Arbitrage Strategy using Negative Cycle Detection Algorithm

Muhammad Rifky Indraputra Bariansyah 13517081
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517081@std.stei.itb.ac.id

***Abstract*—This paper will discuss the implementation of negative cycle detection algorithm, Bellman-Ford algorithm, in finding the best arbitrage strategy. Arbitrage is an action that depends on the market inefficiency, where in this situation it is possible to make profit out of buying and selling valuables at the same moment. This opportunity is in need of fine decision making, which asset to purchase or which asset to sell, in order to acquire maximum profit. Here's where we will implements Bellman-Ford algorithm.**

***Keywords*—Shortest path first algorithm, Dijkstra's algorithm, arbitrage**

I. Introduction

Since the day human form of communication is originated in the prehistory, we have been exchanging goods and services with one another and not a single second has passed without it. From the basic concept of people working hard in one product and relying on trades for other resources from a place called the market. Nowadays it has merely become a part of complex company,j systems to maximize the profit. Trades between nations or international trade has become the foundation that built the economy growth.

A trader, who makes profit from price fluctuations, watches the market and manages monetary investment for a client company by buying and selling assets. He or she has a job to create a strategy with the lowest of risks and highest of profit. Sometimes fundamental anomaly happens in the market. What happen is the condition contradicts the very definition of an efficient market where prices of asset reflect the value of it. In an anomaly or inefficiency market price is inbalance or distorted.

This events then can be exploited as a trading strategy by purchasing and selling goods profiting out of price imbalance. Such opportunity is called arbitrage in trading. If we see from a perspective the environment of assets, prices, and rates of exchanging the assets are sorts of mirroring the structure of a graph. With assets as nodes and exchange rates as lines that connect them.
Many problems in algorithm strategy is solved by creating algorithm around the structure of a graph as a network of information such as graph coloring, minimum spanning tree, shortest path finding etc. If we imagine arbitrage with the structure of a graph, then creating strategy for such opportunity can be done with graph problem solving algorithm. In the addition of rising cryptocurrency which has been around for only a decade, price inefficiencies will much likely to occur hence a higher opportunity in arbitrage exploitation. This correlation leads the writer to exploit an arbitrage opportunity by solving it as a graph problem.

II. Fundamental Theories

A. Arbitrage

Arbitrage is an action of purchasing and selling assets simultaneously to profit from an imbalance in the price [1]. This can occur risk-free for the trader by selling an asset in higher price. As an example, an asset named a is trading at $20 on New York Stock Exchange (NYSE) but at the same time the asset is trading at $20.05 on London Stock Exchange (LSE). In this situation anyone can buy the asset at New York Stock Exchange and simultaneously selling it on London Stock Exchange then earning 5 cent as profit.

1. *Triangular Arbitrage*

In a more complicated form of arbitrage, triangular arbitrage is the result of discrepancy that occurs when rates between currencies exchange does not exactly match[2]. This opportunity assumes low transaction costs or expenses incurred when buying or selling. International banks, who make markets in currencies, exploits an inefficiency in the market where somewhere is overvalued and somewhere else is undervalued. Price differences from selling and purchasing are only fractions of a cent, thus to be profitable large amount of capital needed.

In example, suppose a trader have a capital of $1 million. Exchange rate in the markets are:

TABLE I Currency exchange

	EUR	USD	GBP
EUR	1	1.1586	0.6849
USD	0.8631	1	0.5904
GBP	1.46	1.6939	1

1. Sell dollars for euros: $1 million x 0.8631 = €863,100
2. Sell euros for pounds: €863,100/1.4600 = £591,164.40
3. Sell pounds for dollars: £591,164.40 x 1.6939 = $1,001,373

To find profit subtract final amount with initial amount of investment, $1,001,373 - $1,000,000 = $1,373 as profit without transactional costs.

