Алгоритми

Лекція 4

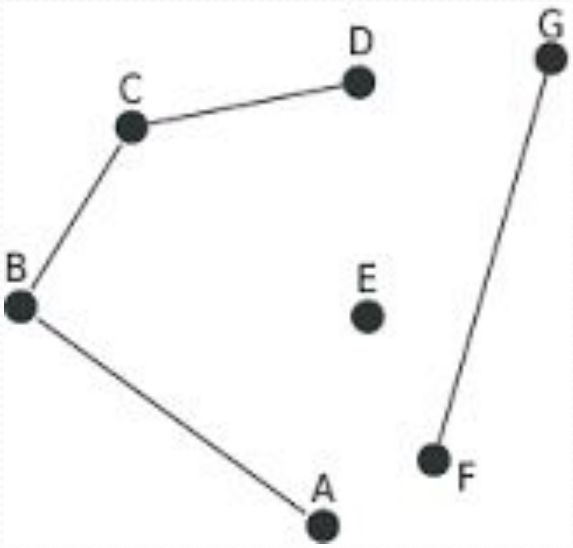
Алгоритми на графах 1

План лекції

- Загальні поняття/теорія
- Сім мостів Кеніґсберґа
- Способи представлення графів в задачах
- Пошук в глибину
- Пошук в ширину
- Топологічне сортування
- Задача

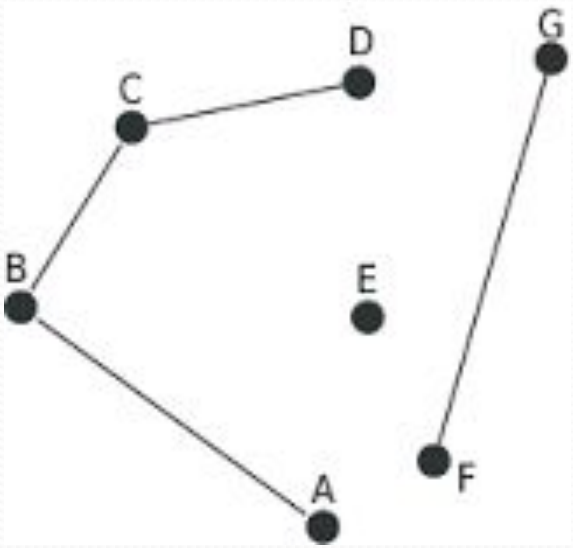
Загальні поняття

- **Граф** - це абстрактне представлення множини об'єктів і зв'язків між ними.
- Об’єкти називають **вершинами** графа.
- Зв’язки називають **ребрами** графа.



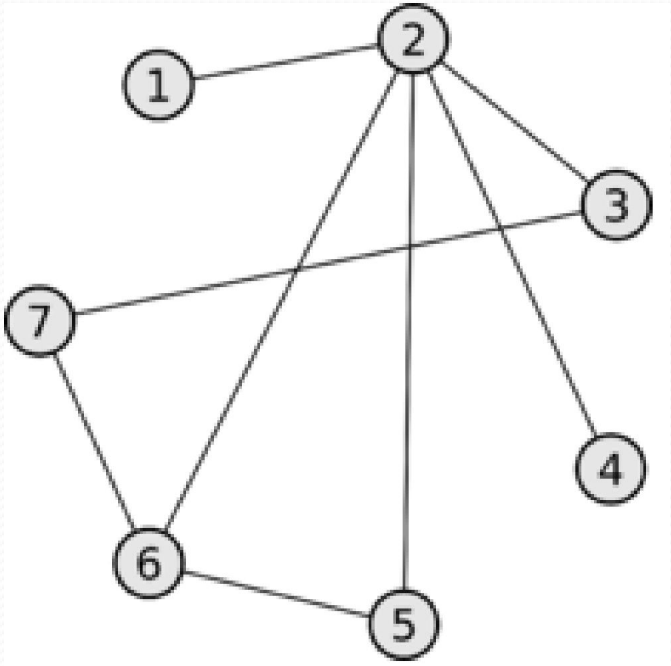
Загальні поняття

- Вершини графа які з’єднані одним ребром називають **суміжними**.
- У даному випадку вершини B і C – суміжні, у той час як B і D не є суміжними.

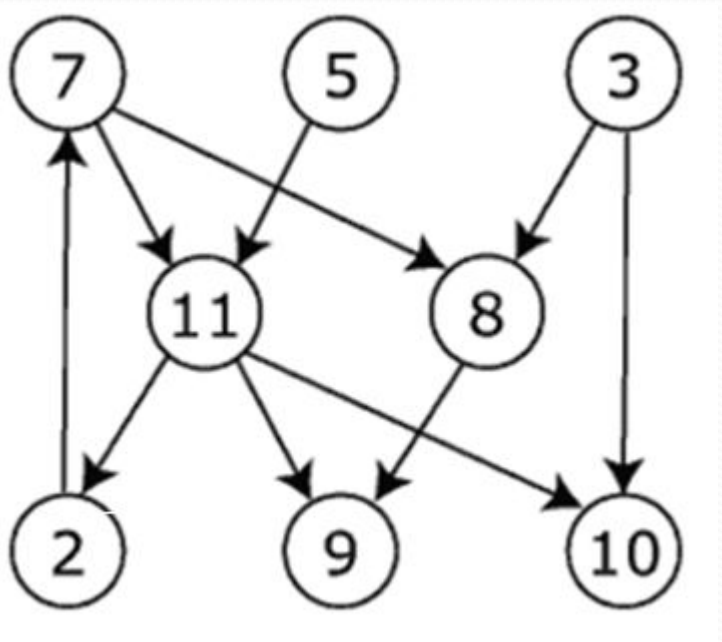


Загальні поняття

Неорієнтовний граф (оргграф)

