

ЗМІСТ

ВСТУП

1. ОСНОВНІ ОЗНАЧЕННЯ ТА ВЛАСТИВОСТІ	4
2.1. Каркаси. Алгоритм Крускала	6
2.2. Програмна реалізація алгоритму Крускала	8
2.3. Бектрекінг	9
3. ЗАВДАННЯ	10
4. ВИМОГИ ДО ЛАБОРАТОРНОЇ РОБОТИ	16
5. КОНТРОЛЬНІ ЗАПИТАННЯ.....	17
<i>Додаток А Титульна сторінка звіту до лабораторної роботи</i>	<i>18</i>

Мета роботи: Вивчення основних означень та властивостей дерев, набуття практичних навичок розв'язування завдань, що базуються на деревах, та програмної реалізації алгоритмів обходу дерев, рекурсії, бектрекінгу та Краскала.

ВСТУП

Поняття дерева широко застосовують у багатьох розділах математики та інформатики. Наприклад, дерева використовують як інструмент обчислень, зручний спосіб збереження даних, їх сортування та пошуку.

1. ОСНОВНІ ОЗНАЧЕННЯ ТА ВЛАСТИВОСТІ

Деревом називають зв'язний граф без простих циклів (див. рис. 1). За необхідності у дереві виділяють вершину, яка є його *коренем* (див. рис. 2). В такому випадку приписується напрямок кожному ребру від кореня до кожної вершини графа. Утворений орієнтований граф називають *кореневим деревом*.

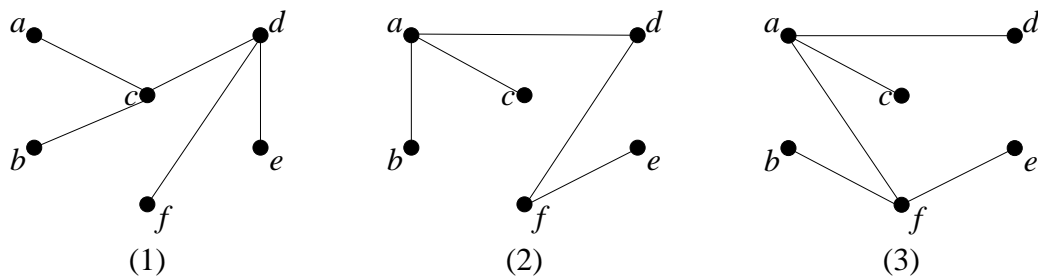


Рис. 1. Деревя

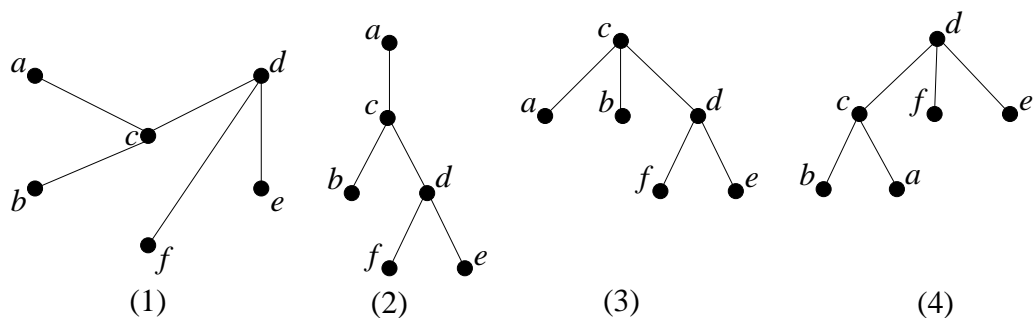


Рис. 2. Побудова корневих дерев та вибір кореня

В залежності від вибору кореня дерева можуть відрізнятися. На рис. 2 показано наслідки вибору кореня: (1) дерево, (2) дерево 1 при виборі кореня a , (3) дерево 1 при виборі кореня c , (4) дерево 1 при виборі кореня d . При поданому зображенні корневих дерев напрямок ребер є зверху вниз.

Якщо в кореновому дереві T є орієнтоване ребро (u, v) , то вершину u називають батьком вершини v , а вершину v сином вершини u . Вершини дерева,

що не мають синів називаються *листяками*. Вершини, що не є коренем та мають синів називають *внутрішніми*. Нехай u – внутрішня вершина кореневого дерева T , тоді всі нащадки вершини u та інцидентні їм ребра утворюють *підграф*.

Детальніше розглянемо дерево 2: a – корінь дерева; c – батько вершин b і d , а вершини b і d – сини вершини c ; c і d – внутрішні вершини; b, f і e – листки; вершини c, b, d, f і e та інцидентні їм ребра утворюють підграф з вершиною c .

