Построение и анализ алгоритмов

Лекция 11

Раздел: Алгоритмы на графах

Тема лекции:

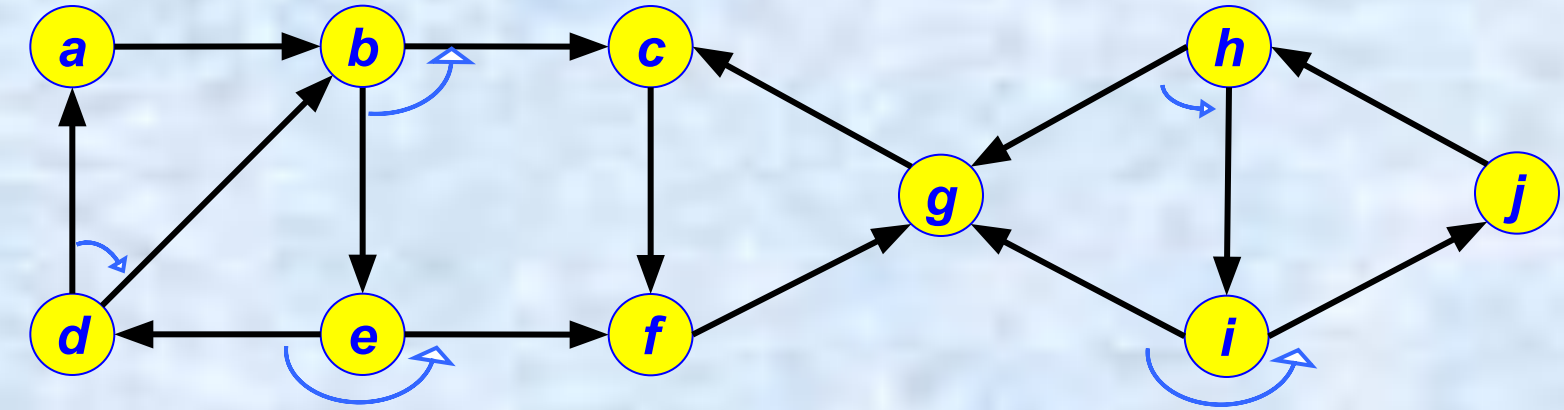
1. Поиск в глубину в ориентированных графах (напомнить!)
2. Топологическая сортировка
3. Сильно связные компоненты (сл.17…)

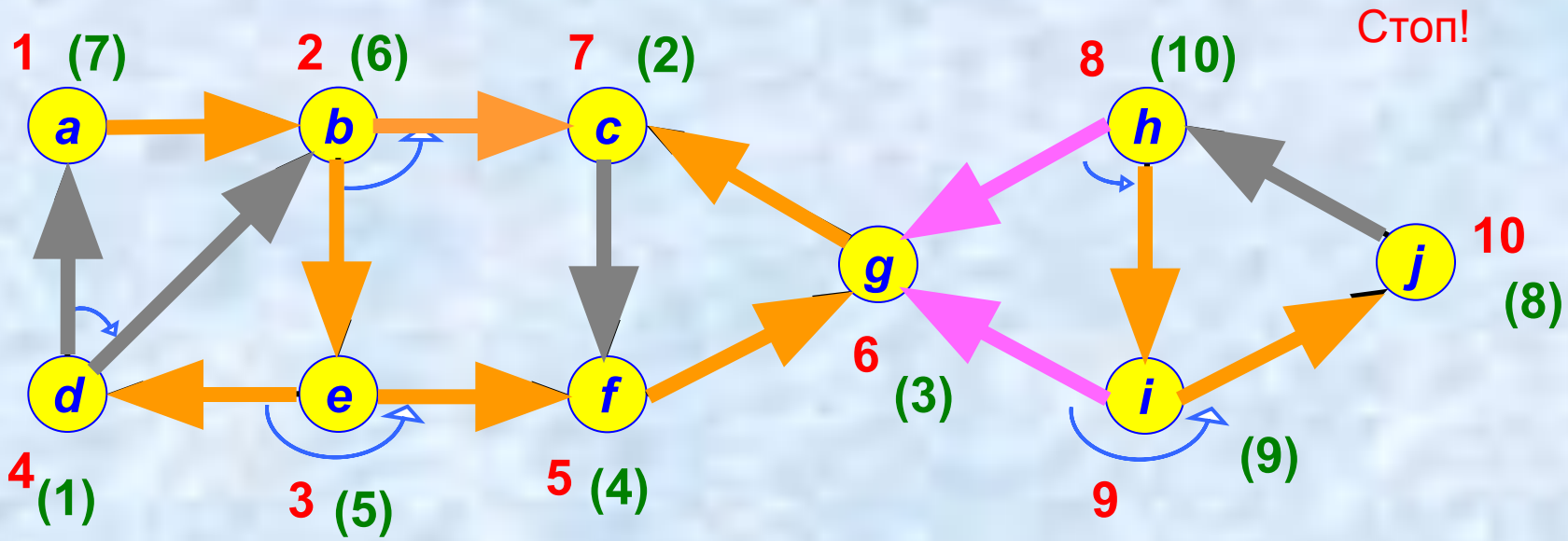
1-2: 21.04.2014

3: 28.04.14

Особенности поиска в глубину (ПВГ) в ориентированных графах

Пример ПВГ в орграфе





Пример ПВГ в орграфе

Древесное ребро – от отца к сыну в глубинном остовном дереве (ГОД)

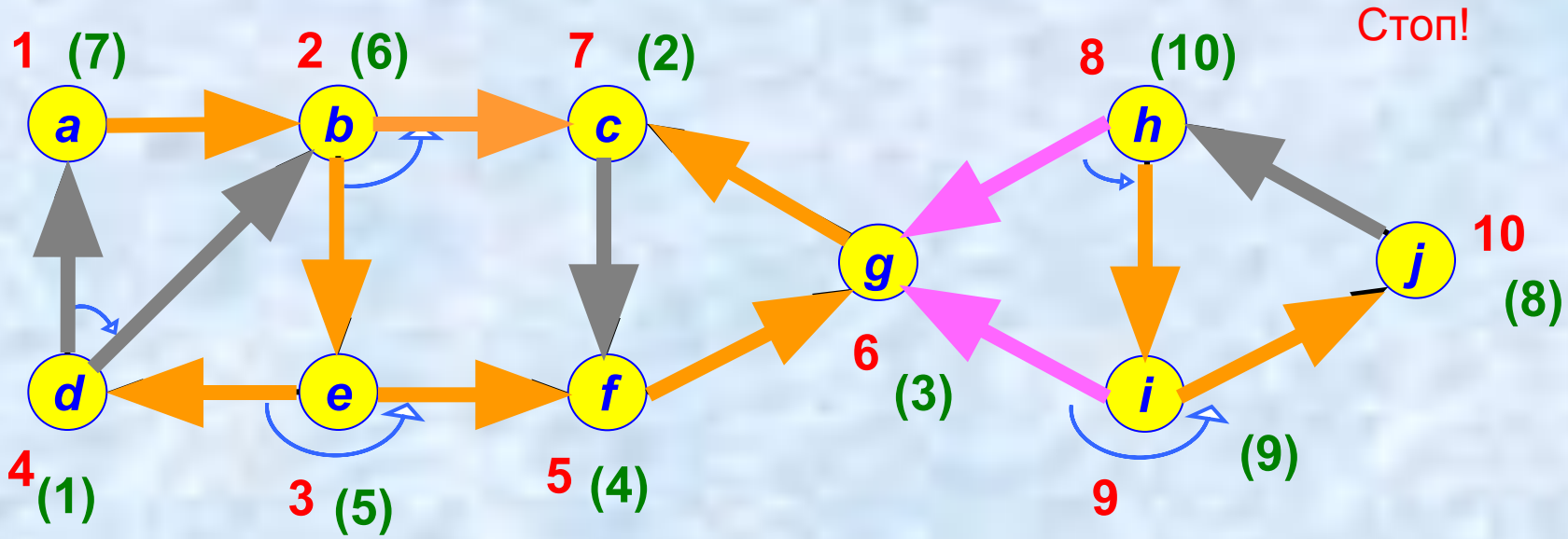
Направленное вперед (***прямое***) ребро – от предка к потомку в ГОД

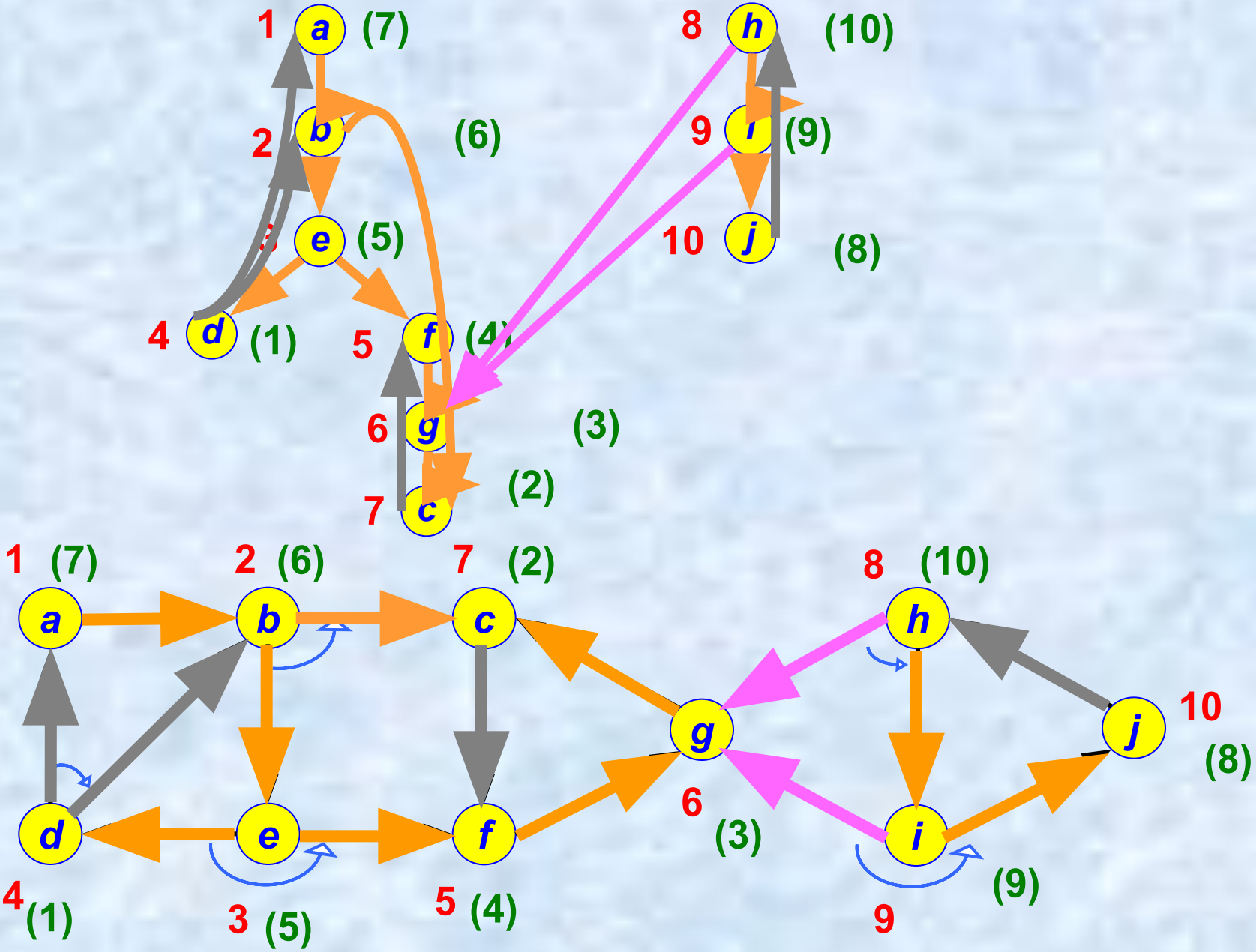
Обратное ребро – от потомка к предку в ГОД

