Лабораторна робота № 5

Теоретичні відомості. Обхід графів

Задача обходу графа полягає у систематичному переборі його вершин, при якому кожна вершина отримує унікальний номер. Ще алгоритми обходу графа називаються *методами пошуку*, оскільки дозволяють знаходити вершини з заданими властивостями. Розглянемо два базових алгоритми обходу графа – *пошук углиб та вшир*. Ці методи є складовими частинами багатьох інших алгоритмів на графах. Зокрема, в процесі обходу будується *каркасне дерево пошуку вглиб або вшир* (див. лабораторну роботу № 9).

Часова обчислювальна складність для обох алгоритмів становить $O(n+m)$, де n – кількість вершин, m – кількість ребер графа.

Зауваження. Розглядатимемо випадок простого, зв'язного, неорієнтованого графа. Звичайно, дані алгоритми, з певними змінами, можна використовувати і для інших типів графів, зокрема, незв'язних та орієнтованих.

Пошук углиб

У процесі пошуку вглиб (інша назва DFS-метод (англ. *Depth First Search*)) вершинам графа надають DFS-номери та певним способом позначають ребра, якщо паралельно відбувається побудова каркасного дерева пошуку вшир. Існує дві реалізації алгоритму пошуку вглиб: нерекурсивна і рекурсивна. У нерекурсивній версії використовується структура даних типу *стек* (див. лабораторну роботу № 4).

Алгоритм пошуку вглиб (нерекурсивна реалізація).

1. Задати вершину з якої починається пошук (`start`). Позначити її як відвідану (`visited[start]=true`). Задати їй перший DFS-номер

(`dfsnumber[start]=1`). Занести цю вершину у стек (`push(start)`).

2. Розглянути вершину, що знаходиться у вершині стеку (`head->data`). Якщо всі суміжні з нею вершини відвідані (`visited[…]==true`), то переходимо до кроку 4, інакше – до кроку 3.
3. Нехай `x` – не відвідана вершина (`visited[x]==false`), суміжна з вершиною у вершині стеку (`head->data`). Позначаємо її як відвідану (`visited[x]=true`), задаємо їй наступний DFS-номер (`dfsnumber[x]=++DFS`), заносимо цю вершину в стек (`push(x)`). У випадку реалізації побудови каркасного дерева, ребро між цими вдома вершинами (`head->data` та `x`) додаємо у каркас. Повертаємось до кроку 2.
4. Видаляємо вершину зі стеку (`pop()`). Якщо стек порожній – припиняємо роботу, інакше – повертаємось до кроку 2.

Алгоритм пошуку вглиб (рекурсивна реалізація)

1. Розглянути поточну вершину (`vertex`). Позначити її як відвідану (`visited[vertex]=true`). Задати їй наступний DFS-номер (`dfsnumber[vertex]=++DFS`).
2. Розглянути вершини, суміжні з вершиною `vertex`. Якщо серед них є не відвідана вершина `x` (`visited[x]==false`), то викликати рекурсивно функцію, що реалізує кроки 1, 2 для цієї вершини `x`. У випадку реалізації побудови каркасного дерева, ребро між цими вдома вершинами (`vertex` та `x`) додаємо у каркас.
3. Задати початкову вершину і з неї почати рекурсивний виклик. Рекурсія зупиняється, якщо всі вершини були відвідані (отримали DFS-номери) або граф є незв’язним.

Зауваження. Щоб не було неоднозначності у виборі не відвіданих

вершин, потрібно домовитись, в якому порядку їх розглядати. Вершини можна аналізувати, наприклад, за зростанням їх порядкових номерів або в алфавітному порядку.

Приклад.

Розглянемо обхід графа, наведеного на рис. 5.1 методом пошуку вглиб, починаючи з вершини v_1. Результат обходу подано на рис. 5.2: біля кожної вершини у дужках подано її DFS-номер. На цьому ж рисунку ребра, позначені потовщеною суцільною лінією утворюють каркасне дерево пошуку вглиб і називаються „прямими". Ребра, позначені пунктирною лінією називаються „зворотними". Не відвідані вершини вибираються у порядку зростання їх номерів. Процес роботи алгоритму відображено в таблиці 5.1 (*протокол пошуку*). У цій таблиці в третьому стовпці вважаємо, що верхівка стеку знаходиться ліворуч.

