Introdução à Teoria dos Grafos

Prof. Alexandre Noma

Aula passada

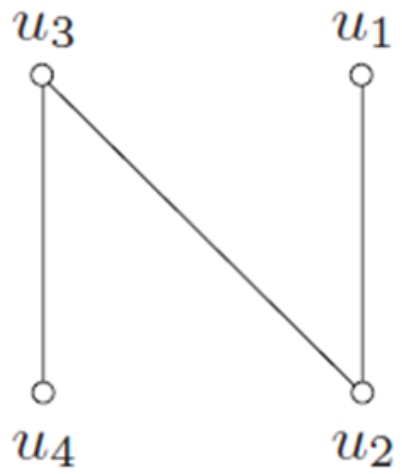
- O que é um grafo?
- Um grafo pode ser
  - simples
  - completo
  - bipartido
  - Conexo
- Mais propriedades:
  - complemento
  - planaridade
  - isomorfismo

Hoje: Mais propriedades...

- caminho
- ciclo

- Um caminho é uma sequência de vértices conectados por arestas.

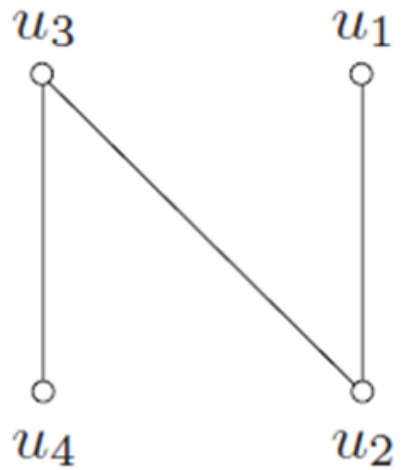
$$\langle u_1, u_2, u_3, u_4 \rangle$$



(a)

- Um caminho é uma sequência de vértices conectados por arestas.
  - Um caminho é simples se não há repetição de vértices.

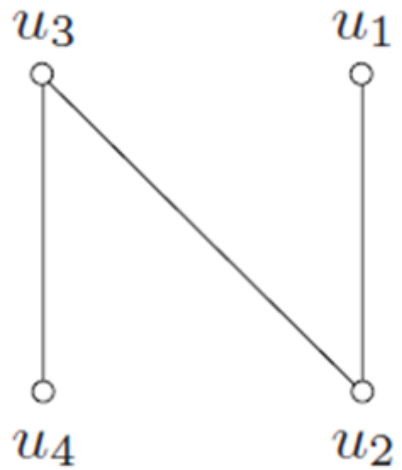
$$\langle u_1, u_2, u_3, u_4 \rangle$$



(a)

- Um caminho é uma sequência de vértices conectados por arestas.
  - Um caminho é simples se não há repetição de vértices.

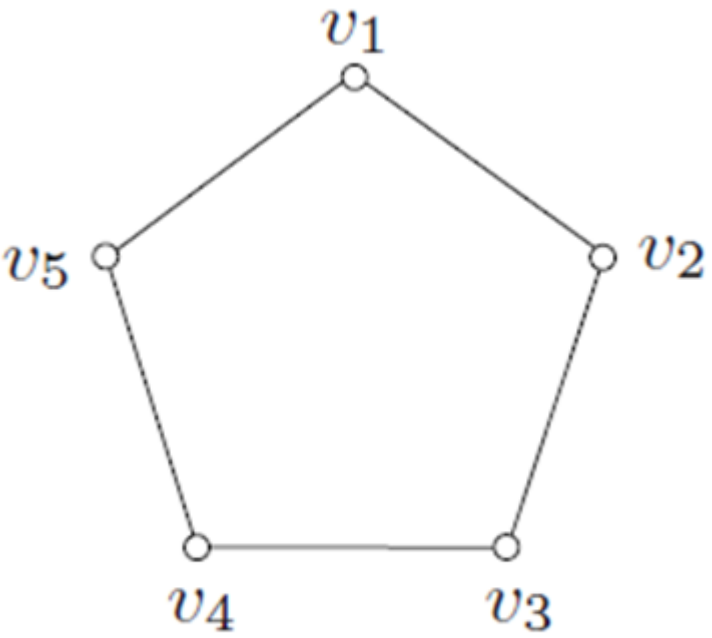
$\langle u_1, u_2, u_3, u_4 \rangle$ ✔



(a)

- Um ciclo é um caminho, sem repetição de vértices, exceto pelo primeiro vértice, que é igual ao último.
  - "É uma sequência que começa e termina num mesmo vértice."

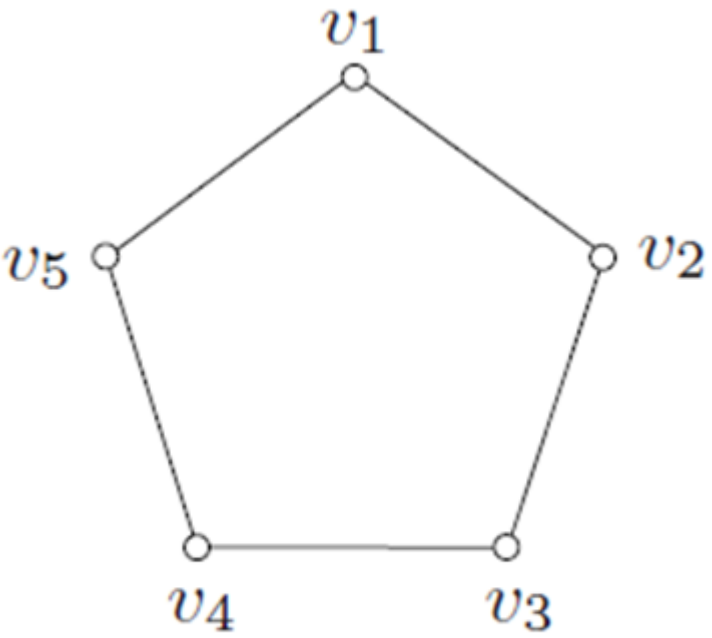
$$\langle v_1, v_2, v_3, v_4, v_5, v_1 \rangle$$



(b)

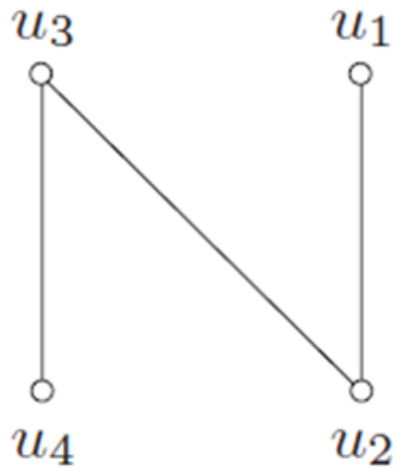
- Um ciclo é um caminho, sem repetição de vértices, exceto pelo primeiro vértice, que é igual ao último.
  - "É uma sequência que começa e termina num mesmo vértice."