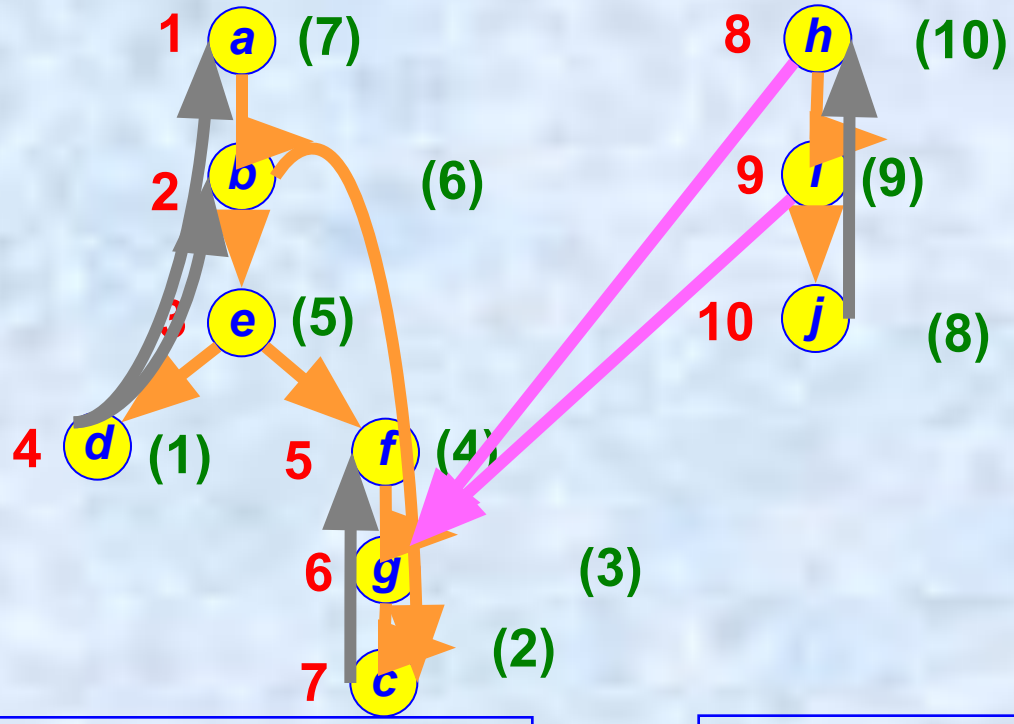
Поперечное ребро – не связывает предка и потомка

1 – номер при ПВГ: *NumVert*(*v*) или *n*(*v*)

(1) – номер в порядке ***использования*** при ПВГ: *fn*(*v*) [finishing]




1 a (7)
2 b (6)
e (5)
4 d (1)
5 f (4)
6 g (3)
7 c (2)
8 h (10)
9 i (9)
10 j (8)
1 (7)
2 (6)
7 (2)
8 (10)
a
b
c
h
j
10
(8)
g
6
(3)
d
e
f
i
4 (1)
3 (5)
5 (4)
9
(9)



Древесное ребро (u,v):
$n(u)<n(v)$,

в момент рассмотрения
$n(v)=0$

Направленное вперед
ребро(u,v): $n(u)<n(v)$,

в момент рассмотрения
$n(v)\neq 0$

Обратное ребро (u, v): $n(u)>n(v)$ и $fn(v)=0$

Поперечное ребро (u, v): $n(u)>n(v)$ и $fn(v)\neq 0$

Топологическая сортировка

Ациклический (бесконтурный) *ор*граф

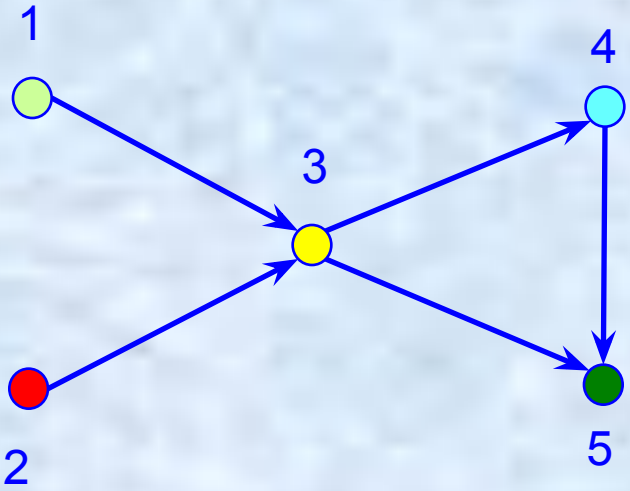
Лемма 1. Орграф G является ациклическим ***ттогда***, когда поиск в глубину в G *не находит обратных ребер.*

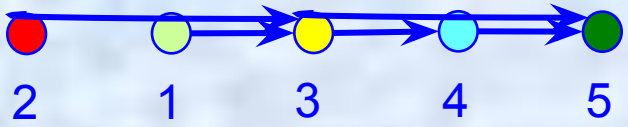
Лемма 2. В произвольном ациклическом орграфе $G=(V,E)$ вершины могут быть *пронумерованы* так, что каждая дуга будет иметь вид (v_i,v_j), где $i<j$.

Примеры:

- *Сетевой график* работ.
- Связи между дисциплинами учебного плана (*пре*реквизиты и *пост*реквизиты).
- *Информационный граф* программы (алгоритма).

Пример 0

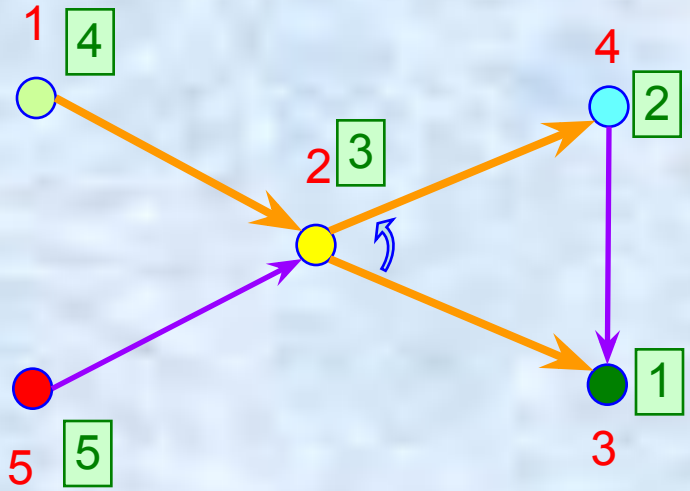


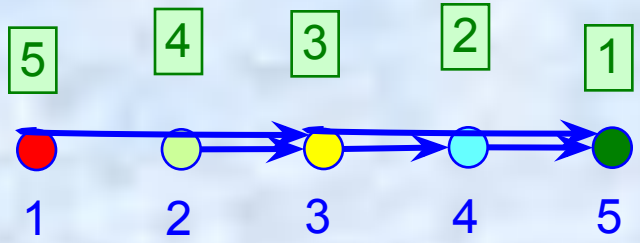


(пере)нумерация

(пере)группировка

Продолжение примера 0 (ПВГ)





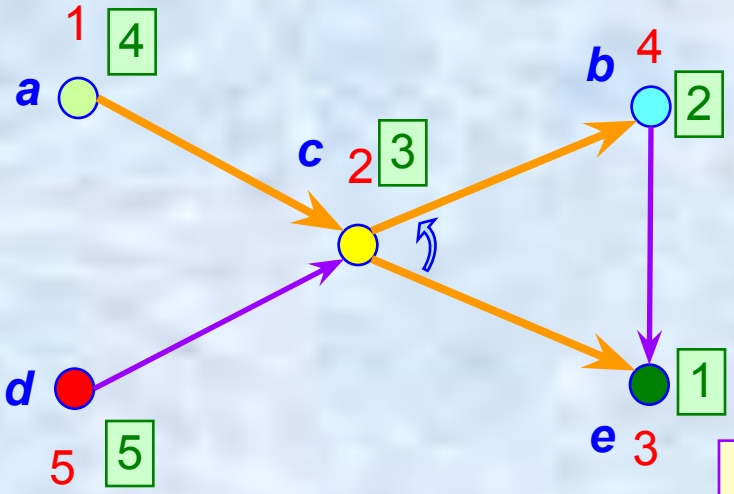
***ПВГ* (поиск в глубину)**

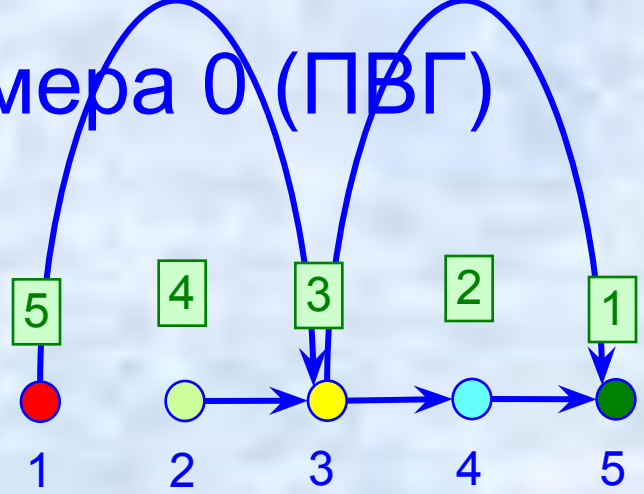
1- номер в порядке посещения

4 – номер в порядке использования

(пере)группировка

Продолжение примера 0 (ПВГ)





	1	2	3	4	5
v	***a***	***b***	***c***	***d***	***e***
nv	**1**	**4**	**2**	**5**	**3**
fn	**4**	**2**	**3**	**5**	**1**
sn	**2**	**4**	**3**	**1**	**5**
tn	**4**	**1**	**3**	**2**	**5**

v [*] – имена вершин *в исходной нумерации;*

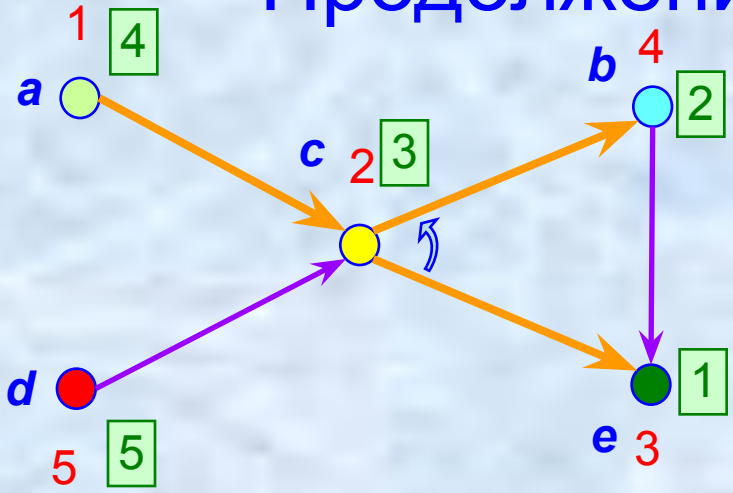
nv [*] - номера в порядке посещения;

