# Music Liking Prediction System & Graph Algorithms

An Industrial Training Report

Submitted in Partial Fulfilment of the Requirement for the Award of Degree of

**BACHELOR OF ENGINEERING**

**(Computer Science & Engineering Department)**

To

**HIMACHAL PRADESH TECHNICAL UNIVERSITY, HAMIRPUR**

**Submitted by**

**Gaurav Kumar**

**Roll Number:17BT020314**

**Semester:5th**

**Submitted to:**

**CSE Department**

**COMPUTER SCIENCE & ENGINEERING DEPARTMENT**

ATAL BIHARI VAJPAYEE GOVERNMENT INSTITUTE OF ENGINEERING & TECHNOLOGY, PRAGATINAGAR DISTT. SHIMLA (H.P)-171202 INDIA

2019

# DECLARATION

I **Gaurav Kumar**, student of Bachelor of Technology (B. Tech.) Computer Science & Engineering Department, Atal Bihari Vajpayee Government Institute of Engineering and Technology, Pragatinagar Shimla, having **17BT020314**, session 2017-2021, hereby declare that the Project entitled "**Music Liking Prediction System & Graph Algorithms** ", submitted to the Computer Science & Engineering Department as a partial fulfillment of the requirements for the award of the degree of Bachelor of Technology, has been carried out by me under the guidance of **ER. Anurag Singh at Guru Nanak Dev Engineering College, Ludhiana**. The **Project** is an original piece of work and has not formed the basis for the award of any other degree of any other college.

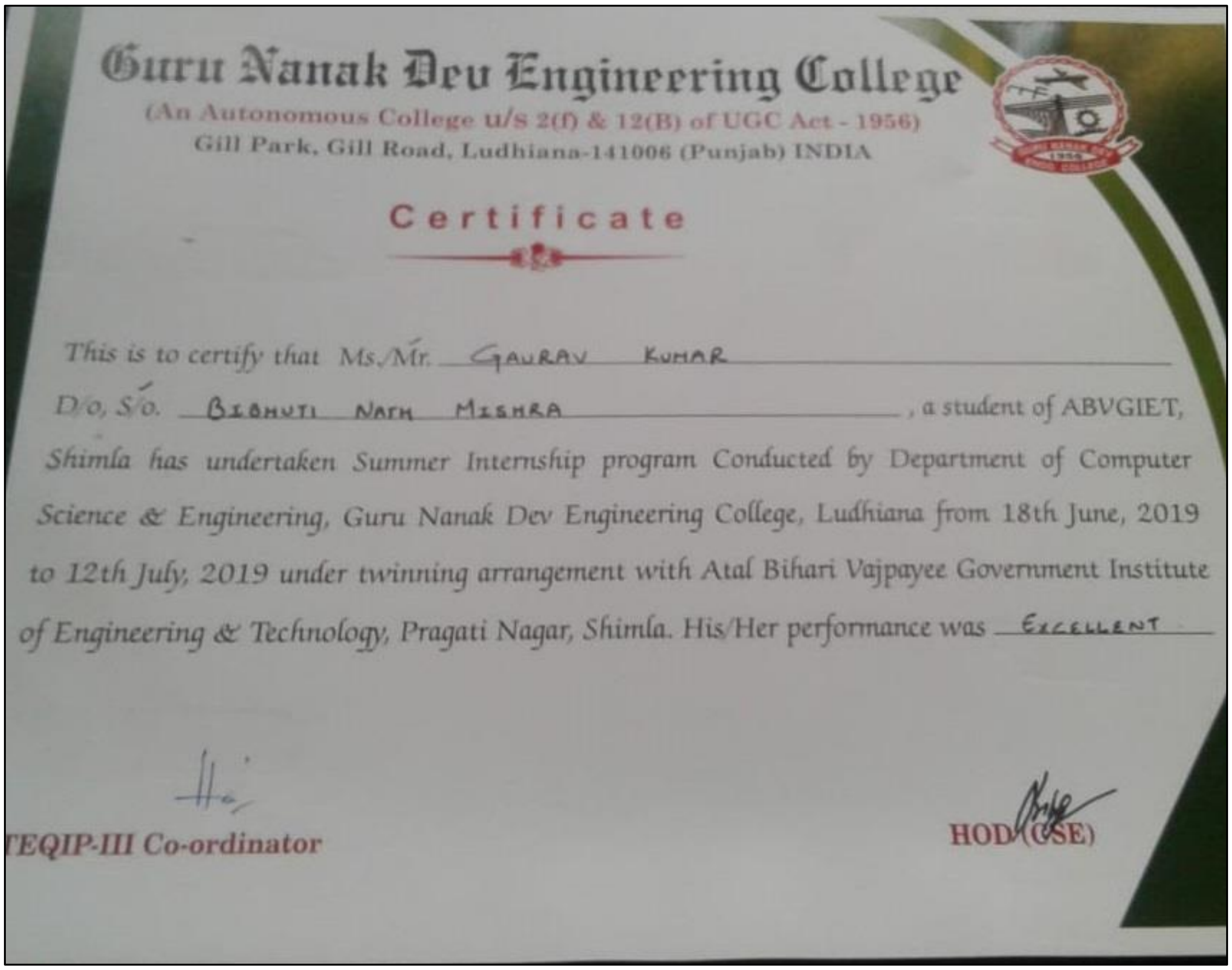
Dated: 24-11-2019

Gaurav Kumar

B. TECH (CSE)

Roll No.17BT020314

# Certificate for Industrial Training



Guru Nanak Dev Engineering College

(An Autonomous College u/s 2(f) & 12(B) of UGC Act - 1956)

