Grafos

Algoritmos elementares

Notas

Conteúdo

Notas

Introdução

- Geralmente medimos o tamanho de um grafo $G = (V, E)$ em termos do número de vértice $|V|$ e do número de arestas $|E|$
  - Dentro da notação assintótica, o termo V representará $|V|$, e o termo E, representará $|E|$
- Um grafo $G = (V, E)$ é
  - **Esparso** se $|E|$ é muito menor que $|V|^2$
  - **Denso** se $|E|$ está próximo de $|V|^2$

Notas

Representação de grafos

- Existem duas maneiras padrão para representar um grafo $G = (V, E)$
  - Lista de adjacências
  - Matriz de adjacências

Notas

Representação de grafos

- Lista de adjacências
  - A **representação de lista de adjacências** consiste de um arranjo `adj` de $|V|$ listas, uma para cada vértice
  - Para cada $u \in V$, a lista de adjacências `adj[`u`]` contém (ponteiros para) todos os vértices v tal que existe a aresta $(u, v) \in E$.

Notas

Representação de grafos

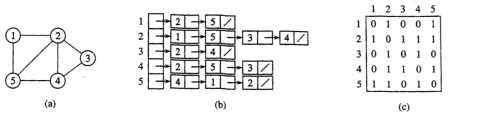


FIGURA 22.1 Duas representações de um grafo não orientado. (a) Um grafo não orientado G que tem cinco vértices e sete arestas. (b) Uma representação de G como uma lista de adjacências. (c) A representação de G como uma matriz de adjacências

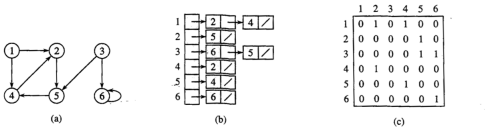


FIGURA 22.2 Duas representações de um grafo orientado. (a) Um grafo orientado G que tem seis vértices e oito arestas. (b) Uma representação de G como uma lista de adjacências. (c) A representação de G como uma matriz de adjacências

Notas

Representação de grafos

- Lista de adjacências
  - Qual é a soma dos comprimentos de todas as listas de adjacências?
    - Se G é um grafo orientado, a soma é $|E|$
    - Se G é um grafo não orientado, a soma é $2|E|$
  - Qual é a quantidade de memória requerida? $\Theta(V + E)$
  - Adequada para grafos esparsos
  - Desvantagem
    - Não existe nenhum modo rápido para determinar se uma dada aresta (u, v) está presente no grafo

Notas

Representação de grafos

- Matriz de adjacências
  - Na **representação de matriz de adjacências**, supomos que os vértices são numerados $1, 2, \ldots, |V|$
  - A representação consiste em uma matriz $|V| \times |V| A = (a_{ij})$ tal que

$$a_{ij} = \begin{cases} 1 \text{ se } (i, j) \in E \\ 0 \text{ caso contrário} \end{cases}$$

  - Esta representação permite consultar se uma aresta faz parte do grafo em tempo constante

Notas

Representação de grafos

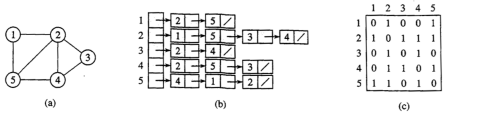


FIGURA 22.1 Duas representações de um grafo não orientado. (a) Um grafo não orientado G que tem cinco vértices e sete arestas. (b) Uma representação de G como uma lista de adjacências. (c) A representação de G como uma matriz de adjacências

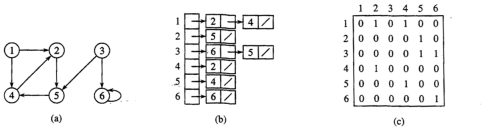


FIGURA 22.2 Duas representações de um grafo orientado. (a) Um grafo orientado G que tem seis vértices e oito arestas. (b) Uma representação de G como uma lista de adjacências. (c) A representação de G como uma matriz de adjacências

Notas

Representação de grafos

- Matriz de adjacências
  - Qual é quantidade de memória requerida? $\Theta(V^2)$. A quantidade de memória independe de E
  - Em um grafo não orientado, a matriz é igual a sua transposta, desta forma é possível usar apenas os elementos abaixo (ou acima) da diagonal principal
  - Adequada para grafos densos

Notas

Atributos

- Muitos algoritmos que operam em grafos precisam manter atributos para vértices e/ou arestas
- Nos códigos, indicamos os atributos como
  - `v.d`, atributo `d` do vértice `v`
  - `(u, v).f`, atributo `f` da aresta `(u, v)`
- Como estes atributos podem ser implementados?
  - Depende da linguagem de programação, algoritmo, etc
  - Os atributos podem ser armazenado diretamente na lista ou matriz de adjacência
  - Se os vértices são enumerados de $1..|V|$ os atributos podem ser representados em arranjos, tais como $d[1..|V|]$
  - Atributos de vértices podem ficar nos registros que representam os vértices

Notas

Exercícios

- 22.1-1 a 22.1-8

Notas

Exercícios

22.1-1 Dada uma representação de lista de adjacências de um grafo orientado, qual o tempo necessário para computar o grau de saída de todo o vértice? Qual o tempo necessário para computar os graus de entrada?

