Київський національний університет імені Тараса Шевченка
Факультет комп'ютерних наук та кібернетики
Кафедра дослідження операцій

Випускна кваліфікаційна робота бакалавра
за спеціальністю 113 «Прикладна математика»
на тему:

Алгоритми розв'язання задачі про максимальний потік у мережі, порівняння та їх застосування

студента 4 курсу
Марухна Тараса Васильовича

Науковий керівник:
доцент, кандидат фізико-математичних наук
Якимів Роман Ярославович

Робота заслухана на засіданні кафедри дослідження операцій та рекомендована до захисту в ЕК, протокол № 9 від 23 травня 2023 р.

Завідувач кафедри ДО проф. Іксанов О.М.

Київ 2023

ЗМІСТ

ВСТУП

Задача про максимальний потік полягає в пошуку найбільшої кількості певного матеріалу, який можна перенести з одного місця в інше через задану мережу. У цій мережі встановлені обмеження на кількість матеріалу, який можна переносити між елементами мережі. Така задача має практичне застосування, й для її розв'язання розробили багато алгоритмів.

Метою дипломної роботи є дослідження алгоритмів розв'язання задач про максимальний потік, а саме: порівняння часу розв'язання при різних вхідних даних, визначення для яких розмірів умови задачі який алгоритм є швидшим і наскільки.

Задача про максимальний потік на практиці з'являється, наприклад, у таких випадках:

- Для знаходження максимальної кількості нафти, яку можна перенести від свердловини до нафтопереробних заводів через мережу нафтопроводів.
- У бейсбольних лігах якщо команда не має можливості виграти достатню кількість разів щоб закінчити сезон на першому місці, то вона вибуває. Обмеженнями в мережі тут є кількість ігор, які команда може провести з іншими командами.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ ПРО МАКСИМАЛЬНИЙ ПОТІК

1.1 Основні визначення

Перед формулюванням математичної постановки задачі про максимальний потік і розгляду алгоритмів її розв'язання варто навести тлумачення понять, які надалі будуть використовуватись.

Граф - це впорядкована пара $G = \{V, E\}$, де $V = \{i, j,...\}$ - непорожня множина вершин, $E = \{(i, j): i, j \in V\}$ - множина впорядкованих пар. Елементи множини E називають дугами, вершину i дуги (i, j) називають початком дуги, вершину j називають кінцем дуги.

Мережа - це граф, елементам якого поставлені у відповідність деякі параметри. Усі мережі в задачах про максимальний потік побудовані так: кожній вершині $i \in V$ поставлено у відповідність число d_i й кожній дузі $(i, j) \in E$ поставлено у відповідність число c_{ij}. Число c_{ij} називають функцією пропускної спроможності. Число d_i називають інтенсивністю вершини i. Якщо $d_i > 0$, то вершину i називають джерелом. Якщо $d_i < 0$, то вершину i називають стоком. Якщо $d_i = 0$, то вершину i називають нейтральною. d_i зазвичай позначає об'єм виробництва (при $d_i > 0$) або споживання (при $d_i < 0$) у вершині i.

Потік - це сукупність величин f_{ij}, які задовольняють такі умови:

$$\sum_{j:(i,j)\in E} f_{ij} - \sum_{k:(k,i)\in E} f_{ki} = d_i,\ i \in V,$$

$$0 \leq f_{ij} \leq c_{ij},\ (i,j) \in E.$$

Число $c_f(i,j) = c_{ij} - f_{ij}$ називають залишковою пропускною спроможністю.

1.2 Постановка задачі про максимальний потік

Розглянемо мережу, що задається графом G, яка має єдиний стік t, єдине джерело s й визначену на множині E функцію пропускної спроможності c_{ij}. Нехай інтенсивність джерела $d_s = d$, інтенсивність стоку $d_t = -d$. Допустимий потік для розглядуваної мережі визначається співвідношеннями:

$$\sum_{j:(s,j)\in E} f_{sj} = d,$$

$$\sum_{j:(i,j)\in E} f_{ij} - \sum_{k:(k,i)\in E} f_{ki} = 0,\ i \neq s,\ i \neq t,$$

$$-\sum_{k:(k,t)\in E} f_{kt} = -d,$$

$$0 \leq f_{ij} \leq c_{ij},\ (i,j) \in E.$$

Задача про максимальний потік полягає у знаходженні максимального значення інтенсивності d, при якому в мережі існує потік. Потік $f^{\star} = \left\{ f_{ij}^{\star}, (i,j) \in E \right\}$, що відповідає максимальному значенню інтенсивності $d^{\star}$, називають максимальним потоком, $d^{\star}$ називають величиною цього потоку.

РОЗДІЛ 2. АЛГОРИТМИ РОЗВ'ЯЗАННЯ ЗАДАЧ ПРО МАКСИМАЛЬНИЙ ПОТІК

2.1 Алгоритм Форда-Фалкерсона

Під час роботи алгоритму будемо покладати вершинам $i \in V$ мітки вигляду $[a_i, b_i]$, де a_i - величина потоку, яка надходить у вершину i , b_i - вершина, з якої надходить потік.

