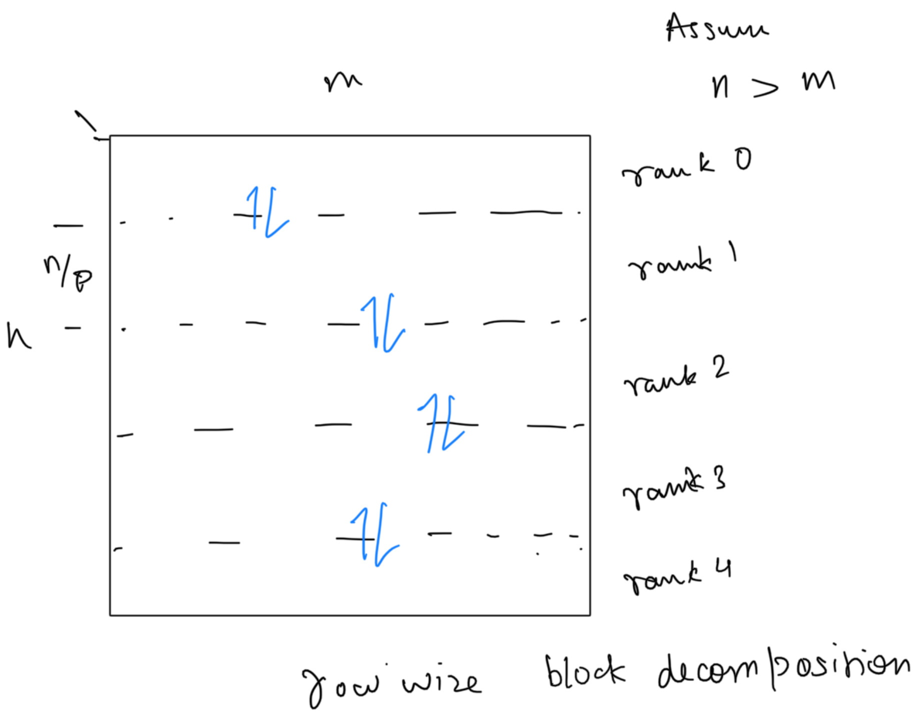
Some Ideas to help you with the Assignment

Question 2

This question is about simulating cellular automata. You will maintain a state for each of the cells. In each iteration you will update the position of each particle based on its current state and the state of the cell it is going towards(checking for collisions).



The way to parallelize this is to decompose the matrix into disjoint batches of rows (or columns) such that each process gets n/p rows (or columns). This is called block decomposition. You should split along the dimension that is larger so as to efficiently use the resources. You should communicate between these blocks in order to process the particles that cross the boundary between these blocks. The way you could communicate is

1. By just sending info about the particles that are moving from one block to another
2. Send the state of all the cells that lie at the boundary of such blocks.

We would accept any one of these solutions.

Here are some test cases:

Input 1

4 6 4 15
0 1 R
2 1 L
3 3 U
1 0 D

Output 1

```
1 3 R
1 3 D
3 2 D
3 3 R
```

Input 2

```
10 12 16 100
0 1 R
2 1 L
3 3 U
9 9 L
6 8 D
4 8 L
7 3 R
5 6 U
0 1 U
2 1 D
3 3 R
9 9 U
6 8 L
4 8 D
7 3 U
5 6 R
```

Output 2

```
0 6 L
0 9 D
2 11 U
3 3 R
3 6 L
3 7 D
4 10 D
5 3 R
5 4 D
6 4 U
7 5 U
7 9 R
7 10 R
8 6 L
9 1 D
9 10 R
```

Question 3

In order to create a BST you would have to sort the keys. The merge sort algorithm is pretty standard, a lot of resources are available on-line. But to help you guys we would allow you to split the vector into P blocks, where P is the number of processes, sort each block independently and then merge them in the rank $= 0$ process.

After you have sorted the pairs of keys and frequency by the keys. Now comes the optimal BST problem, You can have to maintain two arrays cost and root

