

Atividades Desplugadas para Anos Finais

BNCC Computação

Guilherme Pimentel

João Vítor Bernardi

Daniel Farias Nascimento

Miguel Miron Silva

Francisco das Chagas Sousa Júnior

Prof^a Andrea Schwertner Charão

Índice

Apresentação.....	4
Unidade 1.....	5
Capítulo 1: Introdução a listas e matrizes.....	6
Capítulo 2: Algoritmos clássicos de ordenação e pesquisa de dados.....	11
Unidade 2.....	15
Capítulo 1: Introdução a grafos.....	16
Capítulo 2: Quantos apertos de mão nos separam?.....	18
Capítulo 3: Percorrendo grafos.....	20
Unidade 3.....	23
Capítulo 1: Introdução a árvores binárias.....	24
Capítulo 2: Desvendando o akinator.....	27
Capítulo 3: Montando seu akinator.....	31
Unidade 4.....	35
Capítulo 1: Grafos e Algoritmos.....	36
Capítulo 2: Vans de Sorvete.....	38
Unidade 5.....	45
Capítulo 1: Introdução a topologias de redes.....	46
Capítulo 2: Cama de Gato para representar topologias.....	52

Apresentação

Este material apresenta uma coletânea de propostas pedagógicas voltadas ao ensino de conceitos fundamentais da Computação, de forma acessível e interativa, para estudantes do Ensino Fundamental II.

Elaborado por estudantes de Ciência da Computação da UFSM, no âmbito da disciplina de Práticas Extensionistas na Educação em Computação, o conteúdo foi estruturado com base nos objetos do conhecimento e habilidades previstas na Base Nacional Comum Curricular (BNCC) para a área de Computação. O diferencial da abordagem está na proposta de atividades desplugadas, que dispensam o uso de computadores, promovendo o desenvolvimento do raciocínio lógico e da resolução de problemas por meio de jogos, desafios e dinâmicas em grupo.

Dividido em cinco unidades, o material oferece uma sequência didática que parte de conceitos básicos como listas e matrizes, avança por algoritmos de ordenação e pesquisa, aborda grafos e árvores binárias, até chegar a noções de topologias de redes. Cada capítulo apresenta uma breve fundamentação teórica seguida de atividades práticas. A proposta visa, assim, fortalecer o ensino de Computação de forma lúdica e contextualizada, alinhada com a BNCC.

Unidade 1

Batalha Naval



(<https://play.google.com/store/apps/details?id=com.extremedevelopers.forceofwarships&pli=1>)

Capítulos:

- Capítulo 1: Introdução a listas e matrizes (EF08C002)
- Capítulo 2: Algoritmos clássicos de ordenação e pesquisa de dados(EF08C003)

Capítulo 1: Introdução a listas e matrizes

Introdução:

Frequentemente, na Computação, temos a necessidade de fazer uso de estruturas de dados para resolução de problemas. Entre elas, a lista destaca-se como talvez a mais importante, pois serve de bloco de construção para diversas outras estruturas e está presente em inúmeras aplicações. A compreensão do funcionamento das listas e suas aplicações, também de como acessar seus dados e percorrê-la, é fundamental para a resolução de muitos problemas, não só na Computação, mas também em múltiplas outras áreas e assuntos.

O que são listas?

Uma lista é um conjunto de elementos organizados um depois do outro. Cada elemento tem uma posição (ou índice) na lista.

Exemplo:

```
[ maçã, banana, uva, melancia ]  
      1         2         3         4
```

Usamos listas para organizar informações!

O que são matrizes?

Uma matriz é como uma tabela: os elementos ficam organizados em linhas e colunas. Diferente da lista, na qual cada elemento tem uma posição que depende apenas de seu índice, em uma matriz, cada elemento tem uma posição que depende de duas informações:

1. Linha (horizontal)
2. Coluna (vertical)

Exemplo:

| 1, 2, 3 |

| 4, 5, 6 |

| 7, 8, 9 |

Uma matriz também pode ser pensada como uma lista de listas, ou como uma lista com duas dimensões:

```
[ [1, 2, 3],  
  [4, 5, 6],  
  [7, 8, 9] ]
```

Atividade: Batalha naval

Batalha Naval é um jogo de estratégia entre dois jogadores. O objetivo é afundar todos os navios do adversário antes que ele afunde os seus!

Como funciona?

Cada jogador tem dois tabuleiros:

1. Um para posicionar seus próprios navios
2. Outro para tentar acertar os navios do adversário

O tabuleiro é uma grade com linhas e colunas, como uma **matriz**!



(<https://apps.microsoft.com/detail/9wzdnrcrdxdgg?hl=pt-BR&gl=GW>)

Como jogar?

- **Posicione seus navios** secretamente no tabuleiro.
- Depois, os jogadores se revezam dizendo posições (como "B3" ou "D5") para tentar **acertar** os navios do outro.
- O outro jogador responde: "**Água!**" se não acertou. "**Acertou!**" se acertou um navio.
- Quando todas as partes de um navio são atingidas, ele está **afundado!**
- **Ganha quem afundar todos os navios do oponente primeiro!**

Objetivo

Essa atividade tem como objetivo trabalhar o entendimento lógico e intuitivo das matrizes das crianças. No final da atividade, espera-se que os alunos:

- Entendam como funciona o endereçamento de elementos em listas e matrizes (posições codificadas, como "B3" e "D5").
- Comecem a desenvolver um entendimento sobre formas de percorrer listas e matrizes.

Avançando o conhecimento: algoritmos e programação

Vamos programar um **sonar para achar os navios** para nós automaticamente! Primeiro vamos começar com apenas uma linha do tabuleiro (ou uma lista): 'X' representa um navio e '~' um vazio. Como nosso sonar poderia encontrar um navio nesse cenário?

```
linha = [ '~', '~', 'X', '~' ]
```

Resposta:

Começando da posição 1 da lista, podemos percorrer cada posição da lista até o fim. Se o sonar encontra um navio, ele marca essa posição. Vamos escrever uma sequência de instruções (ou um algoritmo) para fazer esta operação:

```
para cada posição i na lista:  
    se achar um barco:  
        marcar posição i
```

Fazendo uso desse algoritmo, podemos encontrar todos os navios em **uma linha** do tabuleiro.

Vamos ver como isso pode ser aplicado para encontrar **todos os navios** no tabuleiro inteiro. Agora temos que percorrer uma matriz inteira (dica: lembre-se que se pode pensar em uma matriz como sendo uma lista de listas...)

```
tabuleiro =  
[ [ '~', '~', '~' ],  
  [ '~', 'X', '~' ],  
  [ '~', '~', '~' ] ]
```

Resposta:

Podemos usar um raciocínio muito parecido para percorrer a matriz:

```
para cada lista L da matriz:  
  para cada posição P na lista:  
    se achar um barco:  
      marcar posição P da lista L
```

Poderíamos também pensar de outra forma muito semelhante

```
para cada linha i da matriz:  
  para cada coluna j da matriz:  
    se achar um barco:  
      marcar posição em (i, j)
```

Observe que a partir da segunda instrução, o algoritmo para encontrar navios em uma matriz é **igual ao de encontrar navios em uma lista**. Isso mostra que, apesar de ser uma das estruturas de dados mais simples da computação, a lista é extremamente útil na resolução de problemas e está por trás de inúmeras outras estruturas de dados.

Saiba mais: Listas e matrizes no mundo real

Você pode não perceber, mas **listas e matrizes** estão por trás de várias tecnologias e situações do dia a dia! Elas são fundamentais para organizar, armazenar e processar informações de maneira eficiente. Vamos ver alguns exemplos?

- Aplicativos e redes sociais: Quando você abre um aplicativo como o **WhatsApp, Instagram ou YouTube**, o que você vê é uma lista (de conversas, fotos, vídeos...)
- Games: Jogos como Minecraft usam matrizes para representar o mundo e a posição de entidades.
- Inteligência Artificial: Um **modelo de linguagem**, como o ChatGPT, usa milhares de **listas e matrizes** para organizar palavras, frases e significados.

Capítulo 2: Algoritmos clássicos de ordenação e pesquisa de dados

No último capítulo, vimos como podemos escrever algoritmos para percorrer listas. Agora, vamos ver como podemos usar e modificar esses algoritmos para resolver problemas mais complexos.

Pesquisa linear de dados

Uma das operações realizadas sobre listas mais comuns é a pesquisa de dados. Essa operação pode ser implementada de diversas formas, porém a mais simples consiste em percorrer os elementos da lista em busca de um valor específico, comparando esse valor com cada item até que seja encontrado ou que todos os elementos tenham sido verificados (Igual ao que foi feito para programar o “sonar” no capítulo 1 desta unidade).

Esse tipo de busca é chamado de **pesquisa linear** porque segue uma verificação sequencial, do início ao fim da lista. Embora simples, esse método pode ser **ineficiente em listas grandes**, pois seu tempo de execução cresce proporcionalmente ao número de elementos. Ainda assim, é amplamente utilizado por sua fácil implementação e por ser eficaz em listas pequenas ou não ordenadas.

Vamos ver novamente como esse algoritmo pode ser descrito para pesquisar um valor **x**:

```
para cada posição i na lista:  
    se lista[i] == x:  
        retornar lista[i]
```

exemplo 1:

Para buscar o valor **30** na lista **[20, 50, 30, 40, 10]**, temos:

[20, 50, 30, 40, 10]

Iterações do algoritmo:

1. Lista[1] == 10 → 20 ≠ 30 (não achou)
2. Lista[2] == 20 → 50 ≠ 30 (não achou)
3. Lista[3] == 30 → 30 = 30 (achou)

Percebe-se que esse algoritmo não é eficiente para listas grandes. Considere uma lista com 1 milhão de elementos, por exemplo. Se queremos encontrar o valor X nessa lista, porém ele está contido na penúltima posição, o algoritmo será repetido 999,999 vezes.

Ordenação de dados

Agora imagine uma situação em que temos que registrar a nossa frota de navios de guerra em uma lista de um sistema. Porém, como existe uma grande variedade de tamanhos de navios, registrá-los de forma aleatória talvez não seria uma boa ideia pois tal registro ficaria mal organizado e pode dificultar a pesquisa.



(<https://apps.microsoft.com/detail/9wzdncrdxddg?hl=pt-BR&gl=GW>)

Para resolver esse problema, seria interessante registrar os navios em **ordem crescente de tamanho**. Como podemos fazer isso? Na computação, isso é o que chamamos de um problema de **ordenação de dados**, os quais são extremamente comuns e críticos para a performance de aplicações.

Vamos começar por dar um número para cada navio que representa seu tamanho:



Adicionando todos a uma lista temos:

$$\text{Navio} = [4, 3, 3, 2, 2, 1, 1, 1]$$

Usando os conhecimentos adquiridos sobre listas e como percorrê-las, como podemos escrever um algoritmo que troque a posição dos elementos da lista para ordená-la por ordem crescente?

Resposta:

Esse é um problema tão comum na computação que existem diversos algoritmos pré-existentes para sua resolução. Vamos analisar um dos algoritmos de ordenação mais simples de todos: o **bubble-sort**.

