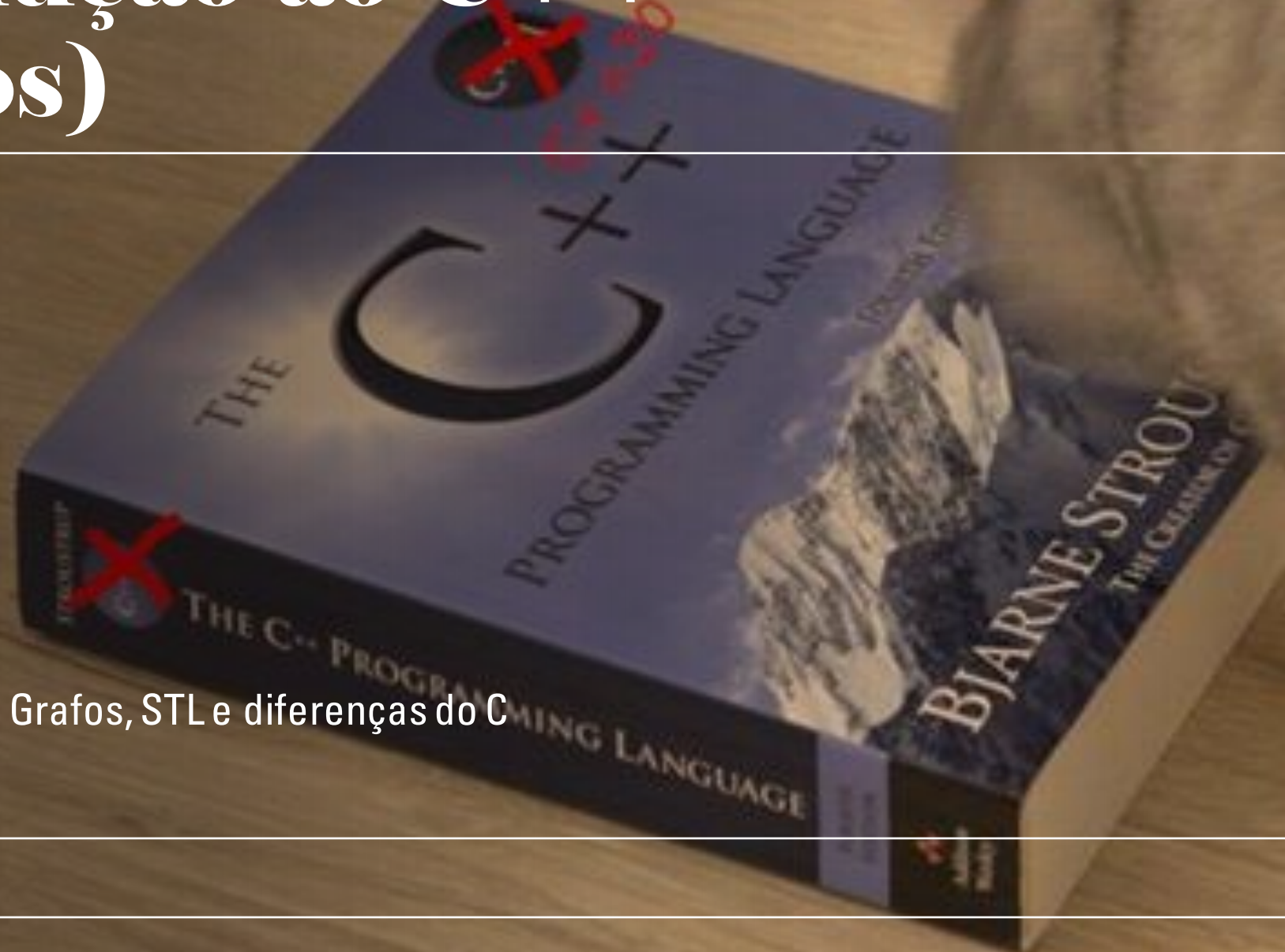


# Introdução ao C++ (Grafos)



Representações de Grafos, STL e diferenças do C pro C++

# Como compilar C++

- No Linux, você pode instalar o GCC (*GNU Compiler Collection*) através do gerenciador de pacotes do sistema.
- Crie um arquivo com extensão **.cpp** que contenha o código-fonte do programa C++. Por exemplo, **main.cpp**.

```
g++ main.cpp -o main
```

# Atribuição por referência

Atribuição por Referência:  
Ocorre quando uma variável  
não contém diretamente o  
valor do objeto, mas sim o  
endereço de memória onde o  
objeto está armazenado.

