

АЛГОРИТМИ ТА МЕТОДИ ОБЧИСЛЕНЬ

д.т.н., проф. Новотарський Михайло Анатолійович

<http://amodm.pp.ua>

Login: student_new

Password: : pbhDcMq3Z

Старостам груп здати списки студентів

СКЛАД КУРСУ

1. Алгоритми. Основні поняття (4 годин)
2. Похибки обчислень (2 години)
3. Методи обчислень (24 години)
4. Основи теорії алгоритмів (6 годин)

Значення рейтингу кредитного модуля RD	Оцінка ECTS	Традиційна оцінка диф. заліку
95-100	A	Відмінно
85-94	B	Дуже добре
75-84	C	Добре
65-74	D	Задовільно
60-64	E	Достатньо
<60	Fx	Незадовільно
Заборгованість по лабораторних роботах	F	Недопущений

1.Лабораторні роботи: 10x5=50

2.Семестрова контрольна робота: 20

3.Поточні контрольні роботи: 30

ТЕМА 1

**Поняття алгоритму, властивості алгоритмів,
способи задавання алгоритмів.**

Етапи розв'язування задачі на комп'ютері

- **розробка алгоритму розв'язування задачі,**
- складання програми розв'язування задачі алгоритмічною мовою,
- ввід програми і даних в комп'ютер,
- налаштування програми (виправлення помилок),
- виконання програми на комп'ютері,
- аналіз отриманих результатів.

Поняття алгоритму

Алгоритм – послідовність, набір правил виконання обчислювального процесу, що обов'язково приводить до розв'язання певного класу задач після скінченного числа операцій

Це теоретичне визначення підходить до алгоритмів, які використовуються в рамках спеціальності «Комп'ютерна інженерія». Воно містить в собі три ознаки алгоритму:

1. Повинен існувати певний набір правил, пов'язаних між собою.
2. Набір правил повинен бути достатнім для розв'язування деякого класу задач.
3. Після застосування даних правил повинні обов'язково одержати результат

Приклад найпростішого алгоритму

Кожний зустрічається з алгоритмами зі шкільної лави. Правила, по яких виконуються арифметичні дії із багаторозрядними числами, є найпростішими прикладами алгоритмів.

$$\begin{array}{r} \times 123 \\ \hline 54 \\ + 492 \\ \hline 615 \\ + 6642 \\ \hline 6642 \end{array}$$
$$\begin{array}{r} \times 2157 \\ \hline 342 \\ + 4314 \\ + 8628 \\ \hline 6471 \\ + 737694 \\ \hline 737694 \end{array}$$

1. Сформулюйте набір правил
2. Вкажіть порядок їх застосування
3. Обґрунтуйте обов'язкове одержання результату

Походження терміну «алгоритм»



Термін "алгоритм" походить від імені стародавнього перського математика **Мухаммеда бен Муса аль Хорезмі**, який ще в IX столітті сформулював правила обчислень у десятковій системі числення.

Поняття алгоритму належить до первісних, основних, базисних понять математики, таких, як **множина** чи **натуральне число**.

Сторінка з «Алгебри» аль-Хорезмі — перського математика, від імені якого походить слово алгоритм.

Приклад застосування алгоритмів

Отримуючи відповідну інтерпретацію в конкретних застосуваннях, вони становлять значну і найбільш істотну **частину математичного апарата**, використовуваного в техніці.



Ця задача виникає при проектуванні доріг, ліній електропередач, трубопроводів та ін.

Алгоритм Прима-Краскала

Знаходження остовного дерева з найменшою вагою.

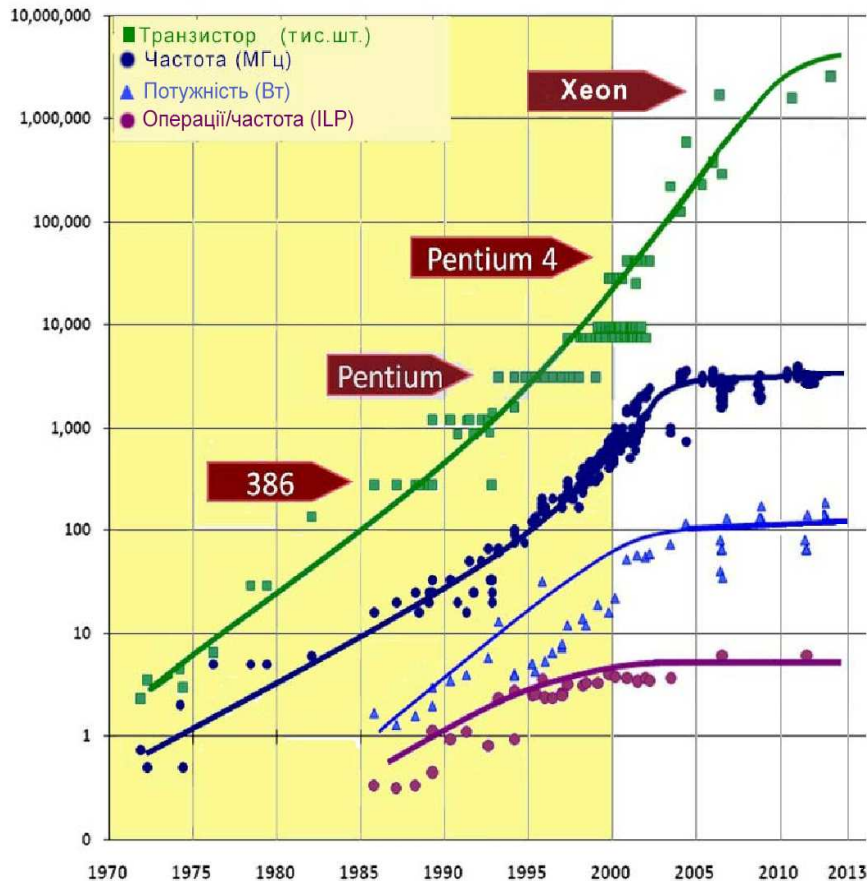
Практичне визначення алгоритму для комп'ютера

Алгоритм – чіткий опис послідовності дій, які необхідно виконати при розв'язуванні задачі.

Загальні етапи алгоритмів для комп'ютера:

1. Ввести початкові дані.
2. Перетворити початкові дані в результати.
3. Вивести результати.

