Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий – РТФ
Школа профессионального и академического образования

ДОПУСТИТЬ К ЗАЩИТЕ ПЕРЕД ГЭК

РОП И.Н. Обабков

_____
(подпись)
«_____»_____2023 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

АЛГОРИТМЫ ПОСТРОЕНИЯ МАКСИМАЛЬНЫХ ГРАФИЧЕСКИХ РАЗБИЕНИЙ, ДОМИНИРУЮЩИХ ЗАДАННОЕ ГРАФИЧЕСКОЕ РАЗБИЕНИЕ

Научный руководитель: Ф.И.О. Е. Ю. Просвиряков _____
ученая степень, ученое звание (должность): д. ф.-м. н., профессор
Нормоконтролер: Ф.И.О. _____ _____
Студент группы РИМ-210990 Ф.И.О. В. В. Зуев _____

Екатеринбург
2023

РЕФЕРАТ

Зуев Валентин Викторович, «Алгоритмы построения максимальных графических разбиений, доминирующих заданное графическое разбиение», работа содержит: стр. 57, рис. 11, библ. 11 назв.

Ключевые слова: граф, пороговый граф, целочисленное разбиение, графическое разбиение, диаграмма Ферре, решетка.

В работе рассмотрены разбиения целых неотрицательных чисел и изучены некоторые детали строения решеток разбиений. Решетка разбиений задается отношением доминирования, которое определяется как покомпонентное доминирование соответственных частичных сумм разбиений.

Особое внимание уделено графическим разбиениям. Графическим называется разбиение, которые составлено из степеней вершин графа с добавлением нулей. Проведено исследование множества максимальных графических разбиений, доминирующих заданное графическое разбиение. Для исследования написана компьютерная программа на языке Python, позволяющая проверять гипотезы для разбиений небольших чисел.

Основной результат работы — алгоритм построения всех максимальных графических разбиений, доминирующих заданное разбиение и имеющих такой же вес. Этот алгоритм базируется на основной теореме, которая описывает точное строение множества таких максимальных графических разбиений. Приведена программная реализация найденного алгоритма.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

N_0 — множество целых неотрицательных чисел;

$G = (V, E)$ — простой граф G с множеством вершин V и множеством ребер E;

$\deg(v_i)$ — степень вершины v_i;

$\mathrm{sum}(\lambda)$ — вес разбиения λ;

$\ell(\lambda)$ — длина разбиения λ;

$\mathrm{r}(\lambda)$ — ранг разбиения λ;

λ^* — разбиение, сопряженное λ;

$\mathrm{hd}(\lambda)$ — голова разбиения λ;

$\mathrm{tl}(\lambda)$ — хвост разбиения λ;

NPL — множество всех разбиений всех целых неотрицательных чисел;

NPL(m) — множество всех разбиений целого неотрицательного числа m;

GPS($2m$) — множество всех графических разбиений веса $2m$.

MGP($2m$) — множество всех максимальных графических разбиений веса $2m$.

ВВЕДЕНИЕ

Разбиение числа — это его представление в виде суммы натуральных чисел. Задачи о разбиениях сыграли важную роль для всей математики. Основу теории разбиений заложил Л. Эйлер, и впоследствии им был разработан метод производящих функций, который оказался эффективным инструментом для комбинаторики и теории чисел. Дальнейшее изучение теории разбиений привело к новым комбинаторным и алгебраическим методам. В книге [1] рассмотрено большое количество задач и способов их решений, связанных с разбиениями чисел.

Теория разбиений находит применение во многих областях математики, в том числе в теории графов. В данной работе рассматриваются только обыкновенные графы, т.е. графы без петель и кратных ребер. Каждому графу соответствует невозрастающая последовательность степеней вершин, которую можно интерпретировать как разбиение числа. Такие разбиения называются графическими. Не каждое разбиение является графическим. В частности, разбиение нечетного числа не может быть графическим, так как сумма степеней графа всегда является четным числом. Кроме того, два не изоморфных друг другу графа могут иметь одни и те же степени вершин, и им будут соответствовать одинаковые графические разбиения.

На множестве всех разбиений целого неотрицательного числа удобно задать отношение доминирования. Определение отношения доминирования (см. [2]) дано далее в тексте работы. Множество всех разбиений образует решетку, а множество графических разбиений числа по отношению доминирования является всего лишь полурешеткой, так как имеет одно минимальное разбиение и несколько максимальных.

Особый интерес среди графов представляют пороговые графы. В монографии [3] дано несколько эквивалентных определений пороговых графов, а также множество их применений. Пороговые графы тесно связаны с разбиениями чисел. В работе [4] показано, что каждое максимальное

графическое разбиение реализует пороговый граф. И наоборот, пороговому графу соответствует максимальное графическое разбиение.

В [5] более подробно изучены максимальные графические разбиения, в том числе указан метод нахождения максимального графического разбиения, доминирующего данное. Однако этот метод позволяет найти только одно такое разбиение, хотя их может быть несколько.

Кроме того во всех прежних работах по данной тематике внимание всегда фокусировалось исключительно на теоретических фактах существования различных разбиений. Мы в нашей работе будем также уделять внимание не только факту существования тех или иных разбиение, но и способу отыскания таких разбиений.

Также важно отметить, что для нас важно будет как описать идеи алгоритмов для поиска различных разбиений, так и реализовать их на языках программирования, чтобы, во-первых, убедиться в работоспособности алгоритма, а, во-вторых, подробней изучить подводные камни, присутствующие в алгоритме.

Итак, цели данной работы :

1. Построить алгоритм нахождения всех максимальных графических разбиений, доминирующих данное и имеющих такой же вес как у данного.

2. Построить алгоритм нахождения количества максимальных графических разбиений, доминирующих данное и имеющих такой же вес как у данного.

3. Реализовать оба алгоритма с помощью языка программирования Python 3.

1 ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ, ИСТОРИЯ ВОПРОСА И ПОСТАНОВКА ЗАДАЧИ

Всюду под графами далее будут пониматься обыкновенные графы, т. е. графы без петель и кратных ребер. и будет использоваться терминология, изложенная в [6].

Целочисленным разбиением или просто разбиением (см. [1]) называют невозрастающую последовательность $\lambda = (\lambda_1, \lambda_2, \ldots)$ целых неотрицательных чисел, которая содержит лишь конечное число ненулевых компонент. Через $sum(\lambda)$ принято обозначать сумму всех компонент разбиения λ, ее называют весом разбиения λ. Принято говорить, что разбиение λ является разбиением целого неотрицательного числа $n = sum(\lambda)$.

Длиной $\ell(\lambda)$ разбиения λ называют число его ненулевых компонент. Часто для удобства ненулевое разбиение λ записывают в виде

$$\lambda = (\lambda_1, \ldots, \lambda_t), \tag{1.1}$$

где t – натуральное число такое, что $t \geq (\lambda)$, т. е. опускают нули, начиная с некоторой нулевой компоненты, при этом помнят, что разбиение является бесконечной последовательностью.

Теория разбиений является важным разделом комбинаторики. На необходимость изучения разбиений обратил внимание Г. В. Лейбниц. Основы же теории разбиений были заложены Л. Эйлером в 18-м веке. Эйлер получил глубокие результаты в перечислительной теории разбиений. Теория разбиений активно развивалась в 19-м и 20-м веках. Многочисленные достижения теории разбиений изложены в монографии [1]. В последние десятилетия этот важный раздел комбинаторики также активно развивается. Опубликовано огромное число научных статей по теории разбиений и ее многочисленным приложениям в различных разделах науки, включая информатику, химию, теорию графов, теорию конечных групп и т.д. (см., например, [3]). Алгоритмы обработки разбиений используются во многих

программных системах.

Через NPL обозначают множество всех разбиений, а через NPL(m), где $m \in N$, – множество всех разбиений λ таких, что $sum(\lambda) = m$.

Дадим определение *отношения доминирования* $\geq$ на множестве NPL и на множествах вида NPL(m) [2]. Два разбиения $\mu = (\mu_1, \mu_2, \ldots)$ и $\lambda = (\lambda_1, \lambda_2, \ldots)$ находятся в отношении $\geq$, т.е. выполняется $\mu \geq \lambda$, тогда и только тогда, когда

$$\begin{gathered} \lambda_1 \leq \mu_1, \\ \lambda_1 + \lambda_2 \leq \mu_1 + \mu_2, \\ \ldots \\ \lambda_1 + \ldots + \lambda_t \leq \mu_1 + \ldots + \mu_t, \\ \ldots \quad , \end{gathered} \tag{1.2}$$

иными словами, префиксные частичные суммы разбиения λ не превосходят соответствующих префиксных частичных сумм разбиения μ.

Определим элементарные преобразования разбиения $\lambda = (\lambda_1, \lambda_2, \ldots)$, которые были введены в работах В.А. Баранского и его учеников (см. [7] – [9]). Таких преобразований имеется два типа.

Пусть i и j – натуральные числа таких, что $1 \leq i < j \leq \ell(\lambda) + 1 = t$, для которых выполняются неравенства:

1) $\lambda_i - 1 \geq \lambda_{i+1}$,

2) $\lambda_{j-1} \geq \lambda_j + 1$,

3) $\lambda_i \geq 2 + \lambda_j$.

Говорят, что разбиение $\eta = (\lambda_1, \ldots, \lambda_i - 1, \ldots, \lambda_j + 1, \ldots, \lambda_t)$ получено из разбиения $\lambda = (\lambda_1, \ldots, \lambda_i, \ldots, \lambda_j, \ldots, \lambda_t)$ с помощью *элементарного преобразования первого типа* (или *перекидывания блока*). Разбиение η отличается от разбиения λ точно на двух компонентах с номерами i и j.

Условия 1) и 2) гарантируют, что после применения элементарного преобразования первого типа снова получается разбиение. Ясно, что элементарные преобразования первого типа сохраняют вес разбиения.

Пусть i – натуральное число такое, что $1 \leq i \leq \ell(\lambda)$, и выполняется условие $\lambda_i - 1 \geq \lambda_{i+1}$.

Говорят, что разбиение $\eta = (\lambda_1, \ldots, \lambda_{i-1}, \lambda_i - 1, \lambda_{i+1}, \ldots)$ получено из разбиения $\lambda = (\lambda_1, \ldots, \lambda_{i-1}, \lambda_i, \lambda_{i+1}, \ldots)$ с помощью элементарного преобразования второго типа (или удаления блока). Условие $\lambda_i - 1 \geq \lambda_{i+1}$ гарантируют, что после применения элементарного преобразования второго типа снова получается разбиение. Ясно, что элементарные преобразования второго типа уменьшают вес разбиения на 1.

В работах [8] и [7] доказана важная теорема о том, что для любых разбиений λ и μ в NPL выполняется условие $\lambda \leq \mu$ тогда и только тогда, когда разбиение λ можно получить из разбиения μ с помощью последовательного применения конечного числа элементарных преобразований (допустимы оба типа элементарных преобразований).

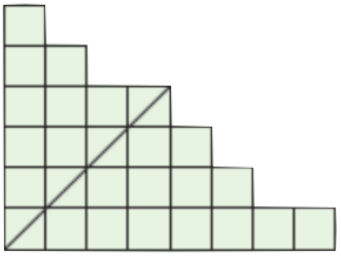
Пусть n – натуральное число. Конечную невозрастающую последовательность $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$ целых неотрицательных чисел принято называть n-последовательностью. Графической называется n-последовательность, для которой существует обыкновенный граф G на n вершинах $v_1, \ldots, v_n$, для которого $\deg(v_1) = \lambda_1, \ldots, \deg(v_n) = \lambda_n$. Граф G называют реализацией n-последовательности λ, также говорят, что n-последовательность λ реализуется графом G.

Разбиение $\lambda = (\lambda_1, \lambda_2, \ldots)$ называют графическим, если графической является последовательность $(\lambda_1, \ldots, \lambda_t)$, где $t = \ell(\lambda)$. Ясно, что разбиение $\lambda = (\lambda_1, \lambda_2, \ldots)$ является графическим тогда и только тогда, когда графической будет любая n-последовательность $(\lambda_1, \ldots, \lambda_n)$ такая, что $n \geq \ell(\lambda)$.

Разбиения принято изображать в виде диаграмм Ферре, которые состоят

из конечного набора квадратных *блоков* одинакового размера, составляющих «ступенчатую» фигуру (см., например, Рис. 1).

На Рис. 1.1 представлена диаграмма Ферре разбиения λ = (6, 5, 4, 4, 3, 2, 1, 1) длины 8 и веса 26.



$$\lambda = (6, 5, 4, 4, 3, 2, 1, 1)$$

$$\lambda^* = (8, 6, 5, 4, 2, 1)$$

$$r(\lambda) = r(\lambda^*) = 4$$

Рис. 1.1

Такие диаграммы изображаются различными способами. Мы используем декартово представление диаграмм (в отличие от диаграмм Юнга).

Дадим для произвольного разбиения = (λ1, λ2, …) определение его ранга. *Рангом* r(λ) разбиения λ или рангом *Дерфи* называется число

$$r(\lambda) = \max\{i \mid \lambda_i \geq i\}. \qquad (1.3)$$

Ясно, что ранг r = r(λ) разбиения λ равен числу блоков на главной диагонали его диаграммы Ферре (см. Рис. 1.1).