Em C, não existe atribuição  
por referência, e  
sim, atribuição por  
valor (copia).

C++ utiliza o `&` para atribuir  
uma variável por referência.

```

9 #include <stdio.h>
8
7 void change(int x) {
6     x += 1;
5 }
4
3 int main() {
2     int n = 10;
1     change(n);
10    printf("%d\n", n);
1 }

```

```

2 → aula_cpp gcc c_code_reference.c -o r
1 10
3 → aula_cpp █
1

```

# Isso não funciona

- A variável `x` recebe apenas o valor 10 de `n`.
- Qualquer mudança não será refletida na variável original.

```
8 #include <stdio.h>
7
6 void change(int *x) {
5     *x += 1;
4 }
3
2 int main() {
1     int n = 10;
9     change(&n);
1     printf("%d\n", n);
2 }
```

```
2 → aula_cpp gcc c_code_reference.c -o
1 11
3 → aula_cpp █
```

# Esse é o jeito de fazer em C

- Passa-se o endereço das variáveis como parâmetros, alterando-as em outros escopos.
- Perceba que foi feita uma atribuição por valor no endereço de n.

```

11 #include <bits/stdc++.h>
10
9 using namespace std;
8
7 void change(int &x) {
6     x += 1;
5 }
4
3 int main() {
2     int n = 10;
1     change(n);
12    printf("%d\n", n);
1 }

```

```

2 → aula_cpp g++ cpp_reference.cpp -o ru
1 11
3 → aula_cpp █
1

```

# Agora em C++

- Em C++ é mais simples.
- Atribuição por referência em todas as variáveis que têm &.

00

C++ suporta a orientação a objetos

C++ Standard Library

Sobrecarga de operadores (cin e cout).

Várias classes, funções e variáveis para facilitar a programação.

string, vector, queue, stack

# cin

- O objeto cin representa o stream de entrada no C++.
- O operador >> sobrecarregado executa a entrada com streams em C++, usando o comando cin para aquisição de dados.
- Variáveis podem ser usadas para o armazenamento das informações.

```
10 #include <bits/stdc++.h>
9
8 using namespace std;
7
6 int main() {
5     int n; cin >> n;
4     int a[n];
3     for(int i = 0; i < n; i++) {
2         cin >> a[i];
1     }
11 printf("end\n");
1 }

1 5
2 0 1 2 3 4

2 → aula_cpp g++ cpp_reference.cpp -o run && ./run < in
1 end
3 → aula_cpp █
1
```



# cout

- O objeto cout representa o stream de saída no C++.
- O operador << sobrecarregado executa a saída (imprime na tela) com streams em C++.
- O objeto cout é usado em conjunto com ele para a impressão de dados.

```
11 #include <bits/stdc++.h>
10
9  using namespace std;
8
7  int main() {
6      int n; cin >> n;
5      int a[n];
4      for(int i = 0; i < n; i++) {
3          cin >> a[i];
2      }
1      for(int i = 0; i < n; i++) {
12         cout << a[i] << ",";
1
2      cout << endl;
3  }
```

```
2 → aula_cpp g++ cpp_reference.cpp -o run && ./run < in
1 0,1,2,3,4,
60 → aula_cpp █
```

```
1 5
2 0 1 2 3 4
```

# Templates de classe

- Templates tornam possível criar classes e funções genéricas.
- `vector<bool>`
- Vetor de **booleanos**.

```
20 #include <iostream>
19 using namespace std;
18
17 template <class T>
16 class Number {
15     private:
14         T num;
13
12     public:
11         Number(T n) : num(n) {}    // constructor
10
9         T getNum() {
8             return num;
7         }
6 };
5
4 int main() {
3     Number<int> numberInt(7);
2     Number<double> numberDouble(7.7);
1
21     cout << "int Number = " << numberInt.getNum() << endl;
1     cout << "double Number = " << numberDouble.getNum() << endl;
2
3     return 0;
4 }
```