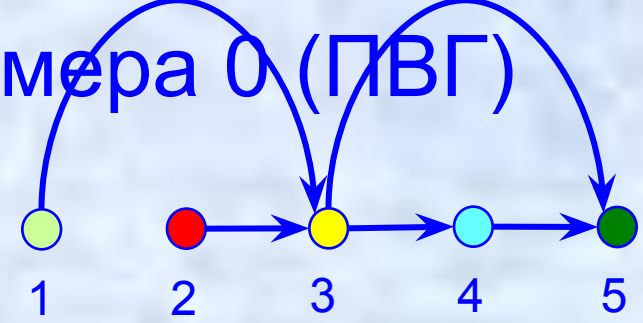
fn [*] - номера в порядке использования;

sn [*] – номера в порядке топ.сортировки, т.е. *sn* [*i*] – новый номер вершины с исходным номером *i*;

tn [*i*] – исходный номер вершины с номером *i* в порядке топ.сортировки

Продолжение примера 0 (ПВГ)





1) Получение *sn* [*] из *fn* [*]:

for *i* := 1..n **do** *sn* [*i*] := n - *fn* [*i*] + 1;

2) Получение *tn* [*] из *sn* [*]:

for *i* := 1..n **do** *tn* [*sn* [*i*]] := *i*;

Или получение *tn* [*] сразу из *fn* [*]:

for *i* := 1..n **do** *tn* [n - *fn* [*i*] + 1] := *i* ;

Оказывается, что

sn [*] и *tn* [*] –

обратные перестановки

(см. след. слайд)

Перестановки

Перестанова sn[*]: $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 1 & 3 & 2 & 5 \end{pmatrix}$ *или*

$$\begin{matrix} 1 & 2 & 3 & 4 & 5 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 4 & 1 & 3 & 2 & 5 \end{matrix}$$

Обратная перестановка tn[*]: $\pi^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 3 & 1 & 5 \end{pmatrix}$

$\pi(1) = 4, \pi(2) = 1, \pi(3) = 3, \pi(4) = 2, \pi(5) = 5$

$\alpha \circ \beta = \alpha(\beta) \qquad (\alpha \circ \beta)(i) = \alpha(\beta(i)) \qquad \pi \circ \pi^{-1} = e$

$\pi \circ \pi^{-1}(1) = \pi(\pi^{-1}(1)) = \pi(2) = 1; \quad \pi \circ \pi^{-1}(2) = \pi(\pi^{-1}(2)) = \pi(4) = 2;$

$\pi \circ \pi^{-1}(3) = \pi(\pi^{-1}(3)) = \pi(3) = 3; \quad \pi \circ \pi^{-1}(4) = \pi(\pi^{-1}(4)) = \pi(1) = 4;$

$\pi \circ \pi^{-1}(5) = \pi(\pi^{-1}(5)) = \pi(5) = 5;$

tn [*sn* [*i*]] := *i* , где *i* – из исходной нумерации,
sn [*tn* [*j*]] := *j* , где *j* – из нумерации топ.сортировки

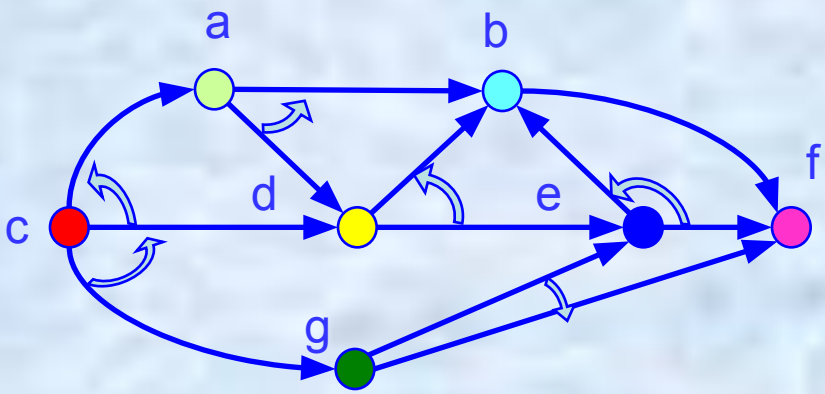
Алгоритм топологической сортировки

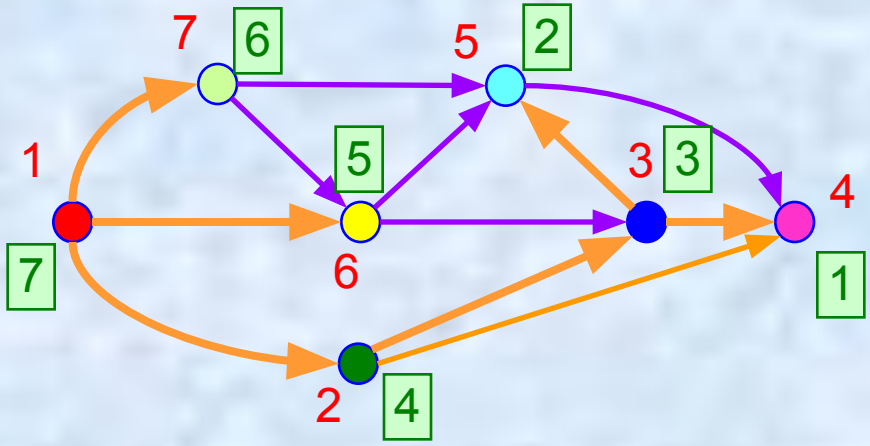
1. ***Выполнить ПВГ*** в орграфе G, заполнив массив fn[*].
2. ***Пронумеровать*** вершины номерами (n - fn[v] + 1), т.е. заполнить *sn* [*] – вектор (последовательность) новых номеров в порядке топологической сортировки (*sn* [*i*] – новый номер вершины с исходным номером *i*).
3. ***Перегруппировать*** вершины, т.е. заполнить *tn* [*] – вектор старых номеров в новом порядке (*tn* [*i*] – исходный номер вершины с номером *i* в порядке топологической сортировки)

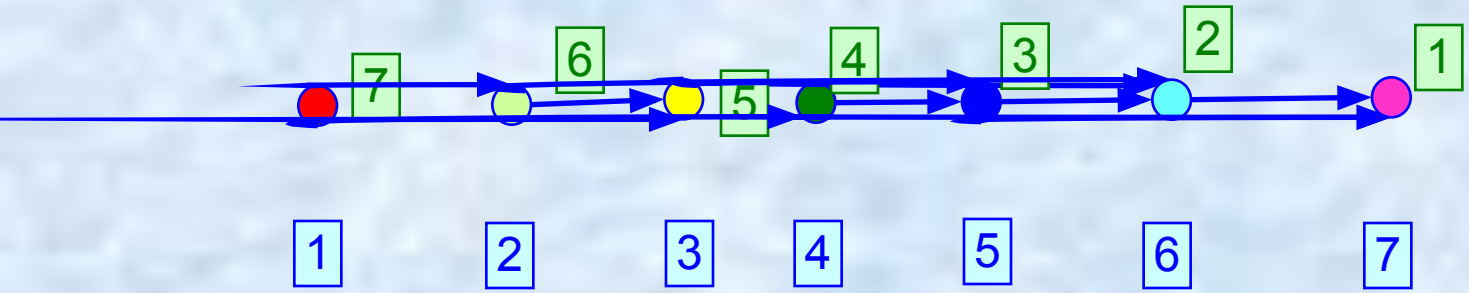
Алгоритм топологической сортировки

Пример 1

	1	2	3	4	5	6	7
v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
nv	**7**	**5**	**1**	**6**	**3**	**4**	**2**
fn	**6**	**2**	**7**	**5**	**3**	**1**	**4**
sn	**2**	**6**	**1**	**3**	**5**	**7**	**4**
tn	**3**	**1**	**4**	**7**	**5**	**2**	**6**

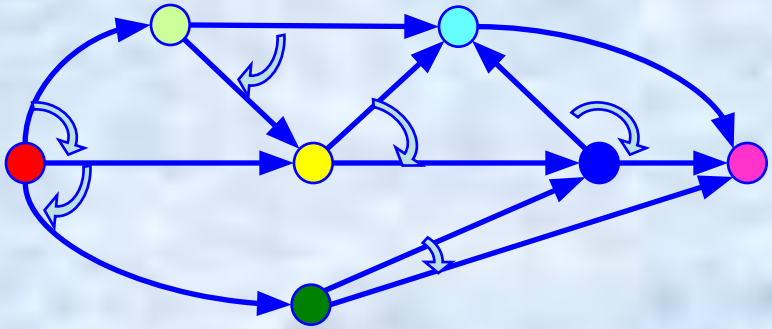


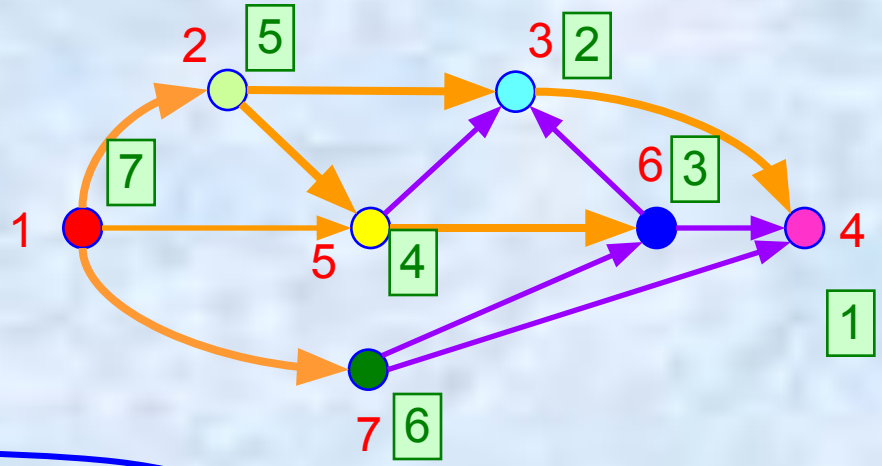


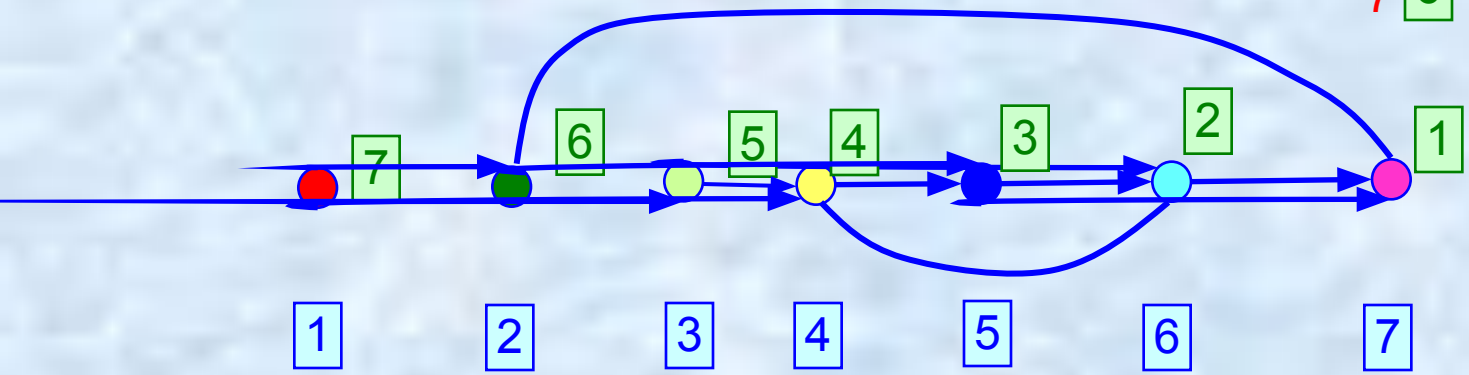


Алгоритм топологической сортировки

Пример 2 (другие списки смежности)



