Лабораторна робота №1. Алгоритм Дейкстри

Мета і задачі

Навчитися реалізовувати алгоритм Дейкстри для пошуку найкоротшого шляху від однієї вершини графа до інших.

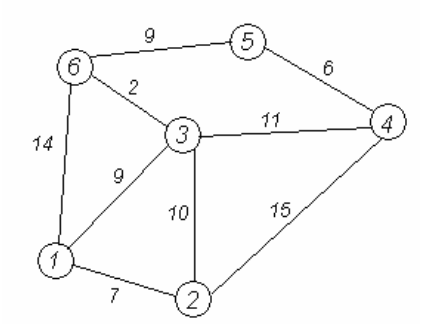
Теоретичні відомості і методичні вказівки

Алгоритм Дейкстри — алгоритм на графах, відкритий Дейкстрою. Знаходить найкоротший шлях від однієї вершини графа до всіх інших вершин. Класичний алгоритм Дейкстри працює тільки для графів без циклів від'ємної довжини.

Наглядне використання алгоритму:

1. Дана мережа автомобільних доріг, що з'єднують міста Львівської області. Знайти найкоротшу відстань від Львова до кожного міста області, якщо рухатись можна тільки по дорогах.
2. Дана карта велосипедних доріжок Латвії та Білорусі. Знайти мінімальну відстань, яку треба проїхати, щоб дістатися від Риги до Бобруйська.
3. Є план міста з нанесеними на нього місцями розміщення пожежних частин. Знайти найближчу до кожного дому пожежну станцію.

Види задання графу:



0	7	9	0	0	14
7	0	10	17	0	0
9	10	0	11	0	2
0	17	11	0	6	0
0	0	0	6	0	9
14	0	2	0	9	0

Графічно Матрично

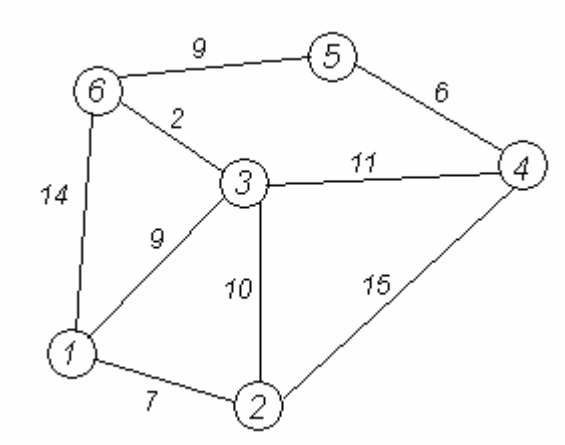
Наглядне використання алгоритму Дейкстри:

Дано неорієнтований зв'язний граф **G(V, U)**. Знайти відстань від вершини **a** до всіх інших вершин **V**.

Зберігатимемо поточну мінімальну відстань до всіх вершин **V** (від даної вершини **a**) і на кожному кроці алгоритму намагатимемося зменшити цю відстань. Спочатку встановимо відстані до всіх вершин рівними нескінченості, а до вершини **a** — нулю.

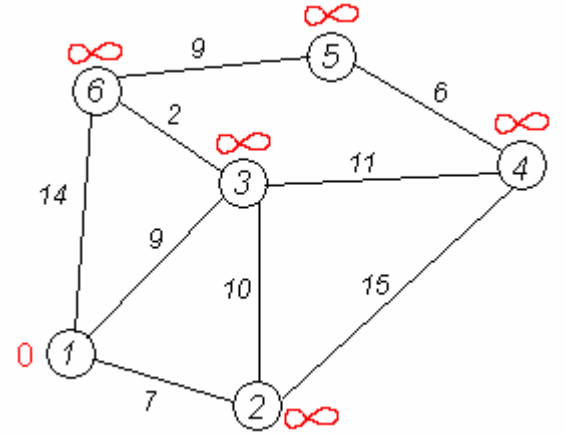
Хай потрібно знайти відстані від 1-ої вершини до всіх інших. Кружечками позначені вершини, лініями — шляхи між ними («дуги»). Над дугами

позначена їх «ціна» — довжина шляху. Надписом над кружечком позначена поточна найкоротша відстань до вершини.



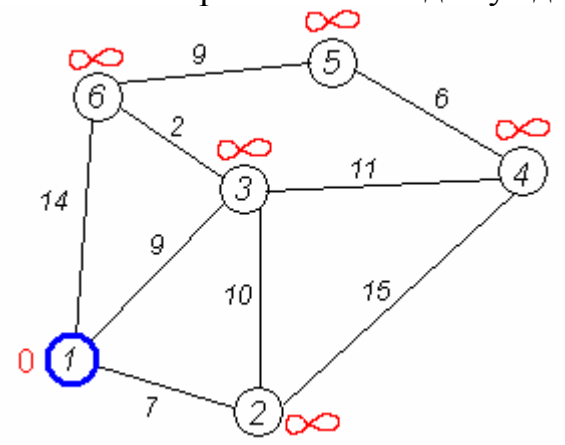
Крок 1

Ініціалізація. Відстань до всіх вершин графа **V** = ∞. Відстань до **a** =**0**. Жодна вершина графа ще не опрацьована.



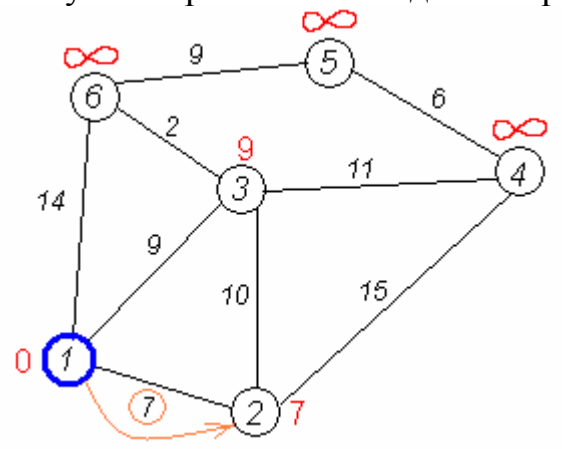
Крок 2

Находимо таку вершину (із ще не оброблених), поточна найкоротша відстань до якої мінімальна. В нашому випадку це вершина 1. Обходимо всіх її сусідів і, якщо шлях в сусідню вершину через 1 менший за поточний мінімальний шлях в цю сусідню вершину, то запам'ятовуємо цей новий, коротший шлях як поточний найкоротший шлях до сусіда.



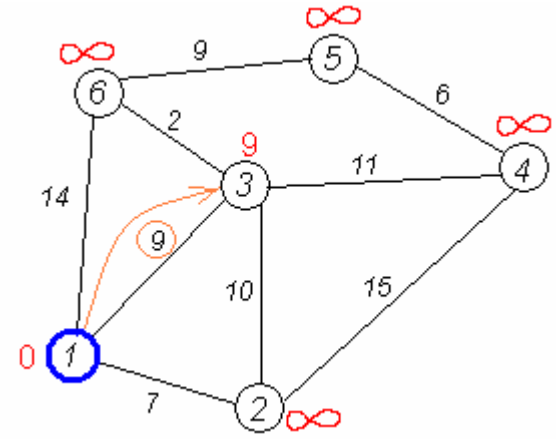
Крок 3

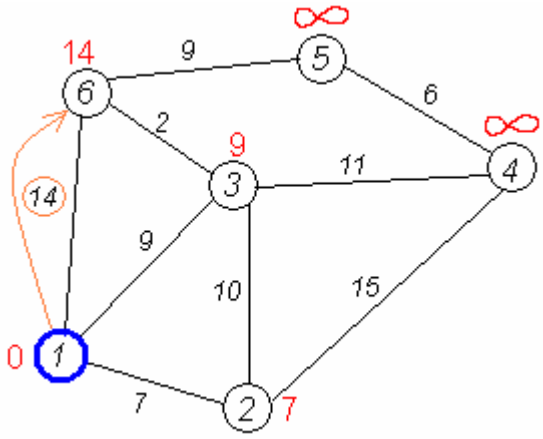
Перший по порядку сусід 1-ї вершини — 2-а вершина. Шлях до неї через 1-у вершину дорівнює найкоротшій відстані до 1-ї вершини + довжина дуги між 1-ю та 2-ю вершиною, тобто 0 + 7 = 7. Це менше поточного найкоротшого шляху до 2-ї вершини, тому найкоротший шлях до 2-ї вершини дорівнює 7.



Кроки 4, 5

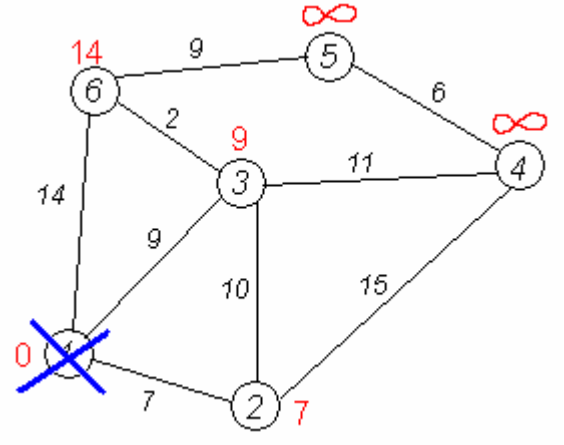
Аналогічну операцію проробляємо з двома іншими сусідами 1-ї вершини — 3-ю та 6-ю.





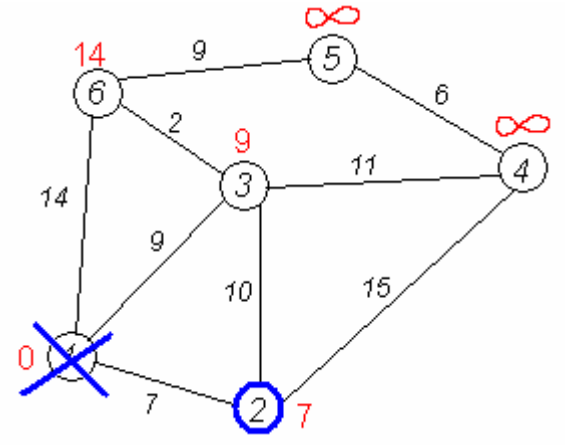
Крок 6

Всі сусіди вершини 1 перевірені. Поточна мінімальна відстань до вершини 1 вважається остаточною і обговоренню не підлягає (те, що це дійсно так, вперше довів Дейкстра). Тому викреслимо її з графа, щоб відмітити цей факт.



Крок 7

Практично відбувається повернення до кроку 2. Знову знаходимо «найближчу» необроблену (невикреслену) вершину. Це вершина 2 з поточною найкоротшою відстанню до неї = 7. І знову намагаємося зменшити відстань до всіх сусідів 2-ої вершини, намагаючись пройти в них через 2-у. Сусідами 2-ої вершини є 1, 3, 4.

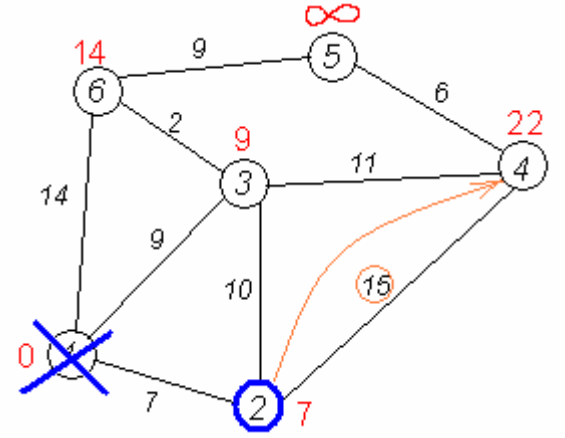


Крок 8

Перший (по порядку) сусід вершини № 2 — 1-а вершина. Але вона вже оброблена (або викреслена — див. крок 6). Тому з 1-ою вершиною нічого не робимо.

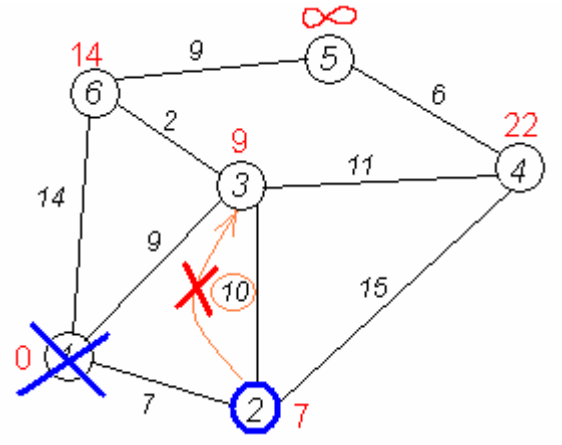
Крок 8 (з іншими вхідними даними)

Інший сусід вершини 2 — вершина 4. Якщо йти в неї через 2-у, то шлях буде = найкоротша відстань до 2-ої + відстань між 2-ою і 4-ою вершинами = 7 + 15 = 22. Оскільки 22 < ∞, встановлюємо відстань до вершини № 4 рівним 22.



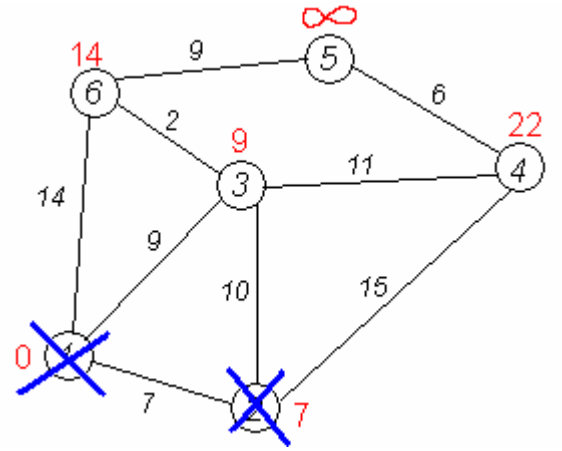
Крок 9

Ще один сусід вершини 2 — вершина 3. Якщо йти в неї через 2-у, то шлях буде = 7 + 10 = 17. Але 17 більше за відстань, що вже запам'ятали раніше до вершини № 3 і дорівнює 9, тому поточну відстань до 3-ої вершини не міняємо.



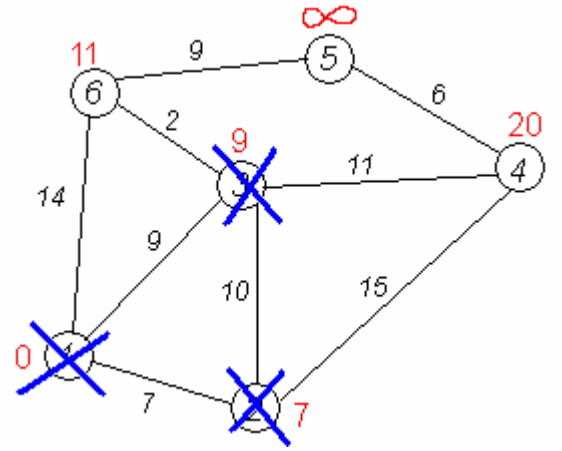
Крок 10

Всі сусіди вершини 2 переглянуті, заморожуємо відстань до неї і викреслюємо її з графа.



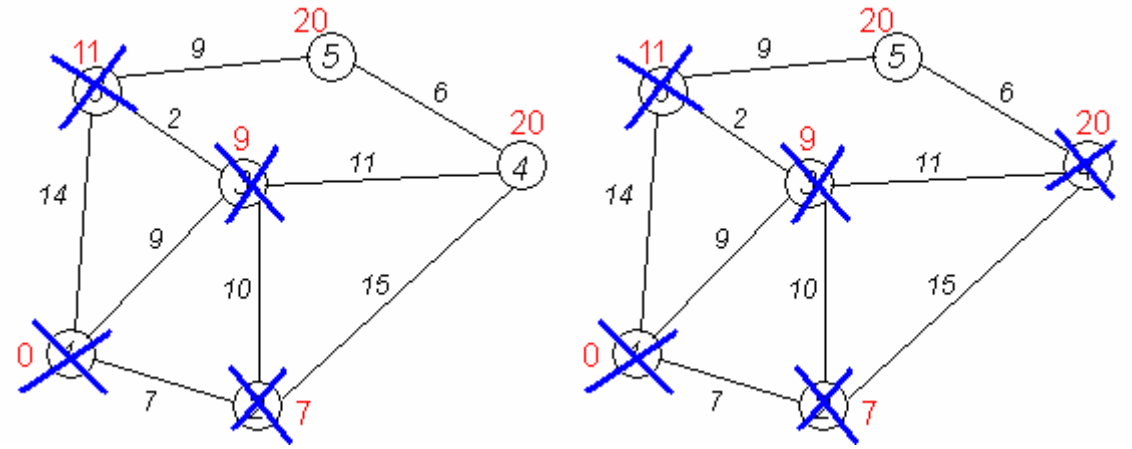
Кроки 11 — 15

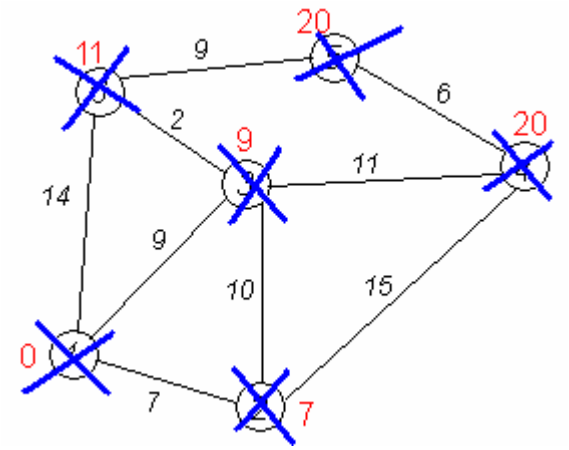
По вже «відпрацьованій» схемі повторюємо кроки 2 — 6. Тепер «найближчою» виявляється вершина № 3. Після її «обробки» отримаємо такі результати:



Наступні кроки

Проробляємо те саме з вершинами, що залишилися (№ по порядку: 6, 4 і 5).