# Iterando em listas de itens

- Parecido com o *in* do Python, o `:` do C++ itera em uma lista de elementos.
- `vector`, `set` e etc.

```
9 #include <bits/stdc++.h>
8 using namespace std;
7
6
5
4 int main() {
3     vector<int> a = {1, 2, 3, 4, 5};
2     for(int x : a) cout << x << " ";
1     cout << endl;
10 }
```

```
2 → aula_cpp g++ cpp_reference.cpp -o run && ./run
1 1 2 3 4 5
3 → aula_cpp █
```

# STL

1. Vector
2. Queue
3. Set
4. Map
5. Pair

## Standard Template Library

- A STL C++ é uma biblioteca padronizada de funções, que oferece ao desenvolver um conjunto de classes de uso genérico, descrevendo contêineres (estruturas de dados, como pilhas, listas e filas), iteradores e algoritmos básicos (principalmente os destinados a busca e classificação).

# Vector

Método	Atributo
operator[ ]	Acessa o elemento.
front	Acessa o primeiro elemento.
back	Acessa o último elemento.
push_back	Adiciona elemento no final.
pop_back	Deleta o último elemento.

# Queue

Método	Atributo
empty	Testa se a fila está vazia.
push	Adiciona um elemento no final da fila.
pop	Remove o primeiro elemento da lista.
front	Acessa o primeiro elemento.
back	Acessa o ultimo elemento.

