Rotten Oramges:

```
class Solution {
public:
    int orangesRotting(vector<vector<int>>& grid) {
        int n=grid.size();
        int m=grid[0].size();
         queue<pair<pair<int, int>, int>> q;
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(grid[i][j]==2){
                    q.push({{i,j},0});
                }
            }
        }

        int maxTime=0;
        while(!q.empty()){
             pair<pair<int, int>, int>p=q.front();
            int x=p.first.first;
            int y=p.first.second;
            int time=p.second;
            q.pop();

            maxTime=max(maxTime,time);

            int dx[4]={0,0,-1,1};
            int dy[4]={1,-1,0,0};

            for(int t=0;t<4;t++){
                int newX=x+dx[t];
                int newY=y+dy[t];
                if(newX>=0&&newX<n&&newY>=0&&newY<m&&grid[newX][newY]==1){
                    grid[newX][newY]=2;
                    q.push({{newX,newY},time+1});
                }
            }
        }
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(grid[i][j]!=0&&grid[i][j]==1){
                    return -1;
                }
            }
        }
        return maxTime;
    }
};
```

Minimum Multiplications To Reach End:

```
class Solution {
  public:
    int mod=(int)(1e5);
    int minimumMultiplications(vector<int>& arr, int start, int end) {
        queue<pair<int,int>>q;
        q.push({start,0});
        vector<int>visi(mod,0);
        visi[start]=1;
        while(!q.empty()){
            int value=q.front().first;
            int time=q.front().second;
            q.pop();

            if(value==end){
                return time;
            }

            for(int i=0;i<arr.size();i++){
                int multiply=(value*arr[i])%mod;
                    if(visi[multiply]==0){
                         q.push({multiply,time+1});
                         visi[multiply]=1;
                    }
                }
            }
             return -1;
        }
};
```

DFS of Graph:

```
class Solution {
  public:
    // Function to return a list containing the DFS traversal of the graph.
    vector<int>ans;

    void  dfs(int node,vector<int> adj[],vector<int>&visi){
        visi[node]=1;
        ans.push_back(node);
         for(int x:adj[node]){
          if(!visi[x]){
             dfs(x,adj,visi);
          }
      }
    }
    vector<int> dfsOfGraph(int V, vector<int> adj[]) {
        vector<int>visi(V,0);
        int node=0;
        dfs(node,adj,visi);
        return ans;
    }
};
```