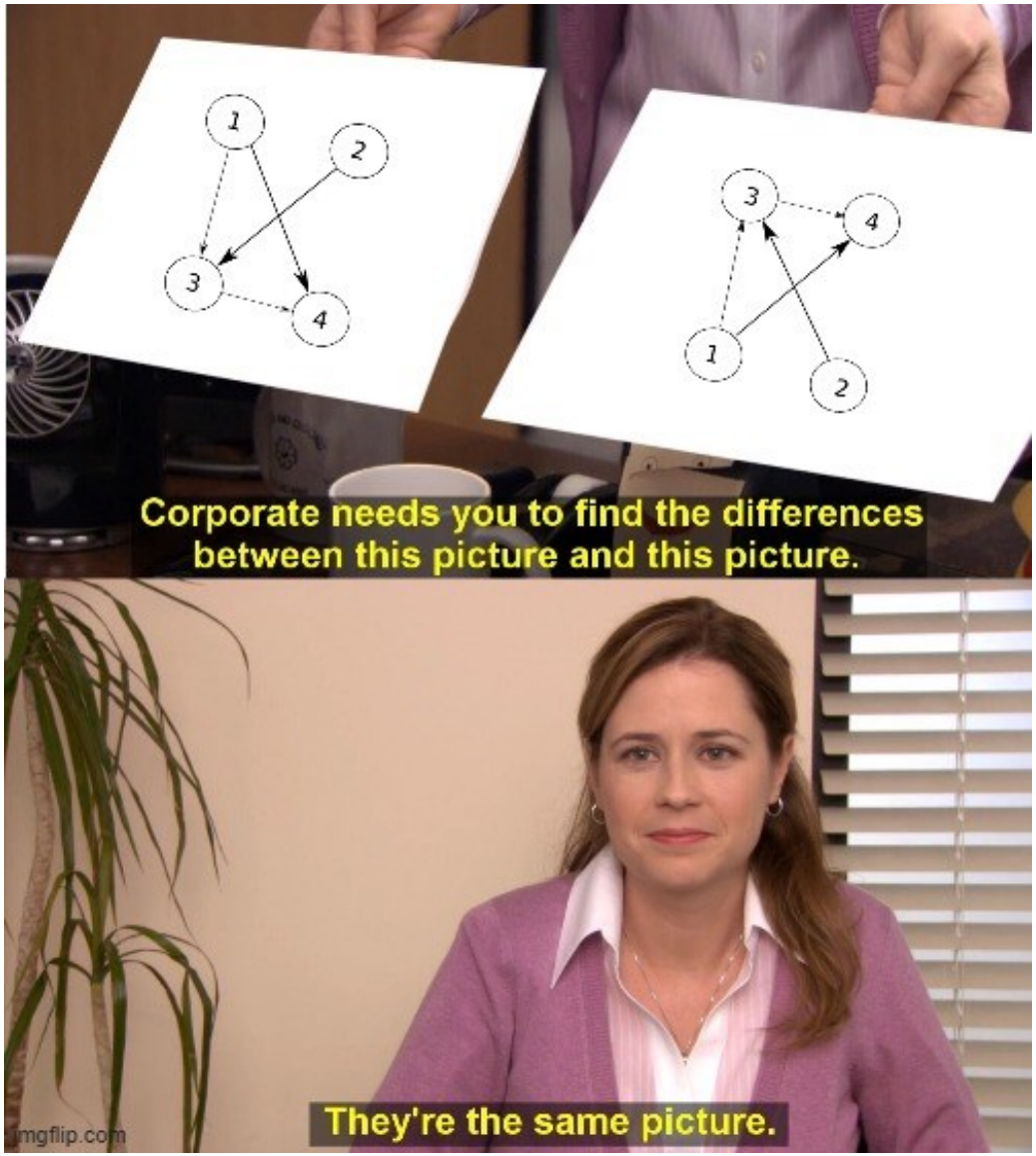
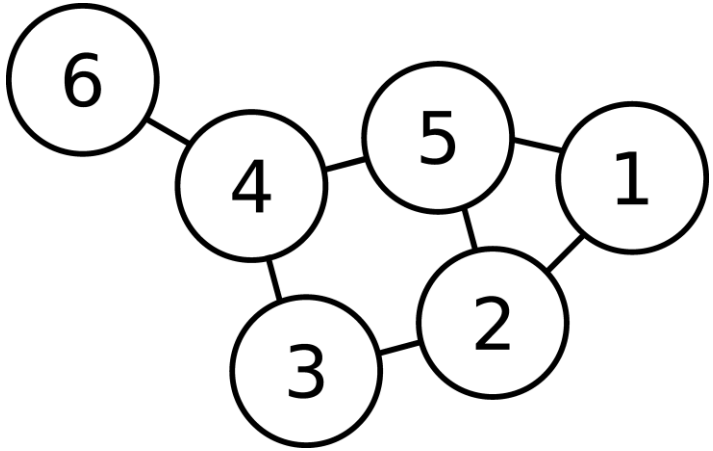
Grafos – aula de lab



Grafos – relembrando ...