Notas

Exercícios - Solução 22.1-1

```
computar-graus-de-saida(G)
1 for v in G.V
2   v.grau-de-saida = 0
3 for v in G.V
4   for u in G.adj[v]
5     v.grau-de-saida += 1

computar-graus-de-entrada(G)
1 for v in G.V
2   v.grau-de-entrada = 0
3 for v in G.V
4   for u in G.adj[v]
5     u.grau-de-entrada += 1
```


Notas

Exercícios

22.1-2 Forneça uma representação de lista de adjacências para uma árvore binária completa sobre 7 vértices. Forneça uma representação de matriz de adjacências equivalente. Suponha que os vértices estejam numerados de 1 até 7 como em um heap binário.

22.1-3 A **transposta** de um grafo orientado $G = (V, E)$ é o grafo $G^T = (V, E^T)$, onde $E^T = \{(v, u) \in V \times V : (u, v) \in E\}$. Deste modo, G^T é G com todas as suas arestas invertidas. Descreva algoritmos eficientes para calcular G^T a partir de G, para a representação de lista de adjacências e também para a representação de matriz de adjacências de G. Analise os tempos de execução de seus algoritmos.

Notas

Exercícios

22.1-4 Dada uma representação de lista de adjacências de um multigrafo $G(V, E)$, descreva um algoritmo de tempo $O(V + E)$ para calcular a representação de lista de adjacência do grafo não orientado "equivalente" $G' = (V, E')$, onde E' consiste nas arestas em E com todas as arestas múltiplas entre dois vértices substituídas por uma aresta única e com todos os autoloops removidos.

Notas

Exercícios - Solução 22.1-4

```
transformar-multigrafo-em-grafo(G)
 1 n = |G.V|
 2 G' = (G.V, 0) // mesmo conjunto de vértices
                 // conjuto vazio de arestas
 3 P = arranjo[1..n]
 4 for i = 1..n
 5   P[i] = 0
 6 for v in G.V
 7   for u in G.adj[v]
 8     if p[u] != v
 9       p[u] = v // marca a aresta (v,u)
                  // como presente em G'
10       G'.adj[v].add(u)
```


Notas

Exercícios

22.1-5 O **quadrado** de um grafo orientado $G = (V, E)$ é o grafo $G^2 = (V, E^2)$ tal que $(u, w) \in E^2$ se e somente se, para algum $v \in V$, tem-se $(u, v) \in E$ e também $(v, w) \in E$. Ou seja, G^2 contém uma aresta entre u e w sempre que G contém um caminho com exatamente duas arestas entre u e w. Descreva algoritmos eficientes para calcular G^2 a partir de G para uma representação de lista de adjacências e para uma representação de matriz de adjacências de G. Analise os tempos de execução de seus algoritmos.

Notas

Exercícios

22.1-6 Quando uma representação de matriz de adjacências é usada, a maioria dos algoritmos de grafos exige $\Omega(V^2)$, mas existem algumas exceções. Mostre que detectar se um grafo orientado G contém um **sorvedor universal** – um vértice com grau de entrada $|V| - 1$ e grau de saída 0 – é uma operação que pode ser realizada no tempo $O(V)$, dada uma matriz de adjacências de G.

Notas

Exercícios

22.1-7 A **matriz de incidência** de um grafo orientado $G = (V, E)$ é uma matriz $|V| \times |E| B = (b_{ij})$ tal que

$$b_{ij} = \begin{cases} -1 & \text{se a aresta } j \text{ sai do vértice } i, \\ 1 & \text{se a aresta } j \text{ entra no vértice } i, \\ 0 & \text{em caso contrário} \end{cases}$$

Descreva o que representa as entrada do produto de matrizes BB^T, onde B^T é a transposta de B.

Notas

Exercícios

22.1-8 Suponha que, em vez de uma lista ligada, cada entrada de arranjo `adj[u]` é uma tabela hash contendo os vértices v para os quais $(u, v) \in E$. Se todas as pesquisas de arestas forem igualmente prováveis, qual será o tempo esperado para determinar se uma aresta está no grafo. Que desvantagens esse esquema apresenta? Sugira uma estrutura de dados alternativa para cada lista de arestas que resolva estes problemas. Sua alternativa tem desvantagens em comparação com a tabela hash?

Notas

Pesquisas

- Pesquisar um grafo significa acompanhar sistematicamente as arestas do grafo de modo a alcançar os vértices
- Um algoritmo de pesquisa aplicado a um grafo pode descobrir muito sobre a sua estrutura
- Alguns algoritmos iniciam com um pesquisa no grafo para descobrir sua estrutura
- Outros algoritmos são alterações simples de algoritmos de pesquisa
- As técnicas de pesquisa de grafo estão no núcleo do campo de algoritmos em grafos
- Veremos dois algoritmos de pesquisa
  - Busca em largura
  - Busca em profundidade

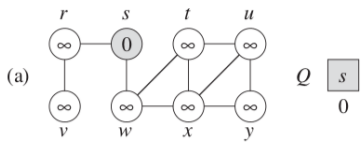
Notas

Busca em largura

- Dados um $G = (V, E)$ e um vértice de origem s, a busca em largura explora sistematicamente as arestas de G até descobrir cada vértices acessível de s
- O algoritmo calcula a distância (menor número de arestas) deste s até todos os vértices acessíveis a partir de s
- O algoritmo produz uma árvore primeiro em extensão
- Recebe este nome porque expande a fronteira entre vértices descobertos e não descobertos uniformemente ao longo da extensão da fronteira. Descobre todos os vértices de distância k de s antes de descobrir quaisquer vértices de distância $k + 1$

