Alpha-Numeric Notation for one Data Structure in Software Engineering

Sead Mašović

Computer Science Department, Faculty of Science and Mathematics, University of Niš, P.O. Box 224, Višegradska 33, 18000 Niš, Serbia
sead.masovic@pmf.edu.rs

Muzafer Saračević

Computer Science Department, Faculty of Science and Mathematics, University of Niš, P.O. Box 224, Višegradska 33, 18000 Niš, Serbia
muzafers@uninp.edu.rs

Predrag Stanimirović

Computer Science Department, Faculty of Science and Mathematics, University of Niš, P.O. Box 224, Višegradska 33, 18000 Niš, Serbia
pecko@pmf.ni.ac.rs

Abstract: This paper presents one way to store balanced parentheses notations in shortened form in order to lower memory usage. Balanced parentheses strings are one of the most important of the many discrete structures. We propose new method in software engineering for storing strings of balanced parentheses in shortened form as Alpha-numeric (AN) notation. In addition, our algorithm allows a simple reconstruction of the original strings. Another advantage of the presented method is reflected in savings of working memory when it comes to deal with combinatorial problems. The proposed method is implemented in Java environment.

Keywords: Balanced Parentheses, Catalan number, Binary tree, Software engineering, Java programming.

1 Introduction and Preliminaries

Data structure and software engineering are an integral part of computer science. A binary tree is a tree of data in which each node has at most two descendant nodes, usually distinguished as "left" and "right". Many powerful algorithms in computer sciences and software engineering are tree based algorithms.

The most renown representation of trees is the balanced parentheses [4,8,10,11]. The tree is represented by a string *P* of balanced parentheses of length 2*n*. A node is represented by a pair of matching parentheses "()" and all sub-trees rooted at the node are encoded in an order between the matching parentheses [2,13].

Suppose you would like to form valid sequences of *n* pairs of parentheses, where a string of parentheses is *valid* if it contains an equal number of open and closed parentheses and each open parenthesis has a matching closed parenthesis. For example, "(()())" is valid, but "())()(" is not. A string of parentheses is valid if there is an equal number of open and closed parentheses. The most simple method for presentation of balanced parentheses is to use Bit-string, which uses bit 1 to represent "(" and to use bit 0 to represent ")" in order to represent a well formed Balanced Parentheses (BP shortly).

For example, the Bit-string of the BP expression ()((()))(()) is given by 101110001100.

The number of different well formed parentheses strings represented by Bit-strings $b_{2n}\, b_{2n-1} \ldots b_1$ is given by the Catalan number [3, 6].

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!} \tag{1}$$

For example, for *n*=3, we have

$$C_3 = \frac{(2\times 3)!}{(3+1)!\times 3!} = \frac{6!}{4!\times 3!} = 5$$

and there are five sequences of six balanced parentheses (three pairs of parentheses) as presented in Figure 1.

((())) ()()() (())() ()(()) (()())

Figure 1
Different valid combinations of balanced parentheses for C_3

The BP representation is obtained by traversing the tree in depth-first order writing an open parenthesis when a node in first encountered and a closing parenthesis when the same node is encountered again while going up after traversing its sub-tree. The first representation of trees was proposed in 1989 by Jacobson [5]. The Jacobson representation is called *Level Order Unary Degree Sequence* (LOUDS), which lists the nodes in a level-order traversal. An alternative tree representation based of strings of open and closing parentheses is proposed ten years after results that gave Jacobson et al. [9,10].

In [1] Benot introduced *Depth First Unary Degree Sequence* (DFUDS) representation of an *n*-node tree. DFUDS combines the LOUDS and BP

representations. Three different ways for representing a tree (LOUDS, DFUDS and BP), are presented on Figure 2.

