Table of Contents

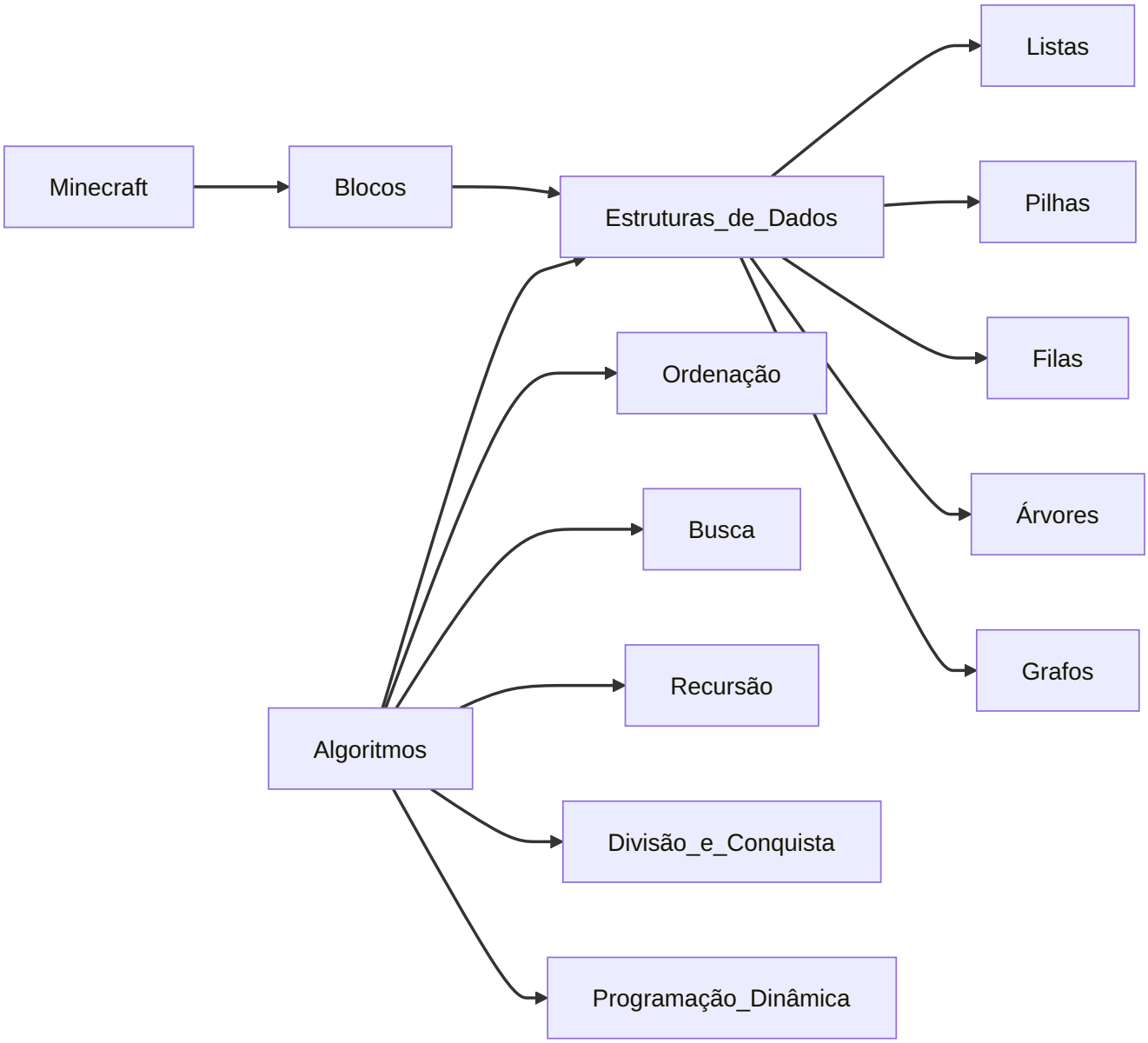
Learn Data Structure

Introdução: Minecraft e Estruturas de Dados

Vamos explorar o mundo de Minecraft como um jogo de blocos, onde cada bloco pode representar diferentes elementos de dados. Nesta seção, vamos aprender sobre as estruturas de dados e algoritmos, utilizando a analogia do Minecraft para facilitar a compreensão.