$$\langle v_1, v_2, v_3, v_4, v_5, v_1 \rangle$$



(b)

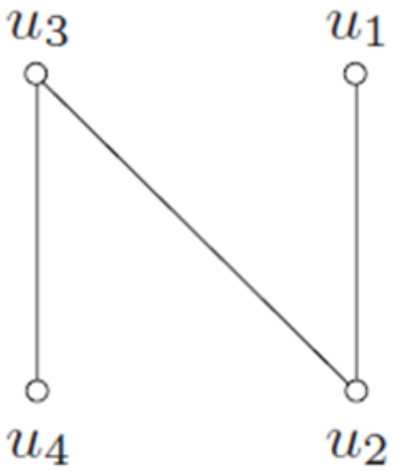
- O grau de um vértice é o “número de arestas conectadas ao vértice”.



(a)

grau(u_1) = ?
grau(u_2) = ?
grau(u_3) = ?
grau(u_4) = ?

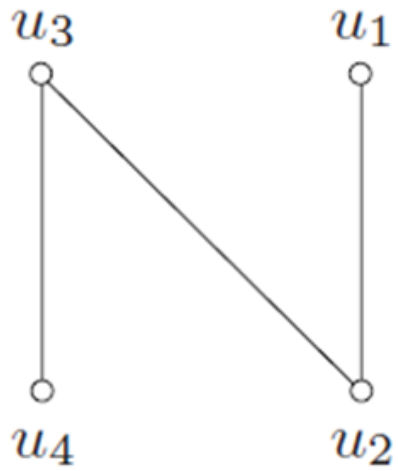
- O grau de um vértice é o “número de arestas conectadas ao vértice”.



(a)

grau(u_1) = 1
grau(u_2) = ?
grau(u_3) = ?
grau(u_4) = 1

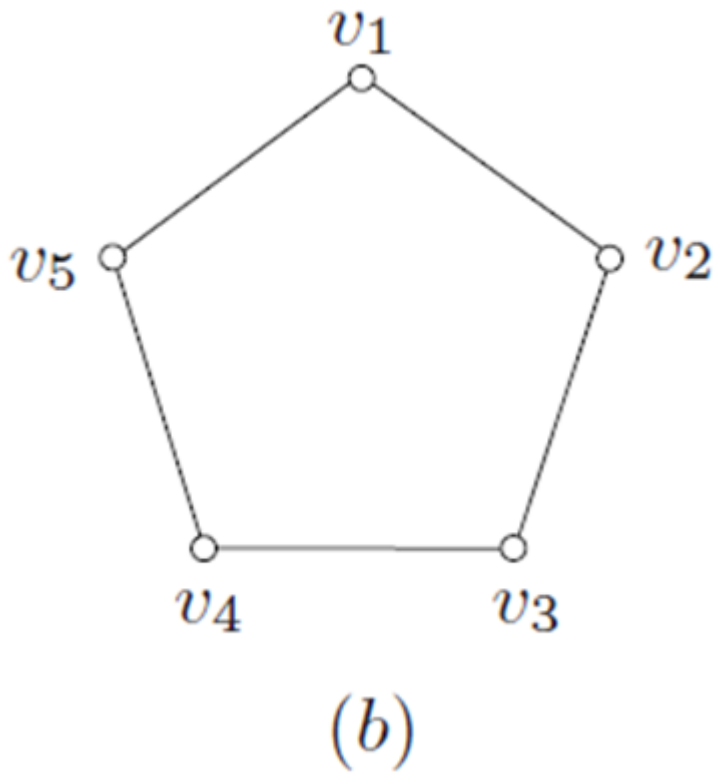
- O grau de um vértice é o “número de arestas conectadas ao vértice”.



(a)

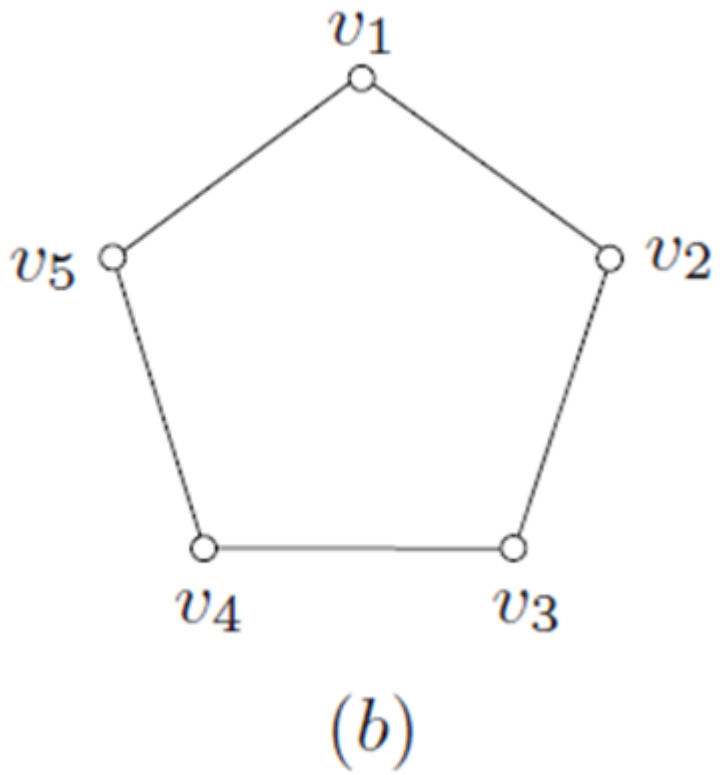
grau(u_1) = 1
grau(u_2) = 2
grau(u_3) = 2
grau(u_4) = 1

- O grau de um vértice é o “número de arestas conectadas ao vértice”.



grau(v_1) = ?
grau(v_2) = ?
grau(v_3) = ?
grau(v_4) = ?
grau(v_5) = ?

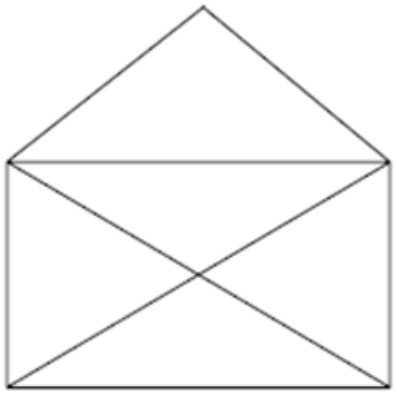
- O grau de um vértice é o “número de arestas conectadas ao vértice”.



