Acta Univ. Sapientiae, Informatica, **4,** 2 (2012) 225–236

Scattered subwords and composition of natural numbers

Zoltán KÁSA
Sapientia Hungarian Univerity of Transylvania
Department of Mathematics and Informatics, Târgu Mureş
email: kasa@ms.sapientia.ro

Zoltán KÁTAI
Sapientia Hungarian Univerity of Transylvania
Department of Mathematics and Informatics, Târgu Mureş
email: katai_zoltan@ms.sapientia.ro

Abstract. Special scattered subwords in which the length of the gaps are bounded by two natural numbers are considered. For rainbow words the number of such scattered subwords is equal to the number of special restricted compositions of natural numbers in which the components are natural numbers from a given interval. Linear algorithms to compute such numbers are given. We also introduce the concepts of generalized scattered subword (duplex-subword) and generalized composition.

1 Introduction

We define a special scattered subword [5] as a generalization of the d-subword [2] and supper-d-subword [4].

Definition 1 *Let n, $d_1 \leq d_2$ and s be positive natural numbers, and let $u = x_1x_2\dots x_n \in \Sigma^n$ be a word over an alphabet Σ. A word $v = x_{i_1}x_{i_2}\dots x_{i_s}$, where*

$i_1 \geq 1$,
$d_1 \leq i_{j+1} - i_j \leq d_2$, *for* $j = 1, 2, \dots, s-1$,
$i_s \leq n$,

*is a (d_1, d_2)-**subword** of length s of u.*

Computing Classification System 1998: G2.1, F2.2, F2.1
Mathematics Subject Classification 2010: 68R15, 05A17, 05A05
Key words and phrases: scattered subwords, composition of integers, complexity of words

For example, in the word *aabcade* the subwords abd, ace, ad are $(2,4)$-subwords.

Definition 2 *The number of different* (d_1, d_2)*-subwords of a word* w *is the* (d_1, d_2)***-complexity*** *of* w.

The $(1,d)$-complexity was studied in [2] and [3], the (d,n)-complexity in [4], while the (d_1, d_2)-complexity in [5].

2 Computing the (d_1, d_2)-complexity by digraphs

The graph method was defined for the general case of scattered subwords in [5], and for this particular case can be used as follows to compute the (d_1, d_2)-complexity of a rainbow word.

Let $G = (V, E)$ be a digraph attached to the rainbow word $a_1 a_2 \dots a_n$ and positive integers $d_1 \le d_2$, where

$V = \{a_1, a_2, \dots, a_n\}$,

$E = \{(a_i, a_j) \mid d_1 \le j - i \le d_2,\ i = 1, 2, \dots, n, j = 1, 2, \dots, n\}$.

The adjacency matrix $A = \left(a_{ij}\right)_{\substack{i=\overline{1,n} \\ j=\overline{1,n}}}$ of the digraph is defined by:

$$a_{ij} = \begin{cases} 1, & \text{if } d_1 \le j - i \le d_2, \\ 0, & \text{otherwise,} \end{cases} \qquad i = 1, 2, \dots, n, j = 1, 2, \dots, n.$$

To compute the (d_1, d_2)-complexity we use a Floyd-Warshall-type algorithm [5]:

FW(A, n)

```
1  W ← A
2  for k ← 1 to n
3      do for i ← 1 to n
4             do for j ← 1 to n
5                    do w_ij ← w_ij + w_ik w_kj
6  return W
```

If $R = I + W$, where I is the unity matrix, then the (d_1, d_2)-complexity is:

$$K(n; d_1, d_2) = \sum_{i=1}^{n} \sum_{j=1}^{n} r_{ij}.$$

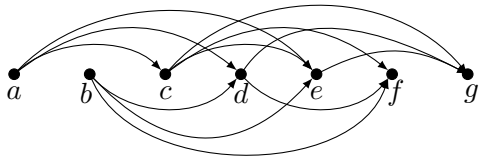

Figure 1: Graph for $n = 7, d_1 = 2, d_2 = 4$.

