# ЗМІСТ

# ВСТУП

Орієнтовану мережу можна інтерпретувати як деяку транспортну мережу і використовувати її для вирішення задач про потоки речовин в системі трубопроводів. Уявімо, що деякий продукт передається по системі від джерела, де даний продукт виробляється, до стоку, де він споживається. Джерело виробляє продукт з деякою максимальною швидкістю, а стік з тією ж швидкістю споживає продукт. Потоком продукту в будь-якій точці системи є швидкість руху продукту. За допомогою транспортних мереж можна моделювати течію рідин по трубопроводах, рух деталей на складальних лініях, передачу струму по електричним мережам, інформації - в інформаційних мереж і т. д. Кожне орієнтоване ребро мережі можна розглядати як канал, по якому рухається продукт. Кожен канал має задану пропускну здатність, яка характеризує максимальну швидкість переміщення продукту по каналу. Вершини є точками перетину каналів. Через вершини, відмінні від джерела і стоку, продукт проходить не накопичуючись.

У задачі про максимальний потік ми хочемо знайти максимальну швидкість пересилання продукту від джерела до стоку, при якій не будуть порушуватися обмеження пропускної здатності. Ця проблема була поставлена Т.Є. Харрісом навесні 1955 року, який разом з відставним генералом Ф.С. Россом запропонував спрощену модель залізничного транспортного потоку і висунув саме цю спеціальну задачу як центральну, підказаною цією моделлю. Незабаром після цього був висловлений в якості гіпотези, а потім і встановлений головний результат - теорема «Про максимальний потік і мінімальний розріз».

У даній роботі розглядається узагальнений метод Форда-Фалкерсона для транспортної мережі з обмеженою пропускною здатністю дуг а також з ціною за транспортування одиниці продукту через мережу. В якості алгоритму побудови рішення був обраний алгоритм Баскера-Гоуєна.

# 1 Задача про максимальний потік

## 1.1 Постановка задачі

Орієнтованої мережею називається граф $G = [V, E]$, який складається із сукупності $V$ елементів $x, y, ...$ разом з множиною $E$ деяких впорядкованих пар $(x, y)$ елементів, взятих з $V$.

Вузли - елементи множини $V$.

Дуги - елементи множини $E$. Можливість дуги $(x,x)$ вилучається.

Поставимо кожній дузі $(x, y)$ у відповідність деяке число $c(x, y)$ ,яке називається пропускною здатністю дуги. Пропускна здатність показує яка кількість речовини може пройти по цій дузі в одиницю часу.

Потоком (flow) в мережі є дійсна функція $f : V x V - > R$ задовольняє трьом умовам:

а) обмеження пропускної здатності

$$\forall u{,}v \in V f(u{,}v) \leqslant c(u{,}v)$$

Потік з однієї вершини в іншу не повинен перевищувати задану пропускну здатність.

б) антисиметричність

$$f(u{,}v) = -f(v{,}u) \forall u{,}v \in V$$

Потік з вершини $u$ в вершину $v$ протилежний потоку у зворотному напрямку.

в) збереження потоку

$$\forall u \in V/s{,}t \sum_{v \in V} f(u{,}v) = 0$$

Сумарний потік, що виходить з вершини, що не є джерелом або стоком дорівнює нулю. Величина потоку визначається як сумарний потік, що виходить з джерела.

$$|f| = \sum_{v \in V} f(s{,}v)$$

Будемо називати $s$ джерелом $t$ стоком, а інші вузли - проміжними.

Задача про максимальний потік (maximum flow poblem) полягає в знаходженні потоку максимальної величини. Математично постановка виглядає так:

$$\begin{aligned} &max \quad v = f(s,V) \\ &f(x,V) - f(V,X) = 0, \qquad x \neq s,t, \\ &0 \leqslant f(x,y) \leqslant c(x,y), \quad\ (x,y) \in E, \end{aligned}$$

### 1.2 Метод Форда-Фалкерсона

Метод Форда-Фалкерсона базується на трьох важливих концепціях. Це залишкові мережі, що збільшують шляхи і розрізи. Метод є ітеративним. Спочатку значення потоку встановлюється нуль. $f(u,v) = 0 \forall u,v \in V$ На кожній ітерації величина потоку збільшується за допомогою пошуку збільшуючого шляху (деякого шляху від джерела до стоку вздовж якого можна відправити більший потік) і подальшого збільшення потоку. Цей процес повторюється до тих пір, поки вже неможливо відшукати збільшуваного шлях.

Залишкові мережі

Нехай задана транспортна мережа $G(V,E)$ з джерелом $s$ і стоком $t$. Нехай $f$ деякий потік в $G$. Розглянемо пару вершин $u,v \in V$. Величина додаткового потоку, який ми можемо направити з $u$ в $v$, щоб не перевищити пропускну здатність $c(u,v)$ є залишковою пропускною здатністю ребра $(u,v)$ і задається формулою:

$$c_f(u,v) = c(u,v) - f(u,v)$$

Для транспортної $G(V,E)$ мережі і потоку $f$ залишковою мережею в $G$, породженою потоком $f$ є мережа $G_f = (V,E_f)$ где

$$E_f = (u,v) \in VxV : c_f(u,v) > 0$$

Таким чином по кожному ребру залишкової мережі або залишковому ребру можна направити потік більше нуля.

