Uma imagem com texto

Descrição gerada automaticamente

Universidade de Coimbra

Faculdade de Ciências e Tecnologia

Departamento de Engenharia Informática

Integração de Sistemas

**Trabalho prático Nº1**

Diogo Jordão Filipe - uc2018288391@student.uc.pt

José Miguel Silva Gomes - uc2018286225@student.uc.pt

Coimbra, 8 de outubro de 2021

Índice

[1 Introdução 3](#_Toc84515879)

[1.1 JSON 3](#_Toc84515880)

[1.2 MessagePack 3](#_Toc84515881)

[2 Ambiente da experiência 3](#_Toc84515882)

[2.1 Estruturas de dados 4](#_Toc84515883)

[2.2 Condições experimentais 5](#_Toc84515884)

[3 Código fonte 6](#_Toc84515885)

[3.1 Geração de dados 6](#_Toc84515886)

[3.2 Serialização e desserialização 6](#_Toc84515887)

[3.3 Medição do tempo 7](#_Toc84515888)

[4 Resultados experimentais 8](#_Toc84515889)

[5 Conclusão 14](#_Toc84515890)

# Introdução

A representação de informação digital é algo importante no mundo da informática e, como tal, existem diversas maneiras de a representar, nomeadamente em formato de texto e formato binário.

Este documento visa comparar e explorar as diferenças entre dois formatos de representação de informação, JSON e MessagePack, de acordo com alguns parâmetros: complexidade de programação, tamanho de serialização, e velocidades de serialização e desserialização.

## JSON

O JSON é um formato simples e rápido de serialização de dados para texto, derivado do JavaScript, que funciona via pares de nomes/valores e utiliza linguagem legível por humanos. É frequentemente usado para enviar dados do servidor ao cliente em aplicações web.

## MessagePack

O MessagePack é um formato eficiente de serialização binária, que também permite a transmissão de dados entre sistemas como o JSON. Este formato codifica inteiros de pequenas dimensões para apenas um byte, e texto, também de pequenas dimensões, é codificado com um byte mais o tamanho inicial.

# Ambiente da experiência

Nesta secção descreveremos as condições do ambiente da experiência realizada, como as estruturas de dados e ferramentas utilizadas, e as especificações dos computadores que executaram o código e recolheram os dados.

## Estruturas de dados

De forma a proceder a uma comparação entre estes dois formatos de representação de dados, desenvolvemos uma estrutura *Pet-Owner* com *many-to-one relationship*, que é composta por duas entidades *Pet* e *Owner*, que contém atributos básicos como identificadores, nomes, datas de nascimento, etc.

Foram então as criadas as duas classes da seguinte forma:

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **Class** Owner: | | |
| **2** |  | **def** \_init\_\_(id, name, birth, phone, address) | |
| **3** |  |  | self.id = id |
| **4** |  |  | self.name = name |
| **5** |  |  | self.birth = birth |
| **6** |  |  | self.phone = phone |
| **7** |  |  | self.address = address |

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **Class** Pet: | | |
| **2** |  | **def** \_init\_\_(id, name, species, gender, weight, birth, description, owner) | |
| **3** |  |  | self.id = id |
| **4** |  |  | self.name = name |
| **5** |  |  | self. species = species |
| **6** |  |  | self. gender = gender |
| **7** |  |  | self. weight = weight |
| **8** |  |  | self. birth = birth |
| **9** |  |  | self. description = description |
| **10** |  |  | self. owner = owner |

Estas classes foram criadas tendo em conta o enunciado disponibilizado e, como se trata de uma relação *many-to-one*, cada objeto *Pet* tem um identificador para o seu *Owner* de forma a poder ser identificado.

## Condições experimentais

IDE: *PyCharm* 2021

Linguagem de Programação: Python 3.9

Especificações do Computador de José Miguel

Processador: 2,6 GHz Intel Core i7

Placa Gráfica: AMD Radeon Pro 5300M 4 GB

Memória: 16 GB 2667 MHz DDR4

Disco: 512GB SSD

Sistema Operativo: MacOS BigSur

Especificações do Computador de Diogo Filipe

Processador: Intel Core i7-7500U CPU 2.70-2.90 GHz

Placa Gráfica: AMD Radeon R7 M340

Memória: 8 GB

Disco: Samsung SSD 860 EVO 500GB

Sistema Operativo: Windows 10 Home

# Código fonte

Nesta secção iremos explicar as diferentes partes do código desenvolvido, como a geração de dados, a serialização e desserialização dos dados, e a medição do tempo das mesmas.

