МІНІСТЕРСТВО ОСВІТИ УКРАЇНИ

Національний університет “Львівська політехніка”

І Н С Т Р У К Ц І Ї

до лабораторних робіт з курсу
“Дискретні моделі в САПР”
для базового напрямку “Комп’ютерні науки”

Затверджено
на засіданні кафедри “Системи
автоматизованого проектування”
Протокол №_____ від _____ р.

Львів - 2012

Інструкції до лабораторних робіт № 1 - 5 з курсу “Дискретні моделі в САПР” для базового напрямку “Комп’ютерні науки”
/Склали: С.П.Ткаченко, І.І.Чура, Р.З. Кривий -Львів: НУ “Львівська політехніка”, 2012.

Укладачі: С.П.Ткаченко, канд.техн.наук, доц.
І.І.Чура, канд.техн.наук, доц..
Р.З. Кривий, канд.техн.наук, асистент

Відповідальний за випуск: _____

Рецензенти: _____
_____

І Н С Т Р У К Ц І Я № 1

до лабораторної роботи

АЛГОРИТМ ПОБУДОВИ ДЕРЕВ

з курсу

“**Дискретні моделі в САПР**”

для базового напрямку “Комп’ютерні науки”

1. МЕТА РОБОТИ

Метою даної лабораторної роботи є вивчення алгоритмів рішення задач побудови остових дерев.

2. КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

ПОЛОЖЕННЯ ТЕОРІЇ ГРАФІВ

Для того, щоб розглянути алгоритми побудови дерев нагадаємо деякі основні положення теорії графів.

Графом G називають скінчену множину V з нерефлексивним симетричним відношенням R на V. Визначим E як множину симетричних пар в R. Кожний елемент V називають вершиною. Кожний елемент E називають ребром, а E множиною ребер G.

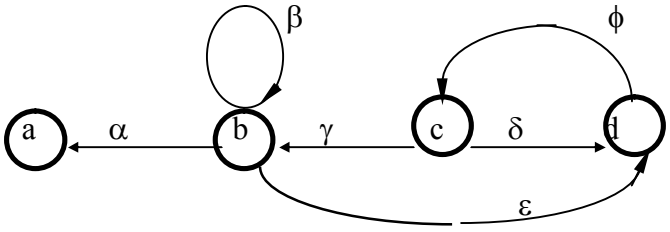
Граф називається *зв’язним*, якщо в ньому для будь-якої пари вершин знайдеться ланцюг, який їх з’єднує, тобто, якщо по ребрах (дугах) можна попасти з будь-якої вершини в іншу.

Цикл - це ланцюг, в якого початкова і кінцева точки співпадають.

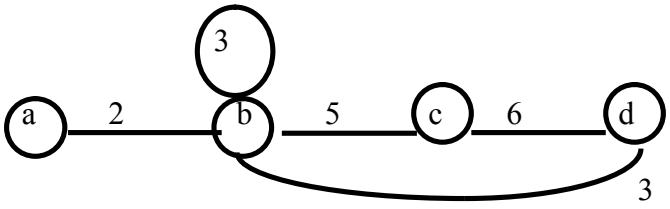
Дерево - це зв’язний граф без циклів.

В графі на рис.1а наступні сукупності дуг і вершин утворюють дерево:

{ α, γ, ε }, { α, γ ,ϕ }, { α, ε, δ }, { ϕ, γ }, { α, γ }, { ε }, { γ }



а)



б)

Рис.1.

Сукупність дуг цього ж графа {ϕ, γ, ε} не є деревом, так як містить цикл.

Покриваючим деревом графа називається любе дерево, що утворене сукупністю його ребер(дуг), які включають всі вершини графа.

Лісом називається будь-яка сукупність дуг (ребер) інцидентних до вершин, які не містять циклів. Таким чином, ліс складається з одного або більше дерев.

Остовним деревом графа називається будь-яке дерево, яке утворене сукупністю дуг, які включають всі вершини графа. В графі, який показано на рис.1, сукупність дуг {α,ε,ϕ} утворює остовне дерево, так як вона включає всі вершини даного графа a,b,c,d. Будь-який зв'язний граф має остовне дерево.

Коренем орієнтованого дерева (прадерева) називається його вершина, в яку не входить жодна з дуг.

Орієнтований ліс визначається як звичайний, тільки складається не з простих дерев, а орієнтованих.

Остовним орієнтованим деревом називається орієнтоване дерево, яке одночасно є і остовим деревом.

Остовним орієнтованим лісом називається орієнтований ліс, який включає всі вершини відповідного графа.

В*ага дерева* - це сума ваг його ребер.

Поставимо у відповідність кожній дузі (x,y) графа G вагу a(x,y). Вага орієнтованого лісу (або орієнтованого дерева) визначається як сума ваг дуг, що входять в даний ліс (дерево).

Максимальним орієнтованим лісом графа G називається орієнтований ліс графа G з максимально можливою вагою.

Максимальним орієнтованим деревом графа G називається орієнтоване дерево графа G з максимально можливою вагою. *Мінімальні орієнтовані ліс і дерево* визначаються аналогічним чином.

Кущ(букет) - зв'язний фрагмент графа.

Представлення графів в ЕОМ, в більшості випадків, здійснюється з допомогою матриць: суміжності, зв'язності, інцидентності і ваг. Так для графа зображеного на рис.1а.:

а) матриця суміжності S(i, j).

	a	b	c	d
a	0	1	0	0
b	1	1	1	1
c	0	1	0	1
d	0	1	1	0

s_{ij} = 1, якщо вершина, якій відповідає і-та стрічка є початковою для дуги, що відповідає j-му стовпчику;

s_{ij} = -1, якщо вершина, якій відповідає і-та стрічка є кінцевою для дуги, що відповідає j-му стовпчику;

s_{ij} = 0 , в інших випадках.

б) матриця зв'язності Z(i,j)

	a	b	c	d
a	0	1	0	0
b	1	1	1	1
c	0	1	0	2
d	0	1	2	0

z_{ij} = n , де n - кількість ребер між і-ою і j-ою вершинами;

в) матриця інцидентності IN(i, j), в якій індекс і - дуги, а індекс j - вершини

	α	β	γ	δ	ε	ϕ
a	1	0	0	0	0	0
b	-1	1	1	0	-1	0
c	0	0	-1	-1	0	1
d	0	0	0	1	1	-1

in_{ij} = -1 , якщо вершина інцидентна дузі і є початковою для цієї дуги;

in_{ij} = 1 , якщо вершина інцидентна дузі і є кінцевою для цієї дуги;

in_{ij} = 0 , в інших випадках.

г) матриця ваг V(i, j) для графа на рис.1б.

	a	b	c	d
a	0	2	0	0
b	2	3	5	3
c	0	5	0	6
d	0	3	6	0

z_{ij} = n , де n - вага ребра між і-ою і j-ою вершинами.

ПОШУК ОСТОВОГО ДЕРЕВА.

Нехай G = (V, E) -- зв'язний граф, у якому кожне ребро (u,v) позначено числом c(u, v), що називається вагою ребра. Остовним деревом графа G називається дерево, що містить всі вершини V графа G. Вага остового дерева обчислюється як сума ваг всіх ребер, що входять у це дерево.

Існують різні методи побудови максимальних остових дерев. Багато з них ґрунтуються на наступній властивості максимальних остових дерев . Нехай G = (V, E) - зв'язний граф із заданою функцією вартості, що задана на множині ребер. Позначимо через U підмножину вершин V. Якщо (i, v) -- таке ребро найбільшої вартості, що й належить U і v належить V \ U, тоді для графа G існує максимальне остове дерево, що містить ребро (i, v).

Існує декілька популярних алгоритмів побудови максимального остового дерева для позначеного графа G = (V, E), що використовують описану властивість.

Алгоритм Борувки.

Це алгоритм знаходження мінімального остового дерева в графі. Вперше був опублікований в 1926 році Отакаром Борувкой, як метод знаходження оптимальної електричної мережі в Моравії. Робота алгоритму складається з декількох ітерацій, кожна з яких полягає в послідовному додаванні ребер до остового лісу графа, до тих пір, поки ліс не перетвориться на дерево, тобто, ліс, що складається з однієї компоненти зв'язності.

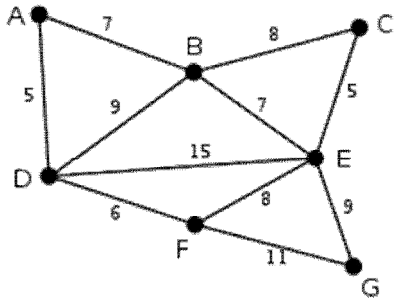
У псевдокоді, алгоритм можна описати так:

1. Спочатку, нехай T - порожня множина ребер (представляє собою остовий ліс, до якого кожна вершина входить в якості окремого дерева).
2. Поки T не є деревом (поки число ребер у T менше, ніж V-1, де V - кількість вершин у графі):
    - Для кожної компоненти зв'язності (тобто, дерева в остовому лісі) в підпункті з ребрами T, знайдемо ребро найменшої ваги, що зв'язує цю компоненту з деякої іншої компонентою зв'язності. (Передбачається, що ваги ребер різні, або як-то додатково впорядковані так, щоб завжди можна було знайти єдине ребро з мінімальною вагою).
    - Додамо всі знайдені ребра в множину T.
3. Отримана множина ребер T є мінімальним остовим деревом вхідного графа.

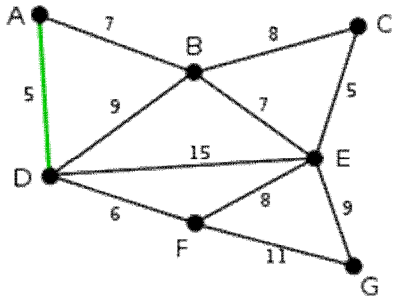
Алгоритм Крускала

Найбільш відомий алгоритм Крускала (Joseph Kruskal) був придуманий автором у 1956 році. Основна стратегія цього алгоритму така: ребра упорядковуються за вагою; на кожному кроці до споруджуваного остового дерева додається найлегше ребро, яке з'єднує вершини з різних компонент. Таким чином на кожному кроці побудована множина складається з однієї або більше нетривіальних компонент, кожна з яких є підграфом деякого мінімального кістяка. Час роботи алгоритму Крускала становить O (ElogE) при використанні для зберігання компонент зв'язності системи непересічних множин з об'єднанням за рангом і стиском шляхів (найшвидший відомий метод). Більша частина часу йде на сортування ребер.

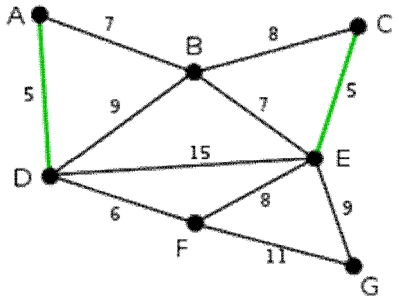
Приклад:



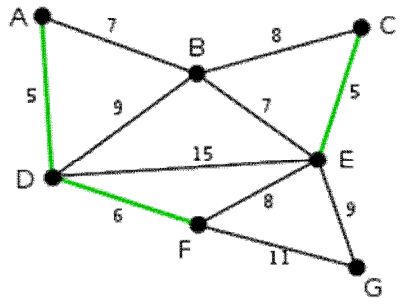
Це наш вихідний граф. Числа біля дуг вказують їх вагу. Жодна з дуг не виділена.



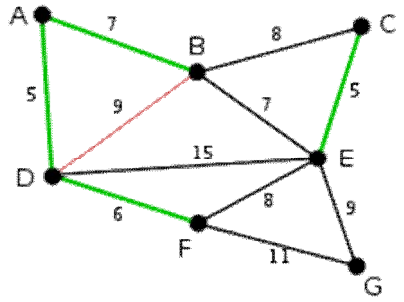
AD і CE є найкоротшими, довжиною 5, AD була вибрана довільно, тому вона виділяється.



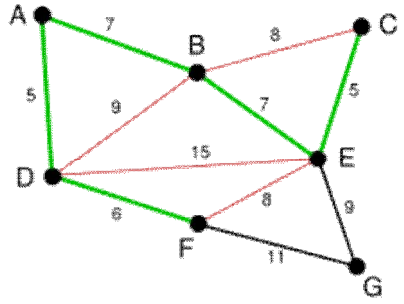
CE тепер найкоротша, що не утворює цикл, довжиною 5, тому вона виділяється в якості другої дуги.



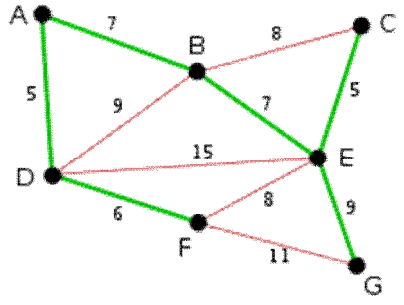
Наступна дуга DF довжиною 6 виділяється так само як і попередня.



Наступні найкоротші дуги AB і BE обидві з довжиною 7. AB вибрана довільно і виділена на малюнку. Дуга BD виділена червоним, тому що цей шлях вже існує, отже воно створить цикл.



Процес продовжує виділяти наступні найменші дуги, BE з довжиною 7 і багато інших арок виділені червоним на цьому етапі, тому що створюють цикли.



Процес закінчується на дузі EG довжиною 9, мінімальне остове дерево знайдене.

Алгоритм Прима.

Цей алгоритм названий на честь американського математика Роберта Прима (Robert Prim), який відкрив цей алгоритм у 1957 р. Втім, ще в 1930 р. цей алгоритм був відкритий чеським математиком Войтеком Ярніком (Vojtěch Jarník). Крім того, Едгар Дейкстра (Edsger Dijkstra) в 1959 р. також винайшов цей алгоритм, незалежно від них.

Максимальне остовне дерево

Даний зважений неорієнтований граф з вершинами і ребрами. Потрібно знайти таке піддерево цього графа, яке б з'єднувало всі його вершини, і при цьому мало найбільшу можливу вагою (тобто сумою ваг ребер). Таке піддерево називається максимальним остовим деревом.

У природному постановці ця задача звучить наступним чином: є міст, і для кожної пари відома вартість з'єднання їх дорогою (або відомо, що з'єднати їх не можна). Потрібно з'єднати всі міста так, щоб можна було доїхати з будь-якого міста в інший, а при цьому вартість прокладання доріг була б максимальною.

Сам алгоритм має дуже простий вигляд. Шуканий максимальний кістяк будується поступово, додаванням до нього ребер по одному. Спочатку остов покладається складається з єдиної вершини (її можна вибрати довільно). Потім вибирається ребро максимальної ваги, що виходить з цієї вершини, і додається в максимальне остове дерево. Після цього остов містить уже дві вершини, і тепер шукається і додається ребро максимальної ваги, що має один кінець в одній з двох обраних вершин, а інший - навпаки, у всіх інших, крім цих двох. І так далі, тобто щоразу шукається максимальне по вазі ребро, один кінець якого - вже взята в остов вершина, а інший кінець - ще не взята, і це ребро додається в остов (якщо таких ребер кілька, можна взяти будь-яке). Цей процес повторюється до тих пір, поки остов не стане містити всі вершини (або, що те ж саме, ребро).

У результаті буде побудований остов, що є максимальним . Якщо граф був спочатку не зв'язний, то остов знайдений не буде (кількість вибраних ребер залишиться менше).

Приклад пошуку максимального остового дерева за алгоритмом Прима.

1. Вибираємо початкову вершину a і ребро з максимальною вагою, що інцидентне вершині a – ai. Будуємо це ребро з інцидентними вершинами.