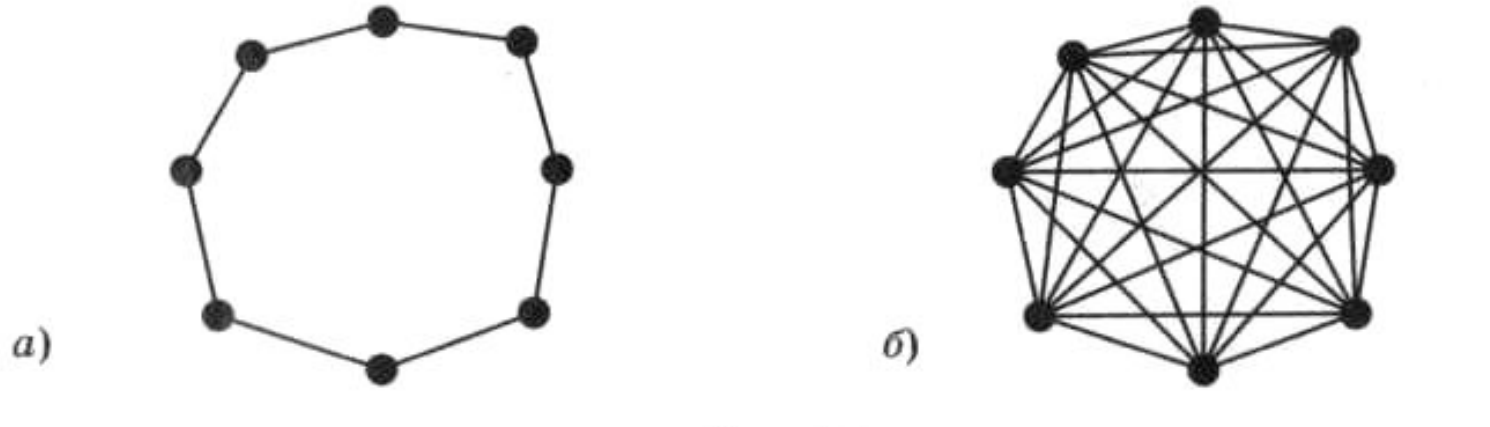


Орієнтовний граф



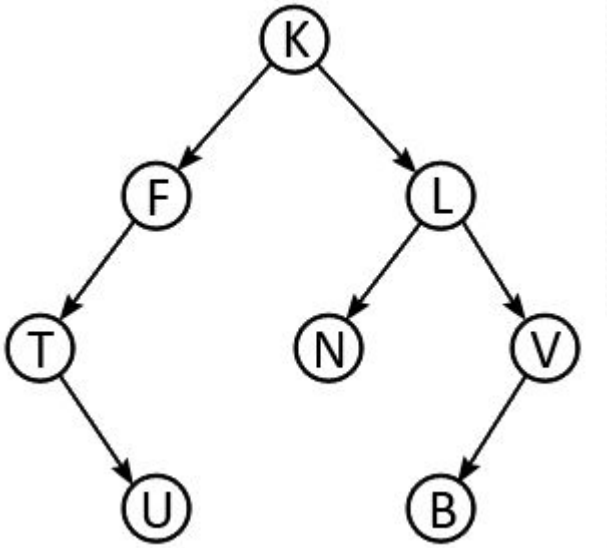
Загальні поняття

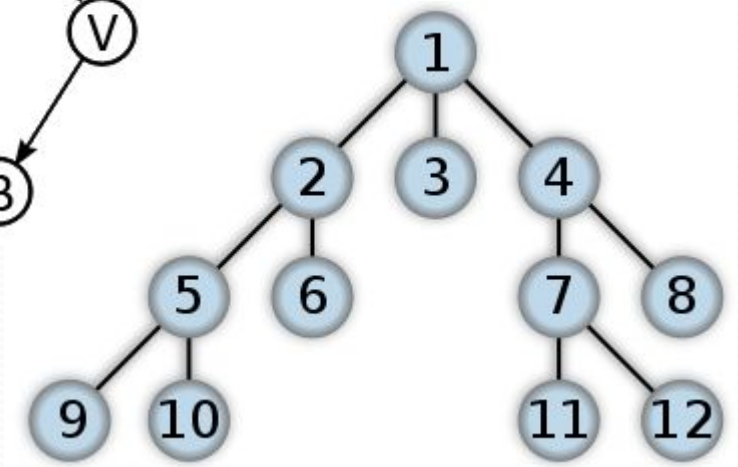
- Граф називається **зв'язним**, якщо між будь-якою парою вершин цього графа існує шлях. (а)
- Не варто плутати його із **повним графом** у якого будь-яка пара вершин з'єднана ребром. (б)

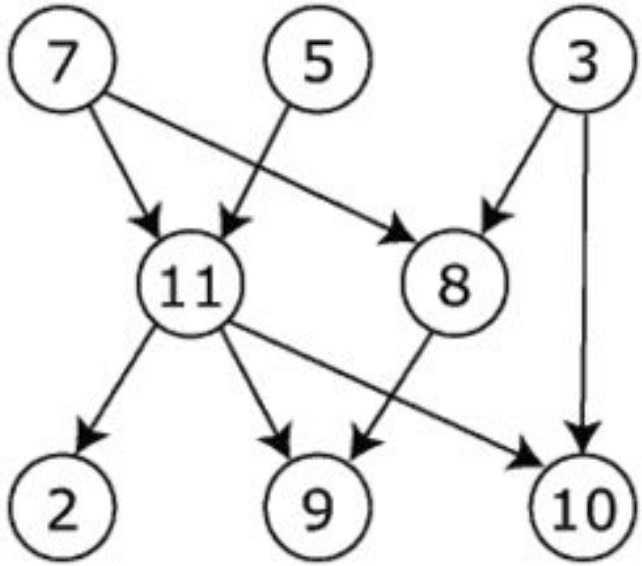


Загальні поняття

- **Дерево** – це зв'язний граф без циклів.

