Лабораторна робота №2

**Тема:** Дерева. Обхід графів.

**Мета:** опрацювати алгоритми обходу графів.

[Теоретичні відомості 1](#_Toc506291654)

[1) Обхід графів. Пошук вшир 1](#_Toc506291655)

[Найкоротші відстані на основі пошуку вшир 2](#_Toc506291656)

[2) Пошук углиб 3](#_Toc506291657)

## ТЕОРЕТИЧНІ ВІДОМОСТІ

### Основні означення та властивості

Поняття дерева широко застосовують у багатьох розділах математики й інформатики. Наприклад, дерева використовують як інструмент обчислень, зручний спосіб збереження даних, їх сортування чи пошуку.

Означення 1. ***Деревом*** називають зв’язний граф без простих циклів.

Граф, який не містить простих циклів і складається з k компонент зв’язності, називають ***лісом*** з k дерев. З означення випливає, що дерева й ліси являють собою прості графи. На рис. 1 зображені приклади дерев.

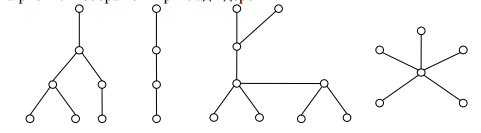


Рис. 1.

Теорема 1. Нехай граф G має n вершин. Тоді такі твердження еквівалентні:

1) граф G – дерево;

2) граф G не містить простих циклів і має (n–1) ребро;

3) граф G зв’язний і має (n–1) ребро;

4) граф G зв’язний, але вилучення довільного ребра робить його незв’язним;

5) довільні дві вершини графу G з’єднані точно одним простим маршрутом;

6) граф G не містить простих циклів, але додавши до нього нове ребро ми отримаємо точно один простий цикл.

## Обхід дерев

Чимало задач можна моделювати з використанням кореневих дерев. Поширене таке загальне формулювання задачі: виконати задану операцію D з кожною вершиною дерева. Тут D – параметр загальнішої задачі відвідування всіх вершин, або так званого обходу дерева. Розглядаючи розв’язання цієї задачі як єдиний послідовний процес відвідування вершин дерева в певному порядку, можна вважати їх розміщеними одна за одною. Опис багатьох алгоритмів істотно спрощується, якщо можна говорити про наступну вершину дерева, маючи на увазі якесь упорядкування. Є три принципи впорядкування вершин, які природно випливають зі структури дерева. Як і саму деревоподібну структуру, їх зручно формулювати за допомогою рекурсії.

Звертаючись до бінарного дерева, де R – корінь, A та B – ліве та праве піддерева (рис. 2), можна означити такі впорядкування.

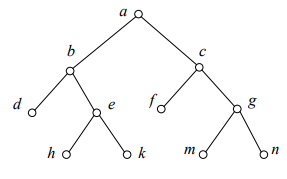


Рис. 2.

* обхід у ***прямому порядку (preorder)***, або згори донизу: R, A, B (корінь відвідують до обходу піддерев). Для дерева з рис. 2 послідовність вершин буде така: a, b, d, e, h, k, c, f, g, m, n. 2.
* обхід у ***внутрішньому порядку (inorder)***, або зліва направо: A, R, B. Для дерева з рис. 2 послідовність: d, b, h, e, k, a, f, c, m, g, n. 3.
* обхід у ***зворотному порядку (postorder)***, або знизу догори: A, B, R (корінь відвідують після обходу піддерев). Для дерева з рис. 2 послідовність:d, h, k, e, b, f, m, n, g, c, a.

Надзвичайно поширене в інформатиці застосування обходу дерев – зіставлення виразам (арифметичним, логічним тощо) дерев і побудова на цій основі різних форм запису виразів. Суть справи зручно пояснити на прикладі. Розглянемо арифметичний вираз:



Подамо його у вигляді дерева (рис. 3). Внутрішнім вершинам цього дерева відповідають символи операцій, а листкам – операнди.

