ЛЕКЦІЯ 13

Пошук найкоротших шляхів у графі

Пошук шляхів у графі.
Алгоритм Террі

Виходячи з **початкової вершини** й здійснюючи послідовний перехід **від кожної досягнутої вершини до суміжної вершини**, додержуватися таких правил.

1. Позначати напрямок, у якому проходимо ребро, як у прямому, так і у зворотному напрямках.
2. З будь-якої вершини просуватися тільки по тому ребру, яке ще не було пройдено або було пройдено в протилежному напрямку.
3. Для будь-якої вершини відзначати перше ребро, по якому до неї потрапили, якщо вершина зустрічається вперше.
4. Будь-яку вершину покидати по ребру, по якому прийшли в цю вершину (у зворотному напрямку) лише тоді, коли немає іншої можливості.

Пояснення алгоритму Террі

Розглянемо алгоритм Террі пошуку шляху в зв'язковому графі, з вершини v_i у вершину v_j, де $v_i \neq v_j$.

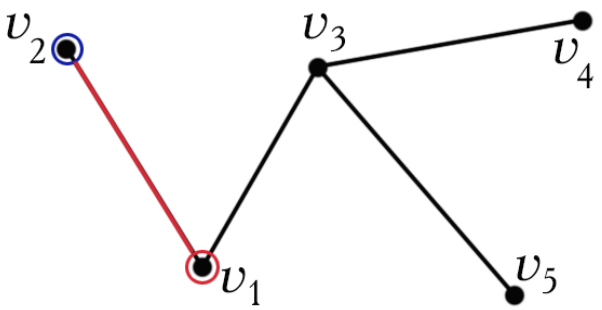
Починаємо рух з вершини v_i.

Переходимо до вершини з множини $v_k \in \Gamma^+\left(v_i\right)$ за такими правилами:

1. Вибираємо довільну дугу, проходимо по ній і відзначаємо напрямок проходження.

Нехай початкова вершина v_1

Вибрали першу суміжну v_2



Як правило, вибираємо суміжну вершину з найменшим номером

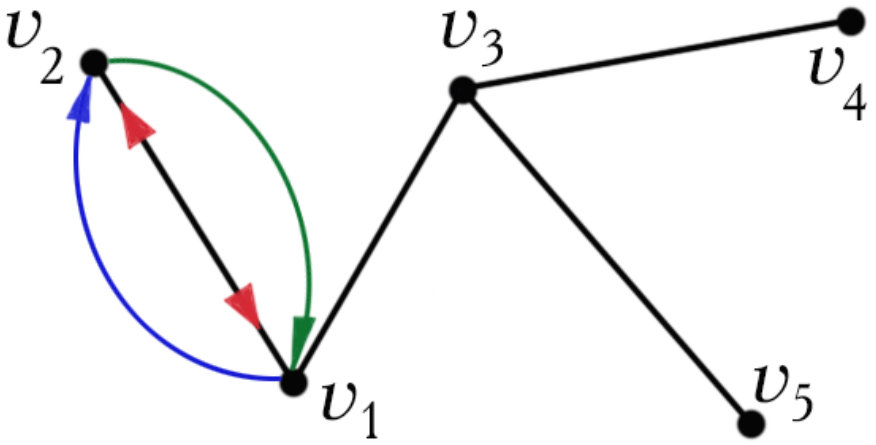
1. Виходимо з вершини v_k по дузі, яка ще не була вибрана, або була пройдена в протилежному напрямку.

2. Відмічаємо проходження ребра в прямому і протилежному напрямку

Перейшли у вершину v_2

Відмітили ребро (v_1, v_2)

Вияснили, що іншого шляху, ніж (v_1, v_2) *не існує*

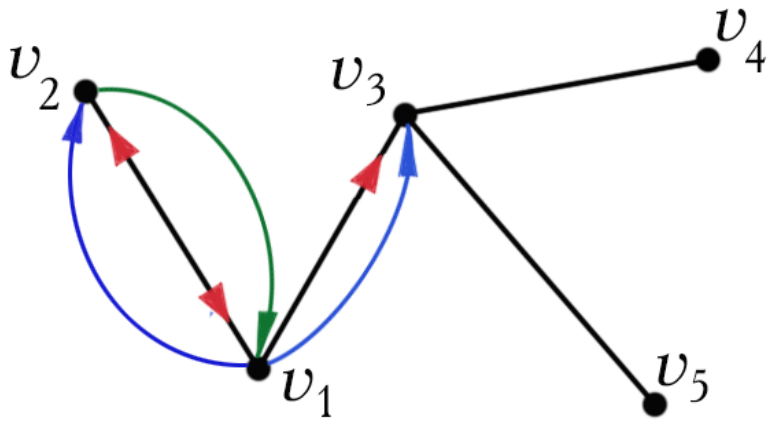


3. Вибираємо те ребро, по якому прийшли, тільки у випадку, коли не існує іншого шляху.

Знову знаходимось у вершині v_1

Відмічаємо ребро (v_1, v_3)

Перейшли у вершину v_3

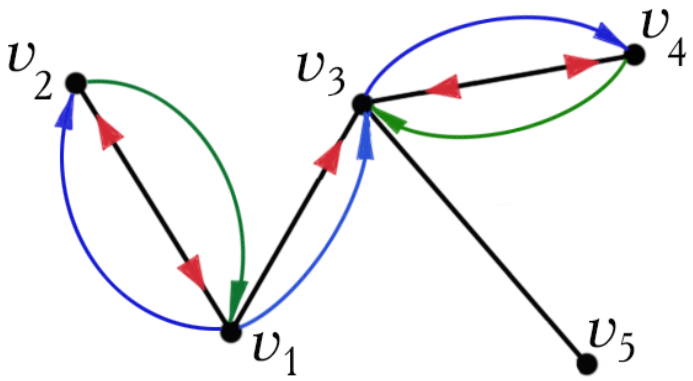


Знаходимось у вершині v_3

Відмічаємо ребро (v_3, v_4)

Перейшли у вершину v_4

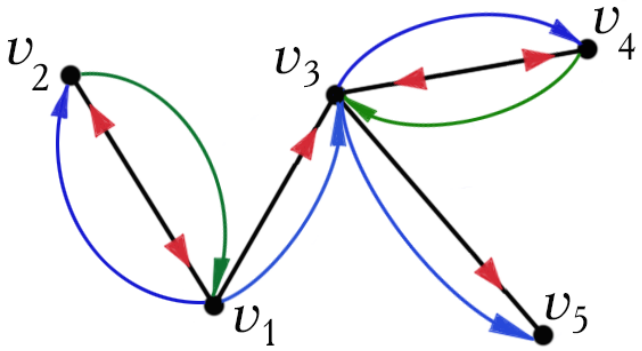
Відмічаємо ребро (v_4, v_3)



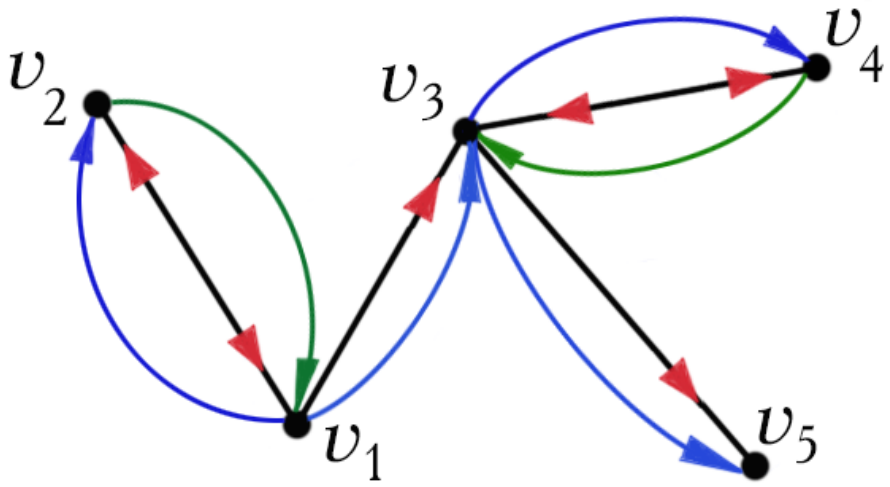
Знаходимось у вершині v_3

Відмічаємо ребро (v_3, v_5)

