ФУНДАМЕНТАЛЬНЫЕ АЛГОРИТМЫ. АЛГОРИТМЫ НА ГРАФАХ

ПЛАН ЛЕКЦИИ

- Обозначения и определения
- Машинное представление графов.
- Алгоритмы поиска в глубину и ширину.
- Алгоритмы построения остовых деревьев
- Фундаментальное множество циклов
- Эйлеровы пути на графах
- Гамильтоновы пути и циклы
- Кратчайшие пути на графах

1. Обозначения и определения

Граф G = <V, E>

V – множество вершин графа;

E – множество рёбер (дуг):

$E \subseteq V \otimes V$, – для дуг ориентированного графа;

$E \subseteq \langle \{x, y\}: x, y \in V \wedge x \neq y \rangle$ – для рёбер неориентированного графа;

$|V| = n$ – мощность множества вершин;

$|E| = m$ – мощность множества рёбер (дуг).

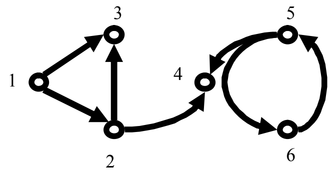
Дерево – произвольный неориентированный связный граф без циклов.

Суграф G_S ***графа*** $G = \langle V, E \rangle$ есть граф, содержащий все вершины V исходного графа.

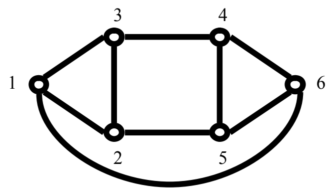
Суграф G_S графа $G = \langle V, E \rangle$ ***являющийся деревом***, называется ***остовым деревом*** (остовом, каркасом, стягивающим деревом)

2. МАШИННОЕ ПРЕДСТАВЛЕНИЕ ГРАФОВ (1)

Ориентированный и неориентированный графы



1)



2)

Соответствующие матрицы инциденций

	<1,2>	<1,3>	<3,2>	<3,4>	<5,4>	<5,6>	<6,5>
1	−1	−1	0	0	0	0	0
2	1	0	1	0	0	0	0
3	0	1	−1	−1	0	0	0
4	0	0	0	1	1	0	0
5	0	0	0	0	−1	−1	1
6	0	0	0	0	0	1	−1

	{1,2}	{1,3}	{1,5}	{2,3}	{2,5}	{3,4}	{4,5}	{4,6}	{5,6}
1	1	1	1	0	0	0	0	0	0
2	1	0	0	1	1	0	0	0	0
3	0	1	0	1	0	1	0	0	0
4	0	0	0	0	0	1	1	1	0
5	0	0	1	0	1	0	1	0	1
6	0	0	0	0	0	0	0	1	1

2. МАШИННОЕ ПРЕДСТАВЛЕНИЕ ГРАФОВ (2)

Матрицы смежности:

	1	2	3	4	5	6
1	0	1	1	0	0	0
2	0	0	0	0	0	0
3	0	1	0	1	0	0
4	0	0	0	0	0	0
5	0	0	0	1	0	1
6	0	0	0	0	1	0

1)

	1	2	3	4	5	6
1	0	1	1	0	1	0
2	1	0	1	0	1	0
3	1	1	0	1	0	0
4	0	0	1	0	1	1
5	1	1	0	1	0	1
6	0	0	0	1	1	0

2)

Пусть P – некоторое свойство графа G, $P(G) = 0$ или $P(G) = 1$, в зависимости от того, обладает ли граф G этим свойством.

Свойство удовлетворяет условиям:

1. $P(G) = P(G^{\nabla})$, если графы G и G^{∇} изоморфны;
2. $P(G) = 0$ для произвольного пустого графа $<V, \varnothing>$ и $P(G) = 1$ для произвольного полного графа $<V, P_2(V)>$ с достаточно большим числом вершин;
3. Добавление ребра не нарушает свойства P, т.е. $P(G) \le P(G^{\nabla})$ для произвольных графов $G = <V, E>$ и $G^{\nabla} = <V, E^{\nabla}>$ таких, что $E = E^{\nabla}$.