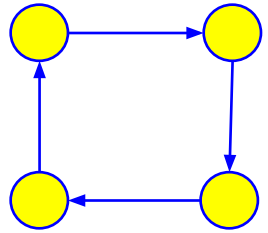
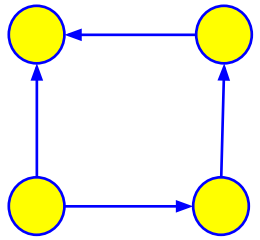
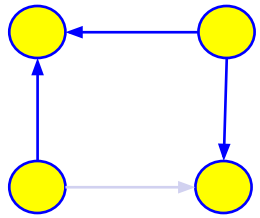




Продолжение
на следующей лекции!
(28 апреля)

Алгоритм на основе поиска в глубину

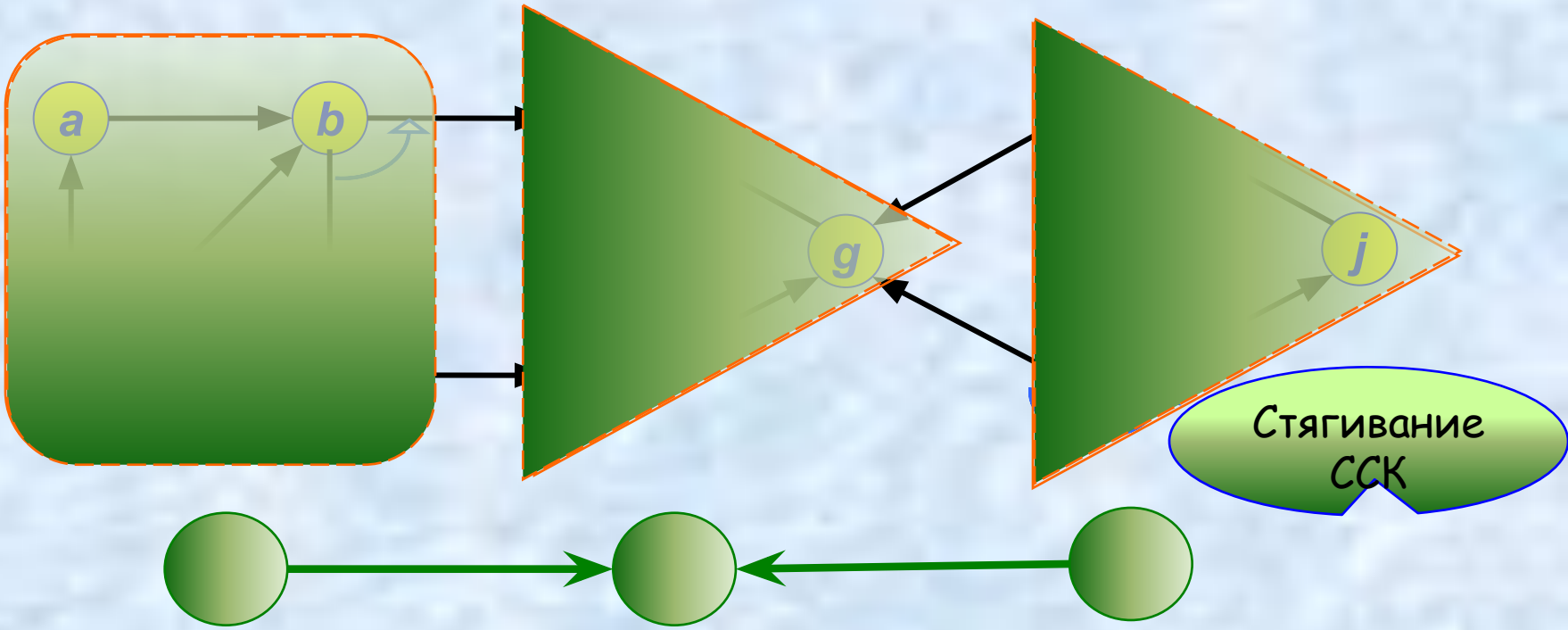
Сильная связность (Strongly Connection)

Определение. Орграф *G сильносвязный*, если любые две его вершины достижимы друг из друга.	
Определение. Орграф *G односторонне связный*, если для любой его пары вершин по меньшей мере одна достижима из другой.	
Определение. Орграф *G слабо связный*, если любые две его вершины соединены *полупутём*.	

Сильно связные компоненты (Strongly Connected Components)

Сильно связная компонента – максимальный сильно связный подграф

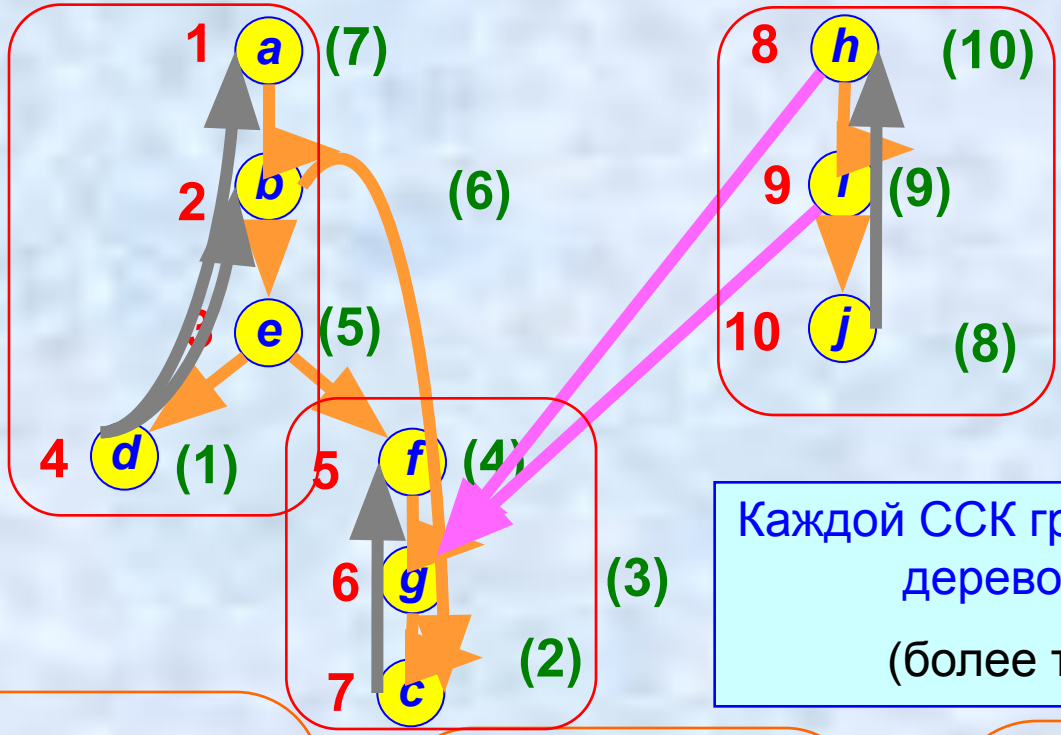
Сильно связная компонента = ССК



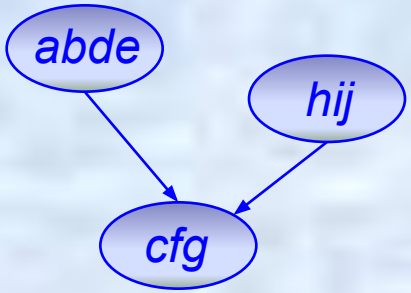
?

При стягивании каждой сильно связной компоненты в одну вершину получается ациклический ориентированный граф. (Почему?)

ПОИСК
в глубину

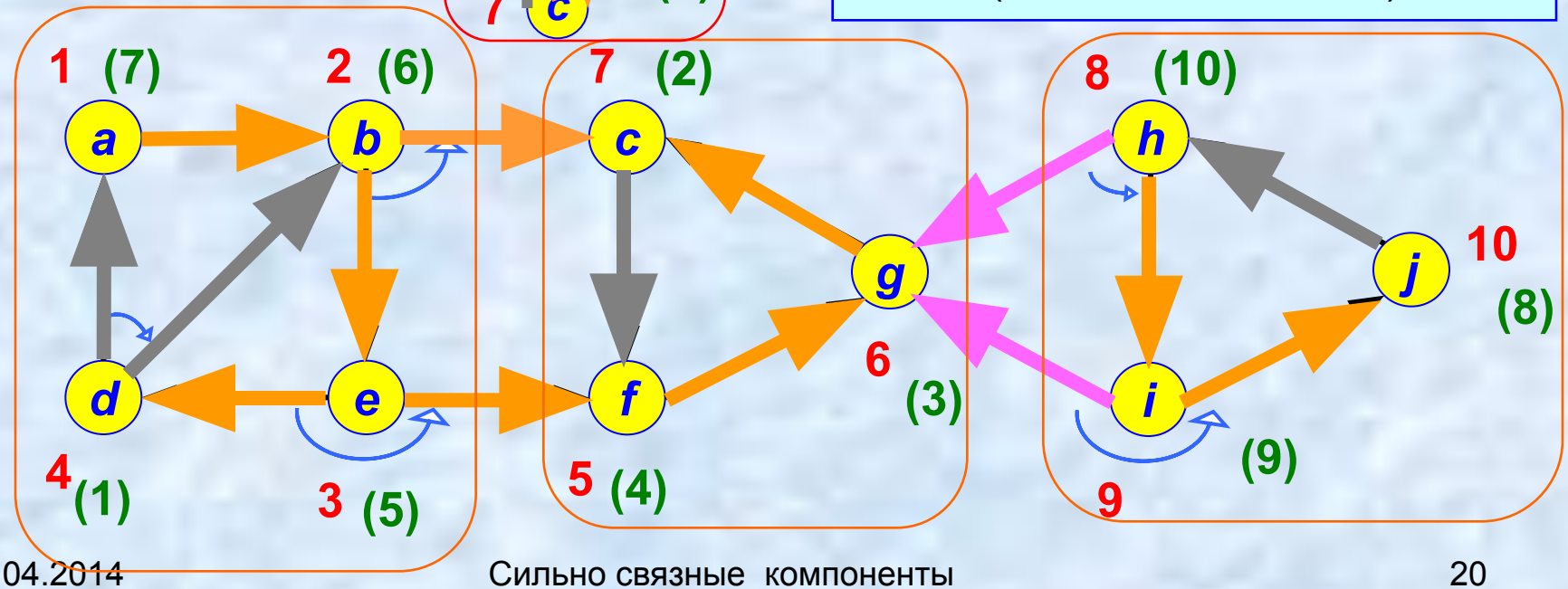


Граф ССК



Каждой ССК графа соответствует дерево в ПВГ-лесу

(более точно далее)