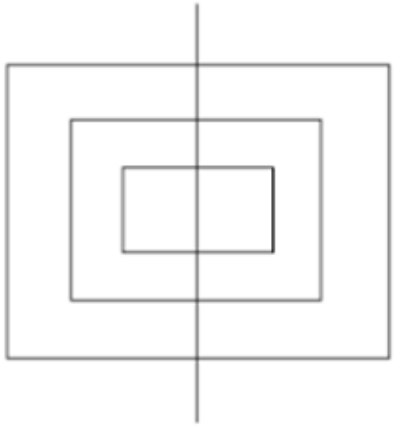
grau(v_1) = 2
grau(v_2) = 2
grau(v_3) = 2
grau(v_4) = 2
grau(v_5) = 2

Exercício

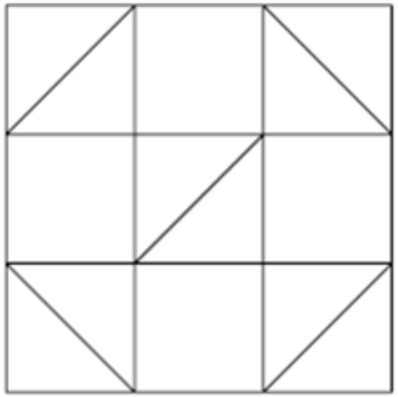
– Será que você consegue desenhar cada figura "sem tirar o lápis do papel"?



(a)

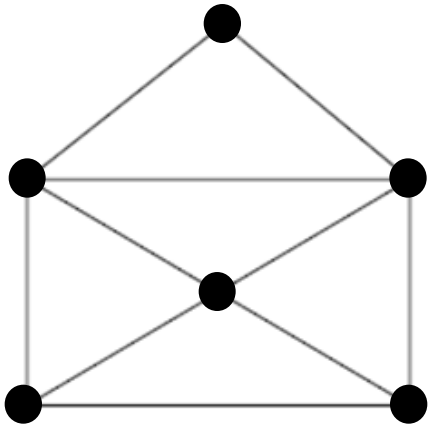


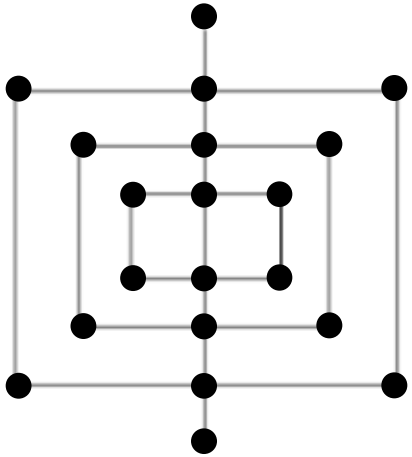
(b)

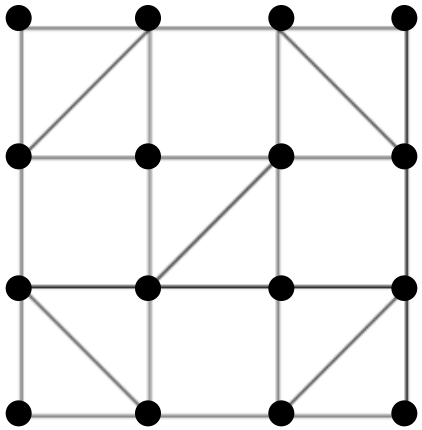


(c)

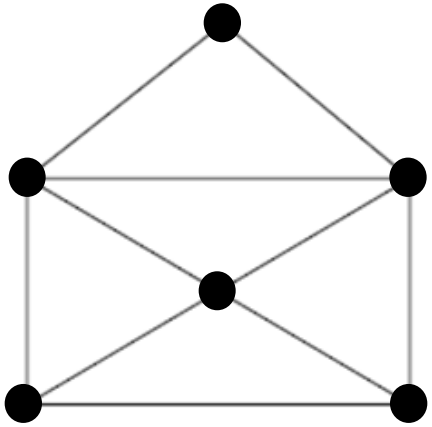
- Um caminho euleriano é um caminho que visita todas as arestas de um grafo, cada aresta visitada uma única vez.

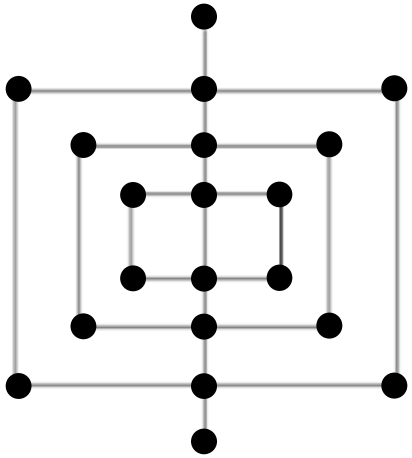


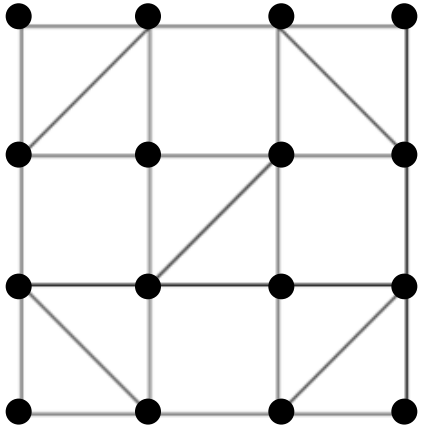




- Um caminho euleriano é um caminho que visita todas as arestas de um grafo, cada aresta visitada uma única vez.