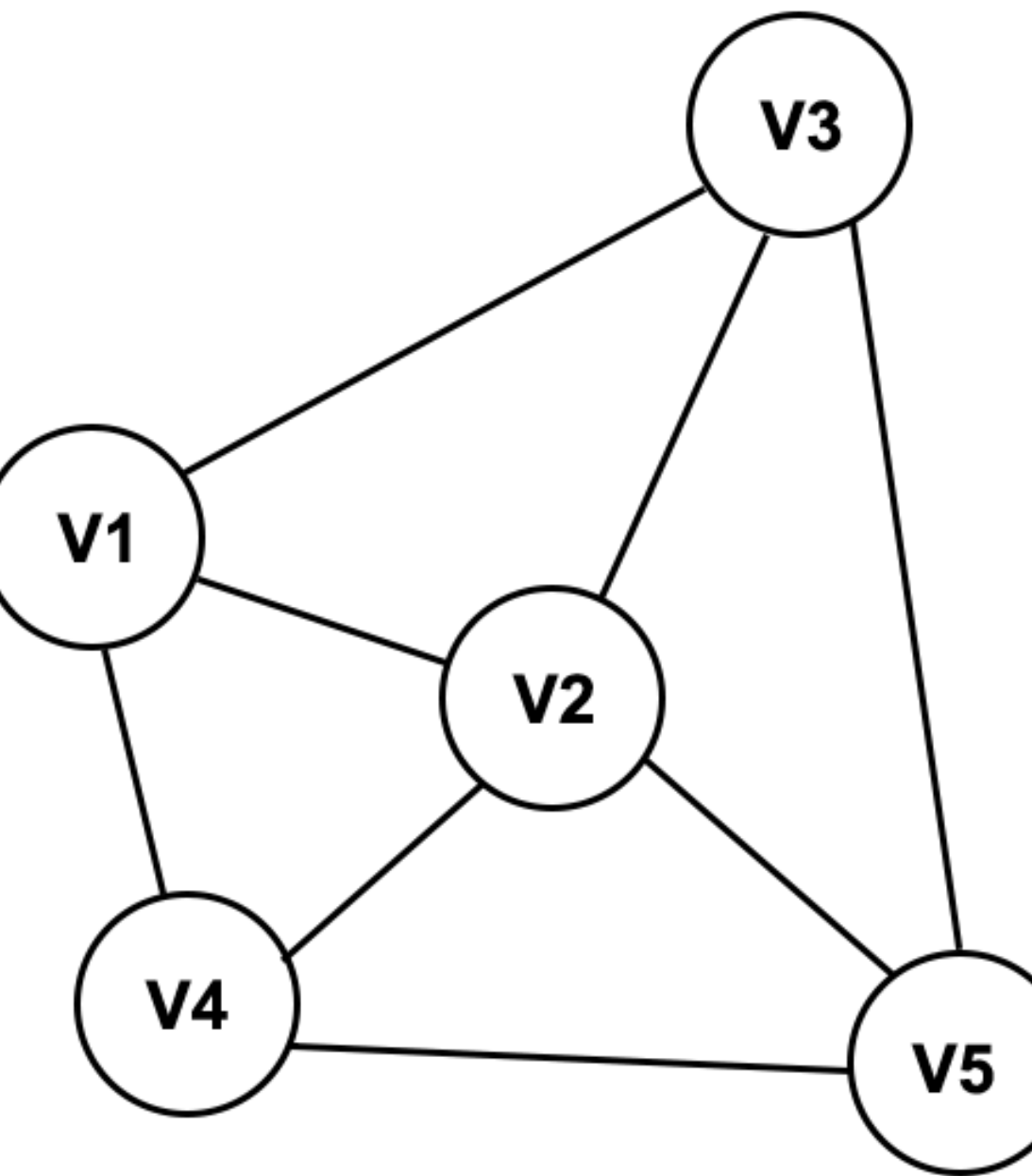
# Set

Método	Atributo
empty	Testa se o set está vazio.
size	Retorna o número de elementos.
insert	Insere um elemento.
erase	Remove um elemento.
clear	Limpa o conjunto.

# Pair

Método	Atributo
first	Acessa o primeiro elemento do par.
second	Acessa o segundo elemento do par.






# Grafos e representações

- Lista de adjacência
- Matriz de adjacência

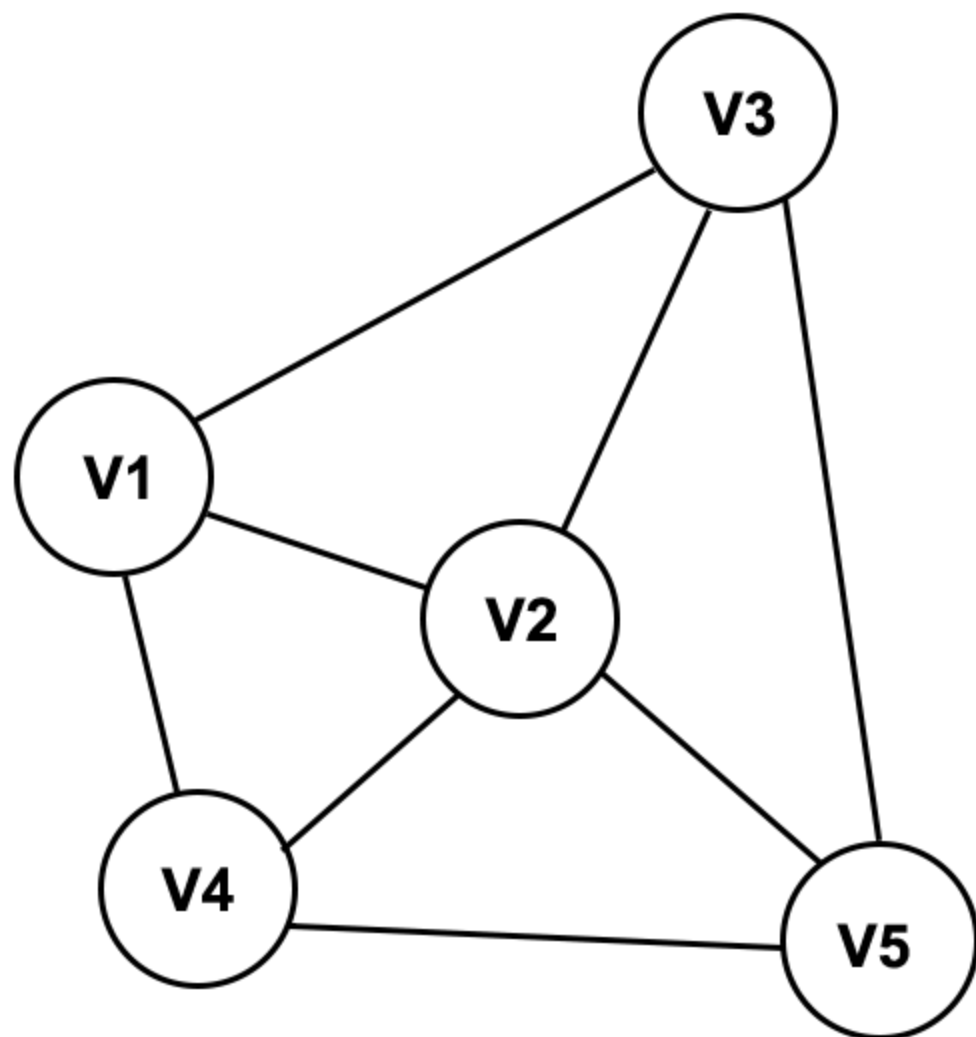
# Receber grafos como entrada

```
n m
u1 v1
u2 v2
u3 v3
...
um-1 vm-1
um vm
```

Dois inteiros  $n$  e  $m$ , seguidos de  $m$  linhas.