Como funciona esse algoritmo?

Comparamos dois elementos vizinhos. Se estão na ordem errada, os trocamos de posição. Esse processo é repetido várias vezes até a lista ficar ordenada.

(Se possível, pedir alguns alunos voluntários para ficar de pé e representar os elementos da lista, se não, usar quadro mesmo)

O algoritmo passa por todas as posições da lista, comparando os dois elementos por vez. Se estiverem fora de ordem, troca eles de posição, se não, não faz nada e passa para a posição seguinte. Depois de um certo número de passadas, a lista estará ordenada. Vamos ver como podemos escrever esse algoritmo:

```
# n = tamanho da lista

para cada posição i na lista:
    para cada posição j em (0, n - 1):
        se lista[ j ] > lista[ j + 1 ]:
            Troca valores de lista[ j ] e lista[ j + 1 ]
```

(Mostrar passos do algoritmo com os voluntários ou no quadro)

Percebe-se que, de forma similar ao algoritmo de pesquisa linear, o bubblesort também é ineficiente para listas grandes visto que percorre a lista inteira no laço mais externo (linha 1) e percorre-a múltiplas vezes dentro do laço (linha 2)

Saiba mais: Onde usamos ordenação e pesquisa de dados na vida real?

Você já parou para pensar como seu celular, seu computador ou até os sites que você visita **encontram as coisas tão rápido**? Ou como eles sabem a **ordem certa** de mostrar as informações? A resposta está nos **algoritmos de pesquisa e ordenação de dados** que estudamos neste capítulo. Vamos ver onde isso aparece no mundo real?.

- Celular e aplicativos: Quando você digita o nome de um contato, seu celular **procura** esse nome em uma lista.
- Sites de compras: Os produtos que aparecem primeiro numa loja online geralmente estão **ordenados por preço, por mais vendidos ou por avaliação dos clientes**.
- **Bancos de dados**: São programas utilizados para armazenar informações como o cadastro de usuários e precisam encontrar e organizar bilhões de informações o tempo todo.

Veja também:

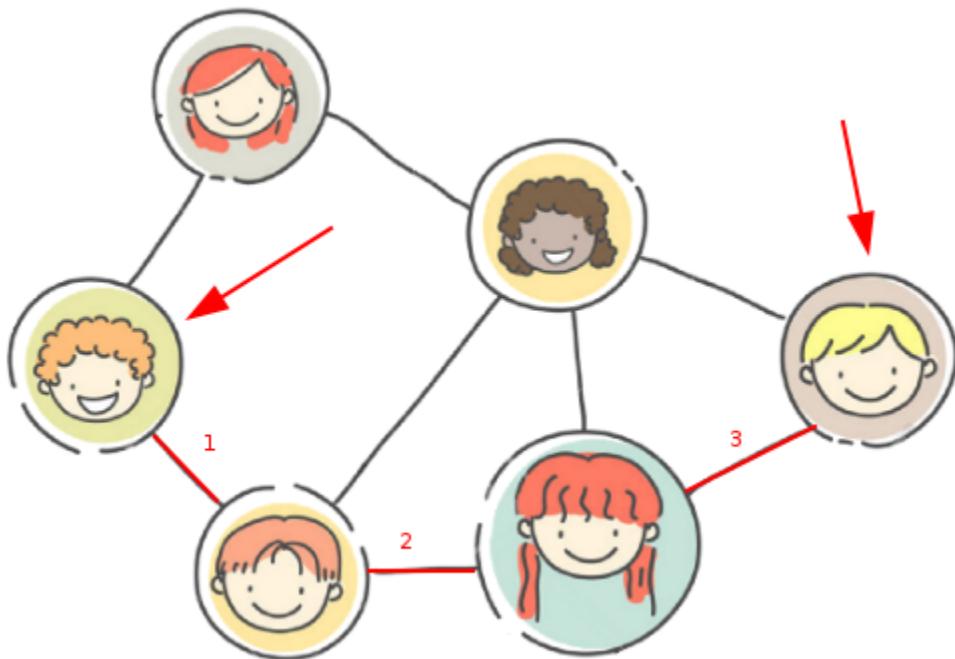
- Visualização do bubble sort:
https://www.youtube.com/watch?v=Cq7SMsQBEUw&ab_channel=TimoBingmann

Referências

- Pesquisa e Ordenação de Dados 2a edição - Gerardo Viana, Glauber Cintra e Ricardo Nobre
- Estrutura de dados - <https://github.com/BenhurUFSM/I224a>

Unidade 2

Grafos no dia a dia



(<https://medium.com/@rsorage/grafos-2-diferentes-tipos-de-grafos-com-exemplos-pr%C3%A1ticos-e4646c4f1ce2>)

Capítulos:

- Capítulo 1: Introdução a grafos (EF05C002), (EF07C004).
- Capítulo 2: Quantos apertos de mão nos separam (EF09C001).
- Capítulo 3: Percorrendo grafos (EF09C001).

Capítulo 1: Introdução a grafos

Introdução:

Por muitas vezes na computação e em outras áreas, surge a necessidade de modelar relações e conexões entre entidades para resolver problemas complexos. Entre as estruturas capazes de representar essas interações, os **grafos** destacam-se como uma das mais poderosas e versáteis, servindo como base para modelagem em contextos que vão desde redes sociais até sistemas de navegação. A compreensão do funcionamento dos grafos – incluindo sua construção, análise e algoritmos associados – é essencial não apenas para a Ciência da Computação, mas também para campos como biologia, engenharia, logística e inteligência artificial, onde padrões de conexão são fundamentais.

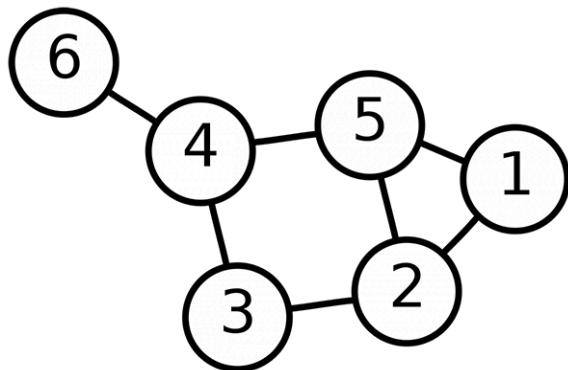
O que são grafos:

Um grafo é uma estrutura matemática usada para representar relações entre objetos. Ele é composto por:

- **Vértices (nós):** Pontos que representam objetos (ex.: cidades, pessoas, computadores).
- **Arestas (ligações):** Linhas que conectam os vértices, indicando relações (ex.: estradas, amizades, cabos de rede).
- **Caminho:** Sequência de arestas que ligam dois vértices.

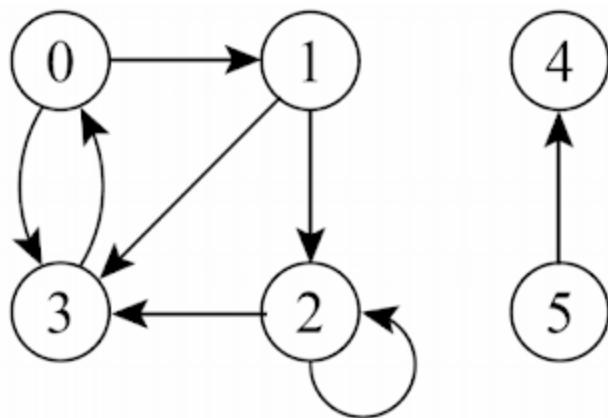
Tipos de Grafos:

- **Não direcionado:** Arestas sem direção (ex: rede social onde amizades são mútuas)



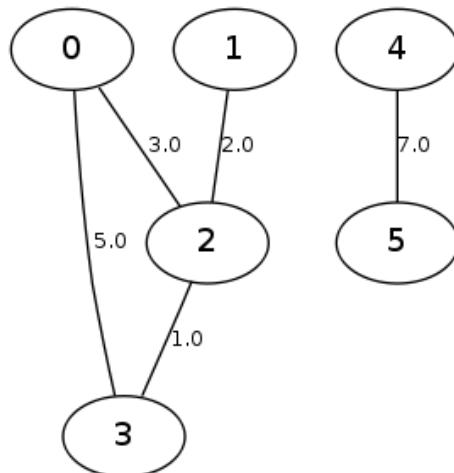
(<https://www.revista-programar.info/artigos/grafos-1a-parte/>)

- **Direcionado:** Arestas com direção (ex: fluxo de tráfego em ruas).



(<https://h3dema.blogspot.com/2016/10/componentes-conectados-em-grafos.html>)

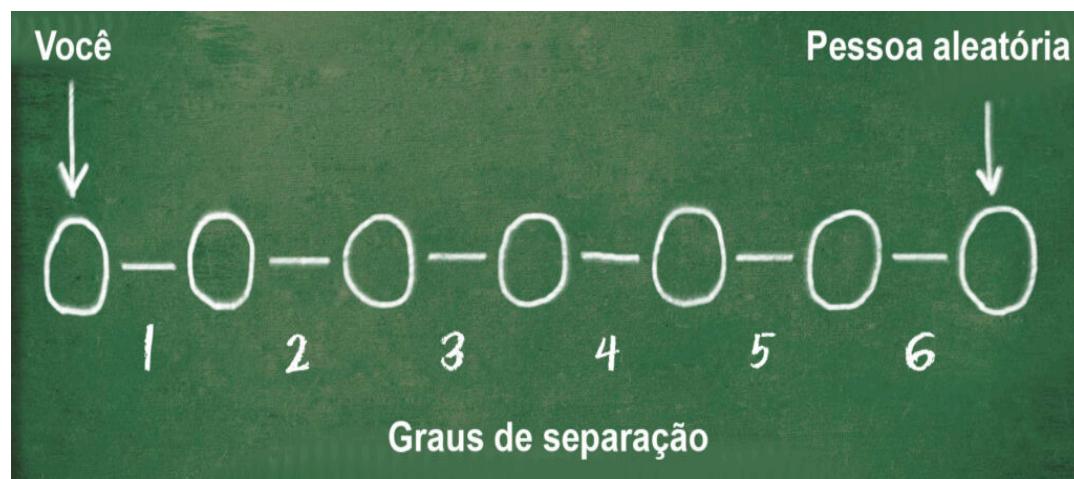
- **Ponderado:** Números nas arestas que representam tempo, distância ou custo de deslocamento (ex: distância em quilômetros entre dois pontos em uma cidade).



(<https://blog.tiagomadeira.com/2006/01/grafos-ponderados/>)

Capítulo 2: Quantos apertos de mão nos separam?

A Teoria dos Seis Graus de Separação (ou "seis apertos de mão") é um conceito que sugere que qualquer pessoa no mundo pode estar conectada a qualquer outra através de uma cadeia de no máximo 6 conexões.



(https://ilhadoconhecimento.com.br/6_graus_separacao/)

Representação em Grafos

- Vértices (nós): Pessoas.
- Areias: Relações de amizade/conhecimento.
- Distância: Número mínimo de arestas entre dois vértices.

