Project 6

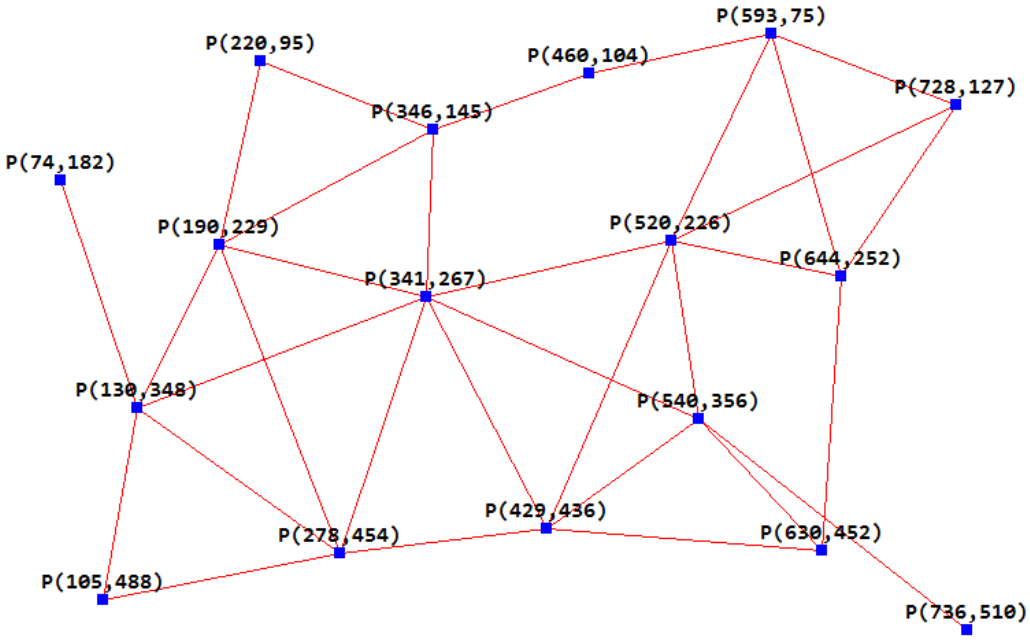
GRAPHS

Graph Code (from book)

- You will need to download following code to do the exercises in this project:
  - graphs.zip
- After extracting this archive you will find several graph files from our textbook materials, and the old eda module (as a subdirectory) for drawing
- This files may be updated from time to time, so be ready to refresh your copy when notified (a Moodle message will be sent to all students in case of update)

Problem 1: Graph Create and Draw

- Download the file graph1.dat
- It consists of two lines:
  - List of vertices (points on canvas plain)
  - List of edges (lines between two points)
- Parse this file, create the graph, and draw it:



Hint to Problem 1

- To be able to parse the file like graphs1.dat you will need the following function:

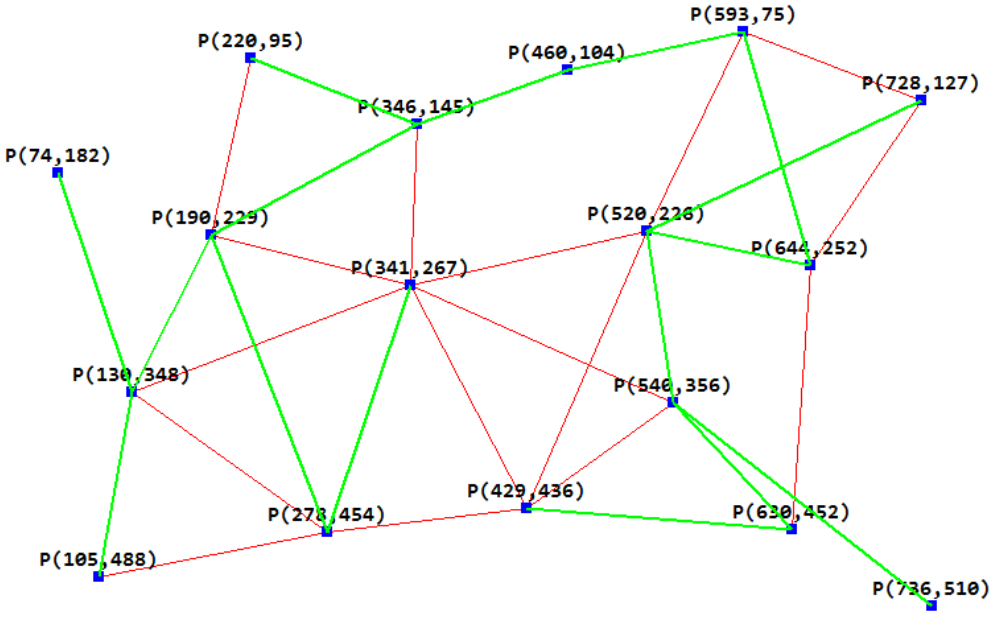
```
# Parse the list of integers inside
# a string like '(10,20)' or '((10,20),(30,40))'

def parse_ints(s):
    t = re.sub('[(),]', ' ', s)
    ints = [int(x) for x in t.split()]
    return ints
```