2. Вибираємо найбільше по вазі ребро, з тих що залишилися в графі – bi. Будуємо його з індидентними вершинами.

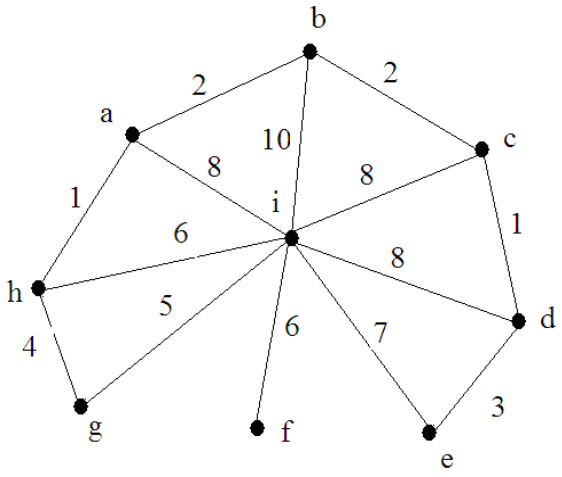


Рис 1. Заданий граф

3. Вибираємо наступне по величині ребро з найбільшою вагою і перевіряємо умову: якщо ребро не створює замкнутого циклу, то будуємо його , в іншому випадку виключаємо його з розгляду (така перевірка здійснюється на всіх інших кроках. Отже вибираємо ребро – ic і будуємо його.

4. Вибираємо ребро id і будуємо його.

5. Вибираємо ребро ie і будуємо його

6. Вибираємо ребро if і будуємо його

7. Вибираємо ребро ih і будуємо його

8. Вибираємо ребро ig і будуємо його.

9. Оскільки при подальшому виборі ребер виникає цикл, то завершуємо побудову максимального остового дерева і підраховуємо сумарний шлях: 58.

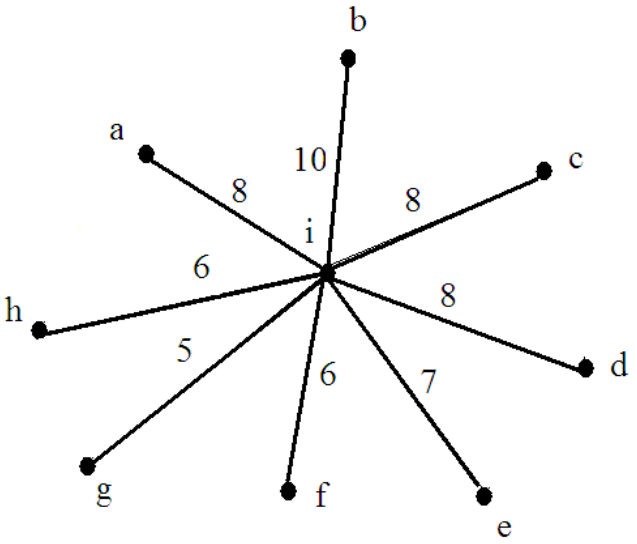


Рис 2. Максимальне остове дерево.

3. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Які Ви знаєте варіанти представлення графів?
2. Яка обчислювальна складність алгоритмів побудови остових дерев?
3. Навести реальні проблеми для вирішення яких можна застосувати алгоритми побудови остових дерев.

4. ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Отримати у викладача індивідуальне завдання.
2. Підготувати програму для вирішення виданого завдання.
3. Запустити на покрокове виконання програму побудови мінімального покриваючого дерева і максимального покриваючого дерева.
4. Здійснити перевірки роботи програм з результатами розрахунків проведених вручну.
5. Зафіксувати результати роботи.
6. Оформити і захистити звіт.

5. ЗМІСТ ЗВІТУ

1. Короткий опис алгоритмів побудови мінімального і максимального покриваючих дерев.
2. Нарисувати граф з виділенням на ньому мінімального і максимального покриваючих дерев.
3. Розрахунки по знаходженню мінімального і максимального покриваючих дерев згідно з відомими алгоритмами.
4. Висновки по результатах перевірок..

І Н С Т Р У К Ц І Я №2

до лабораторної роботи

АЛГОРИТМ РІШЕННЯ ЗАДАЧІ ЛИСТОНОШІ

з курсу

“**Дискретні моделі в САПР**”

для базового напрямку “Комп’ютерні науки”

1. МЕТА РОБОТИ

Метою даної лабораторної роботи є вивчення алгоритмів рішення задачі листоноші.

2. КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

ЗАДАЧА ЛИСТОНОШІ. ОСНОВНІ ПОНЯТТЯ. ВЛАСТИВОСТІ.

Будь-який листоноша перед тим, як відправитись в дорогу повинен підібрати на пошті листи, що відносяться до його дільниці, потім він повинен рознести їх адресатам, що розмістились вздовж маршрута його проходження, і повернутись на пошту. Кожен листоноша, бажаючи втратити якомога менше сил, хотів би подолати свій маршрут найкоротшим шляхом. Загалом, задача листоноші полягає в тому, щоб пройти всі вулиці маршрута і повернутися в його початкову точку, мінімізуючи при цьому довжину пройденого шляху.

Перша публікація, присвячена рішенню подібної задачі, появилась в одному з китайських журналів, де вона й була названа задачею листоноші. Очевидно, що така задача стоїть не тільки перед листоношею. Наприклад, міліціонер хотів би знати найбільш ефективний шлях патрулювання вулиць свого району, ремонтна бригада зацікавлена у виборі найкоротшого шляху переміщення по всіх дорогах.

Задача листоноші може бути сформульована в термінах теорії графів. Для цього побудуємо граф $G = (X , E)$, в якому кожна дуга відповідає вулиці в маршруті руху листоноші, а кожна вершина - стик двох вулиць. Ця задача являє собою задачу пошуку найкоротшого маршруту, який включає кожне ребро хоча б один раз і закінчується у початковій вершині руху.

Нехай S-початкова вершина маршруту і $a(i,j)>0$ - довжина ребра (i, j) . В графі на рис. 1

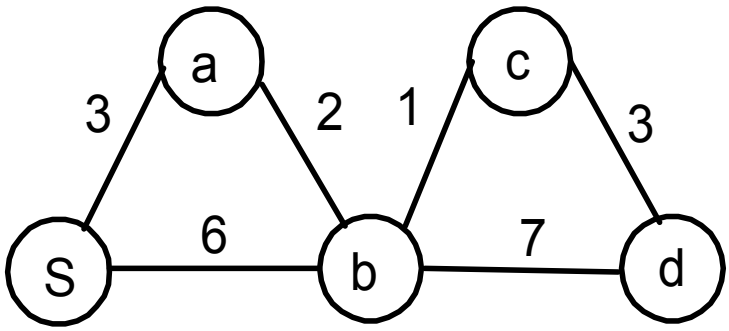


Рис. 1

існує декілька шляхів, по яким листоноша може обійти всі ребра і повернутись у вершину S. Наприклад :

Шлях 1: (S,a), (a,b), (b,c), (c,d), (d,b), (b,S)

Шлях 2: (S,a), (a,b), (b,d), (d,c), (c,b), (b,S)

Шлях 3: (S,b), (b,c), (c,d), (d,b), (d,a), (a,S)

Шлях 4: (S,b), (b,d), (d,c), (c,b), (b,a), (a,S)

В будь-який з чотирьох шляхів кожне ребро входить тільки один раз. Таким чином, загальна довжина кожного маршруту дорівнює

$$3+2+1+3+7+6=22.$$

Кращих маршрутів у листоноші не існує.

ЕЙЛЕРОВИЙ ЦИКЛ

Ейлеревим циклом в графі називається шлях, який починається і закінчується в тій самій вершині, при чому всі ребра графа проходяться тільки один раз.

Ейлеревим шляхом називається шлях, який починається в вершині А, а закінчується в вершині Б, і всі ребра проходяться лише по одному разу.

Граф, який включає в себе ейлерів цикл називається ейлеревим.

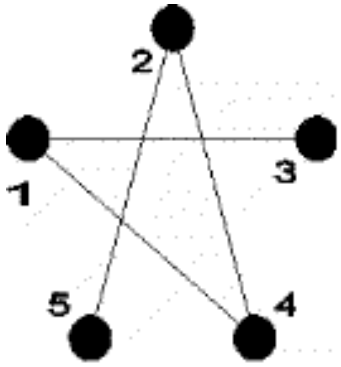


Рис.2. Ейлеровий шлях : 5, 2, 4, 1, 3

Теорема 1.

Якщо у графа всі вершини мають парну степінь, то цей граф Ейлерів, тобто має Ейлерів цикл.

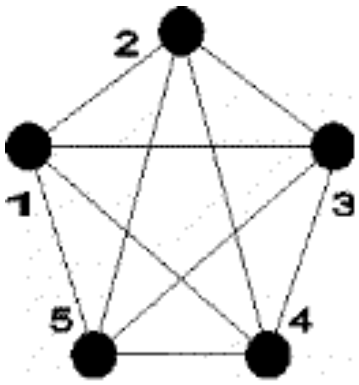
Доведення

Зв”язність графа слідує з визначення ейлеревого циклу. Ейлерів цикл містить кожне ребро і при цьому тільки один раз, тому кількість разів, яка приводить нас в вершину рівна

кількості виходів з вершини. Тобто , степінь кожної вершини повинна складатися з двох однакових складових: одна – результат підрахунків входів в вершину, інша – виходів з вершини.

Також вірне і протилежне твердження.

Якщо в графі існує Ейлерів цикл, то це означає, що в нього всі вершини мають парний степінь.



Теорема 2 .

Якщо в графі існує Ейлерів шлях, то це означає, що в нього є строго дві непарні вершини. При чому шлях починається в одній з них, а закінчується в іншій.

Доведення

Якщо шлях починається в вершині А, а закінчується в вершині Б, то А та Б-непарні, навіть, якщо шлях неодноразово проходив через А та Б.В будь-яку іншу вершину графа шлях повинен був привести і вивести з неї, тобто всі інші вершини повинні бути парними. Справедливе й зворотні твердження. Якщо граф зв”язний і має тільки дві непарні вершини, то він містить Ейлерів шлях, який починається в одній непарній вершині, а закінчується в іншій.

КРИТЕРІЙ ІСНУВАННЯ ЕЙЛЕРОВОГО ШЛЯХУ

Якщо G (ейлерів граф, то будь-який його ейлерів цикл неєдиний і відрізняється від інших ейлерових циклів графа G принаймні або зміною початкової вершини і/або зміною порядку проходження.

Для знаходження деякого ейлерового циклу в ейлеровому графі G можна застосувати так званий алгоритм Фльорі. Фіксуємо довільну початкову вершину циклу. На кожному кроці процедури до шуканого циклу обираємо (доки це можливо) те ребро, після вилучення якого граф не розіб’ється на дві нетривіальні зв’язні компоненти. Кожне обране ребро вилучаємо з G.

Процедура завершується, коли всі ребра буде вичерпано. Неважко обгрунтувати, що сформульований алгоритм будує ейлерів цикл графа G.

Розглянемо граф, який представлений на рис. 2. Очевидно, що на ньому відсутні маршрути листоноші, в яких дуга *(b,c)* входила б тільки один раз, тобто відсутній ейлеровий маршрут.

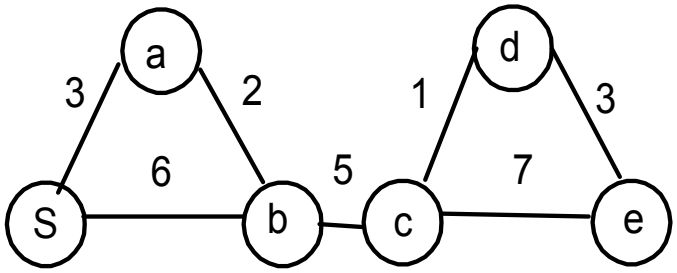


Рис. 3. Граф без Ейлерового циклу

Можливим оптимальним маршрутом (маршрутом найменшої довжини) на цьому графі є :

(S,a), (a,b), (b,c), (c,d), (d,e), (e,c), (c,b), (b,S)

Загальна довжина цього маршрута дорівнює :3+2+5+1+3+7+5+6=32

Виходячи з постановки задачі листоноші:

1. Очевидно, що в незв'язному графі (в графі, який має декілька компонент) не існує маршруту листоноші (не говорячи про існування ейлерового маршруту).

2. 2. Очевидно, що кількість приходів листоноші в деяку вершину повинна бути рівною кількості виходів з цієї вершини. При цьому, якщо листоноша не проходить через деяке ребро більше одного разу, то вершина повинна бути інцидентна парній кількості ребер. Загальну кількість ребер інцидентних вершині x, назвемо *степінню вершини* і будемо позначати через *d(x)*. Якщо в графі *G* всі вершини мають парний (непарний) степінь, то граф називається *парним (непарним)*. В орієнтованому графі число дуг, які входять у вершину *x* називають *внутрішнею степінню вершини x* або *півстепінню входу* (позначається через $d^-(x)$), а які виходять - *зовнішню степінню вершини x* або *півстепінню виходу* (позначається $d^+(x)$. Якщо $d^-(x)=d^+(x)$ то граф називається *симетричним*

Допустимо, що ми знаємо оптимальний маршрут листоноші на графі *G*, який починається і закінчується в вершині "*S"*. Яким буде оптимальний маршрут якщо поміняти початкову точку на вершину "*t"*? Будь-який оптимальний маршрут *R*, який починається в вершині "*S"*, в кінцевому рахунку колись вперше проходить через вершину

"*t*". Назвемо цю частину маршруту R_1, а залишок R_2. Зауважимо, що R_1 починається в вершині "*S*" і закінчується в "*t*". В свою чергу R_2 починається в вершині "*t*" і закінчується в "*S*". Сформуємо новий маршрут *R'* , який складається з R_2 , а після нього R_1. Маршрут *R'* починається в вершині "*t*" і закінчується в вершині "*t*" має сумарну довжину таку ж як і маршрут *R*. Висновок : *R'=R* - оптимальний маршрут. Отже : *Теорема 1*. Сумарна довжина оптимального маршруту листоноші не залежить від того, яка з вершин цього маршруту буде вибрана за початкову.

ЗАДАЧА ЛИСТОНОШІ ДЛЯ НЕОРІЄНТОВАНОГО ГРАФА.

Задача листоноші для неорієнтованого графа *G(X,E)* - це задача для графа, в якому ребра можна проходити в будь-якому з двох напрямків.

Необхідно розгянути окремо наступні два випадки :

Випадок 1 : Граф *G* парний.

Випадок 2 : Граф *G* непарний.

Випадок 1 : Якщо граф парний, то оптимальним рішенням задачі є ейлеровий маршрут. В цьому випадку листоноша не повинен обходити більше одного разу будь-яку вулицю, в даному випадку ребро графа.

Як найти на графі *G* ейлеровий маршрут, в якому "*S*" - початкова вершина? Для цього необхідно пройти будь-яке ребро *(S,x)* інцидентне вершині "*S*", а потім ще невикористане ребро, інцидентне вершині "*x*". Кожен раз, коли листоноша приходить в деяку вершину, є невикористане ребро по якому листоноша покидає цю вершину. Дуги, по яким здійснений обхід, створюють цикл C_1 . Якщо в цикл C_1 ввійшли всі ребра графа *G,* то C_1 є ейлеровим маршрутом (оптимальним для задачі).

