zadanie5

January 13, 2022

1 Algorytmy macierzowe - Algorytmy permutacji macierzy rzadkich

1.1 Wykonali: Robert Kazimirek, Paweł Kruczkiewicz

1. Proszę wziąć macierz rzadką w formacie ze swojego zadania 4
2. 1. Dla podmacierzy o rozmiarze <= 100 np. A[1:100][1:100] wierszy proszę zbudować i narysować graf eliminacji G0
   2. Proszę uruchomić eliminację Gaussa na tym grafie G0 i narysowaćc nowe krawędzie w G0
3. Proszę napisać i opisać kod wybranego algorytm permutacji macierzy
4. Proszę uruchomić macierz permutacji dla macierzy z punktu 2
5. 1. Proszę narysować graf eliminacji G0' dla spermutowanej małej macierz z punktu 2
   2. Proszę uruchomić eliminację Gaussa na tym grafie G0' i narysowaćc nowe krawędzie w G0
6. Proszę uruchomić swój algorytm permutacji dla całej dużej (rozmiar > 100) macierzy z zadania 4
7. Proszę porównać czasy rzadkiej eliminacji Gaussa przed permutacją i po permutacji dla dużej macierzy (rozmiar > 100)

[1]:
```
import sys
!{sys.executable} -m pip install networkx
```

```
Requirement already satisfied: networkx in c:\users\pawel\anaconda3\lib\site-packages (2.6.3)
```

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
```

Wykorzystamy kilka funkcji zaimplementowanych w poprzednim zadaniu.

Algorytm rzadkiej eliminacji Gaussa

W miejscu, bez konwersji

[3]:
```
def sparse_elimination(A):
    n = len(A)
    for k, k_row_vals in enumerate(A):
        akk = k_row_vals[k]
        A[k] = { x: (v/akk if x >= k else v) for x, v in k_row_vals.items()}
        for j in range(k+1, n):
            j_row_vals = A[j]
            if k in j_row_vals:
                ajk = j_row_vals[k]
                A[j] = {i: (v - A[k].get(i, 0)*ajk if i >= k else v) for i, v␣
 ↪in j_row_vals.items()}
```

Z automatyczną konwersją

[4]:
```
def sparse_elimination_with_conversion(A):
    A = matrix_to_row_coord(A)
    sparse_elimination(A)
    return row_coord_to_matrix(A)
```

Zamiana macierzy na postać koordynatów

[5]:
```
def matrix_to_coordinates(A):
    x_coords, y_coords = A.nonzero()
    vals = A[x_coords, y_coords]
    return list(zip(vals, x_coords, y_coords))

def coordinates_to_row_coord(coord_matrix, n):
    row_lists = [{} for _ in range(n)]
    for v, x, y in coord_matrix:
        row_lists[y][x] = v

    return row_lists

def matrix_to_row_coord(A):
    A_coord = matrix_to_coordinates(A)
    return coordinates_to_row_coord(A_coord, A.shape[0])
```

Zamiana postaci koordynatów na macierz

[6]:
```
def row_coord_to_matrix(A):
    result = np.zeros((len(A), len(A)))
    for y, row_dict in enumerate(A):
        for x, val in row_dict.items():
            result[x][y] = val
    return result
```

Mierzenie czasu wykonania kodu

Jak w poprzednim zadaniu używamy 5 powtórzeń i uśredniamy wynik dla bardziej wierzytelnego wyniku testu.

[7]:
```
from time import time

def log_time(func, arg, message):
    number_of_tests = 5
    exec_times = []
    arg_copy = arg.copy()
    for _ in range(number_of_tests):
        t1 = time()
        func(arg_copy)
        t2 = time()
        exec_times.append(round(t2 - t1, 5))

