ПРОГРАММИРОВАНИЕ

Семинар 12.
Деревья, графы

Новосибирский государственный университет, 2019

Реализация бинарных деревьев на Си

```
typedef struct node {
    char *word;
    struct node *left;
    struct node * right;
} tree;

void print_tree(tree *t)
{
    if (!t) return;
    print_tree(t -> left);
    printf("%s\n", t -> word);
    print_tree(t -> right);
}
```

Сбалансированные деревья

Дерево называется сбалансированным, если высоты двух поддеревьев каждой из его вершин отличаются не более, чем на единицу.

АВЛ-дерево — сбалансированное двоичное дерево поиска.

(Адельсон-Вельский, Ландис, 1962 г.)

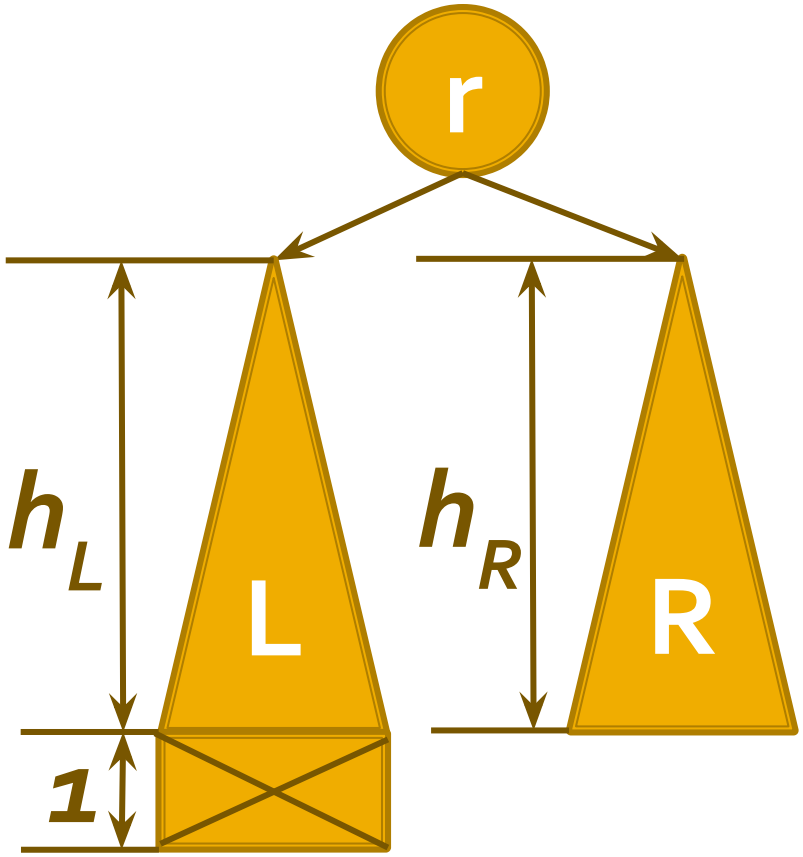
Теорема

Среднее число сравнений, необходимых для вставки n случайных элементов в дерево двоичного поиска, пустое в начале, равно $O(n \log_2 n)$ для $n \geq 1$.

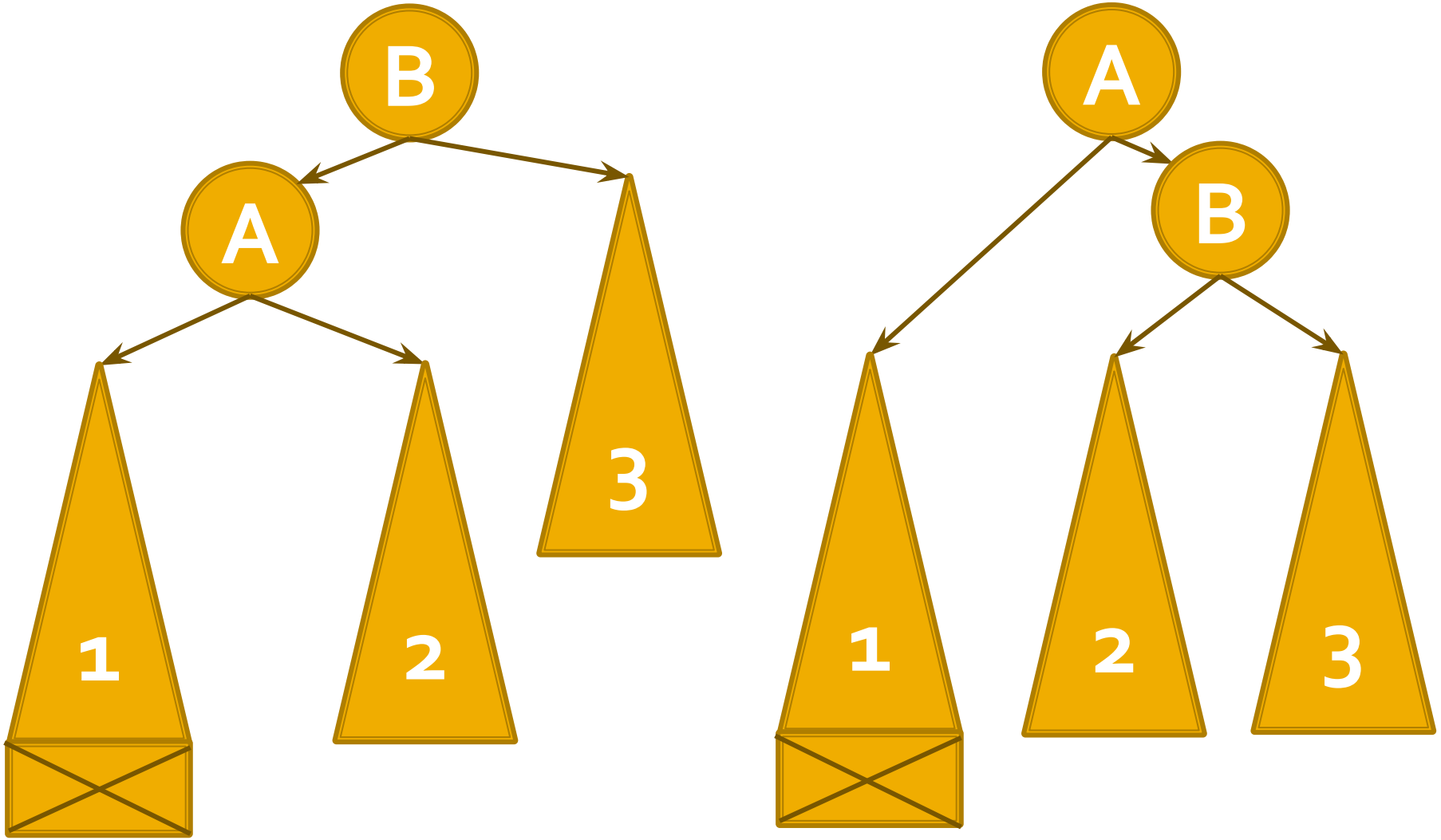
Максимальное число сравнений — $O(n^2)$.

