Білет № 1(Слющинський)

1. Базові поняття алгоритмів та їхні складності. Оцінювання алгоритмів

Алгоритм - це є описана обчислювальна процедура, яка отримує дані на вхід і видає результат обчислень на вихід.

Оцінюються алгоритми за різними критеріями. **Розмір** - це деяке число, яке виражає розмір вхідних даних (розміром задачі про сортування може бути кількість елементів масиву, який треба посортувати). **Часова складність** - це час який витрачає алгоритм. **Асимптотична часова складність** - це є поведінка часової складності зі збільшенням розміру задачі. **Ємнісна складність** - це обсяг пам'яті, який потрібен алгоритму для роботи, а також, аналогічно, **асимптотична ємнісна складність**. **Асимптотична складність** алгоритму показує розмір задач, які цей алгоритм може розв'язувати.

2. Алгоритмічна система Тьюрінга. Машина Тьюрінга.

В працях Е. Поста і А. Тьюрінга сказано, що **алгоритмічні процеси** — це процеси, які може виконувати побудована для цього "машина". **Машина Тьюрінга** - це математична модель пристрою, який породжує обчислювальні процеси. Цю машину використовують для теоретичного уточнення поняття алгоритму та його дослідження.

Машина Тьюрінга (МТ) - це умовна машина, яка складається з трьох головних компонент: інформаційної стрічки, головки для зчитування і запису та пристрою керування. **Інформаційна стрічка** призначена для записування інформації (вхідна/вихідна/проміжна), яка виникає внаслідок обчислень. **Зовнішній алфавіт машини** - це алфавіт $S = \{s_1, s_2, \ldots, s_k\}$, він є фіксований для кожної машини. **Головка зчитування і запису** - це елемент, який читає і змінює комірки стрічки, ходячи по ній вліво і вправо. **Пристрій керування** керує всією роботою машини відповідно до програми обчислень, яка є задана. **Внутрішній алфавіт машини** - це алфавіт $Q = \{q_1, q_2, \ldots, q_n\}$ станів, він є фіксований для кожної машини. Пристрій керування перебуває в одному із цих станів в кожен момент часу.

Функціонує МТ дискретними кроками, кожен з яких має три операції:

1) символ s_i, на який вказує головка, замінюється іншим символом з алфавіту S;
2) головка або зсувається вправо/вліво на одну позицію, або залишається на місці
3) пристрій керування змінює свій стан q_i на інший стан з алфавіту Q.

3. Показати елементарність функції

$$f(x_1, x_2, x_3) = (x_1 + x_2) * x_3.$$

$= S^3(x, S^3(+, I_1^3(x_1, x_2, x_3), I_2^3(x_1, x_2, x_3)), I_3^3(x_1, x_2, x_3))$

Білет № 2(Слющинський)

1. Абстрактні алфавіти й алфавітні оператори. Кодувальні алфавітні оператори.

Абстрактний алфавіт - це довільна скінченна сукупність елементів. **Букви алфавіту** - можуть бути довільні, але повинні бути різні і кількість скінченною. **Слово** в певному алфавіті - це довільна скінченна впорядкована послідовність із букв даного алфавіту. **Довжина слова** - це кількість букв у слові.

Алфавітний оператор φ - це відповідність між словами в одному або різних алфавітах. **Приклад**: нехай $\{P\}_X$ - це деяка множина вхідних слів в алфавіті X, $\{Q\}_Y$ - множина вихідних слів в алфавіті Y. Алфавітним оператором φ буде функція такого вигляду $\varphi: \{P\}_X \to \{Q\}_Y$. **Однозначний оператор** - це оператор, який кожному вхідному слову у відповідність ставить не більше одного вихідного слова. **Багатозначний оператор** - ставить у відповідність декілька різних вихідних слів:

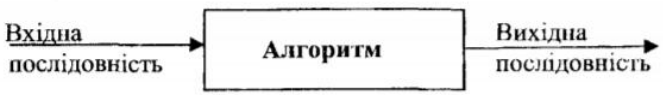
$$P \to \begin{cases} Q_1 \\ \vdots \\ Q_n \end{cases}$$

Нехай A - деякий алфавіт, який називають стандартним, B - довільний алфавіт. Нехай $\{L\}_A$,$\{M\}_B$ - множини слів у відповідних алфавітах. Тоді, множина $\{M\}_B$ закодована в алфавіті A, якщо задано такий однозначний оператор: $\varphi: \{M\}_B \to \{L\}_A$. Оператор φ називають **кодувальним**, а слова з $\{L\}_A$ - кодами об'єктів з $\{M\}_B$.

Кодування об'єктів алфавіту B словами однакової довжини називають **нормальним кодуванням**.

2. Важкорозв`язні задачі. Поліноміальні алгоритми та важкорозв`язні задачі.

Алгоритм - це такий собі "чорний ящик", який, за певною послідовністю отриманою на вхід, будує послідовність на вихід:



Можна вважати, що вхідна і вихідна послідовності складаються з 0 і 1, які кодують вхід і вихід алгоритму. Тоді алгоритм розглядають як послідовність елементарних двійкових операцій, таких як *або, і, не, друк* і тощо, що працюють з пам'яттю двійкових символів, яка може бути досить великою.

Алгоритм з поліноміальним часом - це алгоритм, у якого час роботи є обмежений зверху деяким поліномом Рк (n) (к - степінь полінома). Точні оцінки залежать від реалізації алгоритму.

Експоненціальні алгоритми - це алгоритми, оцінка яких зростає швидше, ніж поліном.

Легкорозв'язні задачі - це задачі**,** які мають поліноміальні алгоритми розв'язування. **Важкорозв'язні задачі** - це задачі, для яких не існує поліноміальних алгоритмів розв'язування.

3. За входом 0^n побудувати на другій стрічці машини Тьюрінга 0^{2n}.

$$(\lambda q_0, \lambda q_0) \to (\lambda q_1 R, \lambda q_1 R)$$
$$(0 q_0, \lambda q_0) \to (0 q_1 R, 0 q_1 R)$$

$$(\lambda q_1, \lambda q_1) \to (\lambda q_2 L, \lambda q_2)$$
$$(0 q_1, \lambda q_1) \to (0 q_1 R, 0 q_1 R)$$

$$(\lambda q_2, \lambda q_2) \to (\lambda q_F, \lambda q_F)$$
$$(0 q_2, \lambda q_2) \to (0 q_2 L, 0 q_2 R)$$

λ λ q0 -> λ λ q1 RH

0 λ q1 -> 00 q1 RR

λ λ q1 -> λ λ q2 LH

0 λ q2 -> λ 0 q2 LR

λ λ q2 -> λ λ qf HH

Білет № 3 (Ганушкевич)

1. Основні властивості алгоритмів. Способи запису алгоритмів.

Властивості:

1) Дискретність алгоритму. Алгоритм описує послідовний процес, який відбувається в дискретному часі. В кожен інтервал часу у системі величин за певними правилами виконують елементарні кроки алгоритму.
2) Ефективність алгоритму. Елементарні кроки алгоритму повинні бути ефективними, тобто виконуватися точно і за короткий відрізок часу.
3) Скінченність. Алгоритм завжди повинен закінчуватися після скінченної кількості кроків.
4) Результативність. Має забезпечувати отримання результату розв'язування задачі, що є наслідком скінченної кількості елементарних кроків та їхньої ефективності.
5) Масовість. Алгоритм можна застосувати до цілого класу задач, а не тільки до однієї.

Способи запису:

·**словесний** (запис на природній мові);

·**графічний** (зображення з графічних символів);

·**псевдокоди** (напівформалізовані описи алгоритмів на умовній алгоритмічній мові, що включають як елементи мови програмування, так і фрази природної мови, загальноприйняті математичні позначення та ін.);

·**програмний** (тексти на мовах програмування).

2. Недетерміновані машини Тьюрінга. Класи *P* та *NP.*

1) К-стрічковою недетермінованою машиною Тьюрінга М називають сімку (S, Q, q_0, q_F, I, Λ, σ), де значення компонент ті ж самі, що і для детермінованої машини Тьюрінга:

S — скінченна множина станів

Q — скінченна множина символів (алфавіт стрічки)

q_0 — початковий стан

q_F — кінцевий стан

Λ — символ пропуску

I — скінченна множина можливих станів

σ — багатозначне відображення з пари стан-символ, що називається функцією переходу.

У НМТ комбінація поточного стану автомата і символу на стрічці може допускати кілька переходів. Тобто на відміну від ДМТ, яка має єдиний «шлях обчислень», НМТ має «дерево обчислень». Вхідний ланцюжок приймається, якщо хоча б якась послідовність кроків для входу цього ланцюжка приводить до допустимого заключного стану.

Мовою L(M), яку розпізнає машина М, називають множину всіх ланцюжків, які допускає М. НМТ може мати часову(T(n)) складність - яка характеризує час, необхідний для виконання алгоритму на даній машині. Цей час, як правило, визначається кількістю операцій, які потрібно виконати для реалізації алгоритму. Або може мати ємнісну S(n) складність - яка характеризує пам'ять, необхідну для виконання алгоритму.

*2) Алгоритми, час роботи яких при розмірі входу n складає не більше $O(n^k)$, називаються **поліноміальними**. Клас поліноміальних задач позначається **P** і має властивість замкнутості..*

*Задачі, для яких існують перевірочні алгоритми, що працюють за поліноміальний час, утворюють клас **NP**. Це означає недетермінований поліноміальний час. Іншими словами, клас **P -** це клас задач, які можна розв'язати швидко, а клас **NP** це клас задач, розв'язок яких може бути швидко перевірений. Такі класи допускають детерміновану(P) та недетерміновану(NP) машини Тюрінга.*

3. Довести примітивну рекурсивність функції: f(x,y,z,v)=x*y+z*v.

Доведення примітивної рекурсивності додавання:

$$f_+(x, y) = x + y$$

$$g(x) = x;\ h(x, y, z) = z + 1;$$

$f_+(x, 0) = g(x) = x$ - тотожна функція $I_1^1(x)$

$$f_+(x, 1) = h(x, 0, x) = x + 1$$

$$f_+(x, 2) = h(x, 1, f_+(x, 1)) = h(x, 1, x + 1) = x + 2$$

$$f_+(x, y-1) = h(x, y-2, f_+(x, y-2)) = h(x, y-2, x+y-2) = x+y-1$$
$$f_+(x, y) = h(x, y-1, f_+(x, y-1)) = h(x, y-1, x+y-1) = x+y$$
$$h(x, y, z) = S(I_3^3(x, y, z)) = z+1$$
$$f_+(x, y) = R^1(I_1^1(x), h(x, y, z))$$

Доведення примітивної рекурсивності множення:

$$f_*(x, y) = x * y$$

$$f_*(x, 0) = O(x)$$
$$f_*(x, 1) = h(x, 0, f_*(x, 0)) = x + O(x) = x$$
$$f_*(x, y-1) = h(x, y-2, f_*(x, y-2)) = x + xy - 2x = xy - x$$
$$f_*(x, y) = h(x, y-1, f_*(x, y-1)) = x + xy - x = xy$$
$$h(x, y, z) = f_+(I_1^3(x, y, z), I_3^3(x, y, z))$$
$$f_*(x, y) = R^1(O(x), h(x, y, z))$$

Можемо зробити висновок, що задана функція також примітивно рекурсивна, тому що примітивно рекурсивними є функції додавання і множення

Білет № 4 (Бугай)

1. Різновиди алгоритмів. Композиція алгоритмів.