2. МАШИННОЕ ПРЕДСТАВЛЕНИЕ ГРАФОВ (3)

4. $P(G) = P(G^{\nabla})$ для произвольно ориентированных графов $G = <V, E>$ и $G^{\nabla} = <V, E \cup <v, v>>$, $v \in V$ (случай наличия петель на вершинах).

Теорема 1. Если P – свойство графа, отвечающее условиям 1 – 3, то каждый алгоритм, проверяющий свойство P (сиречь, вычисляющий значение $P(G)$ для данного графа G) на основе матрицы смежности, выполняет в худшем случае $\Omega(n^2)$ шагов, где n – число вершин графа.

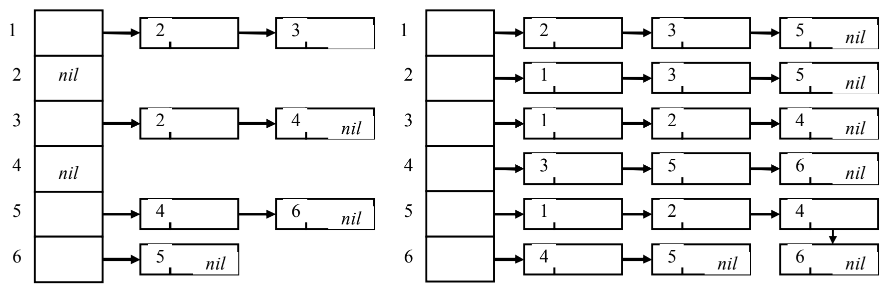
Список рёбер(дуг)

$$\begin{bmatrix} 1 & 2 \\ 1 & 3 \\ 3 & 2 \\ 3 & 4 \\ 5 & 4 \\ 5 & 6 \\ 6 & 5 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 5 \\ 2 & 3 \\ 2 & 5 \\ 3 & 4 \\ 4 & 5 \\ 4 & 6 \\ 5 & 6 \end{bmatrix}$$

(1) (2)

2. МАШИННОЕ ПРЕДСТАВЛЕНИЕ ГРАФОВ (4)

Списки инцидентности



1) 2)

3. АЛГОРИТМЫ ПОИСКА В ГЛУБИНУ И ШИРИНУ (1)

Поиск в глубину (Depth first search)
Tarjan R.E. Depth first search and linear graph algorithms. SIAM J. Comput., 1972, 1, c. 146 – 160.
Гипотеза: список инцидентности *Spisok_In* упорядочен по номерам вершин

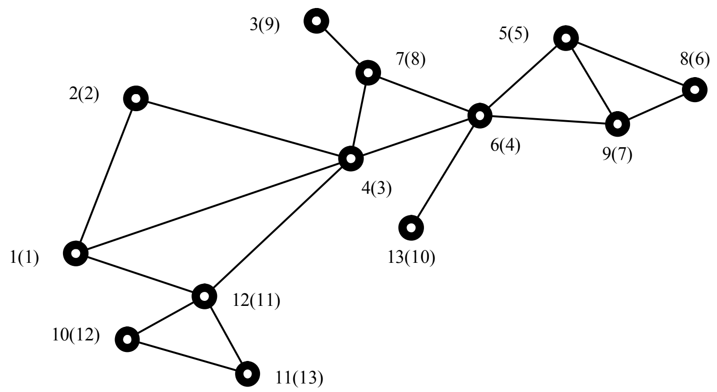
```
procedure WG(v)
(* Поиск из вершины v; переменные New_punkt, Spisok_In – глобальные *)
begin рассмотреть v; New_punkt[v] := false;
    for u ∈ Spisok_In[v] do
        if New_punkt[u] then WG(v);
    end (* Вершина v использована *)

begin
    for v ∈ V do New_punkt[v] := true; (* Инициализация *)
    for v ∈ V do
        if New_punkt[v] then WG(v);
end
```