Збільшуючі шляхи

Для заданої транспортної мережі $G = (V,E)$ і потоку $f$ збільшуючим шляхом $p$ є простий шлях з $s$ в $t$ в залишковій мережі $G_f$. Максимальна величина, на яку можна збільшити потік уздовж кожного ребра $p$ збільшуючого шляху називається пропускною здатністю шляху і задається формулою:

$$c_f(p) = min\{c(u,v) : (u,v) \in p\}$$

.

Розрізи транспортних мереж

Розрізом транспортної мережі $G(V,E)$ називається розбиття множини вершин на множини $S$ та $T$ такі що $s \in S, t \in T$. Якщо $f$ потік - то чистий потік через розріз $(S,T)$ визначимо як $f(S,T)$. Пропускну здатність розрізу $(S,T)$ визначимо відповідно $c(S,T)$. Мінімальним розрізом є розріз, пропускна здатність якого серед всіх розрізів мінімальна. Як видно, потік через розріз, на відміну від пропускної здатності розрізу, може включати і від'ємні доданки.

Теорема 1 ( Про максимальний потік і мінімальний розріз). Для будь-якої мережі максимальна величина потоку з $s$ в $t$ дорівнює мінімальній пропускній здатності розрізу, що відокремлює $s$ и $t$. [1]

# 2 Задача про максимальний потік мінімальної вартості

## 2.1 Постановка задачі

Нехай кожній дузі відповідає не лише пропускна здатність а також і велечина $c_{ij} > 0$ яка дорівнює вартості транспортування одиниці товару через ребро $(i,j) \in E$ мережі. Задача пошука потока із $s$ в $t$ заданої потужності $v$ і мінімальної вартості має вигляд:

$$Z = \sum_{(i,j)\in E} c_{ij}x_{ij} \to min_x;$$

$$\sum_{i:(i,j)\in E} x_{ij} - \sum_{k:(j,k)\in E} x_{jk} = \begin{cases} -v, & j = s; \\ 0, & j \neq s,t; \\ v, & j = t; \end{cases}$$

$$0 \leq x_{ij} \leq b_{ij}, (i,j) \in E$$

## 2.2 Алгоритм Баскера-Гоуєна

Для розв'язування задачі про максимальний потік мінімальної вартостіі будемо використовувати алгоритм Баскера-Гоуєна:

а) Знайдемо поток мінімальної вартості із $s$ в $t$.

б) З'ясуемо максимальну величину потоку, яку можна пропустити через цей шлях.

в) Якщо ця велечина більша за потрібну потужність мережі, візмемо її рівною потрібній потужності.

г) Збільшити поток по цьому потоку на максимальну величину(але так щоб загальний поток через мережу не перевищував потрібний)

д) Розрахувати ціну за транспортування потужності. Додати до загальної ціни потоку.

е) Якщо поток по мережі дорівнюе заданому, то припинити роботу алгоритму. Інакше перейти на крок $a$ з оновленими велечинами потоків.

## 2.3 Програмна реалізація

В програмній реалізації пошук потоку мінімальної вартості реалізований так: Починаючи з джерела кожна вершина пов'язана з данною додається до черги (queue). При цьому для вершини запам'ятовується вершина з якої мі до неї прийшли та загальа вартість потоку по цьому шляху. Для того щоб порахувати загальну вартість потоку треба знайти мінімальну пропускну здатність шляху, а потім знайти сумму добутків мінімальної пропускної здатності шляху на ціну на кожній дузі шляху. В результаті на $n$-му кроці алгоритму ми прийдемо до стоку $t$ і при цьому будемо знати яку величину потоку ми можео пропустити по шляху і його повну вартість. Якщо вершина вже була оброблена алгоритмом і для неї ми знаємо суммарну вартість і батьківську вершину то батьківською оберається та вершина яка дає меншу суммарну ціну за транспортування. Будемо виконувати алгоритм доки в черзі є хоч один елемент. Якщо елементи в черзі закінчилися то перевіримо чи є у вершини-стоку $t$ батькывська вершина. Якщо так то ми можемо дізнатися велечину потока по цьому шляху, його вартість а також вершини і ребра що є елементами цього шляху. Перераховуємо повну пропускну здатність мережі та вартість потоку по ній. Використаемо ці данні для того щоб перерахувати пропускну здатність мережі і перейдемо на перший крок алгоритму з новими данними. Якщо ні то тоді шляху з джеоела в сток немає. Максимальна пропускна здатність і ціна дорівнюють останнім розрахованим велечинам.

Программа виконана у вишляді додатку для командної строки на мові програмування c#. Для запуску программи треба виконати програму передавши їй назву файла (без формату) як перший агрумент. Программа зчитує данні з файлу у форматі .csv (данні розділені запятою). Наприклад якщо початкові данні лежать у файлі main.csv то у вікні командної строки треба виконати наступну команду:

$$MinCostFlow.exe \quad main$$

Після закінчення роботи програми результат буде записано в файл з такоюж назвою як і файл з початковими двнними і текстом _result у кінці назви. Користувач побачить провідомлення про це англійською мовою. В разі аварійної зупинки програми користувач дізнаетья про це за текстом помилки на єкранні (наприклад якщо файла з таким ім'ям немає).

