Лабораторні роботи

за курсом «Алгоритми та структури даних»

Уклали: д.т.н., проф. кафедри ОТ Анатолій Михайлович Сергієнко

ас. кафедри ОТ Анастасія Анатоліївна Сергієнко

1. Програмування графічного вікна

1.1 Принципи функціонування вікон

В операційній системі (ОС) Windows багатозадачність реалізується як паралельні процеси та потоки. Будь-який застосунок Windows після запуску реалізується як процес. Тут процес представляє собою програмний код, що виконується разом з виділеними системними ресурсами.

Під час запуску процеса система створює основний потік (thread), який виконує код програми з даними в адресному просторі процеса. З основного потоку можуть бути запущені вторинні потоки, які виконуються одночасно з основним потоком. Вікно, що відображається на дисплеї комп'ютера, є результатом виконання певного потоку.

У програми, яка написана для Windows, немає прямого доступу до апаратної частини такого пристрою відображення, як екран. Замість цього вона викликає функції графічної підсистеми, яка називається графічним інтерфейсом пристрою (Graphics Device Interface, GDI).

Функції GDI реалізують графічні операції за допомогою виклику програмних драйверів апаратних пристроїв. Реалізація певної операції є різною для пристроїв різного типу. Але вона прихована від програміста, що спрощує розробку застосунку. Отже, застосунки, які написані з викликами функцій GDI, будуть правильно виконуватись з будь-яким типом дисплея, для якого встановлений драйвер Windows.

Інтерфейси функцій GDI перелічені у заголовочних файлах Windows, головним серед яких є windows.h. В ньому є посилання на інші заголовочні файли.

Функції GDI викликаються з DLL-бібліотеки. Вони реалізуються так само, як функції, що програмуються користувачем, але відміна полягає в тому, що зв'язування коду з функціями відкладається під час компіляції та реалізується під час виконання програми (динамічне зв'язування). Більша частина цих бібліотек розміщена в підкаталозі System32.

Вікно — це базовий об'єкт, з яким мають справу як ОС, так і програміст, а також користувач застосунку. Вікно одержує інформацію від клавіатури чи миші користувача та виводить графічну інформацію на екран.

З вікном взаємодіють різноманітні графічні об'єкти, які застосовуються для малювання, такі як пера, пензлі, шрифти, палітри та інші. Незалежно від свого типу, кожен об'єкт у Windows ідентифікується своїм дескриптором (handle1). Дескриптор — це посилання на объект, яке крім його адреси вміщує додаткову інформацію, яка необхідна для керування ним. Усі взаємодії програмного коду з об'єктом виконуються лише через його дескриптор.

Програма користувача взаємодіє з ОС через повідомлення. Повідомлення повідомляє ОС, що сталась певна подія, на яку має зреагувати ОС особливим чином. Такою подією є, наприклад, сигнал від миші чи клавіатури.

Як правило, повідомлення оформлене як структура даних, яка вміщує дескриптор вікна, якому воно адресоване, код повідомлення, додаткову інформацію, як наприклад, координати пікселів вікна.

Повідомлення від кількох джерел, що стосуються одного вікна, направляються у системну чергу повідомлень ОС. Windows періодично опитує цю чергу і направляє чергове повідомлення відповідному адресату, який заданий у дескрипторі (рис. 1).

У кожного віконного застосунку завжди є головна функція WinMain(), яка виконує роль функції main(). В ній програмуються функції для ініціалізації та створення вікна, а також цикл обробки повідомлень і насамкінець, код для закінчення застосунку.

