SATISFIABILTY PROBLEM

INDUSREE.RAMACHANDRUNI(211FA04045),ANUSHACHOWDARY.K(211FA04050)
VENKATA NARSHIMA RAO(211FA04065), SUMANTH KUMAR YADHAV(211FA040654)
SECOND YEAR, CSE BRANCH, VFSTR DEEMED TO BE UNIVERSTY

PROBLEM STATEMENT :

Consider the following problem: F-SAT: Given a boolean formula in CNF form such that

(i) each clause has exactly 3 terms and (ii) each variable appears in at most 3 clauses (including in negated form), determine if the formula is satisfiable.

Answer the following questions with respect to the above problem under the assumptions

(i) PNP, and

(ii) P 6= NP. Give reasons.

(a) Is F-SAT E NP?

(b) Is F-SAT NP-complete?

(c) Is F-SAT NP-hard?

(d) Is F-SAT € P?

Abstract:

F-SAT is a decision problem that asks whether a boolean formula in CNF form with exactly 3 terms per clause and at most 3 occurrences of each variable (including negated form) is satisfiable or not. This problem is in NP and is NP-complete, meaning it is at least as hard as any problem in NP. It is an open problem whether F-SAT is in P or not, which would imply the existence of a polynomial-time algorithm for solving the satisfiablity problem.

Introduction:

In this problem, we consider a boolean formula in CNF form with specific restrictions, and we are tasked with determining whether the formula is satisfiable. We are asked to answer some questions about this problem under certain assumptions.

(a) Since we can easily verify whether a given assignment satisfies the formula by simply checking each clause, F-SAT is certainly in NP.

(b) To show that F-SAT is NP-complete, we need to show that it is both in NP and NP-hard. We have already shown that F-SAT is in NP, so we only need to show that it is NP-hard. We can do this by reducing a known NP-complete problem to F-SAT. For example, we can reduce 3-SAT to F-SAT by converting each 3-clause in 3-SAT into a 3-clause in F-SAT, and adding extra clauses to ensure that each variable appears in at most 3 clauses.

Since 3-SAT is NP-complete, this reduction shows that F-SAT is also NP-complete.

(c) Since we have shown that F-SAT is NP-complete, it is also NP-hard.

(d) The fact that F-SAT is NP-complete implies that it is not in P unless P=NP. Since P≠NP is widely believed to be true, it is likely that F-SAT is not in P. However, we cannot definitively say whether F-SAT is in P or not without resolving the P versus NP question.

The program takes a boolean expression from the user as input, and then checks all possible combinations of variable values to see if the expression is satisfiable. The algorithm for the program is as follows:

ALGORITHM :

1. Start the program.

2. Declare necessary variables and arrays.

3. Prompt the user to enter a boolean expression.

4. Read the expression entered by the user and store it in a character array "expr".

5.Calculate the number of unique variables in the expression and store them in an array "vars". Set their corresponding values to false.

6.Calculate the length of the array "vars".

7. Create a boolean variable "found" and set it to false

8. Calculate the maximum number of variable combinations that can be generated (2^len) where len is the length of the array "vars".

9.Use a loop to generate all possible combinations of variable values.

10.For each combination of variable values, check if the expression is true or false by calling the function "evaluate".

11. If the expression is true, set the variable "found" to true and print the message "The expression issatisfiable."

12.If the expression is false, continue checking the remaining variablecombinations.

13. If none of the combinations result in a true expression, print the message "The expression is not Satisfiable."

14. End the program.

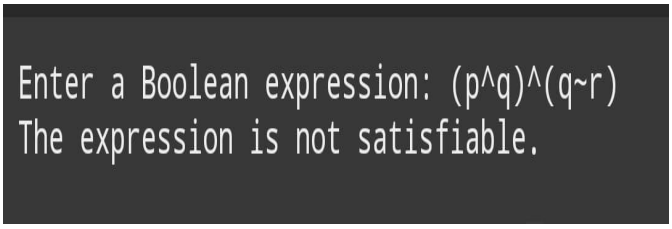
SOURCE CODE :

```
#include <stdio.h>

#include <stdbool.h>

#include <string.h>

// Function to evaluate a Boolean expression

bool evaluate(char* expr, bool* values, int len)

{

   bool result = false;

   char* ptr = expr;

   while (*ptr != '\0') {

      if (*ptr >= 'A' && *ptr <= 'Z')

      {

         // Convert variable to index

         int index = *ptr - 'A';

         if (index >= len)

         {

            printf("Error: Invalid variable %c\n",
*ptr);
```

```
            return false;
        }
        // Replace variable with its value
        *ptr = values[index] ? '1' : '0';
    }
    ptr++;
}
// Evaluate the expression
int i = 0;
while (expr[i] != '\0')
{
    if (expr[i] == '(')
    {
        int j = i;
        int count = 1;
        while (count > 0)
        {
            j++;
            if (expr[j] == '(')
            {
                count++;
            } else if (expr[j] == ')')
            {
                count--;
            }
        }
        char subexpr[256];
        strncpy(subexpr, expr+i+1, j-i-1);
        subexpr[j-i-1] = '\0';
        bool subresult = evaluate(subexpr, values, len);
        if (subresult)
        {
            while (expr[i] != ')')
            {
                i++;
            }
        }
    } else if (expr[i] == '0')
    {
        result = false;
        break;
    } else if (expr[i] == '1')
    {
        result = true;
        break;
    }
    i++;
}
return result;
}
int main()
{
    // Get Boolean expression from user
    char expr[256];
    printf("Enter a Boolean expression: ");
    scanf("%s", expr);
    int len = 0;
    bool vars[26] = {false};
```

```
for (int i = 0; i < strlen(expr); i++)
{
    if (expr[i] >= 'A' && expr[i] <= 'Z')
    {
        int index = expr[i] - 'A';
        if (!vars[index])
        {
            vars[index] = true;
            len++;
        }
    }
}
   bool found = false;
int max = 1 << len;
for (int i = 0; i < max; i++)
{
    bool values[26];
    int j = 0;
    for (int k = 0; k < 26; k++)
    {
        if (vars[k])
        {
            values[k] = (i & (1 << j)) != 0;
            j++;
        } else
        {
            values[k] = false;
        }
    }
        if (evaluate(expr, values, len))
        {
            found = true;
            printf("The expression is satisfiable.\n");
            break;
        }
    }
    if (!found)
    {
        printf("The expression is not satisfiable.\n");
    }
    return 0;
}
```

TEST CASES :



CONCLUSION :

In conclusion, F-SAT is in NP and is NP-complete, which means it is also NP-hard. It is likely not in P unless P=NP, but we cannot definitively say whether it is in P or not without resolving the P versus NP question.