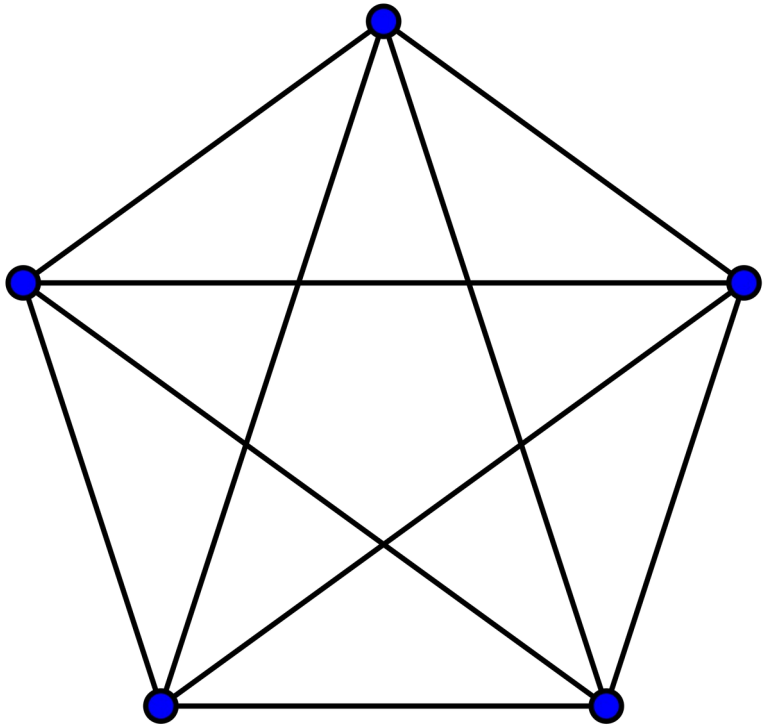
Grafos

Um Grafo

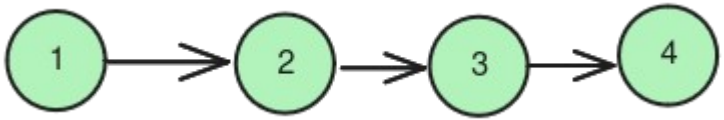
- Representa uma Relação
  - As mesmas que vimos em funções
- Representado visualmente por circulos conectados
- Matematicamente, um conjunto, e uma relação no conjunto



Definição Formal

- Um grafo é definido como uma tupla de dois conjuntos
  - G = (V, E)
- O conjunto V é um conjunto de índices ou objetos
- O conjunto E é um conjunto de tuplas de V^2

- Um exemplo:
  - G = (V, E)
  - V = {1,2,3,4}
  - E = {(1,2), (2,3), (3,4)}



Pra que servem grafos?

- Grafos modelam relações entre estruturas
  - Redes de amigos
  - Redes de computadores
  - Conexões de átomos em moléculas
  - Qualquer outra coisa que tiver relações

- Grafos são estruturas profundamente estudadas
  - Variedade imensa de algoritmos úteis
  - Materiais na internet podem ser um pouco complicados

O que vocês vão ver em ED?

- Em ED, vocês vão apenas ver como implementar grafos em C
  - Mas as propriedades de grafos podem ser úteis em TPs
- Vocês não serão cobrados classes específicas de grafos
  - Kn, Qn, Wn
- Árvores (próxima aula) são um tipo de grafo

