ALGORITHM PROJECT REPORT

Knapsack cryptosystem

Created By:

Ravi Shankar Bharti 1200128

Vikram Patel 1200123

Guided by:

Dr. Kapil Ahuja

Algorithm Objective

To encrypt a given message or string using Huffman tree and knapsack problem.

Input : Message that you want to send

Output : Same message

Intermediate stages : Huffman encryption (output Huffman encoded string)

Knapsack generation (public and private knapsack)

Knapsack encryption (output cipher text)

Knapsack decryption (output decipher text, and encoded binary string)

Huffman decryption (output original message)

Terms and Definitions

Knapsack Problem: Given a knapsack with maximum weight capacity and a set of items with their weights and values, Is there a set of items which completely fills the knapsack considering the weight constraint.

Superincreasing sequence: A sequence of positive real number is called **superincreasing** sequence, if every element of the sequence is greater than the sum of all previous elements in the sequence.

Superincreasing Knapsack: A superincreasing knapsack is a knapsack in which the weights of the n items are in superincreasing sequence.

Modular Inverse: A modular inverse of an integer b (modulo m) is the integer b^{-1} such that $b*b^{-1} \bmod m = 1$.

Huffman Algorithm:

Encryption: Given a string of characters, encode the string into binary form using minimum no. of bits, with the help of a Huffman tree.

Decryption: Using the encoded string and the Huffman tree decode the binary string to original format.

Algorithm Design

The algorithm progresses as follows:

1. Creating Huffman tree: This step is necessary as to encrypt the string into a binary string taking minimum no. of bits Huffman tree is required. In this step , we count the frequency of each and every character appearing in the message, then build a Huffman tree.

   Pseudo code for creating Huffman tree:

   ```
   Create list F from singleton trees formed from elements of W
   WHILE (F has more than one element) DO
   Find $T_1$, $T_2$ in F that have minimum values associated with their root
   Construct new tree T by creating a new node and setting T1 and T2 as its children
    Let the sum of the values associated with the roots of T1 and T2 be associated with the root of T
   Add T to F

   Huffman tree : tree stored in F
   ```

   Implementation of this functionality in code : init() , qinsert(), qremove()

2. Huffman encoding: create binary code for each character present in the input string using Huffman tree by giving ‘0’ to left and ‘1’ to right branch while walking the tree from root node to leaf node. Use the generated code to convert the input string to binary string.
   Implementation of this functionality in code : build_code() ,encode()

3. Knapsack Generation: The encoded string is too long, we can’t create a superincreasing knapsack of that much length, so we need to break the binary string into different parts of length n (which is a small integer) . Create a superincreasing knapsack(easy knapsack) of length n.

Generate two integers ‘p’ and ‘r’ such that p is greater than sum of all the elements of superincreasing sequence and it is prime, r is a positive integer less than p.

Hard knapsack H can be generated by using easy knapsack E and ‘p’ and ‘r’.
$H_i = (k_i * r) \bmod p$.

Hard knapsack is used as a public key, and private key is the easy knapsack combined with ‘p’ and ‘r’.

Implementation of this functionality in code : generate_knapsack() , prime()

4. Creating Cipher Text (knapsack Encryption): Cipher text can be created using Hard knapsack and the huffman encoded binary string.

   To encrypt an *n*-bit binary string

   $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$,

   where α_i is the *i*-th bit of the string and $\alpha_i \in \{0, 1\}$, calculate

   $$c = \sum_{i=1}^{n} \alpha_i \beta_i.$$

   c is the cipher text.

   For every partition of the encoded binary string of length n ,we need to have a cipher text.

   Implemented in main()

5. Deciphering the cipher text: To decipher the cipher text , we need modulo inverse M of r (mod p) i.e. M such that r*M mod p = 1. Decipher text can be calculated using cipher text ,M and p .
   To decrypt a cipher text ,calculate
   d = c * M mod p

   d is the decipher text.

