MARINE POLLUTION CLEAN UP USING MODIFIED BANKER'S ALGORITHM

BHARGAV ROY 18BCE0310

SIDHARTH HEMANT NAHAR 18BCE0318

NIMIT KUMAR JAIN 18BCE0328

Submitted to
Prof. Geraldine Bessie Amali, SCOPE

School of Computer Science and Engineering

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Contents

Abstract

- The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

INTRODUCTION

- Marine pollution occurs when harmful effects result from the entry into the ocean of chemicals, particles , industrial , agricultural, and residential waste , noise, or the spread of invasive organisms. Eighty percent of marine pollution comes from land. Air pollution is also a contributing factor by carrying off pesticides or dirt into the ocean. Land and air pollution have proven to be harmful to marine life and its habitats.

- The main causes of ocean pollution are:
- 1. Acidification
- 2. Plastic waste
- 3. Toxins and metallic waste
- 4. Eutrophication

- Bankers algorithm will be applied to allocate the ocean clean up resources as per the needs. Here the resources are floating screens , skimmers , ocean clean up pipeline etc. Which will be allocated to the different ocean areas depending upon the above mentioned pollution levels .
- This will not only make the purification process efficient but will also reduce the overall costs.
- Priority will be decided based on the various surveys and tests conducted ; and this will decide the flow of the resources which each region of the ocean will receive.

HARDWARE AND SOFTWARE REQUIREMENTS

Hardware: nil

Software : g++ compiler , javascript compiler

EXISTING OCEAN CLEANING METHOD

The current method to clean up the ocean is very time consuming and inefficient. The resources which are needed to clean up the various pollutants are just randomly sent to the river deltas according to the money offered by the local government and the availability of the clean up resources. The existing method is does not take any factors such as distance and pollution level at the ocean into account and is just based on finances and availability of resource.

LIMITATIONS OF THE EXISTING METHOD

The current method does not use any efficient algorithm or process in splitting the available resources which leads to high wastage of fuel, inefficient use of clean up devices and also substantially high wastage of money.

PROPOSED MODEL

We are using a modified Banker's algorithm for optimum allocation of resources to clean up the various types of wastes deposited into the oceans. By using this modified Banker's approach we will not only ensure that no deadlock takes place at any point but also we will ensure that all the ocean wastes are cleaned using minimum possible resources ,so that resources are saved for future use and also the cost of transportation is reduced.

DESIGN:

The following is the Modified Banker's Algorithm:
Let **N** be the number of processes in the system and M be the number of resources types.

Available :

- It is a 1-d array of size **M** indicating the number of available resources of each type.

Max :

- It is a 2-d array of size **N*M** that defines the maximum demand of each process in a system.

Allocation :

- It is a 2-d array of size **N*M** that defines the number of resources of each type currently allocated to each process.

Need :

- It is a 2-d array of size **N*M** that indicates the remaining resource need of each process.

Population :

- It is a 1-d array of size **N** that indicates the population surrounding the delta.

Toxicity :

- It is a 1-d array of size **N** that indicates the toxicity of the ocean.
- Calculated as the sum of metallic wastes, plastic wastes and oil/chemical waste.

For its execution :-

- Need [i, j] = Max [i, j] – Allocation [i, j]

$Allocation_i$ specifies the resources currently allocated to process P_i and
$Need_i$ specifies the additional resources that process P_i may still request to complete its task.

Safety Algorithm

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

1) Let Work and Finish be vectors of length 'm' and 'n' respectively.
done[i] = false; for i=1, 2, 3, 4….n

2) Find an i such that both
a) done[i] = false
b) $Need_i$ <= Available
if no such i exists goto step (7)
3) Calculate Priority using the formula :- $Priority=(Population_i * Toxicity_i)/Distance_i$
4) Based on Priority select the River delta where the resources are to be sent immediately.
5) Available = Available + Allocation[i]

6) done[i] = true
goto step (2)

7) if Finish [i] = true for all i
then the system is in a safe state

After this we get an optimized safe sequence. Now we input this safe sequence into the A-star algorithm to find the shortest route to cover all the river deltas.

