Bioinformation Volume 5 open access

www.bioinformation.net Issue 8 **Software**

DNABIT Compress – Genome compression algorithm

Pothuraju Rajarajeswari[1]*, Allam Apparao[2]

[1]DMSSVH college of Engineering, Machilipatnam; [2]Jawaharlal Nehru Technological University, Kakinada; Pothuraju Rajarajeswari – Email: rajilikhitha@gmail.com; *Corresponding author.

Received September 21, 2010; Accepted October 29, 2010; Published January 22, 2011

Abstract:
Data compression is concerned with how information is organized in data. Efficient storage means removal of redundancy from the data being stored in the DNA molecule. Data compression algorithms remove redundancy and are used to understand biologically important molecules. We present a compression algorithm, "DNABIT Compress" for DNA sequences based on a novel algorithm of assigning binary bits for smaller segments of DNA bases to compress both repetitive and non repetitive DNA sequence. Our proposed algorithm achieves the best compression ratio for DNA sequences for larger genome. Significantly better compression results show that "DNABIT Compress" algorithm is the best among the remaining compression algorithms. While achieving the best compression ratios for DNA sequences (Genomes),our new DNABIT Compress algorithm significantly improves the running time of all previous DNA compression programs. Assigning binary bits (Unique BIT CODE) for (Exact Repeats, Reverse Repeats) fragments of DNA sequence is also a unique concept introduced in this algorithm for the first time in DNA compression. This proposed new algorithm could achieve the best compression ratio as much as 1.58 bits/bases where the existing best methods could not achieve a ratio less than 1.72 bits/bases.

Keywords: Ziv-Lempel, BioCompress, GenCompress, Arithmetic coding, DNA compress.

Background:
Life is strongly associated with organization and structure **[1]**.With the completion of 1000 genomes project, the project is estimated to generate about 8.2 billion bases per day, with the total sequence to exceed 6 trillion nucleotide bases. The DNA molecule is made up of a concatenation of four different kinds of nucleotides namely: Adenine, Thymine, cytosine and Guanine (A,T,C,G).General purpose compression algorithms do not perform well with biological sequences. Giancarlo *et al*. **[2]** have provided a review of compression algorithms designed for biological sequences. Finding the characteristics and comparing Genomes is a major task (Koonin 1999**[3]**; Wooley 1999**[4]**). In mathematical point of view, compression implies understanding and comprehension (Li and Vitanyi 1998) **[5]**. Compression is a great tool for Genome comparison and for studying various properties of Genomes. DNA sequences, which encode life should be compressible. It is well known that DNA sequences in higher eukaryotes contain many tandem repeats, and essentials genes (like rRNAs) have many copies. It is also proved that genes duplicate themselves sometimes for evolutionary purposes. All these facts conclude that DNA sequences should be compressible. The compression of DNA sequences is not an easy task. (Grumback and Tahi 1994**[6]**, Rivals *et al*. 1995 **[7]**; Chen *et al*. 2000 **[8]**) DNA sequences consists of only four nucleotides bases {a,c,g,t}. Two bits are enough to store each base. The standard compression softwares such as "compress", "gzip", "bzip2", "winzip" expanded the DNA genome file more than compressing it. Most of the Existing software tools worked well for English text compression (Bell *et al*. 1990**[9]**) but not for DNA Genomes. Increasing genome sequence data of organisms lead DNA database size two or three times bigger annually. Thus it becomes very hard to download and maintain data in a local system. Other algorithms specifically designed for DNA sequences compression did not manage to achieve average compression rate below 1.7 bits/base. Algorithms for Compressing DNA sequences, such as Ziv-Lempel compression algorithms **[10, 11]**. Biocompress **[12]**, Gencompress **[13]** and DNAcompress **[14]** compress DNA sequences. Their average compression rate is about 1.74 bits per base. Hence we present a new compression algorithm named "DNABIT Compress" whose compression rate is below 1.56 bits per base (for Best case) even for larger genome (nearly 2,00,000 characters).

Existing compression algorithms:
General purpose compression algorithms do not perform well with biological sequences, resulting quite often in expansion rather than compression. Probably one of the most well-known DNA compressors is Ziv-Lempel compression algorithms **[10, 11]** are based on an idea of complexity presented by Lempel and Ziv in **[15]**. Grumbach and Tahi proposed two lossless compression algorithms for DNA sequences, namely Biocompress and Biocompress-2 using the technique of Ziv and Lempel (1997) data compression method. Biocompress-2 detects exact repeats and complementary palindromes located in the target sequence and then encode them by repeat length and the position of a previous repeat occurrence. Biocompress-2 also uses arithmetic coding of order 2 if no significant repetition is found. Biocompress also uses the same methodology as in Biocompress-2 except that it does not use order-2 arithmetic coding. Gencompress (Chen *et al*. 2000 **[13]**) achieves significantly higher compression ratios than either Biocompress-2 or Cfact. Gencompress is a one-pass algorithm. In Gencompress we search for approximate matches that satisfy condition C. This algorithm carefully finds the optimal prefix and uses order-2 arithmetic encoding (Nelson 1991; Bell *et al.* 1990) whenever needed. Gencompress also detects the approximate complemented palindrome in DNA sequences. The Average compression ratio is 1.7428. DNA compress employs the Lempel-Ziv compression scheme as Biocompress-2 and Gencompress. It consists of two components: find all approximate repeats including complemented palindromes; and encode approximate repeat regions and non-repeat regions. The Average ratio used for compressing sequences found to be 1.7254. Behzadi and Fessant find repeats to the cost of a dynamic programming search and select from a second order markov model, a