Перейшли у вершину v_5



ВИСНОВОК



За алгоритмом Террі одержали такий шлях:

$$v_1, v_2, v_1, v_3, v_4, v_3, v_5$$

Якщо виключити цикли, то одержимо простий ланцюг:

$$v_1, v_3, v_5$$

Алгоритм Террі

Структура даних алгоритму

V — *множина вершин графа;*

m — *кількість ребер графа;*

M — *матриця суміжності графа.*

P-*множина ребер, які утворюють орієнтований цикл,*

що проходить через кожне ребро графа по одному разу в кожному з двох напрямків.

begin

Вибрати вершину $a \in V$;

Початкова множина ребер P:=0;

Максимальна кількість кроків: $k := 2m$;

Формування робочої множини вершин: $V' := V$;

While $k \neq 0$ **do**
begin
Вибрати ребро (a,b), *для якого* $M(a,b)=1$,
Причому в останню чергу вибирати ребро (a,b),
для якого $M(b,a)=0$;

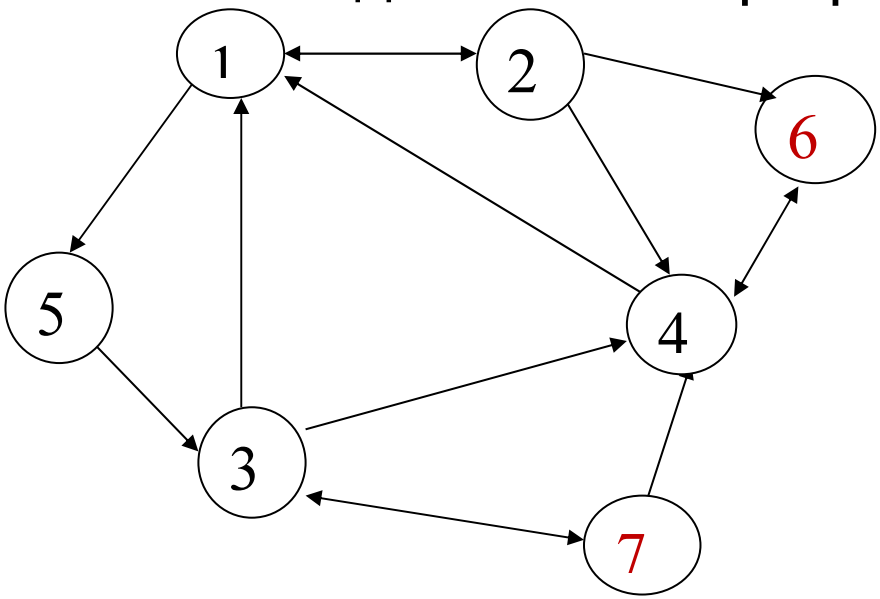
if $b \in V'$ **then**
begin
Вилучити b *з* V'; $M(a,b):=0$;
end else $M(a,b):=0$;
$P:=P\cup\{(a,b)\}$; $k:=k-1;\ a:=b;$
end
end.

Опис послідовності дій в алгоритмі Террі

1. Скласти матрицю суміжності MG[i,j].
2. Задати початкову й кінцеву вершини vs і vf.
3. Задати дві змінні:
4. row – поточний рядок MG[i,j];
5. column – поточний стовпець MG[i,j].
6. row:= vs; column := 1.
7. Якщо (i,j) елемент матриці MG[i,j] одиничний, то обнуляємо цей елемент і елемент (j,i) (якщо він одиничний).
8. Заносимо номер і-го рядка в стек. Переходимо в рядок з номером j.
9. Якщо (i,j) елемент нульовий, то переходимо до аналізу наступного стовпця і-го рядка.
10. Якщо виявилося, що всі елементи аналізованого рядка нульові, то дістаємо зі стека номер вершини й переходимо в рядок з цим номером.
11. Якщо ми переходимо з одного рядка в інший, то номер стовпця робимо одиничним.
12. Аналіз матриці закінчується, коли потрапляємо в рядок з vf -м номером.
13. Вміст стека – маршрут з vs в vf.

Розв'язування задачі вручну

Задамо граф.
Нижче наведена схема графа.

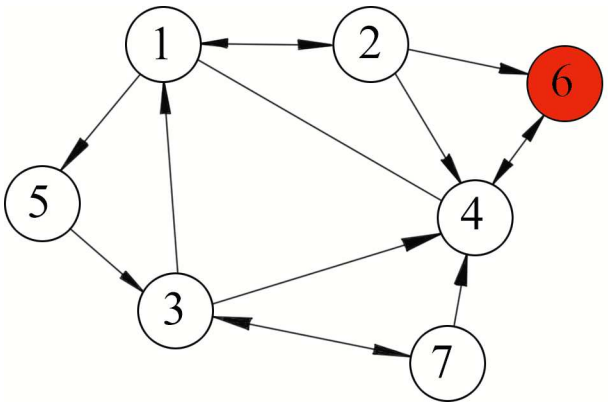


ЗАВДАННЯ: Знайти маршрут з початкової вершини 6 в кінцеву вершину 7.

Уважно розглянувши граф, знайдемо вдповідь: {6,4,1,5,3,7}.

РОЗВ'ЯЗОК: (1)

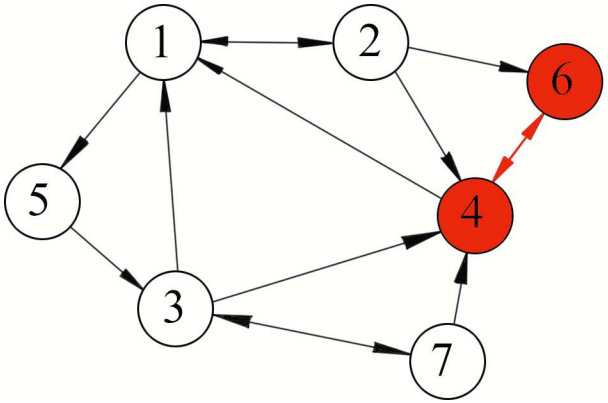
	1	2	3	4	5	6	7
1		1			1		
2	1			1		1	
3	1			1			1
4	1					1	
5			1				
6				1			
7			1	1			



1) Розглянемо перетин рядка 6 і стовпця 1.
   1а) Оскільки елемент (6,1) нульовий, переходимо до елемента (6,2) і т.д. до знаходження одиничного елемента.
2) Знаходимо перший ненульовий елемент у рядку 6: (6,4).
3) Елемент (6,4) одиничний. Заповнюємо стек: {6}.

Переписуємо матрицю (2)

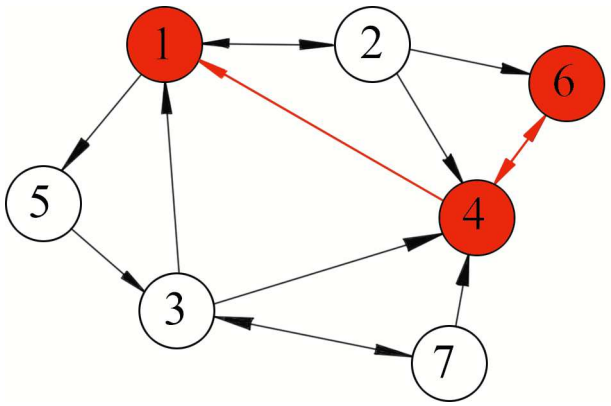
	1	2	3	4	5	6	7
1		1			1		
2	1			1		1	
3	1			1			1
4	1					0	
5			1				
6				0			
7			1	1			



1) Обнуляємо (6,4) і (4,6)
2) Переходимо до аналізу рядка 4 і стовпця 1.
   Вміст стека: {6}.
3) Елемент (4,1) одиничний. Заповнюємо стек: {6,4}.

Переписуємо матрицю (3)

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



1) Обнуляємо (4,1). Елемент (1,4) не обнуляємо, оскільки він нульовий

2) Переходимо до аналізу рядка 1 і стовпця 1.

Вміст стека: {6,4}.

