots, v_n$

- pode-se implementar esse algoritmo em tempo $O(V + E)$ (exercício 22.4-5 CLRS)

Encontrando uma ordenação topológica

A demonstração anterior é construtiva

- é baseada em exibir uma ordenação topológica
- ela sugere um algoritmo recursivo ou iterativo

Algoritmo para ordenação topológica

1. encontre uma fonte v_1 de G
2. recursivamente ou iterativamente, obtenha uma ordenação $v_2, \ldots, v_n$ de $G - v_1$
3. devolva $v_1, v_2, \ldots, v_n$

- pode-se implementar esse algoritmo em tempo $O(V + E)$ (exercício 22.4-5 CLRS)

Encontrando uma ordenação topológica

A demonstração anterior é construtiva

- é baseada em exibir uma ordenação topológica
- ela sugere um algoritmo recursivo ou iterativo

Algoritmo para ordenação topológica

1. encontre uma fonte v_1 de G
2. recursivamente ou iterativamente, obtenha uma ordenação $v_2, \ldots, v_n$ de $G - v_1$
3. devolva $v_1, v_2, \ldots, v_n$

- pode-se implementar esse algoritmo em tempo $O(V + E)$ (exercício 22.4-5 CLRS)

Encontrando uma ordenação topológica

A demonstração anterior é construtiva

- é baseada em exibir uma ordenação topológica
- ela sugere um algoritmo recursivo ou iterativo

Algoritmo para ordenação topológica

1. encontre uma fonte v_1 de G
2. recursivamente ou iterativamente, obtenha uma ordenação $v_2, \ldots, v_n$ de $G - v_1$
3. devolva $v_1, v_2, \ldots, v_n$

- pode-se implementar esse algoritmo em tempo $O(V + E)$ (exercício 22.4-5 CLRS)

Encontrando uma ordenação topológica

A demonstração anterior é construtiva

- é baseada em exibir uma ordenação topológica
- ela sugere um algoritmo recursivo ou iterativo

Algoritmo para ordenação topológica

1. encontre uma fonte v_1 de G
2. recursivamente ou iterativamente, obtenha uma ordenação $v_2, \ldots, v_n$ de $G - v_1$
3. devolva $v_1, v_2, \ldots, v_n$

- pode-se implementar esse algoritmo em tempo $O(V + E)$ (exercício 22.4-5 CLRS)

Encontrando uma ordenação topológica

A demonstração anterior é construtiva

- é baseada em exibir uma ordenação topológica
- ela sugere um algoritmo recursivo ou iterativo

Algoritmo para ordenação topológica

1. encontre uma fonte v_1 de G
2. recursivamente ou iterativamente, obtenha uma ordenação $v_2, \dots, v_n$ de $G - v_1$
3. devolva $v_1, v_2, \dots, v_n$

- pode-se implementar esse algoritmo em tempo $O(V + E)$ (exercício 22.4-5 CLRS)

Encontrando uma ordenação topológica

A demonstração anterior é construtiva

- é baseada em exibir uma ordenação topológica
- ela sugere um algoritmo recursivo ou iterativo

Algoritmo para ordenação topológica

1. encontre uma fonte v_1 de G
2. recursivamente ou iterativamente, obtenha uma ordenação $v_2, \ldots, v_n$ de $G - v_1$
3. devolva $v_1, v_2, \ldots, v_n$

- pode-se implementar esse algoritmo em tempo $O(V + E)$ (exercício 22.4-5 CLRS)

Algoritmo baseado em DFS

Considere um grafo direcionado acíclico

- como não há ciclo, não existe aresta de retorno
- considere o instante em que v fica preto
- nesse instante todos seus vizinhos são pretos
- isso sugere considerar os vértices na ordem de término

Ideia para o algoritmo

- o primeiro vértice a ficar preto não tem arestas saindo
- o segundo só pode ter arestas para o primeiro
- o terceiro só pode ter arestas para os dois primeiros
- etc.

Algoritmo baseado em DFS

Considere um grafo direcionado acíclico

- como não há ciclo, não existe aresta de retorno
- considere o instante em que v fica preto
- nesse instante todos seus vizinhos são pretos
- isso sugere considerar os vértices na ordem de término

Ideia para o algoritmo

- o primeiro vértice a ficar preto não tem arestas saindo
- o segundo só pode ter arestas para o primeiro
- o terceiro só pode ter arestas para os dois primeiros
- etc.

Algoritmo baseado em DFS

Considere um grafo direcionado acíclico

- como não há ciclo, não existe aresta de retorno
- considere o instante em que v fica preto
- nesse instante todos seus vizinhos são pretos
- isso sugere considerar os vértices na ordem de término

Ideia para o algoritmo

- o primeiro vértice a ficar preto não tem arestas saindo
- o segundo só pode ter arestas para o primeiro
- o terceiro só pode ter arestas para os dois primeiros
- etc.

Algoritmo baseado em DFS

Considere um grafo direcionado acíclico

- como não há ciclo, não existe aresta de retorno
- considere o instante em que v fica preto
- nesse instante todos seus vizinhos são pretos
- isso sugere considerar os vértices na ordem de término

Ideia para o algoritmo

- o primeiro vértice a ficar preto não tem arestas saindo
- o segundo só pode ter arestas para o primeiro
- o terceiro só pode ter arestas para os dois primeiros
- etc.

Algoritmo baseado em DFS

Considere um grafo direcionado acíclico

- como não há ciclo, não existe aresta de retorno
- considere o instante em que v fica preto
- nesse instante todos seus vizinhos são pretos
- isso sugere considerar os vértices na ordem de término

Ideia para o algoritmo

- o primeiro vértice a ficar preto não tem arestas saindo
- o segundo só pode ter arestas para o primeiro
- o terceiro só pode ter arestas para os dois primeiros
- etc.

Algoritmo baseado em DFS

Considere um grafo direcionado acíclico

- como não há ciclo, não existe aresta de retorno
- considere o instante em que v fica preto
- nesse instante todos seus vizinhos são pretos
- isso sugere considerar os vértices na ordem de término

Ideia para o algoritmo

- o primeiro vértice a ficar preto não tem arestas saindo
- o segundo só pode ter arestas para o primeiro
- o terceiro só pode ter arestas para os dois primeiros
- etc.

Algoritmo baseado em DFS

Considere um grafo direcionado acíclico

- como não há ciclo, não existe aresta de retorno
- considere o instante em que v fica preto
- nesse instante todos seus vizinhos são pretos
- isso sugere considerar os vértices na ordem de término

Ideia para o algoritmo

- o primeiro vértice a ficar preto não tem arestas saindo
- o segundo só pode ter arestas para o primeiro
- o terceiro só pode ter arestas para os dois primeiros
- etc.

Algoritmo baseado em DFS

Considere um grafo direcionado acíclico

- como não há ciclo, não existe aresta de retorno
- considere o instante em que v fica preto
- nesse instante todos seus vizinhos são pretos
- isso sugere considerar os vértices na ordem de término

Ideia para o algoritmo

- o primeiro vértice a ficar preto não tem arestas saindo
- o segundo só pode ter arestas para o primeiro
- o terceiro só pode ter arestas para os dois primeiros
- etc.

Algoritmo baseado em DFS

Considere um grafo direcionado acíclico

- como não há ciclo, não existe aresta de retorno
- considere o instante em que v fica preto
- nesse instante todos seus vizinhos são pretos
- isso sugere considerar os vértices na ordem de término

Ideia para o algoritmo

- o primeiro vértice a ficar preto não tem arestas saindo
- o segundo só pode ter arestas para o primeiro
- o terceiro só pode ter arestas para os dois primeiros
- etc.

Algoritmo baseado em DFS

Considere um grafo direcionado acíclico

- como não há ciclo, não existe aresta de retorno
- considere o instante em que v fica preto
- nesse instante todos seus vizinhos são pretos
- isso sugere considerar os vértices na ordem de término

Ideia para o algoritmo

- o primeiro vértice a ficar preto não tem arestas saindo
- o segundo só pode ter arestas para o primeiro
- o terceiro só pode ter arestas para os dois primeiros
- etc.