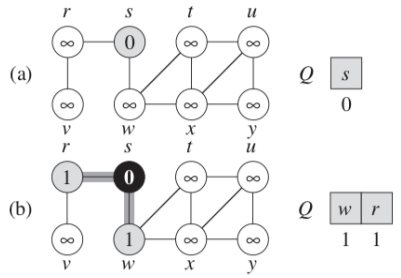
Notas

Busca em largura



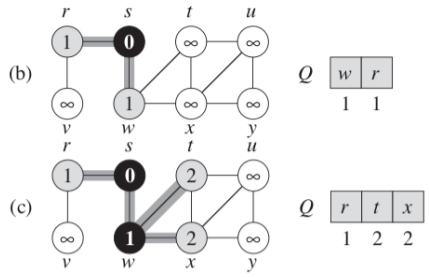
Notas

Busca em largura



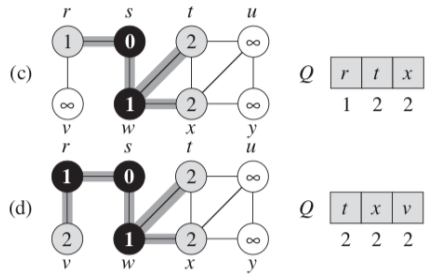
Notas

Busca em largura



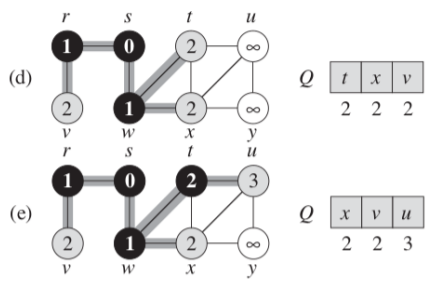
Notas

Busca em largura



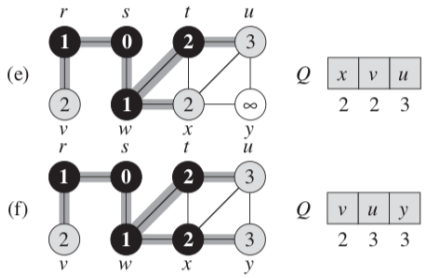
Notas

Busca em largura



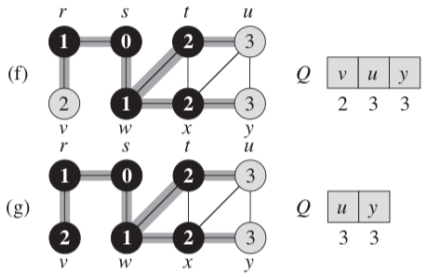
Notas

Busca em largura



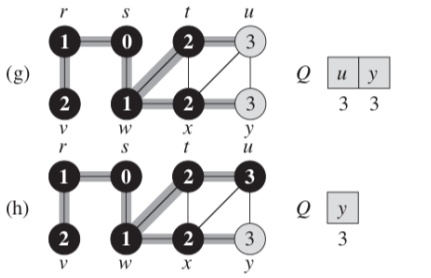
Notas

Busca em largura



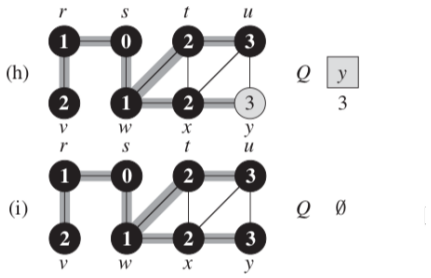
Notas

Busca em largura



Notas

Busca em largura



Notas

Busca em largura

```
bfs(G, s)
 1 for cada vértice u em G.V - {s}
 2   u.d = infinito
 3   u.pai = nil
 4   u.cor = branco
 5 s.d = 0
 6 s.pai = nil
 7 s.cor = cinza
 8 Q = {}
 9 Q.add(s)
10 while Q != {}
11   u = Q.remove()
12   for cada vértice v em u.adj
13     if v.cor = branco
14       v.d = u.d + 1
15       v.pai = u
16       v.cor = cinza
17       Q.add(v)
18   u.cor = preto
```


Notas

Busca em largura

- Análise
  - Análise agregada
  - O teste da linha 13 garante que cada vértice é colocado na fila no máximo uma vez, e portanto, é retirado da fila no máximo uma vez
  - As operações de colocar e retirar da fila demoram $O(1)$, portanto, o tempo total das operações com filas é $O(V)$
  - A lista de adjacência de cada vértice é examinada apenas quando o vértice é retirado da fila, portanto, no máximo uma vez
  - Como a soma dos comprimentos das listas de adjacências é $\Theta(E)$, o tempo para percorrer todas as listas é o máximo $O(E)$
  - O tempo de inicialização é $O(V)$
  - Tempo total de execução do `bfs` é $O(V + E)$

