МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

ЧИСЛО НЕЗАВИСИМОГО ДОМИНИРОВАНИЯ И ЧИСЛО СОВЕРШЕННОГО ГЕОДОМИНИРОВАНИЯ

НАУЧНО-ИССЛЕДОВАТЕЛЬСКАЯ РАБОТА

студента 5 курса 531 группы
направления 10.05.01 — Компьютерная безопасность
факультета КНиИТ
Стаина Романа Игоревича

Проверил
д. ф.-м. н., доцент ______________ М. Б. Абросимов

Саратов 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Введение

1 Основные понятия и определения

2 Алгоритмы

2.1 Нахождение числа независимого доминирования

2.1.1 Необходимые обозначения и обоснование алгоритма

Итак, число независимого доминирования $\gamma_i(G)$ – размер наименьшего независимого доминирующего множества, или, эквивалентно, минимальный размер максимального независимого множества.

В процессе поиска – на некотором этапе k независимое множество вершин S_k расширяется путем добавления к нему подходящим образом выбранной вершины (чтобы получилось новое независимое множество S_{k+1}) на этапе $k+1$, и так поступают до тех пор, пока добавление вершин станет невозможным, а порожденное множество не станет максимально независимым множеством. Пусть Q_k будет на этапе k наибольшим множеством вершин, для которого $S_k \cap Q_k = \varnothing$, то есть после добавления любой вершины из Q_k в S_k получится независимое множество S_{k+1}. В некоторый произвольный момент работы алгоритма множество Q_k состоит из вершин двух типов: подмножества Q_k^- тех вершин, которые уже использовались в процессе поиска для расширения множества S_k, и подмножества Q_k^+ таких вершин, которые ещё не использовались. Тогда дальнейшее ветвление в дереве поиска включает процедуру выбора вершины $x_{ik} \in Q_k^+$, добавления её к S_k

$$S_{k+1} = S_k \cup \{x_{ik}\}$$

и порождения новых множеств

$$Q_{k+1}^- = Q_k^- \setminus \Gamma(x_{ik});$$

$$Q_{k+1}^+ = Q_k^+ \setminus (\Gamma(x_{ik}) \cup \{x_{ik}\}).$$

Шаг возвращения алгоритма состоит в удалении вершины x_{ik} из S_{k+1}, чтобы вернуться к S_k, изъятии x_{ik} из старого множества Q_k^+ и добавлении x_{ik} к старому множеству Q_k^- для формирования новых множеств Q_k^- и Q_k^+. Множество S_k является максимально независимым множеством только тогда, когда невозможно его дальнейшее расширение, то есть когда $Q_k^+ = \varnothing$. Если $Q_k^- = \varnothing$, то текущее множество S_k было расширено на некотором предшествующем этапе работы алгоритма путем добавления вершины из Q_k^-, и поэтому не является максимально независимым множеством. Таким образом, необходимым и достаточным условием того, что S_k – максимально независимое множество, является выполнение равенств:

$$Q_k^+ = Q_k^- = \varnothing.$$

Если очередной этап работы алгоритма наступает тогда, когда существует некоторая вершина $x \in Q_k^-$, для которой $\Gamma(x) \cap Q_k^+ = \varnothing$, то безразлично, какая из вершин, принадлежащих Q_k^+, использовалась для расширения S_k, и это справедливо при любом числе дальнейших ветвлений. Вершина x не может быть удалена из Q_p^- на любом следующем шаге $p > k$. Таким образом, существование x, такого что

$$x \in Q_k^- \text{ и } \Gamma(x) \cap Q_k^+ = \varnothing \qquad (*)$$

является достаточным для осуществления шага возвращения, потому что из S_k при всяком дальнейшем ветвлении уже не получится максимально независимое множество.

Если $k = 0$, то возвращение выполнить невозможно, поэтому при $k = 0$ осуществляется переход на прямой шаг.

2.1.2 Алгоритм нахождения числа независимого доминирования

Начальная установка.

Шаг 1. Пусть $S_0 = Q_0^- = \varnothing, Q_0^+ = X, k = 0$.

