Лекція 12.
Пошук з поверненням (бектрекінг)

§1 Постановка задачі

Багато задач не припускають аналітичного розв'язку, а тому їх залишається розв'язувати методом спроб та помилок, тобто перебираючи фактично варіанти та відкидаючи їх у разі невдачі. У разі, якщо побудова розв'язку є складною процедурою, то фактично під час роботи будується дерево можливих кроків алгоритму, а потім - у разі невдачі - відрізаються відповідні гілки дерева, доки не буде побудовано той шлях, що веде до успіху. Проходження вздовж гілок дерева та відходження у разі невдач і є алгоритм з поверненнями.

Ідея методу:

Розв'язок будують поступово, починаючи з порожньої послідовності λ (довжини 0).

Нехай є неповний розв'язок $(x_1, x_2, \ldots x_i)$, $i<n$. Намагаємося знайти таке допустиме значення x_{i+1}, що не виключає можливості продовження $(x_1, x_2, \ldots x_i, x_{i+1})$ до повного розв'язку.

Якщо таке допустиме, але ще не використане значення x_{i+1} існує, то додаємо його до неповного розв'язку і продовжуємо шукаючи x_{i+2}.

Якщо такого значення x_{i+1} не існує, то повертаємося до послідовності $(x_1, x_2, \ldots x_{i-1})$, і шукаємо нове, ще не використане x_i'.

Роботу алгоритму можна представити як обхід дерева пошуком в глибину.

Кожна вершина відповідає послідовності $(x_1, x_2, \ldots x_i)$. Корінь дерева відповідає порожній послідовності.

Задається предикат P, визначений на всіх вершинах дерева. У разі $P(v)=F$ процес обходу відкидає розгляд вершин піддерева з коренем у вершині *v*, зменшуючи обсяг перебору. Предикат $P(v)$ набуває значення *F* тоді, коли стає зрозумілим, що послідовність $(x_1, x_2, \ldots x_i)$, яка відповідає вершині *v*, ніяким способом не можна побудувати до повного розв'язку.

§2 Знаходження гамільтонових циклів у графі

Ланцюгом (неорієнтованого) графа називають маршрут, всі ребра якого різні.

Ланцюг називають **простим**, якщо він не містить однакових вершин, можливо, за виключенням першої і останньої, якщо ланцюг замкнений.

Замкнений ланцюг називають **циклом** (або контуром).

Замкнений простий ланцюг називають **простим циклом.**

Гамільтоновим циклом графа називають його простий цикл, що проходить через кожну вершину графа.

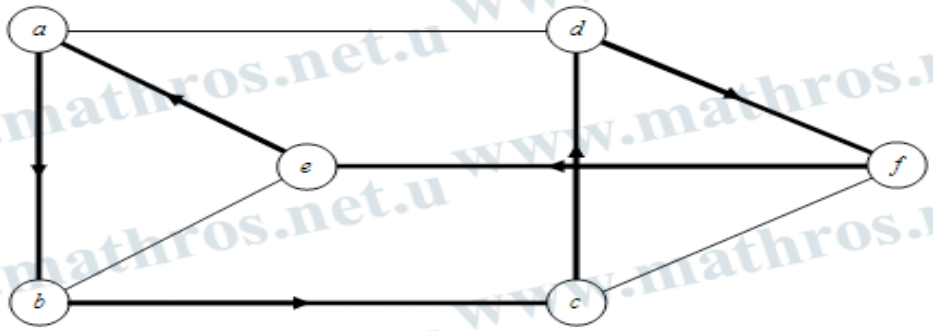
Граф називають **гамільтоновим**, якщо він має гамільтоновий цикл.

Зовні визначення **Гамільтонового циклу** схоже на **визначення Ейлерового циклу**. Однак є кардинальна відмінність в складності розв'язку відповідних задач на розпізнавання і побудову. Тобто, якщо при розгляді Ейлерового циклу ми бачили, що для нього існує досить простий критерій перевірки його існування і ефективний алгоритм його побудови. То для Гамільтонових же циклів невідомо ніяких необхідних і достатніх умов їх існування, а всі відомі алгоритми вимагають, для деяких графів, перебору великого числа варіантів.

Гамильтонів цикл являє собою, з комбінаторної точки зору, просто перестановку вершин графа. При цьому в якості початкової вершини циклу можна вибрати будь-яку вершину, так що можна розглядати перестановки з фіксованим першим елементом. Самий нехитрий план **пошуку Гамільтонового циклу** полягає в послідовному розгляді всіх цих перестановок і перевірці, для кожної з них, чи становить вона цикл в даному графі. Такий спосіб дій вже при не дуже великому числі вершин стає практично неможливим попри швидке зростання кількості перестановок

є (n-1)! перестановок з n елементів з фіксованим першим елементом.

Більш раціональний підхід полягає в розгляді всіляких простих шляхів, що починаються в довільно обраній стартовій вершині *a* до тих пір, поки не буде виявлений **Гамільтонів цикл**, або всі можливі шляхи не будуть досліджені. По суті справи, мова теж йде про перебір перестановок, але в значно скороченому вигляді – якщо, наприклад, вершина *b* не суміжна з вершиною *a,* то всі (n-2)! Перестановок, у яких на першому місці стоїть *a*, а на другому – *b*, не досліджуються. Розглянемо цей алгоритм більш детально.



