Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра прикладної математики

**КУРСОВА РОБОТА**

**З курсу магістерської кваліфікаційної роботи на тему:**

**«Застосування розв'язку задачі пошуку циклів на графах для побудови маршрутів навчальних польотів»**

Виконала:

Ст. гр. ПМКМ-11

Ткачів О. О.

Перевірив:

Алексєєв В.І.

Львів 2020

***Зміст***

1.Вступ

2. Теорія графів

2.1. Графи та діаграфи

**2.2. Ступінь та відстань**

**2.3. Основні структурні поняття**

**2.4. Дерева**

3. Порівняльний аналіз алгоритмів пошуку циклів

3.1. Визначення

**3.2. Комп’ютерне представлення**

**3.3. Алгоритми**

Висновок

Конфігурація вузлів та з'єднань відбувається у великій різноманітності застосувань. Вони можуть представляти фізичні мережі, такі як електричні ланцюги, дороги або органічні молекули. Вони також використовуються для представлення менш відчутних взаємодій, як це може статися в екосистемах, соціологічних взаємозв'язках, базах даних або в управлінні комп'ютерною програмою.

1.Вступ

Важлива роль у сучасній теорії алгоритмів відведена алгоритмам пошуку, оскільки завдання пошуку даних (пошук даних за заданим ключем, пошук послідовності в послідовності) є одним із тих завдань, що має практичне застосування. Використання алгоритму пошуку для вирішення конкретної задачі є досить складною проблемою, вирішення якої вимагає не тільки вільного володіння цим алгоритмом, але і всебічного розгляду алгоритму, що визначає всі його переваги та недоліки. В комп’ютерних науках алгоритм пошуку - це алгоритм пошуку предмета із заданими властивостями серед колекції елементів, кодованих у комп'ютерну програму, які шукають підказки, щоб повернути потрібне. Елементи можуть зберігатися окремо як записи в базі даних; або можуть бути елементами простору пошуку, визначеними математичною формулою або процедурою, такими як корені рівняння з цілими змінними; або комбінація двох, таких як гамільтонові схеми графа. Алгоритми пошуку віртуальних просторів використовуються в задачі задоволення обмежень, де мета - знайти набір присвоєнь значень певним змінним, які задовольнять конкретні математичні рівняння та нерівності. Вони також використовуються, коли мета - знайти призначення змінної, яке дозволить максимально або мінімізувати певну функцію цих змінних.

2. Теорія графів

***2.1. Графи та діаграфи***

Будь-який математичний об'єкт, що включає точки і зв’язки між ними, може називатися графом. Якщо всі з'єднання односпрямовані, це називається диграфом.

*Означення:* Граф складається з двох множин і . Елементи називаються вершинами (або вузлами). В свою чергу елементи називаються ребрами. Кожен край має набір однієї або двох вершин, пов'язаних з ним, які називаються його кінцевими точками. Кажуть, що край з'єднує кінцеві точки.

*Позначення:* Підписані позначення та (або та ) використовуються для наборів вершин та ребер, коли - не єдиний розглянутий граф.

*Означення:* Якщо вершина є кінцевою точкою краю , тоді, як кажуть, походить на , а - на .

*Означення:* Вершина примикає до вершини , якщо вони з'єднані ребром.

*Означення:* Дві сусідні вершини можуть називатися сусідами.

*Означення:* Суміжні краї - це два ребра, які мають спільну кінцеву точку.

*Означення:* Правильний край - це край, який з'єднує дві різні вершини.

*Означення:* Багатобарвний - це сукупність двох або більше ребер, що мають однакові кінцеві точки.

*Означення:* Просте примикання між вершинами виникає, коли між ними рівно один край.

*Означення:* Кратність кратності між парою вершин і - це кількість ребер між ними.

*Означення:* Самостійна петля - це край, який з'єднує єдину кінцеву точку до себе.

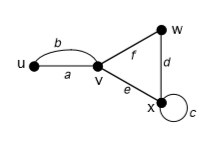
*Приклад 1:* Лінійний малюнок графа показаний на рисунку 1. Він має множину вершин і множину країв . Множина - це багатокрайна грань із кінцевими точками і , а край - самостійна петля.

Рис. 1. Граф

Зауваження:

1. Граф реалізується в площині або в 3-просторі як набір точок, що представляють вершини, і набір вигнутих або прямих відрізків, що представляють ребра. Кривизна або довжина такого відрізка лінії не має значення. Однак якщо вказано напрямок, це значне значення.
2. Іноді граф параметризований так, що кожне ребро розглядається як гомеоморфне зображення реального інтервалу (за винятком випадків, коли для автоциклів кінцеві точки 0 і 1 мають однакове зображення).

*Прості графи*

Більшість теоретичних теорій графів стосуються конкретно простих графів. Почасту це тому, що багато проблем щодо загальних графів можна звести до проблем щодо простих графів.

*Означення:* Простий граф - це граф, який не має самокрут або багатокрайників.

*Означення:* Тривіальний граф - це граф, що складається з однієї вершини і не має ребер.

*Означення:* Нульовий граф - це граф, набір вершин та ребер порожній.

*Позначення країв для простих примикань та для кількох країв*

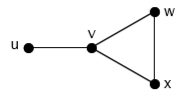
*Позначення:* Край, що з'єднує вершини і графа, може позначатись -супозицією, якщо це єдиний такий край. Іноді в цій ситуації використовується впорядкована пара (, ) замість . Щоб уникнути неоднозначностей, коли існують багатокрайники або коли завгодно інше, країв загального графа можуть бути вказані власні назви, як на рисунку 1 вище.

Рис. 2. Простий граф.

*Приклад 2:* Простий граф, показаний на рисунку 2, має крайову множину .