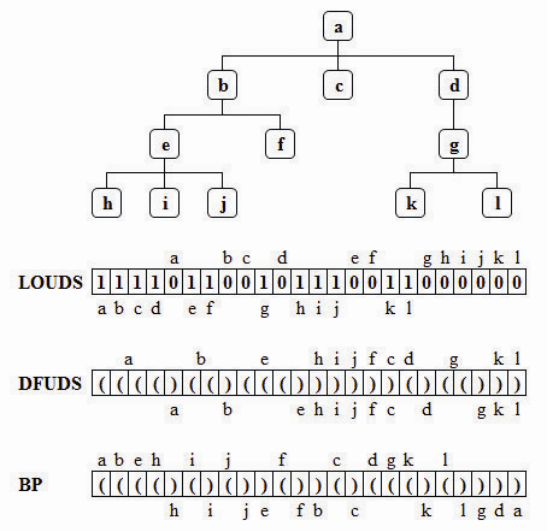


Figure 2
Different representations of tree

To minimize the memory space requirements, we use a bit-string as the basis for introducing further rules for shortened processes.

2 Transformation from BP to AN Notation

In this section we present the way how we get *Alpha-numeric* (shortly AN) notation through shortening process in order to obtain a large reduction of storage space. The shortening procedure is described in the Algorithm 1.

Algorithm 1. Transformation from BP to AN

INPUT : BP notation
m = BP.lenght
Replacement
for (i = 1; $i \leq m$; i++)
(→ 1 AND) → 0
Output1 = BP1 // *Binary Equivalent*
Elimination
Delete first element (1) and last element (0) in BP1
Output2 = BP2 // *Short Binary Eq.*
Selection

```
    e = BP2.element[i];
    m1 = BP2.lenght;
if (Case 2 - binary pairs){
    First binary pair= e[i] for i={1, 2}
    Last binary pair= e[i] for i={m1-1; m1}
    RestBP2= BP2 without First and Last binary pair
    replace (First AND Last binary pair → Alpha2)          // based on Table A3
    Output3 = Alpha2 + RestBP2                             // Alpha Binary
}
if (Case 3 - bit binary group){
    First three bit binary group= e[i] for i={1,2,3}
    Last three bit binary group= e[i] for i={m1-2; m1-1; m1}
    RestBP3= BP2 without First and Last three bit binary group
    replace (First AND Last three bit binary group → Alpha3)   // based on Table A4
    Output3 = Alpha3 + RestBP3                             // Alpha Binary
}
Conversion
if (Case 2 - binary pairs){
    RestBP2 → DecimalEq2.
    Output4 = Alpha2 + DecimalEq2.                         // Alpha Decimal
}
if (Case 3 - bit binary group){
    RestBP3 → DecimalEq3.
    Output4 = Alpha3 + DecimalEq3.                         // Alpha Decimal
}
OUTPUT : AN notation
```

Algorithm 1 consists of four phases, called *replacement, elimination, selection and conversation*):

Step 1 (replacement): Form a binary equivalent of given BP notation, called *b-string*. A bit-string uses bit `1` to represent "`(`" and use bit `0` to represent "`)`".

Step 2 (elimination): The *First shortened form* is obtained by omitting the first and the last bit from the b-string notation. The correctness of this elimination is ensured by the rule that every BP representation begins with the open parenthesis and ends with the closed parenthesis.

Step 3 (selection):

Case 2 - binary pairs*: The *Second shortened form* is obtained by grouping the first two and the last two binary digits of the *First shortened form*. Then the Alpha notation record is based on table grouping (Table A3 - Appendix). The central part (if any) is prescribed.

Case 3 - bit binary group*: In this case, *Second shortened form* is obtained by grouping the first three and the last three binary digits. Then the Alpha notation

record is derived using the special table grouping (Table A4 - Appendix). The central part (if any) is prescribed.

Step 4 (conversation): The final form of the AN notation is obtained by prescribing alpha entry from the second shortened form in the table and converting its central part (the rest) into a decimal number.

The final form of the AN notation is obtained by prescribing alpha entry from the second shortened form in the table and converting its central part (the rest) into a decimal number. Let us mention that Table A1 (Case 2 - binary pairs) and Table A2 (Case 3 - bit binary group) are presented in appendix. Also, the shortening process for the actual value of *n* is illustrated from the initial balanced parentheses over the binary equivalent and usage of special tables to get the AN record as short as possible.