Example 3 *For digraph in Figure 1 we have the following adjacency matrix:*

$$A = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

The above algoithm give us the matrix W, and by completing with 1's on the first diagonal we obtain the corresponding matrix R:

$$W = \begin{pmatrix} 0 & 0 & 1 & 1 & 2 & 2 & 4 \\ 0 & 0 & 0 & 1 & 1 & 2 & 2 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \qquad R = \begin{pmatrix} 1 & 0 & 1 & 1 & 2 & 2 & 4 \\ 0 & 1 & 0 & 1 & 1 & 2 & 2 \\ 0 & 0 & 1 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

The complexity $K(7; 2, 4) = 30$ (the sum of all entries of the matrix R). The corresponding (2,4)-subwords are:

```
{a} {ac} {ad} {ace,ae} {adf,acf} {aeg,aceg,adg,acg}
{b} {bd} {be} {bdf,bf} {beg,bdg}
{c} {ce} {cf} {ceg,cg}
{d} {df} {dg}
{e} {eg}
{f}
{g}.
```

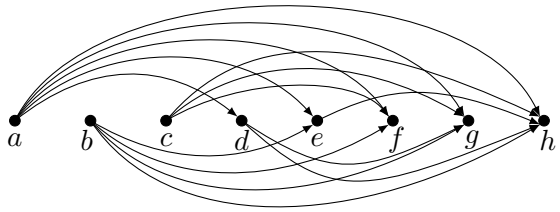

Figure 2: Graph for $n = 8, d_1 = 3, d_2 = 7$.

The Floyd–Warshall-type algorithm combined with the Latin square method can be used to obtain all nontrivial (with length at least 2) (d_1, d_2)-subwords of a given rainbow word $a_1a_2\dots a_n$ of length n [5]. Let us consider a matrix $\mathcal{A}$ with entries A_{ij} which are sets of words. Initially this entries are defined as:

$$A_{ij} = \begin{cases} \{a_ia_j\}, & \text{if } d_1 \le j-i \le d_2, \\ \emptyset, & \text{otherwise,} \end{cases} \qquad \text{for } i = 1, 2, \dots, n,\ j = 1, 2, \dots, n.$$

If $\mathcal{A}$ and $\mathcal{B}$ are sets of words, $\mathcal{AB}$ is the set of concatenation of each word from $\mathcal{A}$ with each word from $\mathcal{B}$:

$$\mathcal{AB} = \big\{ab \,\big|\, a \in \mathcal{A}, b \in \mathcal{B}\big\}.$$

If $s = s_1s_2\dots s_p$ is a word, let us denote by $'s$ the word obtained from s by erasing the first character: $'s = s_2s_3\dots s_p$. Let us denote by $'A_{ij}$ the set A_{ij} in which we erase from each element the first character. In this case $'\mathcal{A}$ is a matrix with elements $'A_{ij}$.

Starting with the matrix $\mathcal{A}$ defined as before, the algorithm to obtain all nontrivial (d_1, d_2)-subwords is the following [5]:

FW-Latin$(\mathcal{A}, n)$

```
1  W ← A
2  for k ← 1 to n
3      do for i ← 1 to n
4             do for j ← 1 to n
5                    do if W_ik ≠ ∅ and W_kj ≠ ∅
6                          then W_ij ← W_ij ∪ W_ik 'W_kj
7  return W
```

The set of (d_1, d_2)-subwords is $\bigcup_{i,j \in \{1,2,\ldots,n\}} W_{ij}$.

Example 4 *For the digraph in Figure 2, when* $n = 8$, $d_1 = 3, d_2 = 7$, *the initial matrix* A *is:*

$$\begin{pmatrix} \emptyset & \emptyset & \emptyset & \{ad\} & \{ae\} & \{af\} & \{ag\} & \{ah\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{be\} & \{bf\} & \{bg\} & \{bh\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{cf\} & \{cg\} & \{ch\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{dg\} & \{dh\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{eh\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}.$$

The result of the algorithm is:

$$\begin{pmatrix} \emptyset & \emptyset & \emptyset & \{ad\} & \{ae\} & \{af\} & \{ag, adg\} & \{ah, adh, aeh\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{be\} & \{bf\} & \{bg\} & \{bh, beh\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{cf\} & \{cg\} & \{ch\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{dg\} & \{dh\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{eh\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}.$$

3 Linear algorithm for the (d_1, d_2)-complexity

In this section linear algorithms to obtain (d_1, d_2)-complexity and (d_1, d_2)-subwords of rainbow words are given.