*Загальні графи*

Багато додатків вимагають непрості графи як моделі. Крім того, деякі непрості графи відіграють важливу роль у теоретичних побудовах, особливо у побудові графічних графів (простих та непростих) на поверхнях.

термінологічне зауваження: Хоча термін "граф" означає, що дозволено самокручування та багатокрайники, іноді для наголосу використовується термін загальний граф.

*Означення:* Граф, що не містить «людей», - це граф, що не має циклів самооцінки. (Можливо, він має кілька ребер.) Іноді граф «безлюдник» називають мультиграфом.

*Означення:* Діполь - це граф «безлюддя» з двома вершинами та краями, що з'єднують їх.

*Означення:* Букет - це граф з однією вершиною та власними петлями.

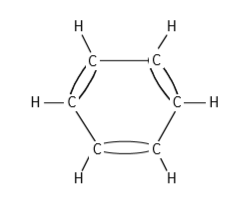
*Приклад 3:* Граф «безлюддя» на рисунку 3 зображує молекулу бензолу .

Рис. 3. Модель графа для бензольного кільця.

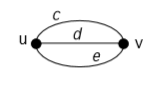


Рис. 4. Граф бездоганності .

*Приклад 4:* Диполь показаний на рисунку 4.

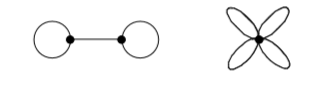
*Приклад 5:* На рисунку 5 зображено два графи із самозаймами.

Рис. 5. Граф з гантелями та букет .

*Атрибути*

Дозволяючи графам мати додаткові атрибути поза вершинами та ребрами, вони можуть слугувати математичними моделями для найрізноманітніших програм. Два найпоширеніші додаткові атрибути краю - це напрямок ребер та вага краю. Ще один поширений атрибут (для країв або вершин) – колір (забарвлення графів).

*Означення:* Атрибут вершини - це функція від набору вершин до деякого набору можливих значень атрибутів.

*Означення:* Атрибут краю - це функція від набору ребер до деякого набору можливих значень атрибутів.

*Диграфи*

Край між двома вершинами створює зв’язок одразу двох протилежних почуттів. Призначення напрямку робить одне з цих відчуттів вперед, а інше назад. Напрямок перегляду як атрибута краю частково мотивовано його впливом на комп’ютерну реалізацію алгоритмів графів. Більше того, з математичної точки зору, щодо спрямованих графів як доповнених графів полегшується перегляд певних результатів, які, як правило, встановлюються окремо для графів, а для діаграфів - як єдиний результат, що стосується обох.

*Означення:* Спрямований край (або дуга) - це край , одна з кінцевих точок якого позначена як хвіст, а інша кінцева точка позначена як голова. Вони позначаються відповідно головою () та хвостом ().

Кажуть, що спрямований край спрямований від хвоста і спрямований до голови. (Хвіст і голова спрямованої петлі - однакова вершина.) D20: Багатодухова дуга - це сукупність двох або більше дуг, що мають однаковий хвіст і однакову голову.

*Означення:* Диграф (або спрямований граф) - це граф, кожен із ребер якого спрямований.

*Означення:* Простий диграф - це диграф, що не має самопіль і без дуг.

*Означення:* Змішаний граф (або частково спрямований граф) - це граф, який має як непрямі, так і спрямовані краї. У змішаному графі за допомогою немодифікованого терміна край уникає вказувати, чи край спрямований чи непрямий.

*Означення:* Основним графом спрямованого або частково спрямованого графа є граф, який є результатом видалення всіх позначень голови та хвоста з спрямованих країв (тобто видалення всіх реберних напрямків).

*Упорядковано-парне представлення дуг*

*Позначення:* У простому діаграфі дугу від вершини до вершини зазвичай позначають (, ) (або іноді ). Коли можливі багатодугові дуги, часто потрібно використовувати різні назви.

*Обчислювальна примітка:* (Застереження для дизайнерів програмного забезпечення) З точки зору об'єктно-орієнтованого програмного забезпечення, впорядковане парне представлення дуг у диграфі трактує графи як різний клас об'єктів із графів. Це може серйозно підірвати повторне використання програмного забезпечення. Великі частини комп’ютерного коду, можливо, доведеться переписати, щоб адаптувати алгоритм, який спочатку був розроблений для того, щоб диграф працював на непрямому графі.

Представлення впорядкованої пари також може виявитись незручним у реалізації алгоритмів, для яких графи або діаграфи є динамічними структурами (тобто вони змінюються під час алгоритму). Кожен раз, коли напрямок на певному краї повинен бути зворотним, пов’язану впорядковану пару потрібно видалити та замінити її зворотним. Ще гірше, якщо спрямований край повинен стати ненаправленим, то впорядковану пару потрібно замінити на не упорядковану пару. Аналогічно, непрямі та спрямовані краї частково спрямованого графа потребують двох різних типів об'єктів.

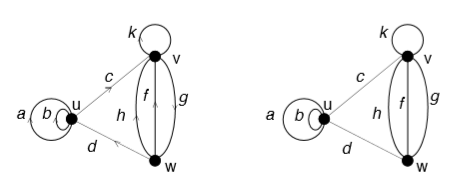
*Приклад 6:* Диграф зліва на рисунку 6 має правильний граф, як його нижній граф. Диграф має дві багатодугові дуги: і .

Рис. 6. Диграф та його граф.

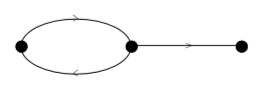
*Приклад 7:* Простий диграф може мати по одній дузі в кожному напрямку між двома вершинами, як проілюстровано на рисунку 7.

Рис. 7. Простий диграф, граф якого лежить не просто.

*Забарвлення вершини*

Коли набір вершин графа розділений, осередкам розділу зазвичай присвоюються різні кольори.

