Zadanie 1

Macierz sąsiedztwa

Wada macierzy sąsiedztwa jest ze w przypadku małej liczby połączen miedzy wierzchołkami, wtedy macierz staje sie macierzą rzadką i zajmuje bardzo dużo pamięci. Zaletą jest natomiast to że możemy w łatwy sposób mieć dostęp do wartości dla poszczegolnych połączeń, ponieważ jest stały czas dostępu.

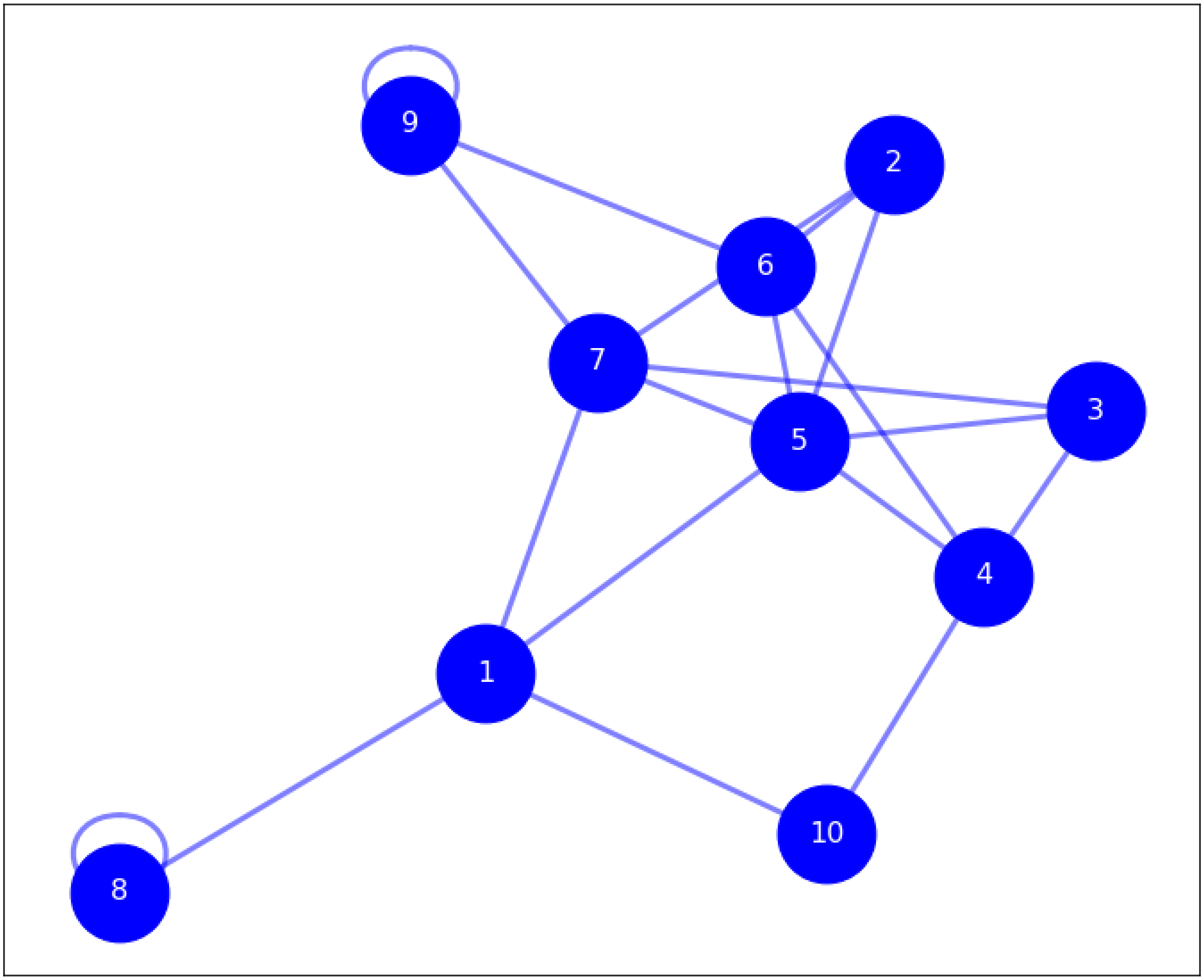
Lista sąsiedztwa

Lista sąsiedztwa w przeciwieństwie do macierzy sąsiedztwa jest używana w przypadku dużej ilości wierzchołków oraz małej ilości połączeń. Wtedy nie zapełniamy pamięci nie potrzebnymi zerami. Wadą jest natomiast to że złożoność dodania, oraz pobrania z listy sąsiedztwa jest $O(n)$.

