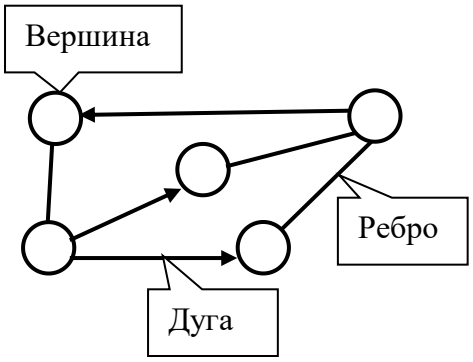
Лабораторна робота №7-8

Графи. Дерева. Алгоритми пошуку в глибину та в ширину

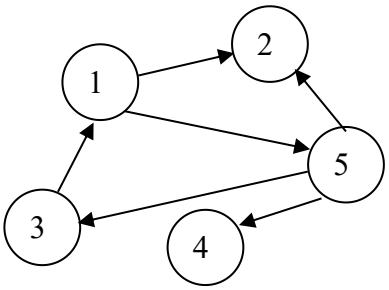
Мета роботи: Освоїти та закріпити прийоми роботи з даними різного типу, організованими у вигляді дерев та їх окремого випадку – бінарних дерев. Здобути практичні навички роботи з графами.

Методичні вказівки



Граф (**graph**) – структура даних, що складається з множини кіл (точок) і множини ліній, які їх з'єднують. Кола (точки) називаються ***вершинами графа*** (**nodes**), лінії зі стрілками – ***дугами*** **(arcs)**, без стрілок – ***ребрами*** (**edges**). Тобто. ***граф*** – це пара **G**=(**V**,**E**), где **V** - множина вершин, а **E** - множина пар вершин $\mathbf{e}_i=(\mathbf{v}_{i1}, \mathbf{v}_{i2})$, $\mathbf{v}_{ij}$ належить **V**. Вершини графа можна використовувати для представлення об'єктів, а ребра та дуги - для відносин між об'єктами.

Графи зазвичай зображуються у вигляді геометричних фігур. У класичному графі відсутні **петлі**, тобто ребра, що з’єднують вершину саму із собою.



Граф у якому напрямок ліній принциповий (лінії є дугами) називається орієнтованим (орграф). На відміну від ребер, дуги з’єднують дві нерівноправні вершини: одна з них називається початком дуги (дуга з неї виходить), друга - кінцем дуги (дуга до неї входить). Приклад:**G**=(**V**,**E**): **V={1, 2, 3, 4, 5}; E={(1,2), (1,5), (3,1), (5,2), (5,3), (5,4)}.**

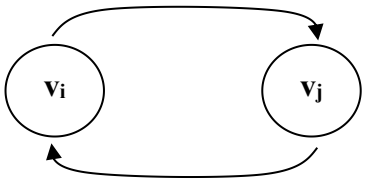
Граф, у якому напрямок ліній не виділяється (всі лінії є ребрами), називається **неорієнтованим** (дві вершини рівноправні, немає жодної різниці між "початком" та "кінцем" ребра).

Найчастіше розглядають графи, де всі ребра мають один тип: або орієнтовані, або неорієнтовані.

Дві вершини **v** та **u** називаються ***суміжними***, якщо вони з'єднані ребром (дугою) **e**: **e=(v,u)**. Суміжні вершини називаються граничними вершинами відповідного ребра (дуги), але це ребро (дуга) - ***інцидентним*** відповідним вершинам. Будь-якому ребру інцидентно рівно дві вершини, а вершині може бути інцидентно довільна кількість ребер.

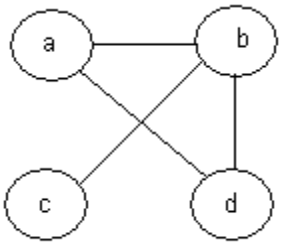
Два ребра називаються ***суміжними***, якщо вони інцидентні одній вершині.