Cada linha contém dois inteiros  $u$  e  $v$ ,  
indicando que os vértices  $u$  e  $v$  são  
vizinhos ( $uv$  é aresta)



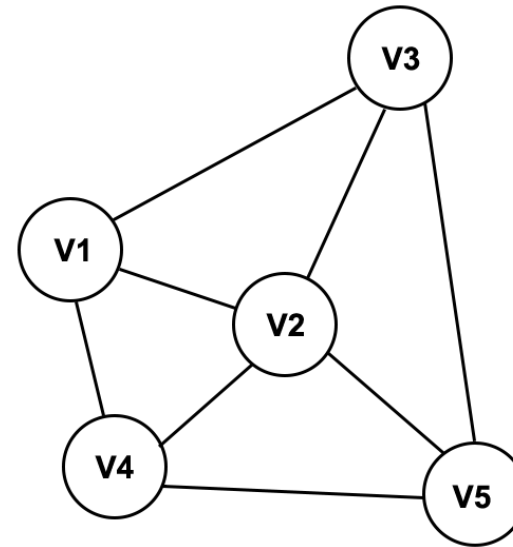
# Exemplo

```
5 8
1 2
1 3
1 4
2 3
2 4
2 5
3 5
4 5
```

# Matriz de adjacência

$M_{uv}$  é 1 se  $uv$  é aresta, 0 c.c.

-	v1	v2	v3	v4	v5
v1	0	1	1	1	0
v2	1	0	1	1	1
v3	1	1	0	0	1
v4	1	1	0	0	1
v5	0	1	1	1	0



```

13 #include <bits/stdc++.h>
12 using namespace std;
11
10 int main() {
9     int n, m;
8     cin >> n >> m;
7     vector<vector<int>> matriz(n + 1, vector<int>(n + 1, 0));
6
5     for(int i = 0; i < m; i++) {
4         int u, v;
3         cin >> u >> v;
2         matriz[u][v] = 1;
1         matriz[v][u] = 1;
14    }
1
2    //lido, agora printar
3
4    for(int i = 1; i <= n; i++) {
5        for(int j = 1; j <= n; j++) {
6            cout << matriz[i][j] << " ";
7
8        }
9        cout << endl;
10 }

```

```

6 5 8
5 1 2
4 1 3
3 1 4
2 2 3
1 2 4
7 2 5
1 3 5
2 4 5

```

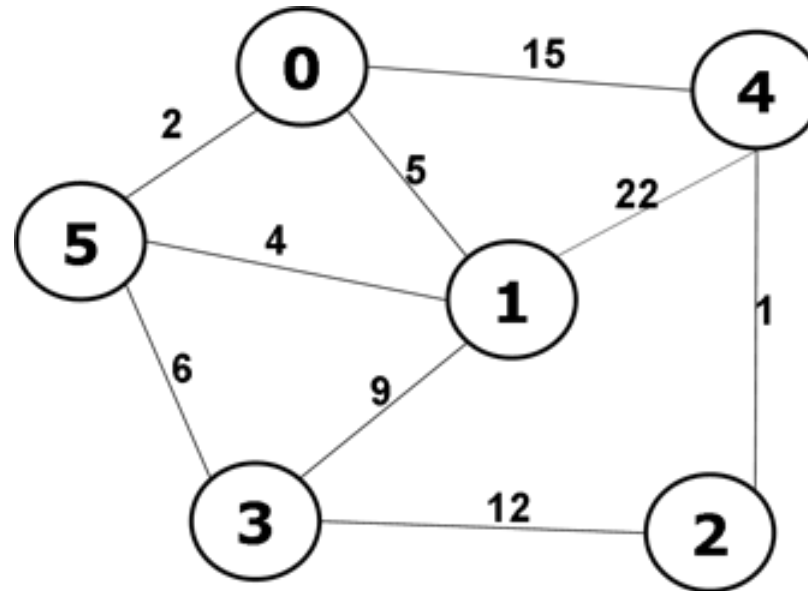
```

6 → aula_cpp g++ cpp_reference.cpp -o run && ./run < in
5 0 1 1 1 0
4 1 0 1 1 1
3 1 1 0 0 1
2 1 1 0 0 1
1 0 1 1 1 0
7 → aula_cpp