Максимальный квадрат, составленный из блоков диаграммы Ферре и симметричный относительно главной диагонали диаграммы Ферре, называют *квадратом Дерфи* разбиения λ (на Рис. 1.1 указана диагональ квадрата Дерфи, здесь ранг r = 4).

Дадим теперь определение головы и хвоста разбиения, введенные в работах В.А. Баранского и его учеников.

Головой hd(λ) разбиения λ называется разбиение, которое получается из разбиения λ уменьшением первых r компонент этого разбиения λ на одно и то же число r – 1 и обнулением всех компонент с номерами r + 1, r + 2, … .

Конечно, верхняя строка квадрата Дерфи всегда входит в диаграмму Ферре головы hd(λ) в качестве первой строки.

Хвостом tl(λ) разбиения λ называется разбиение, для которого диаграмма Ферре получается из диаграммы Ферре разбиения λ удалением первых r = r(λ) столбцов и последующим считыванием компонент по строкам, т. е. заменяя строки последовательно на столбцы, продвигаясь снизу-вверх.

На Рис. 1.2 для разбиения λ = (6, 5, 4, 4, 3, 2, 1, 1) имеем r = r(λ) = 4, hd(λ) = (3, 2, 1, 1) и tl(λ) = (4, 2, 1). Через λ^* обозначают разбиение, сопряженное к разбиению λ, т.е. полученное с помощью зеркальной симметрии относительно главной диагонали. Здесь $tl(\lambda)^*$ – разбиение, сопряженное к разбиению tl(λ).

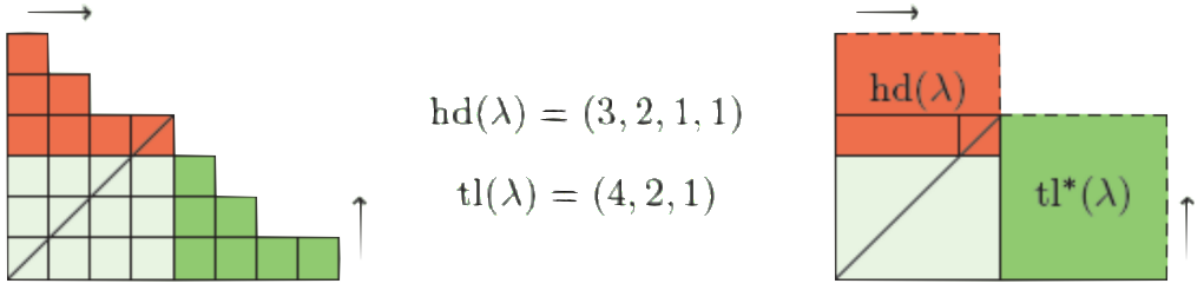


Рис. 1.2

Отметим, что всегда ℓ(hd(λ)) = r(λ), ℓ(λ) – r равно величине первой компоненты хвоста tl(λ) и ℓ(tl(λ)) = λ_{r+1}. Отметим, что разбиение hd(λ) «считывается» по столбцам, «урезанным» на число r – 1, слева направо, а разбиение tl(λ) «считываются» по строкам, «урезанным» на число r, снизу-вверх.

Пусть G – произвольный граф и $v_1, \ldots, v_n$ – множество всех его вершин, упорядоченное таким образом, что $\deg(v_1) = \lambda_1 \geq \deg(v_2) = \lambda_2 \geq \ldots \geq \deg(v_n) = \lambda_n$.

Разбиение $\lambda = (\lambda_1, \ldots, \lambda_n, 0, 0, \ldots)$ называют графическим разбиением, отвечающим графу G, его обозначают через dpt(G). Конечно, граф G является реализацией такого графического разбиения λ.

Очевидно, что добавление к графу или удаление из графа изолированных вершин не меняет отвечающего ему разбиения, т. е. реализации графических разбиений можно исследовать с точностью до изолированных вершин.

Отметим, что по лемме о рукопожатиях для графов любое графическое разбиение имеет четный вес, который равен удвоенному числу ребер в каждой из его реализаций.

Первый критерий графичности n-последовательности был найден П. Эрдешем и Т. Галлаи [10].

В [11] найден следующий ht-критерий графичности разбиения:

Разбиение λ четного веса является графическим тогда и только тогда, когда $hd(\lambda) \leq tl(\lambda)$.

На Рис. 1.2 для разбиения $\lambda = (6, 5, 4, 4, 3, 2, 1, 1)$ выполняется $hd(\lambda) = (3, 2, 1, 1) \leq tl(\lambda) = (4, 2, 1)$, так как в данном примере $hd(\lambda)$ можно получить из $tl(\lambda)$ с помощью одного элементарного преобразования первого типа, перекидывая блок из первой компоненты в четвертую. Поэтому разбиение $\lambda = (6, 5, 4, 4, 3, 2, 1, 1)$ является графическим.

Отметим, что указанный ht-критерий графичности разбиения эквивалентен критерию Эрдеша-Галлаи.

Пусть (x, v, y) – тройка различных вершин графа $G = (V, E)$ такая, что $xv \in E$ и $vy \notin E$. Такую тройку назовем

1) *повышающей*, если $deg(x) \leq deg(y)$;

2) *понижающей*, если $deg(x) \geq 2 + deg(y)$;

3) *сохраняющей*, если $deg(x) = 1 + deg(y)$.

Рассмотрим преобразование φ графа G такое, что $\varphi(G) = G - xv + vy$, т.е. из G сначала удаляется ребро xv, а затем добавляется ребро vy (см. Рис. 1.3).

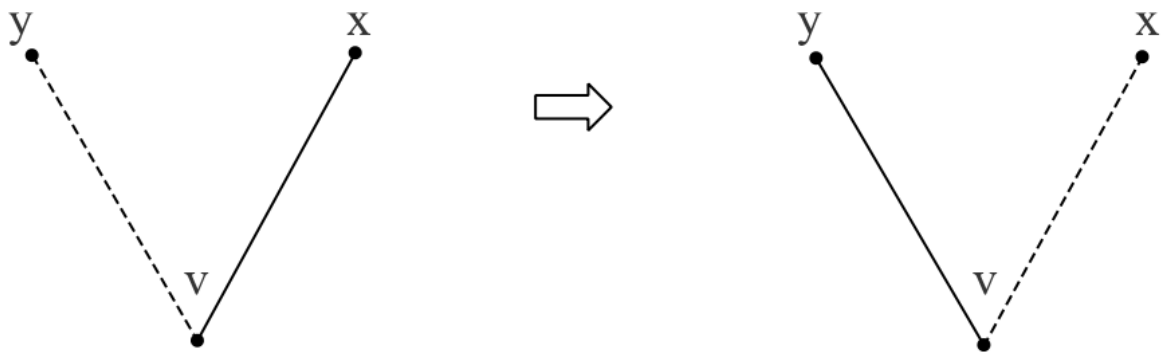


Рис. 1.3

Преобразование φ называется *вращением ребра* (в графе G вокруг вершины v), отвечающим тройке (x, v, y). Вращение ребра в графе φ(G), отвечающее тройке (y, v, x) называется *обратным вращением* ребра для вращения φ.

Вращение ребра в графе G, отвечающее тройке (x, v, y), называется

1) *повышающим,* если тройка (x, v, y) повышающая;

2) *понижающим*, если тройка (x, v, y) понижающая;

3) *сохраняющим*, если тройка (x, v, y) сохраняющая.

Важно заметить, что случаи, когда deg(x) = 1 или deg(y) = 0 рассматриваются в качестве допустимых, т. е. после вращения ребра может возникнуть изолированная вершина или вращение ребра произойдет в графе G с добавлением новой изолированной вершины.

Ясно, что вращение ребра в графе является повышающим тогда и только тогда, когда обратное к нему вращение является понижающим.

Пусть dpt(G) – графическое разбиение, отвечающее графу G, и φ – вращение ребра в графе G, отвечающее тройке (x, v, y), где $xv \in E$ и $vy \notin E$. Тогда не сложно заметить справедливость следующих утверждений.

1) Если φ – повышающее вращение ребра, то dpt(G) < dpt(φ(G)), причем dpt(G) получается из dpt(φ(G)) с помощью одного элементарного

преобразования первого типа, а G получается из $\varphi(G)$ с помощью обратного понижающего вращение ребра.

2) Если φ – понижающее вращение ребра, то $dpt(G) > dpt(\varphi(G))$, причем $dpt(\varphi(G))$ получается из $dpt(G)$ с помощью одного элементарного преобразования первого типа, а G получается из $\varphi(G)$ с помощью обратного повышающего вращение ребра.

3) Если φ – сохраняющее вращение ребра, то $dpt(G) = dpt(\varphi(G))$, причем G получается из $\varphi(G)$ с помощью обратного сохраняющего вращение ребра.

Пусть $\lambda = (\lambda_1, \ldots, \lambda_i, \ldots, \lambda_j, \ldots, \lambda_n)$ – графическое разбиение $1 \le i < j \le n$ и разбиение $\eta = (\lambda_1, \ldots, \lambda_i - 1, \ldots, \lambda_j + 1, \ldots, \lambda_n)$ получено из него с помощью одного элементарного преобразования первого типа. Покажем, что разбиение η также является графическим.

Действительно, пусть $G = (V, E)$ – реализация разбиения λ, $\deg(x) = \lambda_i$ и $\deg(y) = \lambda_j$, где $x, y \in V$. Поскольку $\lambda_i > \lambda_j$, существует вершина $v \in V$ такая, что $xv \in E$ и $vy \notin E$. Пусть φ – понижающее вращение ребра, отвечающее тройке (x, v, y) в графе G. Тогда, очевидно, $\eta = dpt(\varphi(G))$. Следовательно, разбиение η также является графическим.

Для любого натурального числа m множество NPL(2m) является решеткой относительно отношения доминирования (см., например, [2]). Поэтому в силу доказанного множество всех графических разбиений фиксированного веса 2m, где m – натуральное число, образует порядковый идеал в решетке $NPL(2m), \le$. Максимальные элементы этого порядкового идеала называют *максимальными графическими разбиениями* (веса 2m).

Через (V_1, E, V_2), где E не пусто, обозначают двудольный граф, имеющий две непересекающиеся непустые доли V_1, V_2 и множество ребер E, причем каждое ребро соединяет некоторую вершину из V_1 с некоторой вершиной из V_2. Конечно, предполагается, что $(V_1, E, V_2) = (V_2, E, V_1)$.

Пусть G_1 и G_2 – два графа с непересекающимися множествами вершин

V_1 и V_2, соответственно. Тогда через (G_1, E, G_2) обозначают объединение графов G_1 и G_2, дополненное множеством ребер E. Граф (V_1, E, V_2) называют *сэндвич-подграфом* графа (G_1, E, G_2).

Граф G называется *пороговым* (см., например, [4]), если его множество вершин представимо в виде объединения непересекающихся клики V_1 и коклики V_2, т.е. $V_1 \cap V_2 = \varnothing$, V_1 порождает полный подграф $K(V_1)$ и V_2 – нулевой подграф $O(V_2)$ в G, а множество окрестностей в G вершин из V_2 образует цепь подмножеств множества V_1 относительно теоретико-множественного включения.

Случаи $V_1 = \varnothing$ или $V_2 = \varnothing$ допускаются, т. е. полные и нулевые графы являются пороговыми.

Для порогового графа G множество всех ребер представимо в виде объединения множества всех ребер полного подграфа $K(V_1)$ и множества E всех его ребер, соединяющих вершины из V_1 с вершинами из V_2.

Таким образом, пороговый граф можно представить в виде $G = (K(V_1), E, O(V_2))$. Принято писать просто $G = (K(V_1), E, V_2)$. Двудольный подграф $H = (V_1, E, V_2)$ является его *сэндвич-подграфом*. В тривиальных случаях, когда $V_1 = \varnothing$ или $V_2 = \varnothing$, или V_2 состоит из изолированных вершин, сэндвич-подграф H является пустым подграфом в G.

Очевидно, добавление или удаление изолированных вершин не меняет свойство графа быть пороговым. Разнообразные равносильные определения пороговых графов и их многочисленные свойства можно найти в монографии [3].

В работе [4] доказано, что

Граф является пороговым тогда и только тогда, когда он не содержит повышающих троек вершин.

В этой же работе [4] также было доказано, что для произвольного

разбиения λ четного веса следующие условия эквивалентны:

1) λ является максимальным графическим разбиением,

2) $\lambda = gpt(G)$ для некоторого порогового графа G,

3) $hd(\lambda) = tl(\lambda)$.

Отметим, что любое максимальное графическое разбиение с точностью до изоморфизма и изолированных вершин единственным образом реализуется подходящим пороговым графом.

Пусть λ — произвольное графическое разбиение и G — некоторая его реализация. Совершая последовательно повышающие вращения ребер, пока это возможно, граф G мы преобразуем в некоторый пороговый граф H. Граф H находится по G, вообще говоря, неоднозначно. Положим $dpt(H) = \mu$. Ясно, что граф G получается из графа H с помощью обратных понижающих вращений ребер, производимых в обратном порядке, поэтому $\mu \geq \lambda$, $sum(\lambda) = sum(\mu)$ и μ является максимальным графическим разбиением.