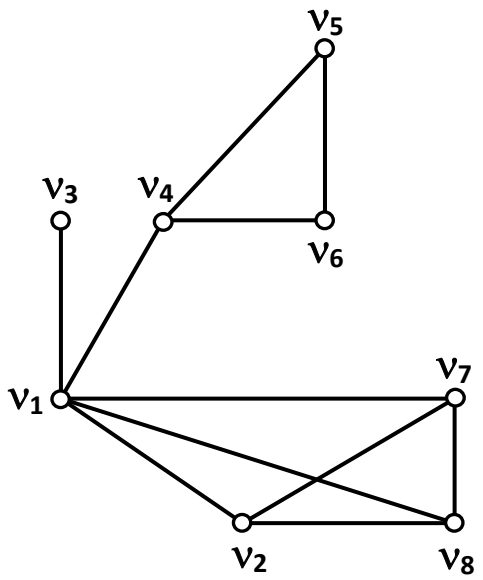

Рис. 5.1

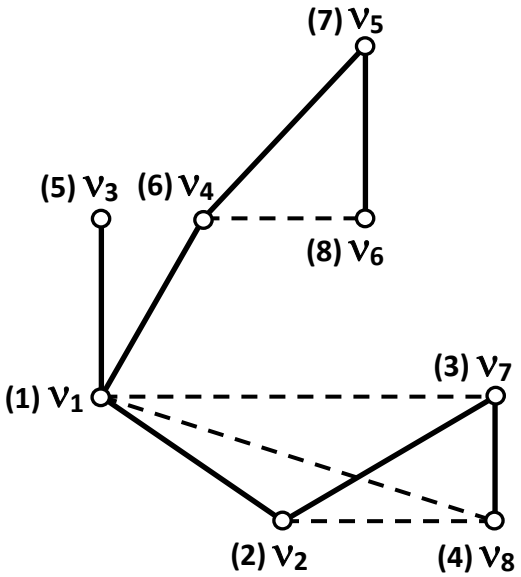

Рис. 5.2

Таблиця 5.1

Вершина	DFS-номер	Вміст стеку
v_1	1	$\rightleftarrows v_1$
v_2	2	$\rightleftarrows v_2 v_1$
v_7	3	$\rightleftarrows v_7 v_2 v_1$
v_8	4	$\rightleftarrows v_8 v_7 v_2 v_1$
–	–	$\rightleftarrows v_7 v_2 v_1$
–	–	$\rightleftarrows v_2 v_1$
–	–	$\rightleftarrows v_1$
v_3	5	$\rightleftarrows v_3 v_1$
–	–	$\rightleftarrows v_1$
v_4	6	$\rightleftarrows v_4 v_1$
v_5	7	$\rightleftarrows v_5 v_4 v_1$
v_6	8	$\rightleftarrows v_6 v_5 v_4 v_1$
–	–	$\rightleftarrows v_5 v_4 v_1$
–	–	$\rightleftarrows v_4 v_1$
–	–	$\rightleftarrows v_1$
–	–	$\rightleftarrows$ NULL

Хід роботи:

Частина 1. Нерекурсивна реалізація пошуку вглиб

1. Створити проект `Lab_5` до якого підключити бібліотеку `List` для роботи з лінійними списками (файли `List.h`, `List.cpp`; див. лабораторну роботу № 4).
2. Згідно описаного вище алгоритму реалізувати нерекурсивний обхід графа методом пошуку вглиб.
3. Для заданого викладачем графа реалізувати обхід DFS-методом. Граф задати матрицею або списком суміжності.
4. Вивести на екран результати роботи програми: номери (назви) вершин, їхні DFS-номери, протокол пошуку та перелік ребер (пар вершин), що утворюють каркас.

Частина 2. Рекурсивна реалізація пошуку вглиб

1. Створити функцію `DFS(…)`, яка реалізовує описаний вище рекурсивний алгоритм пошуку вглиб. Параметром цієї функції має бути змінна, що визначає вершину графа `vertex`. Підключити функцію `DFS(…)` у проект `Lab_5`. У функції `main()` реалізувати меню для вибору рекурсивного чи нерекурсивного методу пошуку вглиб.
2. Для заданого викладачем графа реалізувати обхід рекурсивним DFS-методом. Граф задати матрицею або списком суміжності.
3. Вивести на екран результати роботи програми: номери (назви) вершин, їхні DFS-номери, протокол пошуку та перелік ребер (пар вершин), що утворюють каркас.

Пошук ушир

У процесі пошуку вшир (інша назва BFS-метод (англ. *Breadth First Search*)) вершинам графа надають BFS-номери та певним способом позначають ребра, якщо паралельно відбувається побудова каркасного дерева пошуку вглиб.

При реалізації пошуку вшир використовується структура даних типу *черга* (див. лабораторну роботу № 4).

Алгоритм пошуку вшир.