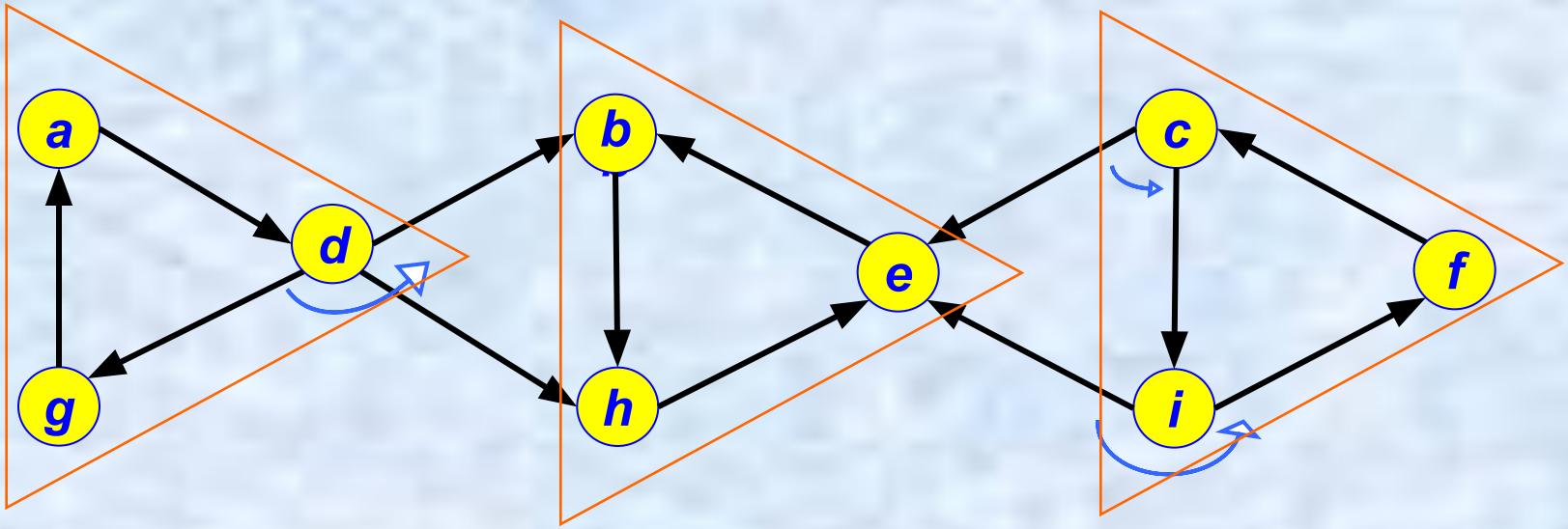
Соответствие «ССК – дерево»

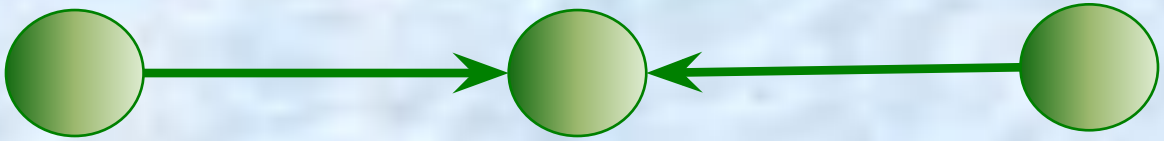
Утверждение

Пусть $G_i = (V_i, E_i)$ – ССК орграфа G, а $F = (V, T)$ - глубинный остовный лес для G. Тогда узлы V_i орграфа G вместе с ребрами из $E_i \cap T$ образуют дерево.

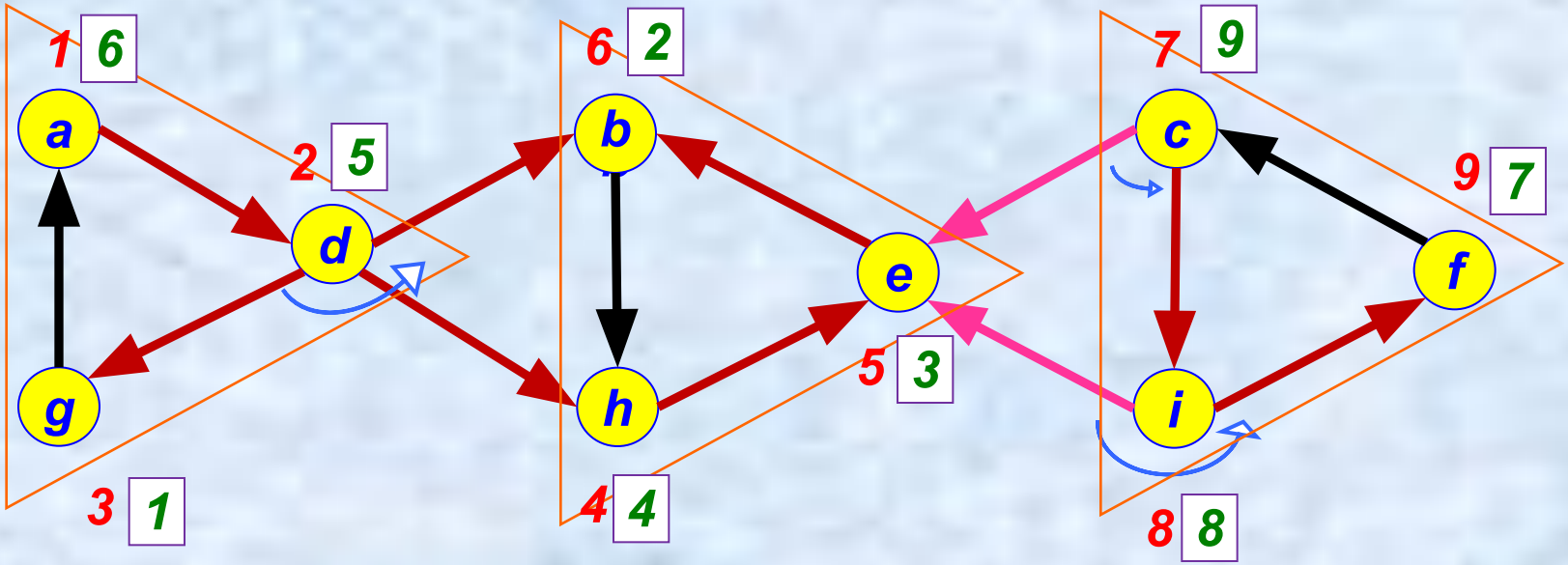
Т.о. «ССК ⇒ дерево», но «дерево ⇒ ССК» - не верно (см. пример) + ещё примеры на след. слайдах

Ещё пример

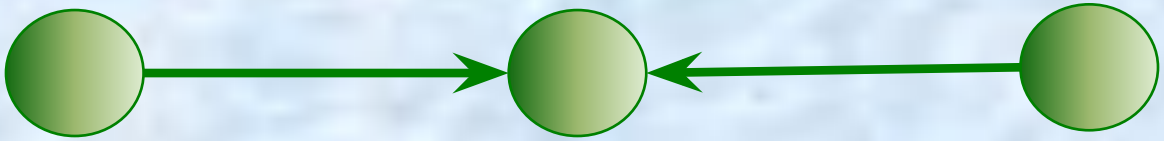




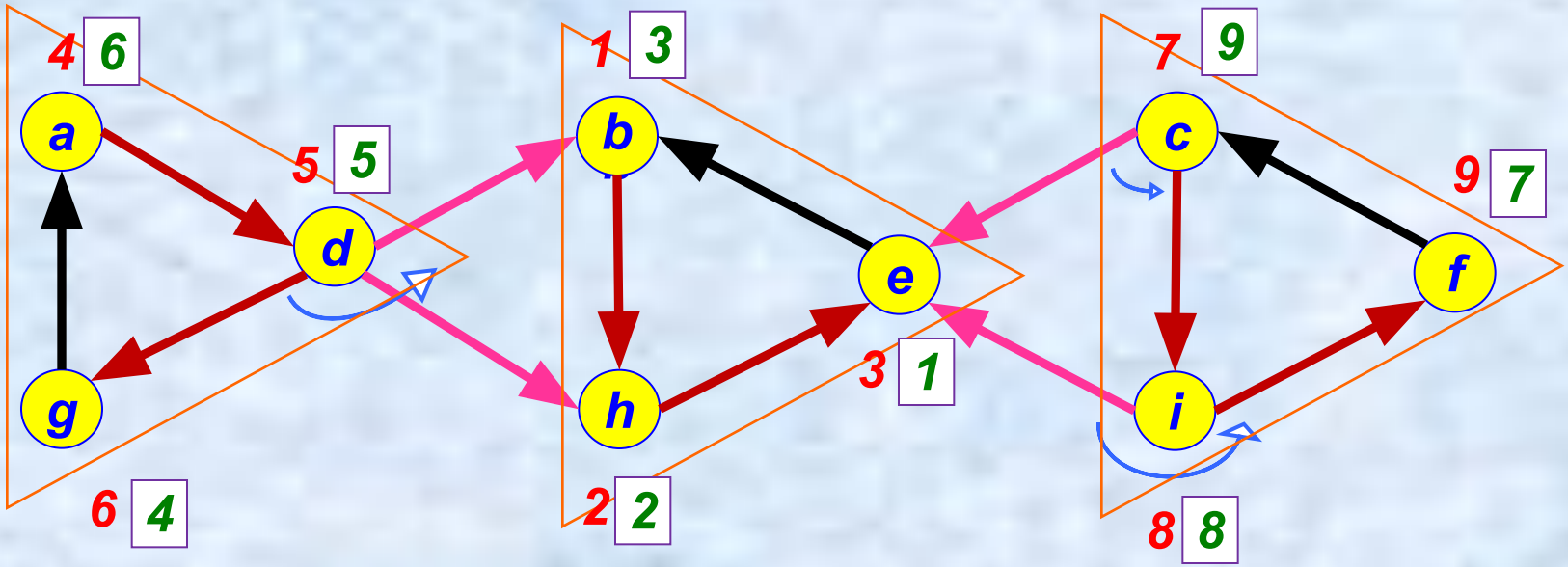
Вариант 1 (начало с «a»)