Bioinformation Volume 5 open access
www.bioinformation.net Issue 8 **Software**

context tree and two-bit coding for the non-repeated parts. They use an expert model with Bayesian averaging over a second order Markov model, a first order Markov model estimated on short term data (last 512 symbols) and a repeat model. Expressed per element complexity can provide information about the structure of the regions and local properties of a genome, or proteome. Perhaps more than their use in compressing biological sequences, compression algorithms, in particular variants of the Ziv-Lempel algorithms **[10, 11]**, have been useful as measures of evolutionary distance (Li *et al.*) **[17]** used. Gencompress is used as the distance estimator in hierarchical clustering of biological sequences as a solution to the phylogeny construction problem.

Methodology:
Please see Supplementary material for methodology.

Experimental Results:
To experiment our algorithm, we tried to compress a standard set of DNA sequences with our algorithm, and we compare with results published for other efficient DNA compressors. A test prototype is implemented to assess the capability of our framework. The code is written in java and compiled in java 1.2 SDK. Tests ran on a system based on intel Pentium we tested our DNABIT program on a dataset of DNA sequences typically used in DNA compression studies. The datasets includes 9 sequences: two chloroplast genomes (CHMPXX AND CHNTXX); 4 HUMAN GENES [HUMDYSTROP, HUMHBB, HUMHDABCD (Humanbeta globinregion on chromosome11) and HUMHPRTB (Humanhypoxanthine phosphoribosyl transferase gene); 1 mitochondria genome (MPOMTCG); and genomes of 2 Viruses (HEHCMVCG AND VACCG)

The results are displayed on **Table 10 (see Supplementary material)**. The table shows that our proposed new DNABIT compress algorithm program performs better than other programs. Other existing algorithms that are used to compare with the proposed DNABIT are Normal CTW, CTW+LZ, BIOCOMPRESS 2, GENCOMPRESS, DNA COMPRESS, DNA PACK. During our experiments, we tried to compress the DNA genomes as long as nearly 2 lakhs length. The time taken to compress is approximately in seconds. Our proposed algorithm DNABIT compress performs better than the best algorithms GENCOMPRESS and DNA PACK whose compression ratio is approximately 1.7. Our DNABIT compress algorithm's compression ratio is as less as 1.58 bits/base.

DNABIT Compress Comparison with other Compressors:
Compression algorithms designed to compress DNA sequences comparison is a difficult task because the source or executable code of compressors are usually not available. Another reason is space and time requirement of most compressors makes it difficult to test them on sequences of larger Genomes. Complete data on the running times is not available. In **[14]** the authors report some compression times for the file hehcmvcg (229354 bases) on a 700MHz Pentium III. According to **[14]** the compression of hehcmvcg takes a few hours for CTW+LZ, 51 seconds for GenCompress-2 **[8]**, but only 10 Seconds for our proposed algorithm DNABIT Compress. Unfortunately, we could not test in depth the performance of DNACompress. Since it is based on the PatternHunter **[17]** search engine which is not freely available.
Finally, we would like to comment on the performance of Off-line which, like our algorithms, only encodes exact repeats. While we use a simple and fast procedure to find repeats, Off-line is designed to search for an "optimal" sequence of textual substitutions. It is therefore not surprising that Off-line is much slower than our algorithms (for the yeast sequences the difference is by a factor 1000 or more). The experimental results show that our algorithms are an order of magnitude faster than the other DNA compressors and that their compression ratio is very close to the one of the best compressors.The speed of our algorithm and its small working space have been able to compress sequences of length up to 220MB, which are well beyond the range of any other DNA compressor. **Figure 1** shows the comparison ratios of proposed DNABIT compress algorithm with popular existing algorithms.

DNA COMPRESS JAVA TOOL:
The proposed DNABIT compress tool screens are shown in **Figure 2, 3, 4**. The tool has the option of selecting the type as either encrypt or decrypt. The DNA genomes of any length can be given as Input in the input column selecting the encryption option. The compression ratio is displayed in the output column with the time taken for computing. The encrypted text can be decrypted back to its original DNA sequence by using the decryption option. The original DNA text is displayed in the output option. Sample Code is given in **Supplementary material.**

Detecting Tandem repeats of Repetitive Nucleotides:
In addition to the compression techniqu , an algorithm is designed to find tandem repeats as DNABIT compress enlighthens on compressing repetitive bases The detection of tandem repeats is important in biology and medicine as it can be used for phylogenic studies and disease diagnosis. This paper proposes two techniques for detecting approximate tandem repeats (ATRs) in DNA sequences. We have proposed a algorithm to calculate the tandem repeats since our proposed DNABIT compress focuses on repetitive Bases. our algorithm calculates consequetive similar two repeats (di nucleotides) and three repeats (tri nucleotides or codons). Tandem repeats (TRs) are defined as two or more contiguous approximate copies of a pattern of nucleotides. Tandem repeats have been known to play important roles in human disease, regulation, and evolution.

