Grafos

Estrutura de dados

O que são

Grafo G=(V, E)

V = conjunto de vértices, cada vértice representado por um número natural

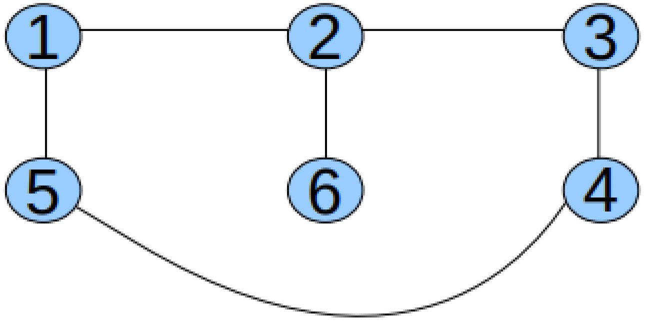
E = conjunto de arestas (pares não-ordenados, ou pares ordenados)

Exemplo V = {1, 2, 3, 4, 5, 6},

E = {(1,2), (1,5), (2,3), (2,6), (3,4), (5,4)}

representação matemática de grafos

Grafo não direcionado
(1,2) = (2,1)



Representação visual de grafos

O que são

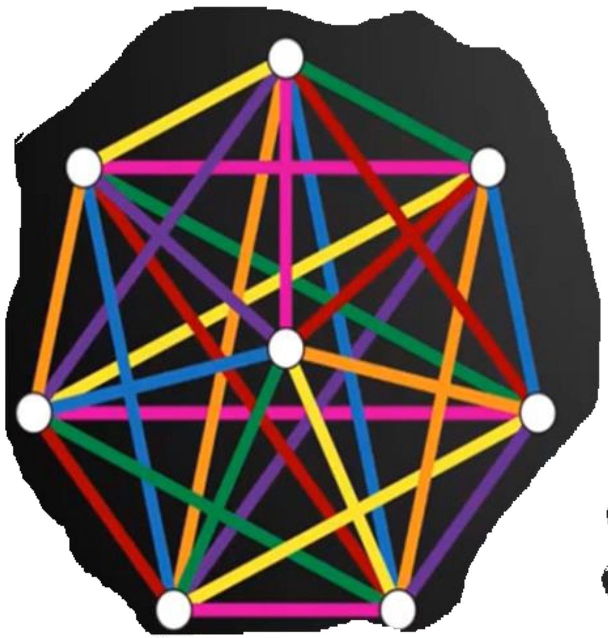
- Um grafo é um conjunto de vértices (ou nós) e arestas (ou arcos) , em que cada aresta conecta dois vértices.
- Árvore é um caso particular de grafos : cada nó só tem um pai, em grafos não tem limites. Um grafo conexo e sem ciclos (acíclico) é chamado de Árvore, com um vértice (nó) especial chamado nó.

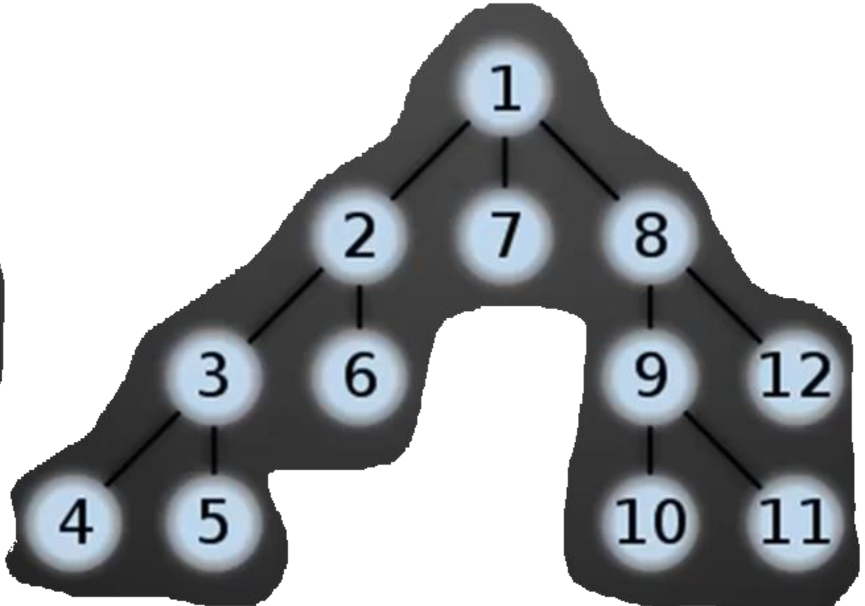
- Exemplos:

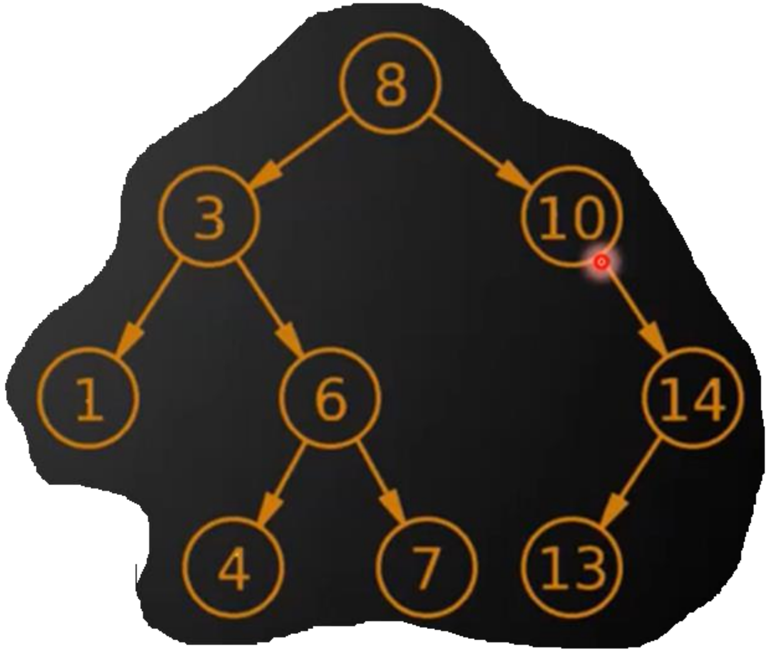
  redes neurais

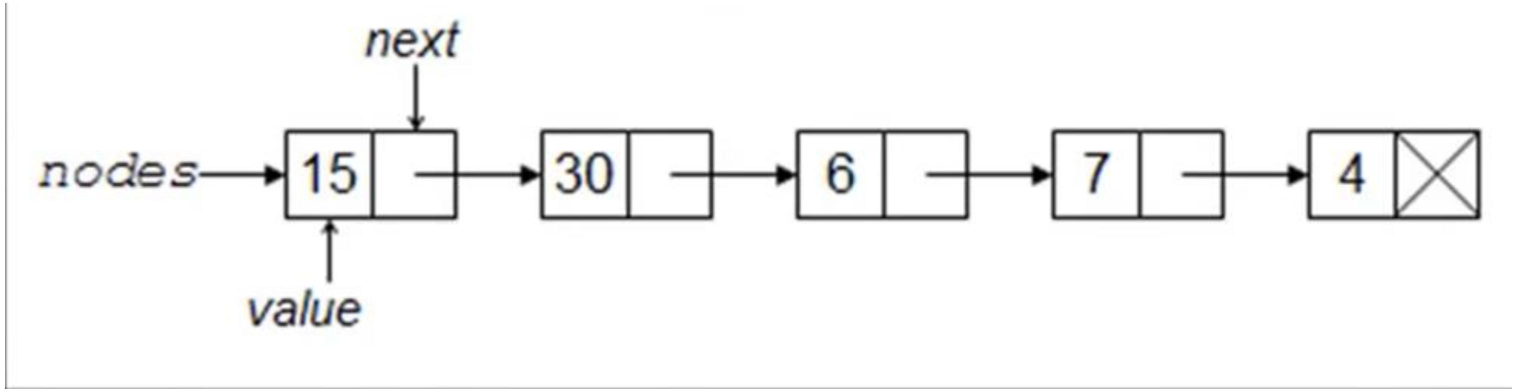
  mapa (cidades (caminhos entre cada cidade)

  rede social (pessoa e pessoas conectadas)

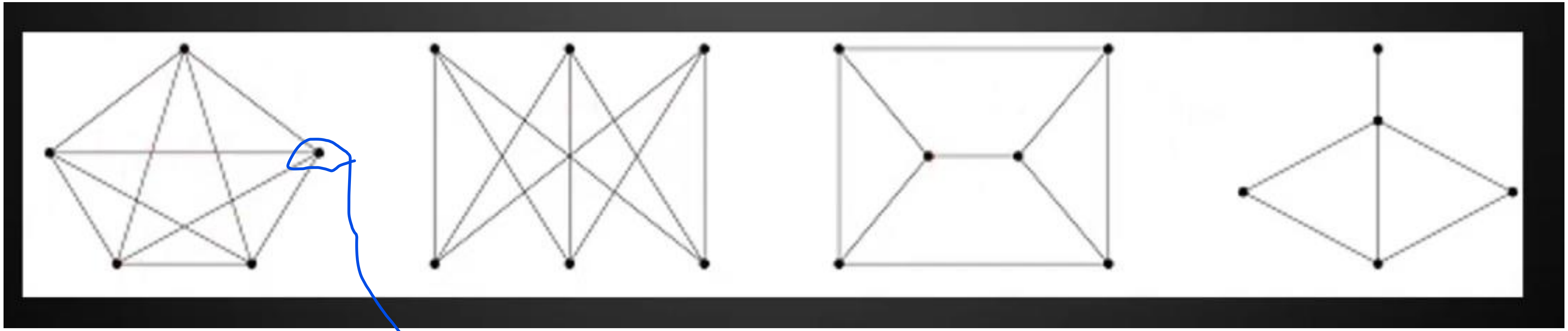



1
2
7
8
3
6
9
12
4
5
10
11


8
3
10
1
6
14
4
7
13


next
nodes
15
30
6
7
4
value

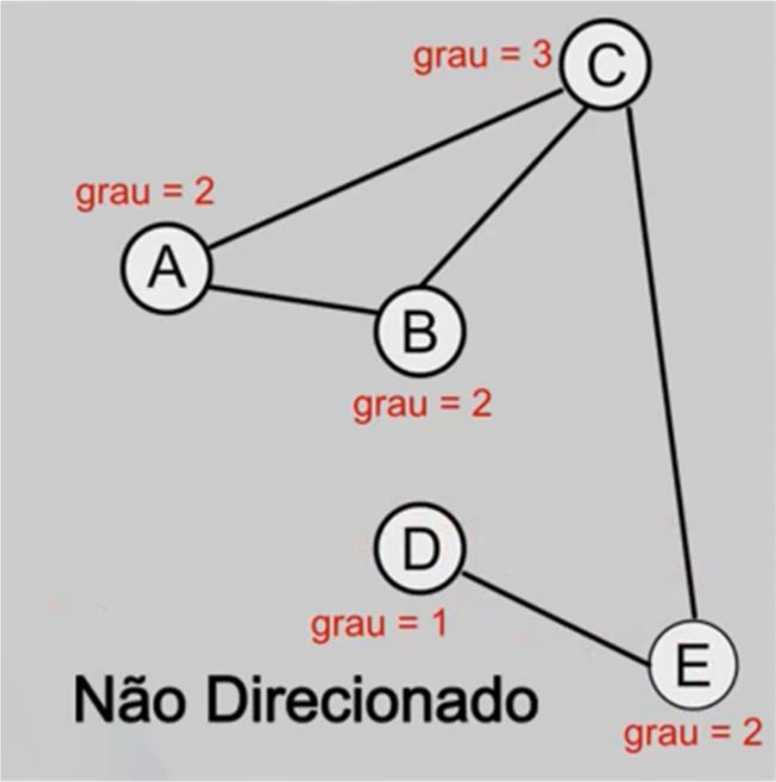
Grafo simples



Não tem laços e nem arestas múltiplas(ou paralelas)

A ordem de um grafo é a quantidade de vértices que ele possui.