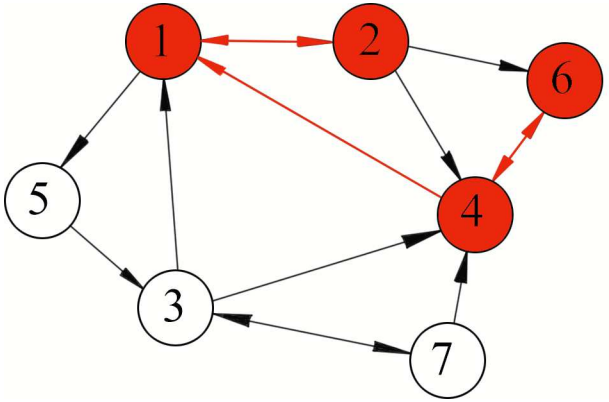
3) Елемент (1,1) нульовий, переходимо до елемента (1,2).

4) Елемент (1,2) одиничний.

Заповнюємо стек: {6,4,1}.

Переписуємо матрицю (4)

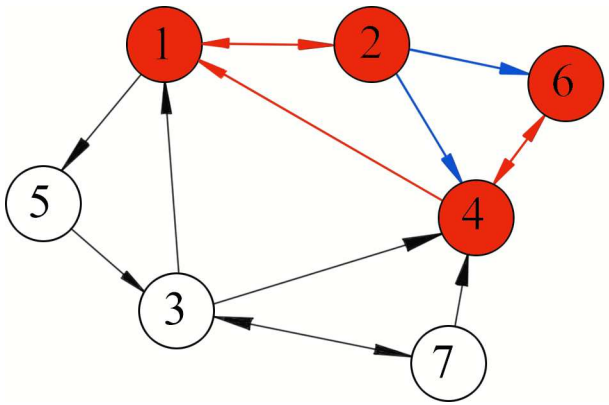
	1	2	3	4	5	6	7
1		0			1		
2	0			1		1	
3	1			1			1
4	0					0	
5			1				
6				0			
7			1	1			



1) Обнуляємо (1,2) і (2,1)
2) Переходимо до аналізу рядка 2 і стовпця 1.
   Вміст стека: {6,4,1}.
3) Знаходимо перший ненульовий елемент у рядку 2: (2,4).
4) Елемент (2,4) одиничний. Заповнюємо стек: {6,4,1,2}.

Переписуємо матрицю (5)

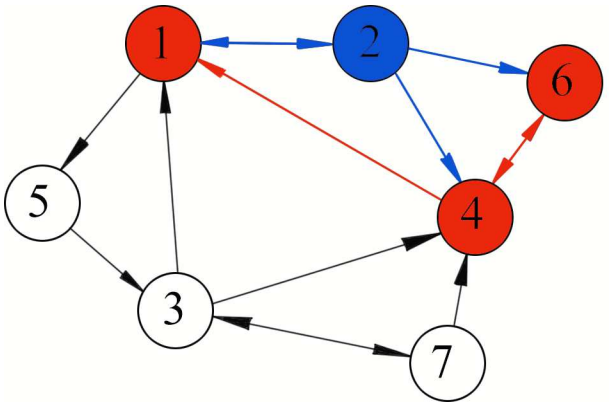
	1	2	3	4	5	6	7
1		0			1		
2	0			0		1	
3	1			1			1
4	0					0	
5			1				
6				0			
7			1	1			



1) Обнуляємо (2,4).
2) Переходимо до аналізу рядка 4 . Вміст стека: {6,4,1,2}.
Рядок 4 нульовий, виймаємо зі стека елемент. Вміст стека: {6,4,1}.
3) Переходимо до аналізу рядка 2 і стовпця 1
4) Знаходимо перший ненульовий елемент у рядку 2: (2,6).
Елемент (2,6) одиничний. Заповнюємо стек: {6,4,1,2}.

Переписуємо матрицю (6)

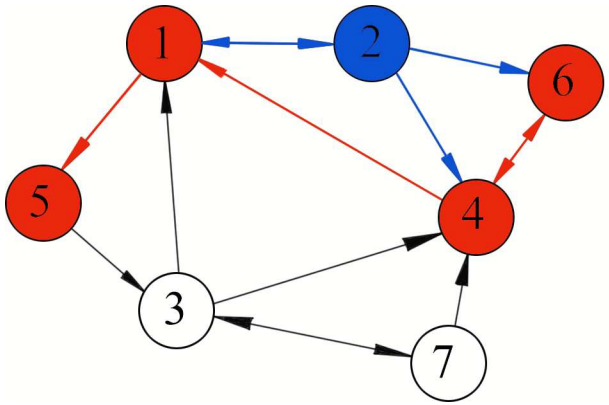
	1	2	3	4	5	6	7
1		0			1		
2	0			0		0	
3	1			1			1
4	0					0	
5			1				
6				0			
7			1	1			



1) Обнуляємо (2,6).
2) Переходимо до аналізу рядка 6. Вміст стека: {6,4,1,2}.
Рядок 6 нульовий, виймаємо зі стека елемент. Вміст стека: {6,4,1}.
3) Переходимо до аналізу рядка 2. Вміст стека: {6,4,1}.
Рядок 2 нульовий, виймаємо зі стека елемент 1. Вміст стека: {6,4}.
4) Переходимо до аналізу рядка 1 і стовпця 1. Вміст стека: {6,4}.
5) Знаходимо перший ненульовий елемент у рядку 1: (1,5).
6) Елемент (1,5) одиничний. Заповнюємо стек: {6,4,1}.

Переписуємо матрицю (7)

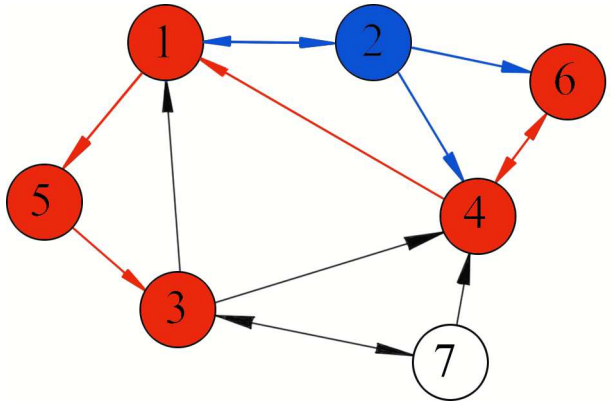
	1	2	3	4	5	6	7
1		0			0		
2	0			0		0	
3	1			1			1
4	0					0	
5			1				
6				0			
7			1	1			



1) Обнуляємо (1,5).
2) Переходимо до аналізу рядка 5 і стовпця 1.
Вміст стека: {6,4,1}.
5) Знаходимо перший ненульовий елемент у рядку 5: (5,3).
6) Елемент (5,3) одиничний. Заповнюємо стек: {6,4,1,5}.

Переписуємо матрицю (8).

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



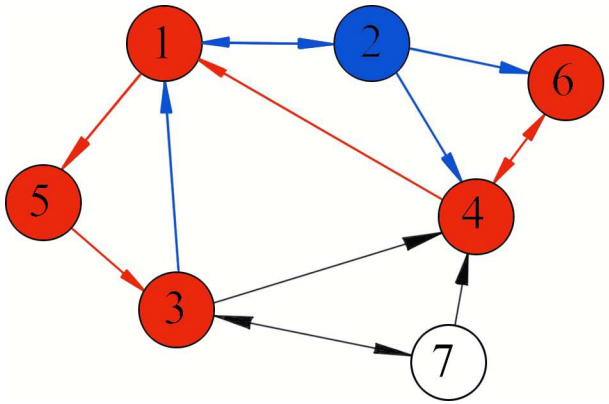
1) Обнуляємо (5,3).

2) Переходимо до аналізу рядка 3 і стовпця 1. Вміст стека: {6,4,1,5}.

3) Елемент (3,1) одиничний. Заповнюємо стек: {6,4,1,5,3}.

Переписуємо матрицю (9).

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



1) Обнуляємо (3,1).

Переходимо до аналізу рядка 1. Вміст стека: {6,4,1,5,3}.

2) Рядок 1 нульовий, виймаємо елемент. Стек: {6,4,1,5}.