    avg_time = round(sum(exec_times)/number_of_tests, 5)
    print(f"{message:8}: {avg_time} [s]")
    return avg_time
```

Funkcja spy

[8]:
```
def spy(matrix):
    mask = matrix == 0
    if matrix.shape[1] == 1:
        plt.plot(mask)
        plt.set_xticklabels(['', '0', '', '', '', '1'])
        plt.matshow(mask, aspect=0.001)
    else:
        plt.matshow(mask, aspect='auto')
```

Pobranie macierzy z pliku CSV

[9]:
```
def get_matrix_from_csv(csv_file):
    return np.loadtxt(open(csv_file, "rb"), delimiter=",", skiprows=0)
```

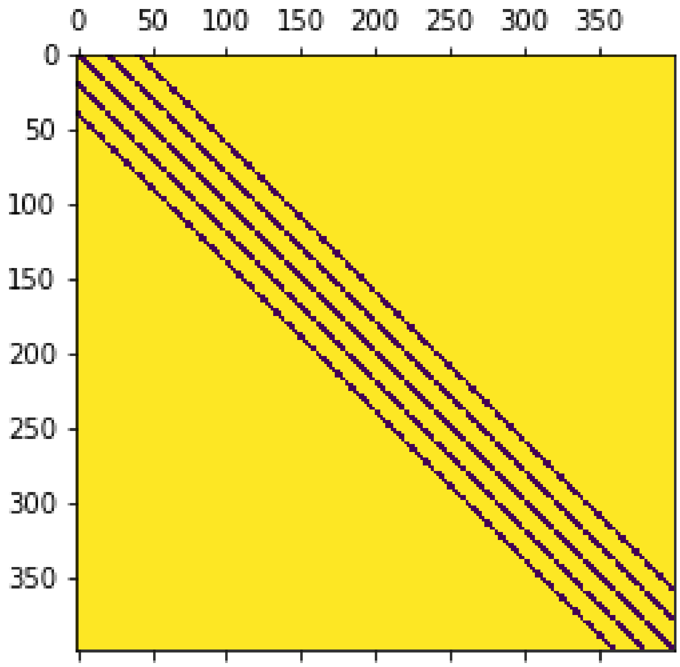
1.1.1 Ad. 1

W zaddaniu wykorzystaliśmy macierze z poprzedniego ćwiczenia.

[10]:
```
A = get_matrix_from_csv("matrices/matrix_0_18_2_0.csv")

print(f"Rozmiar macierzy A: {A.shape}")
print(f"Liczba niezerowych elementów macierzy A: {np.count_nonzero(A)}")
print(f"Procent niezerowych pól macierzy A: {100*np.count_nonzero(A)/(A.
 ↪shape[0])**2} %")
```

```
Rozmiar macierzy A: (400, 400)
Liczba niezerowych elementów macierzy A: 8836
Procent niezerowych pól macierzy A: 5.5225 %
```

[11]:
```
spy(A)
```



1.1.2 Ad. 2a

Funkcja rysująca graf na bazie macierzy

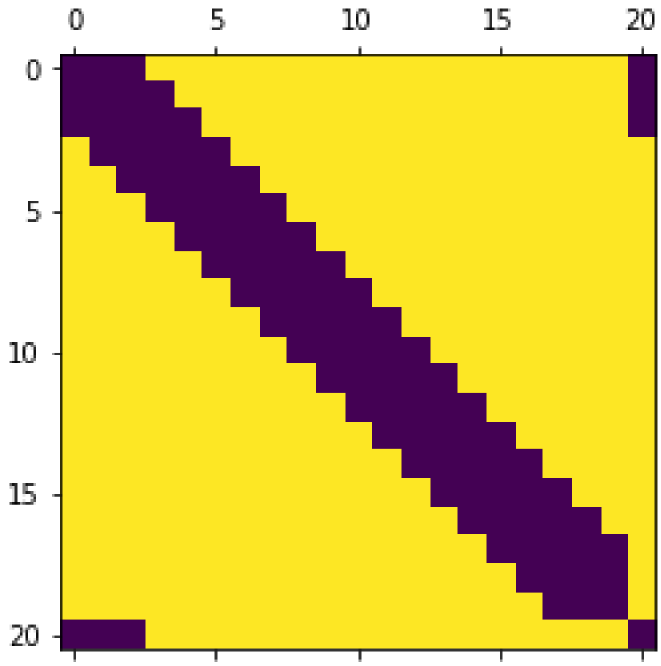
[12]:
```
def get_edges_from_matrix(A):
    A_coord = matrix_to_coordinates(A)
    edges = [(x, y) for _ , x, y in A_coord]
    return list(filter(lambda x: x[0] != x[1], edges))  # getting rid of same␣
↪vertex edges


def draw_graph_from_edges(edges):
    G = nx.Graph(edges)
    nx.draw_networkx(G)

