Busca em profundidade

Marco A L Barbosa
malbarbo.pro.br

Departamento de Informática
Universidade Estadual de Maringá

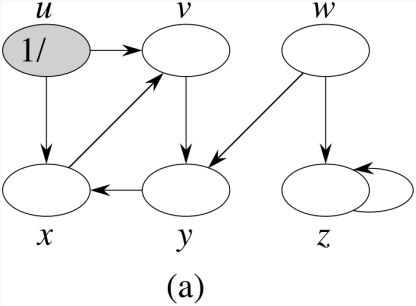
Procurar “mais fundo” no grafo sempre que possível (*Depth-first search* - DFS em inglês)

- As arestas são exploradas a partir do vértice *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

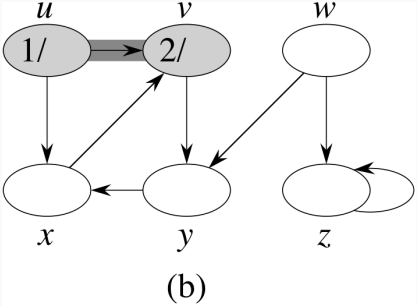
Durante a execução do algoritmo, diversos atributos são definidos para os vértices

- Cor (*cor*): cada vértice inicialmente é branco. Quando um vértice é descoberto ele é colorido de cinza. Quando a lista de adjacências de um vértice é totalmente explorada, o vértice é colorido de preto.
- Pai (π): quando um vértice v é descoberto a partir de um vértice u, o campo predecessor $v.\pi = u$ é definido
- Carimbo de tempo (d e f): cada vértice tem dois carimbos de tempo $v.d$ (quando o vértice é descoberto) e $v.f$ (quando o vértice é terminado)

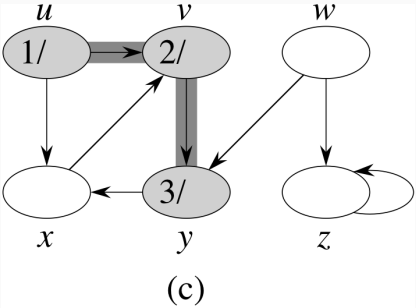
Exemplo de execução

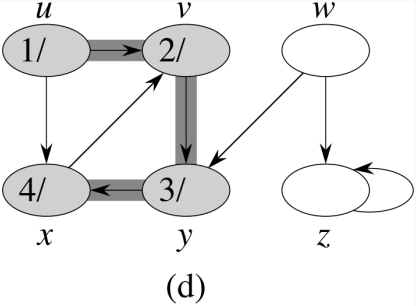


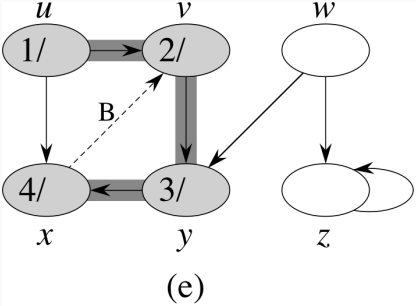
Exemplo de execução

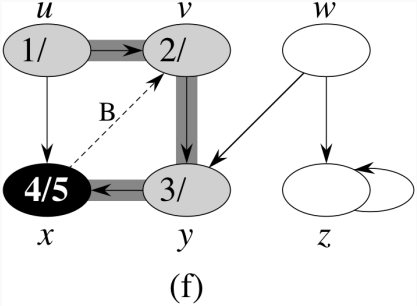


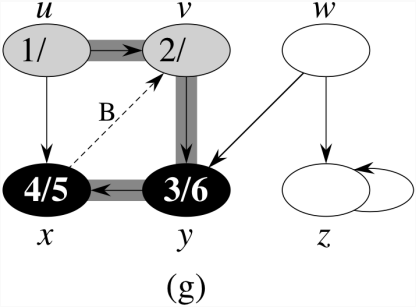
Exemplo de execução

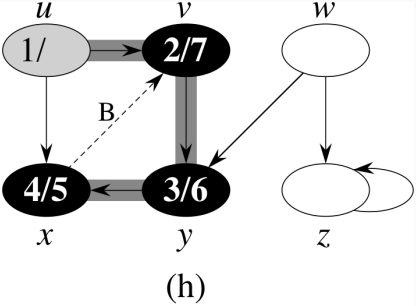


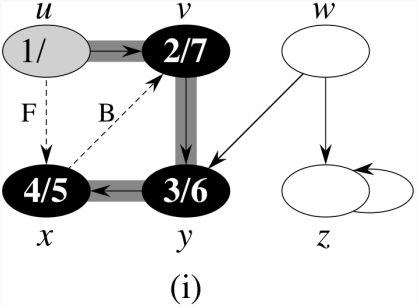

u
v
w
1/
2/
4/
3/
x
y
z
(d)

