Министерство науки и высшего образования РФ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский Авиационный Институт»
Национальный Исследовательский Университет

Факультет №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

КУРСОВОЙ ПРОЕКТ

По дисциплине «Дискретная математика»

на тему: «Теория графов»

Студент:	Хренникова А. С.
Группа:	М8О-108-19
Преподаватель:	Смерчинская С. О.
Подпись:	
Оценка:	
Дата:	

Москва 2020

Задание 1

Определить для орграфа, заданного матрицей смежности:

a) Матрицу односторонней связности

b) Матрицу сильной связности

c) Компоненты сильной связности

d) Матрицу контуров

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Решение:

a) Найдем матрицу односторонней связности по формуле: $\mathrm{T} = \mathrm{E} \vee A \vee A^2 \vee A^3$

$$A^2 = A \times A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$A^3 = A^2 \times A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\mathrm{T} = E \times A \times A^2 \times A^3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \vee \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \vee \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \vee$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} - \text{матрица односторонней связности}$$

b) Матрица сильной связности: $\overline{S} = T \,\&\, T^T$

$$\overline{S} = T \,\&\, T^T = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \& \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$ – матрица

сильной связности

c) Компоненты сильной связности:

$$\overline{S} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \Rightarrow \overline{S_1} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \Rightarrow v_1 = \{v_1,\ v_2,\ v_3\}, v_2 = \{v_4\}.$$

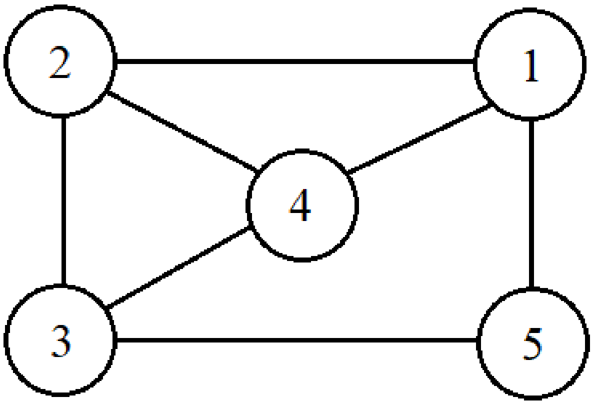
d) Матрица контуров: K = $\overline{S}$ & A

$$\mathrm{K} = \overline{S} \,\&\, \mathrm{A} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \& \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Следовательно, дуги $< v_{1\,,} v_2 >, < v_{1\,,} v_3 >, < v_{2\,,} v_1 >, < v_{3\,,} v_1 >, < v_{3\,,} v_2 >$ принадлежат какому-либо конуру исходного графа.

Задание 2

Используя алгоритм Терри, определить замкнутый маршрут, проходящий ровно по два раза (по одному в каждом направлении) через каждое ребро графа.



Маршрут обхода: 1 -> 2 -> 3 -> 5 -> 1 -> 4 -> 3 -> 2 -> 4 -> 1 -> 5 -> 3 -> 4 -> 2 -> 1

Задание 3

Используя алгоритм «фронта волны», найти все минимальные пути из первой вершины в последнюю орграфа, заданной матрицей смежности.

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Решение:

1. Помечаем вершину v_1 индексом 0. Вершина v_1 принадлежит фронту волны нулевого уровня $W_0(v_1)$.
2. Вершины из множества $Gv_i = GW_0(v_1) = \{v_4, v_6\}$ помечаем индексом 1, они принадлежат фронту волны первого уровня $W_1(v_1)$.
3. Непомеченные ранее вершины из множества $GW_1(v_1) = G\{v_4, v_6\} = \{v_3\}$ помечаем индексом 2, v_3 принадлежит фронту волны второго уровня $W_2(v_1)$.
4. Непомеченные ранее вершины из множества $GW_2(v_1) = G\{v_3\} = \{v_2\}$ помечаем индексом 3, v_2 принадлежит фронту волны второго уровня $W_3(v_1)$.
5. Непомеченные ранее вершины из множества $GW_3(v_1) = G\{v_2\} = \{v_5\}$ помечаем индексом 4, v_5 принадлежит фронту волны второго уровня $W_4(v_1)$.
6. Непомеченные ранее вершины из множества $GW_4(v_1) = G\{v_5\} = \{v_7\}$ помечаем индексом 5, v_7 принадлежит фронту волны второго уровня $W_5(v_1)$.
7. Вершина v_7 достигнута, помечена индексом 5 => длина кротчайшего пути из v_1 в v_7 равна 5.

Промежуточные вершины кратчайших путей находятся согласно приведенным формулам (начиная с последней вершины пути):

1) v_7

2) $W_4(v_1) \cap G^{-1}v_7 = \{v_5\} \cap \{v_5\} = \{v_5\}$

3) $W_3(v_1) \cap G^{-1}v_5 = \{v_2\} \cap \{v_2, v_7\} = \{v_2\}$

4) $W_2(v_1) \cap G^{-1}v_2 = \{v_3\} \cap \{v_3\} = \{v_3\}$

5) $W_1(v_1) \cap G^{-1}v_3 = \{v_4, v_6\} \cap \{v_2, v_4, v_5, v_6, v_7\} = \{v_4, v_6\}$

6) $W_0(v_1) \cap G^{-1}v_4 = \{v_1\} \cap \{v_1, v_2, v_{3,} v_{5,} v_6\} = \{v_1\}$

$W_0(v_1) \cap G^{-1}v_6 = \{v_1\} \cap \{v_1, v_3, v_4, v_5, v_7\} = \{v_1\}$

Кротчайших путей два:

1. $v_1 - v_4 - v_3 - v_2 - v_5 - v_7$
2. $v_1 - v_6 - v_3 - v_2 - v_5 - v_7$

Задание 4

Используя алгоритм Форда, найти минимальные пути из первой вершины во все достижимые вершины в нагруженном графе, заданном матрицей длин дуг.

$$\begin{pmatrix} - & 2 & 13 & - & 4 & - & - & - \\ 2 & - & 10 & - & 1 & - & - & - \\ - & - & - & 4 & - & 3 & - & 6 \\ 5 & - & - & - & - & - & 1 & 3 \\ 6 & 1 & - & - & - & 5 & - & - \\ 3 & - & 3 & - & - & - & 7 & - \\ 8 & - & - & 1 & - & - & - & 5 \\ - & - & - & - & - & 17 & - & - \end{pmatrix}$$

Решение:

1) Составим таблицу итераций:

	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	λ_0	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7
V_1	-	2	13	-	4	-	-	-	**0**	0	0	0	0	0	0	0
V_2	2	-	10	-	1	-	-	-	-	**2**	2	2	2	2	2	2
V_3	-	-	-	4	-	3	-	6	-	13	12	12	**11**	11	11	11
V_4	5	-	-	-	-	-	1	3	-	4	17	17	17	**15**	15	15
V_5	6	1	-	-	-	5	-	-	-	-	**3**	3	3	3	3	3
V_6	3	-	-	-	-	-	7	-	-	-	9	**8**	8	8	8	8
V_7	8	-	1	-	-	-	-	5	-	-	-	18	**15**	15	15	15
V_8	-	-	-	-	-	17	-	-	-	-	19	18	18	**17**	17	17

2) Длины минимальных путей из вершины v_1 во все остальные вершины определены в последнем столбце таблицы.
3) Найдем вершины, входящие в минимальные пути из v_1 во все остальные вершины графа
   - Минимальный путь из v_1 в v_2 : $v_1 - v_2$, его длина равна 2
     $\lambda_1^0 + C_{12} = 0 + 2 = \lambda_2^1$
   - Минимальный путь из v_1 в v_3 : $v_1 - v_2 - v_5 - v_6 - v_3$, его длина равна 11