Серед представлених на рисунку 2 дерев, дерево (2) є бінарним, оскільки кожна його вершина має не більше 2 синів.

2. РЕКУРСІЯ. ОБХІД ДЕРЕВ

Об'єкт називають рекурсивним, якщо він містить сам себе чи його означено за допомогою самого себе. Рекурсія — потужний засіб у математичних означеннях.

Приклад 1. Рекурсивне означення функції $(n)!$ для невід'ємних цілих чисел має такий вигляд;

- $(0)! = 1$;
- якщо $n > 0$, то $(n)! = n(n - 1)!$;

Очевидно, що важливість рекурсії пов'язана з тим, що вона дає змогу означити нескінченну множину об'єктів за допомогою скінченного висловлювання. Так само нескінченні обчислення можна описати за допомогою скінченної рекурсивної програми, навіть якщо вона не містить явних циклів. Проте найдоцільніше використовувати рекурсивні алгоритми тоді, коли розв'язувану задачу, обчислювану функцію чи оброблювані дані задано за допомогою рекурсії.

Обходом дерева називається перебір всіх його вершин в певному порядку (див. рис. 3):

- обхід у *прямому порядку*, або *зверху вниз*: a, b, c ;
- обхід у *внутрішньому порядку*, або *зліва направо*: b, a, c ;
- обхід у *зворотному порядку*, або *знизу вверх*: b, c, a ;

Для прикладу розглянемо дерево на рисунку 4:

- обхід у *прямому порядку*: a, c, b, d, f, e ;
- обхід у *внутрішньому порядку*: a, b, c, f, d, e ;
- обхід у *зворотному порядку*: b, f, e, d, c, a .

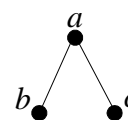


Рис. 3

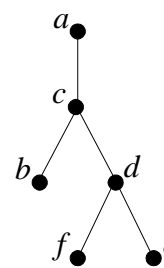


Рис. 4.

Одне з поширених застосувань обходу бінарних дерев зіставлення арифметичним та логічним виразам. В такому випадку арифметичні та логічні операції є коренем та внутрішніми вершинами деревами, а операнди (змінні,

константи) – листками.

Розглянемо прифметичний вираз: $(a + b)/c - (a - c)$. Алгоритм перетворення виразу в дерево є наступним:

- Визначаємо послідовність дій:
(1) $a + b$, (2) $a - c$,
(3) $(a + b)/c$,
(4) $(a + b)/c - (a - c)$.
- Будуємо дерево кожної дії у визначеному порядку (див. рис. 5).

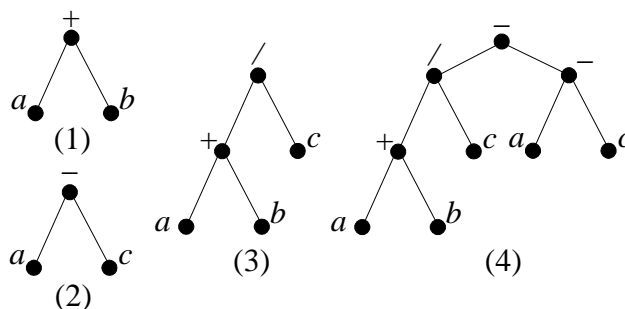


Рис. 5.

При обході побудованого дерева отримаємо три наступні послідовності:

- обхід у прямому порядку – префіксний (польський) запис: $-/+abc-ac$;
- обхід у внутрішньому порядку – інфіксний запис: без дужок – $a+b/c-a-c$, з дужками – $(a+b)/c-(a-c)$;
- обхід у зворотному порядку – постфіксний (зворотній польський) запис: $ab+c/ac--$.

Як помітно з вище поданого прикладу в польському та зворотному польському записях пріоритет операцій визначається за порядком обходу елементів дерева, в порівнянні, інфіксний запис без дужок може відповідати різним деревам.

Для прикладу розглянемо обчислення виразу в польському записі: $+4-*3\ 4/6\ 2$ (див. табл. 1).

Таблиця 1

Крок	Вираз	Дія	Виконання операції
1	$+4-*3\ 4/6\ 2$	$/6\ 2$	$6/2=3$
2	$+4-*3\ 4\ 3$	$*3\ 4$	$3*4=12$
3	$+4-12\ 3$	$-12\ 3$	$12-3=9$
4	$+4\ 9$	$+4\ 9$	$4+9=13$

2.1. Каркаси. Алгоритм Крускала

Каркас (з'єднувальне дерево) – називають підграф просто звязного графа, який являє собою дерево та містить усі вершини цього графа. Взагалі, кістякове дерево складається з деякої підмножини ребер графа, таких, що

рухаючись цими ребрами можна з будь-якої вершини графа потрапити до будь-якої іншої.