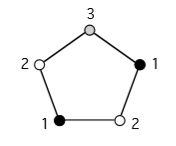
*Означення:* Забарвлення вершин графа - це функція від вершин , встановлених вершинами, до множини , елементи якої називаються кольорами.

*Означення:* Колір вершин належний, якщо двом суміжним вершинам завжди присвоєні різні кольори.

*Означення:* Граф є -кольоровим, якщо він має належне забарвлення вершин або меншими кольорами.

*Означення:* (Вершинне) хроматичне число графа , позначене , є найменшим числом кольорів таким, що є -кольоровим.

Зауваження:

Визначення кольорового крайового, -кольорового крайового та крайового хроматичного числа, позначені , отримуються просто заміною слова "вершини" словом "краю" у вказаних вище визначеннях.

*Приклад 8:* Граф на рисунку 8 зображений 3-ма кольоровим набором вершин. Оскільки він не є двобарвним, його хроматичне число дорівнює 3. Також граф легко бачить як 3-краєвий і, очевидно, не є 2-краєвим кольоровим; отже, .

Рис. 8. Граф з .

**2.2. Ступінь та відстань**

Два найбільш фундаментальних поняття в теорії графа - це ступінь вершини та відстань між двома вершинами.

*Ступінь*

*Означення:* Ступінь (або валентність) вершини у графі , позначена , - кількість правильних ребер, що падають на плюс удвічі більша кількість самокрут. (Для простих графів, звичайно, ступінь - це просто кількість сусідів.)

Застосування теорії графів до фізичної хімії мотивує використання терміна валентність як альтернативу ступеня. Таким чином, вершину ступеня називають також -валентною вершиною.

*Означення:* Градусна послідовність графа - це послідовність, утворена шляхом упорядкування ступенів вершин у не зменшуваному порядку.

*Означення:* Невизначена вершина у графі - це кількість дуг, спрямованих на ; Поверхня вершини - це кількість дуг, спрямованих від . Кожна самокрутка в відлічує одну в бік і одну до невдоволення.

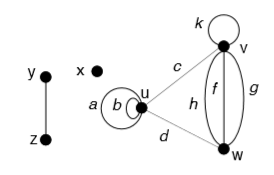
*Означення:* Ізольована вершина в графі - це вершина ступеня 0.

Рис. 9. Граф із ступеневою послідовністю .

*Приклад 9:* Граф на рисунку 9 має послідовність ступенів . Вершини і мають ступінь 6.

*Приклад 10:* На рисунку 10 нижче показані градуси та пониження градусу.

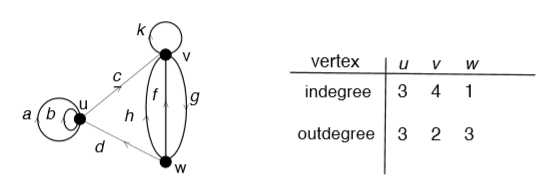


Рис. 10. Не градуси та перевищення вершин вершини диграфа.

Слідують наступні факти:

1. Сума градусів вершин графа вдвічі перевищує кількість ребер.
2. У кожному графі кількість вершин, що мають непарний ступінь, є парним числом.
3. Нетривіальний простий граф повинен мати принаймні одну пару вершин, градуси яких рівні.
4. У графі сума нестандартних значень і сума перевищення дорівнює кількості ребер.
5. Послідовність градусів графа є цілою, не зменшується послідовністю невід’ємних цілих чисел, сума яких парна.
6. І навпаки, будь-яка не зменшується, негативна послідовність цілих чисел, сума яких рівна - ступінь послідовності деякого графа, але не обов'язково простого графа.

*Прогулянки, стежки та ходи*

*Означення:* Прогулянка по графу - це чергується послідовність вершин і ребер,

такі, що при вершини і є кінцевими точками краю . Якщо, крім того, край спрямований від до , то - спрямована хода. У простому графі прогулянка може бути представлена ​​просто переліченням послідовності вершин: такий, що при вершини і суміжні. Початкова вершина – . Кінцева вершина (або кінцева вершина) - . Внутрішня вершина - це вершина, яка не є ні початковою, ні остаточною.

*Означення:* Довжина прогулянки - це кількість ребер (підрахунок повторів).

*Означення:* Прогулянка закрита, якщо початкова вершина також є остаточною вершиною; в іншому випадку вона відкриті.

*Означення:* Слід у графі - це прогулянка таким чином, що жоден край не виникає більше одного разу.

*Означення:* Ейлеровий слід у графі - прогулянка, яка містить кожен край рівно один раз.

*Означення:* Шлях у графі - це слід, такий, що жодна внутрішня вершина не повторюється.

*Означення:* Цикл - це закритий шлях довжиною щонайменше 1.

*Означення:* Тривіальна хода, стежка або шлях складається з однієї вершини і немає ребер.

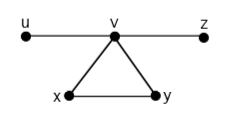
*Приклад 11:* На графі, зображеному на рисунку 11, вершинна послідовність являє собою прогулянку, яка не є стежкою, а послідовність вершин позначає слід, який не є стежкою.

Рис. 11. Граф.

*Відстань та зв’язок*

*Означення:* Відстань між двома вершинами в графі - це довжина найкоротшої ходи між ними.

*Означення:* Спрямована відстань від вершини до вершини в діаграфі - це довжина найкоротшого спрямованого ходу від до .

*Означення:* Граф пов'язаний, якщо між кожною парою вершин є прогулянка.

*Означення:* Диграф (слабко) пов'язаний, якщо підключений його граф.

*Означення:* Диграф є сильно пов’язаним, якщо від кожної вершини до іншої вершини є спрямована хода.

*Означення:* Ексцентриситет вершини у зв’язаному графі - це його відстань до вершини, найбільш віддаленої від .