Для произвольного графического разбиения λ все его реализации G, и только они, получаются из пороговых реализаций H подходящих максимальных графических разбиений μ таких, что $\mu \geq \lambda$ и $sum(\lambda) = sum(\mu)$, с помощью конечных последовательностей понижающих вращений ребер, отвечающих последовательности элементарных преобразований первого типа от μ до λ [4].

В связи с этими рассмотрениями возникает важный для теории разбиений

Открытый вопрос. Для произвольного графического разбиения λ построить алгоритм нахождения всех максимальных графических разбиений μ таких, что $\mu \geq \lambda$ и $sum(\mu) = sum(\lambda)$.

Основная цель данной магистерской диссертации состоит в решении указанного открытого вопроса, т.е.

нашей целью является построение ряда алгоритмов, с помощью которых можно найти все максимальные графические разбиения μ, которые доминируют заданное графическое разбиение λ и для каждого из которых выполняется $\mathrm{sum}(\mu) = \mathrm{sum}(\lambda)$.

Для достижения этой цели мы доказываем теорему 1, используя которую мы и строим алгоритм нахождения всех максимальных графических разбиений μ таких, что $\mu \geq \lambda$ и $\mathrm{sum}(\mu) = \mathrm{sum}(\lambda)$.

Каждое графическое разбиение определяется своим рангом, головой и хвостом. Поскольку для любого максимального графического разбиения μ выполняется $\mathrm{hd}(\mu) = \mathrm{tl}(\mu)$, для задания таких разбиений достаточно знать ранг и голову. Поэтому в первую очередь нужно изучить устройство голов для максимальных графических разбиений μ, доминирующих заданное графическое разбиение λ.

2 СПОСОБЫ И МЕТОДЫ ТЕОРЕТИЧЕСКИХ И АНАЛИТИЧЕСКИХ ИССЛЕДОВАНИЙ

2.1 Отношение частичного доминирования

Пусть $m \in \mathrm{N}$ — натуральное число. Для произвольного числа $t \in \mathrm{N}$ определим отношение $\leq_t$ на NPL(m), полагая $\lambda \leq_t \mu$ тогда и только тогда, когда $(\lambda_1, \lambda_2, ..., \lambda_t, 0, 0, ...) \leq (\mu_1, \mu_2, ..., \mu_t, 0, 0, ...)$, т. е. когда

$$\begin{gathered} \lambda_1 \leq \mu_1, \\ \lambda_1 + \lambda_2 \leq \mu_1 + \mu_2, \\ ..., \\ \lambda_1 + \lambda_2 + ... + \lambda_t \leq \mu_1 + \mu_2 + ... + \mu_t. \end{gathered} \tag{2.1}$$

Лемма 1. Пусть λ и μ — произвольные разбиения. Положим $\ell(\mu) = l$. Пусть $\lambda \leq_l \mu$ и $sum(\lambda) \leq sum(\mu)$. Тогда $\lambda \leq \mu$.

Доказательство. Неравенства

$$\begin{gathered} \lambda_1 \leq \mu_1, \\ \lambda_1 + \lambda_2 \leq \mu_1 + \mu_2, \\ ..., \\ \lambda_1 + \lambda_2 + ... + \lambda_l \leq \mu_1 + \mu_2 + ... + \mu_l. \end{gathered} \tag{2.2}$$

верны по определению $\lambda \leq_l \mu$. По условию выполняется $\ell(\mu) = l$, поэтому $\mu_1 + . . . + \mu_l = \mathrm{sum}(\mu)$. Тогда выполняются и неравенства

$$\begin{gathered} \lambda_1 + ... + \lambda_{l+1} \leq \mu_1 + ... + \mu_{l+1}, \\ \lambda_1 + ... + \lambda_{l+2} \leq \mu_1 + ... + \mu_{l+2}, \\ ... \end{gathered} \tag{2.3}$$

так как сумма в правой части неравенств равна sum(μ), а сумма в левой части не превосходит sum(λ), что в свою очередь меньше или равно sum(μ) по

условию. Системы 2.2 и 2.3 вместе дают неравенство $\lambda \leq \mu$. □

2.2 Максимальные графические разбиения фиксированного ранга

Через $MGP_r(2m)$ обозначим множество всех максимальных графических разбиений веса $2m$ и ранга r. Пусть $\mu \in MGP_r(2m)$. Поскольку $hd(\mu) = tl(\mu)$, то $sum(\mu) = 2sum(hd(\mu)) + r \cdot (r-1)$, следовательно,

$$sum(hd(\mu)) = \frac{sum(\mu)-r(r-1)}{2} = m - \frac{r(r-1)}{2} \geq r. \quad (2.4)$$

Отсюда имеем $2m \geq 2r + r(r-1)$, т. е. $0 \geq r^2 + r - 2m$ и поэтому

$$r \leq \left\lfloor \frac{\sqrt{8m+1}-1}{2} \right\rfloor. \quad (2.5)$$

Обозначим через $NPL(m, t)$, где $m \geq t$, множество всех разбиений веса m и длины t. Нетрудно видеть, что решетка $NPL(m, t)$ является интервалом решетки $MPL(m)$. Обозначим через $minp(m, t)$ наименьшее разбиение веса m и длины t, а через $maxp(m, t)$ — наибольшее разбиение веса m и длины t. Разбиение $maxp(m, t)$ имеет вид $(m - t + 1, 1, 1, \ldots, 1)$, разбиение $minp(m, t)$ — $(\lceil \frac{m}{t} \rceil, \ldots, \lceil \frac{m}{t} \rceil, \lfloor \frac{m}{t} \rfloor, \ldots, \lfloor \frac{m}{t} \rfloor)$, где $\lceil \frac{m}{t} \rceil$ повторяется $m \pmod t$ раз. диаграмма Ферре разбиения $minp(m, t)$ строится добавлением блоков по рядам снизу вверх и в каждом ряду — слева направо. На рис. 2.1 приведены диаграммы Ферре разбиений $minp(8, 3) = (3, 3, 2)$ и $maxp(8, 3) = (6, 1, 1)$.

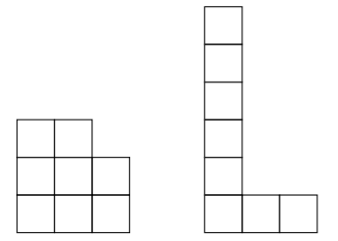


Рис. 2.1

Пусть λ — фиксированное графическое разбиение веса $2m$. Обозначим через $hdMGP_r(\lambda)$ множество всех голов максимальных графических разбиений μ веса $2m$ и ранга r таких, что $\lambda \leq \mu$. Отметим, что выполнение

равенства $r(\lambda) = r$ не предполагается. Поскольку $\mathrm{sum}(\mathrm{hd}(\mu)) = m - \frac{r(r-1)}{2}$, то $\mathrm{hdMGP}_r(\lambda) \subseteq \mathrm{NPL}(m - \frac{r(r-1)}{2}, r)$. Далее будет показано, что любое непустое множество вида $\mathrm{hdMGP}_r(\lambda)$ является подрешеткой решетки $\mathrm{NPL}(m - \frac{r(r-1)}{2}, r)$ и даже ее интервалом.

2.3 Проверка доминирования с помощью частичного сравнения

Лемма 2. Пусть λ и μ — разбиения веса m, $r(\lambda) = r$, $r(\mu) = s$, $\lambda \leq_s \mu$ и $\mu^* \leq_s \lambda^*$. Тогда $\lambda \leq \mu$.

Доказательство. Пусть λ' и μ' — разбиения, диаграммы Ферре которых получаются из диаграмм Ферре разбиений λ и μ, соответственно, переносом всех блоков из первых s столбцов и всех блоков выше s-строки в $(s + 1)$-столбец. При этом $(s + 1)$-столбец сделаем первым. Поскольку $r(\mu) = s$, выполняется $\mu_{s+1} \leq s$, отсюда следует, что в диаграмме Ферре разбиения μ нет блоков, лежащих выше s-строки и расположенных правее s-столбца. Ясно, что диаграмма μ' получается из диаграммы μ просто переносом всех блоков из первых s столбцов в $(s + 1)$-столбец, и выполняются равенства

$$\begin{gathered} \mu'_1 = \mu_1 + \mu_2 + \ldots + \mu_{s+1}, \\ \mu'_1 + \mu'_2 = \mu_1 + \mu_2 + \ldots + \mu_{s+1}, \\ \ldots \\ \mu'_1 + \mu'_2 + \ldots + \mu'_i = \mu_1 + \mu_2 + \ldots + \mu_{s+i}. \\ \ldots \end{gathered} \tag{2.6}$$

В диаграмме Ферре разбиения λ могут быть блоки, лежащие выше s-строки и расположенные правее $(s + 1)$-столбца. Тогда диаграмма λ' получается из диаграммы λ, кроме переносов всех блоков из первых s столбцов в $(s + 1)$-столбец, еще и переносом всех блоков, изначально находившихся выше s-строки и правее $(s + 1)$-столбца, в $(s + 1)$-столбец,

который становится первым. Каждый перенос блока влево увеличивает разбиение (см. [9]), следовательно, выполняются неравенства

$$\lambda'_1 \le \lambda_1 + \lambda_2 + \ldots + \lambda_{s+1},$$

$$\lambda'_1 + \lambda'_2 \le \lambda_1 + \lambda_2 + \ldots + \lambda_{s+1},$$

$$\ldots \qquad (2.7)$$

$$\lambda'_1 + \lambda'_2 + \ldots + \lambda'_i \le \lambda_1 + \lambda_2 + \ldots + \lambda_{s+i}.$$

$$\ldots$$

На рис. 2.2 приведены диаграммы Ферре разбиений $\lambda = (5, 4, 4, 4, 4, 3)$ и $\mu = (6, 6, 4, 3, 3, 2)$, где $m = 24$, $r = 4$, $s = 3$, а также диаграммы Ферре соответствующих разбиений $\lambda' = (18, 3, 3)$ и $\mu' = (19, 3, 2)$.

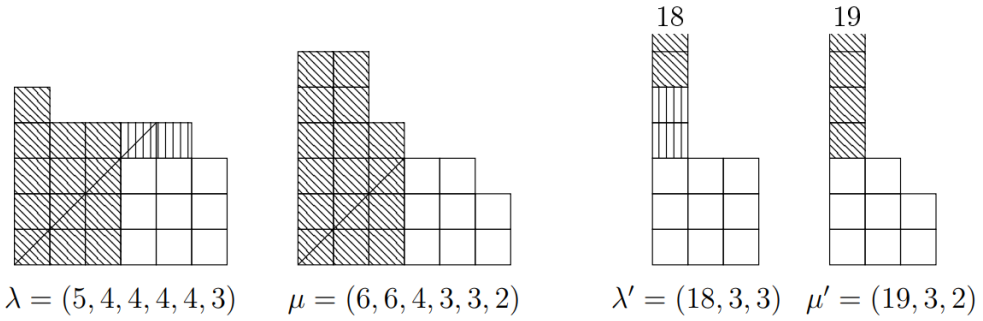


Рис. 2.2

Рассмотрим сопряженные разбиения $\lambda^* = (\lambda^*_1, \lambda^*_2, \ldots)$ и $\mu^* = (\mu^*_1, \mu^*_2, \ldots)$. Пусть $\lambda^{(1)} = (\lambda^*_1, \lambda^*_2, \ldots, \lambda^*_s, 0, 0, \ldots)$ и $\mu^{(1)} = (\mu^*_1, \mu^*_2, \ldots, \mu^*_s, 0, 0, \ldots)$ — разбиения, первые s компонент которых равны первым s компонентам разбиений λ^* и μ^*, соответственно, а остальные компоненты равны нулю. Так как по условию $\mu^* \le_s \lambda^*$, верно соотношение $\mu^{(1)} \le \lambda^{(1)}$. Также выполняется неравенство $\ell(\lambda^{(1)}) \le s$, а для $\mu^{(1)}$ верно равенство $\ell(\mu^{(1)}) = s$, так как $r(\mu) = s$.

Далее проведем следующую цепочку преобразований.

1) Припишем $\mathrm{sum}(\lambda^{(1)}) - \mathrm{sum}(\mu^{(1)})$ единичных компонент в конец разбиения $\mu^{(1)}$, а $\lambda^{(1)}$ оставим без изменений. Отметим, что число $\mathrm{sum}(\lambda^{(1)}) - \mathrm{sum}(\mu^{(1)})$ неотрицательно, так как $\lambda^{(1)} \ge \mu^{(1)}$. Положим $\lambda^{(2)} = \lambda^{(1)}$, $\mu^{(2)} =$

$(\mu^*_1, \mu^*_2, ..., \mu^*_s, 1, ..., 1, 0, ...)$, где единица записана $\mathrm{sum}(\lambda^{(1)}) - \mathrm{sum}(\mu^{(1)})$ раз. Теперь $\mathrm{sum}(\lambda^{(2)}) = \mathrm{sum}(\mu^{(2)})$. Обозначим $\ell(\lambda^{(1)})$ через l, это число также равно $\ell(\lambda^{(2)})$. Так как $\mu^{(1)} \le \lambda^{(1)}$ и $l \le \ell(\mu^{(1)})$, верно соотношение $\mu^{(2)} \le_l \lambda^{(2)}$. Тогда по лемме 1 выполняется соотношение $\mu^{(2)} \le \lambda^{(2)}$.