Grafos – não direcionado

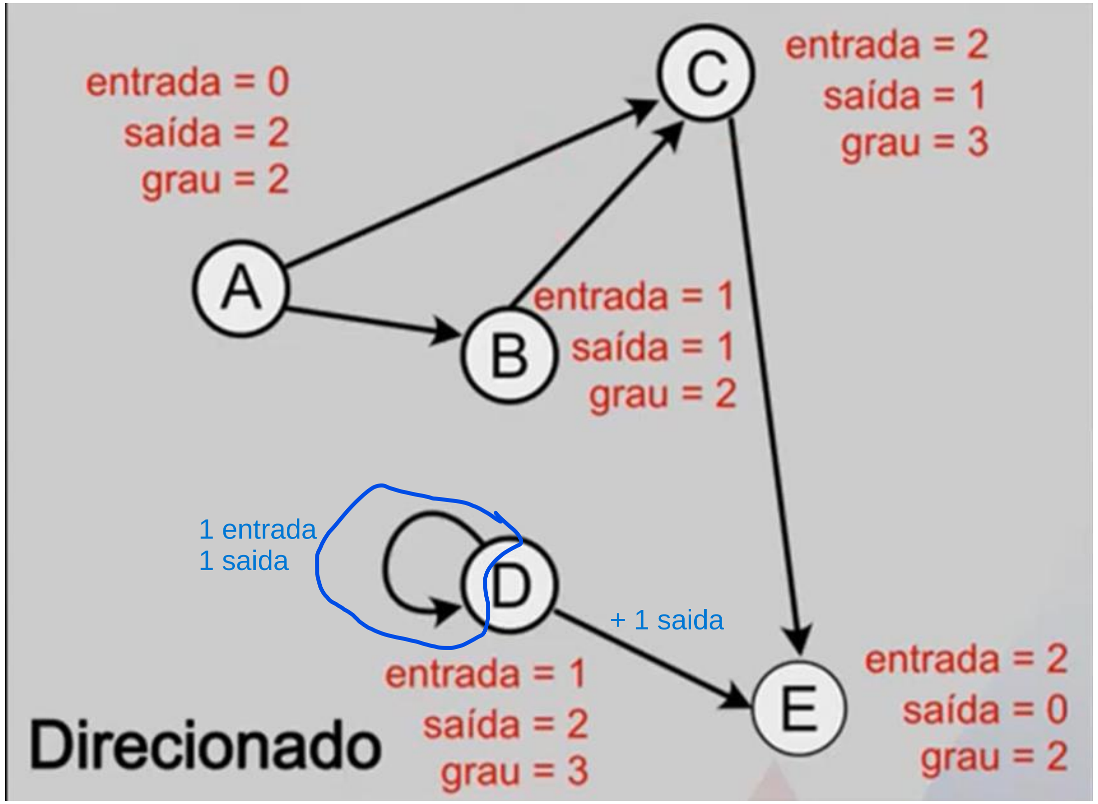


A quantidade de vezes que as arestas incidem (tocam) sobre o vértice v é chamado grau do vértice v.

Se tiver laço incide 2 vezes, toca duas vezes no vértice

Mais gasto computacional

Grafos - direcionado



Cada vértice terá um grau de entrada e um grau de saída do vértice.

Menos gasto computacional por causa que já está direcionado

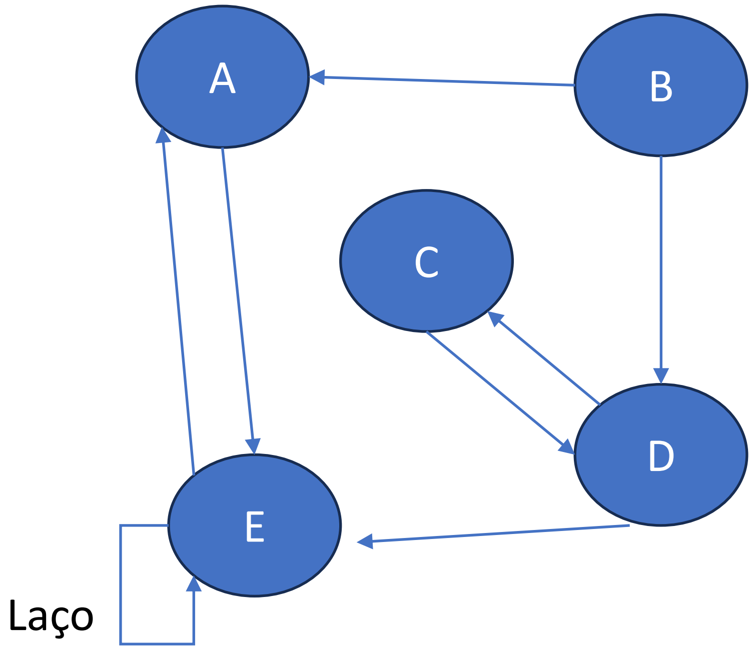
Grau é a quantidade de entrada mais saida

Grafos direcionados

O grau é representado por D

D(v)= qtd entrada(in) + qtd de saída (out)
D(A) = 3

Vértices adjacentes é pq estão conectados



V = {A,B,C,D,E}

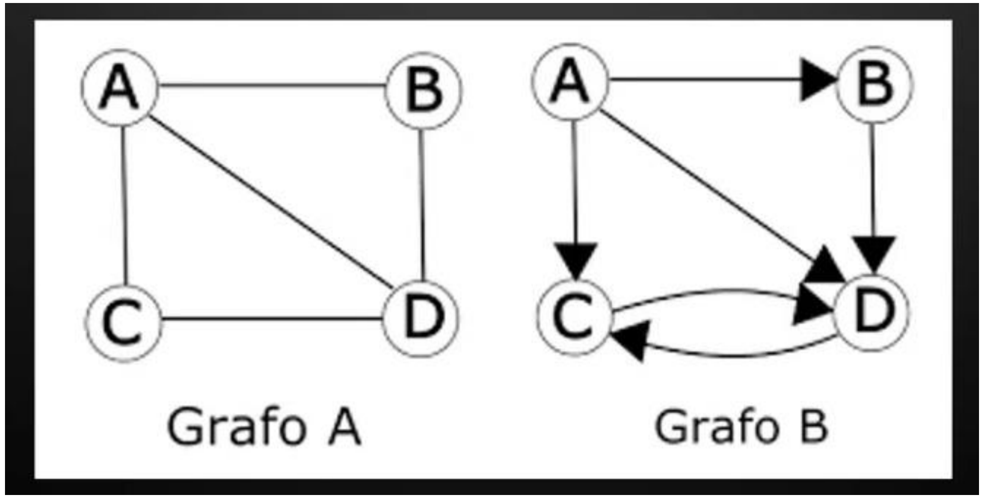
E = {(A,E),(B,A),(B,D),(D,C),(D,E)(C,D)(E,A),(E,E)}

Quando existe uma aresta ligando dois vértices dizemos que os vértices são adjacentes, e que a aresta é incidente aos vértices

Grafo não simples tem laço

Passeio

- Um passeio (ou percurso) é uma sequência de arestas do tipo (v0,v1),(v1,v2),(v2,v3), ...(vs-1,vs)
- **V_0 é o início** do passeio e **V_s é o fim**
- s é o comprimento do passeio (quantidade de arestas que passei)



Exemplo :
Ir de C para B : (C,D),(D,B) -> passeio

Pelo direcionado daria?