Будь-яке каркасне дерево у графі з n вершинами містить рівно $n-1$ ребер. Кількість кістякових дерев у повному графі з n вершинами подається відомою формулою Келі: n^{n-2}

Для того щоб отримати каркас графа G , який має n вершин і m ребер можна використати процедуру вилучення ребер, які належать простим циклам. Потрібно вилучити ребра $Y(G)=m-(n-1)=m-n+1$, де $Y(G)$ – цикломатичне число графа G . Цикломатичне число — це числова характеристика зв'язності графа. $Y(G) \geq 0$ оскільки, якщо граф G має n вершин та k компонентів, то кількість m його ребер задовольняє нерівність $n-k \leq m \leq 1/2(n-k)(n-k+1)$, а цикломатичне число дерева дорівнює 0.

Побудова каркаса

Алгоритм, в основі якого лежить вилучення ребер із простих циклів є неефективний для комп'ютерної реалізації, оскільки для його виконання потрібно ідентифікувати прості цикли, а це складна задача.

З точки зору комп'ютерної реалізації, ефективним алгоритмом побудови каркасів є алгоритм послідовного добору ребер у каркас. Цю дію можна виконувати як і за допомогою пошуку углиб так і за пошуком ушир. Воно складається з усіх пар ребер (u, v) , таких, що алгоритм, переглядаючи вершину u виявляє в її списку суміжності нову, невиявлену раніше вершину v . Кістякове дерево, побудоване обходом графа починаючи з вершини s за алгоритмом Дейкстри, має властивість, що найкоротший шлях у графі з вершини s до будь-якої іншої вершини – це шлях з s до цієї вершини в побудованому дереві.

Теорема. Нехай T каркас графа G , побудований пошуком ушир, починаючи з вершини a . Тоді шлях з a до довільної вершини v в T – найкоротший шлях з a до v в графі G .

Зауваження: Використовуючи для побудови найкоротшого шляху від вершини a алгоритмом Дейкстри (припускаючи, що довжина кожного ребра дорівнює 1), одержуємо те саме дерево, що й за алгоритмом вшир. Але складність першого алгоритму становить $O(n^2)$ або $O(m \log n)$ в залежності від подання графа, а складність другого алгоритму (вшир), $O(m+n)$ де n – кількість вершин, m – кількість ребер у графа G . Алгоритм Дейкстри є більш універсальний, він придатний для всіх додатніх довжин графу, а не тільки 1.

Алгоритм Крускала. Візьмемо зважений зв'язний граф $G=(V, E)$, де V – множина вершин, E – ножина ребер, для кожного з яких задано вагу. Мінімальним каркасним (або кістяковим) дерево називають ациклічну

множину ребер, що поєднують усі вершини графа і чия загальна вага мінімальна.

Алгоритм Крускала слід почати з побудови виродженого лісу, який містить V дерев, кожне з яких складається з однієї вершини. Далі потрібно виконати операції об'єднання двох дерев, для цього використовують

найкращі можливі ребра, поки не утвориться єдине дерево.

Це дерево і буде мінімальним кістяковим деревом.

Приклад: на Рис.6 зображено граф, ребра якого пронумеровано за зростанням ваг. Мінімальний каркас виділено потовщеними лініями. Процес відбору наведено в таблиці 2. Мінімальний каркас утворюють ребра. $e_1, e_2, e_3, e_5, e_8, e_{11}$.

Таблиця 2. Побудова каркасу

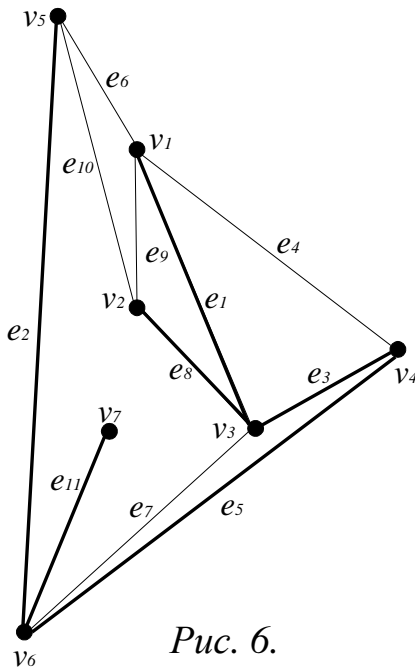


Рис. 6.