Gill Park, Gill Road, Ludhiana-141006 (Punjab) INDIA

Certificate

This is to certify that Ms./Mr. GAURAV KUMAR D/o, S/o. BIBHUTI NATH MISHRA, a student of ABVGIET, Shimla has undertaken Summer Internship program Conducted by Department of Computer Science & Engineering, Guru Nanak Dev Engineering College, Ludhiana from 18th June, 2019 to 12th July, 2019 under twinning arrangement with Atal Bihari Vajpayee Government Institute of Engineering & Technology, Pragati Nagar, Shimla. His/Her performance was EXCELLENT

[illegible]
TEQIP-III Co-ordinator

[illegible]
HOD (CSE)

# ACKNOWLEDGEMENT

I would first like to thank my project advisor **ER. Anurag Singh** of the at **Guru Nanak Dev Engineering College, Ludhiana** for the continuous support of my project, for his/her patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of development and writing of this project. I could not have imagined having a better advisor and mentor for my project work. He consistently allowed this project to be my own work, but steered me in the right direction whenever he thought I needed it.

Besides my advisor, I would like to thank the rest of my faculty members: ER. Ravi Kumar, ER. Anurag Sharma, ER. Shivani Thakur, ER Nivedita Kashyap, ER. Rahul Pal Singh, ER. Navdeep Sharma, ER. Namita Chandel for their encouragement, insightful comments, and hard questions.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this project work. This accomplishment would not have been possible without them. Thank you.

Gaurav Kumar

B. TECH (CSE)

Roll Number:17BT020314

# ABSTRACT

This project consists of two modules: A Music Liking Prediction System and Implementation of Graph algorithms.

**Music Liking Prediction System** is a prediction system which can suggest which genre of music a person will like given his/her age and gender. This system can be used in music apps like Sportify,Ganna.com etc. This system can be integrated with the suggestion system of the stated apps to help in deciding which genre of music should be suggested to the users based on age and gender. This will improve the suggestion quality and will improve user experience. The prediction system is built using decision tree which is a machine learning algorithm.

This project also includes implementation of two famous graph algorithms namely Dijkstra's Algorithm and Algorithm to find Eulerian Path & Circuit in a graph. The implementation helps in better understanding of how does geographical map apps work and how does airplane navigation system suggest a pilot to take a route so that he/she does not have to retrace any of the previously taken route between two airports.

# LIST OF FIGURES

# LIST OF ABBRIVIATIONS

| SYMBOL | ABBRIVIATION |
|---|---|
| PANDAS | Python Data Analysis Library |
| DFD | Data Flow Diagram |
| CSV | Comma Separated Values |

# CONTENTS

# Chapter 1

## INTRODUCTION TO PYTHON

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3. Due to concern about the amount of code written for Python 2, support for Python 2.7 (the last release in the 2.x series) was extended to 2020. Language developer Guido van Rossum shouldered sole respon5sibility for the project until July 2018 but now shares his leadership as a member of a five-person steering council.

Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution(late binding), which binds method and variable names during program execution.

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

Python strives for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology. In contrast to Perl's "there is more than one way to do it" motto, Python embraces a "there should be one—and preferably only one—obvious way to do it" design philosophy.

Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of the CPython reference implementation that would offer marginal increases in speed at the cost of clarity. When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

## Pandas (Python Data Analysis Library)

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

### Library Features

- Data Frame object for data manipulation with integrated indexing.
- Tools for reading and writing data between in-memory data structures and different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of data sets.
- Label-based slicing, fancy indexing, and sub setting of large data sets.
- Data structure column insertion and deletion.
- Group by engine allowing split-apply-combine operations on data sets.
- Data set merging and joining.
- Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.
- Time series-functionality: Date range generation[4] and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging.
- Provides data filtration.

## Scikit-Learn

Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

# Chapter 2

## INTRODUCTION TO PROJECT

### Music Liking Prediction System

The Music Liking Prediction System is a prediction system which can suggest which kind of music a person will like given his/her age and gender.

This is a data analysis system implemented using decision tree which is a type of supervised learning algorithm.

It takes age and gender of people as input and predict which genre of music they will like.

- **Input**-Age, Gender
- **Output**-Genre

### Data Set for the model

Format

| Age | Gender | Genre |
|---|---|---|

Following is a sample dataset which can be used to train the model

| age | gender | genre | | | | |
|---|---|---|---|---|---|---|
| 20 | 1 | HipHop | | | | |
| 23 | 1 | HipHop | | | | |
| 25 | 1 | HipHop | | | | |
| 26 | 1 | Jazz | | | | |
| 29 | 1 | Jazz | | | | |
| 30 | 1 | Jazz | | | | |
| 31 | 1 | Classical | | | | |
| 33 | 1 | Classical | | | | |
| 37 | 1 | Classical | | | | |

# Dijkstra’s shortest path algorithm

Given a graph and a source vertex in the graph, Dijkstra’s algorithm finds the shortest paths from source to all vertices in the given graph.