The following dynamic programming algorithm Complexity computes the (d_1, d_2)-complexity of rainbow word $a_1 a_2 \ldots a_n$. The element x_i $(i = 1, 2, \ldots, n)$ of the array X stores the number of (d_1, d_2)-subwords ending in letter a_i. Letter a_i can be added to all (d_1, d_2)-subwords ending in letters from interval $a_i - d_1 \ldots a_i - d_2$ (assuming that this interval exists). Consequently, value x_i can be computed as the sum of values $x_i - d_1, \ldots, x_i - d_2$. (lines 4, 5, 6) Since letter a_i itself is considered as a (d_1, d_2)-subword, element x_i is initialized with 1 (line 3). The (d_1, d_2)-complexity of a rainbow word of length n is the sum of values x_i $(i = 1, 2, \ldots, n)$ (lines 1 and 7).

COMPLEXITY(n, d_1, d_2)

```
1  k ← 0
2  for i ← 1 to n
3      do x_i ← 1
4          for j ← i − d_2 to i − d_1
5              do if j > 0
6                     then x_i ← x_i + x_j
7          k ← k + x_i
8  return X, k
```

Time complexity of this algorithm is $\Theta(n)$, because the inner loop is executed $n(d_2 - d_1 + 1)$ times.

The following algorithm generates the (d_1, d_2)-subwords of rainbow word $a_1 a_2 \dots a_n$. Bidimensional array (of strings) Y stores the generated (d_1, d_2)-subwords. Array y_i has x_i element and stores the (d_1, d_2)-subwords ending in letter a_i. Operation $s \circ l$ (line 8) means that letter l is added to the end of string s.

SUBWORDS(n, d_1, d_2, A, X)

```
1  for i ← 1 to n
2      do y_i1 ← a_i
3          p ← 1
4          for j ← i − d_2 to i − d_1
5              do if j > 0
6                     then for k ← 1 to x_j
7                              do p ← p + 1
8                                 y_ip ← y_jk ∘ a_i
9  return Y
```

The inner loop is executed $n(d_2 - d_1 + 1) \max_i x_i$ times, so this is a pseudo-linear algorithm.

4 Generalized scattered subwords

Definition 1 can be generalized for rainbow words as we choose letters not only going ahead in the word, but back too at every step.

Definition 5 *Let n, $d_1 \le d_2$ and s be positive natural numbers, and let*

$u = x_1x_2\dots x_n \in \Sigma^n$ be a rainbow word over an alphabet Σ. A rainbow word $v = x_{i_1}x_{i_2}\dots x_{i_s}$, where

$i_1 \geq 1$,
$d_1 \leq |i_{j+1} - i_j| \leq d_2, \quad$ *for* $j = 1, 2, \dots, s-1$,
$i_s \leq n$,

is a ***duplex*** (d_1, d_2)***-subword*** *of length s of u.*

It is worth to note that the definition is given only for rainbow words, and the duplex subwords are rainbow words too.

For example $acfbe$ and $beadfc$ both are duplex (2,4)-subwords of the word $abcdef$.

Definition 6 *The number of all duplex (d_1, d_2)-subwords of a word is the* ***duplex*** (d_1, d_2)***-complexity*** *of that rainbow word.*

We denote the duplex (d_1, d_2)-complexity of a rainbow word of length n by $D(n; d_1, d_2)$, and let x_i, y_i and z_i be the numbers of subwords starting, ending and starting or ending in letter a_i, respectively. Notice that $0 \leq d_2 - d_1 \leq n-1$. For the minimum cases, $d_2 - d_1 = 0$ ($d_1 = d_2 = d$, $d = 1, \dots, n-1$) we have the following recursive formulas (where the sequences X and Y are identical):

$$\begin{aligned}
&x_i = 1, && \text{if } 0 < i \leq d\\
&x_i = x_{i-d} + 1, && \text{if } d < i \leq n\\
&y_i = 1, && \text{if } 0 < i \leq d\\
&y_i = y_{i-d} + 1, && \text{if } d < i \leq n\\
&v_i = x_i + y_i - 1.
\end{aligned}$$