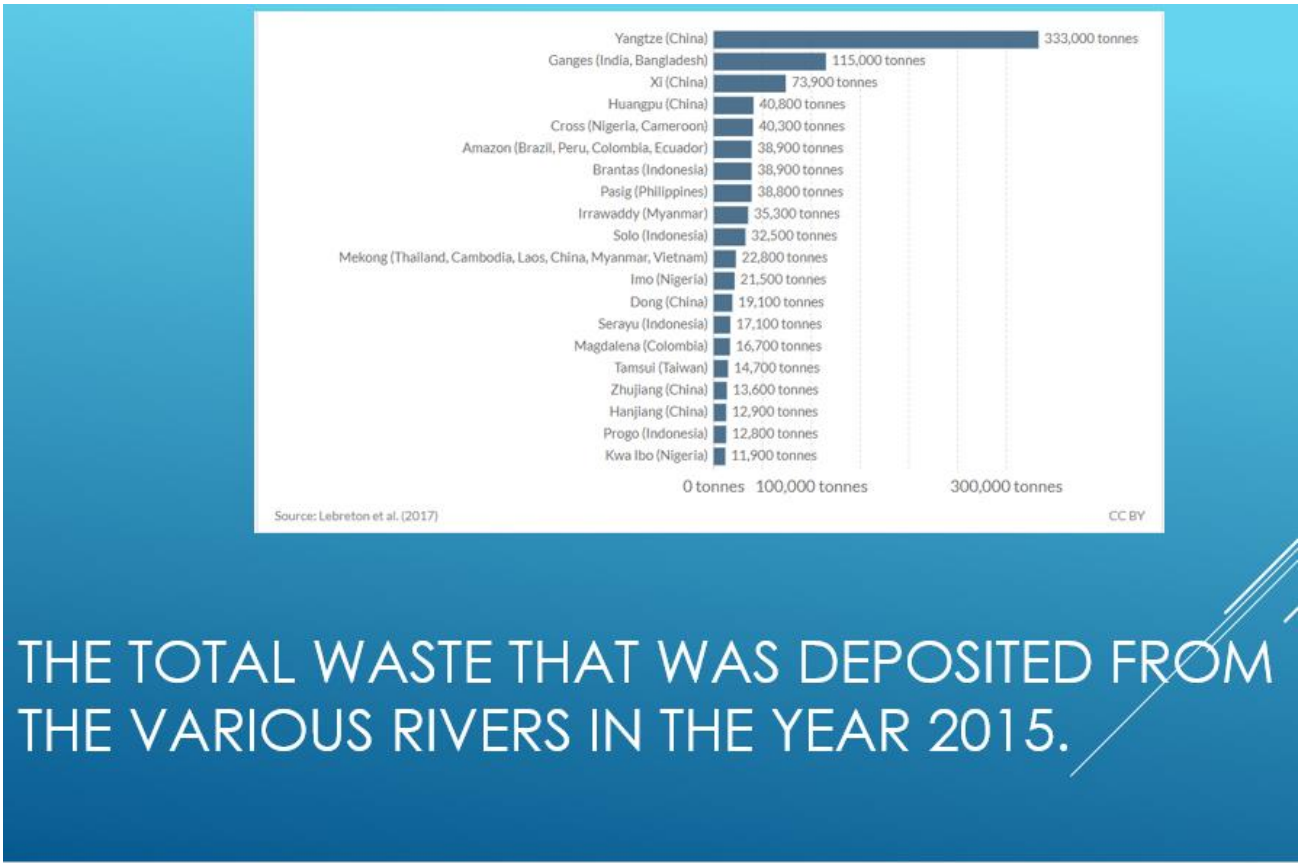
MODULE WISE DESCRIPTION

DATA:

1. Number of rivers deltas.
2. Total number of each type of resources
3. Capacity of collecting waste of each type of resource at a time
4. River ID for each river
5. River name corresponding to each river ID
6. Coordinates corresponding to each river delta
7. Population near the delta of each river.

TYPES OF RESOURCES:

1. BOYAN SLAT OCEAN CLEAN UP (R1)
2. FLOATER (R2)
3. SKIMMER (R3)



PROCESSES	METALLIC WASTE (Tonnes)	PLASTIC WASTE (Tonnes)	OIL AND CHEMICALS (Tonnes)
P1 (YANGTZE)	120500	112000	100500
P2 (GANGA)	47500	44000	23500
P3 (AMAZON)	20100	8450	10350
P4 (MEKONG)	2400	15550	4850
P5 (PROGO)	4000	6800	2000

DISTRIBUTION OF VARIOUS TYPES OF WASTE DUMPED INTO THE DELTAS OF THE RESPECTIVE RIVERS.

Capacity of each resource to clean at one time:

1. **BOYAN SLAT OCEAN CLEAN UP : 50 Tonnes**
2. **Floater : 100 Tonnes**
3. **Skimmer : 80 Tonnes**

PROCESS (RIVER DELTA)	R1 (BOYAN SLAT)	R2 (FLOATER)	R3 (SKIMMER)
P1 (YANGTZE)	2410	1120	1256
P2 (GANGA)	950	440	294
P3 (AMAZON)	402	85	129
P4 (MEKONG)	48	156	61
P5 (PROGO)	80	68	25

MAX RESOURCE REQUIRED BY EACH DELTA FOR CLEAN UP

PROCESSES	R1	R2	R3
P1	1600	550	720
P2	400	220	280
P3	200	50	66
P4	20	104	24
P5	30	36	8

INITIAL ALLOCATION

AVAILABLE
R1=350
R2=440
R3=202

IMPLEMENTATION:

C++ CODE FOR MODIFIED BANKERS:

```
#include<bits/stdc++.h>
#define p_b push_back
#define lli long long int
using namespace std;
int main()
{
	vector<int> rid;
	unordered_map<int,string> mm;
	cout<<"Enter the number of deltas : ";
	int n;
	cin>>n;

	cout<<endl;
	int r1,r2,r3,cr1,cr2,cr3;
	cout<<"Enter the total number of Boyan Slat ocean clean up: ";
	cin>>r1;
	cout<<"Enter the amount of waste that cleaned by Boyan Slat at a time :
";
	cin>>cr1;
	cout<<endl;
	cout<<"Enter the total number of Floaters : ";
	cin>>r2;
	cout<<"Enter the amount of waste that cleaned by Floaters at a time : ";
	cin>>cr2;
	cout<<endl;
	cout<<"Enter the total number of Skimmers : ";
	cin>>r3;
	cout<<"Enter the amount of waste that cleaned by Skimmers at a time :
";
	cin>>cr3;
	cout<<endl;

	vector<double> metal;
	vector<double> plastic;
	vector<double> oil;
	vector<int> alloc_r1;
	vector<int> alloc_r2;
	vector<int> alloc_r3;
	vector<int> max_r1;
	vector<int> max_r2;
	vector<int> max_r3;
	vector<pair<double,double>> coord;
```

```
	vector<lli> population;
	vector<double> toxicity;
	int s1=0,s2=0,s3=0;

	//Taking data for each Delta
	for(int i=0;i<n;i++)
	{
		cout<<"Enter the delta Id of river "<<i+1<<" : ";
		int id;
		cin>>id;
		rid.push_back(id);

		cout<<"Enter the delta name : ";
		string name;
		cin>>name;
		mm[id]=name;

		cout<<"Enter the coordinates(x,y) of the delta : ";
		double x,y;
		cin>>x>>y;
		coord.p_b(make_pair(x,y));

		cout<<"Enter the population near the delta : ";
		lli pop;
		cin>>pop;
		population.p_b(pop);

		cout<<endl;
		double amt;
		cout<<"Enter the amount of metallic and heavy plastic waste in
the delta "<<id<<" : ";
		cin>>amt;
		metal.p_b(amt);
		max_r1.p_b((int)(ceil(amt/cr1)));

		cout<<"Enter the amount of plastic waste in the delta "<<id<<" :
";
		cin>>amt;
		plastic.p_b(amt);
		max_r2.p_b((int)(ceil(amt/cr2)));

		cout<<"Enter the amount of oil and chemical waste in the delta
"<<id<<" : ";
```

```
            cin>>amt;
            oil.p_b(amt);
            max_r3.p_b((int)(ceil(amt/cr3)));

            toxicity.p_b(metal[i]+plastic[i]+oil[i]);

            int nor1=0,nor2=0,nor3=0;
            cout<<"Enter the number of Boyan Slat allocated to delta
"<<id<<" : ";
            cin>>nor1;
            alloc_r1.p_b(nor1);
            s1+=nor1;

            cout<<"Enter the number of Floater allocated to delta "<<id<<" :
";
            cin>>nor2;
            alloc_r2.p_b(nor2);
            s2+=nor2;

            cout<<"Enter the number of Skimmer allocated to river "<<id<<"
: ";
            cin>>nor3;
            alloc_r3.p_b(nor3);
            s3+=nor3;
            cout<<endl;
        }

        //Calculating total toxicity
        double total_tox=0.0;
        for(int i=0;i<n;i++)
        {
            total_tox+=toxicity[i];
        }

        for(int i=0;i<n;i++)
        {
            toxicity[i]=toxicity[i]/total_tox;
        }

        //Calculating available resources
        vector<int> avail(3,0);
        avail[0]=r1-s1;
        avail[1]=r2-s2;
```

```
	avail[2]=r3-s3;

	//For Need Matrix
	vector<int> need_r1,need_r2,need_r3;
	for(int i=0;i<n;i++)
	{
		need_r1.p_b(max_r1[i]-alloc_r1[i]);
		need_r2.p_b(max_r2[i]-alloc_r2[i]);
		need_r3.p_b(max_r3[i]-alloc_r3[i]);
	}


	//Using Banker's Algorithm
	vector<bool> done(n,false);
	vector<int> safe(n);
	int count=0;
	bool flag=true;
	for(int i=0;i<n;i++)
	{
		double priority=0.0;
		int pos=-1;
		for(int j=0;j<n;j++)
		{
			if(need_r1[j]<=avail[0] && need_r2[j]<=avail[1] &&
need_r3[j]<=avail[2] && done[j]==false)
			{
				if(flag)
				{
					safe[count]=j;
					avail[0]=avail[0]+alloc_r1[j];
					avail[1]=avail[1]+alloc_r2[j];
					avail[2]=avail[2]+alloc_r3[j];
					flag=false;
					done[j]=true;
					count++;
					break;
				}
				else
				{
					//Calculating priority
					double p=(population[j]*toxicity[j]);
					int cc=safe[count-1];
					double xd=coord[j].first-coord[cc].first;
```

```
                                double yd=coord[j].second-coord[cc].second;
                                double dist=sqrt(pow(xd,2)+pow(yd,2));
                                p=p/dist;
                                if(priority<p)
                                {
                                        priority=p;
                                        pos=j;
                                }
                        }
                }
        }
        if(pos!=-1)
        {
                done[pos]=true;
                avail[0]=avail[0]+alloc_r1[pos];
                avail[1]=avail[1]+alloc_r2[pos];
                avail[2]=avail[2]+alloc_r3[pos];
                safe[count]=pos;
                count++;
        }
    }

    //Displaying delta information
    cout<<"\nId\tName\tMetal\tPlastic\tOil\tA_r1\tA_r2\tA_r3\tM_r1\tM_r2
\tM_r3\tNeed_r1\tNeed_r2\tNeed_r3\n";
    for(int i=0;i<n;i++)
    {

    cout<<rid[i]<<"\t"<<mm[rid[i]]<<"\t"<<metal[i]<<"\t"<<plastic[i]<<"\t
"<<oil[i]<<"\t";
        cout<<alloc_r1[i]<<"\t"<<alloc_r2[i]<<"\t"<<alloc_r3[i]<<"\t";
        cout<<max_r1[i]<<"\t"<<max_r2[i]<<"\t"<<max_r3[i]<<"\t";
        cout<<need_r1[i]<<"\t"<<need_r2[i]<<"\t"<<need_r3[i]<<endl;
    }

    //Displaying the safe sequence
    cout<<"Safe sequence : ";
    for(int i=0;i<n;i++)
    {
        if(i==n-1)
        {
                cout<<safe[i]+1;
        }
```

```
                else
                {
                        cout<<safe[i]+1<<" --> ";
                }
        }
        return 0;
}
```