Figure 3 presents the shortening process for one case from the Table A1.

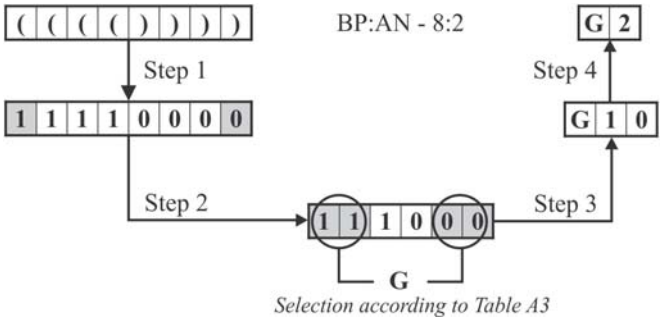


Figure 3
Conversion process of BP notation to AN notation

3 Reverse Transformation from AN to BP Notation

Reverse transformation from AN to BP notation. Based on Algorithm 1 we have possibility to define the reverse transformation of AN notation to the original form of BP notation. The reverse transformation consists of four phases. In the case of the reverse process phase of elimination becomes phase of addition. Description of reverse transformation from AN to BN is given below.
Figure 4 illustrates the reverse construction for the groups of two bits, how to get the original form of BP notation. The same process is taken when it comes for grouping of the initial three bits with ending three bits. The only difference occurs in the selection phase. The conversion in this phase is based on the usage of Table

A3 or Table A4 given in Appendix. More precisely, Table A3 is used for the reverse transformation of the group of two bits, while he reverse transformation of the group of three is based on the usage of Table A4.

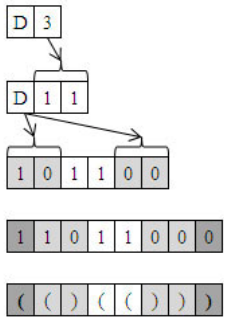


Figure 4
Process of reverse construction

Step 1 (Conversation): Decimal number `3` is converted to binary equivalent. This binary equivalent represents the central part of generated BP notation.

Step 2 (Selection): Alpha notation `D` transforms into corresponding two pairs of binary digits on the basis of rules from the Table A3. The first pair is placed in the beginning and the second at the end of the actual record.

Step 3 (Addition): Add bit `1` at the beginning of string obtained in Step 2 and add the bit `0` at the end of this string.

Step 4 (Replacement): Replace the bit `1` by the open parenthesis `"("` and the bit `0` by the closing parenthesis `")"` and so we get the original form of BP notation.

4 Some Application of the Presented Method

From the aspect of storage, it is very important to make good choices which relates to the application of the most appropriate data structure. A binary tree is an important type of structure which occurs very often in computer science.

BP notation is appropriate representation of binary trees. Balanced parenthesis can be applied in representation of Catalan numbers, which is the basis of many combinatorial problems. Another important application of BP notation is in the process of recording and storing polygon triangulation. This procedure is crucial in computer geometry and graphics in 3D view of images. Increasing the number of polygon vertices drastically increases the number of possible convex polygon triangulations. In order to reduce the memory space requirements, in our paper [12] we propose a shortened form (similar to AN notation) for the storage of generated triangulations. This shortened form presents a unique key for each graph or any combination of triangulations for convex polygons. Another way for representing the polygon triangulation is usage of the Reverse Polish notation, This approach is presented in the paper [7]. Compared to the record that provides Reverse Polish notation, presented AN notation gives much shorter record for the same triangulation. For example, to record one triangulation of hexagon takes eight bits in Reverse Polish notation, while AN notation takes one character and one integer or three bits.

In general case, presented AN method can serve as a model for shortening process in all other problems which are based on Catalan numbers [6]. For example, some of these problems are *Correctly parenthesized expression, Binary records from Lukasiewicz's algorithm, Ballot problem, Problem of the lattice path* and etc. The AN method should be used to save memory space in the storage of records.