Algoritmo Topological-Sort

```
Topological-Sort(G)
    execute DFS(G) e calcule f[v] para cada vértice v
    quando um vértice finalizar, insira-o no início de uma lista
    devolva a lista resultante
```

- inserir cada um dos $|V|$ vértices leva tempo $O(1)$
- além disso, executamos DFS uma vez
- portanto, a complexidade de tempo é $O(V + E)$

Algoritmo Topological-Sort

Topological-Sort(G)

execute $\mathrm{DFS}(G)$ e calcule $f[v]$ para cada vértice v
quando um vértice finalizar, insira-o no início de uma lista
devolva a lista resultante

- inserir cada um dos $|V|$ vértices leva tempo $O(1)$
- além disso, executamos DFS uma vez
- portanto, a complexidade de tempo é $O(V + E)$

Algoritmo Topological-Sort

Topological-Sort(G)
1 execute $\mathrm{DFS}(G)$ e calcule $f[v]$ para cada vértice v
2 quando um vértice finalizar, insira-o no início de uma lista
3 devolva a lista resultante

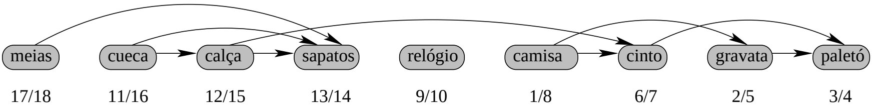
- inserir cada um dos $|V|$ vértices leva tempo $O(1)$
- além disso, executamos DFS uma vez
- portanto, a complexidade de tempo é $O(V + E)$

Algoritmo Topological-Sort

Topological-Sort(G)
1 execute $\mathrm{DFS}(G)$ e calcule $f[v]$ para cada vértice v
2 quando um vértice finalizar, insira-o no início de uma lista
3 devolva a lista resultante

- inserir cada um dos $|V|$ vértices leva tempo $O(1)$
- além disso, executamos DFS uma vez
- portanto, a complexidade de tempo é $O(V + E)$

Exemplo



meias
cueca
calça
sapatos
relógio
camisa
cinto
gravata
paletó
17/18
11/16
12/15
13/14
9/10
1/8
6/7
2/5
3/4

Correção

Teorema

TOPOLOGICAL-SORT(G) devolve ordenação topológica de G.

Demonstração

- a lista devolvida está em ordem decrescente de $f[v]$
- considere uma aresta arbitrária (u, v)
- basta mostrar que $f[u] > f[v]$
- considere o instante em que (u, v) foi examinada
- como (u, v) não é aresta de retorno, v não pode ser cinza
  1. se v for branco, ele será descendente de u e $f[u] > f[v]$
  2. se v for preto, então ele já foi finalizado e $f[u] > f[v]$
- em qualquer caso, obtemos o que desejávamos

Correção

Teorema

TOPOLOGICAL-SORT(G) devolve ordenação topológica de G.

Demonstração

- a lista devolvida está em ordem decrescente de $f[v]$
- considere uma aresta arbitrária (u, v)
- basta mostrar que $f[u] > f[v]$
- considere o instante em que (u, v) foi examinada
- como (u, v) não é aresta de retorno, v não pode ser cinza
  1. se v for branco, ele será descendente de u e $f[u] > f[v]$
  2. se v for preto, então ele já foi finalizado e $f[u] > f[v]$
- em qualquer caso, obtemos o que desejávamos

Correção

Teorema

TOPOLOGICAL-SORT(G) devolve ordenação topológica de G.

Demonstração

- a lista devolvida está em ordem decrescente de $f[v]$
- considere uma aresta arbitrária (u, v)
- basta mostrar que $f[u] > f[v]$
- considere o instante em que (u, v) foi examinada
- como (u, v) não é aresta de retorno, v não pode ser cinza
  1. se v for branco, ele será descendente de u e $f[u] > f[v]$
  2. se v for preto, então ele já foi finalizado e $f[u] > f[v]$
- em qualquer caso, obtemos o que desejávamos

Correção

Teorema

TOPOLOGICAL-SORT(G) devolve ordenação topológica de G.

Demonstração

- a lista devolvida está em ordem decrescente de $f[v]$
- considere uma aresta arbitrária (u, v)
- basta mostrar que $f[u] > f[v]$
- considere o instante em que (u, v) foi examinada
- como (u, v) não é aresta de retorno, v não pode ser cinza
  1. se v for branco, ele será descendente de u e $f[u] > f[v]$
  2. se v for preto, então ele já foi finalizado e $f[u] > f[v]$
- em qualquer caso, obtemos o que desejávamos

Correção

Teorema

TOPOLOGICAL-SORT(G) devolve ordenação topológica de G.

Demonstração

- a lista devolvida está em ordem decrescente de $f[v]$
- considere uma aresta arbitrária (u, v)
- basta mostrar que $f[u] > f[v]$
- considere o instante em que (u, v) foi examinada
- como (u, v) não é aresta de retorno, v não pode ser cinza
  1. se v for branco, ele será descendente de u e $f[u] > f[v]$
  2. se v for preto, então ele já foi finalizado e $f[u] > f[v]$
- em qualquer caso, obtemos o que desejávamos

Correção

Teorema

TOPOLOGICAL-SORT(G) devolve ordenação topológica de G.

Demonstração

- a lista devolvida está em ordem decrescente de $f[v]$
- considere uma aresta arbitrária (u, v)
- basta mostrar que $f[u] > f[v]$
- considere o instante em que (u, v) foi examinada
- como (u, v) não é aresta de retorno, v não pode ser cinza
  1. se v for branco, ele será descendente de u e $f[u] > f[v]$
  2. se v for preto, então ele já foi finalizado e $f[u] > f[v]$
- em qualquer caso, obtemos o que desejávamos

Correção

Teorema

$\textsc{Topological-Sort}(G)$ devolve ordenação topológica de G.

Demonstração

- a lista devolvida está em ordem decrescente de $f[v]$
- considere uma aresta arbitrária (u, v)
- basta mostrar que $f[u] > f[v]$
- considere o instante em que (u, v) foi examinada
- como (u, v) não é aresta de retorno, v não pode ser cinza
    1. se v for branco, ele será descendente de u e $f[u] > f[v]$
    2. se v for preto, então ele já foi finalizado e $f[u] > f[v]$
- em qualquer caso, obtemos o que desejávamos

Correção

Teorema

TOPOLOGICAL-SORT(G) devolve ordenação topológica de G.

Demonstração

- a lista devolvida está em ordem decrescente de $f[v]$
- considere uma aresta arbitrária (u, v)
- basta mostrar que $f[u] > f[v]$
- considere o instante em que (u, v) foi examinada
- como (u, v) não é aresta de retorno, v não pode ser cinza
    1. se v for branco, ele será descendente de u e $f[u] > f[v]$
    2. se v for preto, então ele já foi finalizado e $f[u] > f[v]$
- em qualquer caso, obtemos o que desejávamos

Correção

Teorema

$\textsc{Topological-Sort}(G)$ devolve ordenação topológica de G.

Demonstração

- a lista devolvida está em ordem decrescente de $f[v]$
- considere uma aresta arbitrária (u, v)
- basta mostrar que $f[u] > f[v]$
- considere o instante em que (u, v) foi examinada
- como (u, v) não é aresta de retorno, v não pode ser cinza
  1. se v for branco, ele será descendente de u e $f[u] > f[v]$
  2. se v for preto, então ele já foi finalizado e $f[u] > f[v]$
- em qualquer caso, obtemos o que desejávamos

Correção

Teorema

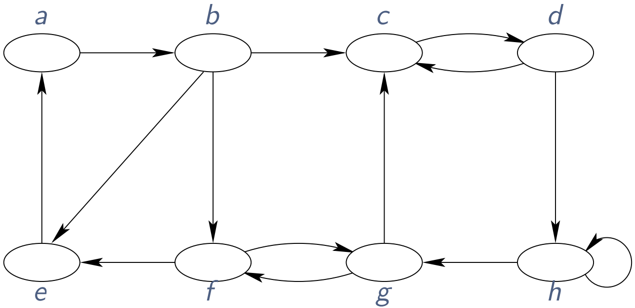
TOPOLOGICAL-SORT(G) devolve ordenação topológica de G.

Demonstração

- a lista devolvida está em ordem decrescente de $f[v]$
- considere uma aresta arbitrária (u, v)
- basta mostrar que $f[u] > f[v]$
- considere o instante em que (u, v) foi examinada
- como (u, v) não é aresta de retorno, v não pode ser cinza
  1. se v for branco, ele será descendente de u e $f[u] > f[v]$
  2. se v for preto, então ele já foi finalizado e $f[u] > f[v]$
- em qualquer caso, obtemos o que desejávamos

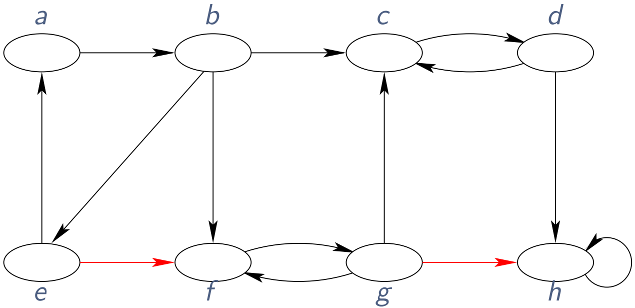
Componentes fortemente conexas

Grafo fortemente conexo



Um grafo direcionado $G = (V, E)$ é fortemente conexo se, para todo par de vértices u, v de G, existe um caminho direcionado de u a v e existe um caminho direcionado de v a u.