Download all your data graphs!

- The file data.zip consists with 12 graph files which you will use in this project. Click it to download!
- All the graphs are planar graphs:
- Vertices are points on a two dimensional plane
- Edges are line segments between two points
- We will use the names graph1, graph2, graph3, …, to refer to the graphs that are generated by these files
- You can generate two types of graphs from this files
  - Undirected graphs (if you ignore the direction)
  - Directed graphs

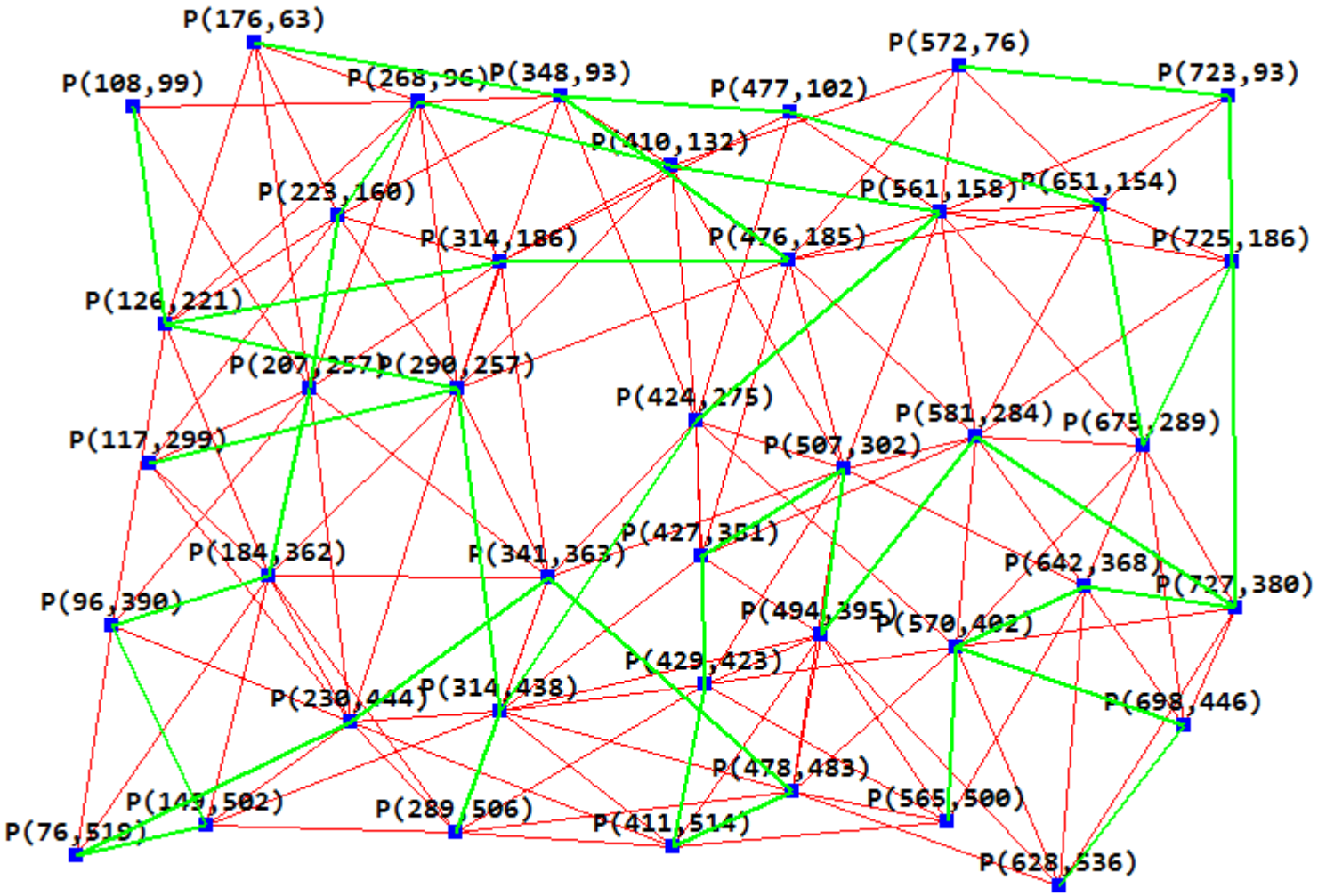
Problem 2: DFS and Spanning Tree

