

# Program #4

## Save the Planet

Nathan Balon  
Dinesh Thandapani  
CIS 350  
Data Structures & Algorithms

<b>What</b> ■ indicates program/other is memo	<b>Points</b>	<b>Due Tuesday April 19th</b>
<b>Fun Team Name</b>	<b>1</b>	
<b>External Documentation</b>	<b>14</b>	
Your Name	1	
Description of the problem	1	
Input Specification	1	
Output Specification	1	
Algorithm Description & Design UML (must be UML)	10	
<b>Data Structure</b>	<b>12</b>	
main data structures	5	
member functions / functions pre/post conditions for each	2	
Implementation and discussion.	5	
<b>Analysis</b>	<b>11</b>	
Worst case time analysis for each function	3	
Worst case space analysis for each function	3	
Test Plan	3	
Sample Runs	2	
<b>Program Listing Style</b>	<b>12</b>	
Your Name	1	
Description of the problem	1	
Variable Names	1	
Data Dictionary	2	
Pre/post conditions	3	
Length of functions	3	
Use of white space	1	
<b>Functionality</b>	<b>50</b>	
<b>Main</b>	<b>5</b>	
<b>Works</b>	<b>45</b>	

# External Documentation

## Description of Problem

The program is to accept a number of points represented by their x and y coordinates of the point. The minimum distance needed to connect all the points entered by the user is then calculated. When calculating the distance, no points should be disconnected from the rest. A possible application for the program would be to determine the minimum distance of road needed to connect cities on map, so that there is a path to all cities. Figure 1 shows a possible combination of points which could be entered and the connections of the points which contain the minimum distance. In this case, the minimum distance needed to connect all the points would be 7.0.

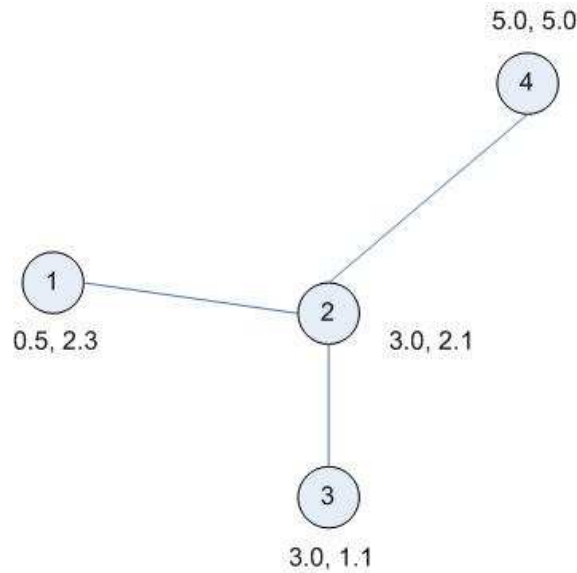


Figure 1. Sample Application

## Input

All input to the program is done through the keyboard. The user of the program will enter the number of points to be used on one line. Next, on each line the user will enter the x and y coordinates of each point. Each of the coordinates entered should be within the range 0.0 to 100.0. The program will stop accepting data when all the points have been entered. At this time the program will calculate the minimum cost to connect all of the entered. After the minimum cost is determined to connect the points, the user then can enter another set of points. If the user enters zero for the number of points in the set the program terminates. A sample of valid input to the program would be:

```
3
1.0 1.1
1.0 2.2
1.0 3.3
0
```

Figure 2. valid input

## Output

All output from the program will go to standard output and be displayed on the user's terminal. After each set of data is entered the program displays the minimum distance needed to calculate all the points in the set. For the sample input given in figure 2 the program would display "2.2".

## Algorithm Description and Design

The algorithm used to determine the minimum distance needed to connect points in the graph is Prim's Algorithm. A complete graph is created from all the points entered by the user. A complete graph was used because it is possible that any vertex in the graph can be connected to any other vertex. Prim's Algorithm then determines the minimum distance to needed to connect all the points in the graph, so there is a path to all nodes in the graph.