Grafo fortemente conexo



Nem todo grafo direcionado é fortemente conexo

Componente fortemente conexa

Uma componente fortemente conexa de um grafo direcionado $G = (V, E)$ é um subconjunto de vértices $C \subseteq V$ tal que

(1) o subgrafo induzido por C é fortemente conexo e

(2) C é maximal com respeito à propriedade (1).

Componente fortemente conexa

Uma componente fortemente conexa de um grafo direcionado $G = (V, E)$ é um subconjunto de vértices $C \subseteq V$ tal que

(1) o subgrafo induzido por C é fortemente conexo e

(2) C é maximal com respeito à propriedade (1).

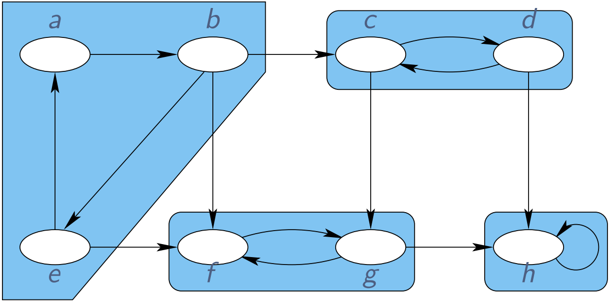
Componente fortemente conexa

Uma componente fortemente conexa de um grafo direcionado $G = (V, E)$ é um subconjunto de vértices $C \subseteq V$ tal que

(1) o subgrafo induzido por C é fortemente conexo e

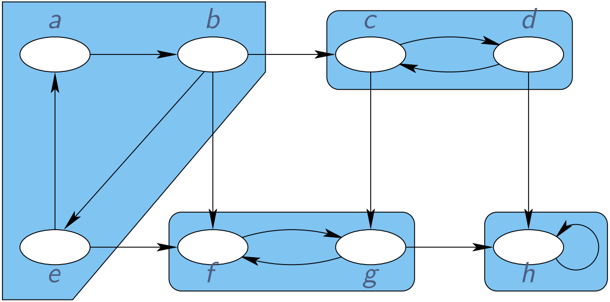
(2) C é maximal com respeito à propriedade (1).

Componente fortemente conexa



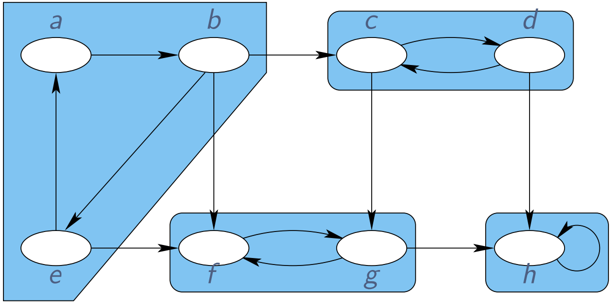
- podemos particionar um grafo direcionado em componentes fortemente conexas
- como encontrar as componentes fortemente conexas?

Componente fortemente conexa



- podemos particionar um grafo direcionado em componentes fortemente conexas
- como encontrar as componentes fortemente conexas?

Componente fortemente conexa

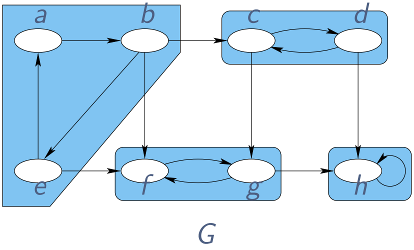


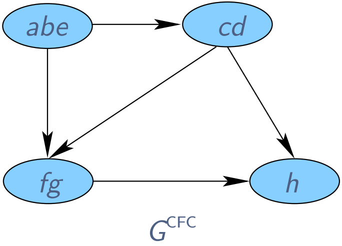
- podemos particionar um grafo direcionado em componentes fortemente conexas
- como encontrar as componentes fortemente conexas?

Grafo componente

O grafo componente de um grafo direcionado $G = (V, E)$ é um grafo direcionado em que

- cada vértice é uma componente fortemente conexa
- existe aresta (C, D) se houver $(u, v) \in E$ com $u \in C$ e $v \in D$



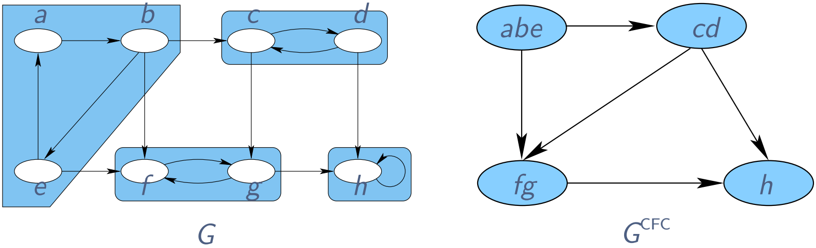


- denotamos o grafo componente por G^{CFC}
- note que G^{CFC} é acíclico (por quê?)

Grafo componente

O grafo componente de um grafo direcionado $G = (V, E)$ é um grafo direcionado em que

- cada vértice é uma componente fortemente conexa
- existe aresta (C, D) se houver $(u, v) \in E$ com $u \in C$ e $v \in D$

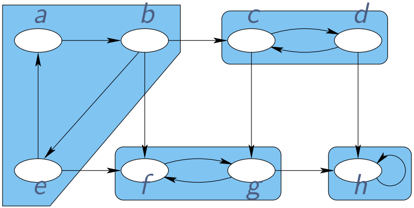


- denotamos o grafo componente por G^{CFC}
- note que G^{CFC} é acíclico (por quê?)

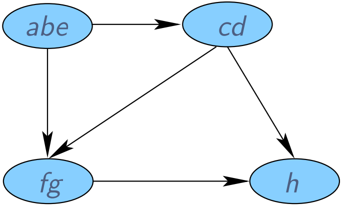
Grafo componente

O grafo componente de um grafo direcionado $G = (V, E)$ é um grafo direcionado em que

- cada vértice é uma componente fortemente conexa
- existe aresta (C, D) se houver $(u, v) \in E$ com $u \in C$ e $v \in D$



G



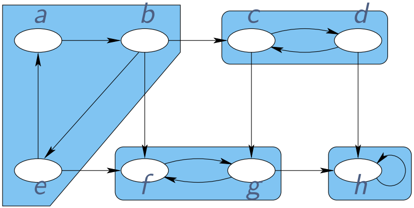
G^{CFC}

- denotamos o grafo componente por G^{CFC}
- note que G^{CFC} é acíclico (por quê?)

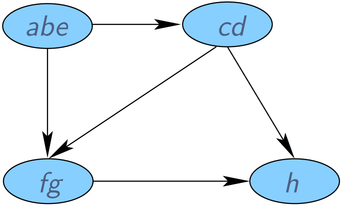
Grafo componente

O grafo componente de um grafo direcionado $G = (V, E)$ é um grafo direcionado em que

- cada vértice é uma componente fortemente conexa
- existe aresta (C, D) se houver $(u, v) \in E$ com $u \in C$ e $v \in D$



G



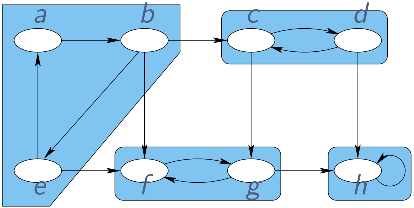
G^{CFC}

- denotamos o grafo componente por G^{CFC}
- note que G^{CFC} é acíclico (por quê?)

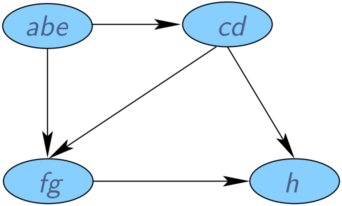
Grafo componente

O grafo componente de um grafo direcionado $G = (V, E)$ é um grafo direcionado em que

- cada vértice é uma componente fortemente conexa
- existe aresta (C, D) se houver $(u, v) \in E$ com $u \in C$ e $v \in D$



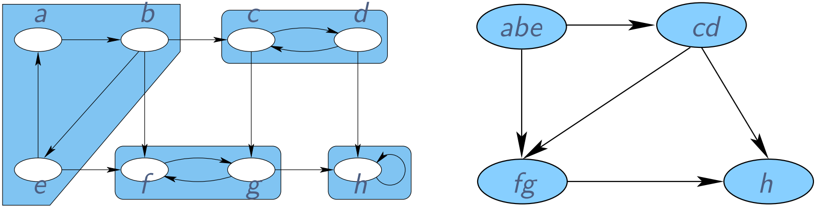
G



G^{CFC}

- denotamos o grafo componente por G^{CFC}
- note que G^{CFC} é acíclico (por quê?)