В іншому випадку треба створити цикл C_2, який складається з невикористаних ребер і який починається з невикористаного ребра. Створення циклів C_3, C_4,..., продовжується доти, доки не будуть використані всі ребра графа. Далі треба об'єднати цикли C_1, C_2, C_3,... в один цикл *C*, який містить всі ребра графа *G*. В цикл *C* кожне ребро графа входить лише один раз, і тому він є оптимальним рішенням задачі листоноші.Два цикла C_1 і C_2 можуть бути з'єднані тільки тоді, коли вони мають спільну вершину "*x*".

Для з'єднання двох таких циклів необхідно вибрати в якості початкового довільне ребро циклу C_1 і рухатися вздовж його ребер до вершини "*x*". Далі потрібно пройти всі ребра циклу C_2 і повернутися у вершину "*x*". На кінець, потрібно продовжити прохід ребер цикла C_1 до повернення назад до початкового ребра. Пройдений маршрут є циклом, отриманим в результаті з'єднання циклів C_1 і C_2. Ця процедура може бути легко

розширена на випадок з'єднання довільної кількості циклів і може виконуватись до тих пір, поки не утвориться дві їх підмножини, які не мають загальних вершин.

Приклад 1. Необхідно знайти оптимальний маршрут листоноші на парному графі, який зображений на рис.3. Почавши з вершини "*S"*, здійснемо обхід ребер графа по периметру до повернення в вершину "*S"*. В результаті обходу цих ребер буде сформований цикл C_1= *(s, b), (b, c), (c,f), (f, i), (i, h), (h, g), (g, d), (d, s)* (Рис.4). Ребра циклу C_1 будемо вважати використаними. Далі, починаючи з невикористаного ребра *(d, b)*, обійдемо невикористані ребра трикутного циклу C_2=*(d, b), (d, e), (e, d)*, після чого ребра циклу C_2 також будемо вважати використаними. Накінець, починаючи з невикористаного ребра *(e, f)*, обійдемо ребра нижнього трикутного циклу C_3=*(e, f), (f, h), (h, e)*. Тепер всі ребра графа використані.

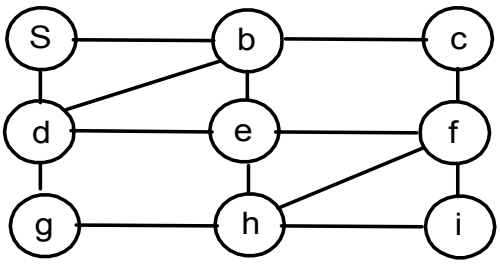


Рис.3

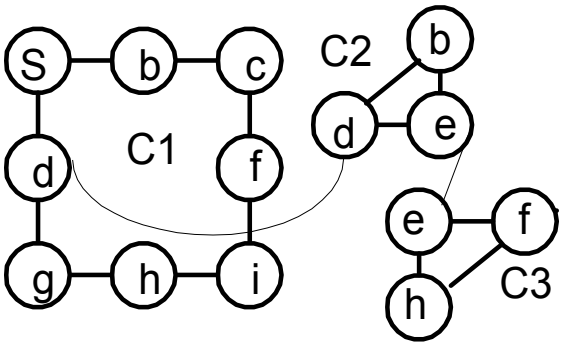


Рис.4

З'єднаємо знайдені вище цикли наступним чином: цикл C_2 вставимо між ребрами *(g, d)* і *(d, s)* циклу C_1; а цикл C_3 - між ребрами *(b, e), (e, d)* цикла C_2. В результаті отримаємо цикл *C=(s, b), (b, c), (c, f), (f, i), (i, h), (h, g), (g, d), (d, b), (b, e), (e, f), (f, h), (h, e), (e, d), (d, s)*.

Очевидно, що *C* - оптимальне рішення задачі листоноші для розглянутого графа, так як *C* містить кожне ребро рівно один раз, до того ж, починається і закінчується у вершині "*S*" Відмітимо, що в цьому прикладі не враховувались довжини ребер.

Випадок 2 : Граф є непарним. В будь-якому маршруті листоноші число входів у вершину рівне числу виходів з неї. Тому, якщо вершина "*x"*, має непарну степінь, то по

крайній мірі одне ребро, інцидентне вершині "*x*", буде обходитись вдруге. Нехай *f(i,j)* - число додаткових проходжень листоношею ребра *(i,j)*. Значить ребро *(i,j)* обходиться *f(i,j)+1* раз (число *f(i,j)+1* - невід'ємне, ціле число).

Побудуємо новий граф *G*=(X,E*)*, в якому ребро *(i,j)* графа *G* повторюється *f(i,j)+1* раз. Очевидно, що ейлеровий маршрут в графі *G** відповідає маршруту в графі *G* . Листоноша прагне вибрати значення змінних *f (i,j)* такими, щоб :

а) граф *G** був парним ;

б) $\sum a(i,j) f(i,j)$ - загальна довжина вдруге пройдених ребер - була мінімальна.

Якщо вершина "*x*" в графі *G* має непарну степінь, то для того, щоб в графі *G** вершина "*x*" мала парну степінь, листоноша повинен вдруге обійти парне число ребер, інцидентних даній вершині. Аналогічно, якщо вершина "*x*" в графі *G* має парну степінь, то для того, щоб в графі *G** вершина "*x*" мала парну степінь, листоноша повинен ще раз обійти парне число ребер, інцидентних цій вершині (нуль є парне число). Якщо ми до кінця прослідкуємо ланцюг ребер, що починається у вершині з непарною степінню, то він обов'язково повинен закінчитись в іншій вершині з непарною степінню. Таким чином, ребра, що обходяться вдруге, створюють ланцюги, початком і кінцем яких є вершини з непарною степінню. Тому листоноша повинен :

1) вирішити, які вершини з непарною степінню будуть з'єднані ланцюгом ребер, що обходяться вдруге.

2) знати точний склад кожного такого ланцюга.

Листоноша може за допомогою любого з алгоритмів побудови найкоротшого шляху визначити на графі *G* найкоротший шлях між кожною парою вершин з непарною степінню.

Для визначення пари вершин з непарною степінню, які повинні бути з'єднані ланцюгом ребер, що обходяться вдруге, листоноша може поступити слідуючим чином. Побудувати граф $G' = (X', E')$, множина вершин якого складається з усіх вершин з непарною степінню, а множина ребер з'єднує кожну пару вершин. Присвоїти кожному ребру графа вагу, рівну деякому дуже великому числу за обрахунком довжини найкоротшого шляху між відповідними вершинами графа *G*.

Далі на графі G' треба побудувати паросполучення з мінімальною вагою. Ці паросполучення з'єднують на графі *G* пари вершин з непарними степенями. Листоноша вдруге повинен обійти ребра, що складають ланцюг найменшої довжини і які з'єднують пару інцидентних паросполученню вершин. Оскільки ці паросполучення мають найменшу загальну вагу, то отриманий в результаті маршрут листоноші повинен мати мінімальну загальну довжину.

Приклад 2. Знайдемо оптимальний маршрут листоноші на неорієнтованому графі, який зображений на рис. 5. Вершини графа *a, c, d* і *f* мають непарні степені. Довжини найкоротших ланцюжків між всіма парами вершин з непарними степенями задаються такою таблицею:

Таблиця 1 Матриця шляхів найменшої довжини.

Паросполучення	Вага
(a, c), (d, f)	4+3=7
(*a, d*), (*c, f*)	2+4=6
(*a, f*), (*c, d*)	3+2=5

	a	*b*	*c*	*d*	*e*	*f*
a	0	1	4	2	4	3
b	1	0	5	3	5	2
c	4	5	0	2	7	4
d	2	3	2	0	6	3
e	4	5	7	6	0	3
f	3	2	4	3	3	0

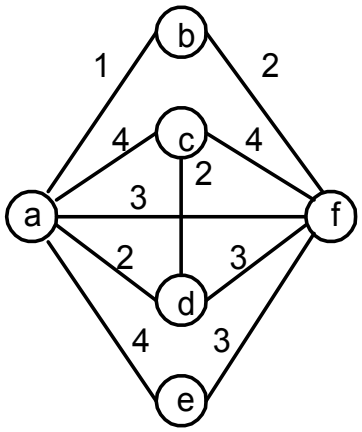


Рис. 5

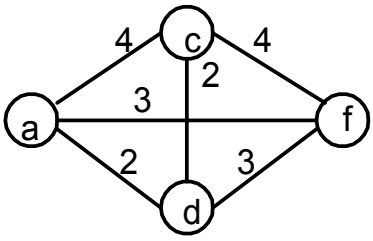


Рис. 6. Граф *G‘*.

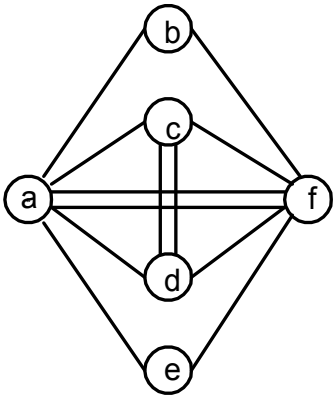


Рис.7. Граф *G**.

Потрібно перевірити ці величини за допомогою алгоритмів Флойда або Данціга.

Сформуємо граф *G’*, що показаний на рис. 6. Вершинами графа *G’* є вершини *a, c, d* і *f* графа *G*, що мають непарну степінь. *G’* є повним графом, оскільки в нього є всі можливі ребра. На щастя, оскільки в *G’* вершин небагато, нам легше отримати паросполучення з мінімальною вагою на графі *G’* за допомогою перебору, ніж використовуючи алгоритм знаходження паросполучення з максимальною вагою. В наведеному на рис.6. графі можливі три паросполучення , що вкзані в табл. 1. Відповідно, паросполучення з мінімальною вагою є (*a*, *f*),(*c*, *d*). Таким чином, листоноші потрібно вибирати для повторного обходу найкоротший шлях від *a* до *f*, що є ребром (*a, f*) в графі *G*, і найкоротший шлях від *c* до *d* , що є ребром (*c, d*) в цьому ж графі *G*. На рис. 7. показаний граф *G**, в якому ребра (*a*, *f*) і (*c*, *d*) один раз продубльовані. Всі вершини в графі *G** мають парну степінь, і оптимальний маршрут листоноші на графі *G* (рис. 5.) відповідає ейлеровому маршруту в графі *G** (рис. 7). Метод рішення задачі листоноші,описаний вище для випадку А, може бути застосований до графа *G**. В розглянутому прикладі оптимальним маршрутом листоноші є шлях (*a, b*), (*b, f*), (*f, e*), (*e, a*), (*a, c*), (*c, f*), (*f, d*), (*d, c*), (*c, d*), (*d, a*), (*a, f*), (*f, a*), в якому кожне ребро графа *G** обходиться лише один раз, а кожне ребро в графі *G* обходиться принаймі один раз. В графі *G* повторно обходяться тільки ребра (*a, f*) і (*c, d*). Загальна довжина цього маршруту дорівнює 34 одиниці, і вона на 5 одиниць більша, ніж сума довжин ребер графа *G*.

Відмітимо, що якщо б використовувався алгоритм паросполучення з максимальною вагою, то вагу кожного ребра в графі *G’* потрібно було б прийняти рівним деякому великому числу, скажімо M, мінус довжина найкоротшого шляху між двома відповідними кінцевими точками в графі *G*. Таким чином, паросполучення (*a, c*), (*d, f*) мало б загальну вагу, рівну (M-4)+(M-3)=2M-7. Паросполучення (*a, d*), (*c, f*) мало б загальну вагу, рівну (M-2)+(M-4)=2M-6. Паросполучення (*a, f*), (*c, d*) мало б загальну вагу, рівну (M-3)+(M-2)=2M-5 і було б вибране в якості паросполучення з максимальною вагою. Використання великого числа M просто може розглядатися як спосіб перетворення задачі побудови паросполучення з мінімальною вагою, що покриває всі вершини графа, в задачу побудови паросполучення з максимальною вагою.

3. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Постановка задачі листоноші.
2. Задача листоноші для орієнтованого і змішаного графа.
3. Алгоритм рішення задачі листоноші для змішаного графа.
4. Дати визначення понять: матриці зв’язності і ейлерового маршруту.

4. ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Отримати у викладача індивідуальне завдання.
2. Підготувати програму для вирішення виданого завдання.
3. Запустити на виконання програму, що розв’язує задачу листоноші.
4. Проглянути результат роботи програми. Результат може бути позитивний (шлях знайдено) або негативний (шлях відсутній).
5. У випадку, коли шлях знайдено (не знайдено), необхідно модифікувати граф, коректуючи два або три зв’язки таким чином, щоб знайти граф, на якому задача листоноші не розв’язується (розв’язується).
6. Здійснити перевірки роботи програм з результатами розрахунків, проведених вручну.
7. Зафіксувати результати роботи.
8. Оформити і захистити звіт.

5. ЗМІСТ ЗВІТУ

1. Опис одного з алгоритмів вирішення задачі листоноші.
2. Нарисувати графи і матриці зв’язності для двох можливих результатів вирішення задачі.
3. Кортеж вершин графа для якого вирішується задача листоноші.
4. Висновки про результати ручного і машинного рішення задачі.

І Н С Т Р У К Ц І Я №3

до лабораторної роботи

ПОТОКОВІ АЛГОРИТМИ

з курсу

“Дискретні моделі в САПР”

для базового напрямку “Комп’ютерні науки”

1. МЕТА РОБОТИ

Метою даної лабораторної роботи є вивчення потокових алгоритмів.

2. КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

ВСТУП.

Останнім часом значно зросла зацікавленість учених та практиків потоковими моделями. Це пов’язано із впровадженням та активним розвитком різноманітних територіально розподілених систем: трубопровідних, транспортних, телекомунікаційних та ін. Основою таких систем є певна мережа (мережа трубопроводів, доріг, каналів зв’язку тощо), в якій циркулюють певні потоки (потоки речовин, транспорту, даних тощо), тому задачі, які доводиться розв’язувати при проектуванні та експлуатації систем з мережною структурою, часто зводяться до розробки математичних моделей розподілу потоків та постановки і розв’язання відповідних оптимізаційних задач.

Відомі моделі розподілу потоків у мережах базуються на поняттях теорії графів. Це пов’язано з тим, що граф дає можливість наочно відобразити структуру мережі, а параметри його вузлів і дуг – представити основними числовими характеристиками її елементів. Набір характеристик залежить від природи модельованої системи, а також характеру розв’язуваних задач, однак у потокових моделях їх, як правило, представляють такими параметрами, як зовнішній потік у вузлі, потік по дузі, пропускна здатність дуги, вартість передавання одиниці потоку по дузі тощо. Потокові задачі, як правило, зводяться до пошуку такого розподілу потоків у мережі, при якому б забезпечувався екстремум деякого критерію. При цьому мають враховуватися обмеження, що накладаються умовами збереження потоків у вузлах і неперевищення потоками пропускної здатності дуг. Типовими потоковими задачами є задача про потік мінімальної вартості, про максимальний потік, транспортна задача, задача про призначення та інші. Для їх розв’язання розроблено чимало ефективних алгоритмів, сформувався навіть відповідний напрям обчислювальних методів під назвою потокового програмування

ПОНЯТТЯ ПРО ПОТОКИ

Потік-визначає спосіб пересилання деяких об’єктів з одного пункту в інший. Розв’язання задачі потоку зводиться до таких основних підзадач:

- Максимізація сумарного обсягу перевезень
- Мінімізація вартості пересилань предметів з одного пункту в інший
- Мінімізація часу перевезень в заданій системі

Сітка - це граф, в якому кожній дузі приписана деяка пропускна здатність. Введемо позначення: c(x,y) - пропускна здатність дуги (x,y), a(x,y) - вартість переміщення одиниці потоку по дузі (x,y), T(x,y) - час проходження потоку, k(x,y) - коефіцієнт підсилення потоку в дузі (x,y).

