Big Data Coursework - Questions

Data Processing and Machine Learning in the Cloud

This is the **INM432 Big Data coursework 2024**. This coursework contains extended elements of **theory** and **practice**, mainly around parallelisation of tasks withSpark and a bit about parallel training using TensorFlow.

Code and Report

Your tasks parallelization of tasks in PySpark, extension, evaluation, and theoretical reflection. Please complete and submit the **coding tasks** in a copy of **this notebook**. Write your code in the **indicated cells** and **include** the **output** in the submitted notebook.
Make sure that **your code contains comments** on its **stucture** and explanations of its **purpose**.

Provide also a **report** with the **textual answers in a separate document**.
Include **screenshots** from the Google Cloud web interface (don't use the SCREENSHOT function that Google provides, but take a picture of the graphs you see for the VMs) and result tables, as well as written text about the analysis.

Submission

Download and submit **your version of this notebook** as an **.ipynb** file and also submit a **shareable link** to your notebook on Colab in your report (created with the Colab 'Share' function) (**and don't change the online version after submission**).

Further, provide your **report as a PDF document**. **State the number of words** in the document at the end. The report should **not have more than 2000 words**.

Please also submit **a PDF of your Jupyter notebook**.

Introduction and Description

This coursework focuses on parallelisation and scalability in the cloud with Spark and TesorFlow/Keras. We start with code based on **lessons 3 and 4** of the *Fast and Lean Data Science* course by Martin Gorner. The course is based on Tensorflow for data processing and MachineLearning. Tensorflow's data processing approach is somewhat similar to that of Spark, but you don't need to study Tensorflow, just make sure you understand the high-level structure.
What we will do here is **parallelising pre-processing**, and **measuring** performance, and we will perform **evaluation** and **analysis** on the cloud performance, as well as **theoretical discussion**.

This coursework contains **3 sections**.

Section 0

This section just contains some necessary code for setting up the environment. It has no tasks for you (but do read the code and comments).

Section 1

Section 1 is about preprocessing a set of image files. We will work with a public dataset "Flowers" (3600 images, 5 classes). This is not a vast dataset, but it keeps the tasks more manageable for development and you can scale up later, if you like.

In **'Getting Started'** we will work through the data preprocessing code from *Fast and Lean Data Science* which uses TensorFlow's `tf.data` package. There is no task for you here, but you will need to re-use some of this code later.

In **Task 1** you will **parallelise the data preprocessing in Spark**, using Google Cloud (GC) Dataproc. This involves adapting the code from 'Getting Started' to use Spark and running it in the cloud.

Section 2

In **Section 2** we are going to **measure the speed of reading data** in the cloud. In **Task 2** we will **paralellize the measuring** of different configurations **using Spark**.

Section 3

This section is about the theoretical discussion, based on one paper, in **Task 3**. The answers should be given in the PDF report.

General points

For **all coding tasks**, take the **time of the operations** and for the cloud operations, get performance **information from the web interfaces** for your reporting and analysis.

The **tasks** are **mostly independent** of each other. The later tasks can mostly be addressed without needing the solution to the earlier ones.

Section 0: Set-up

As usual, you need to run the **imports and authentication every time you work with this notebook**. Use the **local Spark** installation for development before you send jobs to the cloud.

Read through this section once and **fill in the project ID the first time**, then you can just step straight throught this at the beginning of each session - except for the two authentication cells.

Imports

We import some **packages that will be needed throughout**. For the **code that runs in the cloud**, we will need **separate import sections** that will need to be partly different from the one below.

In [ ]:
```
import os, sys, math
import numpy as np
import scipy as sp
import scipy.stats
import time
import datetime
import string
import random
from matplotlib import pyplot as plt
import tensorflow as tf
print("Tensorflow version " + tf.__version__)
import pickle
```

```
Tensorflow version 2.15.0
```

Cloud and Drive authentication

This is for **authenticating with with GCS Google Drive**, so that we can create and use our own buckets and access Dataproc and AI-Platform.

This section **starts with the two interactive authentications**.

First, we mount Google Drive for persistent local storage and create a directory `DB-CW` thay you can use for this work. Then we'll set up the cloud environment, including a storage bucket.

In [ ]:
```
print('Mounting google drive...')
from google.colab import drive
drive.mount('/content/drive')
%cd "/content/drive/MyDrive"
!mkdir BD-CW
%cd "/content/drive/MyDrive/BD-CW"
```

```
Mounting google drive...
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", for
ce_remount=True).
/content/drive/MyDrive
mkdir: cannot create directory ‘BD-CW’: File exists
/content/drive/MyDrive/BD-CW
```

Next, we authenticate with the GCS to enable access to Dataproc and AI-Platform.

In [ ]:
```
import sys
if 'google.colab' in sys.modules:
    from google.colab import auth
    auth.authenticate_user()
```

It is useful to **create a new Google Cloud project** for this coursework. You can do this on the GC Console page by clicking on the entry at the top, right of the *Google Cloud Platform* and choosing *New Project*. **Copy** the **generated project ID** to the next cell. Also **enable billing** and the **Compute, Storage and Dataproc** APIs like we did during the labs.

We also specify the **default project and region**. The REGION should be `us-central1` as that seems to be the only one that reliably works with the free credit. This way we don't have to specify this information every time we access the cloud.

In [ ]:
```
PROJECT = 'daring-healer-421511'  ### USE YOUR GOOGLE CLOUD PROJECT ID HERE. ###
!gcloud config set project $PROJECT
REGION = 'us-west1'
CLUSTER = '{}-cluster'.format(PROJECT)
```

```
!gcloud config set compute/region $REGION
!gcloud config set dataproc/region $REGION

!gcloud config list # show some information
```

```
Updated property [core/project].
WARNING: Property validation for compute/region was skipped.
Updated property [compute/region].
Updated property [dataproc/region].
[component_manager]
disable_update_check = True
[compute]
region = us-west1
[core]
account = Nicholas.Tsioras@city.ac.uk
project = daring-healer-421511
[dataproc]
region = us-west1

Your active configuration is: [default]
```

With the cell below, we **create a storage bucket** that we will use later for **global storage**. If the bucket exists you will see a "ServiceException: 409 ...", which does not cause any problems. **You must create your own bucket to have write access.**

In [ ]:
```
BUCKET = 'gs://{}-storage'.format(PROJECT)
!gsutil mb $BUCKET
```

```
Creating gs://daring-healer-421511-storage/...
ServiceException: 409 A Cloud Storage bucket named 'daring-healer-421511-storage' already exists. Try another n
ame. Bucket names must be globally unique across all Google Cloud projects, including those outside of your org
anization.
```

The cell below just **defines some routines for displaying images** that will be **used later**. You can see the code by double-clicking, but you don't need to study this.

In [ ]:
```
#@title Utility functions for image display **[RUN THIS TO ACTIVATE]** { display-mode: "form" }
def display_9_images_from_dataset(dataset):
  plt.figure(figsize=(13,13))
  subplot=331
  for i, (image, label) in enumerate(dataset):
    plt.subplot(subplot)
    plt.axis('off')
    plt.imshow(image.numpy().astype(np.uint8))
    plt.title(str(label.numpy()), fontsize=16)
    # plt.title(label.numpy().decode(), fontsize=16)
    subplot += 1
    if i==8:
      break
  plt.tight_layout()
  plt.subplots_adjust(wspace=0.1, hspace=0.1)
  plt.show()

def display_training_curves(training, validation, title, subplot):
  if subplot%10==1: # set up the subplots on the first call
    plt.subplots(figsize=(10,10), facecolor='#F0F0F0')
    plt.tight_layout()
  ax = plt.subplot(subplot)
  ax.set_facecolor('#F8F8F8')
  ax.plot(training)
  ax.plot(validation)
  ax.set_title('model '+ title)
  ax.set_ylabel(title)
  ax.set_xlabel('epoch')
  ax.legend(['train', 'valid.'])

def dataset_to_numpy_util(dataset, N):
    dataset = dataset.batch(N)
    for images, labels in dataset:
        numpy_images = images.numpy()
        numpy_labels = labels.numpy()
        break;
    return numpy_images, numpy_labels

def title_from_label_and_target(label, correct_label):
  correct = (label == correct_label)
  return "{} [{}{}{}]".format(CLASSES[label], str(correct), ', shoud be ' if not correct else '',
                              CLASSES[correct_label] if not correct else ''), correct

def display_one_flower(image, title, subplot, red=False):
    plt.subplot(subplot)
    plt.axis('off')
    plt.imshow(image)
    plt.title(title, fontsize=16, color='red' if red else 'black')
    return subplot+1

def display_9_images_with_predictions(images, predictions, labels):
  subplot=331
  plt.figure(figsize=(13,13))
```

```
  classes = np.argmax(predictions, axis=-1)
  for i, image in enumerate(images):
    title, correct = title_from_label_and_target(classes[i], labels[i])
    subplot = display_one_flower(image, title, subplot, not correct)
    if i >= 8:
      break;

  plt.tight_layout()
  plt.subplots_adjust(wspace=0.1, hspace=0.1)
  plt.show()
```

Install Spark locally for quick testing

You can use the cell below to **install Spark locally on this Colab VM** (like in the labs), to do quicker small-scale interactive testing. Using Spark in the cloud with **Dataproc is still required for the final version**.

In [ ]:
```
%cd
!apt-get update -qq
!apt-get install openjdk-8-jdk-headless -qq >> /dev/null # send any output to null device
!tar -xzf "/content/drive/My Drive/Big_Data/data/spark/spark-3.5.0-bin-hadoop3.tgz" # unpack

!pip install -q findspark
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/root/spark-3.5.0-bin-hadoop3"
import findspark
findspark.init()
import pyspark
print(pyspark.__version__)
sc = pyspark.SparkContext.getOrCreate()
print(sc)
```

```
/root
3.5.0
<SparkContext master=local[*] appName=pyspark-shell>
```

Section 1: Data pre-processing

This section is about the **pre-processing of a dataset** for deep learning. We first look at a ready-made solution using Tensorflow and then we build a implement the same process with Spark. The tasks are about **parallelisation** and **analysis** the performance of the cloud implementations.

1.1 Getting started

In this section, we get started with the data pre-processing. The code is based on lecture 3 of the 'Fast and Lean Data Science' course.

This code is using the TensorFlow `tf.data` package, which supports map functions, similar to Spark. Your **task** will be to **re-implement the same approach in Spark**.

We start by **setting some variables for the *Flowers* dataset**.

In [ ]:
```
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' # glob  pattern for input files
PARTITIONS = 16 # no of partitions we will use later
TARGET_SIZE = [192, 192] # target resolution for the images
CLASSES = [b'daisy', b'dandelion', b'roses', b'sunflowers', b'tulips']
    # labels for the data
```

We **read the image files** from the public GCS bucket that contains the *Flowers* dataset. **TensorFlow** has **functions** to execute glob patterns that we use to calculate the the number of images in total and per partition (rounded up as we cannont deal with parts of images).

In [ ]:
```
nb_images = len(tf.io.gfile.glob(GCS_PATTERN)) # number of images
partition_size = math.ceil(1.0 * nb_images / PARTITIONS) # images per partition (float)
print("GCS_PATTERN matches {} images, to be divided into {} partitions with up to {} images each.".format(nb_im
```

```
GCS_PATTERN matches 3670 images, to be divided into 16 partitions with up to 230 images each.
```

Map functions

In order to read use the images for learning, they need to be **preprocessed** (decoded, resized, cropped, and potentially recompressed). Below are **map functions** for these steps. You **don't need to study** the **internals of these functions** in detail.

In [ ]:
```
def decode_jpeg_and_label(filepath):
    # extracts the image data and creates a class label, based on the filepath
    bits = tf.io.read_file(filepath)
    image = tf.image.decode_jpeg(bits)
    # parse flower name from containing directory
```

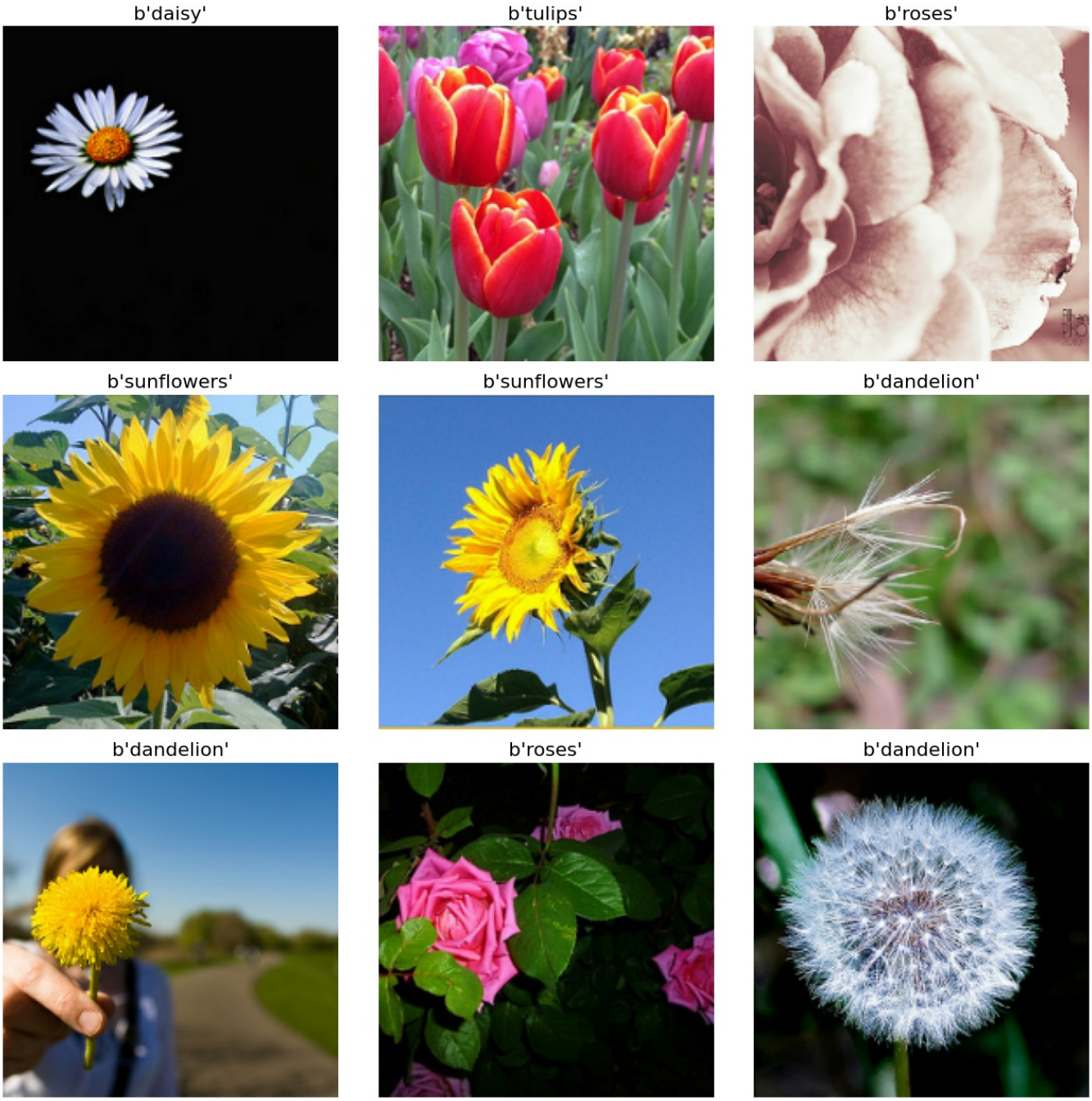
```
    label = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/')
    label2 = label.values[-2]
    return image, label2

def resize_and_crop_image(image, label):
    # Resizes and cropd using "fill" algorithm:
    # always make sure the resulting image is cut out from the source image
    # so that it fills the TARGET_SIZE entirely with no black bars
    # and a preserved aspect ratio.
    w = tf.shape(image)[0]
    h = tf.shape(image)[1]
    tw = TARGET_SIZE[1]
    th = TARGET_SIZE[0]
    resize_crit = (w * th) / (h * tw)
    image = tf.cond(resize_crit < 1,
                    lambda: tf.image.resize(image, [w*tw/w, h*tw/w]), # if true
                    lambda: tf.image.resize(image, [w*th/h, h*th/h])  # if false
                    )
    nw = tf.shape(image)[0]
    nh = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (nw - tw) // 2, (nh - th) // 2, tw, th)
    return image, label

def recompress_image(image, label):
    # this reduces the amount of data, but takes some time
    image = tf.cast(image, tf.uint8)
    image = tf.image.encode_jpeg(image, optimize_size=True, chroma_downsampling=False)
    return image, label
```

With `tf.data`, we can apply decoding and resizing as map functions.

In [ ]:
```
dsetFiles = tf.data.Dataset.list_files(GCS_PATTERN) # This also shuffles the images
dsetDecoded = dsetFiles.map(decode_jpeg_and_label)
dsetResized = dsetDecoded.map(resize_and_crop_image)
```

We can also look at some images using the image display function defined above (the one with the hidden code).

In [ ]:
```
display_9_images_from_dataset(dsetResized)
```



Now, let's test continuous reading from the dataset. We can see that reading the first 100 files already takes some time.

In [ ]:
```
sample_set = dsetResized.batch(10).take(10) # take 10 batches of 10 images for testing
for image, label in sample_set:
    print("Image batch shape {}, {})".format(image.numpy().shape,
        [lbl.decode('utf8') for lbl in label.numpy()]))
```

```
Image batch shape (10, 192, 192, 3), ['roses', 'dandelion', 'daisy', 'daisy', 'tulips', 'daisy', 'dandelion', '
dandelion', 'roses', 'daisy'])
Image batch shape (10, 192, 192, 3), ['sunflowers', 'sunflowers', 'tulips', 'dandelion', 'dandelion', 'tulips',
'sunflowers', 'dandelion', 'tulips', 'roses'])
Image batch shape (10, 192, 192, 3), ['sunflowers', 'sunflowers', 'daisy', 'dandelion', 'daisy', 'daisy', 'rose
s', 'sunflowers', 'dandelion', 'tulips'])
Image batch shape (10, 192, 192, 3), ['daisy', 'sunflowers', 'daisy', 'daisy', 'daisy', 'dandelion', 'daisy', '
dandelion', 'daisy', 'roses'])
Image batch shape (10, 192, 192, 3), ['dandelion', 'dandelion', 'sunflowers', 'roses', 'daisy', 'dandelion', 't
ulips', 'roses', 'dandelion', 'tulips'])
Image batch shape (10, 192, 192, 3), ['sunflowers', 'daisy', 'tulips', 'roses', 'sunflowers', 'tulips', 'sunflo
wers', 'sunflowers', 'tulips', 'sunflowers'])
Image batch shape (10, 192, 192, 3), ['daisy', 'tulips', 'daisy', 'dandelion', 'sunflowers', 'daisy', 'sunflowe
rs', 'dandelion', 'dandelion', 'tulips'])
Image batch shape (10, 192, 192, 3), ['tulips', 'roses', 'daisy', 'sunflowers', 'tulips', 'sunflowers', 'dandel
ion', 'daisy', 'dandelion', 'dandelion'])
Image batch shape (10, 192, 192, 3), ['sunflowers', 'roses', 'tulips', 'roses', 'sunflowers', 'daisy', 'tulips'
, 'roses', 'dandelion', 'sunflowers'])
Image batch shape (10, 192, 192, 3), ['tulips', 'dandelion', 'daisy', 'tulips', 'tulips', 'sunflowers', 'roses'
, 'tulips', 'dandelion', 'sunflowers'])
```

1.2 Improving Speed

Using individual image files didn't look very fast. The 'Lean and Fast Data Science' course introduced **two techniques to improve the**

speed.

Recompress the images

By **compressing** the images in the **reduced resolution** we save on the size. This **costs some CPU time** upfront, but **saves network and disk bandwith**, especially when the data are **read multiple times**.

In [ ]:
```
# This is a quick test to get an idea how long recompressions takes.
dataset4 = dsetResized.map(recompress_image)
test_set = dataset4.batch(10).take(10)
for image, label in test_set:
    print("Image batch shape {}, {})".format(image.numpy().shape, [lbl.decode('utf8') for lbl in label.numpy()])
```

```
Image batch shape (10,), ['dandelion', 'tulips', 'tulips', 'sunflowers', 'dandelion', 'sunflowers', 'sunflowers
', 'dandelion', 'tulips', 'daisy'])
Image batch shape (10,), ['roses', 'dandelion', 'tulips', 'dandelion', 'daisy', 'roses', 'sunflowers', 'dandeli
on', 'sunflowers', 'dandelion'])
Image batch shape (10,), ['tulips', 'tulips', 'sunflowers', 'sunflowers', 'sunflowers', 'tulips', 'sunflowers',
'tulips', 'daisy', 'roses'])
Image batch shape (10,), ['daisy', 'sunflowers', 'tulips', 'daisy', 'sunflowers', 'tulips', 'daisy', 'tulips',
'tulips', 'dandelion'])
Image batch shape (10,), ['dandelion', 'sunflowers', 'roses', 'dandelion', 'sunflowers', 'dandelion', 'tulips',
'roses', 'tulips', 'roses'])
Image batch shape (10,), ['roses', 'tulips', 'sunflowers', 'roses', 'dandelion', 'dandelion', 'sunflowers', 'da
isy', 'dandelion', 'dandelion'])
Image batch shape (10,), ['dandelion', 'sunflowers', 'tulips', 'sunflowers', 'daisy', 'daisy', 'dandelion', 'su
nflowers', 'daisy', 'daisy'])
Image batch shape (10,), ['dandelion', 'tulips', 'sunflowers', 'sunflowers', 'daisy', 'dandelion', 'dandelion',
'dandelion', 'sunflowers', 'daisy'])
Image batch shape (10,), ['sunflowers', 'daisy', 'sunflowers', 'daisy', 'dandelion', 'tulips', 'dandelion', 'su
nflowers', 'roses', 'dandelion'])
Image batch shape (10,), ['dandelion', 'sunflowers', 'tulips', 'daisy', 'roses', 'tulips', 'tulips', 'sunflower
s', 'tulips', 'roses'])
```

Write the dataset to TFRecord files

By writing **multiple preprocessed samples into a single file**, we can make further speed gains. We distribute the data over **partitions** to facilitate **parallelisation** when the data are used. First we need to **define a location** where we want to put the file.

In [ ]:
```
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # prefix for output file names
```

Now we can **write the TFRecord files** to the bucket.

Running the cell takes some time and **only needs to be done once** or not at all, as you can use the publicly available data for the next few cells. For convenience I have commented out the call to `write_tfrecords` at the end of the next cell. You don't need to run it (it takes some time), but you'll need to use the code below later (but there is no need to study it in detail).

There is a **ready-made pre-processed data** versions available here: `gs://flowers-public/tfrecords-jpeg-192x192-2/`, that we can use for testing.

In [ ]:
```
# functions for writing TFRecord entries
# Feature values are always stored as lists, a single data element will be a list of size 1
def _bytestring_feature(list_of_bytestrings):
    return tf.train.Feature(bytes_list=tf.train.BytesList(value=list_of_bytestrings))

def _int_feature(list_of_ints): # int64
    return tf.train.Feature(int64_list=tf.train.Int64List(value=list_of_ints))

def to_tfrecord(tfrec_filewriter, img_bytes, label): # Create tf data records
    class_num = np.argmax(np.array(CLASSES)==label) # 'roses' => 2 (order defined in CLASSES)
    one_hot_class = np.eye(len(CLASSES))[class_num]     # [0, 0, 1, 0, 0] for class #2, roses
    feature = {
        "image": _bytestring_feature([img_bytes]), # one image in the list
        "class": _int_feature([class_num]) #,      # one class in the list
    }
    return tf.train.Example(features=tf.train.Features(feature=feature))

def write_tfrecords(GCS_PATTERN,GCS_OUTPUT,partition_size): # write the images to files.
    print("Writing TFRecords")
    tt0 = time.time()
    filenames = tf.data.Dataset.list_files(GCS_PATTERN)
    dataset1 = filenames.map(decode_jpeg_and_label)
    dataset2 = dataset1.map(resize_and_crop_image)
    dataset3 = dataset2.map(recompress_image)
    dataset4 = dataset3.batch(partition_size) # partitioning: there will be one "batch" of images per file
    for partition, (image, label) in enumerate(dataset4):
        # batch size used as partition size here
        partition_size = image.numpy().shape[0]
        # good practice to have the number of records in the filename
        filename = GCS_OUTPUT + "{:02d}-{}.tfrec".format(partition, partition_size)
        # You need to change GCS_OUTPUT to your own bucket to actually create new files
        with tf.io.TFRecordWriter(filename) as out_file:
```

```
            for i in range(partition_size):
                example = to_tfrecord(out_file,
                                      image.numpy()[i], # re-compressed image: already a byte string
                                      label.numpy()[i] #
                                      )
                out_file.write(example.SerializeToString())
            print("Wrote file {} containing {} records".format(filename, partition_size))
        print("Total time: "+str(time.time()-tt0))

write_tfrecords(GCS_PATTERN,GCS_OUTPUT,partition_size) # uncomment to run this cell
```

```
Writing TFRecords
Wrote file gs://daring-healer-421511-storage/tfrecords-jpeg-192x192-2/flowers00-230.tfrec containing 230 record
s
Wrote file gs://daring-healer-421511-storage/tfrecords-jpeg-192x192-2/flowers01-230.tfrec containing 230 record
s
Wrote file gs://daring-healer-421511-storage/tfrecords-jpeg-192x192-2/flowers02-230.tfrec containing 230 record
s
Wrote file gs://daring-healer-421511-storage/tfrecords-jpeg-192x192-2/flowers03-230.tfrec containing 230 record
s
Wrote file gs://daring-healer-421511-storage/tfrecords-jpeg-192x192-2/flowers04-230.tfrec containing 230 record
s
Wrote file gs://daring-healer-421511-storage/tfrecords-jpeg-192x192-2/flowers05-230.tfrec containing 230 record
s
Wrote file gs://daring-healer-421511-storage/tfrecords-jpeg-192x192-2/flowers06-230.tfrec containing 230 record
s
Wrote file gs://daring-healer-421511-storage/tfrecords-jpeg-192x192-2/flowers07-230.tfrec containing 230 record
s
Wrote file gs://daring-healer-421511-storage/tfrecords-jpeg-192x192-2/flowers08-230.tfrec containing 230 record
s
Wrote file gs://daring-healer-421511-storage/tfrecords-jpeg-192x192-2/flowers09-230.tfrec containing 230 record
s
Wrote file gs://daring-healer-421511-storage/tfrecords-jpeg-192x192-2/flowers10-230.tfrec containing 230 record
s
Wrote file gs://daring-healer-421511-storage/tfrecords-jpeg-192x192-2/flowers11-230.tfrec containing 230 record
s
Wrote file gs://daring-healer-421511-storage/tfrecords-jpeg-192x192-2/flowers12-230.tfrec containing 230 record
s
Wrote file gs://daring-healer-421511-storage/tfrecords-jpeg-192x192-2/flowers13-230.tfrec containing 230 record
s
Wrote file gs://daring-healer-421511-storage/tfrecords-jpeg-192x192-2/flowers14-230.tfrec containing 230 record
s
Wrote file gs://daring-healer-421511-storage/tfrecords-jpeg-192x192-2/flowers15-220.tfrec containing 220 record
s
Total time: 137.7730758190155
```

Test the TFRecord files

We can now **read from the TFRecord files**. By default, we use the files in the public bucket. Comment out the 1st line of the cell below to use the files written in the cell above.

In [ ]:
```
#GCS_OUTPUT = 'gs://flowers-public/tfrecords-jpeg-192x192-2/'
# remove the line above to use your own files that you generated above

def read_tfrecord(example):
    features = {
        "image": tf.io.FixedLenFeature([], tf.string),  # tf.string = bytestring (not text string)
        "class": tf.io.FixedLenFeature([], tf.int64) #,   # shape [] means scalar
    }
    # decode the TFRecord
    example = tf.io.parse_single_example(example, features)
    image = tf.image.decode_jpeg(example['image'], channels=3)
    image = tf.reshape(image, [*TARGET_SIZE, 3])
    class_num = example['class']
    return image, class_num

def load_dataset(filenames):
    # read from TFRecords. For optimal performance, read from multiple
    # TFRecord files at once and set the option experimental_deterministic = False
    # to allow order-altering optimizations.
    option_no_order = tf.data.Options()
    option_no_order.experimental_deterministic = False

    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.with_options(option_no_order)
    dataset = dataset.map(read_tfrecord)
    return dataset


filenames = tf.io.gfile.glob(GCS_OUTPUT + "*.tfrec")
datasetTfrec = load_dataset(filenames)
```

Let's have a look **if reading from the TFRecord** files is **quicker**.

In [ ]:
```
batched_dataset = datasetTfrec.batch(10)
sample_set = batched_dataset.take(10)
```

```
for image, label in sample_set:
    print("Image batch shape {}, {})".format(image.numpy().shape, \
                         [str(lbl) for lbl in label.numpy()]))
```

```
Image batch shape (10, 192, 192, 3), ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0'])
Image batch shape (10, 192, 192, 3), ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0'])
Image batch shape (10, 192, 192, 3), ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0'])
Image batch shape (10, 192, 192, 3), ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0'])
Image batch shape (10, 192, 192, 3), ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0'])
Image batch shape (10, 192, 192, 3), ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0'])
Image batch shape (10, 192, 192, 3), ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0'])
Image batch shape (10, 192, 192, 3), ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0'])
Image batch shape (10, 192, 192, 3), ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0'])
Image batch shape (10, 192, 192, 3), ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0'])
```

Wow, we have a **massive speed-up**! The repackageing is worthwhile :-)

Task 1: Write TFRecord files to the cloud with Spark (40%)

Since recompressing and repackaging is very effective, we would like to be able to do it inparallel for large datasets. This is a relatively straightforward case of **parallelisation**. We will **use Spark to implement** the same process as above, but in parallel.

1a) Create the script (14%)

Re-implement the pre-processing in Spark, using Spark mechanisms for **distributing** the workload **over multiple machines**.

You need to:

i) **Copy** over the **mapping functions** (see section 1.1) and **adapt** the resizing and recompression functions **to Spark** (only one argument). (3%)

ii) **Replace** the TensorFlow **Dataset objects with RDDs**, starting with an RDD that contains the list of image filenames. (3%)

iii) **Sample** the the RDD to a smaller number at an appropriate position in the code. Specify a sampling factor of 0.02 for short tests. (1%)

iv) Then **use the functions from above** to write the TFRecord files. (3%)

v) The code for **writing to the TFRecord files** needs to be put into a function, that can be applied to every partition with the 'RDD.mapPartitionsWithIndex' function. The return value of that function is not used here, but you should return the filename, so that you have a list of the created TFRecord files. (4%)

In [ ]:

```
#We first import the necessary libraries
import os, sys, math
import numpy as np
import scipy as sp
import scipy.stats
import time
import string
import datetime
import random
from matplotlib import pyplot as plt
import tensorflow as tf
print("Tensorflow version " + tf.__version__)
import pickle
import pyspark
from pyspark.sql import SQLContext
from pyspark.sql import Row

#We define the variables that will be needed for this project.
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' #glob pattern for input files.
PROJECT = 'daring-healer-421511'
BUCKET = 'gs://{}-storage'.format(PROJECT) # This is the bucket storage.
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # We prefix for output file names.
PARTITIONS = 16 # This is the number of partitions that will be used later.

#Task 1a) - i) Copy over the mapping functions and adapt the resizing and recompression functions to Spark (onl
def decode_jpeg_and_label(filepath):
    #This function reads and decodes the JPEG image from the file.
    bits = tf.io.read_file(filepath)
    image = tf.image.decode_jpeg(bits) #We decode the read data as JPEG images.
    label = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/') #We extract the label (flower name) of
    label2 = label.values[-2]
    return image, label2

def resize_and_crop_image(data): #We specify one argument
    image, label = data #The input is basically a tuple containing the image data and the label.
    #We determine the dimensions of the input images and the target dimensions.
    w = tf.shape(image)[0]
    h = tf.shape(image)[1]
    tw = TARGET_SIZE[1] #Target Width
    th = TARGET_SIZE[0] #Target Height
    #Calculate resize criterion to maintain the aspect ratio.
```

```
    resize_crit = (w * th) / (h * tw)
    #We resize the images to fill the target dimensions but also preserving the aspect ratio.
    image = tf.cond(resize_crit < 1,
                    lambda: tf.image.resize(image, [w*tw/w, h*tw/w]), # if true
                    lambda: tf.image.resize(image, [w*th/h, h*th/h])  # if false
                    )
    nw = tf.shape(image)[0]
    nh = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (nw - tw) // 2, (nh - th) // 2, tw, th)
    return (image, label)

def recompress_image(data):
    image, label = data #The input is basically a tuple containing the image data and the label.
    #This function is used to reduce the amount of data, but takes some time
    image = tf.cast(image, tf.uint8) #The image is transformed to a uint8 so the images can be compressed to JP
    image = tf.image.encode_jpeg(image, optimize_size=True, chroma_downsampling=False) #We compress the images
    return (image, label)
```

```
Tensorflow version 2.15.0
```

In [ ]:
```
#Task 1a) - ii)  Replace the TensorFlow Dataset objects with RDDs, starting with an RDD that contains the list
from pyspark.sql import SparkSession
sc = pyspark.SparkContext.getOrCreate() #Create or get the existing Spark context
spark = SparkSession.builder.getOrCreate() #Initialize the Spark session

image_paths = tf.io.gfile.glob(GCS_PATTERN) #We retrieve the list of image filenames
image_rdd = sc.parallelize(image_paths) #We create an RDD from the list of image filenames
```

In [ ]:
```
#Task 1a) - iii) Sample the RDD to a smaller number at an appropriate position in the code. Specify a sampling
rdd1_sample = image_rdd.sample(False, 0.02) #We first sample the RDD with a sampling factor of 0.02
rdd2_decode_jpeg_and_label = image_rdd.map(decode_jpeg_and_label) #We decode the jpeg and label of the images o
rdd3_resize_and_crop_image = rdd2_decode_jpeg_and_label.map(resize_and_crop_image) #We resize and crop the imag
rdd4_recompress_image = rdd3_resize_and_crop_image.map(recompress_image) #We recompress the cropped images of t
```

In [ ]:
```
#Task 1a) - iv) Then use the functions from above to write the TFRecord files.
#This function is used to create a TensorFlow feature for storing byte strings. This is for the image data.
def _bytestring_feature(list_of_bytestrings):
    return tf.train.Feature(bytes_list=tf.train.BytesList(value=list_of_bytestrings))
#This function is used for storing integer values. This is for the labels.
def _int_feature(list_of_ints):
    return tf.train.Feature(int64_list=tf.train.Int64List(value=list_of_ints))
#This function is used to create a TFRecord from image bytes and a label.
def to_tfrecord(tfrec_filewriter, img_bytes, label): # Create tf data records
    class_num = np.argmax(np.array(CLASSES)==label) # 'roses' => 2 (order defined in CLASSES)
    one_hot_class = np.eye(len(CLASSES))[class_num]     #[0, 0, 1, 0, 0] for class #2, roses
    feature = {
        "image": _bytestring_feature([img_bytes]), # one image in the list
        "class": _int_feature([class_num]) #,      # one class in the list
    }
    return tf.train.Example(features=tf.train.Features(feature=feature))

print("Writing TFRecords")
#This function is used to write TFRecords, which will be applied to partitions of an RDD.
def write_tfrecords(partition_index,partition):
  filename = GCS_OUTPUT + "{}.tfrec".format(partition_index) #We define the filename for this TFRecord file bas
  #We Open a TFRecordWriter object for the filename.
  with tf.io.TFRecordWriter(filename) as out_file:
    for element in partition:
      image=element[0]
      label=element[1]
      example = to_tfrecord(out_file,
                            image.numpy(), # The re-compressed image is already a byte string
                            label.numpy()
                            )
      out_file.write(example.SerializeToString()) #We convert the image and label to TFRecord format and write
  return [filename] #We return the filename of the written TFRecord.
```

```
Writing TFRecords
```

In [ ]:
```
#Task 1a) - v)  The code for writing to the TFRecord files needs to be put into a function, that can be applied
#should return the filename, so that you have a list of the created TFRecord files.

rdd5_partitions = rdd4_recompress_image.repartition(PARTITIONS) #We repartition the RDD to have the number of p
rdd1_filenames = rdd4_recompress_image.mapPartitionsWithIndex(write_tfrecords) #We specify an index for each pa
```

In [ ]:
```
rdd1_filenames.take(1) #We check to see the filename of the first TFRecord.
```

Out[ ]:
```
['gs://daring-healer-421511-storage/tfrecords-jpeg-192x192-2/flowers0.tfrec']
```

In [ ]:
```
rdd2_decode_jpeg_and_label.take(1) #This is the first element of the RDD2 showing the decoded JPEG na labels of
```

Out[ ]:
```
[(<tf.Tensor: shape=(263, 320, 3), dtype=uint8, numpy=
  array([[[133, 135, 132],
          [136, 138, 135],
          [140, 142, 139],
          ...,
          [152, 152, 150],
          [155, 155, 153],
          [148, 148, 146]],

         [[133, 135, 132],
          [136, 138, 135],
          [140, 142, 139],
          ...,
          [153, 153, 151],
          [155, 155, 153],
          [147, 147, 145]],

         [[132, 134, 129],
          [135, 137, 134],
          [139, 141, 138],
          ...,
          [152, 152, 150],
          [154, 154, 152],
          [146, 146, 144]],

         ...,

         [[ 44,  48,  25],
          [ 44,  48,  25],
          [ 44,  48,  25],
          ...,
          [127, 126, 122],
          [127, 126, 122],
          [127, 126, 122]],

         [[ 44,  48,  25],
          [ 44,  48,  25],
          [ 44,  48,  25],
          ...,
          [128, 127, 123],
          [128, 127, 123],
          [128, 127, 123]],

         [[ 43,  47,  24],
          [ 43,  47,  24],
          [ 43,  47,  24],
          ...,
          [129, 128, 124],
          [129, 128, 124],
          [130, 129, 125]]], dtype=uint8)>,
  <tf.Tensor: shape=(), dtype=string, numpy=b'daisy'>)]
```

In [ ]:
```
rdd3_resize_and_crop_image.take(1) #Take the first element of the RDD3 showing the resized and cropped images.
```

Out[ ]:
```
[(<tf.Tensor: shape=(192, 192, 3), dtype=float32, numpy=
  array([[[154.31648 , 158.31648 , 161.31648 ],
          [154.03465 , 158.03465 , 161.03465 ],
          [152.40732 , 156.40732 , 159.40732 ],
          ...,
          [166.26291 , 167.93884 , 169.71353 ],
          [164.40128 , 168.40128 , 169.40128 ],
          [164.      , 168.      , 169.      ]],

         [[168.3533  , 172.16167 , 175.54494 ],
          [166.39734 , 170.39734 , 173.39734 ],
          [163.65054 , 167.65054 , 170.65054 ],
          ...,
          [165.4297  , 167.10564 , 168.88033 ],
          [164.40128 , 168.40128 , 169.40128 ],
          [164.      , 168.      , 169.      ]],

         [[164.95058 , 168.      , 172.90114 ],
          [163.81895 , 167.81895 , 170.81895 ],
          [162.34184 , 166.34184 , 169.34184 ],
          ...,
          [166.90747 , 167.95851 , 169.9415  ],
          [166.25023 , 167.47679 , 169.40128 ],
          [165.84895 , 167.07552 , 169.      ]],

         ...,

         [[120.384514, 118.384514, 119.384514],
          [123.10339 , 121.159195, 122.131294],
          [124.0755  , 124.0755  , 124.878075],
          ...,
          [127.89167 , 127.34229 , 124.66636 ],
          [126.44648 , 127.44648 , 122.44648 ],
          [126.97421 , 127.97421 , 122.97421 ]],

         [[121.46287 , 119.46287 , 120.46287 ],
          [124.460785, 122.51658 , 123.48868 ],
          [124.8213  , 124.8213  , 125.62388 ],
          ...,
          [129.9137  , 129.46465 , 126.78871 ],
          [126.821304, 128.      , 123.      ],
          [127.      , 128.      , 123.      ]],

         [[122.18799 , 120.18799 , 121.18799 ],
          [124.977264, 123.03305 , 124.00516 ],
          [123.783615, 123.783615, 124.5862  ],
          ...,
          [131.46323 , 131.13916 , 128.46323 ],
          [126.925804, 128.8151  , 123.815094],
          [127.      , 128.02274 , 123.022736]]], dtype=float32)>,
  <tf.Tensor: shape=(), dtype=string, numpy=b'daisy'>)]
```

In [ ]:
```python
rdd4_recompress_image.take(1) #This is the first element of the rdd4 showing the recompressed cropped images.
```

Out[ ]:

```
[(<tf.Tensor: shape=(), dtype=string, numpy=b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x01\x01,\x01,\x00\x00\xf
f\xdb\x00C\x00\x02\x01\x01\x01\x01\x01\x02\x01\x01\x01\x02\x02\x02\x02\x02\x04\x03\x02\x02\x02\x02\x05\x04\x04\
x03\x04\x06\x05\x06\x06\x06\x05\x06\x06\x06\x07\t\x08\x06\x07\t\x07\x06\x06\x08\x0b\x08\t\n\n\n\n\n\x06\x08\x0b
\x0c\x0b\n\x0c\t\n\n\n\xff\xdb\x00C\x01\x02\x02\x02\x02\x02\x02\x05\x03\x03\x05\n\x07\x06\x07\n\n\n\n\n\n\n\n\n
\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\xff\xc0\x00\x11\x08\x00\xc0\
x00\xc0\x03\x01\x11\x00\x02\x11\x01\x03\x11\x01\xff\xc4\x00\x1c\x00\x00\x03\x01\x01\x01\x01\x01\x01\x00\x00\x00
\x00\x00\x00\x00\x00\x04\x05\x06\x07\x03\x02\x01\x08\x00\xff\xc4\x00D\x10\x00\x02\x01\x03\x03\x03\x02\x03\x04\x
08\x03\x06\x05\x05\x01\x00\x01\x02\x03\x04\x05\x11\x00\x06\x12\x07!1\x13A"Qa\x142q\x81\x08\x15#BRb\x91\xa1\x16\
xb1\xc1\x173r\x82\xd1\xe1CS\x92\xb2\xf0\t$4c\xc2\xf1\xff\xc4\x00\x1c\x01\x00\x02\x03\x01\x01\x01\x01\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x03\x04\x01\x02\x05\x00\x06\x07\x08\xff\xc4\x008\x11\x00\x01\x04\x01\x03\x03\x02\x04\
x05\x04\x01\x02\x07\x00\x00\x00\x01\x00\x02\x03\x11!\x04\x121\x05AQ\x13"\x062aq\x14\x81\x91\xa1\xc1B\xb1\xd1\xf
0b#R\x07\x163\x92\xc2\xe1\xf1\xff\xda\x00\x0c\x03\x01\x00\x02\x11\x03\x11\x00?\x00\xfd5\xbc:l\xbfg\xfbK[)\xaa\x
a3\x1d\xfb@;\x8f\xe9\x91\xf9cM9\x96)\x00r\xa2+6F\xd9e+\x1d\x03\xd1\xb6\x7fq\xb9(\xef\xf2n\xff\x00\xdf@14\x8a!\x
10 c\xda\x97\xad\xb1X\x97\x8bd)_N\xa0\xf3Z`y\x05>A_?\xe6>\xba\x10\x89\xd1\xba\xc2\x92A]\xb7\x15\x8e\x92\xedo[\x
b5\xa2Q,N2\x08\x1d\xd4\xfc\x8f\xd4j\xb2g*\x1b\x81I\r\n\xb2\x1cH0\xf1v`}\xd7\xe7\xf9h\x1b\x88WZ\xf6\xd5\xa7\x82\
xfb\xb7b\xad\xa3\x94Hx\x85\x99\x07\x98\xdb\xe4~Z\xd2\x89\xe1\xcdAp+\x95u\x9d\xf2c#\xe2C\xd8\x11\xe7R\xf5!\xd4)z
\xa5\x84\x04\xf4\xe5\\\x0c`\x9flh@\xf6R\x99\xd8\x0bS\xc8\xf6\xe7\x97*FT\x9f\x18\xd5\xb7R\xa9\x04\x94-\xf6\x98\x
c1Pb\x0b\x8eC\xf7|gJ=\xd96\xac\x17]\xb9C\xf6\xa8\x1aR0Q\xc0\xe4?\xb6\x8d\x05l*\x1c/(\xeb\x9d9\x92"Ja\x91\xbb\xf
duw\xb8\xaaU)\xc9\xe3\rP"9\xc6y\x03\xee>\x9aQ\xce\xf7"\x80i.\xdd5\xeb%\x17\xd9\xd8\x10\xe8p\x0e\xa0\xba\xd4\x81
If\xd2\xa252\x19\xd8\x9c!gbuM\xdbl\xab-Kc\x07Khi{r\\\xfe\'N@\x08\x88!\xbb(m\xedsx\xa2g\x07\xca\x80\x07\xd7R\xf7
P\xc2\x86\x82\xb2\x1b\xbc\x8a\xd5/\x17\x12\xceX\xfa\x87K;!\x15\tg\x85\xea*\x16*\x18\x81%\xb0X\x8f:\xb3\x0eT\x1e
\x15\xc5\xaa\x86+u\'\xa2\x1b\x94\x84|o\x9f\x1ap\r\xa1U\xa36\xbe\xd6T\xfai\x85\xc1\x1f#\xaa\xda\xba\xf1K5B\xfcl\
xc0)=\xb8\xe8E\xc6\xd4\x11i\x84W\x04\\\x12\x17\x97\xb0\xd5\xc3\x85*"c\x96i\xc6do\x1e\x01\xed\x8dM\x85\xd5j\x86\
xcd\xbc\xe8\xae\xf1\x08j\xc0\x89\xc8\xfb\xcd\xddO\xd0\xfc\xb4\xc0p*\xa5\xa4!w\x1e\xd1\xa4\xb8#IO\x18\x8aO\x97\x
81\xab\x96\x82\x17\x02F\x14\x15\xd2\x9a\xe5a\x9c\xb2\x93\x1b\xc6{\xe7\xb1\xc6\x82AW\x02\xd2\xe9\xf7\x0cQ\xcc\xf
3\xc9N\x8a\xd2\x91\xf6\x92\x8b\x81\'\xd5\x87\x82~\xbe~\xba\x03\xfc\xab\x01E\x05s\xb7AT?X\xdb@\xc7\xef\x01\xfd\x
f4\x8c\xa0\xf6R\xbcm\xbd\xd1u\xda\x97\x11]l\x9c\xa9=\x99s\xf0\xba\xff\x00\t\xf9\xfb\xe8qN\xe6\x11\x95\x05k\x9bn
\xefg\xdf\xb6\xaf\xd6T\n\x12d\xedQ\x01\xc6S\xea5\xac\xc9\x04\xad\xb1\xca\x13\x81\x0b\xec\xb6\x89)*\x03\x18\xb2\
x87\xce|\x11\xa1\xbb\xd8mH6\xbd\x1bi\x86e\x9a\x01\xd9{\x9f\xc0\xe8\x01\xe2\xd4\xa1\xee\xc0M nJq\xd8\x8d\nRM\x95
\x00R\'f*Esk{7\xc3P\x84\x00|\x07\x1d\xc7\xfa\x8f\xcfQ\xa5\x97k\xe8\xa9Ln\xb4,\xa9\x95\x18*pA\xf9i\x99\r\x1bU\xa
bR\xd7H1;\x11\xd8\x81\xd8\x8d(\xe22\xe4e%\xb8\xd8\xd4\xd4\x10s\x9c\xff\x00}T>\xd7&\x14\xb0\xfe\xa8\xdbe\xc9\xc3
\xc80@\x1d\xf0N\xa9;\xa8\x86\xaeW\xf6\xc7Z\x1a\x14\x90\x90\xa1\x17\xce<\x7f\xf3\x1a\xd4s\x83@CQ\xddD\xbe4\xce!\
x82L)\x07:^G\xf8Vj\xcf\xeb\rD\xd5\x1cQ\xb8\xa1\x19$\xfb\xe8\\\xab&\xb6\x1a\x8a\x1b]u5\xbc\xb05u\x9c\xfe\xcb\x02
\x81\xc9\x91\x00\xf5\x1f\x1ex\x82\xca\t\xc7\x97A\xe5\x80\xd3\x11`*\xbb\x850\x91K\x1a\x86S\xf1.A#\xce\x8a\xe7\xd
0\\\xde\x10RG-@w\x990\xa0\xfe\x1a\x1e\xf5e\xcaA \x94,hr\x17\xe1\\\xe8w}\xd7&\x14\x14\xce"\x12N\x01o\x90\xd1ZqJ\
```

x08\xb2\x98\xc6\xe8\xa7\xb0\xfcs\xa30P\\\x05(\x8b}\xd6\xb2\xdd\xc6XKI\t\xf1\x83\xdck\x81\xa54\xad6\x9e\xff\x00\
x8eH\xd6\xddr\x98\xe0vI\x1b\xbe;\xe8\xad}\xa1\x90\x99_h\xa2\xb9\xc0V\\:\xb0\xf8X\x0c\x91\xf8\x1dK\x95Au\xac\xc7
wY\xaa-s\xb3p\xcac\xb1S\xdb\x1aZAH\xa1!\xa2\xdckAX)%\x9f\xd1Wo\x86F\x19U?\xcd\xf3_\x9f\xd3\xbf\xb6\x96~M.<#\xea
\xd2*\xd4z\x88#(c8\x9a\x12Fco\x97\xd4\x1f \xfc\xb5\x9c\xf1\xb4\xa9D\xec\xdd\xe5_\xb2o\xf1\xdc\xe92T\x10\'\x84\x
f8u\xf7\xfc\xfeZ\x985\x067\xd8\xe1q\x00\x85\xb8\xcd\xbb\xf6\xadU\xba\x0b\xb55j4UQ+\xac`d\xaeGp~Z\xd7s\xc3\xdb\x
f4B\r\xa4\x82\xbb\xa86jW+O\x1b\xc8A\xed\x98\x81\x18\xfc\xce\x80Z{\x14@\x02\xf1\xfe\xd1\xecu\xc3\x15\xb6\xe0\xb8
\x03\x05\xa0Q\xfeCB>\xafb\xba\x826\x82\xe5o\x9f\x15v\xc5G*\xc1\x82\xaf\xc2r<`\x8e\xd9\xfcF\xa9\xb9\xa1\xde\xe0\
xb8\x8bO\x16\xfdl\xbe\xc6\xefLJJ\x8b\x89`\x93\xef\x021\xfdt\xcb\xddm\x07\xb2\xa0\x05M_\x93\xf6\xce\xe8\x07\x8f\
x1a\xcd{\xad\xc5\x11MU[\xe3\xa8\xabW\xf1\x9e\xec51;\xdd~\x17.\x1b\x86\xb1RJj5\xce\x1ed^\xff\x00V\x1a]\xcf\xdf0\
xfb\xaeVw*\xf1\x1d\x03A!\n\xae\xbd\xbf\xeb\xfd\xf5\xae\xf7Z\xa6B\xcew\x05\xc9j\xd5\x84J~\x169\'K\xbd\xe3n\x15\x
85\xa51\xa27+\x85\xc6\xa5)\xe9\xe0\x8d\x9eY\xe5p\xaa\x88\xa3,\xccO`\x00\x19\xc9\xd7F\xe79\xdfE%\x13\xd1\xfaz\xc
d\xca*z\x97u\xa6\xf4\xd2\xf2\xa8\xb6zFB\x1a\x9e\xdc\x99\xf4Cg\xc3I\xc8\xcc\xcb\xfb\xbc\xd1\x0eLy\xd3\xd1\xd5*e^
TS4\x90<\x92F\xcc\x07`W\xb9\xc9\xec\x064\xbe\xa6M\x80\xbb\xc2\xbbE\xa1\xae\x14\x92A\xc66h\xc7\x15\xc7\x10|\x9dQ
\x8f%\x80\x95\xc88\xe1XY\xa4\x90\x8c\x8e\xc7\x1f\xe5\xa2\x85\xc8\x98\xb8\x14R\x1f\x05[-\x83\xfd\xb4f.A\xde\xef\
xd1\xc0\x1a\x9a\x9c\x8e\xfd\xd9\xbd\xb4n\xd8\\\x92\x8bl\xf4\xbd\xc2\xfe\xf6Y\x08\xf6\xd5\xa8\xa8\xdc\x17\x89)Lg
\xed\x14|\xc8\xcf\xc4\x9e\xeb\xf8k\x97r\x9c\xd9w,\xf0\xe2\t\xfb\xc6|\x920F\xb8=Aj\xf1\xb9\x16:\x98\x99\x98\xf2G
\x1fw\x1e\x0e\xaa\xfc\x85\x19\x0b4\xddV\x8fH\xb4\x94\xeeH\x1e\x01\x1d\xd4\xfc\xf4\xa4\x80\x82\xadx\\,;\x92H\x82
E;~\xd2%\xe2\x8cO\xdeO\xe0o\xa7\xcb\xe5\xa4\xe4\xa7`+\x01i\x95\xea\xe1M\x1c\t[\x11\xf8\x9c\x90#\xff\x00\xae\x83
\x14m/\xbe\xca\x0e\x13}\xb7\xba\xaaf\xa5Zy\x98\x80Ga\x9c\xe3\xf0\xfe\xfa\xd2k\xc2\xee\xf4\x88\xa9\xb8\x05\x05\x
85r\xa6O`O\x8dT\xb8\x05\x16\x877Z\xc4\xf8\xe2\x99e\x03!\x95[T\xb0WX]\xad\xdb\xc6zI\xb9D\xef\x04\x87\xeb\xd8\xff
\x00\xa6\xaa\\\xdb\xa2\xa4eUY\xb7\x90\xbb\xcb\x1b\x89\xfd\x1b\x94#(\xcap%\x1e\xe3\xf1\xd5ZK\x0e\x17*\x07\xb9Gw\
x89+\xa3B\x0e\n\xca\x87\xf7[\xfe\xfaOR\xd3\x1b\xb77\x82\xb9,\xb8\xb4t\xb4\xed3/\xde\xec>\x9a\x03e\xd9\x19+\x94\
xbd}\xc1g\xdc\xf6\xfab\xb9\x1fj\x8b\x90\xfc\x18ixfk\xb5\x00)?-\xaa\xed\xcd^\x88\x87\x93\x1f\x80\x11\xc7\xe7\xad
i%\xb7*\x0c\x05\x19Zfw`\xa3\x00\x8c\xaa\x01\xef\xf5\xd0\x8c\x96)[\x94\xa7ql\xaf\xf6\x81\x0c[\x1a\xb0+\xdb&\x91%
\xbf+\xb7\xc3QN\xbd\xc5)_,$n<\x87\x8fLH\xa7\xbb\xae\x9d\x88S,\xaeZm\x0c\x11\xa4h\xbe\x9f\x1f\x01B\x8d2\xd2)reMW
\r\x00v\x97!\x01\xee=\xd8\xfd4\tH"\x97$\xb7\x0b\x84/4\x93\xcb\x18\r\x9e\xca;\xe3Ch\xda\x03B\xe4\xbf\xed\xb19\xe
6T\x02Ol\xe8\xed\x1eW!n\xf7\xf8\x92\x16\x82\x19?hs\xf7F\x8c\x1c8\\\x91\xa2UV\xb9\xee\xccI\xee\x07\xb9\xf9h\x8d"
\x89\\p\xb4\xa6\xdbi[\x18u\x88\x06#\xe1c\xe0\xe9\xb2/\x84\x0b!)\xbam:\x98\xb9\x95\xa4\n\xe3\xc3)\xec\xc3Bp!H&\x
d2*\xeb|\x91;\x19)\x1e)=\xc1=\x8e\x86\xe0A\xb5rm\tQU,I\xe9\xb9\x93\x04wW\xf6\xd0\x9c\xea4T\x81aNnA\x92\xce\x99e
a\xd8g\xc6\x83!\xb2\xacr\xa2\xea\xe2hf\xf5\xe2P\x00\xf6\x1aJL\x1b\n\xcdE%A\xae\xa6\xe2\tfN\xea\xba\xa4o\x02\xca
\xea$\xa2-\xd3K\x13\xe4\x16\xe5\x8c*\x8c\xea\xb2j\xe3\x8c\xd9+\xb6\x94\xf6\x8e\x07\xb8\xc4\xd3\xbcA@\x18=\xbb\x
e99\xb5\xb2m\xdc\xd1aue\x01p\xa4JYK$\xec\x87=\x8a\xf6\xd6[\xfa\xa4\x834\x8a\x19\x84)\xae\x9a5\xc4\xc7\xd6Q\xe1\
xb3\xdfB\x8f\xae\xc6]\xb1\xfc\xab\x98\x88\x08\x8a+\xc4\xb0J\x19fa\xc1\xc1G\x07\xb8>\xdav\x1e\xaa\xc7\x11\x95S\x
11\xa5\xa5l\xdd\xcb\xfa\xce\x93\xd7\x91\xb0\xcc8\xd4\x00?{\xd9\xb5\xa8\xf9\x1b44\x80A\x08\x8d\xc1Z\x1a\x826\x0c
2;\x90O\xd7^zMH\xf4\xf6\x92\xae\xd6\x95\x1dM<\x95{\xda\x95\xd0\x90#\x91X\xf7\xfea\xa1\xe9go\xae\xd3\xf6\xfd\xd1
\x1c\x05*\xfb\xb5JMS)\x90\x12O\xdc\xfakW\xf1-s\x8a\r\x14\x96i\x11eW\xa9\x1f\x070\xaa\x84\xfd\xf2H\x00\x7f\xa7\x
e7\xa3\xe9]\xeb\xbdA\x14\xa8vm\x84[\xe9~!\xceY~9e\x03\x1c\xdb\xf8\x8f\xf9\x01\xec;\x0e\xc3[!\xc0\x1a\n\x11\xb7:
\xf7\xb7\xc4\x07.\xfc\xfd\x86\xa2ICW%7\x0b\xbc\xb2T\x86\x12\x90\x02\xf6\\\xf8\xfa\xe8\x02l\x92W%\xd2\xdc\x1aU"J
\x82[8-\x9f\x1a4N\xb1er\x12\xe5wX\xa1\xfb=;\x92|gF\x04\x95\xcb\x95\x9a\xcb-\xca\xa3\x99\r\x8f,\xc4\xe8\xcd\xa6\
x8c\xaeH:\xb7I\xd4\r\xdf\x1b\xf4\xf7\xa5\x8d\x15\xba\x9d\x93\x8d\xdbpT\xbb(\x8dO\x98 U\x1c\xa4r\x0eK\x8f\x80\x0
3\x8e\\\xb2\x16\xee\x8d\xd2\x8a\x06\x82\xa6\xe2\x80\xe97\xe91\xb86\x8bEl\xbd\x13s\xb7\xa8\xc2\xd3\xcb&$\x88\x7f
#\x1fa\xfc\'\xb7\xcb\x1aR-c\xdb\xca#\x98\xd3\xc2\xdf6\x87V\xfaO\xd4\x1at\x8e\x8e\xfd\x145\x0e0\xf4u\x9f\xb2\x91
I\xf6\x1c\xbb7\x7fu$kA\x9a\x88\xe5\x18(%\x94\x8f\xbc\xec\xa8*)\xcf\t\xbdhH\xf8\x1dXr_\xfa\xe8\x84[Ur\xa276\xd8\
xac\xa1\x84\x8a\x95\xca\xa9\xf8eO8\xfc4\xb4\x8d\xb0\xae\xc2\xa2o\x94|I\x85\xaa\x06Yr\x8d\x8e\xcc>_C\xa5\x1e\xec
\xed(\x8aF\xe7O\x1a\xbeQ\xc9\r\x91\xc7\xe4t\xa4\x84\x10\xa4r\x95\xc57\xd9\xaar\x0e@>3\xa4\x1cj\xd19NbFp\x1a\x95
\xf8\xcf\x8eQ\x90{0\xf9~:Bs#M\x14K\xb5\xdfon\xc9(\xebq:\x13\xcb\xb4\xd0\xe7\x01\xc7\xcc|\x8f\xd3H7Y\xf8|\xf2\xd
e\xe3\xfc}U\xdd\x18!6\xbdCOSL+\xe8\xa4\x12\xc1!<H\xecA\xf3\xc4\xfc\x88\xd2:\xb7\xb5\xa7{\r\xb4\xf7\xf0\xa6;\xaa
!H\xd5\xcd%+\x11\x13\x11\xdf\xdck\x0bW#\x1f\x19.Eo(\xcb|\xb1VQ\x8a\xb8\xfe\x16\xf0\xe3=\x89\xd7\x93\x1doU\xa0\x
d7\x08\xa4u\xb1\xdc\x1f\x1ft\xcf\xa6\xc7\xb6\xc0T[\n\xeb5\x1d\xd1i\x99\x88I{\x1e\xfe~Z\xfa\'J\xeb\xa2H\xcb\\\xe
4\x94\xd0VUE\xea\xbf(bbq\x1cE\x89\xf6\xfb\xda\xf3\x9a\xce\xbb\x08\xd7\xbe&\xbf!Y\x90\x9fL\x12\xa66uW\xda\xb7$\x
95\xc1\x8f\x15\xaaT\x19\xfa\x1c\x9d\x13K\xd6c2\x80\x1dy\xfe\x15\x9d\x11\xda\xb4\x0bm\x1cWP\xe0\xd5\x08\xcf\x80\
xe4v\xc9\xc99\xf1\xd8\x01\xfd\xf5\xa7\xa8\xea\xd3\xe9\xdb\x14p7s\xde@\xaf\xf8\xd8.\'\xec\x0f)v\xc6\x1co\xb2\x82
\xdb\xbb\x9a\x1d\xf5\xd6z\xfd\x9ff\xbbE%\x1e\xd0\xa5\x8c\xdeR\x1f\x88\xcb_RO\xa5\x1f/e\x86\x18\xe4fLwj\x98NG\xa
6A\xf6\xfd=\xde\x9c%\xc4\xe7\xfd\xfe\x12\xee\xe5kP!\xa5\xa7\xe4\x87\x88T\xc7}h6Z\xb7*\xa9=\xc3t\x91\xeb\x96\'\x
a8\xe4\x179\x03\xdc\xe8O\xd4[\xf2\xba\x88\xcaZ\x93%D\xe4\xc8\xb2\x16\xf9\xfb\rZ"v\xe5A\xe1\x15\x1d\x0f\xac\x87\
x8cX\x03\xe6q\xa7\xa3kH\xca\xe2W\x83Gi_\x80\xd7,\xb3\x7f\xe4\xc09c\xf1a\xd8i\xa6\xechU\xca\xedMAYP\xa5\'\xab0S\
xe7\xbcQ\x9f\x88\xfe\'W.iV\x1c"\x96\xb2\xcfl\x8f\xd2\x81\x87\xd3=\xc9\xd4\xef\xb5\xd4\x17\xe5\xe1G(\x00\xf1!\xf
d\xfd\x8e\xb1\xc9\xb4TU-\xc6\xb6\x95G\x04/\x8f#\xdf\x1e\xff\x00\xfc\xef\xae\xb2\x15\x83|\xab\xde\x9f~\x90[\xe7i
2%%\xdeG\x85p\x1a\x9e\xa9\xc9\x18\xf9\x03\xe4}<\x81\xf2\xd31j\xa5guGF;-sl~\x91\x9bos\xc4)o\xb4)\x14\x8d\xd8\x82
\x00\xe4\x0f\xd7\xee\xb7\xf6?M:\xcd\r|\xc8E\x85\xab\xb6\xe2\xda\xf6\xbb\xad!\xb9m\xe7Y\xe3#>\x988*~X\xf6:\xab\
xda\xceB\x80J\xcd\xf7\x05\x96JY\x19\r+\x18\xcfu+\xf7\x90\xfc\x88\xd2R4\x01\xc2\xb5\xa8\xbb\xc4/\x0c\x82hC\x02s\
xd8\x8f:\xca\x98\x96\xe5\x18e\x1b\xb6n\xb1]Q\xad\x13HR\xa4w\xa5l\xe3$~\xeet\x83\xa6d\x8c,<\xf6W\r<\x85\xf6\xe9\
x19\xb8\xc5%bFc\xab\xa68\xab\x88\x0e\xe7\xf9\xc7\xd3\xe7\xaf1\xd4\'s#.o\xe6\x99`\xceWk\x1e\xe1\xaf\xb71\x92d\xc
aH\x80\xcd\x1c\x80\xf1\x94x\r\x8f\x9f\xd7^\x17W\xf1K4l{\x9aw\x01`\x8f\xe3\xe8S\xbf\x84\xde@8]+\xe8M\xe2h\xe5\xb
5\x85h\xe6p\x0b3c\xd1\xc9\x19\xe4\x06Oa\x93\xdb9\x1e5\xe5:\x87\xc6p\xb9\x8c\x92\x12Fr+5\x9c"\xb3H\xf6\xb8\x82\x
82\x8e\x9a\xefj\xbbG\xb7)i}j\x99\x9b\xd4\x8c\xab\xe27L\x16$1\xc0$\x01\xe3\xf2\xf3\x8dy-GY:\xc7\xbbPI\r\x00c\xb8
#\xe9\xf5)\x83\x17\xa6\xd0\xd1\xca\xa3\xd8\xf50\\.t\xd2\x00\xca\x19\x89R\xc3\x00\x80pH\'\xc8\x04\x11\xadM7\xc5\
x8f\xd2FK\xcdVU\x1d\xa7\x12vV\xb51ERja\x9a\xaa\x9e5\xabY"\xa0\xaa\x12s\xe5 8!\xd3\xb7\x10\x1c0\xc9\'8\xf6\xf7\x
f1\x1a\xae\xbb\xaa=RMk\t\xb7\x1bp\xff\x00\x89\xec\x0f\x9e\nz=&\xf8\xb6\xd7\nN\xc7MW\r\xfe-\xbfo\xa6\x91\x9a\x97
\x94\xf7I\xc2abf\xe3\xe9\xa7o\x04\xa9$\x0f8BN5\xeb\xfe\x19\xea:\x89\xb5\xac\x8d\xc6\xb7\x1a\x04\x9ey\xb2O\x81@Z
KQ\x11\x02\xc0\xe1?\xdc\xf5\xfb\x9e\x96\xc756\xd5\xb1\x9a\xea\xf2\x9e\x95\xa6\xdf,\xe2\x05\xac\xa9 p\x0f+ab\x8f
\x969HH\x00\x03\xdf\xb6\xbe\x95\'\xc5\xfa&k\xbd\x00\xea\x8d\xb4\x1c@\xb3Wg\x8b>(\x01d\xac\xf6\xe9\x0e\xc2{\xac\
xdf\xff\x00\xa7\x0e\xc7\xbbmn\x94\xdd\xb7>\xea\xbb-u\xd7q\xee\x9a\xdb\x95mcS\xc9\x13I!+\x1b\xe5\x1c\x021"I\xf4\
xd7\xd3\xe2\xeaq\x17\x86\x0c\x10\x06\rw\xfb\x1f\x14\xb3\x8c/h\xca\xdd\xf7\x15\xda*zFST]\x9d\x8e\x02\xe9\xb7k\xe
f\xda\n\xa0b\x92.\xf3Ne\x9202{\x16\xd3\x9aR_\x97(x\xf0\xbc\x8b\x89O\xd9QG\xc9\xc7\xb7\xbf\xe6}\xb5\xab\x1c\xaco
\xb5\xaa\x9bJ\xf32\xbc\xd1\xfa\xb7\xcb\x89H\x87\xfe\x18n\xc7\xf3\xf7\xd3\x1e\xa3\x8f*)\r\xfe/\xa3\xa4&\x96\xc9E
\x9c~\xf0_:3e\x05u/\xef\xd6w*\xbc\xc9YVc\x8c\x8e\xe1tm\xc0\x05\xcb\xdf\xeb\x1aT\\\xa82\x1f\xe3=\xb1\xab\x83\x85
\xcb3\xac\xd9u\xbe\x9f\xadFV\xad\x07\xdf\x10\x0c\xb8\xff\x00\x93\xc9\x1f\x818\xf7\xd2\xaf\xd38e\xb9V\x05)\x96\x
d0\x17*\x8c\xdd\x8f`\xa3\xc64\xb9i\n\xc1\xc8j\xabc\xe0I\xc1\x94\x8f|\xf7\x1a\x19%\\\x1b\\\xe2\xad\xa8\xb7\x8cA#
\xb6\x0eJ7\xfah^\xa0\xbb\@*\xbbcu\x9a\xeb`\x99\x14UH#\x03\x01\x1c\xfd\xd1\xf2\xfa\x8f\xa6\x8d\x16\xad\xcd\xc1\
xe1Q\xd1\x920\xb4JM\xf1\xb5\xf7\x8d05\xc5a\x97\x18\x12\xc5\xe0}J\xf9\x1al\xcf\x13\xdbHE\xa4)\xdd\xe1\xb4\x9e\x1
8\x1f\xec\xd2\xac\xa8\xe31H\xa7\xe1\'\xfd4\x84\xec\x0ee+4\xd1\xca\xcd.\xf2\xd5QV\xf2\x8d\x82H\x1b\xca\x9c\x15#\
xdf^+\xa9L\xe8M\x84\xfcM/\x14\xaa\xa9*&\xbf\t*g\x9e*[\x95=>*\x04\xb2qI\x00\x03\xb9o\x1d\xfe\x9f]|\x87\xe2\x8f\x
8b\x84\xba\x11\xe8]I\x8b\x1d\x88\xf29\xbf!j\xe9\xb4\x80>\xdc8N\xf7%\x8e\x82\xd9AJ\xb6\xdb\xad\xb5\xe6j4\x92\x9e
\x8e\xa6\xa9c\xaa\x92Le\xa3\x80\x9c\xa3\xb1\'\n\x84\xfcY\xc2\xf28\xd7\xc9\xc6\xafS\xae\xd4\xbc\x90[n7B\xdb\xcf&
\x8d\xfd\xcek\xfa\xb6\xadF\xb1\xac\xab+\xb5\xd6\xdd\xb9/\x95\xb7\nKM\x96:\x0b\x8de\xa4\xc9d\xbaH\x85)\xeaS81\xc
9\xff\x00\x97)R\x98\xed\xe0\x9c\x9e\xd9\xd2\xccn\x9a0\xd7M%\xb0:\x9c\x0eH\xe4\xd8\xf2\xd1\xdf\xb7\xd5D\x8f\xac\
xb5\x17\xb7\xfawk\x93\xa4\x8f\xf6\xdb\x85\xd0]\xed\x0c\xb5uu\x13\xc8=(gU*\xe8\xc5\x10\x80\x88Y\xbb\x02y\x85\x01
\xb2\x18\x8dL\xba\xe7G\xd4\x8e\xc8\xc1c\x8dpM\x8el\xe6\xed\xdc};f\x95\x1c\xfb\x01\xc52\xdb;o{n{\xd4\x15\xf4\x94