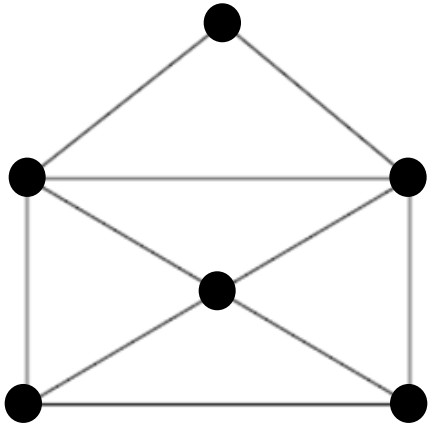


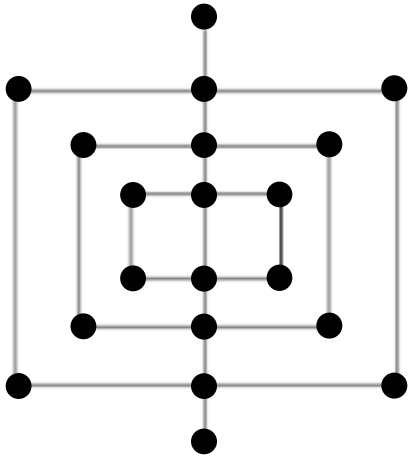


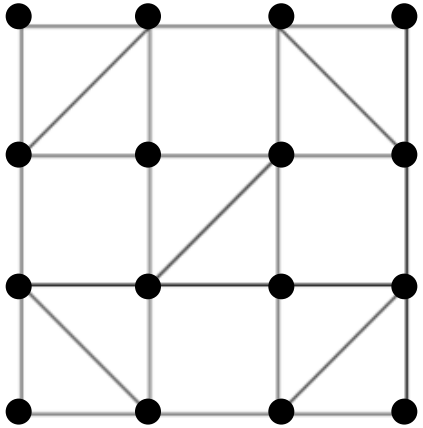
- O que acontece com os graus dos vértices...?

("vértices internos")

- Um caminho euleriano é um caminho que visita todas as arestas de um grafo, cada aresta visitada uma única vez.







- O que acontece com os graus dos vértices...?

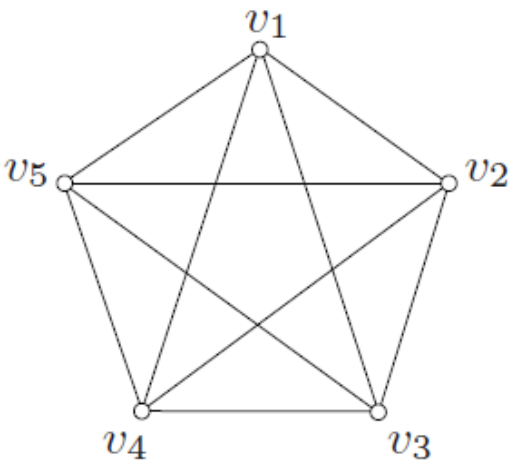
("vértices internos": grau 2)

Como representar de grafos no computador?

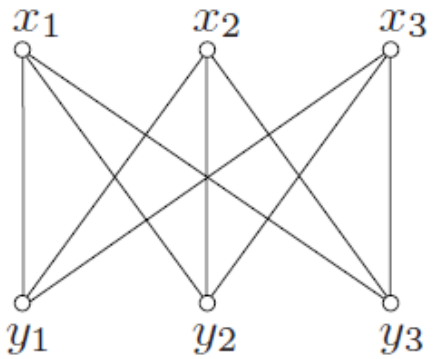
Grafo

- não dirigido ou não orientado
- dirigido ou orientado

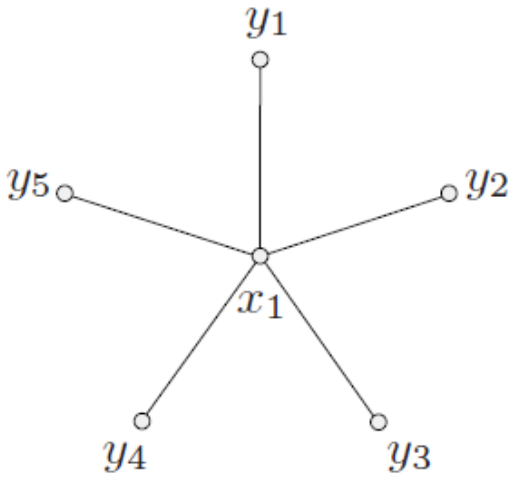
- Orientado (dirigido) ???



(a)

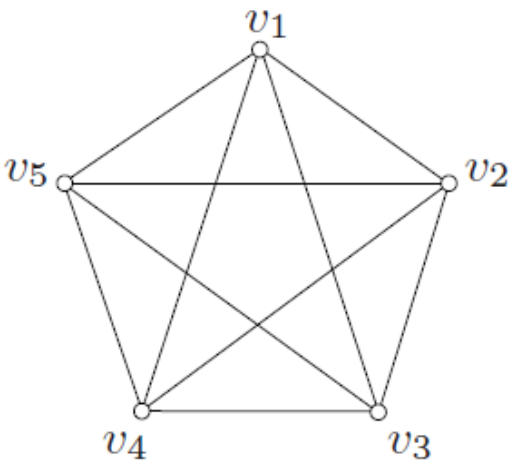


(b)

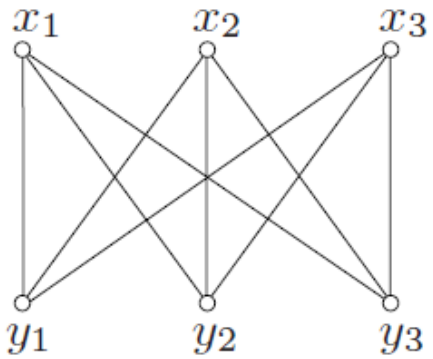


(c)

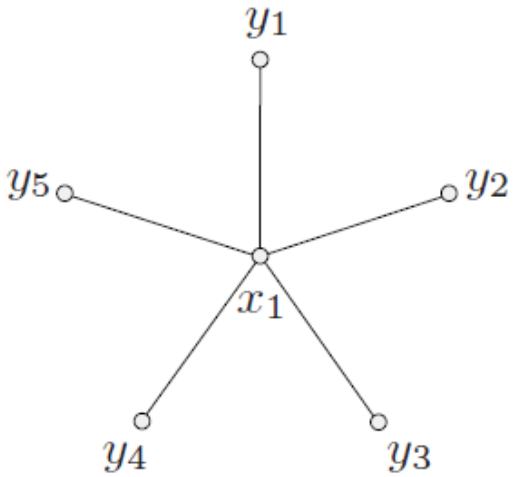
- Em um grafo não-dirigido, os vértices podem ser conectados por arestas.
  (Sem restrições para o sentido de visita de vértices).



