Cpts 415
Assignment 4
Mark Strong-Shinozaki

1. The following questions test your understanding of basic graph algorithms

    a. Given a directed graph $G(V, E, L)$ with V the node set, E the edge set and L a function that assigns to each edge $e \in E$ a label $L(e)$. A label constrained reachability query $Q(s, t, M)$ tests if there exists a path from a source node s to a target node t, which consists of edges having a label from a label set M. Give an algorithm (pseudo-code) to answer query Q.

(hint: A straightforward way is to revise BFS or DFS traversal)

In PSEUDOCODE:

```
function labelConstrainedReachability(G, s, t, M):
        visited = new Set() // Set to keep track of visited nodes
        queue = new Queue() // Queue to store nodes to visit, along with the path
        enqueue(queue, (s, [ ])) // Initialize the queue with the source node
        while not isEmpty(queue):
                current, path = dequeue(queue) // Dequeue the next node and its path
                visited.add(current)
                if current == t: // if we reach the target node, return the path
                        return path
                for each neighbor of current in G:
                        edgeLabel = G.getLabel(current, neighbor)
                        if neighbor not in visited and edgeLabel in M:
                                // Add the neighbor and the updated path to the queue
                                Enqueue(queue, (neighbor, path + [neighbor]))
        Return null // If no path is found, return null
```

- With the PSEUDOCODE above, ‘enqueue’ and ‘dequeue’ are placeholder functions to add and remove elements from a queue, and ‘G.getLabel(current, neighbor)’ is just a function to retrieve the label of the edge between ‘current’ and ‘neighbor’ in the graph.

Cpts 415
Assignment 4
Mark Strong-Shinozaki

b. Consider a network $G(V, E)$ of servers, where each edge $e = (u, v)$ represents a communication channel from a server u to another server v. Each edge has an associated value $r(u, v)$, which is a constant in [0,1]. The value represents the reliability of the channel, i.e., the probability that the channel from server u to server v will not fail. Assume that these probabilities are independent. Give an algorithm (pseudo-code) to find the most reliable path between two given servers. Give a correctness proof and complexity (in Big O notation) of your algorithm.

(hint: Transform the weight to non-negative numbers and the problem may become very familiar to you).

In PSEUDOCODE:

```
function findMostReliablePath(G, s,  t):
        initialize distances to infinity for all nodes
        initialize distances[s] to 0
        create a priority queue Q
        enqueue (s,0) into Q
        while Q is not empty:
            current, current_distance = dequeue(Q)
            if current == t:
                return reconstructPath(s, t, parent) // helper function to reconstruct the path
            if current_distance > distances[current]:
                continue // Skip outdated information
            for each neighbor in G.adjacentNodes(current):
                edge_reliability = G.getReliability(current, neighbor)
                new_distance = current_distance – log(edge_reliability) // Negative logarithm
                if new_distance < distances[neighbor]:                    // transformation
                    distances[neighbor] = new_distance
                    parent[neighbor] = current
                    enqueue (neighbor, new_distance) into Q
        return no path found
function reconstructPath(s, t, parent):
        path = []
        current = t
```

```
while current is not null:

        prepend current to path

        current = parent[current]

return path
```

- Correctness proof:
    - The transformation of taking the negative logs ensure that the maximum product of edge reliabilities becomes equivalent to finding the minimum sum of edge weights, the algorithm will correctly find the most reliable path.
- Complexity:
    - The time complexity of this algorithm is O(E+V log V), where E is the number of edges and V is the amount of vertices in the graph. The extra log(V) factor accounts for the priority queue operations. This is the same time complexity as Dijkstra's algorithm. The space complexity is O(V) to store distances and parent pointers.

2. This question continues our discussion on using data synopsis for query processing on data-driven approximation. You are given a vector of numbers: [127,71,87,59,3,43,99,100,42,0,58,30,88,72,130], each data point records the frequency of communication of server in a 5-minute, 127 contacts are observed. In the next 5 minutes, 71 contacts, …

(hint: Discard the lowest level (high-resolution) coefficient (i.e. only keep the first 50% of coefficients).

A. Give the Haar decomposition and draw a corresponding error tree for the contacts data vector

Level 1 Approximations: [99, 131, 22, 61, 80, 111]
Level 1 Details: [-56, 16, -25, 59, 4, 22]
Level 2 Approximations: [115, 91, 95]
Level 2 Details: [-40, -22, 25]
Level 3 Approximations: [103, 103]
Level 3 Details: [-81, 47]
Level 4 Approximations: [103]
Level 4 Details: [-1]

B. Give the process and result for reconstruction the frequency during time interval [15,20] using Haar decomposition.

406 (Level 1 Approximation)

/ \

-56 16 (Level 1 Details)

/ \ / \

22 61 80 111 (Level 2 Approximations)

/ \ / \ / \

Cpts 415
Assignment 4
Mark Strong-Shinozaki

-25 59 4 22 (Level 2 Details)

/ \ / \

-40 -22 25 (Level 3 Details)

/ \

-81 47 (Level 4 Details)

C. Use Haar decomposition and error tree to compute the total number of communications between time interval [15,30]

Total Communications = 111 (Level 2 Approximations[3]) + 22 (Level 2 Details[2]) + 25 (Level 3 Details[2])

= 111 + 22 + 25

= 158

3. This set of questions test the understanding and application of MapReduce framework.

a. Facebook updates the "common friends" of you and response to hundreds of millions of requests every day. The friendship information is stored as a pair (Person, [List of Friends]) for every user in the social network. Write a MapReduce program to return a dictionary of common friends of the form ((User i, User j), [List of Common Friends of User i and User j]) for all pairs of i and j who are friends. The order if i and j you returned should be the same as the lexicographical order of their names. You need to give the pseudo-code of a main function, and both Map() and Reduce() function. Specify the key/value pair and their semantics (what are they referring to?).