def draw_graph(A):
    edges = get_edges_from_matrix(A)
    draw_graph_from_edges(edges)
```

Parametrem SIZE regulujemy rozmar podmacierzy

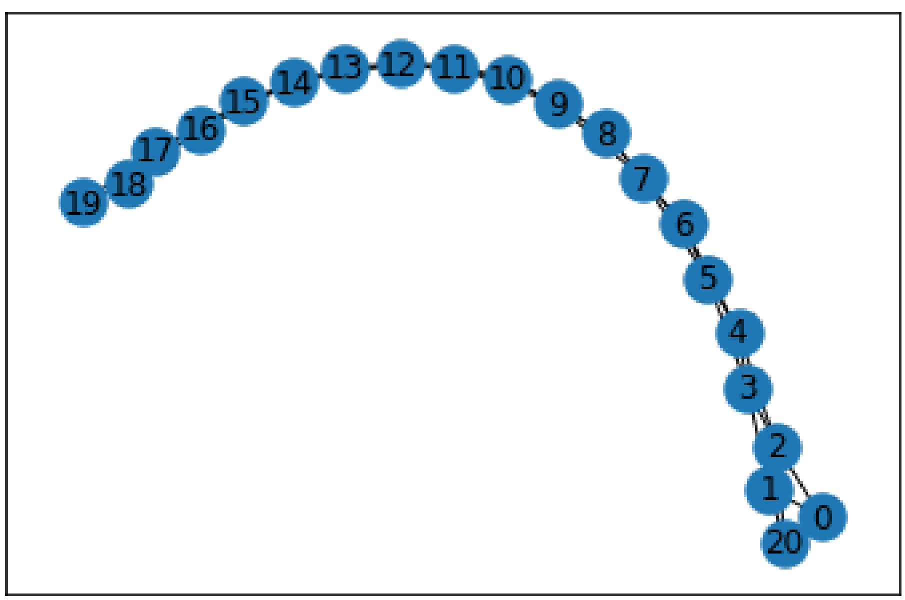
[13]:
```
SIZE = 21
sub_A = A[:SIZE, :SIZE]
```

Zdecydowaliśmy się na 21 elementów, ponieważ wybrana przez nas macierz ma własność taką, że niezerowe elementy tworzą "mozaikę" z podmacierzy 20x20. To oznacza, że mamy w rogach nienależacych do przekątnej elementy, które będą *przechodzić* o wiersz (lub kolumnę) co iterację w eliminacji Gaussa. Zatem w grafie eliminacji wierzchołek nr 20 będzie tym najczęściej dopełnianym.

[14]:
```
spy(sub_A)
```



[15]:
```
draw_graph(sub_A)
```



Funkcja przekształcająca macierz na graf

[16]:
```
def coord_to_graph(A_coord, n):
    graph = {i: set() for i in range(n)}
    for _, x, y in A_coord:
        if x != y:
            graph[x].add(y)

    return graph

def matrix_to_graph(A):
    return coord_to_graph(matrix_to_coordinates(A), A.shape[0])
```

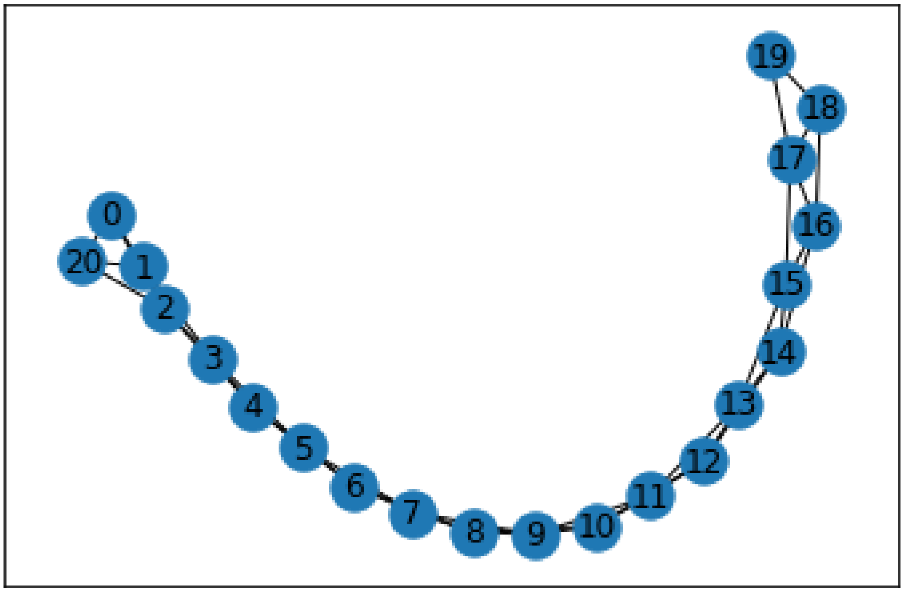
[17]:
```
sub_A_graph = matrix_to_graph(sub_A)
for v, adj_v in sub_A_graph.items():
    print(f"{v:4}: {adj_v}")
```

