Projeto e Análise de Algoritmos

Flávio L. C. de Moura*

24 de março de 2022

1 Algoritmos em Grafos

Nesta seção estudaremos alguns algoritmos em grafos. Iniciaremos com uma sequência de definições para fixar a nomenclatura a ser utilizada.

Definição 1.1. *Um* grafo *(não dirigido) G é um par (V, E) onde V é um conjunto finito não-vazio, e E é um conjunto de pares não-ordenados de elementos de V. Em grafos não-dirigidos arestas de um vértice para ele mesmo (*auto-loop*) são proibidas, e portanto toda aresta liga dois vértices distintos.*

Definição 1.2. *Um* digrafo *(ou um grafo dirigido) G é um par (V, E) onde V é um conjunto finito não-vazio, e E é uma relação binária sobre V. Em digrafos* auto-loops *são permitidos.*

Dado um grafo $G = (V, E)$, a versão dirigida de G é o digrafo $G' = (V, E')$ onde $(u, v) \in E'$ se, e somente se $(u, v) \in E$, isto é, substituímos cada aresta $(u, v) \in E$ pelas duas arestas dirigidas (u, v) e (v, u) na versão dirigida.

Dado um digrafo $G = (V, E)$, a versão não-dirigida (ou o *grafo associado*) de G é o digrafo $G' = (V, E')$ onde $(u, v) \in E'$ se, e somente se $u \neq v$ e $(u, v) \in E$, ou seja, a versão não-dirigida de um grafo é construída removendo a direção das arestas e os auto-loops.

Um grafo (não-dirigido) é dito *completo* se qualquer par de vértices é adjacente.

Diversas definições coincidem para grafos e digrafos, mas algumas diferenças podem ocorrer dependendo do contexto. Por exemplo, se (u, v) é uma aresta de um digrafo $G = (V, E)$ então dizemos que (u, v) sai de u, e entra (ou incide) em v. Já quando (u, v) é uma aresta de um grafo, dizemos que (u, v) incide em u e v.

Definição 1.3. *O* grau *de um vértice em um grafo é o número de arestas que incidem sobre ele. Um vértice de grau 0 é dito* isolado. *Em um digrafo, o* grau de saída *(resp.* grau de entrada*) de um vértice é o número de arestas que saem (resp. chegam) neste vérice. O* grau *de um vértice em um digrafo é a soma dos seus graus de saída e entrada.*

Se (u, v) é uma aresta do (di)grafo $G = (V, E)$ então dizemos que v é adjacente a u. Note que a relação de adjacência é simétrica em grafos, mas não em digrafos. De fato, se um digrafo $G = (V, E)$ possui a aresta (u, v), mas não possui a aresta (v, u) então v é adjacente a u, mas u não é adjacente a v.

Definição 1.4. *Um* caminho *de comprimento k de um vértice u para um vértice v em um grafo $G = (V, E)$ é uma sequência $\langle v_0, v_1, \ldots, v_k \rangle$ de vértices tal que $u = v_0$ e $v = v_k$, e $(v_{i-1}, v_i) \in E$ para $i = 1, 2, \ldots, k$. O comprimento de um caminho é o número de arestas deste caminho. Existe sempre um caminho de comprimento 0 de u para u, qualquer que seja o vértice u. Um* subcaminho *de um caminho $p = \langle v_0, v_1, \ldots, v_k \rangle$ é uma sequência contígua dos vértices de p, isto é, quaisquer que sejam $0 \leq i \leq j \leq k$, a subsequência de vértices $\langle v_i, v_{i+1}, \ldots, v_j \rangle$ é um subcaminho de p.*

Quando existe um caminho p de u para v, dizemos que v é alcançável a partir de u, o que normalmente é denotado por $u \overset{p}{\rightsquigarrow} v$, quando o grafo é dirigido.

