The Mathematics Education **ISSN 0047-6269**
Volume - LIX, No. 2, June 2025
Journal website: https://internationaljournalsiwan.com
ORCID Link: https://orcid.org/0009-0006-7467-6080
International Impact Factor: 7.75 https://impactfactorservice.com/home/journal/2295
Google Scholar: https://scholar.google.com/citations?hl=en&user=UOfM8B4AAAAJ
Refereed and Peer-Reviewed Quarterly Journal

Computer Programs for Antimagic Graphs Used in Encryption

by **Dharmendra Kumar Gurjar[1],** *Research Scholar,*
Auparajita Krishnaa[2], *Department of Mathematics and Statistics,*
University College of Science,
Mohan Lal Sukhadia University, Udaipur-313001, India
Email: 1. phdedudharmendra@mlsu.ac.in 2. akrishnaa1gmail.com

(Received: May 20, 2025; Accepted: June 6, 2025; Published Online: June 30, 2025)

Abstract:

In recent times, securing the data has become an important task while transferring it between two parties. Antimagic graphs exhibiting the antimagic graph labeling scheme play a vital role in developing the encryption techniques for securing the transferred data. Finding the Cipher texts manually costs much time, so developing computer programs for the encryption techniques to obtain the Cipher text easily in minimum time facilitates greatly the task of finding the Cipher-texts. This paper presents computer programs for the encryption techniques based on Antimagic graph labeling scheme which employ in particular the lexicographic order of permutations of edge labels for the Path graphs and Cycle graphs using the C++ programming language. Outputs of the results are also presented.

Keywords: Lexicographic order, permutation, Antimagic graph labeling, Antimagic Graph, Path, Cycle, Encryption, Plain text, Cipher text, Plain text, Antimagic Label Matrix.

1. Introduction:

Encryption is the important aspect for the cryptographic process. Encryption is the process of hiding the message while transferring it from one party to the other party. The original message which the sending party sends is called the *plain text* message and the hidden(encrypted) message is called the *cipher text*. In Graph Theory, a *graph labeling* is an assignment of numbers to either the *edges* or *vertices* or both depending on some conditions. If the vertices(edges) are labeled/numbered then the graph labeling is called *vertex (edge) labeling.* Use of labeled graphs in developing the encryption processes gives a new dimension to Cryptography in enhancement of security since the structure of the graph and the *graph labeling scheme* employed is not known to the unwanted third party.

Antimagic labeling of a graph has been introduced by **Hartsfield and Ringel [1]** and for concepts of Graph Theory, **Balakrishnan and Ranganathan [2]** can be referred to. An *antimagic labeling* of a graph is such that for a graph $G(p, q)$ with p number of vertices and q number of edges, the q edges are numbered (labeled) with $1, 2, 3, \ldots, q$ such that the sum of the edge labels incident on a vertex is distinct for each vertex. A graph exhibiting an *antimagic* labeling is called an *antimagic graph*. **Prihandoko, Dafik and Agustin [3]** present *super (a, d)-H antimagic total* graph to generate encryption keys which can be used to establish a stream cipher.

In this paper, a kind of *antimagic* labeling making use of the concept of *lexicographic order* of permutations in the edge labels is being used. The *lexicographic order* of permutations and combinations is as given in **Liu[4]** and follow the dictionary order. For example, in *factorial 4*, 1 2 3 4 is *permutation 1* while 1 2 4 3 is the *permutation 2*.

The permutations used in *lexicographic order* for labeling the edges in an *antimagic* labeling is a particular way of labeling with *permutation 1* and *permutation 2* for Path and Cycle graphs as given in **Krishnaa and Dulawat [5].** The permutations of *lexicographic order* of *antimagic* labeling, and Paths and Cycles have been chosen respectively for this work as they are easy to remember and these

graphs are relatively simpler in structure for developing the computer programs for demonstration purpose. The use of computers in applications of Graph Theory can also be seen in **Krishnaa[6]** and **Krishnaa[7]** where Krishnaa[6] has developed algorithms and a generalized computer software to test the existence of the major labeling schemes *(harmonious, felicitous, sequential, graceful, antimagic and magic*) for an arbitrary graph based on *lexicographic order of permutations and combinations.* Using this software, Krishnaa [7] has explored opportunities which would not be otherwise possible manually for various kinds of labelings for different kinds of graphs. **Gurjar and Krishnaa[8]** have discussed and introduced encryption techniques using *lexicographic labeled graphs* for different kinds of graphs like *antimagic* and *felicitous* Paths, *antimagic* cycles and *antimagic* Complete Graph. The *antimagic* Path and *antimagic* Cycle along with their cryptographic algorithms have been taken from Gurjar and Krishnaa[8] for developing the computer programs in this proposed paper to automate the process of computing Cipher texts using C++ computer language. Fundamental tools for the C++ programming are being mentioned in **Schildt[9]. Vasuki, Shobana and Roopa [10]** present the use of face antimagic labeling on double duplication of graphs using Hill cipher for encryption and decryption. Antimagic graphs in Cryptology have been presented in **Krishnaa [11]** and using Sanskrit for enhanced secrecy of hiding the *plain text* message in **[12]**.