(a)

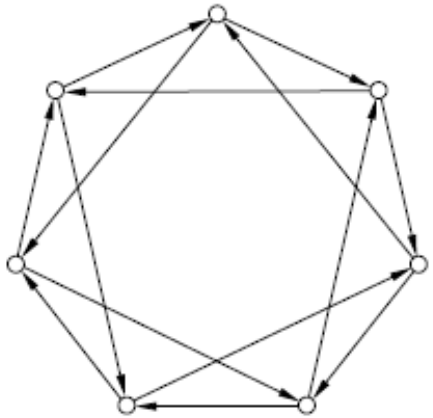


(b)

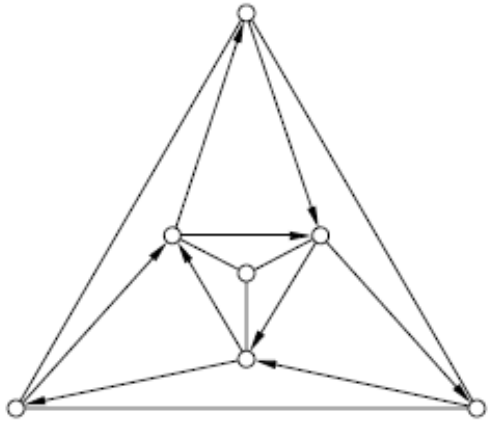


(c)

- Em um grafo dirigido (= digrafo), os vértices podem ser conectados por arcos (com restrições de sentido).



(a)

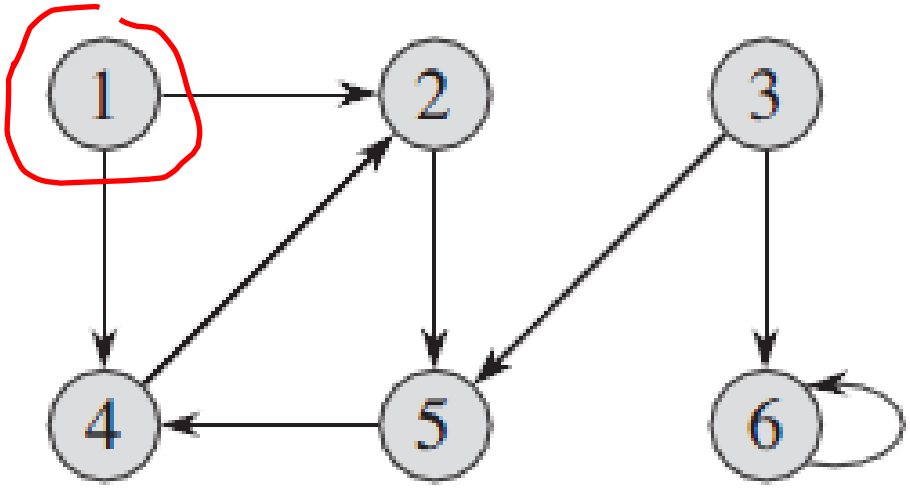


(b)

- Como representar um grafo no computador?

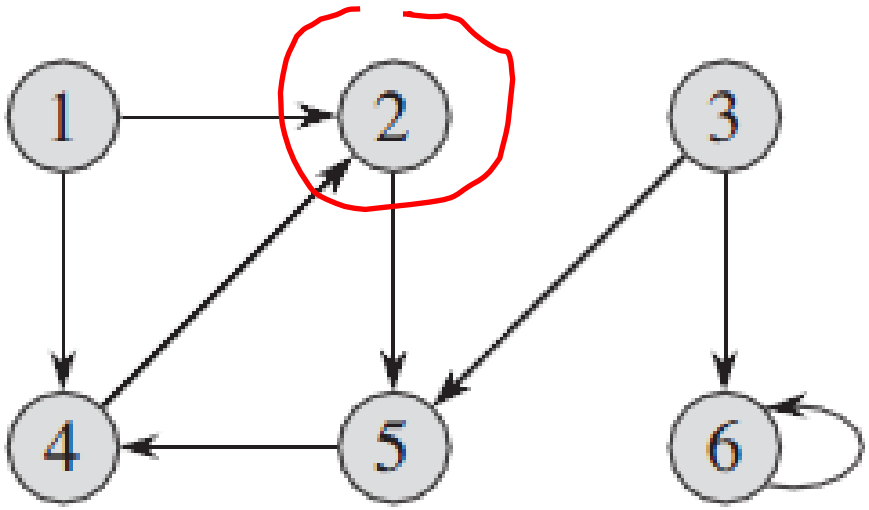
- Como representar um grafo no computador?
- (1) Matriz de adjacências
- (2) Listas de adjacências

(1) Matriz de adjacências



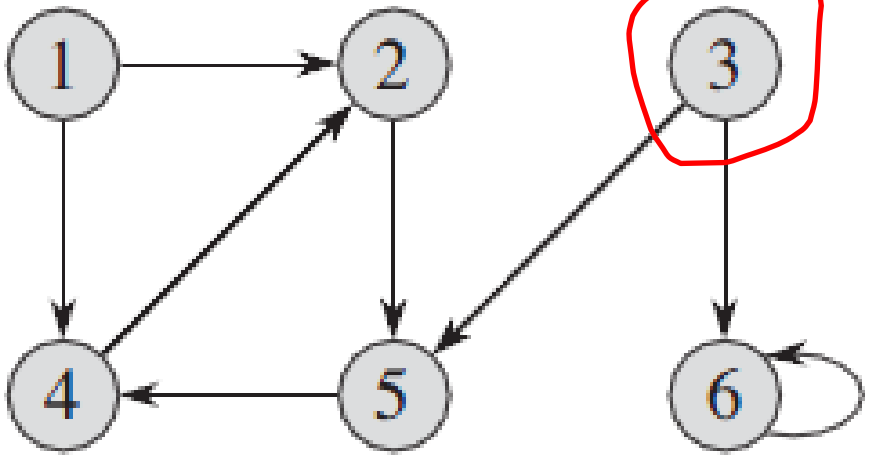
	1	2	3	4	5	6
1	0	1	0	1	0	0
2						
3						
4						
5						
6						

(1) Matriz de adjacências



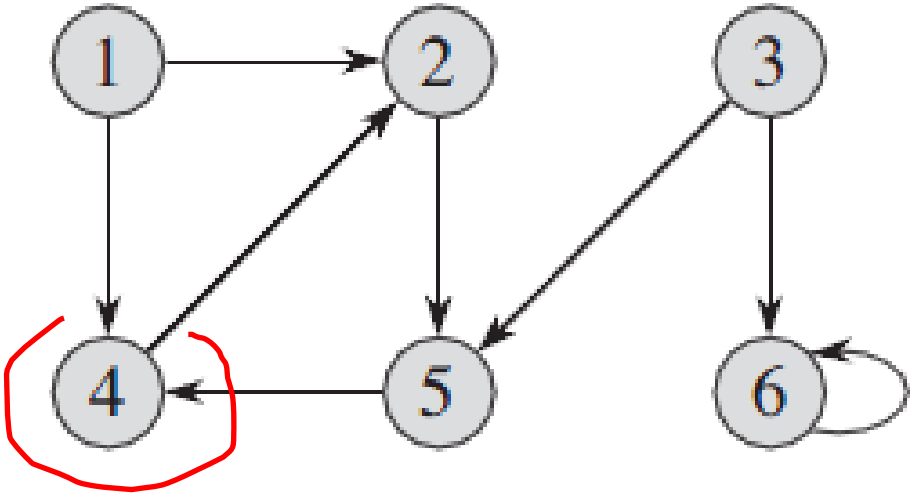
	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3						
4						
5						
6						