Notas

Árvores primeiro na extensão

- BFS constrói uma árvore primeiro na extensão
- Árvore é representada pelo campo pai (π) em cada vértice
- Para um grafo $G = (V, E)$ e um vértice de origem s, definimos o **subgrafo predecessor** de G como $G_\pi = (V_\pi, E_\pi)$ onde
  - $V_\pi = \{v \in V : v.\pi \neq \text{NIL}\} \cup \{s\}$
  - $E_\pi = \{(v.\pi, v) : v \in V_\pi - \{s\}\}$
- O subgrafo predecessor G_π é uma árvore primeiro na extensão
  - V_π consiste nos vértices acessíveis a partir de s
  - Para todo $v \in V_\pi$, existe um caminho único simples desde s até v em G_π, que também é o caminho mais curto de s até v em G
- Uma árvore primeiro na extensão é de fato uma árvore, pois é conexa e $|E_\pi| = |V_\pi| - 1$

Notas

Árvores primeiro na extensão

```
imprimir-caminho(G, s, v)
1  if v = s
2    imprimir s
3  else if v.pai = nil
4    imprimir "nenhum caminho de" s "para" v "existente"
5  else
6    imprimir-caminho(G, s, v.pai)
7    imprimir v
```

- Executado em tempo linear no número de vértices no caminho impresso, pois cada chamada recursiva é feita para um caminho com um vértice menor que o atual

Notas

Exercícios

- 22.2-1, 22.2-2, 22.2-3, 22.2-5, 22.2-6, 22.2-8

Notas

Exercícios

22.2-1 Mostre os valores de d e π que resultam da execução da busca em largura sobre o grafo orientado da figura 22.2(a), usando o vértice 3 como origem.

22.2-2 Mostre os valores de d e π que resultam da execução da busca em largura sobre o grafo não-orientado da figura 22.3, usando o vértice u como origem.

22.2-3 Qual o tempo de execução de `bfs` se o grafo de entrada é representado pos uma matriz de adjacências e o algoritmo é modificado para manipular essa forma de entrada?

Notas

Exercícios

22.2-5 Forneça um exemplo de um grafo orientado $G = (V, E)$, um vértice de origem $s \in V$ e um conjunto de arestas de árvore $E_\pi \subseteq E$ tal que, para cada vértice $v \in V$, o caminho único em (V, E_π) de s até v é um caminho mais curto em G, ainda que o conjunto de arestas E_π não possa ser produzido pela execução de BFS sobre G, não importando o modo como os vértices estão ordenados em cada lista de adjacências.

Notas

Exercícios

22.2-6 Existem dois tipos de lutadores profissionais: "bons sujeitos" e "maus sujeitos". Entre qualquer par de lutadores profissionais pode ou não haver uma rivalidade. Suponha que temos n lutadores profissionais e temos uma lista de r pares de lutadores para os quais existem rivalidades. Dê um algoritmo de tempo $O(n + r)$ que determine se é possível designar alguns dos lutadores como bons sujeitos e os restantes como maus sujeitos, de tal forma que a rivalidade ocorra em cada caso entre um bom sujeito e um mau sujeito. Se for possível realizar tal designação, seu algoritmo deve produzi-la.

Notas

Exercícios

22.2-8 Seja $G = (V, E)$ um grafo conectado não orientado. Forneça um algoritmo de tempo $O(V + E)$ para calcular um caminho em G que percorra cada aresta de E exatamente uma vez em cada sentido. Descreva como você pode encontrar a saída de uma labirinto se receber uma grande provisão de moedas de centavos.

Notas

Busca em profundidade

- Procurar "mais fundo" no grafo sempre que possível
- As arestas são exploradas a partir do vértices v mais recentemente descoberto que ainda tem arestas inexploradas saindo dele
- Quando todas as arestas de v são exploradas, a busca regressa para explorar as arestas que deixam o vértice a partir do qual v foi descoberto
- Este processo continua até que todos os vértices acessíveis a partir da origem tenham sidos descobertos
- Se restarem vértices não descobertos, a busca se repetirá para estes vértices

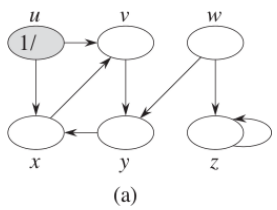
Notas

Busca em profundidade

- Durante a execução do algoritmo, diversos atributos são definidos para os vértices
- Quando um vértice v é descoberto a partir de um vértice u, o campo predecessor $v.\pi = u$ é definido
- Cada vértice é inicialmente branco, o vértice é marcado de cinza quando é descoberto e marcado de preto quando é terminado (sua lista de adjacências é completamente examinada)
- Cada vértice tem dois carimbos de tempo $v.d$ (quando o vértice é descoberto) e $v.f$ (quando o vértice é terminado)

Notas

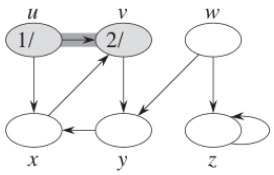
Busca em profundidade



(a)

Notas

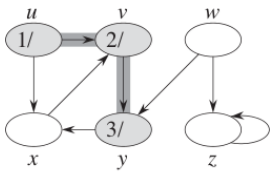
Busca em profundidade



(b)

Notas

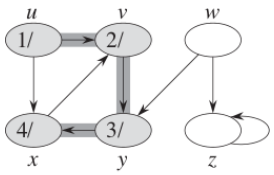
Busca em profundidade



(c)

