Homework Programming Questions using MPI

1. Sparse Matrix multiplication using the row-row format
2. Matrix inversion using the Gauss-Jordan elimination method

Given a non-singular square matrix A of size n×n, compute its inverse using the Gauss-Jordan elimination method. Your implementation should distribute the workload across multiple processes using MPI, where each process handles a subset of rows during the elimination steps.

**1.2 Input Format**

- The first line contains a single integer N (size of the matrix)
- The next N lines each contain N space-separated floating-point numbers (at most 2 decimal places) representing the matrix elements A[i][j]

**1.3 Output Format**

Print N lines of N floating-point numbers each (with exactly 2 decimal places) representing the inverse matrix A□⁻¹[i][j].

**1.4 Constraints**

- 2 ≤ N ≤ 1000
- The matrix is guaranteed to be non-singular (invertible)
- -1000.00 ≤ A[i][j] ≤ 1000.00

**1.5 Example**

3
4.00 7.00 2.00
3.00 6.00 1.00

2.00 5.00 3.00

**Output:**

1.44 -1.22 -0.56
-0.78 0.89 0.22

0.33 -0.67 0.33

**Explanation:** The inverse of matrix A = [[4.00, 7.00, 2.00], [3.00, 6.00, 1.00], [2.00, 5.00, 3.00]] is computed using Gauss-Jordan elimination, resulting in A□⁻¹ as shown above.

3. Find the best way to chain multiply matrices using MPI and parallel dynamic programming

Given a sequence of N matrices: $A_1$, $A_2$....$A_N$, with dimensions $d_0$ x $d_1$, $d_1$ x $d_2$, .....$d_{(n-1)}$ x $d_n$ find the most efficient way to multiply these matrices together using parallelized dynamic programming with MPI. Use DP to determine the optimal parenthesization of the matrix chain that minimizes the total number of scalar multiplications required to compute the product of the N matrices. Distribute the computation of the DP table across multiple processes, where each process handles a subset of the computation.

**Input Format:**

- The first line of the input contains one integer N, representing the number of matrices.
- The second line contains N+1 integers that specify the chain of dimensions. If the matrices are of sizes axb, bxc, cxd... then the integers will be a b c d. The dimensions of the $i^{th}$ matrix ($A_i$) are $d_{i-1}$ x $d_i$. Note that the order of matrices cannot be changed — only the placement of parentheses (i.e., the order of multiplication) can vary.

**Output Format:**

Print one integer representing the minimum number of scalar multiplications needed to multiply the N matrices.

**Constraints:**

- 1 <= N <= 500
- 1 <= $d_i$ <= 1000

**Example:**

**Input:**

4

1 2 3 4 3

**Output:**

30

**Explanation:** The minimum number of scalar multiplications is achieved by inserting parentheses in the following way: (($A_1A_2$ ) $A_3$ ) $A_4$).

Number of multiplications: 1*2*3 + 1*3*4 + 1*4*3 = **30**

4. Find the longest common subsequence across two strings using MPI and parallel dynamic programming.

Given two non-empty strings, compute the longest common subsequence between them using parallel dynamic programming with MPI. Your implementation should compute the longest common subsequence using a parallelized version of the classic dynamic programming algorithm, where each process should handle a subset of the computation for the DP matrix.

**Input Format:**

- The first line contains a single string A (of length n) where 1 <= n <= 10,000.

- The second line contains a single string B (of length m) where 1 <= m <= 10,000.

**Output Format:**

Print one line containing the longest common subsequence between the two strings A and B.

**Example:**

**Input:**

ABCBDAB

BDCABB

**Output:**

BCAB

Homework Programming Questions using Map-Reduce

1. K-Nearest Neighbors

**Goal**: Implement distributed K-NN search using MapReduce framework to find K nearest neighbors for each query point.

**Input**: points.csv where each line is x,y and queries.csv where each line is x,y. Parameter K (neighbors) and file paths provided via command line.

**Requirements**:

- CLI: ./run_knn_mapreduce.sh points.csv queries.csv K output_dir

**Output**: Nearest neighbors in output/results.txt where each line contains K nearest points as (x,y) tuples sorted by distance for each query.

**Evaluation**:

- Correctness of distance computations and neighbor selection; verification that returned points are indeed the K closest. Performance analysis with varying data sizes and cluster configurations.

2. K-Means using Map-Reduce

**Goal**: Implement K-Means using iterative MR jobs.

**Input**: points.csv where each line id,feat1,...,featD and an initial centers.txt with K lines cid,feat1,...,featD.

**Requirements**:

- Implement the standard MapReduce K-Means: mapper assigns points to nearest center; reducer recomputes centers by averaging.
- Provide a driver script that runs iterations until convergence or until T iterations.
- CLI: run_mapreduce.sh K points.csv centers.txt max_iter output_dir

**Output**: final centers in output/centers.txt and a point-to-cluster assignment file.

**Evaluation**:

- Convergence behavior on sample datasets; correctness of the final centers and cluster assignments. Include inertia (sum of squared distances) reported by driver.