Покладаємо $f_{ij} = 0,\ (i,j) \in E$.

Далі виконуємо наступні кроки:

Крок 1. Покласти $i = s$. Дати джерелу мітку $[\infty, -]$. Перейти до кроку 2.

Крок 2. Знайти $S_i = \{j: c_f(i,j) > 0,\ j \text{ не має мітки}\}$. Перейти до кроку 3.

Крок 3. Якщо $S_i \neq \emptyset$, то перейти до кроку 4, інакше перейти до кроку 5.

Крок 4. Знайти вершину $k \in S_i$ для якої виконується $c_{ik} = max\{c_{ij}\}$ для $\forall j \in S_i$. Дати вершині k мітку $[c_{ik}, i]$. Якщо $k = t$, то перейти до кроку 6, інакше покласти $i = k$ і перейти до кроку 2.

Крок 5. Якщо $i = s$, то перейти до кроку 7, інакше вилучити i з множини S_{b_i}, покласти $i = b_i$ і перейти до кроку 3.

Крок 6. Використовуючи мітки створити множину $N = \{s,\dots, t\}$ вершин яка веде з джерела в стік. Знайти $f = min(a_i)$, $i \in N$. Покласти $f_{ij} = f$ і $f_{ji} = -f$, де $i, j \in N$ якщо потік йде з i в j, Прибрати всі мітки. Перейти до кроку 1.

Крок 7. Знайти $d^{\star} = \sum_{j:(s,j)\in E} f_{sj}$.

Складність алгоритму Форда-Фалкерсона становить $O(Ed^{\star})$.

2.2 Алгоритм Дініца

Перед формулюванням цього алгоритму введемо декілька визначень.

Граф $G_f = \{V, E_f\}$, де $E_f = \{(u, v) \in E: c_f(u, v) > 0\}$ називають залишковою мережею.

$dist(v)$ - довжина найкоротшого шляху з s до v у графі G_f.

Покладаємо $f_{ij} = 0,\ (i, j) \in E$.

Виконуємо наступні кроки:

Крок 1. Знайти $G_f = \{V, E_f\}$, де $E_f = \{(u, v) \in E: c_f(u, v) > 0\}$. Перейти до кроку 2.

Крок 2. Знайти $dist(v)$ - довжину найкоротшого шляху з s до v у графі G_f для всіх $v \in V$ використовуючи, наприклад, пошук в ширину. Якщо $\exists v \in V: dist(v) = \infty$, то $\sum_{j:(s,j)\in E} f_{sj}$ - відповідь і закінчити роботу алгоритма, інакше перейти до кроку 3.

Крок 3. Знайти $G_L = \{V, E_L\}$, де $E_L = \{(u,v) \in E_f : dist(v) = dist(u) + 1\}$. Перейти до кроку 4.

Крок 4. Знайти потік f' такий, що граф $G' = (V, E')$, де $E' = \{(u,v): f(u,v) < c_f(u,v), (u,v) \in E_L\}$ не містить шляху з джерела в стік. Перейти до кроку 5.

Крок 5. Обчислити $f \mathrel{+}= f'$. Перейти до кроку 1.

Знайти потік f' (його називають блокуючим) на кроці 4 можна знайти наступною рекурсивною функцією:

блокуючий потік$(u,$ мін_$c_f(i,j))$: # тут мін_$c_f(i,j)$ - це найменша пропускна спроможність на шляху від s до u.

ф = 0

for v: $(u,v) \in E$:

ф1 = 0

якщо $v == t$:

$\phi 1 = min$ (мін_$c_f(i,j), c_f(u,v)$)

$c_f(u,v) \mathrel{-}= \phi 1$

$c_f(v,u) \mathrel{+}= \phi 1$

$\phi \mathrel{+}= \phi 1$

інакше:

$\phi 1 \mathrel{+}=$ блокуючий потік$(v, min(c_f(u,v),$

мін_$c_f(i,j)$))

мін_$c_f(i,j) \mathrel{-}= \phi 1$

$c_f(u,v) \mathrel{-}= \phi 1$

$c_f(v,u) \mathrel{+}= \phi 1$

$\phi \mathrel{+}= \phi 1$

$return\ \phi$

Для знаходження блокуючого потоку f' на кроці 4 потрібно обчислити $f' =$ блокуючий потік(s, ∞). У програмній реалізації замість ∞ можна використати $\sum_{v:(u,v)\in E'} c_f(u,v)$. Під час роботи цієї функції буде

знайдено значення f', а також змінені $c_f(i,j)$ у графі, з яких можна знайти f_{ij} так: $f_{ij} = c_{ij} - c_f(i,j)$

Складність алгоритму Дініца становить $O\left(V^2E\right)$.