2) Переходя к сопряженным разбиениям, положим $\lambda^{(3)} = \lambda^{(2)*}$ и $\mu^{(3)} = \mu^{(2)*}$. Так как $\mathrm{sum}(\lambda^{(2)}) = \mathrm{sum}(\mu^{(2)})$ и $\mu^{(2)} \le \lambda^{(2)}$, верно соотношение $\lambda^{(3)} \le \mu^{(3)}$.

3) Увеличим первую компоненту обоих разбиений на $m - \mathrm{sum}(\lambda^{(3)})$. Отметим, что число $m - \mathrm{sum}(\lambda^{(3)})$ неотрицательно, поскольку $m \ge \mathrm{sum}(\lambda^{(1)}) = \mathrm{sum}(\lambda^{(2)}) = \mathrm{sum}(\lambda^{(3)})$. В результате получим разбиения $\lambda^{(4)}$ и $\mu^{(4)}$, где $\mathrm{sum}(\lambda^{(4)}) = \mathrm{sum}(\mu^{(4)}) = m$ и $\lambda^{(4)} \le \mu^{(4)}$.

4) Для каждого из разбиений $\lambda^{(4)}$ и $\mu^{(4)}$, удалим первые s компонент и прибавим их сумму к $(s + 1)$-й компоненте соответствующего разбиения, при этом $(s + 1)$-ю компоненту сделаем первой. На диаграмме Ферре это соответствует перемещению всех блоков из первых s столбцов в $(s + 1)$-столбец. Получим разбиения $\lambda^{(5)}$ и $\mu^{(5)}$, и соотношение $\lambda^{(5)} \le \mu^{(5)}$ сохраняется, так как i-е частичные суммы разбиений $\lambda^{(5)}$ и $\mu^{(5)}$ — это $(s + i)$-е частичные суммы разбиений $\lambda^{(4)}$ и $\mu^{(4)}$, соответственно. Отметим, что $\mathrm{sum}(\lambda^{(5)}) = \mathrm{sum}(\mu^{(5)}) = m$, а диаграммы Ферре разбиений $\lambda^{(5)}$ и $\mu^{(5)}$ отличаются от диаграмм исходных разбиений λ и μ удалением всех блоков выше s-строки, удалением первых s столбцов и добавлением блоков по числу удаленных в новый первый столбец. Таким образом, верно $\lambda^{(5)} = \lambda'$ и $\mu^{(5)} = \mu'$. Следовательно, выполняется соотношение $\lambda' \le \mu'$.

На рис. 2.3 показаны диаграммы Ферре разбиений $\lambda = (5, 4, 4, 4, 4, 3)$ и $\mu = (6, 6, 4, 3, 3, 2)$, а также всех разбиений, получающихся в процессе преобразований.

По определению отношения доминирования из соотношения $\lambda' \le \mu'$ вытекают неравенства

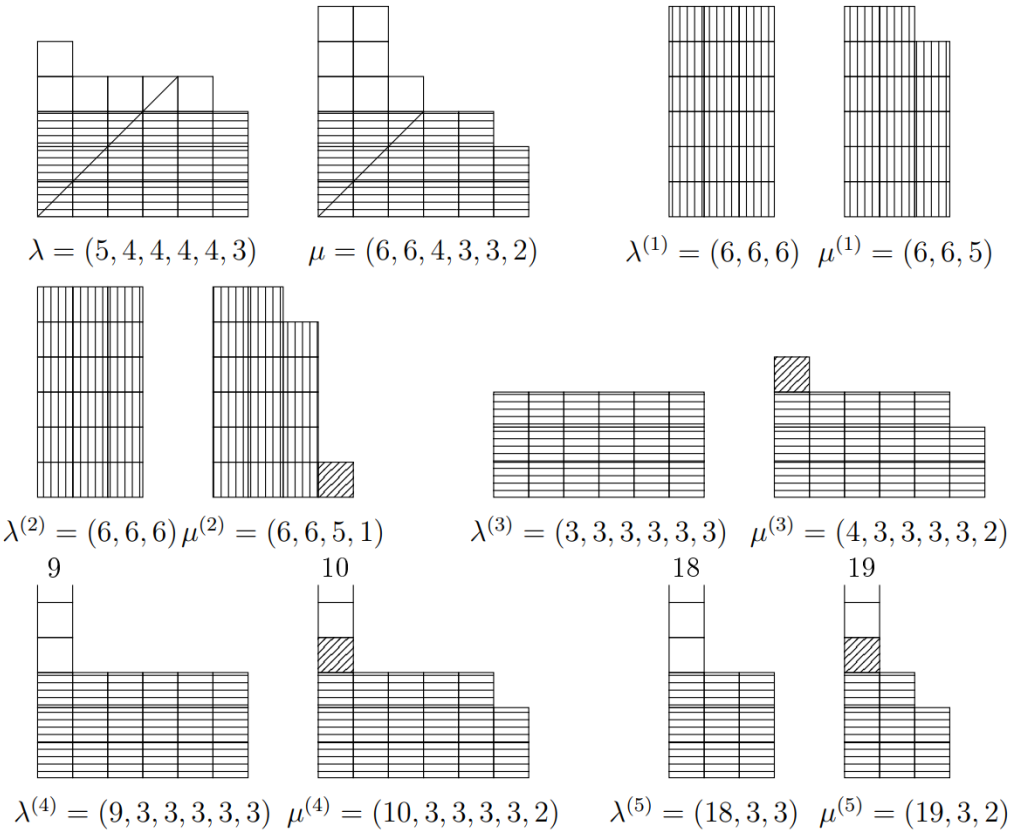


Рис. 2.3

$$\lambda'_1 \leq \mu'_1,$$

$$\lambda'_1 + \lambda'_2 \leq \mu'_1 + \mu'_2,$$

$$\ldots \qquad (2.8)$$

$$\lambda'_1 + \lambda'_2 + \ldots + \lambda'_i \leq \mu'_1 + \mu'_2 + \ldots + \mu'_i.$$

$$\ldots$$

Совмещая эту систему с системами 2.6 и 2.7, получаем неравенства

$$\lambda_1 + \lambda_2 + \ldots + \lambda_{s+1} \leq \mu_1 + \mu_2 + \ldots + \mu_{s+1},$$

$$\lambda_1 + \lambda_2 + \ldots + \lambda_{s+2} \leq \mu_1 + \mu_2 + \ldots + \mu_{s+2},$$

$$\ldots \qquad (2.9)$$

$$\lambda_1 + \lambda_2 + \ldots + \lambda_{s+i} \leq \mu_1 + \mu_2 + \ldots + \mu_{s+i}.$$

$$\ldots$$

Вместе с условием $\lambda \leq_s \mu$ это дает $\lambda \leq \mu$.

Подставляя в условие леммы 2 разбиения μ^* и λ^* вместо λ и μ, соответственно, и r вместо s в соотношения, получаем □

Следствие 1. Пусть λ и μ — разбиения веса m, $r(\lambda) = r$, $r(\mu) = s$, $\lambda \leq_r \mu$ и $\mu^* \leq_r \lambda^*$. Тогда $\lambda \leq \mu$.

2.4 Удаление строк и столбцов диаграммы Ферре

Обозначим через $\mathrm{cut}(\lambda; t, p)$, где $t, p \in \mathrm{N}_0$, разбиение, которое получается из разбиения λ уменьшением первых t компонент на одно и то же число p и обнулением всех компонент с номерами $t + 1$, $t + 2$, … . Диаграмма Ферре разбиения $\mathrm{cut}(\lambda; t, p)$ получается из диаграммы Ферре разбиения λ удалением всех столбцов, начиная с $(t + 1)$-столбца, и первых p строк. Таким образом, $\mathrm{cut}(\lambda; t, p) = (\max(0, \lambda_1 - p), \max(0, \lambda_2 - p), \ldots, \max(0, \lambda_t - p), 0, 0, \ldots)$. На рис. 2.4 представлены диаграммы Ферре разбиений $\lambda = (5, 4, 4, 3, 2, 1, 1)$ и $\mathrm{cut}(\lambda; 4, 1) = (4, 3, 3, 2)$.

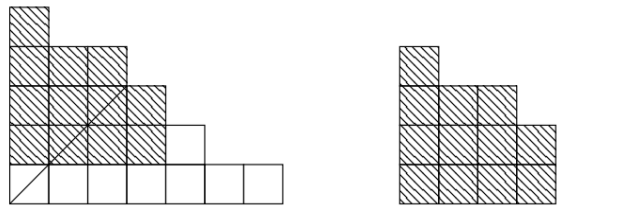


Рис. 2.4

Пусть λ — разбиение веса m и $r(\lambda) = r$. Пусть $t \in \mathrm{N}$ — произвольное натуральное число. Обозначим разбиение $\mathrm{cut}(\lambda; t, t - 1)$ через $\mathrm{hd}_t(\lambda)$ и разбиение $\mathrm{cut}(\lambda^*; t, t)$ через $\mathrm{tl}_t(\lambda)$. Для получения диаграммы Ферре разбиения $\mathrm{hd}_t(\lambda)$ нужно из диаграммы Ферре разбиения λ удалить первые $t - 1$ строк и оставить только первые t столбцов. Для получения диаграммы Ферре разбиения $\mathrm{tl}_t(\lambda)$ нужно диаграмму Ферре разбиения λ отразить относительно главной диагонали, и затем удалить первые t строк и оставить только первые t столбцов. В частности, верно $\mathrm{hd}_r(\lambda) = \mathrm{hd}(\lambda)$ и $\mathrm{tl}_r(\lambda) = \mathrm{tl}(\lambda)$. Обратим внимание,

что для достаточно больших t выполняется $\mathrm{hd}_t(\lambda) = \mathrm{tl}_t(\lambda) = (0, 0, \ldots)$.

В некотором смысле, $\mathrm{hd}_t(\lambda)$ и $\mathrm{tl}_t(\lambda)$ — это голова и хвост разбиения λ, если бы у него был ранг t. Например, на рис. 2.5 представлена диаграмма Ферре разбиения $\lambda = (5, 4, 4, 3, 2, 1, 1)$ ранга 3, а также разбиений $\mathrm{hd}_2(\lambda) = (4, 3)$ и $\mathrm{tl}_2(\lambda) = (5, 3)$.

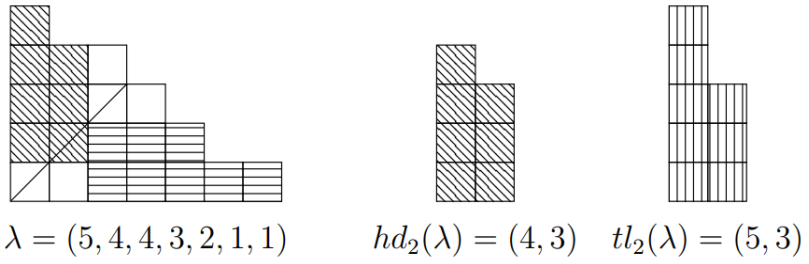


Рис. 2.5

Лемма 3. Пусть λ и μ — произвольные разбиения одинакового веса m такие, что $\lambda \leq \mu$. Пусть $t, p \in \mathrm{N}_0$ — произвольные целые неотрицательные числа. Тогда $\mathrm{cut}(\lambda; t, p) \leq \mathrm{cut}(\mu; t, p)$.

Доказательство. Обозначим $\mathrm{cut}(\lambda; t, p)$ через λ' и $\mathrm{cut}(\mu; t, p)$ через μ'. Положим $l_1 = \ell(\lambda')$ и $l_2 = \ell(\mu')$. По определению cut числа l_1 и l_2 не превосходят t.

Случай 1. $l_1 \leq l_2$.

В этом случае первые l_1 компонент разбиений λ' и μ' получаются из первых l_1 компонент λ и μ, соответственно, уменьшением на p, поэтому из $\lambda \leq \mu$ следует $\lambda \leq_{l1} \mu$. При этом все компоненты разбиения λ' с номерами больше числа l_1 равны нулю. Следовательно, $\lambda' \leq \mu'$.

Случай 2. $l_1 > l_2$.

Аналогично первому случаю, выполняется $\lambda \leq_{l2} \mu$. Первые l_1 компонент разбиения λ' получаются из первых l_1 компонент разбиения λ просто уменьшением их на число p. Отсюда верно равенство

$$sum(\lambda') = \lambda_1 + \lambda_2 + \ldots + \lambda_{l_1} - l_1 \cdot p. \qquad (2.10)$$

Разбиение μ' равно $(\max(0, \mu_1 - p), \max(0, \mu_2 - p), \ldots, \max(0, \mu_t - p))$ и

одновременно с этим оно равно ($\mu_1 - p$, $\mu_2 - p$, …, $\mu_{l2} - p$, 0, 0, …). Следовательно, компоненты разбиения μ' с номерами от ($l_2 + 1$) до l_1 меньше не более чем на p соответствующих компонент разбиения μ. Отсюда выполняется неравенство

$$sum(\mu') \geq \mu_1 + \mu_2 + \ldots + \mu_{l_1} - l_1 \cdot p. \quad (2.11)$$

Наконец, из того, что $\lambda \leq \mu$, следует

$$\lambda_1 + \lambda_2 + \ldots + \lambda_{l_1} - l_1 \cdot p \leq \mu_1 + \mu_2 + \ldots + \mu_{l_1} - l_1 \cdot p. \quad (2.12)$$

Совмещая выражения 2.10, 2.11 и 2.12, получаем неравенство $\text{sum}(\lambda') \leq \text{sum}(\mu')$. Вместе с ранее доказанным соотношением $\lambda' \leq_{l2} \mu'$ по лемме 1 это дает $\lambda' \leq \mu'$. □

Следствие 2. Пусть λ и μ — произвольные разбиения одинакового веса m такие, что $\lambda \leq \mu$. Тогда для любого $t \in \mathbb{N}$ выполняется $\text{hd}_t(\lambda) \leq \text{hd}_t(\mu)$ и $\text{tl}_t(\lambda) \geq \text{tl}_t(\mu)$.