$\lambda_6^3 + C_{63} = 8 + 3 = \lambda_3^4$

$\lambda_5^2 + C_{56} = 3 + 5 = \lambda_6^3$

$\lambda_2^1 + C_{25} = 2 + 1 = \lambda_5^2$

$\lambda_1^0 + C_{12} = 0 + 2 = \lambda_2^1$

- Минимальный путь из v_1 в $v_4 : v_1 - v_2 - v_5 - v_6 - v_3 - v_4$, его длина равна 15

  $\lambda_3^4 + C_{34} = 11 + 4 = \lambda_4^5$

- Минимальный путь из v_1 в $v_5 : v_1 - v_2 - v_5$, его длина равна 3

  $\lambda_2^1 + C_{25} = 2 + 1 = \lambda_5^2$

- Минимальный путь из v_1 в $v_6 : v_1 - v_2 - v_5 - v_6$, его длина равна 8

  $\lambda_5^2 + C_{56} = 3 + 5 = \lambda_6^3$

- Минимальный путь из v_1 в $v_7 : v_1 - v_2 - v_5 - v_6 - v_7$, его длина равна 15

  $\lambda_6^3 + C_{67} = 8 + 7 = \lambda_7^4$

- Минимальный путь из v_1 в $v_8 : v_1 - v_2 - v_5 - v_6 - v_3 - v_8$, его длина равна 17

  $\lambda_3^4 + C_{38} = 11 + 6 = \lambda_8^5$

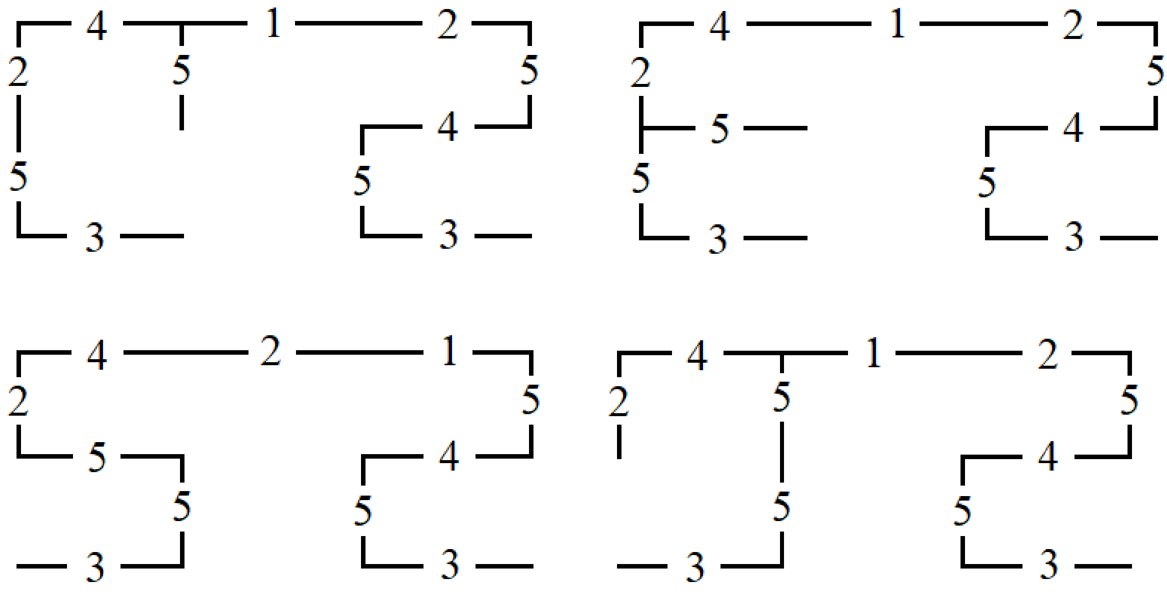
Задание 5

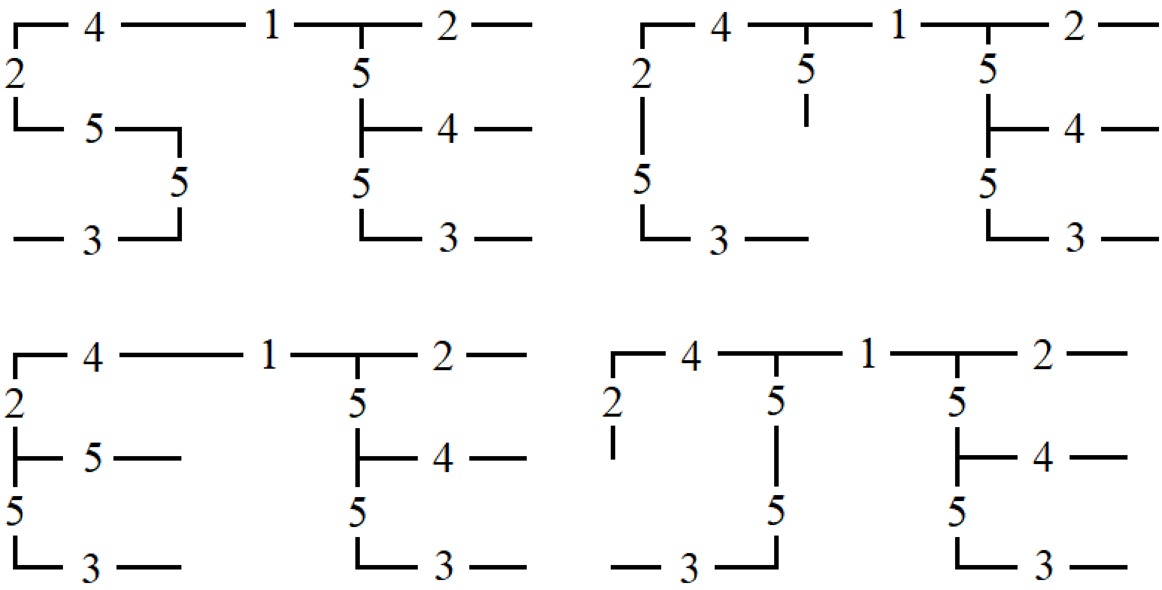
Найдите остовное дерево с минимальной суммой длин входящих в него ребер.

$x_1=5$, $x_2=2$, $x_3=4$, $x_4=5$, $x_5=3$, $x_6=7$, $x_7=6$, $x_8=1$, $x_9=2$, $x_{10}=4$, $x_{11}=3$, $x_{12}=5$, $x_{13}=5$, $x_{14}=5$, $x_{15}=5$, $x_{16}=5$, $x_{17}=5$.

Решение:

1) Выбираем все вершины графа
2) Добавляем все дуги, имеющие минимальный вес – 1. Циклов нет.
3) Добавляем все дуги, имеющие минимальный вес среди оставшихся – 2. Циклов нет.
4) Добавляем все дуги, имеющие минимальный вес среди оставшихся – 3. Циклов нет.
5) Добавляем все дуги, имеющие минимальный вес среди оставшихся – 4. Циклов нет.
6) Добавляем все дуги, имеющие минимальный вес – 5, так, чтобы не было циклов. Получаем восемь возможных вариантов остовных деревьев минимального веса. Минимальный вес остовного дерева L(D)=39.