Zadanie 2

In [3]:
```
import networkx as nx
import matplotlib.pyplot as plt
import random

# Losowy graf nieskierowany
d = {i: [random.randint(1, 10)  for _ in range(2)] for i in range(1, 11)}

G = nx.Graph(d)

fig = plt.figure(figsize=(12, 10))

pos = nx.spring_layout(G)

nx.draw_networkx_nodes(G, pos, node_color='b', node_size=3000)
nx.draw_networkx_edges(G, pos, width=3,alpha=0.5,edge_color='b')

labels = {i: str(i) for i in range(1, 11)}
nx.draw_networkx_labels(G, pos, labels=labels, font_size=16, font_color="white")

plt.title("Graf rzedu 10.")
plt.show()
```

Graf rzedu 10.



Zadanie 3

Algorytm przeszukujacy graf z zadania 2 w glab.

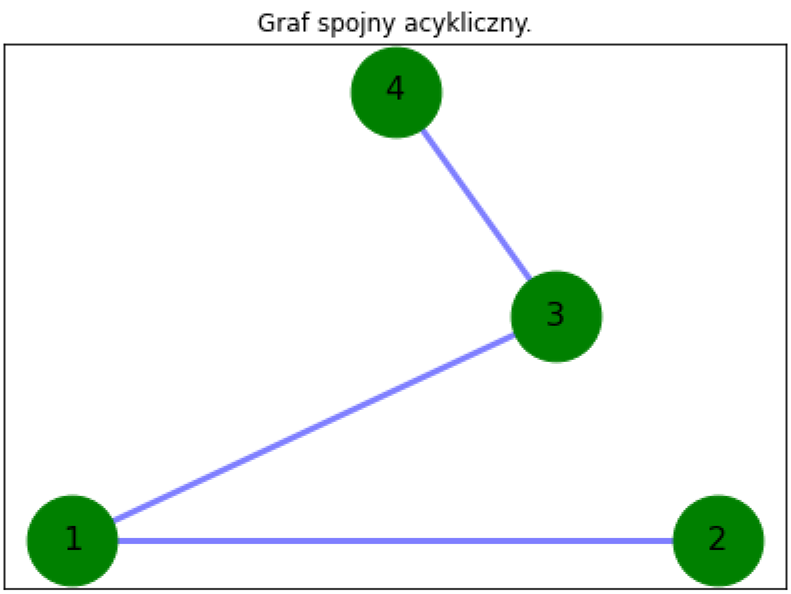
In [32]:
```
def dfs(G, node, visited=[]):
    visited.append(node)
    # Jezeli wierzcholek nie jest lisciem
    if G.get(node):
        for n in G[node]:
            # Jezeli graf jest acykliczny to kazdy kolejny wierzcholek nie bedzie wczesn
            if n not in visited:
                dfs(G, n, visited)
    return visited

dfs(d, 1)
```

Out[32]:
```
[1, 5, 10, 2, 6, 4, 7, 8]
```

Zadanie 4

Graf spojny acykliczny

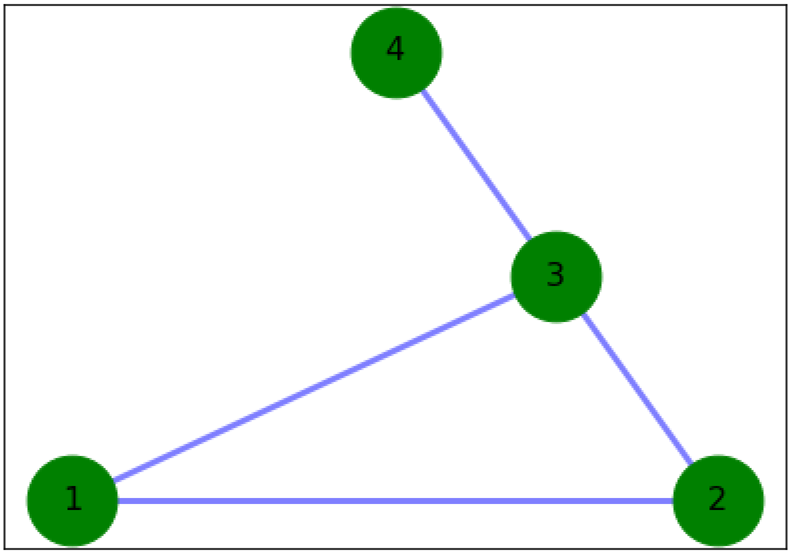
In [33]:
```
g = {1: [2, 3], 3: [4]}

G = nx.Graph(g)

fig = plt.figure(figsize=(7, 5))

pos = nx.planar_layout(G)

nx.draw_networkx_nodes(G, pos, node_color='g', node_size=2000)
nx.draw_networkx_edges(G, pos, width=3,alpha=0.5,edge_color='b')

labels = {i: str(i) for i in range(1, 5)}
nx.draw_networkx_labels(G, pos, labels=labels, font_size=16)

plt.title("Graf spojny acykliczny.")
plt.show()
```



