Тема 2. Окремі випадки застосування методів мережевого планування в управлінні бізнес-процесами

План

1. Мережеві математичні моделі у задачах управління бізнес-процесами
2. Основні поняття та визначення теорії графів
3. Задача мінімізації мережі
4. Задача про найкоротший шлях
5. Задача про максимальний потік в мережі

2.1. Мережеві математичні моделі у задачах управління бізнес-процесами

Значну кількість практичних задач з управління бізнес-процесами, математично можна описати, як задачі лінійного програмування. Для даного типу оптимізаційних задач, існує універсальний алгоритм їхнього вирішення – це симплексний метод.

Однак, достатньо велика кількість задач лінійного програмування при їхньому вирішенні симплексним методом, вимагає наявності значних обчислювальних потужностей й може займати тривалий час, що не виправдовує застосування даного методу.

Сукупність таких задач може бути математично описана й вирішена більш раціонально в рамках спеціальних теорій. Вони розробляють ефективні алгоритми вирішення цих задач, об'єднаних в деякі однотипні класи. Таким чином, існує ряд практичних задач з управління бізнес-процесами, які зручно представляти, наприклад, у вигляді графічних структур (мережевих моделей).

Наприклад, можна формалізувати процес прийняття рішень з функціонування виробничої системи, транспортування продукції, передачі інформації тощо. Перераховані задачі можуть бути сформульовані й вирішені у вигляді задач лінійного програмування. Однак, у зв'язку з величезною кількістю змінних й обмежень, пряме застосування симплексного методу в мережевих задачах є недоцільним. Особлива структура таких задач дозволяє розробити більш ефективні алгоритми їхнього вирішення.

Розглянемо детальніше типові приклади спеціальних ЗЛП й методи їхнього розв’язання, а саме: модифікацію транспортної задачі й її постановку на мережевих моделях (графах).

Транспортна задача й її можливі економічні інтерпретації – це лише одна з багатьох задач, які можуть бути сформульовані й вирішені за допомогою мережевих моделей.

Приклад 1. Задача мінімізації мережі.

Дана оптимізаційна задача виникає в різних сферах управління бізнес-процесами, коли необхідно визначити максимально коротке з'єднання двох й більше заданих об'єктів (наприклад, найкоротший маршрут між обраними містами, розташування двох елементів з мінімальною відстанню на

електронній схемі). *Розглянемо нижче змістовну постановку типової задачі мінімізації мережі.*

Припустимо, в деякому місті, населення якого перевищило 1 мільйон осіб, планується будівництво метрополітену, який повинен з'єднати центр міста й п'ять його районів підземною залізничною колією. Оцінивши можливі варіанти будівництва тунелів, були обрані найбільш ефективні, з точки зору переслідуваної мети. Маршрути прокладання тунелів повинні бути обрані таким чином, щоб мета будівництва досягалась за критерієм мінімальних витрат на будівництво, або мінімальної довжини тунелів. Й щоб кожен з районів міста був напряму з'єднаний підземною залізничною колією з його центром, або через інші райони.

Приклад 2. Задача про найкоротший шлях.

Оптова компанія здійснює продаж товарів через торгівельну мережу магазинів. Відомі всі можливі маршрути доставки товарів зі складу цієї компанії в кожен з магазинів, а також транспортні витрати на кожний маршрут (або, наприклад, довжина маршрутів). Для того щоб скоротити загальні витрати (або, сумарний кілометраж) на доставку товарів до магазинів, керівництву компанії слід обрати таку множину маршрутів, яка би дозволяла доставляти товари зі складу в кожен з магазинів безпосередньо, або через інші магазини, щоб мінімізувати витрати компанії.

Приклад 3. Задача про максимальний потік в мережі.

Дана задача виникає кожного разу, коли через певну мережу пропускається будь-який матеріальний, фінансовий, або інформаційний потік. При цьому, необхідно знайти такий розподіл елементів потоку по наявних каналах зв’язку, щоб за одиницю часу передавався його максимальний обсяг.

Наприклад, газова компанія з Азербайджану підписала контракт на постачання нафтопродуктів до Німеччини трубопроводом. Транспортування нафтопродуктів можливе через трубопроводи та паромні станції, що розташовані на території України. Пропускна здатність трубопроводів на різних ділянках також є різною. Перед керівництвом компанії постає питання: який максимально можливий обсяг нафтопродуктів можна транспортувати по існуючій мережі в одиницю часу?