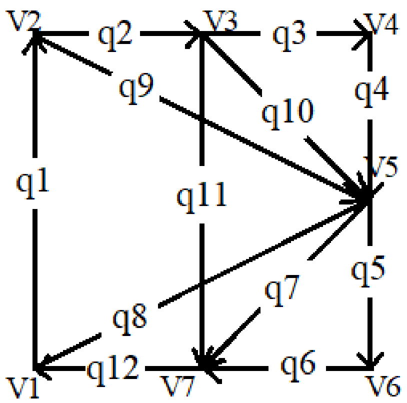




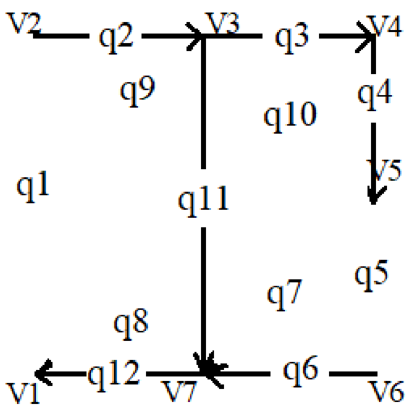
Задание 6

Пусть каждому ребру неориентированного графа соответствует некоторый элемент электрической цепи. Составить линейно независимые системы уравнений Кирхгофа для токов и напряжений. Пусть первому и пятому ребру соответствуют источники тока с ЭДС Ɛ1 и Ɛ2 (полярность выбирается произвольно), а остальные элементы являются сопротивлениями. Используя закон Ома, и, предполагая внутренние сопротивления источника тока равными нулю, получить систему уравнений для токов.

1) Зададим на графе произвольную ориентацию:



2) Построим произвольное остовное дерево D заданного графа:



3) Найдем базис циклов, добавляя к остовному дереву по одному не вошедшему в него ребру. Затем найдем соответствующие вектор-циклы.

$$(D + q_1)\colon \mu_1 : V_1 - V_2 - V_3 - V_4 - V_5 \Rightarrow C(\mu_1) = (1\;1\;0\;0\;0\;0\;0\;0\;0\;0\;1\;1)$$

$$(D + q_5)\colon \mu_2 : V_5 - V_6 - V_7 - V_3 - V_4 - V_5 \Rightarrow C(\mu_2)$$

$$= (0\;0\;1\;1\;1\;1\;0\;0\;0\;0\;\; -1\;0)$$

$(D+q_7)$: $\mu_3 : V_5 - V_7 - V_3 - V_4 - V_5 \Rightarrow C(\mu_3)$
$= (0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ \ -1\ 0)$

$(D+q_8)$: $\mu_4 : V_1 - V_5 - V_4 - V_3 - V_7 - V_1 \Rightarrow C(\mu_4)$
$= (0\ 0\ \ -1\ \ -1\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 1)$

$(D+q_9)$: $\mu_5 : V_5 - V_2 - V_3 - V_4 - V_5 \Rightarrow C(\mu_5) = (0\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0)$

$(D+q_{10})$: $\mu_6 : V_3 - V_5 - V_4 - V_3 \Rightarrow C(\mu_6)$
$= (0\ 0\ \ -1\ \ -1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0)$

4) Циклическая матрица имеет вид:

$$C = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

5) Выпишем закон Кирхгофа для напряжений:

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} * \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \\ u_{10} \\ u_{11} \\ u_{12} \end{pmatrix} = 0$$

Напряжения, соответствующие ребрам, не вошедшим в остовное дерево – базисные переменные системы.

$u_1 + u_2 + u_{11} + u_{12} = 0$

$u_3 + u_4 + u_5 + u_6 - u_{11} = 0$

$u_3 + u_4 + u_7 - u_{11} = 0$

$-u_3 - u_4 + u_8 + u_{11} + u_{12}$
$= 0$

$u_2 + u_3 + u_4 + u_9 = 0$

$-u_3 - u_4 + u_{10} = 0$

$u_1 = -u_2 - u_{11} - u_{12}$

$u_5 = u_{11} - u_3 - u_4 - u_6$

$u_3 = u_{11} - u_4 - u_7$

$u_3 = u_8 + u_{11} + u_{12} - u_4$ $\quad\quad$ $u_3 = u_{10} - u_4$

$u_2 = -u_3 - u_4 - u_9$

6) Выпишем закон Кирхгофа для токов: $B * I = 0$

7) Выпишем уравнения Кирхгофа для токов.

Найдем матрицу инцидентности B орграфа:

	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}
V_1	-1	0	0	0	0	0	0	-1	0	0	0	1
V_2	1	-1	0	0	0	0	0	0	1	0	0	0
V_3	0	1	-1	0	0	0	0	0	0	-1	-1	0
V_4	0	0	1	-1	0	0	0	0	0	0	0	0
V_5	0	0	0	1	-1	0	-1	1	-1	1	0	0
V_6	0	0	0	0	1	-1	0	0	0	0	0	0
V_7	0	0	0	0	0	1	1	0	0	0	1	-1

$$B = \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & -1 & 1 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & -1 \end{pmatrix}$$

$$\begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & -1 & 1 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \\ I_5 \\ I_6 \\ I_7 \\ I_8 \\ I_9 \\ I_{10} \\ I_{11} \\ I_{12} \end{pmatrix} = 0$$

$-I_1 - I_8 + I_{12} = 0$ $\quad\quad$ $I_2 - I_3 - I_{10} - I_{11} = 0$

$I_1 - I_2 + I_9 = 0$ $\quad\quad$ $I_4 - I_5 - I_7 + I_8 - I_9 + I_{10} = 0$

$I_5 - I_6 = 0$

$I_6 + I_7 + I_{11} - I_{12} = 0$

$-I_1 - I_8 + I_{12} = 0$

$I_1 - I_2 + I_9 = 0$

$I_2 - I_3 - I_{10} - I_{11} = 0$

$I_3 - I_4 = 0$

$I_5 - I_6 = 0$

$I_6 + I_7 + I_{11} - I_{12} = 0$

8) Подставим закон Ома: $U = I * R$

$\mathcal{E}_1 = -I_2R_2 - I_{11}R_{11} - I_{12}R_{12}$

$\mathcal{E}_2 = -I_{11}R_{11} - I_3R_3 - I_4R_4 - I_6R_6$

$0 = I_3R_3 + I_4R_4 + I_7R_7 - I_{11}R_{11}$

$0 = I_3R_3 + I_4R_4 - I_8R_8 - I_{11}R_{11} - I_{12}R_{12}$

$0 = I_2R_2 + I_3R_3 + I_4R_4 + I_9R_9$

$0 = I_3R_3 + I_4R_4 - I_{10}R_{10}$

9) Совместная система имеет вид:

$-I_1 - I_8 + I_{12} = 0$

$I_1 - I_2 + I_9 = 0$

$I_2 - I_3 - I_{10} - I_{11} = 0$

$I_3 - I_4 = 0$

$I_5 - I_6 = 0$

$I_6 + I_7 + I_{11} - I_{12} = 0$

$\mathcal{E}_1 = -I_2R_2 - I_{11}R_{11} - I_{12}R_{12}$

$\mathcal{E}_2 = -I_{11}R_{11} - I_3R_3 - I_4R_4 - I_6R_6$

$0 = I_3R_3 + I_4R_4 + I_7R_7 - I_{11}R_{11}$

$0 = I_3R_3 + I_4R_4 - I_8R_8 - I_{11}R_{11} - I_{12}R_{12}$

$0 = I_2R_2 + I_3R_3 + I_4R_4 + I_9R_9$

$0 = I_3R_3 + I_4R_4 - I_{10}R_{10}$

Двенадцать уравнений и двенадцать неизвестных – токи $I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8, I_9, I_{10}, I_{11}, I_{12}$, ЭДС $\mathcal{E}_1, \mathcal{E}_2$ и сопротивления $R_2, R_3, R_4, R_6, R_7, R_8, R_9, R_{10}, R_{11}, R_{12}$ известны.

Задание 7

Построить максимальный поток по транспортной сети.

a=5, b=5, c=6, d=8, e=6, f=10, g=6.

Решение:

1) Построение полного потока

   Ищем пути из источника в сток, не содержащие насыщенных дуг.

