Лабораторна робота № 7

Теоретичні відомості. Пошук найкоротших шляхів у графі

Шляхом довжиною k [1] з вершини v_0 у вершину v_k у неорієнтованому графі називають послідовність ребер $e_1 = \{v_0, v_1\}$, $e_2 = \{v_1, v_2\}$, …, $e_k = \{v_{k-1}, v_k\}$. Шлях довжиною k містить k ребер і кожне ребро враховується стільки разів, скільки воно зустрічається в шляху. Вершини v_0 та v_k називають *крайніми*, а інші вершини шляху – *внутрішніми*. *Циклом* у неорієнтованому графі називається шлях, що з'єднує вершину саму з собою. Шлях або цикл називають *простим*, якщо він не містить ребер, що повторюються. Неорієнтований граф називають *зв'язним*, якщо між будь-якими його двома вершинами існує шлях. У протилежному випадку граф є *незв'язним*. Незв'язний граф складається з двох або більше зв'язних підграфів, які називають його *компонентами*.

Для орієнтовного графа поняття шляху (циклу), довжини шляху (циклу), простого шляху (циклу) вводяться аналогічно. Різниця полягає лише у необхідності врахування у шляху (циклі) орієнтації ребер (дуг). Проте поняття зв'язності для орієнтованого графа визначається інакше. Орієнтований граф називають *сильно зв'язним*, якщо між будь-якими його двома вершинами існує орієнтований шлях. Орієнтований граф називають *слабо зв'язним*, якщо між будь-якими двома вершинами існує шлях у відповідному йому неорієнтованому графі (тобто без врахування напрямку дуг). Очевидно, що сильно зв'язаний граф є одночасно і слабо зв'язаним.

Зваженим називають граф, у якому кожному ребру (дузі) присвоєне дійсне число – вага цього ребра (дуги). Для зваженого графа матриця суміжності перетворюється у матрицю ваг – кожен елемент матриці дорівнює вазі відповідного ребра (дуги). У випадку відсутності ребра (дуги) відповідний елемент матриці ваг кладуть рівним 0 чи ∞, залежно від задачі. При задаванні зваженого графа списком ребер чи списками суміжності кожен елемент списку повинен містити також значення ваги відповідного ребра.

Довжиною шляху (циклу) у зваженому графі називають суму ваг всіх ребер (дуг), що утворюють цей шлях. Очевидно, що якщо граф є не зважений, то вагу кожного ребра вважатимемо рівною 1 (як і було означено раніше).

Задача знаходження найкоротшого шляху у зваженому графі є однією з основних задач теорії графів і має прикладне значення. Цю задачу можна розділити на три підзадачі:

1. Знайти найкоротший шлях між заданою парою вершин.
2. Знайти найкоротші шляхи від заданої вершини до всіх інших.
3. Знайти найкоротші шляхи між усіма парами вершин.

Алгоритм Дейкстри

Одним із найвідоміших алгоритмів для знаходження найкоротших шляхів у зваженому орієнтованому графі є алгоритм Дейкстри (E. Dijkstra), який належить до сімейства „*жадібних*" алгоритмів [3, 7, 8]. Цей алгоритм застосовується для розв'язку задач 1 і 2 (як буде показано далі, при розв'язуванні задачі 1 одночасно, в основному, розв'язується також задача 2). Зауважимо, що алгоритм Дейкстри працює тільки у випадку невід'ємних ваг ребер.

В алгоритмі Дейкстри будується множина вершин U, для якої довжини найкоротших шляхів від початкової вершини вже відомі. На кожному кроці до множини U додається одна вершина, відстань до якої від початкової вершини є менша, ніж до всіх інших вершин, що не належать U.

Описаний процес зручно виконувати за допомогою присвоювання вершинам міток. Вершин, що належать множині U позначаються постійними мітками. Решта вершин має тимчасові мітки, і множину таких вершин позначимо як T, $T = V \setminus U$.

Алгоритм Дейкстри для знаходження найкоротшого шляху між вершинами графу.

