МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УЖГОРОДСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ»
ІНЖЕНЕРНО-ТЕХНІЧНИЙ ФАКУЛЬТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ СИСТЕМ ТА МЕРЕЖ

Балога С. І., Король І. Ю.

МЕТОДИЧНІ ВКАЗІВКИ І ЗАВДАННЯ
ДО ЛАБОРАТОРНИХ РОБІТ
З КУРСУ

ДИСКРЕТНА МАТЕМАТИКА

для студентів 1-го курсу
спеціальності «Комп'ютерні системи та мережі»

Ужгород – 2012

Балога С. І., Король І. Ю. Методичні вказівки і завдання до лабораторних робіт з курсу «Дискретна математика» для студентів 1-го курсу інженерно-технічного факультету спеціальності «Комп'ютерні системи та мережі». – Ужгород: видавництво УжНУ «Говерла», 2012. – 56 с.

Укладачі: **Балога С. І., канд. фіз.-мат. наук, доцент кафедри комп'ютерних систем та мереж;**

Король І. Ю., канд. фіз.-мат. наук, доцент.

Рецензент: **Щобак Н. М., канд. фіз.-мат. наук, доцент кафедри диференціальних рівнянь та математичної фізики.**

Відповідальний за випуск: **Туряниця І. І., канд. фіз.-мат. наук, доцент, декан інженерно-технічного факультету.**

Дані методичні вказівки розглянуто та схвалено на засіданні кафедри комп'ютерних систем та мереж, протокол № 4 від 8 листопада 2012 р. та методичної комісії інженерно-технічного факультету, протокол № 1 від 9 листопада 2012 р.

ВСТУП

Підготовка інженера, в тому числі і фахівця з комп'ютерної інженерії на сучасному етапі, базується не тільки на засвоєнні основних розділів математики та вмінні застосовувати їх на практиці традиційним способом, але і вмінні застосовувати їх з використанням сучасних комп'ютерних технологій.

Практика викладання курсу "Дискретна математика" для інженерів показує, що для досягнення хороших успіхів у засвоєнні знань, одержаних на заняттях з інформатики та математики, є можливим лише при умові, коли ці дві складові поєднуються. При поєднанні цих складових спостерігається більша зацікавленість студентів у вивченні матеріалу, ніж на звичайних практичних чи лабораторних заняттях. Це зумовлено, в першу чергу, можливістю оперативного експерименту та творчого підходу при вирішенні тих чи інших конкретних завдань.

На сьогоднішній день розроблена достатня кількість пакетів прикладних програм, які дозволяють швидко і ефективно виконувати потрібні обчислення, аналітичні перетворення, графічні побудови тощо, а тому є можливість приділити більше уваги постановці задачі, побудові математичної моделі та дослідженню розв'язків, що необхідно студентам інженерних спеціальностей.

Для комп'ютерної підтримки вивчення дискретної математики ми пропонуємо використовувати середовище Maple, правила користування яким вкрай прості, а можливості великі. Середовище Maple містить досить широкий набір функцій для роботи з графами і тому ми пропонуємо лабораторні роботи по теорії графів. Maple володіє вбудованою мовою програмування, яка дає змогу користувачеві запрограмувати розв'язання спеціальних задач.

В рекомендованих методичних вказівках даються короткі теоретичні відомості до кожної лабораторної роботи як з відповідного розділу дискретної математики, так і можливостей середовища Maple. Крім цього, в кожній лабораторній роботі розглядаються типові приклади з відповідної теми та їх розв'язання за допомогою засобів середовища Maple.

Лабораторна робота № 1

Тема: ПОЧАТКОВЕ ЗНАЙОМСТВО З ПРОГРАМОЮ MAPLE ДЛЯ РОБОТИ З ГРАФАМИ

Мета роботи: Оволодіти початковими навичками роботи з програмою Maple.

Зміст роботи:

1. Запуск програми Maple. Знайомство з робочим вікном Maple. Функції бібліотеки networks в Maple.
2. Початкове знайомство з основними функціями для побудови графів.
3. Функція підрахунку кількості ребер і вершин та її використання.
4. Способи побудови графів за допомогою функцій Maple.

Зміст звіту: Короткі теоретичні відомості. Постановка індивідуальних завдань та результати їх виконання.

Теоретичні відомості

1. Історія появи і розвитку теорії графів

Перша робота з теорії графів, яка належала відомому швейцарському математику Л. Ейлеру, з'явилась в 1736 р. Поштовх до розвитку теорія графів одержала на межі XIX і XX століть, коли різко зросло число робіт в галузі топології і комбінаторики, з якими вона дуже тісно пов'язана. Графи стали використовуватися при побудові схем електричних ланцюгів і молекулярних схем. Як окрема математична дисципліна, теорія графів була вперше представлена в роботі угорського математика Кеніга в 30-і роки XX століття.

Останнім часом графи і зв'язані з ними методи досліджень пронизують на різних рівнях чи не всю сучасну математику. Теорія графів розглядається як одна із галузей топології; безпосереднє відношення вона має також до алгебри і до теорії чисел. Графи ефективно використовуються в теорії планування і управління, теорії розкладів, соціології, математичній лінгвістиці, економіці, біології, медицині, географії. Широке застосування знаходять графи в таких галузях як програмування, теорії скінчених автоматів, електроніці, при розв'язуванні імовірнісних і комбінаторних задач, знаходженні максимального потоку в мережі, найкоротшої відстані, перевірці планарності графа та ін. На сучасному етапі теорія графів швидко розвивається і знаходить все нові й нові застосування.

2. Функції бібліотеки networks в Maple

Для роботи з графами в Maple призначена бібліотека **networks.** Команда підключення цієї бібліотеки стандартна, тобто достатньо скористатися оператором **with()**. Граф у Maple подається особливою процедурою типу GRAPH. Для роботи з графами можна скористатися будь-якою із 75-ти функцій, які знаходяться в бібліотеці **networks**. Основні функції, які дають можливість створювати і змінювати графи, наведено нижче.

Команда запуску пакету функцій теорії графів **– with(networks)**.

Функція створення порожнього графа - **new**().
Функція створення порожнього графа (без ребер) - **void**().
Функція створення рисунка графа - **draw**().
Функція створення довільного графа - **graph**().
Функція створення повного графа - **complete**().
Функція створення циклу - **cycle**().
Функція створення графа Петерсона - **petersen**().
Функції створення платонових графів:
Граф куба - **cube**().
Граф тетраедра - **tetrahedron**().
Граф октаедра - **octahedron**().
Граф додекаедра - **dodecahedron**().
Граф ікосаедра - **icosahedron**().
Функція додавання в граф вершин - **addvertex**().
Функція виведення списку усіх вершин графа - **vertices**().
Функція додавання в граф ребер - **addedge**().
Функція виведення списку всіх ребер графа (стандартних позначень) - **edges**().
Функція виведення списку всіх ребер графа через задання початкової і кінцевої вершин ребра - **ends**().
Функція вилучення із графа ребра або вершини - **delete**().
Функція сполучення одних заданих вершин з другими - **connect**().
Функція знаходження початкової вершини орієнтованого ребра - **tail**().
Функція знаходження кінцевої вершини орієнтованого ребра - **head**().
Функции знаходження набору вершин за заданою вершиною - **neighbors**(), **departures**() і **arrivals**().
Функція, яка виводить ребра, інцидентні вершині - **incident**().
Функція створення копії графа - **duplicate**().
Функція створення випадкового графа - **random**().
Функція визначення зв’язності графа - **connectivity**().
Функція знаходження зв’язних компонент і мостів графа - **bicomponents**().
Функція знаходження зв’язних компонент графа - **components**().
Функції визначення степенів вершин у графі - **degreeseq**(), **indegree**(), **vdegree**(), **outdegree**().
Функція, яка знаходить доповнення графа - **complement**().
Функція об’єднання двох графів - **gunion**().
Функція знаходження найкоротшого циклу в графі - **girth**().
Функція знаходження найкоротших шляхів у графі - **shortpathtree**().
Функція визначення планарності графа - **isplanar**().
Функція обчислення матриці інцидентності - **incidence**().
Функція обчислення матриці суміжності - **adjacency**().
Функція виведення таблиці даних про граф - **show()**.

3. Початкове знайомство з роботою основних функцій

Стандартна команда запуску бібліотеки функцій теорії графів – **with(networks).**

> **with(networks):**

Створення порожніх графів:
функція створення порожнього графа (без ребер і вершин) – **new**();
функція створення порожнього графа (без ребер) – **void**().

> **G:=new():**
> **H:=void(5):**

В результаті виконання вказаних функцій ми дістали порожній граф G – без вершин та ребер і порожній граф H – без ребер, але з п'ятьма вершинами. Тепер нарисуємо граф H. Функція перегляду зображення графа **draw().**

Ця функція використовується для візуального показу вершин і ребер графа. Недолік її в тому, що кратні ребра зображуються як одне ребро, орієнтовані ребра рисуються як неорієнтовані (прості відрізки), а петлі не рисуються взагалі. Оскільки граф G не має ні ребер, ні верши, то ми можемо подивитись тільки граф H (рис. 1):

> **draw(H);**

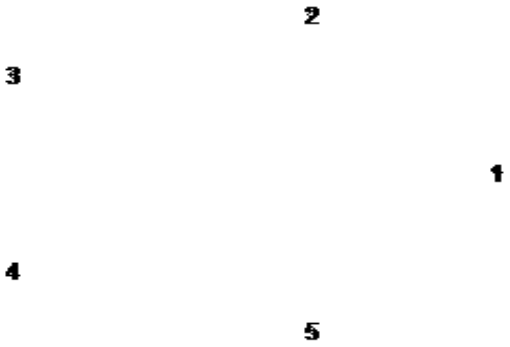


Рис. 1.

Функція створення довільного графа **graph()**. Приклад графа з трьома вершинами і двома ребрами показано на рис. 2.

> **G1:= graph({1, 2, 3}, {{1, 2}, {2, 3}}): edges(G1);**

$\{e1, e2\}$

>**draw(G1);**

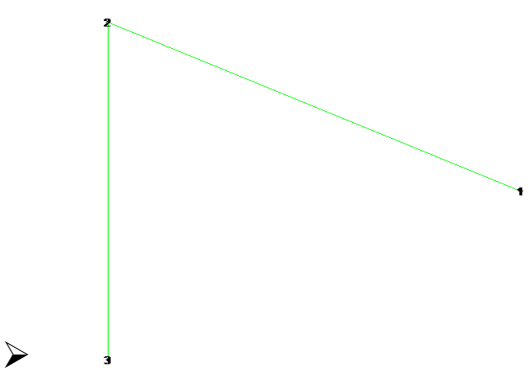


Рис. 2.

Функція створення повного графа **complete()**.

>G2:= complete(4): ends(G2);

{{3, 4}, {1, 2}, {2, 3}, {1, 4}, {1, 3}, {2, 4}}

Ця функція генерує різні типи повних графів. Кількість аргументів у дужках вказує на кількість частин графа, які необхідно з'єднати. Кожна частина визначається цілим числом, яке показує кількість вершин в цій частині. Наприклад, повний двочастинний граф визначений як повний (m,n). Якщо один аргумент, то він задає кількість вершин, які будуть з'єднані одна з одною.

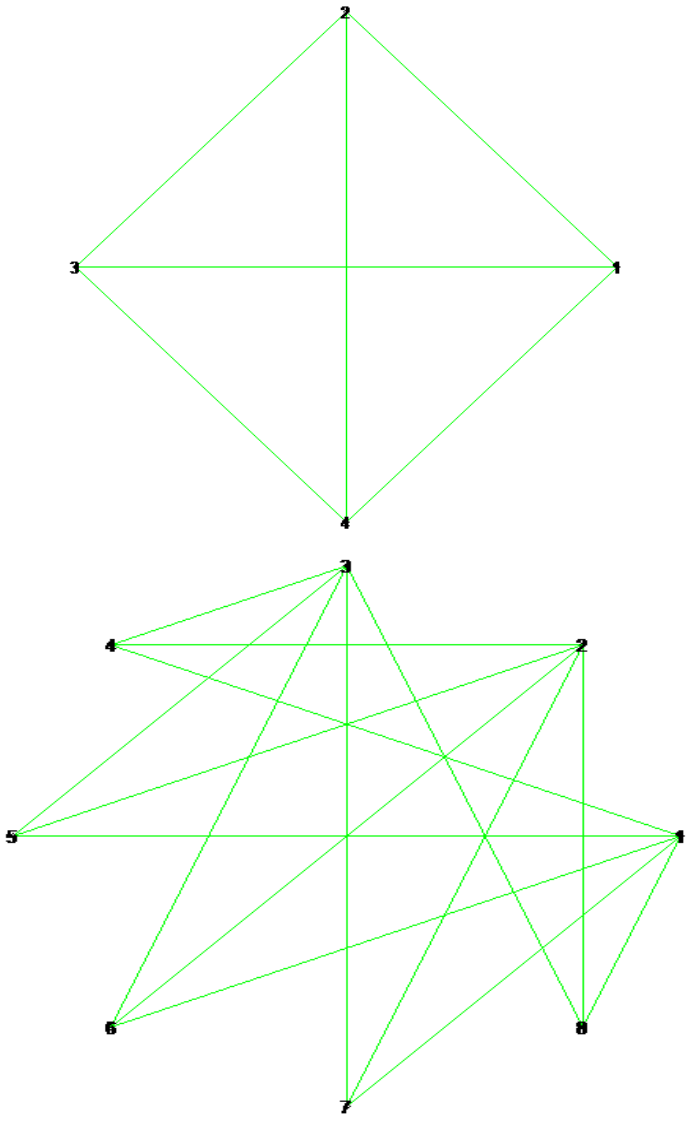
>G3:=complete(3, 5): ends(G3);

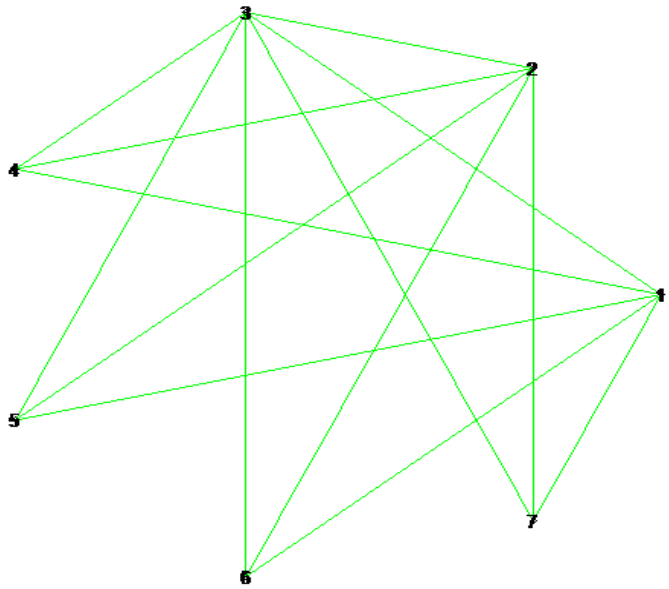
{{3, 4}, {3, 5}, {1, 7}, {2, 7}, {2, 6}, {3, 7}, {3, 6}, {1, 5}, {1, 8}, {2, 5}, {1, 4}, {1, 6}, {2, 8}, {2, 4}, {3, 8}}

>G4:=complete(2, 1, 4): ends(G4);

{{3, 4}, {3, 5}, {1, 7}, {2, 7}, {2, 6}, {3, 7}, {3, 6}, {1, 5}, {2, 3}, {2, 5}, {1, 4}, {1, 6}, {1, 3}, {2, 4}}

> draw(G2); draw(G3); draw(G4);

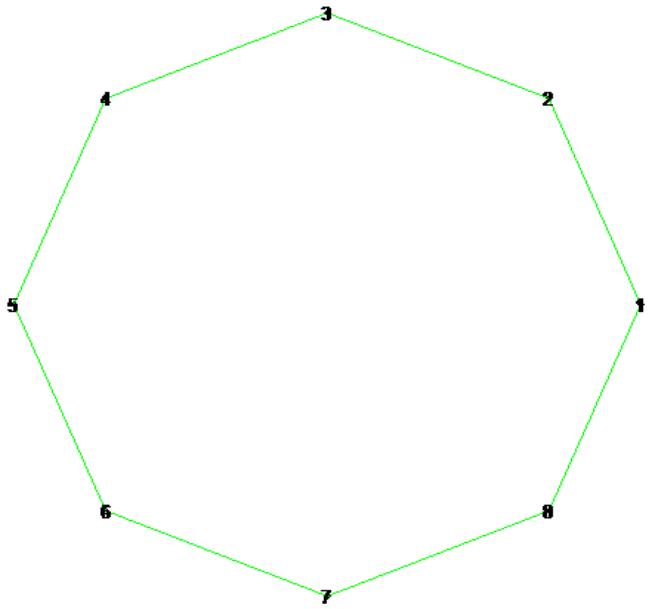




Функція створення циклу **cycle()**.

>**G5:=cycle(8):**
ends(G5); draw(G5);

{{3, 4}, {4, 5}, {5, 6}, {1, 2}, {1, 8}, {2, 3}, {6, 7}, {7, 8}}



Припустимо, що футбольна команда нашого факультету бере участь у змаганнях і грає з командами інших факультетів. Нехай загальне число команд дорівнює шести. Нашу команду позначимо буквою A, а інші команди – буквами B, C, D, E і F. Через декілька тижнів після початку змагань виявиться, що деякі із команд уже зіграли одна з одною, наприклад:

A з C, D, F;
B з C, E, F;
C з A, B;
D з A, E, F;
E з B, D, F;
F з A, B, D, E.

Це можна зобразити за допомогою такої геометричної схеми. Кожну команду зобразимо точкою або маленьким кружечком і з'єднаємо відрізком ті пари точок, які відповідають командам, які уже грали одна з одною. Тоді для даного списку проведених ігор за допомогою спеціальних функцій бібліотеки **networks** ми дістанемо схему, показану на рис.3.

```
> with(networks):
> G:=new():   addvertex({A, B, C, D, E, F}, G):
connect({F}, {E, A, B, D}, G):   connect({D}, {E, A}, G):
connect({B}, {E, C}, G):   addedge({A, C}, G):
draw(Concentric([B, A, F, E, D, C]), G);
```

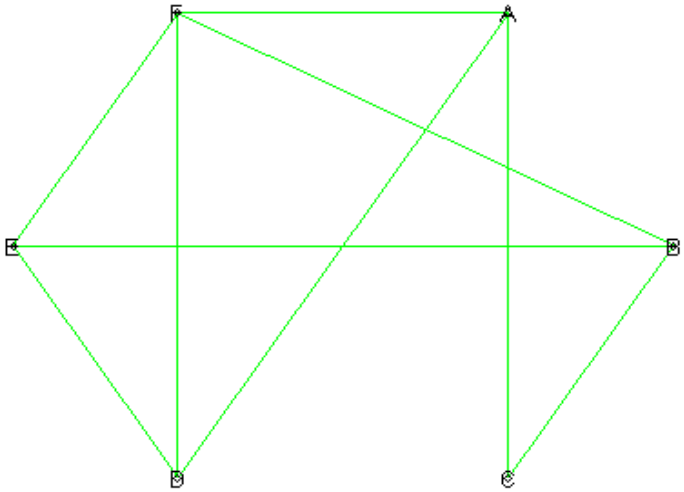


Рис.3.

Схема такого вигляду називається ***графом.*** Вона складається із декількох точок A, B, C, D, E, F, які називаються ***вершинами*** і декількох відрізків, які з'єднують ці точки, таких як AC або EB, які називаються ***ребрами*** графа.

На рис. 3 видно, що точки перетину деяких ребер графа можуть не бути його вершинами; це відбувається тому, що ми зобразили наш граф на площині. Можливо, було би більш зручно уявити собі його ребра нитками, які проходять одна над одною в просторі.

Визначимо поняття графа математично.

