Тема 30. Алгоритми пошуку найкоротших шляхів

30.1. Пошук маршруту у графі

При розв'язанні широкого кола прикладних задач нерідко виникає необхідність знайти маршрут, що зв'язує задані вершини в графі G. Наведемо алгоритм розв'язання такої задачі. В ньому задача зводиться до пошуку маршруту в зв'язаному графі $G = (V, E)$, який з'єднує задані вершини v, $u \in V$, де $v \neq u$.

Алгоритм Террі знаходження маршруту

У зв'язаному графі завжди можна знайти такий маршрут, що зв'язує дві задані вершини v та u, якщо, виходячи з вершини v і здійснюючи послідовний перехід від кожної досягнутої вершини до суміжної з нею, керуватися такими правилами:

1) йдучи по довільному ребру, кожний раз відмічати напрямок, в якому воно було пройдене;
2) виходячи з деякої вершини v_1, завжди рухатися тільки по тому ребру, яке не було пройдене або було пройдене у зворотному напрямку;
3) для кожної вершини v_1, відмінної від v, відмічати те ребро, яке першим заходить у v_1, якщо вершина v_1, зустрічається вперше;
4) виходячи з деякої вершини v_1, відмінної від v, по першому ребру, яке заходить у v_1, рухатися лише тоді, коли немає інших можливостей.

Обґрунтування алгоритму. Припустимо, що, керуючись цим алгоритмом, зупинимося в деякій вершині w (не досягши вершини u), а всі ребра, інцидентні w, вже пройдено в напрямку з w (тоді внаслідок правила 2 вже не можна вийти з w). Покажемо, що в цьому випадку: а) вершина w збігається з v; б) всі вершини графу G пройдено.

Доведемо спочатку твердження а). Якщо вершина w не збігається з v, то нехай у вершині w ми побували k разів (включаючи останній). Тоді ребра, інцидентні w, були пройдені k разів у напрямку до w, $k - 1$ разів у напрямку з w (оскільки кількість заходів у w, за винятком останнього, відповідає кількості виходів із цієї вершини). Таким чином, використовуючи те, що за припущенням були пройдені всі ребра, інцидентні вершині w, в напрямку з w, а також те, що з урахуванням правила 2 по кожному ребру, інцидентному w, маємо $d(w) = k - 1$, а це суперечить тому, що у напрямку до w були пройдені k різних ребер (згідно з правилом 2); отже $d(w) \geq k$. Одержана суперечність підтверджує, що $w = v$.

Доведемо тепер правильність б). Нехай (за твердженням а)): $v_1, v_2, \ldots, v_k$, де $v_1 = v_k = v$ – послідовність вершин, розташованих у тому самому порядку, в якому ми рухалися, діючи згідно з алгоритмом. Очевидно, ця послідовність є маршрутом у графі G. Покажемо, що цей маршрут містить усі вершини графу G. Спочатку доведемо, що кожне ребро, інцидентне будь-якій вершині v_j, $1 \leq j \leq k$, було пройдене по одному разу в обох напрямках. Доведення проведемо індукцією за кількістю вершин j.

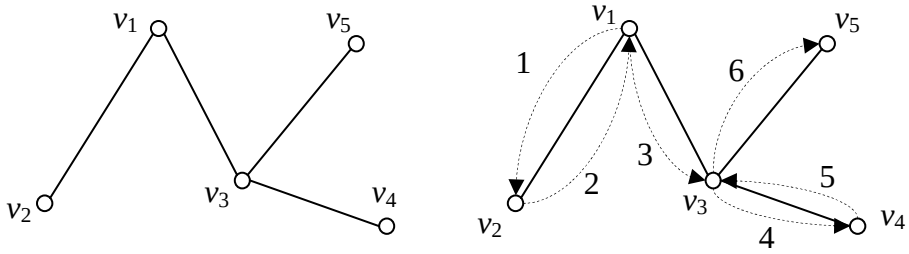
Оскільки в замкненому маршруті для кожної вершини, яка міститься в ньому, кількість виходів з неї дорівнює кількості заходів у неї, внаслідок того, що згідно з твердженням а) і правилом 2 всі ребра, інцидентні вершині $v = v_1$, були пройдені по одному разу в напрямку з v (тобто ми $d(v)$ разів виходили з v), встановлюємо, що рівно $d(v)$ разів ми заходили у v, а оскільки внаслідок правила 2 кожний такий захід у v здійснювався по новому ребру, всі ребра, інцидентні вершині $v = v_1$, були пройдені по разу в обох напрямках.

Припустимо, що при деякому j, $2 \leq j \leq k$, твердження, яке доводиться, справджується для всіх вершин $v_1, \ldots, v_{j-1}$. Доведемо його для вершини v_j. Якщо при деякому $i<j$ виконується рівність $v_i = v_j$, то правильність твердження, що доводиться для вершини v_j, випливає з того, що за індуктивним припущенням воно випливає з v_i. Нехай тепер для всіх $i \in \{1, 2, \ldots, j-1\}$ $v_i \neq v_j$, тобто вершина v_i зустрілася вперше. Тоді (v_{j-1}, v_j) – перше ребро, яке заходить у вершину v_j, за індуктивним припущенням воно буде пройдене в обох напрямках, що з урахуванням правила 4 можливо лише тоді, коли всі інші ребра, інцидентні v_j, будуть пройдені в напрямку з v_j. Далі, оскільки в замкненому маршруті, як уже зазначалося, для кожної вершини, яка

міститься в цьому маршруті, кількість виходів із неї дорівнює кількості заходів в неї, використовуючи правило 2, встановлюємо, що всі ребра, інцидентні v_j, будуть пройдені по разу в обох напрямках.

Отже, кожну вершину в маршруті v_1, v_2, …, v_k проходимо разом з усіма суміжними їй вершинами, звідки внаслідок зв'язності графа G випливає, що цей маршрут проходить через усі вершини графу G, а це суперечить початковому припущенню, за яким вершин u не була досягнута. ►

Наведемо приклад використання алгоритму Террі. Необхідно знайти у графі G (рис. 30.1, *а*) маршрут, який з'єднує вершини v_1 та v_5.



а) б)

Рис. 30.1. Приклад роботи алгоритму Террі.

Пошук вершини v_5 у G будемо здійснювати так, неначе нічого невідомо про цей граф (це можна порівняти з тим, що G – це лабіринт, v_1 та v_5 – вхід та вихід з лабіринту. На рис. 30.1, *б* показаний один із можливих варіантів руху по графу G згідно з алгоритмом Террі. Пронумерованими штриховими дугами зображено схему руху по графу G. Ця схема руху відповідає маршруту (v_1, v_2, v_1, v_3, v_4, v_3, v_5). Зазначимо, що після того, як із вершини v_1 зайшли у вершину v_3 (дуга 3), внаслідок правила 4 неможна повернутися у v_1, оскільки існують інші можливості, а (v_1, v_3) є першим ребром, що заходить у v_3. Далі, після того, як із вершини v_4 зайшли у вершину v_3 (дуга 5), внаслідок правила 2 не можна рухатися до вершини v_1, і, таким чином, залишається єдина можливість – рухатися до вершини v_5.

30.2. Пошук відстані між вершинами графу

Розглянемо деякі властивості мінімальних (шляхів) маршрутів.

Означення 30.1. Назвемо **образом вершини** x в орієнтованому графі G множину кінців дуг, початком яких є вершина x (позначається $D(x)$), а множину початків дуг, кінцем яких є вершина x, назвемо **прообразом вершини** x (позначається $D^{-1}(x)$).

Зрозуміло, що $D(x) \cup D^{-1}(x) = \Gamma(x)$, де $\Gamma(x)$ – множина суміжності вершини x.

Нехай $G = (V, E)$ – орієнтований граф з n вершинами ($n \geq 2$), а v, u – задані вершини з V, де $v \neq u$. Опишемо алгоритм пошуку відстані та відповідного їй мінімального шляху з v до u в орієнтованому графі G. Цей алгоритм також має назву хвильового.

