УДК 681.3

А.Ю. Левченко, інж.
Житомирський державний технологічний університет

АЛГОРИТМ, ЯКИЙ ГЕНЕРУЄ ВСІ ПІРАМІДАЛЬНІ ПЕРЕСТАНОВКИ

(Представлено д.т.н., проф. Панішевим А.В.)

Запропоновано алгоритм, який генерує всі пірамідальні перестановки за 2^{N-2} *ітерацій, де* N *– довжина перестановки.*

Вступ. Пірамідальними називаються перестановки виду:

$$\tau = \{\tau_1, \tau_2, ..., \tau_{m-1}, \tau_m, \tau_{m+1}, ..., \tau_N\}, \quad (1)$$

де $\tau_1 < \tau_2 < ... < \tau_m > \tau_{m+1} > ... > \tau_N$.

Їх загальна кількість дорівнює 2^{N-1} [1], [2], [5]. Запропонований алгоритм розв'язує цю задачу за 2^{N-2} ітерацій, причому є можливість розпочати послідовність з довільної комбінації. Також алгоритм легко піддається розпаралелюванню.

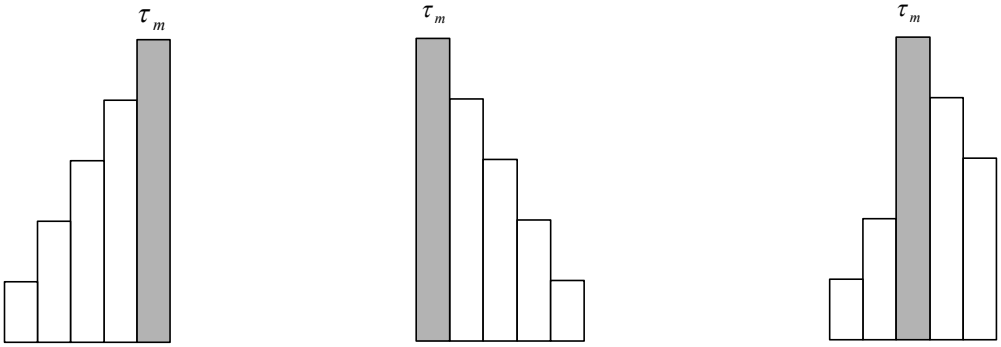


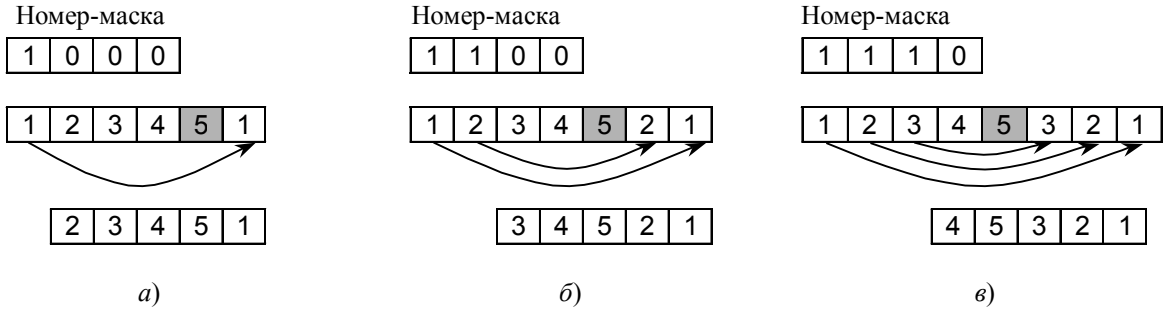
Рис. 1. Графічна інтерпретація пірамідальних перестановок

Максимальний елемент пірамідальної перестановки τ_m, який назвемо *вершиною*, розбиває перестановку на два *схили*. Відносно вершини решта елементів розташовуються за зростанням (рис. 1, *а*), спаданням (рис. 1, *б*) або комбінуючи обидва способи (рис. 1, *в*). Якщо переносити елементи зі схилу на схил усіма можливими способами так, щоб вони задовольняли (1), то отримаємо всі можливі пірамідальні перестановки (рис. 2, *б*, *г*).