Графом *називається не порожня множина точок і множина відрізків, обидва кінці яких належать заданій множині точок.*

В Maple граф будемо позначати великими або малими буквами латинського алфавіту, крім D, але прийнято позначати буквами G і H. Також можна позначати довільними словами, написаними латинськими буквами, крім службових слів. Вершини графа позначають усіма додатними числами від нуля, а також довільними англійськими словами. Ребра графа позначають парою його вершин: {1,2} – ребро з кінцевими вершинами 1 і 2, або, наприклад, {max, min}. За замовчуванням у Maple 8 кожному ребру присвоюється індексне ім'я e1, e2, e3 і т.д.(цифра означає порядок створення даного ребра).

При зображені графів на рисунках або схемах відрізки можуть бути прямолінійними або криволінійними (див. рис. 4), але в математичній системі

Maple графи зображуються тільки прямолінійними відрізками.

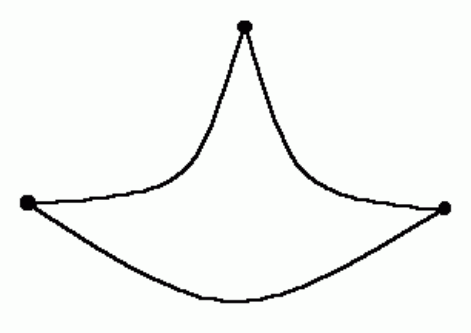


Рис.4.

Одну і ту ж саму пару вершин у графі можуть з'єднувати декілька ребер, а в одної вершини можуть бути так звані "петлі".

Ребра, які з'єднують одну пару вершин, називаються ***кратними.***

Ребро, яке з'єднує вершину саму з собою, називають ***петлею.***

Функція **draw()** виведення схеми графа на екран, на жаль, не виводить петлі і кратні ребра. Їх наявність можна перевірити іншими функціями.

Довжини відрізків і розташування точок довільні, наприклад, той самий граф, що зображений на рис.3, можна зобразити і так:

> **draw(Linear([E, F], [A, B, C, D]), G);**

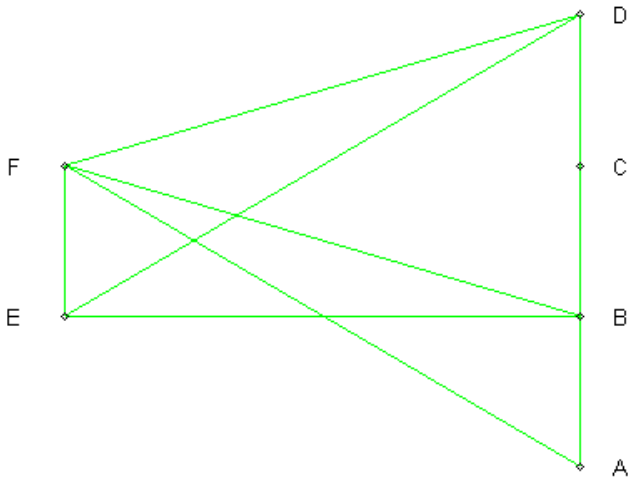


Рис.5.

Завдання для самостійної роботи

1. Дайте повний список проведених ігор, які відповідають графу:

>G:=void(8): connect({1}, {2 ,5 ,4}, G): connect({3}, {2 ,7 ,4}, G): connect({8}, {5, 6, 7, 4}, G): connect({6}, {2, 5, 7, 4}, G): connect({7}, {2, 5}, G): draw(Concentric([6, 5, 8, 7], [2, 1, 4, 3]), G);

2. У змаганнях з волейболу беруть участь 6 команд. Нарисуйте граф, в якому вершинами є команди, а ребрами – ігри, зіграні між командами:

а) якщо відомо, що зіграли наступні команди:

A з D, F
B з C, F
C з B, D, E, F

D з A, C
E з C, F
F з A, B, C, E;

b) якщо змагання закінчились;

c) до початку змагань.

3. Якими функціями Maple можна вияснити, чи містить граф петлі та кратні ребра?

4. Функція підрахунку кількості ребер і вершин

У багатьох задачах необхідно знати кількість ребер і вершин графа. Порахувати кількість ребер і вершин безпосередньо на рисунку часто не є можливим, наприклад, із-за великої їх кількості. В Maple цю можливість надає нам функція **nops().**

Нехай, наприклад, потрібно знайти кількість проведених ігор у змаганнях між командами, зображених графом, поданим на рис. 3, тобто кількість ребер відповідного графа.

Дане завдання можна виконати за допомогою наступної послідовності функцій:

```
> with(networks):
> G:=new(): addvertex({A, B, C, D, E, F}, G): connect({F}, {E, A, B, D},
G):  connect({D}, {E, A}, G):  connect({B}, {E, C}, G): addedge({A, C}, G):
draw(Concentric([B, A, F, E, D, C]), G);
> nops(edges(G));
```

9

Ми одержали 9 ребер-ігор. Аналогічно ми можемо знайти і кількість вершин графа.

```
> nops(vertices(G));
```

6

Завдання для самостійної роботи

1.Скільки ребер і вершин мають графи, зображені на рисунках 6, 7, 8, 9?

```
> G:=new():  G:=random(6, 5):  draw(G);
```

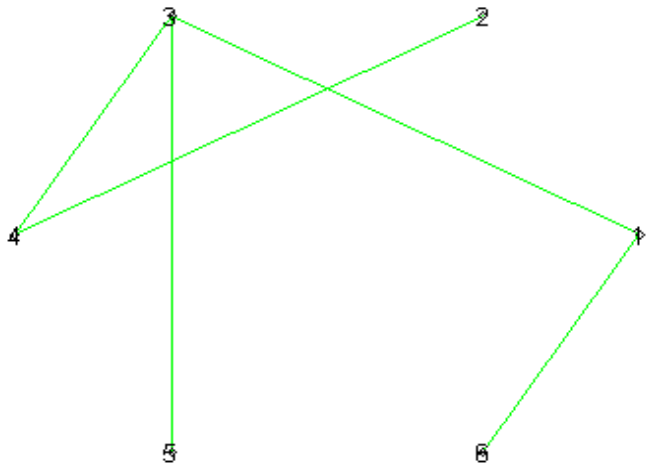


Рис. 6.

> **G1:=icosahedron(): draw(G1);**

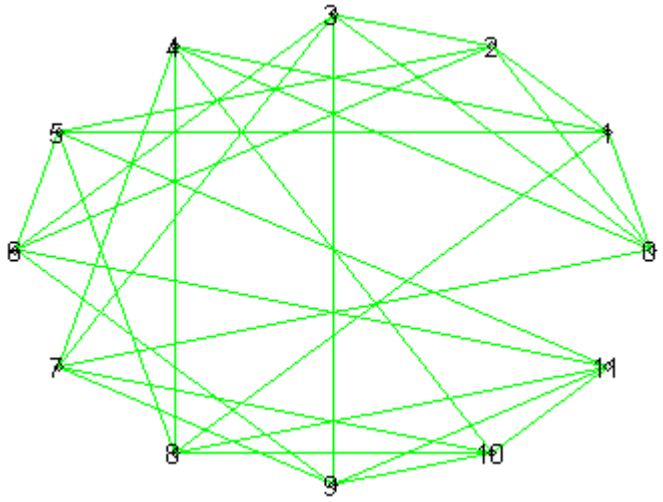


Рис.7.

> **G2:=cube(3): delete({2}, G2): draw(G2);**

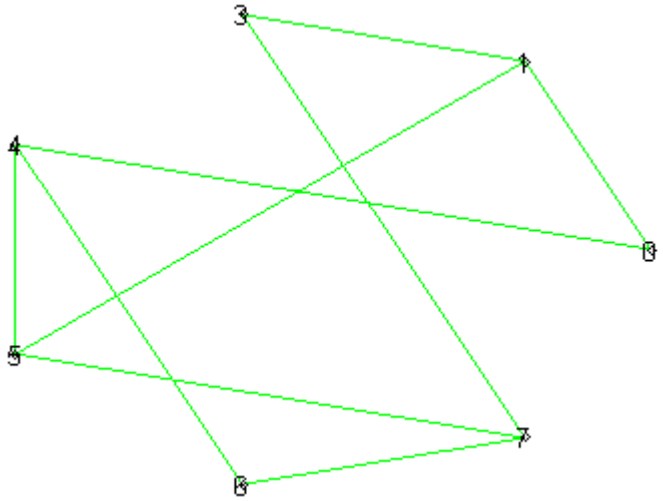


Рис.8.

> **G3:=dodecahedron(): G4:=complement(G3): draw(G4);**

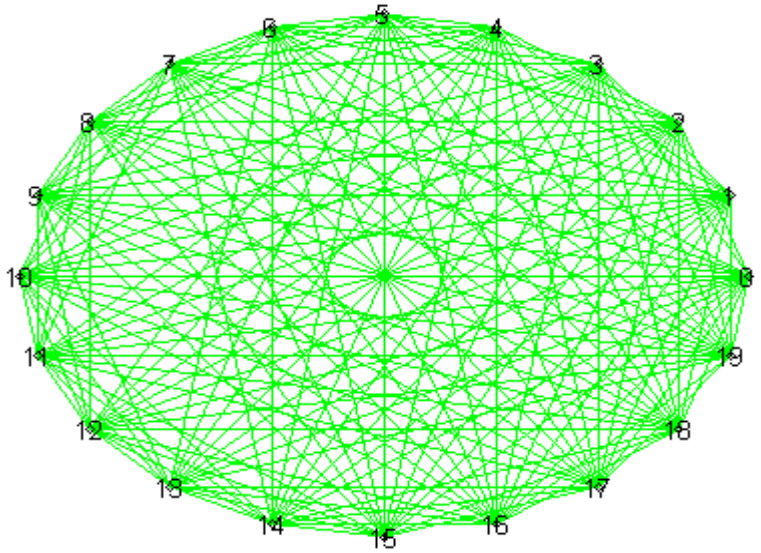


Рис.9.

5. Способи побудови графів за допомогою функцій Maple

Розглянемо деякі способи побудови графів за допомогою функцій Maple. Для цього доцільно нарисувати граф спочатку на папері.

Розглянемо приклад інтерпретації букви у вигляді графа, наприклад Ж. Із множини розв'язків задачі потрібно вибрати оптимальний. При цьому граф повинен бути правильним.

> **with(networks):**
> **G:=void(7): connect({4}, {1, 2, 3, 5, 6, 7}, G): draw(G);**

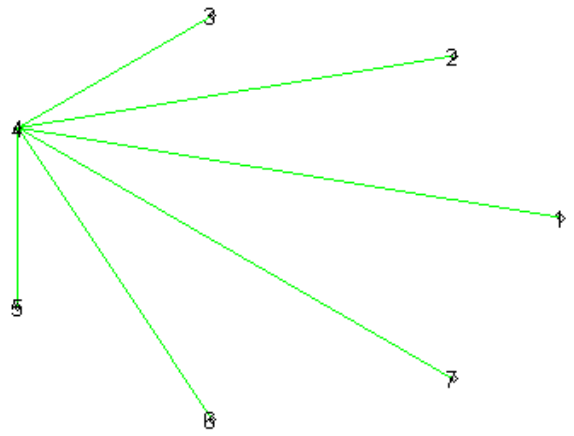


Рис.10.

Ми одержали рисунок зовсім не схожий на букву Ж. Нарисуємо граф інакше.

> **draw(Linear([5, 1], [6, 4, 2], [7, 3]), G);**

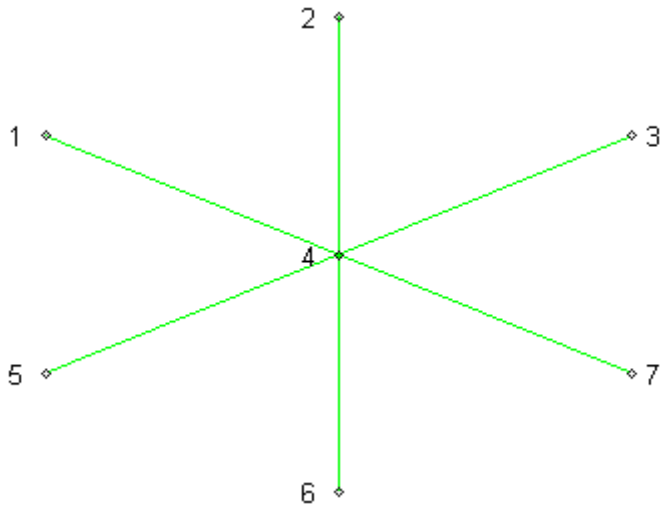


Рис.11.

Розглянемо наступний приклад побудови графа. Побудуємо граф, який відповідає кубу, у якого проведені всі діагоналі. Вершинами графа є вершини куба і точки перетину його діагоналей, а ребрами – відповідні ребра куба і відрізки його діагоналей.

> **with(networks):**
> **G:=cube(3): draw(Concentric([0, 1, 3, 2], [4, 5, 7, 6]), G);**

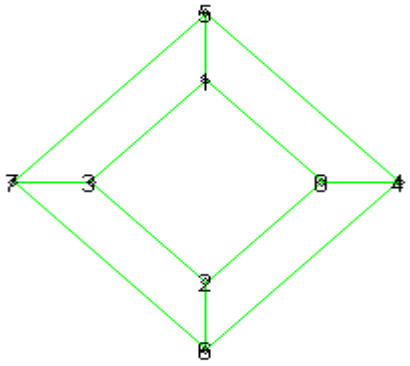


Рис.12.

Одержаний граф відповідає кубу. Додамо в граф вершини, які є точками перетину діагоналей куба:

> **addvertex({8, 9, 10, 11, 12, 13}, G):**
> **draw(Concentric([8], [0, 1, 3, 2], [4, 5, 7, 6]), G);**

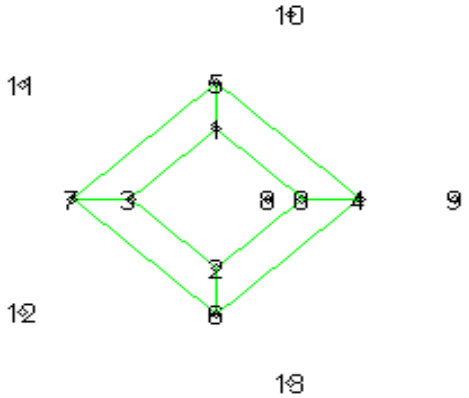


Рис.13.

Додамо ребра, яких не вистачає і граф готовий:

>**connect({8}, {0, 1, 2, 3}, G): connect({9}, {0, 4, 6, 2}, G):**
connect({10}, {1, 5, 4, 0}, G): connect({11}, {7, 3, 1, 5}, G):
connect({12}, {7, 3, 2, 6}, G): connect({13}, {5, 6, 4, 7}, G):
>**draw(Concentric([8], [0, 1, 3, 2], [4, 5, 7, 6]), G);**

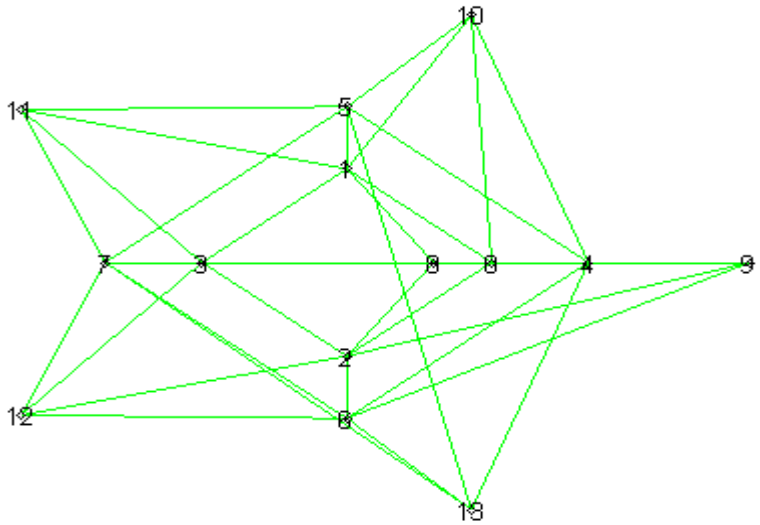


Рис.14.

Завдання для самостійної роботи

1. Нарисуйте графи, які відповідають великим буквам українського алфавіту: К, Щ, П.

2. Створіть граф, який відповідає наступній схемі: за вершини графа взяті усі вершини і точки перетину діагоналей куба, а ребрами є відрізки його діагоналей.

Лабораторна робота №2

Тема: ВИКОРИСТАННЯ ПРОГРАМИ MAPLE ДЛЯ ПОБУДОВИ ДЕЯКИХ РІЗНОВИДІВ ГРАФІВ

Мета роботи: Ознайомлення з можливостями програми Maple для побудови деяких різновидів графів. Набути навички при побудові графів, що задовольняють певним властивостям.

Зміст роботи:

1. Вивчити спеціальні функції із бібліотеки для роботи з графами, що дозволяють будувати різні типи графів.

2. Ознайомитися з функціями визначення степенів вершин орієнтованих та неорієнтованих графів і застосування їх до розв'язування задач.

3. Виконати запропоновані завдання з використанням засобів програми Maple.

Зміст звіту: Короткі теоретичні відомості. Постановка індивідуальних завдань та результати їх виконання.

Теоретичні відомості

1. Основні поняття теорії графів

У графі не всі вершини з'єднанні ребрами. Вершини, які не належать жодному ребру, називаються ***ізольованими.*** Граф, зображений на рис.1, має одну ізольовану вершину, а на рис.2 – усі вершини ізольовані.

> **with(networks):**

> **G:=void(5): connect({1, 3}, {4, 5}, G): draw(G);**

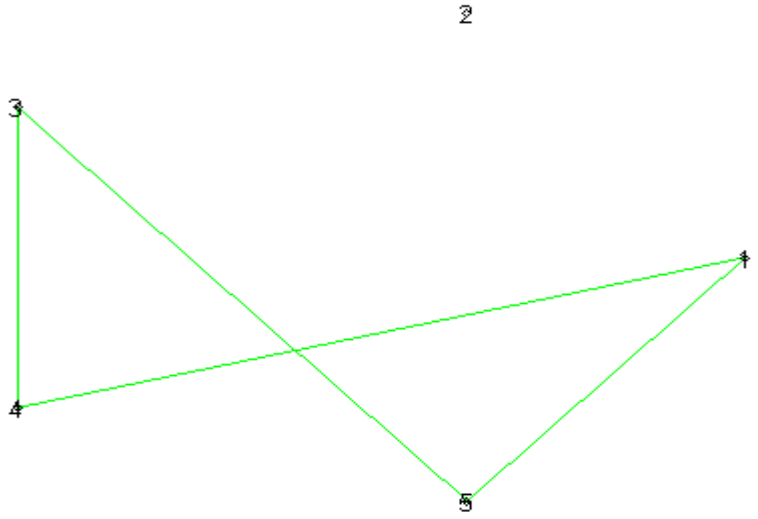


Рис.1.

> **H:=void(6): draw(H);**

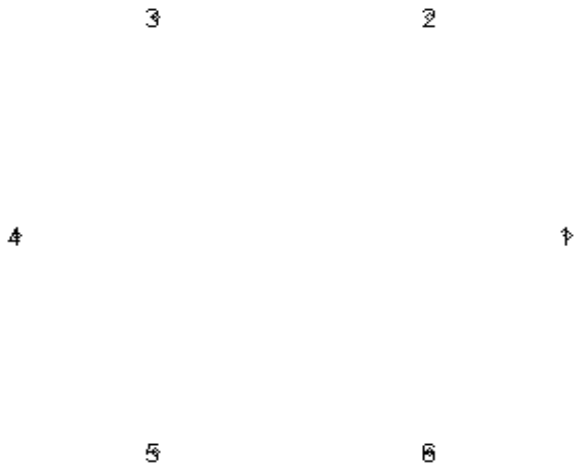


Рис.2.

Граф називається ***повним,*** *якщо кожні дві різні його вершини з'єднані одним і тільки одним ребром.*

Зауваження.