Como realizar

- Escreva em um quadro ou uma folha de cartolina o nome de todos os integrantes da atividade a fim de formar um círculo de nomes.
- Cada integrante deve cumprimentar 3 colegas diferentes e registrar as conexões no círculo ligando os nomes.
- O desafio é encontrar o caminho mais curto entre dois integrantes escolhidos aleatoriamente, contando os “apertos de mão” (arestas).

Analise coletiva

- A turma verifica se é possível conectar qualquer dupla com até 6 arestas (como na teoria original).
- Discussão: "O que acontece se removemos uma aresta? O grafo ainda é conectado?"
- A atividade não precisa se limitar somente aos alunos da turma, podendo ser realizada ligando pessoas famosas (ex: ligar Neymar Jr. à Mahatma Gandhi em até 6 “apertos de mão”).

Objetivo

Essa atividade tem como objetivo trabalhar o entendimento do que é um grafo, seus diferentes tipos e suas aplicações no mundo real nos alunos. No final da atividade, espera-se que os alunos:

- Entendam o que é a estrutura matemática **grafo**
- Comecem a observar as diversas aplicações da teoria dos grafos no seu dia a dia.

Capítulo 3: Percorrendo grafos

Vamos criar um programa em pseudocódigo que percorra os pontos de um grafo de uma cidade a fim de chegar a um destino específico.

Primeiro montamos uma estrutura do lugar em que estamos, que conterá o nome do lugar e uma lista com todos os seus vizinhos.

```
// Estrutura básica de um lugar na cidade  
Estrutura Lugar:  
    nome: texto  
    vizinhos: lista // nomes dos lugares conectados
```

Em seguida, criamos nossa função principal de busca, que opera sobre 2 pontos/lugares (atual e destino) e com uma lista de lugares visitados.

```
// Função principal de busca  
Função buscar_caminho(atual, destino, visitados):  
    // Marca o local atual como visitado  
    visitados.adicionar(atual.nome)
```

Caso o local atual tenha o mesmo nome do destino, então nós chegamos no destino e a função retornará verdadeiro.

```
// Se chegou ao destino  
Se atual.nome == destino.nome:  
    print("Chegou ao destino")  
    Retornar verdadeiro  
Fim
```

Caso o local atual tenha nome diferente do destino, nós obtemos a posição dos vizinhos e vamos em direção aos que não foram visitados ainda.

```
// Se não chegou ao destino, procura nos vizinhos  
Para cada vizinho em atual.vizinhos:
```

```
    lugar = obter_lugar(vizinho) // Função que encontra o
vizinho
    Se lugar.nome não está em visitados: // Se o vizinho
não foi visitado ainda
        buscar_caminho(lugar, destino, visitados): // Vai
até o vizinho
    Fim
Fim
```

Caso não seja possível encontrar o destino a partir do ponto de partida informado, a função retornará falso.

```
// Se não encontrar nenhum caminho para o destino
Retornar falso
Fim
```

```
// Programa principal
Início
```

```
// Pede origem e destino
print("Digite o ponto de partida:")
scan(partida)
print("Digite o destino:")
scan(destino)
```

```
// Busca o caminho
Se buscar_caminho(partida, destino, visitados[]) for
verdadeiro:
    print("Destino encontrado!") // Se conseguir encontrar
um caminho
    Se for falso:
        print("Não há caminho possível.") // Se não for
possível encontrar um caminho
    Fim
Fim
```

Saiba mais: Onde usamos grafos na vida real?

Você já se perguntou como o GPS encontra o caminho mais rápido para sua casa? Ou como as redes sociais sabem quem são seus amigos em comum? Tudo isso é possível graças aos grafos, uma estrutura de dados que estudamos e que está escondida em muitos lugares do nosso dia a dia. Vamos descobrir onde eles aparecem?

- **Navegação e GPS:** Aplicativos como Google Maps e Waze usam grafos para representar ruas (arestas) e cruzamentos (nós), calculando sempre o caminho mais rápido para você.
- **Redes sociais:** Seu perfil é um nó e suas amizades são arestas. Quando o Instagram sugere novos amigos, está analisando conexões no grafo social.
- **Jogos eletrônicos:** Inimigos em jogos usam grafos para navegar pelo mapa e encontrar o jogador, como em labirintos inteligentes.
- **Recomendações:** Netflix e Spotify usam grafos para conectar usuários, filmes e músicas, sugerindo conteúdos baseados nessas relações.
- **Entregas e transportes:** iFood, Uber e outros serviços de entregas calculam rotas ideais usando grafos que consideram distância, trânsito e vias de mão única.

Veja também:

- O problema da ponte de Königsberg:
https://www.youtube.com/watch?v=lwmMItWLqlo&ab_channel=Sejacurioso%E2%80%94TE-D-Ed
- Como resolver um crime com grafos(inglês):
https://www.youtube.com/watch?v=TwHy2DuWB3k&ab_channel=SciencePlease

Unidade 3

Fazendo mágica com árvores binárias



[\(https://editalconcursosbrasil.com.br/noticias/2024/06/viciantes-ao-extremo-3-jogos-secretos-que-todo-dono-da-al
exa-deve-testar/\)](https://editalconcursosbrasil.com.br/noticias/2024/06/viciantes-ao-extremo-3-jogos-secretos-que-todo-dono-da-alexa-deve-testar/)

Capítulos:

- Capítulo 1: Introdução a árvores binárias (EF05C002,EF06C003)
- Capítulo 2: Desvendando o akinator (EF06C003,EF08C001)
- Capítulo 3: Montando seu akinator (EF08C001,EF09C001)

Capítulo 1: Introdução a árvores binárias

Introdução:

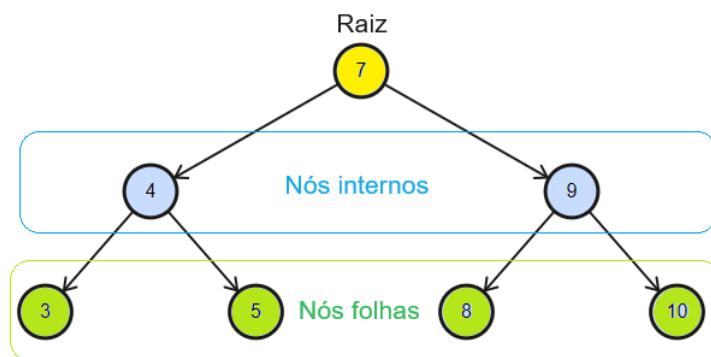
Árvores binárias são amplamente utilizadas para operações de busca, organização e manipulação de dados em estruturas hierárquicas. Sua simplicidade esconde uma poderosa ferramenta computacional, capaz de otimizar algoritmos de pesquisa, ordenação e tomada de decisões. Presentes em áreas como bancos de dados, inteligência artificial e compiladores, as árvores binárias são essenciais para tornar o processamento de informações mais eficiente, rápido e estruturado.

O que são árvores binárias?

É uma estrutura de dados hierárquica onde cada nó pode ter no máximo dois filhos.

Raiz, nós e folhas:

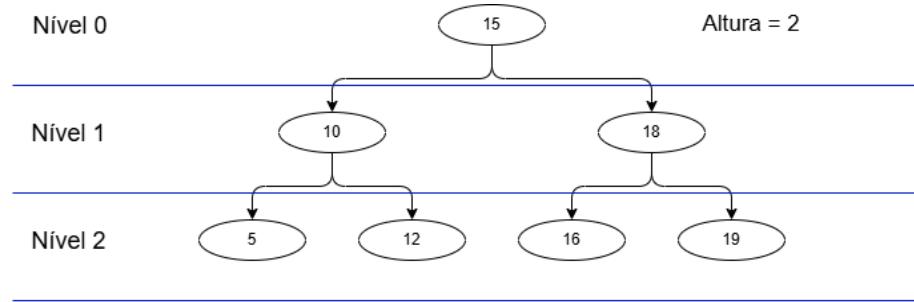
- Nó: Cada ponto (bolinha) da árvore que contém um valor (informação).
- Raiz: Primeiro nó da árvore.
- Folha: Nó que não tem filhos (final da árvore).



(<https://www.estategiaconcursos.com.br/blog/percursos-arvores-binarias/>)

Altura de uma árvore

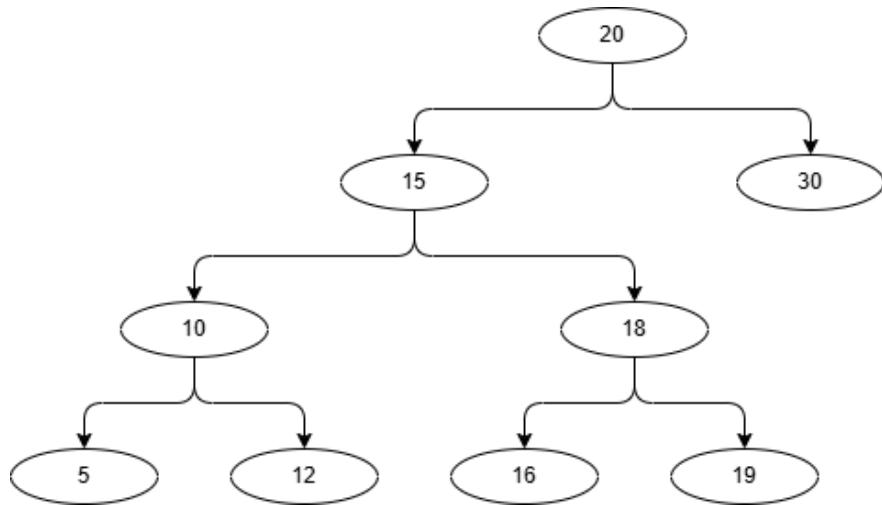
A altura é a quantidade de níveis do caminho mais longo da raiz até uma folha.



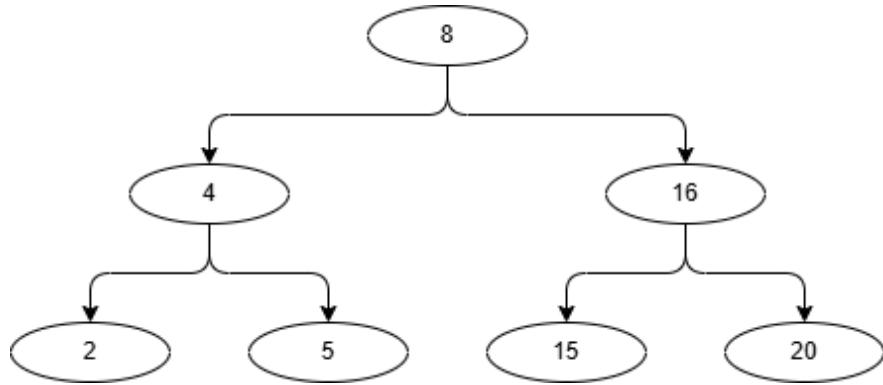
Balanceamento de árvores

Uma das características mais importantes de uma árvore é se ela está balanceada. O balanceamento refere-se à distribuição equilibrada dos nós entre os lados esquerdo e direito de cada subárvore. Quando uma árvore binária está desbalanceada suas operações podem se tornar ineficientes, perdendo sua vantagem em relação a uma simples lista.