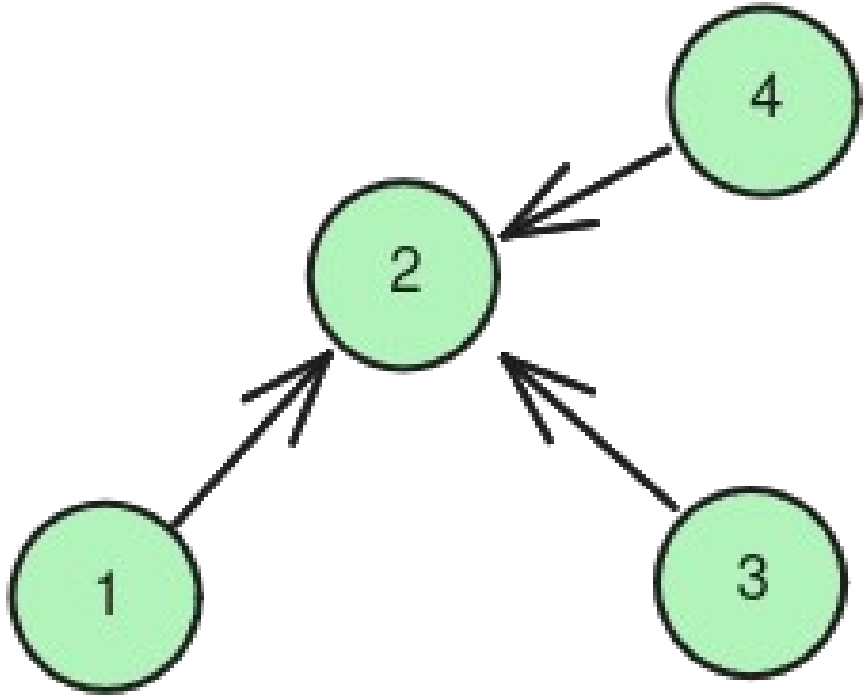
Propriedades de Grafos

- Existem vários tipos de grafos, algumas propriedades:
  - Grafos podem ser direcionados ou não-direcionados (simples)
  - Grafos podem permitir “auto-arestas”
  - Grafos podem ter ciclos ou não
  - Arestas podem ter valores associados

- Todas essas propriedades podem impactar sua ED e algoritmo
  - Algumas EDs precisam ser adaptadas
  - Outras são menos eficientes para alguns tipos

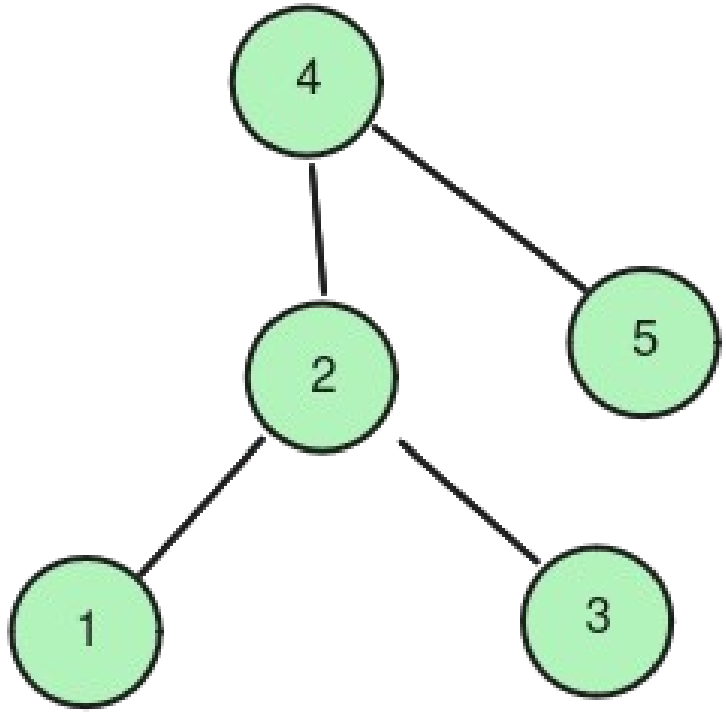
Grafo direcionado

- É um grafo onde relações não são simétricas
- Você pode ter um amigo que não te vê como amigo
- Exemplo: Seguidor no instagram