Definição 1.5. *Um caminho é dito* simples *se todos os vértices no caminho são distintos.*

*flavio@flaviomoura.info

A definição de ciclos em grafos requer cuidado porque difere para grafos e digrafos. Em um digrafo, um *ciclo* é um caminho não-nulo, ou seja, de comprimento estritamente maior do que 0, tal que o primeiro e o último vértices são idênticos. Em um digrafo, um caminho $\langle v_0, v_1, \ldots, v_k \rangle$ forma um *ciclo* se $v_0 = v_k$, e este caminho possui pelo menos uma aresta. Dois caminhos $\langle v_0, v_1, \ldots, v_{k-1}, v_0 \rangle$ e $\langle v'_0, v'_1, \ldots, v'_{k-1}, v'_0 \rangle$ formam o mesmo ciclo se existir j tal que $v'_i = v_{(i+j) \mod k}$ para $i = 0, 1, \ldots, k-1$. Um auto-loop é um ciclo de comprimento 1, e um digrafo sem auto-loops é dito *simples*. Em um grafo, as definições são similares, mas existe um requerimento adicional de que se qualquer aresta aparece mais de uma vez, então ela aparece com a mesma orientação: em um caminho $\langle v_0, v_1, \ldots v_{k-1}, v_k \rangle$, se $v_i = x$ e $v_{i+1} = y$ para $0 \leq i < k$, então não pode existir j tal que $v_j = y$ e $v_{j+1} = x$. Um ciclo é dito *simples* se seus vértices são distintos. Um (di)grafo sem ciclos é dito *acíclico*.

Um grafo acíclico é chamado de *floresta* (não-dirigida), e se o grafo for conexo então é chamado de *árvore* (livre ou não-dirigida). Um digrafo acíclico é normalmente abreviado por DAG. Nenhuma condição de conectividade é assumida em DAGs.

Definição 1.6. *Um* caminho eulerianos *em um (di)grafo conexo G é um caminho que percorre cada aresta apenas uma vez, mas vértices podem ser visitados mais de uma vez. Um* caminho hamiltoniano *em um (di)grafo G é um caminho simples que contém cada vértice de G. Um* ciclo hamiltoniano *em um (di)grafo G é um ciclo simples que contém cada vértice de G.*

Note que em um ciclo hamiltoniano cada vértice do (di)grafo é visitado um única vez. Em um grafo (não dirigido) um caminho $\langle v_0, v_1, \ldots, v_k \rangle$ forma um ciclo se $k \geq 3$ e $v_0 = v_k$.

A definição de *conectividade* exige mais cuidado porque difere entre grafos e digrafos:

- Um grafo é dito *conexo* se para cada par de vértices v e w, existe um caminho entre v e w, ou seja, se qualquer vértice é alcançável a partir de todos os outros. As *componentes conexas* de um grafo são as classes de equivalência dos vértices sob a relação "é alcançável a partir de". Assim, um grafo é conexo se possui apenas uma componente conexa.

- A conectividade em digrafos é dividida em dois casos:

  - Um digrafo é *fortemente conexo* se o vértice u é alcançável a partir do vértice v, e vice-versa, quaisquer que sejam $u, v \in V$. As componentes fortemente conexas de um digrafo são as classes de equivalência dos vértices sob a relação "são mutuamente alcançáceis". Um digrafo é fortemente conexo se possui apenas uma componente fortemente conexa.
  - Um digrafo é *fracamente conexo* se o grafo associado é conexo, mas não é fortemente conexo.

Dois grafos $G = (V, E)$ e $G' = (V', E')$ são *isomorfos* se existir uma bijeção $f : V \to V'$ tal que $(u, v) \in E$ se, e somente se $(f(u), f(v)) \in E'$. Isto significa que podemos renomear os vértices de G como sendo os de G' mantendo as arestas correspondentes em G e G'. Dizemos que $G' = (V', E')$ é um *subgrafo* de $G = (V, E)$ se $V' \subseteq V$ e $E' \subseteq E$. Dado $V' \subseteq V$, o subgrafo de G *induzido* por V' é o grafo $G' = (V', E')$ onde $E' = \{(u, v) \in E : u, v \in V'\}$.