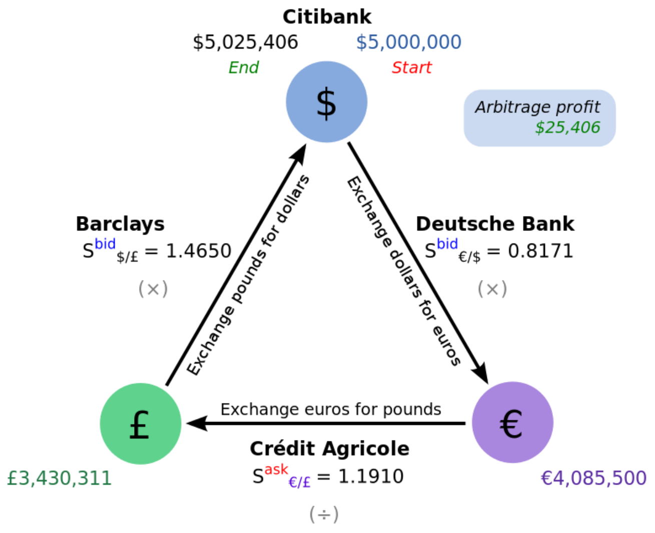


Figure 1 Triangular arbitrage trading Source:(https://commons.wikimedia.org/wiki/File:Triangular-arbitrage.svg,by John Sandy)

Such events in real life will extremely hard to occurs in a long period of time. In the market any imbalance will acted upon quickly by advancement in technologies.

C. Graph

A Graph is an ordered pair *G=(V,E)*

1. V is a set of vertices or nodes or points
2. $E \subseteq \{\{x, y\} \mid (x, y) \in V^2 \wedge x \neq y\}$

   A set of edges or lines or links are pairs of edges. The edge is said to be join *u* and *v* and to be incident on *u* and incident on *v*.
3. Multiple edges

   Two or more edges joining same pair of vertices.
4. Directed Graph

   Is a graph that is its edges have a direction associated with them.

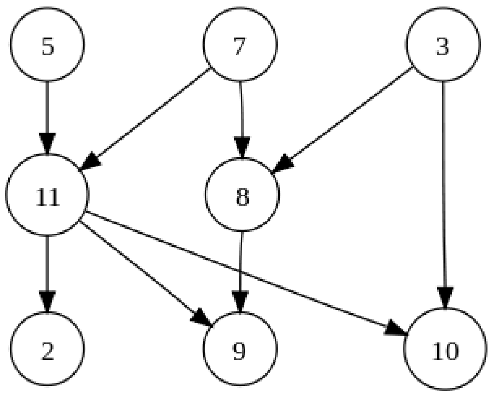


Figure 2 Directed Graph Source:(https://en.wikipedia.org/wiki/File:Directed_acyclic_graph_2.svg,by Johanes Rossel)

D. Shortest-Paths Problem

In a shortest-paths problem, we're given a weighted and directed graph *G=(V,E)*, (*V* = vertices, *E* = Edges), with weight function *w*: E →R mapping edges to real-valued weights. The weight *w(p)* of path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is the sum of the weights of its constituent edges:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i) .$$

Shortest-paths weight δ (*u,v*) from *u* to *v* defined by

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\leadsto} v\} & \text{if there's a path from u to v,} \\ \infty & \text{otherwise.} \end{cases}$$

A shortest path from vertex *u* to vertex *v* is then defined as any path *p* with weight *w(p)* = δ (*u,v*)[3]. Weight of edges can be represented for distances, time, cost, loss or any other quantities. In this problem, our goal is to find shortest-paths between vertices. In a graph, edge's weight can have the value of negative number and called negative-weight edges.

1. *Negative-Weighted Cycle*

A simple cycle or directed cycle is a sequence consisting at least two consecutive vertices starting and ending with the same vertex. Each edge must be aligned from the earlier to the later vertex[4]. After this point of this paper simple cycle will be referred as cycle. A cycle that sum of edges weight is negative is called negative-weighted cycle. If a negative-weighted cycle founds in a graph, any path that has a point on the cycle can be made cheaper in every walk, hence, there is no shortest-paths[5].

