**problem2.**

**Parallel Algorithm for Computing the Diameter of a Graph using the CREW PRAM Model**

**Pseudocode:**

function parallelBFS(graph, startVertex):

n = number of vertices in the graph

distances = array of size n initialized with infinity values

distances[startVertex] = 0

for level = 1 to ceil(log2(n)):

newDistances = array of size n initialized with infinity values

parallel for each vertex v:

for each neighbor u of v in graph:

newDistances[u] = min(newDistances[u], distances[v] + 1)

distances = newDistances

maxDistance = 0

for each d in distances:

maxDistance = max(maxDistance, d)

return maxDistance

function computeDiameter(graph):

n = number of vertices in the graph

diameter = 0

parallel for each sourceVertex = 0 to n-1:

sourceDiameter = parallelBFS(graph, sourceVertex)

diameter = max(diameter, sourceDiameter)

return diameter

function main():

n = number of vertices in the graph

m = number of edges in the graph

graph = adjacency list representation of the graph

diameter = computeDiameter(graph)

print "The diameter of the graph is: " + diameter

**Algorithm Description:**

1. Input: graph
2. Initialization: Set diameter = 0.
3. Parallel Iterations:

Parallel for each source vertex in the graph:

* Run a breadth-first search (BFS) starting from the source vertex.
* Compute the maximum distance (eccentricity) from the source vertex to any other vertex.
* Update diameter by taking the maximum of the current diameter and the computed eccentricity.
* Output: final diameter

**Analysis:**

By performing BFS iterations in parallel, the algorithm can utilize O(n^2) processors, leading to a time complexity of O(n log n).

To explain in details:

* The BFS iterations in the algorithm are performed in parallel.
* Each BFS iteration explores a level of the BFS tree, which represents the shortest paths from the source vertex to all other vertices at that level.
* The number of levels in a BFS tree is proportional to the diameter of the graph.
* In each iteration, the algorithm updates the distances from the source vertex to all other vertices at the current level.
* The updates can be performed concurrently using O(n^2) processors, as each vertex can have connections to all other vertices in the worst case.
* Since there are O(log n) iterations (one for each level of the BFS tree), each iteration takes O(n) time due to the parallelism achieved using O(n^2) processors.

Therefore, the total time complexity of the algorithm is O(n log n).