Ao considerarmos o tempo de execução de um algoritmo sobre um (di)grafo $G = (V, E)$, normalmente consideramos tanto o número de vértices $|V|$, como o número de arestas $|E|$ como parâmetros para considerar o tamanho da entrada. É usual o abuso de notação $O(VE)$ quando se quer dizer $O(|V|.|E|)$, uma vez que a leitura se torna mais fácil e não há risco de ambiguidade. Utilizaremos a notação de atributos de forma que, por exemplo, $G.V$ vai denotar o conjunto dos vértices do grafo G.

1.1 Representação de Grafos

Existem duas formas bastante comuns para representar um (di)grafo $G = (V, E)$: matriz de adjacências ou listas de adjacências. As duas representações se aplicam a grafos e a digrafos.

Definição 1.7. *A representação de um grafo $G = (V, E)$ por* listas de adjacências *consiste de um vetor Adj de $|V|$ listas, uma para cada vértice em V. Para cada $u \in V$, a lista de adjacências $Adj[u]$ contém todos os vértices v tais que $(u, v) \in E$, ou seja, contém todos os vértices adjacentes a u em G. Escreveremos $G.Adj[u]$ para se referir a lista $Adj[u]$ de G.*

Se G é um digrafo então a soma dos comprimentos de todas as listas de adjacências é igual a $|E|$, já que uma aresta (u, v) é representada por uma ocorrência de v em $Adj[u]$. Se G for um grafo (não-dirigido) então a soma dos comprimentos de todas as listas de adjacências é igual a $2|E|$ pois uma aresta (u, v) é representada pela ocorrência de v em $Adj[u]$, e pela ocorrência de u em $Adj[v]$. Em ambos os casos, a representação por listas de adjacências utiliza espaço da ordem de $\Theta(V + E)$.

Uma desvantagem das listas de adjacências é que elas não fornecem uma forma rápida de determinar se a aresta (u, v) está ou não no (di)grafo. Para isto precisamos procurar por v em $Adj[u]$. A representação por matrizes de adjacências contorna este problema a um custo assintoticamente maior de espaço.

Definição 1.8. *A representação de um grafo $G = (V, E)$ por* matrizes de adjacências *assume uma enumeração (qualquer) $1, 2, \ldots, |V|$ dos vértices de G, e consiste de uma matriz $A = (a_{ij})$ de dimensão $|V| \times |V|$ tal que*

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

A representação por matrizes de adjacências requer espaço da ordem de $\Theta(V^2)$, independentemente do número de arestas do grafo.

1.2 Busca em Largura

Busca em largura (BFS) é um dos algoritmos mais simples para busca em grafos, além de ser utilizado em outros algoritmos sobre grafos. O algoritmo funciona tanto para grafos quanto para digrafos. Dado um grafo $G = (V, E)$ e um vértice $s \in V$ que chamaremos de *origem*, o algoritmo de busca em largura sistematicamente explora as arestas de G para descobrir todos os vértices que são alcançáceis a partir de s. O nome *busca em largura* se dá porque o algoritmo separa a fronteira entre os vértices que já foram descobertos dos que ainda não o foram a partir da sua distância até a origem s. Assim, o algoritmo descobre todos os vértices que estão a uma distância k da origem antes de descobrir qualquer vértice que esteja a uma distância $k + 1$ da origem. Este algoritmo nos permite responder dois tipos de questões:

1. Existe um caminho do vértice s para o vértice u?
2. Qual é o menor caminho do vértice s para o vértice u?

A seguir apresentamos o pseudocódigo do algoritmo BFS, que recebe como argumento o grafo G, e o vértice s a partir do qual a busca é iniciada.