1. У повному графі кожна його вершина належить одному і тому ж числу ребер.

2. Для задання повного графа достатньо знати число його вершин.

Ми будемо створювати повні графи за допомогою спеціальної функції **complete().**

> **G:=complete(5): draw(G);**

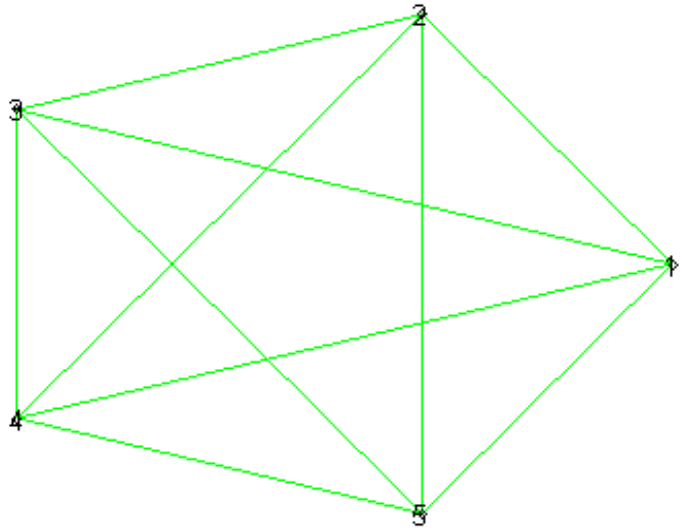


Рис.3.

Граф, який не є повним, можна перетворити в повний із тими самими вершинами, додавши ребра, яких не вистачає.

> **J:=void(4): connect({1, 2}, {3, 4}, J): draw(J);**

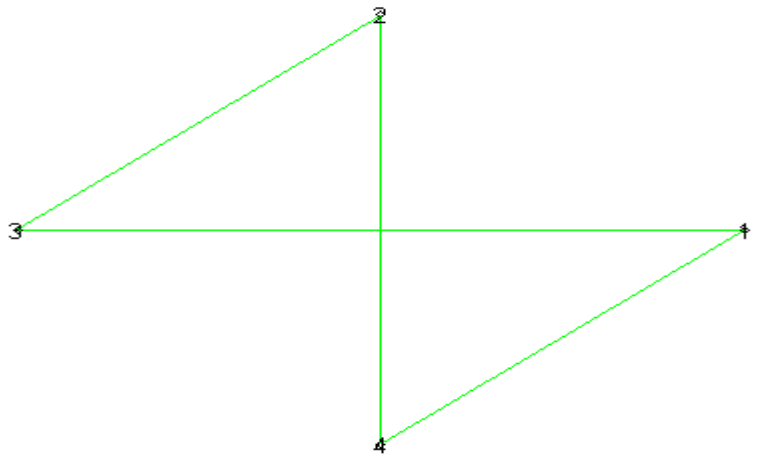


Рис.4.

В граф, зображений на рис. 4, необхідно додати 2 ребра – {1,2} і {3,4}:

> **addedge([{1, 2}, {3, 4}], J): draw(J);**

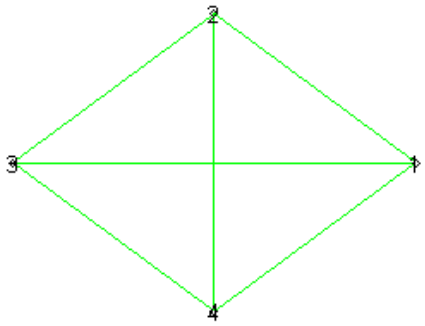


Рис.5.

Вершини графа J і ребра, які добавлені, також утворюють граф, зображений на рис.6. Такий граф називають доповненням графа J.

Доповненням *графа J називається граф з тими самими вершинами, що і граф J і з тими і тільки з тими ребрами, які необхідно добавити до графа, щоб одержати повний граф.*

Доповнення графа задається спеціальною функцією **complement()**:

>**J:=void(4): connect({1, 2}, {3, 4}, J):**
J1:=complement(J): draw(J1);

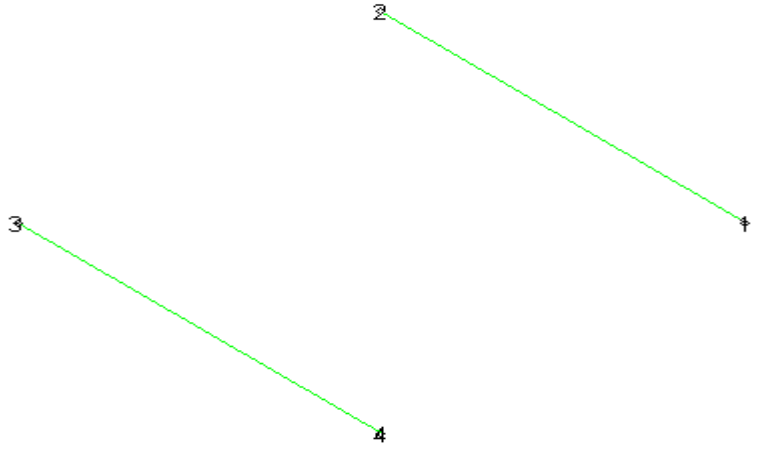


Рис.6.

Завдання для самостійної роботи

1. Нарисуйте повний граф з *n* вершинами і підрахуйте кількість ребер, якщо:
a) n=2; b) n=9; c) n=15.
Дайте відповідь на запитання: скільки ребер у повного графа з *n* вершинами?
2. Побудуйте доповнення графів, зображених на рис.1 і 7.

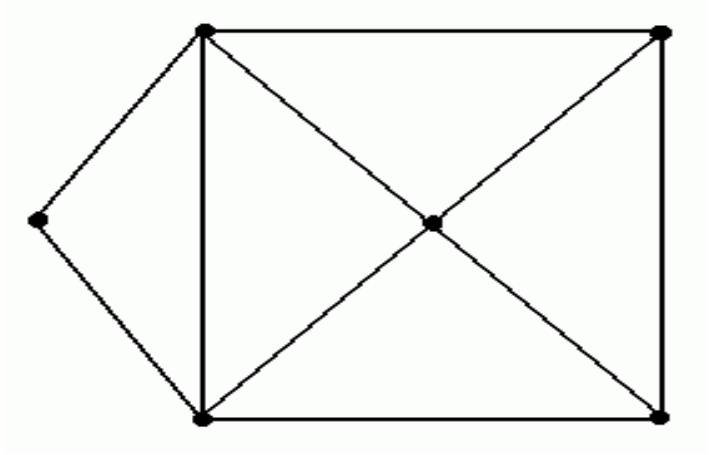


Рис.7.

2. Орієнтовані графи

Повернемось до прикладу змагання футбольних команд (див. лабораторну роботу №1). Ми з'єднували дві команди ребром у тому випадку, якщо ці команди уже грали між собою. Однак такий граф не дає відповіді на одне важливе запитання: хто саме виграв гру?

Цього недоліку легко уникнути, якщо будемо рисувати ребра зі стрілками. Тоді та вершина, в яку направлена стрілка, буде відповідати, наприклад, команді, яка програла (див. рис.8)

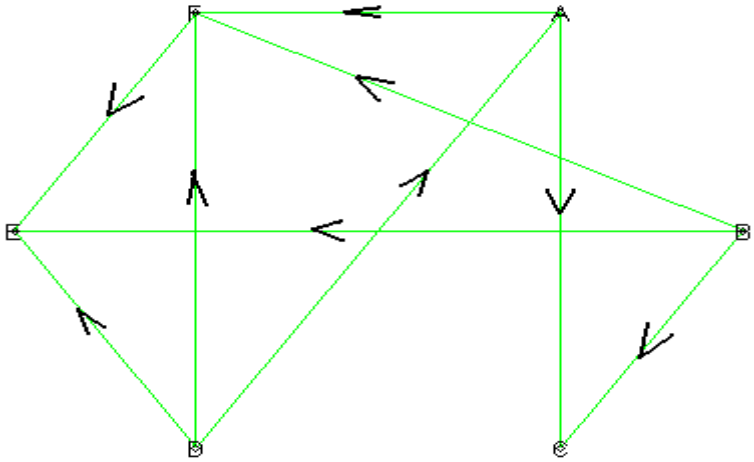


Рис.8.

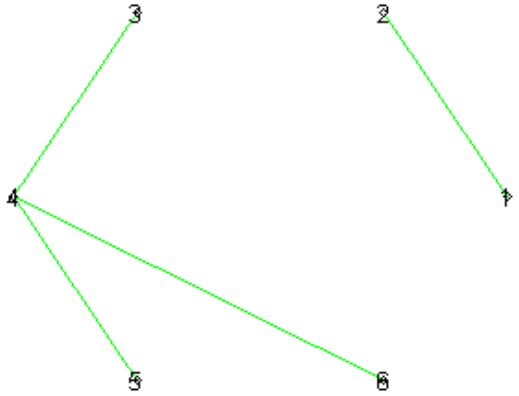
Ребро графа називається ***орієнтованим,*** *якщо одну вершину вважають початком ребра, а другу – кінцем. Граф, усі ребра якого орієнтовані, називається* ***орієнтованим графом*****.**

Якщо команди грають внічию, то ми будемо рисувати неорієнтоване ребро. При цьому ми одержимо так званий ***змішаний граф,*** на якому є як орієнто- вані, так і неорієнтовані ребра.

На жаль, можливості Maple не дозволяють ”рисувати” орієнтовані ребра – вони рисуються як звичайні (неорієнтовані). Але за допомогою деяких функцій можна визначити, чи є ребро орієнтованим. Задаються орієнтовані ребра з допомогою квадратних дужок – [1,2], неорієнтовані (як уже розглядалось раніше) позначаються фігурними – {1,2}.

Розглянемо приклад:

```
> with(networks):
> G:=void(6):  addedge([1, 2], G):
connect({3, 5, 6}, {4}, 'directed',
G):  draw(G);
```



Зауваження. Ребра [A,B] і [B,A] вважаються різними.

Знайдемо початкову вершину ребра [6,4]:

```
> tail(edges([6,4],G),G);
                    {6}
> tail(edges([4,6],G),G);
                    { }
```

Знайдемо кінцеву вершину ребра:

```
> head(edges([6,4],G),G);
                    {4}
> head(edges([4,6],G),G);
                    { }
```

Завдання для самостійної роботи

1. Побудуйте граф, який відповідає наступним результатам змагань: команда A виграла у B і C, і зіграла внічию з D; B програла A, зіграла внічию з C і виграла у D; C програла A, виграла у D і зіграла внічию з B; D зіграла внічию з A і програла B і C. Визначте команду-переможця.
2. Задайте змішаний граф, який відповідає рис.11.

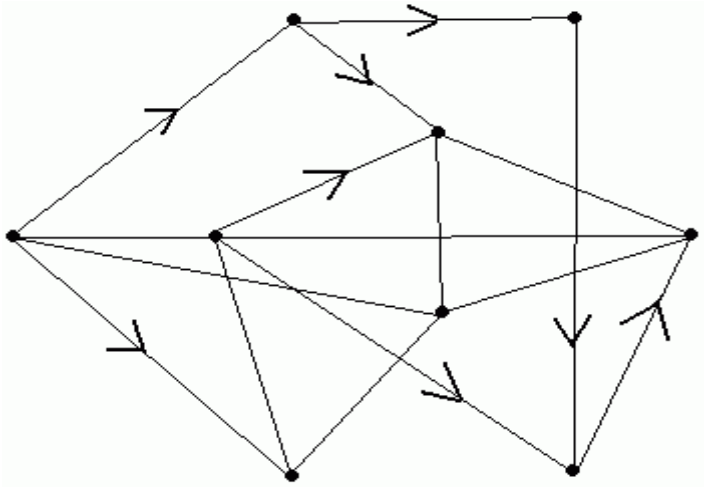


Рис.10.

3. Степінь вершини

Вершини в графі можуть відрізнятися одна від одної тим, скільком ребрам вони належать.

Степенем вершини *називається число ребер графа, яким належить ця вершина.*

Степенем виходу *вершини A орієнтованого графа називають число ребер, які виходять із A.*

Степенем входу *вершини A орієнтованого графа називається число ребер, які входять в A.*

Існує набір функцій, за допомогою яких визначаються степені вершин. Функція **vdegree()** працює з неорієнтованими ребрами, **indegree()** – з орієнтованими ребрами, які входять в дану вершину, а **outdegree()** – з орієнтованими ребрами, які виходять із даної вершини.

```
> with(networks):
> G:=void(7):  connect({1, 2}, {3, 4, 5}, G):
connect({1}, {6, 7}, G):  draw(G);  vdegree(1, G);
indegree(1, G);  outdegree(1, G);
```

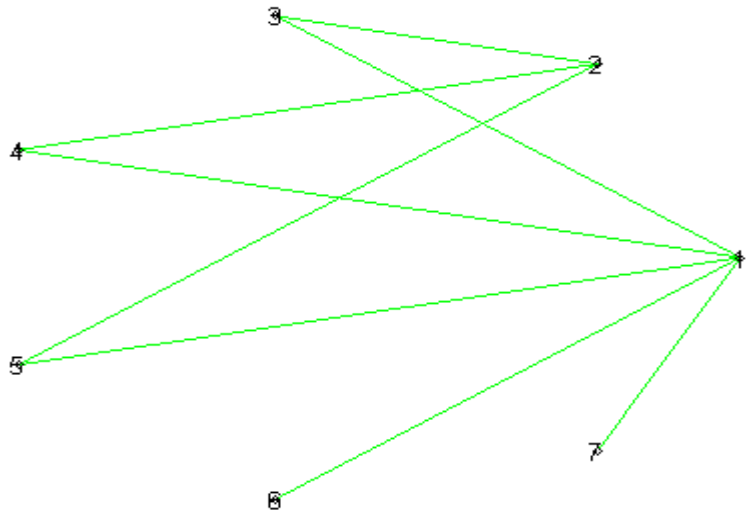


Рис.11.

```
3
0
2
```

Будемо позначати степінь вершини A – p(A).

Вершина називається ***непарною***, якщо її степінь – число непарне. Вершина називається ***парною***, якщо її степінь – число парне.

Число ребер графа ми рахували за допомогою функції **nops()**. Разом з тим можна підрахувати число ребер в кожній вершині $A_1, A_2, \ldots, A_n$ окремо і додати усі ці числа. При цьому кожне ребро буде пораховано два рази – відповідно двом вершинам, які воно з'єднує, тому загальне число ребер графа N буде дорівнювати половині цієї суми:

$$N = \frac{1}{2}(p(A_1) + p(A_2) + \ldots + p(A_n)).$$

Очевидно, що для будь-якого графа *сума степенів усіх його вершин є парним числом,* оскільки вона дорівнює подвоєному числу ребер цього графа. Це легко перевірити. Дійсно, функція **degreeseq()** виводить список усіх степенів вершин графа в порядку зростання, за допомогою функції **add()** можна підраховувати суму усіх степенів вершин графа:

> **degreeseq(G);**

[1, 1, 2, 2, 2, 3, 5]

> **add(i,i=degreeseq(G));**

16

Ми можемо знайти кількість ребер даного графа, розділивши одержану суму навпіл, тобто кількість ребер дорівнює 8. Перевіримо результат за допомогою функції:

> **nops(edges(G));**

8

Розглянемо задачу. Чи можна нарисувати на площині 9 відрізків так, щоб кожний перетинався рівно з трьома іншими?

Припустимо, що можна. Побудуємо граф даної ситуації. Візьмемо за вершини графа дані відрізки. Ребрами будемо з'єднувати ті вершини (відрізки), які перетинаються. Одержимо граф з 9 вершинами і степінь кожної із вершин дорівнює 3. Підрахуємо скільки ребер в одержаному графі:

$$9*3/2=13,5$$

Одержали протиріччя, оскільки число ребер графа – число ціле. Висновок – не можна.

Цікаві факти: 1. Сума степенів входу всіх вершин дорівнює числу ребер графа і дорівнює сумі степенів виходу всіх його вершин.

2. У будь-яком графі з *n* вершинами, де n>1, завжди знайдуться, принаймні, дві вершини з однаковими степенями.

Завдання для самостійної роботи

1. Чи вірне наступне твердження: число непарних вершин довільного графа парне. Чому? Перевірити це на прикладі.
2. Нарисуйте граф з п'ятьма вершинами, у якого рівно дві вершини мають однакову степінь.
3. Чи існують графи, степені вершин яких дорівнюють:
a) 8, 7, 5, 4, 4, 3, 2, 2, 2; b) 8, 7, 6, 5, 4, 4, 3, 2, 1?
Якщо так, то побудуйте їх.
4. На рис. 12 зображено граф. Визначте степінь входу і степінь виходу кожної його вершини.

> **H:=void(7): connect({4, 6, 7}, {2, 3}, 'directed', H):**
addedge([[1, 2], [5, 1]], H):
draw(H);

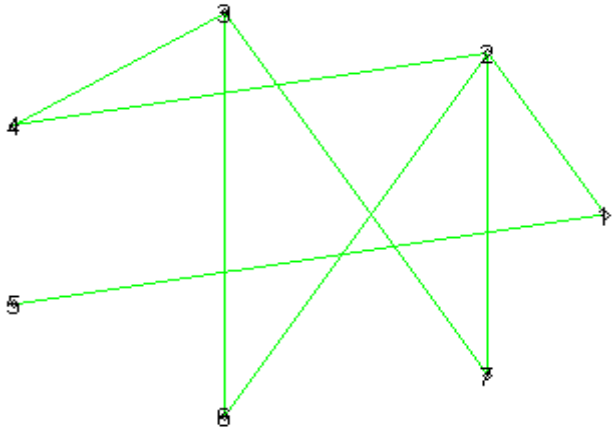


Рис.12.

5. На занятті 20 студентів розв'язали кожний по 6 задач, причому кожна задача була розв'язана рівно двома студентами. Доведіть, що можна організувати аналіз усіх задач так, щоб кожний студент розказав рівно по 3 задачі.

Лабораторна робота №3

Тема: ВЛАСТИВОСТІ ГРАФІВ

Мета роботи: Ознайомитись з основними можливостями програми Maple для вивчення основних властивостей графів.

Зміст роботи:

1. Вивчити можливості програми Maple для побудови графів, що задовольняють деяким властивостям.
2. Ознайомитись із основними властивостями графів.
3. Виконати запропоновані завдання з використанням засобів програми Maple

Зміст звіту: Короткі теоретичні відомості. Постановка індивідуальних завдань та результати їх виконання.

Теоретичні відомості

1. Маршрути, ланцюги та цикли

Маршрутом у заданому графі *G = (V, E)* називається скінченна послідовність його ребер, яка має вигляд

$$(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k).$$

Число *k* ребер маршруту називається ***довжиною*** цього маршруту. Часто можна зустріти і таке означення маршруту в графі.

Послідовність вершин $v_0, v_1, \ldots, v_k$ графа *G = (V, E)* називається ***маршрутом,*** який з'єднує вершини v_0 і v_k, якщо $(v_i, v_{i+1}) \in E, i = 0,1,\ldots,k-1$. В обох випадках вершини $v_0, v_1, \ldots, v_k$ називаються ***вершинами маршруту.***

Маршрут називається ***ланцюгом,*** якщо всі його ребра різні, і ***простим ланцюгом,*** якщо всі його вершини, крім, можливо, першої і останньої, різні. Маршрут називається ***циклічним*** або ***циклом,*** якщо перша і остання його вер-

шини збігаються. Маршрут називається ***простим циклом,*** якщо всі його вершини, крім, можливо, першої і останньої, різні.

Граф називається ***ациклічним графом*** або ***лісом,*** якщо в ньому відсутні цикли. Безпосередньо з означення маршруту і циклу випливають такі твердження.

Твердження 1. *Будь-який маршрут, який з'єднує які-небудь дві вершини, має простий ланцюг, що з'єднує ці вершини*.

Твердження 2. *Будь-який цикл в графі має простий цикл*.