\x14IC\r\xb2A-\xae\x82\xa3\xec\xea\xb5\xee\xe0\xbc\x91\x86*LG\x93\xb9\x1f\x03c\xe1\x00d\x1d/\xa9\x97I\x0b\x1dd\
xee.$8\x8b\xf6\x0b\xc1\xf0p\x05w\xe7\xc8Bl\x96\xeb\x1c\x0f\xee\xa9\xea:}i\xb9n\xba]\xef\x15\x7f\xdb\xed\xf4\xd0
\xa40[\xe3c\x11Y\x89f\xf8\xfb\x96\xe1\x92q\xc7\'\'\xe2#\x199_\x8d-\xd2\xba-\xb4\xe2Nn\xf1\xf4\xfd\xfe\xc9\xb8\x
e4{l\x03\x82\x8c\x8bh\xdc\x92\xfd]O@!\x10\x96T\xb6\xd2L\x9e\x9dF\x02\xb1nm\x8e,\xc4\xf8r{\xf2\x00\xfc\xf4m7Sv\x
8bl\x8c$>\x88\'\x90\x03\x8e\x08\xee\x062>\xea(H\xda\xa4\x93n\xef\xab<\x10H)\xa9\xe7\xb7S,&\xa6j\x89\x91\x832\xb
3\x00\x1c\xe7\xb9\x05\x98`\x0f88\xf0q\xa9\x07P\xeb\x1a\x19\xdb$R{\xb7gos\xe0\xf1\x7fP\xa5\xf0\xc5\xe8\xd9\xee\x
8cZ\xdak\x8cF\xf1KY-I\xc9"R\x00\xecG\x8f\xa8?\xdb\x1a\xf6\x9a/\xfcC\xebR\xea\x19\x06\xa0\x02l\x0b7u\xe1f\xcb\xd
3\xe2km\xaa~\xfb\xb9!\x8e\xa3\x84\xcc\xc5\x87\x80\xbd\xf1\xaf\xb1\xf4n\xb2\xc9\xce\xf6\xfb\x8a\xcb\x92\x1d\xa3(
zz\x9a\xbb\xa2\x83?(\xa2?\xd4\x8d{]4\xefx\xb9\r%\x0bE\xaf\x17\x1b\xf5\x1d\xa2\x98\xd3\xd1\xa0g\xf7o\xae\xb6c\xd
44\x8c!\xed)\x0c\xf55\x17i\x83\xd6\xd5\xb1\xc9\xec\xa0v\x1an7=\xc6\xadT\x9aL\xed\xb6k\x83*\xfd\x85J\xae~\xf0\xf
7\xd3\xad\x07\x81\xca\xa5\xdf)\xed\x0e\xcf\xab\x9c\xabTM)c\xf3^\xdai\x99P\x9c\xc1\xb5`\x83\xf6\x95\xf5P(\xf91\x
d1\xac\x8c.Y}U=U\xb2_V\'\x18\xf6#M].^\x9e]\xb9\xb9I\x8a\xf0\x1a\x9a\xac\xae#\xac\x8b\xefg\xc7\xc4<8\xc7o\x9f\xd
7\xdbKH\xc8\xe49\x0b\xb8J\xef\xdb>\xe3gO\xb4\xd4\xd3\xfd\xa2\x9d\xff\x00\xdd\xd6\xc1\x96\x8d\xfe\x84\xf9S\xfc\x
ad\x83\xee24\xa4\x9a}\xa2\xc2\x90\xe4\x92\xa2\xdb4\xb1\xfa\xf1@\x00\x1e\xf8\x1d\xb4\xa3\xe2\x7f!Y\xaf\xa2\x95\x
d6\xda\xaa\xe3&I\x1b+\xe0\xb0Q\xdbJ\xb9\xaef\n(6\x10\xf4\xf7K\x8d\x9aQ\xe8\xd6\xb8+\xde6Q\xed\xa5\x9f)iD\xda\xd
7&\xd4\x1dQ\xdd\x16\xb8\x83X\xaeTfA0ymw\x80~\xc7R\xbf\xbe\xa1\xd4\x17\xa7s\xed"\xf2P{\xb4o\x9c\x80\x1e\xa1\xe8\
xfc\xfc*\xba\x10x\\\x9e\xe1\xb6\xfa\x8ft\x96\x95lW}\xb9q\x82eZ\x8a\x0b\x9cJ\xf1\xc88\x87\xe5\x0c\xd1\x9e2\xa1\x
1e\x1b\x03\xe5\xd8\xf6\xd7\xce\xfe/\xf8\x8f\xa7h#p\x07\xdcA\xab\xfd9\xed\x94\xfe\x8fK)\xc9\xe1V\x7f\x86\xb6\xdd
5{Mt\xdd5iGKNjc\x8e\x9e!\xea\xc2\xaa\xc0\x00\xc4\x86\xe5\x1eG\x13\x903\xdb\xb8\xecO\xe7Ig\xd5n \xb0o.\xa3g\x04\
x9c\xd8\xcf#\x91\xd9z\x00\xc6\xb8\x84\xef\xa6\x97\xbd\x87\xbcx\xeem\xbfZ\xb6\xea\xbb-\xbaj*J\xfb\x94i.e\xe2\t\x
95\x80\xe3\xe0\x10\xa3\x1f\x17\xc4{\xb1\xc6\x15\xd7\xc3\xaa\xd2;\xd1\x7f\xcb!\x04\x86\xd85|s\x9b\xef~\x06\x00\x
b0\xa1\xc0\x84\xfe\xabs[\xaav\r\xae\xbe\xe5}\x8a\xe2\xd4\xd7@\xc6\x11\x03\x85\x902\xb1c\x13\xa1\xe4\xc0d\x96\x0
7\xca\x8c\xe3 i6\xc6\xe7N\xe8\x9a\x08\xc7z\xc0\xc1\xcf\xf1\xf5!\x05\xee]\xaf\x14\xb77\x16\x99\xf7\x17L\xb7\nz\
x12\xd4\xc6\x8bew\xe4\xf2F\xec\x0bpyXrnD\xbe\x0ep1\x96\xc6\xac\xed8\x91\xdbg\xf6\xb8w\xae\x0f\xd8_\xdb\xf8A2Riw
\xdc\xbb\x9e\x9bq\xd0\xd2n{\xec[~t\xa0d\xb6U2%RV\xb7\x95w\xe3 \xe2\xab\xf7q\xf7\xf0\xf9\x00v$\x1f\x85\x87\xd3$\
xdc\x83\xb8\x06\x8bG\x8f\xe7\xf6*\xa2P8\tu\xa7\x7f\xd9\xae\x16jK5\x81\x8d%\xcaj\xb6ZZ\x9a\x96\xa8h\x98\'.e\xc98
q\xcb\xc3`\x03\x9c)\x18\xc6\xa6M\x0c\xac{\x9d(\xb6\x81\xda\xaf5@W\xdf!\x18J\x15]%\xe2\xacWU\xdb\xf7\rj\xcdX)\xf
e\xcfCs\xa3\xf5\x19\x10\xc8\x84\xc3\xc4`|Ya\x95n\xe7\x00\xf8 \xeb4\xc2\x1a\xd6\xbe1l\xb0h\xfd\x0ew_a\xe4\n\xe4r
\x8e$\xdc,\x1aG\xdc\xe4\xaf\xbfl\xd2\x94\x94\xf4\x91\xd7\xd2\xc4XQ\xdc\xe9@s\x84#\xd0\x95\x8fe?\x17l\xf6\xca\xf
9\xf7\xd0\xe3\r\x87QN\'i<\xb7<\xf7\xfc\xd5\xc1\x04\x8caF\xeeK\xce\xe0\xb4\xd2\xd1\xd9\xb7\r\xb2\xe8k\xea\xe02\x
b5\r%\x0f#\x06H\xe2=Q\xf0\x01\xc5\\\xb7\xde\x03\xe68\xe7ZZh#s\x9c\xf8\xc8\xda/7\xc8\xac\xe3\xfb+\xb8\xb5\xd2SW\
x1a\x94\x86\xc5\x04"\x08\xe3\x86\xa6\xacs\xaa2I\xcd\xd5\x8f~\n\xc7\xbf\x15\xfb\xbe;\xe3\xbeN\xbe\xf7\xd3\xba\xd
7O\xf8/\xa7@\xd3\xef\x96@\x0b\x88\x02\xcf\xef\x808\x03\x8bXRB\xfdc\xcd`\x0e\x17\x99j\xcc\x91\xb4un\x18\x05\xfb\
xd1\x9c\x91\xf5\x04k\xe9]\x07\xe2n\x99\xf1<>\xdf\xedD,\xe9\xb4\xf2i\xdd\x94\x8et\xb5\xd1\xbf\x1b\x84\xa6C\xe5YW
\xd8\xfd5\xe9!\x84B\xfa$Wd\x0c\x91\x84\xca\xd3]\xb5\xca\xf1\x12q\xc7\x91$}\xbf\xb6\xb6a\xd9\xe4 \xba\xc1O\xe9n\
xd6\xa8T-%\x02\x96\xce\x15\x83`g\xf0\xc6\x9e\x8c6\xee\xd5l\x95\xdcOx\xac^p\\~/\xe1\x8d\x80\x00i\x88\xd8G\x0b\x9
7H,\x17\xe9\xfe)\xa9\xdd\xbee\xbd\xff\x00\r\x15\xac\xbeUw\x05\x01x\x06\x9f\xb8\xc6\x08\xfb\xa7\xbe\x99-VR\xb7Z\
xe8\xa9$%\xe9X\x8c\xf6#\xb6\x96\x7f\xd1p\xca\xf7\xb7z\x93]\xb7\xaaA\xa4\xad\xf4\xb9\x0e2E*\x86Y\x17\xdc2\x90A\x
1f\x8et1.\xd55\xe1Z\xed\xfb\x8fI\xb7\xbc\x8b\x1d\xce\x99,\xd5\xd2\x1c\tb\x95\x96\x96F\xf9\xe3#\xd3\xcf\xd4\xe3?
\x8e\xac\xd7\xc3)\xa3\x82\xaaZP\xbb\xef\xa66-\xa3Ln7\x8a\x1b\x92\xd0\xc9\xdc\\\xe8\x1b\xd6\xa7Q\xfc\xd8C\xc0}X\
x81\xf5\xd2\xda\x8d;\x1b\x92\tR\xd2n\x94\x05E\x8bm\xdfT\xc5\xb6\xb7\xad\xba\xa8\x1f\xba$\xc2\x11\xf2\xfb\xa5\xb
f\xafmbN\xc8\x1d\xc3\xab\xee\x8e\xc78vR\xbb\xa7`o\xeag\x8e\x8e\x92\xcd\x14\xc94\x8a\x86\xae\x9e\xa9$H\xc1 r`\xa
4\xb0\x03=\xf25\xe5\xba\xbc\xa3G\xa7t\xae \x86\x8b\xa0E\x9f4\x13\x91\x11#\xaa\x95OM\xb6\xcd\xafh\xdad\xa2\xde\x
fb\xab\xed\xd78]j\xed\xd0Q(x\xa1\x08rS\x93\x9c>IS\xc0\x81\xd8\xb6\t\xceW\xf3\x8f\xc4\xbdr~\xb1\xa9i\xd3\xc5\xb5
\x82\xc1\xb3\x93\x7fA\x91\xfe\xdd/C\xa5\x84F=\xc5P\xec\xca\x9bF\xfaYk/;j\xa2\x96\xa6\xa2\'[_\xead\x04WS\x07e&x\
x00\x93\n\x1c\x14#\n\x08\'8\x1a\xc0\xd4F\xfd.\xd0\xc7\xee\x18\xbb\xfe\x93\xff\x00\x126\xe7\xbf\xf9Vq\x16|#-\x9b
\x1a\x03\xb7?\xc6\xd6\x8d\xb3g\xb4\xdc\xac\xed#^\xac\x91+S\xadX\x8c\x90\xb2<`\x18\xd6NJ\xc5|\x86\xcff\x03\x03C\
x97T\xe7O\xf8yd.\x0e\xf9\\}\xd4Ok\xc1<\xfd\xfc\xa5\x9c\xe7\x11\x8e\x11Kp\xdcvn\xa7\xd9\xaf\xdbj\xd5hi\xafV\xb6\
xf4E%\x7f\xa2\xa2X\xdc\t\x19\x96@\xaa\x81\xc3\xa0c\x18r\xc7\x19\xc7|\xc8l3\xe8^\xe9\x1cF\xc7\x0b\xb6\x93\x87\n\
x1c\x1bw\x1fA\xfb%\x8b\xcby\xe1Cn=\xd3E\xb2\xa8n\x16\xfa\xe8j\xa2\xdc\xb47\x80\xafD\xd5\xc5\xa0r\xe4\xf2D\xf8p\
xa9\xc1\xd5\x94\x80\x07\x10\xa3\xb6\xb4a\xd2;XC\x9a\xdb\x8c\x8f\x9874\x07&\xcf$\x8a?[K:b\t\x08\xed\xa3\xd6]\xcc
\xfb\x92w\xdc[^\x0b\x8d\x05J\xfd\x8a\tkPMF\x14 &\x0f]\xc0\x8dp\xfc\x88\xca\xe7\xe1\x00\x96#&%\xe9\x90\xb2\x10`\
x93i\x16O6\t\xbav\xdc\x9e+\x17_`P\x8c\xc5\xe6\xd5$\xfb\xce\xedi\xb7\xd2\xf4\x8frn\xbbm5=\xd5"\x92\xdbq\xa6\x94L
\x86\'NN`\xf8\x1deU\xe3\xc4\x10U\xbd\xf2\xa3\x1a\xcf\xfc\x1c[\xce\xb1\xad.\r\'sq\xcb\x87\x1b\xb3\x83\xc9\xee(&\
x1b\xa8\xc7\xd54\xd9W\xfe\xacU\xd2]zqo6\xda\xab\x8d\xber\x8bU\xf6\xd5\x8c\xdc\x11\x91dGL\xe4\x02\xc0\xa0\x01\xd
8(c\x82p4\x8e\xabE\xd3\x1b#\'\xdcC]\xc0\xab\x00\xd9\x06\xfb\xd0#&\xb8\xcar9C\xb9\n\xc2\xd7Y\xb8/\x14\xf6\xfd\xd
7\xb9id{r\xc4\x91C\r5a\xf5!.\x03s\x9c(9\x1e\xcb\xdf\x03$09\xc6\xb2\x9f\x140F\xe6Fm\xc0\xe7\x1f\x7f\x97\xf5\xf1\
x7fT\xd0v\xef\x95z\xdc\xb6\xfe\xa0n\xca*[\xdd\xb6Zg\xb4\xdb\xdeIZ\tf+QW\x1a\xb7\xec\xc1P0W\x00\x903\xdc\xb0\xce
\xa9\x00\xd1\xc0\xe3\x13\xac\xbd\xd40,\x02y\xcf?\xee\x11\xda\xebu\xb8`%\x17}\x9b\xc2\xf7[\xbf7\xbb\xc5Tdx\xa1\x
b2\xda)\xea\x1b\x11!\x18&A\x80\x1aFol\x95\xc0\xfd\xe3\x82\xbe\xf3\xa0u\xde\x95\x04\x8d\x83]\xa7t\xef\x03%\xdc\x
0f\xd6\xc9\xe7\x16Ew\xb4\x86\xa2)\\\xed\xcd4>\x8b\xf9v\xed=\xd2\x11[CH\xf0\x12}6\x96\x19\xbdH\xd5\xff\x00\x85\x
d7\xbe\x0fq\xe0\x8e\xdf\x9e\xbe\xc5\xd0\xf5=\x12=l"\x18\xbd7;\x16\xcc\xb2\xfc\x1a\xf6\xdf|\xd1#\x85\x930\x9a\x8
d\x9bR71\xbb\xa1\xba\xcdnM\xadm\x96\x1aw\xe1\x05l\xd7\xae\x0b(\xfe.+\x0c\x87\x19\xc8\xc1*N3\xdb:\xfa\xa4Lk\x88\
xdf \xfc\x86?[H\xee\xae\x02\x16\xd9A\xbb\x92\xe6j\xb7GP-\xd1S\x02\x0cv\xfd\xbf\xb7x8\x1f&\xa9\xa9\x9ep\xe7\xfe\
x18S\xf0\xd6\xacGJ\x05\x0b?\x9f\xff\x00Hn.&\xd3\xa8\xef\x90Q\x90\xb1WU\xca\xa1p\x05S\xc6X\xfc\xc9\x11\xc7\x1a\x
93\xff\x00(\xd3\xec17\x80\xa8o\xba\xfe;\xc6\xb8\x9eP\xbb(\xc7o\x00\xff\x00m2\xc9\xa8(]!\xde\xb7\x15\xff\x00y4\x
c4\x8f\xff\x00a\xd1\x84\x8e\xbc(\xa0\x81\x8e+\x80_FU\x1cq\xe1\x86O\xf4\xd3$\xa9@\xd5Y\x9e~P\xcfC\x9f\xab\x0c\x7
fM.\\J\xee\x12K\xa6\xc9F\xc7\xaa\x8c\x84\xfd\xc6\xc6\x80\xe8\x81]t\x96\xcf\xb7.t\xa4\xa4n\x7f\x96U\xca\xe3\xf3\
x1aNF;\x84@ZBi\xb5\xfa\x8d\xd5\x0e\x9fHc\xb5^O\xa2\xe7-O \xe7\x13}x\xe7\xb1\xfa\x8c\x1f\xae\x8253A\x8b\xc2\xb6\
xc68 \xf7\xfd\x9b\xf4`\xeb\xa5BVu\x97\xa4\xc9h\xbc\x01\x8f\xf16\xdc\xf5)\xa5-\xfcN\xd0\xf1\x91\xbf\x07\xe7\xf3\
xcf\xcco\xd7i$\xf6\xca\xda\xfa\xae0\xc9\xc8BM\xd0K>\xd74\xad\xd3\xed\xcb]\xba,M\x1b\x06\xa9\xac\xdd\xb2K<m\x8f\
x07*\xc8\xff\x00\xf0\xe1\x08\xfe\xda\xf9\x8f\xc7\xba\xdd\x07N\xe9\xfe\xab\x9a\x0e\xec\n4\x7f\x9f\xec\xb4z|rH\xf
a\xf0\x9a\xed\x84\xb3[\xadt\xd7\x9bM5\x15\x15\x15\x9e\xe0MT79\xda:\xd7~9<e@\\)\x07#\x00\xa9\xee\x18c\xb8\xfc\xe
9\xa9\x96ifsey%\xe0\xd6\xdc\x81g\x16\x0f\x7f=\xef!z2\xc6\x81M\t\x8dM\xde\xef\xb6\xf6\xdc\x1dF\xb6\xbd\xa2\n4\x8
7\x91\xa6\xa4V\xf5\xe4\x86W\x88\xab\xbc\x8a\xa4\x11\xdd]\x97\xcf\x92{\x8d\x03\xd2\x8fQ)\xd2I\xbbu\xf2x\xb1\xc8\
x17\xfa\x7f\x94\x07\x10\r\xa4\x96*\xa8-\xf7J\xabwXk+k\xe8\xeb\xf2\xd1\xd3\xc3\x1b\xc1\x15Q$r0<c\x94\xfcX\xe5@-\
x82;\x8f\x03N\xbe2\x03\x1f\xa5`\x04w\xb0Mg\x9c\x90\xdb\xf3B\xd2O\x90\x93Ag\xbb\xab\x7fVlM\xff\x00]x\xbb\xdd\xae
Tq\xd1TKMe\xb2^\xd2X^zPW\x8c\x88dT\xcb3\xa9\xf8\xc2\x8cc\x89\xee\x0e\xb6\xe1\xe9\xcd\xd5h\x99\x1cl\x07p\x0es\x9
bF\x8el\x1a\xc8\x00f\x8f\xdcvK\xbaR\xa8\xef\x7f\xa4\x8fS66\xc5\xa8\xa0\xea^\xc7\xaf\xa7\xff\x00\x15=]\x19\x8a\x
9a\xb2\x07\x8dD\x84\x86\x97\xd4V<HV\xe5\xdcy\xc7c\xdb*i\xba\x06\x8fS\xac\x0e\xd2\xce\xd2"\x01\xd9\x04\x1c\x1b\x
aa9\xe4v\xaeR\x85\xd7u\xddM\xc1\xb4\xfa\x89\x06\xdbk_No\xb2V\xf4\xe2\x96\x15\xa8\xab\xb9\x19\xd2J\xdbl\xcaAH\xf
d#\x96\x95CH\x18\xbf\x15!Y\x81\xc7\x1dh\x89\xba|\xb2\x99u-\xadO\x1bk\xda\xe1\xdc\xdd\xe0\xd0 \x0f#\x08V[\x85M\x
b4\xad\xb1\xed\x87\xb6\xed/\xd2CnV\x08\xddV\xe3n\xa8\xa5\xaffz\x89\xb1\xc3\xd4Y #\x93\x05P\x0cJ\xd9\n\xeb\xdb\x
b8\xce~\xa9\xf2Ist\xd2\x086\xd6\xd2\x08\xc0\x1e\x085\xf5\xa7\x1c\xfdW0\xd9\xb7\'}5\xb8\xac\x9b^\x93\xa9=0\xa5\xbaV^
m\xd5\x9e\x95\xe1\xa9)\xa4\x99\x1e\x95d\xe2\xfe\xba?v\x96%a\x8ca\x94\x00\x08*\t\x19\xda\xf8\x8f\xe2\x1d\xa3\xd5
\x80\x18E\xb6\xcdf\x85m\xae\xce\xee8>l\xd2v\'\xd1\x14\xb4\xed\xa1g\x8fv\xda\xd6\xd1\xb1\xb7k\\-mLV\xe1n\xb8\xd7
\x8530\x1f\x0ckT\xa4`\x102C\x0e\xdd\xbe-y\xcdK\xfd\tw\xcc\xc2\xc7\xdd\x82\x07\xef\xb4\xfe\xde|wO\xb1\xcf\xdbD\x
aa;U\x8a\xd1f\xe9\xf5\xc6;\xee\xf4\xa8>\x8d\xbeqX\xa2\x15V\x8e\x9c\xaf\xc7\x14\\\x8b\x05e\xe5\x85\r\xc86\x00\xf
2\x063_)\x93Z\xd7G\x10\xc9\x15\x9b\x04\xfdH\xf3\xdf\x8a\xe1h\x06\xb8\xc7\\\xa5\x9b.{T\x96\t\x95\xeb+\x02\xc1O\x

1b\x89\xaf\xd1\xc3\x0cr\'/\x86N\x08\x1c\xc6;\x90[\x97~g#G\xd4\x99\x9b\xa9\x05\x80\x02O\xf4n$}\xf0/\xfc+\xd0h\xa
2\xda\xaf+\x9c\xf4\x94\xd4\x8e\xb6\xbb\x04\x1f\xab\xe9\xaa\xe9#\x9a\xa6\x96\x9f\xb2\x06`\x0e\x00\x1d\xbbg\xfb\x
eb\xea\xfd\x0fG>\x9b\xaalas\xe2\xf6\x11\xb9\xd5d\xb1\xae\xb2+\xdcAv;\x85\x8b<\xd1\xbd\x99\xc1\xca\xf4\xbb5*@u\x
a9\x0f\x9f!\xbb`|\xb5\xf7\xdd\x05\x16\x8c\xe7\xbf\xfb\xfe\xfe\xab\x19\xee\xa2\xbe\xae\xc3\xb6\xbb|U!X\x1f\x0cu\
xe94\xe3\x18A.]\x86\xc3\x85\xcf\x11\x18\xed\xe0\x91\xadF\x01yT&\xd7\xb1\xd3xI\xf5\tU#\xe5\xa2\xb4\x80\xefj\x85\
xcaM\x88\xe8ra\xfa\x86\x03\xb6\x98\x04rW&\x0fc\xb14\x823uDs\xdf\x84\x98\xc8\xfc\xc6\x9f,\x04\xe1R\xc8F\x7f\x80\
x92\xa20\x1b\xd2pO\xc3"\xb7\x91\xaa\xba\x1b\xe5F\xfc\xaeo\xd3\x85*c\xf4P\x8f\x99 \xff\x00N\xfa\x13\x9b\xb7\x95`
\xe0Pu]7\x89\x94\xc4\x1c\x05a\x82\x18iw\xb1\xcf\nA\x05 \xb9\xf4\xb6\x02\xac\x82H\xd7\x07\xb3K"\xa8\xfe\xe7IK\xa
6k\x87\nD\x84\x15\x17\xb9\xbad\xa9\x1bz\xb7\xdbr\xf9\xed\xeb1\xc7\xfe\x95#X\xda\xcd\x10\xdb\xf3\x81\xfa\xff\x00
\t\x86JI\xe1B\xd3\xec=\xcbIwt\xdb\x97\x8ao\xb4\xcc\xa7\x06\x92wB@\x1d\xf2\n\x00\xc3\xbf\x825\xf1\xef\x8c\xa4\xd
3\xe9\xe1s\xe4\x98P\xed[\x81\xfdx\xfb\x82\xb6\xb4vx\x19\xfb\xa6\xc2\xba\x8fs\xd0\'H\xb7\x8d\x1dEMe-\x13\xd5Mr\x
b1\xd5`\x16\x05K\t\x96Tu\x8d\x18\xb8\x1c\xc3|\r\x80\x14r\xc0\xf8\xf6\xc6\xc0\xe3\xaf\x84S\t\xad\xae\xff\x00\xe3
DY\xfa\x10l_\xbb\x0ba\xae\x90\n%+\xdb\xbb\x86\xcdj\xb1Um}\xcf\xb6Dr\xd0\xcb%-\x9e\xc7p\xad\x9d\xde\x94`*\x96\x8
eN\xf2\xb1w\xf8\x0ex\x80\xa0\x0c`\xe1\xbd\\:\x87\xea\x1b4f\xcb\xc5\xb9\xc0\n>@\xae+\xbfs\xe5+-\xd1$\xa7\xdb\x8b
\xad\x9b\xfa\x9a\xebn\xb0\xdf\xb6elF\xd3\x1f\xdb\xa6Z\xc9B\xc9Q3\xc0\xe8\x1a\x9d\x86U\xd4\x07\x19bG\xee\xf8\xce
\x91\x8b\xa4\xe9\xc4\x0f{e\x14\xf3B\x85\xd0\xb0}\xc0\xe7\xec\x008\x18I=\xe1Du\xf7mu\x07\xaarY\xba\xb3\xd6\x8b\x
15\xbe\xe1m\x8a\x98\xd2X-\x16y\xdaJ\xa8f\x91C\x07\x98 *X2\x80\x11Kw\xcf\x8dz\x0e\x87\xa8\xd1t\xd8\x9f\xa3\xe9\x
cf;\xdcm\xee"\x9aCq\xed\xbc\x81\x9fq\xc5\xfdRS<\xbd\xd6\xa8\xf6\xcd\x8fcM\xd2*}\xdd\xbdwl\x17\xe1m/[_k\xb9S\x01
K,,\x06HbC\xfa\xd1\xa8`r~"N\x11O\x103f\x9bW\x1fQ|\x1aX\xcce\xd4\xd0o$\xf3\xdb\x1bOo\xc8\x12\x87K\x13\xb8_\xeab\
xa1\xdc\xfbO\xa3\xbb\x84[`\xba\xdc\xa3\xa5\xdb\x9bb\x9d\x9aj\x8a\x85\x98\xa0u\x8d\xdd\x89R\xc5\x9d\x819+\xe7\x9
0\xef\xafb\xcd<S\xea!\x9f[\x1e\xe2\xd6\xdb\xdep\xdb\x03\x16\x05\x03\xb7\xf7\xf0P$\x05\xabf\xe8\xe2o("\x8fv\xf5\
x06\xd5Cw\x8bn3,V\xeb\x84\xe8\xf5\x16\xd6V\xc4\xd22/\xdf\x04\xf9N\xec\x00\xec=\xf5\xe3\xfa\x9f\xa3\xb8\xc3\xa5q
k\x9d\xc9\x00\xd3\xbc\x0c\xff\x00q\xcd\xfd\x94\xb7"\xcaaI\xbbd\x9e\xd1\xbb:\xc5\xd2\x9d\xc3Ak\xb7\xc2~\xddMb\x8
6F*\xf1E\xff\x00\xe4<r\xaf\x11\x1b\xbe\x19\xd4q\xf2\x17\xbf|\x85\xa5\xd2D\xe9\xe0\xd1kZ\xe2\xff\x00\x94\xb8\x81
\xc9>\xc0|\xb4p|\xf8)\xb8\xc8\xa0;+\xed\xb9x]\xf3i\xa4\xea7Glh\x1e\x9e(c\xbe-\xae\x9d$\xa7\x9a\x90\x06\xff\x00v
\x81\x90=Bc\xc2)%W\r\xdc(\xd6&\xa2\x17B\xf7i\xba\x83\xac\x8b\xd9\xbb\xe6\xdcN\x019;O\x9b\xfd\x96\x8cn\x00\x84\x
f6\x86\xbb\xa62\xd6P\x8d\xbf\ro\xd8\xa7\x8aW\xa8\x8a\xb6a$5\xf2a\x1b\xe3\xf5\xc1V|g\x92\xe7\xb7$8\x04gY\xceoQ\x
8d\xae2V\xe1\xdc\n-\xe7\xc7\x1e\x07\xe7\xf5Z\x0c \xf0\xbd^m\xf5\xf1\xdf\xaa\xb6\xefN\xb6\x8a\xcdh\xe2\xb2T\xcdZ
\x14,NH`\xb1\x19[\xbcx>\x10c\xbf\xbet8\x8co\x85\xb2\xea_Rd\x00;\x8e2+\x9f\xe1\x14\xbc?\x90J+pUY-3\xc5[w\x98\xc2
g\xa4F\rM f\x88)\xe3\x87B~\x1flx\xce\x0f\xcb_@\xf8W\xe2-,%\x8c\xd5\xef SK\xa8v\xe2\x85d\x01B\xf9\xc2\xca\xd5i\x
0b\xdeKy\xf0\xbeP\xd4Q]\xb0-;\xb2\t\xc9=\xa3\x91\xbd6\xfe\x84\xeb\xf4OEv\x8fS\x03_\xa7\x94\x10|\xe0\xac)\x83\xd
8h\xb54\x82\xc5r\x81\x96J\xa8Kg\xc3\xc6y\x0f\xed\xaf_\x1c\x122\x92\xa4\x92S\x9bO\x0e\x07\x94\x7f\x1a\xf8\r\x9e\
xfa\xd0c\x0f\x85^\xe9\xbd\x15\xbe\x1a\x8f\x88\xb1\x88\xe3\xba\x9e\xe3M0c\x0b\xac&i\xb7\xe1+\xc62\xb2\x03\xe4)\x
c6\xae\x18\x0f+\xb7\x05\x9e_\xac\x14\x003B\xc7>rs\xdfZ\xc4aS)%\x0e\xf8\xbel\xd9\x88\xa1\xa9%|4R\x0eH\xdf\x88?\x
e7\xe7K\xb9\xeej\xb5Z\xa0\xa5\xebU\x9e\xedDh\xee\x90TZ\xaa\x08\xf8k\xe9#Y\xe3S\xf5\x85\xf1\x9f\xc9\x86\xa1\xcfk
\x86p\xa0\xb2\xb8I.v\xed\xf3\xb8\xe6d\xdb]b\xb2V\x86\x1f\xb3J\xb8M\x13\x9f\xa7\t\x03\'\xe6\x1c\xeb>}>\xa5\xff\x
00\xfar\x02\xac6\x81\x95;\xb8\xbaW\xfaFQ\xc1\xea\xbd\x8e\xa6\xe1\x19\xf3%\xae\xae\x19@\xfc\x90\xe7\xfbk"}\x0fS\
xa3y\xfb#\x07B\xb3\xbd\xcfi\xeam\x96V7\x9d\xa1|^\xdfzK|\xc5G\xe7\xc4\x8daj\xb4\xba\xf64\xdb\na\xaf\x88\x8c\x15=
\x15\xee\xaa\xb6U\xb4=$\xf0O#\x8f\x8e4)"\x80r\xd8\'\x198\xd7\xc8\xbe/\xe9-\x8d\xaf\xd6H\xe7W\xfd\xa7\xe5\xba\x1
5`\x8c\x0f\xaf\x95\xaf\xa3\x9bsC\x07)\xa5\xfe\xd1]|\xda\xf2\xd6l\x9e\xa0V\xdb\xa0\xf5~\xcf\xb8.\xb2!\x92\xa6iD`
\x82!\x85\x83\x91\xe4g\xb0\xe2Gs\xdc\x1f\x98\xe9\xe5\x8a-C\x7f\x13\x108%\x8d\xec<\xd9"\x85\xe3\xecx\x0bL\x8fm\x
0eP\xf4w\x07\xdc\x9d\x1f\x8a\x86\xd9AW{\xb9\xd0I\x9f\xf1U\xde\xd1"\x90\xf0\xc8s3\xc8\xd9+\x12\x87*\x17\x91#\x88
\xf2T\x90\xc3\xdb\xf8~\xaaN\x19\x1b\xb3\xb1\xae\x1c\x11\xc0\xedg\xbe\x12\xafa-\xca\x97\xfd%m{\xde\xd1h\xda\xf3M
\xbb\xe0\x9a\xc8bq5M\xba/Jx\x1d\x86x\x90\xcc\xc4\xc6\xd8\x03=\xc6q\xf7I\xef\xa3\xf0\xeb\xf42z\xdbXL\x9d\x816*\x
fe\xdc\x8c\x1f5\x9e\x12\x13[Z\x91\xedZ:[\xc7Md\xde[\x8f\xad\x15\x92WZ$\x96\xa6\x8e\x1a\xd8czX"F\x91\x18zi\xc5\x
9aN%O2\xc3\x88\x0e;\xe7O\xcf+Y\xae\x1ah\xb4\xa0\x07\x96\x83V\x1cN\x0f4@\x17\xf4\xa3\x8bJ\x80\xe7\x15\x15n\xa1\x
dd\xdb\x82\x8a\x8a\xf1y\xbaGIOr\x02\xe1$\xadS\xc0!.S\xd5A\x82YcS\xc8F\xfd\x80u$\xebfwh\xa2/dm\xbd\xbe\xc0+\xc0\
xe0\x9b\x19$Q#\x92\n+Y\x8c\xa0\x11\xec\xbd8\xb9V\xdd6\xe4"\xa6\xf2\x95.\x9bn\xbe\x9eP\xc2I\xb8`\xb3\xc7\x95(\x1
8\x9cs\x19\xc8q\x8f:!\x13u\x16\xb5\x928\xb62=\xe2\xa8\xd5\xf0\x0eA\xa1\xdb\xb1\n\x8f\x8fp\xa5\xa0t\xf7}\xd6n+\x
05^\xe6\xdc\x95\xc9=\xc2[\xc2\x89,\xae\x00\x8e\xaaN\xdc\xa3d\x7f\x89\xcb\x02<\x90\xa7\'9\x19\xcf\x9e\xd7\xe8Y\x
a7\x99\xb1\xc4\x0e\xcd\xb8}\xd9h\xfa\x10;~\xa8\x05\x84-\x1b\xa84\x1b\xd5w6\xdc\xeaD\xbd<JkF\xe4\xaa\x8e\xdb{\x8
e\xddq\x8d\x85UkqX\x8dM2\xe5\x06Dl\xa5\xc3\x10A\x05\x86<\xe3id\xd3\xcd\xa2\x96\x0382D\x0b\x9b`\x8a`\x19\xda\xee
pM\xd1\x17|e\x12&\xbb\xb2\xbd\xda\x9bKx\xf4\xbfv\xc5k\xa9\xba[(l7\xc5\xa8\xa8\x8a\xa6\xcf\x13\xbc\x96\xf7\xca\x
19i\xf0@L\x9elS#\x18\xe5\xdb\xe1\xce\xb0\xf5\x9a\x8d\x1e\xbbL&\x00\xbaF\x96\x82\x0f\x04d4\xe0\xde\x00\xceS\xf1\
x07\x01G\xb2\xd2,\xd6\x0b\xc6\xf4\xb6\xd6t\xc2\xe3r\xa5\x9a\xd7\x15Dq\xcbQ\rQ\x84\xce\xa82\x0c\n\xb9d\x97\xc6y0
\n\xd9\xcf!\xe7\xce\xc9<z)\x06\xa1\x84\xdeH\x15\xc5\x9eO\x90{VOe\xa2\xc6\x89\x1a,(\xbe\x96\xd8\xb6\xad\x8a\xcdW
M\xbc\xebx-E\xb8VU\xad%|\x89$0\x19\x0f\x07\xf5P\xa9,Y\x18aO\xb0\xf9\x81\xad.\xa7&\xad\xd3\x87@\xdb\xa7\x16\x8b\
x00\x82k"\x8d\xe3\xef\xfb#5\xd1z[j\x8a\x03t\xf5\x93j\xa2S\xed[S\x1b\xa9\xa6q\xeb\xc1z\xa2p\xa1\x15G\tA\x90\xfcd
\xff\x00\x16\x0e{\x9c\xf7\xef\xe9\xfe\x19\xd3\xea:f\xaf\xf1R\xc2\xd3\xbb\x83\x83\xdf\x8a\x02\x87\xed\xf6I\xea\x
1c\xc9\x9b\xb4\x15\xe2\xdb\xd4\xdbutJ\xc2\xcfj\xe0|"\xd1"\xe3\xf0\xc0\xce\xbe\xfb\xd2\xf5\x8f\xda\xd2\xe67\xf4\
xa5\x8b,"\xea\xd5>\xda\xeaE\x05\x0b\x86\x16\x86\x08|\xfd\x8e\xb9\xd3\x1f\x80<\x87\xf6\xd7\xb4\xd1\xeb\xd9\x7f-}
\xad&\xf8H\xe0\xab\xdb\'Rvui\x06{\xb5L-\xee+i\x95\xb1\xff\x002y\xd7\xa3\x87Q\xa7w\xf5%\xde\xd7\x05II\xb8\xb6}Da
\xa1\xbe\xd2\x13\x9fi1\x9f\xeb\xdc\x7f}<\x1b\x19\xe0\x85C`#\x92\xfb\xb7c\xef\x15\xe6\x1c\x91\x929\xe7\xbe\xac#\
x1eTeFU\xd1V\xd6\xc5\xf1F\x14c\xb6XcZ\'!wu5v\xdb2;\x95v\x01\x87\xde\n3\xa09\x80\xa9\xdc\x91T\xedY\xf9sY\x15\x90
{\x91\x8d\x01\xccr\xb3NP\xb3Xj\xc3\x1e$dx\xc0\xd2\xc48\x15e\xc1\xa8\xee\xf4\x871M2\x1cy\x8d\x88\xff\x00]\x05\xe
dq\x16J\x80\x12\xcb\xbe\xeb\xeaM\x99qE\xba\xeeq\x8cvC_!\x07\xfeRH\xd6N\xa5\xda\xa8\xc5\xb5\xc5\x19\xa5\x87\x04(
]\xd5\xd5^\xa6\xd4\xf2\x82\xae\xe0\xb5\x18\x1fzZT$\xfd3\xc4k\xc0\xf5\xf2\xedlF9\x80s~\xc9\xfd9ln\xb6\xae{^\xf1a
\xb8_ \x9a\xdd\xb5nU\xd3\xd6\xabS\xcfI\x1c\xc5\x1e)\x1c\x0cJ\xa3\x90Olrb@V=\xb3\xdb_\x02\xea:-F\x98=\x939\xac\x
ac\xb4\xd61f\x8e/\xc5\xf9+v)Z\xe1\xed\xe7\xba\xe9\xb3\xba\x87\x16\xca\xdc\xd5\xfd7\xb1\xec\xc9)9-Ud\xe2K\xa9\x1
4\xc8Q\t%\xc3\xb1c\x94\x18\xecI\xe4H\x00\x82p\x19tG]\xa4f\xb1\xf2\xddm\x00l\xceM\np\xfa\x93\x82\xde3b\x95\xde\x
d2\xd1\x95?\xb3w\xad\xc7n\xee\xe8.\xf1\xc9\x05\xd6\xa6\xe1E\xeaM\x04\xf3\xaf\xd9\xe9\xe3\x88\xb1\x8e \xad\xe2,\
xbe\x01\xc9c\xdc\x9f\x96\x9a\xd4@\xc9\xa0\xda=\x81\x87\x06\x8d\x92pI\xaa\x17\x8f\xa0\x1f{Y\x92\x82\xe7YU\xe3cmM
\x9f\xd1\xc4\xba\xdd\xaft\xd7KmmlI{\xa0\x9e\xdf\x14dS\xca\xc5\xbd%\x91O)\x00wQ\x95eb\x84\xe4\x12\x0e\x93:\xf9\x
f5}Kd@\xb5\xed\x07a\xbb\xcbk;M\x81\x8e.\xc08\xaaB\xf4\xf6\x84\x97}\xf4\xf3oZd\xba\xee\x1e\x95\xec\xaaJ\xed\x8fS
lH\xf7\x0c\xc5\xc5@\xb0\xcb\x1a\x9eR\xc5\x16C\xb0TbH\x8f\xdd<\xf69{E\xae\x9fT\x1a\xddT\x9bu-u\xb0Uz\x81\xd7@\x9
a\xdb\x920H\xaa<d\x15v\r\xb9Yv\xe2j:{\xfd\xc3fE\xb8\x12\xbe\x86\xfb\n\x06\xbb\xd4\xd0\x80\x86\x12\xb9\x8cD1\xc5
d\xc7%\xe4\x01\x04\xa7?:\xf4\x10\xeft,\xd4ztc\xbfh9\xb1\xcd\x9eH\xefG\xcdg\xbd\xe8^T\x97L\xaf\xdbGmu.\xe1e\x97h
\xdd\xcd%t\xe5L\xd5)\x89s\x90\x17<\xdf(\t\xe5\xc5\x83g\xba\x8c\x0f:\xd1\xea\x90\xeau\x1d9\x93\x19ZKG\x03\x83\xe
7\x81\xfa\xe3\x1eP\x1c\xc0\xe5se\xeb\xc5\xaa\xd3\xd3\xab\xad\xdf|\xc5t\xbcMAr\x9dmB\x02\xa8i\x9c?\xecB:\xba\xf7
\x05pd\xc1l\x16\x039 \xe3\xc9\xd0\x9d/R\x8e-9la\xcdn\xec\x13\xc8\x1b\x89\x14\x7f\xf6\x92\x07~TmsY\xb9l\x96\x1d\
xbb\xd5\x8b\xe5\x8fg\xef}\xe7lj\x9b}=b\x9b\x8c\x14\xd7\x10\xb2\xd3\xb3\xc4r\xef\x14j\xaa\xca\xd9<\x88\xef\xdb\x
e5\xaf\x19\xa9\x9f\xa6\xc2\xedL08\x03\xc0;M\x1a7@\x93`\x8f\xae\x0fd\xfc-;\x86\xe5u\xd4\xde\xa2\xec\xed\x9do\xb7
Q\xec\xdasb\xb9,\xd1\x97AG(\xa7\xab\x85\x98\xb1\x0eO\x1c\x13\xfcA\xf2p\x07q\x9cch4:\x9d[\x9e\xed@\x0fe\x11\xceA
\xc7jwb{c\x9ei:\xdd\x8dpi4\x98\xdbvV\xd7\xd8\x95\xab_\xb8\xbd[\xa4W\xb35M5\x84#UTR\xc4Iii\xa2\x8eF"jp\xf2\x02\x
a4\x80P\x91\xf2\xce\xa1\xfa\xedgQ`\x10\r\x8e\x8e\x86\xfb\xda\xd2j\x81$\x0fk\x8d\x1b\x19\x0e\xcf\xd9\x16J\x000\x
95\x07v\xdeP\\lIAq\xd9u\x17K]=I\x8dh.\x12\xf2\xb8SE\xcd\x82\xb4U\r\x82]\x07\x10\xc8YU\x86A\'\x88\r\xf4\x7f\x84_
&\x9b\\\xe8\x9e\x03\xa3\x90sT\xdb\xae@\xe5\xb7\xce/=\x8fd&\x8d\x924\x10h\x8f\xf7+\xfa\xc9\xb3\xf6\xd6\xe3\x87\x
ed\x1b\x06\xf6$(?il\xbaG\xe8T\xc2~Y \x06\x1f\xcd\x81\xf9\xf9\xd7\xdctz(\xa5\x03\xd3u\x13X\xc7\xf7\xfe\x0e|\xe5b
\xcc\xe7\xb4\x90\xec\xfd\x97\xd9\xed\x17{=O\xa1SC-;\x83\xf1,\x8a@?Q\xf3\x1fQ\xad\x88\xf4\xf2F=\xc1.\\\x1c0\x8c\

xa4\xb8W\xd3\xb7\x02\xe1\x87\xb3\x03\xed\xad\x08\x9b\x8eU\r\x04\xc6\x8fp\xcc\x9f\x0c\x87\x19\xf7:\xd1\x8c<r\xa8
h\xa7v\xcd\xd5\xc4q\xf5\xff\x00\x0e\xf9\xd3m\x95\xca\x85\xb8V4{\xa2I\xd4\x05\xa68>0I\xd6\xf8r\x10i\x08\x87Z\xba
\xac\x9fNL\x9f\x19Q\xe3U*\x84Q_\xd1m\xba\x9a\xa0\x0cq6I\xee\xa7\xbe\xa8\xe8\xfc.n\n*\x9f\xa7\x955 ,\xd02\x83\xe
4/|\xe8N\x8e\xc6T\xefF\xc5\xd2H$\x8c\x89\x1c\x80=\x98hN\x80\x91\x85;\xca\x06\xeb\xd1\xaa\x19``\x91r\xc0\xef\x95
\xce\x92\x97Mb\x81R\x1eVI\xd6]\xad\xb0\xfaq@*\xb7%\xc28\x8c\x991SE\x86\x9ab?\x85s\x9c}N\x00\xf9\xeb\xceu~\x9f\x
a4\x8a-\xd2\x1f\xf2\x9b\x81\xd28\xe1b\xf4;\xe7jT\xdd!\xbePOr\xa0\xa9\xa2--\x11\x87\x8a$G\x88\x1c\x9b\xbf&l\xb1\
xed\x820<c_\x9d>#\xd3\xf5#\xaaxtm\xd8q\\\xd8\xe7\xb8\xac\x8c\x95\xe9\xb4\xef\x887ot\x07\xfbD\xb8\xdb7}\xb7pn\xc
b\xfd\xc6\xf6j\xfb\xdb,\x94\x14q\xc4\x8b4\x84\x01(\nR>~\xde\t\x01\x9b\xeb\xac\xb1\xa1\xf5t\xaf\xd3\xc4\xc6\xc7\
xb7\xe6q$\xd8\x1f\xd3d\x13U\xf5\x03\x14\x8a}\xc3r\xa9\xbd\xde\xfasw\xe9\xe4\xdbR\xef\xb2\xe2\xa0\xdc\xf2W\xcbUh
4\x10\x94\xaa\x9agb\xbc=H\x80\x0e\xbe\x89\x19\x04\x95\x04\x02G\xc2\x0e\xb3\xa0f\xb5\xba\xafU\x92n\x85\xa0\x07Y\
x14\x00\xc94n\xb3\x91B\xc8\xc5\xa1=\x8d-\xcf+\xc0\xd94\xbb\x8e\xa2\xc7e\xdb[\x8a\xb6\x8e{(\x89\xe4\xa1\xba\xcf,
\x90H\x92\x0fI\x01\x8c\xb7r\x83\x91.9*\xe3\xea4F\xea\xdd\x13%\x92V\x02\x1e\x0eE\x03c8\xc7\x18\xaa\xef\xc2Y\xcd\
xb0\xa6S\xab}W\xdc[\xff\x00y\xf4J\xddiZ-\xc6\x1a4K}\x96\x9d\xa5I\x90\x94\x0c\xef!h\xd2(\xe4\xcarc\xc7\x88c\x825
\xae:7M\x83M\xa5\xea\x0fu\xc4\xeb\xf7:\x80\xb3c\x02\x89%\xb9\xaa\xbc\x8b([\xf6\x92\x0f)\x85\xabm\xf5{\xadt\x12.
\xda\xda\x16\r\xc3u\xa5\xa5Z\x1a\xbfZ\xb4Q\xbd\x82\xb1ZB\x81$dy=O\xde\xe2\x00\xf8\xfb\x10\x00V"t\xbd/\xa4J\xd6M
+\xd8\xc2I\xf9won/\x82\x06\xda\xc17\xc7\x9c\xa9\xb7\x91\x85\x1bi\xdd\xb6+\xa4\xb7\xda\x8f\xf1L\xf4\xd2\xc9N\x94
\x15\x95\xb7Kj\x99\x05`FGP\x8c\x15HA\xc4\x84RNT\xf9\xc8:\xd0\x9fM,N\x80\x18\xee\x8d\x80\xd3\x8d\xb6\x08\xe0\x9e
s\x93C?z\xe6\xdb\x81 \xab{oE:\x7f\xd2\x8d\xedo\xbc\xee\xcb\xe5\x92\xebf\x96\xc3Q\x0c\xb0\xd7\xc5\x14+\xf6\x97t\
x89KF_\x94\x8eW\x99\x05T\xb0,\xa7#\xb1\xd6#\xfa\xbe\xbf\xa9\xe8^\xdd3\x1c\xd9\x03\x81\xb0I!\xa2\xcd]\n\xed\x82x
\xe0"\xb1\x94\xec\xad\x02\xe3t\xea\x8d?S!\xda;\x0f\xa7\xd5r\xed\xef\xb3\x19)\xa2\xb9\x87\x8d%\x90\x82\xa3\x9c\x
92\'%T\xfd\xa6\x11\xfc\x83\x92\xc0\x05#\xcf\xfa=0\xf4\xe3>\xa6P$\';H$\x0f\xb05\x93VF~\x89\xc8X\xefP\x808\x1d\xd
1u;\xd7\xa8\xddA\xdc\x14\xbb\x93e\xee\xda*\xfbE\xacIGp\xb6<_\x12U\xc0\x80\xcc\xa6*\x85>\xa69\xaa\x8e\xed\x9f\x8
7\xc1\xf1Y\xba~\x8f\xa6i\xb6j\x19Ow\xb8;\x9a\x0e\xe2\xc8\xe3\x8e\xc6\x95\xa1\x94\xbaM\xd8-VW[M>\xe0\xbb/S\x16\x
9a\x9fqG\x05\x1b-]8\xa62\xcbO"\x07Y\x128\x00#\x89\x00r\x84\xe1\xb9\xa0<\\\x9e\xdd\xd3z\x1fY\xd4A\xf8}<.npG\xcaA
"\x89<\x87\x03\xc1\xa3m\'\x8a\xb2W\xeaa-\xf7\xd7\xfb\x8f\xd3\xf9YN\xe3\xb3>\xec\xaa\xa9\xde\x9d\x0c\xddUu\xd1\x
c14\x82\xba\xce\xb5\xb2\x1a\x98\x1dX\xf2*\x0e\x1aU\x07\x97$\x7f\xda)\xef\x86\xc9+\xf6O\x86\xba\x0c\xed\xd3\xb7\
xf1Qm\x9b\xb88\xba<\x81\xc0\xbf\x03\x1e0)cj\xa5\xd8\xea\x07\xda\xbf\xb6\xee\xe7\xac\xaf\xa7Y\xeba\x02\xa9<\xbf\
x1e,\xbf^C\x07^\xf7K\x06\xc3UT\x93\x7f\xb8+\xdd\xbb\xd4\x96jqn\xdc\xb4\x0bQL\xe3\x0cd\x8c7\x7f\x99\x07\xdf\xea;
\x8f\xae\xbd>\x93V\xf6\x8d\xaf\xc8J>3\xd9>\x87lm\x9b\xd5\x18\x9a\xc9P\xa8\xac~\x1eG\x92\x0f\xcf\xca\x9f\xa1\xd6
\xccqE(\xb6 \x1b\x08:\xfe\x9f\xdc\xe0n\x13 \xc62\xa7\xc8a\xa2\x08\x1c\xd3\x95M\xc5\x03.\xda\xa8\xa7\x1c\xd6\'A\
x9e\xfd\xf4v\xb0\x05k\xb5\xa1\xd9n\xfb~$\x0b\x14j\xa3\x1d\xdd\x9b[ChCv\xe5Km\xbbY\xe4\xe2\x80\x07R?\xf0\xfb\xe7
\xfah\x84\xb1\xca\xb4Sjj\x9a\x0c\xf2Jw\x18\xf1\xc8\x81\x8f\xeas\xfd\xb5\x05\xb5\xc2\xa7t\xde\x8a\xb9\x0c\\B\x0c
\xe7\x00\xf9\xd4\x1ap]XA\xee\x9d\xf3\xb46e?\xad~\xbd,n\xc3+\x02\xfcR?\xfc*2O\xe3\xe3\xeb\xa5\xa5\x9a(\x87\xb8\x
ab\x06\x92\xb1^\xa6\xfe\x92{\xb6\xf0\xd2[\xf65\x19\xb4\xd2\x91\xc5+*\x91Z\xa1\xbe\xa1\x0eV?\xa7\xde#\xcfmbj\xfa
\x8b\xdcKc\x14<\xa3\xb2.\xe5bU\xbb;u\xef\xaa\xb9\xbe\xc1\rE\xc6w\xcf\xdakjY\xdc\x9f\xc5\xceI=\xfd\xb2q\xf8k\xcd
O\xa5\xd5j]y?T\xebd\x8e0\x93]:{c\xdaj\xff\x00\xad\xae\xb0\xa9\x04rH\x90K$\xa4y\x0c\x14\xf0@~@\xb1>\xe7:\xf0\xdd
{\xa1;S\xedt\x94\x07`,\xfeg\xb2wM\xa9\x02\xf0\xa5\xaf;\xba\xcf_C\x15\x92\xd1\xb0\xe5\x10Z\xa9U(\xeaR\xa8\xab\t\
xc1\xc7>@|9\x05\x8e\x17\xe2,Gq\x8c\x8f\x9ck:C\xb4\x1a\x93#\xb5\x16\xe7\x9e(q\xdb\xbdq\xe7\x1c\xad8\xe4\x126\xab
\t\xbe\xec\xa0\xeaN\xe4\xda\xf6\xeb\x8cU\xd0\xd3\xdd*d\x8ekz\xc9A\x08ZJvNIQ4\x87\xba\xcc\xc0\x06U\x1d\xb2\xeb\x
df\xb0\xe5\x8f\xa4wM\xd3k\x1c\xd7\x0fcy\xa2\xeb.\xba-\r\x1f\xd28?@T\xbd\xae{i\xab\xbfR-\x9dA\xb9l\n\xee\xa2\xed
{\x8dl\xea\xd4\xadOG\xba\xa4\x12\xdb\x1a\x96\x912\x8f/\x1av\xe1\'9A\xc6\x00<\xf8\x00\xaeH \x9d.]\x13u\xed\xd2\x
ca\xc0\x0f\xccc\x1b^/$|\xc2\xdbM\xc9\xcf\x04\x92@\x19\xae\xcb\x8c\x9f\xd1I\xec\xdd\xbfS\xb7?H\xcb^\xdf\xa1\xdd)
\xb7\xeb\xe6\xb2C\xfa\xf6\xa2\xad@\xa9\xbc7\xd9\xcf\xda*"\x89\xa31\xe1\xa6Nh$!\xe4*\\(\xe6Tmu\t\x9b\xac\xf8vI\x
8ce\xed\xdcv\x81{Y\x9fh\'p \xed\xc1\xda\x0b[\xf2\xde\x01*\x06\xb88vU\xfd\x05\xa8\xea<}Y\x9ad[\xa5\xa2\x82\xfb_q
\xae\xbaW\\\xda\t\xe7\xa8\x82\x99\x87\xa3S\x12\xb1U2:\x97\x8d\x00\xf8?b\xdd\x8a\x0c.O^\xd3\xf4\xb7\xf4\xe65\xce
k\xcb=6\x80\xdb\r\x05\xf8p\xc7f\xd6\xe7\x0eh\x8c\xd9\xb2h\xda\xfd\xd5XN\xaa:\xa1f\xdd\xdd \xa4\xda]\\\xda\xf6\x
cb%f\xe1\xbeV\xdb\xe77\x1b\x02\x9f\xb2\xc9\x1c\x84\xbdd\xad\xe9\x81\x14\x8b\x0c\x91/\xaa\x030g\x0e\xa8U{(\xee\x
97&\x97\xac\xbfS\xa1\x95\xcellk\xa8;\xe6\xb0@k}\xd9\x16,4\xd3{\x17\x03\xcd\x9a\xdd\xc2\x88\xee\xbdu\x0fc\xfe\x8
e\x94\x1bn\xcfYx\xdbWY\xa0\xb7\xb4\xfc\xa9\xad\x8d (\x92*\x05\xc9\xc8\xc2\x96\x8e5\xecW\x89 \xae\x03>\x94\xe9\x
bdC\xe2\t&\x91\x8c\x91\xa1\xc76\xe02\xe1~3b\xc9\xc8 \xf7\xac+\xec\x8d\xae\x1b\x82evzm\xf7\xd5-\xbb\xd5\x8d\x95\
xba.\xf4W)\xe6Jz\xca\xab\xfb\x12\xd7&\x89c\xcc\x92,l\xfcr\x03!V\xed2\xa3wV\x18q\xc4d\xd2\xf4\xb9\xf4\xd35\xafkh
\x80\xc0\x00n\xed\xd8\x00\x80Oc`{\t\x16\x08\xc8(cD\xdb\xae\xbb\'[\xda\x9fg\xed.\xaa\xd2\xd6m\xfb\xb5,5\x15\xb7!
Q\xb8-\xf4h\xc9\x05=C\xa7\xa5\xf6\x84\x89IT\x90z\xaeIo\x88s|\x1e\xfa\xcf\xd2I\xd45\xbd4\xb2k--\xa6\x1c\x12\xe0\
xd3\xed\x0e\xf2\x0f\xfa;\xa9\xd8\xc68\xedA\xee\x1a\x01\xb4\xf7\x84\x97]\xb1WWj\x9c\xb8%\xa9d(\xd2\xc9\x80\x0b\x
10\xbeyy \xf6$\x9c\x83\xaf\xa9|-\xd3\xba\x9e\x8a\x06\xbak\x0e\xaf\xdb\x90\x0fl,\xddL\xb1<\x908T\xf6\x1d\xeb\xb6
w\r\xc6\n\xce\xa8\xedx\xda\xe0\xa0csX\xc0\xa7\xadS\x8c\x0e|p\xb3\'\x8f\x81\x81Q\xe4(>~\xb9\xa0\x9e9\x99\xb3R\xd
1\x8e\x08\x19\x1f\xef\xef\xddf\xc9\xea\x16\xd08\xf0\xae.\x9d\x0c\xdb\xbdK\xa57\xdd\xb3{\x86\xba^\x18\x17*\x1e1U
g\xf8g\x8f\xb0\x90\xff\x007\xc2\xe4v\xc61\x9fJzl:\x86X7\xf5\xef\xf9\xa4\xf7\x96\x1c\xac\xabq\xecM\xdb\xb0.\x06\
x97p@\xca\x8c\xe4E2\x03\xc1\xc8\xf6\xef\x82\xa7\xf9H\x07Y\xf3hg\xd3\xba\x88\xb1\xe5\x1cJ\xd7\x05\xf2\xd3y\xb8\x
db\'\x15\xb6\xea\xc9#dl\x90\xa7\xb3}\x08\xf0\x7f\x03\xa2D\xf7\xb0\x85\x06\x9c\x16\x9f\xb0z\xa3m\xaf\x0bh\xdc\x8
8\x90\xb3vY\t>\x99\x07\xdf\'\xee\x9f\xc7\xb7\xd7[Pj\xb7\xe1\xc9w1^V\xec\xba*\x98\xbdx#\x05Yr\xa4`\x82>z\xd2k\x0
3\xc2\x16FV\x11d\xdfr\xc2V8\x1d@\xc8\xc8a\x91\xa3\x89p\x8a\xacl{\xedLa\xaa.\xe9\x14~\xeb\xcdT\x7f\x9e\x89\xea6\
x94\x14\xe6\x0e\xb0m\x0bnBWM["\x0c\xf1\xa6^_\xd5\x89\x00\x7f^\xda\x1b\xb5\x910d\xa1\xec\x05,\xb9\xf5\xc3~nWk^\x
db\xa2\x14H\xdd\x88\xa6\xfd\xa4\xe0\x7f\xc4F\x17\xf1\x00~:E\xfa\xd9\xe5;X)X1\xa0\xd9Ji\xf6\x8d\xd6\xb2\xac\xd5\
xde\xaf\x02\x9b\xd4\xef$\x985\x137\xe3\xdc\x0c\xff\x00\xcf\xa0\x1d4\x87/*\xfb\xbc&\x14\xdbCb\xd2\xca%\xaa\xb2\x
d6]\xddpT\xdc*\xcc1g\xe7\xe9\xc3\xdc\x8f\xa1|\x1c\xf7\x1a\x91\x04\x03\xb5\x95\x16\xe5\xf3u*j\xed\xff\x00\xab`
\x82:jd\x18\x8a\x8a\xdf\x00\x86\x15\xfcUq\x93\xf59:\x04\xd6\xff\x00o\x03\xe8\xb8r\xb1\x8d\xed\xb6\xe5\xab\xb9-\
r\x05\xb6Z\x99\xe5<!\x8a5,\xd298\x00\x01\xe4\xf9\xfe\x87\xe4u\xe5u\xfd<\xca\xed\x8c\x17i\xc8\xe4\r\x19N\x93\xa3
\xb7M\xb3\xb4d\xaf\xeal\xb0D=\x15\x8e\x0bE\'\xc3\xc0.\x08y\x1dq\xcaC\x8c\x1e>?\x89\xb91\xd7\x94\xd6|\t\xd3\xf4\
xa5\xda\xbdM\xbaK&\x89$g\xb1\xbe\xc3\xc0M\x8e\xa3#\xbd\x8d\xe1I\xda\x86\xe7\xdd\r[\xb5\xed64\x9a\x8e\x15Y\x04T\
xf4`\xc81\xd9)\xd7\xc7i\x18&r{*\xb7u\x04\x9dx\x7f\xfc\xa3.\xbb\\e\x85\x9f(?k8\x1f\xfe\xf6L\x8dg\xa4\xd0\tJ\xfa\
x83\xb2n\x13X,\xf4\xfdC\xea\x92RVZ\xea\x8dEe\x82\xddvdH\xa6\xf5\x8b\x84\x86\x11\x85\x0e\x00\xe2$Q\xcb\xbeI=\x8e
\xb0\xe4\x87W\xd0\xfa\x84\xdax\xa0\xb6\x9an\xe2\x06}\xa0\x12\\r@\xf0|`\x0c\xa6#{\x1d\x18q+\x86\xf3\xb0\xdcn[\xd
7\xf5\xf6\xcd\xb8\n:\xc3\x1d\x13\xae\xed\xaa\xa0\x12\xd6E]\x1c\xb1\xcd\x1d<j\xa7\x8c\xaa\xa8#FR\xab\xc8\xa1\x0c
\x19@,\x8e\x87V\xc84\x81\xb3\r\xc0n\xff\x00\xa6\t\r\xd8ZZK\x8fbrA\xb3W\x8f!\x874<m\xe2\x90[\x86\xefv\xdb\x11\x7
f\xb4Yk\xe8\xeb\xb7\xb5\xb4\x04\x82[m\xaed\xa7z\xd9}\t\x08x\xb0!!HB\xc0\xa8\xf0Q8\x86*\x0b\x04pj\xa5:V\xb7n\x95
\xf9\xa2\xe0]\xb5\xb6\x05\x1c\xb8]\x9a\xf7\x1a\xbb9\x00\x9a\x00E\x12\xafj\xf7\x05\xf7l\xdd\xf7wT\xfa\x85\xb7V\x
f5k\xb9\xd3%m\x9bm\xba\xfd\xa6\x8d\x0cP\xba\xfaqF\xc5\x9a\x17.\xc1\x04\xaa\x03rg\xe4\xc0\x12N3\x19\x0e\xae-.\x8
3L\xefL\xb4\x90\xe7\xf0\xea\'\x04\xba\xa9\xc2\x81qi\xf06\xf0\x89^\x9d\xbb\xee\xbb\xd8v\xbe\xfa\xbc\xadv\xf0\xa6
\xb8U\xde\xef\x1f\xa9$\xa8\xa3\xa6\x82\xea\x05\xbe\x06\x9a\x88\x1a\\BU\x95d^K\x90C\x89\x02\x10\xf9\xc9\x1a\xac\
xfa\xad\x042E\xa6{\x03#\xdfD\x91nv\xd7\xd3\xf3\xc9\x07\xe9Uv1J\xf5\xed\xb1\xcd m\x1b\xc6\xf6\xdb?o\xd0u\x0e\xad
a\xa9\x82\x13nZ\x98 F\x15qK\x1c|ZY1\'\xa0\xd0\xb1\\\xf1\x00\xb6Kg=\x8b\xa7\xa5\x17\xbai\xf4,\xf6\xd9w$msNCE\x8b
\xdc8\x06\xc0\xa0\x06\x10L\xc4<\x07!\xf7\xc5\xca\xbe\xc9\xd4\xab\x85=d\xd4\xd5\xb4\xd2$Q\xc8\xf1\x90\xe9+\xfajZ
@\xc0\x92\x18\x92Kq>\xe7\xce\x00\xd7\xac\xf8s\xa2A\xaa\xe8\xd1\xbaV\x10I\'8ss\x81\xf9\x0e>\x94\x96\x9aoNR\x01Z}
\x82:\x0e\xa4\xed\x18j\xd1\x03V\xd2F"\x98\xb1\xcb\x12\x07f\'\xea\x07\xf5\xd7\xd8\xb4]:-^\x8cm\xe5\xb8X\xcf\x91\
xccz\x9d\xb8\xd9\xea-\xd3\x98\xe7\x8c\x82\x87\x04\x1f\x97\xcch\xec\xd0\x98\x8e\x17z\xa5\xc8\xbd\xbb\xb9.\x9bz\x
b9j\xed\xf5\x93@\xc0\xe0I\x04\x85[\xfb\x11\xad}!|&\xed\r\xc0=h\xb6\x9e\xb7\xdcn\xb4\x7f\xaa7\\4\xb7([\x00\xc7q\
xa4I\x03\x8fl\xe7\xce\xb7c\xd4\xfa\x8c\xa7e.\xe6\x00m\x0f[`\xe9M\xd2v\x95v\x8c\xf6\xe9_\xc3\xdb\xeeN\x13\xf2G\x
e6\xa0}\x00\xc6\xact\xfaG\xe4\xb7+\x83\x9e8(A\xd3\xcd\xa0\x98j+\xfd|k\xe7\x84\x90\xc7 \xfc;q\xd47C\x00>\xd2\xa7