Аналіз цих прикладів показує, що оптимізаційні мережеві задачі можуть бути описані наступними типами моделей:

1) мінімізація мережі (випадок 1);
2) знаходження найкоротшого шляху (випадок 2);
3) визначення максимальної пропускної здатності (випадок 3).

Всі розглянуті приклади мережевих задач можуть бути сформульовані й вирішені, як задачі лінійного програмування. Однак, у зв'язку з величезною кількістю змінних й обмеженістю обчислювальних можливостей, пряме застосування симплексного методу є недоцільним. Особлива структура цих

задач дозволяє створювати більш ефективні алгоритми, які в більшості випадків базуються на теорії лінійного програмування.

Аналіз графічних структур дозволяє знаходити оптимальні рішення цих задач. Одним з таких представлень є мережа (граф). Теорія, яка розвиває алгоритми розв'язання задач на мережах (або графах), називається теорією графів. У загальному випадку, граф – це пара множин, елементи однієї з яких називаються вершинами, а інші – ребрами (або дугами).

2.2. Основні поняття та визначення теорії графів

Звичайний граф (мережа) G = (V, E) – це впорядкована пара множин, рис. 1, скінченної непорожньої множини V, елементи якої називаються **вершинами графу** G й довільної підмножини E $\subseteq$ <V×V>, елементи якої називаються **ребрами** цього графа (мережі).

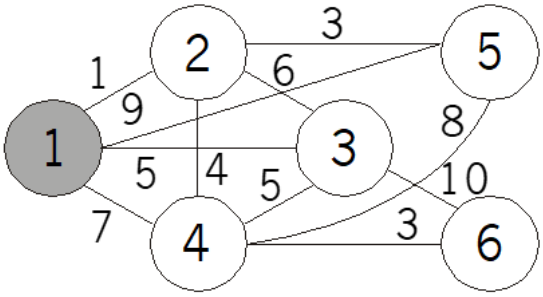

Рис. 1. Приклад звичайного графу

Основними властивостями звичайного графу є:

1. Множина ребер є обмеженою;
2. Ребра графу є неорієнтованими;
3. В графі відсутні петлі, тобто ребра виду *l = (v1, v1)*;
4. Граф *G* не містить кратних ребер (тобто (*v1, v2*) = (*u1, u2*), якщо *v1 = u1* та *v2 = u2*);

Два крайніх випадки звичайних n-вершинних графів:

1. Безреберний граф, де E = ∅;
2. Повний граф, де E = <V×V>, тобто будь-які дві його вершини є суміжними.

Далі, на основі рис. 1, розглянемо основні поняття й визначення теорії графів.

Послідовність вершин й ребер графу *v0 (v0, v1) v1 (v1, v2) v2 ... vn* називається **маршрутом**, що з'єднує вершини *v0* та *vn*.

Маршрут називається **ланцюгом,** якщо всі його ребра є різними (не повторюються):

1 (1, 2) 2 (2, 3) 3 (3, 4) 4 (4, 2) 2 (2, 5) 5.

Ланцюг називається **простим**, якщо всі його вершини є різними (не повторюються):

1 (1, 4) 4 (4, 5) 5.

Ланцюг, в якому початкова вершина *v0* співпадає з кінцевою вершиною *vn* й всі ребра є різними, називається **циклом**:

1 (1, 2) 2 (2, 3) 3 (3, 4) 4 (4, 2) 2 (2, 5) 5 (5, 1) 1.

Цикл називається **простим**, якщо всі його вершини є різними, за винятком початкової *v0* й кінцевої *vn*, які є однаковими:

1 (1, 4) 4 (4, 2) 2 (2, 5) 5 (5, 1) 1.

Граф (мережа) називається **зв'язаним,** якщо будь-які дві його неспівпадаючі вершини з'єднані маршрутом (граф на рис. 1 є зв'язаним). В іншому випадку, він є незв’язаним, рис. 2.

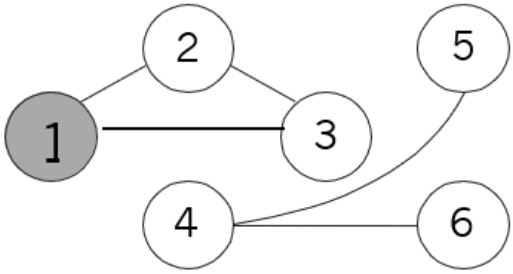

Рис. 2. Приклад незв’язаного графу

Ребро, після видалення якого граф зі зв’язаного перетворюється на незв’язаний, називається **мостом**.

