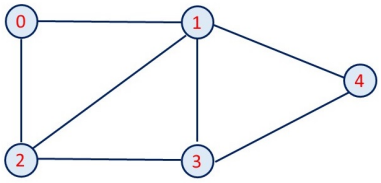
INP7079233 - BIG DATA COMPUTING (Proff. A: Pietracaprina and F. Silvestri) 2022-2023

Assignment of Homework 1 (DEADLINE April 12, 23:59)

The **purpose of this first homework** is to get acquainted with Spark and with its use to implement MapReduce algorithms. In preparation for the homework, you must set up your environment following the instructions given in Moodle Exam, in the same section as this page. After the set up is complete, test it using the WordCountExample program (Java or Python), and familiarize with the Spark methods it uses. The Introduction to Programming in Spark may turn out useful to this purpose.

TRIANGLE COUNTING. In the homework you must implement and test in Spark two MapReduce algorithms to count the number of distinct triangles in an undirected graph $G = (V, E)$, where a triangle is defined by 3 vertices u, v, w in V, such that $(u, v), (v, w), (w, u)$ are in E. In the right image, you find an example of a 5-node graph with 3 triangles. Triangle counting is a popular primitive in social network analysis, where it is used to detect communities and measure the cohesiveness of those communities. It has also been used is different other scenarios, for instance: detecting web spam (the distributions of local triangle frequency of spam hosts significantly differ from those of the non-spam hosts), uncovering the hidden thematic structure in the World Wide Web (connected regions of the web which are dense in triangles represents a common topic), query plan optimization in databases (triangle counting can be used for estimating the size of some joins).



3 DISTINCT TRIANGLES: (0,1,2), (1,2,3), (1,3,4)

Both algorithms use an integer parameter $C \geq 1$, which is used to partition the data.

ALGORITHM 1: Define a **hash function** h_C which maps each vertex u in V into a color $h_C(u)$ in $[0, C-1]$. To this purpose, we advise you to use the hash function

$$h_C(u) = ((a \cdot u + b) \mod p) \mod C$$

where $p = 8191$ (which is prime), a **is a random integer in** $[1, p-1]$**, and** b **is a random integer in** $[0, p-1]$**.**

Round 1:

- Create C subsets of edges, where, for $0 \leq i < C$, the i-th subset, $E(i)$ consist of all edges (u, v) of E such that $h_C(u) = h_C(v) = i$. Note that if the two endpoints of an edge have different colors, the edge does not belong to any $E(i)$ and will be ignored by the algorithm.
- Compute the number $t(i)$ triangles formed by edges of $E(i)$, separately for each $0 \leq i < C$.

Round 2: Compute and return $t_{final} = C^2 \sum_{0 \leq i < C} t(i)$ as final estimate of the number of triangles in G.

In the homework you must develop an implementation of this algorithm as a method/function ***MR_ApproxTCwithNodeColors.*** *(More details below.)*

ALGORITHM 2:

Round 1:

- Partition the edges at random into C subsets $E(0), E(1), \ldots E(C-1)$. Note that, unlike the previous algorithm, now every edge ends up in some $E(i)$.
- Compute the number t(i) of triangles formed by edges of E(i), separately for each $0 \leq i < C$.

Round 2: Compute and return $t_{final} = C^2 \sum_{0 \leq i < C} t(i)$ as final estimate of the number of triangles in G.

In the homework you must develop an implementation of this algorithm as a method/function ***MR_ApproxTCwithSparkPartitions*** *that, in Round 1,* ***uses the partitions provided by Spark****, which you can access through method mapPartitions. (More details below.)*

SEQUENTIAL CODE for TRIANGLE COUNTING. Both algorithms above require (in Round 1) to compute the number of triangles formed by edges in the subsets $E(i)$'s. To this purpose you can use the methods/functions provided in the following files: CountTriangles.java (Java) and CountTriangles.py (Python).

