Aula 6
Grafos I (Introdução e DFS)

Gustavo de Medeiros Carlos
MaratonUSP
Instituto de Matemática e Estatística
Universidade de São Paulo

1 de maio de 2020

Introdução

Definição

Um grafo é formado por um conjunto de vértices e um conjunto[1] de arestas. Uma aresta é par de vértices.

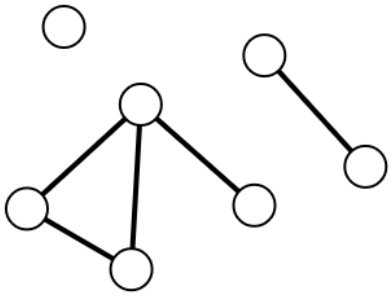

Figura 1: Exemplo simples de um grafo.

Representação

Chamamos de V o conjunto de vértices e de E o conjunto de arestas de um grafo. Nesse caso, iremos indexar os vértices do grafo acima de 0 a

[1] multiconjunto no caso de um multigrafo

$|V| - 1$, sendo $|V|$ o número total de vértices. Para o grafo indexado vemos que

$$V = \{0, 1, 2, 3, 4, 5, 6\},$$
$$E = \{(0, 1), (0, 4), (1, 3), (1, 4), (2, 6)\}.$$

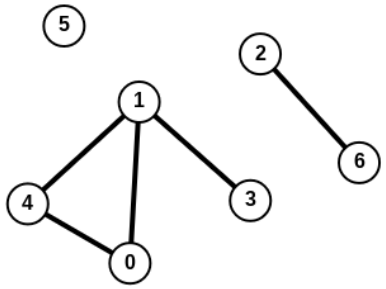

Figura 2: Mesmo grafo da figura 1 com seus vértices indexados.

Exemplos reais

Grafos estão fortemente presentes em nosso dia-a-dia, como por exemplo:

- redes sociais;
- ruas de uma cidade;
- aeroportos;
- internet.

No caso dos aeroportos, podemos considerar que os vértices são os aeroportos e as arestas representam voos entre os aeroportos.

Terminologia

Multigrafo

Um **multigrafo** é aquele que permite a repetição de arestas. Na definição desse tipo de grafo, consideramos um multiconjunto de arestas, como abaixo

$$E = \{(0,1),(0,4),(0,4),(1,3),(1,4),(2,6)\}.$$

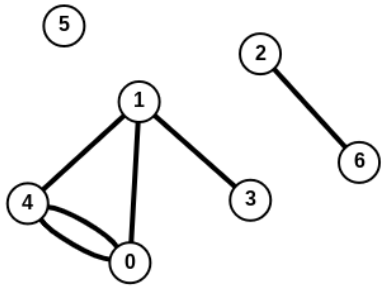

Figura 3: Exemplo de multigrafo.

Perceba que um multigrafo pode ocorrer no exemplo do aeroporto, quando houver múltiplos voos entre dois aeroportos.

Grafo direcionado

Um grafo **direcionado** é aquele em que as arestas conectam um vértice ao outro somente em uma direção. Desse modo, uma aresta representada por (a, b), conecta o vértice a ao vértice b mas não o contrário. As arestas são, então, um par ordenado. No caso de um grafo **não-direcionado** a ordem dos vértices em uma aresta não é relevante. No grafo abaixo, temos o seguinte conjunto de arestas

$$E = \{(0,1),(0,4),(3,1),(4,0),(4,1),(6,2)\}.$$

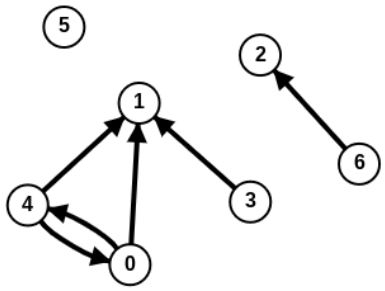

Figura 4: Exemplo de grafo direcionado.

As redes sociais são bons exemplos para destacarmos a diferença entre esses dois tipos de grafos. Considerando que os vértices são pessoas, em uma rede social como o Facebook, na qual as arestas indicam relações de amizade, o grafo é não-direcionado. Mas no caso do Twitter, em que as arestas (a, b) indicam que a segue b, o grafo é direcionado.

