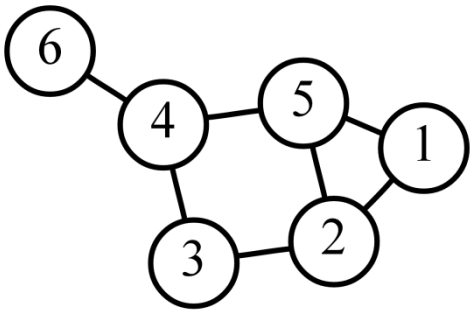
Подання неорієнтованих графів списками суміжності

Навчальний матеріал

Данило Мисак, 21.05.2013

Що таке списки суміжності

Припустімо, в нас є граф:



Як його можна зберегти і як із ним оперувати у програмі мовою Pascal або C++? Мабуть, найбільш простий і звичний спосіб — подати граф таблицею, або, як її називають, — матрицею суміжності. Якщо у графі n вершин, то матриця складатиметься з n рядків та n стовпців. На перетині i-го рядка та j-го стовпця стоятиме або нуль, або одиниця: одиниця, якщо між вершинами i та j графа є ребро, або нуль, якщо ребра немає. Наш граф, поданий матрицею суміжності, виглядатиме таким чином:

	1	**2**	**3**	**4**	**5**	**6**
1	0	1	0	0	1	0
2	1	0	1	0	1	0
3	0	1	0	1	0	0
4	0	0	1	0	1	1
5	1	1	0	1	0	0
6	0	0	0	1	0	0

Зверніть увагу, що на тій діагоналі матриці, яка починається у верхньому лівому куті і закінчується у правому нижньому, стоять нулі. Символічно це означає, що з вершини графа у саму цю вершину ребра не проведено. Крім того, матриця симетрична відносно своєї діагоналі. Для збереження матриці в програмі нам знадобиться двовимірний масив $n \times n$ із комірками числового або булевського типу.

Утім, є й інший спосіб подати граф — за допомогою списків суміжності. Для кожної вершини ми будемо зберігати лише номери вершин, суміжних із даною (тобто тих, які сполучені з даною вершиною ребром). Наш граф набуває такої форми:

1	2	5	
2	1	3	5
3	2	4	
4	3	5	6
5	1	2	4
6	4		

У програмі списки суміжності можна реалізувати як масив із n динамічних масивів або знову-таки як двовимірний масив $n \times n$. В останньому випадку для кожної вершини графа доведеться окремо зберігати кількість суміжних із нею вершин — щоб знати, які саме комірки двовимірного масиву зберігають потрібні номери. До того ж, на відміну від випадку з матрицею суміжності, масиви неодмінно повинні бути числовими.

Для чого потрібні списки суміжності

У випадку, якщо у графі багато ребер, списки суміжності на практиці дають мало переваг порівняно з матрицею суміжності. Зате якщо в графі відносно небагато ребер, то списки суміжності часто виявляються кориснішими й ефективнішими. Наприклад, як видно з попереднього розділу, для збереження списку суміжності може знадобитися менше комірок у таблиці, а значить, — менше пам'яті.

Справді: якщо зберігати граф матрицею суміжності, то незалежно від кількості ребер у графі нам знадобиться рівно n^2 комірок пам'яті (де n, як і раніше, позначає кількість вершин графа). Уже для значень n порядку $10\,000$ виділення такої кількості пам'яті може виявитися проблематичним. Разом із тим у списку суміжності ми фактично зберігаємо лише ребра графа. Щоправда, по два рази кожне: один раз для одного кінця ребра і вдруге — для іншого кінця. Приміром, ребро між вершинами 1 і 5 у нашому графі зберігається у першому рядку списку суміжності (число 5) і в п'ятому рядку (число 1). Таким чином, якщо у графі є m ребер, то, щоб його зберегти, нам знадобиться $2m$ комірок. Давайте порівняємо кількість комірок, потрібних на зберігання нашого графа: $6 \cdot 6 = 36$ для матриці суміжності і $2 \cdot 7 = 14$ для списків суміжності. Виграш суттєвий!