For every partition of the encoded binary string of length n ,we will have a decipher text.

Implemented in main() , and using modular_inverse() function

6. Generate binary string from Decipher text: To generate the binary string from Decipher text ,we need to use private Knapsack(easy knapsack) E .

   Calculate b_i's such that decipher text , d = $\Sigma\, b_i * e_i$
   where b_i can be either 0 or 1 , e_i is element of easy knapsack E

   The string $b_0 b_1 b_2 b_3$…… will give back the binary string.

   Implemented in main()

7. Huffman decoding: After getting back the binary string from the decipher text , we will convert it into our original message using Huffman Tree. Start from the root node of huffman tree, Read the binary string bit by bit and if it is 0 go left and if it is 1 go right node until you encounter a leaf node , write down the character and go back to the root node. Repeat the same procedure until the binary string ends.
   We will get our original message back after completion of Huffman decoding.
   Implementation of this functionality in code : decode()

Analysis of Algorithm

The knapsack cryptography system is based on the subset sum problem (a special case of the knapsack problem). The problem is as follows: given a set of numbers *A* and a number **b**, find a subset of *A* which sums to b. In general, this problem is known to be NP-complete. However, if the set of numbers (called the knapsack) is superincreasing, meaning that each element of the set is greater than the sum of all the numbers before it, the problem is "easy" and solvable in polynomial time with a simple greedy algorithm.

The efficient algorithm of the feasible form of the problem helps to find such a solution easily.

Knapsack Cryptosystem vs. RSA

- Knapsack Cryptosystem is about 100 times faster than RSA

- (Knapsack Cryptosystem : n ~100, RSA: m~500 bits)
- Knapsack Cryptosystem : n bits are encoded in 2n bits , RSA n bits are encoded in n bits
- Knapsack Cryptosystem's public key of size $2n^2$ ~ 20000 for n~100 and RSA's is 2m ~ 1000 for m~500 bits

Importance

The mathematical principles behind knapsack encryption have been the basis for numerous other encryption methods since, specifically those using a public-private key system (such as RSA). The understanding of the principles behind knapsack encryption help enormously with the understanding of other, more complex and more secure systems.