Ознаки поділу алгоритмів за властивостями

Детермінованість (в визначені алгоритму алфавітний оператор однозначний, тоді алгоритм детермінований, інакше недетермінований)

Самозмінність (самозмінний, якщо в процесі переробки алгоритмом вхідних слів система P змінюється, в іншому випадку він не самозмінний)

Самозастосовність (самозастосовний, якщо слово p^{cod} входить в область визначення алгоритму, інакше не самозастосовний)

Універсальність(універсальний, якщо він еквівалентний довільному наперед заданому алгоритму)

Композиція алоритмів

1. Суперпозиція алгоритмів A і B

При суперпозиції двох алгоритмів A і B вихідне слово одного з них розшлядають як вхідне слово іншого

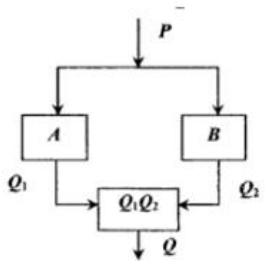
C(P) = B(A(P)), P є D(A), A(P) с D(B)

Суперпозиція може виконуватись для довільної скінченної к-сті алгоритмів

2.

Об’єднання алгоритмів

$$P \in D(A) \cap D(B),\ C(P) = A(P)B(P)$$



На всіх інших словах алгоритм С вважається невизначеним

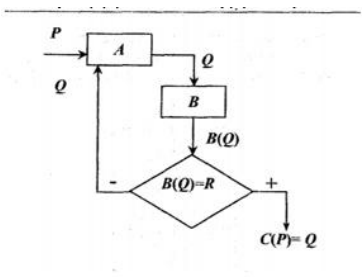
3. Розгалуження алгоритмів

Розгалуженням називають композицію трьох алгоритмів A,B,C, задану співвідношенням

$$F(P)\begin{cases} A(P), \text{якщо } C(P) = R \\ B(P), \text{якщо } C(P) \neq R \end{cases}$$

4. Ітерація алгоритмів

Ітерацією двох алгоритмів A та B наз. Алгоритм C, який визначають так: для довільного вхідного слова P вихідне слово C(P) отримують як результат послідовного багаторазового застосування алгоритму A, доки не буде отримано слово, перетворене алгоритмом B в деяке фіксоване слово R



2. Формальне визначення МТ. Теза Тьюрінга.

МТ – певний алгоритм для переробки вхідних слів. Кожна МТ реалізує ф-ю

$\sigma: S \times \{Q \backslash q_F\} \to S \times Q \times Z$

МТ обчислює деяку словникову ф-ію $\varphi(\mathrm{P})$, яка відображає слово з деякої множини слів у не порожні слова, якщо для довільного слова P процес МТ(P) досягає заключної конфігурації асоційованої зі словом $\varphi(\mathrm{P})$ – результатом ф-ії

Якщо для часткової словникової ф-ії існує МТ, яка її обчислює, то ф-ію називають обчислювальною за Тюрінгом.

Теза Тюрінга

Для довільного алгоритму A = <φ, P> у довільному скінченному алфавіті X існує ф-ія $\varphi(\mathrm{P})$: $\{\mathrm{P}\}_{\mathrm{x}} \to \{Q\}_{\mathrm{x}}$ обчислювана за Тюрінгом.

Будь-який алгоритм заданий у довільній формі можна замінити еквівалентною йому МТ

Теорема 1

Всі часткові словникові ф-ії обчислювані за Тюрінгом є частково-рекурсивні

Теорема 2

Для кожної частково-рекурсивної словникової ф-ії визначеної на алфавіті $\mathrm{A} = \{\mathrm{a}_1, \ldots \mathrm{a}_n\}$ існує МТ з відповідним внутрішнім алфавітом, яка обчислює задану ф-ію.

3. Довести примітивну рекурсивність функції цілочисельного ділення.

$$h(x,y) = h(x-1, y-1)$$
$$h(0,y) = 1$$
$$h(x,0) = 0$$
$$g(x,y) = x - y$$
$$f(z,x,y) = x \div y$$
$$f(0,x,y) = 0$$
$$f(1,x,y) = 1 + f\big(h\big(g(x,y)\big), g(x,y), y\big)$$

або

$$div(x,y) = \begin{cases} div(x,0) = g(x) = O^1(x), \\ div(x,y+1) = h\big(x,y,div(x,y)\big) = \\ = S^3(add, div(x,y), S^2(notsg, S^3(mod, I_1^1(x), S^1(mod(x,y))))) \end{cases}$$

Білет № 5 (SLAVA)

1. Поняття про алгоритмічні системи.
2. Метод “Поділяй і володарюй”.
3. Створити МТ для обчислення функції $I_4^5(x_1, x_2, x_3, x_4, x_5) = x_4$, якщо x_1, x_2, x_3, x_4, x_5– числа записані в алфавіті A= {|}, а між числами x_1, x_2, x_3, x_4, x_5 ставиться *.

Питання 1

$A = < \varphi, \rho >$- деякий алгоритм із системою правил .

Алгоритм $A_f = < \varphi, \rho_f >$ де ρ_f формальний опис системи правил ρ – формальний еквівалент алгоритму А.

Алгоритмічною системою називають спосіб задання алгоритмів у деякому фіксованому формалізмі F, який дозволяє для довільного алгоритму А задати формальний еквівалент A_f .

Алгоритмічні системи є формалізацією поняття алгоритму. Алгоритмічні системи діляться на типи відповідно до визначення алгоритму: Приклади систем: машина Тюрінга, нормальні алгоритми Маркова, рекурсивні функції.

Питання 2

“Поділяй і володарюй” - це метод розробки алгоритмів. Зазвичай алгоритми цього виду є рекурсивними. Такі алгоритми часто розбивають задачу на менші, допоміжні задачі, які стосуються основної і рекурсивно викликають себе для їх вирішення. Після вирішення допоміжних задач їхній розв’язок комбінується для вирішення основної задачі. Прикладом алгоритму такого виду є сортування злиттям. Цей алгоритм ділить масив навпіл і викликає себе для кожної з половин і так доки розмір половин не стане 1, опісля посортовані половини зливаються і отримуємо результат.

Питання 3

$q_0| \rightarrow | \, q_0 R$

$q_0 * \rightarrow * \; q_0 R$

$q_0 \, \lambda \rightarrow \lambda \, q_1 L$

$q_1 | \rightarrow \lambda \, q_1 L$

$q_1 * \rightarrow \lambda \, q_2 L$

$q_2| \rightarrow | \, q_2 L$

$q_2 * \rightarrow \lambda \, q_3 L$

$q_3 * \rightarrow \lambda \, q_3 L$

$q_3| \rightarrow \lambda q_3 L$

$q_3 \lambda \rightarrow \lambda \, q_f S$

_ q0 -> _ q0 R

| q0 -> _ q0 R

*** q0 -> _ q1 R**

| q1 -> _ q1 R

*** q1 -> _ q2 R**

| q2 -> _ q2 R

*** q2 -> _ q3 R**

| q3 -> | q3 R

*** q3 -> _ q4 R**

| q4 -> _ q4 R

_ q4 -> _ qf R

_ - це лямбда

Білет № 6(Васільєва)

1. Система нормальних алгоритмів Маркова. Принцип нормалізації.

Нормальні алгоритми Маркова — це система послідовних застосувань підстановок, які реалізують певні процедури отримання нових слів із базових, побудованих із символів деякого алфавіту.

Нормальний алгоритм Маркова — це упорядкована множина продукцій P1, P2, …, Pn

У свою чергу **простою продукцією** (формулою підстановки) назив. запис « u —> w ».

У цьому разі V не містить символів « —> ».

Заключною продукцією (заключною підстановкою) назива-ють запис вигляду

u, w - рядки в алфавіті V.

Формула « u —> w » може бути застосована до рядка Z є V, якщо є хоча б одне входження *u* в Z. В іншому випадку ця формула не застосовна до рядка Z.

Отже, виконання алгоритму починається з перевірки першої продукції. Якщо застосовна до рядка — то перетворюється, якщо ні — перехід до наступної.

Алгоритм Маркова завершується в одному з двох випадків:

- до рядка не може бути застосована жодна з наявних формул підстановок;
- до рядка застосована заключна підстановка.

Система норм.алгоритмів Маркова задовольняє умову строгості адже наявне розпізнавання входжен-ь і підстановка .

Марков також сформулював і довів необхідну умову універ-сальності.

Теорема. Для того, щоб реалізувати в схемах нормальних алгоритмів довільний алгоритм A = { φ , P } , необхідно, щоб у системі нормальних алгоритмів були як звичайні, так і заключні підстановки.

А. Марков у своїй тезі запропонував, що наявність у системі нормальних алгоритмів заданих двох видів підстановок є не тільки необхідною, а й достатньою умовою універсальності системи.

Принцип нормалізації. Для будь-якого алгоритму А —{р, Р} в довільному алфавіті Х можна побудувати еквіва-лентний йому нормальний алгоритм над алфавітом Х.

2.Недетерміновані машини Тьюрінга.

НМТ має скінченну кількість можливих кроків. Можемо сказати, що **недетермінована машина Тюрінга** — машина Тюрінга, функція переходу якої являє собою недетермінований скінченний автомат.

Вхідний ланцюжок *x допускається*, якщо принаймні одна послідовність кроків для входу *x* приводить до допустимого заключного стану. За заданого вхідного ланцюжка *x* можна вважати, що недетермінована машина Тьюрінга *M* паралельно виконує всі можливі послідовності кроків, доки не досягне допустимого заключного стану або доки не виявиться, що подальші кроки неможливі.

к-стрічковою педетермінованою машиною Тьюрінга М називають сімку

$$(S, Q, q_0, q_F, I, \Lambda, \delta),$$

де значення всіх компонент ті ж, що і для детермінованої машини Тьюрінга окрім того, що

δ ф-я переходів є відображенням множини $\{Q \setminus q_F\} \times S^k$

в множину підмножин у $Q \times (S \times \{L, R, H\})^k$.

3. Скласти нормальний алгоритм, що виконує збільшення десяткового числа на 2

(див. білет 20)

Білет № 7(Васільєва)

1. Принцип нормалізації. Універсальний нормальний алгоритм.

Алгор. система повинна бути математично строгою та універсальною
Марков сформулював і довів необхідну умову універ-сальності(у системі норм. Алгоритмів Маркова наявне розпізнавання входжен-ь і підстановка, вона задовольняє умову строгості).

Теорема. Для того, щоб реалізувати в схемах нормальних алгоритмів довільний алгоритм A = { φ , P } , необхідно, щоб у системі нормальних алгоритмів були як звичайні, так і заключні підстановки.
Наявність у системі нормальних алгоритмів заданих двох видів підстановок є не необхідною й достатньою умовою універсальності за Марковим.

Принцип нормалізації формується наступним чином: для будь-якого алгоритму А —{р, Р} в довільному алфавіті Х можна побудувати еквіва-лентний йому нормальний алгоритм над алфавітом Х.
В іншому випадку алгоритм не нормалізований.
Універсальним нормальним алгоритмом (УНА) називають алгоритм,здатний виконувати роботу нормального алгоритму *А.*
УНА виконує над вхідним словом *Р* підстановки згідно з схемою інформації конкретного алгоритму *А та цього слова Р.*
УНА, будучи нормальним алгоритмом, теж має один стандартний фіксований алфавіт.Щоб він міг сприймати схему, вона повинна бути закодована в стандартному алфавіті УНА.
А.Марков довів наступну теорему про універсальний нормальний алгоритм:
Теорема. Існує такий нормальний алгоритм U, який нази-вають універсальним, що для будь-якого нормального алгоритму А і будь-якого вхідного слова Р з області визначення *А* перетворює наступне зха допомогою конкатенації зображень *А* і *Р*

$$A^{cod} P^{cod} \xrightarrow{U} A(P)^{cod}$$

де Q^{cod}, *де* $Q = A(P)$

2. Рекурсивні функції.

Рекурсивні функції — клас функцій, введений як уточнення класу обчислюваних функцій. Числові функції, значення яких можна обчислити за допо-
могою деякого (єдиного для заданої функції) алгоритму, назива-ють обчислюваними функціями.
Найпростіші функції - наступності, тотожно рівно 0, тотожню.
Існують також примітивно- рекурсивні, загально рекурсивні та частково-рекуурсивні ф-ї.