Ребро	Розбиття множин вершини	Відбір ребра у мінімальний каркас
$e_1 = (V_1, V_3)$	$((V_1), (V_2), (V_3), (V_4), (V_5), (V_6), (V_7))$	e_1
$e_2 = (V_5, V_6)$	$((V_1, V_3), (V_2), (V_4), (V_5), (V_6), (V_7))$	e_2
$e_3 = (V_3, V_4)$	$((V_1, V_3), (V_2), (V_4), (V_5, V_6), (V_7))$	e_3
$e_4 = (V_1, V_4)$	$((V_1, V_3, V_4), (V_2), (V_5, V_6), (V_7))$	—
$e_5 = (V_4, V_6)$	$((V_1, V_3, V_4), (V_2), (V_5, V_6), (V_7))$	e_5
$e_6 = (V_1, V_5)$	$((V_1, V_3, V_4, V_5, V_6), (V_2), (V_7))$	—
$e_7 = (V_3, V_6)$	$((V_1, V_3, V_4, V_5, V_6), (V_2), (V_7))$	—
$e_8 = (V_2, V_3)$	$((V_1, V_2, V_3, V_4, V_5, V_6), (V_7))$	e_8
$e_9 = (V_1, V_2)$	$((V_1, V_2, V_3, V_4, V_5, V_6), (V_7))$	—
$e_{10} = (V_2, V_5)$	$((V_1, V_2, V_3, V_4, V_5, V_6, V_7))$	—
$e_{11} = (V_6, V_7)$	$((V_1, V_2, V_3, V_4, V_5, V_6, V_7))$	e_{11}

2.2. Програмна реалізація алгоритму Крускала

```
int cn; //число вершин
vector< vector<int> > ady; //матриця суміжності
// Повертає матрицю суміжності мінімального дерева
```

```

vector< vector<int> > Grafo :: kruskal(){
    vector< vector<int> > adyacencia = this->ady;
    vector< vector<int> > arbol(cn);
    vector<int> pertenece(cn); // позначає, чи належить дереву вершина

    for(int i = 0; i < cn; i++){
        arbol[i] = vector<int> (cn, INF);
        pertenece[i] = i;
    }
    int nodoA;
    int nodoB;
    int arcos = 1;
    while(arcos < cn){
        // Знайти найлегше ребро, що не утворює циклів і зберегти вершини і
        // вагу.
        int min = INF;
        for(int i = 0; i < cn; i++)
            for(int j = 0; j < cn; j++)
                if(min > adyacencia[i][j] && pertenece[i] != pertenece[j]){
                    min = adyacencia[i][j];
                    nodoA = i;
                    nodoB = j;
                }

        // Якщо вершини не належать до одного дерева, додаємо ребро між ними
        // до дерева.
        if(pertenece[nodoA] != pertenece[nodoB]){
            arbol[nodoA][nodoB] = min;
            arbol[nodoB][nodoA] = min;

            // Усі вершини дерева nodoB зараз належать до дерева nodoA.
            int temp = pertenece[nodoB];
            pertenece[nodoB] = pertenece[nodoA];
            for(int k = 0; k < cn; k++)
                if(pertenece[k] == temp)
                    pertenece[k] = pertenece[nodoA];

            arcos++;
        }
    }
    return arbol;}

```

2.3. Бектрекінг

Пошук з повертанням (англ. backtracking), також пошук з поверненням – загальний алгоритм для знаходження всіх (або деяких) розв'язків деякої обчислювальної задачі, який поступово будує кандидатів на розв'язок, і відкидає кожного неповного кандидата с («повертається») як тільки визначає, що не може бути доповненим до вірного розв'язку.

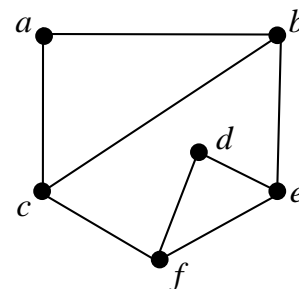


Рис. 7

Проілюструємо застосування алгоритму на конкретному прикладі:

Побудова гамільтонових циклів. Дано граф G , що подано на рисунку 7. Починаємо з довільної вершини. У поданому прикладі, зображеному на рисунку 8 – вершини a . Будуємо шлях без повторення вершин доки це можливо. Якщо вдалось пройти всі вершини, перевіряємо чи існує ребро, що зв'язує першу та останню вершини. Якщо описаний процес в певний момент неможливо продовжити, то повертаємось на крок назад (див. рис. 8).