Графічне представлення алгоритму пошуку Гамільтонового циклу

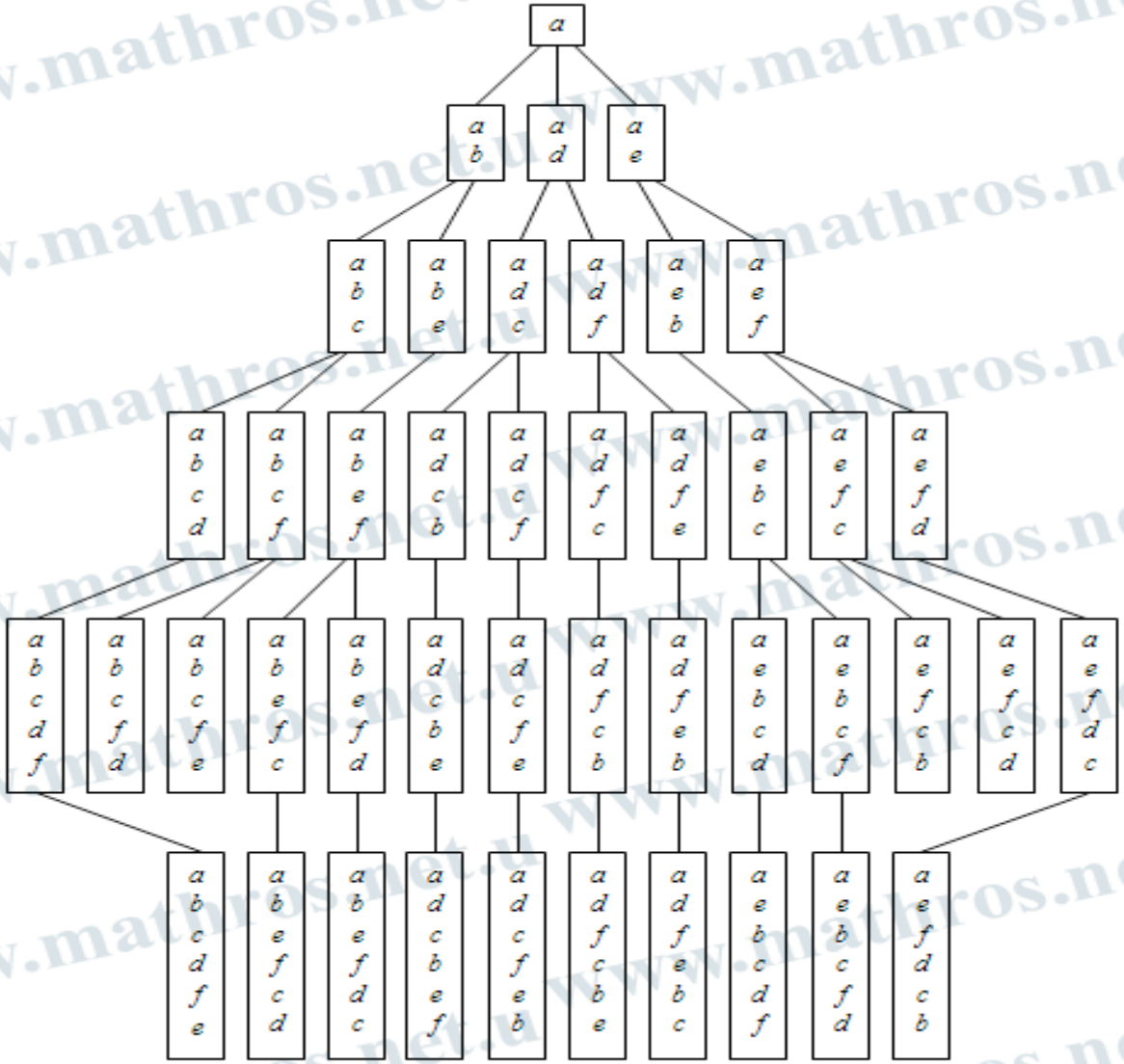
Для цього, побудуємо деякий неорієнтований граф G і в якості стартової вибирається будь-яка з його вершин, наприклад вершина *a* . Ця вершина утворює перший елемент деякої множини S $S = \{a\}$ Множина S на кожному кроці буде зберігати вже знайдені вершини **Гамільтонового ланцюга .** На наступному кроці до S додається перша зустрічна нова вершина, суміжна з вершиною *a* (відмітимо, що під новою розуміється вершина, яка ще не належить множині S). В нашому випадку це вершина *b*. Потім до *S* додається нова суміжна вершина до *b*. Нехай це вершина *c*. Потім до *S* додається перша суміжна вершина до *c* і так далі. Зазначимо, що ітераційний процес продовжується до тих пір, поки:

1. Для деякої вершини виявиться, що суміжних нових вершин не існує. В даному випадку необхідно виконати так звану процедуру повернення, яка полягає у видаленні останньої включеної в S вершини і додаванні до неї нової, суміжної з передостанньої вершиною. Якщо і в цьому випадку не існує ніякої нової вершини, то робиться наступний крок повернення і так далі.

2.Шлях, який визначається послідовністю вершин, що складають множину S, має довжину n-1, де n – кількість вершин графа. В даному випадку необхідно припинити пошук і перевірити, чи в заданому графі існує неорієнтоване ребро що поєвднує перший і останній елемент множини S. Якщо таке ребро існує, то **Гамільтонів цикл** знайдено. В іншому випадку, або у випадку, коли для заданого графа необхідно знайти всі **Гамільтонові цикли** – виконуємо процедуру повернення.

Відмітимо, що пошук закінчується, коли множина S складається з однієї вершини *a* і не існує ніякої нової вершини, якою можна було б доповнити S (наступний крок повернення робить множину S порожньою). Це означає, що всі **Гамільтонові цикли**, якщо вони існують, знайдені.

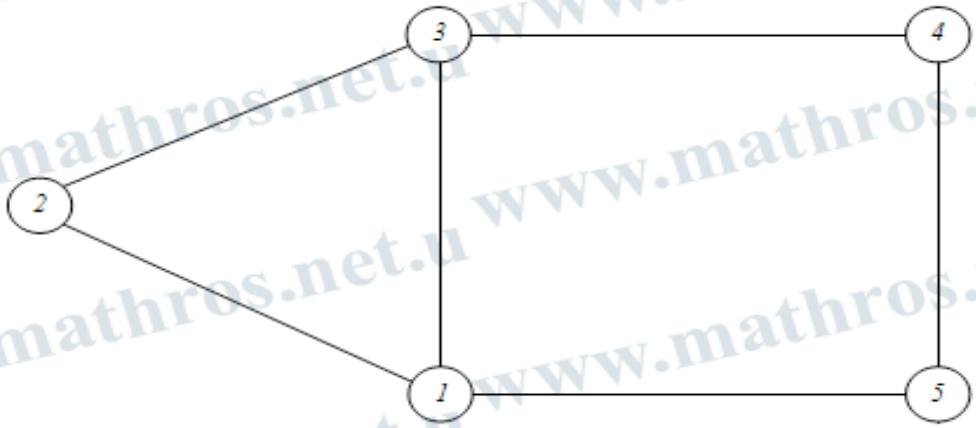
Зауваження: *виходячи з того, що розглянутий вище алгоритм дуже схожий на **алгоритмом обходу графа в глибину**, то результат його роботи, також, можна представити у вигляді дерева, яке у даному випадку, являється деревом шляхів. Вершинами цього дерева є всілякі прості шляхи, що починаються в стартовій вершині, а ребра дерева з'єднують два шляхи, один з яких виходить з іншого додаванням однієї вершини в кінці. На малюнку що міститься нижче показано дерево шляхів для вищевказаного графа, де в якості стартової обрана вершина a.*


a
ab
ad
ae
abc
abe
adc
adf
aeb
aef
abcd
abcf
abef
adcb
adcf
adfc
adfe
aebc
aefc
aefd
abcdf
abcfd
abcfe
abefc
abefd
adcbe
adcfe
adfcb
adfeb
aebcd
aebcf
aefcb
aefcd
aefdc
abcdfe
abefcd
abefdc
adcbef
adcfeb
adfcbe
adfebc
aebcdf
aebcfd
aefdcb

Дерево шляхів неорієнтованого графа

Гамільтонів цикл в неорієнтованому графі – приклад знаходження:

Знайти всі **Гамільтонові ланцюги** для неорієнтованого графа, представленого на наступному малюнку.