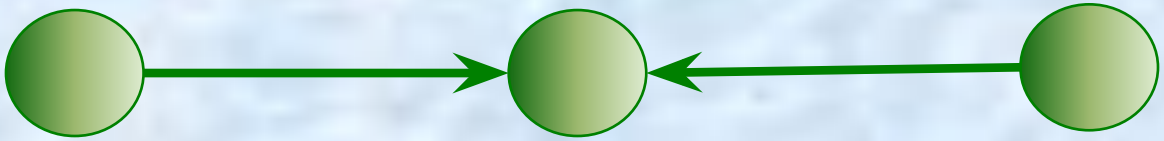
Остовный лес из двух деревьев



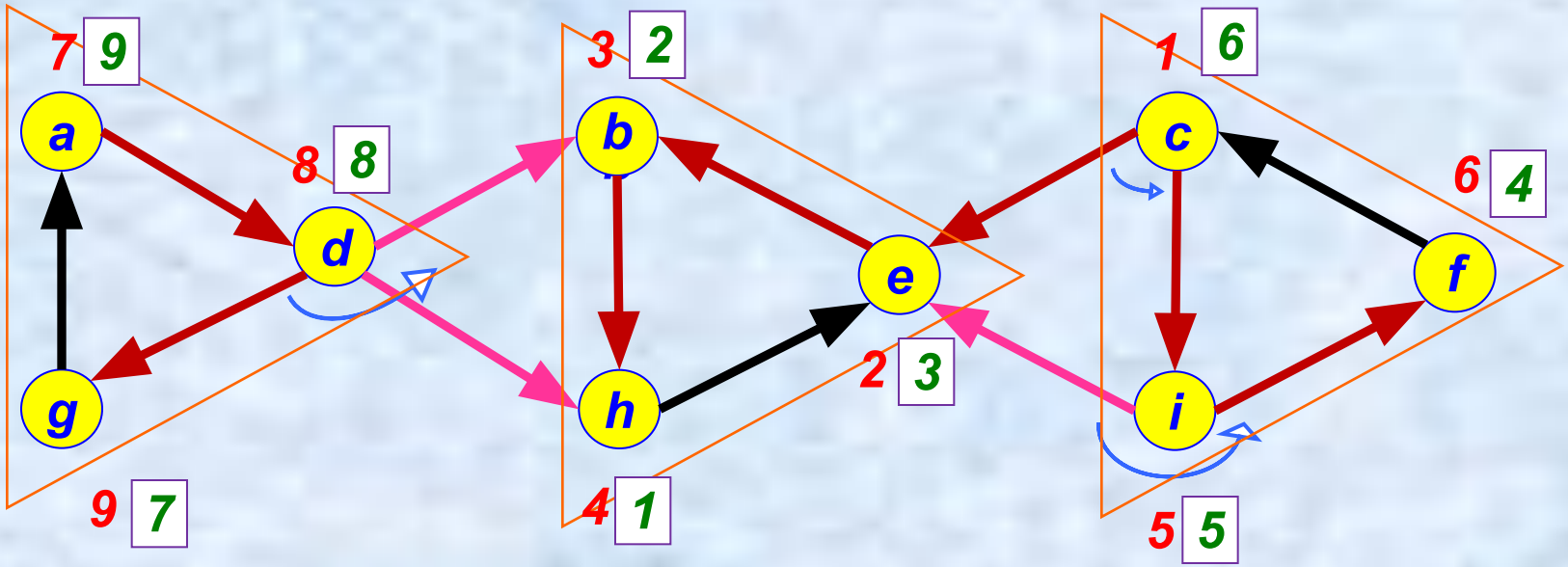
Вариант 2 (начать с «b»)



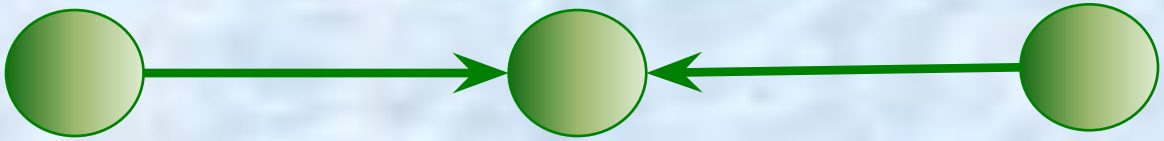
Остовный лес из трех деревьев



Вариант 3 (начать с «c»)



Остовный лес из двух деревьев



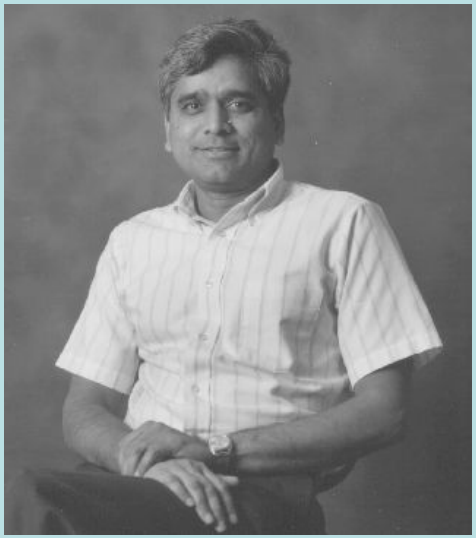
Алгоритм на базе ПВГ (DFS)

Хорошо бы при ПВГ обнаруживать ***корни деревьев***, соответствующих ССК (!)

Алгоритм Тарьяна (Tarjan R.E.) аналогичен ранее рассмотренному алгоритму нахождение двусвязных компонент.

Мы рассмотрим далее другой алгоритм (Алгоритм Косарайю).

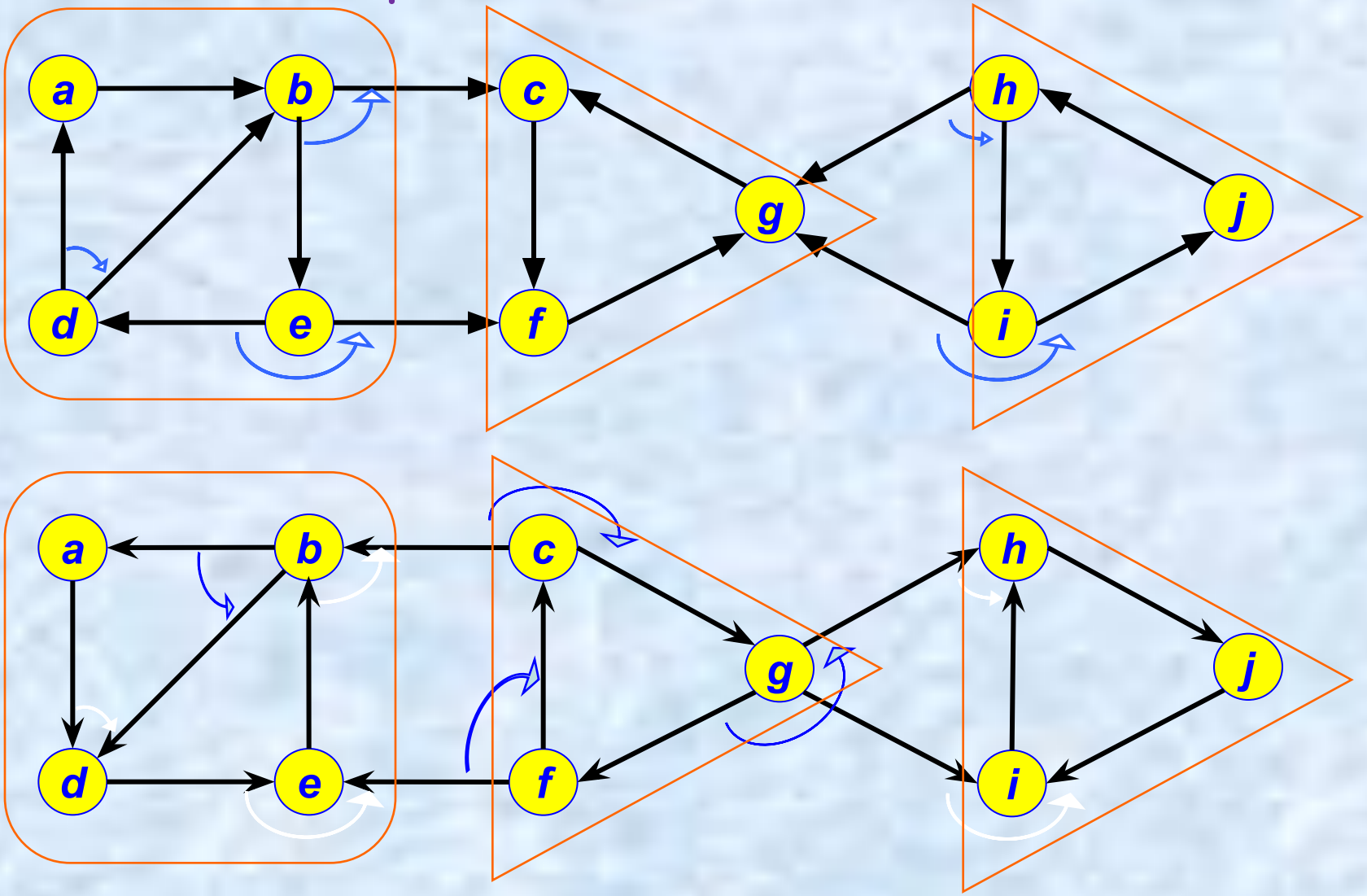
is a professor of Computer Science at Johns Hopkins University is a professor of Computer Science at Johns Hopkins University, and division director for Computing & Communication Foundations at the National Science Foundation is a professor of Computer Science at Johns Hopkins University, and division director for Computing & Communication Foundations at the National Science Foundation.He has done extensive work in the design and analysis of parallel and sequential algorithms