Розглянемо наступний граф:

```
> with(networks):
> H:=void(5): connect({4}, {1, 2, 3, 5}, H):
addedge([{1, 2}, {1, 3}], H): draw(Linear([1], [3, 2], [4], [5]), H);
```

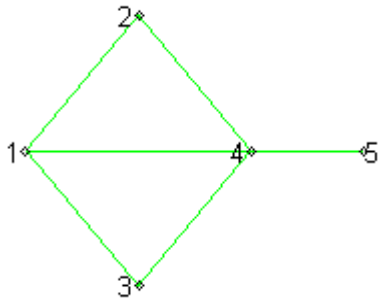


Рис.1.

Як можна пройти ребрами графа, зображеного на рис.1 із вершини 1 у вершину 5?

Для прикладу, можна вказати п'ять послідовностей ребер, рухаючись якими можна потрапити із 1 в 5:

a) {1, 4}, {4, 5}; b) {1, 2}, {2, 4}, {4, 5};

c) {1, 4}, {4, 2}, {2, 1}, {1, 4}, {4, 5};

d) {1, 2}, {2, 4}, {4, 3}, {3, 1}, {1, 4}, {4, 5};

e) {1, 2}, {2, 4}, {4, 1}.

Для створення простого циклу в пакеті Maple користуються функцією **cycle().**

```
> G:=cycle(8):
draw(G);
```

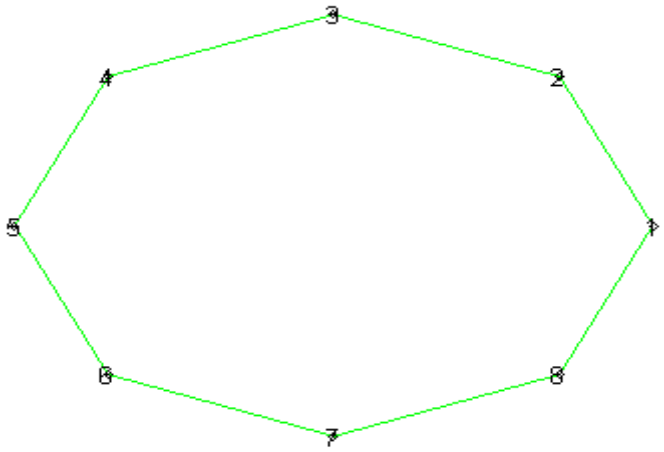


Рис.2.

Завдання для самостійної роботи

1. Знайти ланцюги, які зв’язують вершини 2 і 5 в графі на рис.3. Чи існує простий ланцюг, який з’єднує ці вершини?

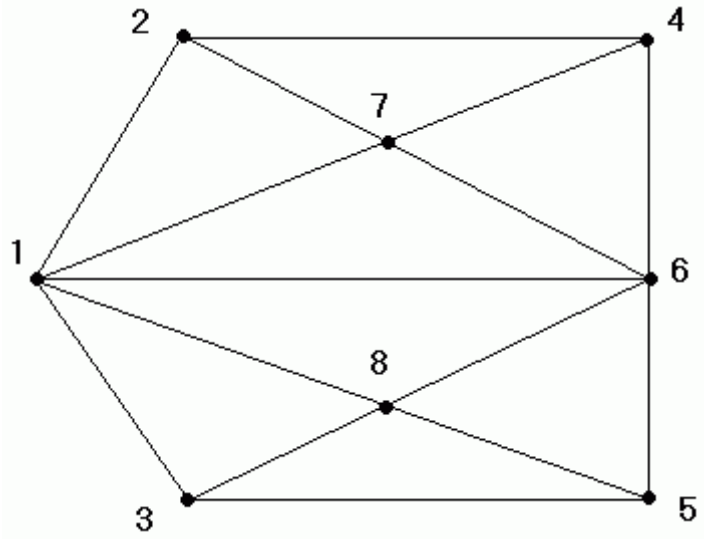


Рис.3.

2. Знайдіть в графі, зображеному на рис.3. цикли, які містять:

а) 4 ребра;

б) 7 ребер;

в) 11 ребер.

Чи є серед них прості?

3. Побудуйте простий цикл з 9 вершинами. Підрахуйте кількість ребер. Скільки ребер містить цикл з 20 вершинами (з *n* вершинами)?

2. Зв’язність графа

Граф називається ***зв'язним,*** якщо довільні дві його вершини зв'язані маршрутом. Зв'язний підграф H графа G називається ***максимальним,*** якщо H не міститься в жодному зв'язному підграфі графа G. Максимальний зв'язний підграф графа називається ***компонентом зв'язності***.

Нехай $G=\{V,E\}$ і $H=\{V_1,E_1\}$ – задані графи. Об’єднання графів $G\cup H$ називається ***диз’юнктивним***, якщо $V\cap V_1=\varnothing$.

Теорема. *Граф зв’язний тоді і тільки тоді, коли його не можна представити у вигляді диз’юнктивного об’єднання двох графів.*

Розглянемо задачу: чи може трапитись, що в одній компанії із шести чоловік кожний знайомий з двома і тільки двома іншими?

Учасника цієї компанії зобразимо вершиною графа, а відношення знайомства між двома учасниками – ребром. Зобразимо графи, які можуть відповідати такій компанії.

```
> with(networks):
> G1:=cycle(6):  draw(G1);
```

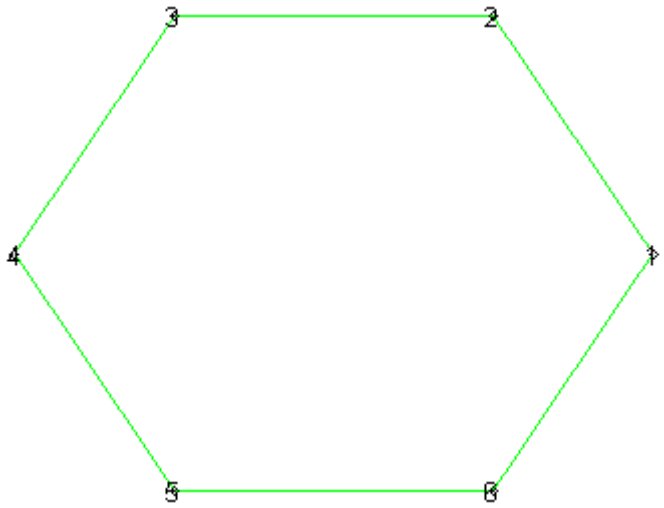


Рис.4.

> **G2:=void(6): addedge([{1, 2}, {2, 3}, {3, 1}, {4, 5}, {5, 6}, {6, 4}], G2): draw(G2);**

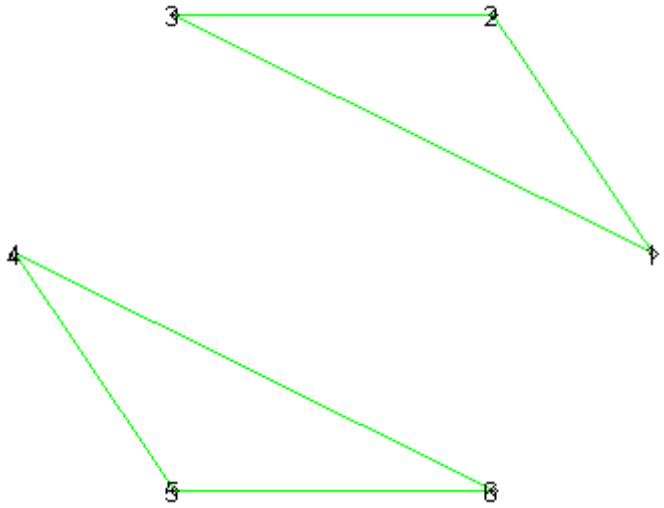


Рис. 5.

Таким чином, ситуація, розглянута в задачі, цілком можлива (рис.4.). Але випадок, розглянутий на рис.5, відповідає не одній, а двом компаніям, де учасники одної із них не знайомі з учасниками другої.

Про граф, зображений на рис.4 говорять, що він зв’язний, оскільки із кожної вершини рухаючись ребрами можна потрапити в будь-яку іншу вершину. Робимо висновок, що в цьому випадку кожен через своїх знайомих може познайомиться із всіма іншими.

У другому випадку одержали дві окремі частини, які не зв’язані між собою ребрами. Такий граф називається ***незв’язним*** , а "частини" – ***компонентами зв’язності.*** Кожна компонента – зв’язний граф.

Функція **components()** знаходить компоненти зв’язності графа, виводячи списки усіх ребер кожної компоненти:

> **components(G2);**

$$\{\{1, 2, 3\}, \{4, 5, 6\}\}$$

> **G3:=void(9): connect({1}, {2, 6, 4}, G3):**
connect({5}, {9, 8}, G3): components(G3); draw(G3);

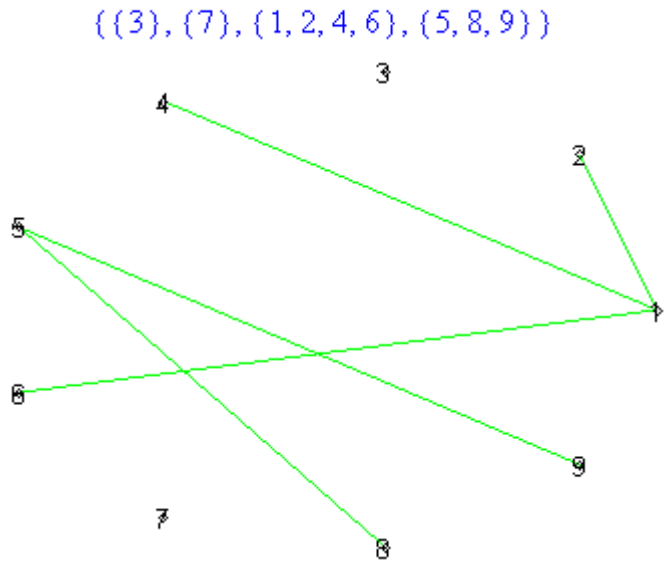


Рис.6.

Розглянемо операцію вилучення ребра із графа, яка здійснюється функцією **delete()**.

При вилученні ребра {A,B} із графа G одержується граф із тими самими вершинами, що і граф G, і всіма ребрами, крім ребра {A,B}.

При вилученні ребра із зв'язного графа, новий граф може виявитися як зв'язним, так і незв'язним.

Ребро {A,B} називається ***мостом*** *графа G, якщо в графі, одержаному після вилучення із G ребра {A,B}, вершини A і B виявляються незв'язними.*

Завдання для самостійної роботи

1. Побудуйте граф з п'ятьма вершинами, який не є зв'язним.
2. "Дорисуйте" граф так, щоб він став зв'язним.

> **J:=void(9): connect({1, 2}, {6, 7, 8}, J):**
connect({3}, {4, 5}, J): draw(J);

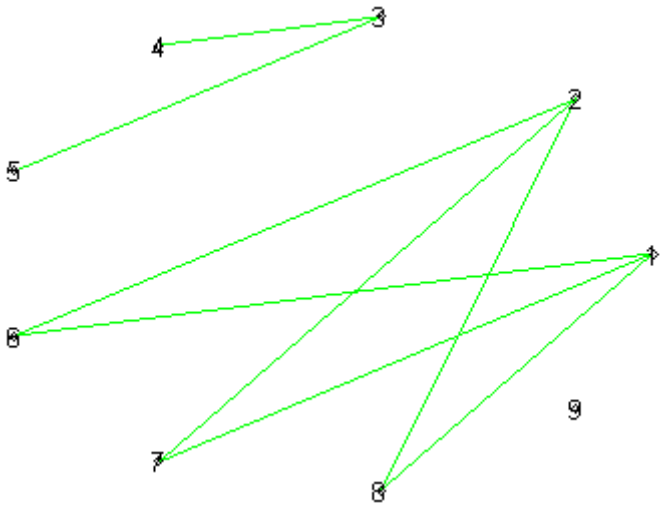


Рис.7.

3. Побудуйте довільний граф і знайдіть, якщо є, мости.
4. Між 9 планетами Сонячної системи введено космічне сполучення. Ракети літають за таким маршрутом: Земля-Меркурій, Плутон-Венера, Земля-Плутон, Плутон-Меркурій, Меркурій-Венера, Уран-Нептун, Нептун-Сатурн, Сатурн-Юпітер, Юпітер-Марс і Марс-Уран. Чи можна добратися із Землі до Марса?

3. Різновиди графів

Розглянемо декілька найбільш відомих різновидів графів.

Граф, будь-які дві вершини якого суміжні, називається ***повним графом***.

Граф, у якого усі вершини мають одну і ту ж степінь r, називається ***регулярним графом*** (або ***граф, регулярний степені r***). Наприклад, на рис. 8 – кубічний граф, а на рис. 9 (див. нижче) – граф, регулярний степені 4.

```
> with(networks):
> G:=void(4): connect({1}, {2, 3, 4}, G):
addedge(Cycle(2, 3, 4), G): draw(Linear([4], [1, 3], [2]), G);
```

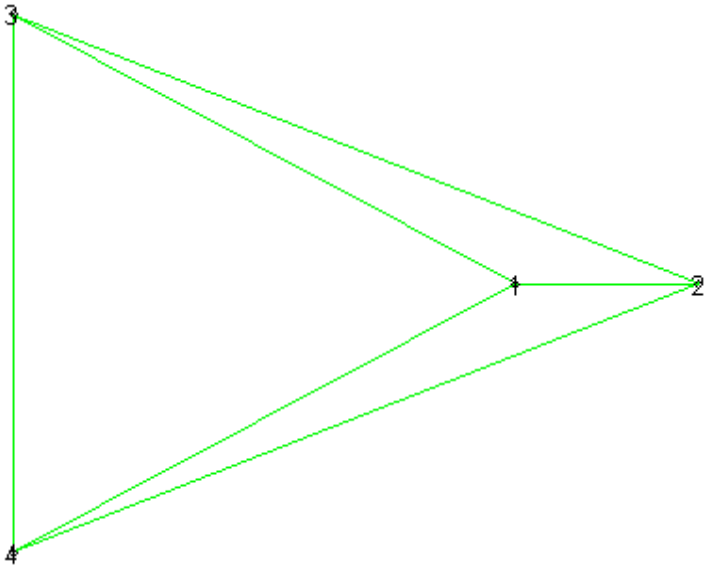


Рис.8.

```
> H:=void(5): connect({1}, {2, 3, 4, 5}, H):
connect({2}, {3, 4, 5}, H): addedge(Cycle(3, 4, 5), H): draw(H);
```

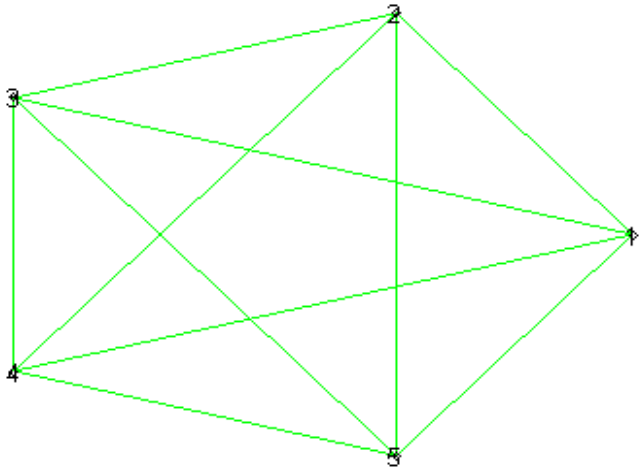


Рис.9.

Відомим прикладом кубічного графа є так званий ***граф Петерсена,*** показаний на рис.10. Для цього дуже відомого графа існує навіть спеціальна функція – **petersen()**. Зауважимо, що графи однакового степеня регулярності можуть бути зовсім різними (див. рис. 8 і 10).

> **P:=petersen(): draw(P);**

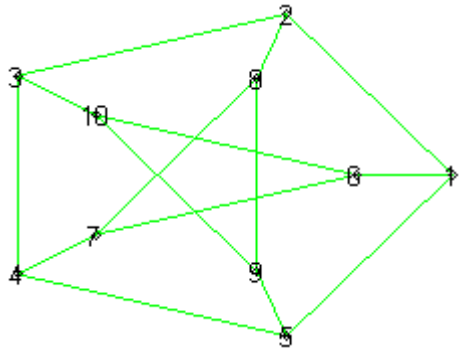


Рис.10.

Серед регулярних графів особливо цікаві ***платонові графи*** – графи, утворені вершинами і ребрами п’яти правильних многокутників – платонових тіл: тетраедра, куба, октаедра, додекаедра і ікосаедра. Для побудови цих графів в Maple є спеціальні функції:

> **Tetr:=tetrahedron(): draw(Tetr);**

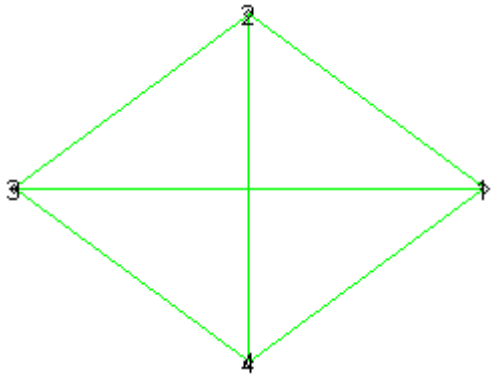


Рис.11 (тетраедр).

> **C:=cube(3): draw(Concentric([0, 1, 3, 2], [4, 5, 7, 6]), C);**

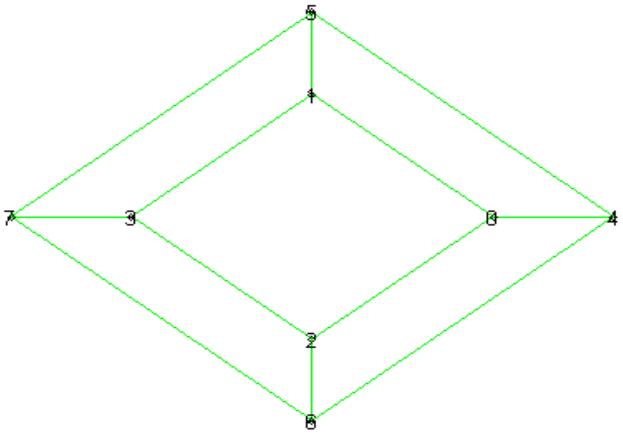


Рис.12 (куб).

> **Oct:=octahedron(): draw(Concentric([1, 2, 0, 3, 4, 5]), Oct);**

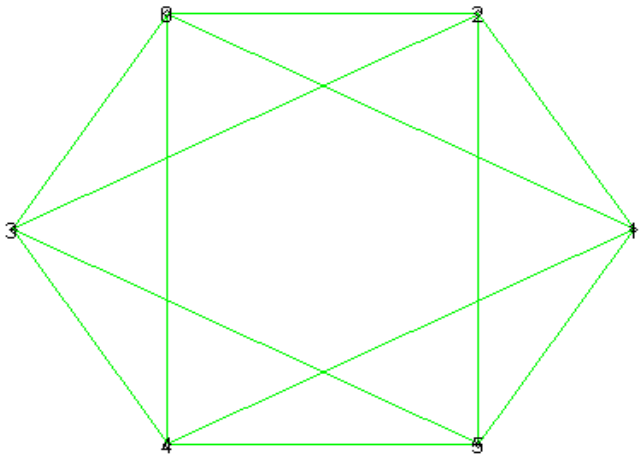


Рис.13 (октаедр).