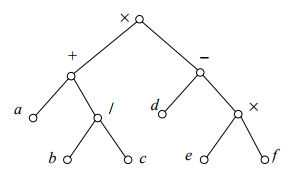


Рис. 3

Обійдемо це дерево, записуючи символи у вершинах у тому порядку, у якому вони зустрічаються в разі заданого способу обходу. Отримаємо такі три послідовності:

• у разі обходу в прямому порядку – ***префіксний*** (польський) запис

***× +a / bc − d × ef***

• у разі обходу у внутрішньому порядку – ***інфіксний*** запис (поки що без дужок, які потрібні для визначення порядку операцій)

***a + b / c × d − e × f***

• у разі обходу в зворотному порядку – ***постфіксний*** (зворотний польський) запис

***abc /+def× −×.***

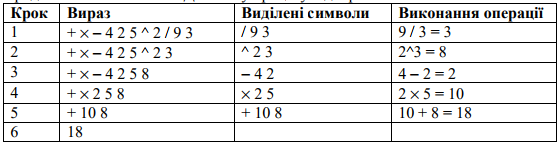
Звернемося спочатку до інфіксної форми запису виразу. Без дужок вона неоднозначна: один запис може відповідати різним деревам. Щоб уникнути неоднозначності інфіксної форми, використовують круглі дужки щоразу, коли зустрічають операцію. Повністю «одужкований» вираз, одержаний під час обходу дерева у внутрішньому порядку, називають інфіксною формою запису.

Отже, для дерева з рис. 3 інфіксна форма така: ((a + b c))/( × (d − (e ×f ))). Наведені міркування свідчать, що інфіксна форма запису виразів незручна. На практиці використовують префіксну та постфіксну форми, бо вони однозначно відповідають виразу й не потребують дужок. Ці форми запису називаються польським записом (на честь польського математика ***Яна Лукасевича***).

Для обчислення значення виразу в польському записі його проглядають справа наліво та знаходять два операнди разом зі знаком операції перед ними. Ці операнди та знак операції вилучають із запису, виконують операцію, а її результат записують на місце вилучення символів.

Для прикладу, обчислимо значення виразу в польському записі (^ - означає піднесення до степеня): «+ × − 4 2 5 ^ 2 / 9 3». За сформульованим правилом виділимо «/ 9 3», ці символи вилучимо й обчислимо 9 / 3 = 3; результат запишемо на місце вилучення символів «+ × − 4 2 5 ^ 2 3». Продовжимо обчислення. Динаміку процесу відображено в табл. 1.

Таблиця 1.

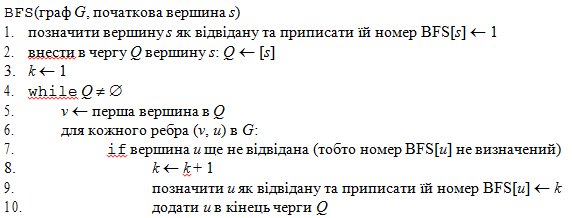


## Пошук вшир

Існує багато алгоритмів на графах, які ґрунтуються на систематичному переборі їх вершин або обході вершин, під час якого кожна вершина отримує унікальний порядковий номер. Виділяється два основних алгоритми обходів графів або пошуку у графах – це пошук вшир та пошук углиб.

Почнемо з методу **пошуку вшир** або BFS-методу (*breadth first search*). Нехай *G* = (*V*, *E*) – простий зв’язний граф, усі вершини якого позначено попарно різними символами. У процесі пошуку вглиб вершинам графу *G* надають номери (BFS-номери). У ході роботи алгоритму використовують структуру даних для збереження множин, яку називають чергою. З черги можна вилучити тільки той елемент, який було додано до неї першим: черга працює за принципом «першим прийшов – першим вийшов» (*first in, first out* – скорочено FIFO). Елемент включається у хвіст черги, а виключається з її голови.

Нижче наводиться процедура пошуку вшир BFS. Вона приймає на вхід два параметри: граф *G* (який може бути представлений будь-яким зручним способом: матрицями суміжності та інцидентності, списком суміжності) та початкова вершина *s*. Процедура починає обхід графу з вершини *s*, додаючи у чергу *Q* нові вершини. Робота закінчується, коли черга *Q* стає порожньою.