Також була протестована швидкість роботи программи. Для цього я сгенерував повний граф (кожна вершина пов'язана з усіма іншими) з тисячею вершин і випадковою пропускною здатністю вузлів (від 0 до 1000). Такимож чинов я сгенерував матрицю вартостей транспортування. Я хотів знайти мінімальну вартість максимально можливого потоку по цій мережі. Як максимальний поток я обрав велечину яку отримав складаннямпропускних здатностей всіх вершин які йдуть до стоку. Тобто граф мав

$1000 * 1000 = 1000000$ ребер. Час роботи программи склав приблизно 25 хвилин.

# 3 Приклади

Задача 1

У деякої компанії "Аврора"є фабрика в Берліні, що виробляє стільці, а в Бремені склад, де вони зберігаються. Компанія орендує місце на вантажівках інших фірм для доставки стільців з фабрики на склад. Оскільки вантажівки їздять по певних маршрутах між містами і мають обмежену вантажопідйомність, компанія може перевозити не більше певної кількості ящиків на день між містами. Також, компанія сплачує певну ціну за перевезення одиниці товару по кожному відсіку маршруту. Компанія не може вплинути на маршрути і пропускну здатність. Її завдання визначити, якими будуть витрати на перевезення 100 ящиків з Берліна до Бремена.

Фабрику будемо вважати джерелом ($s$), склад - стоком ($t$).
Маршрути між містами будемо вважати ребрами мережі.
Вантажопідйомність вантажівок будемо вважати обмеженням пропускної спроможності на ребрах $c(u,v)$. ціну за перевезення однієї коробки будемо вважати ціною ребра мережі $c_{ij}(v)$. Занесемо дані в таблицю, де кожен елемент - обмеження пропускної здатності між відповідними містами (нуль означає що маршрут відсутній):

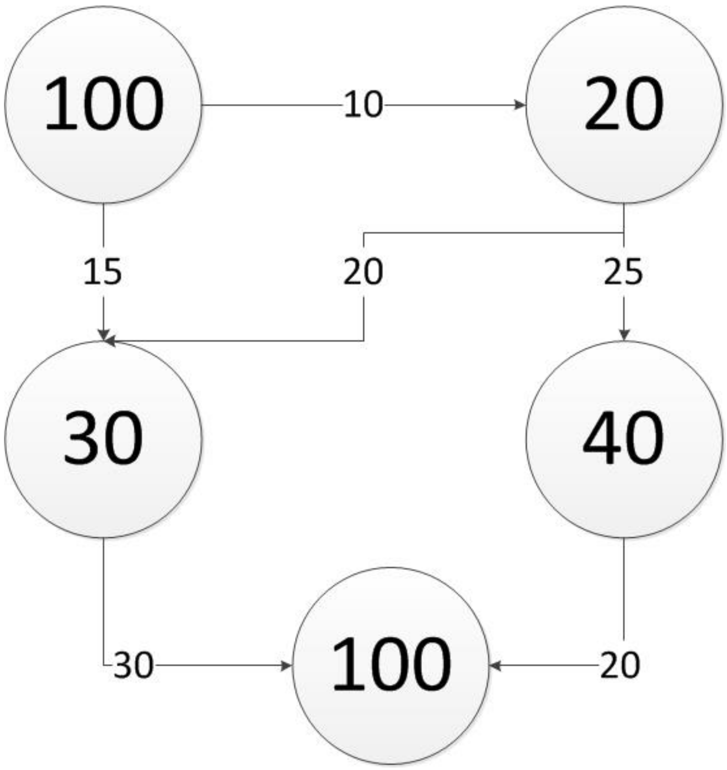



Рис. 3.1 Транспортна мережа

| | | | | |
|---|---|---|---|---|
| 0 | 70 | 0 | 40 | 0 |
| 0 | 0 | 80 | 0 | 20 |
| 0 | 0 | 0 | 100 | 0 |
| 0 | 0 | 0 | 0 | 120 |
| 0 | 0 | 0 | 0 | 0 |

Ціна за транспортування одиниці товару представмо у вигляді таблиці, де коже елемент - ціна за транспортування одиниці товару між містами:

| | | | | |
|---|---|---|---|---|
| 0 | 3 | 0 | 5 | 0 |
| 0 | 0 | 1 | 0 | 5 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 |

В результаті роботи програми було отримано наступне рішення:

| | | | | |
|---|---|---|---|---|
| 0 | 60 | 0 | 40 | 0 |
| 0 | 0 | 60 | 0 | 0 |
| 0 | 0 | 0 | 60 | 0 |
| 0 | 0 | 0 | 0 | 100 |
| 0 | 0 | 0 | 0 | 0 |

Повна вартість транспортування ста одиниць товару через мережу - 600 одиниць.

Задача 2

Компанія постачальник високошвидкісного інтернету отримала потенціального замовника якому треба раз на день синхронізувати основну та запасну бази данних. Компанія замовник хоче робити це як умога дешевше. Очікуваний розмір пакета данних 100 гігабайт. Постачальник послуг високошвидкісного інтернету знає пропускну здатність всіх вузлів своєї мережі, та вартість передачі одного гігабайту трафіку через кожне сполучення в мережі. Їй треба знайти яким шляхом маршрутизовати трафік та розрахувати мінімальну ціну яку їй доведеться заплатити за передачу цих данних.