2.3 Алгоритм Едмондса-Карпа

Покладаємо $f_{ij} = 0,\ (i,j) \in E$.

Виконуємо наступні кроки:

Крок 1. Знайти найкоротший шлях з джерела в стік використовуючи пошук в ширину. Якщо такого шляху нема, то $\sum_{j:(s,j)\in E} f_{sj}$ - відповідь і закінчити роботу алгоритма, інакше перети до кроку 2.

Крок 2. Знайти на знайденому шлясі дугу з найменшою пропускною спроможністю c_{min}. Перейти до кроку 3.

Крок 3. Для кожного ребра на знайденому шлясі збільшити потік на c_{min} у напрямку від джерела до стоку й зменшити потік на c_{min} у протилежному напрямку. Перейти до кроку 1.

Складність алгоритму Едмондса-Карпа становить $O\left(VE^2\right)$.

2.4 Алгоритм проштовхування передпотоку

Введемо 2 позначки перед формулюванням алгоритму:

$\chi_f(u)$ - надлишок вершини u.

$$\chi_f(u) = \sum_{v \in V} f(v,u) - \sum_{v \in V} f(u,v).$$

$l(u)$ - висота вершини u.

Визначимо функції, які потрібні для формулювання алгоритму, в псевдокоді.

проштовхування(u, v):

перед виконанням повинна бути перевірка чи виконується (

$\chi_f(u) > 0$ та $l(u) = l(v) + 1$)

$$\text{потік} = min(\chi_f(u), c_{uv} - f_{uv})$$
$$f_{uv} \mathrel{+}= \text{потік}$$
$$f_{vu} \mathrel{-}= \text{потік}$$
$$\chi_f(u) \mathrel{-}= \text{потік}$$
$$\chi_f(v) \mathrel{+}= \text{потік}$$

перемаркування(u):

перед виконанням повинна бути перевірка чи виконується (

$\chi_f(u) > 0$ та $l(u) <= l(v)$ для всіх v таких що $c_f(u,v) > 0$)

$l(u) = 1 + min(l(v)$ для всіх v таких що $c_f(u,v) > 0)$

розряд(u):

поки $\chi_f(u) > 0$:

якщо u вичерпало всіх сусідів:

перевірка умов перемаркування

$\text{перемаркування}(u)$

інакше:

перевірка умов проштовхування

$\text{проштовхування}(u, v)$

Ще потрібно обрати спосіб обрання активного вузла. У свої реалізації я написав FIFO (first in first out).

Алгоритм проштовхування передпотоку має наступний вигляд:

- покласти $l(s) = |V|, \chi_f(s) = \infty, l(v) = 0$ для всіх $v \in V \backslash \{s\}$
- для всіх v таких що $(s, v) \in E$:
  - $\text{проштовхування}(s, v)$
- поки можна обрати наступний активний вузол:
  - $\text{розряд}(u)$
- return $\sum_{j:(s,j)\in E} f_{sj}$

Складність алгоритму проштовхування передпотоку з правилом вибору вершин FIFO становить $O(V^3)$.

РОЗДІЛ 3. ПОРІВНЯННЯ АЛГОРИТМІВ

Для порівняння алгоритмів проведемо такий тест: згенеруємо 20 графів (спосіб генерування буде наведений нижче) із наперед заданою кількістю вершин і максимальною пропускною спроможністю дуг. Кількість ребер буде визначатись способом генерування. Збережемо згенеровані графи в масиві, розв’яжемо всі ці задачі алгоритмами Форда-Фалкерсона, проштовхування передпотоку, Едмондса-Карпа та Дініца. Вимірюємо час затрачений на розв’язок кожної задачі. Для порівняння алгоритмів будемо використовувати сумарний час затрачений на розв’язок усіх 20 задач. Для зручності проведення тесту будемо задавати відразу декілька кількостей вершин, для кожної кількості генерувати 20 графів та робити вимірювання.

Спосіб генерування графів є таким: створюємо 3 вершини і з'єднуємо їх дугами так: 1<—>2<—>3 (вершини пронумеровані). Далі циклом додаємо вершини й дуги наступним чином: створюємо нову вершину й випадково з'єднуємо її з трьома іншими вершинами в графі. Дуги додаються в обох напрямках. Цикл зупиняється, коли кількість вершин у графі стає рівною заданій кількості вершин. Усім дугам генеруємо пропускні спроможності від 1 до [максимум]. Вершина з номером 1 є джерелом. Вершина з найбільшим номером є стоком. Отже, для n вершин кількість ребер буде $2 + 3 * (n - 3)$.

У всіх тестах максимальною пропускною спроможністю дуг буде 10.

