**Hamilton Path/Circuit & Minimum Hamilton Circuit**

**Year 3-CS department Algorithms – group 3 - 4**

**Dr. Moustafa Reda**

**Team Members**

Fatma Ashraf Ramadan 20170192

Email: fatma.ashraf.275@gmail.com

Ali Samy Mohamed 20170171

Mostafa Badr El-Sayed 20170286

Mona Mohamed Fathy 20170300

Mayada Mohamed Mahmoud 20170304

Mina Farid saad 20170310

Naglaa Essam Fathy 20170313

Youssef Ehab 20170339

Youssef Mohamed Mohamed 20170349

Aya Amr Mohamed 20170359

Contents

[Main purpose of the program 3](file:///F:\mona\My%20Fcai\Level_3_Sem_2\SE_2\Assignments\Project_Phase1\SE-II-2020-ProjectDescription-v1.0\Sprint%201\SE-II-2020-Daily%20scrum%20meeting%20template.docx#_Toc507250131)

[Hamilton Circuit 4](file:///F:\mona\My%20Fcai\Level_3_Sem_2\SE_2\Assignments\Project_Phase1\SE-II-2020-ProjectDescription-v1.0\Sprint%201\SE-II-2020-Daily%20scrum%20meeting%20template.docx#_Toc507250132)

[Hamilton Path 5](file:///F:\mona\My%20Fcai\Level_3_Sem_2\SE_2\Assignments\Project_Phase1\SE-II-2020-ProjectDescription-v1.0\Sprint%201\SE-II-2020-Daily%20scrum%20meeting%20template.docx#_Toc507250133)

[Minimum Hamilton Circuit 6](file:///F:\mona\My%20Fcai\Level_3_Sem_2\SE_2\Assignments\Project_Phase1\SE-II-2020-ProjectDescription-v1.0\Sprint%201\SE-II-2020-Daily%20scrum%20meeting%20template.docx#_Toc507250134)

[Algorithm for Hamilton Circuit 7](file:///F:\mona\My%20Fcai\Level_3_Sem_2\SE_2\Assignments\Project_Phase1\SE-II-2020-ProjectDescription-v1.0\Sprint%201\SE-II-2020-Daily%20scrum%20meeting%20template.docx#_Toc507250135)

[Algorithm for Hamilton Path 7](file:///F:\mona\My%20Fcai\Level_3_Sem_2\SE_2\Assignments\Project_Phase1\SE-II-2020-ProjectDescription-v1.0\Sprint%201\SE-II-2020-Daily%20scrum%20meeting%20template.docx#_Toc507250133)

Algorithm for [Minimum Hamilton Circuit 8](file:///F:\mona\My%20Fcai\Level_3_Sem_2\SE_2\Assignments\Project_Phase1\SE-II-2020-ProjectDescription-v1.0\Sprint%201\SE-II-2020-Daily%20scrum%20meeting%20template.docx#_Toc507250134)

[Class GraphGui 9](file:///F:\mona\My%20Fcai\Level_3_Sem_2\SE_2\Assignments\Project_Phase1\SE-II-2020-ProjectDescription-v1.0\Sprint%201\SE-II-2020-Daily%20scrum%20meeting%20template.docx#_Toc507250133)

[Class Edge 9](file:///F:\mona\My%20Fcai\Level_3_Sem_2\SE_2\Assignments\Project_Phase1\SE-II-2020-ProjectDescription-v1.0\Sprint%201\SE-II-2020-Daily%20scrum%20meeting%20template.docx#_Toc507250134)

[Class Graph 10](file:///F:\mona\My%20Fcai\Level_3_Sem_2\SE_2\Assignments\Project_Phase1\SE-II-2020-ProjectDescription-v1.0\Sprint%201\SE-II-2020-Daily%20scrum%20meeting%20template.docx#_Toc507250135)

[Class Hamilton 11](file:///F:\mona\My%20Fcai\Level_3_Sem_2\SE_2\Assignments\Project_Phase1\SE-II-2020-ProjectDescription-v1.0\Sprint%201\SE-II-2020-Daily%20scrum%20meeting%20template.docx#_Toc507250135)

