МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ЦЕНТР ЗАОЧНОЇ, ДИСТАНЦІЙНОЇ ТА ВЕЧІРНЬОЇ ФОРМ НАВЧАННЯ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

СЕКЦІЯ ІКТ

ВИПУСКНА РОБОТА

на тему:

«Комп'ютерний порівняльний аналіз алгоритмів Дініца та Форда-Фалкерсона»

Завідувач

випускаючої кафедри **Довбиш А. С.**

Керівник роботи **Шаповалов С. П.**

Студент групи ІНз – 61С **Ключка Т. А.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Центр заочної, дистанційної і вечірньої форм навчання
Кафедра комп'ютерних наук
Секція ІКТ

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2020 р.

ЗАВДАННЯ

до випускної роботи

Студента п'ятого курсу, групи ІНз-61С спеціальності "Інформатика" заочної форми навчання Ключки Тараса Анатолійовича

Тема: " Комп'ютерний порівняльний аналіз алгоритмів Дініца та Форда-Фалкерсона»

Затверджена наказом по СумДУ

№_____ от _____2020 р.

Зміст пояснювальної записки: 1) аналітичний огляд методів пошуку максимальних потоків в графах; 2) постановка завдання й формування завдань дослідження; 3) опис основних положень, математичних моделей і алгоритмів, що використовуються для рішення поставленого завдання; 5) розробка інформаційного й програмного забезпечення; 6) аналіз результатів моделювання.

Дата видачі завдання "_____"_____2020 р.

Керівник випускної роботи _____ Шаповалов С. П.

Завдання приняв до виконання _____ Ключка Т. А.

РЕФЕРАТ

Записка: 44 стор., 18 рис., 9 табл., 2 додатки, 9 джерел інформації.

Об’єкт дослідження — алгоритми знаходження максимального потоку в графі.

Мета роботи — комп’ютерний порівняльний аналіз алгоритмів знаходження максимальних потоків в графах.

Методи дослідження — математичне моделювання, комп’ютерна реалізація алгоритмів на ЕОМ.

Результати — розроблено інформаційне та програмне забезпечення комп’ютерного порівняльного аналізу алгоритмів знаходження максимальних потоків в графі. Проведено огляд відомих рішень, обрані алгоритми Дініца та Форда-Фалкерсона для проведення комп’ютерного порівняльного аналізу. Розроблено комп’ютерну реалізацію за допомогою алгоритмічної мови програмування С++.

ЗНАХОДЖЕННЯ МАКСИМАЛЬНИХ ПОТОКІВ В ГРАФАХ,
АЛГОРИТ ДІНІЦА, АЛГОРИТМ ФОРДА-ФПКЕРСОНА,
ПОРІВНЯЛЬНИЙ КОМП’ЮТЕРНИЙ АНАЛІЗ,
МОВА ПРОГРАМУВАННЯ С++

ЗМІСТ

ВСТУП

Мережеві моделі мають в сьогоденні один з основних базових типів моделювання в інформаційних та комп’ютерних науках. Мережі, як графові моделі, дозволяють розв’язати великий пласт проблем, починаючи від повсякденних і закінчуючи великими транспортними артеріями [1-5].

До основних досліджень в рамках мережевого моделювання слідує віднести наступні: 1. Знаходження мінімальних шляхів (the shortest path problem); 2. Знаходження максимальних потоків (maximum network flow problem);
3. Знаходження каркасних (остових) дерев мінімальної ваги (minimum spanning tree): 4. Знаходження ємкісної мережі мінімальної вартості (Minimum-cost capacitated network model) [3]. Це змушує дистриб'юторські компанії, що надають послуги в транспортних мережах вважати основними очільниками в умовах конкурентного середовища для просування продукції - пропускна здатність, час та вартість.

Оптимізаційні завдання з мережевим потоком складають важливий клас проблем оптимізації і є центральними проблемами в дослідженні операцій, інформатиці та комбінаторній оптимізації.

Завдання про максимальний потік в мережі вивчається значний період часу. Інтерес до неї підігрівається величезною практичною значущістю цієї проблеми. Методи вирішення задачі застосовуються на транспортних, комунікаційних, електричних мережах, при моделюванні різних процесів фізики і хімії, в деяких операціях над матрицями, для вирішення родинних завдань теорії графів, і навіть для пошуку Web-груп в WWW.

У завданнях на максимальний потік виконується переміщення об'єктів будь-якої природи через мережу. Розподіляються чи нафтогазові продукти, чи іграшки і одяг в магазини уцінених товарів по автомагістралях країни, чи біти інформації по мережах зв'язку для відображення на моніторах у всіх куточках світу - по суті, це одна і те ж завдання. Алгоритми їх розв'язання широко застосовуються як окремі продукти, так і в численних додатках.

1 АНАЛІТИЧНИЙ ОГЛЯД ПРОБЛЕМИ

1.1 Алгоритми та методи пошуку максимальних потоків в графах

Функціональне призначення більшості фізично реалізованих мереж полягає в тому, що вони служать носіями систем потоків, тобто систем, в яких деякі об'єкти течуть, рухаються або транспортуються по системі каналів (дуг мережі) з обмеженою пропускною здатністю. Обмежена пропускна здатність означає, що інтенсивність переміщення відповідних об'єктів по каналу обмежена зверху певною величиною. Й важливим завданням послуговує рішення проблеми – який максимально можливий потік ця мережа може передати [1-3].

Прикладами можуть служити потоки автомобільного транспорту по мережі автодоріг, вантажів по ділянці залізничної мережі, води у міській мережі водопостачання, електричного струму в електромережі, телефонних або телеграфних повідомлень по каналах зв'язку, програм в обчислювальній мережі. Нижче приведена спрощена модель залізничної мережі (див. рис. 1.1), що є однією з перших моделей розв’язання задачі «maximum network flow problem» (1954 by T. E. Harris and F. S. Ross) [1].

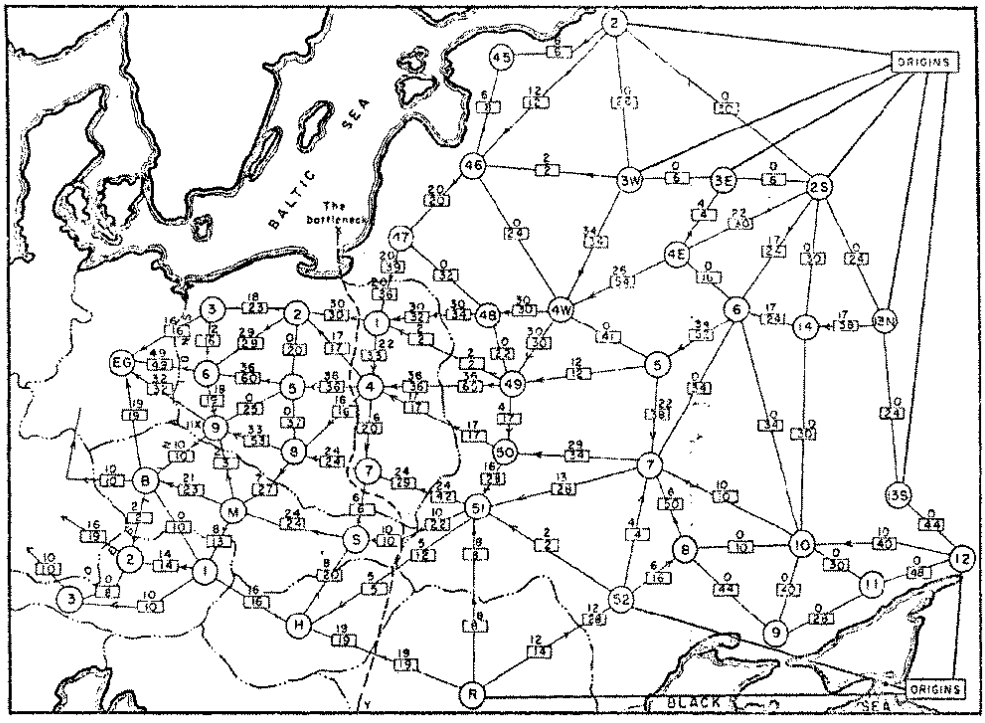

Рисунок 1. 1 – Спрощена модель залізничної мережі

Отже, нас цікавить проблема знаходження максимального потоку в мережі, в англійських джерелах інформації така проблема має назву «maximum network flow problem».

Як взятися за вирішення цього завдання?

З точки зору математики та математичного моделювання для рішення поставленої проблеми, ці дослідження повинні опиратися на модель, що складається з заданого зваженого графа G(V,E) (задається множиною вершин - V та множиною ребер E, на якій задано дійсно-значиму функцію ваги $f : E \rightarrow R$), та визначену функцію цілі, оптимальне значення якої ми будемо відшукувати в рамках потрібного. В даному разі ціль – відшукання потоку максимально можливого, що може передаватися утвореною мережею.

З точки зору інформаційно-комунікаційних технологій, рішення проблеми «maximum network flow problem» потребує декілька речей: 1. Вибір структури даних для моделювання; 2. Вибір парадигми та алгоритму програмування; 3. Вибір алгоритмічної мови для реалізації обраних алгоритмів; 4. Комп'ютерна реалізація алгоритмів на надання рекомендацій для практичного впровадження.

Річ 1. На рис. 1.2 в якості структури даних обрано зважений орієнтовний граф, що не тільки змістовно подає графічний образ мережі, але й дозволяє на «простих мережах» розв'язати поставлену проблему перебором. Для даного графу очевидним є рішення – максимальний потік дорівнює 5.