Ступенем вершини в неорієнтованому графі називається число інцидентних їй ребер. Для орієнтованого графа розрізняють ***вихідний ступінь***, що визначається як число ребер, що виходять з нього, і ***вхідний ступінь***, що визначається як число ребер що входять до неї. Сума вихідного та вхідного ступенів називається ***ступенем вершини***. ***Ізольована вершина*** – це вершина зі ступенем 0 (***нуль-граф***).



Мультиграф - з однієї вершини в іншу можна перейти у різний спосіб (граф з кратними ребрами).

Шляхом називається послідовність вершин v_1, v_2, …, v_n, для якої існують ребра (або дуги в орграфі) $v_1 \rightarrow v_2$, $v_2 \rightarrow v_3$, ..., $v_{n-1} \rightarrow v_n$. Цей шлях починається у вершині v_1 і, проходячи через вершини v_2, v_3, ..., v_{n-1}, та закінчується у вершині v_n. ***Довжина шляху*** – це кількість дуг (ребер), що складають шлях. У даному разі довжина шляху дорівнює n – 1. Шлях називається ***простим***, якщо всі вершини у ньому, крім, можливо, першої і останньої, різні. Для неорієнтованого графа, який показаний на рисунку, шлях буде *adbc* и *abc*.



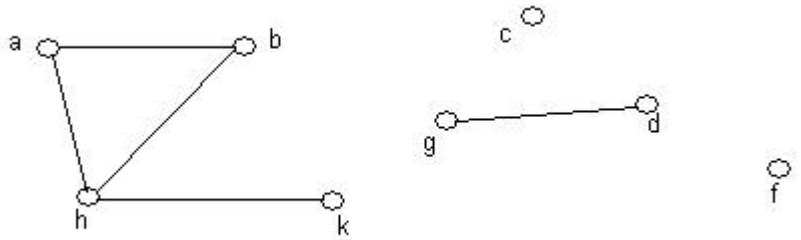
Замкнений шлях без ребер, що повторюються, називається ***циклом*** (або ***контуром*** в орграфі); без повторюваних вершин (крім першої та останньої) - ***простим циклом***.

Повним називається граф, у якому проведено всі можливі ребра. Для графа, що має ***n*** вершин, таких ребер буде ***n(n-1)/2***.

Вершина *v* ***досяжна*** з вершини *u*, якщо існує шлях, що починається в *u* і закінчується у *v*.

Граф називається ***зв'язковим***, якщо його вершини взаємно досяжні. На рисунку зображено зв'язковий граф. Якщо граф не є зв'язковим, його можна розбити на зв'язні підграфи, звані компонентами.

Компонента зв'язності – це максимальна зв'язкова підграфа. У деяких випадках граф може складатися з довільної кількості компонент зв'язності. Будь-яка ізольована вершина є окремим компонентом зв'язності. Наприклад, у наведеному нижче графі міститься 4 компоненти зв'язності: ***abhk, gd, c, f***.



Зважений граф – це граф, деяким елементам якого (вершинам, ребрам чи дугам) зіставлені числа. Числа носять назву ***вага*** (довжина, вартість).

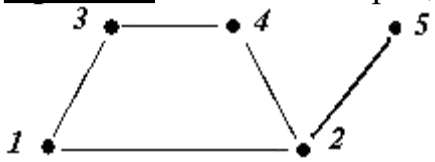
Довжина шляху у зваженому графі – це сума вагів ребер (дуг), з яких складається шлях.

Способи представлення графів у пам'яті

Графічний спосіб представлення графів непридатний для обчислювальної машини. Тому є інші способи представлення графів.

Матриця суміжності – це матриця $n \times n$, де n – число вершин. Рядок із номером i містить “1” у рядку з номером j, якщо існує дуга з вершини i у вершину j.

Приклад: Скласти матрицю суміжності для наступного графа:



Запишемо матрицю:

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

Матриця суміжності для неорієнтованого графа буде симетричною щодо своєї головної діагоналі, а орграфа - несиметричної.

Зручність матриці суміжності полягає у наочності та прозорості алгоритмів, заснованих на її використанні. А незручність - у дещо підвищеній вимогі до пам'яті: якщо граф далекий від повного, то в масиві, що зберігає матрицю суміжності, виявляється багато "порожніх місць" (нулів).