```
   0: {1, 2, 20}
   1: {0, 2, 3, 20}
   2: {0, 1, 3, 4, 20}
   3: {1, 2, 4, 5}
   4: {2, 3, 5, 6}
   5: {3, 4, 6, 7}
   6: {8, 4, 5, 7}
   7: {8, 9, 5, 6}
   8: {9, 10, 6, 7}
   9: {8, 10, 11, 7}
  10: {8, 9, 11, 12}
  11: {9, 10, 12, 13}
  12: {10, 11, 13, 14}
  13: {11, 12, 14, 15}
  14: {16, 12, 13, 15}
  15: {16, 17, 13, 14}
  16: {17, 18, 14, 15}
  17: {16, 18, 19, 15}
  18: {16, 17, 19}
  19: {17, 18}
  20: {0, 1, 2}
```

Funkcja rysująca graf

[18]:
```
def graph_to_edges(graph):
    return [(x, y) for x, adj_x in graph.items() for y in adj_x]

def draw_graph_from_graph_form(graph):
    edges = graph_to_edges(graph)
    draw_graph_from_edges(edges)
```

```
draw_graph_from_graph_form(sub_A_graph)
```



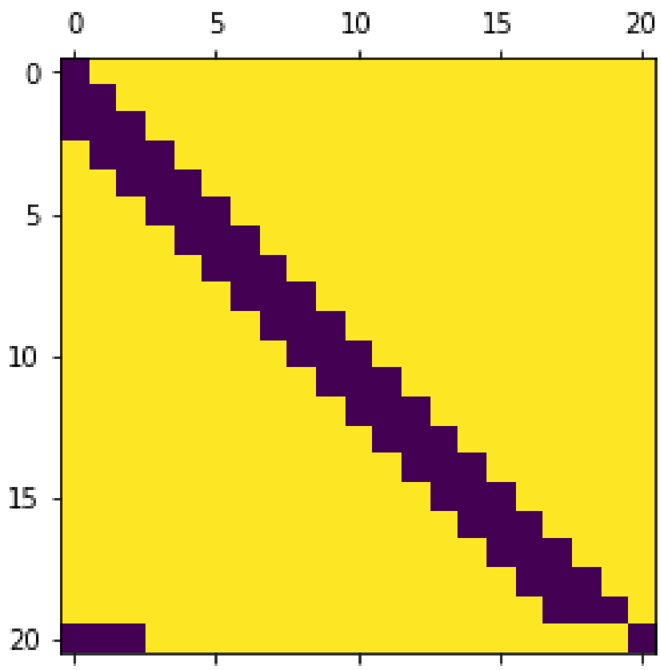
1.1.3 Ad. 2b

Funkcja obliczająca graf złożony jedynie z fill-ins

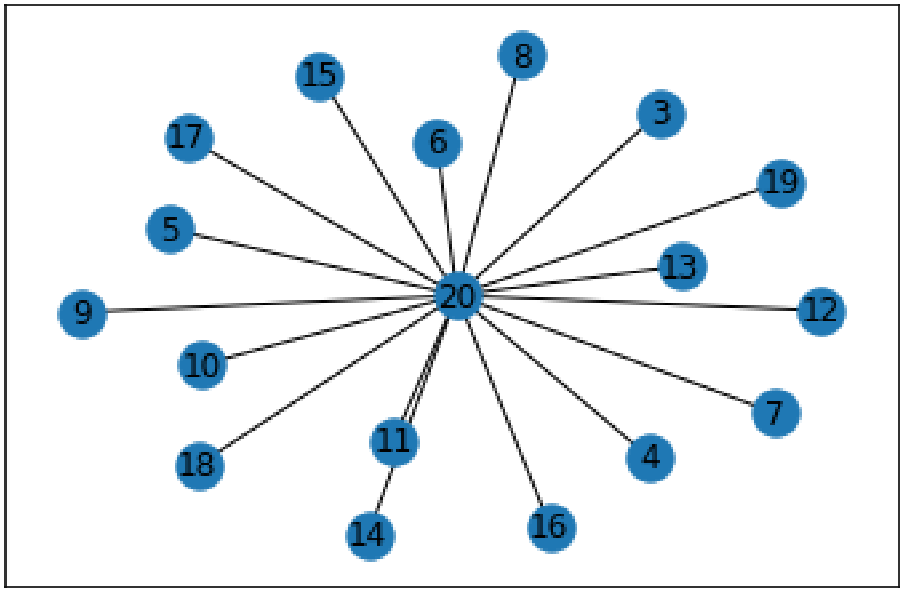
[19]:
```
def subtract_graphs(G, H):
    return {v: (G[v] - H[v] if v in H else G[v]) for v in G}


