**Машине представлення графів. Базові алгоритми**

***Позначення:***

 ***для орієнтованого графа***,

 ***для неорієнтованого графа***.

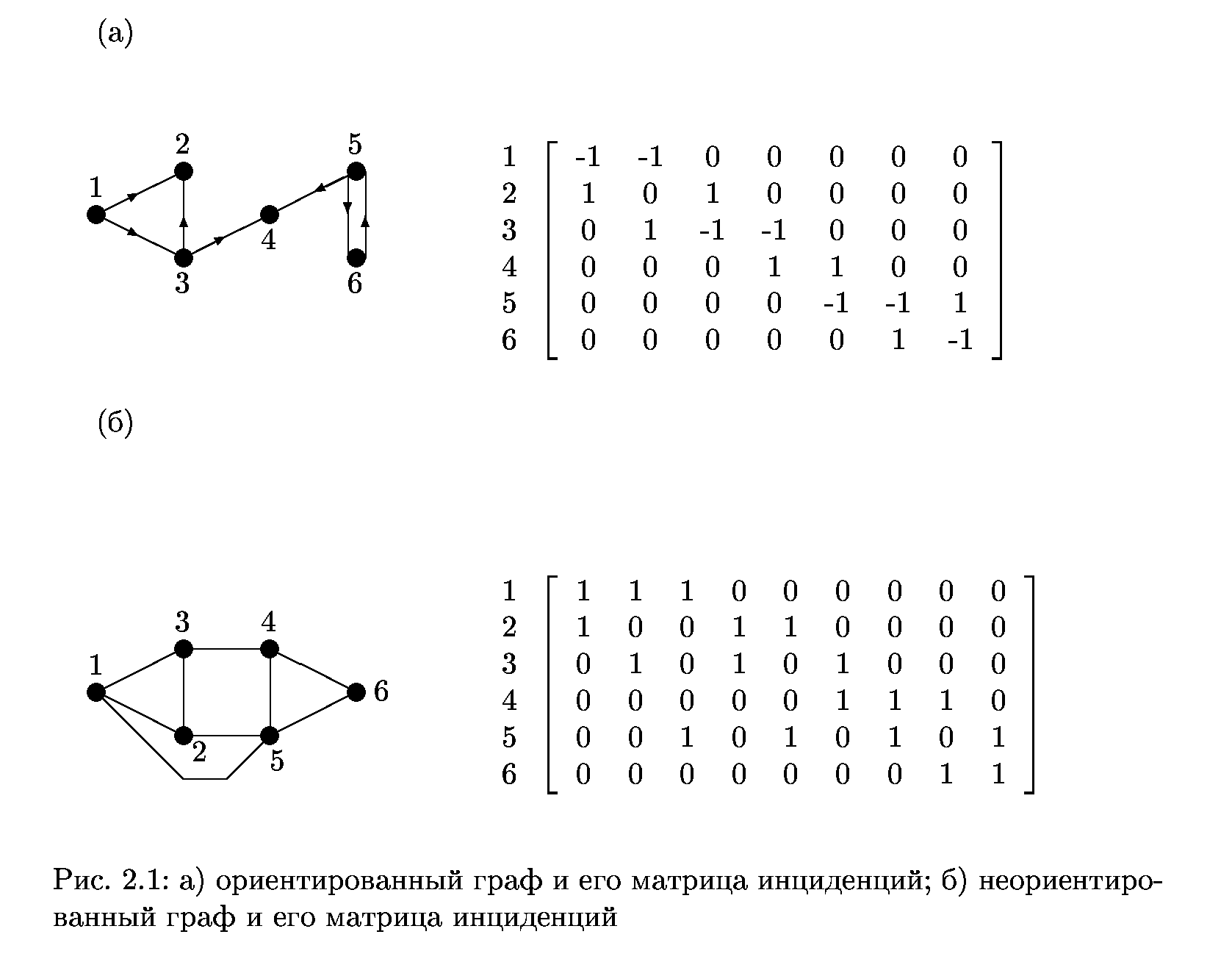
.

Зображення графа на площині у вигляді точок (вершин графа) та з’єднуючих їх ліній (ребер графа), яке ми використовуємо при роз’язанні «вручну» задач на графи, повністю не придатне при ргозв’язанні задач на графи за допомогою ЕОМ.

Вибір способу представлення графа у ЕОМ тісно пов’язаний з ефективністю алгоритмів, кожен спосіб має переваги і недоліки.

**1-й спосіб – матрицею інцидентності** – прямокутна матриця с *п* рядками (відповідає кількості вершин) та  стовпцями (відповідає кількості ребер).

Для орієнтованого графа в *(х,у)-*стовпці в рядку ***х*** стоїть -1, в рядку ***у*** стоїть 1, всі решта елементів дорівнюють *0*. Для неорієнтованого графа в *(х,у)-*стовпці в рядках *х* та *у* стоять 1, всі решта елементів дорівнюють *0*.



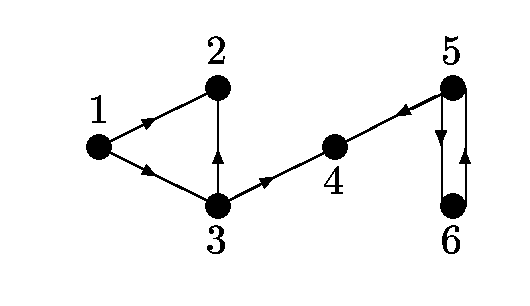
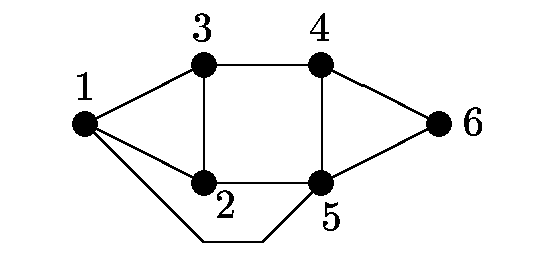
З алгоритмічної точки зору матриця інциденцій є, мабуть, найгіршим способом представлення графа. По-перше, він вимагає mn комірок пам’яті, причому більшість з них зайнята нулями. Не дуже зручним є також доступ до інформації. Щоб відповісти на просте питання «чи існує дуга (ху)?» треба у гіршому випадку перебрати всі стовпці матриці.