Задано деякий алгоритм *A,* застосовний до цілого класу задач, тобто до ряду допустимих вхідних даних - умов задач. Нумеруємо ці умови цілими числами

$n_1, n_2, \ldots, n_k, \ldots$. Результати роботи алгоритму також пронумеруємо цілими додатними числами $m_1, m_2, \ldots, m_k, \ldots$. Тоді

$$m_i = A(n_j),$$

та оскільки m_i та n_j – числа, то алгоритм A визначає деяку числову функцію φ

$$m_i = \varphi(n_j), (i=1,2,\ldots, j=1,2,\ldots)$$

де $\varphi: N \to N$.

Виконання алгоритму *A* є еквівалентним обчисленню значень деякої числової функції φ.

3. Довести, що функція $f(x_1, x_2) = x_1 \div x_2$ є примітивно рекурсивною.

Білет № 8 (Сипко)

1. Рекурсивні функції. Зведення довільних алгоритмів до числових функцій.

Історично першою алгоритмічною системою була система, що ґрунтується на використанні конструктивно визначених арифметичних (цілочислових) функцій, які назвали **рекурсивними функціями**.

Нехай задано деякий алгоритм A, який можна застосувати до цілого класу задач, тобто до ряду допустимих вхідних даних - умов задач. Пронумеруємо ці умови цілими числами $n_1, n_2, \dots, n_k, \dots$. Результати роботи алгоритму також пронумеруємо цілими додатними числами $m_1, m_2, \dots, m_k, \dots$. Тоді $m_i = A(n_j)$, та оскільки m_i та n_j - числа, то алгоритм A визначає деяку числову функцію φ

$m_i = \varphi(n_j), (i = 1,2,\dots, j = 1,2,\dots)$, де $\varphi : N \rightarrow N$.

Отже, виконання довільного алгоритму A є еквівалентним обчисленню значень деякої числової функції φ. Числові функції, значення яких можна обчислити за допомогою деякого (єдиного для заданої функції) алгоритму, називають **обчислюваними функціями**.

2. Класи *P* та *NP*.

Клас $\boldsymbol{P - TIME}$ (або просто P) - множина всіх мов, які допускають ДМТ з поліноміальною часовою складністю.

Клас $\boldsymbol{NP - TIME}$ (або просто $\boldsymbol{NP}$) - це множина всіх мов, які допускають НМТ з поліноміальною часовою складністю.

Інтуїтивно клас $\boldsymbol{P}$ уявимо як клас задач, які можна швидко розв'язати, а $\boldsymbol{NP}$ - як клас задач, розв'язок яких можна швидко перевірити.

Якщо мова $\boldsymbol{L}$ належить до класу $\boldsymbol{NP}$, то її розпізнає деяка ДМТ з часовою складністю $k^{P(n)}$, де k - стала, $P(n)$ - поліном, залежний від L.

Не доведено, чи існують мови з $\boldsymbol{NP}$, які не належать $\boldsymbol{P}$. Отже, не відомо, чи є $\boldsymbol{P}$ власним підкласом класу $\boldsymbol{NP}$.

Мову L_o з $\boldsymbol{NP}$ називають **повною для недетермінованого поліноміального часу** (або $\boldsymbol{NP}$**-повною**), якщо за заданим детермінованим алгоритмом розпізнавання L_o з часовою складністю $T(n) \geq n$ і довільною мовою L з $\boldsymbol{NP}$, можна ефективно знайти детермінований алгоритм, який розпізнає L за час $T(P_L(n))$, де P_L — деякий поліном, що залежить від L. Говорять, що L (поліноміально) зводиться до L_o.

Отже, $\boldsymbol{NP}$ - повноту мови L_o можна довести, довівши, що L_o належить $\boldsymbol{NP}$ і кожну мову $L \in \boldsymbol{NP}$ можна "поліноміально трансформувати" в L_o .

Мову $\boldsymbol{L}$ називають поліноміально трансформовною в L_o, якщо деяка ДМТ M з поліноміально обмеженим часом роботи перетворює кожен ланцюжок w в алфавіті мови $\boldsymbol{L}$ в такий ланцюжок w_o в алфавіті мови L_o, що $w \in \boldsymbol{L}$ тоді і лише тоді, коли $w_o \in L_o$.

Існує алгоритм, який визначає, чи належить w до мови $\boldsymbol{L}$, за час $P(|w|) + +T(P(|w|)) < 2T(P(|w|))$. Якби функція T була поліномом (тобто мова L_o належала б до $\boldsymbol{P}$), то цей алгоритм, який розпізнає L, працював би поліноміально обмежений час, і мова L також належала б до $\boldsymbol{P}$.

Можна також визначити мову L_o як $\boldsymbol{NP}$ - повну, якщо вона належить до $\boldsymbol{NP}$ і кожна мова з $\boldsymbol{NP}$ поліноміально трансформується в L_o.

Якщо деякий детермінований алгоритм розпізнає L_o за поліноміальний час, то всі мови з $\boldsymbol{NP}$ можна розпізнати за поліноміальний час. Отже, або всі $\boldsymbol{NP}$ - повні мови належать до $\boldsymbol{P}$, або жодна з них не належить до $\boldsymbol{P}$. Перше справджується тоді й лише тоді, коли $\boldsymbol{P} = \boldsymbol{NP}$.

3. Скласти нормальний алгоритм, який обчислює частку і залишок від ділення на 7 в унарній системі числення.

$$a1111111 \rightarrow 1a$$
$$a \rightarrow .\lambda$$
$$1 \rightarrow a1$$

Білет № 9(Mipai)

1. Обчислювані функції. Найпростіші функції. Головні оператори.
2. Метод гілок і меж.
3. Заданий алфавіт A=(c,d,e,f). Символи у вхідному слові відсортувати по алфавіту.

Питання 1

Обчислювані функції - це числові функції, значення яких можна обчислити за допомогою якогось одного алгоритму.

Нехай N – множина всіх натуральних чисел, $N^{(n)}$ – множина всеможливих n натуральних чисел. Найпростіші функції:

- Функція наступності($S^1(x)$) : $\varphi: N \longrightarrow N$, якщо $\varphi(x) = x + 1$
- Нуль-функція($O^n(x_1, \dots, x_n)$): $\varphi: N^{(n)} \longrightarrow N$, якщо $\varphi(x_1, \dots, x_n) = 0$
- Тотожна функція($I_i^n(x_1, \dots, x_n)$): $\varphi_i: N^{(n)} \longrightarrow N$, якщо $\varphi_i(x_1, \dots, x_n) = x_i$, $(1 \le i \le n)$

Найпростіші функції є всюди визначені.

Для побудови частково-рекурсивних функцій використовуються три основні оператори:

- Суперпозиції. Нехай:

  $$g^m(x_1,\dots,x_m) = f^n(f_1^m(x_1,\dots,x_m),\dots,f_n^m(x_1,\dots,x_m))$$

  f^n, f_n^m, g^m – це деякі функції. Тоді оператор, який дасть нам g^m називають операторм суперпозиції і позначають $\mathbb{S}^{n+1}(f^n, f_1^m, \dots, f_n^m)$, де n+1 – кількість функцій.

- Примітивної рекурсії. Розглянемо функцію $f^{n+1}: N^{(n+1)} \to N$, яка визначена наступним чином:

  $$\begin{cases} f(x_1,\dots,x_n,0) = g(x_1,\dots,x_n), \\ f(x_1,\dots,x_n,y+1) = h(x_1,\dots,x_n,y,f(x_1,\dots,x_n,y)) \end{cases}$$

  Отже, оператор примітивної рекурсії($\mathbb{R}$) дасть можливість з g, h побудувати функцію f.

- Якщо з функції $f(x_1, \dots, x_n)$ можна утворити функцію $\mu_y(f(x_1, \dots, x_{n-1}, y) = x_n)$, то оператор, який дає таку можливість називають оператором мінімізації і позначають $\mathbb{M}$:

  $$\mu_y(f(x_1, \dots, x_{n-1}, y) = x_n) = \mathbb{M}f$$

Питання 2

Метод гілок і меж широко застосовується при оптимізації. Даний метод має справу із деревоподібними моделями простору розв'язків задачі і допомагає знайти найоптимальніший розв'язок серед усіх можливих. Якщо розглянути множину можлививх розв'язків задачі, то головна властивість, яку повинна мати ця множина є її поділ на підмножини, які не перетинаються. Відповідно для цих частин можна провести оцінку «оптимальності» можливого розв'язку. Варто зауважити, що так званої «оптимальності» на цій підмножині може і не бути. На практиці, коли постає питання розв'язку якоїсь задачі, то можливі варіанти розв'язку розбивають на класи. Потім виконують оцінку для всіх розв'язків з одного класу і якщо вона більша за попередню, то відкидають усі варіанти з цього класу.

Питання 3

V = {c, d, e, f}

$P_1: 'dc' \to 'cd'$

$P_2: 'ec' \to 'ce'$

$P_3: 'fc' \to 'cf'$

$P_4: 'ed' \to 'de'$

$P_5: 'fd' \to 'df'$

$P_6: 'fe' \to 'ef'$

Приклад: $dcedf \to cdedf \to cddef$ (сортування dcedf)

Білет № 10 (Бугай)

1. Оператор примітивної рекурсії.

Нехай задані часткові числові функції $g^n: N^{(n)} \to N$;

$$h^{n+2}: N^{(n+2)} \to N$$

Розглянемо часткову функцію $f^{n+1}: N^{(n+1)} \to N$, визначену так:

$$\begin{cases} f(x_1,\dots,x_n,0) = g(x_1,\dots,x_n), \\ f(x_1,\dots,x_n,y+1) = h(x_1,\dots,x_n,y,f(x_1,\dots,x_n,y)), \end{cases} \quad (2.5)$$

для довільних натуральних значен $x_1, \dots x_n, y$.

У випадку одномісної функції $f^1: N \to N$:

$$\begin{cases} f(0) = a, \\ f(y+1) = h(y, f(y)), \end{cases} \quad (2.6)$$

Де $a \in N$

Оператор, який за формулами (2.5) або (2.6) з функцій g і h дає змогу побудувати функцію f **називають оператором примітивної рекурсії** і позначають R : $f = \mathbb{R}(g,h)$.

Усі три функції g,h,f є числовими (з однією й тією ж областю визначення і значення), тому їхня суперпозиція завжди можлива, що зумовлює існування f^{n+1}, для будь-якої пари g^n, h^{n+2}

Так само, як і у випадку оператора суперпозиції, оператор примітивної рекурсії задає часткову функцію $R : F_{\text{ч.р}} \to F_{\text{ч.р}}$

2. Динамічне програмування.

Поняття введене для характеристики алгоритмів, залежно від зміни обставин.

З точки зору математичної оптимізації, динамічне програмування полягає в спрощенні знаходження загального оптимального розв'язку, шляхом пошуку розв'язків в підзадачах, отриманих розбиттям задачі на послідовні проміжки часу. Це виражається в визначенні послідовності **значень функцій** V_1, V_2, ..., V_n, з аргументом *y*, котрий позначає стан системи в моменти часу *i* від 1 до *n*. Визначенням $V_n(y)$ є значення, отримане в стані *y* в кінцевий момент часу *n*. Значення V_i в попередні моменти часу $i = n-1, n-2, ..., 2, 1$ можуть бути знайдені рухаючись назад, використовуючи рекурсивну залежність, названу **рівняння Беллмана.**

Процес розробки алгоритмів динамічного програмування можна розбити на чотири етапи .

1. Опис структури оптимального розв'язку.

2. Рекурсивне визначення значення, що відповідає оптимальному розв'язку.

3. Обчислення значення, іцо відповідає оптимальному розв'язку, за домогою методу висхідного аналізу.

4. Утворення оптимального розв'язку на підставі інформації, отриманої на попередніх етапах

Для того, щоб до задачі оптимізації можна було застосувати метод динамічного програмування, у ній повинні бути такі складові: оптимальна підструктура та допоміжні програми, що перекриваються.

