МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

ДИСКРЕТНА МАТЕМАТИКА

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних занять з курсу “Дискретна математика” для студентів спеціальностей 6.040301 – “Прикладна математика”, 6.040302 – “Інформатика”

Затверджено
на засіданні кафедри
прикладної математики
Протокол № від 2014р.

Львів – 2014

Дискретна математика. Методичні вказівки до лабораторних занять з курсу “ Дискретна математика ” для студентів спеціальностей 6.040301 – “Прикладна математика”, 6.040302 – “Інформатика”/ Укл.: Л.І.Демків, О.С.Манзій, І.І.Кавалець— Львів: Видавництво Національного університету “Львівська політехніка”, 2014. – 32 с.

Укладачі Демків Л.І., канд. фіз.-мат. наук, доц.
Манзій О.С., канд. фіз.-мат. наук, доц.,
Кавалець І.І., канд

Відповідальний за випуск Уханська О.М., канд.ф.-м.н., доц.

Рецензенти Гнатів Б.В., канд. фіз.-мат. наук, доц.,
Пукач П.Я., канд. фіз.-мат. наук, доц.

Лабораторна робота № 1.

Розділ: Теорія множин.

Тема роботи: *Теоретико-множинні операції. Відношення елементів множини.*

Необхідні теоретичні відомості:

Поняття множини належить до фундаментальних понять математики (як і поняття точки, прямої, площини) і його неможливо задати строго. Тому, вживаючи термін множина, ми під ним будемо розуміти сукупність елементів, наділених спільними властивостями.

Множину прийнято позначати великими літерами (латинськими, грецькими, готичними), наприклад: множина Ω, множина Р або $\Re$, множина A. Множину, що не містить жодного елемента, будемо називати порожньою множиною (позначатимемо $\varnothing$), тобто порожня множина містить нуль елементів. Елементи множини прийнято позначати малими літерами (латинськими, грецькими).

Запис $x \in A$ – означає, що елемент x належить множині A;

$x \notin A$ – означає, що елемент x не належить множині A.

Якщо кожний елемент множини A є разом з тим елементом множини B то множина A називається підмножиною множини B. Цей факт позначається наступним чином: $A \subset B$; якщо ж множина B є підмножиною множини A, цей факт позначають: $B \supset A$. Поряд із символами включення $\subset$ та $\supset$ необхідно розрізняти символи $\subseteq$ і $\supseteq$ $(A \subseteq B)$, які означають, що A включено в B, і тут не виключений варіант рівності множин. Згідно з цим означенням, будь-яка множина A є підмножиною самої себе, тобто $A \subseteq A$.

Порожня множина $\varnothing$ завжди є підмножиною будь-якої множини.

Множину, яка є об'єднанням усіх множин, називають універсальною та позначають літерами U або X.

Операції над множинами та їх властивості.

Об'єднання множин: $A \cup B = \{x | x \in A \text{ або } x \in B\}$.

Різниця множин: $A \setminus B = \{x | x \in A, x \notin B\}$.

Перетин множин: $A \cap B = \{x | x \in A \, i \, x \in B\}$.

Симетрична різниця множин: $(A \Delta B) = (A \cup B) \setminus (A \cap B)$.

Доповнення множин: $\overline{A} = \{x | x \notin A\}$.

Розглянемо множини $A = \{1, 2, 7, 9\}$ та $B = \{1, 3, 6, 7, 9\}$, тоді $A \cup B = \{1, 2, 3, 6, 7, 9\}$; $A \setminus B = \{2\}$; $A \cap B = \{1, 7, 9\}$; $A \Delta B = \{2, 3, 6\}$.

Множина $\overline{A}$ є доповненням множини A до універсальної множини X, то $\overline{A} = X \setminus A$.

Якщо за універсальну множину прийняти множину натуральних чисел, то для розглянутої вище множини A, отримаємо

$\overline{A} = N \setminus A = \{3, 4, 5, 6, 8, 10, 11, 12, \ldots\}$.

Прямим (декартовим) добутком множин A і B ($A \times B$) називається множина всіх можливих впорядкованих пар (кортежів), першим елементом яких є елемент множини A, а другим – елемент множини B $A \times B \equiv \{(a,b) | a \in A, b \in B\}$.

Впорядкованість означає, що елемент декартового добутку (*a*,*b*) не дорівнює елементу (*b*,*a*). В частковому випадку, якщо A≡B, то такий добуток будемо позначати A^2.

Множину можна зберігати в пам'яті як масив. Типові операції над множинами – об'єднання, перетин, перевірка приналежності елемента заданій множині – реалізуються за допомогою засобів алгоритмічної мови або процедур обробки масивів даних.

Можливий ще один метод зображення множин, при якому для кожного потенціально допустимого елемента вказується, належить він даній множині, чи ні. Якщо множину всіх потенціально допустимих елементів пронумерувати індексами від 1 до *n*, то кожній множині поставимо у відповідність вектор логічних змінних довжиною *n*, тобто вектор змінних, що приймають тільки одне з двох значень: істина, хибне. Необхідно зауважити, що зображення множин у вигляді вектора визначає на ній порядок слідування, а різний порядок у послідовності переліку елементів множини приводить до різних відношень впорядкованості на множині. Частіше всього використовується природно виникаючий порядок переліку елементів. Для прикладу, порядок може задаватися часом включення елементів у множину.

Відношення. Способи задання відношення. Властивості відношень.

Нехай *A* – задана множина. Будемо стверджувати, що на множині *A* задане відношення ρ, якщо воно задане для деякої підмножини $\{(a,b) | a \in A, b \in A\} \subseteq A^2$ і позначатимемо його $< a \rho\ b >$.

Відношення ρ називається:

- рефлексивним – якщо існує $< a \rho\ a >$ для всіх $a \in A$;
- симетричним – якщо з $< a \rho\ b > \Rightarrow < b \rho\ a >$;
- транзитивним – якщо з $< a \rho\ b >$ і $< b \rho\ c > \Rightarrow < a \rho\ c >$.

Відношення, яке володіє всіма трьома властивостями називається відношенням еквівалентності.

Відношення на множині *A* достатньо зручно задавати і аналізувати з допомогою матриці відношень *M*,

$$\text{де } m_{i,j} = \begin{cases} 1, & < a_i \rho a_j > \\ 0, & \text{в протилежному випадку.} \end{cases}$$

Поряд з поняттями рефлексивності, симетричності та транзитивності можна ввести поняття антирефлексивності, антисиметричності та антитранзитивності.

Відношення ρ називається

- антирефлексивним – якщо для жодного з $a \in A$ не виконується $< a \rho\ a >$.

- антисиметричним – якщо для будь-яких *a* та *b* з множини *A* буде виконуватися $< a \rho\ b >$ і $< b \rho\ a > \Rightarrow a = b$;

Антисиментичне відношення не є оберненим відношенням до симетричного. Існують відношення, які є одночасно симетричними та антисиметричними відношеннями.

- асиметричним – якщо для будь-яких *a* та *b* з множини *A* з того, що виконується $< a \rho\ b >$ випливає, що не виконується $< b \rho\ a >$.

Відношення є асиметричним, якщо воно одночасно є антисиметричним та антирефлексивним.

Відношення будемо називати повним відношенням, якщо для будь-яких *a* та *b* з множини *A* буде виконуватися $< a \rho\ b >$ або $< b \rho\ a >$.

У множині *A*, на якій встановлено відношення повного порядку, можна визначити мінімальний і максимальний елемент. Мінімальним елементом множини *A*, на якій задано відношення повного порядку ρ, називається такий елемент $a \in A$, що ні для якого іншого елемента $b \in A$ ($a \neq b$), не має місця відношення $b \rho a$. Аналогічно, максимальним елементом множини *A*, на якій задано відношення повного порядку ρ, називається такий елемент $a \in A$, що ні для якого іншого елемента $b \in A$ ($a \neq b$), не має місця відношення $a \rho b$.

За заданим на множині *A* відношенням ρ можна визначити відношення ρ^2 таким чином: $a \rho^2 b$ тоді і тільки тоді, якщо існує таке $c \in A$, що виконується $a \rho c$ та $c \rho b$ для будь-яких двох елементів $a \in A$та $b \in A$.

Останнє твердження можна узагальнити таким чином: *k*-степінь відношення ρ на множині *A* можна визначити рекурсивно:

- $a \rho^1 b$ тоді і тільки тоді, коли $a \rho b$;

- $a \rho^k b$ для *k*>1 тоді і тільки тоді, коли існує $c \in A$ таке, що $a \rho c$ та $c \rho^{k-1} a$ для будь-яких елементів $a, b \in A$.

Операція обертання зберігає всі властивості відношення. Якщо відношення ρ має певні властивості, то і обернене відношення ρ^{-1} теж буде мати ці ж властивості.

Введемо поняття транзитивного замикання відношення ρ на множині *A*. Відношння ρ^+ називається транзитивним замиканням відношення ρ на множині *A*, якщо $a \rho^+ b$ тоді і тільки тоді, якщо знайдеться таке *k*, що $a \rho^k b$ для всіх елементів $a, b \in A$.

Для транзитивного замикання має місце теорема:

Теорема 1. Транзитивне замикання ρ^+ деякого відношення ρ на множині *A* є відношення транзитивне. Якщо ρ' – будь-яке транзитивне відношення на множині *A*, таке, що $\rho \leq \rho'$, тоді $\rho^+ \leq \rho'$, тобто ρ^+ – найменше транзитивне відношення, яке включає ρ.