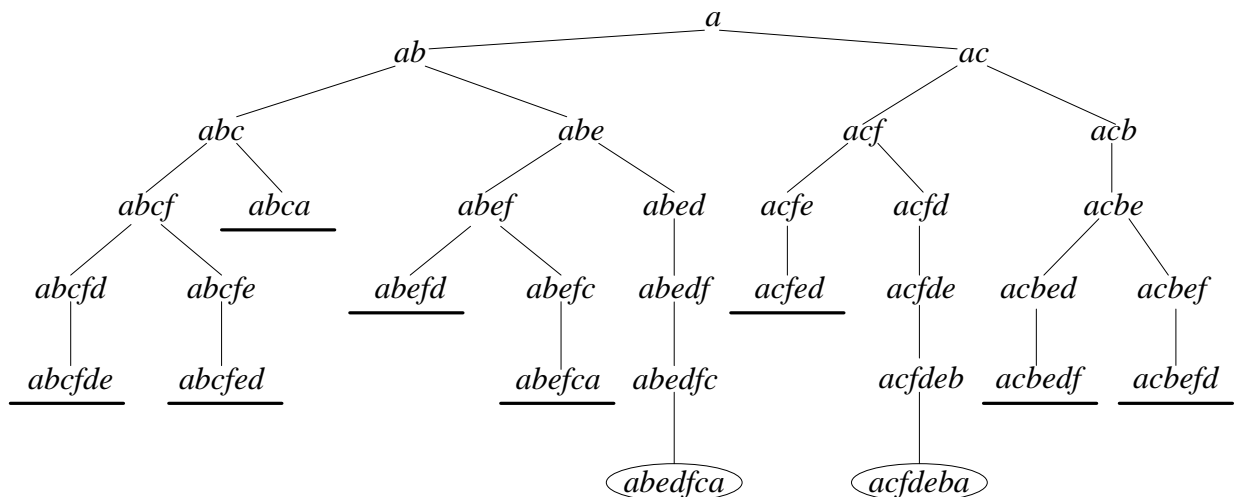


Рис. 8. Бектрекінг

В результаті бектрекінгу отримуємо всі властиві даному графу гамільтонові цикли (див. рис. 9): $P_1 = a, b, e, d, f, c, a$; $P_2 = a, c, f, d, e, b, a$.

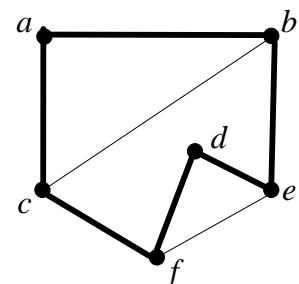


Рис. 9

3. ЗАВДАННЯ

Порядок виконання роботи: Скласти комп'ютерні програми із зазначеними вхідними даними та результатами для завдань 1-3.

Завдання 1: Побудувати кореневі дерева, які відповідають виразам у табл. 1. Записати ці вирази в інфіксній, префіксній та постфіксній формах.

Таблиця 1

Вирази

Варіант	Вираз
1.	$(n+3)-2*n-(m*n/5)$
2.	$(n-1)-m-n*m/5$
3.	$n+(n+m)/4-n*n$

Варіант	Вираз
4.	$(n+m)-4*m-n*m/5$
5.	$n+(n/(4+m))-n*8$
6.	$n-n/4-n*n$

7.	$(n-5)+3*m-n/m$
8.	$n-n/4-(n+1)*n$
9.	$n+n/4-n*n*n$
10.	$(n+3)-2*n-(m*n)/5$
11.	$n+n/4-n*n/(2+m)$
12.	$n+m/6-n/2*m$
13.	$(9+m)/(n-5)-7/n*m$
14.	$4/(2+m)+(6*n)/5$
15.	$(n*m)-1/(6+n)+9$
16.	$(n+8)-2*m-n/4*m$
17.	$n+m/(5+n)-n*3$
18.	$(n-3)-(m-n/5)*$

19.	$(n+m)/7-n*(m+3)$
20.	$(m-5)+3*n-m/(n-1)$
21.	$(n-5/m)+n-(n+1)/m$
22.	$n/(m+8)+4/m-n*m$
23.	$n/(m+1)-(n-5)+7*m$
24.	$2*m+n/6-n/(m-3)$
25.	$(n-m)+5*m-(n+m)/9$
26.	$n-n/8+n/(m+2)$
27.	$m+n/(4-n)-(n+1)*n$
28.	$n/4-(n+1)/n+8$
29.	$5*m/(n+m)+7(n-m)$
30.	$n-(n+2)+5/(m+8)$

Завдання 2: Обчислити значення виразу з табл. 2, записаного у зворотній польській нотації $34 + 42 - 5 * +$. Побудувати кореневі дерева для даного виразу.