Árvore desbalanceada:



Árvore balanceada:



Capítulo 2: Desvendando o akinator

O jogo:

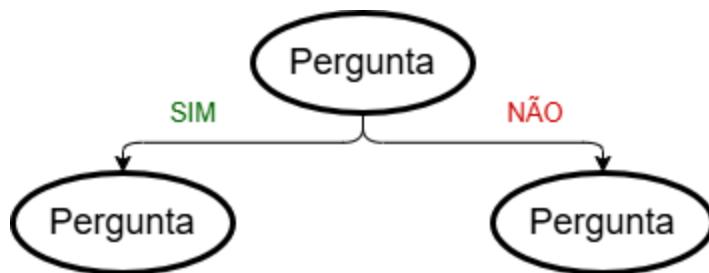
O jogo “Akinator” ganhou muita popularidade na internet devido a seu funcionamento “mágico”, que consiste em fazer uma série de perguntas ao jogador e o gênio iria adivinhar qual personagem ou pessoa real o jogador está pensando. Usualmente, 20 perguntas são suficientes para que o gênio saiba quem é o personagem.

Qual a mágica do jogo?

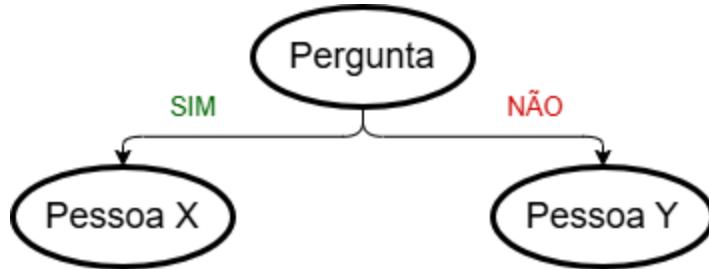
Podemos representar o funcionamento do akinator através de uma árvore binária:

- Nós internos e raiz: guardam uma pergunta.
- Nós folhas: guardam um personagem.

Sempre que a resposta para a pergunta for sim, seguimos para o nó filho esquerdo, sempre que a resposta for não, seguimos para o nó filho direito.

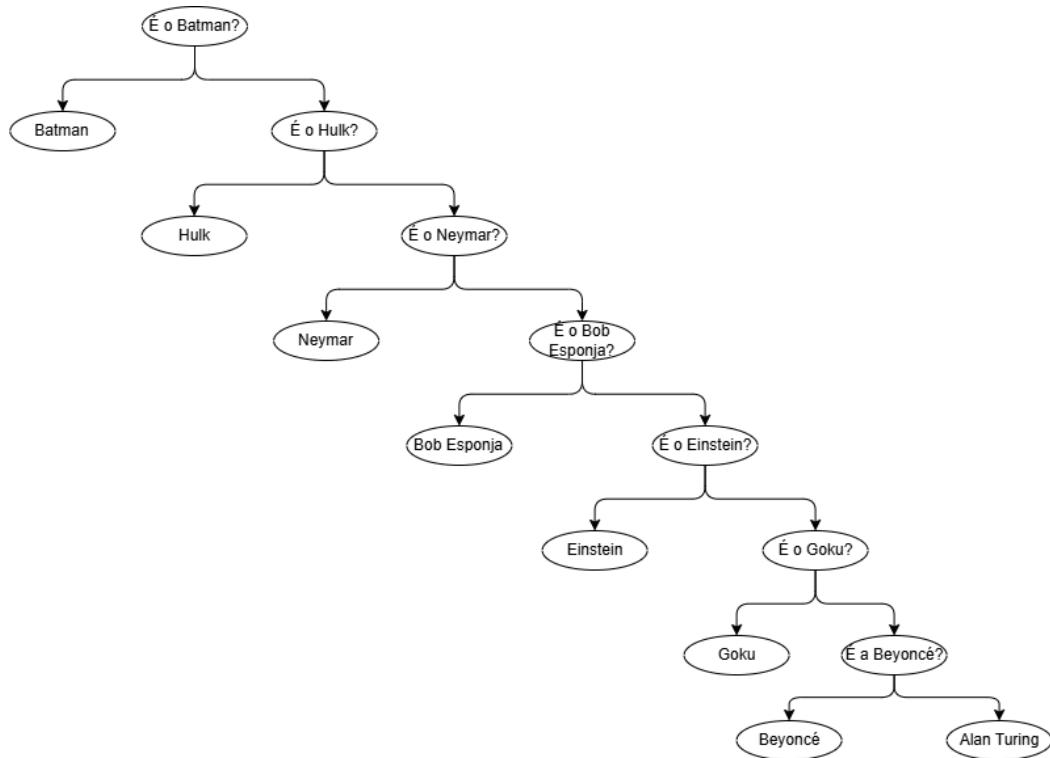


Seguimos o caminho de perguntas até chegarmos em um nó folha:

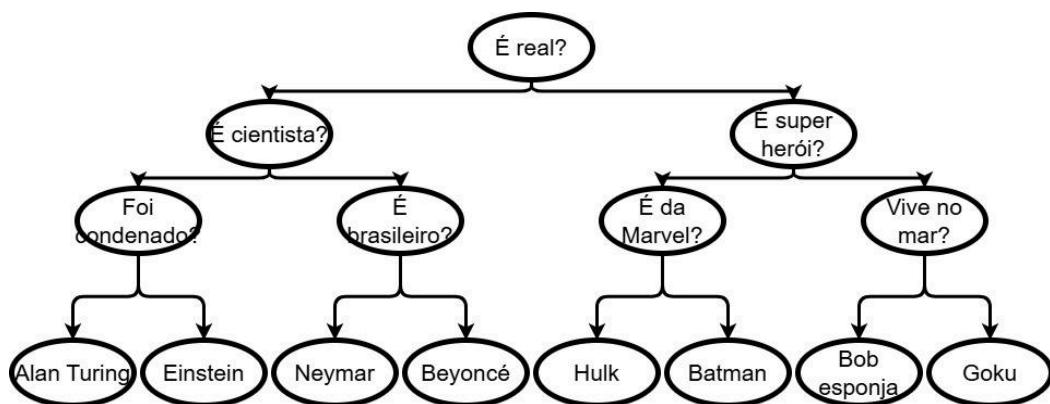


Exemplo:

Árvore desbalanceada:



Árvore balanceada:



Atividade: Vamos montar?

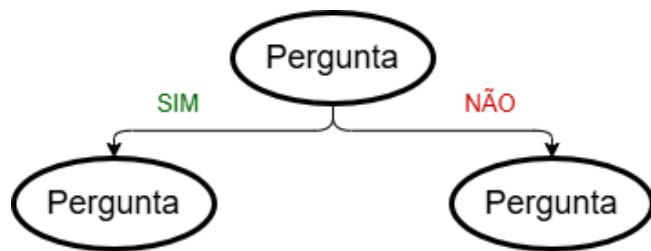
Hora de criar uma árvore de personagens como se fosse o gênio!

Escolha de personagens:

A turma deve escolher, em conjunto, 8 personagens para estarem em suas árvores. Não se limite, podem ser famosos, animais, países, o que for melhor!

Lembre-se:

- Cada nó interno é uma pergunta de SIM/NÃO.
- Nós esquerdos -> SIM e nós direitos -> NÃO.
- Cada nó folha (sem filhos) é um personagem.



Agora ao contrário!

Agora, os alunos receberão uma árvore com perguntas e devem achar personagens que se encaixam em cada nó folha. Dica: aplicar conteúdos de outra disciplina, como árvores filogenéticas da biologia ou figuras históricas.

Objetivo:

Compreender, de forma lúdica e prática, o funcionamento das árvores binárias por meio da construção e interpretação de uma estrutura de decisões, similar ao raciocínio utilizado pelo Akinator. A atividade visa desenvolver o pensamento lógico, estimular o trabalho em grupo e promover a aplicação de conceitos de computação em contextos interdisciplinares, como biologia e história

Trabalhando os conhecimentos

Utilizando seus conhecimentos sobre árvores de personagens, reflita sobre as seguintes perguntas:

- Qual uma vantagem e desvantagem de uma árvore de personagem balanceada?

Gabarito:

- Vantagem: menos perguntas necessárias para o mesmo número de personagens.
- Desvantagem: mais difícil de achar perguntas que separam o número de personagens ao meio.

- Como sabemos quantos personagens temos nela, o que temos que contar?

Gabarito: Contando os nós folhas da árvore.

- Como sabemos qual o número máximo de perguntas que faremos?

Gabarito: Determinando a altura da árvore.

Capítulo 3: Montando seu akinator

Agora que já aprendemos como as árvores binárias podem ser utilizadas para o jogo akinator, podemos simular o funcionamento do jogo em pseudocódigo.

Codificando nós:

Para representar os nós vamos utilizar estruturas. Pense nas estruturas como variáveis que guardam mais variáveis, ou uma caixa grande que guarda mais caixas dentro.

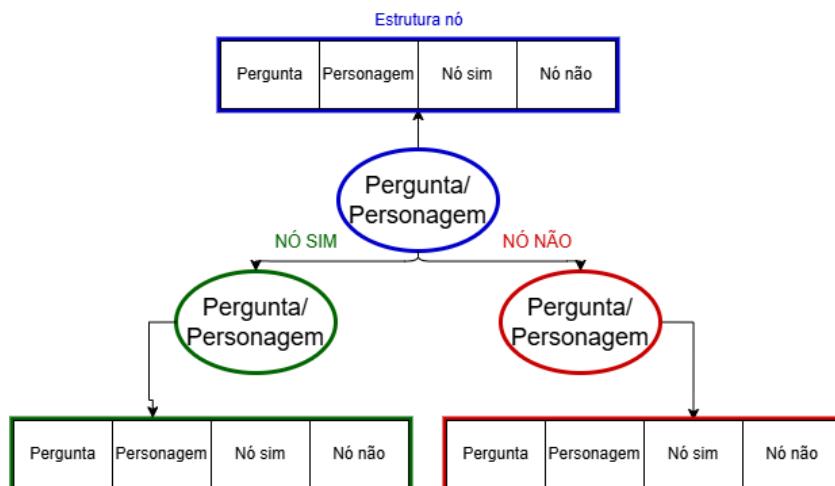
Estrutura Nó:

String pergunta

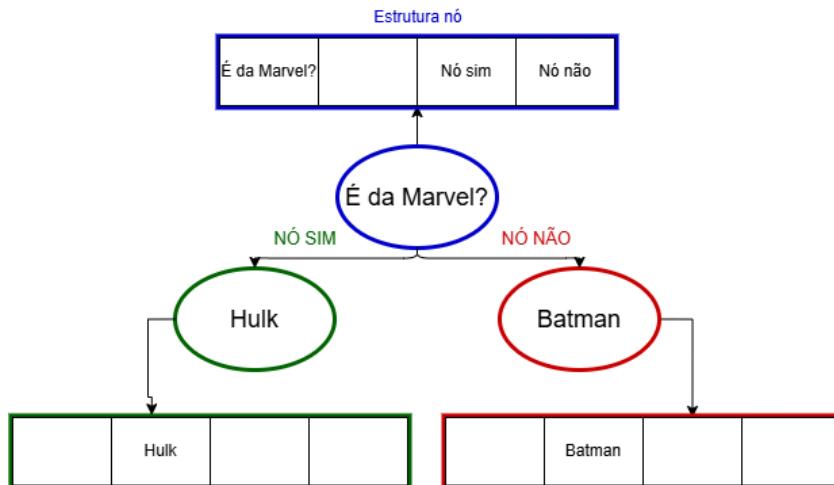
String personagem

Nó sim

Nó não



Exemplo:



Loop jogo:

Com base nesses nós, podemos definir um loop de gameplay.