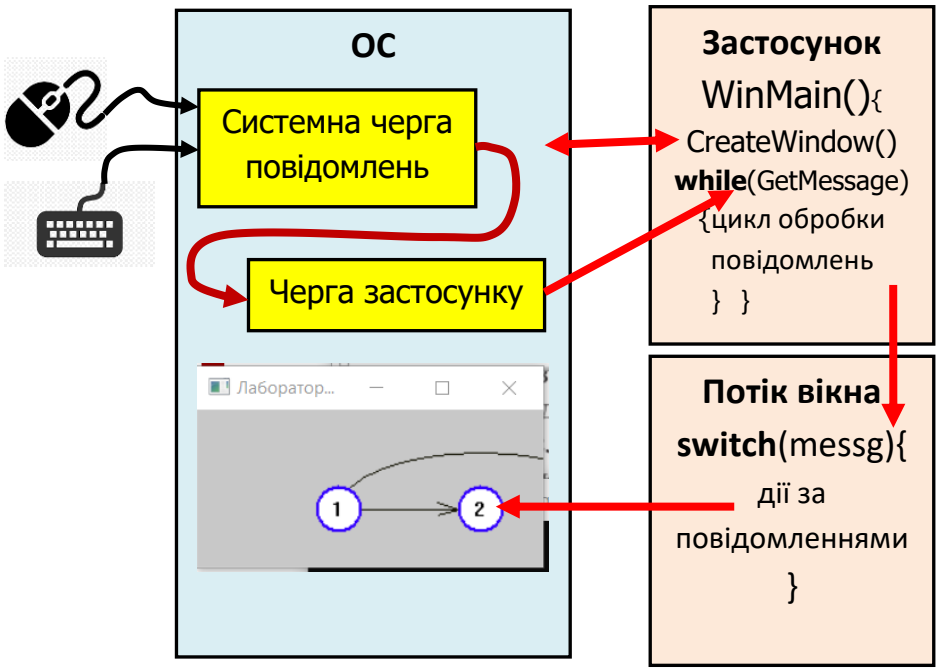

Рис. 1. Взаємодія ОС і застосунку під час графічних операцій

Як правило, в програмі є цикл одержання повідомлення з черги за допомогою функції GetMessage(). Повідомлення, які одержані цією функцією, передаються до циклу обробки повідомлень, який реалізується у окремому потоці, який викликається як функція.

Якщо чергове повідомлення має код WM_DESTROY, то за ним з потоку у ОС передається синхронізуюче повідомлення про завершення роботи з вікном функцією PostQuitMessage(0), виконується вихід з циклу і завершення програми.

1.2 Програмування графічного вікна

Розлянемо простий приклад програмування креслення графа у графічному вікні. Під час опису прикладу приводяться рядки програми з їхнім описом у тому самому порядку, в якому вони стоять у тексті програми. Про

подробиці реалізації структур та опції їх параметрів можна довідатись у керівних матеріалах, довідниках та підручниках по інтерфейсу Win32 API.

Спочатку підключається бібліотека для роботи з застосунками Windows та бібліотека математичних функцій.

```
#include<windows.h>
#include<math.h>
```

Створюється прототип функції потоку вікна, яка буде визначена в кінці програми і буде неявно запускатись з головної функції.

```
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
```

Об'являється рядок — ім'я програми

```
char ProgName[]="Лабораторна робота 3";
```

Викликається головна функція — повертає структуру типу WINAPI.

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE
hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{
```

Спочатку створюється екземпляр w структури WNDCLASS.

```
WNDCLASS w;
```

Ця структура вміщує наступні атрибути вікна.

```
w.lpszClassName=ProgName; //ім'я програми
w.hInstance=hInstance;    //ідентифікатор застосунку
w.lpfnWndProc=WndProc; //вказівник на функцію вікна
w.hCursor=LoadCursor(NULL, IDC_ARROW); // завантажений курсор
w.hIcon=0; //піктограми не буде
w.lpszMenuName=0; // меню не буде
 w.hbrBackground = WHITE_BRUSH;//колір фона вікна
 w.style=CS_HREDRAW|CS_VREDRAW; //стиль: можна
                                        // перемальовувати
   w.cbClsExtra=0; //к-ть додаткових байтів для цього класу
```