Зростання показників продуктивності традиційних обчислювальних систем



До 2000 р. – зростання за **законом Мура** “Подвоєння показників кожні кілька років”. **Причина: закон Деннарда.**

“Зменшення транзисторів при незмінній площі процесора не вимагає зростання енергоспоживання”

Золоте століття: масштабування техпроцесу (зменшення розміру транзисторів) забезпечувало стійке зростання продуктивності протягом кількох десятиліть.

З 1994 по 1998 – частота зросла на 300%.

З 2007 по 2011 – частота зросла на 33%.

Нові парадигми програмування

1. Перехід до мультиядерності.

2. Відсутність необхідності економії пам'яті

Паралелізм на рівні команд (англ. Instruction-level parallelism - ILP)

РІЗНОВИДНОСТІ АЛГОРИТМІВ

1. Логічні алгоритми

- 1.1 Загальні комбінаторні алгоритми
- 1.2 Алгоритми на графах
- 1.3 Алгоритми пошуку
- 1.4 Алгоритми сортування
- 1.5 Комп'ютерна графіка

2. Чисельні алгоритми

- 2.1.Алгоритми інтерполяції
- 2.2.Алгоритми розв'язування нелінійних рівнянь
- 2.3.Алгоритми розв'язування систем алг. рівнянь
- 2.4.Алгоритми розв'язування диф. рівнянь
- 2.5.Алгоритми чисельного інтегрування

3. Алгоритми теорії обчислень та автомати

- 3.1.Алгоритми маніпуляції бітами
- 3.2.Алгоритм обчислення дня тижня (Алгоритм Судного Дня)

Приклад чисельного алгоритму

Типовим прикладом **чисельного алгоритму** є алгоритм Евкліда для знаходження найбільшого спільного дільника двох заданих додатних цілих чисел:

$$a \text{ і } b.$$

Довідково

Найбільшим спільним дільником (НСД) чисел a і b називається деяке найбільше число c , на яке обидва числа a і b діляться без остачі.

Алгоритм Евкліда складається з наступної системи послідовних вказівок:

Опис алгоритму Евкліда

1. Вводимо числа a і b .
2. Перевіряємо умову $a = b$. Якщо ця умова вірна, то розв'язок знайдений. Обидва числа є НСД одне для одного. Переходимо до пункту 6. Якщо $a \neq b$, то переходимо до 3.
3. Якщо $a < b$, то значення a і b міняємо місцями: $a \leftrightarrow b$.
4. Якщо a ділиться без остачі на b , то число b і є НСД. Переходимо до пункту 6, інакше переходимо до пункту 5.
5. Число a заміняємо на остачу від ділення, і переходимо до пункту 3.
6. Вивід результатів.

Приклад роботи алгоритму Евкліда:

1. Вводимо $a:=30$ і $b:=18$. Переходимо до п.2.
2. $30 \neq 18$. Переходимо до п.3.
3. $30 > 18$. Переходимо до п.4.
4. $30/18 = 1$ (остача 12). Переходимо до п.5.
5. $a:=12$. Переходимо до п.3.
3. $12 < 18$. Міняємо місцями a і b . Переходимо до п.4.
4. $18/12 = 1$ (остача 6). Переходимо до п.5.
5. $a:=6$. Переходимо до п.3.
3. $6 < 12$. Міняємо місцями a і b . Переходимо до п.4.
4. $12/6 = 2$ (остача 0). Переходимо до п.6.
6. Вивід НСД $(30, 18) = 6$.

Як видно, після п'ятої вказівки потрібно щоразу повертатися до третьої доти, поки не буде виконана четверта вказівка. Хоча заздалегідь невідомо, скільки буде потрібно таких циклічних переходів, але ясно, що для будь-яких двох чисел мета буде досягнута за **скінченне число кроків**.

C++

```
while (b) {  
    a %= b;  
    swap (a,b) ;  
}  
gcd = a;
```

Python

```
def gcd(a, b):  
    return b and gcd(b, a%b) or a
```

Java

```
public class GCD {  
    public static int gcd(int a,int b) {  
        while (b !=0) {  
            int tmp = a%b;  
            a = b;  
            b = tmp;  
        }  
        return a;  
    }  
}
```

Логічні алгоритми

Логічні алгоритми – це алгоритми, які маніпулюють даними з метою встановлення або визначення деякої закономірності.

До логічних алгоритмів можна віднести алгоритми класифікації, сортування, планування та ін.

Приклад логічного алгоритму

Найпростішим прикладом логічного алгоритму є алгоритми сортування. Розглянемо, наприклад, сортування Хоара.

I етап

Номер кроку	<div> $i \rightarrow$ <div>012345</div> $\leftarrow j$ </div>						Примітки
	40	11	83	21	75	64	
1	40					64	$k_0 < k_j$ ОК
2	40				75		$k_0 < k_j$ ОК
3	40 21			21 40			$k_0 > k_j$ Обмін $k_0 > k_i$ ОК

3	21	11	83	40	75	64	Примітки
4		11		40			$k_0 > k_i$ ОК
5			83 40	40 83			$k_0 < k_i$ Обмін $k_0 > k_i$ ОК
6	21	11	40	83	75	64	Кінець етапу

II етап

Номер кроку	$i_1 \rightarrow$	$\leftarrow j_1$	Примітки	$i_2 \rightarrow$	$\leftarrow j_2$			Примітки
	0	1		0	1	2	3	
	21	11	Поч. спис.	40	83	75	64	Поч. спис.
7	21 11	11 21	$k_0 > k_j$ Обмін $k_0 > k_i$ ОК	40	83 64		64 83	$k_0 > k_j$ Обмін $k_0 > k_i$ ОК
8				40		75	83	$k_0 > k_i$ ОК
9	11	21	Кінець алг.	40	64	75	83	Кінець алг.