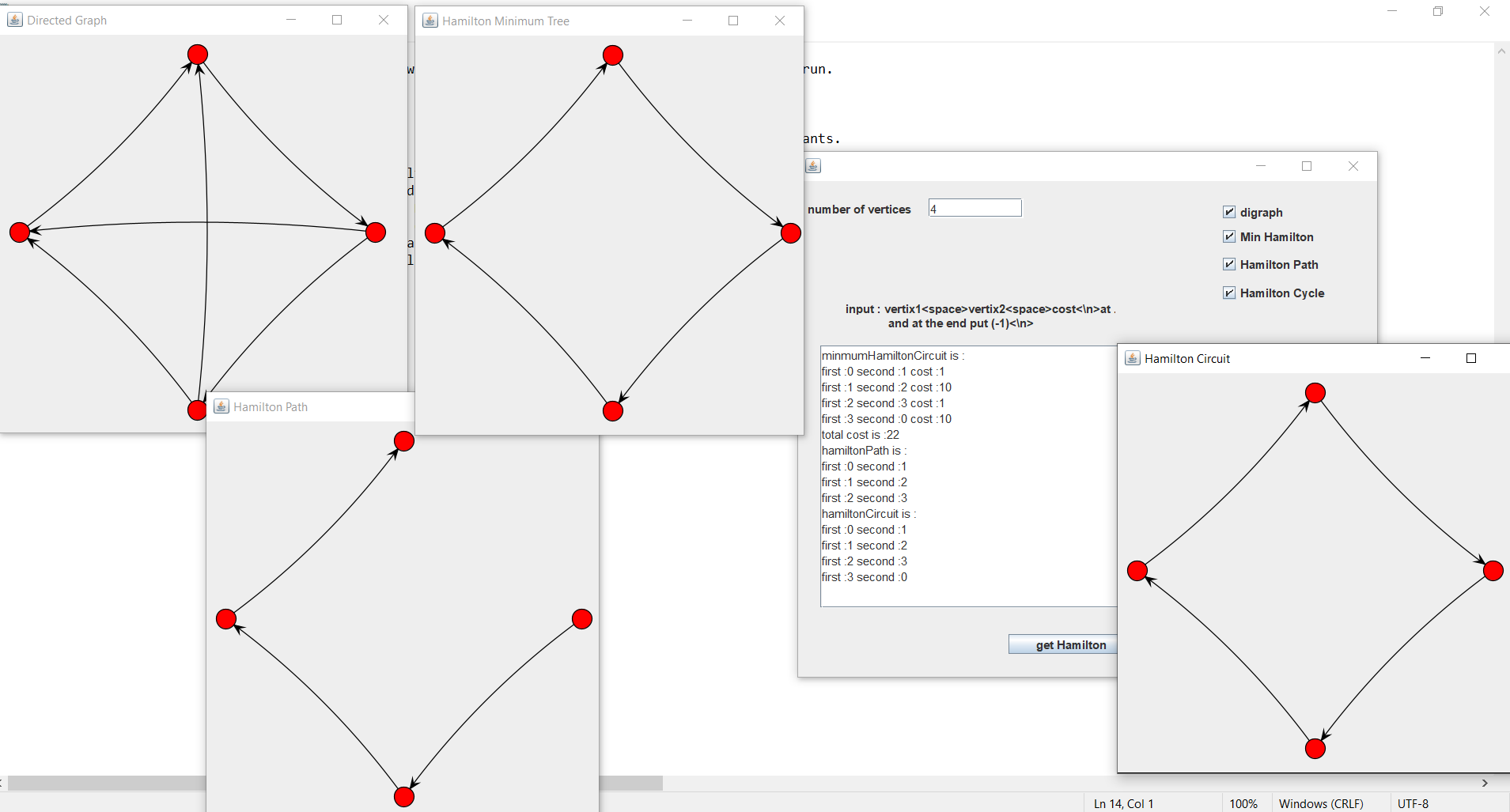
**Main purpose of the program and the presented solution to**

**The graph problem:**

The program generates a small interface for the user to interact with,

the user sets the number of vertices he wants and enters which edges

he’d like to have then if there is Hamilton path, Hamilton circuit or neither, and Minimum Hamilton Circuit and presented to the user after he clicks the get Hamilton button, the program create the graph which can be directed or undirected according to the user’s choice based on a checkbox named “digraph “in the user interface.