3. АЛГОРИТМЫ ПОИСКА В ГЛУБИНУ И ШИРИНУ (2)

```
procedure WG_nr(v) (* Нерекурсивная версия алгоритма *)
 (* P[u] – указатель на первую запись списка инцидентности, New_punkt, Spisok_In, Stek –
    глобальные *)
begin (* очистить стек *)
Stek := ∅;
Stek ⇐ v; рассмотреть v; New_punkt[v] := false;
while Stek ≠ ∅ do
    begin
        t := top(Stek) (* t – верхний элемент стека *)
        if P[t] = nil then b := false;
                else b := not New_punkt[P[t]↑.uzel];
        while b do begin
                P[t] := next P[t]↑; Stek ⇐ v;
                if P[t] = nil then b := false;
                        else b := not New_punkt[P[t]↑.uzel];
                end
                if P[t] ≠ nil then (* найдена новая вершина *)
                                        begin
                                        t := P[t]↑.uzel;
                                        Stek ⇐ t;
                                        рассмотреть t; New_punkt[t] := false;
                                        end
end
```

3. АЛГОРИТМЫ ПОИСКА В ГЛУБИНУ И ШИРИНУ (3)

Обход графа в ходе работы алгоритма в глубину



3. АЛГОРИТМЫ ПОИСКА В ГЛУБИНУ И ШИРИНУ (4)

Алгоритм поиска в ширину (Breadth first search)

procedure *WS* (*v*) (* Поиск в ширину с началом в вершине *v* *)
(* *New_punkt*, *Spisok_In* – глобальные *)
begin (* очистить стек *)
Queue := ∅;
Queue ⇐ *v*; *New_punkt*[*v*] := ***false***;
while *Queue* ≠ ∅ ***do***
    begin
    p ⇐ *Queue*; Посетить *p*;
    for *u* ∈ *Spisok_In*[*p*] ***do***
        if *New_punkt[u]* ***then***
        begin
        Queue ⇐ *u;*
        New_punkt[u] := ***false;***
        end
    end
end

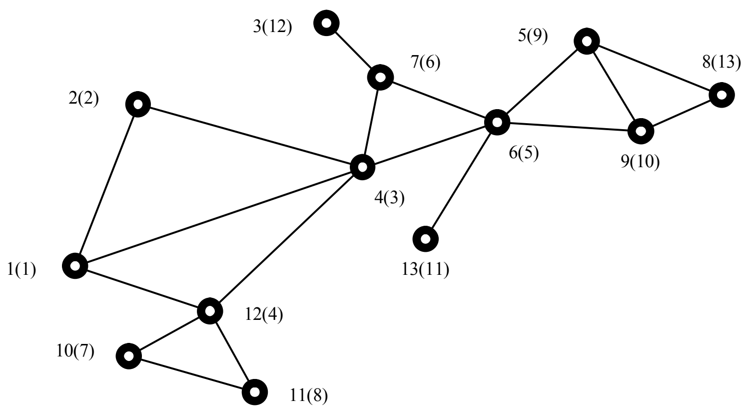
3. АЛГОРИТМЫ ПОИСКА В ГЛУБИНУ И ШИРИНУ (5)

Модифицированный алгоритм поиска в ширину

```
procedure WSm(v) (* Поиск в ширину с началом в вершине v *)
 (* New_punkt, Prefix_punkt[u], Spisok_In – глобальные *)
begin (* очистить стек *)
Queue := ∅;
Queue ⇐ v; New_punkt[v] := false;
while Queue ≠ ∅ do
    begin
    p ⇐ Queue; Посетить p;
    for u ∈ Spisok_In[p] do
            if New_punkt[u] then
            begin
            Queue ⇐ u;
            New_punkt[u] := false;
            Prefix_punkt[u] := p
            end
    end
end
```

АЛГОРИТМЫ ПОИСКА В ГЛУБИНУ И ШИРИНУ (6)

Обход графа в ходе работы алгоритма в глубину



4. АЛГОРИТМЫ ПОСТРОЕНИЯ ОСТОВЫХ ДЕРЕВЬЕВ (1)

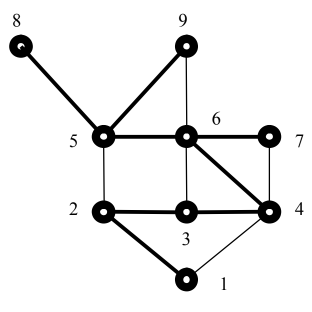
Теорема. Пусть $<V, T>$ – стягивающее (остовое) дерево связного графа $G = <V, E>$, построенное алгоритмом WGD и пусть $\{u, v\} \in E$. Тогда либо u – потомок v, либо v – потомок u.

```
(* Нахождение остова связного графа на базе поиска в глубину *)
(* Переменные New_punkt, Spisok_In, T – глобальные *)
procedure WGD(v)
 begin New_punkt[v] := false;
   for u ∈ Spisok_In[v] do
        if New_punkt[u] then begin (* {v, u} новая ветвь *)
                                    T := T ∪ {v, u};
                                    WGD(u);
                                    end
 end (*WGD *)
```

