Розділ 12. Алгоритми на графах. Основні терміни теорії графів. Представлення графів в комп'ютерних програмах

Теорія графів останнім часом широко використовується в різних галузях науки й техніки. Швидкий розвиток дана теорія одержала зі створенням електронно-обчислювальної техніки, що дозволила вирішити багато задач алгоритмізації.

Ключові терміни

Граф – це сукупність двох скінченних множин: множини точок і множини ліній, що попарно з'єднують деякі із цих точок. Множина точок називається *вершинами (вузлами) графа*. Множина ліній, що з'єднують вершини графа, називається *ребрами (дугами) графа*.

Орієнтований граф (орграф) – граф, у якого всі ребра орієнтовані, тобто ребрам якого привласнений напрямок.

Неорієнтований граф (неорграф) – граф, у якого всі ребра неорієнтовані, тобто ребрам якого не заданий напрямок.

Змішаний граф – граф, що містить як орієнтовані, так і неорієнтовані ребра.

Петлею називається ребро, що з'єднує вершину саму із собою. Дві вершини називаються *суміжними*, якщо існує з'єднуюче їх ребро. Ребра, що з'єднують ті самі пари вершин, називаються *кратними*.

Простий граф – це граф, у якому немає ні петель, ні кратних ребер.

Мультиграф – це граф, у якого будь-які дві вершини з'єднані більш ніж одним ребром.

Маршрутом у графі називається скінченна послідовність суміжних вершин і ребер, що з'єднують ці вершини.

Маршрут називається *відкритим*, якщо його початкова й кінцева вершини різні, у протилежному випадку він називається *замкнутим*.

Маршрут називається **ланцюгом**, якщо всі його ребра різні. Відкритий ланцюг називається **шляхом**, якщо всі його вершини різні.

Замкнутий ланцюг називається **циклом**, якщо різні всі його вершини, за винятком кінцевих.

Граф називається *зв'язним*, якщо для будь-якої пари вершин існує з'єднуючий їх шлях.

Вага вершини – число (дійсне, ціле або раціональне), поставлене у відповідність даній вершині (інтерпретується як вартість, пропускна здатність і так далі). *Вага (довжина) ребра* – число або декілька чисел, які інтерпретуються відносно ребра як довжина, пропускна здатність і так далі.

Зважений граф – граф, кожному ребру якого поставлено у відповідність якесь значення (вага ребра).

Способи представлення графа в комп'ютерній програмі

Вибір структури даних для зберігання графа в пам'яті комп'ютера має принципове значення при розробці ефективних алгоритмів. Розглянемо кілька **способів представлення графа**.

Нехай заданий граф (наприклад, рис. 12.1), у якого кількість вершин дорівнює *n*, а кількість ребер – *m*. Кожне ребро й кожна вершина мають вагу – ціле позитивне число. Якщо вага ребра не вказана, то вважається, що вага дорівнює одиниці.

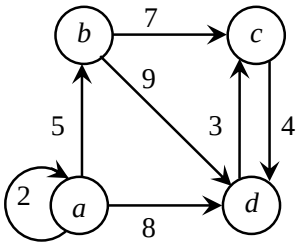

Рисунок 12.1 – Граф

1. *Список ребер* – це множина, утворена парами суміжних вершин (рис. 12.2). Для його зберігання звичайно використовують одномірний масив розміром m, що містить список пар вершин, суміжних з одним ребром графа. Список ребер більш зручний для реалізації різних алгоритмів на графах у порівнянні з іншими способами.

a	*a*	*a*	*b*	*b*	*c*	*d*
a	*b*	*d*	*c*	*d*	*d*	*c*
2	*5*	*8*	*7*	*9*	*4*	*3*

Рисунок 12.2 – Список ребер графа

2. *Матриця суміжності* – це двовимірний масив розмірності *n* x *n*, значення елементів якого характеризуються суміжністю вершин графа (рис. 12.3). При цьому значенню елементу матриці привласнюється кількість ребер, які з'єднують відповідні вершини. Даний спосіб зручний, коли треба перевіряти суміжність або знаходити вагу ребра за двома заданими вершинами.