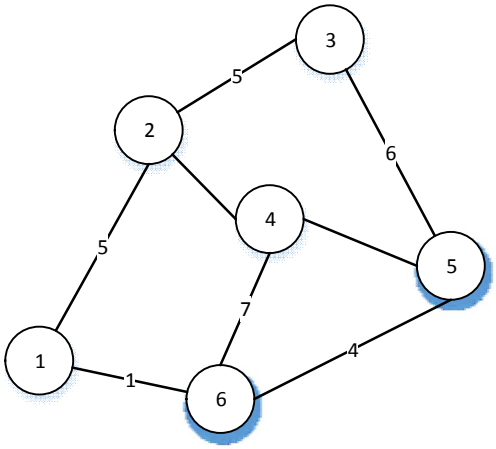
Завершення виконання алгоритму

Алгоритм закінчує роботу, коли викреслені всі вершини. Результат його роботи видно на останньому малюнку: найкоротший шлях від 1-ої вершини до 2-ої становить 7, до 3-ої — 9, до 4-ої — 20, до 5-ої — 20, до 6-ої — 11 умовних одиниць.

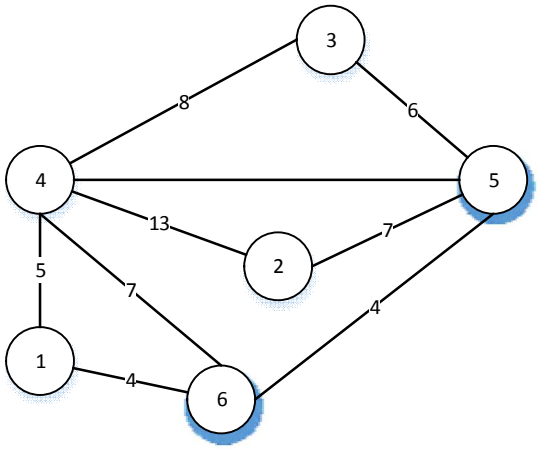
Порядок виконання і звітування

1. Написати програму, яка реалізує алгоритм Дейкстри
2. Відкомпілювати та відлагодити програму.
3. Протестувати роботу програми використовуючи різні вхідні дані
4. Відповісти на контрольні запитання.
5. Зробити висновки.
6. Звіт по лабораторній роботі має складатися з титульної сторінки, мети , лістингів програм, висновків по роботі.

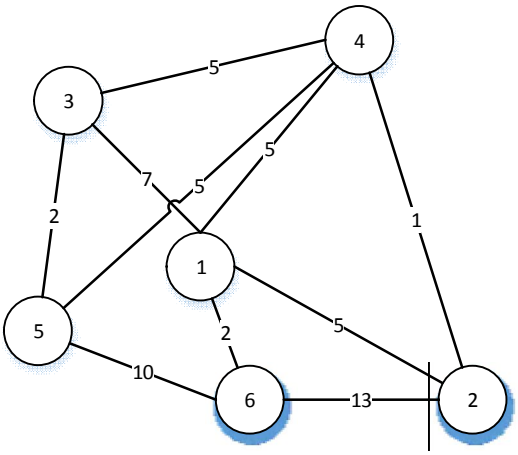
Варіанти завдань


3
5
2
6
4
5
5
7
1
1
6
4

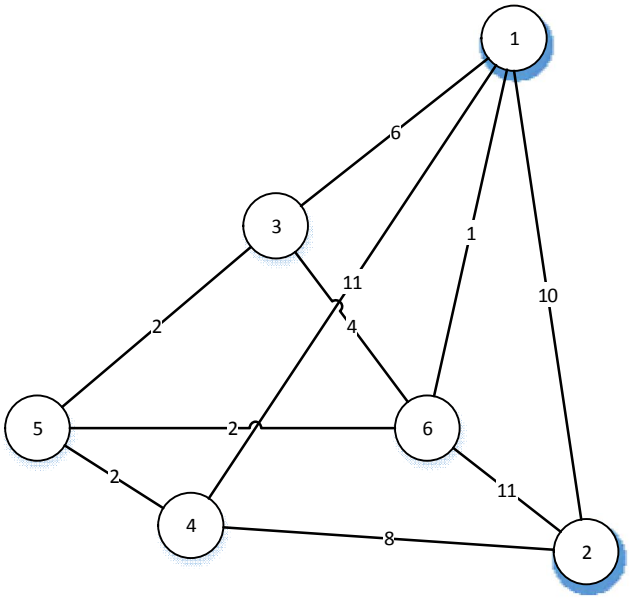
1


3
8
6
4
5
13
2
7
5
7
4
1
4
6

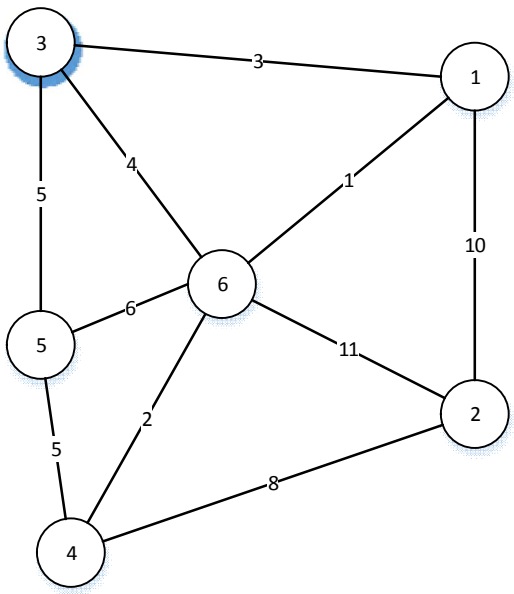
2


4
5
3
7
5
5
2
1
1
5
2
5
10
6
13
2

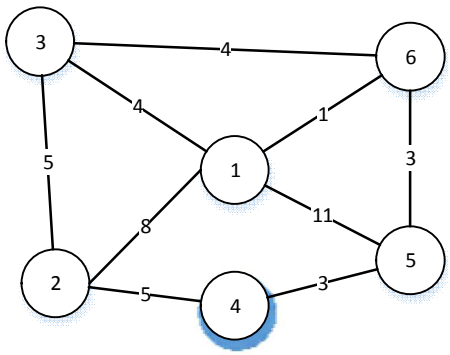
3


1
6
3
1
11
10
2
4
5
2
6
2
11
4
8
2

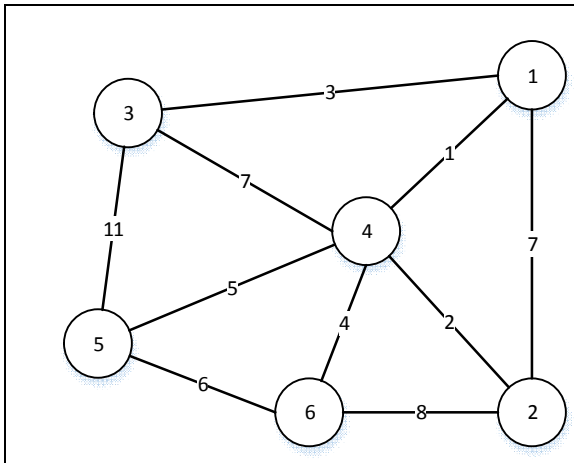
4


3
3
1
4
1
5
10
6
6
11
5
2
5
2
8
4

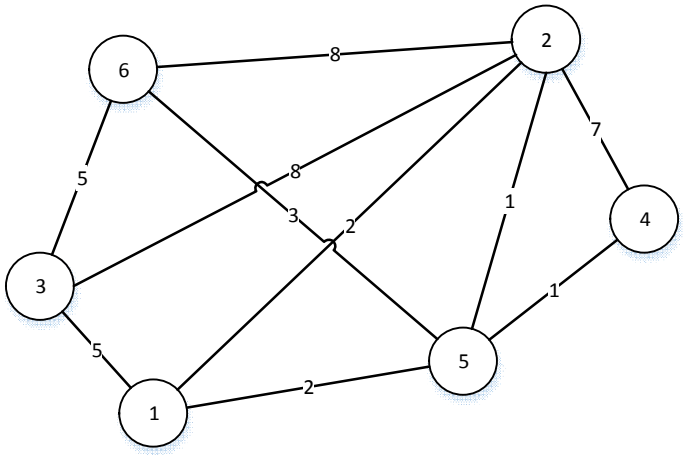
5


3
4
6
4
1
5
1
3
11
8
2
5
4
3
5

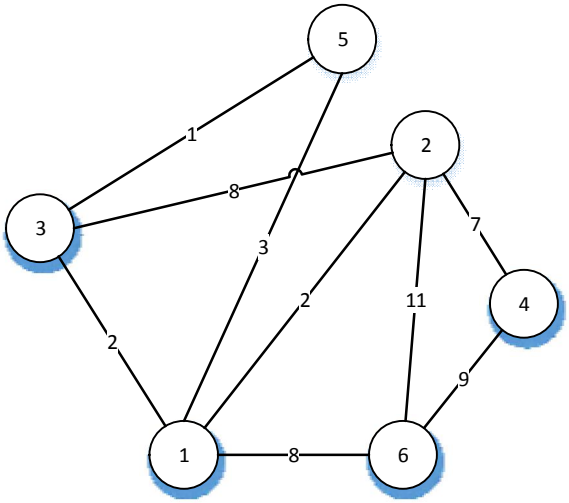
6



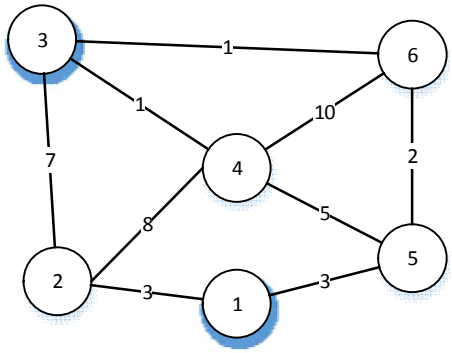
7



8



9



10

Підсумок

Після виконання лабораторної роботи студент повинен вміти реалізовувати алгоритм Дейкстри для пошуку найкоротших шляхів графа.

Контрольні питання

1. Що таке граф?
2. Які є види графів ?
3. Як можуть задаватися графи?
4. Що таке вершина у графі?
5. Що таке ребро у графі?
6. Які алгоритми для пошуку мінімальної відстані у графі вам відомі?

Контрольні вправи

1. Напишіть функцію яка буде графічно відображати граф.
2. Напишіть функцію для виведення усіх пройдених вершин графа, по закінченню алгоритму.

Джерела інформації

1. https://www.youtube.com/watch?v=tyQSgTytc4s
2. http://uk.wikipedia.org/wiki/Алгоритм_Дейкстри
3. http://habrahabr.ru/post/111361/

Лабораторна робота №2. Алгоритм Прима/Крускала

Мета і задачі

Розробити програму на мові Java/C/C++ , яка реалізує алгоритм Прима/ Крускала.

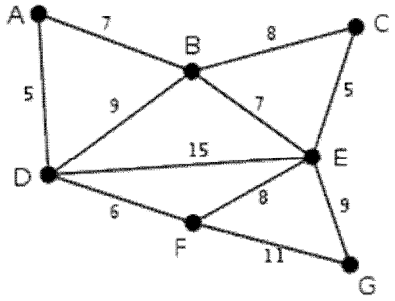
Теоретичні відомості та методичні вказівки

Алгоритм Прима - алгоритм побудови мінімального кістякового дерева зваженого зв'язного неорієнтованого графа. Це жадібний алгоритм.

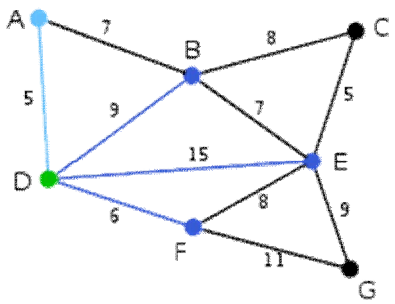
Побудова починається з дерева, що включає в себе одну (довільну) вершину. Протягом роботи алгоритму дерево розростається, поки не охопить всі вершини вихідного графа. На кожному кроці алгоритму до поточного дереву приєднується найлегше з ребер, що з'єднують вершину з побудованого дерева і вершину, що не належить дереву.

Приклад роботи алгоритму:

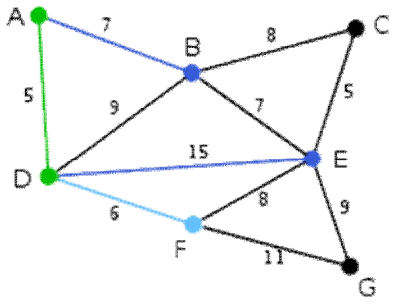
1.Вихідний зважений граф. Числа біля ребер показують їх ваги, які можна розглядати як відстані між вершинами.



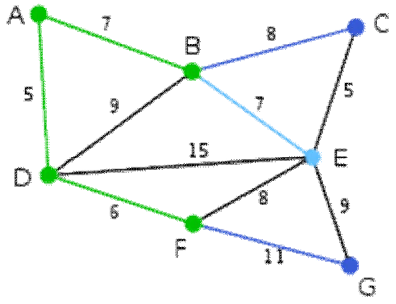
2.В якості початкової довільно вибирається вершина D. Кожна з вершин A, B, E і F з'єднана з D єдиним ребром. Вершина A - найближча до D, і вибирається як друга вершина разом з ребром AD.



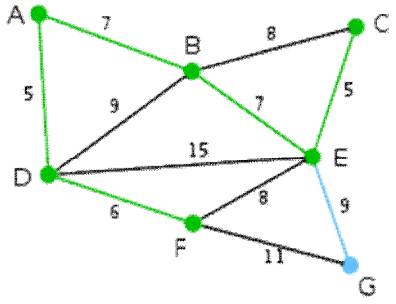
3.Наступна вершина - найближча до будь-якої з обраних вершин D або A. B віддалена від D на 9 і від A - на 7. Відстань до E дорівнює 15, а до F - 6. F є найближчою вершиною, тому вона включається в дерево F разом з ребром DF.



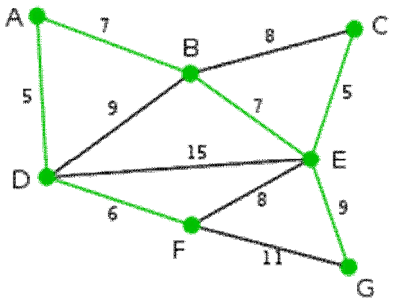
4.Аналогічними кроками приходимо до такого дерева. У цьому випадку є можливість вибрати або C, або E, або G. C віддалена від B на 8, E віддалена від B на 7, а G віддалена від F на 11. E - найближча вершина, тому вибирається E і ребро BE.



5.Єдина вершина, що залишилася - G. Відстань від F до неї одно 11, від E - 9. E ближче, тому вибирається вершина G і ребро EG.



6.Вибрані всі вершини, мінімальне кістякове дерево побудовано



Наглядне використання алгоритму:

Дана карта шляхів між населеними пунктами Вінницької області. Потрібно побудувати схему доріг між цими пунктами, при умові мінімальної затрати ресурсів, які будуть закуплятись для будування цих доріг (асфальту).

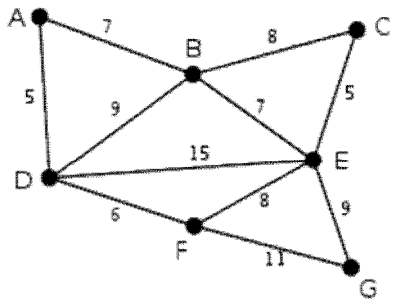
Дана мапа стовпів села міського типу Стрижавка. Для постачання інтернету необхідно провести кабель до кожної домівки, використавши для цього мінімальну довжину кабелю.

Алгоритм Крускала — алгоритм побудови мінімального кістякового дерева зваженого неорієнтовного графа. Алгоритм було вперше описаний Джозефом Крускалом у1956 році.

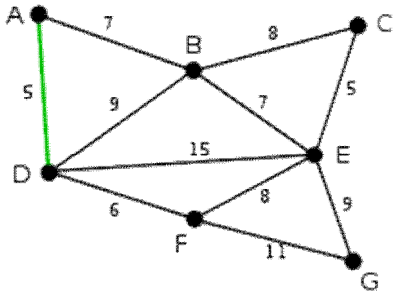
Починається з побудови виродженого лісу, що містить V дерев, кожне з яких складається з однієї вершини. Далі виконуються операції об'єднання двох дерев, для чого використовуються найкоротші можливі ребра, поки не утвориться єдине дерево. Це дерево і буде мінімальним кістяковим деревом.

Приклад роботи алгоритму:

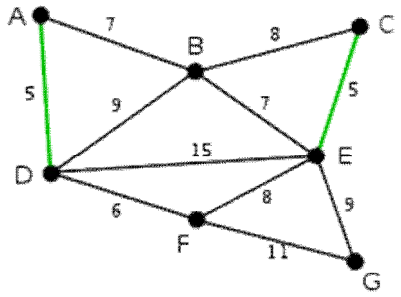
1.Початковий граф. Цифри над ребрами позначають їх вагу. Жодне з ребер не додане до остовного дерева.



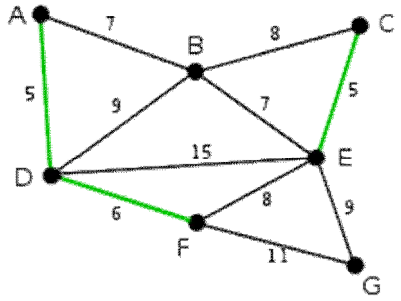
2.AD і CE мають найменшу вагу 5, і AD вибирається з них довільно та додається до остовного дерева.