Зведемо дану задачу до задачі про максимальний потік.

Будемо вважати основний сервер джерелом $s$ а запасний стоком $t$. проміжні сполучення будемо вважати ребрами мережі, з обмеженнями на пропускну здатність $c(x,y)$ заданої в вигляді таблиці. Вартість передачі

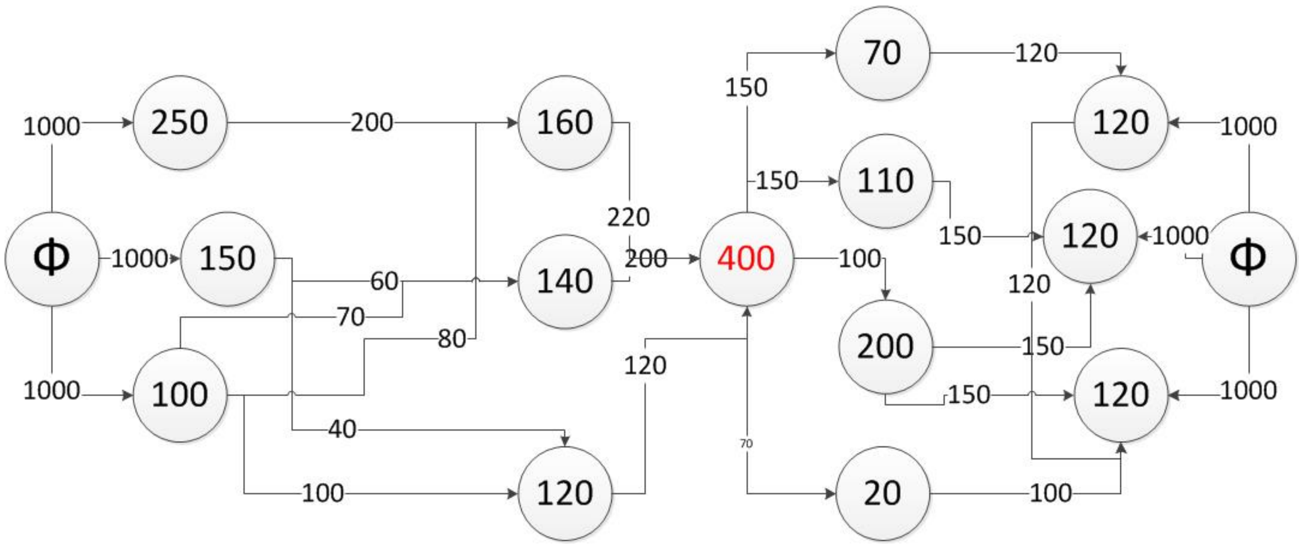



Рис. 3.2 Транспортна мережа

одного гігабайту будемо вважати ціною за транспортування $c_{ij}(v)$

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 100 | 70 | 180 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 120 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 70 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 130 | 60 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 90 | 100 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 70 | 70 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 70 | 30 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 100 | 0 | 120 | 90 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 110 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 60 | 0 | 150 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 100 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Вартість передачі одного гігабайту данних по проміжному ребру мережі задана таблицею.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 10 | 70 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 9 | 10 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 6 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 7 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 3 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 12 | 9 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 3 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

В результаті роботи програми було отримано наступне рішення:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 90 | 10 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 90 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 100 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Компанія постачальник з'ясувала що передача данних для компанії замовника обійдеться їй в 3120 грошових одиниць. Тепер постачальник може підготувати комерційну пропозицію для замовника.

Задача 3

Туристичній компанії треба переправити группу відпочиваючих з Одеси до Барселони.Але оскільки группа зібралася дуже піздно то майже всі квитки на потяг до Києва і літак Київ-Барселона вже викуплені. Компанія зібрала данні про наявні квитки на різні проможні рейси та потяги і тепер їй треба з'ясувати як переправити группу відпочиваючих в 100 чоловік найшвидше.

Зведемо дану задачу до задачі про максимальний потік.

Будемо вважати Одесу джерелом $s$ а Барселону - стоком $t$. Авіа та залізничні рейси вважатимемо ребрами мережі, з обмеженнями на пропускну здатність $k(x,y)$ заданої в вигляді таблиці.

Час переправи будемо вважати ціною ребра мережі (в хвидинах).

