# Comp151 Lab14b

Lab14b consists of **two independent graph applications:**

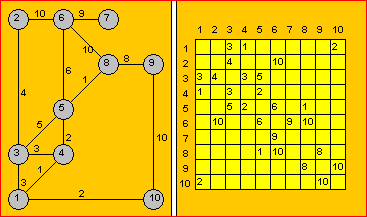
* MinimumSpanningTree
* HowManyQuestions

# MinimumSpanningTree

Implement *Prim’s Algorithm* for finding a **Minimum Spanning Tree** (MST) embedded in a **weighted connected** graph. The MST problem is to find a tree that is made up of all the nodes in the graph and a subset of the edges such that the sum of the edge weights is a minimum.

The algorithm is stated formally as follows: <http://en.wikipedia.org/wiki/Prim%27s_algorithm>

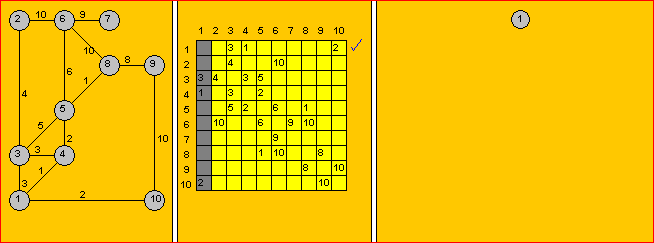
MinimumSpanningTree class uses the **adjacency matrix** to represent the graph. This array is specified by the costs. main asks the user for the number of nodes and the probability of edge existing between two vertices. Constructor is to generate weights and edges randomly. Since we will not use directed graphs, this array will be symmetrical:

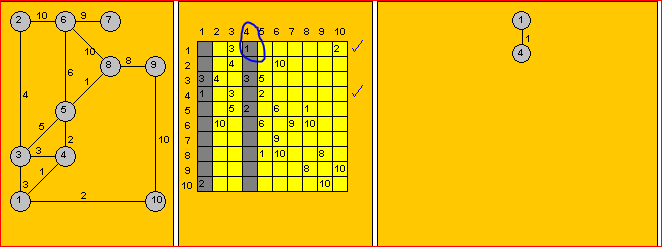


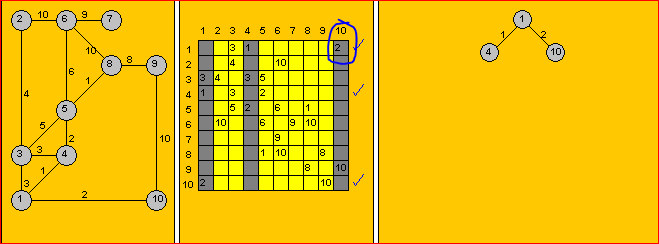
The *Prim’s Algorithm* steps are as follow:

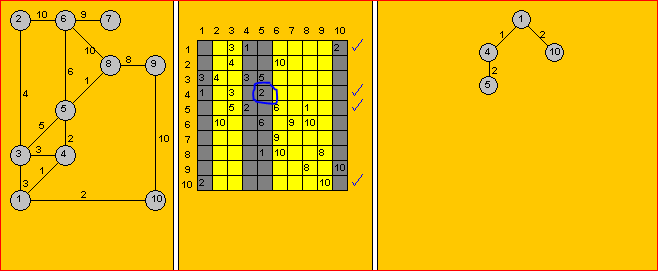
1. *We put a mark beside the first row and we gray-out the first column*
2. *Between the elements that are not grayed-out and belong to a row with a mark we choose the least A(j,k). If all elements are grayed-out, the algorithm terminates.*
3. *We put mark beside the kth row and we gray-out the kth column. We return to step b.*

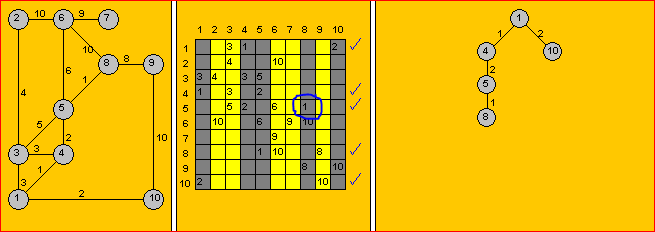
NOTE that in every step *Vk* node becomes *Vj*‘s child:

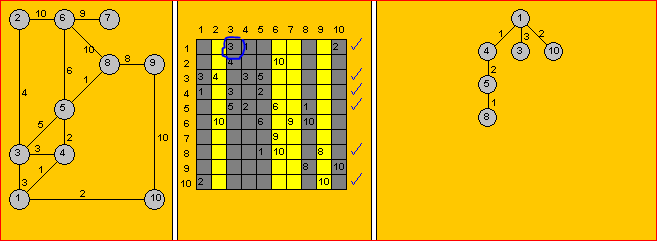


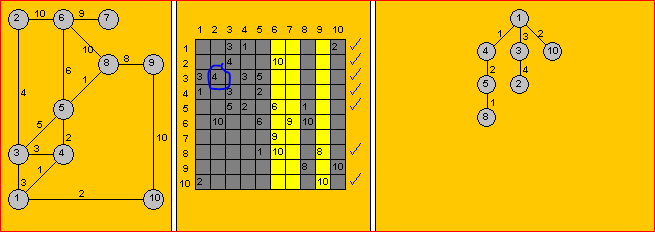


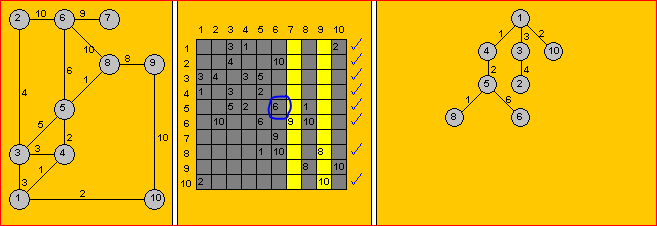


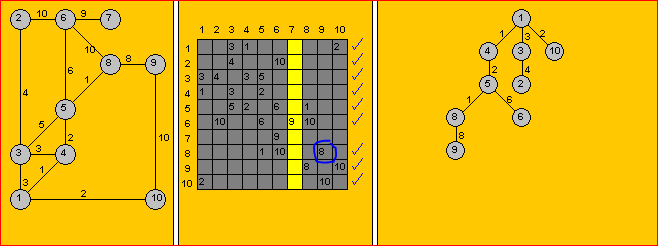


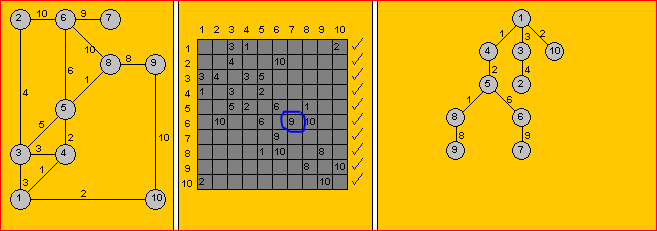












Your task is to complete the class implementation. The output of the finished program should display the adjacency matrix for the graph that is connected, and the computed adjacency matrix for the minimum spanning tree for that graph. Notice that row 0 and column zero are not displayed since they are used for checkmarks. For example, see a sample run below:

How many nodes in your graph?

10

Probability of edge? (type 70 for 70%)

50

\*\*\*\*\* GENERATED GRAPH \*\*\*\*\*

[ 1][ 2][ 3][ 4][ 5][ 6][ 7][ 8][ 9][10]

[ 1] 3 1 2

[ 2] 4 10

[ 3] 3 4 3 5

[ 4] 1 3 2

[ 5] 5 2 6 1

[ 6] 10 6 9 10

[ 7] 9

[ 8] 1 10 8

[ 9] 8 10

[10] 2 10

The graph has cycles.

\*\*\*\*\* MINIMUM SPANNING TREE FOR THE ABOVE GRAPH \*\*\*\*\*

[ 1][ 2][ 3][ 4][ 5][ 6][ 7][ 8][ 9][10]

[ 1] 3 1 2

[ 2] 4

[ 3] 3 4

[ 4] 1 2

[ 5] 2 6 1

[ 6] 6 9

[ 7] 9

[ 8] 1 8

[ 9] 8

[10] 2

The minimum spanning tree is acyclic.

Process finished with exit code 0

If the generated graph is not connected the minimum spanning tree cannot be computed. For example, see a sample run below (to get this matrix I used Random with seed of 101):

How many nodes in your graph?

8

Probability of edge? (type 70 for 70%)

10

\*\*\*\*\* GENERATED GRAPH \*\*\*\*\*

[ 1][ 2][ 3][ 4][ 5][ 6][ 7][ 8]

[ 1]

[ 2] 3 4

[ 3]

[ 4] 3

[ 5] 3

[ 6] 4

[ 7] 3 2

[ 8] 4 4 2

The graph is not connected, the minimum spanning tree will not be calculated

Process finished with exit code 0

The following UML design was utilized in implementation of the above solution:

# /Users/ania/Desktop/Untitled.jpeg

# HowManyQuestions

Your teacher is going to give a test where each student is to answer one question. None of the neighboring students should have the same question. How many questions are needed?