**Hamilton Circuit:**

A Hamiltonian circuit in a graph G is a simple circuit that passes through every vertex in G exactly once.

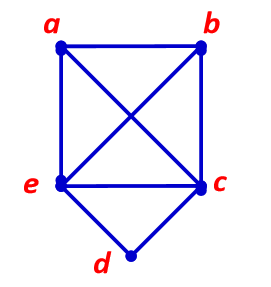
**Certain Properties In Hamiltonian Circuit :-**

A graph with a vertex of degree one cannot have HC, because each vertex in HC is incident with two edges.

-If a vertex has degree two, then both edges that are incident with this vertex must be part of any HC.

-When a HC is being constructed and this circuit passes through a vertex, then all remaining edges incident with this vertex, other than two used in the circuit, can be removed from consideration.

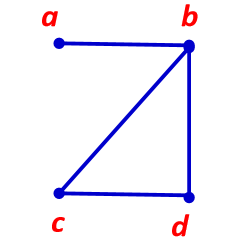
-HC cannot contain a smaller circuit within it.



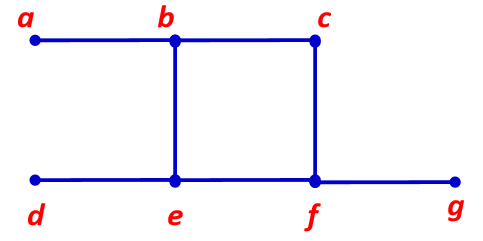


**Hamilton Path:**

A Hamiltonian path in a graph G is a simple path that passes through every vertex in G exactly once.



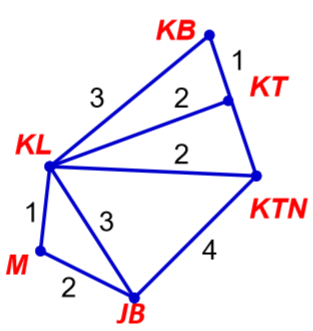


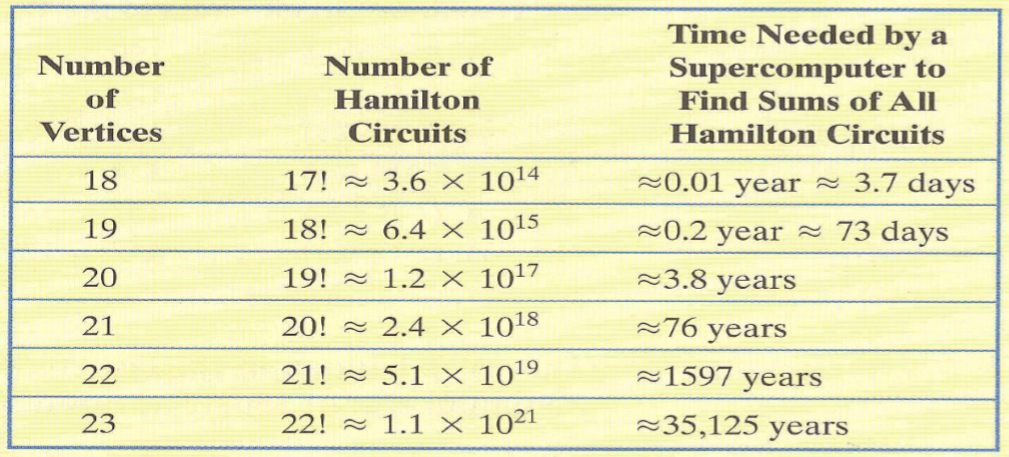


****

**Minimum Hamilton Circuit:**

– Get circuit with shortest total length that visits every vertex of a complete graph exactly once.



