Grafos

Fundamentos

Prof. Edson Alves

2019

Faculdade UnB Gama

Sumário

Fundamentos

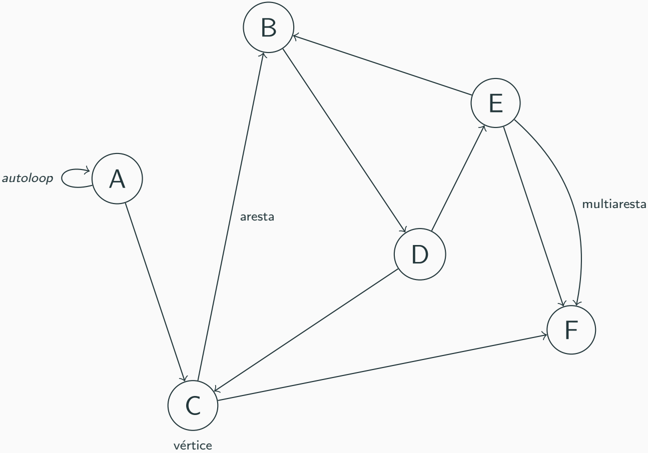
Motivação

- Os grafos são estruturas de dados que permitem a abstração das demais estruturas de dados
- Os grafos expressam relações entre nós e arestas, e a adição de restrições ou condições à estas relações geram novas estruturas de dados
- Muitos problemas reais podem ser modelados por meio de grafos
- Os algoritmos fundamentais de grafos tem tempo de execução proporcional os número de vértices e arestas, sendo aplicáveis a grafos relativamente grandes
- Há uma vasta literatura sobre grafos, e algoritmos clássicos que resolvem problemas que mapeiam aplicações reais

Definição de grafo

- Um grafo $G = (V, E)$ é composto de um conjunto de vértices V e um conjunto de arestas E
- Uma aresta é um par ordenado de vértices $e = (u, v)$, com $u, v \in V$, a qual significa que o vértice u está relacionado ao vértice v
- Os vértices, em geral, representam pontos ou localizações no espaço
- As arestas representam ligações, conexões, rodovias, acessos, etc, entre dois vértices
- A cardinalidade (número de elementos) dos conjuntos V e E é representada pelas notações $|V|, |E|$ ou pelas variáveis N e M
- Multiarestas são repetições do par (u, v) em E
- Um *autoloop* é uma aresta $(u, u) \in E$
- Um vértice v é dito isolado se não existe nenhuma aresta $e \in E$ tal que v seja uma de suas duas coordenadas

Visualização de um grafo

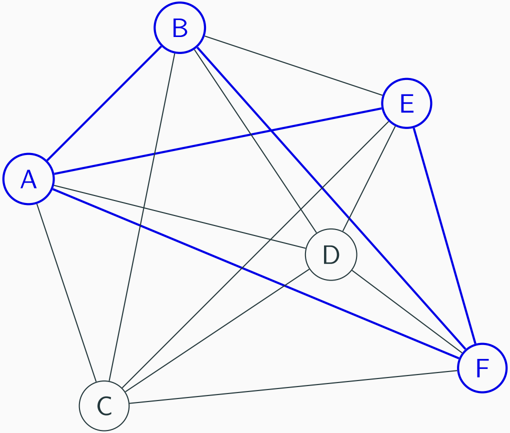


Classificação de grafos

Simples, completo e subgrafos

- Um grafo é dito simples se ele não contém *autoloops* nem multiarestas
- Um grafo que não é simples é denominado multigrafo
- A maior parte dos grafos que modelam aplicações práticas são simples
- Um grafo é dito completo se, para cada par de vértices $u, v \in V, u \neq v$, tem-se que $(u, v) \in E$
- O grafo $S(V', E') \subset G(V, E)$ é denominado subgrafo de G se $V' \subset V, E' \subset E$ e, para qualquer par $(a, b) \in E'$, vale que $a, b \in V'$

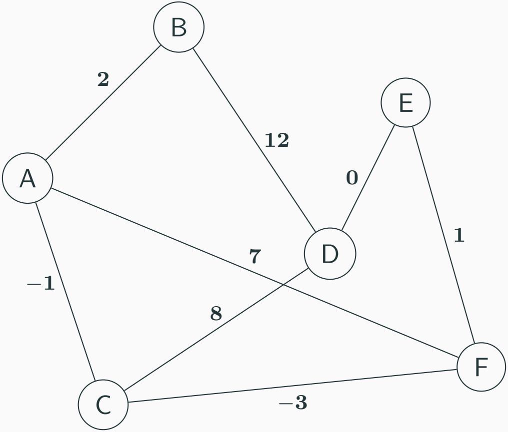
Visualização de um grafo simples completo, com um subgrafo S em azul