The function main is used to reading in all the points that will be contained in the graph. The function main then creates a vertex in the graph for each point entered. The member function PrimAlgorithm then determines the minimum spanning tree for the graph. Figure 3 shows the activity diagram for the function main. Figure 4 shows the activity diagram for the member function PrimAlgorithm.

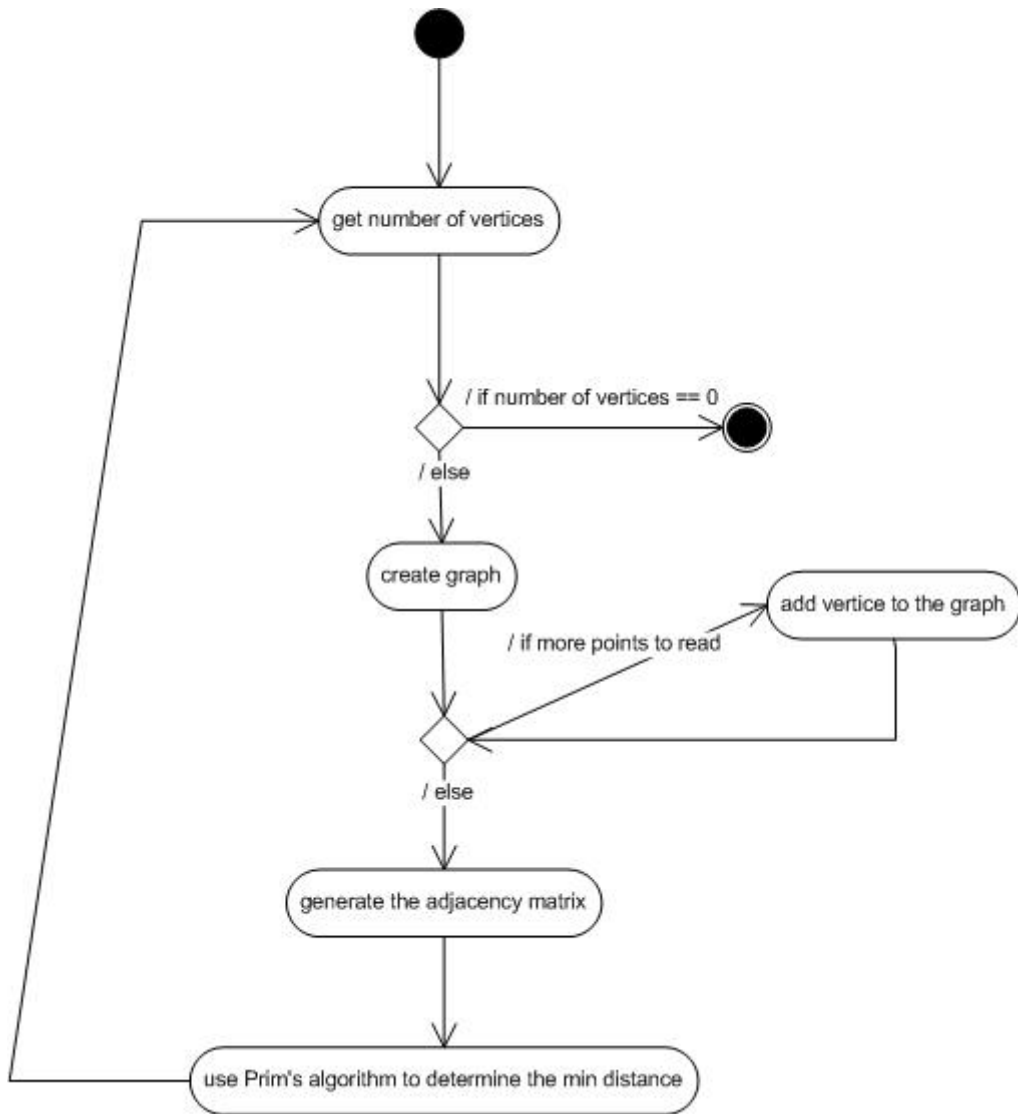