Неорієнтований граф задачі

Для цього, слідуючи розглянутому вище алгоритму, на першому кроці, необхідно вибрати стартову вершину. Нехай це буде вершина номер *«1»*. Збережемо цю вершину в якості першого елемента для множини $S = \{1\}$

Далі, скориставшись одним з інцидентних їй ребер, наприклад (1,2) , переходимо у вершину номер *«2»* (множина S на даному етапі прийме наступного вигляду. $S = \{1,2\}$

На наступному кроці, до S додаємо нову суміжну вершину до вершини *«2»*. В результаті отримаємо $S = \{1,2,3\}$

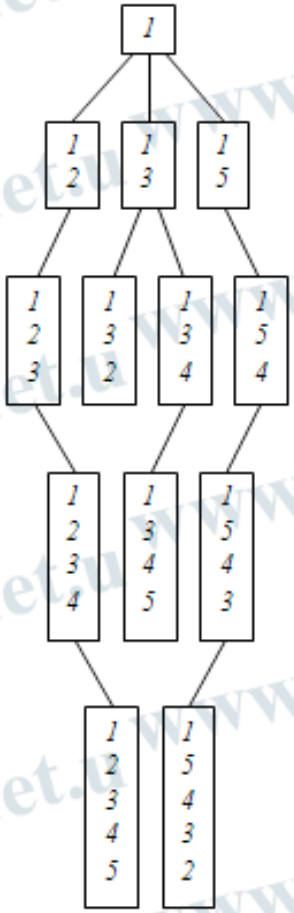
Продовжуючи аналогічні дії далі, множина S прийме наступний вигляд .$S = \{1,2,3,4,5\}$

Після цього, виконавши перевірку, на наявність неорієнтованого ребра між вершинами *«5»* та *«1»* приходимо до висновку, що один з **Гамільтонових циклів** для задаого графа знайдено.

Далі, виконуємо процедуру повернення і робимо це аж до стартової вершини, тобто $S = \{1\}$
після чого, знову-таки, виконуємо наповнення множини S новими, зв'язними між собою, вершинами заданого графа. В результаті будемо мати Тобто $S = \{1,3,2\}$шляху, що визначається множиною S є меншою від числа чотири, що свідчить про те, що отримана таким чином послідовність вершин не являється **Гамільтоновим циклом** і необхідно здійснити повернення, в даному випадку, також до вершини номер *«1»*.
Продовжуючи ітераційний процес далі, отримаємо ще два шляхи, а саме

$S = \{1,3,4,5\}$

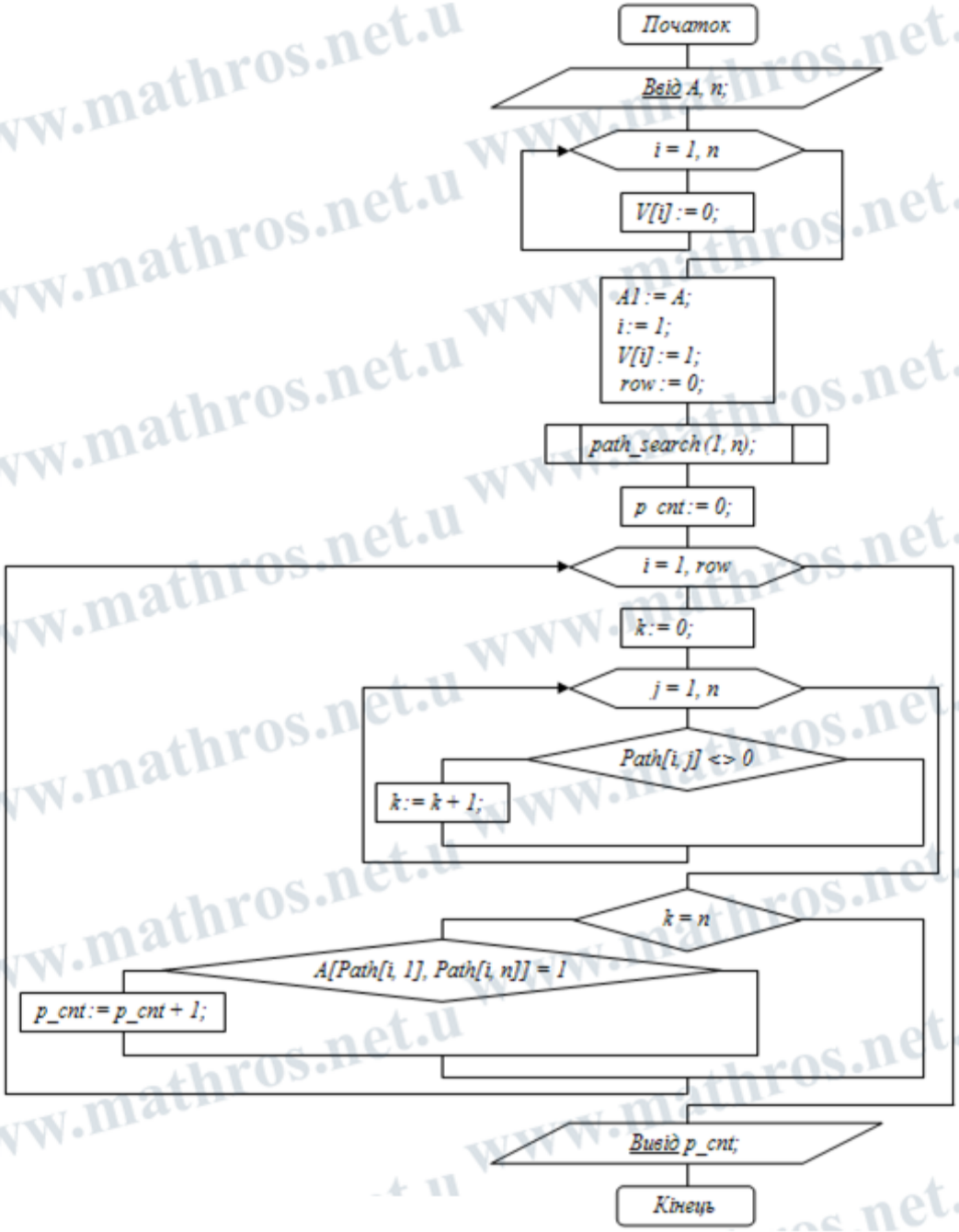
та $S = \{1,5,4,3,2\}$
на цьому алгоритм **пошуку Гамільтонового циклу** можна завершити (подальше повернення приводить до порожньої множини). Як не важко переконатися, з допомогою шляху, отриманому на останній ітерації, також можна **побудувати Гамільтонів цикл**.

