Problem-Sheet 6
My Solution

Michael Single

08-917-445

msingle@students.unibe.ch

Task1:

Given the undirected Graph G =(E, V) as shown in Figure 1

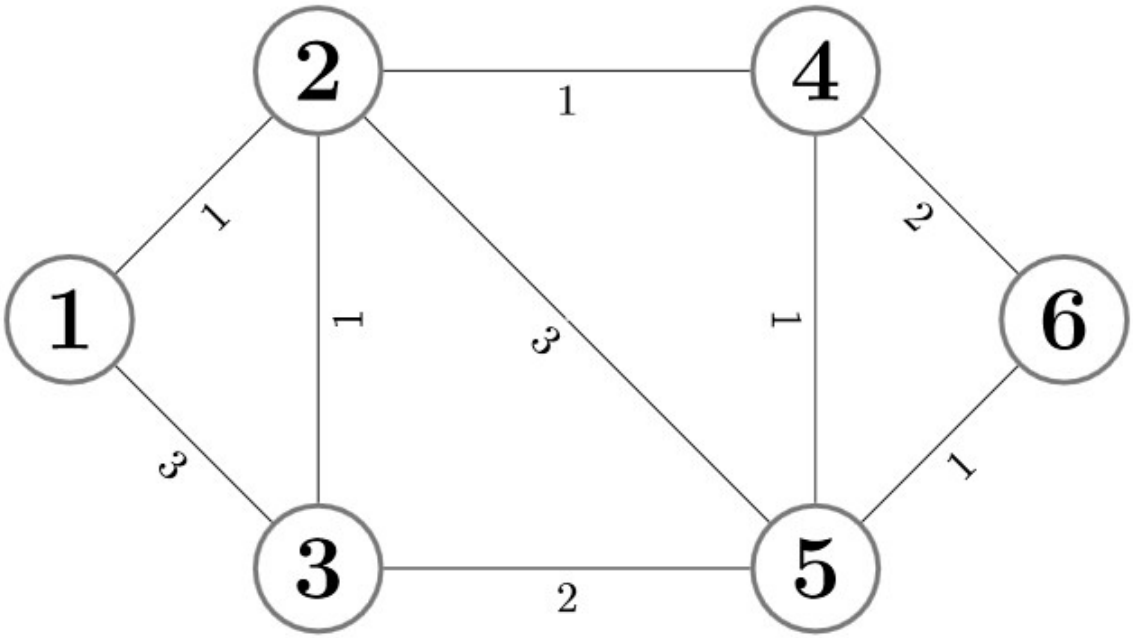

Figure 1.1: Exercise Graph G

a)

Give a linear Program that solves the problem of finding a shortest path from Vertex 1 to Vertex 6.

The general formulation of such a linear program for finding the shortest path from a given starting vertex s to a certain target vertex t is shown in figure 1.2.

Note that the variables

- $\mathbf{y_k}$ denotes the length of the shortest path from the source vertex s to the vertex k in the Graph G.
- **w_ij** denotes the weight of the edge ij – i.e. The edge from vertex i to vertex j.