Grafo ponderado

Um **grafo ponderado** possui um peso associado a cada uma de suas aresta, como abaixo

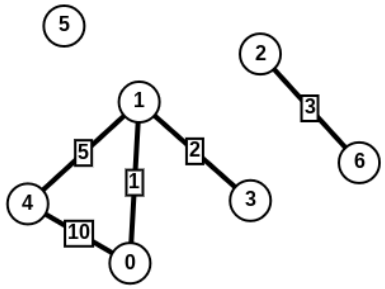

Figura 5: Exemplo de grafo com pesos.

Para o caso de um grafo que representa as ruas de uma cidade, os vértices são as intersecções ou extremos de ruas, as arestas são as ruas em si e o peso de cada aresta é o comprimento da rua que ela representa.

Loop

Chamamos de **loop** ou **self-loop** a aresta que começa e termina em um mesmo vértice.

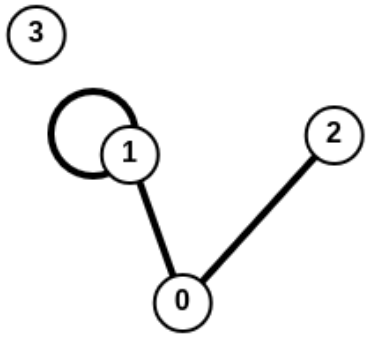

Figura 6: Exemplo de um self-loop $(1, 1)$.

Grau de um vértice

O **grau** de um vértice é o número de arestas incidentes nele. Em um grafo direcionado temos dois tipos de grau, o grau de entrada, número de arestas que chegam no vértice e o grau de saída, número de arestas que saem do vértice. No 2, o vértice 0 tem grau 2, o vértice 5 tem grau 0.

Caminho

Um **caminho** em um grafo é uma sequência de vértices no qual cada dois vértices consecutivos possuem uma aresta com extremos neles. Um exemplo de caminho na figura 2 é a sequência 3, 1, 4, 0, 1.

Um **caminho simples** em um grafo é um caminho que não possui vértices repetidos, geralmente quando se fala de caminho, refere-se a caminhos simples. Um exemplo de caminho simples na figura 2 é a sequência 3, 1, 4, 0.

Ciclo

Chamamos um caminho de **circuito** se o seu começo e fim são o mesmo vértice. Um **ciclo** é um caminho que somente o primeiro e último vértices são iguais, os outros devem ser distintos, além disso, não pode passar pela mesma aresta duas vezes. A figura 2 possui o ciclo 1 - 4 - 0 - 1.

Grafo conexo

Definimos como **grafo conexo** aquele que possui pelo menos um caminho conectando quaisquer dois vértices, caso contrário o grafo é **desconexo**.

Todas as figuras anteriores são exemplos de grafos desconexos, abaixo está um exemplo de um grafo conexo.

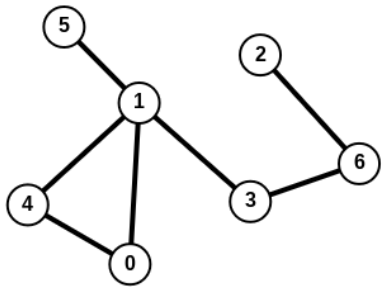

Figura 7: Exemplo de um grafo conexo.

Grafo completo

Dizemos que um grafo é **completo** se, para quaisquer dois vértices desse grafo, temos uma aresta conectando diretamente eles. São no total $|V| \cdot (|V| - 1)$ arestas caso seja um grafo direcionado e $\frac{|V| \cdot (|V|-1)}{2}$ se for não-direcionado.

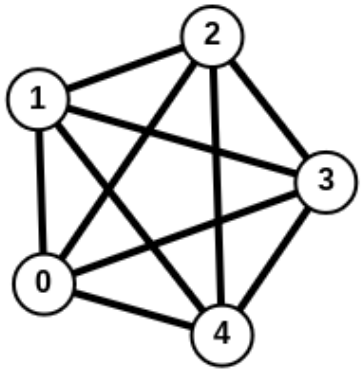

Figura 8: Exemplo de um grafo completo.