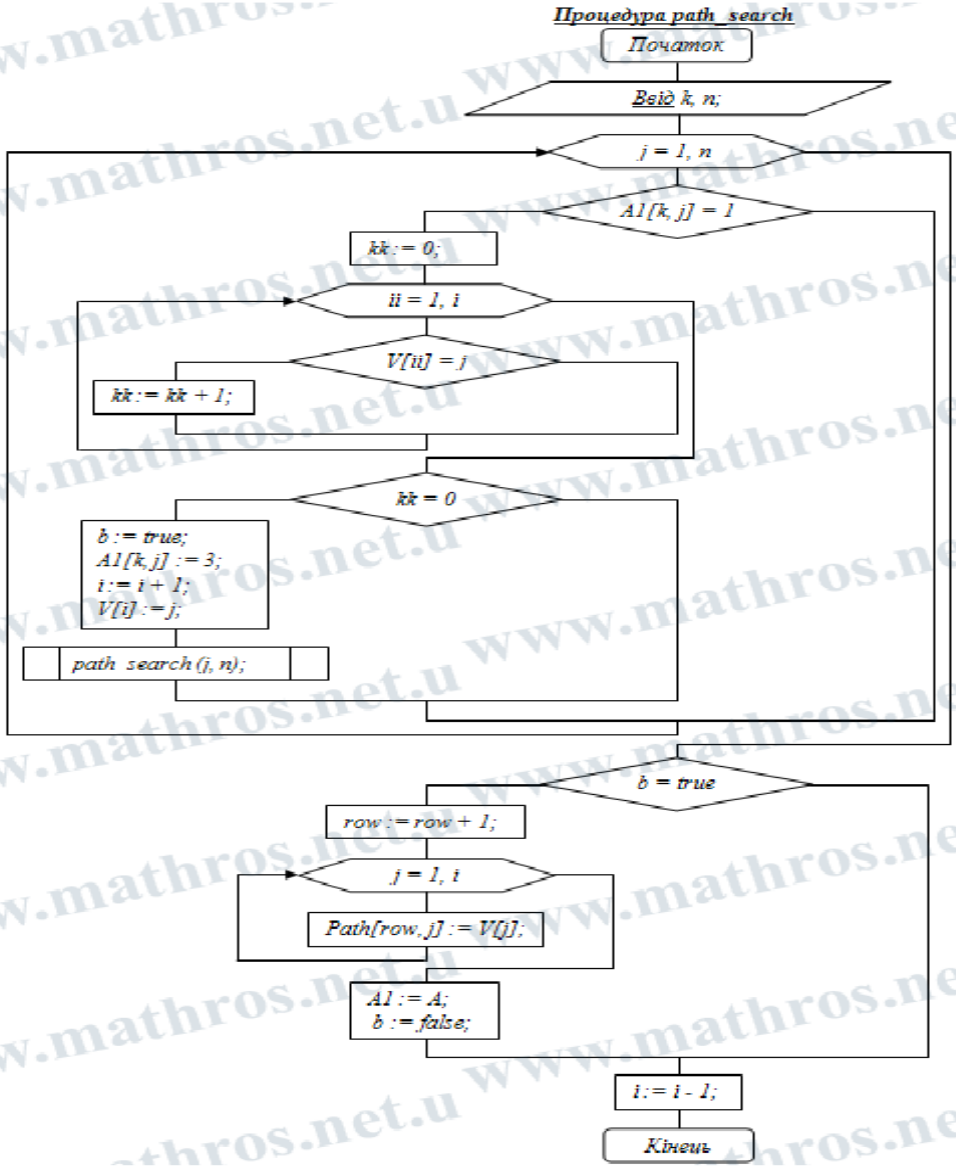


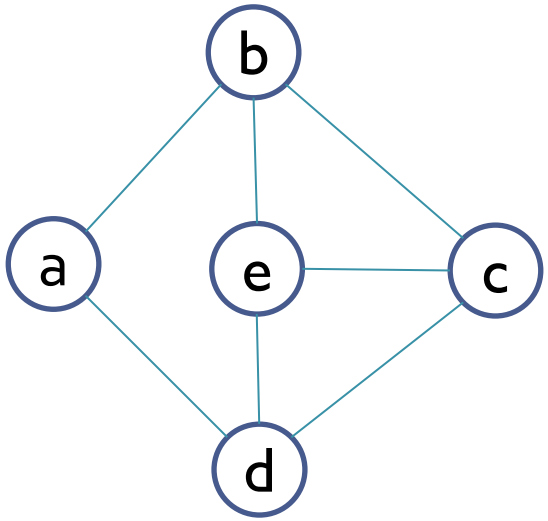
Пошук Гамільтонового циклу в неорієнтованому графі

Звідси, приходимо до висновку, що заданий неорієнтований граф містить два **Гамільтонових цикли**, які визначаються наступними порядками обходу вершин: $1 \to 2 \to 3 \to 4 \to 5 \to 1$ та $1 \to 5 \to 4 \to 3 \to 2 \to 1$

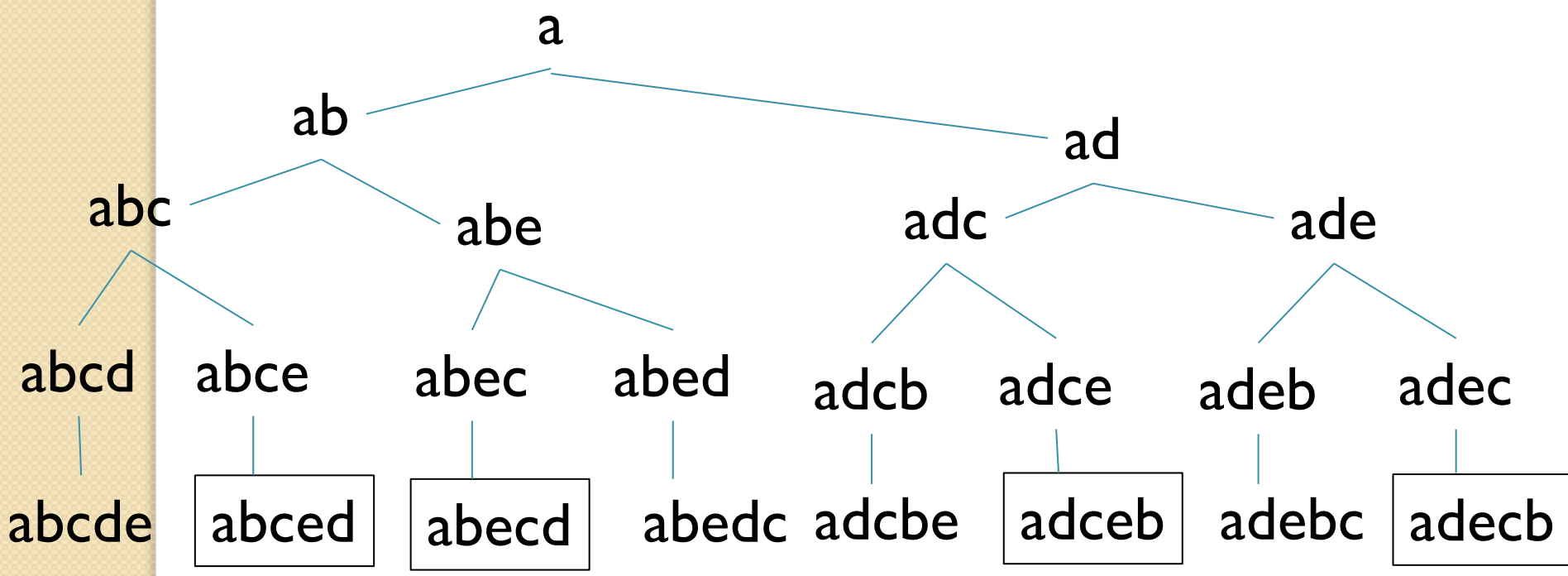
Блок-схема алгоритму пошуку Гамільтонового циклу в неорієнтованому графі




Процедура path_search
Початок
Ввід k, n;
j = 1, n
A1[k, j] = 1
kk := 0;
ii = 1, i
V[ii] = j
kk := kk + 1;
kk = 0
b := true;
A1[k, j] := 3;
i := i + 1;
V[i] := j;
path_search (j, n);
b = true
row := row + 1;
j = 1, i
Path[row, j] := V[j];
A1 := A;
b := false;
i := i - 1;
Кінець

