Computations in Chemical Engineering

1)plasma_djikstars.pdf - Google Drive

Here in this research paper ,I have learned about application of graph theory to find shortest path,store in graph ,graph visualization ,to create network of reactions,also used Djikstra Algorithm with OCARINA in this research paper ,where we can see there is application of to find relative potential when we have lots of datasets ,molecules potential and with kinetics values as weighs,in this application of chemical engineering,with most optimized algorithm we search to reduce complexity,optimization,and to increase efficiency.

Such Simulations with Software which used Djikstra Algorithm ,teach about the shortest path where H has relative low potentila as compare to N2O3 ,but limitations like about dk value ,complexity.

This is example of network delay problem of leetcode where this algorithm used.

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5+10;
const int INF = 1e9+10;

vector<pair<int,int>>g[N];
int dijkstra(int source,int n,vector<pair<int,int>g[N]){
    vector<int>vis(N,0);
    vector<int>dist(N,INF);
     set<pair<int,int>>st;
     st.insert({0,source});
     dist[source]=0;
     while(st.size()>0){
           auto node = *st.begin();
           int v = node.second;
           int v_dist = node.first;
           st.erase(st.begin());
           if(vis[v]) continue;
           vis[v]=1;
           for(auto child:g[v]){
            int child_v = child.first;
            int wt = child.second;
            if(dist[v]+wt<v_dist[child_v]){
                dist[child_v]=dist[v]+wt;
                st.insert({dist[child_v],child_v});

            }

           }
     }
```

```
     int ans = 0;
     for(int i = 1;i<=n;++i){
        if(dist[i]==INF) return -1;
        ans = max(ans,dist[i]);
     }
     return ans;
}
int networkDelayTime(vector<vector<int>>& times, int n, int k){

vector<pair<int,int>>g[N];
for(auto vec:times){
    g[vec[0]].push_back({vec[1],vec[2]});
}
return int dijkstra(int k,int n,vector<pair<int,int>g[N])
}
int main(){
    int n,m;
    for(int i = 0;i<m;++i){
        int x,y ,wt;
        cin>>x>>y>>wt;
        g[x].push_back({y,wt});

    }
}
```

Solutions:-

```
class Solution {
public:
    map<int,pair<int,int>> solve(int size){
       map<int,pair<int,int>> ss;
       int starts=1;
       int e=1,o=0;
```

```
        for(int po=size-1;po>=0;po--){
            if(e==1){
for(int p=0;p<size;p++){
    ss[starts]={po,p};
    starts++;
}e=0,o=1;
            }
            else{
for(int p=size-1;p>=0;p--){
    ss[starts]={po,p};
    starts++;
}
o=0,e=1;
            }
        }
        return ss;
    }
    int snakesAndLadders(vector<vector<int>>& board) {
       queue<pair<int,int>> pose;
    map<int,int> mappss;
        map<int,pair<int,int>> mapp=solve(board.size());
        pose.push({1,0});
       while(pose.size()){
          pair<int,int> pp=pose.front();
          pose.pop();
          if(pp.first==board.size()*board.size())
          return pp.second;
       for(int po=1;po<=6;po++){
           if(mappss.count(po+pp.first)==0&&po+pp.first<=(board.size()*board.size(
))){
               int ss=mapp[po+pp.first].first,p=mapp[po+pp.first].second;
               if(board[ss][p]>=1){
                cout<<board[ss][p];
                pose.push({board[ss][p],pp.second+1});}
               else
                pose.push({po+pp.first,pp.second+1});
             mappss[pp.first+po]++;
        }
       }
       }
       return -1;
    }
};
```

```
class Solution {
    typedef long long ll;
```

```
    pair<ll, ll> dfs(int u, int p, vector<vector<int>> &adj, vector<int> &vals) {
        ll f = 0, s = vals[u];
        for(auto &v : adj[u]) {
            if(v != p) {
                pair<ll, ll> res = dfs(v, u, adj, vals);
                f += res.first, s += res.second;
            }
        }
        f = min<ll>(f ? f : vals[u], vals[u]);
        return make_pair(f, s);
    }
public:
    ll maximumScoreAfterOperations(vector<vector<int>>& edges, vector<int>& values) {
        int n = values.size();

        vector<vector<int>> adj(n);
        for(auto &e : edges) {
            int u = e[0], v = e[1];
            adj[u].push_back(v);
            adj[v].push_back(u);
        }
        pair<ll, ll> res = dfs(0, -1, adj, values);
        return res.second - res.first;
    }
};
```

```
class Solution {
```

```
public:
   map<int,pair<int,int>> solve(int size){
     map<int,pair<int,int>> ss;
     int starts=1;
     int e=1,o=0;
      for(int po=size-1;po>=0;po--){
         if(e==1){
for(int p=0;p<size;p++){
   ss[starts]={po,p};
   starts++;
}e=0,o=1;
         }
         else{
for(int p=size-1;p>=0;p--){
   ss[starts]={po,p};
   starts++;
}
o=0,e=1;
         }
       }
       return ss;
   }
   int snakesAndLadders(vector<vector<int>>& board) {
     queue<pair<int,int>> pose;
   map<int,int> mappss;
      map<int,pair<int,int>> mapp=solve(board.size());
      pose.push({1,0});
     while(pose.size()){
```

```
        pair<int,int> pp=pose.front();
        pose.pop();
        if(pp.first==board.size()*board.size())
        return pp.second;
      for(int po=1;po<=6;po++){
        if(mappss.count(po+pp.first)==0&&po+pp.first<=(board.size()*board.size())){
           int ss=mapp[po+pp.first].first,p=mapp[po+pp.first].second;
           if(board[ss][p]>=1){
            cout<<board[ss][p];
            pose.push({board[ss][p],pp.second+1});}
           else
            pose.push({po+pp.first,pp.second+1});
          mappss[pp.first+po]++;
       }
      }
      }
      return -1;
    }
};
```