```

# Lendo arestas com peso

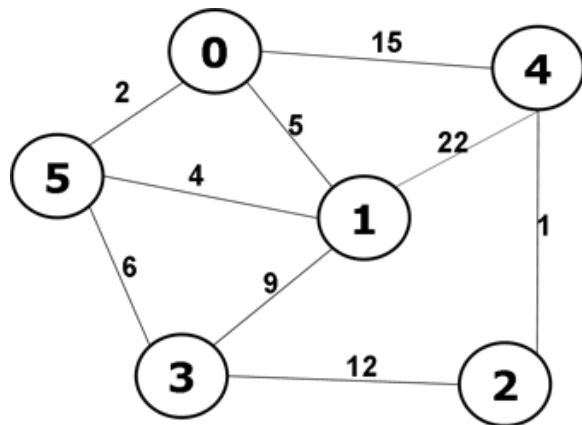
- Cada aresta vem com o seu respectivo peso do lado.



$n$	$m$
$u_1$	$v_1 \ p_1$
$u_2$	$v_2 \ p_2$
$u_3$	$v_3 \ p_3$
$\dots$	
$u_{m-1}$	$v_{m-1} \ p_{m-1}$
$u_m$	$v_m \ p_m$

# Peso em matriz

- O recomendado é ter uma matriz para representar a existência de uma aresta, e outra para representar seu peso



$M_{uv}$  é 1 se  $uv$  é aresta, 0 c.c.

	0	1	2	3	4	5
0	0	1	0	0	1	1
1	1	0	0	1	1	1
2	0	0	0	1	1	0
3	0	1	1	0	0	1
4	1	1	1	0	0	0
5	1	1	0	1	0	0

$P_{uv}$  é o peso de  $uv$

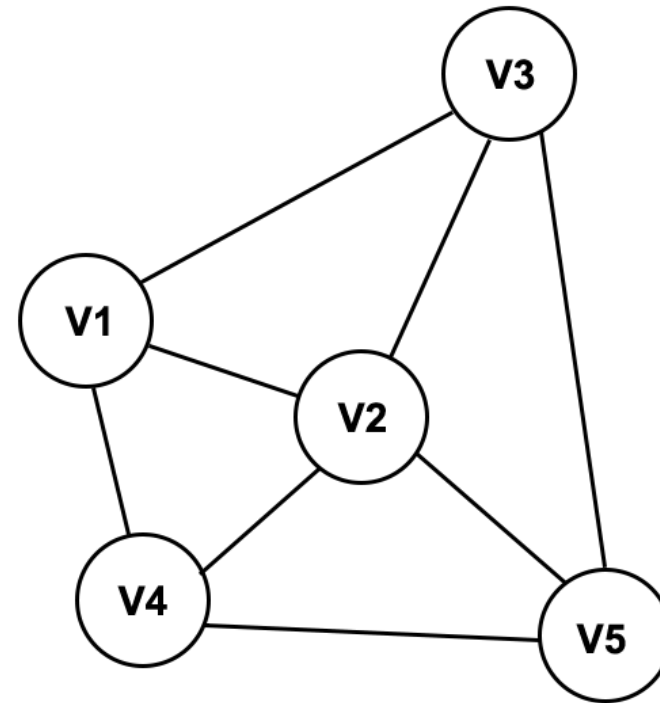
	0	1	2	3	4	5
0		5			15	2
1	5			9	4	4
2				12	1	
3		9	12			6
4	15	22	1			
5	2	4		6		

# Lista de adjacência

- $N(v)$  representa um conjunto com todos os vizinhos de  $v$ .
- Em C++ pode ser representado com um vetor de listas.

$$ADJ_u = N(u)$$

v1	[v2, v3, v4]
v2	[v1, v3, v4, v5]
v3	[v1, v2, v5]
v4	[v1, v2, v5]
v5	[v2, v3, v4]





```

20 #include <bits/stdc++.h>
19 using namespace std;
18
17 int main() {
16     int n, m;
15     cin >> n >> m;
14     vector<vector<int>> adj(n + 1, vector<int>(0));
13
12     for(int i = 0; i < m; i++) {
11         int u, v;
10         cin >> u >> v;
9         adj[u].push_back(v);
8         adj[v].push_back(u);
7     }
6     //lido, agora printar
5     for(int i = 1; i <= n; i++) {
4         cout << i << ": ";
3         for(int neighbor : adj[i]) cout << neighbor << " ";
2         cout << endl;
1     }
21

