Some analysis (e.g. time complexity Big O analysis) is not correct.

Algorithm correctness proof is missing.

1.

(1).

Create a cache, each record of the driven table attempts to match multiple records in the buffer. Then the driven table is scanned and each record of the driven table attempts to match multiple records in the buffer. if the data matches, join the result set. The buffer is larger, the more records the driver table will take out at a time.

In this question, if we have 52 buffers. Relation S is used as the outer relation. Its

So the cost result = (200/ (52-2)) \* 1000 + 200 = 4200

(2).

Sort-merge join is the merging of two or more ordered tables into a new ordered table, that is, the ordered sequence is divided into several sub-sequences, each of which is ordered. Then the ordered subsequence is merged into the whole ordered sequence. Like sort the dataset of table A, and sort the result saved in workspace A; Sort the datasets in table B and save the sorting results in workspace B; Finally, the data in workspace A and B are merged.

Because this problem satisfies this requirement of B(R) + B(S) <=M²

The cost result = 3 (B(R) + B(S)) = 3 \* (1000 + 200) = 3600

(3).

The optimizer uses the smaller table of the two tables, usually the smaller table or data source, and uses the join key to establish the hash table in memory, store the column data in the hash table, and then scan the larger table. Similarly, hash of the join key is conducted and the hash table is detected to find the rows matching the hash table.

The cost result = 3 \* (200 + 1000) = 3600

2

1. for each vertex ∈ G.V

if Lin(s) = ∅ || Lout(t) = ∅；

 output: it doesn’t exist a path from a source s to s target t

bfs (G, s, t):

while Que is nonempty do //

v ← Que.dequeue();

if v = t then return true;

output: have label set M exist a path from s to t.

for all adjacent L(e) = (v, u) of v do

if not flag(u)

then flag(u) ← true; enqueue u onto Que

return false

(2).

//Store the known node in g;

//path = A reachable path consisting of nodes

//F(g): the current value of most reliable path.

if (F(g) < = F(g)’) {

BFS (G, V)

while Que is nonempty do

v ← Que.dequeue();

if v = t then return true;

g← this node //Store the known node in g

for all adjacent edges e = (v, u) of v do

if not flag(u)

then flag(u) ←true; enqueue u onto Que;