DATA FORMAT. To implement the algorithms assume that the vertices (set V) are represented as 32-bit integers (i.e., type Integer in Java), and that the graph G is given in input as the set of edges E stored in a file. Each row of the file contains one edge stored as two integers (the edge's endpoints) separated by comma (','). Each edge of E appears exactly once in the file and E does not contain multiple copies of the same edge.

TASK for HW1:

1) Write the method/function MR_ApproxTCwithNodeColors which implements **ALGORITHM 1**. Specifically, MR_ApproxTCwithNodeColors must take as input an RDD of edges and a number of colors C and must return an estimate t_{final} of the number of triangles formed by the input edges computed through transformations of the input RDD, as specified by the algorithm. *It is important that the local space required by the algorithm be proportional to the size of the largest subset $E(i)$* (hence, you cannot download the whole graph into a local data structure). **Hint:** define the hash function h_C inside MR_ApproxTCwithNodeColors, but before starting processing the RDD, so that all transformations of the RDD will use the same hash function, but different runs of MR_ApproxTCwithNodeColors will use different hash functions (i.e., defined by different values of a and b).

2) Write the method/function MR_ApproxTCwithSparkPartitions which implements **ALGORITHM 2**. Specifically, MR_ApproxTCwithSparkPartitions must take as input an RDD of edges and the number of partitions C, and must return an estimate t_{final} of the number of triangles formed by the input edges computed through transformations of the input RDD, as specified by the algorithm. In particular, the input RDD must be subdivided into C partitions and each partition, accessed through one of the mapPartitions methods offered by Spark, will represent one of the subsets $E(i)$ appearing in the high-level description above. If the RDD is passed to the method already subdivided into C partitions, it is not necessary to repartition it. *It is important that the local space required by the algorithm be proportional to the size of the largest subset Spark partition.*

3) Write a program GxxxHW1.java (for Java users) or **GxxxHW1.py** (for Python users), where xxx is your 3-digit group number (e.g., 004 or 045), which receives in input, as command-line arguments, 2 integers C and R, and a path to the file storing the input graph, and does the following:

- Reads parameters C and R
- Reads the input graph into an RDD of strings (called **rawData**) and transform it into an RDD of edges (called **edges**), represented as pairs of integers, partitioned into C partitions, and cached.
- Prints: the name of the file, the number of edges of the graph, C, and R.
- Runs R times **MR_ApproxTCwithNodeColors** to get R independent estimates t_{final} of the number of triangles in the input graph.
- Prints: the median of the R estimates returned by MR_ApproxTCwithNodeColors and the average running time of MR_ApproxTCwithNodeColors over the R runs.
- Runs **MR_ApproxTCwithSparkPartitions** to get an estimate t_{final} of the number of triangles in the input graph.
- Prints: the estimate returned by MR_ApproxTCwithSparkPartitions and its running time.

File OutputFBsmallC2R5.txt shows you how to format your output. Make sure that your program complies with this format.

4) Test your program using the datasets that we provide in the same section as this page, together with the outputs of some runs of our program on the datasets. Note that while using $C = 1$ should give the exact count of triangles (unique for each graph), using $C > 1$ provides only an approximate count. So, for $C = 1$ your counts should be the same as ours, but for $C > 1$ they may differ. **Fill the table given in this word file** HW1-Table.docx with the required values obtained when testing your program.

SUBMISSION INSTRUCTIONS. Each group must submit a **zipped folder GxxxHW1.zip** (xxx is the group ID). The folder must contain the program (**GxxxHW1.java** or **GxxxHW1.py**) and a file **GxxxHW1table.docx** with the aforementioned table. Only one student per group must submit the zipped folder in Moodle Exam using the link provided in the Homework1 section. Make sure that your code is free from compiling/run-time errors and that you use the file/variable names in the homework description, otherwise your score will be penalized.