Але все ж для деяких задач цей спосіб використовується.

**2-й спосіб**

Більш кращим є представлення графа **матрицею суміжності** – квадратною -матрицею , де , якщо існує ребро  і  у противному випадку (для орієнтованого графа). У випадку неорієнтованого графа ця матриця симетрична, тому що наявність ребра  передбачає також наявність ребра  (див. рис.2.2).

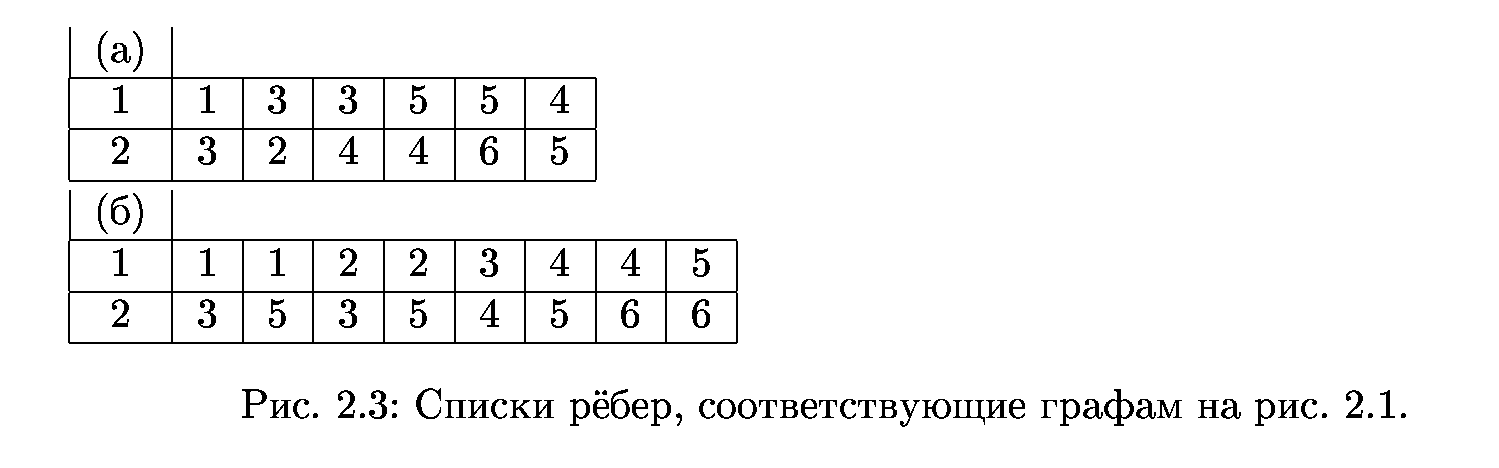


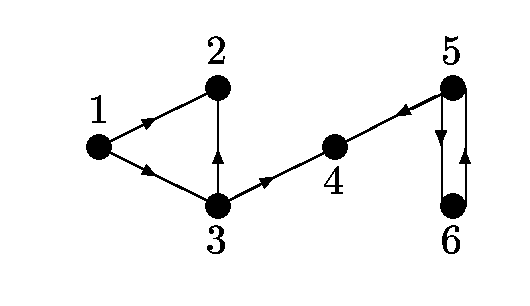
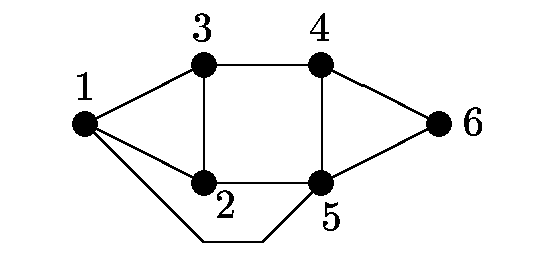
 Рис 2.1 

Основною перевагою матриці суміжності є той факт, що лише за один крок можна отримати відповідь на питання «**чи існує дуга (ху)? (або ребро** **)». Недолік** полягає в тому, що незалежно від числа ребер об’єм зайнятої пам’яті складає . Один із шляхів подолання для «невеликих *п*» – зберігати цілий рядок (або стовпчик) матриці в одному машинному слові.

**3 спосіб**

Більш економно (особливо у випадку «неплотних графів» - коли  значно менше за ) є **метод представлення** **графа** за допомогою **списка пар**, які відповідають ребрам графа. Об’єм пам’яті складає 2m.