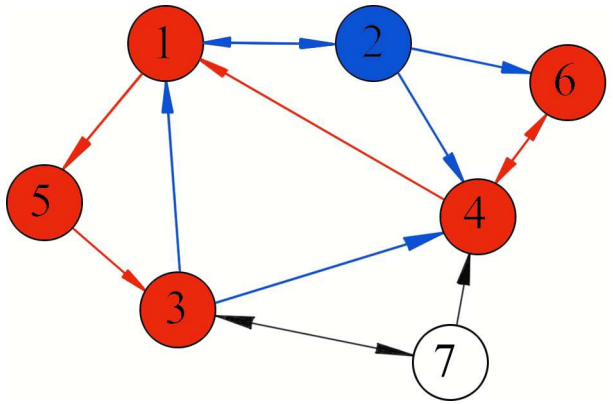
Переходимо до аналізу рядка 3 . Вміст стека: {6,4,1,5}.

3) Знаходимо перший ненульовий елемент у рядку 3: (3,4).

4) Елемент (3,4) одиничний. Заповнюємо стек: {6,4,1,5,3}.

Переписуємо матрицю (10).

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



1) Обнуляємо (3,4).

2) Переходимо до аналізу рядка 4. Стек: {6,4,1,5,3}.

Рядок 4 нульовий, виймаємо елемент. Стек: {6,4,1,5}.

Переходимо до аналізу рядка 3 . Стек: {6,4,1,5}.

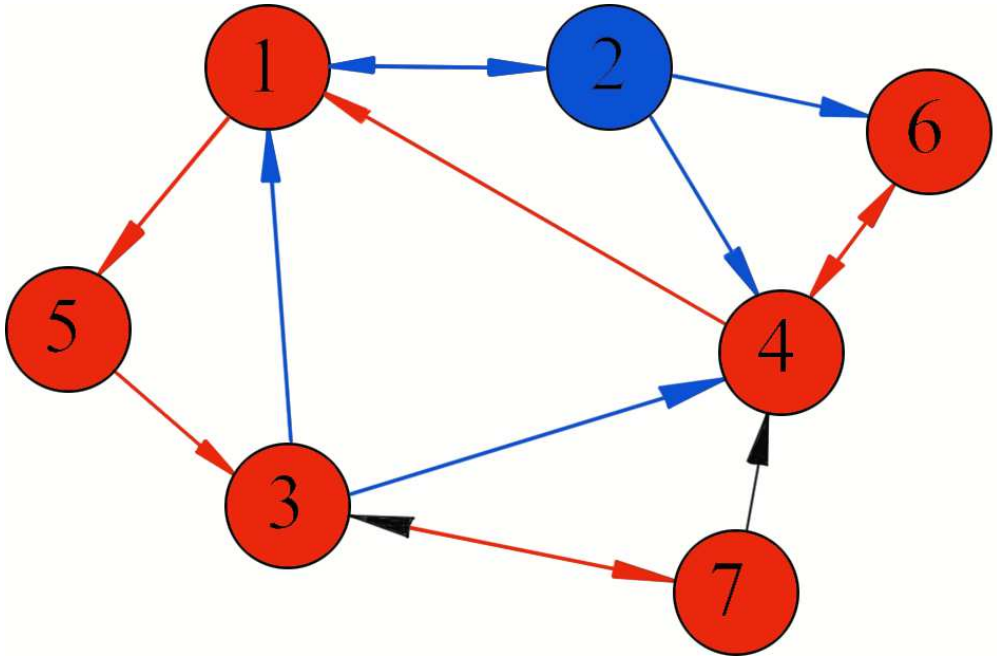
3) Знаходимо перший ненульовий елемент у рядку 3: (3,7).

4) Елемент (3,7) одиничний. Заповнюємо стек: {6,4,1,5,3}.

Результат роботи алгоритму Террі з пошуку найкоротшого шляху в незваженому графі

Оскільки вершина 7 — кінцева за умовою, потрапляння в неї є ознакою закінчення роботи алгоритму Террі.

Отже, відповідь одержуємо шляхом занесення кінцевої вершини в стек: {6,4,1,5,3,7}.



Хвильовий алгоритм. Структура даних

Нехай $G = (V, E)$ непустий граф. Потрібно знайти шлях між вершинами s і t графа ($s \neq t$), який містить мінімальну кількість проміжних вершин (ребер).

Структура даних

1. $Time[N]$ – масив хвильових міток вершин графа G.

Початкова установка: $For\ i := 0\ to\ N\ do\ Time[i] := -1;\ Time[s] = 0$.

2. $OldFront$ – множина старого фронту хвилі.

Початкова установка: $OldFront := \{s\}$.

3. $NewFront$ – множина нового фронту хвилі.

Початкова установка: $NewFront := \{\ \}$.

4. T – змінна поточного часу.

Початкова установка: $T := 0$.

Алгоритм полягає в наступному:

1. Для кожної з вершин, що входять в *OldFront*, переглядаємо суміжні вершини u_i (Спочатку $OldFront := \{s\}$)

2. Якщо $Time[u_i] = -1$, то $Time[u_i] := T + 1$;

$NewFront := NewFront + \{u_i\}$;

3. Якщо $NewFront = \{\ \}$, то КІНЕЦЬ ("немає розв'язку");

4. Якщо $t \in NewFront$ (тобто одна з вершин u_i співпадає з t), то знайдено найкоротший шлях між s і t з $Time[t] := T + 1$ проміжними ребрами; потім КІНЕЦЬ ("розв'язок знайдений"); Якщо ні одна з вершин u_i не співпадає з t, то п.5

5. $OldFront := NewFront$; $NewFront := \{\ \}$; $T := T + 1$; goto (1).

Зауваження: на кроці (1) "**сусідніми**" вершинами для **неорієнтованих графів** вважаються всі **суміжні вершини**, а для **орграфів** – вершини, у які з даної вершини **ведуть дуги**.

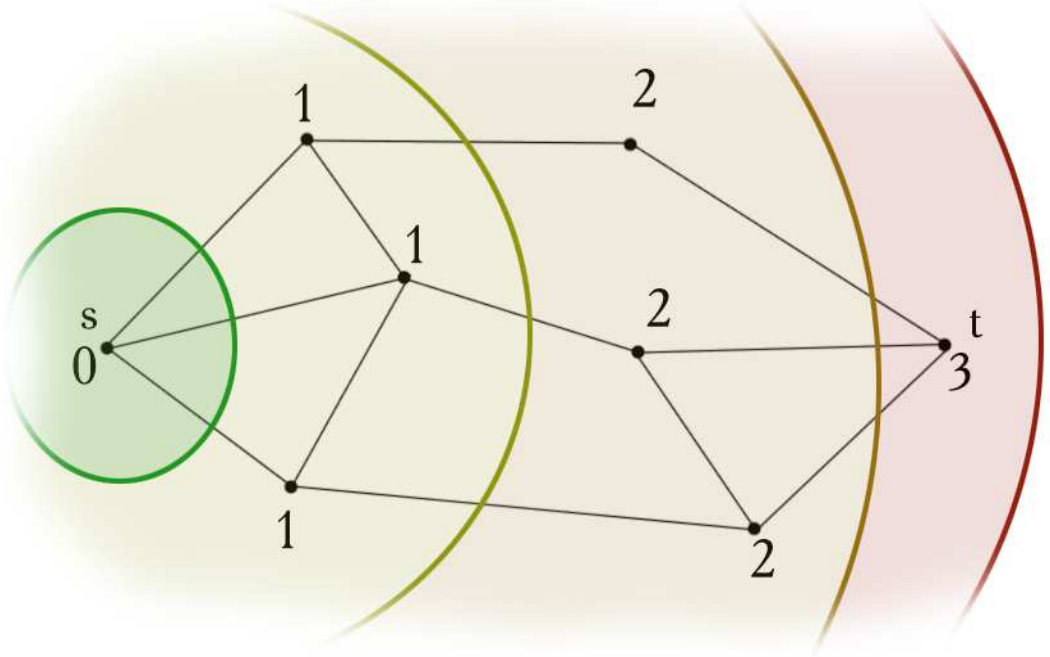
Відновлення найкоротшого шляху.
Якщо на кроці (4) була досягнута вершина t, то алгоритм побудови найкоротшого шляху такий:

1. Серед сусідів вершини t знайдемо будь-яку вершину з хвильовою міткою $Time\left(t\right)-1$.