Для знаходження матриці транзитивного замикання відношення ρ на множині *A* зі скінченною кількістю елементів можна скористуватися наступною теоремою:

Теорема 2. Нехай *M* – матриця, що задає відношення ρ на множині *A*, яка містить *n* елементів. Тоді матриця $M^+ = M \cup M^2 \cup M^3 \cup \ldots \cup M^n$ задає транзитивне замикання відношення ρ.

Варіанти завдань.

1. Скласти програму побудови перетину та різниці за двома множинами *A* та *B*, які зберігаються в файлі як одновимірні масиви.

2. Скласти програму побудови об'єднання та симетричної різниці за двома множинами *A* та *B*, які зберігаються в файлі як одновимірні масиви.

3. Скласти програму побудови декартового добутку заданих множин. Дві задані множини *A* та *B* зберігаються в файлі як одновимірні масиви

4. Скласти програму перевірки еквівалентності двох множин, які зберігаються в файлі як одновимірні масиви. Передбачити варіанти, коли елементи множин впорядковані в лексикографічному порядку; та інший варіант, коли елементи множин не є впорядкованими в лексикографічному порядку.

5. Написати програму перевірки відношення включення $(A \subseteq B)$ для двох множин, які задаються користувачем з клавіатури.

6. Написати програму перевірки взаємної неперетинності двох множн, які задаються користувачем з клавіатури.

7. Розробити процедури перевірки властивостей (рефлективності, симетричності та антисиметричності) відношення, яке задане за допомогою матриці відношення.

8. Розробити процедури перевірки властивостей (антирефлективності, симетричності та асиметричності) відношення, яке задане за допомогою матриці відношення.

9. Розробити процедури перевірки властивості транзитивності відношення, яке задане за допомогою матриці відношення.

10. Розробити процедури перевірки, чи є відношення, задане за допомогою матриці, відношенням часткового порядку.

11. Розробити процедури перевірки, чи є відношення, задане за допомогою матриці, відношенням повного порядку.

12. Побудувати алгоритм знаходження мінімального та максимального елемента відношення повного порядку, яке задане матрицею відношення.

13. Побудувати матрицю відношення ρ^2 для відношення ρ, що задане на множині *A*. Відношення задається з клавіатури за допомогою впорядкованої множини пар, для яких виконується відношення ρ.

14. Побудувати матрицю оберненого відношення ρ^{-1} для відношення ρ, що задане на множині A. Відношення задається з клавіатури за допомогою впорядкованої множини пар, для яких виконується відношення ρ.

15. Побудувати процедуру обчислення матриці транзитивного замикання відношення ρ, що задане на множині A матрицею відношення.

Лабораторна робота № 2.

Розділ: Комбінаторика.

Тема роботи: *Генерування комбінаторних об'єктів*.

Необхідні теоретичні відомості:

В комбінаторних алгоритмах часто необхідно породжувати і досліджувати всі елементи деякого класу комбінаторних об'єктів. Найбільш загальні методи розв'язку таких задач базуються на алгоритмі пошуку з поверненням, хоча в деяких випадках об'єкти настільки прості, що доцільніше застосовувати спеціалізовані методи. Задачі, що потребують генерації комбінаторних об'єктів, виникають при обчисленні комбінаторних формул. Наприклад, часто приходиться обчислювати суми $\sum f(x_1, x_2, ..., x_n)$, де сумування виконується по всіх послідовностях $x_1, x_2, ..., x_n$, які задовольняють певні обмеження.

1. Алгоритм. Пошук з повторенням.

Метод пошуку з повторенням базується на принципі постійного розширення часткового розв'язку. Якщо таке розширення поточного часткового розв'язку неможливе, то повертаються до попереднього коротшого часткового розв'язку і пробують знову його розширити.

В загальному випадку припускаємо, що розв'язок складається із вектора $(a_1, a_2, a_3, ...)$ необмеженої, але скінченної довжини, що задовольняє певним обмеженням. Кожне $a_i \in A_i$, де A_i – скінчена лінійна впорядкована множина. Початковим частковим розв'язком буде пустий вектор ( ) і, на основі заданих обмежень, визначають, які елементи із A_1 є кандидатами на a_1. Позначимо таку підмножину кандидатів через S_1, зрозуміло, що $S_1 \subseteq A_1$. В загальному випадку для розширення часткового розв'язку від $(a_1, a_2, ..., a_{k-1})$ до $(a_1, a_2, ..., a_{k-1}, a_k)$ кандидати на роль a_k вибираються із $S_k \subseteq A_k$. Якщо для часткового розв'язку $(a_1, a_2, ..., a_{k-1})$ не існує розширення $(a_1, a_2, ..., a_{k-1}, a_k)$, то вважаємо $S_k = \varnothing$ і повертаємося до попереднього часткового розв'язку $(a_1, a_2, ..., a_{k-2})$, для якого вибираємо новий елемент a_{k-1}. Якщо ж елемент a_{k-1} вибрати не можна, то повертаємося ще на крок назад і вибираємо новий елемент a_{k-2} і т.д.

Процедура пошуку з поверненням приводить до алгоритмів експоненціальної складності, оскільки з припущення, що всі розв'язки мають довжину не більше *n*, дослідження щодо перебору варіантів здійснюється із загальної кількості $\prod_{k=1}^{n} |A_k|$ елементів. При припущенні, що всі $|A_k| = C$ константи, отримуємо експоненціальну складність $\prod_{k=1}^{n} |A_k| = C^n$.

Важливо пам’ятати, що пошук з поверненням це лише загальний метод, і безпосереднє його застосування може привести до алгоритмів, які будуть досить затратними щодо часу виконання. Тому, необхідно підходити до цього методу як до загальної схеми і пристосовувати її до конкретної задачі в залежності від поставленого завдання.

2. Перестановки.

Без обмеження загальності будемо вважати, що елементами множини є цілі числа і здійснювати перестановки на множині {1,2,3,…,*n*}.

В цьому випадку в загальному алгоритмі пошуку з поверненням перевірка умови $S_k \neq \varnothing$ буде відповідати умові $s_k \leq n$, а замість збереження всієї множини S_k достатньо зберігати в пам’яті лише останній її елемент s_k (оскільки перебір s_k здійснюється в порядку зростання їх значень).

Ефективна генерація перестановок.

Генерація *n!* перестановок на множині {1,2,3,…,*n*}, при якій сусідні дві перестановки будуть мінімально відрізнятися одна від одної. При такому алгоритмі ми мінімізуємо об’єм роботи, необхідний для породження перестановок. Дві сусідні перестановки будуть мінімально відрізнятися одна від одної, якщо вони відрізняються транспозицією двох сусідніх елементів. Таку послідовність перестановок легко побудувати рекурсивно.

Для *n*=1 існує єдина перестановка {1}. Припустимо, що ми маємо послідовність перестановок $\pi_1, \pi_2, \pi_3, \ldots$ на множині {1,2,3,…,*n*}, в якій послідовні перестановки відрізняються транспозицією двох сусідніх елементів. Розширимо кожну з таких (*n*-1)! перестановок, вставляючи елемент *n* на кожне із *n* можливих місць. Порядок згенерованих таким чином перестановок буде наступним:

$$\overbrace{\underbrace{123}_{12}\ \underbrace{132}_{12}\ \underbrace{312}_{12}\quad \underbrace{213}_{21}\ \underbrace{231}_{21}\ \underbrace{321}_{21}}^{1}$$

Таку послідовність перестановок можна породжувати ітеративно, отримуючи кожну перестановку із попередньої плюс деяка додаткова інформація. Це можна здійснити за допомогою трьох векторів: поточна перестановка $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, обернена до неї перестановка $p = (p_1, p_2, \ldots, p_n)$ і запису напрямку d_i, в якому зсуваєтья кожен елемент і (-1 – зсув вліво, +1 – зсув вправо і 0, якщо елемент залишається на місці). Елемент зсувається до тих пір, поки не досягне елемента, більшого, ніж він сам; в цьому випадку зсув припиняється. В цей момент напрямок зсуву елемента міняється на протилежний і пересувається менший за нього елемент, який можна зсунути. Оскільки зберігається перестановка, обернена до поточної π, то в π легко знайти місце наступного меншого елемента.

Даний алгоритм генерації всіх перестановок є лінійним, його складність *n*!+*o*(*n*).

3. Генерація підмножин заданої множини.

Генерація підмножин множини $(a_1, a_2, ..., a_n)$ еквівалентно породженню *n*-розрядних двійкових наборів a_i, які будуть належати підмножині, тоді і тільки тоді, коли *i*-й розряд буде рівний 1. Таким чином, задача генерації всіх можливих підмножин множини зводиться до задачі породження всіх можливих двійкових послідовностей довжини *n*. Очевидно, що найпростіший спосіб породження всіх можливих двійкових наборів довжини *n* є їх перелік в двійковій системі числення.

4. Генерація розміщень з повтореннями.

Генерація множини всіх розміщень з повтореннями довжини *k* із елементів $(a_0, a_1, ..., a_{n-1})$ еквівалентно генерації множини *k*-розрядних чисел в системі числення з основою *n*. Така генерація здійснюється наступним чином: на *r*-му місці в розміщені буде стояти елемент a_i, якщо цифра в *r*-му розряді відповідного числа рівна *i*. Всього розміщень з повтореннями є n^k.

