Aula 4 - Brute force + Backtracking 01

PrepTech Google

Roteiro da Aula

1 - **Introdução**
    Apresentação e conceitos

2 - **Problemas**
    Exemplos

3 - **Prática**
    Questões do LeetCode

1 - Introdução

Apresentação e conceitos

Introdução

- **O que é brute force?**
- **O que é backtracking?**
- **Aplicações**
- **Estrutura de um algoritmo**
- **Vantagens e desvantagens**

Introdução

O que é brute force?

- Define uma **estratégia** ou **paradigma** de algoritmos onde tenta-se **TODAS** as soluções **possíveis** até achar uma que atenda (ou todas), independente de sua eficiência

Exemplos

- **Bogosort**: ordenação utilizando randomização - O(n!)
- **Quebra de senhas**: O(k^n), para k possíveis caracteres e senhas de tamanho n
- **Busca por cadeias de caracteres em texto**: O(n x m)
- Teste da **primalidade** de um número inteiro: O(√n)
- **Caixeiro viajante**: O(n!)

Brute force: Aspectos

- Busca **exaustiva** pela(s) solução(ões)
- **Simples** de implementar, mas normalmente **ineficiente** para problemas de espaço grande: escalabilidade e desempenho ruins
- Complexidade normalmente exponencial ou superior: O(n!), O(2^n), O(n^k), etc.
- Normalmente aplicável para problemas de **espaço pequeno** onde não há uma solução melhor

Brute force: Exemplo 1 - Contador Binário (4 bits) - utilizando iterações

Brute force: Exemplo 1 - Contador Binário (4 bits) - utilizando iterações

```
0000
```

Estado inicial

Brute force: Exemplo 1 - Contador Binário (4 bits) - utilizando iterações

```
0000
0001
```

Variações na 1a casa

Brute force: Exemplo 1 - Contador Binário (4 bits) - utilizando iterações

```
0000
0001
0010
0011
```

Variações na 2a casa

Brute force: Exemplo 1 - Contador Binário (4 bits) - utilizando iterações

```
0000
0001
0010
0011
0100
0101
0110
0111
```

Variações na 3a casa

Brute force: Exemplo 1 - Contador Binário (4 bits) - utilizando iterações

```
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
```

Variações na 4a (última) casa

Implementem o contador binário de 4 bits usando iterações.