У таблиці нижче наведено результати першого тесту. Оскільки в більшості випадків алгоритм Форда-Фалкерсона був найповільнішим, то для порівняння будемо використовувати відношення часу затраченого алгоритмом Форда-Фалкерсона на розв’язання 20 задач до часу затраченого іншими алгоритмами на розв’язання тих самих 20 задач. Також біля відношень часів робіт алгоритмів буду писати t=[число]. t - це час в секундах, який був витрачений на розв’язання 20 задач. Це число може значно відрізнятись при тестуваннях на інших комп’ютерах і залежить від продуктивності ноутбука, але відношення часів робіт алгоритмів повинні залишатись схожими. Процесор ноутбука на якому проводились тести - Intel Core i5-8250U CPU @ 1.60GHz 1.80 GHz.

Таблиця 4.1

Кількість вершин	Кількість ребер	Форд-Фалкерсон/ Форд-Фалкерсон	Форд-Фалкерсон/ проштовхування передпотоку	Форд-Фалкерсон/ Едмондс-Карп	Форд-Фалкерсон/ Дініц
7	14	1 t=0.0070	0.6808 t=0.0102	7.5868 t=0.0009	4.2210 t=0.0016
11	26	1 t=0.2547	6.4939 t=0.0392	120.44 t=0.0021	84.578 t=0.0030
16	41	1 t=20.629	198.39 t=0.1039	5655.7 t=0.0036	4311.5 t=0.0047
19	50	1 t=212.60	1409.4 t=0.1508	46354 t=0.0045	36700 t=0.0057
20	53	1 t=447.87	2495.8 t=0.1794	76377 t=0.0058	68610 t=0.0065

За результатами цих тестів видно, що для таких згенерованих графів з кількістю вершин до 20 алгоритм Едмондса-Карпа є найшвидшим, після Едмондса-Карпа найшвидшим є алгоритм Дініца. Для графів з 7 вершинами найповільнішим був алгоритм проштовхування передпотоку, але для більшої кількості вершин найповільнішим став алгоритм Форда-Фалкерсона, й він ставав дедалі повільнішим у порівнянні з іншими алгоритмами при збільшенні кількості вершин.

Оскільки алгоритм Форда-Фалкерсона витратив 7.5 хвилин на розв’язання 20 задач з 20 вершинами, й збільшення часу роботи з 19

вершин до 20 вершин складає 4 хвилини, то алгоритм Форда-Фалкерсона буде вилучений з тестів для більшої кількості вершин.

Оскільки після алгоритму Форда-Фалкерсона найповільнішим був алгоритм проштовхування передпотоку, то в наступних тестах для порівняння будемо використовувати відношення часу проштовхування передпотоку/інший алгоритм.

Таблиця 4.2

Кількість вершин	Кількість ребер	проштовхування передпотоку/ проштовхування передпотоку	проштовхуванн я передпотоку/ Едмондс-Карп	проштовхува ння передпотоку/ Дініц
16	41	1 t=0.1058	27.534 t=0.0038	22.321 t=0.0047
20	53	1 t=0.1811	32.182 t=0.0056	28.292 t=0.0064
35	98	1 t=0.6352	48.881 t=0.0129	49.115 t=0.0129
40	113	1 t=0.8216	50.147 t=0.0163	45.050 t=0.0182
100	293	1 t=5.9139	67.699 t=0.0873	106.52 t=0.0555
150	443	1 t=13.622	71.096 t=0.1916	131.61 t=0.1035
200	593	1 t=25.318	79.160 t=0.3198	164.65 t=0.1537
250	743	1 t=42.005	80.943 t=0.5189	183.97 t=0.2283
300	893	1 t=64.243	86.941 t=0.7389	209.00 t=0.3073

За результатами цих тестів видно, що для таких згенерованих графів з кількістю вершин до 20 алгоритм Едмондса-Карпа є найшвидшим, при кількості вершин 35 і 40 алгоритми Едмондса-Карпа та Дініца приблизно однакові, та при кількості вершин від 100 алгоритм Дініца стає помітно швидшим порівняно з алгоритмом Едмондса-Карпа.

Точну межу коли алгоритм Дініца стає найшвидшим знайти не вдасться тому, що при кожному запуску тесту результати трішки відрізняються. Така похибка виникає через те, що комп'ютер одні й ті самі обчислення може виконувати за різну кількість часу, та час роботи може залежати від згенерованих графів. Деякі графи алгоритми розв'язують швидше ніж інші, навіть якщо кількість кількість вершин і ребер однакова. Графи відрізняються множиною ребер (між якими вершинами буде ребро визначається випадково) та пропускними спроможностями (теж визначається випадково для кожної дуги).

Далі порівняємо часи виконання алгоритмів збільшивши кількість ребер при тих самих кількостях вершин. Для цього змінимо алгоритм генерування графів: створюємо 6 вершин і з'єднуємо їх дугами так: 1<—>2<—>...<—>6. Далі циклом додаємо вершини й дуги наступним чином: створюємо нову вершину й випадковим чином з'єднуємо її з шістьма іншими вершинами в графі. Дуги додаються в обох напрямках. Цикл зупиняється, коли кількість вершин у графі стає рівною заданій

кількості вершин. Усім дугам генеруємо пропускні спроможності від 1 до [максимум]. Вершина з номером 1 є джерелом. Вершина з найбільшим номером є стоком. У такому зміненому способі генерування графів при n вершин кількість ребер буде $5 + 6 * (n - 6)$.