Оптимальна підструктура виявляється в задачі в тому випадку, якщо в її оптимальному розв'язку містяться оптимальні розв'язки допоміжних підзадач. Друга складова частина полягає в тому, що простір допоміжних задач повинен бути "невеликим" у тому сенсі, що внаслідок виконання рекурсивного алгоритму одні й ті ж допоміжні задачі розв'язуються знову і знову, а нові підзадачі не виникають

3. Заданий алфавіт A=(k,l,m,n). Всі літери m і n замінити порожнім символом, перше входження ll замінити на nnn.

$$m \to \lambda$$
$$n \to \lambda$$
$$ll \to .nnn$$

Білет № 11 (SLAVA)

1. Примітивно-рекурсивні функції.

2. Алгоритмічна система Тьюрінга. Машина Тьюрінга.

3. Заданий алфавіт A=(c,d,e,f). Всі літери c і d замінити порожнім символом, перше входження f замінити на ee.

Питання 1

Функцію називають примітивно-рекурсивною, якщо її можна отримати застосуванням скінченної кількості найпростіших функцій $(S^1, O^1 I_m^n)$, операторів суперпозиції та примітивної рекурсії (в довільній послідовності). Примітивно-рекурсивні функції є всюди визначеними.

Також бувають примітивно рекурсивні функції відносно певної системи, наприклад: дано сукупність функцій $\delta = \{f_1^{n_1}, \dots, f_k^{n_k}\}$.

Функції, які отримують з системи δ, скінченної кількості найпростіших функцій, операторів суперпозиції і рекурсії називають примітивно рекурсивними відносно системи δ.

Питання 2

Машина Тюрінга – це математична модель пристрою, що здатний імітувати будь-які відомі алгоритмічні процеси. Використовується для теоретичного дослідження поняття алгоритму.

Будова:

Машина Тюрінга є умовною машиною, що складається з таких компонентів :

- Інформаційна стрічка – нескінченна стрічка, призначена для записування вхідної, вихідної і проміжної інформації.
- Головка для зчитування і запису – пристрій, що рухається по стрічці і здатний зчитувати символи з комірок і змінювати їх.
- Пристрій керування – логічний блок, що знаходиться в певному стані з множини станів (внутрішнього алфавіту).

На кожному кроці роботи машина змінює стан на якийсь з множини вн. станів. При цьому машина може змінити символ в комірці, на яку вказує головка, і посунути головку на 1 вправо або вліво або залишити на місці. Всі ці дії задаються правилами (інструкціями для машини).

Питання 3

λ – порожній символ

$c \to \lambda$

$d \to \lambda$

$f \to .ee$

Білет № 12(Ганушкевич)

1. Частково-рекурсивні функції. Теза Черча.

1) $\sigma = \{ f_1^{n1}, \dots, f_1^{nk} \}$

Часткову функцію f називають частково-рекурсивною відносно σ , якщо її можна отримати з функцій системи σ і найпростіших функцій із застосуванням скінченної кількості операторів суперпозиції, примітивної рекурсії та мінімізації. АБО її можна отримати з найпростіших функцій із застосуванням скінченної

кількості операторів суперпозиції, примітивної рекурсії та мінімізації:

$$\{S^1, O^1, I_m^n, \sigma\} \xrightarrow{S^{n+1},\ R,\ M} f \quad \text{АБО} \quad \{S^1, O^1, I_m^n\} \xrightarrow{S^{n+1},\ R,\ M} f$$

Тут дані операції можна застосовувати будь-яку кількість разів та у будь-якій послідовності.

Одне з головних понять теорії алгоритмів - клас частково-рекурсивних функції ($К_{ч.р.}$) - найзагальніший клас конструктивно визначених арифметичних функцій.

Кожну задану частково-рекурсивну функцію можна обчислити за допомогою певного алгоритму; крім того, які б класи точно визначених алгоритмів не будували, у всіх випадках числові функції, обчислювані за алгоритмами з цих класів, є частково-рекурсивними(гіпотеза Черча).

2) *Теза Черча. Клас алгоритмічно (механічно) обчислювальних функцій збігається з класом частково-рекурсивних функцій.*

Теза не доводиться (бо у її формулюванні використане інтуїтивне поняття алгоритму). Водночас доведено, що алгоритм тоді і тільки тоді може бути нормалізований, коли він може бути реалізований за допомогою частково-рекурсивних функцій: $К_А \cong К_{ч.р.}$ $К_Н = К_{ч.р.}$

$К_Н$ *- клас нормальних алгоритмів.*

$К_А$ *- клас алгоритмічно (механічно) обчислювальних функцій*

$К_{ч.р.}$ *- клас частково-рекурсивних функцій.*

2. Методи розробки ефективних алгоритмів.

Я наведу приклади декількох основних методів:

- ***Метод часткових цілей**. Зведення важкої або великої задачі до декількох простіших задач. У цьому разі розв'язок початкової задачі отримують з розв'язків легших задач(не існує загального набору правил для визначення класу задач, які можна розв'язати за допомогою такого підходу).*
- ***Метод підйому**. Починається з прийняття початкового припущення або обчислення початкового розв'язку задачі. Потім відбувається якнайшвидший рух "вверх" від початкового розв'язку в напрямі до ліпших розв'язків. Коли алгоритм досягає такої точки, з якої більше неможливо рухатися вверх, алгоритм зупиняється. У цьому разі, немає жодних гарантій, що кінцевий розв'язок буде оптимальним - ми отримаємо лише наближений розв'язок.*
- ***Метод відпрацювання назад**. Його починають з цілі чи розв'язку і рухаються назад за напрямом до початкового формулювання задачі. Далі, якщо ці дії оборотні, рухаються знову від формулювання задачі до розв'язку. Цей метод широко застосовують під час розв'язування різноманітних головоломок.*
- ***Рекурсія**. Процедура, яка прямо чи опосередковано звертається до себе. Застосування рскурсії часто дає змогу побудувати зрозуміліші та стислі алгоритми, ніж це було б зроблено без неї. Рекурсія сама по собі не приводить до ефективнішого алгоритму. Однак у поєднанні з іншими методами, наприклад "поділяй і володарюй", дає алгоритми, одночасно ефективні й елегантні.*

Це фундаментальні методи, з використанням яких побудовані класичні алгоритми.

3. Написати програму для машини Тьюрінга, що додає два числа в унарній системі числення.

_ q0 -> _ q0 R

1 q0 -> _ q1 R

1 q1 -> 1 q1 R

+ q1 -> 1 q2 L

1 q2 -> 1 q2 L

_ q2 -> _ qf H

Білет № 13(Ланчевич)

1. Загальнорекурсивні функції. Теза Тьюрінга.

Для початку розглянемо оператор слабкої мінімізації, який ставить у відповідність довільній заданій функції часткову функцію:

$$M^1 f = \begin{cases} Mf, & \text{якщо } Mf \text{ всюди визначена;} \\ \text{не визначена,} & \text{якщо } Mf \text{ визначена не всюди.} \end{cases}$$

Функції, які можна отримати з найпростіших функцій S^I,O^I ,I^n_m за допомогою оператора примітивної рекурсії, суперпозиції та слабкої мінімізації, називають **загальнорекурсивними**.

Усі загальнорекурсивні функції є всюди визначені.

Усі загальнорекурсивні функції є всюди визначеними частково-рекурсивними функціями.

Функцію f називають **алгоритмічно обчислюваною відносно деякої системи функцій** якщо існує алгоритм, який дає змогу обчислити значення функції f за умови, що якимось чином можна знайти ті значення функцій , які потрібні для алгоритму.

Теза Тьюрінга . Клас функцій, алгоритмічно обчислюваних відносно деякого класу функцій σ , збігається з класом частково - рекурсивних функцій відносно σ .

2. Базові поняття алгоритмів та їхні складності. Оцінювання алгоритмів.

Алгоритм - це формально-описана обчислювальна процедура, яка перетворює вхідні дані(аргумент) і видає результат обчислень на виході.

Оцінюють алгоритми за різними критеріями. **Розмір** задачі - це обсяг вхідних даних, які потрібні для опису задачі. Наприклад, розміром задачі про сортування може бути кількість елементів масиву, який треба посортувати. **Часова складність** - це час витрачений алгоритмом. **Асимптотична часова складність** - це є поведінка часової складності зі збільшенням розміру задачі. **Ємнісна складність** - це обсяг пам'яті, який потрібен алгоритму для

роботи, а також, аналогічно визначається **асимптотична ємнісна складність**. Асимптотична складність алгоритму показує розмір задач, які цей алгоритм може розв'язувати.

3. Створити МТ, яка перетворювала б будь-яке натуральне число k, записане в алфавіті A = {|} в число k+3. Головка МТ нехай стоїть спочатку проти лівого крайнього символу {|} в стані q_0.

|, q_0 -> |, q_0, R

λ, q_0 -> |, q_1, R

λ, q_1 -> |, q_2, R

λ, q_2 -> |, q_f, N

Білет № 14 (Пенькова)

1. Алгоритмічна система Тьюрінга. Машина Тьюрінга.
2. Метод "Поділяй і володарюй".
3. Довести елементарність функції $f(x_1, x_2) = (x_1 + x_2)^2$

1) Алгоритмічна система Тьюрінга. Машина Тьюрінга

Алгоритмічна система, запропонована Тьюрінгом і Постом в 1936-37 роках, полягає в тому що алгоритмічні процеси — це процеси, які може виконувати відповідно побудована "машина". Ця машина може виконувати в кожний окремий відрізок часу лише одну примітивну операцію. Зараз цю машину називають Машиною Тьюрінга. Кожна окрема машина Тьюрінга відображає процес розв'язку однієї конкретної задачі.

Машина Тьюрінга є деякою умовною машиною і складається з трьох головних компонент:

· *Інформаційна стрічка* – на ній записують вхідну, проміжну та вихідну інформацію. Вона є безмежною в обидва боки і складається з так би мовити комірок, в які поміщається по 1 символу. Множина символів для кожної задачі фіксована, її називають *зовнішнім алфавітом машини*. S = {s1, …, sn}

· *Головка зчитування і запису* – ця головка рухається по комірках стрічки і здатна зчитувати або змінювати вміст комірки, на яку вона вказує.

· *Пристрій керування* - керує всією роботою машини відповідно до заданої програми обчислень. У кожний момент часу t пристрій перебуває в одному зі станів. Станів, як і символів, є фіксована кількість для кожної МТ. Множина цих станів називається *внутрішнім алфавітом машини*

Q = {q1,...,qk } . В множині Q виділяють два особливі стани: початковий q0 і заключний qf. Вони відповідають за пуск і за зупинку машини.

Роботу машини Тьюрінга формально можна описати за допомогою конфігурацій.

Перехід МТ від конфігурації Kt до наступної конфігурації К," виконується за один крок, ініціалізований відповідною командою. Команда виглядає так:

$s_i\ q_l$ -> $s_j\ q_r\ Z$, де

s_i – символ на який вказує головка, замінюється іншим символом s_j

пристрій керування змінює свій стан q_l на інший стан q_r

Замість Z може бути R, L або H – напрям, куди зсунеться головка після виконання команди: вправо, вліво або залишиться на місці.

Якщо ж заключна конфігурації K_{tf} не досяжна, то МТ працює безмежно довго без зупинки (зациклюється). Вважають, що така МТ незастосовна до початкової конфігурації K_{t0} .

2) Метод "Поділяй і володарюй"

Для початку потрібно ввести термін рекурсія.

Рекурсія - це визначення об'єкта через самого себе, тобто якщо ми рекурсивно визначаємо функцію, то вона викликає саму себе, безпосередньо (в своєму тілі) або опосередковано (через іншу функцію).

Поділяй і володарюй - важлива парадигма розробки алгоритмів, що полягає в рекурсивному розбитті розв'язуваної задачі на дві або більше підзадач того ж типу, але меншого розміру, і комбінуванні їх рішень для отримання відповіді до вихідного завдання; розбиття виконуються до тих пір, поки всі підзадачі не опиняться елементарними.