Below are the detailed steps used in Dijkstra’s algorithm to find the shortest path from a single source vertex to all other vertices in the given graph.

## Algorithm

1. Create a set sptSet (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.

2. Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.

3. While sptSet doesn’t include all vertices
   ….a) Pick a vertex u which is not there in sptSet and has minimum distance value.
   ….b) Include u to sptSet.
   ….c) Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

## Example

The set *sptSet* is initially empty and distances assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. The vertex 0 is picked, include it in *sptSet*. So *sptSet* becomes {0}. After including 0 to *sptSet*, update distance values of its adjacent vertices. Adjacent vertices of 0 are 1 and 7. The distance values of 1 and 7 are updated as 4 and 8. Following subgraph shows vertices and their distance values, only the vertices with finite distance values are shown. The vertices included in SPT are shown in green colour.

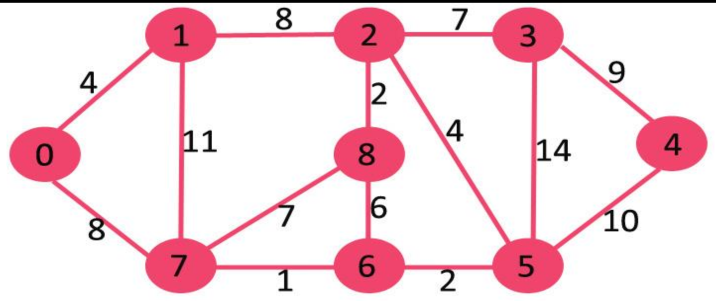


**Fig 1**

Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). The vertex 1 is picked and added to sptSet. So sptSet now becomes {0, 1}. Update the distance values of adjacent vertices of 1. The distance value of vertex 2 becomes 12.

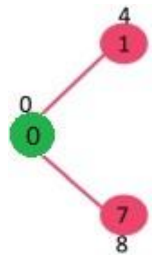


**Fig 2**

Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). The vertex 1 is picked and added to sptSet. So sptSet now becomes {0, 1}. Update the distance values of adjacent vertices of 1. The distance value of vertex 2 becomes 12.

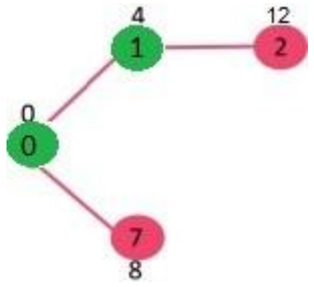


**Fig 3**

Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 7 is picked. So sptSet now becomes {0, 1, 7}. Update the distance values of adjacent vertices of 7. The distance value of vertex 6 and 8 becomes finite (15 and 9 respectively).

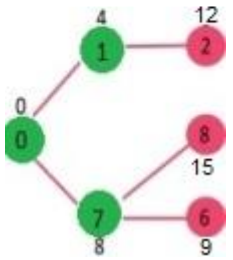


**Fig 4**

Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 6 is picked. So sptSet now becomes {0, 1, 7, 6}. Update the distance values of adjacent vertices of 6. The distance value of vertex 5 and 8 are updated.

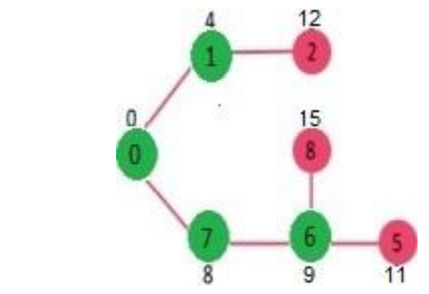


**Fig 5**

We repeat the above steps until *sptSet* doesn't include all vertices of given graph. Finally, we get the following Shortest Path Tree (SPT).

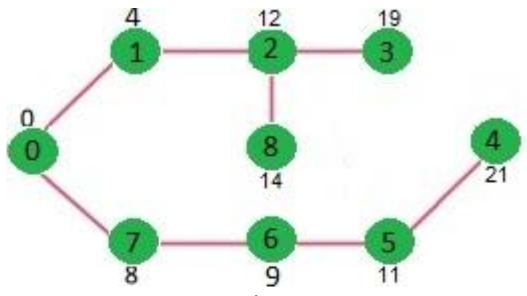


**Fig 6**

## <u>Eulerian path and circuit for undirected graph</u>

Eulerian Path is a path in graph that visits every edge exactly once. Eulerian Circuit is a Eulerian Path which starts and ends on the same vertex.

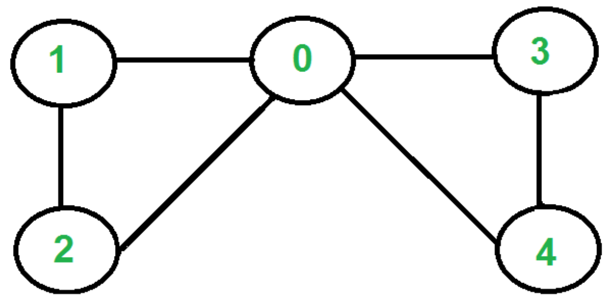
A graph is called Eulerian if it has a Eulerian Cycle and called Semi-Eulerian if it has a Eulerian Path.