5 Implementation Details and Experimental Results

Presented method is implemented in *Java NET Beans environment*. The advantages of Java over the other programming languages are numerous. First of all, programming in the Java programming language is one of the highest degree of programming. These written programs are easily portable from one platform to another.

Our application follows all the steps in obtaining Alpha-numeric notation which is given in Algorithm 1. In an Appendix we gave part of *Java source code*, which is responsible for calling the rules of Alpha notation for groups of binary pairs or bit binary group in order to get the shortened form. Figure 5 present part of the application that implements Algorithm 1 with additional options for presenting and storing results.

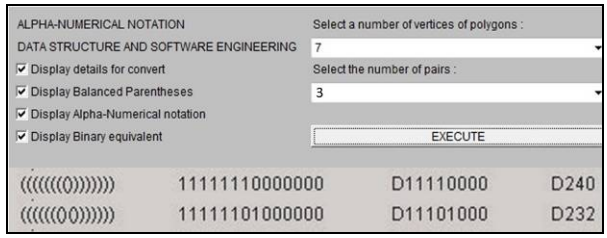
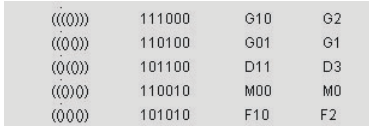
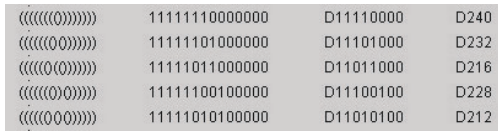


Figure 5
Java application

Figure 6 presents result of the application for both case (binary pairs and bit binary group).



Result screen of the execution of applications for *i=2* and *n=4*



Result screen of the execution of applications for *i=3* and *n=8*

Figure 6
Java panel for both case (binary pairs and bit binary group)

Based on the Algorithm 1 we can set ratio (R) of input (BP notation) and output (AN notation) with the equations:

$$R = \frac{BP}{AN} \tag{2}$$

Where is $BP = 2n$ and $AN = 2(n - i - 1)$ while n is index of Catalan number (C_n) and i is the number of pairs who can take the value from the set {2,3}.

Based on equation (2) in Table 1 are presented ratio of BP and AN notations from two aspects: ratio in the number of characters and ratio in the size of the output file in which the results are stored.

Table 1

Experimental results for testing application

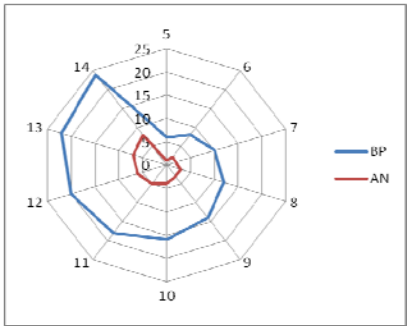
n	**Number of characters**			**File size (KB)**		
	BP	**AN**	***R***	**BP**	**AN**	***R***
5	6	1	6.00	0.5	0.3	1.67
6	8	2	4.00	1	0.6	1.67
7	10	2	5.00	2	1	2.00
8	12	3	4.00	3	1.4	2.14
9	14	3	4.67	7	3	2.33
10	16	4	4.00	26	9	2.89
11	18	5	3.60	95	31	3.06
12	20	6	3.33	386	122	3.16
13	22	7	3.14	1378	446	3.08
14	24	8	3.00	4792	987	4.85

PC performance for testing results: *CPU – Intel (R) Core (TM) 2 Duo, T7700, 2.40 GHz, L2 Cache 4 MB (On-Die, ATC, Full-Speed), RAM Memory - 2 Gb, Graphic card - NVIDIA GeForce 8600M GS.*

Graph 1 presents ratio of shortening process in the number of characters applying AN notation compared to BP notation. Testing was done for n={5,6,…,14}.