s\xca\xb2\xe9u\xea\xe5\xb5d\x16:\xcb\xc4w;S\xb6S\x9eRjs\xe0\x95S\x90\xcb\xf4\xe5\xf8cM\xe9\xe2||\x9b\n\x8fir\xfc\xd1\x054\xa0\x86\x91\x0eq\xf7\x89\xed\xa4H)\x80\xef\x08\xfaX\xa2\x0f\x87U\xc0\x00\x9f\x8c\x81\xff\x00\xcf\xfa\xea\x94\xe2h.\xf6\xf2\xaf6\'L.;\x8a$\xb9\xdd\x8bP[\xd8\xe5[\x18y\x07\xf2\xa9\xf0?\x98\xfe@\xe9\x98tFAo\xe1\t\xcf7\x85\xa6[\xad6\xbb-\x1a\xda\xb6\xfd\x02\x08\x87c\x81\xdd\x8f\xcc\x93\xdd\x8f\xe2t\xf8cX\x03Z\x15\t$\xae\xedi\x8c\x1f^\xe1M\x83\x8c\xf0Q\xd8~:\xa1\x8d\xbc\xb9Sw\x85\xc6Z(U\xcb\xad9\x00wPF2?=\x03kn\xd4\x82J\x9c\xdc2\xc9,\x82%E\xe2I\x08\x91yb}\x87\xcc\xe9IF\xe7\x00\xae\x0e\x15f\xcb\xe9\x85\xabd\xd2\xc9\xb8\xee\xb4\xe8\xf7\x89\xa2\xcb\xb9\xf8\xbe\xcc\xbe\xe8\xa7\xc0\xcfnM\xe4\x9e\xde\x07r\x08\x19\xa7aq\xe5T\xb8\x93K9\xea\x855^\xe9\xafZj8\xdeff\x0b\x0c@g\x93\x13\xd8\x0f\xae\xbc\xde\xbe\x17\xea\xe5\xda2\n3]\xb7*\x8a\xc5\xd3\xa8\xfag\xb3\xda\xcfiU7\x1a\xa1\xce\xe1XG\x87#\xba\x8f\xa6;~\x1f\x89\xd3M\xd13A\xa6\xf4c\x14O\'\xea\xab\xbc\xb9\xd6V\x13\xd4~\x9b\xd6\xd3]\xc9\xc34\x82OVi\x9b<\x8b\xe7\xceO\xd7\xbf\xf4\xf9k\xe7\x1dk\xa0F\xf9K\x9c\xdb\xe7\x9b\xef\xcfu\xa5\x06\xa2\x9bI\x16\xf9\xb2U\xcd\xd6\xab\\\\\xa4Ih\xed\xa2\xaa\x9d\xa1\x00(\x9d\xe0be+\xe3\xefw\xf1\x83\x80\x0eq\xaf\x01\xab\xf8nM\x0bd\xd2D-\xa4\x02I\xc9\xa3Go\xdb\x91\xf9\xa7\xe3\xd4z\x84=\xdc\xab\r\xada\xbe\xd7Z\x13\xa8;\x8c7\xd9%\xb9z\xb4\xb4\xd4\x91\x04\x13\xc8X\x89*\xcf\xcd\xb9s\x0b\x8c\x0c\xe4\xfc\x88\x14?\x05ju\x1d\x1d\xda\x8a!\xe7\x0c\x07\xc0\xed\xf4\x06\x80\xfbeL\x9a\xd66`\xd0x\xe5G]l\xb7;N\xf8\x96\xebe\xac\xa8\x8cMqy}\x17\x99\xc0x\xdd\x94\x98]9p*\xc68\xc9\x18$\x95S\x9c\x80t\xcfN\xe8z\xb6\xc1\xe8\xea#*\xf9v\x8e0H\x1c\x83\xc9\xe4\x8f\xcc\xa8|\xcd.\xdc\xd2\x9eu\x06\xdfu\xe9\xe5\x86\xd3\xb8v\x14\x93\xd2Z\xaa\xaa\xdc\\i\xa9\xe5d\x0f#\x0eK\x1b\xb2\xf7\xce\x04\x85\t\xce8\xf8\xf8@\xd3z\x8f\x83^5-\x94\x8d\xd1\xb8\x0c\x90\ta\x1d\xb3\xe7\xb7\x9c\x83\xe5@\xd6na\x03\x91\xfb\xad\x0fat\x8e\xcd\xd4\x8e\x91WWY\xa1\x8d\x95n\xed53\xc5\x18\x04!\x89;\x11\xecA\n\x08\xf9\xae5\xee>\x19\xf8b8zN\xd7\x12\\\xd7r~\xcb?U\xa9q\x9b\x1e\x14&\xf0\xd8R\xda\x84KQO\xc6H\xfb\x17\xc7\x9fl\xebvN\x98\xd8c\xaaCd\xa5\xcb\xb7OwU\xc7c\xee\x04\xaf\xa6\xa7\x12"\x8e\x154\xfc\xb0\xb3\xc7\xfd\xf0}\xc1\xf6?N\xdak\xa6\xc8\xed$\xe0\x81\xf7Q3C\xdb\xf5[]v\xda\xdb\xddC\xb0&\xe6\xdb\x0e\'\xa7\x91Ha\xe1\xe2otq\xec\xc3\xe5\xfe\x84\x1dz\x93\x04S\xb3{Ra\xc4\x1a+=\xbem\n\xcb<\xe7\x94/\xc0\x1f\xeb\xa0\x9d/\x80\xad\xb8\xa5\xf1#\xd3\xb7\xc2\x18c\xc8\xc6\xae\xd6Q\xcfe. \xa6\x96\xeb\xbdL$DX\x95\'\xb6[\xc6\x9ci\x03\x85T\xda\x0b\x8c\x88K\xac\xac\xb9\x07\xb6r4f\xda\xe4m-\xd6D\x900\x1d\xc7l\xe7\x03G\x16\x17,\xdaZX\xd1\xb9\x00\xde<y\xfe\xdaP\x8f\nM+\xce\x9b\xf4\xf2\x9e\x9dc\xdc[\x8e\x8b\x93\x1c5-$\x83\xfa;\x0f\xae{\x0f\xcc\xf9\x03M\xc3\x00o\xb8\x85B\xef\x0bG\x83\xedU\xec\xa1\xa1|\x020\xbe\xdai\xc0*\xd8\xe5:\xa7\xa0\xa7\xa3\xe3\xfb\x1f\xda7\x81\xdb\xb6\xaa\xa0\x1b]$\xa0\xa8\xe7\xea\xbc\xa1T~\xeb\ru\xd2\xa1\xc1\xc2M~\xb8F\xd2\xb292.0\xa1W\xef\x1f\xfai\x19\x1e\tW\x0b\xb7N\xb6\xccw+\x99\xdd3\xc2\x044\xe4\xad8#\xef8\xf2\xc0\xfbq\xff\x003\xf4\xef\xd0\xc6\xdb\xf5\n\xe7\x13T\x11=A\xbdMR\x8dm\xa5r#/\xfbF\xf1\x9d!\xad\x91\xce;Z\xa5\xa2\xc2\x07d\xec\x99\xa9*SrVBVWC\xf6\x14\x90v@\x7f\x7f\xeaH\xf0}\x80\xf7$\x11h4\xa21\xbc\xa9q\x16\xab`\xda\x14\xc6\x114\xb14\xaf\xdc\xb1\x90g\xbf\xe1\xab\xba&\xd5\xbb\x95P@Y\xafR\xf6J=sO4 \xe6@\xcc\x06?-ak4\xbe\xb3\xa8\xa2\xb4\xed\x16\xb2\xfa\xbd\x83&\xf9\xeb\xb1\xb5[\xf2\x84\xc9\r\xb5d\x03\xee*S\xaa\xcc\xff\x00\xf2\xb0\x94\xfeGXst\xe6\xebz\xa3b\xaf\xbf\xd8&\x9b\'\xa7\x12\xfd\x05\x7f\xe9%\x046\xda{E\r\'\xa7KIN\x91C\x1f\xb2\xa0\x00\x01\x8f\x9e\x06\xbdl\xfd:744\x0c\x0c%=^\xeb\x1f\xea\xe7I\x7fV\xdc>\xd9M\x01\xe2\xb9`@\xc7\x83\xacmGI\x8d\xa6\xda\x11\x99+\x80AZ\xb6\xdd\x16\xee\xb1\xd5l+\xb4\xcb\x08\xb8A\x8ai\xa5\xfb\x91O\x90cf\xfa\x07\x03?\xca[D\x8bD\x1f\x11\x8c\x8eUw\xd3\xb7\x04\xd3\xf4,\xbf\xae\xd8\xdc\x95\xfd3\xbfD\xf0\xa5\xd5\x03S\xa4\x9d\xbd*\xa8\xb2\xb2FF;3\x03\x9f\xc6#\xf3\xd1\xba0\xf4^\xf8\x1f\xc9\xca\xb6\xa3\xfe\xe0\xaf\xba\xd1\xd2\x813\xad}\xba\x94\x14?yx\xf69\x1f\x86\xb5\xe4\xd2\x83B\x90\x03\x97\xe7}\xc5\xb6\xeal\xb7Y*!\xca\x85\'*W?\x96\xb2d\xd2\x16\xcb\xb8p\x8e\xd9\x01\x14\x9et\xc7\xa8\xb7}\x85v[\x95\x1cm%<\xc3\x15\xb4D\xfc3\'\xb7\xe0\xc3\xd8\xfe^\t\xd3\x1aI]\x03\xbe\x8a\x92S\x96\xe1Sk\xdb\xfb\xd6\xcd\x1d\xfe\xc9(\x9a\x9a\xa9y+\xaf\x959\xc1R=\x889\x04}5\xe8X\xc6H\xcd\xcd(\x1b\x8b]K<\xdd{"\xa2\xd8E\\D\x85\xce9\xe3\xb6\x0f\xcfCtde\\8\x15:\xf4F\'hf8u\xee\x06<\x8f\xa6\x86\x1aU\x91t\'\x89\n\x13 \x8e\xe0\xeamr1\x92\xa2\x0f\x8d\xd4\xf1a\x95\xd3\x15k\x90[\x17i\xc7q\xae[\x95t%\xe9\xe19Tc\xd9\xdf\xd8\x1f\x98\xcfs\xf8\x0f\xae\xa6({\x95\x04\xd0Z]\xb2\x8e\xa6\xad\x879\xc6OuP\xbar\x8a\xa1\xe1SZ\xad\xd5\x14\xd8S0,\x008\x03\xc6\xb8\x144\xd2*9=~r\xce\x80\x81\x92\xd8\x07\xf2\xd5r\xb9\x01}\xb8U+}\x9e9c\x1c\x8eO/\'\xeb\xa5\xa4\xde\xe3JG*`\x8a\xcb\xbd\xd5m\xd1L\xa6Y\x9f\x87.>>\xbf\x80\x19?\x96\x96r\xdeU\x89\xa5_p\xb8RX\xadil\xa1!R(\xf8\xc6\xa0{|\xcf\xe3\xfez\xbc\xcf\rm\x05\rh*b\xcdC6\xe2\xbe\xac\x15NE4x\x92r;\xfc9\xf1\xf8\x93\xdbY\xb1D\xe9\xe6\r<\x0c\x95w\xbbhZ\x8d\xb2\xdae\xe0\xd1\xc6\x04\x8c>\x00Gd\x1f\xf6\xd6\xab\xc6\xe0)\x04e;\x9a\xd1\x05=\xb8D0\x02.ZC\xef\x8f\'U\x91\x8dcB\x91\x92\xb2\xed\xe9\x14\x15W\x99**\xa4\xc4\x0b\x99e }\xd8\x91r\xc7\xff\x00Ho\xeb\xac\xd7\xb0\x97\xd9E\x1cR\x94\xfd\x19\xb6\xb1\xbcuy\xef\xb5\x11\x7f\xb8\xa6\x9a\xa5\xf2\xbe$\x91\x80\x1f\xd43\xff\x00]!\xd1\xa1\xf5:\x83\xe6#\x8b\xfd\xd5\xe6?\xf4\xe9o\xd7\xab<f"\x92/\x85\xcf\xc3\xafJ\xe8\x9aE\xa5\xc15J\x1b\xa8\xbbU/\x16\xc7\x94\xd3\x8fV68Q\xfb\xc9\xa5_\x08\'*\xe1\xd4\xb0\x8d\xc5m\xa9\xb1]\x8dW\x0c"\xb0\nq\xe7\xe9\xa4\x1d\x1b\x98\xf2{"\x03iwR kv\xe4\xb5\xf5Ok4\x94\xd2\xd7?\xa93\xa8\xca\xc3p\x87\x1c\x8f\xd0H\xb8\x93\x1e\xe4\xc9\xf2\xec\rLF9\x9b;Q#;\x98ZW\xe9\xcd\x93\xb9v\xff\x00W\xf6%=\xee\x87\x83\x19\xa3\xe3Q\x0f,\xb4\x13\x0f\xbf\x19\xfc\x0fo\xa8 \xfb\xebu\x85\x93\xc6\x1c;\xa5\x8d\xb1d\x9dS\xe9d\x91\xc9Y\x08\xa7\x0cyeX/p}\x8e\x81,"\x8e\x15\x9a\xe2\x167z\xda\xd5\xb6j\xb5u\x84\x80\xa3\x0c\xa4v\xd2o\x83i\xc2\xb86\xa8\xbaQ\xd4\xda\xdd\x81t\xf4+\x11\xa6\xb6U8\x15\xb4\xe3\xca\xfbz\x8b\xe3\x0c\x07\xf5\x1f\x91\x06\xd3J\xe8_G\x85\x0ehr\xd8o1Z\xeb\xedks\xb7\xb2TQ\xd5F\x1e9\x17\xba\xb0:\xd7\x14\xe6\xdfd&\x9d\xa5An=\xafE*5M\x1ca\xe3+\xf1*\x9f\x8a3\xff\x00M\x05\xf1\x8a\xb4@\xebSS\xd1\xcbn\x94:\x06x\xdb\xdc\x0f\xbas\xef\xa0\x96\x00\xae\x99[\xaa)\xab \xe2\xc9\x8e\xdd\xf2upmr\xa4\xb1RE\x1d<p\xd2\xc2\xaa\x91\x8c"\x91\xe3\xfe\xfau\x08\xb9S[\x98FR8Uy\x11\xf18\xec\x06\xadb\x94\x13\x84\xda\x9aaLY\x9a\xaf\x0b\xe4\xf1\xd5K\x8d*\xafS\xd5H\xe8\t\x19\'\xc9-\xf7F\x84I\x05r\x9b\xbd\xd4F\xb5M\xff\x00\xddq\x0b\xe4\xe7\xcf\xd0ii\x1cl\x8bV\x01x\xda\xac\x94\xb5\x12\xde\'\x94\x82\xb1\x94\x89\x0f\xee\x8f\xdeo\xf4\xfe\xba\xab=\x82\xca\x92-\x7f\\\xee\xd5W\xcb\x98\xa7\xa3No6#\x85\x17\xc0\xf9\xfeZX\x87K&\xd0\xac\xda\n\xff\x00dmZ[l\nIi\x18\x00]\xbf\x8d\xbd\xc9\xff\x00!\xad\x16\xc6\x18)\t\xd6\xe2\xaf\xacpG\xe9\x87\x8e<\x05\x1d\xf2;\x9f\x9e\xadM\xac(\x02\x97\xdd\xd5 \xa7\xb2<|\xc02)\x1f\x964\x19\x9b\xecR\xd5\x8a\xf5)\x9b\xf5\x1dHi;\xd4\xbaS\x0fb\xc0\xb7&\x19\xfa\xaa0?\xf1k7Tv\xc7\x84V\xd1T?\xa2\xcd\xa9\r-\xde\xf6\xa9\x87z\x98\xe0F\xfeU\x04\xff\x00\xa8:/I\x8e\xa2s\xc0\xe5V^\xcbR\xbc\x18\x1c\x88\xcb\x1e\\>\xf7\xfakY\xcc4\x84AKo4\xd4\xf5\x10(hG&\xc2\x82=\x86;\x7f}\x0c\xb0\x13\x95\xc0\x95\x8f\xf5\x8b`\xacN%0\xf1S\x19!\x80\xfd\xef\x9e\x83,vp\x88\xc7W+7\xa3\xa2\xfdcGY\xb3\xae2,q\xd6\xf6\x8eIG\xc3\x1c\xeb\x9fM\xcf\xc8\x02H8\xfd\xd6m*\xf8\xf7\xb4\xb5\xc8\x80\xe3\x08\x9f\xd1\x9f\xa8U=;\xea\n\xed\xfb\xb4\xbe\x95\x05\xd2aM[\x0c\xbf\xf83\x83\xc5_\xe9\xdf*~y\x1f!\xa5\xba|\x8e\x86M\x8eV\x95\xa0\xb7\x0b\xf4v\xfc\xdb\x02\xe3F\xd5T\xf0\xf3n8\x90/\x96\x1a\xdc-\xb0\x97X\xee\xf6\xd9\xf4\xb3\xfa\x7fh\xa1r\xdc\x8a\xbbc\xfb\xff\x00\xf3\xe5\xa1I\x10*AY\xb6\xe8\xe9\x95E\xbeg\x9e\xdf\xc9\x979\t\xee\x06\x95|\x07\xb2\xbb]\xd9u\xe9\xc7R.;\n\xaaM\xb7\xb8\x92I-\x93\xbf\xc6\xa5r`o\xe3_\x98\xf9\x8f\xcf\xe7\x92A)\x84\xedr\x976\xc6\x15U\xeeh\xa3\x8c\\m5I,\x12!hg\x8d\xb2\x18\x7f\xae\x9a\x94\x02\xdbb\xa0\x14\xa7\xeb*\xe8\xeb\xe3\xf8\xc0\x8aU\xf0\xe3\xee\x9f\xc4h-%\xe3<\xa3!\xd23\x04\xd9\x91\xb8\x86\xf0\xe9\xe0\xea\xd5K\x95\r\xaa\xe3\x02 \x8a3\xdf\x1e\xdet\xc06\x86k\xba\xa2\x82\xeb\x14H\x8948\xe0\x07`<\x92}\xf5\xd6\x17R\xe9\rx\x9c\x12X\xe4\x9e\xdd\xbb\r\r\xc4\xae,^\xeb\xab\xe1\xa5\xa3\xcaLr\xe7\xb0\'\xb9\xd5O\n6\x14\x9e`\xb1\xd4}\xa6i\x14\xc8\xc3\xb0S\x9c\r\x04\xb7\xb9R\x86\xb9\xde\xa3\xa5\x02\x96)\x0f\xa8\xc326|h\x12>\xf0\xae8U]-\xdb\xcfTM\xdeua\xeb\x0e0\x86\xfd\xd4\xf7?\xf3\x1f\xec;y\xd3ZXv\x8d\xc5\r\xf4\xb5KU%<l W\xceW\xf7}\xb4\xd3\x9bd\xaa*+t\x91\xc6\xa2$<\x7f\x84\x01\xe7U\xa0\x02\xe57\xbe.q\x8a|\x89\x08\xe2H\xe3\xa0J\xdb\x16\xb8wY\x07P*L\xcd\x04,\xfd\xa3\r&\x01\xf2\xc7\n?\xa6\x0e\xb2uc\xda\x02#\x02\xd0\xff\x00F\xf5h6\x1a\xcc\xa7\xbdM|\xce@\xfeW\xe0?\xf6kW\xa703N(*\xc9\x92\xac\xaf\x15\x08\xb3\x16\x85\xc1*\x87\xd4R}\xb3\xe7N<9\xd9\xec\xab\xd9z\x824\xa9\xb7\x02[\'\xe1\xc1o}V\x83\x85\x85\xc9F\xfa\xb3%U\xbd\xe1\xaa\x84\xb2H\xb8\x07\xf8O\xcbU\xa0\x1b\x95\xcb\x04\xdd\xfbzKMa\x88\xfcj\x18\xb4n\x0fr\x0f\xfd\xf4\xbb\x99f\xd5\x9b\xc2\x85\xeaE\n\xd3\xee\x1f\xb6F\n}\xb6\x15\x9f\x9eq\xc5\xf0C\x1f\xcd\x87/\xff\x00\xddfj\xxd9\xb2P\xe4f\x9e\xcb\xf5WF\xb7\xac\x1dD\xe9\x95\r\xce\xa2`\xd3\xa4\x02\x1a\xc0?\xf3\x14`\xff\x00]l\xe9\xde%\x88\x14\x17\x82\n_\xbb\xacI\x0b5<\xa0\x15\xf59F\xe0{|\xb4B+\x95U3-\xa2\x96\xa2c\x15D*\xc5\xb3\x96\xe3\xe7U-\x04\xae\xaaH\xf7\x0fI,\x97\xe8\x8a\xfaA]\xfe\xe3\x01\xdf8\xf1\xaa~\x19\x85\xc4\xa9\xdeB\x89\x9b\xa7W\xfd\xac\xcd\x05\x15\xd1\x96\x02\xe7\x9c\x12/%\xcf\xe1\xec\x7f\r@\x8d\xcc\n\xfb\x85$W\xebU\xfa\xda}sN\xae\xb9\xc24M\xdb\xf0\xef\xa1\xbe\xd9\x93\xc2\xb5\xda\x02\x92\xff\x00:?\xd9\xea\xa21\x95\xf3\x1b\xae\x0f\xf7\xd46Pp\xa7iN\xedW\xd8\xa9\xd5cG\x06B{\x93\xed\xa2\xd9\\E\xa76\xeb\x83LH\x91\xb9r?x\x9f:\x9bR\x9a\xa5J\xfa^\x9cq\xa9|\xe7\x03\xdbR2\xb9r\xaa\xac\xf4\x81\xe5\x97\x91\x88R\xdf!\xe7T"\x94^R\x1f\xf1e\x15MmD\xb4\xf2\x07Jf\xe0\xd2\x03\xd8\xbf\xb8\x1f\x87o\xeb\xa5\xcb\xc1\x18P\x01\x05q\xdb\xabS\xbb7\x146\xc4r}g\xe73\x01\xf7#\x1eO\xf9\x0f\xcchq\xb3\xd4\x92\x978\xaf\xd

```
0\x1br\x92\x9a\xdfL\xb1\xaa\x04P\xb8A\xf2\x03\xc0\xd6\xb0hhA\xb2J\xa5\xb4\xf6\xfd\xa6\x02\xe5F9j\xbd\xc9\\\x9a%
G\x15\x12#v\xf0X\xff\x00\xa6\xa0\n\xe5w\x06\xd4N\xf9\xaa\x03\x9b\x13\xdf\x90\xed\x9e\xda\x0b\x85\x85\xdc\x9c,\x
afy\xdc\x04\x93\xcc\xeb\x8e#\xe0S\xff\x00\x08\xef\x8f\xcc\x9dfL\xd0\xf3\x94v\xf0\xb4\xae\x80\xdc\x10l+||\xc6G\x
acH\x07\xc9\xf5\x9f\xce\xb54x\x85\xa8o\xe5S_\xab\xb8\xc3%Z\x81\x9e\x04v\xf2;\xe9\x87\xe0PTL\xec5\x11Kic\xcb<\xb
8\xb0\\\xfb\xea\x1a\r\xd0\\\x8f\xbaBe\xa1ji\xfb\xab\x1c\x81\xf2\x18\xd5\x88i\xe5A ,k\xa96a\x0c\xcf\xfb,\xc6X\x8
c\x81\xf7\x0fb?\xcf@"\x95\xc2\xc9\xfa\xbbI\x14\x96KE\xd2%\xc1Ie\x82P\x07s\xe0\x8f\xff\x00\xadg\xeb\xda=0Q\x19v\
xad\x7fC-\xe7\xf6;\xdd~\xc8\xa8\x94\xfau0}\xa2\x14?\xc4\xb8\r\xfd\x8e\xaf\xd3\xdeKv\x95\xd2\x0b[~\xe6\xb7\xcbYI
$\x10ad\x1f\x14LG\x93\xed\xadB, \xa8)\xeaXL\x1ad\xe0\xdfvB\x0f`~z\x15VT\x9c\x84\xce\x86!Q\x10\x8e\xa5\x17\x98\x
19R\xbe\x18|\xf4V\xb8\x1c\xa8 \x84\xbbs\xed\xc8\xebi\x9au\x001\x18l|\xfd\x8e\xaeE\x85\xd7Ef{\x92\xd8i\x0c\x94u1
\xfc\x0e\x08a\xf2?O\xect\xb4\x8d\x04R 9Q\xd5\x14\x10\xd5\x03KV\x80I\x19\xc2\xbe<\xeb5\xed\xda\xf2\x028 \xa4\xb6
Z\x96\xca\xbc\xcb\x9c`\x81\xf5\xd3m\xca\xadeU\xdb\xab&d\x19u@N\x00\x03\xbe\x8a\x14#\xab\xaft6\x8a/\xb6\xdcj\x92
\x08\xbc/.\xe5\xcf\xc8\x01\xdc\x9f:\xe9\x1e\xd8\xdbevO\x0b=\xde\x9dR\xb9]\xd1\xa8\xad\x10\xb55;\x02\x19\xf3\xfb
I\x07\xd4\x8e\xca5\x95>\xa8\xba\xc3xElu\x92\x87\xb5V\x1a;\x0c4jH\xc8\xe6\xf8\xf7\'?\xe9\xdb\\\xcc2\x95\\\x05\xa
dc\xf4w\xb3\x9a\x8az\xad\xc7R\xa1\x9aYV\x18\x7f\x95W\xbb\x7fS\xfeZ\xd0\xd26\x9b\xb9\t\xcbd\xa5\x99\x93\x86;\x91
\x9f#M\xb8\xd9B\xc8*\x82\x82R%Y\x193\x95\xf7\xf1\xaa\xf7\\\x02\xed,\xcf\r+&rP\xe7S\x9a\\E\x15\x01\xbe\xaa\xe4Y\
xbfh\xc7\xc9r?\x01\xe3K\xc8}\xa5p\x19\xa5\x98n\xf9\x99\xa0RPd\xfd\xedf\x91\x94n\x15\xaf\xe8\xdd|\x91\xf6e4R\x90
G\xda*\x11O\x9f\x12\xb1\x1f\xe7\xad=\x11\xb8P\xde\r\xab]\xc7s\x968\xa5\xf1\xc3\xd3n\xe0\xf8#L\xb9\xd4\x15Ch\xa7
\x1b*\xbf\xd5\xb6\xa2\x93\x96)\xf0\xb0?\x8f\xfd5\xcd4)A$\xaa\xb9\x0f\xabj!s\xdf\xbb\x1f|\xea\xc4\xd9QV\xb3>\xa4
\xd2\xf2\xa5\x92\xa2\x13\x89\x08\x1f\x83}?\xb6\xa8\xe1aXr\xb1~\xa9*\xc9\xb4\xde\x13\xd8-lr\'\xcd~\x17S\xff\x00\
xb8i\x1d[wEH\xcc\xe5M\xf4ov\x7f\x84:\x95k\xbe\xcc\n\xc2\x95B*\x86\x1f\xc0\xc0\xa9\xcf\xe1\xc8\x1f\xcbHi\x1fR\xa
b\xbc\x12\xda_\xaf\xee\x15\x02h\xd6h\x9f\xb8\xc1Q\x9e\xcc5\xe8\x87\xca\x95\xa3k;\xdf\xf0\xc9n\x90\xdd\xe9\xa4\x
fd\x8c\x84-Dg\xf7O\xb3\r\r\xea@\xf2\xbemK\xf7:\x81H\xee\x0eFc,|\xfd5\xcc"\x94\xd1T\xf5\t\x05M)`0\x1f\xb1P<\x1d\
x15\xb9*\x99\xb5\x9d\xf5\'o\xa8\xa7\x92A\x85(\x98,\x07\xf7\xd5^7\xe1\\,~\xe1UUOXj\x90\x02\xb8\xe3"\x9f\x98\xf7\
xd6d\xe0nFk\xb0\xbf\xff\xd9'>,
  <tf.Tensor: shape=(), dtype=string, numpy=b'daisy'>)]
```

In [ ]:
```
rdd5_partitions.take(1) #This is the first element of the rdd5 showing the information of the repartitioned RDD
```

Out[ ]:
```
[(<tf.Tensor: shape=(), dtype=string, numpy=b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x01\x01,\x01,\x00\x00\xf
f\xdb\x00C\x00\x02\x01\x01\x01\x01\x01\x01\x02\x01\x01\x01\x02\x02\x02\x02\x02\x04\x03\x02\x02\x02\x02\x05\x04\x04\
x03\x04\x06\x05\x06\x06\x06\x05\x06\x06\x06\x07\t\x08\x06\x07\t\x07\x06\x06\x08\x0b\x08\t\n\n\n\n\n\x06\x08\x0b
\x0c\x0b\n\x0c\t\n\n\n\xff\xdb\x00C\x01\x02\x02\x02\x02\x02\x02\x02\x05\x03\x03\x05\n\x07\x06\x07\n\n\n\n\n\n\n\n
\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\xff\xc0\x00\x11\x08\x00\xc0\
x00\xc0\x03\x01\x11\x00\x02\x11\x01\x03\x11\x01\xff\xc4\x00\x1d\x00\x00\x02\x03\x01\x01\x01\x01\x01\x00\x00\x00
\x00\x00\x00\x00\x00\x06\x07\x04\x05\x08\x03\x02\t\x00\x01\xff\xc4\x00?\x10\x00\x01\x03\x03\x04\x01\x02\x04\x04
\x04\x04\x04\x06\x03\x01\x00\x01\x02\x03\x04\x05\x06\x11\x00\x07\x12!1\x13A\x08\x14"Q\x152aq#B\x81\x91\t\x16\x1
7\xa13Rb\xb1$C\xc1\xd1\xe1\xf0c\x82\x92\xf1\xff\xc4\x00\x1c\x01\x00\x03\x01\x01\x01\x01\x01\x01\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x02\x03\x04\x01\x05\x00\x06\x07\x08\xff\xc4\x005\x11\x00\x02\x02\x02\x02\x01\x03\x03\x02\x04\x
05\x03\x05\x00\x00\x00\x01\x02\x00\x11\x03!\x121\x04\x13"A\x05Qa\x062\x14q\x81\xa1\x07\x15#\x91\xb1B\xd1\xf0\x1
6Rb\xc1\xe1\xff\xda\x00\x0c\x03\x01\x00\x02\x11\x03\x11\x00?\x00\xc4\xd6tx\x94zz\x12\xc9\x01i\x18#\x1a\x8e\xa5\
x80\xfd\xa1SS\x90\xe4C!n`\xe3\xc0:\xd1\xa3<KT\x06\xbdj(2\x04\xa6\x15\xf4\xa4x_\xbe\x99q\x0c\xcf\xf7\x96\x1bmz\x
c7\x84\xf9i$##\'\xf5\xd2\xc87\r\x18\x85\xdcg\xb5\x0e\x85s\xc1\x0f\xb6\x11\xea\x9f\x1fc\xfak*\xd8\xc2\x15U(\'X0\
xe1JS\xec\xb5\xc3\x99\xf0\x0e\xb3\x8f\x1d\xcc\xa0\x0c\x1d\xbc\xe8\xaf>\xd0a\x97I \xe3\xcep5\x13\xe4\x05\xa5\x00
X\x81\x91,\xba\xb5\x12\xaf\xf8\x99_!\xc8\x103\xa7cF;\x93\x91F\x17T\xef\xf6\x15E0\x96rR0H\xd3\xcb\x18\xb7\xda\xd
9\x8a\xfb\x8aC\x95\x07V\xf4\x18\xeaP\xcf\xbau\xe5m\xc5\x8dJ8T\xf9oJ\xe2\xeb\x04w\xd8#L\x04\x98U\xcau\xba,vfC\x0
bq\xb4\xe4\x82{N\x9e)\x86\xe08\xa8\xb8sm\x9an_#\x9e$\xe4u\xad\x18\x17\xb9\x97&&\xd3\x93\x00\xf2c\xfac\xce\x94\x
eaW\xa9\xed\x9e\xe5\xa2QVDL6\xb5t<gS\x93\x90\x9a\x9b\xca\x0cT\xd1\\\x93#\xd2\x01\xc5e^F\x83&<\x84B\r;5j\\s\xe2\
x96V\tI\xef\x07\xc8\xd4m\xeb\x03BQ\xc9O\xcc\xa5\xa8\xd8Si\x8er[d\xfb\xe7\x18\xd3\x107fx\x15\x12e\x10\xbc\x15\xe
8%J\xca|\x8dR\xa4\x89\xed4\xbbr\n\x14\x80\x0bd\x93\xe7Vb\xc9K\xb99\x1b\x9e\x1d\xa1\xc7P\x07\x81\xc7\xeb\xa6\x97
R*\x0b,\xe7\x16\x88R\xa5\xf0\xc2G\x9c\x8d!\xac\x18B\x80\x95\x15\xaaL\xc0\xe6}E\x11\x9e\x80\xd1\x077\x04\x80L\xb
4\xb5Yu\x84\x10\xee{\xf0s\xde\x98\xb9E\xd4\x12\xa2j\x9b\xe7o7Si\xe5*\x16\xe6m\xed^\x82\xea|\x1a\x84%\xa1*\xfbaX
\xe2\x7f\xa1\xd7\xb2\xe0t\xddM\xc5\x9dX\xf7)\x99\xbe\xdan\x12\x90\xbe\xcaGY\xd4\xbb\x95\x83b/\xef+\xbaT\xb9\xbf
C\x85(\xf7\t\xd6\xad\xdcSlT\xa3\x81}\xbf\x02JX\x8e\xea\xd2\xa1\xeeU\xa7q\xb8\x17f2\xecM\xd8\x9d\x01i*\x92Jx\xf6
\t\xf7\xd6\xae0^\x15\x92!T\xfd\xecS\xd1\xc9q\xc0\x07\xb6O\x8dfLt\xba\x82\x18\x83\xb8\'R\xdeDFyn!\xd0\xbc\xfbc:\
xe2\xbf\x8e\xe7-\xcb1\xe4Z\xdc\xe9R\xde\x16\'A\n\x08NH\x19\x1c|\xea\xc5u@\x14\xc4\xb5\x93\xa83K\xb8\x8d^\xa8\xa
6\xd4\xac\xa5\xd3\x82\x07\x81\xa7/\x16\x11#\x91\x8e;#l\xe8uZ`\x0bJA(\xc9\xcf\xbe\x8dB\x03\x0c\xa3\x15\xeaR_\x96
m"\x8d(\xb1\x0c#\xd4\x1de>\xda\xf1+z\x82\x15\x84\n\xb9\xf8\xc7\x80\xa5\x82\x14\xb0\x9ccLSBi\x1a\xdc\t\x8a\xd2\x
e5\xbc\x12\x96\xfe\xa2}\xcf\x8d;\x96\xf5\x17[\xb9b\xfd\x1b\xd1\tuH\xe4}\xd3\x9d7\x88n\xe6\x13S\xf2\xa2\xc5y\x03
-\x80}\xf4C\x16 z\x8a7%B\xb7)\xf2@Qd\x1c\x1f cN\\8\xcf\xc4\x02\xe4Kx\xf4JS,\x1e-\xa7\xa1\xdfZNO\x0f\x1c5\xcaz\x
837u\r\x89@\xfc\xb3`\xf7\xec4\xa3\xe3\xe3\x02412\x82\x9db<\x97\x14\xf8\x8f\xef\xed\xa9\x8e!\xca\x10b5,\x17o6\xd
8\r\xfag8\xec\xe9\x9e\x9d\rL\xbb2,\xeaL\x96\x93\xe9\xa1\xb3\x8d\x07\x07"\x11 \x89\x15\x9aS\xe0\x1ec\xf6\xd3B\xd
fp>g\xef\x90J\xdf\x01\xc6@\x1fr5\xe3\x8fSgsBe)\xcbm\x01\x93\xfc\xbaO\x03s\xd3r|6|en\xc4\x8a\xc4\xfd\x9c\xdf\xd5
Tk\x8d&\xddU*\x9fnLy\xb7XR\xd9YJR\x1bq<\x14\xae\x94\x80\x02{\x04}D\x01\x8f\xa4\x0eXq}\xce;\xa8S\xec5\x16\xbf\x1
2\xdf\x0e\x9f\x87Y,|@\xed]\xb6\xf4+rt\x97\x99\xa8\xd19-\xc5R\x9do\x19_"?\xe0\x9eX\x19$\xa0\x8c\x12A\x18\xe5y~\x
11J\xc8\x9d\x19\xd0\xf1<\xd0\xfe\xc6\xeef\xfa\x81C\xee\x92\xa5g\xaf\x03\xef\xa8Qu\xb9[\x13)\xa4P\x9d\x12~a\xa2|
\xf6\t\xd3\x0e\x96\xe0\x8e\xe5\xc5\x1do\xc7h`\x9c\xfe\xba\xf2\x9b7\x08\x89s\xcdr\x1b\x19\xf3\xa7\r\xea\x01\xeeP
\xd6\xdbT\'\xcb\xaaWY\xecj\x1f!H\x06\xa1\xa1\xa3+\x95s4\xa4\x16P\xe6T\x0e\x02s\xae\x0ef\xc9\xce\xa5\x01,\\%\xdb
\xf8\x13W%/\xa0\x13\xf5\xe4k\xa9\xe3\x93\xc2\x02\xacz\xd3\xaez\x85\n\x81\xeb)\xa2\x92\x11\x80@\xd3\xfa\x17\t\x8
d\x08\xb2\xa8n-j\xb3^y\x05\xc0\xa4\xf2\xef\x97\x9d \x13\xceyT\xd4\x89r\xdc\x8c|\x88`\'\xeb\xf2\xac\x9dV\r\x08\'
pf-a1d\x17PR\x00\xfd25\xe0\xc6\x00\x12\xd5\xaa\xf32\x9b\xedI#>\xfazd\xad\xdcS\x89\x1a\xb3YD \x16\xdar\x9cx\x1eF
\x9d\xccT\x1e3\x8d2\xedq\xc2P\x140;\xec\xeb\xc33,\xf1E=\xc9S\xee\x99\r3\xf4\xa3\xdb\xb1\xa6\x1c\xacD\xce\n%O\xf
9\xca;\x8b\xfe\'X=\x81\xa5\xb3\\5\x1a\x96\xf0n\xcat\x96\xf0\xcb\xa9\n\xf7\x1a\xcd\x11<z\x9d\x971\xa7\xc1R|\x9fm
o\x19\x82\xbe\').\xb3\xe8\xe5*\n\xc7\xb1\xd0\xa5\x83\xb9\xe6\xeax\xa6!2\xdcPS\x03\t\x1d\xf5\xa6\x8a\x8a\xf7N\x1
5\xe6#\xc3W\xa8\xca\x0eG\xeb\xacj\x8c[\xa9\x05\x15\x86\xc2Ch\x18Q\xf04\x9e"i\x17>\x8e\xef\xac[\x12\xbfrSib\x89\
x0e\x1d\xd1&\xa2\xdc\x98wJ\x87\xcb\xb9\x12\xacP\x87\x15\x15\xe5\xad\x03-\xbaT\xa2\x82J\x87\xa8\x87FRx\x83\xf4/\
xc7\x98\x9cE\xbe;\x84v\xadnbnH[[\xb8\x97\x9c\xf5"\xe6Z\xbf\x1a\x83!\r5Or\x7f\x13\xea6\xa4%\x1e\x08\xe1\x83\xfc\
xc1@\xfd\xf5\xe0A<L\x1an\xd7\xe2c\x0f\x8f\x8f\x83\x9a\xdf\xc2^\xea\x08\xf0\x18S\xd6\xe5`\xadT\x89\xa9\x19m.\xa4
%N\xc6\xe5\xf7@ZH\xfb\xa1i>\xc7\\?3\x03x\xd9,tg_\xc5\xce3\xa5\x1e\xe2>\x0b\xa9x\x84\xbc0\xaf\x07\xbd)h\xac|\x9e
\x88\r\x95sk\xa0GD\xf9\xd7\x95a\x9dN\xe9p28$\xe4\xfd\xb4_\xb4\xd9\x8b\xbb2\xa2\xe5eS\xe2)\x08\xc7,\x1f\x1a\x0c\
x80:\xcd\x82\xb6\xd5\xa9:]p\xfa\x84p\xe5\x8c\xeb\x98\xf8\x01}\xc6+\x19\xa5v\x9bk\xa1\xc7a\x992;\xce0\t\xd3W\x1a
\xafQ\xbc\x88\x1a\x87\x9b\x85\x06\x89N\xb7\x94\xd7\xd1\xc9(\xfc\xbai\x00,[\x96&\xa6t}\xc6\x99\xa8K\x9a\xcb^\x14
p1\xa8\xf9\x0eR\x81j\x9dAZ\xfd\xc2\xe4\x99>\x8a\x10R\xacw\xde\x9fb(\xf5r\xa6T\x97\xdb!)ec#\xbc\x1dh\xd4\xf4\x99
E\x92\xef\xa9\x85\x12\x07\xdb\'M\x02\xe2\x98T\xbaT\x17j\x0c\x94\xa8c#\xae\xf4\xe5\x107\xf6\x95\x13)\x93!9\xea\x
c6\xe40{\xef\xadk\x03=&\xc1jMI\xb3\xc8\x94\x8f\xd7D\xbe\xee\xe7\x8e\xcc\x83X\xa3\x88\xa8/+\xc6<\xeb\x1e\x95n{\x
a9UM\x9a\xda%\x80\x87}\xfb\x1aZ\xe5\x9a@\x8c\x0bw\xe5\xdfBB\xd5\x9eC\x00j\x95<\x84\x10\x00\x96\xaf\xdb\xd1U\x1b
\x93*\xc7~s\xd6\x9aq\x83\x16Z\xe5L\xde\x14\x91\x96\xde\xc6\x06\x15\xa9\\\x950\xc0\x15(\xea5G\xea\x8a[M\xac~\x84
\xeb\xcb\x92\xe6\x9d\t\x1a\x99\nw>\x0fq=\xf9\xd3j\xe0\xf2\xae\xe7\xd6\xdf\x8c=\x9f\xbc\xae;N\xe5\x85\x1a\xa8\x1
6\xcb\tem\xc8v*\x9dv\x01S\x8f\xc8fG \x06\x19[\x8e(+!A\xb2\xc9=\x8c\x9d}\x16D\xf51\xd4\xe3c!\x1a\xe2\xd5T\xba^\x
f1\xdaT=\xd7\x99!\xb7n{e\xc6D\xd8\x0c\xce,\xa1\xc7R\xe0KHI\xfc\x8aB\xf8\xa9!G\xe9\n\tIQJ\x86\x10W\x92\xd9\xecB\
```

xa2\xadS@T\xee-\xb6\xf8\xb7\xd9\xe5\xd1.{^\x15~m5J\x9a\xfcZ\xa2H\ty\xa7\x03N\x82\x9c\xe5*A\xc2\t\xc9\xca\x16\x0
8R\x808\xd2\x132\xf1apw\x8c\xda\x98\x91\xde\x1f\xf0\xac\xda\xfd\xde\xb5\x9d\xbb\xb6i\xdam\xa3p1\x849L\x8d-O\xc3
u\xd2:B\x8f\x1c\xb6z\xf25>o\x07\x13\x8bM\x18\xfc^[\xa9\xa3\xb9\x827\x16\xc0\xbcv\x86\xee\x9d`\xdf\xb4\x95\xc2\x
a9\xd3\xdc-\xbc\xd9W$+\x07\x1c\x90\xaf\xe6N=\xc6\xb9\x8c\xad\x89\xb8\xb0\xdc\xe9+\x86NB\x0e.[KW\x8c\x1c\xff\x00
}i\x06\xa6\x86\x12|zQ\x92\xd0!\xae\x88\xe8\xe3Ha\xf7\x9a\r\xceT\x8aR\xa9\xb51\x86\xf2\x0b\x83R\xe4Qq\x8a#\xf6\x
c5\x9c\xb4P\x93%\x94\x0c\xb6\x9c\xf9\xf1\xd6\x84]\xcd\xea\x00nm\xe1V\x91=h\x93 \xf0Y\xe3\xc4\x1d\x0b\x1d\xee\x1
e3m Rh\x94W)\x1c\xdfZ\x0b\xab9 \x9e\xf5=\x0b\xb9M\xda\x91\x05e\xd8\xf0$\xd7\xd6\xfcD\x85)^\xdfa\xa2T%\xaeNh\x1a
\x96\xb2\xf6\x8ag\xc8\x054\xc8\nP\xc9\x1f}X\x14(\xa8\xbe\xa05\xc9F\x91C\x96\x14\xd2\x08-\x9f\xac\x13\xa1%\x87S\
xdd\xce-]E\x84%\n\nO\xd3\xe0\x9f:!\x91\xaez\x80\x9c]\xbd#8\xe0\x0f\xb6@\x07\xc9\x1a?Ve\x89\x7f@\xa9Se\xe0\xb201
\x801\xa3\x19\x16\xfa\x81F\xe7+\xbe\x8d&|%|\xaa\x8f\x8e\xba\xd1eR\xcbbz\xa2\xf3\xf0\x8a\x9d2\xa0U\'>|\x9dJ\xaa\
xcag\x80\xb3\r-\xea\x9a\xe3pC\x8b9\x1d\x85j\xa4n3J\xee\x18\xd2n\x14\xba\x03n\x00A\x18\xf3\xa7\x87\xb8\x05D\x972
\x81\x16\xaa\xc9ZR\x02\xbc\xf7\xa2\xe2\xad\x03r\x92\xa9m\x18H\x05lt\x0fJ\x1a\x13\x8b\x8c\xd9N\xb9\xcc\xc3qIZ\xc
2q\xec\xad\rWP\xc6\x84\xfb\x83\xba\xaa\x93n\xee%\xb3s\xcb,\xa9\xd7d?\x02\xa0\xf3jBS!\xb7HSj\xe0\xa2W\xf9\xc9l$+
\x008\xac\xe4\x11\x8f\xa4\x04\t\xf3\xd4\xc5bWz\xe3\xca\xb5\xee\xb9\x82\xdf\xa5\xb7U\xb6j\x0e\xaf\xd5@%)L`\ne1\x
d1\xef\x8a\xb8\x12\x8e\x80\x0e\xa8\x91\x91\xd2[\x92\xb6\xba\x8fJe\xfc\xc5\x1e\xda\xef\x85\xd9\xb6\xfb\xce\xcc\x
c7\xea3gZue\xa2\xa1)\xb6\xe5\xa9\xc4A}IS%\xcc\x9e\xc0QG\x15g\xdf\x919 \x9d,\x1amu4\xaf\xb7q\xd5\xbfp\x9an\xe1\x
84\x9be\xca\x93\x10nP\xca\x9a\xa9\xc1\x96\xa2Q \xaf\x86\x16\x80Bx\xf8\xefDT\x91\xa8\n\xc1w\x07\xf7\xb7\xe1\x0e\
xc2\xdc\xeb\x15\x8d\xba\xbd\x9eq\x17+l\xad\xd8\x15\xd5\xa0\x07\x92\xe2\x86B]?\xce\x9e\xb1\xfbh_\x08\xca\xbcXu\r
2\xb63\xc8O\x9bWf\xdb]{\x7fxJ\xb3\xee\x88\x05\xa9\x90d)\xa7\x12|(\x03\xd2\x87\xdc\x11\xd8\xfbk\x92\xc0\xe1%L\xe
b!\x19\x16\xc4$\xb7iJL,>\x8c\x1cx\x07K\xbeGq\x95R\x05^\x94\xf4I^\xab \x80T1\xfb\xe9\x19S\xe6\x185\n-[\xa6dH\x0b
\x89\xeb\x10\x0fA#KU\xd4\xd2A\x94\x97]\x1e}A\xc4\xcaJI\x1c\xb2u>Eo\x88X\xcf\x19D\xd56\xb8f\xf2\x8f\xcb\x888\x03
>\xfa\xe7\x93\x90d\x8f\xd1]KZM-\xdad\xf4?,\xa4)^\xe4\xea\xccn\x0f}\xc9\xc8#\xb8\xc4\x8fQ\x84\xf5\x18\xbc\xcc\x8
c\xab\x18\xc0:w\xccg\xb6\xa2\x86\xf7\x8d\xf35\x07J\xdb\xe3\xcdG\x03\xcfZx\xf7-\x191\xbb\x82\xea\xa6#\xd4\xe2\xa
4\x0f \x0c\x8d\x08Z0\xee\xbb\x9e\xe7\xda1\xe52\x1ci\x00\x9f|h\x99.\t\xeeE\xa6S\xa6S\x9f\xe2\x84\x90>\xc4\xe96Gp
\xbb\x10\x86\x1dq\xbe>\x8c\x90<c\xcf\xbe\x8ce"a\x06W\xd6\xe3E\x9a\t(I$\xf5\xc7L\x19\x10\x8b0@"V\x9aT\xa6\x13\xc
9\x08\xc8\x03\xc0:\x03\x9f\x10\x86\t\xa9.\x8b1e\xff\x00\xac\x91\xc7\xee|i\xe9\x91[ba\x10\xb2\x91p\xae:p\xf9\x07
=\r<\x18\xb2(N\xb5K\x963\x8c\x94,\xa4\xf5\xe0\xe9\xa1\xf5\x06\tV\xe2\xc4\x9a\x8fQ\x18\xfa\x8f\xb1\xd2\xdf\xf1\t
{\x9fMoM\xcd\xdbo\x88Iv\xa5~\xfe\xbf$"Be8\xf5\x06\xa7MaJa\xe5\xa5?SN!I)nBT\x91\x85\x02\xa2\n\xd3\xd2J\x82u\xdc\
x1e\xed\x89\xc4!\x81:\x8c\x1b\xfa\xcd\xb9\xe6m\xedJ\xf9[\xe9\x14\xf8s\x95&L9m%\x84K\x8f\xcc\xfa\x92\x06\t(J\xa3
\xad\\\x81$\xfd=\xa7\xd8\x9b\xdd\\\xc4\xab\x81\xd4\x9f\x87\xed\xb6\x83P\x9d>\xa4\xb9q\xd8\xa8Gr\x1dR$\xc4\x85ro
\x96\x0b\x89\x07%KS\xcaQR:9\xc1\x18%G@\xaa\xa3bk\xb3\\\xfe\xee\x04\xdb\x9fn6Y\xcau*\xb6*3\xe9\x05S\xa9RT\x82\xa
5\xc9\x84\x95}I\t rR|\x9e\xb3\x81\x93\xac7\x8d5<\xb4\xec9N\x9b}\xf1\xbb\xb4\xb7\x8b\x14\x8a5V\x87T\xa8\x11\t\x1
f4\xf1G\xfc5\x80\x02\x9e\x05\x7fR\x92\n\x8e@\xf6\xc64\xa5\xf2\x94\xb5\x18\xe6\xf1\xd8\x0eRe\xff\x00\xb3\x1f\x0b
\xd7\xd4\xc6\xee\x9d\xd0`\xcd\x9f1\x942f\xa5e\x946\x9eE)_C\xafo\'=\xe8\x9f\x1e\x0c\xdba\x00d\xca\x83Ff\xaf\x89_
\x84\xab\x9fd*\xab\xacY\xb1_\xae\xdb2~\xa85H(.\xa5\xbf\xfaVS\xef\xae~o\x17\xd26\x9b\x12\xec>J\xb8\xa6\xee!\xeas
\x14\xf09o*+\xe0R|\x82=\xbf\xf8\xd4\xa5K\r\xcaC\n\xb3,)TW\xd2\xc2_\x00\x13\x9c\xebB\xd4 E\xc9\x92\xaaq\xdaB\xd2
\xb6\x86G\x8f\xb6\xb3\x80mMf5 \xd3&@K\xa9[\xd2\x1ag.\x00\\t\xe1)\xc9\xc6I\xf6\xff\x00\xef\x8f~~\x7f\x16\xcdMG7\
xa9\xbf\xbe\x1d\xbf\xc2{ow[d\x11]\xbe\xee\xc4\xcb\xae\xcb\xb9\xa9\xaa\xf9\x9bfw?\x96\xa4\xbe\x94!xWm\xb88\xbc%%
\xc4\xe5*CC\xb3\xc8\x81f\x0f\xa6\xa2\xe3\xf7\xf7%\xcd\xe6\xf2zQ\xaa\xfe\xf2\x07\xc6_\xf8_\xdb\x1byhP\xd5\xf0\xc
c\xc5F\xa3W\xa7\xc4}\x8b\x92\x14\xa9\\\x84\xc4\xc5\x88\x97\x9e\x9a\xd8WiR\xd6\xb4\x0fL\x1e9X\x03\xb1\xdb\xb2x\x
aa($\xcc\x1eU\x13\xce|\xf4\xf8\x91\xb6\xea[\'\xb9\xd5\x8d\xb9\xba\x9c\x8a\xba\x95\x1ab\xa2ND)!\xd6\xd0\xea\x7f2
R\xb0\x00V\x0fD\x8e\xb2=\xf5\xc9\xcd\x94az3\xa3\x89FE\xb1\x00\xa2\xd4\xe3\xcd!\xe6\xc8\xef\xc0:<YU\x85\x88/\x8e
\x8c\xec\xba\xa9d)-\xe4\x121\xfb\xe9\xe5\xaa.\x8c\x876\xa8=,qYs\x1e\x123\xa9\xb2\x1eGQ\x8b@J\x8f\x9c\xab\xf2/?\
x01\xd4\xb6\x15\xda\x8e\x87\x8f\xdex\x9b\x84T)\x11\xdd\x01\xc4\xa7 \x0e\xf9\x8d+\xc8\xc8\x13\x16\x8c \xa2\xe7[\
x8e\xbdH\x87\x14\xa5 \x05\x0e\xfa:\xf8\xe7\xf2\xfc\x86\xf2j2\x94\x08\x1d\x16\xe1*\xa8\x9fK%\'\xecu\xf6\x1e)\xca
1\x02\xd1d\x8a\x97\xd1g\xcbq\\\x83\x84\x00:\x19\xd7E2\x12 \x11s\xc5F|\x84\xb6\x14\xea\x89\xc1\xec\x13\xaa\x14\x
98\x04Q\x90\x9e\xab\xa9\t\xfaNS\xfb\xe8\xc1\xdcS\x83>\xa4\x7f\x875:\xd9\xba\xec\x19\x96cp"1\x1e\xdc\xaf\xfc\xec
\xcbu\xd4$\xb9\x12RJ\x8a\x1dB\xb8\x10\xa0\xa4\xfd9*\xc7\xf0\xfb\x1d\xeb\xb9\x84\xa8^3\x91\x9a\xc3\\x\xdc\x97,\x
ab\xd2\xc6\xb9\xac\xe6\xe1\xc5\x90\xfd\x1ab\x9b~\x9eTP\x99\x90$#\xf8\x0b\xc0\xc7\xa6JH\x04\x8c\x8c\xa4\xe7\x19\
x18s\x1b\x15\x15d8=E>\xccQ^\xda;\xd26\xcb\xde\x17C\xaf\xd1\xe3DZ\xa8Q\xcd)\x94\xa9I*J[\x92\n\x96\xa3\x90\xc2\x1
5\x18\x80\xa4\xe1E\x19I\x04\xe1(x\x9a1\xceA\xea\x16\xd6\x1b\xa0\xd9\x92QK\x99T\x8e\x11\x11\xb9\x12P\xddI\xf0\xf
1\xf4\x93\xc5*e<\xc6Rx\x8c\xe18\x1eGC\xc1vn/\xb1\xf9\x99R\xec\xa2\xca\xa4\xfcU?b\xcc\xb5\xc3\x14Z\x8a\x96\xbb}\
xe6\x1d\xf4\x92\xf4W\xcfHiC\xf9Fq\xf6\x03:\x95\xb1\xaf\xad_\x07\xa9H\xc8}\x10D~Y0k\xd5)q\xb6\xd1\xab\xf1\xf8\x9
6\xdb\xa4\xa2\'\xe1\x94v\x96\x80\xebd\xf1IJ\xf2\x08# \x9c\xf6{\xf7\xd5\x01J\x8a\xf8\x8ag\xae\xa4\xfd\xb7\xdeG-\
xcb\xfe&\xde3V\x8e\xe5\x0e\xb0\xd3\xad&<\xf2\x946\xfb\x888%>\xc8=\xfe]o\xa8\x01\xe3\x04co\xdc;\x8a\xef\x89?\x84
7\xaf\xcb\x9aU\xc9\xb6\xb4\x06\xa3\xcem\x05n\xc7`}\x12\x02F0@\xe8+\xf5\xd2sx\xde\xa6\xd7F;\x0f\x91\xc0\xd3L\xc4
\xd5}4\xb5\xc9\xa2T\xe2-\x99Q\x9cSO0\xb4\xf1R\x16\x0f`\x8dr\t\xa6\xe2gUx\x95\xb1(\xea2\x9d}~\xa3I +\xb0\x08\xc9
\xd6\xa9 \xdc\xc6\xaa\xdc9\xf82\x89\xb9R\xbe(md\xedu\xbe\x9a\x94\xe4\xce\xe2\xf4w\xa2\xad\xe6Lu\xfd\x0e\x87\x02
Zp\xa1\x05* \xaf\x89\x03=\xf5\x9d9y6ABO\x93\x8f\xa6I\x9feh{\x97wm\x8c4=\xb8v\xaa\xe3\xd4\x08C\x14\xc9\xb1\xa2\x
b6\x811%D\xa5\x97\x83?B]B\x8a\xfe\xa6\xf0\x87\x07\xd4\x90\x92Ki\xf9?\xd7\xdf\xabq~\x89\xfa\x06O\xa9:s+@/@\x93\x
f7?\x15\xf2~\x04\xab\xe8\x7fK\xff\x007\xf2\xc7\x8e\xadW\xb2O\xc0\x1d\x99#n\xb7>\xda\xdc\xea\xe4\xfaJd3m\xd5&Fu\
x98\xf5\xb1\xc1F\x1a\x14\xb6\xdcp\xb6\x15\xff\x00\x98T\x86\x94\n\xb2\x90YI\xc63\x9f\xcc\xff\x00\xc3\x8f\xf1\x9f
\xff\x00V\xfdW\xfc\xbf\xea8\x17\x0b\xb5\x94!\x89\x07\xff\x00\x89\x04\x0fp\xfe\xff\x00a>\x93\xeb\xff\x00\xa4?\xc
a\xbco_\xc7r\xe0~\xe1_\xdc~?\xf2\xe6#\xff\x00\x13\xdf\x82_\x87\xe9{^\x98\x9bOa\xca\xa6\x8bl.D[\x89\xe9\x14\xd81
\'\xad\xc5z\x92^\x99*R\x84\xda\xa4\xb5\x84\x8e\x1c8\xb4\x0b\x84$\x12r\x7fp\xf2\xfc<yq\xda\x8e\xbeg\xc9x\x9eS\xa
bS\x1e\xff\x00\xa9\x9f:\xa9[\x1e\xa5(|\xb2\xd4\x8c\x1caC\xce\xb8\xde\x83\xe3\xeau\x19\xc3\x1a?\xf6\x93j\x1b\x05
Z\x88\xc0~<\x80\xae\xfc(i\x8ar\x1d\x18\xb2\x00j\x83\x8f\xdbR\xa9\x13\x9bnS(\xc2\x08\nV\x8fbx~%\xb5Z\x93o;OYm\xe
4\x95(~S\xef\xa0cb\x17\x10~`;\xb1\x17\x1b\x9a\xe3\xbb\x81\x93\x80\x93\xed\xaf\x9f\xf3=b\xf5z\x8e\x18\xc5Y\x94UZ
|\xea\x89\xe0T\xa2\x9f\xbfZ?\x0b\xc3NA\x8fql\x84\x89qi[\x90\xe1\xc6S\x8f6\n\xf3\xdeG\x8d}\n\x85\n*\'\xa9p)l\xbe
\xe6"\xa0\x95x!:5 \x1b3\'\x17lz\xcc\xe5\x12\xd4~\x8f\x80F\x983\xa0\x80A\xb9\x16\xa1\xb5\xd7\x1bl\x17\x83!)\x1d\
x9f\xa4\xeb\xdf\xc4\xa4\xce\xe6\x9b\xda\xab\x96\xe8\xdb\x0b\xf2%^\xd6J\xe1L\xabS\x94\xcc\x96b\xd4x\x96\x0bi\xfe
!>\xcaP\xe8\xf19\x19\xc7\xb1\xce\xbb\xf8\x98\x83c\xb9\xcbu\x0c\xbb\xea67\x1b\x7f\xae\x0bz\xb5I\xbc\xe2\xdcl\xd3
\x1e\xb8"\x7f\x97j\xc8\x82\x80\xc2\x94\x14\xdf\xd2\xf3\xbfO\xf0\xd4\x87\x16\x16=\xba$u\x9d5\xb20a\x14\xb8\xd7\x
8e\xfe%\xf6\xe3n=&\xf5\xba\x11m;_\x9fI\x8f\x05\x99\xcf\xb4\x87\x9dK\x8a\xa5:\xa7\x1aKme\xb5\x1c\x80\xe2T\xb0\xb
5t\x12\x1dO\x8c\x1dc\x9bi\xaa-n=\xf6\xde\x87m\xd7\xec*\x85.\xee\x83,\xdcb\x0b\x12T\xfdEeMKW\xa6\x96\xdcz9=\xa1\
xcec\x8a\xfc\xf4\x10}\xf3\xaa\x16\x88\xaa\x88m=\xcc\xcf\xbbW\x9d\x06\xf4\xb0\x9f\xa0\xd4f\xbdI\xb9\xed\xb6I\xb7
\xd6\xa7@uN\xb4\x00SK8\xc0%\x18\xe3\xd7e\x03\xaf:K\xd3\x8a\x1d\xc7\xa7\xb7\x7f\x10\x87\xe1\x9bpm:\xba\\\xb2w\x0
e\xbd.\x0b\x88\x84\xdf\xcb\xcaq\xec\xaf\xd6 \x82\x92G\xe5W?\x04\x0cw\xa0\xc7\x92\xc54\xdc\x89\xbdu"o\xef\xc2\xe
5\xd9N\xa7J\xb9\xec\x9a\xc2\xa6S\xa1\xb6\xb9m\xa0\xac\xa9h^;(=\x11\xd8\xf0>\xda\xcc\x98\\\x8b\x13S(\x06\x13|-\x
fcE\xd2\xa0X\xf4\x9a%\xd7r(U\x94\xff\x00\xa4\xfa\xdcV\x03\x0c\xa9\x04%K\xef9\xe5\x80s\xed\xa3\xc5\x90q\x17\x03"
\x1eV:\x94?\x18\x7f\x0e6\xa6\xe1\xda27\xbbo)\xcd\xb1_\x83$1\\\x8b\x17\xeanns\xfcD\x81\xfc\xddgH\xf2\xfcp\xeb\xc
dF\xfec<l\xed\x8c\xf1\'S"9M\x98\xcc\xa0\xc3\xad)#\x07\x90P\xed\'\xdc\x1f\xd7\\\xb5%Zt\xc8\x0c%\xa5\x99|_[1s\xc5
\xbf6\xe6\xe8\x9dJ\xaa\xc4u*\x8c\xfc\'T\x92\xae\xff\x00"\x92:ZI\xe3\xf4\x90A\xf1\x8f\x1a\xa00\x03]\xc4\xb2|\x1e
\xa6\xd8\xd9\x1f\xf1"\xdd\xcf\x89[\xce\xc3\xf8O\xde\xcb\x82\xd8i\xea\xfd\xc8\xd3\x13nDBR\\\x8e\nV\x10\x82\xa48\
x94\x17\x94\xb2\x86\xfa\x1cA898\xc7\x0b\xf5/\xd1>\x9d\xfa\xab\xc0?N\xf3R\xc1\xdd\xff\x00\xed#\xa2?1\xff\x00N\xf
2|\x8f\xa4\xf9\x07\xca\xf1\xcdW\xf7\xff\x00\xf2h\x9f\x87\xf6\xa4\x9d\xc0\x80\xcd\xdbwZ\xf3\xda\x87uT-\xf7\xdc\x
87P\xe1"EJ\n\xc2W\x1d-:\x80\x958\x12\x03\x81\xbeAKC\x80\xa4\xfd\'_\x86\xfe\x92\xff\x00\x03|\xbf\xa0~\xa7\xc3\xf
5\x1c\xfe@\xe1\x8c\x86P\x07u}\xfd\xbe5\xf8\x9fg\xf5_\xd6X\xbc\xef\xa7\xbe\x0cIE\x86\xff\x00\x91\x97\x1f\xe2oR\x
d9Q\xb05\x9d\xc5\xafX\x14\xeb\x9aM\x15\xa2\xd2\xe9\xd5X.6\xb8.\x85\x00\x1bR\xd2\xe8z99\xf2\x94\x94\x9eC\xac\x1d