The duplex complexity $D(n; d, d)$ is

$$D(n; d, d) = \sum_{i=1}^{n} v_i.$$

By a simple computation we can obtain the generating functions $F_d(z) = \sum_{n\geq 1} x_n z^n$, $G_d(z) = \sum_{n\geq 1} v_n z^n$, $H_d(z) = \sum_{n\geq 1} D(n; d, d) z^n$:

$$F_d(z) = \frac{z}{(1-z)(1-z^d)}, \quad G_d(z) = \frac{z(1+z^d)}{(1-z)(1-z^d)},$$

$$H_d(z) = \frac{1}{1-z} G_d(z) = \frac{z(1+z^d)}{(1-z)^2(1-z^d)}.$$

Based on Proposition 4 in [5] it is easy to prove the following

Proposition 7 *For integers $n, d \geq 1$, where $n = hd + m$, $0 \leq m < d$,*

$$D(n; d, d) = hn + (h+1)m.$$

The graph method for the case of duplex (d_1, d_2)-subwords can be defined as follows. Let $G = (V, E)$ be a graph (with V the set of vertices, E the set of edges) attached to the rainbow word $a_1 a_2 \dots a_n$ and integers d_1, d_2, where
$V = \{a_1, a_2, \dots, a_n\}$,
$E = \{\{a_i, a_j\} \mid d_1 \leq |j - i| \leq d_2,\ i = 1, 2, \dots, n, j = 1, 2, \dots, n\}$.
Using the attached graph, we can prove the following

Proposition 8

$$D(n; 1, n-1) = n! \sum_{k=0}^{n-1} \frac{1}{k!}.$$

Proof. In this case the attached graph is a complete graph on n vertices, and $D(n; 1, n-1) - n$ is equal to the number of all paths in this graph. In [9] the sequence A007526 is defined by the formula $a_n = n(a_{n-1} + 1)$, and "for $n \geq 1$, $a(n)$ is the number of non-empty sequences with n or fewer terms, each a distinct element of $\{1, 2, \dots, n\}$". So, $a(n)$ is the sum of the number of all paths and the number of all vertices. For a_n in [9] the following formula is given too: $a_n = n! \sum_{k=0}^{n-1} \frac{1}{k!}$. From this:

$$D(n; 1, n-1) = n! \sum_{k=0}^{n-1} \frac{1}{k!}.$$

□

The duplex (d_1, d_2)-complexity of a rainbow word of length n is equal to the number of paths in the attached graph ($NumberOfPaths$). The following modified DFS algorithm (based on [8]) generates all paths in the attached graph G and prints the corresponding duplex subwords. Global arrays DEGREE, PREVIOUS and COLOR have elements indexed from 1 to n. Element $degree_i$ stores the degree of vertex i (corresponding to letter i). Arrays PREVIOUS and COLOR are initiated with zero. Element $previous_i$ stores the previous vertex of vertex i on the current path. We use array COLOR to avoid cycles. Bi-dimensional array NEIGHBOUR stores the neighbor-lists of the vertices of graph G. Element $neighbor_{ik}$ stores the k-th neigbour of vertex i. Procedure DFS(i) prints all distinct paths starting with root-vertex a_i (line