Para reduzir o impacto de componentes aleatórias da performance dos computadores, as operações a estudar e a sua cronometração foram parametrizadas com o número de repetições a serem feitas.

## Geração de dados

Para automatizar e parametrizar a geração de dados foram implementadas 3 funções: "gen\_owners", "gen\_pets" e "gen\_data".

---------codigo--------

A partir da chamada da função "gen\_data" com o número desejado de objetos de cada classe a serem gerados ("n"), são chamadas as funções "gen\_owners" e "gen\_pets", que os geram respetivamente, com parâmetros simplificados e por vezes iguais, visto que o foco da experiência é a comparação dos dois formatos e não dos dados em si.

## Serialização e desserialização

JSON:

Para realizar a serialização dos dados com o formato JSON foi importada a biblioteca "json", e usadas as suas funções de serialização e desserialização.

Primeiramente, foi necessário criar uma classe para substituir a que mapeia a codificação dos tipos de objetos da função de serialização "dump", pois o tipo de objetos a ser utilizado não é serializável.

---código encoder---

Enviando então esta classe como parâmetro "cls" na função "dump" é possível serializar os objetos pois estes são transformados em dicionários do Python antes da serialização.

Para armazenar os dados serializados é criado um ficheiro "json\_serialized.txt" e enviado o seu ponteiro como parâmetro na função "dump".

---chamada função dump---

Para a desserialização, apenas é chamada a função "load" com o ponteiro do ficheiro mencionado acima como parâmetro.

---chamada função load---

MessagePack:

O código desenvolvido para o formato MessagePack foi bastante semelhante ao do JSON, mas com a biblioteca "msgpack".

Para contornar a impossibilidade de serializar os objetos com a função "pack", foi criada a função "encoder\_msgpack" e enviada como parâmetro "default" para a função "pack", cumprindo o mesmo fim que a classe "Encoder" criada para o JSON.

---codigo encoder\_msgpack---

Também é criado um ficheiro para guardar os dados serializados, "msgpack\_serialized.txt", e enviado o seu ponteiro como parâmetro na função "pack".

---chamada funcao pack---

A desserialização é igual à do JSON mas com a função "unpack".

---chamada funcao unpack---

## Medição do tempo

Foram importadas duas bibliotecas, a "time" para medir o tempo, e a "statistics" para calcular a média dos tempos e o seu desvio padrão.

Em ambos os formatos usou-se o mesmo código para realizar o mencionado. Primeiramente criam-se dois ficheiros, um para guardar os tempos de serialização ("json\_time\_s.txt"/" msgpack\_time\_s.txt"), e outro para os de desserialização ("json\_time\_d.txt"/" msgpack\_time\_d.txt "). É então usada a função "perf\_counter", e guardado o seu valor, antes e depois da serialização e desserialização, e a sua diferença (depois - antes) corresponde ao tempo em segundos que demorou a operação.

---código de uma medição do tempo---

A cada execução é escrito no ficheiro correspondente o tempo demorado, e guardado numa lista. No fim da última execução, temos duas listas (uma da serialização e outra da desserialização) com os tempos obtidos, e é usada a função "mean" para obter a média de cada lista, e a função "pstdev" para calcular o desvio padrão. Estes dois últimos valores também são escritos no ficheiro correspondente.

# Resultados experimentais

De modo a estabelecer uma comparação entre os dois formatos de dados foram realizadas experiências tendo por base os seguintes princípios:

* Número de dados gerados;
* Média do tempo ocorrido no processo de serialização/desserialização;
* Desvio padrão;
* Tamanho do ficheiro gerado após o processo de serialização para os dois formatos;