return false

F(g)’= The sum of the value of this path node

}

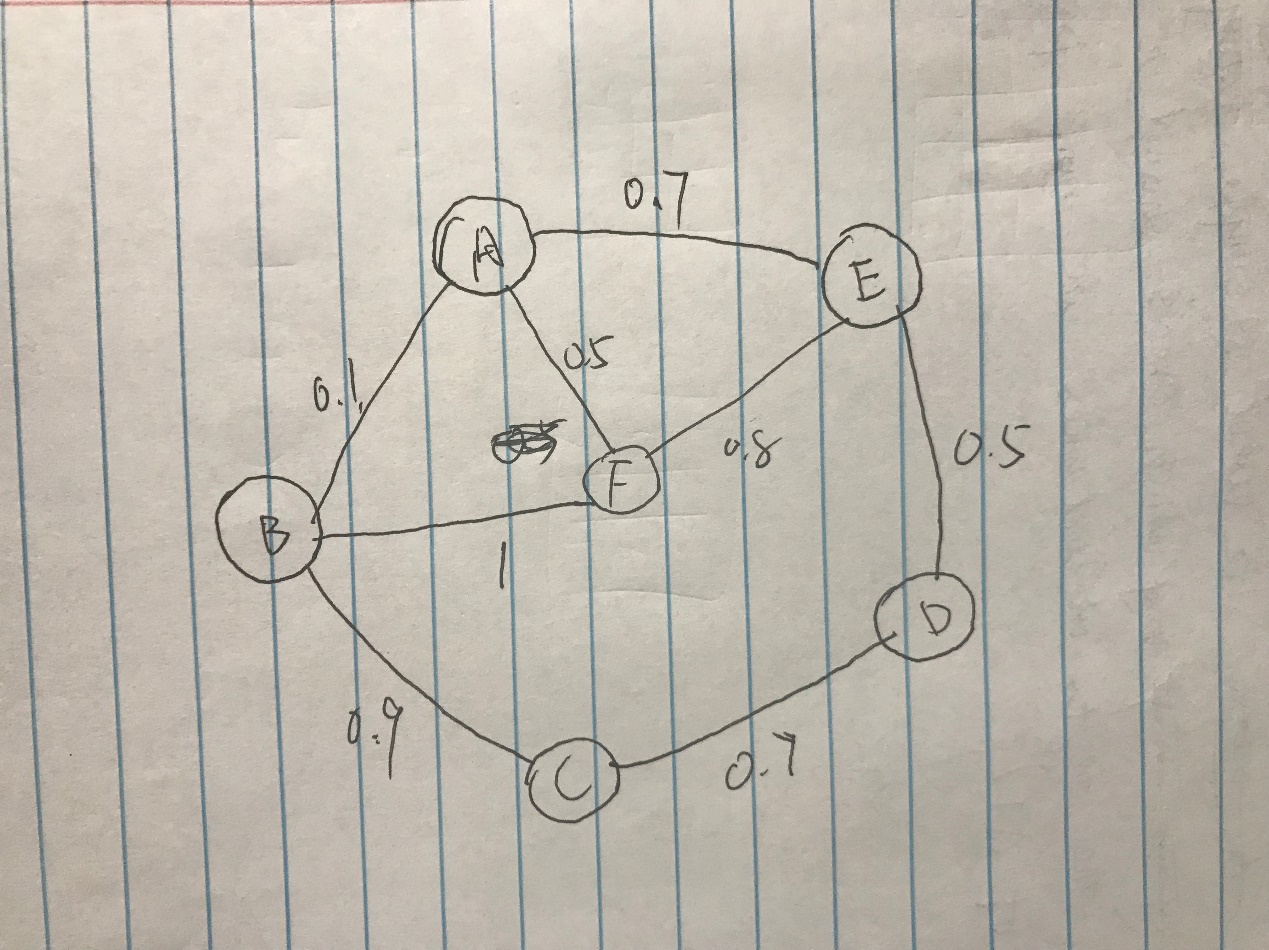
else output F(g), and this path is the most reliable path

compute complexity of my algorithm

in this pseudo-code, If you have n nodes, if-else statement execute n - 1 times.

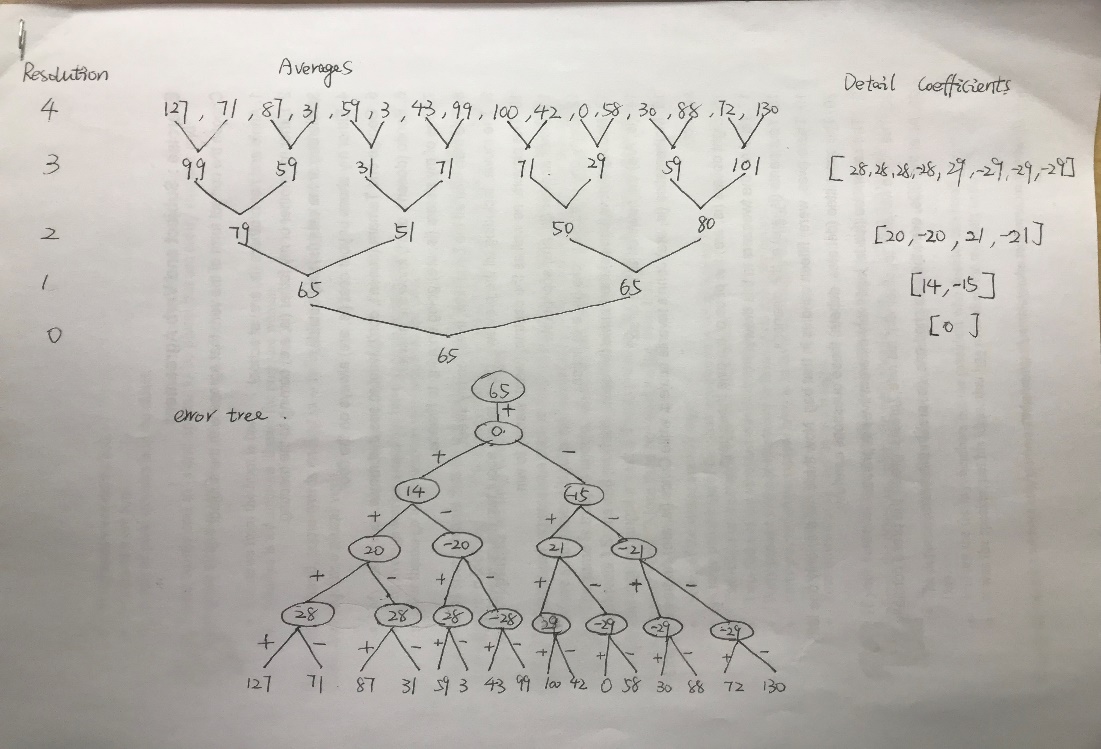
The sentence” else output F(g), and this path is the most reliable path” execute 1 times