def get_fill_in_graph(A):
    def get_graph_without_edges_to_v(graph, v):
        return {u: adj_to_u - {v} for u, adj_to_u in graph.items()}

    n = A.shape[0]
    original_graph = matrix_to_graph(A)
    graph_copy = original_graph.copy()

    fill_ins = {v: set() for v in range(n)}
    for v in range(n):
        adj_to_v = graph_copy.pop(v)
        graph_copy = get_graph_without_edges_to_v(graph_copy, v)
        for u in adj_to_v:
            new_edges_adj_to_u = adj_to_v - {u}
            graph_copy[u] = graph_copy[u] | new_edges_adj_to_u
            fill_ins[u] = new_edges_adj_to_u
```

```python
    return subtract_graphs(fill_ins, original_graph)

def draw_fill_in_graph(A):
    fill_ins = get_fill_in_graph(A)
    draw_graph_from_graph_form(fill_ins)
```

[20]:
```python
sub_A_after_elimination = sparse_elimination_with_conversion(sub_A)
spy(sub_A_after_elimination)
```



[21]:
```python
draw_fill_in_graph(sub_A)
```



UWAGA powyższa funkcja rysuje graf w oparciu o postać wierzchołkową grafu, więc nie rysuje *samotnych* wierzchołków, czyli `v: deg(v) = 0`

1.1.4 Ad. 3

Zaimplementowaliśmy algorytm **minimum degree**. Składa się on u nas z 3 kroków: 1. zamiana macierzy na postać grafową 2. wywołanie `get_perm_vect(A_graph)`, które zwraca wektor permutacji 3. stworzenie macierzy permutacji P na bazie listy z pkt 2 i zwrócenie jej

Proces eliminacji dokonuje się w 2 punkcie powyższego algorytmu

[22]:
```
def get_perm_vect(graph):
    def find_min_degree_vertex(graph):
        return min(graph.items(), key=lambda x: len(x[1]))[0]

    def eliminate_vertex_from_graph(graph, v):
        for v in graph[p]:
            graph[v] = (graph[v] | graph[p]) - {p}
        graph.pop(p)