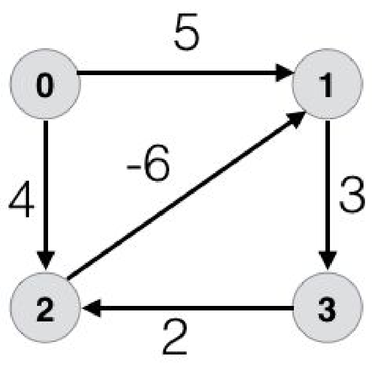


Figure 5 Negative-Weight Cycle
Source:(https://www.dyclassroom.com/)

2→1→3→2 weight total is -1 which mean it is a negative-weight cycle.

E. Relaxation

Relaxation methods are methods of solving partial differential equations involving splitting the sparse matrix that rises from finite differencing and then iterating until a solution is found[6]. To implements relaxation we set distance to starting vertex to 0 and set distance to every other vertices to ∞ To have an edge relaxation means edge from *u* to *v* is tested whether the best current walk from starting vertex to *v* is to go from starting vertex to *u*, if so, we update our distance data.

```
Relaxation(u, v, w)
if v.distanceTo > u.distanceTo + w(u,v) then
    v.distanceTo = u.distanceTo + w(u,v)
    v.predecessor = u
```

Figure 3 Relaxation algorithm
Source:(https://algs4.cs.princeton.edu/44sp/, modified)

F. Bellman-Ford Algorithm

Bellman-Ford algorithm solves shortest-paths problem in general case whereas edge weights may be valued negative. Given graph *G=(V,E)* with weight function *w*: E →R and starting vertex *s*, the algorithms runs by relaxing edges and progressively decrease an estimate *v*.distanceTo on the weight of a shortest path from *s* to each *v* in V until shortest-path weight is achieved.

```
Bellman-Ford(G, w, s)
Initialize-Single-Source(G,s)
    for i = 1 to |G.V| - 1
        for each edge (u,v) ∈ G.E
            Relaxation(u,v,w)
    for each edge (u,v) ∈ G.E
        if v.distanceTo > u.distanceTo + w(u,v)
            return FALSE
return TRUE
```

Figure 4 Bellman-Ford algorithm
Source:(Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms)

The algorithm will return false if and only if the input graph contains negative-weight cycles that are reachable from source[3]. As explained in the preceding part, there is no solution exists if such cycle exists.

The algorithm passes |*V*| - 1 times over every edges with each pass consist of graph relaxation. Then it passes one more time to check if there's still exist *v*.distanceTo > *u*.distanceTo + *w*(*u*,*v*) and if so it will return false indicating a negative-weight cycle in the graph. Bellman-Ford algorithm runs in time *O*(*VE*), Θ(*V*) for each |*V*| *-1* passes that takes Θ(*E*).

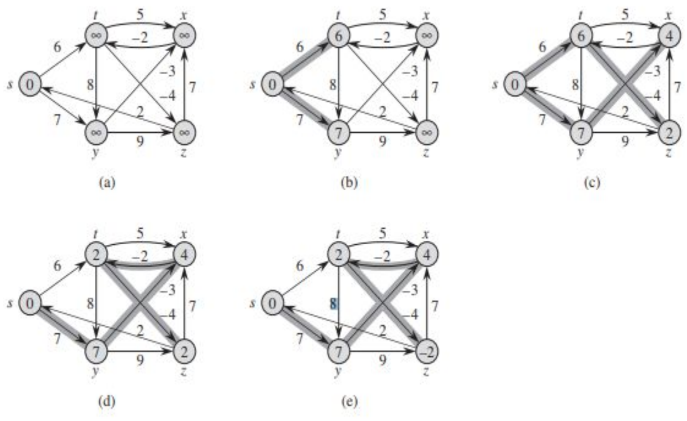


Figure 6 Bellman-Ford algorithm execution
Source:(Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms)

III. Implementation

For this paper we will use currency change as an example for arbitrage opportunity. To exploit the discrepancies in exchange rates, suppose we are given n number of currencies $C_1, C_2, C_3, \dots, C_n$. The currencies have a $n \times n$ sized matrix of exchange rates R. With $R[i,j]$ means currency C_i buys C_j vice versa. To earn profit we have to exchange 1 of currency C with other currency in a way that the exchange will return >1 value of currency C. The statement is equivalent to multiplying rates of exchange and resulting value greater than and can be represented as follows

$$R[i_1,i_2] \,.\, R[i_2,i_3] \,.\, R[i_3,i_4] \dots R[i_{k-1},i_k] \,.\, R[i_k,i_1] > 1 \quad (1)$$

By starting and finishing at the same currency we can call it a cycle of exchange. We can conclude that the problem is to find such cycle of exchange that satisfy the inequality.