(1) Matriz de adjacências



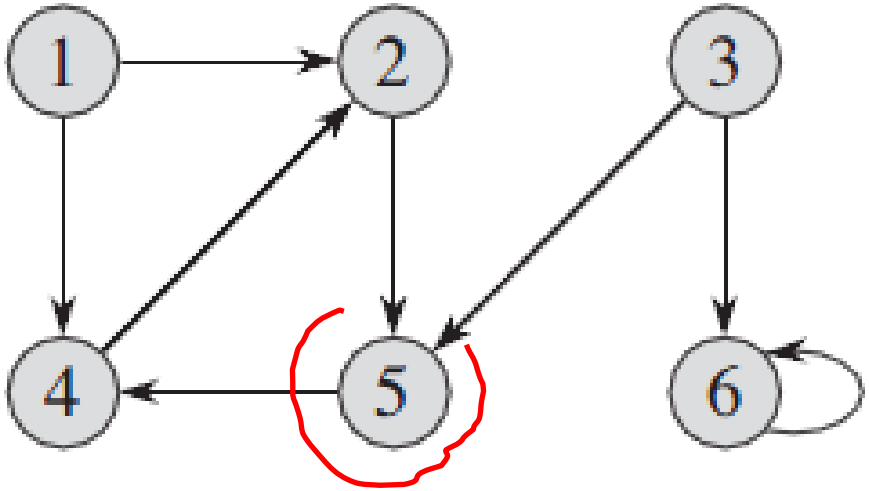
	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4						
5						
6						

(1) Matriz de adjacências



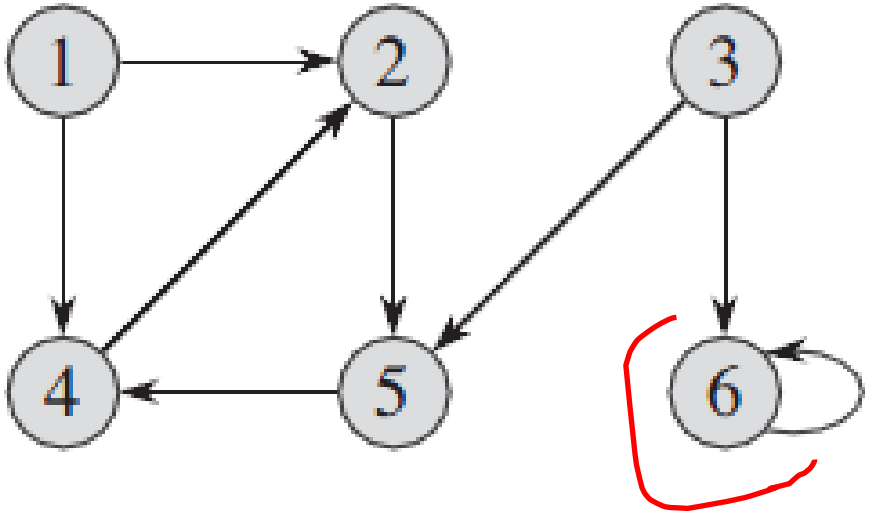
	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5						
6						

(1) Matriz de adjacências



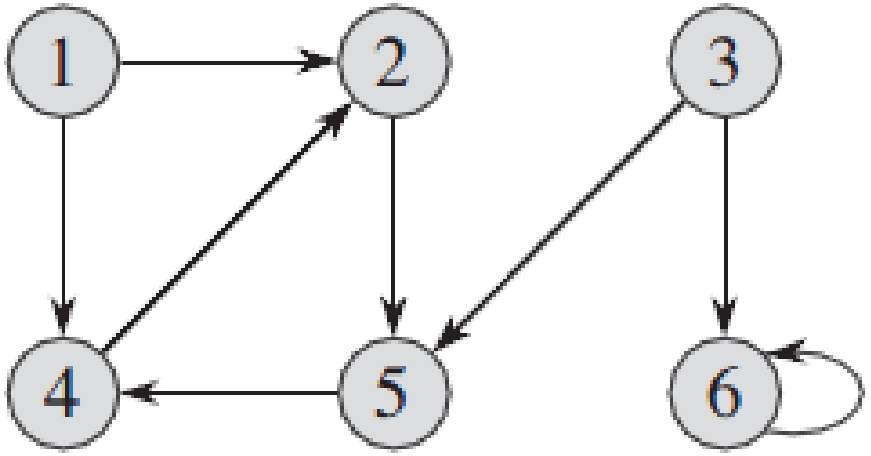
	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6						

(1) Matriz de adjacências



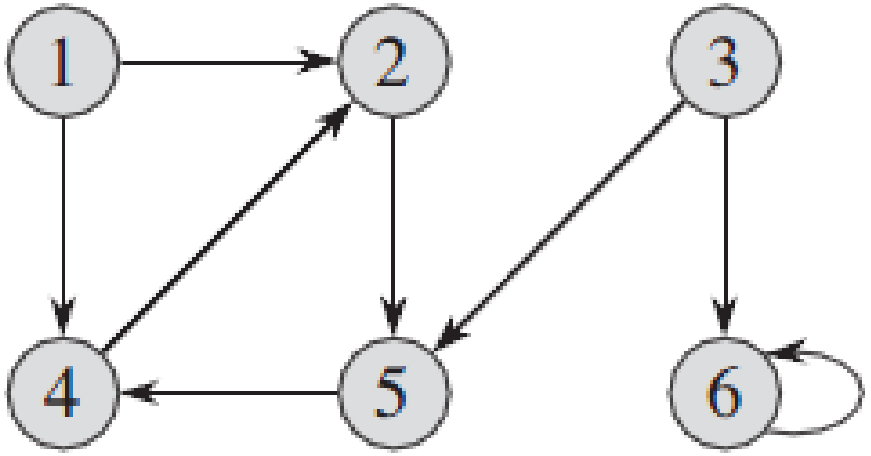
	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(2) Listas de adjacências