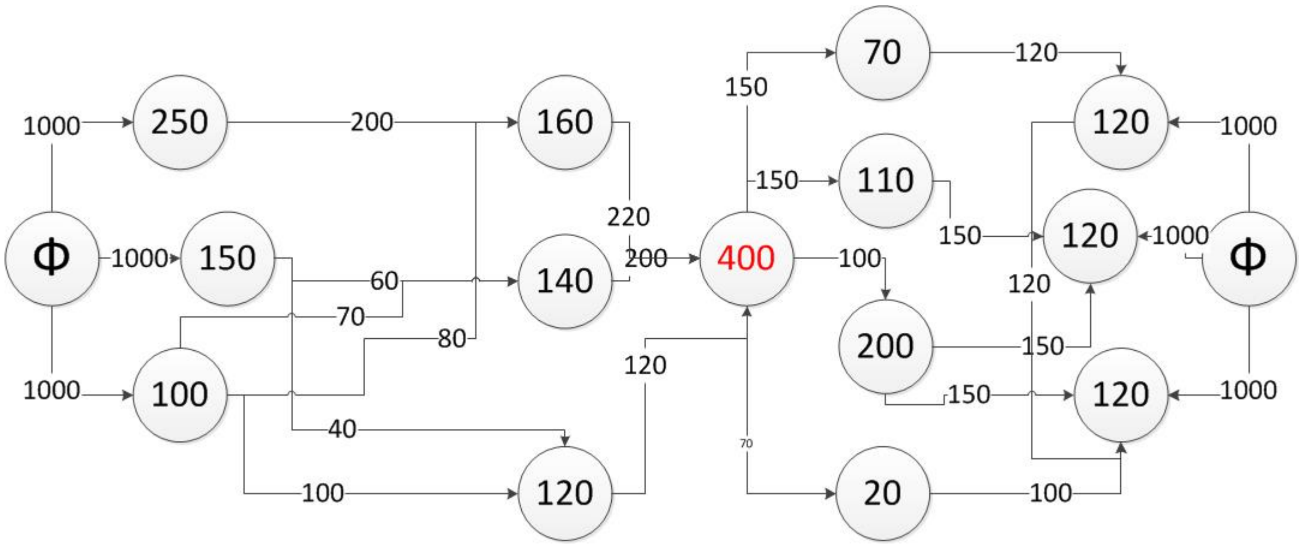



Рис. 3.3 Транспортна мережа

| | | | | | |
|---|---|---|---|---|---|
| 0 | 50 | 0 | 60 | 70 | 0 |
| 0 | 0 | 100 | 0 | 40 | 0 |
| 0 | 0 | 0 | 80 | 0 | 90 |
| 0 | 0 | 0 | 0 | 70 | 0 |
| 0 | 0 | 0 | 0 | 0 | 50 |
| 0 | 0 | 0 | 0 | 0 | 0 |

Час переправі заданий матрицею

| | | | | | |
|---|---|---|---|---|---|
| 0 | 100 | 0 | 200 | 250 | 0 |
| 0 | 0 | 130 | 0 | 160 | 0 |
| 0 | 0 | 0 | 140 | 0 | 0 |
| 0 | 0 | 0 | 0 | 175 | 0 |
| 0 | 0 | 0 | 0 | 0 | 60 |
| 0 | 0 | 0 | 0 | 0 | 0 |

В результаті роботи програми було отримано наступне рішення:

| | | | | | |
|---|---|---|---|---|---|
| 0 | 50 | 0 | 0 | 50 | 0 |
| 0 | 0 | 50 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 50 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 50 |
| 0 | 0 | 0 | 0 | 0 | 0 |

В результаті роботи програми компанія знає яким шляхом найкраще доставити туристів до пунктів відпочинку.

## 4 Висновки

У даній роботі було розглянуто узагальнення задачі про максимальний потік для транспортної мережі з ціною на транспортування одиниці товару через ребро. Як алгоритм рішення був обрний алгоритм Баскера-Гоуєна.Була розроблена програма на мові c# що реалізує даний метод.Робота програми була протестовна на декількох прикладних задачах.

# СПИСОК ВИКОРИСТАННОЇ ЛІТЕРАТУРИ

[1] Форд, Л.Р. Потоки в сетях. /Л.Р. Форд, Д.Р. Фалкерсон - Москва : Мир, 1966. - 273 с.

[2] Таха Хемди А. ВВедение в исследование операций /А. Таха - Москва : Вильямс, 2006. - 912 с.

[3] Вагнер Г. Основы исследования операций. Том 1./Г. Вагнер - Москва : Мир, 1973. - 336 с.

[4] Кормен Томас Х. Алгоритмы: построение и анализ./К.Х. Томас, Ч.Л. Лейзерсон, Р.Д. Риверс, К. Штайн - Москва : Вільямс, 2005. - 1296 с.

[5] Ху Т. Целочислнное програмированние и потоки в сетях/Т. Ху - Москва : Мир, 1974. - 513 с.

[6] Арсирий А.В. Сетевые модели./ А.В.Арсирій, Б.Ф.Трофимов, Є.М. Страхов - Одеса : Одеський національний університет імені І.І.Мечникова, 2011. - 42 с.