2. Серед сусідів вже знайденої вершини знайдемо вершину з міткою $Time\left(t\right)-2$, і т. д., поки не досягнемо s.

Знайдена послідовність вершин визначає один з найкоротших шляхів з s в t

На практиці вигідно зберігати інформацію про те, з якої вершини "хвиля" прийшла у вершину u_i – тоді відновлення шляху відбувається швидше.



На малюнку показана нумерація вершин, отримана в результаті роботи хвильового алгоритму.

Пошук найкоротшого шляху у зваженому графі

Нехай дано граф G з матрицею ваг $C = \left(c_{i,j} \right)$.

Задача про найкоротший шлях полягає в знаходженні найкоротшого шляху від заданої **початкової вершини** $s \in V$ до заданої **кінцевої вершини** $t \in V$ за умови, що такий шлях існує.

Алгоритм Дейкстри знаходить найкоротший шлях **для випадку** $c_{ij} \geq 0$.

1. Вершинам приписують **тимчасові позначки**. Нехай $l\left(v_i \right)$ – позначка вершини v_i.
2. Кожна позначка вершини **дає верхню границю довжини шляху** від s до цієї вершини.
3. Величини цих **позначок зменшують ітераційно**.
4. На кожному кроці ітерації одна з тимчасових позначок **стає постійною**.
5. Величина цієї позначки є точною довжиною найкоротшого шляху від s до розглянутої вершини.

Теоретичний опис алгоритму Дейкстри

Присвоєння початкових значень

Крок 1. Встановимо $l\left(s\right) := 0$ і вважатимемо цю позначку постійною. Встановимо $l\left(v_i\right) := \infty$ для всіх $v_i \neq s$ і вважатимемо ці позначки тимчасовими. Встановимо $p = s$.

Відновлення позначок

Крок 2. Для всіх $v_i \in \Gamma\left(p\right)$, позначки яких тимчасові, змінимо позначки у відповідності з таким виразом:

$$l\left(v_i\right) \leftarrow \min\left[l\left(v_i\right), l\left(p\right) + c\left(p, v_i\right)\right]$$

Перетворення позначки в постійну

Крок 3. Серед вершин з тимчасовими позначками знайдемо таку, для якої

$$l\left(v_i^*\right) = \min l\left(v_i\right),\ v_i \in \Gamma\left(p\right)$$

Крок 4. Будемо вважати позначку $l\left(v_i^*\right)$ постійною і встановлюємо $p = v_i^*$.

Крок 5. **Якщо потрібно знайти шлях від s до t.** Якщо $p = t$, то $l\left(p\right)$ є довжиною найкоротшого шляху від вершини s до вершини t. Останов.

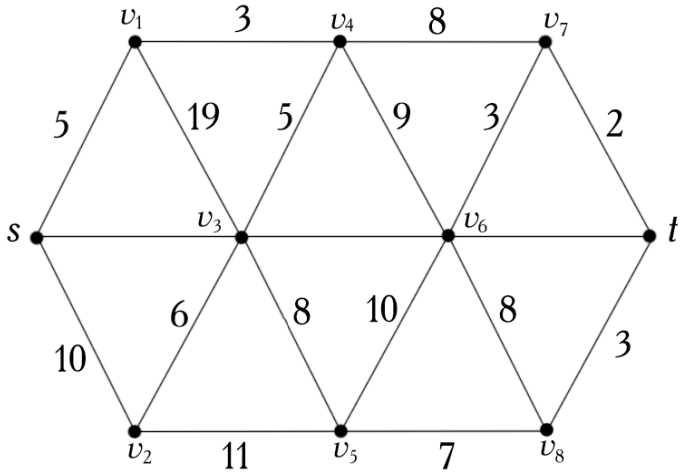
Крок 6. Якщо $p \neq t$, перейти до кроку 2.

Крок 7. **Якщо потрібно знайти шляхи від s до всіх інших вершин.** Якщо всі вершини позначені як постійні, то ці позначки дають довжини найкоротших шляхів. Останов.

Крок 8. Якщо деякі позначки залишаються тимчасовими, то перейти до кроку 2.

Приклад ручного розв'язування задачі

Розглянемо граф, зображений на рисунку, де кожне неорієнтоване ребро розглядається як пара протилежно орієнтованих дуг рівної ваги. Ця вага виписана на малюнку, що зображує граф, поруч із ребром.



Потрібно знайти найкоротші шляхи від вершини s до всіх інших вершин. Для цього використовуємо алгоритм Дейкстри. Постійні позначки відзначаються знаком +, інші позначки є тимчасовими.

Схема застосування алгоритму

Крок 1. $l(s)=0^+, l(v_i)=\infty,\ i=1,\ldots,8,\ p=s.$

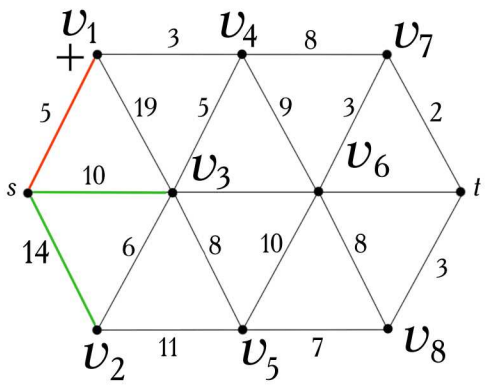
Перша ітерація

Крок 2. $\Gamma(s)=\{v_1,v_2,v_3\}-$ усі позначки тимчасові.

$$l(v_1)=\min\left[\infty, 0^+ +5\right]=5,$$

$$l(v_2)=\min\left[\infty, 0^+ +14\right]=14,$$

$$l(v_3)=\min\left[\infty, 0^+ +10\right]=10.$$



Крок 3. $l(v_1)=\min\limits_{i=1,2,3} l(v_i)=5.$

Крок 4. $l(v_1)=5^+ - v_1$ одержує постійну позначку; $p=v_1$.

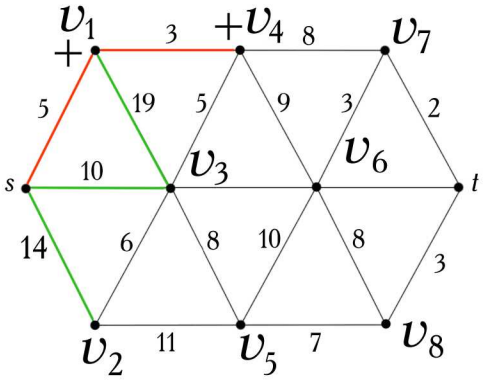
Крок 5. Не всі вершини мають постійні позначки, тому переходимо до кроку 2.

Друга ітерація

Крок 2. $\Gamma(p)=\Gamma(v_1)=\{s,v_3,v_4\}$. Вершина s має постійну позначку;

$$l(v_3)=\min\left[10,5^{+}+19\right]=10,$$

$$l(v_4)=\min\left[\infty,5^{+}+3\right]=8.$$



Крок 3. $l(v_4)=\min\limits_{i=3,4} l(v_i)$.

Крок 4. Вершина v_4 одержує постійну мітку: $l(v_4)=8^{+};\ p=v_4$.

Крок 5. Не всі вершини мають постійні позначки, тому переходимо до кроку 2.

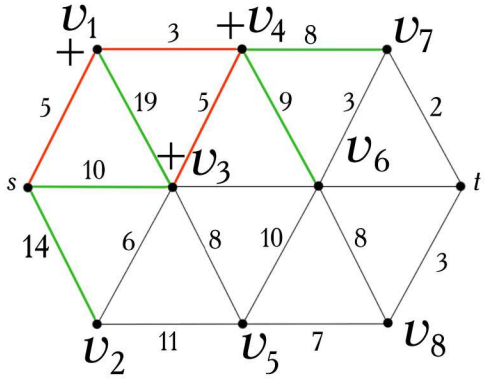
Третя ітерація

Крок 2. $\Gamma(p)=\Gamma(v_4)=\{v_1,v_3,v_6,v_7\}$. Вершина v_1 має постійну позначку;

$$l(v_3)=\min\left[10,8^{+}+5\right]=10,$$

$$l(v_7)=\min\left[\infty,8^{+}+8\right]=16,$$

$$l(v_6)=\min\left[\infty,8^{+}+9\right]=17.$$



Крок 3. $l(v_3)=\min\limits_{i=3,6,7} l(v_i)=10.$

Крок 4. Вершина v_3 одержує постійну мітку: $l(v_3)=10^{+};\ p=v_3.$

Крок 5. Не всі вершини мають постійні позначки, тому переходимо до кроку 2.