```

```

6 5 8
5 1 2
4 1 3
3 1 4
2 2 3
1 2 4
7 2 5
1 3 5
2 4 5

```

cpp\_reference.cpp [+]

21,1

All

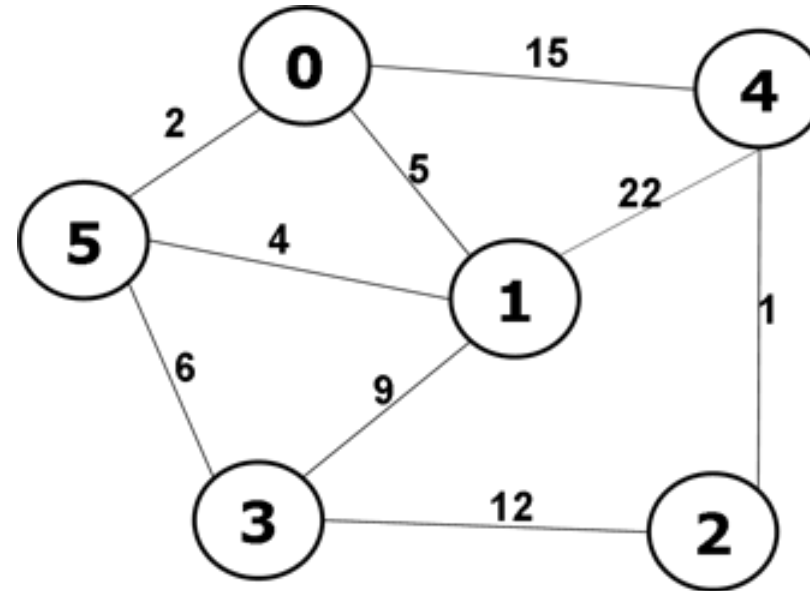
```

6 → aula_cpp g++ cpp_reference.cpp -o run && ./run < in
5 1: 2 3 4
4 2: 1 3 4 5
3 3: 1 2 5
2 4: 1 2 5
1 5: 2 3 4
7 → aula_cpp

```

# Peso em lista

- Podemos usar o `pair` para não só guardar o valor do vértice vizinho, como o peso de sua aresta.
- $ADJ_u = \{(P_{uv}, v) \text{ para todo } v \text{ em } N(u)\}$  onde  $P_{uv}$  é o peso da aresta  $uv$ .
- A lista é composta por listas de pares (*peso, vizinho*).



v0	[(5, v1), (15, v4), (2, v5)]
v1	[(5, v0), (22, v4), (4, v5)]
v2	[(12, v3), (1, v4)]
v3	[(9, v1), (6, v5)]
v4	[(15, v0), (22, v1), (1, v2)]
v5	[(2, v0), (4, v1), (6, v3)]

```

24 #include <bits/stdc++.h>
23
22 using namespace std;
21
20 int main() {
19     int n, m; cin >> n >> m;
18     vector<vector<pair<int, int>>> adj(n);
17     //n coloquei n + 1 pq eh 0-indexado
16     for(int i = 0; i < m; i++) {
15         int u, v, peso;
14         cin >> u >> v >> peso;
13         adj[u].push_back({peso, v});
12         adj[v].push_back({peso, u});
11     }
10
9     //printando os pares
8     for(int u = 0; u < n; u++) {
7         cout << u << ": ";
6         cout << '[';
5         for(pair<int, int> par : adj[u]) {
4             cout << '(' << par.first << ' ' << par.second << ')' << ' ';
3         }
2         cout << ']' << endl;
1     }
25

```

# Implementando breadth-first search (BFS)

## Busca em largura

- Explora cada vértice uma única vez priorizando os vizinhos.
- Utiliza uma fila em seu funcionamento.
- **Explicar no Jamboard.**

## Algoritmo

1. Adicione o nó raiz na fila
2. Enquanto a fila não estiver vazia:
  1. Pegue o primeiro elemento da fila.
  2. Percorra todos os seus vizinhos, adicionando os que não foram visitados ainda na fila.
  3. Marque-os como visitados.
  4. Remova o primeiro elemento da fila.