- **Definição (Grafo):** Um grafo **G = G(V, E)** é uma estrutura matemática constituída pelos seguintes conjuntos:
  1. **V:** denominado conjunto de vértices **i** (ou nós do grafo).
  2. **E:** denominado conjunto de arestas **(i,j)** (ou arcos), que conectam dois vértices **i** e **j** do conjunto **V**.

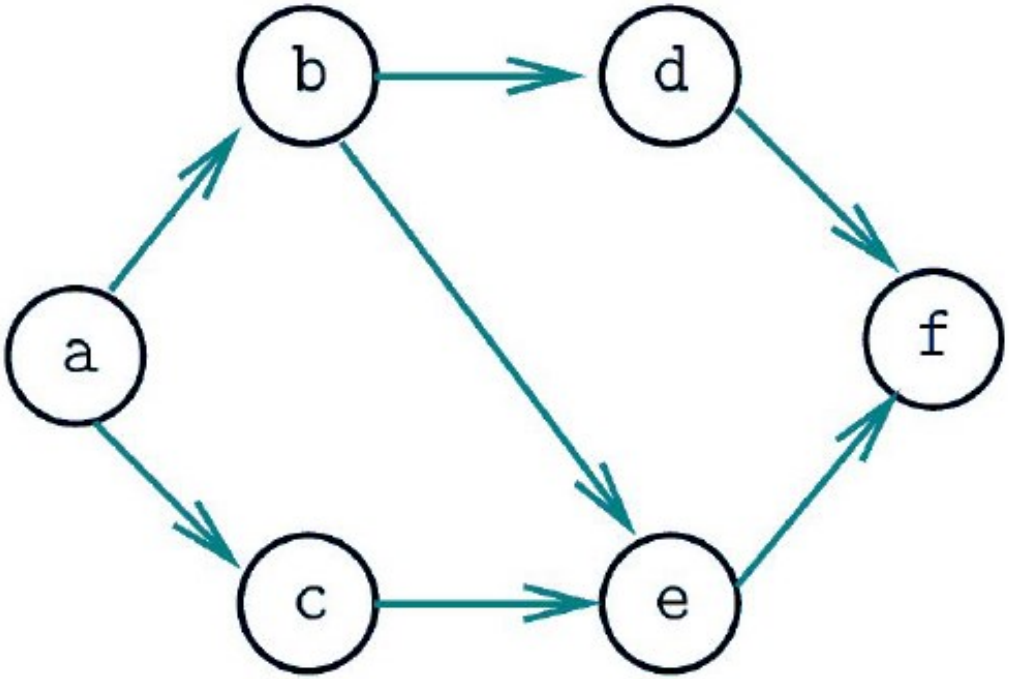


Exemplo:

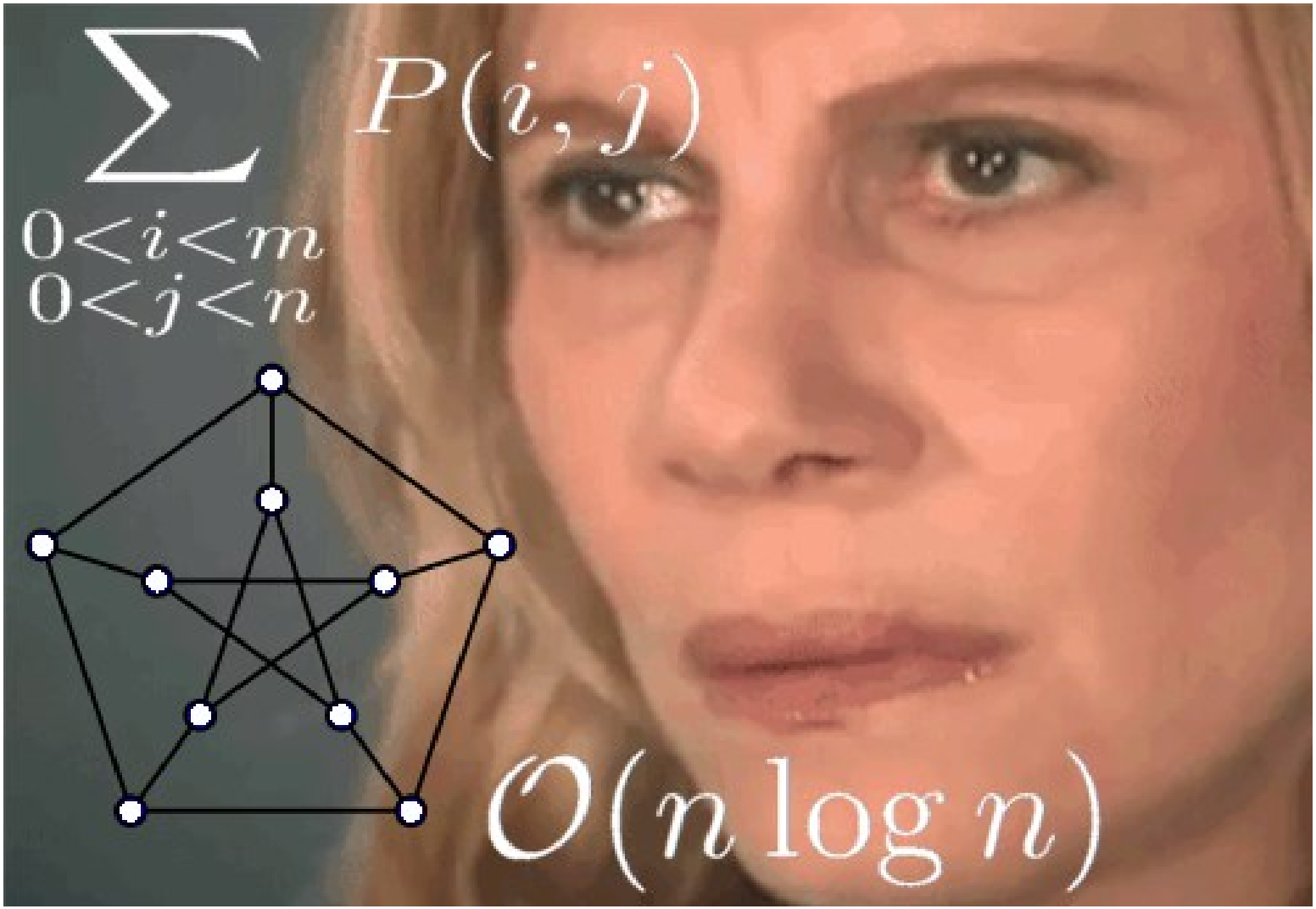
- V = {1,2,3,4,5,6}
- A = {(1,2), (1,5), (2,3), (2,5), (3,4), (5,4), (4,6)}