1. *Problem Identification*

In order to solve the problem we can model it as a graph, the currencies as vertices and the rates of exchange as edges. We know by definition a sequence starting and ending in the same vertex, in this case representing rates of exchange, is a cycle. Using this approach means we will have to find a cycle on the graph that multiplies to a number that is greater than one.

We can solve this problem by using the brute force approach by creating every possible cycle in the graph and finally find a cycle with an arbitrage opportunity. From the second chapter we know an arbitrage event occurs only in a short moment of time. Time is a key component of success in this trading and that is why a faster algorithm is needed. From inequality (1) we observe that it satisfies if and only if

$$(1/R[i_1,i_2]) \,.\, (1/R[i_2,i_3]) \dots (1/R[i_{k-1},i_k]) \,.(1/R[i_k,i_1]) < 1 \quad (2)$$

and by taking logs of both sides

$$\log(1/R[i_1,i_2]) + \log(1/R[i_2,i_3]) + \dots + \log(1/R[i_{k-1},i_k]) + \log(1/R[i_k,i_1]) < 0 \quad (3)$$

From inequality (3), rather than using $R[i,j]$ as the weight of edges, we define the weight of $C_i \rightarrow C_j$ as

$$w(C_i,C_j) = \log(1/R[i,j]) = -\log(R[i,j])$$

referring the weight definition and inequality (3) we will have to find a cycle with negative sum of edges.

2. *Bellman-Ford algorithm implementation*

We can determine if a negative-weight cycle exists within a graph using Bellman-Ford algorithm, this means we can find the existence of arbitrage opportunity within exchange rates. At the stage of preparation, using the Bellman-Ford algorithm in chapter 2, if a negative cycle exists it will return false. For the purpose of the implementation we will have to make some modification to the algorithm :

1. At line 8, rather than returning True it will return null indicating arbitrage opportunity does not exist.
2. At line 7, rather than returning False at this point we will have to retract every vertices that is on the negative-weight cycle in order to show every exchange in the arbitrage opportunity.

Secondly, after modifying the Bellman-Ford algorithm, we will then have to initialize the graph. In this implementation with n currencies we'll represent the graph as a two dimensional $n \times n$ array. *Graph*[*i*][*j*] structure signify i as from-currency and j as to-currency with its value as negative logarithm of exchange rate.

After we have the graph, n sized array of distance and array of predecessor will be initialized at the first step of Bellman-Ford algorithm. Array of distance, d, will firstly all be filled by infinity and the array of predecessor, p, will all be filled by Null. The relaxation algorithm will also be modified. At the last part of relaxation we will assign p[j] with i.

Lastly, biggest different from the basic Bellman-Ford algorithm we will add an algorithm to save the negative-weighted cycle. We will name it Negative-Retracing. we will name starting vertes as s.

```
Negative-Retracing(p, s)
1 aCycle = [s]
2 next =s
3 while TRUE
4         next ←p[next]
5         if next in aCycle
6                 insert next into aCycle
7                 aCycle = aCycle[from index of next]
8                 return aCycle
9         else
10                insert next into aCycle
```

Figure 7 Negative-Retracing algorithm
Source:(Writer's documentation)

Negative-Retracting algorithm uses the array of predecessors to return an array that describes the negative-weight cycle.

IV. Case Studies

For this paper we will implement the algorithm using python programming language with additional math library. To best represent the problem solving we will have a case studies using a small set of currencies example.

TABLE II Set of Currencies example, case 1
(source:https://fx.priceonomics.com/v1/rates/)

	JPY	EUR	USD
JPY	1	0.0083286	0.0107064
EUR	144.9169585	1	1.414403
USD	92.4489261	0.7019005	1

1. *Program Implementation and Output*

Using math library, table below will represent in the form of negative logarithm with base e (e = 2.718281828459045).