	a	*b*	*c*	*d*
a	2	5	0	8
b	0	0	7	9
c	0	0	0	4
d	0	0	3	0

Рисунок 12.3 – Матриця суміжності графа

3. *Матриця інцидентності* – це двовимірний масив розмірності *n* x *m*, у якому вказуються зв'язки між інцидентними елементами графа (ребро й вершина). Стовпці матриці відповідають ребрам, рядки – вершинам (рис. 12.4). Ненульове значення в комірці матриці вказує зв'язок між

вершиною й ребром. Даний спосіб є самим ємним для зберігання, але полегшує знаходження циклів у графі.

	a	*b*	*c*	*d*
(a, a)	2	0	0	0
(a, b)	0	5	0	0
(a, d)	0	0	0	8
(b, c)	0	0	7	0
(b, d)	0	0	0	9
(c, d)	0	0	0	4
(d, c)	0	0	3	0

Рисунок 12.4 – Матриця інцидентності графа

4. Списки суміжності. Цей спосіб представлення *графів* має на увазі, що для кожної *вершини* буде зазначений список всіх суміжних з нею *вершин* (для *орграфа* – список *вершин*, що є кінцями вихідних *дуг*). Конкретний формат вхідного файлу, що містить *списки суміжності*, необхідно обговорювати окремо. Наприклад, можна використати наступний формат – початкова *вершина* відділена від *списку суміжності* двокрапкою:

<номер_початкової_вершини>: <номери_суміжних_вершин>

Найбільш природно застосовувати цей спосіб для представлення *орграфів*, однак і для інших варіантів він теж підходить.

a: a 2 b 5 d 8
b: c 7 d 9
c: d 4
d: c 3

Рисунок 12.5 – Список суміжності графа

5. *Ієрархічний список.* Цей спосіб представлення графів є всього лише внутрішньою реалізацією списку суміжності: в одному лінійному списку містяться номери "початкових вершин", а в інших – номери суміжних вершин або вказівники на ці вершини.

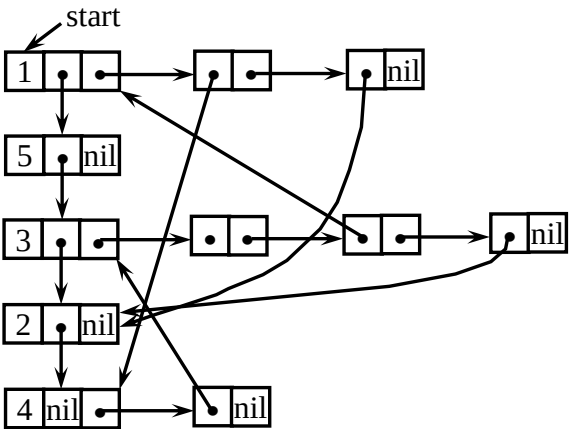

Рисунок 12.6 – Приклад ієрархічного списку

Очевидна перевага такого способу представлення *графів* полягає в економічному використанні пам'яті. І навіть невелика надмірність даних, до якої доводиться прибігати у випадку неорієнтованого *графа*, задаючи кожне *ребро* як дві *дуги*, окупається гнучкістю всієї структури, що особливо зручно при необхідності частих перебудовувань у процесі роботи програми.

Якщо в наведені описи типів даних додати поля, які могли б зберігати вагу *вершин* і *дуг*, то таким же способом можна задавати й зважені *графи*.

Дерева

Дерево – це окремий випадок *графа*, що найбільш широко застосовується у програмуванні.

Основні визначення

Існує досить багато рівносильних визначень *дерев*, от лише деякі з них.

1. *Дерево* – це зв'язаний *граф* без *циклів*.

2. *Дерево* – це зв'язаний *граф*, у якому при *N вершинах* завжди рівно (*N* – 1) *ребро*.