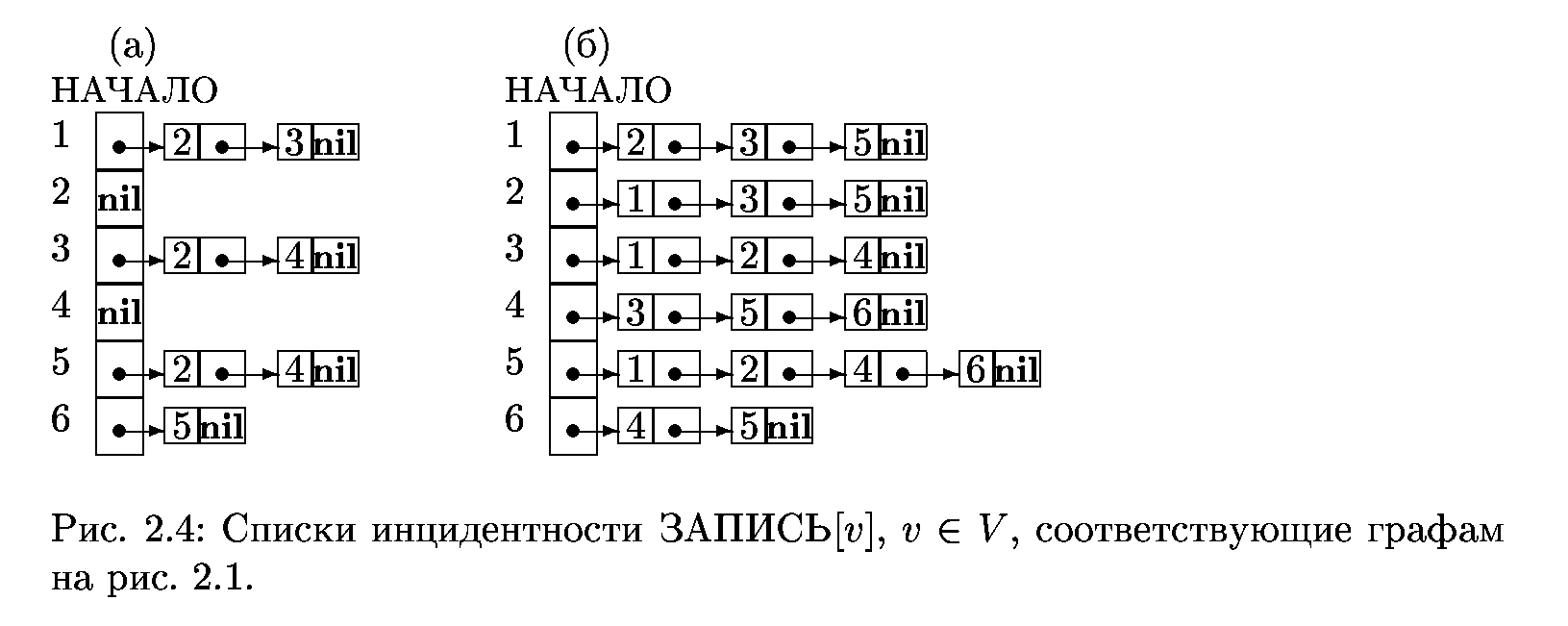
 Рис 2.1 

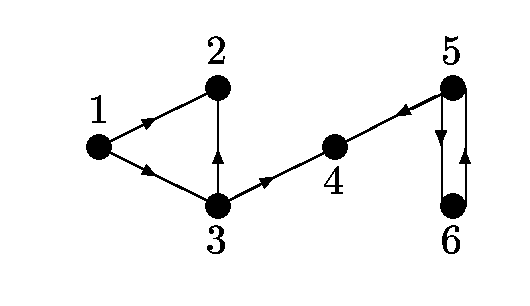
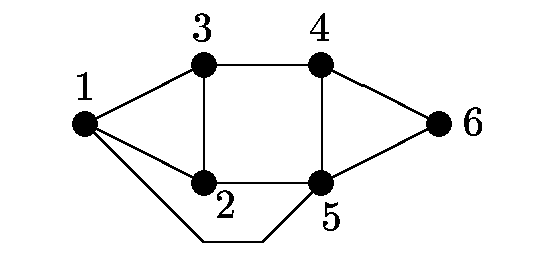
**Недоліком** є велика кількість кроків – порядку m у гіршому випадку – необхідних для отримання множини вершин, до яких ідуть ребра з данної вершини. Ситуація значно покращиться, якщо впорядкувати множину пар лексикографічно й застосувати двійковий пошук.

**4 спосіб**

Найкращим у багатьох випадках є структура даних, яку називають **списками інцидентності.**

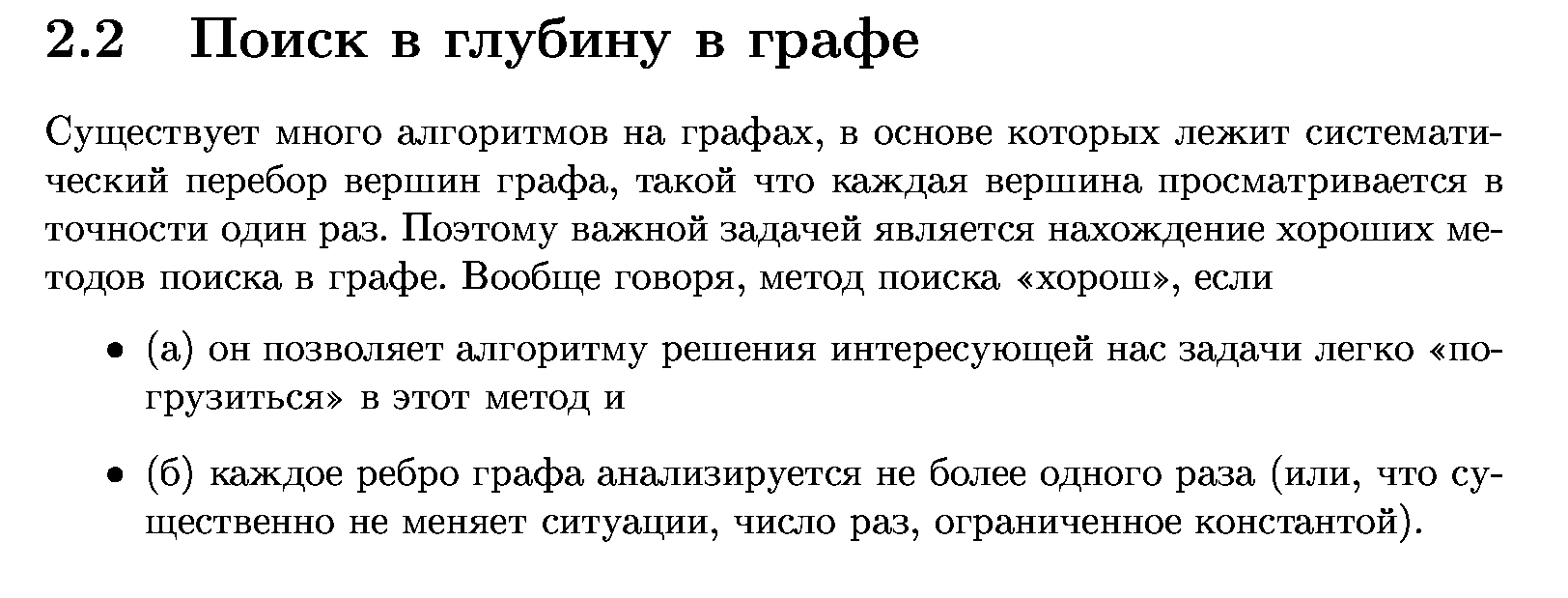
Ця структура для кожної вершини  містить список таких вершин , що  (або  для неорієнтованого графа).