> **Dod:=dodecahedron():**
draw(Concentric([0, 1, 2, 3, 4], [5, 6, 7, 8, 9], [10, 11, 12, 13, 14], [15, 16, 17, 18, 19]), Dod);

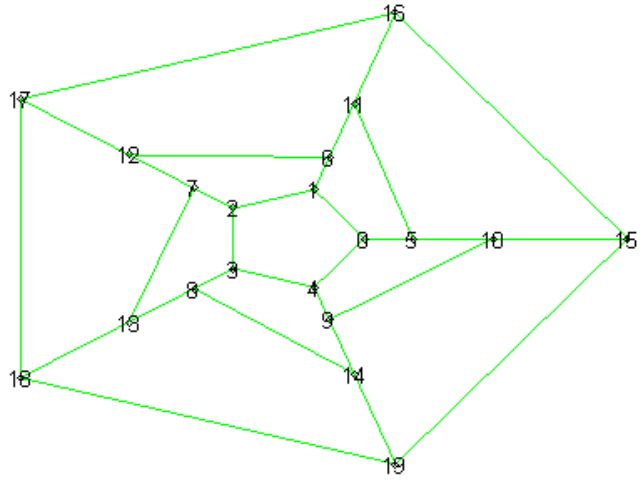


Рис.14 (додекаедр).

> **Ico:=icosahedron(): draw(Ico);**

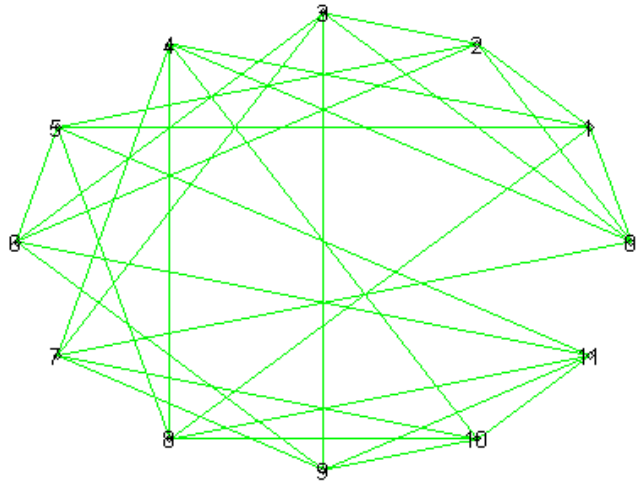


Рис.15 (ікосаедр).

Завдання для самостійної роботи

1. Чи є нуль-граф регулярним?
2. Побудуйте регулярний граф степені 7.
3. Підрахуйте кількість вершин і ребер усіх платонових графів, а також визначте степінь регулярності кожного із них.

Припустимо, що множину вершин графа можна розбити на дві підмножини V_1 і V_2, які не перетинаються так, що кожне ребро в G з'єднує яку-небудь вершину із V_1 з якою-небудь вершиною із V_2. Тоді G називається ***дводольним графом***. На рис.16 показано дводольний граф, у якого V_1={1, 3, 5} і V_2 ={2, 4}.

> **with(networks):**
> **G:=void(5): addedge(Path(1, 2, 3, 4, 5), G): draw(Linear([1, 3, 5], [2, 4]), G);**

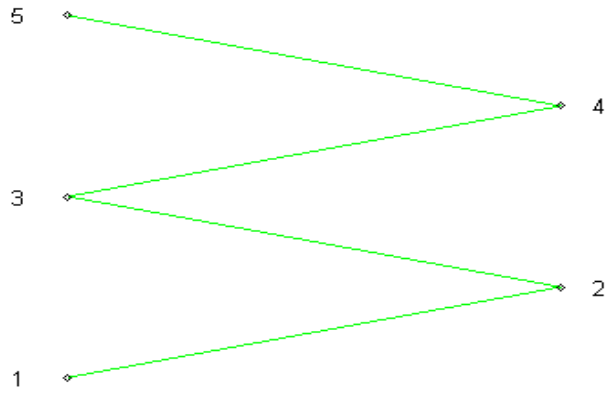


Рис.16.

Якщо вершини дводольного графа розфарбувати двома кольорами, наприклад, червоним і синім, то будь-яке його ребро має один кінець червоний, а другий – синій. Слід підкреслити, що в дводольному графі зовсім не обов'язково, щоб кожна вершина із V_1 була з'єднана з кожною вершиною із V_2; якщо ж це так, то він називається ***повним дводольним графом*** і позначається через $K_{m,n}$ (m і n – число вершин в V_1 і V_2). На рис.17 зображено граф $K_{3,4}$:

> **H:=void(7): connect({1}, {4, 5, 6, 7}, H): connect({2},{4,5,6,7},H): connect({3}, {4, 5, 6, 7}, H): draw(Linear([3, 2, 1], [7, 6, 5, 4]), H);**

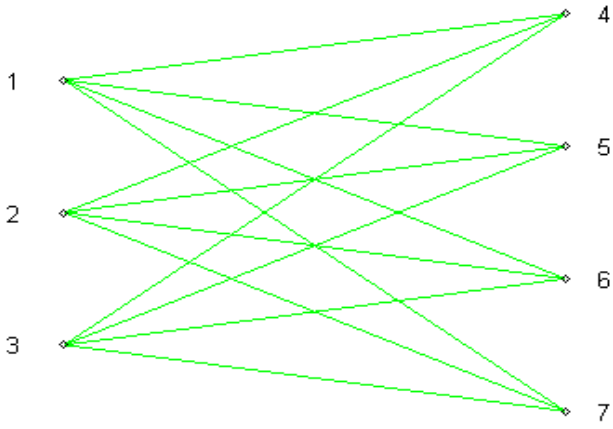


Рис.17.

На рис.18 маємо граф $K_{3,3}$:

> J:=void(6): connect({1}, {4, 5, 6}, J): connect({2}, {4, 5, 6}, J): connect({3}, {4, 5, 6}, J): draw(Linear([3, 2, 1], [6, 5, 4]), J);

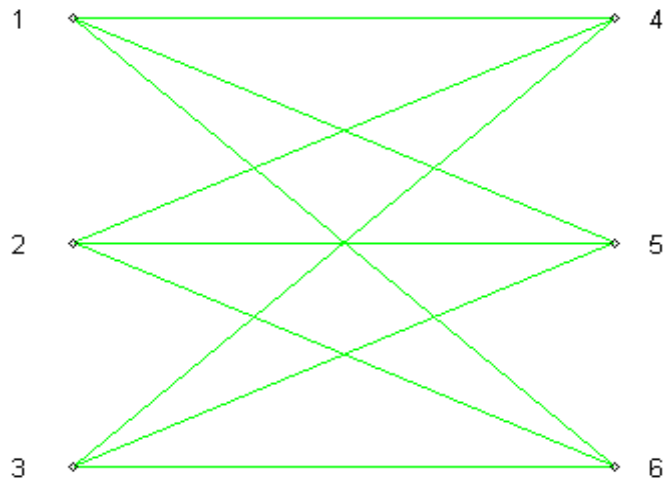


Рис.18.

Повний дводольний граф вигляду $K_{1,n}$ називається ***зірковим графом*** (див. рис.19 – граф $K_{1,5}$).

> L:=void(6): connect({1}, {2, 3, 4, 5, 6}, L): draw(Concentric([1], [2, 3, 4, 5, 6]), L);

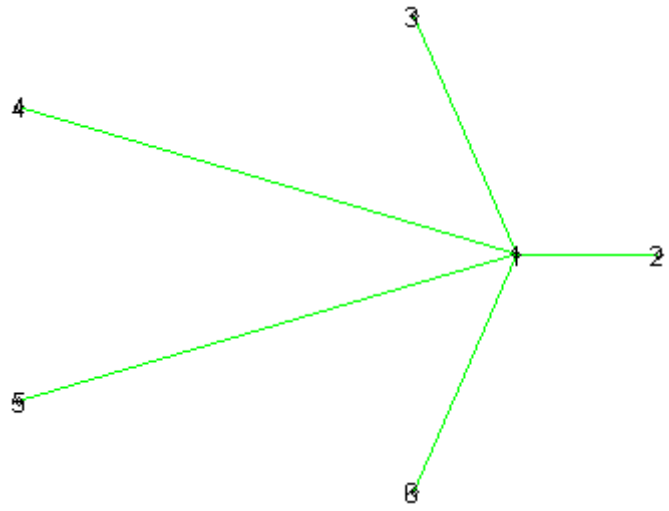


Рис.19.

Завдання для самостійної роботи

1. Підрахуйте кількість ребер і вершин графа $K_{m,n}$.
2. Побудуйте графи $K_{2,5}$ і $K_{7,4}$. Підрахуйте кількість їх ребер і вершин.

Лабораторна робота №4

Тема: Ейлерові та гамільтонові графи

Мета роботи: Вивчити можливості програми Maple для відшукання найкоротших та ейлерових шляхів і циклів та програми Mathcad для розв'язання задачі комівояжера алгебраїчним алгоритмом Йоу.

Зміст роботи:

1. Вивчити можливості програми Maple для перевірки графа на існування в ньому ейлеревого циклу.
2. Ознайомитись з можливостями програми Maple для відшукання найкоротших шляхів у графі алгоритмом Дейкстри.
3. Використання програми Mathcad для розв'язання задачі комівояжера алгебраїчним алгоритмом Йоу.
4. Виконати запропоновані завдання з використанням засобів програм Maple і Mathcad.

Зміст звіту: Короткі теоретичні відомості. Постановка індивідуальних завдань та результати їх виконання.

Теоретичні відомості

1. Ейлерові графи, цикли і шляхи

Історично теорії графів, як розділу математики, зародилася при розв'язуванні Ейлером задачі про кенігсбергські мости. Ця знаменита у свій час задача полягає в наступному. Сім мостів міста Кенігсберг були розташовані на річці Прегель так, як зображено на рис. 1. Питається, чи можна вийти із дому, пройти через кожен міст точно по одному разі і повернутися назад.

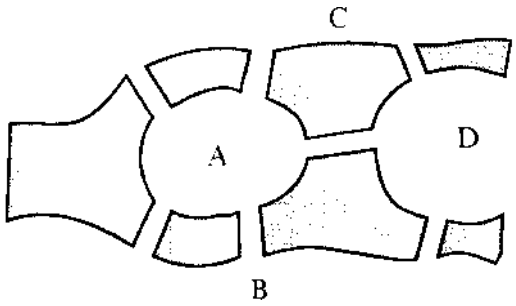


Рис. 1

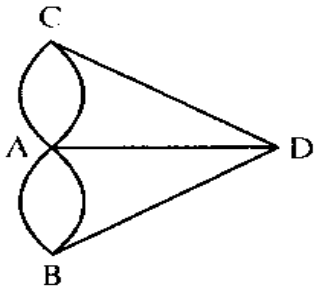


Рис. 2

Розв'язання даної задачі розглянемо нижче, а зараз зауважимо, що в результаті розв'язання цієї задачі з'явився ще один тип графів – ейлерові графи.

Для розв'язання поставленої задачі співставимо плану міста граф G, вершини якого відповідають чотирьом розділеним рікою ділянок суші A, B, C і D , а ребра – мостам. Цей граф (точніше, мультиграф) зображено на рис. 2. Повністю побачити цей граф за допомогою функції **draw()** ми не можемо, тому що в графі є кратні ребра.

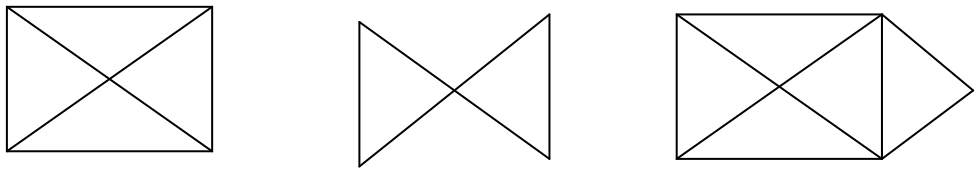
Щоб відповісти на запитання задачі, досить з'ясувати чи є граф ейлеровим, тобто достатньо з'ясувати чи є в мультиграфі G цикл, що містить усі ребра цього мультиграфа?

Ейлер довів нерозв'язність задачі про кенігсбергські мости. У своїй роботі, опублікованій в 1836 році, він сформулював і розв'язав наступну загальну проблему теорії графів: при яких умовах зв'язний граф містить цикл, що проходить через кожне його ребро?

Ейлеровим *ланцюгом* у графі називається маршрут, що містить усі ребра графа (тобто маршрут проходить по кожному ребру рівно по одному разу, але не обов'язково замкнутий). ***Ейлеровим циклом*** у графі називається цикл, що містить усі ребра графа (тобто це замкнутий ейлерів ланцюг). Граф, що володіє ейлеровим циклом, називається ***ейлеровим графом.*** Такий граф можна нарисувати, не відриваючи олівця від паперу і не повторюючи ліній.

Зв'язний граф G називається напівейлеровим, якщо в ньому існує ланцюг, який включає кожне його ребро. Таким чином, всякий ейлерів граф буде напівейлеровим.

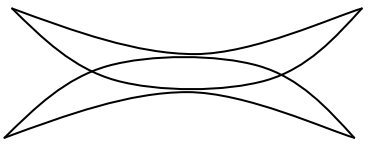
На рис.3 наведені графи: неейлерів (а), ейлерів (б) і напівейлерів (в).



а *б* *в*

Рис. 3. Різновиди графів

Крім задачі про кенігсбергські мости, відомо ряд інших цікавих задач (головоломок), розв'язання яких зводиться до вияснення питання "чи є граф ейлерів?". В одній із них потрібно обвести фігуру, яка називається *шаблями* (*знаком*) *Магомета* (рис. 4), не відриваючи олівця від паперу і не повторюючи ліній.



Виникає природне питання: як встановити, що заданий граф є ейлеровим? Відповідь на це питання дають такі теореми.

Теорема 1. *Якщо степінь кожної вершини скінченного графа не менше двох, то граф має цикл.*

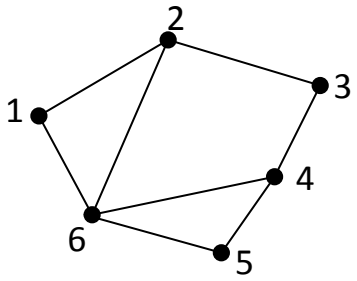
Доведення. Нехай v – довільна вершина графа G. Користуючись методом побудови за індукцією, побудуємо маршрут $v, v_1, v_2, \ldots, v_k, \ldots$ так, що v_{i+1} суміжна з v_i і відмінна від v_{i-1}. Існування такої вершини випливає з умов теореми. Оскільки граф G скінченний, то на деякому кроці побудови нашого маршруту прийдемо до вершини, вибраної раніше. Нехай v_k – перша з таких вершин. Тоді частина маршруту, яка лежить між першою і другою появою v_k в цьому маршруті, і є шуканим циклом. Теорема доведена.

Зауважимо, що дана теорема справедлива для псевдографа (який має петлі) і для мультиграфа (який має кратні ребра).

Теорема 2. *Зв'язний граф G є ейлеровим тоді і тільки тоді, коли кожна вершина G має парний степінь*.

Для встановлення степені вершин графа в програмі Maple є функція **degreeseq(G)**.

Приклад 1. Користуючись програмою Maple перевірити чи графи, зображені на рис. 2 і 5, є ейлеровими.



Для графа, зображеного на рис. 5 наявність ейлеревого циклу легко встановити – це цикл {1, 2, 3, 4, 5, 6, 4, 2, 6, 1}. У цьому графі є й інші ейлерові цикли. Однак, будь-які два такі цикли відрізняються один від другого тільки порядком обходу ребер.

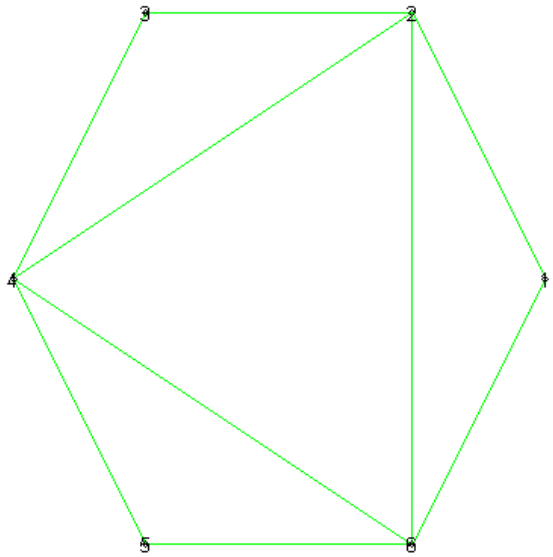
Що стосується графа наведеного на рис. 2, то встановити відсутність в цьому графі ейлеревого циклу безпосередньою перевіркою теж можна, але трохи важче, ніж у попередньому випадку.

Дослідимо задані графи за допомогою програми Maple. Для цього задамо графи і знайдемо степені всіх їх вершин.

Програми та результати їх роботи наведено нижче

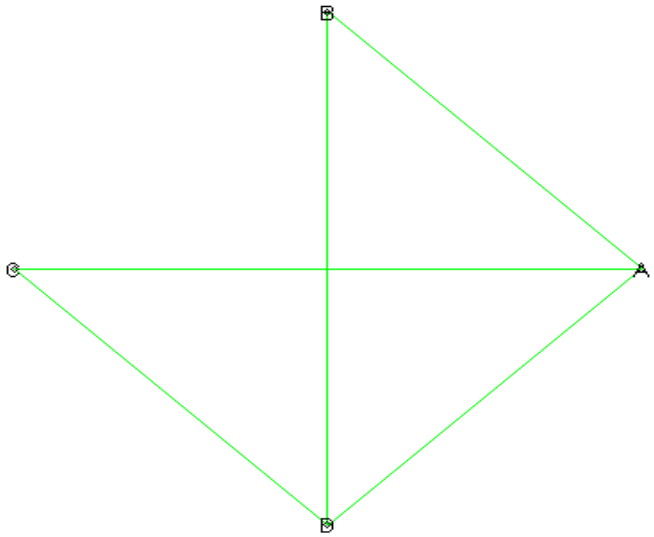
> **with(networks): G:=void(6): connect({2}, {3, 4, 6, 1}, G): connect({4}, {3,5,6}, G): connect({6}, {1, 5},G): degreeseq(G); draw(G);**

[2, 2, 2, 4, 4, 4]



> **K:=new(): addvertex({A,B,C,D},K): connect({D}, {A, B, C}, K): connect({A}, {B}, K): connect({B}, {A}, K): connect({A}, {C}, K): connect({C}, {A}, K): degreeseq(K); draw(K);**

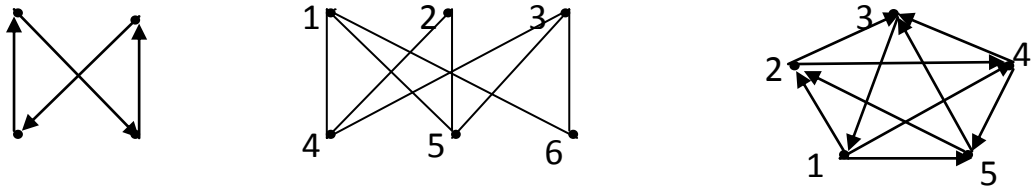
[3, 3, 3, 5]



З одержаних результатів бачимо, що в графі G усі вершини мають парні степені, а це означає, що ейлерів цикл існує. У випадку графа K (кенігсбергські мости) степені усіх вершин непарні, а отже граф не має ейлеревих циклів. Зауважимо, що повністю побачити граф K за допомогою функції **draw()** ми не можемо, оскільки в графі є кратні ребра (K – мультиграф).

2. Гамільтонові графи, цикли і шляхи. Задача комівояжера

Гамільтоновим циклом у графі $G=(V,E)$ називаєтьєся *простий цикл* $Q=(V,E_0)$, що містить всі вершини графа, незалежно від того чи є G орієнтованим. Вимога простоти циклу є принциповою: за гамільтоновим циклом Q можна обійти всі вершини v графа G, відвідуючи кожну проміжну (тобто не початкову і не кінцеву) вершину тільки один раз. Сам граф G, в якому існує гамільтонів цикл, називається ***гамільтоновим графом.*** Зрозуміло, що гамільтонів граф є зв'язним і, більше того, *двозв'язним,* оскільки між кожною парою вершин існує не менше, ніж два різних простих ланцюги. При будь-якому $n\geq 3$ повний простий граф K_n є гамільтоновим. Повний двочастинний граф $K_{p,p}$ з рівнопотужними частинами (однокольоровими множинами) також гамільтонів, див. $K_{2,2}, K_{3,3}, K_5$ на рис. 4.