Grafo componente

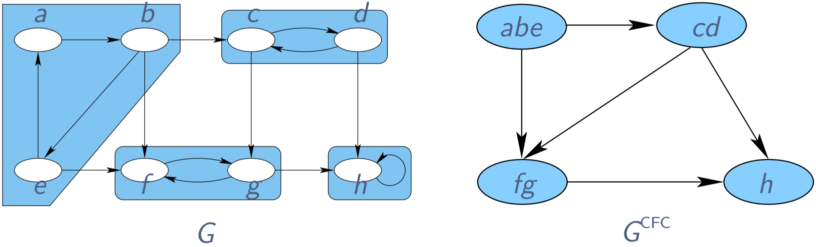


G $\qquad$ G^{CFC}

Considere uma busca em profundidade sobre G

- seja u o último vértice finalizado
- então u deve pertencer a uma fonte de G^{CFC} (por quê?)
- visitamos as componentes de G^{CFC} em ordem topológica

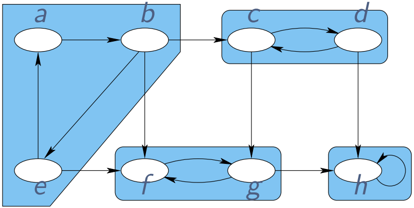
Grafo componente



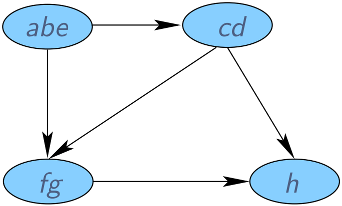
Considere uma busca em profundidade sobre G

- seja u o último vértice finalizado
- então u deve pertencer a uma fonte de G^{CFC} (por quê?)
- visitamos as componentes de G^{CFC} em ordem topológica

Grafo componente



G

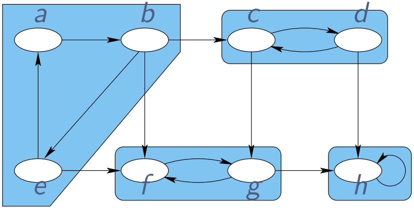


G^{CFC}

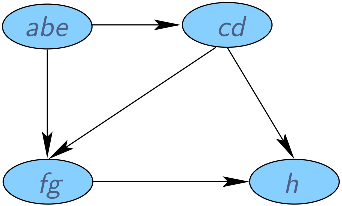
Considere uma busca em profundidade sobre G

- seja u o último vértice finalizado
- então u deve pertencer a uma fonte de G^{CFC} (por quê?)
- visitamos as componentes de G^{CFC} em ordem topológica

Grafo componente



G



G^{CFC}

Considere uma busca em profundidade sobre G

- seja u o último vértice finalizado
- então u deve pertencer a uma fonte de G^{CFC} (por quê?)
- visitamos as componentes de G^{CFC} em ordem topológica

Grafo transposto

O grafo transposto de um grafo direcionado $G = (V, E)$ é um grafo direcionado que

- tem o mesmo conjunto de vértices
- tem uma aresta (u, v) se houver aresta (v, u) em G

Observações

- denotamos o grafo transposto por G^T
- ele é obtido invertendo-se as arestas de G
- podemos calcular G^T em tempo $O(V + E)$

Grafo transposto

O grafo transposto de um grafo direcionado $G = (V, E)$ é um grafo direcionado que

- tem o mesmo conjunto de vértices
- tem uma aresta (u, v) se houver aresta (v, u) em G

Observações

- denotamos o grafo transposto por G^T
- ele é obtido invertendo-se as arestas de G
- podemos calcular G^T em tempo $O(V + E)$

Grafo transposto

O grafo transposto de um grafo direcionado $G = (V, E)$ é um grafo direcionado que

- tem o mesmo conjunto de vértices
- tem uma aresta (u, v) se houver aresta (v, u) em G

Observações

- denotamos o grafo transposto por G^T
- ele é obtido invertendo-se as arestas de G
- podemos calcular G^T em tempo $O(V + E)$

Grafo transposto

O grafo transposto de um grafo direcionado $G = (V, E)$ é um grafo direcionado que

- tem o mesmo conjunto de vértices
- tem uma aresta (u, v) se houver aresta (v, u) em G

Observações

- denotamos o grafo transposto por G^{T}
- ele é obtido invertendo-se as arestas de G
- podemos calcular G^{T} em tempo $O(V + E)$

Grafo transposto

O grafo transposto de um grafo direcionado $G = (V, E)$ é um grafo direcionado que

- tem o mesmo conjunto de vértices
- tem uma aresta (u, v) se houver aresta (v, u) em G

Observações

- denotamos o grafo transposto por G^{T}
- ele é obtido invertendo-se as arestas de G
- podemos calcular G^{T} em tempo $O(V + E)$

Grafo transposto

O grafo transposto de um grafo direcionado $G = (V, E)$ é um grafo direcionado que

- tem o mesmo conjunto de vértices
- tem uma aresta (u, v) se houver aresta (v, u) em G

Observações

- denotamos o grafo transposto por G^{T}
- ele é obtido invertendo-se as arestas de G
- podemos calcular G^{T} em tempo $O(V + E)$

Grafo transposto

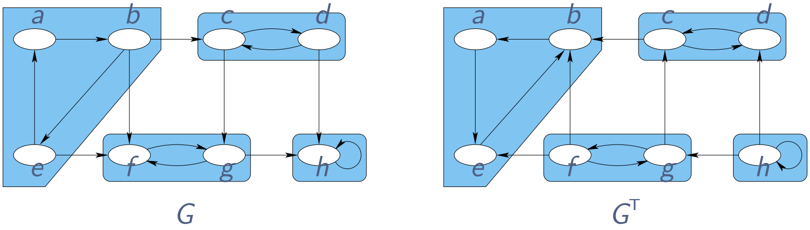
O grafo transposto de um grafo direcionado $G = (V, E)$ é um grafo direcionado que

- tem o mesmo conjunto de vértices
- tem uma aresta (u, v) se houver aresta (v, u) em G

Observações

- denotamos o grafo transposto por G^{T}
- ele é obtido invertendo-se as arestas de G
- podemos calcular G^{T} em tempo $O(V + E)$

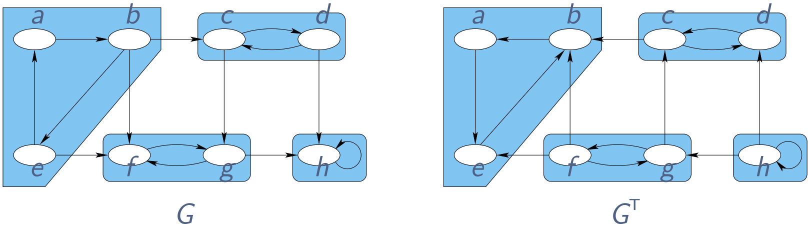
Grafo transposto



Como encontrar uma componente fortemente conexa?

- note que G e G^T têm as mesmas componentes
- mas componentes fontes para G são sorvedouros para G^T
- suponha que temos um vértice u de uma fonte em G^{CFC}
- os vértices alcançáveis de u formam uma componente!

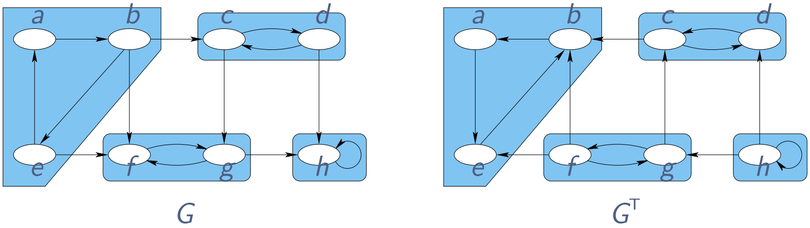
Grafo transposto



Como encontrar uma componente fortemente conexa?

- note que G e G^T têm as mesmas componentes
- mas componentes fontes para G são sorvedouros para G^T
- suponha que temos um vértice u de uma fonte em G^{CFC}
- os vértices alcançáveis de u formam uma componente!

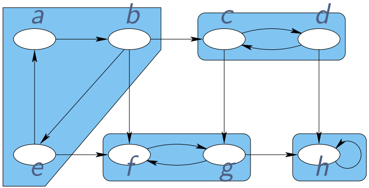
Grafo transposto



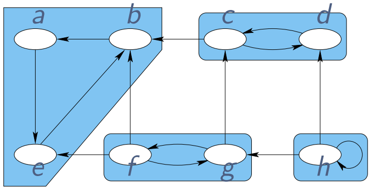
Como encontrar uma componente fortemente conexa?

- note que G e G^{T} têm as mesmas componentes
- mas componentes fontes para G são sorvedouros para G^{T}
- suponha que temos um vértice u de uma fonte em G^{CFC}
- os vértices alcançáveis de u formam uma componente!

