                 1

             2        3

         4     5

     (1 / 2)

t = 6

directed tree with n vertices.

time > number of vertices.

u = 2  P[2] 1/2

P[v] = P[u] \* 1/#outgoing edges

P[4] = 1/4

P[5] = 1/4

P[3] = 1/2

vector<double> getProbability(vector <vector<int>> &graph, int root) {

int n = graph.size();

vector<double> prob(n, 0);

queue<int> q;

q.push(root);

vis[root] = true;

prob[root] = 1;

while(!q.empty()){

  int u = q.front(); q.pop();

  int out\_sz = graph[u].size();

  for(int v : graph[u]){

    prob[v] = (double)prob[u]\* (1/out\_size);

    q.push(v);

  }

  if(out\_sz > 0 ) prob[u] = 0;

}

return prob;

}

T = O(V+E) = (n+n-1) = O(n)

                     1

              2             3

                     4

  5

indegree   [0, 1, 1, 2, 1] => (V+E)

updateCount[0, 1 ,1, 1, 0] =>

Q = (V+E)

Q = []

vector<double> getProbability(vector <vector<int>> &graph, int root) {

int n = graph.size();

vector<double> prob(n, 0);

queue<int> q;

q.push(root);

vis[root] = true;

prob[root] = 1;

vector<int> indegree(n, 0);

vector<int> updates(n,0);

for(int u = 0; u < n; u++){

  for(int v : graph[u]){

    indegree[v]++;

  }

}

while(!q.empty()){

  int u = q.front(); q.pop();

  int out\_sz = graph[u].size();

  for(int v : graph[u]){

    prob[v] += (double)prob[u]\* (1/out\_size);

    updates[v]++;

    if(indegree[v] == updates[v])q.push(v);

  }

  if(out\_sz > 0 ) prob[u] = 0;

}

return prob;

}

T = O(V+E)