1. Ініціалізація (присвоєння початкових значень). Вибираємо початкову вершину v_0 від якої будуть розраховуватись найкоротші шляхи. Для цієї вершини довжина найкоротшого шляху $\text{length}(v_0) = 0$ і вважаємо її мітку постійною ($U = \{v_0\}$). Для усіх решта вершин довжини найкоротших шляхів $\text{length}(v \neq v_0) = \infty$ і вважаємо їхні мітки тимчасовими. Поточна вершина $x = v_0$.
2. Оновлення довжин шляхів. Для всіх вершин з тимчасовими мітками ($v \in T$) замінимо значення довжин найкоротших шляхів за таким правилом: $\text{length}(v) = \min\{\text{length}(v); \text{length}(x) + \text{weigth}(x,v)\}$, де $\text{weigth}(x,v)$ – вага дуги між вершинами x та v.
3. Перетворення мітки в постійну. Серед усіх вершин із тимчасовими мітками знайдемо вершину v^* з мінімальною довжиною шляху, тобто $\text{length}(v^*) = \min\{\text{length}(v)\}, v \in T$. Перетворюємо мітку вершини v^* у постійну ($U = U \cup \{v^*\}$) (вершину v^* включаємо в множину *U*). Поточна вершина $x = v^*$.

4.1. Випадок пошуку найкоротшого шляху між двома вершинами: якщо x – кінцева вершина, то $\text{length}(x)$ – шукана довжина найкоротшого шляху і задача виконана; якщо ні – то переходимо до кроку 2.

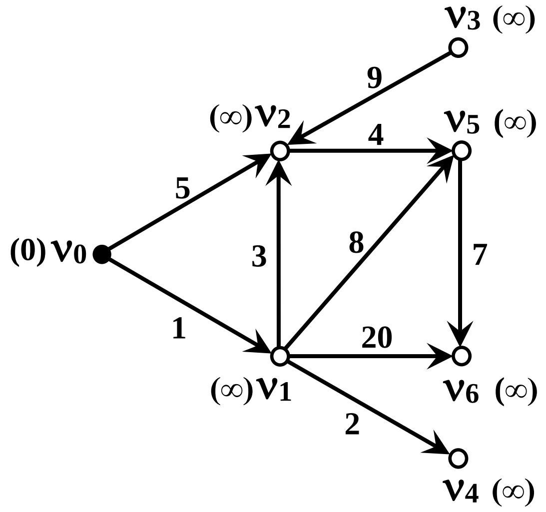
4.2. У випадку пошуку шляхів від v_0 до всіх інших вершин: якщо всі вершини отримали постійні мітки (включені в множину *U*), то ці мітки дорівнюють довжинам найкоротших шляхів і задача виконана; якщо деякі вершини мають тимчасові мітки, то переходимо до кроку 2.

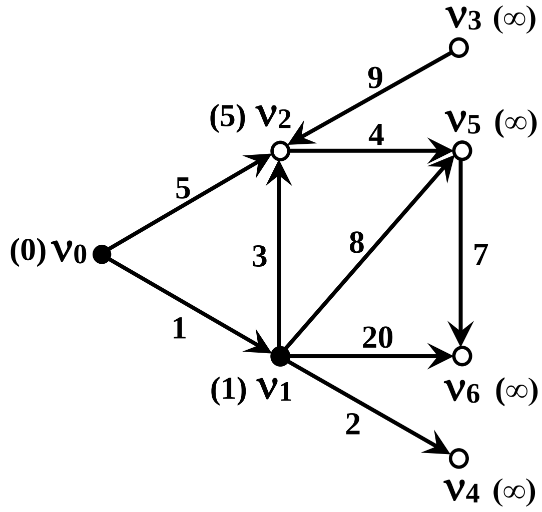
Якщо граф задано матрицею суміжності, складність алгоритму Дейкстри складає $O(n^2)$. Коли кількість дуг *m* значно менша, ніж n^2, то найкраще задавати орієнтований граф списками суміжності. Тоді алгоритм можна реалізувати зі складністю $O(m\log n)$ [3, 7, 8].

Описана вище методика дозволяє обчислити довжину найкоротшого шляху між вершинами. Для знаходження же самого шляху, для кожної вершини потрібно ввести додаткову змінну для запам'ятовування вершини, що передує даній у найкоротшому шляху. У цьому випадку, якщо змінюється довжина шляху для якоїсь вершини $v \in T$: $\text{length}(v) = \text{length}(x) + \text{weigth}(x, v)$ (крок 2 алгоритму), додатковій змінній для цієї вершини потрібно присвоїти значення *x*.