Brute force: Exemplo 1 - Contador Binário (4 bits) - utilizando iterações

main.py

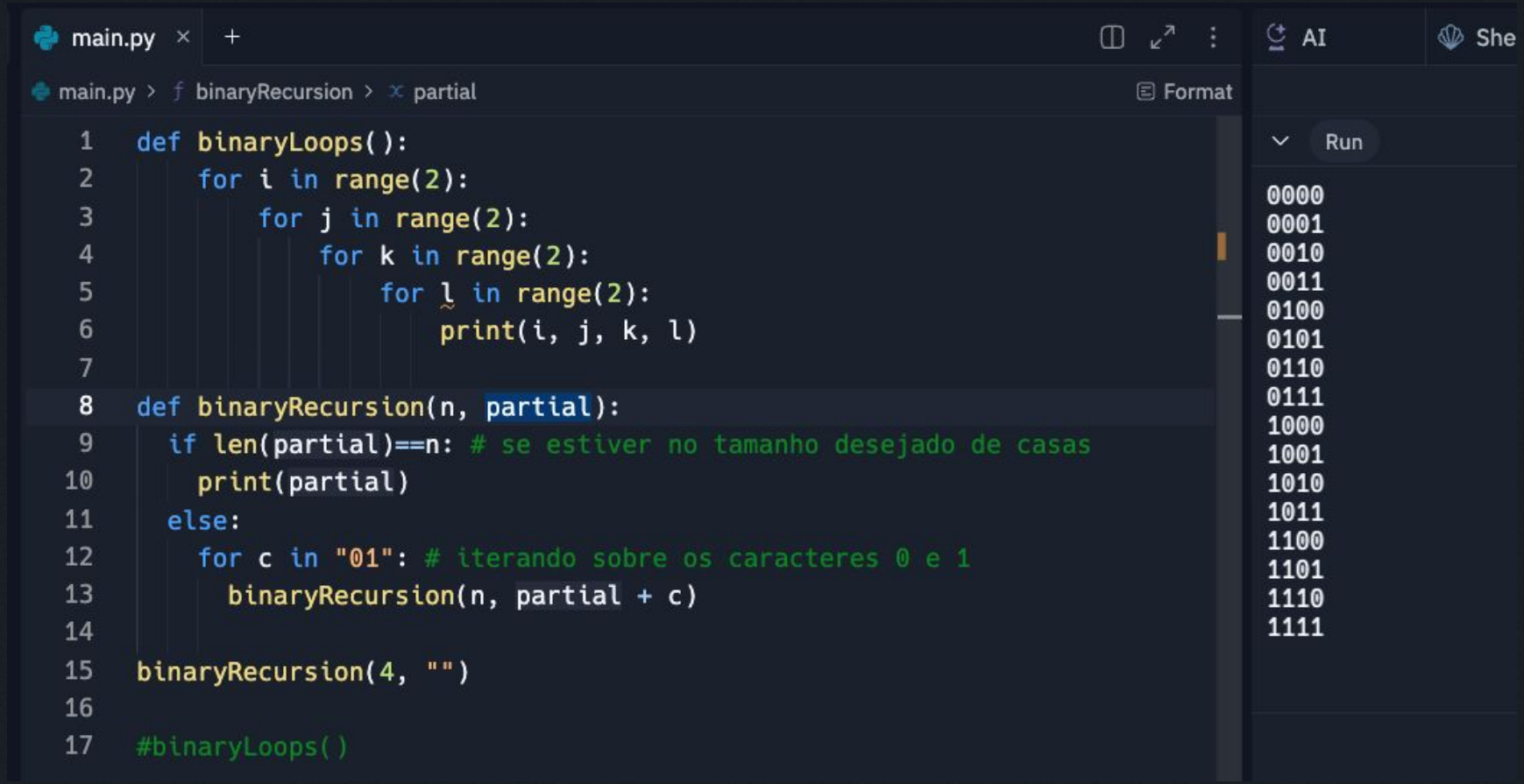
```python
def binaryLoops():
    for i in range(2):
        for j in range(2):
            for k in range(2):
                for l in range(2):
                    print(i, j, k, l)

binaryLoops()
```

```
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
1 0 0 1
1 0 1 0
1 0 1 1
1 1 0 0
1 1 0 1
1 1 1 0
1 1 1 1
```

Qual o problema dessa solução?

Brute force: Exemplo 1 - Contador Binário (4 bits) - utilizando recursão

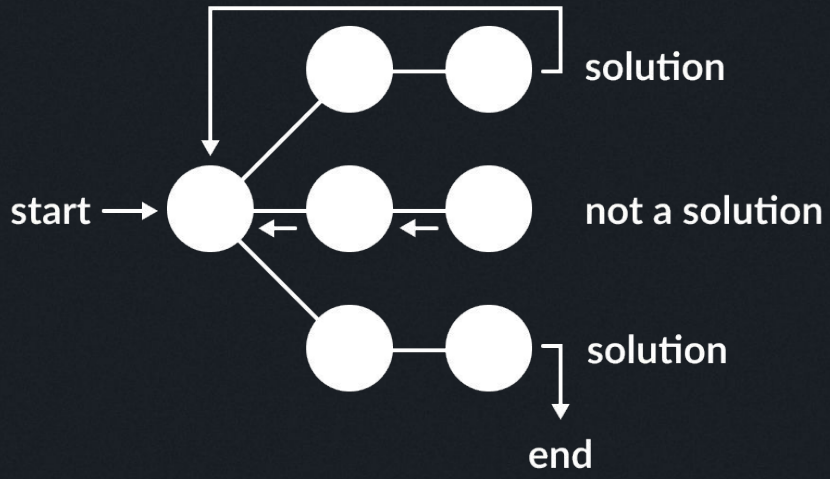
Brute force: Exemplo 1 - Contador Binário (4 bits) - utilizando recursão



Backtracking: O que é backtracking?