TABLE III Set of $-{}^{e}\log$(Currencies), case 1

	JPY	EUR	USD
JPY	0	4.788059904155156	4.536913585482109
EUR	-4.97616087836529	0	-0.3467075339397166
USD	-4.526656341757014	0.353963622893828	0

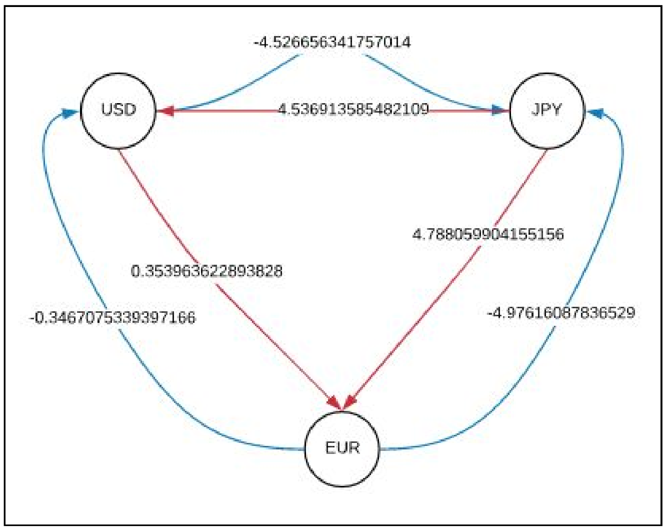


Figure 8 Graph representation, case 1
Source:(Writer's documentation)

Consequently we will do three iteration for each currency, the first two is to find the shortest path and the third will be used to find negative-weighted cycle in the graph. We will start with 100 unit of JPY and initialize array of distance [JPY, EUR, USD] with infinity, except the source as we will be setting it to zero , and array of predecessor with zero. :

$d=[0,\infty,\infty]$ $\quad p=[0,0,0]$

In the first iteration, relaxation will change *d*[EUR] to *w*(JPY, EUR) and *d*[USD] to *w*(JPY, USD).

$d=[0,4.788059,4.536913]$ $\quad p=[0,\mathrm{JPY},\mathrm{JPY}]$

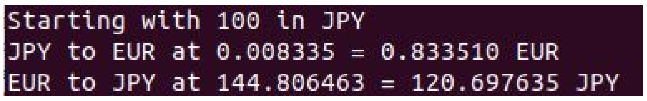
d[USD] will change to *d*[EUR] + *w*(EUR, USD) and *d*[JPY] will change to *d*[EUR] + *w*(EUR, JPY).

$d=[-0,1881,4.788,4.441]$ $\quad p=[\mathrm{EUR},\mathrm{JPY},\mathrm{EUR}]$

In the second iteration, relaxation will change *d*[USD] to *d*[JPY] + *w*(JPY, USD), *d*[EUR] to *d*[JPY] + *w*(JPY, EUR), *d*[USD] to *d*[EUR] + *w*(EUR, USD), and *d*[JPY] to *d*[EUR] + *w*(EUR, JPY).

$d=[-0,376,4,599,4,252]$ $\quad p=[\mathrm{EUR},\mathrm{JPY},\mathrm{EUR}]$

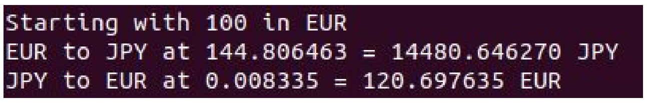
In the last part of third iteration, relaxation still changes *d*[JPY] to *d*[EUR] + *w*(EUR, JPY) this indicates an existence of a negative-weighted cycle. By using Negative-Retracting algorithm, it will return an output below:



```
Starting with 100 in JPY
JPY to EUR at 0.008335 = 0.833510 EUR
EUR to JPY at 144.806463 = 120.697635 JPY
```

Figure 9 Program output 1, case 1
Source:(Writer's documentation)

The algorithm will continue running with different starts at EUR and USD, there is also an arbitrage opportunity by starting with EUR. This concludes a profit of 20.697%



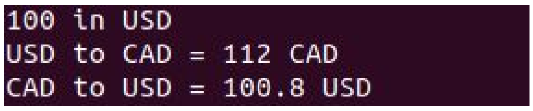
```
Starting with 100 in EUR
EUR to JPY at 144.806463 = 14480.646270 JPY
JPY to EUR at 0.008335 = 120.697635 EUR
```

Figure 10 Program output 2, case 1
Source:(Writer's documentation)

Using the program, here are some other example case studies.