Main-Function PSEUDO-CODE:

```
Function Main():

        Initialize MapReduce job

        Set input data (friendship information as (Person, [List of Friends]))

        Set output data (key: ((User I, User j), value: [List of Common Friends]))

        Set Map function to MapFriends

        Set Reduce function to ReduceCommonFriends

        Run MapReduce job
```

Map Function PSEUDO-CODE:

```
        function MapFriends(Person, Friends):

                For each friend in Friends:

                        If Person < friend:

                                emit ((Person, friend), Friends)

                        else:
```

```
                    emit ((friend, Person), Friends)
```

Reduce Function Pseudo-Code (ReduceCommonFriends):

```
function ReduceCommonFriends(Key, FriendLists):
    common_friends = [ ]
    for each friend_list in FriendLists:
        if common_friends is empty:
            common_friends = friend_list
        else:
            common_friends = intersection(common_friends, friend_list)
    emit(Key, common_friends)
```

b. Top-10 Keywords. Search engine companies like Google maintains hot webpages in a set R for keyword search. Each record $r \in R$ is an article, stored as a sequence of keywords. Write a MapReduce program to report the top 10 most frequent keywords appeared in the webpages in R. Give the pseudo-code of your MR program.

MapReduce program to find the top 10 most frequent keywords in a set of webpages:

```
function Main():
    Initialize MapReduce job
    Set input data (webpages with keywords)
    Set output data (key: keyword, value: frequency)
    Set Map function to MapKeywords
    Set Reduce function to ReduceFrequency
    Run MapReduce job
```

Map Function PSEUDO-CODE (MapKeywords):

```
Function MapKeywords(r):
    keywords = extract_keywords(r)
    For each keyword in keywords:
        emit(keyword, 1)
```

Reduce Function PSEUDO-CODE (MapKeywords):

```
Function ReduceFrequency(keyword, counts):
    total_frequency = sum(counts)
```

```
                    emit (keyword, total_frequency)
```

In both cases, the Main function will initialize the MapReduce job, sets input and output data, specifies the Map and Reduce functions, and then runs the job. The Map functions extract relevant data and emit key-value pairs, while the reduce functions process and aggregate the emitted data.

4. [Graph Parallel Models] (20) This sets of questions relate to MapReduce for graph processing

A. (10) Consider the common friends problem in Problem 3.a. We study a "2-hop common contact problem", where a list should be returned for any pair of friends i and j, such that the list contains all the users that can reach both i and j within 2 hops. Write a MR algorithm to solve the problem and give the pseudo code.

4.A – PSEUDO-CODE for 2-Hop Common Contact Problem (MapReduce):

Main function:

```
function Main():
        Initialize MapReduce job
        Set input data (friendship information as (Person, [List of Friends]))
        Set output data (key: (User i, User j, 2-hop), value: List of 2-hop Common Contacts)
        Set Map function to Map2HopCommonContacts
        Run MapReduce job
```

Map Function Pseudo-code (Map2HopCommonContacts):

```
Function Map2HopCommonContacts(Person, Friends):
        For each friend in Friends:
                If Person < friend:
                        emit((Person, friend, 2-hop), Friends)
                else:
                        emit ((friend, Person, 2-hop), Friends)
```

B. (10) We described how to compute distances with mapReduce. Consider a class of d-bounded reachability queries as follows. Given a graph G, two nodes u and v and an integer d, it returns a Boolean answer YES, if the two nodes can be connected by a path of length no greater than d. Otherwise, it returns NO. Write an MR program to compute the query $Q(G, u, v, d)$ and give the pseudo code. Provide necessary correctness and complexity analysis

```
Map(G, u, d):
```

Cpts 415
Assignment 4
Mark Strong-Shinozaki

```
            For each node v in G:
                If v == u:
                    Emit(u, (u, O))
                else:
                    emit(v, (u, Inf))
Reduce(node, distances):
            min_distance = Inf
            for each (source, distance) in distances:
                if distance < min_distance:
                    min_distance = distance
            if min_distance <= d:
                emit(node, "YES")
            else:
                emit(node, "NO")
Main():
            Input: Graph G, source node u, distance bound d
            Initialize MapReduce job

            MapReduceJob.Map(Map, G, u, d)
            MapReduceJob.Reduce(Reduce)

            Wait for MapReduce job to complete
            Output: Final result, which is either "YES" or "NO" for the query Q(G, u, v, d)
```