    n = len(graph)
    graph_copy = graph.copy()
    perm_vect = []
    for i in range(n):
        p = find_min_degree_vertex(graph_copy)
        eliminate_vertex_from_graph(graph_copy, p)
        perm_vect.append(p)
```

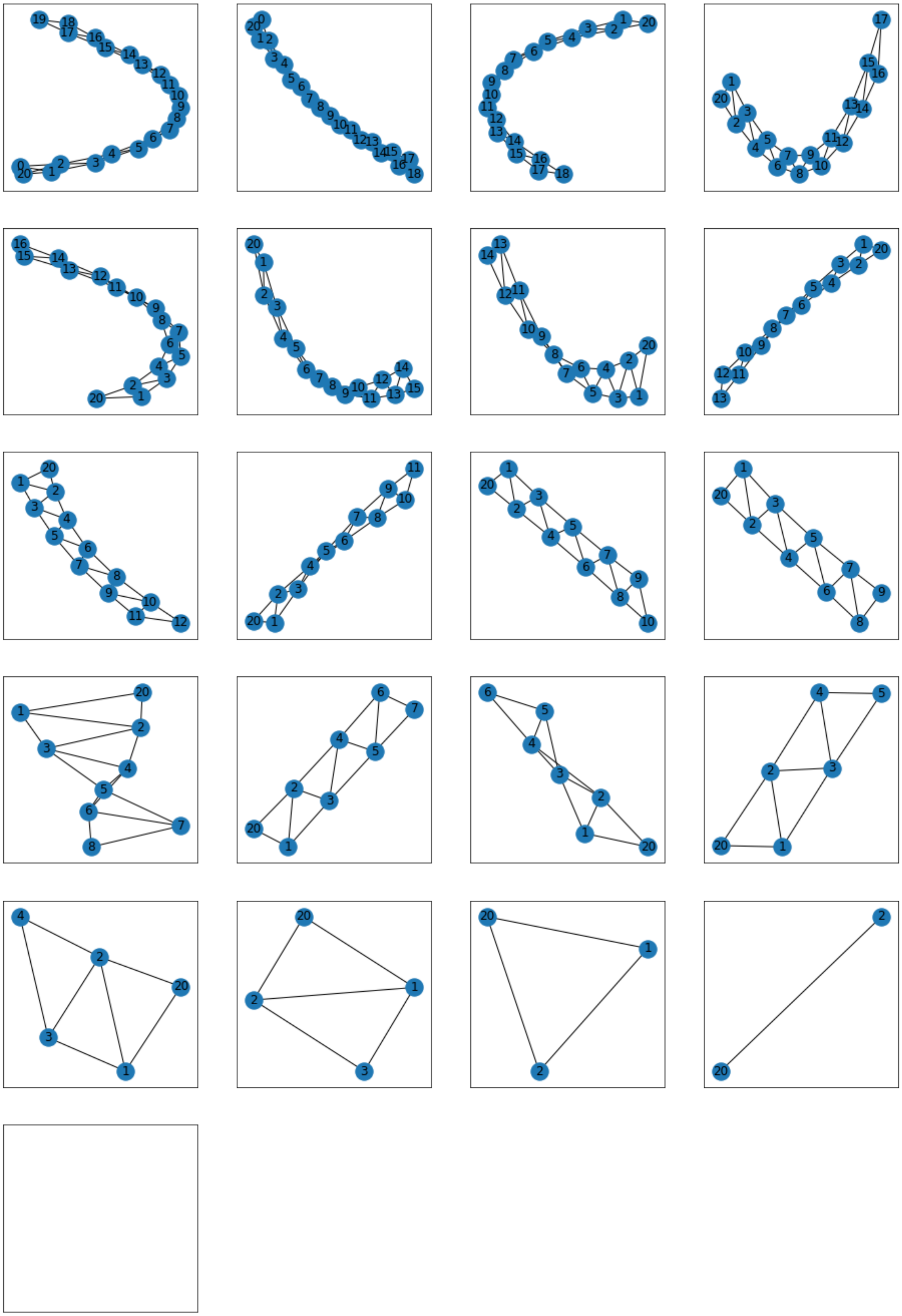
```python
    return np.array(perm_vect)


def get_perm_matrix(p_vect):
    n = p_vect.shape[0]
    I = np.eye(n)
    return I[p_vect,:]


def min_degree(A):
    A_graph = matrix_to_graph(A)
    perm_list = get_perm_vect(A_graph)
    return get_perm_matrix(perm_list)
```

[23]:
```python
get_perm_vect(sub_A_graph)
```

[23]:
```
array([19,  0, 18, 17, 16, 15, 14, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4,
        3,  1,  2, 20])
```

Wizualizacja procesu

[24]:
```python
def GO_visualize(graph):
    fig = plt.figure(figsize=(16, (SIZE//4+1)*4))
    graph = graph.copy()

    for i in range(len(graph)):
        G = nx.Graph()
        for v_1 in graph:
            for v_2 in graph[v_1]:
                if v_1 != v_2:
                    G.add_edge(v_1, v_2)

        plt.subplot(SIZE//4+1, 4, i+1)
        nx.draw_networkx(G)

        p = min(graph.items(), key=lambda x: len(x[1]))[0]
        for v in graph[p]:
            graph[v] = (graph[v] | graph[p]) - {p}
        graph.pop(p)