Таблиця 4.3

Кількість вершин	Кількість ребер	проштовхування передпотоку/ проштовхування передпотоку	проштовхування передпотоку/ Едмондс-Карп	проштовхування передпотоку/ Дініц
11	35	1 t=0.0316	12.900 t=0.0024	9.0590 t=0.0034
16	65	1 t=0.1125	20.021 t=0.0056	20.626 t=0.0054
20	89	1 t=0.2338	22.395 t=0.0104	26.744 t=0.0087
40	209	1 t=1.2435	30.202 t=0.0411	57.108 t=0.0217
100	569	1 t=9.3688	38.782 t=0.2415	133.54 t=0.0701
150	969	1 t=21.338	43.328 t=0.4924	166.71 t=0.1279

За результатами цих тестів видно, що для таких згенерованих графів з 11 вершинами алгоритм Едмондса-Карпа ще є швидшим ніж алгоритм

Дініца, але вже при 16 та 20 вершинах вони стають приблизно однаковими, й при 40 вершинах алгоритм Дініца стає швидшим.

Отже, збільшення кількості ребер при незмінній кількості вершин призводить до того, що кількість вершин, при якій алгоритм Дініца стає швидшим ніж алгоритм Едмондса-Карпа, стає меншою. Також можна помітити, що алгоритм Едмондса-Карпа став повільнішим порівняно з алгоритмом проштовхування передпотоку: у тесті до збільшення кількості ребер при 100 вершинах він був швидшим у 67 разів, а після збільшення кількості ребер став у 38 разів швидшим.

Спробуємо ще збільшити кількість ребер. Тепер на початку буде створено 18 вершин, і в циклі нові вершини будуть з’єднані з 18 іншими вершинами.

Таблиця 4.4

Кількість вершин	Кількість ребер	проштовхування передпотоку/ проштовхування передпотоку	проштовхування передпотоку/ Едмондс-Карп	проштовхування передпотоку/ Дініц
25	143	1 t=0.0360	5.5493 t=0.0064	4.2074 t=0.0085
40	413	1 t=0.5656	7.0407 t=0.0803	17.251 t=0.0327
70	953	1 t=8.4714	16.237 t=0.5217	126.69 t=0.0668
100	1493	1	17.323	184.07

		t=19.978	t=1.1532	t=0.1085

Після ще одного збільшення кількості ребер зміна швидкості алгоритму Едмондса-Карпа при збільшенні кількості вершин порівняно з алгоритмом проштовхування передпотоку стала ще меншою. Також стало помітно, що при менших кількостях вершин, швидкість роботи алгоритму Дініца порівняно з алгоритмом проштовхування передпотоку теж стає меншою, але при більших кількостях вершин час роботи алгоритму Дініца порівняно з часом роботи алгоритму проштовхуванням передпотоку стає більшим ніж у попередніх тестах.

ВИСНОВКИ

Алгоритм Форда-Фалкерсона є найповільнішим алгоритмом розв’язання задачі про максимальний потік серед розглянутих. Його можна застосовувати лише для розв’язання задач дуже малих розмірів на кшталт 11 вершин та 26 ребер, і його швидкість буде меншою лише в 120 разів порівняно з алгоритмом Едмондса-Карпа, який виявився найшвидшим при такому розмірі, але вже при 20 вершинах та 53 ребрах буде в десятки тисяч разів повільнішим ніж алгоритми Дініца та Едмондса-Карпа.

Алгоритм проштовхування передпотоку в усіх випадках, окрім 7 вершин і 14 ребер, був швидшим ніж алгоритм Форда-Фалкерсона, але повільнішим ніж алгоритми Едмондса-Карпа та Дініца. При збільшенні кількості ребер відношення часу роботи алгоритму Едмондса-Карпа до алгоритму проштовхування передпотоку ставало меншим, але менше 1 не було.

У кожному тесті, де були графи розмірами більші ніж 20 вершин та 53 ребер, починаючи з певної кількості вершин та ребер алгоритм Дініца ставав найшвидшим й іноді навіть в декілька разів порівняно з алгоритмом Едмондса-Карпа. Ще було помічено, що при збільшенні кількості ребер, кількість вершин, при якій алгоритм Дініца стає швидшим, зменшується.