Complexity

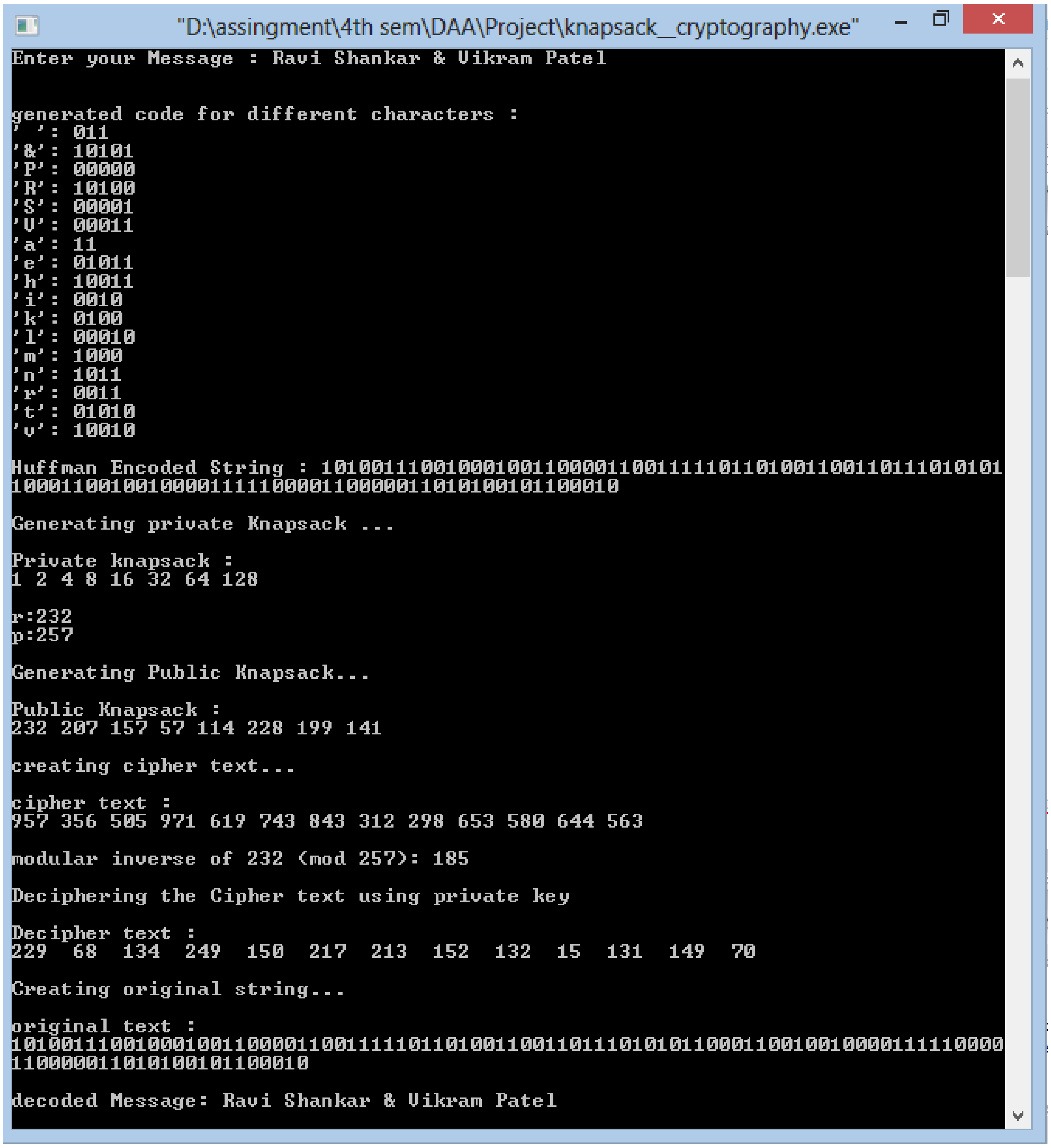
Huffman code takes O(nlogn) time for encoding and decoding both functionality , n is length of input string

Generating the knapsack will take O(klogk) , k is length of knapsack.

Knapsack encryption and decryption takes O(n) time

Total Time Complexity : T(n) = O(nlogn)

Sample Run



```
"D:\assingment\4th sem\DAA\Project\knapsack__cryptography.exe"
Enter your Message : Ravi Shankar & Vikram Patel


generated code for different characters :
' ': 011
'&': 10101
'P': 00000
'R': 10100
'S': 00001
'V': 00011
'a': 11
'e': 01011
'h': 10011
'i': 0010
'k': 0100
'l': 00010
'm': 1000
'n': 1011
'r': 0011
't': 01010
'v': 10010

Huffman Encoded String : 1010011100100010011000011001111101101001100110111010101
1000110010010000111110000110000011010100101100010

Generating private Knapsack ...

Private knapsack :
1 2 4 8 16 32 64 128

r:232
p:257

Generating Public Knapsack...

Public Knapsack :
232 207 157 57 114 228 199 141

creating cipher text...

cipher text :
957 356 505 971 619 743 843 312 298 653 580 644 563

modular inverse of 232 (mod 257): 185

Deciphering the Cipher text using private key

Decipher text :
229  68  134  249  150  217  213  152  132  15  131  149  70

Creating original string...

original text :
10100111001000100110000110011111011010011001101110101011000110010010000111110000
110000011010100101100010

decoded Message: Ravi Shankar & Vikram Patel
```

Reference

www.wikipedia.com

www.csis.bits-pilani.ac.in/faculty/murali/netsec-09/seminar/refs/rajibbrep.pdf