0	1 2 3 4 5	0000	1	2	3	4	5			
1	2 3 4 5 1	1000		2	3	4	5	1		
2	1 3 4 5 2	0100		1	3	4	5	2		
3	3 4 5 2 1	1100			3	4	5	2	1	
4	1 2 4 5 3	0010		1	2	4	5	3		
5	2 4 5 3 1	1010			2	4	5	3	1	
6	1 4 5 3 2	0110			1	4	5	3	2	
7	4 5 3 2 1	1110				4	5	3	2	1
8	1 2 3 5 4	0001		1	2	3	5	4		
9	2 3 5 4 1	1001			2	3	5	4	1	
10	1 3 5 4 2	0101			1	3	5	4	2	
11	3 5 4 2 1	1101				3	5	4	2	1
12	1 2 5 4 3	0011			1	2	5	4	3	
13	2 5 4 3 1	1011				2	5	4	3	1
14	1 5 4 3 2	0111				1	5	4	3	2
15	5 4 3 2 1	1111					5	4	3	2	1
а)	*б*)	*в*)					*г*)			

Рис. 2. Пірамідальні перестановки, які отримані комбінацією елементів відносно вершини для $N = 5$

Помітимо, що перебір починається з перестановки номер 0: $\tau_0 = \{1, 2, 3, 4, 5\}$, яка є пірамідальною. Назвемо її *еталонною комбінацією*. Представивши номер, який відповідає кожній перестановці (рис. 2, *а*) у двійковому вигляді, причому зліва направо (рис. 2, *в*), отримаємо 2^{N-1} комбінацій $N-1$ нулів та одиниць, які не повторюються, з якими можна однозначно зіставити всі пірамідальні перестановки таким чином. Якщо в i-м розряді двійкового номера знаходиться одиниця, то елемент τ_i переноситься на правий схил відносно вершини τ_m, яка позначена темним кольором (рис. 3, *а*, *б*, *в*).



а) *б*) *в*)

Рис. 3. Переміщення елементів τ_0 для отримання комбінацій τ_1 а), τ_3 б), τ_7 в)

Кількість елементів, які переміщуються на правий схил, дорівнює кількості одиниць у двійковому номері-масці. Кожна наступна комбінація отримується проекцією відповідної маски на перестановку τ_0, причому генерацію можна продовжити з довільного номера, однозначно відновивши послідовність. За еталонну комбінацію також можливо використовувати $\tau = \{N, N-1, ..., 2, 1\}$, але тоді елементи перестановки слід переносити з правого схилу на лівий.

Програмна реалізація даного алгоритму вимагає особливої уваги до організації структури даних. Зберігання у вигляді масиву на перший погляд суттєво програє зв'язним спискам через велику кількість операцій видалення-вставки елементів з перестановки. Але побудова кожної комбінації починається з еталонної, тобто слід або щоразу відновлювати її з тільки що отриманої перестановки, або зберігати в окремій області пам'яті та працювати з копією. З огляду на це, звернення до елементів масиву-еталона в довільному порядку без зміни їх значень може виявитися набагато простіше та швидше, ніж переміщення елементів у зв'язному списку [3], [4].

Візьмемо масив розмірності $2N-1$, який заповнений значеннями 1, 2, …, $N-1$, N, $N-1$, …, 2, 1 (рис. 4). У випадку, якщо i-й розряд номера-маски дорівнює 1, то слід звертатися не до τ_i, а до τ_{2N-1-i}, що еквівалентно переносу елементу по праву сторону від вершини τ_N.

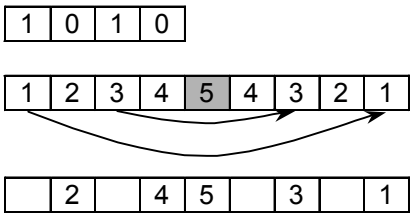


Рис. 4. Вивід еталонного масиву розміром $2N-1$ з пропуском елементів, які «перенесені»

Вивід результату виконується за час $O(N)$, а на зберігання еталонної комбінації використовується N елементів пам'яті, замість передбачуваних $O(2N-1)$ та $2N-1$ відповідно, що буде показано нижче. Очевидно, дана задача є підзадачею більш складних задач, тому перестановку-результат слід накопичувати в масиві для подальших обчислень. Через високу швидкість операцій звернення до елементів масиву в масиві-результаті можна формувати одночасно обидва схили з різних сторін. Лічильником, який вказує на перше вільне місце в масиві-результаті зліва, є змінна n, а праворуч – змінна m. З масиву-еталона можна використовувати тільки його половину через його симетрію (рис. 5).