   1. $V_1 - V_2 - V_3 - V_4 - V_9$: $min\{5, 5, 8, 14\} = 5$
   2. $V_1 - V_6 - V_7 - V_8 - V_9$: $min\{5, 5, 10, 17\} = 5$
   3. $V_1 - V_5 - V_9$: $min\{7, 6\} = 6$
   4. $V_1 - V_3 - V_4 - V_9$: $min\{9, 8 - 5, 14 - 5\} = 3$
   5. $V_1 - V_7 - V_8 - V_9$: $min\{10, 10 - 5, 17 - 5\} = 5$
   6. $V_1 - V_5 - V_8 - V_9$: $min\{7 - 6, 6, 17 - 10\} = 1$

   Величина полного потока: $\Phi_{\text{полн.}} = 5 + 5 + 6 + 3 + 5 + 1 = 25$

2) Построение максимального потока:

   Найдем увеличивающие цепи:

   1. $V_1 - V_3 - V_2 - V_5 - V_4 - V_9$

      $\triangle_1 = \min\{9 - 3, 5, 7, 14 - 8\} = 5$

   2. $V_1 - V_7 - V_6 - V_5 - V_8 - V_9$

      $\triangle_2 = \min\{10 - 5, 5, 7, 6, 17 - 11\} = 5$

   Величина потока увеличилась на 5+5=10.

   Величина максимального потока: $\Phi_{\text{макс.}} = \Phi_{\text{полн.}} + \triangle_1 + \triangle_2 =$ $25 + 5 + 5 = 35$

Задание 8

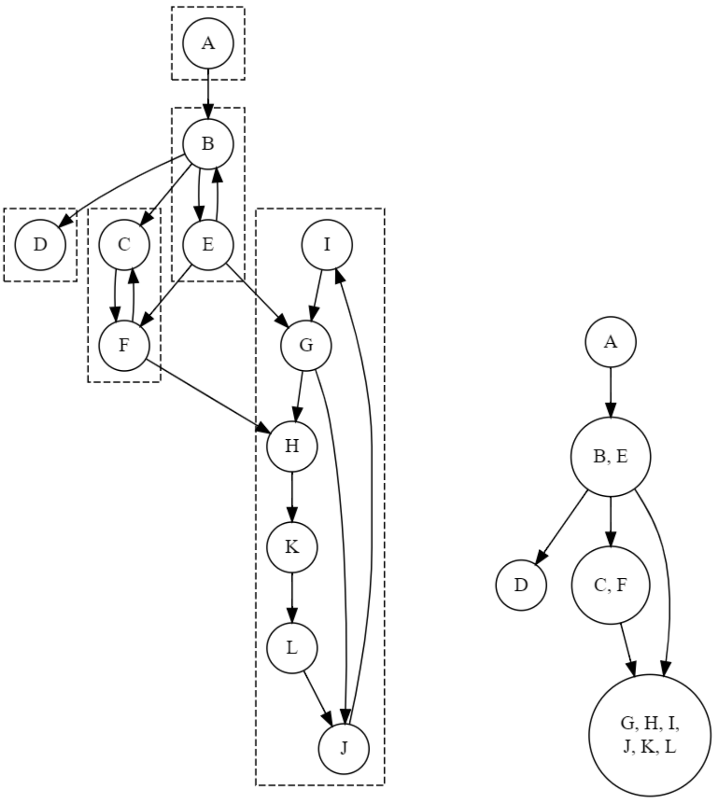
<u>Граф конденсации для графа, заданного матрицей смежности</u>

Основные понятия и определения, теоретическое описание алгоритма.

Связность для ориентированных графов.

Что означает связность для ориентированного графа? Тут надо быть аккуратным. Несвязный неориентированный граф можно разбить на связные компоненты, не соединенные друг с другом. С *ориентированными* графами ситуация сложнее. Граф на рис. 1 нельзя разбить на две части, не соединенные друг с другом. Но мы не будем называть этот граф связным, поскольку в нем нет пути из G в B или из F в A. Дадим такое определение: Вершины u и v ориентированного графа называются *связанными (connected),* если в нем есть путь из u в v, а также путь из v в u.

Такое отношение на вершинах разбивает все множество вершин на непересекающиеся подмножества связанных вершин, называемые *компонентами сильной связности (strongly connected components)*. Граф на рис. 1 имеет пять таких компонент.



а. б.

Рис 1. (а) Ориентированный граф и его компоненты сильной связности. (б) Конденсация ориентированного графа.

Стянем теперь каждую компоненту сильной связности в отдельную вершину *(метавершину)* и оставим только ребра между метавершинами (см. рис. 1), удалив дубликаты. Полученный граф называется *метаграфом(meta-graph)*(также *графом копонент* или *конденсацией*) исходного. Он не содержит циклов: если бы несколько компонент образовывали цикл, то вершины этих компонент были бы в исходном графе достижимы друг из друга и вошли бы в одну компоненту.

Свойство 0. *Граф конденсации любого ориентированного графа является ациклическим (и может быть топологически упорядочен)*.

Но как построить конденсацию для данного графа? Компоненты сильной связности и метаграф могут быть построены за линейное время. Начнем с такого замечания, которое уже было доказано в предыдущих лекциях.

Свойство 1. *Процедура Explore, вызванная для вершины* u*, заканчивает работу, когда посещены все вершины, достижимые из* u.

Значит, если вызвать Explore для вершины, которая лежит в *компоненте-стоке*, то мы обойдем как раз все вершины этой компоненты. Например, граф рис. 1 имеет две такие компоненты, и вызов Explore для вершины K обойдет большую из них.

Остается понять, (а) как найти вершину, которая гарантированно лежит в компоненте-стоке, и (б) что делать после нахождения компоненты-стока.

Для начала ответим на первый вопрос. Сначала покажем, как решать симметричную задачу: найти вершину в *компоненте-истоке.*

Свойство 2. *Вершина, которой поиск в глубину присваивает максимальное* tout*-значение, лежит в компоненте-истоке*.

Свойство 3. *Пусть* C *и* C′ *— компоненты сильной связности графа и в графе есть ориентированное ребро из* C *в* C′*. Тогда максимальное* tout*-значение вершин в* CC *больше, чем максимальное* tout*-значение вершин в* C′.

Доказательство. Дождемся момента, когда при поиске в глубину впервые появится вершина v из C или C′. Если v∈C, то вызов Explore(v) не завершится, пока не будут обработаны все вершины обеих компонент (по свойству 1). Поэтому tout[v] будет больше, чем у всех вершин из C′. Если же v∈C′, то вызов Explore(v) обойдет все вершины C′, но до C дело еще не дойдет — и максимальное touttout-значение в C тоже будет больше.
Следствием этого является следующее утверждение: *компоненты сильной связности можно топологически упорядочить, расположив их по убыванию максимальных* touttout-*значений вершин в них.* Это наблюдение обобщает рассмотренный нами алгоритм топологической сортировки ориентированных ациклических графов (в таких графах каждая компонента сильной связности состоит из одной вершины).
Таким образом, мы научились находить компоненту-исток и вершину в ней. Как найти все вершины только данной компоненты?
Рассмотрим обращенный граф GR, получаемый из G изменением направлений всех ребер (см. рис. 2). В GR будут те же компоненты сильной связности, как и у G(покажите). Также, найденная вершина *компоненты-истока* G будет вершиной *компоненты-стока* графа GR. Тогда вызов Explore из найденной вершины на графе GR обойдет только вершины компоненты истока графа G.
Теперь нам необходимо удалить все пройденные вершины из графа и найти следующую вершину с максимальным значением tout, после чего запустить Explore в обращенном графе из нее. Повторяя эту процедуру, найдем все компоненты сильной связности в порядке их топологической сортировки.
Заметим, что явно удалять вершины не нужно. Достаточно просто обойти вершины в порядке убывания tout и вызвать Explore из каждой непосещенной.
Итак, мы построили следующий алгоритм выделения компонент сильной связности:

Запустить обход в глубину графа GG, который вернет вершины в порядке убывания tout. Заметим, что для этого достаточно добавлять вершину в массив при выходе Explore из этой вершины.
Построить обращенный граф GR. Запустить процедуру Explore из каждой непосещенной вершины в порядке убывания tout-значений. Каждое множество вершин, достигнутое в результате очередного запуска обхода, и будет очередной компонентой сильной связности.
Данный алгоритм не просто находит все компоненты связности, но и строит ациклический граф конденсации в порядке топологической сортировки компонент.