2. Main Results:

There are *graph labeling* schemes in which the edges are labeled, which results in induced vertex labels. *Antimagic labeling* is one of those labeling. Antimagic labeled graphs are the graphs which are having antimagic labeling.

Antimagic labeling is defined as if the edges for any graph are numbered with 1, 2,…, q then the induced vertex label which is obtained by adding the edge labels of all the incident edges on a vertex is distinct for each vertex.

2.1 Antimagic Labeling of Path and Cycle Graphs:

Path graph with odd number of edges can be made *antimagic* if we use *permutation 2* on edge labels of the graph. If they have even number of edges then we will use *permutation 1* on edge labels for making them *antimagic*.

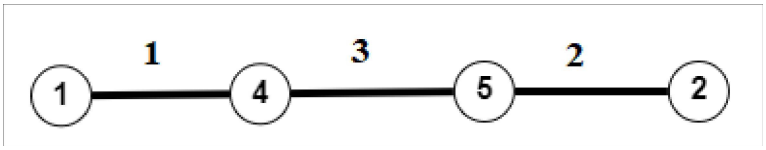

Figure 2.1.1: Antimagic Path $G(4, 3)$; Permutation 2 is used on edge labels

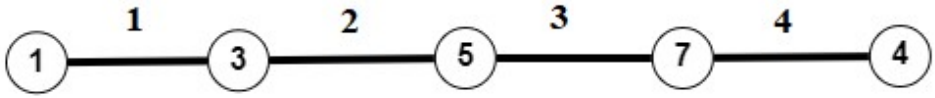

Figure 2.1.2: Antimagic Path $G(5, 4)$; Permutation 1 is used on edge labels

In case of *Cycle* graphs, which have even number of vertices/edges, we will use *permutation 2* on the edge labels of the graph and for the cycles having odd numbers of vertices/edges, we will use *permutation 1* on the edge labels for making the Cycle graphs *antimagic*.

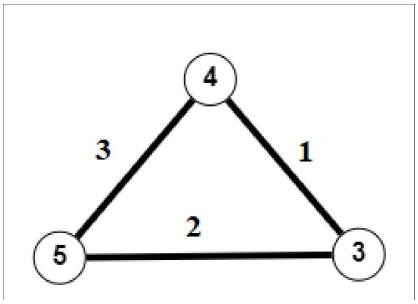

Figure 2.1.3: Antimagic Cycle with odd vertices; Permutation 1 is used on edge labels clockwise.

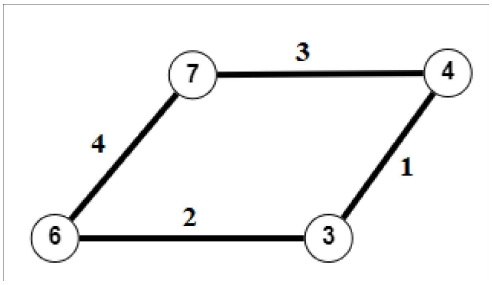

Figure 2.1.4: Antimagic Cycle with even vertices; Permutation 2 is used on edge labels clockwise.

***Antimagic Label Matrix*:**

For any graph $G(p, q)$ where p is the number of vertices and q is the number of edges, a square matrix of order $p \times p$ is said to be *antimagic label matrix* if entries $a(i, j)$ of this matrix are the edge label assigned to the edge formed by vertices labeled with induced vertex labels i and j. If no such edge label exists then entry $a(i, j) = 0$.

2.2 Encryption Algorithm:

Antimagic Path:

a) Label the edges with *permutation 2* to obtain *antimagic labeling* for Path of n even vertices and *permutation 1* to obtain *antimagic* labeling for Path of odd vertices.

b) Assign the letters of plain message to the *antimagic* edge labels from left to right.

c) Prepare the *Antimagic Label Matrix* for antimagic Path of even number of vertices using permutation 2 and for *antimagic* Path of odd number of vertices also.

d) Four schemes for encryption are being presented here to obtain the *cipher text*:

Scheme (**a**):

C_i = M_i + (vertex label to the right of M_i) for $i = 1$ to $(n - 1)$.