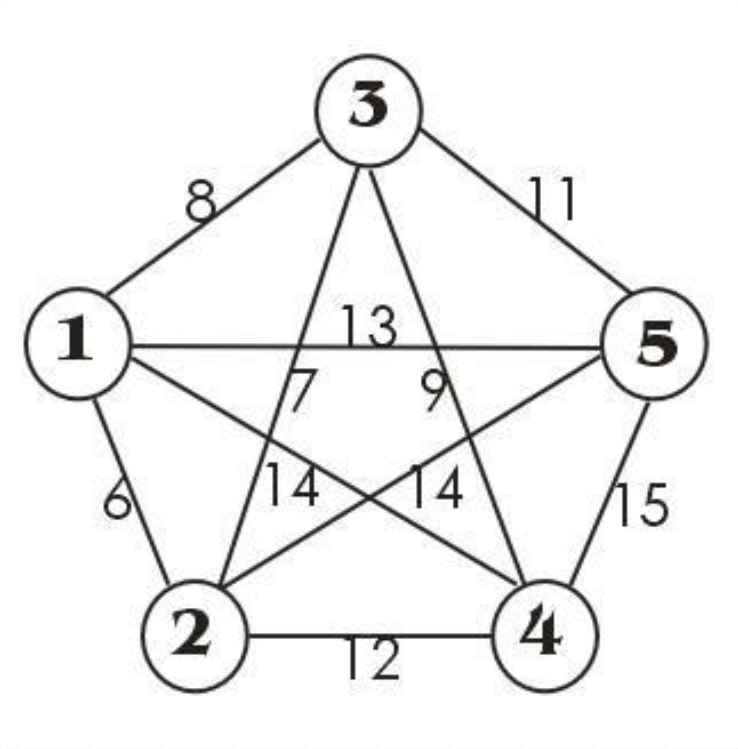


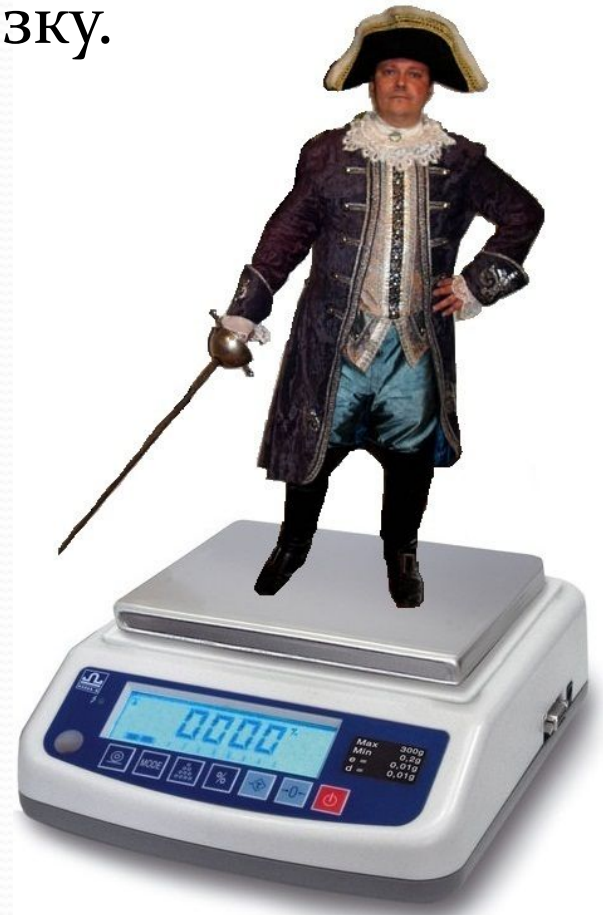




Загальні поняття

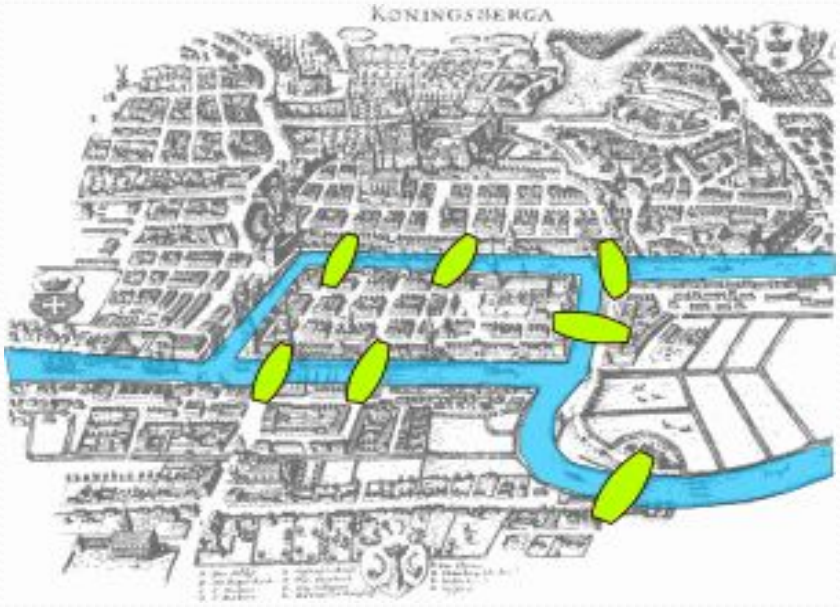
- **Зважений граф** – граф який має певну вагу ребер. Тобто враховується не лише наявність зв’язку між двома вершинами а й вага цього зв’язку.





Сім мостів Кеніґсберґа

- Необхідно було знайти такий маршрут через місто, щоб пройти всі сім мостів і кожним мостом пройти рівно один раз. На острів не можна було потрапити інакше як через міст, і кожен з мостів мав бути пройденим за один раз (тобто не можна було пройти на середину мосту і повернутися назад, а потім з іншого берега пройти другу половину).