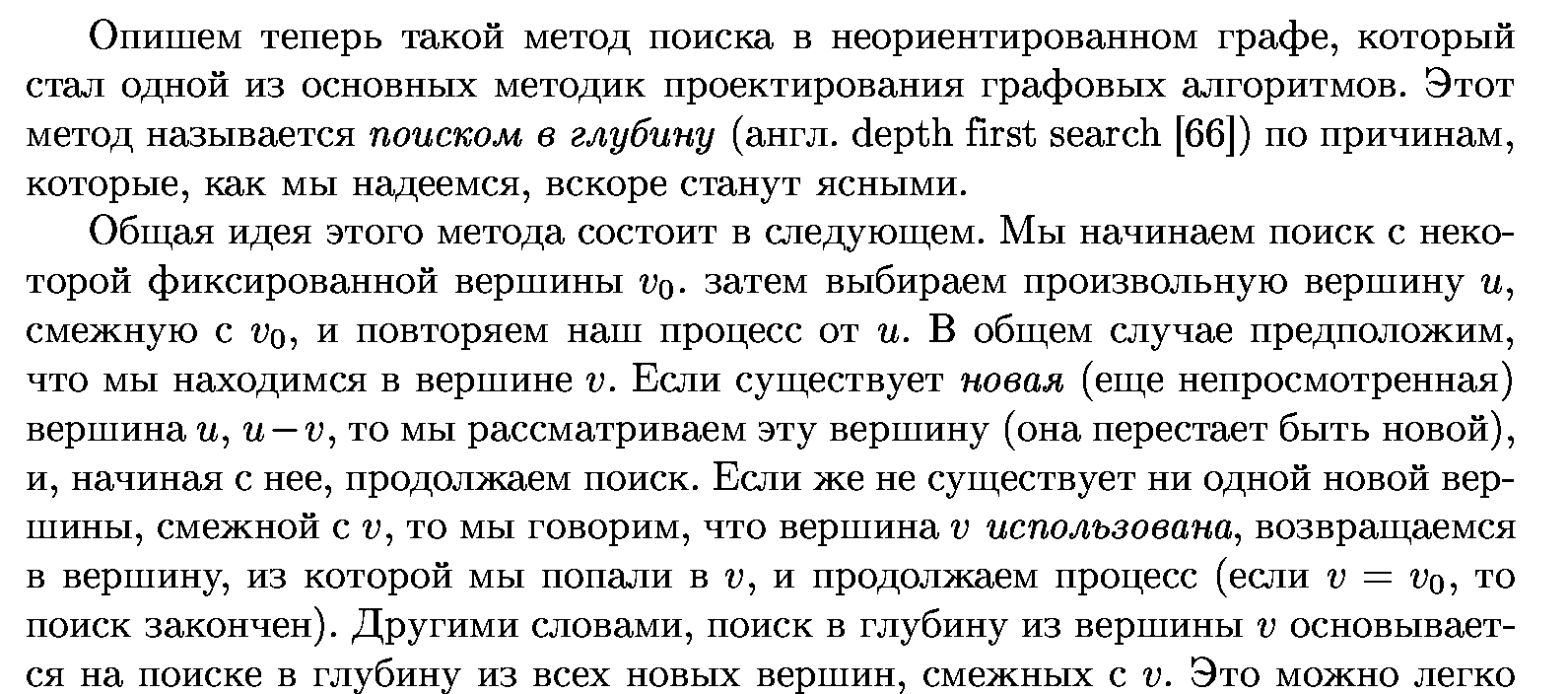


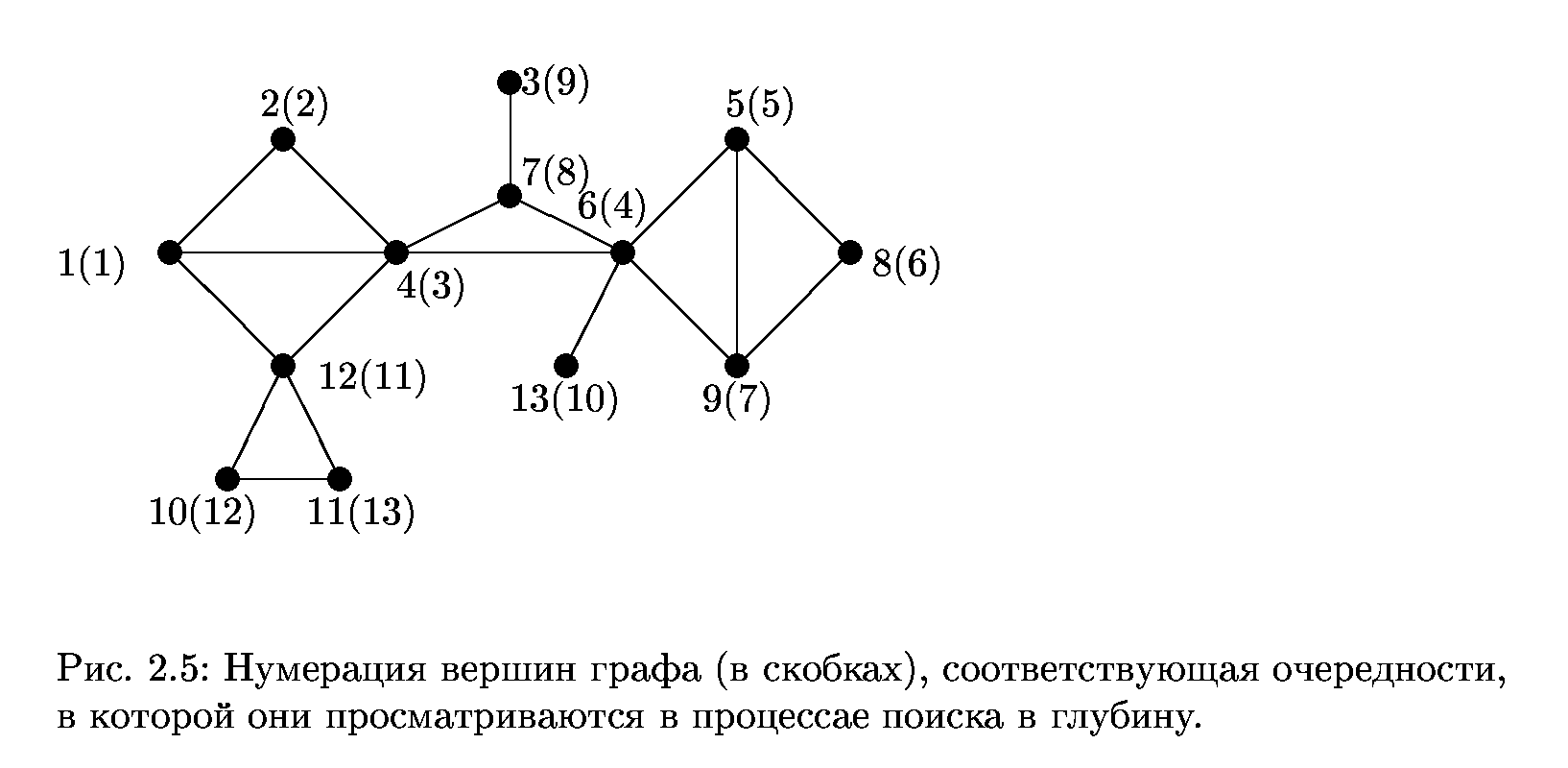
 Рис 2.1 

Число комірок пам’яті, необхідне для представлення графа списками інцидентності дорівнює m+n.