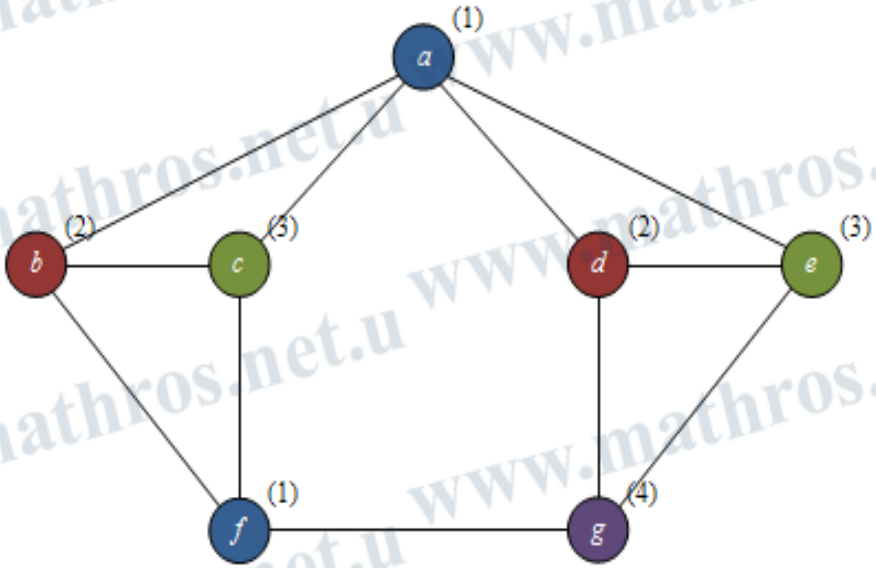


Починаємо з довільної вершини. Будуємо шлях без повторення вершин, доки це можливо. Якщо вдалося пройти всі вершини, то перевіряємо, чи існує ребро, яке з'єднує останню і початкову вершини.



Переборний алгоритм для розфарбування вершин графа

Розфарбуванням вершин графа називається процес призначення певного кольору кожній з його вершин. Зазвичай кольори – це числа 1,2,3,…,k.. Тоді, розфарбування є функцією, визначеною на множині вершин графа, яка приймає значення з множини $\{1,2,3,\ldots,k\}$ Розмальовку можна також розглядати як розбиття множини вершин графа на підмножини, кожна з яких являється множиною вершин певного кольору. Відмітимо, що такі підмножини називаються кольоровими класами. Розфарбування називається правильним, якщо кожен кольоровий клас є незалежною множиною. Інакше кажучи, в правильному розфарбуванні будь-які дві суміжні вершини повинні мати різні кольори. Задача про розфарбовування полягає в знаходженні правильної розмальовки графа в найменше число кольорів. Це число називається **хроматичним числом графа** і позначається $\chi(G)$



Розфарбування вершин графа найменшим набором кольорів

У правильному розфарбуванні повного графа G , кожна з його вершин повинна бути зафарбована у свій колір, тому **хроматичне числа графа** такого типу дорівнює кількості його вершин, тобто $\chi(G) = n$

Якщо в графі існує повний підграф з k вершинами, то для розмальовки цього підграфа необхідно k кольорів. Звідси випливає, що для будь-якого графа виконується нерівність $\chi(G) \geq \omega(G)$, де $\omega(G)$ - клікове число графа G .

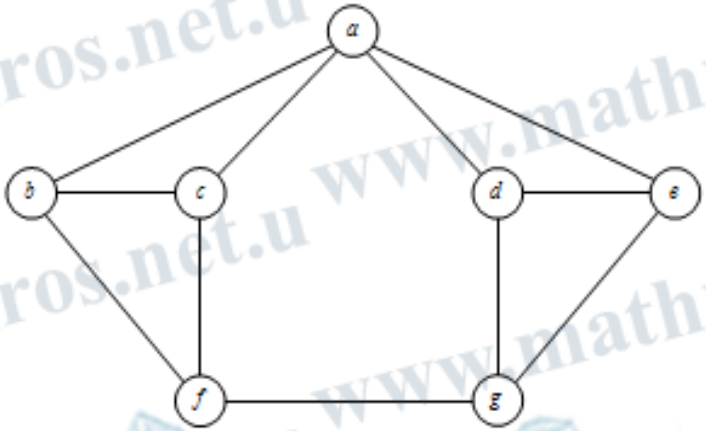
Відмітимо, що **хроматичне число графа** може бути і строго більше за його клікове число. Наприклад, на рисунку, що міститься вище, зображений граф, вершини якого розфарбовані в чотири кольори (номер кольору кожної вершини показано в дужках). Неважко перевірити, що трьох кольорів для правильного розфарбування цього графа недостатньо. Отже, його **хроматичної число** дорівнює чотири. Очевидно також, що клікове число цього графа дорівнює три.

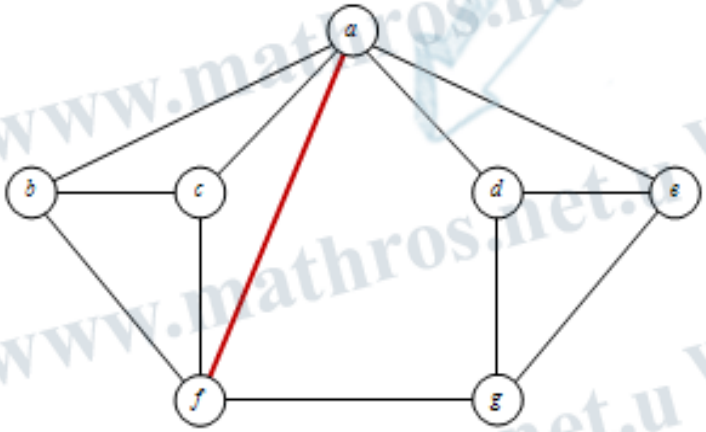
Також хочеться зазначити, що, $\chi(G) = 1$ тоді і тільки тоді, коли G – порожній граф. Неважко охарактеризувати і графи з **хроматичним числом** рівним два (точніше, не більше два). За визначенням, це такі графи, множину вершин яких можна розбити на дві незалежних підмножини. Виходячи з того, що дане визначення збігається з визначенням **дводольного графа**, приходимо до висновку, що дводольні графи також називають біхроматичними. Тоді, скориставшись теоремою, яка вказує на критерій дводольності графа, запишемо критерій його біхроматичності. Отже, граф називається біхроматичним тоді і тільки тоді, коли всі його прості цикли мають парну довжину.

