Pratyush gupta

Mid term Evaluation simutech

Summary:-

What I understood from the research paper is that plasma chemical reactions have great potential for various applications like treating biomass and working with reactors. However, controlling these reactions under normal pressure is challenging, making it difficult to choose the right reaction path.

Scientists are exploring solutions such as plasma catalysis and special waveforms, but understanding these reactions is complex due to the multitude of reactions and substances involved. It's like trying to navigate a map with many roads to choose from.

They're turning to graph theory to help with this complexity. By representing chemical reactions and species interactions as nodes and edges in a graph, they can visualize and analyze the intricate network of reactions. This helps them understand the connections between different reactions, akin to mapping out important roads on a map.

However, there are limitations to using algorithms like Dijkstra's shortest path in plasma chemical reaction engineering. Plasma reactions are influenced by factors like energy, pressure, and temperature, making them highly dynamic. Dijkstra's algorithm may not accurately depict the entire picture because it focuses only on the fastest pathways, overlooking other important ones. Additionally, it struggles to account for how changing conditions affect reaction rates.

Moreover, Dijkstra's algorithm relies on rate coefficients, which may not fully capture the complexities of reactions and species interactions in dynamic plasma environments. While it's efficient for small to moderately sized graphs, scaling it to large networks with hundreds or thousands of nodes and edges can pose computational challenges.

Limitations:-

1. Oversimplification: Dijkstra's algorithm finds the shortest path based solely on reaction rates, which may oversimplify the complex dynamics of plasma reactions influenced by various factors such as energy, pressure, and temperature.
2. Focus on Fastest Pathways: The algorithm prioritizes the fastest pathways between species, potentially overlooking slower but equally significant reaction pathways. This narrow focus may not fully capture the complexity of plasma chemical reaction networks.
3. Inadequate Consideration of Changing Conditions: Plasma reactions are highly dynamic, and conditions such as temperature and pressure can significantly influence reaction rates. Dijkstra's algorithm may not adequately account for these changing conditions, leading to potentially misleading results.
4. Limited Representation of Reaction Kinetics: The algorithm relies on rate coefficients to quantify reaction kinetics, which may not fully represent the intricate interactions and behaviors of different reactions and species in dynamic plasma environments.
5. Computational Challenges with Large Networks: While efficient for small to moderately sized graphs, Dijkstra's algorithm may encounter computational challenges when applied to large-scale networks with hundreds or thousands of nodes and edges, limiting its scalability in analyzing complex plasma chemical reaction systems

Solution:-

**1.)Stone Game**

class Solution {

int n ;

vector<int>dp ;

int STONE(vector<int>& stoneValue, int cur)

{

if (cur == n)

{

return 0;

}

if (dp[cur] > INT\_MIN)

{

return dp[cur];

}

int sum = 0 ;

for(int i = cur; (i < cur + 3) && (i < n ); i++)

{

sum += stoneValue[i] ;

dp[cur] = max(dp[cur], sum - STONE(stoneValue, i+1)) ;

}

return dp[cur] ;

}

public:

string stoneGameIII(vector<int>& stoneValue)

{

n = (int)stoneValue.size() ;

dp.resize(n, INT\_MIN) ;

int score = STONE(stoneValue, 0 ) ;

if (score > 0)

{

return "Alice";

}

else if (score < 0)

{

return "Bob";

}

else

{

return "Tie";

}

}

};

**2.)Vertical order transversal of binary tree**

class Solution {

public:

vector<vector<int>> verticalTraversal(TreeNode\* root) {

vector<vector<int>> ans;

// we are performing the level order traversal:-

queue<pair<TreeNode\*, pair<int, int>>>q; // Node , {row, col};

q.push({root,{0,0}});

// creating the map that really important:-

map<int, map<int, multiset<int>>>mp; // Col -> {row,{x,y,z...}};

while(!q.empty()){

auto front = q.front();

q.pop();

TreeNode\* &node = front.first;

auto coordinate = front.second;

auto &row = coordinate.first;

auto &col = coordinate.second;

mp[col][row].insert(node -> val);

if(node -> left){

q.push({{node -> left},{row + 1, col - 1}});

}

if(node -> right){

q.push({{node -> right},{row + 1, col + 1}});

}

}

// now traversal:-

for(auto i:mp){

auto &colmp = i.second;

vector<int> verticalLine;

for(auto colmpi : colmp){

auto &mset = colmpi.second;

verticalLine.insert(verticalLine.end(), mset.begin(), mset.end());

}

ans.push\_back(verticalLine);

}

return ans;

}

};

**3.)Maximum Score After Applying Operations on a Tree**

class Solution {

public:

vector<vector<int>> adj;

long long dfs(int node,int parent,vector<int>& values){

if(adj[node].size()==1 && node!=0)

return values[node];

long long sum = 0;

for(auto it : adj[node]){

if(it==parent)

continue;

sum += dfs(it,node,values);

}

return min(sum,1LL\*values[node]);

}

long long maximumScoreAfterOperations(vector<vector<int>>& edges, vector<int>& values) {

int n = values.size();

adj.resize(n);

for(auto it : edges){

adj[it[0]].push\_back(it[1]);

adj[it[1]].push\_back(it[0]);

}

long long ans = 0;

for(int i=0;i<n;i++)

ans += values[i];

long long x = dfs(0,-1,values);

return ans-x;

}

};