Обчислювальна складність: $O(n \log_2 n)$

Послідовні і паралельні алгоритми

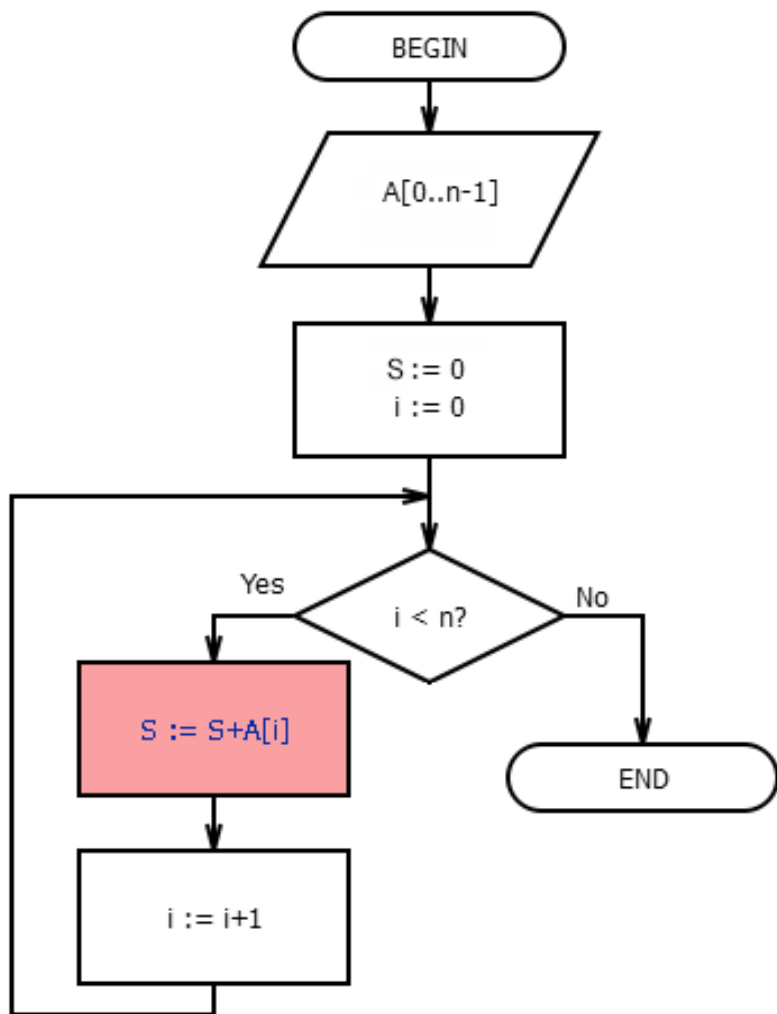
Послідовний алгоритм – це алгоритм, у якому кожний наступний етап використовує результати деякої підмножини попередніх етапів.

Приклад. Алгоритм послідовного додавання n чисел.

На кожному кроці до часткової суми додається черговий доданок:

$$S := S + A[i].$$

Додавання наступного числа може виконуватися тільки після завершення операції додавання попереднього.



Паралельний алгоритм – алгоритм, який може бути реалізований по частинах на окремих обчислювальних засобах з наступним об'єднанням часткових результатів для одержання глобального розв'язку задачі.

Приклад. Паралельний алгоритм додавання n чисел.

Розіб'ємо всі доданки на пари і здійснимо додавання двох чисел усередині кожної пари.

$A = [2, 3, 5, 12, 11, 17]$

$A[0] + A[1]$

$2 + 3 = 5$

$A[2] + A[3]$

$5 + 12 = 17$

$A[4] + A[5]$

$11 + 17 = 28$

Усі ці операції незалежні. Отримані часткові суми також розіб'ємо на пари і знову здійснимо додавання двох чисел усередині кожної пари.