Вставка элемента в сбалансированное дерево

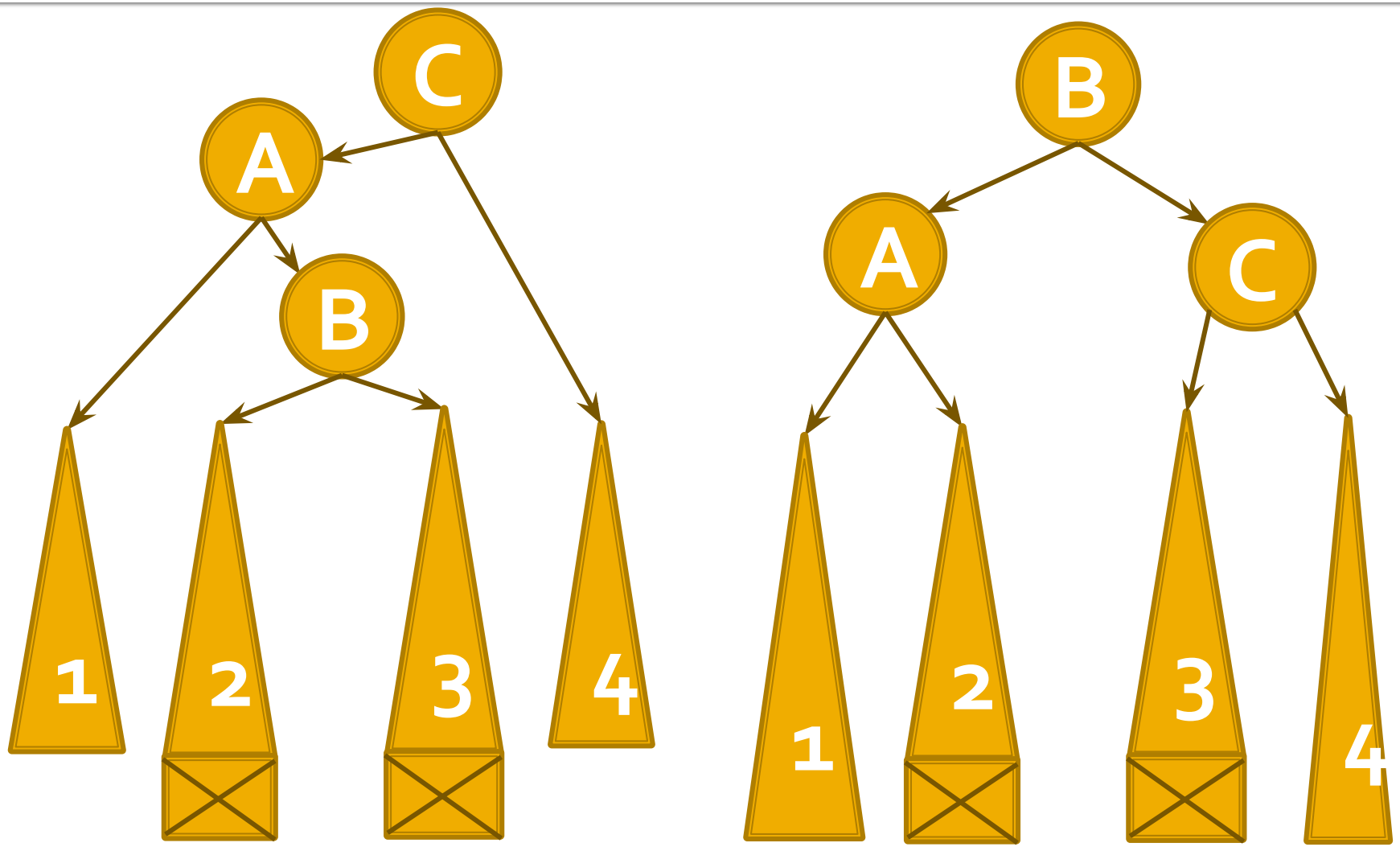
1. $h_L = h_R$
2. $h_L < h_R$
3. $h_L > h_R$



