

Мухамадиев Владимир

Задание 6

Загрузка и предварительная обработка

```
In[1]:= data1 = Import[NotebookDirectory[] <> "\\bio-yeast.txt", "Data"];  
           импорт  директория файла блокнота  
  
In[2]:= e11 = Table[data1[[i, 1]] ↔ data1[[i, 2]], {i, 1, Length[data1]}];  
           таблица значений                                длина  
  
In[3]:= g1 = Graph[e11];  
           граф  
  
In[4]:= data2 = Import[NotebookDirectory[] <> "\\ca-netscience.txt", "Data"];  
           импорт  директория файла блокнота  
  
In[5]:= e12 = Table[data2[[i, 1]] ↔ data2[[i, 2]], {i, 1, Length[data2]}];  
           таблица значений                                длина  
  
In[6]:= g2 = Graph[e12];  
           граф  
  
In[7]:= data3 = Import[NotebookDirectory[] <> "\\proteins.txt", "Data"];  
           импорт  директория файла блокнота  
  
In[8]:= e13 = Table[data3[[i, 1]] ↔ data3[[i, 2]], {i, 1, Length[data3]}];  
           таблица значений                                длина  
  
In[9]:= g3 = Graph[e13];  
           граф
```

1. Матрица корреляций степеней вершин

Матрица корреляций степеней вершин заданного графа

```
In[10]:= DegreeCorrelationMatrix[graph_] :=  
           Module[{el = EdgeList[graph], ec = EdgeCount[graph]}, Divide[Normal[  
           |программный... |список рёбер |число рёбер |разде... |нормальное выражение  
           SparseArray[Normal[Counts[Partition[Flatten[{Table[{VertexDegree[graph, el[[i, 1]]],  
           |разрежённый... |норма... |встре... |разбиение... |уплостить |таблиц... |степень вершины  
           VertexDegree[graph, el[[i, 2]]}], {i, 1, ec}], Table[{VertexDegree[graph,  
           |степень вершины |таблиц... |степень вершины  
           el[[i, 2]], VertexDegree[graph, el[[i, 1]]}], {i, 1, ec}]]], 2]]], 2 ec]]  
           |степень вершины
```

Визуализация матрицы корреляций степеней вершин заданного графа

```
In[11]:= DegreeCorrelationMatrixPlot[graph_, zeros_]:=
  If[zeros == True, MatrixPlot[DegreeCorrelationMatrix[graph],
    |условный о... |ист... |визуализация матрицы
    DataReversed -> {True, False}, PlotLegends -> Automatic], MatrixPlot[
    |обратный порядок... |ист... |ложь |легенды графика |автоматичес... |визуализация матрицы
    DeleteCases[DeleteCases[DegreeCorrelationMatrix[graph], {0..}]]^T, {0..}]]^T,
    |удалить случ... |удалить случаи по образцу
    DataReversed -> {True, False}, FrameTicks -> None, PlotLegends -> Automatic]]
    |обратный порядок... |ист... |ложь |деления на о... |ни о... |легенды графика |автоматический
```

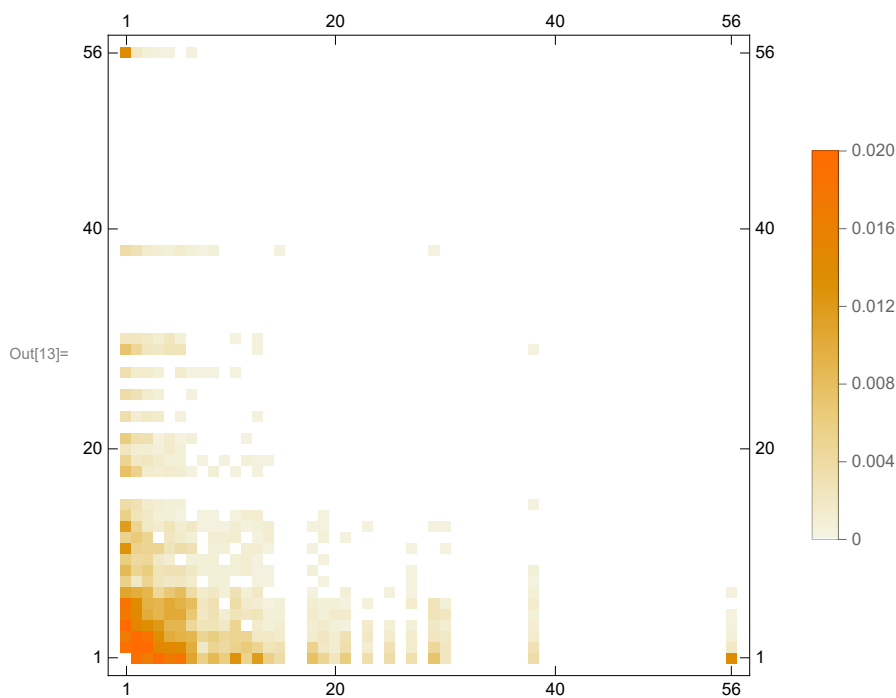
Коэффициент ассортативности по заданной матрице корреляций степеней вершин графа

```
In[12]:= AssortativityCoefficient[mat_] :=
  Module[{n = Length[mat]}, 
$$\frac{\sum_{i=1}^n \sum_{j=1}^n i j (\text{mat}[[i, j]] - (\sum_{j=1}^n \text{mat}[[i, j]] \sum_{i=1}^n \text{mat}[[j, i]])}{\sum_{i=1}^n i^2 \sum_{j=1}^n \text{mat}[[i, j]] - (\sum_{i=1}^n i \sum_{j=1}^n \text{mat}[[i, j]])^2}$$
]
  |программны... |длина
```