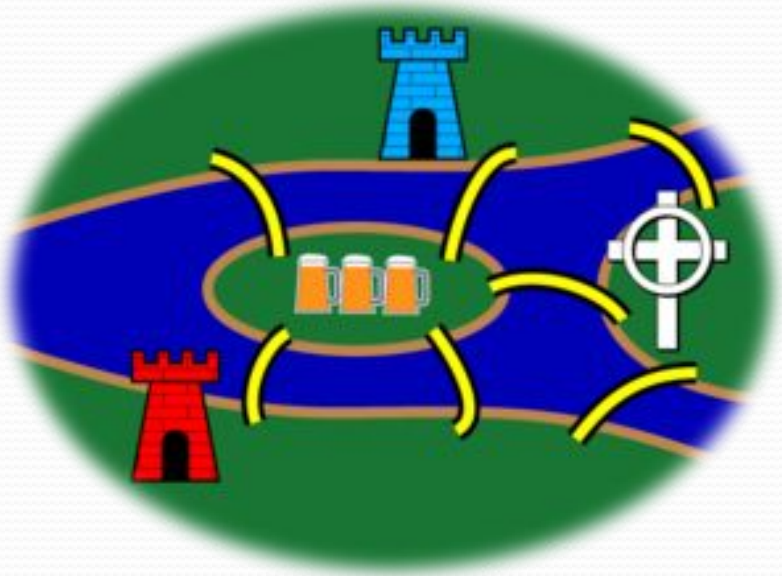


- Леонард Ейлер у 1735 р. довів що розв'язку **не** існує. Це положило початок створення теорії графів.

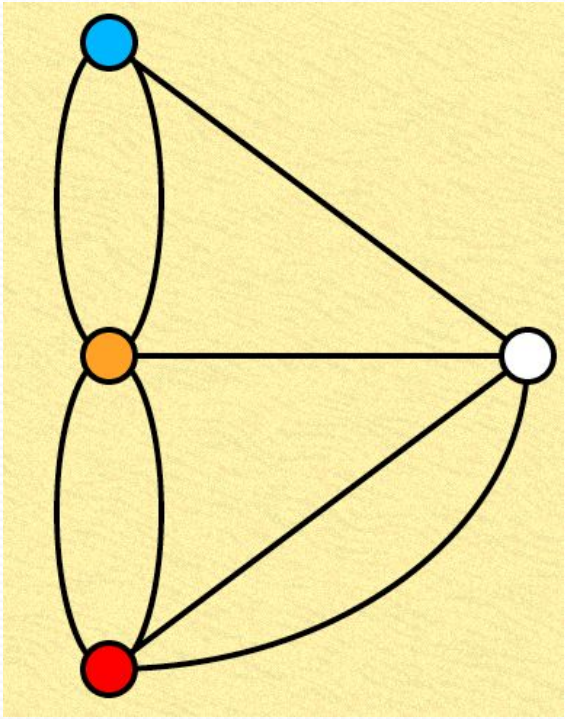
- Ейлер довів що для вирішення цієї задачі повинно бути тільки 2 вершини непарного степеня (у одній з них почати шлях а в іншій закінчити). Ми ж спостерігаємо 4 чершини непарного степеня, що говорить про неможливість проходу даним способом
- Детальніше: https://uk.wikipedia.org/wiki/Сім_мостів_Кеніґсберґа

Варіація задачі

- Північний берег річки зайнятий замком Синього Принца, південний — Червоного Принца. Східний берег це церква; і маленький острів це корчма.
- Серед місцевих жителів було звичаєм, після кількох годин в корчмі, намагатися *обійти мости*, багато з них повертались в корчму, щоб освіжитися для успішного завершення задачі.



- Синій Принц проаналізував міські мости з позиції теорії графів і дійшов висновку, що обійти їх всіх по одному разу неможливо. Він придумав хитрий план будівництва восьмого мосту так, щоб він міг звечора почати від свого замку, обійти всі мости і закінчити в корчмі, де похвалився б своєю перемогою. Звісно, він хотів, щоб Червоний Принц не міг повторити його подвиг від свого замку. *Де Синій Принц має побудувати восьмий міст?*



Способи представлення графів

Існує два способи представлення графа:

- у вигляді списку суміжності;
- у вигляді матриці суміжності;

Обидва способи підходять для представлення орієнтованих і неорієнтованих графів.

Матриця суміжності

- Цей спосіб є зручним для представлення щільних графів, в яких кількість ребер E приблизно дорівнює кількості вершин у квадраті V^2.
- Ми заповнимо матрицю розміром V на V таким чином:

  A[i][j] = 1 (Якщо існує ребро з i в j)

  A[i][j] = 0 (Інакше)
- Можна швидко перевірити чи є ребро між вершинами v і u, просто подивившись в матрицю за адресою A[v][u].

Матриця суміжності

- Даний спосіб підходить для орієнтованих і неорієнтованих графів. Для неорієнтованих графів матриця А є симетричною.

	1	2	3	4	5
1	0	0	1	1	0
2	0	0	1	0	0
3	1	1	0	0	1
4	1	0	0	0	1
5	0	0	1	1	0

неорієнтований граф

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	1	0	0
3	1	0	0	0	0
4	0	0	1	0	0
5	0	0	0	1	0

орієнтований граф

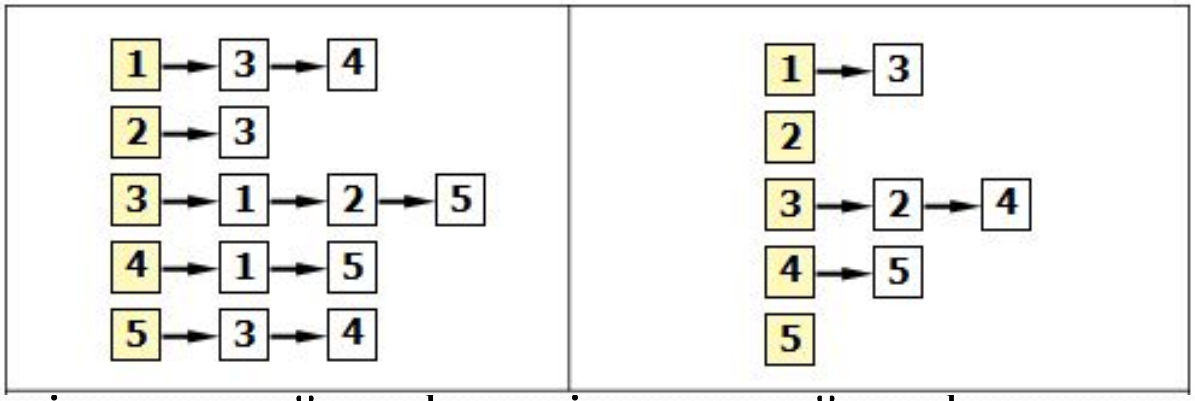
- Недоліком цього способу є великі затрати по пам’яті (матриця вимагає V^2 памяті).