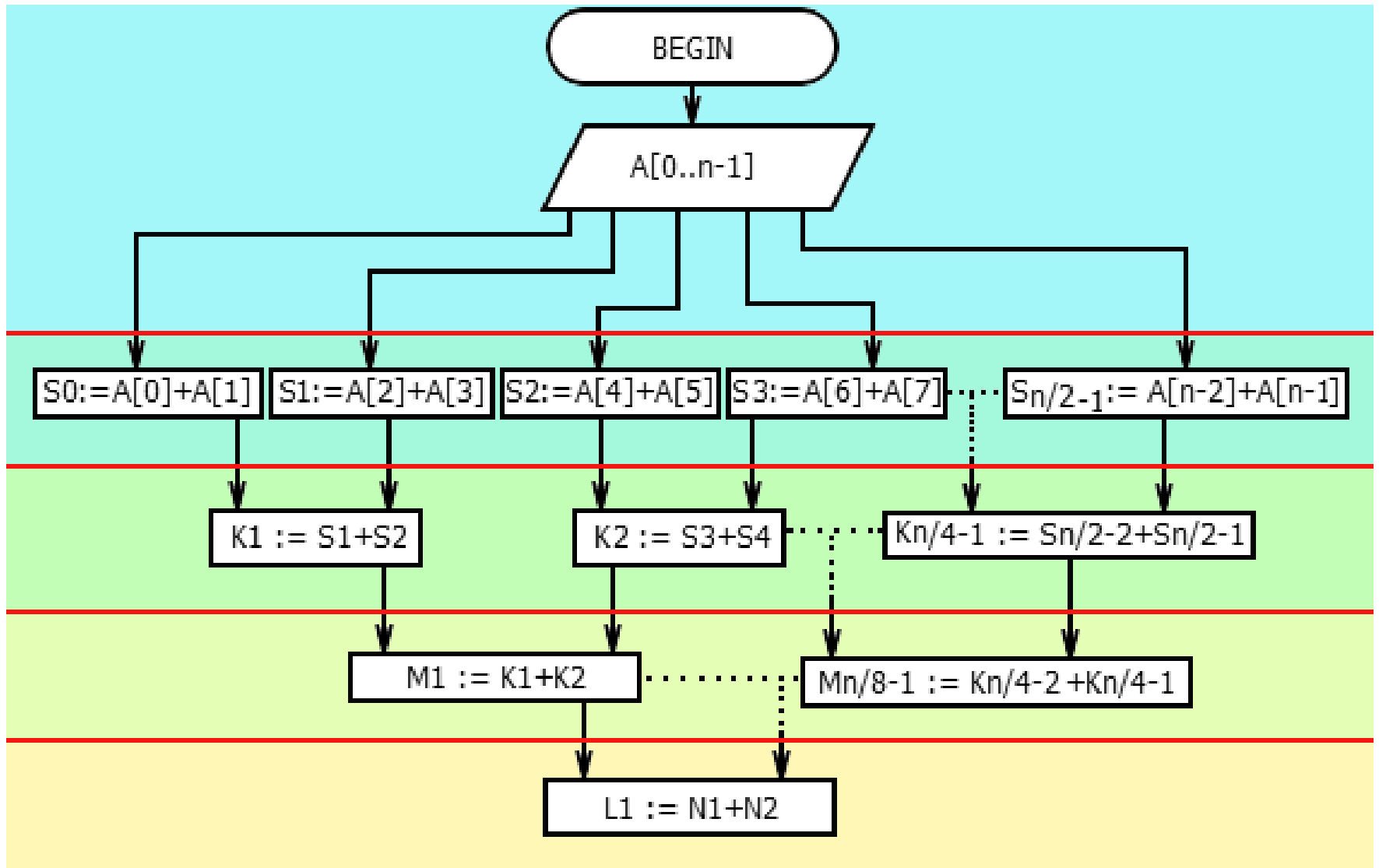
$5 + 17 = 22$

28

Знову всі операції незалежні.

$22 + 28 = 50$

Уся сума буде отримана через $\log_2 n$ кроків.



Загальні властивості послідовних алгоритмів

Послідовні алгоритми мають ряд загальних властивостей:

1) **Дискретність алгоритму:** алгоритм можна розглядати як процес послідовної побудови величин, що йде в дискретному часі за певною послідовністю інструкцій, яку називають програмою.

2) **Масовість алгоритму:** алгоритм придатний для розв'язування цілого класу однотипних задач.

У цьому аспекті таблиця множення не є алгоритмом, а множення стовпчиком багатозначних чисел – алгоритм. Наприклад, в алгоритмі Евкліда числа a і b вибираються з нескінченної (зліченної) множини цілих чисел.

3) **Детермінованість алгоритму:** після виконання чергового етапу однозначно визначено, що робити на наступному етапі.

4) **Елементарність кроків алгоритму:** розв'язування задачі розбивається на етапи, кожний з яких повинен бути простим і локальним. Це означає, що відповідна операція повинна бути елементарною для виконавця алгоритму (людини або машини).

Наприклад, операції в алгоритмі Евкліда: порівняння, ділення і перестановки чисел, – стандартні і звичні. У той же час **алгоритм Евкліда може фігурувати як елементарна операція** більш складного алгоритму.

5) **Результативність алгоритму:** алгоритм через скінченне число кроків повинен привести до зупинки, яка свідчить про досягнення необхідного результату.

Наприклад. Алгоритм пошуку шляху в лабіринті результативний, якщо шлях знайдений.

Загальні властивості паралельних алгоритмів

1. Кожний паралельний алгоритм має граф свого виконання.

Процес обробки даних може бути виражений у вигляді односпрямованого плоского графа. В такому випадку висота графа буде рівна мінімальному числу кроків / етапів розв'язування задачі на ідеальній паралельній обчислювальній системі з необмеженим числом процесорів.

2. Степінь паралелізму

Степенем паралелізму етапу чисельного алгоритму називається число операцій, які на даному етапі можна виконувати паралельно (ширина графа, побудованого описаним вище способом).

Способи задавання алгоритмів

Існує велика кількість формальних засобів, що дозволяють однозначно задавати сучасні алгоритми. У рамках даного курсу будемо розглядати тільки традиційні, найпоширеніші способи:

1. Спосіб задавання алгоритму у вербальній (словесній) формі.

Це найдавніший спосіб задавання алгоритмів. Перші алгоритми формулювалися винятково словесно. Прикладом словесної форми задавання може служити алгоритм Евкліда.

Як правило, словесний опис складається з пунктів, у кожному з яких чітко **формулюється найпростіша дія**. Потім дається **вказівка про подальші дії**.

2. Спосіб задавання алгоритму в графічній формі.

Блок-схема – це графічне зображення алгоритму. При її побудові **вміст кожного кроку алгоритму записується в довільній формі усередині блоку**, представленого геометричною фігурою. **Порядок виконання кроків вказується за допомогою стрілок**, що з'єднують блоки. Використання різних геометричних фігур відображає різний характер виконуваних дій.

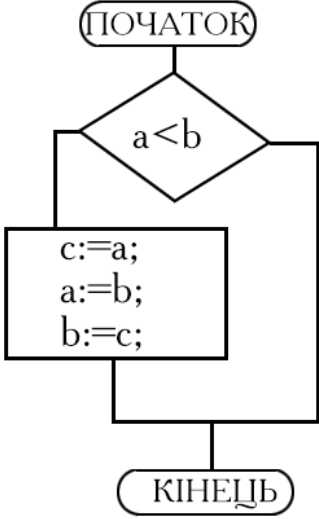
3. Спосіб задавання алгоритму у вигляді псевдокоду.

Псевдокод – це напівформалізований опис алгоритмів на умовній алгоритмічній мові, що включає в себе як елементи мови програмування, так і фрази природньої мови, загальноприйняті математичні позначення і ін. Такий спосіб опису часто зустрічається в навчальній і науковій літературі і використовується з метою забезпечення компактного представлення алгоритму.

4. Спосіб задавання алгоритму у вигляді програми для комп'ютера.

Для задавання алгоритму у вигляді програми використовуються конструкції конкретної мови програмування. Представлений у вигляді програми алгоритм після компіляції перетворюється в файл, що виконується на комп'ютері.

Приклад.

<p>Якщо значення змінної <i>a</i> менше за значення <i>b</i>, то значення <i>b</i> присвоїмо змінній <i>a</i> і навпаки</p>	 <pre>graph TD; Start([ПОЧАТОК]) --> Decision{a < b}; Decision -- Yes --> Process[c:=a; a:=b; b:=c;]; Decision -- No --> End([КІНЕЦЬ]); Process --> End;</pre>	$\text{If } a < b : a \leftrightarrow b$	<pre>Program Flip; Var a,b,c : Integer; begin If a < b then begin c := a; a := b; b := c; end; end.</pre>
---	---	--	--

Зображення алгоритму у вигляді блок-схеми

Блок-схема

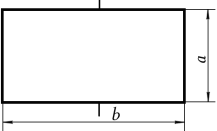
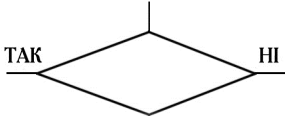
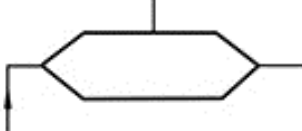
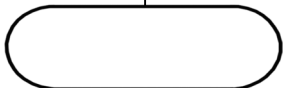
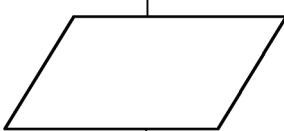
Блок-схемою називається графічне зображення алгоритму, коли окремі його етапи зображуються за допомогою різних геометричних фігур – блоків, а зв'язки між етапами (послідовність виконання етапів) вказуються за допомогою стрілок, що з'єднують ці фігури.