Função jogar(no):

```
Se no.personagem:  
    imprime("Você pensou em:" + no.personagem)  
    return  
    imprime(no.pergunta + "s/n")  
    resp = ler()  
    se resp == "s":  
        jogar(no.sim)  
    senão:  
        jogar(no.nao)
```

Atividade: Algoritmos úteis

Utilizando seus conhecimentos sobre árvores e recursão desenvolva um algoritmo para:

1. Verificar se o personagem X está na árvore.
2. Descobrir o número de personagens na árvore.
3. Descobrir a quantidade máxima de perguntas necessária para descobrir o personagem.

Gabarito:

1.

```
Função pertence(no, alvo):
    se no está vazio:
        return false
    se no.personagem == alvo:
        return verdadeiro
    return pertence(no.sim, alvo) or pertence(no.nao, alvo)
```

2.

```
Função contar_folhas(nó):
    se nó está vazio:
        return 0
    se nó.personagem:
        return 1
    return contar_folhas(nó.sim) + contar_folhas(nó.nao)
```

3.

```
Função altura(no):
    se no está vazio or no.personagem:
        return 0
    return 1 + max(altura(no.sim), altura(no.não))
```

Unidade 4

Cidade Turística



(<https://www.shutterstock.com/image-photo/aerial-view-typical-buildings-barcelona-600nw-2165020871.jpg>)

Capítulos:

- Capítulo 1: Grafos e Algoritmos (EF05CO01, EF05CO02)
- Capítulo 2: Vans de Sorvete (EF07MA05, EF07MA06)

Capítulo 1: Grafos e Algoritmos

Introdução:

Vários tipos de problemas reais que envolvem conexões entre entidades, como otimização de rede de computadores, a busca pela melhor rota, a distribuição de serviços como estações de bombeiros em uma cidade, e et cetera, podem ser abstraídos (isto é, simplificados escondendo-se os detalhes desnecessários) e modelados como problemas envolvendo **grafos**, e resolvidos utilizando-se algum tipo de **algoritmo** - isto é, uma dada sequência de passos que sempre nos dará uma solução boa, ou até a melhor solução possível. Porém, devemos sempre prosseguir com cuidado - existem muitos problemas para quais soluções ótimas não são dadas por nenhum algoritmo, e ainda outros onde elas nem existem.

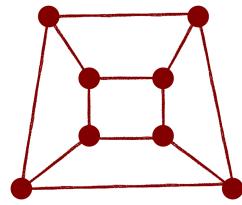
Revisando conceitos:

Como já visto antes, um grafo é uma estrutura matemática usada para representar relações entre quaisquer objetos, sendo composto pelos seguintes componentes :

- **Vértices (nós ou junções):** Pontos que representam objetos (ex.: cidades, pessoas, computadores).
- **Arestas (ligações):** Linhas que conectam os vértices, indicando relações (ex.: estradas, amizades, cabos de rede).
- **Caminho:** Sequência de arestas que ligam dois vértices.

Modelagem de um problema usando grafos:

Para começar, vamos utilizar um grafo muito simples de exemplo, com uma tarefa igualmente simples: Imagina que temos uma pequena cidade, e as pessoas dela nunca se dão ao trabalho de atravessar mais de uma rua para pegar algo. Cada vértice representa uma rua que deve ser atravessada, e cada nó uma loja em um quarteirão. Dada que a imagem abaixo representa a cidade, quantas lojas são necessárias para que nenhuma pessoa precise cruzar mais de uma rua?



(<https://desplugada.ime.unicamp.br/atividade15/cuboarestas.png>)

Podemos facilmente resolver esse exercício marcando apenas dois nós, com os dois posicionados diagonalmente opostos. Esse problema consiste em preencher os nós do grafo de modo que todos os nós estejam, no máximo, a uma distância de qualquer nó marcado.

A resolução desse problema pode ser feita de diversas formas. Podemos explorar todas as possíveis configurações de lojas até chegar a apenas duas lojas - isso é denominado o algoritmo de **força bruta**. Também podemos começar marcando nós a partir do nó mais conectado (qualquer que seja ele, caso haja mais de uma opção), e preencher o nó seguinte mais conectado - isso é denominado de algoritmo de **abordagem gulosa**. Logo veremos como esses algoritmos podem ou não ser usados em grafos maiores e mais complexos.

Capítulo 2: Vans de Sorvete

Apresentação do Problema:

Dado o seguinte mapa (da **Folha de Atividade: Vans de Sorvete**), temos o mapa de uma cidade, onde as linhas (ou arestas) são ruas e os pontos (ou nós) são esquinas. A cidade se localiza em um país muito quente onde, no verão, as vans estacionam nas esquinas das ruas e vendem sorvetes para turistas. Queremos posicionar as vans de modo que qualquer pessoa possa chegar até elas andando até o final da rua e, então, no máximo, mais um quarteirão. Quantas vans são necessárias, **no mínimo**, para isso, e em quais cruzamentos elas devem ser alocadas?

Se o problema for difícil demais de imaginar, considere que as pessoas da cidade têm suas casas localizadas em cruzamentos; a partir de onde se encontram, elas devem conseguir comprar um sorvete andando, no máximo, apenas mais um quarteirão.

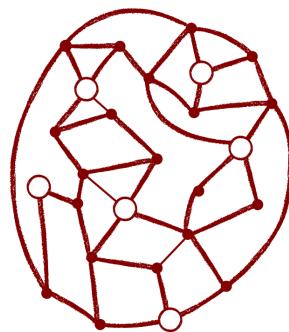
Deve ser notado que a atividade deve ser feita preferencialmente em grupos - o seu objetivo é fomentar discussões, primeiramente entre os alunos, e depois entre os professores, de como resolver problemas complexos.

Etapas da atividade:

- 1) Divida os alunos em pequenos grupos, ou até duplas ou indivíduos. Dê a cada grupo uma cópia da **Folha de Atividade: Vans de Sorvete** e algo para marcar a folha - lápis, ou algum tipo de ficha. Após isso, conte a história apresentada na Apresentação de Problema.
- 2) Mostre aos alunos como devem marcar ou preencher os cruzamentos para marcar as vans de sorvete - as pessoas que têm casas nos cruzamentos, ou ao longo das ruas localizadas entre eles, devem ser consideradas clientes das vans de sorvete. Pode-se marcar algum tipo de limite de tempo (8 minutos para atividades rápidas, até 20 minutos para discussões mais aprofundadas), e também pode-se informar que o número mínimo é de seis vans. Pode ou não prometer-se algum tipo de recompensa àqueles que acertarem o número mínimo necessário, com a resposta correta (existe apenas uma).
- 3) Incentive os alunos a experimentarem diferentes configurações de posicionamento para as vans. Enquanto eles encontram posições que abrangem todos os moradores da cidade, lembre-os de que as vans são

caras e que a ideia desse exercício é fazê-los utilizar o menor número possível de vans para resolver o problema proposto (é óbvio que as condições podem ser cumpridas se houver vans suficientes para alocar em todos os cruzamentos).

- 4) Após o término da atividade, recolha as folhas e revise as respostas. Caso tenham sido prometidas recompensas por acertos, distribua elas agora à seu critério. Apresente a solução na folha **Resolução da atividade - Vans de Sorvete**, ou desenhe a mesma em um quadro:

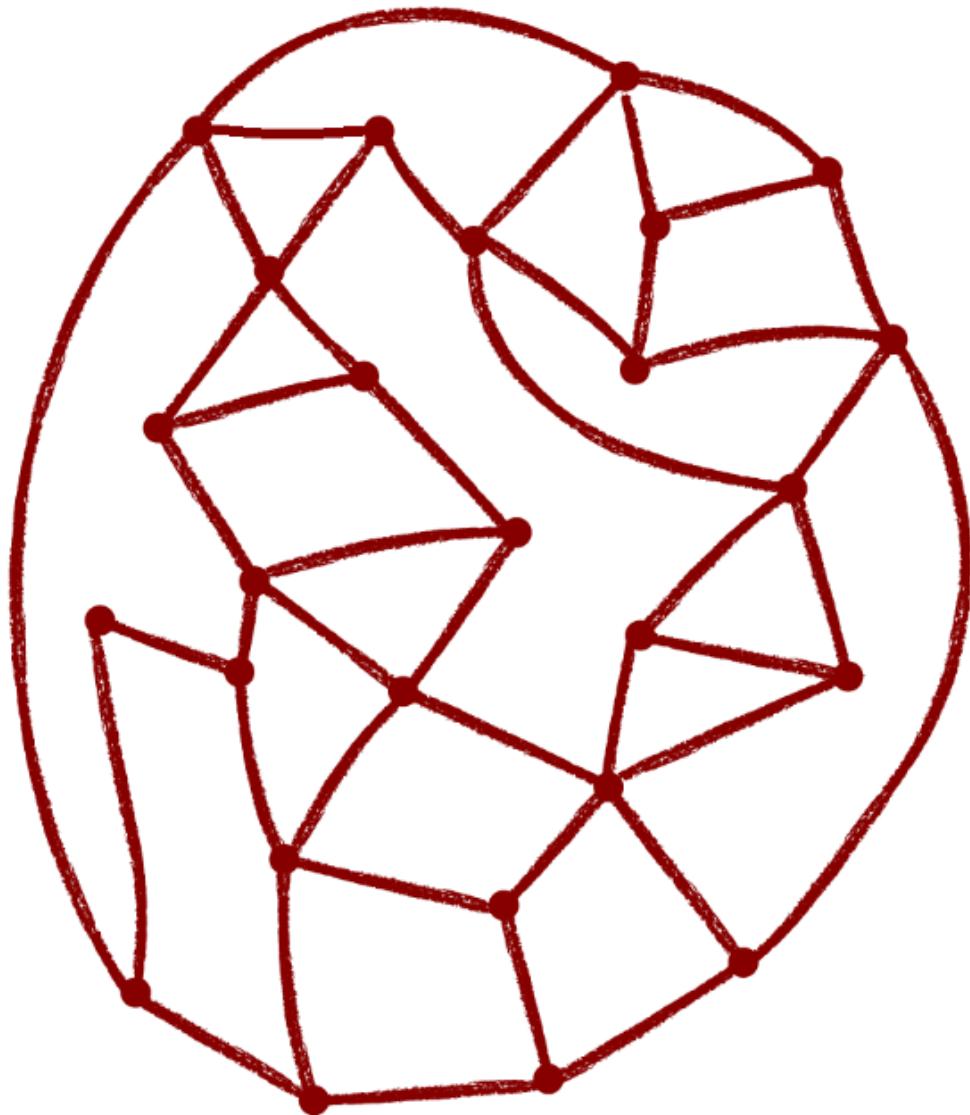


(<https://desplugada.ime.unicamp.br/atividade15/mapa1.png>)

- 5) Discuta com os alunos se eles usaram algum tipo de raciocínio para encontrar a solução deles. Observe que é ineficiente inserir conexões de rua entre os cruzamentos da resolução que contêm nós abertos (brancos), mas sim entre aqueles pontos a mais, que contêm nós sólidos (pretos), pois todos os pontos já tem pelo menos um ponto aberto adjacente.
- 6) Caso queira prosseguir mais com a atividade, incentive os alunos a elaborar seus próprios mapas, em nível de dificuldade alto, utilizando a mesma técnica apresentada a eles ao final da atividade. Eles podem testar essas criações com seus amigos e família. Dessa forma, descobrirão que podem criar enigmas que eles são capazes de resolver, mas que as outras pessoas não. Na computação, esse fenômeno é denominado “Função de mão única”. Isto é, criar um quebra-cabeça muito difícil de montar pode ser fácil, mas montar ele será complexo - a não ser que a pessoa que esteja montando seja a criadora do quebra-cabeça.