Notas

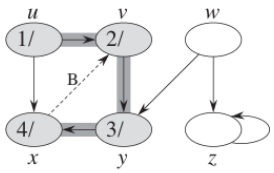
Busca em profundidade



(d)

Notas

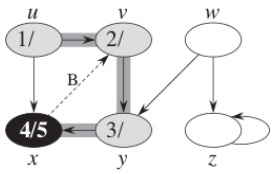
Busca em profundidade



(e)

Notas

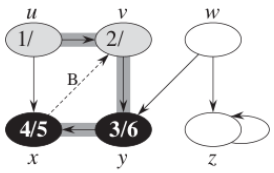
Busca em profundidade



(f)

Notas

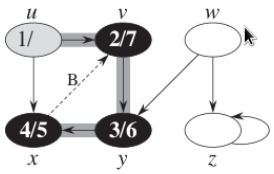
Busca em profundidade



(g)

Notas

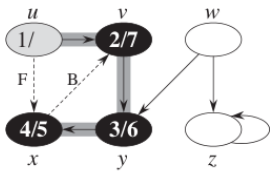
Busca em profundidade



(h)

Notas

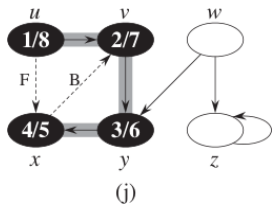
Busca em profundidade



(i)

Notas

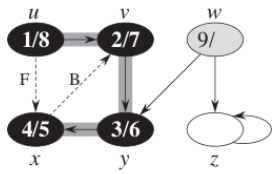
Busca em profundidade



(j)

Notas

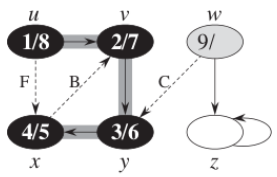
Busca em profundidade



(k)

Notas

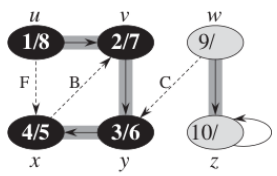
Busca em profundidade



(l)

Notas

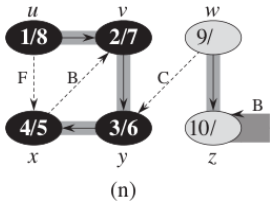
Busca em profundidade



(m)

Notas

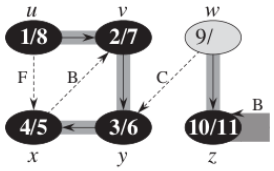
Busca em profundidade



(n)

Notas

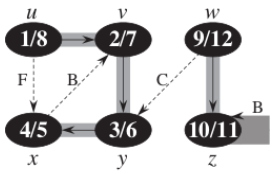
Busca em profundidade



(o)

Notas

Busca em profundidade



(p)

Notas

Busca em profundidade

```
dfs(G)
 1 for cada vértice u em G.V
 2   u.cor = branco
 3   u.pai = nil
 4 tempo = 0
 5 for cada vértice u em G.V
 6   if u.cor = branco
 7     dfs-visit(v)

dfs-visit(u)
 1 u.cor = cinza
 2 tempo = tempo + 1
 3 u.d = tempo
 4 for cada vértice v em u.adj
 5   if v.cor = branco
 6     v.pai = u
 7     dfs-visit(v)
 8 u.cor = preto
 9 u.f = tempo = tempo + 1
```


Notas

Busca em profundidade

- Análise
  - Os loops nas linhas 1 a 3 e nas linhas 5 a 7 demoram tempo $\Theta(V)$ (sem contar o tempo das chamadas a `dfs-visit`)
  - Usamos a análise agregada
  - O procedimento `dfs-visit` é chamado exatamente uma vez para cada vértice, isto porque `df-visit` é chamado para os vértices brancos, e no início de `df-visit` o vértice é pintado de cinza
  - Durante a execução de `dfs-visit(v)`, o loop nas linhas 4 a 7 é executado $|v.\text{adj}|$ vezes, como $\sum_{v \in V} |v.adj| = \Theta(E)$, o custo total da execução das 4 a 7 de `dfs-visit` é $\Theta(E)$
  - Portanto, o tempo de execução do `dfs` é $\Theta(V + E)$

Notas

Floresta primeiro na profundidade

- DFS constrói uma floresta primeiro na profundidade, contendo diversas árvores primeiro na profundidade
- Para um grafo $G = (V, E)$, definimos o **subgrafo predecessor** de uma busca primeiro na profundidade de G como o grafo $G_\pi = (V, E_\pi)$ onde
  - $E_\pi = \{(v.\pi, v) : v \in V \text{ e } v.\pi \neq \text{NIL}\}$
- As arestas em E_π são **arestas da árvore**