Правила оформлення

Конфігурація і розміри блоків, а також порядок графічного оформлення блок-схем регламентовані ГОСТ 10.002-80 і ГОСТ 10.003-80 "Схеми алгоритмів і програм". Блоки супроводжуються написами.

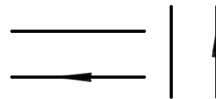
Блоки

Це найчастіше використовувані блоки. Блоки і елементи зв'язків називають елементами блок-схем.

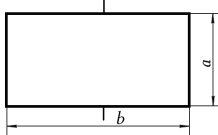
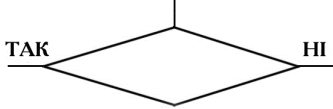
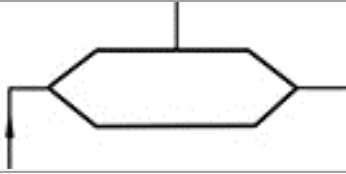
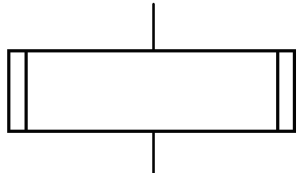
 Процес	 Розв'язок	 Модифікація	 Початок, Кінець	 Введення/Висновок
---	--	---	---	--

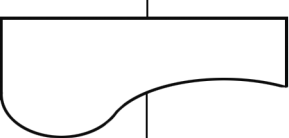
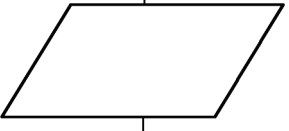



Потоки

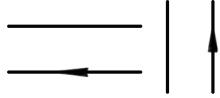


Горизонтальні потоки, що мають напрямок праворуч, і вертикальні потоки, що мають напрямок знизу нагору, повинні бути обов'язково позначені стрілками.



Таблиця 1

Назва	Елемент	Коментар
Процес		Обчислювальна дія або послідовність обчислювальних дій
Розв'язок		Перевірка умови
Модифікація		Заголовок циклу
Визначений процес		Звертання до процедури

Документ		Вивід даних, друк даних
Введення/Висновок		Ввід/Вивід даних
З'єднувач		Розрив лінії потоку
Початок, Кінець		Початок, кінець, пуск, зупинка, вхід і вихід у допоміжних алгоритмах
Коментар		Використовується для розміщення написів

Горизонтальні і вертикальні потоки		Лінії зв'язків між блоками, напрямом потоків
Злиття		Злиття ліній потоків
Міжсторінковий з'єднувач		

Відстані

Відстань між паралельними лініями потоків повинне бути не менше 3 мм, між іншими елементами схеми – не менше 5 мм.

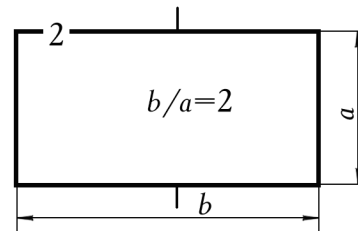
Розміри

Горизонтальний і вертикальний розміри блоку повинні бути кратними 5 мм (ділитися на 5 без остачі). Співвідношення горизонтального і вертикального

розмірів блоку $b/a = 1.5$ є основним. При ручному кресленні блоку припустиме відношення $b/a = 2$. Блоки "Початок", "Кінець" і "З'єднувач" мають висоту $a/2$, тобто вдвічі менше основної висоти блоків. Для розміщення блоків рекомендується поле аркуша розбивати на горизонтальні і вертикальні (для схем, що розгалужуються) зони.

Нумерація

Для зручності опису блок-схеми кожний її блок слід пронумерувати. Бажано використовувати наскрізну нумерацію блоків. Номер блоку розташовують у розриві в лівій верхній частині рамки блоку.



Лінійні, розгалужені і циклічні алгоритми

По характеру зв'язків між блоками розрізняють алгоритми лінійні, що розгалужуються та циклічної структури.

Лінійний алгоритм – це алгоритм, у якому блоки виконуються послідовно зверху вниз від початку до кінця. На рис. 1 наведений приклад блок-схеми алгоритму обчислення периметра P і площі S квадрата зі стороною довжини A .

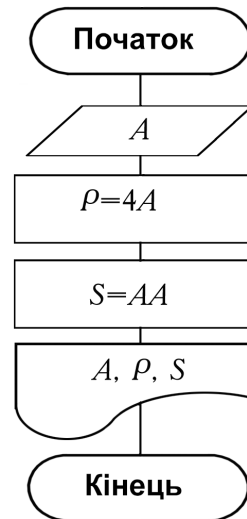
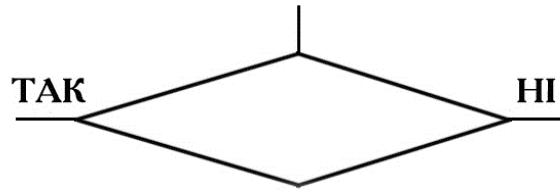


Рис. 1. Приклад лінійного алгоритму

Розгалужені алгоритми

У блок-схемах розгалуження починається на виходах елемента "Розв'язок", за допомогою якого в алгоритмі виконується перевірка умови.



Кількість розгалужень тим більше, чим більше умов, що перевіряються.

Для пояснення розглянемо розв'язування задачі знаходження значення функції $z = y/x$.

Приклад розгалуженого алгоритму

По-перше, потрібно з обчислень виключити варіант
 $x = 0$.