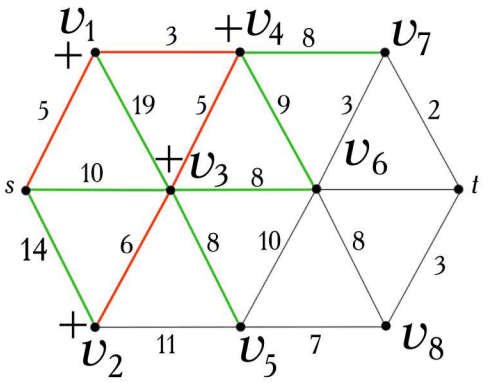
Четверта ітерація

Крок 2. $\Gamma(p)=\Gamma(v_3)=\{s,v_1,v_2,v_4,v_5,v_6\}$. Вершини s,v_1,v_4 мають постійні позначки;

$$l(v_2)=\min\left[14,10^{+}+6\right]=14,$$

$$l(v_5)=\min\left[\infty,10^{+}+8\right]=18,$$

$$l(v_6)=\min\left[17,10^{+}+8\right]=17.$$



Крок 3. $l(v_2)=\min\limits_{i=2,5,6} l(v_i)=14.$

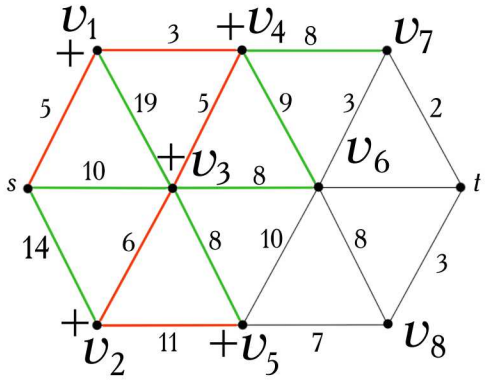
Крок 4. Вершина v_2 одержує постійну мітку: $l(v_2)=14^{+};\ p=v_2.$

Крок 5. Не всі вершини мають постійні позначки, тому переходимо до кроку 2.

П'ята ітерація

Крок 2. $\Gamma(p)=\Gamma(v_2)=\{s,v_3,v_5\}$. Вершини s,v_3 має постійні позначки;

$$l(v_5)=\min\left[18,14^{+}+11\right]=18.$$



Крок 3. $l(v_5)=\min\limits_{i=5} l(v_i)=18.$

Крок 4. Вершина v_5 одержує постійну мітку: $l(v_5)=18^{+};\ p=v_5$.

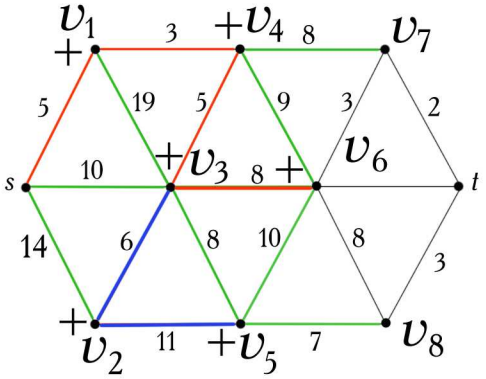
Крок 5. Не всі вершини мають постійні позначки, тому переходимо до кроку 2.

Шоста ітерація

Крок 2. $\Gamma(p)=\Gamma(v_5)=\{v_2,v_3,v_6,v_8\}$. Вершини v_2, v_3 мають постійні позначки;

$$l(v_6)=\min\left[17,18^{+}+10\right]=17,$$

$$l(v_8)=\min\left[\infty,18^{+}+7\right]=25.$$



Крок 3. $l(v_6)=\min\limits_{i=6,8} l(v_i)=17.$

Крок 4. Вершина v_6 одержує постійну мітку: $l(v_6)=17^{+};\ p=v_6.$

Крок 5. Не всі вершини мають постійні позначки, тому переходимо до кроку 2.

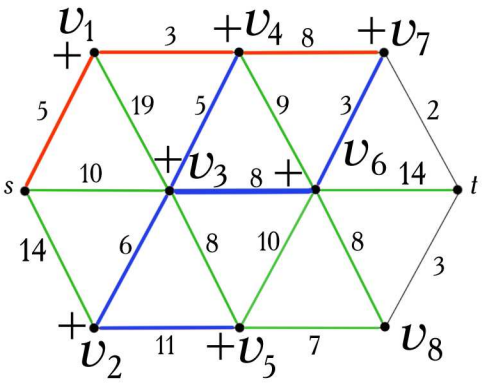
Сьома ітерація

Крок 2. $\Gamma(p)=\Gamma(v_6)=\{v_3,v_4,v_4,v_7,v_8,t\}$. Вершини v_3,v_4,v_5 мають постійні позначки;

$$l(v_7)=\min\left[16,17^{+}+3\right]=16,$$

$$l(v_6)=\min\left[25,17^{+}+8\right]=25,$$

$$l(t)=\min\left[\infty,17^{+}+14\right]=31.$$



Крок 3. $l(v_7)=\min\limits_{i=7,8,t} l(v_i)=16.$

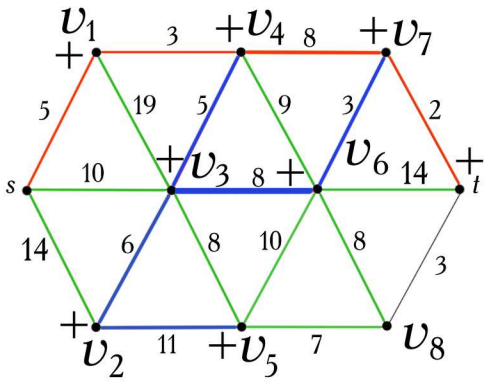
Крок 4. Вершина v_7 одержує постійну мітку: $l(v_7)=16^{+};\ p=v_7.$

Крок 5. Не всі вершини мають постійні позначки, тому переходимо до кроку 2.

Восьма ітерація

Крок 2. $\Gamma(p)=\Gamma(v_7)=\{v_4,v_6,t\}$. Вершини v_4, v_6 мають постійні позначки;

$$l(t)=\min\left[31,16^{+}+2\right]=18.$$



Крок 3. $l(v_t)=\min\limits_{i=t} l(v_i)=18.$

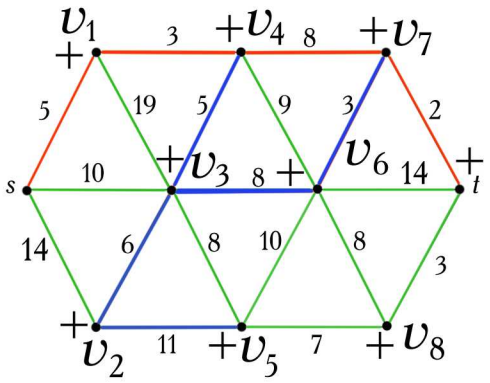
Крок 4. Вершина t одержує постійну мітку: $l(t)=18^{+};\ p=t$.

Крок 5. Не всі вершини мають постійні позначки, тому переходимо до кроку 2.

Дев'ята ітерація

Крок 2. $\Gamma(p)=\Gamma(t)=\{v_6,v_7,v_8\}$. Вершини v_6,v_7 мають постійні позначки;

$$l(v_8)=\min\left[25,18^{+}+3\right]=21.$$



Крок 3. $l(v_8)=\min\limits_{i=8} l(v_i)=21.$

Крок 4. Вершина v_8 одержує постійну мітку: $l(v_8)=21^{+};\ p=v_8$.

