Bansilal Ramnath Agarwal Charitable Trust's

Vishwakarma Institute of Technology Pune-37

(Anautonomous Institute of Savitribai Phule Pune University)

Department of Computer Engineering

Division	C
Batch	2
GR-no	12111180
Rollno	17
Name	Divya Mahajan

Title: Implementation of Uninformed strategies.

Description:

Implementation of Water-jug problem using Breadth First Search (BFS):

Following are the steps:

1. A state is defined as (x, y) representing the amount of water in the 4-gallon and 3-gallon jugs, respectively.

2. Initialize a queue for BFS traversal. Start by adding the initial state (0, 0) and a list of states to store the path (which is initially empty), to the queue.

3. Start a BFS loop while the queue is not empty:

4. Dequeue the front state (x, y) from the queue.

5. Check if (x, y) is the desired amount. If yes, you have found the solution.

6. Otherwise, generate all possible next states by applying the following actions:

    1) If x < 4, Fill the 4-gallon jug: (4, y)
    2) If y < 3, Fill the 3-gallon jug: (x, 3)
    3) If x > 0, Empty the 4-gallon jug: (0, y)
    4) If y > 0, Empty the 3-gallon jug: (x, 0)
    5) If x + y >= 4 and y > 0, Pour water from the 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full: (4, y-(4-x))
    6) If x + y >= 3 and x > 3, Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full: (x-(3-y), 3)
    7) If x + y <= 4 and y > 0, Pour all the water from the 3-gallon jug into the 4-gallon jug: (x + y, 0)
    8) If x + y <= 3 and x > 0, Pour all the water from the 4-gallon jug into the 3-gallon jug: (0, x + y)

    Every time the state is added to the queue, the path is updated.

Implementation of Water-jug problem using Depth First Search (DFS):

1. A state is defined as (x, y) representing the amount of water in the 4-gallon and 3-gallon jugs, respectively.

2. Make a DFS call by passing the initial state (0, 0), final state, a list of states to store the path (which is initially empty), a set to track visited states and depth of the DFS tree (initially zero).

3. In the DFS function, if the depth has reached the threshold value, return from the DFS call.

4. Else, insert the state in the visited set.

5. Check if (x, y) is the desired amount. If yes, you have found the solution.

6. Otherwise, generate all possible next states by applying the actions mentioned in the above BFS solution.

   Every time the possible non-visited state is passed in the DFS function and the path is updated.

Note: If the depth is used to restrict the DFS calls, the method is called **Depth Limited Search (DLS)**.

Implementation of 8-Puzzle problem using Breadth First Search (BFS):

Following are the steps:

1. A state is represented as a 2D matrix.

2. Initialize a queue for BFS traversal. Start by adding the initial state matrix and a list of 2D matrices (initially empty) to represent the path, to the queue.

3. Start a BFS loop while the queue is not empty:

4. Dequeue the front state from the queue.

5. Check if the current state is equal to the goal state. If yes, you have found the solution.

6. Otherwise, generate all possible next grid configurations by sliding the empty

space in all four directions (up, down, left, and right).

Every time the state is added to the queue, the path is updated.

Implementation of 8-Puzzle problem using Depth First Search (DFS):

Following are the steps:

1. A state is represented as a 2D matrix.
2. Make a DFS call by passing the initial state, final state, a list of states to store the path (which is initially empty), the coordinates of empty cell, a set to track visited states and depth of the DFS tree (initially zero).
3. In the DFS function, if the depth has reached the threshold value, return from the DFS call.
4. Else, insert the state in the visited set.
5. Check if the current state is equal to the goal state. If yes, you have found the solution.
6. Otherwise, generate all possible next grid configurations by sliding the empty space in all four directions (up, down, left, and right).

Every time the possible non-visited state is passed in the DFS function and the path is updated.

Note: As the depth is used to restrict the DFS calls, the method is called **Depth Limited Search (DLS)**.

Code:

- Water-jug problem (BFS)

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

int main()
```

```
{
    pair<int, int> start = {0, 0};
    pair<int, int> end = {2, 0};
    queue<pair<pair<int, int>, vector<pair<int, int>>>> q;
    q.push({start, {{0, 0}}});
    while (!q.empty())
    {
        bool flag = false;
        int size = q.size();
        while (size--)
        {
            pair<int, int> curr = q.front().first;
            vector<pair<int, int>> vec = q.front().second;
            int x = curr.first;
            int y = curr.second;
            q.pop();
            if (curr == end)
            {
                cout << "Possible solution:" << endl;
                for (auto it : vec)
                    cout << it.first << " " << it.second << endl;
                flag = true;
                break;
            }
            if (x < 4)
            {
                pair<int, int> node = {4, y};
                vec.push_back(node);
                q.push({node, vec});
                vec.pop_back();
            }
            if (y < 3)
            {
                pair<int, int> node = {x, 3};
                vec.push_back(node);
                q.push({node, vec});
                vec.pop_back();
            }
            if (x > 0)
```