3. Find the number of triangles in a given graph. Find the number also against each vertex.

**Goal**: Count the total number of triangles in an undirected graph, and output counts per vertex (how many triangles each vertex participates in) using one or more MR jobs.

**Input**: Edge list file: each line u v with u < v.

**Requirements**:

- Use the standard MapReduce triangle counting pattern: (1) build adjacency lists, (2) emit wedge candidates and check closure. Provide careful handling to avoid double-counting.
- Produce two outputs:
  - global — a single line total_triangles (each triangle counted once).
  - per_vertex — lines v triangle_count_for_v.
- Provide a correctness argument in README showing why your job(s) avoid duplicates.

**Considerations**:

- Graphs may be skewed (high-degree nodes). Include degree-based precautions (e.g., orient edges by degree) and document memory considerations.

**Evaluation**:

- Correctness on small graphs and scalability on larger graphs with skew.

4. Find the number of 4-cycles in a given graph. Find the number also against each vertex.

**Goal**: Count simple 4-cycles (cycles of length 4 with distinct vertices) in an undirected graph; also compute per-vertex participation counts.

**Input**: Edge list file: each line u v with u < v.

**Requirements**:

- Provide an MR algorithm (or small sequence of MR jobs). A common approach: enumerate pairs of length-2 paths (u - x - v) and then detect connections between u and v that close a 4-cycle; take care to avoid overcounting (each 4-cycle may be enumerated multiple times).
- Output both global count and per-vertex participation counts.
- Explain how you deduplicate cycles (canonical ordering, orientation by degrees, or combinatorial division factor) in README.

**Evaluation**:

- Correctness on small graphs; performance/space considerations on larger graphs.

Homework Programming Questions using gRPC

1. The gRPC server runs on a machine such as a datacenter, and the client would like to know the status of the machine via commands such as a list of users, a list of jobs (use streaming RPC), the longest running job, and the like.

   Design a gRPC-based monitoring service.

   a. **Server**: Runs on a datacenter machine, exposing RPCs:
      i. ListUsers() – returns a list of currently logged-in users.
      ii. ListJobs() – returns a stream of currently running jobs (use streaming RPC).
      iii. GetLongestRunningJob() – returns the job with maximum runtime.
   b. **Client**: Can invoke any of the above RPCs.
   c. **Input/Output**:
      i. Input to server is simulated: assume jobs and users are read from a text file.
      ii. Output is returned in text/JSON format.
   d. **Task**: Implement server + client. Test by simulating multiple clients querying status concurrently.

2. The server gets two graphs, one from each client, and answers queries on the combined graph. For instance, does the combined graph have a maximal independent set of size more than a given threshold, does the graph have a matching of size k, and so on. The clients may not be acting in synchrony.

    a. Two clients each hold a graph (represented as adjacency lists). They submit their graphs to the gRPC server, which maintains the **union** of the two graphs. Clients may connect at different times (not synchronized).
    b. **Server Queries**:
        i. HasIndependentSet(k) – returns true if the combined graph has a maximal independent set of size ≥ k.
        ii. HasMatching(k) – returns true if the combined graph has a matching of size ≥ k.
    c. **Input**: Each client provides its graph in adjacency list form.
    d. **Output**: Boolean results to the above queries.
    e. **Task**: Implement graph merging, handle asynchronous arrivals of client graphs, and answer queries.

3. The server gets two matrices, one from each client, and answers queries on the combined matrix. For instance, does the combined matrix have a rank of at least r? Does the matrix have a determinant of d or more? The clients may not be acting in synchrony. Each client may provide one row of the combined matrix.

    a. Two clients each contribute rows of a matrix to the server. The server forms the combined matrix and answers queries.
    b. **Server Queries**:
        i. HasRankAtLeast(r) – check if rank ≥ r.
        ii. HasDeterminantAtLeast(d) – check if determinant ≥ d.
    c. **Input**: Each client provides 1 row at a time. Clients may not be synchronized.
    d. **Output**: Boolean answers to queries.
    e. **Task**: Implement incremental matrix building and query answering.

4. Consider once again two clients and one server. One of the clients feeds 2-dimensional data to the server. The other client makes queries on the data, such as the number of points in a given rectangle, the point with the largest y-coordinate in a given rectangle, etc. The server has to essentially build a range tree for the 2-d point set and answer the client. The client providing the data may change the data every once in a while, and the server builds the tree all over again.

    a. Two clients interact with the server:
    b. **Client A** provides a stream of 2D points (x, y).
    c. **Client B** queries the server with rectangles [x1, x2] × [y1, y2].
    d. **Server Tasks**:
        i. Build a range tree from Client A's points.
        ii. Support queries like:
            - CountPointsInRectangle(rect) – number of points inside.
            - GetTopPoint(rect) – point with maximum y in the rectangle.
    e. **Constraints**: Client A may update points; when this happens, the server rebuilds the range tree. During this time, the server does not send any replies to the queries from client B.
    f. **Output**: Query results returned to Client B.