Where M_i is the i^{th} letter of the plain text and C_i is the i^{th} letter of the *cipher text* from left to right.

Scheme (**b**):

C_i = M_i + (vertex label to the left of M_i) for $i = 1$ to $(n - 1)$.

Where M_i is the i^{th} letter of the *plain text* and C_i is the i^{th} letter of the *cipher text* from left to right.

Scheme (**c**):

$C_i = M_i$ + (sum of right and left vertex labels) for $i = 1$ to $(n - 1)$.

Scheme (**d**):

$C_i = M_i$ + (absolute value of difference of right and left vertex labels) for $i = 1$ to $(n - 1)$.

e) Send the *Antimagic Label matrix* of this *antimagic* Path graph along with notifying the scheme of encryption from among the 4 schemes in above step and send this information to the receiver.

Antimagic Cycle:

a) Label the vertices of Cycle of even number of vertices/edges with *permutation 2* to obtain *antimagic labeling* and with *permutation 1* for Cycle with odd number of vertices/edges taking the labels are clockwise.

b) Assign the 1st letter of message as M_1 and so on. Assign these letters of the plain text to the induced edge *antimagic* labels clockwise.

c) Prepare the *Antimagic Label Matrix* for the Cycle graph.

d) Four schemes for encryption are being presented here to obtain the *cipher text*:

Scheme (**a**):

$C_i = M_i$ + (vertex label to the right of M_i) for $i = 1$ to $(n - 1)$.

Where M_i is the i^{th} letter of the *plain text* and C_i is the i^{th} letter of the *cipher text*.

Scheme (**b**):

$C_i = M_i$ + (vertex label to the left of M_i) for $i = 1$ to $(n - 1)$.

Where M_i is the i^{th} letter of the *plain text* and C_i is the i^{th} letter of the *cipher text*.

Scheme (**c**):

$C_i = M_i$ + (sum of right and left vertex labels) for $i = 1$ to $(n - 1)$.

Scheme (**d**):

$C_i = M_i$ + (absolute value of difference of right and left vertex labels) for $i = 1$ to $(n - 1)$.

e) Send the *Antimagic Label Matrix* of this Cycle graph along with notifying the scheme of encryption from among the four schemes in above step and send this information to the receiver.

2.3 Computer Program for the Encryption Technique:

(A) Antimagic Path (Upper Case Letter):

```
//Code for ANTIMAGIC Path labeled in lexicographic order (Upper case letters).

#include<iostream>
#include<stdio.h>
#include<string.h>
#include<math.h>
using namespace std;
int main()
{
int edge[100], ver[100], sum[100], diff[100], sw, n;
char rstr[100], lstr[100], add[100], mod[100];
cout<< "This program computes the cipher text using lexicographic order for antimagic paths (permutation 1 is used for labeling in path having odd number of vertices/even number of edges and permutation 2 is used for labeling in path having even number of vertices/odd number of edges to maintain antimagic labeling).\n\n";
cout<< "In this programme, 4 schemes have been presented to find cipher text:\n";
cout<< "1. Adding the vertex label of the right side to the character of the message.\n";
cout<< "2. Adding the vertex label of the left side to the character of the message.\n";
cout<< "3. Adding sum of right & left vertex label to the character of the original message.\n";
cout<< "4. Adding absolute value of the difference of right & left vertex label to the character of the original message.\n\n";
```

```
        cout<< "This program is for Upper case letters.\n\n";
        cout << "Enter the number of character (NO. OF EDGES) in message? = ";
        cin >> n;
        cout << "Enter the numbers from 1 to number of edges: ";
        edge[0]=0;
        for (int i = 1; i<n+1; ++i)
        {
                cin >> edge[i];
        }
        if(n%2==1)
        {
        sw = edge[n];
        edge[n]=edge[n-1];
        edge[n-1]=sw;
        }
        cout << "\nThe edge labelings are: ";
// Print edges
        for (int i = 1; i<n+1; ++i)
        {
        cout << edge[i] << "  ";
        }
        cout << "\nThe vertex labelings are: ";
// Print vertices
        edge[n+1]=edge[0];
        for (int i = 0; i < n+1; ++i)
        {
        ver[i+1]=edge[i]+edge[i+1];
        cout << ver[i+1] << "  ";
        }
        cout << "\nENTER THE ORIGINAL MESSAGE STRING IN UPPER CASE =\n";
        cin >>lstr;
        strcpy(rstr, lstr);
        strcpy(add, lstr);
        strcpy(mod, lstr);
        cout<<"OUTPUT CIPHER MESSAGE =\n";
        for(int i=0; lstr[i]!= '\0'; i++)
        {
```

```
ver[i+1]=ver[i+1]%26;
ver[i+2]=ver[i+2]%26;
lstr[i]=lstr[i]+ver[i+1];
rstr[i]=rstr[i]+ver[i+2];
sum[i]=ver[i+1]+ver[i+2];
sum[i]=sum[i]%26;
add[i]=add[i]+sum[i];
diff[i]=ver[i+1]-ver[i+2];
diff[i]=diff[i]%26;
mod[i]=mod[i]+abs(diff[i]);
if ((lstr[i]<= 90)&&(lstr[i]>=65))
{
lstr[n+2]=lstr[i];
}
else
{
        lstr[i] =65+lstr[i]-91;
        lstr[n+2]=lstr[i];
        }
if ((rstr[i]<= 90)&&(rstr[i]>=65))
{
        rstr[n+2]=rstr[i];
}
else
{
        rstr[i] =65+rstr[i]-91;
                rstr[n+2]=rstr[i];
}
if ((add[i]<= 90)&&(add[i]>=65))
{
        add[n+2]=add[i];
}
else
{
        add[i] =65+add[i]-91;
        add[n+2]=add[i];
}
```

```
        if ((mod[i]<= 90)&&(mod[i]>=65))
        {
                mod[n+2]=mod[i];
        }
        else
        {
                mod[i] =65+mod[i]-91;
                mod[n+2]=mod[i];
        }}
        cout <<"Cipher Text for Scheme 1 is: " << rstr<< endl;
        cout <<"Cipher Text for Scheme 2 is: " << lstr<< endl;
        cout <<"Cipher Text for Scheme 3 is: " << add<< endl;
        cout <<"Cipher Text for Scheme 4 is: " << mod<< endl;
return 0; }
```