4. АЛГОРИТМЫ ПОСТРОЕНИЯ ОСТОВЫХ ДЕРЕВЬЕВ (2)

```
begin (* Главная программа *)
   for u ∈ V do New_punkt[u] := true; (* Инициализация *)
   T := ∅; (*T – множество найденных к этому шагу алгоритма ветвей *)
   WGR(r); (*r – произвольная вершина графа *)
end
```

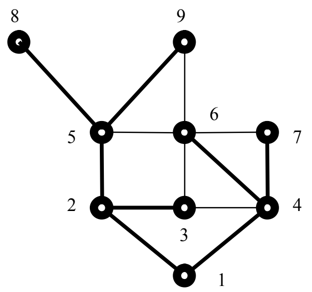
Результат работы алгоритма “в глубину”



4. АЛГОРИТМЫ ПОСТРОЕНИЯ ОСТОВЫХ ДЕРЕВЬЕВ (3)

Теорема. Пусть $<V, T>$ – стягивающее (остовое) дерево связного графа $G = <V, E>$, построенное при помощи алгоритма. Тогда путь в $<V, T>$ из произвольной вершины v до корня r является кратчайшим путём из v в r в графе G.

Результат работы алгоритма “в ширину”

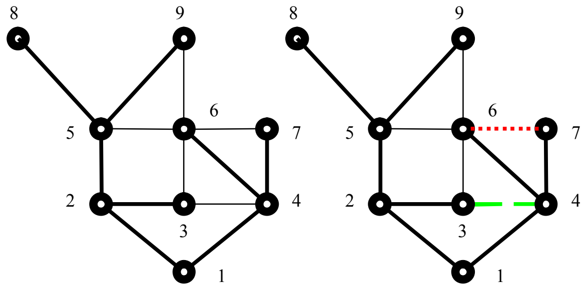


4. АЛГОРИТМЫ ПОСТРОЕНИЯ ОСТОВЫХ ДЕРЕВЬЕВ (4)

```
(* Вход: связный граф G , заданный списком инцендентности *)
 (* New_punkt, Spisok_In – глобальные, Выход: суграф G_S *)
begin
for u ∈ V do New_punkt[u] := true; (* инициализация *)
T := ∅; (*T множество найденных на текущий момент ветвей *)
Queue := ∅; Queue ⇐ r; New_punkt[r] := false; (*r корень дерева *)
while Queue ≠ ∅ do
    begin
    v ⇐ Queue;
    for u ∈ Spisok_In[v] do
        if New_punkt[u] then (* {v, u} – новая ветвь *)
        begin
        Queue ⇐ u;
        New_punkt[u] := false;
        T := T := ∪ {v, u};
        end
    end
end
```

5. ФУНДАМЕНТАЛЬНОЕ МНОЖЕСТВО ЦИКЛОВ (1)

Фундаментальный цикл графа $G = \langle V, E \rangle$ **относительно остова** — простой цикл C, полученный путем добавления к остову ребра $v_i v_j$, не принадлежащего к этому остову.



5.ФУНДАМЕНТАЛЬНОЕ МНОЖЕСТВО ЦИКЛОВ (2)

```
procedure CIKL(v) (* Нахождение множества фундаментальных циклов *)
(* Переменные Stek, Spisok_In, WGN,d, num – глобальные *)
begin
d := d + 1; Stek[d] := v; num := num + 1;
WGN[v] := num;
   for u ∈ Spisok_In[v] do
        if WGN[u] == 0 then CIKL(u)
        else if (u ≠ Stek[d – 1]) and (WGN[v] > WGN[u])
                then begin (* {v, u} замыкает новый цикл *)
                                Выписать последовательность вершин:
                                Stek[d], Stek[d – 1], ..., Stek[u]
                     end
d := d – 1; (*вершина v удаляется из стека *)
end
begin (* главная программа *)
for v ∈ V do WGN[v] := 0; num := 0; (* инициализация *)
d := 0; Stek[0] := 0; (* d счётчик элементов стека *)
for v ∈ V do
   if WGN[v] == 0 then CIKL(v)
end
```

