Busca em largura x busca em profundidade

Algoritmos em grafos

Professor: Rennan Dantas

Universidade Federal do Ceará
Campus de Crateús

04 de abril de 2022

Por que grafos?

- Uma ampla variedade de problemas pode ser expressa com clareza e precisão na linguagem dos grafos
- Por exemplo, considere a tarefa de colorir um mapa político
- Qual o número de cores necessário, com a restrição que países vizinhos devem ter cores diferentes?
- Modelo convencional é normalmente carregada de informação irrelevante

Representação de Grafos

- Podemos escolher entre dois padrões para representar um grafo $G = (V, E)$: como uma coleção de listas de adjacências ou como matriz de adjacências
- Qualquer desses modos se aplica a grafos dirigidos (direcionados) e não dirigidos (não direcionados)
- Como a representação por listas de adjacências nos dá um modo compacto de representar grafos esparsos - aqueles para os quais $|E|$ é muito menor que $|V|^2$ - ela é, em geral, o método preferido
- A maioria dos algoritmos que veremos supõe que o grafo de entrada é representado sob a forma de lista de adjacências
- Contudo, uma representação por matriz de adjacências pode ser preferível quando o grafo é denso - quando $|E|$ está próximo de $|V|^2$ - ou quando precisamos saber rapidamente se há uma aresta conectando dois vértices dados

Representação por lista de adjacências - Grafo não direcionado

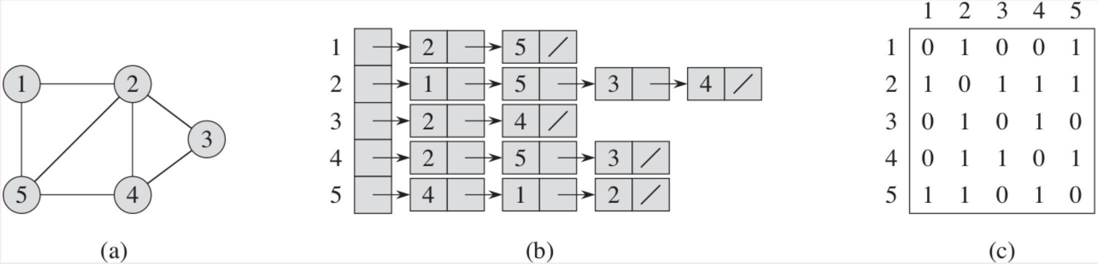

Figura: Fonte: Livro Algoritmos - Cormen

Representação por lista de adjacências - Grafo direcionado

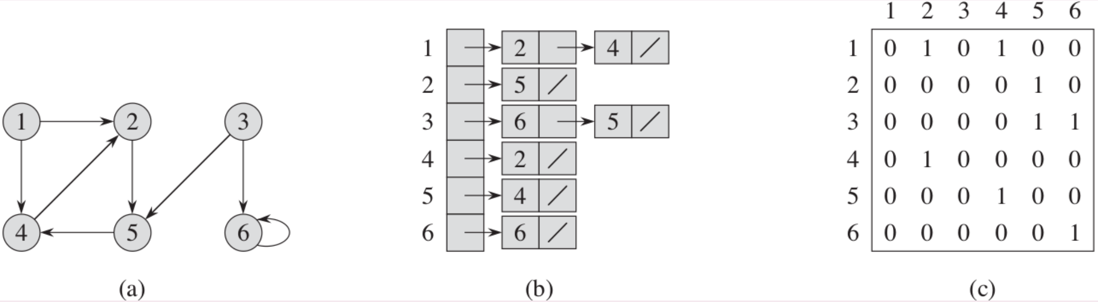

Figura: Fonte: Livro Algoritmos - Cormen

Representação de grafos

- Se um grafo G for dirigido, a soma dos comprimentos de todas as listas de adjacências é $|E|$, já que uma aresta da forma (u, v) é representada fazendo com que v apareça em $Adj[u]$
- Se um grafo G não for dirigido, a soma dos comprimentos de todas as listas é $2|E|$
- Quantidade de memória exigida é $\Theta(V + E)$