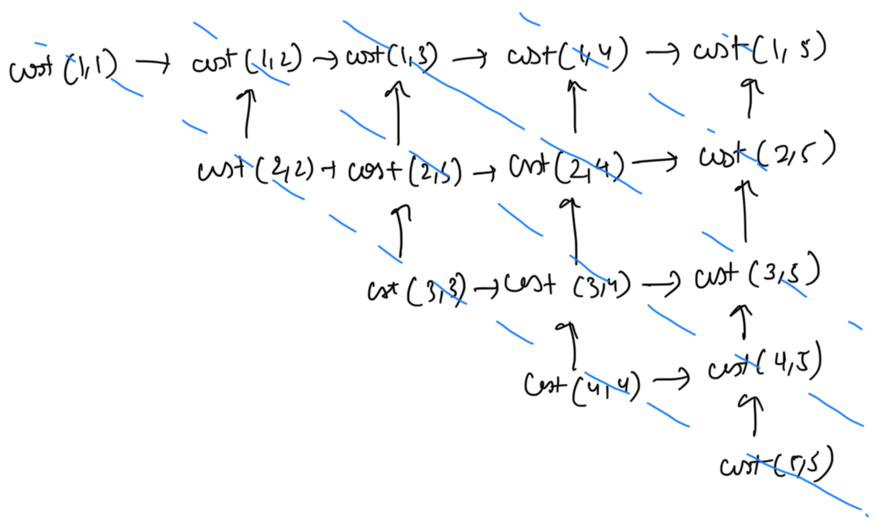
$cost(i, j)$ = best cost of the optimal BST from keys i to j

$root(i, j)$ = The root node for the optimal BST from keys i to j

You construct these in a recursive fashion ussing dynamic programming naively in $O(n^3)$, Going as follows: first finding all the values for $cost(i, i)$ for all i then solving for the values of $cost(i, i + 1)$ for all i and so on till you reach $cost(0, n - 1)$ which the is final answer, in a bottom up fashion. The final dependency is as follows, each element is dependent on the result of all the elements in the row and the column in the diagram below e.g. $cost(1, 4)$ would be computed as a result of $cost(1, 1)$ ... $cost(1, 3)$ and $cost(2, 4)$ ... $cost(4, 4)$

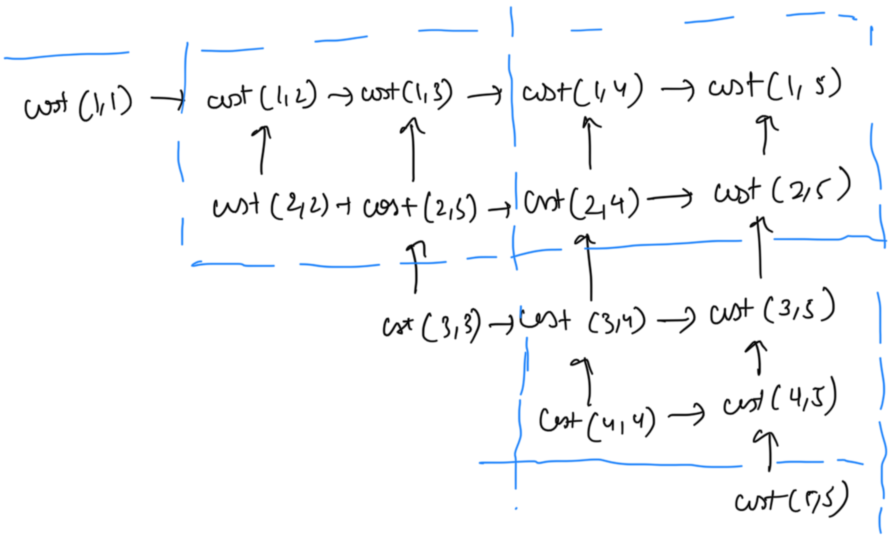
$$\begin{tikzcd}
cost(1,1) \arrow[r] & cost(1,2) \arrow[r] & cost(1,3) \arrow[r] & cost(1,4) \arrow[r] & cost(1,5) \\
 & cost(2,2) \arrow[r] \arrow[u] & cost(2,3) \arrow[r] \arrow[u] & cost(2,4) \arrow[r] \arrow[u] & cost(2,5) \arrow[u] \\
 & & cost(3,3) \arrow[r] \arrow[u] & cost(3,4) \arrow[r] \arrow[u] & cost(3,5) \arrow[u] \\
 & & & cost(4,4) \arrow[r] \arrow[u] & cost(4,5) \arrow[u] \\
 & & & & cost(5,5) \arrow[u]
\end{tikzcd}$$

Now comes the part to parallelize this algorithm, we can see that we can solve each diagonal one after the other. like below



Because of this, we can fill the matrix diagonal by diagonal. The results in the same diagonal can be calculated independently. Then these calculation tasks will be distributed to different processors and use `MPI_Allreduce` for this.

There is also another solution that involves block decomposition:



You divide the upper triangular matrix into blocks equal to the number of processors and each block depends on the results of the blocks in the same row and the column. Again we would allow both diagonal decomposition and block wise decomposition, however it is highly recommended that you try the block decomposition as it reduces the message complexity by a lot.

We would allow the $O(n^3)$ solution but there also exists a $O(n^2)$ solution