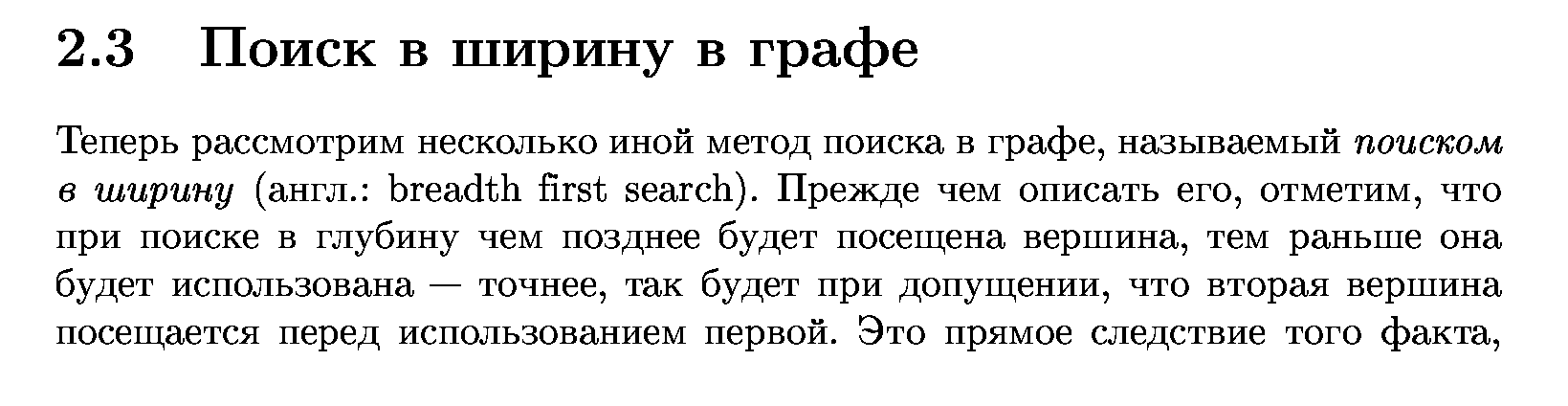
**БАЗОВІ АЛГОРИТМИ НА ГРАФАХ:**

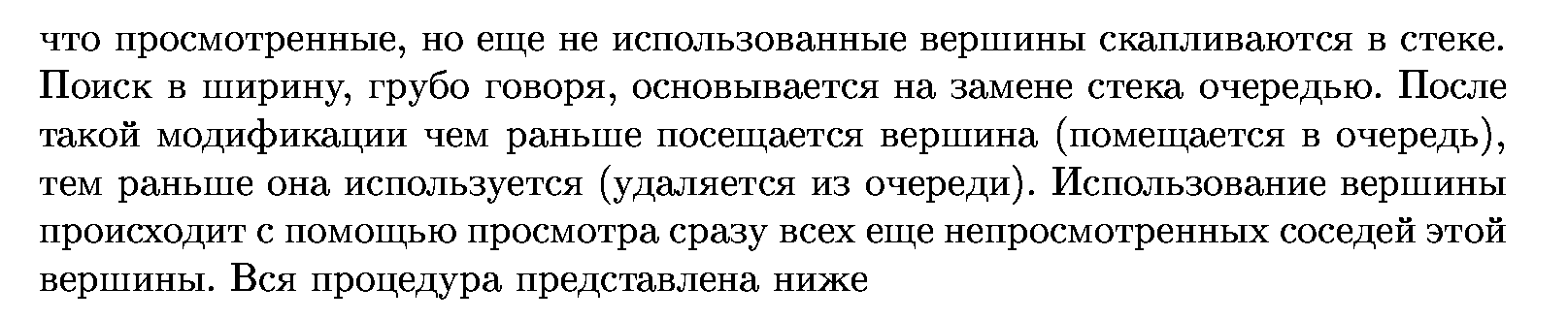


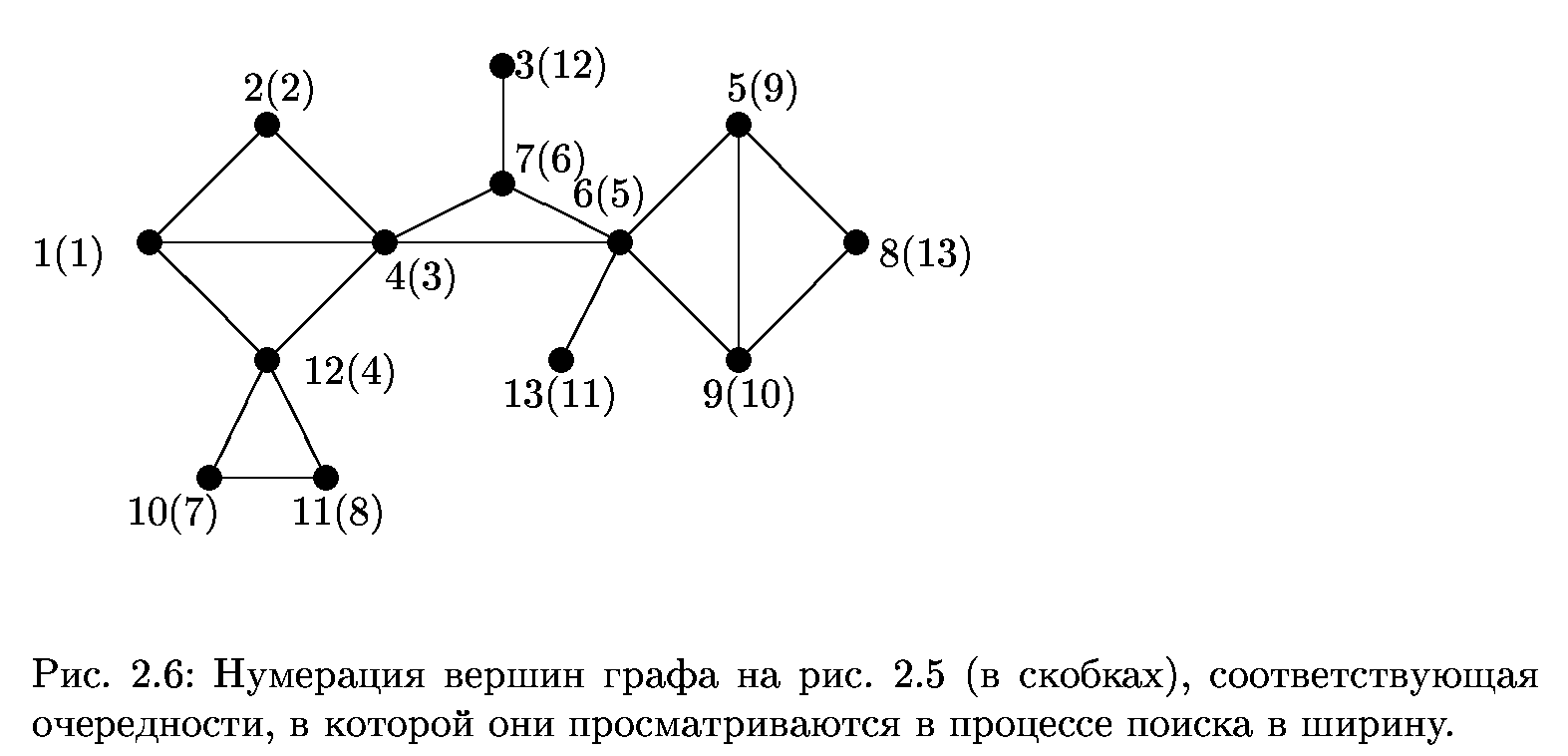




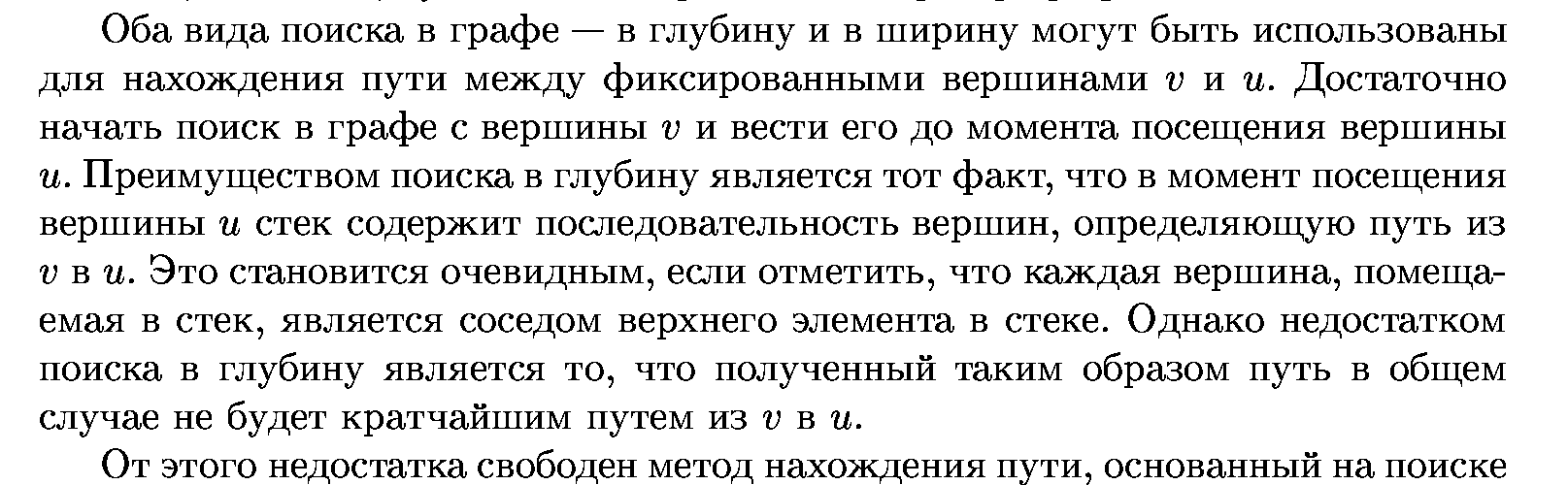
При пошуку в глибину в якості активної вибирається та з відкритих вершин, яка була відвідана останньою. Найбідьш зручною структурою для збереження множини відкритих вершин є **стек**: вершини, що відкриваються, складаються в стек в тому порядку, в якому вони відвідуються, а в якості активної вибирається остання вершина.







При пошуку в ширину в якості активної вибирається та з відкритих вершин, яка була відвідана раніше інших. Зручно використовувати чергу для зберігання відкритих вершин. Після відвідування вершини, вона помічається (стає не новою) й поміщається в чергу.



в ширину.

**Топологичне сортування**

Задача топологічного сортування графа полягає в тому, щоб вказати такий лінійний порядок на його вершинах, щоб будь-яке ребро вело від вершини з меншим номером до вершини з більшим номером. Очевидно, що коли в графі є цикли, то такого порядку не існує.

**Топологічне сортування вершин орграфа** *G=(V,E)* заключається в присвоюванні його вершинам номерів *1,..,n* таким чином, щоб для будь-якої дуги цього **графа** виконувалась умова: *(Vi,Vj)єE* при *i<j*.

Це можливо в тому випадку, коли **граф** не має **контурів**, є **ациклічним**.