Алгоритм (хвильовий)

1. Позначаємо вершину v індексом 0, а вершини, що належать образу вершини v, – індексом 1. Множину вершин з індексом k позначаємо $F_k(v)$. Вважаємо $k = 1$.
2. Якщо $F_k(v) = \varnothing$ або виконується $k = n - 1$ і $u \notin F_k(v)$, то вершина u є незв'язаною з v і робота алгоритму на цьому завершується. В іншому випадку перейти до пункту 3.
3. Якщо $u \notin F_k(v)$, то переходимо до пункту 4. В іншому випадку існує шлях із v до u завдовжки k, причому цей шлях є мінімальним. Послідовність вершин v, u_1, u_2,…, u_{k-1}, u, де

$$u_{k-1} \in F_{k-1}(v) \cap D^{-1}(u),$$
$$u_{k-2} \in F_{k-2}(v) \cap D^{-1}(u_{k-1}),$$
$$\ldots\ldots\ldots\ldots$$
$$u_1 \in F_1(v) \cap D^{-1}(u_2),$$

і є шуканим мінімальним шляхом з v у w. На цьому робота алгоритму завершується.

4. Позначаємо індексом $k + 1$ всі непозначені вершини, які належать образу множини вершин з індексом k. Множину вершин з індексом $k + 1$ позначаємо $F_{k+1}(v)$. Збільшуємо індекс k на 1 і переходимо до пункту 2.

Назва алгоритму – хвильовий – пов'язана з тим, що визначення індексів k вершин графу G відбувається, як розповсюдження з початкової вершини v певної хвилі, яка спрямовується за напрямком дуг. Коли хвиля дійде до кінцевої вершини u, це буде означати, що алгоритм закінчив свою роботу. Значення індексу, „принесеного хвилею", у вершині u буде відповідати довжині знайденого маршруту. А для того, щоб визначити цей маршрут (послідовність вершин), потрібно з кінцевої вершини u повертатися в зворотному до розповсюдження хвилі напрямку і відзначати послідовно одну довільну вершину зі значеннями індексу $k–1, k–2, \ldots, 1, 0$. Зрозуміло, що вершина з індексом 0, – це початкова вершина v.

Вершини $u_1, u_2, \ldots, u_{k–1}$, взагалі, можуть бути визначені неоднозначно. Ця неоднозначність відповідає випадкам, коли існує кілька різних мінімальних шляхів з v до u в орграфі G.

Наприклад, визначимо мінімальний шлях з v_1 до v_6 в орієнтованому графі G, заданому матрицею суміжності (відповідний граф представлено на рис. 30.2):

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0	0	0	1	1	0
v_2	1	0	0	0	0	1
v_3	0	1	0	0	0	1
v_4	0	1	0	0	1	0
v_5	1	0	1	0	0	0
v_6	0	0	1	0	1	0

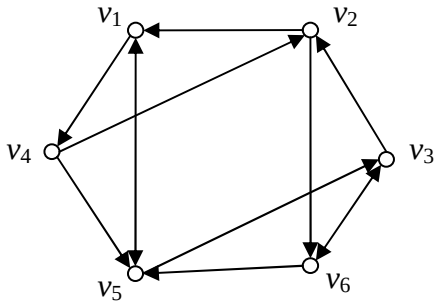


Рис. 30.2. Приклад роботи хвильового алгоритму.

Діючи згідно з хвильовим алгоритмом, послідовно знаходимо $F_1(v_1) = \{v_4, v_5\}$; $F_2(v_1) = D(F_1(v_1)) \setminus \{v_1, v_4, v_5\} = \{v_2, v_3\}$; $F_3(v_1) = D(F_2(v_1)) \setminus \{v_1, v_2, v_3, v_4, v_5\} = \{v_6\}$. Таким чином, $v_6 \in F_3(v_1)$, а, отже, за пунктом 3 існує шлях з v_1 до v_6 завдовжки 3, і цей шлях є мінімальним.

Знайдемо тепер мінімальний шлях із v_1 до v_6. Визначимо множину

$$F_2(v_1) \cap D^{-1}(v_6) = \{v_2, v_3\} \cap \{v_2, v_3\} = \{v_2, v_3\}.$$

Виберемо будь-яку вершину зі знайденої множини, наприклад, v_3. Визначимо далі множину

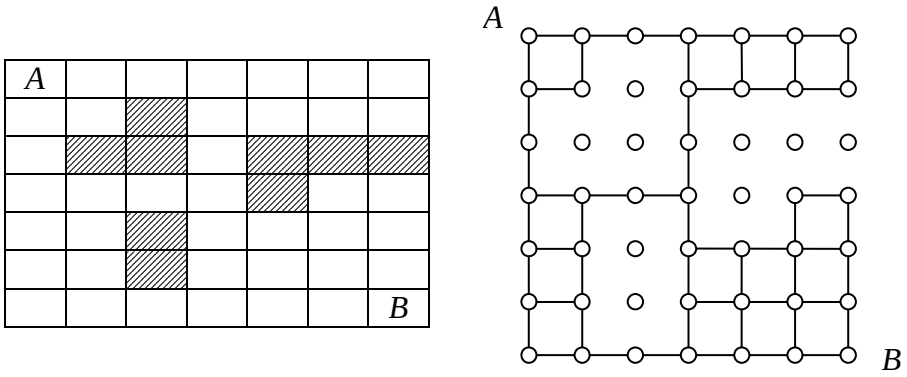
$$F_1(v_1) \cap D^{-1}(v_3) = \{v_4, v_5\} \cap \{v_4, v_5, v_6\} = \{v_4, v_5\}.$$

Виберемо будь-яку вершину зі знайденої множини, наприклад, v_5. Тоді (v_1, v_5, v_3, v_6) – шуканий мінімальний шлях з v_1 до v_6 в орієнтованому графі G, а відстань між v_1 та v_6 дорівнює 3.

Очевидно, хвильовий алгоритм може застосовуватися не тільки для орієнтованих, а й для неорієнтованих графів. В останньому випадку, пересування з однієї вершини до іншої можливі в обидві сторони.

Хвильовий алгоритм широко застосовується у розробці комп'ютерних ігор – коли необхідно визначити оптимальний маршрут пересування гравця або певного „юніта" з однієї

точки віртуальної місцевості (карти) до іншої. Наведемо приклад такої задачі. Нехай потрібно знайти найкоротший маршрут з точки A до точки B на карті, яка зображена на рис. 30.3, *а*. На ній заштриховані комірки відповідають певним перепонам на шляху, тобто в цих частинах місцевості гравець не зможе пройти. Також будемо вважати, що гравець може пересуватись тільки по вертикалі та горизонталі. Відповідний цієї карті граф представлено на рис. 30.3, *б*.

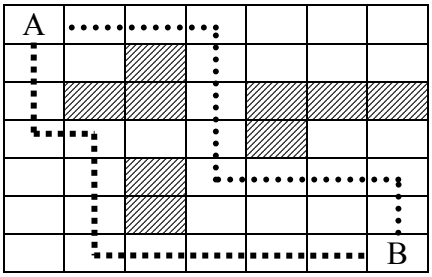


а) б)

Рис. 30.3. Приклад карти території та відповідного графу.

Після роботи хвильового алгоритму отримаємо наступні індекси вершин – комірок карти (рис. 30.4, *а*). На рис. 30.4, *б* зображено два зі знайдених маршрутів з вершини A у вершину B. Як можна побачити, довжина знайденого маршруту дорівнює 12.

0	1	2	3	4	5	6
1	2		4	5	6	7
2			5			
3	4	5	6		10	11
4	5		7	8	9	10
5	6		8	9	10	11
6	7	8	9	10	11	12



а) б)

Рис. 30.4. Робота хвильового алгоритму для графу на рис. 30.3.

Можна зробити процес пошуку найкоротшого маршруту за допомогою хвильового алгоритму більш економнішим, а відтак, й більш швидким. Для цього будемо розповсюджувати хвилю не тільки з початкової вершини A (перша хвиля), а й з кінцевої вершини B (друга хвиля). Для того, щоб відрізняти індекси першої хвилі від другої, індекси останньої будемо позначати зі штрихом. Робота модифікованого алгоритму закінчується коли обидві хвилі зустрінуться (рис. 30.5).

0	1	2	3	4	5	6
1	2		4	5	6	
2			5			
3	4	5	6/6’		4’	3’
4	5		5’	4’	3’	2’
5	6/6’		4’	3’	2’	1’
6/6’	5’	4’	3’	2’	1’	0’

Рис. 30.5. Двонаправлений хвильовий алгоритм.