Граф називається **зваженим**, якщо кожному з його ребер відповідає певне число ω(i), яке називається **вагою ребра**.

Тоді, під **вагою графу** розуміють суму ваг всіх його ребер, тобто:

$\boldsymbol{\omega(G) = \sum \omega(i)}$.

Зв’язаний граф, який не містить циклів, називається **деревом**, рис. 3.

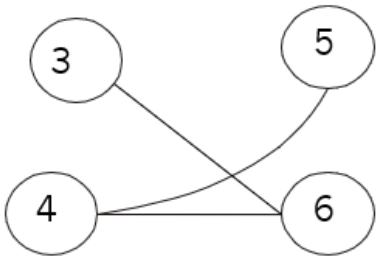

Рис. 3. Приклад дерева

Орієнтований граф – це пара множин (V, A), де V – множина вершин; A – множина орієнтованих ребер, які називаються **дугами**. Приклад орієнтованого графу наведений на рис. 4.

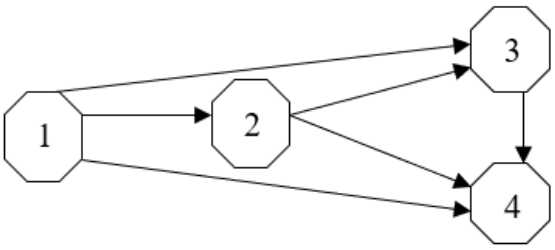

Рис. 4. Орієнтований граф

Якщо $a = (v1, v2)$ – це дуга, то вершини $v1$ й $v2$ називаються її **початком** та **кінцем** відповідно.

Якщо кожна дуга графу має певну вагу, то такий граф називається **орієнтованим зваженим**.

2.3. Задача мінімізації мережі

Задача мінімізації мережі полягає в пошуку ребер, які з'єднують між собою всі вузли мережі (тобто, всі вершини графу) й мають мінімальну загальну довжину, або вагу. Очевидно, що оптимальне вирішення цієї задачі не повинно містити циклів.

Для її вирішення існують ефективні алгоритми, що застосовуються до довільного зв'язного графа.

Алгоритм Краскала

1. Будуємо граф $T_1 = O_n + l_1$, приєднавши ребро l_1 мінімальної ваги до порожнього графа O_n на множині вершин V графа G. Якщо таких ребер декілька, тобто $\omega(l_1) = \omega(l_2) = min\ \omega(l)$, тоді обирається будь-яке з цих ребер. Це вказує на неоднозначність рішення (оптимальних рішень може бути декілька).
2. Якщо ми вже маємо побудований граф T_i, причому $i < (n - 1)$, то будуємо граф $T_{i+1} = T_i + l_{i+1}$, де l_{i+1} – ребро графу G, яке має мінімальну вагу серед ребер, що не входять до складу T_i й не утворюють з ними циклів.
3. В іншому випадку, якщо $i = (n - 1)$, то алгоритм завершує свою роботу. Тобто, ми побудували граф мінімальної ваги, який охоплює всі вершини.

Даний алгоритм на кожному кроці будує ациклічний граф, який на останньому кроці стає зв’язаним.

Розглянемо ще один алгоритм для вирішення задачі про мінімізацію мережі.

Алгоритм Прима

1. Будуємо граф $T_1 = O_n + l_1$, приєднавши ребро l_1 мінімальної ваги до порожнього графа O_n на множині вершин V графа G. Якщо таких ребер декілька, тобто $\omega(l_1) = \omega(l_2) = min\ \omega(l)$, тоді обирається будь-яке з цих ребер.

По аналогії, це вказує на неоднозначність рішення (оптимальних рішень може бути декілька).

2. Якщо ми вже маємо побудований граф T_i, причому $i < (n - 1)$, то будуємо граф $T_{i+1} = T_i + l_{i+1}$, де l_{i+1} – ребро графу G мінімальної ваги, яке додається до однієї з вершин цього графу T_i.

3. В іншому випадку, якщо $i = (n - 1)$, то алгоритм завершує свою роботу. Тобто, ми побудували граф мінімальної ваги, який охоплює всі вершини.

На відміну від алгоритму Краскала, алгоритм Прима будує на кожному кроці зв'язний ациклічний граф.

Примітка

У деяких ситуаціях доводиться вирішувати задачу з максимізації мережі, будуючи зв’язаний граф не мінімальної, а максимальної ваги. В таких випадках, алгоритми Краскала й Прима також можуть використовуватись. Необхідно лише всюди замінити максимальні ваги ребер на максимальні.