Лістинг 1. Процедура BFS пошуку вшир в графі

Щоб результат виконання алгоритму був однозначним, вершини, які суміжні з вершиною *v*, аналізують за зростанням їх порядкових номерів (або в алфавітному порядку). Динаміку роботи алгоритму зручно відображати за допомогою таблиці з трьома стовпцями: вершина, BFS-номер, уміст черги. Цю таблицю називають протоколом обходу графу пошуком вшир.

Для прикладу виконаємо обхід графу, який зображено на рис. 3, починаючи з вершини *b*. Протокол обходу наведено в таблиці 2.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  | | --- | --- | --- | | **Вершина** | **BFS-номер** | **чергa** | | *b* | 1 | *b* | | *c* | 2 | *bc* | | *d* | 3 | *bcd* | | - | - | *cd* | | *f* | 4 | *cdf* | | *g* | 5 | *cdfg* | | *h* | 6 | *cdfgh* | | - | - | *dfgh* | | *a* | 7 | *dfgha* | | - | - | *fgha* | | 8 | *e* | *fghae* | | - | - | *ghae* | | - | - | *hae* | | - | - | *ae* | | - | - | *e* | | - | - | ∅ | |
| Рис. 3. | Табл. 2. |

У процесі роботи алгоритму пошуку вшир будується дерево *T* пошуку. Це дерево є підграфом графу *G* і складається з тих ребер (*v*, *u*), які були розглянуті на кроці 6 процедури BFS. Якщо граф *G* зв’язаний, то дерево *T* буде включати в себе всі вершини графу. Таке дерево називається **покриваючим деревом графу**. На рис. 3 ребра дерева *T* позначені потовщеними лініями.

Досліджуючи алгоритм пошуку вшир можна прийти до наступного висновку.

Лема 1. Якщо вершина *v* графу *G* зустрілась під час роботи процедури BFS(*G*, *s*), то в графі *G* існує шлях від *s* до *v*.

Доведення леми тривіальне та ґрунтується на аналізі коду процедури BFS (див. лістинг 1).

### Найкоротші відстані на основі пошуку вшир

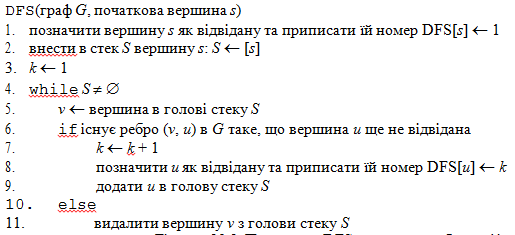
Пошук вшир може використовуватись не тільки для обходу графу, але й для інших практичних задач. По-перше, метод пошуку вшир в графі *G* дозволяє знайти найкоротші шляхи від початкової вершини *s* до будь-якої іншої досяжної від неї вершини *v* графу.

Щоб реалізувати можливість обрахунку найкоротших відстаней від початкової вершини *s* до всіх інших досяжних від неї вершин, процедура BFS потребує мінімальних змін. Необхідно ввести додатковий параметр *dist*[*v*], для будь-якої вершини, ініціалізувавши його 0 для початкової вершини *s* та +1для всіх інших вершин графу. А також після рядка 8 процедури BFS необхідно додати інструкцію: *dist*[*v*] <*dist*[*v*] + 1.

## Пошук углиб

Другий базовий метод обходу графу – алгоритм **пошуку углиб** або DFS-метод (*depth first search*). У процесі його роботи всім вершинам графу надаються номери (DFS-номери). На відміну від методу пошуку вшир, пошук углиб використовує структуру даних стек для збереження вершин-кандидатів. Зі стеку можна вилучити тільки той елемент, який було додано до нього останнім: стек працює за принципом «останнім прийшов – першим вийшов» (*last in, first out* – скорочено LIFO). Інакше кажучи, додавання й вилучення елементів у стеку відбувається з одного кінця, який називається верхівкою стеку.