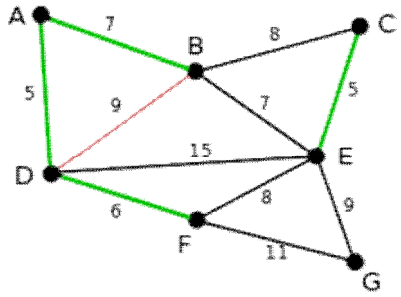
3. На цьому кроці **CE** є найлегшим ребром з вагою 5, тому воно також додається до дерева.



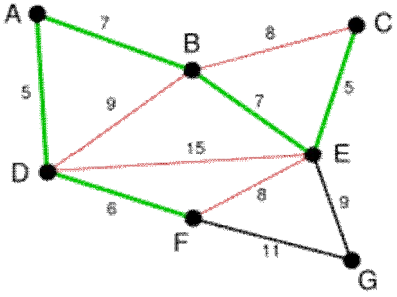
4.Аналогічним чином обирається найлегше з недоданих ребер графа DF з вагою 6 і додається до остовного дерева.



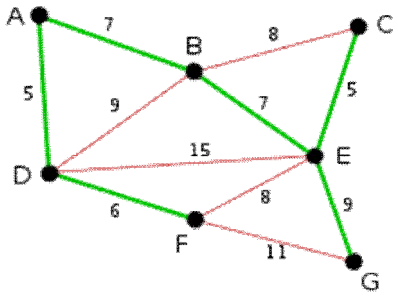
5.Наступними найлегшими ребрами є AB і BE, обидва вагою 7. AB обирається довільно і додається до остовного дерева. BD фарбується у червоний колір, оскільки воно є частиною циклу ABD.



6.Наступнимдодається ребро BE з вагою 7. Червоним забарвлюємо ребра BC (цикл BCE), DE (цикл DEBA) і FE (цикл FEBAD).



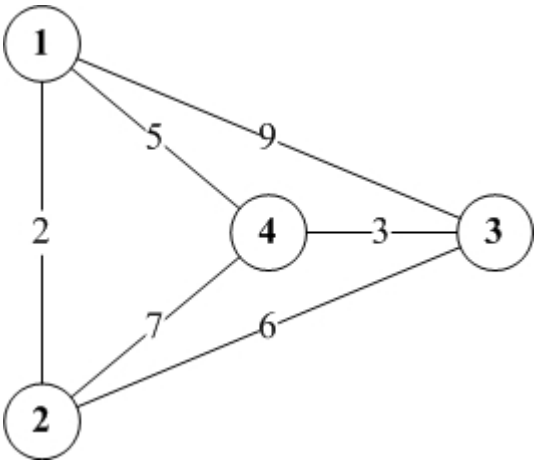
7.Додаємо ребро EG вагою 9 і отримуємо мінімальне остовне дерево.



Наглядне використання алгоритму:

1. Дана карта шляхів між населеними пунктами Вінницької області. Потрібно побудувати схему доріг між цими пунктами, при умові мінімальної затрати ресурсів, які будуть закуплятись для будування цих доріг (асфальту).
2. Дана мапа стовпів села міського типу Стрижавка. Для постачання інтернету необхідно провести кабель до кожної домівки, використавши для цього мінімальну довжину кабелю.

Види задання графу:



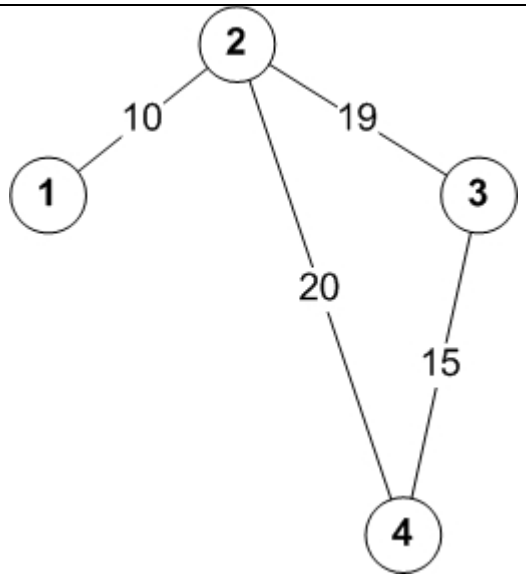
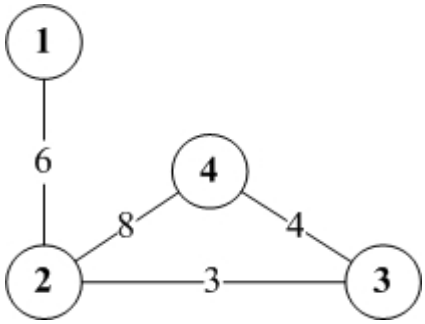
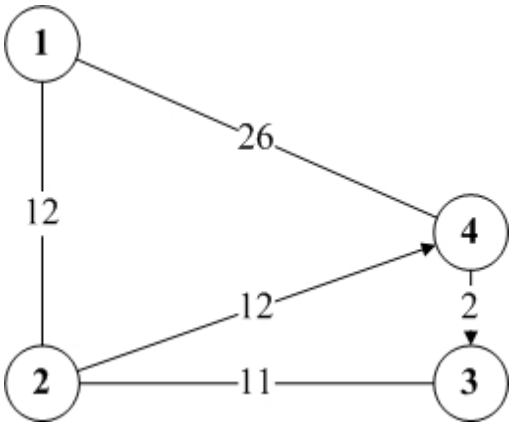
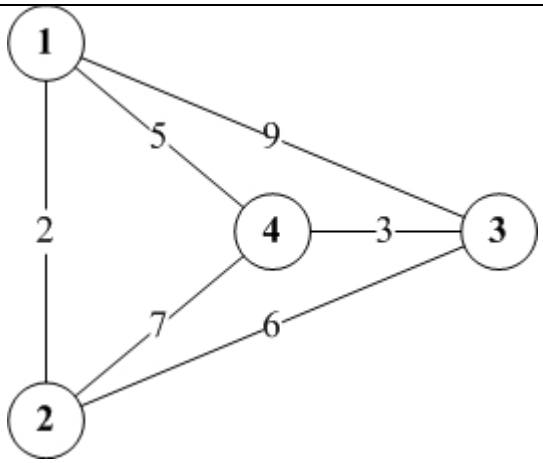
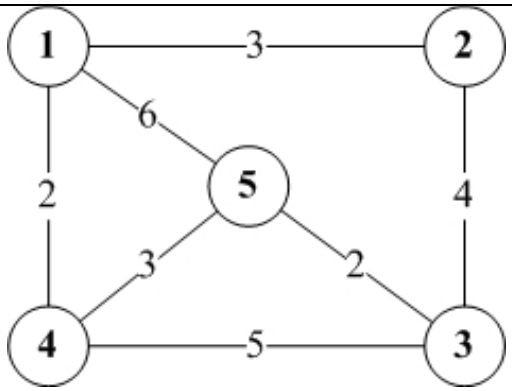
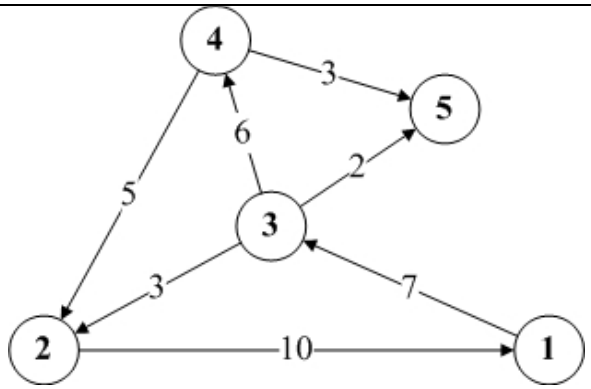
0	2	9	5
2	0	6	7
9	6	0	9
5	7	3	0

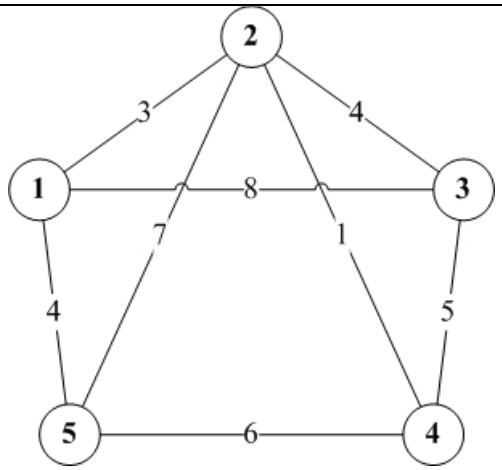
графічно матрично

Порядок виконання і звітування:

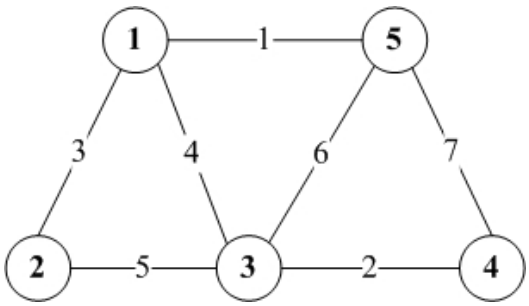
1. Скласти програму для реалізації алгоритму Прима (Java/C++) згідно варіанту.
2. Відкомпілювати та відлагодити програму.
3. Протестувати роботу програми використовуючи різні вхідні дані.
4. Відповісти на контрольні запитання.
5. Зробити висновки.
6. Звіт по лабораторній роботі має складатися з титульної сторінки, мети , лістингів програм, висновків по роботі.

Варіанти завдань(алгоритм Прима):

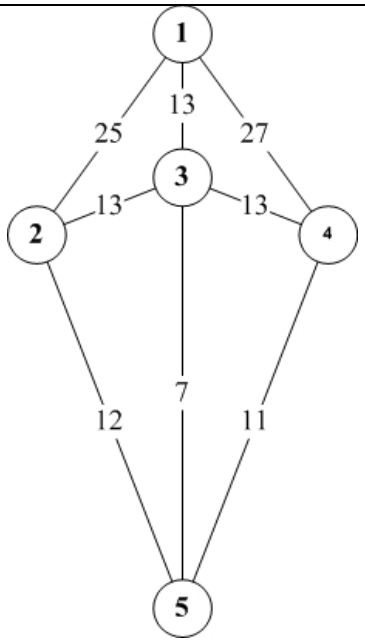
<table>
<tr><td>Варіант - 1</td><td>Варіант - 2</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>Варіант - 3</td><td>Варіант - 4</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>Варіант - 5</td><td>Варіант - 6</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>Варіант - 7</td><td>Варіант - 8</td></tr>
</table>



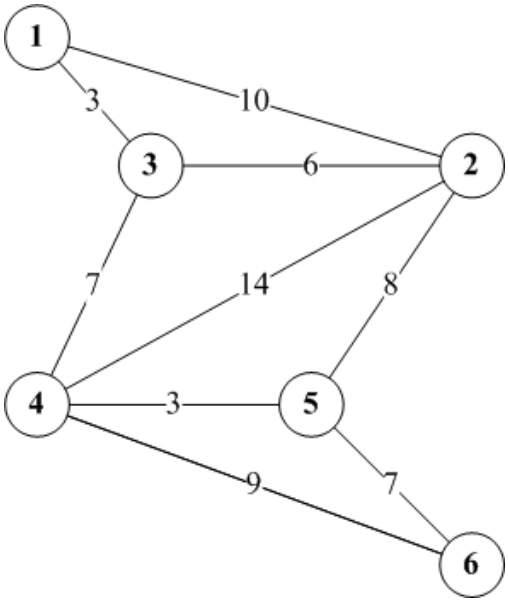
Варіант – 9



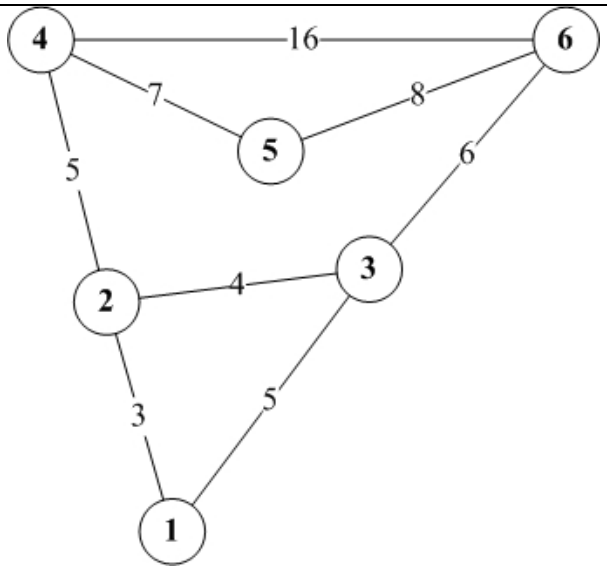
Варіант – 10

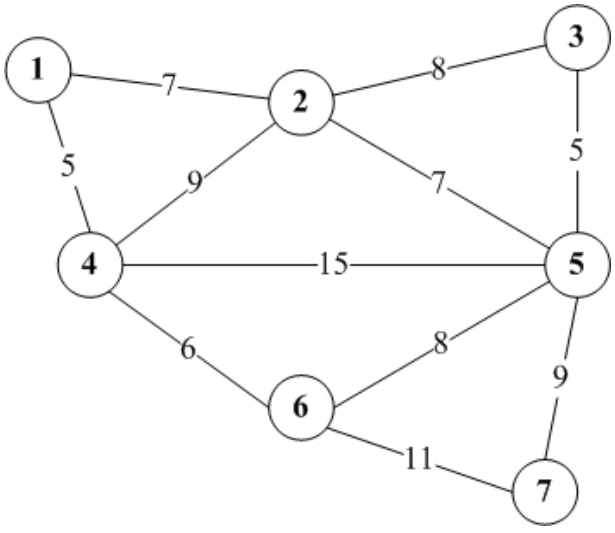


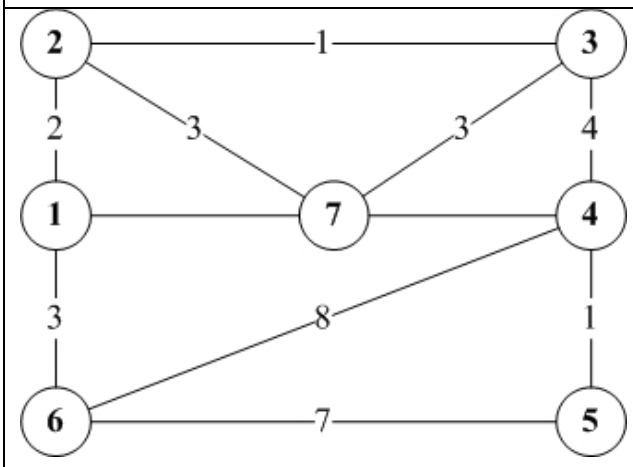
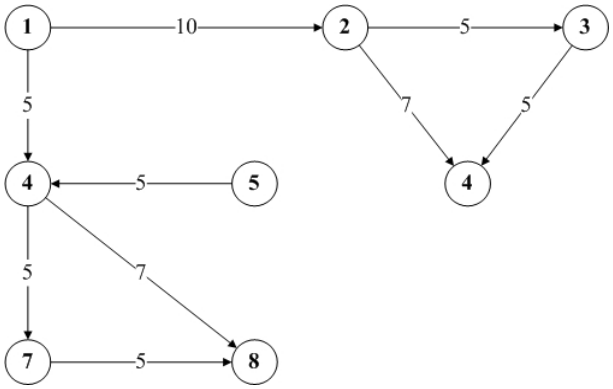
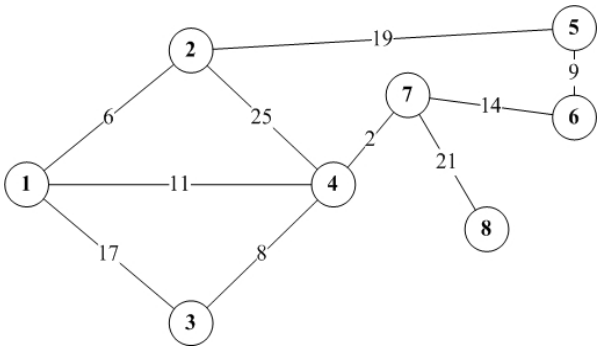
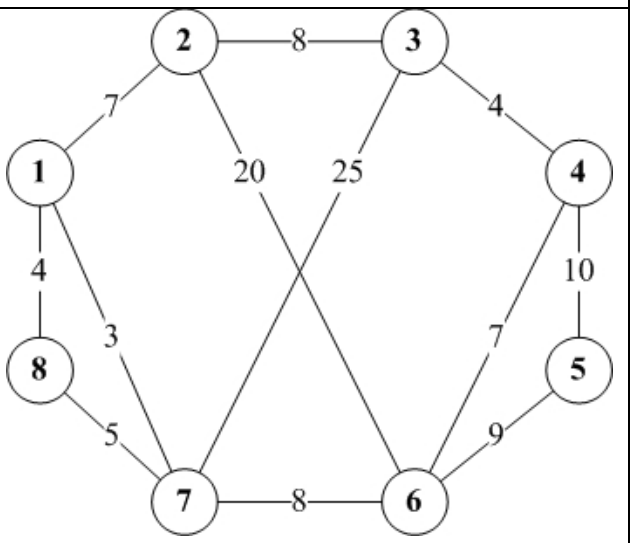
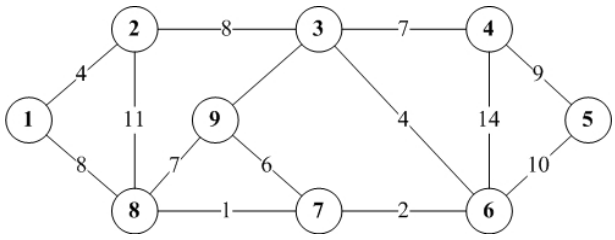
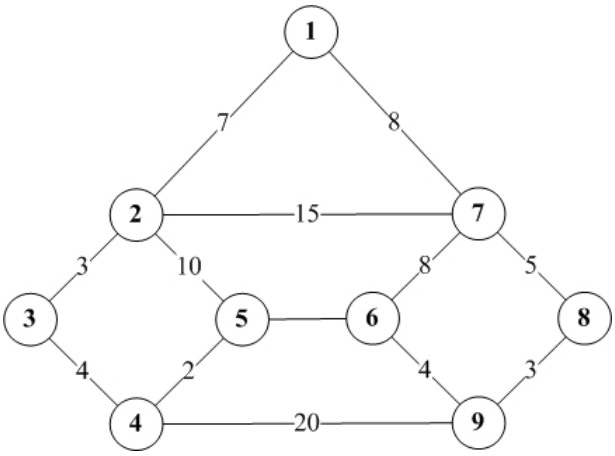
Варіант - 11