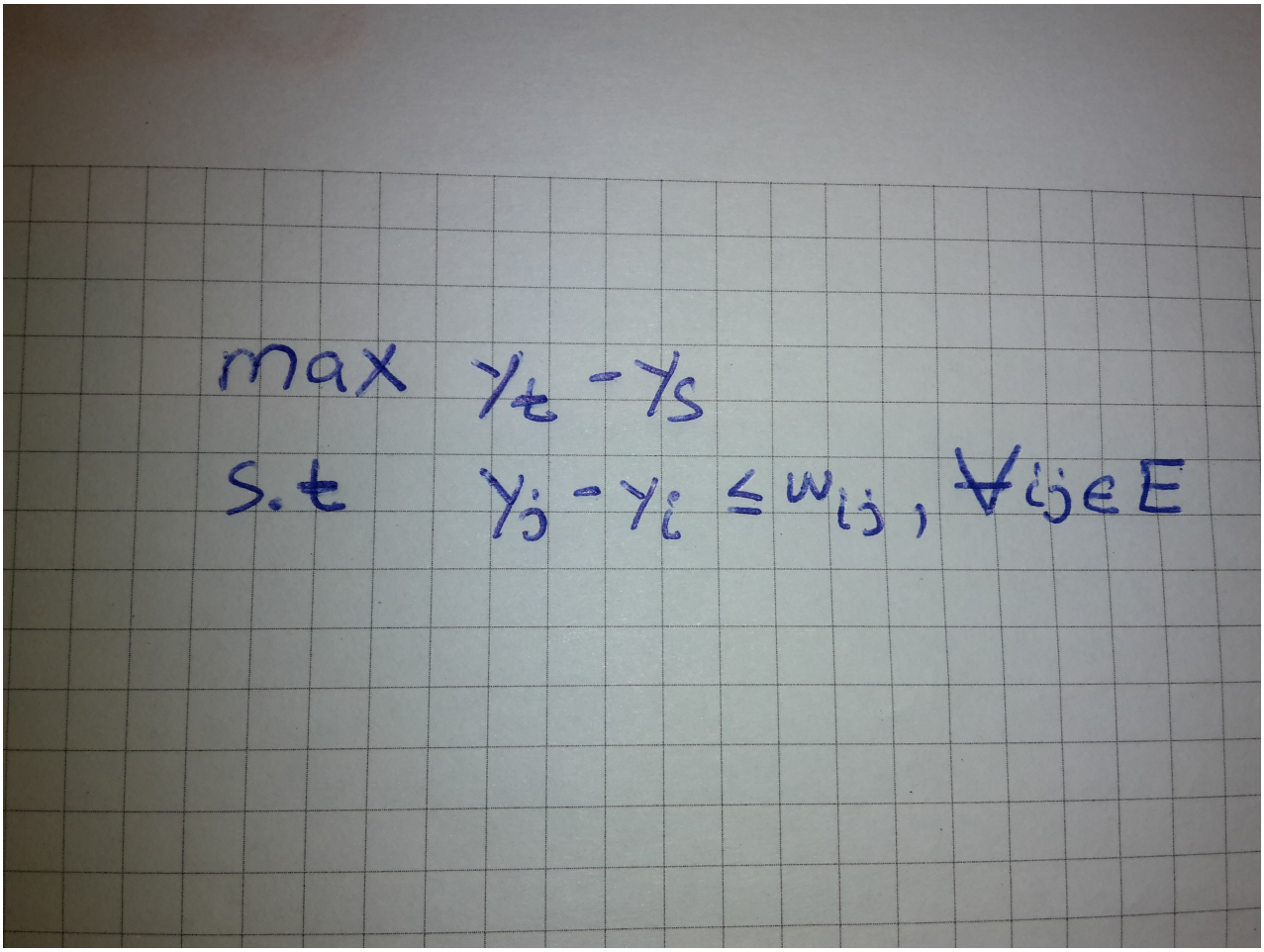

Figure 1.2:General LP for finding the Shortest Path

For the graph in figure 1.1, it has the following weight vector

w= [w_12, w_13, w_21, w_23, w_24, w_25, w_31,

w_32, w_35, w_42, w_45, w_46, w_52, w_53, w_54, w_56, w_64, w_65]

= [1,3,1,1,1,3,3,1,2,1,1,2,3,2,1,1,2,1]

Applying this LP to our given example graph G from figure 1.1 gives us the LP as shown in figure 1.3 below:

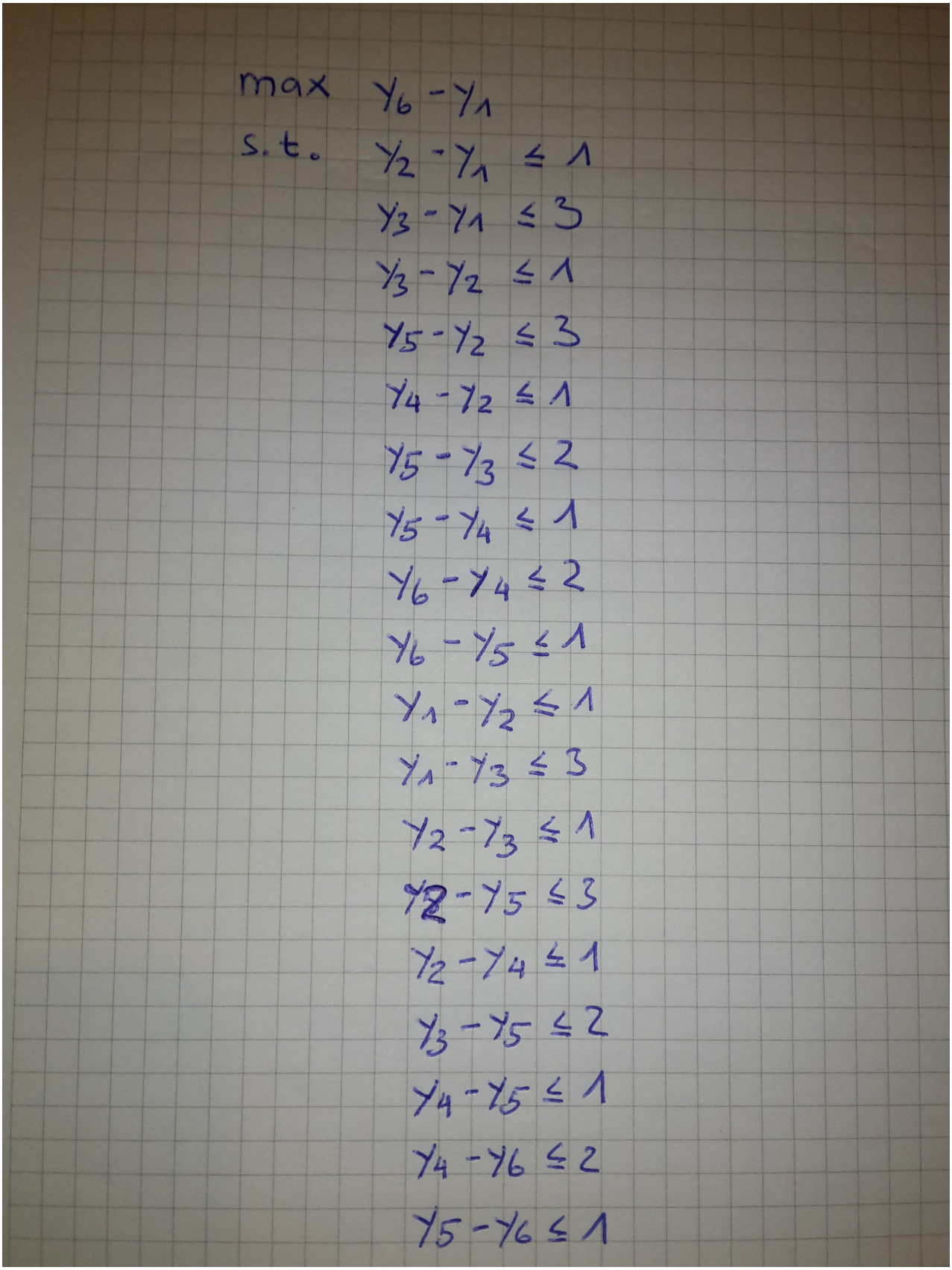

Figure 1.3: LP from figure 1.2 applied to the example graph from figure 1.1

b) ***Solve the linear program***

Solving the LP from a) we get **y = [y1, y6] = [0, 4]**

thus, the solution (for the maximum) is then: y_6 – y_1 = 4 – 0 = 4.

c) Derive the dual problem and explain it.

The general formulation of the dual problem of a) for finding the shortest path from a given starting vertex s to a certain target vertex t is shown in figure 1.4.