		n								
d_1	d_2	2	3	4	5	6	7	8	9	10
1	1	***4***	*9*	*16*	*25*	*36*	*49*	*64*	*81*	*100*
1	2		**15**	42	101	224	469	944	1849	3552
1	3			**64**	227	716	2111	6058	16971	46546
1	4				**325**	1434	5707	21244	76487	273580
1	5					**1956**	10437	50624	229541	1000106
1	6						**13699**	86114	495161	2662784
1	7							**109600**	794607	5299996
1	8								**986409**	8110482
1	9									**9864100**
2	2		*5*	*8*	*13*	*18*	*25*	*32*	*41*	*50*
2	3			16	45	106	225	474	983	2000
2	4				69	264	853	2432	6683	18560
2	5					378	1855	7708	28209	97200
2	6						2497	14832	75865	343674
2	7							19184	133497	812746
2	8								167513	1334960
2	9									1635970
3	3			*6*	*9*	*12*	*17*	*22*	*27*	*34*
3	4				17	36	91	194	389	756
3	5					62	243	912	2783	7390
3	6						345	1914	9405	37448
3	7							2524	17295	103560
3	8								21901	174694
3	9									214930
4	4				*7*	*10*	*13*	*16*	*21*	*26*
4	5					18	37	64	153	306
4	6						63	186	699	2580
4	7							290	1559	8832
4	8								2075	15794
4	9									19660
5	5					*8*	*11*	*14*	*17*	*20*
5	6						19	38	65	100
5	7							64	187	482
5	8								291	1204
5	9									1722
6	6						*9*	*12*	*15*	*18*
6	7							20	39	66
6	8								65	188
6	9									292

Table 1: Duplex (d_1, d_2)-complexity for rainbow words of length n

3). Procedure PrintCurentPathTo(i) prints the currently generated path from the current root-vertex to vertex a_i (line 8).

```
DuplexSubwords()
 1  NumberOfPaths ← 0
 2  for i ← 1 to n
 3       do DFS(i)
 4  PRINT(NumberOfPaths)
```

```
DFS(i)
 1  color_i ← 1
 2  for k ← 1 to degree_i
 3       do j ← neighbor_ik
 4            if color_j = 0
 5               then previous_j ← i
 6                    DFS(j)
 7  NumberOfPaths ← NumberOfPaths + 1
 8  PrintCurrentPathTo(i)
 9  color_i ← 0
10  previous_i ← 0
```

```
PrintCurrentPathTo(i))
 1  if previous_i ← 0
 2     then PrintCurrentPathTo(previous_i)
 3  PRINT(a_i)
```

5 (d_1, d_2)-complexity and (d_1, d_2)-compositions

Compositions [1, 6, 7] are partitions in which the order of the summands (components) does matter. A (d_1, d_2)-***composition*** is a restricted composition in which the components are natural numbers from the interval $[d_1, d_2]$.

For example, for the word $abcdefg$ the (2,4)-subwords, which begin in a and end in g are: aeg, $aceg$, adg, acg, which correspond to the following compositions in which the components are the distances between the letters in the original word:

$$6 = 4 + 2 = 2 + 2 + 2 = 3 + 3 = 2 + 4.$$

In general, if $a_1 a_{i_1} \cdots a_{i_s} a_{n+1}$ is a (d_1, d_2)-subword of the rainbow word $a_1 a_2 \cdots a_{n+1}$, then this subword corresponds to a composition:

$$n = (i_1 - 1) + (i_2 - i_1) + \ldots + (i_s - i_{s-1}) + (n + 1 - i_s).$$

Let us consider the following (2,4)-subwords: $a_1 a_5 a_7$, $a_1 a_3 a_5 a_7$, $a_1 a_4 a_7$, and $a_1 a_3 a_7$ of the word $a_1 a_2 a_3 a_4 a_5 a_6 a_7$. Then the corresponding (2,4)-compositions are

$$6 = 4 + 2 = 2 + 2 + 2 = 3 + 3 = 2 + 4.$$

So, each (d_1, d_2)-subword of a rainbow word of length $n + 1$, which begins by the first, and ends by the last letter of the rainbow word, corresponds to a (d_1, d_2)-composition of n.

By a simple reasoning we can obtain a formula between the (d_1, d_2)-complexity and (d_1, d_2)-compositions. Let us denote the (d_1, d_2)-composition of n by $C(n; d_1, d_2)$.