**Топологичне сортування** може розглядатись як процес знаходження лінійного порядка на множині вершин, в який може бути вкладеним частковий порядок, заданий множиною дуг. Топологичне сортування починається зі знаходження вершин, з якої не виходить жодної дуги. Така вершина завжди існує, якщо в графі немає контурів. Їй присвоюється **найбільший номер** *п* і вона видаляється з графа разом з дугами, які входять в цю вершину. **Граф**, що залишився, також не має **контурів**. Процес повторюється й новій вершині, з якої не виходять дуги, присвоюється найбільший номер (*n-1*) і т.д. Розглянутий підхід неважко реалізувати на матриці суміжності, але це потребує *n2* операцій. Трудоємність можна зменшити, якщо виключити дії по знаходженню вершин, з яких не виходять дуги (ці дії повторюються). Це можна досягти п**ошуком в глибину**. Для **ациклічного графа** ця процедура забезпечує послідовний вибір вершин, що не мають вихідних дуг.

**Алгоритм топологической сортировки**

Вычислим входящую степень для каждой вершины. Вершины с нулевой входящей степенью занесем в очередь. Пока очередь не пуста, достаем вершину из очереди и заносим в конец строящегося топологического порядка. Для каждой вершины *v*, извлеченной из очереди, моделируем удаление всех выходящих из нее дуг (*v*, *u*). То есть для каждой такой дуги входящую степень вершины *u* следует уменьшить на единицу. Если после этого уменьшения входящая степень вершины *u* стала равной нулю, то помещаем *u* в очередь. Алгоритм работает, пока очередь не станет пустой. Если все вершины были помещены в очередь, то построен топологический порядок. Иначе после удаления некоторых вершин получим граф, в котором нет вершин входящей степени ноль. А это возможно лишь в том случае, когда в графе присутствует цикл. И тогда топологического упорядочивания не существует.

**Решение задачи топологической сортировки при помощи поиска в глубину**

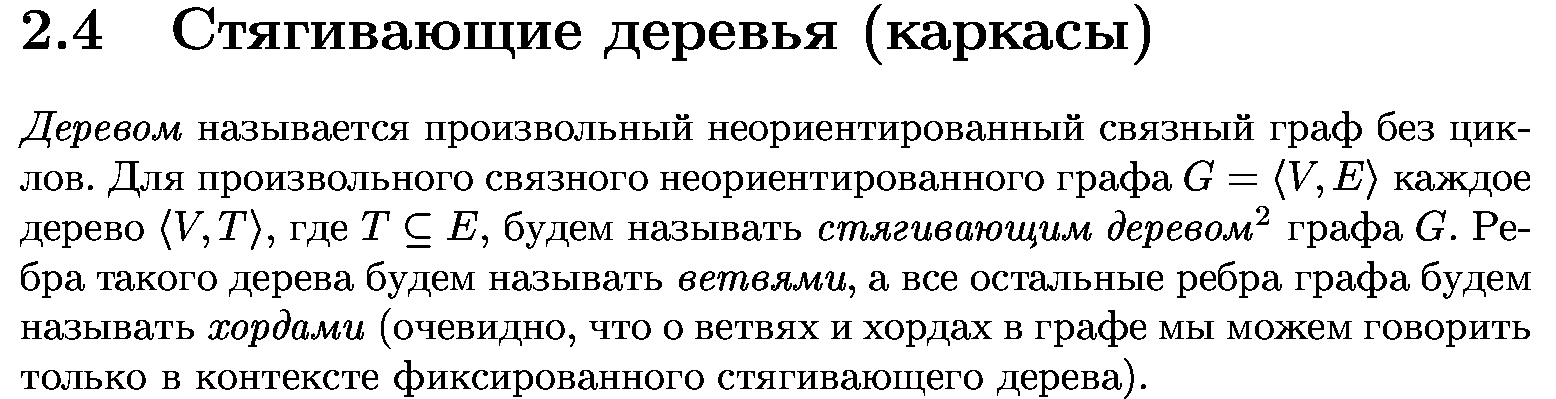
Задача топологической сортировки решается и при помощи поиска в глубину. Изначально все вершины являются белыми. Когда обход в глубину входит в вершину, она становится серой. Когда обработка вершины завершается, она становится черной. Порядок вершин при топологической сортировке соответствует порядку, обратному тому, в котором вершины принимают черный цвет.

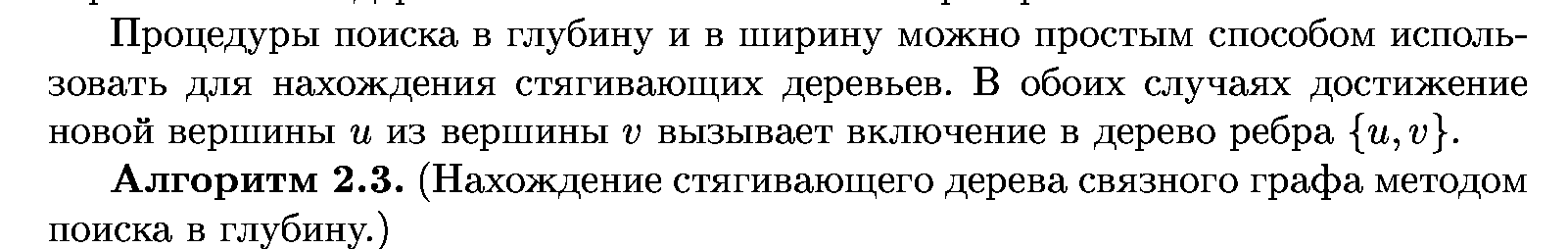
**Время работы** алгоритма топологической сортировки равно времени обхода всех вершин графа в глубину, то есть O(*n* + *m*).

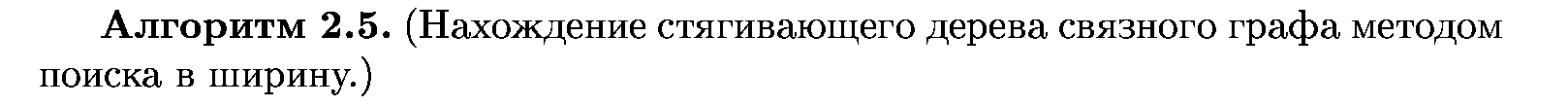
**Пример.** Запустим обход в глубину на графе. Возле каждой вершины *v* отметим метки *d[v] / f[v].* Для определения топологической сортировки следует отсортировать вершины графа по убыванию меток *f[v].*