Прямой шаг.

Шаг 2. Выбрать вершину $x_{ik} \in Q_k^+$ и сформировать S_{k+1}, Q_{k+1}^- и Q_{k+1}^+, оставив Q_k^- и Q_k^+ нетронутыми. Положить $k = k + 1$.

Проверка.

Шаг 3. Если выполняется условие $(*)$, то перейти к шагу 5, иначе к шагу 4.

Шаг 4. Если $Q_k^+ = Q_k^- = \varnothing$, то сохранить максимальное независимое множество S_k и перейти к шагу 5. Если $Q_k^+ = \varnothing$, а $Q_k^- \neq \varnothing$, то перейти к шагу 5, иначе – к шагу 2.

Шаг возвращения.

Шаг 5. Положить $k = k - 1$. Удалить x_{ik} из S_{k+1}, чтобы получить S_k. Исправить Q_k^+ и Q_k^-, удалив вершину x_{ik} из Q_k^+ и добавив ее к Q_k^-. Если $k \neq 0$, то перейти к шагу 3, иначе если $Q_0^+ = \varnothing$, то остановиться (к этому моменту будут сохранены все максимальные независимые множества), иначе перейти к шагу 2.

Результат.

Шаг 6. Найти среди сохранённых максимальных независимых множеств S_k-ых то, у которого мощность меньше остальных. Вернуть её как результат.

2.2 Нахождение числа совершенного геодоминирования

Алгоритм нахождения числа совершенного геодоминирования представляет собой перебор неповторяющихся подмножеств вершин S множества $V = \{x_0, x_1, \ldots, x_n\}$ до тех пор, пока какое-либо из этих множеств не удовлетворит определению *совершенного геодоминирующего множества* (то есть каждое такое S рассматривается как потенциальное совершенное геодоминирующее множество).

Стоит отметить, что перебор можно ускорить, если к таким подмножествам S добавлять вершины, степень которых равна 1. Это обусловлено тем, что такие вершины не будут находиться на пути между двумя другими отличными вершинами. Обозначим множество таких вершин B.

Так же в алгоритме используется вспомогательная матрица M, в которой сохраняются все кратчайшие пути между каждой парой вершин из V. Это необходимо, чтобы не считать кратчайший путь между вершинами несколько раз.

2.2.1 Алгоритм нахождения числа совершенного геодоминирования

Шаг 1. Построить множество B вершин степени 1. Пусть $k = 2 - |B|$. Если $k < 0$, то $k = 0$.

Шаг 2. Вычислить следующее сочетание вершин без повторений C длиной k из множества $V \setminus B$ и добавить к нему B. В итоге $S = B \cup C$.

Шаг 3. Пусть $P = \varnothing$.

Шаг 4. Для каждой пары вершин x_i, x_j из S: если в M_{ij} не сохранёны пути между вершинами x_i и x_j $P_0, \ldots, P_l$, то вычислить их. Затем добавить к P – $P = P_0 \cup \cdots \cup P_l$ эти пути без начальных и конечных вершин (то есть x_i и x_j).

Шаг 5. Если $P = \varnothing$, то $k = k + 1$ и вернуться к шагу 2.

Шаг 6. Для каждой вершины из $V \setminus S$ проверяется, что она встречается в P всего 1 раз. Если это не так, то $k = k + 1$ и вернуться к шагу 2. Иначе S – совершенное геодоминирующее множество. Вернуть $|S|$ в качестве результата.

3 Исследование количества

3.1 Использованные программные средства

3.2 Результаты исследования

Число вершин	Число графов	Время работы
1	1	1.18 с
2	1	1.18 с
3	2	1.21 с
4	6	1.22 с
5	21	1.28 с
6	112	1.3 с
7	853	1.45 с
8	11117	5.62 с
9	261080	3.37 мин
10	11716571	4.73 ч
11	1006700565	30 дней