Ще одна вагома причина використовувати при розв'язанні великої кількості задач списки суміжності — ефективність перебору вершин, суміжних із даною. Якщо, наприклад, нам потрібно здійснити на графі т. зв. пошук у ширину або в глибину, то на кожному кроці пошуку ми якраз перебираємо всі вершини, сполучені ребром із вершиною, розглянутою на попередньому кроці. Якщо користуватися матрицею суміжності, то для виконання такої операції потрібно пройти весь рядок, що складається з n вершин, шукаючи в цьому рядку одиниці. Якщо ж користуватися списками суміжності, то ми відразу маємо потрібні списки суміжних вершин. У випадку, коли кількість ребер, які виходять з даної вершини, значно

менша від загальної кількість вершин, такий підхід суттєво скорочує перебір і пришвидшує виконання програми.

У той же час принаймні в одному аспекті списки суміжності програють матриці. Якщо, використовуючи матрицю суміжності, для будь-яких двох вершин можна миттєво визначити, чи наявне між ними ребро, то у випадку зі списками суміжності так не вийде. Доведеться брати одну з вершин і проходити по списку всіх вершин, суміжних із нею, в пошуках другої вершини. З огляду на це списки суміжності не можна назвати однозначно «кращою» структурою, ніж матриця суміжності. Для різних класів задач можна застосовувати різні способи подання графа, а для деяких задач їх можна навіть комбінувати, використовуючи обидва способи відразу.

Як оперувати зі списками суміжності

Розгляньмо приклад реалізації списків суміжності мовою Pascal. Нехай задано число $maxN$, що його не перевищуватиме кількість вершин графа. Тоді списки суміжності можна подати масивом динамічних масивів

```
a: array[1..maxN] of array of integer;
```

Нам також потрібно пам’ятати кількість вершин у графі:

```
n: integer;
```

Зауважимо, що, хоч динамічні масиви у мові Pascal мають індексацію з нуля, індексація вершин у графі все одно починається з одиниці, тобто вершини занумеровано числами від 1 до n включно.

Ініціалізація списків суміжності полягатиме у присвоєнні змінній n справжньої кількості вершин графа, а також у спустошенні всіх списків (на випадок, якщо сам Pascal їх раптом не спустошить):

```
for i := 1 to n do setLength(a[i], 0);
```

Додавання ребра між вершинами u та v передбачає два кроки: спершу ми додаємо вершину v до списку суміжності вершини u, а далі додаємо u до списку суміжності v (або у зворотному порядку):

```
setLength(a[u], length(a[u]) + 1);  // Збільшуємо розмірність
списку суміжності вершини u
a[u][length(a[u]) - 1] := v;  // Додаємо v до списку вершини u

setLength(a[v], length(a[v]) + 1);  // Збільшуємо розмірність
списку суміжності вершини v
a[v][length(a[v]) - 1] := u;  // Додаємо u до списку вершини v
```

Зверніть увагу, що порядок, в якому лежатимуть номери у списку кожної з вершин, збігатиметься з порядком, у якому їх туди додавали. Тому на виході ми не обов’язково

отримаємо відсортовані від меншого до більшого номери. Інакше кажучи, ми можемо одержати результат, подібний до такого:

1	5	2	
2	5	1	3
3	2	4	
4	6	5	3
5	1	4	2
6	4		

Якщо ж конче необхідно одержати відсортовані списки, їх можна відсортувати вручну вже після заповнення.