Таким чином, процедура DFS повністю повторює процедуру BFS, відрізняючись від неї тільки тим, що замість черги тут використовується стек.



Лістинг 2. Процедура DFS пошуку вглиб в графі

Аналогічно до пошуку вшир, для відслідковування роботи процедури DFS можна використовувати протокол обходу. Для прикладу виконаємо обхід графу з рис. 4, починаючи так само з вершини *b*. Результат обходу зображено на рис.6, протокол обходу – в таблиці 3.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  | | --- | --- | --- | | **Вершина** | **DFS-номер** | **Cтек** | | *b* | 1 | *b* | | *c* | 2 | *bc* | | *d* | 3 | *bcd* | | *a* | 4 | *bcda* | | - | - | *bcd* | | - | - | *bc* | | *f* | 5 | *bcf* | | *e* | 6 | *bcfe* | | *g* | 7 | *bcfeg* | | - | - | *bcfe* | | - | - | *bcf* | | - | - | *bc* | | *h* | 8 | *bch* | | - | - | *bc* | | - | - | *b* | | - | - | ∅ | |
| Рис. 4. | Табл. 3. |

У процесі роботи алгоритму пошуку вглиб, так само як і у випадку пошуку вшир, будується дерево *T* пошуку. Це дерево є підграфом графу *G* і складається з тих ребер (*v*, *u*), які були розглянуті на кроці 6 процедури DFS. Якщо граф *G* зв’язаний, то дерево *T* буде включати в себе всі вершини графу. На рис. 4 ребра дерева *T* позначені потовщеними лініями.

Процедура пошуку вглиб у такому вигляді застосовується у багатьох практичних задачах.

## УМОВИ ПО ВАРІАНТАХ

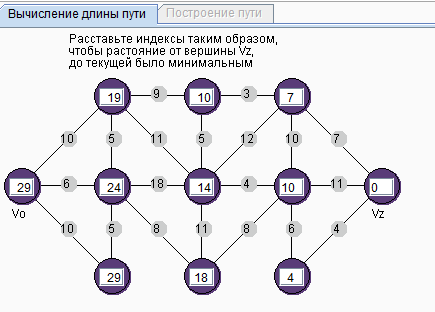
### Завдання 1.