Repetitive structures are present in over one-third of the human genome **[18]**. The expansion of the trinucleotide repeat results in anticipation or progression in severity of the disorder through each generation. In general, there is a correlation between the size of the expansion and the severity of the phenotype. Furthermore, instabilities in dinucleotide repeat sequences have been observed in colon cancer **[19]** Some biological mechanisms for the expansion of repeats include: defect in mismatch repair system, polymerase slippage during replication, and genetic instability of some DNA structures **[20, 21]**. Repeats play a role in gene regulation when present in regions with transcription factors **[22]**. Sample code is given in **Supplementary material**.

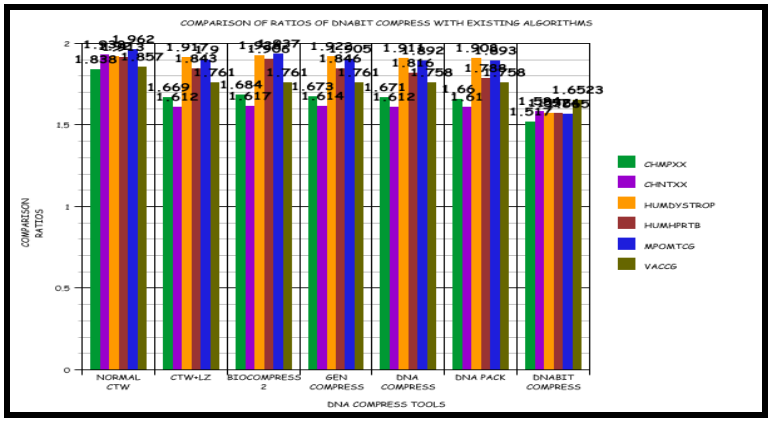


Figure 1: Comparison of ratios of DNABIT Compress with existing algorithms

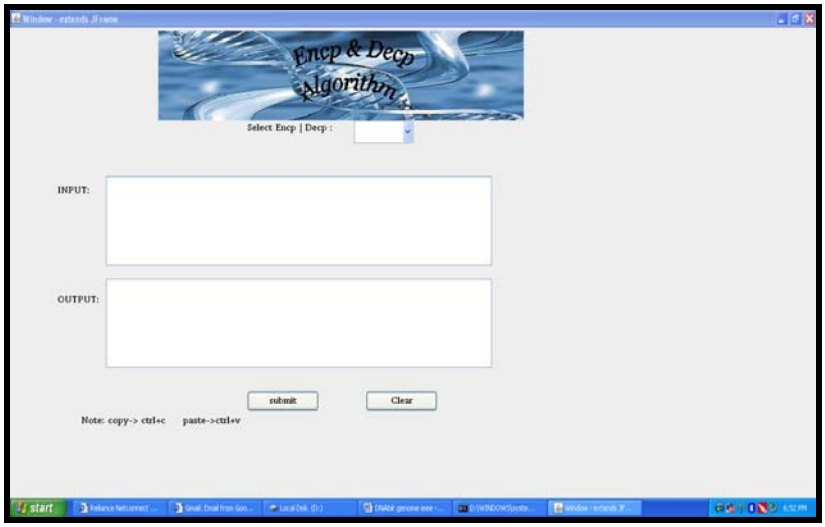


Figure 2: DNAcompress Tool Screen

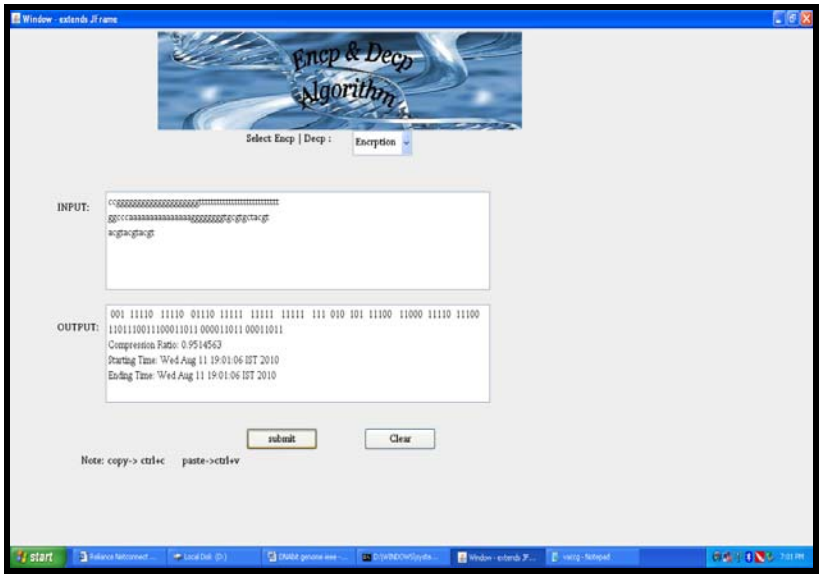


Figure 3: Compression Ratio displayed in the output Box.