Research

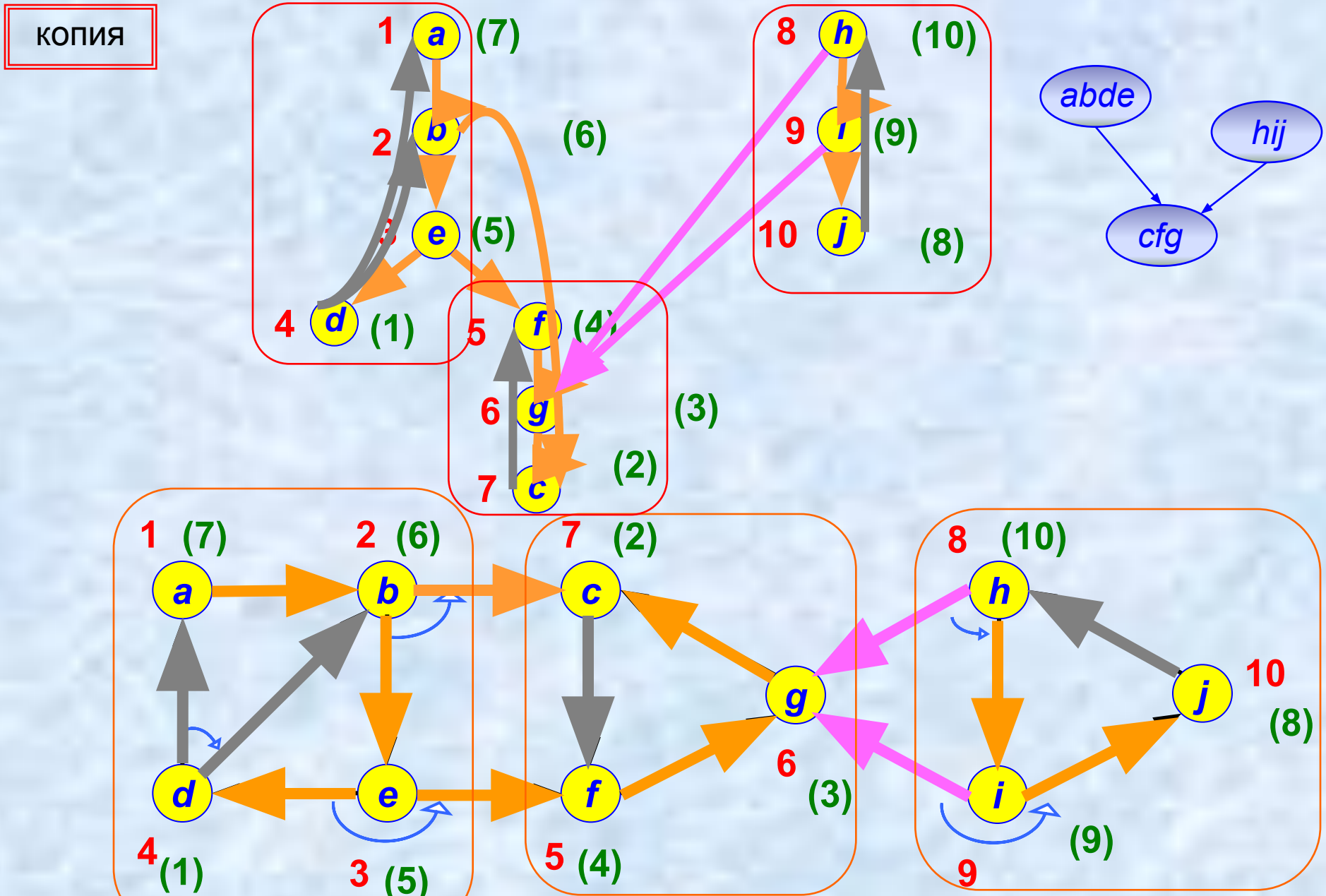
Prof. Kosaraju has done extensive work in the design and analysis of parallel and sequential algorithms. Recent research efforts include efficient algorithms for pattern matching, data structure simulations, universal graphs, DNA sequence assembly, and derandomization. Current research interests also include investigations of immune system responses.

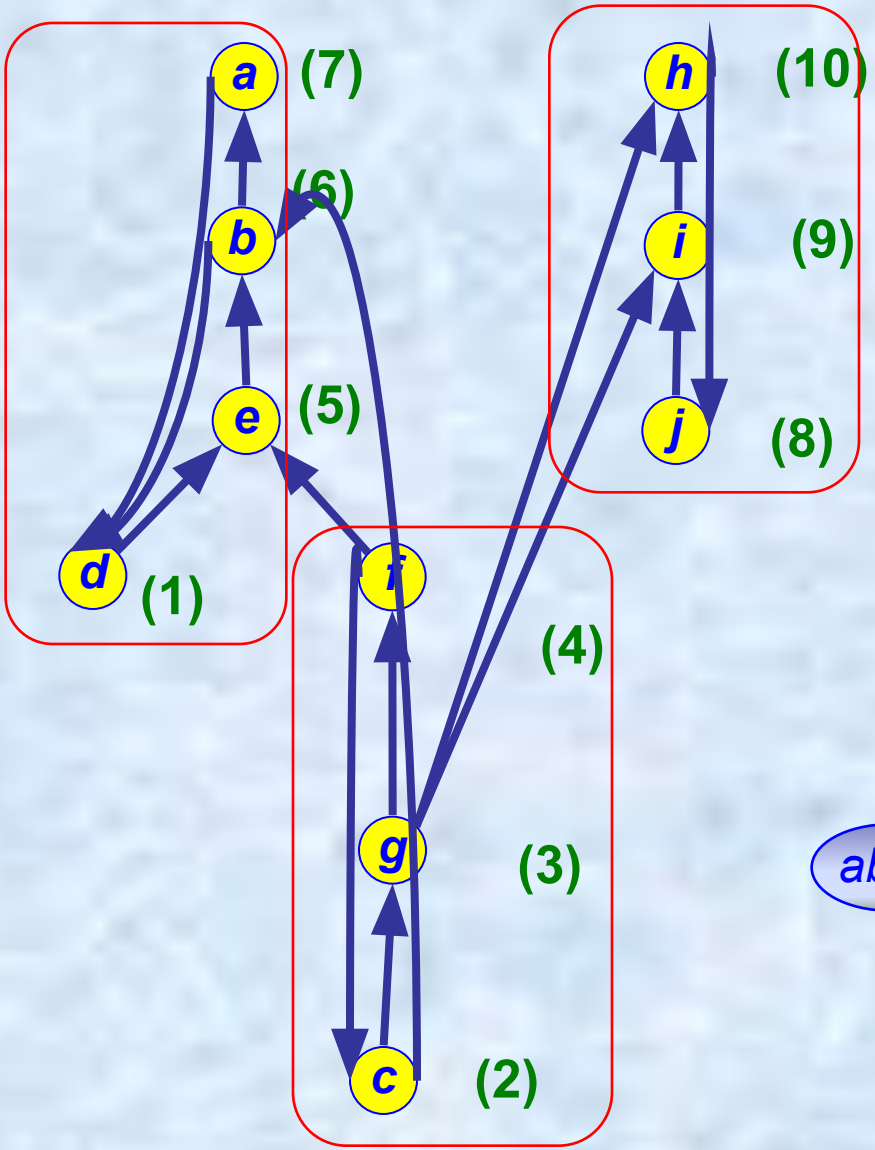
Обращение орграфа (понадобится далее), при этом ССК – те же (!)



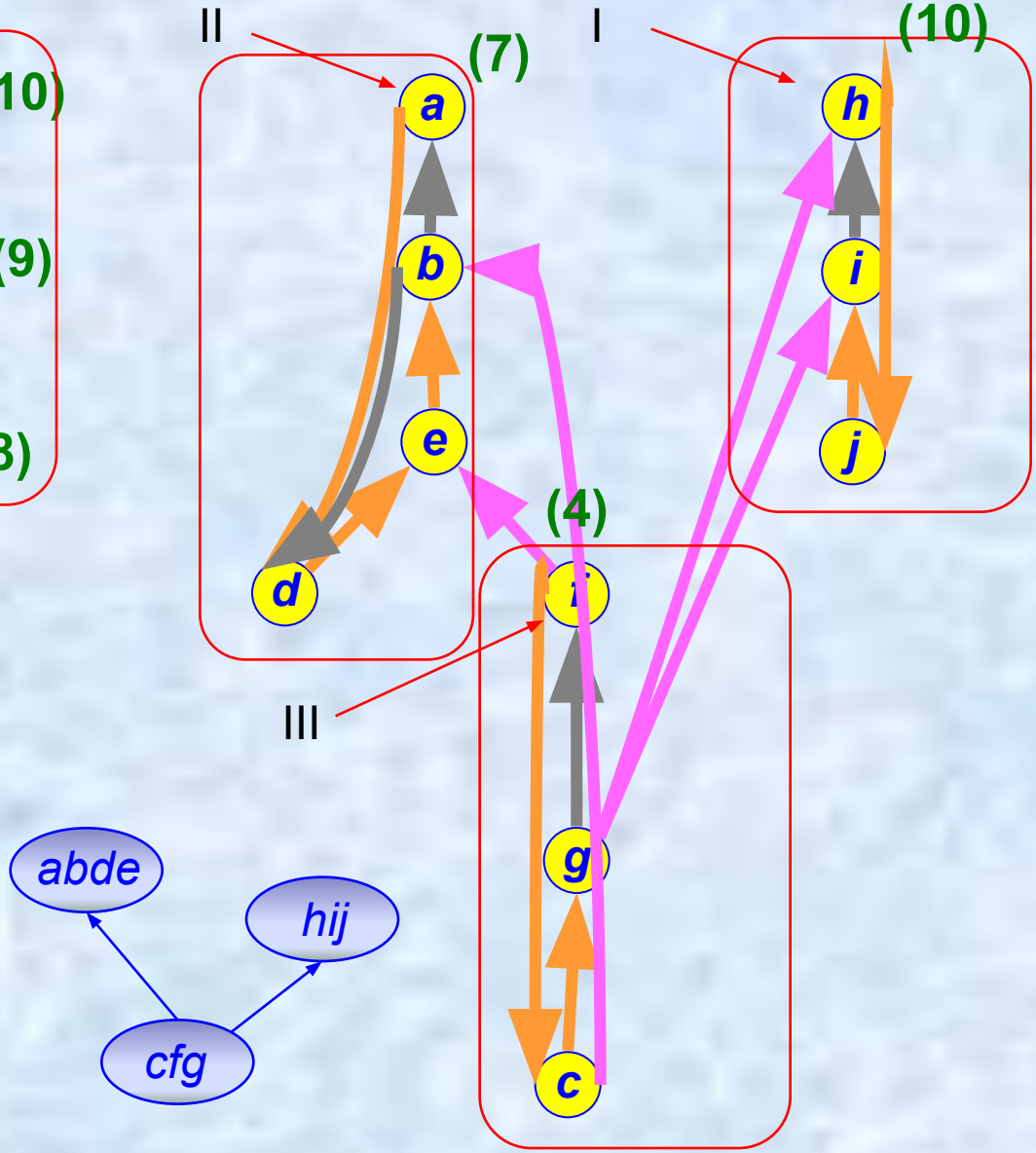
Алгоритм Косарайю (S.R.Kosaraju) нахождения ССК орграфа (на основе двух ПВГ)

1. Выполнить ПВГ орграфа G и вычислить номера вершин в порядке их использования fn[*].
2. Сконструировать новый (обращенный) орграф G^R , путем обращения всех дуг исходного графа.
3. Выполнить ПВГ орграфа G^R , начиная с вершины, имеющей наибольший номер fn[ ]. Если ПВГ не завершен, то продолжить, начиная с вершины, имеющей наибольший номер fn[ ] среди оставшихся.
4. Каждое дерево полученного глубинного остовного дерева (леса) орграфа G^R является ССК орграфа G.