Описание разработанной программы.

Алгоритм Тарьяна:

Основой для алгоритма является структура данных "Система непересекающихся множеств", которая и была изобретена Тарьяном (Tarjan).
Алгоритм фактически представляет собой обход в глубину из корня дерева, в процессе которого постепенно находятся ответы на запросы. А именно, ответ на запрос (v, u)находится, когда обход в глубину находится в вершине u, а вершина vуже была посещена, или наоборот.
Итак, пусть обход в глубину находится в вершине v(и уже были выполнены переходы в её сыновей), и оказалось, что для какого-то запроса (v, u)вершина uуже была посещена обходом в глубину. Научимся тогда находить LCAэтих двух вершин.
Заметим, что $\mathrm{LCA}(v, u)$является либо самой вершиной v, либо одним из её предков. Получается, нам надо найти самую нижнюю вершину среди предков v(включая её саму), для которой вершина u является потомком.
Заметим, что при фиксированном vпо такому признаку (т.е. какой наименьший предок vявляется и предком какой-то вершины) вершины дерева дерева распадаются на совокупность непересекающихся классов. Для

каждого предка $p \neq v$ вершины v её класс содержит саму эту вершину, а также все поддеревья с корнями в тех её сыновьях, которые лежат "слева" от пути до v(т.е. которые были обработаны ранее, чем была достигнута v).

Нам надо научиться эффективно поддерживать все эти классы, для чего мы и применим структуру данных "Система непересекающихся множеств".

Каждому классу будет соответствовать в этой структуре множество, причём для представителя этого множества мы определим величину $\mathrm{ANCESTOR}$— ту вершину p, которая и образует этот класс.

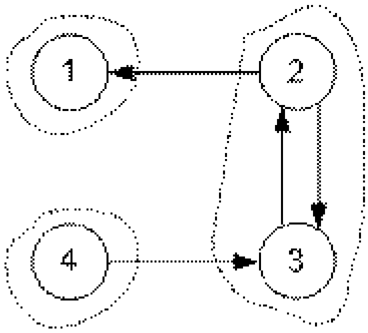
Рассмотрим подробно реализацию обхода в глубину. Пусть мы стоим в некоторой вершине v. Поместим её в отдельный класс в структуре непересекающихся множеств, $\mathrm{ANCESTOR}[v] = v$. Как обычно в обходе в глубину, перебираем все исходящие рёбра (v, to). Для каждого такого to мы сначала должны вызвать обход в глубину из этой вершины, а потом добавить эту вершину со всем её поддеревом в класс вершины v. Это реализуется операцией Union структуры данных "система непересекающихся множеств", с последующей установкой $\mathrm{ANCESTOR} = v$ для представителя множества (т.к. после объединения представитель класса мог измениться). Наконец, после обработки всех рёбер мы перебираем все запросы вида (v, u), и если u была помечена как посещённая обходом в глубину, то ответом на этот запрос будет вершина $\mathrm{LCA}(v, u) = \mathrm{ANCESTOR}[\mathrm{FindSet}(u)]$. Нетрудно заметить, что для каждого запроса это условие (что одна вершина запроса является текущей, а другая была посещена ранее) выполнится ровно один раз.

Оценим **асимптотику**. Она складывается из нескольких частей. Во-первых, это асимптотика обхода в глубину, которая в данном случае составляет $O(n)$. Во-вторых, это операции по объединению множеств, которые в сумме для всех разумных n затрачивают $O(n)$ операций. В-третьих, это для каждого запроса проверка условия (два раза на запрос) и определение результата (один раз на запрос), каждое, опять же, для всех разумных n выполняется за

$O(1)$. Итоговая асимптотика получается $O(n + m)$, что означает для достаточно больших m($n = O(m)$) ответ за $O(1)$на один запрос.

Пример

Граф, приведенный в примере, имеет следующий вид:



Конденсация графа состоит из трех вершин и двух ребер.

Реализация алгоритма

```
// Программа на C ++ для поиска сильно связанных компонентов в данном
// направленный граф по алгоритму Тарьяна (одиночная DFS)

// Класс, представляющий ориентированный граф
class Graph {
    int V; // Количество вершин
    std::list<int>* adj; // Динамический массив списков смежности

    // Рекурсивная DFS-функция, используемая SCC ()
    void SCCUtil(int u, int disc[], int low[], std::stack<int>* st, bool stackMember[]);
public:
    Graph(int V); // Конструктор
    void addEdge(int v, int w); // функция для добавления ребра на график
```

```
        void SCC(); // печатает сильно связанные компоненты
};

Graph::Graph(int V) {
        this->V = V;
        adj = new std::list<int>[V];
}

void Graph::addEdge(int v, int w) {
        adj[v].push_back(w);
}

// Рекурсивная функция, которая находит и печатает сильно связанные
// компоненты, использующие обход DFS
// u -> вершина, которую нужно посетить затем
// disc [] -> Хранит время обнаружения посещенных вершин
// low [] - » самая ранняя посещенная вершина (вершина с минимальным
// время обнаружения), которое может быть достигнуто из поддерева
// корень с текущей вершиной
// * st - » Для хранения всех подключенных предков (может быть частью
// ГТК)
// stackMember [] -> бит / индексный массив для быстрой проверки
// узел находится в стеке
void Graph::SCCUtil(int u, int disc[], int low[], std::stack<int>* st, bool
stackMember[]) {
        // Статическая переменная используется для простоты, мы можем
избежать использования
        // статической переменной путем передачи указателя.

        static int time = 0;
```

```
// Инициализация времени обнаружения и низкого значения
disc[u] = low[u] = ++time;
st->push(u);
stackMember[u] = true;

// Пройти через все вершины, смежные с этим
std::list<int>::iterator i;
for (i = adj[u].begin(); i != adj[u].end(); ++i) {
	int v = *i; // v текущее смежное с 'u'

	// Если v еще не посещено, то повторить его
	if (disc[v] == -1) {
		SCCUtil(v, disc, low, st, stackMember);

		// Проверяем, имеет ли поддерево с корнем 'v'
		// соединение с одним из предков 'u'
		// Случай 1 (в приведенном выше обсуждении Disc и
Low value)
		low[u] = min(low[u], low[v]);
	}

	// Обновляем низкое значение 'u', только v остается в стеке
	// (т.е. это задний край, а не поперечный край).
	// Случай 2 (в приведенном выше обсуждении Disc и Low
value)
	else if (stackMember[v] == true)
		low[u] = min(low[u], disc[v]);
}
```

```
        // найден головной узел, вытолкнуть стек и распечатать SCC
        int w = 0; // Для хранения извлеченных в стеке вершин
        if (low[u] == disc[u]) {
                while (st->top() != u) {
                        w = (int)st->top();
                        std::cout << 'X' << w + 1 << " ";
                        stackMember[w] = false;
                        st->pop();
                }
                w = (int)st->top();
                std::cout << 'X' << w + 1 << "\n";
                stackMember[w] = false;
                st->pop();
        }

}

// Функция для обхода DFS. Он использует SCCUtil ()
void Graph::SCC() {
        int* disc = new int[V];
        int* low = new int[V];
        bool* stackMember = new bool[V];
        std::stack<int>* st = new std::stack<int>();

        // Инициализируем дисковые и низкие массивы и массивы stackMember
        for (int i = 0; i < V; i++) {
                disc[i] = NIL;
                low[i] = NIL;
                stackMember[i] = false;
```