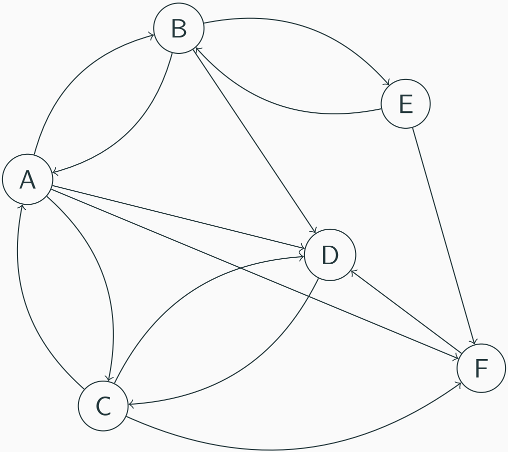
Grafos direcionados, ponderados e densos

- Se, para qualquer aresta $(u, v) \in E$, vale que $(v, u) \in E$, o grafo é dito não direcionado
- Caso contrário, o grafo é direcionado, e a aresta (u, v) significa que u está relacionado a v, mas não necessariamento o contrário
- Se a cada aresta $e \in E$ for associado um valor w, denominado peso, o grafo é denominado grafo ponderado
- Um grafo que não tem pesos associados às arestas é chamado grafo não poderado
- Um grafo é dito esparso se o número de aresta E é "pequeno" (em geral, $O(V)$)
- Um grafo é dito denso se o número de arestas E é "grande" (em geral, $O(V^2)$)

Visualização de um grafo esparso ponderado não direcionado



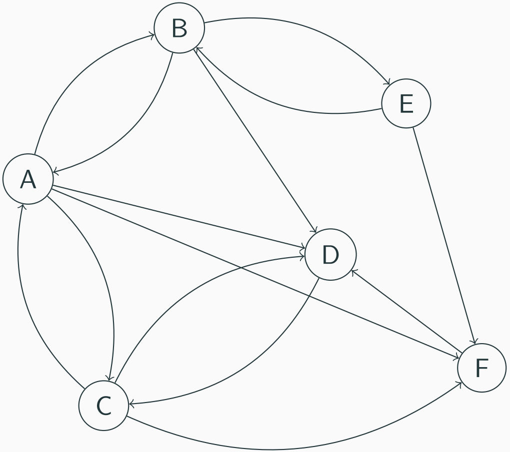
Visualização de um grafo denso, direcionado e não ponderado



Graus de chegada e de saída

- O graup de entrada $g_i(u)$ de um vértice u é igual ao número de arestas $e \in E$ cuja segunda coordenada é igual a u
- Em outras palavras, é igual a o número de arestas que "chegam" em u
- De forma semelhante, o grau de saída $g_o(u)$ de um vértice u é dado pelo número de arestas $e \in E$ tais que a primeira coordenada é igual a u
- Isto é, é o número de arestas que "partem" de u
- Em um grafo não-direcionado, $g_i(u) = g_o(u)$, $\forall u \in V$

Visualização dos graus de chegada e saída



u	$g_i(u)$	$g_o(u)$
A	2	4
B	2	3
C	2	3
D	4	1
E	1	2
F	3	1

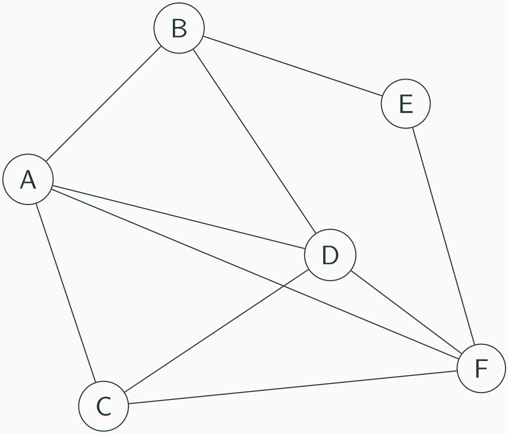
Caminhos

- Um caminho entre dois vértices u e v é uma sequência de arestas que conectam ambos vértices
- Em outros termos, é uma sequência não-nula de arestas