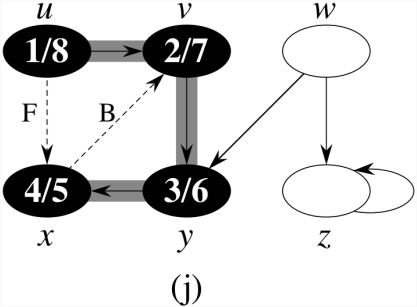

u
v
w
1/
2/
B
4/
3/
x
y
z
(e)

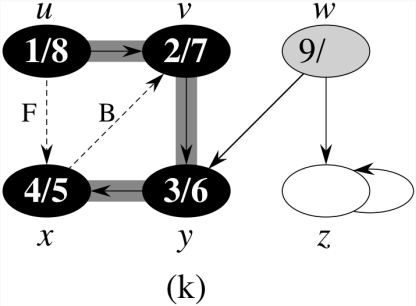

u
v
w
1/
2/
B
4/5
3/
x
y
z
(f)

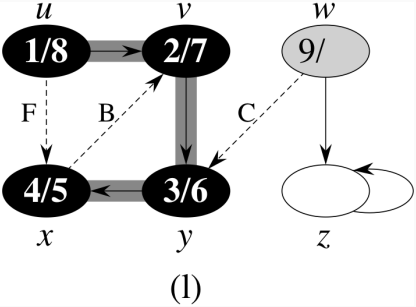

u
v
w
1/
2/
B
4/5
3/6
x
y
z
(g)

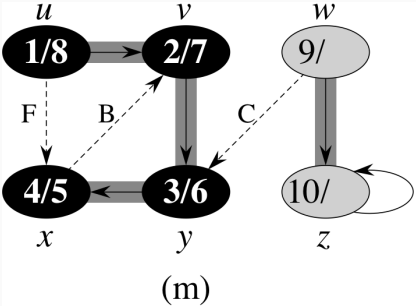

u
v
w
1/
2/7
B
4/5
3/6
x
y
z
(h)

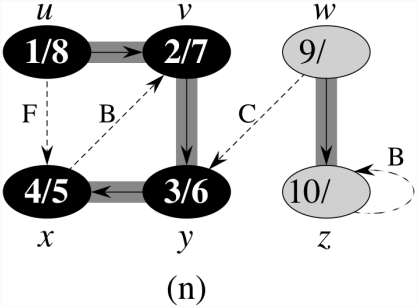

u
v
w
1/
2/7
F
B
4/5
3/6
x
y
z
(i)

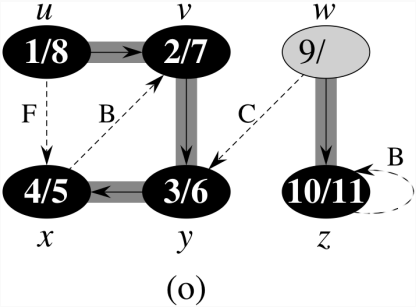

u
v
w
1/8
2/7
F
B
4/5
3/6
x
y
z
(j)

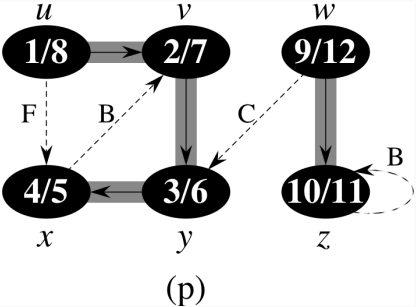

u
v
w
1/8
2/7
9/
F
B
4/5
3/6
x
y
z
(k)


u
v
w
1/8
2/7
9/
F
B
C
4/5
3/6
x
y
z
(l)


u
v
w
1/8
2/7
9/
F
B
C
4/5
3/6
10/
x
y
z
(m)


u
v
w
1/8
2/7
9/
F
B
C
B
4/5
3/6
10/
x
y
z
(n)


u
v
w
1/8
2/7
9/
F
B
C
B
4/5
3/6
10/11
x
y
z
(o)


u
v
w
1/8
2/7
9/12
F
B
C
B
4/5
3/6
10/11
x
y
z
(p)