Imagine que você está construindo uma casa com blocos de madeira, pedra, e argila. Cada tipo de bloco tem uma função específica na estrutura da casa. Da mesma forma, as estruturas de dados e algoritmos são como as regras e técnicas que nos ajudam a organizar e manipular dados em nossos computadores, programas, e mundos virtuais.

Aqui está um mapa mental Mermaid que ilustra a relação entre Minecraft, estruturas de dados, e algoritmos:



Neste mapa mental, você pode ver como os blocos de Minecraft estão relacionados às estruturas de dados, como listas, pilhas, filas, árvores, e grafos. Além disso, os algoritmos, como ordenação, busca, recursão, divisão e conquista, programação dinâmica, e estruturas de dados, trabalham juntos para manipular e organizar os dados em nossos jogos e aplicações.

Vamos começar a explorar as estruturas de dados e algoritmos utilizando a analogia do Minecraft!

Listas

Uma lista é como uma caixa de ferramentas onde você pode armazenar vários blocos de uma só vez. Cada bloco tem uma posição específica na lista, e você pode acessá-los por essa posição.

No Minecraft, você pode construir uma lista de blocos de madeira, pedra, e argila para fazer uma estrutura de muro. Da mesma forma, em programação, você pode usar listas para armazenar dados de diferentes tipos, como números, strings, e objetos.

Aqui está um exemplo de como você pode declarar e usar uma lista em Python:

```
# Declarando uma lista vazia
blocos = []

# Adicionando blocos à lista
blocos.append("madeira")
blocos.append("pedra")
blocos.append("argila")

# Acessando um bloco específico na lista
primeiro_bloco = blocos[0]

# Imprimindo a lista e o primeiro bloco
print(blocos)
print(primeiro_bloco)
```

Continuando

Com isso DS (Structure) é um caminho para armazenar e organizar os dados seguindo um modelo lógico e sistematico para a construção de estruturas de dados.

Pense assim, quando andamos na rua e olhamos um ser com uma esquizoide jeito de se vestir e existir e não sabemos definir qual o sexo, ou seja, se é masculino ou feminino temos então o que chamamos de **dados abstratos** ou de se compararmos com o audio/video que é emitido pelo tal ser temos uma visualização abstrata.

Esses dados abstratos chamamos na computação de **tipos de dados abstratos** (abstract data types) que estão no mesmo cesto de modelos lógicos/matematicos quando falamos de Estrutura de Dados.

Abstract Data Type vs. Concrete Implementation

Abstract Data Type (vulgo: ADT para os intimos e famigerados cheiradores de pó-pólenta) é a abstração de um modelo de tipo de dado, que vai definir os pontos chaves para construir determinada estrutura de dados baseado naquele modelo

Abstraindo a abstracão

Transformando em palavras mais gostosas de ouvir, todo ser humano precisa de uma, logo você me pergunta:

```
Mas o que raios isso tem a ver com o assunto Punk, se tá biruleibe das ideias?
```

Calma meu bom acefalo, com celebro lisinho, toda casa (pelo menos como conhecemos sendo uma boa casa) para ser uma casa ela precisa de algumas coisas:

- Fundação
- Paredes
- Colunas
- Porta
- Janela

Isso é uma abstração de uma casa correto? Isso é de suma o essencial que uma casa precisa ter, esse é o modelo lógico que o que estou querendo construir deve ter, mas isso **NÃO É PROPRIAMENTE** uma casa, isso é uma abstração de uma casa

```
Logo o que temos é um ADT então Punk?
```

[Respondo com um meme do Kirby]

Kirby yep yep yep

Agora pense que você é um empresário e vai abrir um novo negócio que é uma casa noturna (podemos chamar de bordel caso queira) em, *Xique-Xique Bahia* 📍, e o que ele vai construir é uma casa, certo?

cafetaomaciota.png

Sim, então ele vai seguir um ADT um modelo abstrado de tipo de dado que define a abstração, que no nosso caso esse ADT seria uma casa.

Logo ao fazer a construção do bordel você vai precisar seguir a abstração do modelo que exige:

- Fundação
- Paredes
- Colunas
- Porta
- Janela

Claro que nesse modelo de negócio que você vai construir vai exigir mais coisas.

```
Então o bordel que foi feito é a implementação concreta (concrete implementation) de um ADT?
```

Sim, está correto meu carro frequentador de la casa delas primas

> ⚠ Claro que não precisamos nos fecharmos apenas ao que modelo nos entrega, mas para a forma concreta ser de determinado ADT ela precisa implementar as questões do modelo

Falando com programação

Quando vamos para programação o nosso ADT poderia ser uma **Lista** onde ela tem:

- Armazena elementos de diferentes tipos de dados
- Ler um elemento por uma posição
- Modifica um elemento numa determinada posição

Agora nossa implementação concreta seria um **Array** que implementa a abstração de nosso modelo ADT que é a **Lista**

Com isso vamos entender as Data Structures.