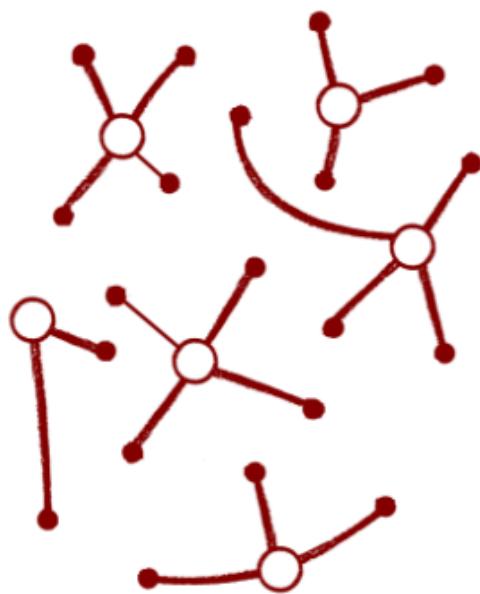
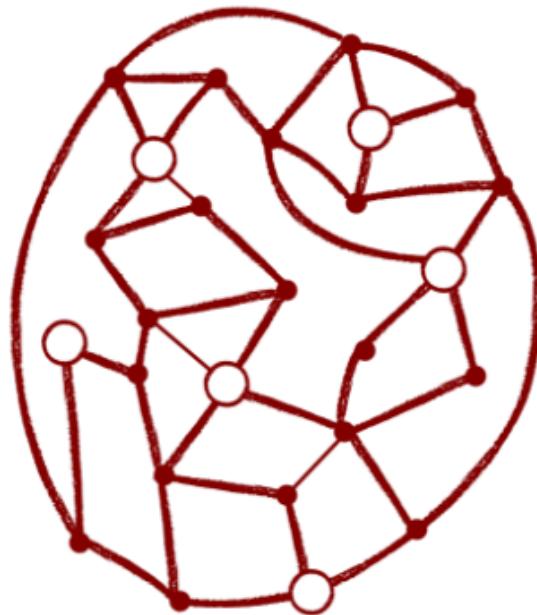
Folha de Atividade: Vans de Sorvete

Descubra como colocar as vans de sorvete nos cruzamentos das ruas a fim de que todos os outros cruzamentos estejam conectados a outro que tenha uma van posicionada.



Resolução da atividade: Vans de Sorvete

Exiba esta resolução para a turma a fim de mostrar a técnica por trás da criação do quebra-cabeça.



Fontes das imagens:

(<https://desplugada.ime.unicamp.br/atividade15/mapa1.png>)

(<https://desplugada.ime.unicamp.br/atividade15/resol.png>)

Trabalhando o Conhecimento:

Utilizando os conhecimentos adquiridos na prática durante a atividade, podemos propôr várias perguntas para maior reflexão.

- Quais tipos de situações nas quais se pode enfrentar o tipo de problema proposto na atividade anterior, especialmente no contexto do planejamento urbano?

Gabarito: Alocação de caixas postais, poços, unidades de corpo de bombeiros, etc

- Um algoritmo guloso consiste em marcar sempre o nó que tiver o maior número de arestas que o conectam aos outros, e depois marcar o ponto com o segundo maior número de arestas, e assim por diante. Crie esse algoritmo usando linguagem coloquial.

Gabarito:

Estrutura Nó:

// Armazena os dados de cada nó

```
String nome  
Lista<Nó> conexões  
Número grau  
Booleano visitado
```

Função AlgoritmoGulosoGrafo(grafo):

// Calcula quantas conexões cada nó tem (grau)

Para cada nó em grafo:

```
nó.grau = nó.conexões.tamanho  
nó.visitado = falso
```

// Ordena pelos mais conectados primeiro

OrdenarNós(grafo, por grau, decrescente)

resultado = []

// Pega sempre o mais conectado que ainda não foi visitado

Para cada nó em grafo:

Se não nó.visitado:

resultado.adicionar(nó)

nó.visitado = verdadeiro

// Marca os vizinhos como visitados (necessário para o nosso problema)

Para cada vizinho em nó.conexões:

vizinho.visitado = verdadeiro

Imprimir("Selecionando " + nó.nome + " (grau: " + nó.grau + ")")

Retornar resultado

- Agora, determine se ele serve para resolver o problema das Vans de Sorvete.

Gabarito: Não serve - o cruzamento com o maior número da cidade, que tem cinco ruas, não é um bom lugar para alocar uma van

- Temos 26 nós ou esquinas na cidade, e assim 26 formas diferentes de alocar uma van. Como há duas possibilidades diferentes em cada esquina - ou há ou não há uma van - qual o número possível de configurações de vans na cidade?

Gabarito: Como há dois estados possíveis para cada esquina, o número total de configurações possíveis é 2^{26} , ou seja, 67.108.864.

Algoritmos e Tempo Computacional:

Trabalhando os conhecimentos prévios, podemos tirar algumas conclusões perante a atividade. Primeiro, o único método que é realmente confiável para se obter a solução do problema é o algoritmo de **força bruta**, que consiste na verificação de todas as soluções possíveis. Apesar de ser possível encontrar a solução correta durante o tempo da atividade através da intuição do aluno, não há um algoritmo que nos dê uma solução garantida, que seria útil para uma máquina procedural como o computador. Porém, o próprio algoritmo de força bruta apresenta problemas sérios, que só se amplificam quanto maior o problema

que ele deve resolver.

Considere que, apenas neste problema, há 67.108.864 configurações diferentes para o mapa, desde ele com nenhuma vans até ele com vans em todos os seus 26 pontos. Para chegar à solução ótima, e se assegurar que nenhuma foi pulada sem conhecimento prévio da solução correta, um computador terá que verificar todas essas soluções uma por uma. Imagine que uma configuração inteira leve apenas um segundo para testar - o nosso computador levaria cerca de 67 milhões de segundos para verificar todas elas, e considerando que há 86.400 segundos em um dia, precisaríamos de 777 dias, ou cerca de dois anos, para chegar à uma solução certeira ao nosso problema!

Isso se torna rapidamente um problema extremamente grande quando aumentamos apenas um pouco o escopo do nosso problema - dado um computador que verifique uma solução em um milésimo de segundo, ele seria capaz de chegar à uma solução certeira em cerca de dois anos para um algoritmo com 36 cruzamentos, ou seja, 2^{36} soluções, que é aproximadamente 1000 multiplicado por 2^{26} . Note que essas cidades são bem pequenas!

Saiba mais: Onde são eficientes algoritmos, como o algoritmo guloso, na vida real?

Os algoritmos gulosos são extremamente eficientes em muitos problemas do mundo real onde precisamos tomar decisões rápidas e encontrar soluções "boas o suficiente" sem explorar todas as possibilidades.

- **Sistemas de Tempo Real** Em trading de alta frequência, os algoritmos gulosos tomam decisões de compra/venda em microssegundos.
- **Sistemas de Recomendação** Muitas plataformas usam estratégias gulosas como "sempre mostrar o item com os maiores pontos de relevância disponível".

Veja também:

- A coleção Matemática Multimídia da Unicamp sobre grafos:
<https://m3.ime.unicamp.br/recursos?filter=grafos>
- O módulo do Portal da OBMEP sobre grafos:
<https://portaldaobmep.impa.br/index.php/modulo/ver?modulo=84>

Referência:

Atividade de computação desplugada número 15 da Unicamp:
<https://desplugada.ime.unicamp.br/atividade15/index.html>

Unidade 5

Como a internet é organizada



[\(https://www.hardware.com.br/artigos/internet-tem-dono-conheca-a-organizacao-invisivel-que-controla-toda-a-internet/\)](https://www.hardware.com.br/artigos/internet-tem-dono-conheca-a-organizacao-invisivel-que-controla-toda-a-internet/)

Capítulos:

- Capítulo 1: Introdução a topologias de redes (EF08C006)
- Capítulo 2: Cama de Gato para representar topologias (EF08C006)

Capítulo 1: Introdução a topologias de redes

Introdução:

Topologias de rede são as diferentes maneiras de conectar dispositivos em uma rede de computadores, definindo a estrutura física e lógica dos seus componentes. Em outras palavras, a topologia descreve como os nós (computadores, roteadores, switches) e links (cabeamento, conexões sem fio) estão organizados para que a comunicação entre eles ocorra.

Importância das topologias:

A escolha da topologia de rede adequada depende dos requisitos de cada rede, como o tamanho da rede, a necessidade de redundância, a facilidade de manutenção e o custo. Uma topologia bem planejada pode otimizar o desempenho da rede, melhorar a segurança e facilitar a expansão futura.

Dispositivos de Rede:

Existem alguns dispositivos necessários para que as redes de computadores funcionem normalmente, vamos classificá-los pelas suas funções em quatro grupos e dar exemplos.

Dispositivos de conexão:

Roteador: Conecta redes diferentes (ex: Wi-Fi doméstico à internet).