5. Генерація вибірок (сполук).

Як правило, в реальних задачах потребується не всі вибірки з множини $\{a_0, a_1, ..., a_n\}$, а лише ті, що задовольняють умови задачі. Часто необхідно знаходити вибірки вказаної довжини *k* із множини *n* елементів C_n^k. Без обмеження загальності можна припустити, що основною множиною є множина *n* натуральних чисел {1,2,3,…,*n*}. Таким чином, будемо генерувати всі вибірки довжиною *k* із множини {1,2,3,…,*n*}.

Алгоритм генерації сполук в лексикографічному порядку.

Початкова вибірка {1,2,3,…,*k*}. Для утворення наступної вибірки, переглядаємо попередню справа наліво з метою визначити останній (самий правий) елемент, який ще не досяг свого максимального значення. Збільшуємо цей елемент на одиницю і всім наступним елементам справа від нього присвоюємо нові можливі найменші значення.

Даний алгоритм генерації є лінійним, його складність $C_n^k + o(C_n^k)$

6. Генерація композицій.

Розглянемо задачу генерації розбиттів натурального числа *n* на послідовність цілих невід'мних чисел $\{z_1, z_2, ..., z_k\}$, таку, що $z_1 + z_2 + .. + z_k = n$.

Послідовність $\{z_1, z_2, ..., z_k\}$ будемо називати композицією числа *n*, коли враховуються порядок чисел z_i. Як правило, шукають таку композицію, у якій всі z_i>0, або всі $z_i \geq 0$.

Послідовність $\{z_1, z_2, ..., z_k\}$ будемо називати розбиттям числа *n*, якщо всі z_i>0 і порядок чисел z_i не є важливим.

Розглянемо приклад розбиття числа *n*=3.

При *k*=2 композиціями числа 3 будуть: (1,2) і (2,1), якщо z_i>0.

При *k*=3 композицією числа 3 буде (1,1,1), якщо z_i>0.

При *k*=2 композиціями числа 3 будуть: (0,3), (1,2) і (2,1), (3,0) якщо $z_i \geq 0$.

При *k*=3 композицією числа 3 буде (0,0,3), (0,3,0), (3,0,0), (0,1,2), (0,2,1), (1,0,2), (1,2,0), (2,0,1), (2,1,0), (1,1,1), якщо $z_i \geq 0$.

А всі розбиття числа 3 будуть наступними: (3), (2,1), (1,1,1).

Алгоритм побудови композицій числа *n*, якщо всі $z_i \geq 0$. (для заданого *k*).

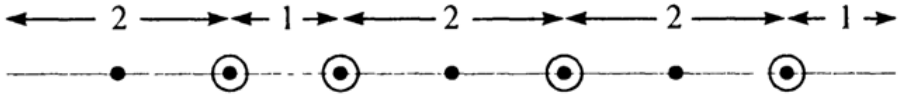
Композицію $\{z_1, z_2, ..., z_k\}$, де $z_i \geq 0$, числа $z_1 + z_2 + .. + z_k = n$ можна інтерпретувати наступним чином. Кожне значення z_i представимо, як наступну суму одиниць $z_i = 1_i + 1_i + ... + 1_i$, кількість яких z_i, а індекс *i* буде вказувати на приналежність одиниці до розбиття числа z_i. Таким чином, ми ввели *k* типів різних елементів $\{1_1, 1_2, ..., 1_k\}$. Тепер довільну композицію числа *n* можна представити, як суму, складену із *n* довільних одиниць множини $\{1_1, 1_2, ..., 1_k\}$. Сумуючи одиниці з однаковими індексами, отримаємо відповідне значення композиції z_i. Така відповідність є взаємнооднозначною, звідки випливає, що число композицій дорівнює числу вибірок з повтореннями $\overline{C}_n^k = C_{n+k-1}^{k-1} = C_{n+k-1}^n$. Кожну із вибірок C_{n+k-1}^n можна інтерпретувати як розміщення {0,1} довжини *n*+*k*–1, в якому буде *n* одиниць і *k*–1 нулів. Сумуючи в такій послідовності зліва направо одиниці між нулями будемо отримувати відповідне значення членів композиції z_i. Наприклад, вибірці (111011110011010111) з C_{13+6-1}^{13} буде відповідати композиція (3,4,0,2,1,3) числа 13.

Звичайно, що в цьому випадку легко застосувати алгоритм генерації підмножин деякої множини для послідовної побудови кожної композиції z_i.

Алгоритм побудови композицій числа *n*, якщо всі $z_i > 0$. (для заданого *k*).

Даний алгоритм отриманий з інтерпретації цілого числа *n* як відрізка прямої, що складається із деякої суми відрізків одиничної довжини.

На рисунку лінія розділена *n*-1 точками і композиція отримується через деякий набір цих точок і наступне сумування одиничних відрізків між ними. На малюнку бачимо композицію (2,1,2,2,1) числа 8.



Таким чином, елементами композиції будуть просто віддалі між суміжними точками. Очевидно, що кожна композиція числа *n* буде відповідати певному способу вибору підмножини із *n*–1 точок. Кожній точці можна поставити у відповідність двійкове число. І, таким чином, композиції буде відповідати якесь (*n*–1)-розрядне двійкове число. В такій інтерпретації композиція із *k* частин буде відповідати (*n*–1)-розрядному числу, в якому буде (*k*–1) одиничок. Отже, існує C_{n-1}^{k-1} таких композицій.

Розбиття числа *n*.

Розбиття відрізняються від композицій тим, що порядок компонент не є важливим. Композиції (1,1,2), (1,2,1) та (2,1,1) є одним і тим же розбиттям числа 4 (4=1+1+2).

Таким чином, розбиття *n* можна розглядати як мультимножину, яка записується наступним чином: $\{m_1 \bullet z_1, m_2 \bullet z_2, ..., m_k \bullet z_k\}$, де $n = \sum_{i=1}^{k} m_i \bullet z_i$, m_i входження z_i. І кожне розбиття задовольняє умову $z_1 > z_2 > ... > z_k$.

Ідея побудови послідовного списку розбиттів полягає в тому, щоб переходити від одного розбиття до іншого, розглядаючи самий правий елемент $m_k \bullet z_k$ розбиття. Якщо добуток $m_k z_k$ достатньо великий ($m_k > 1$), то можна виключити два z_k для того, щоб добавити ще одне z_k+1, (або включити одне z_k+1, якщо в поточний момент його немає. Якщо m_k=1, то $m_{k-1} z_{k-1} + m_k z_k$ достатньо велике для того, щоб добавити $z_{k-1} + 1$. Все, що залишається перетворюється у відповідне число одиниць і формується нове розбиття.

Даний алгоритм генерації розбиттів для *n*=5 буде генерувати такі розбиття: $\{1,1,1,1,1\}$, $\{2,1,1,1\}$, $\{2,2,1\}$, $\{3,1,1\}$, $\{3,2\}$, $\{4,1\}$, $\{5\}$.

Варіанти завдань:

1. Написати програму генерації слів, які можна отримати перестановками чотирьох перших літер заданого слова. Вхідне слово вводиться з клавіатури, причому літери в слові можуть повторюватися.

2. Написати програму генерації способів розселення 8 студентів в кімнатах: одномісній, тримісній і чотиримісній? На екран вивести 20 перших згенерованих способів.

3. Написати програму генерації дванадцятизначних чисел, які можна скласти з трьох двійок, чотирьох трійок і п’яти четвірок. На екран вивести 20 перших згенерованих чисел.

4. Написати програму генерації перестановок елементів скінченої послідовності, що складаються з букв. Встановлене обмеження: задана послідовність повинна містити не більше 4 букв.

5. Написати програму генерації способів, якими можна розсади 4 студентів в 2 ряди, причому в кожний ряд по 2 студенти.

6. Написати програму генерації п’ятизначних чисел, які можна утворити за допомогою цифр 0, 1, 2, 3, 4, якщо кожна з них у числі зустрічається один раз. На екран вивести 20 перших згенерованих чисел.

7. Написати програму генерації всіх підмножин деякої *n*-елементної множини літер латинського алфавіту {a, b, ..., z}. Розмірність вхідної

множини задається з клавіатури. Встановлене обмеження: розмірність вхідної множини не більша 5.

8. Написати програму генерації *k*-елементних підмножин множини з *n* чисел. Наприклад, для множини A = {1, 2, 3} (*n*= 3) та *k*=2 повинні бути згенеровані підмножини {1, 2}, {1, 3}, {2, 3}.

9. Написати програму генерації комбінації з трьох елементів з деякої вхідної множини *M*, яка містить 2 елементи.

10. Написати програму генерації різних тризначних телефонних номерів, які можна скласти з цифр від 0 до 9, якщо цифри в номері можуть повторюватись. На екран вивести 20 перших згенерованих номерів.

11. Написати програму генерації слів, які можна скласти, використавши три літери зі слова "*camel*".

12. Множина *M* утворена з чотирьох букв *A, B, C, D*. Написати програму генерації комбінації з трьох букв, що відрізняються один від одного хоча б одним елементом.

13. Написати програму генерації сполук з повтореннями по два елементи, які можна утворити із множини цифр 1, 2, 3, 4.