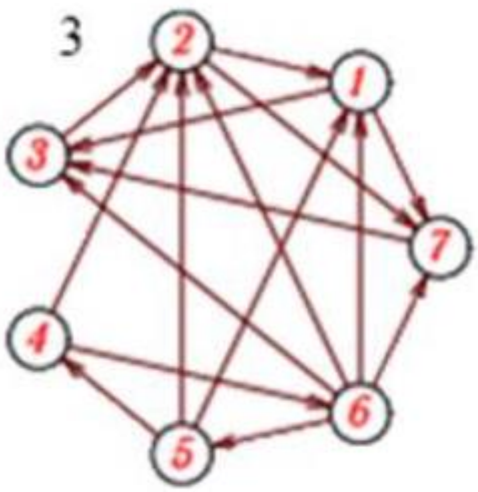
```
        }

        // Вызываем рекурсивную вспомогательную функцию, чтобы
найти сильно
        // связанные компоненты в дереве DFS с вершиной 'i'
        for (int i = 0; i < V; i++)
            if (disc[i] == NIL)
                SCCUtil(i, disc, low, st, stackMember);
    }
```

Сложность алгоритма

Вышеупомянутый алгоритм в основном вызывает DFS, DFS принимает O (V + E) для графа, представленного с использованием списка смежности.

Тестовый пример с решением

Орграф из 7 вершин и 16 рёбер:



Матрица смежности:

0	0	1	0	0	0	1
1	0	0	0	0	0	1
0	1	0	0	0	0	0
0	1	0	0	0	1	0
1	1	0	1	0	0	0
1	1	1	0	1	0	1
0	0	1	0	0	0	0

Найдем матрицу достижимости вершин орграфа:

$$R := \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Построим матрицу контрдостижимости:

$$Q := R^{T} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

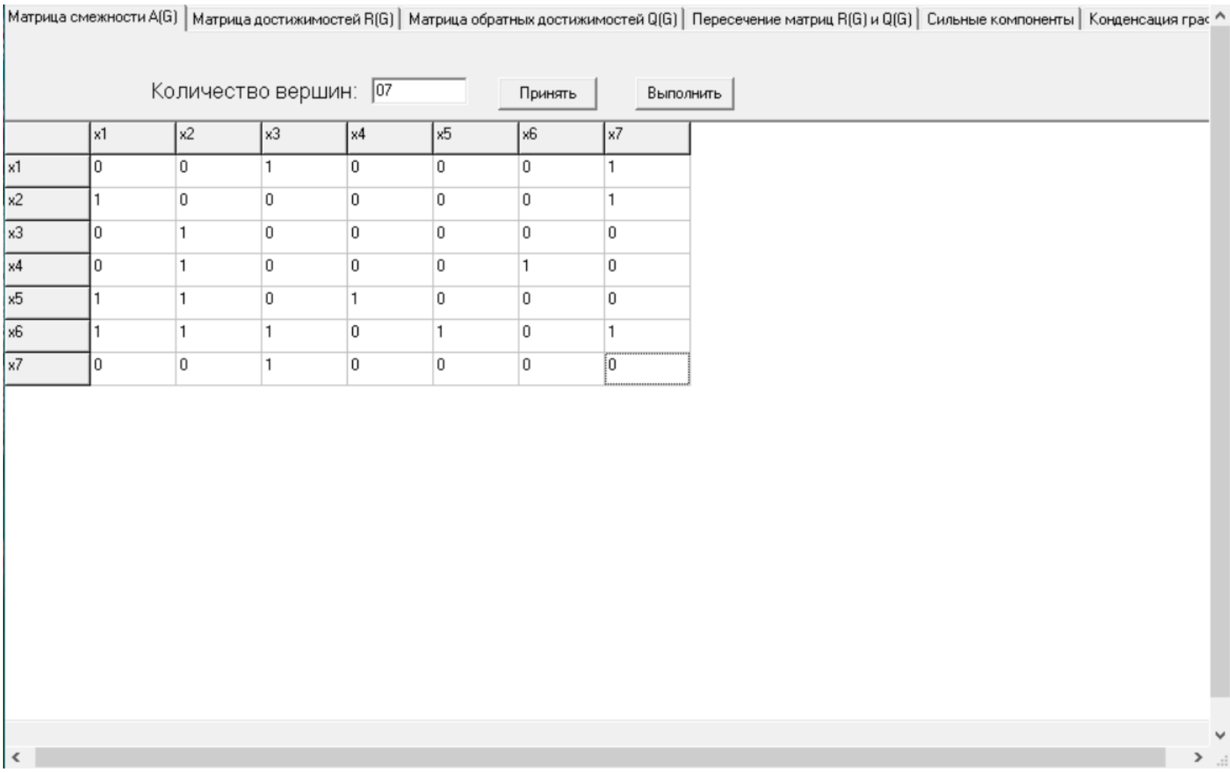
Отсюда получаем матрицу сильных компонент связности орграфа:

$$S := \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Таким образом, данный орграф содержит сильные компоненты связности $G1 = (1,2,3,7)$, G2 = (4,5,6).

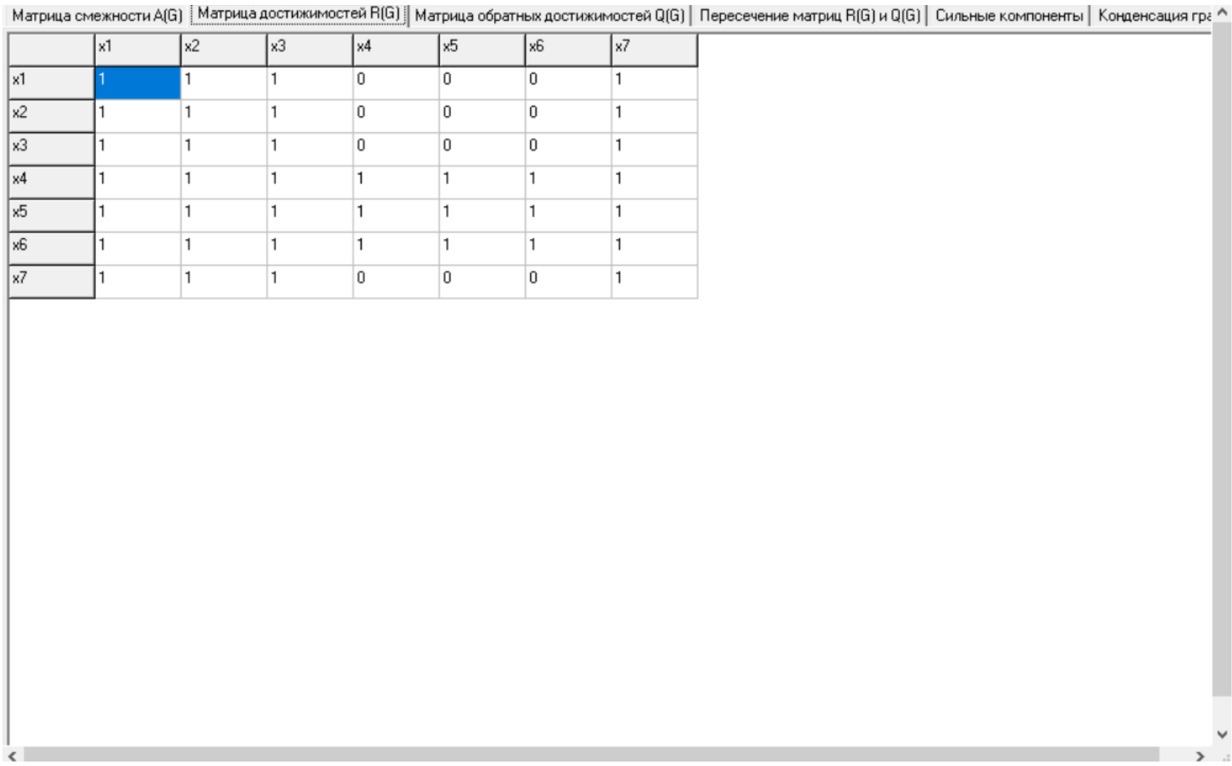
Скриншоты программы

Программа для примера из предыдущего пункта.