- **x_ij** is an indicator variable telling us whether an edge ij in E is part of the shortest path. If ij is in the shortest path, then x_ij is equal 1 otherwise it is equal 0.
- Note that since our graph G is undirected we have to consider both, x_ij AND x_ji
  - Corresponds to the same as we would have a directed graph having an edge from Vertex i to Vertex j and one edge from j to i.
- Here: **x** = [*x_12, x_13, x_21, x_23, x_24, x_25, x_31,*
  x_32, x_35, x_42, x_45, x_46, x_52, x_53, x_54, x_56, x_64, x_65]

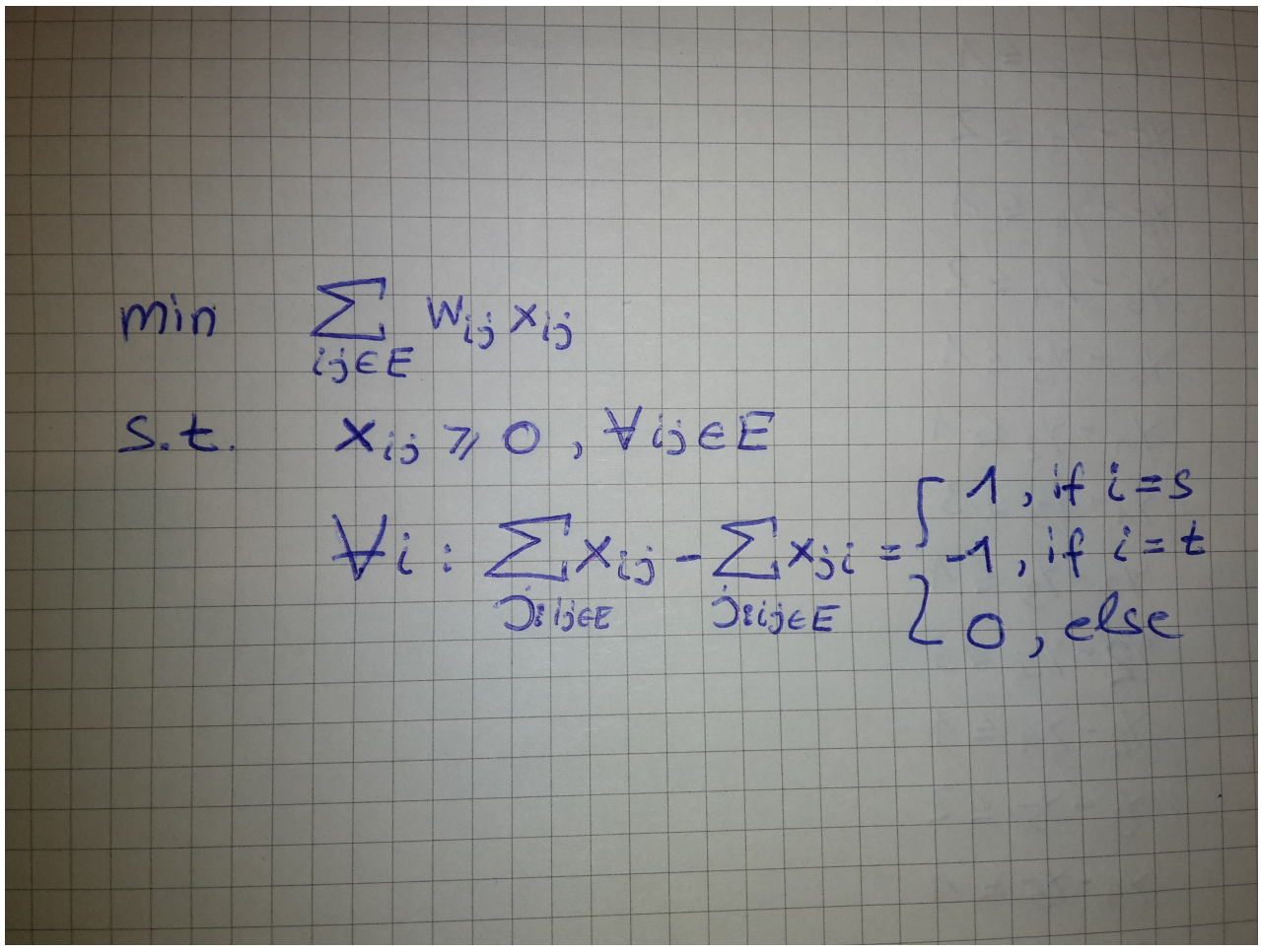

Figure 1.4: General Dual Problem from c)