Для заданных графов

bio-yeast

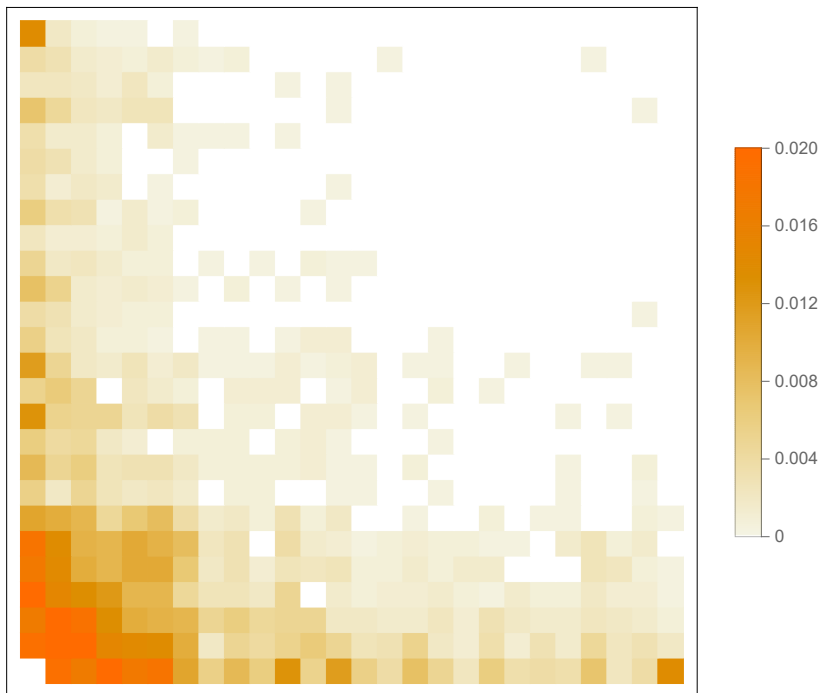
```
In[13]:= DegreeCorrelationMatrixPlot[g1, True]
|истина
```



In[14]:= DegreeCorrelationMatrixPlot[g1, False]

ложь

Out[14]=



In[15]:= N[AssortativityCoefficient[DegreeCorrelationMatrix[g1]]]

численное приближение

Out[15]= -0.209541

In[16]:= N[GraphAssortativity[g1]]