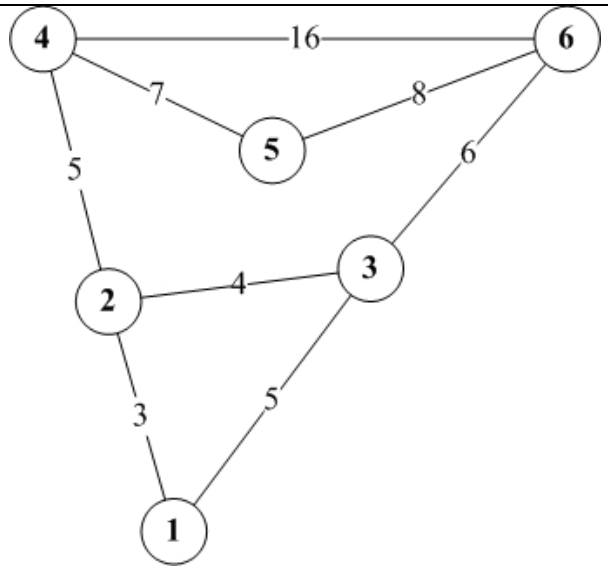
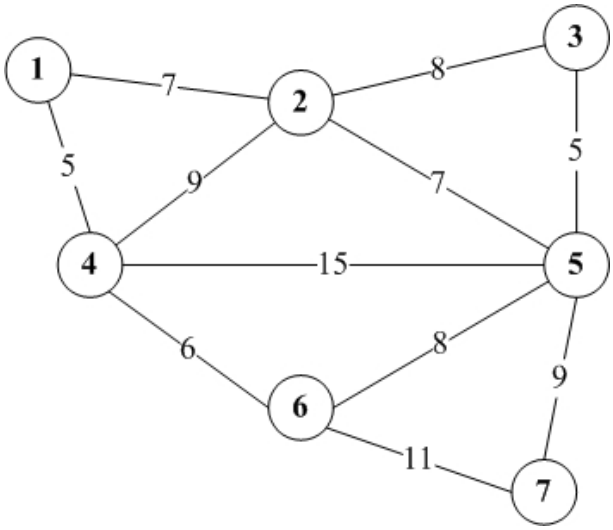
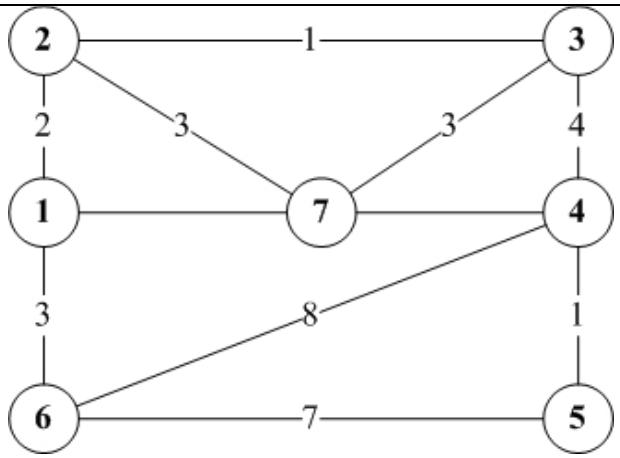
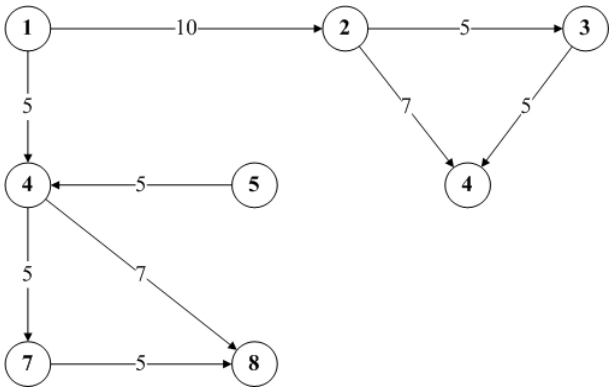
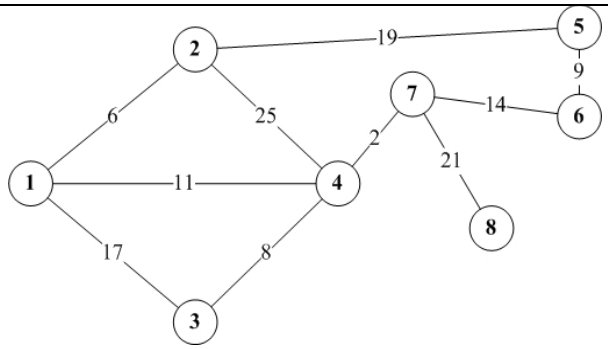
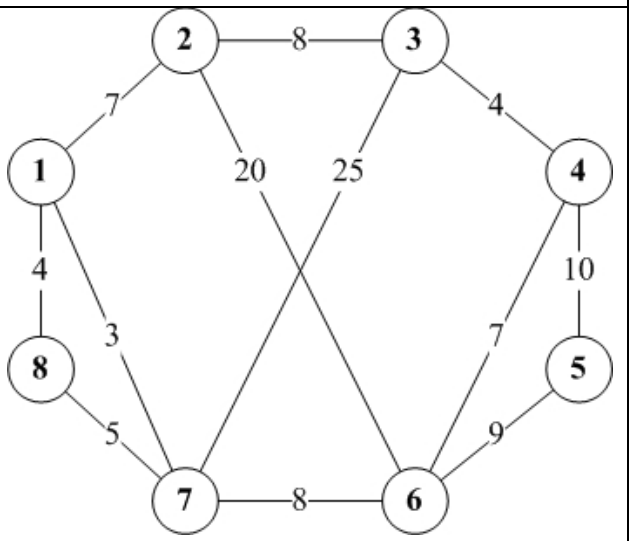
Варіант - 12

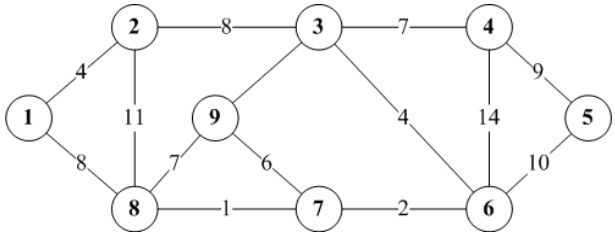
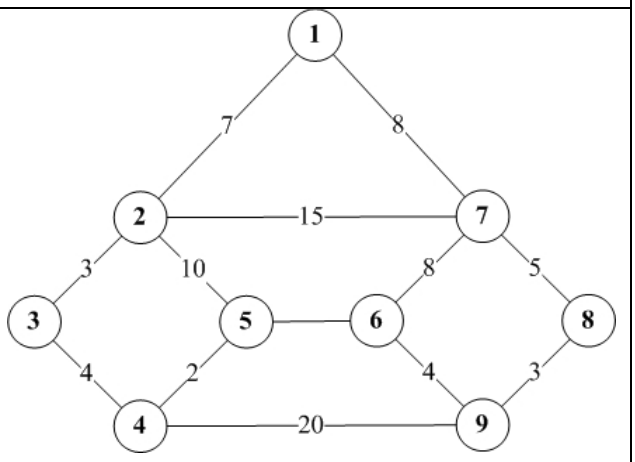
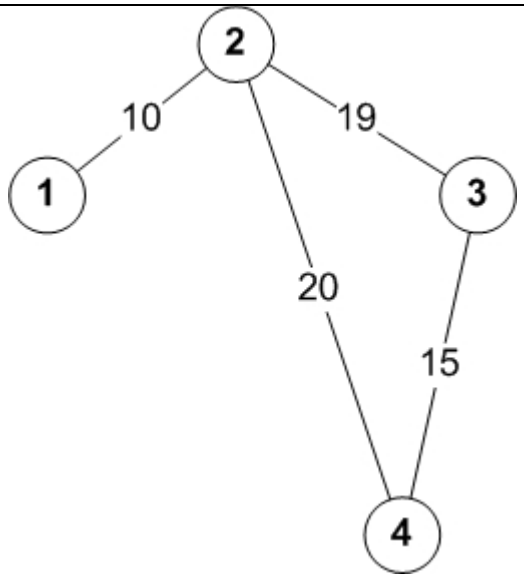
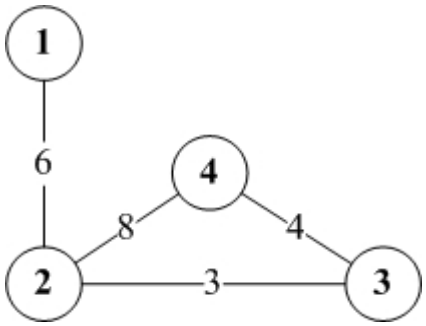
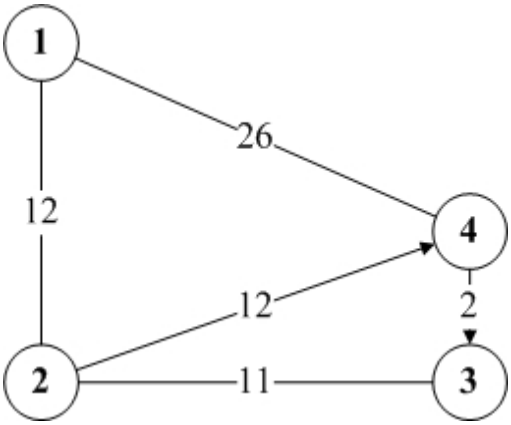
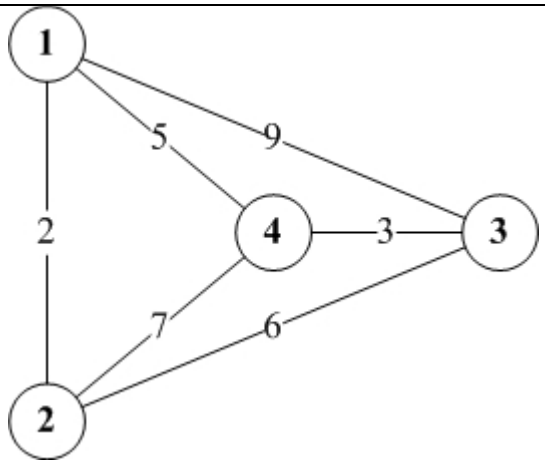


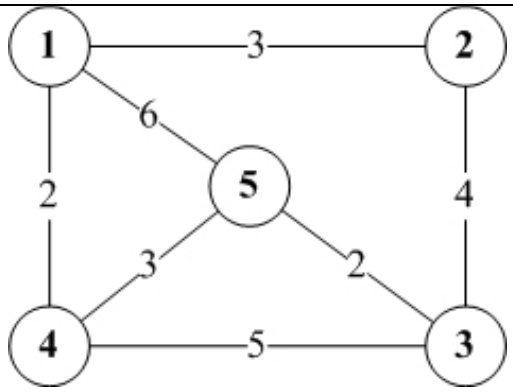
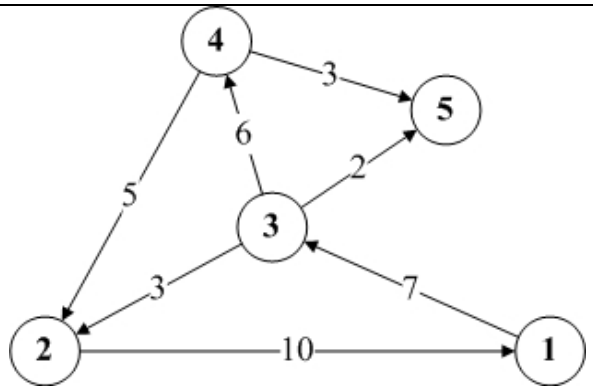
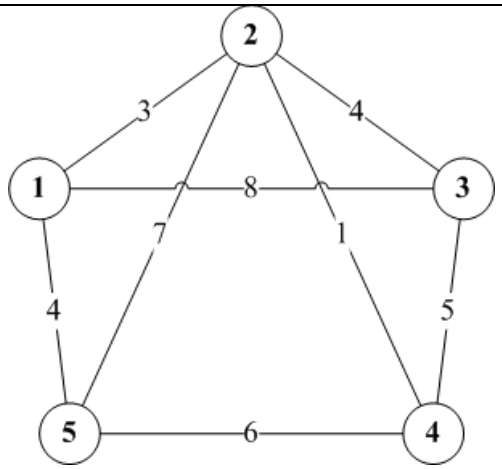
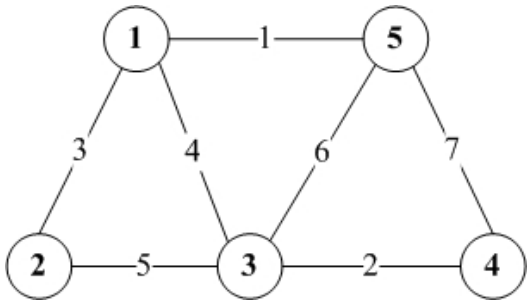
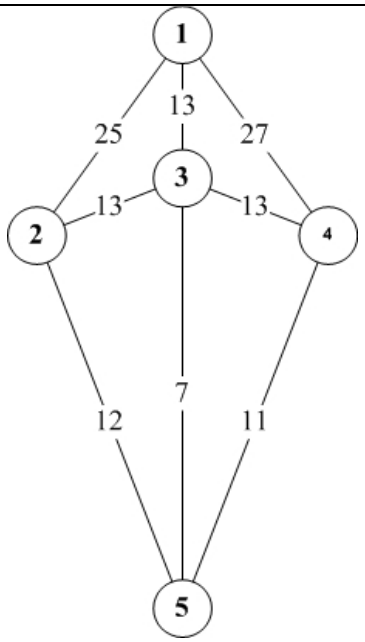
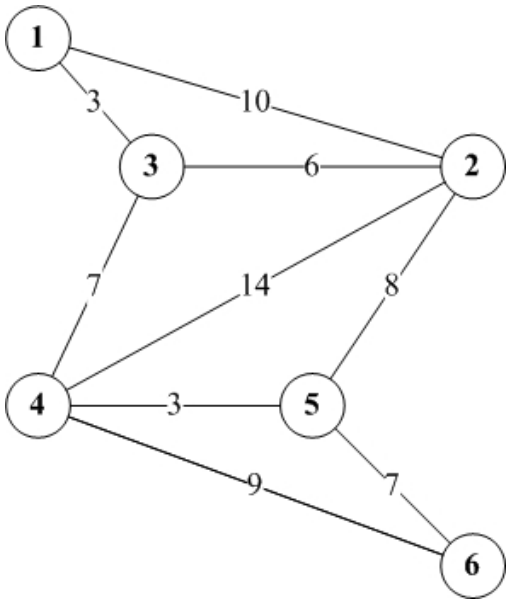


Варіант - 13	Варіант - 14
	
Варіант - 15	Варіант - 16
	
Варіант - 17	Варіант - 18
	

Варіанти завдань(алгоритм Крускала):

Варіант - 19	Варіант - 20
	
Варіант - 21	**Варіант - 22**
	
Варіант - 23	**Варіант - 24**
	

Варіант - 25	Варіант - 26
	
Варіант – 27	Варіант – 28
	
Варіант - 29	Варіант - 30
	

Варіант - 31	Варіант - 32
	
Варіант - 33	Варіант - 34
	
Варіант - 35	Варіант - 36
	

Підсумок

Після виконання лабораторної роботи студент повинен розуміти сутність алгоритмів Прима та Крускала та вміти створювати програми з його застосуванням.

Контрольні питання

1. Що таке граф?
2. Що таке кістякове дерево?
3. Що називають вагою кістякового дерева?
4. В чому полягає алгоритм Прима/Крускала?
5. Де застосовуються алгоритми в реальному житті?

Лабораторна робота №3. Бінарні дерева пошуку

Мета і задачі

Навчитися реалізовувати структури даних у вигляді двійкового дерева пошуку.

Теоретичні відомості і методичні вказівки

В програмуванні *двійкове дерево* — структура даних у вигляді дерева, в якому кожна вершина має не більше двох дітей. Зазвичай такі діти називаються правим та лівим. На базі двійкових дерев будуються такі структури, як двійкові дерева пошуку та двійкові купи.

Реалізація двійкових дерев

В залежності від задач, які вирішуються цими структурами та можливостей тої чи іншої мови програмування, існує декілька варіантів конструювання двійкових дерев.

Реалізація з використанням вказівників передбачає зберігання в кожній вершині дерева x, разом з даними двох полів (правим та лівим) right[x] та left[x], вказівників на відповідних дітей цієї вершини.

Також іноді додається вказівник p[x] на батьківську вершину. Це спрощує деякі алгоритми та виявляється корисним, коли необхідно забезпечити швидкий доступ до батьківської вершини. Іноді достатньо тільки вказівника на батьківську вершину. Взагалі будь-яке орієнтоване дерево можна описати, знаючи тільки зв'язки від дітей до батьківської вершини.

Деякі різновиди двійкових дерев (наприклад, червоно-чорні дерева або AVL-дерева), вимагають збереження в вершинах і деякої додаткової інформації. Якщо у вершини відсутня одна чи обидві дитини, відповідні вказівники ініціалізуються спеціальними "пустими" значеннями.

