Курс Дискретної Математики 2023
Комп’ютерний проєкт

Студентам курсу Дискретної математики пропонується виконання комп'ютерного проекту, який перевіряє здобуті за семестр знання з дискретної математики та вміння використати їх на практиці (у програмуванні).

Загальні правила виконання комп’ютерних проектів:

- Мова написання - рекомендується написання на Python 3, допускається C, C++, Rust, Java. Інші мови не рекомендуються. Усі завдання і вимоги спроектовані так, що за пристойної реалізації потужності Python 3 буде цілком достатньо з запасом.
- Використання сторонніх бібліотек - не рекомендується, можливе лише за крайньої необхідності. Краще запитати асистента якщо плануєте використовувати.
- Всі алгоритми повинні бути максимально ефективними - будуть протестовані на великих даних. Якщо Ваша програма виконується більше умовних 5 - 10 хв, то завдання не зараховується. Більш детальні коментарі до часу роботи - в описі.
- Код повинен бути, чистим, зрозумілим, ефективним. Якщо використовуєте ООП - пишіть нормальний код, дотримуйтесь принципів ООП. Якщо використовуєте не Python 3 - вважатиметься, що ви знайомі з патернами програмування на цій мові.
- Плагіат недопустимий!!! Ви повинні розуміти принципи дискретної математики та алгоритмічного мислення. Якщо ви не можете пояснити власний код - завдання не зараховується, додатково буде висунута підозра на плагіат.
- Завдання можна (і треба) виконувати в команді. Розмір команди: 5 людей, розподіл можна знайти тут.

- Обов'язково повинен бути вказаний розподіл роботи в коамнді, однак це не відміняє того, що кожен учасник повинен розуміти кожну частину проекту. Відмазки у стилі “Це не я робив, не знаю”, “Ми не зробили цього бо нас було 2 людини в команді” - не приймаються.
- Велике прохання та рекомендація - здавайте завдання як репозитарій на GitHub. Хто не вміє/не хоче - здавайте архівом в CMS. Хоч це не обов’язково, та роботу здану на github легше перевіряти) Більшість з вас має бути знайомою з

GitHub з Лінукс клубу та курсу з програмування. Відповідні матеріали щодо цього тут: https://docs.google.com/document/d/1-MMKTdX_SflkiueoGb-FYWusTW6X-nDYAm9v7Jo5Kp4/edit

- Після виконання завдань - напишіть звіт (хто здає на GitHub - обов'язково в форматі readme.md). Решта здають у форматі .md чи .pdf файлу. У звіті детально опишіть усі написані алгоритми, принципи дискретної математики, які Ви використали, процес виконання і використання проекту. Опишіть розподіл роботи, враження від виконання та фідбек викладачам та асистентам.

Вам запропоновані 9 тем для Комп'ютерного проекту. Є можливість вибору власної теми - але вимоги відповідні. Вона повинна спиратися на вивчене за семестр з дискретної математики, містити необхідність написання ефективних алгоритмів. Усі пропозиції повинні узгоджуватись з асистентом чи викладачем.

Оцінювання проекту буде відбуватись у дві стадії:

1. Тестування коректності та адекватності коду, алгоритмів, звіту тощо. Тестування ефективності та працездатності рішень на різних наборах даних.

2. Усний захист - презентація вашого проєкту.

1. Пошук найкоротшого шляху між двома точками поверхні.

Багато задач в будівництві, логістиці, топології тощо потребують визначення найкоротшого шляху між двома точками Землі з урахуванням ландшафту. Вам потрібно розробити алгоритм, який знаходить найкоротший шлях між двома заданими точками прямокутної ділянки якого ландшафту. Ця ділянка задана прямокутною сіткою точок з заданими в них висотами.

Вхідні дані:

- Двовимірний масив цілих чисел M - відповідає сітці графу.
- Ціле число step - відповідає відстані між двома сусідніми точками в сітці графу.
- Точки A і B - пари індексів, які не виходять за межі масиву M.

Вихідні дані:

- Шлях найменшої довжини між точками A і B. Має представляти собою список пар індексів (список вершин шляху).