### <u>How to find whether a given graph is Eulerian or not?</u>

#### Eulerian Cycle

An undirected graph has Eulerian cycle if following two conditions are true.

1. All vertices with non-zero degree are connected. We don't care about vertices with zero degree because they don't belong to Eulerian Cycle or Path (we only consider all edges).
2. All vertices have even degree.



The graph has Eulerian Cycles, for example "2 1 0 3 4 0 2"
Note that all vertices have even degree

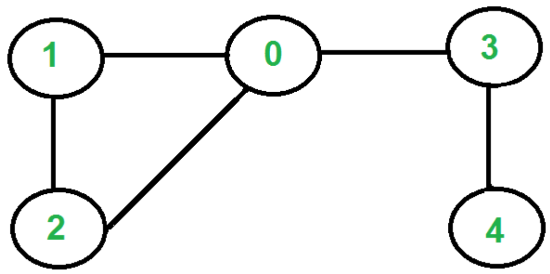
**Fig 7**

## Eulerian Path

An undirected graph has Eulerian Path if following two conditions are true.

1. All vertices with non-zero degree are connected. We don't care about vertices with zero degree because they don't belong to Eulerian Cycle or Path (we only consider all edges).

2. If two vertices have odd degree and all other vertices have even degree. Note that only one vertex with odd degree is not possible in an undirected graph (sum of all degrees is always even in an undirected graph).



The graph has Eulerian Paths, for example "4 3 0 1 2 0", but no Eulerian Cycle. Note that there are two vertices with odd degree (4 and 0)

**Fig 8**

# Chapter 3

## OBJECTIVE

The objective of the project is as follows:

1. To Implement basic graph algorithms using python.

   This project contain implementation of graph algorithms as stated below

   - Dijkstra’s Algorithm to find single source shortest path.
   - Algorithm to find Eulerian Cycle and Eulerian Circuit in a graph.

2. To build a **Music Liking Prediction System** using decision tree algorithm.

# Chapter 4

## DATA FLOW DIAGRAM

DFD For Music Liking Prediction System

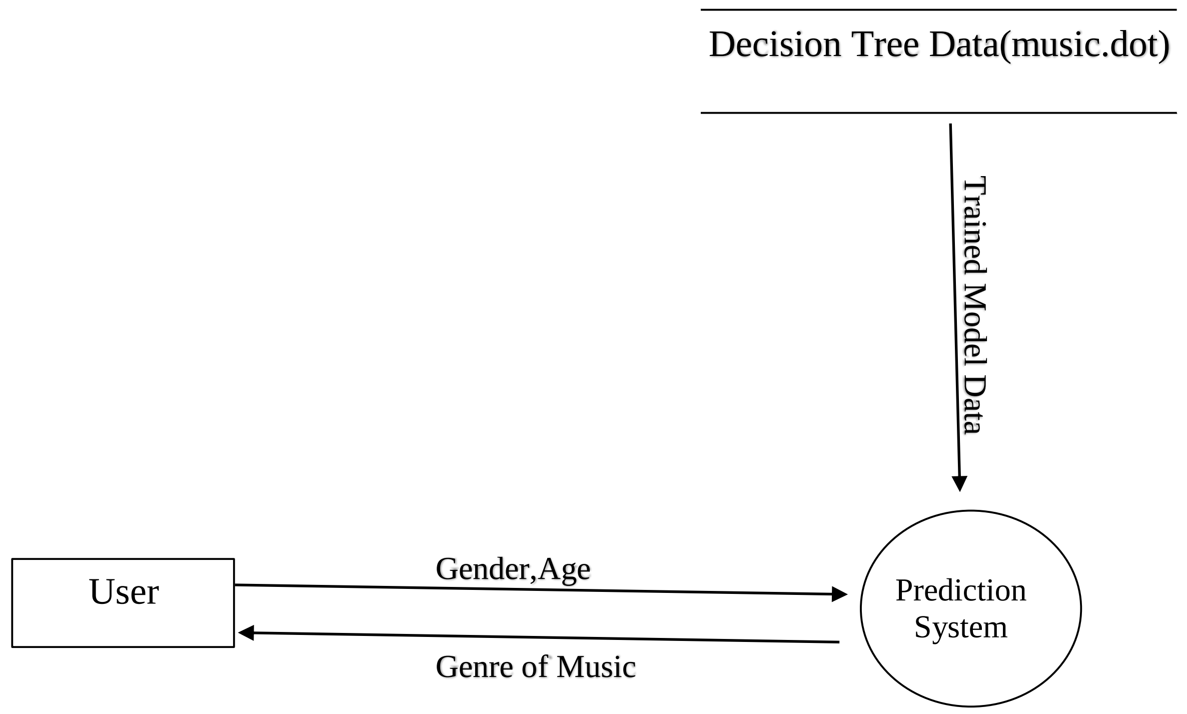


**Fig 9**

# Chapter 5

## SOFTWARE AND HARDWARE REQUIREMENT

### HARDWARE REQUIREMENTS

- RAM:
  - Required: 2GB
  - Recommended: 4 GB
- Processor:
  - Required: Intel Pentium 4
  - Recommended: Inter Core i-series (Developer’s system)

