Why we chose vector of vectors structure ?

We decided to choose the data structure of vector of vectors because it has a lot of benefits such as: easy access to each vertice, the capacity of the vector increased according to the edges we added. The data-structure vector has a lot of functions we can use for example: push\_back, pop\_back, size(), assign, empty and clear.

Text

Description automatically generated

This function receives the number of vertices and the probability. We build vector of vectors that saved the neighbors of each vertices if it has one.

A screenshot of a computer

Description automatically generated with medium confidenceThis function receives the probability and define the lower to 1 and the upper to 100000. Then we get the result. If the result smaller or equal to k the probability will be p else 1-p.

Text

Description automatically generated

This function initialize all the vertices with white color, and then send each vertices with white color to another function(BFS\_Source()).

Before that we define the size and the content for each of the vectors:

Color, d(distance) and p(parent).

In the end it returns the vector d that include all the distances.

Text

Description automatically generated

This function receives the graph and the verticies with white color. We created a queue that we push to him each vertice we mate(and the first one we received). For each verticie we change his color from white to green. as long as the queue not empty we extract the first vertice in the queue and run on all its neighbors. Than we change the neighbores color to green, increase the distance by 1 and change theirs "father" (to the vertice we popped out from the queue). In the next step we push the neighbors to the queue. For each vertice we finished run over all his neighbors, we change the color to dark green.

Text

Description automatically generated

This function finds and return the max distance between all the distances from the distance vector that we got from the BFSFull() function.

Text

Description automatically generated

This function check if the graph is connected by checking if each vertice has a neighbor. If it doesn’t we add 1 to the counter (=the amount of the components are increasing). If the counter bigger then 1 its means we received more than one component and then we return 0. Otherwise return 1.

Text

Description automatically generated

This function check if the graph has isolated vertice by checking if each vertice has neighbors. If it doesn’t, we return 1 else 0.

Graphical user interface

Description automatically generated with medium confidence

This function is responsible to create and write in a specific csv file.

Main():

Text

Description automatically generated

We defined our threshold arrays that contains the probability which the firsts 5 indexes are smaller than their specific threshold and the other 5 indexes are bigger.

Text

Description automatically generated

When we enter the first loop, we define to which of the files to write to (we send the probability – threshold array).

In this section depands on the first loop, we starts to create the graphs using build\_random\_graph(), than we started running on the other functions: is\_isolated(), is\_connectivity() and diameter(). (again depends on the first loop). We are using counters which help us detect how many graphs are isolated, connected and their diameter are equal and bigger than 2.

Text

Description automatically generated

In context to the part above, we stores the counters divided by 500 in their designate counter-array, than we initialize the counters to 0.

Text

Description automatically generated

In this part we call write\_record\_to\_file() function again, but this time we sent the counter-arrays, that contains the results.

**SIMULATIONS**

**Chart, histogram

Description automatically generated**

**Chart, bar chart

Description automatically generated**

**Chart

Description automatically generated**