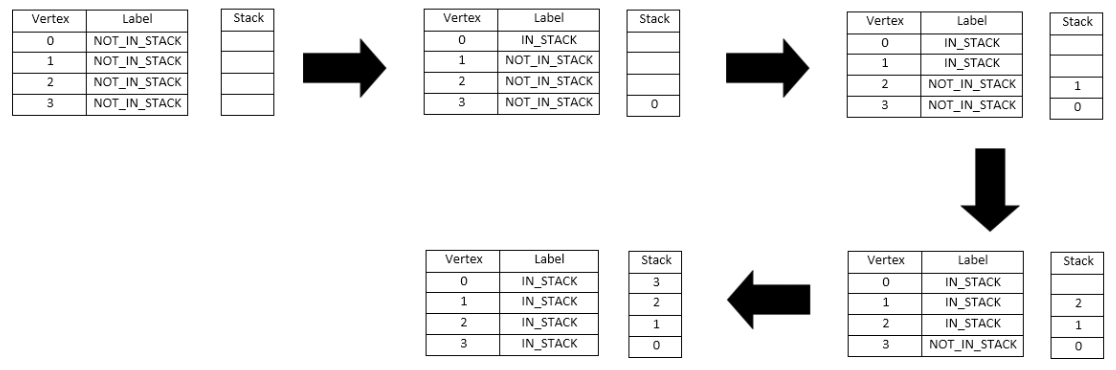
Suppose that a supercomputer can find the sum of the weights of one billion, or 109 Hamilton circuits per second. Because there are 31,536,000 seconds in a year, the computer can calculate the sums for approximately 3.2 x 1016Hamilton circuits in one year. The table below shows that as the number of vertices increases the Brute Force Method is useless even with a powerful computer.

**Algorithm for implementing the Hamilton Circuit:**

Create an empty path array and add vertex 0 to it. Starting from the vertex 1, we will add other vertices. Before adding a vertex, check for whether it is adjacent to the previously added vertex and not already added. If we find such a vertex, we add the vertex to the path array. If we do not find a vertex then we return false, that means that there is no Hamilton circuit, and finally if there is Hamilton circuit, fill the output graph with the edges that is in the path array, if not, return null.

**Algorithm for implementing the Hamilton Path:**

We apply depth first search starting from every vertex v and do labeling of all the vertices. All the vertices are labeled with either 1 or 0. A vertex is labeled by 1 if it is visited and added to the output graph, but some of its adjacent vertices are not yet visited and is labeled by 0 if it is not visited.  
If at any instant the number of vertices with label 1 is equal to the total number of vertices in the graph then a Hamilton Path exists in the graph, but if the number of vertices with label 1 become greater than total number of vertices in the graph then return false, that means that there is no Hamilton path, and return null graph.

****

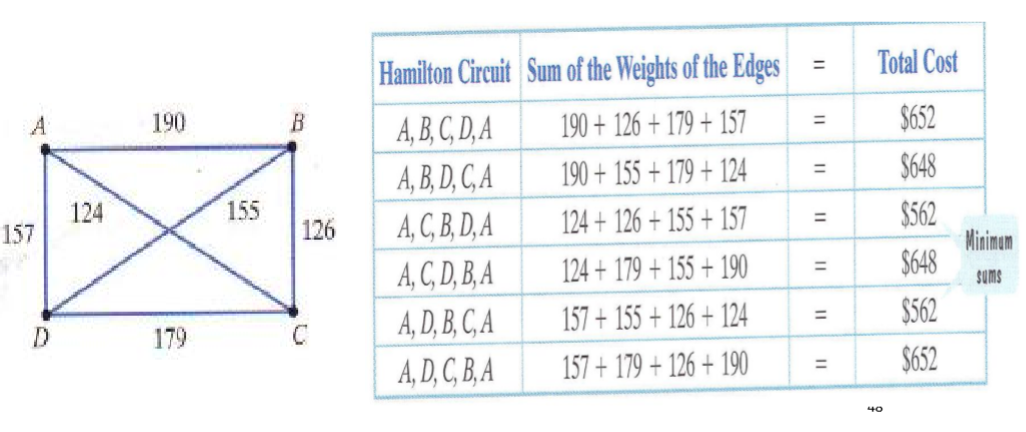
**Algorithm for implementing the Minimum Hamilton Circuit:**