```
            {
                pair<int, int> node = {0, y};
                vec.push_back(node);
                q.push({node, vec});
                vec.pop_back();
            }
            if (y > 0)
            {
                pair<int, int> node = {x, 0};
                vec.push_back(node);
                q.push({node, vec});
                vec.pop_back();
            }
            if (x + y >= 4 && y > 0)
            {
                pair<int, int> node = {4, y - (4 - x)};
                vec.push_back(node);
                q.push({node, vec});
                vec.pop_back();
            }
            if (x + y >= 3 && x > 0)
            {
                pair<int, int> node = {x - (3 - y), 3};
                vec.push_back(node);
                q.push({node, vec});
                vec.pop_back();
            }
            if (x + y <= 4 && y > 0)
            {
                pair<int, int> node = {x + y, 0};
                vec.push_back(node);
                q.push({node, vec});
                vec.pop_back();
            }
            if (x + y <= 3 && x > 0)
            {
                pair<int, int> node = {0, x + y};
                vec.push_back(node);
                q.push({node, vec});
```

```
                vec.pop_back();
            }
        }
        if (flag)
            break;
    }
    return 0;
}
```

- Water-jug Problem (DFS)

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

bool dfs(pair<int, int> curr, pair<int, int> end, vector<pair<int,
int>> vec, set<pair<int, int>> &vis, int depth)
{
    // if (depth > 10)
    //     return false;

    bool ans = false;
    int x = curr.first;
    int y = curr.second;
    vis.insert(curr);

    if (curr == end)
    {
        cout << "Possible solution:" << endl;
        for (auto it : vec)
            cout << it.first << " " << it.second << endl;
        return true;
    }
    if (x < 4)
    {
        pair<int, int> node = {4, y};
        if (!vis.count(node))
        {
            vec.push_back(node);
```

```
            if (dfs(node, end, vec, vis, depth + 1))
                ans = true;
            vec.pop_back();
        }
    }
    if (y < 3)
    {
        pair<int, int> node = {x, 3};
        if (!vis.count(node))
        {
            vec.push_back(node);
            if (dfs(node, end, vec, vis, depth + 1))
                ans = true;
            vec.pop_back();
        }
    }
    if (x > 0)
    {
        pair<int, int> node = {0, y};
        if (!vis.count(node))
        {
            vec.push_back(node);
            if (dfs(node, end, vec, vis, depth + 1))
                ans = true;
            vec.pop_back();
        }
    }
    if (y > 0)
    {
        pair<int, int> node = {x, 0};
        if (!vis.count(node))
        {
            vec.push_back(node);
            if (dfs(node, end, vec, vis, depth + 1))
                ans = true;
            vec.pop_back();
        }
    }
    if (x + y >= 4 && y > 0)
```

```
    {
        pair<int, int> node = {4, y - (4 - x)};
        if (!vis.count(node))
        {
            vec.push_back(node);
            if (dfs(node, end, vec, vis, depth + 1))
                ans = true;
            vec.pop_back();
        }
    }
    if (x + y >= 3 && x > 0)
    {
        pair<int, int> node = {x - (3 - y), 3};
        if (!vis.count(node))
        {
            vec.push_back(node);
            if (dfs(node, end, vec, vis, depth + 1))
                ans = true;
            vec.pop_back();
        }
    }
    if (x + y <= 4 && y > 0)
    {
        pair<int, int> node = {x + y, 0};
        if (!vis.count(node))
        {
            vec.push_back(node);
            if (dfs(node, end, vec, vis, depth + 1))
                ans = true;
            vec.pop_back();
        }
    }
    if (x + y <= 3 && x > 0)
    {
        pair<int, int> node = {0, x + y};
        if (!vis.count(node))
        {
            vec.push_back(node);
            if (dfs(node, end, vec, vis, depth + 1))
```

```
                ans = true;
            vec.pop_back();
        }
    }
    return ans;
}

int main()
{
    pair<int, int> start = {0, 0};
    pair<int, int> end = {2, 0};
    vector<pair<int, int>> ans;
    set<pair<int, int>> vis;
    if (!dfs(start, end, ans, vis, 0))
        cout << "No possible solution." << endl;
    return 0;
}
```

- 8-Puzzle Problem (BFS)

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

bool checkEqual(vector<vector<int> > curr, vector<vector<int> >
final){
   for(int i=0; i<3; i++){
      for(int j=0; j<3; j++){
         if(curr[i][j] != final[i][j])
            return false;
      }
   }
   return true;
}

pair<int, int> findBlank(vector<vector<int> > vec){
   pair<int, int> ans;
   for(int i=0; i<3; i++){
      for(int j=0; j<3; j++){
```