(B) Antimagic Cycle (Upper Case Letter):

```
//Code for ANTIMAGIC Cycle labeled in lexicographic order (Upper case letters).

#include<iostream>
#include<stdio.h>
#include<string.h>
#include<math.h>
using namespace std;
int main()
{
    int edge[100], ver[100], sum[100], diff[100], sw, n, g;
    char rstr[100], lstr[100], add[100], mod[100];
    cout<< "This program computes the cipher text using lexicographic order for antimagic
cycle \n\n";
    cout<< "In this programme, 4 schemes have been presented to find cipher text:\n";
    cout<< "1. Adding the vertex label of the right side to the character of the message.\n";
    cout<< "2. Adding the vertex label of the left side to the character of the message.\n";
    cout<< "3. Adding sum of right & left vertex label to the character of the original
message.\n";
    cout<< "4. Adding absolute value of the difference of right & left vertex label to the
character of the original message.\n\n";
```

```
        cout<<"This program is for Upper case letters.\n\n";
        cout << "Enter the number of characters (NO. OF EDGES) in Original message? = ";
        cin >>n;
        cout << "Enter the numbers from 1 to number of characters in Original message: ";
        for (int i = 1; i<n+1; ++i)
        {
        cin >> edge[i];
        }
        if(n%2!=1)
        {
        sw = edge[n];
        edge[n]=edge[n-1];
        edge[n-1]=sw;
        }
        cout << "The edge labelings are: ";
// Print edge
        for (int i = 1; i<n+1; ++i) {
        cout << edge[i] << "  ";
        }
        cout << "\nThe vertex labelings are: ";
// Print vertices
        edge[n+1]=edge[1];
        for (int i = 1; i < n+1; ++i) {
        ver[i]=edge[i]+edge[i+1];
        cout << ver[i] << "  ";
        }
        ver[0]=ver[n];
    cout << "\nENTER THE ORIGINAL MESSAGE STRING IN UPPER CASE =\n";
        cin >>lstr;
        strcpy(rstr, lstr);
        strcpy(add, lstr);
        strcpy(mod, lstr);
        cout<<"OUTPUT CIPHER MESSAGE =\n";
        for(int i=0; lstr[i]!= '\0'; i++)
        {
                ver[i]=ver[i]%26;
                ver[i+1]=ver[i+1]%26;
```

```
lstr[i]=lstr[i]+ver[i];
rstr[i]=rstr[i]+ver[i+1];
add[i]=add[i]+(ver[i]+ver[i+1])%26;
diff[i]=ver[i]-ver[i+1];
diff[i]=diff[i]%26;
mod[i]=mod[i]+abs(diff[i]);
if ((lstr[i]<= 90)&&(lstr[i]>=65))
{
        lstr[n+1]=lstr[i];
}
else
{
        lstr[i] =65+lstr[i]-91;
        lstr[n+1]=lstr[i];
}
if ((rstr[i]<= 90)&&(rstr[i]>=65))
{
        rstr[n+1]=rstr[i];
}
else
{
        rstr[i] =65+rstr[i]-91;
        rstr[n+1]=rstr[i];
}
if ((add[i]<= 90)&&(add[i]>=65))
{
        add[n+1]=add[i];
}
else
{
        add[i] =65+add[i]-91;
        add[n+1]=add[i];
}
if ((mod[i]<= 90)&&(mod[i]>=65))
{
        mod[n+1]=mod[i];
}
```

```
        else
        {
                mod[i] =65+mod[i]-91;
                mod[n+1]=mod[i];
        }}
        cout <<"Cipher Text for Scheme 1 is: " << rstr<< endl;
        cout <<"Cipher Text for Scheme 2 is: " << lstr<< endl;
        cout <<"Cipher Text for Scheme 3 is: " << add<< endl;
        cout <<"Cipher Text for Scheme 4 is: " << mod<< endl;
return 0; }
```