14. Написати програму генерації розбиття заданого натурального числа *n* на натуральні доданки. Розбиття, що відрізняються лише порядком доданків, вважаються однаковими. Наприклад, при *n*=4 розбиттями будуть 1+1+1+1, 2+1+1, 2+2, 3+1, 4. Встановлене обмеження: задане натуральне число не більше 5.

15. Написати програму генерації розбиття заданого натурального числа *n* на 3 натуральні доданки. Розбиття, що відрізняються лише порядком доданків, вважаються однаковими. Наприклад, при *n*=4 розбиттями будуть 1+1+1+1, 2+1+1, 2+2, 3+1, 4. Встановлене обмеження: задане натуральне число не менше 3 і не більше 7.

Лабораторна робота № 3.

Розділ: Алгебра логіки.

Тема роботи: *Еквівалентні перетворення функцій алгебри логіки. Побудова досконалих нормальних форм, канонічного полінома Жегалкіна та мінімізація функції алгебри логіки. Класи функцій.*

Необхідні теоретичні відомості:

Основними об'єктами алгебри логіки є елементарні висловлювання: висловлювання, про які однозначно можна сказати що вони істинні або хибні.

Функцією алгебри логіки (логічною функцією) від n змінних називатимемо функцію, яка приймає значення 0 та 1 (істинне, хибне), і аргументи якої теж можуть приймати значення 0, 1 (істинне, хибне).

Логічні операції можна зобразити у вигляді наступної таблиці:

x_1	x_2	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Основними вважають такі операції алгебри логіки:

$f_3(x_1) = \overline{x_1}$ – операція заперечення;

$f_1(x_1, x_2) = x_1 \wedge x_2 = x_1 x_2$ – операція кон'юнкції (логічне множення);

$f_7(x_1, x_2) = x_1 \vee x_2$ – операція диз'юнкції (логічне додавання);

$f_{13}(x_1, x_2) = x_1 \to x_2$ – операція імплікації;

$f_9(x_1, x_2) = x_1 \approx x_2$ – операція еквівалентності (рівнозначність);

$f_{14}(x_1, x_2) = x_1 / x_2 = \overline{x_1 x_2}$ – операція Шеффера;

$f_8(x_1, x_2) = x_1 \downarrow x_2 = \overline{x_1 \vee x_2}$ – операція Пірса;

$f_6(x_1, x_2) = x_1 \oplus x_2 = \overline{x_1 \approx x_2}$ – операція додавання за модулем два (нерівнозначність).

Порядок виконання операцій такий: при відсутності дужок виконуються операція заперечення (яка відіграє роль дужок, коли стоїть над виразом алгебри логіки), потім кон'юнкція, а всі інші операції виконуються в порядку їх слідування у виразі зліва направо.

Основні закони операцій алгебри логіки.

Закон подвійного заперечення $\overline{\overline{x}} = x$.

Закони де-Моргана (закони двоїстості) $\overline{x \wedge y} = \overline{x} \vee \overline{y}$, $\overline{x \vee y} = \overline{x} \wedge \overline{y}$.

Закони ідемпотентності $x \wedge x = x$, $x \vee x = x$.

Закони поглинання $x \vee (x \wedge y) = x$, $x \wedge (x \vee y) = x$, $x \oplus x = 0$.

Операції з константами $\bar{0}=1,\ \bar{1}=0,\ 1\wedge x=x,\ 1\vee x=1,\ 0\wedge x=0,$
$0\vee x=x,\ x\oplus 0=x.$

Закон виключення третього $x\vee\bar{x}=1.$

Закон суперечливості (протиріччя) $x\wedge\bar{x}=0.$

Для спрощення формул, крім вказаних вище законів, використовують властивості операцій (комутативність, асоціативність та дистрибутивність), а також співвідношення між операціями.

Якщо функція алгебри логіки на всіх можливих наборах значень своїх змінних приймає істинне значення, вона називається тавтологією (тотожно істинною функцією). Якщо функція алгебри логіки на всіх можливих наборах значень своїх змінних приймає хибне значення, вона називається суперечністю (тотожно хибною функцією).

Якщо значення функції алгебри логіки не залежить від зміни значень деякої змінної, то така змінна називається неістотною змінною.

Булевою алгеброю називається множина М, що складається не менше, ніж з двох елементів, на якій визначені три операції – диз'юнкції $(x\vee y)$, кон'юнкції (xy), заперечення $(\bar{x})$.

Основні формули, що виражають логічні функції через еквівалентні формули булевої алгебри:

$$x/y=\overline{xy}=\bar{x}\vee\bar{y},\quad x\downarrow y=\overline{x\vee y}=\bar{x}\ \ \bar{y},$$

$$x\equiv y=xy\vee\bar{x}\ \ \bar{y},\quad x\oplus y=\bar{x}y\vee x\bar{y},\quad x\to y=\overline{x\bar{y}}=\bar{x}\vee y.$$

Алгебра, побудована на базисі функцій кон'юнкції, додавання за модулем два та функції-константи одиниці називається алгеброю Жегалкіна. Будь-яку функцію алгебри логіки можна привести до функції у алгебрі Жегалкіна, тобто записати формулою із використанням лише функцій кон'юнкції, додавання за модулем два та функції-константи одиниці.

Основні формули, що виражають логічні функції через еквівалентні формули алгебри Жегалкіна:

$$\bar{x}=x\oplus 1,\quad x\vee y=x\oplus y\oplus xy,$$

$$x\to y=1\oplus y\oplus xy,\quad x\oplus y=\bar{x}y\vee x\bar{y}.$$

Нормальні форми булевої алгебри.

Базовими поняттями при побудові нормальних форм у алгебрі Буля є елементарні диз'юнкція та кон'юнкція, які в свою чергу є першим етапом мінімізації булевих функцій.

Елементарною кон'юнкцією (елементарною диз'юнкцією) називають вираз, що є кон'юнкцією (диз'юнкцією) елементарних висловлювань. Тотожно істиний (хибний) вираз теж вважають елементарною кон'юнкцією (елементарною диз'юнкцією).

Диз'юнктивною нормальною формою (ДНФ) (Кон'юнктивною нормальною формою (КНФ)) називається диз'юнкція (кон'юнкція) елементарних кон'юнкцій (диз'юнкцій). Для кожної нетотожно істиної (хибної) функції алгебри логіки існує еквівалентна їй ДНФ (КНФ). Тотожно хибний (істиний) вираз теж вважається ДНФ (КНФ).

Елементарну кон'юнкцію (диз'юнкцію), що включає в себе всі змінні заданої функції алгебри логіки називатимемо конституентою одиниці (конституентою нуля).

Досконалою диз'юнктивною нормальною формою (ДДНФ) (Досконалою кон'юнктивною нормальною формою (ДКНФ)) називається така ДНФ (КНФ), у якої кожний елементарний добуток є конституентою одиниці (конституентою нуля).

Будь-яка булева функція $f(x_1, x_2, ..., x_n) \neq 0$ ($f(x_1, x_2, ..., x_n) \neq 1$) єдиним способом може бути представленою у вигляді ДДНФ (ДКНФ).

Алгоритм побудови відповідної до заданої функції алгебри логіки ДДНФ (ДКНФ). (на основі таблиці істинності)

1. Побудувати таблицю істинності заданої функцій алгебри логіки.
2. Для кожного набору вхідних змінних, для якого функція приймає істине (хибне) значення записуємо відповідну йому конституенту одиниці (конституенту нуля).
3. Будуємо диз'юнкцію (кон'юнкцію) отриманих в попередньому пункті елементарних кон'юнкцій (диз'юнкцій).

При мінімізації булевих функцій на основі побудованих ДДНФ визначають відповідні заданій формулі скорочену та тупікову диз'юнктиіну нормальні форми. Мінімальна нормальна форма функції алгебри логіки вважається така її тупікова диз'юнктивна нормальна форма, яка містить найменшу кількість складових. При мінімізації ефективними є метод Мак-Класкі, Блейка-Порецького чи Квайна.

Функціонально повні системи булевих функцій

Булева функція $f(x_1, x_2, ..., x_n)$ належить до класу функцій, які зберігають константу 0, якщо $f(0,0,...,0) = 0$.

Булева функція $f(x_1, x_2, ..., x_n)$ належить до класу функцій, що зберігають константу 1, якщо $f(1,1,...,1) = 1$.

Булеві функції $f_1(x_1, x_2, ..., x_n)$ і $f_2(x_1, x_2, ..., x_n)$ називаються двоїстими одна одній, якщо виконується співвідношення: $f_1(x_1, x_2, ..., x_n) = \overline{f_2(\overline{x_1}, \overline{x_2}, ..., \overline{x_n})}$.

Булева функція $f(x_1, x_2, ..., x_n)$ належить до класу самодвоїстих функцій, якщо $f(x_1, x_2, ..., x_n) = \overline{f(\overline{x_1}, \overline{x_2}, ..., \overline{x_n})}$. Очевидно, що булева функція є самодвоїстою, якщо на будь-яких двох протилежних наборах $(\gamma_1, \gamma_2, ..., \gamma_n)$ та $(\overline{\gamma_1}, \overline{\gamma_2}, ..., \overline{\gamma_n})$ вона приймає протилежні значення.

Булева функція $f(x_1, x_2, ..., x_n)$ належить до класу лінійних функцій, якщо її канонічний поліном Жегалкіна є лінійним, тобто це функція, яка може бути подана у вигляді $f_1(x_1, x_2, ..., x_n) = c_0 \oplus c_1 x_1 \oplus c_2 x_2 \oplus ... \oplus c_n x_n$, де коефіцієнти $c_i \in \overline{0,1}$.