Applying this dual Problem to our given example graph G from figure 1.1 gives us the LP as shown in figure 1.5 below:

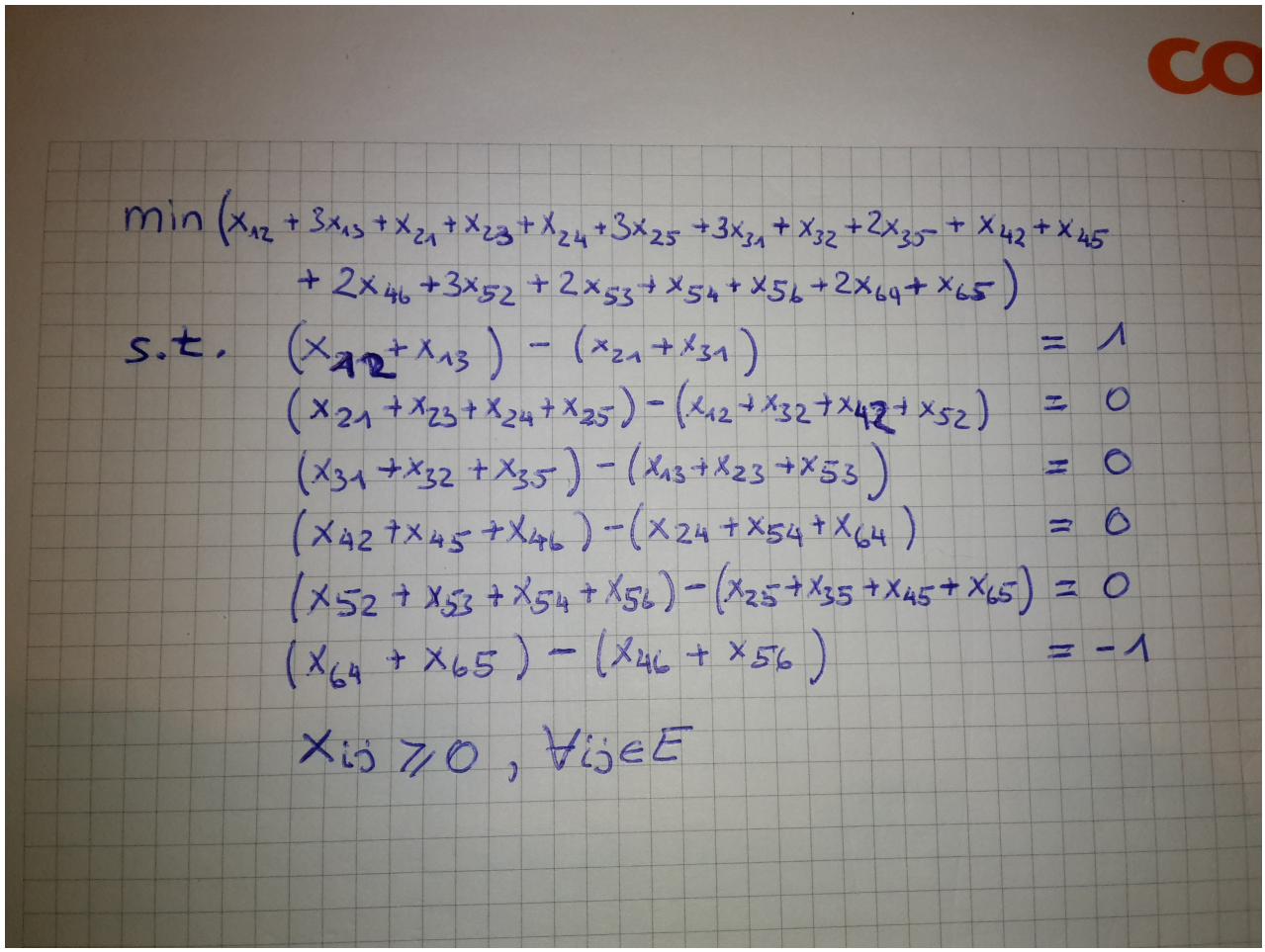

Figure 1.5: dual problem from figure 1.4 applied to the example graph from figure 1.1.