```
   w.cbWndExtra=0;//
```
Якщо структуру вікна ОС не зареєструвала, то програма закінчується.
```
if(!RegisterClass(&w))
        return 0;
```
Створюється екземпляр вікна в пам'яті та змінна типу повідомлення
```
HWND hWnd;
MSG lpMsg;
```
Вікно заповнюється параметрами та даними
```
hWnd=CreateWindow(ProgName, //Ім'я програми
        "Лабораторна робота 3. Виконав А.М.Сергієнко", //Заголовок
        WS_OVERLAPPEDWINDOW, //Стиль вікна - комплексний
        100,        //положення верхнього кута вікна на екрані по x
        100,        // положення верхнього кута вікна на екрані по y
        460,                //ширина вікна
        240,                //висота вікна
        (HWND)NULL,         // ідентифікатор породжуючого вікна
        (HMENU)NULL,        //ідентифікатор меню (немає)
        (HINSTANCE)hInstance, //ідентифікатор екземпляра вікна
        (HINSTANCE)NULL);     // додаткові параметри відсутні
```
Вікно виводиться на екран
```
   ShowWindow(hWnd, nCmdShow);
```
Організується цикл читання повідомлень з черги
```
  while(GetMessage(&lpMsg, hWnd, 0, 0)) { // повідомлення з черги
          TranslateMessage(&lpMsg); //перетворення повідомлення
                                    // у рядок
          DispatchMessage(&lpMsg); //Передача повідомлення
                                    //до функції вікна
        }
```

```
    return(lpMsg.wParam); // кінець основної функції
    }
```

Опис функції вікна — потоку, який власне, креслить граф — є наступний.

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT messg,
                         WPARAM wParam, LPARAM lParam)
    {
```

Створюється контекст вікна та екземпляр структури графічного виводу

```
HDC hdc;                // контекст
PAINTSTRUCT ps;         // екземпляр структури
```

Описується допоміжна функція, яка виводить стрілочку ребра графа.

```
void arrow(float fi, int px,int py){
        fi = 3.1416*(180.0 - fi)/180.0;
       int lx,ly,rx,ry;
        lx = px+15*cos(fi+0.3);
        rx = px+15*cos(fi-0.3);
        ly = py+15*sin(fi+0.3);
        ry = py+15*sin(fi-0.3);
        MoveToEx(hdc, lx, ly, NULL);
        LineTo(hdc, px, py);
        LineTo(hdc, rx, ry);
        return 0;
     }
```

Ця функція вимальовує два вектори так, як показано на рис.2.

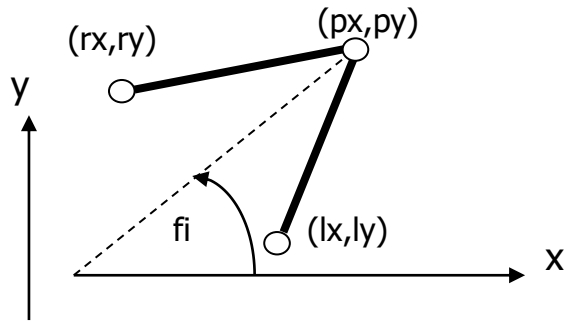

Рис. 2. Графічне представлення кінця стрілки за функцією arrow()

Кут напрямленості стрілки fi задається в градусах і відкладається у напрямку, показаному на рис. 2. Параметри px, py задають координати кінця стрілки.

Тут функція MoveToEx(hdc, lx, ly, NULL); переставляє перо у точку вікна з дескриптором hdc з координатами lx, ly і повертає попередні координати пера, якщо останній параметр не NULL.

Функція LineTo(hdc, px, py); креслить у вікні з дескриптором hdc пряму лінію пером від точки з координатами пера до точки зі вказаними координатами px, py.

Цикл обробки повідомлень відкривається так:

```
switch(messg){
    case WM_PAINT :
```

У ньому за першою альтернативою – повідомленням WM_PAINT програмують усі дії по кресленню задуманого графічного образу у вікні з дескриптором hdc. Спочатку одержується дескриптор вікна.

```
hdc=BeginPaint(hWnd, &ps);
```

Потім задаються мітки та координати вершин графа, який слід накреслити, та декларуються допоміжні змінні.

```
hdc=BeginPaint(hWnd, &ps);
char *nn[3] = {"1", "2", "3"};
int nx[3] = {100,200,300};
int ny[3] = {100,100,100};
int dx = 16, dy = 16, dtx = 5;
int i;
```

Тут dx, dy, dtx – це параметри радіуса кола вершини та зміщення для друку в колі мітки вершини у кількості пікселів. Далі задаються пера з дескрипторами BPen, KPen, які креслять лінію типу PS_SOLID, тобто, безперервну пряму.

```
HPEN BPen = CreatePen(PS_SOLID, 2, RGB(50, 0, 255));
HPEN KPen = CreatePen(PS_SOLID, 1, RGB(20, 20, 5));
```

Тут цифри 2, 1 означають товщину лінії у пікселах. Параметри RGB(50, 0, 255), RGB(20, 20, 5) задають синій та чорний колір пера, відповідно.