- Find a spanning tree for graph1 that start with the point (278,454)
- Hint: look at the function **DFS** at the file: **dfs.py**
- Try to draw your spanning tree by coloring its edges green
- Can you see more than one spanning tree?
- Count total length of edges in tree (write code). Hint: Use line length method.
- Use lecture notes Part_07_dfs.pdf to read about spanning trees
- Check your solution also on graph2, graph3, …



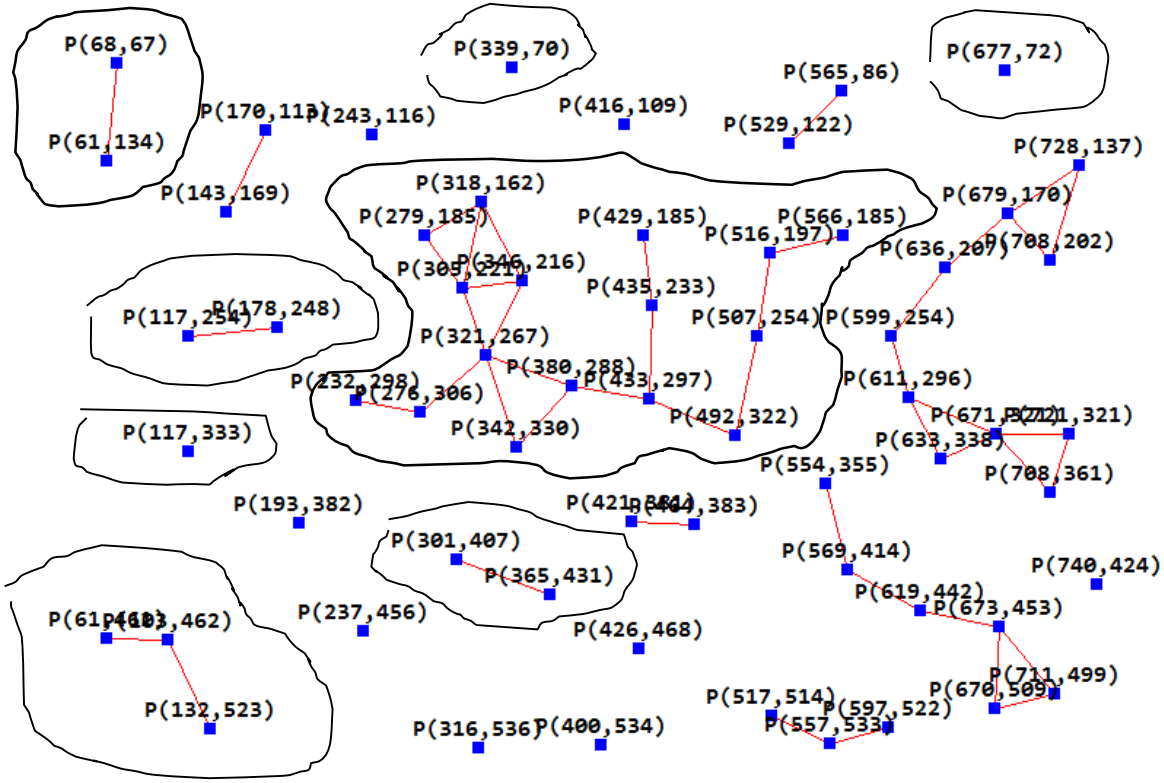
DFS and Spanning Tree

- Here is a Spanning Tree solution for graph4 that starts with the point (727,380)