### SOFTWARES USED

- Programming Language: Python
- External Libraries: Scikit-Learn, Pandas

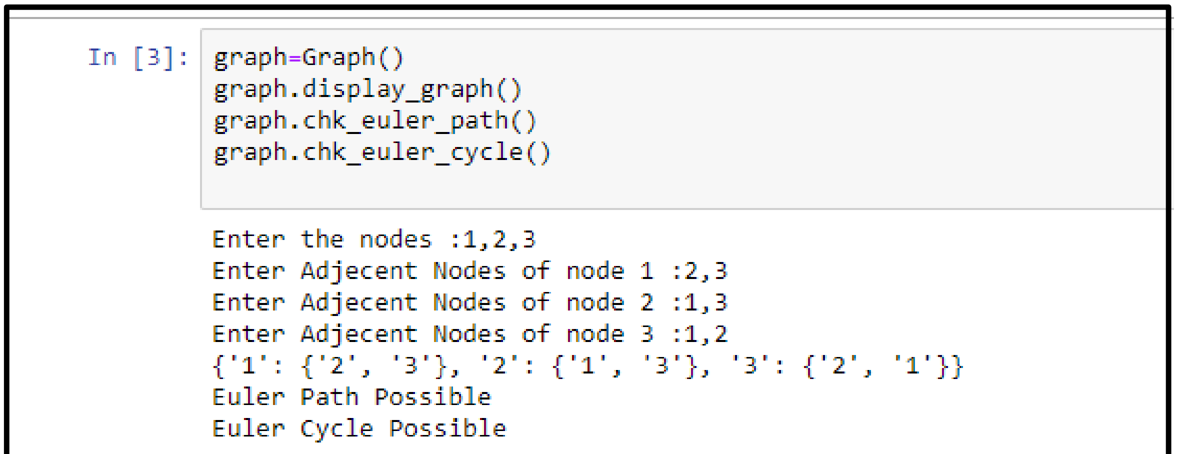
# Chapter 6

# PROJECT IMPLEMENTATION

## SCREENSHOTS & DISCRIPTION

### Detection of Eulerian path and circuit for undirected graph

1. First enter the nodes of the graph
2. Add adjacent nodes for each node in the graph
3. It will be displayed whether Eulerian Cycle and Eulerian Path is possible or not.



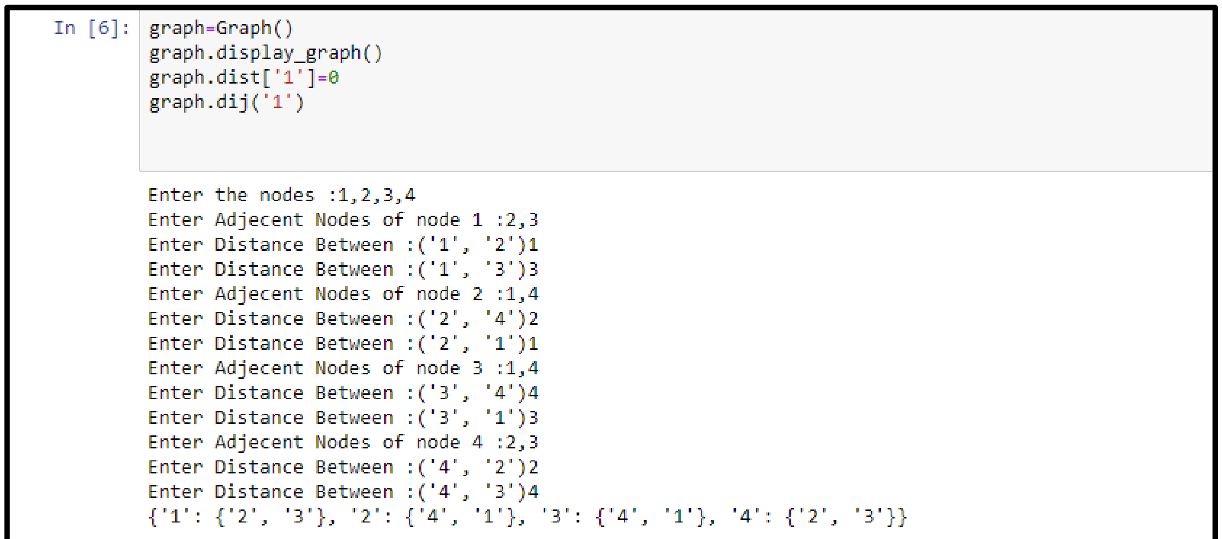
```
In [3]: graph=Graph()
        graph.display_graph()
        graph.chk_euler_path()
        graph.chk_euler_cycle()

        Enter the nodes :1,2,3
        Enter Adjecent Nodes of node 1 :2,3
        Enter Adjecent Nodes of node 2 :1,3
        Enter Adjecent Nodes of node 3 :1,2
        {'1': {'2', '3'}, '2': {'1', '3'}, '3': {'2', '1'}}
        Euler Path Possible
        Euler Cycle Possible
```