Grafos – relembrando ...

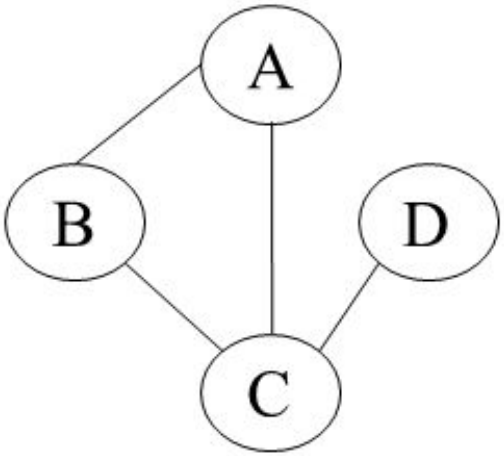
- **Dígrafo:** É um **grafo direcionado**, cujas arestas possuem direções específicas (são na verdade flechas).
  - O primeiro vértice do par ordenado é a **ponta inicial do arco**, e o segundo, **a ponta final**.

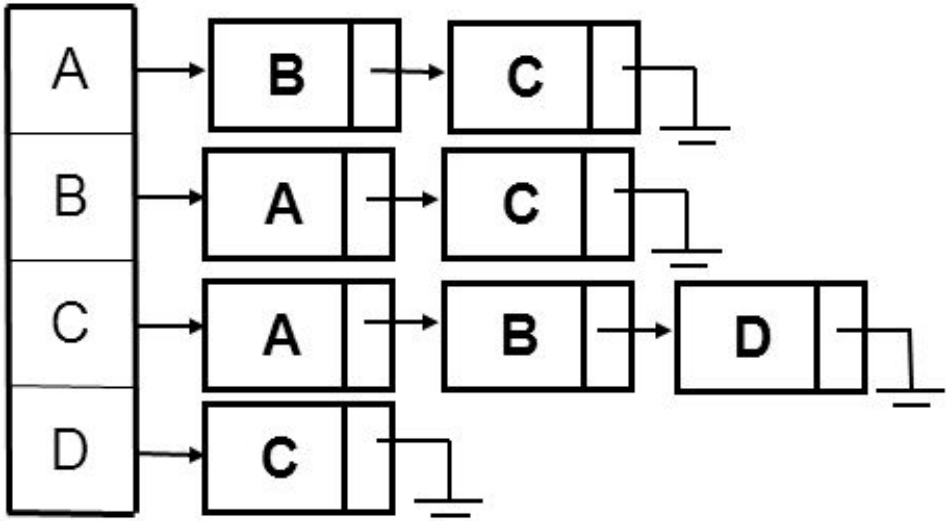


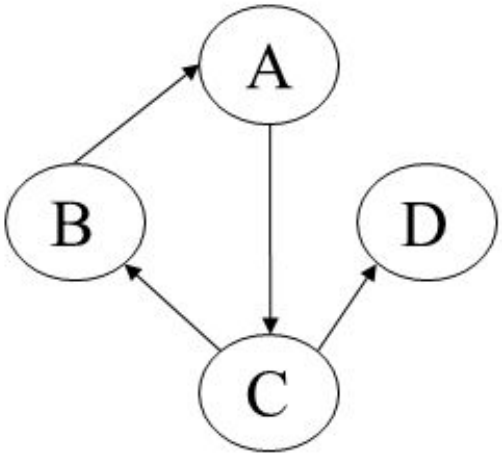
TAD Grafos – lista de adjacência

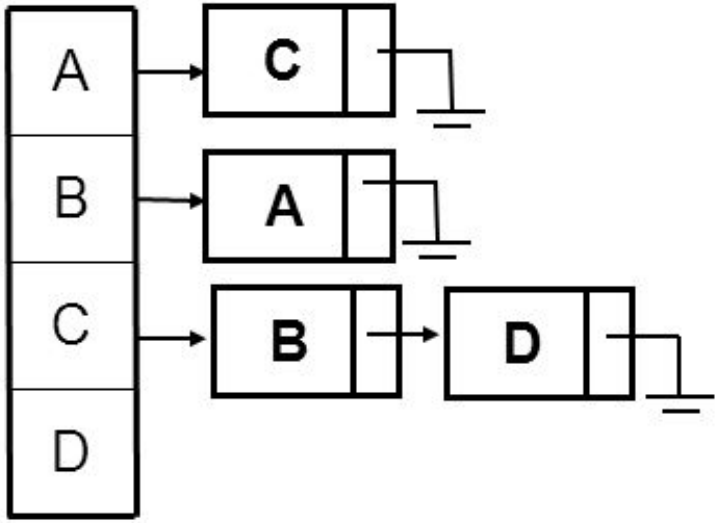


Grafos – TAD lista de adjacência







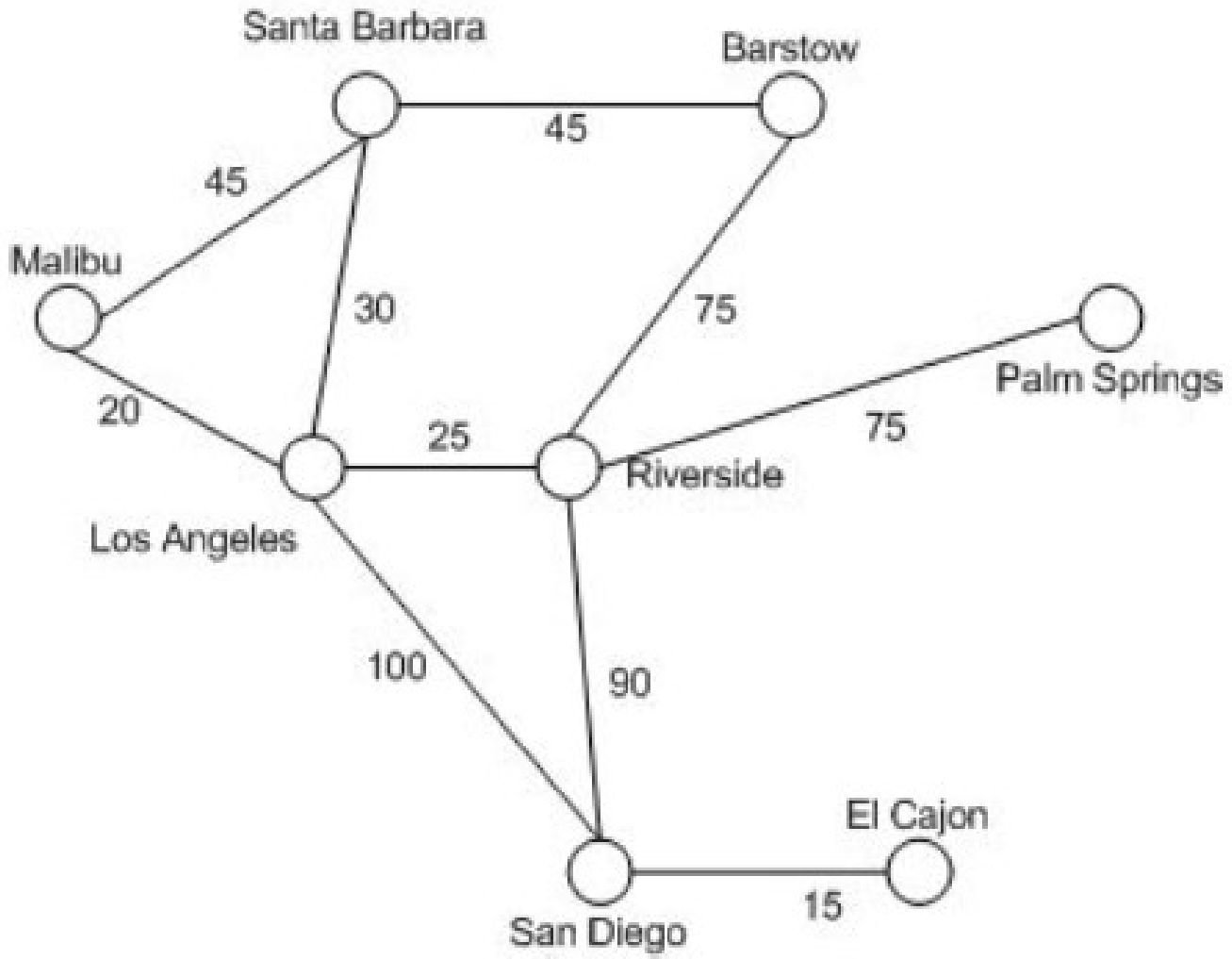


```c
//EDs para nó e grafo
//---------------------------------
struct no {
    int id;
    int val;
    struct no *prox;
};

typedef struct no *No;

struct grafo {
    int id;
    int nNo; //nro de nós
    No vertices; //array de vértices
};

typedef struct grafo *Grafo;
```

```
No criaNo(int id, int val){
    No n = (No) malloc(sizeof(struct no));
    n->id = id;
    n->prox = NULL;
    n->val = val;
    return n;
}
 void addNo(No n, int id, int val){
     No novo = criaNo(id, val);
     if(n == NULL){
         return;
     }
     while(n->prox != NULL){
         n = n->prox;
     }
     n->prox = novo;
 }
```