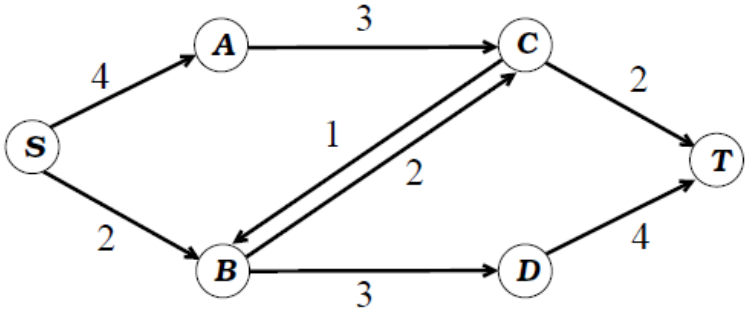

Рисунок 1. 2 – Діаграма мережевого потоку

Відносно потокових мереж, з якими ми будемо мати справу, пов'язані деякі означення та припущення, про які ми зазначимо нижче [1-5].

1) Кожне ребро графа має пропускну здатність, яка представляється невід'ємним числом й позначена на рисунку поряд з ребром. Потік більшою величини, ніж зазначено не може протікати через це ребро.
2) Серед вершин графа виділяється вершина в яку потік не заходить, а тільки виходить, вона має назву – джерело. На рис. 1.2 такою вершиною є вершина S.
3) Існує в графі також вершина, з якої потік не виходить, а тільки входить, вона має назву стік. На рис. 1.2 такою вершиною є вершина T.
4) Вузли, відмінні від s і t, будуть називатися внутрішніми вузлами.
5) Для кожного внутрішнього вузла притаманна властивість, що описується математично рівнянням:

$$\sum_{u:(u,v)\in E} f_{uv} = \sum_{u:(v,u)\in E} f_{vu}$$

для $\forall\ v \in$ V/{S, T} (закон збереження потоків: сума потоків, що входять у вузол, повинна дорівнювати сумі потоків, що виходять з вузла, за винятком вузлів джерела і стоку).
6) Важливою концепцією в теорії графів, яка корисна для моделювання пропускної спроможності мережі, є розрізом. Розріз XY - це набір дуг, видалення яких відключає вузол X від вузла Y.

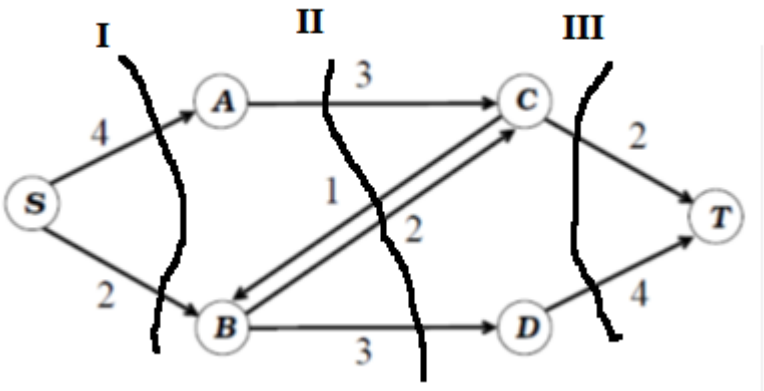

Рисунок 1. 3 – Діаграма мережевого потоку з розрізами I, II, III

Теорема про мінімальний розріз. максимальний потік між будь-якими двома довільними вузлами будь-якого графа не може перевищувати ємність мінімального розрізу, що розділяє ці два вузли.

Річ 2. Отже, потрібно знайти максимальний потік в мережі. Як взятися за вирішення цього завдання? Огляд джерел інформації дає змогу визнати той факт, що такі відомі парадигми програмування, як жадібні алгоритми чи динамічне програмування, не працюють - принаймні для завдання про знаходження максимального потоку не існує алгоритму, який можна було б природно розглядати як такий, що до парадигми динамічного програмування. У відсутності інших ідей можна поміркувати про прості жадібними методами, щоб зрозуміти, чому вони не працюють.

Застосовність алгоритмів жадібної стратегії.

Жадібний підхід до проблеми максимального потоку полягає в тому, щоб починати з потоку з нульовим витратою і жадібно створювати потоки з постійно більш високим значенням. Природний спосіб переходу від одного до наступного полягає в тому, щоб посилати більше потоку по деякому шляху від s до t.

```
E number of edge
f(e) flow of edge
C(e) capacity of edge

1) Initialize : max_flow = 0
                f(e) = 0 for every edge 'e' in E

2) Repeat search for an s-t path P while it exists.
   a) Find if there is a path from s to t using BFS
      or DFS. A path exists if f(e) < C(e) for
      every edge e on the path.
   b) If no path found, return max_flow.
   c) Else find minimum edge value for path P

      // Our flow is limited by least remaining
      // capacity edge on path P.
      (i) flow = min(C(e)- f(e)) for path P ]
             max_flow += flow
      (ii) For all edge e of path increment flow
             f(e) += flow

3) Return max_flow
```

На рис. 1.4 надано просту мережу, яка є контр прикладом для демонстрації тієї тези, що жадібна стратегія в цій задачі не спрацьовує, а надає хибного результату. Очевидно (шляхом простого перебору), що максимальний потік в

ції мережі становить 5, на відміну від 4, що одержуємо за жадібним алгоритмом.

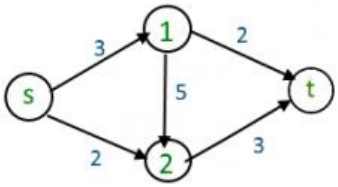

Рисунок 1.4 – Приклад мережі, коли «жадібний» алгоритм не діє

Отже проблема як ми можемо знайти максимальний потік і довести, що він правильний залишається відкритою.

Ось один з підходів – проста парадигма й дуже природна стратегія:

1) Знайдіть шлях від s до t і натисніть на нього якомога більше потоку.
2) Потім подивіться на залишки, що залишилися (важливим питанням буде те, як саме ми це визначимо, але це питання ми розглянемо в розділі 2) і повторіть.
3) Продовжуйте кроки до тих пір, поки не залишиться жодної траси, на якій є можливість просунути додатковий потік.

Звичайно, нам потрібно довести, що це працює: що ми не можемо якось прийти до неоптимального рішення, зробивши поганий вибір на цьому шляху. Такий підхід, при правильному визначенні "ємності, що залишився", називається алгоритмом Форда-Фулкерсона.

1.2 Постановка задачі

Нехай задано зважений граф, в якому ваги представляють собою пропускну спроможність графа, одна з вершин має призначення джерела, інша – стоку.

Поставимо наступне завдання дослідження.

А) *Для заданого зваженого графа розв'язати завдання знаходження найбільшої пропускної спроможності від джерела до стоку:*

Б) Провести порівняльний аналіз алгоритмів Дініца та Форда-Фалкерсона на предмет швидкодії алгоритмів та кількості операцій порівняння та присвоєння при реалізації алгоритмів на ЕОМ.

2 ВИБІР АЛГОРИТМІВ РІШЕННЯ ПРОБЛЕМИ

2.1 Короткий огляд відомих рішень

Повернемось до поставленої проблеми «maximum network flow problem».

Мережа задана зваженим графом й має ряд особливостей та припущень, що викладені в п. 1.1 (див. також рис. 1.2). Наша мета – просунути якомога більше потоку від s до t по заданому графу.

Наприклад, у наведеній раніше мережі (рис. 1.2, п. 1.1), поставимо питання – який максимальний потік від s до t? Відповідь: 5. Використовуючи позначення “ємність [потік]”, рішення проблеми «maximum network flow problem» виглядає так, як представлено на рисунку 2.1. Звернемо увагу, що в цьому рішенні потік може розгалужуватись і знову потім приєднатися.

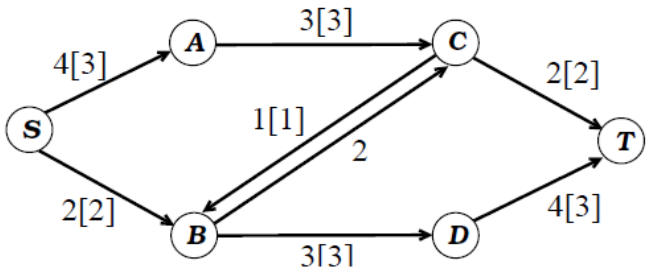

Рисунок 2. 1 – Знаходження максимального мережевого потоку, показаного за допомогою позначення "ємність [потік]"

Опісля рішення задачі виникає ще одне питання – яким чином ми зможемо довести, що вищезнайдений потік відповідає рішенню поставленого завдання, тобто є максимально можливим через задану мережу?

Спершу зауважимо, що знайдений потік насичує ребра a → c і s → b, і, якщо їх вилучити, то тим самим від'єднаємо t від s. Іншими словами, задана мережа має "s-t розріз" розміром 5 (набір пропускної спроможності загальною ємністю 5 таким чином, що якщо ви видалите їх, це від'єднує джерело від стоку). Справа в тому, що будь-яка одиниця потоку, що йде від s до t, повинна займати принаймні 1 одиницю потужності в цій мережі. Отже, ми досягли оптимальності.

Якщо стверджувати в цілому, то ми досягли виконуваності теореми про мінімальний розріз, подану в п.1.1.

Важливою властивістю потоків, яку було підтверджено в процесі пошуку рішення проблеми, є те, що максимальний s-t потік насправді дорівнює потужності мінімального s-t розрізу.

Правила полягають у тому, що жоден зважений граф не може мати потік, що перевищує його ємність, і для будь-якої вершини, крім s і t, потік у вершину повинен дорівнювати потоку з вершини. Тобто обмеження ємності: на будь-якому ребрі e має бути $f(e) \leq c(e)$.