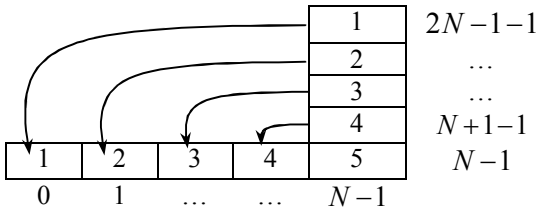


Рис. 5. Перетворення масиву розмірності $2N-1$ *в* N

Позначимо масив-еталонну комбінацію $P[0..N-1]$, а масив, який містить результат роботи алгоритму – $C[0..N-1]$. Оскільки нумерація елементів масивів ведеться від 0 до $N-1$, з усіх індексів елементів за вертикаллю віднята додаткова одиниця, і $m=1$. Якщо i-й елемент маски дорівнює одиниці, то $P[i]$ пишеться в $C[N-m]$, та m збільшується на 1, в протилежному випадку пишемо в $C[n]$ та на 1 збільшуємо n.

Розглянемо цей процес детальніше при слові-масці 1010 (рис. 6). З рисунка видно, як елементи в залежності від значення i-го біту маски переносяться на інший схил, і збільшуються значення лічильників вільного місця. На ітерації $N-1$, $N=5$, побудову перестановки завершено за $O(N)$.

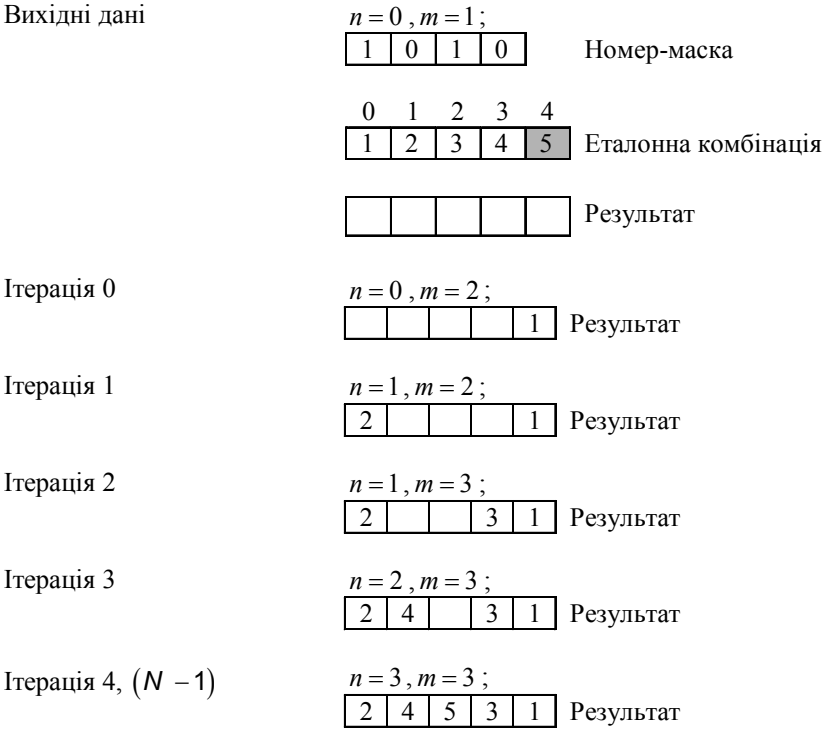


Рис.6. Ілюстрація роботи алгоритму

Для початку роботи алгоритму з довільної перестановки достатньо вказати потрібний номер-маску, який, до речі, дуже легко визначається за перестановкою: якщо елемент τ_i знаходиться праворуч від τ_m, то в розряді τ_i масиву-маски повинна міститись одиниця.

Аналіз рис. 2, *б* показує, що комбінації розділені лінією симетрії, яка проходить для $N=5$ між перестановками 7 і 8. Якщо прочитати перестановки вище за лінії симетрії справа наліво, утворюються всі перестановки, які знаходяться нижче за неї. Перестановці 0 відповідає перестановка 15, перестановці 1 – 14 і т.д. Отже, для отримання всіх 2^{N-1} перестановок достатньо згенерувати половину і прочитати їх зліва направо (*прямий хід*) та справа наліво (*зворотний хід*), що призведе до зниження кількості ітерацій до 2^{N-2}.