- **Representação** do grafo a partir de um **padrão pré-definido** em um arquivo de texto.

```
8
0 1 45
0 2 20
1 2 30
1 3 45
2 4 25
2 5 100
3 4 75
4 5 90
4 6 75
5 7 15
```



Leitura de arquivo p/ lista adjacência

```
//Efetuar a leitura do grafo via arquivo
void readGraph(Grafo G, const char *filename){
    FILE *fp;
    int bsize = 20;
    int i, o, d, v;

    char buffer[bsize]; //cadeia de caracteres de 19 elem. no máximo
    fp = fopen(filename,"r");

    //Primeira linha do arquivo indica o número de vértices
    fgets(buffer, bsize, fp);
    sscanf(buffer, "%d", &G->nNo); //Salva o numero de vertices
    G->vertices = (No) malloc(G->nNo * sizeof(struct no)); // Cria vet. nós
```

Leitura de arquivo p/ lista adjacência

```
//Primeiro elemento de cada lista identifica o nó que a lista representa
for(i = 0; i < G->nNo; i++){
    (G->vertices + i)->id = i;   //Notação de ponteiro para vetor
    (G->vertices + i)->val = -1; //Valor default (nao usado nesta aplicação)
    (G->vertices + i)->prox = NULL;
}
//Percorre o arquivo
while(!feof(fp)) {
    fgets(buffer, bsize, fp);
    sscanf(buffer, "%d %d %d", &o, &d, &val);
    //Adiciona um novo nó com id = d e valor = val na lista de adj. de 'o'
    addNo((G->vertices + o), d, val);
}

fclose(fp);
return;
```

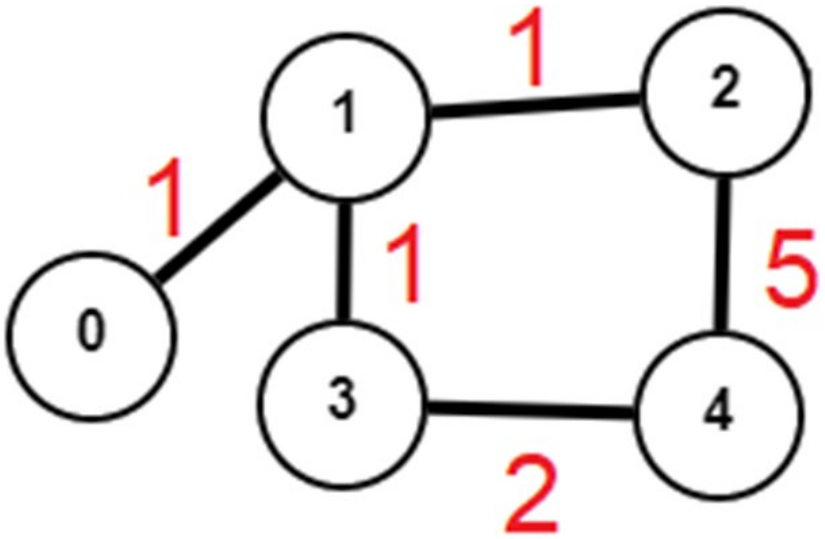

Dijkstra's algorithm
CS students
Implementation
vk/ld440

Grafos – definições relacionadas

- **Caminho (patch):** Um caminho de um vértice **v** a um vértice **u** é uma **sequência de vértices** em que, para cada vértice, do primeiro ao penúltimo, há uma aresta conectando esse vértice ao próximo da sequência.

Exemplo:

- São caminhos: (0, 1, 2), (1,2,4,3), (4,3,1,0), (1,2,4,3,1)

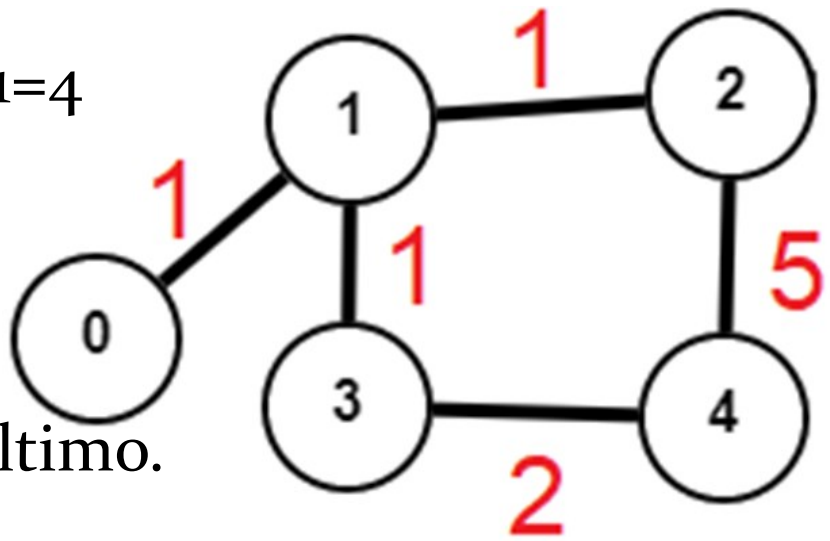


Grafos – definições relacionadas

- **Comprimento:** Número de arestas de um caminho, ou, no caso de um grafo com pesos nas arestas, é a soma de todos os pesos das arestas do caminho.