*Означення:* Радіус з'єднаного графа - це його мінімальний ексцентриситет.

*Означення:* Діаметр підключеного графа - його максимальний ексцентриситет.

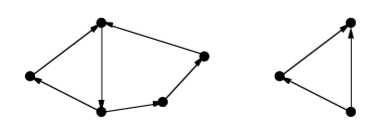
*Приклад 12:* Диграф, показаний зліва на рисунку 12, сильно пов'язаний; правопис підключений праворуч, але сильно не пов'язаний.

Рис. 12. Сильно пов'язаний диграф та слабко пов'язаний.

**2.3. Основні структурні поняття**

Ми стурбовані можливою еквівалентністю двох графів, симетріями окремого графа та можливим появою одного графа в іншому графі.

Ізоморфізм

У концепції два графи є ізоморфними, якщо вони структурно однакові, це означає, що вони відповідають у всіх структурних деталях. Формальна відповідність вершини до вершини та краю до краю називається ізоморфізмом.

*Означення:* Ізоморфізм між двома простими графами і - вершинна біекція така, що для вершина примикає до вершини у графі , якщо і лише тоді, коли дорівнює поруч з у графі . Неоднозначно також існує крайова біекція така, що . D50: Ізоморфізм між двома загальними графами і є парою біекцій і таким, що для кожної пари вершин множина ребер в , що приєднується і відображається бієктивно на множині ребер у , що з'єднує вершини та .

*Означення:* Ми говоримо, що і - ізоморфні графи, і пишемо , якщо є ізоморфізм .

*Означення:* Матриця примикання для простого графа , вершини якого явно впорядковані - матриця така, що

(1)

*Означення:* Властивість, пов'язана з усіма графами, є інваріантною ізоморфізмом, якщо вона має однакове значення (або однакове) для будь-яких двох ізоморфних графів.

*Приклад 13:* Два графи на рисунку 13 є ізоморфними за відображенням

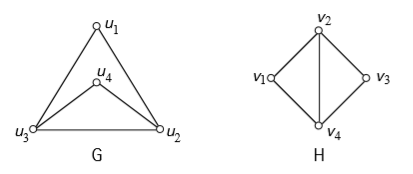
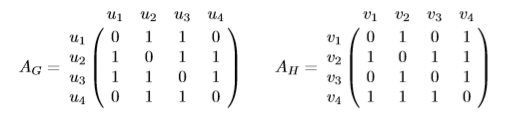
Якщо перевернути вершину графа вниз до низу і обернути зображення на чверть обороту проти годинникової стрілки, то отримане зображення графа “виглядає так само” як граф . Їх матриці суміжності:

Рис. 13. Два ізоморфні графи.



Ми спостерігаємо, що перенесення рядків та , а також транспонування стовпців та перетворює матрицю в матрицю .

*Приклад 14:* Два графи на рисунку 14 є ізоморфними, навіть якщо малюнки виглядають досить різними. Мітки вершин вказують на ізоморфізм.

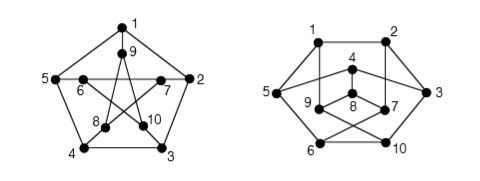


Рис. 14. Два ізоморфні графи, які виглядають досить різними.

*Приклад 15:* На рисунку 15 показані два неізоморфні графи з однаковою ступеневою послідовністю. (Неважко показати, що зв’язок є інваріантом ізоморфізму.)

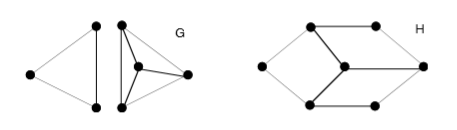


Рис. 15. Два графи, ступінь послідовностей яких обох

.

Слідують наступні факти:

1. Розгляд усіх можливих біекцій вершинних наборів двох -вершинних графів вимагає кроків .
2. Хоча відомі деякі швидкі евристики, не існує відомого багаточленного алгоритму для тестування ізоморфізму графа.
3. Кількість вершин, кількість ребер та ступінь послідовності - це всі інваріанти ізоморфізму. З іншого боку, наявність однакових значень для всіх трьох цих інваріантів не означає, що два графи є ізоморфними.
4. Кожна сума рядків (і сума стовпців) у матриці суміжності дорівнює ступеню відповідної вершини.

*Автоморфізми*

Поняття симетрії в графі формалізується з точки зору ізоморфізмів графа до самого себе.

*Означення:* Автоморфізм графа - це ізоморфізм самого графа.

*Означення:* Орбіта вершини графа - це множина всіх вершин , така що існує автоматизм такий, що .

*Означення:* Орбіта краю графа дорівнює множина всіх ребер така, що існує автоморфізм такий, що .

*Означення:* Граф є перехідним шляхом, якщо всі вершини знаходяться на одній орбіті.

*Означення:* Граф є перехідним по краю, якщо всі ребра знаходяться на одній орбіті.

Слідують наступні факти:

1. Вершина орбіти розділяє набір вершин графа.
2. Край орбіти розділяє крайовий набір графа.

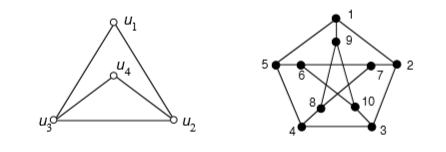
*Приклад 16:* Для графа зліва на рисунку 16 орбіти вершин - і , а крайові орбіти - і . Граф праворуч - вершинно-перехідний та реброперехідний.

Рис. 16. Граф та граф Петерсена.