Graph Coloring Algorithm is used to solve this type of problems. It does not guarantee to use the minimum number of questions, but it guarantees an upper bound on the number of questions. The algorithm never uses more than d+1 questions where d is the maximum degree of vertices in the given graph – where the *degree of a vertex* is the number of edges connected to the vertex.

The Basic Greedy Graph Coloring Algorithm is as follow:

1. Assign first vertex with the first color.
2. Do following for remaining V-1 vertices.
   1. Consider the currently picked vertex and color it with the lowest numbered color that has not been used on any previously colored vertices adjacent to it. If all previously used colors appear on vertices adjacent to v, assign a new color to it.
   2. If there are no more colors available to choose from the problem cannot be solved

Your task is to utilize the above concept to compute how many questions are needed and which question should be assigned to each student. Constructor is given a boolean matrix that indicates if a connection exists between vertexes. Based on the passed parameter the number of vertices is saved in instance variable and a non-directed graph that is represented as **adjacency list** is created. A **vertex** in the graph represents a **student**, an **edge** connects a pair of students that are **neighbors**.

Your program starts with the number of questions set to two and tries to find the solution where none of the neighbors have the same question. If the solution is not found one more question is added, and the process is repeated. The program stops when the solution is found and displays to the user the number of questions needed to satisfy the requirement.

See the sample run below:

Created graph 1:

The graph has 5 vertexes with the following neighbors:

Vertex 0 has neighbors: 1 2

Vertex 1 has neighbors: 0 2 3

Vertex 2 has neighbors: 0 1 3

Vertex 3 has neighbors: 1 2 4

Vertex 4 has neighbors: 3

=================

Created graph 2:

The graph has 5 vertexes with the following neighbors:

Vertex 0 has neighbors: 1 2 3

Vertex 1 has neighbors: 0 2 3 4

Vertex 2 has neighbors: 0 1 4

Vertex 3 has neighbors: 0 1 4

Vertex 4 has neighbors: 1 2 3

=================

Created graph 3:

The graph has 5 vertexes with the following neighbors:

Vertex 0 has neighbors: 1 2 3

Vertex 1 has neighbors: 0 2 3 4

Vertex 2 has neighbors: 0 1 3

Vertex 3 has neighbors: 0 1 2 4

Vertex 4 has neighbors: 1 3

=================

Created graph 4:

The graph has 24 vertexes with the following neighbors:

Vertex 0 has neighbors: 1 23

Vertex 1 has neighbors: 0 2

Vertex 2 has neighbors: 1 3

Vertex 3 has neighbors: 2 4

Vertex 4 has neighbors: 3 5

Vertex 5 has neighbors: 4 6

Vertex 6 has neighbors: 5 7

Vertex 7 has neighbors: 6 8

Vertex 8 has neighbors: 7 9

Vertex 9 has neighbors: 8 10

Vertex 10 has neighbors: 9 11

Vertex 11 has neighbors: 10 12

Vertex 12 has neighbors: 11 13

Vertex 13 has neighbors: 12 14

Vertex 14 has neighbors: 13 15

Vertex 15 has neighbors: 14 16

Vertex 16 has neighbors: 15 17

Vertex 17 has neighbors: 16 18

Vertex 18 has neighbors: 17 19

Vertex 19 has neighbors: 18 20

Vertex 20 has neighbors: 19 21

Vertex 21 has neighbors: 20 22

Vertex 22 has neighbors: 21 23

Vertex 23 has neighbors: 0 22

=================

Created graph 5:

The graph has 24 vertexes with the following neighbors:

Vertex 0 has neighbors: 1 10 11

Vertex 1 has neighbors: 0 2 9 10 11

Vertex 2 has neighbors: 1 3 8 9 10

Vertex 3 has neighbors: 2 4 7 8 9

Vertex 4 has neighbors: 3 5 6 7 8

Vertex 5 has neighbors: 4 6 7

Vertex 6 has neighbors: 4 5 7 16 17

Vertex 7 has neighbors: 3 4 5 6 8 15 16 17

Vertex 8 has neighbors: 2 3 4 7 9 15 16 17

Vertex 9 has neighbors: 1 2 3 8 10 13 14 15

Vertex 10 has neighbors: 0 1 2 9 11 12 13 14

Vertex 11 has neighbors: 0 1 10 12 13

Vertex 12 has neighbors: 10 11 13 22 23

Vertex 13 has neighbors: 9 10 11 12 14 21 22 23

Vertex 14 has neighbors: 8 9 10 13 15 20 21 22

Vertex 15 has neighbors: 7 8 9 14 16 19 20 21

Vertex 16 has neighbors: 6 7 8 15 17 18 19 20

Vertex 17 has neighbors: 6 7 16 18 19

Vertex 18 has neighbors: 16 17 19

Vertex 19 has neighbors: 15 16 17 18 20

Vertex 20 has neighbors: 14 15 16 19 21

Vertex 21 has neighbors: 13 14 15 20 22

Vertex 22 has neighbors: 12 13 14 21 23

Vertex 23 has neighbors: 12 13 22

=================

\*\*\*\*\*\* Checking if 2 questions are enough \*\*\*\*\*\*

\*\*\* Checking graph 1 \*\*\*

--> The solution does not exist - not enough choices

\*\*\* Checking graph 2 \*\*\*

--> The solution does not exist - not enough choices

\*\*\* Checking graph 3 \*\*\*

--> The solution does not exist - not enough choices

\*\*\* Checking graph 4 \*\*\*

--> The solution exists with 2 questions:

Student 0 ---> Question 0

Student 1 ---> Question 1

Student 2 ---> Question 0

Student 3 ---> Question 1

Student 4 ---> Question 0

Student 5 ---> Question 1

Student 6 ---> Question 0

Student 7 ---> Question 1

Student 8 ---> Question 0

Student 9 ---> Question 1

Student 10 ---> Question 0

Student 11 ---> Question 1

Student 12 ---> Question 0

Student 13 ---> Question 1

Student 14 ---> Question 0

Student 15 ---> Question 1

Student 16 ---> Question 0

Student 17 ---> Question 1

Student 18 ---> Question 0

Student 19 ---> Question 1

Student 20 ---> Question 0

Student 21 ---> Question 1

Student 22 ---> Question 0

Student 23 ---> Question 1

\*\*\* Checking graph 5 \*\*\*

--> The solution does not exist - not enough choices

\*\*\*\*\*\* Checking if 3 questions are enough \*\*\*\*\*\*

\*\*\* Checking graph 1 \*\*\*

--> The solution exists with 3 questions:

Student 0 ---> Question 0

Student 1 ---> Question 1

Student 2 ---> Question 2

Student 3 ---> Question 0

Student 4 ---> Question 1

\*\*\* Checking graph 2 \*\*\*

--> The solution exists with 3 questions:

Student 0 ---> Question 0

Student 1 ---> Question 1

Student 2 ---> Question 2

Student 3 ---> Question 2

Student 4 ---> Question 0

\*\*\* Checking graph 3 \*\*\*

--> The solution does not exist - not enough choices

\*\*\* Checking graph 5 \*\*\*

--> The solution does not exist - not enough choices

\*\*\*\*\*\* Checking if 4 questions are enough \*\*\*\*\*\*

\*\*\* Checking graph 3 \*\*\*

--> The solution exists with 4 questions:

Student 0 ---> Question 0

Student 1 ---> Question 1

Student 2 ---> Question 2

Student 3 ---> Question 3

Student 4 ---> Question 0

\*\*\* Checking graph 5 \*\*\*

--> The solution exists with 4 questions:

Student 0 ---> Question 0

Student 1 ---> Question 1

Student 2 ---> Question 0

Student 3 ---> Question 1

Student 4 ---> Question 0

Student 5 ---> Question 1

Student 6 ---> Question 2

Student 7 ---> Question 3

Student 8 ---> Question 2

Student 9 ---> Question 3

Student 10 ---> Question 2

Student 11 ---> Question 3

Student 12 ---> Question 0

Student 13 ---> Question 1

Student 14 ---> Question 0

Student 15 ---> Question 1

Student 16 ---> Question 0

Student 17 ---> Question 1

Student 18 ---> Question 2

Student 19 ---> Question 3

Student 20 ---> Question 2

Student 21 ---> Question 3

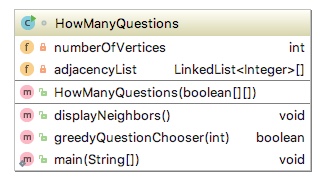
Student 22 ---> Question 2

Student 23 ---> Question 3

\*\*\*\*\* DONE - all graphs were assigned solutions \*\*\*\*\*

Process finished with exit code 0

The following UML design was utilized in implementation of the above solution:

****

**This is your last Comp151 project, have fun!!!**