Figure 3. main

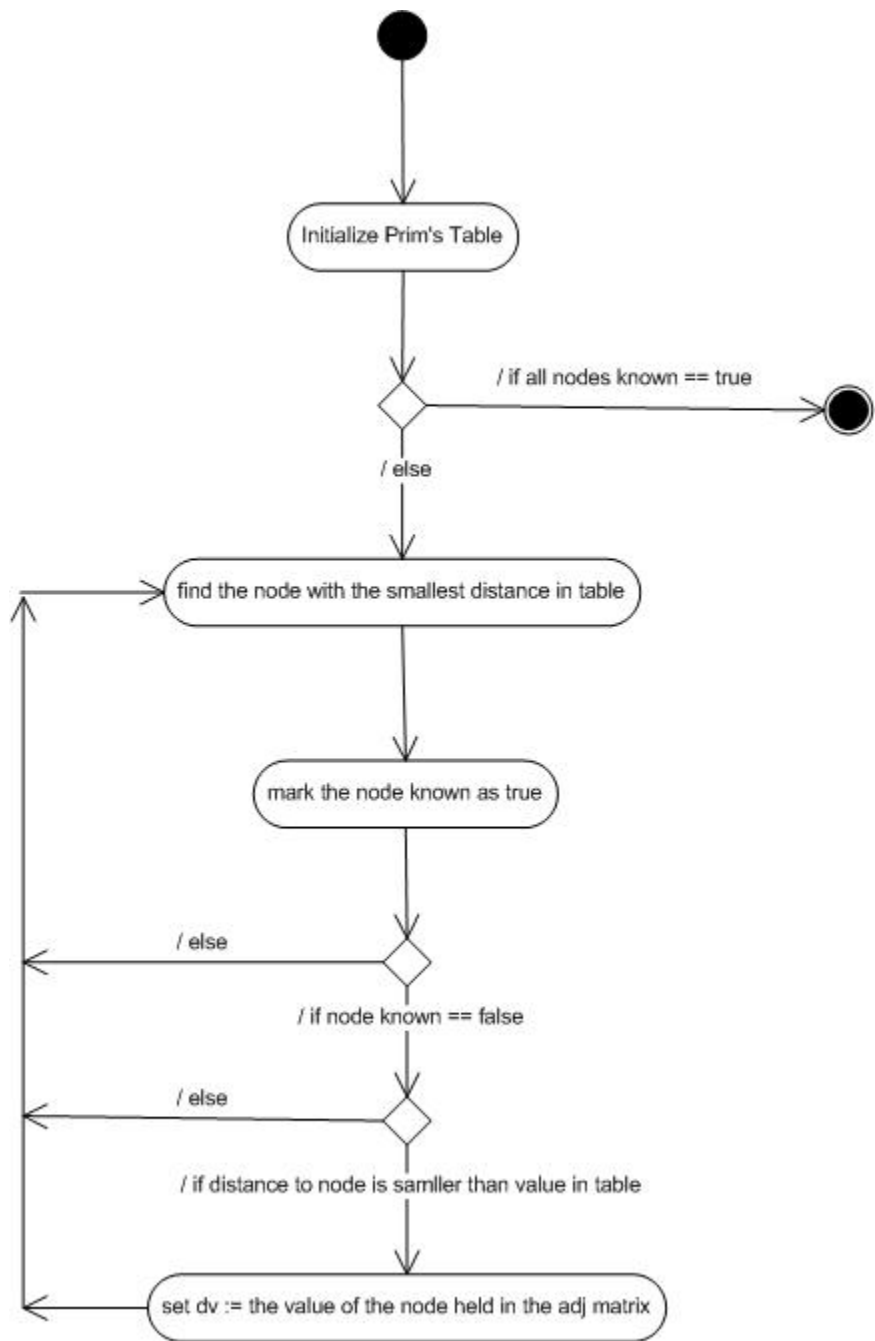


Figure 4 Prim's Algorithm

## Data Structure

### Main Data Structures:

1. **Point** : The Structure Point is used to store the x and y co-ordinates of the vertex (Exploration Site). The co-ordinates entered are of double data type.

```
struct Point{
    double x_;    // the x coordinate
    double y_;    // the y coordinate

    Point(int x, int y);
    Point();
};
```