$K_{2,2}$ $K_{3,3}$ K_5

Рис. 4. Гамільтонові графи

Безпосередньо перевіряється, що графи G_1, G_2, G_3 на рис. 5 не містять гамільтонових циклів і, отже, не є гамільтоновими графами. Однак всі ці графи містять Гамільтонові ланцюги.

Гамільтоновим ланцюгом у графі $G=(V,E)$ називається простий ланцюг

$Q = (V, E_2)$, що містить всі вершини графа. Гамільтоновим ланцюгом у графі G_1 є ланцюг (2,3,5,1,4), у графі G_2 – ланцюг (6,2,3,4,5,1), у графі G_3 – ланцюг (6,4,3,2,1,5), а також (2,6,4,3,1,5).

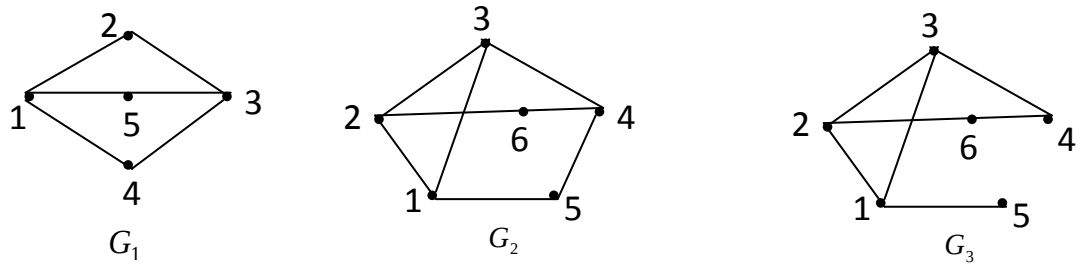


Рис. 5. Гамільтонові ланцюги

На рис. 6 зображено два негамільтонових графи $K_{2,3}$, $K_{3,4}$. Обидва вони двозв'язні. В $K_{2,3}$ існують гамільтонові ланцюги, наприклад, (4, 1, 5, 2, 3), (3, 1, 4, 2, 5), (4, 2, 5, 1, 3). В графі $K_{3,4}$ немає ані гамільтонових циклів, ані гамільтонових ланцюгів.

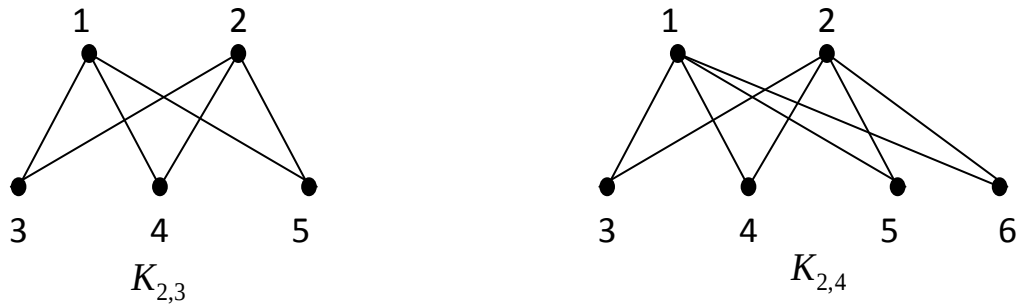


Рис. 6. Негамільтонові графи

Завдання для самостійної роботи

1. На рис. 7 зображена решітка. Чи можна провести неперервну лінію, що перетинає точно по одному разу кожну сторону решітки?

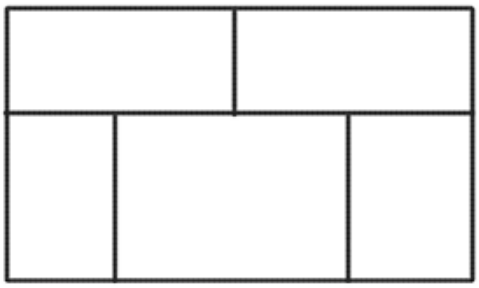


Рис.7.

Створимо граф, у якого вершини – це області, відзначені на рис. 8 чорними крапками, а ребра – з'єднуючі їх лінії. Ці лінії перетинають сторони решітки, а також з'єднують крапки, що знаходяться за межами решітки між собою (і не перетинають решітки). Таким чином, крім ліній, проведених на рисунку, треба провести ще лінії, що з'єднують кожну зовнішню крапку з всіма іншими зовнішніми крапками. Таким чином, до кожної такої крапки

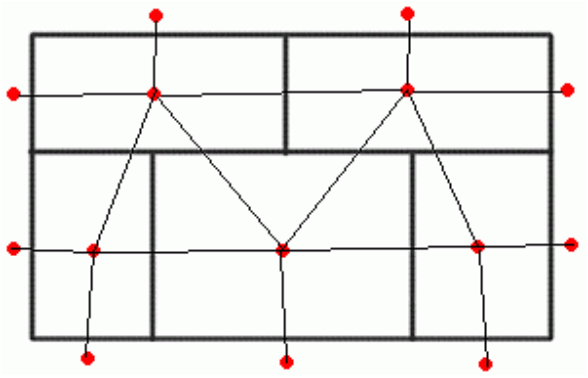


Рис.8

добавиться по 8 ребер. Ці лінії будуть означати "вихід" неперервної лінії за межі решітки. Таким чином, задача зводиться до відшукання ейлеревого шляху в отриманому графі. Створимо цей граф і виведемо список степенів усіх його вершин.

> **M:=void(14): connect({1}, {2, 3, 4, 5, 6, 7, 8, 9, 10}, M):**
connect({2}, {3, 4, 5, 6, 7, 8, 9, 10}, M): connect({3},{4, 5, 6, 7, 8, 9, 11},M):
connect({4}, {5, 6, 7, 8, 9, 11}, M): connect({5}, {6, 7, 8, 9, 12}, M):
connect({6}, {7, 8, 9, 12}, M): connect({7}, {8, 9, 13}, M):
connect({8}, {9, 14}, M): connect({14}, {9, 10, 13}, M):
connect({13}, {10, 11, 12}, M): connect({11}, {10, 12}, M):
draw(Concentric([12, 11, 10, 14, 13], [5, 4, 3, 2, 1, 9, 8, 7, 6]), M);
degreeseq(M);

З одержаного результату зробити висновок чи буде даний граф містити ейлерів цикл, а отже чи можна провести неперервну лінію, яка перетинає точно по одному разу кожну сторону решітки.

2. Поле розбите межами на кілька ділянок (див. рис. 9). Землемір захотів, не виходячи за межі поля, пройти по ньому так, щоб перетнути кожну межу рівно один раз. Чи удасться йому це?

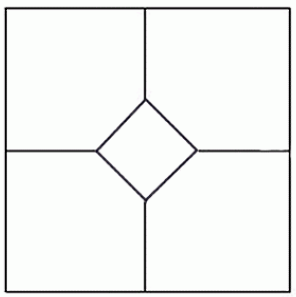


Рис.9.

3. Перевірити всі платонові графи на наявність у них ейлеревого циклу або шляху.

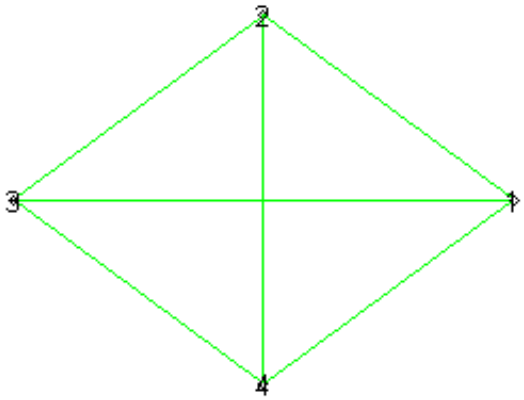
3. Задача про існування найкоротшого шляху в графі

Широко поширені задачі існування найкоротшого шляху. У таких задачах задається граф (мережа доріг) і початкова вершина (пункт відправлення). Кожному ребру можна привласнити вагу довжину дороги. Крім цього, ребра мо-

жуть бути як орієнтованими, так і неорієнтованими. Задача знаходження найкоротшого шляху розв'язується за допомогою алгоритму Дейкстри. Результат роботи алгоритму – дерево з початком у початковій вершині, причому до всіх інших вершин йдуть найкоротші шляхи. У Maple задачу існування найкоротшого шляху можна розв'язати за допомогою функції **shortpathtree()**.

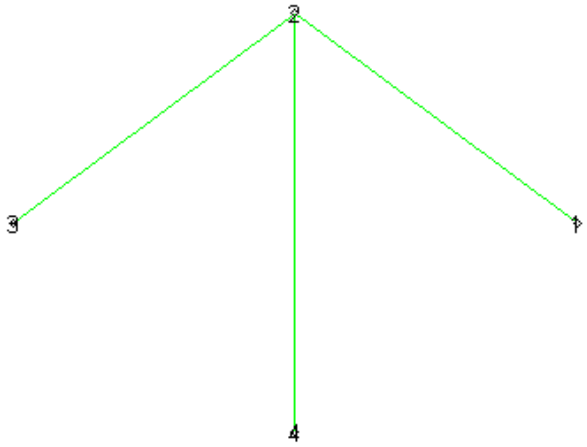
У наступному прикладі створимо повний граф з чотирма вершинами, причому вага кожного ребра дорівнює 1 (за замовчуванням):

> **G:=complete(4): draw(G);**



Одержимо граф G1 із графа G, застосувавши команду **shortpathtree()**:

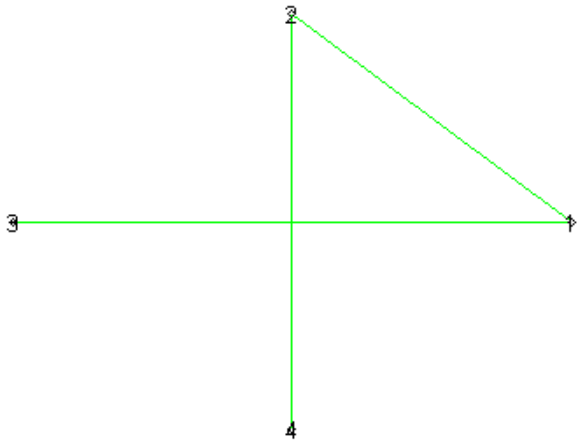
> **G1:=shortpathtree(G, 2): draw(G1);**



Таким чином, G1 – дерево, отримане в результаті роботи алгоритму Дейкстри. Очевидно, що з вершини 2 у вершину 1 найшвидше можна потрапити по ребру {2,1}.

Тепер змінимо вихідний граф G: ребру, що з'єднує вершини 2 і 3 привласнимо вагу, рівну 100. Ми видалимо ребро {2,3} (вага якого за замовчуванням дорівнює 1) і відновимо його з вагою 100:

> **d:=edges({2, 3}, G): delete(d[1], G):**
addedge({2, 3}, weights=100, G):
> **H:=shortpathtree(G, 2): draw(H);**



Таким чином, у зміненому графі знайдені нові найкоротші шляхи до усіх вершин з вершини 2.

Функція **shortpathtree()** привласнює довжини найкоротших шляхів вагам вершин. Скористаємося цим і за допомогою наступної команди одержимо інформацію про відстані від вершини 2 до вершин 1, 2, 3, 4:

> **vweight(H);**

```
table(sparse, [
1 = 1
2 = 0
3 = 2
4 = 1
])
```

З даної таблиці видно, що для того, щоб потрапити з вершини 2 у вершину 3, потрібно пройти відстань рівну двом (тобто два ребра з одиничними вагами).

Завдання для самостійної роботи

Дано карту доріг між містами, де зазначена довжина кожної дороги (дані не збігаються з реальними). Знайти: а) усі найкоротші шляхи із Санкт-Петербурга до Омська; б) усі найкоротші шляхи із Санкт-Петербурга до Магнітогорська.

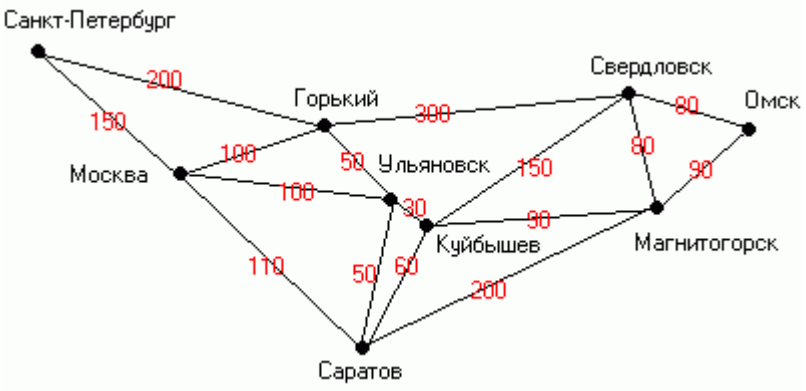


Рис.10.

Задача комівояжера

Загальна задача комівояжера полягає в наступному: використовуючи задану систему транспортних сполучень (доріг і т.п.) між пунктами (містами, фі-

рмами і т.п.) у конкретній зоні обслуговування, відвідати всі пункти у такій послідовності, щоб пройдений маршрут був найкоротшим із всіх можливих.

На мові теорії графів або мереж загальна задача комівояжера має таке формулювання: у зваженому зв'язному графі *G* знайти найкоротший маршрут, що проходить через усі вершини графа. У постановці задачі можна додати вимоги *замкненості* маршруту комівояжера (повернення комівояжера у пункт вихідного перебування). У такій постановці задача завжди має розв'язок.

Існує ще одна постановка, в якій додатково до попередньої задачі треба, щоб кожний пункт обслуговування комівояжер відвідував *тільки один раз*. Очевидно, що в такій постановці задача комівояжере не завжди має розв'язок, а якщо має, то маршрут комівояжера в графі *G* є найкоротшим гамільтоновим ланцюгом або циклом. Останню постановку називають "***гамільтоновою*" *задачею комівояжера*** (на відміну від загальної). Якщо шукати замкнений маршрут, то для розв'язності "гамільтонової" задачі комівояжера необхідно і достатньо, щоб граф *G* був гамільтоновим, для незамкненого гамільтонового маршруту – граф *G* повинен мати гамільтонів ланцюг.

В орієнтованому зваженому графі *G* зустрічається також задача пошуку ***загального орієнтованого маршруту комівояжера*** – найкоротшого орієнтованого маршруту, що містить всі вершини графа, і відповідно, ***орієнтованого шляху*** або ***контуру комівояжера*** (що містить кожну вершину один раз). Замкнений орієнтований (або неорієнтований) маршрут комівояжера при загальній негамільтоновій постановці задачі не обов'язково є гамільтоновим контуром (відповідно гамільтоновим циклом). Наприклад, граф *G* на рис. 11 має один гамільтонів маршрут і кілька гамільтонових циклів, всі їх довжини дорівнюють 24 і кожний містить три дуги. Однак найкоротшим замкненим ормаршрутом комівояжера (непростим) є замкнений маршрут з чотирма дугами (a,b), (b,a), (a,c), (c,a), який проходить через кожну вершину двічі і має довжину 8.

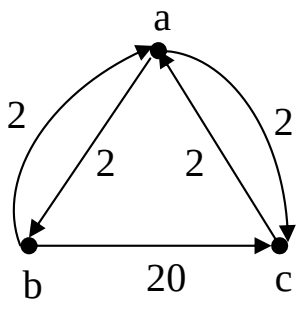


Рис. 11

У яких випадках гамільтонів контур (або цикл) є розв'язком загальної задачі комівояжера?

Теорема. *Якщо функція* $d(x,v)$ *ваг (довжин) ребер (або дуг) між парами вершин* x, v *у графі* $G=(X,Y)$ *з числом вершин* $n=|X|\geq 3$ *задовольняє нерівність трикутника* $d(x,v)\leq d(x,z)+d(z,v),\ \forall z\neq x,\ z\neq v,\ z\in X,$ *та існує розв'язок загальної задачі комівояжера, то існує також розв'язок "гамільтонової" задачі комівояжера*.

Алгебраїчний алгоритм Йоу

В даному алгоритмі послідовно беруться всі прості шляхи простого орграфа *G* за допомогою послідовного перемноження $(n\times n)$ – матриць, що містять символи вершин $x_1, x_2,\ldots, x_n$. Нехай $A=\{a_{ij}\}_1^n$ – матриця суміжності орграфа *G*.

Через $V = \{v_{ij}\}_1^n$ позначимо так звану модифіковану матрицю суміжності графа G, в якій $v_{ijj} = x_j$, якщо існує дуга з x_i до x_j і $v_{ij} = 0$ – якщо такої дуги немає. Тобто, на місці одиниць у матриці суміжності A ставляться символи x_j вершин, в які заходять відповідні рядкам дуги:

$$a_{ij} = 1 \Leftrightarrow v_{ij} = x_j;\ \ a_{ij} = 0 \Leftrightarrow v_{ij} = 0.$$

"Внутрішнім добутком вершин" шляху $(x_1, x_2, x_3, .., x_{k-1}, x_k)$ називається формальний алгебраїчний вираз (слово) $x_2 \cdot x_3 \cdot ... \cdot x_{k-1}$, що не містить початкової і кінцевої вершин шляху. При $k = 2$ добуток вважається рівним 1.

Запропонований алгоритм будує послідовність матриць

$$P_1, P_2, ..., P_{n-1}, V \cdot P_{n-1},$$

де $P_s = \{p_s(i, j)\}_{i,j=1}^n$ – матриця, елемент якої $p_s(i, j)$ дорівнює сумі внутрішніх добутків *всіх простих шляхів з* x_i до x_j довжини s $(1 \le s \le n-1)$, якщо $i \ne j$ і $p_s(i, i) = 0$. Зрозуміло, що $P_1 = A$ (матриця суміжності). Якщо матриця P_s вже обчислена, то для одержання P_{s+1} спочатку будується матриця

$$P_{s+1}^0 = V \cdot P_s = \{p_{s+1}^0(i, j)\},\ \ p_{s+1}^0(i, j) = \sum_k V_{ik} \cdot p_s(k, j). \qquad (1)$$

Її елемент $p_{s+1}^0(i, j)$ дорівнює сумі внутрішніх добутків *всіх таких ланцюгів* (не тільки простих!) з вершини x_i до вершини x_j довжини $s+1$, серед яких *непростими* є в точності ті ланцюги, внутрішні добутки яких містять у чинниках $p_s(k, j)$ з (1) для вершини x_i. Виключивши з суми у (1) доданки, що містять x_i, одержуємо алгебраїчний вираз $p_{s+1}(i, j)$, що дорівнює сумі внутрішніх добутків всіх *простих* (x_i, x_j) – шляхів довжини $s+1$, де $i \ne j$. Вважаючи $p_{s+1}(i, j) = 0$, одержимо шукану матрицю P_{s+1} всіх простих незамкнених шляхів довжини $s+1$.

Нарешті, при $s = n-1$ матриця P_{n-1} дає всі *гамільтонові шляхи* (що мають довжину $n-1$) у графі G між всіма парами вершин. Гамільтонові контури одержуються додаванням дуги (якщо вона існує), що з'єднує кінець гамільтонового шляху з його початком. Інакше кажучи, гамільтонові контури перелічуються членами внутрішніх добутків вершин, що містяться на діагональних елементах матриці $V \cdot P_{n-1}$.

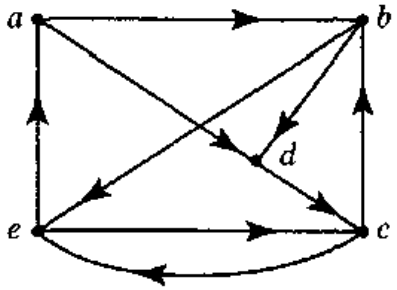


Рис. 12

Проілюструємо роботу алгоритму на прикладі орграфа G рис. 12 з множиною вершин X={*a, b, c, d, e*}.