*If you have questions about the assignment, contact the teaching assistants (TAs) by email to bdc-course@dei.unipd.it . The subject of the email must be "**HW1 - Group xxx**", where xxx is your group ID. If needed, a zoom meeting between the TAs and the group will be organized.*

Last modified: Wednesday, 5 April 2023, 12:01 PM

◄ Machine Setup

Jump to...

Submission form for HW1 ►

Data retention summary

INP7079233 - BIG DATA COMPUTING (Proff. A: Pietracaprina and F. Silvestri) 2022-2023

Assignment of Homework 2 (DEADLINE: May 22, 23:59)

In this homework, you will run a Spark program on the CloudVeneto cluster. As for Homework 1, the objective is to estimate (approximately or exactly) the number of triangles in an undirected graph $G = (V, E)$. More specifically, your program must implement two algorithms:

ALGORITHM 1. The same as Algorithm 1 in Homework 1, so you must recycle method/function MR_ApproxTCwithNodeColors devised for Homework 1, fixing any bugs that we have pointed out to you.

ALGORITHM 2. A 2-round MapReduce algorithm which returns the exact number of triangles. The algorithm is based on node colors (as Algorithm 1) and works as follows. Let $C \geq 1$ be the number of colors and let $h_C(\cdot)$ ne the hash function that assigns a color to each node used in Algorithm 1.

Round 1

- For each edge $(u, v) \in E$ separately create C key-value pairs $(k_i, (u, v))$ with $i = 0, 1, \ldots C-1$ where each key k_i is a triplet containing the three colors $h_C(u), h_C(v), i$ sorted in non-decreasing order.
- For each key $k = (x, y, z)$ let L_k be the list of values (i.e., edges) of intermediate pairs with key k. Compute the number t_k of triangles formed by the edges of L_k whose node colors, in sorted order, are x, y, z. Note that the edges of L_k may form also triangles whose node colors are not the correct ones: e.g., (x, y, y) with $y \neq z$.

An example of Round 1 is given in this picture.

Round 2. Compute and output the sum of all t_k's determined in Round 1. It is easy to see that every triangle in the graph G is counted exactly once in the sum. You can assume that the total number of t_k's is small, so that they can be garthered in a local structure. Alternatively, you can use some ready-made reduce method to do the sum. Both approaches are fine.

Using the cluster

A brief description of the cluster available for the course, together with instructions on how to access the cluster and how to run your program on it are given in this User guide for the cluster on CloudVeneto.

TASK for HW2:

1) Fix bugs (if any) of method/function MR_ApproxTCwithNodeColors written for HW1, which implements **ALGORITHM 1**.

2) Write a method/function MR_ExactTC which implements **ALGORITHM 2**. Specifically, MR_ExactTC must take as input an RDD of edges and the number of colors C, and must return the exact triangle count.

- **Hint (for Java users).** To represent triplets of colors, you can use the scala type Tuple3<Integer,Integer,Integer>, importing scala.Tuple3 at the beginning of your code.
- **Hint.** In Round 1, in order to compute the number of triangles for each key $k = (x, y, z)$ you can run on the set of edges L_k a simple modification of method/function CountTriangles (the one used by MR_ApproxTCwithNodeColors and provided by us for HW1) which before incrementing the count for a new triangle, checks if the colors of its 3 nodes are x, y, z. You can use the the following code for the modified method/function: CountTriangles2.java and CountTriangles2.py. Note that our code receives in input the parameters a,b,p and C that define the hash function.

3) Write a program GxxxHW2.java (for Java users) or **GxxxHW2.py** (for Python users), where xxx is your 3-digit group number (e.g., 004 or 045), which receives in input, as command-line arguments, 2 integers C and R, a binary flag F, and a path to the file storing the input graph, and does the following:

- Reads parameters C, R and F
- Reads the input graph into an RDD of strings (called **rawData**) and transform it into an RDD of edges (called **edges**), represented as pairs of integers, partitioned into 32 partitions, and cached.
- Prints: the name of the file, the number of edges of the graph, C, and R
- If F=0:
  - Runs R times **MR_ApproxTCwithNodeColors** to get R independent estimates of the number of triangles in the input graph.