Комірки, де дві хвилі зустрічаються, позначені подвійними лініями. З порівняння рис. 30.4 та 30.5 видно, що знайдені найкоротші маршрути співпадають. І хоча в цьому прикладі

економія склала всього лиш одну комірку (яка не була відмічена), можна зрозуміти, що на більш складних, тобто насичених „перепонами", картах робота модифікованого хвильового алгоритму буде більш ефективнішою за простий хвильовий алгоритм.

30.3. Зважені графи

У реальних задачах на графах часто потрібно брати до уваги додаткову інформацію – фактичну віддаль між окремими пунктами, вартість проїзду, час проїзду тощо. Для цього використовують поняття зваженого графа.

Означення 30.2. **Зваженим** називають граф, кожному ребру e якого приписано дійсне число $w(e)$. Це число називають **вагою ребра** e. Аналогічно означають **зважений орієнтований граф**: це такий орієнтований граф, кожній дузі e якого приписане дійсне число $w(e)$, яке називається **вагою дуги**.

Розглянемо два способи зберігання зваженого графа $G = (V, E)$ в пам'яті комп'ютера. Нехай $|V| = n$, $|E| = m$.

Перший – подання графу матрицею ваг W, яка являє собою аналог матриці суміжності. Її елемент $w_{ij} = w(v_i, v_j)$, якщо ребро або дуга $(v_i, v_j) \in E$. Якщо ж ребро або дуга $(v_i, v_j) \notin E$, то $w_{ij} = 0$ чи $w_{ij} = \infty$ залежно від розв'язуваної задачі.

Другий спосіб – поданням графу списком ребер. Для зваженого графу під кожний елемент списку E можна відвести три комірки – дві для ребра й одну для його ваги, тобто всього потрібно $3m$ комірок.

Означення 30.3. **Довжиною шляху в зваженому графі** називають суму ваг ребер (дуг), які утворюють цей шлях. Якщо граф не зважений, то вагу кожного ребра (кожної дуги) уважають рівною одиниці та отримають раніше введене поняття довжини шляху як кількості ребер (дуг) у ньому.

30.4. Алгоритм Дейкстри

Задача про найкоротший шлях полягає в знаходженні найкоротшого шляху від заданої початкової вершини a до заданої вершини z. Наступні дві задачі – безпосередні узагальнення сформульованої задачі про найкоротший шлях.

1. Для заданої початкової вершини a знайти найкоротші шляхи від a до всіх інших вершин.
2. Знайти найкоротші шляхи між усіма парами вершин.

Виявляється що майже всі методи розв'язання задачі про найкоротший шлях від заданої початкової вершини a до заданої вершини z також дають змогу знайти й найкоротші шляхи від вершини a до всіх інших вершин графа. Отже, за їх допомогою можна розв'язати задачу 1 із невеликими додатковими обчислювальними витратами. З іншого боку, задачу 2 можна розв'язати або n разів застосувавши алгоритм задачі 1 із різними початковими вершинами, або один раз застосувавши спеціальний алгоритм.

Найефективніший алгоритм визначення довжини найкоротшого шляху від фіксованої вершини до будь-якої іншої запропонував 1959 р. датський математик Е. Дейкстра. Цей алгоритм застосований лише тоді, коли вага кожного ребра (дуги) додатна. Опишемо докладно цей алгоритм для орієнтованого графа.

Нехай $G = (V, E)$ – зважений орієнтований граф, $w(v_i, v_j)$ – вага дуги (v_i, v_j). Почавши з вершини a, знаходимо віддаль від a до кожної із суміжних із нею вершин. Вибираємо вершину, віддаль від якої до вершини a найменша; нехай це буде вершина v^*. Далі знаходимо віддалі від вершини a до кожної вершини суміжної з v^* вздовж шляху, який проходить через вершину v^*. Якщо для якоїсь із таких вершин ця віддаль менша від поточної, то замінюємо нею поточну віддаль. Знову вибираємо вершину, найближчу до a та не вибрану раніше; повторюємо процес.

Описаний процес зручно виконувати за допомогою присвоювання вершинам міток. Є мітки двох типів: тимчасові та постійні. Вершини з постійними мітками групуються у множину M, яку називають **множиною позначених вершин**. Решта вершин має тимчасові мітки, і множину таких вершин позначимо як T, $T = V \setminus M$. Позначатимемо мітку (тимчасову

чи постійну) вершини v як $l(v)$. Значення постійної мітки $l(v)$ дорівнює довжині найкоротшого шляху від вершини a до вершини v, тимчасової – довжині найкоротшого шляху, який проходить лише через вершини з постійними мітками.

Фіксованою початковою вершиною вважаємо вершину a; довжину найкоротшого шляху шукаємо до вершини z (або до всіх вершин графа). Тепер формально опишемо алгоритм Дейкстри.

Алгоритм Дейкстри

1. Присвоювання початкових значень. Виконати $l(a) = 0$ та вважати цю мітку постійною. Виконати $l(v) = \infty$ для всіх $v \neq a$ й уважати ці мітки тимчасовими. Виконати $x = a$, $M = \{a\}$.
2. Оновлення міток. Для кожної вершини $v \in \Gamma(x) \setminus M$ замінити мітки: $l(v) = \min\{ l(v), l(x) + w(x, v) \}$, тобто оновлювати тимчасові мітки вершин, у які з вершини x іде дуга.
3. Перетворення мітки в постійну. Серед усіх вершин із тимчасовими мітками знайти вершину з мінімальною міткою, тобто знайти вершину v^* з умови $l(v^*) = \min\{l(v)\}$, $v \in T$, де $T = V \setminus M$.
4. Уважати мітку вершини v^* постійною й виконати $M = M \cup \{v^*\}$; $x = v^*$ (вершину v^* включено в множину M).
5. а) Для пошуку шляху від a до z: якщо $x = z$, то $l(z)$ – довжина найкоротшого шляху від a до z, зупинитись; якщо $a \neq z$, то перейти до кроку 2.
   б) Для пошуку шляхів від a до всіх вершин: якщо всі вершини отримали постійні мітки (включені в множину M), то ці мітки дорівнюють довжинам найкоротших шляхів, зупинитись; якщо деякі вершини мають тимчасові мітки, то перейти до кроку 2.

Обґрунтування алгоритму Дейкстри. Для доведення коректності алгоритму Дейкстри достатньо відмітити, що при кожному застосуванні кроку 3, вершина v^* вибирається як мінімальна серед вершин з тимчасовими мітками. Для визначення значень цих міток (крок 2) використовувались вершини з постійними мітками, тобто вершини, для яких вже відомий найкоротший маршрут.

Проведемо доведення по індукції за кількістю застосувань кроку 3. В перший раз на кроці 3 обираються такі вершини $v \in \Gamma(a)$, тобто вершини суміжні з початковою вершиною a, для якої найкоротший шлях порожній і дорівнює 0. Нехай це справджується для кроків 2, 3, …, $k - 1$. На k-й ітерації кроку 3 обирається така вершина v^*, для якої $l(v^*) = \min\{l(v)\}$, $v \in T$. Відмітимо, що якщо відомий шлях, який проходить через вершину з постійною міткою, то тим самим відомий найкоротший шлях.

Припустимо, що $l(v^*)$ більше за довжину найкоротшого шляху від a до v^* (позначимо цей шлях P^*, а його довжину – $w(P^*)$), тобто $l(v^*) > w(P^*)$. Тоді на цьому шляху мають бути вершини з тимчасовими мітками. Розглянемо найпершу серед них на цьому шляху – вершину $v' \in T$. Частину шляху P^* від a до v' позначимо P', довжину P' позначимо відповідно $w(P')$. Маємо: $l(v') = w(P') \leq w(P^*) < l(v^*)$, тобто $l(v') < l(v^*)$. Але це суперечить способу обрання нової вершини з постійною міткою на кроці 3.

Доведення випадку б), тобто задачі пошуку найкоротших шляхів від вершини a до всіх інших вершин графа, є аналогічним. ►

Якщо граф подано матрицею суміжності, складність алгоритму Дейкстри становить $O(n^2)$. Коли кількість дуг значно менша ніж n^2, то найкраще подавати орієнтований граф списками суміжності. Тоді алгоритм можна реалізувати зі складністю $O(m \lg n)$, що в цьому разі істотно менше ніж $O(n^2)$.