Двійкові дерева також можуть бути побудовані на базі масивів (рис. 3.1). Такий метод набагато ефективніший щодо економії пам'яті. В такому представленні, якщо вершина має порядковий номер i, то її діти знаходяться за індексами 2i+1 та 2i+2, а батьківська вершина за індексом ((i-1)/2) (за умов, що коренева вершина має індекс 0).

Інший варіант зберігання дерева в масиві — зберігати індекси дітей.

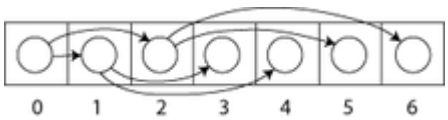


Рисунок 3.1 – Двійкове дерево на базі масиву

Існують такі різновиди двійкових дерев:

- повне (закінчене) двійкове дерево — таке двійкове дерево, в якому кожна вершина має нуль або двох дітей.

- ідеальне двійкове дерево — це таке повне двійкове дерево, в якому листя (вершини без дітей) лежать на однаковій глибині (відстані від кореня).

Двійкове дерево на кожному n-му рівні має від 1 до 2n вершин.

Двійкове (або Бінарне) дерево пошуку — двійкове дерево, в якому кожній вершині x співставлене певне значення val[x]. При цьому такі значення повинні задовольняти умові впорядкованості:

Нехай x — довільна вершина двійкового дерева пошуку. Якщо вершина y знаходиться в лівому піддереві вершини x, то $val[y] \leq val[x]$.

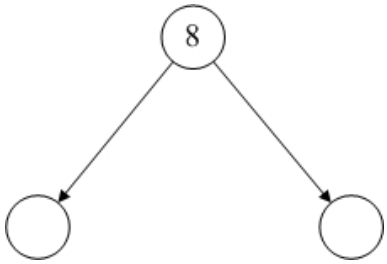
Якщо y знаходиться у правому піддереві x, то $val[y] \geq val[x]$.

Таке структурування дозволяє надрукувати усі значення у зростаючому порядку за допомогою простого алгоритму центрованого обходу дерева.

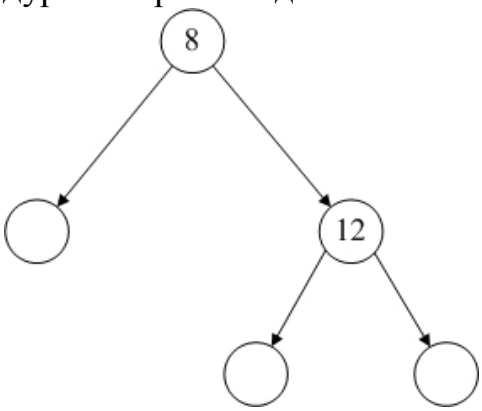
Приклад 3.1.

В бінарне дерево потрібно помістити такі числа: *8, 12, 4, 10, 6, 14, 2.*

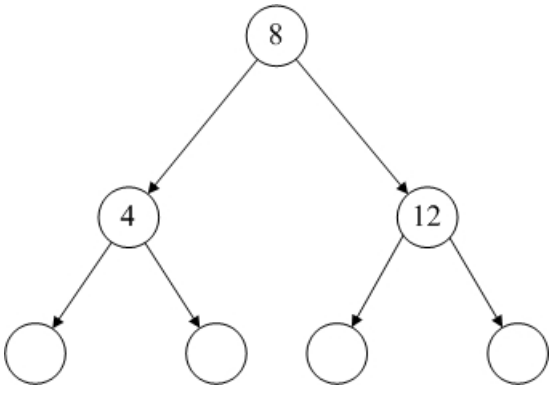
Крок 1: перше число буде «коренем» дерева. Потім створюються ліва та права підкомірка.



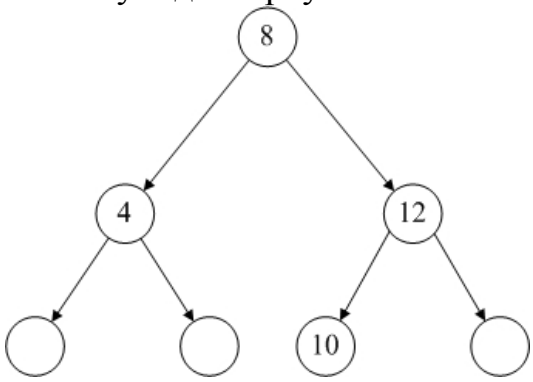
Крок 2: 12 > 8? Так, тому записуємо число у *праву* підкомірку, після чого знов повторюється процедура створення «дітей».



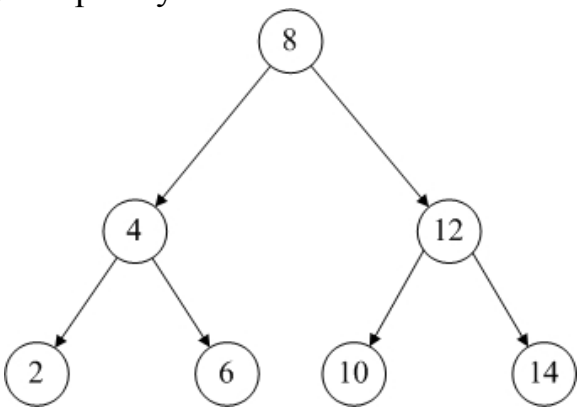
Крок 3: 4>8? Ні, отже записуємо його в ліву підкомірку та знов робимо розгалуження.



Крок 4: Добавляємо число 10. 10>8? Так, рухаємось направо, права підкомірка зайнята числом 12, перевіряємо умову відносно цього числа. 10>12? Ні, отже записуємо число в ліву підкомірку.



Аналогічно записуємо решту чисел:



Побудова бінарного дерева завершено.

Операції з двійковим деревом пошуку.

Пошук

Для пошуку вузла із заданим ключем в бінарному дереві пошуку використовується наступна процедура Tree_Search, яка отримує в якості параметрів покажчик на корінь бінарного дерева і ключ k, а повертає покажчик на вузол з цим ключем (якщо такий існує; в іншому випадку повертається значення NIL).

```
TREE_SEARCH (x, k)

1. if x = nil або k = key [k]
2.    then return x
3. if k <key [x]
4.    then return TREE_SEARCH (left [x], k)
5.    else return TREE_SEARCH (right [x], k)
```

Процедура пошуку починається з кореня дерева і проходить вниз по дереву. Для кожного вузла x на шляху вниз його ключ key[x] порівнюється з переданим в якості параметра ключем k. Якщо ключі однакові, пошук завершується. Якщо k менше key[x], пошук триває в лівому піддереві x; якщо більше - то

пошук переходить в праве піддерево. Ту ж процедуру можна записати ітеративно, "розгортаючи" рекурсію в цикл while.

Реалізація за допомогою C++ :

```
int TREE_SEARCH(X, K)
{
     if(x = NIL)&&(k = key)
       return x;
  if (k < key[x])
       return TREE_SEARCH(left[x], k);
  else
          return TREE_SEARCH(left[x], k);
}
```

Ітеративна версія процедури Пошук

ITERATIVE_TREE_SEARCH(X, K)
1. while x ≠ NIL и k ≠ key[x] 2. doif k ← key[x] 3. then x ← left[x] 4. else x ← right[x] 5. return x

Реалізація за допомогою C++ :

```
int ITERATIVE_TREE_SEARCH(X, K)
{
     while (x != NIL)&&(k != key[x])
     {
          if (k < key[x])
            left[x] -> x;
       else right[x] -> x;
     }
     return x;
}
```

Пошук мінімального (максимального) елемента

Алгоритм пошуку мінімального елемента.

Елемент з мінімальним значенням ключа легко знайти, слідуючи за вказівниками left від кореневого вузла до тих пір, поки не зустрінеться значення NIL. Процедура TREE_MINIMUM(x) повертає покажчик на знайдений елемент піддерева з коренем x.

TREE_MINIMUM(X)
1. whileleft[x] ≠ NIL

```
2.    do x ← left[x]
3. return x
```

Реалізація за допомогою C++ :

```
int TREE_MINIMUM(X)
{
  while(left[x] != NIL)
       left[x] -> x;
  return x;
}
```

Алгоритм пошуку максимального елемента симетричний:

```
TREE_MAXIMUM(X)

1. while right[x] ≠ NIL
2.    do x ← right[x]
3. return x
```

Обидва алгоритми вимагають часу O(h), де h — висота дерева.
Реалізація за допомогою C++ :

```
int TREE_MAXIMUM(X)
{
  while(right[x] != NIL)
       right[x] -> x;
  returnx;
}
```

Наступний і попередній елементи

Якщо x – покажчик на деякий вузол дерева, то процедура TREE_SUCCESSOR(X) повертає покажчик на вузол з наступним за x елементом або nil, якщо зазначений елемент - останній в дереві:

```
                TREE_SUCCESSOR(X)

1. if right[x] ≠  NIL
2.    then return TREE_MINIMUM(right[x])
3. y ← p[x]
4. while y ≠ NIL та x = right[y]
5.    do      x ← y
6.            y ← p[y]
7. return y
```

Наведена процедура окремо розглядає два випадки. Якщо праве піддерево вершини не пусте, то наступний за x елемент є крайнім лівим вузлом у правому піддереві, який виявляється процедурою TREE_MINIMUM(right[x]). З іншого боку, якщо праве піддерево вузла x пусте, та у x існує наступний за ним елемент y, то y є найменшим предком x, чий лівий вузол також є предком x. Для того щоб знайти y, ми просто піднімаємося вгору по дереву до тих пір, поки не зустрінемо вузол, який є лівим дочірнім вузлом свого батька. Ця дія виконується в рядках 3-7 алгоритму.

Час роботи алгоритму TREE_SUCCESSOR в дереві заввишки h складає O(h), оскільки ми або рухаємося по шляху вниз від вихідного вузла, або по шляху нагору. Процедура пошуку подальшого вузла в дереві TREE_PREDECESSOR симетрична процедурі TREE_SUCCESSOR і також має час роботи O(h).

Якщо в дереві є вузли з однаковими ключами, ми можемо просто визначити наступний і попередній вузли як ті, що повертаються процедурами TREE_SUCCESSOR та TREE_PREDECESSOR відповідно.

Реалізація за допомогою C++ :

```
int TREE_SUCCESSOR(X)
{
  if (right[x] != NIL)
       return TREE_MINIMUM(right[x]);
  p[x] -> y;
  while(y != NIL)&&(x = right[y]
      {
        p[x] -> y;
        p[y] -> y;
      }
      return y;
}
```

Додавання елемента

Для вставки нового значення v в бінарне дерево пошуку T ми скористаємося процедурою TREE_INSERT. Процедура отримує як параметр вузол z, у якого key[z] = v,left[z] = NIL і right[z] = NIL, після чого вона таким чином змінює T і деякі поля z, що z виявляється вставленим в відповідну позицію в дереві.

TREE_INSERT (T, Z)
1. y ← NIL 2. x ← root[T] 3. while x ≠ NIL 4. do y ← x 5. ifkey[z] <key [x] 6. then x ← left[x]

```
7.                  else x ← right[x]
8. p [z] ← y
9. if y = NIL
10.   thenroot[T] ← z                    // Дерево T - пусте
11.   elseifkey[z] <key[y]
12.         thenleft[y] ← z
13.         elseright[y] ← z
```

Реалізація за допомогою C++ :

```
int TREE_INSERT(T, Z)
{
  y -> NIL;
  x -> root[T];
  while(x != NIL)
{
  x -> y;
  if(key[z] < key[x])
       left[x] -> x;
  else right[x] -> x;
}
  y -> p[z];
  if (y = NIL)
       z -> root[T];
  else if (key[z] < key[y])
            z -> left[y];
       else z -> right[y];
}
```

Видалення елемента

Процедура видалення даного вузла z з бінарного дерева пошуку отримує в якості аргументу покажчик на z. Процедура розглядає три можливі ситуації:

1. Якщо у вузла z немає дочірніх вузлів, то ми просто змінюємо його батьківський вузол p[z], замінюючи в ньому покажчик на z значенням NIL.

2. Якщо у вузла z лише один дочірній вузол, то ми видаляємо вузол z, створюючи новий зв'язок між батьківським і дочірнім вузлом вузла z.

3. Якщо у вузла z два дочірніх вузла, то ми знаходимо наступний за ним вузол y, у якого немає лівого дочірнього вузла, прибираємо його з позиції, де він перебував раніше, шляхом створення нового зв'язку між його батьком і нащадком, і замінюємо ним вузол Z.

```
DELETE (T, Z)
1  if left[z] = NULL or right[z]=NULL
2    then y:=z
3    else y:=SUCCESSOR(z)
```

```
4  if left[y] <> NULL
5    then x:=left[y]
6    else x:= right[y]
7  if x <> NULL
8    then p[x]:=p[y]
9  if p[y]=NULL
10   then root[T]:=x
11   elseif y=left[p[y] ]
12     then left[p[y] ] :=x
13     else right[p[y] ]:=x
14 if y <> z
15   then val[z]:=val[y]
16 return y
```

Час на виконання цієї процедури є також O(h).
Реалізація за допомогою C++ :

```
int DELETE(T, Z)
{
  if (left[z] == NULL) or (right[z] == NULL)
       y = z;
  else y =SUCCESSOR(z);
  if(left[y] != NULL)
       x = left[y];
  else x = right[y]
  if(x != NULL)
       p[x] = p[y];
  if (p[y] == NULL)
       root[T] = x;
  else (if y = left[p[y]])
            left[p[y]] = x;
       else right[p[y]] = x;
  if (y != z)
       val[z] = val[y];
  return y;
}
```

Порядок виконання і звітування

1. Написати програма яка реалізує двійкове дерево пошуку.
2. Відкомпілювати та відлагодити програму.
3. Протестувати роботу програми використовуючи різні вхідні дані
4. Відповісти на контрольні запитання.
5. Зробити висновки.
6. Звіт по лабораторній роботі має складатися з титульної сторінки, мети , лістингів програм, висновків по роботі.

Варіанти завдань

Варіант 1

Реалізувати функцію додавання/видалення елемента.

Варіант 2

Реалізувати функцію пошуку елемента.

Варіант 3

Реалізувати функцію прямого ходу.

Варіант 4

Реалізувати функцію зворотного ходу.

Варіант 5

Реалізувати функцію пошуку максимального/мінімального елементу

Варіант 6

Реалізувати функцію заміни розташування елемента

Підсумок

Після виконання лабораторної роботи студент повинен вміти реалізовувати необхідні функції для бінарних дерев пошуку.

Контрольні питання

1. Що таке дерево(структура даних)?
2. Які є різновиди дерев?
3. Що таке бінарне дерево?
4. Як є різновиди бінарного дерева?
5. Що таке бінарне дерево пошуку?
6. Що таке вершина, корінь, предки, нащадки?
7. Поясніть алгоритм додавання значень у бінарне дерево пошуку.
8. Наведіть приклад застосування алгоритму бінарного дерева пошуку.

Контрольні вправи

1. Напишіть функцію яка буде графічно відображати бінарне дерево.
2. Напишіть функцію для виведення усіх предків/нащадків.

Джерела інформації

1. http://uk.wikipedia.org/wiki/Дерево_(структура_даних)
2. http://uk.wikipedia.org/wiki/Двійкове_дерево
3. http://uk.wikipedia.org/wiki/Двійкове_дерево_пошуку
4. http://sergiyshumakov.wordpress.com/2012/01/02/бінарне-дерево-пошуку/
5. http://www.rsdn.ru/article/alg/binstree.xml
6. http://comp-science.narod.ru/Progr/BinTree.htm
7. http://posibnyky.vntu.edu.ua/database/gl_31.html
8. http://www.cprogramming.com/tutorial/lesson18.html
9. http://www.declic.narod.ru/ossio/files/book/part_7_3.html#_Toc499287827

Лабораторна робота №4. Функції розподілу

Мета і задачі

Набути навиків використання інструментів Statistics Toolbox 5.0 в програмному середовищі MatLab при дослідженні функцій розподілу.

Теоретичні відомості та методичні вказівки

Завдання групи функцій одновимірних розподілів.

Основними завданнями функцій групи одновимірних розподілів є: розрахунок значень диференціального й інтегрального закону розподілів, обернених функцій розподілу, генерація псевдовипадкових чисел за заданим законом розподілу, розрахунок математичного очікування і дисперсії для заданого закону з відомими параметрами, розрахунок точкових та інтервальних оцінок параметрів законів розподілу за вибірковими даними, розрахунок негативного логарифма функції максимальної правдоподібності. Наведені перші 6 завдань є основними. Останнє завдання виконує допоміжні функції при розрахунку точкових оцінок параметрів закону розподілу.

Розрахунок функції розподілу щільності ймовірності.

Функція розподілу щільності ймовірності дискретного закону являє собою ряд розподілу, що ставить у відповідність значення дискретної випадкової величини і ймовірності появи цих значень.

Приклад 4.1.