Exemplos:

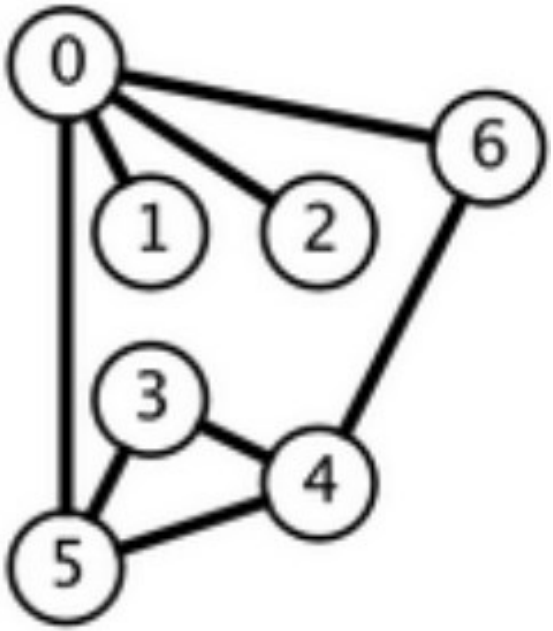
- Comprimento de (0, 1, 2): 1+1=2
- Comprimento de (1,2,4,3): 1+5+2=8
- Comprimento de (4,3,1,0): 2+1+1=4
- Comprimento de (1,2,4,3,1): 9



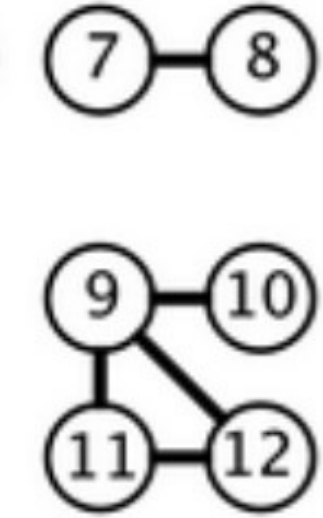
Ciclo: Caminho fechado, em que o primeiro vértice é igual ao último.

Grafos – definições relacionadas

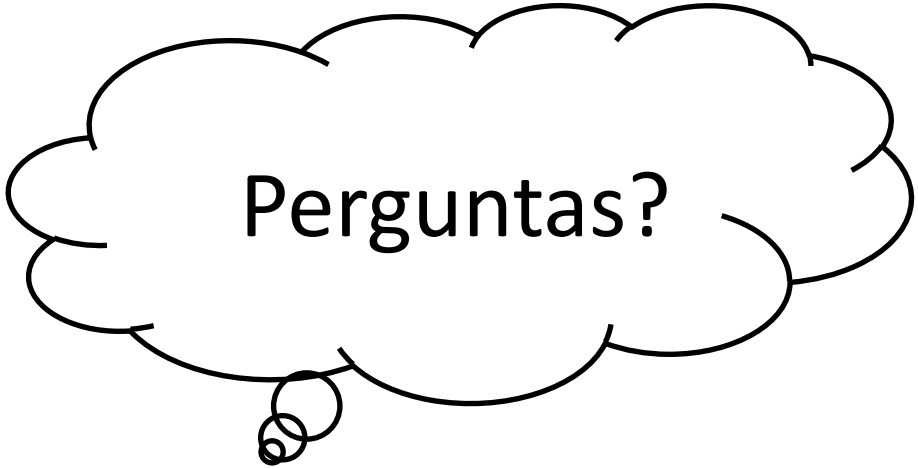
- **Grafo Conexo:** Um grafo G é dito conexo se cada par de vértices de G existir pelo menos um caminho. Caso contrário, o grafo é dito desconexo.



- Grafo conexo.