So the time complexity is O(n).



3.

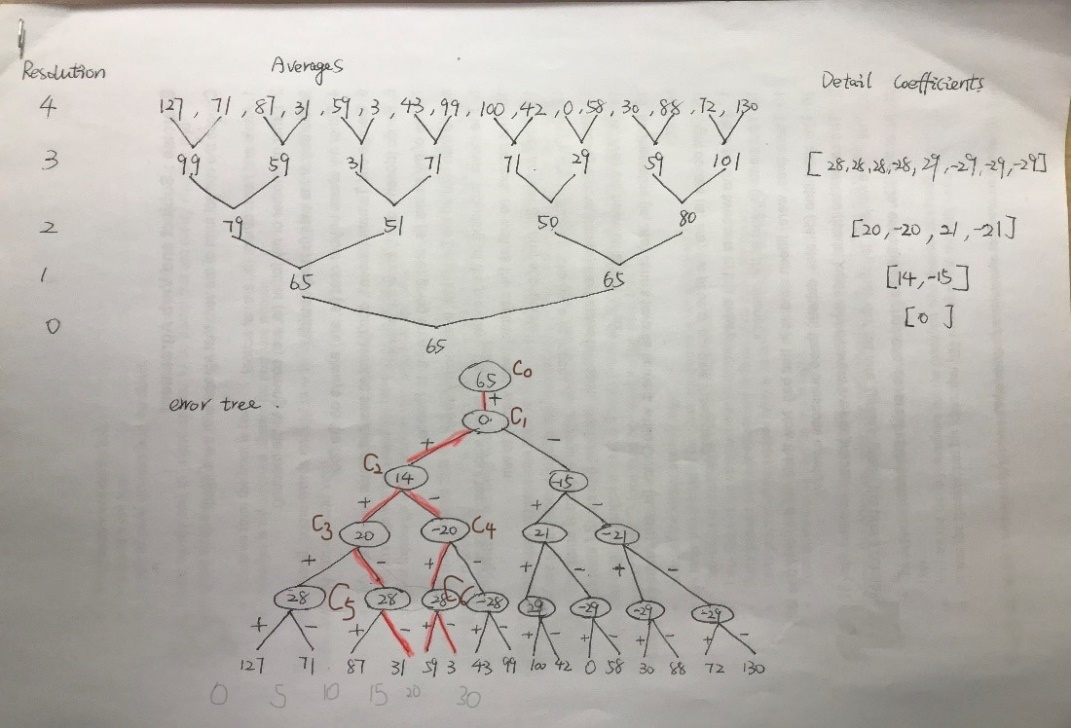
(1).



（2）.

Interval [15,20] value = -28 – 20 + 14 + 0 + 65 = 31

（3）.



According this picture,

We can know h = 3, l = 5

A (3:5) = 3\*C0 + 3\*C1 + (-1)\*C2 + (-1)\*C3 + 2\*C4 + (-)1\*C5 = 3\*65 + 0 - 14 – 20 – 40 - 28 = 93

So the total number of communication between time interval [15, 30] is 93.