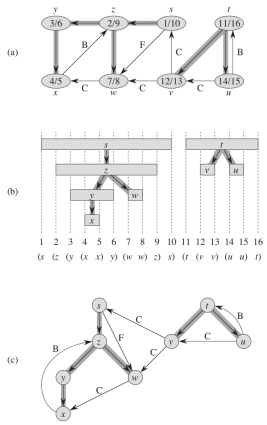
Notas

Propriedades da busca primeiro na profundidade

- Teorema 22.7 (Teorema do parênteses)
  - Para dois vértices quaisquer u e v, exatamente uma das três condições a seguir é verdadeira
    - Os intervalos $[u.d, u.f]$ e $[v.d, v.f]$ são disjuntos e nem u e nem v são descendentes um do outro na floresta primeiro na profundidade
    - O intervalo $[u.d, u.f]$ está contido inteiramente no intervalo $[v.d, v.f]$ e u é descendente de v em uma árvore primeiro na profundidade
    - O intervalo $[v.d, v.f]$ está contido inteiramente no intervalo $[u.d, u.f]$ e v é descendente de u em uma árvore primeiro na profundidade

Notas

Propriedades da busca primeiro na profundidade



Notas

Propriedades da busca primeiro na profundidade

- Classificação das arestas
  - Podemos definir quadro tipos de arestas em termos da floresta primeiro na profundidade G_π
    - **Arestas da árvore**, são as arestas na floresta primeiro na profundidade G_π. Uma aresta (u, v) é uma aresta da árvore se v foi descoberto primeiro pela exploração da aresta (u, v)
    - **Arestas de retorno** são as arestas (u, v) que conectam um vértice u a um ancestral v na árvore primeiro na profundidade
    - **Arestas para frente** são as arestas (u, v) que não são arestas da árvore e conectam o vértice u a um descendente v na árvore primeiro na profundidade
    - **Arestas cruzadas** são todas as outras arestas

Notas

Exercícios

- 22.3-1 a 22.3-3, 22.3-6 a 22.3-11

Notas

Exercícios

22.3-1 Faça uma diagrama 3 por 3 com linhas e colunas com identificações branco, cinza e preto. Em cada célula (i, j), indique se, em qualquer instante durante uma busca em profundidade de um grafo orientado, pode existir uma aresta de um vértice de cor i até um vértice de cor j. Para cada aresta possível, indique quais tipos de arestas ela pode ser. Crie um segundo diagrama como esse para a busca em profundidade de um grafo não orientado.

Notas

Exercícios

22.3-2 Mostre como a busca em profundidade funciona sobre o grafo da figura 22.6. Suponha que o loop for das linhas 5 a 7 do procedimento `dfs` considere os vértices em ordem alfabética, e suponha que cada lista de adjacência esteja em ordem alfabética. Mostre os tempos de descoberta e término para cada vértice, e mostre também a classificação cada arresta.

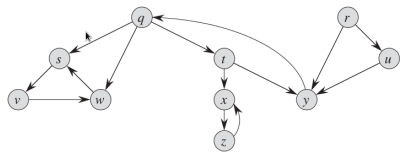


Figure 22.6 A directed graph for use in Exercises 22.3-2 and 22.5-2.

Notas

Exercícios

22.3-3 Mostre a estrutura de parênteses da busca em profundidade apresentada na figura 22.4

22.3-6 Reescreva o procedimento `dfs`, utilizando uma pilha para eliminar a recursão.

22.3-7 Forneça um contra-exemplo para a hipótese de que, se existe um caminho de u para v em um grafo orientado G, e se $u.d < v.d$ em uma busca em profundidade de G, então v é um descendente de u na floresta primeiro na profundidade produzida.

22.3-8 Forneça um contra-exemplo para a hipótese de que, se existe um caminho de u para v em um grafo orientado G, então qualquer busca em profundidade deve resultar em $v.d \leq u.f$.

Notas

Exercícios

22.3-9 Modifique o pseudocódigo para a busca em profundidade, de tal modo que ele imprima toda aresta no grafo orientado G, juntamente com seu tipo. Mostre quais modificações, se for o caso, devem ser feitas se G for não orientado.

22.3-10 Explique como um vértice u de um grafo orientado pode acabar em uma árvore primeiro na profundidade contendo apenas u, embora tenha tanto arestas de entrada quando de saída de G.

Notas

Exercícios

22.3-11 Mostre que uma busca em profundidade de um grafo não orientado G pode ser usada para identificar os componentes conexos de G, e que a floresta primeiro na profundidade contém tantas árvores quantos componentes conexos existem em G. Mais precisamente, mostre como modificar a busca em profundidade de modo que cada vértice v receba a atribuição de uma etiqueta inteira $v.cc$ entre 1 e k, onde k é o número de componentes conexos de G, de tal forma que $u.cc = v.cc$ se e somente se u e v estiverem no mesmo componente conectado.

Notas

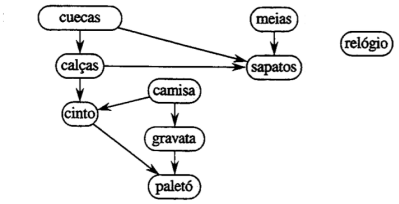
Ordenação topológica

- Uma **ordenação topológica** de um grafo acíclico orientado $G = (V, E)$ é uma ordenação linear de todos os vértices, que que se G contém uma aresta (u, v), então u aparece antes de v na ordenação
- Se os vértices forem dispostos em uma linha horizontal, todas as arestas devem ter a orientação da esquerda para direita
- Aplicação
  - Definição da ordem de execução de tarefas dependentes. Ex: `Makefile`