Булева функція $f(x_1, x_2, ..., x_n)$ належить до класу монотонних функцій, якщо для будь-яких двох наборів $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n)$ та $\beta = (\beta_1, \beta_2, ..., \beta_n)$ таких, що $\alpha \geq \beta$ має місце нерівність $f(\alpha_1, \alpha_2, ..., \alpha_n) \geq f(\beta_1, \beta_2, ..., \beta_n)$. Двійковий набір $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n)$ не менше двійкового набору $\beta = (\beta_1, \beta_2, ..., \beta_n)$, (тобто $\alpha \geq \beta$), якщо для кожної пари $(\alpha_i, \beta_i), i = \overline{1,n}$ справедливе співвідношення $\alpha_i \geq \beta_i$.

Варіанти завдань:

1. Для заданої функції алгебри логіки (варіанти алгебраїчних формул подані нижче) виконати наступні завдання:

1.1. Спростити задану формулу та звести її до формули у алгебрі Буля та до формули у алгебрі Жегалкіна.

1.2. Побудувати таблицю істинності до заданої формули.

1.3. Записати ДДНФ та ДКНФ для заданої формули та мінімізувати її.

1.4. Знайти методом невизначених коефіцієнтів відповідний канонічний многочлен Жегалкіна.

2. Здійснити програмну реалізацію одного із наступних пунктів завдання

(вхідними даними вважати вектор значень логічної функції)

Варіанти завдань для програмної реалізації.

2.1. Визначити, чи є задана функція самодвоїстою.

2.2. Визначити, чи задана функція зберігає константу 0 і чи вона є суперечністю.

2.3. Визначити, чи задана функція зберігає константу 1 і чи вона є тавтологією.

2.4. Визначити, чи є задана функція є монотонною.

2.5. Визначити, чи є задана функція є лінійною.

2.6. Побудувати всі конституенти нуля відповідні до даної функції.

2.7. Побудувати всі конституенти одиниці відповідні до даної функції.

2.8. Визначити, чи містить задана функція неістотні змінні.

Варіанти заданої формули алгебри логіки:

1. $f(x,y,z) = \overline{x \to y\bar{z}} \oplus (x \vee \bar{y} / z);$

2. $f(x,y,z) = \overline{x \,\square\, y \vee \bar{z}} \to (x\bar{y} \downarrow z);$

3. $f(x,y,z) = x\bar{z}(x \vee y\bar{z}) \,\square\, (x \to \bar{y});$

4. $f(x,y,z) = \overline{x \oplus y\bar{z}} \vee (x \to \bar{y}z);$

5. $f(x,y,z)=\overline{x\downarrow y\bar{z}}\to(x\vee\bar{y}\vee z);$

6. $f(x,y,z)=(\bar{x}\to y\bar{z})\to(x\downarrow\bar{y}\vee z);$

7. $f(x,y,z)=\overline{x\oplus y\bar{z}}\ \square\ (x\bar{y}/z);$

8. $f(x,y,z)=\overline{\bar{x}\to y}\ \square\ (x\vee\bar{y}/z);$

9. $f(x,y,z)=\overline{x\downarrow y\bar{z}}\ \square\ ((x\bar{z}\vee\bar{y})\to z);$

10. $f(x,y,z)=\overline{x/y\bar{z}}\oplus(x\vee(\bar{y}\to z));$

11. $f(x,y,z)=\overline{xy\bar{z}}\to(x\oplus\overline{\bar{y}\vee z});$

12. $f(x,y,z)=(x\to\bar{y}z)\ \square\ \overline{x\bar{y}\to yz};$

13. $f(x,y,z)=y\bar{z}\,\overline{x\to y\bar{z}}/(x\ \square\ \bar{y}z);$

14. $f(x,y,z)=(x\vee(\bar{y}\oplus z))\to\overline{x\vee y\bar{z}};$

15. $f(x,y,z)=\overline{x\oplus y\bar{z}}\to(x\bar{y}\ \square\ \bar{z});$

Лабораторна робота № 4.

Розділ: Теорія графів.

Тема роботи: *Основні елементи та типи графів. Методи задання графів.*

Необхідні теоретичні відомості:

Основні означення.

Нехай V − деяка не порожня скінченна множина, а $V^{(2)}$ − множина всіх невпорядкованих пар елементів множини V.

Графом (неорієнтованим графом) $G=(V, E)$ називається пара множин (V, E), де E − довільна підмножина множини $V^{(2)}$ ($E \subseteq V^{(2)}$).

Елементи множини V називаються вершинами графа G, а елементи множини E − ребрами графа G. Відповідно V називається множиною вершин і E − множиною ребер графа G.

Якщо $(v, w) \in E$, то кажуть, що вершини v і w є суміжними, у противному разі вершини v і w є несуміжними. Якщо $e=(v,w)$ – ребро графа, то вершини v і w називаються кінцями ребра e. У цьому випадку кажуть також, що ребро *е з'єднує* вершини v і w. Вершина v і ребро e називаються інцидентними, якщо v є кінцем e.

Два ребра називаються суміжними, якщо вони мають спільну вершину.

Існує декілька способів задання графа $G=(V, E)$.

1. Задання кожної з множин V і E за допомогою переліку їх елементів.

2. За допомогою матриці суміжності (квадратна $n \times n$-матриця, в якій елемент a_{ij} i-го рядка і j-го стовпчика дорівнює 1, якщо вершини v_i та v_j з номерами i та j суміжні, і дорівнює 0 у противному разі).

3. За допомогою матриці інцидентності ($n \times m$-матриця, в якій елемент b_{ij} i-го рядка і j-го стовпчика дорівнює 1, якщо вершина v_i з номером i інцидентна ребру e_j з номером j, і дорівнює 0 у противному разі).

4. За допомогою К-списку (множиною ребер графа, кожне з яких визначене як впорядкована пара вершин, які утворюють ребро графа).

Деякі поняття в теорії графів:

Шляхом (маршрутом) у графі $G=(V, E)$ називається така послідовність ребер, що кожні два сусідні ребра в цій послідовності мають спільну вершину. Вершина v_1 називається початком шляху, а вершина v_{k+1} – кінцем шляху. Всі інші вершини цього шляху називаються проміжними, або внутрішніми, вершинами.

Довжина шляху дорівнює кількості ребер, які його утворюють. Шляхом довжини 0 вважається послідовність, що складається з єдиної вершини. Шлях, в якому всі ребра попарно різні, називається простим. Шлях, в якому всі проміжні вершини попарно різні, називається елементарним.

Шлях у якому початкова та кінцева вершини співпадають називається замкненим (або циклічним).

Деякі алгоритми в теорії графів.

1. Алгоритм пошуку шляхів і контурів.

Нехай задано граф G(x) матрицею суміжності A{a_{ij}}. Знайти кількість шляхів і контурів довжини S в даному графі.

Для розв'зку здачі пошуку шляхів та контурів графа скористаємось наступною теоремою:

Теорема 1. Нехай A − матриця суміжності графа $G=(V, E)$ з n вершинами ($|V|=n$). Тоді елемент $a_{ij}^{(k)}$ i-го рядка і j-го стовпчика матриці A^k дорівнює кількості шляхів довжини k, які ведуть в графі G з вершини v_i у вершину v_j.

Ще одним наслідком цієї теореми є те, що в графі G(x) існує хоча б один шлях довжини S тоді і лише тоді, коли $A^s \neq 0$ і не існує шляхів довжини S тоді і лише тоді, коли $A^s=0$, починаючи з деякого S. Кількість контурів довжини S буде виражати відповідний діагональний елемент матриці A^s.

2. Алгоритм пошуку числа контурів довжини «3» в повному антисиметричному графі.

Граф називається повним антисиметричним, якщо кожна пара його вершин з'єднана тільки в одному напрямку. Нехай G(x) – повний антисиметричний граф з матрицею суміжності A={a_{ij}}, то кількість контурів довжини 3 (контур довжини 3 в графі з елементарним контуром найменшої довжини) такого графа дорівнює:

$$\varepsilon = \left| \frac{1}{12} n(n-1)(2n-1) - \frac{1}{2} \sum_{i=1}^{n} (r_i)^2 \right| \quad \text{де } r_i = \sum_{j=1}^{n} a_{ij}.$$

3. Алгоритм перевірки графа на зв'язність.

Дві вершини v і w називаються ***досяжними (зв'язаними)***, якщо існує шлях із кінцями v та w. Граф називається ***зв'язним***, якщо будь-яка пара його вершин є зв'язною.

Нехай A − матриця суміжності графа $G=(V, E)$ з n вершинами.

В графі G вершини v_i і v_j ($i \neq j$) є зв'язаними тоді і тільки тоді, коли елемент i-го рядка і j-го стовпчика матриці $A+A^2+A^3+\ldots+A^{n-1}$ не дорівнює нулю.

Це випливає з теореми 1 та тієї простої властивості, що коли в графі G з n вершинами існує шлях між вершинами v_i і v_j ($i \neq j$), тоді між цими вершинами обов'язково існує шлях довжини не більшої, ніж n-1.

Крім того, щоб вилучити умову $i \neq j$ для встановлення зв'язності між будь-якими вершинами графа G можна використовувати матрицю $M^{(n)}=I_n+A+A_2+A_3+\ldots+A_{n-1}$, де I_n – одинична матриця порядку n.