Отметим, что неравенство $\text{tl}_t(\lambda) \leq \text{tl}_t(\mu)$ выводится из леммы 3 подстановкой μ^* и λ^* вместо λ и μ, соответственно.

2.5 Проверка доминирования с помощью сравнения голов и хвостов

Лемма 4. Пусть λ и μ — произвольные ненулевые разбиения одинакового веса m, $r(\lambda) = r$, $r(\mu) = s$. Пусть

$$hd_s(\lambda) \leq hd(\mu) \text{ и } tl_s(\lambda) \geq tl(\mu). \quad (2.13)$$

Тогда $\lambda \leq \mu$.

Доказательство. Случай 1. $r \geq s$.

В этом случае головы $\text{hd}_s(\lambda)$ и $\text{hd}(\mu)$ получаются уменьшением первых s компонент разбиений λ и μ, соответственно, на одно и то же число $s - 1$. Тогда из соотношения $\text{hd}_s(\lambda) \leq \text{hd}(\mu)$ следует $\lambda \leq_s \mu$. Аналогично, из $\text{tl}_s(\lambda) \geq \text{tl}(\mu)$ следует $\lambda^* \geq_s \mu^*$. Отсюда по лемме 2 верно $\lambda \leq \mu$.

Случай 2. $r < s$.

Положим $l_1 = \ell(\mathrm{hd}_s(\lambda))$ и $l_2 = \ell(\mathrm{tl}_s(\lambda))$. В этом случае только первые l_1 компонент разбиения $\mathrm{hd}_s(\lambda)$ получаются уменьшением на $s-1$ соответствующих компонент разбиения λ, и только первые l_2 компонент разбиения $\mathrm{tl}_s(\lambda)$ получаются уменьшением на s соответствующих компонент разбиения λ^*. Аналогично первому случаю выполняется $\lambda \leq_{l1} \mu$ и $\lambda^* \geq_{l2} \mu^*$.

Поскольку для всех $i \in \mathrm{N}$, таких что $l_1 < i \leq r$, выполняется $\lambda_i \leq s$ и $\mu_i \geq s$, из неравенства

$$\lambda_1 + \lambda_2 + \ldots + \lambda_{l_1} \leq \mu_1 + \mu_2 + \ldots + \mu_{l_1} \tag{2.14}$$

следуют неравенства

$$\begin{gathered}
\lambda_1 + \lambda_2 + \ldots + \lambda_{l_1+1} \leq \mu_1 + \mu_2 + \ldots + \mu_{l_1+1}, \\
\lambda_1 + \lambda_2 + \ldots + \lambda_{l_1+2} \leq \mu_1 + \mu_2 + \ldots + \mu_{l_1+2}, \\
\ldots \\
\lambda_1 + \lambda_2 + \ldots + \lambda_r \leq \mu_1 + \mu_2 + \ldots + \mu_r.
\end{gathered} \tag{2.15}$$

Таким образом, $\lambda \leq_r \mu$.

Рассмотрим сопряженные разбиения $\lambda^* = (\lambda^*_1, \lambda^*_2, \ldots)$ и $\mu^* = (\mu^*_1, \mu^*_2, \ldots)$. Поскольку $\mathrm{r}(\lambda) = r$, то

$$m = \lambda_1 + \ldots + \lambda_r + \lambda_1^* + \ldots + \lambda_r^* - r^2. \tag{2.16}$$

Пусть k — количество блоков в диаграмме Ферре разбиения μ, расположенных правее r-столбца и выше r-строки. Число k положительно, так как $s > r$. Тогда верно равенство

$$m = \mu_1 + \ldots + \mu_r + \mu_1^* + \ldots + \mu_r^* - r^2 + k. \tag{2.17}$$

На рис. 2.6 изображены разбиения $\lambda = (6, 6, 5, 3, 3, 3, 3, 2, 2, 1)$ и $\mu = (6, 6, 6, 5, 5, 4, 2)$, где $m = 34$, $r = 3$, $s = 5$ и $\lambda \leq \mu$. Для разбиения λ выполняется

$hd_s(\lambda) = (2, 2, 1)$, $tl_s(\lambda) = (5, 4, 2)$, $l_1 = 3$ и $l_2 = 3$. Для разбиения μ выполняется $hd(\mu) = (2, 2, 2, 1, 1)$, $tl(\mu) = (2, 2, 1, 1)$, $k = 5$.

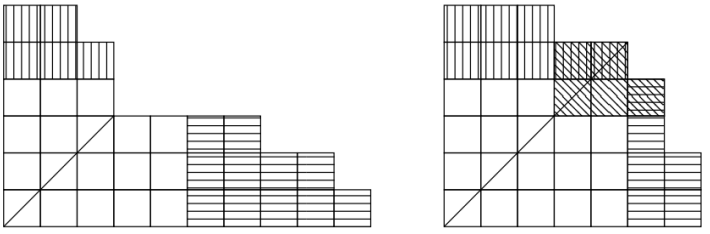


Рис. 2.6

Из ранее доказанного соотношения $\lambda \leq_r \mu$ следует, что $\lambda_1 + \ldots + \lambda_r \leq \mu_1 + \ldots + \mu_r$. Совмещая это неравенство с формулами 2.16 и 2.17, получаем

$$\lambda_1^* + \ldots + \lambda_r^* - r^2 \geq \mu_1^* + \ldots + \mu_r^* - r^2 + k. \tag{2.18}$$

Отсюда следует неравенство

$$\lambda_1^* + \ldots + \lambda_r^* \geq \mu_1^* + \ldots + \mu_r^*. \tag{2.19}$$

Поскольку для всех $i \in \mathrm{N}$, таких что $l_2 < i \leq r$, выполняется $\lambda^*_i \leq s$ и $\mu^*_i \geq s$, из неравенства 2.19 следуют неравенства

$$\begin{gathered}
\lambda_1^* + \ldots + \lambda_{r-1}^* \geq \mu_1^* + \ldots + \mu_{r-1}^*, \\
\lambda_1^* + \ldots + \lambda_{r-2}^* \geq \mu_1^* + \ldots + \mu_{r-2}^*, \\
\ldots \\
\lambda_1^* + \ldots + \lambda_{l_2+1}^* \geq \mu_1^* + \ldots + \mu_{l_2+1}^*.
\end{gathered} \tag{2.20}$$

Эта система и ранее доказанное соотношение $\lambda^* \geq_{l2} \mu^*$ вместе дают $\lambda^* \geq_r \mu^*$. Итак, $\lambda \leq_r \mu$ и $\lambda^* \geq_r \mu^*$, отсюда по следствию 1 выполняется $\lambda \leq \mu$.

□

2.6 Основная теорема

Теорема 1. Пусть λ — произвольное графическое разбиение веса $2m$ и

ранга r. Пусть $t \in \mathrm{N}$ – произвольное натуральное число. Положим

$$\eta_1 = hd_t(\lambda) \vee minp(m - \tfrac{t(t-1)}{2}, t) \text{ и}$$

$$\eta_2 = tl_t(\lambda) \wedge maxp(m - \tfrac{t(t-1)}{2}, t). \qquad (2.21)$$

Тогда

1) если $\eta_1 \leq \eta_2$, то множество hdMGPt(λ) голов максимальных графических разбиений веса $2m$ и ранга t, лежащих над λ, является интервалом решетки NPL($m - \frac{t(t-1)}{2}$, t) от η_1 до η_2.

2) Если $\eta_1 \not\leqslant \eta_2$, то не существует максимальных графических разбиений веса $2m$ и ранга t, лежащих над λ, т.е. множество hdMGPt(λ) пусто.

Доказательство. Разобьем доказательство на две части.

1) Докажем, что любая голова максимального графического разбиения веса $2m$ и ранга t, доминирующего λ, лежит в интервале от η_1 до η_2.

Пусть μ — произвольное максимальное графическое разбиение веса $2m$ и ранга t, доминирующее λ. Положим $\eta = \mathrm{hd}(\mu) = \mathrm{tl}(\mu)$. Так как $\mu \in \mathrm{MGP}_t(2m)$, имеем $\eta \in \mathrm{NPL}(m - \frac{t(t-1)}{2}, t)$. Следовательно,

$$minp(m - \tfrac{t(t-1)}{2}, t) \leq \eta \leq maxp(m - \tfrac{t(t-1)}{2}, t). \qquad (2.22)$$

Кроме того, в силу следствия 2, $\mathrm{hd}_t(\lambda) \leq \mathrm{hd}_t(\mu)$ и $\mathrm{tl}_t(\lambda) \geq \mathrm{tl}_t(\mu)$. Так как $\mathrm{hd}_t(\mu) = \mathrm{tl}_t(\mu) = \eta$, отсюда выводим

$$hd_t(\lambda) \leq \eta \leq tl_t(\lambda). \qquad (2.23)$$

Совмещая двойные неравенства 2.22 и 2.23, получаем $\eta_1 \leq \eta \leq \eta_2$. Итак, если $\eta \in \mathrm{hdMGP}_t(\lambda)$, то $\eta_1 \leq \eta \leq \eta_2$.

2) Докажем, что любое разбиение, лежащее в интервале от η_1 до η_2, является головой максимального графического разбиения веса $2m$ и ранга t, доминирующего λ.

Пусть η — любое такое разбиение, что $\eta_1 \leq \eta \leq \eta_2$. Тогда верно minp($m - \frac{t(t-1)}{2}$, t) $\leq \eta \leq$ maxp($m - \frac{t(t-1)}{2}$, t). Следовательно, выполняется $\eta \in$ NPL($m -$

$\frac{t(t-1)}{2}$, t). Обозначим через μ максимальное графическое разбиение с головой η. Тогда верны равенства $r(\mu) = \ell(\eta) = t$ и $sum(\mu) = 2 \cdot sum(\eta) + t \cdot (t-1) = 2m$, поэтому $\mu \in MGP_t(2m)$. Из двойного неравенства $\eta_1 \leq \eta \leq \eta_2$ также следует, что $hd_t(\lambda) \leq \eta \leq tl_t(\lambda)$. Поскольку $hd(\mu) = tl(\mu) = \eta$, $hd_t(\lambda) \leq hd(\mu)$ и $tl(\mu) \geq tl_t(\lambda)$, следовательно, по лемме 4 верно $\lambda \leq \mu$. Таким образом, если $\eta_1 \leq \eta \leq \eta_2$, то $\eta \in hdMGP_t(\lambda)$.

Итак, множество $hdMGP_t(\lambda)$ голов максимальных графических разбиений веса $2m$ и ранга t, лежащих над λ, совпадает с множеством всех разбиений η таких, что $\eta_1 \leq \eta \leq \eta_2$. Ясно, что если выполняется $\eta_1 \not\leq \eta_2$, то не существует максимальных графических разбиений веса $2m$ и ранга t, лежащих над λ. □

2.7 Ранги максимального графического доминирования

Пусть λ — произвольное графическое разбиение веса $2m$ и ранга r. Основная теорема позволяет для произвольного натурального числа t построить множество $hdMGP_t(\lambda)$. Чтобы построить множество всех максимальных графических разбиений веса $2m$, доминирующих λ, нужно перебрать все возможные ранги, для которых множество $hdMGP_t(\lambda)$ непусто.

Для фиксированного разбиения λ возьмём число $t \in N_0$. Если множество $hdMGP_t(\lambda)$ непусто, назовем *t рангом максимального графического доминирования* λ. Тогда часть поставленной задачи — найти множество всех рангов максимального графического доминирования λ.

Согласно неравенству 2.5, для всех рангов максимального графического доминирования выполняется $t \leq \lfloor \frac{\sqrt{8m+1}-1}{2} \rfloor$. В статье [5] доказано, что число $r = r(\lambda)$ обязательно является рангом максимального графического доминирования.

В качестве примера рассмотрим разбиение λ = (5, 4, 4, 2, 2, 2, 1, 0), где m = 10. Его доминируют четыре максимальных графических разбиения (5, 4, 4,

3, 3, 1), (5, 5, 3, 3, 2, 2), (6, 4, 3, 3, 2, 1, 1) и (6, 5, 2, 2, 2, 2, 1) веса 20. Таким образом, множество рангов максимального графического доминирования λ имеет только числа 2 и 3.

Теорема 2. Пусть λ — произвольное графическое разбиение веса $2m$ и ранга r. Тогда множество рангов максимального графического доминирования λ является отрезком целых неотрицательных чисел.