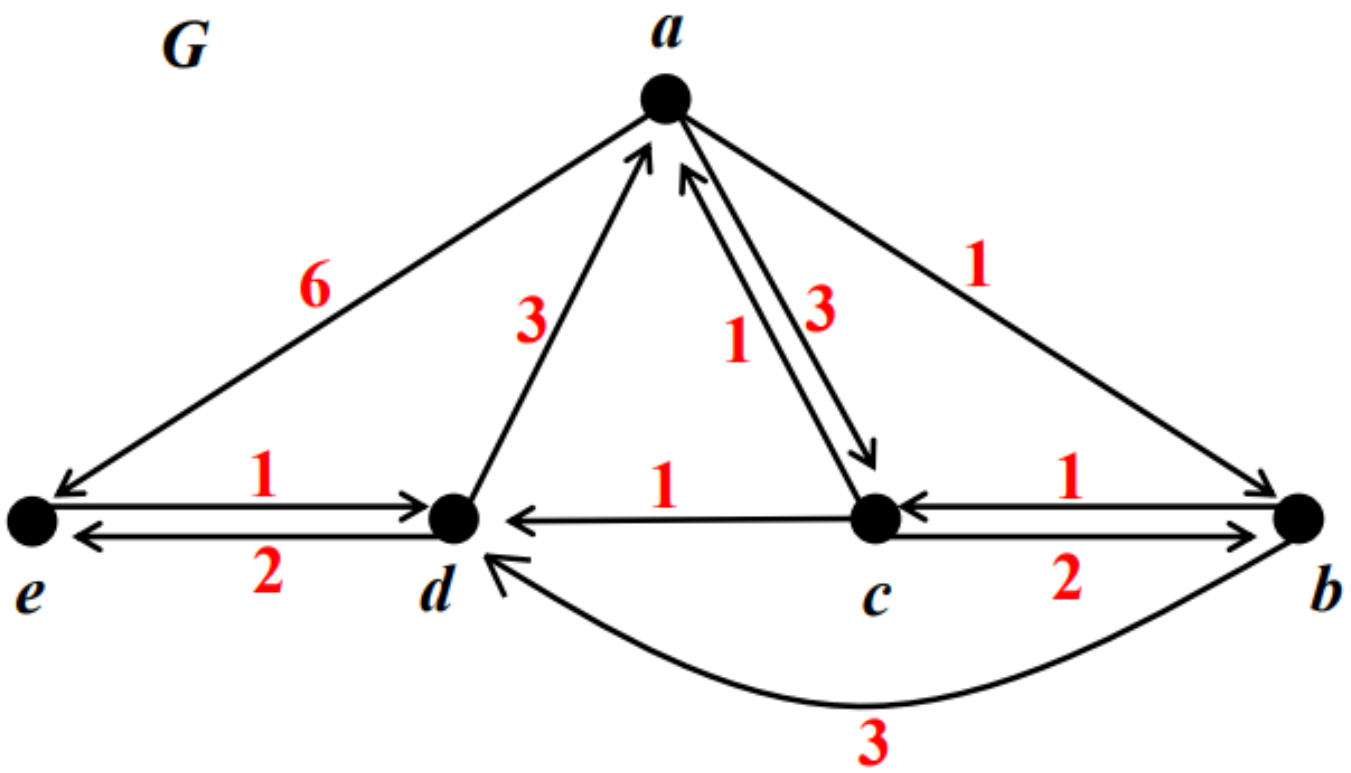


- Grafo desconexo.


Perguntas?

Grafos – caminho mínimo

- **Problema (caminho mínimo):** Dado um grafo G conexo com pesos positivos nas arestas, determinar os caminhos mínimos de um dado vértice **v** a todos os demais vértices.