Як зазначено вище, проблема «maximum network flow problem» входить до групи таких проблем, що мають широке коло різноманітних застосувань. Це одна з причин, за якою література з проблем побудови алгоритмів та методів рішення цієї проблеми різноманітна й розкидана по багатьох галузях та сферах свого застосування [1-9].

За останні двадцять років відбувся значний сплеск досліджень, що стосується вдосконалюванню алгоритмів визначення максимальних потоків в мережах. Існує багато наукових внесків щодо покращення часу роботи алгоритмів максимального потоку (поліпшення асимптотичних оцінок алгоритмів) за допомогою вдосконалених структур даних, методів масштабування проблемних даних тощо. В основному це роботи в англомовних виданнях [3-9].

Проаналізувавши більшість алгоритмів знайдення максимального потоку, розроблені дотепер, то можна виділити два глобальних напрями:

1. алгоритми нарощування шляху;

2. алгоритми передпотоку.

Зауважимо, що ці напрями зберігаються з часів перших робіт в постановці сучасного розуміння проблеми «maximum network flow problem» та її рішення – це дослідження Л. Р. Форда-молодшого та Фалкерсона Д. Р., що проводились в середині 1950-х років (хоча іноді під словосполученням алгоритм Форда-Фалкерсона розуміють алгоритм Едмондса-Карпа, який є точною реалізацією методів Форда та Фалкерсона) до сучасних алгоритмів. Цікавими в історичному плані винайдення алгоритмів розв’язання проблеми «maximum network flow problem» є порівняння викладені в роботі [7] (див. табл. 2.1).

Таблиця 2.1 Порівняльна таблиця алгоритмів рішення «maximum network flow problem» (англомовний варіант)

year	authors	complexity
1955	Ford-Fulkerson [19]	$O(mnU)$
1970	Dinic [15]	$O(mn^2)$
1970	Edmonds-Karp [17]	$O(m^2n)$
1972	Dinic [15], Edmonds-Karp [17]	$O(m^2 \log U)$
1973	Dinic [16], Gabow [20]	$O(mn \log U)$
1974	Karzanov [37]	$O(n^3)$
1977	Cherkassky [11]	$O(n^2m^{1/2})$
1980	Galil-Naamad [21]	$O(mn(\log n)^2)$
1983	Sleator-Tarjan [44]	$O(mn \log n)$
1986	Goldberg-Tarjan [26]	$O(mn \log(n^2/m))$
1987	Ahuja-Orlin [3]	$O(mn + n^2 \log U)$
1987	Ahuja-Orlin-Tarjan [4]	$O(mn \log(2 + n\sqrt{\log U}/m))$
1990	Cheriyan-Hagerup-Mehlhorn [9]	$O(n^3/\log n)$
1990	Alon [5]	$O(mn + n^{8/3} \log n)$
1992	King-Rao-Tarjan [38]	$O(mn + n^{2+\epsilon})$
1993	Phillips-Westbrook [42]	$O(mn \log_{m/n} n + n^2(\log n)^{2+\epsilon})$
1994	King-Rao-Tarjan [39]	$O(mn \log_{m/(n \log n)} n)$
1997	Goldberg-Rao [23]	$O(\min\{m^{1/2}, n^{2/3}\}m \log(n^2/m) \log U)$

Трохи іншого вигляду має порівняльна таблиця в російськомовному варіанті, але основні дати та асимптотичні оцінки співпадають (див. табл. 2. 3).

В тому та іншому варіанті введені позначення – n - число вершин графа, m - число ребер, U - найбільша величина максимальної пропускної здатності мережі.

Додаємо до списку ті алгоритми, що в нього не ввійшли, бо список складався до їх винайдення.

Таблиця 2.2 Новітні алгоритми рішення «maximum network flow problem»

Рік	Автори	Асимптотична оцінка
2010	Келнер - Мондр - Спілман - Тен	$O(nm^{1/3}\varepsilon^{-11/3}\log^c(nm^{1/3}\varepsilon^{-11/3}))$
2012	Дж. Орлін 1	O(nm)
2012	Дж. Орлін 2	$O(n^2/\log n)$

Таблиця 2.3 Порівняльна таблиця алгоритмів рішення «maximum network flow problem» (російськомовний варіант)

Хронологическая таблица достижений решения задачи о максимальном потоке

Год	Автор	Оценка алгоритма
1951	Данциг	$O(n^2mU)$
1956	Форд , Фалкерсон	$O(nm^2 \cdot U)$
1970	Едмондс, Карп	$O(nm^2)$
1970	Диниц	$O(n^2m)$
1972	Эдмондс, Карп	$O(m^2 \cdot \log U)$
1973	Диниц, Габоу	$O(nm \cdot \log U)$
1974	Карзанов	$O(n^3)$
1978	Малхотри, Кумар, Махешвари	$O(n^3)$
1978	Галил, Намад	$O(nm \cdot \log^2 n)$
1980	Слейтор, Тарьяян	$O(nm \cdot \log n)$
1986	Голдберг, Тарьян	$O(nm \cdot \log \frac{n^2}{m})$
1989	Чериян, Хагеруп, Мелхорн	$O(nm + n^2 \log^2 n)$
1992	Кинг, Рао, Тарьян	$O(nm + n^{2+\varepsilon})$
1997	Голдберг, Рао	$O(\min\{m^{\frac{2}{3}}, n^{\frac{1}{2}}\} m \cdot \log(\frac{n^2}{m}) \log U)$

Модель мережевого потоку з'єднує кілька різноманітних і, здавалося б, не пов'язаних між собою областей комбінаторної оптимізації.

Намітились та вже використовуються новітні шляхи розвитку алгоритмів по рішенню проблеми «maximum network flow problem» [5-9]:

- Спроба оптимізації та підвищення швидкодії існуючих методів.
- Розробка нових методів.
- Нарощування обчислювальної потужності, використання паралельних обчислень, кластерні, розподілені, хмарні обчислення.

У планарній мережі виникають також практичні контексти, такі як V [S] дизайнерські та комунікаційні мережі, і тому цікаво знайти алгоритми швидкого потоку для цього класу графіків.

Проблема максимального потоку в плоских мережах привернула значну увагу дослідників. В останні кілька десятиліть інтенсивно вивчалися проблеми зворотної комбінаторної оптимізації. Для подібного роду проблем ідея полягає в тому, щоб змінити вектор параметрів (потужностей, витрат), таким чином,

щоб задане можливе рішення задачі прямої оптимізації стало оптимальним рішенням, а відстань між початковим вектором та модифікованим вектором параметрів мінімальною. .

Модифікація ємності проводиться для максимального потоку та мінімального розрізання. Алгоритми сильно поліноміального часу для вирішення задачі оберненого максимального потоку (IMF) за нормою були представлені [6]. Для рішення оберненої проблема максимального потоку пропонують сильно поліноміальні алгоритми для МВФ на відстані Хеммінга.

Перш ніж перейти до алгоритмів, що будуть тестовано в даній роботі коротко опишемо алгоритм Нагамочі-Ібаракі , який обчислює сертифікат розрідженої сполученості непрямого підключеного графа G = (V, E) за 0 (м) часу на основі своєрідний пошук на широті, який має пріоритет.

Nagamochi-Ibaraki algorithm for finding a sparse connectivity certificate

1. All the edges $e \in E$ are initialized to be unscanned.
   Set $r[v] := 0$ for each $v \in V$ and *insert* v into priority queue Q as key $r[v]$.
2. **while** Q is not empty **do**
   (a) $v :=$ *deletemax* (Delete vertex v of maximum $r[v]$ from Q).
   (b) **for** each unscanned edge $e = (v, u)$ incident to v **do** (e becomes scanned).
   Set $t[e] := r[u] + 1$; $r[u] := r[u] + 1$ (*increasekey* of u by one in Q).
   {**comment:** If $r[v] < r[u]$ (i.e., $r[u] = r[v] + 1$) then set $r[v] := r[u]$.}

Дійшла мода застосувань нейронних мереж та наближених методів, я то генетично-еволюційні алгоритми й до рішення «maximum network flow problem».

Пошук рішення заснований на виборі з деякою «популяції» найбільш вдалих «особин» і отриманні на їх основі такої «популяції». Критерій оптимальності може використовуватися будь-хто. В основі використання цього класу методів лежить так звана «теорема шим» (scheme), з докази якої випливає, що при певних умовах генетичні алгоритми дають експоненціально швидку збіжність рішення до локально-оптимального. Застосування нейронних мереж можна знайти, наприклад, в [9].

Застосування паралельної версії алгоритму пошуку максимального потоку дає змогу скорочення часу роботи програми [8], наприклад, на тестовій мережі показано значне скорочення часу виконання алгоритму.

2.2 Алгоритм Форда-Фалкерсона

Алгоритм Форда-Фалкерсона для рішення проблеми «maximum network flow problem» має в своїй реалізації ітеративну парадигму й базується в своєму виконанні на трьох «китах»: А) залишкові мережі; Б) шляхи що збільшуються; і В) розрізи.

Ключову роль у методі Форда-Фалкерсона грають два поняття: залишкові мережі і доповнюючи шляхи. Дані концепції лежать в основі важливої теореми про максимальний потік і мінімальний розріз, яка визначає значення максимального потоку за допомогою розрізів траспортної мережі.

Алгоритм Форда-Фалкерсона є ітеративним.

1.Спочатку величині потоку присвоюється значення 0: f(u,v)=0 при будь-яких u, v Є V.

2. На кожній ітерації величина потоку збільшується за допомогою пошуку «шляху, що збільшується» (тобто деякого шляху від джерела s до стоку t, уздовж якого можна послати більший потік) і подальшого збільшення потоку.

