Name : Ashutosh Ardu

Regno : 20BRS1262

WATER JUG PROBLEM:

Problem: There are two jugs of **volume A litre** and **B litre.** Neither has any **measuring mark** on it.There is a pump that can be used to fill the jugs with water.How can you get exactly **x litre** of water into the **A litre jug.**Assuming that we have unlimited supply of water.

a) Let's assume we have **A=2 litre and B= 1 litre jugs**. And **we want exactly 1 Litre water into jug 1 (i.e 4 litre jug)** how we will do this.
b) Let's assume we have **A=4 litre and B= 3 litre jugs**. And **we want exactly 2 Litre water into jug A (i.e 4 litre jug)** how we will do this.

Implement the water jug problem to find the path from initial state to goal state

Solving using BFS

a)

Code:

```
#include "iostream"
#include "vector"
#include "algorithm"
using namespace std;

int counter=0,cur=-1;

struct state{
  int a,b;
};

void path(int *parent,vector<int> &ar,int i){
  if(parent[i]==i){
    ar.push_back(i);
    return;
  }
  else{
    ar.push_back(i);
    path(parent,ar,parent[i]);
  }
}

void print(vector<int> ar,state s[]){
  for(int i=0;i<ar.size();++i){
    if(i==ar.size()-1)
      cout<<"("<<s[ar[i]].a<<","<<s[ar[i]].b<<")"<<"\n";
    else
      cout<<"("<<s[ar[i]].a<<","<<s[ar[i]].b<<")"<<"-->";
  }
}

void show(int *list,state s[],int start,int end){
  for(int i=start;i<=end;++i){
    if(i==end)
      cout<<"("<<s[list[i]].a<<","<<s[list[i]].b<<")"<<endl;
    else if(i==start)
      cout<<"("<<s[list[i]].a<<","<<s[list[i]].b<<")"<<"::-->";
    else
      cout<<"("<<s[list[i]].a<<","<<s[list[i]].b<<")"<<",";
  }
}

void bfs(int **g,int v,int n,int *visited,int *list,int *parent,int
&c,int dest,state s[]){
  for(int i=0;i<6;++i){
    if(g[v][i] && !visited[g[v][i]]){
      parent[g[v][i]]=v;
      list[++cur]=g[v][i];
      visited[g[v][i]]=1;
      if(g[v][i]==dest){
          c=1;
          return;
        }
    }
  }
  show(list,s,counter,cur);
  if(++counter<=cur)
    bfs(g,list[counter],n,visited,list,parent,c,dest,s);
}

int main(){
  int n,start,dest,c=0;
  cout<<"Number of States: ";
  cin>>n;

  state s[n];
  vector<int> ar,op;

  cout<<"States:\n";
  for(int i=0;i<n;++i)
    cin>>s[i].a>>s[i].b;

  cout<<"Adjacency Matrix : \n";
  int **g=new int*[n];
  int *visited=new int[n];
  int *list=new int[n];
  int *parent=new int[n];

  for(int i=0;i<n;++i){
    g[i]=new int[n];
    visited[i]=0;
    parent[i]=i;
    for(int j=0;j<6;++j)
      cin>>g[i][j];
  }

  cout<<"Starting Point: ";
  cin>>start;
  cout<<"Destination Point: ";
  cin>>dest;
  list[++cur]=start;
  visited[start]=1;
  bfs(g,start,n,visited,list,parent,c,dest,s);
  if(c){
    show(list,s,counter,cur);
    cout<<"Path : \n";
    path(parent,ar,dest);
    reverse(ar.begin(),ar.end());
    print(ar,s);
  }
  else
    cout<<-1<<endl;
}
```

Input:

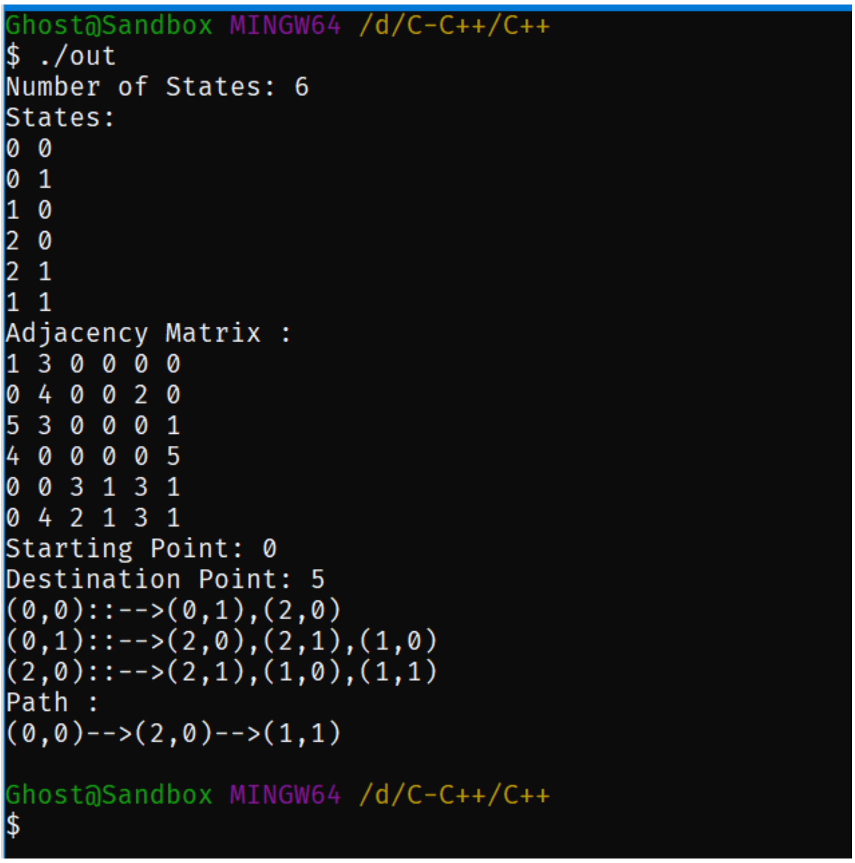
The Adjacency Matrix for Jug1(2L) and Jug2 (1L)

Index	Points	Fill J1	Fill J2	Empty J1	Empty J2	Transfer J1→J2	Transfer J2→J1
0	(0,0)	1	3	0	0	0	0
1	(0,1)	0	4	0	0	2	0
2	(1,0)	5	3	0	0	0	1
3	(2,0)	4	0	0	0	0	5
4	(2,1)	0	0	3	1	3	1
5	(1,1)	0	4	2	1	3	1

("0" here means either the operation cannot be performed or after performing the operation the initial & the final states remain the same)

From (0,0) → (1,1)

Output:



```
Ghost@Sandbox MINGW64 /d/C-C++/C++
$ ./out
Number of States: 6
States:
0 0
0 1
1 0
2 0
2 1
1 1
Adjacency Matrix :
1 3 0 0 0 0
0 4 0 0 2 0
5 3 0 0 0 1
4 0 0 0 0 5
0 0 3 1 3 1
0 4 2 1 3 1
Starting Point: 0
Destination Point: 5
(0,0)::-->(0,1),(2,0)
(0,1)::-->(2,0),(2,1),(1,0)
(2,0)::-->(2,1),(1,0),(1,1)
Path :
(0,0)-->(2,0)-->(1,1)

Ghost@Sandbox MINGW64 /d/C-C++/C++
$
```

b)

Input:

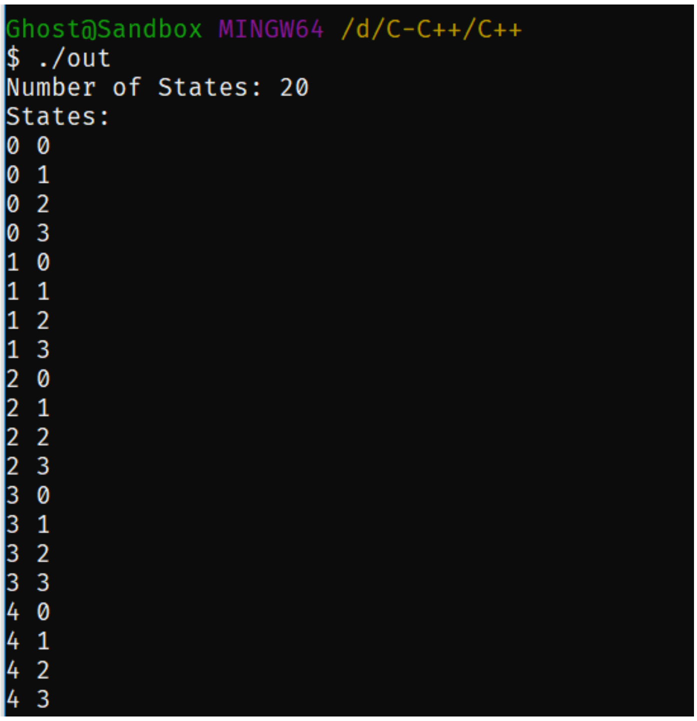
The Adjacency Matrix for Jug1(4L) and Jug2 (3L)

Index	Points	Fill J1	Fill J2	Empty J1	Empty J2	Transfer J1→J2	Transfer J2→J1
0	(0,0)	3	16	0	0	0	0
1	(0,1)	3	17	0	0	4	0
2	(0,2)	3	18	0	0	8	0
3	(0,3)	0	19	0	0	12	0
4	(1,0)	7	16	0	0	0	1
5	(1,1)	7	17	4	1	8	2
6	(1,2)	7	18	4	2	12	3
7	(1,3)	0	19	4	3	16	0
8	(2,0)	11	16	0	0	0	2
9	(2,1)	11	17	8	1	12	3
10	(2,2)	11	18	8	2	16	7
11	(2,3)	0	19	8	3	17	0
12	(3,0)	15	16	0	0	0	4
13	(3,1)	15	17	12	1	16	7
14	(3,2)	15	18	12	2	17	11
15	(3,3)	0	19	12	3	18	0
16	(4,0)	19	0	0	0	0	7
17	(4,1)	19	0	16	1	0	11
18	(4,2)	19	0	16	2	0	15
19	(4,3)	0	0	16	3	0	0

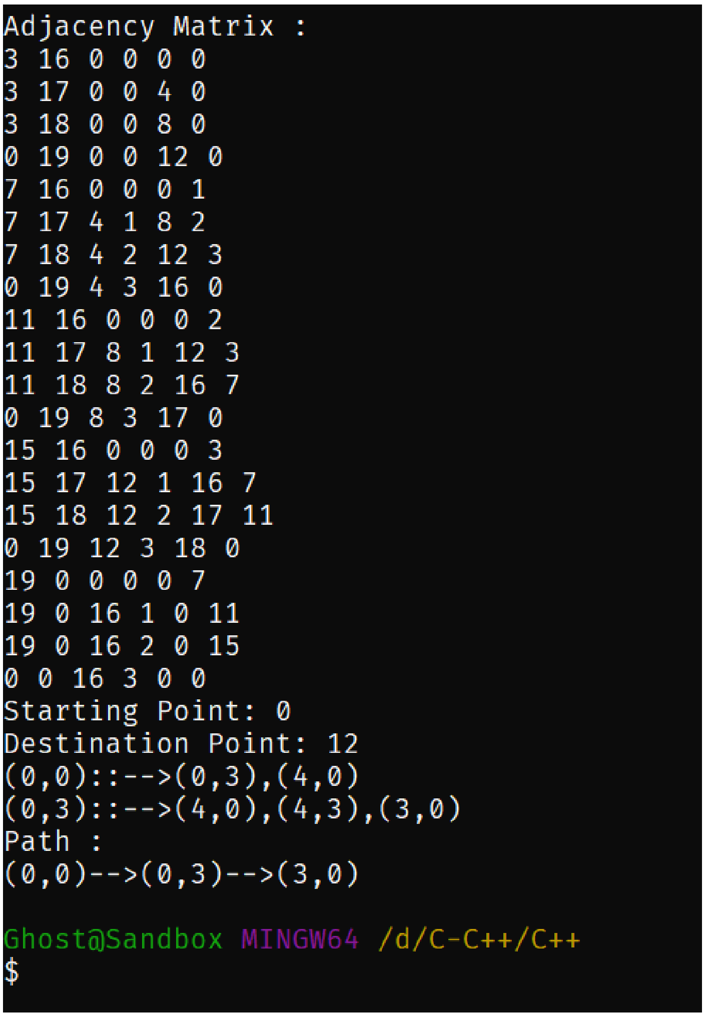
("0" here means either the operation cannot be performed or after performing the operation the initial & the final states remain the same)