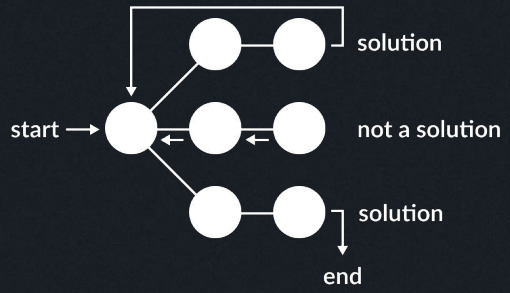
- Define uma estratégia ou paradigma de projeto de algoritmos onde tenta-se construir uma solução ou um conjunto de soluções de uma forma incremental (por partes), removendo soluções que não satisfazem as restrições do problema

- Processo de tentativa e corte (*pruning*)



Backtracking: conceitos importantes

- **Recursão**: divisão e conquista (pedaços menores)
- **Espaço de soluções**: conjunto total de soluções **possíveis**
- **Corte (*pruning*)**: soluções **parciais** que não são válidas
- **Backtracking**: retorno de um passo quando este não apresenta uma solução ou não atende às restrições
  - **in**: explora soluções internas a partir da atual
  - **out**: volta de uma tentativa interna e segue com novas tentativas



Backtracking: estrutura de um algoritmo

- **Passo 1:** iniciar com uma solução vazia
- **Passo 2:** gerar uma solução parcial e verificar se ela atende às restrições
- **Passo 3:** Se não atender, descarta, volta (backtrack) e tenta outra solução
- **Passo 4:** Repetir os passos acima até que uma solução seja aceita ou que TODAS as possibilidades tenham sido geradas de forma exaustiva

Backtracking: exemplos de problemas/algoritmos

- **Sudoku Solver**
- **Problemas combinatórios**
- **Subset sum (soma de subconjuntos)**
- **Passeio do cavaleiro/rainha/cavalo em tabuleiros de xadrez**

Aplicações de brute force e backtracking

- **Problemas de satisfação de restrições: sudoku, quebra-cabeças, desafios, labirintos, etc**
- **Busca por combinações e/ou permutações**
- **Problemas em grafos: busca por um caminho, coloração de nós, circuito hamiltoniano**

Vantagens e Desvantagens

Vantagens:

- **Simplicidade e clareza no uso da recursão :-)**
- **Permite reduzir o tamanho do problema/entrada através de cortes (*pruning*)**

Desvantagens:

- **Em casos onde o espaço do problema é grande, a estratégia é altamente ineficiente: n!, n^n, 2^n, etc**
- **Pior caso: complexidade exponencial em relação ao tempo**

Problemas

https://leetcode.com/problems/subsets/description/

Problema 1: Subsets de sets

- **Subsets de sets: como montar todos os subconjuntos deste conjunto (permutação)**