Вставка в левое поддерево

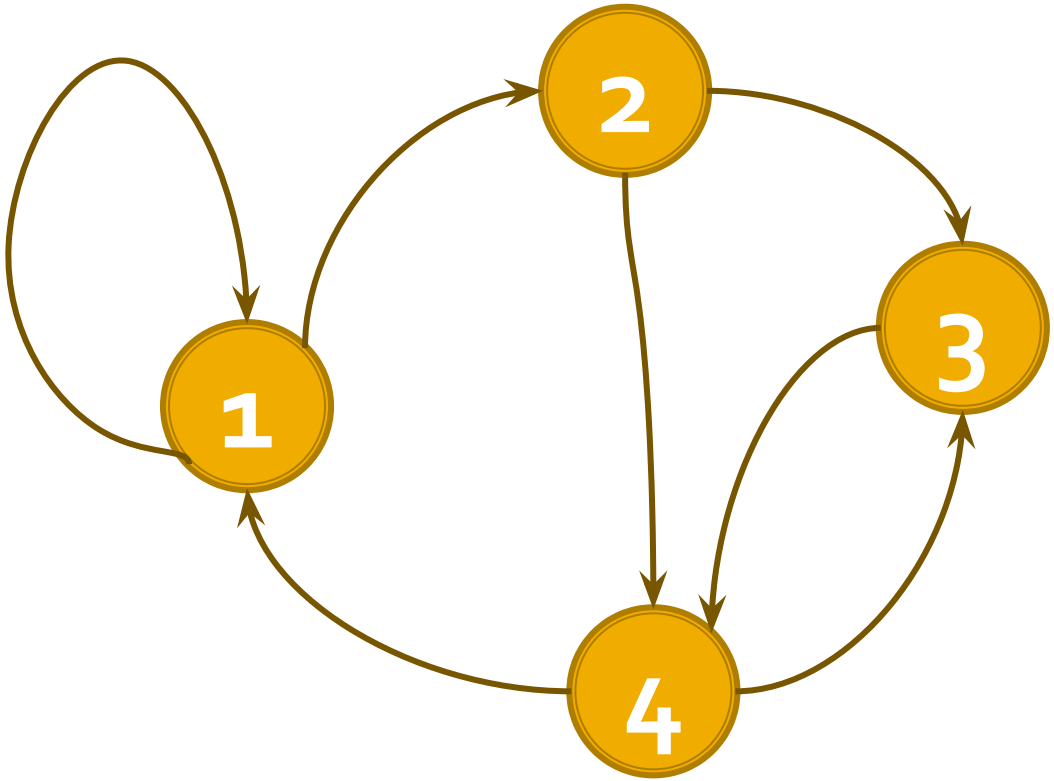


Вставка в правое поддерево



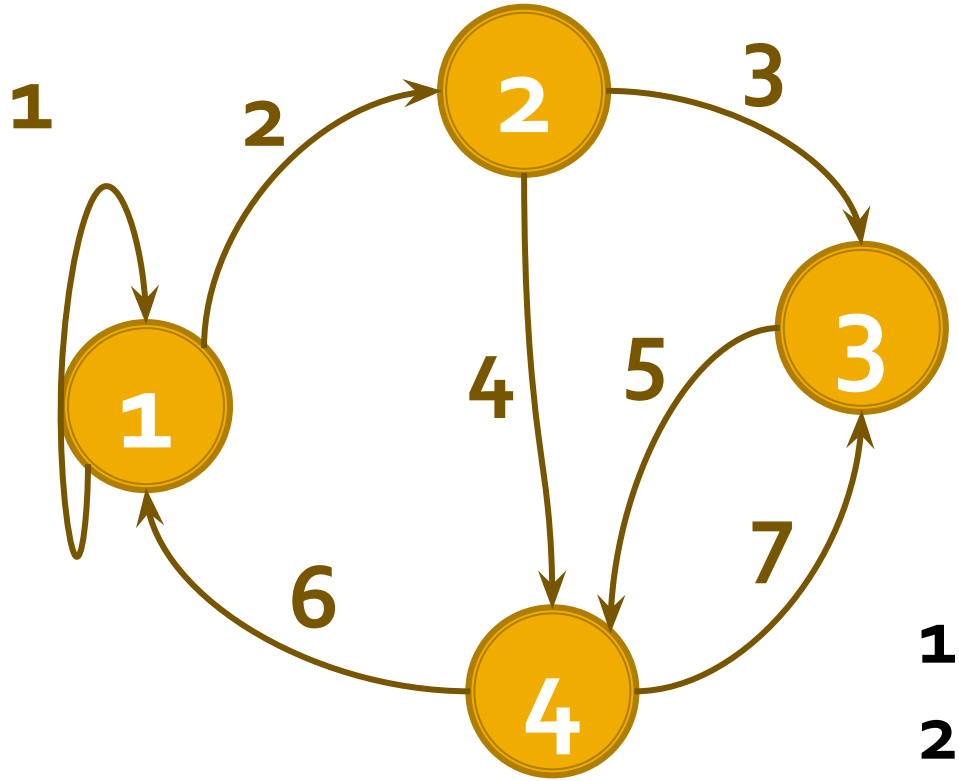
Представление графов

Матрица смежностей



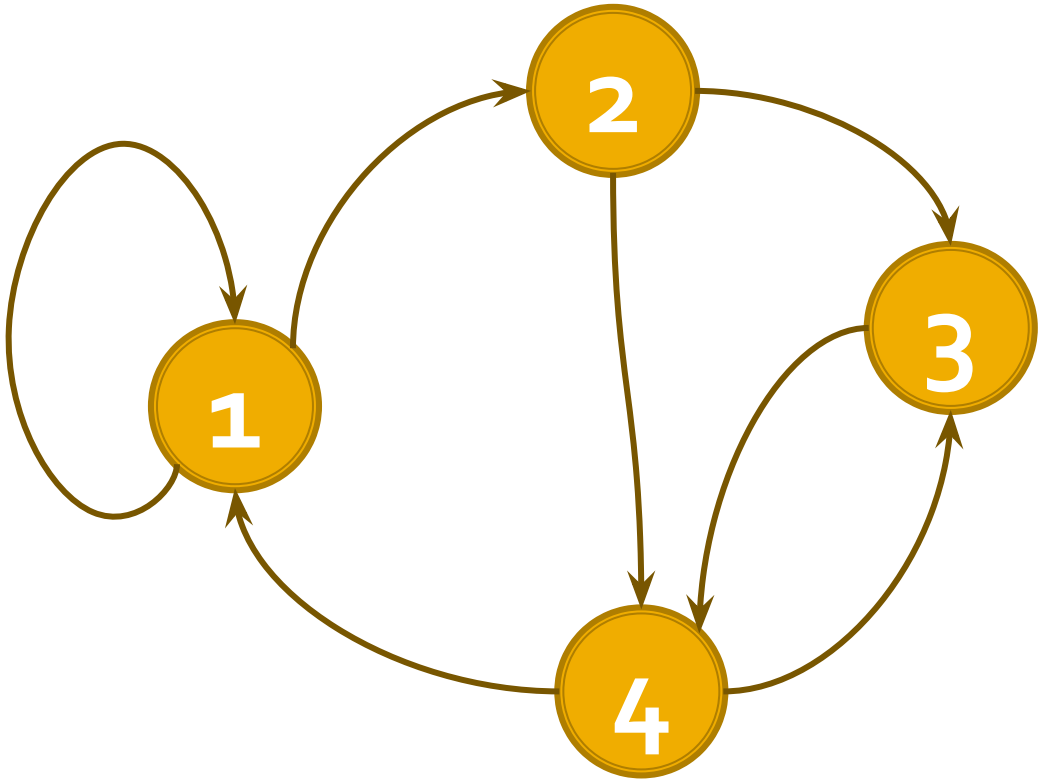
	1	2	3	4
1	1	1	0	0