From (0,0) → (3,0)

Output:



```
Ghost@Sandbox MINGW64 /d/C-C++/C++
$ ./out
Number of States: 20
States:
0 0
0 1
0 2
0 3
1 0
1 1
1 2
1 3
2 0
2 1
2 2
2 3
3 0
3 1
3 2
3 3
4 0
4 1
4 2
4 3
```



```
Adjacency Matrix :
3 16 0 0 0 0
3 17 0 0 4 0
3 18 0 0 8 0
0 19 0 0 12 0
7 16 0 0 0 1
7 17 4 1 8 2
7 18 4 2 12 3
0 19 4 3 16 0
11 16 0 0 0 2
11 17 8 1 12 3
11 18 8 2 16 7
0 19 8 3 17 0
15 16 0 0 0 3
15 17 12 1 16 7
15 18 12 2 17 11
0 19 12 3 18 0
19 0 0 0 0 7
19 0 16 1 0 11
19 0 16 2 0 15
0 0 16 3 0 0
Starting Point: 0
Destination Point: 12
(0,0)::-->(0,3),(4,0)
(0,3)::-->(4,0),(4,3),(3,0)
Path :
(0,0)-->(0,3)-->(3,0)

Ghost@Sandbox MINGW64 /d/C-C++/C++
$
```

Solving using DFS

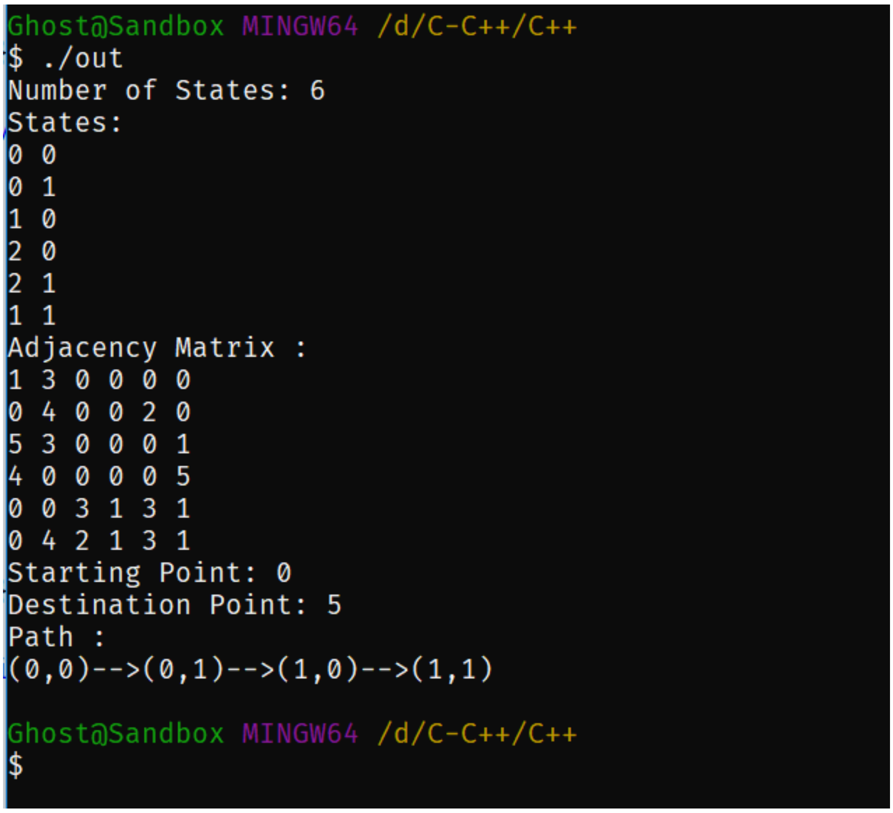
a)