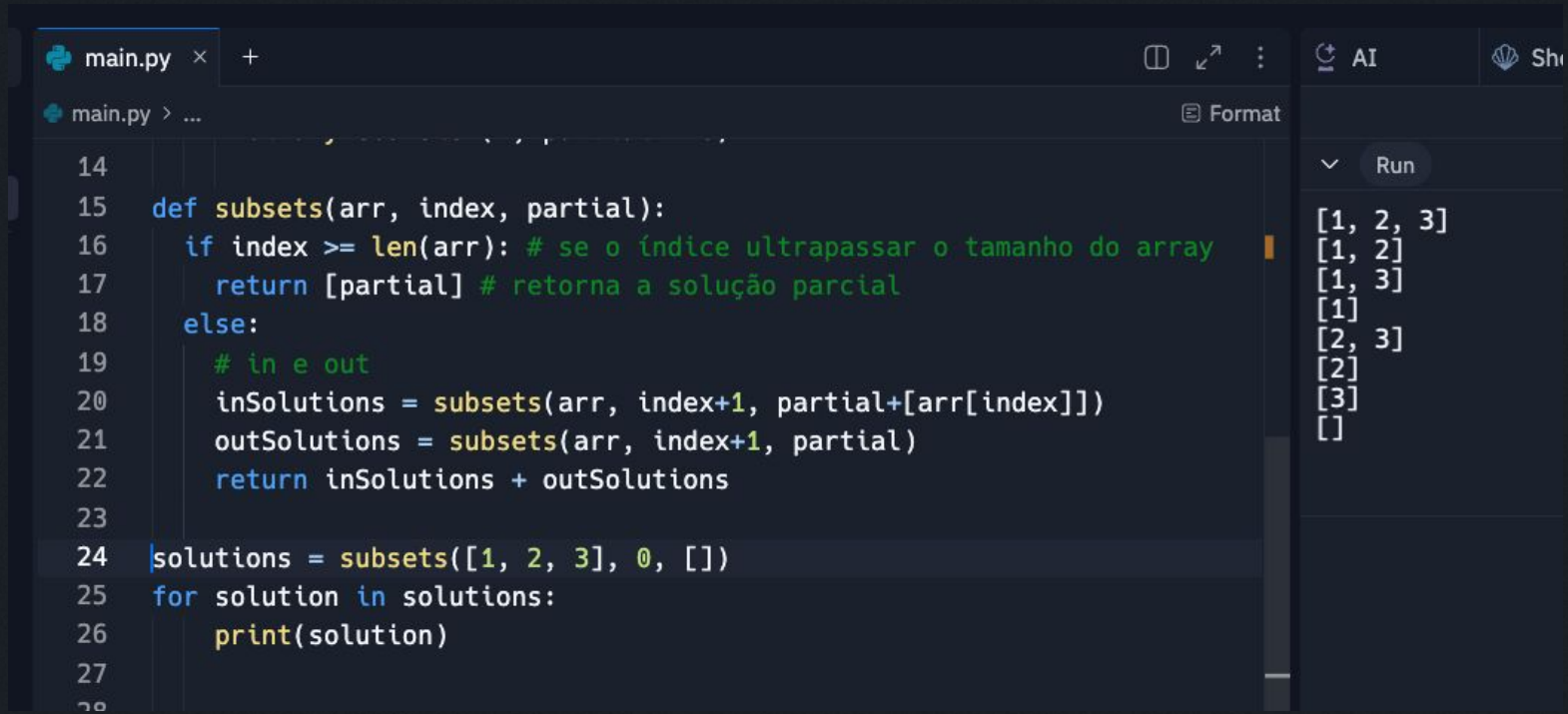
Ex:

- **Entrada: [1, 2, 3]**
- **Saída:**
  - **[1, 2, 3], [1, 2], [1, 3], [1], [2, 3], [2], [3], []**

Problemas
https://leetcode.com/problems/subsets/description/

Problema 1: Subsets de sets

- **Subsets de sets: como montar todos os subconjuntos deste conjunto (permutação)**



Problemas
https://leetcode.com/problems/subsets

Problema 1: Subsets de sets

- **Subsets de sets: como montar (permutação)**

main.py

```python
def subsets(arr, index, partial):
    if index >= len(arr): # se o índice ultrapassar o tamanho do array
        return [partial] # retorna a solução parcial
    else:
        # in e out
        inSolutions = subsets(arr, index+1, partial+[arr[index]])
        outSolutions = subsets(arr, index+1, partial)
        return inSolutions + outSolutions

solutions = subsets([1, 2, 3], 0, [])
for solution in solutions:
    print(solution)
```

	IT	arr	index	partial	inSolutions	outSolutions	return
start	0	[1, 2, 3]	0	[]			
in	1	[1, 2, 3]	1	[1]			
in	1.1	[1, 2, 3]	2	[1, 2]			
in	1.1.1	[1, 2, 3]	3	[1, 2, 3]	[1, 2, 3]		
out	1.1.2	[1, 2, 3]	3	[1, 2]		[1, 2]	[[1, 2, 3], [1, 2]]
out	1.2	[1, 2, 3]	2	[1]		[1]	[[1, 2, 3], [1, 2], [1]]
out	2	[1, 2, 3]	1	[]			
in	2.1	[1, 2, 3]	2	[2]			
in	2.1.1	[1, 2, 3]	3	[2, 3]	[2, 3]		
out	2.1.2	[1, 2, 3]	3	[2]		[2]	[[1, 2, 3], [1, 2], [1], [2, 3], [2]]
out	2.2	[1, 2, 3]	2	[]			
in	2.2.1	[1, 2, 3]	3	[3]	[3]		
out	2.2.2	[1, 2, 3]	3	[]		[]	[[1, 2, 3], [1, 2], [1], [2, 3], [2], [3], []]

ada

Mais problemas

- https://leetcode.com/problems/permutations/
- https://leetcode.com/problems/combinations/
- https://leetcode.com/problems/find-array-given-subset-sums/

Mais Problemas - Vide slide de Recursão (:

Obrigado