Список суміжності

- Підходить для розріджених графів, тобто графів у яких кількість ребер набагато менше ніж кількість вершин в квадраті $E << V^2$.
- В даному представленні використовується масив adj, що містить V списків. У кожному списку adj [v] містяться всі вершини u, так що між v і u є ребро.
- Іншими словами у списку adj [v] містяться всі вершини в які ми можемо потрапити із v.

Список суміжності

- Пам'ять необхідна для списку суміжності дорівнює O (E + V) що є кращим показником ніж матриця суміжності для розріджених графів.

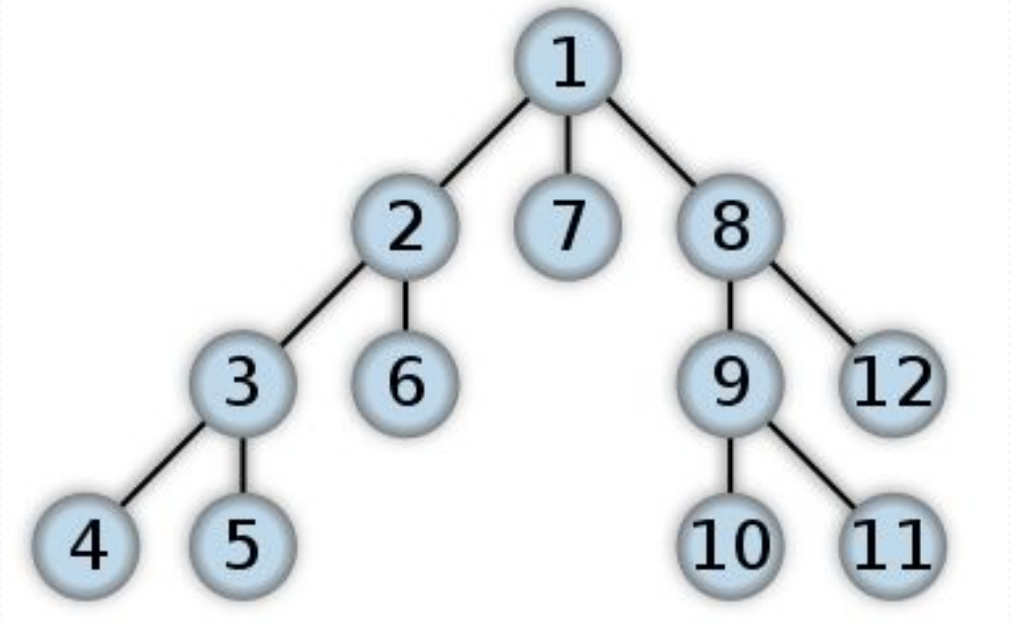


неорієнтований граф орієнтований граф

- Головний недолік цього способу представлення в тому, що немає швидкого способу перевірити чи існує ребро (u, v).

Пошук в глибину

- *Depth-first search*, ***DFS*** - один з методів обходу графа, стратегія якого полягає в тому, щоб йти «вглиб» графа, наскільки це можливо. Реалізується рекурсивним перебором всіх суміжний вершин із поточною.



Алгоритм пошуку в глибину

- Проходимо по всіх вершинах, якщо знаходимо вершину *v* зафарбовану в білий колір запускаємо для неї DFS(*v*).
- Процедура DFS(*v*):
  1. Зафарбовуємо *v* в сірий колір.
  2. Для кожної вершини *w*, суміжної з *v* і зафарбованої в білий колір запускаємо DFS(*w*).
  3. Зафарбовуємо *v* в чорний колір.
- Часто використовують двоколірні мітки - без сірого, тоді на 1-му кроці фарбують відразу в чорний колір.

Алгоритм пошуку в глибину з часовими мітками

- Для кожної з вершин встановимо два числа: час входу *entry[v]* та час виходу з вершини *leave[u]*. Тоді процедура DFS буде мати такий вигляд:
  1. Збільшуємо «поточний час» t на 1. *entry[v] = t*.
  2. Зафарбовуємо *v* в сірий колір.
  3. Для кожної вершини *w*, суміжної з *v* і зафарбованої в білий колір запускаємо DFS(*w*).
  4. Зафарбовуємо *v* в чорний колір.
  5. Збільшуємо «поточний час» t на 1. *leave[v] = t*.
- Вважаємо, що граф орієнтований, тоді очевидно, для будь-якої вершини *entry[v] < leave[v]*.

