**Лекція №1**

***Поняття алгоритму***

Алгоритм Евкліда (визначення НСД).

*Вхід*. *р* і *q*, додатні цілі числа.

*Вихід*. *g*, НСД чисел *p* і *q*.

*Метод*. 1. Знайти *r*, остачу від ділення *p* на *q*.

2. Якщо *r* = 0, покласти *g* = *q* і зупинитись.

Інакше, покласти *p* = *q*, *q* = *r* і перейти на l.

Алгоритм – всюди визначений, якщо він зупиняється на всіх входах, тобто на всіх значеннях вхідних даних.

**Приклад 1.1.**

*Вхід*. *p* і *q*, додатні цілі числа.

*Вихід*. *g*, сума чисел *p* і *q*.

*Метод*. Покласти *g = p + q* і зупинитись.

Алгоритм – частковий, якщо він не зупиняється на деяких входах.

**Приклад 1.2.**

*Вхід*. *p* – довільне ціле число.

*Вихід*. *p.*

*Метод*. 1. Якщо *p =* 0, то перейти на 1. Інакше – зупинитись.

При *р =* 0 алгоритм не зупиняється.

*Загальні риси алгоритмів*

1. *Дискретність.* В процесі виконання алгоритму із початкової скінченної системи величин та величин, знайдених в попередні моменти часу послідовно (в дискретному часі) будується система величин по деякому закону (програмі).

2. *Елементарність.* Закон повинен бути простим (конструктивним).

3. *Детермінованість.* Система величин, які одержуються в якийсь момент часу однозначно визначається системою величин, одержаних в попередні моменти часу.

4. *Результативність.* Якщо спосіб одержання величини не дає результату, то повинно бути вказано, що вважати результатом алгоритму.

5. *Масовість.* Початкова система величин (вхід алгоритму) може вибиратись із потенційно нескінченної множини.

Алгоритм, визначений загальними рисами 1-5 називається *інтуїтивним* поняттям алгоритму. Такому інтуїтивному поняттю алгоритму задовольняє наступне: алгоритм – це скінченна множина правил (програма), яка дозволяє кожному елементу *x* із деякої нескінченної області (області визначення алгоритму) ставити у відповідність скінченну послідовність елементів <*x*0, *x*1, …, *xk*> (процес обчислень) так, що кожний наступний елемент *xi*+1 будується за попереднім елементом *xi* застосуванням до нього деякого елементарного правила (крок алгоритму), а застосування правила до елемента *xk* уже неможливе. В цьому випадку елемент *xk* вважається результатом застосування алгоритму до елемента *x*.

Отже, алгоритм – це скінченна послідовність правил «конструктивного» перетворення вхідних даних у вихідні.

Якщо алгоритм зупиняється через скінченну кількість кроків (скінченна кількість застосувань правил перетворення), то говорять, що алгоритм визначений на вхідних даних, в іншому випадку – коли алгоритм працює нескінченно довго – говорять, що алгоритм невизначений на вхідних даних.

**Приклад 1.3.** Необхідно знайти алгоритм, який дозволяє для кожної четвірки цілих чисел *a*, *b*, *c*, *d* вияснити, існують чи ні цілі числа *x*, *y*, які задовольняють рівнянню

*ax*2 + *bxy* + *cy*2 = *d*.

Такий алгоритм був побудований.

**Приклад 1.4** (проблема відповідностей Поста). Нехай (*x*, *y*) ⊆ Σ\*×Σ\* – скінченна множина пар непустих слів в алфавіті Σ.

**Проблема.** Чи існує скінченна послідовність пар (*x*1, *y*1), ..., (*xn*, *yn*) (не обов’язково різних) таких, що

*x*1*x*2…*xn* = *y*1*y*2…*yn*.

Така послідовність називається розв’язуючою.

**Приклад 1.5.** Нехай {(*abbb*, *b*), (*a*, *aab*), (*ba*, *b*)} – скінченна множина пар в алфавіті Σ = {*a*, *b*}.

Тоді послідовність (*a*, *aab*), (*a*, *aab*), (*ba*, *b*), (*abbb*, *b*) – розв’язуюча так, як

(*a*)(*a*)(*ba*)(*abbb*) = (*aab*)(*aab*)(*b*)(*b*).

Скінченна множина пар {(*ab*, *aba*), (*aba*, *baa*), (*baa*, *aa*)} пар в цьому ж алфавіті розв’язуючих послідовностей не має.

Побудувати алгоритм, який розв’язує цю проблему неможливо. Але для цього потрібно довести неіснування алгоритму. Як це зробити? Для цього треба точно знати, що таке алгоритм, тобто мати якесь математичне поняття, еквівалентне поняттю алгоритму.

Розв’язання задачі точного визначення алгоритму було одержано в роботах Гільберта, Геделя, Чьорча, Кліні, Поста, Т’юрінга. Для цього розглядалися функції *f*:*N*→*N* і клас функцій, які обчислюються алгоритмами ототожнювався з класом спеціально побудованих функцій – рекурсивних (таке ототожнення або гіпотеза відома під назвою тези Чьорча).