2.4 Outputs of the Computer Programs:

Here we have taken two examples one for Antimagic Path and another for Antimagic Cycle.

Ex. 1. Antimagic Path *G*(6, 5):

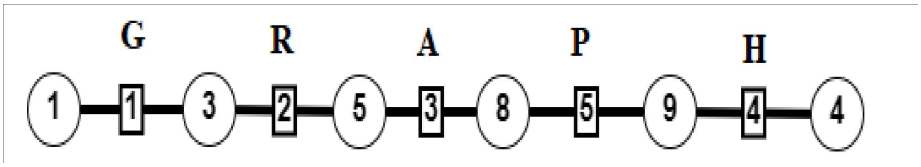

Figure 2.4.1: Antimagic Path *G*(6, 5) with even number of vertices with Permutation 2 of edge labels and assigned the letters G, R, A, P, H

Plain Text: GRAPH

Cipher Texts: JWIYL (Scheme 1)

HUFXQ (Scheme 2)

KZNGU (Scheme 3)

ITDQM (Scheme 4)

	1	3	5	8	9	4
1	0	1	0	0	0	0
3	1	0	2	0	0	0
5	0	2	0	3	0	0
8	0	0	3	0	5	0
9	0	0	0	5	0	4
4	0	0	0	0	4	0

Figure 2.4.2: *Antimagic* Label Matrix for chosen Path

Ex. 2. Antimagic Cycle:

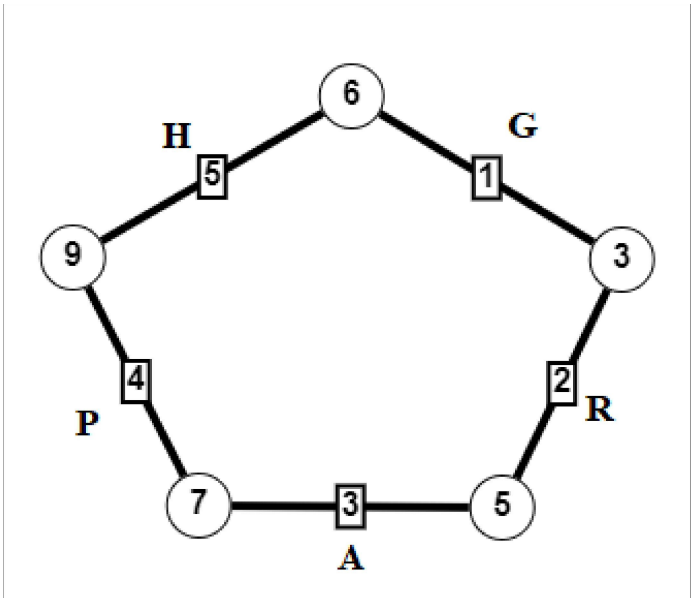

Figure 2.4.3: Antimagic odd Cycle with letters of message G, R, A, P, H assigned clockwise to the edge labels with Permutation 1

Plain Text: GRAPH

Cipher Texts: JWHYN (Scheme 1)

MUFWQ (Scheme 2)

PZMFW (Scheme 3)

JTCRK (Scheme 4)