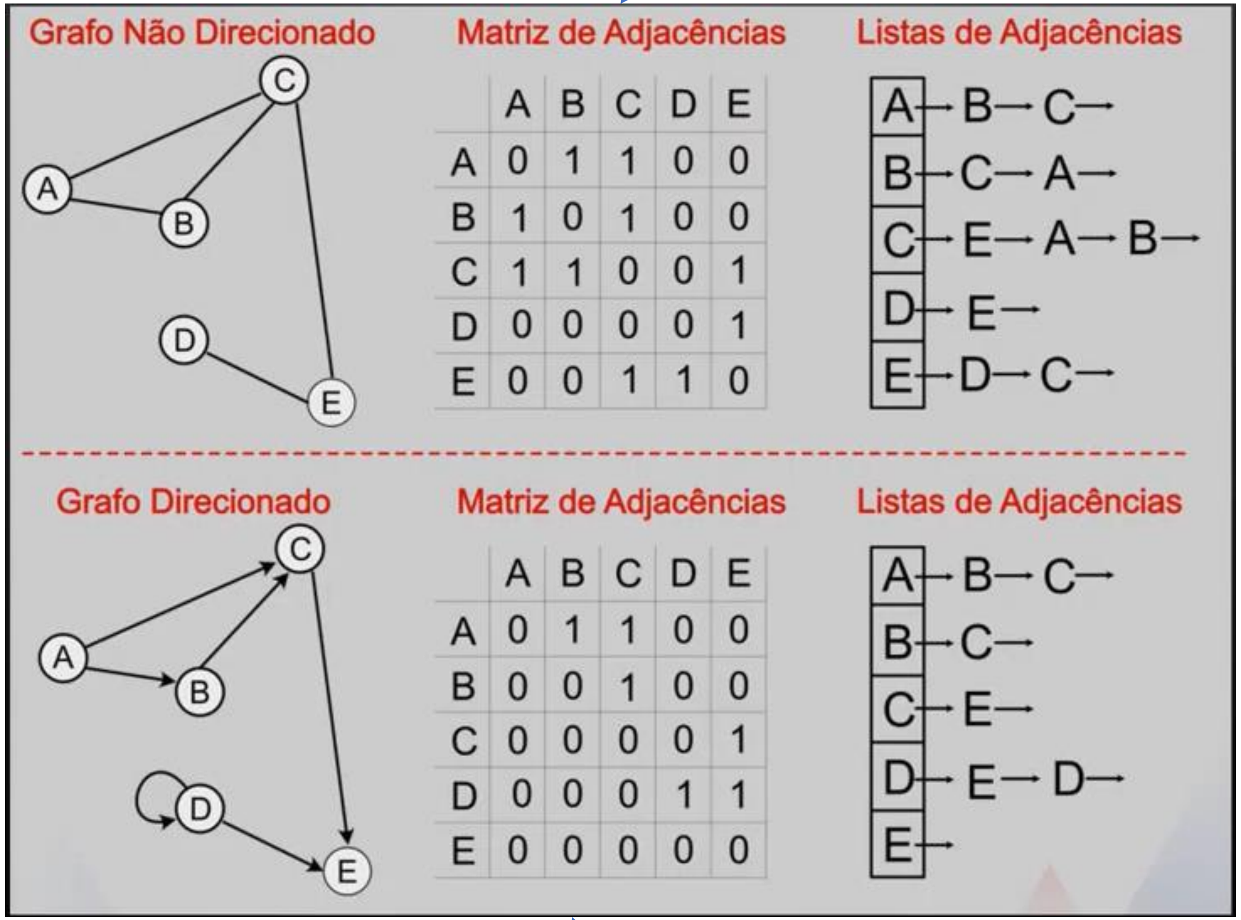
Obs : caminho não repete vértice

Na computação

Estruturas de dados fundamentais:

- vetor
- matriz
- lista
- tabela hash (dicionário)

Todos os elementos da matriz são 1, não tem arestas múltiplas.
Grafo simples (só tem 1)
Matriz simétrica :
Diagonal principal – corte no meio (espelho)
Grau do vértice : soma a linha da matriz



	A	B	C	D	E
A	0	1	1	0	0
B	1	0	1	0	0
C	1	1	0	0	1
D	0	0	0	0	1
E	0	0	1	1	0

	A	B	C	D	E
A	0	1	1	0	0
B	0	0	1	0	0
C	0	0	0	0	1
D	0	0	0	1	1
E	0	0	0	0	0

Lista encadeada

Não é matriz simétrica
Grau do vértice : entrada (coluna) e saída (linha) e soma os dois

- Como representar utilizando matrizes?

Idéia: associar vértices às linhas e colunas da matriz elemento da matriz indica se há aresta

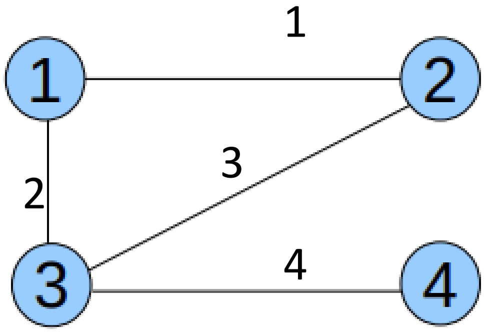
Matriz de adjacência Matriz n x n (n é número de vértices)

a_{ij} = 1 , se existe aresta entre vértices *i* e *j*

a_{ij} = 0 , caso contrário.