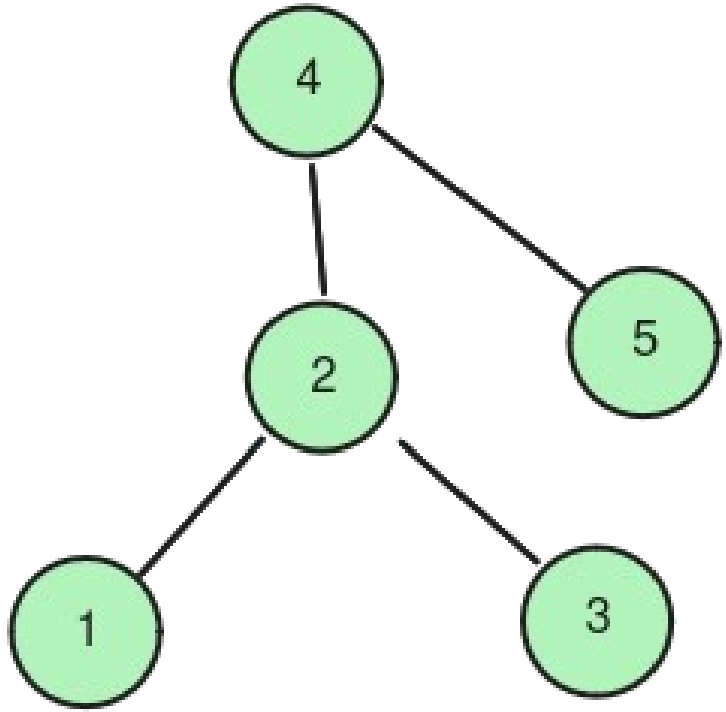
Grafo Não-Direcionado (ou Simples)

- Toda relação é simétrica
- Mais fácil para vários algoritmos
- Ordem não importa nas tuplas do conjunto E
- Exemplo: amizade no Facebook



Propriedade: Distância

- Número mínimo de arestas entre dois nós
- Não é trivial, precisa ser computado
- No grafo ao lado:
  - Distância(1, 5) = 3



Como Representar Grafos?

- Na matemática, desenhos são apenas para humanos
  - O computador não sabe desenhar

- Existem várias formas equivalentes de representar grafos
  - Mas algumas são melhores para casos específicos
- Existem 3 formas principais de representar um grafo
  - Conjunto de Arestas
  - Matriz de Adjacência
  - Matriz de Incidência

Conjunto de Arestas

- Remete à forma G = (V, E)
  - V será um conjunto com todos os índices de nós do grafo
  - E será um conjunto com todos as arestas que existem
- Exemplo: E = {(1,2), (3,2)}

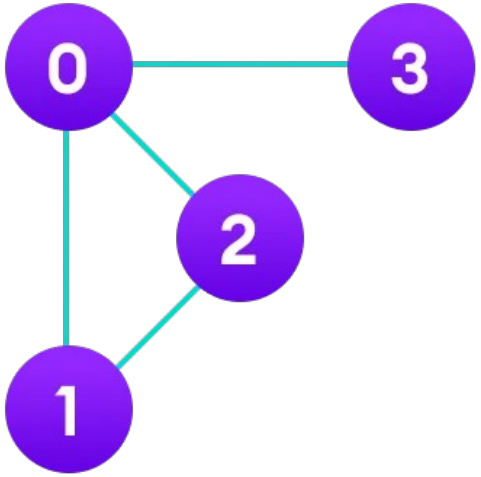
- Mas conjuntos não são fáceis de implementar em computadores
  - E podemos estrutrar esses dados para melhor acesso
  - Daí que vêm a Lista de Adjacência