Notas

Ordenação topológica

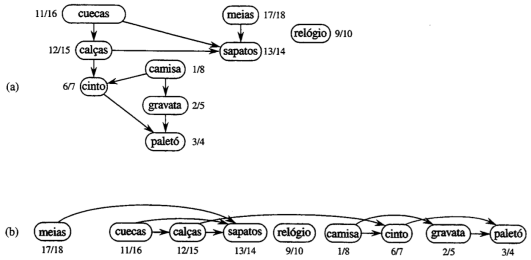
- Exemplo: o professor Bumstead deve se vestir pela manhã



Notas

Ordenação topológica

```
topological-sort(G)
1 chamar DFS(G) para calcular o tempo de término v.f
  para cada vértice v
2 à medida que cada vértice é terminado, inserir o
  vértice à frente de uma lista ligada
3 return a lista ligada de vértices
```



Notas

Ordenação topológica

- Análise do tempo de execução
  - O tempo de execução da busca em profundidade é $\Theta(V + E)$
  - O tempo para inserir cada vértice na lista de saída é $O(1)$, cada vértice é inserido apenas uma vez e portanto o tempo total gasto em operações de inserções é de $\Theta(V)$
  - Portanto, o tempo de execução do algoritmo é $\Theta(V + E)$

Notas

Ordenação topológica

- Corretude
  - Precisamos mostrar que se $(u, v) \in E$, então $v.f < u.f$
  - Quando a aresta (u, v) é explorada, quais são as cores de u e v?
  - u é cinza
  - v é cinza também?
    - Não, porque isto implicaria que v é ancestral de u, e portando a aresta (u, v) seria uma aresta de retorno. Gaos não contém arestas de retorno
  - v é branco?
    - Então v torna-se um descendente de u. Pelo teorema do parênteses $u.d < v.d < \mathbf{v.f} < \mathbf{u.f}$
  - v é preto?
    - Então v já foi finalizado. Como a aresta (u, v) está sendo explorada, u não foi finalizado. Logo $v.f < u.f$

Notas

Exercícios

- 22.4-1 a 22.4-5

Notas

Exercícios

22.4-1 Mostre a ordenação de vértices produzidas por `topological-sort` quando ele é executado sobre o gao da figura 22.8. Considere os vértices em ordem alfabética, e suponha que cada lista de adjacência esteja em ordem alfabética.

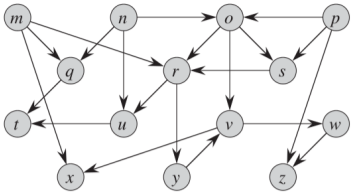


Figure 22.8 A dag for topological sorting.

Notas

Exercícios

22.4-2 Forneça um algoritmo de tempo linear que receba com entrada um grafo acíclico orientado $G = (V, E)$ e dois vértices s e t, e retorne o número de caminhos de s para t em G. Por exemplo, no gao da figura 22.8, existem exatamente quatro caminhos do vértice p para o vértice v: pov, $poryv$, $posryv$ e $psryv$. Seu algoritmo só precisa contar os caminhos, não listá-los.

22.4-3 Forneça uma algoritmo que determine se um dado grafo não orientado $G = (V, E)$ contém um ciclo. Seu algoritmo deve ser executado no tempo $O(V)$, independente de E.

Notas

Exercícios

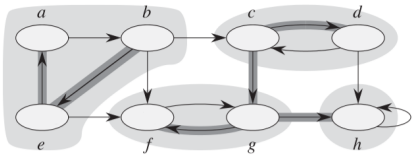
22.4-4 Prove ou conteste: se um grafo orientado G contém ciclos, então `topological-sort(G)` produz uma ordenação de vértices que minimiza o número de arestas "ruins" que são incompatíveis com a ordenação produzida.

22.4-5 Outro modo de executar a ordenação topológica sobre um grafo acíclico orientado $G = (V, E)$ é encontrar repetidamente um vértice de grau de entrada 0, colocá-lo na saída e removê-lo do grafo, bem como todas as suas arestas de saída. Explique como implementar essa ideia, de tal forma que ela seja executada no tempo $O(V + E)$. O que acontecerá a esse algoritmo se G tiver ciclos?

Notas

Componentes fortemente conexos

- Um **componente fortemente conectado** (SCC) de um grafo orientado $G = (V, E)$ é um conjunto máximo de vértices $C \subseteq V$, tal que, para todo par de vértice u e v
  - $u \rightsquigarrow v$
  - $v \rightsquigarrow u$



Notas

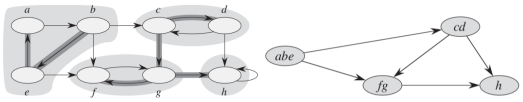
Componentes fortemente conexos

- Algoritmo usa a transposta de G
  - $G^T = (V, E^T), E^T = \{(u, v) : (v, u) \in E\}$
  - G^T é G com todas as arestas invertidas
  - G^T pode ser calculado em tempo $\Theta(V + E)$ para a representação de lista de adjacências
  - G e G^T tem os mesmos SCC's

Notas

Componentes fortemente conexos

- Grafo de componentes
  - $G^{SCC} = (V^{SCC}, E^{SCC})$
  - V^{SCC} tem um vértice para cada SCC em G
  - E^{SCC} contém uma aresta se existe uma aresta correspondente entre os SCC's de G