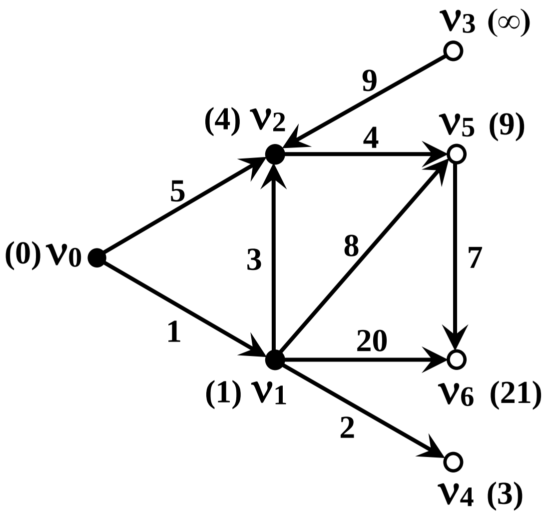
<u>Приклад.</u>

На рис. 7.1, *а–д* наведене покрокове виконання алгоритму Дейкстри для пошуку довжини найкоротшого шляху з вершини v_0 у вершину v_6. Біля кожної вершини у круглих дужках наведене значення її мітки length(…). Вершини, які включено в множину *U*, виділено чорним кольором; мітки таких вершин оголошують постійними. Як видно з цих рисунків, найкоротшим шляхом з вершини v_0 у вершину v_6 є шлях $v_0 \to v_1 \to v_2 \to v_5 \to v_6$ довжиною 15. На рис. 7.1, *е* наведено результат пошуку найкоротших шляхів з вершини v_0 до всіх інших вершин графу. Зауважимо, що вершина v_3 у цьому графі є недосяжною з інших вершин.

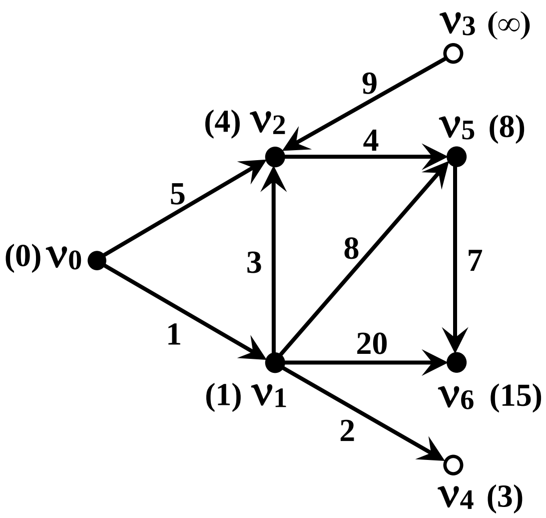




а *б*



в *г*



д *е*

Рисунок 7.1

Хід роботи:

Частина 1. Пошук найкоротших шляхів у графі за алгоритмом Дейкстри

1. Створити новий проект `Lab_7_1`.
2. Для заданого викладачем варіанту графа задати його матрицю ваг (суміжності) `Adj[N][N]`, де `N` – кількість вершин графа.

3. Описати та ініціалізувати наступні масиви: `Length[N]` – масив довжин шляхів (ініціалізується нескінченістю), `Label[N]` – логічний масив міток (`false` – тимчасова мітка, `true` – постійна мітка, спочатку всі мітки тимчасові), `Vertex[N]` – масив, що містить номер вершини що передує даній вершині у шляху (потрібний для відображення самого шляху).
4. Реалізувати крок 1 алгоритму – ініціалізацію.
5. Реалізувати крок 2 алгоритму – оновлення міток за правилом $\mathrm{length}(v) = \min\left\{\mathrm{length}(v); \mathrm{length}(x) + \mathrm{weight}(x, v)\right\};\ v \in T$.
6. Реалізувати крок 3 алгоритму – пошук нової вершини–кандидата на присвоєння постійної мітки за правилом $\mathrm{length}(v^*) = \min\left\{\mathrm{length}(v)\right\},\ v \in T$.
7. Передбачити можливість відсутності шляху між вершинами у графі (випадок слабо зв'язного графа).
8. Реалізувати вивід на екран довжин найкоротших шляхів, а також самих шляхів у графі.
9. Передбачити роботу алгоритму у двох варіантах задачі: пошуку найкоротшого шляху між двома вершинами та пошуку найкоротшого шляху від заданої вершини до інших вершин.
10. Отримати від викладача номер початкової і кінцевої вершин (для варіанту пошуку найкоротшого шляху між двома вершинами) або тільки номер початкової вершини (для варіанту пошуку найкоротшого шляху від заданої вершини до інших вершин). Продемонструвати роботу програми.