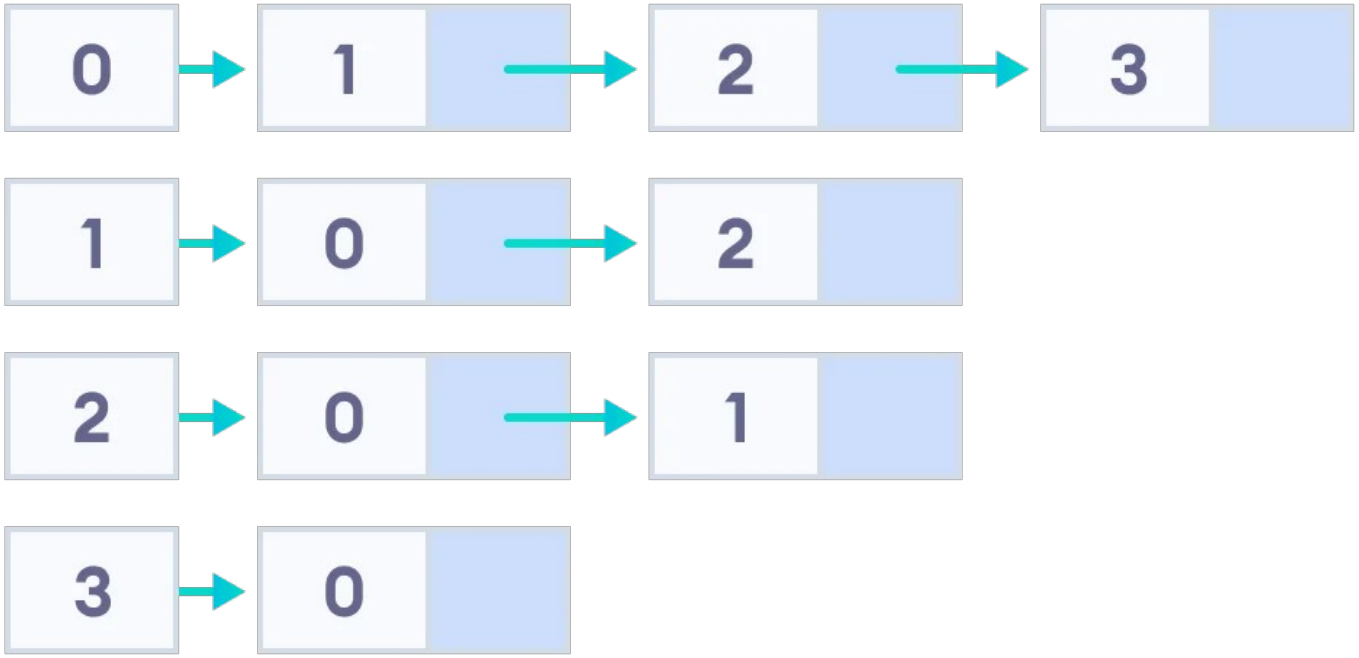
Lista de Adjacência

- Suponha que você tem uma lista (array) de índices V, de 0 até n
- Crie n listas (ou array) vazias para cada nó
  - Para cada vizinho daquele nó i, adicione o seu índice j na lista de I

- No fim, todos os pares estarão representados
- Isso pode ser implementado com arrays
  - Mas caso você não saiba o número de vizinhos, pode usar lista/vector
  - Essa é a forma mais usada para armazenar grafos

Exemplo: Lista de Adjacência



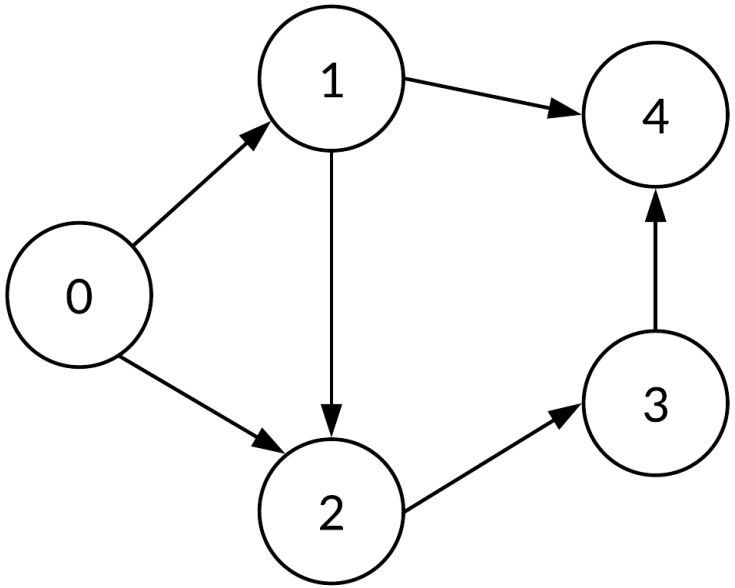


Matriz de Adjacência

- Dado que você tem N nós, crie uma matriz NxN só com zeros
- Chamaremos a matriz de A