d) ***Solve the linear program***

Given our dual problem, we are looking for the entries of the vector

x = *[x_12, x_13, x_21, x_23, x_24, x_25, x_31,*

x_32, x_35, x_42, x_45, x_46, x_52, x_53, x_54, x_56, x_64, x_65]

There exists actually two possible solutions:

- solution#1: **x_12** = 1, **x_24** = 1 and **x_46** = 1 and every other **x_ij** of **x** is zero
- solution#2: **x_12** = 1, **x_24** = 1, **x_45** = 1 and **x_56** = 1 and every other **x_ij** in **x** is zero

Using the solution#1, we get the following minimum: x_21+x_24+x_46 = 1 + 1 + 2*1 = 4

Using the solution#2, we get the following minimum: x_21+x_24+x_45+x_56 = 1 + 1 + 1 + 1 = 4

Since the solution for the Primal and dual Problem give us both 4 – for every case – we have found the correct solution.

Note that there are two possible shortest path:

1. From v1=>v2=>v4=>v6
2. From v1=>v2=>v4=>v5=>v6

Both having a cost of 4.

Task2:

A directed graph G is given by the table from figure 2.1.

d_{ij}	1	2	3	4	5	6
1		2			9	
2	2					6
3	15	2		3	3	9
4						1
5			10	9		
6				4	3	

Figure 2.1: Adjacency matrix of example graph for task 2.

Implment the algorithm of Floyd and Warshall in a programming language of your chouce and test it on the graph from figure 2.1. Give the shortest paths and their lengths for all pairs of noces.

I have implemented the „ Floyd and Warshall“ Algoirthm in Ruby. You can find my source-code on github: https://github.com/simplay/combo

Please read the READMEmd + the WIKI on the Repository.

Usage: simply run: **ruby combo -t 2** in your console for solving for the shortest paths for the graph shown in figure 2.1.

Running my code I get the following output:

```
directed Graph G=(V,E,w)

6 vertices V with their label values l:
1 l: -1
2 l: -1
3 l: -1
4 l: -1
5 l: -1
6 l: -1

14 edges E with their weight w:
```

1->2 w: 2.0
1->5 w: 9.0
2->1 w: 2.0
2->6 w: 6.0
3->1 w: 15.0
3->2 w: 2.0
3->4 w: 3.0
3->5 w: 3.0
3->6 w: 9.0
4->6 w: 1.0
5->3 w: 10.0
5->4 w: 9.0
6->4 w: 4.0
6->5 w: 3.0