Таблиця 2

Вирази у зворотній польській нотації

Варіант	Вираз
1.	$4\ 2\ 1 + * 2 - 3 *$
2.	$3\ 4 + 4\ 2 - 5 * +$
3.	$2\ 7 - 5\ 3 + 8 * -$
4.	$3\ 5 + 4\ 3 - 5 * +$
5.	$5\ 2\ 1 + * 2 - 4 *$
6.	$8\ 4\ 1 + - 2\ 2 + *$
7.	$9\ 3/5 + 7\ 2 - *$
8.	$3\ 5 + 4\ 1 - 2 * +$
9.	$3\ 6 + 3\ 1 - 3 * +$
10.	$8\ 3 - 6\ 7 + 1\ 8 * + *$
11.	$6\ 2\ 3 + * 3 + 2 *$
12.	$5\ 2\ 1 - - 2\ 5\ 6 + + *$
13.	$8\ 2\ 3\ 4 * 10\ 5 / + * +$
14.	$15\ 7\ 1\ 1 + - / 3 \times 2\ 1\ 1 + + -$
15.	$1\ 3\ 2 + 4\ 1 * * +$

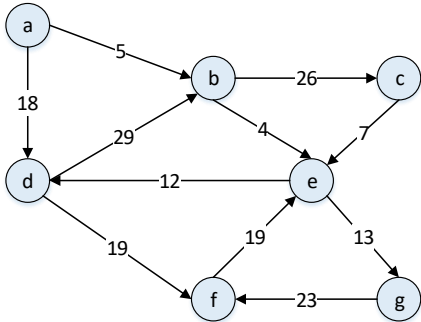
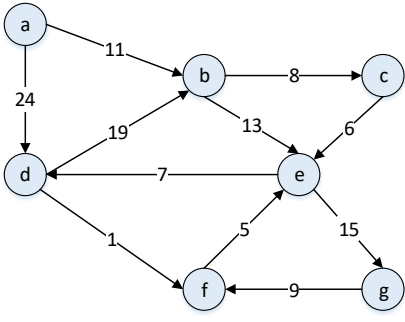
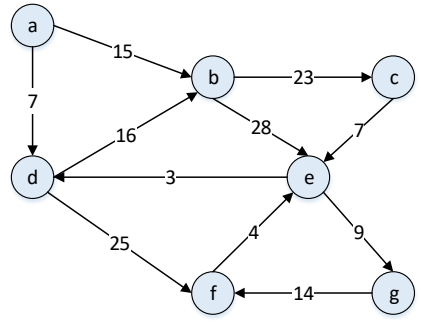
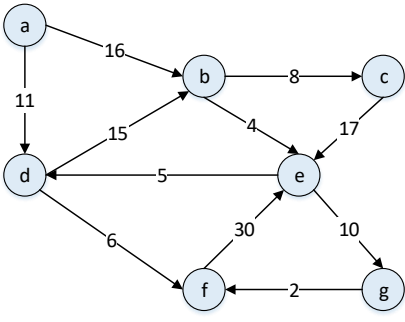
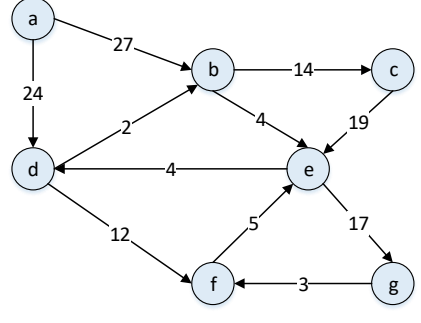
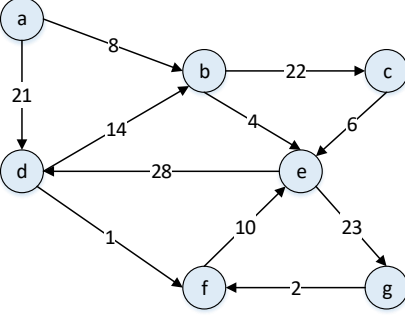
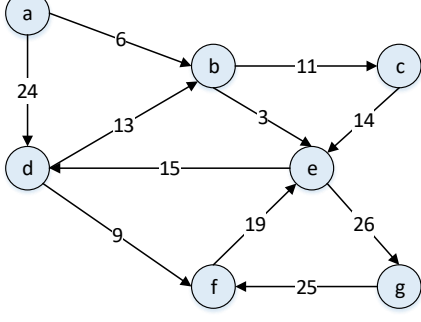
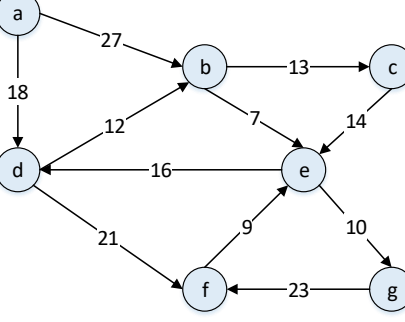
Варіант	Вираз
16.	$7\ 8 + 4\ 7 - 9 * -$
17.	$3\ 2\ 1 - + 7\ 6 + *$
18.	$6\ 4\ 2 + - 9\ 9 + +$
19.	$6\ 9 - 5\ 3\ 7 - * +$
20.	$5\ 9 + 8\ 7\ 3 + - *$
21.	$7\ 2\ 1 + - 2\ 3 \uparrow *$
22.	$8\ 4\ 1 - + 4\ 5 * +$
23.	$8\ 2 / 9 + 5\ 8 - *$
24.	$4\ 1 / 3\ 4 + - 6\ 5 - *$
25.	$3\ 4 + 7\ 5 - * 6\ 3 / \uparrow$
26.	$3\ 7\ 5 - + 4 * 6\ 2 / +$
27.	$5\ 2 - 4\ 2 - 4 \uparrow +$
28.	$8\ 2\ 5 * + 1\ 3\ 2 * + 4 - /$
29.	$10\ 5 / 3\ 4 * + 7\ 2 + -$
30.	$16\ 8 / 4\ 2 * + 3\ 5 + 7\ 6 - + -$