\x7fKd\xa6^Gg\xf3\xf9\xb9\xf9\xd6\x0b\xe7\xc6\xe7\xc6\x9aV\xe9N\x84\xff\x00\xcc\xa9\x8eag\x96\x12z\xfe\xfe\xfa\
xe6\x92\xadv\x04\xe8\xd9S\xa3.\xa6\xef\xf4\x0fM,\xc9\x84\xb4\xa8\x0f\x03\xbck\x00\xc4\x8b\xd4b\xe4bw\x04n{\xaa\
x8f_\xe4\xf4c\x85+\xfea\x8e\xf4\x8c\x9c\x0e\xd6\x10}Jx\xec\xbb<pH\x03\xff\x00\xdbS|\xc7\x0b:2\xa2\xe6\x8f\xf2\x
a1<\x96{8\xc0\xfb\xea,\xd8\x96\xec\xcaR\x94T\xe5M\xa2\xd4&D\xf5\x9b`q\x1e\xe7\xadyq\xbbu0\x95&36\xcbia]\x94\xf5
\x05\x12\x15\x8c)I_\x83\xaaC\x11\xdc\xe5yY]\x1fCS\xa5wk\x7f\xd3\x97\x8c\x87^. \x1c\x8eG[\x94\xb6QK\x17\x87;1\xd
8\x93\xed\xeb\x92\x92\xfb\x00)\x86\xc1#\x1d\r|\xef\x98\xden\x02x\xc7\x9b=O\xed\xc5r\xb1\x1e\x9c\xe2}$\x10RA\x03
\xdb\xf5\xd4\x9e?\x99\xe5d\xcbF\x00\x0c\x18\\\xba\xa5\xc1\x91\n\xa3\r\xd7\xa8\xeb\x9fS\xa8\x05\x07\xaa/Ou>\xb8\
xc0JO\x12\x94\xe7>F\x15\x95\x0cg\x19\x07_\xaa\x05a\xf1"%XP\xf8\x87\xceT\xb6f\xdf1\x99v\xad\\\xa5\xdc\x94\xe1\x1
d\xd9l\xca}\xb8\xebp\xb4\xe1)PB\x8a\x12\xfa\xd3\x92@\xceF@\xe4\x0f\x9a-\x06\xbebO\xa8\x04\xef\xbdw\xbd\xacX\xa6
\xa2\x0c\xc0n\x06e\xado\xd4\x1a\x90\x97[_\xae[r3\x05%G\x82\x82[*\x19\xe4\x95\x820\xb2R\x90E\x9c\x1e\xa7\x90\x1b
\xdcel\x17\xc5\x1d\xcf2\xb1T\xb2n+RC\xf7 e\xc5|\xfc\xd6\x7f\x86":\xb4\xa1\xa7X\x01I_\x02y\x90\x92FF0\xb3\xde\x1
9\x8b0f\xe3\xf33."G!\xf1*~ \xb6\xdau\xd1x\\\x96\xbd\x1a\xa4\xd4i\x10\xe3\xc6\x9a\xc4\xb7\x96\xe8J\xe4\x8c\xa1\x
d4\xb6T\x8c+\x9a\x02U\x82F\n\xfaQ\xf1\xa6:\x96j\x11h\xd4\xa0\x18\th\\&\x99Uv\xe6\xa7R\x90\x8fA\tC\xec\xc3q!\x05
\x07\xeaK\xa0\xe0\x92\xe6F}\x8f]\xe99.\xc1?\x11\xc2\xc9\x8c\xb8;\xddT\xadR\xe6\xc6r\xe7\xaa\x84\xbdNR=E\xa0-\n%
9B\xceI?\x9b!Y\xeb\xb1\xa2\x0e\xdb\x9e\xe2\x03\xcc\xc4\xc5\xdb\xfe\x99\xdf\x10\xef\x10\xdc\x97\x1a\xa9\x17\x1b\
xa9\xb6\xb4c\x8a\x81O&\xb0\xae\x81\xc9$\x0f\xd3P\x8c\x9e\x9b\xf22\xa6\xc6\x1d8\x89\xb5>\x1e\x8d\xe1qUi\x11l\t\x
8c\xbdG\x9f\x99M\xca\x99\xf5\xf8\x1d4\xe2=\xcfx\xce\xba\x00\xb6\x80\xeas\xd9M\x9b\xee*~96.\xa1\xb5[\xa2k\xf2\xe
2\xc6K\x15\xd6\x8c\x84.+<\x1a\x0e\xe4\xf3H\x1e\xdeA\xfe\xba\xe6y\xd8\xdb\x1e`GFt<<\xbc\xf1\xf1n\xc4\xcc\xb7u\xc
3\xf8q\rF\x8f\xc9\xf7\x16\x11\x19D\x91\xc1D\x1c\xaf\xae\xf2\x94\xf2\xc7c\xb2\x0e\xa5,H\xd4\xa8\x8eS\xa6\xc0]\xe
3i7r\x85\xba\xaf\xd0QW\xfc\np\x94\xd55\xe9E\xb0\xeb\x88B\xbd.K\t$\x04\xac\xa5D\x81\x93\xc7\x00\x8c\xe4{\x1b\x00
\xd7\x04\xa1*@\x8c\xbb\xcf\xe3\x1e\xeb\xbf\xed-\xc4\xb7791\xd6\xe5\xe7R\x87p6\x9ad\x10\x86\xda\xaf\xb3\xe9\xb6\
xe4\xd4\xfdYl\xba\xd7\xaa\\\t8R\xd6:\x00\x00\x18s\x06fc\xf3\x17\xe8\x85\x0b]\x0f\xf8\xf8\x90m\xbb\xcf\xe2\x83|\
xad\xd6m:\xce\xe9\xd5\x8d\x14SE9\xc6]\x90\xa2eC\x0b\xe6\x96^>^B\x15\xf99\xe7\x86HN\x07Zn.yuz\x80B\xa9\xb5\x14a=
/\xe0M\x9a\x84!"%i\xd5<\xb5\x9c\xf2t\x1c\xfd\xcf\x8d1\xfcLl;\x80s8\x9d.\xcf\xf0\xc8\xdd\x194QQ\xb6^2\x15\xc7!+G
G\xfb\x1dM\x93\xc1c\xfbf\x8f,\x8f\xdc"\x8a\xad\xf0\x9b\xbb\xd6D\xa5\xc2\xaf\xd2\t\x01]\x04\x05d\xfe\xbd\xeaV\xf
1\xb3\'bT\x99\xf1:\xcfh\xd9[\xae\x14c!\xda[\x88O\xea\x8c\x9f\xf6\xd0\x9c/]F\xfa\xe2\xf4e\xf5\x9b\xf0\xeb\n\xf5R
\rA\xb4\xa5}\xe3?\xfc\xeb\x9d\xe4\xab\x05&?\x13\x860\xbe\xb5\xf0\xedG\xb4hD"?\xa8J{\x18\xeb:\xe4b\xf3\xb3\x06\x
d8\x9a\xf9Qu\x00\xed\xfal\xcbZ\xab\x84\xadl\xa0\xaf(R\x0f\xd2\x7fC\xa2O0\xber\x1aA\x93\xc8G4\xd2\xee\xebjM\xc1\
x03\x13\x16\xda\x8a\xbc\xab=c]\x8fY\x14\n\x80\x8a\x08\xf6\xc1#i@\xa4\xc5[\xad\xc9<\xb0O\xd2z\xd4\x1eve\xf4\xa58
\xd4\xde\xc4\x0b\xbe\xee\x18\x94\xeaz\xf9<H#\xfa\xea\x1f\xa7"\xb3\x92F\xe5\x8c\x14-K\xbag\xc4t;\x1a\xe7hV\xa8H\
xacQ(\xf9o\x0eH\xf4\x89R\x00F\x12|\xfd8\xe8\xf7\xd7\xdf_||\x82\xa7S\x8a1{lve\xa3\x7f\x13\xd6\x96\xe7\xdf\x92\xa
eW\xad\x03p|\xa4\x10\x12\x85pi\x96=.9\xe0\x95+\x0eq@Rr~\xb5\x03\xd0\x18\xc6\xb7\xf8\x84g\xb8>\x8eD^\xe1\x9d\x0b
t6\xb2\xec\xb6T\x9d\xb0\xb1\x1a\x80\xc3\x0c\x06\xaaQ&4\x14V\xa5\xc9.6\xd1Q=\xa5`4\x82q\xd1H=\x14\x8d<e\xc6\xe3B
\xa2\xbd7V\xb67\n\xed\x9b\x12\xd1\xdc\xdb9\xab\xf2\xdd\xa8?\x06\xe9\xb5f\x1aS\xad\xfc\xc3hzCJ.-*\xc8\xfc\xc9\n\
xeb\x91\x03\x05g\x1a5\n}\xc3\xb1<I]|\x18M\xb8\xbb\xb7&\xbf\xb3\xf1w.\xd9\x83&%A\x0e\xa2\x97Z\x8e\xb9.4\xa6\x14\
x12\x82\xc2\xca[<V\x85\x80q\x8f\x05$w\xd6\x1e23 z\x8a\x18\xed\xcf\xda\x04];\x8c\x8b\x963NS\x9d\x8a\x85\xc5}FKP\
x1d\xe3\xeb\xc8R\x89R\xd3\x8ce*\x18\x1fl\xf7\xef\xd0d\xc8\x1cj\x1a)S<Zu\xfa\xccJ\xd3\xf4XP\x1fJ\'\xa8\x08A\n\x0
5A\x1d\x85\xb4r{\x05_\x7f\xb0\xd2\x03\x91\xa8\xd2\x00\x17\x0e*6.\xdb^v\xddn\x9dY\xb3WPv\xa0\xf3i\x872#\xbc\xbd\
x07\x1aO\xd4\xb5c\xae\x7fP\xcf\xdf\x1aaT \xd8\x8bVpl\x18\xd3\xf8[\x81P\xda\xc5\xc6\xb0\xaa\xf57\x1a\xfc6(\x97N\
xa86r$4\xbc\xe1\n\x1f\xf3\x0e\xb3\xa7\xa0\xe2\xbcd\xee\xfc\x8d\x88K\xf1\xe5)\xad\xcf\xd8\xe4*\x94\xc7\xce\xd4(\
xce&I}\x07\xc0#\x8a\xc7\xdf\xc0\xff\x00m\'\xcbNX5\xd8\x8c\xf1\x9c&M\xcf\x9bS\x9a\x9d[\xb8Z\x83\x1d\xc0\xa54\xca
\x96\xb4\xf2\xed%D$\x1c~\xc1C\\\x12h\xf1\x9d\xb0?\xd3,>f\x83\xd9\xad\x84\xb1\x1f\xa52\xf5\xcfSk\xd6_g\xd5\xeb\x
ce\xa8\xc6\xa9\xf3\x17n;\x967\xb6\xc6m\xd5\x16\xa1\xf3p[a\xdc\xa7\xa2FI\xfe\x9asb\xc6\x16\xd6)I-FE\xa5J\xaaR[E*
\x8b\x11L\xa0\xf5\xc8\x8e\'\xf6\x1a\x00Yz\x8c\xf6\xc3+Nu\xfdD}\xa8\xa9a\xc0\x97\x15\x94\xbaVH\xefF=`"\xd9q\x99\
xa2,?\x89z\x9d\xa7\x01\x8a\x1dZ\x98\xe3\x84\x802\x07@j\x9fP\xa8\xdc\x99\xf1\x83\xd4\xb9\xdc\xaa\xa5\xb3}R\x1b\x
95\x1e\x90\x85\xbe\xe0\nJB<\xfd\xfb\xd5"\xdc\\\x9c\xd2\x1b\x81\xb4\xba\r\x8f \x1aEv\x94\x86])!$\xe3B\xd8\xc5\xd
1\x9b\xcc\x91bQK\xd9\xaaKUw\x05\x11e($\x14\x94\x8f\xcb\xa9\xf3\xf8X\xf3-T~/ \xa1\xdc\x06\xdd\xc9\xf1mv\\\xa0V\x
1dG2\x93\xc1C\xdf\xff\x00\x9d|\xfb\xf8)\x85\xe9\xd6\\Tf]w3\xc6\xea\xd7\x97o\xd3\x9c\x9e\xdc\x1em\x04\xe5*\x039\
xd77?\xd3\x8f;Y1\xc0\xc0\xc1\xab?r\xa3\xdc\xd6\xf0n \xcb\xc7#\x87\x1e\xc9\xfb\rca8\xf1~e\xfe&%\x02\xa4\xf9\x96\
xf5\xd2\xe5\rFbJ\x1c\xc1)l\'\xb0?]B\x9fM\xf2\xbc\x86\xb64%9rc_\xdb\x14\x17%\x99w\xdc\x13\xd7\rQ\xde-\x85~d\xa7\
xce\xbb>\x1f\xd3\xce#\xd4\x81\xf3\xd9\xa1\x17\xd4\x8d\xb6\xa8M\x8e\x995\xa7V\xf2T\xa3\xc1\xb59\xc5\x19\xef\xb2\
xa1\x90A\xf3\x9f\xd3\xf5\xd7i1\x92\xa0\x98\xb2\xcaF\xa3R\xc0\xb2\xef[B\x98\xc5\xc7Ef\x1c\x96\x15\xc5ap\x9cJ\x9d
\x86\x90\xa0PT1\xd7\xf78#\x19\xefL\xe0\xcb\xb1\xf1\x04\xba\xb0\xe2a\xad\x03s\xf6\xd0\xd5*3\xef{l\xbb:\xe6a\x86\
x1e\x99\x11\xc5Ff*\xdb\x19T\x82\x11\xd1*X\x04d\x8c\x16\xceU\xde\x9a\x99\x10\xb5\x9e\xe2[\x16@\xb4#\xd7\xe1z\x9f
\x07v,\xe9\xf4\xaa\x15L\xd6j\x11\x80\x02\\.HT\x97\n\xca\x92\x16\x08\xc0Y\xcaS\xc0\xf1)Q\x1d\xa8x\xb3\x0b\x07\xd
0\x93\xe4\x1c\r\x99EsU)\x16\xec:\xedr\xc8\xba\xa14\xf2k\x11!\xd6b0\xe1\xf4p\xb4\x14\xc7ymd\x84(\x04\x94\xf4\x12
\t\xec\r\x11\xa4$\x8e\xefs\x07\xb8T]\xd9\xd4\xdb\x86\x0c\xf1L,%n,%)\xf5\x00\xe6\x85{\x04c\xb3\xf6\xc6=\xf4\xb1\
xd5\x18\xdd^\xa5\xef\xe0w\x1d \xc0\xb8-\xe7\x1dzW\xd0\xff\x00\xa3!\\\xb2\xa0H\xe0A8\t$g\xfa\x8d\x0b\x03z\x87\xa
2\xb54N\xd5V\x19\xbdl\xdf\xc5\xec\x86\xe3R\x1c{\xea\xaaB\x96\x8c4\xcb\x98\xc1P\x03\x1d\x9cc\xaf\xdfN\xc6\xdc\x8
5\x8e\xfed\xce\xa5N\xc4\x17\xb4o\xcb\xd6\xd6\xbe\xda\xa2]\xd5\xa8Ri,)_#;\xd6\'\x01g\xa4\xfd\x8ar:\x1a$r\x998\x9
3\xa9\x8e\xa0\x8b\x9a\x1e\x8dv\xda\xf3lY\xd4\xea\xa4\x98\xaf;.\x9c\xeb\x11\xa6\x14}*P\x1c\x8aN:9\'\xad5\xa8\x82
LU\x1eQA\xf0\xb7\xb4\x1b\'\xb9w\xb6\xe8^\xd5\xea3m\xfe\x18\xe52\x9fLCH\x05!\xc4\xc7R\xdf__\xfeE\x7f\xbe\xa3\xc3
\x8b\x16L\xaeH\xea[\xe4e|8\x10\x7fS"\xefW\xc3\xb5v\x90\xb4^V\xcc\xf4\xbfH\x07\'\xd2=\xb2?\xa7D\x7f\xdbK\xf2<B\x
9e\xe5\xeao\x8f\xe4\x064\xd09\xfaeBT6\xc5\x19\xc7$\xcbQ\xc6\x163\x9f\xbe\x96\x83\x92T{Z\x9b\x87\x9bQ\xb2\xf5J\x
b5M\x0f\xdd\xd3\x12\xc2\xd0\x9ea\x07\xc7\xe8;\xd1\xa62\xa7pK\xf2\x97i\xaa\xce\xab\xdfOY\x94x\xc8"0\t.p\x1cs\xe3
N\xfd\xc7\x8cXf\xb9GwS\xf7n\xd2\xba\x13*\xabAK\x90\x82\x81K\xc8\x07\x18\x1a\x00\x8c\x1bs[\'-F]#r*w\x05\x1d\x943
C,:\x94\xe08\xa4\x0c+\x03\xc6t\xfce\x81\xd0\x92\xe4\x1e\xe9EQ\xae2\xd5E\xd97\x13\x89m\xd6\xd2\nP\x93\xe0\xfd\xf
4\xf6(E\x9e\xe7\x80\xe3\xa3<\xbd\xbe\xd4X\x14\xd13\xd7\x1e\xab\'\x8a\x94\x9e\xc2\x80\xd03\x80\xb3\xc1Kj-\xf7f\x
ea\xb5\xb7R\x9e\xaa\x81\x90\x94>\xc8Q\x04t\t\xd4y\x97\x16m\x9f\x89f2\xe8(DL\xbb\xca$\x80\xf5\x16\xa5I.Fo\xa2\xe
6\x01\x1f\xdb\\\x8c\xd8\x1d?h\xb1:8|\x8cex\xb4\xad\xb4$m\xfd\xbdXUn#(\x01$\x92\x94{\x1f\xdb\xdbR\x15\x16\x0f\xd
aR\x118\xfbL\xe3woK\xb7\r\xc2\x98\x94T\xf0\x8a\x07\x17\x1c>V>\xda\xaf\x1b\x9e]jG\x98\x03\xf3\tm\n\xcd\x08\xa1H~
\x19o\x05I\x19#W\xa9R5%\xe1\xee\x8bwm\xbam\xad\x1ae\xd3FC\xf3\xe1z%M%\xd5\x90\xb0\xd7\x1f\xcb\xd0\'#\xa4\xf41\x
e4\x82z\xc3\x98\x15\xdcAc\x16\xf2.\xdd\xf9\xdce\xbb\t\x10$A\x8f)H%\xb6$\xb8\xd6R\x9f\x03%XW\x93\xe4\x7fmFNW\xb8
\xef\xf4\xc0\xb1\xb9mK\xa2*\x95>56\xfe\xb7\x94\xa4\x95\x96\x95&k\xa1\xa5\xc7X)Qq\'\xa0\xe18#\xc8\xc7,\xe1Y\xfaX
\x82\x850\x9e\xc8MX3S|\'n\x8d\x03h\xdfb\x8f\x12\xe7\xabIL\xca\x8b\xad\xbe\xda\x1b@\x8c\xc0R\x909\x12\xa0J\x943\
x95\x14\x91\x9e)\x1a\xa7\x0b\xa6<\x94\x0c\x9b*\xb3\xa7#Z\x85\xf6\xdd\x936U\xb5pA\xba\xe6\xba\xb82X}_)\x16\xa3\x
eb\xb5$!YK\xccq\x00\xb4\xb0\xd2\xc0\x089\xc1\t)\'8\x17\x81hA\xeeN\xc6\x9b\x90\x8a{J\xd9\x15J;l\\\xf1>R\xad\x06{
\xd0*\xcc\xa1!\xa4/\x8f\xd2\xd3\xe9\xc1>T\x0eq\x81\x92:\x1djT\xe4\xc9M\xdc~B\x15\xb5\t#U\x95F\xb4\x9aa\xca{jN\x
14\x85\xcbJ\x01u\xe5\x95\x1f\xcc\xaf$\xa4\x8f\x7f:\xc6\x01D\xd4^[\x9f\xdd\xaa,Rb\xcaG\xf9\xb28\x89"\xa2\x87\x9c
W\xce\x9f\xc8\x9f\xe4\t>\x14\t\xc8\xcfZR\x90\xad\xa35\xc1aFF\xb9\xa9\xd0\xe7\xdd\x0c\xca\x15\x14T#\xc6p\xfa\xb0
\x98\xfe\x1a\xdfFp\x9eG\xc2\x8et\xd7%\x9a\xc4\x105_\x10\xa2\xcb\xben\xfbjK\xf61\xa7\x890j\xaa*\x88\xcb\x87\xb8k
J\xc1Rp\x7f0\xc1\xd3\x10\xb0\x05\x07\xcc\x03@\xdc\xbd\xf8\x1b\xde\x9b~\x1c\xbb\x8d\x8a\xcc\xd4\xb5\xf8\xe5\xd12
Aqg\t[^\xa7\xa4\x9c\x0f\xb7\x16\xc64\x1e>E\xb6c\xf2I\x8c\xf2q\xdf\x15\xfb\x01\xff\x00\x11\x87\xba;\xab\xfeI\xa8
I\xb2m\xa8\xe8]"\xa1\x95\xb8\xf3\xca\xe9D\xf9#Udp\x06\xbed\xd8\x92\xacAI\xf1\'l\xf4\xc87T\xfah4\xa9\x88\ne\xf4\
x9f\xcaO\xb1\xd2\x1d8JW)}F\x05\xa7\xbdt\x0b\xa9\x85\x11J\x1e\xa6\x08i\xee\xb1\xfb\xfe\x9a\x10n\x19E\xf9\x95\x94
\x8d\x9e\xdd9U\xb7o\x8bB!\xc1Y$\xf2\xc62|\x83\xef\xafQ\'PK\x00:\x8d\x0b\n\xe8\xa7\xde<\xf6\xff\x00r\xdag\xe6\x8
2BK\x9c|\xff\x00\xed\xa6\x86\xedOQL\xbc\xa0f\xedR\xa5\xed\xf5i\x14;RB\x16\xc1QP\x1c\xb2:\xfd~\xfa\xc6FSW4r\n\x0
1\x82\x15G!_\x8e\x8aK\x8a\x02Z\x13\x85\x0eC#\xfa\xfb\xebT+h\xf76\xabqq\xbdVMN\xd8\x88\x9e\x12O\x11\x8eI\n\xceF\
xb3(\xa5\x9a\xa3w\x16\xb4k\x1e\xbbY\x9aT\x97\x96\x86\x942\x018\x07R>6v\xd4\x7f\xa9\xc4\xdc\xba\x99c=n\xdb\xaf\x

b3\xf8cO\xadI\xcf<a_\xae\xb367U\xa1=\x8b"\x16\xb3\x16\xb3hq\x12\x85"5-yZ\x8a\x94Bs\x85\x7f\xed\xaex\xf1\x9a\xe5
\xc3:\x05\xd4\x02\xad\xd3*\xd1\xab\xbe\xa3\x8c\xfa=\xf6G@~\xb8\xd5\x0b\x88\xdd\x19;\xb8\xf8\x86\xd4\xeb\x9a\xdb
\xa5Z\xc5\x85\xcaW\xae\xbf\xcc\xb0\xa1\x9c}\xb4nP{D\x05S\xfb\xa1\xc4\x1d\xb7\x85\xbc\xf6t\xcb\x8a\xd4\x0e\xd2\x
dd\x86\xf2\x9c]5\x84\x14%\xb4\x15vr\x0fh>\xe0\xff\x00\xcd\xd6\xaeN9V\xcc\x99\xef\x11\xa3:*\xd3\x81L\x81\x16\x83
6DgW\x1eb\xd8\x91\x15\xb4\xf1u.\xe7)P9\xc9\x1f\xa8\x04\x0e\xb3\xa1\xe0\xa7F\x0f&\x02\xc4\x91{\xd9\xb5\x14m-b\xa
dW\xa4\xcb\x96jT\xa92\xe9\xb2\xe3\xc7K\xee\xc7\x11\xbe\xaeEJ\xe9MeA\x04\xe3\xdb=\x105\xe6U\x0b\xb9\xe0l\x80 \x9
5\x83R\xdb\xea\xdd\x93N\x95K\xdc%Gu\x08K\x14\xc6\xe16\x8f[\xe6\xb0\x14\x83\xd7i\x00\x0e\xd4\x08\xe3\xd7\x9c\xea
t(@?"5\xc3\x03B;6M\x8a|M\xc9^\xde\xc3\x92\xe3Q\xd7\x19t\x8a\xacGx\xba\x18y\xe4\xb5\xdbn \x94\xf2\xf5\xbde\xf7\x
82R\x9cw\xed~\x12I\xa3\xd4\x9b!\xd5\xc8\x9b\x85m\xdd\xb6\x9c\xc6\xa9\xb5\xcaJZg\x82X\x94\x12\xac!\xeeC\x1c\x9c\
xeb(X\x08\xce\x0e>\xac\xfb\x8dk#\xad\xd4\xc5`N\xfb\x8b\xba\xa5:\x15\xb0\xd3\x8a\x9fQ\x91\xf2kq\x0e\x14\x8c\xa8r
\xe5\x8fQ\x03\xc8\xe8\xa4\xff\x00]!\x80\x11\xe8MJ\nL\xf8S\xab\x02\x81"r\xe3\xc6zPq/F\x00\xa8#=\x84\x95\x0ct\x7f
\xdbI`\x1b\xb8\xd0H\x10\xda\xdf\xb5\x95ef+\xd5\xd4\xd5_\x99\xc8F\x96\xd1\x04\x85g\x90\x03\xc7_\xae\x8dTc]\x98\x
a6,\xf7\x08w\xd9\x86 \xd9*\xbd\xa2:\xecj\x84\x08\xe6J\xcbJ\xc7\xd3\xc4\x85\xf6?c\xfd\xb4Y\xc8\x18\xecv\x04\xf6
y\x80bsc\xeb\x8d[\x94:C\xd5H\xd9e1\x1amxV>\xa2y\x13\xfe\xfa\x9f\x00Q\x89nS\xe4Q\xc8\xd55\x9c\xaf\xf4\xff\x00~m\
t\x16\x1a\x14\xb8u\x18\x10\x12\xf5.R\xd5\xf5\x95c\xc6}\xc6\xae,\x99G\t\x01V^\xa0\x8d\x8f\xbc\xcb\x9bBsi7\x88\xb
4\xe8\x8e\x92\x88\xc3\x96q\x8e\xb4\x91\x96\xbd\xad\x08b\xbd\xac\xea\xdd\xcbdQ\x1f\x8b\x16\xd6Z\xd4\xb6\x0e\x16\
xdaOg\xff\x00}g%\xfb\xca\x00=\xcdS\xb1\xdb\xfbC\x8dg~\x19P\x88\xa8\xce\x96\xfe\xafP{}\xf4\xd5")\x94\xddE6\xedUm
\xf7\xee\t\x13\xe8\x15\x8fFS\xca\xe4\x87\x90\xaf\x07\xed\xa5\xf2\x13YO\xee\x88\x9b\xbe\xff\x00\xddj}t\xc5\x939r
H8C\x8aI\xe8jW\xc9\x95MJ1"\x1f\xdd%[\xd5[\x8e\x15~5\xce\xd2\xdc+R\x81|\x00z\xfb\xe9\x98\xb2\xb7+32c@4a\xa5\xdbN
;\x8f=\x90\xb9\n\xc0G\xfc,\x7f\xe9\xa7\xb9.bV\x81\x97\x16\xdd\xablP!\xaa\x14\xf8@\xbc\x81\x82T\x93\xe3Z\xab\xee
\xd4\x1c\x85*\xe0\xad\xca\x98\xb3]T6\x14\x92\xd6O\xd0~\xdf\xbe\xa9m\x8ftB\x1f\xb4\xae\xa5\xd9\x14\xa6\x08\x96\x
a6\x19[J\xcf O\xbe\x97\xc1\x08\xb1\t\x9b\x89\x82\xfb\xb5\xb4V\xdd\xc2\xe8\\\x04\xa1\n\t\n!\xaci90\xa8\xea9\x18\
\@\xee\x86\xd2\xd5\xad\xa7\x0c\x88\xec\xa9H\xf2\x13\x83\x9dA\x93\x1b)\xeaT\xbe\xee\xa6\x9e\xdb\x9a\xcd\xb5i]\x9
3\xe8\xfbS\xf2\xf55\x84\xa5\xe7 \x19\xa8Qu\xa5\x14\xfd\x04c\x8f\x12G\xbfc\x07\xbd^\x18)\xd4\x99\x81e\xf7K\xeb\x
dffa\xdeRO\xe1\xd4\x98\x94\xafQ\xa7$ST\xb7\xdaZV\xea\xb9z\x8d\x05\x15%@~R\x9c\xa9Dg\xa1\x8d<\x80\xc4}\xa2\xb6\x
b3\xf5r\xcf\xb7d\xdaA\x0b\xb4^m\xb4\xd1\x11\x12l\x838!+qM\xa8)\xa01\xcdHVH)%#<\x0f\x95dk"\x95\xa8\xae\\Ze\x1d\x
93\xff\x00)\xed\x8c\xaa\x84J\xbdA\x95\xc8\x95\x15l\xd1\xaa\xfc\x96\xa4a\xc4\x8c\x80JO\x12\x142\xa4\xe3\xc9\x1d\
xf5\xae^ \xab\xdfs\xa1\x97\x90\xaa\x8e\xdb\x8e\xf8\xac[\x8e[\x14{j\xbbV\x85Tf\x84\xcc\x9a\xb9\x89\x18\xfc\xe4\x
84\x89-8\xe1\'(.)G\x05!G\xa2\xd1\xf6s\xe9\xb1X\xa8\x0b$)d\x99eP\xdc[\x96\xa9Y\xa8\xd476\x93J\r\xcd\xa8\x87\x83\
xcc\xad+i\xa7\x17\xc7\x98*\xe2\x81\x8eG=g\n\'\xf6\x0f\\\x8cM\x18\xb3\x8cV\xa0\x0e\xf2\xd2\xafs\x16\x05b\xc9\xa9
\xc6r8\x8aW\xf2\xb3\x9bP\x0f\x00\xa0\x14\x90\xbc\x0e\\s\x8e\x81\xf68\xd23\x06\r\xec\x8e\xc6@\x14`f\xd6\xdemN\xb
d\x93E\xbd-wX\xff\x00\xc3a.\x179\x1eg\xa29\x00?\x97\xd8\x81\xe3H\x0f\xbfp\x8e)\xafl`C\xa3T\x9d-&\x9c\xa9>\x8f\x
a6\xa1\x01\xd2\xc6P\xf0O\\B\xbe\xf8\x1a\xf1\xe8\t\x80\xa8\x9e\xf7B\xf3\xa6\xb3\xf0\xd5r\xc7\x0f\x94>\xb8E\x88\x
a4\xab+)qa\nH\xff\x00\xfa:\xcc\xce?\x86hxT\x9c\xc0\x88\xbc\xb0aR%\xd4E\xbdXJ\x03\x08HNR\xa3\x90\xb4\xf5\x8f\xf6
\xd1\xa8\x1c@\x82I\xb20\xe69\xb6\xfa\\95F\x04j\xc3\xac?\x18\x16\\\xf4\xce\x07\xa6<d\xe8\xb5w\x07l\xbdJ\r\xcf\xd
b\x88\xd3\xaf\xf5V\xe8\x15\x85\xa29\x00\x8c\xab**\xf7\xd4>N,\xd9|\x9fa\xd4~&\xc6\x98,\xf7#\xd5\xac\xcb\x82\x88\
xccz\xcd\x1a[\xa1\xf2\xb1\x9eI=\xea\xac8\x8ai\xb7\x15\x93)\xab\x021\xf6\xee\xabt\x99&5\xc5Yq\x87\x16\x80\x02B\x
08\x18?m?jj,\x90\xc2\xc9\xdc,\x89dEUH\x86\x9erZ\xd5\xf5\xe1j\xeck\xdc\x0b\x1fl\xd1\x90\x05\xa3=\\\xcd\xd2\x9dmI
b\x00C\xcc$\x85(\x8c\xf8\xd1Z\x11Q|\xdcO\xd6%J5n\x12\xe9R)\x08l\xa3\xff\x005J\xfaU\xfa\xeb\x00\x0c54\x92\x05\xc
e\xf3\xdcU\x9fQB[\x08\xfa\x8e\x01K\x87\x03\xfa\xe9\xc2\x97\xb8\xb1\xbb\x9c)\xf7\xd4\xa6n\x84\xbd46\xe3*9__\xfd\
xce\x87\x9d>\xa6\x84\xbd\x99S\xba\xef\xc5\x95Q2\xa0\xb1\x82\xb4\xe3\x03\x1e\x7f\xbe\x96\xf9\x0b\x18\xd4E\ts\xfb
\xb5\x94\xbb~-)\xf9W\x8c\xf7\x12\x14\xa3\xfc,\xf4?]\x166\x00Q\x9et\xe5\x047~-\x1e\x83^\x8cl\xab\xa5o=%_\xc3\x8a
\x15\xd2\xf3\xed\x8d\x13\xd0\xe8\xee(\x06\x1d\xce\x97\x16\xde\xde5\x8a[\x1f\x8c\xd2\xdbm\xc2\xd0#\xeb\xc1\x1f\x
be\x95\x95Y\x85F\xa3\xf1\xdcC\xb3\xb9;\x91H\xb8\x0e\xf0X\x16a\xa6D\x11\x983\x92\x98\xb8\xc6F\x14\x94\xa4v\x06H\
xf3\xaex\xca\xce\xd2\xb3\x8c*\xfb\xba\x84\xd6\xd6\xe7\xd7wF:mj\xea\xdd\x0f\n\x82$\x08\xad4O\xaa\x8c\xfdC\xfe\x8
cc\xfe\xda\xa7\x16s\x93\xdab\x9d\x15G !\xa2\xb7\xb9Wm\xc4\xfd\xa1]\xa6F\x8f\x1e2\x04\xa9\xd5\x04;\xe9\xf1t+ \xa
9C\xaey$\x94\xe3\xd8\r=r?>?\x11\r\x8dN;\xbd\x99\x1a\x85k\xd3\xd8\xbd\x1f\xb1\xa51M\x87=\xea\x9a\xdcT\x86;)o\x88
X_%\xf4\xdaOJ>\xddk\x17\x17\x17 |\xc1g\x0c\x9c\xe5\x1e\xfa\xc9\x85k\xee\x1c+\xf2EDT\xcc\x1a\xba\x92\xf3.\xba\x9
4\x85\xa1\x05\n(\xc2\xbf?I\x03\xea\x039Vs\x8d-\xe9\x1a\xfb\xdcj{\xd2\xa0\xe5gw\x9e\xb8\xa9R\xed\xea\r5\xa8\xb4\
xd7j\xab\tJ\x17\xc9\xc6\xdaJ\xca\x9bIWx\xc2V\x13\xd9\'\xa0}\xb4+\x91\x8b\\\xf1\xc6\x07q\xcfhP\xc5\xff\x00\xb6\x
ee\xbfP\x88\xa7\xaa4\xd7\xcc\x87)\x92\xd7\xc9m\x81\x92\xe3@c=\xa5!X\x1d\xe5GW{Y$\xe4Q\xd4S\xdf\x1by^\xaa\xcb\x9
7U\xb4\xa9\xed\xbd\x1a;\xc9u\x95\xb2\xzL\x81\xf9\x08\'\xb2\x0erN\xa6\xc8\x8c\xc65\x1c\x08Ain\x0bS\xe8\xac\xd8\xf
4y\x06\x0c\x98H\x0bm.\x02R\x1d\x19?\xdc\x8c\xfe\xfa\x12\xc4j\x18\x00\xae\xe0\x96\xe4\xd5a\xd7h4\xdb\x10$\t\xd5;
\x82\x13E\tH\x01(\x0e\xf3Z\x87\xdf)I\xd4\xd9\x81la~\xe4J\xb0\x9e\x0cO\xc0\x12\x9a\xec\xbdWL1\xd8E!\xb68\xc98\x9
1\x91\x9eC\xa5\x03\x8f\xdbL\xf5E\x90>\xf1<mE\xcb\x8d\xbb\xdf\x08v\xcc\xf1\n\xa7\x04:_p\x95\xb8<\x11\xed\xde\xbd
\xea\x83\xa34-F5\n\x88\xfd\xd1=5\x06\x14\xafEn\x850\x07\x8f?\xef\xa7\xe2fq`D\xe4\x1cEFe\x1e]\xac\xee\xe2\xd3m\x
ca\xfdA\xd3\x1f\xe9\x12#\xfag \x0f8:{en\x14\x06\xe4\xc4(z\'SA[\xd6\xee\xc0\xdc\xd5)\x94\x97\x9ej#Im)\x86\xf4\x8
5\x80\xaeX\xfb\x9e\xf5\xc6l\xdef\x0c\xdc\xb2\xad\x83:H\x982c\xe2\x87c\xef\x15\x97\xce\xd8\xdd6M\xc3*%.\xe7J\xa3
\xa9C\xe5\xa5\xb4\xae\x88>\xd9\xd7O\x1b\x86\x1a\x90\x10\xe1\xa8\xff\x00\xbc\xfddY\xf0cJ\x9a\x8b\xcd\xd5\xccIo\x
92\eC\xc9\x1e\xf8\xd3q\xa0m\x89\xa4\xb0\xd1\x83mR\xa5D\xaf\xbd.D\x87"C\n\xc3\x0c\x04\xe0\xa8g\xee5\x85H\x1b\x9
9`\xf7.+P\xad\xba\xb5%Q\xbd\x17\x9c|\xa0\xf1\xc6T\xac\xfe\xda\xd2A\x00O-\r\xc0\xf9\xf6\xe2!\xa6;\xf1ja+K\x98SNt
q\xfbhH\x00\xf77\xf7-\x88C[\x87Gj\x03B\xbfSi\x05\xc4\x8c-.w\xfdt#.\x12\xc4\\w\x0c\xa8\xbb\xd4\x16\xb9\xec\xa9B\
x1b\x86\x8bU.\xb4\xa4\x85#\xdc\x91\x8f:\xf5)\xea\x10\xfd\xb1W]\xa7Lv\xf2\x8f_KyM5\x03\xf8\x89\x1d\x05{\xe9\x01\
x89\xcbC\xe2{\x80\xe3\xb9ep\xef\x8d~\xadm\xbc\xf3\xd5\x8e+k)m)8#\x1f\xf7\xd1;\xb5\xdd\xc3\x01x\xee\t\xec\xef\xc
4\x8d.\x9beI\xb6\xeej\x0cy\x0c\xad\r\xa0-LeAhI\x01\n?\xce\x92\x00\xef\xdb\x03Sbe\xc6,\x88Y\x15\xb2\x0e7=\xed5\x
d3m\xd3\xdbr\xe2\xa2Q\xfeb[\x92\x94\xcb\xc2{\x03\x929$\xe3\xd3?|\xe7\x03\x1a\xa5\n\xdf!\x12\xdc\xca\xf1&/wq\xaa
\x85\x1e\xf1\x9c\xa6!\xceL\x97\x1eT\xb7\tP\xc3\xaaqg\n8\xeb$\x93\xd7\xe9\xfd\xd5\x96\xc1\xb1\x19\x89\x87\n0\x9e
\xcc\xde-\xec\xdb\xea\xd2\xf7\x9d\x16\n\xeb\n\x91\x01pW*KE+\x8aR\x9e(^\x148\x1e\x88O\xb7\x8f\xb7Zkf\xc8=\xfd\xc
4zx\x98p\xbf\xcc\x85&\xf5N\xe4\xd0\x1e\xb9w\x16\n\xa9\xd2\x1c\xaf-\xa9\ti\x1c\xd6\xda\x80\x04\x05\xfa\x85!Y\xca
p\xac\x94\x9c\x90s\xd1\x0b\xb3\x959\x18\xc0=6\x00}\xa4j+T&o#\x19\x15\x03\x1e++qM\x98\xb8um\x8c\xab\x8a\x88\xe4B
\xb0H\xcf~\x0eA\xf04H\x00\x9ea\xba\x11\x93\xb1[\xd2\xd6\xd1\xdem\xdcW\x02\x94\xfc\x7f\x96t%\xd7\x96J\x1eq\x04\x
1e\x7fI\xc9\xc2\x14\x7f0\xce\x0f\xb0\xd3\x06@\x82\xe2\xd99\xd0\x8c\xda\x97\xf9B\xa5}K\xdcM\x93\xb8#\xbbB\xac\xc
b\xe5\x0e5B)J\x10\xa5\xf6[\x1dxP\xcf^1\x8c\x0c\x1c\x06\x93k\xcdb\x82\x95\xf6\x99\x9d\xdf\xaa\xd5`\xee\x85Yo\xda
r\xdf"Cn\xa5\x8a[\xdcAo\x9e\x14\x07]\x1c\x0c\x0f\xb6\xb9>nf\xf1\xcf)\xd1\xf1\xb1\x96Z\x12\xb6\xe3\x99Z\xbd7V\x9
bo\xda\x1bkq\xd3\xdb\x8b"]V\x9e\xf5u\x9f\xe3>\x96XP!!\x03<\x02\x9c\x1d\xff\x00\xd4>\xda\x93\xc7\xfa\x96\x1f/\x8
b\x124w\xfc\xe3N\x17Dj\x1d\xcbJf\xc4\xd5wV\x04\x9b\x8a\xe5\xaa.\x8e\xa0\xe9\x94\xe4e\'\xf3\xa3\xc2\xb1\x9frF\xb
a$\x96\xda\rI\x14(4\xdd\xc9\x96M\xaff\xd2\xaa\xec\xc4\xaa\xda3\x95\r\x94\x95\x034\x80_\x03\xf9\xd0tx1\xe4\xe5o=
\x93"W\x10.=,\xf5S\xee\tl=lR\xdf-0\x80\xdc(\xe9N9\xaf\xed\x81\xef\x8cj\xb5\xe2\xa3f\x84CY\xf8\x97[\xa1h\xdd6#pk
w\xac)\x14g\xaa\x0e\xf2a\xd7Y\n\xc7\xeeF\xb0\xe4\xc2\x07\xbd\xbb\x82\x13!\xe8T\x87;\x9d\xc8\xa6\xd1\n[\xee\xbe\
xd3aY)\xe2\x1e\x03\xdf\xce\x8c*_\xb7\xa84\xd7\xde\xe5\x8c-\xd1\x94\xe3B\x1bN\xa6Dx\xf8C\xad\x97r[V\x8a\x94\x1bQ
4\x83^\xe96\x15\x1e|w\x7f\x1b\x89W[I\xe4V\xf3\x01_\x9b\xac\xf8\xd1\xa0P,\xc5\xb1 \xea\x17\xbbI\xaf~\x1bN\xab\xc
8\xa5\xa9\xd4N\'\x8a\x9d\x8eABG\xef\xac9R\xe8B\x17Ve|w\x05\xb5x3p\xa9/,:\xa2\x86\xdb1y!G\xf5\xf6\x07B\xb6\r\xfc
\xcd$K\t4j\x04\x8a\xe2\xa5\xd7%5\xebL\x07\x8b(G\xe5\xf7\xf6\x1ae\'\xee1a\x8f(\xaf\xba\xad%H\xb8\x9d\xa6\xb1W\xf
9\x94\xfa\xdcY#\'\x07=\rJp\x12\xd6\x9dJW \xaa}\xcb\xd9\x13&\xed\xc2\xe2S\xeb\xd0\x0c\xf4\xf4\x1de\x03\xcaG\xb6\
x7fl\xe9\x80\x00\xb4L\xc2M\xe8N\xbb\xe5z\xec\xae\xe7\xda\x91\xe8[ob\xb9G\x9e\x94q\x94\xa52\x94\xf7\xef\x92\x9e\
x8f\xef\xa9q\xe2p\xed\xf6\x8cg\xc6\xcb@Q\x99\xb3q\xed\xebJ\x8b\x01T\xcaE\xdd\r\xe9\xec\xc8m\x12\xd9C\xa1E\xa0\x
af$\xebr\xe4\xc6)A\xdcb.B,\x88\x89\x9bY\x84\x80\xc4\x8b~o\xd6\xd3)nK!\x19\xc3\x99\x04\x10>\xd8\xd4\xdd\x9a\x8e\
x876\xbe\xe2\xc0k\xd3zchm\xd5)\x1e\x9aR\x83\x86\xc8#\'\x00\xf6I\xff\x00\xbe\x9c\x8eR(\x8b1\x996\xb1HT)\xf7]N\x0
7\'"1\x15\xf9p\xa6$\x06\xa5) \x83\xc7\x1d\xf2\xec\xf8\xff\x00\x9b\xf4\xd5A\x817&\x1c\x80\xa8On\xde\x96\x9d\xdbi

\xfc\xb52\x8e\x9a{\xef,9.+\xc7\x83rY\xc2\x81\x19H\xc0!\x1d\x0c\x0c\x8e\xcfy\xc6\x880qDAe*}\xb1ysQ\xa9\xd6\xab\x
a9\xb1Y\x92\xe3\r\xaa\xb7\n\xa3O\x98\xb4(\xb6\xe4oE\xf4\xff\x00\x11\x1d\x04\xa5*\x0c{\x80pI\x1eF\x90P\xa5\x88\x
dcg\x9e\xfe\xd3\xbd\xc7kQ\xad\xfb"\x91\\\x8d2D\xd2\xe3\x8e\xb1Pq\x04(\x82\x00(\x08\x1fl\x85y\xf0\x00\xeb\xbdz\x
80I\xa0\xdbT\xa2\x99\x12\xd6\x81_]B\x85-\x84\xb4\xd3l<\xc2\x1f\t(W1\x85\x87>\xe4+\xdb\xc6\x15\xac\xe1\xb9\xe2\x
da FU:\xe2\x89l\xd3\xe5[\xf2\x1dn\x9d\x1a\xb3\x11\xb7\xa0>\x85+1\x9fo\x1c]\xcfD\xfen<\xbe\xc7\xfb\x137\x18\xb0\
xa4\xa92\xden\xcaV.\x06cHv|d\xd6`GRW!\xa5\xab2\x12\xa5\x15%\xd3\xfb\xfb\xfb\xeb\x89\xe4\xf9\x1e\xb6m\x0b\x03\xb
1\x19\xe3\xf9XP\x95&\xbf?\x12\xdbm.\xe9\xd2>,`1r2\x85\xae\xda\xdbw\x90]m\x1e\\\x931\xa4x\xfdP\xc6\x7f\xa9\xd7\x
cc\xe5\xf1\xb2\x9cl1\xfc\xb8\xff\x00i\xd5\x7f\'\x0e<a\x98\xea\xbf\xbc2\xb8\xad\xba\xbd.\x9d>m\x93[\xf5\x83\xeaq
\xc94\xe7\xa3 \xa1EC\xc2s\xe3\xbf\xfb\xeb\xeb\xfc\x1f\'\xf8l\x01\x18np\xb2\xf9X\xb3\xe5\xd9\xab\x83\xcc\xd9\xb6
\xd5\xf3m4E\x9f"mU\xb0\x86\xe5\xd1[\xc8\x94\x85\x1cv\xdaA\xf7\xf6\x03\xa3\xaf\x02\xc1\xdb,<Y\xb2_\xa7\x94h\xf4D
\xedV\xb3i\x96\xcdK\xfd7\xac\xddK\xa4M\x81\x97M5\xd7\xcbS\xc2\x9cH)G\x01\x91\x91\xf7\xcfX\xd0y\x01\xfc\x83\xc0\
xb7[\x96\x02q\xfb\xd4w\x07\xee\xeb\xb6\xf2\xb6\x1f\xa1\xdb\xe2\xafX\x9b\x1a#n917\x03^\xaa\xca\x01\xcf\x06\x902\
x7f\xae\xbaX\xdb2\xa0\x0c\xa3_\x88\xa6\xf4\x99\xa9X\xff\x00S\x1a\xbb\x19j\\\x1b\x9d25z\xe1\x92\xd52\x0b\xac\xa8
R\x90\x96xHY\xc1!>\x8f \xb3\x9e$\x0eX\x07Ta\x19\xf3\x0eDP\x93\xe5lX\xb4\r\xc2\x8b\xff\x00`\xfe\x1c6J\x9d*\xed\x
b9\xea\x13\xdb\x8a\xfb\x8d\xba\xba\x8cw[mR\xc2\xf3\xc8\xb6\xd9\x0b)\xc2\xb1\xf9\xb1\x90G\xdcj\x91\x8d\x11I1\x03
#;\x80\xa28>\x1d,\xbd\xb2\xda\xa9\xb0w\x82\xd9\xa9\xd3\xae\xcbq\xf62\x7f\x13m-\xb8\xcf!\xd1O\x94\xa9@\xfd\'\xc6
\tH\xf7\xd4\xf9\xfc\\\x8e\x97\x88\xc3\xc7\x9d\x15\xfd\xe2\x11n\xb6\xe9Rw_yZ\xb7l\xcbl\xcb\x80\x9aX\xf5\xdab1Ba=
\x9e\x81\xce;\xc6\xa2\xf1\xbco/\x81?2\xcc\xb9\xfctj\x1d\x18\xa7\xde\x9bKvYT*S6\xda\xe8\xb4\xe4\xb8\xafM\xd7\xf0
\x96\xdep\xf4\x0f/l\x8c\xea\xaf\x1f\xc5\xf2\x95\xf9e\x88\xcd\xe4\xe0(\x15`\rZ\x04\x9bf\x0c\xaa5Z\xf1\x80\xe4\x8
6\xf3\xf2\xcbR9\x9e`~^@\xea\xee\x0c\x82\x8e\xe4\xbas\xa1\x11\x8e\xfcP]{MW\x05v\xb4\t2$-y\x98\xb4g\x00t1\xfa\xfe
\xba\xe1y~g\x95\x81\xf5\xd4\xee\xf8^7\x8d\x99m\x86\xe1\x15\xb7\xbbT]\xe2\x8d*\\z\xacv\xaa\xef4@\x87!a?\xc4>N|\x
804\xbc?Q\xbd8\x85\x93\xe9\xd4\xc4\xe31{\xf1\x07L\xf8\x89\xb5hOQ,\x8d\xb9\x97PxF\xfe%J:\xc7\x00T:\xe3\x8f:\xf6\
x7f;!R\x14P\x99\x8b\xc2=\x9e\xe6-\xb86c\xe26\x91Rr\xbfY\xb4\xaa\xd1\xdeZ\xc3\x8f\x95%__\xd4U\x83\xdfc\\\xcf\xc9
\xeetJ{8\x81\x0b"Z.Sjk\x93\x1aK\x0bQY[\\\xd6s\xd7\xb1\x1a\xec\x90\x18\xd8\x9c`\xf5\xa1\n(\xd0\xebT\xb57TR\x10\x
daR\xf2U\xcb\xd3\x1d\x93\xd89#\xc7Z\xde\\MOP1\xd1o\xff\x00\x91e\xd9\x0f\xce\xbd#\xb7:\\iM\xfc\xb4GG\xd6\x16\xb2
x\xa989\x00\xfb\x9e\xfd\xb5f6^\xe4\x99C\x03\xadGm\xb7\xb2\x94\xfd\xd6\xa2D\xb8,\xe6\xda\x89@r\x00i\xc8r\x07\'9\
xe0\x95\xb8\n@)PO\x1e\xfd\x89\xd5h\x8b\x90r\x12vwSQ!\xba\x12\xeb\xed\xed\xa4\x8aeF\xac\x85K\xb3*\xce\xd1\xabA\x
03\x9b\x92c\xb8}F\x1c\xec\x03\xd7\x15\x0f|\x1c\x1e\xc7Z\x9d\xc3\x04\xa6\xf8\x8eF\xb6\xd7\xcc\x1aj\xf3\xa3\xb7n@
\xb1\xa8\xd5i\x12\x9dbk\xc8\x7f\xd4g\x01\xd0\xa5\xa4\xa5I)\x19$\x02RG\x7f\x97\xf5\xd4\xae\xca\x08\xa9B\xa9\x07p
\x99v\xb5\x16\xde\xb7\x9c~\xa2\xd4eH\x8a\xfbR#\xa4<2\xdb*mIP\xe2S\x9c\xf2\xc1P `$\x03\xd9\xd1\x06\x00\xd4\x16\x
17\xb86\xfdm\xcd\xcb\x0f\xd1\xeeK\x91\xb8\xa8\xa7\xba\xb8p\x9bH\x1e\xa3\xa5y\xcfD\xe4\x04\xa4$\xe3\xcf\xd4<jO3\
xcb_\x1c\xaa\xd5\x93\x14\xee\x10\x88}\xf8\xee\xec\xed\x83v\xf6\xe09\x11U\x18\x14\xc6\x8cj\xdb\x90\x81%\xd8\xd8O
\x17\x88?\x94\xa7\x888\xef\xdf\xbdp\xb26L\x19\xc6B=\xa4\xf7\x10x\xe6b\x82\x16\xecu\xe5a\xdf\x9f\x15\x97\xe5n\xb
3W\x97J\xa2J\xa4\xd2Xf\xa4\x9a*\x9dmm\xb4\xcb\xae\x94\x879\'\xd3R\x8b\xa9)\xfaT\x17\x9fmt\x9f\x12#5\x8dw\x7f\xf
9\xf7\xf8\x94\xb6\x1c\x7f\xc2*do\x93\xfe\xf1\x93G\\\'\xa9\xb3%\xd0\xaa\r\xbc\\l\xfd%\xc4\x97X\x04u\xcd\x00\x90\
x95\x01\x8e\x89\xf3\xa9X\xfbk\xa19\xab\x89W(\xe6\x0f\x1f\xe5\xf13\xfe\xd2\xee\x86\xd9\xed\xb6\xef\xd5\xa5\xee\x
16\xf5\xd4\xa8\xf5\x08\xef.4\xa94\xf9\xe9[\xe1\xa5v\x85\x83\xc7\x8a\x8a|\xe1$`u\xe7F\xa8\xbe\x98<\xc8\xfcO\xa07
\x90\x86\xc7\xa8Im/cwR\xe6\xaf4\xad\xdf~\xe5q\xba\xca\x9b\xa7\\\xad\x00\x97d2\x10\x08K\tR\xb9\x1c\xab=\xab\xbc\
x83\x8d\\\xbeN4\x04q\xbd\xf7&e\xce\x1f\xb1_\xcf\xff\x00\xa8\xc1\xd8\r\xbbnU9K\xdf;\xa5\x0f\x1a|\xf2\xf5\x16<E\x
81!\xe4\x0f\xa7\xd3\x0e\x92U\xcc\x9f\xa7\t\x04\x82\x7f\x9b\xc6\xbb\x18q\x0f\xfa\xcf\xf2\x92\xe7\xca\xfc\xa9usM|
?Rl\x08t\xff\x00\xf3\xcc8u:\xaak\xb2\x95>\x9bN\xf4\x90\xc3\xf1\xe2\xa4\x12\x86\x9dS\x9c\xd6\xa7\x8eTTS\x90\x06\
x08\x1fV\xaeZT\xa9\x19\xe4_\xf9E\xbe\xfc\xfcHY5*\xd4\xa6-+z\x0c\xc7\xdc!\xb0\x891VT\xf3\x8a\xcaT\x9fU\xd3\xc8\x
80A\x05@w\xc4`{i92/U\x18\x11\x80\xb2jU\xd3\xe6o\x0cz{\x11\xb6\xf1n\xc2\x8fV\x88e7@\x92\xca\xa7F\x9e\x94\x8f\xe2
5\xe9\x15}%_\xc3\xc7\xd4\x8eJR\xbc\xe0\rx\x13v\xb3\xc0\xa5wp\x9a\xd2\xf8\x85M\x1d\xe9t\xe8Q\xe1\xc9z\x96\xe3\x8
c?#\x8b\xcd\xcai\\\x96\xa4%1\xf8\xa8\xac%\x00\x01\x95$\xf5\xd9\xf1\xa2\xf58\xeag\x00M\x89iW\xf8\xd2\xba\xe6Mr\x
d0\xaa\xd2Z\xad\xc2\xacAu\x88\t\x8a\xdbO\xbc\x1e\t)RP\xd70J\xb9\x14\x80\x8c\x95\x02G\xdfZ3\x064E\xcf6?\xfa\xacL
\xd9|ViRB-\x1a-"\x9b]\x94\xd4\x80\xe2\xd7\xf8\x8a\xa3\xc8m\xb4\xe1\x05\x0bB\x88R\x1d*8)\xec\x0e\xbb\xf3\x84\xbb
+t!\xa0\xe2,\xca\x0b\xbfm7\x1e\xec\xb4?\x0c\xa9l\xc5^\x995+\naO\xf3r:\xa3$\xf6R\xbe\x18V|e\'J\xc9\x85r\xaf\x17\
x16cS3#Z\x98-\xb6\x1bo\xb3\xd0\xearcn\t\x95o\xcb\nW\xcaJ\x83!iZ\x9c\t\xfaQ\x82<\x13\x90I\xd4M\xe0\xf8\xd7EjT<\x
ac\xdd\xa9\xdc,\xbaju\xf9v\xdc\xa1d\xee\x84\x83*\x08\r\xa6\x9fQAH_\xdb\x8a\xcf\x93\xa4?\xd2\xedI\xc2\xd7\xf82\x
cc_Sua\xea\t\x9f\xaf\xbf\x88=\xed\xb5$\xa9\x17\xa5\x81T\x01M\x14\xa5\xc9\x8c\x10\x85\x91\xee\x0f\xdb\\\xc3\xe3\
xe7^\xc1\x9d\x15\xf3\xb1\\\x12\xfcJ\x8b\x0eJ L\x98\xca\\\x90\x12\xb5JGa\xb4\x9f={\x90u\xd4]\x1a&r\x98X\xb1\'\\\
x9b\x8b@n\x04Jm1J|2\xc9j@p\xe0+9\xfa\xff\x00O~\xbd\xb1\xa6\xdat"\xbd\xc4\xcfv\r\xfb\xfe\x9f\xdd\x08\xb8\x19\xaa
\xaar\xbeMA\x86\x1c\xc6\x19Z\xc1G5$\x82\t\x19\xe8\x1f\x1d\xf7\xacC\xe9\xb5\xcda\xea\n\x87\xdbK\xf1Ip\xedl\x97\x
e9J\x96\xc9\x87>\xac\xb3!\x98\xefe\x01\xb73\xc9Hk\x18\xebL\xc5\x9d\x94\x9d\xf7\x01\xf0\x8c\x80Ew\xc4\x06\xe4\xd
e\x117\x16e!JJ\x1e\xac\xb0%<\x94\xa0\x84\x90\xf1!%\x03\xbf\xa8\xa4\xab\xcf\xb8=\xe9~C\x92\xd5{\x85\x85T\xae\x84
\x15\x1b\x92-\xf7P\xfb\xd2S%\xe4\x12\x1e}\x08\xff\x00\x8a\xb0I+ \xf8\xef\xf4\xfb\xf64\xb3\x93P\xcaV\xc48o\xe2%\
xed\xc3\xb5\x9e\xa4\xb5\x03\xd3\xaa\xcd!\xb9n\xbaB\x81Q\xc8\x1d\xf5\xe4\x91\xef\x81\x8f\x7f:\xc4$\xb0\x98\xc18\
x1a\x8c\xcb\x06e\xc7N\xb3!]\xd6U\xbb%t\xfa\x9f\xd7X\x84\xd2\xc1u\xf0\x01\xfa\xc3d\x80\x1cOc\x00\x8eIO\xdc`p\xfe
\xa7\x97\xc9L\xfe\xb3~\xce\x8f\xe3\xf3\x11\x9f\x10\xf4\xc4n[uj^\xf3\xed\xf4\x9bf\xd7\x84YiK\x0cN\x97:\n\xd9\r\x
902\xa0\x10\xa0\x0b\x87\xf4\xff\x00s\xa7\xe1\xf2F\\c\x86\xc4\xe5\x9c_\xc3\xe5\xb61a\xbe6\x9d\xf9in\x03\xb7M\xa7
\xb9U(\x91*1\xe3\xc0\xa8\xba\xc2S\xc5\xa4\xa1Y\n\r\x81\xc7\x88\xeb8\xc1\x1c\x89\x07\xbd7\xca\xf3r\xd8:#\xe7\xfe
\xe3\xf1-\xc5\xe4\xab\na\x1a\xfb{\xb6\x16\xdd\xa2\xf4J\xdd\xc9lQ\xebug8\xb8k\xb5\x04\x17$\xb8p>\xa0\xb79\x11\xf
7\xc6}\xf4\x95\xc2Yh\xeeE\xe4\xf9\\\xd8\xa84!\xb5\xe7il\xcd\xc3Gv\xe3\xdc\x0b"\xdf\rEIy\xe9\xd3\xa3\xb2\x9e\x03
8\xc9R\x92=\xfd\xbd\xf4\xf4L\x97\xf6\xad\xc1\xc5\x9b?H\xd7\x12\xd7>\xe2l\x9d\x9fOv\xd1\xda\xea\xadZk\xce\xc8RM:
\x1c\xe2\x18\x92\xb2p\xb5\xadE*R\xd2\x94\x05w\x9e=\xf9\xd7W\x12\x15[m~\'K\x1bea\xef\xa9+\xe1v\xb3Y\xdc\xcd\xebf
4\xeaL\xf9q\x12\xf0\x8e\x9f\x90\x90\x96\x98\x05*\t\n/\x94\xe1\xa5)\tVV\x80I\xcf\xb7\x91w\x8fl\xf6F\xa0g\xe21\xf
76Uwn\xf7\x9a\xb3O\x93{[7=\xc2\xa7\xa0CC+\xa3\xc1S+W\xa0\xda\xb2\x0bK(\x05@x\x15\x82\xb4\xe0\x15\x02@\xc1\x1d\xd
al\x93R@\xc2\x80\xbf\xeb\x11W%\xe7Cb=>\xef\xac\xc8r\x92i\xeb[\xd4\xb9\x8d\xac\x06]qh\'\x8b\x8d\xa7\x91R\x8a\xb3
\x91\xc8\x11\xfe\xda\x90\xe4\x1c\xad\xb5(\xf4\xd8\x8dn.\xa0\xfcT\xddV\x15bs\xf7\x07\xcdE\x94\x96\x1a\x0e\xb0\xe
3>\xbbOc\x1c\x94\xa6\xd6\xa1\xc8((+9\xe8\x01\xe3\xd9?\xc58\xc8CG\x1f\x1d\x1b\x18)\xd4\xf5{o\x85\x17u*\xcd\x07\x
a9m.)^cJR\xca\xdc,\xad@\xad\x0e!\'\xb0T3\xfd0F0\xa2\xc3\x95^-q2\xc3\x8b{p\xa2\xedU\x9f\x1e\xb7C\xb9\xdfT\xa4\xd
5\x0bu\xb8k\x88\xc4d\xcc\x8a\xeb|Zr9\r(6\xdb`<\x0f \xe8\xe6\xb4\x05 \xe0\x1d\x1f\xed[\xbb\x80)\x9a\xaaN\xbc\xb7
\n\x8d\x19\xc4\xee\xbd\xd1h\xd1\xefJ;\xb8\x80\xedJ-4.Z\x14\x9c\xf1Z\xdaqN\x96T[RJ\x90T\xe2\x12\\\xc2\x15\xc5*\x
cf\x98\x93\xb5\xdf\xfc\xcd\xe3]\xeb\xf9\xf5\x04\xa9\x7f\x19\xb7\xfd\xcf2\xb1\x17h\xea\xad\xd2i\xc2\xa0\xbfB4w=C
#\x07\x88\x0bB\xf1\xc5\x0bN:\xef\xbcw\xa1\x1eVF\xb5\x15\xdc6\xc01\xad\x98Ion4;\x92\xd4\\\x9b\xee\xce\xa5M\x99\x
1eb\xe2\xcfZc$\xb8\xdbJ))\x04\x8e\xd2\xa4\x82\xa2\n\x89\x03\xa1\x8f}92\x03a\xb7\x14P\x82\x08\x88\r\xc1^\xde\xd0
\xee\x85=nn\x03sb\x05\x97!\xb36Am\xf6\x93\xc8\xe5\xb5\x8f\n\xc6<\xfe\x9a\x99\x86%:2\xa59\x0fpf\xa6\xe5[sg\xbb:\
xb9\x0e\xa1>\x96\xa5%,%r2\xe1\xc7\x92\x90z\t\xf6\xcfZ_\xa4r5\x9e\xa1\x97\x18\xc6\xa2\xcfo\xac\x8a\x15\xca\xc4\x
95\xcf\x98\xb6\x9dn+\x8al\xab\xbc/\x1fH\x03P\xa0\x17\xb9S\xd8\x102\xae\xd3\r\xce\r\xb0\xe9K\xa5\xd22\x1d\xfd=\x
fe\xdf\xfc\xebH\x17\x17\xc8\xcfR`\xb1\r\x98\xef6\xa2\x07\x0c%*\xf7>\xff\x00\xbfz6\x00\x0b\x9a\x18\x96\xa92\x8dt
G\xa1V\x9a\xae|\xa2e\xc8?B\x10\x8c\x10\xaf \x80<\xfb\xf9\xd2\xc3\x10A\x10\x98\x06\x15,w\xe5\xef\xf3]z\x81\xb8\x
90\x17%\xef\x9d\xa6\xa62\x99\xc9\xe7\x1c\xb4\xa0\x953\xd7\x8f={\xe0\x8dfR\x18\x87\xfb\xc1\xc5\xedb\xb1\x7fQ\x82
\xbak\xca\x92\x85\x05!eIR\x01\x05M\x8f\x1e\xddx\x07\xfbh@\ncI\x12t\x13\x14J\x8f=\xc7HB\x0fi\x8c\x90\x85\xfd>\t\
x19\xc1\xf1\xdf~\x0e\x9d\xc1\x8d\x10b\x81Pw4\xb6\xc9\xfcB\xdbU\x9bB\r\xad\x12\xd6\xa9B\xaf\xc6\t\xf5\x1f\xf4\x1
7",\xc3\xcb\xfe#e)>\x9a\x8f\x95\x03\xee<\xeb\x83\xe5\x8f(+([\x06/\xc8b\x05V\x8cu\xfc,V\xaf\x9b\xda\x87X\x81M\xb
5jR\x17O\xaa:\xdc\xb9)\x80\xb4!\xb0q\xc4\x95\xac\x04\x9e\xbd\xff\x00M?\xc1\xf1<\xa7A\xed3\x95\x9b\x19\x14n\x19\
xdf\x1boZ~\xd5]^\xa7OJR\xb73\xc5?Q\x01g\x888\x1d\xf7\xd7\xf5\x1a\xe9\x9f\xa5\xe7u\xa6\xea\'\x1e2Z\xa4\xdd\x8a\x
d9]\xec\x9bCT\xdb\xd6e\x12e\xbe\x84\xb8\xe4\x18\xd1y\n\x84t\x82x0\xb1\x8e+\xc0\xf0|\xf7\xfaj\xdc\x7fNuP\xacE\x0