Procedimento DFS

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.cor = BRANCO
3      u.π = NIL
4  tempo = 0
5  for u ∈ G.V
6      if u.cor == BRANCO
7          DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
 1  tempo = tempo + 1
 2  u.d = tempo
 3  u.cor = CINZA
 4  for each vertex v ∈ G.Adj[u]
 5      if v.cor == BRANCO
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.cor = PRETO
 9  tempo = tempo + 1
10  u.f = tempo
```

Tempo de execução

- Os laços nas linhas 1 a 3 e nas linhas 5 a 7 de DFS demoram tempo $\Theta(V)$, sem contar o tempo das chamadas a DFS-VISIT
- O procedimento DFS-VISIT é chamado exatamente uma vez para cada vértice, isto porque DFS-VISIT é chamado para os vértices brancos, e no início de DFS-VISIT o vértice é pintado de cinza
- Durante a execução de DFS-VISIT(v), o laço nas linhas 4 a 7 é executado $G.Adj[v]$ vezes, como $\sum_{v \in V} |G.Adj[v]| = \Theta(E)$, o custo total da execução das linhas 4 a 7 de DFS-VISIT é $\Theta(E)$
- Portanto, o tempo de execução do DFS é $\Theta(V + E)$

Propriedades

A busca em profundidade produz informações interessantes sobre a estrutura do grafo.

Estas informações podem ser usadas diretamente ou na construção de outros algoritmos.

Vamos ver algumas dessas informações.

O procedimento DFS constrói uma floresta da busca em profundidade (floresta DFS), contendo diversas árvores da busca em profundidade.

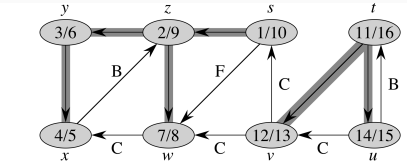
Para um grafo $G = (V, E)$, definimos o **subgrafo predecessor** de uma busca em profundidade de G como o grafo $G_\pi = (V, E_\pi)$ onde $E_\pi = \{(v.\pi, v) : v \in V \text{ e } v.\pi \neq \text{NIL}\}$.

As arestas em E_π são **arestas da floresta**.

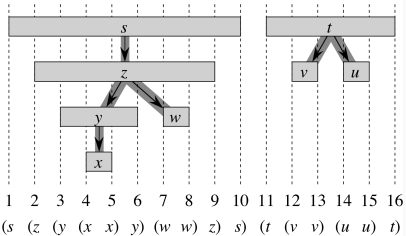
O tempo de descoberta e término dos vértices têm estrutura de parênteses.

Quando representamos a descoberta de um vértice v por "$(v$" e o término por "$v)$", então a sequência de descobertas e términos gera uma expressão bem formada (os parênteses estão certos).

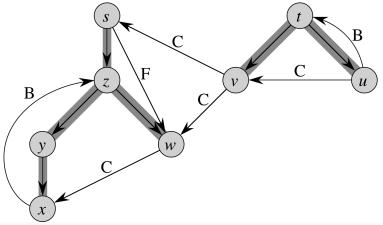
(a)


y
z
s
t
3/6
2/9
1/10
11/16
B
F
C
B
4/5
7/8
12/13
14/15
x
w
v
u
C
C
C

(b)


s
t
z
v
u
y
w
x
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
(s (z (y (x x) y) (w w) z) s) (t (v v) (u u) t)

(c)


s
t
C
B
z
F
v
C
u
B
C
y
w
C
x

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
- O intervalo $[u.d, u.f]$ está contido inteiramente no intervalo $[v.d, v.f]$ e u é descendente de v em uma árvore DFS
- O intervalo $[v.d, v.f]$ está contido inteiramente no intervalo $[u.d, u.f]$ e v é descendente de u em uma árvore DFS

Prova

Primeiro observamos que $u.d < u.f$ para todo vértice u (22.2).

É verdade que $u.d < v.d$? Se sim, é verdade que $v.d < u.f$?

- Se sim, qual era a cor de u quando v foi descoberto? CINZA. v é descente de u? Sim. Então todas as arestas que saem de v são exploradas e v é finalizado antes que a busca retorne para finaliza u. Logo $v.f < u.f$ e o intervalo $[v.d, v.f]$ está inteiramente contido no intervalo $[u.d, u.f]$.
- Se não, $u.f < v.d$ e pela equação 22.2 $u.d < u.f < v.d < v.f$. Portanto os intervalos $[u.d, u.f]$ e $[v.d, v.f]$ são disjuntos.

Se não ($v.d < u.d$), temos a mesma situação anterior, mas o u faz o papel de v e v faz o papel do u.

Corolário 22.8 (Aninhamento dos intervalos de descendentes)

Um vértice v é um descendente próprio de u na floresta DFS de um grafo G se e somente se $u.d < v.d < v.f < u.f$.