Приклад 4. Телекомунікаційна компанія планує створити кабельну мережу для обслуговування п'яти нових будинків, рис. 6.

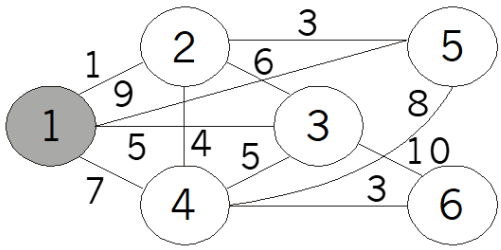

Рис. 6. Схема можливої прокладки кабелів

Цифри на ребрах вказують на довжину кабелю (км), що з'єднує відповідні вершини графу. Вузол №1 представляє собою ретранслятор сигналу, а решта вузлів (№2 – №6) відповідають п'яти новим будівлям.

Необхідно знайти такі ребра, які з’єднають всі вершини графу між собою, причому, вага графу повинна бути мінімальною. Таким чином, маємо задачу мінімізації мережі.

Рішення

Алгоритм Краскала

Упорядкуємо ребра графу, рис. 6, в порядку зростання їхніх ваг:

Ребра графу	Ваги ребер	Додаємо (+), або не додаємо (–) ребра
(1,2)	1	+
(4,6)	3	+
(2,5)	3	+
(2,4)	4	+
(1,3)	5	+ (–)
(3,4)	5	– (+)
(2,3)	6	–

(1,4)	7	–
(4,5)	8	–
(1,5)	9	–
(3,6)	10	–

Знаком «+» в таблиці позначені ті ребра, які ми включили до складу графу. Одне з ребер (1, 3), або (3, 4) можуть бути включені до даного графу.

Таким чином, мінімальна довжина кабелю, що з'єднує ретранслятор сигналу з новозбудованими будівлями, дорівнює:

$\omega = 1 + 3 + 3 + 4 + 5 = 16$ (км).

Алгоритм Прима

Логічним є розпочати вирішення даної задачі з вершини №1, де розташований ретранслятор сигналу.

T_1: $V_1 = \{1, 2\}$, $E_1 = \{(1, 2)\}$ (множина вершин та множина ребер, відповідно);

$T_2 = T_1 \cup (2, 5)$: $V_2 = \{1, 2, 5\}$, $E_2 = \{(1, 2), (2, 5)\}$;

$T_3 = T_2 \cup (2,4)$: $V_3 = \{1, 2, 4, 5\}$, $E_3 = \{(1, 2), (2, 5), (2, 4)\}$;

$T_4 = T_3 \cup (4,6)$: $V_4 = \{1,2,4,5,6\}$, $E_2 = \{(1, 2), (2, 5), (2, 4), (4, 6)\}$;

$$T_5 = \begin{cases} T_4 \cup (1,3)\text{: } V_5 = \{1,2,3,4,5,6\},\ E_5 = \{(1,2), (2,5), (2,4), (4,6), (1,3)\} \\ T_4 \cup (3,4)\text{: } V_5 = \{1,2,3,4,5,6\},\ E_5 = \{(1,2), (2,5), (2,4), (4,6), (3,4)\} \end{cases}$$

Оскільки $i = (n - 1) = 6 - 1 = 5$, то ітераційний процес побудови зв'язаного графу завершується, рис. 7.

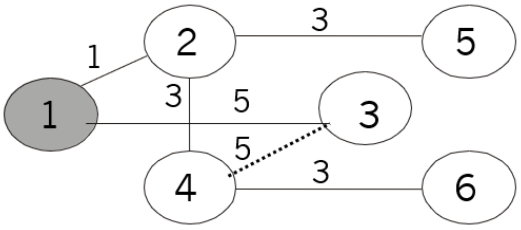

Рис. 7. Вирішення задачі мінімізації мережі

Таким чином, оптимальна мережа підключень нових будинків до ретранслятору, показана на рис. 7. Для її практичної реалізації, необхідний кабель мінімальної довжини у 16 км.

2.4. Задача про найкоротший шлях

Задача про найкоротший шлях, зазвичай вирішується на орієнтованих графах.

Нехай G = (V, A) – орієнтований зважений граф. Задача про найкоротший шлях полягає в тому, щоб знайти шлях мінімальної ваги, що з'єднує задані початкову й кінцеву вершини графа G за умови, що існує хоча б один такий шлях.