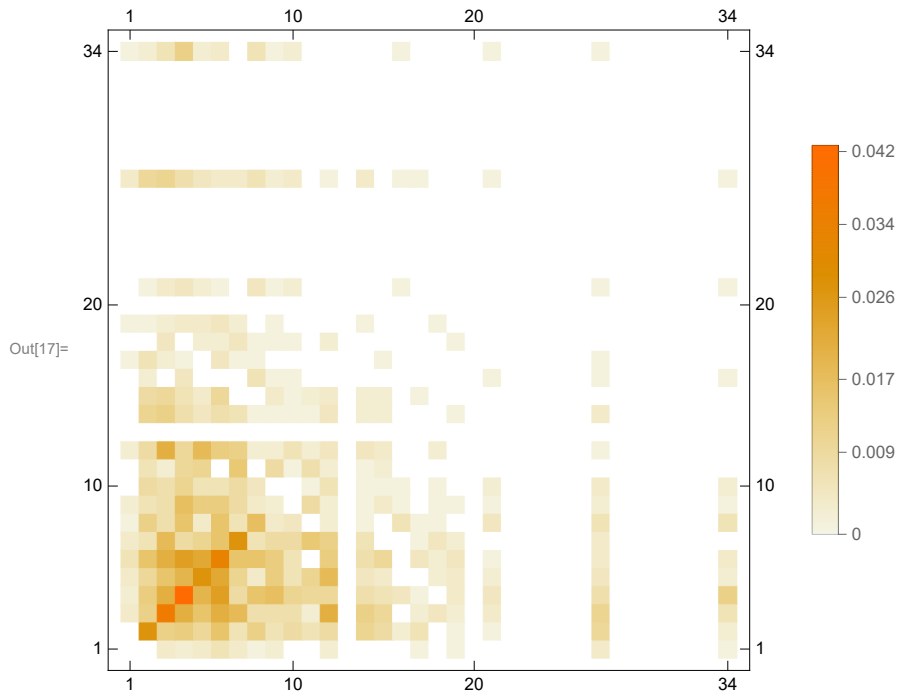
ассортативность графа

Out[16]= -0.209541

ca-netscience

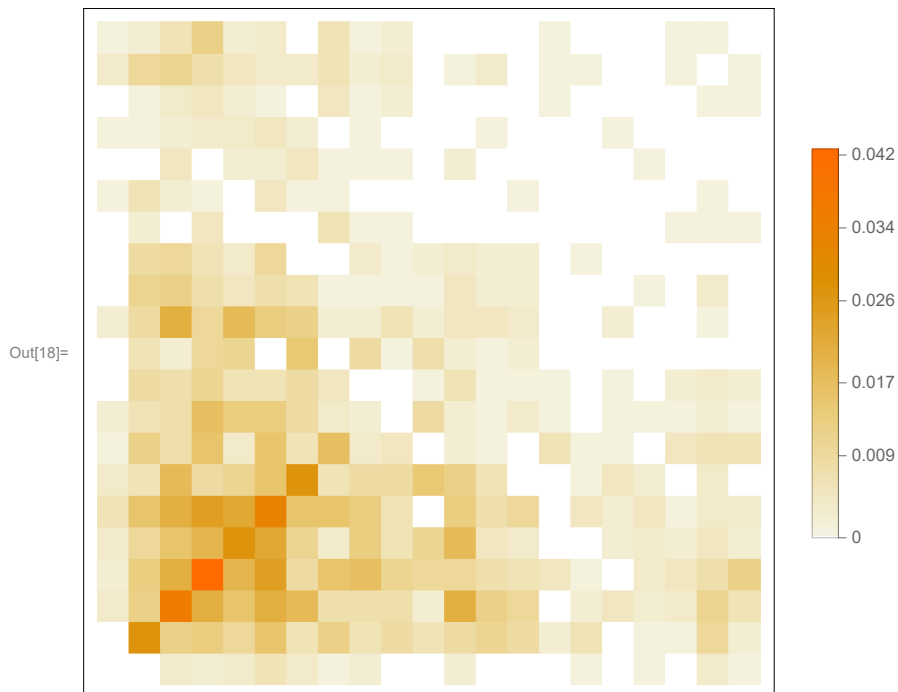
In[17]:= DegreeCorrelationMatrixPlot[g2, True]

Истина



In[18]:= DegreeCorrelationMatrixPlot[g2, False]

ложь



In[19]:= N[AssortativityCoefficient[DegreeCorrelationMatrix[g2]]]

численное приближение

Out[19]= -0.0816778

In[20]:= **N[GraphAssortativity[g2]]**

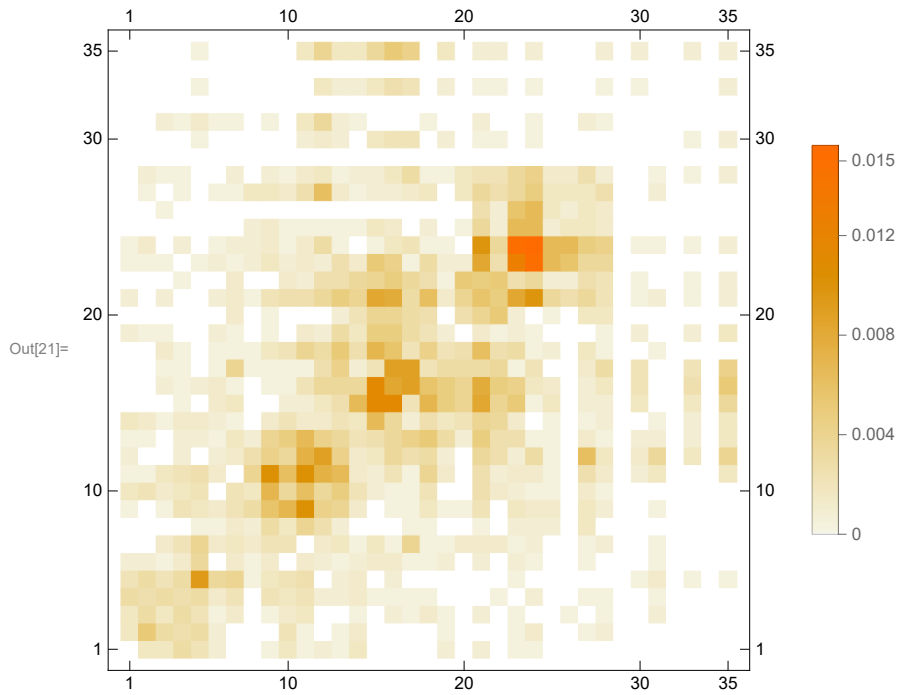
·· [ассортативность графа]

Out[20]:= **-0.0816778**

proteins

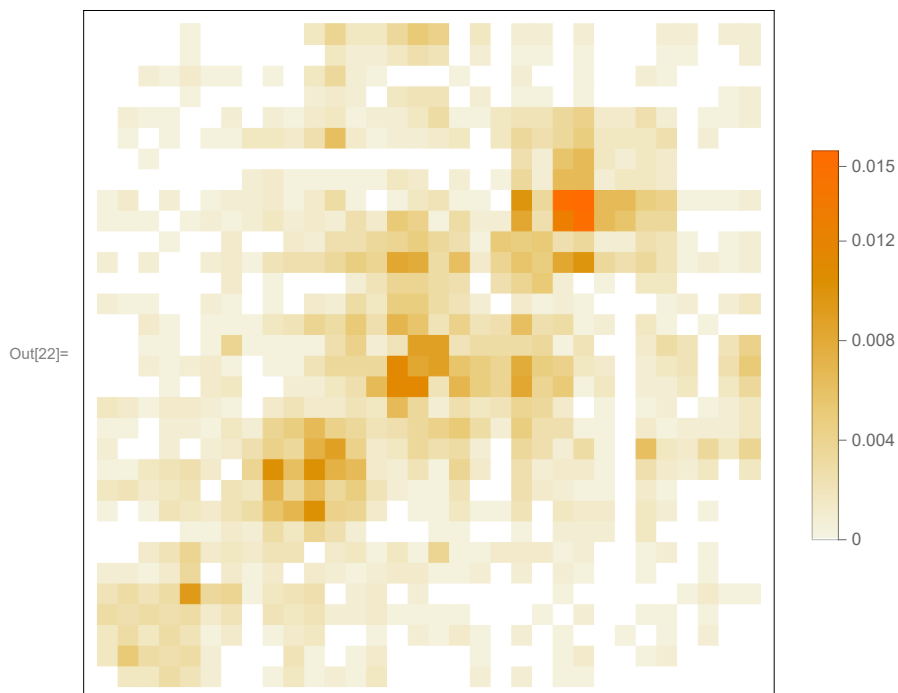
In[21]:= **DegreeCorrelationMatrixPlot[g3, True]**