3. *Дерево* – це *граф*, між будь-якими двома *вершинами* якого існує рівно один *шлях*.

Аналогічним чином визначається й **орієнтоване** *дерево* – як *оргграф*, у якому між будь-якими двома *вершинами* існує не більше одного *шляху*.

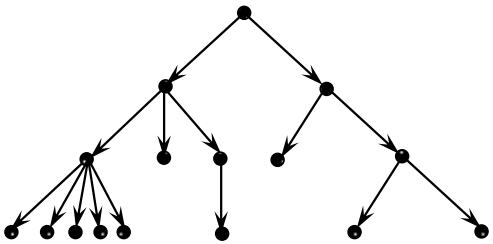

Рисунок 12.7 – Кореневе дерево з висотою 3

Кореневе дерево – це орієнтоване *дерево*, у якому можна виділити *вершини* трьох видів: *корінь*, *листи* (інша їхня назва: ***термінальні вершини***) й інші *вершини* (***нетермінальні***); причому повинні виконуватися дві обов'язкових умови:

1) з *листів* не виходить жодна *дуга*; з інших *вершин* може виходити безліч *дуг*;

2) у *корінь* не заходить жодна *дуга*; в усі інші *вершини* заходить рівно по одній *дузі*.

Традиційно в математиці й близьких їй науках (у тому числі й у теоретичному програмуванні) *дерева* "ростуть" вниз

головою: це робиться просто для зручності нарощування *листів* якщо буде потреба. Таким чином, на малюнках *корінь дерева* виявляється самою верхньою *вершиною*, а *листи* – самими нижніми.

***Предок** вершини* v – це *вершина*, з якої виходить *дуга*, що заходить у *вершину* v. *Нащадок вершини v* – це *вершина*, у яку заходить *дуга*, що виходить із *вершини v*. У цих термінах можна дати інші визначення поняттям *корінь* і *лист*: у *кореня* немає *предків*, в *листа* немає *нащадків*.

***Висота** кореневого дерева* – це максимальна кількість *дуг*, що відокремлюють *листи* від кореня. Якщо *дерево* не зважене, то його *висота* – це проста *відстань* від кореня до самого віддаленого листа.

Каркас графа – це *дерево*, отримане після викидання із *графа* деяких ребер (рис. 12.8).

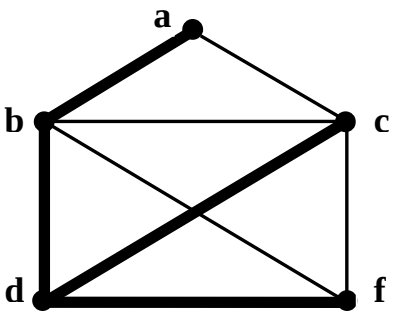

Рисунок 12.8 – Каркас графа

Прикладом *каркаса* є (*кореневе*) *дерево* найкоротших *шляхів* від деякої виділеної *вершини* (вона буде коренем *каркаса*) до всіх інших *вершин графа*.

Питання до розділу:

1. Поясніть поняття граф, вузли графа, дуги графа, корінь графа, орграф, неорграф, змішаний граф, петля, суміжні вершини, кратні ребра, простий граф, мультиграф, маршрут у графі, відкритий маршрут, замкнутий маршрут, ланцюг, шлях, цикл, зв’язний граф, вага вершини, вага ребра, зважений граф.
2. Назвіть способи представлення графів в комп’ютерній програмі. Які переваги має кожний із цих способів?
3. Дайте означення дерева, орієнтованого дерева, кореневого дерева, термінальних вершин, нетермінальних вершин.
4. Чому в математиці і близьких їй науках дерева "ростуть" вниз головою?
5. Що таке предок, нащадок, висота кореневого дерева, каркас?

Розділ 14. Алгоритми на графах. Алгоритми обходу графа

Існує багато алгоритмів на графах, в основі яких лежить систематичний перебір вершин графа, такий що кожна вершина проглядається (відвідується) у точності один раз. Тому важливою задачею є знаходження гарних методів пошуку в графі.