Graph 1

Shortening process in the characters



Based on the testing results the advantage of using *AN* notation is when it come to the larger value for *n*, which can be seen in the graph 1. From another point of view if we look at the size of the output file, we can see that with increasing value of *n* is dramatically increasing shortening ratio of the output file for the AN notation (for example, *n*=14, for BP notation output file is 4792 kb while for AN notation is 987kb, which is almoust five times less). For all testing results specified in Table 1 for *n*={5,..,14} we get average shortening of 2,69 time less.

Conclusion

We develop an algorithm that can convert Balanced Parentheses notation in a new shortened Alpha-Numeric notation. With this work we present a way how to reduce BP notation using the appropriate rules in order to lower memory usage in storing. Advantages of presented method is reflected in savings of working memory in the process of testing implementations and thus achieve much better results in terms of speed of execution. Another advantage is that the stored values can be converted through reverse construction to get initial BP notation. We provide an evaluation of the algorithm on a Java implementation.

AN notation may find their application mainly in the problem of triangulation of a convex polygon . It is important to note that the application of this method of recording results may find usage in some combinatorial problems which are based on Catalan numbers.

References

[1] Benoit, D., Demaine, E. D., Munro, J. I., Raman, R., Raman, V., Rao S. S. Representing Trees of Higher Degree, Algorithmica, 2005, Vol. 43, No. 4, pp. 275-292.

[2] Evans D. J., Abdollahzadeh, F. Ecient Construction of Balanced Binary Trees, The Computer Journal, 1983, Vol. 26, No. 3, pp. 193-195.

[3] Geary, R. F., Rahman, N., Raman, R., Raman, V., A Simple Optimal Representation for Balanced Parentheses, Theoretical Computer Science, 2006, Vol. 368, No.3, pp. 231-246.

[4] Gog, S., Fischer, J. Advantages of Shared Data Structures for Sequences of Balanced Parentheses, DCC'10 Proceed. Data Compression Conf. 2010, pp. 406-415.

[5] Jacobson, G. Space-efficient static trees and graphs, In Proceedings of the 30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, IEEE, 1989, pp. 549–554.

[6] Koshy, T. Catalan Numbers with Applications, Oxford University Press, New York, 2009.

[7] Krtolica, P.V., Stanimirovic, P.S., Stanojević, R. Reverse Polish notation method in constructing the algorithms for polygon triangulation, Filomat, 2001, Vol. 15, pp.25–33.

[8] Lu, H., Yeh, C. Balanced Parentheses Strike Back, ACM Transactions on Algorithms (TALG), 2008, Vol. 4, No.3, pp. 1-13.

[9] Munro, I., Raman, V. Succinct Representation of Balanced Parentheses and Static Trees, SIAM Journal on Computing, 2001, Vol. 31, No.3, pp. 762-776

[10] Munro, I., Raman, V., Succinct representation of Balanced Parentheses, static trees and planar graphs, Proceedings of the 38th Annual Symposium on Foundations of Computer Science, IEEE, Miami Beach, Florida, 1997, pp. 118–126.

[11] Ruskey, F., Williams, A. Generating balanced parentheses and binary trees by prefix shifts, In CATS '08: Fourteenth Computing: The Australasian Theory Symposium, Vol. 77 of CRPIT, Wollongong, Australia, 2008.

[12] Saračević, M., Stanimirović, P., Mašović, S., et al., Implementation of some algorithms in computer graphics in Java, TTEM - Technics Technologies Education Management, 2013, Vol. 8, No.1, pp. 293-300.

[13] Tsay, J. Designing a systolic algorithm for generatng well-formed parenthesis strings, Parallel Process. Lett. 2004, Vol. 14, pp.83-97.