[истина]



In[22]:= **DegreeCorrelationMatrixPlot[g3, False]**

[ложь]



```
In[23]:= N[AssortativityCoefficient[DegreeCorrelationMatrix[g3]]]
```

численное приближение

```
Out[23]= 0.396777
```

```
In[24]:= N[GraphAssortativity[g3]]
```

ассортативность графа

```
Out[24]= 0.396777
```

2. Функция корреляции степеней

Аппроксимация в двойном логарифмическом масштабе функции средней степени соседа по степеням по заданному графу

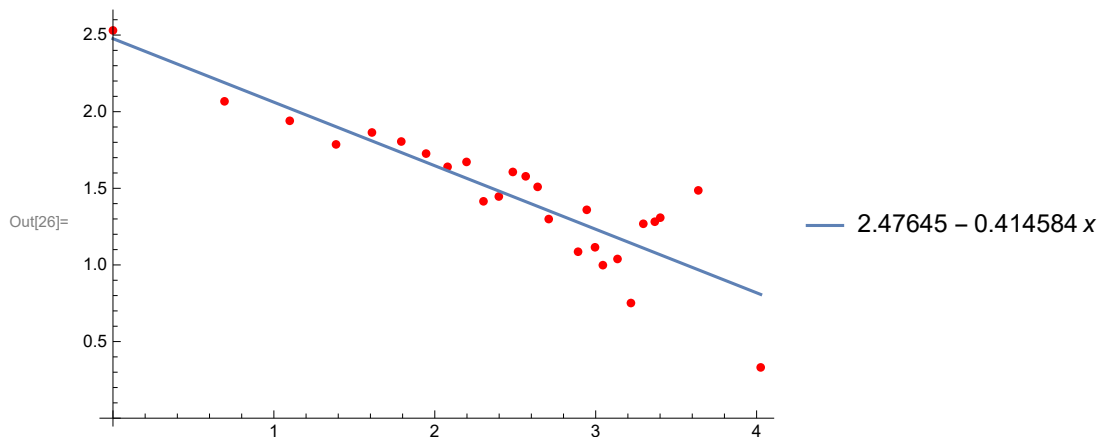
```
In[25]:= MDCLA[graph_] := Module[{data = Log[DeleteCases[{Range[Max[VertexDegree[graph]]],
    программный м... на... удалить случаи... диап... ма... степень вершины
    MeanDegreeConnectivity[graph][[2 ;; -1]]^T, {_, 0}]]}, Show[
    средняя связность по степеням
    ListPlot[data, PlotStyle -> Red], Plot[Evaluate[Normal[LinearModelFit[data, x, x]]],
    диаграмма разб... стиль графика кра... гр... вычислить норма... модель линейной регрессии
    {x, 0, Max[data]}, PlotLegends -> "AllExpressions"]]
```

