Lab 10

Name: Yuvraj Singh
Roll No: 23072021
M.Tech CSE

Problem: Write a program that takes an $m \times n$ matrix A and an $m \times 1$ matrix B over Z_p and outputs whether B belongs to the range space of A along with two solutions for the systems of equations $AX = B$ (if this system has more than one solution), the prime p, m, n, A, B are to be taken as input.

Solution:

Range Space of a Matrix: The range space (or column space) of a matrix A consists of all possible linear combinations of its column vectors. It represents the subspace of the vector space spanned by the column vectors of A.
To check if the vector B belongs to the range space of matrix A, we perform Gaussian elimination on the augmented matrix $[A \,|\, B]$ to obtain its RRE form. If all the non-zero rows of the resulting matrix correspond to non-zero elements in B, then B lies in the range space of A.
Finding Solutions (if multiple): If the system of equations $AX = B$ has multiple solutions, we can provide two solutions by setting $n - 2$ free variables to arbitrary values and solving for the remaining variables using back-substitution.
Example:

Implementation in C++

```
#include <iostream>
#include <vector>
using namespace std;

class ModularArithmetic {
private:
    int p;
    int mod(int a, int b) {
        int result = a % b;
        if (result < 0) result += b;
        return result;
    }

    int modExp(int base, int exponent) {
        long long result = 1;
        while (exponent > 0) {
            if (exponent & 1)
                result = (result * base) % p;
            exponent >>= 1;
            base = (base * base) % p;
        }
        return static_cast<int>(result);;
    }
```

```
24
25      int modInverse(int a) {
26          return modExp(a, p - 2);
27      }
28
29      int add(int a, int b) {
30          return mod(a + b, p);
31      }
32
33      int subtract(int a, int b) {
34          return mod(a - b, p);
35      }
36
37      int multiply(int a, int b) {
38          return mod(a * b, p);
39      }
40
41      int divide(int a, int b) {
42          return multiply(a, modInverse(b));
43      }
44
45  public:
46      ModularArithmetic(int prime) : p(prime) {}
47
48      void gaussianElimination(vector<vector<int>>& A, vector<int>& B) {
49          int m = A.size();
50          int n = A[0].size();
51          int pivot_col = 0;
52          for (int r = 0; r < m; ++r) {
53              if (pivot_col >= n) break;
54              int i = r;
55              while (A[i][pivot_col] == 0) {
56                  ++i;
57                  if (i == m) {
58                      i = r;
59                      ++pivot_col;
60                      if (pivot_col == n) break;
61                  }
62              }
63              swap(A[i], A[r]);
64              int pivot = A[r][pivot_col];
65              for (int j = 0; j < n; ++j)
66                  A[r][j] = divide(A[r][j], pivot);
67
68              B[r] = divide(B[r], pivot);
69              for (int i = 0; i < m; ++i) {
70                  if (i != r) {
71                      int pivot = A[i][pivot_col];
72                      for (int j = 0; j < n; ++j)
73                          A[i][j] = subtract(A[i][j], multiply(pivot, A[r][j]))
74                      B[i] = subtract(B[i], multiply(pivot, B[r]));
```

```
75                  }
76              }
77              ++pivot_col;
78          }
79      }
80
81      bool isInRangeSpace(const vector<vector<int>>& A, const vector<int>& B) {
82          vector<vector<int>> RRE_A = A;
83          vector<int> RRE_B = B;
84          gaussianElimination(RRE_A, RRE_B);
85          for (int i = 0; i < RRE_A.size(); ++i) {
86              bool allZero = true;
87              for (int j = 0; j < RRE_A[i].size(); ++j) {
88                  if (RRE_A[i][j] != 0) {
89                      allZero = false;
90                      break;
91                  }
92              }
93              if (allZero && RRE_B[i] != 0) return false;
94          }
95          return true;
96      }
97
98      void backSubstitution(const vector<vector<int>>& A, const vector<int>& B)
99          int n = A[0].size();
100         vector<int> solution(n);
101         for (int i = n - 1; i >= 0; --i) {
102             solution[i] = B[i];
103             for (int j = i + 1; j < n; ++j) {
104                 solution[i] = subtract(solution[i], multiply(A[i][j], solutio
105             }
106         }
107         cout << "Solutions␣for␣AX␣=␣B:" << endl;
108         for (int i = 0; i < n; ++i) {
109             cout << "x" << i + 1 << "␣=␣" << solution[i] << endl;
110         }
111     }
112 };
113
114 int main() {
115     int p, m, n;
116     cout << "Enter␣the␣prime␣number␣p:␣";
117     cin >> p;
118     cout << "Enter␣the␣dimensions␣of␣matrix␣A␣(m␣x␣n):␣";
119     cin >> m >> n;
120
121     vector<vector<int>> A(m, vector<int>(n));
122     cout << "Enter␣the␣elements␣of␣matrix␣A:" << endl;
123     for (int i = 0; i < m; ++i)
124         for (int j = 0; j < n; ++j)
125             cin >> A[i][j];
```

```
126
127     vector<int> B(m);
128     cout << "Enter␣the␣elements␣of␣matrix␣B:" << endl;
129     for (int i = 0; i < m; ++i) cin >> B[i];
130
131     ModularArithmetic mod(p);
132
133     // Check if B belongs to the range space of A
134     if (mod.isInRangeSpace(A, B)) {
135         cout <<"Matrix␣B␣belongs␣to␣the␣range␣space␣of␣matrix␣A."<<endl;
136         // Perform back substitution and print solutions
137         mod.backSubstitution(A, B);
138     } else {
139         cout <<"Matrix␣B␣does␣not␣belong␣to␣the␣range␣space␣of␣matrix␣A.\n";
140     }
141     return 0
142 }
```

Time Complexity: $O(m \times n^2)$

Output:

```
Enter the prime number p: 5
Enter the dimensions of matrix A (m x n): 3 3
Enter the elements of matrix A:
2 1 2
1 2 2
3 3 1
Enter the elements of matrix B:
1 1  1

Matrix B belongs to the range space of matrix A.

Solutions for AX = B:
x1 = 0
x2 = 4
x3 = 1
```