Code:

```
#include "iostream"
#include "vector"
#include "algorithm"
using namespace std;

int counter=0,cur=-1;

struct state{
  int a,b;
};

void path(int *parent,vector<int> &ar,int i){
  if(parent[i]==i){
    ar.push_back(i);
    return;
  }
  else{
    ar.push_back(i);
    path(parent,ar,parent[i]);
  }
}

void print(vector<int> ar,state s[]){
  for(int i=0;i<ar.size();++i){
    if(i==ar.size()-1)
      cout<<"("<<s[ar[i]].a<<","<<s[ar[i]].b<<")"<<"\n";
    else
      cout<<"("<<s[ar[i]].a<<","<<s[ar[i]].b<<")"<<"-->";
  }
}

void show(int *list,state s[],int start,int end){
  for(int i=start;i<=end;++i){
    if(i==end)
      cout<<"("<<s[list[i]].a<<","<<s[list[i]].b<<")"<<endl;
    else if(i==start)
      cout<<"("<<s[list[i]].a<<","<<s[list[i]].b<<")"<<"::-->";
    else
      cout<<"("<<s[list[i]].a<<","<<s[list[i]].b<<")"<<",";
  }
}

void output(int *parent,int n){
  for(int i=0;i<n;++i)
    cout<<parent[i]<<" ";
  cout<<endl;
}

void dfs(int **g,int v,int n,int *visited,int *parent,int &c,int
dest,state s[]){
    visited[v]=1;
  for(int i=0;i<6;++i){
    if(g[v][i] && !visited[g[v][i]]){
      parent[g[v][i]]=v;
      if(g[v][i]==dest){
          c=1;
          return;
        }
            dfs(g,g[v][i],n,visited,parent,c,dest,s);
    }
  }
}

int main(){
  int n,start,dest,c=0;
  cout<<"Number of States: ";
  cin>>n;

  state s[n];
  vector<int> ar;

  cout<<"States:\n";
  for(int i=0;i<n;++i)
    cin>>s[i].a>>s[i].b;

  cout<<"Adjacency Matrix : \n";
  int **g=new int*[n];
  int *visited=new int[n];
  int *list=new int[n];
  int *parent=new int[n];

  for(int i=0;i<n;++i){
    g[i]=new int[n];
    visited[i]=0;
    parent[i]=i;
    for(int j=0;j<6;++j)
      cin>>g[i][j];
  }

  cout<<"Starting Point: ";
  cin>>start;
  cout<<"Destination Point: ";
  cin>>dest;
  dfs(g,start,n,visited,parent,c,dest,s);
  if(c){
    cout<<"Path : \n";
    path(parent,ar,dest);
    reverse(ar.begin(),ar.end());
    print(ar,s);
  }
  else
    cout<<-1<<endl;
}
```

From (0,0) → (1,1)

Output:

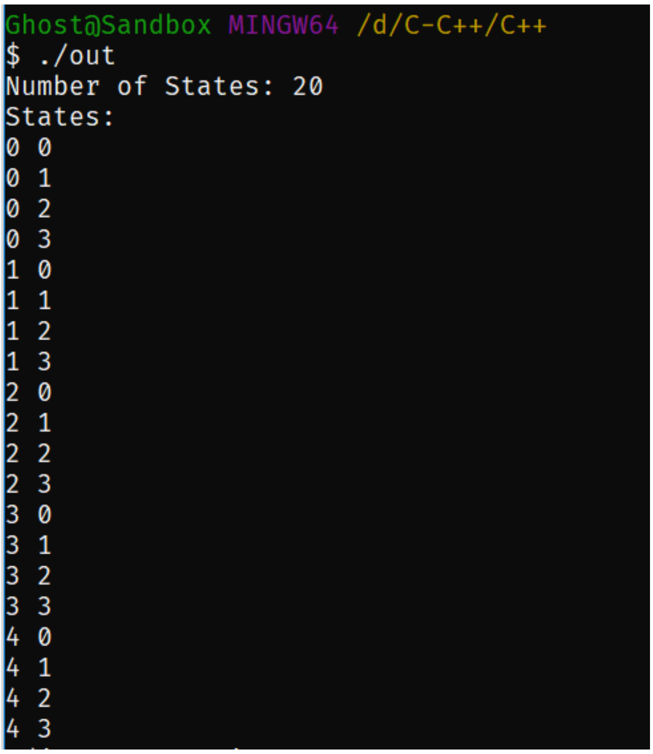


```
Ghost@Sandbox MINGW64 /d/C-C++/C++
$ ./out
Number of States: 6
States:
0 0
0 1
1 0
2 0
2 1
1 1
Adjacency Matrix :
1 3 0 0 0 0
0 4 0 0 2 0
5 3 0 0 0 1
4 0 0 0 0 5
0 0 3 1 3 1
0 4 2 1 3 1
Starting Point: 0
Destination Point: 5
Path :
(0,0)-->(0,1)-->(1,0)-->(1,1)

Ghost@Sandbox MINGW64 /d/C-C++/C++
$
```