Крок 5. Позначки всіх вершин постійні. Останов.

Зауваження

Як тільки всі позначки вершин графа стають постійними, тобто знайдені довжини найкоротших шляхів від вершини *s* до будь-якої іншої вершини, самі шляхи можна одержати, використовуючи співвідношення

$$l\left(v_i'\right)+c\left(v_i',v_i\right)=l\left(v_i\right).$$

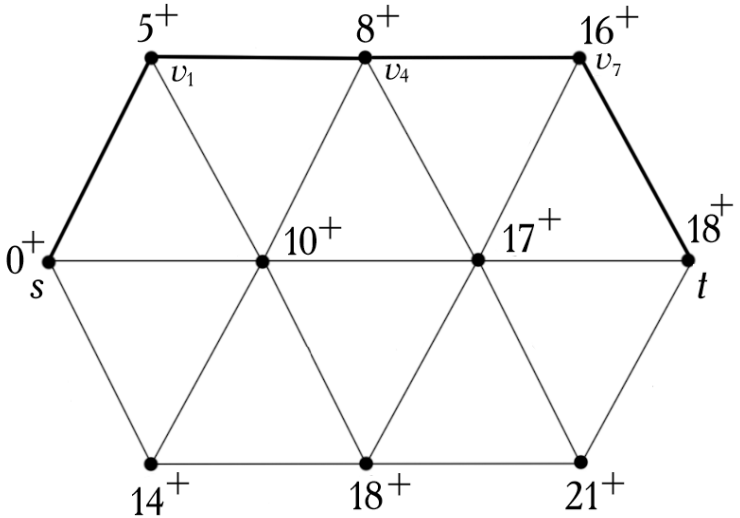
Тут $c\left(v_i',v_i\right)$ – вага дуги, що з'єднує вершину v_i' з вершиною v_i. Це співвідношення дає можливість для кожної вершини v_i знайти попередню до неї вершину v_i' й відновити весь шлях від *s* до v_i.

У розглянутому прикладі найкоротший шлях від вершини *s* до вершини *t* можна відновити зі співвідношень:

$$l(t) = l(v_6) + c(v_6, t),$$
$$l(v_6) = l(v_4) + c(v_4, v_7),$$
$$l(v_4) = l(v_1) + c(v_1, v_4),$$
$$l(v_1) = l(s) + c(s, v_1),$$

отриманих в 7-й, 2-й і 1-й ітераціях.

Йому відповідає послідовність вершин s, v_1, v_4, v_7, t.



Машинний алгоритм Дейкстри

Структура даних

1. Логічний масив $Visited[N]$: $false$ – вершина не розглянута; $true$ – вершина розглянута.

2. Масив $Len[N]$ містить поточні найкоротші відстані від початкової до відповідної вершини.

3. Масив $Path[n]$ містить номера вершин.
$Path[k]$ містить номер передостанньої вершини на поточному найкоротшому шляху з початкової вершини в k–у.

4. $Matrix[i, j]$ – матриця відстаней; $1 \le i \le N, 1 \le j \le N$.

Кроки алгоритму Дейкстри:

1. **Ініціалізація**

Очистимо масив $Visited$ для всіх вершин графа:

For $i := 1$ **to** N **do** $Visited[i] := false$;

Виберемо стартову вершину s;

Заповнимо масив шляху стартовою вершиною s.

For $i := 1$ **to** N **do** $Path[i] := s$;

Перепишемо рядок стартової вершини з матриці відстаней у масив Len

For $i := 1$ **to** N **do** $Len[i] := Matrix[s, i]$;

Виконаємо початкову установку для стартової вершини s

$Visited[s] := true$; $Path[s] := 0$.

2. **Загальний крок**

Виконуємо перевірку на те, чи залишилися невідмічені вершини

```
Function Possible: Boolean;
Var i: integer;
begin
   Possible:=True;
   For i:=1 to n do
   If not Visited[i] then     Exit;
   Possible:=False;
end;
```

Знайдемо серед невідмічених ту вершину, що має мінімальну відстань від поточної вершини

```
Function Min: Integer;
Var i, minvalue, currentmin: integer;
begin
  Minvalue:=Infinity;
  For i:=1 to N do
  If Not Visited[i] then
  If Len[i]<minvalue then
  begin
    currentmin:=i;
    minvalue:=Len[i]
  end;
  Min:=currentmin;
end;
```

Знайдену вершину k з мінімальною відстанню від поточної відзначаємо як позначену:

```
Visited[k]:=True; (Позначка стає постійною)
```

Потім модифікуємо масиви *Len* й *Path* з метою визначення суміжних вершин і відстані до них від стартової вершини

```
For i:=1 to n do
    If Len[i]>Len[k]+Mattr[i, k] then
    begin
      Len[i]:=Len[k]+Mattr[i, k];
      Path[i]:=k;
    end;
  end;
```

3. **Завершальні дії**

Якщо всі елементи масиву $Visited$ дорівнюють $true$, тобто всі вершини графа позначені, то довжина шляху від v_i до v_k дорівнює $Len[k]$.

Вивід вершин, що входять в шлях

```
Write('Кінцева вершина: '); Readln(Finish);
Write(Finish);
k:=Finish;
Finish:=Path[Finish];
While Finish<>Start do
begin
  Write(' <-  ', Finish);
  Finish:=Path[Finish];
end;
Writeln(' <-  ', Start);
Write('Довжина шляху = ', Len[k]);
```

Алгоритм Форда – Беллмана знаходження мінімального шляху

Передбачається, що орієнтований граф не містить контурів від'ємної довжини.

Основними величинами, які потрібно обчислювати в цьому алгоритмі, є величини $\lambda_i\left(k\right)$, де $i = 1,2,\ldots,n$ (n – число вершин графа); $k = 1,2,\ldots,n-1$.

Для фіксованих i і k величина $\lambda_i\left(k\right)$ дорівнює довжині мінімального шляху, що веде із заданої початкової вершини v_1 у вершину v_i і складається з не більше як k дуг.

Крок 1. Установка початкових умов. Ввести кількість вершин графа n і матрицю ваг $C = \left|c_{ij}\right|$.

Крок 2. Встановити $k = 0$. Встановити $\lambda_i\left(0\right) = \infty$ для всіх вершин, крім v_1; встановити $\lambda_1\left(0\right) = 0$.

Крок 3. У циклі по k, $k = 1,2,\ldots,n-1$, кожній вершині v_i на k-му кроці приписати індекс $\lambda_i\left(k\right)$ за наступним правилом:

$$\lambda_i\left(k\right) = \min_{1\le j\le n}\left\{\lambda_j\left(k-1\right) + c_{ji}\right\} \qquad (1)$$

для всіх вершин, крім v_1, встановити $\lambda_1\left(k\right) = 0$.

У результаті роботи алгоритму формується таблиця індексів

$$\lambda_i\left(k\right),\ i = 1,2,\ldots,n;\ k = 0,1,2,\ldots,n-1.$$

При цьому $\lambda_i\left(k\right)$ визначає довжину мінімального шляху з першої вершини в i-у, що містить не більше, ніж k дуг.

Крок 5. Відновлення мінімального шляху. Для будь-якої вершини v_s попередня до неї вершина v_r визначається зі співвідношення:

$$\lambda_r\left(n-2\right)+c_{rs}=\lambda_s\left(n-1\right),\ v_r\in G^{-1}\left(v_s\right), \qquad (2)$$

де $G^{-1}\left(v_s\right)$ – прообраз вершини v_s.

Для знайденої вершини v_r попередня до неї вершина v_q визначається зі співвідношення:

$$\lambda_q\left(n-3\right)+c_{qr}=\lambda_r\left(n-2\right),\ v_q\in G^{-1}\left(v_r\right),$$

де $G^{-1}\left(v_r\right)$ – прообраз вершини v_r, і т. д.

Послідовно застосовуючи це співвідношення, починаючи від останньої вершини v_i, знайдемо мінімальний шлях.

Алгоритм Флойда-Уоршелла

Метод Флойда безпосередньо ґрунтується на тому факті, що в графі з додатними вагами ребер будь-який неелементарний (довжиною більше 1 ребра) найкоротший шлях складається з інших найкоротших шляхів.

