Busca em largura

Marco A L Barbosa
malbarbo.pro.br

Departamento de Informática
Universidade Estadual de Maringá

Este trabalho está licenciado com uma Licença Creative Commons - Atribuição-CompartilhaIgual 4.0 Internacional.
http://github.com/malbarbo/na-grafos

Dado um grafo $G = (V, E)$ e um vértice de origem s, a **busca em largura** (*Breadth-first search* - BFS, em inglês) explora sistematicamente as arestas de G até descobrir cada vértice acessível a partir de s.

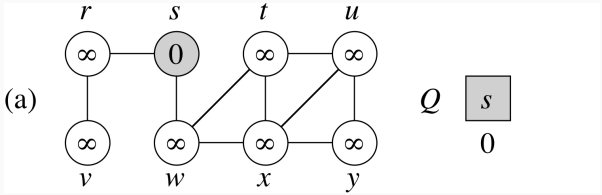
- Calcula a distância mínima (menor número de arestas) de s até todos os vértices acessíveis a partir de s
- Produz uma árvore de busca em largura
- É a base para outros algoritmos
- Funciona para grafos orientados e não orientados

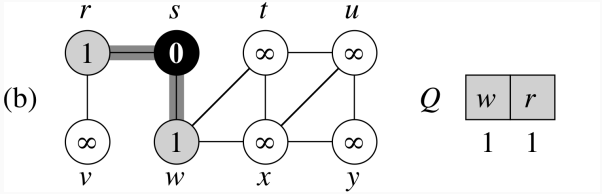
Recebe este nome porque expande a fronteira entre vértices descobertos e não descobertos uniformemente ao longo da extensão da fronteira. Descobre todos os vértices de distância k de s antes de descobrir quaisquer vértices de distância $k + 1$.

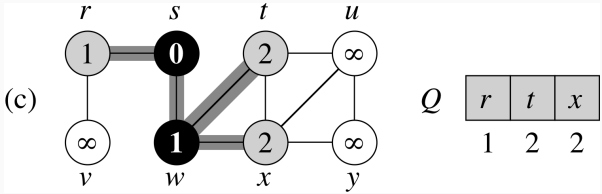
Durante a execução do algoritmo, diversos atributos nos vértices são utilizados

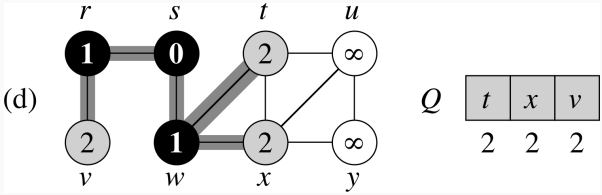
- Cor (*cor*): cada vértice inicialmente é branco. Quando um vértice é **descoberto** (encontrado pela primeira vez na busca) ele é colorido de cinza. Quando a lista de adjacências de um vértice é totalmente explorada, o vértice é colorido de preto.
- Pai (π): quando um vértice v é descoberto através da aresta (u, v), dizemos que o vértice u é **predecessor** ou **pai** de v. O atributo π é utilizado para armazenar o predecessor de cada vértice.
- Distância (d): a distância mínima em relação ao vértice inicial.