Algorithm 1: BFS(G, s)

```
1  for each vertex u ∈ G.V − {s} do
2  |  u.color = WHITE;
3  end
4  s.color = GRAY;
5  Q = ∅;
6  enqueue(Q, s);
7  while Q ≠ ∅ do
8  |  u = dequeue(Q);
9  |  for each v ∈ G.Adj[u] do
10 |  |  if v.color == WHITE then
11 |  |  |  v.color = GRAY;
12 |  |  |  enqueue(Q, v);
13 |  |  end
14 |  end
15 end
```

A inicialização (linhas 1-3) percorre todos os vértices, exceto s, e portanto tem tempo de execução limitado pelo número de vértices do grafo, i.e. $\Theta(V)$. As operações das linhas 4, 5 e 6 são executadas em tempo constante. O *loop* das linhas 7-15 precisa de uma análise mais cuidadosa. Os vértices marcados com WHITE durante a inicialização correspondem aos vértices que ainda não foram visitados e, uma

vez que um vértice é visitado, ele é imediatamente marcado com GRAY (linha 11) e inserido na fila Q (linha 12). Durante a primeira execução do *loop*, s é retirado da fila, e cada um dos vértices da lista de adjacências de s é marcado como visitado (GRAY), e então colocado na fila Q. Assim, o percorrimento do grafo inicia pelo vértice s, e em seguida são percorridos todos os vértices adjacentes a s perfazendo um total de $1 + |Adj[s]|$ vértices visitados nesta etapa. Na segunda execução do laço, um vértice adjacente a s, digamos u, é retirado da fila Q e todos os vértices adjacentes a u, que ainda não tenham sido visitados, serão marcados como visitados, e inseridos na fila, de forma que, no máximo, $|Adj[u]|$ vértices serão visitados porque alguns dos elementos em $Adj[u]$ já podem ter sido visitados: de fato, se um vértice v for simultaneamente adjacente aos vértices s e u, que por sua vez é também adjacente a s, então v será inserido na fila na primeira execução do laço, i.e. durante o percorrimento de $Adj[s]$, e será ignorado durante o percorrimento de $Adj[u]$. Portanto $1 + |Adj[s]| + |Adj[u]|$ é uma cota superior para o número de vértices visitados até este momento. Note que um vértice só é inserido na fila uma única vez. Mais ainda, para que um vértice seja inserido na fila Q é necessário que ele seja alcançável a partir de s, e portanto, o processo de enfileiramento e desenfileiramento tem custo limitado por $O(V)$. Cada um dos vértices enfileirados terá sua lista de adjacências percorrida, mas apenas alguns de seus elementos serão visitados. Assim, se $\{s, v_1, v_2, \ldots, v_k\}$ é o conjunto de todos os vértices alcançáveis a partir de s no grafo, então todos eles serão inseridos na fila Q logo após serem visitados, o que tem custo limitado por $O(V)$. Em seguida, para cada vértice $u \in \{s, v_1, v_2, \ldots, v_k\}$, os vértices de $Adj[u]$ que ainda não foram visitados são marcados com GRAY, o que tem custo $O(Adj[u])$. Somando este custo para cada um dos vértices do conjunto $\{s, v_1, v_2, \ldots, v_k\}$, temos custo limitado por $\sum_{u \in \{s, v_1, v_2, \ldots, v_k\}} |Adj[u]| \leq \sum_{u \in V} |Adj[u]| = \Theta(E)$. Assim, o custo total para percorrer o grafo é limitado por $O(V + E)$.