Прикладом алгоритму такого виду є сортування злиттям. Цей алгоритм ділить масив навпіл і викликає себе для кожної з половин і так доки розмір половин не стане 1, опісля посортовані половини зливаються і отримуємо результат.

Інші приклади важливих алгоритмів, в яких застосовується парадигма «розділяй і володарюй»: двійковий пошук, метод бісекції, швидке сортування та інше.

3) Довести елементарність функції $f(x1,\ x2) = (x1 + x2)^{-2}$

$(x_1 + x_2)^{-2} = (x_1 + x_2)^{-}*(x_1 + x_2)^{-}$

$$f(x_1, x_2) = (x_1 + x_2)^2 = S^2(*, S^2(+, I_1^2, I_2^2),\ S^2(+, I_1^2, I_2^2))$$

Білет № 15

1. Проблема розпізнавання самозастосованості алгоритмів.

Алгоритм A=<φ,P > називають самозастосовним, якщо слово P_{cod} (cod це степінь) входить в область визначення A, і несамозастосовним в іншому випадку.

Проблема розпізнавання самозастосовності алгоритмів звучить так: Знайти алгоритм, здатний за кодом P_{cod} довільного алгоритму A=<φ,P > визначити, чи є A самозастосовним.

Проблема розпізнавання самозастосовності алгоритму є алгоритмічно нерозв 'язною.

Доведення

Ця теорема доводиться на підставі алгоритмічної системи Тьюрінга. Застосовність МТ до вхідного слова означає зупинку цієї машини через скінченну кількість кроків.

Нехай на стрічці деякої МТ $T_к$ зображено її власний код $T_к$ cod (cod - степінь, треба зверху писати), записаний в алфавіті машини Тк. Якщо Тк застосовна до Tk cod, то будемо називати її самозастосовною, в іншому випадку - несамозастосовною.

Припустимо, що існує МТ М, яка розпізнає самозастосовність довільної машини Тк. Тоді в М кожен самозастосовний код Тк cod перетворюється в деякий символ V, який позначає самозастосовність, а кожен несамозастосовний код - у символ т (тау грецька). Звідси випливає, що можна побудувати і таку машину М1, яка перероблятиме несамозастосовні коди в т , тоді як до самозастосовних кодів машина М1 вже буде незастосовною

Отже, М1 буде застосовною до кожного несамозастосовного коду і незастосовна до самозастосовних кодів Tkcod. Це призводить до суперечностей -

машина М1 самозастосовна, тоді вона застосовна до свого коду М1cod і переробляє його в символ т (тау грецька). Однак поява цього символа саме й повинна означати, що М, несамозастосовна;

нехай М1 несамозастосовна, тоді вона застосовна до М1 cod, а це означає, що М1 є самозастосовною.

Отримана суперечність доводить теорему, оскільки припущення про існування машини М, яка розпізнає самозастосовність, неправильне.

2. Метод гілок і меж. (в інших білетах по іншому написано, мб краще) в **20** заебісь

Метод гілок і меж — один з комбінаторних методів. Застосовується як до повністю, так і частково цілочисельних задач. Його суть полягає в упорядкованому переборі варіантів і розгляді лише тих з них, які виявляються за певними ознаками корисними для знаходження оптимального рішення.

Результатом роботи алгоритму є знаходження максимуму функції на допустимій множині. При чому множина може бути як дискретною, так і раціональною. В ході роботи алгоритму виконується дві операції: розбиття вихідної множини на підмножини(гілки), та знаходження оцінок(меж). Існує оцінка множини згори та оцінка знизу. Оцінка згори — точка що гарантовано не менша за максимум на заданій підмножині. Оцінка знизу — точка що гарантовано не більша за мінімум на заданій підмножині. Множина що має найбільшу оцінку зверху зветься рекордною. На початку вся множина вважається рекордною.

- Рекордна множина розбивається на підмножини;
- Знайти оцінки згори та знизу для нових підмножин;
- Визначити максимальну оцінку знизу серед усіх підмножин;
- Видалити ті множини у яких оцінка зверху менша за максимальну оцінку знизу;
- Знайти максимальну оцінку згори серед усіх підмножин та вважати її рекордною;
- Якщо не досягнуто дискретності, або необхідної точності перейти по пункту 1;

Результатом роботи є значення між оцінкою згори та знизу для рекордної множини. Точністю є різниця між верхньою та нижньою оцінками, тобто для дискретних множин алгоритм завершений тоді, коли ці оцінки збігаються.

Метод використовується для вирішення деяких NP-повних задач. Швидкість алгоритму залежить від вигляду функції та способу визначення оцінок, але гарантовано не більше за повний перебір.

3. На машині Тьюрінга скласти програму інверсії двійкового числа (замінити всі 0 на 1 і всі 1 на 0).

Правила:

Початковий стан q_0, R - вправо, L - вліво, S - на місці, λ- пустий символ, машина починає роботу на λ перед початком вхідного слова

q_0 λ -> q_1 λ R - починаємо роботу

q_1 1 -> q_1 0 R - йдемо до кінця і міняємо всі 1 на 0, а 0 на 1

q_1 0 -> q_1 1 R

q_1 λ -> q_f λ S - доходячи до кінця, завершуєм роботу машини

Приклади:

10101 -> 01010

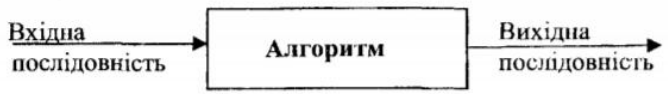
1111 -> 0000

Білет № 16 (Швецов)

1. Важкорозв'язні задачі. Поліноміальні алгоритми та важкорозв'язні задачі.
2. Основні властивості алгоритмів. Способи запису алгоритмів.
3. Довести елементарність функції $f(x_1, x_2) = (x_1 + x_2)^2$.

Питання 1

Алгоритм можна розглядати, як "чорний ящик", який за вхідною послідовністю будує вихідну послідовність:



Зокрема, можна вважати, що вхідна і вихідна послідовності складаються з 0 і 1, які кодують вхід і вихід алгоритму. Тоді алгоритм розглядають як послідовність елементарних двійкових операцій, таких як *або, і, не, друк* і тощо, що працюють з пам'яттю двійкових символів, яка може бути досить великою.

Алгоритм з поліноміальним часом - це алгоритм, час роботи якого обмежений зверху деяким поліномом Рк (n) (к - степінь полінома). Точні оцінки залежать від реалізації алгоритму.

Якщо оцінка алгоритму зростає швидше, ніж поліном, то такі алгоритми називають **експоненціальними**.

Задачі вважають **легкорозв'язними,** якщо вони мають поліноміальні алгоритми розв'язування. Задачу називають **важкорозв'язною**, якщо не існує поліноміальних алгоритмів для її розв'язування.

Питання 2

В алгоритмі A~<*φ,P*> система *P* повинна мати такі властивості:

Дискретність алгоритму. Алгоритм описує процес послідовної побудови величин, який відбувається в дискретному часі.

Ефективність алгоритму. Елементарні кроки повинні виконуватись точно і за короткий відрізок часу.

Скінченність. Алгоритм завжди повинен закінчуватися після скінченної кількості кроків.

Результативність. Алгоритм завжди забезпечує отримання певного результату розв'язування задачі. Якщо деякий елементарний крок не дає результату, то має бути зазначено, що саме має повертати алгоритм.

Якщо система *P* з пари <*φ, P*> задовольняє всі перелічені вище властивості, крім властивості скінченності, то таку пару називають **обчислювальним методом**.

Найбільш поширеними способами запису алгоритму є:

- **словесний** (псевдокоди)
- **графічний** (блок-схеми)
- **запис мовою програмування** (алгоритмічною мовою)

Питання 3

Функція є елементарною, оскільки

$$f(x_1, x_2) = (x_1 + x_2)^2 = S^2(*, S^2(+, I_1^2, I_2^2), S^2(+, I_1^2, I_2^2))$$

Білет 17(Пенькова)

1. Недетерміновані машини Тьюрінга.
2. Динамічне програмування.
3. Довести, що функція $f(x_1, x_2) = x_1 \times x_2$ є примітивно рекурсивною.

1) Недетерміновані машини Тьюрінга

Недетермінованість можна пояснити як алгоритм, який виконує обчислення до певного місця, у якому повинен бути зроблений вибір з кількох альтернатив.

Наприклад, *X* на стрічці і стан 3 допускає як стан 4 із записом на стрічку символу *Y* та зміщенням головки вправо, так і стан 5 з записом на стрічку символу *Z* і зміщенням головки вліво.

НМТ має скінченну кількість можливих кроків, з яких у черговий момент вибирається якийсь один.

Можна уявити, що в разі неоднозначності переходу (поточна комбінація стану і символу на стрічці допускає кілька переходів) НМТ ділиться на кілька НМТ, кожна з яких діє за одним з можливих переходів.

Тобто на відміну від ДМТ, яка має єдиний «шлях обчислень», НМТ має «дерево обчислень».

2) Динамічне програмування

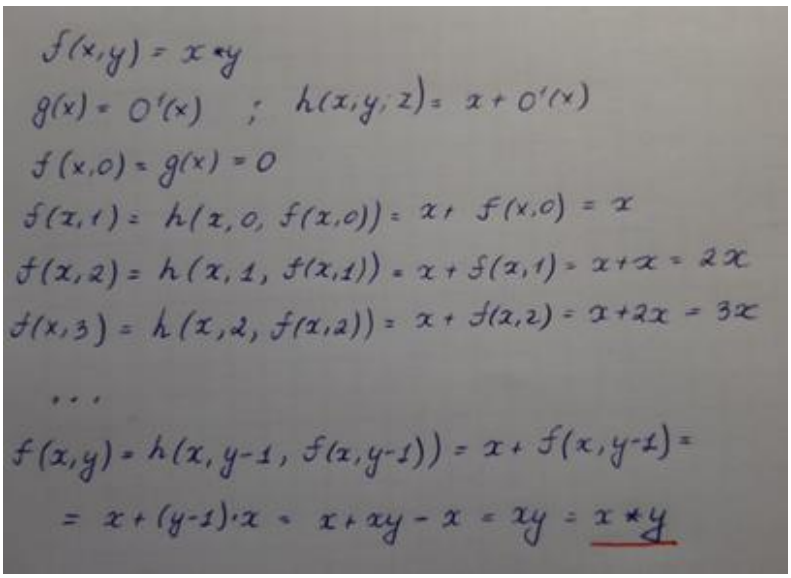
Динамічне програмування - спосіб вирішення складних завдань шляхом розбиття їх на більш прості підзадачі.

Динамічне програмування корисно, якщо на різних шляхах багаторазово зустрічаються одні й ті ж підзадачі; основний технічний прийом - запам'ятовувати рішення підзадач на випадок, якщо та ж підзадача зустрінеться знову.

У типовому випадку динамічне програмування застосовується до завдань оптимізації. У такої задачі може бути багато можливих рішень, але потрібно вибрати оптимальне рішення, при якому значення деякого параметра буде мінімальним або максимальним.

Найпростішим прикладом будуть числа Фібоначчі – щоб обчислити деяке число в цій послідовності, нам треба спершу обчислити третє число, склавши перші два, потім четверте *так само* на основі другого і третього, і так далі

3) Довести, що функція $f(x_1, x_2) = x_1 * x_2$ є примітивно рекурсивною



Білет № 18 (Швецов)

1. Класи *P* та *NP.*

2. Евристичні алгоритми.

3. Показати елементарність $f(x_1, x_2, x_3) = (x_1 + x_2) * x_3$. функції .

Питання 1

Визначимо клас ***P*** як множину всіх мов, які допускають ДМТ(детерміновану машину тюрінга) з поліноміальною часовою складністю, тобто

P={L| існують такі ДМТ *M* і поліном P(n), що часова складність машини *M* дорівнює P(n) і L(*M*)=*L*}.