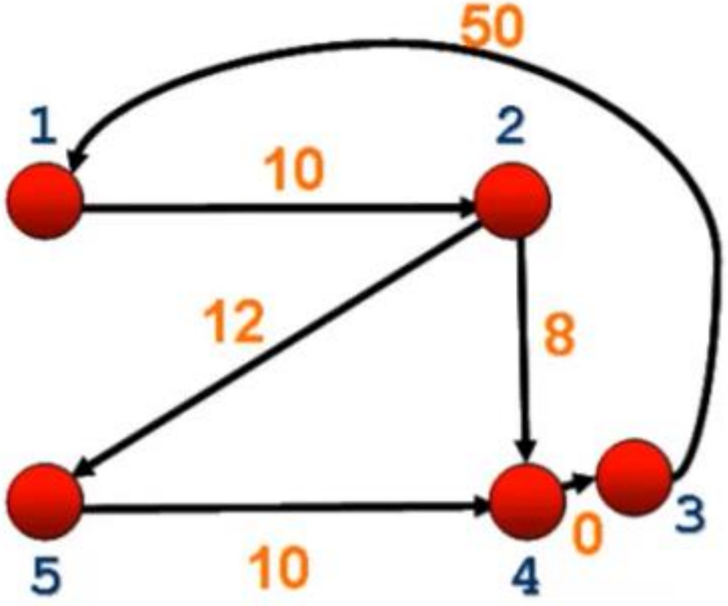
Matriz de distâncias – com pesos
Grafos valorados

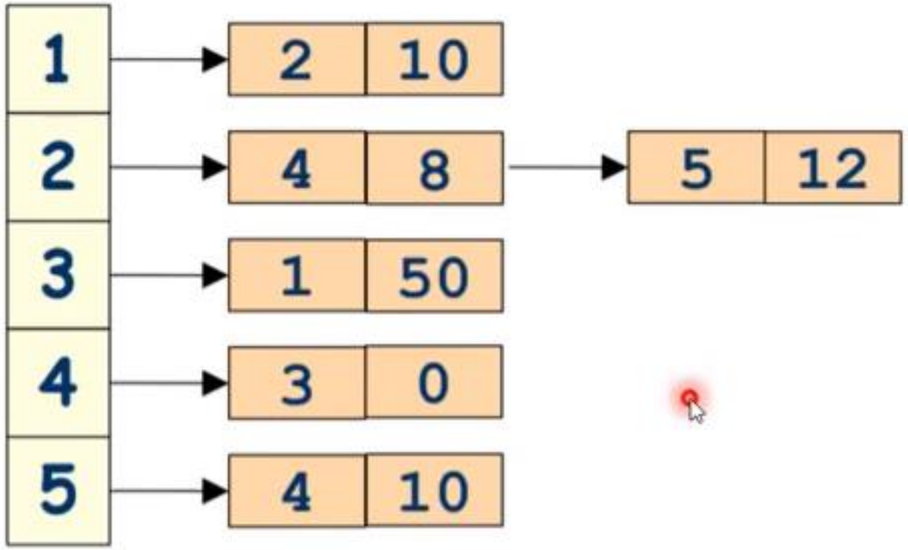
Exemplo



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

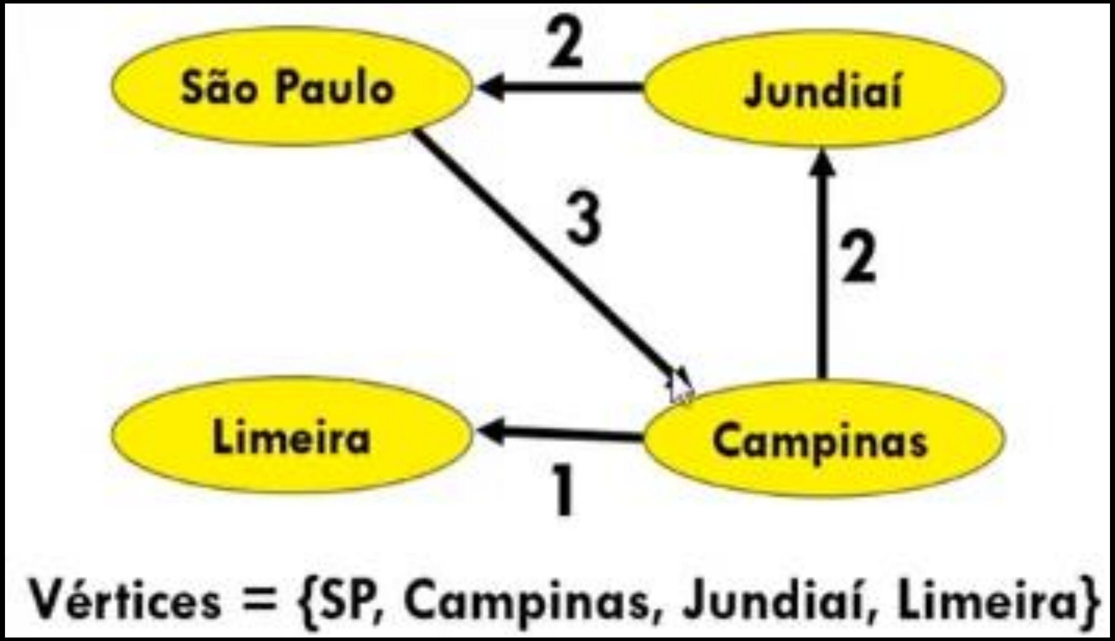
Listas de adjascentes





Pesos

- Dado o grafo abaixo, como representar na matriz adjascente por pesos?



Vantagens e Desvantagens

1-Com matrizes de adjacências, alocaremos espaços para a matriz inteira no momento da declaração da matriz (como em arranjos) , antes de sabermos o número de vértices e o número de arestas.

2-Se o grafo for denso (muitas arestas em relação ao número de vértices), a lista de adjacências ocupa muito espaço em memória.

3-As matrizes de adjacências ocupam o mesmo espaço em grafos esparsos muitos números 0) e densos (muitos números 1).

4-Buscas são melhores com listas de adjascências , pois já temos os adjascentes de um nó.

5-Testar se existe uma aresta entre dois vértices dados os índices, é melhor com matrizes de adjacência.