Припустимо, що є граф, в якому деяка кількість одиниць потоку проходить від джерела до стоку і для кожної одиниці потоку відомий маршрут руху. Назвемо кількість одиниць, що проходять по дузі (x,y), потоком в даній дузі. Будемо потік в дузі(x,y) позначати через f(x,y) вочевидь $0 \leq f(x,y) \leq c(x,y)$. Дуги графа можна віднести до трьох різних категорій:

1) дуги, в яких потік не може ні збільшуватись, ні зменшуватись (множина таких дуг позначається через - N);
2) дуги, в яких потік може збільшуватись (множина таких дуг позначається через - I);
3) дуги, в яких потік може зменшуватись (множина таких дуг позначається через - R);

Наприклад, дуги, що мають нульову пропускну здатність або значну вартість проходження потоку, повинні належати множині N. Дуги, в яких потік менше пропускної здатності, повинні належати множині I. Дуги, по яких вже проходить деякий потік, повинні належати множині R. Дуги з множини I називають збільшуючими, а дуги з множини R - зменшуючими.

Будь-яка дуга графа належить хоча б одній з трьох введених множин - I, R або N. Можливо, що якась дуга належить як множині I, так і множині R. Це має місце в тому випадку, коли по дузі вже протікає деякий потік, який можна збільшувати чи зменшувати. Відповідні дуги називаються проміжними.

Позначимо через i(x,y) максимальну величину, на яку може бути збільшений потік в дузі (x,y). Відповідно позначимо через r(x,y) максимальну величину, на яку може бути зменшений потік в дузі (x,y). Очевидно,

$$\mathbf{i(x,y) = c(x,y) - f(x,y)\ ,\ a\ r(x,y) = f(x,y)}$$

Припустимо, що ми хочемо переслати додаткову кількість одиниць потоку з витоку в стік. Можливі декілька способів розв’язування даної задачі (якщо вона взагалі має рішення):

1) Цей спосіб міг би бути реалізований, якщо б ми знайшли шлях P з вершини “s” у вершину “t”, який би цілком складався із збільшуючих дуг (рис.1). Скільки в цьому випадку можна було б додатково переслати одиниць потоку з “s” в “t” по шляху P?

Оскільки i(x,y) являє собою максимально можливе збільшення потоку в дузі (x,y), то величина додаткового потоку з “s” в “t” по шляху P буде складати:

$$\min_{(x,y)\in P} \{i(x,y)\}$$

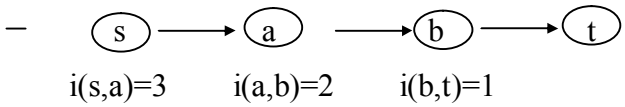


Рис.1. Ланцюг, що збільшує потік і включає лише прямі дуги.

Для даних прикладу (рис.1) по шляху P можна переслати з “s” в “t” максимум одну додаткову одиницю потоку, оскільки:

$$\min\{i(s,a), i(a,b), i(b,t)\} = \min\{3, 2, 1\}=1$$

2) Цей спосіб міг би бути реалізований, якщо б ми знайшли шлях P з вершини “t” в вершину “s”, який цілком складався би із зменшуючих дуг (рис.2). При цьому можна було б зменшити потік в кожній дузі (x,y), що привело б до зменшення потоку з вершини “t” в вершину ”s”, тобто, до збільшення чистого потоку з вершини “s” в вершину “t”. На яку максимальну величину можна було б зменшити потік з витоку в стік по вказаному шляху? Оскільки в кожній дузі (x,y) шляху P потік можна зменшити на максимальну величину r(x,y), то максимальне зменшення потоку вздовж шляху P визначається величиною

$$\min_{(x,y)\in P} \{r(x,y)\}$$

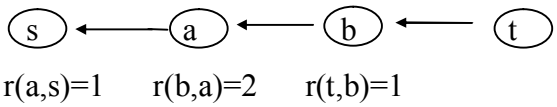


Рис.2. Ланцюг, що збільшує потік і включає лише зворотні дуги.

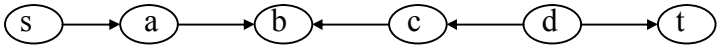
Для даних прикладу на рис.2 по шляху P можна переслати назад з “t” в “s” максимум одну одиницю потоку, оскільки:

$$\mathbf{\min\{r(t,b), r(b,a), r(a,s)\} = \min\{1, 2, 1\}=1}$$

3)Цей спосіб є комбінацією двох попередніх. Це означає, що необхідно знайти ланцюг, що з’єднує вершини “s” і “t”, дуги якого задовольняють наступним умовам:

- всі прямі дуги ланцюга, що мають напрям від “s” до “t”, належать множині I; - всі зворотні дуги ланцюга, що мають напрям від “t” до “s”, належать множині R.

Для прикладу розглянемо ланцюг C, що з’єднує вершини “s”,”t” і зображений на рис.3. Прямі дуги цього ланцюга (s,a), (a,b), (d,t), а зворотні - (c,b) і (d,c). Якщо кожна пряма дуга належить множині I, а кожна зворотна дуга - множині R, то вздовж розглянутого ланцюга можна переслати додатковий потік з “s” в “t”.



i(s,a)=4 i(a,b)=3 r(c,b)=5 r(d,c)=2 i(d,t)=3

Рис.3. Ланцюг, що збільшує потік і включає прямі і зворотні дуги.

Це здійснюється шляхом збільшення потоку в прямих дугах, що є збільшуючими, і зменшення потоку в зворотних дугах, що є зменшуючими. Максимальна величина додаткового потоку, який можна переслати вздовж відповідного ланцюга з “s” в “t”, визначається як мінімум з наступних двох величин:

$$\min \{i(x,y):(x,y) - \text{пряма дуга}\}$$

$$\min \{r(x,y):(x,y) - \text{зворотна дуга}\}$$

Мінімальна з двох вказаних величин називається максимальним збільшенням потоку по відповідному ланцюгу. Для прикладу на рис.3, маємо можливе збільшення потоку в прямих дугах i(s,a)=4, i(a,b)=3, i(d,t)=3. Мінімум з цих трьох величин рівний 3. Можливе зменшення потоку в зворотних дугах визначають величини r(c,d)=5, r(d,c)=2. Мінімум з цих двох величин рівний 2. Максимальне збільшення потоку по ланцюгу C дорівнює min(3,2)=2. Таким чином, в результаті збільшення на дві одиниці потоку в прямих дугах і зменшення на дві одиниці потоку в зворотних дугах по ланцюгу можна додатково переслати з “s” в “t” дві одиниці потоку.

Кожен ланцюг з “s” в “t” будь-якого з трьох розглянутих вище типів, по якому можуть бути додатково переслані одиниці потоку, називається збільшуючим ланцюгом.

ПОНЯТТЯ ПОТОКОВИХ АЛГОРИТМІВ

Поняття потокові алгоритми включає в себе ряд алгоритмів.

1) Алгоритм пошуку збільшую чого ланцюга.

Основна ідея алгоритму: побудова дерева, що росте з вершини “s” і складається з розфарбованих дуг, по яких з вершини “s” можуть пересилатись додаткові одиниці потоку.

В процесі виконання алгоритму можуть виникнути дві різні ситуації: а) стік “t” є розфарбованим???, тоді в побудованому з розфарбованих дуг дереві єдиний ланцюг з “s” в “t” є збільшуючим потік ланцюгом; б) стік “t” не вдається розфарбувати, що означає: що у вихідній сітці не існує збільшуючого ланцюга між “s” і “t” .

2) Алгоритм пошуку максимального потоку.

Основна задача алгоритму пошуку максимального потоку полягає в пошуку способів пересилання максимальної кількості одиниць потоку з витоку в стік при умові відсутності перевищення пропускних здатностей дуг вихідного графа. В основі алгоритму пошуку максимального потоку лежить наступна ідея: вибираємо початковий потік з витоку “s” в стік “t”, потім використовуємо алгоритм пошуку збільшуючого ланцюга. Цей алгоритм дозволяє знайти єдиний збільшуючий ланцюг з “s” в “t”, якщо той існує. Послідовність, в якій повинні розфарбовуватись вершини і дуги, конкретно не визначається. Тому можливі два варіанти розфарбування вершин і дуг: 1) вибирається яка-небудь вершина, а потім проводиться розфарбування максимальної кількості вершин; 2) проводиться розфарбування, виходячи з останньої розфарбованої вершини. Який саме спосіб розфарбування буде вибраний, буде залежати від характеру більш загальної задачі, тобто від алгоритму пошуку максимального потоку, який в якості підалгоритму використовує алгоритм пошуку збільшуючого ланцюга. Якщо пошук збільшуючого ланцюга вдалий, тобто знайдено збільшуючий ланцюг з “s” в “t”, то за допомогою алгоритму пошуку максимального потоку здійснюється максимально можливе збільшення потоку вздовж знайденого ланцюга. Потім повторюється пошук нового збільшуючого ланцюга і т.д. Виконання алгоритму завершується за скінчене число кроків, коли ланцюг, що збільшує потік, знайти не вдається: це означає, що біжучий потік з “s” в “t” є максимальним.

***3)* Алгоритм пошуку потоку мінімальної вартості.**

Дана задача полягає в організації пересилання з мінімальними витратами заданої кількості v одиниць потоку з витоку в стік в графі з заданими на дугах пропускними здатностями і вартостями проходження одної одиниці потоку.

4) Алгоритм дефекту.

Основна ідея алгоритму: рішення задачі про потік мінімальної вартості, але на відміну від попереднього алгоритму алгоритм дефекту вирішує задачу про потік мінімальної вартості у випадку, коли найменша кількість одиниць потоку, яка повинна протікати по дузі, більша або рівна 0 для всіх дуг.

5) Алгоритм пошуку динамічного потоку.

Попередні алгоритми використовували потоки, які задовольняли деяким умовам, які визначались заданими на дугах пропускними здатностями і вартостями. Даний алгоритм використовує сітки, дуги яких характеризуються ще одним показником - часом проходження потоку (кожна одиниця в цих потоках проходить з витоку в стік за час, що не перевищує заданий).

6) Потоки з підсиленням.

Попередні розгляди потоку показували, що потік не змінювався: одиниця ввійшла - одиниця вийшла. При проходженні потоку через дугу нові одиниці не створювались, але і старі не щезали. Потоки з підсиленням усувають припущення, згідно якого при проходженні по дугам потік залишається незмінним. Висувається припущення, що кількість одиниць в потоці, що проходить по дузі, може збільшуватись або зменшуватись. Точніше, вважають, що якщо в будь-яку дугу (x,y) в вершині “x” входить f(x,y) одиниць потоку, то з цієї дуги в вершині “y” вийде k(x,y)f(x,y) одиниць потоку. Можна вважають, що кожна одиниця потоку, що проходить по дузі (x,y), помножується на величину k(x,y) (яка називається підсиленням дуги (x,y).

ПРИКЛАД РОБОТИ АЛГОРИТМУ ПОШУКУ ЗБІЛЬШУЮЧОГО ЛАНЦЮГА

Як було сказано вище, алгоритм знаходить перший збільшуючий ланцюг, якщо той існує. Процедура розфарбування буде організована таким чином, що розфарбування наступної вершини буде проводитись виходячи з останньої розфарбованої вершини.

Нехай задано граф, що має наступний вигляд(рис.4):

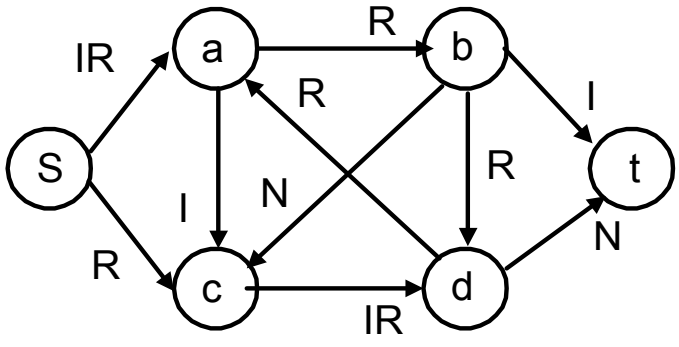


Рис.4.

(поряд з дугами графа вказані букви I,R,N, що визначають, до якої множини належать відповідні дуги)

Насамперед розфарбовується вершина “s” (початкова вершина). З вершини “s” може бути розфарбована вершина “a” і дуга (s,a), оскільки (s,a) $\in$I. Розфарбовані вершини запам’ятовуються в масиві збільшуючих ланцюгів в порядку їх роçфарбування. З вершини “a” не можуть бути розфарбовані вершина “b” і дуга (a,b), оскільки (a,b) $\notin$I. Вершина “c” і дуга (a,c) можуть бути зафарбовані, оскільки (a,c)$\in$I. Вершина “d” і дуга (d,a) можуть бути розфарбовані з вершини “a”, оскільки (d,a) $\in$ N. Це завершує процедуру зафарбовування з вершини “a”.

Далі будемо зафарбовувати вершини і дуги з вершини “b”. Вершини “a”, “c” і “d”, які вже розфарбовані, можна вже не розглядати. З вершини “b” може бути розфарбована вершина “t” і дуга (b,t), оскільки (b,t) є I.

Отже збільшуючим ланцюгом з “s” в “t” є (s,a), (a,c), (b,c), (b,t), в якому максимальне збільшення потоку по цьому рівне min{i(s,a), i(a,c),r(b,c), t(b,t)}

На цьому процедура розфарбування завершується, оскільки виявляється розфарбованою вершина “t”. Підграф, що складається з розфарбованих вершин і дуг, має вигляд зображений на рис. 5.

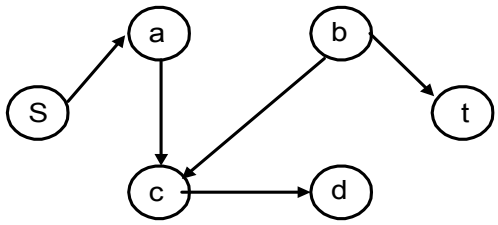


Рис.5.

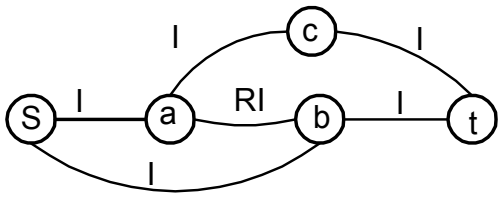
ПРИКЛАД РОБОТИ АЛГОРИТМУ ПОШУКУ МАКСИМАЛЬНОГО ПОТОКУ

Крок 1. Алгоритм починається з задання нульового потоку, тобто для всіх дуг (x,y) графа, що розглядається, задають f(x,y)=0.

Крок 2. Оскільки для всіх дуг f(x,y)=c(x,y), кожна дуга може бути віднесена до множини I, при цьому i(x,y)=c(x,y)-f(x,y)=c(x,y). Оскільки f(x,y)=0, тому на цьому кроці ні одну з дуг неможливо включити в множину R.

Крок 3. Використовується алгоритм пошуку збільшуючого ланцюга для визначення відповідного ланцюга з “s” в “t”.

З прикладу (рис. 6) видно, що у графі є декілька таких збільшуючих ланцюгів: sabt, sact, sbact, sbt.