Цей алгоритм більш загальний у порівнянні з алгоритмом Дейкстри, тому що він знаходить найкоротші шляхи між будь-якими двома вершинами графа.

В алгоритмі Флойда використовується матриця A розміром $n \times n$, у якій обчислюються довжини найкоротших шляхів. Елемент $A\left[i, j\right]$ дорівнює відстані від вершини i до вершини j, яка має скінченне значення, якщо існує ребро $\left(i, j\right)$, і дорівнює нескінченності в протилежному випадку.

Алгоритм Флойда

Основна ідея алгоритму.

Нехай є три вершини i, j, k і задані відстані між ними.

Якщо виконується нерівність $A\left[i,k\right] + A\left[k,j\right] < A\left[i,j\right]$, то доцільно замінити шлях $i \to j$ шляхом $i \to k \to j$.

Така заміна виконується систематично в процесі виконання даного алгоритму.

Ініціалізація. Визначаємо початкову матрицю відстані T_0. Діагональні елементи матриці дорівнюють 0, у такий спосіб показуючи, що ці елементи в обчисленнях не беруть участь. Якщо між вершинами немає прямого ребра, то відповідний елемент матриці T_0 має значення ∞.

Далі встановлюємо $k = 1$.

Основний крок

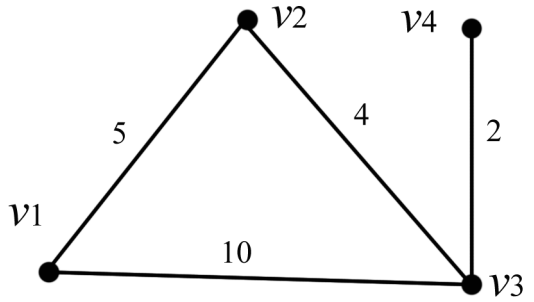
На кожному кроці алгоритм модифікує матрицю T .
Матриця містить довжини найкоротших шляхів між усіма вершинами графа.

```
for k = 1 to n
  for i = 1 to n
    for j = 1 to n
      T[i,j] = min(T[i,j], T[i,k] + T[k,j])
```

Таким чином, алгоритм Флойда виконує n ітерацій, після i-ї ітерації матриця T буде містити довжини найкоротших шляхів між будь-якими двома парами вершин за умови, що ці шляхи проходять через вершини від першої до i-ї.

На кожній ітерації перебирають все пари вершин і шлях між ними скорочується за допомогою проміжної k-ї вершини.

Приклад. Розглянемо роботу алгоритму Флойда на прикладі графа G, що складається з чотирьох вершин.



1. Будуємо початкову матрицю T_0:

$$T_0 = \begin{pmatrix} & v_1 & v_2 & v_3 & v_4 \\ v_1 & 0 & 5 & 10 & \infty \\ v_2 & 5 & 0 & 4 & \infty \\ v_3 & 10 & 4 & 0 & 2 \\ v_4 & \infty & \infty & 2 & 0 \end{pmatrix}$$

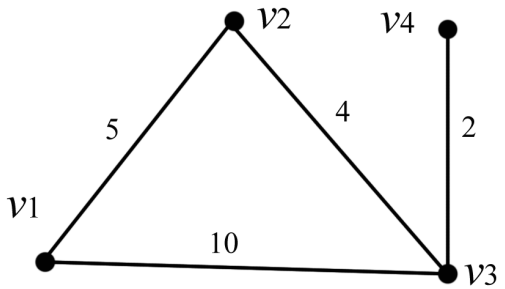
Продовження прикладу роботи алгоритму Флойда

2. Модифікуємо матрицю T_0 при $k=1$

```
for i = 1 to 4
 for j = 1 to 4
   T[i,j] = min(T[i,j], T[i,1] + T[1,j])
```

Замінюємо елемент матриці, якщо шлях між вершинами i та j виявиться коротшим через вершину $k=1$

$$T_1 = \begin{pmatrix} & v_1 & v_2 & v_3 & v_4 \\ v_1 & 0 & 5 & 10 & \infty \\ v_2 & 5 & 0 & 4 & \infty \\ v_3 & 10 & 4 & 0 & 2 \\ v_4 & \infty & \infty & 2 & 0 \end{pmatrix}$$



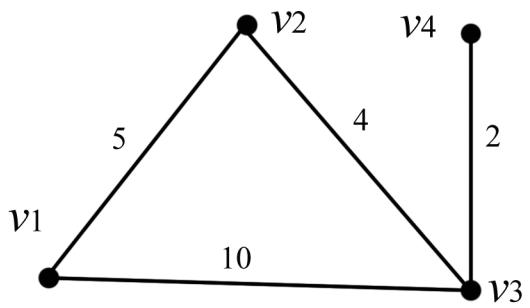
На першому етапі в матриці немає змін, оскільки не знайдено більш коротких шляхів між вершинами через вершину 1

2. Модифікуємо матрицю T_1 при $k = 2$

```
for i = 1 to 4
 for j = 1 to 4
  T[i,j] = min(T[i,j], T[i,2] + T[2,j])
```

Замінюємо елемент матриці, якщо шлях між вершинами i та j виявиться коротшим через вершину $k = 2$

$$T_2 = \begin{pmatrix} & v_1 & v_2 & v_3 & v_4 \\ v_1 & 0 & 5 & 9 & \infty \\ v_2 & 5 & 0 & 4 & \infty \\ v_3 & 9 & 4 & 0 & 2 \\ v_4 & \infty & \infty & 2 & 0 \end{pmatrix}$$



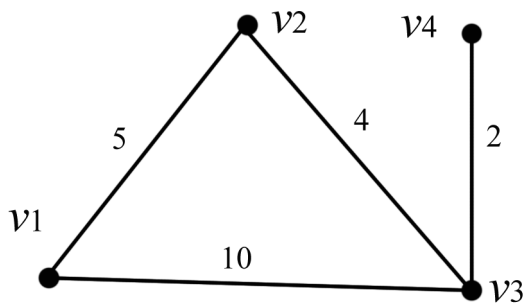
На другому етапі шлях з вершини 1 у вершину 3 виявився коротшим через вершину 2.

3. Модифікуємо матрицю T_2 при $k = 3$

```
for i = 1 to 4
 for j = 1 to 4
  T[i,j] = min(T[i,j], T[i,3] + T[3,j])
```

Замінюємо елемент матриці, якщо шлях між вершинами i та j виявиться коротшим через вершину $k = 3$

$$T_3 = \begin{pmatrix} & v_1 & v_2 & v_3 & v_4 \\ v_1 & 0 & 5 & 9 & 12 \\ v_2 & 5 & 0 & 4 & 6 \\ v_3 & 9 & 4 & 0 & 2 \\ v_4 & 12 & 6 & 2 & 0 \end{pmatrix}$$



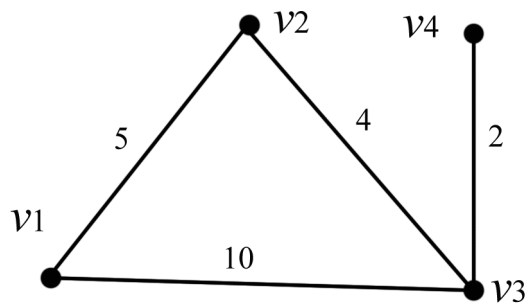
На 3-му етапі встановили шлях у вершину 4 через вершину 3.

4. Модифікуємо матрицю T_2 при $k = 3$

```
for i = 1 to 4
 for j = 1 to 4
  T[i,j] = min(T[i,j], T[i,4] + T[4,j])
```

Замінюємо елемент матриці, якщо шлях між вершинами i та j виявиться коротши через вершину $k = 4$

$$T_4 = \begin{pmatrix} & v_1 & v_2 & v_3 & v_4 \\ v_1 & 0 & 5 & 9 & 12 \\ v_2 & 5 & 0 & 4 & 6 \\ v_3 & 9 & 4 & 0 & 2 \\ v_4 & 12 & 6 & 2 & 0 \end{pmatrix}$$



На 4-му етапі не знайшли оптимальних шляхів через вершину 4, оскільки вона висяча.