Клас **NP** - це множина всіх мов, які допускають НМТ(недетерміновану машину тюрінга) з поліноміальною часовою складністю.

Клас *P* можна уявити як клас задач, які можна швидко розв'язати, а *NP* - як клас задач, розв'язок яких можна швидко перевірити.

Мову L_0 з *NP* називають *NP*-повною, якщо за заданим детермінованим алгоритмом розпізнавання L_0 з часовою складністю T(n) >= n і довільною мовою *L* з *NP*, можна ефективно знайти детермінований алгоритм, який розпізнає *L* за час $T(P_L(n))$, де P_L — деякий поліном, що залежить від *L*. Говорять, що *L* поліноміально зводиться до L_0.

Мову *L* називають **поліноміально трансформовною в L_0**, якщо деяка ДМТ *M* з поліноміально обмеженим часом роботи перетворює кожен ланцюжок w в алфавіті мови L в такий ланцюжок w_0 в алфавіті мови L_0, що $w \epsilon$ *L* тоді і лише тоді, коли w_0<=L_0.

Питання 2

Евристичний алгоритм - це алгоритм, спроможний видати прийнятне рішення проблеми серед багатьох рішень, але неспроможний гарантувати, що це рішення буде найкращим.

Наприклад, якщо для розв'язку задачі всі відомі точні алгоритми потребують декількох років машинного часу, то можна прийняти довільний наближений розв'язок, який може бути отримано за розумний час.

Приклад (задача про комівояжера). Нехай задано N міст та матрицю вартостей M, елементи якої m_{ij} дорівнюють вартості подорожі з міста i в місто j. Починаючи з деякого міста *r,* необхідно відвідати всі міста (лише один раз) і повернутися в початкове місто. Отриманий таким способом шлях називають туром. Задача полягає в знаходженні туру мінімальної вартості.

Якщо ми захочемо розв'язати цю задачу для повного графа з 25 вершинами, то потрібно розглянути 24!/2 різних гамільтонових циклів. Якщо припустити, що для розгляду одного такого циклу потрібно одну наносекунду, то загалом буде потрібно приблизно десять мільйонів років для знаходження гамільтонового циклу найменшої довжини. Однак для цієї задачі існує наближений алгоритм (наприклад GTS), який знаходить відповідь з невеликою похибкою від точного результату.

Питання 3

Функція є елементарною, оскільки

$$f(x_1, x_2, x_3) = (x_1 + x_2) * x_3 = S^3(*, S^3(+, I_1^3, I_2^3), I_3^3)$$

Білет № 19

1. Мови і задачі. Приклади NP –повних задач.

   Мови і задачі:

   практично те саме, що і 1 завдання в 18 білеті

   Приклади NP –повних задач:

   1) Задача про кліку.

   2) Задача про виконаиність булевих формул.

   3) Задача про вершинне покриття: чи існує в заданому неорієнтованому графі вершинне покриття потужністю k?

   4) Задача про гамільтонів цикл: чи існує в заданому неорієнтованому графі гамільтонів цикл?

   5) Задача про розфарбовування: чи існує розфарбування неорієнтованого графа в k кольорів?

   6) Задача про множину вершин, які розрізають цикли: чи існує в заданому орієнтованому графі k-елементна множина вершин, які розрізають цикли?

   7) Задача про множину ребер, які розрізають цикли: чи існує в заданому орієнтованому графі k-елементна множина ребер, які розрізають цикли?

   8) Задача про оріентованний гамільтонів цикл: чи існує в заданому орієнтованому графі орієнтований гамільтонів цикл?

9) Задача про покриття множинами: чи існує для заданої сім'ї множин $S_1, S_2, \ldots S_n$ $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ така підсім'я з k множин , що k п їхнє об 'єднання задовольняє такі рівності

$$\bigcup_{j=1}^{k} S_{i_j} = \bigcup_{j=1}^{n} S_j$$

10) Задача про точне покриття: чи існує для заданої сім'ї множин $S_1, S_2, \ldots S_n$ підсім'я множин, які попарно не перетинаються, що є покриттям?

2. Базові поняття алгоритмів та їхні складності. Оцінювання алгоритмів.

Відповідь в білеті №4 питання 1

3. Довести примітивну рекурсивність функції: f(x,y,z,v)=x*y+z*v.

Доведення примітивної рекурсивності додавання:

$$f_+(x, y) = x + y$$

$$g(x) = x;\ h(x, y, z) = z + 1;$$

$f_+(x, 0) = g(x) = x$ - тотожна функція $I_1^1(x)$

$$f_+(x, 1) = h(x, 0, x) = x + 1$$

$$f_+(x, 2) = h(x, 1, f_+(x, 1)) = h(x, 1, x + 1) = x + 2$$

$$f_+(x, y - 1) = h(x, y - 2, f_+(x, y - 2)) = h(x, y - 2, x + y - 2) = x + y - 1$$

$$f_+(x, y) = h(x, y - 1, f_+(x, y - 1)) = h(x, y - 1, x + y - 1) = x + y$$

$$h(x, y, z) = S(I_3^3(x, y, z)) = z + 1$$

$$f_+(x, y) = R^1(I_1^1(x), h(x, y, z))$$

Доведення примітивної рекурсивності множення:

$$f_*(x, y) = x * y$$

$$f_*(x, 0) = O(x)$$
$$f_*(x, 1) = h(x, 0, f_*(x, 0)) = x + O(x) = x$$
$$f_*(x, y - 1) = h(x, y - 2, f_*(x, y - 2)) = x + xy - 2x = xy - x$$
$$f_*(x, y) = h(x, y - 1, f_*(x, y - 1)) = x + xy - x = xy$$
$$h(x, y, z) = f_+(I_3^1(x, y, z), I_3^3(x, y, z))$$
$$f_*(x, y) = R^1(O(x), h(x, y, z))$$

Можемо зробити висновок, що задана функція також примітивно рекурсивна, тому що примітивно рекурсивними є функції додавання і множення

Білет № 20

1. Застосування теорії *NP*-повноти для аналізу задач.

Однією з найскладніших проблем теорії обчислень є так звана проблема “P = NP”, тобто чи тотожні класи P та NP

Найсерйознішою причиною вважати, що ці класи не тотожні це існування NP - повних задач. У класі NP є NP-повні (універсальні) задачі, тобто такі, що до них поліноміально зводиться довільна задача з класу NP. їх використовують як еталони складності.

Задача є NP-трудною, якщо будь-яка задача з класу до неї зводиться, а NP-повною є задача, яка є NP-трудною, але належить до класу NP. Важливою властивістю NP-повних задач є те, що всі вони “еквівалентні” в такому сенсі: якщо хоча б для однієї з них буде доведено, що вона є легкорозв’язною, то такими будуть і решта задач з цього класу. Якщо в класі NP існує задача, нерозв’язна за поліноміальний час, то всі NP-повні задачі такі ж - то N = NP.

Отже, гіпотеза N = NP означає, що NP -повні задачі не можуть бути розв’язані за поліноміальний час. Доведення NP -повноти деякої задачі є суттєвим аргументом на користь її практичної нерозв’язності. Для такої задачі доцільніше будувати достатньо точні наближені алгоритми, ніж затрачати час на пошук швидких алгоритмів, що розв’язують її точно.

“Трудність” задач можна порівнювати, зводячи одну задачу до іншої. Метод зведення є головним у доведенні NP-повноти багатьох задач.

Сьогодні визначено NP -повноту багатьох задач, еквівалентних між собою щодо поліноміальної звідності. NP-повні задачі й задачі P сильно відрізняються за трудомісткістю розв’язування, проте в строгому сенсі ця різниця і, отже, різниця між класами P та NP не доведена.

Знайти точний розв’язок задачі з класу NP важко, оскільки кількість можливих комбінацій вхідних значень, що потребують перевірки, надзвичайно велика. Для кожного набору вхідних значень I можна створити множину можливих розв’язків PSi. Тоді оптимальним уважають такий розв’язок $S_{optimal}$ є PSi, що:

$$\text{Value}(S_{optimal}) < \text{Value}(S') \text{ для всіх } S' \in PS_I,$$

якщо ми маємо справу з задачою мінімізації, або

$$\text{Value}(S_{optimal}) > \text{Value}(S') \text{ для всіх } S' \in PS_I,$$

якщо розв’язуємо задачу максимізації.

2. Метод гілок і меж.

Метод гілок і меж орієнтований на розв’язування задач оптимізації. Він досліджує деревоподібну модель простору розв’язків задачі та дає змогу серед елементів множини можливих розв’язків знайти найліпший (найоптимальніший). Головна властивість, яку повинна мати ця множина, - це можливість розділяти її на підмножини, що не перетинаються, для яких можна виконати деяку оцінку “оптимальності” можливого розв’язку. “Оптимальність” такого розв’язку означає, що не існує кращого розв’язку в межах вибраної підмножини . У разі практичного застосування цього методу всі можливі варіанти розв’язків задачі розбивають на класи, виконують оцінку знизу для всіх розв’язків з одного класу, і якщо вона більша від раніше отриманої, то відкидають усі варіанти з цього класу.

Два головні кроки в цьому алгоритмі - це галуження та обчислення меж.

На прикладі задачі про комівояжера

Корінь пошукового дерева відповідатиме множині всіх можливих турів. У загальному випадку корінь буде відображати повну множину всіх (N-1)! можливих турів. Гілки, що

виходять з кореня, визначені вибором одного ребра, наприклад, (i,j). Завданням є розділити множину всіх турів на дві підмножини: одна, яка, досить імовірно, містить оптимальний тур, та інша, яка, імовірно, не містить його. Для цього вибираємо ребро (i, j) і розділяємо множину всіх турів на дві підмножини: {i, j} - множина турів, які містять ребро (i, j), і $\overline{\{i, j\}}$ множина турів, які не містять ребра (i, j).

Метод використовується для вирішення деяких NP-повних задач. Швидкість алгоритму залежить від вигляду функції та способу визначення оцінок, але гарантовано не більше за повний перебір.

3. Скласти нормальний алгоритм, що виконує збільшення десяткового числа на 2.

0 -> 0 **пересуваєм * до кінця числа**
1 -> 1
2 -> 2
3 -> 3
4 -> 4
5 -> 5
6 -> 6
7 -> 7
8 -> 8
9 -> 9
0* ->. 2 **збільшуєм на 2 число**
1* ->. 3
2* ->. 4
3* ->. 5
4* ->. 6
5* ->. 7
6* ->. 8
7* ->. 9
8* -> #0 **якщо остання цифра 8 або 9 то створюєм # - наступний порядок**
9* -> #1 **збільшується на 1**
0# ->. 1
1# ->. 2
2# ->. 3
3# ->. 4
4# ->. 5
5# ->. 6
6# ->. 7
7# ->. 8
8# ->. 9
9# -> #0
 # ->. 1
0 -> 0* **спочатку виконується одне з цих правил - створюєм * з допомогою якої йдем**
1 -> 1* **до кінця числа**
2 -> 2*
3 -> 3*
4 -> 4*
5 -> 5*
6 -> 6*
7 -> 7*
8 -> 8*
9 -> 9*

123 -> 1*23 -> 12*3 -> 123* -> 125

999 -> 9*99 -> 99*9 -> 999* -> 99#1 -> 9#01 -> #001 -> 1001

Білет № 21 (Сипко)

1. Система нормальних алгоритмів Маркова. Принцип нормалізації.

Система нормальних алгоритмів Маркова

Алгоритми Маркова — це формальна математична система. В алгоритмічній системі Маркова існує лише оператор підстановки та розпізнавач входження.

Загальна схема: застосувавши декілька разів оператор підстановки до вхідного рядка R, перетворити його у вихідний рядок Q.

Простою продукцією називають запис вигляду $u \rightarrow w$. u - ангицедент, w - консеквент.

Уважають, що формула $u \rightarrow w$ може бути застосована до рядка $Z \in V$, якщо є хоча б одне входження и в Z.