Зауваження та рекомендації:

1. В комірках масиву M зберігаються значення висоти даної точки.
2. Ми вважаємо, що шлях по поверхні - шлях по таблиці (графу) M, де сусідні вершини відрізняються у рівно одному індексі на значення 1.
3. Довжина ребра шляху - фізична відстань між точками на поверхні - квадратний корінь з суми квадратів різниці висот та числа step.
4. Розмір даних на яких буде тестуватись алгоритм може бути великим. Гарантується що всі висоти не перевищують 10 000, а розміри (лінійні) масиву M не перевищують 5000. Обмеження по часу виконання програми - 1 хв.
5. Вам буде запропонований певний набір даних різного розміру для власного тестування алгоритму.
6. Для розв’язку варто використовувати теорію графів й алгоритми на графах. Оцініть розмір вхідних даних та обмеження на час виконання для вибору найкращого алгоритму та структури даних.
7. Достатні матеріали для виконання завдання можете знайти на вікіпедії, хоч може доведеться пострибати по лінках) Shortest path problem - Wikipedia

2. Аналіз рекурентних рівнянь:

Рекурентні рівняння мають дуже широке коло застосування у теорії алгоритмів, інформатиці, різних розділах математики. Пропонується дослідження знаходження розв’язків рекурентних рівнянь. Вам потрібно якимось чином знайти певний розв’язок рекурентного рівняння за початковими даними.

Ви повинні виконати наступні завдання:

1. Знайдіть загальний розв’язок рекурентного рівняння аналітично.
2. Деякі надані Вам рівняння потребують розв’язання поліноміальних рівнянь, які важко розв'язати аналітично. Напишіть функцію, що буде знаходити їх відносно точні корені.
3. Напишіть програму, що буде знаходити перші n членів послідовності, використовуючи формулу з завдання 1.
4. Розробіть ефективний алгоритм (який працює за логарифмічний час), який буде точно знаходити n-тий член послідовності.
5. Порівняйте результат перших трьох пунктів з четвертим.

Зауваження та рекомендації:

1. Усі рівняння, запропоновані до розв’язання, будуть однорідними та лінійними, їх коефіцієнти будуть цілими числами.
2. У пункті 2 точність підрахунку коренів може не бути ідеальною, однак повинна бути досить високою.
3. У пункті 4 лінійного алгоритму буде не достатньо для обчислення - n будуть дуже великими, Вам потрібно реалізувати логарифмічний алгоритм, матеріали щодо цього можете знайти тут або ж тут.

3. Числа Каталана

Числа Каталана - одна з фундаментальних послідовностей в комбінаториці. Їх застосування проявляється в дуже великій кількості комбінаторних проблем. Задля їх дослідження пропонується виконати наступне завдання:

1. Знайдіть методом комп'ютерної симуляції кількість монотонних цілочисельних шляхів з точки (0, 0) до точки (n, n).
2. Знайдіть методом комп'ютерної симуляції кількість способів провести n діагоналей (які попарно не перетинаються) у правильному 2n-кутнику.
3. Знайдіть методом комп'ютерної симуляції кількість правильних дужкових послідовностей довжини 2n

Усі послідовності які ви отримаєте в цих симуляціях будуть однаковими - це числа Каталана. Спробуйте вивести рекурентну формулу для них та виконайте наступні завдання:

4. Знайдіть початкові умови для чисел Каталана та напишіть програму, яка ефективно знаходить перші n чисел Каталана, використовуючи рекурентну формулу
5. Спробуйте аналітично вивести формулу (коротку) для n-того числа Каталана з рекурентної формули (використайте степеневі ряди). Напишіть програму яка розраховує перші n чисел Каталана згідно з цією формулою.
6. Порівняйте результати підрахунків в задачах 1 - 5 та зробіть короткі висновки.

Зауваження та рекомендації:

1. Перші три пункти можете виконати довільним способом - хоч перебором. Симуляція повинна працювати для n, що не перевищують 10.
2. Для пункту 4 та 5 число n вже може бути достатньо великим - до 1000.
3. Для виведення аналітичної формули з рекурентної можна використати степеневі ряди та твірні функції.
4. Матеріали щодо чисел Каталана тут: [Catalan number - Wikipedia]

4. Знаходження максимального планарного підграфу.

Планарні графи – дуже корисний концепт, який використовується у дизайні процесорів та плануванні міст.

Ваша задача – відтворити алгоритм знаходження планарного підграфу – тобто, коли вам дано непланарний граф, ви повинні повернути планарний граф із максимальною кількістю ребер.

