**SIMUTECH**

**Submitted By: Priyanshu Maurya**

**A summary of what I learnt from research paper:**

Atmospheric pressure plasmas are becoming more attractive and interesting for chemical engineers as they have some of the great advantages in the field of chemical reactions. These advantages are:

1. They can produce a wide variety of reactive chemical species.
2. They can create high energy conditions without needing expensive high pressure/temperature equipment.
3. The raw materials used for this such as air, water, electricity and biomass are widely available.

Graph theory helps us to visualize the plasma chemistry as a set of nodes which can be any species or a reaction connected by edges which are reactions. We can visualize such a network using mathematical algorithms to identify important pathways for optimized reaction, and observe other patterns.

The goal is that using graph theory will help scientists better understand and control the selectivity and energy efficiency of plasma-based chemical processes.

**Limitations:**

The major challenge is that plasmas at atmospheric pressure lack selectivity. It's hard to control exactly which chemical reactions will happen and which products will be formed.

Researchers are trying to solve this by using things like plasma catalysts and tailored electrical signals. But the chemistry in plasmas is extremely complex, with potentially over 1800 different reactions and 70 chemical species in air alone.

**Methodology for building the graph of a plasma chemical reaction system:**

The starting data is a list of chemical reactions involved in the plasma system, along with the rate coefficients for each reaction. The rate coefficients describe how fast each reaction occurs under different conditions like temperature or electron energy.

* To convert this data into a graph structure, we convert each chemical species or reaction into a node.
* Then the nodes are connected with each other using edges representing a reaction.
* Usually a reaction is either represented as a node or as an edge.
* In previous plasma chemistry graphs, we represent species as nodes and reactions as edges.
* Representing a reaction as an edge, makes it easy to visualize what reaction is happening and what species are involved in it.
* For the specific plasma chemistry data used in this study, the reactions were all one-way, so we used a directed graph.

**Visual representation of the plasma chemical reaction graph:**

The goal of the visualization is to provide an overview of the plasma chemical reaction system. The open-source software Gephi was used to generate the visualization of the directional, weighted graph. The thickness of the edges was used to represent the reaction rate, with faster reactions having thicker edges. Other attributes of the reactions and species, like whether they are influenced by electron energy, were represented by visual properties like edge or node color and size.

Using the filtering functions of Gephi, information relevant to single or small numbers of species of interest can be viewed in increasing levels of complexity.

**Snakes and Ladders**

class Solution {

public:

using pii = pair<int, int>;

int snakesAndLadders(vector<vector<int>>& board) {

int n = board[0].size();

int end = n\*n;

int visited[n\*n+1];

int bd[n\*n+1];

memset(visited, 0, sizeof(int)\*(n\*n+1));

int rev = 0, cnt = 1;

for(int i=n-1; i>=0; i--){

if(rev){

for(int j=n-1; j>=0; j--) bd[cnt++] = board[i][j];

}

else{

for(int j=0; j<n; j++) bd[cnt++] = board[i][j];

}

rev = !rev;

}

pii cur(1, 0);

queue<pii> q;

q.push(cur);

while(!q.empty()){

cur = q.front(); q.pop();

if(cur.first == end) return cur.second;

for(int i=1; i<=6 && cur.first+i<=end; i++){

if(visited[cur.first+i]) continue;

if(bd[cur.first+i]==-1) q.push({cur.first+i, cur.second+1});

else q.push({bd[cur.first+i], cur.second+1});

visited[cur.first+i] = 1;

}

}

return -1;

}

};

**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;

}

};