	S	a	b	c	t
S	0	2	5	0	0
a	0	0	3	4	0
b	0	2	0	0	2
c	0	0	0	0	1
t	0	0	0	0	0

Рис.6

Допустимо, що у використаному алгоритмі на даному кроці формується ланцюг sabt. Максимальне збільшення потоку по цьому ланцюгу

$$\min\{i(s,a), i(a,b), i(b,t)=\min\{2,3,2\}=2.$$

Отже потік в цих трьох дугах збільшиться на дві одиниці. Після цього переходимо на крок 2.

Крок 2. Для нових значень потоку в дугах вхідного графа перераховуються величини i(x,y) і r(x,e); окрім цього, коректуються множини I і R:

$f(s,a)=2=c(s,a)$	$(s,a) \notin I$	$(s,a) \in R$, $r(s,a)=2$,
$f(a,b)=2<c(a,b)$	$(a,b) \in I$, $i(a,b)=1$	$(a,b) \in R$, $r(a,b)=2$,
$f(b,t)=2=c(b,t)$	$(b,t) \notin I$	$(b,t) \in R$, $r(b,t)=2$,
$f(s,b)=0<c(s,b)$	$(s,b) \in I$, $i(s,b)=3$	$(s,b) \notin R$,
$f(a,c)=0<c(a,c)$	$(a,c) \in I$, $i(a,c)=4$	$(a,c) \notin R$,
$f(c,t)=0<c(c,t)$	$(c,t) \in I$, $i(c,t)=1$	$(c,t) \notin R$.

Крок 3: Знову використовується алгоритм пошуку збільшуючого ланцюга для визначення відповідного ланцюга з "s" в "t". В даному випадку ланцюг, що збільшує потік, єдиний - це ланцюг (s,b), (b,a), (a,c), (c,t). Його і формує алгоритм. Максимально можливе збільшення потоку вздовж знайденого ланцюга дорівнює:

$$\mathbf{\min\{i(s,b), r(a,b), i(a,c)\}\ i(c,t) = \min\{5,2,4,1\} = 1}$$

Таким чином, з вершини "s" в вершину "t" може бути додатково пропущена одиниця потоку. При цьому значення потоку в кожній з трьох прямих дуг (s,b), (a,c) і (c,t), що належать знайденому ланцюгу, на одиницю збільшується, а значення потоку в зворотній дузі (a,b) цього ланцюга на одиницю зменшується. Тепер потоки в дугах графа, що розглядається, будуть мати наступні значення:

$$\mathbf{f(s,a)=2,\ f(a,b)=2,\ f(b,t)\}=2,\ f(s,b)=1,\ f(a,c)=1,\ f(c,t)=1.}$$

Кожна з цих трьох одиниць побудованого до цього моменту потоку проходить по наступним маршрутам:

1 одиниця проходить з "s" в "t" по шляху (s,a), (a,b), (b,t);

1 одиниця проходить з "s" в "t" по шляху (s,b), (b,t);

1 одиниця проходить з "s" в "t" по шляху (s,a), (a,c), (c,t)

Далі здійснюється повернення до кроку 2.

Крок 2. Для нових значень потоку в дугах вхідного графа перераховуються величини i(x,y) і r(x,e). Крім цього, коректується множини склад множин I і R:

f(s,a)=2=c(s,a) $(s,a)\notin I$ $(s,a)\notin R$, r(s,a)=2,

f(a,b)=1<c(a,b) $(a,b)\in I$, i(a,b)=2 $(a,b)\in R$, r(a,b)=1,

f(b,t)=2=c(b,t) $(b,t)\in I$ $(b,t)\in R$, r(b,t)=2,

f(s,b)=1<c(s,a) $(s,a)\in I$, i(s,b)=2 $(s,a)\in R$, r(s,a)=1,

f(a,c)=1<c(a,c) $(a,c)\in I$, i(a,c)=3 $(a,c)\in R$, r(a,c)=1,

f(c,t)=1<c(c,t) $i(c,t)\notin I$ $(c,t)\in R$, r(c,t)=1.

Крок 3. Знову використовується алгоритм пошуку збільшуючого ланцюга для визначення відповідного ланцюга з “s” в “t”. При цьому виявляється, що при біжучих значеннях потоку в дугах графа в ньому не існує ні одного збільшуючого ланцюга. В процесі виконання алгоритму у вихідному графі виявляються розфарбованими вершинами “s”, “b”, “a”, “c” (причому у вказаному порядку), однак вершина “t” не розфарбовується. Оскільки ланцюг, що збільшує потік, знайти не вдається, алгоритм пошуку максимального потоку завершує свою роботу. Останній з побудованих потоків є максимальним. Тобто в розглянутому прикладі з “s” в “t” може бути додатково пропущено 3 одиниці потоку.

Відмітимо, що останнє застосування алгоритму пошуку збільшуючого ланцюга приводить до розрізу з дуг (b,t),(c,t), що мають на одній кінцевій і одній незафарбованій вершини. Пропускна здатність цього розрізу рівна c(b,c)+c(c,t)=2+1=3, що співпадає з мінімально можливим числом одиниць потоку з “s” в “t”.

3. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Які Ви знаєте потокові алгоритми і яка схема їх вкладеності?
2. Які основні ідеї кожного з потокових алгоритмів?
3. Розкрити і зобразити блок-схему одного з потокових алгоритмів.

4. ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Отримати у викладача індивідуальне завдання.
2. Підготувати програму для вирішення виданого завдання.
3. Запустити на виконання програму, що вирішує задачу пошуку збільшуючого ланцюга.
4. Проглянути результат роботи програми. Результат роботи програми, що шукає збільшуючий ланцюг, може бути позитивний (стік є розфарбований) або негативний (стік не вдається розфарбувати).
5. У випадку, коли результат позитивний (або негативний) необхідно модифікувати граф (коректуючи два або три зв’язки), що дозволить знайти такий граф, на якому стік, відповідно, не вдається розфарбувати (розфарбовується).
6. Запустити на виконання програму, що вирішує задачу пошуку максимального потоку.
7. Проглянути результат роботи програми. Вибрати маршрут, який має максимальне збільшення потоку, за зразком, описаним в інструкції для роботи з учбовою програмою.
8. Зафіксувати результати роботи.
9. Оформити і захистити звіт.

5. ЗМІСТ ЗВІТУ

1. Описи і блок-схеми для алгоритмів пошуку збільшуючого ланцюга і пошуку максимального потоку.
2. Нарисувати графи і записати вхідні дані.
3. Записати проміжні дані і результати розрахунків (ланцюг і потік).
4. Висновки.

І Н С Т Р У К Ц І Я № 4

до лабораторної роботи

АЛГОРИТМ РІШЕННЯ ЗАДАЧІ КОМІВОЯЖЕРА

з курсу

“**Дискретні моделі в САПР**”

для базового напрямку “Комп’ютерні науки”

1. МЕТА РОБОТИ

Метою даної лабораторної роботи є вивчення і дослідження алгоритмів рішення задачі комівояжера.

2. КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

ФОРМУЛЮВАННЯ ТА ДЕЯКІ ВЛАСТИВОСТІ ЗАДАЧІ КОМІВОЯЖЕРА

Невідомо, коли проблему комівояжера було досліджено вперше. Однак, відома видана в 1832 році книжка з назвою «Комівояжер — як він має поводитись і що має робити для того, аби доставляти товар та мати успіх в своїх справах — поради старого Кур'єра», в якій описано проблему, але математичний апарат для її розв'язання не застосовується. Натомість, в ній запропоновано приклади маршрутів для деяких регіонів Німеччини та Швейцарії.

Раннім варіантом задачі може розглядатись англ. Icosian Game Вільяма Гамільтона 19 століття, яка полягала в тому, щоб знайти маршрути на графі з 20 вузлами. Перші згадки в якості математичної задачі на оптимізацію нелажать Карлу Менґеру (нім. Karl Menger), який сформулював її в математичному колоквіумі в 1930 році наступним чином:

> « Ми називаємо проблемою женця (оскільки це питання виникає в кожного листоноші, зокрема, її вирішують багато мандрівників) завдання віднайти найкоротший шлях між скінченною множиною місць, відстань між якими відома. »

Невдовзі з'явилась відома зараз назва задача мандруючого продавця (англ. Traveling Salesman Problem), яку запропонував Гаслер Вітні (англ. Hassler Whitney) з Принстонського Університету

Для можливості застосування математичного апарату для розв'язання проблеми, її слід представити у вигляді математичної моделі. Проблему комівояжера можна представити у вигляді моделі на графі, тобто, використовуючи вершини та ребра між ними. Таким чином, вершини графу відповідають містам, а ребра (i, j) між вершинами і та j сполучення між цими містами. У відповідність кожному ребру (i, j) можна зіставити вагу $c_{ij} \geq 0$, яку можна розуміти як, наприклад, відстань між містами, час або вартість подорожі. Маршрутом (також гамільтоновим маршрутом) називається маршрут на цьому графі до якого входить по одному разу кожна вершина графа. Задача полягає у відшуканні найкоротшого маршруту.

З метою спрощення задачі та гарантії існування маршруту, зазвичай вважається, що модельний граф задачі є повністю зв'язним, тобто, що між довільною парою вершин існує ребро. Це можна досягти тим, що в тих випадках, коли між окремими містами не існує сполучення, вводити ребра з максимальною вагою (довжиною, вартістю тощо). Через велику довжину таке ребро ніколи не потрапить до оптимального маршруту, якщо він існує.

Загальною задачею комівояжера називають задачу пошуку маршруту найменшої довжини.

Задачею комівояжера називають задачу пошуку гамільторового контура найменшої довжини.

Контур комівояжера, який має найменшу довжину, називають *оптимальним гамільтоновим контуром* він є оптимальним рішенням задачі комівояжера. Оптимальний маршрут комівояжера не обов`язково є гамільтоновим контуром.

Виникає питання, в яких випадках гамільтоновий контур є рішенням загальної задачі комівояжера? Відповідь дає наступна теорема.

Теорема 1. Якщо для кожної пари вершин (x,y) виконується умова:

$$a(x,y) \le a(x,z)+a(z,y), \quad \text{для всіх } z \ne x \ne y. \qquad (1)$$

то гамільтоновий контур є рішенням загальної задачі комівояжера на графі G (якщо рішення задачі взагалі існує).

Умова (1) говорить лише про те, що відстань безпосередньо від *x* до *y* ніколи не перевищує відстані від *x* до *y* через будь-яку іншу вершину *z*. Умову (1) називають нерівностю трикутника.

УМОВИ ІСНУВАННЯ ГАМІЛЬТОНОВОГО КОНТУРУ. НИЖНІ ГРАНИЦІ.

Рішенням задачі комівояжера є оптимальний гамільтоновий контур. Нажаль, не всі графи містять гамільтоновий контур. Отже перед тим, ніж перейти до пошуку оптимального гамільтонового контура потрібно довести факт його існування в даному графі.

Введемо ряд визначень, які в подальшому нам знадобляться. Граф називається *сильно зв`язаним,* якщо в ньому для будь-яких двох вершин "*x*" і "*y*" існує шлях від "*x*" до "*y*". Підмножина вершин X деякого графа називається *сильно зв`язаною,* якщо для будь-яких пар вершин $x \in X$ і $y \in X$ існує шлях з "*x*" в "*y*" і X не є підмножиною ніякої іншої

множини вершин, які володіють тими ж властивостями. Сформулюємо загальну теорему Гуйя-Урі.

Теорема 2. Якщо граф G(X,A) задовільняє умовам :

1) граф G силно зв'язаний ;

2) $d(x) \geq n$ для всіх $x \in X$, де d(x) - степінь вершини x, n - кількість вершин, то він містить гамільтоновий контур.

На практиці достатньо легко встановити, чи задовільняє деякий граф умовам (I.) і (II.) теореми 2.

Умова (1.) перевіряється застосуванням до кожної пари вершин алгоритма Флойда або алгоритма Данцига для побудови найкоротшого шляху. Ця умова виконується, якщо для кожної пари вершин в графі існує зв'язуючий їх шлях скінченної довжини.

Умова (2.) легко перевіряється підрахунком дуг, інцидентних кожній вершині графа.

Умови існування гамільтонового циклу на неорієнтованому графі сформульовані Квателем. Нехай, як і раніше, через n позначимо кількість вершин в графі. Пронумеруємо вершини так, щоб виконувалось відношення :

$$d(x_i) \leq d(x_i) \leq ... \leq d(x_n)$$

Теорема 3. Граф G=(X,.A) містить гамільтоновий цикл,якщо :

$$\mathbf{n \geq 3 \quad i \quad d(x_к) \leq k \leq 1/2\ n \rightarrow d(x_{n-к}) \geq n-k} \qquad (2)$$

Умову (2) легко перевірити. Необхідно впорядкувати вершини по зростанню їх степеней і перевірити, чи виконується умова для перших n/2 вершин.

Розглянемо тепер методи розрахунку нижніх границь довжини оптимальних гамільтонових контурів на орієнтованому графі G=(X,A). Якщо від довжини довільного гамільтонового контура відняти значення нижньої границі, то отримана різниця рівна максимальному значенню, на яке довжина гамільтонового контура може перевищувати довжину оптимального гамільтонового контура. Значення цієї вершини потрібно для оцінки різниці довжин знайденого і оптимального гамільтонового контура.

Для орграфа дуги, що входять в гамільтоновий контур, повинні володіти наступними двома властивостями:

1. Кожна вершина повинна бути інцидентна двом іншим дугам, одна з яких направлена до неї, а інша від неї;

2. Всі дуги повинні бути зв'язані. Дуги, які входять в будь-яку множину незв'язаних контурів, що містять всі вершини, володіють властивістю 1.

Будь-яка зв'язана множина дуг володіє властивістю 2. І тільки гамільтоновий контур володіє властивостями 1 і 2 одночасно.

Розглянемо сімейство F всіх підмножин множини A, які володіють властивістю 1. (Будемо вважати, що сімейство F не пусте). Зрозуміло, що кожен гамільтоновий контур входить в F. Таким чином, нижньою границею довжини оптимального гамільтонового контура є найменше значення суми довжин дуг, які утворюють підмножини в F. Позначимо через L_1 цю нижню границю. Нижню границю оптимального гамільтонового контура можна обчислити за наступним алгоритмом.

Крок 1. На основі графа G (рис.1.) будується граф G'=(X',A')(рис.2).

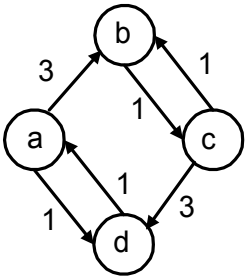


Рис.1.

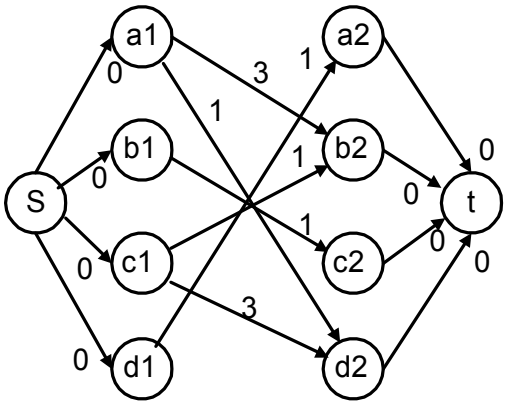


Рис.2.