Реалізація алгоритму з високою швидкодією і деякими обмеженнями. Розглянемо реалізацію алгоритму, швидкодію якої було максимізовано.

S0. Алгоритм, який генерує всі 2^{N-1} пірамідальні перестановки. N – розмір перестановки. $P[0..N-1]$ – еталонна комбінація, де $P[0]=1$, $P[1]=2$, …, $P[N-1]=N$, $e=2^{N-2}$ – кількість перестановок, які слід згенерувати, number = 0 – номер початкової комбінації;

S1. while number<>e begin

S2. $n:=0$, $m:=1$; w:=1;

S3. for i :=0 to $N-1$ do
begin
if w and number then
begin $C[N-m]=P[i]$; $m:=m+1$ end
else begin $C[n]=P[i]$; $n:=n+1$ end;
Виконаємо порозрядний зсув вліво для w;
end;

S4. Вивід $C[0..N-1]$; { прямий хід}
Вивід $C[N-1..0]$; {зворотний хід}

S5. number:=number+1;
end;

Висока швидкодія цієї реалізації пояснюється використовуванням інструкцій порозрядного зсуву, які виконуються за один такт машинного часу. Двійкове слово-маска зберігається в змінній цілого типу number. Для генерації послідовності з довільної перестановки достатньо присвоїти змінній number номер відповідної комбінації. Змінна *e* містить кількість комбінацій, які слід згенерувати. Завдяки операції виведення у напрямку зворотного ходу, кількість ітерацій можливо знизити до 2^{N-2}. Змінна *w* служить для виділення i-го біта з номера комбінації number. Отримання наступного номера комбінації здійснюється за одну операцію додавання.

Недоліком даної реалізації є обмеження на розмірність задачі $N \le N_{CPU}$, де N_{CPU} – розрядність процесора, на якому виконується програма, оскільки номер-маску представляє ціле число.

Розрахуємо кількість елементарних операцій, які потрібні для генерації однієї перестановки. В найгіршому випадку для кожного елемента виконується одне порівняння, одне віднімання, одне додавання, одне присвоювання і один порозрядний зсув вліво, які будемо вважати елементарними. Отже, для генерації перестановки розмірністю N буде потрібно $5N$ елементарних операцій. Зважаючи на те, що зовнішній цикл робить 2^{N-2} ітерацій, точна продуктивність алгоритму оцінюється як $5N \cdot 2^{N-2}$ елементарних операцій.

Реалізація програми, яка не має обмежень на розмірність. Недоліки попередньої реалізації призвели до розробки програми, яка замість цілого числа в якості номера-маски використовує масив нулів та одиниць, який імітує N –розрядне двійкове число, що знімає обмеження на розмірність задач, які розв'язуються.

S0. Алгоритм, який генерує всі 2^{N-1} пірамідальні перестановки. N – розмір перестановки, $P[0..N-1]$ – еталонна комбінація, де $P[0]=1$, $P[1]=2$, …, $P[N-1]=N$, $PR[0..N-1]$ – двійковий номер-маска, при генерації всіх комбінацій $PR[0]=0$, $PR[1]=0$, …, $PR[N-1]=0$, $k:=1$;

S1. while $PR[N-2]<>1$ begin

S2. $n:=0$; $m:=1$;

S3. for $i:=0$ to $N-1$ do
begin
if $PR[i]=1$
then begin $C[N-m]=P[i]$ {$C[0..N-1]$ - масив-результат}; $m:=m+1$ end
else begin $C[n]=P[i]$; $n:=n+1$ end;
end;

S4. Вивід $C[0..N-1]$; {прямий хід}
Вивід $C[N-1..0]$; {зворотний хід}

S5. for $I:=0$ to $N-1$ do
begin

if $PR[I]=0$ then begin $PR[I]:=1$, $k:=1$ end
else begin $PR[I]:=0$, $k:=k+1$, if $I>N-2$ then перейти до S1 end;
end;
end;

Продуктивність даної реалізації алгоритму дещо гірша, ніж у розглянутої раніше, що пояснюється додатковими обчислювальними витратами на визначення наступного слова-маски в масиві PR (S5): у гіршому випадку $N-2$ порівнювань та $N-2$ присвоювань. При цьому загальна продуктивність програми оцінюється як $\left(5N+2(N-2)\right)\cdot 2^{N-2}=(7N-4)\cdot 2^{N-2}$ елементарних операцій. Ця програма була використана для експерименту по визначенню часових характеристик алгоритму на машині з конфігурацією, вказаною в табл. 1.