- Prints: the median of the R estimates returned by **MR_ApproxTCwithNodeColors** and the average running time of **MR_ApproxTCwithNodeColors** over the R runs.

- If F=1:
- Runs R times **MR_ExactTC** to get the exact number of triangles in the input graph
- Prints: the last value returned by **MR_ExactTC** (they are all equal) and the average running time over the R runs.

This file shows how to format your output. Make sure that your program complies with this format.

4) Test and debug your program in local mode on your PC *to make sure that it runs correctly. The program must be stand-alone in the sense that it should run without requiring additional files.*

5) Test your program on the cluster using the datasets which have been preloaded in the HDFS available in the cluster. Use various configurations of parameters and report your results using the table given in this word file.

WHEN USING THE CLUSTER, YOU MUST STRICTLY FOLLOW THESE RULES:

- **To avoid congestion, groups with even (resp., odd) group number must use the clusters in even (resp., odd) days.**
- **Do not run several instances of your program at once.**
- **Do not use more than 16 executors.**
- **Try your program on a smaller dataset first.**
- **Remember that if your program is stuck for more than 1 hour, its execution will be automatically stopped by the system.**

SUBMISSION INSTRUCTIONS. Each group must submit a **zipped folder GxxxHW2.zip**, where xxx is your group number. The folder must contain the program (**GxxxHW2.java** or **GxxxHW2.py**) and a file **GxxxHW2table.docx** with the aforementioned table. Only one student per group must do the submission using the link provided in the Homework 2 section. Make sure that your code is free from compiling/run-time errors and that you comply with the specification, otherwise your grade will be penalized.

*If you have questions about the assignment, contact the teaching assistants (TAs) by email to bdc-course@dei.unipd.it . The subject of the email must be "**HW2 - Group xxx**", where xxx is your group number. If needed, a zoom meeting between the TAs and the group will be organized.*

Last modified: Friday, 26 May 2023, 12:34 PM

◄ User Guide for the Cluster provided by Cloud Veneto

Jump to...

Submission form for HW2 ►

INP7079233 - BIG DATA COMPUTING (Proff. A: Pietracaprina and F. Silvestri) 2022-2023

Assignment of Homework 3 (DEADLINE: June 18, 23.59)

In this homework, you will use the Spark Streaming API to devise a program which processes a stream of items and assesses experimentally the space-accuracy tradeoffs featured by the count sketch to estimate the individual frequencies of the items and the second moment F_2.

Spark streaming setting that will be used for the homework

For the homework, we created a server which generates a continuous stream of **integer items**. The server has been already activated on the machine **algo.dei.unipd.it** and emits the items as strings on **port 8888**. Your program will define a **Spark Streaming context** that accesses the stream through the method **socketTextStream** which transforms the input stream (coming from the specified machine and port number) into **DStream** (*Discretized Stream*) of batches of items arrived during a time interval whose duration is specified at the creation of the context. A method **foreachRDD** is then invoked to process the batches one after the other. Each batch is seen as an RDD and a set of RDD methods are available to process it. Typically, the processing of a batch entails the update of some data structures stored in the driver's local space (i.e., its *working memory*) which are needed to perform the required analysis. The beginning/end of the stream processing will be set by invoking **start/stop** methods from the context sc. For the homework, the stop command will be invoked after (approximately) 10M items have been read. The **threshold 10M** will be hardcoded as a constant in the program.

To learn more about Spark Streaming you may refer to the official Spark site. Relevant links are:

- Spark Streaming Programming Guide (full documentation)
- Transformations on Streams (list of transformations applicable to the RDDs of a DStream)

Running the program and template

Your program will be run in local mode, exactly as the one devised for Homework 1. The **master should be set to local[*]** (however, take notice that if you do not set the master it is also ok, since local[*] is the default master).