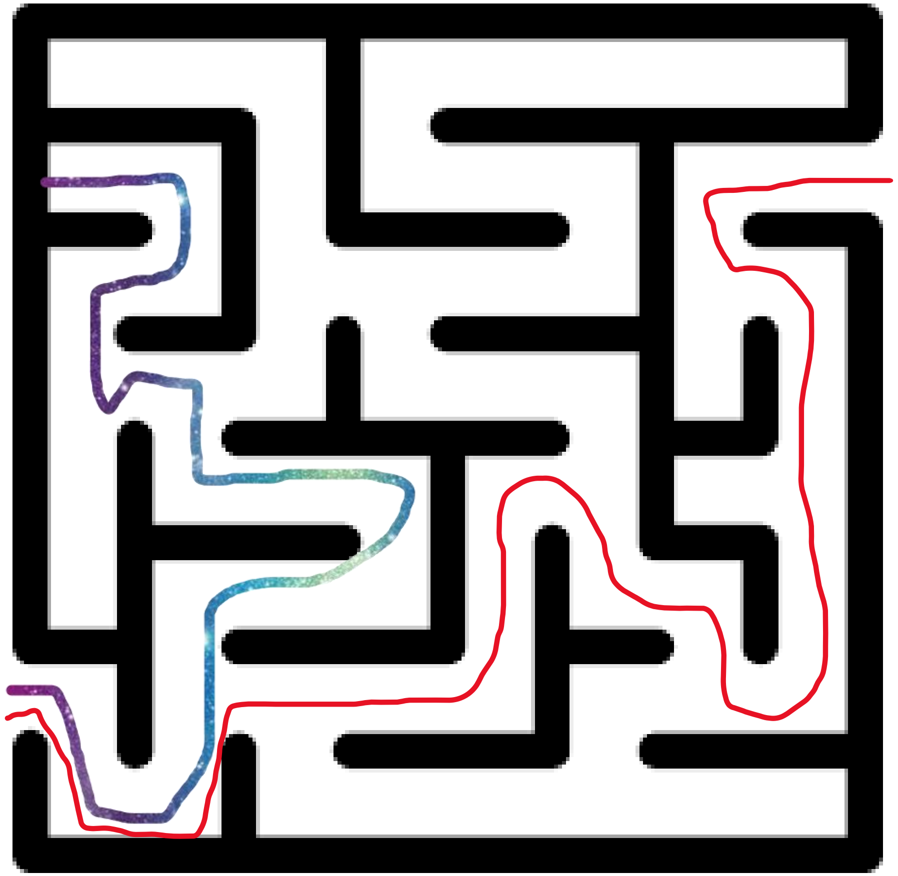
6-Encontrar os predecessores de um nó é melhor com matrizes de adjacência (vc olha a coluna), pois basta olhar a coluna do nó na matriz. Precisaríamos varrer todas as listas se usássemos listas de adjacência)

Busca em profundidade

- A estratégia consiste em se aprofundar no grafo sempre que possível.
- Se há um ponto do grafo e já percorremos tudo ao redor, voltamos para o vértice anterior (backtracking) procurando caminhos não explorados.

  A busca finaliza quando :

- Encontramos o que queremos
- Visitamos todos os vértices e não achamos o que queremos.



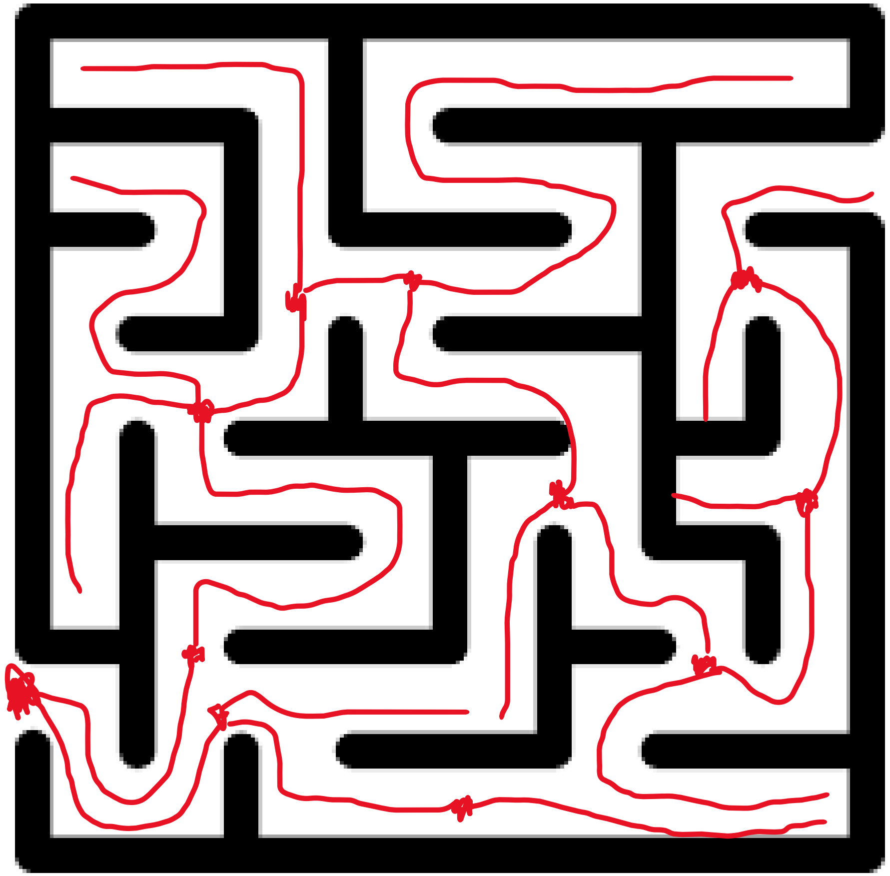
Exemplo labirinto

Busca em Profundidade

Busca em largura

- A estratégia consiste em explorar sem se afastar tanto do ponto inicial.
- Primeiro devemos seguir um caminho próximo a origem (caminho curto) . Se não acharmos o que queríamos, voltamos para o início e tentamos outro caminho.

- Ao encontrar o que desejamos , garantimos que sabemos uma maneira rápida de chegar até ele.
- Em geral só identificamos vértices a uma distância k+1 se todos os vértices de distância k já tiverem sido visitados.



