8-Puzzle Problem

```
#include <bits/stdc++.h>
using namespace std;



int dx[] = {1, 0, -1, 0};
int dy[] = {0, 1, 0, -1};

int cal_cost(vector <vector<int>> &initial, vector <vector<int>> &finall){
    int cost = 0;
    for(int i = 0; i < 3; i++)
        for(int j = 0; j < 3; j++){
            if(initial[i][j] != finall[i][j])
                cost++;
        }
    return cost;
}

void print(vector <vector<int>> &node, vector <vector<int>> &finall){
    for(int i = 0; i < 3; i++){
        for(int j = 0; j < 3; j++){
            cout << node[i][j] << " ";
        }
        cout << "\n";
    }
    cout << "cost : " << cal_cost(node, finall) << "\n";
    cout << "\n" << "\n";
}

bool ok(int x, int y){
    if(x < 0 || y < 0 || x >= 3 || y >= 3)
        return false;
    return true;
}

bool find_path(vector <vector<int>> &initial, vector <vector<int>> &finall){

    queue <vector <vector<int>>> q;
    q.push(initial);
    while(true){
        vector <vector<int>> node = q.front();
        q.pop();
        print(node, finall);
        if(node == finall)
            break;
```

```
		int x, y;
		for(int i = 0; i < 3; ++i){
			for(int j = 0; j < 3; ++j){
				if(!node[i][j]){
					x = i, y = j;
				}
			}
		}

		//cout << x << " " << y << "\n";
		int fin_cost = INT_MAX;
		int ind = -1;
		cout << "row" << " " << "coloumn" << " " << "cost" << "\n";
		for(int i = 0; i < 4; ++i){
			int tx = x + dx[i];
			int ty = y + dy[i];
			if(ok(tx, ty)){
				swap(node[tx][ty], node[x][y]);
				int cost = cal_cost(node, finall);
				cout << tx << " " << ty << " " << cost << "\n";
				if(cost < fin_cost){
					ind = i;
					fin_cost = cost;
				}
				swap(node[tx][ty], node[x][y]);
			}
		}
cout << "\n";
		int tx = x + dx[ind];
		int ty = y + dy[ind];
		swap(node[tx][ty], node[x][y]);
		q.push(node);
	}
}

int main()
{

	int N = 3;

	vector <vector<int>> initial(N, vector <int> (N, 0));
	vector <vector<int>> finall(N, vector <int> (N, 0));

	cout << "Enter initial matrix (3x3)" << "\n";
	for(int i = 0; i < N; ++i)
		for(int j = 0; j < N; ++j)
			cin >> initial[i][j];
```

```
    cout << "Enter final matrix (3x3)" << "\n";
    for(int i = 0; i < N; ++i)
        for(int j = 0; j < N; ++j)
            cin >> finall[i][j];

    cout << "\n";

    find_path(initial, finall);


    return 0;
}
```

DFS

```
#include<bits/stdc++.h>
using namespace std;

const int MAX = 1e3 + 7;
std::vector <int> G[MAX];

void dfs(int node, vector <bool> &vis, vector <int> &parent){
    vis[node] = true;
    for(auto u : G[node]){
        if(!vis[u]){
            parent[u] = node;
            dfs(u, vis, parent);
        }
    }
}

void detect_path(int start, int goal, vector <int> &parent){
    int curr = goal;
    vector <int> path;
    while(curr != start){
        path.push_back(curr);
        curr = parent[curr];
        //cout << curr << "\n";
    }
    path.push_back(curr);
    reverse(path.begin(), path.end());
    for(auto u : path){
        cout << u << " ";
    }
}

int main(){
```

```
    cout << "Enter Number Of Nodes and Edges" << "\n";
    int n, m;
    cin >> n >> m;
    cout << "Enter the edges - (u - v)" << "\n";
    for(int i = 0; i < m; ++i){
            int x, y;
            cin >> x >> y;
            G[x].push_back(y);
            G[y].push_back(x);
    }
    cout << "Enter Staring Node And Goal Node" << "\n";
    int start, goal;
    cin >> start >> goal;
    vector <bool> vis(n + 1, 0);
    vector <int> parent(n + 1, 0);
    dfs(start, vis, parent);
    detect_path(start, goal, parent);

          return 0;

}
```