Наслідок. Граф G буде зв'язним тоді і тільки тоді, коли в матриці $M^{(n)}$ немає нульових елементів.

4. Алгоритм перевірки графа на повноту.

Граф $G=(V, E)$ називається повним, якщо будь-які дві його вершини суміжні (тобто $E=V^{(2)}$). Повний граф з n вершинами позначається K_n.

5. Алгоритм перевірки чи заданий граф є ейлеровим.

Цикл, який містить усі ребра графа, називається ейлеровим циклом. Зв'язний граф, який має ейлерів цикл, називається ейлеровим графом**.**

Теорема 2. (теорема Ейлера). Зв'язний граф G є ейлеревим тоді і тільки тоді, коли степені всіх його вершин парні.

Степенем $deg(v)$ вершини v називається кількість ребер, інцидентних цій вершині.

6. Алгоритм пошуку найкоротшого шляху

Нехай задано довільний граф G. Необхідно побудувати такий шлях, що з'єднує дві задані вершини a і b, який містить найменше число дуг (шлях найкоротшої довжини).

1. Присвоїти вершині a мітку 0.

2. Якщо вершина $x \neq a$, є суміжна з вершиною a, то присвоїти кожній такій вершині мітку 1.

3. Нехай $A(m)$ (m = 0,1,2,…) – множина вершин, що мають мітку m. Позначимо $\Gamma(A(m+1))$ – множину вершин, суміжних із вершинами множини $A(m)$. Присвоїти непоміченим ще вершинам множини $A(m+1)$ мітку m+1.

4. Процес присвоєння вершинам міток зупинити, як тільки вершина b отримає деяку мітку n ,($b \in A(n)$).

5. Довжина шляху, що з'єднує дві задані вершини a і b, який містить найменше число дуг буде дорівнювати n. Розв'язком задачі є шлях $\mu = \{a, x_1, x_2, ..., x_{n-1}, b\}$, де x_i – вершина, яка має мітку i та є суміжна з вершиною x_i.

Зауваження 1. Найкоротший шлях між двома заданими вершинами може бути не єдиним.

Зауваження 2. Якщо на деякому кроці неможливе присвоєння мітки m+1 вершинам в силу того, що множина $\Gamma(A(m))$ – пуста, і вершина b не отримала мітки, то це означає, що в графі G не існує жодного шляху, що з'єднує вершину a з вершиною b.

7. Алгоритм знаходження хроматичного числа простого, неорієнтованого графа

Хроматичне число графа дорівнює мінімальній кількості фарб, необхідній для його правильного розфарбування. Правильним розфарбуванням графа вважається таке розфарбування його вершин, при якому жодні дві суміжні вершини не мають однакового кольору.

1. Обчислити степені вершин. Покладемо k=1.

2. Проглядаємо вершини в порядку незростання степеня і розфарбовуємо першу не розфарбовану вершину в колір k.

3. Проглядаємо вершини в порядку незростання степеня і розфарбовуємо в колір k всі вершини, які є несуміжними вершинам, уже розфарбованим у колір k.

4. Якщо всі вершини отримали колір, то k – хроматичне число заданого графа. Інакше збільшуємо k на 1 і повертаємося до пункту 2.

Варіанти завдань:

1. Задається початкова і кінцева вершини шляху та його довжина. Знайти кількість таких шляхів у графі, що заданий матрицею суміжності.

2. Задається вершина та довжина циклу. Знайти кількість таких циклів у графі, що заданий матрицею суміжності.

3. Знайти число контурів в повному антисиметричному графі довжини «3», який заданий матрицею суміжності. (задається лише розмірність матриці)

4. Знайти кількість шляхів (включаючи їх перелік) від вершини x_i до x_j (які задаються користувачем) для графа, заданого *K*-списком.

5. Знайти найкоротший шлях від вершини x_i до x_j (які задаються користувачем) для графа, заданого *K*-списком.

6. Перетворити задану матрицю суміжності *A* у матрицю інцидентності.

7. Перевірити чи граф, заданий *K*-списком, є зв'язним.

8. Перевірити чи граф, заданий матрицею інцидентності, є повний і чи є цей граф ейлеровим.

9. Знайти хроматичне число графа, який заданий *K*-списком.

10. Перетворити *K*-список, яким задано граф, у матрицю суміжності.

11. Перетворити *K*-список, яким задано граф, у матрицю інцидентності.

12. Перетворити матрицю суміжності, якою зданий граф, у *K*-список.

13. Перетворити матрицю інцидентності, якою зданий граф, у *K*-список.

14. Перетворити матрицю суміжності, якою задано граф, у матрицю інцидентності.

15. Перетворити матрицю інцидентності, якою задано граф, у матрицю суміжності.

Лабораторна робота № 5.

Розділ: Теорія графів.

Тема роботи: *Алгоритм Форда-Фалкерсона для знаходження максимального потоку мережі*.

Необхідні теоретичні відомості:

Алгоритм Форда-Фалкерсона призначений для знаходження потоку максимальної величини. Алгоритм працює на мережевих графах (транспортних сітках) – ваговий орграф, що має одну вершину виходу (витік) та одну вершину входу (стік). Вага дуги визначає величину максимально допустимого потоку по даній дузі; і виражається додатнім дійсним числом.

Схематично алгоритм складається із 2 кроків:

1. насичення маршрутів;
2. перерозподіл потоку.

При насиченні маршрутів:

a. довільним чином перебираємо всі існуючі маршрути від витоку до стоку;
b. визначаємо мінімальну вагу дуги в кожному маршруті;
c. насичуємо нею всі дуги знайденого маршруту.

Дугу будемо вважати насиченою, якщо сума величин потоків, які проходть по ній з усіх знайдених маршрутів буде дорівнювати її вазі.

Якщо всі існуючі маршрути знайдено, а у графі залишилися ненасичені інцидентні стоку чи витоку дуги, переходимо до другого кроку – перерозподіл потоку, який дає можливість збільшити значення результуючого потоку графа.

При перерозподілі потоку в графі:

d. знаходимо «псевдомаршрути», що проходять по ненасичених дугах, у встановленому напрямку і по «непустих» дугах (дугах, які вже були задіяні у попередніх маршрутах) у протилежному до встановленого напрямку;
e. визначаємо мінімальну вагу дуг, що становлять знайдений псевдомаршрут;
f. додаємо цю вагу до тих дуг, по яких ми рухалися у вказаному напрямку та віднімаємо її від тих дуг, по яких рух відбувався в протилежному напрямку.

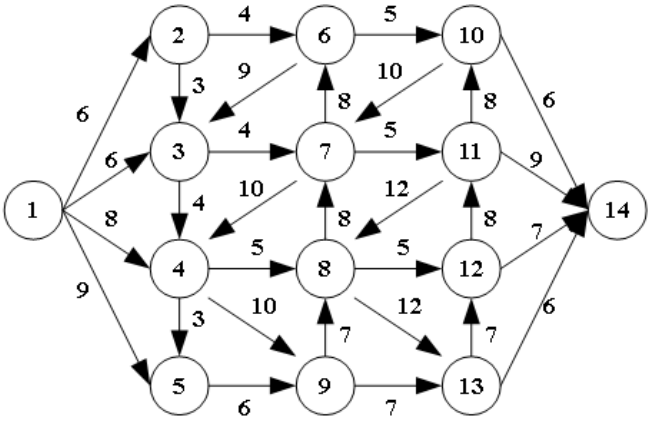
Виконання перерозподілу потоку закінчується при умові насичення всіх дуг, які інцидентні стоку чи витоку, чи перебору всіх можливих ланцюгів, що проходять по ще ненасичених дугах.

Величина максимально можливого потоку буде дорівнювати сумі величин потоків, що проходять по інцидентних витоку дугах, що дорівнює сумі величин потоків, що виходять з інцидентних джерелу дуг.

Крім того, доцільно здійснити перевірку «умови рівноваги» для кожної вершини графа: сума величин потоків, що входить у вершину дорівнює сумі величин потоків, які виходять з неї.

Проілюструємо виконання алгоритму Форда-Фалкерсона на прикладі.

Задана мережа. Визначити максимальний допустимий потік в цій мережі.



Розв'язок:

Спираючись на алгоритм Форда-Фалкерсона, розглянемо такі маршрути:

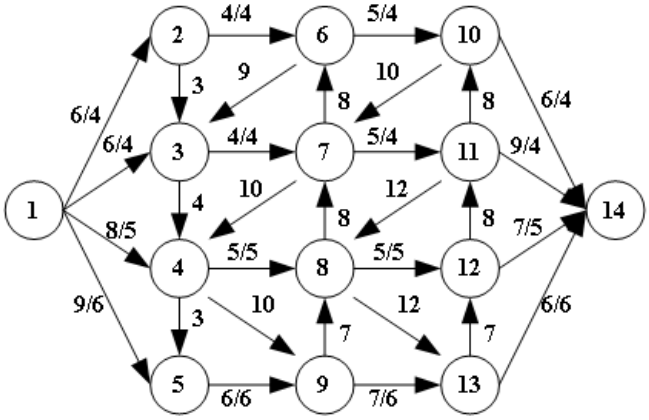
1-5-9-13-14;

1-4-8-12-14;

1-3-7-11-14;

1-2-6-10-14;

Розставивши потоки біля відповідних дуг, отримаємо:



Таким чином, насиченими стали дуги (2,6); (3,7); (4,8); (8,12); (5,9); (13,14).