Алгоритм Флойда-Воршелла

Алгоритм Флойда-Воршелла (Floyd-Warshall) також служить для знаходження найкоротших шляхів у зваженому орієнтованому графі, але на відміну від алгоритму Дейкстри він дозволяє зразу знайти найкоротші шляхи між усіма парами вершин. Окрім того, даний алгоритм допускає використання як додатних, так і від'ємних значень ваг ребер, однак граф не повинен містити циклів від'ємної довжини [1, 3].

Алгоритм Флойда-Воршелла належить до динамічних алгоритмів [3] і працює ітераційним методом. Введемо матрицю, елемент якої $d_{ij}^{(k)}$ буде рівний довжині найкоротшого шляху з вершини i у вершину j, у якому як внутрішні можуть бути лише перші k вершин графа $G = (V, E)$. Спочатку задається матриця $d_{ij}^{(0)}$ за якою обчислюються послідовно матриці $d_{ij}^{(1)}$, $d_{ij}^{(2)}$, …, $d_{ij}^{(n)}$.

Елементи матриці $d_{ij}^{(0)}$ вибираються рівними вазі дуги між вершинами i та j. Якщо такої дуги не існує, то $d_{ij}^{(0)} = \infty$. Вважатимемо також, що діагональні елементи матриці $d_{ii}^{(0)} = 0$. Зі сказаного слідує, що $d_{ij}^{(0)}$ – це матриця ваг (суміжності) графа G. Матрицю $d_{ij}^{(0)}$ можна розглядати також, як матрицю найкоротших шляхів між вершинами графа, у яких відсутні внутрішні вершини.

Ідея формування матриці $d_{ij}^{(k)}$ з $d_{ij}^{(k-1)}$ є наступною: нехай відомо найкоротші шляхи із вершини i у вершину k, із вершини k у вершину j та із вершини i у вершину j, у яких як внутрішні використано лише перші $(k-1)$ вершин (тобто відомі елементи $d_{ik}^{(k-1)}$, $d_{kj}^{(k-1)}$, $d_{ij}^{(k-1)}$). Включення у шлях із вершини i у вершину j k-ої вершини означає об'єднання шляхів $d_{ik}^{(k-1)}$ та $d_{kj}^{(k-1)}$. Тому, з врахуванням того, що граф G не містить циклів із від'ємною довжиною, на наступній ітерації матриця формується за співвідношенням: $d_{ij}^{(k)} = \min\left\{d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, d_{ij}^{(k-1)}\right\}$. Тобто ми вибираємо менший із двох шляхів:

зі включенням k-ої вершини або без неї.

Тепер можна описати кроки алгоритму.

Алгоритм Флойда-Воршелла для знаходження найкоротшого шляху між усіма парами вершин графу.

1. Ініціалізація (присвоювання початкових значень. Будуємо матрицю ваг $d_{ij}^{(0)}$ за таким правилом: $d_{ij}^{(0)} = \begin{cases} \text{weight}(i,j), \text{ якщо } (i,j) \in E; \\ \infty, \text{ якщо } (i,j) \notin E; \\ 0, \text{ якщо } i = j \end{cases}$.
2. Для $k = 1, 2, ..., n$ за рекурентним співвідношенням $d_{ij}^{(k)} = \min\left\{d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, d_{ij}^{(k-1)}\right\}$ визначаємо елементи матриці наступної ітерації.

Після закінчення цієї процедури елементи матриці $d_{ij}^{(n)}$ дорівнює довжині найкоротшого шляху з вершини i у вершину j.