3. Цей процес повторюється до тих пір, поки вже неможливо буде відшукати шляхів для збільшення потоку.

Логічна інтуїція , що лежить в основі цього методу, проста: знайдіть шлях (що збільшує потік) невикористаної ємності і збільште потік уздовж цього шляху. Повторюйте, поки такі шляхи не знайдені.

Те, що робить це нетривіальним, є очевидним парадоксом: загальний потік іноді може бути збільшений шляхом зменшення потоку уздовж певних ребер (тому що вони течуть в «неправильному» напрямку або переміщують ємність в ту частину мережі, яка також не може впоратися з цим).

Ford Fulkerson управляє цим шляхом побудови паралельної мережі з наявних або залишкових спроможностей. Ми повернемося до методу після пояснення цих понять.

Залишкова мережа. Залишкової пропускною спроможністю (англ. residual capacity) ребра (*u, v*) називається величина додаткового потоку, який ми

можемо направити з *u* в *v*, що не перевищивши пропускну спроможність *c (u, v)*. Іншими словами

$$c_f(u, v) = c(u, v) - f(u, v). \qquad (2.1)$$

Для заданої транспортної мережі *G = (V, E)* і потоку *f*, залишкової мережею, (яка доповнює мережу, англ. residual network) в G, породженої потоком f, є мережа

$$G_f = (V, E_f), \text{ де } E_f = \{(u, v) \in V \times V | c_f(u, v) > 0\} \qquad (2.2)$$

Для заданої транспортної мережі *G = (V, E)* і потоку *f* доповнює шляхом (англ. augmenting path) *p* є шлях з джерела до стоку в залишкової мережі $G_f = (V, E_f)$.

Наприклад,

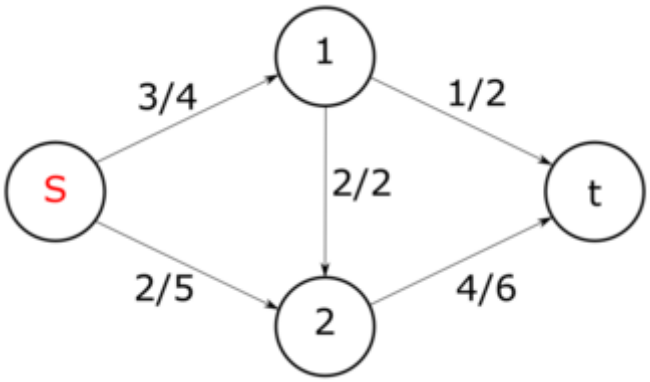

Рисунок 2. 2 – Граф з неповним потоком

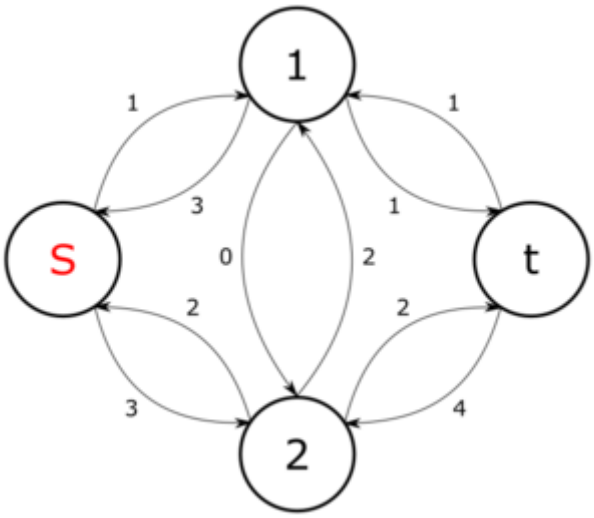

Рисунок 2. 3 – Залишкова мережа графа 2.2

Псевдокод алгоритму Форда- Фалкерсона.

Algorithm Ford–Fulkerson

Inputs Given a Network G(V, E) with flow capacity *c*, a source node *s*, and a sink node *t*
Output Compute a flow *f* from *s* to *t* of maximum value

for all edges

While there is a path *p* from *s* to *t* in , such that for all :

Find

For each edge

(*Send flow along the path*)

(*The flow might be "returned" later*)

Покрокове виконання комп'ютерної реалізації алгоритму Форда-Фалкерсона (див. ДОДАТОК А).

Ініціалізація.

Для всіх ребер(i,j) покладемо залишкову пропускну спроможність рівній початковій пропускній спроможності тобто(c_{ij}, c_{ji}) = (C_{ij}, C_{ji})

Позначимо $a_1 = \infty$ і помітимо вузол 1 міткою [∞, -].

Крок 1

Беремо i = 1 і переходимо до другого кроку.

Крок 2.

Визначаємо множину S_i як множину вузлів, в які можна перейти з i, по ребру з додатною залишковою пропускною спроможністю. При цьому вузол j має бути не помічений. Якщо отримуємо не порожню множину переходимо до кроку 3, інакше до кроку 4.

Крок 3.

У множині S_i знаходимо вузол k такий, що $c_{ik} = \max\{c_{ij}\}$. Приймаємо $a_k = c_{ik}$ і помічаємо вузол k міткою [a_k, i]. Якщо останньою міткою є стік, то розріз-

ний шлях знайдено і переходимо до кроку 5, інакше приймаємо I = k і переходимо до кроку 2.

Крок 4(відкат назад).

Якщо i = 1, то переходимо до кроку 6. Інакше знаходимо вузол r що стоїть перед вузлом i, видаляємо вузол i із множини вузлів, приймаємо i = r і переходимо до кроку 2.

Крок 5.

Максимальний потік зо проходить по знайденому шляху розраховується як:

$$f_p = \min\{a_1, a_{k1}, \dots, a_{kn}\}$$

Підраховуємо залишкові пропускні спроможності ребер відповідно до формул:

$(c_{ij} - f_p, c_{ji} + f_p)$, якщо потік іде від вузла i до j

$(c_{ij} + f_p, c_{ji} - f_p)$, якщо потік іде від вузла j до i

Крок 6.

Видаляємо усі мітки крім з витоку та переходимо на перший крок пошуку.

Аналіз алгоритму Форда-Фалкерсона.

На кожній ітерації відбувається:

1. Пошук розрізного шляху.
2. Підрахунок пропускної спроможності шляху.
3. Підрахунок залишкових спроможностей ребер.

Максимальний потік буде знайдено не більш ніж за O(F) ітерацій, де F – максимальний потік у мережі.

Кожна ітерація вимагає O(E) часу, де E – число ребер у мережі.

Отже загальний час роботи алгоритму: O(E*F).

2.3 Алгоритм Дініца

Алгоритм Дініца відносять в теорії складності алгоритмів до поліноміальних алгоритмів обчислення максимального потоку в транспортній мережі.

Цей алгоритм, опублікований в 1970 р, мав величезне значення для подальшого пошуку удосконалень в плані поліпшення асимптотичних оцінок. На той час всі покращання у вирішенні задачі про максимальний потік були засновані на алгоритмі Дініца [1].

Алгоритм Дініца в своїй реалізації теж має ітеративну парадигму й базується в своєму виконанні на трьох «китах»: А) залишкові мережі; Б) багатошарові мережі; і В) блокуючий потік.

Визначення багатошарової мережі.

Для початку визначимо для кожної вершини v даної мережі G довжину найкоротшого s⇝v шляху з джерела і позначимо її d [v] (для цього можна скористатися наприклад, одним із обходів графа).

Тоді у багатошарову мережу будуть включатися лише ті ребра (u, v) вихідної мережі, для яких d [u] + 1 = d [v]. Отримана мережа ациклічна, і будь-який s⇝t шлях в багатошаровій мережі є найкоротшим шляхом у вихідній, з властивостей обходу графа. На рис. 2. 4 представлено приклад багатошарової мережі.

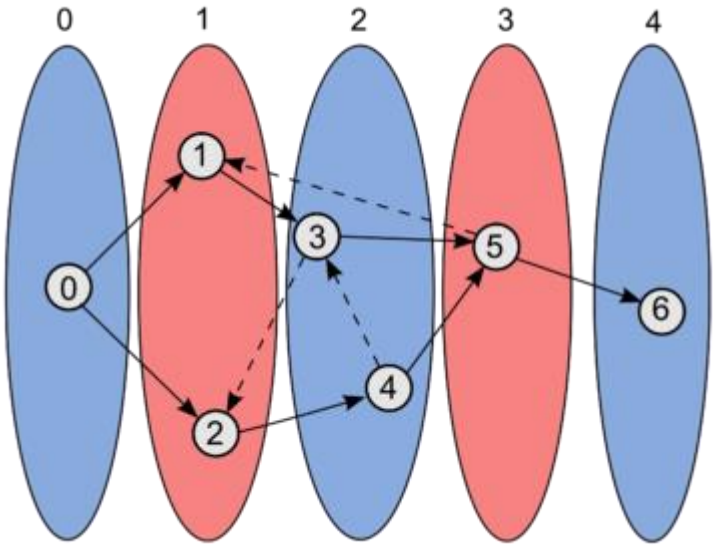

Рисунок 2.4 – Багатошарова мережа з S=0, t=6

Алгоритм складається з декількох ітерацій. На кожній ітерації будується залишкова мережу, потім на основі залишкової мережі будується багатошарова мережа, а в багатошаровій мережі знаходиться блокуючий потік. Знайдений блокуючий потік використовують для збільшення шуканого результату.

Основна схема алгоритму Дініца.