Задається активне перо KPen.

```
SelectObject(hdc, KPen);
```

Креслиться перша напрямлена дуга графа, використовуючи координати 0-ї та 1-ї вершин.

```
MoveToEx(hdc, nx[0], ny[0], NULL);
LineTo(hdc, nx[1], ny[1]);
arrow(0,nx[1]-dx,ny[1]);
```

Креслиться друга напрямлена дуга графа, використовуючи координати 0-ї та 2-ї вершин.

```
Arc(hdc, nx[0], ny[0]-40, nx[2], ny[2]+40, nx[2], ny[2], nx[0], ny[0]);
arrow(-45.0,nx[2]-dx*0.5,ny[2]-dy*0.8);
```

Функція Arc креслить дугу у вікні hdc. Чотири наступні її параметри задають координати верхнього лівого кута та нижнього правого кута прямокутника, в який вписаний еліпс, частиною якого є дуга. Решта параметрів задають координати початкової та кінечної точок дуги.

Далі у циклі кресляться три кола вершин пером BPen.

```
SelectObject(hdc, BPen);
for(i=0;i<=2;i++){
        Ellipse(hdc, nx[i]-dx,ny[i]-dy,nx[i]+dx,ny[i]+dy);
        TextOut(hdc, nx[i]-dtx,ny[i]-dy/2,  nn[i],1);
      }
```

Тут функція Ellipse креслить коло (еліпс). Її параметри задають координати верхнього лівого кута та нижнього правого кута прямокутника, в який

вписаний еліпс. Функція TextOut виводить текст nn[i] у рядок, з заданими початковими координатами. Її останній параметр (1) дорівнює кількості символів, які виводяться.

Нарешті, процес малювання закінчується.

```
EndPaint(hWnd, &ps);
break;
```

Іншою альтернативою є видалення вікна, про що ОС повідомляє сигналом WM_DESTROY. Видалення виконується ОС коли користувач активує кнопку закриття вікна.

```
case WM_DESTROY:
        PostQuitMessage(0);
        break;
```

При цьому програма повертає повідомлення 0 про нормальне завершення.

У решті випадків у черзі повідомлень залишаються нерозпізнані повідомлення, які просто видаляються з черги.

```
    default:
        return(DefWindowProc(hWnd, messg, wParam, lParam));
    }
}
```

У результаті виконання цієї програми одержується зображення, як на рис.3.

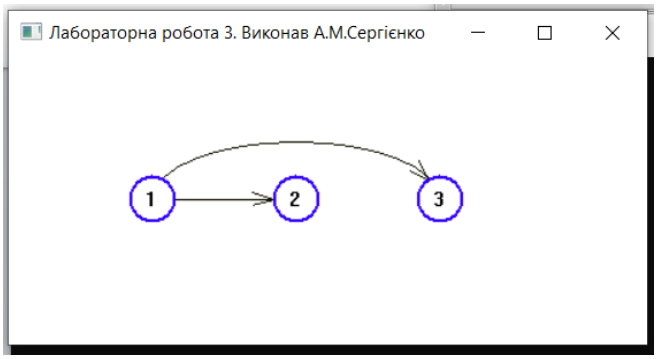

Рис.3 — Зображення графа побудоване функціями з пакету Windows

У лабораторних роботах задіяні випадкові числа. Для генерації таких чисел використовуються функції rand() та srand(). Функція rand() генерує цілі числа в діапазоні від 0 до RAND_MAX, де RAND_MAX – це константа, яка задана в бібліотеці <cstdlib>.

Для одержання початкового випадкового значення функції rand() слід перед нею викликати функцію srand(seed), де seed — випадкове натуральне число, яке необхідне для запуску генератора випадкових чисел у функції rand().

Для одержання випадкового числа у діапазоні (0.0 – 1.0) результат функції rand() слід поділити на системну константу RAND_MAX.

2 Лабораторні роботи

2.1 Лабораторна робота 3. Графічне представлення графів

Мета лабораторної роботи

Метою лабораторної роботи №3. «Графічне представлення графів» є набуття практичних навичок представлення графів у комп'ютері та ознайомлення з принципами роботи ОС.