Алгоритмічна обчислюваність функцій означає існування алгоритму, який знаходить їх значення (у випадку визначеності функцій) і працює нескінченно довго в іншому.

Тези Чьорча достатньо, щоб доводити нерозв’язність алгоритмічних проблем. Тому, що питання про алгоритмічну обчислюваність функцій еквівалентне питанню про її рекурсивність.

Поняття рекурсивної функції – математично точне. Тому можна безпосередньо доводити, що функція, яка розв’язує задачу не може бути рекурсивною.

*Поняття примітивно рекурсивної*, *частково рекурсивної та рекурсивної*

*функцій. Алгебри рекурсивних функцій*

Розглянемо спосіб опису алгоритмічно обчислюваних функцій, що грунтується на породженні таких функцій за допомогою певних обчислюваних операцій із так званих базових функцій.

Надалі під функцією будемо розуміти функцію натуральних аргументів і значень.

Базовими функціями називаються найпростіші функції

*o*(*x*) = 0,

*s*(*x*) = *x* + 1 та функції-селектори

 (*x*1, …, *xn*) = *xm*, де *n* ≥ *m* ≥ 1.

Всі базові функції всюди визначені.

Основними обчислювальними операціями будуть операції *суперпозиції* ***S****n*+1, *примітивної рекурсії* ***R*** та *мінімізації* ***M***.

Операція суперпозиції ***S****n*+1 дозволяє із *n*-арної функції *g*(*x*1, …, *xn*) та *n* функцій *g*1(*x*1, …, *xm*), ... , *gn*(*x*1, …, *xm*), однакової арності утворити функцію

*f*(*x*1, …, *xm*) = *g*(*g*1(*x*1,…, *xm*), …, *gn*(*x*1,…, *xm*)).

Таку функцію позначають ***S****n*+1(*g*, *g*1, …, *gn*).

Операція примітивної рекурсії ***R*** дозволяє із *n*-арної функції *g* та *n*+2-арної функції *h* утворити *n*+1-арну функцію *f* за допомогою наступних співвідношень:

*f*(*x*1, …, *xn*, 0) = *g*(*x*1, …, *xn*)

*f*(*x*1,…, *xn*, *y* + 1) = *h*(*x*1,…, *xn*, *y*, *f*(*x*1,…, *xn*, *y*))

Таку функцію позначають ***R***(*g*, *h*).

Операція мінімізації ***М*** дозволяє із *n*-арної функції *g* утворити *n*-арну функцію *f*, що задається співвідношенням:

*f*(*x*1, …, *xn*) = μ*y*(*g*(*x*1, …, *xn*-1, *y*) = *xn*).

Тобто, значення функції *f*(*x*1, …, *xn*) дорівнює найменшому *y* для якого *g*(*x*1, …, *xn*-1, *y*) = *xn*.

Значення функції *f*(*x*1, …, *xn*) вважається невизначеним у випадках:

1) для всіх *y* значення *g*(*x*1, …, *xn*-1, *y*) ≠ *xn*;

2) для всіх *y* < *t* значення *g*(*x*1, …, *xn*-1, *y*) визначене *і* ≠ *xn*, а значення *g*(*x*1, …, *xn*-1, *t*) невизначене;

3) значення *g*(*x*1, …, *xn*-1, 0) невизначене.

Таку функцію позначають ***M***(*g*).

Функцію, яку можна одержати з базових функцій за допомогою скінченної кількості застосувань операцій суперпозиції та примітивної рекурсії, називають *примітивно рекурсивною функцією* (скорочено ПРФ).

Функцію, яку можна одержати з базових функцій за допомогою скінченної кількості застосувань операцій суперпозиції, примітивної рекурсії та мінімізації, називають *частково рекурсивною функцією* (скорочено ЧРФ).

Всюди визначену ЧРФ називають *рекурсивною функцією*(скорочено РФ).

Із визначень ПРФ, ЧРФ, РФ маємо такі співвідношення між класами функцій:

ПРФ ⊆ РФ ⊆ ЧРФ.

****Приклад 1.6.** За визначенням, функції

*o*(*x*) = 0, *s*(*x*) = *x* + 1, (*x*1, …, *xn*) = *xm*,

де *n*≥*m*≥1 – базові.

Для *n*-місної функції *on*(*x*1, …, *xn*) = 0 маємо

**

*on*(*x*1, …, *xn*) = ***S***2(*o*, (*x*1, …, *xn*))

і тому функція *on*– примітивно рекурсивна.

Зрозуміло, що існують алгоритми для обчислення значень базових функцій

*o*(*x*) = 0, *s*(*x*) = *x* + 1, (*x*1, …, *xn*) = *xm*.