Таблица 1 – Время работы программы

g_p \ γ_i	1
2	1

Таблица 2 – Количество 2-вершинных графов, имеющих заданные g_p и γ_i

g_p \ γ_i	1
2	1
3	1

Таблица 3 – Количество 3-вершинных графов, имеющих заданные g_p и γ_i

g_p \ γ_i	1	2
2	1	2
3	0	0
4	3	0

Таблица 4 – Количество 4-вершинных графов, имеющих заданные g_p и γ_i

g_p \ γ_i	1	2
2	2	3
3	0	2
4	2	1
5	7	4

Таблица 5 – Количество 5-вершинных графов, имеющих заданные g_p и γ_i

g_p \ γ_i	1	2	3
2	4	11	0
3	0	8	0
4	8	14	0
5	7	11	0
6	15	28	4

Таблица 6 – Количество 6-вершинных графов, имеющих заданные g_p и γ_i

γ_i / g_p	1	2	3
2	11	23	1
3	0	70	2
4	24	122	8
5	53	108	17
6	26	100	14
7	42	192	40

Таблица 7 – Количество 7-вершинных графов, имеющих заданные g_p и γ_i

γ_i / g_p	1	2	3	4
2	34	137	5	0
3	0	581	4	0
4	126	1638	107	2
5	314	1489	238	1
6	283	1467	314	4
7	145	1249	350	0
8	142	1865	601	21

Таблица 8 – Количество 8-вершинных графов, имеющих заданные g_p и γ_i

γ_i / g_p	1	2	3	4	5
2	156	888	1	0	0
3	0	8002	182	0	0
4	861	31921	1853	13	0
5	3422	38055	6310	27	0
6	3477	33948	7715	66	0
7	2461	29432	9134	144	0
8	1210	25377	10380	184	0
9	759	31055	13511	535	1

Таблица 9 – Количество 9-вершинных графов, имеющих заданные g_p и γ_i

g_p \ γ_i	1	2	3	4	5
2	1044	11456	7	4	0
3	0	189606	4209	1	0
4	10658	998277	69100	281	4
5	56296	1607983	274765	639	0
6	81435	1586103	388261	2498	5
7	60293	1222947	385507	4654	8
8	37819	1036069	434625	9015	28
9	18948	1004799	495269	13250	0
10	8175	1105022	575185	22158	168

Таблица 10 – Количество 10-вершинных графов, имеющих заданные g_p и γ_i

g_p \ γ_i	1	2	3	4	5	6
2	12346	262322	66	9	0	0
3	699	8237385	315318	288	0	0
4	242749	52602584	4781780	11974	51	0
5	1636063	110808715	22172688	44650	56	0
6	3216573	129754442	36050965	152111	124	0
7	2839470	100490898	34970157	307986	387	0
8	1830607	74502192	33823940	523234	1027	1
9	1196547	73676844	38671436	854752	2438	0
10	727558	81534788	38182632	1192088	3404	0
11	302556	97320312	51779224	1650914	11205	10