1. Для кожного ребра (u, v) даної мережі G поставимо f (u, v) = 0.
2. Побудуємо багатошарову мережу G_L з залишкової мережі G_f даного графа G. Якщо d [t] = ∞, зупинитися і вивести f.
3. Знайдемо блокуючий потік f 'в GL.
4. Доповнимо потік f знайденим потоком f 'і перейдемо до кроку 2.
5. Ітерації повторюються, поки можливо знайти блокуючих потік.

Основна ідея методу - алгоритм складається з фаз, на яких потік збільшується відразу уздовж всіх найкоротших ланцюгів певної довжини. Для цього на і-тій фазі будується багатошарова мережа (layered network). Ця мережа містить всі збільшуючи ланцюги, довжина яких не перевищує і, де і - довжина найкоротшого шляху з джерела в стік. Величина і це довжина допоміжної мережі.

Покажемо, що якщо алгоритм завершується, то на виході ми отримаємо потік саме максимальної величини.

Насправді, припустимо, що в і-й момент в багатошаровій мережі, побудованої для залишкової мережі, не вдалося знайти блокуючий потік. Це означає, що до стоку не можна взагалі потрапити в допоміжній мережі з джерела. Але оскільки багатошарова мережа містить в собі всі найкоротші шляхи з джерела в залишкової мережі, це в свою чергу означає, що не існує шляху з джерела до стоку. Отже, застосовуючи теорему Форда-Фалкерсона, отримуємо, що поточний потік справді максимальний. Отже, ми отримали саме максимальний потік в даній мережі.

Покрокове виконання комп'ютерної реалізації алгоритму Дініца (див. ДОДАТОК Б).

Крок 1

Будуємо залишкову мережу для заданого графа.

Крок 2

Поки в мережі є шлях із s в t, виконати наступні кроки:

- Знаходимо найкоротший шлях з s в t. Якщо його немає, вийти з циклу.

- На знайденому шляху в залишковій мережі знаходимо ребро з мінімальною пропускною спроможністю C_{min}
- Для кожного ребра на знайденому шляху збільшуємо потік C_{min} , а в протилежному йому — зменшуємо на C_{min}
- Видаляємо всі ребра, які досягли насичення.
- Видаляємо всі глухі кути (тобто вершини, крім стоку, звідки не виходить ребер, і вершини, крім джерела, куди ребер не входить) разом з усіма інцидентними їм ребрами.
- Повторюємо попередній крок, поки є що видаляти.

Крок 3

Якщо знайдений потік ненульовий, додаємо його до загального потоку і повертаємося на крок 1.

Аналіз алгоритму Дініца.

Можна показати [1], що щоразу кількість ребер у блокуючому потоці збільшується принаймні на одне, тому в алгоритмі не більше n-1 блокуючих потоків, де n — кількість вершин у мережі. Допоміжна мережа G_L може бути побудована обходом у ширину за час O(E), а блокуючий потік на кожному рівні графа може бути знайдений за час O(VE). Тому час роботи алгоритму Дініца дорівнює $O(V) * (O(E) + O(VE)) = O(VE^2)$.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ТЕСТОВІ РОЗРАХУНКИ

3.1 Комп’ютерна реалізація алгоритмів та основні її складові

Комп’ютерна реалізація обраних алгоритмів проводилися на комп'ютері з наступною конфігурацією:

- ✓ ПРОЦЕССОР Intel (R) Celeron CPU 3050 @ 1.6 GHz;
- ✓ ОПЕРАТИВНА ПАМ'ЯТЬ (ОРЕ) 2.0 ГБ;
- ✓ ОПЕРАЦІЙНА СИСТЕМА Windows 7;
- ✓ Тип системи 32-розрядна операційна система.

Вхідна транспортна мережа задавалася графом G = (V, E), в якому V – множина вершин, E – множина ребер, з заданими вартостями c_{ij} на кожному ребрі (i, j), що вказують на його пропускну спроможність. Вершина з номером 1 обиралась за джерело мережі, з останнім номером в множині вершин – за стік.

Потрібно знайти максимальний потік, який можна пропустити через задану мережу від джерела до стоку.

Математично це означає, що потрібно знайти такий шлях P між вершинами 1 та n P(v_1, v_2, …,v_n), щоб

$$\sum_{i=1}^{n} \pi_{ij} \rightarrow \max \qquad (3.1)$$

де π_{ij} – потік між вершинами i та j.

Для рішення проблеми «maximum network flow problem» та проведення комп’ютерного порівняльного аналізу швидкодії алгоритмів Форда-Фалкерсона та Дініца було обрано алгоритмічну мову програмування C ++ . Вибір мови був зумовлений тим, що вона являє собою сучасну мову програмування високого рівня, що використовує об'єктно-орієнтований підхід.

Для задання вхідної інформації створена структура edge:

```
struct edge {
    int a, b, f, c;
    int ind;
};
```

А також вектор vector <edge> е за допомогою якого ребра можна задати списком й вказати їх пропускні спроможності: Задаються також кількості вершин та ребер - int n, m;

Наприклад, нижче задається граф, що має 5 вершин та 8 ребер:

5 8

1 2 2

1 3 3

1 4 1

2 3 4

2 5 3

3 5 2

4 5 2

3 4 1

Для дослідження швидкодії алгоритму було реалізовано консольний додаток на мові C ++ в середовищі розробки Microsoft Visual Studio 2010. Консольний додаток надає змогу заміряти процесорний час роботи алгоритму.

Окрім часу виконання алгоритмів для порівняльного аналізу ми будемо знаходити дві порівняльні характеристики: кількість операцій порівнянь в досліджуваних алгоритмах, та кількість операцій присвоєнь, які на початку дії алгоритмів обнуляються:

```
unsigned int start_time;
unsigned int end_time;
int prisf = 0;
int porivn = 0;
```

В кінці роботи алгоритмів порівняльні характеристики ми виводимо на екран

```
   unsigned int end_time = clock();
        unsigned int search_time = end_time - start_time;
   cout << maxFlow << endl <<"Час:"<< search_time<< endl <<"Порівняння:"<< porivn<< endl <<"Пересилання:"<< prisf;
```

return 0;

Результати розрахунків виводяться на екран в такому вигляді

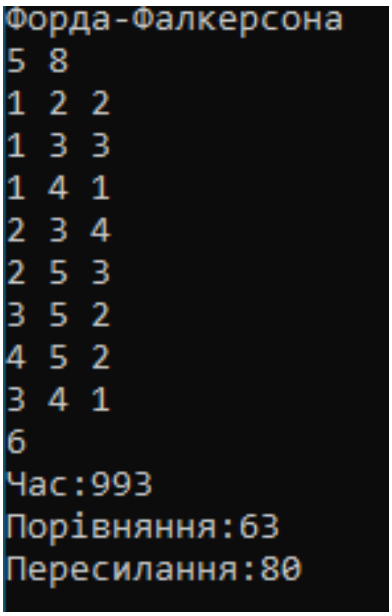

Рисунок 3. 1 – Розрахунки за алгоритмом Форда-Фалкерсона

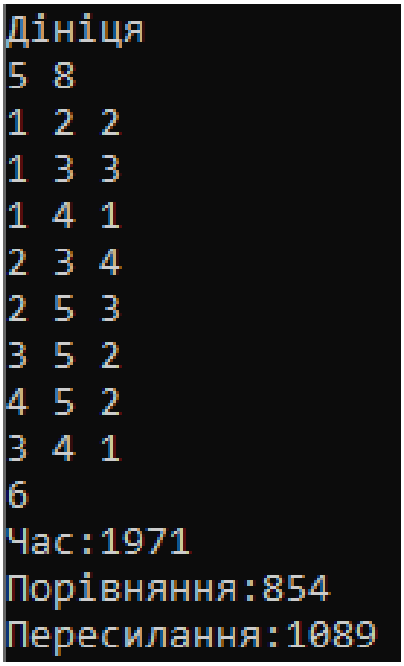

Рисунок 3.2 – Розрахунки за алгоритмом Дініца

3.2 Тестові розрахунки та порівняльний аналіз

Для порівняльного аналізу алгоритмів між собою проводились тестові розрахунки на мережах різної конфігурації (задавались різні кількості вершин та ребер графа), при цьому пропускні спроможності обирались випадковим чином. Тим самим асимптотичні теоретичні оцінки часу виконання алгоритмів перевірялись на конкретних прикладах.

Тест 3.1. Мережа, що має 5 вершин на 8 ребер.