Grafo transposto



G

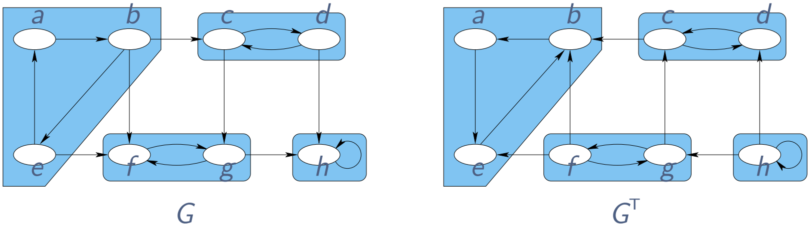


G^{T}

Como encontrar uma componente fortemente conexa?

- note que G e G^{T} têm as mesmas componentes
- mas componentes fontes para G são sorvedouros para G^{T}
- suponha que temos um vértice u de uma fonte em G^{CFC}
- os vértices alcançáveis de u formam uma componente!

Grafo transposto



Como encontrar uma componente fortemente conexa?

- note que G e G^{T} têm as mesmas componentes
- mas componentes fontes para G são sorvedouros para G^{T}
- suponha que temos um vértice u de uma fonte em G^{CFC}
- os vértices alcançáveis de u formam uma componente!

Algoritmo

Componentes-Fortemente-Conexas(G)

1 execute $\mathrm{DFS}(G)$ e calcule $f[v]$ para cada $v \in V$

2 compute G^{T}

3 execute $\mathrm{DFS}(G^{\mathsf{T}})$, mas no laço principal de DFS, considere os vértices em ordem decrescente de $f[v]$

4 devolva os conjuntos de vértices de cada árvore da floresta de busca encontrada

- a complexidade de tempo é $O(V + E)$

Algoritmo

Componentes-Fortemente-Conexas(G)

1 execute $\mathrm{DFS}(G)$ e calcule $f[v]$ para cada $v \in V$

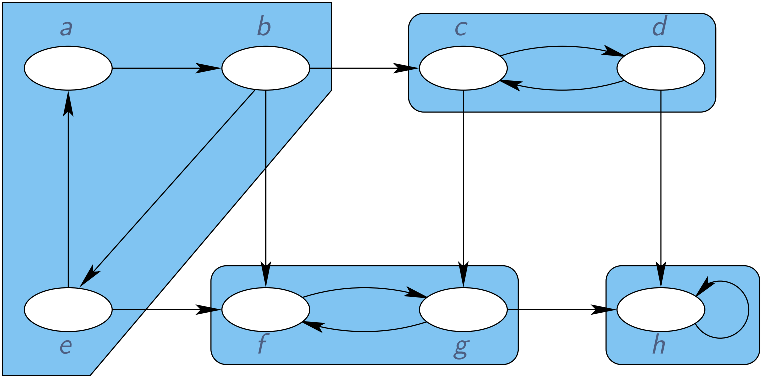
2 compute G^{T}

3 execute $\mathrm{DFS}(G^{\mathsf{T}})$, mas no laço principal de DFS, considere os vértices em ordem decrescente de $f[v]$

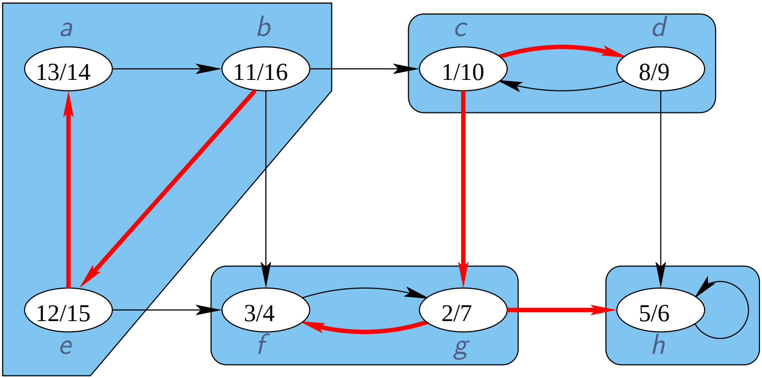
4 devolva os conjuntos de vértices de cada árvore da floresta de busca encontrada

- a complexidade de tempo é $O(V + E)$

Exemplo

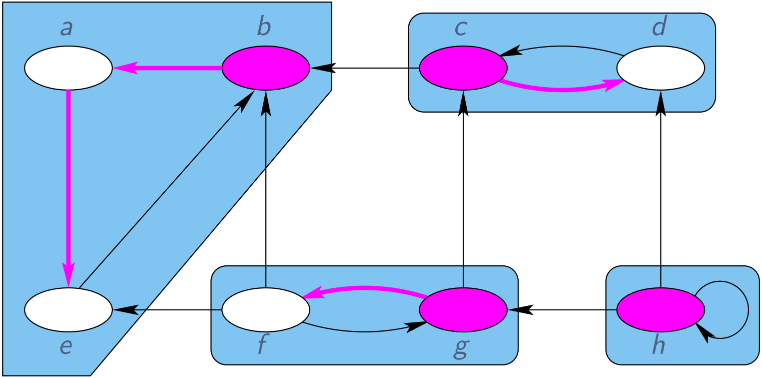


Exemplo



1 execute $\mathrm{DFS}(G)$ e calcule $f[v]$ para cada $v \in V$

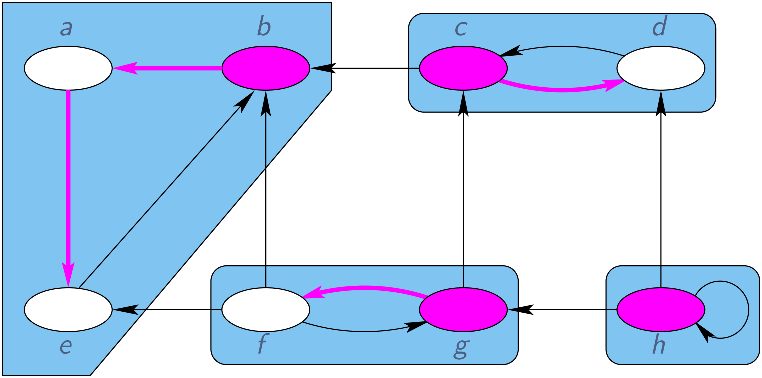
Exemplo



2 execute $\mathrm{DFS}(G^{\mathsf{T}})$ considerando os vértices em ordem decrescente de $f[v]$

3 devolva os conjuntos de vértices de cada árvore da floresta de busca encontrada

Exemplo



2 execute $\mathrm{DFS}(G^{\mathsf{T}})$ considerando os vértices em ordem decrescente de $f[v]$

3 devolva os conjuntos de vértices de cada árvore da floresta de busca encontrada

Anexos

Correção

Componentes-Fortemente-Conexas(G)

1 execute $\mathrm{DFS}(G)$ e calcule $f[v]$ para cada $v \in V$

2 compute G^{T}

3 execute $\mathrm{DFS}(G^{\mathsf{T}})$, mas no laço principal de DFS, considere os vértices em ordem decrescente de $f[v]$

4 devolva os conjuntos de vértices de cada árvore da floresta de busca encontrada

Teorema

O algoritmo Componentes-Fortemente-Conexas determina as componentes fortemente conexas de G em tempo $O(V + E)$.

- antes da demonstração, precisamos de uma preparação

Correção

Componentes-Fortemente-Conexas(G)

1 execute $\mathrm{DFS}(G)$ e calcule $f[v]$ para cada $v \in V$

2 compute G^{T}

3 execute $\mathrm{DFS}(G^{\mathsf{T}})$, mas no laço principal de DFS, considere os vértices em ordem decrescente de $f[v]$

4 devolva os conjuntos de vértices de cada árvore da floresta de busca encontrada

Teorema

O algoritmo Componentes-Fortemente-Conexas determina as componentes fortemente conexas de G em tempo $O(V + E)$.

- antes da demonstração, precisamos de uma preparação

Lema auxiliar

Lema 1

Sejam C e D duas componentes fortemente conexas e considere vértices $u, v \in C$ e $u', v' \in D$.

- Se existe algum caminho $u \rightsquigarrow u'$,
- então não existe um caminho $v' \rightsquigarrow v$.

- segue da maximalidade de C e D
- o lema significa que G^{CFC} é acíclico

Lema auxiliar

Lema 1

Sejam C e D duas componentes fortemente conexas e considere vértices $u, v \in C$ e $u', v' \in D$.

- Se existe algum caminho $u \rightsquigarrow u'$,
- então não existe um caminho $v' \rightsquigarrow v$.

- segue da maximalidade de C e D
- o lema significa que G^{CFC} é acíclico

Lema auxiliar

Lema 1

Sejam C e D duas componentes fortemente conexas e considere vértices $u, v \in C$ e $u', v' \in D$.

- Se existe algum caminho $u \rightsquigarrow u'$,
- então não existe um caminho $v' \rightsquigarrow v$.

- segue da maximalidade de C e D
- o lema significa que G^{CFC} é acíclico

Lema auxiliar

Lema 1

Sejam C e D duas componentes fortemente conexas e considere vértices $u, v \in C$ e $u', v' \in D$.