GO_visualize(sub_A_graph.copy())
```

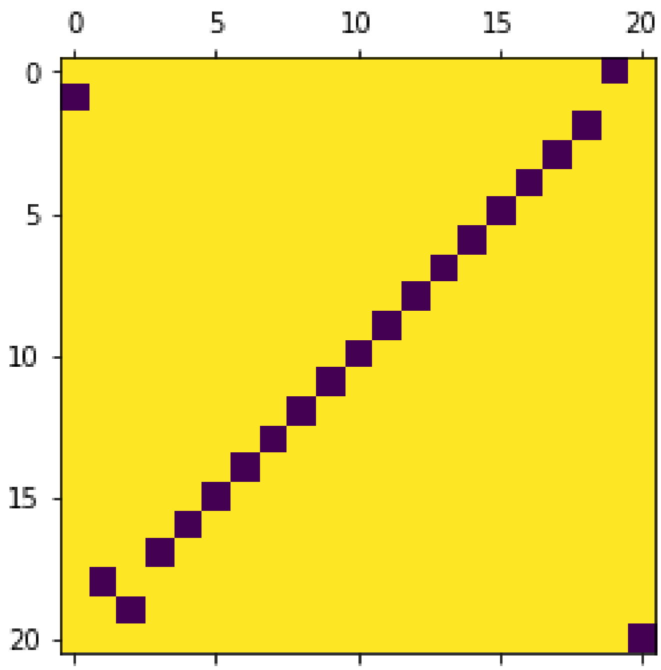


UWAGA powyższa funkcja rysuje graf w oparciu o postać wierzchołkową grafu, więc nie rysuje *samotnych* wierzchołków, czyli `v: deg(v) = 0`. Na ostatnim wykresie powinnien być wierzchołek o numerze 20

1.1.5 Ad. 4

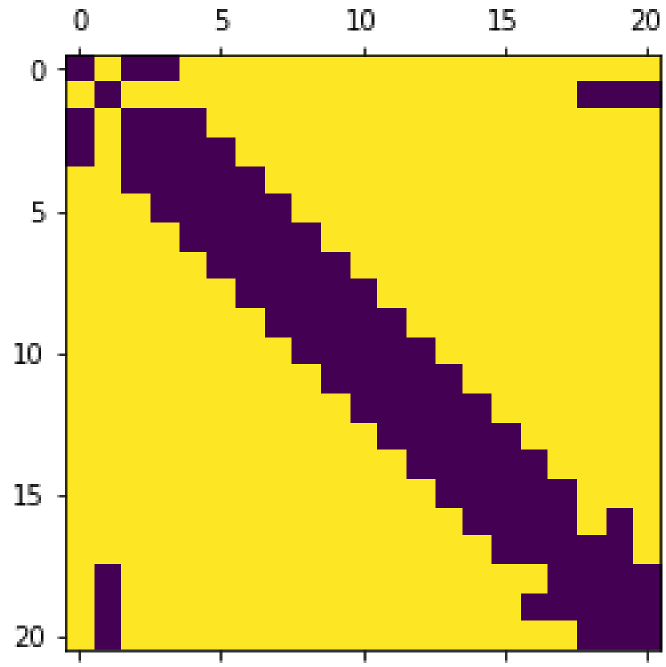
Uruchomienie pow. algorytmu na macierzy `sub_A`

[25]:
```
sub_A_perm_matrix = min_degree(sub_A)
spy(sub_A_perm_matrix)
```



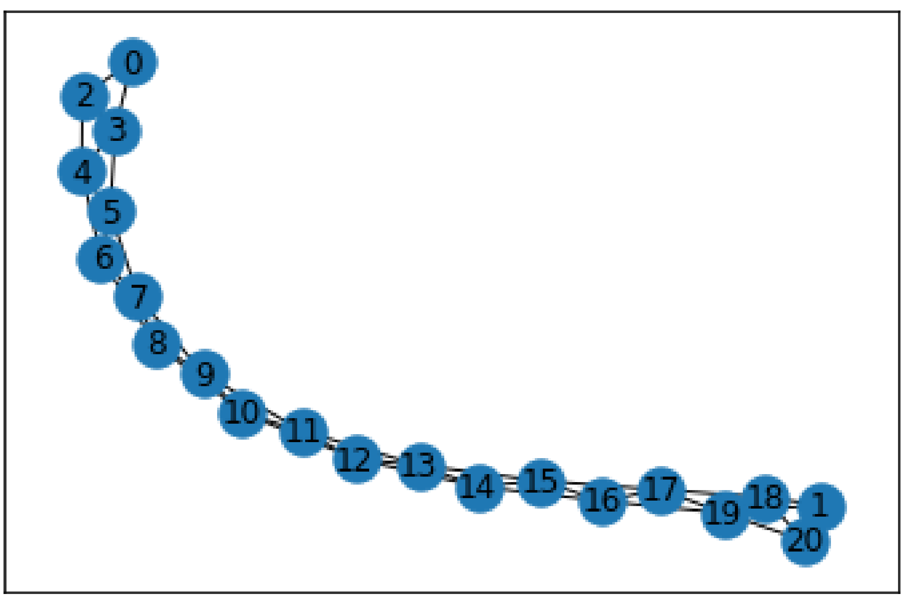
Wygląd wejściowej macierzy po spermutowaniu, czyli `A0 = PA(P^T)`

[26]:
```
def permute_matrix(A, P):
    P_T = P.transpose()
    return P@A@P_T

def permute_matrix_min_degree(A):
    P = min_degree(A)
    return permute_matrix(A, P)