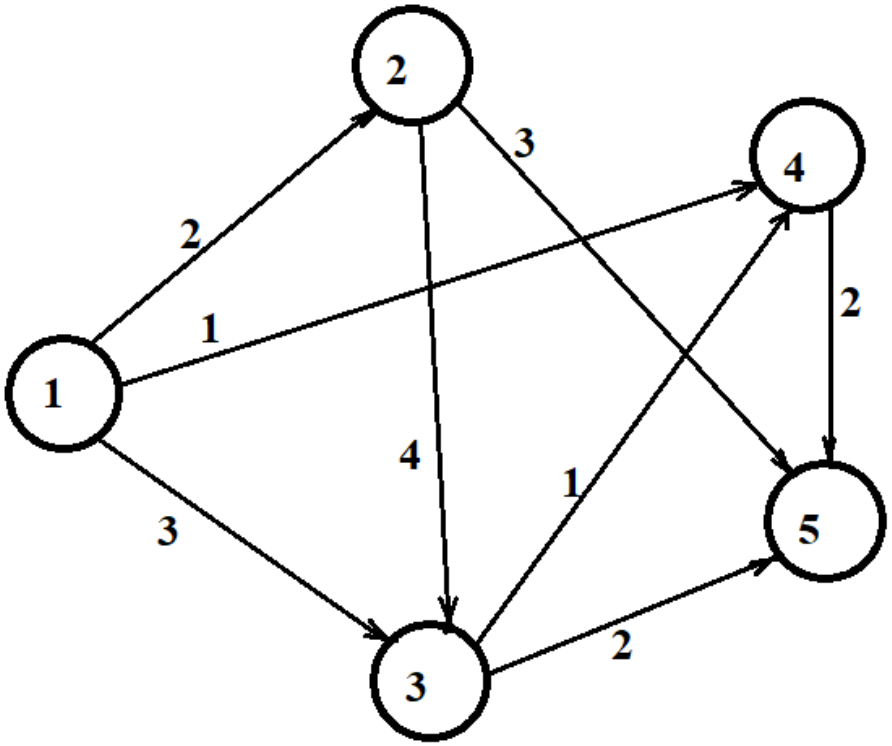

Рисунок 3.3 – Мережа 3.1 для тестових розрахунків

Вхідні данні:

5 8
1 2 2
1 3 3
1 4 1
2 3 4
2 5 3
3 5 2
4 5 2
3 4 1

Не важко для мережі 3.1 розрахувати максимальний потік від джерела 1 до стоку 5 шляхом перебору. Це 6 умовних одиниць. Надаємо комп’ютерні розрахунки алгоритмів в таблиці 3.1 та 3.2

Таблиця 3.1 Розрахунки мережі 3.1 за алгоритмом Форда-Фалкерсона

Час 1 тесту	1277
Час 2 тесту	1958
Час 3 тесту	1141
Середній час	1458,67
Порівнянь	63
Пересилань	80

Максимальний потік: 6

Таблиця 3.2 Розрахунки мережі 3.1 за алгоритмом Дініца

Час 1 тесту	416
Час 2 тесту	1087
Час 3 тесту	849
Середній час	784
Порівнянь	854
Пересилань	1089

Максимальний потік: 6

Неважко помітити що обидва алгоритми знайшли однаковий максимальний потік 6, що відповідає дійсності. Цікавим є той факт, що за часом виконання алгоритм Дініца має перевагу, а за кількістю операцій переважає алгоритм Форда –Фалкерсона.

Графічно це представлено гістограмою (див. рис. 3.4)

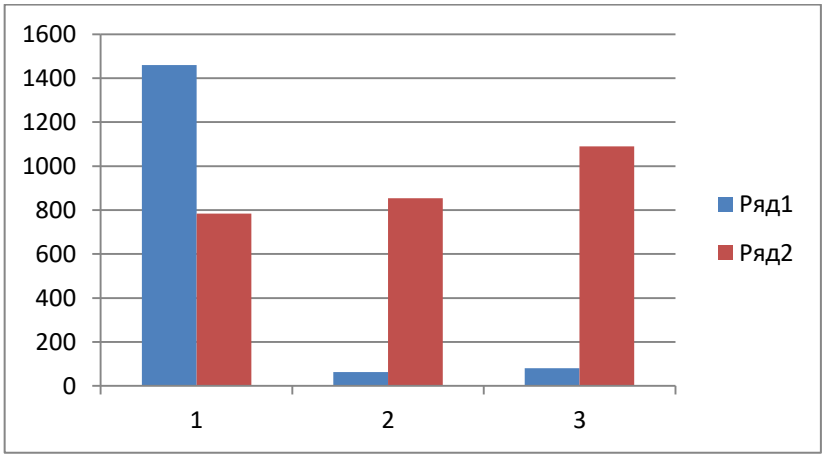

Рисунок 3.4 – Гістограма порівняльного аналізу розрахунків мережі 3.1

Тест 3.2. Збільшена в 10 разів пропускна спроможність ребер в порівнянні зі тестом 3. 1.

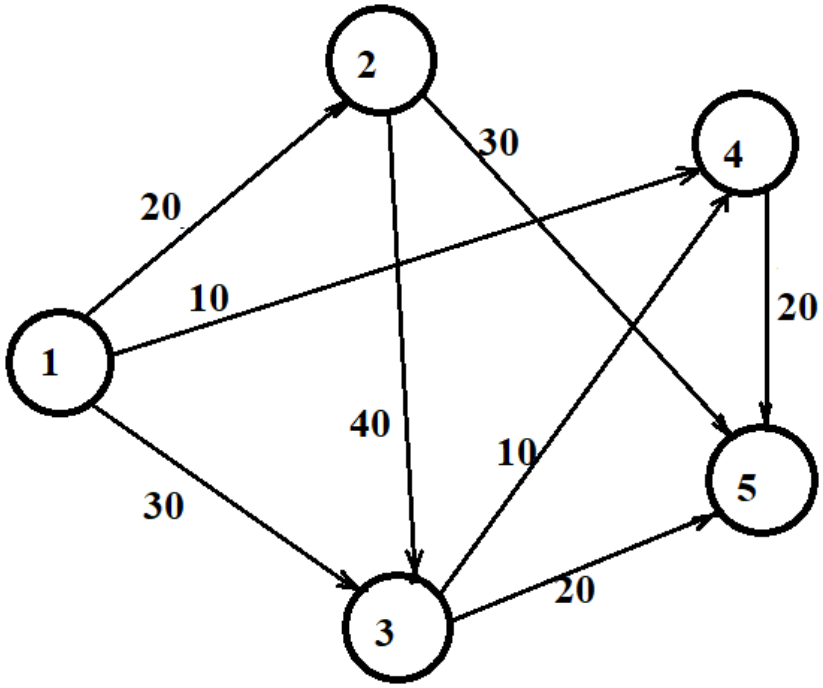

Рисунок 3. 5 – Мережа 3.2 для тестових розрахунків

Вхідні данні:
5 8
1 2 2
1 3 3
1 4 1
2 3 4
2 5 3
3 5 2
4 5 2
3 4 1

Таблиця 3.3 Розрахунки мережі 3.2 за алгоритмом Форда-Фалкерсона

Час 1 тесту	2258
Час 2 тесту	2380
Час 3 тесту	1247
Середній час	1961,67
Порівнянь	63
Пересилань	80

Максимальний потік: 60

Таблиця 3.4 Розрахунки мережі 3.2 за алгоритмом Дініца

Час 1 тесту	743
Час 2 тесту	624
Час 3 тесту	989
Середній час	785,33
Порівнянь	1150
Пересилань	1371

Максимальний потік: 60

Неважко помітити, що результат максимального потоку зріс теж в десять разів в порівнянні з тестом 3.1, що знову говорить про правильність розрахунків (тому що кожну пропускну спроможність ми теж збільшили в 10 разів). Цікавим є той факт, що за часом виконання алгоритм Дініца знову дістав перемогу, але за кількістю операцій алгоритм Форда –Фалкерсона залишився на тому ж рівні в порівнянні з тестом 3.1., а ось кількість операцій в алгоритмі Дініца значно зросла.

Графічно це представлено гістограмою (див. рис. 3.6).

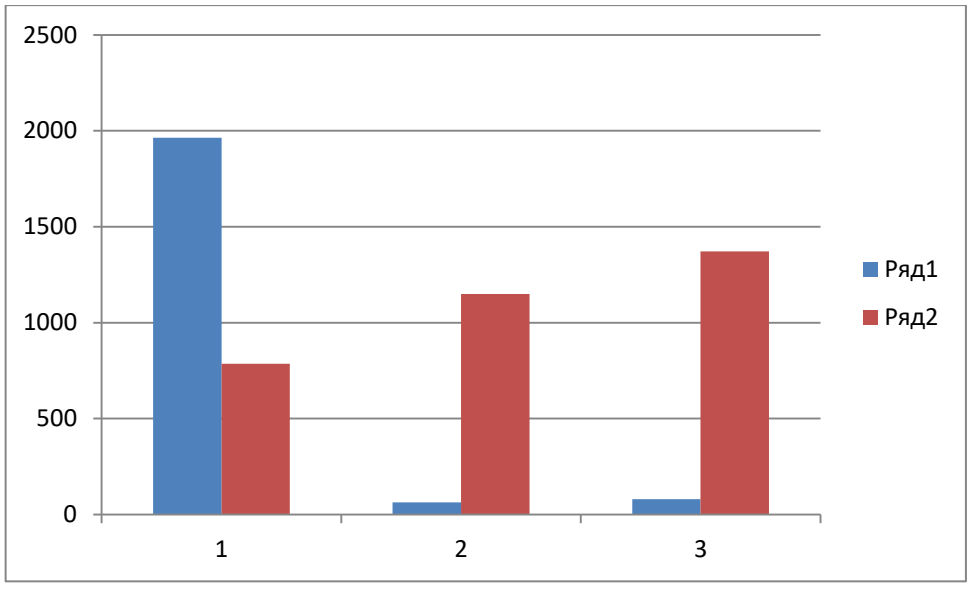

Рисунок 3.6 – Гістограма порівняльного аналізу розрахунків мережі 3.2

Тест 3.3. Мережа, що має 10 вершин та 19 ребер.