Початкова й кінцева вершини позначаються відповідно *s* й *t*. Таким чином, *(s, t)* – це шлях мінімальної ваги, який будемо називати **найкоротшим *(s, t)*-шляхом.**

Розглянемо випадок, коли ваги всіх дуг графу є невід'ємними. Тобто, ($\omega(l) \geq 0, \forall\ l \in A$). На сьогоднішній день існує ефективний алгоритм побудови найкоротшого шляху в графі з невід'ємними вагами дуг, запропонований Е. Дейкстрою у 1959 році.

Сутність алгоритму. На кожній ітерації цього алгоритму, кожна вершина *v* графа G має мітку *l(v)*, яка може бути *постійною*, або *тимчасовою*:

– якщо мітка постійна, то *l(v)* – це вага найкоротшого *(s, v)* шляху, який тільки існує;

– якщо мітка тимчасова, то *l(v)* – це вага найкоротшого *(s, v)* шляху, який було знайдено на поточний момент, але який в подальшому може зменшитись в ході роботи алгоритму.

Ставши на деякій ітерації постійною, мітка *l(v)* залишається такою до завершення роботи алгоритму.

Окрім мітки *l(v)*, кожна вершина *v* графа G, за винятком початкової вершини *s*, також пов'язана з іншою міткою – $\theta(v)$. На кожній ітерації, $\theta(v)$ показує номер вершини, що передує *v* у *(s, v)*-шляху, який має найменшу вагу серед усіх *(s,v)*-шляхів, що проходять через вершини, які отримали постійні мітки *l(v)* до цього часу. За допомогою міток $\theta(v)$ можна легко відновити послідовність вершин, які складають найкоротший *(s,v)* шлях.

Перед початком першої ітерації алгоритму, вершина *S* має постійну мітку *l(s) = 0*, а *l*-мітки всіх інших вершин графу дорівнюють нескінченності й вони є тимчасовими.

Загальна ітерація алгоритму виглядає наступним чином. Нехай *p* – це вершина, яка отримала постійну мітку *l(p)* на попередній ітерації. Перевіряємо всі вершини *v*, що є суміжними з вершиною *p* й які мають тимчасові мітки *l*, з метою покращення їхніх значень (зменшення):

– тимчасові мітки *l(v)* суміжних до *p* вершин змінюють значення на $l(p) + \omega(p, v)$, якщо виконується умова: $l(v) > l(p) + \omega(p, v)$. Також змінюється мітка $\theta(v) = p$.

– якщо $l(v) \leq l(p) + \omega(p, v)$, то мітки θ та *l* вершини *v* на даній ітерації не змінюють своїх значень.

Алгоритм завершує свою роботу, коли мітка *l(t)* для останньої вершини *t* стає постійною. Тоді *l(t)* – це вага найкоротшого *(s, t)* шляху, який позначається, як P^*.

Отриманий оптимальний шлях з мінімальною вагою визначається за допомогою θ міток у зворотному порядку, від вершини *t* до вершини *s*.

Алгоритм Дейкстри

1. Перед початком ітераційного процесу, мітка *l* для першої вершини *s* буде дорівнювати *l(s) = 0* й вона вважається постійною. Для всіх інших вершин *v* нашого графу, мітки приймають нескінченне значення $l(v) = \infty$ й вважаються

тимчасовими. Також, остання вершина, яка отримала статус постійної мітки l – це вершина p, тобто, $p = s$.

2. Перевіряємо всі вершини v, що є суміжними з вершиною p й які мають тимчасові мітки l, з метою покращення їхніх значень (зменшення):

– якщо виконується умова: $l(v) > l(p) + \omega(p, v)$, то $l(v) = l(p) + \omega(p, v)$. Також змінюється мітка $\theta(v) = p$.

– якщо $l(v) \leq l(p) + \omega(p, v)$, то мітки θ та l вершини v на даній ітерації не змінюють своїх значень.

3. Обираємо зі всієї множини вершин зі змінними мітками l ту вершину, для якої мітка l приймає найменше значення й переводимо її в статус постійної.

4. Запам'ятовуємо вершину, яка на останньому кроці отримала статус постійної: $p = v$.

5. Якщо мітка l останньої вершини графу отримала статус постійної, тобто, $p = t$, то ітераційний процес завершується. Переходимо до пункту 7.

6. Переходимо до пункту 2.

7. За допомогою міток θ визначаємо оптимальний маршрут $(s; t)$ з мінімальною вагою. Він визначається у зворотному порядку, від вершини t до вершини s.