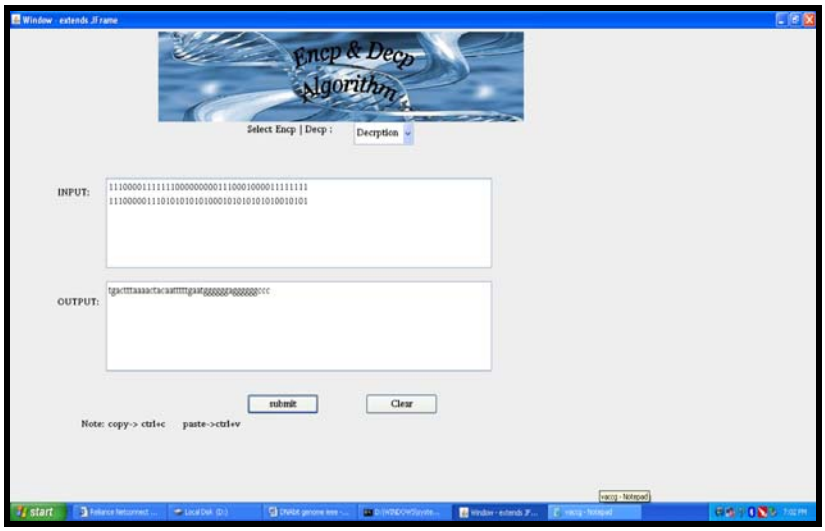


Figure 4: Decompressed Text in the output Box.

Conclusion:
A simple DNA compression algorithm which is completely new in its design is proposed to compress DNA sequences which are repetitive as well as non repetitive in nature. Data compression reveals certain theoretical ideas such as entropy, mutual information and complexity between sequences of different genomes. Data compression also plays a vital role in analyzing biological sequences to discover hidden patterns, infer phylogenetic relationship between organisms which are areas of active research in bioinformatics. If the sequence is compressed using DNABIT Compress algorithm, it will be easier to compress large bytes of DNA sequences with the average compression ratio of 1.5359 bits per base which will be very useful in sequence comparisons and Multiple sequence Alignment analysis. The simplicity and flexibility of DNABIT Compress algorithm could make it an invaluable tool for DNA compression in clinical research.

Limitations:
Summative evaluation of learning outcomes such as testing the real biological sequences on this algorithm, performance with the tools, or the transfer of knowledge to similar tasks could not be performed.

Future work:
(1)The compression algorithm can be applied to calculate phylogeny and multiple sequence alignment of various DNA sequences.
(2) Developing a new DNA compressor with compressed text data rather than Binary bits.

Acknowledgements:
Authors are thankful for the support rendered by V. K. Kumar during its development phase.

References:
[1] E Schrodinger. *Cambridge University Press*: Cambridge, UK, 1944. [PMID: 15985324]
[2] R Giancarlo *et al. A synopsis Bioinformatics* **25**:1575 (2009) [PMID: 19251772]
[3] EV Koonin. *Bioinformatics* **15**: 265 (1999)
[4] JC Wooley. *J.Comput.Biol* **6**: 459 (1999) [PMID: 10582579]
[5] CH Bennett *et al. IEEE Trans.Inform.Theory* **44**: 4 (1998)
[6] S Grumbach & F Tahi. *Journal of Information Processing and Management* **30**(6): 875 (1994)
[7] E Rivals *et al.* A guaranteed compression scheme for repetitive DNA sequences. LIFL, Lille I University, technical report IT-285 (1995)
[8] X Chen *et al.* A compression algorithm for DNA sequences and its applications in Genome comparison. In Proceedings of the Fourth Annual International Conference on Computational Molecular Biology, Tokyo, Japan, April 8-11, 2000. [PMID: 11072342]
[9] TC Bell *et al.* Newyork:Prentice Hall (1990)
[10] J Ziv & A Lempel. *IEEE Trans. Inf. Theory* **23:** 337 (1977)
[11] J Ziv & A Lempel. *IEEE Trans. Inf.Theory*, **24:** 530 (1978) [PMID: 20157474]
[12] A Grumbach & F Tahi. In Proceedings of the IEEE Data Compression Conference, Snowbird, UT, USA, March 30–April 2, 1993.
[13] X Chen *et al.* In Proceedings of the Fourth Annual International Conference on Computational Molecular Biology, Tokyo, Japan, April 8-11, 2000.
[14] X Chen *et al. Bioinformatics* **18**: 1696 (2002) [PMID: 12490460]
[15] A Lempel & J Ziv. *IEEE Trans. Inf. Theory* **22:** 75 (1976)
[16] M Li *et al. IEEE Trans. Inf. Theory* **50:** 3250 (2003)
[17] B Ma *et al. Bioinformatics* **18**:440 (2002) [PMID: 11934743]
[18] ES Lander *et al. Nature* **409**: 860 (2001) [PMID: 11237011]
[19] S Thibodeau *et al. Science* **260**: 816 (1993) [PMID: 14988818]
[20] M Mitas. *Nucleic Acids Research* **25**: 2245 (1997) [PMID: 9618442]
[21] RD Wells. *Journal of Biological Chemistry* **271**: 2875 (1996) [PMC: 178294]
[22] M Perutz. *Current Opinion in Structural Biology* **6**: 848 (1996) [PMC: 1692597]