Завдання 3: Використовуючи бектрекінг, розфарбувати в n кольорів графі із лабораторної роботи № 4.

Графи

Варіант	m	1	1	1	1	1	2	2	2	2	3	3	3	2	2	5
	n	2	3	4	5	6	3	4	5	6	4	5	6	5	6	6
1	Rmn	1	2	1	1	1	0	0	0	1	1	0	0	1	1	1
2	Rmn	0	0	1	1	1	0	1	0	1	0	1	0	0	0	2
3	Rmn	1	0	0	1	0	0	1	1	1	0	0	0	1	0	1
4	Rmn	1	0	1	0	0	1	1	0	2	0	0	1	0	2	0
5	Rmn	2	1	1	0	0	1	1	0	1	0	0	0	1	1	0
6	Rmn	1	0	0	1	0	1	0	0	0	1	0	0	1	2	2
7	Rmn	1	1	1	0	0	0	2	0	2	0	0	1	0	1	0
8	Rmn	0	2	0	2	0	1	0	0	0	0	1	0	1	1	1
9	Rmn	1	2	1	0	0	0	0	0	1	1	1	0	1	0	0
10	Rmn	1	1	1	1	0	0	0	0	1	0	1	0	1	1	1
11	Rmn	1	2	0	0	1	1	1	0	0	2	1	0	1	0	0
12	Rmn	0	0	0	1	1	1	0	0	2	0	0	1	1	0	2
13	Rmn	0	1	0	1	0	0	1	2	0	1	0	0	0	1	1
14	Rmn	1	1	0	0	0	1	0	1	1	0	1	1	1	0	1
15	Rmn	1	0	2	0	2	0	0	0	1	0	0	0	1	0	1
16	Rmn	1	1	0	0	0	1	1	0	1	0	0	0	1	1	2
17	Rmn	1	0	0	0	0	1	1	2	0	0	0	1	1	0	1
18	Rmn	2	1	0	1	0	1	1	0	0	0	1	0	1	1	1
19	Rmn	1	1	0	0	0	2	1	0	2	1	0	0	1	1	1
20	Rmn	2	0	1	1	0	0	1	0	1	0	0	0	0	2	1
21	Rmn	1	1	0	0	0	1	1	0	1	1	0	1	1	1	0
22	Rmn	2	0	0	0	0	1	1	1	1	0	1	0	1	0	0
23	Rmn	1	1	1	1	0	1	1	0	1	0	0	0	1	1	0
24	Rmn	0	0	1	0	1	2	1	0	1	1	0	1	0	1	0
25	Rmn	2	0	1	0	1	1	0	0	1	0	0	1	1	0	1
26	Rmn	0	2	1	0	1	0	0	1	1	0	0	0	0	1	1
27	Rmn	0	1	0	1	0	1	1	0	0	1	1	0	1	1	1
28	Rmn	2	0	1	1	0	1	1	0	0	0	1	0	1	1	1
29	Rmn	1	0	1	0	2	1	1	1	0	0	0	1	1	1	0
30	Rmn	1	1	0	0	0	2	0	1	2	1	0	0	1	0	0

Завдання 4: За алгоритмом Крускала побудувати мінімальний каркас для наступних графів:

№	Орієнтований граф	№	Орієнтований граф
1.		2.	
3.		4.	
5.		6.	
7.		8.	