копия
1
a
(7)
2
b
(6)
3
e
(5)
4
d
(1)
5
f
(4)
6
g
(3)
7
c
(2)
8
h
(10)
9
i
(9)
10
j
(8)
abde
hij
cfg
1 (7)
2 (6)
7 (2)
8 (10)
10
(8)
6
(3)
4 (1)
3 (5)
5 (4)
9
(9)



Обращенный граф



ПВГ в обращенном графе

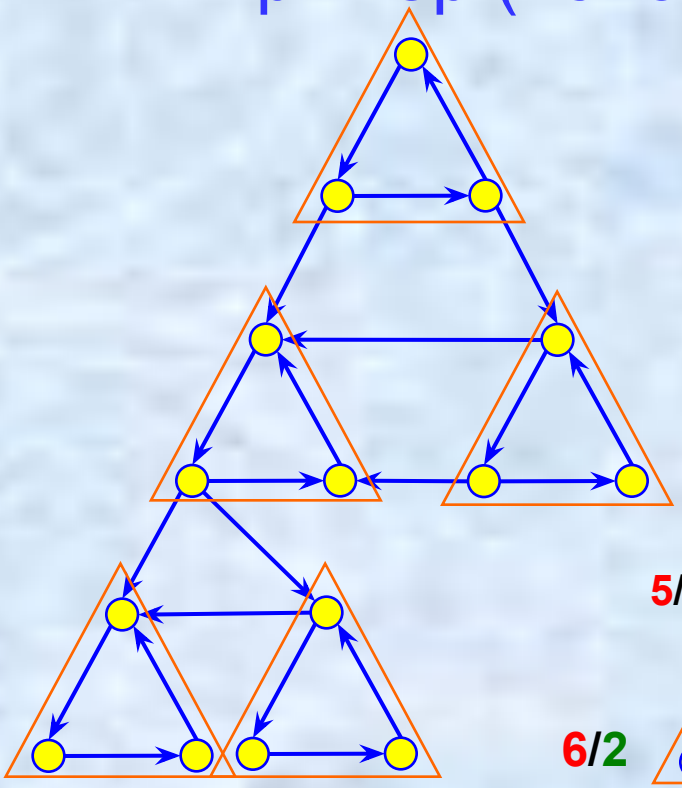
Основные факты (утверждения), на которые опирается доказательство правильности алгоритма

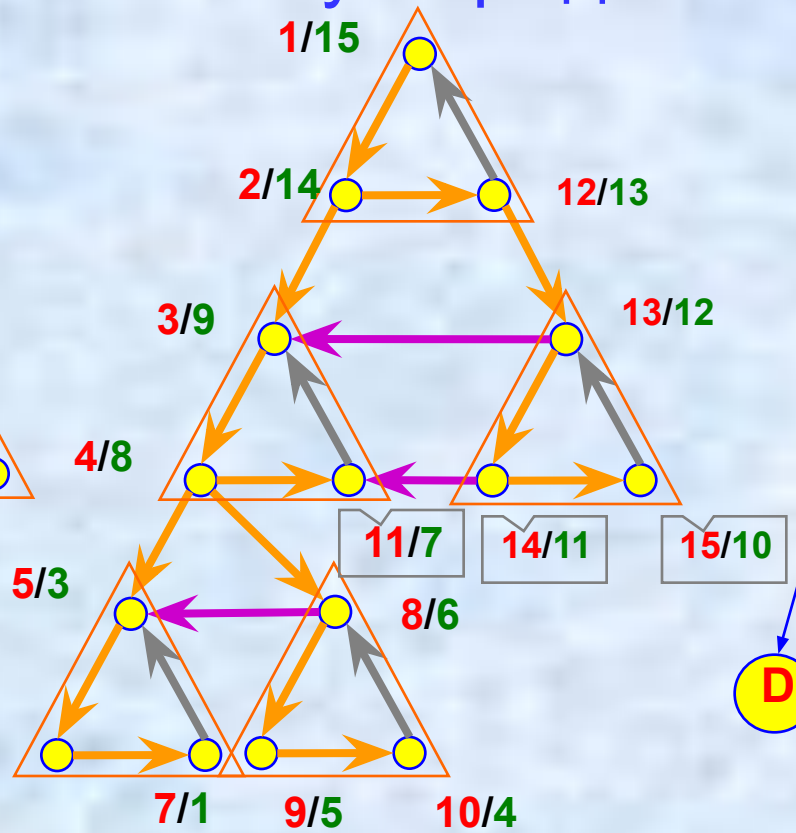
1. Граф ССК $G^{ССК}$ орграфа G – ациклический
2. ССК орграфов G и G^R совпадают (как множества вершин)
3. При втором ПВГ граф ССК $(G^R)^{ССК}$ орграфа G^R проходится в порядке, обратном топологической сортировке

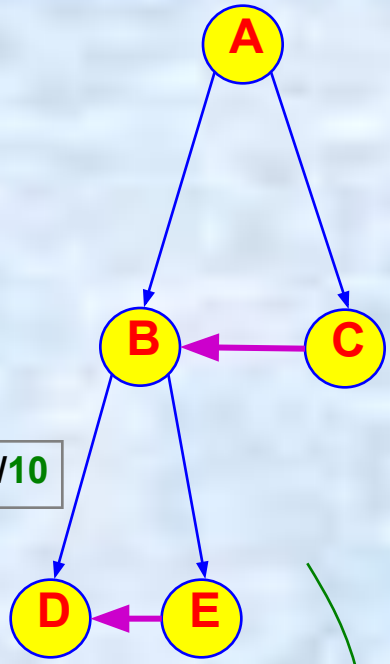
Иначе говоря

- Если начать с вершины, которая лежит в стоке стянутого графа, то мы найдем компоненту, соответствующую стоку.
- Если мы нашли компоненту-сток, то мы можем выкинуть её из графа и продолжить поиск.
- Вершина графа с самым большим значением fn лежит в истоке стянутого графа.
- Сильно связные компоненты можно топологически упорядочить по максимальным значениям fn содержащихся в них вершин.

Пример (пояснение к утверждениям)



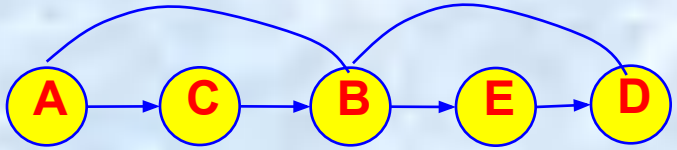




G

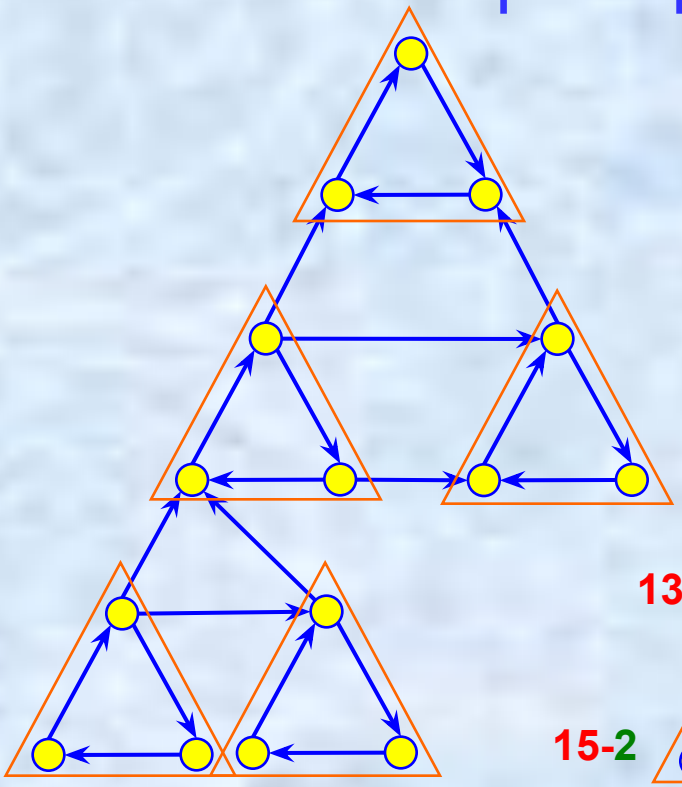
ПВГ в G

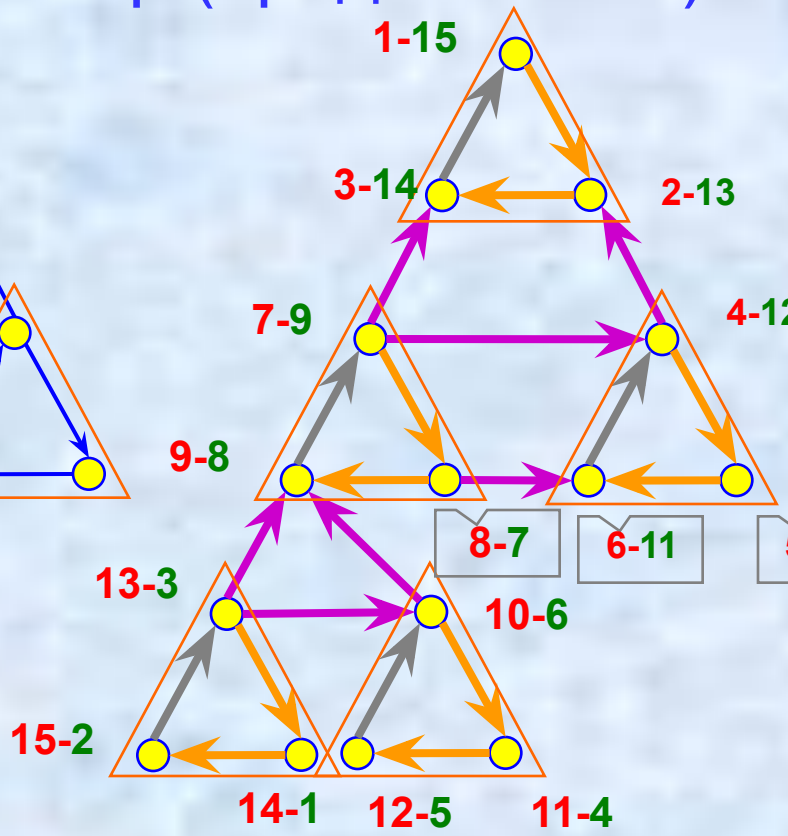
$G^{ССК}$

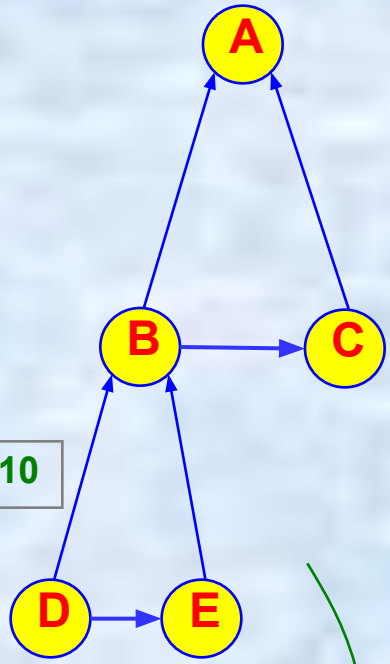


Топ. сорт. графа

Пример (продолжение)





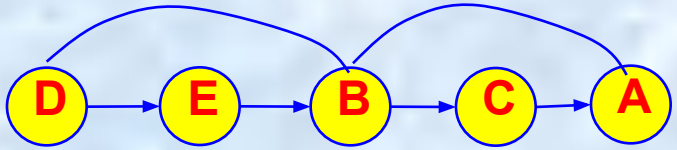


G^R

ПВГ в G^R

$(G^R)^{ССК}$

A→ C → B → E → D



Топ. сорт. графа

Сложность алгоритмов нахождения ССК

Алгоритм Тарьяна (Tarjan R.E.)

Алгоритм Косарайю (S.R.Kosaraju)

$$O(n+m)$$

КОНЕЦ ССК

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