**Fig 10**

## Dijkstra’s shortest path algorithm for Undirected Graph

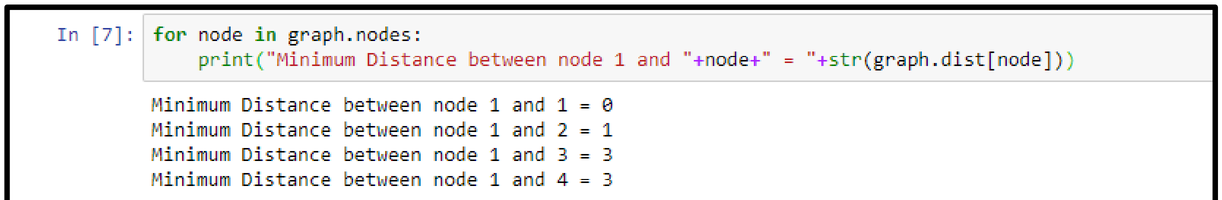
1. First enter the nodes of the graph.
2. Add adjacent nodes for each node in the graph.
3. Enter distance between a selected node and other nodes.
4. Repeat steps 2&3 for each node.
5. The minimum distance between source and each node will be displayed.



```
In [6]: graph=Graph()
        graph.display_graph()
        graph.dist['1']=0
        graph.dij('1')
```

```
Enter the nodes :1,2,3,4
Enter Adjecent Nodes of node 1 :2,3
Enter Distance Between :('1', '2')1
Enter Distance Between :('1', '3')3
Enter Adjecent Nodes of node 2 :1,4
Enter Distance Between :('2', '4')2
Enter Distance Between :('2', '1')1
Enter Adjecent Nodes of node 3 :1,4
Enter Distance Between :('3', '4')4
Enter Distance Between :('3', '1')3
Enter Adjecent Nodes of node 4 :2,3
Enter Distance Between :('4', '2')2
Enter Distance Between :('4', '3')4
{'1': {'2', '3'}, '2': {'4', '1'}, '3': {'4', '1'}, '4': {'2', '3'}}
```

**Fig 11**



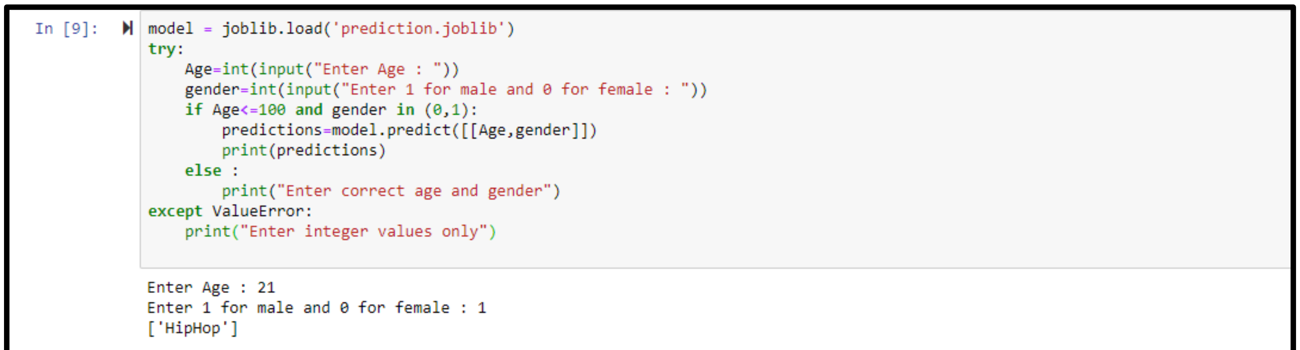
```
In [7]: for node in graph.nodes:
            print("Minimum Distance between node 1 and "+node+" = "+str(graph.dist[node]))
```

```
Minimum Distance between node 1 and 1 = 0
Minimum Distance between node 1 and 2 = 1
Minimum Distance between node 1 and 3 = 3
Minimum Distance between node 1 and 4 = 3
```

**Fig 12**

## Music Liking Prediction System

1. Enter age
2. Enter gender
3. Get suggestion of music genre.



```
In [9]:  model = joblib.load('prediction.joblib')
         try:
             Age=int(input("Enter Age : "))
             gender=int(input("Enter 1 for male and 0 for female : "))
             if Age<=100 and gender in (0,1):
                 predictions=model.predict([[Age,gender]])
                 print(predictions)
             else :
                 print("Enter correct age and gender")
         except ValueError:
             print("Enter integer values only")

         Enter Age : 21
         Enter 1 for male and 0 for female : 1
         ['HipHop']
```