Для цього кожній вершині x, що належить множині X ставиться у відповідність пара вершин $x_1 \in X'$ і $x_2 \in X'$. Кожній дузі $(x,y) \in A$ ставиться у відповідність проміжна дуга $(x_1,y_2) \in A'$. Нехай кожна проміжна дуга має пропускну здатність, рівну 1, вартість, рівну довжині відповідної дуги в A. Введемо вершину-джерела "s" і вершину-стік "t" і з'єднаємо вершину "s" з направленими з неї дугами (s, x_1) з кожною з вершин $x_1 \epsilon X'$. З'єднаємо вершину "t" з кожною з вершин $x_2 \epsilon X'$ направленими в неї дугами (x_2, t). Нехай кожна з дуг (s, x_1) і (x_2, t) для всіх x_1 і $x_2 \in X$ має пропуску здатність, рівну 1, і нульову вартість (рис.2).

Крок 2. Визначимо на графі G' максимально можливий потік з "s" в "t", що має мінімальну загальну вартість. Такий потік шукається за допомогою алгоритму побудови потоку мінімальної вартості. Нехай L_1 рівна вартості отриманого потоку. Так як пропускні здатності всіх дуг в графі G' рівні 1, отриманий потік складається з одиничних потоків, що протікають в дугах графа G' від вершини "s" до "t". Крім того, кожна з проміжних вершин інцидентна не більше ніж одній дузі, по якій протікає одиничний потік. Далі, кожній проміжній дузі в A' відповідає деяка дуга в A. Розглянемо множину всіх проміжних дуг, по

яким протікає одиничний потік. Нехай через M позначимо відповідну множину дуг в A. Дуги в множині M утворюють незв’язні контури в G. В противному випадку існувала би деяка вершина xєX, яка не була б інцидентна направленій до неї чи з неї дуги в M. Це відповідало тому, що в графі G’ у вершину “x_2”не входив би чи з вершини “x_1“ не виходив би ні один з одиничних потоків. В кожному з цих двох випадків сумарний потік з “s” в “t” складався б менше, ніж з |X| одиничних потоків. Але це неможливо, так як кожній підмножині в F відповідає потік в G’, рівний |X| одиниць, і крім того множина F не пуста. Оскільки не нульову вартість мають лише проміжні дуги, то сумарна вартість знайденого потоку рівна сумі довжин дуг, які входять у відповідні підмножини в F. Оскільки знайдений потік має найменшу сумарну вартість, то відповідна йому підмножина в F повинна мати мінімальну суму довжин дуг. Таким чином, L_1 є нижньою границею довжини оптимального намільтонового контура в графі G.

Якщо в граф G входять не орієнтовані дуги, то кожна неорієнтована дуга може бути замінена парою протилежно направлених дуг, які з’єднують цю ж пару вершин. Нехай кожна з цих направлених дуг має довжину, рівну довжині неорієнтованої дуги. Граф, який при цьому отримується, буде містити лише направлені дуги і для нього може бути застосований описаний вище метод обчислення L_1.

Якщо множина M утворює гамільтоновий контур, то ми знайшли не тільки нижню границю довжини оптимального гамільтонового контура, але й сам оптимальний контур.

МЕТОДИ РІШЕННЯ ЗАДАЧІ КОМІВОЯЖЕРА

Можна знайти точний розв'язок задачі комівояжера, тобто, обчислити довжини всіх можливих маршрутів та обрати маршрут з найменшою довжиною. Однак, навіть для невеликої кількості міст в такий спосіб задача практично нерозв'язна. Для простого варіанта, симетричної задачі з n містами, існує $(n - 1)! / 2$ можливих маршрутів, тобто, для 15 міст існує 43 мільярдів маршрутів та для 18 міст вже 177 більйонів. Те, як стрімко зростає тривалість обчислень можна показати в наступному прикладі. Якщо існував би пристрій, що знаходив би розв'язок для 30 міст за годину, то для для двох додаткових міст в тисячу раз більше часу; тобто, більш ніж 40 діб.

Відомо багато різних методів рішення задачі комівояжера. Серед них можна виділити методи розроблені Белмором і Немхаузером, Гарфинкелем і Немхаузером, Хелдом і Карном, Стекханом. Всі ці методи відносяться до одного з двох класів: а) методи рішення, які завжди приводять до знаходження оптимального рішення, але потребують для цього, в найгіршому випадку, недопустимо великої кількості операцій(метод гілок та границь); б) методи, які не завжди приводять до находження оптимального результату, але потребують

для цього допустимої великої кількості операцій (метод послідовного покращення рішення).

ЗАСТОСУВАННЯ МЕТОДУ ГІЛОК І ГРАНІЦЬ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ КОМІВОЯЖЕРА

Нехай розглядається граф $G= (X , A)$. Для отримання нижньої границі L_1 довжини гамільтонового контура найменшої довжини на графі G може бути використаний алгоритм побудови потоку мінімальної вартості.

Якщо отриманий за допомогою цього алгоритма оптимальний потік відповідає деякому контуру на графі G, то цей контур є оптимальним гамільтоновим контуром, і на цьому рішення задачі закінчується. Але існує достатньо велика ймовірність того, що оптимальний потік, отриманий для будь-якого графа, буде відповідати декільком незв'язним контурам. В цьому випадку довільно вибираємо один контур і позначаємо його через G_i, тоді $X_c = \{ x_1, x_2, \ldots, x_k \}$ множина вершин, що входить в нього.

В оптимальному рішенні комівояжер, виходячи з вершини $x_1 \in X_c$ переміщується або у вершину, яка належить множині X_c, або у вершину, яка не належить множині X_c. Якщо він прийшов у вершину $x \in X_c$, то з неї він знову переміщується або у вершину множини X_c, або у вершину, що не належить X_c і т. д. Таким чином, оптимальне рішення повинно бути представлено по крайній мірі в одному з наступних графів :

1. Графі G, з якого виключені всі дуги (x_1, y), $y \in X_c$;
2. Графі G, з якого виключені всі дуги (x_1, y), $y \notin X_c$ і дуги (x_2, z), $z \in X_c$;
3. Графі G, з якого виключені всі дуги (x_1, y), (x_2, y), $y \notin X_c$ і дуги (x_3, z), $z \in X_c$;
4. Графі G, з якого виключені всі дуги (x_1, y), (x_2, y), (x_3, y), $y \notin X_c$ і дуги (x_4, z), $z \in X_c$;

…

k. Графі G, з якого виключені всі дуги (x_1, y), (x_2, y), (x_3, y), (x_{k-1}, y), $y \notin X_c$ і дуги (x_k, z), $z \in X_c$.

Назвемо перераховані вище графи відповідно $G_1, G_2,\ldots G_k$. Отже довжина кожної дуги в графі G_i, i=1,2,…,k не менша довжини оптимального гамільтонового контура в графі G. Більше того, оптимальне рішення на графі G є також оптимальним рішенням по крайній мірі на одному з графів $G_1, G_2,\ldots,G_k$. Визначимо далі за допомогою алгоритма знаходження потоку мінімальної вартості нижню границю L_1 довжини оптимального гамільтонового контура для кожного з графів $G_1, G_2,\ldots,G_k$. Назвемо ці нижні границі відповідно $L_1(G_1), L_1(G_2), \ldots , L_1(G_k)$.

Якщо отриманий для деякого графа G_c оптимальний потік протікає по дугам, які утворюють гамільтоновий контур, то в подальшому не розглядаються графи G_j , для яких $L_1(G_j) \geq L_1(G_c)$, так як $L_1(G_c)$ є нижньою границею, що досягається. Таким чином деякі з графів $G_1, G_2,\ldots,G_k$ відкидаються з подальшого перегляду.

Для кожного з графів G_i, для яких $L_1(G_i) \le L_1(G_c)$, повторюється дана операція, що описана вище. При цьому граф G_i заміняється графами G_{i1}, G_{i2},… . На кожному з графів G_{ij} вираховується границя $L_1(G_{ij})$ довжини оптимального гамільтонового контура. Після цього, як і раніше, виключається з перегляду, як можна більшу кількість знову створених графів.

В кінці кінців один граф, в якому знайдений гамільтоновий контур мінімальної довжини, виключить з подальшого перегляду всі графи, що залишилися. Цей гамільтоновий контур повинен бути гамільтоновим контуром мінімальної довжину в вихідному графі G.

Приклад. Скористаємось методом гілок і границь для знаходження оптимального гамільтонового контура в графі, що зображений на рис. 3.

Граф G. ***Контури мінімальної вартості графа G.***

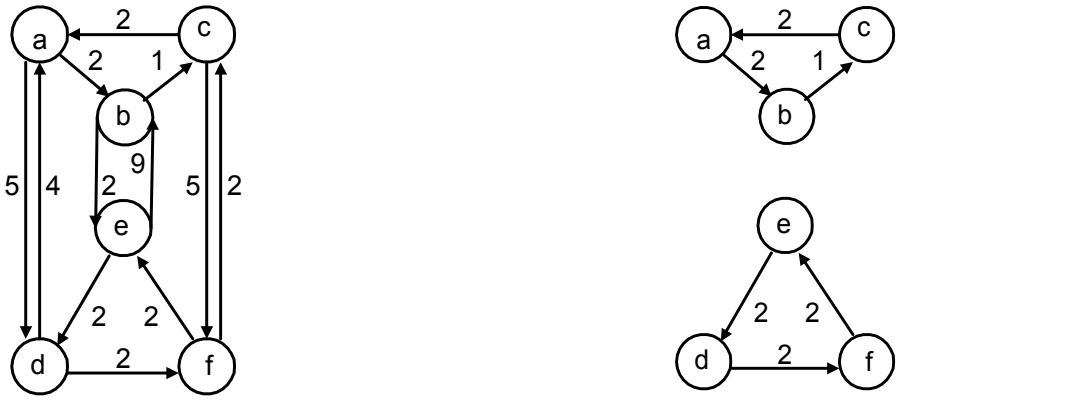


Рис.3.

Оскільки граф *G* має невелику розмірність, ми відразу можемо помітити, що в результаті використання алглритму знаходження потоку мінімальної потужності(вартості) на графі *G* отримується потік, що відповідає двом контурам (*a, b*), (*b, c*), (*c, a*) і (*f, e*), (*e, d*), (*d, f*). Загальна довжина цих двох контурів дорівнює 11. Таким чином, $L_1(G)$ =11. Використовуючи вершини Xc={a,b,c} першого контуру, ми можемо побудувати три графа Ga, Gb, Gc, зображених на рис.4.

Граф Ga ***Граф Gb*** ***Граф Gc***

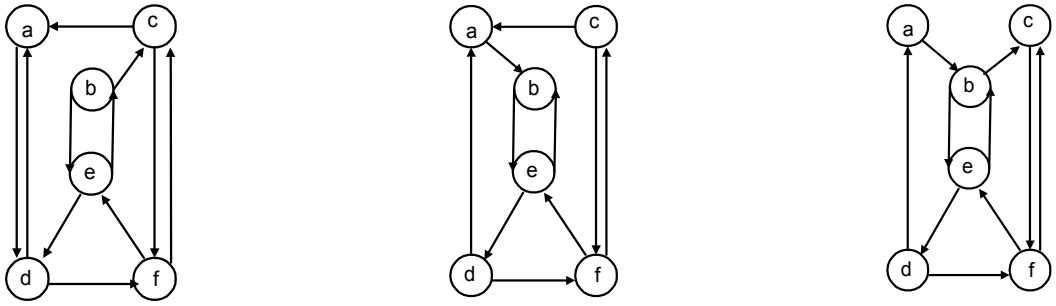


Рис.4а Рис.4б Рис.4в

Граф Ga отриманий шляхом виключення з графа G дуги, направленої з *a* в *b* або *c*, тобто дуги (*a, b*). Граф Gb отриманий шляхом виключення з графа G всіх дуг, що направлені з *a* в *d*,*e*,*f*, і всіх дуг, що направлені з *b* в *a* і *c*. Таким чином, граф Gb отримується шляхом виключення з графа *G* дуг (*a,d*) і (*b,c*). Граф Gc отримується шляхом виключення з графа *G* всіх дуг, що направлені з *a* і *b* в *d*, *e*, *f*, і всіх дуг, що направлені з *c* в *a* і *b*. Таким чином, граф Gc отриманий шляхом виключення з графа *G* дуг (*a, d*), (*b, e*) і (*c, a*).

Оскільки граф має невелику розмірність, ми можемо зразу побачити, що використання алгоритму знаходження потоку мінімальної вартості привело б до таких контурів і нижніх границь довжини оптимальних гамільтонових контурів в графах Ga, Gb, Gc.

Граф	Контур	Нижня границя
Ga	(*a, d*), (*d, f*), (*f, e*), (*e, b*), (*b, c*), (*c, a*)	$L_1(Ga) = 21$
Gb	(*a, b*), (*b, e*), (*e, d*), (*d, f*), (*f, c*), (*c, a*)	$L_1(Ga) = 12$
Gc	(*a, b*), (*b, c*), (*c, f*), (*f, e*), (*e, d*), (*d, a*)	$L_1(Ga) = 16$

Оскільки при Gc визначенні нижньої границі довжини оптимального гамільтонового контура в графі Gb одержаний гамільтоновий контур довжиною 12, то графи Ga і Gc можуть в подальшому не розглядатися (дійсно $L_1(Gb) < L_1(Gc) < L_1(Ga)$. Відповідно, гамільтоновий контур (*a, b*), (*b, e*), (*e, d*), (*d, f*), (*f, c*), (*c, a*) є гамільтоновим контуром в початковому графі *G*.

ЗАСТОСУВАННЯ МЕТОДУ ПОСТУПОВОГО ПОКРАЩЕННЯ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ КОМІВОЯЖЕРА

Метод поступового покращення рішення задачі комівояжера використовується для знаходження в графі G близького до оптимального (а

інколи і оптимального) гамільтонового контура і полягає в наступному. Спочатку береться деякий гамільтоновий контур. Нехай x_1, x_2, ..., x_n визначає послідовність , в якій в цьому контурі обходяться вершини графа G.

Для i=1, 2,..., n-1 і j=i+1,...,n визначити, чи зменшується довжина гамільтонового контура при перестановці в заданій вище послідовності обходу вершин пари вершин x_i і x_j. Якщо це приводить до зменшення довжини гамільтонового контура, то внести таку перестановку в порядок обходу вершин графа G. Повторювати цей процес до тих пір, поки за допомогою попарних перестановок досягається зменшення довжини гамільтонового контура.

Перестановка вершин “i” і “j” в дійсності означає заміну дуг :

$$(x_{i-1}, x_i), (x_i, x_{i+1}), (x_{j-1}, x_j), (x_j, x_{j+1})$$

дугами :

$$(x_{i-1}, x_j), (x_j, x_{i+1}), (x_{j-1}, x_i), (x_i, x_{j+1})$$

Процес попарних перестановок повинен бути скінченним, так як :

- існує тільки скінченне число різних методів обходу n вершин графа (різних перестановок) ;
- кожен раз формується нова послідовність обходу, відмінна від попередніх.

При цьому може бути отриманий новий гамільтоновий контур, який має строго меншу довжину, ніж попередній.

Потрібно зауважити, що при використанні метода покращення рішень результуючий гамільтоновий контур залежить від того, яким був вибраний початковий гамільтоновий контур.

Вдосконалення метода послідовного покращення рішення можна знайти в роботах Стекхана, а також Лина і Кернінга.

3. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Які ви знаєте алгоритми рішення задачі комівояжера?
2. Ідея рішення задачі комівояжера методом гілок та границь.
3. Ідея рішення задачі комівояжера методом послідовного покращення рішення.

4. ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Отримати у викладача індивідуальне завдання.
2. Підготувати програму для вирішення виданого завдання.
3. Запустити на виконання програму відповідного методу.
4. Проглянути результат роботи програм. Результат роботи може бути позитивним (шлях знайдено) або негативним (шлях відсутній).
5. У випадку, коли шлях знайдено (не знайдено), необхідно модифікувати граф, коректуючи два або три зв’язки, щоб знайти такий граф, на якому задача комівояжера не вирішується (вирішується).
6. Порівняти результати, отримані за допомогою різних алгоритмів і зробити висновок.
7. Зафіксувати результати роботи у викладача.
8. Оформити і захистити звіт.

5. ЗМІСТ ЗВІТУ

1. Опис двох алгоритмів рішення задачі комівояжера.
2. Нарисувати графи і вхідні дані, що їх описують для двох результатів розрахунку.
3. Результати розрахунків (гамільтоновий контур).
4. Висновки по результатах розрахунків.

1.

І Н С Т Р У К Ц І Я № 5

до лабораторної роботи

ІЗОМОРФІЗМ ГРАФІВ

з курсу

“**Дискретні моделі в САПР**”

для базового напрямку “Комп’ютерні науки”

1. МЕТА РОБОТИ

Метою лабораторної роботи є вивчення і дослідження основних підходів до встановлення ізоморфізму графів.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ

Теорія графів дає простий, доступний і потужній інструмент побудови моделей і рішення задач впорядкування взаємозвязаних обєктів. Нині є багато проблем де необхідно дослідити деякі складні системи з допомогою впорядкування їх елементів. До таких проблем відносяться і задачі ідентифікації в електричних схемах, в авіації, в органічній хімії і т.д. Вирішення таких проблем досягається з допомогою встановлення ізоморфізму графів.

Два графа G=(X,U,P) і G'=(X',U',P') називаються *ізоморфними*, якщо між їх вершинами, а також між їхніми ребрами можна встановити взаємно однозначне співвідношення X <-> X', U <-> U', що зберігає інцидентність, тобто таке, що для всякої пари (x,y)єX ребра u є U, що з'єднує їх, обов'язково існує пара (x',y') є X' і ребро u' є U', що з'єднує їх, і навпаки. Тут P - предикат, інцидентор графа G. Зауважимо, що відношення ізоморфізму графів рефлексивне, симетричне і транзитивне, тобто представляє собою еквівалентність.

На даний час існує досить детальна класифікація розроблених методів рішення такого типу задач /1/. Розглядаючи комбінаторно-логічну природу вказаної задачі можна всі роботи в цьому напрямку розділити на дві групи:

1) рішення теоретичної задачі встановлення ізоморфізму простих графів;
2) розробка наближених методів, які найбільш повно враховують обмеження і специфіку задачі з застосуванням характерних ознак об’єкту дослідження.

До першої групи відносяться алгоритми: повного перебору і почергового “підвішування” графів за вершини.

а) Одним з найпростіших з точки зору програмної реалізації, **є алгоритм перевірки ізоморфізму графів повним перебором(можливої перенумерації вершин),** але складність цього алгоритму є факторіальною.

б) **Почергове “підвішування” графів за вершини (всі ребра зрівноважені).**

Суть цього алгоритму полягає в знаходженні однакових “підвішаних” графів (за довільні вершини), ізоморфність яких визначаємо. При чому в одному з графів почергово змінюється вершина за яку він “підвішується”. Ізоморфізм графів визначається по їх матрицях суміжності, які формуються по однотипних правилах:

- індекс в матриці вершини за яку закріплений (“підвішаний”) граф рівний одиниці;
- кортеж вершин в матриці визначається рівнями сусідів;
- кортеж вершин в межах кожного рівня сусідів визначається степінню вершини, а також кількістю ребер над нею і нижче її.

Для графа G(рис.1) представлені різні варіанти його “підвіски” рис.2

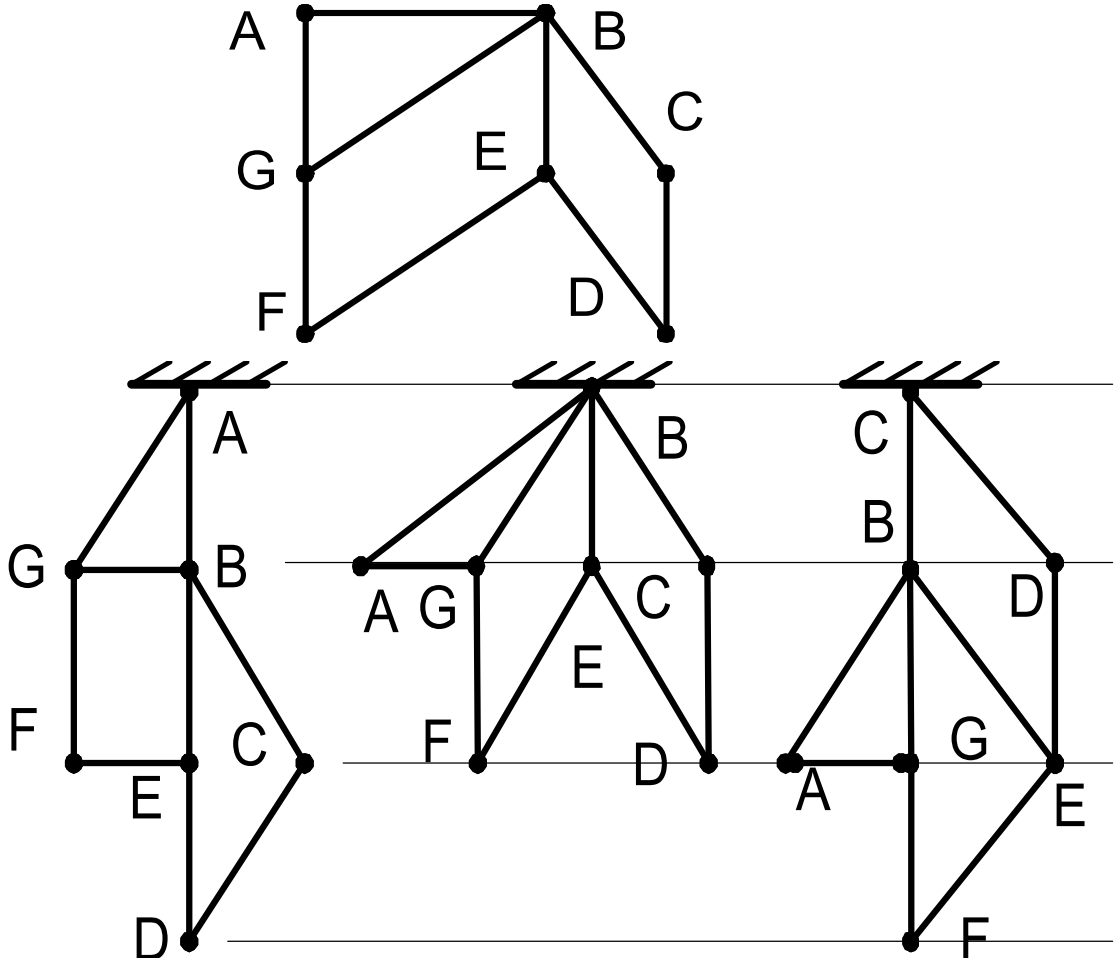


Рис. 1

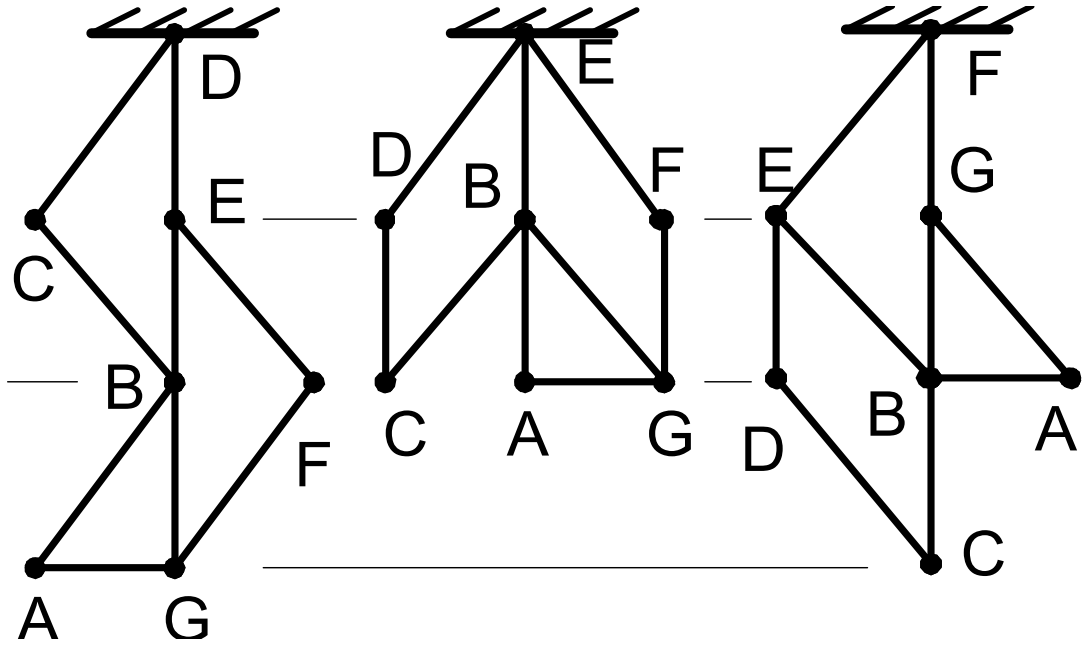


Рис.2.

Роботи першої групи рідко використовуються для розробки безпосередньо алгоритмів рішення задачі ідентифікації, але вони дозволяють дати оцінку обчислювальної складності алгоритмів її рішення, які відносяться до задач зі складним рішенням (NP -

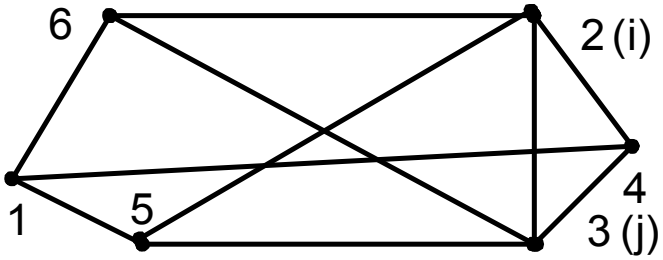
повні задачі), складність алгоритмів яких є експоненціальною і в деяких випадках сягає O(N!), (N - кількість вершин), тому доводиться відмовлятись від спроб досягнути точними методами їх рішення.

Наближені алгоритми використовують або допустимі рішення точних методів, або побудованні на використанні евристичних прийомів. На даний час створена певна кількість алгоритмів встановлення ізоморфізму графів, які за рахунок різних евристичних прийомів знижують складність задачі від факториальної до степеневої функції/2/. Серед них можна виділити наступні:

в) Метод побудови оптимального коду графа.

Цей метод базується на алгоритмі формування еквівалентних матриць зв'язності щляхом ідентифікації вершин з однаковими топологічними характеристиками /3/. Згідно цього алгоритму перетворюються матриці зв'язності графів однотипними методами. У випадку, коли модифіковані матриці однакові графи ізоморфні. Ізоморфізм між двома чи більше графами може бути визначений при допомозі використання їхніх оптимальних кодів. *Оптимальний код* графу отримується з верхнього трикутника матриці суміжності після перенумерації вершин графа/4/.

Згідно з методом побудови оптимального коду графа здійснюється перенумерація вершин графів, які досліджуються на ізоморфізм. Перенумерація вершин графа



відбувається по певних вибраних критеріях оцінки вершин графу. Оцінка здійснюється по їх числових значеннях, які використовуються при формуванні оптимального коду графа.

Число U. Число U кожного першого сусіда *j* деякої даної вершини *i* є кількість вершин, які є першими сусідами сусіда *j*, а також першими сусідами вершини *i*. Для графа на рис. 3 Uij=3.

Рис. 3

Число MW. Число MW вершини *i* є найбільше з всіх U чисел сусідів вершини *i*.

$$MW_i = \max U_{ij}$$

Число VN. Нехай вершина *j* є сусідом деякої взятої вершини *i* з максимальним числом U. Число VN вершини *i* є степінь вершини *j*.

$$VN_i = \text{val j } (U_{ij} \text{ -> max}).$$

Число B_d (0 < d). Число B_d якоїсь непозначеної вершини *j* є число, яке вказує мінімальну відстань між вершиною *j* та позначеною вершиною A_d, де

$$1 <= d <= \text{остання позначена вершина.}$$

Число C. Число C сусіда *j* позначеної вершини *i* показує кількість сусідів вершини *j*, які зв'язані з сусідами вершини *i*. Для графа на рис. 4 Cij=3.

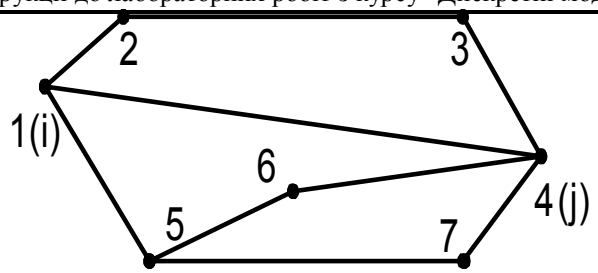


Рис. 4

Дані критерії можуть бути використані для рішення задачі встановлення ізоморфізму графів як регулярних (в яких всі вершини мають рівну степінь), так і нерегулярних.

Опишемо алгоритм побудови оптимального коду ненаправленого нерегулярного графа. Даний алгоритм використовує представлення графів у вигляді матриці суміжності MSUM=[m_{ij}] та матриці найменшої відстані MNV = [a_{ij}]. Матриця найменшої відстані графу G з P вершинами є P*P матриця, в якій a_{ij} показує довжину найменшого шляху від вершини *i* до вершини *j*.

Алгоритм побудови оптимального коду графа представлений такими процедурами:

1. Будуємо матрицю суміжності MSUM(N*N) та матрицю найменшої відстані MNV(N*N) для графу G (N,L), де N - кількість вершин в графі, а L - кількість зв'язків між вершинами. Приймаємо k = 1.

2. З головної діагоналі матриці найменшої відстані MNV знаходимо вершину (вершини) найбільшої степені. Якщо є єдина вершина з найбільшою степенню, то позначаємо її як A_k та переходимо на 10-й крок. В противному випадку (якщо маємо більше ніж одну вершину з найбільшою степенню) переходимо на 3-й крок.

3. Нехай, скажемо, є *j* вершин з найбільшою степенню (M_1,M_2,...,M_j). Для кожної вершини знаходимо число MW. Якщо є єдина вершина з найбільшим числом MW, то позначеємо цю вершину як A_k та переходимо на 10-й крок. Якщо маємо більше, ніж одну вершину з найбільшим числом MW, то переходимо на 4-й крок.

4. Нехай є *j* вершин з найбільшим числом MW (M_1,M_2,...,M_j). Для кожної вершини знаходимо число VN. Якщо є єдина вершина з найбільшим числом VN, то позначимо цю вершину як A_k та переходимо на 10-й крок. Якщо є більше ніж одна вершина з найбільшим числом VN, то переходимо на 5-й крок.

5. Нехай маємо *i* вершин з найбільшим числом VN(M_1,M_2,...,M_i). На даному етапі ми не можемо виділити з *I* вершин одну кращу вершину на місце A_k. Всі *I* вершин є можливими претендентами на A_k, отже ми повинні тепер виконувати процедури пошуку наступних вершин для $_I$ можливих претендентів на A_k, поки та чи інша єдина вершина не стане кращою виборкою для A_k, або поки не завершиться алгоритм. Переходимо на 10-й крок.