sub_A0 = permute_matrix_min_degree(sub_A)
spy(sub_A0)
```



1.1.6 Ad. 5a

[27]:
```
draw_graph(sub_A0)
```



1.1.7 Ad. 5b

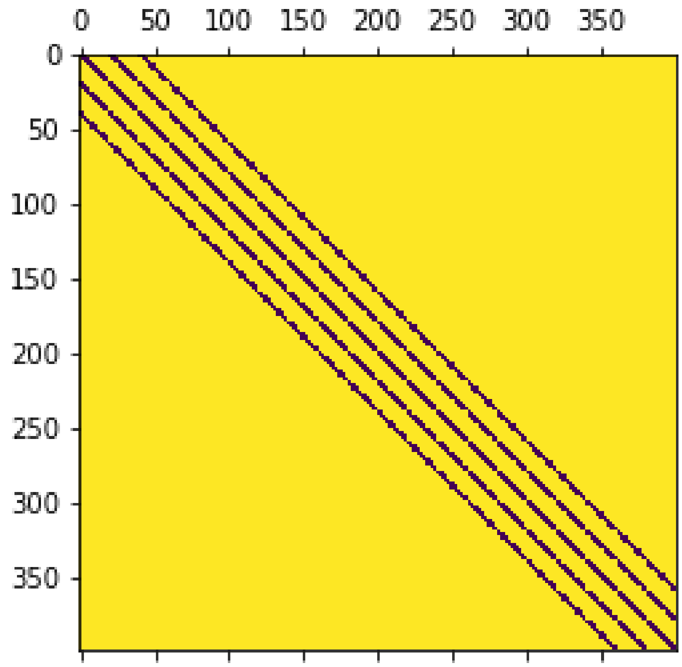
[28]:
```
draw_fill_in_graph(sub_A0)
```

Graf dodatkowych niezerowych krawędzi okazał się próżny

1.1.8 Ad. 6

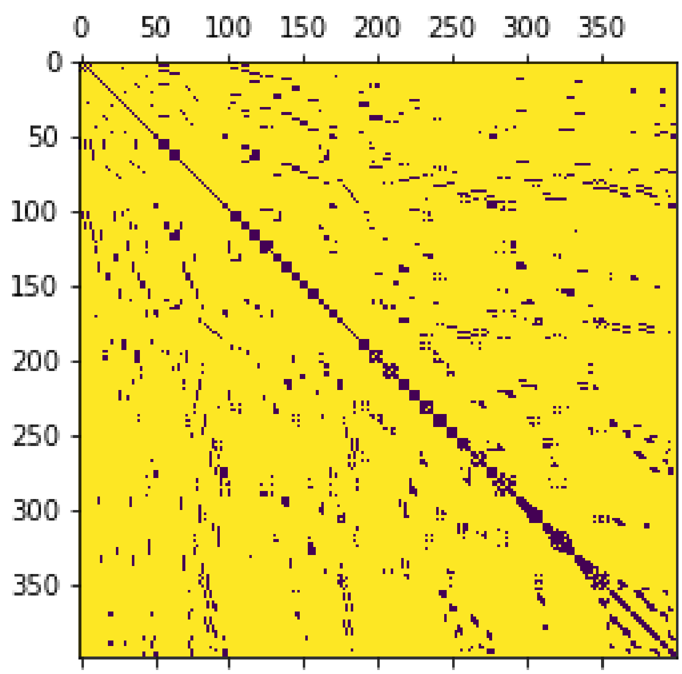
Macierz przed permutacją

[29]:
```
spy(A)
```



Macierz po permutacji

[30]:
```
A0 = permute_matrix_min_degree(A)
spy(A0)
```



1.1.9 Ad. 7

[33]:
```
def log_sparse_gauss_elim_time(A, text):
    A_row_coord = matrix_to_row_coord(A)
    return log_time(sparse_elimination, A_row_coord, text)
```

[36]:
```
G0_bef_perm = log_sparse_gauss_elim_time(A, "Czas eliminacji dla␣
 ↪niespermutowanej macierzy")
G0_aft_perm = log_sparse_gauss_elim_time(A0, "Czas eliminacji dla spermutowanej␣
 ↪macierzy")
```

```
Czas eliminacji dla niespermutowanej macierzy: 0.05733 [s]
Czas eliminacji dla spermutowanej macierzy: 0.05061 [s]
```

[37]:
```
print('macierz A')
plt.bar(['przed permutacją', 'po permutacji'],
        [G0_bef_perm, G0_aft_perm])
```

```
macierz A
```

[37]: <BarContainer object of 2 artists>