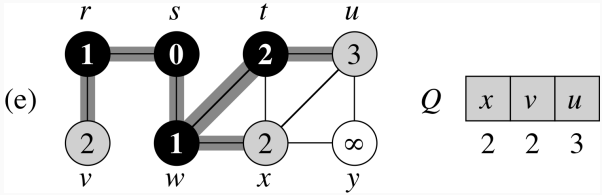
Exemplo de execução



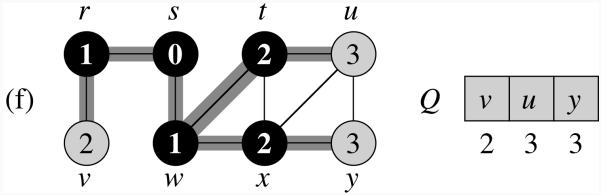

(b)
r
s
t
u
1
0
∞
∞
∞
1
∞
∞
v
w
x
y
Q
w
r
1
1


(c)
r
s
t
u
1
0
2
∞
∞
1
2
∞
v
w
x
y
Q
r
t
x
1
2
2

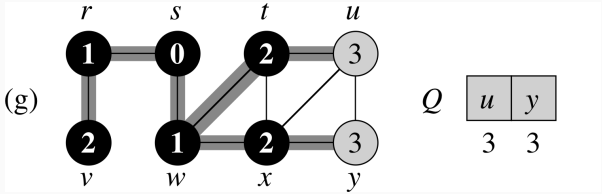

(d)
r
s
t
u
1
0
2
∞
2
1
2
∞
v
w
x
y
Q
t
x
v
2
2
2

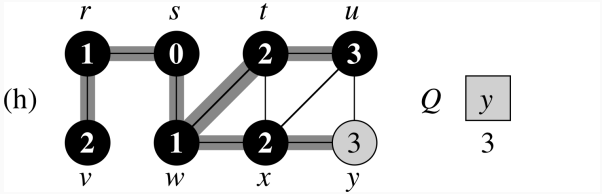

(e)
r
s
t
u
1
0
2
3
2
1
2
∞
v
w
x
y
Q
x
v
u
2
2
3

Exemplo de execução

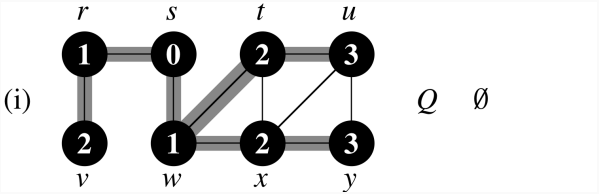


Exemplo de execução




(h)
r
s
t
u
1
0
2
3
2
1
2
3
v
w
x
y
Q
y
3

Exemplo de execução



Baseado nesta ideia de funcionamento, vamos escrever o algoritmo BFS. Vamos assumir que o grafo de entrada é representado por uma lista de adjacências.

Nós construímos em sala o algoritmo “do zero”. Como exercício, tente escrever novamente o algoritmo, mas desta vez sozinho!

BFS(G, s)

```
1  for each vertex v ∈ G.V − {s}
2      v.d = ∞
3      v.π = NIL
4      v.cor = BRANCO
5  s.d = 0
6  s.π = NIL
7  s.cor = CINZA
8  Q = ∅
9  ENQUEUE(Q, s)
10 while Q ≠ ∅
11     u = DEQUEUE(Q)
12     for each vertex v ∈ G.Adj[u]
13         if v.cor == BRANCO
14             v.cor = CINZA
15             v.d = u.d + 1
16             v.π = u
17             ENQUEUE(Q, v)
18     u.cor = PRETO
```

Tempo de execução

- 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 tem tempo $O(1)$, assim, 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, desta forma, no máximo uma vez
- Como a soma dos comprimentos das listas de adjacências é $\Theta(E)$, o tempo para percorrer todas as listas é no máximo $O(E)$
- O tempo de inicialização é $O(V)$
- Tempo total de execução do BFS é $O(V + E)$

Afirmamos no início dos slides que o procedimento BFS encontra as distâncias mínimas de s para todos os vértices acessíveis a partir de s. Vamos ver porque isto é verdade.

Definimos como a **distância do caminho mínimo** $\delta(s, v)$ de s para v como o número mínimo de arestas de qualquer caminho de s para v, se não existe caminho de s para v então $\delta(s, v) = \infty$.

Chamamos um caminho de comprimento $\delta(s, v)$ entre s e v de **caminho mínimo**.

Seja $G = (V, E)$ um grafo orientado ou não orientado e $s \in V$ um vértice arbitrário.

Lema 22.1

Se $(u, v) \in E$, então $\delta(s, v) \leq \delta(s, u) + 1$.

Prova

u é acessível a partir de s?

- Se sim, então v também é. Além disso, o menor caminho entre s e v não pode ser maior que o menor caminho de s para u seguido de (u, v), então a desigual se mantém.
- Se não, $\delta(s, u) = \infty$ e a desigualdade se mantém. $\square$

Note que está é uma propriedade da definição de menor caminho e não do algoritmo em si.

Precisamos mostrar que o valor $v.d$ de cada vértice do grafo, produzido pela chamada BFS(G, s), é igual a $\delta(s, v)$.

Vamos começar mostrando um limite inferior para $v.d$.

Lema 22.2

Se BFS(G, s) é executado, então após a execução
$v.d \geq \delta(s, v)$ para todo vértice $v \in V$.