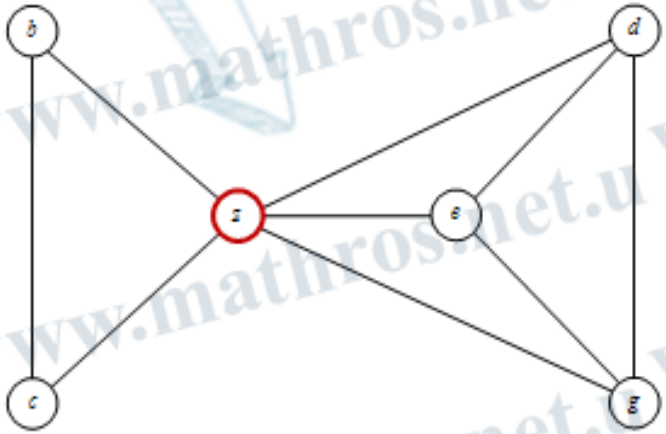
Для графів з **хроматичним числом** три та більше, такого простого опису невідомо. Невідомо й простих алгоритмів, які б дозволили перевірити, чи можна даний граф розфарбувати в три кольори. Проте, алгоритми для подібних задач розроблялися і продовжують розроблятися і в деяких випадках вони можуть бути корисні. Всі ці алгоритми в тій чи іншій формі здійснюють перебір варіантів (число яких може бути дуже великим).

Далі, розглянемо один із способів такого перебору для задачі про **розфарбовування графа**, який базується на ідеї, що задача такого типу, для заданого графа, зводиться до тієї ж задачі для двох інших графів. Тобто в результаті виконання даного алгоритму, виникає дерево варіантів, обхід якого дозволяє знайти рішення задачі. Розглянемо даний алгоритм більш детально.

Отже, в заданому графі G, виберемо дві не суміжні між собою вершини, наприклад *a* та f. На наступному кроці, побудуємо два нових графа: G_1 що виходить додаванням ребра (a,f) до графа G і G_2, що виходить з G в результаті злиття вершин *a* та *f* (операція злиття полягає у видаленні вершин *a* та *f* і додаванні нової вершини, наприклад *z* та ребер, що з'єднують її з кожною вершиною, з якою була суміжна хоча б одна з видалених вершин *a* I *f*







Використання алгоритму методу перебору для розв'язку задачі про розфарбування графа

Відмітимо, що якщо в правильному **розфарбуванні графа** G вершини *a* і *f* мають різні кольори, то воно буде правильним і для графа G_1

Якщо ж кольори вершин a і f у розфарбуванні графа G однакові, то граф G_2

можна розфарбувати в ту саму кількість кольорів: нова вершина фарбується в той колір, в який пофарбовані вершини a I f , а всі інші вершини зберігають ті кольори, які вони мали в графі G.

Справедливе і обернене твердження, тобто розфарбування кожного з графів G_1 G_2

очевидно, дає розмальовку графа G в ту саму кількість кольорів.

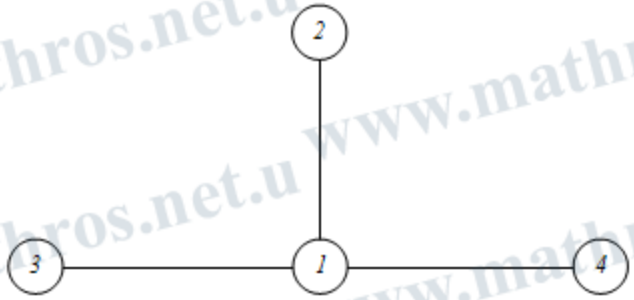
Тому, $\chi(G) = \min(\chi(G_1), \chi(G_2))$

що дає можливість рекурсивного знаходження розмальовки графа в мінімальне число кольорів. Зауважимо, що граф G_1

має стільки ж вершин, скільки вихідний граф, але у нього більше ребер. Тому рекурсія в кінцевому рахунку призводить до повних графів, для яких задача про розфарбовування вирішується тривіально.

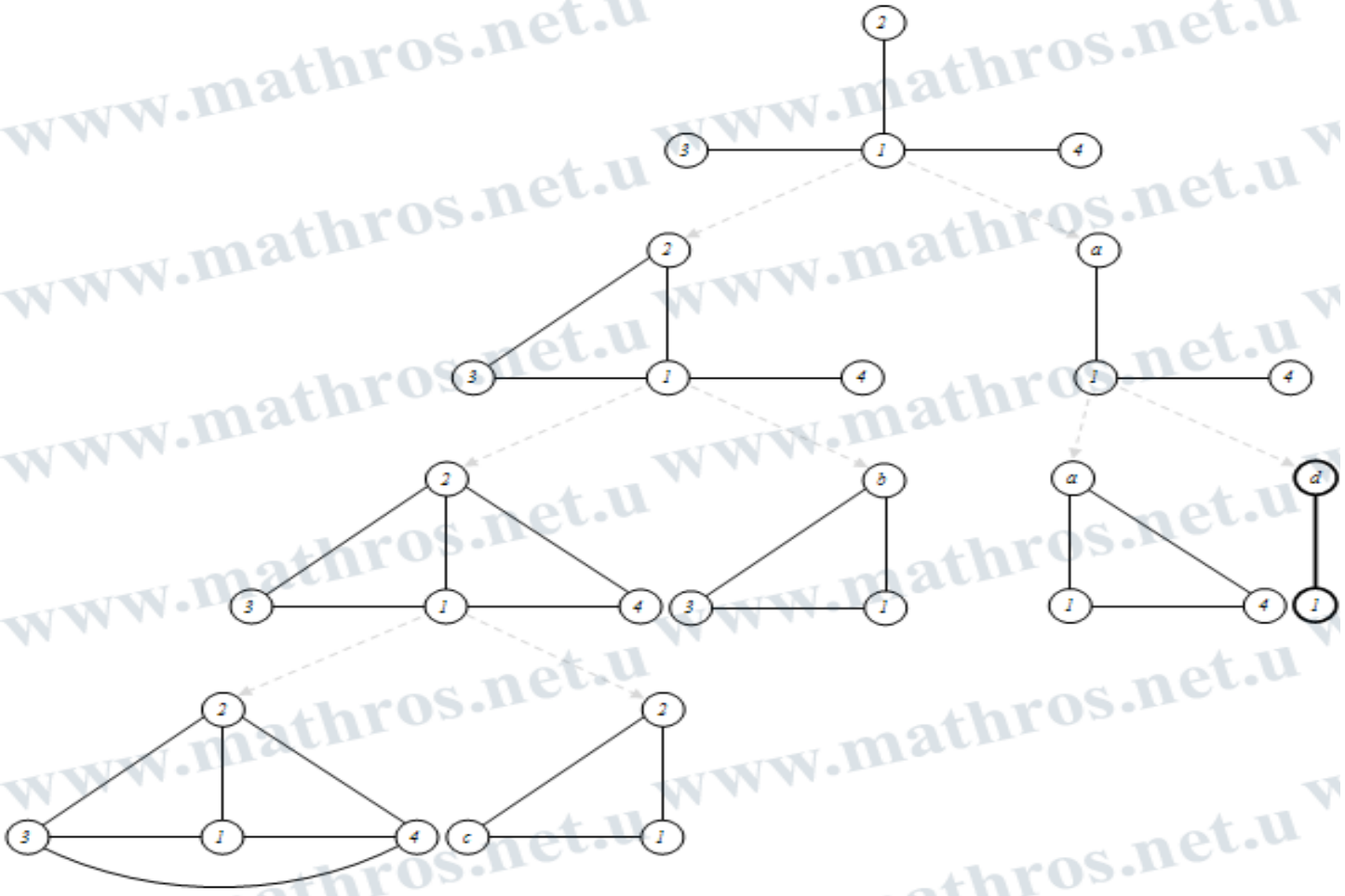
Розфарбування вершин графа використовуючи переборний алгоритм – приклад:

Знайти розв’язок задачі про **розфарбування вершин неорієнтованого графа** зображеного на наступному малюнку.