Зауваження

1. Алгоритм Дейкстри може застосовуватись не лише до орієнтованих, але й до неорієнтованих графів. Для цього, кожне неорієнтоване ребро графа (u, v), з вагою $\omega(u, v)$, замінюється на пару дуг (u, v) та (v, u) однакової ваги.

3. Якщо етап 5 алгоритму модифікувати так, щоб він завершувався лише після того, як всі вершини отримають постійні мітки, то даний алгоритм побудує найкоротші шляхи від початкової вершини s до кожної з інших вершин графу.

Приклад 5. Задача про найкоротший шлях

Нехай, вхідний граф має вигляд, як показано на рис. 8. Необхідно знайти найкоротший шлях з вершини I до вершини IX.

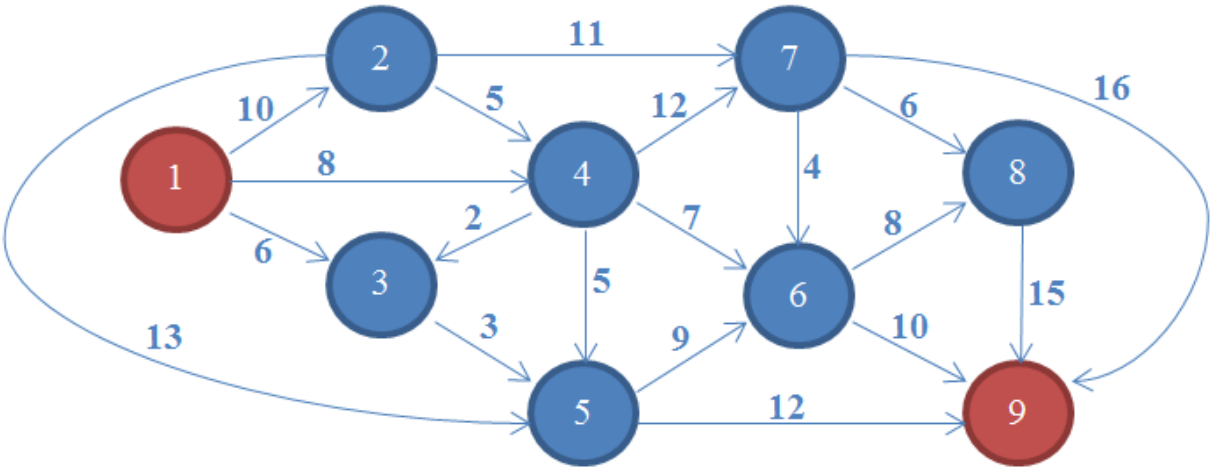

Рис. 8. Задача про пошук найкоротшого шляху з вершини I у IX

Рішення. Ітераційний процес зі знаходження постійних міток l й міток θ показаний в таблиці нижче:

	A	B	C	D	E	F	G	H	I	J	K
1	Ітерація	Мітки	Вершини графу								
2			1	2	3	4	5	6	7	8	9
3	0	L	0	∞	∞	∞	∞	∞	∞	∞	∞
4		Q									
5	1	L	0	10	6	8	∞	∞	∞	∞	∞
6		Q		1	1	1					
7	2	L	0	10	6	8	9	∞	∞	∞	∞
8		Q		1	1	1	3				
9	3	L	0	10	6	8	9	15	20	∞	∞
10		Q		1	1	1	3	4	4		
11	4	L	0	10	6	8	9	15	20	∞	21
12		Q		1	1	1	3	4	4		5
13	5	L	0	10	6	8	9	15	20	∞	21
14		Q		1	1	1	3	4	4		5
15	6	L	0	10	6	8	9	15	20	23	21
16		Q		1	1	1	3	4	4	6	5
17	7	L	0	10	6	8	9	15	20	23	21
18		Q		1	1	1	3	4	4	6	5
19	8	L	0	10	6	8	9	15	20	23	21
20		Q		1	1	1	3	4	4	6	5

На сьомій ітерації кінцева вершина IX набула постійного статусу мітки l. Це означає, що найкоротший шлях до неї був знайдений й він дорівнює $l = 21$. Далі відновимо маршрут до кінцевої вершини IX, за допомогою міток θ:

– до IX вершини ми прийшли з вершини $\theta = 5$;

– до V вершини ми прийшли з вершини $\theta = 3$;

– до III вершини ми прийшли з вершини $\theta = 1$.

Таким чином, найкоротший маршрут має вигляд: I → III → V → IX.