In order to see a concrete application of the above setting you can download the following **example program** which computes the exact number of distinct elements in the stream:

- (**Java version**) DistinctItemsExample.java (updated 04/06/23 at 9.32)
- (**Python version**) DistinctItemsExample.py (updated 04/06/2023 at 21.36).

We strongly encourage to use this program as a template for your homework.

WARNING: When executing your programs, if you receive an error message such as ***ERROR ReceiverTracker: Deregistered receiver for stream 0: Stopped by driver*** do not worry. This is triggered by the stop signal and is due to an unclean management of the signal. However, it **has no consequence on the correctness of the execution.**

TASK for HW3.

You must write a program **GxxxHW3.java** (for Java users) or **GxxxHW3.py** (for Python users), where xxx is your 3-digit group number (e.g., 004 or 045), which receives in input the following **6 command-line arguments (in the given order):**

- **An integer** D: the number of rows of the count sketch
- **An integer** W: the number of columns of the count sketch
- **An integer** *left*: the left endpoint of the interval of interest
- **An integer** *right*: the right endpoint of the interval of interest
- **An integer** K: the number of top frequent items of interest
- **An integer** *portExp*: the port number

The program must read the first (approximately) 10M items of the stream Σ generated from **machine algo.dei.unipd.it** at port *portExp* and compute the following statistics. Let R denote the interval $[left, right]$ and let Σ_R be the substream consisting of all items of Σ belonging to R. The program must compute

- A $D \times W$ count sketch for Σ_R. To this purpose, you can use the same family of hash functions used in Homeworks 1 and 2, namely ((ax+b) mod p) mod C, where p=8191, a is a random integer in [1,p-1] and b is a random integer in [0,p-1]. The value C depends on the

range you want for the result.
- The exact frequencies of all distinct items of Σ_R
- The true second moment F_2 of Σ_R. To avoid large numbers, normalize F_2 by dividing it by $|\Sigma_R|^2$.
- The approximate second moment $\tilde{F}_2$ of Σ_R using count sketch, also normalized by dividing it by $|\Sigma_R|^2$.
- The average relative error of the frequency estimates provided by the count sketch where the average is computed over the items of $u \in \Sigma_R$ whose true frequency is $f_u \geq \phi(K)$, where $\phi(K)$ is the K-th largest frequency of the items of Σ_R. Recall that if $\tilde{f}_u$ is the estimated frequency for u, the relative error of is $|f_u - \tilde{f}_u|/f_u$.

The program should print:

- The input parameters provided as command-line arguments
- The lengths of the streams ($|\Sigma|$ and $|\Sigma_R|$)
- The number of distinct items in Σ_R
- The average relative error of the frequency estimates for the items of Σ_R with the top-K highest true frequencies
- (Only if $K \leq 20$) True and estimated frequencies of the items of Σ_R with the top-K highest true frequencies (no specific order required).

THIS FILE shows how to format your output. Make sure that your program complies with this format.

The program that you submit should run without requiring additional files. Test your program on your local or virtual machine using various configurations of parameters, and **report your results using the table given in** THIS WORD FILE.

SUBMISSION INSTRUCTIONS. Each group must submit a zipped folder GxxxHW3.zip, where xxx is your group number. The folder must contain the program (GxxxHW3.java or GxxxHW3.py) and a file GxxxHW3table.docx with the aforementioned table. Only one student per group must do the submission using the link provided in the Homework 3 section. Make sure that your code is free from compiling/run-time errors and that you comply with the specification, otherwise your grade will be penalized.

If you have questions about the assignment, contact the teaching assistants (TAs) by email to bdc-course@dei.unipd.it . The subject of the email must be "HW3 - Group xxx", where xxx is your group number. If needed, a zoom meeting between the TAs and the group will be organized.

Last modified: Monday, 5 June 2023, 12:12 PM

◄ Submission form for HW2

Jump to...

Submission form for HW3 ►