Subgrafo

Aqui chamaremos de V_I e E_I o conjunto de vértice e arestas do grafo I, respectivamente. Dizemos que o grafo A é um **subgrafo** do grafo B se $V_B \subset V_A$ e $E_B \subset E_A$ de modo que cada aresta em E_B possui como extremos vértices contidos em V_B.

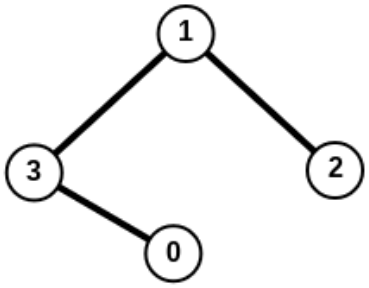

Figura 9: Subgrafo da figura 2.

Note que alguns índices mudaram, podemos fazer isso visto que eles são somente um modo de nos referirmos a cada vértice.

Componente conexa

Uma **componente conexa** de um grafo é um subgrafo conexo no qual não existe nenhum caminho entre um vértice do subgrafo e um vértice fora dele. O grafo da figura 2 possui 3 componentes conexas,

- $V_1 = \{5\}$
  $E_1 = \{\}$,
- $V_2 = \{2, 6\}$
  $E_2 = \{(2, 6)\}$,
- $V_3 = \{0, 1, 3, 4\}$
  $E_3 = \{(0, 1), (0, 4), (1, 3), (1, 4)\}$.

Pontes em um grafo

Aresta Arestas ponte são aquelas que o número de componentes conexas do grafo aumenta se removidas. Como exemplo, as arestas $(1, 3)$ e $(2, 6)$ no grafo da figura 2.

Vértice Vértices ponte são aqueles que o número de componentes conexas do grafo aumenta se removidos (junto com as arestas incidentes). Como exemplo, o vértice 1 no grafo da figura 2.

Grafo bipartido

Um grafo é **bipartido** se conseguimos separar os vértices em dois conjuntos de modo que, para toda aresta, os seus extremos estão em conjuntos distintos. É análogo a pintar os vértices com duas cores de modo que as arestas conectam vértices de cores diferentes.

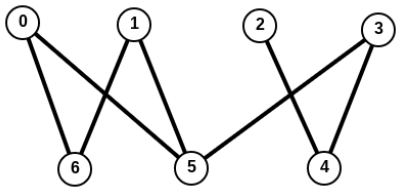

Figura 10: Exemplo de grafo bipartido.

Na figura podemos colocar os vértices 0, 1, 2 e 3 em um conjunto e os vértices 4, 5 e 6 no outro. Todas as arestas levam de um conjunto ao outro.

Árvore

Definição

Um grafo é uma árvore se é conexo e não possui ciclo.

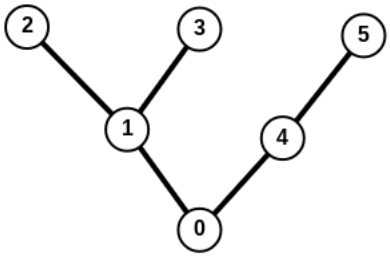

Figura 11: Exemplo de uma árvore.

Propriedades

Se uma árvore possui $|V|$ vértices, ela terá $|V| - 1$ arestas.

Para quaisquer dois vértices em uma árvore, existe somente um caminho simples que conecta os dois. Se tirarmos qualquer aresta da árvore, ela vai ficar desconexa.

Os vértices de grau 1 da árvore são chamados de **folhas**. Podemos definir um dos vértices da árvore como a **raiz**. Em um caminho simples partindo

da raiz, ao irmos do vértice v para o vértice u (consecutivos), consideramos que v é o pai de u e que u é filho de v.

Na árvore da figura 11, chamamos o vértice 0 de raiz, 1 e 4 são seus filhos. Os vértices 2, 3 e 5 são folhas e o pai de 5 é 4.

Floresta

Um grafo é chamado de **floresta** se todas as suas componentes conexas forem árvores.