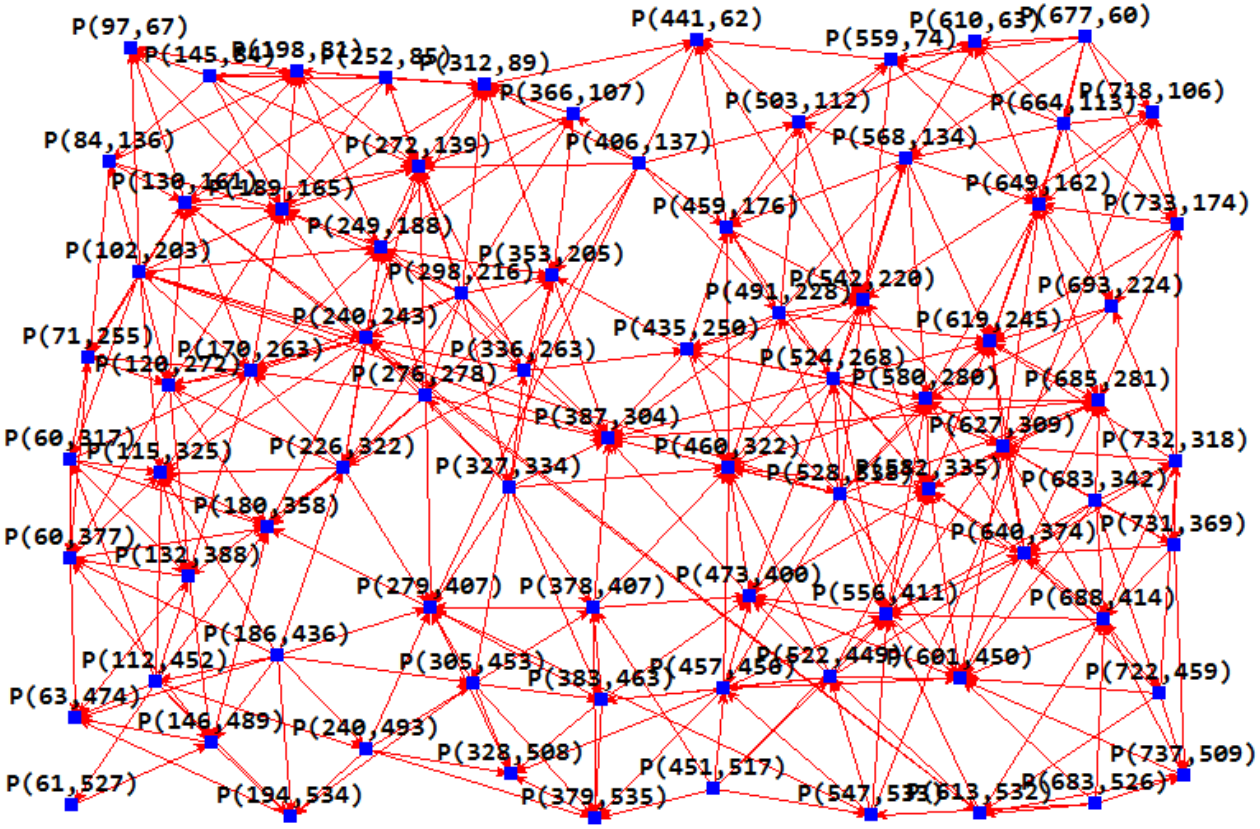
Problem 3: Connectivity Components

- Create and Draw graph11.dat
- This is a disconnected graph, so it consists of several connectivity components
- Find the connectivity component that contains the point P(215,85)
- You need to write an algorithm for solving this problem
- Here is an example of how a disconnected graph looks like:



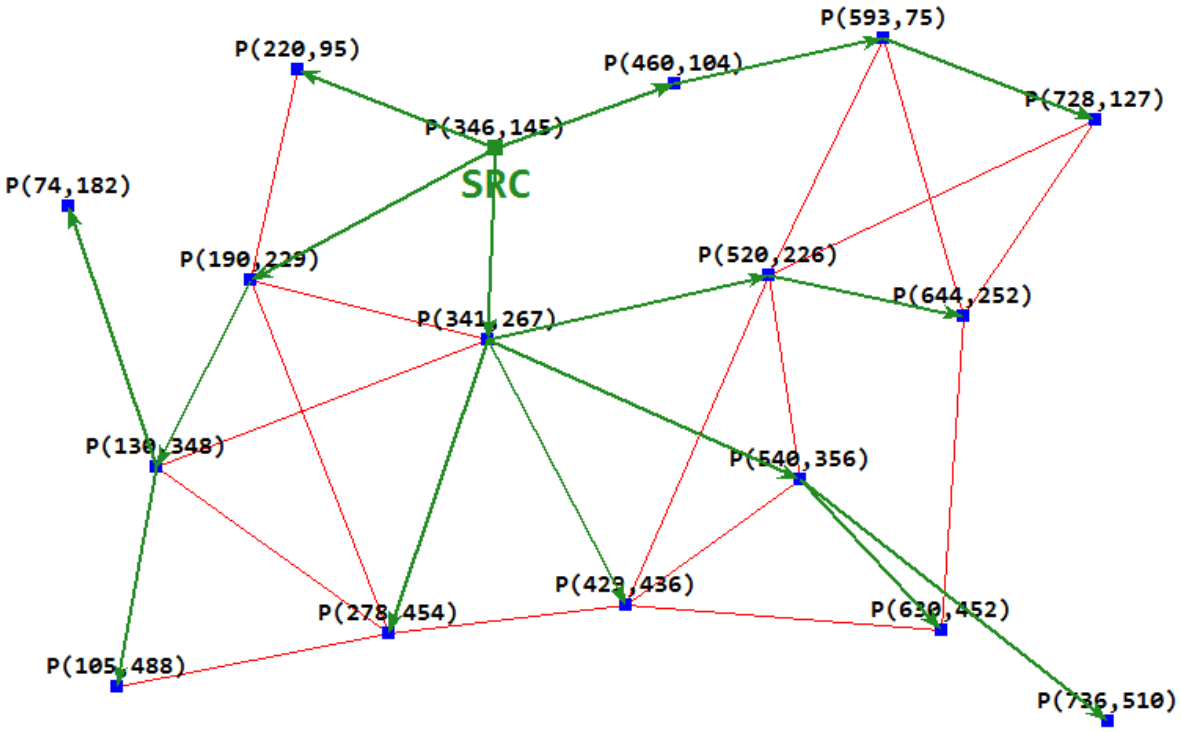
Problem 4: Cycles

- Create and Draw graph12.dat as a **directed graph**!
- A cycle is a list of 2 or more vertices v[0], v[1], …, v[n], such that v[i] is connected by an edge to v[i+1], and v[n] is connected to v[0].
- Write a program for finding all cycles in graph12:



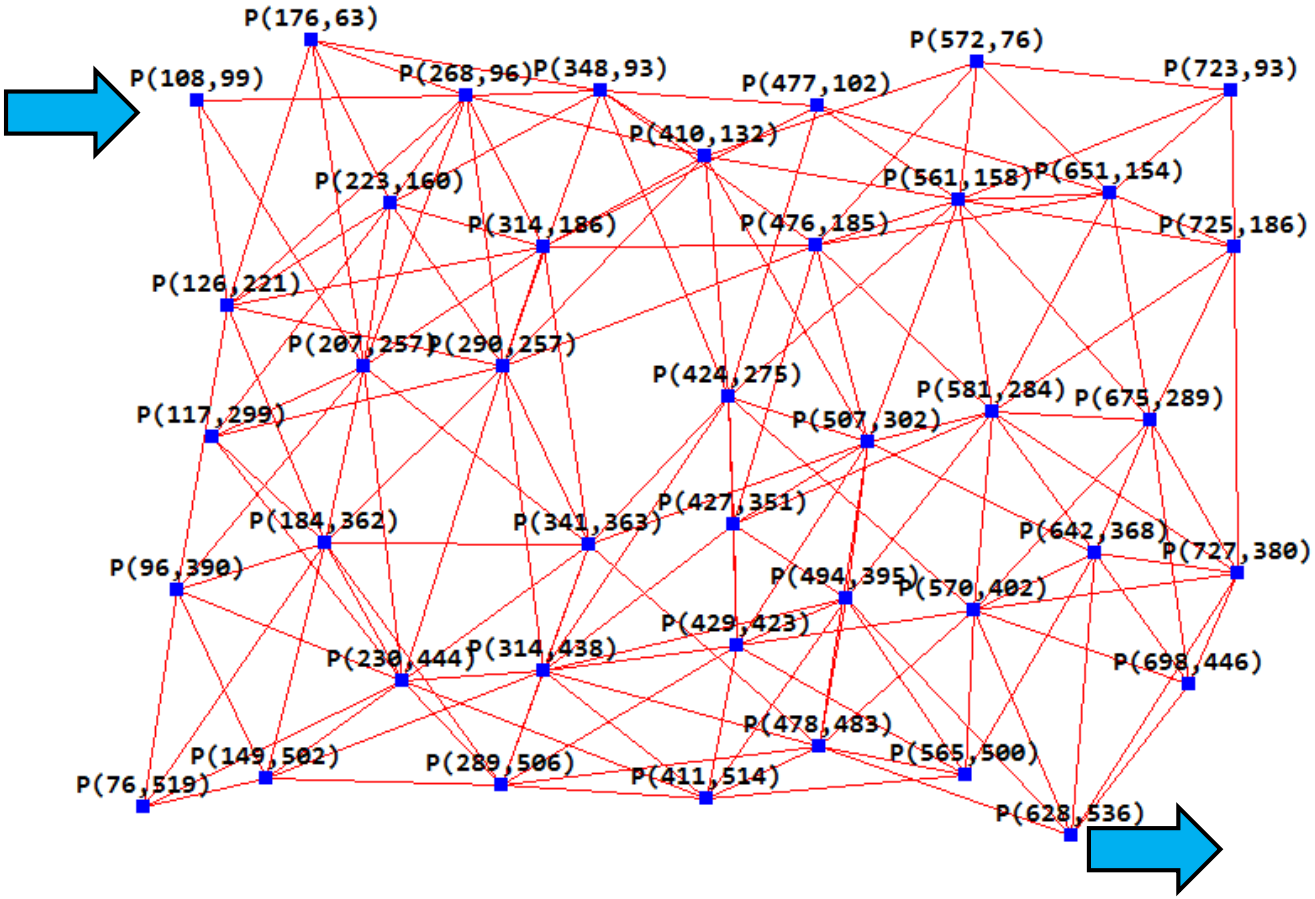
Dijkstra Algorithm (Shortest Path)

- Read the file dijksta.py and try to understand Dijkstra's algorithm for finding all shortest paths from a source vertex src to all other vertices in the graph