1. Задати вершину з якої починається пошук (`start`). Позначити її як відвідану (`visited[start]=true`). Задати їй перший BFS-номер (`bfsnumber[start]=1`). Занести цю вершину у чергу (`enqueue(start)`).
2. Розглянути вершину, що знаходиться у голові черги (`head->data`). Якщо всі суміжні з нею вершини відвідані (`visited[…]==true`), то переходимо до кроку 4, інакше – до кроку 3.
3. Нехай `x` не відвідана вершина (`visited[x]==false`), суміжна з вершиною у голові черги (`head->data`). Позначаємо її як відвідану (`visited[x]=true`), задаємо їй наступний BFS-номер (`bfsnumber[x]=++BFS`), заносимо цю вершину в чергу (`enqueue(x)`). У випадку побудови каркасного дерева, ребро між цими вдома вершинами (`head->data` та `x`) додаємо у каркас. Повертаємось до кроку 2.
4. Видаляємо вершину зі черги (`dequeue()`). Якщо черга порожня – припиняємо роботу, інакше – повертаємось до кроку 2.

Зауваження. Щоб не було неоднозначності у виборі не відвіданих вершин, потрібно домовитись, в якому порядку їх розглядати. Вершини можна аналізувати, наприклад, за зростанням їх порядкових номерів або в

алфавітному порядку.

Приклад.

Розглянемо обхід графа, наведеного на рис. 5.1 методом пошуку вшир, починаючи з вершини ν_1. Результат обходу подано на рис. 5.3: біля кожної вершини у дужках подано її BFS-номер. На цьому ж рисунку ребра, позначені потовщеною суцільною лінією утворюють каркасне дерево пошуку вшир. Не відвідані вершини вибираються у порядку зростання їх номерів. Протокол пошуку наведено в таблиці 5.2. У цій таблиці в третьому стовпці вважаємо, що голова черги знаходиться ліворуч, а хвіст - праворуч.

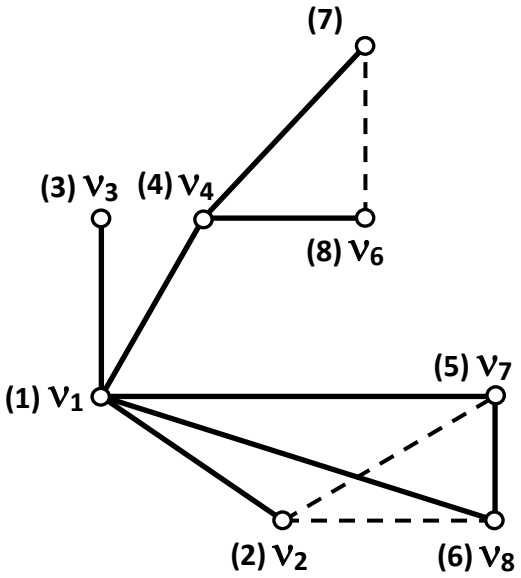

Рис. 5.3

Таблиця 5.2

Вершина	BFS-номер	Вміст черги
v_1	1	$\leftarrow v_1 \leftarrow$
v_2	2	$\leftarrow v_1 v_2 \leftarrow$
v_3	3	$\leftarrow v_1 v_2 v_3 \leftarrow$
v_4	4	$\leftarrow v_1 v_2 v_3 v_4 \leftarrow$
v_7	5	$\leftarrow v_1 v_2 v_3 v_4 v_7 \leftarrow$
v_8	6	$\leftarrow v_1 v_2 v_3 v_4 v_7 v_8 \leftarrow$
–	–	$\leftarrow v_2 v_3 v_4 v_7 v_8 \leftarrow$
–	–	$\leftarrow v_3 v_4 v_7 v_8 \leftarrow$
–	–	$\leftarrow v_4 v_7 v_8 \leftarrow$
v_5	7	$\leftarrow v_4 v_7 v_8 v_5 \leftarrow$
v_6	8	$\leftarrow v_4 v_7 v_8 v_5 v_6 \leftarrow$
–	–	$\leftarrow v_7 v_8 v_5 v_6 \leftarrow$
–	–	$\leftarrow v_8 v_5 v_6 \leftarrow$
–	–	$\leftarrow v_5 v_6 \leftarrow$
–	–	$\leftarrow v_6 \leftarrow$
–	–	$\leftarrow$ NULL $\leftarrow$

Хід роботи:

Частина 3. Реалізація пошуку вшир

1. Доповнити проект `Lab_5` реалізацією описаного вище алгоритму обходу графа методом пошуку вшир.
2. У функції `main()` доповнити створене у першій частині роботи меню з можливістю вибору методу пошуку вшир.
3. Для заданого викладачем графа реалізувати обхід BFS-методом. Граф задати матрицею або списком суміжності.
4. Вивести на екран результати роботи програми: номери (назви) вершин, їхні BFS-номери, протокол пошуку та перелік ребер (пар вершин), що утворюють каркас.