6. ЭЙЛЕРОВЫ ПУТИ НА ГРАФАХ (1)

Эйлеровым путём в графе называется произвольный путь, проходящий через каждое ребро графа в точности один раз. Каждое ребро $e \in E$ появится в последовательности $v_1, v_2, \ldots, v_{m+1}$ ровно один раз как $e = \{v_i, v_{i+1}\}$ для некоторого i. Если $v_1 = v_{m+1}$, то такой путь называется эйлеровым циклом.
Задача существования решена Л.Эйлером в 1736 г.

Теорема. Эйлеров путь в графе существует тогда и только тогда, когда граф связный и содержит не более чем две вершины нечётной степени.

Если вершина v, отличная от v_1 и v_{m+1}, появляется в эйлеровом пути $v_1, v_2, \ldots, v_{m+1}$ ровно k раз, это означает, что степень этой вершине в графе составляет 2^k. Отсюда следует, что вершины нечётной степени, паче такие найдутся, являются концами эйлерова пути. Пусть $d(v)$ – степень вершины v. Каждая вершина на ребре $\{u, v\}$ подсчитывается два раза. Поэтому число вершин нечётной степени является чётным.

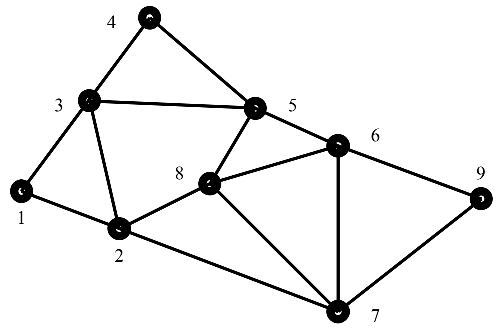
$$\sum_{v \in V} d(v) = 2m$$

6. ЭЙЛЕРОВЫ ПУТИ НА ГРАФАХ (2)

```
begin
Stek := ∅; EilerCircl := ∅;
v := u; (* произвольная вершина графа *)
Stek ⇐ v;
while Stek ≠ ∅ do
   begin
   v := top(Stek); (*v верхний элемент стека *)
   if Spisok_In[v] ≠ ∅ then
        begin
              u := first(Spisok_In[v]);
              Stek ⇐ u;
              (* удаление ребра {v, u} из графа *)
              Spisok_In[v] := Spisok_In[v] \ {u};
              Spisok_In[u] := Spisok_In[u] \ {v};
              v := u
        end
```

6. ЭЙЛЕРОВЫ ПУТИ НА ГРАФАХ (3)

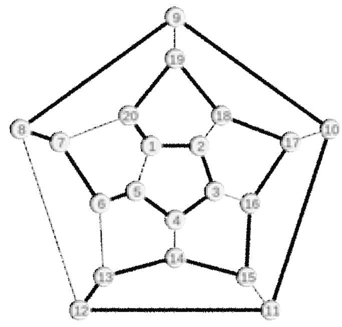
```
    else begin
                v ⇐ Stek;
                EilerCircl ⇐ v
         end
    end
end
```



Цикл: 1, 2, 3, 4, 5 , 6, 2, 8, 6, 9, 7, 8, 5, 3, 1

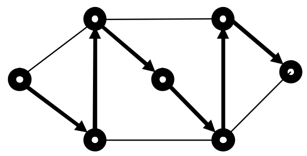
7. ГАМИЛЬТОНОВЫ ПУТИ И ЦИКЛЫ (1)

Название “гамильтонов цикл” возникло как следствие решения задачи “Кругосветное путешествие” в постановке математика Вильяма Гамильтона в 1859 году. По условию задачи требуется, выйдя из исходной вершины графа, обойти все его вершины и вернуться в исходную точку. Маршруты интерпретировались как проекция додекаэдра, при этом каждой вершине графа соответствовало название всемирно известного города.