Алгоритм Дейкстри дає змогу обчислити довжину найкоротшого шляху від початкової вершини a до заданої вершини z. Для знаходження самого шляху потрібно лише збільшувати вектор вершин, з яких найкоротший шлях безпосередньо потрапляє в дану вершину. Для цього з кожною вершиною v графу G, окрім вершини a, зв'язують іще одну мітку – $\theta(v)$. Крок 2 модифікують так. Для кожної вершини $v \in \Gamma(x) \setminus M$ якщо $l(v) > l(x) + w(x,v)$, то

$l(v) = l(x) + w(x,v)$ та $\theta(v) = x$, а ні, то не змінювати $l(v)$ та $\theta(v)$. Коли мітка $l(v)$ стане постійною, найкоротший $\langle a, v\rangle$–шлях буде потрапляти у вершину v безпосередньо з вершини x. Із постійних міток $l(v)$ та $\theta(v)$ утворюємо вектори l і θ.

Знайдемо довжину найкоротшого шляху від початкової вершини a до вершини z у графі на рис. 30.6, *а*. Послідовність дій зображено на рис. 30.6, *б – е*, мітки записано в дужках біля кожної вершини. Вершини, які включені в множину M, обведено колами; мітки таких вершин оголошують постійними. У процесі роботи алгоритму будують два вектори: вектор l постійних міток (довжини найкоротших шляхів від вершини a до даної вершини) і вектор θ вершин, з яких у дану вершину безпосередньо потрапляє найкоротший шлях. У табл. 30.1 у першому рядку містяться довільно впорядковані вершини графу, у другому – відповідні постійні мітки (компоненти вектору l), а в третьому – компоненти вектору θ.

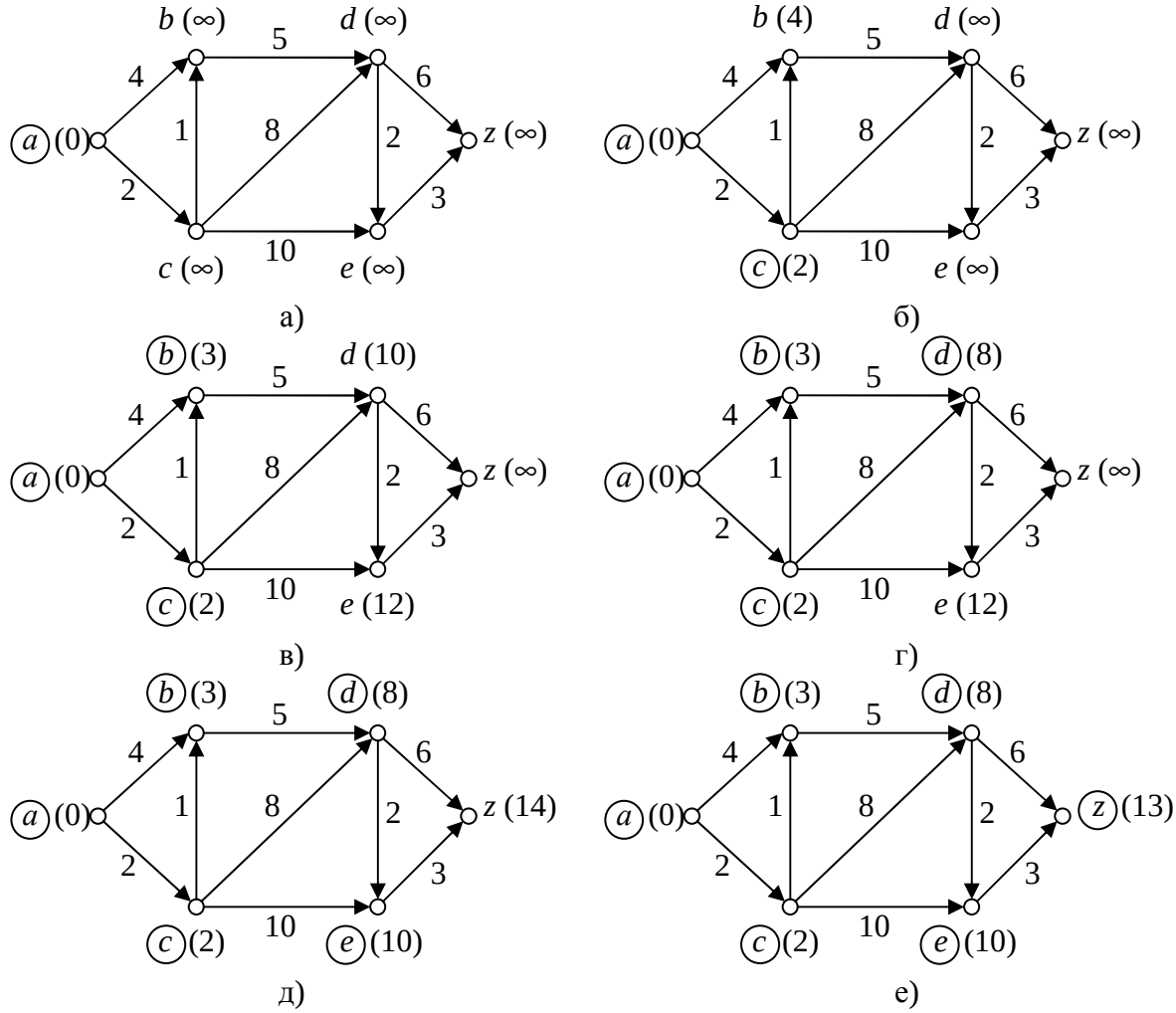


Рис. 30.6. Приклад роботи алгоритму Дейкстри.

Вершини графа (елементи множини V)	a	b	c	d	e	z
Вектор l (постійні мітки вершин)	0	3	2	8	10	13
Вектор θ (вершини, з яких у дану вершину заходить найкоротший шлях)	-	c	a	b	d	e

Табл. 30.1.

Постійна мітка вершини z дорівнює 13. Отже, довжина найкоротшого шляху від a до z дорівнює 13. Сам шлях знаходять за допомогою першого й третього рядків таблиці та будують у зворотному порядку. Кінцева вершин – z; у неї потрапляємо з вершини e (див. вектор θ). У вершину e потрапляємо з вершини d, у d – з b та продовжуємо цей процес до вершини a: $z \leftarrow e \leftarrow d \leftarrow b \leftarrow c \leftarrow a$. Отже, найкоротший шлях такий: a, c, b, d, e, z.

30.5. Алгоритм Белмана-Форда

Як зазначалось вище, алгоритм Дейкстри не знаходить коректно найкоротший шлях у випадку, коли граф містить ребра з негативними вагами. Пропонуємо самостійно в цьому переконатись шляхом побудови прикладу графу з негативними ребрами так, щоб алгоритм Дейкстри давав збій.

Для розв'язку задачі знаходження найкоротших шляхів у графах з негативними вагами ребер може використовуватись алгоритм Белмана-Форда. Цей алгоритм був запропонований незалежно Річардом Белманом та Лестером Фордом у 1958 та 1956 роках. Він дозволяє знаходити найкоротший шлях від заданої вершини до всіх інших вершин графу у випадку існування в графі ребер з негативними вагами. Єдиним обмеженням застосування цього алгоритму є відсутність негативних циклів у графі. Дійсно, якщо в графі є хоча б один цикл, сумарна вага ребер якого є від'ємною, то це дозволить нам знайти маршрут з якою завгодно малою довжиною між двома вершинами в графі. Алгоритм Белмана-Форда у випадку існування негативних циклів в графі дозволяє визначити цей факт та коректно завершити свою роботу.

Алгоритм Белмана-Форда заснований на ідеях динамічного програмування. В двох словах ідею динамічного програмування можна описати наступним чином. Початкова задача розбивається на менші підзадачі, які розв'язуються рекурсивно. Пізніше розв'язок початкової задачі отримується із розв'язку підзадач. Причому кожна з окремих підзадач може розв'язуватись більше ніж один раз. Для того щоб уникнути зайвою роботи в алгоритмах динамічного програмування вводяться спеціальні таблиці (матриці), які зберігають значення розв'язків вже обрахованих задач. Через це часто такі алгоритми вимагають значних ресурсів пам'яті для збереження відповідних значень.