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

Prova

Vamos usar indução no número de operações ENQUEUE. A nossa hipótese de indução é que $v.d \geq \delta(s, v)$ para todo $v \in V$.

Base (imediatamente após a linha 9)

- Para o vértice inicial s, temos $s.d = 0 = \delta(s, s)$. Para todos os vértices $v \in V - \{s\}$, temos $v.d = \infty \geq \delta(s, v)$, então a hipótese é verdadeira.

Passo de indução

- Considere um vértice branco v que é descoberto a partir de um vértice u
- Pela hipótese de indução $u.d \geq \delta(s, u)$
- Pela linha 15, temos $v.d = u.d + 1$, logo $v.d \geq \delta(s, u) + 1$
- Como pelo lema 22.1 $\delta(s, u) + 1 \geq \delta(s, v)$, então $v.d \geq \delta(s, v)$
- Como a cor de v é alterado para CINZA as linhas de 14-17 não serão mais executadas para v e a sua distância não será mais alterada, portando a hipótese de indução se mantém. □

Lema 22.3

Se em um determinado momento da execução de BFS(G, s) a fila Q contém $\langle v_1, v_2, \ldots, v_r \rangle$, então

- $v_r.d \leq v_1.d + 1$; e
- $v_i.d \leq v_{i+1}.d$ para $i = 1, 2, \ldots, r - 1$

Prova

Indução no número de operações com a fila.

Exercício: leia a prova no livro (tente entender frase por frase, se está difícil entender uma afirmação, tente ver a afirmação como pergunta)

Corolário 22.4

Se v_i é inserido antes de v_j na fila durante a execução do BFS, então $v_i.d \leq v_j.d$ no momento que v_j é inserido na fila.

Prova

Direto do Lema 22.3 e do fato que cada vértice tem o valor de d atribuído no máximo uma vez.

Teorema 22.5 - Corretude do BFS

Durante a execução do BFS(G, s), todos os vértices acessíveis a partir de s são descobertos, e depois do término da execução $v.d = \delta(s, v)$ para todo $v \in V$.

Prova

Por contradição.

Corretude

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

Vamos assumir, com o propósito de contradição, que algum vértice v receba um d diferente de $\delta(s, v)$. Seja v o vértice com o menor $\delta(s, v)$ que recebeu o d incorreto.

- O vértice v pode ser igual a s? Não, pois o valor $s.d = 0$ está correto.
- O vértice v é acessível a partir de s? Sim, porque senão o algoritmo teria determinado que $v.d = \infty$ que é $\delta(s, v)$ e portanto está correto.
- Sabendo que v é acessível a partir de s e $v \neq s$, seja u o vértice predecessor de v em um menor caminho de s para v. Temos que $\delta(s, v) = \delta(s, u) + 1$.
- O algoritmo calculou $u.d$ corretamente, isto é, $u.d = \delta(s, u)$? Sim, pois $\delta(s, u) < \delta(s, v)$ e v é o vértice com menor δ que recebeu d incorreto.
- Do Lema 22.2 sabemos que $v.d \geq \delta(s, v)$. Será que $v.d$ pode ser igual a $\delta(s, v)$? Não! Logo $v.d > \delta(s, v)$.
- Juntando estas propriedades temos $v.d > \delta(s, v) = \delta(s, u) + 1 = u.d + 1$

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

Sabendo que $v.d > u.d + 1$ (Eq 22.1), vamos analisar o momento em que o BFS retira u da fila (linha 11). Nesse momento o vértice v pode ser BRANCO, CINZA ou PRETO. Vamos mostrar que em cada um desses casos nos derivamos uma contradição para a equação 22.1.

- Se v é BRANCO, qual é o valor que o BFS atribui para $v.d$? O valor $u.d + 1$ (linha 15), que contradiz a Eq 22.1.
- Se v é PRETO então ele já foi removido da fila, como $v.d$ se compara com $u.d$? Pelo Corolário 22.4 $v.d \leq u.d$, contrariando a Eq 22.1.
- Se v é CINZA, então ele foi pintado de cinza quando um vértice w estava sendo explorado e $v.d = w.d + 1$. w foi removido antes de u da fila? Sim, e portanto, pelo Corolário 22.4, $w.d \leq u.d$.
- Logo $v.d = w.d + 1 \leq u.d + 1$, mais uma vez um contradição com a Eq 22.1.