Appendix

Table A1
The process of converting from BP to AN for case of binary pair, n={1,2,...4}

n	Number of combinations for Catalan number of n	Balanced parentheses	Binary equivalent	First shortened form	Second shortened form	Final form
1	1	( )	1 0			0
2	1	( ) ( )	1 0 1 0	0 1		1
	2	(( ))	1 1 0 0	1 0		2
3	1	((( )))	1 1 1 0 0 0	1 1 0 0		G
	2	(( ) ( ))	1 1 0 1 0 0	1 0 1 0		F
	3	(( )) ( )	1 1 0 0 1 0	1 0 0 1		E
	4	( ) (( ))	1 0 1 1 0 0	0 1 1 0		C
	5	( ) ( ) ( )	1 0 1 0 1 0	0 1 0 1		B
4	1	(((( ))))	1 1 1 1 0 0 0 0	1 1 1 0 0 0	G 1 0	G 2
	2	((( ) ( )))	1 1 1 0 1 0 0 0	1 1 0 1 0 0	G 0 1	G 1
	3	(( ) (( )))	1 1 0 1 1 0 0 0	1 0 1 1 0 0	D 1 1	D 3
	4	((( )) ( ))	1 1 1 0 0 1 0 0	1 1 0 0 1 0	M 0 0	M 0
	5	(( ) ( ) ( ))	1 1 0 1 0 1 0 0	1 0 1 0 1 0	F 1 0	F 2
	6	( ) ((( )))	1 0 1 1 1 0 0 0	0 1 1 1 0 0	A 1 1	A 3
	7	( ) (( ) ( ))	1 0 1 1 0 1 0 0	0 1 1 0 1 0	C 1 0	C 2
	8	(( )) (( ))	1 1 0 0 1 1 0 0	1 0 0 1 1 0	F 0 1	F 1
	9	( ) ( ) (( ))	1 0 1 0 1 1 0 0	0 1 0 1 1 0	C 0 1	C 1
	10	((( ))) ( )	1 1 1 0 0 0 1 0	1 1 0 0 0 1	H 0 0	H 0
	11	(( ) ( )) ( )	1 1 0 1 0 0 1 0	1 0 1 0 0 1	E 1 0	E 2
	12	( ) (( )) ( )	1 0 1 1 0 0 1 0	0 1 1 0 0 1	B 1 0	B 2
	13	(( )) ( ) ( )	1 1 0 0 1 0 1 0	1 0 0 1 0 1	E 0 1	E 1
	14	( ) ( ) ( ) ( )	1 0 1 0 1 0 1 0	0 1 0 1 0 1	B 0 1	B 1

Table A2
The process of converting from BP to AN for case of bit binary group, n={6,7,8,9}

n	Number of combinations for Catalan number of n	Balanced parentheses (several examples)	Binary equivalent	First shortened form	Second shortened form	Final form
6	132	((((((())))))) (((((()())))))	1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0 0 0 0	1 1 1 1 1 0 0 0 1 1 1 1 0 1 0 0 0	D 1 1 0 0 D 1 0 1 0	D 1 2 D 1 0
7	429	(((((((()))))))) ((((((())))))) (((((()()))))	1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 0 0 0 0 0 1 1 1 1 1 0 1 1 0 0 0 0 0 0	1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 0 1 0 0 0 0 1 1 1 1 0 1 1 0 0 0 0	D 1 1 1 0 0 0 D 1 1 0 1 0 0 D 1 0 1 1 0 0	D 5 6 D 5 2 D 4 4
8	1430	((((((((())))))))) (((((((()(()))))))) ((((((()(())))))))	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 1 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 1 1 1 1 0 1 1 0 0 0 0 0 0	D 1 1 1 1 1 0 0 0 0 D 1 1 1 1 1 0 1 0 0 0 D 1 1 0 1 1 0 0 0	D 2 4 0 D 2 3 2 D 2 1 6
9	4862	(((((((((()))))))))) (((((((((())))))))) (((((((()(())))))))	1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 1 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 1 1 1 1 1 0 1 1 0 0 0 0 0 0	D 1 1 1 1 1 0 0 0 0 0 D 1 1 1 1 1 0 1 0 0 0 D 1 1 1 0 1 1 0 0 0 0	D 9 9 2 D 9 7 6 D 9 4 4