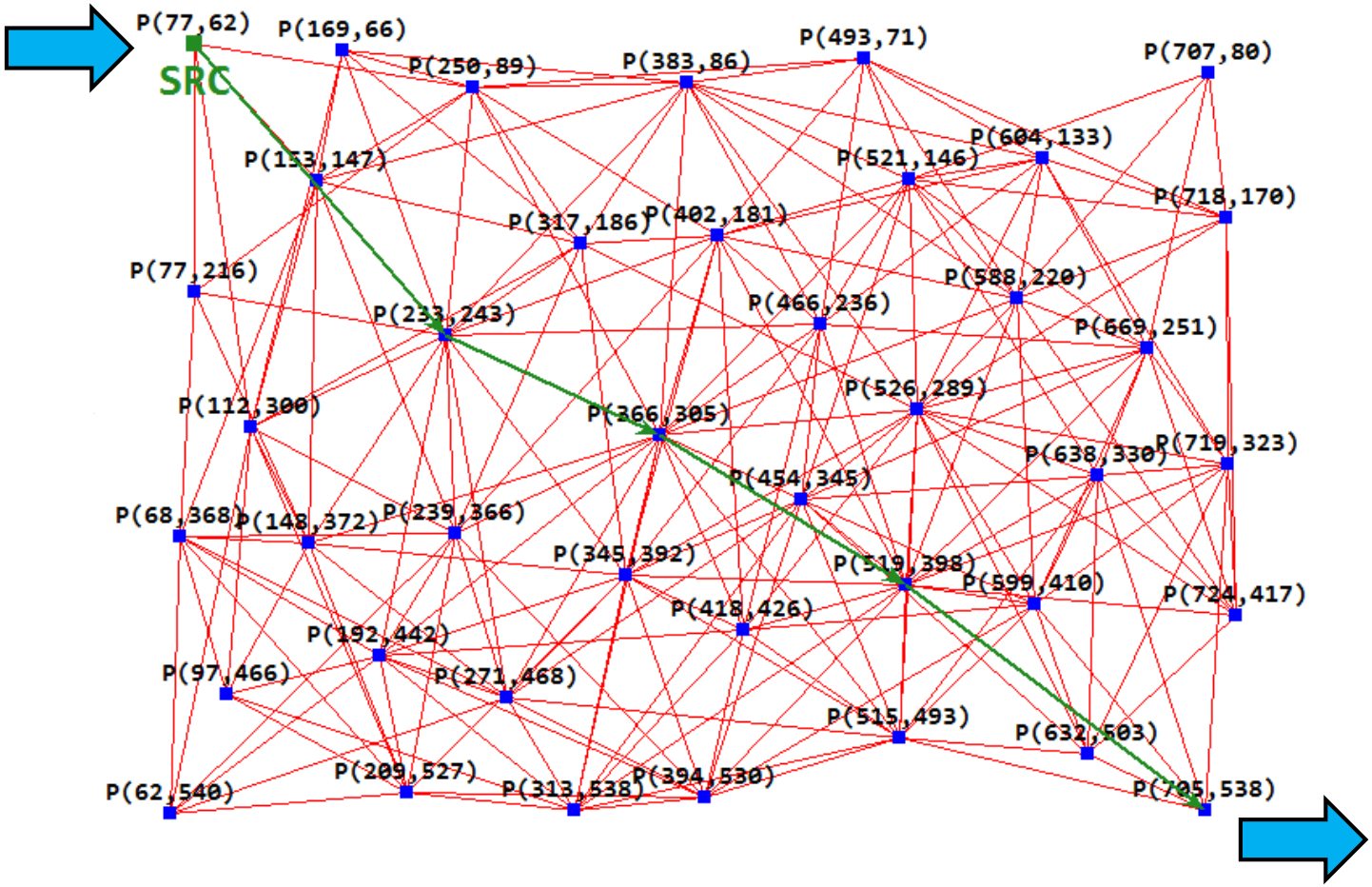
Problem 5: Shortest Path

- Compute the shortest path from the point P(108,99) to point P(628,536) in graph4 (add distance element to graph!)



