# Лабораторная работа номер 9, Вариант 67 Задание: Алгоритмы поиска максимальных паросочетаний

Граф задан списком рёбер:

(9, 14) (16, 15) (2, 10) (7, 16) (2, 7) (2, 4) (8, 4) (2, 14) (11, 16) (3, 15) (6, 3) (5, 9) (11, 3) (7, 3) (2, 11) (10, 12) (2, 6) (5, 12) (6, 16) (16, 13) (12, 13) (12, 6) (10, 16) (9, 7) (8, 7) (4, 3) (9, 15) (3, 14) (2, 5) (13, 2)

Задание:

1. Проверить, является ли граф двудольным. Если нет, удалить несколько (желательно наименьшее число) рёбер для получения двудольного графа.

2. Найти наибольшее паросочетание.

3. Визуализировать результат.

## Код

### А) Алгоритм Форда-Фалкерсона

from collections import defaultdict  
  
class Graph:  
 def \_\_init\_\_(self, vertices):  
 self.V = vertices  
 self.graph = defaultdict(list)  
  
 def add\_edge(self, u, v):  
 self.graph[u].append(v)  
 self.graph[v].append(u)  
  
 def bfs(self, s, t, parent):  
 visited = [False] \* (self.V)  
 queue = []  
 queue.append(s)  
 visited[s] = True  
  
 while queue:  
 u = queue.pop(0)  
 for v in self.graph[u]:  
 if not visited[v]:  
 visited[v] = True  
 parent[v] = u  
 queue.append(v)  
 if v == t:  
 return True  
 return False  
  
 def ford\_fulkerson(self, source, sink):  
 parent = [-1] \* (self.V)  
 max\_flow = 0  
  
 while self.bfs(source, sink, parent):  
 path\_flow = float("Inf")  
 s = sink  
 while(s != source):  
 path\_flow = min(path\_flow, 1) # Каждое ребро имеет пропускную способность 1  
 s = parent[s]  
  
 max\_flow += path\_flow  
  
 return max\_flow  
  
# Пример использования  
g = Graph(17) # 17 вершин (0-16)  
edges = [(9, 14), (16, 15), (2, 10), (7, 16), (2, 7), (2, 4),   
 (8, 4), (2, 14), (11, 16), (3, 15), (6, 3), (5, 9),   
 (11, 3), (7, 3), (2, 11), (10, 12), (2, 6), (5, 12),   
 (6, 16), (16, 13), (12, 13), (12, 6), (10, 16), (9, 7),   
 (8, 7), (4, 3), (9, 15), (3, 14), (2, 5), (13, 2)]  
  
for edge in edges:  
 g.add\_edge(edge[0], edge[1])  
  
source = 2  
sink = 15  
print("Максимальный поток (наибольшее паросочетание):", g.ford\_fulkerson(source, sink))

### Б) Алгоритм Хопкрофта–Карпи

class BipartiteGraph:  
 def \_\_init\_\_(self, vertices):  
 self.V = vertices  
 self.graph = defaultdict(list)  
  
 def add\_edge(self, u, v):  
 self.graph[u].append(v)  
  
 def bpm(self, u, matchR, seen):  
 for v in self.graph[u]:  
 if not seen[v]:  
 seen[v] = True  
 if matchR[v] == -1 or self.bpm(matchR[v], matchR, seen):  
 matchR[v] = u  
 return True  
 return False  
  
 def max\_bpm(self):  
 matchR = [-1] \* self.V  
 result = 0  
 for u in range(self.V):  
 seen = [False] \* self.V  
 if self.bpm(u, matchR, seen):  
 result += 1  
 return result  
  
# Пример использования  
b\_graph = BipartiteGraph(17)  
for edge in edges:  
 b\_graph.add\_edge(edge[0], edge[1])  
  
print("Максимальное паросочетание (алгоритм Хопкрофта–Карпи):", b\_graph.max\_bpm())

### В) Визуализация результатов

import matplotlib.pyplot as plt  
import networkx as nx  
  
def visualize\_graph(edges):  
 G = nx.Graph()  
 G.add\_edges\_from(edges)  
 pos = nx.spring\_layout(G)  
   
 plt.figure(figsize=(10, 10))  
 nx.draw(G, pos, with\_labels=True, node\_color='lightblue', node\_size=500, font\_size=10)  
 plt.title('Граф')  
 plt.show()  
  
visualize\_graph(edges)

**Ответы на контрольные вопросы.**

1. Какой граф называется двудольным?

Двудольный граф — это граф, вершины которого можно разделить на два disjoint множества так, что каждое ребро соединяет вершину из одного множества с вершиной из другого.

2. Дайте понятие паросочетания.

Паросочетание в графе — это набор рёбер, не имеющих общих вершин.

3. Назовите несколько (не менее 3-х) алгоритмов поиска паросочетаний и укажите их свойства.

Алгоритм Форда-Фалкерсона: базируется на концепции потоков, подходит для общего случая.

Алгоритм Хопкрофта–Карпи: эффективен для двудольных графов, имеет время O(E√V).

Алгоритм Тарьяна: применяется для нахождения максимального паросочетания в общем графе, основан на поиске в глубину.