Representação de grafos

- Podemos adaptar imediatamente as listas de adjacências para representar grafos ponderados, isto é, grafos nos quais cada aresta tem um peso associado, normalmente dado por uma função peso w:$E \rightarrow \mathbb{R}$
- Por exemplo, seja $G = (V, E)$ um grafo ponderado com função peso w
- Simplesmente armazenamos o peso $w(u, v)$ da aresta $(u, v) \in E$ com o vértice v na lista de adjacência de u
- A representação por lista de adjacências é bastante robusta no sentido de que podemos modificá-la para suportar outras variantes de grafos

Representação de grafos

- Uma desvantagem potencial de representação por lista de adjacências é que ela não proporciona nenhum modo mais rápido de determinar se uma dada aresta (u, v) está presente no grafo do que procurar v na lista de adjacência $Adj[u]$
- Essa desvantagem pode ser contornada por uma representação por matriz de adjacências do grafo, porém ao custo de utilizar assintoticamente mais memória

Representação de grafos

- No caso da representação por matriz de adjacências de um grafo $G = (V, E)$, supomos que os vértices são numerados $1, 2, ..., |V|$ de alguma maneira arbitrária
- Então a representação por matriz de adjacências de um grafo G consiste em uma matriz $|V|x|V|$ $A = (a_{ij})$ tal que

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

Representação de grafos

- Assim como a representação por lista de adjacências de um grafo, uma matriz de adjacências pode representar um grafo ponderado
- Podemos simplesmente armazenar o peso $w(u, v)$ da aresta $(u, v) \in E$ como a entrada na linha u e coluna v da matriz de adjacências
- Se uma aresta não existe, podemos armazenar o valor NIL como sua entrada de matriz correspondente, se bem que em muitos problemas é conveniente usar um valor como 0 ou ∞