JAVASCRIPT CODE FOR SHORTEST PATH:

```
var
items=[["P1","P2","P4","P3","P5","P6","P7","P8","P9","P10","P11"],[5,3,11,18,21,30,45,4
8,55,60,69],[0,5,10,13,16,20,27,3,33,1,40]];

function removeFromArray(arr,elt){
        for(var i=arr.length-1;i>=0;i--){
                if(arr[i]==elt){
                        arr.splice(i,1);
                }
        }
}

function heuristic(a,b){
        var d=dist(a.i,a.j,b.i,b.j);
        return d;
}

var display=[];
var count=0;
var record=[];
var names=[];
var namecount=0;

function add(){
  var a=input.value();
  var f=0;
  for(var i=0;i<11;i++)
  {
    if(items[0][i]==a)
    {
      f=1;
      names.push(a);
      namecount=namecount+1;
    }
  }
  if(f==0)
  {
```

```
    console.log("Invalid! Enter another product.");
  }
  input.value('');
}


function change(){
  t=1;
  count=count+1;
  for(var j=0;j<namecount;j++){
    for(var i=0;i<11;i++){
      if(items[0][i]==names[j]){
                    end=grid[items[1][i]][items[2][i]];
                    end.wall=false;
        record.push(end);
        display.push(items[0][i]);
      }
    }
  }
  loop();
}
var rows=40;
var cols=70;
var grid=new Array(cols);

var openset=[];
var closedset=[];
var start;
var end;
var w,h;
var path =[];
var finalpath=[];
var button;
var t;

function spot(i,j){
          this.i=i;
          this.j=j;
          this.f=0;
          this.g=0;
          this.h=0;
          this.neighbors=[]
          this.previous=undefined;
          this.wall=false;


                    if(random(1)<0.1){
                              this.wall=true;
      }
```