Доказательство. Опишем процесс, который пометит все разбиения полурешетки GPS($2m$). Устроим процесс так, чтобы доказать утверждение теоремы для всех помеченных графических разбиений.

Если λ — максимальное графическое разбиение, то множество рангов максимального графического разбиения λ состоит из единственного числа $r(\lambda)$. Тогда оно является вырожденным отрезком. Таким образом, можно пометить все максимальные графические разбиения.

Если λ — не максимальное графическое разбиение, то рассмотрим все графические разбиения, покрывающие λ в полурешетке GPS(2m). Если какое-то из них не помечено, возьмем его в качестве текущего разбиения, и так далее. Поскольку множество GPS(2m) конечно, этот процесс за конечное число шагов придет в такое разбиение, что все покрывающие его графические разбиения помечены.

Рассмотрим все графические разбиения $\mu_1, \mu_2, \ldots, \mu_n$, покрывающие λ в полурешетке GPS($2m$). Поскольку эти разбиения покрывают λ, то любое максимальное разбиение из MGP($2m$), доминирующее λ, будет доминировать одно из разбиений μ_i. Тогда множество рангов доминирования λ — это объединение множеств рангов доминирования $\mu_1, \mu_2, \ldots, \mu_n$.

В [8] доказано, что μ_i покрывает λ тогда и только тогда, когда λ получается из μ_i одним элементарным преобразованием. Элементарное преобразование заключается в удалении одного блока и добавлении одного блока диаграммы Ферре. Удаление и добавление блока изменяют ранг разбиения не более чем на 1, следовательно $r(\lambda) - 1 \le r(\mu_i) \le r(\lambda) + 1$.

Таким образом, если все разбиения μ_i помечены, то множество рангов

максимального графического доминирования λ — это объединение отрезков целых неотрицательных чисел. Как отмечено выше (см. [5]), для разбиения μ_i число $r(\mu_i)$ является рангом максимального графического доминирования. Таким образом, множество рангов максимального графического доминирования λ — это объединение нескольких отрезков целых чисел, каждый из которых содержит хотя бы одно из трех чисел $r(\lambda) - 1$, $r(\lambda)$ и $r(\lambda) + 1$. Наконец, множество рангов максимального графического доминирования λ само должно содержать $r(\lambda)$.

Следовательно, множество рангов максимального графического доминирования λ представляет собой отрезок целых неотрицательных чисел. Таким образом, описываемый нами процесс пометит любое разбиение λ. □

По теореме 2, чтобы перебрать все максимальные графические разбиения, доминирующие λ и имеющие такой же вес, можно начать с множества $\mathrm{hdMGP}_r(\lambda)$, которое гарантированно непусто. Затем перебирать множества $\mathrm{hdMGP}_t(\lambda)$, двигая t влево и вправо от r. Если с одного из концов встретится пустое множество $\mathrm{hdMGP}_t(\lambda)$, это направление можно больше не перебирать.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ЗАДАЧИ

3.1 Реализация известных алгоритмов

В этом разделе рассматривается практическая реализация алгоритмов, связанных с поставленной задачей. Код приведен на языке Python 3. Программная реализация важна для того, чтобы проверять доказанные утверждения и выдвигать новые гипотезы.

Нужно отметить, что в программной реализации элементы разбиения нумеруются с нуля, тогда как в математическом определении элементы нумеруются с единицы. Это связано с тем, что в языках программирования стандартные массивы нумеруются с нуля, и нумерация с единицы требовала бы создания лишнего элемента.

Для начала создан класс Partition (приложение А), который хранит разбиение числа. Интерфейс этого класса состоит из методов, а внутреннее устройство класса приватное. Это позволит утверждать, что объекты класса Partition неизменяемые. Конструктор класса из последовательности целых чисел создает разбиение, делая все необходимые проверки целостности данных.

Также конструктор имеет подсчет веса и ранга разбиения. Так как вычисление веса и ранга делается проходом по всему разбиению, эффективнее посчитать их заранее и сохранить. Чтобы вычислить ранг разбиения, обратимся к формуле $r(\lambda) = \max\{i \mid \lambda_i \geq i\}$ (1.3). В программе нумерация идет с нуля, поэтому имеет место формула $r(\lambda) = \max\{i + 1 \mid \lambda_i \geq i + 1\}$. Так как $i + 1$ возрастает а λ_i невозрастает, можно развернуть знак неравенства, вместо этого вычислив ранг как $r(\lambda) = \min\{i + 1 \mid \lambda_i < i + 1\} - 1$. Эта формула упрощается до $r(\lambda) = \min\{i \mid \lambda_i \leq i\}$, что проще реализовать в коде.

Сами значения веса и ранга, а также массив элементов разбиения, являются приватными переменными. Таким образом, объекты класса Partition

неизменяемые. И нельзя поменять только сами элементы разбиения, тем самым сделав переменные _weight и _rank неактуальными. Доступ к весу и рангу реализован через декоратор *property*.

Чтобы вычислить λ^*, нужно по сути отразить диаграмму Ферре разбиения λ относительно главной диагонали. Для этого создается пустой массив, и за каждый элемент разбиения λ_i первые λ_i чисел в массиве увеличиваются на 1. Тогда *i*-столбец превратится в строку. Используя эту функцию, нетрудно реализовать вычисление головы $hd(\lambda)$ и хвоста $tl(\lambda)$ разбиений.

Самыми сложными операциями являются пересечение и объединение. Для пересечения реализован алгоритм из [9]. Для объединения использованы свойства решетки. Если у двух разбиений λ и μ вес совпадает: $\text{sum}(\lambda) = \text{sum}(\mu) = m$, то в силу свойств NPL($m$) выполняется $\lambda \vee \mu = (\lambda^* \wedge \mu^*)^*$. Если вес одного разбиения меньше другого, то в конец разбиения меньшего веса нужно приписать единичные компоненты, пока вес не сравняется.

Чтобы проверять различные утверждения о разбиениях чисел, необходимо уметь их перебирать. Для этого написан программный код для классов, отвечающие за различные множества разбиений (приложение Б). Есть три важных множества: NPL(m), GPS($2m$) и MGP($2m$) — множество всех разбиений, графических разбиений и максимальных графических разбиений соответственно.

При написании этих классов реализуем паттерн *singleton*. Это позволит нам утверждать, что два экземпляра класса с одним и тем же параметром веса m на самом являются одним и тем же объектов. В Python это можно реализовать, перегрузив метод конструктора __new__.

Рассмотрим перебор разбиений из NPL(m). Перебирать разбиения можно в лексикографическом порядке. Лексикографически наименьшее разбиение веса m — это разбиение из m единичных компонент. Чтобы найти следующее

разбиение, идущее после данного разбиения λ, нужно увеличить его в самой правой возможной позиции. Для этого берется самый правый индекс i кроме последнего, для которого $\lambda_i < \lambda_{i-1}$, или i = 1 при его отсутствии. Элемент λ_i увеличивается на 1, а все элементы правее него заполняются единицами. Заканчивается перебор лексикографически максимальным разбиением λ = (*m*).

Реализация классов GPS и MGP во многом совпадает. Интерес представляет генерация множества максимальных графических разбиений веса *m*. Если λ — максимальное графическое разбиение веса *m* и ранга *r*, то ℓ(hd(λ)) = *r* и $\mathrm{sum}(\mathrm{hd}(\lambda)) = \frac{m-r(r-1)}{2}$. Таким образом, нужно перебрать все разбиения веса $\frac{m-r(r-1)}{2}$ и длины *r*, и они будут головой и хвостом максимального графического разбиения.

3.2 Построение интервала решетки разбиений

Согласно теореме 1, для построения множества доминирующих максимальных разбиений нужно построить несколько решеток голов разбиения. Построить все разбиения длины не более *l*, элементы которых не превосходят *k*, можно следующим процессом.

В начале процесса возьмем одно разбиение (*k*, 0, …). На каждом шаге к разбиению применим одно из двух действий:

1) Уменьшим последний ненулевой элемент разбиения на 1, если он больше единицы.

2) Добавим в конец разбиения новый элемент, равный последнему нулевому, если длина разбиения меньше *l*.

Нетрудно видеть, что этот процесс может получить любое ненулевое разбиение длины не больше *l*, элементы которого не превосходят *k*. Более того, для каждого такого разбиения существует единственная последовательность действий, получающая это разбиение. На самом деле, длина *l* в этом процессе не важна, она используется только как точка

отсечения.

На этом процессе основывается алгоритм генерации всех разбиений, лежащих в данном интервале. Пусть λ и μ — соответственно нижняя и верхняя граница интервала. Если на очередном шаге пересечения имеется разбиение η длины $\ell(\eta) = i$, которое строго доминируется разбиением (λ_1, λ_2, …, λ_i), то двумя приведенными действиями уже нельзя получить разбиение, доминирующее λ. На этом процесс можно прервать.

Чтобы генерировать только разбиения лежащие под μ, нужно модифицировать второй шаг процесса. В самом деле, пусть в текущее разбиение η добавлен новый последний элемент на позицию $i + 1$, и после добавления разбиение η стало строго больше μ. Далее первым шагом можно уменьшить добавленный элемент η_{i+1}, и условие $\eta \leq \mu$ сможет снова выполится.

Исправление заключается в том, что вторым шагом добавляется элемент, меньший либо равный последнего ненулевого и не нарушающий условие $\eta \leq \mu$.

Пусть на данном шаге построено разбиение η длины $\ell(\eta)=i$. Тогда второй шаг добавит на позицию $i + 1$ элемент $\min(\eta_i, \mu_1 + \mu_2 + \ldots + \mu_i + \mu_{i+1} - \mathrm{sum}(\eta))$, если это число не равно нулю.

Итак, в процессе перебора текущее разбиение всегда будет доминироваться верхней границей μ. Также, если длина текущего разбиения $\ell(\eta)$ равна i, то должно выполняться $\lambda \leq_i \eta$.

Для эффективного перебора также важно не рассматривать разбиения, которые не могут быть продлены до разбиения из интервала от λ до μ. Текущее разбиение всегда лежит не выше верхней границы. Чтобы проверить, что текущее разбиение η можно продлить до разбиения, доминирующего λ, добавим в конец к нему элементы, равные последнему ненулевому элементу η, пока длина не станет равна $\ell(\lambda)$. Эта вспомогательная функция реализована в программном коде (приложение В).

Этой функции и приведенного выше процесса с модификацией

достаточно, чтобы сгенерировать все разбиения из интервала. Как было сказано ранее, каждое разбиение будет получено таким процессом ровно один раз, так что можно не делать проверку на уникальность. Реализовано это в функциях generate_interval и generate_dominating (приложение В). С помощью теоремы 2 можно сократить перебор рангов максимального графического доминирования, что применяется в функции generate_dominating.

На основе алгоритма перебора можно реализовать алгоритм подсчета разбиений, принадлежащих данному интервалу. Поскольку для подсчета не нужно запоминать все компоненты разбиений, достаточно хранить только последний ненулевой элемент разбиения, его позицию, а также текущий вес разбиения. Таким образом можно построить алгоритм динамического программирования, считающий количество разбиений без перебора. Это реализовано в функциях count_interval и count_dominating (приложение В).

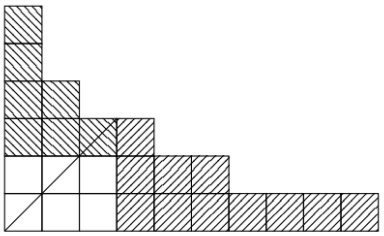
4 РЕЗУЛЬТАТЫ И ИХ ОБСУЖДЕНИЕ

Для фиксированного графического разбиения λ веса $2m$ множество всех максимальных графических разбиений μ веса $2m$ таких, что $\lambda \le \mu$, — это множество максимальных графических разбиений, головы которых лежат в интервале от $\mathrm{hd}_t(\lambda) \vee \mathrm{minp}(m - \frac{t(t-1)}{2}, t)$ до $\mathrm{tl}_t(\lambda) \wedge \mathrm{maxp}(m - \frac{t(t-1)}{2}, t)$ для всех таких $t \in \mathrm{N}$, что $\mathrm{t} \le \lfloor \frac{\sqrt{8m+1}-1}{2} \rfloor$ (см. неравенство 2.5).

В качестве примера рассмотрим разбиение λ = (6, 4, 3, 3, 2, 2, 1, 1, 1, 1) (см. рис. 4.1), где $m = 12$. Поскольку sum(λ) = 24 и hd(λ) = (4, 2, 1) ≤ tl(λ) = (7, 3, 1), разбиение λ является графическим. Максимальный возможный ранг доминирующих его максимальных графических разбиений того же веса вычисляется по формуле

$$\lfloor \frac{\sqrt{8m+1}-1}{2} \rfloor = 4, \qquad (4.1)$$

т. е., все искомые максимальные графические разбиения имеют ранг не более 4.



$\lambda = (6, 4, 3, 3, 2, 2, 1, 1, 1, 1)$

Рис. 4.1

1) $t = 1$: $\mathrm{hd}_t(\lambda) = (6)$, $\mathrm{tl}_t(\lambda) = (9)$,

$\mathrm{minp}(m - \frac{t(t-1)}{2}, t) = (12)$, $\mathrm{maxp}(m - \frac{t(t-1)}{2}, t) = (12)$,

$\eta_1 = (12)$, $\eta_2 = (9)$, $\eta_1 \not\trianglelefteq \eta_2$.