Edited by R Sowdhamini
Citation: **Rajarajeswari & Apparao,** Bioinformation 5(8): 350-360 (2011)

Bioinformation Volume 5 open access

www.bioinformation.net Issue 8 **Software**

Supplementary material:

Methodology:

Proposed Algorithm – DNABIT Compress:

In this paper we consider the problem of DNA compression both for repetitive and non repetitive DNA sequences. To improve the compression rate, a new technique named DNABIT Compress has been devised, which is much effective with respect to compression rate. Simple AG/TC similarities are not considered in this abstraction of data into primary bits. The new proposed algorithm to compress DNA sequences of complete Genomes is of Two Phases. The Two phases in turn uses five different techniques namely 2 Bit Technique, 3, 5, 7, and 9 Bit Technique.

The two Phases are: (1) Even Bit Technique: 2Bit Technique; (2) Odd Bit Technique: 3Bit Technique, 5Bit Technique, 7Bit Technique, 9Bit Technique.

Even Bit Technique:

The DNA sequence is assigned two bits for every individual base. Bases are {a,c,g,t}. Two bits are assigned to bases of non-repeat regions. The bits assigned to individual bits are shown in **Table 1**.

Odd Bit Technique:

The Odd Bit technique employs 4 different Techniques. They are 3Bit Technique, 5Bit Technique, 7Bit Technique, 9 Bit Technique

3 Bit Technique:

In the DNA sequence if two or three similar bases exists next to one another, this 3 Bit technique is applied. The encoded string is represented as 3 Bit CODE. In the 3 Bit CODE, first significant bit is allocated as either "1"or "0". "0"represents the significant bit if the base repeat is two times. "1" represents the significant bit if the base repeat is three times. The 3 bit code is in **Table 2, 3**.

5 BIT Technique:

In the DNA sequence if there exists more than 3 repeats upto 8 repeats (4,5,6,7,8) Three to eight similar bases next to one another, this 5 Bit technique is applied. The encoded string is represented as 5 Bit CODE **(Table 4-8)**.

7 BIT Technique:

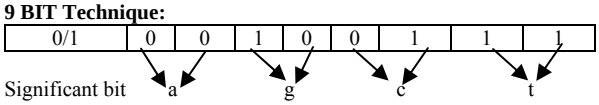
7 BIT CODE for 2 EXACT BASE REPEATS:

In the given string if there is 2 characters repeat more than 1time upto 8 times, we represent it in 7 bit sequence. In this 7 bit code, first 3 bits represent the number of repeats of that characters. The other 4 bits represent the code for that characters **(Table 9)**.

9 BIT Technique:

In this 9 BIT CODE, there are two techniques. If the consecutive 4 bases are same, then the encoded string is taken to be 9 bit CODE. The first significant bit either represents as "0"or "1". "0" indicates that the repeat is exact repeat. "1" indicates that the repeat is reverse repeat.

9 BIT Technique:



In the 9 bits the first significant bit represents same or reverse. The other 8 bits represents CODE for each base.

Calculation of Compression Ratio:

LET £ = Total number of bases in the given sequence.

Compression Rate = Number of Bits/Total number of Bases.

μ = Number of 3 BIT CODE * 3

β = Number of 5 BIT CODE * 5

δ = Number of 7 BIT CODE * 7

γ = Number of 9 BIT CODE * 9

þ = Number of even BIT CODE * 2(individual base sequences)

Total number of Encoded Binary Bits = (¥ + Ψ + δ + γ+ þ)

Ratio Ř = (μ + β + δ + γ+ þ) / £ Bits/Bases.

Example: Gggggagctacgagctagctacaccacatatatatatatatata

Encoded:

1000100011011001001000011011100011000001111011111100

Ratio = 2*3 + 5*1 +7*1 + 9*2 +2*8 / 50

= 52/50

= 1.04 bits/bases

Best Case: The Best-case efficiency is proved in our proposed new DNABIT COMPRESS algorithm since its compression Rate = 1.04 for Best case and 1.58 for Worst case. Worst case is defined when the number of repetitions in the input DNA sequence is neglible, which is the best among all the present existing DNA compression Algorithms.

