Презентація курсової роботи на тему «Королі в турнірах»

Керівник: к.ф.-м.н. Козеренко С.О.

Виконав студент 3-го року навчання спеціальності №113 «Прикладна математика» Севергін Олександр Вадимович

1. Основні поняття

1. Турнір – це повний орієтований граф.
2. k-король (англ. k-king) у турнірі T - це така вершина $v \in T$, що має елементарний шлях довжини k або менше у будь-яку іншу вершину з турніра T.
3. Сильний король (англ. strong king) у турнірі T - це така вершина $v \in T$, що для кожної іншої вершини $u \in T$ $b(v,u) > b(u,v)$, де b(v,u) - кількість елементарних шляхів довжиною 2 з вершини v у вершину u.
4. Тензорний добуток $G \times H$ графів G і H - це граф, у якого множина вершин є декартовим добутком $V(G) \times V(H)$ і різні пари (u,u') і (v,v') є суміжними в $G \times H$ тоді й тільки тоді, коли u суміжна з v і u' суміжна з v'.

2. Королі в турнірах

2.1. Королі в турнірах Т

Теорема 2.1 [1, p.66]
Для кожного $n \in \mathbb{N}$ існує турнір на n вершинах, у якому кожна вершина є королем, тоді й тільки тоді, коли $n \neq \{2; 4\}$.

Теорема 2.2 [1, p.66]
Кожний турнір T має короля. Нехай, T - турнір, у якого для кожної вершини $v \in V$ виконується $\deg^+(v) = 0$.
А) Якщо x є королем в T, тоді T має іншого короля в $N^-(x)$.
Б) Використовуючи, частину А), турнір T має принаймні три королі.
В) Для кожної $n \geq 3$, побудуйте турнір T з $\delta^-(T) > 0$ і має тільки 3 королі.
(Коментар: На n вершинах існує турнір, що має рівно k королів, коли $n \geq k \geq 1$, за винятком випадків, коли $k = 2$ і коли $n = k = 4$)

2.2 Сильні королі в турнірах T

Теорема 2.3 [3]
Кожна вершина в турнірі з найбільшим рахунком є сильним королем.

2.3 Королі в тензорному добутку орієнтованих графів $G_1 \times G_2$

Теорема 2.4 [4, p.600]
Якщо орієнтовані графи $G_1, G_2, \dots, G_r$ - сильнозв'язні орграфи, то тензорний добуток $G_1 \times G_2 \times .. \times G_r$ матиме рівно

$$\frac{d(G_1) * d(G_2) * .. * d(G_r)}{lcm(d(G_1), d(G_2), \dots, d(G_r))}$$

сильних компонент.

Теорема 2.5 [5, p.17]
Якщо D_1 і D_2 є сильнозв'язними орграфами, то $D_1 \times D_2$ має $\gcd\big(d(D_1), d(D_2)\big)$ сильних компонент. Зокрема, $D_1 \times D_2$ є сильнозв'язним орграфом тоді й тільки тоді, коли $\gcd\big(d(D_1), d(D_2)\big) = 1$.

Теорема 2.6 [5, p.23]
Вершина (v, w) є королем в $G_1 \times G_2$ тоді й тільки тоді, коли вершина v є королем в D_1, вершина w є королем в D_2 і $\gcd\big(d(D_1), d(D_2)\big) = 1$.

3. Алгоритми

3.1. Основні функції

1. Спочатку треба імпортувати необхідні бібліотеки Python

```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import random
```

2. Перевірка чи є задана матриця суміжності турніром

```
def is_basic_tournament(adj_matrix):
    n = len(adj_matrix)
    for i in range(n):
        for j in range(n):
            if i != j and adj_matrix[i][j] + adj_matrix[j][i] != 1:
                return False
    return True
```

3. Перевірка чи є задана матриця суміжності двочастковим турніром

```python
def is_bipirtite_tournament(adj_matrix):
    n = len(adj_matrix)
    identity_matrix = [[int(i == j) for j in range(n)] for i in range(n)]
    res = True
    for i in range(n):
        for j in range(i+1, n):
            if identity_matrix[i][j] != 0 and identity_matrix[j][i] != 0:
                res = False
                break
        if not res:
            break
    return res
```