**Fig 13**

## Generated Decision Tree

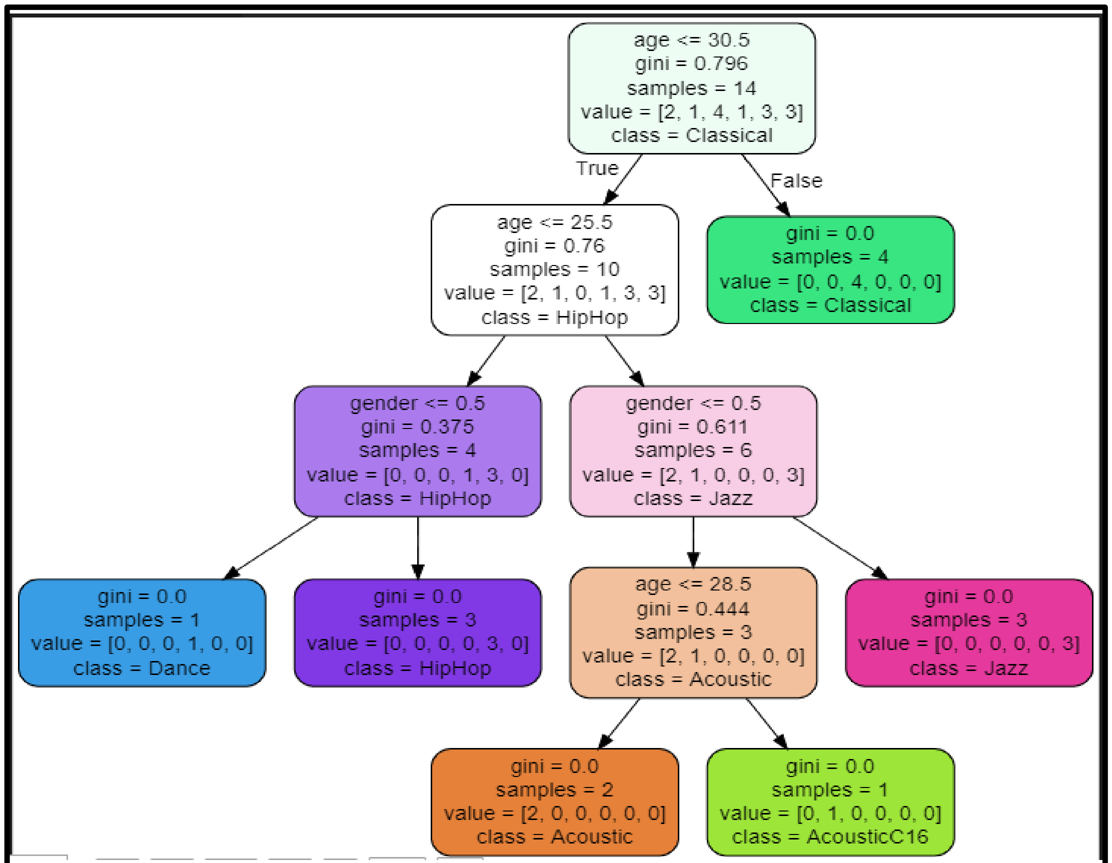


**Fig 14**

## CODING WITH DESCRIPTION

### Detection of Eulerian path and circuit for undirected graph

```
class Graph:
    def __init__(self):
        self.count_not_even=0
        self.visited={}
        self.Set=set()
        self.nodes=(input("Enter the nodes :")).split(',')
        self.Adjecent_Node_Dict={}
        for node in self.nodes:
            self.add_adjecent_node(node)

    def add_node(self,node):
        self.nodes.append(node)
        self.add_adjecent_node(node)

    def add_adjecent_node(self,node):
        adjecent_nodes=set((input("Enter Adjecent Nodes of node %s :"%node)).split(','))
        self.Adjecent_Node_Dict[node]=adjecent_nodes
        for ad_node in adjecent_nodes:
            try:
                self.Adjecent_Node_Dict[ad_node].add(node)
            except KeyError:
                self.Adjecent_Node_Dict[ad_node]={node,}
    def remove_adjecent_node(self,node,adjecent_node):
        self.Adjecent_Node_Dict[node].remove(adjecent_node)
        self.Adjecent_Node_Dict[adjecent_node].remove(node)

    def remove_node(self,node):
        for ad_node in self.Adjecent_Node_Dict[node]:
            print(ad_node)
            self.remove_adjecent_node(node,ad_node)
        self.Adjecent_Node_Dict.pop(node)

    def display_graph(self):
        print(self.Adjecent_Node_Dict)

    def chech_Connected(self):
        for node in self.nodes:
            self.set=set()

            for node in self.nodes:
                self.visited[node]=False
            if not(len(self.Adjecent_Node_Dict[node])==0):
```

```
            self.dfs(node)
            if len(self.Set)<len(self.nodes):
                print("%s is not connected to every other node "%node)
                return False
        return True

    def check_even(self):
        count_not_even=0
        for node in self.nodes:
            if not(len(self.Adjecent_Node_Dict[node])%2==0):
                count_not_even+=1
        return count_not_even

    def chk_euler_cycle(self):

        if (self.chech_Connected()and self.check_even()==0):
            print("Euler Cycle Possible")
        else:
            print("Euler Cycle not Possible")


    def chk_euler_path(self):
        if (self.chech_Connected()and ( self.check_even()==0 or self.check_even()==2)):
            print("Euler Path Possible")
        else:
            print("Euler Path not Possible")

    def dfs(self,node):
            if self.visited[node]==False:
                self.Set.add(node)
                self.visited[node]=True
                for ad_node in self.Adjecent_Node_Dict[node]:
                    self.dfs(ad_node)
```