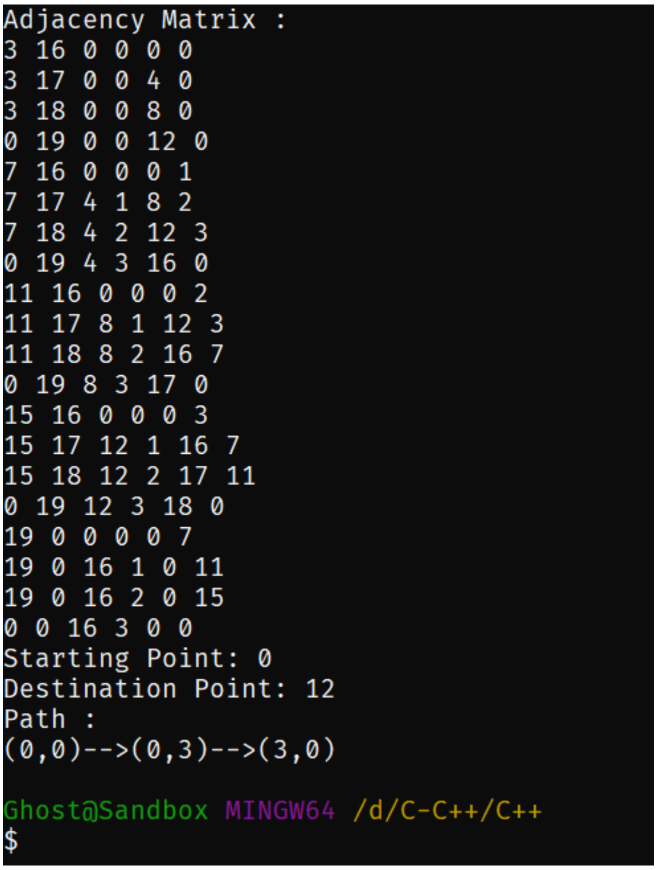
b)

From (0,0) → (3,0)

Output:



```
Ghost@Sandbox MINGW64 /d/C-C++/C++
$ ./out
Number of States: 20
States:
0 0
0 1
0 2
0 3
1 0
1 1
1 2
1 3
2 0
2 1
2 2
2 3
3 0
3 1
3 2
3 3
4 0
4 1
4 2
4 3
```



```
Adjacency Matrix :
3 16 0 0 0 0
3 17 0 0 4 0
3 18 0 0 8 0
0 19 0 0 12 0
7 16 0 0 0 1
7 17 4 1 8 2
7 18 4 2 12 3
0 19 4 3 16 0
11 16 0 0 0 2
11 17 8 1 12 3
11 18 8 2 16 7
0 19 8 3 17 0
15 16 0 0 0 3
15 17 12 1 16 7
15 18 12 2 17 11
0 19 12 3 18 0
19 0 0 0 0 7
19 0 16 1 0 11
19 0 16 2 0 15
0 0 16 3 0 0
Starting Point: 0
Destination Point: 12
Path :
(0,0)-->(0,3)-->(3,0)

Ghost@Sandbox MINGW64 /d/C-C++/C++
$
```