4. Піднесення матриці суміжності D^+ до заданого степеня

```python
def count_D_plus_in_degree(adj_matrix, degree):
    n = len(adj_matrix)
    D_plus_degree = adj_matrix
    for p in range(degree-1):
        D_plus_degree = D_plus_degree @ adj_matrix
    return D_plus_degree
```

5. Обчислення матриці $M = I + D^{+} + .. + (D^{+})^{n}$

```
def count_M(adj_matrix, t_as_dict, max_degree):
    D_plus = adj_matrix
    n = len(D_plus)
    I = np.identity(n)
    M = [[I[i][j] + D_plus[i][j] for j in range(n)] for i in range(n)]
    degree = 2
    while degree <= max_degree:
        D_plus_deg = count_D_plus_in_degree(D_plus, degree)
        for i in range(n):
            for j in range(n):
                M[i][j] += D_plus_deg[i][j]
        degree += 1
    return M
```

6. Малювання турніра заданого суміжною матрицею і турнір заданою класом networkx.DiGraph()

```
def draw_adj_matrix(graph, kings):
    G = nx.DiGraph(graph)
    pos = nx.spring_layout(G)
    nx.draw_networkx_nodes(G, pos, nodelist=kings, node_color='violet')
    nx.draw_networkx_nodes(G, pos, nodelist=set(graph.keys())-set(kings), node_color='aquamarine')
    nx.draw_networkx_edges(G, pos)
    nx.draw_networkx_labels(G, pos)
    plt.title("Tournament T")
    plt.axis('off')
    plt.show()
```

```
def draw_DiGraph(T, K):
    pos = nx.circular_layout(T)
    nx.draw_networkx_nodes(T, pos, nodelist=T.nodes() - K, node_color='aquamarine', node_size=500)
    nx.draw_networkx_nodes(T, pos, nodelist=K, node_color='violet', node_size=500)
    nx.draw_networkx_edges(T, pos)
    nx.draw_networkx_labels(T, pos)
    plt.axis('off')
    plt.show()
```

3.2. Алгоритми на турнірах T

1. Алгоритм пошуку короля x в турнірі T [1, p.65]

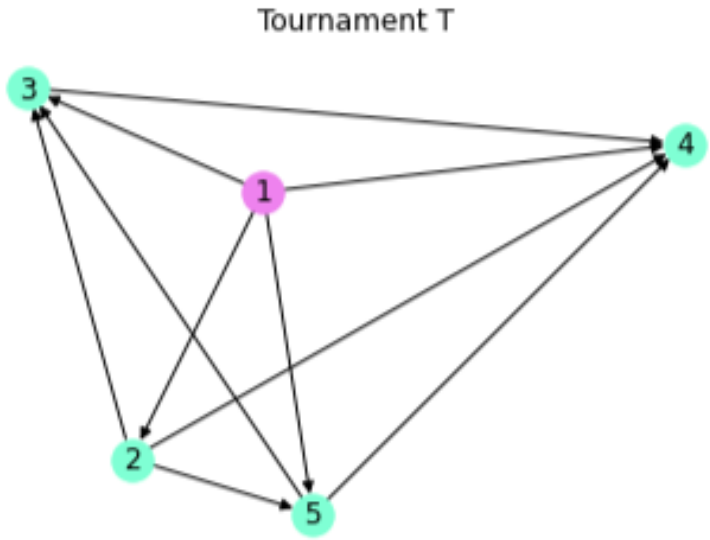
```python
# Gets indegrees for all vertexes
def get_indegrees(graph):
    indegrees = {v: 0 for v in graph.keys()}
    for neighbors in graph.values():
        for neighbor in neighbors:
            indegrees[neighbor] += 1
    return indegrees

# Finds a king in a Tournamet
def find_king(G, T):
    indegrees = get_indegrees(T)
    for x in T:
        if indegrees[x] == 0:
            return x
    x = next(iter(T))
    N = {y for y in T if x in G[y]}
    T_prime = T - {x} - N
    find_king(G, T_prime)

# Define a directed graph as an adjacency matrix
graph = np.array([[0, 1, 1, 1, 1], [0, 0, 1, 1, 1], [0, 0, 0, 1, 0], [0, 0, 0, 0, 0], [0, 0, 1, 1, 0]])
is_bipartite = False
if is_tournament(graph, is_bipartite):
    king = []
    T = adj_matrix_to_dict(graph)
    G = nx.DiGraph(T)
    king.append(find_king(G, T)) # Find the kings
    print("The king is:", king)
    draw_graph_with_kings(T, king, False)
else:
    print("A matrix given is not a tournament!")
```