Analysis for various cases:

CASE 1: DNA sequence applying EVEN BIT TECHNIQUE.

Examples:

Input String = agctatcg

The above strings are non-repeated bases where every individual base is not a repetition of the next base.

The encoded string is: 00 01 10 11 00 11 10 01

CASE 2: DNA SEQUENCE applying ODD BIT TECHNIQUE-3 BIT TECHNIQUE.

a) 3 BITCODE for Exact 2 Repeat Bases.

Example:

Input string = aa t c a

Encoded string = 000 11 01 00

b) 3 BIT CODE for Exact 3 Repeat Bases

Example:

Input string = t c a ggg

Encoded string = 11 01 00 110

CASE 3: DNA SEQUENCE applying ODD BIT TECHNIQUE-5 BIT TECHNIQUE.

a) 5 BIT CODE for EXACT 4 Repeat Bases.

Example:

Input string = aaaa gggg g t

Encoded String = 01100 01110 10 11

b) 5 BIT CODE for EXACT 5 Repeat Bases.

Example:

Input string = aaaaa ggggg g t

Encoded String = 10000 10010 10 11

c) 5 BIT CODE for EXACT 6 Repeat Bases.

Example:

Input string = tttttt aaaaaa c

Encoded string = 10111 10100 01

d) 5 BIT CODE for EXACT 7 Repeat Bases.

Example :

Input string = ccccccc aaaaaaa c

Encoded string = 11001 11000 01

5 BIT CODE for EXACT 8 Repeat Bases.

Example:

Input String = tttttt a c gggggggg

Encoded string = 10111 00 01 11110

In the above sequence "t" repeat is 6 times. So binary representation of 6 is "101". The first 3 bits are code for 6. The other 2 bits are the code for character "t". 11110 = "g" is repeat in 8 times binary code for 8 is "111". The next 10 is code for "g".

CASE 4: DNA SEQUENCE applying ODD BIT TECHNIQUE-7 BIT TECHNIQUE.
a) 7 BIT CODE For EXACT 2 Repeat Bases 6 times.
Example:
Input String : cacacacacaca
Encoded String : 101 01 00

b) 7 BIT CODE For EXACT 2 Repeat Bases 8 times.
Example:
Input String : cacacacacacaca
Encoded String: 111 01 00

c) 7 BIT CODE For EXACT 2 Repeat Bases 4 times.
Example:
Input String: ta ta ta ta
Encoded String: 011 11 00

CASE 5: DNA SEQUENCE applying ODD BIT TECHNIQUE-9 BIT TECHNIQUE.
a) 9 BIT CODE for EXACT 4 Repeat Bases.
Example:
Input String = agct agct
Encoded String = 0 00 10 01 11

b) 9 BIT CODE for EXACT REVERSE 4 Repeat Bases.
Example:
Input String = tata atat
Encoded String = 1 00 1100 11

DNABIT COMPRESS ENCODING ALGORITHM:
Input: Input String (INSTRING) Containing A, T, G and C
Output: Encoded String (OUTSTRING)

Procedure Encode:
Begin
1: Divide the given DNA sequence in to fragments, where each fragment consists of 2 characters, 4 characters.
2: Generate all possible combinations of DNA sequence (A, C, G, T).
3: Apply Even Bit Technique if the simultaneous bases do not match with each other. (The DNA sequence is assigned two bits for every individual base of non-repeat regions.)
4: If there exists two or three similar bases next to one another, the 3 Bit technique is applied.
5: If there exists more than 3 repeats upto 8 repeats (4,5,6,7,8) Three to eight similar bases next to one another, the 5 Bit technique is applied. The encoded string is represented as 5 Bit CODE.
6: In the given string if there is 2 characters repeat more than 1 time upto 8 times, it is represented as 7 bit code. In this 7 bit code, first 3 bits represent the number of repeats of that character. (The other 4 bits represent the code for that character.)
7: If the consecutive 4 bases are same, then the encoded string is taken to be 9 bit CODE. The first significant bit either represents as "0"or "1". "0" indicates that the repeat is Exact repeat. "1" indicates that the repeat is reverse repeat.
8: Transfer the binary bits to the output String (OUTSTRING).
End
The Decryption algorithm involves the same procedure as Encryption in the reverse form.

DNABIT COMPRESS DECODING ALGORITHM:
Input: Input String
Output: Decoded String (DECSTRING)

Procedure Decode:
Begin
1: Generate all possible combinations for {A, C, G, T}.
2: Allocate unique binary bit number (0 and 1) to each combination.
3: Divide given binary code in to segments.
4: According to the Binary code (either 3 BIT CODE, 5 BIT CODE, 7 BIT CODE, or 9 BIT CODE) assign appropriate base {a, c, g, t}.
5: Repeat step 4, until the end of the input sequence is reached.
6: If there are any individual bases (non repeat regions) the corresponding binary code gets transformed. (Assigned values for bases are :a="00", G="01",c="10",t="11").
End