Notas

Componentes fortemente conexos

- Lema 22.13
  - G^{SCC} é um gao
  - Sejam C e C' SCC distintos em G, seja $u, v \in C$ e seja $u', v' \in C'$. Suponha que exista um caminho $u \rightsquigarrow u'$ em G. Então, não pode existir um caminho $v' \rightsquigarrow v$ em G
  - Prova: Suponha que exista um caminho $v' \rightsquigarrow v$ em G. Então existem caminhos $u \rightsquigarrow u' \rightsquigarrow v'$ e $v' \rightsquigarrow v \rightsquigarrow u$ em G. Portanto, u e v' são acessíveis um a partir do outro, e não podem estar em SCC separados

Notas

Componentes fortemente conexos

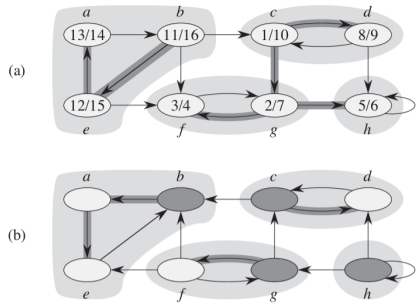
```
strongly-connected-components(G)
1 chamar DFS(G) para calcular o tempo de término u.f
  para cada vértice u
2 calcular GT
3 chamar DFS(GT) mas, no laço principal de DFS, considerar
  os vértices em ordem decrescente de u.f
4 os vértices de cada árvore na floresta primeiro na
  profundidade formada na linha 3 formam um componente
  fortemente conectado
```

Qual é o tempo de execução do algoritmo? $\Theta(V + E)$

Notas

Componentes fortemente conexos

Exemplo da execução do algoritmo `strongly-connected-components`



Por que este algoritmo funciona?

Notas

Componentes fortemente conexos

- Ideia
  - Considerando os vértices na segunda execução do DFS na ordem decrescente dos tempos de términos obtidos na primeira execução do DFS, estamos visitando os vértices do grafo de componentes na ordem topológica
- Vamos definir duas questões de notação
  - As referências a $u.d$ e $u.f$ se referem aos valores do primeiro DFS
  - Para um conjunto $U \subseteq V$, definimos
    - $d(U) = \min_{u \in U}\{u.d\}$ (tempo de descoberta mais antigo)
    - $f(U) = \max_{u \in U}\{u.f\}$ (tempo de término mais recente)

Notas

Componentes fortemente conexos

- Lema 22.14
  - Sejam C e C' SCC distintos em $G = (V, E)$. Suponha que exista uma aresta $(u, v) \in E$, tal que $u \in C$ e $v \in C'$. Então $f(C) > f(C')$
- Corolário 22.15
  - Sejam C e C' SCC distintos em $G = (V, E)$. Suponha que exista uma aresta $(u, v) \in E^T$, tal que $u \in C$ e $v \in C'$. Então $f(C) < f(C')$

Notas

Componentes fortemente conexos

- Teorema 22.16: `strongly-connected-components(G)` calcula corretamente os SCC de um grafo orientado G
  - A segunda DFS, começa com um SCC C tal que $f(C)$ é máximo
  - Seja $x \in C$ o vértice inicial, a segunda DFS visita todos os vértices de C. Pelo corolário, como $f(C) > f(C')$ para todo $C \neq C'$, não existe aresta de C para C'. Logo, a DFS visita apenas os vértices de C (descobrindo este SCC)
  - A próxima raiz escolhida na segunda DFS está em um SCC C' tal que $f(C')$ é máximo em relação a todos os outros SCC (sem considerar C). A DFS visita todos os vértices de C', e as únicas arestas fora de C' vão para C, cujo os vértices já foram visitados
  - O processo continua até que todos os vértices sejam visitados
  - Cada vez que uma raiz é escolhida pelo segunda DFS, ele só pode alcançar
    - vértices no SCC dele (através de arestas da árvore)
    - vértices que já foram visitados na segunda DFS

Notas

Exercícios

- 22.5-1 a 22.5-3, 22.5-5 a 22.5-7

Notas

Exercícios

22.5-1 De que maneira o número de componentes fortemente conectados de um grafo se altera se uma nova aresta é adicionada?

22.5-2 Mostre como o procedimento `strongly-connected-components` funciona sobre o grafo da figura 22.6. Especificamente, mostre os tempos de término calculados na linha 1 e a floresta produzida na linha 3. Suponha que o laço das linhas de 5 a 7 de dfs considere os vértices em ordem alfabética e que as listas de adjacências estejam em ordem alfabética.

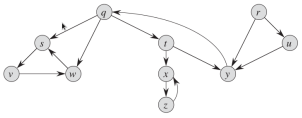


Figure 22.6 A directed graph for use in Exercises 22.3-2 and 22.5-2.

Notas

Exercícios

22.5-5 Forneça um algoritmo de tempo $O(V + E)$ para calcular o grafo de componentes de um grafo orientado $G(V, E)$. Certifique-se de que existe no máximo uma aresta entre dois vértices no grafo de componentes que o seu algoritmo produz.

Notas

Referências

- Thomas H. Cormen et al. Introdução a Algoritmos. 2ª edição em português. Capítulo 22.

Notas

Notas

Notas

Notas