По-друге, проінформувати користувача алгоритму про помилку, яка виникла. Якщо цього не зробити, то при обчисленнях може виникнути аварійний вихід до одержання результату.

У професійній практиці аварійні завершення вкрай небажані, тому що до цього моменту вже може бути накопичена певна кількість результатів.

Розв'язок задачі представлений блок-схемою на рис. 2.

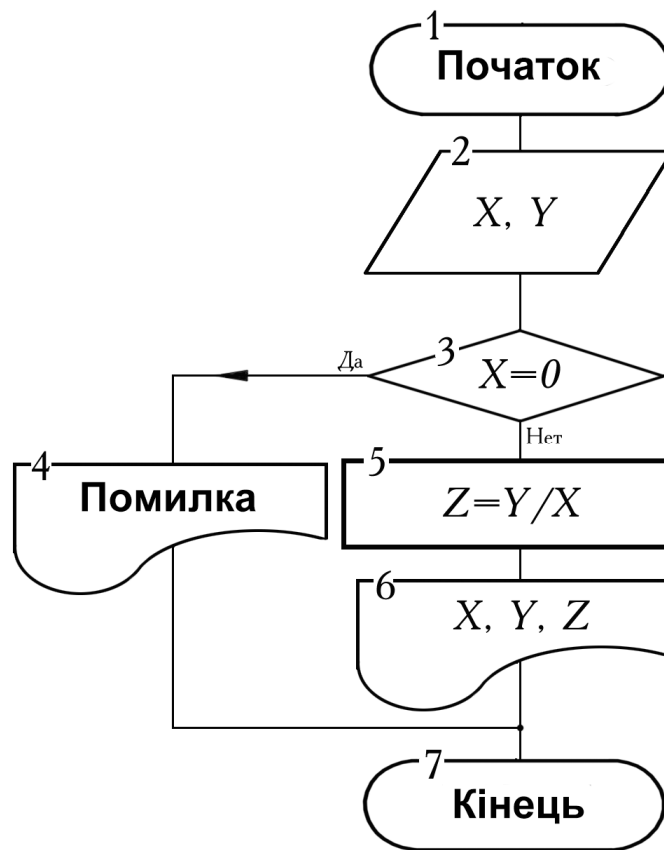


Рис.2.Алгоритм, що розгалужується

Циклічний алгоритм

Цикл — різновид керуючої конструкції, призначеної для організації **багаторазового виконання** тих самих ділянок алгоритму при різних значеннях змінних.

Алгоритми, що містять цикли, називаються циклічними. Використання циклів суттєво скорочує обсяг алгоритму.

Етапи організації циклу

- ◎ підготовка (ініціалізація) циклу ;
- ◎ виконання обчислень циклу (тіло циклу);
- ◎ модифікація параметрів ;
- ◎ перевірка умови закінчення циклу

Розрізняють *цикли з наперед відомою і наперед невідомою кількістю проходів*.

Цикл називається **детермінованим**, якщо число повторень тіла циклу заздалегідь відоме або визначене.

For** $i := 0$ **to** n **do

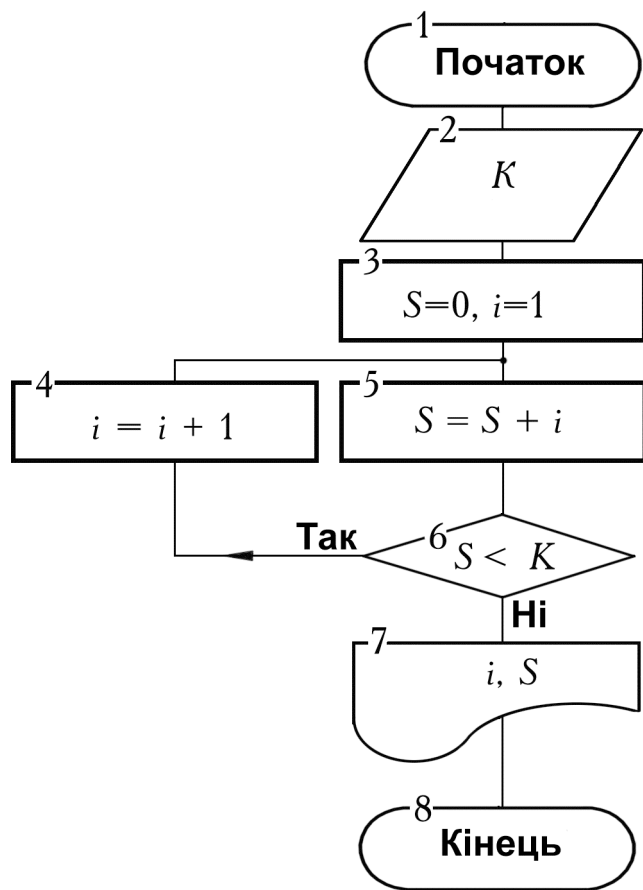
Цикл називається **ітераційним**, якщо число повторень тіла циклу заздалегідь невідоме, а залежить від значень параметрів (деяких змінних), що брали участь в обчисленнях

Цикл з передумовою

```
while <умова> do  
begin  
    <тіло циклу>  
end;
```

Цикл з післяумовою

```
repeat  
    <тіло циклу>  
until <умова>
```



Алгоритм із ітераційним циклом

Задача. Вказати найменшу кількість членів ряду натуральних чисел 1, 2, 3, ..., сума яких більше числа K .