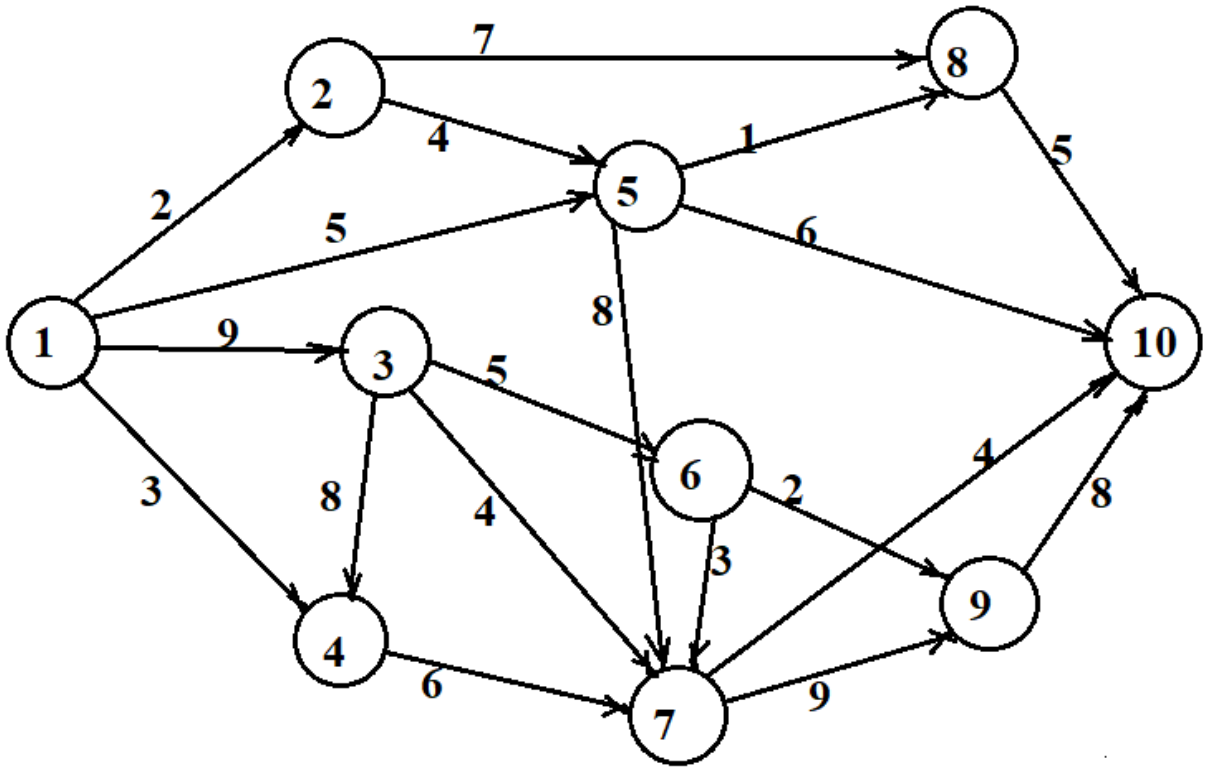

Рисунок 3. 7 – Мережа 3.3 для тестових розрахунків

Вхідні данні:
10 19
1 2 2
1 3 9
1 4 3
1 5 5
2 5 4
2 8 7
3 4 8
3 6 5
3 7 4
4 7 6
5 7 8
5 8 1
5 10 6
6 7 3
6 9 2
7 9 9
7 10 4
8 10 5
9 10 8

Таблиця 3.5 Розрахунки мережі 3.3 за алгоритмом Форда-Фалкерсона

Час 1 тесту	2020
Час 2 тесту	1657
Час 3 тесту	2312
Середній час	1996,33
Порівнянь	155
Пересилань	195

Максимальний потік: 19

Таблиця 3.6 Розрахунки мережі 3.3 за алгоритмом Дініца

Час 1 тесту	1105
Час 2 тесту	1458
Час 3 тесту	1238
Середній час	1267
Порівнянь	2496
Пересилань	2834

Максимальний потік: 19

Гістограму порівняльного аналізу надано нижче.

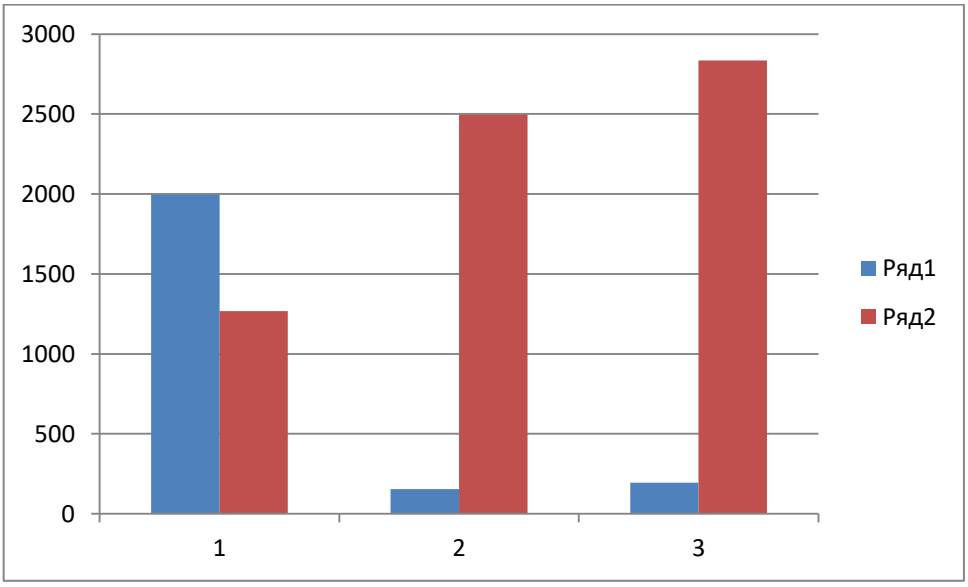

Рисунок 3.8 – Гістограма порівняльного аналізу розрахунків мережі 3.3

За підрахунками помічаємо таку тенденцію – за часом виконання алгоритми майже зрівнялись (мають один й той же порядок), а за кількістю операцій – алгоритм Дініца має на порядок операцій більше ніж Форда-Фалкерсона.

Тест 3.4. Мережа, що має в порівнянні з 3.3 в 10 разів збільшену пропускну спроможність ребер в порівнянні з тестом 3.3

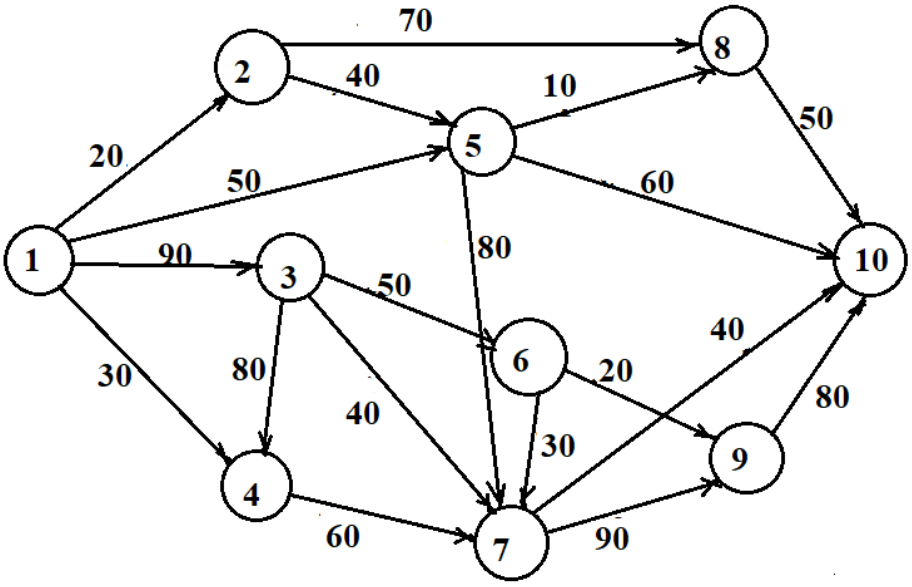

Рисунок 3. 9 – Мережа 3.4 для тестових розрахунків

Вхідні данні:

10 19
1 2 20
1 3 90
1 4 30
1 5 50
2 5 40
2 8 70
3 4 80
3 6 50
3 7 40
4 7 60
5 7 80
5 8 10
5 10 60
6 7 30
6 9 20
7 9 90
7 10 40
8 10 50
9 10 80

Таблиця 3.7 Розрахунки мережі 3.4 за алгоритмом Форда-Фалкерсона

Час 1 тесту	3123
Час 2 тесту	2837
Час 3 тесту	3012
Середній час	2990,67
Порівнянь	155
Пересилань	195

Максимальний потік: 190

Таблиця 3.6 Розрахунки мережі 3.4 за алгоритмом Дініца

Час 1 тесту	1105
Час 2 тесту	1458
Час 3 тесту	1238
Середній час	1267
Порівнянь	3543
Пересилань	3811

Максимальний потік: 190

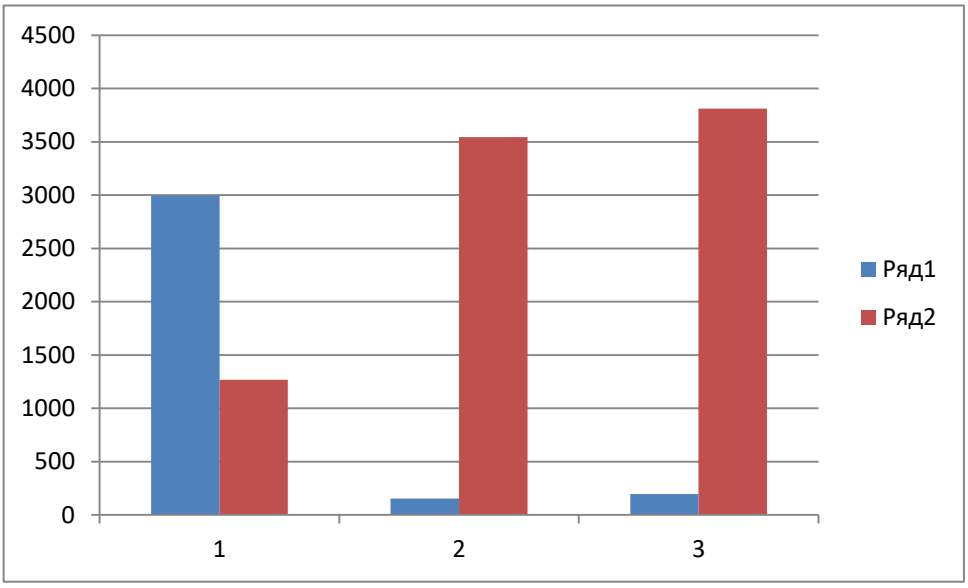

Рисунок 3.10 – Гістограма порівняльного аналізу розрахунків мережі 3.3

ВИСНОВКИ

В випускній роботі проведені дослідження по застосуванні алгоритмів пошуку максимальних потоків в мережах будь якої направленості. Для порівняльного аналізу швидкодії таких алгоритмів між собою було обрано алгоритми Форда-Фалкерсона та алгоритм Дініца. Визначалися час роботи алгоритмів та кількісні оцінки операцій, що відбувалися в реалізації алгоритмів.