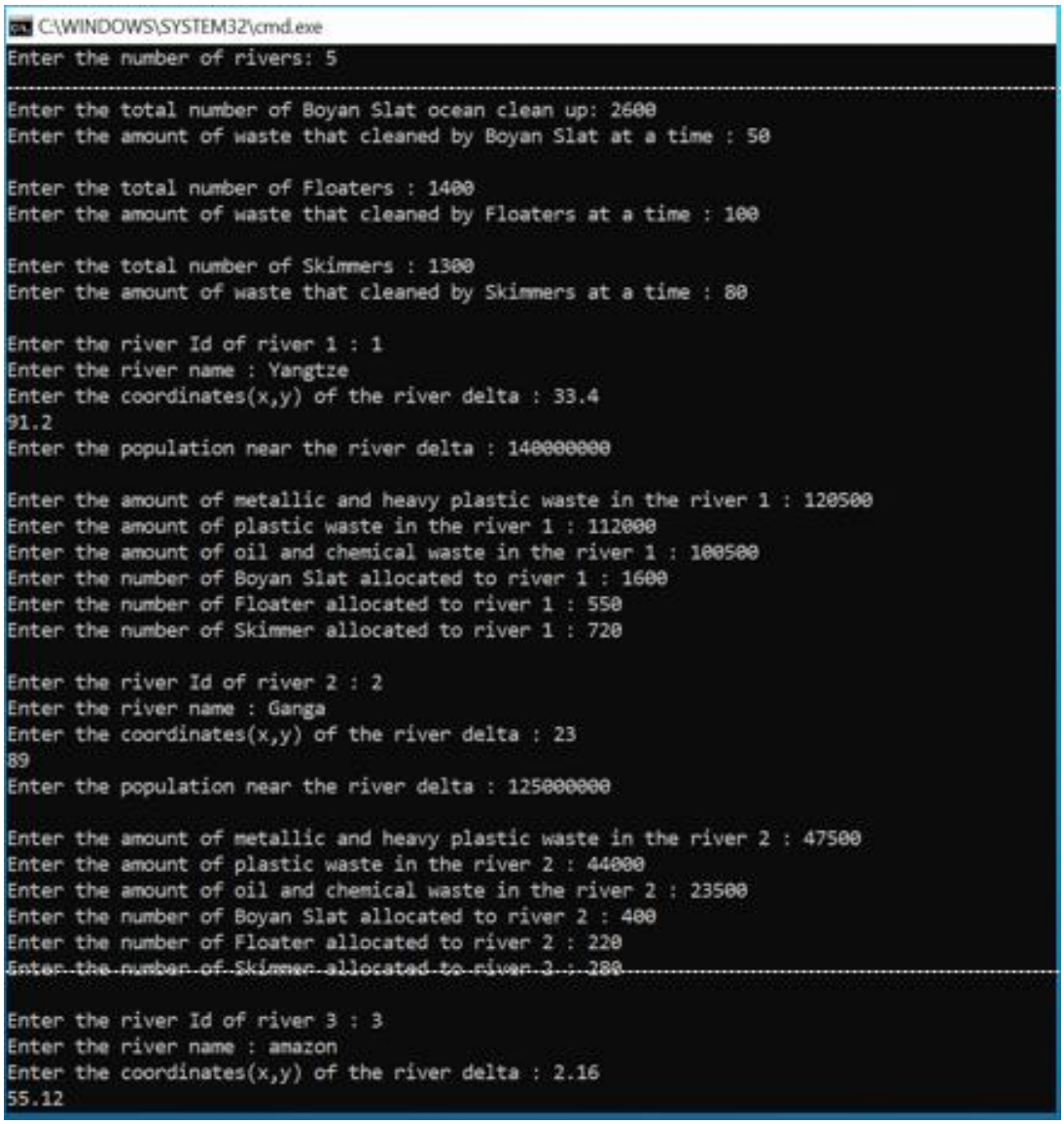
```
	this.show=function(col){
		fill(col);
		if(this.wall){
			fill(0);
		}
		noStroke();
		rect(this.i*w,this.j*h,w-1,h-1)
	}

	this.addneighbors=function(grid){
		var i=this.i;
		var j=this.j;
		if(i<cols-1){
			this.neighbors.push(grid[i+1][j]);
		}
		if(i>0){
			this.neighbors.push(grid[i-1][j]);
		}
		if(j<rows-1){
			this.neighbors.push(grid[i][j+1]);
		}
		if(j>0){
			this.neighbors.push(grid[i][j-1]);
		}
		if(i>0 && j>0){
			this.neighbors.push(grid[i-1][j-1]);
		}
		if(i<cols-1 && j>0){
			this.neighbors.push(grid[i+1][j-1]);
		}
		if(i>0 && j<rows-1){
			this.neighbors.push(grid[i-1][j+1]);
		}
		if(i<cols-1 && j<rows-1){
			this.neighbors.push(grid[i+1][j+1]);
		}
	}

}

function setup() {
  createCanvas(700,400);

	w=width/cols;
	h=height/rows;

  t=0;

	for (var i=0;i<cols;i++){
```