Якщо під час роботи алгоритму виявиться, що $d_{ii}^{(k)} < 0,$ то в графі G існує цикл із від’ємною довжиною, який містить вершину i. Тоді роботу алгоритму потрібно припинити.

Складність алгоритму Флойда-Воршелла становить $O(n^3)$. Це більше, ніж для алгоритму Дейкстри, однак у цьому випадку одночасно обраховуються найкоротші шляхи між усіма парами вершин та ваги дуг можуть приймати від’ємні значення.

Матриця $d_{ij}^{(n)}$ містить довжини найкоротших шляхів між вершинами графа. Щоб отримати самі шляхи, потрібно, аналогічно як і в алгоритмі Дейкстри, виконати додаткові операції. Для цього, поряд з матрицею $d_{ij}^{(k)}$ формують матрицю $p_{ij}^{(k)}$ наступним чином. Початкова матриця $p^{(0)} = \begin{cases} p_{ij}^{(0)} = i, \text{ якщо } i \neq j \\ p_{ij}^{(0)} = 0, \text{ якщо } i = j \end{cases}$ Далі на кожній ітерації $p_{ij}^{(k)} = p_{ij}^{(k-1)},$ якщо

$d_{ij}^{(k)} = d_{ij}^{(k-1)}$ або $p_{ij}^{(k)} = p_{kj}^{(k-1)}$, якщо $d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$. Таким чином, елемент матриці $p_{ij}^{(k)}$ містить номер вершини, яка є перед вершиною *j* у поточному шляху з вершини *i* у вершину *j*. За допомогою матриці $p_{ij}^{(n)}$ вершини, через які проходить найкоротший шлях з вершини *i* у вершину *j*, визначають так: $i, ..., j_3, j_2, j_1, j$, де $j_1 = p_{ij}^{(n)}, j_2 = p_{ij_1}^{(n)}, j_3 = p_{ij_2}^{(n)}, ...$.

Приклад.

Розглянемо покрокове виконання ітерацій алгоритму Флойда-Воршелла для орієнтованого графа, зображеного на рис. 7.2.

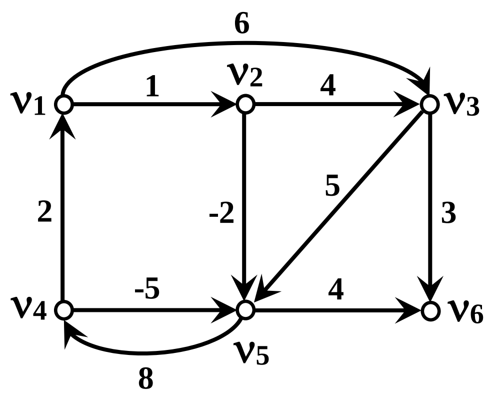


Рисунок 7.2

Нижче наведено матриці $d^{(k)}$ та $p^{(k)}$ для кожної з 6-ти ітерацій алгоритму. Жирним шрифтом у матрицях виділено елементи, що змінилися, порівняно з попередньою ітерацією.

$$d^{(0)} = \begin{pmatrix} 0 & 1 & 6 & \infty & \infty & \infty \\ \infty & 0 & 4 & \infty & -2 & \infty \\ \infty & \infty & 0 & \infty & 5 & 3 \\ 2 & \infty & \infty & 0 & -5 & \infty \\ \infty & \infty & \infty & 8 & 0 & 4 \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}; \qquad p^{(0)} = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 2 & 0 & 2 & 2 & 2 & 2 \\ 3 & 3 & 0 & 3 & 3 & 3 \\ 4 & 4 & 4 & 0 & 4 & 4 \\ 5 & 5 & 5 & 5 & 0 & 5 \\ 6 & 6 & 6 & 6 & 6 & 0 \end{pmatrix};$$

$$d^{(1)}=\begin{pmatrix} 0 & 1 & 6 & \infty & \infty & \infty \\ \infty & 0 & 4 & \infty & -2 & \infty \\ \infty & \infty & 0 & \infty & 5 & 3 \\ 2 & \mathbf{3} & \mathbf{8} & 0 & -5 & \infty \\ \infty & \infty & \infty & 8 & 0 & 4 \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}; \qquad p^{(1)}=\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 2 & 0 & 2 & 2 & 2 & 2 \\ 3 & 3 & 0 & 3 & 3 & 3 \\ 4 & \mathbf{1} & \mathbf{1} & 0 & 4 & 4 \\ 5 & 5 & 5 & 5 & 0 & 5 \\ 6 & 6 & 6 & 6 & 6 & 0 \end{pmatrix};$$