- Se existe algum caminho $u \rightsquigarrow u'$,
- então não existe um caminho $v' \rightsquigarrow v$.

- segue da maximalidade de C e D
- o lema significa que G^{CFC} é acíclico

Lema auxiliar

Lema 1

Sejam C e D duas componentes fortemente conexas e considere vértices $u, v \in C$ e $u', v' \in D$.

- Se existe algum caminho $u \rightsquigarrow u'$,
- então não existe um caminho $v' \rightsquigarrow v$.

- segue da maximalidade de C e D
- o lema significa que G^{CFC} é acíclico

Definições auxiliares

Vamos adotar alguma convenção

- vamos considerar a primeira execução do algoritmo DFS
- d e f referem-se à busca em profundidade da linha 1

Para cada subconjunto U de vértices, defina

$$d(U) = \min_{u \in U} \left\{ d[u] \right\} \quad \text{e} \quad f(U) = \max_{u \in U} \left\{ f[u] \right\}$$

Em outras palavras

- $d(U)$ é o primeiro instante em que um vértice é descoberto
- $f(U)$ é o último instante em que um vértice é finalizado

Definições auxiliares

Vamos adotar alguma convenção

- vamos considerar a primeira execução do algoritmo DFS
- d e f referem-se à busca em profundidade da linha 1

Para cada subconjunto U de vértices, defina

$$d(U) = \min_{u \in U} \left\{ d[u] \right\} \quad \text{e} \quad f(U) = \max_{u \in U} \left\{ f[u] \right\}$$

Em outras palavras

- $d(U)$ é o primeiro instante em que um vértice é descoberto
- $f(U)$ é o último instante em que um vértice é finalizado

Definições auxiliares

Vamos adotar alguma convenção

- vamos considerar a primeira execução do algoritmo DFS
- d e f referem-se à busca em profundidade da linha 1

Para cada subconjunto U de vértices, defina

$$d(U) = \min_{u \in U} \{d[u]\} \quad \text{e} \quad f(U) = \max_{u \in U} \{f[u]\}$$

Em outras palavras

- $d(U)$ é o primeiro instante em que um vértice é descoberto
- $f(U)$ é o último instante em que um vértice é finalizado

Definições auxiliares

Vamos adotar alguma convenção

- vamos considerar a primeira execução do algoritmo DFS
- d e f referem-se à busca em profundidade da linha 1

Para cada subconjunto U de vértices, defina

$$d(U) = \min_{u \in U} \left\{ d[u] \right\} \quad \text{e} \quad f(U) = \max_{u \in U} \left\{ f[u] \right\}$$

Em outras palavras

- $d(U)$ é o primeiro instante em que um vértice é descoberto
- $f(U)$ é o último instante em que um vértice é finalizado

Definições auxiliares

Vamos adotar alguma convenção

- vamos considerar a primeira execução do algoritmo DFS
- d e f referem-se à busca em profundidade da linha 1

Para cada subconjunto U de vértices, defina

$$d(U) = \min_{u \in U} \{d[u]\} \quad \text{e} \quad f(U) = \max_{u \in U} \{f[u]\}$$

Em outras palavras

- $d(U)$ é o primeiro instante em que um vértice é descoberto
- $f(U)$ é o último instante em que um vértice é finalizado

Definições auxiliares

Vamos adotar alguma convenção

- vamos considerar a primeira execução do algoritmo DFS
- d e f referem-se à busca em profundidade da linha 1

Para cada subconjunto U de vértices, defina

$$d(U) = \min_{u \in U} \{d[u]\} \quad \text{e} \quad f(U) = \max_{u \in U} \{f[u]\}$$

Em outras palavras

- $d(U)$ é o primeiro instante em que um vértice é descoberto
- $f(U)$ é o último instante em que um vértice é finalizado

Definições auxiliares

Vamos adotar alguma convenção

- vamos considerar a primeira execução do algoritmo DFS
- d e f referem-se à busca em profundidade da linha 1

Para cada subconjunto U de vértices, defina

$$d(U) = \min_{u \in U} \left\{ d[u] \right\} \quad \text{e} \quad f(U) = \max_{u \in U} \left\{ f[u] \right\}$$

Em outras palavras

- $d(U)$ é o primeiro instante em que um vértice é descoberto
- $f(U)$ é o último instante em que um vértice é finalizado

Definições auxiliares

Vamos adotar alguma convenção

- vamos considerar a primeira execução do algoritmo DFS
- d e f referem-se à busca em profundidade da linha 1

Para cada subconjunto U de vértices, defina

$$d(U) = \min_{u \in U} \left\{ d[u] \right\} \quad \text{e} \quad f(U) = \max_{u \in U} \left\{ f[u] \right\}$$

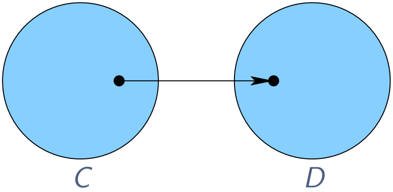
Em outras palavras

- $d(U)$ é o primeiro instante em que um vértice é descoberto
- $f(U)$ é o último instante em que um vértice é finalizado

Outro lema auxiliar

Lema 2

Sejam C e D duas componentes fortemente conexas. Se existe aresta (u, v) tal que $u \in C$ e $v \in D$, então $f(C) > f(D)$.



Corolário

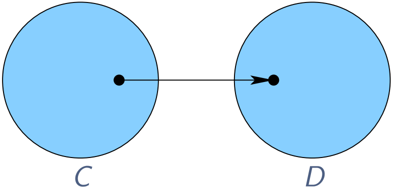
Se G^T tem aresta (u, v) tal que $u \in C$ e $v \in D$, então $f(C) < f(D)$.

- segue do fato de que G e G^T têm as mesmas componentes

Outro lema auxiliar

Lema 2

Sejam C e D duas componentes fortemente conexas. Se existe aresta (u, v) tal que $u \in C$ e $v \in D$, então $f(C) > f(D)$.



Corolário

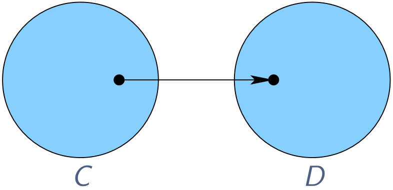
Se G^{T} tem aresta (u, v) tal que $u \in C$ e $v \in D$, então $f(C) < f(D)$.

- segue do fato de que G e G^{T} têm as mesmas componentes

Outro lema auxiliar

Lema 2

Sejam C e D duas componentes fortemente conexas. Se existe aresta (u, v) tal que $u \in C$ e $v \in D$, então $f(C) > f(D)$.

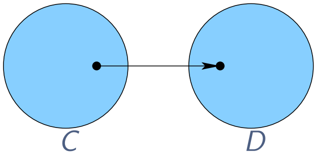


Corolário

Se G^T tem aresta (u, v) tal que $u \in C$ e $v \in D$, então $f(C) < f(D)$.

- segue do fato de que G e G^T têm as mesmas componentes

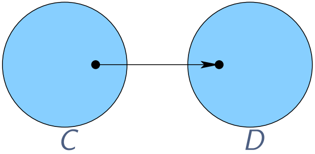
Prova do lema



Demonstração

- primeiro suponha que $d(C) < d(D)$
- isso é, suponha que descobrimos C antes de D
- seja x um vértice de C tal que $d[x] = d(C)$
- assim, x é o primeiro vértice de C a ser descoberto
- no instante $d[x]$, existia um caminho branco de x a cada um dos vértices em $C \cup D$
- então todos os vértices de $C \cup D$ são descendentes de x
- e portanto $f(D) < f[x] \leq f(C)$

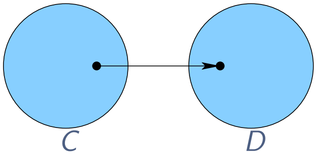
Prova do lema



Demonstração

- primeiro suponha que $d(C) < d(D)$
- isso é, suponha que descobrimos C antes de D
- seja x um vértice de C tal que $d[x] = d(C)$
- assim, x é o primeiro vértice de C a ser descoberto
- no instante $d[x]$, existia um caminho branco de x a cada um dos vértices em $C \cup D$
- então todos os vértices de $C \cup D$ são descendentes de x
- e portanto $f(D) < f[x] \leq f(C)$

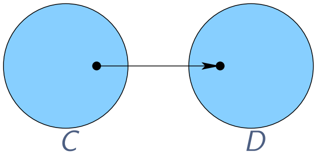
Prova do lema



Demonstração

- primeiro suponha que $d(C) < d(D)$
- isso é, suponha que descobrimos C antes de D
- seja x um vértice de C tal que $d[x] = d(C)$
- assim, x é o primeiro vértice de C a ser descoberto
- no instante $d[x]$, existia um caminho branco de x a cada um dos vértices em $C \cup D$
- então todos os vértices de $C \cup D$ são descendentes de x
- e portanto $f(D) < f[x] \leq f(C)$