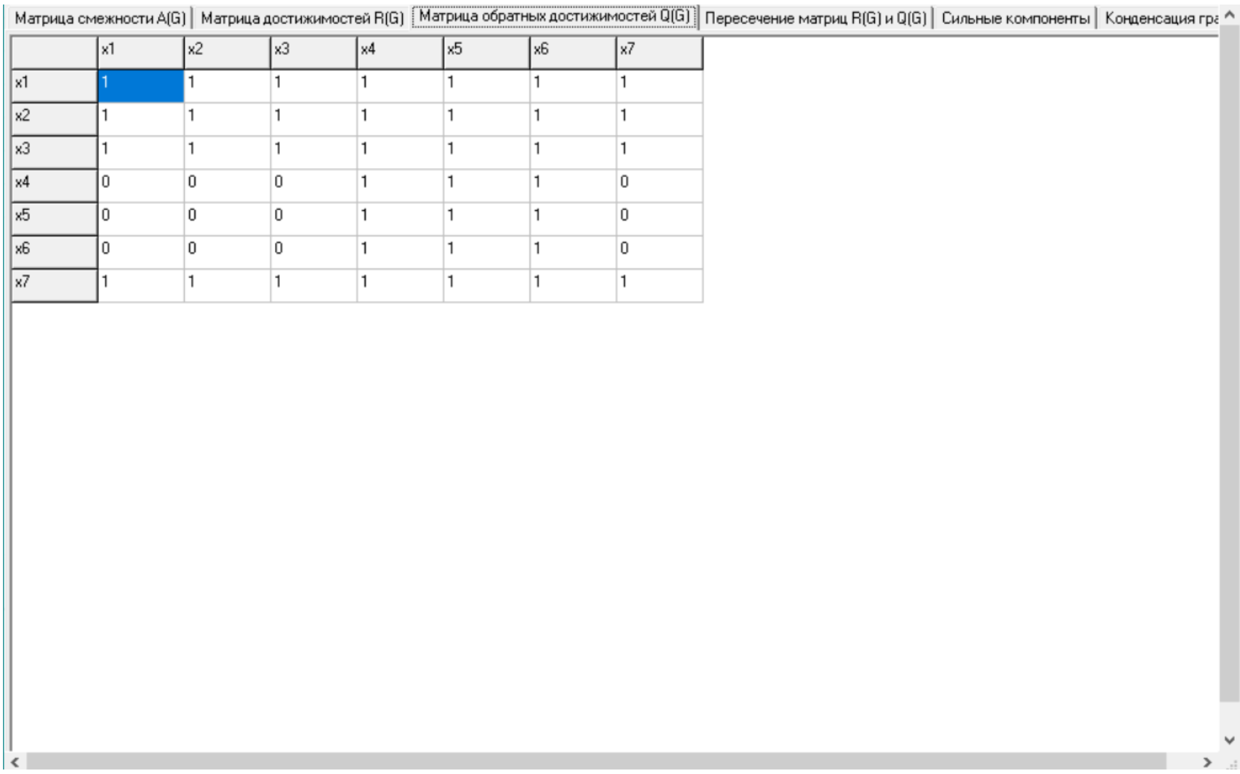
	x1	x2	x3	x4	x5	x6	x7
x1	0	0	1	0	0	0	1
x2	1	0	0	0	0	0	1
x3	0	1	0	0	0	0	0
x4	0	1	0	0	0	1	0
x5	1	1	0	1	0	0	0
x6	1	1	1	0	1	0	1
x7	0	0	1	0	0	0	0

Рис.2-матрица смежности



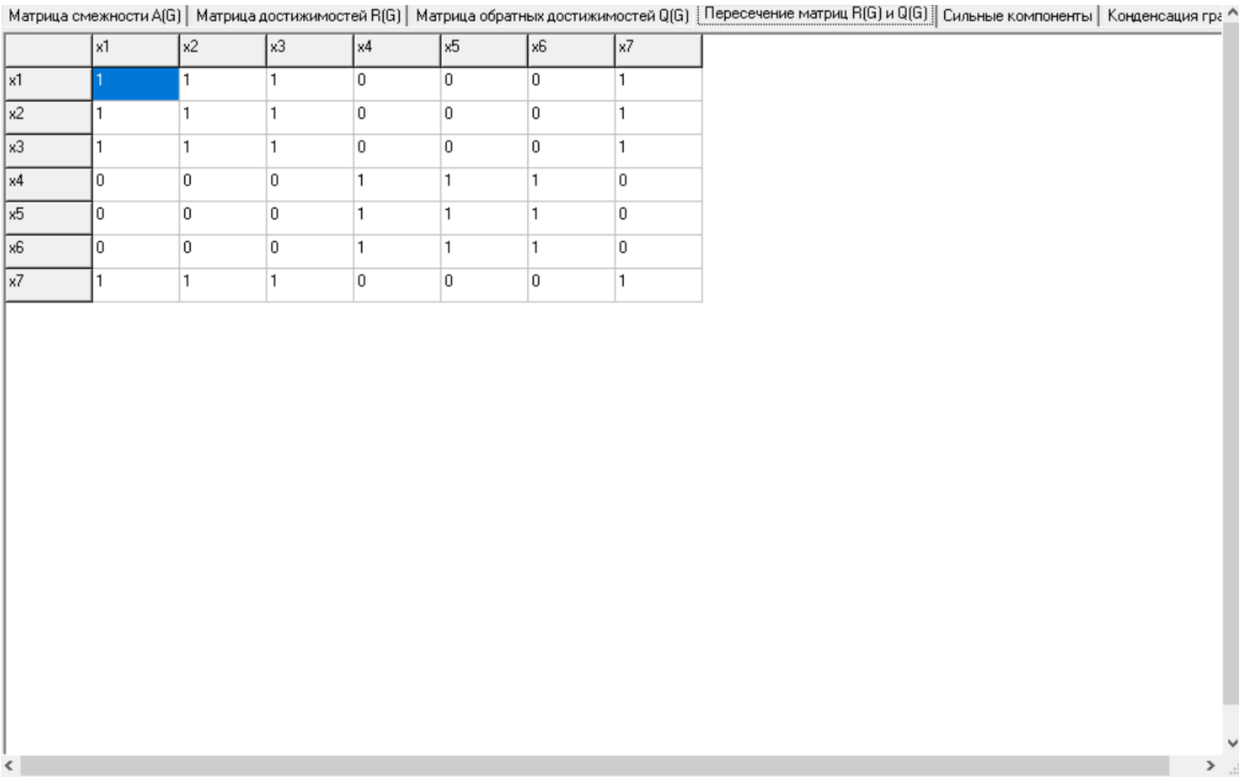
	x1	x2	x3	x4	x5	x6	x7
x1	1	1	1	0	0	0	1
x2	1	1	1	0	0	0	1
x3	1	1	1	0	0	0	1
x4	1	1	1	1	1	1	1
x5	1	1	1	1	1	1	1
x6	1	1	1	1	1	1	1
x7	1	1	1	0	0	0	1

