Projeto e Análise de Algoritmos

Flávio L. C. de Moura

20 de junho de 2023

Nesta aula estudaremos outro algoritmo de busca em grafos, o algoritmo de busca em profundidade (*Depth-first search* - DFS). Neste caso, diferentemente do algoritmo BFS visto anteriormente, a busca vai o mais profundo possível no grafo visitando um vértice adjacente ao vértice que acaba de ser visitado, e em seguida visita outro vértice adjacente ao vértice adjacente visitado até que não seja mais possível, quando a busca retorna (*backtrack*) para o vértice a partir do qual o vértice que não tem mais adjacentes a serem visitados foi descoberto. Este processo de busca continua até que todos os vértices alcançáveis a partir do vértice inicial (fonte) forem visitados. Caso ainda existam vértices não visitados, DFS seleciona algum destes vértices como fonte, e repete esta estratégia de busca. O algoritmo para depois que todos os vértices tenha sido visitados. Vejamos o pseudocódigo de DFS[1]:

```
1  for each vertex u ∈ G.V do
2  |  u.color = WHITE;
3  |  u.π = NIL;
4  end
5  time = 0;
6  for each vertex u ∈ G.V do
7  |  if u.color == WHITE then
8  |  |  DFS-Visit(G, u)
9  |  end
10 end
```

Algoritmo 1: DFS(G)

```
1  time = time + 1;
2  u.d = time;
3  u.color = GRAY;
4  for each v ∈ G.Adj[u] do
5  |  if v.color == WHITE then
6  |  |  v.π = u;
7  |  |  DFS-Visit(G, v)
8  |  end
9  end
10 u.color = BLACK;
11 time = time + 1;
12 u.f = time;
```

Algoritmo 2: DFS-Visit(G, u)

Qual o tempo de execução de DFS? O laço das linhas 1-4 é executado em tempo $\Theta(V)$. Já o laço das linhas 6-10 exige um pouco mais de atenção porque este laço faz uma chamada ao algoritmo DFS-visit. Então vamos determinar o custo de DFS-Visit primeiro. Em cada execução de DFS-Visit(G, u), o *loop* das linhas 4-9 é executado $|Adj[u]|$ vezes. Como

$$\sum_{v \in V} |Adj[v]| = \Theta(E)$$

o custo total de DFS-Visit é $\Theta(E)$, e portanto, o custo total de DFS é $\Theta(V + E)$.

Definição 1. *O* subgrafo predecessor *da busca em profundidade é definido por:*

- $G_\pi = (V, E_\pi)$*, onde* $E_\pi = \{(v.\pi, v) : v \in V \text{ and } v.\pi \neq NIL\}$

O subgrafo predecessor de uma busca em profundidade forma uma floresta.

Teorema 2. *Para dois vértices u e v quaisquer de um (di)grafo G, apenas uma das seguintes propriedades ocorre em uma busca em profundidade (DFS) em G, considerando que $u.d < v.d$:*

1. $u.d < v.d < v.f < u.f$*, e v é um descendente de u no subgrafo predecessor de G;*
2. $u.d < u.f < v.d < v.f$*, e u não é um descendente de v no subgrafo predecessor de G, ou vice-versa.*

Teorema 3. *Seja $G = (V, E)$ um grafo, e considere a floresta F obtida após a execução de DFS(G). As componentes de F são precisamente as componentes conexas de G.*

Corolário 4. *As componentes conexas de um grafo $G = (V, E)$ podem ser encontradas em tempo $\Theta(V + E)$.*

1 Aplicação: Componentes fortemente conexas

Nesta seção veremos como determinar as componentes fortemente conexas de um digrafo. É fácil ver que a estratégia que utilizamos para determinar as componentes conexas de um grafo não funciona para digrafos. De fato, ao executarmos DFS a partir de um vértice u, percorremos todos os vértices alcançáveis a partir de u, e terminamos com uma árvore que tem u como raiz e cujos vértices não estão fortemente conectados. Uma componente fortemente conexa de um digrafo $G = (V, E)$ é um conjunto maximal de vértices $C \subseteq V$ tal que para qualquer par de vértices u e v em C, temos que $u \rightsquigarrow v$ e $v \rightsquigarrow u$, ou seja, u e v são mutuamente alcançáveis. O algoritmo que apresentaremos a seguir utiliza o digrafo G^T, chamado de *transposto de G*, definido por $G^T = (V, E^T)$, onde $E^T = \{(u, v) : (v, u) \in E\}$.

Exercício 5. *Construa um algoritmo eficiente para computar o digrafo transposto G^T de G a partir da lista de adjacências de G, e em seguida faça a análise assintótica do algoritmo.*

Exercício 6. *Construa um algoritmo eficiente para computar o digrafo transposto G^T de G a partir da matriz de adjacências de G, e em seguida faça a análise assintótica do algoritmo.*

A ideia do algoritmo que veremos a seguir está baseada no fato de que, para qualquer digrafo G, tanto G quanto G^T possuem as mesmas componentes fortemente conexas. De fato, u e v são mutualmente alcançáveis em G, se e somente se, u e v são mutualmente alcançáveis em G^T.

1 Execute DFS(G) para computar o tempo de finalização $u.f$ para cada vértice u;
2 Compute G^T;
3 Execute DFS(G^T) escolhendo os vértices em ordem decrescente do tempo de finalização;
4 Retorne os vértices de cada árvore da floresta computada na linha anterior como uma componente fortemente conexa separada.

Algoritmo 3: SCC(G)

Cada conjunto de vértices retornado pelo algoritmo SCC constitui um vértice do grafo $G^{scc} = (V^{scc}, E^{scc})$, chamado de *grafo das componentes fortes* de G, cujos vértices correspondem às componentes fortemente conexas de G, digamos $C_1, C_2, \ldots, C_k$ e, $(C_i, C_j) \in E^{scc}$ se $i \neq j$ e existe alguma aresta de G que sai de um vértice de C_i e chega em um vértice de C_j. O grafo das componentes conexas tem como propriedade fundamental não possuir ciclos. Um digrafo sem ciclo é normalmente chamado de DAG (*directed acyclic graph*).

A seguir, estenderemos a noção de tempo de início e finalização de um vértice para conjuntos de vértices. Esta extensão será necessária para provarmos a correção do algoritmo SCC.

Definição 7. *Se $U \subseteq V$ então definimos $d(U) = \min_{u \in U}\{u.d\}$ e $f(U) = \max_{u \in U}\{u.f\}$*

Ou seja, $d(U)$ representa o tempo do primeiro vértice que foi visitado do conjunto U, enquanto que $f(U)$ denota o tempo do último vértice de U a ter a visita finalizada.

Lema 8. *Sejam C e C' duas componentes fortemente conexas de um digrafo $G = (V, E)$. Suponha que exista uma aresta $(u, v) \in E$ tal que $u \in C$ e $v \in C'$. Então $f(C) > f(C')$.*

Corolário 9. *Sejam C e C' duas componentes fortemente conexas de um digrafo $G = (V, E)$, e suponha que $f(C) > f(C')$. Então E^T não possui aresta (v, u) tal que $u \in C'$ e $v \in C$.*

Teorema 10. *O algoritmo SCC computa corretamente as componentes fortemente conexas de um digrafo G dado como argumento.*

Referências

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.