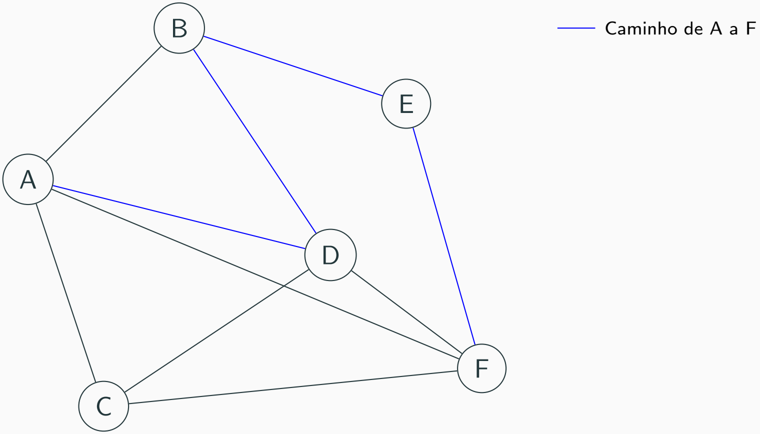
$$(u, w_1), (w_1, w_2), (w_2, w_3), \ldots, (w_{n-1}, w_n), (w_n, v),$$

  onde u é o ponto de partida e v o ponto de chegada
- Obserque, para quaisquer duas arestas consecutivas do caminho $e = (e_1, e_2), f = (f_1, f_2)$, vale que $e_2 = f_1$
- Nem sempre existe um caminho de u até v, e pode existir mais de um caminho de u a v
- Um ciclo é um caminho cujo ponto de partida é igual ao ponto de chegada
- Um vértice é atingível a partir de u se existir um caminho de u até v
- Um grafo é dito conectado se todos os seus vértices são atingíveis a partir de qualquer vértice de V

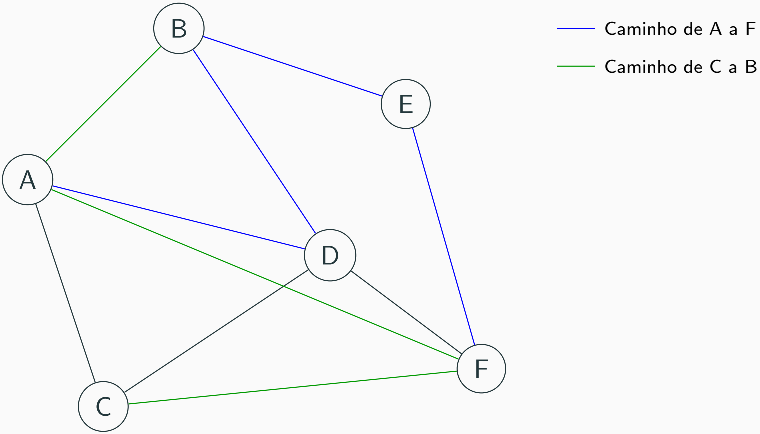
Visualização de um grafo conectado e caminhos



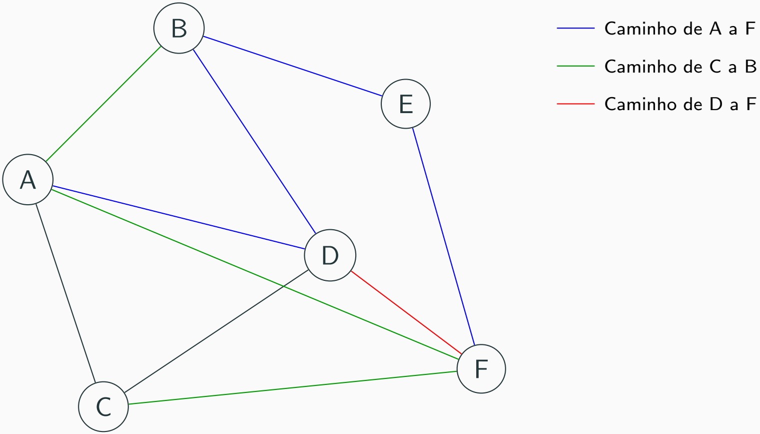
Visualização de um grafo conectado e caminhos