максимум легенды графика

Для заданных графов

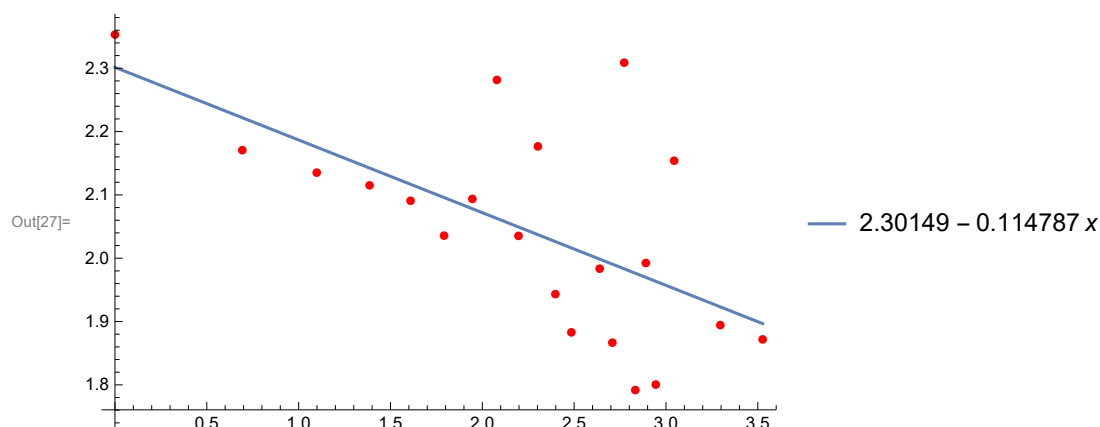
bio-yeast

```
In[26]:= MDCLA[g1]
```



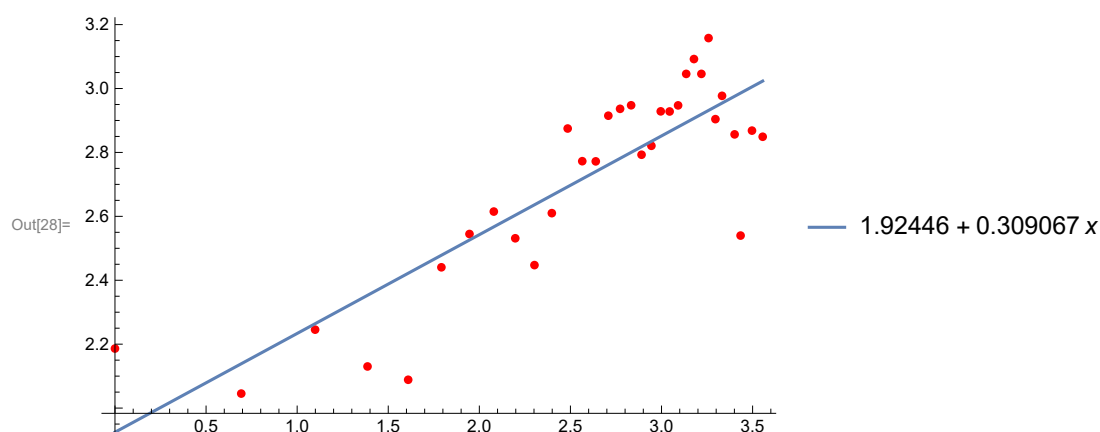
ca-netscience

In[27]:= MDCLA[g2]



proteins

In[28]:= MDCLA[g3]



3. Алгоритм Xalvi-Brunet & Sokolov

Функция которая приводит список ребер к виду, где в каждом ребре

$V_1 \rightarrow V_2, V_1 \leq V_2$

```
In[29]:= EdgeSort[edgelist_] := Module[{out = edgelist},
  (* программный модуль *)
  Do[If[edgelist[[i, 1]] > edgelist[[i, 2]], out[[i]] = edgelist[[i, 2]] → edgelist[[i, 1]],
    (* условный оператор *)
    {i, 1, Length[edgelist]}];
  (* длина *)
  out]
```

Рандомизация графа которая не создает новых петель и мультиребер и приводит к ассортативному/дисассортативному виду

```

In[30]:= GraphRandomizationByAssortativity[g_, n_, assortativity_] :=
  Module[{el = EdgeSort[EdgeList[g]], eltemp = {}, vltemp = {}, temp = {{}}, v = {},
    программный модуль список рёбер
    vd = {}, k = 0}, Do[temp[[1]] = RandomChoice[el] (*Выбираем первое ребро/петлю*);
    оператор цикла случайный выбор
    eltemp = DeleteCases[el, temp[[1]]] (*Создаем временную выборку из ребер/петель
    удалить случаи по образцу
    и удаляем из нее все мультиребра/мультипетли для выбранного ребра/петли*);
    If[temp[[1, 1]] == temp[[1, 2]], (*Алгоритм выбрал петлю*
    условный оператор
    eltemp = DeleteCases[eltemp, v_ -> v_] (*удаляем из временной выборки все петли*);
    удалить случаи по образцу
    vltemp = Flatten[{Cases[eltemp, temp[[1, 1]] -> _], Cases[eltemp, _ -> temp[[1, 1]]]}];
    уплостить случаи по образцу случаи по образцу
    vltemp = DeleteDuplicates[Flatten[{vltemp[[All, 1]], vltemp[[All, 2]]}]]
    удалить дубликаты уплостить все все
    (*Список вершин с которыми граничит выбранная петля*);
    Do[eltemp = DeleteCases[eltemp, vltemp[[j]] -> _];
    оператор цикла удалить случаи по образцу
    eltemp = DeleteCases[eltemp, _ -> vltemp[[j]], {j, 1, Length[vltemp]}] (*Удаляем из
    удалить случаи по образцу длина
    временной выборки все ребра/петли, вершины которых связаны с данной петлей*);
    If[eltemp == {}, i++;
    условный оператор
    Goto[end] (*Алгоритм выбрал петлю которую нельзя изменить,
    перейти
    так как до любой вершины из нее можно добраться в два шага. Переходим в
    конец цикла и считаем что на этом шаге ничего не поменялось*), temp[[2]] =
    RandomChoice[eltemp] (*Выбираем второе ребро*), (*Алгоритм выбрал ребро*
    случайный выбор
    vltemp = Flatten[{Cases[eltemp, temp[[1, 1]] -> _], Cases[eltemp, _ -> temp[[1, 1]]],
    уплостить случаи по образцу случаи по образцу
    Cases[eltemp, temp[[1, 2]] -> _], Cases[eltemp, _ -> temp[[1, 2]]]}];
    случаи по образцу случаи по образцу
    vltemp = DeleteDuplicates[Flatten[{vltemp[[All, 1]], vltemp[[All, 2]]}]]
    удалить дубликаты уплостить все все
    (*Список вершин с которыми граничит выбранное ребро*);
    Do[eltemp = DeleteCases[eltemp, vltemp[[j]] -> _];
    оператор цикла удалить случаи по образцу
    eltemp = DeleteCases[eltemp, _ -> vltemp[[j]], {j, 1, Length[vltemp]}] (*Удаляем из
    удалить случаи по образцу длина
    временной выборки все ребра/петли, вершины которых связаны с данным ребром*);
    If[eltemp == {}, i++;
    условный оператор
    Goto[end] (*Алгоритм выбрал ребро которое нельзя изменить,
    перейти
    так как до любой вершины из него можно добраться в два шага. Переходим
    в конец цикла и считаем что на этом шаге ничего не поменялось*),
    temp[[2]] = RandomChoice[eltemp] (*Выбираем второе ребро/петлю*)];
    случайный выбор
    v = {temp[[1, 1]], temp[[1, 2]], temp[[2, 1]], temp[[2, 2]]} (*Вершины выбранных ребер*);
    vd = {VertexDegree[g, temp[[1, 1]], VertexDegree[g, temp[[1, 2]], VertexDegree[g,
    степень вершины степень вершины степень вершины