За результатами досліджень можна зробити наступні висновки.

1. Для незначної кількості вершин у графі обидва алгоритми по швидкодії практично співпадають.

2. Зі значному збільшенні вершин графа алгоритм Форда-Фалкерсона має переваги над алгоритмом Дініца за кількістю операцій при реалізації.

3. За часом виконання алгоритм Дініца завжди має кращі результати:

СПИСОК ЛІТЕРАТУРИ

1. Paul E. Black. Dictionary of Algorithms and Data Structures [Електронний ресурс] – Режим доступу до ресурсу: https://xlinux.nist.gov/dads/
2. Мелешко Є.В., Якименко М.С., Поліщук Л.І. Алгоритми та структури даних: Навчальний посібник для студентів технічних спеціальностей денної та заочної форми навчання. – Кропивницький: Видавець – Лисенко В.Ф., 2019. – 156 с.
3. Steven S. Skiena The Algorithm Design Manual. Second Edition. - Springer, 2012. – 730 p.
4. Faruque Ahmed, Md. Al-Amin Khan, Aminur Rahman Khan, Syed Sabbir Ahmed and Md. Sharif Uddin An Efficient Algorithm for Finding Maximum Flow in a Network-Flow// Journal of Physical Sciences, Vol. 19, 2014, 41-50.
5. Gharehbolagh H., Hafezalkotob A., Makui A., Raissi S. Cooperative Strategies for Maximum-Flow Problem in Uncertain Decentralized Systems Using Reliability Analysis// Mathematical Problems in Engineering Volume 2016. – p. 1-9.
6. Asano T., Asano Y. Recent developments in maximum flow algorithms // Journal of the Operations Research Society of Japan, Vol. **43,** No. 1, 2000. – p. 2-31.
7. Hochbaum D., S. . Lecture Notes for IEOR 266: Graph Algorithms and Network Flows [Електронний ресурс] – Режим доступу до ресурсу: https://hochbaum.ieor.berkeley.edu/files/ieor266-2014.pdf
8. Ciupala L., Deaconu A. Inverse maximum flow problem in planar networks// Buletin of the Transilvania University of Brasol. Series III: Mathematics, Informatics, Physics, vol12(61`), № 1, 2019. – p. 113-122.
9. Alireza N., Farahnaz O. A capable neural network model for solving the maximum flow problem// Journal of Computational & Applied Mathematics, vol. 236, 2012. - p. 3498-3513.

ДОДАТОК А

Алгоритм Форда-Фалкерсона

```
#include <iostream>
#include <ctime>
using namespace std;
const int MAX_E = (int) 1e6;
const int MAX_V = (int) 1e3;
const int INF   = (int) 1e9;
unsigned int start_time;
unsigned int end_time;
int prisf = 0;
int porivn = 0;
int numOfVertex, numOfEdge, sourceVertex, destinationVertex;
int capacity[MAX_E], onEnd[MAX_E], nextEdge[MAX_E], edgeCount;
int firstEdge[MAX_V], visited[MAX_V];
void addEdge(int u, int v, int cap) {
    onEnd[edgeCount] = v;
      firstEdge[u] = edgeCount;
    capacity[edgeCount++] = cap;
    onEnd[edgeCount] = u;
     nextEdge[edgeCount] = firstEdge[v];
    firstEdge[v] = edgeCount;
    capacity[edgeCount++] = 0;
}
int findFlow(int u, int flow) {
          unsigned int start_time = clock();
```

```
    if (u == destinationVertex) {
        porivn++;
            return flow;
        }
    visited[u] = true;
    for (int edge = firstEdge[u]; edge != -1; edge = nextEdge[edge]) {
      int to = onEnd[edge];
      prisf+=2;
      porivn++;
      if (!visited[to] && capacity[edge] > 0) {
        porivn+=3;
         int minResult = findFlow(to, min(flow, capacity[edge]));
         prisf+=2;
         if (minResult > 0) {
              porivn++;
            capacity[edge]-= minResult;
            capacity[edge ^ 1]+= minResult;
            prisf+=2;
            return minResult;
         }
      }
    }
    prisf+=2;
    return 0;
}
int main() {
```

```
        setlocale(LC_ALL,"Russian");
    fill(firstEdge, firstEdge + MAX_V, -1);
    cin >> numOfVertex >> numOfEdge;
    sourceVertex = 1;
    destinationVertex = numOfVertex;
    for (int i = 0, u, v, cap; i < numOfEdge; i++) {
        cin >> u >> v >> cap;
        addEdge(u, v, cap);
    }
    int maxFlow = 0;
    int iterationResult = 0;
    while ((iterationResult = findFlow(sourceVertex, INF)) > 0) {
          porivn++;
        fill(visited, visited + MAX_V, false);
        maxFlow += iterationResult;
        prisf++;
    }
    unsigned int end_time = clock();
          unsigned int search_time = end_time - start_time;
    cout << maxFlow << endl <<"Час:"<< search_time<< endl <<"Порівняння:"<<
porivn<< endl <<"Пересилання:"<< prisf;
    return 0;
}
```

ДОДАТОК В

Алгоритм Дініца

```
#include <iostream>
#include <vector>
#include <queue>
#include <ctime>
using namespace std;
struct edge {
	int a, b, f, c;
	int ind;
};
int prisf = 0;
int porivn = 0;
const int inf = 1000 * 1000 * 1000;
const int MAXN = 1050;
unsigned int start_time;
unsigned int end_time;
int n, m;
vector <edge> e;
int pt[MAXN];
vector <int> g[MAXN];
long long flow = 0;
queue <int> q;
int d[MAXN];
int lim;
```

```
void add_edge(int a, int b, int c, int ind) {
	edge ed;
	ed.a = a; ed.b = b; ed.f = 0; ed.c = c; ed.ind = ind;
	g[a].push_back(e.size());
	e.push_back(ed);
	ed.a = b; ed.b = a; ed.f = c; ed.c = c; ed.ind = ind;
	g[b].push_back(e.size());
	e.push_back(ed);
}

bool bfs() {
	prisf++;
	for (int i = 1; i <= n; i++){
		d[i] = inf;
		porivn++;
		prisf+=2;
	}
	d[1] = 0;
	q.push(1);
	prisf+=2;
	while (!q.empty() && d[n] == inf) {
		porivn+=2;
		int cur = q.front();
		q.pop();
		prisf+=3;
		for (size_t i = 0; i < g[cur].size(); i++) {
```

```
                porivn+=1;
                int id = g[cur][i];
                int to = e[id].b;
                prisf+=3;
                porivn+=2;
                if (d[to] == inf && e[id].c - e[id].f >= lim) {
                        d[to] = d[cur] + 1;
                        q.push(to);
                        prisf+=2;
                }
        }
    }
    while (!q.empty()){
        porivn+=2;
        q.pop();
        prisf+=1;
    }
    return d[n] != inf;
}
bool dfs(int v, int flow) {
    porivn+=1;
    if (flow == 0)
        return false;
    porivn+=2;
    if (v == n) {
        return true;
```

```
        }
        for (; pt[v] < g[v].size(); pt[v]++) {
                porivn+=1;
                int id = g[v][pt[v]];
                int to = e[id].b;
                prisf+=2;
                porivn+=2;
                if (d[to] == d[v] + 1 && e[id].c - e[id].f >= flow) {
                        int pushed = dfs(to, flow);
                        prisf+=2;
                        porivn+=1;
                        if (pushed) {
                                e[id].f += flow;
                                e[id ^ 1].f -= flow;
                                prisf+=2;
                                return true;
                        }
                }
        }
        return false;
}
void dinic() {
        unsigned int start_time = clock();
        prisf+=1;
        for (lim = (1 << 30); lim >= 1;) {
                porivn+=2;
```

```
            if (!bfs()) {
                lim >>= 1;
                prisf+=1;
                continue;
                unsigned int end_time = clock();
            }
            for (int i = 1; i <= n; i++){
                porivn+=1;
                pt[i] = 0;
                prisf+=1;
            }
            int pushed;
            while (pushed = dfs(1, lim)) {
                porivn+=1;
                flow = flow + lim;
                prisf+=1;
            }
        }
}
int main() {
        setlocale(LC_ALL,"Russian");
        scanf("%d %d", &n, &m);
        for (int i = 1; i <= m; i++) {
            int a, b, c;
            scanf("%d %d %d", &a, &b, &c);
            add_edge(a, b, c, i);
```

```
	}
	dinic();
	unsigned int end_time = clock();
	unsigned int search_time = end_time - start_time;
	cout << flow << endl <<"Час:"<< search_time<< endl <<"Порівняння:"<< porivn<< endl <<"Пересилання:"<< prisf;
	return 0;
}
```