Então não pode existir um vértice v cujo o d foi calculado incorretamente, portanto concluímos que $v.d = \delta(s, v)$ para todo $v \in V$. □

BFS constrói uma árvore de busca em largura.

A árvore é definida pelo campo pai (π) em cada vértice.

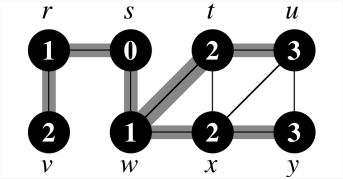
Para um grafo $G = (V, E)$ e um vértice de origem s, definimos o **subgrafo predecessor** de G como $G_\pi = (V_\pi, E_\pi)$ onde

- $V_\pi = \{v \in V : v.\pi \neq \text{NIL}\} \cup \{s\}$ (vértices acessíveis a partir de s)
- $E_\pi = \{(v.\pi, v) : v \in V_\pi - \{s\}\}$ (arestas que conectam os vértices acessíveis a partir de s - com exceção do próprio s - ao pai)

O subgrafo predecessor G_π é uma árvore de busca em largura

- 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 um caminho mais curto de s até v em G

Uma árvore de busca em largura é de fato uma árvore, pois é conexa e $|E_\pi| = |V_\pi| - 1$

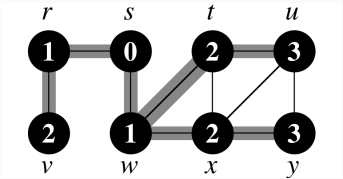


Como exibir o caminho de s para v?

Quais são os casos mais simples?

- $s = v$, qual é a saída? s.
- $v.\pi$ == NIL, qual é saída? Indicação que não existem caminho.

E os outros casos? Se tivermos o caminho de s até o predecessor de v (que é $v.\pi$), então basta adiciona o v no caminho.




```
PRINT-PATH(G, s, v)
1  if v == s
2        PRINT s
3  elseif v.π == NIL
4        PRINT "não existe nenhum caminho de" s para v
5  else
6        PRINT-PATH(G, s, v.π)
7        PRINT 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.

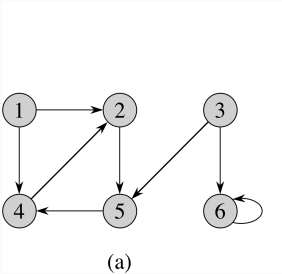
Exercícios

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

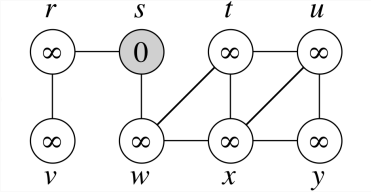
É possível simplificar o BFS de forma que após a sua execução ainda possamos identificar os vértices acessíveis a partir do vértice inicial?

- Podemos remover o cálculo do valor d? Sim, o fluxo de execução do algoritmo não depende de d.
- Podemos remover o cálculo do valor π? Sim, o fluxo de execução do algoritmo não depende de π.
- Podemos remover a linha 18? Sim, o fluxo de execução é determinado por apenas duas situações, vértice branco ou não branco, não é necessário diferenciar entre cinza e preto.

22.2-1 Mostre os valores de d e π resultantes da execução da busca em largura no grafo direcionado da Figura 22.2, usando o vértice 3 como inicial.



22.2-5 Argumente que em uma busca em largura, o valor de $u.d$ atribuído para um vértice u é independente da ordem que os vértices aparecem em cada lista de adjacência. Usando a Figura 22.3 como exemplo, mostre que a árvore produzida pelo BFS pode depender da ordem dentro das listas de adjacências.



Exercícios

Usando como base o BFS, explique como

1) Verificar se um grafo não orientado é conexo.
2) Identificar quantas componentes conexas um grafo não orientado tem.
3) Verificar se um grafo não orientado é bipartido.

Dicas

- Faça diversos exemplos de grafos com e sem as características desejadas e mostre o resultado da execução do BFS neles
- Baseado nesses exemplos, tente identificar aspectos nos resultados que possam ajudar na solução das questões

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 22.2.

Rascunho da prova de corretude do BFS.