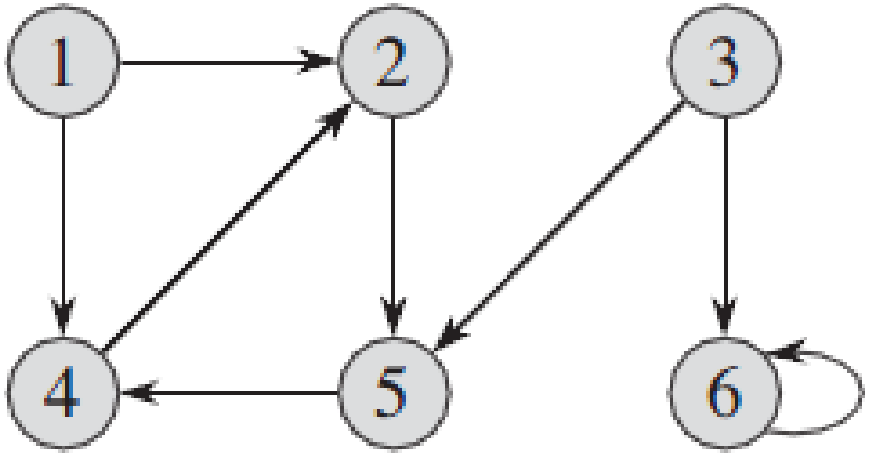
1: 2, 4
2: ?
3: ?
4: ?
5: ?
6: ?

(2) Listas de adjacências



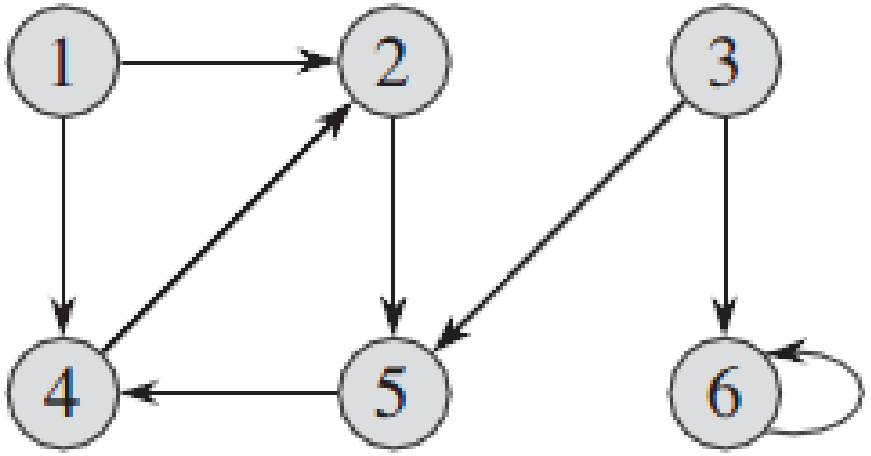
1: 2, 4
2: 5
3: ?
4: ?
5: ?
6: ?

(2) Listas de adjacências



1: 2, 4
2: 5
3: 5, 6
4: ?
5: ?
6: ?

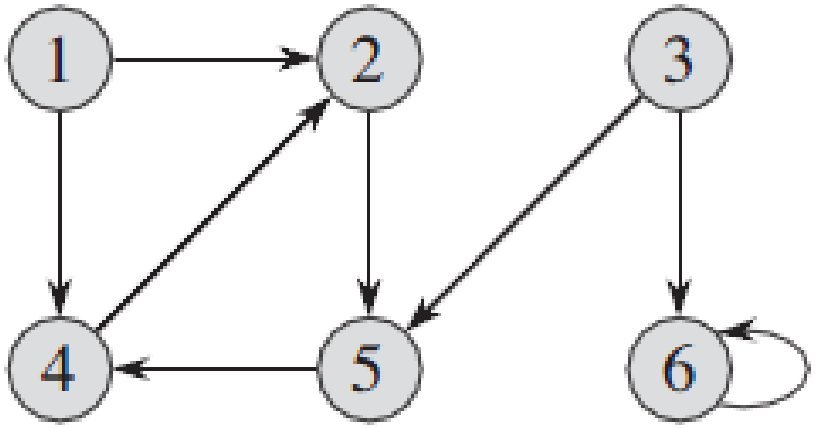
(2) Listas de adjacências



1: 2, 4
2: 5
3: 5, 6
4: 2
5: 4
6: 6

Notação

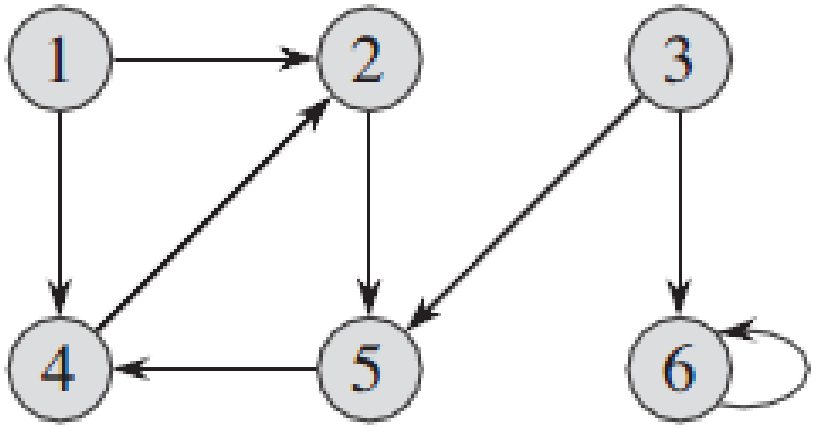
- adjacência = "vizinhança"
- lista de adjacências = lista dos vizinhos



Exemplo: Adj(1) = [2, 4]

Notação

- adjacência = "vizinhança"
- lista de adjacências = lista dos vizinhos



Exemplo: Adj(1) = [2, 4]
Adj(2) = [5]
Adj(3) = [5, 6]
...

Exercício

- Dadas as listas de adjacência:

```
1: 2
2: 3
3: 4, 5, 8
4:
5: 6, 7
6: 1
7: 9
8: 10
9: 10
10:
```

- Construir o grafo.

Exercício Programa

- 01-leGrafo.py

Consumo de Tempo

Notação O ("ó grande")

Notação O ("ó grande")

- O que é?

- Pra que serve?

Alguns exemplos

- $O(1)$: ???
- $O(\log n)$: ???
- $O(n)$: ???
- $O(n \log n)$: ???
- $O(n^2)$: ???
- $O(n^3)$: ???
- $O(2^n)$: ???