Постановка задачі

1. Представити у програмі напрямлений і ненапрямлений графи з заданими параметрами:

— число вершин n;

— розміщення вершин;

— матриця суміжності А.

Параметри задаються на основі номера групи, представленого десятковими цифрами n_1, n_2 та номера студента у списку групи — десяткового числа n_3, n_4.

Число вершин n дорівнює $10 + n_3$.

Розміщення вершин:

— колом при $n_4 = 0,1$;

— прямокутником (квадратом) при $n_4 = 2,3$;

— трикутником при $n_4 = 4,5$;

— колом з вершиною в центрі при $n_4 = 6,7$;

— прямокутником (квадратом) з вершиною в центрі при $n_4 = 8,9$.

Наприклад, при $n_4 = 10$ розміщення вершин прямокутником з вершиною в центрі повинно виглядати так, як на прикладі графа рис.4.

Матриця А напрямленого графа за варіантом формується за функціями:

srand(n_1 n_2 n_3 n_4);

T = randm(n,n);

A = mulmr((1.0 - n_3*0.02 - n_4*0.005 - 0.25),T);

де randm(n,n) – розроблена функція, яка формує матрицю розміром n·n, що складається з випадкових чисел у діапазоні (0, 2.0);

mulmr() — розроблена функція множення матриці на коефіцієнт та округлення результату до 0 чи 1 (0, якщо результат менший за 1.0 і 1 — якщо більший за 1.0).

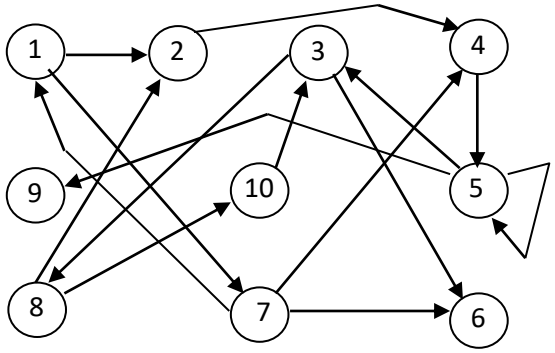

Рис.4 — Приклад зображення графа

2. Створити програму для формування зображення напрямленого і ненапрямленого графів у графічному вікні.

Зміст звіту

1. Загальна постановка задачі та завдання для конкретного варіанту.

2. Текст програми мовою С.

3. Згенеровані матриці суміжності напямленого і ненапрямленого графів.

4. Скриншоти напрямленого і ненапрямленого графів, які побудовані за варіантом.

2.2 Лабораторна робота 4. Характеристики та зв'язність графа.

Мета лабораторної роботи

Метою лабораторної роботи №4. «Характеристики та зв'язність графа» є дослідити характеристики графів та навчитись визначати їх на конкретних прикладах, вивчення методу транзитивного замикання.

Постановка задачі

1. Представити напрямлений граф з заданими параметрами так само, як у лабораторній роботі №3.

Відміна: матриця А напрямленого графа за варіантом формується за функціями:

srand(n_1 n_2 n_3 n_4);

T = randm(n,n);

A = mulmr((1.0 − n_3*0.01 − n_4*0.01 − 0.3)*T);

Перетворити граф у ненапрямлений.

2. Визначити степені вершин напрямленого і ненапрямленого графів. Програма на екран виводить степені усіх вершин ненапрямленого графу і напівстепені виходу та заходу напрямленого графу. Визначити, чи граф є однорідним та якщо так, то вказати степінь однорідності графу.

3. Визначити всі висячі та ізольовані вершини. Програма на екран виводить перелік усіх висячих та ізольованих вершин графу.

4. Змінити матрицю графу за функцією

A = mulmr((1.0 − n_3*0.005 − n_4*0.005 − 0.27)*T);

Створити програму для обчислення наступних результатів:

1) матриця суміжності;
2) півстепені вузлів;
3) всі шляхи довжини 2 і 3;
4) матриця досяжності;
5) компоненти сильної зв’язності;
6) матриця зв’язності;
7) граф конденсації.

Шляхи довжиною 2 і 3 слід шукати за матрицями A^2 і A^3, відповідно. Матриця

досяжності та компоненти сильної зв’язності слід шукати за допомогою операції транзитивного замикання.