Следовательно, множество $\mathrm{hdMGP}_1(\lambda)$ пусто.

2) $t = 2$: $\mathrm{hd}_t(\lambda) = (5, 3)$, $\mathrm{tl}_t(\lambda) = (8, 4)$,

$\mathrm{minp}(m - \frac{t(t-1)}{2}, t) = (6, 5)$, $\mathrm{maxp}(m - \frac{t(t-1)}{2}, t) = (10, 1)$,

$\eta_1 = (6, 5)$, $\eta_2 = (8, 3)$, $\eta_1 \leq \eta_2$.

Следовательно, множество $hdMGP_2(\lambda)$ не пусто и содержит разбиения (6, 5), (7, 4) и (8, 3). Максимальные графические разбиения, построенные по этим головам — это (7, 6, 2, 2, 2, 2, 2, 1), (8, 5, 2, 2, 2, 2, 1, 1, 1), (9, 4, 2, 2, 2, 1, 1, 1, 1, 1).

3) $t = 3$: $\mathrm{hd}_t(\lambda) = (4, 2, 1)$, $\mathrm{tl}_t(\lambda) = (7, 3, 1)$,

$\mathrm{minp}(m - \frac{t(t-1)}{2}, t) = (3, 3, 3)$, $\mathrm{maxp}(m - \frac{t(t-1)}{2}, t) = (7, 1, 1)$,

$\eta_1 = (4, 3, 2)$, $\eta_2 = (7, 1, 1)$, $\eta_1 \leq \eta_2$.

Следовательно, множество $\mathrm{hdMGP}_3(\lambda)$ не пусто и является интервалом от разбиения (4, 3, 2) до разбиения (7, 1, 1). В интервале от η_1 до η_2 лежат разбиения (4, 3, 2), (4, 4, 1), (5, 2, 2), (5, 3, 1), (6, 2, 1) и (7, 1, 1). Максимальные графические разбиения, построенные по этим головам — это (6, 5, 4, 3, 3, 2, 1), (6, 6, 3, 3, 2, 2, 2), (7, 4, 4, 3, 3, 1, 1, 1), (7, 5, 3, 3, 2, 2, 1, 1), (8, 4, 3, 3, 2, 1, 1, 1, 1) и (9, 3, 3, 3, 1, 1, 1, 1, 1, 1).

4) $t = 4$: $\mathrm{hd}_t(\lambda) = (3, 1)$, $\mathrm{tl}_t(\lambda) = (6, 2)$,

$\mathrm{minp}(m - \frac{t(t-1)}{2}, t) = (2, 2, 1, 1)$, $\mathrm{maxp}(m - \frac{t(t-1)}{2}, t) = (3, 1, 1, 1)$,

$\eta_1 = (3, 1, 1, 1)$, $\eta_2 = (3, 1, 1, 1)$, $\eta_1 \leq \eta_2$.

Следовательно, множество $\mathrm{hdMGP}_4(\lambda)$ не пусто и содержит единственный элемент (3, 1, 1, 1). Максимальное графическое разбиение, построенное по этой голове — (6, 4, 4, 4, 4, 1, 1).

Итак, существует 10 максимальных графических разбиений, доминирующих λ. Они, конечно, не сравнимы между собой, но их головы образуют три подрешетки в NPL. Строения этих подрешеток показаны на рис. 4.2. Обратим внимание, что множество голов всех максимальных графических разбиений веса $2m$, доминирующих λ, не является ординальной суммой подрешеток $\mathrm{hdMGP}_t(\lambda)$. Кроме того, существует максимальное

графическое разбиение (5, 5, 5, 3, 3, 3) веса 24, не доминирующее λ, голова которого (3, 3, 3) лежит в интервале от (3, 1, 1, 1) до (8, 3).

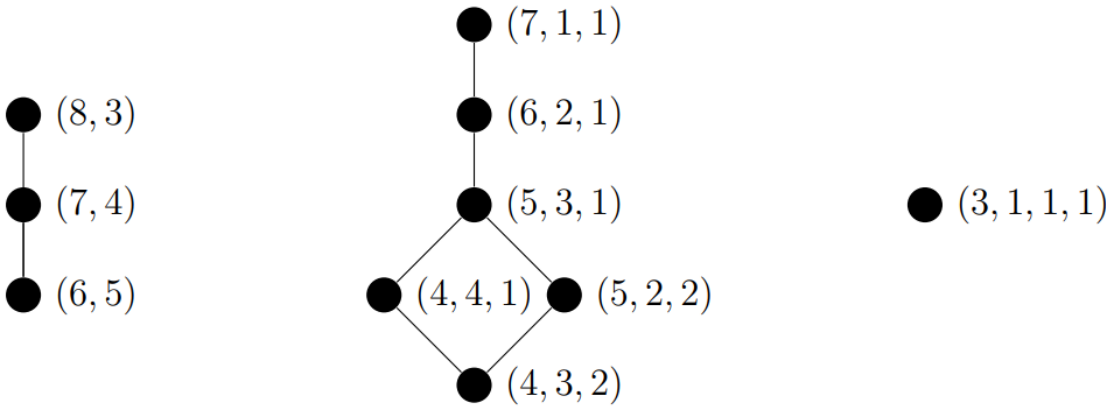


Рис 4.2

ЗАКЛЮЧЕНИЕ

По итогам выполненной работы могут быть сделаны следующие выводы.

1) Доказана теорема 1, описывающая общий вид множества максимальных графических разбиений, доминирующих заданное графическое разбиение и имеющих с ним одинаковый вес.
2) С использованием теоремы 1 построен полиномиальный алгоритм нахождения всех максимальных графических разбиений, доминирующих заданное графическое разбиение и имеющих с ним одинаковый вес. Указана также серия алгоритмов для отдельных модулей построенного алгоритма.
3) Разработаны программные реализации всех указанных алгоритмов.
4) Проведен компьютерный эксперимент по проверке Теоремы 1 с помощью переборного алгоритма на всех графических разбиениях веса не более 50.

Таким образом, все поставленные цели были достигнуты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. G.E. Andrews. The theory of partitions // Cambridge University Press. — 1976. — MR0557013.
2. T. Brylawski. The lattice of integer partitions // Discrete Mathematics. — 1973. — Vol. 6. — P. 210—219. — MR0325405.
3. Mahadev N.V.R., Peled U.N. Threshold Graphs and Related Topics // Annals of Discrete Mathematics. — 1995. — Vol. 56. — P. 542.
4. Баранский В.А., Сеньчонок Т.А. О пороговых графах и реализациях графических разбиений // Труды Института математики и механики УрО РАН. — 2017. — Т. 23, № 1. — С. 1–10.
5. Баранский В.А., Сеньчонок Т.А. О максимальных графических разбиениях // Сибирские электронные математические известия. — 2017. — Т. 14. — С. 112—124.
6. Асанов М.О., Баранский В.А., Расин В.В. Дискретная математика: графы, матроиды, алгоритмы. 2-е изд., испр. и доп. – СПб: Лань, 2010, 368 с.
7. Baransky V.A., Koroleva T.A. The lattice of partitions of a positive integer // Doklady Math., 2008, Vol. 77, № 1, P. 72-75.
8. Баранский В.А., Королева Т.А., Сеньчонок Т.А. О решетке разбиений натурального числа // Труды Института математики и механики УрО РАН. — 2015. — Т. 21, № 3. — С. 30—36.
9. Баранский В.А., Королева Т.А., Сеньчонок Т.А. О решетке разбиений всех натуральных чисел // Сибирские электронные математические известия. — 2016. — Т. 13. — С. 744—753.
10. Erdös P., Gallai T. Graphs with given degree of vertices // Math. Zapok, 11 (1960), P. 264-274.
11. Баранский В.А., Сеньчонок Т.А. О максимальных графических разбиениях, ближайших к заданному графическому разбиению // Sib. Elect. Math. Reports, 2020, Vol. 17, P. 338-363. [Russian, English abstract] DOI 10.33048/semi.2020.17.022.

ПРИЛОЖЕНИЕ А

(Справочное)

ТЕКСТОВЫЙ КОД МОДУЛЯ PARTITION, ОТВЕЧАЮЩЕГО ЗА РАЗБИЕНИЕ ЧИСЛА

```
 1 class Partition:
 2     """Partition of a non-negative integer"""
 3     @staticmethod
 4     def zip(*partitions):
 5         """Yields tuples where the i-th element comes from the
   i-th partition. Continues until the longest partition is
   exhausted."""
 6         for i in range(max(len(e) for e in partitions)):
 7             yield tuple(e[i] for e in partitions)
 8 
 9     @staticmethod
10     def maxp(weight, length):
11         """Returns maxp(weight, length), which is the largest
   partition with given weight and length"""
12         if length > weight:
13             raise ValueError('Cannot create a partition with
   more length than weight')
14         if length == 0:
15             if weight > 0:
16                 raise ValueError('Cannot create a partition
   with zero length non-zero weight')
17             return Partition([])
18         return Partition([weight - length + 1] + [1] * (length
   - 1))
19 
20     @staticmethod
21     def minp(weight, length):
22         """Returns minp(weight, length), which is the smallest
   partition with given weight and length"""
23         if length > weight:
24             raise ValueError('Cannot create a partition with
   more length than weight')
25         if length == 0:
26             if weight > 0:
27                 raise ValueError('Cannot create a partition
   with zero length non-zero weight')
28             return Partition([])
29         m = weight % length
30         d = weight // length
31         return Partition([d + 1] * m + [d] * (length - m))
32 
33     @staticmethod
```

```
34      def from_ht(hd, tl):
35          """Generates a partition by its head and tail"""
36          rank = len(hd)
37          if len(tl) > rank:
38              raise ValueError('Cannot create a partition with
    tail longer than rank')
39          return Partition([e + rank - 1 for e in hd] +
    tl.transpose()._ar)
40
41      def __init__(self, sequence):
42          self._ar = list(sequence)
43          if not all(isinstance(x, int) for x in self._ar):
44              raise TypeError('Non-integer element found')
45          if min(self._ar, default=0) < 0:
46              raise ValueError('Negative element found')
47          for i in range(len(self._ar) - 1):
48              if self._ar[i] < self._ar[i + 1]:
49                  raise ValueError('Increasing pair of elements
    found')
50          while self._ar and self._ar[-1] == 0:
51              self._ar.pop()
52          self._weight = sum(self._ar)
53          for i, e in enumerate(self._ar):
54              if e <= i:
55                  self._rank = i
56                  break
57          else:
58              self._rank = len(self._ar)
59
60      def __len__(self):
61          return len(self._ar)
62
63      def __getitem__(self, index):
64          if index < 0:
65              raise IndexError('Negative index of partition')
66          if index >= len(self._ar):
67              return 0
68          return self._ar[index]
69
70      def __iter__(self):
71          return iter(self._ar)
72
73      @property
74      def rank(self):
75          """Rank r(self) of the partition"""
76          return self._rank
77
78      @property
79      def weight(self):
80          """Weight sum(self) of the partition"""
```

```
 81          return self._weight
 82
 83      def transpose(self):
 84          """Returns the transposed partition self"""
 85          if not self._ar:
 86              return Partition([])
 87          res = [0] * self._ar[0]
 88          for e in self._ar:
 89              res[e - 1] += 1
 90          for i in range(len(res) - 1, 0, -1):
 91              res[i - 1] += res[i]
 92          return Partition(res)
 93
 94      def head(self):
 95          """Returns the head hd(self) of the partition"""
 96          return Partition(self._ar[i] - self.rank + 1 for i in
    range(self.rank))
 97
 98      def tail(self):
 99          """Returns the tail tl(self) of the partition"""
100          return Partition(self._ar[self.rank:]).transpose()
101
102      def __str__(self):
103          if not self._ar:
104              return '(0, ...)'
105          return f'({", ". join(map(str, self._ar))}, 0, ...)'
106
107      def __repr__(self):
108          return f'Partition({self._ar})'
109
110      def __eq__(self, other):
111          return self._ar == other._ar
112
113      def __ne__(self, other):
114          return self._ar != other._ar
115
116      def __le__(self, other):
117          balance = 0
118          for a, b in Partition.zip(self, other):
119              balance += a - b
120              if balance > 0:
121                  return False
122          return True
123
124      def __ge__(self, other):
125          balance = 0
126          for a, b in Partition.zip(self, other):
127              balance += a - b
128              if balance < 0:
129                  return False
```

```
130         return True
131 
132     def __hash__(self):
133         return hash(tuple(self._ar))
134 
135     def intersection(self, other):
136         """Returns the intersection self ^ other of two
    partitions"""
137         bal1 = 0
138         bal2 = 0
139         res = []
140         for a, b in Partition.zip(self, other):
141             a += bal1
142             b += bal2
143             if a <= b:
144                 res.append(a)
145                 bal1 = 0
146                 bal2 = b - a
147             else:
148                 res.append(b)
149                 bal1 = a - b
150                 bal2 = 0
151         return Partition(res)
152 
153     def elongate(self, add):
154         """Appends components of value 1 to the end of the
    partition, in the amount of add"""
155         return Partition(self._ar + [1] * add)
156 
157     def union(self, other):
158         """Returns the union self v other of two partitions"""
159         if self.weight < other.weight:
160             self = self.elongate(other.weight - self.weight)
161         elif self.weight > other.weight:
162             other = other.elongate(self.weight - other.weight)
163         self = self.transpose()
164         other = other.transpose()
165         return self.intersection(other).transpose()
166 
167     def head_ranked(self, rank):
168         """Returns partition hd_rank(self)"""
169         return Partition(max(self._ar[i] - rank + 1, 0) for i
    in range(min(len(self._ar), rank)))
170 
171     def tail_ranked(self, rank):
172         """Returns partition tl_rank(self)"""
173         res = [min(e, rank) for e in self._ar[rank:]]
174         return Partition(res).transpose()
175 
176     def is_graphical(self):
```

```
177         """Returns True if the partition is graphical"""
178         return self.weight % 2 == 0 and self.head() <=
    self.tail()
179
180     def is_maximal(self):
181         """Returns True if the partition is maximal
    graphical"""
182         return self.head() == self.tail()
183
```