- Para cada aresta com um par de vértices (i, j)
  - Coloque 1 em A[i, j]
  - Caso o grafo seja não-direcionado, coloque também 1 em A[j, I]
- Se quiser checar se dois nós são vizinhos, cheque A[i,j]
- Muito ineficiente em espaço

Exemplo: Matriz de Adjacência



Adjacency Matrix

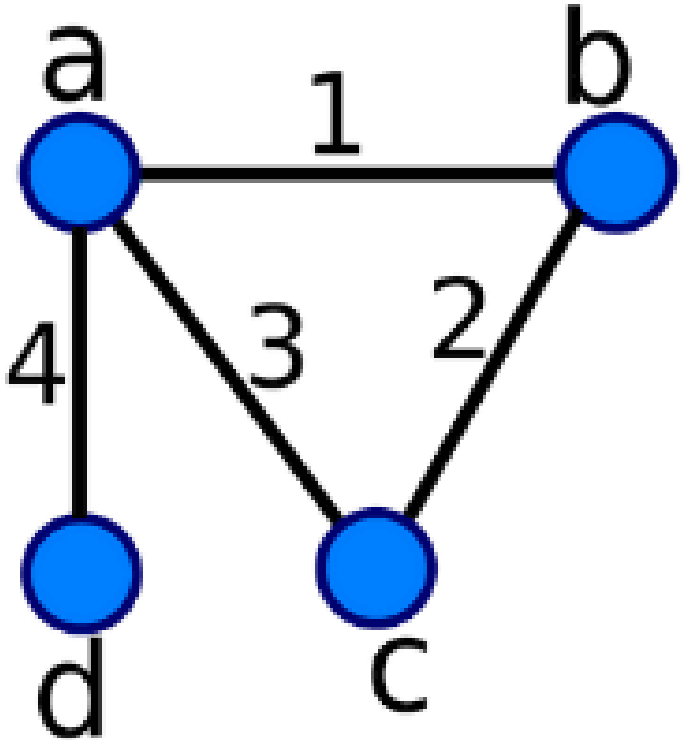
	0	1	2	3	4
0	0	1	1	0	0
1	0	0	1	0	1
2	0	0	0	1	0
3	0	0	0	0	1
4	0	0	0	0	0

Matriz de Incidência

- Dado que você tem N vértices e M arestas
  - Crie uma matriz A, NxM, com apenas zeros
- Para cada aresta (i, j) em E, pegue o índice e dela
  - Coloque 1 em A[i, e] e A[j, e]
  - Caso seja direcionada, um nó terá +1, o outro -1
- Para consultar os nós de uma aresta **e**, percorra a coluna **e**

- Mais compacto que a anterior
  - Mas mais confusa

Exemplo: Matriz de Incidência



	1	2	3	4
a	1	0	1	1
b	1	1	0	0
c	0	1	1	0
d	0	0	0	1

É isto

Mas será que rola um exercício?

Exercício Tentativo: Vizinhança em Matriz de Adj.

- O objetivo desse exercício é já tentar implementar uma ED
  - Nesse caso a estrutura será a Matriz de Adjacência
- O seu programa deve receber uma lista pares de índices e criar uma matriz de adjacência desses índices
- Em seguida, ele deve listar todos os vizinhos de cada nó
  - Sim, lista de adjacência seria mais eficiente
  - Mas é pra ser Matriz mesmo

Formato de Entrada

- Vou disponibilizar casos de teste
  - Mas você pode criar algumas entradas simples enquanto isso
- Sua entrada sempre será da forma:

  n_nós n_arestas

  nó nó

  nó nó

  ...
- n_arestas vezes

Exemplo: Entrada

- Para o grafo ao lado, uma entrada possível seria:

  5 4

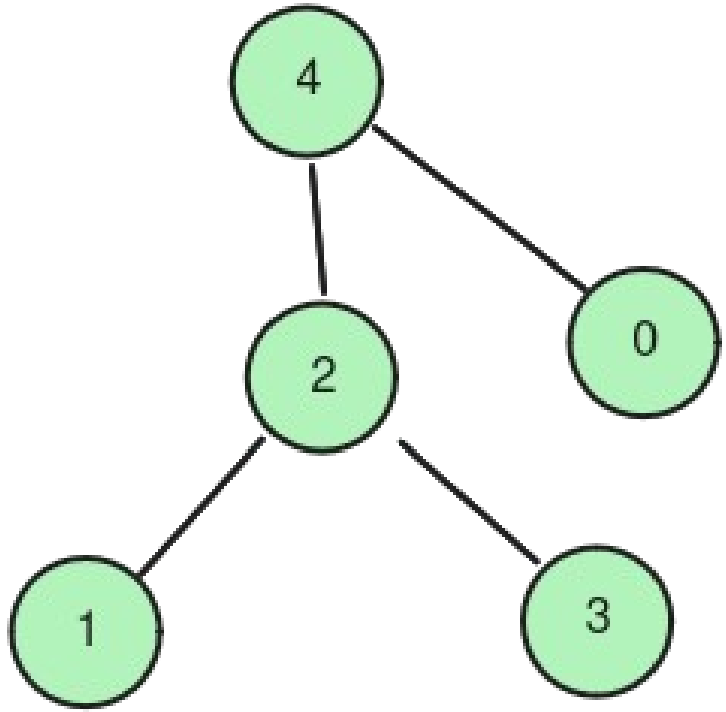
  1 2

  3 2

  2 4

  0 4

- Note que o grafo é sempre simples



Exemplo: Saída

- Para o grafo ao lado, uma saída possível seria:

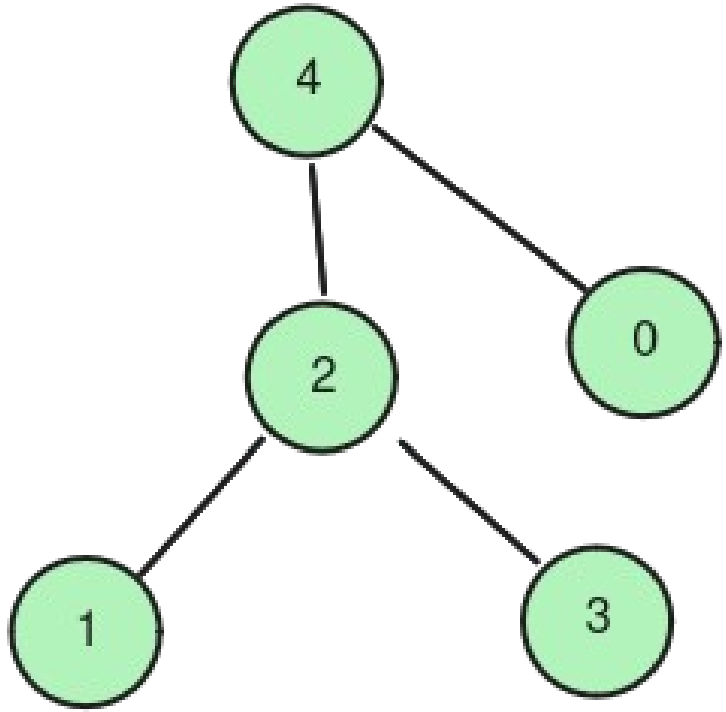
  0: 4

  1: 2

  2: 1 3 4

  3: 2

  4: 2 0



Como fazer isso?

- Primeiro você vai precisar criar a matriz A
  - Em C, uma matriz pode ser só um array de arrays
  - Essa sua matriz será n por n

- Depois de criar essa matriz, leia os pares das arestas um por um
  - Como serão números inteiros, use scanf
- Para cada par de vértices (i, j)
  - Coloque 1 nas posições A[i,j] e A[j,i]

Segunda parte: Imprimir

- Para imprimir a vizinhança de um nó
  - Itere pela linha (ou coluna) com índice daquele nó
  - Lembre-se da indexação por zero

- Para cada elemento de A_i (a linha do índice i)
  - Cheque se $A_i[j]$ é 1
    - Se for, você sabe que j é um vizinho: imprima
    - Se não for, continue o loop
- Faça isso para todas as linhas

Dúvidas?

- Entrada:

  5 4

  1 2

  3 2

  2 4

  0 4

- Saída

  0: 4

  1: 2

  2: 1 3 4

  3: 2

  4: 2 0