BFS

```
#include<bits/stdc++.h>
using namespace std;

const int MAX = 1e3 + 7;
std::vector <int> G[MAX];

void bfs(int start, vector <bool> &vis, vector <int> &parent){
    queue <int> q;
    q.push(start);
    vis[start] = true;
    while(!q.empty()){
            int node = q.front();
            q.pop();
            for(auto u : G[node]){
                    if(!vis[u]){
                            parent[u] = node;
                            vis[u] = true;
                            q.push(u);
                    }
            }
    }
}
```

```
void detect_path(int start, int goal, vector <int> &parent){
    int curr = goal;
    vector <int> path;
    while(curr != start){
            path.push_back(curr);
            curr = parent[curr];
            //cout << curr << "\n";
    }
    path.push_back(curr);
    reverse(path.begin(), path.end());
    for(auto u : path){
            cout << u << " ";
    }
}

int main(){
    cout << "Enter Number Of Nodes and Edges" << "\n";
    int n, m;
    cin >> n >> m;
    cout << "Enter the edges - (u - v)" << "\n";
    for(int i = 0; i < m; ++i){
            int x, y;
            cin >> x >> y;
            G[x].push_back(y);
            G[y].push_back(x);
    }
    cout << "Enter Staring Node And Goal Node" << "\n";
    int start, goal;
    cin >> start >> goal;
    vector <bool> vis(n + 1, 0);
    vector <int> parent(n + 1, 0);
    bfs(start, vis, parent);
    detect_path(start, goal, parent);

            return 0;

}
```

N-queen Problem

```
#include <bits/stdc++.h>
using namespace std;

bool check(int qn, int c, vector <int> &col){

            //qn = row, c = column number of queen to be checked
            //col[i] = row, i = col of current position
```

```
        for(int i = 0; i < col.size(); i++){
        if(col[i] == -1)
        continue;
        if(abs(qn - col[i]) == abs(c-i))
        return false;
        }
        return true;

}

void nqueens(int n, int qn, vector <int> &col, vector <vector<string>> &ans, vector <string>
&curr){

        if(qn == n){
        ans.push_back(curr);
        return ;
        }

        // check if any column is available or not for queen qn(row is already fixed)
        for(int c = 0; c < n; c++){
        if(col[c] != -1)
        continue;
        else if(check(qn, c, col)){
        col[c] = qn;
        curr[qn][c] = 'Q';
        nqueens(n, qn + 1, col, ans, curr);
        col[c] = -1;
        curr[qn][c] = '.';

        }
        else
        continue;
        }
}

void solveNQueens(int n) {
        vector <vector<string>> ans;
        vector <int> col(n, -1);
        vector <string> curr(n, string(n, '.'));
        nqueens(n, 0, col, ans, curr);
        for(auto u : ans){
        for(auto r : u){
        cout << r << "\n";
        }
        cout << "\n" << "\n";
        }
        return;
}
```

```
int main()
{
	int n;
	cin >> n;
	solveNQueens(n);
	return 0;
}
```

Kruskal Algorithm

```
#include <bits/stdc++.h>
using namespace std;

using namespace std;
const int MAX = 1e4 + 5;
int id[MAX], nodes, edges;
pair <long long, pair<int, int> > p[MAX];

void initialize()
{
	for(int i = 0;i < MAX;++i)
	id[i] = i;
}

int root(int x)
{
	while(id[x] != x)
	{
	id[x] = id[id[x]];
	x = id[x];
	}
	return x;
}

void union1(int x, int y)
{
	int p = root(x);
	int q = root(y);
	id[p] = id[q];
}

long long kruskal(pair<long long, pair<int, int> > p[])
{
	int x, y;
	long long cost, minimumCost = 0;
	for(int i = 0;i < edges;++i)
```

```
        {
        // Selecting edges one by one in increasing order from the beginning
        x = p[i].second.first;
        y = p[i].second.second;
        cost = p[i].first;
        // Check if the selected edge is creating a cycle or not
        if(root(x) != root(y))
        {
        minimumCost += cost;
        union1(x, y);
        }
        }
        return minimumCost;
}

int main()
{
        int x, y;
        long long weight, cost, minimumCost;
        initialize();
        cin >> nodes >> edges;
        for(int i = 0;i < edges;++i)
        {
        cin >> x >> y >> weight;
        p[i] = make_pair(weight, make_pair(x, y));
        }
        // Sort the edges in the ascending order
        sort(p, p + edges);
        minimumCost = kruskal(p);
        cout << minimumCost << endl;
        return 0;
}
```