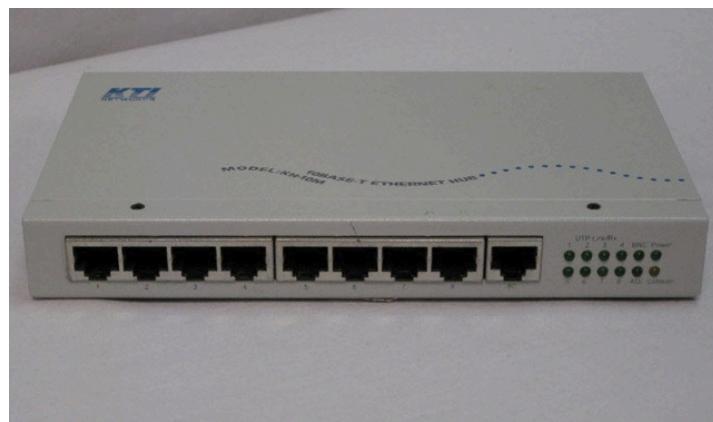
(<https://amplinet.commercesuite.com.br/informatica/roteador/roteador-wi-fi-alta-potencia-4-antenas-5g-remotize-ipv6>)

Switch: Distribui dados apenas para os dispositivos de uma mesma rede.



[\(https://www.oficinadosbits.com.br/produto/switch-24-portas-tp-link-tl-sg1024d-gigabit/\)](https://www.oficinadosbits.com.br/produto/switch-24-portas-tp-link-tl-sg1024d-gigabit/)

Hub (está obsoleto): Envia dados para todos os dispositivos (ineficiente).



[\(https://pplware.sapo.pt/tutoriais/networking/redes-como-funciona-um-hub/\)](https://pplware.sapo.pt/tutoriais/networking/redes-como-funciona-um-hub/)

Dispositivos de transmissão:

Modem: Converte sinais digitais em analógicos (ex: modem de internet a cabo).



(<https://www.astemar.com.br/produtos/p.asp?id=1741&produto=modem-para-fibra-optica-onu-fiberhome-an5506-02b-plus>)

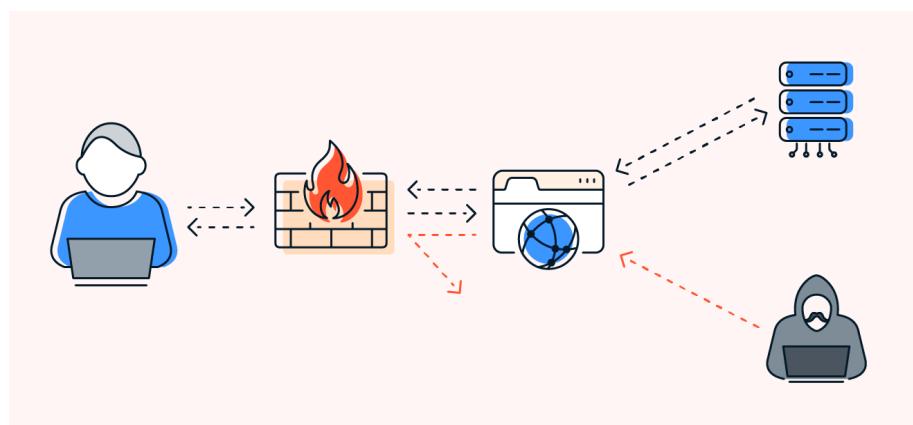
Repetidor: Amplifica o sinal para alcançar distâncias maiores.



(<https://www.zoom.com.br/modem-e-roteador/wi-fi-sem-fio-repetidor-wifi-alcance-extensor-300mbps-de-rede-wi-fi-amplificador-sinal-impulsionador>)

Dispositivos de segurança:

Firewall: Filtra acessos não autorizados à rede.



(<https://www.avast.com/pt-br/c-what-is-a-firewall>)

Dispositivos de acesso:

Placa de rede: Permite que um dispositivo (ex: computador, celular, notebook) se conecte à rede.



<https://www.waz.com.br/placa-de-rede-4x-gigabit-pci-e-intel-vt-quad-port-server-adapter-expi9404vt-113883-html/p>

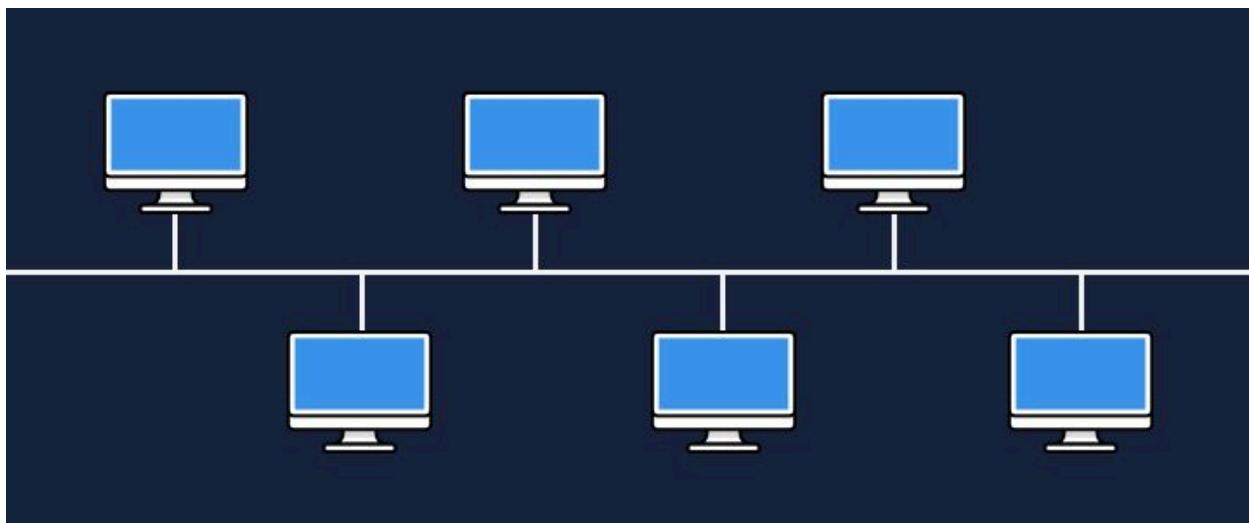
Access Point: Estende o sinal Wi-Fi em uma área.



<https://www.oficinadosbits.com.br/produto/access-point-corporativo-tp-link-eap670-ax5400-wi-fi-6-poe-4804mbps-tecnologia-mu-mimo-omada-sdn-montavel-em-teto-ou-parede/>

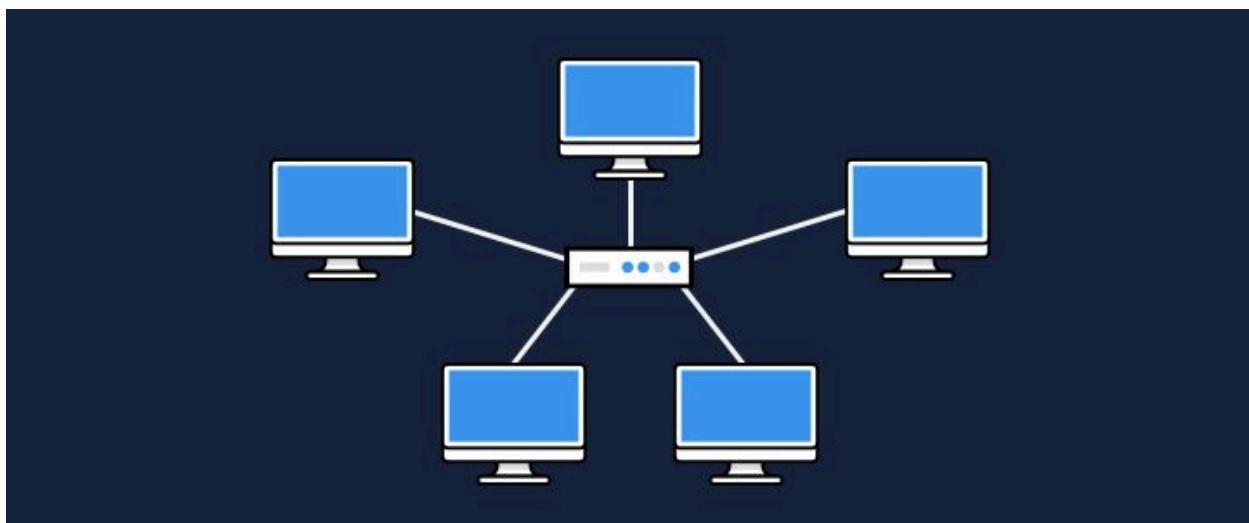
Exemplos de topologias:

- Topologia em Barramento: Todos os dispositivos são conectados a um único cabo central, como uma linha de transmissão.



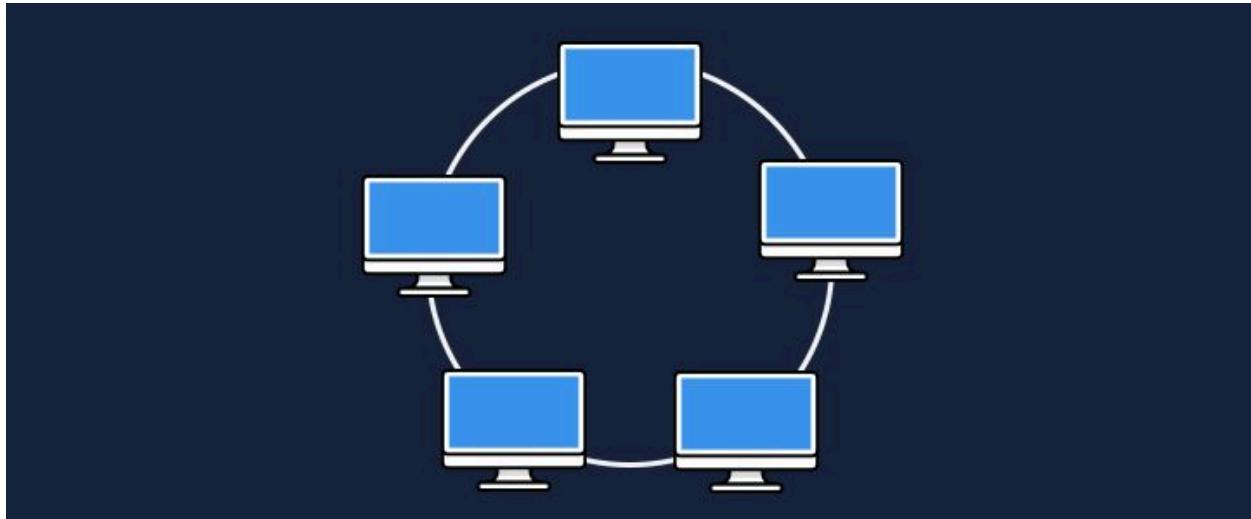
[\(<https://www.gerenciatec.com.br/topologia-de-rede/#seis>\)](https://www.gerenciatec.com.br/topologia-de-rede/#seis)

- Topologia Estrela: Cada dispositivo se conecta a um dispositivo central (hub ou switch), criando uma estrutura em forma de estrela.