$$d^{(2)}=\begin{pmatrix} 0 & 1 & \mathbf{5} & \infty & \mathbf{-1} & \infty \\ \infty & 0 & 4 & \infty & -2 & \infty \\ \infty & \infty & 0 & \infty & 5 & 3 \\ 2 & 3 & \mathbf{7} & 0 & -5 & \infty \\ \infty & \infty & \infty & 8 & 0 & 4 \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}; \qquad p^{(2)}=\begin{pmatrix} 0 & 1 & \mathbf{2} & 1 & \mathbf{2} & 1 \\ 2 & 0 & 2 & 2 & 2 & 2 \\ 3 & 3 & 0 & 3 & 3 & 3 \\ 4 & 1 & \mathbf{2} & 0 & 4 & 4 \\ 5 & 5 & 5 & 5 & 0 & 5 \\ 6 & 6 & 6 & 6 & 6 & 0 \end{pmatrix};$$

$$d^{(3)}=\begin{pmatrix} 0 & 1 & 5 & \infty & -1 & \mathbf{8} \\ \infty & 0 & 4 & \infty & -2 & \mathbf{7} \\ \infty & \infty & 0 & \infty & 5 & 3 \\ 2 & 3 & 7 & 0 & -5 & \mathbf{10} \\ \infty & \infty & \infty & 8 & 0 & 4 \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}; \qquad p^{(3)}=\begin{pmatrix} 0 & 1 & 2 & 1 & 2 & \mathbf{3} \\ 2 & 0 & 2 & 2 & 2 & \mathbf{3} \\ 3 & 3 & 0 & 3 & 3 & 3 \\ 4 & 1 & 2 & 0 & 4 & \mathbf{3} \\ 5 & 5 & 5 & 5 & 0 & 5 \\ 6 & 6 & 6 & 6 & 6 & 0 \end{pmatrix};$$

$$d^{(4)}=\begin{pmatrix} 0 & 1 & 5 & \infty & -1 & 8 \\ \infty & 0 & 4 & \infty & -2 & 7 \\ \infty & \infty & 0 & \infty & 5 & 3 \\ 2 & 3 & 7 & 0 & -5 & 10 \\ \mathbf{10} & \mathbf{11} & \mathbf{15} & 8 & 0 & 4 \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}; \qquad p^{(4)}=\begin{pmatrix} 0 & 1 & 2 & 1 & 2 & 3 \\ 2 & 0 & 2 & 2 & 2 & 3 \\ 3 & 3 & 0 & 3 & 3 & 3 \\ 4 & 1 & 2 & 0 & 4 & 3 \\ \mathbf{4} & \mathbf{1} & \mathbf{2} & 5 & 0 & 5 \\ 6 & 6 & 6 & 6 & 6 & 0 \end{pmatrix};$$

$$d^{(5)}=\begin{pmatrix} 0 & 1 & 5 & \mathbf{7} & -1 & \mathbf{3} \\ \mathbf{8} & 0 & 4 & \mathbf{6} & -2 & \mathbf{2} \\ \mathbf{15} & \mathbf{16} & 0 & \mathbf{13} & 5 & 3 \\ 2 & 3 & 7 & 0 & -5 & \mathbf{-1} \\ 10 & 11 & 15 & 8 & 0 & 4 \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}; \qquad p^{(5)}=\begin{pmatrix} 0 & 1 & 2 & \mathbf{5} & 2 & \mathbf{5} \\ \mathbf{4} & 0 & 2 & \mathbf{5} & 2 & \mathbf{5} \\ \mathbf{4} & \mathbf{1} & 0 & \mathbf{5} & 3 & 3 \\ 4 & 1 & 2 & 0 & 4 & \mathbf{5} \\ 4 & 1 & 2 & 5 & 0 & 5 \\ 6 & 6 & 6 & 6 & 6 & 0 \end{pmatrix};$$