# ДОДАТОК А

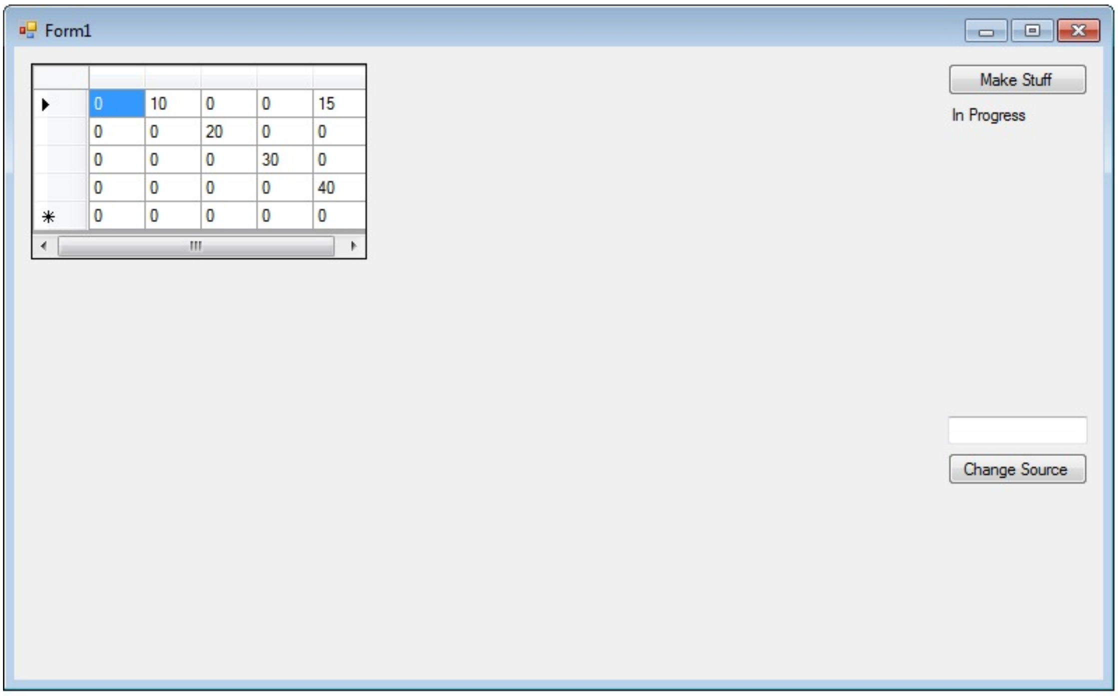
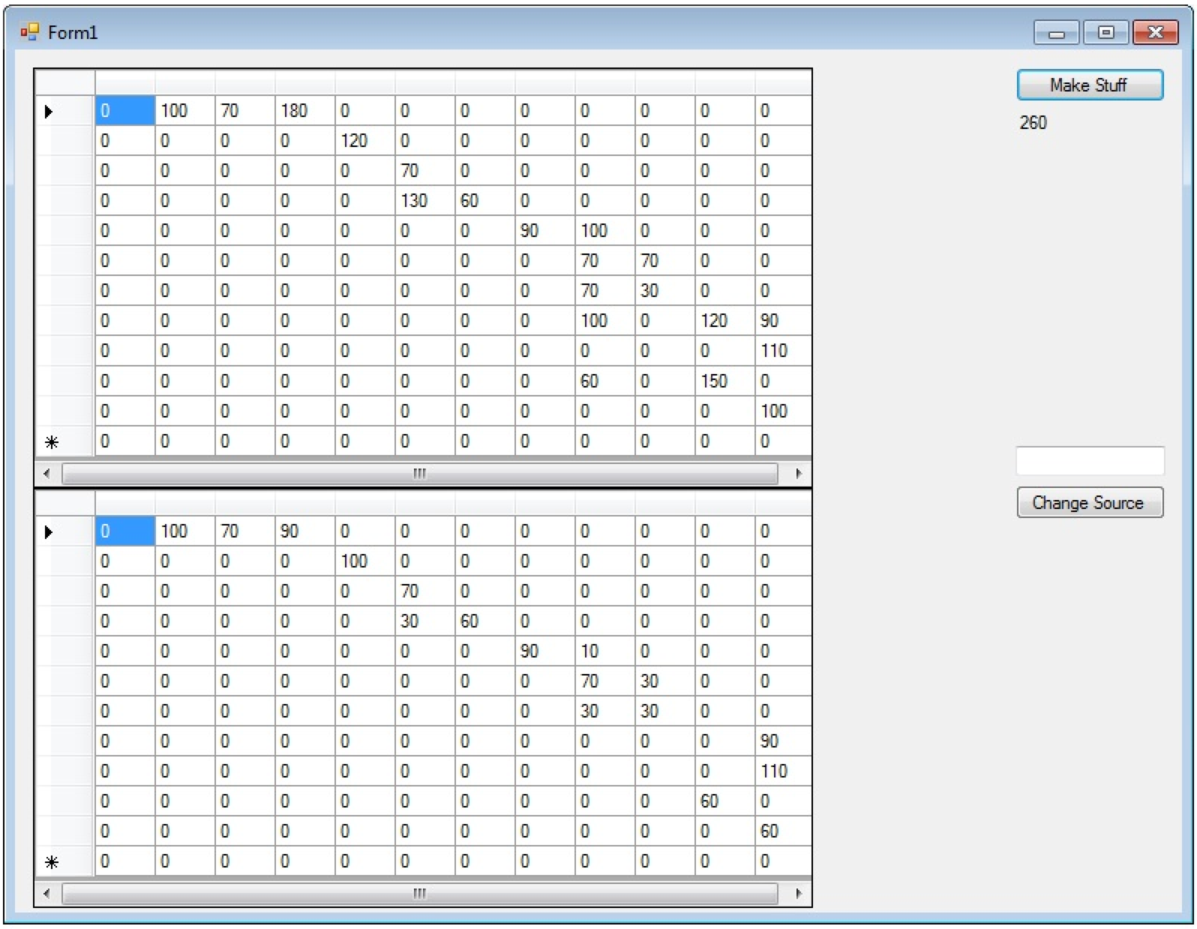