```



```

temp[[2, 1]], VertexDegree[g, temp[[2, 2]]] (*Степени вершин выбранных ребер*);
If[Length[DeleteDuplicates[vd]] == 1, Goto[end] (*Все степени вершин у выбранных
ребер совпали, при переключении ассортативность не изменится. Переходим
в конец цикла и считаем что на этом шаге ничего не поменялось*)];
k = 1;
While[(el[[k]] == temp[[1]]) == False, k++];
el = Delete[el, k];
k = 1;
While[(el[[k]] == temp[[2]]) == False, k++];
el = Delete[el, k] (*Удаляем из списка ребер выбранные*);
If[assortativity == True,
temp = ReverseSortBy[{EdgeSort[{v[[1]] -> v[[2]], v[[3]] -> v[[4]]}],
Correlation[{vd[[1]], vd[[2]], vd[[3]], vd[[4]]}, {vd[[2]], vd[[1]], vd[[4]], vd[[3]]}],
{EdgeSort[{v[[1]] -> v[[3]], v[[2]] -> v[[4]]}], Correlation[{vd[[1]], vd[[3]], vd[[2]], vd[[4]]}, {vd[[3]], vd[[1]], vd[[4]], vd[[2]]}], {EdgeSort[{v[[1]] -> v[[4]], v[[2]] -> v[[3]]}], Correlation[{vd[[1]], vd[[4]], vd[[2]], vd[[3]]}, {vd[[4]], vd[[1]], vd[[3]], vd[[2]]}]}],
Last][[1, 1]] (*Выбор переключения дающий наибольшую ассортативность*),
temp = SortBy[{EdgeSort[{v[[1]] -> v[[2]], v[[3]] -> v[[4]]}], Correlation[
{vd[[1]], vd[[2]], vd[[3]], vd[[4]]}, {vd[[2]], vd[[1]], vd[[4]], vd[[3]]}], {EdgeSort[
{v[[1]] -> v[[3]], v[[2]] -> v[[4]]}], Correlation[{vd[[1]], vd[[3]], vd[[2]], vd[[4]]}, {vd[[3]],
vd[[1]], vd[[4]], vd[[2]]}], {EdgeSort[{v[[1]] -> v[[4]], v[[2]] -> v[[3]]}], Correlation[
{vd[[1]], vd[[4]], vd[[2]], vd[[3]]}, {vd[[4]], vd[[1]], vd[[3]], vd[[2]]}]}], Last][[1, 1]]
(*Выбор переключения дающий наибольшую дисассортативность*);
AppendTo[el, temp[[1]]];
AppendTo[el, temp[[2]]] (*Добавляем новые ребра*);
Label[end], {i, 1, n}];
Graph[el]

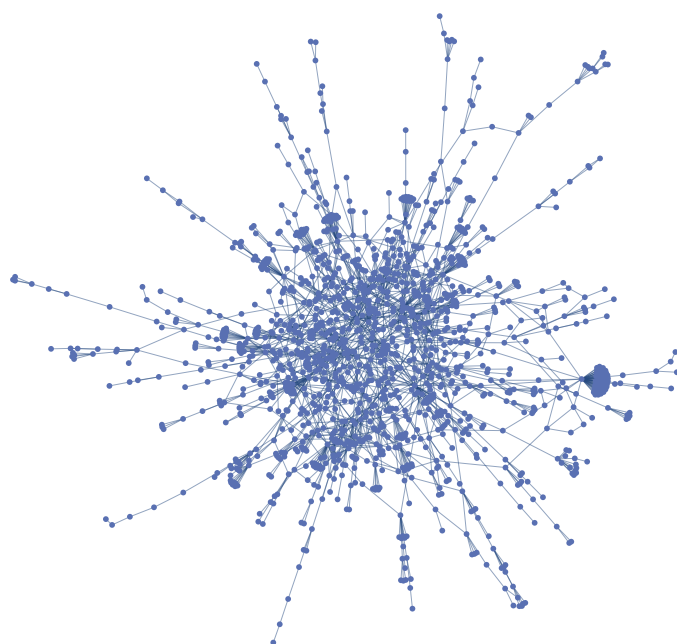
```