Visualização de um grafo conectado e caminhos



Visualização de um grafo conectado e caminhos

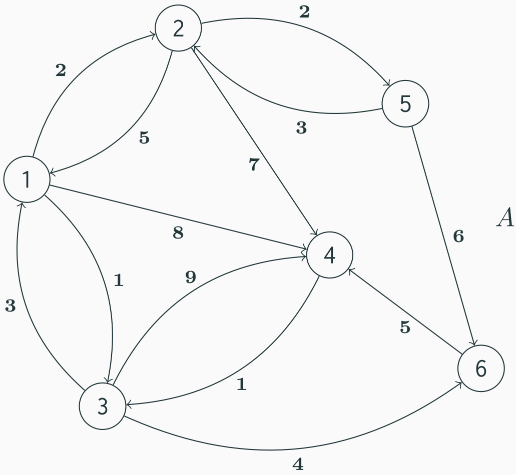


Representação de grafos

Matriz de adjacências

- Seja $N = |V|$ e considere que cada vértice $v \in V$ é representado por um número positivo distinto do intervalo $[1, N]$
- A representação de um grafo por matriz de adjacências utiliza uma matriz $A_{N \times N}$, onde o elemento a_{ij} armazena o peso da aresta que une o vértice i ao vértice j
- Em um grafo não-ponderado, $a_{ij} = 1$ representa a existência da aresta, e $a_{ij} = 0$ a inexistência da mesma
- Outra interpretação possível é considerar a_{ij} o número de ocorrências da aresta (i, j) em um multigrafo não-ponderado
- Em um grafo simples, $a_{ii} = 0, \forall i \in V$
- Esta representação responde perguntas do tipo "Há uma aresta entre u e v? Qual é o seu peso?" em $O(1)$
- Contudo, a complexidade de memória é alta ($O(N^2)$)

Visualização da representação por matriz de adjacências



$$A = \begin{bmatrix} 0 & 2 & 1 & 8 & 0 & 0 \\ 5 & 0 & 0 & 7 & 2 & 0 \\ 3 & 0 & 0 & 9 & 0 & 4 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 6 \\ 0 & 0 & 0 & 5 & 0 & 0 \end{bmatrix}$$

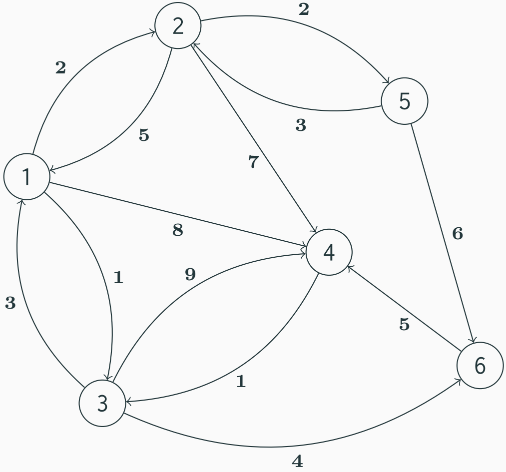
Representação por matriz de adjacências

```cpp
#include <bits/stdc++.h>

const int N { 6 };
int A[N + 1][N + 1];

int main()
{
    A[1][2] = 2, A[1][3] = 1, A[1][4] = 8;
    A[2][1] = 5, A[2][4] = 7, A[2][5] = 2;
    A[3][1] = 3, A[3][4] = 9, A[3][6] = 4;
    A[4][3] = 1;
    A[5][2] = 3, A[5][6] = 6;
    A[6][4] = 5;

    for (int i = 1; i <= N; ++i)
        for (int j = 1; j <= N; ++j)
            std::cout << A[i][j] << (j == N ? '\n' : ' ');

    return 0;
}
```

Representação por lista de adjacências

- Na representação por lista de adjacências, a cada vértice u é associada uma lista que contém os identificadores dos vértices v relacionados a u
- Caso o grafo seja ponderado, cada entrada da lista contém duas informações: o identificador do vértice v e o peso w da aresta (u, v)
- Em C++, estas listas podem ser contêniers do tipo `list`, `vector` ou `forward_list`
- A escolha depende do tipo de travessia que será feita em cada aresta que parte de u
- Esta representação é mais adequada para grafos esparsos do que as matrizes de adjacência, uma vez que tem complexidade de memória $O(N + M)$
- A maioria dos algoritmos clássicos de grafos utiliza esta representação

Visualização da representação por lista de adjacências



1	$(2,2)$	$(3,1)$	$(4,8)$
2	$(1,5)$	$(4,7)$	$(5,2)$
3	$(1,3)$	$(4,9)$	$(6,4)$
4	$(3,1)$		
5	$(2,3)$	$(6,6)$	
6	$(4,5)$		