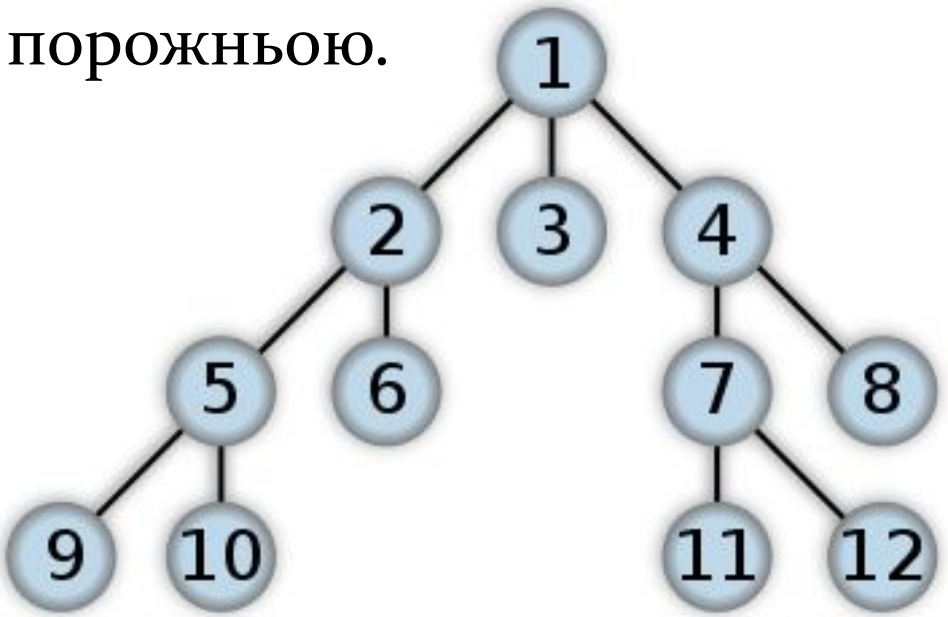
Використовується для вирішення задач:

- Пошук будь-якого шляху в графі.
- Перевірка, чи є одна вершина дерева предком іншої.
- Задача LCA (найменший спільний предок).
- Топологічне сортування.
- Перевірка графа на ациклічність і знаходження циклу.
- Пошук компонент сильної зв'язності.
- Пошук мостів.

- Алгоритм працює за O (n + m), де n - число вершин, m - число ребер.

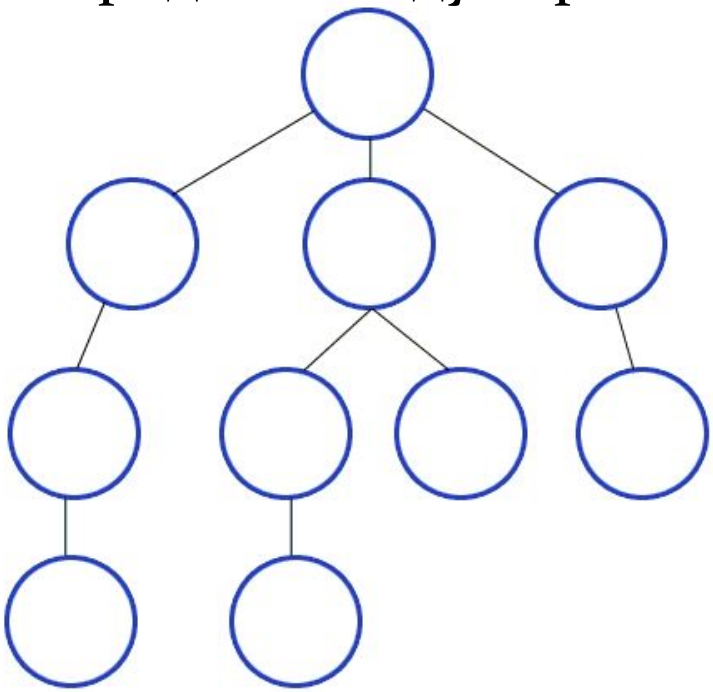
Пошук в ширину

- *Breadth-first search*, ***BFS*** - один з методів обходу графа, працює шляхом пошарового послідовного прошодження всіх вершин. Для реалізації алгоритму використовується черга, в яку додаються всі суміжні з поточною вершини, а потім нова поточна вершина зарається з черги, поки вона не стане порожньою.

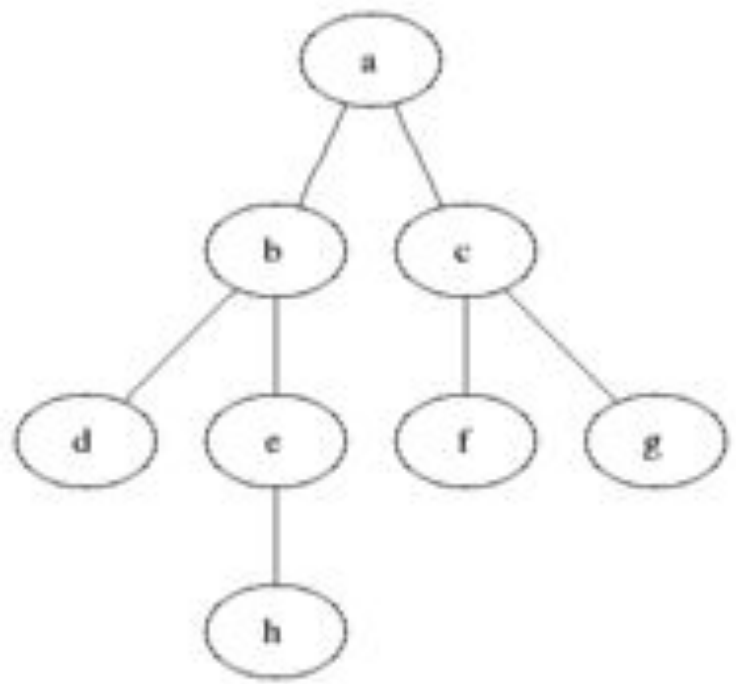


Пошук в ширину (демо)

Порядок обходу вершин



Процес обходу графа



Білий - вершина, ще не розглянута.
Сірий - вершина, додана в чергу.
Чорний - вершина, витягнута з черги.

Алгоритм пошуку в ширину

1. Помістити вузол, з якого починається пошук, в порожню чергу.
2. Витягти з початку черги вузол *u* і позначити його як відвіданий *used[u] = true*. Якщо вузол u є цільовим вузлом, то завершити пошук з результатом «успіх». В іншому випадку, в кінець черги додаються всі суміжні з *u* вершини, які ще не відвідані.
3. Повернутися до п. 2.
4. Якщо черга порожня, то всі вузли зв'язкового графа були переглянуті, отже, цільової вузол недосяжний з початкового; пошук завершується з результатом «невдача».