Рис. 3. Циклічний алгоритм

Блок-схема алгоритму Евкліда

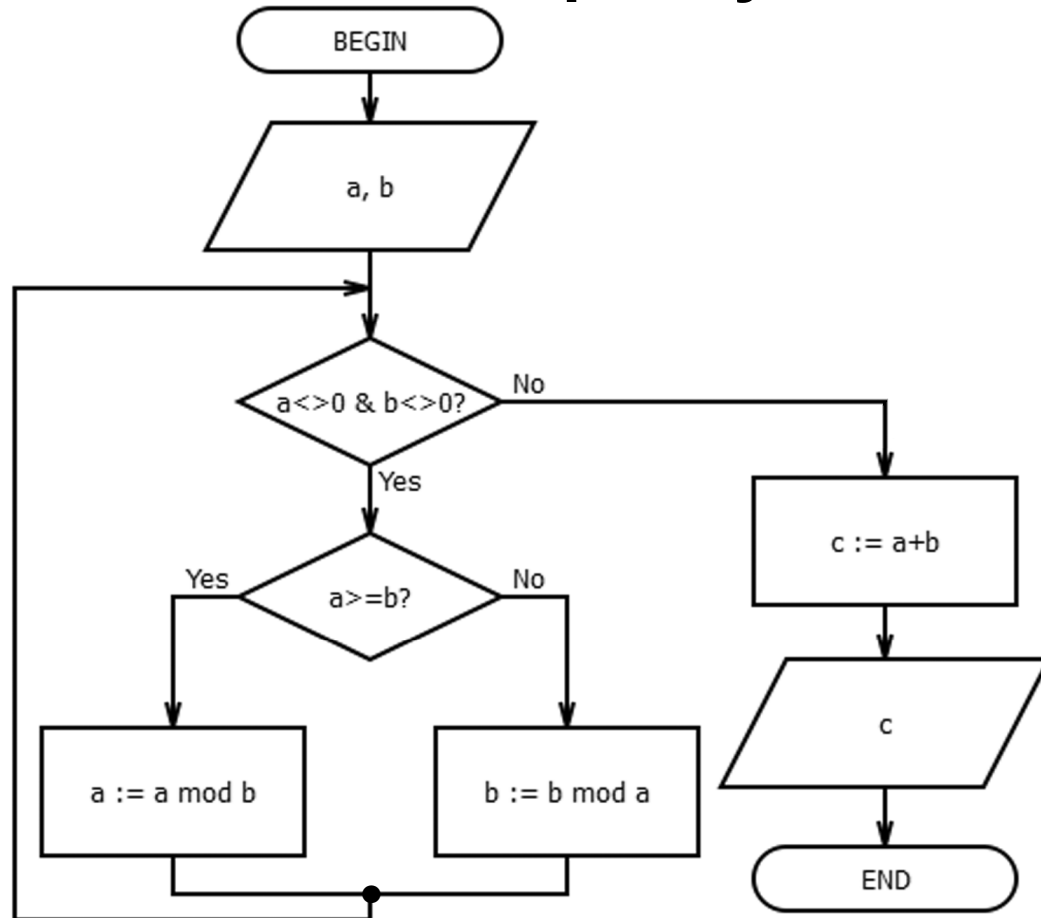


Рис.4. Приклад алгоритму Евкліда

Псевдокод алгоритму Евкліда

1. **Input** a, b .
2. **If** $a < b$ **then** $a \leftrightarrow b$ **else** знайти q, r з $a = bq + r$.
3. **If** $r = 0$ **then** $c := b$ **else** $a := b; b := r$; **goto** 2.

Приклад задавання алгоритму Евкліда мовою Pascal і Python

```
function nod( a, b: longint): longint;  
  begin  
    while (a <> 0) and (b <> 0) do  
      if a >= b then  
        a := a mod b  
      else  
        b := b mod a;  
    nod := a + b;  
  end;  
def nod(a, b):  
  while b:  
    a, b = b, a % b  
  return a
```

Інші способи задавання алгоритмів

Мережі Петрі

Мережу Петрі задають у вигляді кортежу:

$$\Phi = (P, T, F, M),$$

де $P = \{p_1, p_2, \dots, p_n\}$ – скінченна множина позицій, $n > 0$;

$T = \{t_1, t_2, \dots, t_m\}$ – скінченна множина переходів, $m > 0$;

$F \subseteq \{P \times T\} \cup \{T \times P\}$ – скінченна множина ребер, які з'єднують переходи і позиції;

M – множина маркувань.

$I \subset \{P \times T\}$ – підмножина вхідних ребер,

$O \subset \{T \times P\}$ – підмножина вихідних ребер.

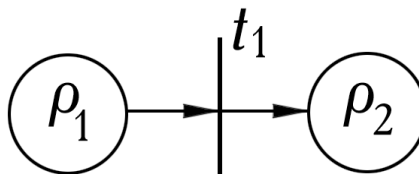


Рис. 5. Найпростіша мережа Петрі

Приклад.