При цьому конструктивне перетворення вхідних даних функції *f*(*x*1, …, *xn*) в результати означає, що для будь-яких натуральних чисел *x*1, …, *xn*вхідне слово  #…# перетворюється в слово , якщо *f*(*x*1, …, *xn*) визначене і працює нескінченно довго, якщо *f*(*x*1, …, *xn*) не визначене (–*xi* одиниць).Так, обчислення функції *o*(*x*) зводиться до витирання вихідного слова, обчислення функції *s*(*x*) до дописування одиниці до вхідного слова, обчислення (*x*1, …, *xn*) до стирання всіх вхідних компонент, крім *m*-ї.

Оператори суперпозиції, примітивної рекурсії та мінімізації теж породжують алгоритмічно обчислювані функції із алгоритмічно обчислюваних функцій.

Дійсно, у випадку суперпозиції, якщо ми можемо обчислювати значення функцій *g*, *g*1, …, *gn*, то значення їх суперпозиції *f* = ***S****n*+1(*g*, *g*1, … , *gn*) в точці (*a*1, …, *am*) можна обчислити за допомогою наступного алгоритму

function *f*(*x*1, …, *xm*)

begin

*b*1 = *g*1(*x*1, … , *xm*)

. . . . . . . . . . . .

*bn* = *gn*(*x*1, … , *xm*)

*b* = *g*(*b*1, … , *bn*)

end.

Число *b* буде значенням функції *f* в точці (*a*1, … , *am*).

У випадку примітивної рекурсії, якщо ми можемо обчислювати значення функцій *g*, *h*, то значення функції *f* = ***R***(*g*, *h*) в точці (*a*1, … , *an*, *m*+1) може бути обчислене за допомогою наступного алгоритму

function *f*(*x*1, … , *xn*, *y*)

begin

if *y* = 0 then *f* = *g*(*x*1, … , *xn*)

else *f* = *h*(*x*1, …, *xn*, *y*, *f*(*x*1, … , *xn*, *y* – 1))

end

В результаті рекурсивних викликів, одержимо наступну послідовність чисел:

*b*0 = *g*(*a*1,…, *an*),

*b*1 = *h*(*a*1,…, *an*, 0, *b*0),

*b*2 = *h*(*a*1,…, *an*, 1, *b*1),

………………………,

*bm*+1 = *h*(*a*1,…, *an*, *m*, *bm*).

Число *bm*+1 буде значенням функції *f* в точці (*a*1, … , *an*, *m* + 1).

У випадку операції мінімізації, якщо ми можемо обчислювати значення функції *g*, то значення функції *f* = ***M***(*g*) в точці (*a*1, …, *an*) може бути обчислене за допомогою наступного алгоритму

function *f*(*x*1, … , *xn*)

begin

*i* = 0

while *g*(*x*1, …, *xn*-1, *i*) ≠ *xn*

do *i* = *i* + 1

*f* = *i*

end.

Іншими словами, ми послідовно обчислюємо значення

*g*(*a*1, …, *an*, 0),

*g*(*a*1, …, *an*, 1),

… .

Найменше значення *a*, для якого

*g*(*a*1, … , *an-*1, *a*) = *an*

і є значенням функції *f* в точці (*a*1, … , *an*).

*Характеристичною функцією* χ*А* множини натуральних чисел *А* називається одномісна функція, рівна 0 в точках множини *А* і рівна 1 в точках, які не належать *А*.

*Частковою характеристичною функцією* множини натуральних чисел *А* називається одномісна функція, рівна 0 в точках множини *А* і не визначена в точках, які не належать *А*.

Множина натуральних чисел *А* називається *примітивно рекурсивною*, якщо її характеристична функція примітивно рекурсивна.

Множина натуральних чисел *А* називається *рекурсивною*, якщо її характеристична функція рекурсивна.

Зрозуміло, що кожна примітивно рекурсивна функція є рекурсивною (за визначенням). Звідси випливає, що кожна примітивно рекурсивна множина є рекурсивною.

**Теза Черча**. *Клас алгоритмічно обчислюваних числових функцій співпадає з класом всіх частково рекурсивних функцій.*

Далі алгоритми будемо записувати в мові ПсевдоPaskal, яка є спрощеним діалектом мови Paskal. Операторами цієї мови будуть наступні:

<ідентифікатор> = <вираз>,

while <вираз> do {<оператор>, …, <оператор>}

for <вираз> to <вираз> {<оператор>, …, <оператор>}

Нехай χ(*х*) – характеристична функція множини натуральних чисел *А*.

Тоді функція χ*ч*(*х*) = 0 – χ(*х*) – часткова характеристична функція множини *А*.

**Теорема 1.1.** Нехай *f*(*x*) – примітивно рекурсивна функція, *A* – примітивно рекурсивна множина. Тоді часткова функція *fч*(*х*) = *f*(*x*), якщо *x*∈*A* і невизначена, якщо *х*∉*А* є частково рекурсивною.

**Доведення**. Існує алгоритм, який обчислює її значення в точках, де вона визначена і працює нескінченно довго в точках, де вона невизначена.

function *fч*(*х*)

begin

*i* = 0

while χ*A*(*х*) ≠ 0 do

*i* = *i* + 1

*fч*= *f*(*x*)

end.

Всюди визначені частково рекурсивні функції називаються *загальнорекурсивними*.