TABLE IV Set of Currencies example, case 2

	USD	CAD	EUR
USD	1	1.12	0.72
CAD	0.90	1	0.64
EUR	1.38	1.56	1



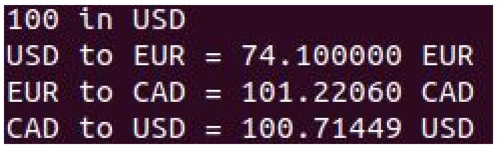
```
100 in USD
USD to CAD = 112 CAD
CAD to USD = 100.8 USD
```

Figure 11 Program output, case 2
Source:(Writer's documentation)

This concludes a profit of 0.8%

TABLE V Set of Currencies example, case 3

	USD	EUR	GBP	CAD
USD	1	0.741	0.657	1.005
EUR	1.349	1	0.888	1.366
GBP	1.521	1.126	1	1.538
CAD	0.995	0.732	0.65	1



```
100 in USD
USD to EUR = 74.100000 EUR
EUR to CAD = 101.22060 CAD
CAD to USD = 100.71449 USD
```

Figure 12 Program output, case 3
Source:(Writer's documentation)

This concludes a profit of 0.71449%

V. Conclusion and Opinion

In the market arbitrage opportunity is a very time-sensitive events that could appear in one second and be gone in another. This condition made the algorithm that is being used is a key component of success. In this paper it has been proven Negative-weight cycle detection algorithm can be used to detect an arbitrage opportunity rather than using brute force and generating every possible cycle. But, in line with the improvement of technologies that keeps the market effective and prices in balance there's also have to be a vigorous furtherance in making a faster algorithm that could detect these ineffectiveness going on.

After implementing the algorithm, using a case that is close to real world condition, this strategy only returns a very small percentage amount of profit. Regardless, being a risk-less strategy is still valuable for company or entity with a large amount of capital. Since there is always a relation between profit and risks in making a strategy, this strategy is the one where the main goal is not losing any money, that is why this strategy promise a slow growth and might not be suitable for a small capital trader

In conclusion, Bellman-Ford algorithm works for negative-weight cycle detection for arbitrage. Few points that could raise the impact of this trading strategy is improvement of algorithm and starting with a large capital.

Acknowledgment

First and foremost, I am grateful Allah swt. has given me strength to finish this piece of work in time. I would like to express my very great appreciation to M.T., Dr. Nur Ulfa Maulidevi, S.T., M.T., Dr. Masayu Leylia Khodra, S.T and Pak Dr. Ir. Rinaldi Munir for your guidance and appreciation towards my work through the semester. I'd like to mention that you have been a big part of inspiration in writing this paper. During the last parts of doing this paper, it has made me realize how this semester goes really fast and I might have taken these times for granted. For closure, I genuinely want to thank my friends for getting me to survive the semester, I wish there was a way to know you're in the good old days before you've actually left them.

References

[1] Chen, James. Arbitrage (https://www.investopedia.com/terms/a/arbitrage.asp)

[2] Chen, James. Triangular Arbitrage (https://www.investopedia.com/terms/t/triangulararbitrage.asp)

[3] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms.* The MIT Press.

[4] Balakrishnan, V.K. (2005). *Schaum's outline of theory and problems of graph theory* ([Nachdr.]. ed.). McGraw–Hill.

[5] Kleinberg, Jon; Tardos, Éva (2006). *Algorithm Design*. Pearson Education, Inc.

[6] Jeffreys, H. and Jeffreys, B. S.(1988). *Methods of Mathematical Physics*, 3rd ed. Cambridge, England: Cambridge University Press.

[7] Bender, Edward A.; Williamson, S. Gill (2010). Lists, Decisions and Graphs. With an Introduction to Probability.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 25 April 2019

[illegible]

M. Rifky I. Bariansyah
13517081