Під **обходом графів (пошуком на графах)** розуміється процес систематичного перегляду всіх ребер або вершин графа з метою відшукання ребер або вершин, що задовольняють деякій умові.

При рішенні багатьох задач, що використовують графи, необхідні ефективні методи регулярного обходу вершин і ребер графів. До стандартних й найпоширеніших методів відносяться:

- пошук у глибину (Depth First Search, DFS);
- пошук у ширину (Breadth First Search, BFS).

Ці методи найчастіше розглядаються на орієнтованих графах, але вони можуть застосовуватися і для неорієнтованих, ребра яких вважаються двоспрямованими. Алгоритми обходу в глибину і в ширину лежать в основі рішення різних задач обробки графів, наприклад, побудови каркасного лісу, перевірки зв'язності, ациклічності, обчислення відстаней між вершинами тощо.

Пошук у глибину

При пошуку в глибину відвідується перша вершина, потім необхідно йти уздовж ребер графа, до потрапляння в глухий кут. Вершина графа є *глухим кутом*, якщо всі суміжні з нею вершини вже відвідані. Після влучення в глухий кут потрібно вертатися назад уздовж пройденого шляху, поки не буде виявлена вершина, у якої є суміжна ще не відвідана

вершина, а потім необхідно рухатися в цьому новому напрямку. Процес є завершеним при поверненні в початкову вершину, причому всі суміжні з нею вершини вже повинні бути відвідані.

Таким чином, основна ідея пошуку в глибину – коли можливі шляхи по ребрах, що виходять із вершин, розгалужуються, потрібно спочатку повністю досліджувати одну гілку й тільки потім переходити до інших гілок (якщо вони залишаться нерозглянутими).

Алгоритм пошуку в глибину

Крок 1. Всім вершинам графа привласнюється значення «не відвідана». Вибирається перша вершина й позначається як відвідана.

Крок 2. Для останньої позначеної як відвідана вершини вибирається суміжна вершина, що є першою позначеною як не відвідана, і їй привласнюється значення відвідана. Якщо таких вершин немає, то береться попередня позначена вершина.

Крок 3. Повторити крок 2 доти, поки всі вершини не будуть позначені як відвідані (рис. 14.1).

Псевдокод алгоритму пошуку в глибину

```
function DepthFirstSearch(int n, int **Graph, bool
*Visited, int Node)
 begin
    visited[Node] = true
    write(Node + 1)

    for int i = 0 ; i < n ; i++
        if Graph[Node][i] And (Not Visited[i])
            DepthFirstSearch(n, Graph, Visited, i)

 end
```

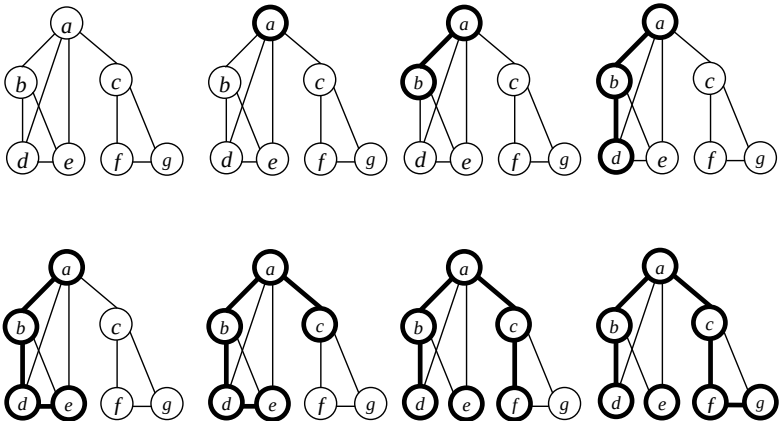

Рисунок 14.1 – Демонстрація алгоритму пошуку в глибину

Також часто використовується нерекурсивний алгоритм пошуку в глибину. У цьому випадку рекурсія заміняється на стек. Як тільки вершина переглянута, вона поміщається в стек, а використаною вона стає, коли більше немає нових вершин, суміжних з нею.