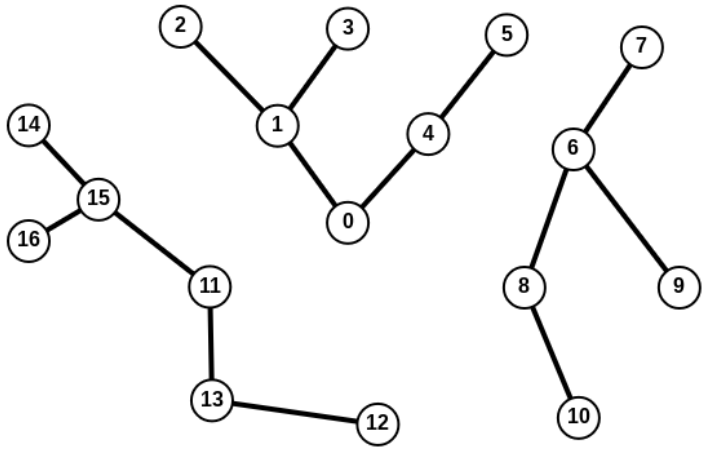

Figura 12: Exemplo de uma floresta.

Representação no computador

Vamos representar o grafo a seguir de 3 maneiras diferentes.

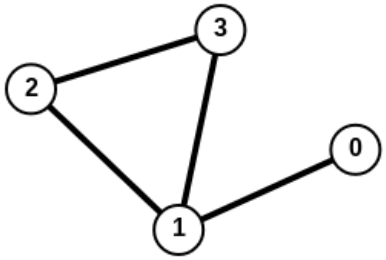

Figura 13: Grafo simples.

Lista de arestas

Podemos representar um grafo com uma lista de arestas,

$$E = \{(0, 1), (1, 2), (1, 3), (2, 3)\}.$$

Em C++

```cpp
int main() {
    vector<pair<int, int>> arestas;
    arestas.push_back({0, 1});
    arestas.push_back({1, 2});
    arestas.push_back({1, 3});
    arestas.push_back({2, 3});
    return 0;
}
```

Matriz de adjacência

Um grafo pode ser representado em uma matriz $A = |V| \times |V|$, em que o elemento a_{ij} recebe o peso da aresta (ou 1 se não houver pesos) que leva do vértice i ao vértice j. Se não há aresta entre esses dois vértices, a_{ij} recebe 0.

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Note que a matriz é simétrica por se tratar de um grafo não-direcionado.

Em C++

```cpp
#define V 100 // número de vértices

/* int adj[V][V]; se a declarássemos aqui,
a matriz já seria inicializada com zeros */

int main() {
    int adj[V][V];
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
```

```
            adj[i][j] = 0;
        }
    }

    adj[0][1] = adj[1][0] = 1;
    adj[1][2] = adj[2][1] = 1;
    adj[1][3] = adj[3][1] = 1;
    adj[2][3] = adj[3][2] = 1;
    return 0;
}
```

Lista de adjacências

Também podemos representar um grafo com vetor em que cada elemento é uma lista de vértices adjacentes ao vértice correspondente ao índice no vetor.

$$
\begin{aligned}
&0\text{: } 1\\
&1\text{: } 0, 2, 3\\
&2\text{: } 1, 3\\
&3\text{: } 1, 2
\end{aligned}
$$

```
#define V 100 // número de vértices

int main() {
    vector<int> adj[V];
    adj[0].push_back(1);
    adj[1].push_back(0);
    adj[1].push_back(2);
    adj[1].push_back(3);
    adj[2].push_back(1);
    adj[2].push_back(3);
    adj[3].push_back(1);
    adj[3].push_back(2);
    return 0;
}
```

Comparação entre representações

Abaixo está a complexidade de diferentes operações em cada uma das estruturas apresentadas. $\max(E_i)$ é maior grau de um vértice no grafo.