Proposition 9 *If $n \geq 1$, then*

$$K(n; d_1, d_2) = n + \sum_{i=1}^{n-1} iC(n - i; d_1, d_2)$$

Example 10 *If $n = 7$, $d_1 = 2$ and $d_2 = 4$, then $K(7; 2, 4) = 30$.*
$C(6; 2, 4) = 4$, because $6 = 2 + 2 + 2 = 2 + 4 = 3 + 3 = 4 + 2$.
$C(5; 2, 4) = 2$, because $5 = 2 + 3 = 3 + 2$.
$C(4; 2, 4) = 2$, because $4 = 2 + 2 = 4$.
$C(3; 2, 4) = 1$, because $3 = 3$.
$C(2; 2, 4) = 1$, because $3 = 2$.
$C(1; 2, 4) = 0$,
and $K(7; 2, 4) = 7 + 1 \cdot 4 + 2 \cdot 2 + 3 \cdot 2 + 4 \cdot 1 + 5 \cdot 1 + 6 \cdot 0 = 30$.

Similarly, if we extend the compositions to the case when instead of the natural numbers, we consider nonzero integers, a relation with the duplex subwords can be given.

Definition 11 *A **generalized composition** of a natural number is one way of writing this number as an ordered sum of nonzero integers, such that all partial sums[1] are positive and different. If the absolute value of the summands are from an interval $[d_1, d_2]$, we call this a **generalized** (d_1, d_2)**-compositions**.*

[1] Partial sums are $\sum_{i=1}^{k} s_i$.

Example 12 *The generalized (2,4)-compositions of 6 are:*

$2+2-3+2+3$	$3+2-4+3+2$
$2+2-3+4-2+3$	$3+2-3+2+2$
$2+2+2$	$3+2-3+4$
$2+3-4+2+3$	$3+3$
$2+3-4+2+2$	$4-3+2+2-3+4$
$2+3-2-2+3+2$	$4-3+2+3$
$2+3-2+3$	$4-3+4-3+4$
$2+4$	$4-3+4-2+3$
$3-2+3-2+4$	$4-2+3-4+2+3$
$3-2+3+2$	$4-2+3-2+3$
$3-2+4-3+2+2$	$4-2+4$
$3-2+4-3+4$	$4+2$.
$3+2-4+3-2+4$	

If $a_1 a_{i_1} \cdots a_{i_s} a_{n+1}$ is a duplex (d_1, d_2)-subword of the rainbow word $a_1 a_2 \cdots a_{n+1}$, then this subword corresponds to a generalized (d_1, d_2)-composition:

$$n = (i_1 - 1) + (i_2 - i_1) + \ldots + (i_s - i_{s-1}) + (n + 1 - i_s).$$

References

[1] S. Heubach, A. Knopfmacher, M. E. Mays, A. Munagi, Inversions in compositions of integers, *Quaest. Math.* **34**, 2 (2011) 187–202. ⇒234

[2] A. Iványi, On the *d*-complexity of words, *Ann. Univ. Sci. Budapest. Sect. Comput.*, **8** (1987) 69–90. ⇒225, 226

[3] Z. Kása, On the *d*-complexity of strings, *Pure Math. Appl.*, **9,** 1–2 (1998) 119–128. ⇒226

[4] Z. Kása, Super-*d*-complexity of finite words, *8th Joint Conf. on Math. and Comput. Sci., Selected Papers*, Komárno, July 14–17, 2010. Novodat, 2011 (eds. H. F. Pop, A. Bege), pp. 257–266. ⇒225, 226

[5] Z. Kása, On scattered subword complexity, *Acta Univ. Sapientiae, Inform.* **3,** 1 (2011) 127–136. ⇒225, 226, 228, 231

[6] C. Kimberling, Enumeration of paths, compositions of integers, and Fibonacci numbers, *Fibonacci Quart.* **39,** 5 (2001) 430–435. ⇒234

[7] C. Kimberling, Path-counting and Fibonacci numbers, *Fibonacci Quart.* **40,** 4 (2002) 328–338. ⇒234

[8] R. Sedgewick, *Algorithms in C, Part 5: Graph Algorithms*, Addison Wesley Professional, 3rd ed., 2001. ⇒232

[9] N. J. A. Sloane, The on-line encyclopedia of integer sequences, `http://www.research.att.com/~njas/sequences/`. ⇒232

Received: March 31, 2012 • Revised: November 30, 2012