2	0	0	1	1
3	0	0	0	1
4	1	0	1	0

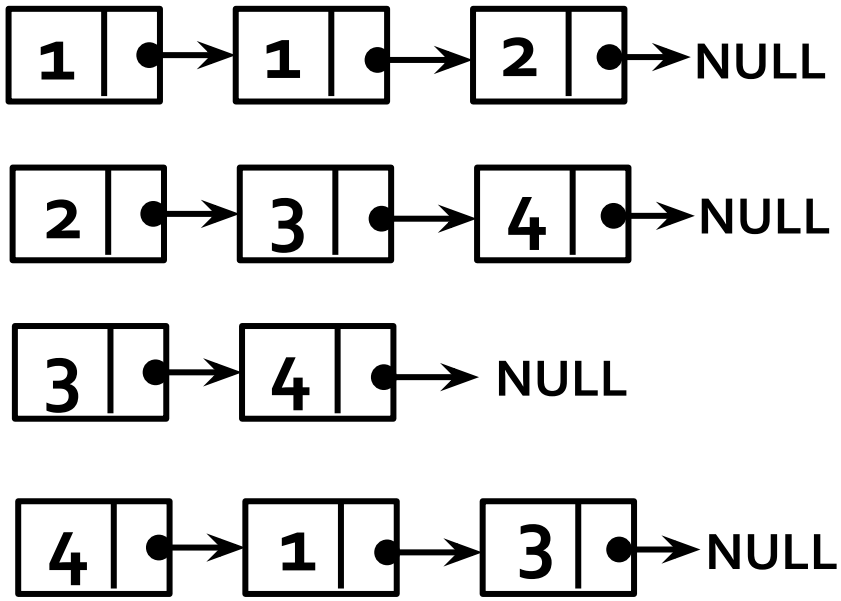
Матрица инцидентностей



	1	2	3	4	5	6	7
1	1	1	0	0	0	-1	0
2	0	-1	1	1	0	0	0
3	0	0	-1	0	1	0	-1
4	0	0	0	-1	-1	1	1

Списки смежностей





Частичный порядок

Определение?