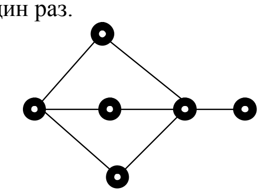


Гамильтоновы пути и гамильтоновы циклы (2)

Гамильтоновым путём (*Hamiltonian path*) называется простой путь, проходящий через каждую вершину графа ровно один раз.



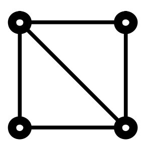
Существует гамильтонов путь



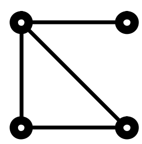
Не существует пути

Гамильтоновым циклом (*Hamiltonian cycle*) называют замкнутый гамильтонов путь.
Граф называется **полугамильтоновым** (*Semihamiltonian graph*), когда он содержит гамильтонов путь.
Граф называется **гамильтоновым** (*Hamiltonian graph*), если он содержит гамильтонов цикл.
Эти определение можно распространить также и на ориентированные графы, если пути считать ориентированными.

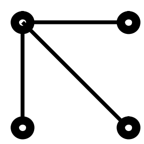
7. ГАМИЛЬТОНОВЫ ПУТИ И ЦИКЛЫ (3)



Гамильтонов граф



Полугамильтонов граф



Негамильтонов граф

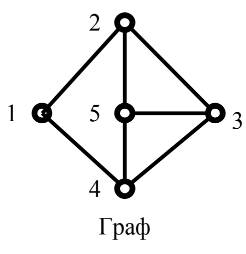
(* Нахождение всех гамильтоновых циклов графа *)
(*Вход: связный граф G , представленный инцендентности *Spisok_In*[*v*] *)
(* *X*, *Spisok_In* – глобальные, Выход: гамильтоновы циклы *)
procedure *HAMILT*(*k*)
(* Построение циклов, расширением ряда <*X*[1], *X*[2], ... , *X*[*k* – 1]> *)
begin
    for $y \in$ *Spisok_In*[*X*[*k* – 1]] ***do***
        if (*k* == *n* + 1) ***and*** (y == v_0) ***then***
            (* Вывод последовательности <*X*[1],... , *X*[*n*], v_0> *)

7. ГАМИЛЬТОНОВЫ ПУТИ И ЦИКЛЫ (4)

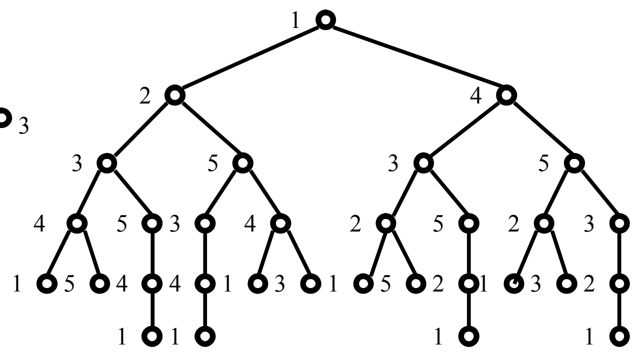
```
        else if DOP[y]
                then begin
                        X[k] := y;
                        DOP[y] := false;
                        HAMILT(k + 1)
                        DOP[y] := true
                     end
end
begin (* главная программа *)
for v ∈ V do DOP[v] := true (* инициализация *)
X[1] := v0; (*v0 – произвольная фиксированная вершина графа *)
DOP[v0] := false;
HAMILT(2)
end
```

7. Гамильтоновы пути и гамильтоновы циклы (5)

Результаты работы программы



Граф



Гамильтоновы циклы

8. Кратчайшие пути на графах (1)

ОБЩИЕ СВЕДЕНИЯ

Задан граф $G = <V, E>$, рёбрам (дугам) которого приписаны веса.
Каждому ребру (дуге) $<u, v>$ приписано вещественное число $w(u, v)$, называемое весом данного ребра. Предполагается, что $w(u, v) = \infty$, если ребро $<u, v>$ отсутствует в графе.
Когда задана последовательность вершин $v_0, v_1, \dots, v_p$, которая определяет путь в G, то его длина определяется как

$$\sum_{i=1}^{p} w(v_{i-1}, v_i).$$

(* Вход *D*[*v*] – расстояние от фикс. вершины до *v*, *W*[*u, v*] –веса *)
(* *t* – конец; *s* – начало, выход: *Stek* – кратчайший путь из *s* в *t* *)
begin
Stek := ∅; *Stek* ⇐ *t*; *v* := *t*;
while *w* ≠ *s* ***do***
    begin