Заключною продукцією називають запис вигляду $u \rightarrow w$, де u,w - рядки в V .

Нормальним алгоритмом, чи алгоритмом Маркова, називають упорядковану множину продукцій $P_1, P_2, \ldots, P_n$.

Кожна з продукцій містить розпізнавання входження підрядка u в рядок Z та підстановку w замість u у разі успішного розпізнавання. Послідовність виконання продукцій залежить від того, чи може бути застосована до рядка чергова формула підстановки.

Алгоритм Маркова завершується в одному з двох випадків:

- до рядка не може бути застосована жодна з наявних формул підстановок;
- до рядка застосована заключна підстановка.

Тоді алгоритм вважають **застосовним** до заданого вхідного слова.

Нормальні алгоритми іноді зображають за допомогою граф-схеми. Алгоритми, які задають граф-схемами, складеними винятково з розпізнавачів входження і операторів підстановки, називають **нормальними**, якщо їхні граф-схеми задовольняють такі умови:

1. Кожний розпізнавальний вузол Q і відповідний йому операторний вузол Q→ R об 'єднані в один узагальнений вузол, який називають **операторно-розпізнавальним**; усі узагальнені вузли схеми впорядковані за допомогою нумерації від 1 до n;
2. негативний вихід i-го вузла приєднаний до (i +1)-го вузла (i =$\overline{1, n-1}$), а негативний вихід n -го вузла — до вихідного вузла граф-схеми;
3. позитивні виходи всіх узагальнених вузлів приєднані або до першого, або до вихідного вузла граф-схеми;
4. вхідний вузол приєднаний до першого узагальненого вузла.

Нормальні алгоритми прийнято задавати не граф-схемами, а впорядкованими наборами підстановок, у яких кожна підстановка відповідає операторно-розпізнавальному вузлу. Q →R - звичайна підстановка, а заключна - Q→ .R

Упорядкований набір підстановок визначеного типу називають **схемою** заданого алгоритму.

Виконання алгоритму починається з першої формули. Послідовність виконання алгоритму залежить від того, чи може бути застосована до рядка чергова формула підстановки. Процес виконання підстановок закінчується лише тоді, коли жодна з підстановок схеми не може бути застосована до отриманого слова, або коли виконана деяка заключна підстановка.

Принцип нормалізації.

Будь-яка алгоритмічна система має задовольняти 2 вимоги: бути математично строгою та універсальною. За допомогою певного математичного апарату досягають математичної строгості. Універсальність теорії нормальних алгоритмів формулюють у вигляді принципу нормалізації.

Теорема. Для того, щоб реалізувати в схемах нормальних алгоритмів довільний алгоритм $A = \{\varphi, P\}$, необхідно, щоб у системі нормальних алгоритмів були як звичайні, так і заключні підстановки.

Принцип нормалізації. Для будь-якого алгоритму $A = \{\varphi, P\}$ в довільному алфавіті X можна побудувати еквівалентний йому нормальний алгоритм над алфавітом X.

Перехід від інших способів опису алгоритмів до еквівалентних нормальних алгоритмів називають **зображенням у нормальній формі, або нормалізацією**.

Алгоритм A = (φ,P) в алфавіті X називають **нормалізованим**, якщо можна побудувати еквівалентний йому нормальний алгоритм над алфавітом X. В іншому випадку алгоритм називають **ненормалізованим**.

Усі алгоритми є нормалізовані. Принцип нормалізації не може бути математично доведений або заперечений.

Головні факти, що підтверджують принцип нормалізації:

1. Правильність алгоритму ґрунтується на тому, що всі відомі алгоритми - нормалізовані.
2. Якщо вихідні алгоритми вже нормалізовані, то нормалізованою буде і композиція цих алгоритмів.
3. Еквівалентність системи нормальних алгоритмів усім іншим алгоритмічним системам.
4. Усі спеціальні спроби побудови алгоритмів найбільш загального вигляду не вивели за межі класу нормалізованих алгоритмів.

Систему нормальних алгоритмів прийнято вважати практично універсальною алгоритмічною системою.

2. Приклади NP -повних задач.

Нехай $G = (V, E)$ - орієнтований граф.

Множиною вершин, які розрізають цикли, називають таку підмножину і $S \subseteq V$, що кожен цикл у G містить вершину з S.

Множиною ребер, які розрізають цикли, називають таку підмножину $F \subseteq E$, що кожен цикл у G містить ребро з F.

Орієнтованим гамільтоновим циклом називають цикл у графі G, який містить усі вершини з V.

Приклади задач, які належать до класу **NP**:

1) Задача про кліку
2) Задача про виконанність булевих формул.
3) Задача про вершинне покриття: чи існує в заданому неорієнтованому графі вершинне покриття потужністю k?
4) Задача про гамільтонів цикл: чи існує в заданому неорієнтованому графі гамільтонів цикл?
5) Задача про розфарбовування: чи існує розфарбування неорієнтованого графа в k кольорів?
6) Задача про множину вершин, які розрізають цикли: чи існує в заданому орієнтованому графі k-елементна множина вершин, які розрізають цикли?
7) Задача про множину ребер, які розрізають цикли: чи існує в заданому орієнтованому графі k-елементна множина ребер, які розрізають цикли?
8) Задача про оріентованний гамільтонів цикл: чи існує в заданому орієнтованому графі орієнтований гамільтонів цикл?
9) Задача про покриття множинами: чи існує для заданої сім'ї множин $S_1, S_2, \dots S_n$ така підсім'я з k множин $S_{i_1} S_{i_2}, \dots S_{i_k}$, що їхнє об 'єднання задовольняє такі рівності $\bigcup_{j=1}^{k} S_{i_j} = \bigcup_{j=1}^{n} S_j$?
10) Задача про точне покриття: чи існує для заданої сім "і множин $S_1, S_2, \dots S_n$ підсім'я множин, які попарно не перетинаються, що є покриттям?

3. Скласти нормальний алгоритм, який обчислює частку і залишок від ділення на 7 в унарній системі числення.

$$a1111111 \to 1a$$
$$a \to .\lambda$$
$$1 \to a1$$

Білет № 22(Мірai)

1. Способи запису алгоритмів. Різновиди алгоритмів. Композиція алгоритмів.
2. Недетерміновані машини Тьюрінга.
3. Заданий алфавіт A=(c,d,e,f). Символи у вхідному слові відсортувати по алфавіту.

Питання 1

Способи запису алгоритмів:

- Словесний - опис на природній мові
- Графічний - зображення у вигляді так званої блок схеми
- Мовою програмування(алгоритмічною мовою) - за допомогою алфавіту, синтаксису, семантики.

Різновиди алгоритмів:

В залежності від наявності чи відсутності у системі правил P тієї чи іншої властивості множина всіх алгоритмів розбивається на відповідні класи.

Алгоритм – детермінований, якщо система величин σ_{i+1} однозначно визначається системою величин σ_i.

Алгоритм є самозмінний, якщо в процесі роботи система P змінюється.

Розглянемо систему правил P алгоритму, закодовану певним чином у вхідному алфавіті алгоритму A. Позначимо цей код P^{cod}. Алгоритм - самозастосовний, якщо слово P^{cod} є в області визначення A.

Алгоритм – універсальний, якщо він еквівалентний довільному наперед заданому алгоритму.

Композиція алгоритмів

- Суперпозиція. При суперпозиції двох алгоритмів вихідне слово одного з них розглядають як вхідне слово іншого
- Об'єднання. Об'єднанням алгоритмів A, B є алгоритм C в цьому ж алфавіті, який перетворює довільне вхідне слово $P \in D(A) \cap D(B)$ в конкатенацію слів A(P) і B(P).
- Розгалуження. Розгалуженням – це композиція трьох алгоритмів A, B, C задана співвідношенням:

$$F(P) = \begin{cases} A(P), & C(P) = R \\ B(P), & C(P) \neq R \end{cases}$$

  R – деяке фіксоване слово

- Ітерація. Ітерацією алгоритмів A, B є алгоритм C такий, що для довільного вхідного слова P вхідне слово C(P) отримують в результаті багаторазового застосування алгоритму A до моменту, поки не буде отримано слово, перетворюване алгоритмом B в якесь фіксоване слово R.

Питання 2

Недетермінована машина Тьюрінга має скінченну кількість кроків. Кожного разу вибирається якийсь один. Ланцюжок *x* допускається, якщо хоча б одна послідовність кроків для входу *x* приводить до допустимого заключного стану. Коли ланцюжок *x* заданий, то недетермінована машина Тьюрінга паралельно виконує всі можливі послідовності кроків до моменту, коли досягне заключного стану або коли подальші кроки будуть неможливі.

Формальне визначення недетермінованої машини Тьюрінга:

k-стрічковою недетермінованою машиною Тьюрінга називають сімку $(S, Q, q_0, q_F, I, \Lambda, \delta)$. Значення всіх компонент такі самі як у детермінованої машини Тьюрінга, але в даному випадку функція переходів δ є відображенням множини $\{Q \backslash q_F\} \times S^k$ у множину підмножин у $Q \times (S \times \{L, R, H\})^k$.

Множину всіх ланцюжків, які допускає машина називають мовою. Машина має часову складність T(n), якщо кожен допустимий ланцюжок довжини n матиме послідовність, яка складатиметься не більше ніж з T(n) кроків і приводитиме до допустимого стану. Машина має ємнісну складність S(n), якщо кожен допустимий ланцюжок довжини n матиме послідовність кроків, яка приводитиме до допустимого стану і в якій кількість клітинок, переглянутих головкою на кожній стрічці не перевищуватиме S(n).

Питання 3

V = {c, d, e, f}

$P_1: 'dc' \to 'cd'$

$P_2: 'ec' \to 'ce'$

$P_3: 'fc' \to 'cf'$

$P_4: 'ed' \to 'de'$

$P_5: 'fd' \to 'df'$

$P_6: 'fe' \to 'ef'$

Приклад: $ecfd \to cefd \to cedf \to cdef$ (сортування ecfd)

Білет № 23(Ланчевич)

1. Абстрактні алфавіти й алфавітні оператори. Кодувальні алфавітні оператори. Способи задання алфавітних операторів.

Абстрактним алфавітом називають довільну скінченну сукупність елементів.

Слово у довільному алфавіті - це будь-яка скінченна впорядкована послідовність елементів цього алфавіту, які називаються **буквами**. **Довжина слова** - кількість букв у слові.

На множині слів визначено дві операції: конкатенація (приєднання) та підстановка (заміна).

Конкатенація двох слів A та B - це слово AB, отримане приписуванням слова B до слова A.

Слово A називають **підсловом** слова B, якщо B можна записати в такому вигляді:

$$B = CAD, \qquad (1.1)$$

де C та D — деякі слова, можливо, порожні. Слово A є входженням у слово B.

Якщо в розкладі (1.1) підслово C має мінімальну довжину, то таке входження слова A в слово B називають **першим входженням**.

Нехай задано слова A, B, F і A є i-м входженням у слово B:

$$B=CAD.$$

Тоді слово G отримують операцією заміни (підстановки) i-го входження слова A словом F, якщо

$$G=CFD.$$

Якщо в операції заміни i=1, то підстановку називають **стандартною** (або канонічною).

Алфавітним оператором, або **алфавітним відображенням**, **φ** називають відповідність між словами в одному або різних алфавітах.

Функцію називають **словниковою**, якщо вона перетворює слово одного алфавіту в слово іншого алфавіту.

Нехай P - вхідне слово. Якщо слову P оператор φ не ставить у відповідність жодного вихідного слова, то кажуть, що на слові P оператор φ **не визначений**. Сукупність усіх слів, на яких оператор φ визначений, називають **областю визначення** (областю застосування) оператора φ.