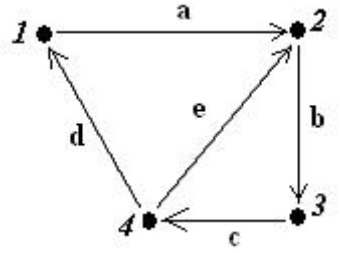
Матриця інцидентності

Матриця інцидентності має розмір *mxn*, де n – кількість вершин, m – кількість дуг графа. Елемент матриці, що відповідає k-дузі та i-ій вершині, дорівнює

- +1, якщо дуга виходить із вершини;
- –1, якщо дуга входить у вершину
- та нулі у всіх інших рядках.

Приклад.

Скласти матрицю інцидентності для наступного графа:



Запишемо матрицу:

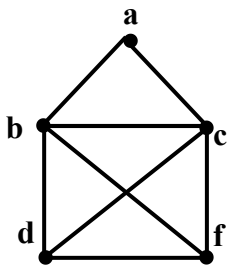
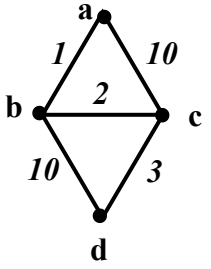
	a	*b*	*c*	*d*	*e*
1	1	0	0	-1	0
2	-1	1	0	0	-1
3	0	-1	1	0	0
4	0	0	-1	1	1
5	0	0	0	0	0

Список ребер

Цей спосіб завдання найбільш зручний для зовнішнього представлення вхідних даних. Кожен рядок вхідних даних містить інформацію про одне ребро або дугу у наступному вигляді:

<номер початкової вершини> <номер кінцевої вершини> [<вага ребра>]

Наприклад, для наступних графів отримаємо списки:

	**a b** **a c** **b c** **b d** **c d** **c f** **f d** **b f**
	**a b 1** **a c 10** **b c 2** **b d 10** **c d 3**

Списки суміжності

Цей спосіб завдання графів передбачає, що кожної вершини буде вказано список всіх суміжних із нею вершин (для орієнтованого графа – список вершин, є кінцями вихідних дуг):

<номер початкової вершини>: <номера суміжних вершин>

Приклад: (рис. графів див. вище)

1) **a: b c**
b: c d f
c: d f
d: f

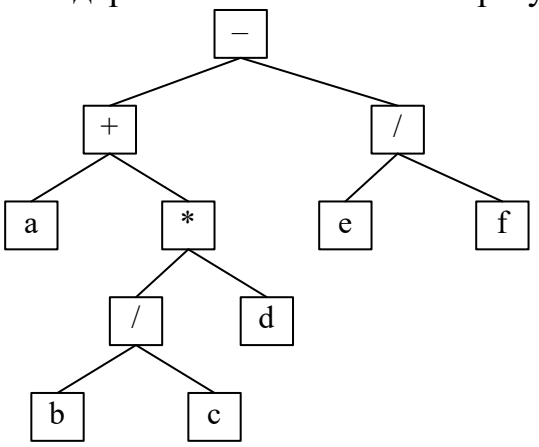
2) **a: b 1 c 10**
b: a 1 c 2 d 10
c: a 10 d 3

d: b 10 c 3

Цей метод представлення графів є внутрішньої реалізацією списку суміжності: у одному лінійному списку містяться номери початкових вершин, а в інших – списки вихідних їх дуг.

Приклад: Алгоритм аналізу формули з використанням графа.

Існуємо досить багато алгоритмів розбору математичного виразу (формул), що зберігається у вигляді символьного рядка. Одним із них є алгоритм розкладання формули з використанням графа, який розбирає вихідний математичний вираз та будує по ньому відповідне бінарне дерево. На малюнку представлено дерево математичного виразу: «a+b/c*d – e/f».



Вузлом дерева формули або знак операції, або ім'я змінної, або число. Для вузла зі знаком операції операндами є лівий та правий вузли-нащадки (а точніше, піддерев'я, корінням яких є лівий та правий вузол-нащадок) відповідного вузла.