Таблиця 1

Конфігурація ЕОМ для експерименту

ЦП	Celeron 1.8 GHz
Пам'ять	128 Mb
ОС	Windows XP
HDD	40 Gb

Таблиця 2

Результати обчислювального експерименту

N	Час початку обчислень	Час закінчення обчислень	Час обчислень	Розмір результату обчислень: Кількість перестановок (Об'єм на диску)
5	12:22:24	12:22:24	Практично миттєво	16 (1,25 Кбайт)
10	12:22:24	12:22:24	Практично миттєво	512 (80 Кбайт)
15	12:22:24	12:22:27	3 сек.	16384 (3,75 Мбайт)
20	12:22:27	12:24:41	2 хв.	524288 (160 Мбайт)
25	12:24:41	13:51:34	1 год.26 хв.	16777216 (6,25 Гбайт)
28	13:51:34	15:54	2 год.03 хв.	134217728 (56 ГБайт)
30			3 год.46 хв.	536870912 (240 Гбайт)
35			1 дн.10 год.	17179869184 (8,75 Тбайт)
40			12 дн.20 год.	549755813888 (320 Тбайт)
45			3 міс.25 дн.	17592186044416 (11520 Тбайт)
50			2 р.10 міс.	562949953421312 (409600 Тбайт)
55			25 р.	18014398509482000 (14417920 Тбайт)
60			**234 р.**	576460752303423000 (503316480 Тбайт)

– екстрапольовані дані

При $N=30$ експеримент був перерваний та його результати екстрапольовані. З табл. 2 видно, що часові характеристики роботи алгоритму задовільні для генерації перестановок довжиною до $N=28$. У випадку необхідності збереження повного набору перестановок на диску розраховано необхідний простір з розрахунку 2 байта на кожен елемент перестановки.

Паралельне виконання алгоритму на декількох машинах (процесорах). Алгоритм легко піддається розпаралелюванню, оскільки його виконання можна розпочати та закінчити в довільному місці, вказавши маску-слово та скільки перестановок потрібно згенерувати. Структурна схема ПО для паралельної генерації всіх пірамідальних перестановок на багатопроцесорних системах (обчислювальних кластерах) наведена нижче (рис. 7). Програма-арбітр розбиває половину основної задачі (бо друга половина є дзеркальним відображенням) на M підзадач розмірністю 25–30 і передає початкові слова-маски та кількість комбінацій, які слід згенерувати машинам (процесорам), зв'язаних в обчислювальну мережу. Спільні дані, необхідні для розрахунків, або копіюються по мережі, або знаходяться у загальнодоступній області пам'яті. З табл. 2 видно, що згенеровані перестановки уже при $N=20$ не має сенсу передавати по мережі іншим машинам або зберігати на диску. Необхідні розрахунки можна вести одразу, посилаючи

арбітру лише результат по конкретному піддіапазону пірамідальних перестановок. Завдяки такому підходу час генерації та обробки всіх перестановок зменшується приблизно в M разів, де M – кількість машин (процесорів).

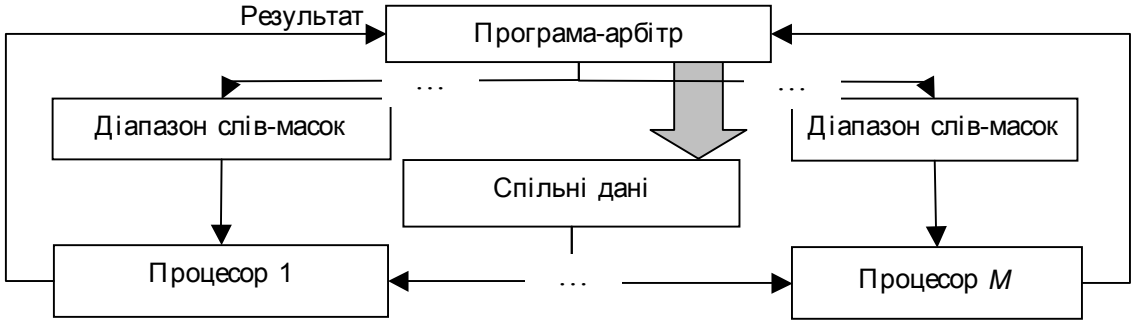


Рис. 7. Структурна схема ПО для паралельної генерації пірамідальних перестановок на декількох процесорах (машинах)

При розрахунках продуктивності даного програмного комплексу слід врахувати зростання кількості елементів у перестановках від N . Домовимось навантажувати кожну машину підзадачами розмірністю не більш, ніж $N = 30$, тоді окрема машина виконає її за час, пропорційний

$$t = 5N_M 2^{28}, \tag{2}$$

де N_M – розмірність головної задачі.