Використовується для вирішення задач:

- Пошук найкоротшого шляху в незваженому графі.
- Пошук компонент зв'язності в графі за O (n + m).
- Знаходження рішення будь-якої задачі (гри) з найменшим числом ходів (стан гри – вершини а переходи між станами - ребра).
- Пошук циклів в графі.
- Задачі мінімізації (перестановок, відстаней і т.д.).

- Алгоритм працює за O (n + m), де n - число вершин, m - число ребер.

Топологічне сортування

- Це таке впорядкування вершин направленого графа, що для будь якого ребра (*u* -> *v*) вершина *u* має бути перед *v*.
- Сортування працює тільки для орієнтованого ациклічного графа.
- В результаті сортування графа виходить послідовність, така що всі ребра будуть направлені зліва направо.

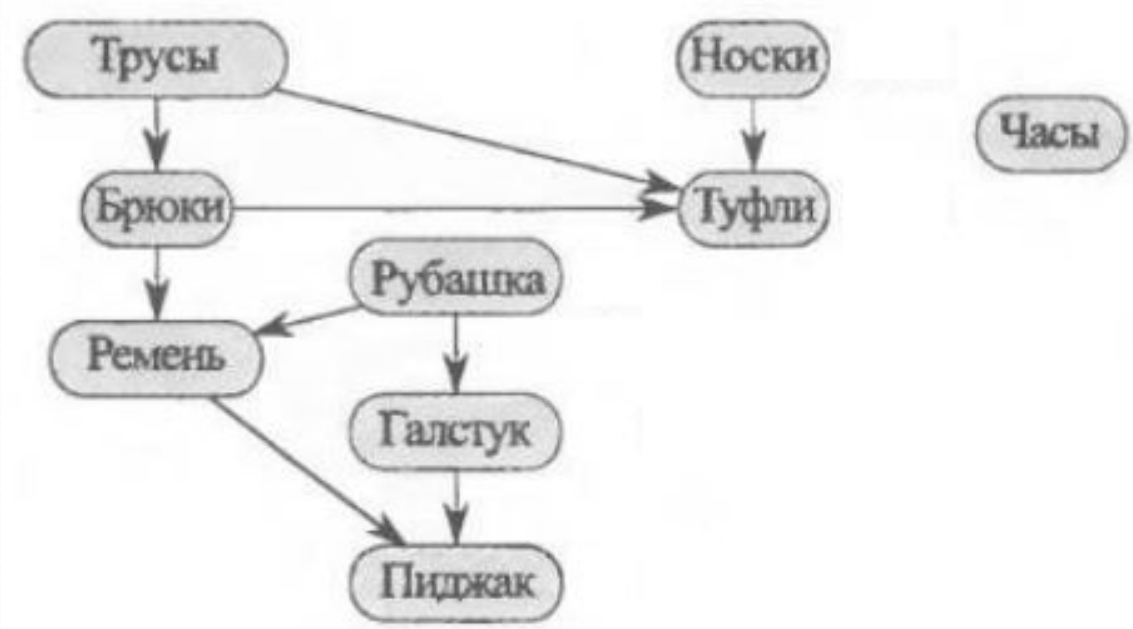
Реалізація

- Топологічне сортування реалізується за допомогою пошуку в глибину, все що необхідно додати це запис поточної вершини після виходу із рекурсії.
- Якщо використати пошук в глибину с часовими мітками то можна побачити що результуюча послідовність впорядкована по спаданню часу виходу із вершини.

Задача 1

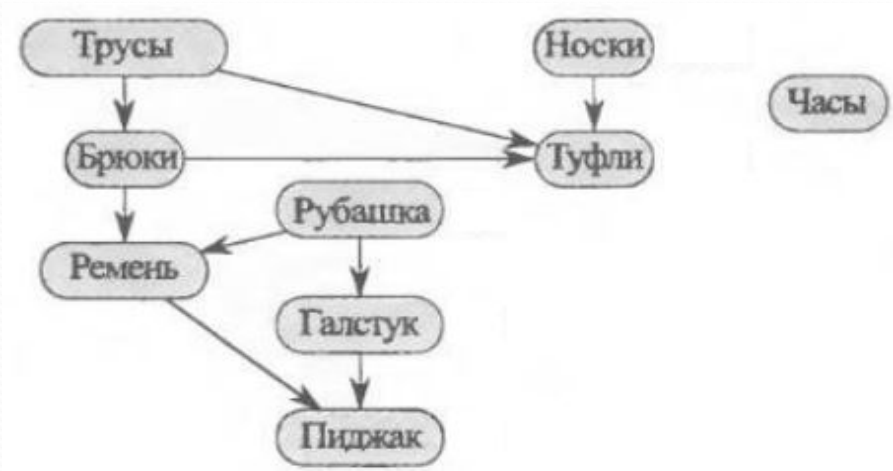
Є одяг який треба одягти, і є строго визначений порядок для деяких пар одягу. Наприклад шкарпетки мі не можемо одягнути після того як взули черевики.