Результат:

```
The king is: [1]
```



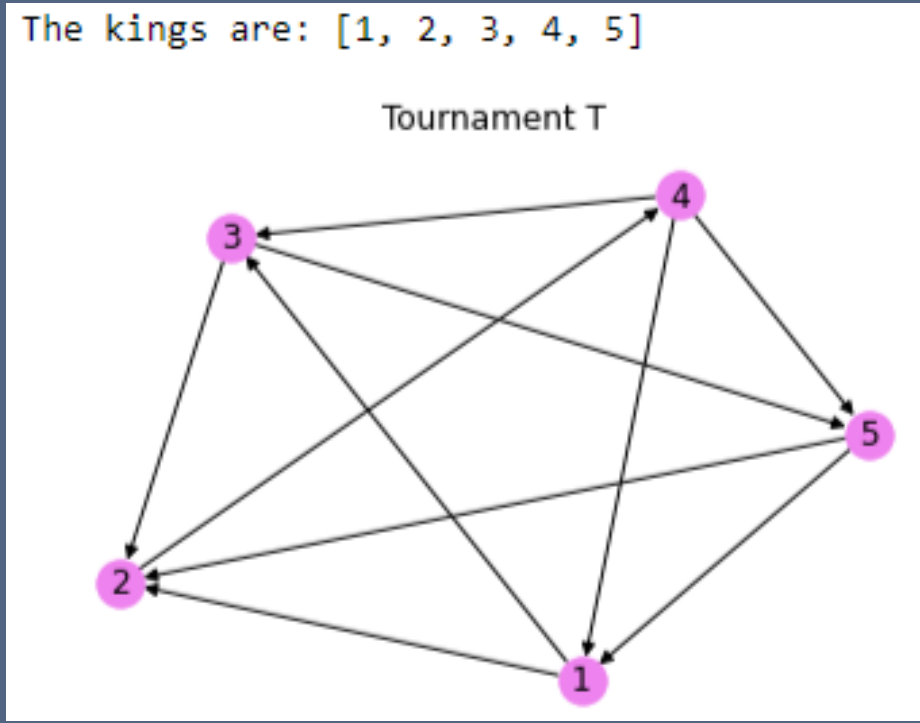
2. Алгоритм пошуку всіх королів у турнірі T [3]

```
# Finds kings
def find_kings(graph, T):
    max_degree = 2
    M = count_M(graph, T, max_degree)
    kings = []
    for v in T.keys():
        counter = 0
        for i in range(len(M[0])):
            if M[v-1][i] == 0:
                counter += 1
        if counter == 0:
            kings.append(v)
    return kings

# Define a directed graph as an adjacency matrix
graph = np.array([[0, 1, 1, 0, 0], [0, 0, 0, 1, 0], [0, 1, 0, 0, 1], [1, 0, 1, 0, 1], [1, 1, 0, 0, 0]])
is_bipartite = False
if is_tournament(graph, is_bipartite):
    T = adj_matrix_to_dict(graph)
    kings = find_kings(graph, T) # Find the kings
    print("The kings are:", kings)
    draw_graph_with_kings(T, kings, False)
else:
    print("A matrix given is not a tournament!")
```