Работа алгоритма рандомизации на примере графа bio-yeast

Изначальный граф

In[31]:= **g1**

Out[31]=



In[32]:= **N[GraphAssortativity[g1]]**

... ассортативность графа

Out[32]= -0.209541

Граф после ассортативной рандомизации

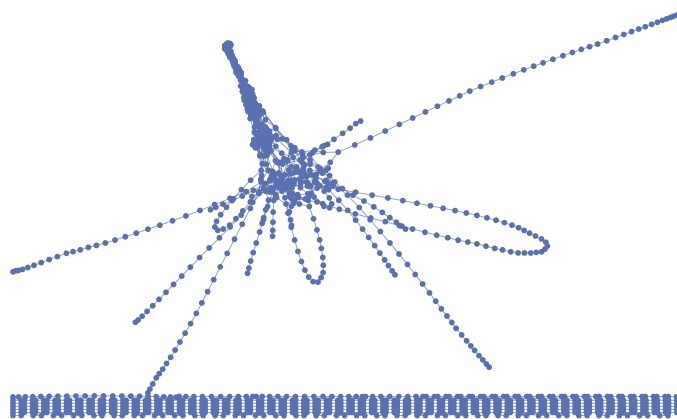
In[33]:= **g1a = Import[NotebookDirectory[] <> "//g1a.m"]**

... импорт ... директория файла блокнота

(*g1a=GraphRandomizationByAssortativity[g1,100000,True]*)

... истина

Out[33]=



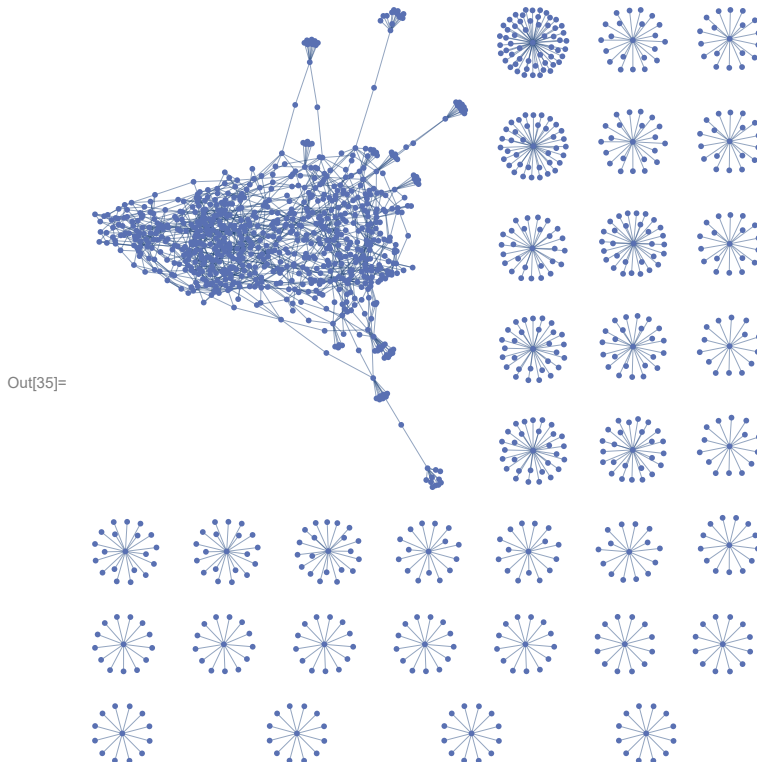
In[34]:= **N[GraphAssortativity[g1a]]**

... ассортативность графа

Out[34]= 0.570559

Граф после дисассортативной рандомизации

```
In[35]:= g1d = Import[NotebookDirectory[] <> "//g1d.m"]
           импорт  директория файла блокнота
           (*g1d=GraphRandomizationByAssortativity[g1,100000,False]*)
           ложь
```



```
In[36]:= N[GraphAssortativity[g1d]]
           .. ассортативность графа
```

Out[36]= -0.386286

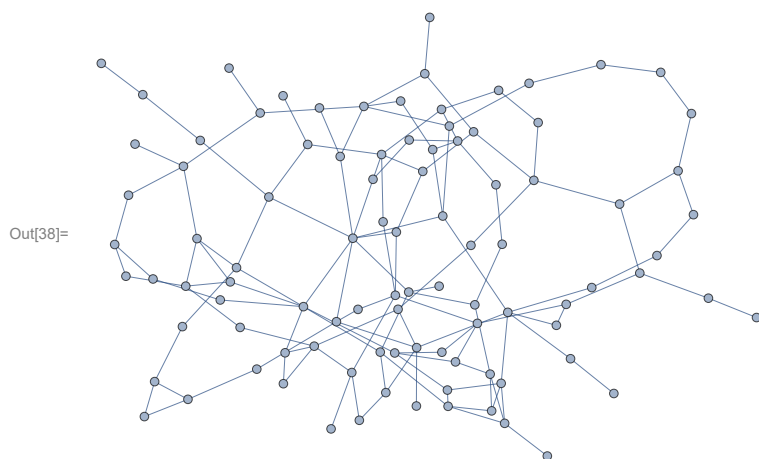
Генерация случайного графа Эрдеша-Реньи по заданному числу вершин и вероятности появления ребра

```
In[37]:= ErdősRenyiGraph[n_, p_] :=
           Graph[Flatten[Table[RandomChoice[{p, 1 - p} -> {i -> j, {}}, {i, 1, n - 1}, {j, i + 1, n}]]]
           граф  упростить табл... случайный выбор
```