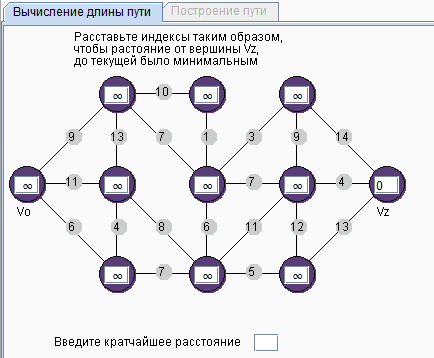
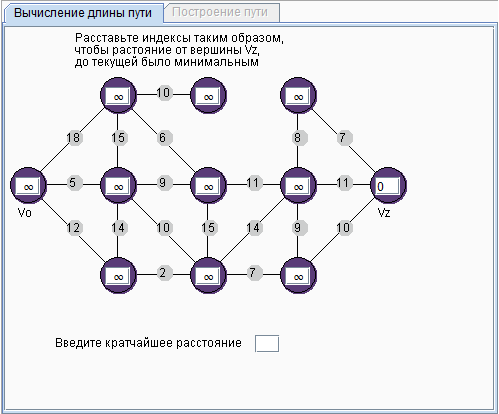
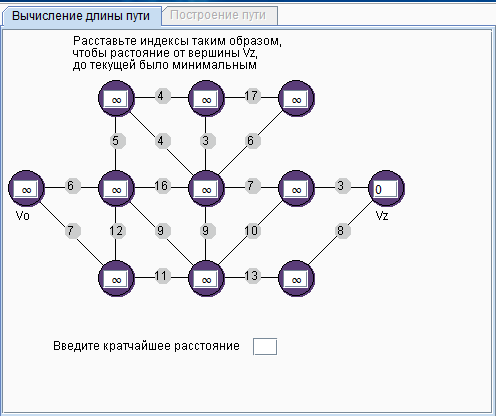
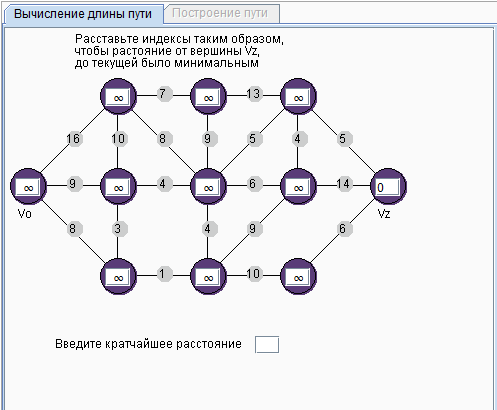
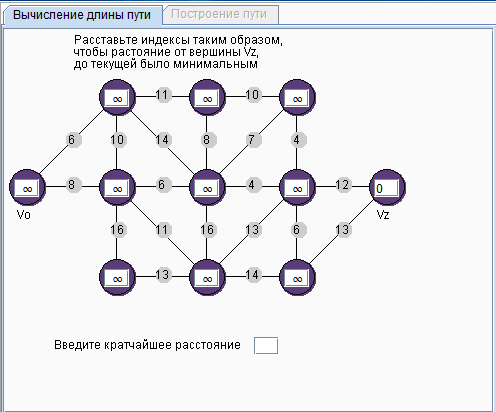
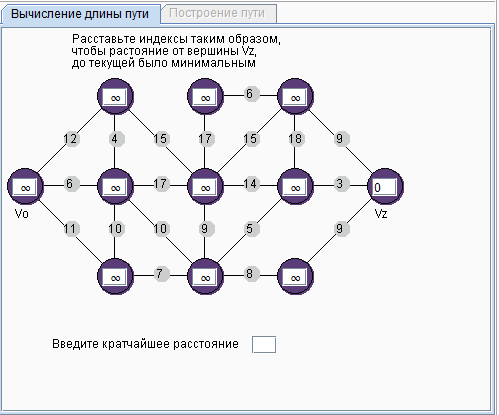
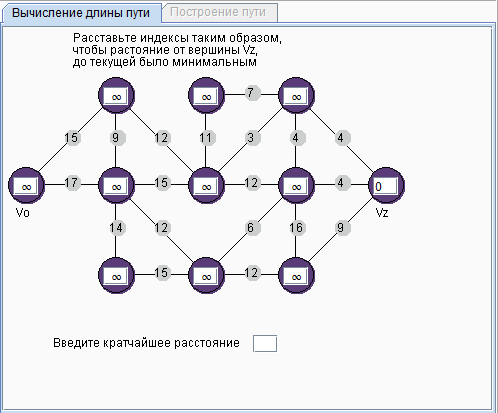
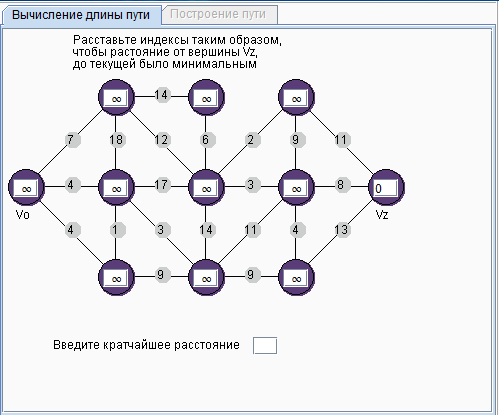