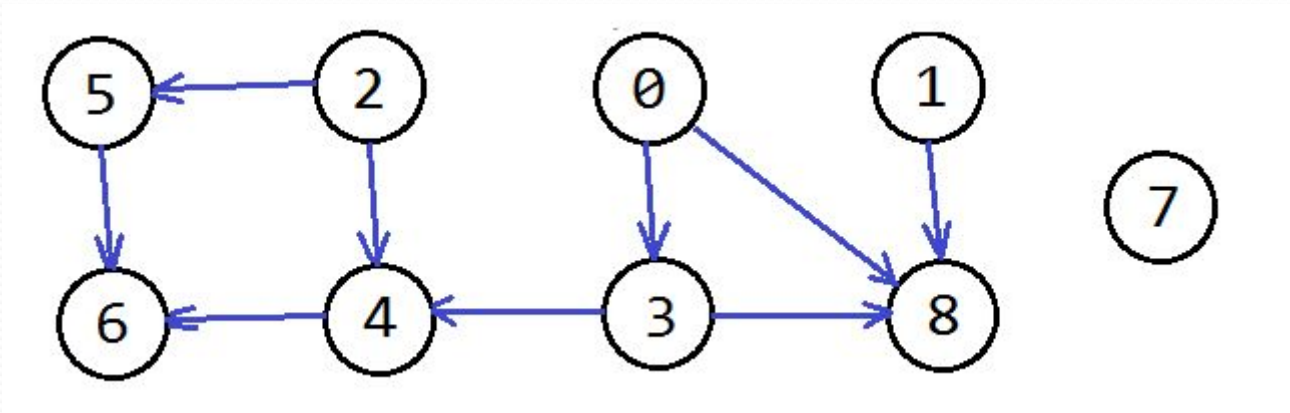
Необхідно побудувати чіткий порядок одягання із врахуванням таких залежностей:



Задача легко розв’язується за допомогою потологічного сортування даного графа. Пронумеруємо вершини:

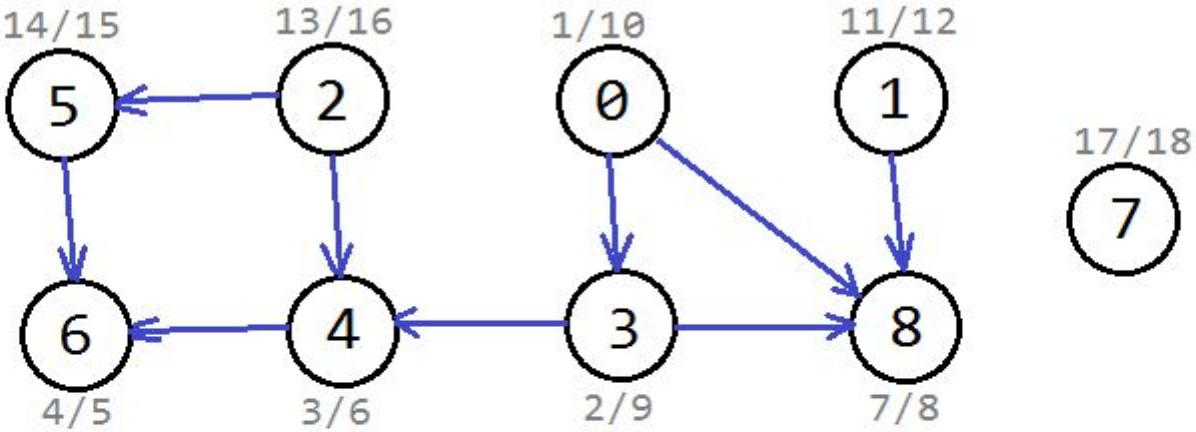
0. Нижня білизна
1. Шкарпетки
2. Сорочка
3. Штани
4. Ремінь
5. Галстук
6. Піджак
7. Годинник
8. Взуття

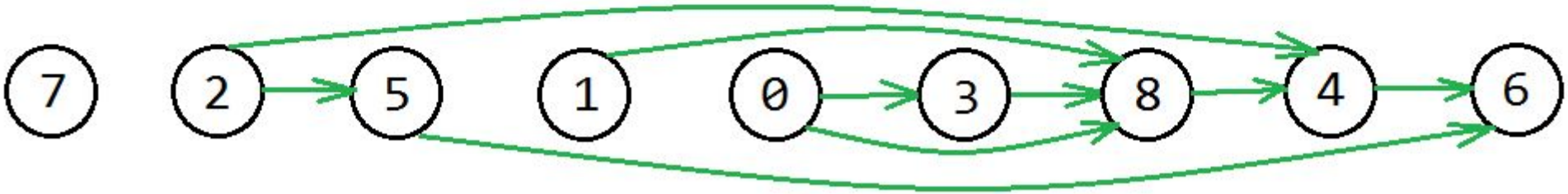




Аналіз розв’язку

Номерами біля вершин позначено час входу/виходу із вершини





Задача 2

В останніх маркетингових дослідженнях було виявлено, що цікаві рекламні ролики в соц. мережах розходяться дуже швидко між усіма друзями, тому для поширення вірусних роликів в конкретному колі спілкування досить показати цей ролик в стрічці новин соц. мережі одній людині. Такий вид реклами сильно зменшує витрати на її поширення. У маркетологів є дані хто з ким "дружить" в соц. мережі і тепер виникає питання порахувати мінімальну кількість людей, яким необхідно додати вірусний рекламний ролик в новини, щоб він розійшовся по всій мережі.

Аналіз розв’язку

Задача може бути розв’язана пошуком в глибину або в ширину, я вибираю пошук в ширину. Суть в тому що ми повинні визначити кількість зв’язних груп у графі, тобто якщо в кожній зв’язній групі одна людина отримає вірусний ролик, то він поширеться на всю мережу(граф).

На вході маємо матрицю знайомств, а на виході маємо дати кількість людей для поширення ролика на всю мережу.

Аналіз розв’язку

Маємо матрицю знайомств в якій m[i][j]=1, якщо і друг j.

У першій матриці юзер1 друг з юзером2
а також юзер3 друг з юзером4, тобто маємо
2 незалежні групи людей, що відразу дає
нам відповідь 2.

```
0 1 0 0
1 0 0 0
0 0 0 1
0 0 1 0
Відповідь 2
```

У другій матриці юзер1 друг з юзер2, юзер2
друг з юзер3, і так далі до юзера 5, отже всі
юзери мережі дружать тому відповідь 1.

```
0 1 0 0 0
1 0 1 0 0
0 1 0 1 0
0 0 1 0 1
0 0 0 1 0
Відповідь 1
```

Домашнє завдання:

Знайти найбільший цикл в графі (найбільший цикл той, що містить максимальну кількість вершин) і вивести його вершини. Якщо таких циклів декілька вивести всі можливі варіанти.