Зміст звіту

1. Загальна постановка задачі та завдання для конкретного варіанту.

2. Текст програми мовою С.

3. Згенеровані матриці суміжності обох варіантів графів, півстепенів вузлів, шляхів довжини 2 і 3, досяжності, зв’язності, суміжності графа конденсації.

4. Таблиці степенів, напівстепенів вузлів.

5. Перелік висячих та ізольованих вершин.

6. Скриншоти заданих орієнтованого і неорієнтованого графів, модифікованого графа та його графа конденсації.

7. Висновки.

2.3 Лабораторна робота 5. Обхід графа

Мета лабораторної роботи

Метою лабораторної роботи №5. «Обхід графа» є вивчення методу дослідження графа за допомогою обходу його вершин в глибину або в ширину.

Постановка задачі

1. Представити напрямлений граф з заданими параметрами так само, як у лабораторній роботі №3. Відміна: матриця A за варіантом формується за функцією:

A = mulmr((1.0 − n_3*0.01 − n_4*0.005 − 0.15)*T);

2. Створити програми для обходу в глибину та в ширину. Обхід починати з вершини, яка має вихідні дуги. При цьому у програмі:

— встановити зупинку у точці призначення номеру черговій вершині за допомогою повідомлення про натискання кнопки,

— виводити зображення графа у графічному вікні перед кожною зупинкою.

3. Під час обходу графа побудувати дерево обходу. Вивести побудоване дерево у графічному вікні.

Зміст звіту

1. Загальна постановка задачі та завдання для конкретного варіанту.

2. Текст програми.

3. Згенерована матриця суміжності.

4. Матриця дерева обходу і матриця відповідності вершин і одержаної нумерації.

5. Скриншоти зображення графа з одержаною нумерацією та дерева обходу.

2.4 Лабораторна робота 6. Мінімальний кістяк графа

Мета лабораторної роботи

Метою лабораторної роботи №5. «Мінімальний кістяк графа» є вивчення методів розв'язання задачі знаходження мінімального кістяка графа.

Постановка задачі

1. Представити зважений ненапрямлений граф із заданими параметрами так само, як у лабораторній роботі №1. Відміна: матриця A за варіантом формується за командами:

A = **mulmr**((1.0 - n_3*0.01 - n_4*0.005 - 0.05)*T)

Матриця ваг W формується за наступним чином:

1) Wt = **roundm**((**randm**(n,n)*100) ¤ A);

де roundm — це функція, що округляє кожен елемент матриці до найближчого цілого числа,

символ «¤» — поелементне множення;

2) одержується матриця **B**, у якій

b_{ij} = 0, якщо w_{ij} = 0,

b_{ij} = 1, якщо w_{ij} > 0, b_{ij} ∈ B, w_{ij} ∈ Wt;

3) одержується матриця **C**, у якій

c_{ij} = 1, якщо b_{ij} != b_{ji},

та c_{ij} = 0 в іншому випадку;

4) одержується матриця **D**, у якій

d_{ij} = 1, якщо b_{ij} = b_{ji} = 1,

та d_{ij} = 0 в інших випадках;

5) Wt = (C + (D ¤ Tr)) ¤ Wt;

де Tr — верхній трикутник одиничної матриці (без головної діагоналі),

+ — поелементна сума матриць;

6) одержується матриця ваг W шляхом симетризування матриці Wt.

2. Створити скрипт для Scilab для знаходження мінімального кістяка за алгоритмом Краскала при n_4 — парному і за алгоритмом Пріма — при непарному. При цьому у скрипті:

— встановити функцію **halt** у точці додавання чергового ребра до кістяка,

— виводити зображення графа у графічному вікні перед кожною зупинкою по функції **halt**.

3. Під час обходу графа побудувати дерево його кістяка. Вивести побудоване дерево у графічному вікні. При зображенні як графа, так і його кістяка, вказати ваги ребер.

Зміст звіту

1. Загальна постановка задачі та завдання для конкретного варіанту.

2. Текст програми-скрипту для Scilab.

3. Згенерована матриця суміжності та матриця суміжності ненапрямленого графу.

4. Матриця знайденого кістяка графу.

5. Скриншоти зображення графа зі вказаними вагами ребер та знайденого кістяка.