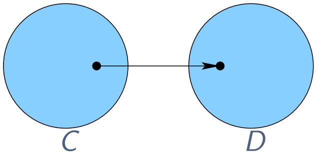
Prova do lema



Demonstração

- primeiro suponha que $d(C) < d(D)$
- isso é, suponha que descobrimos C antes de D
- seja x um vértice de C tal que $d[x] = d(C)$
- assim, x é o primeiro vértice de C a ser descoberto
- no instante $d[x]$, existia um caminho branco de x a cada um dos vértices em $C \cup D$
- então todos os vértices de $C \cup D$ são descendentes de x
- e portanto $f(D) < f[x] \leq f(C)$

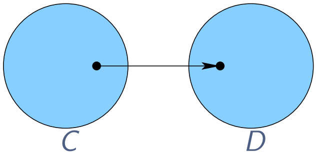
Prova do lema



Demonstração

- primeiro suponha que $d(C) < d(D)$
- isso é, suponha que descobrimos C antes de D
- seja x um vértice de C tal que $d[x] = d(C)$
- assim, x é o primeiro vértice de C a ser descoberto
- no instante $d[x]$, existia um caminho branco de x a cada um dos vértices em $C \cup D$
- então todos os vértices de $C \cup D$ são descendentes de x
- e portanto $f(D) < f[x] \leq f(C)$

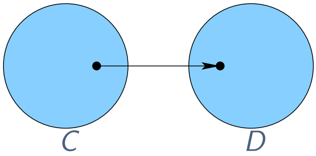
Prova do lema



Demonstração

- primeiro suponha que $d(C) < d(D)$
- isso é, suponha que descobrimos C antes de D
- seja x um vértice de C tal que $d[x] = d(C)$
- assim, x é o primeiro vértice de C a ser descoberto
- no instante $d[x]$, existia um caminho branco de x a cada um dos vértices em $C \cup D$
- então todos os vértices de $C \cup D$ são descendentes de x
- e portanto $f(D) < f[x] \leq f(C)$

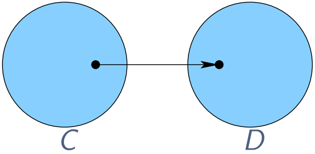
Prova do lema



Demonstração

- primeiro suponha que $d(C) < d(D)$
- isso é, suponha que descobrimos C antes de D
- seja x um vértice de C tal que $d[x] = d(C)$
- assim, x é o primeiro vértice de C a ser descoberto
- no instante $d[x]$, existia um caminho branco de x a cada um dos vértices em $C \cup D$
- então todos os vértices de $C \cup D$ são descendentes de x
- e portanto $f(D) < f[x] \leq f(C)$

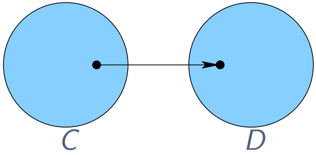
Prova do lema



Demonstração

- primeiro suponha que $d(C) < d(D)$
- isso é, suponha que descobrimos C antes de D
- seja x um vértice de C tal que $d[x] = d(C)$
- assim, x é o primeiro vértice de C a ser descoberto
- no instante $d[x]$, existia um caminho branco de x a cada um dos vértices em $C \cup D$
- então todos os vértices de $C \cup D$ são descendentes de x
- e portanto $f(D) < f[x] \leq f(C)$

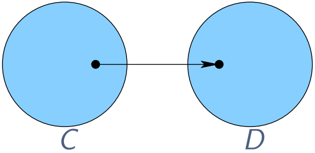
Prova do lema (cont)



Continuando

- agora suponha que $d(C) > d(D)$
- assim, o primeiro vértice a ser descoberto está em D
- logo, cada um dos vértices de D é finalizado antes de qualquer vértice de C ser descoberto
- portanto $f(C) > f(D)$

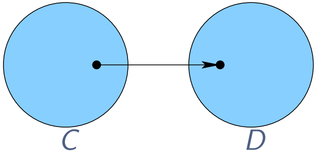
Prova do lema (cont)



Continuando

- agora suponha que $d(C) > d(D)$
- assim, o primeiro vértice a ser descoberto está em D
- logo, cada um dos vértices de D é finalizado antes de qualquer vértice de C ser descoberto
- portanto $f(C) > f(D)$

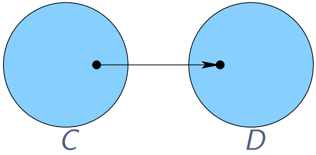
Prova do lema (cont)



Continuando

- agora suponha que $d(C) > d(D)$
- assim, o primeiro vértice a ser descoberto está em D
- logo, cada um dos vértices de D é finalizado antes de qualquer vértice de C ser descoberto
- portanto $f(C) > f(D)$

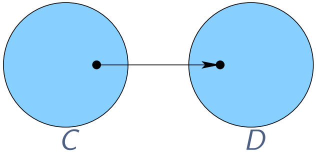
Prova do lema (cont)



Continuando

- agora suponha que $d(C) > d(D)$
- assim, o primeiro vértice a ser descoberto está em D
- logo, cada um dos vértices de D é finalizado antes de qualquer vértice de C ser descoberto
- portanto $f(C) > f(D)$

Prova do lema (cont)



Continuando

- agora suponha que $d(C) > d(D)$
- assim, o primeiro vértice a ser descoberto está em D
- logo, cada um dos vértices de D é finalizado antes de qualquer vértice de C ser descoberto
- portanto $f(C) > f(D)$

Prova do teorema

Teorema

O algoritmo COMPONENTES-FORTEMENTE-CONEXAS determina as componentes fortemente conexas de G em tempo $O(V + E)$.

Demonstração

- vamos provar que as k primeiras árvores produzidas na linha 3 correspondem a componentes fortemente conexas
- a prova é por indução em k
- quando $k = 0$, a afirmação é trivial, então tome $k \geq 1$
- suponha que as primeiras $k - 1$ primeiras árvores produzidas correspondem a componentes

Prova do teorema

Teorema

O algoritmo COMPONENTES-FORTEMENTE-CONEXAS determina as componentes fortemente conexas de G em tempo $O(V+E)$.

Demonstração

- vamos provar que as k primeiras árvores produzidas na linha 3 correspondem a componentes fortemente conexas
- a prova é por indução em k
- quando $k=0$, a afirmação é trivial, então tome $k \geq 1$
- suponha que as primeiras $k-1$ primeiras árvores produzidas correspondem a componentes

Prova do teorema

Teorema

O algoritmo COMPONENTES-FORTEMENTE-CONEXAS determina as componentes fortemente conexas de G em tempo $O(V + E)$.

Demonstração

- vamos provar que as k primeiras árvores produzidas na linha 3 correspondem a componentes fortemente conexas
- a prova é por indução em k
- quando $k = 0$, a afirmação é trivial, então tome $k \geq 1$
- suponha que as primeiras $k - 1$ primeiras árvores produzidas correspondem a componentes

Prova do teorema

Teorema

O algoritmo COMPONENTES-FORTEMENTE-CONEXAS determina as componentes fortemente conexas de G em tempo $O(V+E)$.

Demonstração

- vamos provar que as k primeiras árvores produzidas na linha 3 correspondem a componentes fortemente conexas
- a prova é por indução em k
- quando $k = 0$, a afirmação é trivial, então tome $k \geq 1$
- suponha que as primeiras $k-1$ primeiras árvores produzidas correspondem a componentes

Prova do teorema

Teorema

O algoritmo COMPONENTES-FORTEMENTE-CONEXAS determina as componentes fortemente conexas de G em tempo $O(V+E)$.

Demonstração

- vamos provar que as k primeiras árvores produzidas na linha 3 correspondem a componentes fortemente conexas
- a prova é por indução em k
- quando $k=0$, a afirmação é trivial, então tome $k \geq 1$
- suponha que as primeiras $k-1$ primeiras árvores produzidas correspondem a componentes

Prova do teorema

Teorema

O algoritmo COMPONENTES-FORTEMENTE-CONEXAS determina as componentes fortemente conexas de G em tempo $O(V+E)$.