З цього можна зробити висновок, що найшвидшими алгоритмами розв’язання задачі про максимальний потік є алгоритми Едмондса-Карпа та Дініца. Алгоритм Едмондса-Карпа варто застосовувати, коли мережі не є великими (приблизно до 20 вершин). Для більших мереж час роботи алгоритму Дініца буде приблизно таким як Едмондса-Карпа, або швидше.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Попов Ю.Д., Тюптя В.І., Шевченко В.І. Методи оптимізації. Навчальний електронний посібник для студентів спеціальностей “Прикладна математика”, “Інформатика”, “Соціальна інформатика”. – Київ: Електронне видання. Ел. бібліотека факультету кібернетики Київського національного університету імені Тараса Шевченка, 2003.–215 с.
2. Hamdy Taha. Operations Research: An Introduction 10th Edition. – Pearson, 2016. – 848 c.
3. Oded Goldreich, Arnold L. Rosenberg, Alan L. Selman. Theoretical Computer Science: Essays in Memory of Shimon Even. – Springer, 2006. – 398 pg.
4. Ravindra K. Ahuja, Thomas L. Magnanti, James B. Orlin. Network flows: Theory, algorithms and applications. – Prentice-Hall, Inc. Upper Saddle River, New Jersey 07458, 1993 – 848 pg.
5. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms. – MIT Press, 1990. – 1312 pg.
6. Jonathan L. Gross, Jay Yellen. Graph Theory and Its Applications 2nd Edition. – Chapman and Hall/CRC, 2005. – 800 pg.

7. Christos H. Papadimitriou, Kenneth Steiglitz. Combinatorial Optimization: Algorithms and Complexity. – Dover Publications, 1998. – 528 pg.

8. Mark Newman. Networks: An Introduction 1st Edition. – Oxford University Press, 2010. – 772 pg.

9. Jeremy Kepner, John Gilbert. Graph Algorithms in the Language of Linear Algebra. – Dover Publications, 1998. – 528 pg.

10. Gabriel Valiente. Algorithms on Trees and Graphs. – Springer, 2002. – 450 pg

11. Alfred Aho. Design and Analysis of Computer Algorithms, The 1st Edition. – Addison-Wesley, 1974. – 470 pg.

12. L.R. Ford, D.R. Fulkerson. Flows in Networks. Princeton University Press, 41 William St. Princeton, NJ, 1962. – 208 pg.