Figura 1 - Tabela com os dados gerados pelo computador de José Miguel

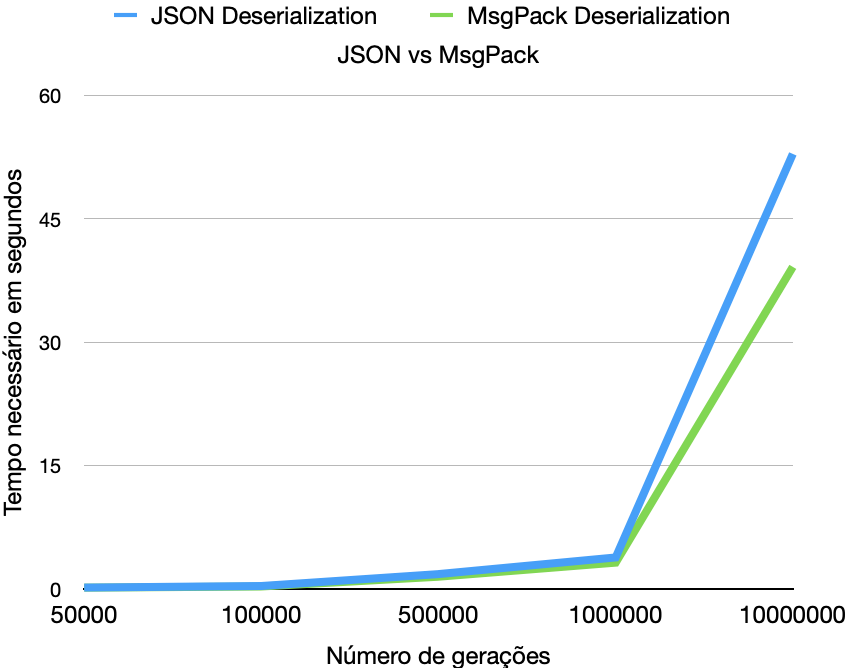
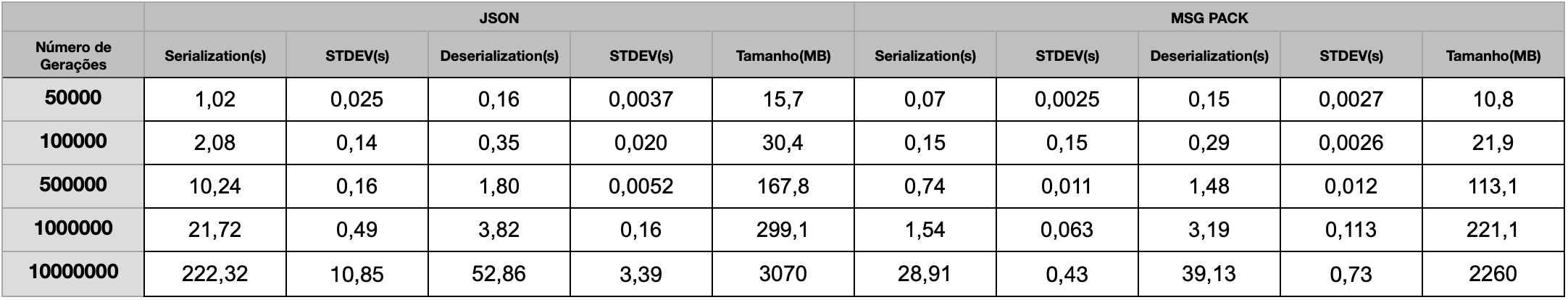