2. **Prim**: The Structure Prim is used to create the PrimTable with Vertex Number , distance , source vertex number and a Flag to show vertex is visited or not. The vertices are of integer datatype and the distance is double.

```
struct Prim{
    int vNo;      // Vertex number
    bool known;  // If vertex is visited true else false
    double dv;   //minimum distance from source to destination
    int pv;      // source vertex number
};
```

3. **Graph**: Class Graph is used to store all the vertices and edges (distance between the vertices) in a graph using a adjacency Matrix. The main purpose of the class is to store a graph of points containing x and y coordinates of the points and to compute the minimum spanning tree of the graph by implementing Prim's Algorithm. It contains the following public and private members and member functions.

```
public:
    typedef vector <double> row;
    Graph(int noVertices);
    void addVertex(Point p);
    void genAdjacencyMatrix();
    void displayGraph();
    void printAdjacencyMatrix();
    void primAlgorithm();
    void displayPrimTable();
```

```
private:
    void displayDistance();
    void initializePrimTable();
```

```
vector<Point> vertices_; // the vertices contained in the graph
vector<row> adjMatrix; // a complete graph of all points
vector<Prim> primTable; // table used to calculate the min spanning tree
```

### **Global Functions and Pre/Post Conditions:**

**getNumberOfVertices:** Gets the number of vertices in the graph from standard input and checks that the number of vertices is even.

Precondition: None.

Postcondition: Return the number of vertices the user has entered.

If the user enters an odd number of vertices an error is displayed and the program is exited.

**validCoordinate:** checks that the coordinate is within a valid range.

Precondition: none

Postcondition: if an invalid coordinate was entered an error will be displayed and the program will terminate.

**displayHelp:** display help to the user to run the program.

Precondition: None

Postcondition: the user will be display information concerning how to run the program.

### **Member Functions of Class Graph and Pre/Post Conditions.**

**Graph:** creates an instance of a graph.

Precondition: Resources are available to creates the object and the vertex objects contain in the vector have been properly initialized.

Postcondition: A graph object is created. The graph will be created with the number of vertices specified by the numberOfVertices argument.

**displayGraph :** Displays the vector of vertices entered by user

Precondition : All co-ordinates should be given in the standard input.

Postcondition : Displays the graph (Coordinates)

**genAdjacencyMatrix :** Creates the Adjacency Matrix of distance between the vertices

Precondition : All the vertice co-ordinates should be entered in the standard input.

Postcondition : Generates the Adjacency Matrix of Distance between the vertices.

**printAdjacencyMatrix :** Displays the Adjacency Matrix of distance between the vertices



Precondition : Adjacency Matrix of distances should be generated using genAdjacency matrix

Postcondition : Prints the distance adjacency Matrix for checking.

**primAlgorithm:** Determines the minimum spanning tree of the graph.

Precondition: The method genAdjacencyMatrix must be called on the graph object before calling primAlgorithm.

Postcondition: The minimum spanning tree of the graph will be determined and the total cost of the graph will be displayed.

**intializePrimTable:** Intializes the data in the table used to determine the minimum spanning tree.

Precondition: None

Postcondition: The table used to determine the minimum spanning tree will be initialized.

Called by: primAlgorithm

**displayPrimTable:** Displays the contents of the table used to calculate the minimum spanning tree. Used to debug the program.

Precondition: The function initializePrimTable must be called first so that the table contains valid values.

Postcondition: The values used to calculate the minimum spanning tree will be displayed.

**displayDistance:** Display the total distance of the graph

Precondition: The minimum spanning tree of the graph must be determined before calling.