Додаток А.Код програми

```
from copy import deepcopy

from random import randint, choice

from timeit import timeit

"""

n - це кількість вершин. вершина з номером 1  це джерело, вершина з номером n це стік

дані про ребра зберігаються в словниках такого вигляду:

{..., вершина: {інша вершина: пропускна спроможність в напрямку до іншої вершини, ...}, ...}

"""



def ford_falkerson(graf):

    n = len(graf.keys())

    f = []  # потоки

    S = {}  # множина суміжних вершин

    mitky = {1: [sum(graf[1].values())*10, None]}
```

```
i = 1
krok5 = False


while True:
    k = 0
    max_c_ik = 0
    if not krok5:
        S[i] = []
        for vershyna in graf[i].keys():
            if (graf[i][vershyna] > 0) and (vershyna not in mitky.keys()):
                S[i].append(vershyna)
                if graf[i][vershyna] > max_c_ik:
                    k = vershyna
                    max_c_ik = graf[i][vershyna]
    else:
        for vershyna in S[i]:
            if (vershyna not in mitky.keys()) and graf[i][vershyna] > max_c_ik:
```

```
            k = vershyna
            max_c_ik = graf[i][vershyna]
    krok5 = False


if k == 0:  # S[i] == []
    if i == 1:
        return sum(f)
    else:
        krok5 = True
        S[mitky[i][1]].remove(i)
        i = mitky.pop(i)[1]
        continue


mitky[k] = [max_c_ik, i]


if k == n:
    min_c_ik = mitky[1][0]
```

```
        for vershyna in mitky.keys():
            if mitky[vershyna][0] < min_c_ik:
                min_c_ik = mitky[vershyna][0]
        f.append(min_c_ik)

        mitky.pop(1)  # щоб не намагалось змінити залишкові пропускні спроможності для потоку що йде в джерело
        for vershyna in mitky.keys():
            graf[mitky[vershyna][1]][vershyna] -= min_c_ik
            graf[vershyna][mitky[vershyna][1]] += min_c_ik
        mitky = {1: [sum(graf[1].values()) * 10, None]}
        i = 1
        S = {}

    else:
        i = k
```

```
def dinits(graf):
    n = len(graf.keys())
    f = 0
    while True:
        graf_L = {}
        dist1 = {0: [1]}  # {відстань до джерела: [номер вершини]}
        dist2 = {}  # {номер вершини: відстань до джерела}
        neproydeni_vershyny = []
        for i in range(1, n+1):
            dist1[i] = []
            graf_L[i] = {}
            neproydeni_vershyny.append(i)
            dist2[i] = n+1  # після цикла зроблю {1: 0}
        dist2[1] = 0
        neproydeni_vershyny.remove(1)


        for vidstan in range(0, n + 1):  # пошук в ширину
```

```
            for vershyna1 in dist1[vidstan]:
                for vershyna2 in graf[vershyna1].keys():
                    if (vershyna2 in neproydeni_vershyny) and
(graf[vershyna1][vershyna2] != 0):
                        neproydeni_vershyny.remove(vershyna2)
                        dist1[vidstan + 1].append(vershyna2)
                        dist2[vershyna2] = vidstan + 1
                        graf_L[vershyna1][vershyna2] = graf[vershyna1][vershyna2]
                    elif dist2[vershyna2] == vidstan + 1:
                        graf_L[vershyna1][vershyna2] = graf[vershyna1][vershyna2]

        if dist2[n] == n+1:
            return f

        f += blokujuchyj_potik(n, graf, graf_L, 1, sum(graf_L[1].values()) * 10)


def blokujuchyj_potik(n, graf, graf_L: dict, vershyna1, najmenshe_c_ij):
```

```
    f = 0
    for vershyna2 in graf_L[vershyna1].keys():
        f1 = 0
        if vershyna2 == n:
            f1 = min(najmenshe_c_ij, graf_L[vershyna1][vershyna2])
            graf_L[vershyna1][vershyna2] -= f1
            graf[vershyna1][vershyna2] -= f1
            graf[vershyna2][vershyna1] += f1
            f += f1
        else:
            f1 += blokujuchyj_potik(n, graf, graf_L, vershyna2,
min(najmenshe_c_ij,
                                    graf_L[vershyna1][vershyna2]))
            najmenshe_c_ij -= f1
            graf_L[vershyna1][vershyna2] -= f1
            graf[vershyna1][vershyna2] -= f1
            graf[vershyna2][vershyna1] += f1
            f += f1
```

```
    return f



def zgeneruvaty_graf(n, max_c_ij, kilkist_zjednan):
    graf = {}
    graf[1] = {}
    graf[1][2] = randint(1, max_c_ij)
    for vershyna in range(2, kilkist_zjednan):
        graf[vershyna] = {}
        graf[vershyna][vershyna - 1] = randint(1, max_c_ij)
        graf[vershyna][vershyna + 1] = randint(1, max_c_ij)
    graf[kilkist_zjednan] = {}
    graf[kilkist_zjednan][kilkist_zjednan - 1] = randint(1, max_c_ij)


    for vershyna in range(kilkist_zjednan + 1, n + 1):
        vershyny = list(graf.keys())
        graf[vershyna] = {}
```

```
        for i in range(0, kilkist_zjednan):
            obrana_vershyna = choice(vershyny)
            vershyny.remove(obrana_vershyna)
            graf[vershyna][obrana_vershyna] = randint(1, max_c_ij)
            graf[obrana_vershyna][vershyna] = randint(1, max_c_ij)
    return graf



def edmonds_karp(graf):
    n = len(graf.keys())
    f = 0
    while True:
        zbilshujuchyj_shlyah = najkorotshyj_shljah(n, graf)
        if not zbilshujuchyj_shlyah:  # zbilshujuchyj_shlyah == []
            return f
        vershyna = n
        min_c_ij = graf[zbilshujuchyj_shlyah[vershyna]][vershyna]
```

```
        while vershyna != 1:
            min_c_ij = min(graf[zbilshujuchyj_shlyah[vershyna]][vershyna],
min_c_ij)
            vershyna = zbilshujuchyj_shlyah[vershyna]
        f += min_c_ij
        vershyna = n
        while vershyna != 1:
            graf[zbilshujuchyj_shlyah[vershyna]][vershyna] -= min_c_ij