Exemplo labirinto

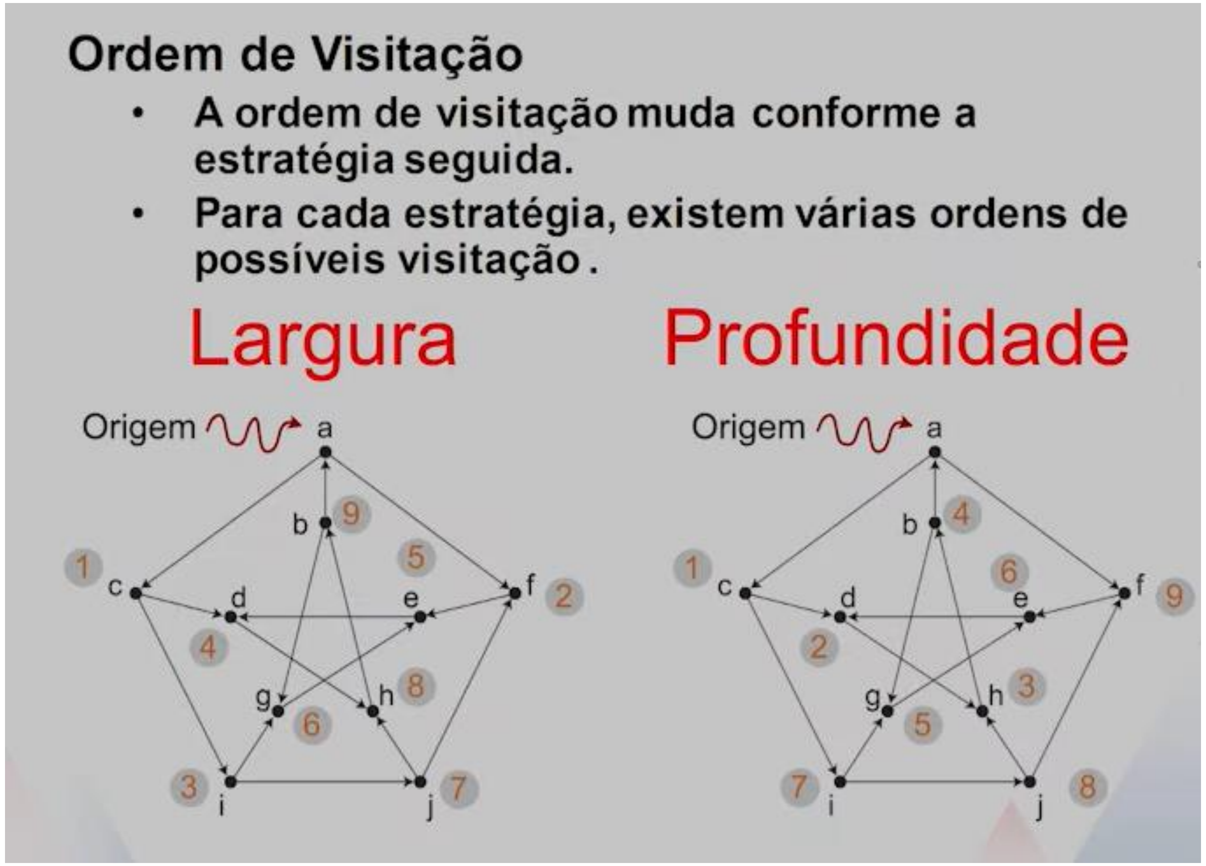
Busca em Largura

Como funciona a busca em Largura

- Os nós a serem visitados são colocados em uma fila, inicialmente contendo apenas o nó inicial.

- Na medida em que visitamos um nó, colocamos seus vizinhos na fila, mas somente se não estiverem lá.
- Continuamos até achar o nó alvo da busca ou a fila ficar vazia
- Se a fila estiver vazia e ainda restarem nós não visitados, reiniciamos o procedimento a partir de um destes.

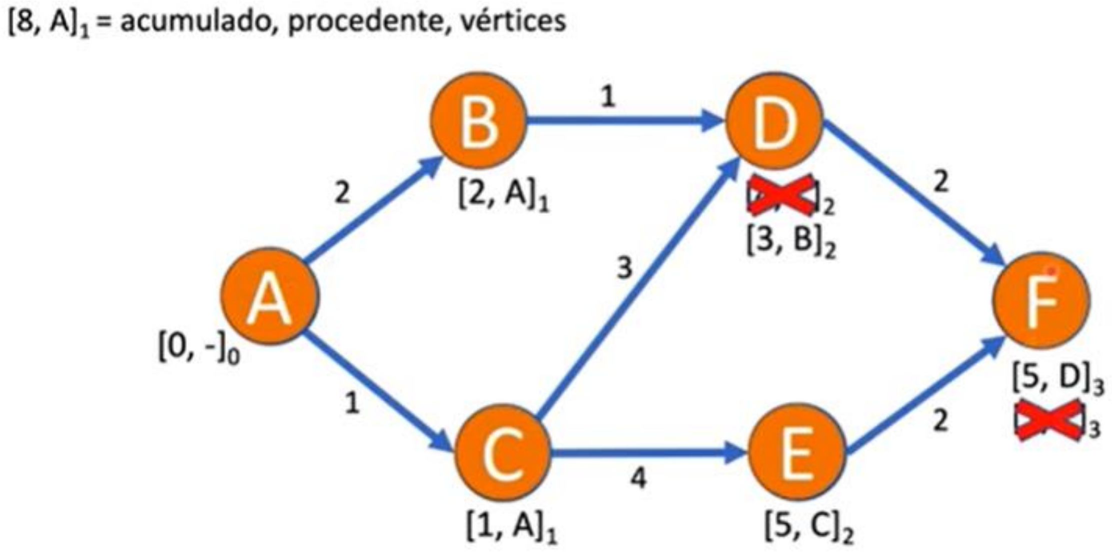
Buscas no Grafo



Algoritmo de Dijkstra

- Já estudamos que a busca em largura nos dá o menor caminho entre um vértice inicial e todos os demais no grafo.
- O caminho é medido em número de arestas, ignorando quaisquer pesos que estas tenham.
- Para vários problemas , contudo, o peso nas arestas é crucial, como quando queremos achar a rota mais curta entre duas cidades.
- No algoritmo de Dijkstra é calculado o caminho mais curto, em termos do peso total das arestas, entre um nó inicial e todos os demais nós no grafo.