Сукупність усіх слів, на яких оператор φ невизначений, називають **областю заборони φ.**

Нехай A - деякий алфавіт, який називають **стандартним**, B - довільний алфавіт. Нехай $\{L\}_A$, $\{M\}_B$ - множини слів у відповідних алфавітах. Кажуть, що множина $\{M\}_B$ **закодована** в алфавіті A, якщо задано такий однозначний оператор:

$$\varphi : \{M\}_B \rightarrow \{L\}_A.$$

Оператор φ називають **кодувальним**, а слова з $\{L\}_A$ - **кодами об'єктів** з $\{M\}_B$.

Теорема. Кодувальний оператор є взаємно однозначним тоді й тільки тоді, коли:

1) коди різних букв алфавіту B різні;

2) код довільної букви алфавіту B не може бути першим входженням у коди інших букв цього алфавіту.

Кодування об’єктів алфавіту В словами однакової довжини називають **нормальним кодуванням**.

Розглянемо декілька кодувань, які трапляються найчастіше.

Кодування слів у багатобуквеному алфавіті. Нехай В - довільний алфавіт з n букв, А - стандартний алфавіт з m букв (m> 1). Для довільної пари (n, m) завжди можна задати таке число *l*, що

$$m^l >= n.$$

Кодування безконечних алфавітів. Згідно з визначенням алфавіту, він повинен бути скінченною множиною. Проте на практиці зручно розглядати безконечні алфавіти, які складаються зі скінченної кількості букв з індексами, які набувають натуральних значень 0, 1, 2,... Такий алфавіт формально безконечний.

Розрізняють два способи задання операторів.

1. Табличний спосіб задання операторів. Такий спосіб застосовують тоді, коли область визначення оператора скінченна. Оператор φ задають таблицею відповідності, у якій для кожного вхідного слова P з області визначення φ виписано відповідне вихідне слово Q:

$$P_1 \to Q_1,$$
$$\cdots$$
$$P_n \to Q_n.$$

2. Задання оператора скінченною системою правил, яка дає змогу за скінченну кількість кроків знайти значення φ на довільному вхідному слові, наприклад, система правил для додавання натуральних чисел у системі числення з основою р або система правил знаходження найбільшого спільного дільника двох додатних чисел та ін.

2. Рекурсивні функції.

Рекурсивні функції - це алгоритмічні системи, що ґрунтуються на використанні конструктивно визначених арифметичних (цілочислових) функцій.

Числові функції, значення яких можна обчислити за допомогою деякого (єдиного для заданої функції) алгоритму, називають **обчислюваними функціями**.

Найпростіші функції - наступності, тотожно рівно 0, тотожню(вибору аргументу).

Числову функцію φ : N -> N називають **функцією наступності**, якщо φ(x) = x + 1.

Числову функцію $\varphi : N^{(n)} \longrightarrow N$ називають **нуль-функцією**, якщо $\varphi(x_1,...,x_n) = 0$.

Числову функцію $\varphi_i : N^{(n)} \longrightarrow N$ називають **функцією вибору аргументів**, якщо вона повторює значення свого і-го аргумента:

$\varphi(x_1,...,x_n) = x_i, (1 < i < n)$.

Існують також примітивно- рекурсивні, загально рекурсивні та частково-рекуурсивні ф-ї.

Функцію f називають ***примітивно-рекурсивною****, якщо її можна отримати із застосуванням скінченної кількості операторів суперпозиції і примітивної рекурсії на підставі лише найпростіших функцій S^1, O^1, I_m^n:*

$$\{S^1, O^1, I_m^n\} \xrightarrow{S^{n+1},\ R} f .$$

Часткову функцію f називають ***частково-рекурсивною відносно*** *σ, якщо її можна отримати з функцій системи σ і найпростіших функцій із застосуванням скінченної кількості операторів суперпозиції, примітивної рекурсії та мінімізації:*

$$\{S^1, O^1, I_m^n, \sigma\} \xrightarrow{S^{n+1},\ R,\ M} f .$$

Часткову функцію f називають ***частково-рекурсивною****, якщо її можна отримати з найпростіших функцій із застосуванням скінченної кількості операторів суперпозиції, примітивної рекурсії та мінімізації:*

$$\{S^1, O^1, I_m^n\} \xrightarrow{S^{n+1},\ R,\ M} f .$$

Функції, які можна отримати з найпростіших функцій S^1, O^1, I_m^n за допомогою оператора примітивної рекурсії, суперпозиції та слабкої мінімізації, називають ***загальнорекурсивними****.*

---

3. На машині Тьюрінга скласти програму інверсії двійкового числа (замінити всі 0 на 1 і всі 1 на 0). - Позичив в Ундрея - білет №15

Початковий стан q0, R - вправо, L - вліво, S - на місці, λ - пустий символ

q0 λ -> q1 λ R - починаємо роботу

q1 1 -> q1 0 R - йдемо до кінця і міняємо всі 1 на 0, а 0 на 1

q1 0 -> q1 1 R

q1 λ -> q_f λ S - доходячи до кінця, завершуєм роботу машини

Приклади:

1101101 -> 0010010

10101 -> 01010

11110 -> 00001

Білет № 24(Ружко)

1. Обчислювані функції. Найпростіші функції.

2. Метод “Поділяй і володарюй”.

3. Довести, що функція f(x1,x2) = x1*x2 (двомісного множення) є примітивно рекурсивною.

Питання 1

Обчислювальними функціями називають числові функції, значення яких можна обчислити за допомогою деякого(єдиного) алгоритму.

Нехай N - множина натуральних чисел, тоді існують такі найпростіші функції:

- Функція наступності($S^1(x)$) : $\varphi: N \to N$, якщо $\varphi(x) = x + 1$
- Нуль-функція($O^n(x_1, \dots, x_n)$): $\varphi: N^{(n)} \to N$, якщо $\varphi(x_1, \dots, x_n) = 0$
- Тотожна функція($I_i^n(x_1, \dots, x_n)$): $\varphi_i: N^{(n)} \to N$, якщо $\varphi_i(x_1, \dots, x_n) = x_i$, $(1 \le i \le n)$

Найпростіші функції є всюди визначені.

Приклад найпростіших функцій:

$S^1(5) = 6$,

$O^4(3,6,2,1) = 0$,

$I_2^3(4,3,8) = 3$.

Питання 2

“Поділяй і володарюй” - це метод розробки алгоритмів. Зазвичай алгоритми цього виду є рекурсивними. Для розв’язування задачі вони рекурсивно викликають самі себе один або декілька разів, розбиваючи задачу на менші, допоміжні задачі, які стосуються основної. Після вирішення допоміжних задач їхній розв’язок комбінується для вирішення основної задачі. Варто зазначити, що допоміжні задачі зазвичай мають менший обсяг, але один і той самий елемент може потрапити в декілька наборів.

Прикладом алгоритму такого виду є сортування злиттям. Цей алгоритм ділить масив навпіл і викликає себе для кожної з половин і так доки розмір половин не стане 1, опісля посортовані половини зливаються і отримуємо результат.

Питання 3

$$f_*(x,y) = x * y$$
$$f_*(x,0) = O(x)$$
$$f_*(x,1) = h(x,0,f_*(x,0)) = x + O(x) = x$$
$$f_*(x,y-1) = h(x,y-2,f_*(x,y-2)) = x + xy - 2x = xy - x$$
$$f_*(x,y) = h(x,y-1,f_*(x,y-1)) = x + xy - x = xy$$
$$h(x,y,z) = f_+(I_3^1(x,y,z), I_3^3(x,y,z))$$
$$f_*(x,y) = R^1(O(x), h(x,y,z))$$

Де f_+ функція двомісного додавання $f_+(x,y) = x + y$

Про всякий доведення додавання тоже:

$$f_+(x,y) = x + y$$
$$g(x) = x;\ h(x,y,z) = z + 1;$$
$$f_+(x,0) = g(x) = x \text{ - тотожна функція } I_1^1(x)$$
$$f_+(x,1) = h(x,0,x) = x + 1$$
$$f_+(x,2) = h(x,1,f_+(x,1)) = h(x,1,x+1) = x + 2$$
$$f_+(x,y-1) = h(x,y-2,f_+(x,y-2)) = h(x,y-2,x+y-2) = x + y - 1$$
$$f_+(x,y) = h(x,y-1,f_+(x,y-1)) = h(x,y-1,x+y-1) = x + y$$
$$h(x,y,z) = S(I_3^3(x,y,z)) = z + 1$$
$$f_+(x,y) = R^1(I_1^1(x), h(x,y,z))$$

Білет 25(Ружко)

1. Головні оператори. Оператор примітивної рекурсії. Оператор мінімізації.
2. Метод гілок і меж.
3. Написати програму для машини Тьюрінга, що додає два числа в унарній системі числення.

Питання 1

Операції над функціями називаються операторами. Під час побудови частково-рекурсивних функцій використовують три оператори:

- Суперпозиції. Нехай:

  $$g^m(x_1,...,x_m) = f^n(f_1^m(x_1,...,x_m),...,f_n^m(x_1,...,x_m))$$

  f^n, f_n^m, g^m – це деякі функції. Тоді оператор, який дасть нам g^m називають операторм суперпозиції і позначають $\mathbb{S}^{n+1}(f^n, f_1^m, ..., f_n^m)$, де n+1 – кількість функцій.

- Примітивної рекурсії. Розглянемо функцію $f^{n+1}: N^{(n+1)} \to N$, яка визначена наступним чином:

  $$\begin{cases} f(x_1,...,x_n,0) = g(x_1,...,x_n), \\ f(x_1,...,x_n,y+1) = h(x_1,...,x_n,y,f(x_1,...,x_n,y)) \end{cases}$$

  Отже, оператор примітивної рекурсії($\mathbb{R}$) дасть можливість з g, h побудувати функцію f.

- Якщо з функції $f(x_1, \dots, x_n)$ можна утворити функцію $\mu_y(f(x_1, \dots, x_{n-1}, y) = x_n)$, то оператор, який дає таку можливість називають оператором мінімізації і позначають $\mathbb{M}$:

  $$\mu_y(f(x_1, \dots, x_{n-1}, y) = x_n) = \mathbb{M}f$$

Питання 2

Метод гілок і меж орієнтований на розв'язування задач оптимізації. Цей метод досліджує деревоподібну модель множини розв'язків задачі і дає змогу серед елементів цієї множини вибрати найоптимальніший. Основна властивість, яку має мати ця множина розв'язків - можливість розділити її на множини, в перетині яких буде пуста множина, тобто вони не будуть мати спільних елементів.
Оптимальність розв'язку означає, що кращого розв'язку в межах вибраної підмножини не існує, хоча й такої оптимальності на підмножині може і зовсім не бути.
В ході роботи алгоритму, який працює за даним методом виконується дві операці: розбиття вихідної множини на підмножини(гілки) та знаходження оцінок(меж).
Оцінки бувають дві - зверху і знизу. Оцінка зверху - точка, яка гарантовано не менша за максимум на заданій підмножині. Оцінка знизу - точка, яка гарантовано не більша за мінімум на заданій підмножині. Множина, яка має найбільшу оцінку називається рекордною.
Алгоритм:

- Рекордна множина розбивається на підмножини;
- Знайти оцінки згори та знизу для нових підмножин;
- Визначити максимальну оцінку знизу серед усіх підмножин;
- Видалити ті множини у яких оцінка зверху менша за максимальну оцінку знизу;
- Знайти максимальну оцінку згори серед усіх підмножин та вважати її рекордною;
- Якщо не досягнуто дискретності, або необхідної точності перейти по пункту 1;

Результатом є значення між оцінкою зверху та оцінкою знизу для рекордної множини.
Метод застосовується для розв'язку деяких NP-повних задач. Швидкість алгоритму залежить від вигляду функції та способу визначення оцінок, але гарантовано не більше за повний перебір.

Питання 3

q0 1 -> q0 1 R
q0 + -> q1 _ R
q1 1 -> q2 + L
q2 _ -> q0 1 R
q1 _ -> qf _ N

Приклад:

1111+111
1111 _ 111
1111 _ +11
11111+11
11111_11
11111_+1
111111+1
111111_1
111111_+
1111111+
1111111_
1111111