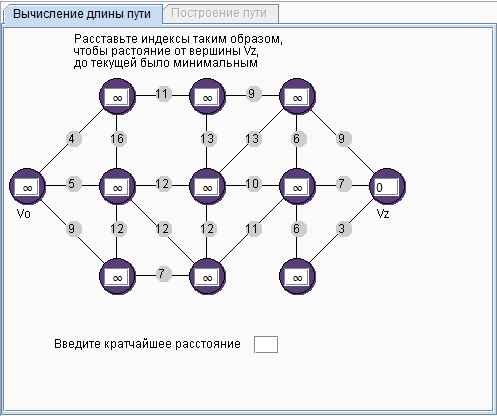
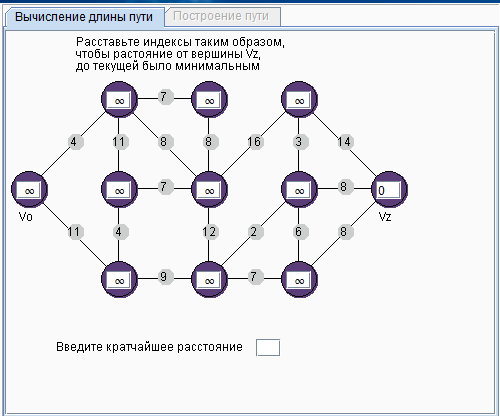
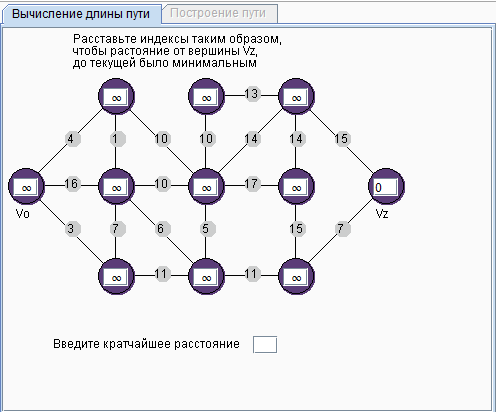
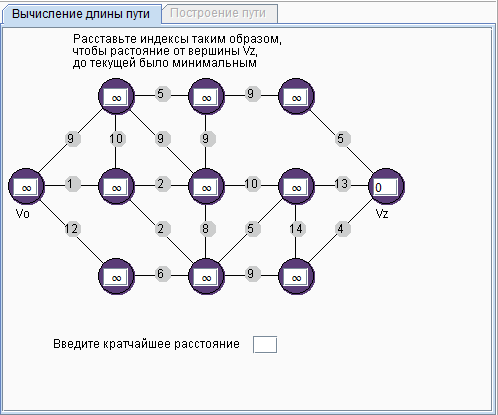
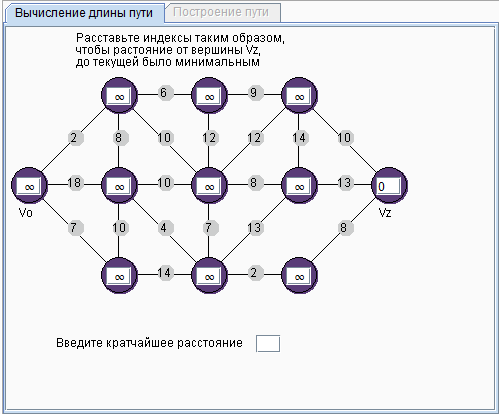
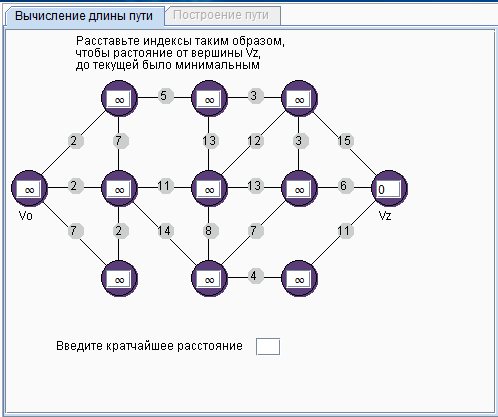
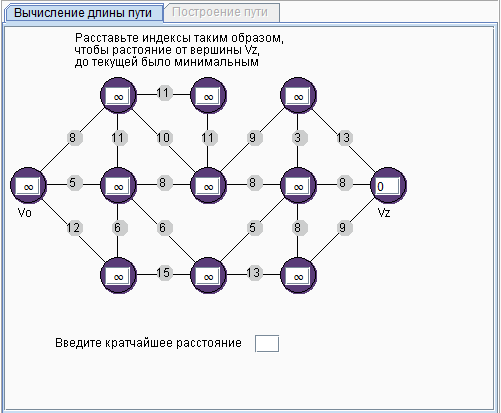
Побудувати три види обходів дерев.