9.		10.	
11.		12.	
13.		14.	
15.		16.	

17.		18.	
19.		20.	
21.		22.	
23.		24.	

25.	<pre> graph TD a((a)) -- 19 --> b((b)) a -- 2 --> d((d)) b -- 21 --> c((c)) b -- 23 --> e((e)) d -- 23 --> b d -- 29 --> e d -- 3 --> f((f)) e -- 17 --> c e -- 15 --> f e -- 16 --> g((g)) f -- 8 --> g </pre>	26.	<pre> graph TD a((a)) -- 2 --> b((b)) a -- 12 --> d((d)) b -- 25 --> c((c)) b -- 28 --> e((e)) d -- 22 --> b d -- 11 --> e d -- 10 --> f((f)) e -- 4 --> c e -- 7 --> f e -- 8 --> g((g)) f -- 15 --> g </pre>
27.	<pre> graph TD a((a)) -- 16 --> b((b)) a -- 17 --> d((d)) b -- 12 --> c((c)) b -- 2 --> e((e)) d -- 8 --> b d -- 9 --> e d -- 21 --> f((f)) e -- 14 --> c e -- 5 --> f e -- 7 --> g((g)) f -- 2 --> g </pre>	28.	<pre> graph TD a((a)) -- 5 --> b((b)) a -- 21 --> d((d)) b -- 3 --> c((c)) b -- 1 --> e((e)) d -- 24 --> b d -- 22 --> e d -- 5 --> f((f)) e -- 17 --> c e -- 1 --> f e -- 16 --> g((g)) f -- 13 --> g </pre>
29.	<pre> graph TD a((a)) -- 4 --> b((b)) a -- 10 --> d((d)) b -- 3 --> c((c)) b -- 30 --> e((e)) d -- 6 --> b d -- 13 --> e d -- 21 --> f((f)) e -- 12 --> c e -- 2 --> f e -- 24 --> g((g)) f -- 20 --> g </pre>	30.	<pre> graph TD a((a)) -- 16 --> b((b)) a -- 4 --> d((d)) b -- 19 --> c((c)) b -- 8 --> e((e)) d -- 12 --> b d -- 28 --> e d -- 6 --> f((f)) e -- 25 --> c e -- 30 --> f e -- 5 --> g((g)) f -- 9 --> g </pre>

4. ВИМОГИ ДО ЛАБОРАТОРНОЇ РОБОТИ

1. Кожен студент отримує набір завдань відповідно до свого порядкового номеру у списку групи або відповідно до номеру залікової книжки.
2. Звіт про виконання роботи оформляються у вигляді завдань та розв'язку до них.
3. Звіт акуратно оформляється на аркушах А4 та скріпляються скріпкою.
4. Звіт про виконання лабораторної роботи необхідно захистити у строго визначені терміни.

5. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Основні означення та властивості.
2. Обхід дерев, рекурсія.
3. Префіксна та постфіксна форми записів.
4. Побудова бінарного дерева пошуку.
5. Бектрекінг.

6. ЗМІСТ ЗВІТУ ПО РОБОТІ

1. Назва роботи.
2. Мета роботи.

Теоретична частина (основні означення та властивості, обхід дерев, рекурсія, префіксна та постфіксна форми записів, побудова бінарного дерева пошуку, бектрекінг).

3. Опис виконаної роботи та отриманих результатів (запрограмувати завдання 1-3):

- завдання;
 - текст програми;
 - результати виконання програми;
4. Висновки.

Додаток А
ТИТУЛЬНА СТОРІНКА ЗВІТУ ДО ЛАБОРАТОРНОЇ РОБОТИ

Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра інформаційних
систем та мереж

ДИСКРЕТНА МАТЕМАТИКА

Звіт

до лабораторної роботи №__

(назва лабораторної роботи)

Виконав:

студент гр. _____

(назва групи)

(прізвище та ініціали студента)

Прийняв:

(прізвище та ініціали викладача)

Львів – 20__