Таблица 11 – Количество 11-вершинных графов, имеющих заданные g_p и γ_i

3.3 Построение гипотез

ЗАКЛЮЧЕНИЕ

Заключение

ПРИЛОЖЕНИЕ А

Примеры графов

ПРИЛОЖЕНИЕ Б

Листинг main.py

```
# число независимого доминирования и число совершенного геодоминирования
from sys import stdin
import networkx as nx
import time
from itertools import combinations
import multiprocessing as mp


def find_maximal_independent_sets(g):
    # 1 Начальная установка
    k, s, q_minus, q_plus = 0, [], [[]], [list(g.nodes)]
    result = []

    def step2(k):
        xk = q_plus[k][0]
        s.append(xk)
        if k >= len(q_minus) - 1:
            q_minus.append([])
        else:
            q_minus[k + 1] = list(set(q_minus[k]) - set(g.adj[xk]))
        t_plus_k = q_plus[k].copy()
        t_plus_k.remove(xk)
        if k >= len(q_plus) - 1:
            q_plus.append(list(set(t_plus_k) - set(g.adj[xk])))
        else:
            q_plus[k + 1] = list(set(t_plus_k) - set(g.adj[xk]))
        k += 1
        # print(f'Шаг 2 \nk = {k}, xk = {xk} \nS = {s} \nQ+ = {q_plus} \nQ- = {q_minus}')
        return xk, k

    def step5(s, k):
        k -= 1
        xk = s[k]
        s.pop()
        q_plus[k].remove(xk)
        q_minus[k].append(xk)
        # print(f'Шаг 5 \nk = {k}, xk = {xk} \nS = {s} \nQ+ = {q_plus} \nQ- = {q_minus}')
        return k

    while len(q_plus[0]) != 0:
        if k != 0:
            # 3 Проверка
```

```
            if xk in q_minus and set(g.adj[xk]) & set(q_plus[k]) == set():
                k = step5(s, k)
            else:
                # 4
                if len(q_plus[k]) == 0:
                    if len(q_minus[k]) == 0:
                        # print('=' * 40)
                        # print(f'{s} к результату')
                        result.append(s.copy())
                        # print('=' * 40)
                        k = step5(s, k)
                    else:
                        k = step5(s, k)
                else:
                    xk, k = step2(k)
        else:
            xk, k = step2(k)
    return result


def independent_domination_number(g):
    g6 = g[0:-1]
    g = nx.from_graph6_bytes(g[0:-1].encode())
    maximal_independent_sets = find_maximal_independent_sets(g)
    maximal_independent_sets.sort(key=len)
    for s in maximal_independent_sets:
        if nx.is_dominating_set(g, s):
            return g6, len(s)


def find_perfect_geodominating_sets(g):
    g6 = g[0:-1]
    g = nx.from_graph6_bytes(g[0:-1].encode())
    nodes = set(g.nodes)
    num_nodes = len(nodes)
    all_paths = [[None] * num_nodes for _ in range(num_nodes)]

    base_s = set()
    for node in nodes:
        if g.degree[node] == 1:
            base_s.add(node)
    nodes = nodes - base_s

    for k in range(1, len(nodes)):
        # Всевозможные варианты множества вершин S
        for si in combinations(nodes, k):
            s = set(si).union(base_s)
            v = nodes - s
```

```
            # print('S =', si)
            # print('V\\S =', v)

            paths = []
            # Всевозможные кратчайшие пути между вершинами из S
            for a, b in combinations(s, 2):
                if all_paths[a][b] is None:
                    all_paths[a][b] = list(nx.all_shortest_paths(g, a, b))
                for p in all_paths[a][b]:
                    paths += p[1:-1]

            # print('Пути', paths)
            if len(paths) == 0:
                continue

            flag = True
            for node in v:
                count = paths.count(node)
                # Если какая-то вершина из V\S встречается больше одного раза, значит
                # она геодоминируется несколькими парами вершин из S. Или вершина не геод-ся вовсе
                if count != 1:
                    # print('No', si)
                    flag = False
                    break

            if flag:
                # print('S', si)
                # print('V\\S', v)
                return g6, len(si)
    # Если не получилось найти, значит в S должны быть все вершины
    # print('S', nodes)
    return g6, num_nodes


if __name__ == '__main__':
    graphs6 = stdin.readlines()
    graphs6 = graphs6[len(graphs6) // 2:]
    t0 = time.time()
    g = nx.from_graph6_bytes(graphs6[0][0:-1].encode())
    num_nodes = len(g.nodes)
    f_name = 'res' + str(num_nodes) + '.txt'
    res_file = open(f_name, 'a')
    res_file.truncate(0)

    agents = mp.cpu_count()
    chunksize = 1
    with mp.Pool(processes=agents) as pool:
```

```
        result = pool.map(find_perfect_geodominating_sets, graphs6, chunksize)
        result2 = pool.map(independent_domination_number, graphs6, chunksize)

    table = [[0] * num_nodes for _ in range(num_nodes)]
    for a, b in zip(result, result2):
        table[a[1] - 1][b[1] - 1] += 1
        # Первое число геодоминирования, второе доминирования
        res_file.write(a[0] + str(a[1]) + ',' + str(b[1]) + '\n')

    for row in table:
        res_file.write(str(row) + '\n')

    print(f'Время работы: {time.time() - t0} сек.')
    res_file.close()
```