Postcondition: The cost of the minimum spanning tree is displayed

Called By: primAlgorithm

**addVertex:** Add vertices to the graph

PreCondition: Graph Object should be created and the Point Input should be got

PostCondition: Vertice is added in the Graph.

## Implementation and Discussion

Structures are used in the program for the points of the graph and for the primTable (used by Prim's Algorithm). Also, vectors are used to store the points and the primTable data because they allocate memory during runtime.

The class Graph is used to add the vertices, to calculate the distance between the vertices and to implement the Prim's Algorithm to calculate the minimum spanning tree which gives the minimum total distance of all vertices.

Prim's Algorithm was chosen because it is a minimum spanning tree algorithm which is used to calculate the minimal cost spanning tree of the connected graph (i.e. A spanning tree in a graph is a tree that visits every node in the graph). In our case, the cost of the edge is the distance between the two vertices (exploration sites) in the graph. There are several algorithms for finding the minimal spanning tree, the reason we chose the Prim's algorithm is its accuracy and its simplicity.

Prim's Algorithm begins by adding the lowest cost (distance) edge and its two endpoints to the solution set. It then loops adding the lowest cost edge that connects a vertex in the solution set to one outside it. It also adds the endpoint of this edge that is not already in the solution set. The algorithm terminates when all vertices are in the solution set. The edges and vertices in the solution set at this point constitute a minimal cost spanning tree of the input graph.

The Adjacency Matrix was implemented using two dimensional vectors to store the distances of the edges. We chose Adjacency Matrix for the following two reasons.

1. It is efficient for finding the shortest path of a graph.
2. The graph will be a dense graph since we are calculating the edge distance for all pairs of vertices.

## Analysis

For the analysis of space and time the following variables are used:

- $v$  = the number of vertices in the graph
- $n$  = the number of graphs input to the program

### Worst Case Time Analysis

Function	Worst Case Time
Point::Point(int x, int y)	$O(1)$
Point::Point()	$O(1)$
Graph::Graph(int noVertices)	$O(v^2)$
void Graph::displayGraph()	$O(v)$
void Graph::genAdjacencyMatrix()	$O(v^2)$
void Graph:: printAdjacencyMatrix()	$O(v^2)$
void Graph::primAlgorithm()	$O(v^2)$
void Graph::initializePrimTable()	$O(v)$
void Graph::displayPrimTable()	$O(v)$
void Graph::displayDistance()	$O(v)$
void Graph::addVertex(Point p)	$O(1)$
int getNumberOfVertices()	$O(1)$
void validCoordinate(const double & coord)	$O(1)$
void displayHelp()	$O(1)$
int main(int argc, char *argv[])	$O(n*v^2)$

### Worst Case Space Analysis

Function	Worst Case Space
Point::Point(int x, int y)	$O(1)$
Point::Point()	$O(1)$
Graph::Graph(int noVertices)	$O(v^2)$
void Graph::displayGraph()	$O(1)$
void Graph::genAdjacencyMatrix()	$O(1)$
void Graph:: printAdjacencyMatrix()	$O(1)$
void Graph::primAlgorithm()	$O(1)$
void Graph::initializePrimTable()	$O(v)$
void Graph::displayPrimTable()	$O(1)$
void Graph::displayDistance()	$O(1)$
void Graph::addVertex(Point p)	$O(1)$
int getNumberOfVertices()	$O(1)$
void validCoordinate(const double & coord)	$O(1)$
void displayHelp()	$O(1)$
int main(int argc, char *argv[])	$O(v^2)$

## Test Plan

<i>Test #</i>	<i>Data</i>	<i>Expected Output</i>
1	1.0 1.1 1.0 2.2 1.0 3.3	2.2
2	4.0 4.0 10.0 4.0 4.0 6.0 2.0 4.0	10.0
3	1.0 1.0 2.0 2.0 2.0 4.0	3.4
4	0.5 2.3 3.0 2.1 3.0 1.1 5.0 5.0	7.0
5	50 60 45 67 89 23 76.99 87.56 23.67 2.22 34.90 67.5 23.55 90.34 10 20 89.67 45.90 12 34 45 67 78 97 12 56 90 35 67 53	239.0