Demonstração

- vamos provar que as k primeiras árvores produzidas na linha 3 correspondem a componentes fortemente conexas
- a prova é por indução em k
- quando $k=0$, a afirmação é trivial, então tome $k \geq 1$
- suponha que as primeiras $k-1$ primeiras árvores produzidas correspondem a componentes

Prova do teorema (cont)

Considere a k-ésima árvore produzida pelo algoritmo

- seja u a raiz dessa árvore de busca
- e seja C a componente fortemente conexa que contém u
- vamos mostrar que a árvore produzida contém todos os vértices de C e somente os vértices de C
- isso completará a indução e a prova do teorema

Prova do teorema (cont)

Considere a k-ésima árvore produzida pelo algoritmo

- seja u a raiz dessa árvore de busca
- e seja C a componente fortemente conexa que contém u
- vamos mostrar que a árvore produzida contém todos os vértices de C e somente os vértices de C
- isso completará a indução e a prova do teorema

Prova do teorema (cont)

Considere a k-ésima árvore produzida pelo algoritmo

- seja u a raiz dessa árvore de busca
- e seja C a componente fortemente conexa que contém u
- vamos mostrar que a árvore produzida contém todos os vértices de C e somente os vértices de C
- isso completará a indução e a prova do teorema

Prova do teorema (cont)

Considere a k-ésima árvore produzida pelo algoritmo

- seja u a raiz dessa árvore de busca
- e seja C a componente fortemente conexa que contém u
- vamos mostrar que a árvore produzida contém
  todos os vértices de C e somente os vértices de C
- isso completará a indução e a prova do teorema

Prova do teorema (cont)

Considere a k-ésima árvore produzida pelo algoritmo

- seja u a raiz dessa árvore de busca
- e seja C a componente fortemente conexa que contém u
- vamos mostrar que a árvore produzida contém todos os vértices de C e somente os vértices de C
- isso completará a indução e a prova do teorema

Prova do teorema (cont)

A árvore contém todos vértices de C

- considere o instante em que u é descoberto
- por indução nenhum vértice de C foi finalizado
- então nesse instante $d[u]$ os vértices de C são brancos
- assim, todos os vértices de C tornam-se descendentes de u na árvore de busca de G^T

A árvore contém somente vértices de C

- suponha que existe aresta (u, v) que sai de C
- seja D a componente fortemente conexa que contém v
- pelo corolário do Lema 2, temos $f(C) < f(D)$
- então descobrimos vértices de D antes de u
- por indução, todos vértices de D já foram finalizados
- portanto, a árvore só contém vértices de C

Prova do teorema (cont)

A árvore contém todos vértices de C

- considere o instante em que u é descoberto
- por indução nenhum vértice de C foi finalizado
- então nesse instante $d[u]$ os vértices de C são brancos
- assim, todos os vértices de C tornam-se descendentes de u na árvore de busca de G^T

A árvore contém somente vértices de C

- suponha que existe aresta (u, v) que sai de C
- seja D a componente fortemente conexa que contém v
- pelo corolário do Lema 2, temos $f(C) < f(D)$
- então descobrimos vértices de D antes de u
- por indução, todos vértices de D já foram finalizados
- portanto, a árvore só contém vértices de C

Prova do teorema (cont)

A árvore contém todos vértices de C

- considere o instante em que u é descoberto
- por indução nenhum vértice de C foi finalizado
- então nesse instante $d[u]$ os vértices de C são brancos
- assim, todos os vértices de C tornam-se descendentes de u na árvore de busca de G^T

A árvore contém somente vértices de C

- suponha que existe aresta (u, v) que sai de C
- seja D a componente fortemente conexa que contém v
- pelo corolário do Lema 2, temos $f(C) < f(D)$
- então descobrimos vértices de D antes de u
- por indução, todos vértices de D já foram finalizados
- portanto, a árvore só contém vértices de C

Prova do teorema (cont)

A árvore contém todos vértices de C

- considere o instante em que u é descoberto
- por indução nenhum vértice de C foi finalizado
- então nesse instante $d[u]$ os vértices de C são brancos
- assim, todos os vértices de C tornam-se descendentes de u na árvore de busca de G^T

A árvore contém somente vértices de C

- suponha que existe aresta (u, v) que sai de C
- seja D a componente fortemente conexa que contém v
- pelo corolário do Lema 2, temos $f(C) < f(D)$
- então descobrimos vértices de D antes de u
- por indução, todos vértices de D já foram finalizados
- portanto, a árvore só contém vértices de C

Prova do teorema (cont)

A árvore contém todos vértices de C

- considere o instante em que u é descoberto
- por indução nenhum vértice de C foi finalizado
- então nesse instante $d[u]$ os vértices de C são brancos
- assim, todos os vértices de C tornam-se descendentes de u na árvore de busca de G^T

A árvore contém somente vértices de C

- suponha que existe aresta (u, v) que sai de C
- seja D a componente fortemente conexa que contém v
- pelo corolário do Lema 2, temos $f(C) < f(D)$
- então descobrimos vértices de D antes de u
- por indução, todos vértices de D já foram finalizados
- portanto, a árvore só contém vértices de C

Prova do teorema (cont)

A árvore contém todos vértices de C

- considere o instante em que u é descoberto
- por indução nenhum vértice de C foi finalizado
- então nesse instante $d[u]$ os vértices de C são brancos
- assim, todos os vértices de C tornam-se descendentes de u na árvore de busca de G^T

A árvore contém somente vértices de C

- suponha que existe aresta (u, v) que sai de C
- seja D a componente fortemente conexa que contém v
- pelo corolário do Lema 2, temos $f(C) < f(D)$
- então descobrimos vértices de D antes de u
- por indução, todos vértices de D já foram finalizados
- portanto, a árvore só contém vértices de C

Prova do teorema (cont)

A árvore contém todos vértices de C

- considere o instante em que u é descoberto
- por indução nenhum vértice de C foi finalizado
- então nesse instante $d[u]$ os vértices de C são brancos
- assim, todos os vértices de C tornam-se descendentes de u na árvore de busca de G^{T}

A árvore contém somente vértices de C

- suponha que existe aresta (u, v) que sai de C
- seja D a componente fortemente conexa que contém v
- pelo corolário do Lema 2, temos $f(C) < f(D)$
- então descobrimos vértices de D antes de u
- por indução, todos vértices de D já foram finalizados
- portanto, a árvore só contém vértices de C

Prova do teorema (cont)

A árvore contém todos vértices de C

- considere o instante em que u é descoberto
- por indução nenhum vértice de C foi finalizado
- então nesse instante $d[u]$ os vértices de C são brancos
- assim, todos os vértices de C tornam-se descendentes de u na árvore de busca de G^T

A árvore contém somente vértices de C

- suponha que existe aresta (u, v) que sai de C
- seja D a componente fortemente conexa que contém v
- pelo corolário do Lema 2, temos $f(C) < f(D)$
- então descobrimos vértices de D antes de u
- por indução, todos vértices de D já foram finalizados
- portanto, a árvore só contém vértices de C

Prova do teorema (cont)

A árvore contém todos vértices de C

- considere o instante em que u é descoberto
- por indução nenhum vértice de C foi finalizado
- então nesse instante $d[u]$ os vértices de C são brancos
- assim, todos os vértices de C tornam-se descendentes de u na árvore de busca de G^{T}

A árvore contém somente vértices de C

- suponha que existe aresta (u, v) que sai de C
- seja D a componente fortemente conexa que contém v
- pelo corolário do Lema 2, temos $f(C) < f(D)$
- então descobrimos vértices de D antes de u
- por indução, todos vértices de D já foram finalizados
- portanto, a árvore só contém vértices de C

Prova do teorema (cont)

A árvore contém todos vértices de C

- considere o instante em que u é descoberto
- por indução nenhum vértice de C foi finalizado
- então nesse instante $d[u]$ os vértices de C são brancos
- assim, todos os vértices de C tornam-se descendentes de u na árvore de busca de G^T

A árvore contém somente vértices de C

- suponha que existe aresta (u, v) que sai de C
- seja D a componente fortemente conexa que contém v
- pelo corolário do Lema 2, temos $f(C) < f(D)$
- então descobrimos vértices de D antes de u
- por indução, todos vértices de D já foram finalizados
- portanto, a árvore só contém vértices de C

Prova do teorema (cont)

A árvore contém todos vértices de C

- considere o instante em que u é descoberto
- por indução nenhum vértice de C foi finalizado
- então nesse instante $d[u]$ os vértices de C são brancos
- assim, todos os vértices de C tornam-se descendentes de u na árvore de busca de G^T

A árvore contém somente vértices de C

- suponha que existe aresta (u, v) que sai de C
- seja D a componente fortemente conexa que contém v
- pelo corolário do Lema 2, temos $f(C) < f(D)$
- então descobrimos vértices de D antes de u
- por indução, todos vértices de D já foram finalizados
- portanto, a árvore só contém vértices de C

Prova do teorema (cont)

A árvore contém todos vértices de C

- considere o instante em que u é descoberto
- por indução nenhum vértice de C foi finalizado
- então nesse instante $d[u]$ os vértices de C são brancos
- assim, todos os vértices de C tornam-se descendentes de u na árvore de busca de G^T

A árvore contém somente vértices de C

- suponha que existe aresta (u, v) que sai de C
- seja D a componente fortemente conexa que contém v
- pelo corolário do Lema 2, temos $f(C) < f(D)$
- então descobrimos vértices de D antes de u
- por indução, todos vértices de D já foram finalizados
- portanto, a árvore só contém vértices de C