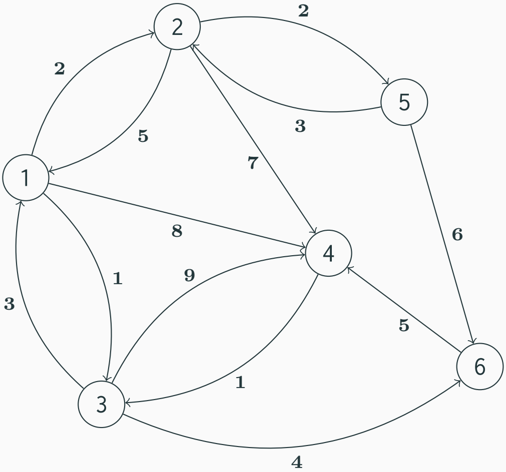
Representação por lista de adjacências

```cpp
#include <bits/stdc++.h>

using namespace std;
using ii = pair<int, int>;

vector<ii> adj[] { {}, { { 2, 2 }, { 3, 1 }, { 4, 8 } },
    { { 1, 5 }, { 4, 7 }, { 5, 2 } }, { { 1, 3 }, { 4, 9 }, { 6, 4 } },
    { { 3, 1 } }, { { 2, 3 }, { 6, 6 } }, { { 4, 5 } }, };

int main()
{
    for (int u = 1; u <= 6; ++u) {
        cout << u << ": ";

        for (size_t v = 0; v < adj[u].size(); ++v)
            cout << "(" << adj[u][v].first << ", " << adj[u][v].second
                << ")" << (v + 1 == adj[u].size() ? '\n' : ' ');
    }

    return 0;
}
```

Representação por lista de arestas

- Na representação por lista de arestas o grafo é representado pelo conjunto E das arestas
- Se o grafo é ponderado, cada aresta corresponderá a uma tripla com os valores (u, v, w), que indica que a aresta (u, v) tem peso w
- Observe que é possível obter o conjunto V a partir de E caso o grafo $G(V, E)$ não possua vértices isolados
- Esta representação tem complexidade $O(M)$, onde $M = |E|$
- Esta representação é utilizada no algoritmo de Kruskall para árvore mínima geradora
- Nos demais algoritmos a implementação pode ficar mais trabalhosa, ou mesmo ter maior complexidade assintótica, uma vez que os vértices V não podem ser acessados diretamente

Visualização da representação por lista de arestas



$(1, 2, 2)$
$(1, 3, 1)$
$(1, 4, 8)$
$(2, 1, 5)$
$(2, 4, 7)$
$(2, 5, 2)$
$(3, 1, 3)$
$(3, 4, 9)$
$(3, 6, 4)$
$(4, 3, 1)$
$(5, 2, 3)$
$(5, 6, 6)$
$(6, 4, 5)$

Representação por lista de arestas

```
#include <bits/stdc++.h>

using namespace std;

struct edge { int u, v, w; };

vector<edge> es { { 1, 2, 2 }, { 1, 3, 1 }, { 1, 4, 8 }, { 2, 1, 5 },
    { 2, 4, 7 }, { 2, 5, 2 }, { 3, 1, 3 }, { 3, 4, 9 }, { 3, 6, 4 },
    { 4, 3, 1 }, { 5, 2, 3 }, { 5, 6, 6 }, { 6, 4, 5 } };

int main()
{
    for (const auto& [u, v, w] : es)
        cout << "(" << u << ", " << v << ", " << w << ")\n";

    return 0;
}
```

Representação implícita

- Na representação implícita os vértices e arestas são dadas implicitamente, por meio de relações matemáticas ou pelo contexto do problema
- Por exemplo, o grafo completo formado pelos pontos do plano cartesiano cujas coordenadas são números inteiros, e cujos pesos das arestas são dados pela distância euclidiana, é um grafo implícito
- Esta representação é adequada quando o grafo for muito complexo ou quando a relação entre os vértices é óbvia ou diretamente computável
- Também pode ser utilizada para construir um grafo sob demanda, o que é útil em problemas de teoria dos jogos ou em simulações

Referências

1. **HALIM**, Felix; **HALIM**, Steve. *Competitive Programming 3*, 2010.
2. **LAAKSONEN**, Antti. *Competitive Programmer's Handbook*, 2018.
3. **SKIENA**, Steven S.; **REVILLA**, Miguel A. *Programming Challenges*, 2003.