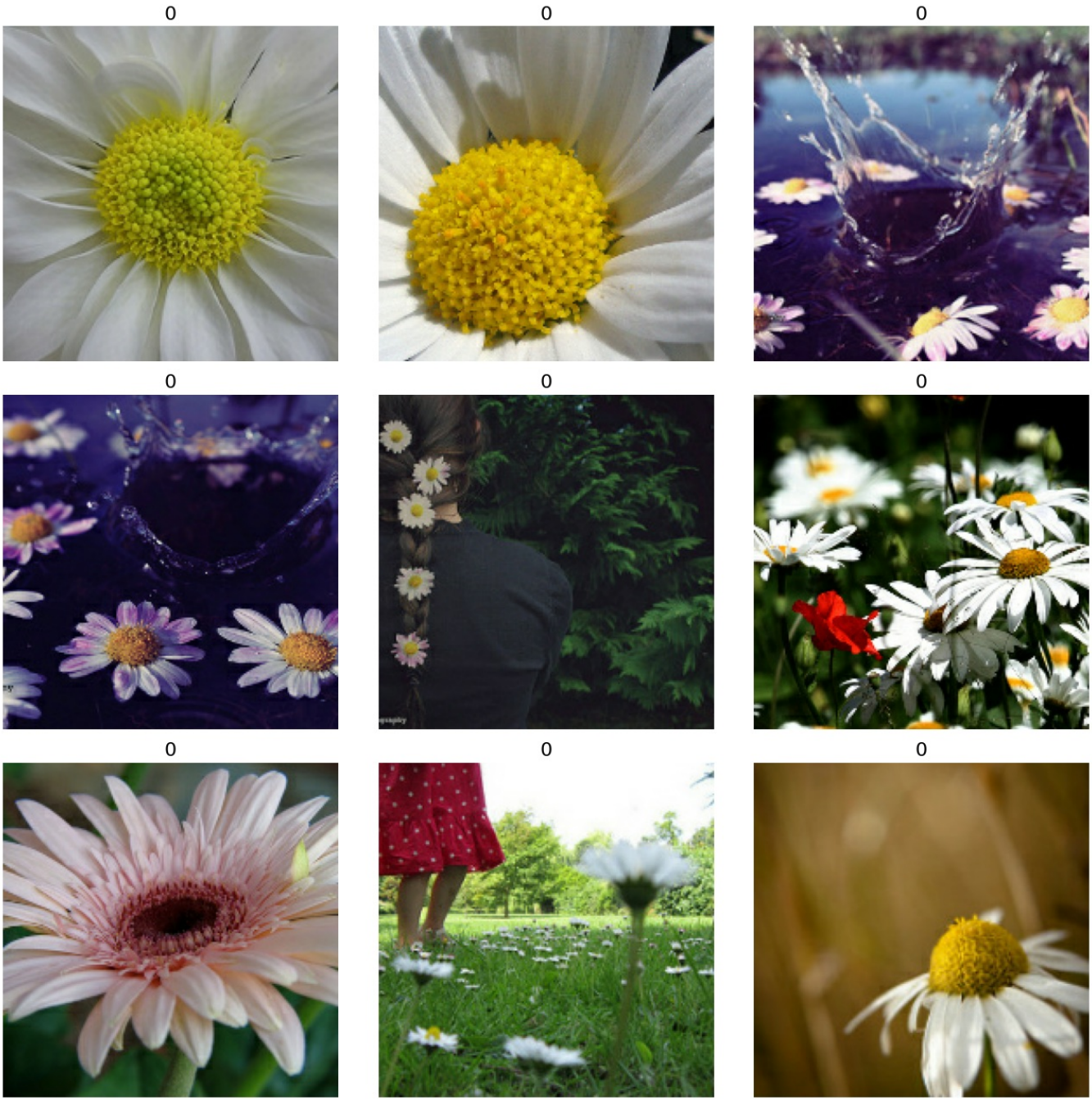
8\xec\xc9\x81\xb4\x01\x07\xfb\x19\xde\xff\x00\xb1\xed\xcd\xd1\xb3%Z\x9burU\xe82\x84`*\xea\x9a\xda\x1c\xe0A!m\xa
4\x1cy\x18\x07\xcfY\xd5\x0b\xe1aR\x19{\x99\x8d\x17\x13\x86 \x18\x94\x85\xf0\xb9\xb5t\xe7\x98\x89X\xafT\x8b\xed\
xb4\x1aT\x86$\x16\xb1\x81\x8f\x03\xcfd\x93\xdf\xbe\xb4\xa69h.E\x88sg\xed\xf5\xbd\xb74\xef\xf4\xf3kk\xef\xae\xa7
R.\x96*\x93f\x04\xfc\xaeGa#\x1d\x13\xf7\x1a.!E$\x02\xc4\xfb\x98J\x03z\xef\x96\xd9R\'\xb5M\xdd\x17\xa2q\xa2>\xf4
\x85\xb39JO\xce4x\x16\x81\xe5\x82V\x82\x15\xfb\x8f\xd3K\x07*1!\xa3\t\xc7\x90u\xdcLXT{\xf2\xe1\xb7\xa4\xd6\xef\x
cal\xea\xacXR\x19\xfc\x16C\x1c\xc2bq_\xf1\x12R3\x9eIQ\xca\xfc\x924\xb0\xaevv!6@\x1a\x86\xa3&\xe8\xde;l\\\x8a\x8
e\xf5:\x03ra\xd4\x81\x8f"F\x1dBXJ\x12\x0f\xa9\xc9$\x80\x90\x92\xa0?\xea:\xf3\xe5\x1c\xe0\xa6*@Le\xedESa~"-\xe5:
\xd2SJ\xbc \xc7\x13\xa2\xd6h\xed\x06J\x9bC\x84\x14\xa5\x04\x06\xdf\x18\xef\x18I\xc1\xebL\xc61\xe4\xd7\xcc[\x87\
xc7\xd7QY\xbb;m\x7f\xd37\xca\x05!-7*\x1dJ\x9c\xe3\x11$\x87\x01\x8b=\x0b\xe6\x14\xa4c\x04+\x91NS\x80A \x11\xe3@\
xe9\x90e\x02\x1a\xb60\x86\xa0\xd5\x9d~n\xdd\xbfK\xaf\xfc=\x15*\x8e\xfa\xa48\xfb\x8d\xa9*nH\x92\xdbg\x01*\xc8\n\
n\x04\x14\x8f\xe6\xe0\x7f\xe6\xc6\x92\xad\x95A\xc5\xd11\xc5Q\x8f\xa9\xdf\xc4\t\xbb66\xe9\xb9\xac\xaav\xe5\xed\x
cd\xe7\n\xafL\x92\xf6+\xe6\x9d,z\xb4\xc9)WA\xf6\t\n\xe2\xa2\xa0R\xe7iQP\x1d\x13\xa4\x94\x0c,C\r\xc7Fp\xa1\xeef\
xf4\xd3\xdb\xff\x00K,*\xb9\x8a\x89\x12\x0b\xaf9\x15\x92T\xea\x94\x8e \x1e\xb3\xde2\x06}\x8e\xb7\x9eP\xb4\x9d\xc
d\xe3\x8c\x1bh\'j\xd7\xaa4j\xf3\xcd^\xf5\x07jR\xfd,\xc6+\x82\xa4\xe4\x9f\xe4\x05C\xa3\x83\xcb^\xc4H{\xc8n{ \xd7
\xb6:\\\xb6\xd4\xed\x04\xc3\x81A\xaa\xd4\x16G\x06\x89\x92\xdb`\xb8\xa1\xe0$\x9f\xab\x8e;\x03\xc6G\xdf]\x12\tY\'
%\xfb\xc4E\xb1p\xbdA\xab\xa8J\x86\xae*x\xa1n\xa1y\xe2\x01\xc1\xfe\xfa\xe3\xa9\xdd\x99\xd2\x7fv\xa0\xcd\xcdET;\x
9eg\xe1M9\xe9%\xc2\xea\x0b\x9fP\x089 ~\xbet\xd2\xa0\x1dE\x0e\xb7+WS\x97Q|\xb4\xd2x\x020V\xfa\x89\r\x8f|\x01\xef
\xa0j:\x84\xa2W=M\x99DqS\x13\xeb\xb8\xa0T\x19\'\xca\xfa\xcfX\xecg\xef\xa4\x11B\x1d\x93\xdc\x99L\x97X~*ioS\x16\n
\x9dZ\x99d\xb9\x84\x97W\xc0\x14\x95\xfb\x03\xc5)\xef\xc1>{\xd2\xb1\xb1\xc4\xadf\xcck\x9fP\x82\x05K\xda\x1d\x1a\
xa3\xb8\xa8\x83@\xa4C\x0eH~\xa6\xb8\xf4\xe2\x92\x82W\x84\x8c\xe5_\xcd\xe0w\xfai\xd8\xfc\x9cl\x80\x1e\xccC\xe3ek
\x1dC+"\xc7\xb2-:+\xb5\x9b\xb2\x8c\xfdRr$\x14\xb4\xdb\xb2\x0bl\x0ft\x85%\'+\xcf\x1fb3\x91\xd8\xd7C\x1a\x05\x16\
xc3q\x0eX\xfe\xd3\x1b[-\x7f\\\xf6\xe3*\xa6\xe51R\x86\xc2`A\x86\xb5\x0e\x1eF\x00\x04\x921\xf79\xd3\xb1\x82\x82\x
93Qy\x15;h\xf5\xb4\xea\xf7\xae\xe6P$[P\xee\x9a\x9bQZAMZb\x01m\x86\x92\xae\x87\x04\x8c\x15/9\x04\x1f\x1a\xacs+@\
x999U\raD\xfeV!\xee\x95\x99\xeaB\xa5U\xe7TXq\xf6\x9a\x8e+\x19\xe2R<+\xae\x87\xe9\xac\xac\xc3\xe6\x15c;\xfb}\xa6
\x84\xb3w\x9a\xd5\xb5,H\xb0+6\xb3\xd1\'\xb6\xdf\xa4\xf2\xd3\xd8qC\xb0A>F\xa8V\xd6\xfb\x93\x15\xe4\xda0^\xc7\xa7
X\xf5\xbb\xf2Ue\x88\x92T\x99j\xe4PTx%D\xe4\x9f\xd7A|\x8d\xc3\xa6\xe3:^\xb6U*\x89W[\xb3\xe02\xe4\x17\xfe\xa4;\xd
1)W\xdb\xc6\x96\xeac\x95\x95\x87\xe6*wR-.\xbbu\'\xf0\xd6J\x13\x1a? \xeb\x07\xdcx\xf1\x8d)\xa8\x9dB\x07P\x0e\xde
\xb3\xad\x9a\xa4\n\x94k\xd6|\xf8\xce\xb8\xfa\xdcb9!-8\x95y\xc8\'\xfd\xf4*\x89\xca\xdaiw\x03]IWE>\xeej\x8f\x12\x
da\xdb]\xc9\x8d\x11\x88C\x98\xa6\xca*\xe1#\xfe\x92\xb4\xfd\x87\xdch\x98\xb8P\x10\xea`\xa0m\x96\xe2f\xee\x91\x0e
\xafr"\xa7zV\x05\r\xb9\xe9z-js-!\xd5\xb6p\x12\x95\xa59\x1ex\xe3\'\x1dgR\xd5\x8fv\xa3h\xf1\xf6\xee\x1e\xedD*\x1e
\xd7\\\x14}\xd1\xb5\xf7Z\x896,U\x98\xd2\x98\x98\xea\xa2:\xbc\xb7\xf4\x85 \x85\'\xfeo\n\xefO\xc7\x8dq\x8b\r\x16\
xec\xcd\xede\xa8\xc5\xde\xba\xd3\x17=\xbb\x1e\xeb\xa7W\x11\x02\x86\xe4\x84\xbc\x87\x1a\x8eL\xaas\x8a\xe4K\xad!$
\x15\x04\x92\x15\xf4\x9c\x14\x9e\xfc\xe9\x99O%\xd1\xd4V2C~G\xfbL\xef\xba[\x8f{1Q\x85\xb8\xbb\xb5C\x13\xaa\ni\x1
1bWiR\xd2\xca\xfdX\xc1?W \x08P!hp\xa5I\xe5\xc99\xcay\x9dH\xee\xd7\xc9\xf7)UZ\xe2\xbf3_\xfc\x08\xd6\xfe\x1b~,v\x
81FV\xd9Re\\\x94h\xaa\xa5\xdc\x8cS\xd8)~tG\x0f\xd2\xf2\xd2\t[\x9c\x82PrJ\x88XV\nF3\xc8|\x8d\xeb\x12:\x96\xa2\x8
fN\xa2\x83y\xbe\x16\xab;#~\xd4\x19\xb0\xe5\xa9\xf7*\xec<\xfd\x1d\x1c\xf2\xeb\x90\x1e\xe5\x84\x85\x14$\xa5\xd6\x
c8\xf4\xd4rK\x84\x92\x021\xf5up\xb3\xbe\x1ekD\xc8\\p\xcb\xc0\xfe\xd9\x9fc\xae\xe2\xdc;\xc64ZuB\xdd\xa5\xc3h\xb8
\x10\xaa\x94wW\x1d\xb5\x94\x80\x96\x9bi9R\x94z\xc6U\x8e\xfb\xe8h\xb1\xabdk$\t\xae8\xa5\x99\xde\xe2\xa8\xdd\xb4:
\xdc\xaa\x1dN\xfa}\xaf\xc3\xdcO\xcdR\xa1\xb4}7\x1dJ\xf2RZ\x01 \xe0\x8c\xe5^3\xd09\xe9\x84\x90h\x98*\x96,\x89\x0
6\xef\xb4\x98\x94\xfa\xbeI(e\x01\x90\xa5 \x1e%\n\x03\xb1\x8f\x7fn\xf5\x0b2\x06\xeaW\xc5\xce\xcc\x0f\xbd\xabT\xf
6\x18j\x89NC\x8eHe\xaf\xe2\xb8\xa3\xf9\xbe\xda>B\xaa\x07\x13\xdd\xc5\xf0\x90\xd4\xc9\xdf.\xd2\x12\xd2\xdet\xa5\
x0f\xfa\xbc\x00$\xe3\xea\xfd<\x7f}L\xc0\x9d\xc6\xea]"\x99Zv\xe2\x89C\xad\xbe\xea\x13\x1b\x02cL\xb7\xf9\x1b\x07%
|\xbe\xc5\'\xf3x\xebH\xccH\xa0cPr\x9a\x13c\xf6\xc3a*\xd1*\x97\xd6\xe8mE\xd8\xed\x8d\x0e\x12e\xc1\x9bo#2d\x1f]\x
b4\x14\xba\xa5yl\x9ceIH\x19\xf1\x8c\xe9\x012\xf3\xa4\xec\xf7\x0cq)\xb3U)\xf7*\xef\x9a\xab\xe6m\xbf\xb7\xb6\xcc:
](\xcd\xe4\xdcv\xd9\t\x92\xd8\x03\xeamN\xe4\xac\x800\x08\xce2\x0f\xe9\x8e\xae\x1c\x0c\x94\xcd\xd8\x91>`\xe2\xc1
\xd4\xae2#\x8aZ\xaa\t\x93\x15\xc0\x8f\xe38\x94y8\xc7d\xf5\x9f=j\x90E\x13\x16(\t\xd2\xd2\xdc+\xe9u\xc1\x16\xd2[\
ti`\x00\xe2R\x02\x92=\xd5\x93\xe0\xeb\x15\xdc\x19\x8e1\xd5\xb0\x9aN\xda\xdf\nV\xd7X\x85\x98u\x96j\x157\xdb\x00\
x8c\xfd)Q\x1d\x95}\xce\xad\x19\xb8,A\xc7\xea0\xfbJ\xd7~!/K\xce\x98)\xd3\xea\xad\x8fEa~\xb2\xbe\x9cc\xc0\x1a_\xa
e\xc4\xf7\x180\xe3A`A}\xd2\xdfz\xe5Q\x1f%\x1e\xebq\xf5\xb0\xd7^\x99\xe9*\xc6493X\xd4\xd4\xc6\xa2\xcdFn\xcb\xfcN
@\xb4\xb6\xca$J\xc7\xf1%\x92\x7f\x8a}\xfa\xf1\x9d3\xd6\xacp\x0e\x1eM\xa8t\xc6\xfdB\xdc\x08M\xd2\xa0\xc5\xf5T\xe
1\xfa\x8b\x87\xaf\xf7\xd3U\xc3\xcc\xf4\xf8\x99f\xce\xd4\xd1f\xb0\xeb\xb0\xe5\x96\xdf}\x92\x93\xf5\xfeR}\xb5\xbc
T\x9dAr{\x89\xbb\x8bo\xea\xb6\xd3U\ne\xe9\xea:\xf7\xa6\xb1\x0e_|x\x9f\x03\xfd\xb4\x86B\x01\xb8a\xfd\xa3\x8cGRk\
xf7\xc5\x9bS]2\xa7%\xd6VVLw\xd7\xdaV\x92z\xd4\xdc\x9c\x18\xefknA\xdc\n\x8d\xc5w\x1f\xf3\x1df\x89\x14Ga_+.Kl\x05
6\xebg\xb0V\x07\xbe}\xf5\xbc\xb9w\x04\x00\x8d_yS.\xc0\xbbi\xd6\xfcz\x85\xbdn\xb7\xf8[\xab\xc9y\xa6\xd4[q\xd1\x9
c`\x9f\x03\x04\xeb\xcc\x8cE\x8e\xa7\x94\xa8\xd7\xcc<\xda]\xcc\x87o[j\xa6]44K\x87\xea\xf1\xc3\xae\x1f]\x87P<\xb7
\x93\x8e \x7f)\xf3\xe3\xafvc\xca\x15)\xc6\xa0\xb2\x02\xde\xde\xe7\xbd\xcc\xbev\xe3r6\xf1V\xcf\xe1\xabf"\x9aqK\x
89\x17\x8b,\xa5\xf0\x9e\x01\xfe \x1c\xaf\xc1\xeb\xc6u\xec\xaf\x87"\xd0\xea\n\xe3|mw\xb8\x89\xf8Z\xde\xcd\xc3\xf
8R\xdf\x08;\x95L\x84\xfc\x88\xedH\x11\xe5\xc3m\xe5\xb4\x9a\x8b|\x80Sa]s\x04\x0e\xbe\xc4\xa4\xeb\x83\x94:\x9d\x8
9\xd3B\x18P3\xe9L\xbf\x8d?\x85\xff\x00\x8ax-XP\xe7\xc1v\xb7&\x9b%\xc8\t\xb8b\x853\x0eC\xd8ml\xa5j\x1fC\xca\t\xc
1I\x04e(P$\xf0\x1a\x7f\x87\xe4\xae7\x00\xf5\x03\xc8\xf1\xd9\xb1r\x1f\x11\x0fv\xfc\x18T\'\xb9&\xa7\xb6,S\xa1\xc4
\xa5 \x1b\x86%ji\r\xaf\x91 \xa9+\xc0(\x07\x89\xc0\xc0\x07\x07\x18\xc65\xdd\xf4\x10\x9b\xc7\xfc\xear\xbdb4\xd1w+
\xe1\x8a\xb0\xe5\xe1:\x99:\x8c\xe5\xb5I\x88\xca\xbd*\x8dEIr,\x87\x00%^\x9a\x92\xaf\n\t\xfaU\xef\x9c~\xba\x11\x8
7\x9b\x1b\xd7\xf3\x8c9\xbd\xa2\xb7\x01w*\x04\xc8\x95\x7f\xc6\x9b\xa9/\xd6+)q$\xe1*A\xfb\x7f\xf7\xc6\xb9\xb91\x1
5iz\xb6\xa4\xb9\xdbW`\xeee\x94\xc5\xebhU%~!\x11A\x15\xd6\x10\x84\x92\x92N\x10\xa4\x7f\xd3\xf7\xd6\xb9C\x8c7\xcc
\x04W\xe7\xc6\xbb\x82\xb3\xfe\x0ekw\x12\x83T\x9a\xb2\x18\x92\xa6\x96\xa7[u9J\x88#\x18\xfd\xc6\x94\xa8r\xae\xa3\
n\xf04a\xf6\xdf\xed\x9d\n\xdcb5\x17w\xae\x06(\xff\x00\x86\xb4\xc2eK\x95\x11\xf7\x95PCk\xe4\xa8\xe8\xe0\x8e\'\x9
0\xcap\xb5$`\xfb\xe82"\xb0\xa3\xd8\x85\x8c\xb2\x9b\x1dMW\xf1-\xf1\x89\xb0\xa9\xdb\x99{e\xf0\xf7\xb5\xca\xb6\xc5
m\x96\x15R\xaeJJG\xccG\x18R[i(\x18JRz#\xc0\xc6:\xd2\xb1\xb2.@\xcbw\x1a\xd8\x8f\x02\xaez\x99\xc7t(\x96\x85\x87A\
x8dwP\xa5\xfe!Q\xa9E^T\x83\x90J\xc7\xd4\xae\xbaJ\xbc\x8f\xbe\xbbY\x02\xa2\x03\xf7\x9c\xacl\xcc\xdcOB(\xe94+\xae
\xfb\x94\xd4\x18q\xdeu(O\x14\xb6\xcbg\xea\x03\xbf\xab\xdb\xff\x00\xf3H\\E\xa5\x05\x95E\x98\x7f\xb5\x9bM]\xaf\\\
xec\xd1#\xc7u\x94\x971=\xe6NT\x84\xfb\x9f\xed\xaa\x13\x15\x9a\x8a|\x80%\x98\xe2\xdf\x8d\x89\xdb\x9b&\xd8j\xabaT
\x97!\xbar@\x9a\xfa\xdd\xc9$\xf9\x1f\xdfL\xcb\x8d\x15lE`\xcb\x91\x8f\xfa\x91[l\xcb\xb3\xae\xb8\xd3\xdf\x99VS&4b
\x19i+\xc7%}\xf52\x94k\xa9A,\x00\x02W\xd8\t\xb4bJz5\xc6\xb5\x16W\xda\x96\xb3\xff\x00}j\x80\xa7s\xd9\x03\x93bL\x
bb\xb7\xaflW@\x91iPp\xca\x9bW\xf0\x9d\xe6;:\x06\xf2\x16\xaaj\xe2\xc8\x1c9\x92\xb6sv\xa5%q\xe2\xb6\xfa\x80K\x9fR
\xc9\xf3\xac\xc7\x97\x8e\xa1\x1cA\xb74\xb3\xdb\x9d\xe9Z\xc9\x95\nX\x0e\xa59\xc8W~5_"V$\xad\x1a\x82_\xeb!\xaf\xd
0\x1f\xa4\\\xf1U%eD4\xa4\x80U\x8f\xef\xa1\xe6\xc5j\x0b \x07\xdad8[qB\xdc\x98\xaai\xaak\x8c<\xd2\x7f\xf0\xea}?\x
9b\xf4\x19\xf3\xa6\xaa\xab\r\xc0b\xc8"\xa7q6\xe3s6\xfeH\xa3\nC\x86,\x99\t<\x1eI\x08q\x00\xfb\x1f\xfd4\x86\xc6\x
d8\xcf\xe21r#\xee\x18\xcb\xbemxv\xbbv\xcc\xc1\xf8*\xd4\xe8ndy=\xb7\xc4\xe3\xea@\xce4\xe0\xca\x13\x89\x8a\nKs]\x
c1\x8b\xd6\x95jS-\x01\x1aC\r\xbf\x1a\x9c\xe2\xa4@\x9f\x1cd\xba\xa5\x0f\xa8,x9\xfb\x1d\x06E\x01a nV;\xfbE\xcd\x9
1\xbbv\xc5\xa7^\x89K\xaf\xdb+\x93\x00\xbc}\x06%2\n\x14O`\x9f\xb0\xfa\xbc\x0f\xb6\x92\xae\x8a\xf4z\x8cl|\x85\x88
\xc6\xdcjM\xb1\xba6R\xac\x9b[n\xdbm\x97\x9d\x1f#\xf8\xab\xc1\xbf\xc3\x16\xbcam)\x00\x95\x8c\xfb\x12|cMu\\\xcb\x
c5V-\x03b\xa6\xbf\xf6\x93!|)\xdc\xb6\x95V=\x83\xbe\x94x\x91\xdd\xac\xc4\xe3I\xbe)\xcf\x93\xf2\xcf\xa9!hD\x84\x1
3\xf5\xa7\x92\x13\x8eD`\xa5XV{\x1c\x0f3\xc1\xcb\xe3\x1b\xecN\xb7\x8b\xe5\xe3\xce,h\x88\xd7\x8dpom\xb7kU\xb6\xde
\xef\x9c\x99\xd5\x84DK\r7n\xcd-"\xa4\x86\xf9$\t\r\xba\x83\xc8\xa5\x03\t\xec\x8e\xf3\xd1\xc9\xd5~\x17\xd4Y@\xc2\
xff\x00=Dy>\x18?\xea\xa7\xf5\x99\xfe\xec\xf8\x89\xa8\xc8\xb6\xcd\xaf\x16$\xd8s2\x1bt\x89\x8bq\x95\xb4\x12\xa4qq
\xa2N@8%8\xeb*\x18\xc0\xc6\xbaO\x9d\xb8\xd4\x8cb\xde\xe7\xff\xd9'>,
  <tf.Tensor: shape=(), dtype=string, numpy=b'daisy'>)]

1b) Testing (3%)

1b) Testing (5%)

i) Read from the TFRecord Dataset, using `load_dataset` and `display_9_images_from_dataset` to test.

In [ ]:
```
#GCS_OUTPUT = 'gs://flowers-public/tfrecords-jpeg-192x192-2/' # We should remove this line to use our own files
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers' #These are the files that we generated above.

def read_tfrecord(example):
    features = {
        "image": tf.io.FixedLenFeature([], tf.string),  # tf.string = bytestring (not text string)
        "class": tf.io.FixedLenFeature([], tf.int64) #,   # shape [] means scalar
    }
    # decode the TFRecord
    example = tf.io.parse_single_example(example, features)
    image = tf.image.decode_jpeg(example['image'], channels=3)
    image = tf.reshape(image, [*TARGET_SIZE, 3])
    class_num = example['class']
    return image, class_num

def load_dataset(filenames):
    # read from TFRecords. For optimal performance, read from multiple
    # TFRecord files at once and set the option experimental_deterministic = False
    # to allow order-altering optimizations.
    option_no_order = tf.data.Options()
    option_no_order.experimental_deterministic = False

    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.with_options(option_no_order)
    dataset = dataset.map(read_tfrecord)
    return dataset


filenames = tf.io.gfile.glob(GCS_OUTPUT + "*.tfrec")
DatasetDec = load_dataset(filenames)
display_9_images_from_dataset(DatasetDec)
```



ii) Write your code above into a file using the *cell magic* `%%writefile spark_write_tfrec.py` at the beginning of the file. Then, run the file locally in Spark.

In [ ]:

```
%%writefile spark_write_tfrec.py
#We first import the necessary libraries
import os
os.environ['PROTOCOL_BUFFERS_PYTHON_IMPLEMENTATION'] = 'python'
import os, sys, math
import numpy as np
#import scipy as sp
#import scipy.stats
import time
import string
import datetime
import random
from matplotlib import pyplot as plt
#import matplotlib.pyplot as plt
import tensorflow as tf
print("Tensorflow version " + tf.__version__)
import pickle
import pyspark
from pyspark.sql import SQLContext
from pyspark.sql import Row

#We define the variables that will be needed for this project.
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' #glob pattern for input files.
PROJECT = 'daring-healer-421511'
BUCKET = 'gs://{}-storage'.format(PROJECT) # This is the bucket storage.
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # We prefix for output file names.
PARTITIONS = 16 # This is the number of partitions that will be used later.
TARGET_SIZE = [192, 192]
```

```python
#Task 1a) - i) Copy over the mapping functions and adapt the resizing and recompression functions to Spark (onl
def decode_jpeg_and_label(filepath):
    #This function reads and decodes the JPEG image from the file.
    bits = tf.io.read_file(filepath)
    image = tf.image.decode_jpeg(bits) #We decode the read data as JPEG images.
    label = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/') #We extract the label (flower name) of
    label2 = label.values[-2]
    return image, label2

def resize_and_crop_image(data): #We specify one argument
    image, label = data #The input is basically a tuple containing the image data and the label.
    #We determine the dimensions of the input images and the target dimensions.
    w = tf.shape(image)[0]
    h = tf.shape(image)[1]
    tw = TARGET_SIZE[1] #Target Width
    th = TARGET_SIZE[0] #Target Height
    #Calculate resize criterion to maintain the aspect ratio.
    resize_crit = (w * th) / (h * tw)
    #We resize the images to fill the target dimensions but also preserving the aspect ratio.
    image = tf.cond(resize_crit < 1,
                    lambda: tf.image.resize(image, [w*tw/w, h*tw/w]), # if true
                    lambda: tf.image.resize(image, [w*th/h, h*th/h])  # if false
                    )
    nw = tf.shape(image)[0]
    nh = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (nw - tw) // 2, (nh - th) // 2, tw, th)
    return (image, label)

def recompress_image(data):
    image, label = data #The input is basically a tuple containing the image data and the label.
    #This function is used to reduce the amount of data, but takes some time
    image = tf.cast(image, tf.uint8) #The image is transformed to a uint8 so the images can be compressed to JP
    image = tf.image.encode_jpeg(image, optimize_size=True, chroma_downsampling=False) #We compress the images
    return (image, label)

#Task 1a) - ii)  Replace the TensorFlow Dataset objects with RDDs, starting with an RDD that contains the list
from pyspark.sql import SparkSession
sc = pyspark.SparkContext.getOrCreate() #Create or get the existing Spark context
spark = SparkSession.builder.getOrCreate() #Initialize the Spark session

image_paths = tf.io.gfile.glob(GCS_PATTERN) #We retrieve the list of image filenames
image_rdd = sc.parallelize(image_paths) #We create an RDD from the list of image filenames

#Task 1a) - iii) Sample the RDD to a smaller number at an appropriate position in the code. Specify a sampling
rdd1_sample = image_rdd.sample(False, 0.02) #We first sample the RDD with a sampling factor of 0.02
rdd2_decode_jpeg_and_label = image_rdd.map(decode_jpeg_and_label) #We decode the jpeg and label of the images o
rdd3_resize_and_crop_image = rdd2_decode_jpeg_and_label.map(resize_and_crop_image) #We resize and crop the imag
rdd4_recompress_image = rdd3_resize_and_crop_image.map(recompress_image) #We recompress the cropped images of t

#Task 1a) - iv) Then use the functions from above to write the TFRecord files.
#This function is used to create a TensorFlow feature for storing byte strings. This is for the image data.
def _bytestring_feature(list_of_bytestrings):
    return tf.train.Feature(bytes_list=tf.train.BytesList(value=list_of_bytestrings))
#This function is used for storing integer values. This is for the labels.
def _int_feature(list_of_ints):
    return tf.train.Feature(int64_list=tf.train.Int64List(value=list_of_ints))
#This function is used to create a TFRecord from image bytes and a label.
def to_tfrecord(tfrec_filewriter, img_bytes, label): # Create tf data records
    class_num = np.argmax(np.array(CLASSES)==label) # 'roses' => 2 (order defined in CLASSES)
    one_hot_class = np.eye(len(CLASSES))[class_num]     #[0, 0, 1, 0, 0] for class #2, roses
    feature = {
        "image": _bytestring_feature([img_bytes]), # one image in the list
        "class": _int_feature([class_num]) #,      # one class in the list
    }
    return tf.train.Example(features=tf.train.Features(feature=feature))

print("Writing TFRecords")
#This function is used to write TFRecords, which will be applied to partitions of an RDD.
def write_tfrecords(partition_index,partition):
  filename = GCS_OUTPUT + "{}.tfrec".format(partition_index) #We define the filename for this TFRecord file bas
  #We Open a TFRecordWriter object for the filename.
  with tf.io.TFRecordWriter(filename) as out_file:
    for element in partition:
      image=element[0]
      label=element[1]
      example = to_tfrecord(out_file,
                            image.numpy(), # The re-compressed image is already a byte string
                            label.numpy()
                            )
      out_file.write(example.SerializeToString()) #We convert the image and label to TFRecord format and write
  return [filename] #We return the filename of the written TFRecord.

#Task 1a) - v)  The code for writing to the TFRecord files needs to be put into a function, that can be applied
#should return the filename, so that you have a list of the created TFRecord files.
rdd5_partitions = rdd4_recompress_image.repartition(PARTITIONS) #We repartition the RDD to have the number of p
rdd1_filenames = rdd4_recompress_image.mapPartitionsWithIndex(write_tfrecords) #We specify an index for each pa

#Task 1b) i) Read from the TFRecord Dataset, using load_dataset and display_9_images_from_dataset to test.
```

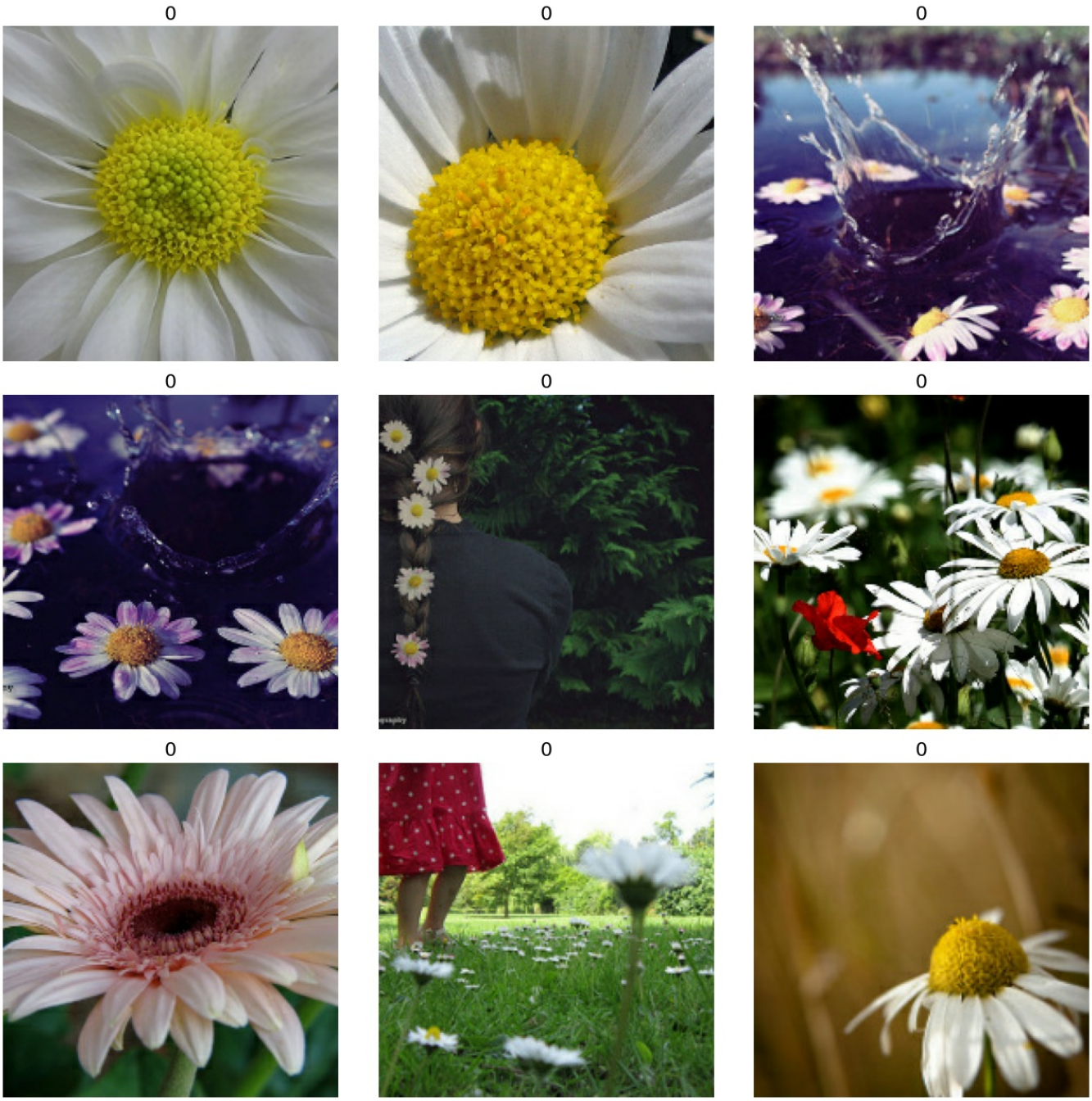
```
#GCS_OUTPUT = 'gs://flowers-public/tfrecords-jpeg-192x192-2/' # We should remove this line to use our own files
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers' #These are the files that we generated above.

def read_tfrecord(example):
    features = {
        "image": tf.io.FixedLenFeature([], tf.string),  # tf.string = bytestring (not text string)
        "class": tf.io.FixedLenFeature([], tf.int64) #,   # shape [] means scalar
    }
    # decode the TFRecord
    example = tf.io.parse_single_example(example, features)
    image = tf.image.decode_jpeg(example['image'], channels=3)
    image = tf.reshape(image, [*TARGET_SIZE, 3])
    class_num = example['class']
    return image, class_num

def load_dataset(filenames):
    # read from TFRecords. For optimal performance, read from multiple
    # TFRecord files at once and set the option experimental_deterministic = False
    # to allow order-altering optimizations.
    option_no_order = tf.data.Options()
    option_no_order.experimental_deterministic = False

    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.with_options(option_no_order)
    dataset = dataset.map(read_tfrecord)
    return dataset

def display_9_images_from_dataset(dataset):
  plt.figure(figsize=(13,13))
  subplot=331
  for i, (image, label) in enumerate(dataset):
    plt.subplot(subplot)
    plt.axis('off')
    plt.imshow(image.numpy().astype(np.uint8))
    plt.title(str(label.numpy()), fontsize=16)
    # plt.title(label.numpy().decode(), fontsize=16)
    subplot += 1
    if i==8:
      break
  plt.tight_layout()
  plt.subplots_adjust(wspace=0.1, hspace=0.1)
  plt.show()


filenames = tf.io.gfile.glob(GCS_OUTPUT + "*.tfrec")
DatasetDec = load_dataset(filenames)
display_9_images_from_dataset(DatasetDec)
```

```
Writing spark_write_tfrec.py
```

In [ ]:

```
%run spark_write_tfrec.py #We run the script locally on spark
```

```
Tensorflow version 2.15.0
Writing TFRecords
```




```
<Figure size 640x480 with 0 Axes>
```

1c) Set up a cluster and run the script. (6%)

Following the example from the labs, set up a cluster to run PySpark jobs in the cloud. You need to set up so that TensorFlow is installed on all nodes in the cluster.

i) Single machine cluster

Set up a cluster with a single machine using the maximal SSD size (100) and 8 vCPUs.

Enable **package installation** by passing a flag `--initialization-actions` with argument `gs://goog-dataproc-initialization-actions-$REGION/python/pip-install.sh` (this is a public script that will read metadata to determine which packages to install). Then, the **packages are specified** by providing a `--metadata` flag with the argument `PIP_PACKAGES=tensorflow==2.4.0`.

Note: consider using `PIP_PACKAGES="tensorflow numpy"` or `PIP_PACKAGES=tensorflow` in case an older version of tensorflow is causing issues.

When the cluster is running, run your script to check that it works and keep the output cell output. (3%)

In [ ]:
```
#In this code we create a cluster in google cloud with a single machine using the maximal SSD size (100) and 8
CLUSTER = '{}-cluster'.format(PROJECT)
REGION = 'us-west1'
PIP_PACKAGES = 'tensorflow==2.4.0'
!gcloud dataproc clusters create $CLUSTER \
    --region $REGION \
    --bucket $PROJECT-storage \
```

```
    --image-version 1.5-ubuntu18 --single-node \
    --master-machine-type n1-standard-8 \
    --master-boot-disk-type pd-ssd --master-boot-disk-size 100\
    --initialization-actions gs://goog-dataproc-initialization-actions-$REGION/python/pip-install.sh \
    --metadata PIP_PACKAGES='tensorflow==2.4.0 matplotlib'

#Reference link for code: https://cloud.google.com/dataproc/docs/guides/create-cluster
```

```
Waiting on operation [projects/daring-healer-421511/regions/us-west1/operations/8c9777f5-feba-342a-b726-4932076
4b1b3].

WARNING: Don't create production clusters that reference initialization actions located in the gs://goog-datapr
oc-initialization-actions-REGION public buckets. These scripts are provided as reference implementations, and t
hey are synchronized with ongoing GitHub repository changes—a new version of a initialization action in public
buckets may break your cluster creation. Instead, copy the following initialization actions from public buckets
into your bucket : gs://goog-dataproc-initialization-actions-us-west1/python/pip-install.sh
WARNING: Failed to validate permissions required for default service account: '869525608453-compute@developer.g
serviceaccount.com'. Cluster creation could still be successful if required permissions have been granted to th
e respective service accounts as mentioned in the document https://cloud.google.com/dataproc/docs/concepts/conf
iguring-clusters/service-accounts#dataproc_service_accounts_2. This could be due to Cloud Resource Manager API
hasn't been enabled in your project '869525608453' before or it is disabled. Enable it by visiting 'https://con
sole.developers.google.com/apis/api/cloudresourcemanager.googleapis.com/overview?project=869525608453'.
WARNING: The firewall rules for specified network or subnetwork would allow ingress traffic from 0.0.0.0/0, whi
ch could be a security risk.
WARNING: The specified custom staging bucket 'daring-healer-421511-storage' is not using uniform bucket level a
ccess IAM configuration. It is recommended to update bucket to enable the same. See https://cloud.google.com/st
orage/docs/uniform-bucket-level-access.
Created [https://dataproc.googleapis.com/v1/projects/daring-healer-421511/regions/us-west1/clusters/daring-heal
er-421511-cluster] Cluster placed in zone [us-west1-c].
```

In [ ]:
```
!gcloud dataproc clusters describe $CLUSTER --region $REGION #This is the main information of the single machin
```

```
clusterName: daring-healer-421511-cluster
clusterUuid: bbfc6d78-16be-44aa-b3ee-b7f62780dc7a
config:
  configBucket: daring-healer-421511-storage
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow==2.4.0 matplotlib
    networkUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/global/networks/default
    serviceAccountScopes:
    - https://www.googleapis.com/auth/bigquery
    - https://www.googleapis.com/auth/bigtable.admin.table
    - https://www.googleapis.com/auth/bigtable.data
    - https://www.googleapis.com/auth/cloud.useraccounts.readonly
    - https://www.googleapis.com/auth/devstorage.full_control
    - https://www.googleapis.com/auth/devstorage.read_write
    - https://www.googleapis.com/auth/logging.write
    - https://www.googleapis.com/auth/monitoring.write
    zoneUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/zones/us-west1-c
  initializationActions:
  - executableFile: gs://goog-dataproc-initialization-actions-us-west1/python/pip-install.sh
    executionTimeout: 600s
  masterConfig:
    diskConfig:
      bootDiskSizeGb: 100
      bootDiskType: pd-ssd
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-5-ubu18-20
230909-165100-rc01
    instanceNames:
    - daring-healer-421511-cluster-m
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/zones/us-west1-c/machin
eTypes/n1-standard-8
    minCpuPlatform: AUTOMATIC
    numInstances: 1
    preemptibility: NON_PREEMPTIBLE
  softwareConfig:
    imageVersion: 1.5.90-ubuntu18
    properties:
      capacity-scheduler:yarn.scheduler.capacity.root.default.ordering-policy: fair
      core:fs.gs.block.size: '134217728'
      core:fs.gs.metadata.cache.enable: 'false'
      core:hadoop.ssl.enabled.protocols: TLSv1,TLSv1.1,TLSv1.2
      dataproc:dataproc.allow.zero.workers: 'true'
      distcp:mapreduce.map.java.opts: -Xmx768m
      distcp:mapreduce.map.memory.mb: '1024'
      distcp:mapreduce.reduce.java.opts: -Xmx768m
      distcp:mapreduce.reduce.memory.mb: '1024'
      hdfs:dfs.datanode.address: 0.0.0.0:9866
      hdfs:dfs.datanode.http.address: 0.0.0.0:9864
      hdfs:dfs.datanode.https.address: 0.0.0.0:9865
      hdfs:dfs.datanode.ipc.address: 0.0.0.0:9867
      hdfs:dfs.namenode.handler.count: '20'
      hdfs:dfs.namenode.http-address: 0.0.0.0:9870
      hdfs:dfs.namenode.https-address: 0.0.0.0:9871
      hdfs:dfs.namenode.lifeline.rpc-address: daring-healer-421511-cluster-m:8050
      hdfs:dfs.namenode.secondary.http-address: 0.0.0.0:9868
```

```
      hdfs:dfs.namenode.secondary.https-address: 0.0.0.0:9869
      hdfs:dfs.namenode.service.handler.count: '10'
      hdfs:dfs.namenode.servicerpc-address: daring-healer-421511-cluster-m:8051
      hive:hive.fetch.task.conversion: none
      mapred-env:HADOOP_JOB_HISTORYSERVER_HEAPSIZE: '4000'
      mapred:mapreduce.job.maps: '21'
      mapred:mapreduce.job.reduce.slowstart.completedmaps: '0.95'
      mapred:mapreduce.job.reduces: '7'
      mapred:mapreduce.jobhistory.recovery.store.class: org.apache.hadoop.mapreduce.v2.hs.HistoryServerLeveldbS
tateStoreService
      mapred:mapreduce.map.cpu.vcores: '1'
      mapred:mapreduce.map.java.opts: -Xmx2457m
      mapred:mapreduce.map.memory.mb: '3072'
      mapred:mapreduce.reduce.cpu.vcores: '1'
      mapred:mapreduce.reduce.java.opts: -Xmx2457m
      mapred:mapreduce.reduce.memory.mb: '3072'
      mapred:mapreduce.task.io.sort.mb: '256'
      mapred:yarn.app.mapreduce.am.command-opts: -Xmx2457m
      mapred:yarn.app.mapreduce.am.resource.cpu-vcores: '1'
      mapred:yarn.app.mapreduce.am.resource.mb: '3072'
      spark-env:SPARK_DAEMON_MEMORY: 4000m
      spark:spark.driver.maxResultSize: 3840m
      spark:spark.driver.memory: 7680m
      spark:spark.executor.cores: '4'
      spark:spark.executor.instances: '2'
      spark:spark.executor.memory: 11171m
      spark:spark.executorEnv.OPENBLAS_NUM_THREADS: '1'
      spark:spark.extraListeners: com.google.cloud.spark.performance.DataprocMetricsListener
      spark:spark.scheduler.mode: FAIR
      spark:spark.sql.cbo.enabled: 'true'
      spark:spark.ui.port: '0'
      spark:spark.yarn.am.memory: 640m
      yarn-env:YARN_NODEMANAGER_HEAPSIZE: '4000'
      yarn-env:YARN_RESOURCEMANAGER_HEAPSIZE: '4000'
      yarn-env:YARN_TIMELINESERVER_HEAPSIZE: '4000'
      yarn:yarn.nodemanager.address: 0.0.0.0:8026
      yarn:yarn.nodemanager.resource.cpu-vcores: '8'
      yarn:yarn.nodemanager.resource.memory-mb: '24576'
      yarn:yarn.resourcemanager.nodemanager-graceful-decommission-timeout-secs: '86400'
      yarn:yarn.scheduler.maximum-allocation-mb: '24576'
      yarn:yarn.scheduler.minimum-allocation-mb: '1024'
  tempBucket: dataproc-temp-us-west1-869525608453-ejfnpe5x
labels:
  goog-dataproc-autozone: enabled
  goog-dataproc-cluster-name: daring-healer-421511-cluster
  goog-dataproc-cluster-uuid: bbfc6d78-16be-44aa-b3ee-b7f62780dc7a
  goog-dataproc-location: us-west1
metrics:
  hdfsMetrics:
    dfs-blocks-corrupt: '0'
    dfs-blocks-default-replication-factor: '1'
    dfs-blocks-missing: '0'
    dfs-blocks-missing-repl-one: '0'
    dfs-blocks-pending-deletion: '0'
    dfs-blocks-under-replication: '0'
    dfs-capacity-present: '92799987712'
    dfs-capacity-remaining: '92799963136'
    dfs-capacity-total: '103865303040'
    dfs-capacity-used: '24576'
    dfs-nodes-decommissioned: '0'
    dfs-nodes-decommissioning: '0'
    dfs-nodes-running: '1'
  yarnMetrics:
    yarn-apps-completed: '0'
    yarn-apps-failed: '0'
    yarn-apps-killed: '0'
    yarn-apps-pending: '0'
    yarn-apps-running: '0'
    yarn-apps-submitted: '0'
    yarn-containers-allocated: '0'
    yarn-containers-pending: '0'
    yarn-containers-reserved: '0'
    yarn-memory-mb-allocated: '0'
    yarn-memory-mb-available: '24576'
    yarn-memory-mb-pending: '0'
    yarn-memory-mb-reserved: '0'
    yarn-memory-mb-total: '24576'
    yarn-nodes-active: '1'
    yarn-nodes-decommissioned: '0'
    yarn-nodes-decommissioning: '0'
    yarn-nodes-lost: '0'
    yarn-nodes-new: '0'
    yarn-nodes-rebooted: '0'
    yarn-nodes-shutdown: '0'
    yarn-nodes-unhealthy: '0'
    yarn-vcores-allocated: '0'
    yarn-vcores-available: '8'
    yarn-vcores-pending: '0'
    yarn-vcores-reserved: '0'
```

```
    yarn-vcores-total: '8'
projectId: daring-healer-421511
status:
  state: RUNNING
  stateStartTime: '2024-05-03T14:43:08.195104Z'
statusHistory:
- state: CREATING
  stateStartTime: '2024-05-03T14:40:09.247203Z'
```

Run the script in the cloud and test the output.

In [ ]:
```
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER --region $REGION ./spark_write_tfrec.py
%time
#We submit the python script ('spark_write_tfrec.py') which we created above as a pyspark job to the cluster th
```

```
Job [1c9eb9587aaf4ea483df1b2951b88af1] submitted.
Waiting for job output...
2024-05-03 10:23:50.231590: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dyna
mic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or di
rectory; LD_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-03 10:23:50.231632: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if
you do not have a GPU set up on your machine.
Tensorflow version 2.4.0
24/05/03 10:23:54 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
24/05/03 10:23:54 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
24/05/03 10:23:54 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
24/05/03 10:23:54 INFO org.spark_project.jetty.util.log: Logging initialized @11232ms to org.spark_project.jett
y.util.log.Slf4jLog
24/05/03 10:23:54 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-SNAPSHOT; built: unknown; git: unknow
n; jvm 1.8.0_382-b05
24/05/03 10:23:54 INFO org.spark_project.jetty.server.Server: Started @11358ms
24/05/03 10:23:54 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@380d866e{HTTP/
1.1, (http/1.1)}{0.0.0.0:33907}
24/05/03 10:23:55 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at daring-healer-42
1511-cluster-m/10.138.0.24:8032
24/05/03 10:23:56 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at dari
ng-healer-421511-cluster-m/10.138.0.24:10200
24/05/03 10:23:56 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
24/05/03 10:23:56 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.
24/05/03 10:23:56 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = memory
-mb, units = Mi, type = COUNTABLE
24/05/03 10:23:56 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = vcores
, units = , type = COUNTABLE
24/05/03 10:23:59 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application
_1714731723280_0001
Writing TFRecords
2024-05-03 10:24:09.241539: I tensorflow/compiler/jit/xla_cpu_device.cc:41] Not creating XLA devices, tf_xla_en
able_xla_devices not set
2024-05-03 10:24:09.241879: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dyna
mic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared object file: No such file or directory; L
D_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-03 10:24:09.241909: W tensorflow/stream_executor/cuda/cuda_driver.cc:326] failed call to cuInit: UNKNOW
N ERROR (303)
2024-05-03 10:24:09.241944: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel driver does not a
ppear to be running on this host (daring-healer-421511-cluster-m): /proc/driver/nvidia/version does not exist
2024-05-03 10:24:09.243069: I tensorflow/compiler/jit/xla_gpu_device.cc:99] Not creating XLA devices, tf_xla_en
able_xla_devices not set
2024-05-03 10:24:09.385526: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR op
timization passes are enabled (registered 2)
2024-05-03 10:24:09.386110: I tensorflow/core/platform/profile_utils/cpu_utils.cc:112] CPU Frequency: 220017000
0 Hz
24/05/03 10:24:11 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@380d866e{HTTP/1.1, (http
/1.1)}{0.0.0.0:0}
Job [1c9eb9587aaf4ea483df1b2951b88af1] finished successfully.
done: true
driverControlFilesUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/c1a54c08-bd2b-46c7-82c7
-f2b3332be0a3/jobs/1c9eb9587aaf4ea483df1b2951b88af1/
driverOutputResourceUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/c1a54c08-bd2b-46c7-82
c7-f2b3332be0a3/jobs/1c9eb9587aaf4ea483df1b2951b88af1/driveroutput
jobUuid: ad1d1035-1936-3399-a2bc-9e50ece128a0
placement:
  clusterName: daring-healer-421511-cluster
  clusterUuid: c1a54c08-bd2b-46c7-82c7-f2b3332be0a3
pysparkJob:
  mainPythonFileUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/c1a54c08-bd2b-46c7-82c7-f
2b3332be0a3/jobs/1c9eb9587aaf4ea483df1b2951b88af1/staging/spark_write_tfrec.py
reference:
  jobId: 1c9eb9587aaf4ea483df1b2951b88af1
  projectId: daring-healer-421511
status:
  state: DONE
  stateStartTime: '2024-05-03T10:24:15.184231Z'
statusHistory:
- state: PENDING
  stateStartTime: '2024-05-03T10:23:41.595317Z'
- state: SETUP_DONE
  stateStartTime: '2024-05-03T10:23:41.631552Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2024-05-03T10:23:41.904189Z'
yarnApplications:
- name: spark_write_tfrec.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://daring-healer-421511-cluster-m:8088/proxy/application_1714731723280_0001/
CPU times: user 4 µs, sys: 1 µs, total: 5 µs
Wall time: 9.78 µs
```

In [ ]:

```
CLUSTER = '{}-cluster'.format(PROJECT)
REGION = 'us-west1'
!gcloud dataproc clusters delete $CLUSTER --region $REGION
#Due to the fact that we have quota related problems as we seem to have few quota available, after creating a c
```

```
The cluster 'daring-healer-421511-cluster' and all attached disks will be
deleted.

Do you want to continue (Y/n)?  Y

Waiting on operation [projects/daring-healer-421511/regions/us-west1/operations/db4513c4-e753-3ed7-8ea1-54f2c19
0d8db].
Deleted [https://dataproc.googleapis.com/v1/projects/daring-healer-421511/regions/us-west1/clusters/daring-heal
er-421511-cluster].
```

In the free credit tier on Google Cloud, there are normally the following **restrictions** on compute machines:

- max 100GB of *SSD persistent disk*
- max 2000GB of *standard persistent disk*
- max 8 *vCPU*s
- no GPUs

See here for details The **disks are virtual** disks, where **I/O speed is limited in proportion to the size**, so we should allocate them evenly. This has mainly an effect on the **time the cluster needs to start**, as we are reading the data mainly from the bucket and we are not writing much to disk at all.

ii) Maximal cluster

Use the **largest possible cluster** within these constraints, i.e. **1 master and 7 worker nodes**. Each of them with 1 (virtual) CPU. The master should get the full *SSD* capacity and the 7 worker nodes should get equal shares of the *standard* disk capacity to maximise throughput.