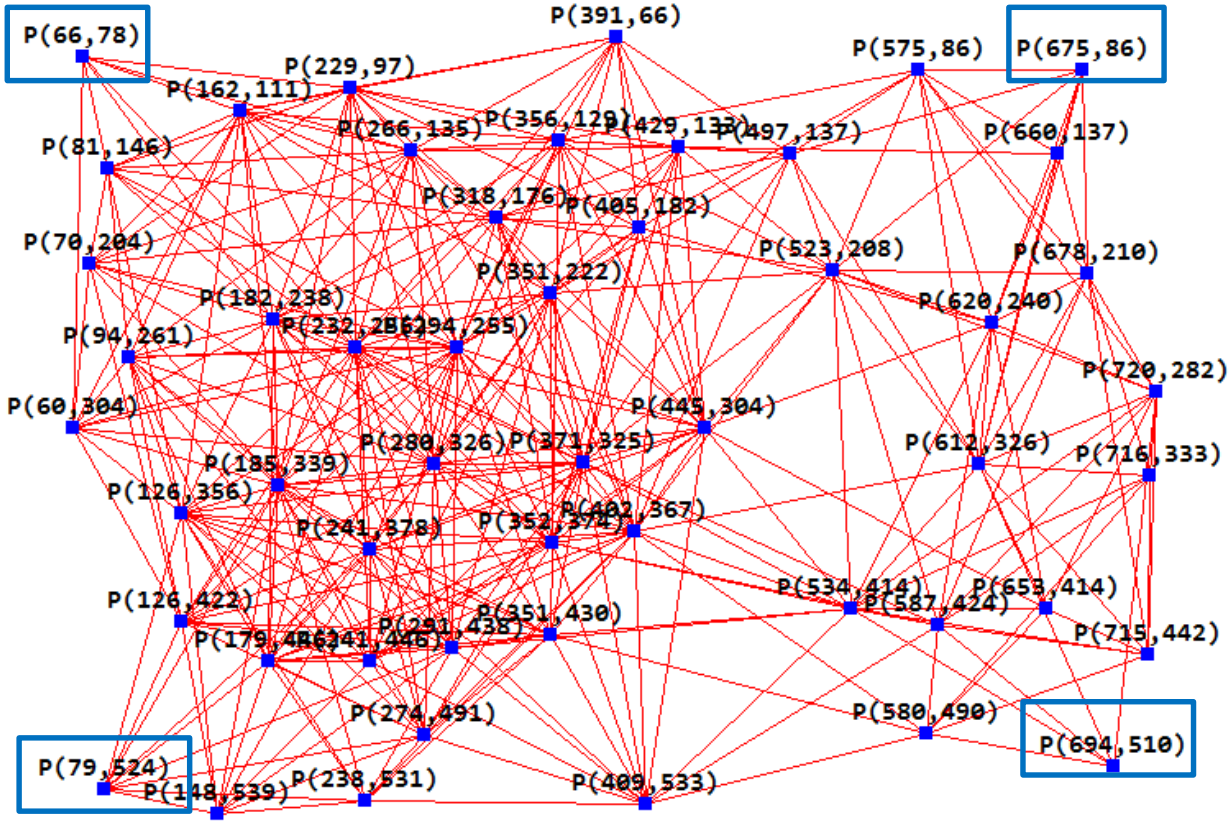
Shortest Path

- After solving problem 3, try to draw your solution like in the following graph5.dat, source=P(77,62), target=P(705,538)



Problem 6: Shortest Travel

- Create and draw graph6 (make sure to add distance to each edge!)
- Find the shortest path that starts and ends in **P(66,78)** and visits the cities: **P(675,86)**, **P(694,510)**, **P(79,524)** – in whatever order is required to make the trip as short as possible



EXTRA PROBLEMS (from book)

1. Draw a simple, connected, directed graph with 8 vertices and 16 edges such that the in-degree and out-degree of each vertex is 2. Show that there is a single (non-simple) cycle that includes all the edges of your graph, that is, you can trace all the edges in their respective directions without ever lifting your pencil. (Such a cycle is called an ***Euler tour***.)
2. Suppose we represent a graph G having n vertices and m edges with the edge list structure. Why, in this case, does the **insert_vertex** method run in $O(1)$ time while the **remove_vertex** method runs in $O(m)$ time?

EXTRA PROBLEMS (from book)

3. Bob loves foreign languages and wants to plan his course schedule for the following years. He is interested in the following nine language courses: LA15, LA16, LA22, LA31, LA32, LA126, LA127, LA141, and LA169.
   - The course prerequisites are:
     - LA15: (none)
     - LA16: LA15
     - LA22: (none)
     - LA31: LA15
     - LA32: LA16, LA31
     - LA126: LA22, LA32
     - LA127: LA16
     - LA141: LA22, LA16
     - LA169: LA32
   - In what order can Bob take these courses, respecting the prerequisites?

EXTRA PROBLEMS (from book)

4. Draw a simple, connected, weighted graph with 8 vertices and 16 edges, each with unique edge weights. Identify one vertex as a “start” vertex and illustrate a running of Dijkstra’s algorithm on this graph
5. Implement an algorithm that returns a cycle in a directed graph *G*, if one exists
6. Given an undirected graph G. Write a function **component(v)** which returns the connectivity component to which the vertex v belongs to
7. Test the solution to the previous problem on the graph:

   V = [0, 1, 2, 3, 4, …, 300]

   E = {(x,y) | (x-y)%7 == 0}

   Compute component(3)

EXTRA PROBLEMS (from book)

8. A graph G is bipartite if its vertices can be partitioned into two sets X and Y such that every edge in G has one end vertex in X and the other in Y.
   Design and analyze an efficient algorithm for determining if an undirected graph G is bipartite (without knowing the sets X and Y in advance).
9. An Euler tour of a directed graph G with n vertices and m edges is a cycle that traverses each edge of G exactly once according to its direction. Such a tour always exists if G is **connected** and **the in-degree equals the out-degree** of each vertex in G. Describe an O(n+m)-time algorithm for finding an Euler tour of such a directed graph G. Use Google to get help.