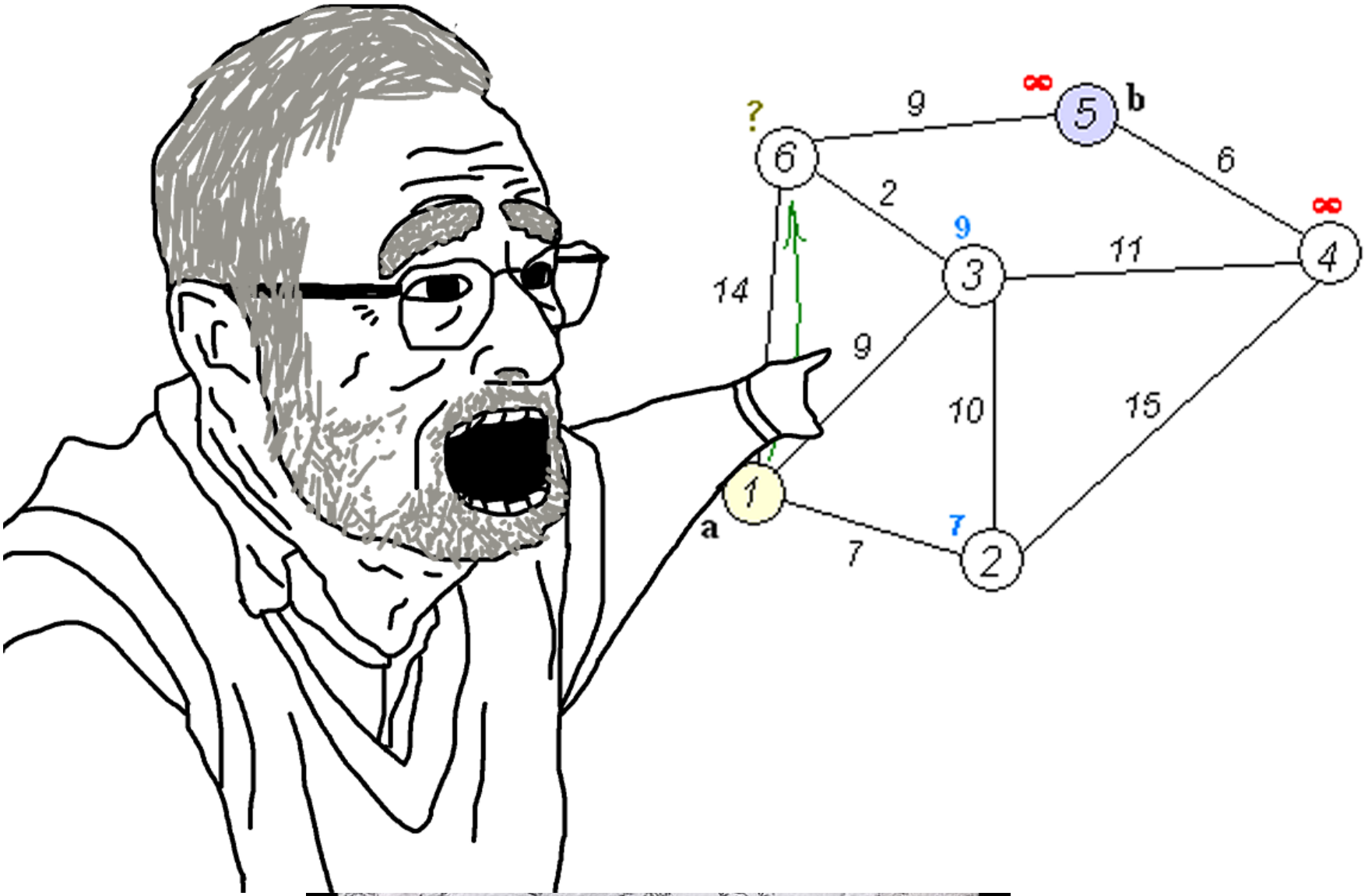
Grafos – caminho mínimo

- **Problema (caminho mínimo):** Uma possível solução é usar o algoritmo de Dijkstra.
- **Algoritmo de Dijkstra**: calcula o caminho mais curto, em termos de peso total das arestas, entre um dado nó $\boldsymbol{u}$ e todos os demais nós $\boldsymbol{v} \in \boldsymbol{V}$ do grafo.
- **Ideia chave do algoritmo**: usar um processo iterativo que:
  - Iteração #1: determina o nó mais próximo de $\boldsymbol{u}$.
  - Iteração #2: determina o segundo nó mais próximo de $\boldsymbol{u}$.
  - E assim sucessivamente, até que o último nó seja atingido.

Algoritmo de Dijkstra (modificado via fila)

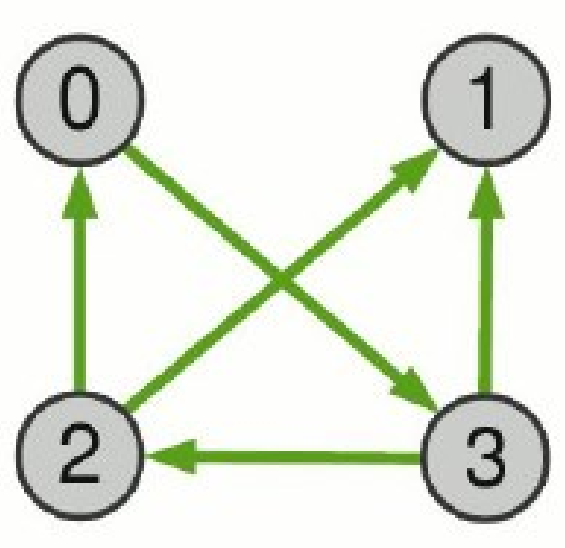
- Para cada nó $\boldsymbol{v} \in \boldsymbol{V}$, atribuímos um componente de índice $\boldsymbol{v}$ em um vetor de distâncias $\boldsymbol{d}$, de $\#V$ elementos, que representa a estimativa do caminho mais curto do nó inicial $\boldsymbol{u}$ até $\boldsymbol{v}$.
- Inicialmente, setamos $d[\boldsymbol{v}] = \infty$, exceto para o nó $\boldsymbol{u}$, que será $d[\boldsymbol{u}] = 0$.
- Marque todos os nós como “abertos” (vamos usar fila aqui!).
- Adicione na fila o primeiro elemento $\boldsymbol{u}$ e, para cada nó $\boldsymbol{v}$ aberto (ainda não acessado/incluído na fila) adjacente a $\boldsymbol{u}$, inserir na fila e atualizar $d[\boldsymbol{v}]$.
- Avançar com todos os nós abertos a partir da sua inclusão na fila.

When you don't know Dijkstra's algorithm..



1. Altere o código para que ele leia de um arquivo (formato estabelecido no ‘dígrafo.txt’), os pesos das arestas para uma matriz de adjacência com pesos.

2. Altere o algoritmo de Dijkstra para que ele calcule o caminho mínimo (considerando agora os pesos positivos das arestas) de um dado nó para todos os demais vértices



	0	1	2	3
0	0	0	0	**1**
1	0	0	0	0
2	**1**	**1**	0	0
3	0	**1**	**1**	0