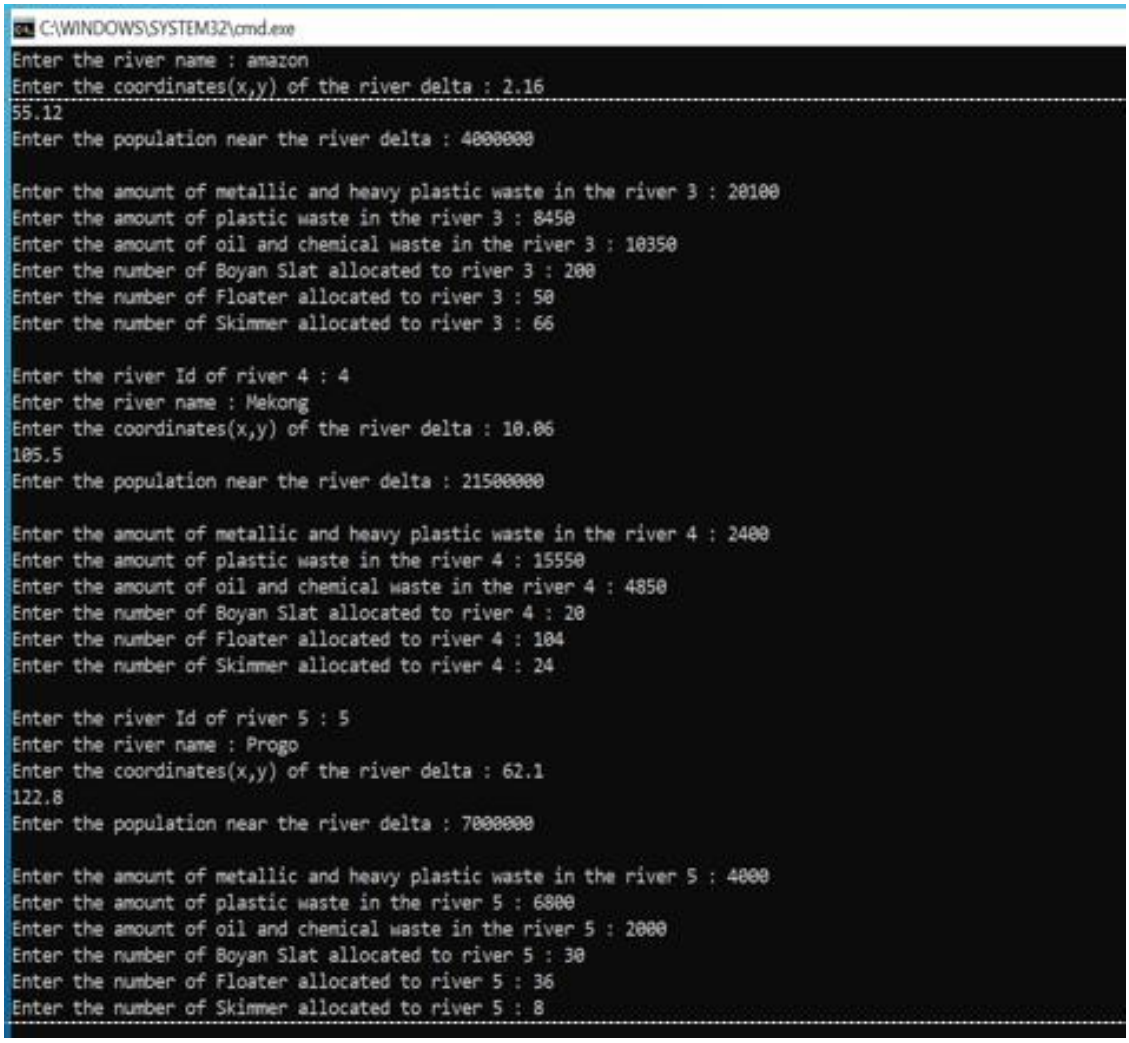
```
            grid[i]=new Array(rows);
        }

        for(var i=0;i<cols;i++){
            for(var j=0;j<rows;j++){
                grid[i][j]=new spot(i,j);
            }
        }

        for(var i=0;i<cols;i++){
            for(var j=0;j<rows;j++){
                grid[i][j].addneighbors(grid);
            }
        }

        start=grid[0][0];
        end=grid[cols-1][rows-1];
   end.wall=false;
   record.push(end);
        start.wall= false;
        input = createInput();
   input.position(100,410);
   addbutton = createButton('Add');
   addbutton.position(input.x + input.width,410);
   addbutton.mousePressed(add);
        openset.push(start);
   button = createButton('Start');
   button.position(input.x + input.width + addbutton.width,410);
   button.mousePressed(change);
}

function draw() {
  if(t==0){
  noLoop();
  }
        if(openset.length>0){

            var winner=0;
            for(var i=0;i<openset.length;i++){
                if(openset[i].f<openset[winner].f){
                    winner=i;
                }
            }
            var current = openset[winner];

            if(current === record[count]){
        console.log(display[count-1]);
        if(count==namecount)
        {
                    noLoop();
```