Результат:

```
The kings are: [1, 2, 3, 4, 5]
```



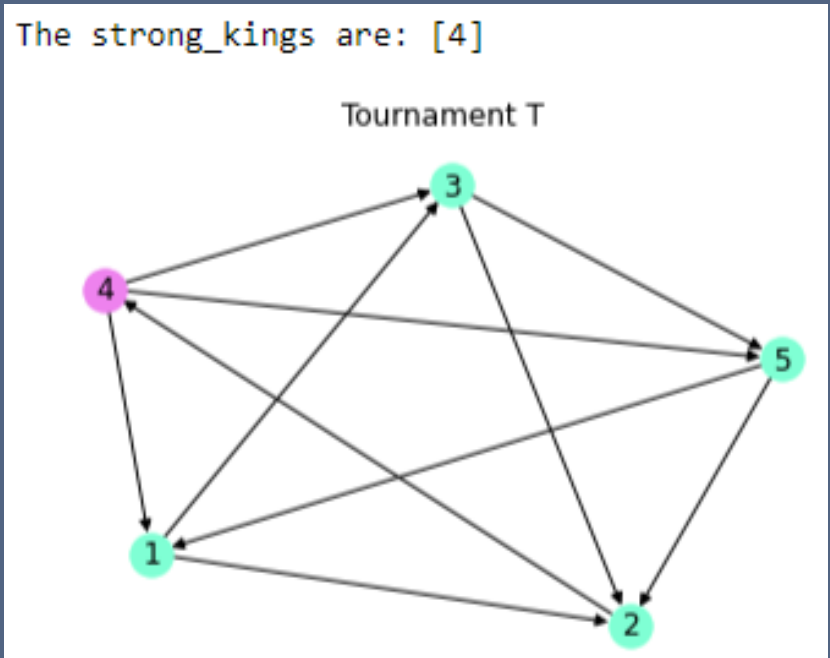
3. Алгоритм пошуку сильних королів у турнірі T [3]

```
# Finds strong kings
def find_strong_kings(graph, T):
    max_degree = 2
    M = count_M(graph, T, max_degree)
    strong_kings = []
    for v in T.keys():
        counter = 0
        for u in T.keys():
            if M[v-1][u-1] < M[u-1][v-1]:
                counter += 1
        if counter < 1:
            strong_kings.append(v)
    return strong_kings

# Define a directed graph as an adjacency matrix
graph = np.array([[0, 1, 1, 0, 0], [0, 0, 0, 1, 0], [0, 1, 0, 0, 1], [1, 0, 1, 0, 1], [1, 1, 0, 0, 0]])
is_bipartite = False
if is_tournament(graph, is_bipartite):
    T = adj_matrix_to_dict(graph)
    strong_kings = find_strong_kings(graph, T) # Find the kings
    print("The strong_kings are:", strong_kings)
    draw_graph_with_kings(T, strong_kings, False)
else:
    print("A matrix given is not a tournament!")
```

Результат:

```
The strong_kings are: [4]
```



4. Алгоритм побудови турнірів на n вершинах з k королями [3]

```python
# Generates a Tournament with n vetexes and k strong kings
def generate_tournament(n, k):
    if check_n_k(n,k) == True:
        K = set()
        if k % 2 == 1:  # Odd k
            T = generate_regular_tournament(k)
            for i in range(k):
                K.add(i + 1)
            if n != k:
                for v in range(k + 1, n + 1):
                    T.add_node(v)
                    for u in range(1, n):
                        if u != v:
                            if not T.has_edge(v, u):
                                T.add_edge(u, v)

        else:  # Even k
            T = generate_regular_tournament(k - 1)
            x, y, z = k, k + 1, k + 2
            for i in range(k - 1):
                K.add(i + 1)
            K.add(x)
            T.add_node(x)
            T.add_node(y)
            T.add_node(z)
            T.add_edge(x, z)
            T.add_edge(y, x)
            T.add_edge(y, z)
            u = random.choice(list(T.nodes()))
            while u == x or u == y or u == z:
                u = random.choice(list(T.nodes()))
```

```python
    v1 = find_first_predecessor(T, u, x, y, z)
    v2 = find_first_successor(T, u, x, y, z)
    T.add_edge(x, v1)
    T.add_edge(x, u)
    T.add_edge(v2, x)
    T.add_edge(v1, y)
    T.add_edge(u, y)
    T.add_edge(v2, y)
    T.add_edge(v1, z)
    T.add_edge(u, z)
    T.add_edge(z, v2)
    if n != k + 2:
        for v in range(k + 1, n + 1):
            T.add_node(v)
            for u in range(1, n):
                if u != v:
                    if not T.has_edge(v, u):
                        T.add_edge(u, v)
print("Strong kings are: ", K)
draw_graph_with_kings(T, K, True)
return T, K
```

Перевірка значень k і n

```python
# checks if n >= k >= 1 and k != n - 1, when n is odd or k != n, when n is even
def check_n_k(n, k):
    conditions_satisfied = True
    if n < k:
        conditions_satisfied = False
        print("n is less than k!")
    elif n < 1 and k < 1:
        conditions_satisfied = False
        print("n and k are less than 1!")
    elif n >= 1 and k < 1:
        conditions_satisfied = False
        print("k is less than 1!")
    else:
        if n % 2 == 0:
            if k == n:
                conditions_satisfied = False
                print("It is impossible to build a tournament with", n, "vertexes and", k, "kings, because k = n")
        if n % 2 == 1:
            if k == n - 1:
                conditions_satisfied = False
                print("It is impossible to build a tournament with", n, "vertexes and", k, "kings, because k = n - 1")

    return conditions_satisfied
```