Работа алгоритма рандомизации на примере случайного графа

Изначальный граф

```
In[38]:= rg = ErdősRenyiGraph[100,  $\frac{1}{40}$ ]
```



```
In[39]:= N[GraphAssortativity[rg]]
```

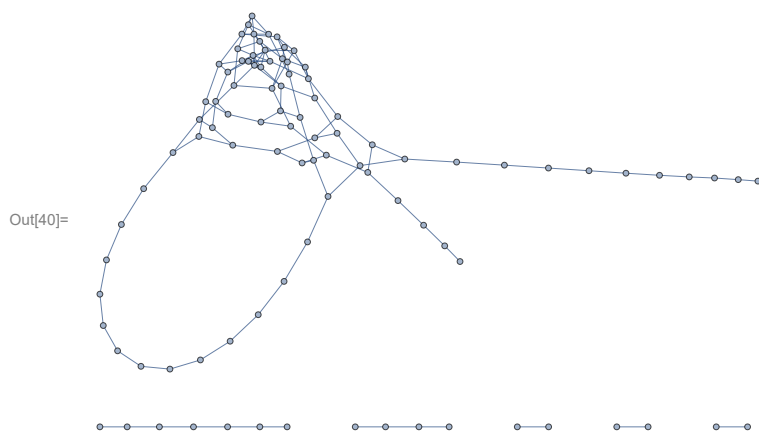
... ассортативность графа

Out[39]= 0.062069

Граф после ассортативной рандомизации

```
In[40]:= rga = GraphRandomizationByAssortativity[rg, 1000, True]
```

... истина



```
In[41]:= N[GraphAssortativity[rga]]
```

... ассортативность графа

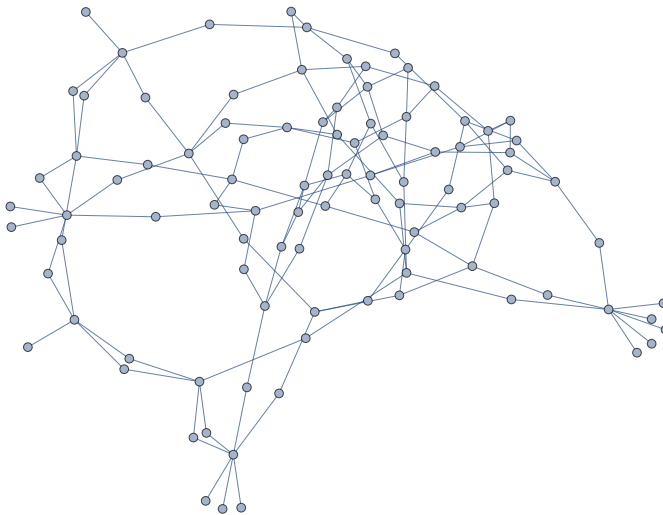
Out[41]= 0.758818

Граф после дисассортативной рандомизации

```
In[42]:= rgd = GraphRandomizationByAssortativity[rg, 1000, False]
```

ложь

Out[42]=



```
In[43]:= N[GraphAssortativity[rgd]]
```

ассортативность графа

Out[43]= -0.798818

Специальная функция. Создает заданное количество графов Эрдеша-Реньи с заданными параметрами. С каждым графом производится ассортативная и дисассортативная рандомизация с заданным числом шагов. На выходе три числа: средний размер главной компоненты у дисассортативных, исходных и ассортативных графов.

```
In[44]:= SF[n_, p_, na_, nr_] :=
Module[{gp = Table[ErdősRenyiGraph[n, p], {i, 1, na}], ga = {}, gd = {}, out = {}},
  {программный... таблица значений
  ga = Table[GraphRandomizationByAssortativity[gp[[i]], nr, True], {i, 1, na}];
  {таблица значений истина
  gd = Table[GraphRandomizationByAssortativity[gp[[i]], nr, False], {i, 1, na}];
  {таблица значений ложь
  out = {Mean[Table[VertexCount[ConnectedGraphComponents[gd[[i]]][1]], {i, 1, na}]],
    {сре... табл... число вершин связанные граф-компоненты
    Mean[Table[VertexCount[ConnectedGraphComponents[gp[[i]]][1]], {i, 1, na}]],
    {сре... табл... число вершин связанные граф-компоненты
    Mean[Table[VertexCount[ConnectedGraphComponents[ga[[i]]][1]], {i, 1, na}]]];
  out]
```

```
In[45]:= sim1000 = Import[NotebookDirectory[] <> "//sim1000.m"];
```

импорт директория файла блокнота

```
(*sim1000=Table[SF[1000,1,100,1000],{1,10-4,5 10-3,10-4}] ;*)
```

таблица значений

```

In[46]:= ListLinePlot[{{Range[10-4, 5 × 10-3, 10-4], sim1000[All, 1]}T,
  {Range[10-4, 5 × 10-3, 10-4], sim1000[All, 2]}T,
  {Range[10-4, 5 × 10-3, 10-4], sim1000[All, 3]}T}, PlotLabel →
  "Число вершин: 1000, размер ансамбля: 100, число шагов рандомизации: 1000",
  AxesLabel → {"Вероятность\добавления\ребра", "Размер главной\пкомпоненты"},
  ImageSize → Large, PlotLegends →
  Placed[{"Размер главной компоненты после дисассортативной рандомизации",
    "Размер главной компоненты до рандомизации",
    "Размер главной компоненты после ассортативной рандомизации"}, Below]

```