Матриця суміжності A і модифікована матриця суміжності V мають вигляд:

$$A=\begin{array}{c} \\ a\\ b\\ c\\ d\\ e\end{array}\begin{array}{c}\begin{array}{ccccc} a & b & c & d & e\end{array}\\ \begin{bmatrix}0&1&0&1&0\\0&0&0&1&1\\0&1&0&0&1\\0&0&1&0&0\\1&0&1&0&0\end{bmatrix}\end{array};\quad V=\begin{array}{c} \\ a\\ b\\ c\\ d\\ e\end{array}\begin{array}{c}\begin{array}{ccccc} a & b & c & d & e\end{array}\\ \begin{bmatrix}0&b&0&d&0\\0&0&0&d&e\\0&b&0&0&e\\0&0&c&0&0\\a&0&c&0&0\end{bmatrix}\end{array}.$$

Вважаємо $P_1 \equiv A$. Обчислюючи матрицю $P_2^0 = V \cdot P_1$ і, замінюючи в ній підкреслені діагональні елементи нулями, одержуємо P_2. Викреслені з діагоналі вершини *e, c* є проміжними у 2-контурі *(c, e, c)* і *(e, c, e)* відповідно.

$$P_2^0=\begin{array}{c} \\ a\\ b\\ c\\ d\\ e\end{array}\begin{array}{c}\begin{array}{ccccc} a & b & c & d & e\end{array}\\ \begin{bmatrix}0&0&d&b&b\\e&0&d+e&0&0\\e&0&\underline{e}&b&b\\0&c&0&0&c\\0&a+c&0&a&\underline{c}\end{bmatrix}\end{array};\quad P_2=\begin{array}{c} \\ a\\ b\\ c\\ d\\ e\end{array}\begin{array}{c}\begin{array}{ccccc} a & b & c & d & e\end{array}\\ \begin{bmatrix}0&0&d&b&b\\e&0&d+e&0&0\\e&0&0&b&b\\0&c&0&0&c\\0&a+c&0&a&0\end{bmatrix}\end{array}.$$

Далі знаходимо $P_3^0 = V \cdot P_2$ і після заміни діагональних елементів нулями – матрицю простих 3-шляхів P_3

P_3^0 =

	a	*b*	*c*	*d*	*e*
a	$\underline{be}$	dc	$bd+be$	0	dc
b	0	$\underline{dc+ea+ec}$	0	ea	dc
c	be	$ea+\underline{ec}$	$\underline{bd+be}$	ea	0
d	ce	0	0	$\underline{cb}$	cb
e	$\underline{ce}$	0	ad	$ab+cb$	$\underline{ab+cb}$

P_3 =

	a	*b*	*c*	*d*	*e*
a	0	dc	$bd+be$	0	dc
b	0	0	0	ea	dc
c	be	ea	0	ea	0
d	ce	0	0	0	cb
e	0	0	ad	$ab+cb$	0

Викреслені у матриці P_3^0 діагональні елементи відповідають 3-контурам, які з відповідної причини не можуть бути далі частинами простих 4-шляхів. Викреслені у P_3^0 елементи *ec, ce* на місцях (3, 2), (5, 1) відповідають непростим шляхам *(c, e, c, b), (e, c, e, a)* відповідно, точніше, вершини *c, e* є внутрішніми (проміжними) у цих шляхах. У матриці P_3 значення (*bd*+*be*) елемента (1, 3) означає, що послідовність вершин {*bd*} є внутрішньою у простому 4-шляху (*a, b, d, c*) і послідовність вершин {*b, e*} є внутрішньою у простому 4-шляху (*a, b, e, c*).

В матриці $P_4^0 = V \cdot P_3$ підкреслені діагональні елементи відповідають 4-контурам у графі *G*, а підкреслені недіагональні елементи – непростим 4-шляхам, що містять контур довжини 3 або 2 (довжину контуру вказує відстань від початкової вершини шляху до місця її повторення у внутрішній послідовності). Після заміни у P_3^0 підкреслених елементів нулями одержується матриця P_4 всіх гамільтонових шляхів графа *G*, оскільки тут $4 = n - 1$. Загальна кількість гамільтонових 4-шляхів дорівнює 10: це число всіх ненульових доданків у матриці P_4. Якщо гамільтонові шляхи (*a, b, d, c, e*) і (*a, d, c, b, e*), що відповідають елементу (1, 5) матриці P_4, доповнити дугою *(e, a),* то ми одержимо два *гамільтонових контури графа* $G: Q_1 = (a,b,d,c,e,a),\ Q_2 = (a,d,c,b,e,a)$.

$P_4^0 =$

	a	*b*	*c*	*d*	*e*
a	*dce*	0	0	*bea*	*bdc* + *dcb*
b	*dce*	0	*ead*	*eab* + *ecb*	*dcb*
c	0	0	*ead*	*bea* + *ecb* + *eab*	*bdc*
d	*cbe*	*cea*	0	*cea*	0
e	*cbe*	*adc* + *cea*	*abd* + *abe*	*cea*	*adc*

$P_{n-1} = P_4 =$

	a	*b*	*c*	*d*	*e*
a	0	0	0	0	*bdc* + *dcb*
b	*dce*	0	*ead*	0	0
c	0	0	0	*bea* + *eab*	0
d	*cbe*	*cea*	0	0	0
e	0	*adc*	*abd*	0	0

Якщо будь-який з решти восьми гамільтонових 4-шляхів доповнити відповідною дугою, одержується один з двох контурів Q_1 або Q_2, так що інших гамільтонових контурів у графі *G* немає. Доповнення всіх десяти незамкнених

гамільтонових шляхів до гамільтонового контуру Q_1 або Q_2 перелічуються за допомогою внутрішніх добутків вершин, що стоять на головній діагоналі матриці $V \cdot P_4 = \{\gamma_{ik}\}$. Наприклад, елемент $\gamma_{22} = dcea + eadc$ відповідає другій вершині «*b*» і гамільтоновим контурам *(b, d, c, e, a, b)* і *(b, e, a, d, c, b)*, з яких перший співпадає з Q_1, другий – з Q_2. Таким чином, *кожний діагональний елемент матриці* $V \cdot P_{n-1}$ *перелічує всі гамільтонові контури графа за* допомогою внутрішніх добутків, що виникають при обході контурів з початком у тій вершині, якій відповідає діагональний елемент γ_{ii}.

Якщо не треба знаходити всі незамкнені гамільтонові шляхи, а необхідно лише перелічити *всі гамільтонові контури графа,* то *алгебраїчний алгоритм* можна дещо спростити, скоротивши об'єм обчислень орієнтовно у *n* разів. А саме, достатньо обчислити лише один діагональний елемент матриці $V \cdot P_{n-1}$, наприклад, γ_{11}, а для цього можна знаходити і зберігати у пам'яті не всі елементи матриць $P^0_{n-1}, P_{n-1}; P^0_{n-2}, P_{n-2}; ...; P_1$, *а тільки їх перші стовпці.*

Зауваження. Після одержання всіх гамільтонових незамкнених шляхів або гамільтонових контурів розв'язок задачі комівояжера (відповідно незамкнений або замкнений) зводиться до порівняння ваг шляхів або контурів і знаходження мінімальної ваги. Описаний алгебраїчний метод розв'язку задачі комівояжера завжди приводить до точного розв'язку, однак є досить трудомістким, припускає використання обчислювальних або програмних засобів високого рівня, що ефективно оперують з великими об'ємами символьних перетворень.

Нижче наведено результати розв’язання розглянутої задачі за допомогою програми Mathcad.

ORIGIN := 1 e := e

$$A := \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix} \quad V := \begin{pmatrix} 0 & b & 0 & d & 0 \\ 0 & 0 & 0 & d & e \\ 0 & b & 0 & 0 & e \\ 0 & 0 & c & 0 & 0 \\ a & 0 & c & 0 & 0 \end{pmatrix} \quad P1 := A \qquad P20 := V \cdot P1$$

$$P20 \rightarrow \begin{pmatrix} 0 & 0 & d & b & b \\ e & 0 & d+e & 0 & 0 \\ e & 0 & e & b & b \\ 0 & c & 0 & 0 & c \\ 0 & a+c & 0 & a & c \end{pmatrix} \qquad P2 := P20 - \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & e & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & c \end{pmatrix}$$

$$P30 := V \cdot P2 \qquad P30 \rightarrow \begin{bmatrix} e \cdot b & d \cdot c & b \cdot (d+e) & 0 & d \cdot c \\ 0 & d \cdot c + e \cdot (a+c) & 0 & e \cdot a & d \cdot c \\ e \cdot b & e \cdot (a+c) & b \cdot (d+e) & e \cdot a & 0 \\ c \cdot e & 0 & 0 & b \cdot c & b \cdot c \\ c \cdot e & 0 & a \cdot d & b \cdot a + b \cdot c & b \cdot a + b \cdot c \end{bmatrix}$$

$$P3 := P30 - \begin{bmatrix} b\cdot e & 0 & 0 & 0 & 0 \\ 0 & d\cdot c + e\cdot(a + c) & 0 & 0 & 0 \\ 0 & e\cdot c & b\cdot(d + e) & 0 & 0 \\ 0 & 0 & 0 & c\cdot b & 0 \\ c\cdot e & 0 & 0 & 0 & a\cdot b + c\cdot b \end{bmatrix}$$

$$P3 \rightarrow \begin{bmatrix} 0 & d\cdot c & b\cdot(d + e) & 0 & d\cdot c \\ 0 & 0 & 0 & e\cdot a & d\cdot c \\ e\cdot b & e\cdot(a + c) - c\cdot e & 0 & e\cdot a & 0 \\ c\cdot e & 0 & 0 & 0 & b\cdot c \\ 0 & 0 & a\cdot d & b\cdot a + b\cdot c & 0 \end{bmatrix} \text{ simplify } \rightarrow \begin{bmatrix} 0 & d\cdot c & b\cdot(d + e) & 0 & d\cdot c \\ 0 & 0 & 0 & e\cdot a & d\cdot c \\ e\cdot b & e\cdot a & 0 & e\cdot a & 0 \\ c\cdot e & 0 & 0 & 0 & b\cdot c \\ 0 & 0 & a\cdot d & b\cdot a + b\cdot c & 0 \end{bmatrix}$$

$$P40 := V\cdot P3 \qquad P40 \rightarrow \begin{bmatrix} e\cdot d\cdot c & 0 & 0 & b\cdot e\cdot a & 2\cdot c\cdot b\cdot d \\ e\cdot d\cdot c & 0 & d\cdot e\cdot a & e\cdot(b\cdot a + b\cdot c) & c\cdot b\cdot d \\ 0 & 0 & d\cdot e\cdot a & b\cdot e\cdot a + e\cdot(b\cdot a + b\cdot c) & c\cdot b\cdot d \\ e\cdot b\cdot c & c\cdot[e\cdot(a + c) - c\cdot e] & 0 & e\cdot a\cdot c & 0 \\ e\cdot b\cdot c & d\cdot a\cdot c + c\cdot[e\cdot(a + c) - c\cdot e] & a\cdot b\cdot(d + e) & e\cdot a\cdot c & d\cdot a\cdot c \end{bmatrix}$$

$$P40 \text{ simplify } \rightarrow \begin{bmatrix} e\cdot d\cdot c & 0 & 0 & b\cdot e\cdot a & 2\cdot c\cdot b\cdot d \\ e\cdot d\cdot c & 0 & d\cdot e\cdot a & e\cdot b\cdot(a + c) & c\cdot b\cdot d \\ 0 & 0 & d\cdot e\cdot a & 2\cdot b\cdot e\cdot a + e\cdot b\cdot c & c\cdot b\cdot d \\ e\cdot b\cdot c & e\cdot a\cdot c & 0 & e\cdot a\cdot c & 0 \\ e\cdot b\cdot c & d\cdot a\cdot c + e\cdot a\cdot c & a\cdot b\cdot(d + e) & e\cdot a\cdot c & d\cdot a\cdot c \end{bmatrix}$$

$$P4 := P40 - \begin{bmatrix} d\cdot c\cdot e & 0 & 0 & b\cdot e\cdot a & 0 \\ 0 & 0 & 0 & e\cdot(a\cdot b + c\cdot b) & b\cdot d\cdot c \\ 0 & 0 & e\cdot a\cdot d & c\cdot b\cdot e & b\cdot d\cdot c \\ 0 & 0 & 0 & c\cdot e\cdot a & 0 \\ c\cdot b\cdot e & c\cdot e\cdot a & a\cdot b\cdot e & c\cdot e\cdot a & a\cdot d\cdot c \end{bmatrix} \quad P4 \text{ simplify } \rightarrow \begin{pmatrix} 0 & 0 & 0 & 0 & 2\cdot c\cdot b\cdot d \\ e\cdot d\cdot c & 0 & d\cdot e\cdot a & 0 & 0 \\ 0 & 0 & 0 & 2\cdot b\cdot e\cdot a & 0 \\ e\cdot b\cdot c & e\cdot a\cdot c & 0 & 0 & 0 \\ 0 & d\cdot a\cdot c & a\cdot b\cdot d & 0 & 0 \end{pmatrix}$$

$$P4 := \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & b\cdot d\cdot c + d\cdot c\cdot b \\ d\cdot c\cdot e & 0 & e\cdot a\cdot d & 0 & 0 \\ 0 & 0 & 0 & b\cdot e\cdot a + e\cdot a\cdot b & 0 \\ c\cdot b\cdot e & c\cdot e\cdot a & 0 & 0 & 0 \\ 0 & a\cdot d\cdot c & a\cdot b\cdot d & 0 & 0 \end{pmatrix} \end{array}$$

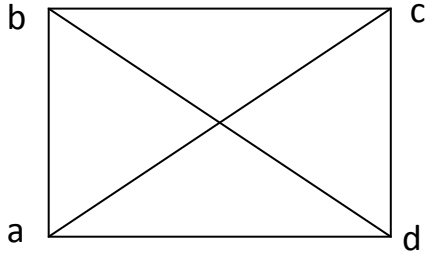
$$V\cdot P4 \text{ simplify } \rightarrow \begin{pmatrix} 2\cdot e\cdot c\cdot b\cdot d & e\cdot a\cdot d\cdot c & b\cdot d\cdot e\cdot a & 0 & 0 \\ e\cdot c\cdot b\cdot d & 2\cdot e\cdot a\cdot d\cdot c & b\cdot d\cdot e\cdot a & 0 & 0 \\ e\cdot c\cdot b\cdot d & e\cdot a\cdot d\cdot c & 2\cdot b\cdot d\cdot e\cdot a & 0 & 0 \\ 0 & 0 & 0 & 2\cdot c\cdot b\cdot e\cdot a & 0 \\ 0 & 0 & 0 & 2\cdot c\cdot b\cdot e\cdot a & 2\cdot a\cdot c\cdot b\cdot d \end{pmatrix}$$

$$
\begin{array}{c}
 & \begin{array}{ccccc} a & b & c & d & e \end{array} \\
\begin{array}{c} a \\ b \\ c \\ d \\ e \end{array} &
\begin{pmatrix}
b\cdot d\cdot c\cdot e + d\cdot c\cdot b\cdot e & d\cdot c\cdot e\cdot a & b\cdot e\cdot a\cdot d & 0 & 0 \\
b\cdot d\cdot c\cdot e & d\cdot c\cdot e\cdot a + e\cdot a\cdot d\cdot c & b\cdot e\cdot a\cdot d & 0 & 0 \\
b\cdot d\cdot c\cdot e & d\cdot c\cdot e\cdot a & b\cdot e\cdot a\cdot d + e\cdot a\cdot b\cdot d & 0 & 0 \\
0 & 0 & 0 & c\cdot b\cdot e\cdot a + c\cdot e\cdot a\cdot b & 0 \\
0 & 0 & 0 & 2\cdot c\cdot b\cdot e\cdot a & a\cdot b\cdot d\cdot c + a\cdot d\cdot c\cdot b
\end{pmatrix} =
\end{array}
$$

Завдання для самостійної роботи

1. Користуючись програмою Mathcad підрахувати кількість усіх гамільтонових циклів у графах, зображених на рис 13.

а)



б)

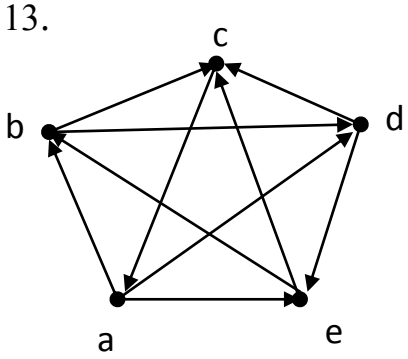


Рис.13.

Лабораторна робота №5

Тема: ЛАБІРИНТИ. ДЕРЕВА. МАТРИЦІ ГРАФІВ

Мета роботи: Вивчити можливості програми Maple для розв’язування задач про лабіринти та дерева. Задання графа матрицею.

Зміст роботи:

1. Вивчити можливості програми Maple для розв’язання задач про лабіринти.

2. Ознайомитись з можливостями програми Maple для побудови дерева та лісу і вивчення їх основних властивостей.

3. Ознайомитись з можливостями програми Maple для задання графа матрицею суміжності чи інцидентності.

Зміст звіту: Короткі теоретичні відомості. Постановка індивідуальних завдань та результати їх виконання.

1. Лабіринти

Розглянемо задачу про пошук виходу з лабіринту, коридори якого не обов'язково знаходяться на одному рівні. Подібна ситуація виникає, наприклад, при блуканні в печерах або катакомбах.

Ототожнивши коридори лабіринту з ребрами, а перехрестя, тупики, входи і виходи з вершинами, ми прийдемо до зв'язного графу, який представляє

схему лабіринту. На рис. 1 зображено цікавий приклад лабіринту в саду Хемптон Корт:

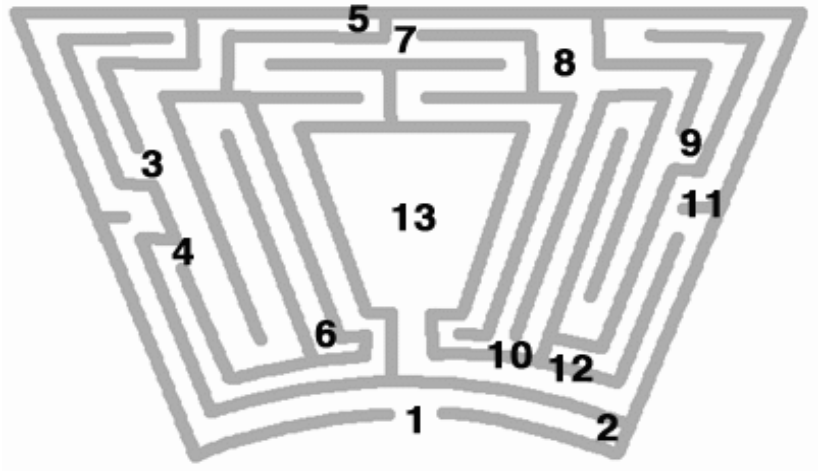


Рис.1.

Побудуємо відповідний йому граф.

```
> L:=void(13):  addedge(Path(2, 1, 3, 4, 7, 8, 9, 11, 12), L):
connect({10}, {7, 8, 9}, L):  addedge({13, 11}, L):
addedge({3, 5}, L):  addedge({4, 6}, L):
draw(Linear([1, 4, 3], [2, 6, 5], [13, 7], [10], [12, 8], [11, 9]), L);
```

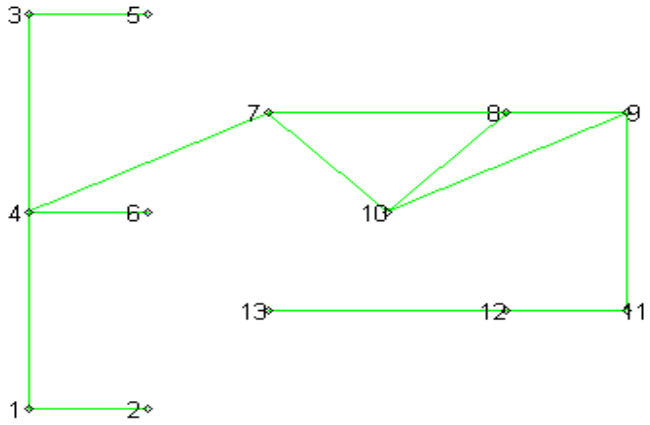


Рис.2.