Рис.3-матрица достижимостей



	x1	x2	x3	x4	x5	x6	x7
x1	1	1	1	1	1	1	1
x2	1	1	1	1	1	1	1
x3	1	1	1	1	1	1	1
x4	0	0	0	1	1	1	0
x5	0	0	0	1	1	1	0
x6	0	0	0	1	1	1	0
x7	1	1	1	1	1	1	1

Рис.4-матрица обратных достижимостей



	x1	x2	x3	x4	x5	x6	x7
x1	1	1	1	0	0	0	1
x2	1	1	1	0	0	0	1
x3	1	1	1	0	0	0	1
x4	0	0	0	1	1	1	0
x5	0	0	0	1	1	1	0
x6	0	0	0	1	1	1	0
x7	1	1	1	0	0	0	1

Рис.5-пересечение матрицы смежности и матрицы обратных достижимостей

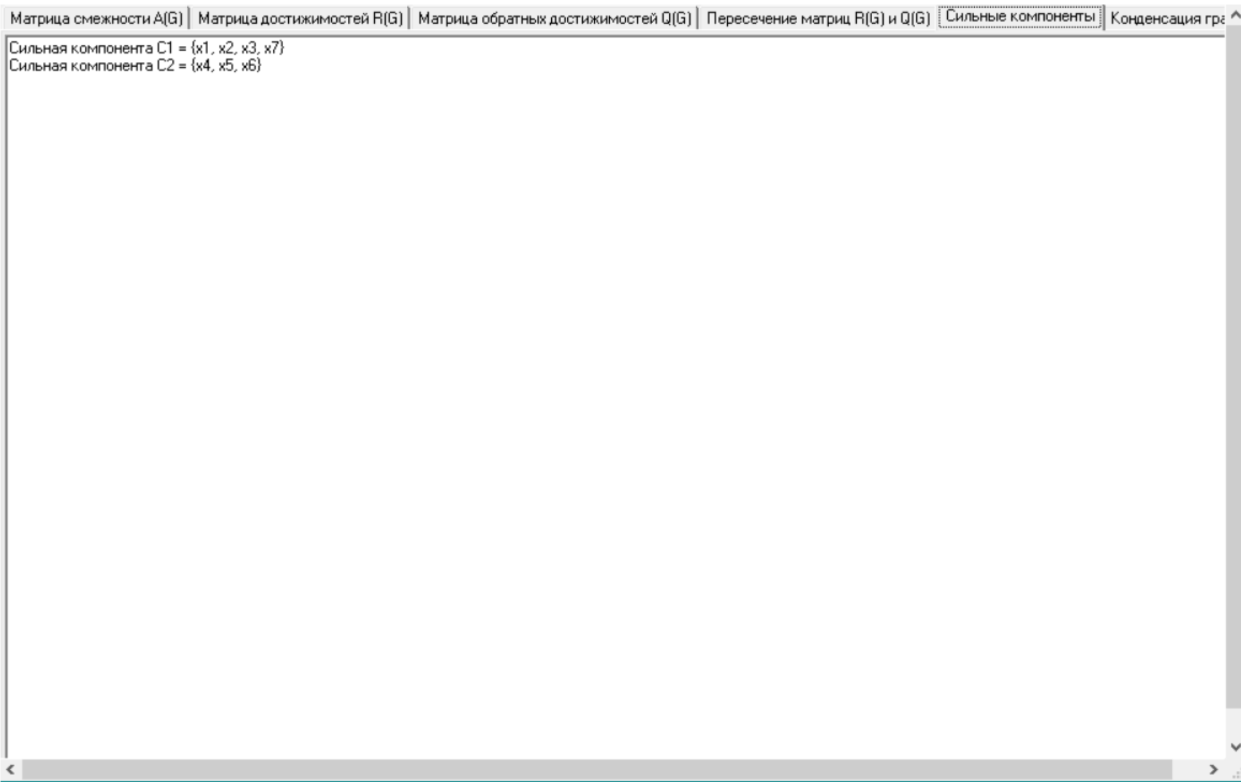


Рис.6-сильные компоненты

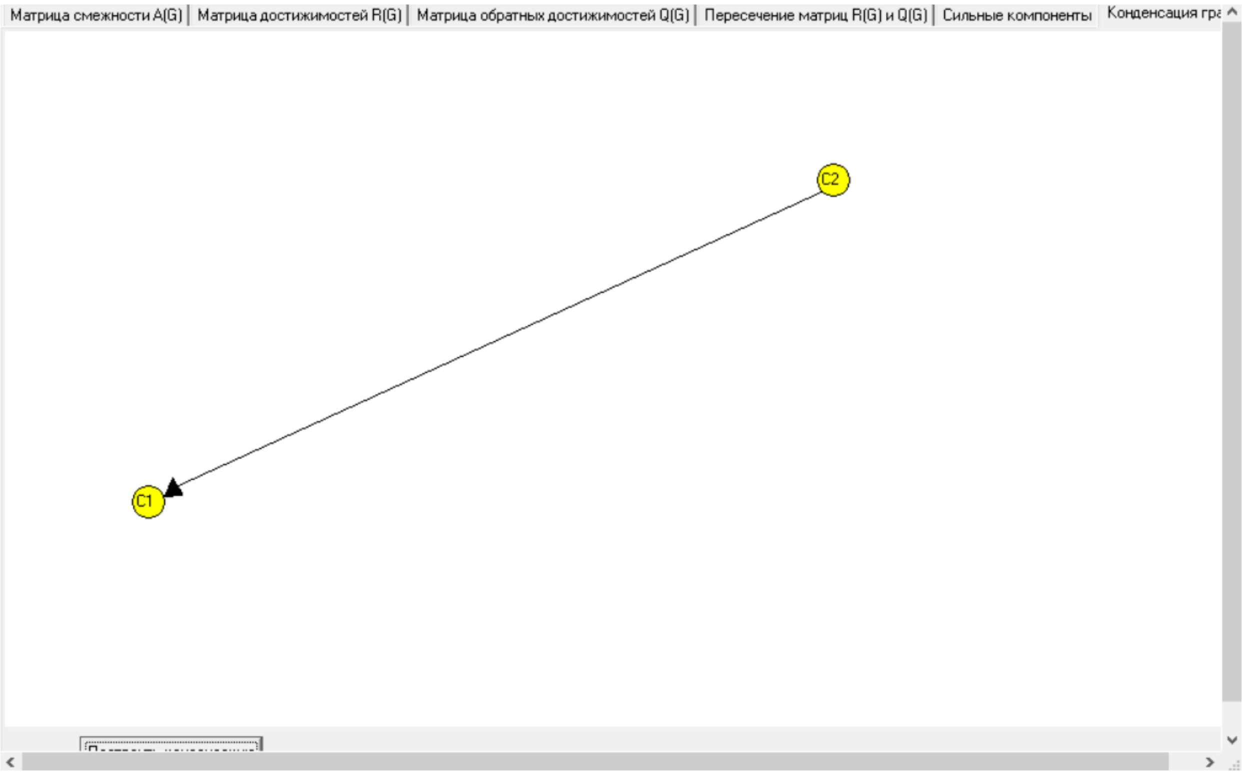


Рис.7-граф конденсации

Прикладная задача

На дистанции присутствуют пункты отдыха и остановок, от которых отходит некоторое количество дорог, в разных направлениях. Несколько бегунов решили проверить сколько пунктов соединены циклами на их дистанции. Для этого бегуны распределились по остановкам и уже от них смотрели, где они окажутся по окончании пути. Пробежав все возможные варианты, с помощью общей полученной информации, они узнали ответ на свой вопрос.