Examples:
The Total number of bits per base (Ř) is calculated as: LET £ = Total number of bases in the given sequence.
Total number of Encoded Binary Bits = (μ + β + δ + γ+ þ)
Ratio Ř = (μ + β + δ + γ+ þ)/ £ bits/bases.

a) InputDNAString:
aaaaggggggttttttccccccacgtacgttgcaacgttgcacacacagtgtgt
Encoded string:
01100 11010 10111 10101 000011011 100011011 1110 0110100 0101011
Total number of encoded binary bits = 56.0
Total number of Bases = 55.0
Compression Ratio = 1.018 bits/bases.

b) InputString:
GAATTTGCAAAAAAAAGCTAATGCCTAGGGTTTTTGCCCCCCCCC AAAATCAGTTGCATA
GGACG
Encoded String:
10 000 111 1001 11100 100111 000 1110 001 1100 110 10011 10 11101 01100 11010010 011 1001001100 010 000110
Total number of encoded binary bits = 87.0
Total number of Bases = 64.0
Compression Ratio = 1.359375

Methodology of DNABIT COMPRESS:
The length of the DNA sequence is divided in to fragments of four and two. Each fragment (ACGT) is replaced with binary code (0 or 1). Then the total number of bits required to encode the DNA sequence obtained is shown below.
The Total number of bits per base (Ř) is calculated as following:

Ratio Ř = (μ + β + δ + γ+ þ) / £ bits/bases.

Approximately1. 3593 bits per base is required to encode each base if the DNA sequence contains more number of repeated bases.
Let us consider the sequence:
GAAT TTGC AAAA AAAA GCTA ATGC CTAG GGTT TTTG CCCC CCCC AAAA TCAG TTGC ATAG GACG.
SequenceLength = 64.
Bytes to store in a text file = 64 bytes.
Windows XP zip size = 163 bytes.
Biocompress = 14 bytes.
DNABIT Compress algorithm = 10 bytes.

Thus our proposed algorithm DNABIT compress has the following advantages:
i) Compression ratio of 1.5359 bits per base compared to 1.76 bits per base for the other DNA compression algorithms.
ii) Because the method doesn't use dynamic programming technique which was used by other methods e.g., BioCompress, GenCompress etc, it is simple and takes less execution time.

Table 1: The bits assigned to individual bits

BASE	BINARY BITS
A	00
G	01
C	10
T	11

Table 2: 3 BITCODE for Exact 2 Repeat Bases

BASES	3BIT CODE
aa	000
gg	010
tt	011
cc	001

3 BIT CODE for "aa":

0	0	0

Significant Bit, Base value for "a"

3 BIT CODE for "gg":

0	1	0

3 BIT CODe for "tt":

0	1	1

3 BIT CODE for "cc":

0	0	1

Table 3: 3BIT CODE for Exact three repeat bases

BASES	3 BIT CODE
Aaa	100
Ggg	110
Ttt	111
Ccc	101

First 3 bits represents code for number "4" (repeats). [4 = "011"]

3 BIT CODE for "aaa":

1	0	0

Significant Bit, Base value for aa

3 BIT CODE for "ggg":

1	1	0

3 BIT CODE for "ttt":

1	1	1

3 BIT CODe for "ccc":

1	0	1

Table 4: 5 BITCODE for 4 repeats

BASES	5BIT CODE
aaaa	01100
gggg	01110
Tttt	01111
Cccc	01101

First 3 bits represents code for number (repeat) 5. [5 ="100".]

5BITCODE for "aaaa":

0	1	1	0	0

5BIT CODE for "gggg":

0	1	1	1	0

5BIT CODE FOR "tttt":

0	1	1	1	1

5BIT CODE FOR “cccc”:

0	1	1	0	1

Table 5: 5 BITCODE for 5 repeats

BASES	5BIT CODE
Aaaaa	10000
Ggggg	10010
Tttt	10011
Ccccc	10001

5BITCODE for “aaaaa”:

1	0	0	0	0

5BITCODE for“ggggg”:

1	0	0	1	0

5BITCODE for “tttt”:

1	0	0	1	1

5BITCODE for “ccccc”:

1	0	0	0	1

Table 6: 5 BITCODE for 6 repeats

BASES	5BIT CODE
aaaaaa	10100
gggggg	10110
Ttttt	10111
cccccc	10101

First 3 bits represents code for “6”. (repeats) [6 = 101].
5BITCODE for “aaaaaa”:

1	0	1	0	0

5BITCODE for “gggggg”:

1	0	1	1	0

5BITCODE for “tttttt”:

1	0	1	1	1

5BITCODE for “cccccc”:

1	0	1	0	1

Table 7: 5 BITCODE for 7 repeats

BASES	5BIT CODE
aaaaaaa	11000
ggggggg	11010
ttttttt	11011
ccccccc	11001

First 3 bits represents code for number “7” (repeats)[7= 110]
5BITCODE for “aaaaaaa”:

1	1	0	0	0

5BITCODE for “ggggggg”:

1	1	0	1	0

5BITCODE for “ttttttt”:

1	1	0	1	1

5BITCODE for “ccccccc”:

1	1	0	0	1

Table 8: 5 BITCODE for 8 repeats

BASES	5BIT CODE
aaaaaaaa	11100
gggggggg	11110
tttttttt	11111
cccccccc	11101

First 3 bits represents code for number "8" (repeats) [8=111].
5BITCODE for "aaaaaaaa":

1	1	1	0	0

5BITCODE for "gggggggg":

1	1	1	1	0

5BITCODE for "tttttttt"

1	1	1	1	1

5BITCODE for"cccccccc":

1	1	1	0	1

Table 9: 7 BIT Technique

NUMBER OF REPEATS	3 BIT CODE
2	001
3	010
4	011
5	100
6	101
7	110
8	111

Input String = cacacacacaca

1	0	1	0	1	0	0

Number of Repeats CODE FOR "c" CODE FOR "a"
In the above sequence the first 3 bits represent "6" in binary form. Ca repeat is 6 times. The next 4 bits represent the code for that 2 characters. 101=6, c=01 a=00.

Table 10: Comparison of Compression Ratios for different algorithms (bits/base)

SEQUENCE NAME	SEQ LENGTH	Normal CTW	CTW +LZ	BIOCOM PRESS2	Gen Com Press	DNA Com press	DNA PACK	DNABIT Compress
CHMPXX	121024	1.838	1.669	1.684	1.673	1.671	1.660	1.5170
CHNTXX	155844	1.933	1.612	1.617	1.614	1.612	1.610	1.5843
HEHCMVCG	229354	1.958	1.841	1.848	1.847	1.849	1.834	1.5731
HUMDY STROP	33770	1.920	1.917	1.926	1.922	1.911	1.908	1.5721
HUMHBB	73308	1.892	1.808	1.88	1.820	1.789	1.777	1.606
HUMHDABCD	58864	1.897	1.821	1.877	1.819	1.795	1.739	1.606
HUMHPRTB	56737	1.913	1.843	1.906	1.846	1.816	1.788	1.5744
MPOMTCG	186609	1.962	1.900	1.937	1.905	1.892	1.893	1.5652
VACCG	191737	1.857	1.761	1.761	1.761	1.758	1.758	1.6523
Average		1.907	1.796	1.826	1.800	1.788	1.774	1.583

SAMPLE CODE:

```
/*************************************************************
 /*                ENCP & DECP
/*************************************************************
 import java.awt.*;
 import java.awt.event.*;
 import javax.swing.*;
 import java.io.*;
```

```
import java.util.*;
import java.util.Calendar;
import java.util.Date;
import java.util.Locale;
import java.text.DateFormat;
/**
class window extends JFrame
{
        // Variables declaration

        JLabel labtme;
        JLabel lpassion;
        JLabel lcombo;
        JLabel image;
        JLabel comment;
        JButton Submit;
        JButton Cancel;
        JEditorPane abtme;
        JScrollPane abtme1;
        JEditorPane passion;
        JScrollPane passion1;

        JPanel contentPane;
        String str;
        String user,combo,a="",strin="",a1="";
        float ratio=0,count=0;
        float loop=0;
        int strlen=0;
        String c="00",d="01",e="10",f="11";
        String
   Sindou[]={"000","001","010","011","100","101","110","111"};

// End of variables declaration
        public window()
        {
                super();
                JFrame.setDefaultLookAndFeelDecorated(true);
                JDialog.setDefaultLookAndFeelDecorated(true);
                try
                {
                        UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
*****************************************************************
```

SAMPLE CODE:

```
/*****************************************************
 /*                TANDEM REPEATS
/*****************************************************
class tand
{
        public static void main(String arg[])throws Exception
        {
        String msg="";
        int i,j=0,seq=0,pr=0,re=0,value=0,val=64,value1=0,rec=0,mat=0,t=0;
        int v=0,c1=0,c2=0,c3=0,c4=0,mat1=0,mat2=0,pr1=0,ss=0;
        int tan1=0,tan2=0,tan3=0;
        float loop=0;
        String temp="",comp="",che1="",che2="";
        char r,c;
        /*************************************************
        DataInputStream dis=new DataInputStream(System.in);
        String text=dis.readLine();
        loop=text.length();
        for(i=0;i<loop;i++)
        {
                if(i<loop-1)
                {
```

```
                c1=0;
                che1="";
                che2="";
        /************************************************
                if(i<=loop-8&&pr==0)
                {
                        comp=text.substring(i,i+4);
                        temp=text.substring(i+4,i+8);
                        c1=0;
                        mat1=0;
                        mat2=0;
                /**********************comparison*****************
                while(temp.equals(comp))
                {

                        che1=comp.substring(0,2);
                        che2=comp.substring(2,4);
                        temp=" ";
                        if(comp.charAt(0)==comp.charAt(1))
                        {
                        mat1=1;
                        }

                        if(che1.equals(che2))
                        {
                        mat2=1;
                        }
*****************************************************
```