Alguns exemplos

- $O(1)$: constante
- $O(\log n)$: log de n
- $O(n)$: linear
- $O(n \log n)$: n log de n
- $O(n^2)$: quadrático
- $O(n^3)$: cúbico
- $O(2^n)$: exponencial

Alguns exemplos

- $O(1)$: constante
- $O(\log n)$: log de n
- $O(n)$: linear
- $O(n \log n)$: n log de n
- $O(n^2)$: quadrático
- $O(n^3)$: cúbico
- $O(nk)$: polinomial
- $O(2^n)$: exponencial
- $O(kn)$, $O(n!)$, $O(n^n)$: exponencial

Alguns exemplos

- $O(1)$: constante
- Exemplos?

Alguns exemplos

- $O(n)$: linear
- Exemplo?

Alguns exemplos

- $O(n)$: linear
- Exemplo?
- O que é n ?

Alguns exemplos

- $O(n)$: linear
- Exemplo?
- O que é n ?
  - n é o tamanho da "entrada".

- $O(n)$: linear?

```c
#include <stdio.h>
#include <stdlib.h>
int main () {
    int n, i, *v;
    // tamanho n do vetor
    scanf ("%d", &n);
    // aloca memoria para vetor
    v = malloc (sizeof (int) * n);
    // le n inteiros e guarda no vetor
    for (i = 0; i < n; i++) {
        scanf ("%d", &v[i]);
    }
    // libera memoria
    free (v);
    return 0;
}
```

- $O(n)$: linear?

```
#include <stdio.h>                                  consumo
#include <stdlib.h>                                 de tempo
int main () {
1.   int n, i, *v;                  --------------   ???
     // tamanho n do vetor
2.   scanf ("%d", &n);              --------------   ???
     // aloca memoria para vetor
3.   v = malloc (sizeof (int) * n);      -----   ???
     // le n inteiros e guarda no vetor
4.   for (i = 0; i < n; i++) {           --------   ???
5.       scanf ("%d", &v[i]);           --------   ???
6.   }                          --------------------   ???
     // libera memoria
7.   free (v);                  --------------------   ???
8.   return 0;                  --------------------   ???
}                                              Total:   ???
```

- $O(n)$: linear?

```
#include <stdio.h>                                  consumo
#include <stdlib.h>                                 de tempo
int main () {
1.   int n, i, *v;                  -------------- O(1)
     // tamanho n do vetor
2.   scanf ("%d", &n);              -------------- O(1)
     // aloca memoria para vetor
3.   v = malloc (sizeof (int) * n);      -----   ???
     // le n inteiros e guarda no vetor
4.   for (i = 0; i < n; i++) {           --------  ???
5.       scanf ("%d", &v[i]);           --------  ???
6.   }                          -------------------- O(1)
     // libera memoria
7.   free (v);                  -------------------- O(1)
8.   return 0;                  -------------------- O(1)
}                                            Total:  ???
```

- $O(n)$: linear?

```
#include <stdio.h>                                  consumo
#include <stdlib.h>                                 de tempo
int main () {
1.   int n, i, *v;                  -------------- O(1)
     // tamanho n do vetor
2.   scanf ("%d", &n);              -------------- O(1)
     // aloca memoria para vetor
3.   v = malloc (sizeof (int) * n);      ----- O(n)
     // le n inteiros e guarda no vetor
4.   for (i = 0; i < n; i++) {          -------- O(n)
5.       scanf ("%d", &v[i]);          --------  ???
6.   }                          -------------------- O(1)
     // libera memoria
7.   free (v);                  -------------------- O(1)
8.   return 0;                  -------------------- O(1)
}                                          Total:  ???
```

- $O(n)$: linear?

```
#include <stdio.h>                                  consumo
#include <stdlib.h>                                 de tempo
int main () {
1.   int n, i, *v;                  -------------- O(1)
     // tamanho n do vetor
2.   scanf ("%d", &n);              -------------- O(1)
     // aloca memoria para vetor
3.   v = malloc (sizeof (int) * n);      ----- O(n)
     // le n inteiros e guarda no vetor
4.   for (i = 0; i < n; i++) {          -------- O(n)
5.       scanf ("%d", &v[i]);          --- O(n) * O(1)
6.   }                          -------------------- O(1)
     // libera memoria
7.   free (v);                  -------------------- O(1)
8.   return 0;                  -------------------- O(1)
}                                          Total:  ???
```

- $O(n)$: linear?

```
#include <stdio.h>                                  consumo
#include <stdlib.h>                                 de tempo
int main () {
1.   int n, i, *v;                  -------------- O(1)
     // tamanho n do vetor
2.   scanf ("%d", &n);              -------------- O(1)
     // aloca memoria para vetor
3.   v = malloc (sizeof (int) * n);      ----- O(n)
     // le n inteiros e guarda no vetor
4.   for (i = 0; i < n; i++) {          -------- O(n)
5.       scanf ("%d", &v[i]);          --- O(n) * O(1)
6.   }                                  -------------------- O(1)
     // libera memoria
7.   free (v);                          -------------------- O(1)
8.   return 0;                          -------------------- O(1)
}                  Total: 5 * O(1) + 2 * O(n) + O(1) * O(n)
```

- $O(n)$: linear?

```
#include <stdio.h>                                  consumo
#include <stdlib.h>                                 de tempo
int main () {
1.   int n, i, *v;                  -------------- O(1)
     // tamanho n do vetor
2.   scanf ("%d", &n);              -------------- O(1)
     // aloca memoria para vetor
3.   v = malloc (sizeof (int) * n);      ----- O(n)
     // le n inteiros e guarda no vetor
4.   for (i = 0; i < n; i++) {          -------- O(n)
5.       scanf ("%d", &v[i]);          --- O(n) * O(1)
6.   }                          -------------------- O(1)
     // libera memoria
7.   free (v);                  -------------------- O(1)
8.   return 0;                  -------------------- O(1)
}                                        Total: T(n) = O(n)
```

Notação O ("ó grande")

- O que é $\mathrm{T}(n)$?
  - É a função de consumo de tempo do algoritmo.

Notação O ("ó grande")

- Por que basta considerar o termo "dominante"?

Notação O ("ó grande")

- Definição
  - Sejam $\mathrm{T}(n)$ e $f(n)$ funções dos inteiros nos reais. Dizemos que $T(n)$ é $O(f(n))$ se existem constantes positivas c e n_0 tais que

$$\mathrm{T}(n) \leq cf(n)$$

para todo $n \geq n_0$.

Notação O ("ó grande")

- Definição