Наразі припустимо, що у вхідному графі G немає негативних циклів. При цьому потрібно знайти відстані найкоротших маршрутів від заданої вершини a до всіх інших вершин графу. Позначимо через $L_{i,v}$ довжину мінімального маршруту від a до v, який містить не більше i ребер. Будемо вважати, що якщо такого маршруту не існує, то значення $L_{i,v}$ дорівнюватиме $+\infty$.

Тепер розглянемо як може бути отримане значення $L_{i,v}$ з величин $L_{(i-1),u}$, тобто з довжин мінімальних маршрутів між a та деякими вершинами u, в яких використовується не більше $(i-1)$ ребер. Перший варіант – коли просто $v = u$, тобто $L_{i,v} = L_{(i-1),v}$. Другий варіант – коли ми розглядаємо маршрут від a до u, який містить не більше $(i-1)$ ребер, а також ребро (u, v). Ми шукаємо найкоротший маршрут, а тому нас цікавить мінімальне з усіх можливих значень, які отримуються у другому варіант, а також значення з першого варіанту. Таким чином

$$L_{i,v} = \min\left\{ \begin{array}{c} L_{(i-1),v} \\ \min\limits_{(u,v)\in E}\left\{L_{(i-1),u} + w(u,v)\right\} \end{array} \right\}. \qquad (30.1)$$

З припущення, що у графі немає негативних циклів, випливає, що найкоротші маршрути не повинні містити жодного циклу. А відтак найбільша кількість ребер, яка може бути присутня у маршрутах дорівнює $n-1$, де n – кількість вершин графу. З цього факту та наведеної вище рекурентної формули слідує, що довжина найкоротшого маршруту від початкової вершини a до деякої вершини v у графі буде визначитись величиною $L_{(n-1),v}$. Щоб її обрахувати потрібно вирахувати усі значення $L_{i,v}$ для всіх $i = 1, 2, \ldots, n-1$ та всіх вершин $v \in V$. Саме це і відповідає ідеї динамічного програмування.

Для запису псевдокоду алгоритму Белмана-Форда використаємо двовимірний масив (матрицю) A розмірності $n \times n$, значення $A[i, v]$ якого будуть відповідати величинам $L_{i,v}$ розглянутим раніше. Для базового випадку $i = 0$ значення масиву A ініціалізуються наступним

чином: $A[0, a] = 0$ та $A[i, v] = +\infty$ для всіх $v \in V$ відмінних від a. Нижче наведений псевдокод алгоритму Белмана-Форда.

```
1  for v = 1 to n
2      A[0, v] ← +∞
3  A[0, a] ← 0
4  for i = 1 to n-1
5      for v = 1 to n
6          A[i, v] = A[i-1, v]
7      for всіх ребер (u, v)
8          if A[i, v] > A[i-1, u] + w(u, v)
9              then A[i, v] ← A[i-1, u] + w(u, v)
```

Як видно з наведеного лістингу, формула (30.1) замінена одним внутрішнім циклом **for** у рядку 7. Дійсно, під час обрахунку формули (30.1) для всіх $v \in V$ будуть переглянуті всі ребра графу (u, v). Відтак це можна зробити явно за допомогою одного окремого циклу.

Час роботи алгоритму Белмана-Форда становить $O(mn)$, де n – кількість вершин та m – кількість ребер графу G. Це більше, ніж для алгоритму Дейкстри, для якого час роботи визначається як $O(m \lg n)$.

Якщо необхідно визначити не тільки відстані найкоротших маршрутів від заданої вершини до всіх інших вершин графу, а й, власне, самі маршрути, то для цього потрібно зберігати вершини-попередники для кожної вершини. Позначимо такі вершини через $P[v]$ – вершина, яка передує вершині v у найкоротшому маршруті (аналогічно до вектору θ в алгоритмі Дейкстри). Масив P ініціалізується нульовими значеннями. Тепер, коли ми оновлюємо значення $A[i, v]$ у рядку 9, необхідно також оновлювати значення $P[v]$, яке буде дорівнювати вершині u. По закінченню роботи алгоритму маршрут від початкової вершини a до довільної вершини v відновлюється ззаду наперед за допомогою відповідних значень масиву P.

Розглянемо приклад на рис. 30.7. Початковий граф із ініціалізованими значеннями $A[0, v]$ наведений у частині *а* цього рисунку разом із початковими значеннями масиву P. На першій ітерації ($i = 1$) розглядаються маршрути від вершини a довжиною 1 (рис. 30.7, *б*): до вершини b (довжина 6) та вершини c (довжина 7). При цьому для вершин b та c значення масиву P стає рівним вершині a. На другій ітерації ($i = 2$; рис. 30.7, *в*) розглядаються маршрути довжиною 2. Тепер ми вже можемо потрапити у вершину d, причому двома шляхами: через вершину b (довжина $11 = 6 + 5$) та через вершину c (довжина $4 = 7 - 3$); з них обирається мінімальний і $P[d] = c$. Так само до вершини e можна дійти двома шляхами: через b (довжина $2 = 6 - 4$) та через c (довжина $16 = 7 + 9$); обирається мінімальний і $P[e] = b$. На третій ітерації ($i = 3$; рис. 30.7, *г*) покращується значення для вершини b, адже тепер ми можемо розглянути маршрут довжиною 3: $a - c - d - b$, довжина якого 2 ($P[b]$ стає рівним d). На останній ітерації оновлюється значення для вершини e: завдяки ребру (b, e) воно стає рівним –2, а $P[e]$ лишається рівним b.

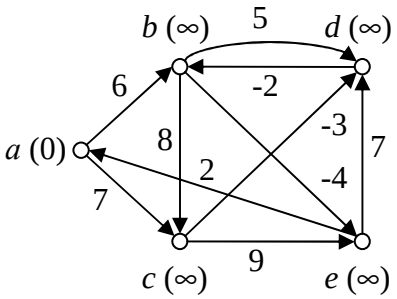
Можна модифікувати алгоритм Белмана-Форда, щоб він закінчував свою роботу раніше. Дійсно, якщо на деякі ітерації i виконується умова $A[i, v] = A[i - 1, v]$ для всіх $v \in V$, то це означає, що всі подальші ітерації ніяким чином не зможуть змінити значення $A[i, v]$. А отже, в цьому випадку роботу алгоритму можна закінчити раніше.

Для того, щоб алгоритм Белмана-Форда міг визначати негативні цикли в графі потрібно його трохи модифікувати. Припустимо замість того, щоб спинитись на ітерації $i = n - 1$, ми виконаємо ще одну додаткову ітерацію $i = n$. Інтуїтивно зрозуміло (і це також легко довести формально), що якщо граф G містить хоча б один негативний цикл, то значення $A[n, v]$ повинні стати меншими за значення $A[n - 1, v]$ для деяких $v \in V$. Отже, в кінець вище наведеного псевдокоду алгоритму Белмана-Форда потрібно додати такі рядки:

```
10 for всіх ребер (u, v)
11     if A[n-1, v] > A[n-1, u] + w(u, v)
```

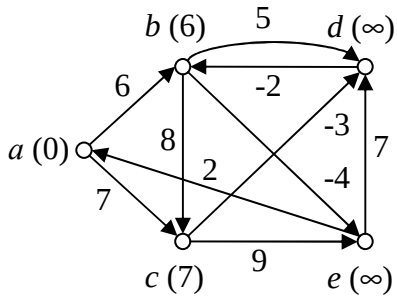
```
12          then error "Граф містить негативні цикли"
```

Час роботи при цьому лишається незмінним: O(*mn*).