Dijkstra Algorithm

```
#include <bits/stdc++.h>
using namespace std;

int V;

int minDistance(int dist[], bool sptSet[]){
        int min = INT_MAX, min_index;
        for (int i = 0; i < V; i++)
        if (sptSet[i] == false && dist[i] <= min)
        min = dist[i], min_index = i;
        return min_index;
}
```

```
void printPath(int parent[], int j){
        // Base Case : If j is source
        if (parent[j] == -1)
        return;
        printPath(parent, parent[j]);
        cout << j << " ";
}

void printSolution(int dist[], int n, int parent[]){
        int src = 0;
        cout << "Vertex\t Distance\tPath";
        for (int i = 1; i < V; i++) {
        printf("\n%d -> %d \t\t %d\t\t%d ", src, i, dist[i],
        src);
        printPath(parent, i);
        }
}

void dijkstra(vector <vector<int>> &graph, int src){

        int dist[V];
        bool sptSet[V] = { false };

        // Parent array to store shortest path tree
        int parent[V] = { -1 };

        // Initialize all distances as INFINITE
        for (int i = 0; i < V; i++)
        dist[i] = INT_MAX;

        // Distance of source vertex from itself is always 0
        dist[src] = 0;

        for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet);
        sptSet[u] = true;
        for (int v = 0; v < V; v++)
        if (!sptSet[v] && graph[u][v]
                && dist[u] + graph[u][v] < dist[v]) {
                parent[v] = u;
                dist[v] = dist[u] + graph[u][v];
        }
        }
        // print the constructed distance array
        printSolution(dist, V, parent);
}
```

```
int main()
{
	cout << "Enter the NUmber of vertices : " << " ";
	cin >> V;
	vector <vector<int>> graph(V, vector <int> (V, 0));
	cout << "Enter Adjacency Matrix" << "\n";
	for(int i = 0; i < V; ++i){
	for(int j = 0; j < V; ++j)
	cin >> graph[i][j];
	}
	dijkstra(graph, 0);
	return 0;
}
```

Job Scheduling Problem

```
//job scheduling problem
#include<bits/stdc++.h>
using namespace std;

struct Job{

  char id;
  int dead;
  int profit;

};

bool comparison(Job a, Job b){
	return (a.profit > b.profit);
}

void scheduleJobs(Job arr[], int n){

	sort(arr, arr+n, comparison);

	int result[n];
	bool slot[n];

	for(int i = 0; i < n; i++)
	slot[i] = false;

	int maxProfit = 0;

	for(int i=0; i<n; i++){
	for(int j = min(n, arr[i].dead) - 1; j >= 0; j--){
	if(slot[j] == false){
```