Для біноміального закону з параметрами: вірогідністю вдалого результату з одному досвіді 0,5 і числом випробувань 10, ряд розподілу P для значень випадкової величини k, числа вдалих результатів в серії дослідів, прийме наступний вигляд:

```
>> p=0.5;
>> n=10;
>> k=0:1:n;
>> P=binopdf(k,n,p);
>> [k' P']
```

Результат виконання наведеного коду наступний:

```
ans =
0         0.0010
1.0000    0.0098
2.0000    0.0439
3.0000    0.1172
4.0000    0.2051
5.0000    0.2461
```

```
6.0000   0.2051
7.0000   0.1172
8.0000   0.0439
9.0000   0.0098
10.0000  0.0010
```

Для виведення графіку функції розподілу виконаємо наступний код:

```
>>plot(k,P,'d')
>>grid on
```

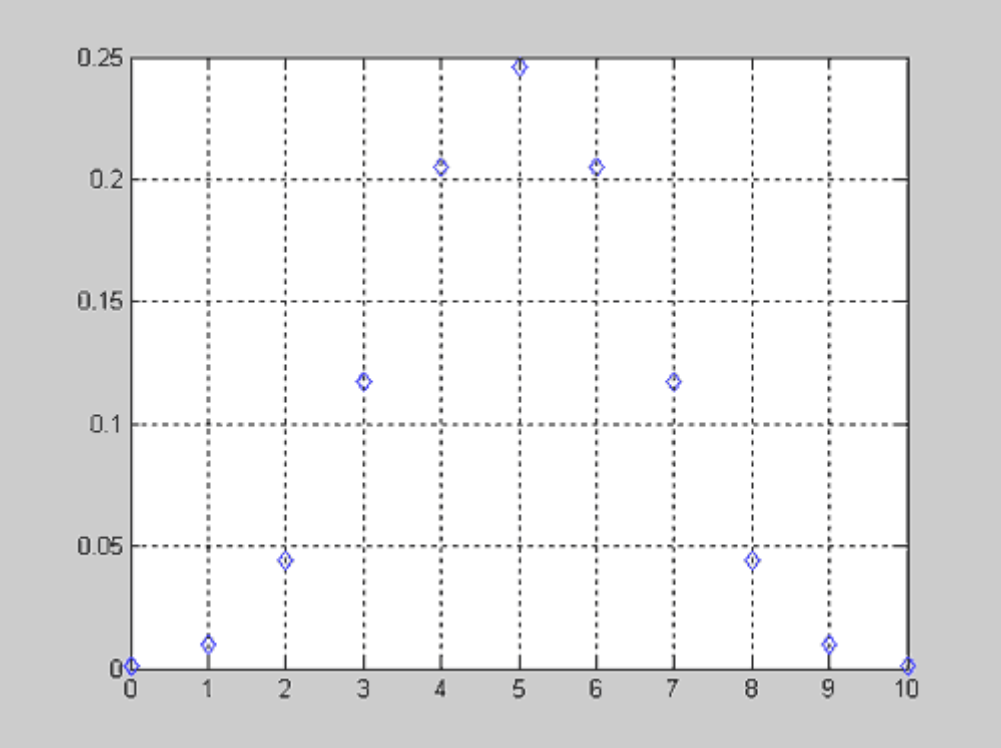


Рисунок 4.1 – Графік функції біноміального розподілу

Одномірні розподіли служать для опису ймовірності появи значень одновимірної випадкової величини в результаті одного або серії дослідів. Прикладом використання дискретної ймовірнісної моделі може служити розрахунок ймовірності числа дефектів на одиницю довжини стрічки з магнітним носієм. Так як кількість дефектів не може приймати дробові значення, то для опису розподілу ймовірностей числа дефектів повинен бути використаний дискретний закон. Якщо випадковою величиною є товщина стрічки з магнітним носієм, то моделлю розподілу ймовірності має бути безперервний розподіл, внаслідок того, що товщина стрічки може приймати будь-які значення з деякого діапазону.

Функція розподілу щільності ймовірності безперервного закону є безперервною функцією і визначається як похідна від інтегрального закону розподілу :

$$f(X) = \frac{dF(X)}{dX},$$

де f - функція розподілу щільності ймовірності, X - одномірна випадкова величина, F - інтегральний закон розподілу випадкової величини.

Ймовірність влучення значень випадкової величини в заданий інтервал визначається як площа під кривою функції для цього інтервалу. Отже, значення функції розподілу щільності ймовірності не відповідає ймовірності заданого значення випадкової величини, оскільки значення інтеграла для однакових меж дорівнює нулю.

Функція розподілу щільності ймовірності характеризується двома властивостями:

Значення функції не може бути негативним:

$f(X) \le 0$.

Вид функцій розподілу щільності ймовірності для нормального, рівномірного, експоненціального і бета законів прийме вигляд:

```
>> x=0:0.01:2;
>> f1=normpdf(x,1,1/3);
>> f2=betapdf(x,2,3);
>> f3=exppdf(x,1);
>> f4=unifpdf(x,0.5,1.5);
>> plot(x,f1,'r-',x,f2,'b.-',x,f3,'g--',x,f4,'m--')
```

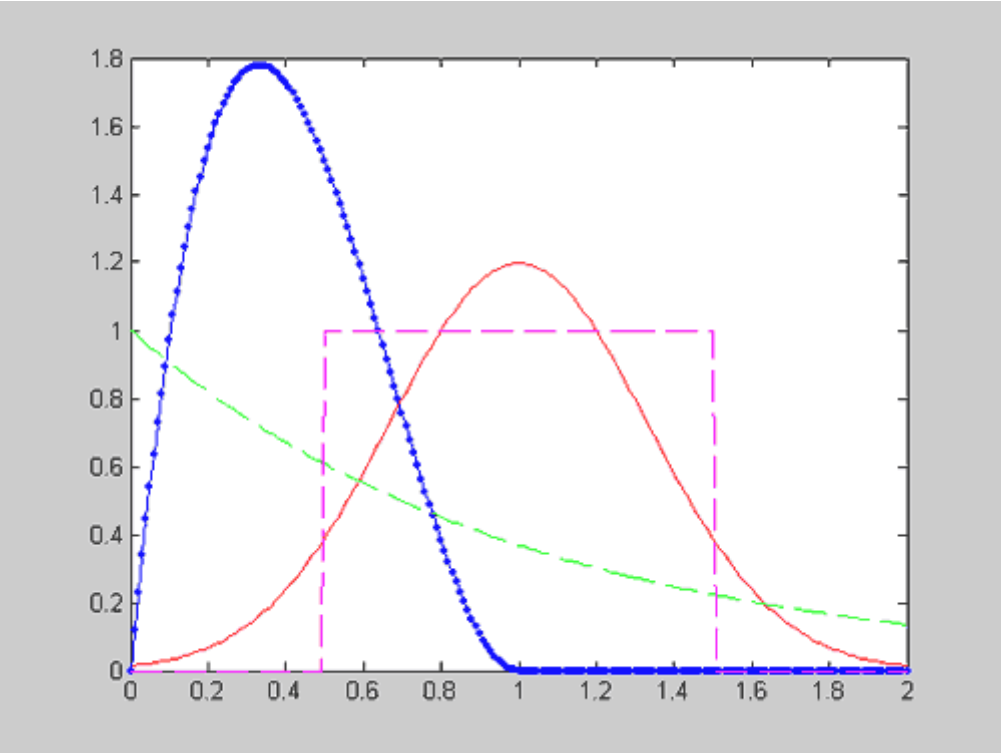


Рисунок 4.2 – Графіки функцій розподілу щільності ймовірності для нормального, рівномірного, експоненціального і бета законів

Для опису розподілу значень одновимірної випадкової величини крім виду закону розподілу (рис. 4.1) необхідно знати параметри використовуваного закону. Тому закон розподілу одновимірної випадкової величини можна розглядати як параметричне сімейство функцій щільності розподілу ймовірностей.

Правило формування ідентифікаторів функцій щільності ймовірності засноване на використанні приставки, яка описує назву закону розподілу, і

суфікса ' pdf '. Список законів розподілу і відповідних їм приставок наведено в табл. 4.1.

Таблиця 4.1 – Список функцій розподілу реалізованих в Matlab

Вид розподілу	Приставка
Бета	' beta ', ' Beta '
Біноміальний	' bino ', ' Binomial '
Xi - квадрат	' chi2 ', ' Chisquare '
Експоненціальне	' exp ', ' Exponential '
Розподіл екстремальних значень	' ev ', ' ExtremeValue '
Фішера	' f ', 'F'
Гамма	' gam ', ' Gamma '
Геометричне	' geo ', ' Geometric '
Гіпергеометричний	' hyge ', ' Hypergeometric '
Логнормальний	' logn ', ' Lognormal '
Від'ємний біноміальний	' nbin ', ' NegativeBinomial '
Зміщене Фішера	' ncf ', ' Noncentral F '
Зміщене Стьюдента	' nct ', ' Noncentral T '
Зміщене хі-квадрат	' ncx2 ', ' NoncentralChi - square '
Нормальне	' norm ', ' Normal '
Пуассона	' poiss ', ' Poisson '
Релея	' rayl ', ' Rayleigh '
Стьюдента	' t ', 'T '
Рівномірний	' unid ', ' DiscreteUniform '
Безперервне рівномірне	' unif ', ' Uniform '
Вейбулла	' weib ', ' Weibull ', ' wbl '

Наприклад, ідентифікатор функції щільності ймовірності нормального закону - normpdf, безперервного рівномірного - unifpdf і т.д.

При виклику функцій розподілу щільності ймовірності одновимірних випадкових величин передбачений наступний порядок передачі аргументів:

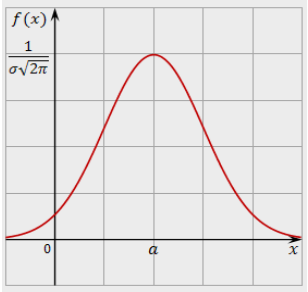
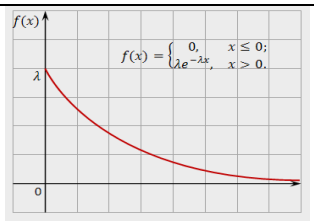
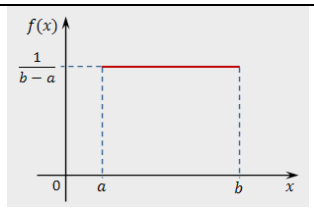
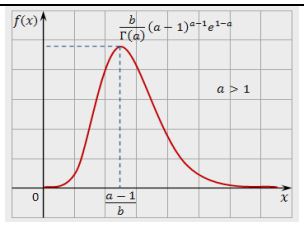
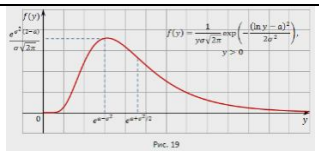
1. Скаляр, вектор або матриця значень випадкової величини;

2. Скаляр, вектор або матриця значень першого параметра закону розподілу;

3. Скаляр, вектор або матриця значень другого параметра закону розподілу і т.д.

Якщо значення параметрів законів розподілу задаються у вигляді вектора або матриці, то розмірності всіх переданих параметрів повинні збігатися. Як правило, порядок передачі параметрів функції розподілу щільності ймовірності збігається з загальноприйнятим в літературі. Скалярний параметр доповнюється до розмірності інших параметрів. Розмірність вихідного параметра збігається з розмірностями вхідних аргументів.

Таблиця 4.2 – Типові розподіли ймовірностей

Закон розподілу	Щільність	Графік щільності	Момент розподіл
Нормальний (Гаусівський закон)	$f(x)=\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(x-a)^2}{2\sigma^2}\right).$		$M(X)=a;$ $D[X]=\sigma^2.$
Експоненціальний	$f(x)=\begin{cases}0, & x\leqslant 0;\\ \lambda e^{-\lambda x}, & x>0.\end{cases}$		$M(X)=\frac{1}{\lambda};$ $D[X]=\frac{1}{\lambda^2}.$
Рівномірний	$f(x)=\begin{cases}0, & x\notin[a;b],\\ \frac{1}{b-a}, & x\in[a;b].\end{cases}$		$M(X)=\frac{a+b}{2};$ $D[X]=\frac{(b-a)^2}{12}.$
Гамма-розподіл	$f(x)=\begin{cases}0, & x\leqslant 0;\\ \frac{b^a}{\Gamma(a)}x^{a-1}e^{-bx}, & x>0.\end{cases}$		$M(X)=\frac{a}{b};$ $D[X]=\frac{a}{b^2}.$
Логарифмічний нормальний	$f(y)=\frac{1}{y\sigma\sqrt{2\pi}}\exp\left(-\frac{(\ln y-a)^2}{2\sigma^2}\right),$		$M(Y)=\exp\left(a+\frac{\sigma^2}{2}\right);$

			$D[Y] = e^{2(2\sigma^2+a)^2-a}$

Приклад 4.2.

Потрібно описати функцію розподілу щільності ймовірності по закону Релея та побудувати графік цієї функції.

```
mu = 0:1:20; r1 = raylpdf(mu,3)
r2 = raylpdf(mu,8)
r3 = raylpdf(mu,5)
plot(mu,p,'-r',mu,p1,'-g',mu, p2,'-b')
gridon
```

Результат роботи програми наведений на рисунку 4.3:

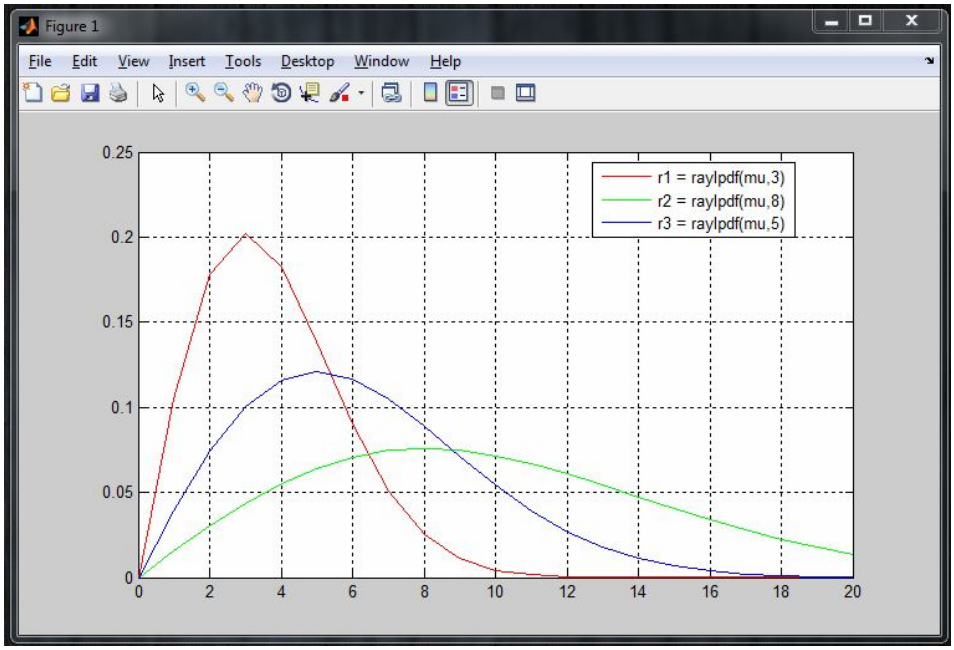


Рисунок 4.3 – Графік функції розподілу щільності ймовірності по закону Релея

Порядок виконання і звітування

1. В програмному середовищі MatLab описати функції розподілу по трьом законам згідно варіанту.
2. Побудувати їх графіки.
3. Продемонструвати роботу програми.
4. Відповісти на контрольні запитання.
5. Зробити висновки.
6. Звіт по лабораторній роботі має складатись з титульної сторінки, лістингів програм, висновків по роботі.

Варіанти завдань:

1	**betapdf** - Бета розподіл **binopdf** - Біноміальний розподіл **pdf**- Параметризована функція розподілу
2	**chi2pdf** - Функція розподілу хі - квадрат **exppdf** - Експоненціальний розподіл **epdf**- Емпірична функція розподілу (на основі оцінки Каплана - Мейера)
3	**fpdf** - Розподіл Фішера **gampdf** - Гамма розподіл **geopdf** – Геометричний розподіл
4	**hygepdf** - Гіпергеометричний розподіл **lognpdf** - Логнормальний розподіл **nbinpdf** - Від'ємний біноміальний розподіл
5	**ncfpdf** - Зміщений розподіл Фішера **nctpdf** - Зміщений розподіл Стьюдента **ncx2pdf** - Зміщений хі-квадрат розподіл
6	**normpdf** – Нормальний розподіл
	poisspdf - Розподіл Пуассона
	raylpdf - Розподіл Релея
7	**tpdf** - Розподіл Стьюдента
	unidpdf – Дискретний рівномірний розподіл
	unifpdf – Безперервний рівномірний розподіл

Підсумок

Після виконання лабораторної роботи студент повинен розуміти сутність основних положень даного завдання та вміти працювати із Statistics Toolbox в MatLab.

Контрольні запитання

1. Що таке MatLab?
2. Що таке StatisticsToolbox?
3. Що являє собою функція розподілу щільності ймовірності?
4. Як описати функції розподілу в MatLab?

5. Як будувати графік? Що будуть показувати вісі Хта Y?

Список використаної літератури

1. http://matlab.exponenta.ru/statist/book2/
2. http://uk.wikipedia.org/wiki/Функція_розподілу_ймовірностей
3. http://mathhelpplanet.com/static.php?p=osnovnye-zakony-raspredeleniya-nepreryvnyh-sluchainyh-velichin

Лабораторна робота №5. Функції щільності розподілу випадкових величин. Генерація псевдовипадкових чисел. GUI у Matlab.

Мета і задачі

Навчитись створювати та відлагоджувати програми, в яких будуються графіки функцій щільності розподілу випадкових величин з можливістю зміни параметрів за допомогою користувацького інтерфейсу у середовищі Matlab. Навчитися генерувати послідовність псевдовипадкових чисел по заданому закону розподілу у середовищі Matlab.

Теоретичні відомості та методичні вказівки

У Matlab для побудови графіка будь-якої функції щільності розподілу випадкових величин потрібно визвати відповідну функцію та передати у неї необхідні параметри.

Приклад 5.1.

Написати код програми, яка будує графік зміщеного закону Стьюдента розподілу випадкової величини:

```
h=0.01
x=-5:h:5;
f1=nctpdf(x,5,-2);
plot(x,f1,'r');
title('Розподіл Стьюдента');
text(-4.5,0.21,'DELTA=-2');
text(-3.8,0.19,'V=5');
xlabel('x');
ylabel('y');
grid on
```

Приклад роботи програми наведений на рисунку 5.1.