Далі будемо вибирати маршрути так, щоб наситились дуги, інцидентні витоку.

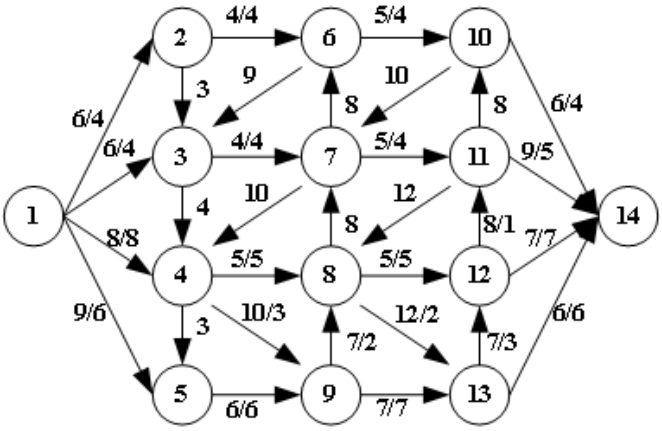
1-4-9-13-12-14 (тут потік кожної дуги збільшується на одиницю);

1-4-9-8-13-12-14 (тут також на одиницю);

Дуга (12,14) вже стала насиченою.

При маршруті 1-4-9-8-13-12-11-14 потік кожної дуги збільшуємо на одиницю.

Після цієї послідовності ланцюгів отримаємо насичену дугу 1-4.



Наступна послідовність ланцюгів дає нам остаточний результат:

1-3-4-9-8-13-12-11-14 (тут потік кожної дуги збільшується на двійку);

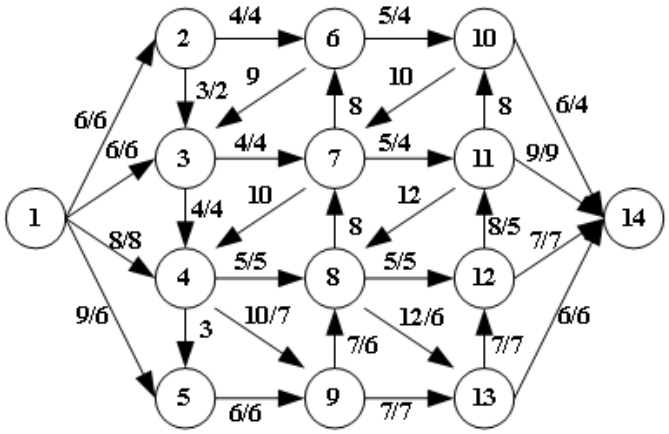
Таким чином, дуга (1,3) вже стала насиченою.

1-2-3-4-9-8-13-12-11-14 (тут потік кожної дуги знову збільшується на двійку).

Таким чином, дуга (1,2) вже стала насиченою.

Оскільки ненасиченою серед дуг, інцидентних витоку залишилася лише одна дуга (1,5), спробуємо провести перерозподіл потоків. Єдино-можливим варіантом псевдомаршруту буде такий, що починатиметься із 1-5-4-…, але дуга (4,5) є пустою, тому рухатися по ній в протилежному напрямку не можна. Таким чином, побудувати псевдомаршрут для перерозподілу потоків неможливо.

Результуючий граф матиме вигляд:



Отже, максимально можливий потік в мережі дорівнює: 4 +9 +7 +6 = 26

Насичені дуги: 1-4,1-3,1-2, 5-9, 9-13, 13-14, 13-12, 4-8, 8-12, 12-14, 3-7, 11-14 , 2-6, 3-4.

Здійснимо перевірку (виконання «умови рівноваги»):

- сумарний потік з вершини входу (витоку) дорівнює 6+6+8+6 = 26.
- сумарний потік на вході дорівнює 4+9+7+6=26;
- для вершини 3 – сумарний потік, що входить у вершину дорівнює 6+2=8 і сумарний потік, що виходить з неї дорівнює 4+4=8;

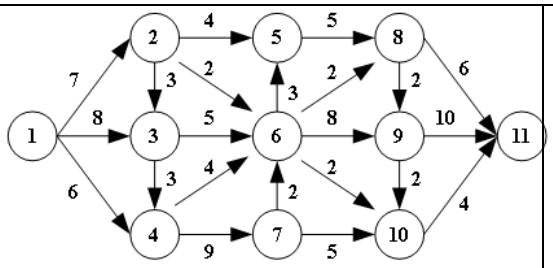
аналогічно можна перевірити «умови рівноваги» для кожної вершини графа.

Завдання:

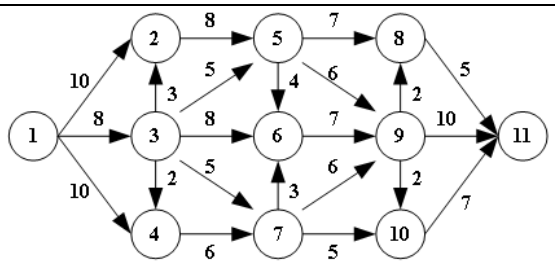
Знайти максимальний потік у заданому графі за допомогою алгоритма Форда-Фалкерсона і провести перевірку «умови рівноваги» для кожної вершини.

Варіанти завдань:

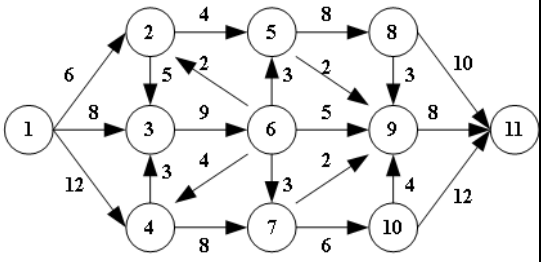
1



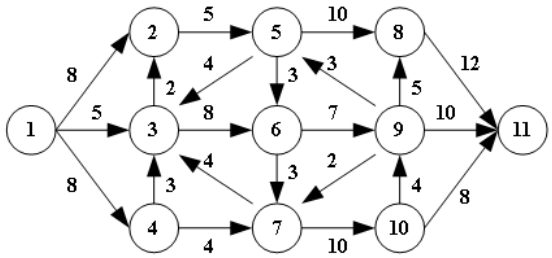
2



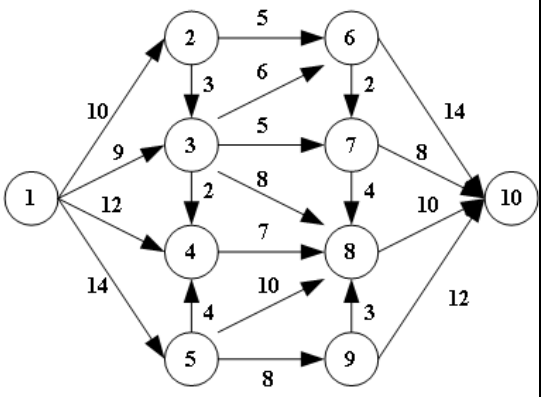
3



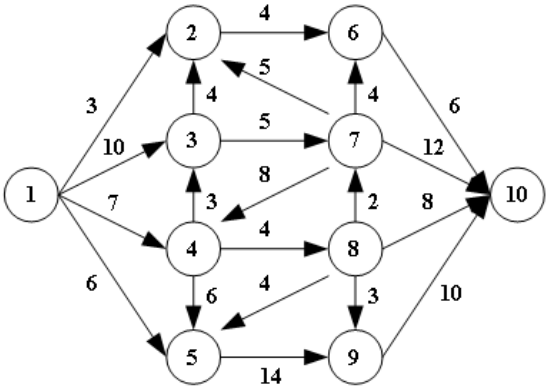
4.



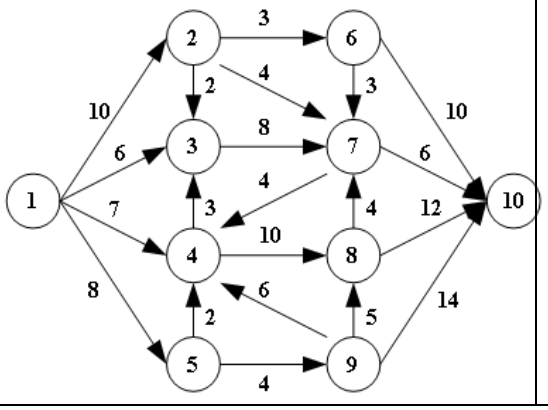
5



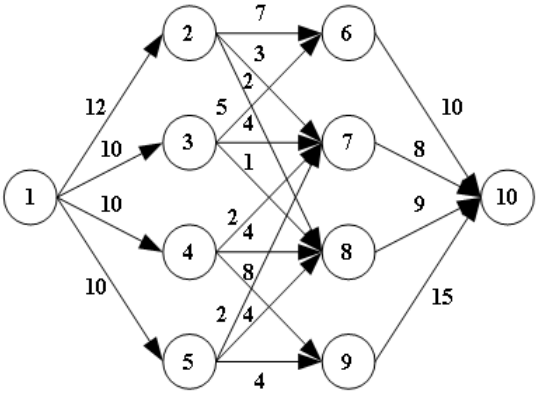
6.