	3	5	7	9	6
3	0	2	0	0	1
5	2	0	3	0	0
7	0	3	0	4	0
9	0	0	4	0	5
6	1	0	0	5	0

Figure 2.4.4: *Antimagic* Label Matrix for chosen Cycle

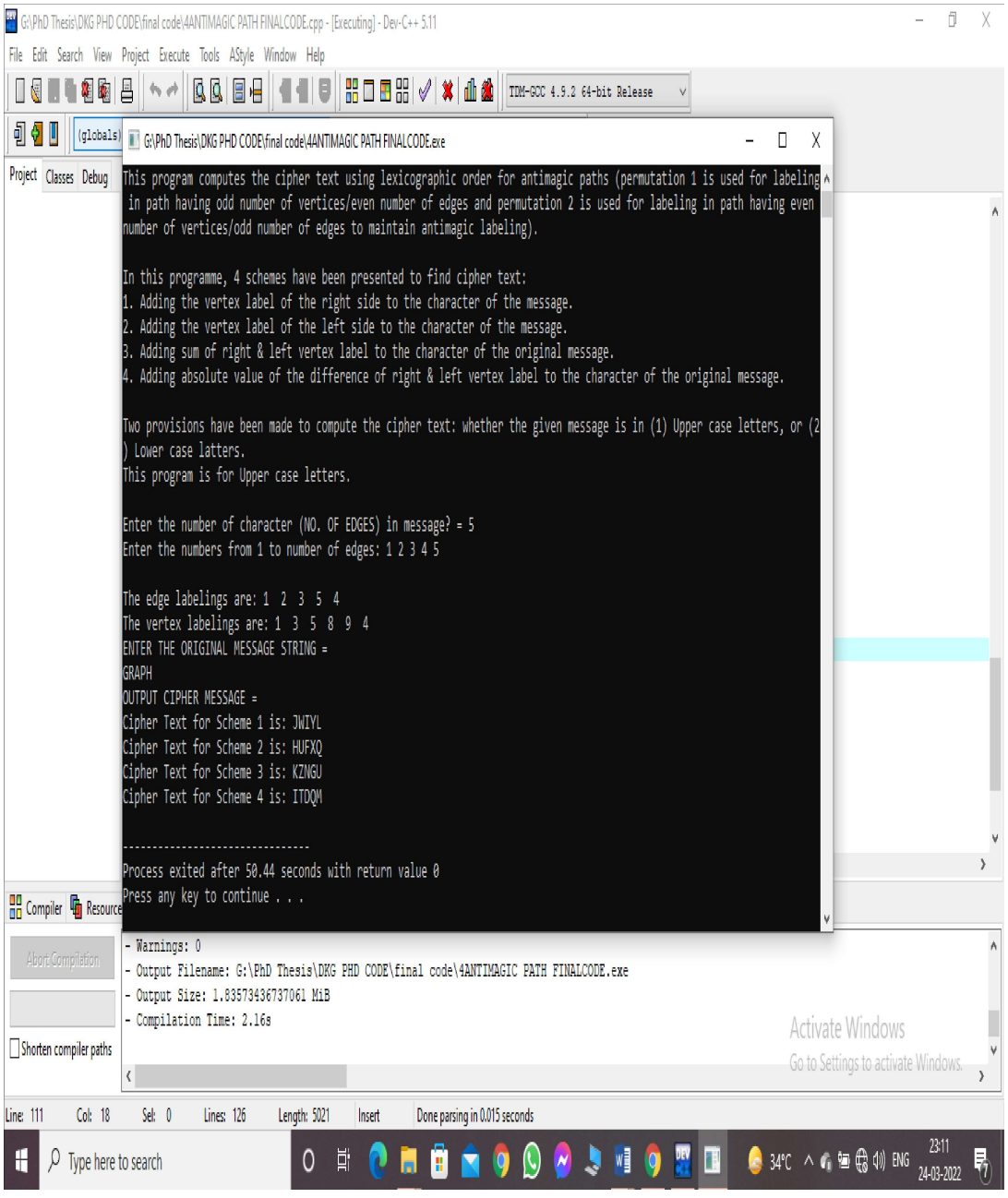

Figure 2.4.5: Code result for Antimagic Path $G(6, 5)$ with Plain Text 'GRAPH'