Once the cluster is running, test your script. (3%)

In [ ]:
```
#In this code cell we we create a cluster in google cloud with the maximal SSD size (100) and 7 worker nodes wi
CLUSTER = '{}-maximalcluster7'.format(PROJECT)
REGION = 'us-west1'
PIP_PACKAGES = 'tensorflow==2.4.0 matplotlib'

!gcloud dataproc clusters create $CLUSTER \
    --bucket $PROJECT-storage \
    --image-version 1.5-ubuntu18 \
    --master-machine-type n1-standard-1 \
    --master-boot-disk-type pd-ssd --master-boot-disk-size 100 \
    --num-workers 7 --worker-machine-type n1-standard-1 --worker-boot-disk-size 100 \
    --initialization-actions gs://goog-dataproc-initialization-actions-$REGION/python/pip-install.sh \
    --metadata PIP_PACKAGES='tensorflow==2.4.0 matplotlib'
#Reference link for code: https://cloud.google.com/dataproc/docs/guides/create-cluster
```

```
Waiting on operation [projects/daring-healer-421511/regions/us-west1/operations/417b8a18-0c87-3066-8688-14120e5
e2674].

WARNING: Creating clusters using the n1-standard-1 machine type is not recommended. Consider using a machine ty
pe with higher memory.
WARNING: Don't create production clusters that reference initialization actions located in the gs://goog-datapr
oc-initialization-actions-REGION public buckets. These scripts are provided as reference implementations, and t
hey are synchronized with ongoing GitHub repository changes—a new version of a initialization action in public
buckets may break your cluster creation. Instead, copy the following initialization actions from public buckets
into your bucket : gs://goog-dataproc-initialization-actions-us-west1/python/pip-install.sh
WARNING: Failed to validate permissions required for default service account: '869525608453-compute@developer.g
serviceaccount.com'. Cluster creation could still be successful if required permissions have been granted to th
e respective service accounts as mentioned in the document https://cloud.google.com/dataproc/docs/concepts/conf
iguring-clusters/service-accounts#dataproc_service_accounts_2. This could be due to Cloud Resource Manager API
hasn't been enabled in your project '869525608453' before or it is disabled. Enable it by visiting 'https://con
sole.developers.google.com/apis/api/cloudresourcemanager.googleapis.com/overview?project=869525608453'.
WARNING: For PD-Standard without local SSDs, we strongly recommend provisioning 1TB or larger to ensure consist
ently high I/O performance. See https://cloud.google.com/compute/docs/disks/performance for information on disk
I/O performance.
WARNING: The firewall rules for specified network or subnetwork would allow ingress traffic from 0.0.0.0/0, whi
ch could be a security risk.
WARNING: The specified custom staging bucket 'daring-healer-421511-storage' is not using uniform bucket level a
ccess IAM configuration. It is recommended to update bucket to enable the same. See https://cloud.google.com/st
orage/docs/uniform-bucket-level-access.
Created [https://dataproc.googleapis.com/v1/projects/daring-healer-421511/regions/us-west1/clusters/daring-heal
er-421511-maximalcluster7] Cluster placed in zone [us-west1-c].
```

In [ ]:
```
!gcloud dataproc clusters describe $CLUSTER --region $REGION #This is the main information of the maximal machi
```

```
clusterName: daring-healer-421511-maximalcluster7
clusterUuid: 3774a390-9e84-4997-9102-8e033a20bf5d
config:
  configBucket: daring-healer-421511-storage
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow==2.4.0 matplotlib
    networkUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/global/networks/default
    serviceAccountScopes:
```

```
    - https://www.googleapis.com/auth/bigquery
    - https://www.googleapis.com/auth/bigtable.admin.table
    - https://www.googleapis.com/auth/bigtable.data
    - https://www.googleapis.com/auth/cloud.useraccounts.readonly
    - https://www.googleapis.com/auth/devstorage.full_control
    - https://www.googleapis.com/auth/devstorage.read_write
    - https://www.googleapis.com/auth/logging.write
    - https://www.googleapis.com/auth/monitoring.write
    zoneUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/zones/us-west1-c
  initializationActions:
  - executableFile: gs://goog-dataproc-initialization-actions-us-west1/python/pip-install.sh
    executionTimeout: 600s
  masterConfig:
    diskConfig:
      bootDiskSizeGb: 100
      bootDiskType: pd-ssd
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-5-ubu18-20
230909-165100-rc01
    instanceNames:
    - daring-healer-421511-maximalcluster7-m
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/zones/us-west1-c/machin
eTypes/n1-standard-1
    minCpuPlatform: AUTOMATIC
    numInstances: 1
    preemptibility: NON_PREEMPTIBLE
  softwareConfig:
    imageVersion: 1.5.90-ubuntu18
    properties:
      capacity-scheduler:yarn.scheduler.capacity.root.default.ordering-policy: fair
      core:fs.gs.block.size: '134217728'
      core:fs.gs.metadata.cache.enable: 'false'
      core:hadoop.ssl.enabled.protocols: TLSv1,TLSv1.1,TLSv1.2
      distcp:mapreduce.map.java.opts: -Xmx576m
      distcp:mapreduce.map.memory.mb: '768'
      distcp:mapreduce.reduce.java.opts: -Xmx576m
      distcp:mapreduce.reduce.memory.mb: '768'
      hdfs:dfs.datanode.address: 0.0.0.0:9866
      hdfs:dfs.datanode.http.address: 0.0.0.0:9864
      hdfs:dfs.datanode.https.address: 0.0.0.0:9865
      hdfs:dfs.datanode.ipc.address: 0.0.0.0:9867
      hdfs:dfs.namenode.handler.count: '60'
      hdfs:dfs.namenode.http-address: 0.0.0.0:9870
      hdfs:dfs.namenode.https-address: 0.0.0.0:9871
      hdfs:dfs.namenode.lifeline.rpc-address: daring-healer-421511-maximalcluster7-m:8050
      hdfs:dfs.namenode.secondary.http-address: 0.0.0.0:9868
      hdfs:dfs.namenode.secondary.https-address: 0.0.0.0:9869
      hdfs:dfs.namenode.service.handler.count: '30'
      hdfs:dfs.namenode.servicerpc-address: daring-healer-421511-maximalcluster7-m:8051
      hive:hive.fetch.task.conversion: none
      mapred-env:HADOOP_JOB_HISTORYSERVER_HEAPSIZE: '1000'
      mapred:mapreduce.job.maps: '60'
      mapred:mapreduce.job.reduce.slowstart.completedmaps: '0.95'
      mapred:mapreduce.job.reduces: '7'
      mapred:mapreduce.jobhistory.recovery.store.class: org.apache.hadoop.mapreduce.v2.hs.HistoryServerLeveldbS
tateStoreService
      mapred:mapreduce.map.cpu.vcores: '1'
      mapred:mapreduce.map.java.opts: -Xmx819m
      mapred:mapreduce.map.memory.mb: '1024'
      mapred:mapreduce.reduce.cpu.vcores: '1'
      mapred:mapreduce.reduce.java.opts: -Xmx1638m
      mapred:mapreduce.reduce.memory.mb: '2048'
      mapred:mapreduce.task.io.sort.mb: '256'
      mapred:yarn.app.mapreduce.am.command-opts: -Xmx819m
      mapred:yarn.app.mapreduce.am.resource.cpu-vcores: '1'
      mapred:yarn.app.mapreduce.am.resource.mb: '1024'
      spark-env:SPARK_DAEMON_MEMORY: 1000m
      spark:spark.driver.maxResultSize: 480m
      spark:spark.driver.memory: 960m
      spark:spark.executor.cores: '1'
      spark:spark.executor.instances: '2'
      spark:spark.executor.memory: 2688m
      spark:spark.executorEnv.OPENBLAS_NUM_THREADS: '1'
      spark:spark.extraListeners: com.google.cloud.spark.performance.DataprocMetricsListener
      spark:spark.scheduler.mode: FAIR
      spark:spark.sql.cbo.enabled: 'true'
      spark:spark.ui.port: '0'
      spark:spark.yarn.am.memory: 640m
      yarn-env:YARN_NODEMANAGER_HEAPSIZE: '1000'
      yarn-env:YARN_RESOURCEMANAGER_HEAPSIZE: '1000'
      yarn-env:YARN_TIMELINESERVER_HEAPSIZE: '1000'
      yarn:yarn.nodemanager.address: 0.0.0.0:8026
      yarn:yarn.nodemanager.resource.cpu-vcores: '1'
      yarn:yarn.nodemanager.resource.memory-mb: '3072'
      yarn:yarn.resourcemanager.nodemanager-graceful-decommission-timeout-secs: '86400'
      yarn:yarn.scheduler.maximum-allocation-mb: '3072'
      yarn:yarn.scheduler.minimum-allocation-mb: '256'
  tempBucket: dataproc-temp-us-west1-869525608453-ejfnpe5x
  workerConfig:
    diskConfig:
```

```
      bootDiskSizeGb: 100
      bootDiskType: pd-standard
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-5-ubu18-20
230909-165100-rc01
    instanceNames:
    - daring-healer-421511-maximalcluster7-w-0
    - daring-healer-421511-maximalcluster7-w-1
    - daring-healer-421511-maximalcluster7-w-2
    - daring-healer-421511-maximalcluster7-w-3
    - daring-healer-421511-maximalcluster7-w-4
    - daring-healer-421511-maximalcluster7-w-5
    - daring-healer-421511-maximalcluster7-w-6
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/zones/us-west1-c/machin
eTypes/n1-standard-1
    minCpuPlatform: AUTOMATIC
    numInstances: 7
    preemptibility: NON_PREEMPTIBLE
labels:
  goog-dataproc-autozone: enabled
  goog-dataproc-cluster-name: daring-healer-421511-maximalcluster7
  goog-dataproc-cluster-uuid: 3774a390-9e84-4997-9102-8e033a20bf5d
  goog-dataproc-location: us-west1
projectId: daring-healer-421511
status:
  state: RUNNING
  stateStartTime: '2024-05-03T10:37:45.555871Z'
statusHistory:
- state: CREATING
  stateStartTime: '2024-05-03T10:33:06.890157Z'
```

In [ ]:

```
#We run the script in the cloud to test the output of the script
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER --region $REGION ./spark_write_tfrec.py
%time
#We submit the python script ('spark_write_tfrec.py') which we created above as a pyspark job to the cluster th
```

```
Job [eccf04a36ba14d34b51dd455c1b14992] submitted.
Waiting for job output...
2024-05-03 10:38:03.407410: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dyna
mic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or di
rectory; LD_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-03 10:38:03.407568: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if
you do not have a GPU set up on your machine.
Tensorflow version 2.4.0
24/05/03 10:38:08 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
24/05/03 10:38:08 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
24/05/03 10:38:08 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
24/05/03 10:38:08 INFO org.spark_project.jetty.util.log: Logging initialized @13639ms to org.spark_project.jett
y.util.log.Slf4jLog
24/05/03 10:38:08 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-SNAPSHOT; built: unknown; git: unknow
n; jvm 1.8.0_382-b05
24/05/03 10:38:08 INFO org.spark_project.jetty.server.Server: Started @13871ms
24/05/03 10:38:08 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@45915c69{HTTP/
1.1, (http/1.1)}{0.0.0.0:45707}
24/05/03 10:38:10 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at daring-healer-42
1511-maximalcluster7-m/10.138.0.27:8032
24/05/03 10:38:11 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at dari
ng-healer-421511-maximalcluster7-m/10.138.0.27:10200
24/05/03 10:38:11 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
24/05/03 10:38:11 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.
24/05/03 10:38:11 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = memory
-mb, units = Mi, type = COUNTABLE
24/05/03 10:38:11 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = vcores
, units = , type = COUNTABLE
24/05/03 10:38:16 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application
_1714732543370_0001
Writing TFRecords
2024-05-03 10:38:34.435846: I tensorflow/compiler/jit/xla_cpu_device.cc:41] Not creating XLA devices, tf_xla_en
able_xla_devices not set
2024-05-03 10:38:34.438248: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dyna
mic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared object file: No such file or directory; L
D_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-03 10:38:34.438489: W tensorflow/stream_executor/cuda/cuda_driver.cc:326] failed call to cuInit: UNKNOW
N ERROR (303)
2024-05-03 10:38:34.438566: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel driver does not a
ppear to be running on this host (daring-healer-421511-maximalcluster7-m): /proc/driver/nvidia/version does not
exist
2024-05-03 10:38:34.441707: I tensorflow/compiler/jit/xla_gpu_device.cc:99] Not creating XLA devices, tf_xla_en
able_xla_devices not set
2024-05-03 10:38:34.607747: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR op
timization passes are enabled (registered 2)
2024-05-03 10:38:34.611708: I tensorflow/core/platform/profile_utils/cpu_utils.cc:112] CPU Frequency: 219999500
0 Hz
24/05/03 10:38:36 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@45915c69{HTTP/1.1, (http
/1.1)}{0.0.0.0:0}
Job [eccf04a36ba14d34b51dd455c1b14992] finished successfully.
done: true
driverControlFilesUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/3774a390-9e84-4997-9102
-8e033a20bf5d/jobs/eccf04a36ba14d34b51dd455c1b14992/
driverOutputResourceUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/3774a390-9e84-4997-91
02-8e033a20bf5d/jobs/eccf04a36ba14d34b51dd455c1b14992/driveroutput
jobUuid: b5709024-1cf6-3814-91a2-a60770ba687c
placement:
  clusterName: daring-healer-421511-maximalcluster7
  clusterUuid: 3774a390-9e84-4997-9102-8e033a20bf5d
pysparkJob:
  mainPythonFileUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/3774a390-9e84-4997-9102-8
e033a20bf5d/jobs/eccf04a36ba14d34b51dd455c1b14992/staging/spark_write_tfrec.py
reference:
  jobId: eccf04a36ba14d34b51dd455c1b14992
  projectId: daring-healer-421511
status:
  state: DONE
  stateStartTime: '2024-05-03T10:38:37.377671Z'
statusHistory:
- state: PENDING
  stateStartTime: '2024-05-03T10:37:51.316818Z'
- state: SETUP_DONE
  stateStartTime: '2024-05-03T10:37:51.357006Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2024-05-03T10:37:51.729851Z'
yarnApplications:
- name: spark_write_tfrec.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://daring-healer-421511-maximalcluster7-m:8088/proxy/application_1714732543370_0001/
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 7.87 µs
```

In [ ]:

```
CLUSTER = '{}-maximalcluster7'.format(PROJECT)
REGION = 'us-west1'
!gcloud dataproc clusters delete $CLUSTER --region $REGION
#Due to the fact that we have quota related problems as we seem to have few quota available, after creating a c
```

```
The cluster 'daring-healer-421511-maximalcluster3' and all attached disks will
be deleted.

Do you want to continue (Y/n)?  Y

Waiting on operation [projects/daring-healer-421511/regions/us-west1/operations/a60de350-2357-347e-b122-078365b
c2121].
Deleted [https://dataproc.googleapis.com/v1/projects/daring-healer-421511/regions/us-west1/clusters/daring-heal
er-421511-maximalcluster3].
```

1d) Optimisation, experiments, and discussion (17%)

i) Improve parallelisation

If you implemented a straightfoward version, you will **probably** observe that **all the computation** is done on only **two nodes**. This can be adressed by using the **second parameter** in the initial call to **parallelize**. Make the **suitable change** in the code you have written above and mark it up in comments as `### TASK 1d ###`.

Demonstrate the difference in cluster utilisation before and after the change based on different parameter values with **screenshots from Google Cloud** and measure the **difference in the processing time**. (6%)

ii) Experiment with cluster configurations.

In addition to the experiments above (using 8 VMs),test your program with 4 machines with double the resources each (2 vCPUs, memory, disk) and 1 machine with eightfold resources. Discuss the results in terms of disk I/O and network bandwidth allocation in the cloud. (7%)

iii) Explain the difference between this use of Spark and most standard applications like e.g. in our labs in terms of where the data is stored. What kind of parallelisation approach is used here? (4%)

Write the code below and your answers in the report.

In [ ]:
```
#SUBTASK i)
```

In [ ]:
```
#For this task, We first create a cluster with 8 machines with 1 master with 7 workers type n-1 machine using t
CLUSTER = '{}-cluster-task1d-7workers'.format(PROJECT)
REGION = 'us-west1'
PIP_PACKAGES = 'tensorflow==2.4.0'
!gcloud dataproc clusters create $CLUSTER \
    --region $REGION \
    --bucket $PROJECT-storage \
    --image-version 1.5-ubuntu18 \
    --master-machine-type n1-standard-8 \
    --master-boot-disk-type pd-ssd --master-boot-disk-size 100 \
    --num-workers 7 --worker-machine-type n1-standard-1 --worker-boot-disk-size 100 \
    --initialization-actions gs://goog-dataproc-initialization-actions-$REGION/python/pip-install.sh \
    --metadata PIP_PACKAGES='tensorflow==2.4.0 matplotlib'

#This is the cluster that is going to be used to test parallelization.

#Reference link for code: https://cloud.google.com/dataproc/docs/guides/create-cluster
```

```
Waiting on operation [projects/daring-healer-421511/regions/us-west1/operations/01117f6d-e733-35b0-8974-a3d4d76
aa9bb].

WARNING: Creating clusters using the n1-standard-1 machine type is not recommended. Consider using a machine ty
pe with higher memory.
WARNING: Don't create production clusters that reference initialization actions located in the gs://goog-datapr
oc-initialization-actions-REGION public buckets. These scripts are provided as reference implementations, and t
hey are synchronized with ongoing GitHub repository changes—a new version of a initialization action in public
buckets may break your cluster creation. Instead, copy the following initialization actions from public buckets
into your bucket : gs://goog-dataproc-initialization-actions-us-west1/python/pip-install.sh
WARNING: Failed to validate permissions required for default service account: '869525608453-compute@developer.g
serviceaccount.com'. Cluster creation could still be successful if required permissions have been granted to th
e respective service accounts as mentioned in the document https://cloud.google.com/dataproc/docs/concepts/conf
iguring-clusters/service-accounts#dataproc_service_accounts_2. This could be due to Cloud Resource Manager API
hasn't been enabled in your project '869525608453' before or it is disabled. Enable it by visiting 'https://con
sole.developers.google.com/apis/api/cloudresourcemanager.googleapis.com/overview?project=869525608453'.
WARNING: For PD-Standard without local SSDs, we strongly recommend provisioning 1TB or larger to ensure consist
ently high I/O performance. See https://cloud.google.com/compute/docs/disks/performance for information on disk
I/O performance.
WARNING: The firewall rules for specified network or subnetwork would allow ingress traffic from 0.0.0.0/0, whi
ch could be a security risk.
WARNING: The specified custom staging bucket 'daring-healer-421511-storage' is not using uniform bucket level a
ccess IAM configuration. It is recommended to update bucket to enable the same. See https://cloud.google.com/st
orage/docs/uniform-bucket-level-access.
Created [https://dataproc.googleapis.com/v1/projects/daring-healer-421511/regions/us-west1/clusters/daring-heal
er-421511-cluster-task1d-7workers] Cluster placed in zone [us-west1-c].
```

In [ ]:
```
%%writefile spark_write_tfrec_paral.py
#We improve parallelization
#We first import the necessary libraries
import os
```

```
os.environ['PROTOCOL_BUFFERS_PYTHON_IMPLEMENTATION'] = 'python'
import os, sys, math
import numpy as np
#import scipy as sp
#import scipy.stats
import time
import string
import datetime
import random
from matplotlib import pyplot as plt
import tensorflow as tf
print("Tensorflow version " + tf.__version__)
import pickle
import pyspark
from pyspark.sql import SQLContext
from pyspark.sql import Row

#We define the variables that will be needed for this project.
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' #glob pattern for input files.
PROJECT = 'daring-healer-421511'
BUCKET = 'gs://{}-storage'.format(PROJECT) # This is the bucket storage.
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # We prefix for output file names.
PARTITIONS = 16 # This is the number of partitions that will be used later.
TARGET_SIZE = [192, 192]


#Task 1a) - i) Copy over the mapping functions and adapt the resizing and recompression functions to Spark (onl
def decode_jpeg_and_label(filepath):
    #This function reads and decodes the JPEG image from the file.
    bits = tf.io.read_file(filepath)
    image = tf.image.decode_jpeg(bits) #We decode the read data as JPEG images.
    label = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/') #We extract the label (flower name) of
    label2 = label.values[-2]
    return image, label2

def resize_and_crop_image(data): #We specify one argument
    image, label = data #The input is basically a tuple containing the image data and the label.
    #We determine the dimensions of the input images and the target dimensions.
    w = tf.shape(image)[0]
    h = tf.shape(image)[1]
    tw = TARGET_SIZE[1] #Target Width
    th = TARGET_SIZE[0] #Target Height
    #Calculate resize criterion to maintain the aspect ratio.
    resize_crit = (w * th) / (h * tw)
    #We resize the images to fill the target dimensions but also preserving the aspect ratio.
    image = tf.cond(resize_crit < 1,
                    lambda: tf.image.resize(image, [w*tw/w, h*tw/w]), # if true
                    lambda: tf.image.resize(image, [w*th/h, h*th/h])  # if false
                    )
    nw = tf.shape(image)[0]
    nh = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (nw - tw) // 2, (nh - th) // 2, tw, th)
    return (image, label)

def recompress_image(data):
    image, label = data #The input is basically a tuple containing the image data and the label.
    #This function is used to reduce the amount of data, but takes some time
    image = tf.cast(image, tf.uint8) #The image is transformed to a uint8 so the images can be compressed to JP
    image = tf.image.encode_jpeg(image, optimize_size=True, chroma_downsampling=False) #We compress the images
    return (image, label)

#Task 1a) - ii)  Replace the TensorFlow Dataset objects with RDDs, starting with an RDD that contains the list
from pyspark.sql import SparkSession
sc = pyspark.SparkContext.getOrCreate() #Create or get the existing Spark context
spark = SparkSession.builder.getOrCreate() #Initialize the Spark session

image_paths = tf.io.gfile.glob(GCS_PATTERN) #We retrieve the list of image filenames
### TASK 1d ###
image_rdd = sc.parallelize(image_paths, 16) #We create an RDD from the list of image filenames with 16 partitio

#Task 1a) - iii) Sample the RDD to a smaller number at an appropriate position in the code. Specify a sampling
rdd1_sample = image_rdd.sample(False, 0.02) #We first sample the RDD with a sampling factor of 0.02
rdd2_decode_jpeg_and_label = image_rdd.map(decode_jpeg_and_label) #We decode the jpeg and label of the images o
rdd3_resize_and_crop_image = rdd2_decode_jpeg_and_label.map(resize_and_crop_image) #We resize and crop the imag
rdd4_recompress_image = rdd3_resize_and_crop_image.map(recompress_image) #We recompress the cropped images of t

#Task 1a) - iv) Then use the functions from above to write the TFRecord files.
#This function is used to create a TensorFlow feature for storing byte strings. This is for the image data.
def _bytestring_feature(list_of_bytestrings):
    return tf.train.Feature(bytes_list=tf.train.BytesList(value=list_of_bytestrings))
#This function is used for storing integer values. This is for the labels.
def _int_feature(list_of_ints):
    return tf.train.Feature(int64_list=tf.train.Int64List(value=list_of_ints))
#This function is used to create a TFRecord from image bytes and a label.
def to_tfrecord(tfrec_filewriter, img_bytes, label): # Create tf data records
    class_num = np.argmax(np.array(CLASSES)==label) # 'roses' => 2 (order defined in CLASSES)
    one_hot_class = np.eye(len(CLASSES))[class_num]     #[0, 0, 1, 0, 0] for class #2, roses
    feature = {
        "image": _bytestring_feature([img_bytes]), # one image in the list
```

```
        "class": _int_feature([class_num]) #,      # one class in the list
    }
    return tf.train.Example(features=tf.train.Features(feature=feature))

print("Writing TFRecords")
#This function is used to write TFRecords, which will be applied to partitions of an RDD.
def write_tfrecords(partition_index,partition):
  filename = GCS_OUTPUT + "{}.tfrec".format(partition_index) #We define the filename for this TFRecord file bas
  #We Open a TFRecordWriter object for the filename.
  with tf.io.TFRecordWriter(filename) as out_file:
    for element in partition:
      image=element[0]
      label=element[1]
      example = to_tfrecord(out_file,
                            image.numpy(), # The re-compressed image is already a byte string
                            label.numpy()
                            )
      out_file.write(example.SerializeToString()) #We convert the image and label to TFRecord format and write
  return [filename] #We return the filename of the written TFRecord.

#Task 1a) - v)  The code for writing to the TFRecord files needs to be put into a function, that can be applied
#should return the filename, so that you have a list of the created TFRecord files.
rdd5_partitions = rdd4_recompress_image.repartition(PARTITIONS) #We repartition the RDD to have the number of p
rdd1_filenames = rdd4_recompress_image.mapPartitionsWithIndex(write_tfrecords) #We specify an index for each pa

#Task 1b) i) Read from the TFRecord Dataset, using load_dataset and display_9_images_from_dataset to test.

#GCS_OUTPUT = 'gs://flowers-public/tfrecords-jpeg-192x192-2/' # We should remove this line to use our own files
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers' #These are the files that we generated above.

def read_tfrecord(example):
    features = {
        "image": tf.io.FixedLenFeature([], tf.string),  # tf.string = bytestring (not text string)
        "class": tf.io.FixedLenFeature([], tf.int64) #,   # shape [] means scalar
    }
    # decode the TFRecord
    example = tf.io.parse_single_example(example, features)
    image = tf.image.decode_jpeg(example['image'], channels=3)
    image = tf.reshape(image, [*TARGET_SIZE, 3])
    class_num = example['class']
    return image, class_num

def load_dataset(filenames):
    # read from TFRecords. For optimal performance, read from multiple
    # TFRecord files at once and set the option experimental_deterministic = False
    # to allow order-altering optimizations.
    option_no_order = tf.data.Options()
    option_no_order.experimental_deterministic = False

    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.with_options(option_no_order)
    dataset = dataset.map(read_tfrecord)
    return dataset

def display_9_images_from_dataset(dataset):
  plt.figure(figsize=(13,13))
  subplot=331
  for i, (image, label) in enumerate(dataset):
    plt.subplot(subplot)
    plt.axis('off')
    plt.imshow(image.numpy().astype(np.uint8))
    plt.title(str(label.numpy()), fontsize=16)
    # plt.title(label.numpy().decode(), fontsize=16)
    subplot += 1
    if i==8:
      break
  plt.tight_layout()
  plt.subplots_adjust(wspace=0.1, hspace=0.1)
  plt.show()


filenames = tf.io.gfile.glob(GCS_OUTPUT + "*.tfrec")
DatasetDec = load_dataset(filenames)
display_9_images_from_dataset(DatasetDec)
```

```
Writing spark_write_tfrec_paral.py
```

In [ ]:

```
!gcloud dataproc clusters describe $CLUSTER --region $REGION #We get the main information of the cluster
```

```
clusterName: daring-healer-421511-cluster-task1d-7workers
clusterUuid: 4085db72-fcba-4bd1-af18-508d2b7a67be
config:
  configBucket: daring-healer-421511-storage
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow==2.4.0 matplotlib
    networkUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/global/networks/default
```

```
    serviceAccountScopes:
    - https://www.googleapis.com/auth/bigquery
    - https://www.googleapis.com/auth/bigtable.admin.table
    - https://www.googleapis.com/auth/bigtable.data
    - https://www.googleapis.com/auth/cloud.useraccounts.readonly
    - https://www.googleapis.com/auth/devstorage.full_control
    - https://www.googleapis.com/auth/devstorage.read_write
    - https://www.googleapis.com/auth/logging.write
    - https://www.googleapis.com/auth/monitoring.write
    zoneUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/zones/us-west1-c
  initializationActions:
  - executableFile: gs://goog-dataproc-initialization-actions-us-west1/python/pip-install.sh
    executionTimeout: 600s
  masterConfig:
    diskConfig:
      bootDiskSizeGb: 100
      bootDiskType: pd-ssd
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-5-ubu18-20
230909-165100-rc01
    instanceNames:
    - daring-healer-421511-cluster-task1d-7workers-m
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/zones/us-west1-c/machin
eTypes/n1-standard-8
    minCpuPlatform: AUTOMATIC
    numInstances: 1
    preemptibility: NON_PREEMPTIBLE
  softwareConfig:
    imageVersion: 1.5.90-ubuntu18
    properties:
      capacity-scheduler:yarn.scheduler.capacity.root.default.ordering-policy: fair
      core:fs.gs.block.size: '134217728'
      core:fs.gs.metadata.cache.enable: 'false'
      core:hadoop.ssl.enabled.protocols: TLSv1,TLSv1.1,TLSv1.2
      distcp:mapreduce.map.java.opts: -Xmx576m
      distcp:mapreduce.map.memory.mb: '768'
      distcp:mapreduce.reduce.java.opts: -Xmx576m
      distcp:mapreduce.reduce.memory.mb: '768'
      hdfs:dfs.datanode.address: 0.0.0.0:9866
      hdfs:dfs.datanode.http.address: 0.0.0.0:9864
      hdfs:dfs.datanode.https.address: 0.0.0.0:9865
      hdfs:dfs.datanode.ipc.address: 0.0.0.0:9867
      hdfs:dfs.namenode.handler.count: '60'
      hdfs:dfs.namenode.http-address: 0.0.0.0:9870
      hdfs:dfs.namenode.https-address: 0.0.0.0:9871
      hdfs:dfs.namenode.lifeline.rpc-address: daring-healer-421511-cluster-task1d-7workers-m:8050
      hdfs:dfs.namenode.secondary.http-address: 0.0.0.0:9868
      hdfs:dfs.namenode.secondary.https-address: 0.0.0.0:9869
      hdfs:dfs.namenode.service.handler.count: '30'
      hdfs:dfs.namenode.servicerpc-address: daring-healer-421511-cluster-task1d-7workers-m:8051
      hive:hive.fetch.task.conversion: none
      mapred-env:HADOOP_JOB_HISTORYSERVER_HEAPSIZE: '4000'
      mapred:mapreduce.job.maps: '60'
      mapred:mapreduce.job.reduce.slowstart.completedmaps: '0.95'
      mapred:mapreduce.job.reduces: '7'
      mapred:mapreduce.jobhistory.recovery.store.class: org.apache.hadoop.mapreduce.v2.hs.HistoryServerLeveldbS
tateStoreService
      mapred:mapreduce.map.cpu.vcores: '1'
      mapred:mapreduce.map.java.opts: -Xmx819m
      mapred:mapreduce.map.memory.mb: '1024'
      mapred:mapreduce.reduce.cpu.vcores: '1'
      mapred:mapreduce.reduce.java.opts: -Xmx1638m
      mapred:mapreduce.reduce.memory.mb: '2048'
      mapred:mapreduce.task.io.sort.mb: '256'
      mapred:yarn.app.mapreduce.am.command-opts: -Xmx819m
      mapred:yarn.app.mapreduce.am.resource.cpu-vcores: '1'
      mapred:yarn.app.mapreduce.am.resource.mb: '1024'
      spark-env:SPARK_DAEMON_MEMORY: 4000m
      spark:spark.driver.maxResultSize: 3840m
      spark:spark.driver.memory: 7680m
      spark:spark.executor.cores: '1'
      spark:spark.executor.instances: '2'
      spark:spark.executor.memory: 2688m
      spark:spark.executorEnv.OPENBLAS_NUM_THREADS: '1'
      spark:spark.extraListeners: com.google.cloud.spark.performance.DataprocMetricsListener
      spark:spark.scheduler.mode: FAIR
      spark:spark.sql.cbo.enabled: 'true'
      spark:spark.ui.port: '0'
      spark:spark.yarn.am.memory: 640m
      yarn-env:YARN_NODEMANAGER_HEAPSIZE: '1000'
      yarn-env:YARN_RESOURCEMANAGER_HEAPSIZE: '4000'
      yarn-env:YARN_TIMELINESERVER_HEAPSIZE: '4000'
      yarn:yarn.nodemanager.address: 0.0.0.0:8026
      yarn:yarn.nodemanager.resource.cpu-vcores: '1'
      yarn:yarn.nodemanager.resource.memory-mb: '3072'
      yarn:yarn.resourcemanager.nodemanager-graceful-decommission-timeout-secs: '86400'
      yarn:yarn.scheduler.maximum-allocation-mb: '3072'
      yarn:yarn.scheduler.minimum-allocation-mb: '256'
  tempBucket: dataproc-temp-us-west1-869525608453-ejfnpe5x
  workerConfig:
```

```
    diskConfig:
      bootDiskSizeGb: 100
      bootDiskType: pd-standard
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-5-ubu18-20
230909-165100-rc01
    instanceNames:
    - daring-healer-421511-cluster-task1d-7workers-w-0
    - daring-healer-421511-cluster-task1d-7workers-w-1
    - daring-healer-421511-cluster-task1d-7workers-w-2
    - daring-healer-421511-cluster-task1d-7workers-w-3
    - daring-healer-421511-cluster-task1d-7workers-w-4
    - daring-healer-421511-cluster-task1d-7workers-w-5
    - daring-healer-421511-cluster-task1d-7workers-w-6
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/zones/us-west1-c/machin
eTypes/n1-standard-1
    minCpuPlatform: AUTOMATIC
    numInstances: 7
    preemptibility: NON_PREEMPTIBLE
labels:
  goog-dataproc-autozone: enabled
  goog-dataproc-cluster-name: daring-healer-421511-cluster-task1d-7workers
  goog-dataproc-cluster-uuid: 4085db72-fcba-4bd1-af18-508d2b7a67be
  goog-dataproc-location: us-west1
projectId: daring-healer-421511
status:
  state: RUNNING
  stateStartTime: '2024-05-03T10:50:07.233406Z'
statusHistory:
- state: CREATING
  stateStartTime: '2024-05-03T10:46:26.816792Z'
```

In [ ]:

```
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER --region $REGION ./spark_write_tfrec_paral.py
%time

#We submit the python script as a job to our cluster
```

```
Job [e173d9441aff4f42a9bf94c81018d4f4] submitted.
Waiting for job output...
2024-05-03 10:50:26.828368: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dyna
mic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or di
rectory; LD_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-03 10:50:26.828415: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if
you do not have a GPU set up on your machine.
Tensorflow version 2.4.0
24/05/03 10:50:30 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
24/05/03 10:50:30 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
24/05/03 10:50:30 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
24/05/03 10:50:31 INFO org.spark_project.jetty.util.log: Logging initialized @11432ms to org.spark_project.jett
y.util.log.Slf4jLog
24/05/03 10:50:31 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-SNAPSHOT; built: unknown; git: unknow
n; jvm 1.8.0_382-b05
24/05/03 10:50:31 INFO org.spark_project.jetty.server.Server: Started @11576ms
24/05/03 10:50:31 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@600eaa78{HTTP/
1.1, (http/1.1)}{0.0.0.0:43279}
24/05/03 10:50:32 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at daring-healer-42
1511-cluster-task1d-7workers-m/10.138.0.32:8032
24/05/03 10:50:33 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at dari
ng-healer-421511-cluster-task1d-7workers-m/10.138.0.32:10200
24/05/03 10:50:33 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
24/05/03 10:50:33 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.
24/05/03 10:50:33 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = memory
-mb, units = Mi, type = COUNTABLE
24/05/03 10:50:33 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = vcores
, units = , type = COUNTABLE
24/05/03 10:50:37 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application
_1714733279670_0001
Writing TFRecords
2024-05-03 10:50:53.187454: I tensorflow/compiler/jit/xla_cpu_device.cc:41] Not creating XLA devices, tf_xla_en
able_xla_devices not set
2024-05-03 10:50:53.187792: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dyna
mic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared object file: No such file or directory; L
D_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-03 10:50:53.187818: W tensorflow/stream_executor/cuda/cuda_driver.cc:326] failed call to cuInit: UNKNOW
N ERROR (303)
2024-05-03 10:50:53.187847: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel driver does not a
ppear to be running on this host (daring-healer-421511-cluster-task1d-7workers-m): /proc/driver/nvidia/version
does not exist
2024-05-03 10:50:53.188967: I tensorflow/compiler/jit/xla_gpu_device.cc:99] Not creating XLA devices, tf_xla_en
able_xla_devices not set
2024-05-03 10:50:53.307246: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR op
timization passes are enabled (registered 2)
2024-05-03 10:50:53.307686: I tensorflow/core/platform/profile_utils/cpu_utils.cc:112] CPU Frequency: 219999500
0 Hz
24/05/03 10:50:54 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@600eaa78{HTTP/1.1, (http
/1.1)}{0.0.0.0:0}
Job [e173d9441aff4f42a9bf94c81018d4f4] finished successfully.
done: true
driverControlFilesUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/4085db72-fcba-4bd1-af18
-508d2b7a67be/jobs/e173d9441aff4f42a9bf94c81018d4f4/
driverOutputResourceUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/4085db72-fcba-4bd1-af
18-508d2b7a67be/jobs/e173d9441aff4f42a9bf94c81018d4f4/driveroutput
jobUuid: eaffcb97-d15f-3f07-bcc5-21bbfaeca036
placement:
  clusterName: daring-healer-421511-cluster-task1d-7workers
  clusterUuid: 4085db72-fcba-4bd1-af18-508d2b7a67be
pysparkJob:
  mainPythonFileUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/4085db72-fcba-4bd1-af18-5
08d2b7a67be/jobs/e173d9441aff4f42a9bf94c81018d4f4/staging/spark_write_tfrec_paral.py
reference:
  jobId: e173d9441aff4f42a9bf94c81018d4f4
  projectId: daring-healer-421511
status:
  state: DONE
  stateStartTime: '2024-05-03T10:50:59.474905Z'
statusHistory:
- state: PENDING
  stateStartTime: '2024-05-03T10:50:17.217414Z'
- state: SETUP_DONE
  stateStartTime: '2024-05-03T10:50:17.253100Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2024-05-03T10:50:17.548975Z'
yarnApplications:
- name: spark_write_tfrec_paral.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://daring-healer-421511-cluster-task1d-7workers-m:8088/proxy/application_1714733279670_0001/
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 5.72 µs
```

In [ ]:

```
CLUSTER = '{}-cluster-task1d-7workers'.format(PROJECT)
REGION = 'us-west1'
!gcloud dataproc clusters delete $CLUSTER --region $REGION
#Due to the fact that we have quota related problems as we seem to have few quota available, after creating a c
```

```
The cluster 'daring-healer-421511-cluster-task1d' and all attached disks will be
 deleted.

Do you want to continue (Y/n)?  Y

Waiting on operation [projects/daring-healer-421511/regions/us-central1/operations/e28e3fc9-4b65-39c4-bc2e-bcb6
09da03bd].
Deleted [https://dataproc.googleapis.com/v1/projects/daring-healer-421511/regions/us-central1/clusters/daring-h
ealer-421511-cluster-task1d].
```

In [ ]:
```
#SUBTASK ii)
```

In [ ]:
```
#Experiment with different cluster configurations
#In this subtask, we aim to test and compare the performance of the cluster created above with 8 machines with
#type n - 2 and 1 cluster with 1 master and 0 workers type n-8). The disk sizes for all clusters are set the sa

#This cluster has 1 master with 3 workers type n - 2 using the max SSD size of 100.
CLUSTER = '{}-cluster-task1d-3workers'.format(PROJECT)
REGION = 'us-west1'
PIP_PACKAGES = 'tensorflow==2.4.0'
!gcloud dataproc clusters create $CLUSTER \
    --region $REGION \
    --bucket $PROJECT-storage \
    --image-version 1.5-ubuntu18 \
    --master-machine-type n1-standard-8 \
    --master-boot-disk-type pd-ssd --master-boot-disk-size 100 \
    --num-workers 3 --worker-machine-type n1-standard-2 --worker-boot-disk-size 100 \
    --initialization-actions gs://goog-dataproc-initialization-actions-$REGION/python/pip-install.sh \
    --metadata PIP_PACKAGES='tensorflow==2.4.0 matplotlib'
#Reference link for code: https://cloud.google.com/dataproc/docs/guides/create-cluster
```

```
Waiting on operation [projects/daring-healer-421511/regions/us-west1/operations/86da704f-37d6-35f1-aa76-b968fe2
24bde].

WARNING: Don't create production clusters that reference initialization actions located in the gs://goog-datapr
oc-initialization-actions-REGION public buckets. These scripts are provided as reference implementations, and t
hey are synchronized with ongoing GitHub repository changes—a new version of a initialization action in public
buckets may break your cluster creation. Instead, copy the following initialization actions from public buckets
into your bucket : gs://goog-dataproc-initialization-actions-us-west1/python/pip-install.sh
WARNING: Failed to validate permissions required for default service account: '869525608453-compute@developer.g
serviceaccount.com'. Cluster creation could still be successful if required permissions have been granted to th
e respective service accounts as mentioned in the document https://cloud.google.com/dataproc/docs/concepts/conf
iguring-clusters/service-accounts#dataproc_service_accounts_2. This could be due to Cloud Resource Manager API
hasn't been enabled in your project '869525608453' before or it is disabled. Enable it by visiting 'https://con
sole.developers.google.com/apis/api/cloudresourcemanager.googleapis.com/overview?project=869525608453'.
WARNING: For PD-Standard without local SSDs, we strongly recommend provisioning 1TB or larger to ensure consist
ently high I/O performance. See https://cloud.google.com/compute/docs/disks/performance for information on disk
I/O performance.
WARNING: The firewall rules for specified network or subnetwork would allow ingress traffic from 0.0.0.0/0, whi
ch could be a security risk.
WARNING: The specified custom staging bucket 'daring-healer-421511-storage' is not using uniform bucket level a
ccess IAM configuration. It is recommended to update bucket to enable the same. See https://cloud.google.com/st
orage/docs/uniform-bucket-level-access.
Created [https://dataproc.googleapis.com/v1/projects/daring-healer-421511/regions/us-west1/clusters/daring-heal
er-421511-cluster-task1d-3workers] Cluster placed in zone [us-west1-c].
```

In [ ]:
```
#We get information on the cluster
!gcloud dataproc clusters describe $CLUSTER --region $REGION
```

```
clusterName: daring-healer-421511-cluster-task1d-3workers
clusterUuid: c50abefa-bdd7-4df3-8d73-ef0060e0b8f5
config:
  configBucket: daring-healer-421511-storage
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow==2.4.0 matplotlib
    networkUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/global/networks/default
    serviceAccountScopes:
    - https://www.googleapis.com/auth/bigquery
    - https://www.googleapis.com/auth/bigtable.admin.table
    - https://www.googleapis.com/auth/bigtable.data
    - https://www.googleapis.com/auth/cloud.useraccounts.readonly
    - https://www.googleapis.com/auth/devstorage.full_control
    - https://www.googleapis.com/auth/devstorage.read_write
    - https://www.googleapis.com/auth/logging.write
    - https://www.googleapis.com/auth/monitoring.write
    zoneUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/zones/us-west1-c
  initializationActions:
  - executableFile: gs://goog-dataproc-initialization-actions-us-west1/python/pip-install.sh
    executionTimeout: 600s
  masterConfig:
    diskConfig:
      bootDiskSizeGb: 100
      bootDiskType: pd-ssd
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-5-ubu18-20
230909-165100-rc01
    instanceNames:
```

```
    - daring-healer-421511-cluster-task1d-3workers-m
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/zones/us-west1-c/machin
eTypes/n1-standard-8
    minCpuPlatform: AUTOMATIC
    numInstances: 1
    preemptibility: NON_PREEMPTIBLE
  softwareConfig:
    imageVersion: 1.5.90-ubuntu18
    properties:
      capacity-scheduler:yarn.scheduler.capacity.root.default.ordering-policy: fair
      core:fs.gs.block.size: '134217728'
      core:fs.gs.metadata.cache.enable: 'false'
      core:hadoop.ssl.enabled.protocols: TLSv1,TLSv1.1,TLSv1.2
      distcp:mapreduce.map.java.opts: -Xmx576m
      distcp:mapreduce.map.memory.mb: '768'
      distcp:mapreduce.reduce.java.opts: -Xmx576m
      distcp:mapreduce.reduce.memory.mb: '768'
      hdfs:dfs.datanode.address: 0.0.0.0:9866
      hdfs:dfs.datanode.http.address: 0.0.0.0:9864
      hdfs:dfs.datanode.https.address: 0.0.0.0:9865
      hdfs:dfs.datanode.ipc.address: 0.0.0.0:9867
      hdfs:dfs.namenode.handler.count: '40'
      hdfs:dfs.namenode.http-address: 0.0.0.0:9870
      hdfs:dfs.namenode.https-address: 0.0.0.0:9871
      hdfs:dfs.namenode.lifeline.rpc-address: daring-healer-421511-cluster-task1d-3workers-m:8050
      hdfs:dfs.namenode.secondary.http-address: 0.0.0.0:9868
      hdfs:dfs.namenode.secondary.https-address: 0.0.0.0:9869
      hdfs:dfs.namenode.service.handler.count: '20'
      hdfs:dfs.namenode.servicerpc-address: daring-healer-421511-cluster-task1d-3workers-m:8051
      hive:hive.fetch.task.conversion: none
      mapred-env:HADOOP_JOB_HISTORYSERVER_HEAPSIZE: '4000'
      mapred:mapreduce.job.maps: '24'
      mapred:mapreduce.job.reduce.slowstart.completedmaps: '0.95'
      mapred:mapreduce.job.reduces: '8'
      mapred:mapreduce.jobhistory.recovery.store.class: org.apache.hadoop.mapreduce.v2.hs.HistoryServerLeveldbS
tateStoreService
      mapred:mapreduce.map.cpu.vcores: '1'
      mapred:mapreduce.map.java.opts: -Xmx1638m
      mapred:mapreduce.map.memory.mb: '2048'
      mapred:mapreduce.reduce.cpu.vcores: '1'
      mapred:mapreduce.reduce.java.opts: -Xmx1638m
      mapred:mapreduce.reduce.memory.mb: '2048'
      mapred:mapreduce.task.io.sort.mb: '256'
      mapred:yarn.app.mapreduce.am.command-opts: -Xmx1638m
      mapred:yarn.app.mapreduce.am.resource.cpu-vcores: '1'
      mapred:yarn.app.mapreduce.am.resource.mb: '2048'
      spark-env:SPARK_DAEMON_MEMORY: 4000m
      spark:spark.driver.maxResultSize: 3840m
      spark:spark.driver.memory: 7680m
      spark:spark.executor.cores: '1'
      spark:spark.executor.instances: '2'
      spark:spark.executor.memory: 2688m
      spark:spark.executorEnv.OPENBLAS_NUM_THREADS: '1'
      spark:spark.extraListeners: com.google.cloud.spark.performance.DataprocMetricsListener
      spark:spark.scheduler.mode: FAIR
      spark:spark.sql.cbo.enabled: 'true'
      spark:spark.ui.port: '0'
      spark:spark.yarn.am.memory: 640m
      yarn-env:YARN_NODEMANAGER_HEAPSIZE: '1920'
      yarn-env:YARN_RESOURCEMANAGER_HEAPSIZE: '4000'
      yarn-env:YARN_TIMELINESERVER_HEAPSIZE: '4000'
      yarn:yarn.nodemanager.address: 0.0.0.0:8026
      yarn:yarn.nodemanager.resource.cpu-vcores: '2'
      yarn:yarn.nodemanager.resource.memory-mb: '6144'
      yarn:yarn.resourcemanager.nodemanager-graceful-decommission-timeout-secs: '86400'
      yarn:yarn.scheduler.maximum-allocation-mb: '6144'
      yarn:yarn.scheduler.minimum-allocation-mb: '512'
  tempBucket: dataproc-temp-us-west1-869525608453-ejfnpe5x
  workerConfig:
    diskConfig:
      bootDiskSizeGb: 100
      bootDiskType: pd-standard
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-5-ubu18-20
230909-165100-rc01
    instanceNames:
    - daring-healer-421511-cluster-task1d-3workers-w-0
    - daring-healer-421511-cluster-task1d-3workers-w-1
    - daring-healer-421511-cluster-task1d-3workers-w-2
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/zones/us-west1-c/machin
eTypes/n1-standard-2
    minCpuPlatform: AUTOMATIC
    numInstances: 3
    preemptibility: NON_PREEMPTIBLE
labels:
  goog-dataproc-autozone: enabled
  goog-dataproc-cluster-name: daring-healer-421511-cluster-task1d-3workers
  goog-dataproc-cluster-uuid: c50abefa-bdd7-4df3-8d73-ef0060e0b8f5
  goog-dataproc-location: us-west1
projectId: daring-healer-421511
```

```
status:
  state: RUNNING
  stateStartTime: '2024-05-03T11:02:21.732121Z'
statusHistory:
- state: CREATING
  stateStartTime: '2024-05-03T10:58:51.411629Z'
```

In [ ]:

```
#We submit the job to the cluster
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER --region $REGION ./spark_write_tfrec_paral.py
%time
```

```
Job [f380694063b348f1950d63a4568b9a85] submitted.
Waiting for job output...
2024-05-03 11:02:37.277570: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dyna
mic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or di
rectory; LD_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-03 11:02:37.277620: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if
you do not have a GPU set up on your machine.
Tensorflow version 2.4.0
24/05/03 11:02:40 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
24/05/03 11:02:40 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
24/05/03 11:02:40 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
24/05/03 11:02:41 INFO org.spark_project.jetty.util.log: Logging initialized @10563ms to org.spark_project.jett
y.util.log.Slf4jLog
24/05/03 11:02:41 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-SNAPSHOT; built: unknown; git: unknow
n; jvm 1.8.0_382-b05
24/05/03 11:02:41 INFO org.spark_project.jetty.server.Server: Started @10700ms
24/05/03 11:02:41 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@77391f2f{HTTP/
1.1, (http/1.1)}{0.0.0.0:41511}
24/05/03 11:02:42 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at daring-healer-42
1511-cluster-task1d-3workers-m/10.138.0.42:8032
24/05/03 11:02:42 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at dari
ng-healer-421511-cluster-task1d-3workers-m/10.138.0.42:10200
24/05/03 11:02:42 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
24/05/03 11:02:42 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.
24/05/03 11:02:42 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = memory
-mb, units = Mi, type = COUNTABLE
24/05/03 11:02:42 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = vcores
, units = , type = COUNTABLE
24/05/03 11:02:45 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application
_1714734019364_0001
Writing TFRecords
2024-05-03 11:02:58.967230: I tensorflow/compiler/jit/xla_cpu_device.cc:41] Not creating XLA devices, tf_xla_en
able_xla_devices not set
2024-05-03 11:02:58.967545: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dyna
mic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared object file: No such file or directory; L
D_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-03 11:02:58.967577: W tensorflow/stream_executor/cuda/cuda_driver.cc:326] failed call to cuInit: UNKNOW
N ERROR (303)
2024-05-03 11:02:58.967601: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel driver does not a
ppear to be running on this host (daring-healer-421511-cluster-task1d-3workers-m): /proc/driver/nvidia/version
does not exist
2024-05-03 11:02:58.968611: I tensorflow/compiler/jit/xla_gpu_device.cc:99] Not creating XLA devices, tf_xla_en
able_xla_devices not set
2024-05-03 11:02:59.085344: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR op
timization passes are enabled (registered 2)
2024-05-03 11:02:59.085877: I tensorflow/core/platform/profile_utils/cpu_utils.cc:112] CPU Frequency: 219999500
0 Hz
24/05/03 11:03:00 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@77391f2f{HTTP/1.1, (http
/1.1)}{0.0.0.0:0}
Job [f380694063b348f1950d63a4568b9a85] finished successfully.
done: true
driverControlFilesUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/c50abefa-bdd7-4df3-8d73
-ef0060e0b8f5/jobs/f380694063b348f1950d63a4568b9a85/
driverOutputResourceUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/c50abefa-bdd7-4df3-8d
73-ef0060e0b8f5/jobs/f380694063b348f1950d63a4568b9a85/driveroutput
jobUuid: bbf2efb6-571a-3009-9e0b-154bcc19526e
placement:
  clusterName: daring-healer-421511-cluster-task1d-3workers
  clusterUuid: c50abefa-bdd7-4df3-8d73-ef0060e0b8f5
pysparkJob:
  mainPythonFileUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/c50abefa-bdd7-4df3-8d73-e
f0060e0b8f5/jobs/f380694063b348f1950d63a4568b9a85/staging/spark_write_tfrec_paral.py
reference:
  jobId: f380694063b348f1950d63a4568b9a85
  projectId: daring-healer-421511
status:
  state: DONE
  stateStartTime: '2024-05-03T11:03:02.187354Z'
statusHistory:
- state: PENDING
  stateStartTime: '2024-05-03T11:02:29.004432Z'
- state: SETUP_DONE
  stateStartTime: '2024-05-03T11:02:29.053704Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2024-05-03T11:02:29.334748Z'
yarnApplications:
- name: spark_write_tfrec_paral.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://daring-healer-421511-cluster-task1d-3workers-m:8088/proxy/application_1714734019364_0001/
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 8.82 µs
```

In [ ]:

```
CLUSTER = '{}-cluster-task1d-3workers'.format(PROJECT)
REGION = 'us-west1'
!gcloud dataproc clusters delete $CLUSTER --region $REGION
#Due to the fact that we have quota related problems as we seem to have few quota available, after creating a c
```

```
The cluster 'daring-healer-421511-cluster-task1d-3vms' and all attached disks
will be deleted.

Do you want to continue (Y/n)?  Y

Waiting on operation [projects/daring-healer-421511/regions/us-west1/operations/25c44884-0470-3592-884e-a3bb2c8
ba58f].
Deleted [https://dataproc.googleapis.com/v1/projects/daring-healer-421511/regions/us-west1/clusters/daring-heal
er-421511-cluster-task1d-3vms].
```

In [ ]:
```
#This is the cluster with  with 1 master and 0 workers type n-8 using the max SSD size of 100.
CLUSTER = '{}-cluster-task1d-0workers'.format(PROJECT)
REGION = 'us-west1'
PIP_PACKAGES = 'tensorflow==2.4.0'
!gcloud dataproc clusters create $CLUSTER \
    --region $REGION \
    --bucket $PROJECT-storage \
    --image-version 1.5-ubuntu18 \
    --single-node \
    --master-machine-type n1-standard-8 \
    --master-boot-disk-type pd-ssd --master-boot-disk-size 100 \
    --initialization-actions gs://goog-dataproc-initialization-actions-$REGION/python/pip-install.sh \
    --metadata PIP_PACKAGES='tensorflow==2.4.0 matplotlib'
#Reference link for code: https://cloud.google.com/dataproc/docs/guides/create-cluster
```

```
Waiting on operation [projects/daring-healer-421511/regions/us-west1/operations/4d0b5784-02a0-348f-9cb8-6189e67
7ef18].

WARNING: Don't create production clusters that reference initialization actions located in the gs://goog-datapr
oc-initialization-actions-REGION public buckets. These scripts are provided as reference implementations, and t
hey are synchronized with ongoing GitHub repository changes—a new version of a initialization action in public
buckets may break your cluster creation. Instead, copy the following initialization actions from public buckets
into your bucket : gs://goog-dataproc-initialization-actions-us-west1/python/pip-install.sh
WARNING: Failed to validate permissions required for default service account: '869525608453-compute@developer.g
serviceaccount.com'. Cluster creation could still be successful if required permissions have been granted to th
e respective service accounts as mentioned in the document https://cloud.google.com/dataproc/docs/concepts/conf
iguring-clusters/service-accounts#dataproc_service_accounts_2. This could be due to Cloud Resource Manager API
hasn't been enabled in your project '869525608453' before or it is disabled. Enable it by visiting 'https://con
sole.developers.google.com/apis/api/cloudresourcemanager.googleapis.com/overview?project=869525608453'.
WARNING: The firewall rules for specified network or subnetwork would allow ingress traffic from 0.0.0.0/0, whi
ch could be a security risk.
WARNING: The specified custom staging bucket 'daring-healer-421511-storage' is not using uniform bucket level a
ccess IAM configuration. It is recommended to update bucket to enable the same. See https://cloud.google.com/st
orage/docs/uniform-bucket-level-access.
Created [https://dataproc.googleapis.com/v1/projects/daring-healer-421511/regions/us-west1/clusters/daring-heal
er-421511-cluster-task1d-0workers] Cluster placed in zone [us-west1-c].
```

In [ ]:
```
#We get information on the cluster
!gcloud dataproc clusters describe $CLUSTER --region $REGION
```

```
clusterName: daring-healer-421511-cluster-task1d-0workers
clusterUuid: 830a382c-b5c5-4426-834e-622eb4cdd050
config:
  configBucket: daring-healer-421511-storage
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow==2.4.0 matplotlib
    networkUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/global/networks/default
    serviceAccountScopes:
    - https://www.googleapis.com/auth/bigquery
    - https://www.googleapis.com/auth/bigtable.admin.table
    - https://www.googleapis.com/auth/bigtable.data
    - https://www.googleapis.com/auth/cloud.useraccounts.readonly
    - https://www.googleapis.com/auth/devstorage.full_control
    - https://www.googleapis.com/auth/devstorage.read_write
    - https://www.googleapis.com/auth/logging.write
    - https://www.googleapis.com/auth/monitoring.write
    zoneUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/zones/us-west1-c
  initializationActions:
  - executableFile: gs://goog-dataproc-initialization-actions-us-west1/python/pip-install.sh
    executionTimeout: 600s
  masterConfig:
    diskConfig:
      bootDiskSizeGb: 100
      bootDiskType: pd-ssd
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-5-ubu18-20
230909-165100-rc01
    instanceNames:
    - daring-healer-421511-cluster-task1d-0workers-m
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/zones/us-west1-c/machin
eTypes/n1-standard-8
    minCpuPlatform: AUTOMATIC
    numInstances: 1
    preemptibility: NON_PREEMPTIBLE
  softwareConfig:
    imageVersion: 1.5.90-ubuntu18
    properties:
```

```
      capacity-scheduler:yarn.scheduler.capacity.root.default.ordering-policy: fair
      core:fs.gs.block.size: '134217728'
      core:fs.gs.metadata.cache.enable: 'false'
      core:hadoop.ssl.enabled.protocols: TLSv1,TLSv1.1,TLSv1.2
      dataproc:dataproc.allow.zero.workers: 'true'
      distcp:mapreduce.map.java.opts: -Xmx768m
      distcp:mapreduce.map.memory.mb: '1024'
      distcp:mapreduce.reduce.java.opts: -Xmx768m
      distcp:mapreduce.reduce.memory.mb: '1024'
      hdfs:dfs.datanode.address: 0.0.0.0:9866
      hdfs:dfs.datanode.http.address: 0.0.0.0:9864
      hdfs:dfs.datanode.https.address: 0.0.0.0:9865
      hdfs:dfs.datanode.ipc.address: 0.0.0.0:9867
      hdfs:dfs.namenode.handler.count: '20'
      hdfs:dfs.namenode.http-address: 0.0.0.0:9870
      hdfs:dfs.namenode.https-address: 0.0.0.0:9871
      hdfs:dfs.namenode.lifeline.rpc-address: daring-healer-421511-cluster-task1d-0workers-m:8050
      hdfs:dfs.namenode.secondary.http-address: 0.0.0.0:9868
      hdfs:dfs.namenode.secondary.https-address: 0.0.0.0:9869
      hdfs:dfs.namenode.service.handler.count: '10'
      hdfs:dfs.namenode.servicerpc-address: daring-healer-421511-cluster-task1d-0workers-m:8051
      hive:hive.fetch.task.conversion: none
      mapred-env:HADOOP_JOB_HISTORYSERVER_HEAPSIZE: '4000'
      mapred:mapreduce.job.maps: '21'
      mapred:mapreduce.job.reduce.slowstart.completedmaps: '0.95'
      mapred:mapreduce.job.reduces: '7'
      mapred:mapreduce.jobhistory.recovery.store.class: org.apache.hadoop.mapreduce.v2.hs.HistoryServerLeveldbS
tateStoreService
      mapred:mapreduce.map.cpu.vcores: '1'
      mapred:mapreduce.map.java.opts: -Xmx2457m
      mapred:mapreduce.map.memory.mb: '3072'
      mapred:mapreduce.reduce.cpu.vcores: '1'
      mapred:mapreduce.reduce.java.opts: -Xmx2457m
      mapred:mapreduce.reduce.memory.mb: '3072'
      mapred:mapreduce.task.io.sort.mb: '256'
      mapred:yarn.app.mapreduce.am.command-opts: -Xmx2457m
      mapred:yarn.app.mapreduce.am.resource.cpu-vcores: '1'
      mapred:yarn.app.mapreduce.am.resource.mb: '3072'
      spark-env:SPARK_DAEMON_MEMORY: 4000m
      spark:spark.driver.maxResultSize: 3840m
      spark:spark.driver.memory: 7680m
      spark:spark.executor.cores: '4'
      spark:spark.executor.instances: '2'
      spark:spark.executor.memory: 11171m
      spark:spark.executorEnv.OPENBLAS_NUM_THREADS: '1'
      spark:spark.extraListeners: com.google.cloud.spark.performance.DataprocMetricsListener
      spark:spark.scheduler.mode: FAIR
      spark:spark.sql.cbo.enabled: 'true'
      spark:spark.ui.port: '0'
      spark:spark.yarn.am.memory: 640m
      yarn-env:YARN_NODEMANAGER_HEAPSIZE: '4000'
      yarn-env:YARN_RESOURCEMANAGER_HEAPSIZE: '4000'
      yarn-env:YARN_TIMELINESERVER_HEAPSIZE: '4000'
      yarn:yarn.nodemanager.address: 0.0.0.0:8026
      yarn:yarn.nodemanager.resource.cpu-vcores: '8'
      yarn:yarn.nodemanager.resource.memory-mb: '24576'
      yarn:yarn.resourcemanager.nodemanager-graceful-decommission-timeout-secs: '86400'
      yarn:yarn.scheduler.maximum-allocation-mb: '24576'
      yarn:yarn.scheduler.minimum-allocation-mb: '1024'
  tempBucket: dataproc-temp-us-west1-869525608453-ejfnpe5x
labels:
  goog-dataproc-autozone: enabled
  goog-dataproc-cluster-name: daring-healer-421511-cluster-task1d-0workers
  goog-dataproc-cluster-uuid: 830a382c-b5c5-4426-834e-622eb4cdd050
  goog-dataproc-location: us-west1
projectId: daring-healer-421511
status:
  state: RUNNING
  stateStartTime: '2024-05-03T11:13:19.818370Z'
statusHistory:
- state: CREATING
  stateStartTime: '2024-05-03T11:10:06.777603Z'
```

In [ ]:

```
#We submit the job to the cluster
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER --region $REGION ./spark_write_tfrec_paral.py
%time
```

```
Job [cfe4265100f34c1190312be7992be102] submitted.
Waiting for job output...
2024-05-03 11:13:39.247642: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dyna
mic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or di
rectory; LD_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-03 11:13:39.247702: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if
you do not have a GPU set up on your machine.
Tensorflow version 2.4.0
24/05/03 11:13:43 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
24/05/03 11:13:43 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
24/05/03 11:13:43 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
24/05/03 11:13:43 INFO org.spark_project.jetty.util.log: Logging initialized @11482ms to org.spark_project.jett
y.util.log.Slf4jLog
24/05/03 11:13:43 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-SNAPSHOT; built: unknown; git: unknow
n; jvm 1.8.0_382-b05
24/05/03 11:13:43 INFO org.spark_project.jetty.server.Server: Started @11632ms
24/05/03 11:13:43 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@4a63dd5d{HTTP/
1.1, (http/1.1)}{0.0.0.0:46221}
24/05/03 11:13:45 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at daring-healer-42
1511-cluster-task1d-0workers-m/10.138.0.45:8032
24/05/03 11:13:45 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at dari
ng-healer-421511-cluster-task1d-0workers-m/10.138.0.45:10200
24/05/03 11:13:45 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
24/05/03 11:13:45 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.
24/05/03 11:13:45 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = memory
-mb, units = Mi, type = COUNTABLE
24/05/03 11:13:45 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = vcores
, units = , type = COUNTABLE
24/05/03 11:13:49 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application
_1714734718789_0001
Writing TFRecords
2024-05-03 11:14:00.851941: I tensorflow/compiler/jit/xla_cpu_device.cc:41] Not creating XLA devices, tf_xla_en
able_xla_devices not set
2024-05-03 11:14:00.852373: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dyna
mic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared object file: No such file or directory; L
D_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-03 11:14:00.852414: W tensorflow/stream_executor/cuda/cuda_driver.cc:326] failed call to cuInit: UNKNOW
N ERROR (303)
2024-05-03 11:14:00.852460: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel driver does not a
ppear to be running on this host (daring-healer-421511-cluster-task1d-0workers-m): /proc/driver/nvidia/version
does not exist
2024-05-03 11:14:00.853899: I tensorflow/compiler/jit/xla_gpu_device.cc:99] Not creating XLA devices, tf_xla_en
able_xla_devices not set
2024-05-03 11:14:00.995579: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR op
timization passes are enabled (registered 2)
2024-05-03 11:14:00.996035: I tensorflow/core/platform/profile_utils/cpu_utils.cc:112] CPU Frequency: 220016500
0 Hz
24/05/03 11:14:02 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@4a63dd5d{HTTP/1.1, (http
/1.1)}{0.0.0.0:0}
Job [cfe4265100f34c1190312be7992be102] finished successfully.
done: true
driverControlFilesUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/830a382c-b5c5-4426-834e
-622eb4cdd050/jobs/cfe4265100f34c1190312be7992be102/
driverOutputResourceUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/830a382c-b5c5-4426-83
4e-622eb4cdd050/jobs/cfe4265100f34c1190312be7992be102/driveroutput
jobUuid: e0851ff2-fea4-3015-9428-35a3a45039d0
placement:
  clusterName: daring-healer-421511-cluster-task1d-0workers
  clusterUuid: 830a382c-b5c5-4426-834e-622eb4cdd050
pysparkJob:
  mainPythonFileUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/830a382c-b5c5-4426-834e-6
22eb4cdd050/jobs/cfe4265100f34c1190312be7992be102/staging/spark_write_tfrec_paral.py
reference:
  jobId: cfe4265100f34c1190312be7992be102
  projectId: daring-healer-421511
status:
  state: DONE
  stateStartTime: '2024-05-03T11:14:04.262895Z'
statusHistory:
- state: PENDING
  stateStartTime: '2024-05-03T11:13:29.718852Z'
- state: SETUP_DONE
  stateStartTime: '2024-05-03T11:13:29.752786Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2024-05-03T11:13:30.186087Z'
yarnApplications:
- name: spark_write_tfrec_paral.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://daring-healer-421511-cluster-task1d-0workers-m:8088/proxy/application_1714734718789_0001/
CPU times: user 6 µs, sys: 1e+03 ns, total: 7 µs
Wall time: 22.4 µs
```

In [ ]:

```
CLUSTER = '{}-cluster-task1d-0workers'.format(PROJECT)
REGION = 'us-west1'
!gcloud dataproc clusters delete $CLUSTER --region $REGION
#Due to the fact that we have quota related problems as we seem to have few quota available, after creating a c
```

```
The cluster 'daring-healer-421511-cluster-task1d-6vms' and all attached disks
will be deleted.

Do you want to continue (Y/n)?  Y

Waiting on operation [projects/daring-healer-421511/regions/us-central1/operations/bf22992a-d60e-37b9-851f-2a1e
6bf233b8].
Deleted [https://dataproc.googleapis.com/v1/projects/daring-healer-421511/regions/us-central1/clusters/daring-h
ealer-421511-cluster-task1d-6vms].
```

Section 2: Speed tests

We have seen that **reading from the pre-processed TFRecord files** is **faster** than reading individual image files and decoding on the fly. This task is about **measuring this effect** and **parallelizing the tests with PySpark**.

2.1 Speed test implementation

Here is **code for time measurement** to determine the **throughput in images per second**. It doesn't render the images but extracts and prints some basic information in order to make sure the image data are read. We write the information to the null device for longer measurements `null_file=open("/dev/null", mode='w')`. That way it will not clutter our cell output.

We use batches (`dset2 = dset1.batch(batch_size)`) and select a number of batches with (`dset3 = dset2.take(batch_number)`). Then we use the `time.time()` to take the **time measurement** and take it multiple times, reading from the same dataset to see if reading speed changes with mutiple readings.

We then **vary** the size of the batch (`batch_size`) and the number of batches (`batch_number`) and **store the results for different values**. Store also the **results for each repetition** over the same dataset (repeat 2 or 3 times).

The speed test should be combined in a **function** `time_configs()` that takes a configuration, i.e. a dataset and arrays of `batch_sizes`, `batch_numbers`, and `repetitions` (an array of integers starting from 1), as **arguments** and runs the time measurement for each combination of batch_size and batch_number for the requested number of repetitions.

In [ ]:
```python
# Here are some useful values for testing your code, use higher values later for actually testing throughput
batch_sizes = [2,4]
batch_numbers = [3,6]
repetitions = [1]

def time_configs(dataset, batch_sizes, batch_numbers, repetitions):
    dims = [len(batch_sizes),len(batch_numbers),len(repetitions)]
    print(dims)
    results = np.zeros(dims)
    params = np.zeros(dims + [3])
    print( results.shape )
    with open("/dev/null",mode='w') as null_file: # for printing the output without showing it
        tt = time.time() # for overall time taking
        for bsi,bs in enumerate(batch_sizes):
            for dsi, ds in enumerate(batch_numbers):
                batched_dataset = dataset.batch(bs)
                timing_set = batched_dataset.take(ds)
                for ri,rep in enumerate(repetitions):
                    print("bs: {}, ds: {}, rep: {}".format(bs,ds,rep))
                    t0 = time.time()
                    for image, label in timing_set:
                        #print("Image batch shape {}".format(image.numpy().shape),
                        print("Image batch shape {}, {})".format(image.numpy().shape,
                            [str(lbl) for lbl in label.numpy()]), null_file)
                    td = time.time() - t0 # duration for reading images
                    results[bsi,dsi,ri] = ( bs * ds) / td
                    params[bsi,dsi,ri] = [ bs, ds, rep ]
    print("total time: "+str(time.time()-tt))
    return results, params
```

Let's try this function with a **small number** of configurations of batch_sizes batch_numbers and repetions, so that we get a set of parameter combinations and corresponding reading speeds. Try reading from the image files (dataset4) and the TFRecord files (datasetTfrec).