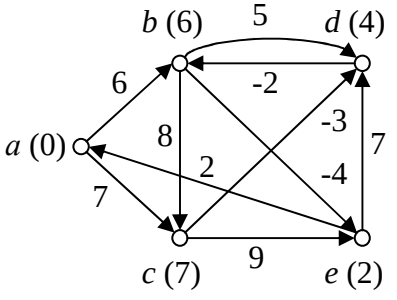
v	*P*(*v*)
a	–
b	–
c	–
d	–
e	–

а) $i = 0$



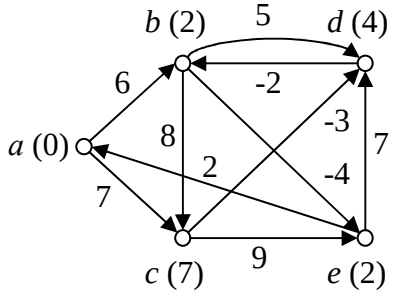
v	*P*(*v*)
a	–
b	*a*
c	*a*
d	–
e	–

б) $i = 1$



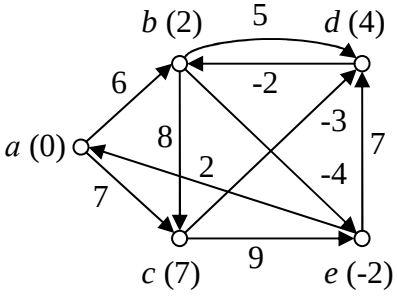
v	*P*(*v*)
a	–
b	*a*
c	*a*
d	*c*
e	*b*

в) $i = 2$



v	*P*(*v*)
a	–
b	*d*
c	*a*
d	*c*
e	*b*

г) $i = 3$



v	*P*(*v*)
a	–
b	*d*
c	*a*
d	*c*
e	*b*

д) $i = 4$

Рис. 30.7. Приклад роботи алгоритму Белмана-Форда.

30.6. Алгоритм Флойда-Уоршола

Розглянемо задачу пошуку в графі найкоротшого шляху між кожною парою вершин. Звичайно, цю загальнішу задачу можна розв'язати багатократним застосуванням алгоритмів Дейкстри або Белмана-Форда з послідовним вибором кожної вершини графу як початкової. Проте є прямий спосіб розв'язання цієї задачі за допомогою алгоритму Флойда-Уоршола. У ньому довжини дуг можуть бути від'ємними, проте не може бути циклів із від'ємною довжиною.

Нехай $G = (V, E)$ – орієнтований граф. Внутрішніми вершинами шляху $a, x_1, x_2, \ldots, x_{m-1}, b$ в графі G є вершини $x_1, x_2, \ldots, x_{m-1}$. Пронумеруємо вершини графу цілими числами від 1 до n. Позначимо як $\omega_{ij}^{(k)}$ довжину найкоротшого шляху з вершини i у вершину j, у якому як внутрішні можуть бути лише перші k вершин графу G. Якщо між вершинами i та j не існує жодного шляху, то вважатимемо, що $\omega_{ij}^{(k)} = \infty$. Зі сказаного випливає, що $\omega_{ij}^{(0)}$ – це вага дуги (i, j), а якщо такої дуги немає, то $\omega_{ij}^{(0)} = \infty$. Для довільної вершини i вважатимемо $\omega_{ii}^{(0)} = 0$. Отже, $\omega_{ij}^{(n)}$ дорівнює довжині найкоротшого шляху з вершини i у вершину j. Позначимо як $W^{(k)}$ матрицю розмірності $n \times n$, (i, j)-й елемент якої дорівнює $\omega_{ij}^{(k)}$. Якщо в заданому орієнтованому графі G відома вага кожної дуги, то, виходячи з попередніх міркувань, можна

сформувати матрицю $W^{(n)}$, елементи якої дорівнюють довжинам найкоротших шляхів між усіма парами вершин графу G.

В алгоритмі Флойда-Уоршола як початкову беруть матрицю $W^{(0)}$. Спочатку за нею обчислюють матрицю $W^{(1)}$, потім – $W^{(2)}$ і процес повторюють доти, доки за матрицею $W^{(n-1)}$ не буде обчислено матрицю $W^{(n)}$. Розглянемо ідею, на якій ґрунтується алгоритм Флойда-Уоршола. Припустимо, що відомі:

1. Найкоротший шлях із вершини i у вершину k, у якому як внутрішні використано лише перші $(k-1)$ вершин.
2. Найкоротший шлях із вершини k у вершину j, у якому як внутрішні використано лише перші $(k-1)$ вершин.
3. Найкоротший шлях із вершини i у вершину j, у якому як внутрішні використано лише перші $(k-1)$ вершин.

Оскільки за припущенням граф G не містить циклів із від'ємною довжиною, то один із двох шляхів – шлях із п. 3 чи об'єднання шляхів із пп. 1 і 2 – найкоротший шлях із вершини i у вершину j, у якому як внутрішні використано лише перші k вершин. Отже

$$\omega_{ij}^{(k)} = \min\left\{\omega_{ik}^{(k-1)} + \omega_{kj}^{(k-1)},\ \omega_{ij}^{(k-1)}\right\} \qquad 30.2$$

Зі співвідношення (30.1) видно, що для обчислення елементів матриці $W^{(k)}$ потрібні тільки елементи матриці $W^{(k-1)}$. Тепер ми можемо формально описати алгоритм Флойда-Уоршола для знаходження в графі найкоротших шляхів між усіма парами вершин.

Алгоритм Флойда-Уоршола

1. Присвоювання початкових значень. Пронумерувати вершини графа G цілими числами від 1 до n. Побудувати матрицю $W^{(0)}$, задавши кожний її (i, j)-й елемент таким, що дорівнює вазі дуги, котра з'єднує вершину i з вершиною j. Якщо в графі G ці вершини не з'єднано дугою, то виконати $\omega_{ij}^{(0)} = \infty$. Крім того, для всіх i виконати $\omega_{ii}^{(0)} = 0$.
2. Цикл по всіх k, що послідовно набуває значення $1, 2, \ldots, n$, визначити за елементами матриці $W^{(k-1)}$ елементи матриці $W^{(k)}$, використовуючи рекурентне співвідношення (30.1).

Після закінчення цієї процедури (i, j)-й елемент матриці $W^{(n)}$ дорівнює довжині найкоротшого шляху з вершини i у вершину j.

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

Якщо заздалегідь відомо, що в графі G немає циклів із від'ємною довжиною, то обсяг обчислень можна дещо зменшити. У цьому разі для всіх i та всіх k має бути $\omega_{ii}^{(k)} = 0$. Тому не потрібно обчислювати діагональні елементи матриць $W^{(1)}, W^{(2)}, \ldots, W^{(n)}$. Окрім того, для всіх $i = 1, 2, \ldots, n$ справджуються співвідношення $\omega_{ik}^{(k-1)} = \omega_{ik}^{(k)}$, $\omega_{ki}^{(k-1)} = \omega_{ki}^{(k)}$, які випливають з того, що коли немає циклів із від'ємною довжиною, вершина k не може бути внутрішньою в будь-яких найкоротших шляхах, котрі починаються чи закінчуються в самій вершині k. Отже обчислюючи матрицю $W^{(k)}$, немає потреби переобчислювати елементи k–го рядка й k–го стовпця матриці $W^{(k-1)}$. Отже, у матриці $W^{(k)}$ за формулою (30.2) потрібно обчислювати лише $n^2 - 3n + 2$ елементів. Очевидно, що складність алгоритму Флойда-Уоршола становить $O(n^3)$.

Щоб після закінчення його роботи можна було швидко знайти найкоротший шлях між будь-якою парою вершин, на k-й ітерації разом із матрицею $W^{(k)}$ побудуємо матрицю $\Theta^{(k)} = \left[\theta_{ij}^{(k)}\right]$. Спочатку беремо $\theta_{ij}^{(0)} = i$ для всіх $i, j = 1, \ldots, n$, $i \neq j$; $\theta_{ii}^{(0)} = 0$. Далі, на k-й ітерації візьмемо $\theta_{ij}^{(k)} = \theta_{ij}^{(k-1)}$, якщо $\omega_{ij}^{(k)} = \omega_{ij}^{(k-1)}$, і $\theta_{ij}^{(k)} = \theta_{kj}^{(k-1)}$, якщо $\omega_{ij}^{(k)} = \omega_{ik}^{(k-1)} + \omega_{kj}^{(k-1)}$. Отже, $\theta_{ij}^{(k)}$ –

номер вершини, яка є перед вершиною j у поточному $[i, j]$–шляху, тобто найкоротшому $[i, j]$–шляху, усі вершини якого містяться в множині $\{1, 2, \ldots, k\}$. Матриця $\Theta^{(n)} = \left[\theta_{ij}^{(n)}\right]$ відіграє ту саму роль, що й вектор θ в алгоритмі Дейкстри. За допомогою матриці $\Theta^{(n)}$ вершини, через які проходить найкоротший $[i, j]$ – шлях, визначають так: $i, .., j_3, j_2, j_1, j$, де $j_1 = \theta_{ij}^{(n)}$, $j_2 = \theta_{ij1}^{(n)}$, …

Розглянемо приклад. Потрібно знайти найкоротші шляхи між усіма вершинами в заданому на рис. 30.8 графі за допомогою алгоритму Флойда-Уоршола.