2.5. Задача про максимальний потік в мережі

Нехай, маємо G = (V, A) – орієнтований зважений граф (мережу), де його вершина:

– $S^{/} \in V$ називається **джерелом**, якщо немає дуг, які закінчуються у $S^{/}$;

– $S^{//} \in V$ називається **стоком,** якщо немає дуг, що починаються у $S^{//}$.

Припустимо, що G має рівно одне джерело $S^{/}$ та рівно один сток $S^{//}$. Тоді, функція $\varphi(l)$ – це **ємність дуги** l, яка визначає **пропускну здатність**, або максимальну величину потоку, що проходить через дугу l.

Потоком в графі є функція μ, яка має властивості:

1. $\forall\ l \in A: 0 \le \mu(l) \le \varphi(l)$, де $\mu(l)$ – потік через дугу l. Тобто, для будь-якої дуги графу l, пропущений через неї потік $\mu(l)$ не може перевищувати її пропускну здатність $\varphi(l)$;

2. $\forall\ u \in V$, $u \ne S^{/}$ и $u \ne S^{//}$: для будь-якої вершини u (якщо вона не є джерелом, чи стоком), сумарний потік, що увійшов до неї, дорівнює сумарному потоку, що вийшов з неї.

Величиною потоку $\boldsymbol{\rho(\mu)}$, який був пропущений через граф, називається сумарний потік, що вийшов з джерела $S^{/}$; або сумарний потік, що увійшов до стоку $S^{//}$.

Теорема. На графі G = (V, A), сумарний потік по всіх дугах, що вийшов з джерела, дорівнює сумарному потоку по всіх дугах, що увійшов у сток.

Постановка задачі про максимальний потік в мережі

Нехай, граф G = (V, A) має одне джерело $S^{/}$ й один сток $S^{//}$. Пропускна здатність кожної дуги l визначається функцією $\varphi(l)$. Величина потоку ресурсів, що були пропущені цим графом, визначається функцією $\boldsymbol{\rho(\mu)}$. Необхідно знайти максимальну величину потоку, який можна пропустити вказаним графом, тобто:

$$\rho(\mu) \to max$$

Для вирішення задачі про максимальний потік в мережі, застосовують алгоритм Форда-Фалкерсона. Розглянемо його більш детально.

Алгоритм Форда-Фалкерсона

1. Знаходимо довільний маршрут від вершини-джерела $S^{/}$ до вершини-стоку $S^{//}$, по якому може бути пропущений потік;
2. На знайденому маршруті визначаємо, яка величина потоку може бути пропущена ним, як мінімальне значення від невикористаної пропускної потужності кожної дуги, з яких складається цей маршрут;
3. Пропускаємо потік даним маршрутом. Коригуємо (зменшуємо) невикористану пропускну потужність кожної дуги, з яких складається цей маршрут;
4. Етапи (1)-(3) повторюються до тих пір, поки з вершини-джерела $S^{/}$ до вершини-стоку $S^{//}$ може бути пропущений потік. Якщо таких маршрутів більше не існує, то переходимо до етапу (5);
5. Обчислюємо максимальний потік, який був пропущений нашим графом, одним з наступних способів:

– сума всіх пропущених потоків на етапі (3) роботи алгоритму;
– сума всіх потоків, що вийшли з вершини-джерела $S^{/}$;
– сума всіх потоків, що увійшли до вершини-стоку $S^{//}$.

Зауваження

Якщо початковий граф G = (V, A) має дві вершини-джерела $S^{/}_1$ та $S^{/}_2$, то до нього додається ще одна вершина-джерело $S^{/}$, від якої йдуть дуги до $S^{/}_1$ та $S^{/}_2$. Пропускна здатність дуги ($S^{/}$; $S^{/}_1$) дорівнює пропускній здатності всіх дуг, які виходять з $S^{/}_1$; пропускна здатність дуги ($S^{/}$; $S^{/}_2$) дорівнює пропускній здатності всіх дуг, які виходять з $S^{/}_2$.

Аналогічно, якщо початковий граф G = (V, A) має дві вершини-стоки $S^{//}_1$ та $S^{//}_2$, то до нього додається ще одна вершина-сток $S^{//}$, до якої йдуть дуги від

S''_1 та S''_2. Пропускна здатність дуги (S''_1; S'') дорівнює пропускній здатності всіх дуг, які входять до S''_1; пропускна здатність дуги (S''_2; S'') дорівнює пропускній здатності всіх дуг, які входять до S''_2.