Частичный порядок

Отношение R на множестве A:

$\forall a \in A \neg aRa$;

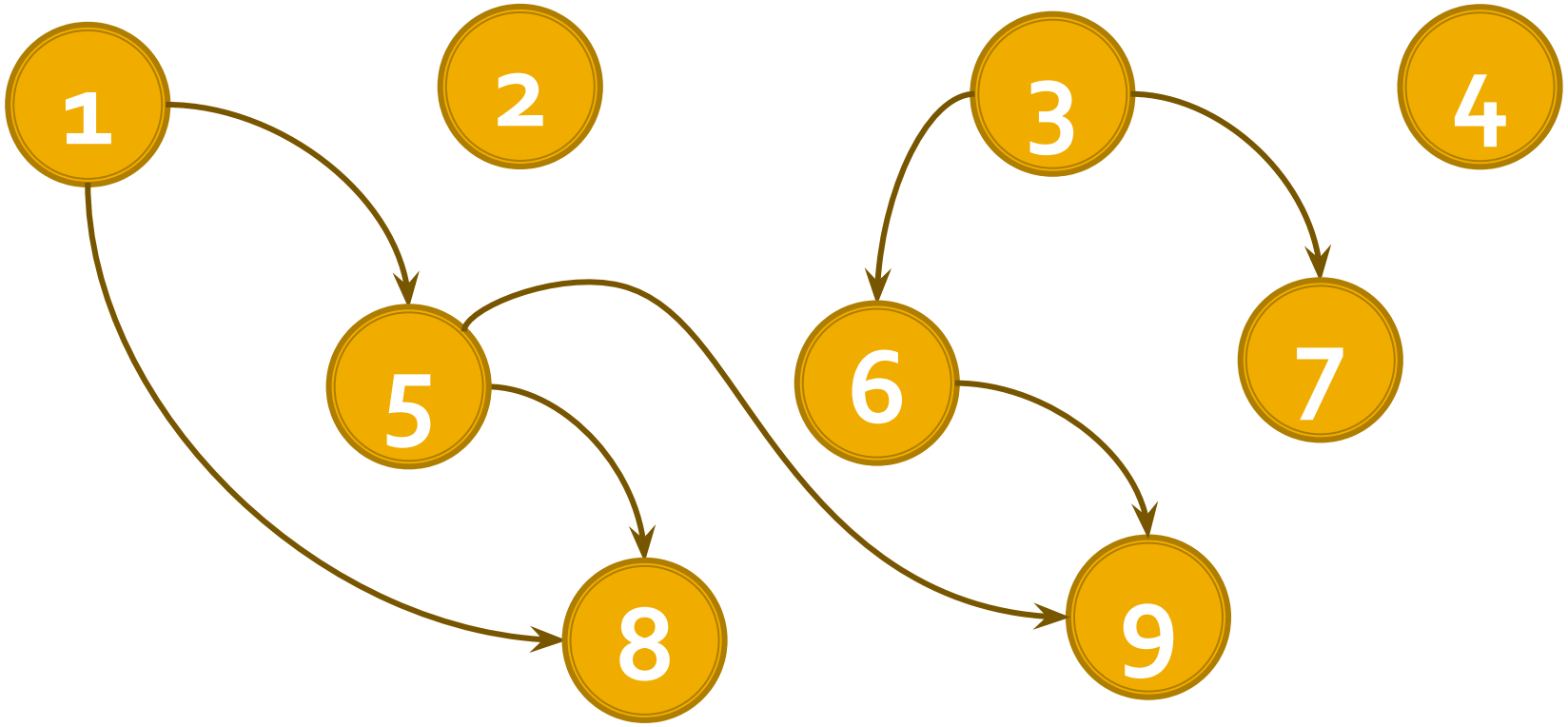
$\forall a, b, c \in A\ aRb \,\&\, bRc \Rightarrow aRc$.

Следствие: $\forall a, b \in A\ aRb \Rightarrow \neg bRa$.

Частичный порядок: примеры

- решение большой задачи разбивается на ряд подзадач;
- последовательность чтения тем в учебном курсе;
- выполнение работ: одну следует выполнить раньше другой;
- и т. п.