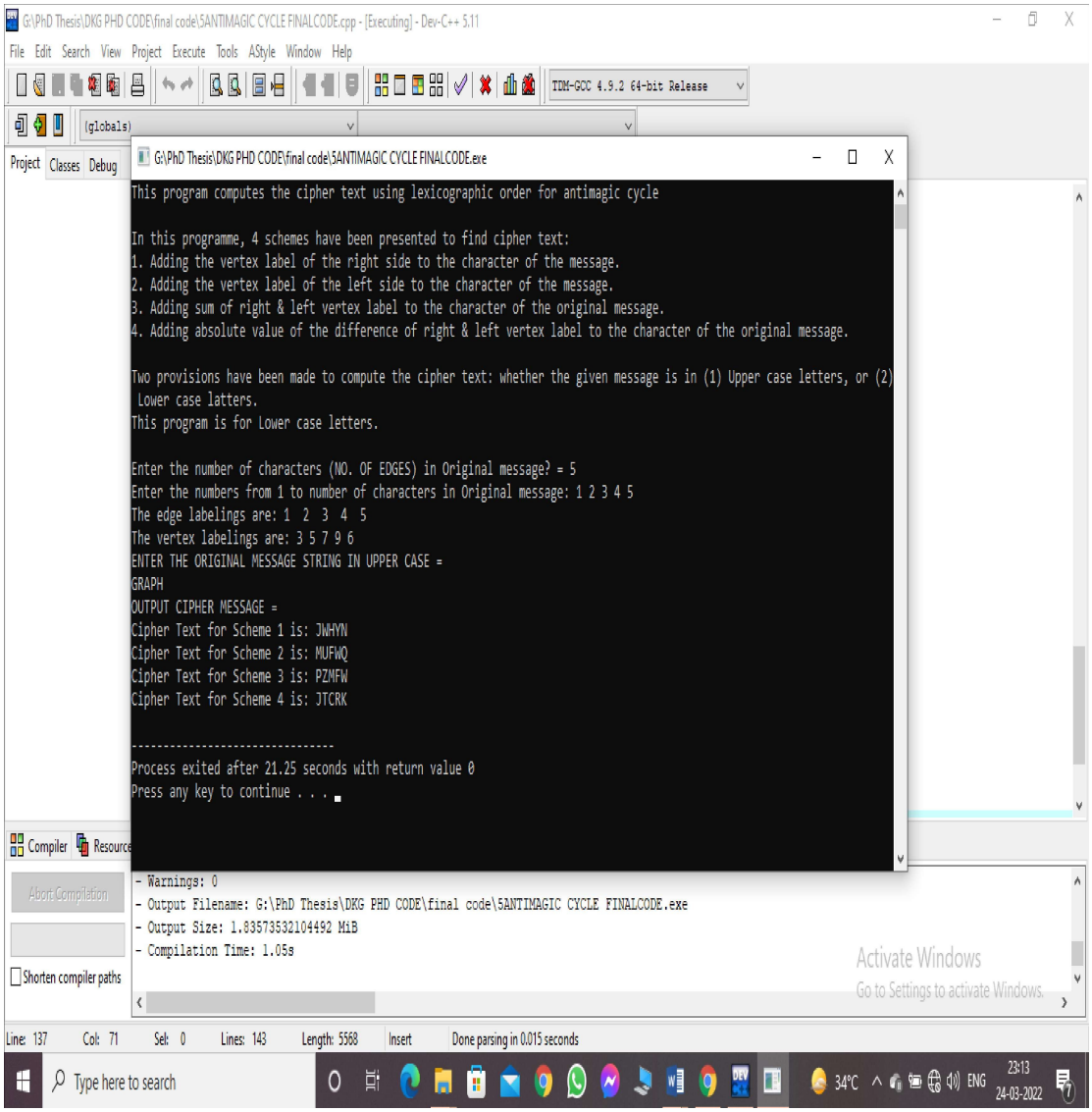

Figure 2.4.6: Code result for Antimagic Cycle with Plain Text 'GRAPH'

3. Results and discussion:

Output Cipher texts for some plain texts are being mentioned here in table given below for antimagic path and antimagic cycle.