            graf[vershyna][zbilshujuchyj_shlyah[vershyna]] += min_c_ij
            vershyna = zbilshujuchyj_shlyah[vershyna]



def najkorotshyj_shljah(n, graf):
    O = [1]
    mitky = {1: None}
    vidvidani = [1]
    while O:  # O != []
        u = O.pop(0)
```

```
        for v in graf[u].keys():
            if graf[u][v] == 0:
                continue
            if v not in vidvidani:
                vidvidani.append(v)
                mitky[v] = u
                O.append(v)
                if v == n:
                    vidpovid = {}
                    while v is not None:
                        vidpovid[v] = mitky[v]
                        v = mitky[v]
                    return vidpovid
    return []


def proshtovhuvannja_peredpotoku(graf):
```

```
    """правило вибору активного вузла: перший увійшов перший вийшов
(FIFO)"""
    n = len(graf.keys())
    vysota = {}
    nadlyshok = {}
    peredpotik = {}
    cherga = []
    for vershyna1 in graf.keys():
        vysota[vershyna1] = 0
        nadlyshok[vershyna1] = 0
        peredpotik[vershyna1] = {}
        cherga.append(vershyna1)
        for vershyna2 in graf[vershyna1]:
            peredpotik[vershyna1][vershyna2] = 0
    vysota[1] = n
    nadlyshok[1] = sum(graf[1].values())*10
    cherga.remove(1)
    cherga.remove(n)
```

```
    for v in graf[1].keys():
        potik = graf[1][v]
        peredpotik[1][v] += potik
        peredpotik[v][1] -= potik
        nadlyshok[v] += potik

    kilkist_projdenyh_vershyn_bez_zmin = 0
    while kilkist_projdenyh_vershyn_bez_zmin != n - 2: # якщо = n-2 то це значить що операції проштовхування й перемаркування незастосовні до всіх вершин
        cherga.append(cherga[0])
        u = cherga.pop(0)

        if nadlyshok[u] > 0:
            kilkist_projdenyh_vershyn_bez_zmin = 0

            # розряд
```

```
while nadlyshok[u] > 0:
    nema_susidiv = True
    for v in graf[u].keys():
        if vysota[u] == vysota[v] + 1:
            # проштовхування
            potik = min(nadlyshok[u], graf[u][v] - peredpotik[u][v])
            if potik == 0:
                continue
            peredpotik[u][v] += potik
            peredpotik[v][u] -= potik
            nadlyshok[u] -= potik
            nadlyshok[v] += potik
            nema_susidiv = False
    if nema_susidiv:
        # перемаркування
        umovy_peremarkuvannja_vykonujutsja = True
        min_vysota_v = vysota[u]
```

```
                for v in graf[u].keys():
                    if graf[u][v] - peredpotik[u][v] > 0:
                        if vysota[u] <= vysota[v]:
                            min_vysota_v = min(min_vysota_v, vysota[v])
                        else:
                            umovy_peremarkuvannja_vykonujutsja = False
                            break
                if umovy_peremarkuvannja_vykonujutsja:
                    vysota[u] = 1 + min_vysota_v
        else:
            kilkist_projdenyh_vershyn_bez_zmin += 1
    return sum(peredpotik[1].values())



def chas_vykonannya_algorytmu(algorytm_str, grafy):
    vidpovid = 0
```

```
    for i in range(0, len(grafy)):
        print(f"i = {i}")
        print(grafy[i])
        setup = f"from __main__ import {algorytm_str}, deepcopy; graf_kopija =
deepcopy(graf)"
        globals_dict = {"graf": grafy[i]}
        chas = timeit(f"{algorytm_str}(graf_kopija)", setup=setup,
globals=globals_dict, number=5)
        # роздокументувати щоб побачити результат роботи алгоритмів. ще
треба розкоментовати algorytmy_func
        '''
        graf_kopija = deepcopy(grafy[i])
        algorytm = algorytmy_func[algorytmy.index(algorytm_str)]
        rez = algorytm(graf_kopija)
        print(f"rez = {rez}")
        print(f"chas = {chas}")
        '''
        vidpovid += chas
```

```
    return vidpovid



def porahuty_kilkist_reber(graf: dict):
    E = 0
    for vershyna1 in graf.keys():
        for vershyna2 in graf[vershyna1].keys():
            E += 1
    return E//2



max_c_ij = 10
kilkist_zjednan = 6
kilkist_vymirjuvan = 20
kilkosti_vershyn = [25, 40]
algorytmy = ["proshtovhuvannja_peredpotoku", "edmonds_karp", "dinits"]
#algorytmy_func = [proshtovhuvannja_peredpotoku, edmonds_karp, dinits]
```

```
#algorytmy_func = [ford_falkerson, proshtovhuvannja_peredpotoku,
edmonds_karp, dinits]
#algorytmy = ["ford_falkerson", "proshtovhuvannja_peredpotoku",
"edmonds_karp", "dinits"]
spysok_grafiv = []
for kilkist_vershyn in kilkosti_vershyn:
    spysok_grafiv.append({"V": kilkist_vershyn, "E": 0, "grafy": [], "chas": {}})


for i in range(0, len(spysok_grafiv)):
    print(f"V = {spysok_grafiv[i]['V']}")
    for j in range(0, kilkist_vymirjuvan):
        novyj_graf = zgeneruvaty_graf(spysok_grafiv[i]["V"], max_c_ij,
kilkist_zjednan)
        spysok_grafiv[i]["grafy"].append(deepcopy(novyj_graf))
    for algorytm in algorytmy:
        print(algorytm)
        spysok_grafiv[i]["chas"][algorytm] =
chas_vykonannya_algorytmu(algorytm,
```

```
                                                spysok_grafiv[i]["grafy"])
    spysok_grafiv[i]["E"] = porahuty_kilkist_reber(spysok_grafiv[i]["grafy"][0])



print()
for dani_pro_grafy in spysok_grafiv:
    print(f"V = {dani_pro_grafy['V']}, E = {dani_pro_grafy['E']}")
    for algorytm in algorytmy:
        print(f"{algorytm} = {dani_pro_grafy['chas'][algorytm]}")
    print()
    for algorytm in algorytmy[1:]:
        print(f"{algorytmy[0]}/{algorytm} = " +

f"{dani_pro_grafy['chas'][algorytmy[0]]/dani_pro_grafy['chas'][algorytm]}")
    print()
```