Неорієнтований граф задачі

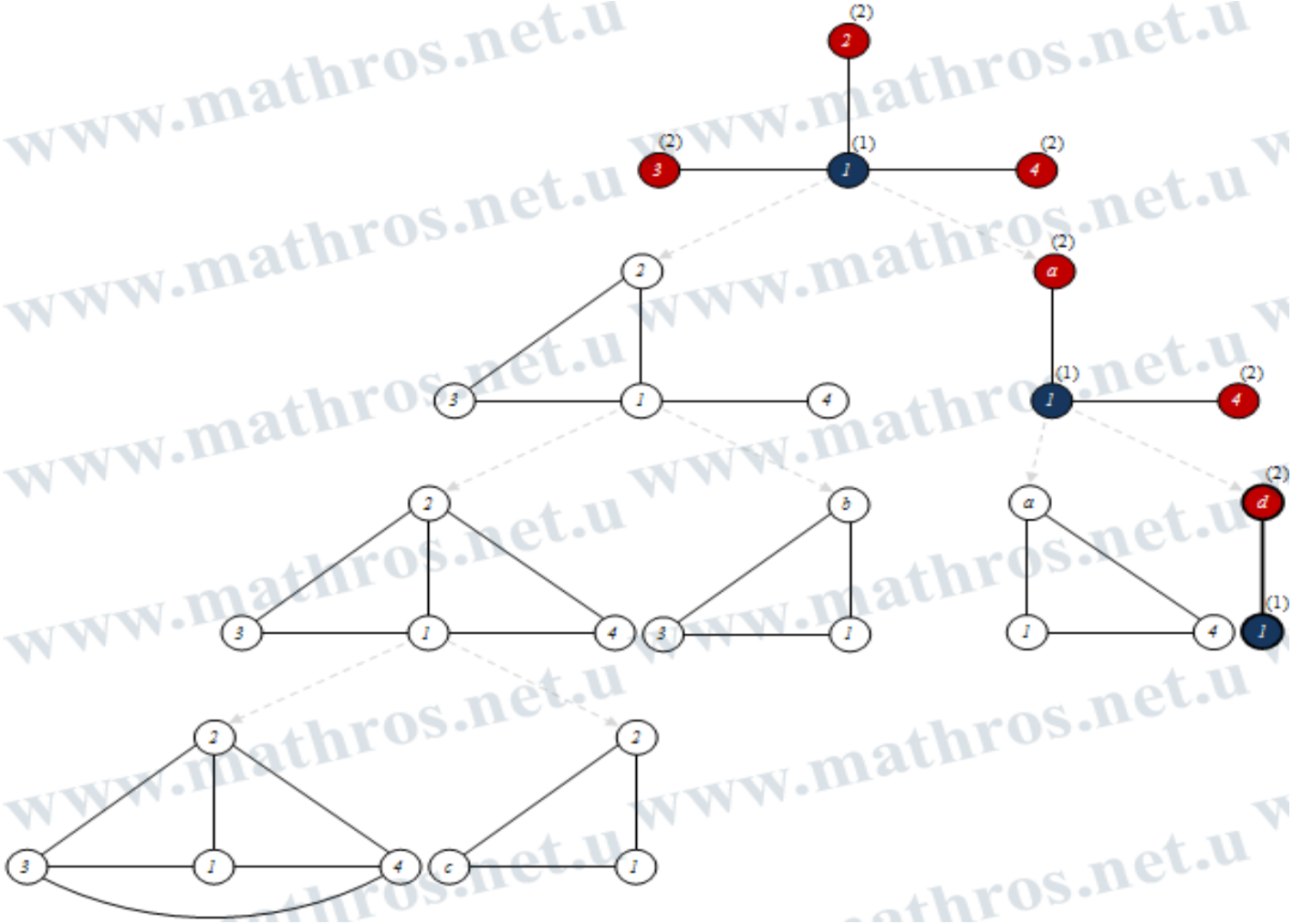
Для цього, скориставшись розглянутим вище алгоритмом, побудуємо дерево варіантів, обхід якого дозволить знайти шуканий розв’язок.