ПРИЛОЖЕНИЕ Б

(Справочное)

ТЕКСТОВЫЙ КОД МОДУЛЯ SETS, ОТВЕЧАЮЩЕГО ЗА ПЕРЕБОР МНОЖЕСТВ РАЗБИЕНИЙ

```
from partition import Partition


class NPL:
    """Set of all partitions of given non-negative integer"""
    _NPL = {}

    def __init__(self, m):
        if not isinstance(m, int):
            raise TypeError('Non-integer m given')
        if m < 0:
            raise ValueError('Negative integer m given')
        self._m = m

    def __new__(cls, m):
        if not isinstance(m, int):
            raise TypeError('Non-integer m given')
        if m not in NPL._NPL:
            NPL._NPL[m] = super(NPL, cls).__new__(cls)
        return NPL._NPL[m]

    def __contains__(self, x):
        if not isinstance(x, Partition):
            return NotImplemented
        return x.weight == m

    def __iter__(self):
        ar = [1] * self._m
        yield Partition(ar)
        while len(ar) > 1:
            s = ar[-1]
            for i in range(len(ar) - 2, -1, -1):
                if i == 0 or ar[i] < ar[i - 1]:
                    ar = ar[:i] + [ar[i] + 1] + [1] * (s - 1)
                    break
                s += ar[i]
            yield Partition(ar)


class GPS:
    """ Set of all graphical partitions of given non-negative
even integer"""
```

```
    _GPS = {}

    def __init__(self, m):
        if not isinstance(m, int):
            raise TypeError('Non-integer m given')
        if m < 0:
            raise ValueError('Negative integer m given')
        if m % 2:
            raise ValueError('Odd integer m given')
        self._m = m

    def __new__(cls, m):
        if not isinstance(m, int):
            raise TypeError('Non-integer m given')
        if m not in GPS._GPS:
            GPS._GPS[m] = super(GPS, cls).__new__(cls)
        return GPS._GPS[m]

    def __contains__(self, x):
        if not isinstance(x, Partition):
            return NotImplemented
        return x.weight == self._m and x.is_graphical()

    def __iter__(self):
        return filter(lambda x: x in self, NPL(self._m))


class MGP:
    """Set of all maximal graphical partitions of given
non-negative even integer"""
    _MGP = {}

    def __init__(self, m):
        if not isinstance(m, int):
            raise TypeError('Non-integer m given')
        if m < 0:
            raise ValueError('Negative integer m given')
        if m % 2:
            raise ValueError('Odd integer m given')
        self._m = m

    def __new__(cls, m):
        if not isinstance(m, int):
            raise TypeError('Non-integer m given')
        if m not in MGP._MGP:
            MGP._MGP[m] = super(MGP, cls).__new__(cls)
        return MGP._MGP[m]

    def __contains__(self, x):
        if not isinstance(x, Partition):
```

```
            return NotImplemented
        return x.weight == self._m and x.is_maximal()

    def __iter__(self):
        rank = 0
        while rank * (rank + 1) <= self._m:
            if rank > 0 or self._m == 0:
                for e in NPL((self._m - rank * (rank - 1)) //
2):
                    if len(e) == rank:
                        yield Partition.from_ht(e, e)
            rank += 1

```

ПРИЛОЖЕНИЕ В

(Справочное)

ТЕКСТОВЫЙ КОД МОДУЛЯ THEOREM, ОТВЕЧАЮЩЕГО ЗА РЕАЛИЗАЦИЮ УТВЕРЖДЕНИЙ ТЕОРЕМЫ

```
 1 from partition import Partition
 2
 3
 4 def get_bounds(partition, rank):
 5     """Returns min and max heads of dominating MGP of given
   rank, according to the main theorem"""
 6     m = partition.weight // 2
 7     lower = partition.head_ranked(rank).union(Partition.minp(m
   - rank * (rank - 1) // 2, rank))
 8     upper = partition.tail_ranked(rank).intersection(
   Partition.maxp(m - rank * (rank - 1) // 2, rank))
 9     return lower, upper
10
11
12 def can_be_extended(lower, upper, cur):
13     """Returns True if given prefix of partition can be
   extended to lie between lower and upper"""
14     if not(lower <= upper):
15         return False
16     if len(cur) == 0:
17         return True
18     if not (Partition(cur) <= upper):
19         return False
20     balance = 0
21     for i in range(len(lower)):
22         balance += (cur[i] if i < len(cur) else cur[-1]) -
   lower[i]
23         if balance < 0:
24             return False
25     return True
26
27
28 def generate_interval(lower, upper, cur=None):
29     """Yields all partitions within a given interval"""
30     if cur is None:
31         cur = [upper[0]]
32     if not can_be_extended(lower, upper, cur):
33         return
34     if cur[-1] == 0:
35         yield Partition(cur)
36         return
37
```

```
    next_value = min(cur[-1], sum(upper[i] for i in
range(len(cur) + 1)) - sum(cur))
    cur.append(next_value)
    yield from generate_interval(lower, upper, cur)
    cur.pop()

    while cur[-1] > 0:
        cur[-1] -= 1
        yield from generate_interval(lower, upper, cur)


def count_interval(lower_partition, upper_partition):
    """Counts all partitions within a given interval"""
    if lower_partition == upper_partition:
        return 1
    if not (lower_partition <= upper_partition):
        return 0

    lower = []
    upper = []
    sum1 = 0
    sum2 = 0
    for a, b in Partition.zip(lower_partition,
upper_partition):
        sum1 += a
        sum2 += b
        lower.append(sum1)
        upper.append(sum2)

    lower += [lower[-1]] * (upper[-1] - lower[-1] + 1)
    upper += [upper[-1]] * (upper[-1] - lower[-1] + 1)
    max_length = len(upper)
    max_value = upper_partition[0]
    max_sum = max(upper)
    dp = [[[0] * (max_sum + 1) for j in range(max_value + 1)]
for i in range(max_length)]
    dp[0][max_value][max_value] = 1

    for pos in range(max_length - 1):
        for value in range(max_value, -1, -1):
            for pref_sum in range(lower[pos], upper[pos] + 1):
                if value > 0 and pref_sum - 1 >= lower[pos]:
                    dp[pos][value - 1][pref_sum - 1] +=
dp[pos][value][pref_sum]
                next_value = min(value, upper[pos + 1] -
pref_sum)
                if pref_sum + next_value >= lower[pos + 1]:
                    dp[pos + 1][next_value][pref_sum +
next_value] += dp[pos][value][pref_sum]

```

```
 82      return sum(dp[-1][i][j] for i in range(max_value + 1) for j
    in range(max_sum + 1))
 83
 84
 85  def generate_dominating(partition):
 86      """Yields all MGP of the same weight, dominating the given
    partition"""
 87      if not partition.is_graphical():
 88          return
 89      if partition.is_maximal():
 90          yield partition
 91          return
 92      m = partition.weight // 2
 93
 94      rank = partition.rank
 95      while rank * (rank + 1) <= m * 2:
 96          lower, upper = get_bounds(partition, rank)
 97          found = False
 98          for e in generate_interval(lower, upper):
 99              yield Partition.from_ht(e, e)
100              found = True
101          if not found:
102              break
103          rank += 1
104
105      rank = partition.rank - 1
106      while rank > 0:
107          lower, upper = get_bounds(partition, rank)
108          found = False
109          for e in generate_interval(lower, upper):
110              yield Partition.from_ht(e, e)
111              found = True
112          if not found:
113              break
114          rank -= 1
115
116
117  def count_dominating(partition):
118      """Counts all MGP of the same weight, dominating the given
    partition"""
119      if not partition.is_graphical():
120          return 0
121      if partition.is_maximal():
122          return 1
123      m = partition.weight // 2
124      rank = 1
125      res = 0
126
127      rank = partition.rank
128      while rank * (rank + 1) <= m * 2:
```

```
129         lower, upper = get_bounds(partition, rank)
130         add = count_interval(lower, upper)
131         if add == 0:
132             break
133         res += add
134         rank += 1
135 
136     rank = partition.rank - 1
137     while rank > 0:
138         lower, upper = get_bounds(partition, rank)
139         add = count_interval(lower, upper)
140         if add == 0:
141             break
142         res += add
143         rank -= 1
144     return res
145 
```

ПРИЛОЖЕНИЕ Г

(Справочное)

ТЕКСТОВЫЙ КОД МОДУЛЯ MAIN, ОТВЕЧАЮЩЕГО ЗА ПРОВЕРКУ РЕАЛИЗОВАННЫХ АЛГОРИТМОВ

```
 1 import sets
 2 import collections
 3 import theorem
 4 from partition import Partition
 5
 6
 7 def NPL_by_length(m, length):
 8     """Yields all partitions of given weight and length"""
 9     if m < 0:
10         raise ValueError('Negative integer m given')
11     if length < 0:
12         raise ValueError('Negative integer l given')
13     if length > m:
14         raise ValueError('Given l is greater than m')
15     if m and not length:
16         raise ValueError('Zero l given while m is non-zero')
17     yield from theorem.generate_interval(Partition.minp(m,
   length), Partition.maxp(m, length))
18
19
20 def MGP_by_rank(m, rank):
21     """Yields all maximal graphical partitions of given weight
   and rank"""
22     if m < 0:
23         raise ValueError('Negative integer m given')
24     if m % 2:
25         raise ValueError('Odd integer m given')
26     if rank < 0:
27         raise ValueError('Negative integer r given')
28     if rank > m:
29         raise ValueError('Given r is greater than m')
30     if m and not rank:
31         raise ValueError('Zero r given while m is non-zero')
32     if rank * (rank + 1) > m:
33         raise ValueError('Rank is too big for given m')
34     for e in NPL_by_length((m - rank * (rank - 1)) // 2, rank):
35         yield Partition.from_ht(e, e)
36
37
38 def test_theorem(partition):
39     """Generates all sets hdMGP according to the main
   theorem"""
40     rank = 1 if partition.weight > 0 else 0
```

```
    res = []
    while rank * (rank + 1) <= partition.weight:
        for e in MGP_by_rank(partition.weight, rank):
            if partition <= e:
                res.append(e)
        rank += 1
    return res


def main():
    # Testing NPL_by_length and MGP_by_rank
    partitions = []
    for i in range(17):
        partitions.extend(sets.NPL(i))

    partition_set = collections.defaultdict(lambda: [])
    for e in partitions:
        partition_set[(e.weight, len(e))].append(e)
    for (weight, length), expected in partition_set.items():
        result = list(NPL_by_length(weight, length))
        if len(expected) != len(result) or set(expected) !=
set(result):
            print(f'NPL_by_length fails with weight {weight}
length {length}')
            return

    maximal_partition_set = collections.defaultdict(lambda: [])
    for e in partitions:
        if e.is_maximal():
            maximal_partition_set[(e.weight, e.rank)].append(e)
    for (weight, rank), expected in
maximal_partition_set.items():
        result = list(MGP_by_rank(weight, rank))
        if len(expected) != len(result) or set(expected) !=
set(result):
            print(f'MGP_by_rank fails with weight {weight} rank
{rank}')
            return

    # Testing the theorem
    for i in range(0, 51, 2):
        for partition in sets.GPS(i):
            expected = test_theorem(partition)

            # Explicitly testing the statement
            rank = 1 if partition.weight > 0 else 0
            bounds = []
            while rank * (rank + 1) <= partition.weight:
                bounds.append(theorem.get_bounds(partition,
rank))
```

```
                rank += 1
            for e in expected:
                if not any(a <= e.head() <= b for a, b in
bounds):
                    print(f'Partition {partition} fails the
test, being dominated by {e}')
                    return

            # Testing generate_dominating
            result = list(theorem.generate_dominating(
partition))
            if len(expected) != len(result) or set(expected) !=
set(result):
                print(f'Partition {partition} fails the test,
dominating sets not equal')
                return

            # Testing count_dominating
            cnt = theorem.count_dominating(partition)
            if len(expected) != cnt:
                print(f'Partition {partition} fails, expected
{len(expected)} dominating, found {cnt}')
                return


if __name__ == '__main__':
    main()

```