1. Читання графу з файлу - функція яка вміє прочитати файл і записати дані в відповідну структуру. Формат файлу - csv, перша колонка містить перші вершини кожного ребра, друга - другі вершини (порядок вершин не грає ролі в випадку неорієнтованого графу, і є важливим коли граф орієнтований).
2. Запис графу в файл - записує граф у файл (структура файлу описана вище)
3. Алгоритм знаходження максимального планарного підграфу, описаний за цим посиланням

Зауваження та рекомендації:

1. Можна реалізувати інакший алгоритм, але його доведеться шукати вам самим
2. Тести - корисна річ для зневадження Вашого коду.
3. Можна реалізувати рекурсивне застосування алгоритму, аби зробити полділ графу на декілька планарних.

5. Аналіз зв'язності графів

Звязність графу - одна з ключових його властивостей. У цьому завданні Ви повинні написати бібліотеку для аналізу зв'язності графів. Бібліотека повинна містити наступні можливості:

1. Читання графу з файлу - функція яка вміє прочитати файл і записати дані в відповідну структуру. Формат файлу - csv, перша колонка містить перші вершини кожного ребра, друга - другі вершини (порядок вершин не грає ролі в випадку неорієнтованого графу, і є важливим коли граф орієнтований).
2. Запис графу в файл - записує граф у файл (структура файлу описана вище)
3. Пошук компонент зв’язності - повинен знаходити усі компоненти зв'язності неорієнтованого графу і повертати список компонент зв'язності (вважаємо що компонента ідентифікується вершиною найменшого номеру, який їй належить).
4. Аналогічна функція пошуку компонент сильної зв'язності для орієнтованого графу.
5. Пошук точок сполучення - знаходить усі точки сполучення неорієнтованого графу (повертає список вершин).
6. Пошук мостів - знаходить усі мости неорієнтованого графу (повертає список ребер, кожне ребро - впорядкована пара точок).

Зауваження та рекомендації:

1. Для простоти вважаємо що наш граф не містить петель і кратних ребер.

2. Бібліотека буде тестуватись на даних великого розміру, тому реалізація функцій повинна буде якнайефективнішою.

3. Ви можете використовувати будь-яке внутрішнє представлення графу, однак будьте вкрай уважні з форматом файлів - якщо ввід чи вивід не буде працювати або працює некоректно - завдання вважається невиконаним, оскільки немає можливості його протестувати.

4. Для виконання всіх завдань достатньо знати та вміти використовувати алгоритми пошуку в глибину та в ширину. Матеріали щодо реалізації потрібних алгоритмів можна знайти тут:
https://uk.wikipedia.org/wiki/%D0%97%D0%B2%27%D1%8F%D0%B7%D0%BD%D0%B8%D0%B9_%D0%B3%D1%80%D0%B0%D1%84
Connectivity (graph theory) - Wikipedia

6. Задача комівояжера

Задача комівояжера (travelling salesman problem) – відома задача пошуку найвигіднішого (найкоротшого) шляху у графі, що проходить через усі вершини хоча б раз та повертається до початкової позиції. Ваша задача полягатиме у тому, щоб реалізувати пошук такого шляху для неорієнтованого графу точним алгоритмом та «жадібним» приблизним алгоритмом.

Завдання:

1. Реалізувати зчитування графу з файлу – функція повинна приймати шлях до csv файлу, що міститиме у першій колонці першу вершину ребра, у другій – другу вершину, у третій – вагу ребра.
2. Реалізувати функції (дві окремі), що знаходитимуть розв’язок (у форматі списку вершин, що починається і закінчується на 1) задачі комівояжера для даного графу з початком у вершині 1 (таким номером вона буде передана у csv) двома алгоритмами. Якщо шляху немає (потрібна перевірка на це), функція повинна повернути повідомлення про це.

Зауваження та рекомендації:

1. Корисно ознайомитись з точним алгоритмом динамічного програмування Хелда-Беллмана-Карпа. Тут інформація про «жадібний» алгоритм найближчого сусіда.
2. Для тестування точного алгоритму використовуватимуться графи розміром до 12 вершин, тому звичайного перебору варіантів за n! не вистачить.
3. Для тестування жадібного алгоритму буде використано тестові кейси з <= 1000 вершинами.