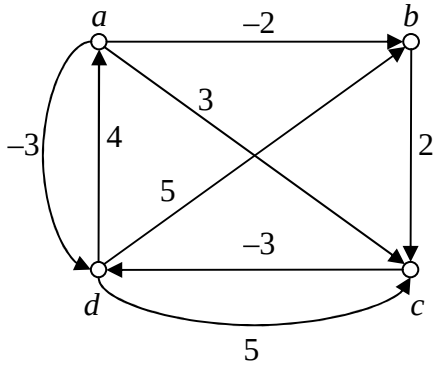


Рис 30.8.

Нижче наведемо результати виконання кожної з чотирьох ітерації алгоритму Флойда-Уоршола:

$$W^{(0)} = \begin{pmatrix} 0 & -2 & 3 & -3 \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & -3 \\ 4 & 5 & 5 & 0 \end{pmatrix}; \quad \Theta^{(0)} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 2 & 0 & 2 & 2 \\ 3 & 3 & 0 & 3 \\ 4 & 4 & 4 & 0 \end{pmatrix};$$

$$W^{(1)} = \begin{pmatrix} 0 & -2 & 3 & -3 \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & -3 \\ 4 & 2 & 5 & 0 \end{pmatrix}; \quad \Theta^{(1)} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 2 & 0 & 2 & 2 \\ 3 & 3 & 0 & 3 \\ 4 & 1 & 4 & 0 \end{pmatrix};$$

$$W^{(2)} = \begin{pmatrix} 0 & -2 & 0 & -3 \\ \infty & 0 & 2 & \infty \\ \infty & \infty & 0 & -3 \\ 4 & 2 & 4 & 0 \end{pmatrix}; \quad \Theta^{(2)} = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 2 & 2 \\ 3 & 3 & 0 & 3 \\ 4 & 1 & 2 & 0 \end{pmatrix};$$

$$W^{(3)} = \begin{pmatrix} 0 & -2 & 0 & -3 \\ \infty & 0 & 2 & -1 \\ \infty & \infty & 0 & -3 \\ 4 & 2 & 4 & 0 \end{pmatrix}; \quad \Theta^{(3)} = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 2 & 3 \\ 3 & 3 & 0 & 3 \\ 4 & 1 & 2 & 0 \end{pmatrix};$$

$$W^{(4)} = \begin{pmatrix} 0 & -2 & 0 & -3 \\ 3 & 0 & 2 & -1 \\ 1 & -1 & 0 & -3 \\ 4 & 2 & 4 & 0 \end{pmatrix}; \quad \Theta^{(4)} = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 4 & 0 & 2 & 3 \\ 4 & 1 & 0 & 3 \\ 4 & 1 & 2 & 0 \end{pmatrix}.$$

Матриця $W^{(4)}$ дає довжини найкоротших шляхів між усіма парами вершин графу на рис. 30.8. Зокрема, $\omega_{32}^{(4)} = -1$, тобто довжина найкоротшого шляху від вершини 3 до вершини

2 дорівнює –1. Для знаходження самого шляху скористаємося матрицею $\Theta^{(4)}$ та запишемо у зворотному порядку вершини через які він проходить: $\theta_{32}^{(4)} = 1$, $\theta_{31}^{(4)} = 4$, $\theta_{34}^{(4)} = 3$. Останнє значення свідчить про те, що найкоротший шлях з вершини 3 у вершину 4 йде безпосередньо з вершини 3, а отже процедуру можна зупинити. Таким чином отримали шлях: 3 – 4 – 1 – 2.

Очевидно, що як алгоритм Дейкстри, так і алгоритм Флойда-Уоршола можна застосовувати без жодних змін й до неорієнтованих графів. Для цього достатньо кожне ребро (u, v), що має вагу $w(u, v)$, розглядати як пару дуг (u, v) та (v, u) з тією самою вагою. Слід ураховувати, що неорієнтоване ребро із від'ємною вагою одразу приводить до циклу із від'ємною довжиною, що робить алгоритм Флойда-Уоршола незастосовним.

30.7. Алгоритм Джонсона

Алгоритм Джонсона дозволяє знайти найкоротші шляхи між усіма парами вершин зваженого орієнтованого графу. Цей алгоритм працює, якщо у графі містяться ребра з додатними чи від'ємними вагами, але відсутні негативні цикли.

Ідея алгоритму Джонсона полягає в наступному. Уявимо собі, що граф не містить ребер з від'ємними вагами. Тоді для знаходження найкоротших відстаней між усіма парами вершин ми можемо застосувати n разів алгоритм Дейкстри. У випадку, якщо в графі присутні негативні ребра, можна змінити вагу ребер таким чином, щоб всі вони стали додатними і тоді знову застосувати алгоритм Дейкстри n разів. Нехай $w(u, v)$ – вага ребра (u, v). Введемо нові ваги ребер $\hat{w}$, для яких виконуються наступні дві властивості:

1. Для всіх ребер (u, v) вага $\hat{w}(u, v) \geq 0$.
2. Для всіх пар вершин u, v шлях p є найкоротшим шляхом з u до v із використанням вагової функції w тоді й тільки тоді, коли p – також найкоротший шлях з u до v із використанням вагової функції $\hat{w}$.

Лема 30.1. Нехай задано зважений граф $G = (V, E)$ з ваговою функцією $w: E \to R$ і нехай $h: V \to R$ – довільна функція, що відображає вершини на дійсні числа. Для кожного ребра $(u, v) \in E$ визначимо $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$. Нехай p – довільний шлях з вершини v_1 до вершини v_k з ваговою функцією w тоді й тільки тоді, коли він є найкоротшим шляхом з ваговою функцією $\hat{w}$. Крім того, граф G містить цикл з негативною вагою з використанням вагової функції w тоді й тільки тоді, коли він містить цикл з негативною вагою з використанням вагової функції $\hat{w}$.

Отже, для того, щоб перейти до застосування алгоритму Дейкстри декілька разів, нам лишається навчитись визначати значення нової вагової функції $\hat{w}$ для кожного ребра. А для цього, відповідно, потрібно знати значення функцій $h(v)$ для всіх $v \in V$. Це можна зробити наступним за наступним алгоритмом.

Алгоритм визначення нових ваг ребер

1. Для даного графу G створимо новий граф $G' = (V', E')$, де $V' = V \cup \{s\}$, для деякої нової вершини $s \notin V$, та $E' = E \cup \{(s, v): v \in V\}$.
2. Розширимо вагову функцію w таким чином, щоб для всіх вершин $v \in V$ виконувалась рівність $w(s, v) = 0$.
3. Визначимо для всіх вершин $v \in V$ величину $h(v)$ як довжину найкоротшого шляху від вершини s до вершини v і нові ваги для всіх ребер $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$.

Зазначимо, що додавання нової вершини s так, як це зазначено вище, не змінює величину найкоротших маршрутів між будь-якими двома вершинами u та v у початковому графі G.

Прийнятним способом визначення найкоротших відстаней від нової вершини s до всіх вершин графу G є алгоритм Белмана-Форда. Наведемо приклад роботи алгоритму визначення нових ваг ребер для графу, який представлений на рис. 30.8. Робота алгоритму Белмана-Форда для графу із доданою вершиною s наведений на рис. 30.9. Ребра, які виходять з вершини s та позначені пунктиром, мають вагу 0. Алгоритм зупиняє роботу на ітерації $i = 3$, адже за попередній крок не відбулось жодних змін ваг ребер (у порівнянні з ітерацією $i = 2$).