Рис. 4.1 Інтерфейс програми




| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 100 | 70 | 180 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 120 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 70 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 130 | 60 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 90 | 100 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 70 | 70 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 70 | 30 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 100 | 0 | 120 | 90 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 110 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 60 | 0 | 150 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 100 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 100 | 70 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 70 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 30 | 60 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 90 | 10 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 70 | 30 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 30 | 30 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 90 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 110 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 60 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 60 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Рис. 4.2 Приклад використання програми

# ДОДАТОК Б

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MinCostFlow
{
    class Program
    {
        static void Main(string[] args)
        {
            dataReader data;
            string source = "main";
            try
            {
                if (args.Length > 0)
                {
                    data = getData(args[0]);
                    source = args[0];
                }
                else
                    data = getData(source);
            }
            catch (FileNotFoundException Ex)
            {
                Console.WriteLine($"File␣{Ex.FileName}␣was␣not␣found,␣please␣make␣sure␣file
                    ␣is␣present␣and␣has␣a␣propper␣format");
                Console.ReadKey();
                return;
            }
            int[,] flowsCopy = new int[data.flows.GetLength(0), data.flows.GetLength(0)];
            Array.Copy(data.flows, 0, flowsCopy, 0, data.flows.Length);
            FlowCost result = CalculateFlow(data.flows, data.costs, 0, data.flows.
                GetLength(0) - 1, data.neededFlow);
            var usedFlow = getUsedFlow(flowsCopy, result.resultingFlows);
            writeResult(result.totalCost, usedFlow, source);
            Console.WriteLine($"Result␣is␣written␣to␣file␣{source}_result.csv");
            Console.ReadKey();
        }

        private static void writeResult(int totalCost, int[,] usedFlow, string source = "
            main")
        {
            string fullAppName = System.Reflection.Assembly.GetExecutingAssembly().
                Location;
            string fullAppPath = System.IO.Path.GetDirectoryName(fullAppName);
            string root = System.IO.Path.GetDirectoryName(fullAppPath);
            root = System.IO.Path.GetDirectoryName(root);
```

```
        string DataPath = String.Concat(root, "\\Data", "\\" + source + "_result.csv")
            ;

        writeData(DataPath, totalCost, usedFlow);
    }

    private static void writeData(string dataPath, int totalCost, int[,] usedFlow)
    {
        StreamWriter sw = new StreamWriter(dataPath);
        sw.WriteLine(totalCost);
        sw.WriteLine("");
        for (int i = 0; i < usedFlow.GetLength(0); i++)
        {
            var line = "";
            for (int j = 0; j < usedFlow.GetLength(0); j++)
            {
                line += usedFlow[i, j] + ",";
            }
            line = line.Trim(',');
            sw.WriteLine(line);
        }
        sw.Close();
    }

    private static dataReader getData(string source = "main")
    {
        string fullAppName = System.Reflection.Assembly.GetExecutingAssembly().
            Location;
        string fullAppPath = System.IO.Path.GetDirectoryName(fullAppName);
        string root = System.IO.Path.GetDirectoryName(fullAppPath);
        root = System.IO.Path.GetDirectoryName(root);
        string DataPath = String.Concat(root, "\\Data", "\\" + source + ".csv");

        if (File.Exists(DataPath))
            return readData(DataPath);
        else
            throw new FileNotFoundException();
    }

    private static dataReader readData(string filePath)
    {
        var result = new dataReader();
        //int[,] flows;
        //int[,] costs;
        try
        {
            StreamReader sr = new StreamReader(filePath);

            string line;

            line = sr.ReadLine();
            result.neededFlow = System.Convert.ToInt32(line.Trim(','));
            line = sr.ReadLine();
```

```
            var size = line.Split(',').Length;
            result.flows = new int[size, size];
            result.costs = new int[size, size];
            var lineNum = 0;
            while (line.Length > 2)
            {
                string[] LimitersStr = line.Split(',');
                int len = LimitersStr.Length;
                for (int i = 0; i < size; i++)
                {
                    result.flows[lineNum, i] = Int32.Parse(LimitersStr[i]);
                }
                lineNum++;
                line = sr.ReadLine();
            }
            lineNum = 0;
            while (sr.Peek() > 0)
            {
                line = sr.ReadLine();
                string[] LimitersStr = line.Split(',');
                //int len = LimitersStr.Length;
                for (int i = 0; i < size; i++)
                {
                    result.costs[lineNum, i] = System.Convert.ToInt32(LimitersStr[i]);
                }
                lineNum++;
            }
            sr.Close();
            //result.flows = flows;
            //result.costs = costs;
            return result;
        }
        catch (Exception ex)
        {
            return readData("main");
        }
    }

    private static int[,] getUsedFlow(int[,] flows, int[,] resultingFlows)
    {
        var Size = flows.GetLength(0);
        int[,] result = new int[Size, Size];
        for (int i = 0; i < Size; i++)
            for (int j = 0; j < Size; j++)
            {
                result[i, j] = flows[i, j] - resultingFlows[i, j];
            }
        return result;
    }

    private static FlowCost CalculateFlow(int[,] flows, int[,] costs, int s, int t,
        int neededFlow)
    {
```