Кількість машин M для розв'язання задачі розмірністю N_M за прийнятний час:

$$M = N_M / 30. \tag{3}$$

Підзадача обсягом N для задачі розмірністю N_M буде виконуватись в k раз довше, ніж задача розмірністю N , де $N_M > N$;

$$k = \frac{N_M}{N}. \tag{4}$$

Виходячи з (2), (3), (4), при розв'язанні задачі розмірністю $N = 300$ знадобиться 10 машин, кожна з яких буде працювати в 10 раз довше, ніж при виконанні на одній машині задачі розмірністю $N = 30$, що становить приблизно 30 годин.

Висновки.

1. Робота із задачами, розмірність яких перевищує розрядність процесора, вимагає універсальну версію програми, продуктивність якої в гіршому випадку становить $(7N - 4) \cdot 2^{N-2}$. Цю версію програми можна назвати апаратно-незалежною. При $N \le N_{CPU}$, де N_{CPU} – розрядність процесора, продуктивність в гіршому випадку можна покращити до $5N \cdot 2^{N-2}$, використовуючи спеціалізовану версію програми.
2. За одну ітерацію алгоритм генерує дві пірамідальні перестановки (прямий та зворотний ходи), що скорочує перебір з 2^{N-1} до 2^{N-2} ітерацій.
3. Роботу алгоритму можна починати з довільної перестановки за номером комбінації, а також переривати після отримання необхідної кількості перестановок. Номер комбінації у двійковій формі однозначно визначається за елементами перестановки, які стоять праворуч від максимального елемента.
4. Експериментально доведено, що часові характеристики алгоритму задовільні, якщо розмірність задачі не перевищує 30.
5. Алгоритм легко розпаралелюється шляхом розбиття 2^{N-2} номерів перестановок на піддіапазони та їх паралельної обробки на декількох процесорах (машинах).
6. З ростом розмірності головної задачі лінійно росте час виконання підзадач, що залежить від довжини перестановки. Коефіцієнт, від якого залежить швидкість, обчислюється за форму лою (4). Для його зниження є сенс скорочувати піддіапазони, на які розбивається основна задача.

ЛІТЕРАТУРА:

1. *Панішев А.В., Данильченко О.М., Скачков В.О.* Вступ до теорії складності дискретних задач: Монографія. – Житомир: ЖДТУ, 2004. – 236 с
2. *Липский В*. Комбинаторика для программистов. – М.: Мир, 1988. – 203 с.

3. *Panishev A.V., Plechystyy D.D.* An effective exact algorithm for one particular case of the traveling-salesman problem // International Journal “Information Theories & Applications”. – 2003. – Vol.10. – P.355–359.
4. *Axsaeter S.* On scheduling in a semi-ordered flow shop without intermediate queues // AIIE Trans. – 1982. – Vol.14, №2. – P. 128–130.
5. *Танаев В.С., Сотсков Ю.Н., Струсевич В.А.* Теория расписаний. Многостадийные системы. – М.: Наука, 1989. – 328 с.

ЛЕВЧЕНКО Антон Юрійович – інженер Житомирського державного технологічного університету

Наукові інтереси:
- комп’ютерно-інформаційні технології;
- комбінаторна оптимізація.

Подано 23.01.06

Левченко А.Ю. Алгоритм, який генерує всі пірамідальні перестановки
Левченко А.Ю. Алгоритм, генерирующий все пирамидальные перестановки
Levchenko A.U. Algorithm for generating all pyramidal transpositions

УДК 681.3

Алгоритм, генерирующий все пирамидальные перестановки / А.Ю. Левченко

Предложен алгоритм, генерирующий все пирамидальные перестановки за 2^{N-2} итераций, где N - длина перестановки

УДК 681.3

Algorithm for generating all pyramidal transpositions / A.U. Levchenko

Algorithm for generating all pyramidal transpositions by 2^{N-2} iterations, where N – length of transposition, is presented