*Підграф*

*Означення:* Підграф графа є графом таким, що та . (Зазвичай, будь-який граф, ізоморфний підграфу , також називається підграфом .)

*Означення:* У графі індукований підграф на наборі вершин позначається , має свій набір вершин, і він містить кожен край , кінцеві точки якого є у . Тобто

*Означення:* Підграф графа - це проміжний підграф, якщо . (Крім того, якщо ізоморфний відносно проміжного підграфа , можна сказати, що охоплює .)

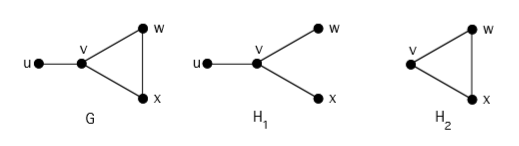
*Означення:* Компонент графа є підключеним підграфом таким чином, що жоден підграф , який належним чином містить , не з'єднаний. Іншими словами, компонент - це максимально пов'язаний підграф.

Рис. 17. Проміжний підграф та індукований підграф .

*Приклад 17:* Для графа на рисунку 17, - це проміжний підграф, але не індукований підграф, а - індукований підграф, але не проміжний підграф.

Слідують наступні факти:

1. Нехай графічний ізоморфізм, а - підграф . Тоді обмеження на підграф є ізоморфізмом на його зображення .
2. Якщо граф є підграфом графа , але не підграфом графа , то . Це є наслідком попереднього факту.

*Графічні операції*

Операції додавання та видалення вершин та ребер графа розглядаються як основні операції, оскільки вони є основою для інших операцій, які можна назвати вторинними операціями.

*Означення:* Операція додавання вершини до графа , така що , отримує новий граф з набором вершин і набором ребер , який позначається . (Нова вершина не має сусідів.)

*Означення:* Операція видалення вершини з графа не тільки видаляє вершину , але й видаляє кожен край, у якого є кінцевою точкою. Отриманий граф позначається .

*Означення:* Операція додавання ребра (або ) до графа , що приєднується до вершин і , дає новий граф з набором вершин і набором ребер (або ), що позначається (або ).

*Означення:* Операція видалення ребра (або ) з графа видаляє лише цей край. Отриманий граф позначається (або ).

*Означення:* Вершина розрізу (або точка вирізання) - це вершина, видалення якої збільшує кількість компонентів.

*Означення:* Передній край - це край, видалення якого збільшує кількість компонентів.

*Означення:* Крайовим доповненням простого графа є граф (альтернативно позначається ), який має такий самий набір вершин, що і , таким, що є ребром , якщо і лише якщо він не є краєм .

*Означення:* З'єднання (або призупинення) двох графів і позначається . Він має такі вершини і крайові множини:

*Означення:* Декартовий добуток (або добуток) двох графів і позначається . Множина вершин і множина ребер такі:

Кінцеві точки краю - вершини та , де і - кінцеві точки ребра . Кінцевими точками краю є вершини і , де і є кінцевими точками краю .

*Означення:* Графічне об'єднання двох графів і є графом , набір вершин і набір ребер є з'єднаннями, що перетинаються, відповідно множин вершин і множин ребер і .

*Означення:* -кратне самоз'єднання - це ітераційне роз'єднане з'єднання з копій графа .

Рис. 18. Доповнення краю.

*Приклад 18:* Рисунок 18 ілюструє операцію доповнення краю.

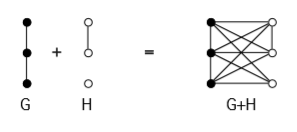


Рис. 19. Доповнення краю.

*Приклад 19:* Рисунок 19 ілюструє операцію приєднання.

*Приклад 20:* На рисунку 18 вершина у верхньому лівому куті креслення графа є вершиною розрізу, а край від цієї вершини до центральної вершини є передній край.

*Приклад 21:* Рисунок 20 ілюструє роботу продукту.

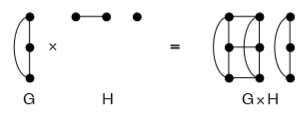


Рис. 20. декартовий продукт.

**2.4. Дерева**

Дерева важливі для структурного розуміння графів та алгоритміки обробки інформації, і вони відіграють центральну роль у проектуванні та аналізі підключених мереж.

*Ациклічні графи*

*Означення:* Дерево - це зв'язаний граф без циклів (тобто, ациклічний).

*Означення:* Ліс - це (не обов'язково з'єднаний) граф без циклів.

*Означення:* Центральна вершина в графі - це вершина, ексцентриситет якої дорівнює радіусу графа.

*Означення:* Центр графа - це підграф, індукований на його наборі центральних вершин.

Класично слова центр і біцентр використовувались для позначення набору центральних вершин дерева, коли було лише одна вершина або дві вершини відповідно.

*Приклад 22:* Граф зліва на рисунку 21 - дерево; інші два графи - ні.

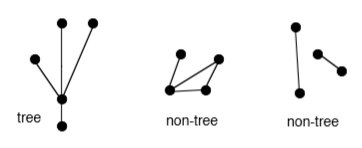
Як факт можна вважати, що центр дерева ізоморфний до або до .

Рис. 21. Дерево та два не дерева.

*Дерева як підграф*

Кілька різних алгоритмів вирішення проблем передбачають вирощування дерева в межах графа, одного ребра та однієї вершини. Усі ці методи - це відмітки та розширення однієї і тієї ж основної схеми вирощування дерев, наведеної в цьому розділі.

Для даного дерева у графі , краї та вершини називаються ребрами дерев і вершинами дерева, а краї та вершини , які не є у , називаються недеревними краями та недеревними вершинами .

*Означення:* Прикордонний край для даного дерева на графі - це недеревний край з однією кінцевою точкою в і однією кінцевою точкою, не з .

*Означення:* Діапазон, що охоплює граф , - це підпіграф , що є деревом.