Програма починається з оголошення змінної `h`, що зберігає значення кроку та `x`, яка приймає значення від мінус п’яти до п’яти з кроком `h`, всі значення `x` зберігаються у масиві. Змінна `f1` приймає значення, яке повертає функція `nctpdf()`, та записує у масив який за розміром дорівнює масиву `x`. Функція `nctpdf()` приймає три параметри:

- масив значень для яких буде розрахована функція;
- степінь вільності функції;
- параметр зміщення.

Функція `plot()` виводить у вікні графік, у неї передаємо два масиви `x` та `f1`, третій параметр вказує на формат виведення, а саме літера позначає колір графіка, а символ після літери означає якими символами буде виводитись графік.

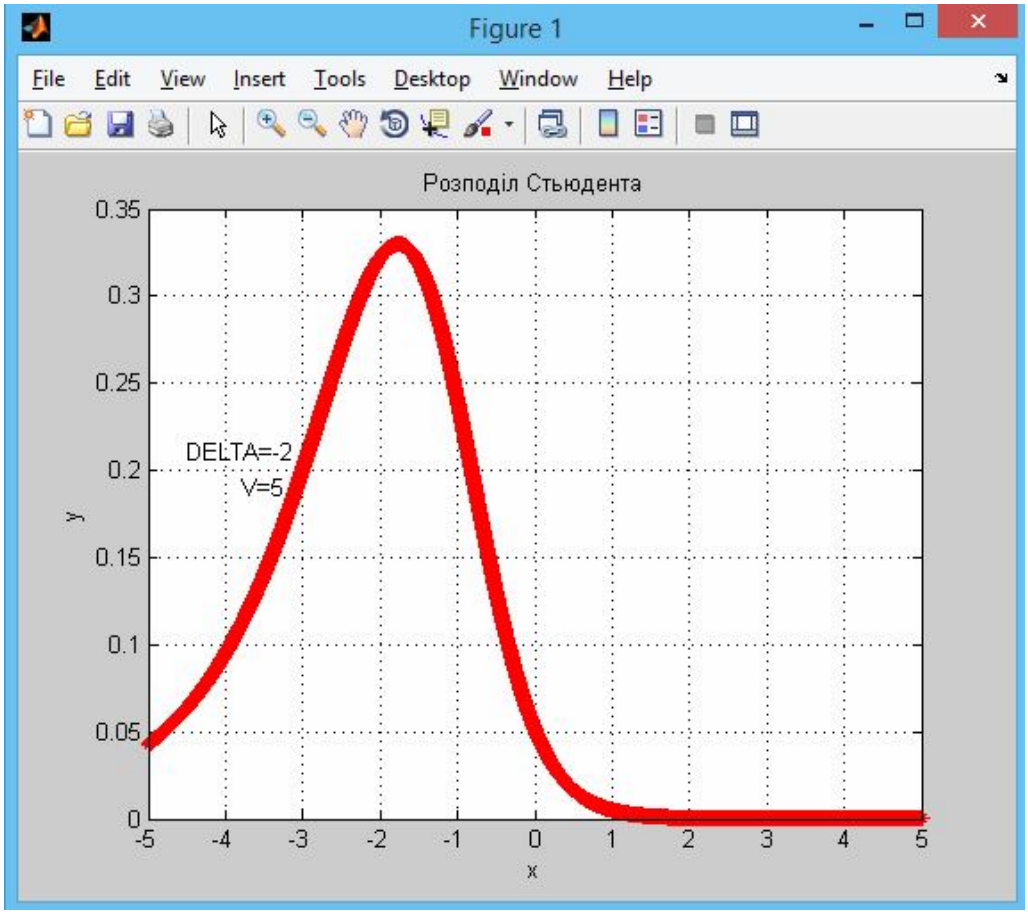


Рисунок 5.1 – Приклад роботи програми

Функції `xlabel()` та `ylabel()` підписують координатні осі. Наступний рядок коду `gridon` він вмикає сітку під час виведення графіка.

Генератор псевдовипадкових чисел (ГПВЧ, англ. *Pseudo random number generator*, PRNG) - алгоритм, який породжує послідовність чисел, елементи якої майже незалежні один від одного і підкоряються заданому розподілу (зазвичай рівномірному).

Для прикладу розглянемо генерацію псевдовипадкових чисел за нормальний розподілом:

```
R = normrnd(0,1,[1 100]);
hist(R, 9);
```

В першій стрічці коду описується змінна, яка буде зберігати масив даних, які повертає функція `normrnd()`, ця функція дозволяє отримати матрицю псевдовипадкових чисел з розмірністю m на n елементів розподілених по нормальному закону для заданих параметрів MU, SIGMA. У функцію передаються відповідні параметри:

- MU – перший параметри функції, математичне очікування;
- SIGMA – другий параметр функції, середнє квадратичне відхилення;
- масив чисел розмірністю m на n.

Друга стрічка коду будує гістограму відповідно заданого закону розподілу. Функція `hist()` приймає два параметри, де перший – це масив повернутих

значень функцією `normrnd()`, а другий параметр кількість стовпців гістограми. На рисунку 5.2 наведений приклад роботи програми.

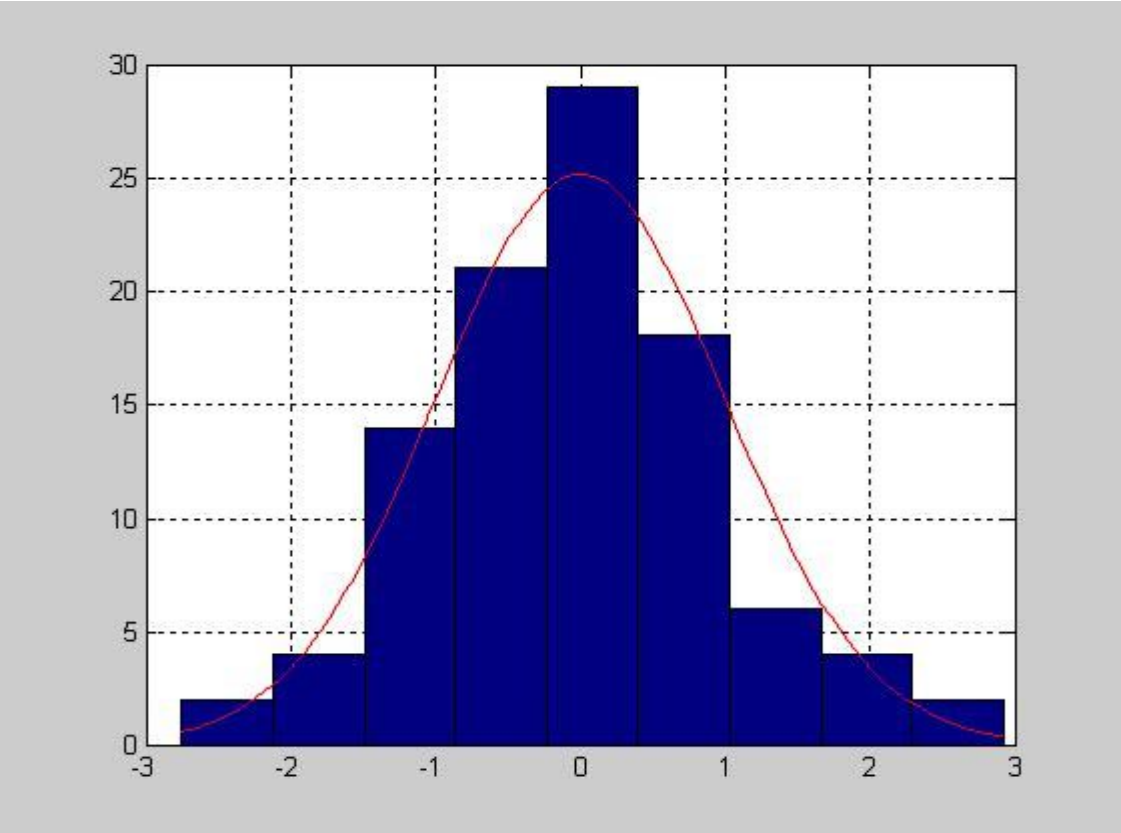


Рисунок 5.2 – Генерація псевдовипадкових чисел за нормальним законом

Для початку роботи з GUI(guide user interfaces) потрібно прописати у командній стрічці Matlab команду `guide`, після чого відкриється MatlabGUIBuilder за допомогою якого можна побудувати відповідне вікно використовуючи потрібні форми зображені на рисунку 5.3.

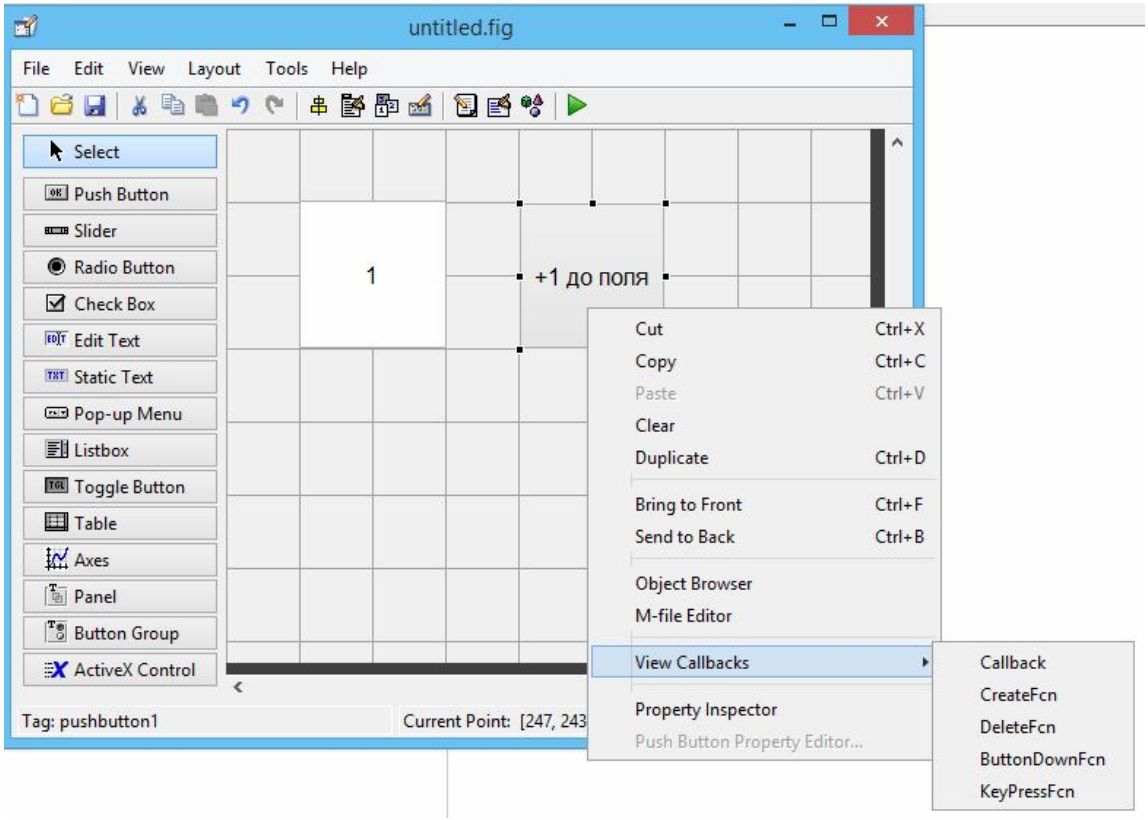


Рисунок 5.3 – GUI Builder Matlab

Після того, як на вікно добавлені відповідні форми потрібно обробляти запити натискання відповідних кнопок. Щоб описати дію, яка буде відбуватись під час натискання кнопки потрібно вибрати кнопку *ПКМ/View Callbacks/Callback*. Для зміни параметрів відповідної форми потрібно викликати *Property Inspector*. На рисунку 5.4 наведено приклад вікна для зміни параметрів кнопки.

Приклад програми, яка при натисканні на кнопку буде додавати одиницю до числа, яке знаходиться в полі *EditText*.

```
function pushbutton15_Callback(hObject, eventdata, handles)
temp_value = get(handles.edit1,'String');
temp_value1 = str2double(temp_value) + 1;
set(handles.edit1,'String',temp_value1);
end
```

Функція, яка описує натискання кнопки починається з назви функції `pushbutton15_Callback()`, яка приймає три стандартних параметри. У наступній стрічці в змінну `temp_value` присвоюється значення з текстового поля. Функція `str2double()` перетворює тип `String` у `Double` для того, щоб можна було збільшити його на одиницю. У четвертій стрічці функція `set()` встановлює значення `temp_value1` у текстове поле `handles.edit1`, `handles` – це вікно у якому знаходиться форма.

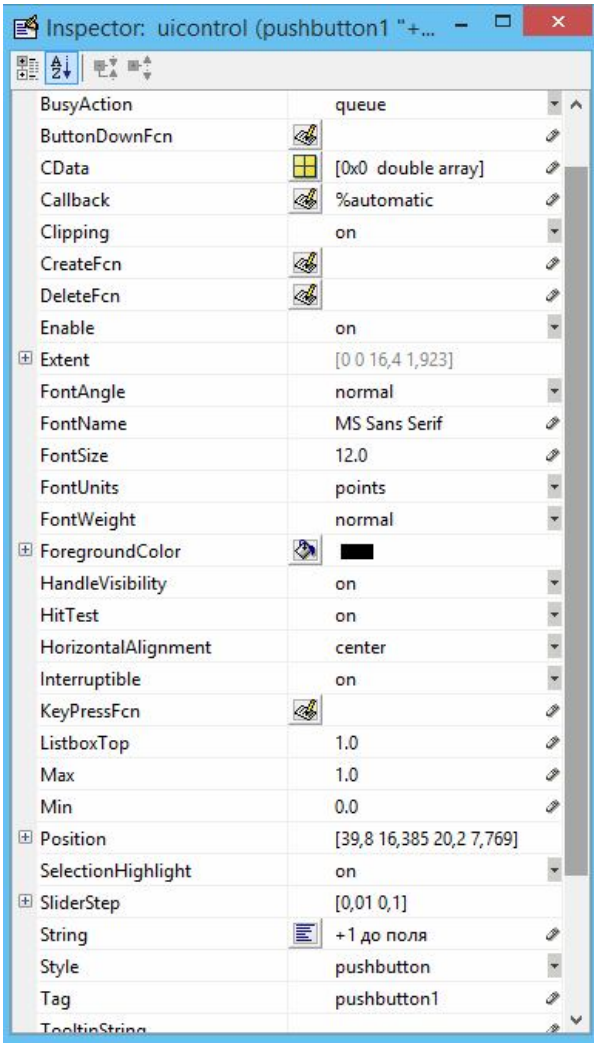


Рисунок 5.4 – Меню зміни параметрів для кнопки

Порядок виконання і звітування

1. Написати програму, яка демонструє роботу функцій заданої відповідно до варіанту використавши САПР Matlab.
2. Відлагодити та запустити програму.
3. Розробити набір тестів і перевірити роботу програми на них.
4. Відповісти на контрольні запитання.
5. Зробити висновки.
6. Звіт по лабораторній роботі має складатись з титульної сторінки, лістингів програм, висновків по роботі.

Варіанти завдань

Написати програму з використанням GUI Matlab, яка буде виводити на екран графік функцій розподілу щільності випадкової величини та послідовність псевдовипадкових чисел з можливістю зміни параметрів функції.

Варіант 1.
Бета розподіл
Біноміальний розподіл
Хі-квадрат розподіл

Варіант 2.
Експоненціальний розподіл
Розподіл Фішера
Гамма розподіл

Варіант 3.
Геометричний розподіл
Гіпергеометричний розподіл
Логнормальний розподіл

Варіант 4.
Від'ємно-біноміальний розподіл
Біноміальний розподіл
Зміщений розподіл Фішера

Варіант 5.
Зміщений розподіл Стьюдента
Зміщений розподіл хі-квадрат
Нормальний розподіл

Варіант 6.
Розподіл Пуассона
Розподіл Релея
Розподіл Стьюдента

Варіант 7.
Дискретно-рівномірний розподіл
Неперервно-рівномірний розподіл
Розподіл Вейбулла

Підсумок

Після виконання лабораторної роботи студент повинен вміти створювати програми, в яких використовуються функції Matlab та знати їх математичне представлення.

Контрольні питання

1. Що таке функція?
2. Як створюється функція у Matlab?
3. Де і як використовуються функції?
4. Що є аргументом функції?
5. Що може робити викликана функція?
6. Що таке псевдовипадкове число?
7. Де використовуються функції розподілу випадкових величин?

Список використаної літератури

1. Потемкин В.Г. Система MATLAB 5 для студентов. – М.: Диалог-МИФИ, 1998 – 314 с.
2. Хьюбер П. Робастность в статистике. М.: Мир, 1984.
3. Катышев П.К., Пересецкий А.А. Сборник задач к начальному курсу эконометрики 72 стр. Москва: Дело, 1999 ISBN: 5-7749-0137-8
4. Лимер Э. Сатистический анализ неэксперементальных данных. М.: Финансы и статистика, 1983.
5. http://inftech.webservis.ru/it/database/datamining.
6. infoscope.forth.ru.
7. http://algolist.manual.ru/maths/.
8. http://matlab.exponenta.ru/.

Лабораторна робота №6. Обробка зображень

Мета і задачі:

Набути навиків використання інструментів Image Processing Toolbox в середовищі Matlab при дослідженні функцій оброблення зображень. Навчитися створювати користувацький інтерфейс в середовищі Matlab для роботи з вхідними зображеннями.