6. Знаходимо число B_1 для всіх непозначених сусідів вершини A_q (визначення подано на 11-тому кроці). Якщо є єдина вершина, яка має найменше число B_1, то позначаємо її як A_k та переходимо на 10-й крок. В противному випадку підраховуємо числа B_d (2 <= d <= k-1) для тих вершин, які мають мінімальне число B_{d-1}. Якщо знайдена єдина вершина з найменшим числом B_d, то позначаємо цю вершину як A_k та переходимо на 10-й крок. Якщо є більше ніж одна вершина з найменшим числом B_d, та якщо d = k-1, то перейти на 7-й крок. Якщо d < k-1, то збільшити d на одиницю та повторити обчислення.

7. Підраховуємо число U для кожних перших ссусідів вершини A_q, що мають найменше число B_d. Якщо є єдина вершина з найбільшим числом U, то позначити цю вершину як A_k та перейти на 10-й крок. В противному випадку переходимо на 8-й крок.

8. Обчислюємо степінь кожного першого непозначеного сусіда A_q, який має найменше число B_d та найбільше число U. Якщо є єдина вершина з найбільшою степенню, то позначити цю вершину як A_k та перейти на 10-й крок. В противному разі перейти на 9-й крок.

9. Підраховуємо число C для кожного першого сусіда A_q, який має найменше число B_d та найбільші число U і степінь. Якщо є єдина вершина з з найбільшим числом C то позначаємо цю вершину як A_k та переходимо на 10-й крок. Нехай, скажемо, є L вершин з найбільшим числом C ($M_1,M_2,...,M_l$). Тоді всі ці L вершин є претендентами на A_k, отже ми повинні тепер викрнувати процедури пошуку наступних вершин для L можливих претендентів на A_k. Переходимо на 10-й крок.

10. Якщо k = 1, то перейти на 11-й крок. Якщо є тільки одна виборка для A_{k-1}, то перейти на 11-й крок. Припустимо що є P виборок для A_{k-1}. Якщо вершина A_k не позначена для всіх виборок, то визначаємо A_q та переходимо на 6-й крок для позначення A_k для ще одніє] виборки A_{k-1} (порядок не важливий). В противному випадку для кожно] виборки виводимо числа B_d, число U, степінь та число *c* вершин, позначених як A_k у вигляді підкодів. Найкращими варіантами є підкоди з найменшими числами B_d та найбільшим числом U, степенню та числом *C*. Припустимо, що тільки R підходів є оптимальними. Далі ми будемо розглядати тільки ці підкоди. Переходимо на 11-й крок, щоб призначити A_{k+1} для вершин A_k, що залишились.

11. Приймаємо k = k+1. Якщо k = N, то переходимо на 12-й крок. Якщо k = N, то знаходимо A_q, де A_q - вже позначена вершина з найнижчим індексом, яка має принаймні одного непозначеного першого сусіда. Якщо A_q має тільки одного непозначеного першого сусіда, то призначити цього сусіда як A_k та перейти на 10-й крок. Якщо A_q має більше, ніж одного непозначеного першого сусіда, то перейти на 6-й крок.

12. Коли k=N, то вершина, яка залишається непозначеною, позначається як A_k для всіх виборок A_{k-1}. Якщо ми маємо P виборок, то тепер будуємо P оптимальних кодів. Якщо ці коди не є тотожніми, то код з найбільшою вагою вибирається як оптимальний код графа.

Поза тим обчислювальна складність такого алгоритму залишається досить високою, як правило не менше $O(N^5)/5/$.

б) З використанням згортки графа.

В наш час створена велика кількість алгоритмів встановлення ізоморфізму графів, які за рахунок різних евристичних прийомів знижують складність задачі від факторіальної (експоненціальної) до степеневої функції. Тим не менш обчислювальна складність цих алгоритмів залишається досить високою, як правило не менше $O(N^5)$, або ж її можна знизити ще на 1-2 порядки для графів, що мають якісь специфічні властивості.

Враховуючи характер задачі встановлення ізоморфізму графів, а вона є як складною комбінаторно-логічною задачею, так і задачею розпізнавання, для її рішення можна з успіхом використати метод паралельного згортання (редукції) схем /6/. При цьому можна отримати алгоритми, придатні для встановлення ізоморфізму графів (гіперграфів) любих видів, що мають обчислювальну складність, яка визначається базовим алгоритмом і не перевищує, відповідно, $O(N^3)$. Алгоритми базуються на двох наслідках з теорем, доведених в /7/. Вони носять евристичний характер і знаходять підстановку ізоморфізма для двох графів (гіперграфів), якщо ізоморфізм існує, і не знаходять її, якщо він відсутній.

В силу своєї евристичності при встановленні ізоморфізму нерегулярних ізоморфних графів (гіперграфів), які мають деякі регулярні вершини з однаковим

ступенем, відповідна підстановка може бути не знайдена з огляду на відсутність повного перебору можливих варіантів згортки. Інакше можна прийти до схеми методу гілок та границь із відповідною обчислювальною складністю. Для рішення задачі використовується модифікація алгоритму /8/ з нарощуванням по "динамічному " образу.

В процесі пошуку рішення задачі проходить паралельне нарощування ізоморфних підграфів (кусків), що знаходиться у повній відповідальності із гіпотезою Ілама /9/. При цьому на нульовому кроці роботи алгоритму необхідно провести класифікацію вершин графів по їх степеням, що допомагає при згортці графів і утворенні в процесі редукції нових кусків належним чином ранжувати зовнішні ребра цих кусків. В якості критеріїв згортки для одного з графів (обраного у якості образу) можна використати любий з відомих критеріїв (наприклад, c_{ij} max, де c_{ij} – елемент матриці зв'язності С) або їх комбінацію адитивну чи мультиплікативну, чи з ранжуванням до можливого усунення невизначеності редукції. Для іншого графа критерієм оптимальної редукції буде утворення кусків (підграфів), ізоморфних образу на даному кроці редукції. Інакше кажучи, повинні будуватись в процесі рішення задачі тотожні (ізоморфні) дерева паралельної редакції /6/ для двох досліджуваних графів (еквівалентні зображення).

Для зменшення ймовірності виникнення невизначеностей в процесі редукції її слід з першого кроку зробити направленою (примусовою), використовуючи в якості обов'язкових "центрів кристалізації" (зародків по критерію c_{ij} max) вершини того класу, який має для графу-образа мінімальну потужність. Попередньою необхідною умовою рішення задачі буде співпадіння на нульовому кроці потужностей відповідних класів вершин, які мають однакові степені.

Таким чином, існує два варіанта створення подібних алгоритмів. Це утворення динамічного образа з примусовою "кристалізацією" (з наперед визначеними одновершинними "зародками" – кусками), а також зі свобідною редукцією образа по наперед обраних критеріях (напр., $\Delta = \max c_{ij}$, $\delta = \min (c_I + c_j - 2c_{ij})$, де Δ - основний, а δ – додатковий критерій згортки, і т.д.).

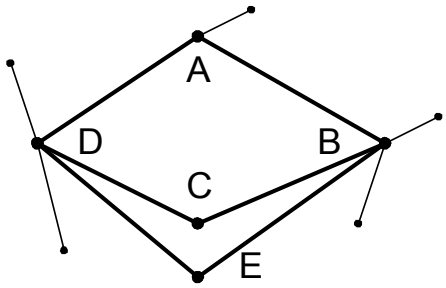
В любому випадку спочатку робиться крок паралельної згортки на графі-образі, а потім еквівалентий йому крок на іншому графі, ізофорфізм з яким досліджується. Причому тут реалізується принцип зворотнього зв'язку, тобто, наприклад, у випадку примусового нарощення в графі-образі розглядаються по черзі всі зовнішні ребра кожного нарощуваного куска, якщо це необхідно для знаходження еквівалентного йому зв'язку (ребра) в іншому графі.

Неможливість побудови на якомусь кроці для двох досліджуваних на ізоморфізм графів тотожніх дерев паралельної редукції свідчит про відсутність ізоморфізма або неможливість його встановлення даним алгоритмом .

Згідно з /7/ код, тобто вага відповідної вершини дерева оптимальної (паралельної) редукції, порівняння пар кусків (вершин), які можуть об'єднатись у процесі редукції повинен містити наступні впорядковані характеристики:

1) комбінацію класів вершин (кусків);
2) кількість вершин (кусків), що об'єднуються інцидентних ребрам, які стають внутрішніми при об'єднанні;
3) кількість спільних ребер для кусків, що об'єднуються;
4) кортеж класів вершин, інцидентних спільним ребрам;
5) кількість зовнішніх ребер утворюованого куска;
6) кортеж класів вершин, інцидентних зовнішнім ребрам утворюваного куска.

Для гіперграфів характерні будуть також “змішані” ребра, в зв’яку з чим з’являться ще дві характеристики в ході порівняння, аналогічні чотирьом останнім з описаних вище. Зауважимо, що формування коду простіше в тому випадку, коли процес



редукції є направленим, тобто тоді, коли проходить паралельна редукція графів з обмеженою кількістю “зародків” у вигляді наперед визначених вершин.

В роботі /10/ код був доповнений характеристиками:

7) кількість паралельно-суміжних вершин, по типам(рис.5, типи вершин С і Е є різні);

Рис.5

8) кількість трьохвершинних циклів(рис.6, для вершини А кількість рівна трьом), що включають дану вершину;

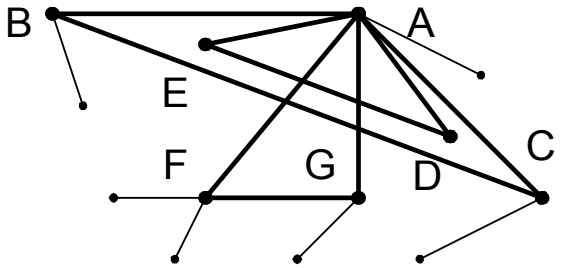


Рис.6.

9) кількість чотирьохвершинних циклів(рис.7., для вершини А кількість рівна двом), що включають дану вершину;

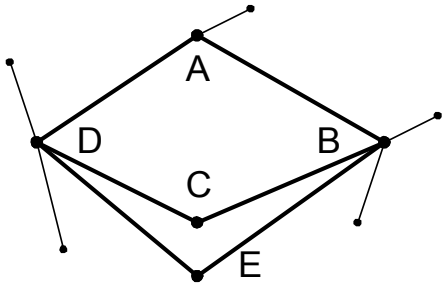


Рис.7

Отже можна зробити висновок, що серед існуючих методів визначення ізоморфізму їх редукція, що відображається бінарним деревом, дозволяє звести встановлення ізоморфізму до встановлення тотожності інформаційних кодів графів, а запропоновані характеристики інформаційного коду вершини(куска або підграфа) графа в більшості випадків дозволяють ідентифікувати вершини(куски або підграфи) для вирішення задачі покриття.

4. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Які Ви знаєте алгоритми рішення задачі встановлення ізоморфізму графів?
2. Як вирішується теоретична задача встановлення ізоморфізму простих графів?
3. Яка ідея рішення задачі встановлення ізоморфізму наближеними методами?
4. Які основні кроки алгоритму побудови оптимального коду ненаправленого нерегулярного графа?
5. Яка сутність методів згортки(редукції) графа?

4. ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Отримати у викладача індивідуальне завдання.
2. Підготувати програму для вирішення виданого завдання.
3. Запустити на виконання програму відповідного методу.
4. Проглянути результат роботи програм. Результат роботи може бути: ізоморфізм встановлено або не встановлено.
5. У випадку, коли ізоморфізм встановлено (не встановлено), необхідно модифікувати граф, коректуючи два або три зв’язки, щоб знайти такий граф, на якому ізоморфізм не встановлюється (встановлюється).
6. Зафіксувати результати роботи у викладача.
7. Оформити і захистити звіт.

5. ЗМІСТ ЗВІТУ

1. Опис методів рішення задачі комівояжера.
2. Нарисувати графи і вхідні дані, що їх описують для двох результатів.
3. Результати розрахунків.
4. Висновки.

ЛІТЕРАТУРА

1. **Ас**ельдеров З.М. и др. Представление графов и операции над ними. -Киев, 1987. 18с.
2. Зыков А.А. Основы теории графов. -М.: Наука, 1987. -384с.
3. Оцуки Т. Эвристические алгоритмы для комбинаторных задач с большим обьемом вычислений. “Дэнси цусим гакайси”. 1975. 58. N4. -С.416-423.
4. Е. Майніка “ Алгоритмы оптимизации на сетячх и графах. “Москва, “ Мир “, 1981.
5. Руссо, Вольф. Машинно-ориентированный метод разбиения и отображения графов логических схем ЭВМ. В кн.: Автоматизация в проектировании. М.: Мир, 1972.
6. Оре О. Графы и их применение. М., "Мир", 1965.
7. Оре О. Теория графов. -М.: Наука. Главная редакция физико-математической литературы, 1980. -336с.
8. Карелин В.П., Миронов Б.Н. Алгоритм направленного случайного поиска для распознавания изоморфного вложения графов на автоматной модели. В кн.:Однородные цифровые вычислительные и интегрирующие структуры, вып.6. Таганрог, ТРТИ,1976, с.84-93.
9. Горинштейн Л.Л. Метод упорядоченного перебора для разрезания графов. Таганрог, ТРТИ, 1970.
10. Базилевич Р.П., Ткаченко С.П. Решение задачи методом параллелього свертивания. В кн.: Вычислительная техника,VII, Каунас, КПИ, 1975, с. 295-298.
11. Ткаченко С.П. Разработка и исследование алгоритмических методов решения задач декомпозиции для использования в автоматизированных системах конструкторского проектирования. Диссертация на соискание ученой степени кандидата технических наук. Львов, ЛПИ, 1980.
12. Ткаченко С.П. и др. Поиск в схеме подсхем с заданными характеристиками. В сб.: Автоматизация проектирования радиоэлектронной аппаратуры на промышленных предприятиях. Киев, РДЭНТП 1976.
13. Харари Ф. Теория графов. М; “Мир”, 1973, 302 с.
14. Павлів О.О., Ткаченко С.П Чура І.І. Методика перевірки відповідності спроектованої топології МЕП вхідній ПС.Вісник Державного університету “Львівська політехніка” № 322 ”Комп’ютерна іженерія та інформаційні технології”. -Львів, 1994р.

ДОДАТКОВА ЛІТЕРАТУРА:

1. Абукаримов Р.Т. Алгоритмы распознавания, основанные на поиске признаков классов."Вопросы кибернентики", вып. 88, 1976, с.3-20.
2. Асельдеров З.М., Г.А.Донец. Представление графов и востановление графов . - Киев, Наук. думка, 1991, - 192с.
3. Асельдеров З.М.и др. Представление графов и операции над ними. - Киев, 1987, - 18с.
4. Кохов В.А., Лазарев В.А. Алгоритм идентификации обыкновеных графов. -"Тр. Моск. энерг.ин-та", 1976, вып.299, с.53-60.
5. Курейчик В.М., Королев А.Г. Применение алгоритма изоморфизма графов для контроля схем БИС. - Микроэлектроника, 1976, т.5, вып 5, с.400.
6. Курейчик В.М., Королев А.Г. Об одном методе изоморфного вложения графов. В кн.: Методы расчета и автоматизации проектирования устройств микроэлектронных ЦВМ. Киев, ИК АН УССР, 1975, с.6-16.
7. Мелихов А.Н., Курейчик В.М., Королев А.Г. Решение задач контроля при техническом проектировании на основе распознавания изоморфизма графов. В кн.:Вычислительная техника, IX, Каунас, КПИ, 1977, с.139-141.