Figura 2 - Diferença entre o tempo de desserialização do formato JSON e MsgPack

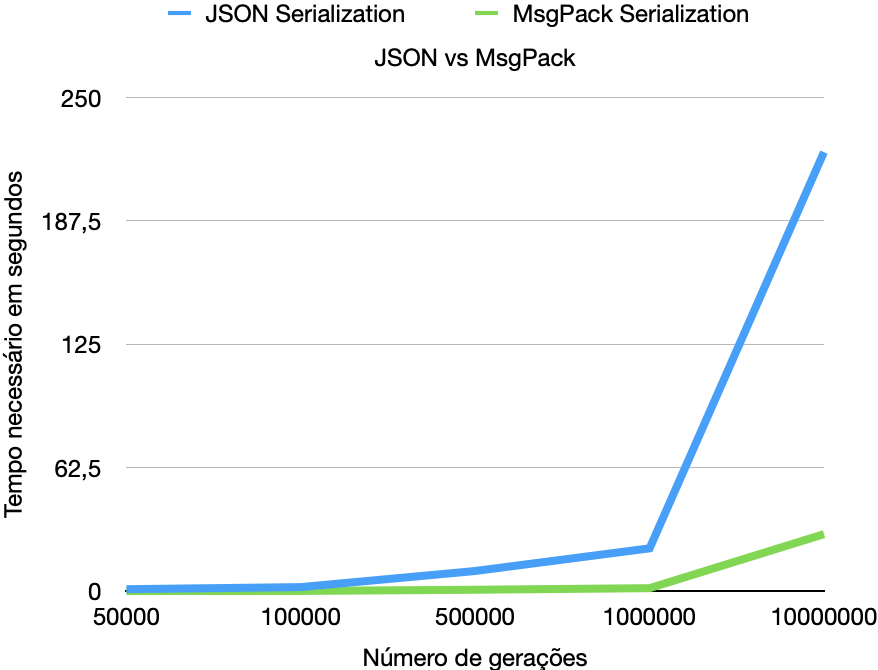


Figura 3 - Diferença entre o tempo de serialização do formato JSON e MsgPack

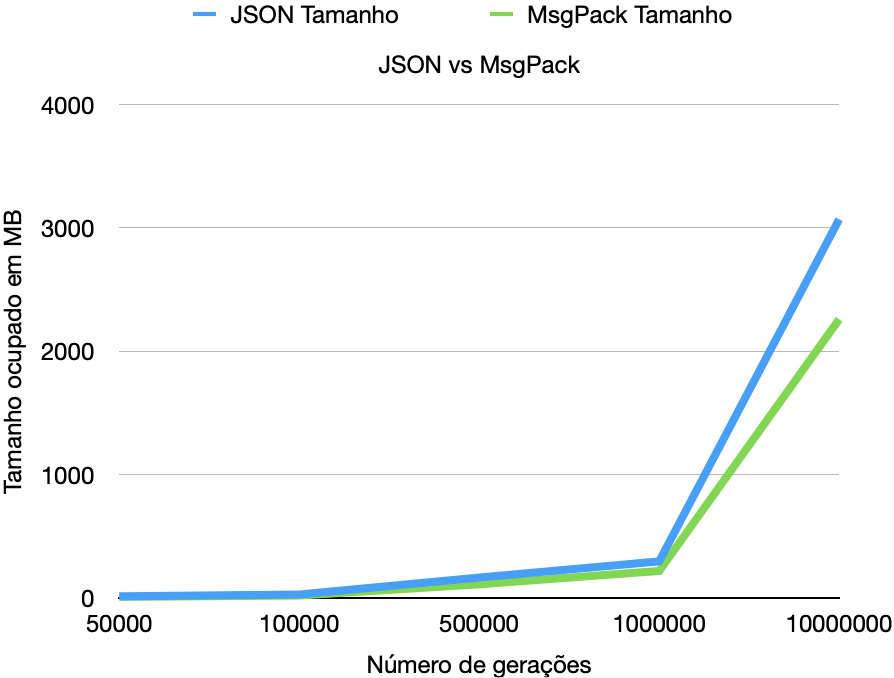


Figura 4 - Diferença do tamanho ocupado em disco pelos ficheiros serializados no formato JSON e MsgPack