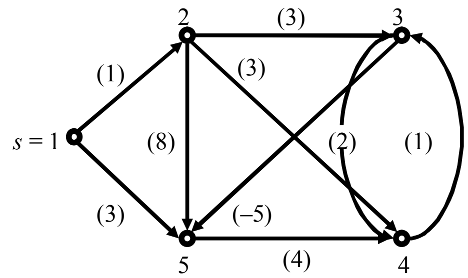
8. КРАТЧАЙШИЕ ПУТИ НА ГРАФАХ (2)

    u := (* вершина, для которой: *D*[*v*] = min{*D*[*u*] + *W*[*u*, *v*]} *)
    Stek ⇐ *u*;
    v := *u*;
    end
end

Нахождение расстояния от источника до всех вершин методом (по алгоритму) Л.Р. Форда и Р.Е. Беллмана.
Ford L.R. Network flow theory. The Rand. Corp., P-923, August 1956.
Bellman R.E. On a routing problem. Quart. Appl. Math. 1958, 16, pp. 87 – 90.
(* Вход *W*[*u, v*] – матрица весов, *s*– начальная вершина *)
(* Выход: *D*[*v*] – расстояние от *s* всех вершин графа *D*[*v*] = *W* (*s*, *v*), *v* ∈ *V* *)
begin
    for *v* ∈ *V* ***do*** *D*[*v*] := *W* (*s*, *v*);
    D[*s*] := *0*;
    for *k* := 1 ***to*** *n* – 2 ***do***
    for *u* ∈ *V* ***do*** *D*[*v*] := *min*(*D*[*v*], *D*[*u*] + *W* (*s*, *v*));
end

8. Кратчайшие пути на графах (3)

Пример работы алгоритма Форда-Беллмана



$$W = \begin{pmatrix} 1 & \infty & \infty & \infty & 3 \\ \infty & \infty & 3 & 3 & 8 \\ \infty & \infty & \infty & 1 & -5 \\ \infty & \infty & 2 & \infty & \infty \\ \infty & \infty & \infty & 4 & \infty \end{pmatrix}$$

k	D[1]	D[2]	D[3]	D[4]	D[5]
	0	1	∞	∞	3
1	0	1	4	4	-1
2	0	1	4	3	-1
3	0	1	4	3	-1

8. КРАТЧАЙШИЕ ПУТИ НА ГРАФАХ (4)

АЛГОРИТМ ДЕЙКСТРЫ

Dijkstra E.W. A note on two problems in connection with graphs. Numer. Math., 1959, 1, pp. 269 – 271.

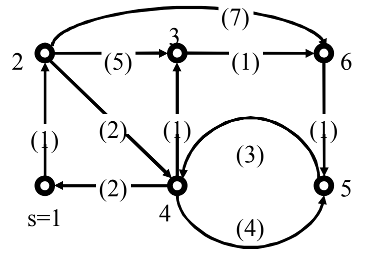
(* Вход: $W[u, v]$ – матрица весов, s– начальная вершина *)

(* Выход: $D[v]$ – расстояние от s всех вершин графа $D[v] = W(s, v)$, $v \in V$ *)

begin
    for $v \in V$ ***do*** $D[v] := W(s, v)$;
    $D[s] := 0$;
    $T := V \setminus \{s\}$;
    while $T \neq \varnothing$ ***do***
        begin
        u := произвольная вершина $r \in T$ такая, что $D[r] := \min\{D[p]: p \in T\}$
        $T := T \setminus \{s\}$;
        for $v \in V$ ***do*** $D[v] := \min(D[v], D[u] + W(s, v))$;
        end
end

8. КРАТЧАЙШИЕ ПУТИ НА ГРАФАХ (5)

Иллюстрация работы алгоритма Дейкстры



D[1]	D[2]	D[3]	D[4]	D[5]	D[6]
0	(1)	(∞)	(∞)	(∞)	(∞)
0	1	(6)	(3)	(∞)	(8)
0	1	((4))	3	(7)	(8)
0	1	4	3	(7)	((5))
0	1	4	3	((6))	5