### Завдання 2.

Для зваженого графа у варіанті побудувати дерева за алгоритмами BFS, DFS за початкову вершину взяти вказану в таблиці.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |  | | --- | --- | --- | --- | --- | | Варіант | Вершина |  | Варіант | Вершина | | 1 | 4 |  | 9 | 8 | | 2 | 1 |  | 10 | 4 | | 3 | 5 |  | 11 | 5 | | 4 | 9 |  | 12 | 2 | | 5 | 6 |  | 13 | 9 | | 6 | 2 |  | 14 | 3 | | 7 | 3 |  | 15 | 1 | | 8 | 7 |  | 0 | 7 | |



1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 
10. 
11. 
12. 
13. 
14. 
15. 

### Приклад розвязку 0 варіанту.

Завдання 1. Побудувати три види обходів дерев.

|  |  |
| --- | --- |
| * обхід у *прямому порядку (preorder)*   ***\* - 7 / 4 b / + 2 x 5***   * обхід у *внутрішньому порядку (inorder)*   ***7 - 4 / b \* 2 + x / 5***   * обхід у *зворотному порядку (postorder)*   ***7 4 b / - 2 x + / 5*** |  |

Завдання 2. Для зваженого графа у варіанті побудувати дерева за алгоритмами BFS, DFS за початкову вершину взяти вказану в таблиці.

За початкову вершину взято 7

|  |  |
| --- | --- |
|  |  |
| |  |  |  | | --- | --- | --- | |  | **BFS-номер** | **чергa** | | 7 | 1 | 7 | | 3 | 2 | 7,3 | | 6 | 3 | 7,3,6 | | 8 | 4 | 7,3,6,8 | | 10 | 5 | 7,3,6,8,10 | | 11 | 6 | 7,3,6,8,10,11 | | - | - | 3,6,8,10,11 | | 2 | 7 | 3,6,8,10,11,2 | | - | - | 6,8,10,11,2 | | 1 | 8 | 6,8,10,11,2,1 | | 5 | 9 | 6,8,10,11,2,1,5 | | - | - | 8,10,11,2,1,5 | | - | - | 10,11,2,1,5 | | 9 | 10 | 10,11,2,1,5,9 | | - | - | 11,2,1,5,9 | | - | - | 2,1,5,9 | | - | - | 1,5,9 | | 4 | 11 | 1,5,9,4 | | - | - | 5,9,4 | | - | - | 9,4 | | - | - | 4 | | - | - |  | | |  |  |  | | --- | --- | --- | |  | **DFS-номер** | **стек** | | 7 | 1 | 7 | | 3 | 2 | 3,7 | | 2 | 3 | 2,3,7 | | 1 | 4 | 1,2,3,7 | | 4 | 5 | 4,1,2,3,7 | | 5 | 6 | 5,4,1,2,3,7 | | 6 | 7 | 6,5,4,1,2,3,7 | | 10 | 8 | 10,6,5,4,1,2,3,7 | | 9 | 9 | 9,10,6,5,4,1,2,3,7 | | - | - | 10,6,5,4,1,2,3,7 | | 11 | 10 | 11,10,6,5,4,1,2,3,7 | | 6 | 11 | 6,11,10,6,5,4,1,2,3,7 | | - | - | 11,10,6,5,4,1,2,3,7 | | - | - | 10,6,5,4,1,2,3,7 | | - | - | 6,5,4,1,2,3,7 | | - | - | 5,4,1,2,3,7 | | - | - | 4,1,2,3,7 | | - | - | 1,2,3,7 | | - | - | 2,3,7 | | - | - | 3,7 | | - | - | 7 | | - | - |  | |