Проходження по списку вершин, суміжних із заданою вершиною u, здійснюється за допомогою простого циклу:

```
for i := 0 to length(a[u]) - 1 do begin
    v := a[u][i];  // Номер чергової суміжної з u вершини
    ...            // Робимо потрібні операції з вершиною v
end;
```

Вилучення ребра між вершинами u та v може знадобитися під час розв'язання певних задач, де граф динамічно модифікують. Дану операцію зі списками суміжності здійснити дещо важче, ніж інші. Для цього нам потрібно знайти вершину v у списку суміжності вершини u, записати на її місце останній елемент зі списку і скоротити кількість елементів на 1, а потім аналогічні дії провести зі списком вершини v:

```
for i := 0 to length(a[u]) - 1 do
    if a[u][i] = v then begin  // Якщо ми знайшли вершину v
        a[u][i] := a[u][length(a[u]) - 1];  // Записуємо на її
місце останню вершину зі списку суміжності вершини u
        setLength(a[u], length(a[u]) - 1);  // Після цього
видаляємо останній елемент списку
        break;  // Перебирати далі не потрібно — виходимо
    end;
// А тепер те саме робимо для списку суміжності вершини v:
for i := 0 to length(a[v]) - 1 do
    if a[v][i] = u then begin
        a[v][i] := a[v][length(a[v]) - 1];
        setLength(a[v], length(a[v]) - 1);
        break;
    end;
```

Приклади застосування

Найпростіший приклад застосування списків суміжності — знаходження кількості вершин, суміжних із заданою вершиною u:

```
answer := length(a[u]);
```

Звичайно, це не єдина перевага в ефективності, яку забезпечує використання списків суміжності. Наприклад, як уже було сказано вище, саме на їх базі зручно реалізовувати пошук у ширину або в глибину. Частковим випадком задачі пошуку на графі є знаходження всіх вершин, віддалених від даної вершини u щонайбільше на два ребра. Інакше кажучи, потрібно знайти всі вершини, куди можна потрапити із заданої вершини, зробивши не більше ніж два переходи по ребрах. Використання списку суміжності дозволяє ефективно розв'язати цю задачу.

Спершу заведемо булевський масив на $maxN$ елементів, у якому зберігатимемо інформацію про те, в які ж вершини нам вдалося потрапити не більше ніж за два кроки.

```
b: array[1..maxN] of boolean;
```

Перед запуском основної частини алгоритму нам потрібно ініціалізувати масив — заповнити його значеннями, що відповідають тимчасовій відсутності вершин, куди б ми вже встигли потрапити:

```
for i := 1 to n do b[i] := false;
```

Але вершину u, з якої починається шлях, позначимо як уже досягнуту:

```
b[u] := true;
```

Тепер перейдімо до основної частини алгоритму. Вона зводиться до перебору всіх вершин, суміжних з u, а для кожної суміжної вершини — всіх вершин, суміжних із нею самою. При цьому, щойно ми натрапляємо на деяку вершину, заносимо інформацію про неї в масив b.

```
for i := 0 to length(a[u]) - 1 do begin
// Перебираємо всі вершини, суміжні з u
    v := a[u][i];  // Чергова суміжна з u вершина
    b[v] := true;  // Запам'ятовуємо вершину як досягнуту
    for j := 0 to length(a[v]) - 1 do begin
    // Перебираємо всі вершини, суміжні з v
        w := a[v][j];  // Чергова суміжна з v вершина
        b[w] := true;  // Запам'ятовуємо вершину як досягнуту
    end;
end;
```

Залишається опрацювати результат, наприклад вивести досягнуті вершини графа:

```
for i := 1 to n do
    if b[i] then writeln(i);  // Якщо вершина досяжна, виводимо її
```

Слід зауважити, що відразу виводити номери досягнутих вершин замість того, щоб модифікувати масив b, не можна: інакше номери деяких вершин, можливо, будуть виведені по кілька разів.

Якби граф було задано матрицею суміжності, ми також могли б розв'язати дану задачу, але, щоб знайти відповідь, програмі знадобилося б зробити більше дій, а отже, витратити більше часу.

Підсумок

Списки суміжності є нескладним способом подання графа, альтернативним до використання матриці суміжності. Цей спосіб має певні переваги, такі як ефективніше використання пам'яті і швидший обхід суміжних вершин, але має також і недоліки.

Списки суміжності і такі операції над ними, як додавання й вилучення ребер із графа та проходження по суміжних вершинах, можна реалізувати на базі динамічних масивів.

Списки суміжності доцільно використовувати при розв'язанні широкого класу задач, зокрема в задачах пошуку на графі.