7. Графи etc

У цьому завданні пропонується реалізувати декілька функцій, які забезпечують виконання різноманітних операцій над графами, наприклад розфарбування, пошук ейлерового та гамільтонового циклів, перевірка на дводольність, тощо. Дозволяється реалізувати 4 з 5 функцій, описаних нижче (ввід/вивід обов'язкові).

Вам потрібно написати наступні функції:

1. Читання графу з файлу - функція яка вміє прочитати файл і записати дані в відповідну структуру. Формат файлу - csv, перша колонка містить перші вершини кожного ребра, друга - другі вершини (порядок вершин не грає ролі в випадку неорієнтованого графу, і є важливим коли граф орієнтований).
2. Функція приймає граф та повертає Гамільтоновий цикл (чи повідомлення про його відсутність) у формі списку вершин.
3. Функція приймає граф та повертає Ейлерів цикл (чи повідомлення про його відсутність) у формі списку вершин.
4. Функція приймає граф та перевіряє чи є він дводольним. Повертає булеве значення.
5. Функція приймає два графи і перевіряє чи є вони ізоморфними. Повертає булеве значення.
6. Функція приймає зв'язний граф і повертає його розфарбування у три кольори чи повідомлення про неможливість такого розфарбування. Повинна повертати список пар вершина - колір.

Зауваження та рекомендації:

1. Пункт 2 та 6 можна розв'язати за допомогою пошуку з поверненням (backtracking), потрібні алгоритми були описані в лекціях.

2. Пункт 3 вартує виконувати з використанням алгоритму Флері, однак не обмежується використання інших алгоритмів.

3. У пункті 5 кількість вершин кожного графу не перевищує 10. Будьте уважні з складністю Вашого алгоритму.

4. Додаткові матеріали щодо алгоритмів на графах можна знайти тут: https://en.wikipedia.org/wiki/Category:Graph_algorithms

8. Задача 2-SAT

У неорієнтованому графі кожну вершину пофарбували у червоний, зелений чи синій колір. Розфарбування графа є ***вдалим,*** якщо будь-яке ребро з'єднує вершини різного кольору. Вам потрібно змінити колір кожної вершини так, щоб розфарбування графу стало ***вдалим.*** Обов'язково потрібно змінити колір кожної вершини графу, також колір не можна змінювати на той самий, що був.

Завдання:

1. Читання графу з файлу - функція яка вміє прочитати файл і записати дані в відповідну структуру. Формат файлу - csv, перша колонка містить перші вершини кожного ребра, друга - другі вершини, третя - колір першої вершини, четверта - колір другої вершини.

2. Напишіть функцію, яка б приймала граф з розфарбованими вершинами і повертала б список пар (вершина, колів) який відповідає ***вдалому*** розфарбуванню, за умов початкового розфарбування. Якщо такого розфарбування не існує, поверніть повідомлення про це.

Зауваження та рекомендації:

1. Ефективне виконання цього завдання потребує використання 2-SAT Problem - відомого розв'язку до задачі розподілу булевих значень булевим змінним так, щоб вони задовольняли якісь умови. Ключовими моментами є побудова конюктивної форми та перехід до графу імплікацій. Основні матеріали можна знайти тут: 2-satisfiability - Wikipedia
2. Завдання буде тестуватись на даних великого розміру, тому алгоритм перебору не спрацює. Якщо коректно реалізувати 2-SAT, то часова складність не буде проблемою.
3. Гарантується, що кількість вершин в графі не перевищує 1000, а кількість ребер в графі - 20 000.

9. PageRank algorithm

PageRank - алгоритм оцінки важливості веб-сторінок. Для кожної сторінки обчислює дійсне число, чим більше число - тим “важливіша” сторінка.

Основне припущення полягає в тому, що більш важливі веб-сайти, швидше за все, отримають більше посилань з інших веб-сайтів.

Мережу можна представити як орієнтований граф, де вузли представляють веб-сторінки, а ребра - зв’язки між ними.,

Сам алгоритм приписує кожній із веб-сторінок її PageRank, чим більший - тим важливішою є сторінка.

Завдання:

1. На вхід вашого алгоритму подаватиметься граф (як саме його представити - вибирати вам, важливо лише, що кожна вершина представляє один сайт), на вихід маєте повернути значення PageRank для кожного із сайтів.

Корисні посилання:

1. https://towardsdatascience.com/pagerank-algorithm-fully-explained-dc794184b4af
2. https://towardsdatascience.com/pagerank-3c568a7d2332