## Dijkstra’s shortest path algorithm for Undirected Graph

```
class Graph:
    def __init__(self):
        self.selected={}
        self.dist={}
        self.cost_dict={}
        self.Set=set()
        self.nodes=(input("Enter the nodes :")).split(',')
        self.Adjecent_Node_Dict={}
        for node in self.nodes:
```

```
        self.add_adjecent_node(node)
        self.selected[node]=False
        self.dist[node]="inf"

    def add_node(self,node):
        self.nodes.append(node)
        self.add_adjecent_node(node)

    def add_adjecent_node(self,node):
        adjecent_nodes=set((input("Enter Adjecent Nodes of node %s :"%node)).split(','))
        self.Adjecent_Node_Dict[node]=adjecent_nodes
        for ad_node in adjecent_nodes:
            self.add_dist(node,ad_node)
            try:
                self.Adjecent_Node_Dict[ad_node].add(node)
            except KeyError:
                self.Adjecent_Node_Dict[ad_node]={node,}


    def remove_adjecent_node(self,node,adjecent_node):
        self.Adjecent_Node_Dict[node].remove(adjecent_node)
        self.Adjecent_Node_Dict[adjecent_node].remove(node)

    def remove_node(self,node):
        for ad_node in self.Adjecent_Node_Dict[node]:
            print(ad_node)
            self.remove_adjecent_node(node,ad_node)
        self.Adjecent_Node_Dict.pop(node)

    def display_graph(self):
        print(self.Adjecent_Node_Dict)

    def add_dist(self,node1,node2):
        edge=(node1,node2)
        edge_dist=int(input("Enter Distance Between :"+str(edge)))
        self.cost_dict[edge]= edge_dist

    def min_dist(self,node1,node2):
        if(self.dist[node2]=="inf" or (self.dist[node2]>self.dist[node1]+self.cost_dict[(node1,node2)])):
            self.dist[node2]=self.dist[node1]+self.cost_dict[(node1,node2)]
    def min_dist_node(self,node):
        ptr=None
        for ad_node in self.Adjecent_Node_Dict[node]:
            if(self.selected[ad_node]==False ):
                ptr=ad_node
                self.selected[ad_node]=True
                break
        for ad_node in self.Adjecent_Node_Dict[node]:
            if(self.selected[ad_node]==False ):
                if self.dist[ad_node]<self.dist[ptr]:
                    ptr=ad_node
                    self.selected[ad_node]=True
        return ptr

    def dij(self,node):
        for ad_node in self.Adjecent_Node_Dict[node]:
            if self.selected[ad_node]==False:
                self.min_dist(node,ad_node)
        ptr=self.min_dist_node(node)
        if not (ptr==None):
            self.dij(ptr)
```

## Music Liking Prediction System

**#Training The Model**

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd
from sklearn.externals import joblib
music_data = pd.read_csv('music.csv')
#creating input set
x=music_data.drop(columns=['genre'])
#creatnig output set
y=music_data['genre']
#Spliting Dataset
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
#Building a model
model=DecisionTreeClassifier()
model.fit(x_train,y_train)
joblib.dump(model,'prediction.joblib')
Prediction=model.predict(x_test)
accuracy=accuracy_score(y_test,Prediction)
Prediction=model.predict([[20,1],[21,0]])
print(Prediction)
print(accuracy)
```

## #Loading the trained model

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd
from sklearn.externals import joblib
```

## #Making Predictions

```
model = joblib.load('prediction.joblib')
try:
   Age=int(input("Enter Age : "))
   gender=int(input("Enter 1 for male and 0 for female : "))
   if Age<=100 and gender in (0,1):
      predictions=model.predict([[Age,gender]])
      print(predictions)
   else :
      print("Enter correct age and gender")
except ValueError:
   print("Enter integer values only")
```

## #Creating The Decision Tree File

```
tree.export_graphviz(model,out_file='music.dot',
               feature_names=['age','gender'],
               class_names=sorted(y.unique()),
               label='all',
               rounded=True,
                filled=True)
```

# Chapter 7

## CONCLUSION AND FUTURE SCOPE

In this project I have presented how we can implement Dijkstra’s algorithm and algorithm to find eulerian path and circuit in an undirected graph. I have also implemented decision tree algorithm to build a music liking prediction system. There are many areas where we can implement the graph algorithms like in computer based geographical map systems and in airplane navigation systems. The music liking prediction system can be used in music apps like Spotify and Ganna.com. The decision tree algorithm can be used to build other supervised learning models. The music liking prediction system can include more parameters like language, country etc. to give better results. We can also use more sophisticated machine learning algorithms like neural networks to train our model so that we get better

# Chapter-8

# REFERENCES

## Books Preferred:

- Python: The Complete Reference.
- Python Programming Fundamentals- A Beginner's Handbook.
- Natural Language Processing With Python.

## Online Sources:

- https://www.youtube.com/watch?v=_uQrJ0TkZlc&t=14157s
- https://goo.gl/P64rZ8
- http://bit.ly/30PPkaC