- Edsger Dijkstra , 1959
- Encontra o caminho mais curto de um nó específico até todos os outros.

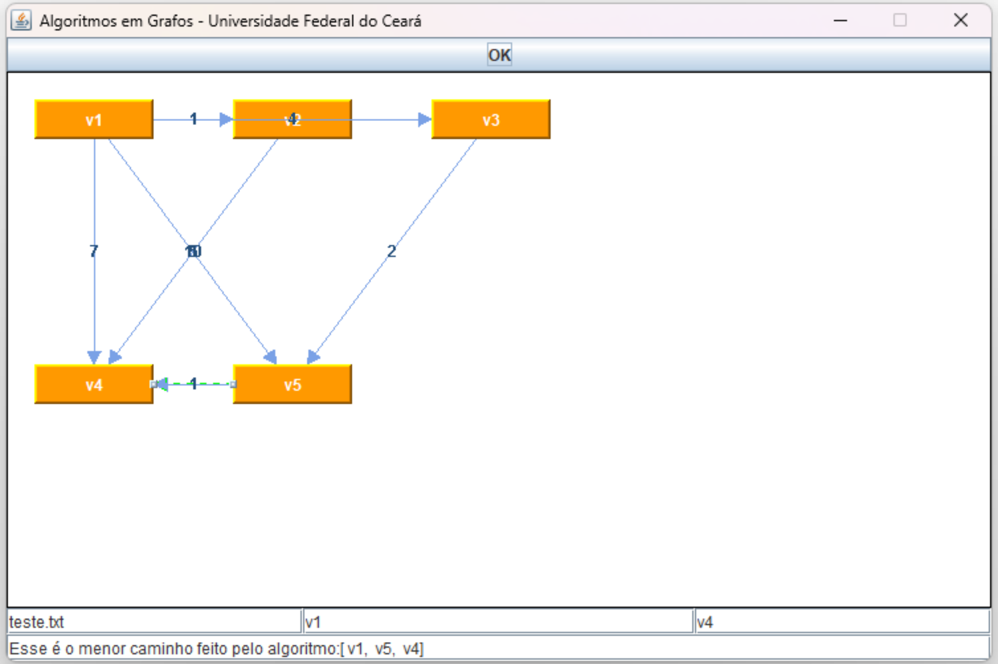


Dijkstra - Desvantagens

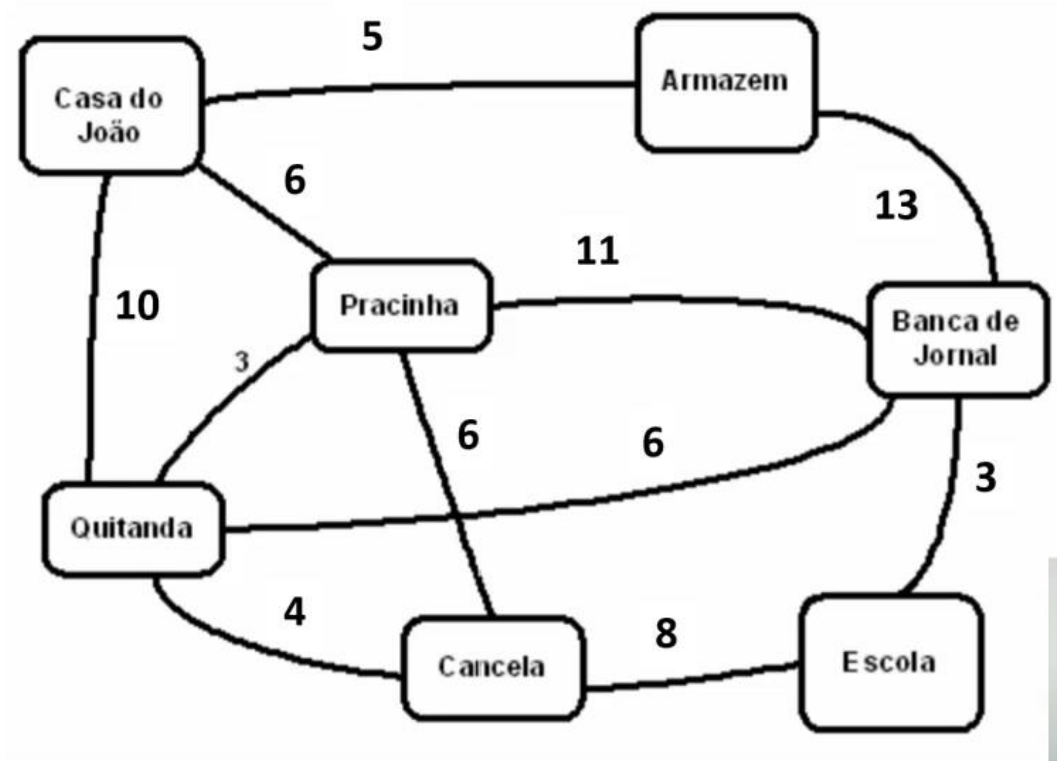
- Problemas de grafos com pesos negativos.
- Grafos muito grandes demandam muito poder de processamento.

Exemplo implementado

```
run:
Pegou esse vertice:   v1
Olhando o vizinho de  v1:  v4
Olhando o vizinho de  v1:  v3
Olhando o vizinho de  v1:  v5
Olhando o vizinho de  v1:  v2
Nao foram visitados ainda:[ v2,  v3,  v5,  v4]
Pegou esse vertice:   v2
Olhando o vizinho de  v2:  v4
Nao foram visitados ainda:[ v3,  v5,  v4]
Pegou esse vertice:   v3
Olhando o vizinho de  v3:  v5
Nao foram visitados ainda:[ v5,  v4]
Pegou esse vertice:   v5
Olhando o vizinho de  v5:  v4
Nao foram visitados ainda:[ v4]
Pegou esse vertice:   v4
Nao foram visitados ainda:[]
Esse é o menor caminho feito pelo algoritmo:[ v1,  v5,  v4]
```



Melhor Caminho



- Implementação em Java

- Atualize o teste.txt com os caminhos do grafo anterior.
- Substitua os nomes por v1, v2, v3 ....