Prova

Direta do Teorema 22.7.

Teorema 22.9 (Teorema do caminho de vértices brancos)

Na floresta DFS de um grafo $G = (V, E)$, um vértice v é descende de um vértice u se e somente se no momento $u.d$, que a busca descobre u, existe um caminho de u para v que consiste apenas de vértices brancos.

Prova

Exercício: leia e compreenda a prova no livro.

Podemos definir quadro tipos de arestas em termos da floresta G_π

- **Arestas de árvore**, são as arestas em G_π. Uma aresta (u, v) é uma aresta de 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 em uma árvore DFS (consideramos laços como arestas de retorno)
- **Arestas diretas** são as arestas (u, v) que não são arestas da árvore e conectam o vértice u a um descendente v em uma árvore DFS
- **Arestas cruzadas** são todas as outras arestas

Classificação das arestas

Quando uma aresta (u, v) é explorada, a cor do vértice v nos indica o tipo de aresta:

- BRANCO - aresta da árvore,
- CINZA - aresta de retorno, e
- PRETO - aresta direta ou cruzada (como diferenciar as duas?)

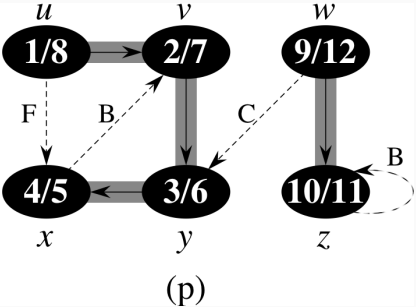
Teorema 22.10

Na busca em profundidade de um grafo não orientado G, cada aresta de G ou é uma aresta de árvore ou uma aresta de retorno.

Exercício

- Faça exemplos de grafos não orientados;
- Execute o DFS e classifique as arestas;
- Leia e compreenda a prova no livro.

22.3-3 Mostre a estrutura de parênteses da busca em profundidade da Figura 22.4.



22.3-9 Dê um contraexemplo para a seguinte conjectura: se um grafo direcionado G contém um caminho de u para v, então qualquer busca em profundidade resulta em $v.d \leq u.f$.

22.3-11 Explique como um vértice u de um grafo direcionado pode acabar em uma árvore do DFS contendo apenas u, mesmo u tendo arestas de entrada e saída.

22.3-13 Um grafo direcionado G é **singularmente conexo** se $u \rightsquigarrow v$ implica que G contém no máximo um caminho simples de u para v para todos os vértices $u, v \in V$. Projete um algoritmo eficiente para determinar se um grafo é singularmente conectado ou não.

Como resolver este problema?

O primeiro passo é termos certeza que entendemos o problema.

E Depois? Fazer exemplos de grafos que são e não são singularmente conexos. Note que os exemplos também ajudam na compreensão do problema.

E depois? Como estamos no capítulo do DFS, podemos executar o DFS em cada exemplo e tentar identificar características nos resultados do DFS que possam ser usados para classificar o grafo e então criar uma hipótese (algoritmo).

E depois? Antes de tentar provar que a hipótese é verdadeira, tentamos encontrar contra exemplos que mostre que ela não é verdadeira. Se encontrarmos contraexemplos, ajustamos a hipótese (algoritmo) e repetimos o processo.

Apos isso temos uma hipótese que não conseguimos mostrar que é falsa encontrado contraexemplos. Então a nossa solução está correta e resolvemos o problema? Não! Agora precisamos provar (argumentar) que o nosso algoritmo está correto. Como fizemos diversos exemplos e possíveis contraexemplos, ganhamos intuição do porque o algoritmo funciona e estamos mais preparados para fazer uma argumentação formal.

Exercícios

Veja a lista completa de exercícios e algumas soluções na página da disciplina.

Referências

Thomas H. Cormen et al. Introduction to Algorithms. 3rd edition. Capítulo 22.3.