Створення регулярного турнірах на k вершинах

```python
# Creates a regular tournamnt on k vertexes using nx
def generate_regular_tournament(k):
    T = nx.DiGraph()
    score = int (k/2)
    vertices = list(range(1, k+1))
    for i in vertices:
        for v in range(1, score + 1):
            j = (i + v) % k
            if j != 0:
                T.add_edge(i, j)
            else:
                T.add_edge(i, k)
    return T
```

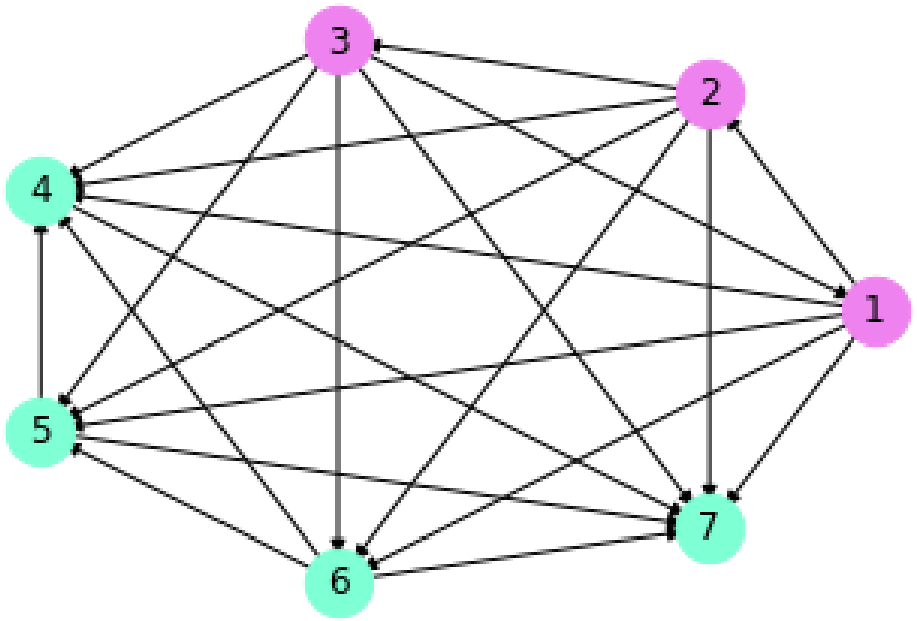
Знаходження перших сусідів для вершини u в $N^-(u)$ і $N^+(u)$

```python
# Find first predecessor for vertex u in a tournament T
def find_first_predecessor(T, u, x, y, z):
    predecessors = list(T.predecessors(u))
    if not predecessors: # no predecessors
        return None
    else:
        for i in predecessors:
            if i != x and i != y and i != z:
                return i
```

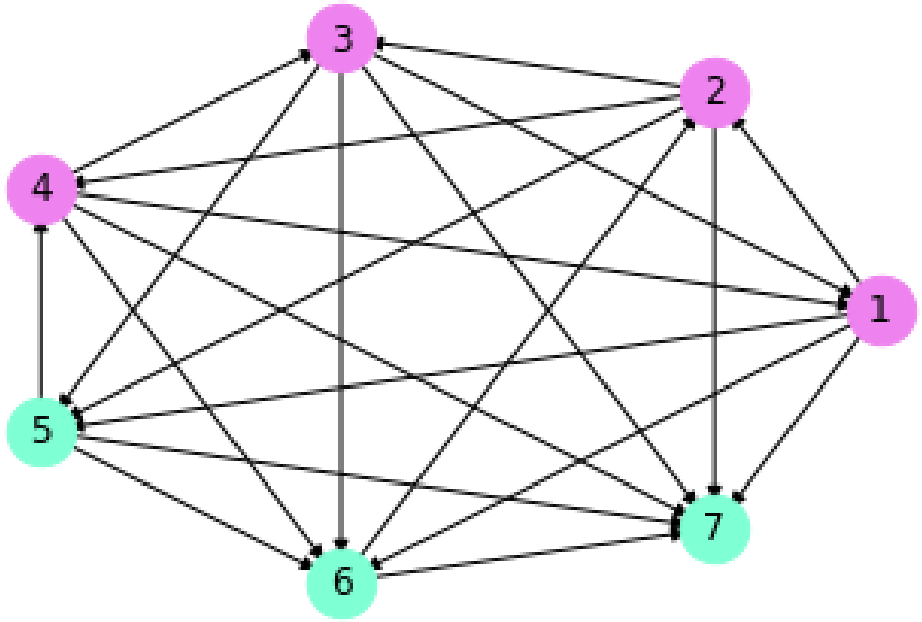
```python
# Find first successor for vertex u in a tournament T
def find_first_successor(T, u, x, y, z):
    successors = list(T.successors(u))
    if not successors: # no successors
        return None
    else:
        for i in successors:
            if i != x and i != y and i != z:
                return i
```