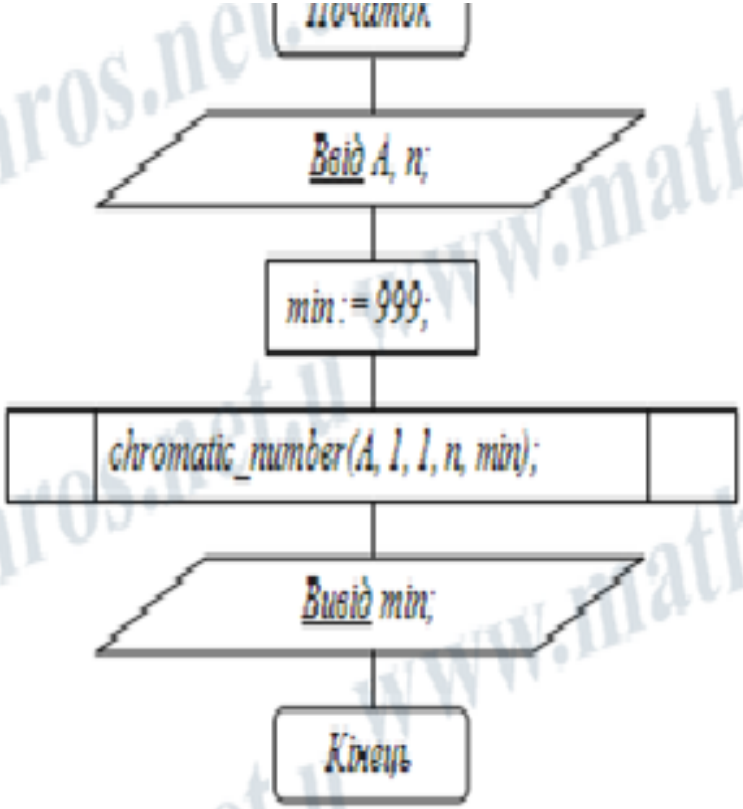
Дерево варіантів

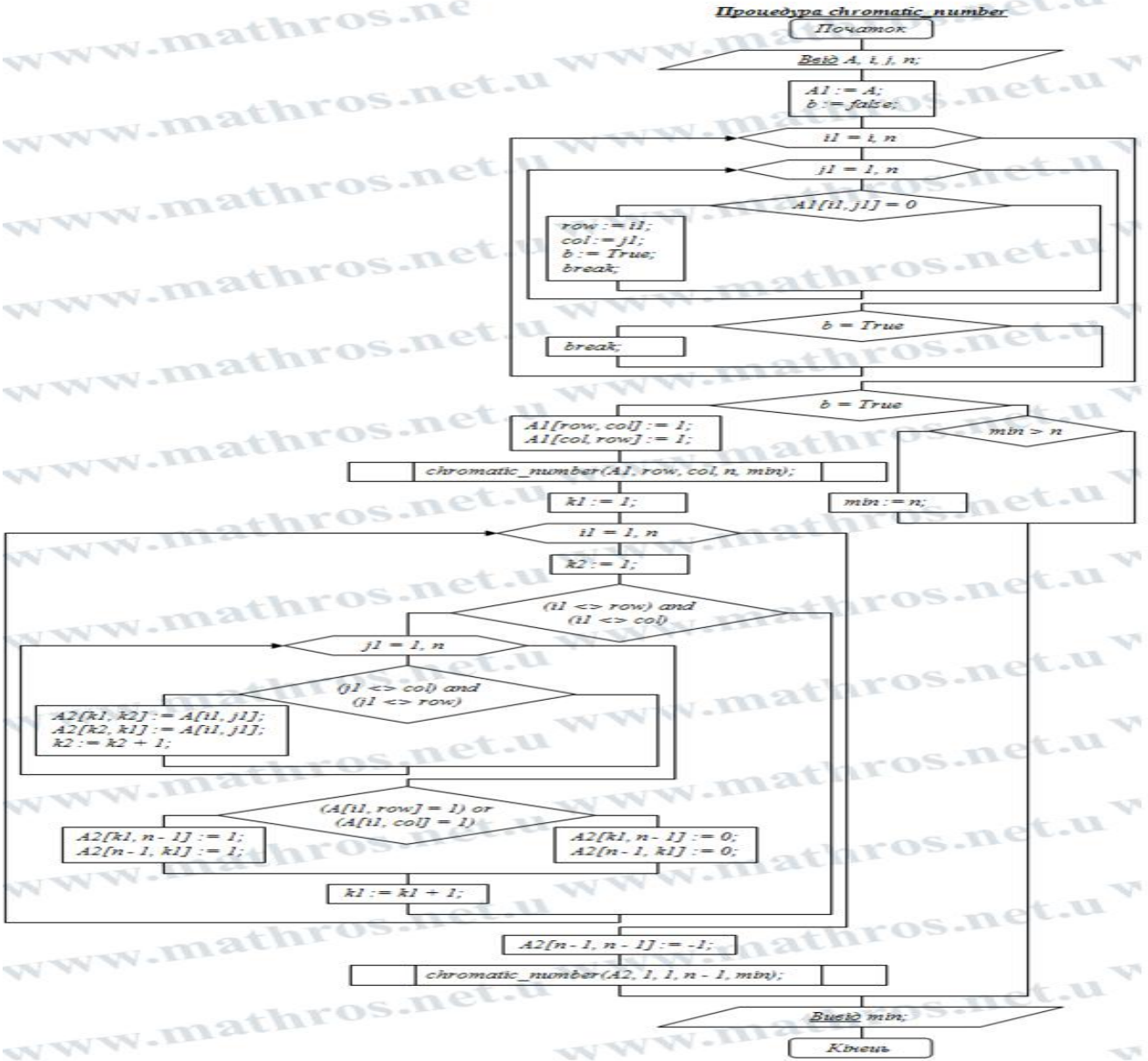
Переглянувши отримане дерево бачимо, що серед його зовнішніх вузлів міститься граф, для якого **хроматичне число** являється мінімальним і дорівнює два (на рисунку даний граф виділений жирними лініями). Звідси, приходимо до висновку, що для того, щоб здійснити правильне розфарбування заданого графа, знадобиться лише два кольори. Покажемо, яким чином це реалізується.

Отже, на першому кроці, розфарбуємо кожну вершину графа з мінімальним **хроматичним числом** в свій колір. Наприклад, вершину *«d»* в червоний колір, і вершину *«1»* – в синій. Після цього, виходячи з того, що вершина *«d»* була отримана в результаті злиття двох вершин *«4»* та *«a»* фарбуємо їх також в червоний колір. І на останньому кроці, з аналогічних міркувань, фарбуємо вершини *«2»* та *«3»* також в червоний колір. В результаті отримаємо:


(2)
2
(2)
3
(1)
1
(2)
4
2
3
1
4
(2)
a
(1)
1
(2)
4
2
3
1
4
b
3
1
a
(2)
d
1
4
(1)
1
2
3
1
4
2
c
1

Правильне розфарбування вершин неорієнтованого графа
Блок-схема алгоритму знаходження числа кольорів необхідного для реалізації правильного розфарбування графа




Процедура chromatic_number
Початок
Ввід A, i, j, n;
A1 := A;
b := false;
i1 = 1, n
j1 = 1, n
A1[i1, j1] = 0
row := i1;
col := j1;
b := True;
break;
b = True
break;
b = True
A1[row, col] := 1;
A1[col, row] := 1;
chromatic_number(A1, row, col, n, min);
min > n
min := n;
k1 := 1;
i1 = 1, n
k2 := 1;
(i1 <> row) and
(i1 <> col)
j1 = 1, n
(j1 <> col) and
(j1 <> row)
A2[k1, k2] := A[i1, j1];
A2[k2, k1] := A[i1, j1];
k2 := k2 + 1;
(A[i1, row] = 1) or
(A[i1, col] = 1)
A2[k1, n - 1] := 1;
A2[n - 1, k1] := 1;
A2[k1, n - 1] := 0;
A2[n - 1, k1] := 0;
k1 := k1 + 1;
A2[n - 1, n - 1] := -1;
chromatic_number(A2, 1, 1, n - 1, min);
Вивід min;
Кінець

§3 Розфарбування графа в n кольорів

Нехай вершини графа позначені a,b,c. Спочатку розфарбовуємо вершину a в колір 1. Потім вершину b в колір 1, якщо b не суміжна з a. Інакше розфарбовуємо b в колір 2. Вершину c розфарбовуємо в колір 1, якщо це можливо, якщо ні перевіряємо колір 2. Тільки якщо жоден не підходить використовуємо 3 колір.

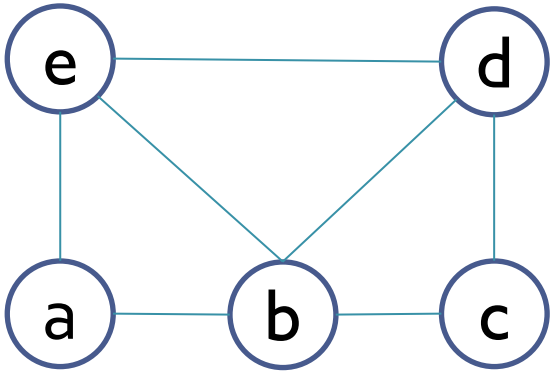
Продовжуємо цей процес, доки це можливо, використовуючи один з n кольорів для кожної нової вершини, причому завжди використовуємо перший можливий колір зі списку.

Кольори

1- червоний

2 – синій

3- зелений



a - червоний

a – червоний
b - синій

a – червоний
b – синій
c - червоний

a – червоний
b – синій
c – червоний
d – зелений

a – червоний
b – синій
c - зелений

a – червоний
b – синій
c – зелений
d – червоний

a – червоний
b – синій
c – зелений
d – червоний
e - зелений