На рис. 30.10, *а* наведені розраховані значення $h(v)$ для всіх вершин $v \in V$ графу, а на рис. 30.10, *б* – граф із новими вагами ребер.

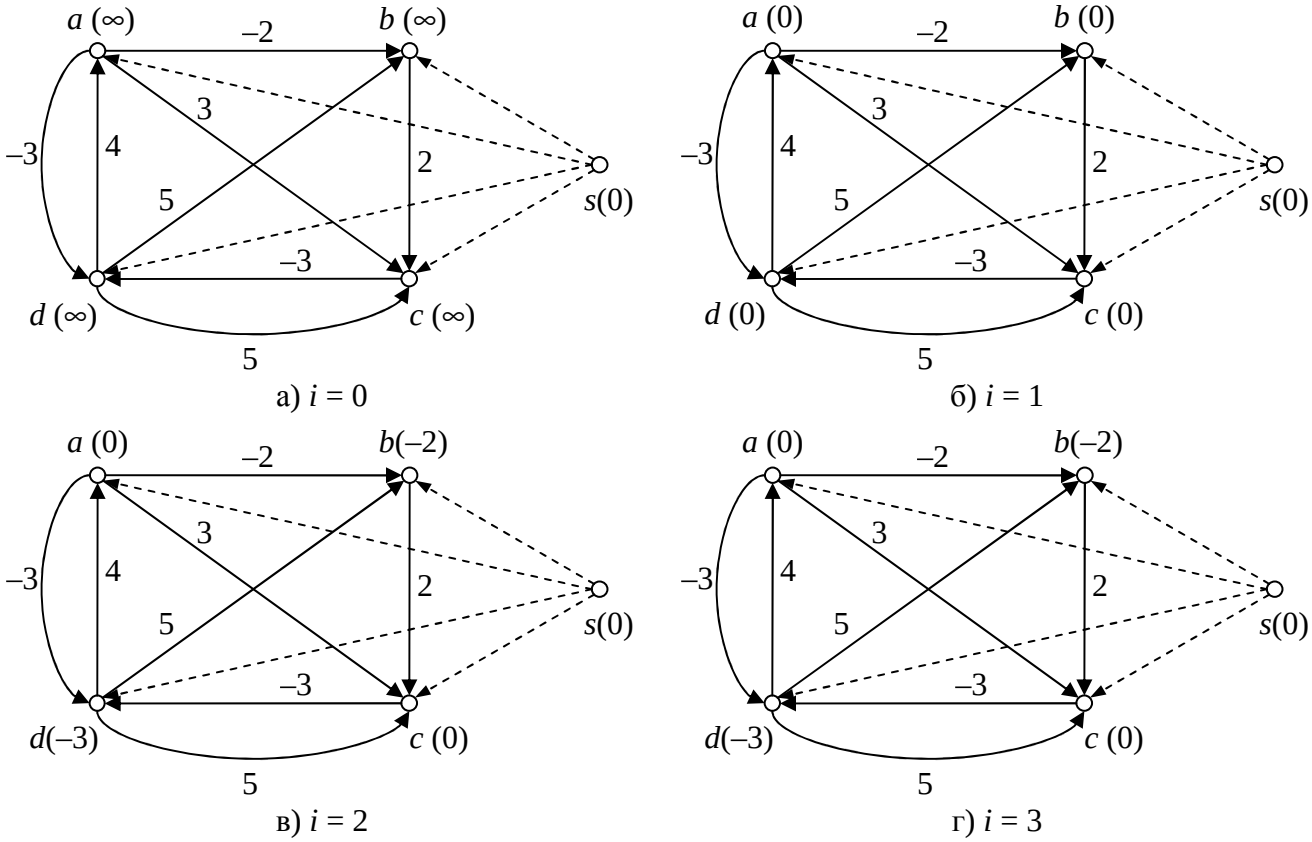
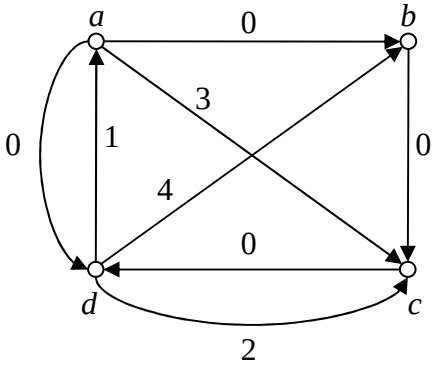


Рис. 30.9. Визначення нових ваг ребер за допомогою алгоритму Белмана-Форда.

v	a	b	c	d
$h(v)$	0	–2	0	–3

а)



б)

$$D' = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

в)

$$D = \begin{pmatrix} 0 & -2 & 0 & -3 \\ 3 & 0 & 2 & -1 \\ 1 & -1 & 0 & -3 \\ 4 & 2 & 4 & 0 \end{pmatrix}$$

г)

Рис. 30.10. Значення функції $h(v)$ для всі вершин (а); граф з новими вагами ребер (б); матриця відстаней для нового графу (в); матриця відстаней для оригінального графу (г).

Алгоритм Джонсона.

1. Створити новий граф G' шляхом додавання нової вершини s і ребер (s, v) з вагою 0 для всіх $v \in V$.
2. Запустити алгоритм Белмана-Форда для графу G' і початкової вершини s. Якщо алгоритм Белмана-Форда сповістив про знаходження негативного циклу, то закінчити роботу алгоритму з помилкою.
3. Для всіх вершин $v \in V$ визначити величину $h(v)$ як довжину найкоротшого шляху від вершини s до вершини v (за результатами алгоритму Белмана-Форда) і нові ваги для всіх ребер $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$.
4. Для кожного вершини $u \in V$ запустити алгоритм Дейкстри для знаходження найкоротшої відстані від вершини u до всіх інших вершин графу $v \in V$. Сформувати на основі отриманих відстаней матрицю D'.
5. Визначити реальну відстань $d(u, v) = d'(u, v) - h(u) + h(v)$, де $d'(u, v)$ – елемент матриці D'.

Для розглянутого вище прикладу матриця відстаней для модифікованого графу D' наведена на рис. 30.10, *в*, а матриця відстаней для початкового графу D – на рис. 30.10, *г*. Як бачимо, отримана матриця D дорівнює матриці $W^{(4)}$, яка була знайдена під час роботи алгоритму Флойда-Уоршола для цього самого графу (стор. 196).

Конкретні найкоротші маршрути, а не тільки їх довжина, визначаються на етапі запуску алгоритмів Дейкстри для кожної окремо взятої початкової вершини u.

Визначимо час роботи алгоритму Джонсона. Перший крок виконується за час $O(n)$, другий – за час $O(nm)$, третій – за час $O(m)$, четвертий – за час $O(nm \lg n)$, п'ятий – за час $O(n^2)$. Таким чином, загальний час роботи визначається як найбільший з наведений етапів, тобто $O(nm \lg n)$. Для розряджених графів це значення виявляється набагато меншим, ніж величина $O(n^3)$ яку дає алгоритм Флойда-Уоршола.

Підіб'ємо підсумки. Якщо необхідно знайти відстань від однієї вершини до іншої або до всіх вершин графу і ваги всіх ребер графу є додатними або дорівнюють нулю, то найбільш ефективним виявляється алгоритм Дейкстри із часом роботи $O(m \lg n)$. Якщо ж ваги ребер можуть бути від'ємними, то необхідно застосовувати алгоритм Белмана-Форда, час роботи якого $O(nm)$.

Якщо необхідно знайти відстані між усіма парами вершин графу, граф є розрядженим і всі ребра мають невід'ємні ваги, то можна виконати n разів алгоритм Дейкстри. Якщо ж граф є розрядженим, але в ньому можуть бути ребра з від'ємними вагами, то необхідно використовувати алгоритм Джонсона. Якщо необхідно знайти відстані між усіма парами вершин, ваги ребер можуть бути від'ємними і граф не є розрядженим (m прямує до n^2), то необхідно використовувати алгоритм Флойда-Уоршола.

Жоден з наведених алгоритмів не може бути застосований для графів, які містять негативні цикли. Проте алгоритм Белмана-Форда (як і алгоритм Джонсона), а також алгоритм Флойда-Уоршола можуть виявити такі цикли.