Busca em Largura

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

Figura: Fonte: Livro Algoritmos - Cormen

Busca em Largura

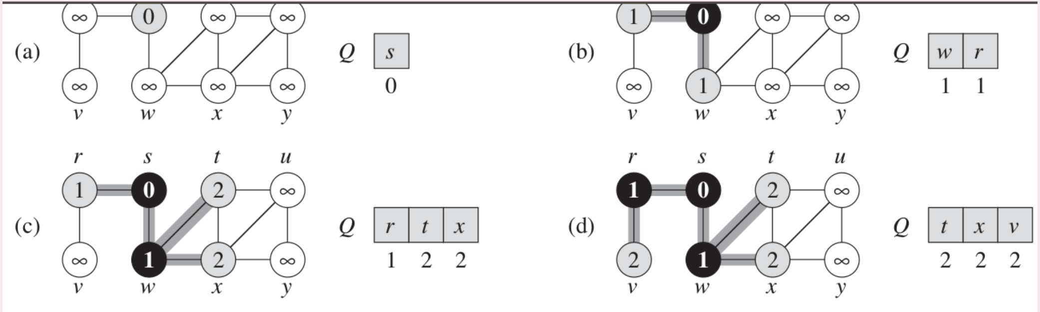

Figura: Fonte: Livro Algoritmos - Cormen

Busca em Largura

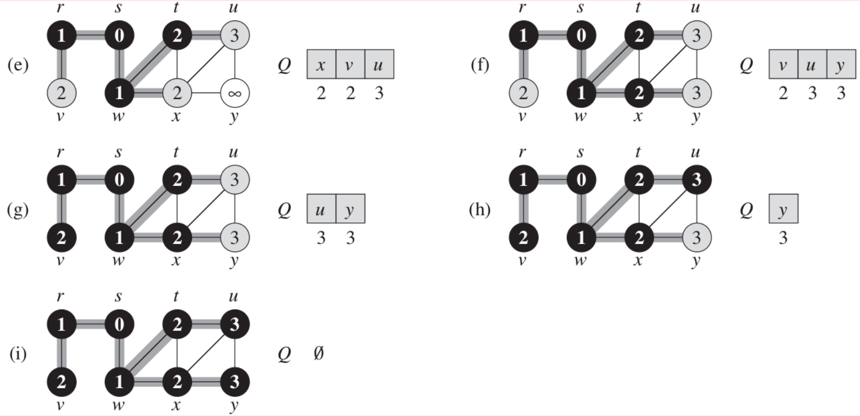

Figura: Fonte: Livro Algoritmos - Cormen

Menores caminhos

- O algoritmo de busca em largura encontra a distância de uma fonte para cada vértice alcançável no grafo $G = (V, E)$
- Definimos o caminho de distância mínima $\delta(s, v)$ de s para v como o menor número de arestas em qualquer caminho do vértice s ao vértice v
- Se não existe caminho de s para v, então $\delta(s, v) = \infty$
- Nós chamamos o caminho de comprimento $\delta(s, v)$ de s para v um menor caminho de s para v

Lema

Seja $G = (V, E)$ um grafo direcionado ou não direcionado e seja $s \in V$ um vértice arbitrário. Então, para qualquer aresta $(u, v) \in E$,

$$\delta(s, v) \leq \delta(s, u) + 1$$

Busca em Profundidade

```
DFS(G)
for each vertex u ∈ G.V
        u.color = WHITE
        u.π = NIL
time = 0
for each vertex u ∈ G.V
        if u.color == WHITE
                DFS-VISIT(G, u)
```

Figura: Fonte: Livro Algoritmos - Cormen

Busca em Profundidade

```
DFS-VISIT(G, u)
time = time + 1                // white vertex u has just been discovered
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]          // explore edge (u, v)
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK                // blacken u; it is finished
time = time + 1
u.f = time
```