Результати для $n = 7, k = 3$ і $n = 7, k = 4$:

Strong kings are: {1, 2, 3}



Strong kings are: {1, 2, 3, 4}



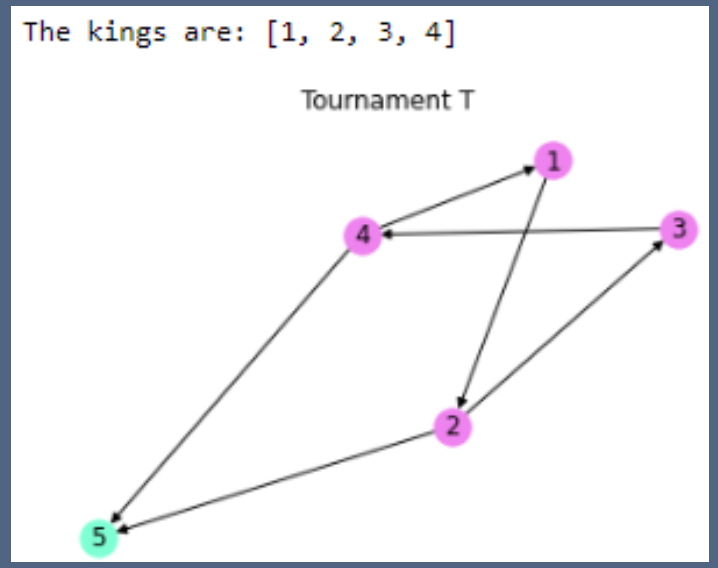
5. Алгоритм пошуку королів у двочастковому турнірі T [3]

```python
# Finds kings
def find_kings_in_bipartite(graph, T):
    max_degree = 4
    M = count_M(graph, T, max_degree)
    kings = []
    for v in T.keys():
        counter = 0
        for i in range(len(M[0])):
            if M[v-1][i] == 0:
                counter += 1
        if counter == 0:
            kings.append(v)
    return kings

# Define a directed graph as an adjacency matrix
graph = np.array([[0, 1, 0, 0, 0], [0, 0, 1, 0, 1], [0, 0, 0, 1, 0], [1, 0, 0, 0, 1], [0, 0, 0, 0, 0]])
is_bipartite = True
if is_tournament(graph, is_bipartite):
    T = adj_matrix_to_dict(graph)
    kings = find_kings_in_bipartite(graph, T) # Find the kings
    print("The kings are:", kings)
    draw_graph_with_kings(T, kings, False)
else:
    print("A matrix given is not a bipartite tournament!")
```

Результат:

```
The kings are: [1, 2, 3, 4]
```



Література

1. Douglas B. West, Introduction in Graph Theory, Pearson Education Inc., 2001, 589 p.
2. J.W. Moon, Topics on tournaments, Holt, Rinehart and Winston Inc., 1968, 137 p.
3. Kings and serfs in tournaments, [Електронний ресурс] https://gyires.inf.unideb.hu/KMITT/a53/ch07s04.html.
4. M. H. McAndrew., On the Product of Directed Graphs, Proceedings of the American Mathematical Society Volume 14, 1963, 606 p.
5. Morgan Norge, Kings in the Direct Product of Digraphs, Virginia Commonwealth University, 2019, 32 p.

Дякуємо за увагу!