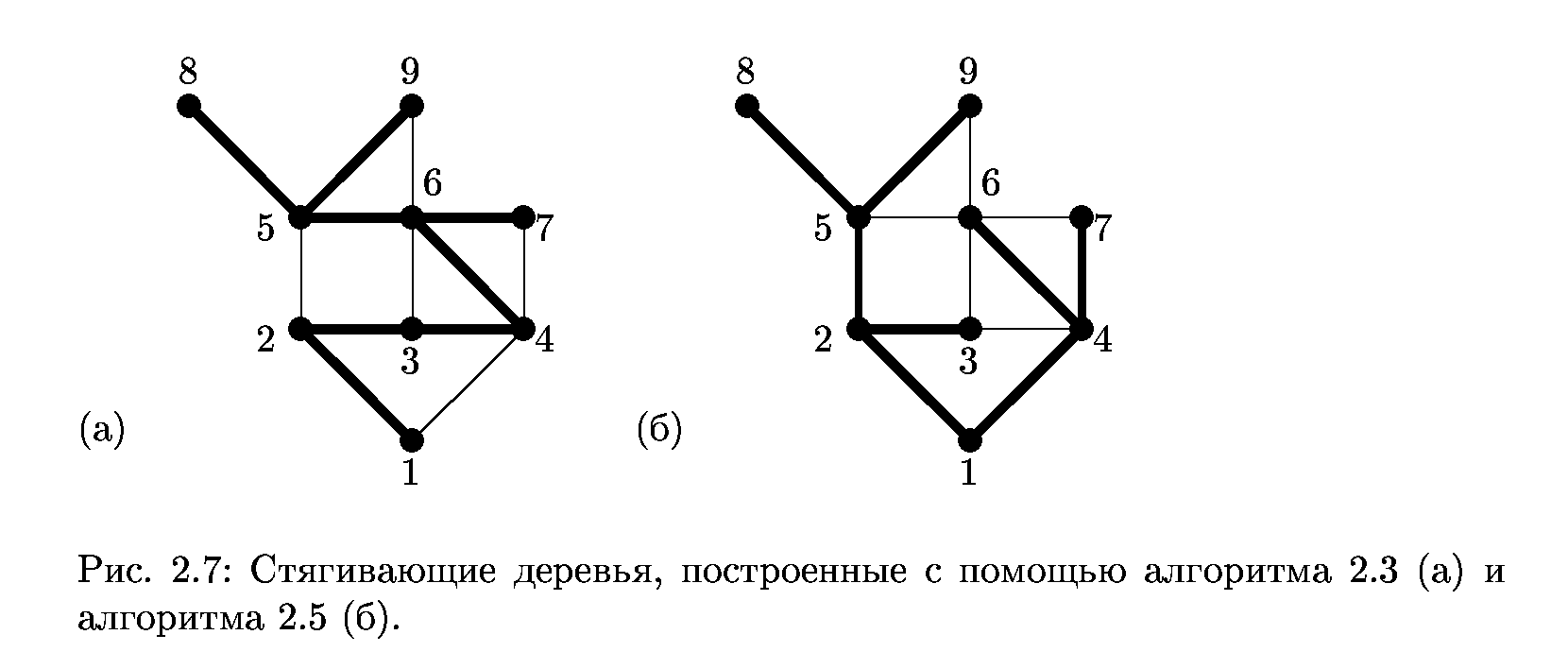


Первой покрашенной в черный цвет будет вершина с номером 1. Она будет последней при топологической сортировке. Второй покрашенной в черный цвет будет вершина 6. Последней будет вершина 2.





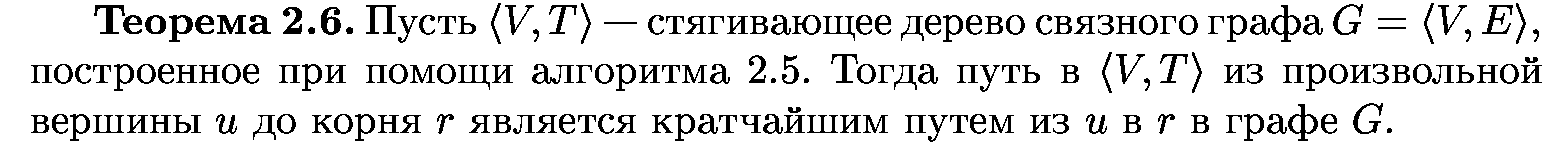


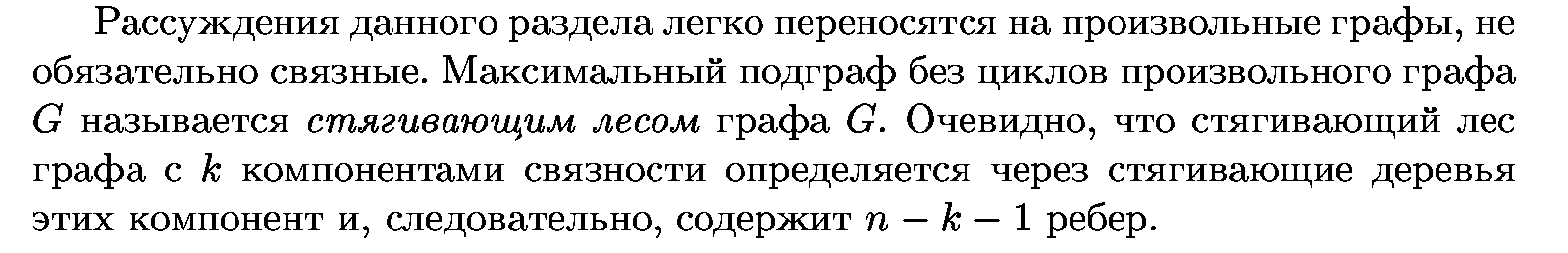


(рис.2.7а) **Порядок** просматривания вершин при построении стягивающего дерева «в глубину»: **1,2,3,4,6,7,5,8,9.**

(рис.2.7б) **Пример** записи очереди (в скобках). **Порядок** (жирным) просматривания вершин при построении стягивающего дерева «в ширину»:

1(2,4),**1,**2(3,5),**2**,4(6,7),**4,3,**5(8,9),**5**,**6**,**7,8,9**





[**http://intuit.valrkl.ru/course-147/index.html#ID.12.lecture**](http://intuit.valrkl.ru/course-147/index.html#ID.12.lecture)

**(Костюкова Н.И. Комбинаторные алгоритмы для программистов)**