1. Model the problem with a complete, weighted graph.

2. Make a list of all possible Hamilton circuits.

3. Determine the sum of the weights of the edges for each of these Hamilton circuits.

4.The Hamilton circuit with the minimum sum of weights is the optimal solution.



**Classes we implemented in order to make the program:**

**The GraphGui class:**

A class that generates the graphical user interface which will be called on by the Main class when we run. We set the size of the window and the text areas and create one button.it is where the window where the user will enter the number of vertices , the edges and costs he/she wants. There is Four Check Boxes one for digraph , Min Hamilton Tree , Hamilton Path and Hamilton Circuit.

There is a button that we have created that is called get Hamilton, when clicked on , and check any one of the check Boxes if we check Min Hamilton the Min Hamilton graph will be displayed,

if we check Hamilton Path the Hamilton Path will be displayed ,

if we check Hamilton Circuit the Hamilton Circuit graph will be displayed , if we check Directed the Directed graph will be displayed ,

else it will show undirected Graph.

**The Edge class:**

This class contains main data for each edge as source vertex, destination vertex and its cost.

Edge(source, destination, cost);

//display edge:

Print source , destination , cost;

**The Graph class:**

This class creates, initializes graph.

create edges list

create representation matrix

function add edge(x , y , cost):

representation matrix[x][y]++

edges.add(x, y, cost);

**function reset:**

edges.clear

for i = 0 : i < number of vertices

for j = 0 : j < number of vertices

representation[i][j] = 0;

end for

end for

**function copy(Graph g):**

edges = g.edges;

vertices = g.vertices;

representation = g.representation

**The Hamilton class:**

This class contains the main functions of Hamilton problems.

**function HamiltonPath:**

create finalGraph

flag = isValid(representation matrix, number of vertices and finalGraph)

if (flag = false)

return null

end if

else

return finalGraph

**function isValid:**

create array path = 0;

for i=0 : i<vertices

path[i] = 1

if (check(i , representation , path , 1 , vertices , finalGraph) = true)

return true

end if

return false

**function check:**

if (count = vertices)

return true

end if

for i=0 : i<vertices

if (representation[v][i] > 0 & path[i] = 0)

path[i] = 1

finalGraph.addEdge(v , I , representation[v][i])

if(check(i , representation , path , count+1 , vertices , finalGraph) = true)

return true

end if

end if

end for

return false

**function hamilton circuit:**

create finalGraph

create array path = -1

path[0] = 0

if(hamCycleUtil(representation , path , 1 , vertices) = false)

return null

end if

fillGraph(path , finalgraph)

return finalGraph

**function hamCycleUtil:**

if (pos = vertices)

if (graph[path[pos-1]][path[0]] = 1)

return true

end if

else

return false

end if

for v=1 : v<vertices

if (isSafe(v,graph,path,pos) = true)

path[pos] = v

if (hamCycleUtil(graph,path,pos+1,vertices) = true)

return true

end if

path[pos] = -1

end if

end for

return false

**function isSafe:**

if (graph[path[pos-1]][path[v]] = 0)

return false

for i=0 : i<pos

if (path[i] = v)

return false

end if

end for

return true

**functions of MinimumHamiltonCircuit:**

if number of vertices >7

sort the edges by first

paths [];

for i=0 to numper vertices

paths.add(getpaths(i)).//getpaths(i) is a function to get possible paths for this vertix

res[];

for 0 -> paths.size

remove the Repetition paths

and store the not repeat paths in res[];

min=100000

finalgrath

for 0-> res

if(res[i]<min)

min= res

finalgrath=res

return finalpath;

else

//the algorithm will take too much time so it return empty graph