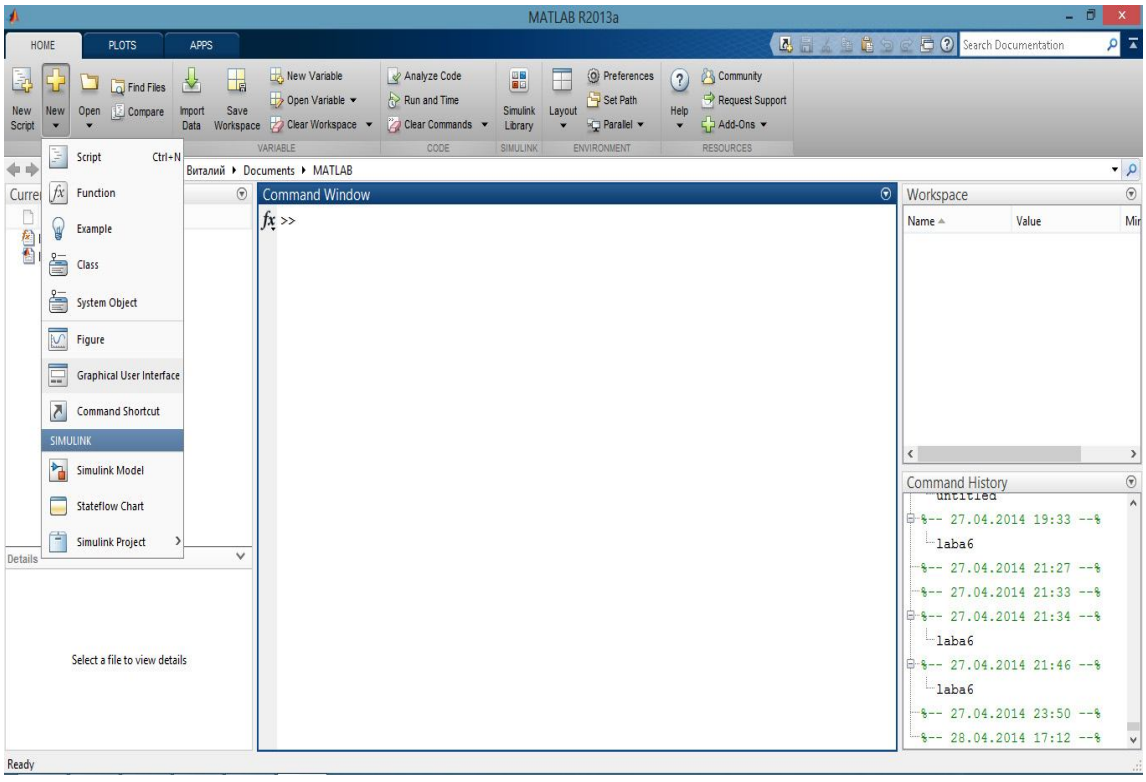
Теоретичні відомості і методичні вказівки

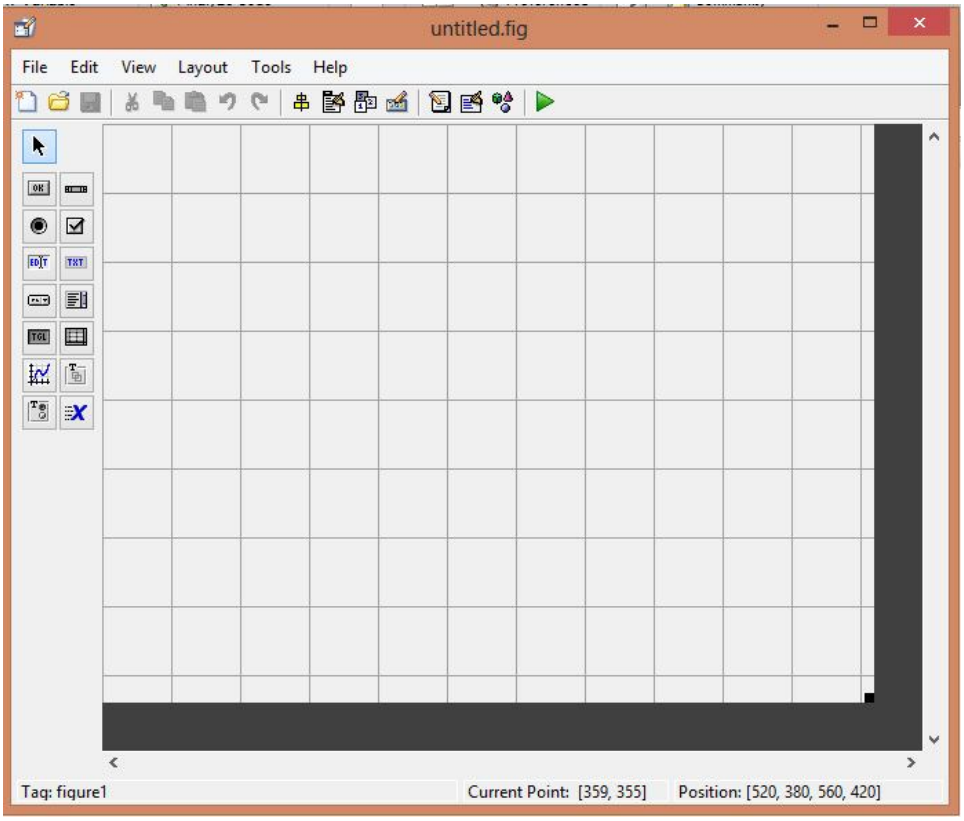
Обробка зображень – будь-яка форма обробки інформації, для якої вхідні дані представлені зображенням, наприклад, фотографіями. Обробляння зображень може здійснюватися для отримання зображення на виході (наприклад, підготовка до поліграфічного тиражування, до телетрансляції і т.д.).

Будь-яка обробка зображень включає в себе процес переходу початкового зображення через фільтр функціональної необхідності. Кожен функціонал програми зручно відображати окремою кнопкою та потрібними вхідними даними з текстового поля.

Основні кроки роботи з Matlab:

1. Запускаємо Matlab.
2. Створюємо новий графічний інтерфейс:

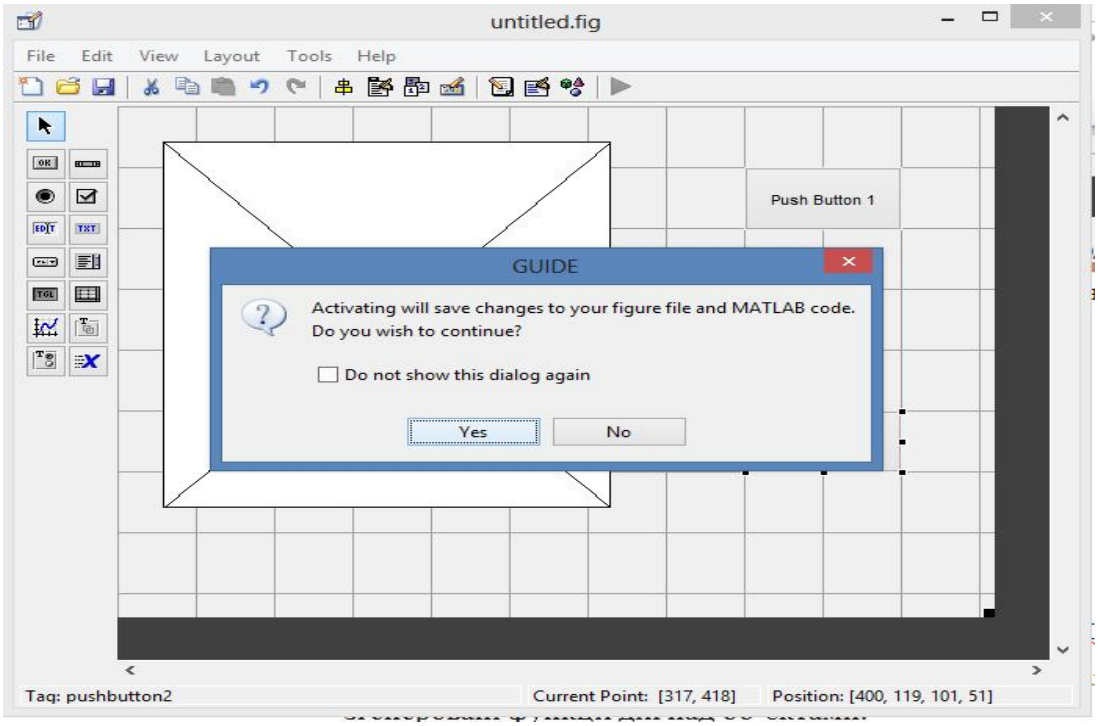


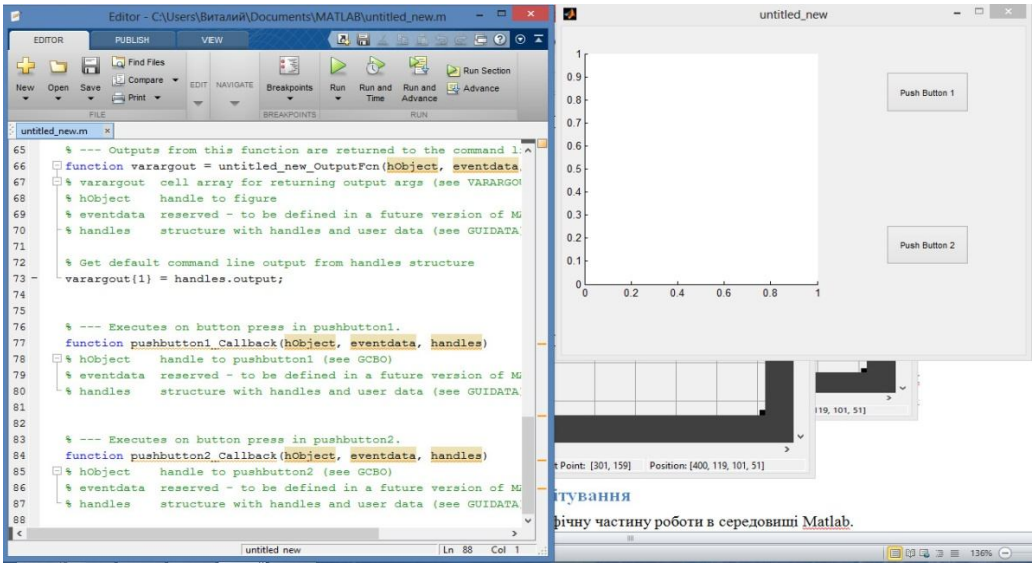


Використовуючи набір об’єктів переносимо на полотно необхідні елементи:

- axes1 – для відображення зображення;
- Push Button 1 – для завантаження графічного файлу;
- Push Button 2 – для обробки.

1. Запускаємо на перевірку:





В function pushbutton1_Callback(hObject, eventdata, handles) запишемо код для зображення його на об’єкт axes1:

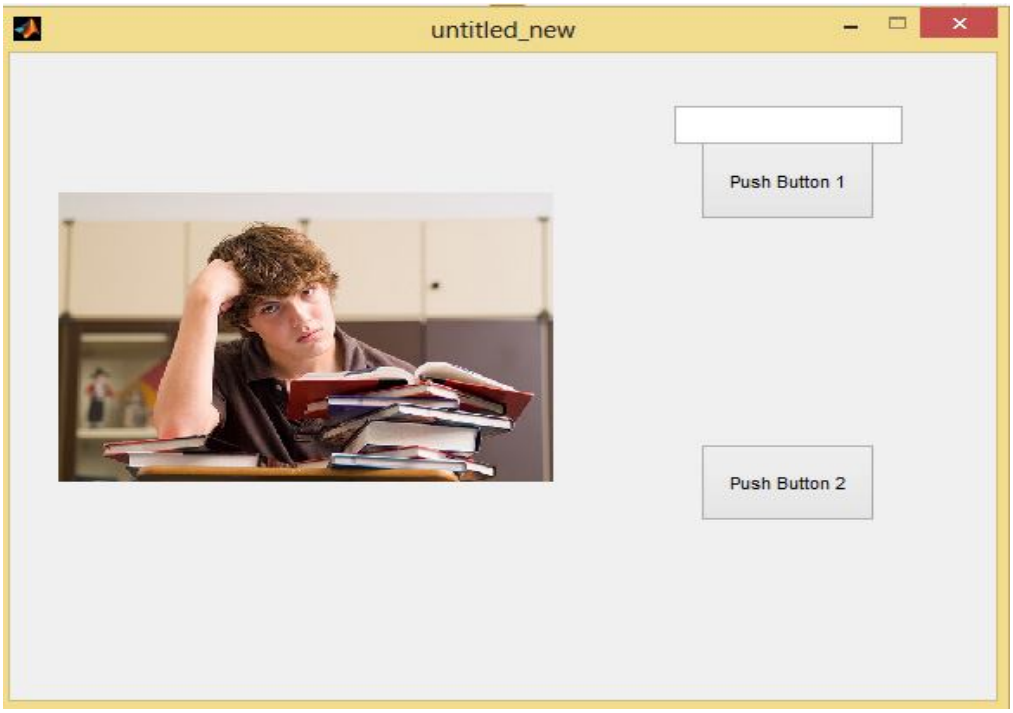
```
axes( handles.axes1)
A=get(handles.edit1,'string');
imshow(A)
```

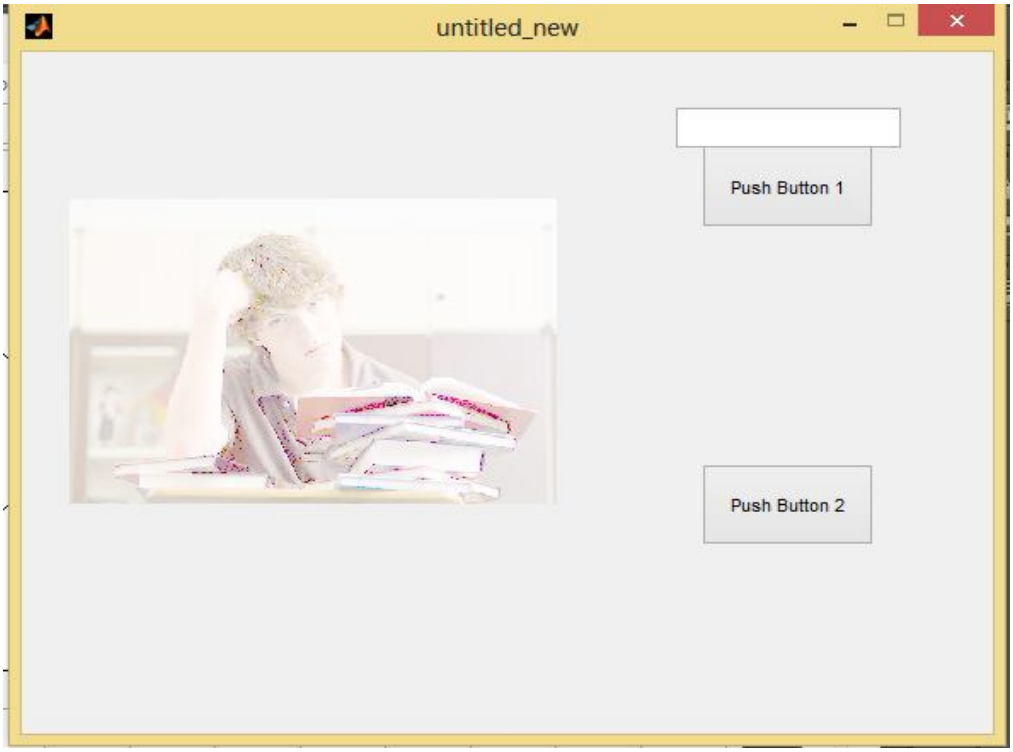
handles.edit1 має містити в собі подібну адресу:
C:\Users\chelovek_anegdot\Desktop\laba6\work_met\IMG.jpg

В function pushbutton2_Callback(hObject, eventdata, handles) запишемо код для обробки та виведення зображення:

```
axes( handles.axes1)
A=get(handles.edit1,'string');
I=imread(A)
J = imadjust(I,[],[],0.1);
imshow(J)
```

Результат компілювання:





Порядок виконання і звітування

1. Спроектувати графічну частину роботи в середовищі Matlab.
2. Написати програму шляхом додавання до основного файлу з кодом автоматично згенеровані функції дій над об’єктами.
3. Заповнити функції відповідно до завдання.
4. Відкомпілювати та відлагодити програму.
5. Протестувати роботу програми використовуючи різні вхідні формати та розміри зображень.
6. Відповісти на контрольні запитання.
7. Зробити висновки.

8. Звіт по лабораторній роботі має складатися з титульної сторінки, мети , лістингів програм, висновків по роботі.

Варіанти завдань

Варіант 1.

Виділення кордонів зображення. Фільтр Канні.
Знаходження об'єктів з допомогою сегментації зображень.

Варіант 2.

Виділення кордонів зображення. Фільтр Собеля.
Підвищення контрастності зображення. Покращення півтонових зображень.

Варіант 3.

Гамма корекція зображення.
Зменшення кількості градацій кольорових зображень.

Варіант 4

Додавання шуму "сіль і перець". Медіанна фільтрація зображення.
Підвищення контрастності зображення. Покращення кольорових зображень.

Варіант 5.

Додавання гаусівського шуму. Вінеровський фільтр.
Покращення візуальної якості зображення з використанням методу Канні.

Варіант 6.

Додавання шуму. Лінійна фільтрація. Фільтр Гауса.
Текстурна сегментація з додаванням текстурних фільтрів.

Підсумок

Після виконання лабораторної роботи студент повинен вміти створювати програми в середовищі Matlab з функціями обробки зображень.

Контрольні питання

1. Що таке обробка зображень?
2. Які є види обробки зображень ?
3. Що є основним принципом обробки зображень?

4. Що представляє з себе графічне середовище Matlab ?
5. Як пов’язується графічна оболонка та код програми?
6. Що таке фільтр зображення ?
7. Яким чином відбувається відображення зображення?

Контрольні вправи

1. Напишіть функцію збереження зображення.
2. Напишіть функцію конвертації зображення в менш контрастне.
3. Написати функцію, яка обчислює максимальну глибину вибраного кольору.
4. Написати функцію з випадковими параметрами, які впливають на часову зміну зображення.

Джерела інформації

1. http://www.mathworks.com
2. http://posibnyky.vntu.edu.ua/k_m/t2/213..htm
3. http://uk.wikipedia.org/wiki/Обробказображень