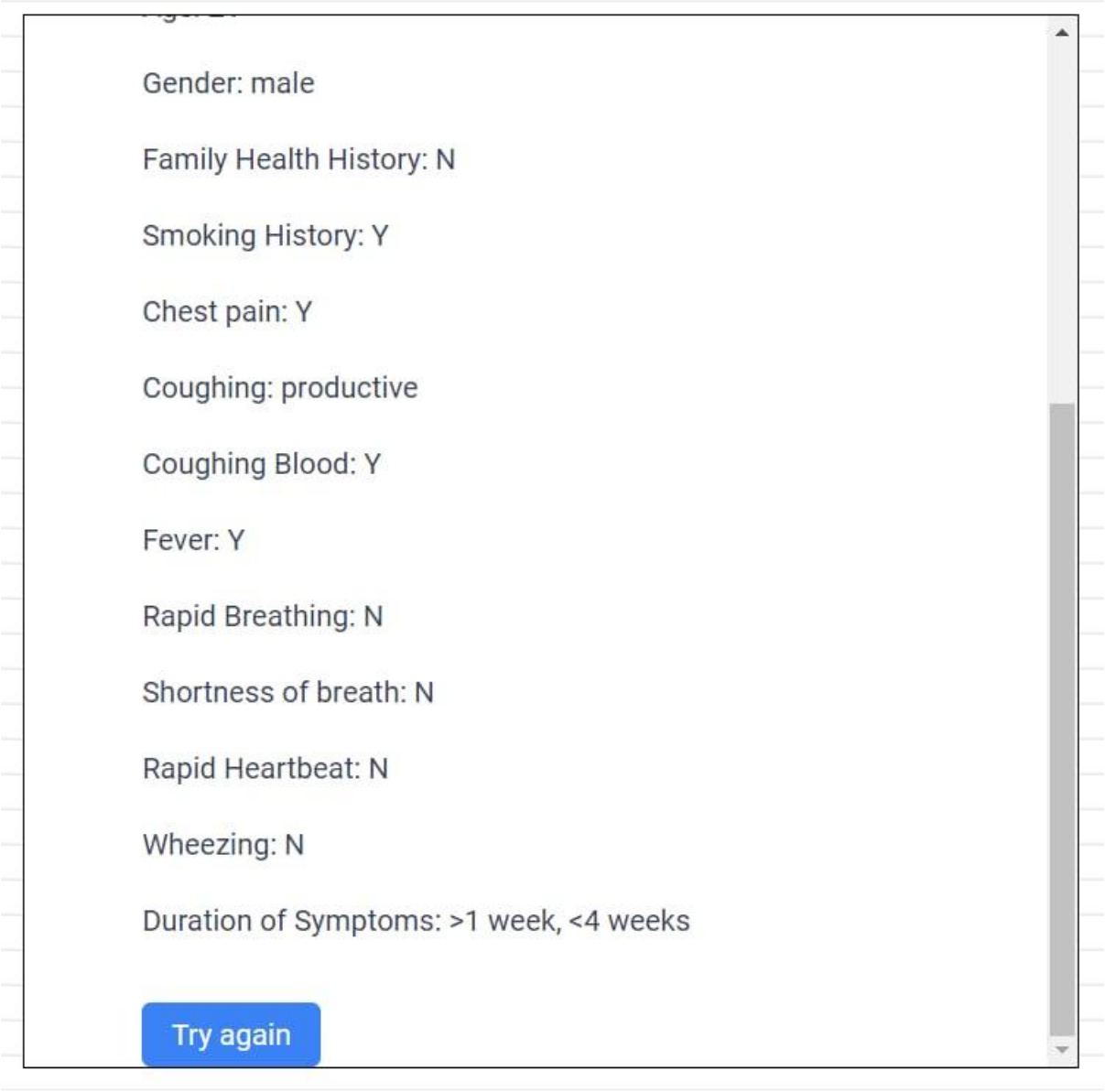
```
                maxProfit += arr[i].profit;
                result[j] = i;
                slot[j] = true;
                break;
        }
        }
        }


        for(int i = 0; i < n; i++)
        if(slot[i])
        cout<<"\t" <<arr[result[i]].id << " ";

        cout <<"\n The maximum profit earned is --> "<< maxProfit <<endl;
}

int main()
{
        cout<< "Enter the number of Jobs : ";
        int n;
        cin >> n;
        Job arr[n];
        cout<< "Enter the Job id, Deadline and Profit for all jobs --> \n";
        for(int i = 0; i < n; i++){
        cin >> arr[i].id;
        cin >> arr[i].dead;
        cin >> arr[i].profit;
        }
        cout << "\n";
        cout << " ---- The Sequence in which the Jobs are completed ----\n";
        scheduleJobs(arr, n);
        return 0;
}
```

Graph Colouring Problem

```
//graph coloring problem
#include<bits/stdc++.h>
using namespace std;

int V, m;

bool isSafe(vector <vector<int>> graph, int color[]){
        for(int i = 0; i < V; i++)
        for(int j = i + 1; j < V; j++)
        if (graph[i][j] && color[j] == color[i])
                return false;
```

```
        return true;
}

void printSolution(int color[]){
        cout << "Colors assigned are as follows \n";
        for (int i = 0; i < V; i++)
        cout << "\t" << i << " vertex is given color " << color[i] << endl;
        cout << "\n";
}

bool graphColoring(vector <vector<int>> &graph, int m, int i, int color[]){
        if (i == V) {
        if (isSafe(graph, color)) {
        printSolution(color);
        return true;
        }
        return false;
        }
        for(int j = 1; j <= m; j++) {
        color[i] = j;
        if(graphColoring(graph, m, i + 1, color))
        return true;
        color[i] = 0;
        }
        return false;
}

signed main()
{

        cout <<"Enter number of vertices : \n";
        cin >> V;
        cout << "Enter the number of colors : \n";
        cin >> m;
        cout << "Enter the Adjacency matrix : \n";
        vector <vector<int>> graph(V, vector<int> (V,0));
        for(int i = 0; i < V; i++){
        for(int j = 0; j < V; j++)
        cin >> graph[i][j];
        }
        int color[V];
        for (int i = 0; i < V; i++)
        color[i] = 0;

        if (!graphColoring(graph, m, 0, color))
        cout << "Solution does not exist!";
        return 0;
}
```

Expert System For Diagnosis Of Respiratory Disease



Amit kumar

you probably have

Rhinosinusitis

Diagnosis confidence: 15.38%

▼ Review your answer

Name: Amit kumar

Age: 21

Gender: male

Family Health History: N

Smoking History: Y

Chest pain: Y

Coughing: productive

Coughing Blood: Y