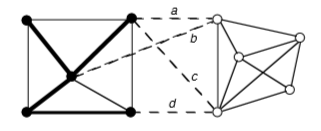
*Приклад 23:* Для графа на рисунку 22 краї дерева зображуються жирним шрифтом. Вершини дерева чорні, а недеревні вершини - білі. Прикордонні ребра для , що виступають пунктирними лініями, є ребрами . Рівні краї - це недеревні краї, які не є прикордонними краями для .

Рис. 22. Дерево з прикордонними краями a,b,c і d.

Зауважимо, що коли будь-який з прикордонних ребер на рисунку 22 додається до дерева , то отриманий підграф все ще є деревом. Ця властивість є загальним, і застосовуючи її ітераційно, формує серцевину схеми вирощування дерев цього розділу.

Візьмемо до уваги наступний факт: нехай - дерево у графі , а - прикордонний край для . Тоді підграф , утворений додаванням краю до дерева , є деревом. (Формально додавання прикордонного краю до дерева включає в себе додавання нової вершини до поточного дерева , тобто, його недеревна кінцева точка.)

*Основний алгоритм вирощування дерев*

Основна схема вирощування дерев використовує мітки вершин, щоб відслідковувати порядок додавання вершин до дерева.

*Термінологічна примітка:* Стандартне (на основі 0) маркування вершин -вершинного графа - це присвоєння один до одного цілих чисел вершинам цього графа.

*Алгоритм 1: Основне вирощування дерев за допомогою вершинних міток*

Вхід: граф та початкова вершина .

Вихід: дерево , що охоплює , і стандартне маркування вершини .

Ініціалізуйте дерево як вершину .

Напишіть мітку 0 на вершину v.

Ініціалізуйте лічильник міток

Поки дерево ще не має проміжного компонента

Виберіть прикордонний край для дерева .

Нехай буде кінцевою точкою краю що лежить поза .

Додати край та вершину до дерева .

Напишіть мітку i на вершину .

Вернеться дерево і маркування вершини

Зауваження:

Унікальність виведення дерева з дереворощування

Без правила вибору прикордонного краю (включаючи спосіб розірвати зв'язки), дерево виводу з алгоритму 1 не було б унікальним (у такому випадку багато вчених-комп'ютерів будуть вагатися з використанням терміна алгоритм). Унікальність виводу залежить від певного пріоритету за замовчуванням, заснованого на впорядкованості ребер (і вершин) у структурі даних, обраній для реалізації алгоритму. Пріоритет за замовчуванням використовується, коли не задається інше правило, і як спосіб розриву зв'язків, що залишилися від інших правил.

Слідують наступні факти:

1. Якщо виконання основного алгоритму вирощування дерев починається з вершини графа , то підграф, що складається з мічених вершин і країв дерева, є діючим деревом компонента .
2. Граф пов'язаний тоді і лише тоді, коли основний алгоритм вирощування дерев позначає всі його вершини.

*Визначення пріоритету вибору краю*

Алгоритм вирощування пріоритетів дерев, алгоритм 2, є пріоритетом базового вирощування дерев.

*Алгоритм 2: Вирощування дерев з пріоритетним краєм*

Вхід: приєднаний граф , початкова вершина та правило для визначення пріоритетності кордонів кордону.

Вихід: охоплююче дерево і стандартне маркування .

Ініціалізуйте дерево як вершину .

Ініціалізуйте набір прикордонних країв для дерева як порожній.

Запишіть мітку 0 на вершину .

Ініціалізуйте лічильник міток

Поки дерево ще не проходить

Оновіть набір прикордонних країв для .

Нехай граничним краєм для з найвищим пріоритетом.

Нехай - незамічена кінцева точка краю .

Додати дерево (і вершину ) до дерева .

Напишіть мітку на вершину .

Повернеться дерево з його вершинним маркуванням.

Слідує факт, що різні правила визначення пріоритетності прикордонних країв породжують різні охоплюючі дерева: дерево глибини першого пошуку (пріоритет останнього першого), широке дерево пошуку (перше пріоритетне перше), первинне дерево (пріоритет з найменшими витратами) та дерево Дійкстра (пріоритет, найближчий до кореня).

3. Порівняльний аналіз алгоритмів пошуку циклів

В комп’ютерних науках виявлення циклу або пошук циклу - це алгоритмічна проблема пошуку циклу в послідовності ітераційних значень функції.

Для будь-якої функції , яка відображає кінцевий набір до себе, і будь-яке початкове значення в , послідовність ітераційних значень функції

в кінцевому підсумку необхідно використовувати одне і те ж значення двічі: має бути якась пара різних індексів і , така що . Як тільки це відбувається, послідовність повинна продовжуватися періодично, повторюючи однакову послідовність значень від до . Пошук циклу - це проблема знаходження і , заданих і .

Відомо декілька алгоритмів пошуку циклів швидко та з малим використанням пам'яті. Алгоритм «черепахи і зайця» Роберта Флойда переміщає два покажчики з різною швидкістю через послідовність значень, поки вони обидва не вказують на однакові значення. Як варіант, алгоритм Брента заснований на ідеї експоненціального пошуку. І алгоритм Флойда, і Брента використовують лише постійну кількість комірок пам'яті і приймають ряд оцінок функції, пропорційне відстані від початку послідовності до першого повторення. Кілька інших алгоритмів вилучають більшу кількість пам'яті для меншої кількості функцій.

Програми виявлення циклів включають тестування якості генераторів псевдовипадкових чисел та криптографічних хеш-функцій, алгоритмів теорії чисельних чисел, виявлення нескінченних циклів у комп’ютерних програмах та періодичних конфігурацій у клітинних автоматах та автоматизований аналіз форм пов'язаних структур даних списку.