Mais sucintamente: depois da inicialização de BFS (linhas 1-5) nenhum vértice volta a ser marcado com WHITE. Então o teste da linha 13 garante que cada vértice é enfileirado apenas uma vez, e portanto desenfileirado apenas uma vez também. As operações de enfileiramento e desenfileiramento tomam tempo constante $\Theta(1)$, e portanto o tempo requerido pela operação de enfileiramento é da ordem de $O(V)$. Como BFS percorre a lista de adjacências somente quando o vértice é desenfileirado, concluímos que cada lista de adjacências é percorrida apenas uma vez. Como a soma dos comprimentos das listas de adjacências é $\Theta(E)$, o tempo total utilizado no percorrimento das listas de adjacências é limitado por $O(E)$. Portanto BFS possui tempo de execução limitado por $O(V + E)$, isto é, é linear no tamanho da representação de G. Observe que se $|E| \geq |V|$ então $|V| + |E| \leq |E| + |E| = 2.|E|$, e portanto neste caso, $O(V + E)$ significa $O(E)$. Analogamente, se $|E| < |V|$ então $O(V + E)$ significa $O(V)$. Em geral, $O(x + y)$ significa $O(max(x, y))$.

1.2.1 Caminhos de comprimento mínimo

O algoritmo BFS também computa a menor distância (menor número de arestas) de s até cada um dos vértices que são alcançáveis a partir de s. Por fim, o algoritmo constrói uma árvore com raiz s que

contém todos os vértices alcançáveis a partir de s.

Algorithm 2: BFS(G, s)

```
1  for each vertex u ∈ G.V − {s} do
2  |  u.color = WHITE;
3  |  u.d = ∞;
4  |  u.π = NIL;
5  end
6  s.color = GRAY;
7  s.d = 0;
8  s.π = NIL;
9  Q = ∅;
10 enqueue(Q, s);
11 while Q ≠ ∅ do
12 |  u = dequeue(Q);
13 |  for each v ∈ G.Adj[u] do
14 |  |  if v.color == WHITE then
15 |  |  |  v.color = GRAY;
16 |  |  |  v.d = u.d + 1;
17 |  |  |  v.π = u;
18 |  |  |  enqueue(Q, v);
19 |  |  end
20 |  end
21 |  u.color = BLACK;
22 end
```

Denote por $\delta(s, v)$ o número de arestas do *caminho de comprimento mínimo* de s para v, isto é, o menor número de arestas de qualquer caminho de s para v, e $\delta(s, v) = \infty$ se não existe caminho de s para v.

Lema 1.9. *Seja $G = (V, E)$ um (di)grafo, e $s \in V$. Então para qualquer aresta $(u, v) \in E$, temos $\delta(s, v) \leq \delta(s, u) + 1$.*

Lema 1.10. *Seja $G = (V, E)$ um (di)grafo, e $s \in V$. Então após a execução de BFS(G, s), para cada vértice $v \in V$, temos que $v.d \geq \delta(s, v)$.*

Lema 1.11. *Suponha que durante a execução de BFS no grafo $G = (V, E)$, a fila Q contenha os vértices $v_1, v_2, \ldots, v_r$, i.e. $Q = \langle v_1, v_2, \ldots, v_r \rangle$, onde v_1 é o primeiro elemento, e v_r, o último. Então $v_r.d \leq v_1.d + 1$ e $v_i.d \leq v_{i+1}.d$ para todo $i = 1, 2, \ldots, r - 1$.*

Corolário 1.12. *Suponha que o vértice v_i tenha sido enfileirado antes do vértice v_j durante a execução do algoritmo BFS. Então $v_i.d \leq v_j.d$ no momento em que v_j é enfileirado.*

Por fim, o teorema a seguir estabelece a correção do algoritmo BFS:

Teorema 1.13. *Seja $G = (V, E)$ um (di)grafo, e considere a execução de BFS em G a partir da origem $s \in V$. Então, durante sua execução, BFS descobre cada vértice $v \in V$ que é alcançável a partir de s, e após o término de sua execução, $v.d = \delta(s, v), \forall v \in V$. Adicionalmente, para cada vértice $v \neq s$ alcançável a partir de s, um dos menores caminhos de s para v é o menor caminho de s para $v.\pi$ seguido da aresta $(v.\pi, v)$.*

Exercício 1.14. *Modifique o algoritmo BFS para que ele receba o grafo G na forma de matriz de adjacências. Qual é a complexidade de tempo de BFS neste caso?*