<i>Test #</i>	<i>Data</i>	<i>Expected Output</i>
6	0 0 2 0 5 5 10 40 1 50 75 75 100 100	165.8
7	1234 1 2 3.6 99.9 43.0 87.5	Error Message because of out of range data

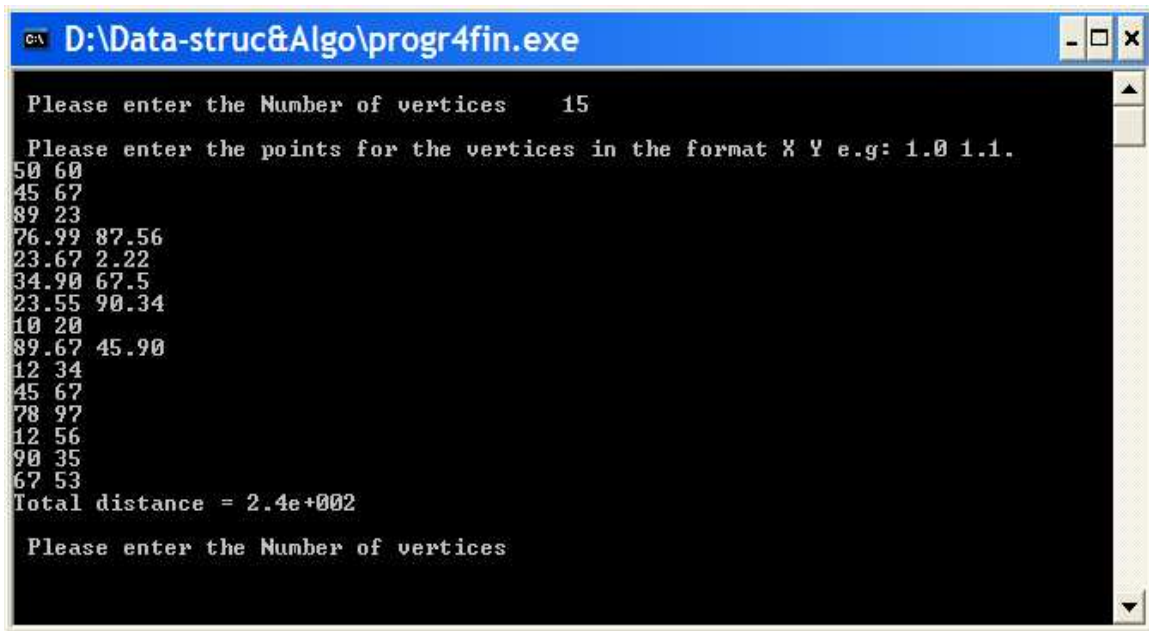
- Initially we tested the member functions that stores the vertices of the graph (by displaying the co-ordinates)
- We then tested the accuracy with which it calculates the distance between the vertices (by displaying the adjacency matrix).
- We checked the valid co-ordinates function by entering the values out range (greater than 100 and negative values)
- We then checked the shortest distance calculation of the Prim's Algorithm by entering the sample data from the question and then by entering other values.

## Sample Run.

The sample data from test 3 which displays the adjacency matrix and Prim's Table and the shortest distance.