In [ ]:
```python
[res,par] = time_configs(dataset4, batch_sizes, batch_numbers, repetitions)
print(res)
print(par)

print("=============")

[res,par] = time_configs(datasetTfrec, batch_sizes, batch_numbers, repetitions)
print(res)
print(par)
```

```
[2, 2, 1]
(2, 2, 1)
bs: 2, ds: 3, rep: 1
```

```
Image batch shape (2,), ["b'dandelion'", "b'sunflowers'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encodin
g='UTF-8'>
Image batch shape (2,), ["b'dandelion'", "b'tulips'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='U
TF-8'>
Image batch shape (2,), ["b'sunflowers'", "b'sunflowers'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encodi
ng='UTF-8'>
bs: 2, ds: 6, rep: 1
Image batch shape (2,), ["b'dandelion'", "b'tulips'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='U
TF-8'>
Image batch shape (2,), ["b'dandelion'", "b'tulips'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='U
TF-8'>
Image batch shape (2,), ["b'roses'", "b'roses'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'
>
Image batch shape (2,), ["b'sunflowers'", "b'tulips'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='
UTF-8'>
Image batch shape (2,), ["b'daisy'", "b'dandelion'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UT
F-8'>
Image batch shape (2,), ["b'dandelion'", "b'dandelion'"]) <_io.TextIOWrapper name='/dev/null' mode='w' encoding
='UTF-8'>
bs: 4, ds: 3, rep: 1
Image batch shape (4,), ["b'daisy'", "b'daisy'", "b'roses'", "b'tulips'"]) <_io.TextIOWrapper name='/dev/null'
mode='w' encoding='UTF-8'>
Image batch shape (4,), ["b'tulips'", "b'dandelion'", "b'daisy'", "b'daisy'"]) <_io.TextIOWrapper name='/dev/nu
ll' mode='w' encoding='UTF-8'>
Image batch shape (4,), ["b'tulips'", "b'roses'", "b'dandelion'", "b'dandelion'"]) <_io.TextIOWrapper name='/de
v/null' mode='w' encoding='UTF-8'>
bs: 4, ds: 6, rep: 1
Image batch shape (4,), ["b'dandelion'", "b'tulips'", "b'daisy'", "b'dandelion'"]) <_io.TextIOWrapper name='/de
v/null' mode='w' encoding='UTF-8'>
Image batch shape (4,), ["b'tulips'", "b'roses'", "b'daisy'", "b'sunflowers'"]) <_io.TextIOWrapper name='/dev/n
ull' mode='w' encoding='UTF-8'>
Image batch shape (4,), ["b'tulips'", "b'tulips'", "b'sunflowers'", "b'dandelion'"]) <_io.TextIOWrapper name='/
dev/null' mode='w' encoding='UTF-8'>
Image batch shape (4,), ["b'tulips'", "b'daisy'", "b'tulips'", "b'dandelion'"]) <_io.TextIOWrapper name='/dev/n
ull' mode='w' encoding='UTF-8'>
Image batch shape (4,), ["b'daisy'", "b'dandelion'", "b'daisy'", "b'dandelion'"]) <_io.TextIOWrapper name='/dev
/null' mode='w' encoding='UTF-8'>
Image batch shape (4,), ["b'tulips'", "b'sunflowers'", "b'dandelion'", "b'dandelion'"]) <_io.TextIOWrapper name
='/dev/null' mode='w' encoding='UTF-8'>
total time: 3.4361982345581055
[[[ 9.77924234]
  [14.86563858]]

 [[17.00530245]
  [19.15936353]]]
[[[[2. 3. 1.]]

  [[2. 6. 1.]]]


 [[[4. 3. 1.]]

  [[4. 6. 1.]]]]
=============
[2, 2, 1]
(2, 2, 1)
bs: 2, ds: 3, rep: 1
Image batch shape (2, 192, 192, 3), ['0', '0']) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['0', '0']) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['0', '0']) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
bs: 2, ds: 6, rep: 1
Image batch shape (2, 192, 192, 3), ['0', '0']) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['0', '0']) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['0', '0']) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['0', '0']) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['0', '0']) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
Image batch shape (2, 192, 192, 3), ['0', '0']) <_io.TextIOWrapper name='/dev/null' mode='w' encoding='UTF-8'>
bs: 4, ds: 3, rep: 1
Image batch shape (4, 192, 192, 3), ['0', '0', '0', '0']) <_io.TextIOWrapper name='/dev/null' mode='w' encoding
='UTF-8'>
Image batch shape (4, 192, 192, 3), ['0', '0', '0', '0']) <_io.TextIOWrapper name='/dev/null' mode='w' encoding
='UTF-8'>
Image batch shape (4, 192, 192, 3), ['0', '0', '0', '0']) <_io.TextIOWrapper name='/dev/null' mode='w' encoding
='UTF-8'>
bs: 4, ds: 6, rep: 1
Image batch shape (4, 192, 192, 3), ['0', '0', '0', '0']) <_io.TextIOWrapper name='/dev/null' mode='w' encoding
='UTF-8'>
Image batch shape (4, 192, 192, 3), ['0', '0', '0', '0']) <_io.TextIOWrapper name='/dev/null' mode='w' encoding
='UTF-8'>
Image batch shape (4, 192, 192, 3), ['0', '0', '0', '0']) <_io.TextIOWrapper name='/dev/null' mode='w' encoding
='UTF-8'>
Image batch shape (4, 192, 192, 3), ['0', '0', '0', '0']) <_io.TextIOWrapper name='/dev/null' mode='w' encoding
='UTF-8'>
Image batch shape (4, 192, 192, 3), ['0', '0', '0', '0']) <_io.TextIOWrapper name='/dev/null' mode='w' encoding
='UTF-8'>
Image batch shape (4, 192, 192, 3), ['0', '0', '0', '0']) <_io.TextIOWrapper name='/dev/null' mode='w' encoding
='UTF-8'>
total time: 0.8428754806518555
[[[ 24.65395851]
```

```
  [ 68.53605889]]

 [[ 58.35654881]
  [118.46587362]]]
[[[[2. 3. 1.]]

  [[2. 6. 1.]]]


 [[[4. 3. 1.]]

  [[4. 6. 1.]]]]
```

Task 2: Parallelising the speed test with Spark in the cloud. (36%)

As an exercise in **Spark programming and optimisation** as well as **performance analysis**, we will now implement the **speed test** with multiple parameters in parallel with Spark. Runing multiple tests in parallel would **not be a useful approach on a single machine, but it can be in the cloud** (you will be asked to reason about this later).

2a) Create the script (14%)

Your task is now to **port the speed test above to Spark** for running it in the cloud in Dataproc. **Adapt the speed testing** as a Spark program that performs the same actions as above, but **with Spark RDDs in a distributed way**. The distribution should be such that **each parameter combination (except repetition)** is processed in a separate Spark task.

More specifically:

- i) combine the previous cells to have the code to create a dataset and create a list of parameter combinations in an RDD (2%)
- ii) get a Spark context and create the dataset and run timing test for each combination in parallel (2%)
- iii) transform the resulting RDD to the structure (parameter_combination, images_per_second) and save these values in an array (2%)
- iv) create an RDD with all results for each parameter as (parameter_value,images_per_second) and collect the result for each parameter (2%)
- v) create an RDD with the average reading speeds for each parameter value and collect the results. Keep associativity in mind when implementing the average. (3%)
- vi) write the results to a pickle file in your bucket (2%)
- vii) Write your code it into a file using the *cell magic* `%%writefile spark_job.py` (1%)

Important: The task here is not to parallelize the pre-processing, but to run multiple speed tests in parallel using Spark.

In [ ]:
```
#Task 2a) --> i) Combine the previous cells to have the code to create a dataset and create a list of parameter

import itertools
import time
import tensorflow as tf
import pyspark
from pyspark import SparkContext

def read_tfrecord(record): #This function is created to read and parse a TFRecord file
    feature_description = {
        "image": tf.io.FixedLenFeature([], tf.string),
        "class": tf.io.FixedLenFeature([], tf.int64)
    }
    example = tf.io.parse_single_example(record, feature_description)
    image_decoded = tf.image.decode_jpeg(example['image'], channels=3)
    image_reshaped = tf.reshape(image_decoded, [*TARGET_SIZE, 3])
    class_label = example['class']
    return image_reshaped, class_label

def load_dataset(filenames): #This function is created to create a dataset from the TFRecord files
    option_no_order = tf.data.Options()
    option_no_order.experimental_deterministic = False
    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.with_options(option_no_order)
    dataset_new = dataset.map(read_tfrecord)
    return dataset_new

def decode_jpeg_and_label(filepath): #This function is created to decode the JPEG images from the filepath and
    image_data = tf.io.read_file(filepath)
    decoded_image = tf.image.decode_jpeg(image_data)
    label = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/')
    class_label = label.values[-2]
    return decoded_image, class_label

def resize_and_crop_image(image, label): #This function is created to resize and crop the images to the specifi
    image_width = tf.shape(image)[0]
    image_height = tf.shape(image)[1]
    target_width = TARGET_SIZE[1]
    target_height = TARGET_SIZE[0]
```

```
    resize_crit = (image_width * target_height) / (image_height * target_width)
    image = tf.cond(resize_crit < 1,
                    lambda: tf.image.resize(image, [image_width*target_width/image_width, image_height*target_w
                    lambda: tf.image.resize(image, [image_width*target_height/image_height, image_height*target
    new_width = tf.shape(image)[0]
    new_height = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (new_width - target_width) // 2, (new_height - target_height)
    return image, label

def recompress_image(image, label): #This function then recompresses the processed images to JPEG
    uint8_image = tf.cast(image, tf.uint8)
    jpeg_image = tf.image.encode_jpeg(uint8_image, optimize_size=True, chroma_downsampling=False)
    return jpeg_image, label

def load_image_jpeg_dataset(): #We load the dataset of the images from the JPEG files which are stored in a Goo
    GCS_PATTERN = 'gs://flowers-public/*/*.jpg'
    files_dataset = tf.data.Dataset.list_files(GCS_PATTERN)
    decoded_images_dataset = files_dataset.map(decode_jpeg_and_label)
    resized_dataset = decoded_images_dataset.map(resize_and_crop_image)
    image_jpeg_dataset = resized_dataset.map(recompress_image)
    return image_jpeg_dataset

def parameter_combinations(batch_sizes, batch_numbers, repetitions): #This function is used to generate all of
    return list(itertools.product(batch_sizes, batch_numbers, repetitions))
```

In [ ]:

```
#Task 2a) --> ii) Get a Spark context and create the dataset and run timing test for each combination in parall

sc = SparkContext.getOrCreate() #We set up the spark context and then define the parameter combinations

#These are the batch sizes, batch numbers and repetitions that we will be testing
batch_sizes = [2, 4]
batch_numbers = [3, 6, 9, 12]
repetitions = [1, 2, 3]

# We create an RDD from the parameter combinations
parameters_rdd = sc.parallelize(parameter_combinations(batch_sizes, batch_numbers, repetitions))

#This function is created to time the processing of the TFRecord batches
def time_configs(params):
    batch_size, batch_number, repetition = params
    TRF_dataset = load_dataset(filenames)
    results = []
    for repet in range(repetition):
        start_time = time.time()
        for _ in TRF_dataset.batch(batch_size).take(batch_number):
            pass
        total_time = time.time() - start_time
        images_per_second = (batch_size * batch_number) / total_time
        results.append(('TFRecord', params, images_per_second))
    return results

#This function is created to time the processing of the image data batches
def image_configs(params):
    batch_size, batch_number, repetition = params
    img_dataset = load_image_jpeg_dataset()
    image_processing_results = []
    for rep in range(repetition):
        start_time = time.time()
        for _ in img_dataset.batch(batch_size).take(batch_number):
            pass
        total_time = time.time() - start_time
        images_per_second = (batch_size * batch_number) / total_time
        image_processing_results.append(('Image', params, images_per_second))
    return image_processing_results

#These are the results from the time processing of the TFRecord and image data batches
resulting_rdd = parameters_rdd.flatMap(time_configs)
resulting_rdd_2 = parameters_rdd.flatMap(image_configs)
```

In [ ]:

```
#Task 2a) iii) --> Transform the resulting RDD to the structure ( parameter_combination, images_per_second ) an

# We merge the 2 resulting RDDs (TFRecord and image results) into a combined RDD using the .union function
results_rdd_combined = resulting_rdd.union(resulting_rdd_2)
final_results = results_rdd_combined.collect()
```

In [ ]:

```
#We print the results
print('Combined Results:')
for result in final_results:
  print(f'Dataset: {result[0]}, Parameters: (Batch Size, Batch Number, Repetitions): {result[1]}, Average Image
```

```
Combined Results:
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 1), Average Images per Second: 8
.672589037158751
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 2), Average Images per Second: 1
7.993788002722752
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 2), Average Images per Second: 1
7.54538818553928
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 1
```

7.0174340997521
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 1
6.36668498496057
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 1
7.577459122308287
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 1), Average Images per Second: 3
5.06518022480552
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 2), Average Images per Second: 6
2.61775185620161
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 2), Average Images per Second: 3
1.016898202212708
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 3
3.80220214169865
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 5
8.11136511631129
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 6
5.3718044131342
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 1), Average Images per Second: 5
2.836780669628894
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 2), Average Images per Second: 2
6.943425476985976
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 2), Average Images per Second: 5
2.70142005415514
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 6
2.243933486626176
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 8
5.23035205265268
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 5
3.89746859028771
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 1), Average Images per Second:
124.76518488588562
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 2), Average Images per Second:
70.49156628868383
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 2), Average Images per Second:
87.7592618908337
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second:
71.84318629014818
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second:
87.86152095131028
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second:
71.25399383750289
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 1), Average Images per Second: 1
7.126621922719373
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 2), Average Images per Second: 4
3.010672416047335
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 2), Average Images per Second: 4
0.26013289488314
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 4
4.41032989450576
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 3
2.35012318096772
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 3
5.68059083365648
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 1), Average Images per Second: 6
1.20923252605381
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 2), Average Images per Second: 6
0.09684433241912
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 2), Average Images per Second: 9
5.43860500747101
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 8
5.0917593197616
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 8
1.96495799290952
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 7
3.74270893388648
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 1), Average Images per Second: 1
37.92969186753922
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 2), Average Images per Second: 1
78.86751041256994
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 2), Average Images per Second: 2
11.33602994073996
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 1
50.4910997978767
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 1
55.1860537537269
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 1
84.60055137507564
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 1), Average Images per Second:
214.45601835577742
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 2), Average Images per Second:
234.73556619414836
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 2), Average Images per Second:
198.80063118022406
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second:
224.66301316666053
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second:
239.5821040455824
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second:
193.4701559088533

```
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 1), Average Images per Second: 10.2
39854168743527
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 2), Average Images per Second: 11.6
10985436568447
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 2), Average Images per Second: 11.1
21079467266252
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 10.8
6299562818131
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 12.8
04695732433617
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 11.0
02564193367723
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 1), Average Images per Second: 15.4
94775417772631
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 2), Average Images per Second: 11.8
03802045956633
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 2), Average Images per Second: 13.2
15869552374013
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 11.2
48791291803045
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 13.9
96541718784673
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 12.5
3376085680006
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 1), Average Images per Second: 21.3
6227155044353
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 2), Average Images per Second: 21.3
10168169671876
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 2), Average Images per Second: 20.4
68539749879625
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 22.7
0781641243588
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 21.8
51567605804203
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 20.8
05933665155454
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 1), Average Images per Second: 21.
41216905657873
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 2), Average Images per Second: 19.
367231355802254
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 2), Average Images per Second: 14.
21013999903443
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second: 16.
971485769016976
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second: 17.
76585315748315
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second: 16.
49439215583462
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 1), Average Images per Second: 17.6
12642334744724
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 2), Average Images per Second: 14.9
2172885974805
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 2), Average Images per Second: 15.4
66155180715782
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 14.5
6793520968833
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 13.6
00325121704469
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 11.8
24722610910998
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 1), Average Images per Second: 16.1
94989917476793
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 2), Average Images per Second: 17.1
6617579423364
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 2), Average Images per Second: 23.9
8909301748649
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 22.4
83141211436678
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 23.1
2113530106908
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 25.4
79811719823385
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 1), Average Images per Second: 24.6
85937601669412
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 2), Average Images per Second: 23.4
76991001014675
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 2), Average Images per Second: 18.5
90310381164603
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 13.2
57758419046294
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 16.3
97971014366842
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 20.1
09736781367477
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 1), Average Images per Second: 28.
02988234016515
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 2), Average Images per Second: 32.
78318570092852
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 2), Average Images per Second: 33.
```

```
119408568453366
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second: 31.
423293743860157
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second: 33.
204511786621985
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second: 30.
22334579437569
```

In [ ]:
```python
#Task 2a --> iv) Create an RDD with all results for each parameter as (parameter_value,images_per_second) and c

#This function extracts the results for each parameter
def param_results(index_param):
  return results_rdd_combined.map(lambda x: (x[1][index_param], x[2]))

#We create separate RDDs for each parameter
rdd_batch_size = param_results(0)
rdd_batch_number = param_results(1)
rdd_repetition = param_results(2)

#This function is created to calclate the average images per second for each parameter separately
def average_images_param(data):
  total_images = sum(images for images in data)
  count= len(data)
  return total_images / count

avg_batch_size_rdd = rdd_batch_size.groupByKey().mapValues(average_images_param).collect()
avg_batch_number_rdd = rdd_batch_number.groupByKey().mapValues(average_images_param).collect()
avg_repetition_rdd = rdd_repetition.groupByKey().mapValues(average_images_param).collect()

#We print the average images per second for each unique parameter
print("Average Images per Second By Each Batch Size (Batch_Size: AVG_Images_per_Sec):", avg_batch_size_rdd)
print("Average Images per Second By Each Batch Number (Batch_Number: AVG_Images_per_Sec):", avg_batch_number_rd
print("Average Images per Second By Each Repetition (Repetition: AVG_Images_per_Sec):", avg_repetition_rdd)
```

```
Average Images per Second By Each Batch Size (Batch_Size: AVG_Images_per_Sec): [(4, 63.0808506251448), (2, 29.6
6050117001694)]
Average Images per Second By Each Batch Number (Batch_Number: AVG_Images_per_Sec): [(12, 85.01437109141926), (9
, 72.21338865151826), (6, 46.229904480424985), (3, 23.234236315690193)]
Average Images per Second By Each Repetition (Repetition: AVG_Images_per_Sec): [(1, 43.104113930752455), (2, 51
.080036857344126), (3, 48.66364517169166)]
```

In [ ]:
```python
#Task 2a --> v)  Create an RDD with the average reading speeds for each parameter value and collect the results

#This function calculates the average reading speed using a tuple (total, count) for the associativity
def seq_op(accumu, n):
  return (accumu[0] + n, accumu[1] +1)

def comb_op( accumu1, accumu2):
  return (accumu1[0] + accumu2[0], accumu1[1] + accumu2[1])

#Now we can create a function that calculates the average images per second while also keeping associativity in
def average_assoc(rdd_params):
  return rdd_params.aggregateByKey((0,0), seq_op, comb_op).mapValues(lambda x: x[0] / x[1]).collect()

avg_batch_size = average_assoc(rdd_batch_size)
avg_batch_number = average_assoc(rdd_batch_number)
avg_repetition = average_assoc(rdd_repetition)

#Now we print the average reading speed for the associativity-respecting method
print("Average Images per Second By Each Batch Size (Associativity-method), (Batch_Size: AVG_Images_per_Sec):",
print("Average Images per Second By Each Batch Number (Associativity-method), (Batch_Number: AVG_Images_per_Sec
print("Average Images per Second By Repetition (Associativity-method), (Repetition: AVG_Images_per_Sec):", avg_
```

```
Average Images per Second By Each Batch Size (Associativity-method), (Batch_Size: AVG_Images_per_Sec): [(4, 71.
50003330036951), (2, 33.94330722930943)]
Average Images per Second By Each Batch Number (Associativity-method), (Batch_Number: AVG_Images_per_Sec): [(12
, 95.15781559985273), (9, 68.57971692643322), (6, 48.87393143768083), (3, 25.32879120167082)]
Average Images per Second By Repetition (Associativity-method), (Repetition: AVG_Images_per_Sec): [(1, 46.96156
111641283), (2, 54.5184233486253), (3, 49.760002537199995)]
```

In [ ]:
```python
#Task 2a --> vi) Write the results to a pickle file in your bucket
import pickle
from google.cloud import storage

#We store the results from previous tasks to a dictionary
results_task2a = {'Final Results Task iii': final_results,
                  'AVG Batch Size Task iv': avg_batch_size_rdd,
                  'AVG Batch Number Task iv': avg_batch_number_rdd,
                  'AVG Repetition Task iv': avg_repetition_rdd,
                  'AVG Batch Size Task v': avg_batch_size,
                  'AVG Batch Number Task v': avg_batch_number,
                  'AVG Repetition Task v': avg_repetition}

#We write the results to a local pickle file
with open('/tmp/all_results.pkl', 'wb') as f:
    pickle.dump(results_task2a, f)

#This function is used to upload the pickle file to the bucket that we specify
def upload_to_bucket(blob_name, path_to_file, bucket_name):
```

```
    try:
        client = storage.Client()
        bucket = client.get_bucket(bucket_name)
        blob = bucket.blob(blob_name)
        blob.upload_from_filename(path_to_file)
        print("File uploaded successfully")
    except Exception as e:
        print(f"An error occurred: {e}")

upload_to_bucket('results/results_task2a.pkl', '/tmp/all_results.pkl', 'daring-healer-421511-storage')
```

```
File uploaded successfully
```

In [ ]:

```
%%writefile spark_job.py
#Task 2a --> vii) Write your code it into a file using the cell magic %%writefile spark_job.py
import itertools
import time
import tensorflow as tf
import pyspark
from pyspark import SparkContext
#We declare the variables
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' #glob pattern for input files.
PROJECT = 'daring-healer-421511'
BUCKET = 'gs://{}-storage'.format(PROJECT) # This is the bucket storage.
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # We prefix for output file names.
filenames = tf.io.gfile.glob(GCS_OUTPUT + "*.tfrec")
TARGET_SIZE = [192,192]
PARTITIONS = 16 # This is the number of partitions that will be used later.

#Task 2a) --> i) Combine the previous cells to have the code to create a dataset and create a list of parameter
import itertools
import time
import tensorflow as tf
import pyspark
from pyspark import SparkContext

def read_tfrecord(record): #This function is created to read and parse a TFRecord file
    feature_description = {
        "image": tf.io.FixedLenFeature([], tf.string),
        "class": tf.io.FixedLenFeature([], tf.int64)
    }
    example = tf.io.parse_single_example(record, feature_description)
    image_decoded = tf.image.decode_jpeg(example['image'], channels=3)
    image_reshaped = tf.reshape(image_decoded, [*TARGET_SIZE, 3])
    class_label = example['class']
    return image_reshaped, class_label

def load_dataset(filenames): #This function is created to create a dataset from the TFRecord files
    option_no_order = tf.data.Options()
    option_no_order.experimental_deterministic = False
    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.with_options(option_no_order)
    dataset_new = dataset.map(read_tfrecord)
    return dataset_new

def decode_jpeg_and_label(filepath): #This function is created to decode the JPEG images from the filepath and
    image_data = tf.io.read_file(filepath)
    decoded_image = tf.image.decode_jpeg(image_data)
    label = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/')
    class_label = label.values[-2]
    return decoded_image, class_label

def resize_and_crop_image(image, label): #This function is created to resize and crop the images to the specifi
    image_width = tf.shape(image)[0]
    image_height = tf.shape(image)[1]
    target_width = TARGET_SIZE[1]
    target_height = TARGET_SIZE[0]
    resize_crit = (image_width * target_height) / (image_height * target_width)
    image = tf.cond(resize_crit < 1,
                    lambda: tf.image.resize(image, [image_width*target_width/image_width, image_height*target_w
                    lambda: tf.image.resize(image, [image_width*target_height/image_height, image_height*target
    new_width = tf.shape(image)[0]
    new_height = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (new_width - target_width) // 2, (new_height - target_height)
    return image, label

def recompress_image(image, label): #This function then recompresses the processed images to JPEG
    uint8_image = tf.cast(image, tf.uint8)
    jpeg_image = tf.image.encode_jpeg(uint8_image, optimize_size=True, chroma_downsampling=False)
    return jpeg_image, label

def load_image_jpeg_dataset(): #We load the dataset of the images from the JPEG files which are stored in a Goo
    GCS_PATTERN = 'gs://flowers-public/*/*.jpg'
    files_dataset = tf.data.Dataset.list_files(GCS_PATTERN)
    decoded_images_dataset = files_dataset.map(decode_jpeg_and_label)
    resized_dataset = decoded_images_dataset.map(resize_and_crop_image)
    image_jpeg_dataset = resized_dataset.map(recompress_image)
    return image_jpeg_dataset
```

```python
def parameter_combinations(batch_sizes, batch_numbers, repetitions): #This function is used to generate all of
    return list(itertools.product(batch_sizes, batch_numbers, repetitions))

#Task 2a) --> ii) Get a Spark context and create the dataset and run timing test for each combination in parall

sc = SparkContext.getOrCreate() #We set up the spark context and then define the parameter combinations

#These are the batch sizes, batch numbers and repetitions that we will be testing
batch_sizes = [2, 4]
batch_numbers = [3, 6, 9, 12]
repetitions = [1, 2, 3]

# We create an RDD from the parameter combinations
parameters_rdd = sc.parallelize(parameter_combinations(batch_sizes, batch_numbers, repetitions))

#This function is created to time the processing of the TFRecord batches
def time_configs(params):
    batch_size, batch_number, repetition = params
    TRF_dataset = load_dataset(filenames)
    results = []
    for repet in range(repetition):
        start_time = time.time()
        for _ in TRF_dataset.batch(batch_size).take(batch_number):
            pass
        total_time = time.time() - start_time
        images_per_second = (batch_size * batch_number) / total_time
        results.append(('TFRecord', params, images_per_second))
    return results

#This function is created to time the processing of the image data batches
def image_configs(params):
    batch_size, batch_number, repetition = params
    img_dataset = load_image_jpeg_dataset()
    image_processing_results = []
    for rep in range(repetition):
        start_time = time.time()
        for _ in img_dataset.batch(batch_size).take(batch_number):
            pass
        total_time = time.time() - start_time
        images_per_second = (batch_size * batch_number) / total_time
        image_processing_results.append(('Image', params, images_per_second))
    return image_processing_results

#These are the results from the time processing of the TFRecord and image data batches
resulting_rdd = parameters_rdd.flatMap(time_configs)
resulting_rdd_2 = parameters_rdd.flatMap(image_configs)

#Task 2a) iii) --> Transform the resulting RDD to the structure ( parameter_combination, images_per_second ) an

# We merge the 2 resulting RDDs (TFRecord and image results) into a combined RDD using the .union function
results_rdd_combined = resulting_rdd.union(resulting_rdd_2)
final_results = results_rdd_combined.collect()

#We print the results
print('Combined Results:')
for result in final_results:
  print(f'Dataset: {result[0]}, Parameters: (Batch Size, Batch Number, Repetitions): {result[1]}, Average Image

#Task 2a --> iv) Create an RDD with all results for each parameter as (parameter_value,images_per_second) and c

#This function extracts the results for each parameter
def param_results(index_param):
  return results_rdd_combined.map(lambda x: (x[1][index_param], x[2]))

#We create separate RDDs for each parameter
rdd_batch_size = param_results(0)
rdd_batch_number = param_results(1)
rdd_repetition = param_results(2)

#This function is created to calclate the average images per second for each parameter separately
def average_images_param(data):
  total_images = sum(images for images in data)
  count= len(data)
  return total_images / count

avg_batch_size_rdd = rdd_batch_size.groupByKey().mapValues(average_images_param).collect()
avg_batch_number_rdd = rdd_batch_number.groupByKey().mapValues(average_images_param).collect()
avg_repetition_rdd = rdd_repetition.groupByKey().mapValues(average_images_param).collect()

#We print the average images per second for each unique parameter
print("Average Images per Second By Each Batch Size (Batch_Size: AVG_Images_per_Sec):", avg_batch_size_rdd)
print("Average Images per Second By Each Batch Number (Batch_Number: AVG_Images_per_Sec):", avg_batch_number_rd
print("Average Images per Second By Each Repetition (Repetition: AVG_Images_per_Sec):", avg_repetition_rdd)

#Task 2a --> v)  Create an RDD with the average reading speeds for each parameter value and collect the results

#This function calculates the average reading speed using a tuple (total, count) for the associativity
def seq_op(accumu, n):
  return (accumu[0] + n, accumu[1] +1)
```

```
def comb_op( accumu1, accumu2):
  return (accumu1[0] + accumu2[0], accumu1[1] + accumu2[1])

#Now we can create a function that calculates the average images per second while also keeping associativity in
def average_assoc(rdd_params):
  return rdd_params.aggregateByKey((0,0), seq_op, comb_op).mapValues(lambda x: x[0] / x[1]).collect()

avg_batch_size = average_assoc(rdd_batch_size)
avg_batch_number = average_assoc(rdd_batch_number)
avg_repetition = average_assoc(rdd_repetition)

#Now we print the average reading speed for the associativity-respecting method
print("Average Images per Second By Each Batch Size (Associativity-method), (Batch_Size: AVG_Images_per_Sec):",
print("Average Images per Second By Each Batch Number (Associativity-method), (Batch_Number: AVG_Images_per_Sec
print("Average Images per Second By Repetition (Associativity-method), (Repetition: AVG_Images_per_Sec):", avg_

#Task 2a --> vi) Write the results to a pickle file in your bucket
import pickle
from google.cloud import storage

#We store the results from previous tasks to a dictionary
results_task2a = {'Final Results Task iii': final_results,
                  'AVG Batch Size Task iv': avg_batch_size_rdd,
                  'AVG Batch Number Task iv': avg_batch_number_rdd,
                  'AVG Repetition Task iv': avg_repetition_rdd,
                  'AVG Batch Size Task v': avg_batch_size,
                  'AVG Batch Number Task v': avg_batch_number,
                  'AVG Repetition Task v': avg_repetition}

#We write the results to a local pickle file
with open('/tmp/all_results.pkl', 'wb') as f:
    pickle.dump(results_task2a, f)

#This function is used to upload the pickle file to the bucket that we specify
def upload_to_bucket(blob_name, path_to_file, bucket_name):
    try:
        client = storage.Client()
        bucket = client.get_bucket(bucket_name)
        blob = bucket.blob(blob_name)
        blob.upload_from_filename(path_to_file)
        print("File uploaded successfully")
    except Exception as e:
        print(f"An error occurred: {e}")

upload_to_bucket('results/results_task2a.pkl', '/tmp/all_results.pkl', 'daring-healer-421511-storage')
```

```
Writing spark_job.py
```

2b) Testing the code and collecting results (4%)

i) First, test locally with `%run`.

It is useful to create a **new filename argument**, so that old results don't get overwritten.

You can for instance use `datetime.datetime.now().strftime("%y%m%d-%H%M")` to get a string with the current date and time and use that in the file name.

In [ ]:
```
#Task 2b) i) --> We test locally on spark with the %run command
%run ./spark_job.py
```

```
Combined Results:
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 1), Average Images per Second: 1
6.01438670375341
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 2), Average Images per Second: 1
6.577173719008155
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 2), Average Images per Second: 2
2.828628966418112
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 1
6.226321565310016
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 1
6.18267529800162
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 2
4.96795297269142
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 1), Average Images per Second: 3
0.460940705692675
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 2), Average Images per Second: 3
3.01654840442734
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 2), Average Images per Second: 2
8.606320465552194
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 3
0.931654528146392
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 2
9.76851229737019
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 2
9.99403952790582
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 1), Average Images per Second: 4
```

9.307337373404884
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 2), Average Images per Second: 4
2.828155207624235
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 2), Average Images per Second: 4
7.21232939675631
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 6
3.09369351149472
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 5
1.9894172169925
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 2
5.908899885001812
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 1), Average Images per Second:
87.57145156281128
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 2), Average Images per Second:
77.0613944700729
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 2), Average Images per Second:
66.78145677790353
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second:
56.90651265967334
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second:
67.97688348417053
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second:
69.47974448083434
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 1), Average Images per Second: 3
1.52193937814943
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 2), Average Images per Second: 3
1.43246799557599
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 2), Average Images per Second: 4
4.46396134156091
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 4
4.04590144997545
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 4
4.619029086966584
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 3
0.213648462233877
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 1), Average Images per Second: 7
2.62733067585361
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 2), Average Images per Second: 7
0.3238844674953
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 2), Average Images per Second: 7
5.87460673276522
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 6
6.30935006317165
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 7
9.35835517732062
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 9
5.28710239573049
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 1), Average Images per Second: 1
07.36883781785085
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 2), Average Images per Second: 1
19.71241419662353
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 2), Average Images per Second: 1
27.62555605611993
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 1
27.85151792775385
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 9
5.42045112002715
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 1
25.39348214411764
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 1), Average Images per Second:
146.6222648509173
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 2), Average Images per Second:
170.81595802538058
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 2), Average Images per Second:
167.11498033981368
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second:
140.75947781146175
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second:
130.56851217602954
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second:
156.44319561985827
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 1), Average Images per Second: 7.54
0068767729742
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 2), Average Images per Second: 8.61
9339088047957
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 2), Average Images per Second: 7.73
3997311552028
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 6.97
5080114480141
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 8.03
8260419611245
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 6.84
2594483306547
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 1), Average Images per Second: 13.0
03755799890817
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 2), Average Images per Second: 21.5
66938233505773
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 2), Average Images per Second: 18.4
6983475305065

```
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 19.2
4964708861011
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 18.2
3090589748503
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 16.5
52477495550413
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 1), Average Images per Second: 20.1
96836668196863
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 2), Average Images per Second: 20.7
55415764142402
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 2), Average Images per Second: 20.4
63712962827437
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 20.7
8452852503642
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 18.9
06466453421295
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 13.8
6166440282501
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 1), Average Images per Second: 13.
992706960803215
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 2), Average Images per Second: 17.
81707072947241
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 2), Average Images per Second: 16.
35004650986014
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second: 15.
499468252920991
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second: 15.
482644963168015
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second: 18.
913996368766885
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 1), Average Images per Second: 10.6
60236987121296
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 2), Average Images per Second: 13.5
89437467549631
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 2), Average Images per Second: 10.8
26890508292205
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 12.2
23032315298738
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 10.8
23679793004171
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 13.4
58764663302576
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 1), Average Images per Second: 25.7
7477232731199
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 2), Average Images per Second: 22.9
00370497935942
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 2), Average Images per Second: 22.9
63110580660704
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 22.5
14957888163416
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 21.2
68000355793163
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 23.0
65777244955758
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 1), Average Images per Second: 21.0
5119878741035
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 2), Average Images per Second: 18.4
02186301685063
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 2), Average Images per Second: 16.5
50284022248274
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 18.6
81516371309428
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 18.3
06123057637123
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 17.2
21007093125543
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 1), Average Images per Second: 28.
068570281359662
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 2), Average Images per Second: 33.
269154544638816
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 2), Average Images per Second: 35.
24326580170728
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second: 35.
524408467914476
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second: 34.
96797752340617
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second: 34.
515883438473146
Average Images per Second By Each Batch Size (Batch_Size: AVG_Images_per_Sec): [(4, 63.135423818380644), (2, 31
.324376398417105)]
Average Images per Second By Each Batch Number (Batch_Number: AVG_Images_per_Sec): [(12, 83.40902202446058), (9
, 53.827162916524486), (6, 38.74526696990669), (3, 22.04843169570539)]
Average Images per Second By Each Repetition (Repetition: AVG_Images_per_Sec): [(1, 60.39748965139749), (2, 60.
39259585420249), (3, 56.14607246572556)]
Average Images per Second By Each Batch Size (Associativity-method), (Batch_Size: AVG_Images_per_Sec): [(4, 63.
02181885053674), (2, 29.031656682472033)]
Average Images per Second By Each Batch Number (Associativity-method), (Batch_Number: AVG_Images_per_Sec): [(12
, 74.2086218855064), (9, 50.873770902546404), (6, 42.07954322345875), (3, 21.54856210188757)]
Average Images per Second By Repetition (Associativity-method), (Repetition: AVG_Images_per_Sec): [(1, 50.39164
```

```
548695515), (2, 56.650266009866115), (3, 55.550553089101506)]
File uploaded successfully
```

In [ ]:

```
#New filename argument

%%writefile spark_job_2b.py
#Task 2a --> vii) Write your code it into a file using the cell magic %%writefile spark_job.py
import os
os.environ['PROTOCOL_BUFFERS_PYTHON_IMPLEMENTATION'] = 'python'
import itertools
import time
import tensorflow as tf
import google.protobuf
import pyspark
from pyspark import SparkContext
#We declare the variables
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' #glob pattern for input files.
PROJECT = 'daring-healer-421511'
BUCKET = 'gs://{}-storage'.format(PROJECT) # This is the bucket storage.
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # We prefix for output file names.
filenames = tf.io.gfile.glob(GCS_OUTPUT + "*.tfrec")
TARGET_SIZE = [192,192]
PARTITIONS = 16 # This is the number of partitions that will be used later.

#Task 2a) --> i) Combine the previous cells to have the code to create a dataset and create a list of parameter
import itertools
import time
import tensorflow as tf
import pyspark
from pyspark import SparkContext

def read_tfrecord(record): #This function is created to read and parse a TFRecord file
    feature_description = {
        "image": tf.io.FixedLenFeature([], tf.string),
        "class": tf.io.FixedLenFeature([], tf.int64)
    }
    example = tf.io.parse_single_example(record, feature_description)
    image_decoded = tf.image.decode_jpeg(example['image'], channels=3)
    image_reshaped = tf.reshape(image_decoded, [*TARGET_SIZE, 3])
    class_label = example['class']
    return image_reshaped, class_label

def load_dataset(filenames): #This function is created to create a dataset from the TFRecord files
    option_no_order = tf.data.Options()
    option_no_order.experimental_deterministic = False
    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.with_options(option_no_order)
    dataset_new = dataset.map(read_tfrecord)
    return dataset_new

def decode_jpeg_and_label(filepath): #This function is created to decode the JPEG images from the filepath and
    image_data = tf.io.read_file(filepath)
    decoded_image = tf.image.decode_jpeg(image_data)
    label = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/')
    class_label = label.values[-2]
    return decoded_image, class_label

def resize_and_crop_image(image, label): #This function is created to resize and crop the images to the specifi
    image_width = tf.shape(image)[0]
    image_height = tf.shape(image)[1]
    target_width = TARGET_SIZE[1]
    target_height = TARGET_SIZE[0]
    resize_crit = (image_width * target_height) / (image_height * target_width)
    image = tf.cond(resize_crit < 1,
                    lambda: tf.image.resize(image, [image_width*target_width/image_width, image_height*target_w
                    lambda: tf.image.resize(image, [image_width*target_height/image_height, image_height*target
    new_width = tf.shape(image)[0]
    new_height = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (new_width - target_width) // 2, (new_height - target_height)
    return image, label

def recompress_image(image, label): #This function then recompresses the processed images to JPEG
    uint8_image = tf.cast(image, tf.uint8)
    jpeg_image = tf.image.encode_jpeg(uint8_image, optimize_size=True, chroma_downsampling=False)
    return jpeg_image, label

def load_image_jpeg_dataset(): #We load the dataset of the images from the JPEG files which are stored in a Goo
    GCS_PATTERN = 'gs://flowers-public/*/*.jpg'
    files_dataset = tf.data.Dataset.list_files(GCS_PATTERN)
    decoded_images_dataset = files_dataset.map(decode_jpeg_and_label)
    resized_dataset = decoded_images_dataset.map(resize_and_crop_image)
    image_jpeg_dataset = resized_dataset.map(recompress_image)
    return image_jpeg_dataset

def parameter_combinations(batch_sizes, batch_numbers, repetitions): #This function is used to generate all of
    return list(itertools.product(batch_sizes, batch_numbers, repetitions))

#Task 2a) --> ii) Get a Spark context and create the dataset and run timing test for each combination in parall
```

```
sc = SparkContext.getOrCreate() #We set up the spark context and then define the parameter combinations

#These are the batch sizes, batch numbers and repetitions that we will be testing
batch_sizes = [2, 4]
batch_numbers = [3, 6, 9, 12]
repetitions = [1, 2, 3]

# We create an RDD from the parameter combinations
parameters_rdd = sc.parallelize(parameter_combinations(batch_sizes, batch_numbers, repetitions))

#This function is created to time the processing of the TFRecord batches
def time_configs(params):
    batch_size, batch_number, repetition = params
    TRF_dataset = load_dataset(filenames)
    results = []
    for repet in range(repetition):
        start_time = time.time()
        for _ in TRF_dataset.batch(batch_size).take(batch_number):
            pass
        total_time = time.time() - start_time
        images_per_second = (batch_size * batch_number) / total_time
        results.append(('TFRecord', params, images_per_second))
    return results

#This function is created to time the processing of the image data batches
def image_configs(params):
    batch_size, batch_number, repetition = params
    img_dataset = load_image_jpeg_dataset()
    image_processing_results = []
    for rep in range(repetition):
        start_time = time.time()
        for _ in img_dataset.batch(batch_size).take(batch_number):
            pass
        total_time = time.time() - start_time
        images_per_second = (batch_size * batch_number) / total_time
        image_processing_results.append(('Image', params, images_per_second))
    return image_processing_results

#These are the results from the time processing of the TFRecord and image data batches
resulting_rdd = parameters_rdd.flatMap(time_configs)
resulting_rdd_2 = parameters_rdd.flatMap(image_configs)

#Task 2a) iii) --> Transform the resulting RDD to the structure ( parameter_combination, images_per_second ) an

# We merge the 2 resulting RDDs (TFRecord and image results) into a combined RDD using the .union function
results_rdd_combined = resulting_rdd.union(resulting_rdd_2)
final_results = results_rdd_combined.collect()

#We print the results
print('Combined Results:')
for result in final_results:
  print(f'Dataset: {result[0]}, Parameters: (Batch Size, Batch Number, Repetitions): {result[1]}, Average Image

#Task 2a --> iv) Create an RDD with all results for each parameter as (parameter_value,images_per_second) and c

#This function extracts the results for each parameter
def param_results(index_param):
  return results_rdd_combined.map(lambda x: (x[1][index_param], x[2]))

#We create separate RDDs for each parameter
rdd_batch_size = param_results(0)
rdd_batch_number = param_results(1)
rdd_repetition = param_results(2)

#This function is created to calclate the average images per second for each parameter separately
def average_images_param(data):
  total_images = sum(images for images in data)
  count= len(data)
  return total_images / count

avg_batch_size_rdd = rdd_batch_size.groupByKey().mapValues(average_images_param).collect()
avg_batch_number_rdd = rdd_batch_number.groupByKey().mapValues(average_images_param).collect()
avg_repetition_rdd = rdd_repetition.groupByKey().mapValues(average_images_param).collect()

#We print the average images per second for each unique parameter
print("Average Images per Second By Each Batch Size (Batch_Size: AVG_Images_per_Sec):", avg_batch_size_rdd)
print("Average Images per Second By Each Batch Number (Batch_Number: AVG_Images_per_Sec):", avg_batch_number_rd
print("Average Images per Second By Each Repetition (Repetition: AVG_Images_per_Sec):", avg_repetition_rdd)

#Task 2a --> v)  Create an RDD with the average reading speeds for each parameter value and collect the results

#This function calculates the average reading speed using a tuple (total, count) for the associativity
def seq_op(accumu, n):
  return (accumu[0] + n, accumu[1] +1)

def comb_op( accumu1, accumu2):
  return (accumu1[0] + accumu2[0], accumu1[1] + accumu2[1])

#Now we can create a function that calculates the average images per second while also keeping associativity in
```

```
def average_assoc(rdd_params):
  return rdd_params.aggregateByKey((0,0), seq_op, comb_op).mapValues(lambda x: x[0] / x[1]).collect()

avg_batch_size = average_assoc(rdd_batch_size)
avg_batch_number = average_assoc(rdd_batch_number)
avg_repetition = average_assoc(rdd_repetition)

#Now we print the average reading speed for the associativity-respecting method
print("Average Images per Second By Each Batch Size (Associativity-method), (Batch_Size: AVG_Images_per_Sec):",
print("Average Images per Second By Each Batch Number (Associativity-method), (Batch_Number: AVG_Images_per_Sec
print("Average Images per Second By Repetition (Associativity-method), (Repetition: AVG_Images_per_Sec):", avg_

#Task 2a --> vi) Write the results to a pickle file in your bucket
import pickle
from google.cloud import storage

#We store the results from previous tasks to a dictionary
results_task2b = {'Final Results Task iii': final_results,
                  'AVG Batch Size Task iv': avg_batch_size_rdd,
                  'AVG Batch Number Task iv': avg_batch_number_rdd,
                  'AVG Repetition Task iv': avg_repetition_rdd,
                  'AVG Batch Size Task v': avg_batch_size,
                  'AVG Batch Number Task v': avg_batch_number,
                  'AVG Repetition Task v': avg_repetition}

#We write the results to a local pickle file
with open('/tmp/all_results.pkl', 'wb') as f:
    pickle.dump(results_task2b, f)

#This function is used to upload the pickle file to the bucket that we specify
def upload_to_bucket(blob_name, path_to_file, bucket_name):
    try:
        client = storage.Client()
        bucket = client.get_bucket(bucket_name)
        blob = bucket.blob(blob_name)
        blob.upload_from_filename(path_to_file)
        print("File uploaded successfully")
    except Exception as e:
        print(f"An error occurred: {e}")

upload_to_bucket('results/results_task2b.pkl', '/tmp/all_results.pkl', 'daring-healer-421511-storage')
```

```
Writing spark_job_2b.py
```

ii) Cloud

If you have a cluster running, you can run the speed test job in the cloud.

While you run this job, switch to the Dataproc web page and take **screenshots of the CPU and network load** over time. They are displayed with some delay, so you may need to wait a little. These images will be useful in the next task. Again, don't use the SCREENSHOT function that Google provides, but just take a picture of the graphs you see for the VMs.

In [ ]:
```
#We will set up the maximal cluster which is a cluster with 1 machine and 7 workers using maximal SSD size (100
CLUSTER = '{}-maximalcluster7task2b'.format(PROJECT)
REGION = 'us-west1'
PIP_PACKAGES = 'tensorflow==2.4.0'
!gcloud dataproc clusters create $CLUSTER \
    --region $REGION \
    --bucket $PROJECT-storage \
    --image-version 1.5-ubuntu18 \
    --master-machine-type n1-standard-1 \
    --master-boot-disk-type pd-ssd --master-boot-disk-size 100\
    --num-workers 7 --worker-machine-type n1-standard-1 --worker-boot-disk-size 100 \
    --initialization-actions gs://goog-dataproc-initialization-actions-$REGION/python/pip-install.sh \
    --metadata PIP_PACKAGES='tensorflow==2.4.0 matplotlib google-cloud-storage==1.42.0 google-api-core==1.31.0
#Reference link for code: https://cloud.google.com/dataproc/docs/guides/create-cluster
```

```
Waiting on operation [projects/daring-healer-421511/regions/us-west1/operations/9dbcbb79-60f8-3463-a9a4-10ba233
54652].

WARNING: Creating clusters using the n1-standard-1 machine type is not recommended. Consider using a machine ty
pe with higher memory.
WARNING: Don't create production clusters that reference initialization actions located in the gs://goog-datapr
oc-initialization-actions-REGION public buckets. These scripts are provided as reference implementations, and t
hey are synchronized with ongoing GitHub repository changes—a new version of a initialization action in public
buckets may break your cluster creation. Instead, copy the following initialization actions from public buckets
into your bucket : gs://goog-dataproc-initialization-actions-us-west1/python/pip-install.sh
WARNING: Failed to validate permissions required for default service account: '869525608453-compute@developer.g
serviceaccount.com'. Cluster creation could still be successful if required permissions have been granted to th
e respective service accounts as mentioned in the document https://cloud.google.com/dataproc/docs/concepts/conf
iguring-clusters/service-accounts#dataproc_service_accounts_2. This could be due to Cloud Resource Manager API
hasn't been enabled in your project '869525608453' before or it is disabled. Enable it by visiting 'https://con
sole.developers.google.com/apis/api/cloudresourcemanager.googleapis.com/overview?project=869525608453'.
WARNING: For PD-Standard without local SSDs, we strongly recommend provisioning 1TB or larger to ensure consist
ently high I/O performance. See https://cloud.google.com/compute/docs/disks/performance for information on disk
I/O performance.
WARNING: The firewall rules for specified network or subnetwork would allow ingress traffic from 0.0.0.0/0, whi
ch could be a security risk.
WARNING: The specified custom staging bucket 'daring-healer-421511-storage' is not using uniform bucket level a
ccess IAM configuration. It is recommended to update bucket to enable the same. See https://cloud.google.com/st
orage/docs/uniform-bucket-level-access.
Created [https://dataproc.googleapis.com/v1/projects/daring-healer-421511/regions/us-west1/clusters/daring-heal
er-421511-maximalcluster7task2b] Cluster placed in zone [us-west1-c].
```

In [ ]:
```
#We get information on the single machine cluster we created
!gcloud dataproc clusters describe $CLUSTER --region $REGION
```

```
clusterName: daring-healer-421511-maximalcluster7task2b
clusterUuid: 8abd7648-fc9d-4875-9486-ada5038b61b3
config:
  configBucket: daring-healer-421511-storage
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow==2.4.0 matplotlib google-cloud-storage==1.42.0 google-api-core==1.31.0
        protobuf==3.17.3
    networkUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/global/networks/default
    serviceAccountScopes:
    - https://www.googleapis.com/auth/bigquery
    - https://www.googleapis.com/auth/bigtable.admin.table
    - https://www.googleapis.com/auth/bigtable.data
    - https://www.googleapis.com/auth/cloud.useraccounts.readonly
    - https://www.googleapis.com/auth/devstorage.full_control
    - https://www.googleapis.com/auth/devstorage.read_write
    - https://www.googleapis.com/auth/logging.write
    - https://www.googleapis.com/auth/monitoring.write
    zoneUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/zones/us-west1-c
  initializationActions:
  - executableFile: gs://goog-dataproc-initialization-actions-us-west1/python/pip-install.sh
    executionTimeout: 600s
  masterConfig:
    diskConfig:
      bootDiskSizeGb: 100
      bootDiskType: pd-ssd
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-5-ubu18-20
230909-165100-rc01
    instanceNames:
    - daring-healer-421511-maximalcluster7task2b-m
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/zones/us-west1-c/machin
eTypes/n1-standard-1
    minCpuPlatform: AUTOMATIC
    numInstances: 1
    preemptibility: NON_PREEMPTIBLE
  softwareConfig:
    imageVersion: 1.5.90-ubuntu18
    properties:
      capacity-scheduler:yarn.scheduler.capacity.root.default.ordering-policy: fair
      core:fs.gs.block.size: '134217728'
      core:fs.gs.metadata.cache.enable: 'false'
      core:hadoop.ssl.enabled.protocols: TLSv1,TLSv1.1,TLSv1.2
      distcp:mapreduce.map.java.opts: -Xmx576m
      distcp:mapreduce.map.memory.mb: '768'
      distcp:mapreduce.reduce.java.opts: -Xmx576m
      distcp:mapreduce.reduce.memory.mb: '768'
      hdfs:dfs.datanode.address: 0.0.0.0:9866
      hdfs:dfs.datanode.http.address: 0.0.0.0:9864
      hdfs:dfs.datanode.https.address: 0.0.0.0:9865
      hdfs:dfs.datanode.ipc.address: 0.0.0.0:9867
      hdfs:dfs.namenode.handler.count: '60'
      hdfs:dfs.namenode.http-address: 0.0.0.0:9870
      hdfs:dfs.namenode.https-address: 0.0.0.0:9871
      hdfs:dfs.namenode.lifeline.rpc-address: daring-healer-421511-maximalcluster7task2b-m:8050
      hdfs:dfs.namenode.secondary.http-address: 0.0.0.0:9868
      hdfs:dfs.namenode.secondary.https-address: 0.0.0.0:9869
      hdfs:dfs.namenode.service.handler.count: '30'
```

```
      hdfs:dfs.namenode.servicerpc-address: daring-healer-421511-maximalcluster7task2b-m:8051
      hive:hive.fetch.task.conversion: none
      mapred-env:HADOOP_JOB_HISTORYSERVER_HEAPSIZE: '1000'
      mapred:mapreduce.job.maps: '60'
      mapred:mapreduce.job.reduce.slowstart.completedmaps: '0.95'
      mapred:mapreduce.job.reduces: '7'
      mapred:mapreduce.jobhistory.recovery.store.class: org.apache.hadoop.mapreduce.v2.hs.HistoryServerLeveldbS
tateStoreService
      mapred:mapreduce.map.cpu.vcores: '1'
      mapred:mapreduce.map.java.opts: -Xmx819m
      mapred:mapreduce.map.memory.mb: '1024'
      mapred:mapreduce.reduce.cpu.vcores: '1'
      mapred:mapreduce.reduce.java.opts: -Xmx1638m
      mapred:mapreduce.reduce.memory.mb: '2048'
      mapred:mapreduce.task.io.sort.mb: '256'
      mapred:yarn.app.mapreduce.am.command-opts: -Xmx819m
      mapred:yarn.app.mapreduce.am.resource.cpu-vcores: '1'
      mapred:yarn.app.mapreduce.am.resource.mb: '1024'
      spark-env:SPARK_DAEMON_MEMORY: 1000m
      spark:spark.driver.maxResultSize: 480m
      spark:spark.driver.memory: 960m
      spark:spark.executor.cores: '1'
      spark:spark.executor.instances: '2'
      spark:spark.executor.memory: 2688m
      spark:spark.executorEnv.OPENBLAS_NUM_THREADS: '1'
      spark:spark.extraListeners: com.google.cloud.spark.performance.DataprocMetricsListener
      spark:spark.scheduler.mode: FAIR
      spark:spark.sql.cbo.enabled: 'true'
      spark:spark.ui.port: '0'
      spark:spark.yarn.am.memory: 640m
      yarn-env:YARN_NODEMANAGER_HEAPSIZE: '1000'
      yarn-env:YARN_RESOURCEMANAGER_HEAPSIZE: '1000'
      yarn-env:YARN_TIMELINESERVER_HEAPSIZE: '1000'
      yarn:yarn.nodemanager.address: 0.0.0.0:8026
      yarn:yarn.nodemanager.resource.cpu-vcores: '1'
      yarn:yarn.nodemanager.resource.memory-mb: '3072'
      yarn:yarn.resourcemanager.nodemanager-graceful-decommission-timeout-secs: '86400'
      yarn:yarn.scheduler.maximum-allocation-mb: '3072'
      yarn:yarn.scheduler.minimum-allocation-mb: '256'
  tempBucket: dataproc-temp-us-west1-869525608453-ejfnpe5x
  workerConfig:
    diskConfig:
      bootDiskSizeGb: 100
      bootDiskType: pd-standard
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-5-ubu18-20
230909-165100-rc01
    instanceNames:
    - daring-healer-421511-maximalcluster7task2b-w-0
    - daring-healer-421511-maximalcluster7task2b-w-1
    - daring-healer-421511-maximalcluster7task2b-w-2
    - daring-healer-421511-maximalcluster7task2b-w-3
    - daring-healer-421511-maximalcluster7task2b-w-4
    - daring-healer-421511-maximalcluster7task2b-w-5
    - daring-healer-421511-maximalcluster7task2b-w-6
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/zones/us-west1-c/machin
eTypes/n1-standard-1
    minCpuPlatform: AUTOMATIC
    numInstances: 7
    preemptibility: NON_PREEMPTIBLE
labels:
  goog-dataproc-autozone: enabled
  goog-dataproc-cluster-name: daring-healer-421511-maximalcluster7task2b
  goog-dataproc-cluster-uuid: 8abd7648-fc9d-4875-9486-ada5038b61b3
  goog-dataproc-location: us-west1
projectId: daring-healer-421511
status:
  state: RUNNING
  stateStartTime: '2024-05-03T11:39:06.978113Z'
statusHistory:
- state: CREATING
  stateStartTime: '2024-05-03T11:34:20.326499Z'
```

In [ ]:
```
#We submit the job to the cluster
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER --region $REGION ./spark_job_2b.py
%time
```

```
Job [8d81a4061c7c48d394aff80aaa97b886] submitted.
Waiting for job output...
2024-05-03 11:39:24.770395: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dyna
mic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or di
rectory; LD_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-03 11:39:24.770563: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if
you do not have a GPU set up on your machine.
24/05/03 11:39:30 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
24/05/03 11:39:30 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
24/05/03 11:39:30 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
24/05/03 11:39:30 INFO org.spark_project.jetty.util.log: Logging initialized @10831ms to org.spark_project.jett
y.util.log.Slf4jLog
24/05/03 11:39:31 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-SNAPSHOT; built: unknown; git: unknow
```

```
n; jvm 1.8.0_382-b05
24/05/03 11:39:31 INFO org.spark_project.jetty.server.Server: Started @11135ms
24/05/03 11:39:31 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@7d7e2198{HTTP/
1.1, (http/1.1)}{0.0.0.0:40305}
24/05/03 11:39:33 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at daring-healer-42
1511-maximalcluster7task2b-m/10.138.0.47:8032
24/05/03 11:39:33 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at dari
ng-healer-421511-maximalcluster7task2b-m/10.138.0.47:10200
24/05/03 11:39:34 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
24/05/03 11:39:34 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.
24/05/03 11:39:34 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = memory
-mb, units = Mi, type = COUNTABLE
24/05/03 11:39:34 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = vcores
, units = , type = COUNTABLE
24/05/03 11:39:39 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application
_1714736218059_0001
Combined Results:
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 1), Average Images per Second: 1
1.369661144139451
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 2), Average Images per Second: 7
.8505496452628565
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 2), Average Images per Second: 1
7.323960183387715
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 1
7.188889913426568
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 1
7.184699596840275
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 1
6.63440409603418
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 1), Average Images per Second: 3
8.29543201356769
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 2), Average Images per Second: 2
3.905252238936406
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 2), Average Images per Second: 3
9.82397319941481
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 3
4.056861525802994
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 3
4.43764274424747
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 9
.214617020065164
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 1), Average Images per Second: 6
1.179955316878214
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 2), Average Images per Second: 4
9.82233091846418
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 2), Average Images per Second: 5
4.25165562913907
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 5
5.24701143907551
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 5
7.78529632603326
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 4
8.66346443903005
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 1), Average Images per Second:
71.11857677071757
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 2), Average Images per Second:
61.74181285348116
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 2), Average Images per Second:
78.01106657615013
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second:
67.78056944318193
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second:
75.69579498285508
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second:
79.48367779607017
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 1), Average Images per Second: 2
0.401973409798707
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 2), Average Images per Second: 3
7.67209212856444
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 2), Average Images per Second: 3
6.63630216207521
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 3
5.79483043705529
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 3
7.14337330192522
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 3
6.850271114814156
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 1), Average Images per Second: 7
2.94346595479189
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 2), Average Images per Second: 7
1.89911882536714
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 2), Average Images per Second: 7
3.626529843514
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 5
8.701036302364
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 6
3.07010665052686
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 6
6.692480612708
```

```
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 1), Average Images per Second: 109.9235488877694
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 2), Average Images per Second: 103.66651493774651
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 2), Average Images per Second: 117.5995065328324
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 97.45054638672254
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 116.02910814428664
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 93.80164326102704
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 1), Average Images per Second: 151.0340618220119
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 2), Average Images per Second: 156.2462637395898
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 2), Average Images per Second: 146.74174458938512
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second: 109.30903985262331
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second: 150.42929225380263
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second: 145.95218649254278
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 1), Average Images per Second: 6.9876417335559085
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 2), Average Images per Second: 6.543571635657332
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 2), Average Images per Second: 7.5447910267745115
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 7.30519625849185
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 7.38000920234874
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 6.076178514283562
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 1), Average Images per Second: 7.9265932986021435
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 2), Average Images per Second: 8.55937174601895
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 2), Average Images per Second: 8.080641292603397
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 7.711285916325252
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 7.539569535084589
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 7.801167114344657
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 1), Average Images per Second: 8.123648873594158
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 2), Average Images per Second: 7.4021766657734736
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 2), Average Images per Second: 7.598119105662378
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 7.7753561567074065
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 6.667990181916812
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 7.364068832061518
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 1), Average Images per Second: 7.90195553258504
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 2), Average Images per Second: 7.961398683209109
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 2), Average Images per Second: 8.270428402544212
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second: 8.112498439567775
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second: 8.03660792161969
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second: 7.9943956880280105
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 1), Average Images per Second: 8.223015651809085
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 2), Average Images per Second: 7.263332569936329
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 2), Average Images per Second: 7.211954745426906
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 8.30895283878551
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 7.521007175281098
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 7.788392904670593
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 1), Average Images per Second: 7.7819173982622605
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 2), Average Images per Second: 7.715355177154957
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 2), Average Images per Second: 8.52
```

```
4045132435072
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 8.08
0181415244072
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 7.56
6064553118483
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 7.98
7819806843288
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 1), Average Images per Second: 7.98
1031782166987
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 2), Average Images per Second: 8.11
7958423155816
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 2), Average Images per Second: 7.97
0156202345711
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 7.94
1531540401554
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 8.10
40069806282
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 7.85
8210241537205
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 1), Average Images per Second: 7.9
36212787606353
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 2), Average Images per Second: 7.9
046289761598745
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 2), Average Images per Second: 8.0
27201001636847
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second: 7.9
51445479009535
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second: 8.1
80730524417962
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second: 8.5
06152393386104
Average Images per Second By Each Batch Size (Batch_Size: AVG_Images_per_Sec): [(4, 47.23018529843363), (2, 27.
13569634339385)]
Average Images per Second By Each Batch Number (Batch_Number: AVG_Images_per_Sec): [(12, 56.62886134251874), (9
, 44.842599084098005), (6, 31.829199036760468), (3, 16.58136666064826)]
Average Images per Second By Each Repetition (Repetition: AVG_Images_per_Sec): [(1, 39.0461379275475), (2, 36.4
6443658775471), (3, 38.2961960396541)]
Average Images per Second By Each Batch Size (Associativity-method), (Batch_Size: AVG_Images_per_Sec): [(4, 48.
56638437485924), (2, 27.83507904407864)]
Average Images per Second By Each Batch Number (Associativity-method), (Batch_Number: AVG_Images_per_Sec): [(12
, 57.78135044340478), (9, 47.186230283853604), (6, 33.118208895422974), (3, 18.099416014608707)]
Average Images per Second By Repetition (Associativity-method), (Repetition: AVG_Images_per_Sec): [(1, 39.06135
827504427), (2, 38.7484547386544), (3, 37.948242768235396)]
File uploaded successfully
24/05/03 11:52:03 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@7d7e2198{HTTP/1.1, (http
/1.1)}{0.0.0.0:0}
Job [8d81a4061c7c48d394aff80aaa97b886] finished successfully.
done: true
driverControlFilesUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/8abd7648-fc9d-4875-9486
-ada5038b61b3/jobs/8d81a4061c7c48d394aff80aaa97b886/
driverOutputResourceUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/8abd7648-fc9d-4875-94
86-ada5038b61b3/jobs/8d81a4061c7c48d394aff80aaa97b886/driveroutput
jobUuid: 68f7a417-e99e-3728-8216-6f871b03c190
placement:
  clusterName: daring-healer-421511-maximalcluster7task2b
  clusterUuid: 8abd7648-fc9d-4875-9486-ada5038b61b3
pysparkJob:
  mainPythonFileUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/8abd7648-fc9d-4875-9486-a
da5038b61b3/jobs/8d81a4061c7c48d394aff80aaa97b886/staging/spark_job_2b.py
reference:
  jobId: 8d81a4061c7c48d394aff80aaa97b886
  projectId: daring-healer-421511
status:
  state: DONE
  stateStartTime: '2024-05-03T11:52:07.702944Z'
statusHistory:
- state: PENDING
  stateStartTime: '2024-05-03T11:39:16.693589Z'
- state: SETUP_DONE
  stateStartTime: '2024-05-03T11:39:16.724107Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2024-05-03T11:39:17.171750Z'
yarnApplications:
- name: spark_job_2b.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://daring-healer-421511-maximalcluster7task2b-m:8088/proxy/application_1714736218059_0001/
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 8.58 µs
```

In [ ]:
```
CLUSTER = '{}-maximalcluster7task2b'.format(PROJECT)
REGION = 'us-west1'
!gcloud dataproc clusters delete $CLUSTER --region $REGION
#Due to the fact that we have quota related problems as we seem to have few quota available, after creating a c
```

2c) Improve efficiency (6%)

If you implemented a straightfoward version of 2a), you will **probably have an inefficiency** in your code.

Because we are reading multiple times from an RDD to read the values for the different parameters and their averages, caching existing results is important. Explain **where in the process caching can help**, and **add a call to `RDD.cache()`** to your code, if you haven't yet. Measure the the effect of using caching or not using it.

Make the **suitable change** in the code you have written above and mark them up in comments as `### TASK 2c ###`.