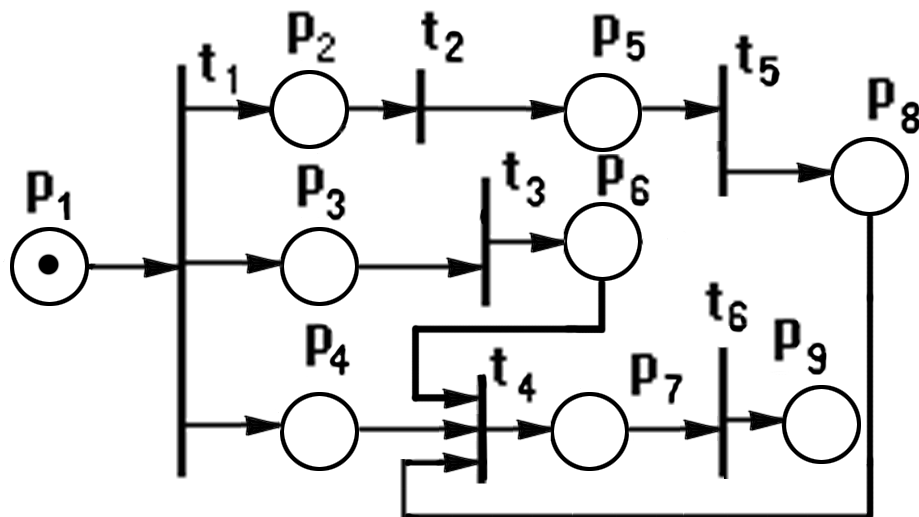


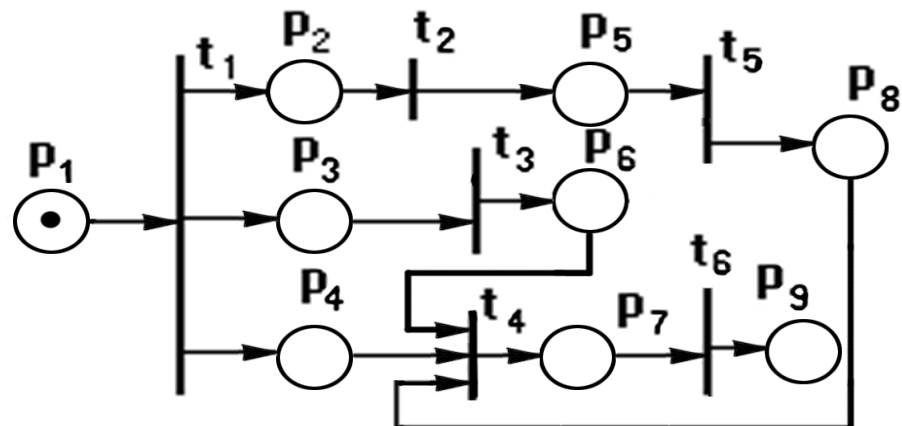
Рис.6. Приклад мережі Петрі

$$P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, \}$$

$$T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$$

$$I = \{(p_1, t_1), (p_2, t_2), (p_3, t_3), (p_6, t_4), (p_4, t_4), (p_8, t_4), (p_5, t_5), (p_7, t_6)\}$$

$$O = \{(t_1, p_2), (t_1, p_3), (t_1, p_4), (t_2, p_5), (t_3, p_6), (t_4, p_7), (t_5, p_8), (t_6, p_9)\}$$



$$F = \begin{pmatrix} & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 & p_9 \\ t_1 & 1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ t_2 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ t_3 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ t_4 & 0 & 0 & 0 & 1 & 0 & 1 & -1 & 1 & 0 \\ t_5 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ t_6 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & -1 \end{pmatrix}$$

Динаміка мереж Петрі

базується на двох основних правилах

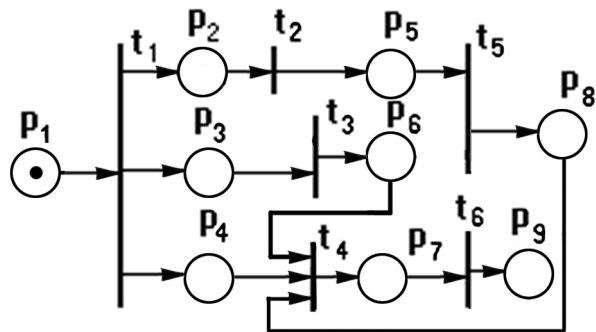
1. Правило активації

Перехід $t \in T$ активується відповідним маркуванням μ його вхідних позицій тоді і тільки тоді, коли вони містять задану кількість міток.

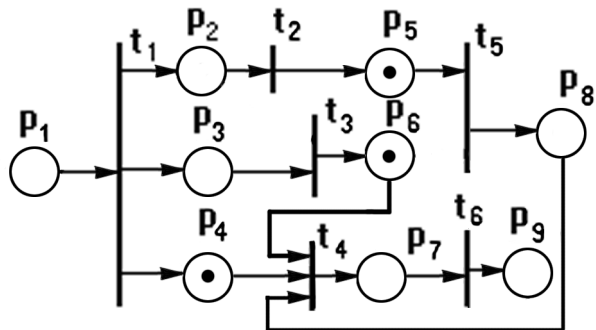
Активация переходів створює передумови їх спрацьовування, але не кожний активований перехід може спрацювати.

2. Правило спрацьовування

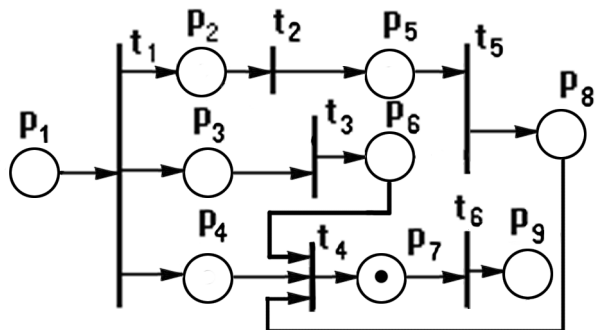
Спрацьовування переходу t , активованого маркуванням $\mu^{(\tau)}$, породжує нове маркування мережі $\mu^{(\tau+1)}$ шляхом вилучення міток із вхідних позицій і розміщення деякої кількості міток на вихідних позиціях.



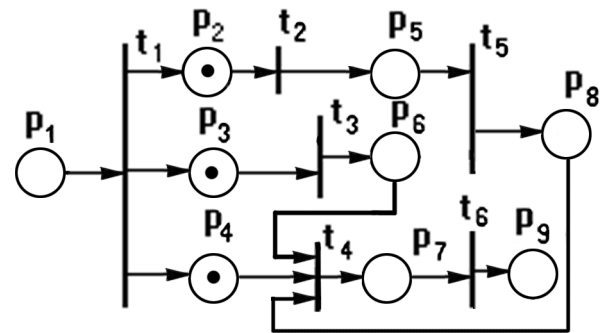
$\tau = 0$



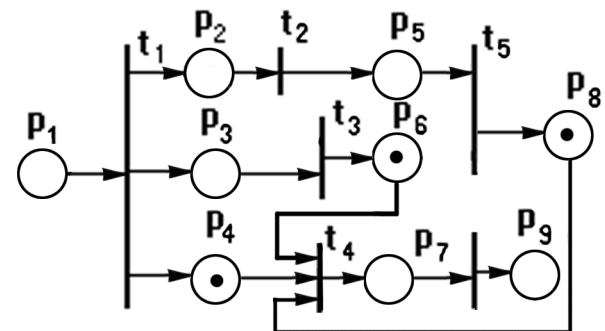
$\tau = 2$



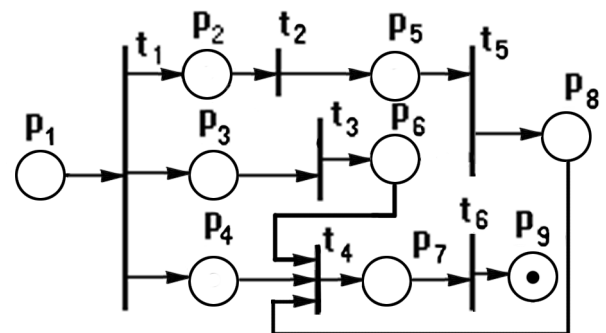
$\tau = 4$



$\tau = 1$



$\tau = 3$



$\tau = 5$