Часова складність залежить від представлення графа. Якщо застосовано матрицю суміжності, то часова складність дорівнює $O(n^2)$, а якщо нематричне представлення – O(n+m): розглядаються всі вершини й всі ребра.

Пошук в ширину

При пошуку в ширину, після відвідування першої вершини, відвідуються всі сусідні з нею вершини. Потім відвідуються всі вершини, що перебувають на відстані двох ребер від початкової. При кожному новому кроці відвідуються вершини, відстань від яких до початкової на одиницю більша попередньої. Щоб запобігти повторному відвідуванню вершин, необхідно вести список відвіданих вершин. Для зберігання тимчасових даних, необхідних для

роботи алгоритму, використовується черга – упорядкована послідовність елементів, у якій нові елементи додаються в кінець, а старі видаляються із початку.

Таким чином, основна ідея пошуку в ширину полягає в тому, що спочатку досліджуються всі вершини, суміжні з початковою вершиною (вершина з якої починається обхід). Ці вершини перебувають на відстані 1 від початкової. Потім досліджуються всі вершини на відстані 2 від початкової, потім всі на відстані 3 і так далі. При цьому для кожної вершини відразу визначається довжина найкоротшого маршруту від початкової вершини.

Алгоритм пошуку в ширину

Крок 1. Всім вершинам графа привласнюється значення не відвідана. Вибирається перша вершина й позначається як відвідана (і заноситься в чергу).

Крок 2. Відвідується перша вершина із черги (якщо вона не позначена як відвідана). Всі її сусідні вершини заносяться в чергу. Після цього вона видаляється із черги.

Крок 3. Повторюється крок 2 доти, поки черга не є порожньою (рис. 14.2).

Псевдокод алгоритму пошуку в ширину

```
void BreadthFirstSearch(int n, int **Graph,
                        bool *Visited, int Node)
 begin
    int *List = new int[n] //черга
    int Count, Head        // вказівники черги
    int i
    for i = 0; i < n ; i++
        List[i] = 0 // початкова ініціалізація
    Count = Head = 0
  // розміщення в чергу вершини Node
    List[Count++] = Node
    Visited[Node] = true
    while (Head < Count)
```

```
    begin
       //взяття вершини із черги
       Node = List[Head++]
       write(Node + 1)
    //перегляд всіх вершин, в'язаних з вершиною Node
       for i = 0 ; i < n ; i++
         // якщо вершина раніше не переглянута
         if (Graph[Node][i] && !Visited[i])
         begin
           // заносимо її в чергу
           List[Count++] = i
           Visited[i] = true
         end
     end
end
```

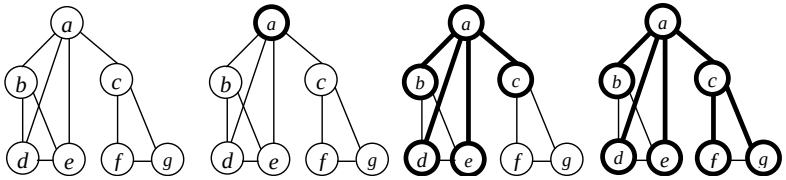

Рисунок 14.2 – Демонстрація алгоритму пошуку в ширину

Складність пошуку в ширину при нематричному представленні графа дорівнює $O(n + m)$, тому що розглядаються всі n вершин і m ребер. Використання матриці суміжності приводить до оцінки $O(n^2)$.

Питання до розділу:

1. Що розуміють під терміном "обхід графу" ?
2. В чому полягає основна ідея методу пошуку в глибину?
3. Поясніть алгоритм пошуку в глибину?
4. В чому полягає основна ідея методу пошуку в ширину?
5. Поясніть алгоритм пошуку в ширину?