$$d^{(6)}=\begin{pmatrix} 0 & 1 & 5 & 7 & -1 & 3 \\ 8 & 0 & 4 & 6 & -2 & 2 \\ 15 & 16 & 0 & 13 & 5 & 3 \\ 2 & 3 & 7 & 0 & -5 & -1 \\ 10 & 11 & 15 & 8 & 0 & 4 \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}; \qquad p^{(6)}=\begin{pmatrix} 0 & 1 & 2 & 5 & 2 & 5 \\ 4 & 0 & 2 & 5 & 2 & 5 \\ 4 & 1 & 0 & 5 & 3 & 3 \\ 4 & 1 & 2 & 0 & 4 & 5 \\ 4 & 1 & 2 & 5 & 0 & 5 \\ 6 & 6 & 6 & 6 & 6 & 0 \end{pmatrix};$$

Матриця $d^{(6)}$ дає довжини найкоротших шляхів між усіма парами вершин графа, зображеного на рис. 7.2.

Знайдемо найкоротший шлях з вершини v_5 у вершину v_3, довжина якого $d_{53}^{(6)}=15$. Елемент $p_{53}^{(6)}=2$. Це означає, що у шляху з вершини v_5 у вершину v_3, вершині v_3 передує вершина v_2. Аналогічно записуємо у зворотному порядку інші вершини шуканого шляху: $p_{52}^{(6)}=1$, $p_{51}^{(6)}=4$, $p_{54}^{(6)}=5$ – початкову вершину v_5 досягнуто. Таким чином отриманий нами шлях має зворотній вигляд: $v_3 \leftarrow v_3 \leftarrow v_1 \leftarrow v_4 \leftarrow v_5$.

<u>Зауваження</u>. Щоб відобразити шлях у звичайному вигляді $v_5 \to v_4 \to v_1 \to v_2 \to v_3$ доцільно використати структуру даних типу *стек* з бібліотеки `List` (див. лабораторну роботу № 4), записавши в нього вершини шляху, а потім видобувши їх у зворотному порядку.

Алгоритми Дейкстри та Флойда-Воршелла можна застосовувати і до неорієнтованих графів. Для цього кожне неорієнтоване ребро потрібно розглядати як пару протилежно напрямлених дуг з тією самою вагою. Очевидно, що в такому випадку від'ємні ваги дуг не допускаються, оскільки це приводить до появи циклу від'ємної довжини.

Хід роботи:

Частина 2. Пошук найкоротших шляхів у графі за алгоритмом Флойда-Воршелла

1. Створити новий проект `Lab_7_2`.
2. Для заданого викладачем варіанту графа задати його матрицю ваг (суміжності) $d^{(0)}$ розміру $n \times n$, де n – кількість вершин у графі.
3. Згенерувати матрицю $p^{(0)}$ за правилом $p^{(0)} = \begin{cases} p_{ij}^{(0)} = i, \text{ якщо } i \neq j \\ p_{ij}^{(0)} = 0, \text{ якщо } i = j \end{cases}$.
4. Вивести матриці $d^{(0)}$ та $p^{(0)}$ на екран.
5. Реалізувати процедуру генерування матриць $d^{(k)}$ та $p^{(k)}$ для $k = \overline{1,n}$ за правилом:

$$d_{ij}^{(k)} = \min\left(d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, d_{ij}^{(k-1)}\right) \text{ та}$$

$$p^{(k)} = \begin{cases} p_{ij}^{(k)} = p_{ij}^{(k-1)}, \text{ якщо } d_{ij}^{(k)} = d_{ij}^{(k-1)} \\ p_{ij}^{(k)} = p_{kj}^{(k-1)}, \text{ якщо } p_{ij}^{(k)} = p_{ik}^{(k-1)} + p_{kj}^{(k-1)} \end{cases}.$$

   Передбачити можливість виявлення у графі циклу від'ємної довжини.
6. В циклі по $k = \overline{1,n}$ обчислити матриці $d^{(k)}$ і $p^{(k)}$ та вивести їх на екран (або записати у текстовий файл).
7. Реалізувати процедуру відображення найкоротшого шляху між двома заданими вершинами при використанні матриці $p^{(n)}$. Передбачити можливість відсутності шляху між вершинами у графі (випадок слабо зв'язного графа).
8. Продемонструвати роботу програми.