LinkedList

Start typing here...

List (Lista)

Uma lista é um Tipo Abstrato de Dados (ADT) que:

- Permite armazenar elementos de qualquer tipo de dado
- Permite inserir um elemento em uma posição específica
- Permite ler um elemento de uma posição específica
- Permite remover um elemento de uma posição específica
- Permite atualizar um elemento em uma posição específica

Uma implementação concreta deste ADT é um Array, onde:

- Os elementos do array podem ser de qualquer tipo de dado
- Todas as operações mencionadas acima são possíveis

Exemplo de declaração e uso de um array em C:

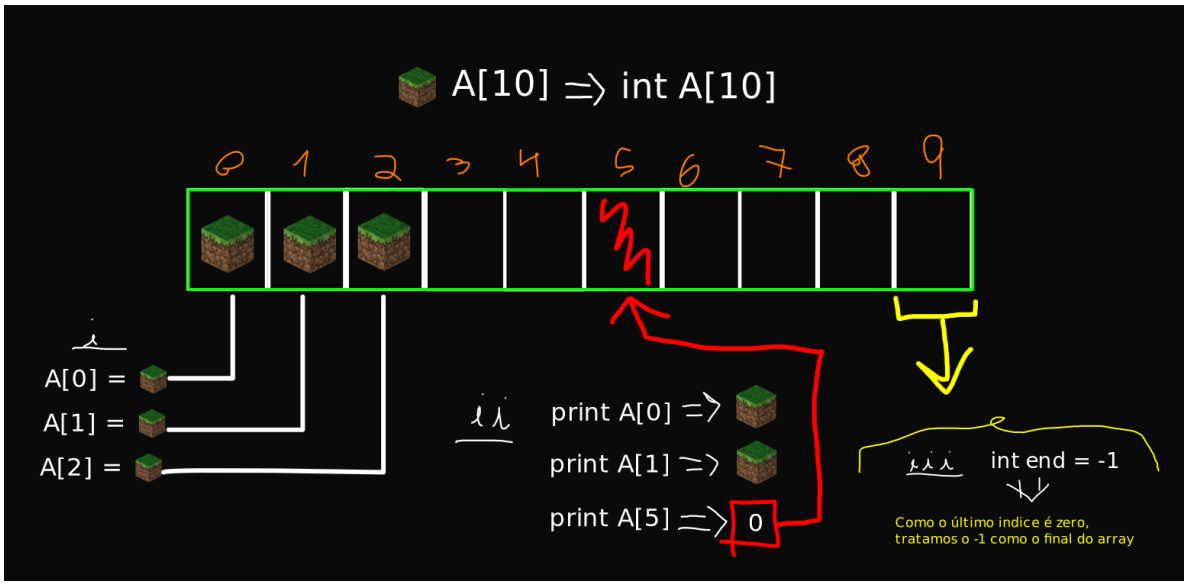
```
int A[10];  // Declara um array de inteiros com 10 elementos

A[0] = 10;  // Atribui o valor 10 ao primeiro elemento do array

printf("%d", A[0]);  // Imprime o valor do primeiro elemento do
array
```

As funcionalidades básicas de uma lista podem ser resumidas como:

```
List
|__ Criação de uma lista vazia (tamanho inicial 0)
|__ Inserção de elementos
|__ Remoção de elementos
|__ Contagem de elementos
```

```
|__ Leitura/modificação de elementos em posições específicas
|__ Especificação do tipo de dado dos elementos
```

Estas operações podem ser visualizadas na seguinte imagem:



Exemplificação de operações de uma lista

Nota: Quando um array está cheio, uma implementação típica cria um novo array maior, copia os elementos do array antigo para o novo, e libera a memória do array antigo.

Complexidade das operações principais:

1. **Acesso**: Ler/Escrever um elemento em um índice específico → tempo constante, Big O notation: O(1)
2. **Inserção**: O tempo é proporcional à quantidade de elementos → O(n)
3. **Remoção**: O tempo é proporcional à quantidade de elementos → O(n)
4. **Adição**: O tempo é proporcional à quantidade de elementos → O(n)