All shortest paths for each vertex pair (i,j):
Distance from vertex i=1 to vertex j=1: 0.0 path: does not exist
Distance from vertex i=1 to vertex j=2: 2.0 path: [1,2]
Distance from vertex i=1 to vertex j=3: 19.0 path: [1,5,3]
Distance from vertex i=1 to vertex j=4: 12.0 path: [1,2,6,4]
Distance from vertex i=1 to vertex j=5: 9.0 path: [1,5]
Distance from vertex i=1 to vertex j=6: 8.0 path: [1,2,6]
Distance from vertex i=2 to vertex j=1: 2.0 path: [2,1]
Distance from vertex i=2 to vertex j=2: 0.0 path: does not exist
Distance from vertex i=2 to vertex j=3: 19.0 path: [2,6,5,3]
Distance from vertex i=2 to vertex j=4: 10.0 path: [2,6,4]
Distance from vertex i=2 to vertex j=5: 9.0 path: [2,6,5]
Distance from vertex i=2 to vertex j=6: 6.0 path: [2,6]
Distance from vertex i=3 to vertex j=1: 4.0 path: [3,2,1]
Distance from vertex i=3 to vertex j=2: 2.0 path: [3,2]
Distance from vertex i=3 to vertex j=3: 0.0 path: does not exist
Distance from vertex i=3 to vertex j=4: 3.0 path: [3,4]
Distance from vertex i=3 to vertex j=5: 3.0 path: [3,5]
Distance from vertex i=3 to vertex j=6: 4.0 path: [3,4,6]
Distance from vertex i=4 to vertex j=1: 18.0 path: [4,6,5,3,2,1]
Distance from vertex i=4 to vertex j=2: 16.0 path: [4,6,5,3,2]
Distance from vertex i=4 to vertex j=3: 14.0 path: [4,6,5,3]
Distance from vertex i=4 to vertex j=4: 0.0 path: does not exist
Distance from vertex i=4 to vertex j=5: 4.0 path: [4,6,5]
Distance from vertex i=4 to vertex j=6: 1.0 path: [4,6]
Distance from vertex i=5 to vertex j=1: 14.0 path: [5,3,2,1]
Distance from vertex i=5 to vertex j=2: 12.0 path: [5,3,2]
Distance from vertex i=5 to vertex j=3: 10.0 path: [5,3]
Distance from vertex i=5 to vertex j=4: 9.0 path: [5,4]
Distance from vertex i=5 to vertex j=5: 0.0 path: does not exist
Distance from vertex i=5 to vertex j=6: 10.0 path: [5,4,6]
Distance from vertex i=6 to vertex j=1: 17.0 path: [6,5,3,2,1]
Distance from vertex i=6 to vertex j=2: 15.0 path: [6,5,3,2]
Distance from vertex i=6 to vertex j=3: 13.0 path: [6,5,3]
Distance from vertex i=6 to vertex j=4: 4.0 path: [6,4]
Distance from vertex i=6 to vertex j=5: 3.0 path: [6,5]
Distance from vertex i=6 to vertex j=6: 0.0 path: does not exist

The most important code is in the file „src/adj_matrix.rb“

```
class AdjMatrix

  attr_accessor :schema

  ALMOST_INF = (1 << (1.size * 8 - 2) - 1)

  def initialize(graph)
    @vertices = graph.vertices
    @v_count = @vertices.count

    @schema = []
    @next = []

    # init adj. mat
    @v_count.times do
      a_row = []
      a_next_row = []
      @v_count.times do
        a_row << ALMOST_INF
        a_next_row << nil
      end
      @schema << a_row
      @next << a_next_row
    end

    #foreach edge label
    graph.edges.each do |edge|
      @schema[edge.u][edge.v] = edge.weight
      @next[edge.u][edge.v] = edge.to
    end

    graph.vertices.each do |vertex|
      v_idx = vertex.idx
      @schema[v_idx][v_idx] = 0.0
    end

  end

  # Floyd-Warshall algorithm
  def shortes_paths
    distances = copied_schema

    @v_count.times do |k|
      @v_count.times do |i|
        @v_count.times do |j|
          relax(distances, i, j, k)
        end
      end
    end
    distances
  end
```

```
  # get path from a given vertex to a given vertex
  # convention: report 'path does not exist'
  # in case we query for path from Vertex v to Vertex v.
  # @param from Vertex start
  # @param to Vertex destination
  def shortest_path(from, to)
    u = from; v = to
    return "does not exist" if @next[u.idx][v.idx].nil?

    path = ("["+u.to_s + ",")

    until(u.eql?(v))
      u = @next[u.idx][v.idx]
      path += (u.to_s + ",")
    end
    path = path[0..path.size-2]
    path += "]"
    path
  end

  def to_s
    @schema.each do |row|
      row.each do |element|
        value = element.eql?(ALMOST_INF) ? "inf" : element.to_s
        print value + " "
      end
      puts
    end
  end

  private

  # update (i,j)-th element of given container
  # if it can be relaxed
  def relax(container, i, j, k)
    if(container[i][j] > container[i][k] + container[k][j])
      container[i][j] = container[i][k] + container[k][j]
      @next[i][j] = @next[i][k]
    end
  end

  # copy each value in the :schema into
  # a new instantiated 2d array.
  def copied_schema
    c = []
    @v_count.times do |i|
      c_row = []
      @v_count.times do |j|
        c_row << @schema[i][j]
      end
      c << c_row
    end
    c
  end

end
```