```
    var totalFlow = 0;
    var totalCost = 0;
    if (s == t)
        throw new ArgumentException("input␣and␣output␣should␣be␣different␣points");
    MilestoneHist[] path = bfs(flows, costs, s, t);
    while (path[t] != null)
    {
        int maxFlow = getMaxFlowForPath(flows, path, s, t);
        if (neededFlow - totalFlow < maxFlow)
        {
            maxFlow = neededFlow - totalFlow;
            flows = updateFlows(maxFlow, flows, path, s, t);
            totalCost += maxFlow * path[t].totalCost;
            path = bfs(flows, costs, s, t);
            totalFlow += maxFlow;
            return new FlowCost { totalCost = totalCost, resultingFlows = flows,
                totalFlow = totalFlow };
        }
        flows = updateFlows(maxFlow, flows, path, s, t);
        totalCost += maxFlow * path[t].totalCost;
        path = bfs(flows, costs, s, t);
        totalFlow += maxFlow;
    }
    return new FlowCost { totalCost = totalCost, resultingFlows = flows, totalFlow
         = totalFlow };
}

private static int[,] updateFlows(int maxFlow, int[,] flows, MilestoneHist[] path,
     int s, int t)
{
    int to = t;
    int from = path[to].pointNum;
    while (from != -1)
    {
        flows[from, to] -= maxFlow;
        if (flows[from, to] < 0)
            throw new Exception("result␣flow␣can't␣be␣less␣than␣0,␣some␣error␣in␣
                logic");
        to = from;
        from = path[to].pointNum;
    }
    return flows;
}

private static int getMaxFlowForPath(int[,] flows, MilestoneHist[] path, int s,
    int t)
{
    int to = t;
    int from = path[to].pointNum;
    var maxFlow = flows[from, to];
    while (from != -1)
    {
        maxFlow = Math.Min(maxFlow, flows[from, to]);
```

```
        to = from;
        from = path[to].pointNum;
    }
    return maxFlow;
}

public static MilestoneHist[] bfs(int[,] rGraph, int[,] costGraph, int s, int t)
{
    var debug = 0;
    int Size = rGraph.GetLength(0);
    MilestoneHist[] parent = new MilestoneHist[Size];
    parent[s] = new MilestoneHist { totalCost = 0, pointNum = -1 };
    bool[] visited = new bool[Size];

    Queue q = new Queue();
    q.Enqueue(s);

    while (q.Count != 0)
    {
        int u = (int)q.Dequeue();
        //debug++;
        //if (debug > 10000)
        // throw new StackOverflowException();

        for (int v = 0; v < Size; v++)
        {
            if (v != s && rGraph[u, v] > 0 /*&& IsNotCycle(u,v,parent)*/)
            {
                if (parent[v] == null)
                {
                    parent[v] = new MilestoneHist { pointNum = u, totalCost = parent
                        [u].totalCost + costGraph[u, v] };
                    q.Enqueue(v);
                }
                else
                {
                    int newCost = parent[u].totalCost + costGraph[u, v];
                    int oldCost = parent[v].totalCost;
                    if (newCost < oldCost)
                    {
                        parent[v] = new MilestoneHist { pointNum = u, totalCost =
                            newCost };
                        q.Enqueue(v);
                    }
                }
            }
        }
    }
    // If we reached sink in BFS starting from source, then return
    // true, else false
    return parent;
}
```

```
    public static Milestone[,] ConvertToMilestones(int[,] costs)
    {
        int size = costs.GetLength(0);
        Milestone[,] res = new Milestone[size, size];
        for (int i = 0; i < size; i++)
            for (int j = 0; j < size; j++)
            {
                res[i, j] = new Milestone { InitialFlow = costs[i, j], RemainingFlow =
                    costs[i, j] };
            }
        return res;
    }
}

class Milestone
{
    public int InitialFlow { get; set; }
    public int RemainingFlow { get; set; }
    public int Cost { get; set; }
    public Milestone prevMilestone { get; set; }

    public int getTotalCost()
    {
        var maxFlow = this.InitialFlow;
        var totalCost = 0;
        var prevMilestone = this.prevMilestone;
        while (prevMilestone != null)
        {
            maxFlow = Math.Min(maxFlow, prevMilestone.InitialFlow);
            prevMilestone = prevMilestone.prevMilestone;
        }
        prevMilestone = this.prevMilestone;
        while (prevMilestone != null)
        {
            totalCost += prevMilestone.Cost * maxFlow;
            prevMilestone = this.prevMilestone;
        }
        return totalCost;
    }
}

class MilestoneHist
{
    public int pointNum { get; set; }
    public int totalCost { get; set; }

    public static int GetCostByHistory(MilestoneHist[] history)
    {
        throw new NotImplementedException();
    }
}

class FlowCost
```

```
    {
        public int totalFlow { get; set; }
        public int totalCost { get; set; }
        public int[,] resultingFlows { get; set; }
    }
    class dataReader
    {
        public int neededFlow { get; set; }
        public int[,] flows { get; set; }
        public int[,] costs { get; set; }
    }
}
```