Приклад 6. Задача про максимальний потік в мережі

Двоє користувачів інтернету обмінюються один з одним якоюсь інформацією. Їх робочі станції пов'язані між собою через мережу серверів. Більш того, передача інформації можлива за допомогою різних комбінацій серверів й каналів зв'язку, що з'єднують їх.

Який максимально можливий обсяг інформації може бути переданий за одиницю часу даною мережею від одного користувача до іншого, якщо нам відома пропускна здатність кожного каналу зв'язку?

Побудова математичної моделі

Припустимо, що користувачі S' та S'' з'єднані між собою через 5 серверів (вузлів мережі), відповідними каналами зв'язку (дугами). Припустимо також, що нам відомі обсяги інформації, яку здатний пропускати кожний канал зв'язку за одиницю часу (ваги дуг).

Тоді, початкову задачу можна формалізувати у вигляді наступної мережевої моделі, рис. 9.

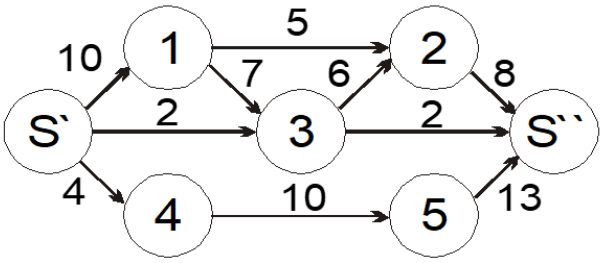

Рис. 9. Мережева модель задачі про максимальний потік

Знайти, який максимальний обсяг інформації, користувач S' може передати користувачу S'' в одиницю часу.

Дана задача може бути ефективно вирішена за допомогою **алгоритму Форда-Фалкерсона**.

Рішення

1. Знайдемо довільний шлях, який з'єднує вершини графу S' та S''

$S' \to 1 \to 3 \to 2 \to S''$

2. Обираємо мінімальну пропускну здатність з усіх дуг, що входять до складу обраного шляху

$\alpha_1 = \min\{10, 7, 6, 8\} = 6$

3. Ми знайшли максимально можливий обсяг потоку, який можна пропустити цим шляхом. Позначаємо його на графі над цими дугами, в дужках. Поряд з дужками записуємо пропускну здатність дуг, яка залишилась після пропуску даного потоку.

Дуга з нульовою залишковою пропускною здатністю, виключається з розгляду.

Кроки (1)-(3) повторюються до тих пір, поки існують шляхи, що з'єднують вершини S' та S''.

Процес вирішення даної задачі представлений нижче, а проілюстрований на рис. 10.

$S' \to 4 \to 5 \to S''$, $\alpha_2 = \min\{4, 10, 13\} = 4$;

$S' \to 1 \to 2 \to S''$, $\alpha_3 = \min\{4, 5, 2\} = 2$;

$S' \to 3 \to S''$, $\alpha_4 = \min\{2, 2\} = 2$.

Після останнього маршруту, алгоритм завершує свою роботу, оскільки інших шляхів, які з'єднують вузли S' та S'' більше не існує.

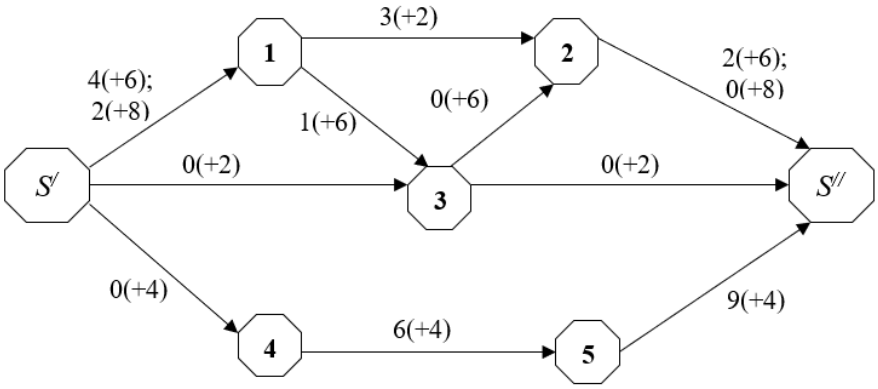

Рис. 10. Вирішення задачі про максимальний потік в мережі

Тоді, максимальний потік, який було пропущений нашою мережею, буде становити:

$\rho_{max} = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 6 + 4 + 2 + 2 = 14$.

Таким чином, максимальний обсяг інформації, який може бути переданий даною мережею за одиницю часу, від вершини S' до S'', або у зворотному напрямку, дорівнює 14 одиницям.