Обхід графів

Обхід полягає у відвідуванні всіх вершин та дуг графа, виведенні всієї інформації про граф.

Обхід (пошук) в глибину

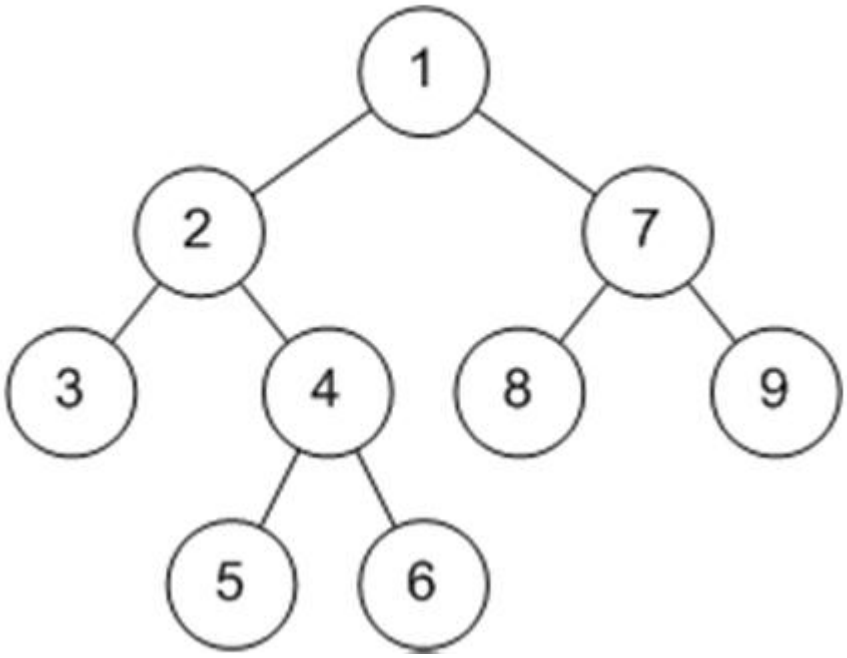
Метод пошуку в глибину (**depth first search, DFS**) — узагальнення методу обходу дерева у прямому порядку.

Обхід у глибину, є обхід графа за наступними правилами:

1. Додаємо початкову вершину в стек і помічаємо її як використану.
2. Переглядаємо вершину стека і додаємо в стек першу суміжну їй невикористану вершину, що потрапила, позначаючи її як використану. Якщо такої вершини немає, витягуємо вміст вершини стека.
3. Якщо стек не порожній, то переходимо до пункту 2.

Таким чином, цей алгоритм обходить всі вершини, що досягаються з початкової, і кожну вершину обробляє не більше одного разу.

Черговість перегляду вершин при пошуку в глибину має вигляд:



Приклад реалізації на псевдокоді наведено у лістингу:

```
const
  MAX_N = 10;
var
  graph:   array [1..MAX_N, 1..MAX_N] of boolean;  // масив для визначення
графа
  visited: array [1..MAX_N] of boolean;

procedure dfs(v: integer);
var
  i: integer;
begin
  visited[v] := true;

  for i := 1 to MAX_N do
    if graph[v, i] and not visited[i] then
      dfs(i);
end;
```

Обхід (пошук) у ширину

Обхід графа у ширину (**breadth first search, BFS**) ґрунтується на заміні стеку чергою:

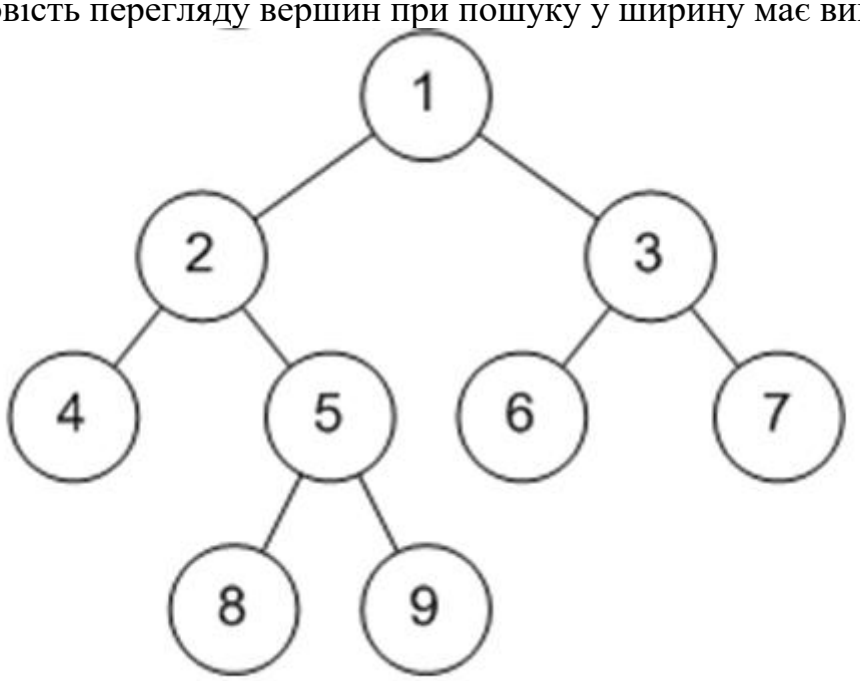
1. Додаємо початкову вершину в чергу та позначає її як використану.

2. Достаємо наступну вершину з черги і додаємо в чергу суміжні невикористані вершини, позначаючи їх як використані.

3. Якщо черга не є порожньою, переходимо до пункту 2.

Після такої модифікації чим раніше відвідується вершина (вміщується в чергу), тим раніше вона використовується (видаляється з черги). Використання вершини відбувається за допомогою перегляду відразу всіх ще не переглянутих вершин, суміжних до цієї вершини. Таким чином, "пошук ведеться як би у всіх можливих напрямках одночасно".

Черговість перегляду вершин при пошуку у ширину має вигляд:



Найчастіше пошук у ширину використовується для знаходження найкоротшого шляху від однієї вершини *X* до іншої - *Y*: потрібно запустити пошук у ширину з вершини *X* і при додаванні нової вершини в чергу дивитися, чи вона не є вершиною *Y*. Якщо програма не завершить роботу в умовному циклі, це означає, що шляху з *X* до *Y* не існує.

Завдання

Маршрути руху автобусів зі станції Київа;

1. Kyiv –(135) Zhytomyr –(80) Novohrad-Volynskyi –(100) Rivne –(68) Lutsk
2. Kyiv –(135) Zhytomyr –(38) Berdychiv –(73) Vinnytsia –(110) Khmelnytskyi –(104) Ternopil
3. Kyiv – (135) Zhytomyr – (115) Shepetivka
4. Kyiv – (78) Bila Tserkva – (115) Uman
5. Kyiv – (78) Bila Tserkva – (146) Cherkasy – (105) Kremenchuk
6. Kyiv – (78) Bila Tserkva – (181) Poltava – (130) Kharkiv
7. Kyiv – (128) Pryluky – (175) Sumy
8. Kyiv – (128) Pryluky – (109) Myrhorod

Порядок виконання роботи

1. Записати матрицю суміжності відповідно до завдання.
2. Реалізувати алгоритми DFS та BFS.
3. Записати можливі маршрути руху та відстані всього маршруту від центрального вокзалу (кореня) до всіх інших вершин.
4. Результати вивести на екран.

Формат вхідних даних:

Єдиний рядок вхідних даних, що містить число 1 або 2:

N

де:

N – 1 - DFS, 2 - BFS.

Формат вихідних даних:

Програма повинна вивести розраховані маршрути та відстань по одному в рядку.
Приклад:
Kyiv–Zhytomyr=135
Kyiv–Novohrad-Volynskyi=215
Kyiv–Rivne=315
тощо.

Зміст звіту

1. Описати алгоритм (словесна форма, блок-схема алгоритму).
2. Побудувати граф.
3. Привести текст функцій DFS та BFS із коментарями.
4. Висновки щодо роботи.