Uma imagem com texto

Descrição gerada automaticamente

Figura 5 - Tabela com os dados gerados pelo computador Diogo Filipe

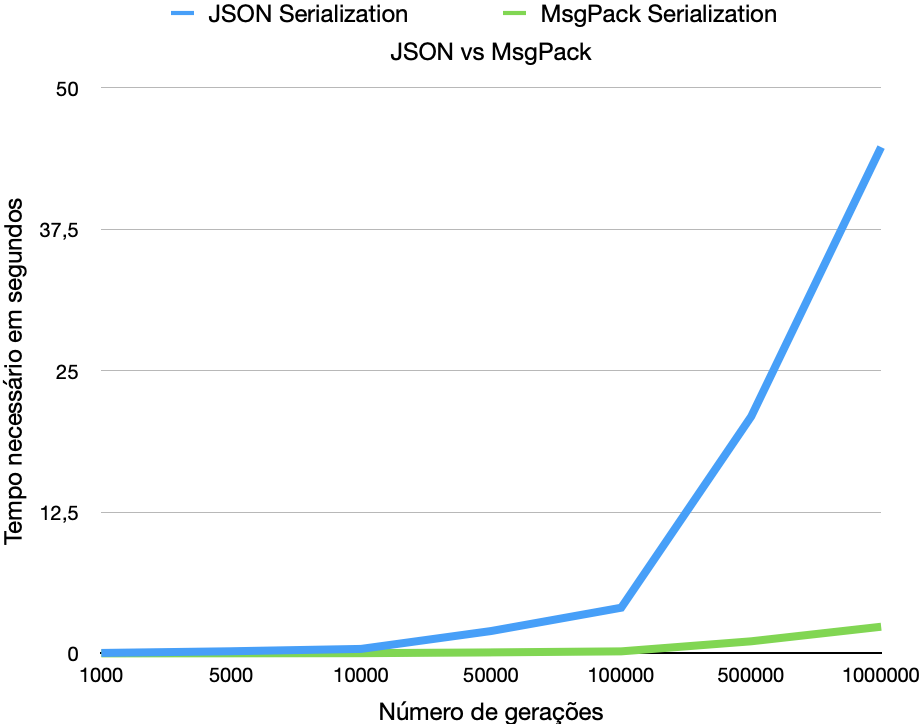


Figura 7 - Diferença entre o tempo de serialização do formato JSON e MsgPack

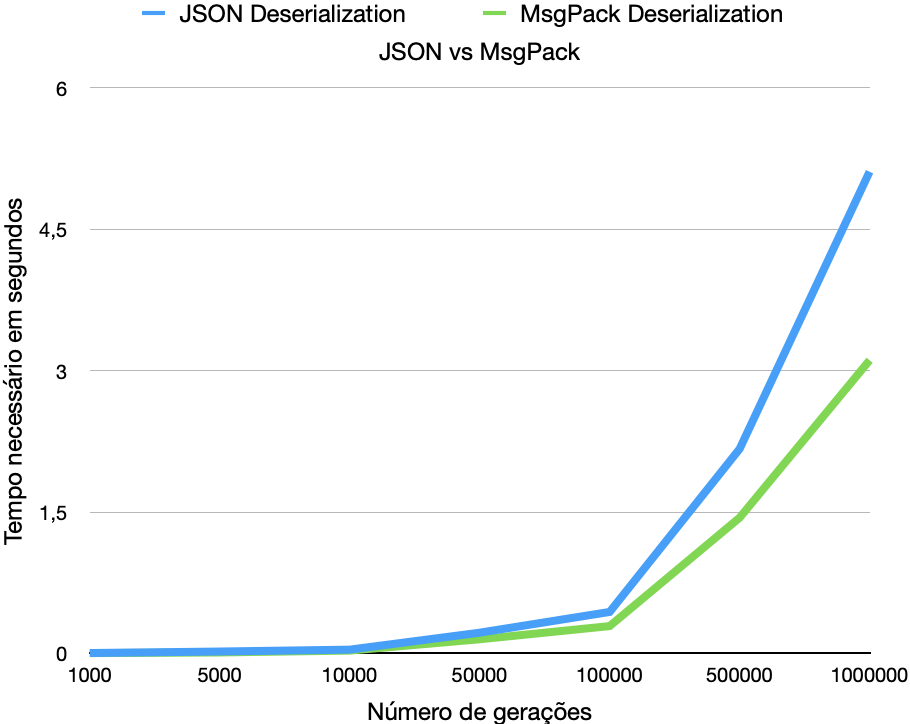


Figura 6 - Diferença entre o tempo de desserialização do formato JSON e MsgPack

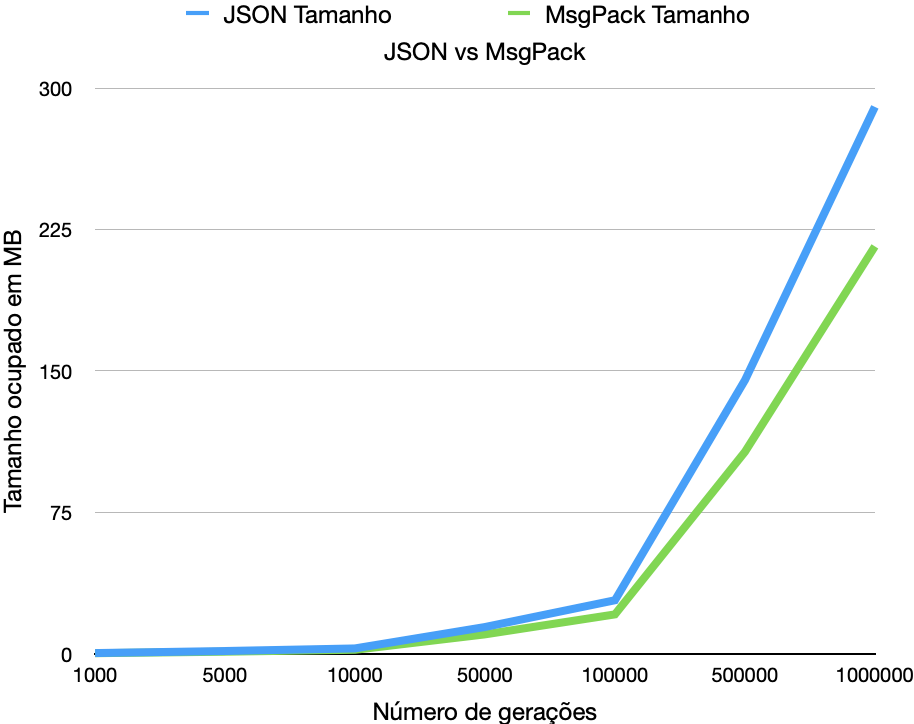


Figura 8 - Diferença do tamanho ocupado em disco pelos ficheiros serializados no formato JSON e MsgPack

# Conclusão