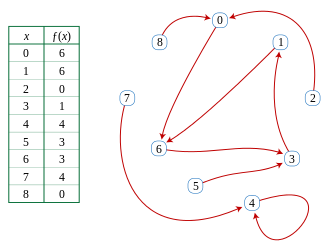
Як приклад, на рисунку 23 показана функція , яка відображає послідовність до себе. Якщо починається з і повторно застосовується , видно послідовність значень

Рис. 23. Функція від та до послідовності { та відповідний функціональний граф

Цикл у цій величині значень становить .

3.1. Визначення

Нехай – довільна скінченна множинна, - будь-яка функція від до себе, а - будь-який елемент з . Для будь-якого , нехай . Нехай - найменший показник, такий, що значення з'являється нескінченно часто в послідовності значень , а (довжина циклу) - найменше додатне ціле число, таке що . Проблема пошуку циклу - це завдання знаходження і .

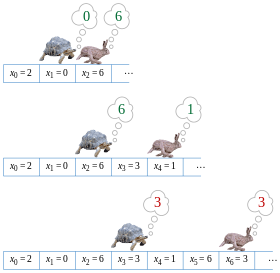
Звертаючись до теорії графів, побудувавши функціональний граф (тобто спрямований граф, у якому кожна вершина має єдине вихідне ребро), вершинами якого є елементи та ребра яких відображають елемент у відповідне значення функції, як показано на рисунку 23. Безліч вершин, доступних від початкової вершини , утворюють підграф з формою, що нагадує грецьку букву Rho (): шлях довжиною від до циклу з вершиною .

**3.2. Комп’ютерне представлення**

Як правило, не буде вказано у вигляді таблиці значень, як це показано на рисвунку 23. Скоріше, алгоритму пошуку циклу може бути наданий доступ або до послідовності значень , або до підпрограми для обчислення . Завдання полягає у знаходженні та , вивчаючи якнайменше значень із послідовності або виконуючи якомога менше викликів підпрограми. Як правило, важливим є також складність простору алгоритму для проблеми пошуку циклу: найбільша проблема, використати об'єм пам'яті, значно менший, ніж потрібно для зберігання всієї послідовності.

У деяких додатках, зокрема, у алгоритмі Полларда Rho для цілочислової факторизації, алгоритм має набагато обмежений доступ до та . Наприклад, у алгоритмі Полларда, , це множина цілих чисел, модульованих невідомим простим коефіцієнтом числа, що підлягає факторизації, тому навіть розмір невідомий алгоритму. Щоб дозволити використовувати алгоритми пошуку циклу з такими обмеженими знаннями, вони можуть бути розроблені на основі наступних можливостей. Спочатку передбачається, що алгоритм містить у своїй пам'яті об'єкт, що представляє вказівник на початкове значення . На будь-якому кроці він може виконати одну з трьох дій: він може скопіювати будь-який вказівник, який він має, на інший об’єкт у пам'яті, він може застосувати та замінити будь-який його покажчик покажчиком на наступний об’єкт у послідовності, або він може застосувати підпрограму для визначення того, чи є два його вказівники рівними значеннями в послідовності. Дія тесту на рівність може включати деякі нетривіальні обчислення: наприклад, в алгоритмі Полларда реалізується тестування, чи різниця між двома збереженими значеннями має нетривіальний найбільший спільний дільник з числом, яке має враховуватися. У цьому контексті за аналогією з моделлю обчислювальної машини, алгоритм, який використовує лише копіювання покажчика, просування в межах послідовності та тести рівності, можна назвати алгоритмом покажчика.

**3.3. Алгоритми**

Якщо введення подано як підпрограму для обчислення , проблема виявлення циклу може бути вирішена тривіально за допомогою лише додатків функцій , просто обчисливши послідовність значень та використовуючи структуру даних, таку як хеш-таблиця для зберігання цих значень і перевірити, чи було збережено кожне наступне значення. Однак складність простору цього алгоритму пропорційна , зайво великій. Крім того, для реалізації цього методу як алгоритму вказівника потрібно буде застосувати тест рівності до кожної пари значень, що призводить до загального квадратичного часу. Таким чином, дослідження в цій області сконцентрувались на двох цілях: використанні менше місця, ніж цей наївний алгоритм, і пошуку алгоритмів вказівника, які використовують менше тестів на рівність.

*«Черепаха і заєць» Флойда*

Алгоритм пошуку циклу Флойда - це алгоритм покажчика, який використовує лише два покажчики, які рухаються по послідовності з різною швидкістю. Його також називають "алгоритмом черепахи та зайця", що натякає на байку Езопа "Черепаха та Заєць".

Алгоритм названий на честь Роберта У. Флойда, якому його винахід приписував Дональд Кнут. Однак алгоритм не відображається у опублікованій роботі Флойда, і це може бути неправильним розподілом: Флойд описує алгоритми для перерахування всіх простих циклів у спрямованому графі в документі 1967 року, але ця стаття не описує проблему пошуку циклу у функціональних графах, що є предметом цієї статті. Насправді вислів Кнута (1969 р.) про приписування його Флойду без цитування - це перша відома поява в друку, і, таким чином, це може бути народною теоремою, не приписуваною жодній особі.

Рис. 24. Алгоритм пошуку циклу "черепаха та заєць" Флойда, застосований до послідовності

Ключове розуміння алгоритму полягає в наступному. Якщо існує цикл, то для будь-яких цілих чисел і , , де - довжина петлі, яку потрібно знайти, і - індекс першого елемента циклу. Виходячи з цього, може бути показано, що для деякого тоді і лише тоді, коли . Таким чином, алгоритму потрібно лише перевірити наявність повторних значень цієї спеціальної форми, одне вдвічі більше від початку послідовності, ніж інше, щоб знайти період повторення, кратний . Після того, як знайдений, алгоритм відслідковує послідовність від його початку, щоб знайти перше повторне значення у послідовності, використовуючи той факт, що ділить і, отже, . Нарешті, як тільки значення відоме це тривіально знайти довжину найкоротшого повторюваного циклу, шукаючи перше положення , для якого .