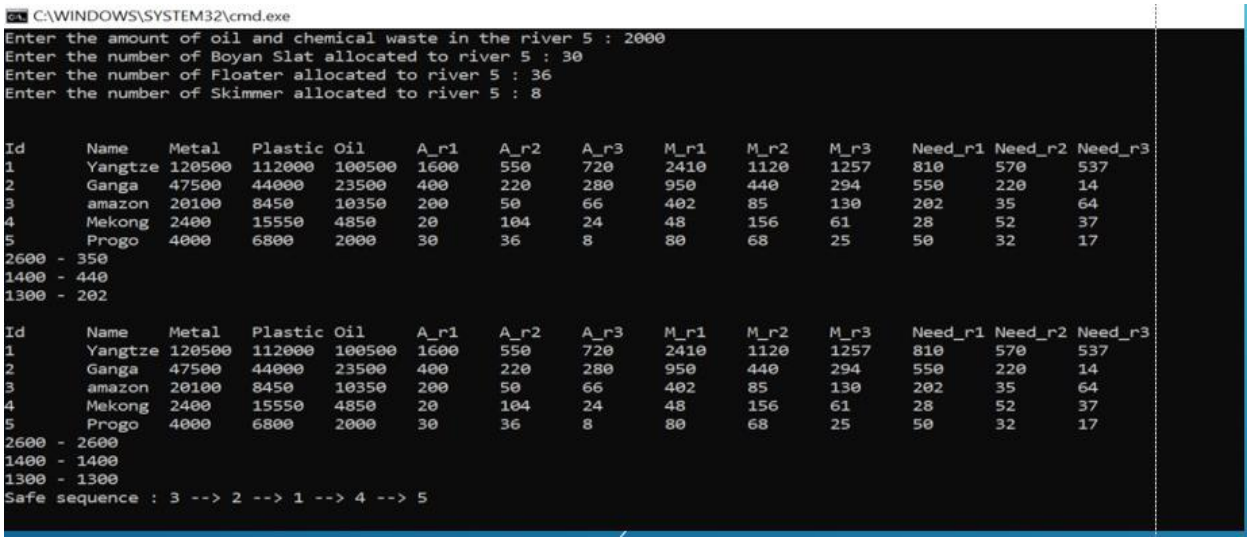
```
        count=count-1;
      }
      openset=[];
      openset.push(current);
      count=count+1;
            }

            removeFromArray(openset,current);

            closedset.push(current);

            var neighbors=current.neighbors;
            for(var i=0;i<neighbors.length;i++){
                  var neighbor=neighbors[i];

                  if(!closedset.includes(neighbor) && !neighbor.wall){
                        var tempg=current.g + 1;

                        var newpath=false;
                        if(openset.includes(neighbor)){
                              if(tempg<neighbor.g){
                                    neighbor.g=tempg;
                                    newpath=true;
                              }
                        } else{
                              neighbor.g=tempg;
                              newpath=true;
                              openset.push(neighbor);
                        }

                        if(newpath){
                        neighbor.h=heuristic(neighbor,record[count]);
                        neighbor.f=neighbor.g+neighbor.h;
                        neighbor.previous=current;
                        }

                  }
            }
      } else{
            console.log("No solution");
            noLoop();
            return;
      }
background(220);
      for(var i=0;i<cols;i++){
            for(var j=0;j<rows;j++){
                  grid[i][j].show(color(255));
            }
      }
      for(var i=0;i<closedset.length;i++){
```

```
                closedset[i].show(color(255,255,255));
        }
        for(var i=0;i<openset.length;i++){
                openset[i].show(color(255,255,255));
        }
        path=[];
        var temp=current;
        path.push(temp);
        while(temp.previous){
                path.push(temp.previous);
                temp=temp.previous;
        }
        for(var i=0;i<path.length;i++){
    for(var j=0;j<=count;j++)
    {
      if(path[i]==record[j])
      {
                 path[i].show(color(0,255,0));
       break;
      }
      else
      {
       path[i].show(color(0,0,255));
      }
    }
        }
}
```

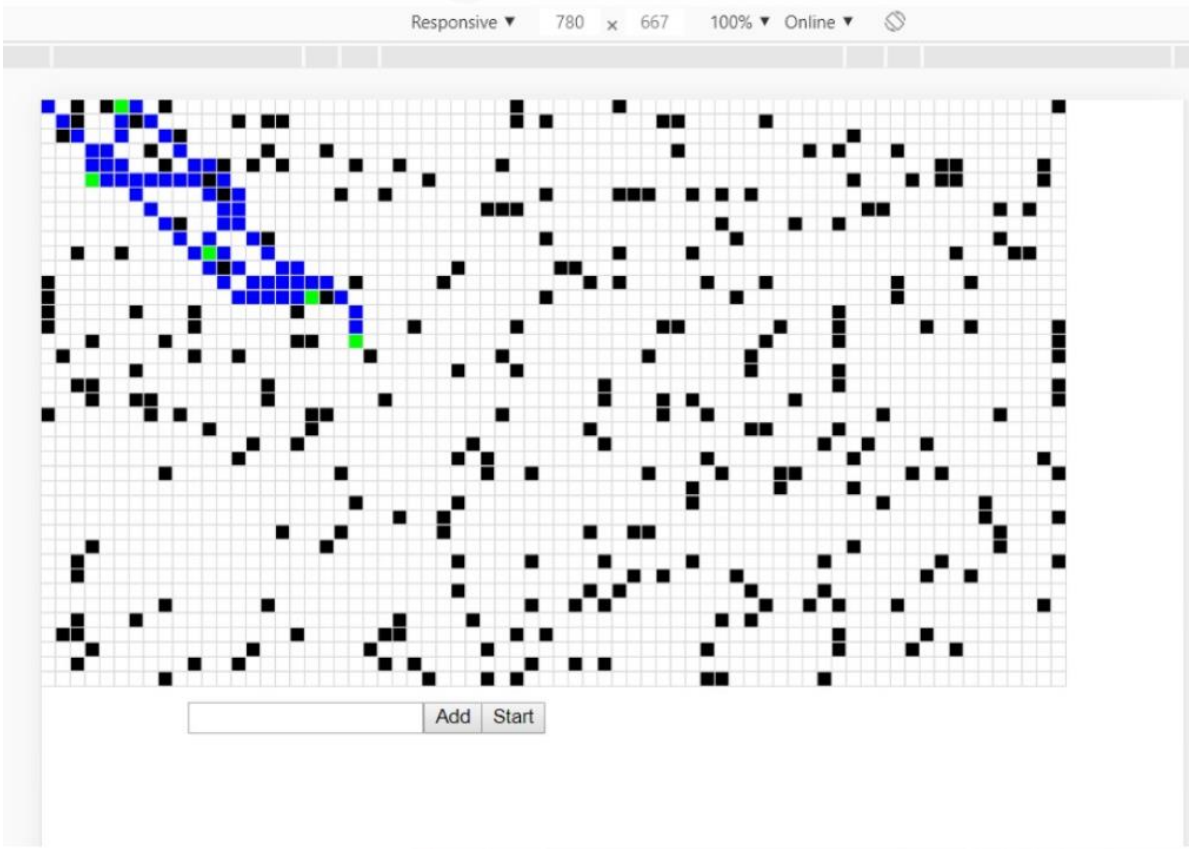
RESULTS:
C++ CODE OUTPUT:




C:\WINDOWS\SYSTEM32\cmd.exe
Enter the river name : amazon
Enter the coordinates(x,y) of the river delta : 2.16
55.12
Enter the population near the river delta : 4000000
Enter the amount of metallic and heavy plastic waste in the river 3 : 20100
Enter the amount of plastic waste in the river 3 : 8450
Enter the amount of oil and chemical waste in the river 3 : 10350
Enter the number of Boyan Slat allocated to river 3 : 200
Enter the number of Floater allocated to river 3 : 50
Enter the number of Skimmer allocated to river 3 : 66
Enter the river Id of river 4 : 4
Enter the river name : Mekong
Enter the coordinates(x,y) of the river delta : 10.06
105.5
Enter the population near the river delta : 21500000
Enter the amount of metallic and heavy plastic waste in the river 4 : 2400
Enter the amount of plastic waste in the river 4 : 15550
Enter the amount of oil and chemical waste in the river 4 : 4850
Enter the number of Boyan Slat allocated to river 4 : 20
Enter the number of Floater allocated to river 4 : 104
Enter the number of Skimmer allocated to river 4 : 24
Enter the river Id of river 5 : 5
Enter the river name : Progo
Enter the coordinates(x,y) of the river delta : 62.1
122.8
Enter the population near the river delta : 7000000
Enter the amount of metallic and heavy plastic waste in the river 5 : 4000
Enter the amount of plastic waste in the river 5 : 6800
Enter the amount of oil and chemical waste in the river 5 : 2000
Enter the number of Boyan Slat allocated to river 5 : 30
Enter the number of Floater allocated to river 5 : 36
Enter the number of Skimmer allocated to river 5 : 8


C:\WINDOWS\SYSTEM32\cmd.exe
Enter the amount of oil and chemical waste in the river 5 : 2000
Enter the number of Boyan Slat allocated to river 5 : 30
Enter the number of Floater allocated to river 5 : 36
Enter the number of Skimmer allocated to river 5 : 8
Id Name Metal Plastic Oil A_r1 A_r2 A_r3 M_r1 M_r2 M_r3 Need_r1 Need_r2 Need_r3
1 Yangtze 120500 112000 100500 1600 550 720 2410 1120 1257 810 570 537
2 Ganga 47500 44000 23500 400 220 280 950 440 294 550 220 14
3 amazon 20100 8450 10350 200 50 66 402 85 130 202 35 64
4 Mekong 2400 15550 4850 20 104 24 48 156 61 28 52 37
5 Progo 4000 6800 2000 30 36 8 80 68 25 50 32 17
2600 - 350
1400 - 440
1300 - 202
Id Name Metal Plastic Oil A_r1 A_r2 A_r3 M_r1 M_r2 M_r3 Need_r1 Need_r2 Need_r3
1 Yangtze 120500 112000 100500 1600 550 720 2410 1120 1257 810 570 537
2 Ganga 47500 44000 23500 400 220 280 950 440 294 550 220 14
3 amazon 20100 8450 10350 200 50 66 402 85 130 202 35 64
4 Mekong 2400 15550 4850 20 104 24 48 156 61 28 52 37
5 Progo 4000 6800 2000 30 36 8 80 68 25 50 32 17
2600 - 2600
1400 - 1400
1300 - 1300
Safe sequence : 3 --> 2 --> 1 --> 4 --> 5

JAVASCRIPT CODE OUTPUT:



By using the above codes we have managed to efficiently supply the ocean cleaning resources to clean up the oceans . By using this modified Banker's algorithm we are not only using the resources very efficiently but we are also able to save fuel needed for transportation of the resources. By applying this approach we also ensure that the severity of the pollution level at the different deltas is taken into account.

CONCLUSION:

By using this optimized approach we were able to use the various resources to the fullest and efficiently and we were also able to save fuel required for transportation. We also have successfully cleaned all the ocean deltas without causing any deadlock.

LIMITATIONS:

The model can be further improved by dynamically applying the Banker's algorithm by taking the changes of the input every time a process is completed.

REFERENCES

https://www.geeksforgeeks.org/
https://ourworldindata.org/plastic-pollution
https://www.nationalgeographic.com/environment/oceans/critical-issues-marine-pollution/
https://en.wikipedia.org/wiki/Banker%27s_algorithm