Graf spojny z cyklami

In [4]:
```
g = {1: [2, 3], 2: [3], 3: [4]}

G = nx.Graph(g)

fig = plt.figure(figsize=(7, 5))

pos = nx.planar_layout(G)

nx.draw_networkx_nodes(G, pos, node_color='g', node_size=2000)
nx.draw_networkx_edges(G, pos, width=3,alpha=0.5,edge_color='b')

labels = {i: str(i) for i in range(1, 5)}
nx.draw_networkx_labels(G, pos, labels=labels, font_size=16)

plt.title("Graf spojny cykliczny.")
plt.show()
```

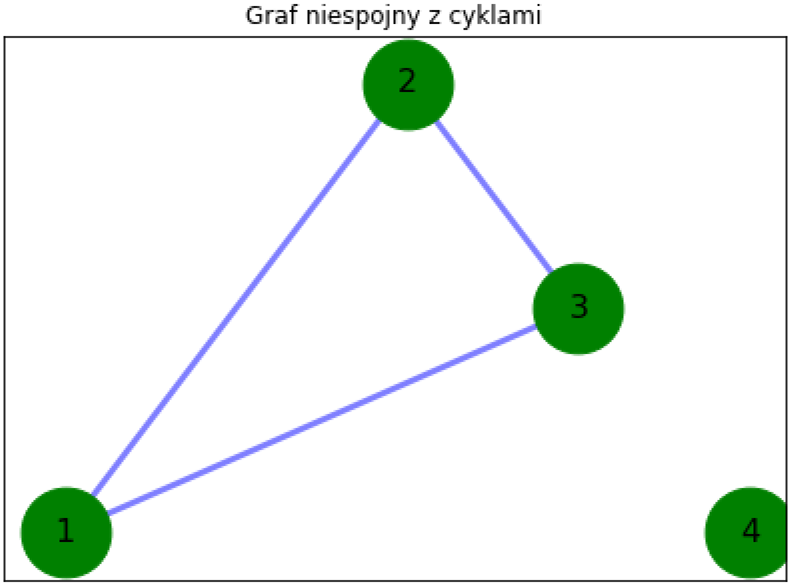
Graf spojny cykliczny.



Graf niespojny z cyklami

In [7]:
```
g = {1: [2, 3], 2: [3], 4: []}

G = nx.Graph(g)

fig = plt.figure(figsize=(7, 5))

pos = nx.planar_layout(G)

nx.draw_networkx_nodes(G, pos, node_color='g', node_size=2000)
nx.draw_networkx_edges(G, pos, width=3,alpha=0.5,edge_color='b')

labels = {i: str(i) for i in range(1, 5)}
nx.draw_networkx_labels(G, pos, labels=labels, font_size=16)

plt.title("Graf niespojny z cyklami")
plt.show()
```

Graf niespojny z cyklami



Zadanie 5

Wierzchołek rozspajający grafu

Jest to wierzchołek grafu spójnego, którego usunięcie spowoduje rozspójnienie go. Moja propozycja na znalezienie go to usuwanie po kolei wierzchołkow grafu i sprawdzanie czy graf stanie sie niespójny.

Centrum grafu

Jest to wierzchołek grafu spójnego, którego najdłuższa droga łącząca go z pozostałymi jest niewiększa od długości dróg łączących pozostałe wierzchołki. Aby znaleźć centrum grafu musimy dla każdego wierzchołka znaleźść sumę wszystkich odległości tego wierzchołka od każdego innego wierzcholka. Centrum grafu będzie wierzchołek o najmniejszej sumie.

Jakie inne wlasnosci grafu mozna badac analizujac dane i przebieg algorytmu

Możemy analizować centralność grafu. Jeżeli chodzi o przebieg algorytmu mozemy analizować jego złożoność pamieciowa i czasowa.