Figura: Fonte: Livro Algoritmos - Cormen

Busca em Profundidade

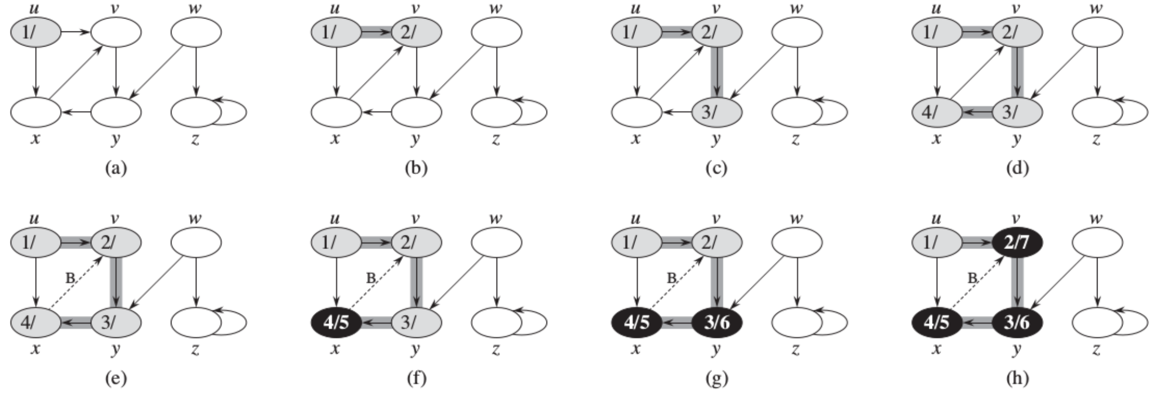

Figura: Fonte: Livro Algoritmos - Cormen

Busca em Profundidade

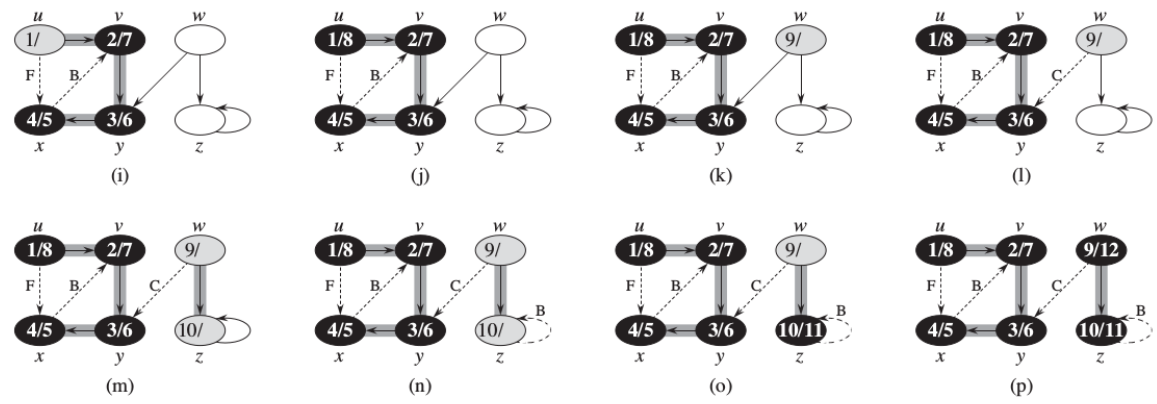

Figura: Fonte: Livro Algoritmos - Cormen

Corolário

Um vértice v é um descendente próprio de um vértice u em uma de busca em profundidade para um grafo G se e somente se $u.d < v.d < v.f < u.f$

Teorema do caminho branco

Em uma floresta em profundidade de um grafo (dirigido ou não dirigido) G=(V,E), o vértice v é um descendente do vértice u se e somente se no momento $u.d$ em que a busca descobre u, há um caminho de u a v que consiste inteiramente em vértices brancos.

Prova

- ($\Rightarrow$) Se $v = u$, então o caminho de u para v contém apenas o vértice u, o qual ainda é branco quando atribuímos o valor de $u.d$
- Agora suponha que v é um descendente próprio de u na floresta de busca em profundidade
- Pelo Corolário anterior, $u.d < v.d$ e então v é branco no tempo $u.d$
- Como v pode ser qualquer descendente de u, todos os vértices no único caminho de u para v na floresta de busca em profundidade são brancos no tempo $u.d$

Prova

- ($\Leftarrow$) Suponha que existe um caminho de vértices brancos de u para v no tempo $u.d$, mas v não se torna descendente de u na árvore de busca em profundidade
- Sem perda de generalidade, assuma que todo vértice no caminho, exceto v, se torna descendente de u
- Seja w o predecessor de v no caminho, tal que w é um descendente de u
- Pelo Corolário anterior, $w.f \leq u.f$
- Pelo fato de que v deve ser descoberto após u ser descoberto, mas antes de w ser finalizado, temos $u.d < v.d < w.f \leq u.f$
- Pelo Corolário anterior, v deve ser um descendente de u

Classificações das arestas

- Uma outra propriedade interessante da busca em profundidade é que a busca pode ser usada para classificar as arestas de um grafo G
- O tipo de cada aresta pode prover informação importante sobre o grafo
- Por exemplo: um grafo direcionado é acíclico se e somente se a busca em profundidade não gera arestas de retorno

Classificações das arestas

Podemos definir quatro tipos de arestas em termos de floresta G_π produzida por uma busca em profundidade em G:

1. **Arestas de árvore:** são arestas na floresta em profundidade G_π. A aresta (u, v) é uma aresta de árvore se v foi descoberto primeiro pela exploração da aresta (u, v).
2. **Arestas de retorno:** são arestas (u, v) que conectam um vértice u a um ancestral v em uma árvore em profundidade. Consideramos laços, que podem ocorrer grafos dirigidos, como arestas de retorno.
3. **Arestas de avanço:** não pertencem à arvore de busca em profundidade mas conectam um vértice a um descendente em uma árvore de busca em profundidade
4. **Arestas de cruzamento:** são todas as outras arestas. Podem conectar vértices na mesma árvore de busca em profundidade, ou em duas árvores diferentes

Próxima Aula

O que vem por aí?

- Ordenação topológica e conectividade
- Exercícios
- Teste 01
- Prova 01

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS DE CRATEÚS

Busca em largura x busca em profundidade

Algoritmos em grafos

Professor: Rennan Dantas

Universidade Federal do Ceará
Campus de Crateús

04 de abril de 2022