Table A3
Codebook for binary pair grouping

Number of combination	First binary pair		Alpha notation	Last binary pair	
	1	2		m - 1	m
1	0	1	A	0	0
2	0	1	B	0	1
3	0	1	C	1	0
4	1	0	D	0	0
5	1	0	E	0	1
6	1	0	F	1	0
7	1	1	G	0	0
8	1	1	H	0	1
9	1	1	M	1	0

Table A4
Codebook for bit binary grouping

Three bit binary group			Alpha notation	Three bit binary group		
1	2	3		m - 2	m - 1	m
0	1	0	A	0	0	0
0	1	1	B	0	0	0
1	0	1	C	0	0	0
1	1	1	D	0	0	0
1	1	0	E	0	0	0
1	0	0	F	0	0	0
0	1	0	G	0	1	1
0	1	1	H	0	1	1
1	0	1	I	0	1	1
1	1	1	J	0	1	1
1	1	0	K	0	1	1
1	0	0	L	0	1	1
0	1	0	M	1	0	1
0	1	1	N	1	0	1
1	0	1	O	1	0	1
1	1	1	P	1	0	1
1	1	0	Q	1	0	1
1	0	0	R	1	0	1
0	1	0	S	1	1	1
0	1	1	T	1	1	1
1	0	1	U	1	1	1
1	1	1	V	1	1	1
1	1	0	W	1	1	1
1	0	0	X	1	1	1

Three bit binary group			Alpha notation	Three bit binary group		
1	2	3		m - 2	m - 1	m
0	1	0	Y	1	1	0
0	1	1	Z	1	1	0
1	0	1	a	1	1	0
1	1	1	b	1	1	0
1	1	0	c	1	1	0
1	0	0	d	1	1	0
0	1	0	e	1	0	0
0	1	1	f	1	0	0
1	0	1	g	1	0	0
1	1	1	h	1	0	0
1	1	0	i	1	0	0
1	0	0	j	1	0	0
0	1	0	k	0	1	0
0	1	1	l	0	1	0
1	0	1	m	0	1	0
1	1	1	n	0	1	0
1	1	0	o	0	1	0
1	0	0	p	0	1	0
0	1	0	q	0	0	1
0	1	1	r	0	0	1
1	0	1	s	0	0	1
1	1	1	t	0	0	1
1	1	0	u	0	0	1
1	0	0	v	0	0	1

Java source code:

Part of the Java source code which convert the expression into binary equivalent in the case where we have shortening process with two binary pairs. The main class `AN_Notation` contain the method `notation()` which realize Algorithm 1 through four phase:

Step 1 (*Replace*): Converts BP notation into binary equivalent.

```
BP1 = BP.replace("(","1");
BP1 = BP.replace(")","0");
binaryEq = BP1;
```

Step 2 (*Elimination*): Process of elimination of the first and the last bit (`1-0`)

```
BP2 = binaryEq.substring(1,LabelBP.length()-1);
```

Step 3 (*Selection*): Selection of the first and the last two binary pairs of record and we write Alpha notation record based on the table grouping (Table A3)

```
int aLenght = BP2.length();
String aFirst = BP2.substring(0,2);
String aLast = BP2.substring(aLenght-2,aLenght);
String str0="00";
String str1="01";
String str2="10";
String str3="11";
// codebook – table A3 (FROM A TO M)
if (aFirst.equals(str1) && aLast.equals(str0)) Alfa="A";
if (aFirst.equals(str2) && aLast.equals(str0)) Alfa="B";
if (aFirst.equals(str1) && aLast.equals(str2)) Alfa="C";
...
if (aFirst.equals(str3) && aLast.equals(str0)) Alfa="M";
```

Step 4 (*Conversion*): Converts the central part (the rest) into Decimal number.

```
CentralBin = BP2.substring(2,BP2.length()-2);
long numSk = Long.parseLong(CentralBin);
long remSk;
while(numSk > 0){
remSk = numSk % 10;
numSk = numSk / 10;}
int CentDec= Integer.parseInt(CentralBin,2);
CentralDec = Integer.toString(CentDec);
```