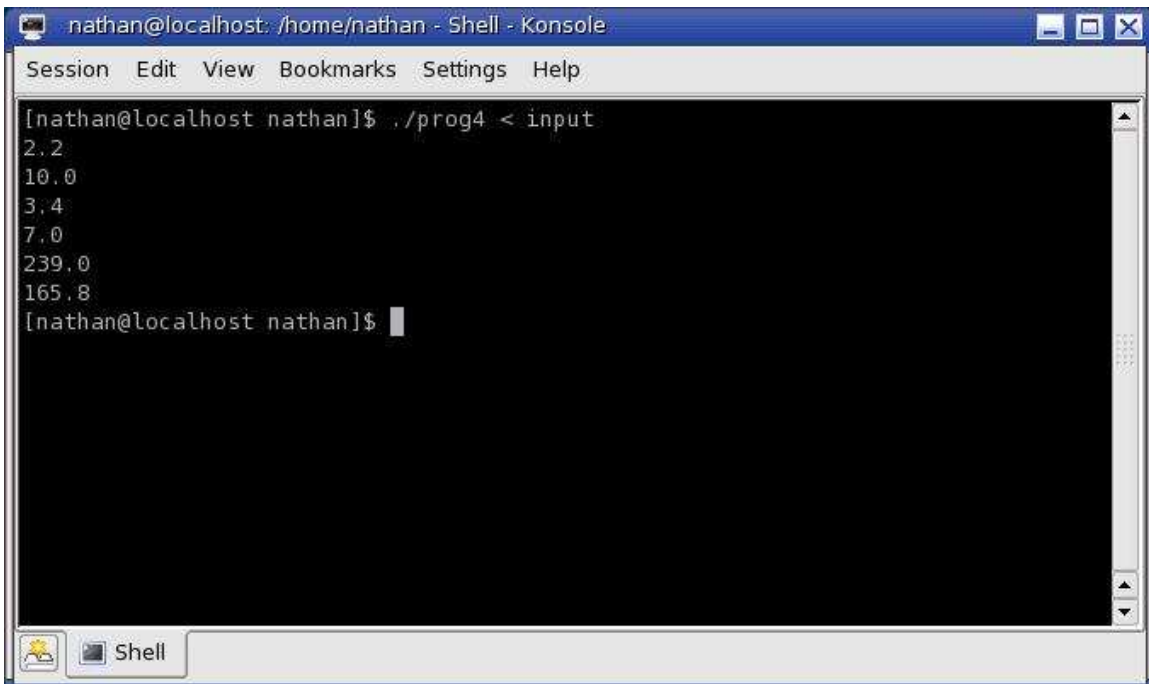
```
D:\Data-struct&Algo\progr4fin.exe
Please enter the Number of vertices      3
Please enter the points for the vertices in the format X Y e.g: 1.0 1.1.
1.0 1.0
2.0 2.0
2.0 4.0
The Generated graph is:
-----
1      1
2      2
2      4
Adjacency Matrix:
-----
          0      1.4      3.2
          1.4      0      2
          3.2      2      0
RPrim's Table
v      known      dv      pv
-----
0      false      0      0
1      false      INF      0
2      false      INF      0
RPrim's Table
v      known      dv      pv
-----
0      true      0      0
1      false      1.4      0
2      false      3.2      0
RPrim's Table
v      known      dv      pv
-----
0      true      0      0
1      true      1.4      0
2      false      2      1
RPrim's Table
v      known      dv      pv
-----
0      true      0      0
1      true      1.4      0
2      true      2      1
Total distance = 3.4
Please enter the Number of vertices      -
```

The output from test case 5 for a graph with 15 vertices.



```
D:\Data-struct&Algo\progr4fin.exe
Please enter the Number of vertices 15
Please enter the points for the vertices in the format X Y e.g: 1.0 1.1.
50 60
45 67
89 23
76.99 87.56
23.67 2.22
34.90 67.5
23.55 90.34
10 20
89.67 45.90
12 34
45 67
78 97
12 56
90 35
67 53
Total distance = 2.4e+002
Please enter the Number of vertices
```

Output from the test cases 1 through 6 with the tables of the program disabled for output.



```
nathan@localhost: /home/nathan - Shell - Konsole
Session Edit View Bookmarks Settings Help
[nathan@localhost nathan]$ ./prog4 < input
2.2
10.0
3.4
7.0
239.0
165.8
[nathan@localhost nathan]$
```