Пример



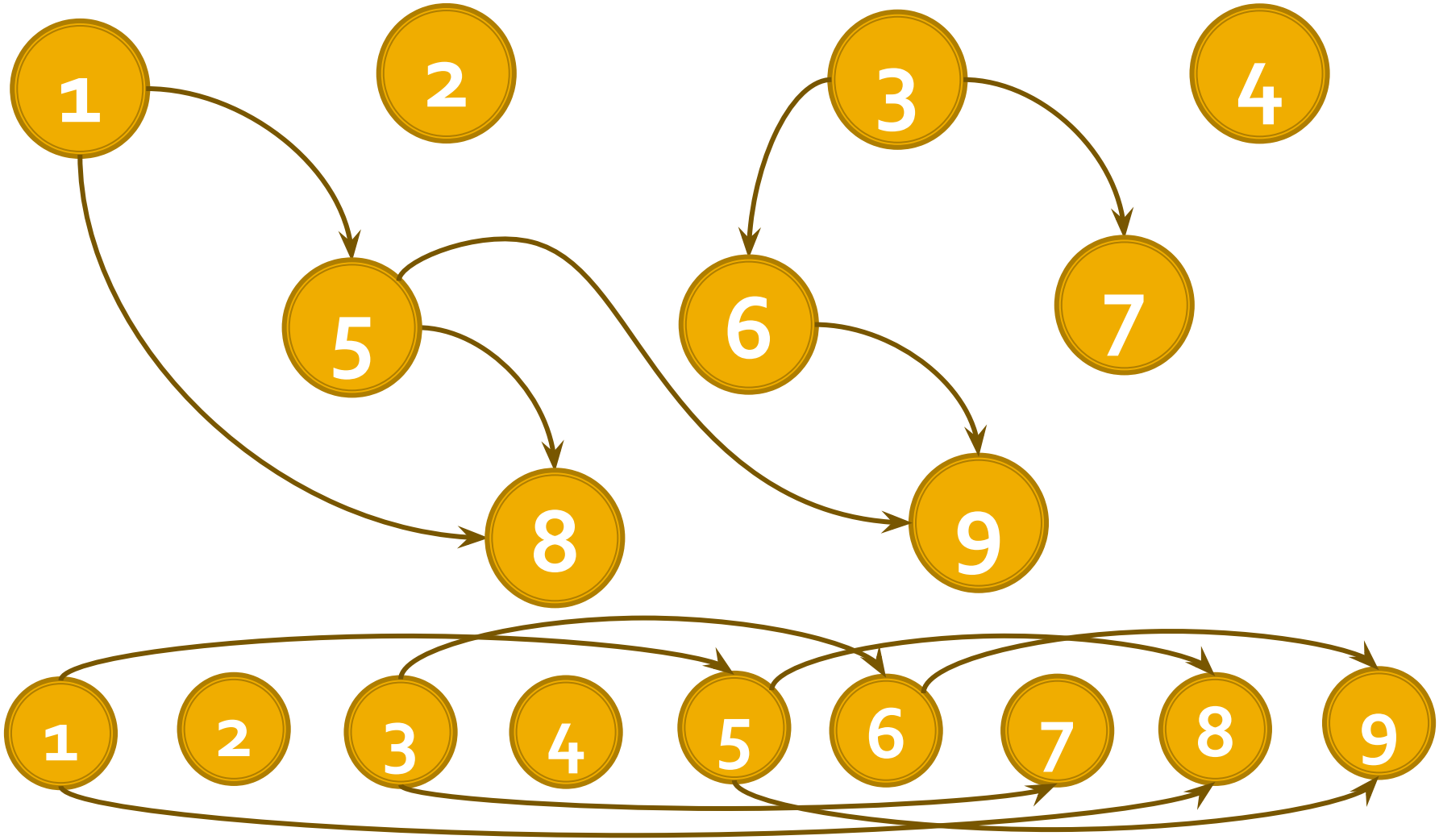
Линейный порядок

Определение?

Линейный порядок

$\forall a, b \in A$ aRb либо bRa либо $a = b$.

Пример



Алгоритм топологической сортировки

Вход: частичный порядок R на конечном множестве $A = \{a_1, a_2, \ldots, a_n\}$.

Выход: линейный порядок R' на A: $R \subseteq R'$.

Метод: представить R' в виде списка $L_n = a_1, a_2, \ldots, a_n$, для которого $a_i R' a_j$ при $i < j$.

Шаг 1: положить $i = 1$, $A_1 = A$, $R_1 = R$.

Шаг 2: если A_i пусто, то остановиться и выдать $L_i = a_1, a_2, \ldots, a_i$ в качестве R'.

Иначе выбрать в A_i такой a_{i+1}, что $\forall a' \in A \ \neg a' R a_{i+1}$.

Шаг 3: положить $A_{i+1} = A_i \setminus \{a_{i+1}\}$, $R_{i+1} = R_i \setminus (\{a_{i+1}\} \times A_{i+1})$, i++, на **шаг 2**.

Алгоритм топологической сортировки

Реализация на матрице смежности:

1. Найти вершину, в которую не входит ни одна дуга (нулевой столбец).

2. Удалить все исходящие из неё дуги (обнулить соответствующую строку).

3. Пока не перебрали все вершины, повторять сначала.

Пути в графах

Определения

Пусть $G = (V, E)$ — ориентированный граф.
Поставим в соответствие каждой дуге $e \in E$ в графе G неотрицательную стоимость $c(e)$.

$c: E \rightarrow R^+$ — функция стоимости.

Стоимость пути определяется как сумма стоимостей образующих его дуг.

Задача о нахождении кратчайшего пути состоит в нахождении для каждой пары узлов (v, w) пути наименьшей стоимости.

Нахождение кратчайшего пути из одного источника

Идея: строится множество *S*, содержащее вершины графа, кратчайшие расстояния до которых от источника известны. На каждом шаге добавляется тот из оставшихся узлов, кратчайшее расстояние до которого меньше всех других оставшихся узлов (т. н. "жадный" алгоритм).

Алгоритм Дейкстры

Вход: ориентированный граф $G = (V, E)$,
источник $v_0 \in V$,
функция стоимости $c: E \to R^+$.
Полагаем, что
$c(v_i, v_j) = +\infty$, если $(v_i, v_j) \notin E$;
$c(v, v) = 0$.

Выход: для всех вершин $v \in V$ наименьшая сумма стоимостей дуг из E, взятая по всем путям, идущим из v_0 в v.

Алгоритм Дейкстры

Метод: строим такое множество $S \subseteq V$, что кратчайший путь из источника в каждый узел $v \in S$ целиком лежит в S.

Массив $D[v]$ содержит стоимость текущего кратчайшего пути из v_0 в v.