Table 3.1: Cipher Texts for given Plain Text in different Cases

Plain Text (n is the number of character in Plain text)	Scheme	Cipher Text	
		Antimagic Path	Antimagic Cycle
GRAPH ($n = 5$)	1	JWIYL	JWHYN
	2	HUFXQ	MUFWQ
	3	KZNGU	PZMFW
	4	ITDQM	JTCRK
ENCRYPTION ($n = 10$)	1	HSJAJCIZHX	HSJAJCIAHX
	2	FQHYHAGXFG	OQHYHAGXGG
	3	IVOHSNVOYQ	RVOHSNVPZQ
	4	GPETARVKQW	LPETARVLPW
CRYPTO-GRAPHY PROCESS ($n = 19$)	1	FWFYEBVITKEXQU TJODK	FWFYEBVITKEXQUT JNDM
	2	DUDWCZTGRICVO SRHLCD	WUDWCZTGRICVO SRHLBD
	3	GZKFNMIXKDZUP VWOVNV	ZZKFNMIXKDZUP VWOUMX
	4	ETARVQITCRJAN TQEHTZ	TTARVQITCRJAN TQEGUB
LEXICOGRA-PHIC LABELED GRAPHS ($n = 26$)	1	OJERNBVITKEHDO FINWRSXKVMGS	OJERNBVITKEHD OFINWRSXKVNGS
	2	MHCPLZTGRICFBM DGLUPQVITKER	LHCPLZTGRICFBM DGLUPQVITKFR
	3	PMJYWMIXKDZECP INUFCFMBOHDR	OMJYWMIXKDZEC PINUFCFMBOIER
	4	NGZKEQITCRJKAN CDGNGFITCRJR	OGZKEQITCRJKA NCDGNGFITCSIR
ANTIMAGIC GRAPHS USED IN ENCRYPTION ($n = 31$)	1	DSARXNVZVBOZQKX BBPQXEXIZQZSYQXR	DSARXNVZVBOZQKX BBPQXEXIZQZSYPXT
	2	BQYPVLTXTZMXOIVZZ NOVCVGXOXQWNWW	GQYPVLTXTZMXOIVZZ NOVCVGXOXQWNVW
	3	EVFYGYIOMUJWPLAG IYBKTOBUNYTBVFA	JVFYGYIOMUJWPLAG IYBKTOBUNYTBUEC
	4	CPVKOCIKEITCNJUW UGFKPGPETWRVLPS	DPVKOCIKEITCNJUW UGFKPGPETWRVKQQ

Similarly, we can make computer programs for encryption techniques converting plain text message in small case letters to cipher text message in small case letters also.

4. Conclusion:

This paper has presented the computer programs developed using C++ for *antimagic* Paths and *antimagic* Cycle graphs employing the *lexicographic order* of permutations of edge labels. The outputs of the computer programs also have been presented. Manual calculations take a lot of time when the graphs are large so these computer programs can be used for large graphs as well when the message to be transferred is large thus saving a lot of time as well along with ease of computing the Cipher-texts.

Acknowledgement:

The INSPIRE Fellowship has been provided to the first author to carry out his Ph.D. work by the Department of Science and Technology (DST), Ministry of Science and Technology, Government of India under the supervision of the second author.

References:

1. Hartsfield N. and Ringel G. (1990) : "Pearls in Graph Theory : A Comprehensive Introduction", *Academic Press*, Boston.
2. Balakrishnan R. and Ranganathan K. (2012) : " A Textbook of Graph Theory", *Springer.*
3. Antonius Cahya Prihandoko, D. Dafik and Ika Hesti Agustin (2019) : "Implementation of super *H*-antimagic total graph on establishing Stream cipher", Indonesian Journal of Combinatorics, 3(**1**), 14-23.
4. C.L. Liu (1978) : "Discrete Mathematics", *Tata McGraw Hill*, India.
5. Auparajita Krishnaa and M.S. Dulawat (2009) : "Lexicographic Ordering in Graph Labellings of Cycles, Paths and Complete Bipartite Graphs", *South East Asian Journal of Mathematics and Mathematical Sciences*, 7(**2**), 87-93.

6. Auparajita Krishnaa (2001) : "Computer Modelling of Graph Labellings", proc. National Conference on Mathematical and Computational Models, 293-301, Coimbatore, India, *Allied Publishers*, India.

7. Auparajita Krishnaa (2012) : "On the Use of Computers in Graph labeling"; *International Journal of Computer Science and Communication*, 3(**1**), 191-197.

8. Dharmendra Kumar Gurjar and Auparajita Krishnaa (2021) : "Lexicographic Labeled Graphs in Cryptography"; *Advances and Applications in Discrete Mathematics*, 27(**2**); 209-232. doi: http://dx.doi.org/10.17654/DM027020209

9. Herbert Schildt : "The Complete Reference C++", Third Edition, *Tata-McGraw Hill*, India, 1999.

10. B. Vasuki, L. Shobana and B. Roopa (2022) : "Data Encryption Using Face Antimagic Labeling and Hill Cipher", *Mathematics and Statistics*, 10(**2**), 431-445.

11. Auparajita Krishnaa (2024) : "Some Applications of Antimagic Graphs in Cryptology", *The Mathematics Education*, LVIII(**1**), 1-10.

12. Auparajita Krishnaa (2025) : "Complex Cipher Texts from Complete Graphs Including by using Sanskrit in Cryptology", *Arya Bhatta Journal of Mathematics and Informatics*, 17(**1**), 19-38.