Explain in your report what the **reasons for this change** are and **demonstrate and interpret its effect**

In [ ]:
```
#New filename argument

%%writefile spark_job_2c.py
#Task 2a --> vii) Write your code it into a file using the cell magic %%writefile spark_job.py
import os
os.environ['PROTOCOL_BUFFERS_PYTHON_IMPLEMENTATION'] = 'python'
import itertools
import time
import tensorflow as tf
import google.protobuf
import pyspark
from pyspark import SparkContext
#We declare the variables
GCS_PATTERN = 'gs://flowers-public/*/*.jpg' #glob pattern for input files.
PROJECT = 'daring-healer-421511'
BUCKET = 'gs://{}-storage'.format(PROJECT) # This is the bucket storage.
GCS_OUTPUT = BUCKET + '/tfrecords-jpeg-192x192-2/flowers'  # We prefix for output file names.
filenames = tf.io.gfile.glob(GCS_OUTPUT + "*.tfrec")
TARGET_SIZE = [192,192]
PARTITIONS = 16 # This is the number of partitions that will be used later.

#Task 2a) --> i) Combine the previous cells to have the code to create a dataset and create a list of parameter
import itertools
import time
import tensorflow as tf
import pyspark
from pyspark import SparkContext

def read_tfrecord(record): #This function is created to read and parse a TFRecord file
    feature_description = {
        "image": tf.io.FixedLenFeature([], tf.string),
        "class": tf.io.FixedLenFeature([], tf.int64)
    }
    example = tf.io.parse_single_example(record, feature_description)
    image_decoded = tf.image.decode_jpeg(example['image'], channels=3)
    image_reshaped = tf.reshape(image_decoded, [*TARGET_SIZE, 3])
    class_label = example['class']
    return image_reshaped, class_label

def load_dataset(filenames): #This function is created to create a dataset from the TFRecord files
    option_no_order = tf.data.Options()
    option_no_order.experimental_deterministic = False
    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.with_options(option_no_order)
    dataset_new = dataset.map(read_tfrecord)
    return dataset_new

def decode_jpeg_and_label(filepath): #This function is created to decode the JPEG images from the filepath and
    image_data = tf.io.read_file(filepath)
    decoded_image = tf.image.decode_jpeg(image_data)
    label = tf.strings.split(tf.expand_dims(filepath, axis=-1), sep='/')
    class_label = label.values[-2]
    return decoded_image, class_label

def resize_and_crop_image(image, label): #This function is created to resize and crop the images to the specifi
    image_width = tf.shape(image)[0]
    image_height = tf.shape(image)[1]
    target_width = TARGET_SIZE[1]
    target_height = TARGET_SIZE[0]
    resize_crit = (image_width * target_height) / (image_height * target_width)
    image = tf.cond(resize_crit < 1,
                    lambda: tf.image.resize(image, [image_width*target_width/image_width, image_height*target_w
                    lambda: tf.image.resize(image, [image_width*target_height/image_height, image_height*target_
    new_width = tf.shape(image)[0]
    new_height = tf.shape(image)[1]
    image = tf.image.crop_to_bounding_box(image, (new_width - target_width) // 2, (new_height - target_height)
    return image, label

def recompress_image(image, label): #This function then recompresses the processed images to JPEG
    uint8_image = tf.cast(image, tf.uint8)
    jpeg_image = tf.image.encode_jpeg(uint8_image, optimize_size=True, chroma_downsampling=False)
    return jpeg_image, label

def load_image_jpeg_dataset(): #We load the dataset of the images from the JPEG files which are stored in a Goo
    GCS_PATTERN = 'gs://flowers-public/*/*.jpg'
    files_dataset = tf.data.Dataset.list_files(GCS_PATTERN)
```

```
    decoded_images_dataset = files_dataset.map(decode_jpeg_and_label)
    resized_dataset = decoded_images_dataset.map(resize_and_crop_image)
    image_jpeg_dataset = resized_dataset.map(recompress_image)
    return image_jpeg_dataset

def parameter_combinations(batch_sizes, batch_numbers, repetitions): #This function is used to generate all of
    return list(itertools.product(batch_sizes, batch_numbers, repetitions))

#Task 2a) --> ii) Get a Spark context and create the dataset and run timing test for each combination in parall

sc = SparkContext.getOrCreate() #We set up the spark context and then define the parameter combinations

#These are the batch sizes, batch numbers and repetitions that we will be testing
batch_sizes = [2, 4]
batch_numbers = [3, 6, 9, 12]
repetitions = [1, 2, 3]

##TASK 2C --> We add the cache function to the RDD
parameters_rdd = sc.parallelize(parameter_combinations(batch_sizes, batch_numbers, repetitions)).cache()

#This function is created to time the processing of the TFRecord batches
def time_configs(params):
    batch_size, batch_number, repetition = params
    TRF_dataset = load_dataset(filenames)
    results = []
    for repet in range(repetition):
        start_time = time.time()
        for _ in TRF_dataset.batch(batch_size).take(batch_number):
            pass
        total_time = time.time() - start_time
        images_per_second = (batch_size * batch_number) / total_time
        results.append(('TFRecord', params, images_per_second))
    return results

#This function is created to time the processing of the image data batches
def image_configs(params):
    batch_size, batch_number, repetition = params
    img_dataset = load_image_jpeg_dataset()
    image_processing_results = []
    for rep in range(repetition):
        start_time = time.time()
        for _ in img_dataset.batch(batch_size).take(batch_number):
            pass
        total_time = time.time() - start_time
        images_per_second = (batch_size * batch_number) / total_time
        image_processing_results.append(('Image', params, images_per_second))
    return image_processing_results

#These are the results from the time processing of the TFRecord and image data batches
resulting_rdd = parameters_rdd.flatMap(time_configs)
resulting_rdd_2 = parameters_rdd.flatMap(image_configs)

#Task 2a) iii) --> Transform the resulting RDD to the structure ( parameter_combination, images_per_second ) an

# We merge the 2 resulting RDDs (TFRecord and image results) into a combined RDD using the .union function
results_rdd_combined = resulting_rdd.union(resulting_rdd_2)
final_results = results_rdd_combined.collect()

#We print the results
print('Combined Results:')
for result in final_results:
  print(f'Dataset: {result[0]}, Parameters: (Batch Size, Batch Number, Repetitions): {result[1]}, Average Image

#Task 2a --> iv) Create an RDD with all results for each parameter as (parameter_value,images_per_second) and c

#This function extracts the results for each parameter
def param_results(index_param):
  return results_rdd_combined.map(lambda x: (x[1][index_param], x[2]))

#We create separate RDDs for each parameter
rdd_batch_size = param_results(0)
rdd_batch_number = param_results(1)
rdd_repetition = param_results(2)

#This function is created to calclate the average images per second for each parameter separately
def average_images_param(data):
  total_images = sum(images for images in data)
  count= len(data)
  return total_images / count

avg_batch_size_rdd = rdd_batch_size.groupByKey().mapValues(average_images_param).collect()
avg_batch_number_rdd = rdd_batch_number.groupByKey().mapValues(average_images_param).collect()
avg_repetition_rdd = rdd_repetition.groupByKey().mapValues(average_images_param).collect()

#We print the average images per second for each unique parameter
print("Average Images per Second By Each Batch Size (Batch_Size: AVG_Images_per_Sec):", avg_batch_size_rdd)
print("Average Images per Second By Each Batch Number (Batch_Number: AVG_Images_per_Sec):", avg_batch_number_rd
print("Average Images per Second By Each Repetition (Repetition: AVG_Images_per_Sec):", avg_repetition_rdd)
```

```
#Task 2a --> v)  Create an RDD with the average reading speeds for each parameter value and collect the results

#This function calculates the average reading speed using a tuple (total, count) for the associativity
def seq_op(accumu, n):
  return (accumu[0] + n, accumu[1] +1)

def comb_op( accumu1, accumu2):
  return (accumu1[0] + accumu2[0], accumu1[1] + accumu2[1])

#Now we can create a function that calculates the average images per second while also keeping associativity in
def average_assoc(rdd_params):
  return rdd_params.aggregateByKey((0,0), seq_op, comb_op).mapValues(lambda x: x[0] / x[1]).collect()

avg_batch_size = average_assoc(rdd_batch_size)
avg_batch_number = average_assoc(rdd_batch_number)
avg_repetition = average_assoc(rdd_repetition)

#Now we print the average reading speed for the associativity-respecting method
print("Average Images per Second By Each Batch Size (Associativity-method), (Batch_Size: AVG_Images_per_Sec):",
print("Average Images per Second By Each Batch Number (Associativity-method), (Batch_Number: AVG_Images_per_Sec
print("Average Images per Second By Repetition (Associativity-method), (Repetition: AVG_Images_per_Sec):", avg_

#Task 2a --> vi) Write the results to a pickle file in your bucket
import pickle
from google.cloud import storage

#We store the results from previous tasks to a dictionary
results_task2c = {'Final Results Task iii': final_results,
                  'AVG Batch Size Task iv': avg_batch_size_rdd,
                  'AVG Batch Number Task iv': avg_batch_number_rdd,
                  'AVG Repetition Task iv': avg_repetition_rdd,
                  'AVG Batch Size Task v': avg_batch_size,
                  'AVG Batch Number Task v': avg_batch_number,
                  'AVG Repetition Task v': avg_repetition}

#We write the results to a local pickle file
with open('/tmp/all_results.pkl', 'wb') as f:
    pickle.dump(results_task2c, f)

#This function is used to upload the pickle file to the bucket that we specify
def upload_to_bucket(blob_name, path_to_file, bucket_name):
    try:
        client = storage.Client()
        bucket = client.get_bucket(bucket_name)
        blob = bucket.blob(blob_name)
        blob.upload_from_filename(path_to_file)
        print("File uploaded successfully")
    except Exception as e:
        print(f"An error occurred: {e}")

upload_to_bucket('results/results_task2c.pkl', '/tmp/all_results.pkl', 'daring-healer-421511-storage')
```

```
Writing spark_job_2c.py
```

In [ ]:

```
#We will set up the maximal cluster which is a cluster with 1 machine and 7 workers using maximal SSD size (100
CLUSTER = '{}-maximalcluster7task2c'.format(PROJECT)
REGION = 'us-west1'
PIP_PACKAGES = 'tensorflow==2.4.0'
!gcloud dataproc clusters create $CLUSTER \
    --region $REGION \
    --bucket $PROJECT-storage \
    --image-version 1.5-ubuntu18 \
    --master-machine-type n1-standard-1 \
    --master-boot-disk-type pd-ssd --master-boot-disk-size 100\
    --num-workers 7 --worker-machine-type n1-standard-1 --worker-boot-disk-size 100 \
    --initialization-actions gs://goog-dataproc-initialization-actions-$REGION/python/pip-install.sh \
    --metadata PIP_PACKAGES='tensorflow==2.4.0 matplotlib google-cloud-storage==1.42.0 google-api-core==1.31.0
#Reference link for code: https://cloud.google.com/dataproc/docs/guides/create-cluster
```

```
Waiting on operation [projects/daring-healer-421511/regions/us-west1/operations/de2f5a46-f3e0-3ee6-b3dc-0387d83
1ed04].

WARNING: Creating clusters using the n1-standard-1 machine type is not recommended. Consider using a machine ty
pe with higher memory.
WARNING: Don't create production clusters that reference initialization actions located in the gs://goog-datapr
oc-initialization-actions-REGION public buckets. These scripts are provided as reference implementations, and t
hey are synchronized with ongoing GitHub repository changes—a new version of a initialization action in public
buckets may break your cluster creation. Instead, copy the following initialization actions from public buckets
into your bucket : gs://goog-dataproc-initialization-actions-us-west1/python/pip-install.sh
WARNING: Failed to validate permissions required for default service account: '869525608453-compute@developer.g
serviceaccount.com'. Cluster creation could still be successful if required permissions have been granted to th
e respective service accounts as mentioned in the document https://cloud.google.com/dataproc/docs/concepts/conf
iguring-clusters/service-accounts#dataproc_service_accounts_2. This could be due to Cloud Resource Manager API
hasn't been enabled in your project '869525608453' before or it is disabled. Enable it by visiting 'https://con
sole.developers.google.com/apis/api/cloudresourcemanager.googleapis.com/overview?project=869525608453'.
WARNING: For PD-Standard without local SSDs, we strongly recommend provisioning 1TB or larger to ensure consist
ently high I/O performance. See https://cloud.google.com/compute/docs/disks/performance for information on disk
I/O performance.
WARNING: The firewall rules for specified network or subnetwork would allow ingress traffic from 0.0.0.0/0, whi
ch could be a security risk.
WARNING: The specified custom staging bucket 'daring-healer-421511-storage' is not using uniform bucket level a
ccess IAM configuration. It is recommended to update bucket to enable the same. See https://cloud.google.com/st
orage/docs/uniform-bucket-level-access.
Created [https://dataproc.googleapis.com/v1/projects/daring-healer-421511/regions/us-west1/clusters/daring-heal
er-421511-maximalcluster7task2c] Cluster placed in zone [us-west1-c].
```

In [ ]:
```
#We get information on the single machine cluster we created
!gcloud dataproc clusters describe $CLUSTER --region $REGION
```

```
clusterName: daring-healer-421511-maximalcluster7task2c
clusterUuid: e629e54c-5347-4af4-ab88-888fe4773d32
config:
  configBucket: daring-healer-421511-storage
  endpointConfig: {}
  gceClusterConfig:
    internalIpOnly: false
    metadata:
      PIP_PACKAGES: tensorflow==2.4.0 matplotlib google-cloud-storage==1.42.0 google-api-core==1.31.0
        protobuf==3.17.3
    networkUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/global/networks/default
    serviceAccountScopes:
    - https://www.googleapis.com/auth/bigquery
    - https://www.googleapis.com/auth/bigtable.admin.table
    - https://www.googleapis.com/auth/bigtable.data
    - https://www.googleapis.com/auth/cloud.useraccounts.readonly
    - https://www.googleapis.com/auth/devstorage.full_control
    - https://www.googleapis.com/auth/devstorage.read_write
    - https://www.googleapis.com/auth/logging.write
    - https://www.googleapis.com/auth/monitoring.write
    zoneUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/zones/us-west1-c
  initializationActions:
  - executableFile: gs://goog-dataproc-initialization-actions-us-west1/python/pip-install.sh
    executionTimeout: 600s
  masterConfig:
    diskConfig:
      bootDiskSizeGb: 100
      bootDiskType: pd-ssd
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-5-ubu18-20
230909-165100-rc01
    instanceNames:
    - daring-healer-421511-maximalcluster7task2c-m
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/zones/us-west1-c/machin
eTypes/n1-standard-1
    minCpuPlatform: AUTOMATIC
    numInstances: 1
    preemptibility: NON_PREEMPTIBLE
  softwareConfig:
    imageVersion: 1.5.90-ubuntu18
    properties:
      capacity-scheduler:yarn.scheduler.capacity.root.default.ordering-policy: fair
      core:fs.gs.block.size: '134217728'
      core:fs.gs.metadata.cache.enable: 'false'
      core:hadoop.ssl.enabled.protocols: TLSv1,TLSv1.1,TLSv1.2
      distcp:mapreduce.map.java.opts: -Xmx576m
      distcp:mapreduce.map.memory.mb: '768'
      distcp:mapreduce.reduce.java.opts: -Xmx576m
      distcp:mapreduce.reduce.memory.mb: '768'
      hdfs:dfs.datanode.address: 0.0.0.0:9866
      hdfs:dfs.datanode.http.address: 0.0.0.0:9864
      hdfs:dfs.datanode.https.address: 0.0.0.0:9865
      hdfs:dfs.datanode.ipc.address: 0.0.0.0:9867
      hdfs:dfs.namenode.handler.count: '60'
      hdfs:dfs.namenode.http-address: 0.0.0.0:9870
      hdfs:dfs.namenode.https-address: 0.0.0.0:9871
      hdfs:dfs.namenode.lifeline.rpc-address: daring-healer-421511-maximalcluster7task2c-m:8050
      hdfs:dfs.namenode.secondary.http-address: 0.0.0.0:9868
      hdfs:dfs.namenode.secondary.https-address: 0.0.0.0:9869
      hdfs:dfs.namenode.service.handler.count: '30'
```

```
      hdfs:dfs.namenode.servicerpc-address: daring-healer-421511-maximalcluster7task2c-m:8051
      hive:hive.fetch.task.conversion: none
      mapred-env:HADOOP_JOB_HISTORYSERVER_HEAPSIZE: '1000'
      mapred:mapreduce.job.maps: '60'
      mapred:mapreduce.job.reduce.slowstart.completedmaps: '0.95'
      mapred:mapreduce.job.reduces: '7'
      mapred:mapreduce.jobhistory.recovery.store.class: org.apache.hadoop.mapreduce.v2.hs.HistoryServerLeveldbS
tateStoreService
      mapred:mapreduce.map.cpu.vcores: '1'
      mapred:mapreduce.map.java.opts: -Xmx819m
      mapred:mapreduce.map.memory.mb: '1024'
      mapred:mapreduce.reduce.cpu.vcores: '1'
      mapred:mapreduce.reduce.java.opts: -Xmx1638m
      mapred:mapreduce.reduce.memory.mb: '2048'
      mapred:mapreduce.task.io.sort.mb: '256'
      mapred:yarn.app.mapreduce.am.command-opts: -Xmx819m
      mapred:yarn.app.mapreduce.am.resource.cpu-vcores: '1'
      mapred:yarn.app.mapreduce.am.resource.mb: '1024'
      spark-env:SPARK_DAEMON_MEMORY: 1000m
      spark:spark.driver.maxResultSize: 480m
      spark:spark.driver.memory: 960m
      spark:spark.executor.cores: '1'
      spark:spark.executor.instances: '2'
      spark:spark.executor.memory: 2688m
      spark:spark.executorEnv.OPENBLAS_NUM_THREADS: '1'
      spark:spark.extraListeners: com.google.cloud.spark.performance.DataprocMetricsListener
      spark:spark.scheduler.mode: FAIR
      spark:spark.sql.cbo.enabled: 'true'
      spark:spark.ui.port: '0'
      spark:spark.yarn.am.memory: 640m
      yarn-env:YARN_NODEMANAGER_HEAPSIZE: '1000'
      yarn-env:YARN_RESOURCEMANAGER_HEAPSIZE: '1000'
      yarn-env:YARN_TIMELINESERVER_HEAPSIZE: '1000'
      yarn:yarn.nodemanager.address: 0.0.0.0:8026
      yarn:yarn.nodemanager.resource.cpu-vcores: '1'
      yarn:yarn.nodemanager.resource.memory-mb: '3072'
      yarn:yarn.resourcemanager.nodemanager-graceful-decommission-timeout-secs: '86400'
      yarn:yarn.scheduler.maximum-allocation-mb: '3072'
      yarn:yarn.scheduler.minimum-allocation-mb: '256'
  tempBucket: dataproc-temp-us-west1-869525608453-ejfnpe5x
  workerConfig:
    diskConfig:
      bootDiskSizeGb: 100
      bootDiskType: pd-standard
    imageUri: https://www.googleapis.com/compute/v1/projects/cloud-dataproc/global/images/dataproc-1-5-ubu18-20
230909-165100-rc01
    instanceNames:
    - daring-healer-421511-maximalcluster7task2c-w-0
    - daring-healer-421511-maximalcluster7task2c-w-1
    - daring-healer-421511-maximalcluster7task2c-w-2
    - daring-healer-421511-maximalcluster7task2c-w-3
    - daring-healer-421511-maximalcluster7task2c-w-4
    - daring-healer-421511-maximalcluster7task2c-w-5
    - daring-healer-421511-maximalcluster7task2c-w-6
    machineTypeUri: https://www.googleapis.com/compute/v1/projects/daring-healer-421511/zones/us-west1-c/machin
eTypes/n1-standard-1
    minCpuPlatform: AUTOMATIC
    numInstances: 7
    preemptibility: NON_PREEMPTIBLE
labels:
  goog-dataproc-autozone: enabled
  goog-dataproc-cluster-name: daring-healer-421511-maximalcluster7task2c
  goog-dataproc-cluster-uuid: e629e54c-5347-4af4-ab88-888fe4773d32
  goog-dataproc-location: us-west1
projectId: daring-healer-421511
status:
  state: RUNNING
  stateStartTime: '2024-05-03T12:26:43.751203Z'
statusHistory:
- state: CREATING
  stateStartTime: '2024-05-03T12:21:01.701269Z'
```

In [ ]:

```
#We submit the job to the cluster
!gcloud dataproc jobs submit pyspark --cluster $CLUSTER --region $REGION ./spark_job_2c.py
%time
```

```
Job [69593d22d7bd43898ae6b01d05cf64f2] submitted.
Waiting for job output...
2024-05-03 12:27:00.352146: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dyna
mic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or di
rectory; LD_LIBRARY_PATH: :/usr/lib/hadoop/lib/native
2024-05-03 12:27:00.352390: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if
you do not have a GPU set up on your machine.
24/05/03 12:27:07 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
24/05/03 12:27:07 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
24/05/03 12:27:07 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
24/05/03 12:27:08 INFO org.spark_project.jetty.util.log: Logging initialized @13032ms to org.spark_project.jett
y.util.log.Slf4jLog
24/05/03 12:27:08 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-SNAPSHOT; built: unknown; git: unknow
```

```
n; jvm 1.8.0_382-b05
24/05/03 12:27:08 INFO org.spark_project.jetty.server.Server: Started @13391ms
24/05/03 12:27:08 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@52e43b3d{HTTP/
1.1, (http/1.1)}{0.0.0.0:40269}
24/05/03 12:27:11 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at daring-healer-42
1511-maximalcluster7task2c-m/10.138.0.61:8032
24/05/03 12:27:11 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at dari
ng-healer-421511-maximalcluster7task2c-m/10.138.0.61:10200
24/05/03 12:27:12 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
24/05/03 12:27:12 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.
24/05/03 12:27:12 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = memory
-mb, units = Mi, type = COUNTABLE
24/05/03 12:27:12 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = vcores
, units = , type = COUNTABLE
24/05/03 12:27:14 WARN org.apache.hadoop.hdfs.DataStreamer: Caught exception
java.lang.InterruptedException
        at java.lang.Object.wait(Native Method)
        at java.lang.Thread.join(Thread.java:1257)
        at java.lang.Thread.join(Thread.java:1331)
        at org.apache.hadoop.hdfs.DataStreamer.closeResponder(DataStreamer.java:980)
        at org.apache.hadoop.hdfs.DataStreamer.endBlock(DataStreamer.java:630)
        at org.apache.hadoop.hdfs.DataStreamer.run(DataStreamer.java:807)
24/05/03 12:27:17 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application
_1714739042107_0001
Combined Results:
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 1), Average Images per Second: 1
7.953838778165878
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 2), Average Images per Second: 1
8.369925259882315
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 2), Average Images per Second: 2
0.27824074596342
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 1
9.35515488616096
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 1
7.48685592942393
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 1
9.704195411613547
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 1), Average Images per Second: 3
7.18915084720215
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 2), Average Images per Second: 3
7.84741366928925
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 2), Average Images per Second: 3
4.96822046556997
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 4
0.629882207067574
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 3
5.216830756937654
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 3
8.56707140459847
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 1), Average Images per Second: 4
7.97504959705964
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 2), Average Images per Second: 5
5.96924331497283
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 2), Average Images per Second: 6
0.37435915070024
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 2
3.752435335368876
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 5
8.262224286977684
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 5
1.887690659284395
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 1), Average Images per Second:
62.34862928555369
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 2), Average Images per Second:
81.52854620555601
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 2), Average Images per Second:
76.81682008923768
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second:
77.89962374769583
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second:
71.26241777410284
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second:
81.78887688195195
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 1), Average Images per Second: 3
2.09494640708425
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 2), Average Images per Second: 3
7.04462874151012
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 2), Average Images per Second: 3
1.921356673488763
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 4
0.088799398488575
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 3
4.3754318985094
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 3
7.726562213059275
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 1), Average Images per Second: 7
0.82470579792331
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 2), Average Images per Second: 7
8.82591601000131
```

```
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 2), Average Images per Second: 7
3.55718068781677
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 7
3.83195236066014
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 6
4.7188819803098
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 7
7.03792229419471
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 1), Average Images per Second: 8
7.67362950206098
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 2), Average Images per Second: 1
22.97907982505437
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 2), Average Images per Second: 1
09.3887240356083
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 9
8.46721453659516
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 1
15.09173270688812
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 9
9.85797471987577
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 1), Average Images per Second:
133.80347244929698
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 2), Average Images per Second:
140.67086317740004
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 2), Average Images per Second:
139.4111656848699
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second:
136.45633973886262
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second:
125.38057100722166
Dataset: TFRecord, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second:
144.78969216297153
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 1), Average Images per Second: 4.06
4098414980844
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 2), Average Images per Second: 7.81
5475719091838
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 2), Average Images per Second: 8.50
1094482188316
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 8.12
7611432476504
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 7.92
70627012806365
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 3, 3), Average Images per Second: 7.74
3485331245086
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 1), Average Images per Second: 8.31
7956662435384
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 2), Average Images per Second: 8.39
149607323187
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 2), Average Images per Second: 8.91
3268221794803
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 8.39
626679469456
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 8.70
0475079132318
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 6, 3), Average Images per Second: 8.68
4269753263008
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 1), Average Images per Second: 8.04
3541842414175
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 2), Average Images per Second: 8.48
4966103392155
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 2), Average Images per Second: 8.24
7591595302582
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 8.49
7821051957418
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 6.82
9483878169507
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 9, 3), Average Images per Second: 8.51
9150732970521
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 1), Average Images per Second: 8.0
66017058113966
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 2), Average Images per Second: 8.2
92142739464232
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 2), Average Images per Second: 9.3
50642633595742
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second: 9.0
19686124125132
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second: 8.6
63754400158362
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (2, 12, 3), Average Images per Second: 8.3
21030140551866
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 1), Average Images per Second: 7.99
7422095615084
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 2), Average Images per Second: 8.43
0377711434254
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 2), Average Images per Second: 9.23
6254479198188
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 8.32
24005296531
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 9.12
```

```
8501993495815
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 3, 3), Average Images per Second: 9.08
1366414911935
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 1), Average Images per Second: 9.23
8312577331396
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 2), Average Images per Second: 9.41
9957547009579
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 2), Average Images per Second: 9.37
4353276013142
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 8.95
8867542238396
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 9.89
7539923408175
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 6, 3), Average Images per Second: 9.22
5686013058693
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 1), Average Images per Second: 8.92
81531597319
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 2), Average Images per Second: 8.83
297536160838
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 2), Average Images per Second: 9.62
8148701342425
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 9.31
1832188998288
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 9.76
4547156549492
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 9, 3), Average Images per Second: 9.16
865751604007
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 1), Average Images per Second: 9.6
24447629435906
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 2), Average Images per Second: 9.6
36691213608945
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 2), Average Images per Second: 9.8
95435430121797
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second: 9.9
55975575525757
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second: 9.8
23049979595346
Dataset: Image, Parameters: (Batch Size, Batch Number, Repetitions): (4, 12, 3), Average Images per Second: 9.4
7422823238864
Average Images per Second By Each Batch Size (Batch_Size: AVG_Images_per_Sec): [(4, 50.779586230278774), (2, 28
.62060671733013)]
Average Images per Second By Each Batch Number (Batch_Number: AVG_Images_per_Sec): [(12, 58.10352390521206), (9
, 46.57077368978701), (6, 33.20917095792271), (3, 18.672581908502966)]
Average Images per Second By Each Repetition (Repetition: AVG_Images_per_Sec): [(1, 39.902618464093095), (2, 38
.518019118374426), (3, 39.9435729524705)]
Average Images per Second By Each Batch Size (Associativity-method), (Batch_Size: AVG_Images_per_Sec): [(4, 48.
79294820323631), (2, 28.13458388555063)]
Average Images per Second By Each Batch Number (Associativity-method), (Batch_Number: AVG_Images_per_Sec): [(12
, 56.804326011146564), (9, 44.49631314070965), (6, 31.256141239586494), (3, 18.08197688324735)]
Average Images per Second By Repetition (Associativity-method), (Repetition: AVG_Images_per_Sec): [(1, 37.88179
564401834), (2, 37.09477353654646), (3, 39.47990942038509)]
File uploaded successfully
24/05/03 12:38:21 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@52e43b3d{HTTP/1.1, (http
/1.1)}{0.0.0.0:0}
Job [69593d22d7bd43898ae6b01d05cf64f2] finished successfully.
done: true
driverControlFilesUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/e629e54c-5347-4af4-ab88
-888fe4773d32/jobs/69593d22d7bd43898ae6b01d05cf64f2/
driverOutputResourceUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/e629e54c-5347-4af4-ab
88-888fe4773d32/jobs/69593d22d7bd43898ae6b01d05cf64f2/driveroutput
jobUuid: 215ffd5d-4bc0-381a-a2d5-c18c0e27bd29
placement:
  clusterName: daring-healer-421511-maximalcluster7task2c
  clusterUuid: e629e54c-5347-4af4-ab88-888fe4773d32
pysparkJob:
  mainPythonFileUri: gs://daring-healer-421511-storage/google-cloud-dataproc-metainfo/e629e54c-5347-4af4-ab88-8
88fe4773d32/jobs/69593d22d7bd43898ae6b01d05cf64f2/staging/spark_job_2c.py
reference:
  jobId: 69593d22d7bd43898ae6b01d05cf64f2
  projectId: daring-healer-421511
status:
  state: DONE
  stateStartTime: '2024-05-03T12:38:24.247580Z'
statusHistory:
- state: PENDING
  stateStartTime: '2024-05-03T12:26:51.422476Z'
- state: SETUP_DONE
  stateStartTime: '2024-05-03T12:26:51.455865Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2024-05-03T12:26:51.874547Z'
yarnApplications:
- name: spark_job_2c.py
  progress: 1.0
  state: FINISHED
  trackingUrl: http://daring-healer-421511-maximalcluster7task2c-m:8088/proxy/application_1714739042107_0001/
CPU times: user 82 µs, sys: 0 ns, total: 82 µs
Wall time: 181 µs
```

In [ ]:
```
CLUSTER = '{}-maximalcluster7task2c'.format(PROJECT)
REGION = 'us-west1'
!gcloud dataproc clusters delete $CLUSTER --region $REGION
#Due to the fact that we have quota related problems as we seem to have few quota available, after creating a c
```

2d) Retrieve, analyse and discuss the output (12%)

Run the tests over a wide range of different paramters and list the results in a table.

Perform a **linear regression** (e.g. using scikit-learn) over **the values for each parameter** and for the **two cases** (reading from image files/reading TFRecord files). List a **table** with the output and interpret the results in terms of the effects of overall.
Also, **plot** the output values, the averages per parameter value and the regression lines for each parameter and for the product of batch_size and batch_number

Discuss the **implications** of this result for **applications** like large-scale machine learning. Keep in mind that cloud data may be stored in distant physical locations. Use the numbers provided in the PDF latency-numbers document available on Moodle or here for your arguments.

How is the **observed** behaviour **similar or different** from what you'd expect from a **single machine**? Why would cloud providers tie throughput to capacity of disk resources?

By **parallelising** the speed test we are making **assumptions** about the limits of the bucket reading speeds. See here for more information. Discuss, **what we need to consider** in **speed tests** in parallel on the cloud, which bottlenecks we might be identifying, and how this relates to your results.

Discuss to what extent **linear modelling** reflects the **effects** we are observing. Discuss what could be expected from a theoretical perspective and what can be useful in practice.

Write your **code below** and **include the output** in your submitted `ipynb` file. Provide the answer **text in your report**.

In [ ]:
```
!gsutil cp gs://daring-healer-421511-storage/results/results_task2c.pkl ./results_task2c.pkl #We first read the
```

```
Copying gs://daring-healer-421511-storage/results/results_task2c.pkl...
/ [1 files][  2.5 KiB/  2.5 KiB]
Operation completed over 1 objects/2.5 KiB.
```

In [ ]:
```
import pandas as pd

#We load the pickle file
final_res = pd.read_pickle('results_task2c.pkl')
print(final_res)
```

{'Final Results Task iii': [('TFRecord', (2, 3, 1), 17.953838778165878), ('TFRecord', (2, 3, 2), 18.369925259882315), ('TFRecord', (2, 3, 2), 20.27824074596342), ('TFRecord', (2, 3, 3), 19.35515488616096), ('TFRecord', (2, 3, 3), 17.48685592942393), ('TFRecord', (2, 3, 3), 19.704195411613547), ('TFRecord', (2, 6, 1), 37.18915084720215), ('TFRecord', (2, 6, 2), 37.84741366928925), ('TFRecord', (2, 6, 2), 34.96822046556997), ('TFRecord', (2, 6, 3), 40.629882207067574), ('TFRecord', (2, 6, 3), 35.216830756937654), ('TFRecord', (2, 6, 3), 38.56707140459847), ('TFRecord', (2, 9, 1), 47.97504959705964), ('TFRecord', (2, 9, 2), 55.96924331497283), ('TFRecord', (2, 9, 2), 60.37435915070024), ('TFRecord', (2, 9, 3), 23.752435335368876), ('TFRecord', (2, 9, 3), 58.262224286977684), ('TFRecord', (2, 9, 3), 51.887690659284395), ('TFRecord', (2, 12, 1), 62.34862928555369), ('TFRecord', (2, 12, 2), 81.52854620555601), ('TFRecord', (2, 12, 2), 76.81682008923768), ('TFRecord', (2, 12, 3), 77.899623747695583), ('TFRecord', (2, 12, 3), 71.26241777410284), ('TFRecord', (2, 12, 3), 81.78887688195195), ('TFRecord', (4, 3, 1), 32.09494640708425), ('TFRecord', (4, 3, 2), 37.04462874151012), ('TFRecord', (4, 3, 2), 31.921356673488763), ('TFRecord', (4, 3, 3), 40.088799398488575), ('TFRecord', (4, 3, 3), 34.3754318985094), ('TFRecord', (4, 3, 3), 37.726562213059275), ('TFRecord', (4, 6, 1), 70.82470579792331), ('TFRecord', (4, 6, 2), 78.825916010000131), ('TFRecord', (4, 6, 2), 73.55718068781677), ('TFRecord', (4, 6, 3), 73.83195236066014), ('TFRecord', (4, 6, 3), 64.7188819803098), ('TFRecord', (4, 6, 3), 77.03792229419471), ('TFRecord', (4, 9, 1), 87.67362950206098), ('TFRecord', (4, 9, 2), 122.97907982505437), ('TFRecord', (4, 9, 2), 109.3887240356083), ('TFRecord', (4, 9, 3), 98.46721453659516), ('TFRecord', (4, 9, 3), 115.09173270688812), ('TFRecord', (4, 9, 3), 99.85797471987577), ('TFRecord', (4, 12, 1), 133.80347244929698), ('TFRecord', (4, 12, 2), 140.67086317740004), ('TFRecord', (4, 12, 2), 139.4111656848699), ('TFRecord', (4, 12, 3), 136.45633973886262), ('TFRecord', (4, 12, 3), 125.38057100722166), ('TFRecord', (4, 12, 3), 144.78969216297153), ('Image', (2, 3, 1), 4.064098414980844), ('Image', (2, 3, 2), 7.815475719091838), ('Image', (2, 3, 2), 8.501094482188316), ('Image', (2, 3, 3), 8.127611432476504), ('Image', (2, 3, 3), 7.9270627012806365), ('Image', (2, 3, 3), 7.743485331245086), ('Image', (2, 6, 1), 8.317956662435384), ('Image', (2, 6, 2), 8.39149607323187), ('Image', (2, 6, 2), 8.913268221794803), ('Image', (2, 6, 3), 8.39626679469456), ('Image', (2, 6, 3), 8.700475079132318), ('Image', (2, 6, 3), 8.684269753263008), ('Image', (2, 9, 1), 8.043541842414175), ('Image', (2, 9, 2), 8.484966103392155), ('Image', (2, 9, 2), 8.247591595302582), ('Image', (2, 9, 3), 8.497821051957418), ('Image', (2, 9, 3), 6.829483878169507), ('Image', (2, 9, 3), 8.519150732970521), ('Image', (2, 12, 1), 8.066017058113966), ('Image', (2, 12, 2), 8.292142739464232), ('Image', (2, 12, 2), 9.350642633595742), ('Image', (2, 12, 3), 9.019686124125132), ('Image', (2, 12, 3), 8.663754400158362), ('Image', (2, 12, 3), 8.321030140551866), ('Image', (4, 3, 1), 7.997422095615084), ('Image', (4, 3, 2), 8.430377711434254), ('Image', (4, 3, 2), 9.236254479198188), ('Image', (4, 3, 3), 8.3224005296531), ('Image', (4, 3, 3), 9.128501993495815), ('Image', (4, 3, 3), 9.081366414911935), ('Image', (4, 6, 1), 9.238312577331396), ('Image', (4, 6, 2), 9.419957547009579), ('Image', (4, 6, 2), 9.374353276013142), ('Image', (4, 6, 3), 8.958867542238396), ('Image', (4, 6, 3), 9.897539923408175), ('Image', (4, 6, 3), 9.225686013058693), ('Image', (4, 9, 1), 8.9281531597319), ('Image', (4, 9, 2), 8.83297536160838), ('Image', (4, 9, 2), 9.628148701342425), ('Image', (4, 9, 3), 9.311832188998288), ('Image', (4, 9, 3), 9.764547156549492), ('Image', (4, 9, 3), 9.16865751604007), ('Image', (4, 12, 1), 9.624447629435906), ('Image', (4, 12, 2), 9.636691213608945), ('Image', (4, 12, 2), 9.895435430121797), ('Image', (4, 12, 3), 9.955975575525757), ('Image', (4, 12, 3), 9.823049979595346), ('Image', (4, 12, 3), 9.47422823238864)], 'AVG Batch Size Task iv': [(4, 50.779586230278774), (2, 28.62060671733013)], 'AVG Batch Number Task iv': [(12, 58.10352390521206), (9, 46.57077368978701), (6, 33.20917095792271), (3, 18.672581908502966)], 'AVG Repetition Task iv': [(1, 39.902618464093095), (2, 38.5180191183744 26), (3, 39.9435729524705)], 'AVG Batch Size Task v': [(4, 48.79294820323631), (2, 28.13458388555063)], 'AVG Batch Number Task v': [(12, 56.804326011146564), (9, 44.49631314070965), (6, 31.256141239586494), (3, 18.08197688324735)], 'AVG Repetition Task v': [(1, 37.88179564401834), (2, 37.09477353654646), (3, 39.47990942038509)]}

In [ ]:

```
#We store the results into variables to then put them in dataframes
final_res_task_iii = final_res['Final Results Task iii']
avg_batch_size_task_iv = final_res['AVG Batch Size Task iv']
avg_batch_number_task_iv = final_res['AVG Batch Number Task iv']
avg_repetition_task_iv = final_res['AVG Repetition Task iv']
avg_batch_size_task_v = final_res['AVG Batch Size Task v']
avg_batch_number_task_v = final_res['AVG Batch Number Task v']
avg_repetition_task_v = final_res['AVG Repetition Task v']

#We print out all of the results we got
print('Final Results')
print(final_res_task_iii)
print("Average Images per Second By Each Batch Size (Batch_Size: AVG_Images_per_Sec)")
print(avg_batch_size_task_iv)
print("Average Images per Second By Each Batch Number (Batch_Number: AVG_Images_per_Sec)")
print(avg_batch_number_task_iv)
print("Average Images per Second By Each Repetition (Repetition: AVG_Images_per_Sec)")
print(avg_repetition_task_iv)
print("Average Images per Second By Each Batch Size (Associativity-method), (Batch_Size: AVG_Images_per_Sec)")
print(avg_batch_size_task_v)
print("Average Images per Second By Each Batch Number (Associativity-method), (Batch_Number: AVG_Images_per_Sec
print(avg_batch_number_task_v)
print("Average Images per Second By Repetition (Associativity-method), (Repetition: AVG_Images_per_Sec)")
print(avg_repetition_task_v)
```

```
Final Results
[('TFRecord', (2, 3, 1), 17.953838778165878), ('TFRecord', (2, 3, 2), 18.369925259882315), ('TFRecord', (2, 3,
2), 20.27824074596342), ('TFRecord', (2, 3, 3), 19.35515488616096), ('TFRecord', (2, 3, 3), 17.48685592942393),
('TFRecord', (2, 3, 3), 19.704195411613547), ('TFRecord', (2, 6, 1), 37.18915084720215), ('TFRecord', (2, 6, 2)
, 37.84741366928925), ('TFRecord', (2, 6, 2), 34.96822046556997), ('TFRecord', (2, 6, 3), 40.629882207067574),
('TFRecord', (2, 6, 3), 35.216830756937654), ('TFRecord', (2, 6, 3), 38.56707140459847), ('TFRecord', (2, 9, 1)
, 47.97504959705964), ('TFRecord', (2, 9, 2), 55.96924331497283), ('TFRecord', (2, 9, 2), 60.37435915070024), (
'TFRecord', (2, 9, 3), 23.752435335368876), ('TFRecord', (2, 9, 3), 58.262224286977684), ('TFRecord', (2, 9, 3)
, 51.887690659284395), ('TFRecord', (2, 12, 1), 62.34862928555369), ('TFRecord', (2, 12, 2), 81.52854620555601)
, ('TFRecord', (2, 12, 2), 76.81682008923768), ('TFRecord', (2, 12, 3), 77.89962374769583), ('TFRecord', (2, 12
, 3), 71.26241777410284), ('TFRecord', (2, 12, 3), 81.78887688195195), ('TFRecord', (4, 3, 1), 32.0949464070842
5), ('TFRecord', (4, 3, 2), 37.04462874151012), ('TFRecord', (4, 3, 2), 31.921356673488763), ('TFRecord', (4, 3
, 3), 40.088799398488575), ('TFRecord', (4, 3, 3), 34.3754318985094), ('TFRecord', (4, 3, 3), 37.72656221305927
5), ('TFRecord', (4, 6, 1), 70.82470579792331), ('TFRecord', (4, 6, 2), 78.82591601000131), ('TFRecord', (4, 6,
2), 73.55718068781677), ('TFRecord', (4, 6, 3), 73.83195236066014), ('TFRecord', (4, 6, 3), 64.7188819803098),
('TFRecord', (4, 6, 3), 77.03792229419471), ('TFRecord', (4, 9, 1), 87.67362950206098), ('TFRecord', (4, 9, 2),
122.97907982505437), ('TFRecord', (4, 9, 2), 109.3887240356083), ('TFRecord', (4, 9, 3), 98.46721453659516), ('
TFRecord', (4, 9, 3), 115.09173270688812), ('TFRecord', (4, 9, 3), 99.85797471987577), ('TFRecord', (4, 12, 1),
133.80347244929698), ('TFRecord', (4, 12, 2), 140.67086317740004), ('TFRecord', (4, 12, 2), 139.4111656848699),
('TFRecord', (4, 12, 3), 136.45633973886262), ('TFRecord', (4, 12, 3), 125.38057100722166), ('TFRecord', (4, 12
, 3), 144.78969216297153), ('Image', (2, 3, 1), 4.064098414980844), ('Image', (2, 3, 2), 7.815475719091838), ('
Image', (2, 3, 2), 8.501094482188316), ('Image', (2, 3, 3), 8.127611432476504), ('Image', (2, 3, 3), 7.92706270
12806365), ('Image', (2, 3, 3), 7.743485331245086), ('Image', (2, 6, 1), 8.317956662435384), ('Image', (2, 6, 2
), 8.39149607323187), ('Image', (2, 6, 2), 8.913268221794803), ('Image', (2, 6, 3), 8.39626679469456), ('Image'
, (2, 6, 3), 8.700475079132318), ('Image', (2, 6, 3), 8.684269753263008), ('Image', (2, 9, 1), 8.04354184241417
5), ('Image', (2, 9, 2), 8.484966103392155), ('Image', (2, 9, 2), 8.247591595302582), ('Image', (2, 9, 3), 8.49
7821051957418), ('Image', (2, 9, 3), 6.829483878169507), ('Image', (2, 9, 3), 8.519150732970521), ('Image', (2,
12, 1), 8.066017058113966), ('Image', (2, 12, 2), 8.292142739464232), ('Image', (2, 12, 2), 9.350642633595742),
('Image', (2, 12, 3), 9.019686124125132), ('Image', (2, 12, 3), 8.663754400158362), ('Image', (2, 12, 3), 8.321
030140551866), ('Image', (4, 3, 1), 7.997422095615084), ('Image', (4, 3, 2), 8.430377711434254), ('Image', (4,
3, 2), 9.236254479198188), ('Image', (4, 3, 3), 8.3224005296531), ('Image', (4, 3, 3), 9.128501993495815), ('Im
age', (4, 3, 3), 9.081366414911935), ('Image', (4, 6, 1), 9.238312577331396), ('Image', (4, 6, 2), 9.4199575470
09579), ('Image', (4, 6, 2), 9.374353276013142), ('Image', (4, 6, 3), 8.958867542238396), ('Image', (4, 6, 3),
9.897539923408175), ('Image', (4, 6, 3), 9.225686013058693), ('Image', (4, 9, 1), 8.9281531597319), ('Image', (
4, 9, 2), 8.83297536160838), ('Image', (4, 9, 2), 9.628148701342425), ('Image', (4, 9, 3), 9.311832188998288),
('Image', (4, 9, 3), 9.764547156549492), ('Image', (4, 9, 3), 9.16865751604007), ('Image', (4, 12, 1), 9.624447
629435906), ('Image', (4, 12, 2), 9.636691213608945), ('Image', (4, 12, 2), 9.895435430121797), ('Image', (4, 1
2, 3), 9.955975575525757), ('Image', (4, 12, 3), 9.823049979595346), ('Image', (4, 12, 3), 9.47422823238864)]
Average Images per Second By Each Batch Size (Batch_Size: AVG_Images_per_Sec)
[(4, 50.779586230278774), (2, 28.62060671733013)]
Average Images per Second By Each Batch Number (Batch_Number: AVG_Images_per_Sec)
[(12, 58.10352390521206), (9, 46.57077368978701), (6, 33.20917095792271), (3, 18.672581908502966)]
Average Images per Second By Each Repetition (Repetition: AVG_Images_per_Sec)
[(1, 39.902618464093095), (2, 38.518019118374426), (3, 39.9435729524705)]
Average Images per Second By Each Batch Size (Associativity-method), (Batch_Size: AVG_Images_per_Sec)
[(4, 48.79294820323631), (2, 28.13458388555063)]
Average Images per Second By Each Batch Number (Associativity-method), (Batch_Number: AVG_Images_per_Sec)
[(12, 56.804326011146564), (9, 44.49631314070965), (6, 31.256141239586494), (3, 18.08197688324735)]
Average Images per Second By Repetition (Associativity-method), (Repetition: AVG_Images_per_Sec)
[(1, 37.88179564401834), (2, 37.09477353654646), (3, 39.47990942038509)]
```

In [ ]:

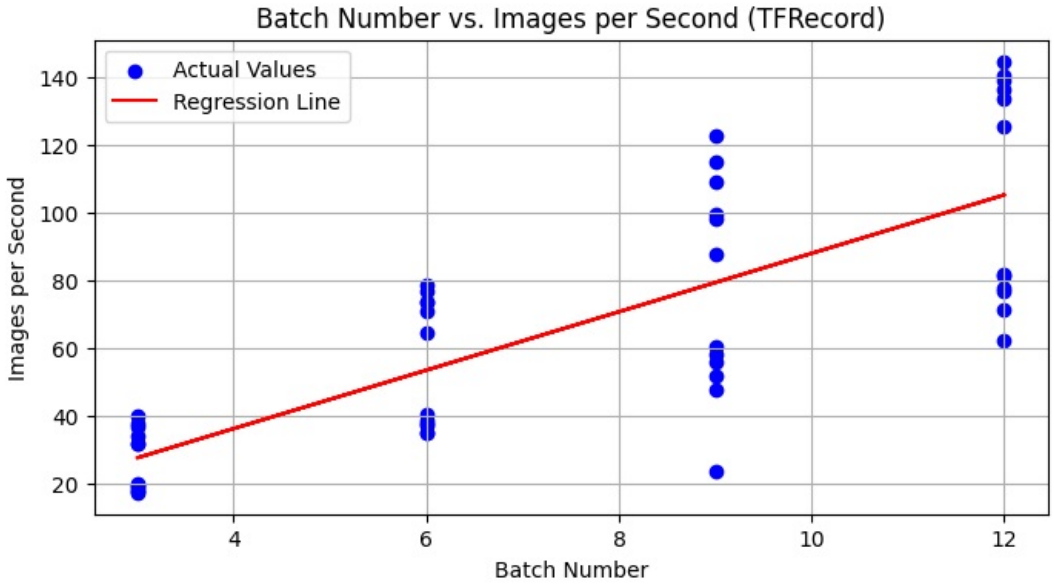
```python
import pandas as pd

#We create a dataset with each combination of parameters and the images per second for each dataset separately
data = {
    'Dataset': [item[0] for item in final_res_task_iii],
    'Batch Size': [item[1][0] for item in final_res_task_iii],
    'Batch Number': [item[1][1] for item in final_res_task_iii],
    'Repetitions': [item[1][2] for item in final_res_task_iii],
    'Images per Second': [item[2] for item in final_res_task_iii],
}

dataframe = pd.DataFrame(data)
dataframe['Dataset Volume (per Batch Setup)'] = dataframe['Batch Size'] * dataframe['Batch Number']
print(dataframe)

#Reference link for code: https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/pan
```

```
     Dataset  Batch Size  Batch Number  Repetitions  Images per Second  \
0   TFRecord           2             3            1          17.953839
1   TFRecord           2             3            2          18.369925
2   TFRecord           2             3            2          20.278241
3   TFRecord           2             3            3          19.355155
4   TFRecord           2             3            3          17.486856
..       ...         ...           ...          ...                ...
91     Image           4            12            2           9.636691
92     Image           4            12            2           9.895435
93     Image           4            12            3           9.955976
94     Image           4            12            3           9.823050
95     Image           4            12            3           9.474228

    Dataset Volume (per Batch Setup)
0                                  6
1                                  6
2                                  6
3                                  6
4                                  6
..                               ...
91                                48
92                                48
93                                48
94                                48
95                                48

[96 rows x 6 columns]
```

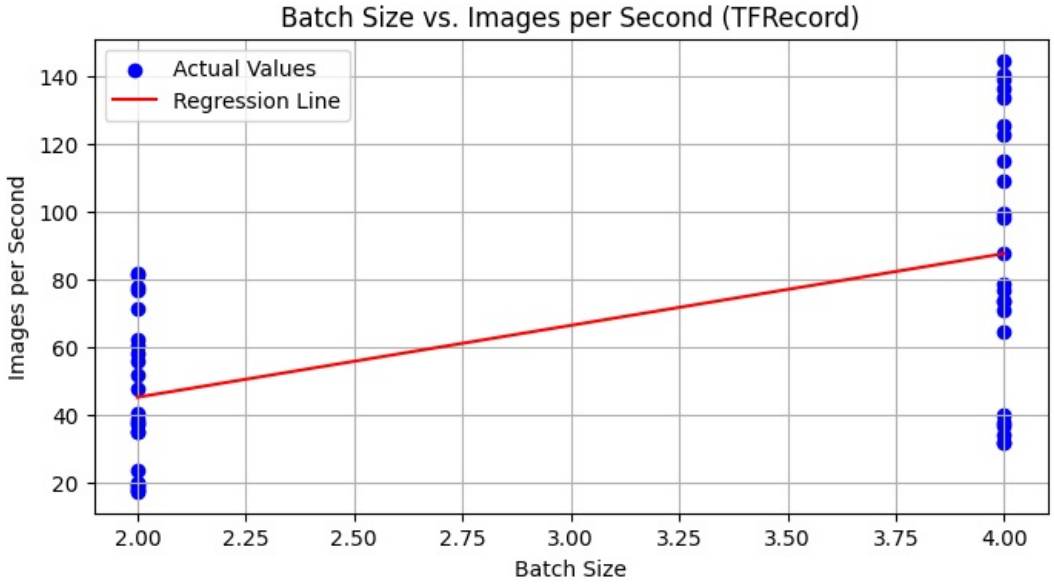
In [ ]:

```python
#Now we are going to perform a linear regression (e.g. using scikit-learn) over the values for each parameter a

#We import the necessary libraries
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from scipy.stats import t
import numpy as np
#This function is used to create the plots of the images per second by each feature and also shows the regressi
def regression_by_parameter(dataset, parameter, title):
  filtered_data = dataframe[dataframe['Dataset']==dataset] #We filter out the data for each dataset (TFRecord,

  #We fit the linear regression model on each parameter and make predictions
  X = filtered_data[[parameter]]
  y = filtered_data['Images per Second']
  linear_model = LinearRegression()
  linear_model.fit(X, y)
  model_predictions = linear_model.predict(X)

  #We calculate the metrics for the linear regression model
  slope = float(linear_model.coef_[0])
  intercept = float(linear_model.intercept_)
  residuals = y - model_predictions
  mse = mean_squared_error(y, model_predictions)
  n = len(y)
  d_o_f = n - 2  #Degrees of freedom
  t_statistic = slope / np.sqrt(mse / (n * np.var(X, ddof=1)))
  p_value = (1 - t.cdf(abs(t_statistic), d_o_f)) * 2  #We perform a two-tailed test

  #We plot the actual values vs the predicted values from the linear regression model for each parameter
  plt.figure(figsize=(8,4))
  plt.scatter(X[parameter], y, color='blue', label='Actual Values')
  plt.plot(X[parameter], model_predictions, color='red', label='Regression Line')
  plt.title(f'{title} ({dataset})')
  plt.xlabel(parameter)
  plt.ylabel('Images per Second')
  plt.legend()
  plt.grid(True)
  print(f'Slope: {slope:.2f}')
  print(f'Intercept: {intercept:.2f}')
  print('P-Value:', p_value)
  plt.show()

#We specify the parameters
parameters = ['Batch Number', 'Batch Size', 'Repetitions', 'Dataset Volume (per Batch Setup)']
titles = ['Batch Number vs. Images per Second', 'Batch Size vs. Images per Second',
          'Repetitions vs. Images per Second', 'Dataset Volume (per Batch Setup) vs. Images per Second']

#We generate the plots for the TFRecord Dataset
for parameter, title in zip(parameters, titles):
    regression_by_parameter('TFRecord', parameter, title)

#We generate the plots for the Image Dataset
for parameter, title in zip(parameters, titles):
    regression_by_parameter('Image', parameter, title)

#Reference link for code: https://colab.research.google.com/github/csmastersUH/data_analysis_with_python_2020/b
```

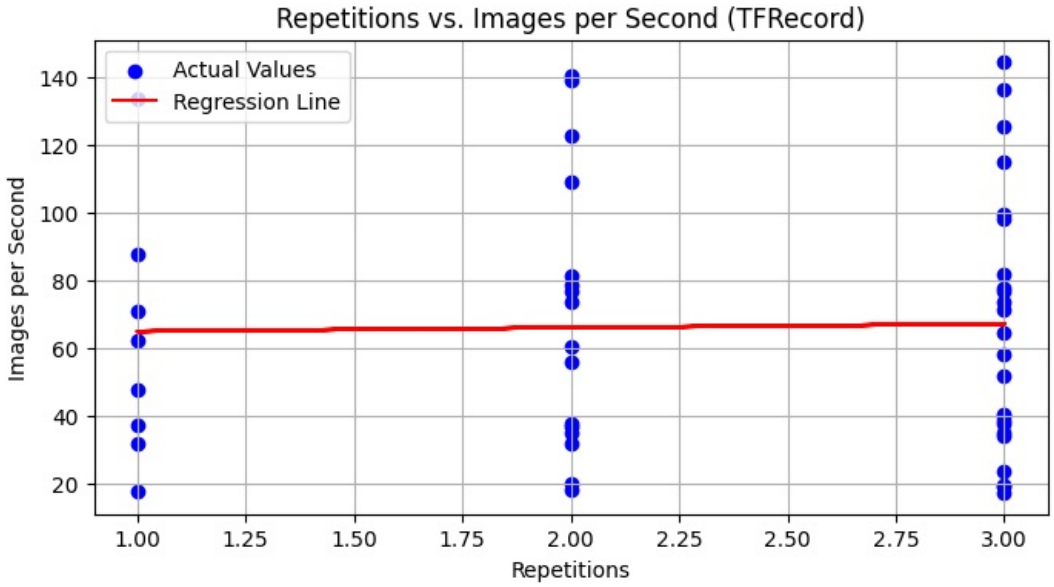
```
Slope: 8.63
Intercept: 1.83
P-Value: [7.48789919e-11]
```



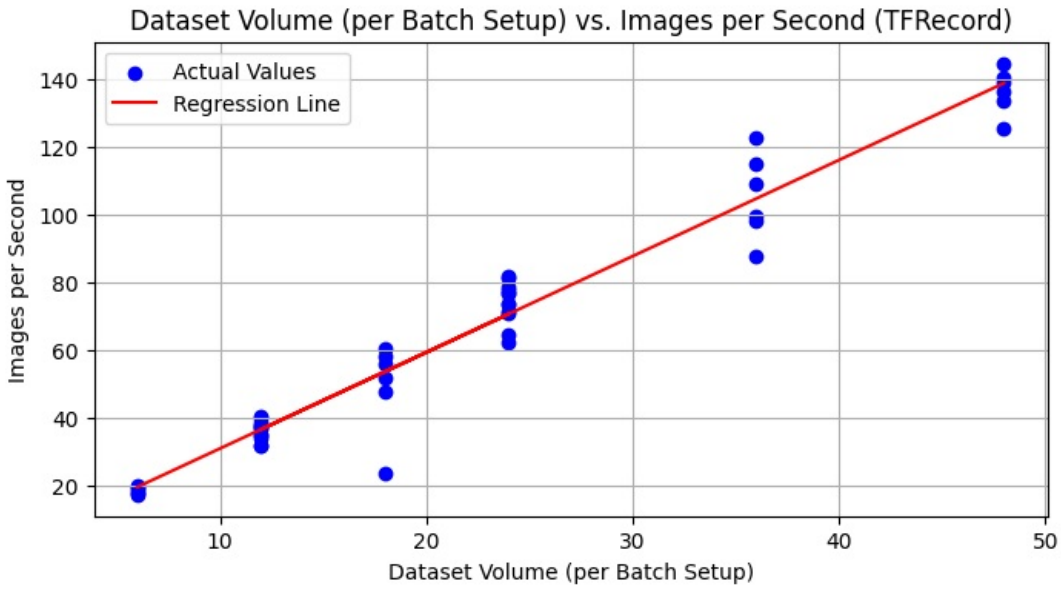

```
Slope: 21.22
Intercept: 2.87
P-Value: [1.85617356e-05]
```



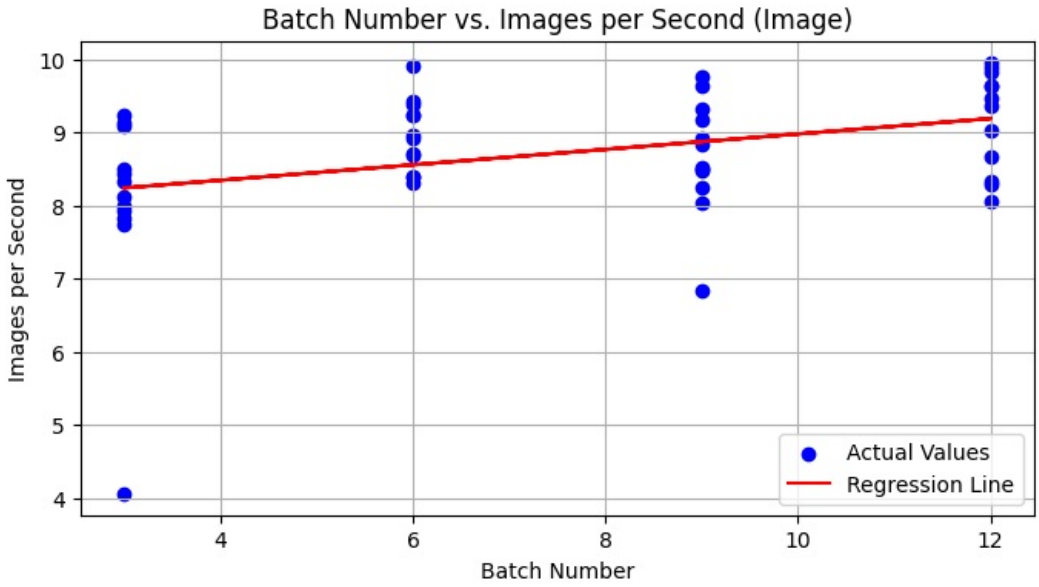

```
Slope: 1.10
Intercept: 63.97
P-Value: [0.8796578]
```




```
Slope: 2.84
Intercept: 2.62
P-Value: [0.]
```

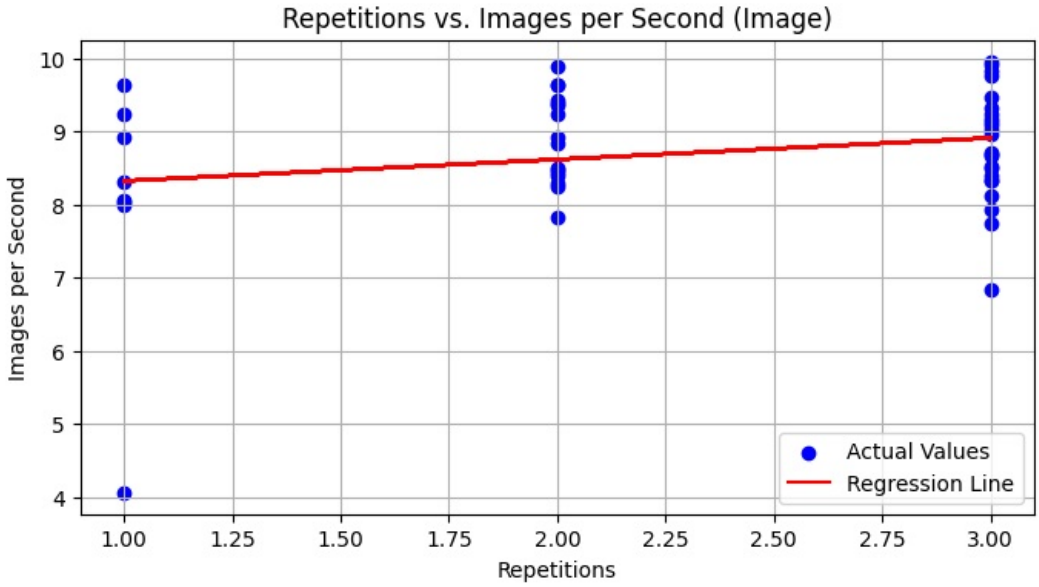



```
Slope: 0.11
Intercept: 7.92
P-Value: [0.00747793]
```

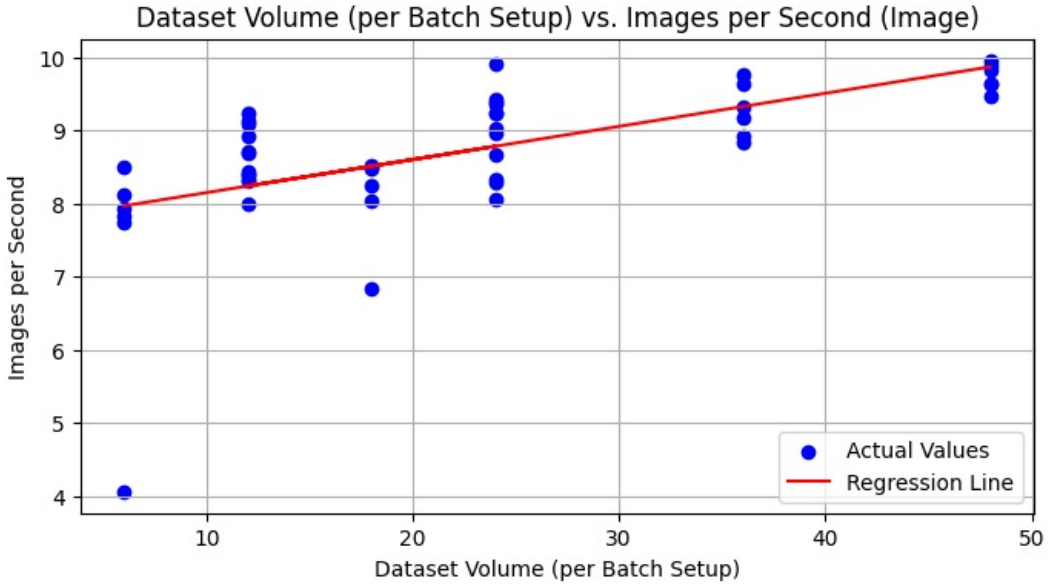



```
Slope: 0.55
Intercept: 7.06
P-Value: [1.02428895e-05]
```


```
Slope: 0.29
Intercept: 8.03
P-Value: [0.10514642]
```




```
Slope: 0.05
Intercept: 7.70
P-Value: [1.6921853e-06]
```



Section 3. Theoretical discussion

Task 3: Discussion in context. (24%)

In this task we refer an idea that is introduced in this paper:

- Alipourfard, O., Liu, H. H., Chen, J., Venkataraman, S., Yu, M., & Zhang, M. (2017). Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics.. In USENIX NSDI 17 (pp. 469-482).

Alipourfard et al (2017) introduce the prediction an optimal or near-optimal cloud configuration for a given compute task.

3a) Contextualise

Relate the previous tasks and the results to this concept. (It is not necessary to work through the full details of the paper, focus just on the main ideas). To what extent and under what conditions do the concepts and techniques in the paper apply to the task in this coursework? (12%)

3b) Strategise

Define - as far as possible - concrete strategies for different application scenarios (batch, stream) and discuss the general relationship with the concepts above. (12%)

Provide the answers to these questions in your report.

Final cleanup

Once you have finshed the work, you can delete the buckets, to stop incurring cost that depletes your credit.

In [ ]:
```
!gsutil -m rm -r $BUCKET/* # Empty your bucket
!gsutil rb $BUCKET # delete the bucket
```