Таким чином, насправді поставлено завдання про знаходження шляху, що з'єднує дві задані вершини графа. Існує декілька алгоритмів відшукання цього шляху, ми наводимо найкоротший.

Зауваження. Для вирішення цього завдання можна було б скористатися алгоритмом знаходження найкоротшого шляху, однак він містить занадто багато операцій. Застосовуючи його, довелося б проходити по деяких ребрах двічі, щоб прикріпити мітку всім перехрестям, суміжним із даними (якщо таких перехресть більше одного).

Алгоритм: Домовимося, що можна якимось чином відзначати, в якому напрямку пройдено даний коридор і який коридор приводить на дане перехрестя перший раз. Нехай ми знаходимося на перехресті a. Тоді пошук виходу (або перехрестя b) з лабіринту може бути описаний таким алгоритмом:

1. Ніколи не проходити по одному і тому ж коридору в одному і тому ж напрямку двічі. 2. Перебуваючи на деякому перехресті, не вибирати коридор, який привів на це перехрестя перший раз, якщо тільки є можливість іншого вибору.

Ми можемо знайти вихід за допомогою функції знаходження найкоротших шляхів **shortpathtree()** (див. лабораторну роботу №4). Знайдемо в графі, зображеному на рис.1, шлях від входу в лабіринт (вершина 1) в центр лабіринту (вершина 13).

> **G:=shortpathtree(L, 1):**
draw(Linear([1, 4, 3], [2, 6, 5], [13, 7], [10], [12, 8], [11, 9]), G);

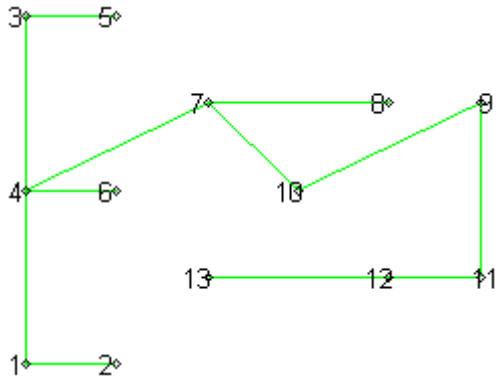


Рис.3.

Тепер добре видно, що в центр лабіринту можна потрапити, слідуючи за наступними вершинами:

1 - 4 - 7 - 10 - 9 - 11 - 12 – 13,

і, відповідно, вийти з центру лабіринту за маршрутом:

13 - 12 - 11 - 9 - 10 - 7 - 4 - 1.

Завдання для самостійної роботи

1. Нарисуйте граф, відповідний лабіринту на рис.4. Знайдіть шлях, по якому можна пройти від пункту A до B лабіринту, використовуючи запропонований вище алгоритм.

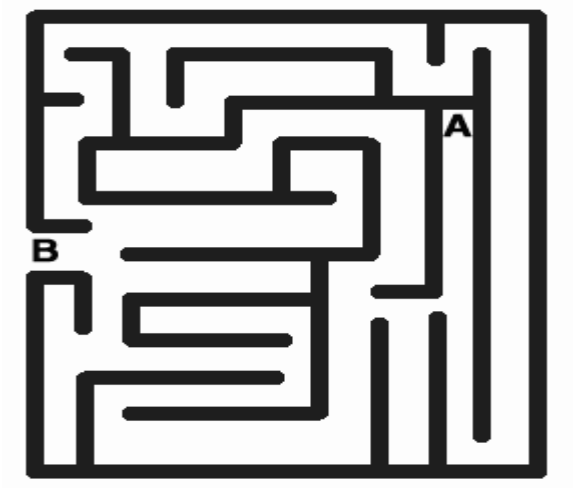


Рис.4.

2. Дерева, ліс

Розв’яжемо наступне завдання: аркуш паперу розрізають на три частини. Деякі з отриманих листів також розрізаються на три частини. Кілька нових листочків знову розрізають на три більш дрібні частини і т.д. Скільки вийде листочків паперу, якщо розрізають k листів?

Розв’язання. Аркуші паперу позначимо вершинами графа, причому кількість всіх вершин не буде збігатися з кількістю одержаних листочків. Відповідь ми отримаємо, якщо будемо рахувати вершини, від яких далі не відходить ні одного ребра (тобто частина аркуша не розрізається далі).

Очевидно (див. рис. 5 нижче), що при розрізанні одного листочка на три частини число листочків збільшується на два (з'являються три нових замість одного):

> G1:=void(4): connect({1}, {2, 3, 4}, G1): draw(G1);

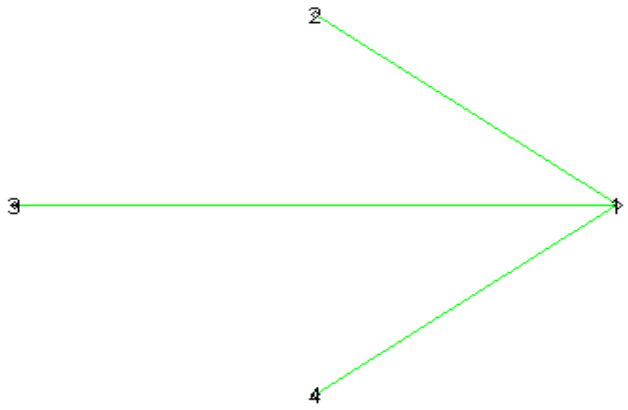


Рис.5.

Якщо ж розрізати k листів, то утворюється 1 +2 k листів:

> G2:=void(16): connect({1}, {2, 3, 4}, G2):
connect({2}, {5, 6, 7}, G2): connect({4}, {8, 9, 10}, G2):
connect({6}, {11, 12, 13}, G2): connect({12}, {14, 15, 16}, G2):
draw(Linear([1], [4, 3, 2], [10, 9, 8, 7, 6, 5], [13, 12, 11], [16, 15, 14]), G2);

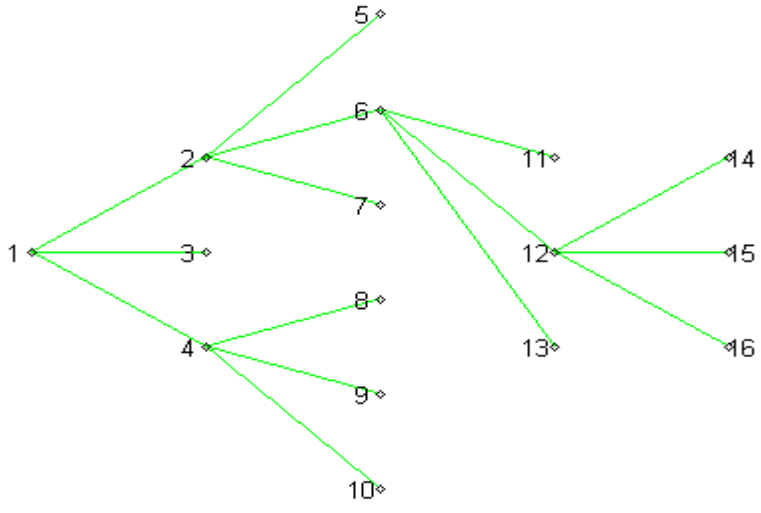


Рис.6.

До речі, схеми на рис. 5 і 6 нагадують гілку дерева з листочками? Математики, звернувши увагу на цю подібність, назвали такі схеми "деревами".

Дамо визначення дерева.

Деревом називається всякий зв'язний граф, що не має циклів. Вершина дерева, степінь якої дорівнює одиниці, називається ***висячою*** вершиною (або ***листком***).

Цікаві факти:

Граф, що складається з ізольованої вершини, теж є деревом.

Дерево володіє мінімальною кількістю ребер: n-1. Тобто, якщо видалити будь-яке ребро з дерева, то ми порушимо зв'язність графа.

Якщо до дерева додати будь-яке нове ребро, ми отримаємо рівно один цикл.

Як вирішити розглянуте вище завдання, якщо ми візьмемо два листочки?

Розв'язання. Відповідь очевидна: якщо ми будемо один лист розрізати на k частин, а інший лист – на t частин, то всього листочків вийде

$$1 + 2k + 1 + 2t = 2(1 + k + t)$$

Проілюструємо це:

```
> with(networks):
> G3:=void(32): connect({1}, {2, 3, 4}, G3): connect({2}, {5, 6, 7}, G3):
 connect({4}, {8, 9, 10}, G3): connect({6}, {11, 12, 13}, G3):
 connect({12}, {14, 15, 16}, G3): connect({17}, {18, 19, 20}, G3):
 connect({18}, {21, 22, 23}, G3): connect({20}, {24, 25, 26}, G3):
 connect({22}, {27, 28, 29}, G3): connect({28}, {30, 31, 32}, G3):
 draw(Linear([1], [4, 3, 2], [10, 9, 8, 7, 6, 5], [13, 12, 11], [16, 15, 14], [17],
[20, 19, 18], [26, 25, 24, 23, 22, 21], [29, 28, 27], [32, 31, 30]), G3);
```

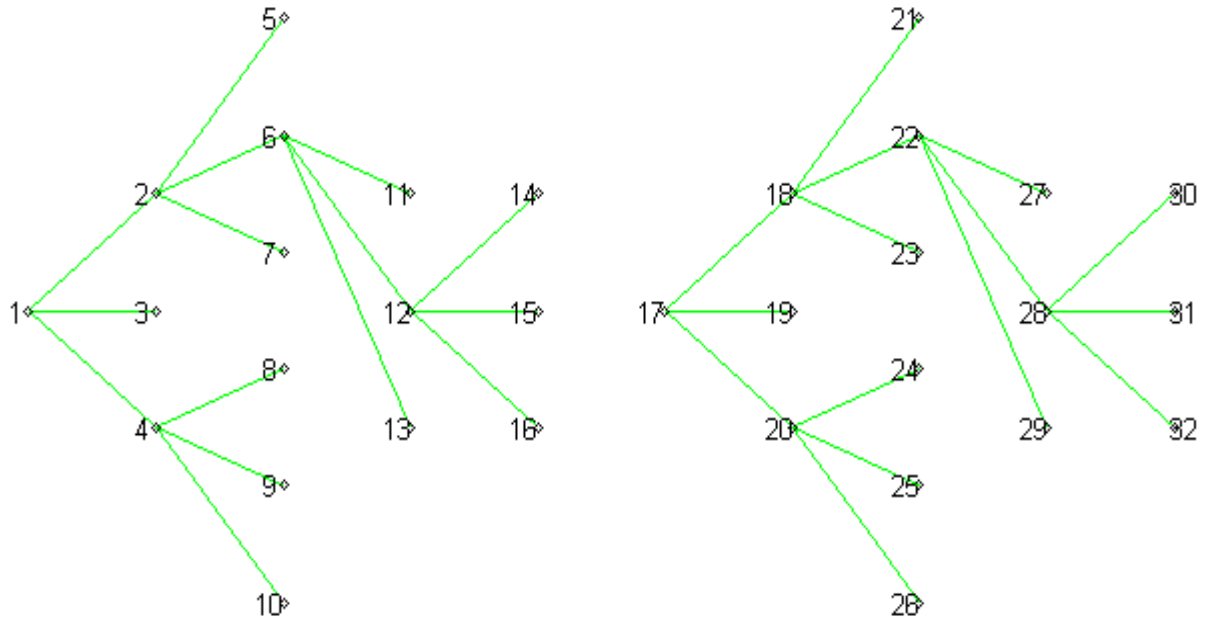


Рис.7.

Отриманий граф трохи складніший, ніж граф на рис.2. Він складається з двох дерев. Таку схему називають *лісом*.

Лісом називається незв'язних граф, що є об'єднанням дерев – компонент зв'язності.

Цікаві факти: В будь-якому лісі з n вершинами і k компонентами nk ребер.

Завдання для самостійної роботи

1. У дереві є 100 вершин степеня 5, 100 вершин степеня 3, а всі інші – висячі. Скільки висячих вершин у цьому дереві?
2. Скільки ребер потрібно видалити з повного графа з 15 вершинами, щоб отримати дерево?
3. Ліс складається з 10 дерев. Всього в лісі 200 вершин. Скільки в ньому ребер?
4. Створіть дерево з 20 вершинами.
5. Створіть ліс, що складається з 3 дерев із 10 вершинами кожне.

3. Матриці графів

При великій кількості вершин і ребер рисунок графа втрачає наглядність. В таких випадках для задання графів і в роботі з ними використовують таблиці, які називаються матрицями.

Опишемо граф с п’ятьма вершинами, зображеного на рис.8. Для цього кожній вершині поставимо у відповідність рядок і стовпець з однаковим номером. При цьому $b_{i,j}=1$, якщо ребро $\{b_i, b_j\}$ належить графу G і $b_{i,j}=0$, якщо не належить. Така матриця називається ***матрицею суміжності*** графа G.

Обчислюється вона функцією **adjacency()**. Знайдемо матрицу суміжності для графа, зображеного на рис.8:

```
> with(networks):
G:=void(5): connect({1}, {2, 3}, G): addedge({2, 5}, G):
draw(G);
```

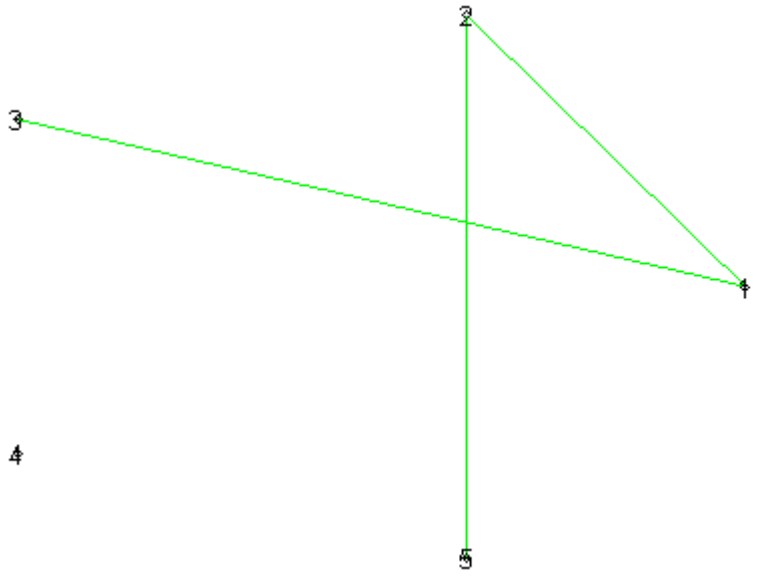


Рис.8

```
> adjacency(G);
```

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Якщо ребра графа орієнтовані, то $b_{i,j}=1$, якщо ребро $[b_i, b_j]$ належить графу (тобто воно виходить із вершини b_i), і $b_{i,j}=0$, якщо ребро $[b_i, b_j]$ або не належить графу, або входить у вершину b_i.

> **addedge([1, 5], G): K:=adjacency(G);**

$$K := \begin{bmatrix} 0 & 1 & 1 & 0 & 2 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Всякий граф можна задати матрицею суміжності.

Будь-яку квадратну матрицу, що складена тільки із нулів і одиниць можна розглядати як матрицу суміжності деякого графа і по ній побудувати відповідний граф.

Крім матриці суміжності, існує матриця інцидентності графа.

Якщо кожній вершині поставити у відповідність рядок, а кожному ребру – стовпець з однаковим номером. При цьому $b_{i,j}=1$, якщо вершина b_j інцидентна (належить) ребру e_j, $b_{i,j}=0$ в протилежному випадку. Така матриця називається ***матрицею інцидентності*** графа G.

Коли ребро орієнтоване: якщо ребро e_j виходить із вершини b_j, то $b_{i,j}=-1$, а якщо входить – $b_{i,j}=1$ і якщо не належить графу, то $b_{i,j}=0$.

Обчислюється матриця інцидентності функцією **incidence().** Знайдемо матрицю інцидентності для графа, зображеного на рис. 8.

> **T:=incidence(G);**

$$T := \begin{bmatrix} 1 & 1 & 0 & -1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

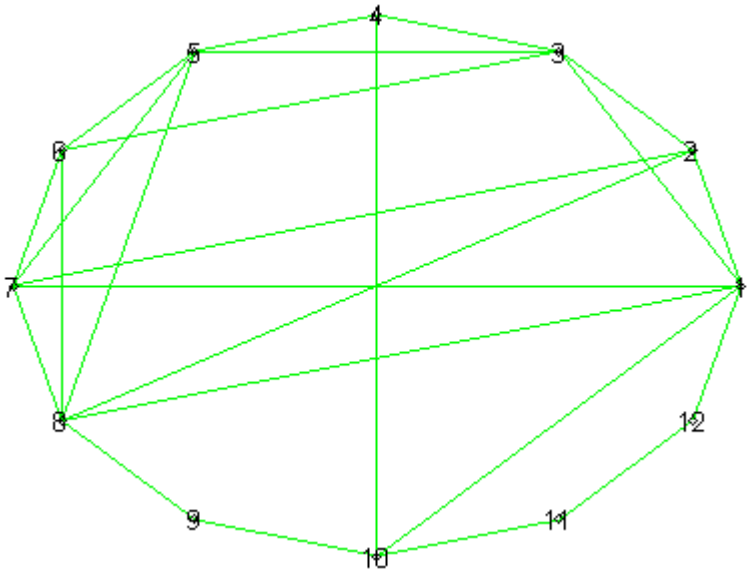
Завдання для самостійної роботи

1. Що означає, якщо $b_{1,1}=1$. Якою особливістю володіє граф, якщо в матриці суміжності всі елементи на "головній діагоналі" рівні нулю?
2. Побудувати граф, що відповідає матриці суміжності:

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

3. Знайти матрицю суміжності й інцидентності графа

> **X:=cycle(12): connect({1, 2, 5, 6}, {3, 7, 8}, X): connect({10}, {1, 11, 4}, X): draw(X);**



СПИСОК ЛІТЕРАТУРИ

1. Акимов О. Е. Дискретная математика. Логика, группы, графы / О. Е. Акимов. – М.: Лаборатория Базовых Знаний, 2003. – 376 с.
2. Бондаренко М. Ф. Комп'ютерна дискретна математика / М. Ф. Бондаренко, Н. В. Білоус, А. Г. Руткас. – Харків: «Компанія СМІТ», 2004. – 480 с.
3. Донской В. И. Дискретная математика / В. И. Донской. – Симферополь: Сонат, 2000. – 360с.
4. Капітонова Ю. В. Основи дискретної математики / Ю. В. Капітонова, С. Л. Кривий, О.А. Летичевський. – К.: Наукова думка, 2002. – 578 с.
5. Коваленко Л. Б. Дискретна математика: Навчальний посібник для студентів економічних, менеджерських та електротехнічних спеціальностей вищих навчальних закладів / Л. Б. Коваленко, С. О. Станішевський. – Харків: ХНАМГ, 2006. – 192 с.
6. Кузнецов О. П. Дискретная математика для инженера / О.П. Кузнецов, Г. М. Адельсон-Вельский. – М.: Энергия, 1980. – 344 с.
7. Прохоров Г. В. Пакет символьных вычислений Maple V / Г. В. Прохоров, М. А. Леденев, В. В. Колбеев. – М.: Петит, 2001. – 200 с.
8. Тевяшев А. В. Основы дискретной математики в примерах и задачах / А. В. Тевяшев, И. Г. Гусарова. – Харьков: ХНУРЭ, 2003. – 272 с.
9. Яблонский С. В. Введение в дискретную математику / С. В. Яблонский. – М.: Наука, 1979.– 384 с.
10. Уилсон Р. Введение в теорию графов / Р. Уилсон. – М.: Мир, 1977. – 205 с.

ЗМІСТ