Алгоритм Дейкстры

{

$S = \{v_o\}$;

$D[v_o] = 0$;

для всех $v \in V \setminus \{v_o\}$ выполнить: $D[v] = c(v_o, v)$;

пока $S \neq V$ выполнять:

{

выбрать узел $w \in V \setminus S$, для которого $D[w]$ принимает наименьшее значение;

$S = S \cup \{w\}$;

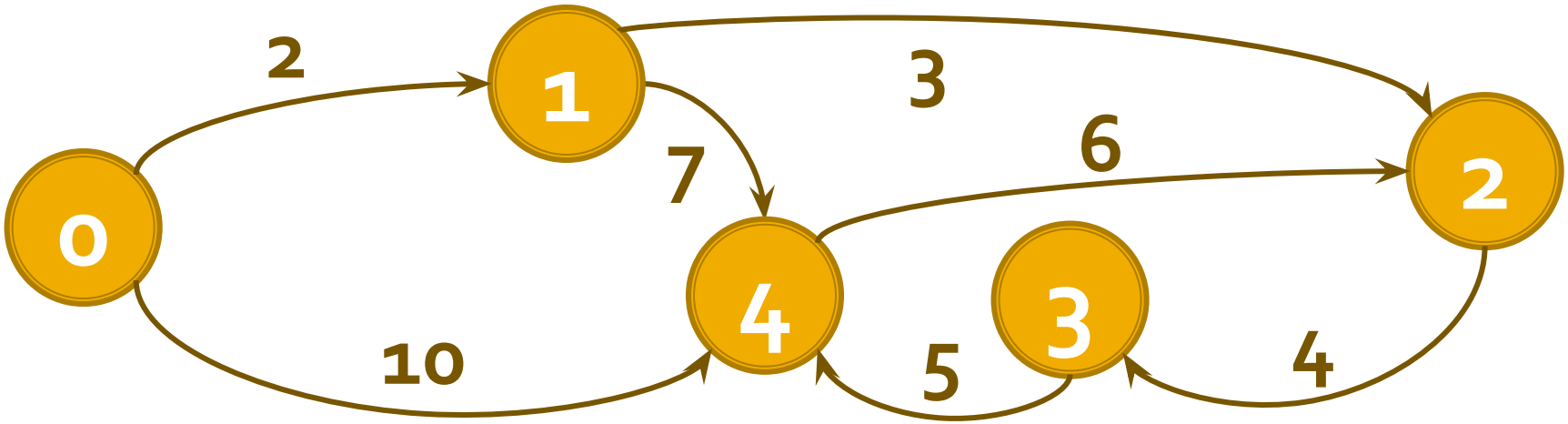
для всех $v \in V \setminus S$ выполнить

$D[v] = min(D[v], D[w] + c(w, v))$;

}

}

Пример



S	w	D[w]	D[1]	D[2]	D[3]	D[4]
{0}	-	-	2	+∞	+∞	10
{0, 1}	1	2	2	5	+∞	9
{0, 1, 2}	2	5	2	5	9	9
{0, 1, 2, 3}	3	9	2	5	9	9
{0, 1, 2, 3, 4}	4	9	2	5	9	9

Нахождение кратчайших путей между всеми парами вершин

Строим матрицу стоимостей:

$$M[i,j] = \begin{cases} c(i,j), & \text{если дуга } (i,j) \in E; \\ +\infty, & \text{если дуга } (i,j) \notin E; \\ 0, & \text{если } i = j. \end{cases}$$

Обозначим через $d[i,j]$ матрицу кратчайших путей между всеми вершинами.

Вершины занумеруем числами от 1 до n.

Алгоритм Флойда-Уоршелла

Обозначим через $d_{ij}^{(k)}$ стоимость кратчайшего пути из вершины с номером i в вершину с номером j с промежуточными вершинами из множества $\{1, 2, \ldots, k\}$.

$$d_{ij}^{(k)} = \begin{cases} M[i, j], \text{ если } k = 0, \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}), \text{ если } k \geq 1. \end{cases}$$

$D^{(n)}$ содержит искомое решение.