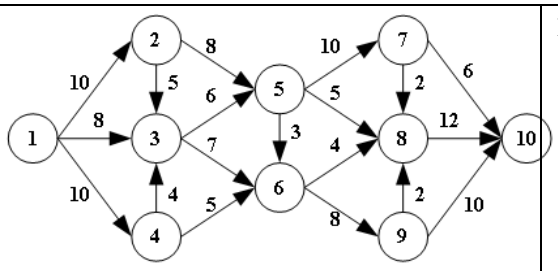
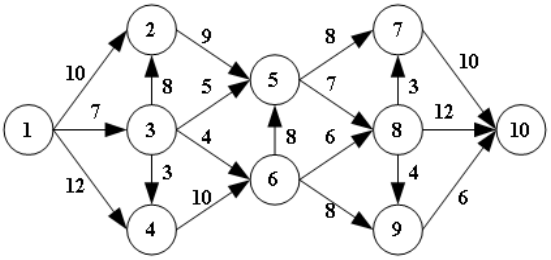
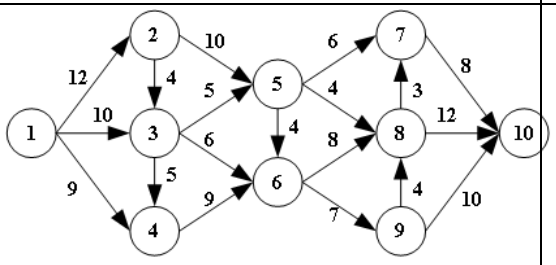
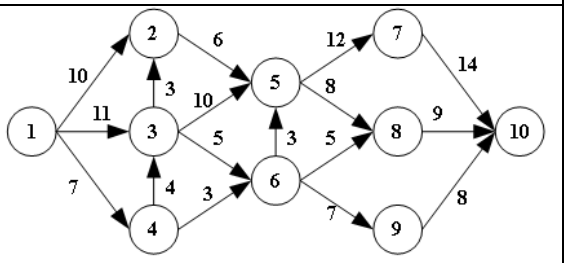


7



8.



9	 1 2 3 4 5 6 7 8 9 10 10 8 10 8 5 4 6 5 7 3 10 5 4 8 10 2 2 12 6 10	10.	 1 2 3 4 5 6 7 8 9 10 10 7 12 9 8 5 4 10 8 5 8 6 8 3 4 12 10 6
11	 1 2 3 4 5 6 7 8 9 10 12 10 9 10 4 5 6 5 9 4 6 8 7 3 4 12 8 10	12.	 1 2 3 4 5 6 7 8 9 10 10 11 7 6 3 10 5 4 3 3 5 8 3 5 12 8 7 14 9 8

Лабораторна робота № 6.

Розділ: **Теорія граматик та формальних мов**.

Тема роботи: *Побудова граматики та розпізнавання виразу*.

Вхідні дані – символьний ланцюжок (у вхідному файлі або з клавіатури), вихід – відповідь, чи даний ланцюжок генерується заданою граматикою.

Підготовка до лабораторної роботи: побудувати граматику за заданим варіантом (описати вхідні алфавіти та множину правил виводу), продумати організацію програми.

Варіанти завдань

1. Граматика генерує послідовність ідентифікаторів і цілих чисел без знаку. Ідентифікатори складаються з букв, числа − з цифр. Довжина ідентифікаторів і чисел довільна. Елементи послідовності відокремлюються комою, після якої може бути довільна кількість пробілів.
2. Граматика генерує простий (без дужок) логічний вираз. У виразі можуть зустрічатись операції **||** , **&&**, **!** та ідентифікатори. Перший символ ідентифікатора − буква, далі − буква або цифра. Довжина ідентифікатора не обмежена.
3. Граматика генерує простий (без дужок) арифметичний вираз. У виразі можуть зустрічатись ідентифікатори (складаються тільки з букв, довжина довільна), цілі десяткові числа (складаються тільки з цифр, довжина довільна) та односимвольні арифметичні операції (довільна кількість).
4. Граматика генерує послідовність цілих двійкових та десяткових чисел. Двійкове число закінчується символом **b** або **B** і складається лише з нулів та одиниць, десяткове − складається з цифр і може закінчуватись символом **d** . Числа відокремлюються принаймні одним пробілом, на початку ланцюжка і в його кінці пробілів немає.
5. Граматика генерує цілі десяткові та шістнадцяткові числа без знаку. Довжина чисел не обмежена. Десяткове число складається з цифр і може закінчуватись символом **d** або **D**, шістнадцяткове число може утворюватись з цифр, букв **A, B, C, D, E, F, a, b, c, d, e, f** і закінчується буквою **H** або **h** .
6. Граматика генерує двійкові та шістнадцяткові числа без знаку. Довжина чисел не обмежена. Двійкове число записується за допомогою лише нулів та одиниць і закінчується символом **b** або **B**,

шістнадцяткове число може утворюватись з цифр, букв **A, B, C, D, E, F, a, b, c, d, e, f** і закінчується буквою **H** або **h** .

7. Граматика генерує послідовність відношень вигляду

<змінна > <операція відношення> <число>

де **<змінна >** складається з букв і має довільну довжину,

<операція відношення> : < <= > >= <> =

<число> − ціле число без знаку довільної довжини.

Відношення відокремлюються одне від одного комами, кома не може бути після останнього у послідовності відношення.

8. Граматика генерує послідовність десяткових чисел без знаку. Числа відокремлюються одне від одного комами, можуть бути цілі (довільної довжини) і з фіксованою крапкою, що мають такий формат

<ціле> . <ціле> ,

< ціле > складається з цифр (довжина довільна).

9. Граматика генерує десяткові числа з фіксованою та плаваючою крапкою. Формат чисел такий:

[<знак.] <ціле> . <ціле> (для чисел з фіксованою крапкою)

0.< мантиса > E < знак > < порядок > (для чисел з плаваючою крапкою),

< мантиса > складається з цифр, перша з яких не 0, **< знак >** − це + або -

< порядок > − 2 десяткові цифри.

10. Граматика генерує послідовність імен простих та індексованих змінних, що відокремлюються одне від одного комами. Ім'я простої змінної складається з довільної кількості букв, індексованої змінної − з довільної кількості букв, дужки **[,** цілого числа без знаку , дужки] .

11. Граматика генерує індексовані змінні, які мають такий формат:

<ім'я > [**<індексний вираз>**]

<ім'я > складається з букв, **<індексний вираз>** − одне, два або три цілих десяткових числа довільної довжини, відокремлених комами.

12. Граматика генерує послідовність, що складається з цілих десяткових чисел із знаком або без нього та коментарів, що записуються за правилом

{ <Текст коментаря> }

де у **<Текст коментаря>** можуть входити будь-які символи, крім символу }.

13. Граматика генерує вирази вигляду

<ім'я > [**<список>**]

<ім'я > складається з букв, **<список>** − послідовність елементів, відокремлених комою, або один елемент, кожен з яких має вигляд **<ціле> . . <ціле>** , де **<ціле>** − ціле десяткове число довільної довжини без знаку.

14. Граматика генерує вирази вигляду

<ім'я > [**<список>**]

<ім'я > складається з букв, **<список>** − послідовність елементів, відокремлених комою, або один елемент. Кожен елемент − ціле десяткове число довільної довжини без знаку або послідовність букв, перша з яких − **i, j, k, l, m, n .**

15. Граматика генерує послідовність чисел від 1 до 399, записаних у римській системі числення. Між числами може бути довільна кількість пробілів. Символи римської системи та їх зміст

I − 1 , V − 5 , X − 10 , L − 50, C − 100.

Рекомендована література

1.Виленкин Н.Я. Комбинаторика. – М.: Наука, 1969. – 161 с.

2.Капітонова Ю.В., Кривий С.Л., Летичевський О.А. та ін. Основи дискретної математики. – К.: Наукова думка, 2002. – 560 с.

3.Бушмакін В.М., Гануліч В.К., Мохонько А.З. та ін. Комбінаторика – Л.: В-во НУ "ЛП", 2002. – 196 с.

1. О.Оре. Теория графов.

2. Ф.Харари. Теория графов.

3. Белов В.В., Воробьев Е.М., Шаталов В.Е. Теория графов.

4. К.В.Ніколаєва, В.В.Койбічук. Дискретний аналіз. Графи та їх застосування. Суми.УАБС НБУ, 2007.

5. Р.М.Трохимчук. Теорія графів. Київ, 1998.

1. Зыков А.А. Теорія кінцевих графів. - Новосибірськ: Наука, 1969.

2. Харари Ф. Теорія графів. - М.: Світ, 1973.

3. Зыков А.А. Основи теорії графів. - М.: Наука, 1987.

4. Кристофидес Н. Теорія графів. Алгоритмічний підхід. - М.: Світ, 1978.

5. Майника Э. Алгоритми оптимізації на мережах і графах. - М.: Світ, 1981.

6. Ловас Л., Пламмер М. Прикладні задачі теорії графів. Теорія паросочетаний у математику, фізику, хімії. - М.: Світ, 1998.

НАВЧАЛЬНЕ ВИДАННЯ

ДИСКРЕТНА МАТЕМАТИКА

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних занять з курсу “Дискретна математика”
для студентів спеціальностей
6.040301 – “Прикладна математика”, 6.040302 –“Інформатика”

Укладачі Демків Любомир Ігорович
Манзій Олександра Степанівна
Кавалець Ірина Іванівна

Редактор

Комп’ютерне верстання