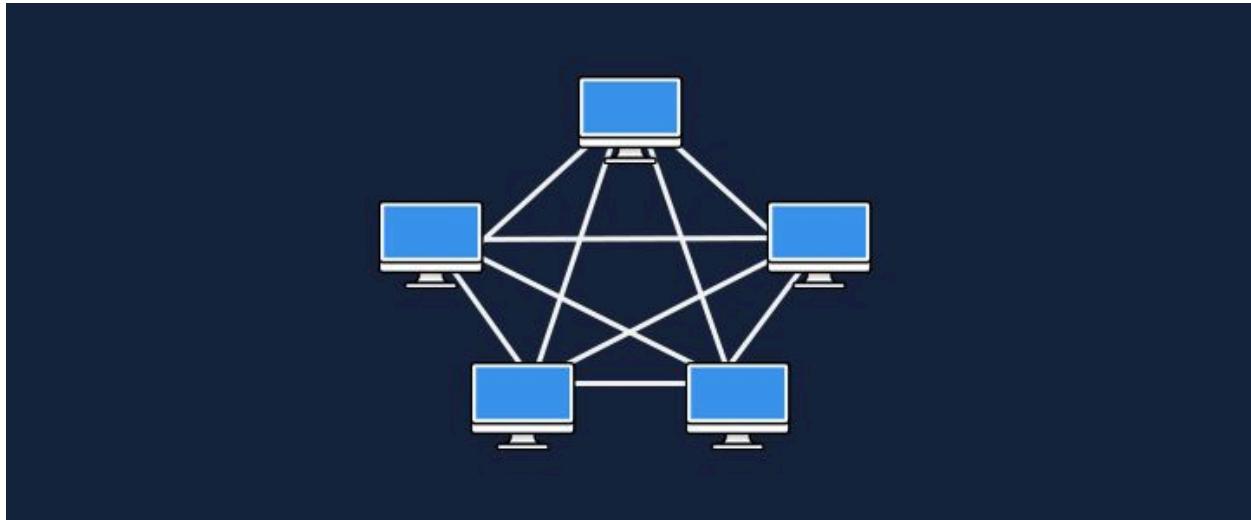
[\(<https://www.gerenciatec.com.br/topologia-de-rede/#seis>\)](https://www.gerenciatec.com.br/topologia-de-rede/#seis)

- Topologia Anel: Os dispositivos são conectados em um círculo, formando uma cadeia.



(<https://www.gerenciatec.com.br/topologia-de-rede/#seis>)

- Topologia em Malha: Cada dispositivo tem uma conexão direta com vários outros dispositivos, criando uma rede complexa e redundante.



(<https://www.gerenciatec.com.br/topologia-de-rede/#seis>)

Capítulo 2: Cama de Gato para representar topologias

O que é a Cama de Gato?

A Cama de Gato (também chamada de "Jogo do Fio") é uma brincadeira tradicional de origem indígena, em que os participantes criam padrões entrelaçados com um barbante esticado entre os dedos. Dois ou mais jogadores cooperam para formar figuras como "rede", "escada" ou "diamante", passando o fio de uma mão para outra sem desfazer o desenho.

Materiais Utilizados:

- Barbantes coloridos: Basta cortar fios de aproximadamente 1 metro de comprimento e unir as pontas. Eles funcionam como se fossem os cabos ligando os "nós" ou "dispositivos";
- Site explicando a brincadeira Cama de Gato.
Link: <https://pt.wikihow.com/Jogar-Cama-de-Gato>
- Vídeo explicando a brincadeira Cama de Gato.
Link: <https://youtu.be/qaxEdtco9JU?si=E8XMCzPQqNXMQtj9> (título: Vivo Brincar: Tutorial de Cama de Gato)
- Mão dos participantes: Os dedos funcionam como "nós" ou "dispositivos" da rede.



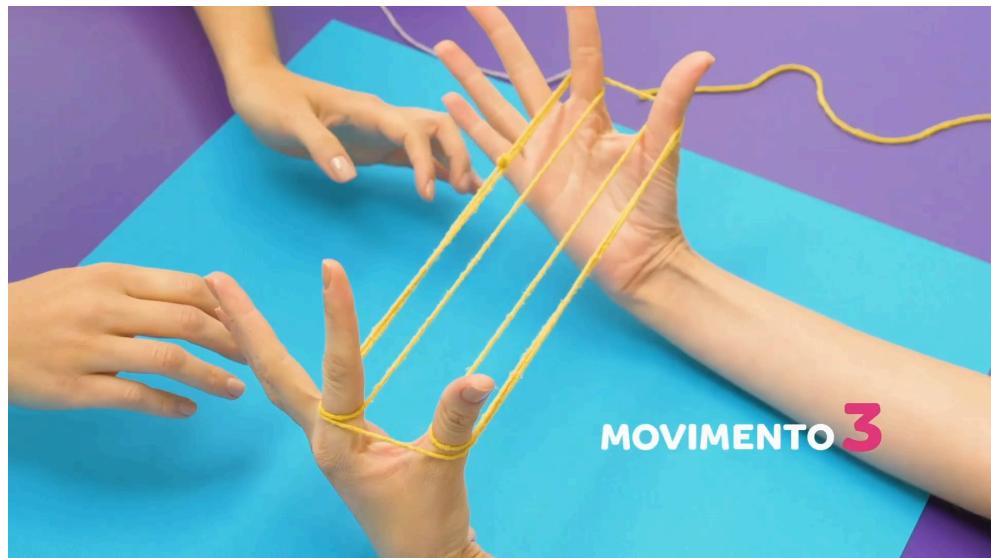
(<https://youtu.be/qaxEdtco9JU?si=E8XMCzPQqNXMQtj9>)

O objetivo é tentar representar as topologias (anel, barramento, estrela e malha) estudadas no capítulo anterior através da brincadeira.

Anel:

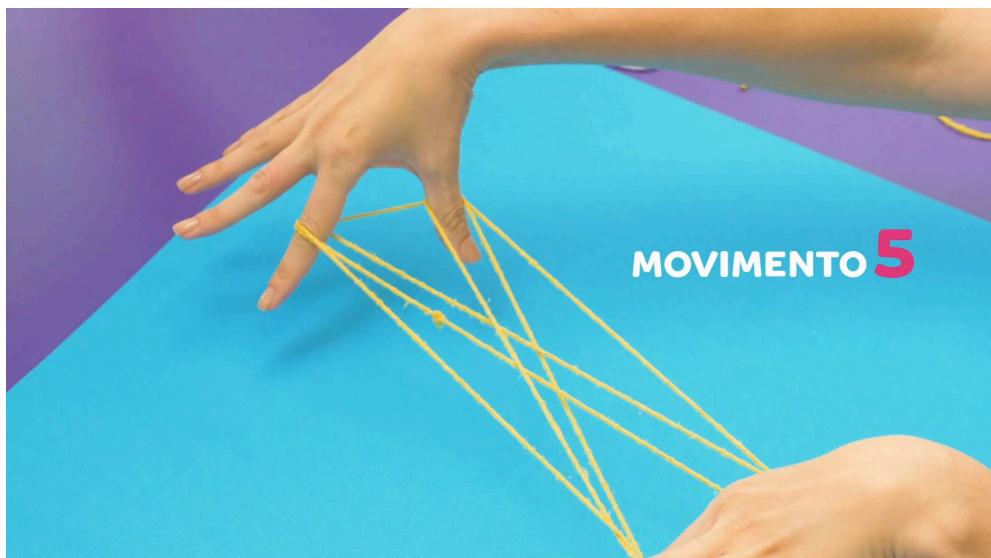
Basta fazer um círculo com o barbante ao redor das mãos.

Barramento:



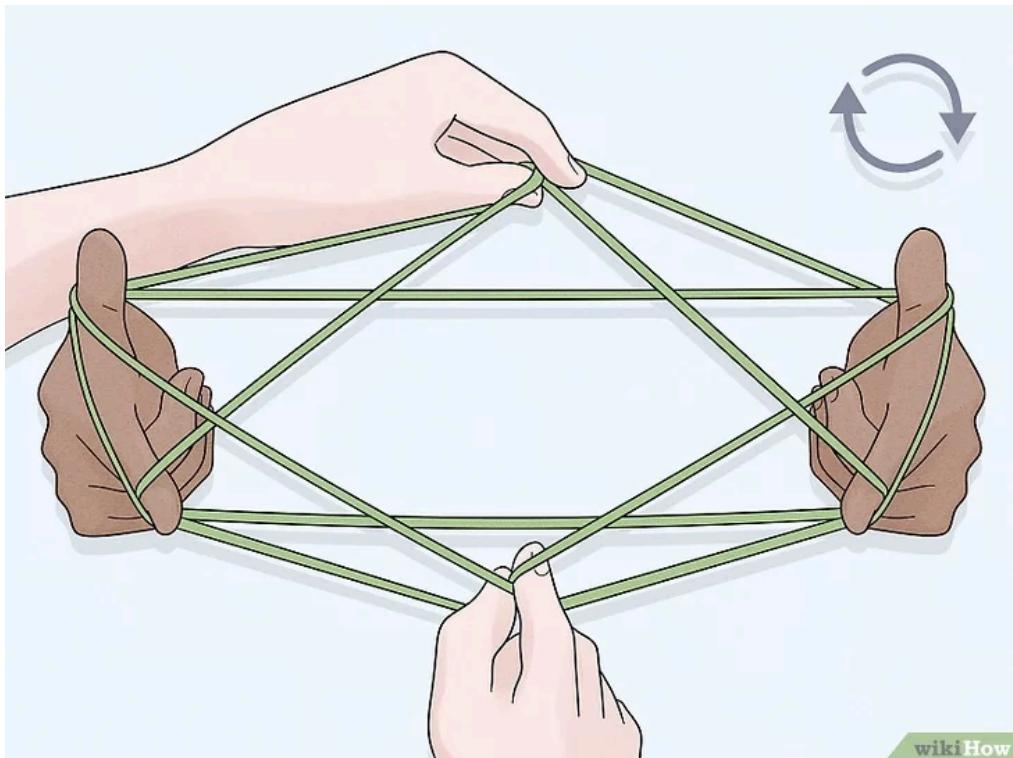
(<https://youtu.be/qaxEdtco9JU?si=E8XMCzPQqNXMQtj9>)

Estrela:



(<https://youtu.be/qaxEdtco9JU?si=E8XMCzPQqNXMQtj9>)

Malha:



(<https://pt.wikihow.com/Jogar-Cama-de-Gato>)

Saiba mais: Onde estão presentes as topologias de rede?

As topologias de rede estão presentes em diversos ambientes, desde redes domésticas até grandes redes corporativas, e podem ser físicas ou lógicas. Elas descrevem como os dispositivos (nós) estão interconectados e como a informação flui entre eles.

Topologia física x Topologia lógica:

Topologia Física:

É a disposição concreta dos cabos e dispositivos (ex.: estrela, anel, barramento).

Mostra como tudo está fisicamente conectado.

Topologia Lógica:

Define como os dados trafegam na rede (ex.: Ethernet, Token Ring).

Pode ser diferente da física (ex.: rede física em estrela pode funcionar como barramento lógico).

Diferença chave:

Física = hardware (cabeamento, layout).

Lógica = software/protocolos (fluxo de dados).

Veja também:

- Ressignificando a brincadeira de origem indígena “Cama de Gato” para o ensino de computação na educação básica
Disponível em: <https://sol.sbc.org.br/index.php/sbceb/article/view/28677/28481>