Таким чином, алгоритм підтримує два покажчики на задану послідовність, один (черепаха) на , а другий (заєць) на . На кожному кроці алгоритму він збільшує на один, переміщуючи черепаху на один крок вперед, а заєць на два кроки вперед в послідовності, а потім порівнює значення послідовності в цих двох покажчиках. Найменше значення , для якого черепаха та заєць вказують на однакові значення, є бажаним значенням .

*Алгоритм Брента*

Річард П. Брент описав альтернативний алгоритм виявлення циклу, який, як і алгоритм «черепахи та зайця», потребує лише двох покажчиків у послідовності. Однак він ґрунтується на іншому принципі: пошук найменшої потужності двох , більших за і . Для , алгоритм порівнює з кожним наступним значенням послідовності до наступної потужності двох, зупиняючись, коли знаходить відповідність. Він має дві переваги порівняно з алгоритмом «черепахи та зайця»: він знаходить правильну довжину циклу безпосередньо, а не потребує його пошуку на наступному етапі, а його кроки передбачають лише одну оцінку , а не три.

Кількість оцінок функції ніколи не може бути вищою, ніж для алгоритму Флойда. Брент стверджує, що в середньому його алгоритм пошуку циклу працює приблизно на 36% швидше, ніж Флойд, і що він прискорює алгоритм Полларда на 24%. Він також проводить середній аналіз випадків для рандомізованої версії алгоритму, в якій послідовність індексів, простежених повільнішими двома вказівниками, - це не сили двох самих, а скоріше рандомізоване кратне значення двох. Хоча його головне призначення було в алгоритмах цілої факторизації, Брент також обговорює програми для тестування генераторів псевдовипадкових чисел.

*Алгоритм Госпера*

Алгоритм Р. В. Госпера знаходить період, але не є початковою точкою першого циклу. Його головна особливість полягає в тому, що він ніколи не створює резервного копіювання для переоцінки функції генератора, і є економічним як в просторі, так і в часі. Наприклад, якщо, в апріорі, відомо, що функція генератора має період 232, то 33 слова буде достатньо.

*Часово-космічні компроміси*

Ряд авторів вивчав методи виявлення циклів, які використовують більше пам'яті, ніж методи Флойда та Брента, але швидше виявляють цикли. Як правило, ці методи зберігають декілька раніше обчислених значень послідовності та перевіряють, чи кожне нове значення дорівнює одному з раніше обчислених значень. Для того, щоб зробити це швидко, вони зазвичай використовують хеш-таблицю або подібну структуру даних для зберігання раніше обчислених значень, а отже, не є алгоритмами покажчиків: зокрема, вони зазвичай не можуть бути застосовані до Rho-алгоритму Полларда. Де ці методи відрізняються, це те, як вони визначають, які значення зберігати.

Брент вже описує варіанти його методики, в яких індекси збережених значень послідовності є потужностями числа , крім двох. Вибираючи для числа, близького до одиниці, і зберігаючи значення послідовності в індексах, що знаходяться поблизу послідовності послідовних потужностей , алгоритм пошуку циклу може використовувати ряд оцінок функції, що знаходиться в межах довільно малого коефіцієнта від оптимального .

Sedgewick, Szymanski і Yao забезпечують метод, який використовує комірки пам'яті і вимагає в гіршому випадку лише оцінки функцій для деякої константи , для якої вони виявляються оптимальними.Методика включає підтримання числового параметра , зберігання в таблиці лише тих позицій у послідовності, які кратні , та очищення таблиці та подвоєння , коли було збережено занадто багато значень.

Кілька авторів описали розрізнені точкові методи, які зберігають значення функцій у таблиці на основі критерію, що включає значення, а не (як у методі Sedgewick) на основі їх позицій. Наприклад, значення, що дорівнюють нульовому модулю, може бути збережене деяке значення . Простіше кажучи, Ніваш приписує Д. П. Вудруфу пропозицію зберігати випадкову вибірку раніше бачених значень, роблячи відповідний випадковий вибір на кожному кроці, щоб вибірка залишалася випадковою.

Ніваш описує алгоритм, який не використовує фіксований об'єм пам'яті, але для якого очікуваний об'єм пам'яті, що використовується (за умови, що функція введення є випадковим), є логарифмічним у довжині послідовності. Елемент зберігається в таблиці пам'яті за допомогою цієї методики, коли жоден пізній елемент не має меншого значення. Як показує Ніваш, елементи з цією технікою можна підтримувати, використовуючи структуру даних стека, і кожне наступне значення послідовності потрібно порівнювати лише у верхній частині стека. Алгоритм припиняється, коли знайдеться повторний елемент послідовності з найменшим значенням. Запуск того ж алгоритму з декількома стеками, використовуючи випадкові перестановки значень для впорядкування значень у кожному стеці, дозволяє компроміс у часовому просторі, аналогічний попереднім алгоритмам. Однак навіть версія цього алгоритму з одним стеком не є алгоритмом покажчика, завдяки порівнянням, необхідним для визначення, яке з двох значень є меншим.

Будь-який алгоритм пошуку циклу, який зберігає максимум значень із послідовності введення, повинен виконувати принаймні оцінки функцій.

Висновок

В даній курсовій роботі було розглянуто теорію графів, а саме графи та діаграфи, ступінь та відстань, основні структурні поняття та дерева. Також подано визначення алгоритмів пошуку циклів, їх комп’ютерне представлення, та власне самі алгоритми та їх порівняльний аналіз.