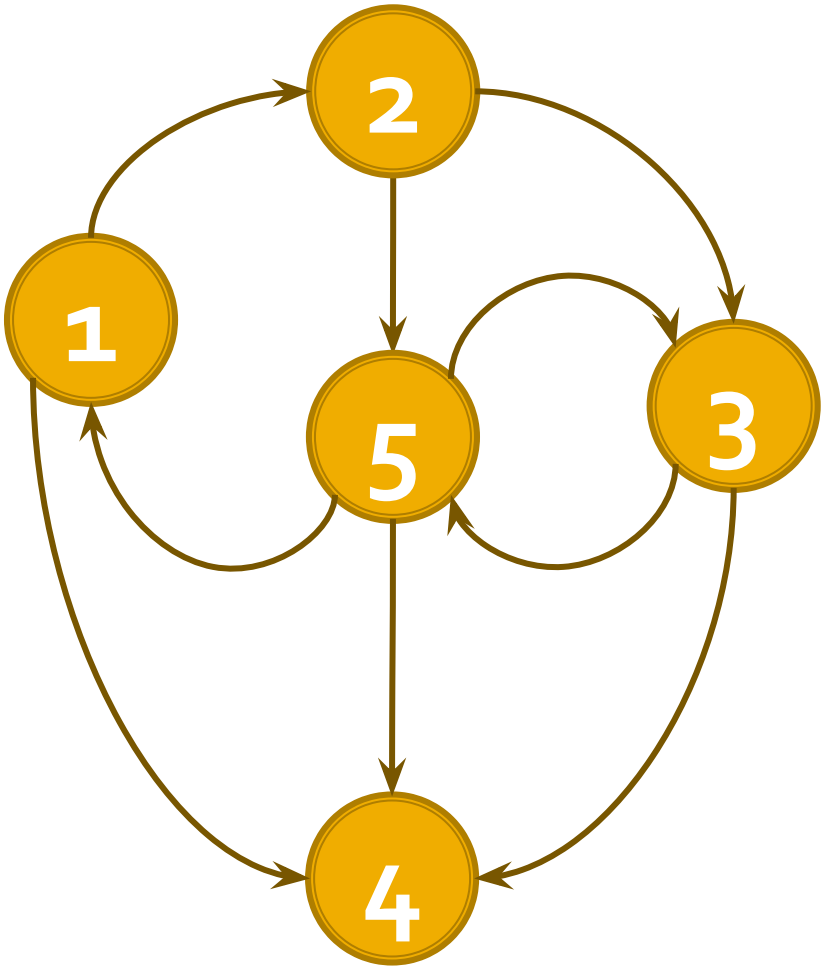
Алгоритм Флойда-Уоршелла

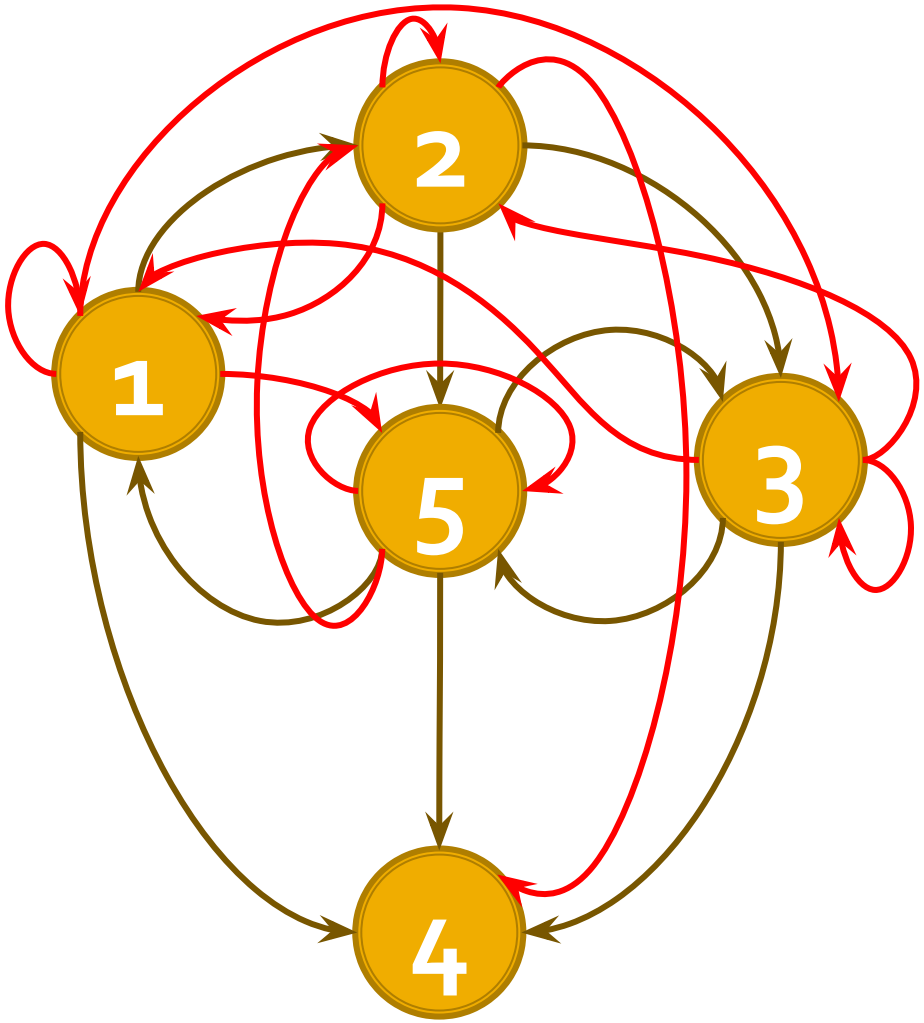
```
Floyd-Warshall(M, n)
{
    D(0) = M;
    for k = 1 to n do
        for i = 1 to n do
            for j = 1 to n do
                d_ij^(k) = min(d_ij^(k - 1), d_ik^(k - 1) + d_kj^(k - 1));
    return D(n);
}
```

Транзитивное замыкание графа

Транзитивное замыкание орграфа $G = (V, E)$ — орграф $G' = (V, E')$, в котором $(v, w) \in E' \Leftrightarrow$ существует путь (длины 0 или больше) из v в w в G.

Пример





Построение транзитивного замыкания графа

Обозначим через $t_{ij}^{(k)}$ наличие пути из вершины с номером i в вершину с номером j с промежуточными вершинами из множества $\{1, 2, \ldots, k\}$. M — матрица смежностей G.

$$t_{ij}^{(k)} = \begin{cases} M[i, j], \text{ если } k = 0, \\ t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \,\&\, t_{kj}^{(k-1)}), \text{ если } k \geq 1. \end{cases}$$

$T^{(n)}$ содержит искомое решение.

Построение транзитивного замыкания графа

```
Transitive_Closure(M, n)
{
    T(0) = M;
    for k = 1 to n do
        for i = 1 to n do
            for j = 1 to n do
                t_ij^(k) = t_ij^(k - 1) V (t_ik^(k - 1) & t_kj^(k - 1));
    return T(n);
}
```