	Lista de arestas	Matriz de adjacências	Lista de adjacências
Espaço	$O(\|E\|)$	$O(\|V\|^2)$	$O(\|E\|)$
Procurar aresta	$O(\|E\|)$	$O(1)$	$O(\max(E_i))$
Listar vizinhos	$O(\|E\|)$	$O(\|V\|)$	$O(\max(E_i))$
Adicionar aresta	$O(1)$	$O(1)$	$O(1)$
Remover aresta	$O(\|E\|)$	$O(1)$	$O(\max(E_i))$

Busca em profundidade (DFS)

Definição

A busca em profundidade ou *depth-first search (DFS)* é um algoritmo recursivo que começando em um vértice inicial, segue o caminho mais longo possível em cada um de seus vértices adjacentes que ainda não foram visitados. Após visitar todos os vértices adjacentes possíveis, faz *back-tracking* para o vértice anterior e continua dele. O conjunto de vértices e arestas que uma DFS percorre é uma árvore.

Exemplo

Aplicamos a DFS partindo do vértice 0 no grafo da figura 2. Vemos que a ordem de travessia é 0 - 1 - 3 - 4.

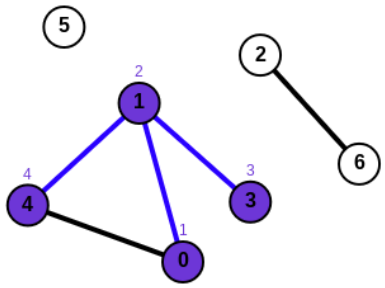

Figura 14: DFS em grafo.

Algoritmo

Podemos implementar a DFS como uma função recursiva, dado uma representação do grafo em uma lista de adjacências.

Pseudo código

```
dfs(v) {
    marca o vértice 'v' como visitado
    para todo vértice 'av' não visitado na lista de adjacências de 'v' {
        dfs(av)
    }
}
```

Implementação em C++

Nesse exemplo iremos imprimir os vértices visitados na dfs e salvar o pai de cada um em um vetor, a raiz é o vértice inicial e não tem pai. A entrada consiste no número de v de vértices e a de arestas na primeira linha e os extremos de cada aresta nas próximas v linhas.

```cpp
#include <bits/stdc++.h>
using namespace std;

#define V 100 // número máximo de vértices

int pai[V];
bool visitado[V]; // vetor que indica os vértices visitados
vector<int> adj[V]; // vetor de adjacências

void dfsUtil(int v) {
    visitado[v] = true; // marca como visitado
    // for (int av : adj[v])  também é possivel usar essa sintaxe
    for (int a = 0; a < adj[v].size(); a++) {
        int av = adj[v][a];
        if (!visitado[av]) {
            cout << av << " "; // imprime o vértice adjacente
            dfsUtil(av); // chama dfs para o vértice adjacente não visitado
            pai[av] = v; // salva o pai do vértice av
        }
    }
}

void dfs(int r, int v) {
    for (int i = 0; i < v; i++) {
        visitado[i] = false;
        pai[i] = -1;
```

```
    }
    cout << r << " ";
    dfsUtil(r);
    cout << "\n";
}

int main() {
    int v, a;
    cin >> v >> a;
    for (int i = 0; i < a; i++) {
        int x, y;
        cin >> x >> y;
        adj[x].push_back(y);
        adj[y].push_back(x);
    }
    dfs(0, v);
    for (int i = 0; i < v; i++) {
        cout << "pai de " << i << " = " << pai[i] << "\n";
    }
    return 0;
}
```

Para a entrada

```
7 5
0 1
1 3
1 4
0 4
2 6
```

obtemos

```
0 1 3 4
pai de 0 = -1
pai de 1 = 0
pai de 2 = -1
pai de 3 = 1
pai de 4 = 1
pai de 5 = -1
pai de 6 = -1
```

Complexidade

A função dfs será chamada no máximo uma vez para cada vértice e quando é chamada para um determinado vértice, o loop repete E_i vezes, sendo E_i o número de vértices adjacentes ao vértice i. Observamos que

$$\sum_{i=0}^{|V|-1} E_i = 2E$$

para um grafo não-direcionado. Temos que a complexidade da DFS é $O(|V| + 2|E|) = O(|V| + |E|)$. Se $|V| \approx |E|$, podemos considerar que é um algoritmo linear no número de vértices.

Aplicações

São algumas aplicações do algoritmo de DFS:

- achar componentes conexas;
- achar as pontes de um grafo;
- gerar uma árvore a partir de um grafo;
- detectar ciclos;
- distância entre vértices quando o grafo for uma árvore.