```
            if(vec[i][j] == 0)
                ans = {i, j};
        }
    }
    return ans;
}

void printAns(vector<vector<vector<int> > > vec){
    for(auto mat : vec){
        for(auto row : mat){
            for(auto el : row){
                cout<<el<<" ";
            }
            cout<<endl;
        }
        cout<<endl;
    }
}

int main(){
    vector<vector<int> > start = {{2, 8, 3},
                                  {1, 6, 4},
                                  {7, 0, 5}};
    vector<vector<int> > final = {{1, 2, 3},
                                  {8, 0, 4},
                                  {7, 6, 5}};
    vector<vector<vector<int> > > ans;
    queue<pair<vector<vector<int> >, vector<vector<vector<int> > > > >
q;
    q.push({start, ans});
    //              R  D   L  U
    int delrow[] = {0, 1, 0, -1};
    int delcol[] = {1, 0, -1, 0};
    while(!q.empty()){
        bool flag = false;
        int size = q.size();
        while(size--){
            vector<vector<int> > curr = q.front().first;
            vector<vector<vector<int> > > vec = q.front().second;
```

```
            q.pop();
            if(checkEqual(curr, final)){
               cout<<"Possible solution:"<<endl<<endl;
               printAns(vec);
               flag = true;
               break;
            }
            pair<int, int> blank = findBlank(curr);
            int row = blank.first;
            int col = blank.second;
            for(int i=0; i<4; i++){
               int nrow = row + delrow[i];
               int ncol = col + delcol[i];
               if(nrow >= 0 && nrow < 3 && ncol >= 0 && ncol < 3){
                  swap(curr[row][col], curr[nrow][ncol]);
                  vec.push_back(curr);
                  q.push({curr, vec});
                  vec.pop_back();
                  swap(curr[row][col], curr[nrow][ncol]);
               }
            }
         }
         if(flag)
            break;
      }
      return 0;
}
```

- 8-Puzzle Problem (DFS)

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

bool checkEqual(vector<vector<int>> &curr, vector<vector<int>> &final)
{
   return curr == final;
}

void findBlank(vector<vector<int>> curr, int &row, int &col)
```

```
{
   for (int i = 0; i < 3; i++)
   {
      for (int j = 0; j < 3; j++)
      {
         if (curr[i][j] == 0)
         {
            row = i;
            col = j;
            return;
         }
      }
   }
}

void printAns(vector<vector<vector<int>>> vec)
{
   for (auto mat : vec)
   {
      for (auto row : mat)
      {
         for (auto el : row)
         {
            cout << el << " ";
         }
         cout << endl;
      }
      cout << endl;
   }
}

bool dfs(vector<vector<int>> &curr, vector<vector<int>> &final, int
depth, int row, int col, set<vector<vector<int>>> &vis,
vector<vector<vector<int>>> &vec)
{
   if (depth > 10)
      return false;

   vec.push_back(curr);
```

```
    vis.insert(curr);

    if (checkEqual(curr, final))
    {
        cout << "Possible solution:" << endl
             << endl;
        printAns(vec);
        return true;
    }

    int delrow[] = {0, 0, -1, 1};
    int delcol[] = {1, -1, 0, 0};

    for (int i = 0; i < 4; i++)
    {
        int nrow = row + delrow[i];
        int ncol = col + delcol[i];

        if (nrow >= 0 && nrow < 3 && ncol >= 0 && ncol < 3)
        {
            swap(curr[row][col], curr[nrow][ncol]);
            if (!vis.count(curr))
            {
                if (dfs(curr, final, depth + 1, nrow, ncol, vis, vec))
                    return true;
            }
            swap(curr[row][col], curr[nrow][ncol]);
        }
    }
    vec.pop_back();

    return false;
}

int main()
{
    vector<vector<int>> start = {{2, 8, 3},
                                 {1, 6, 4},
                                 {7, 0, 5}};
```

```
    vector<vector<int>> final = {{1, 2, 3},
                                 {8, 0, 4},
                                 {7, 6, 5}};

    int row, col;
    findBlank(start, row, col);
    vector<vector<vector<int>>> ans;
    set<vector<vector<int>>> vis;
    if (!dfs(start, final, 0, row, col, vis, ans))
        cout << "No possible solution." << endl;

    return 0;
}
```

Screenshots/Output:

Water-jug Problem (BFS):

```
PS D:\AI\Lab> cd "d:\AI\Lab\Assign2\"
Possible solution:
0 0
0 3
3 0
3 3
4 2
0 2
2 0
PS D:\AI\Lab\Assign2>
```

Water-jug Problem (DFS):

```
PS D:\AI\Lab> cd "d:\AI\Lab\Assign2\" ;
Possible solution:
4 0
4 3
0 3
3 0
3 3
4 2
0 2
2 0
PS D:\AI\Lab\Assign2>
```

8-Puzzle problem (BFS):

```
PS D:\AI\Lab> cd "d:\AI\Lab\Assign2\" ;
Possible solution:

2 8 3
1 0 4
7 6 5

2 0 3
1 8 4
7 6 5

0 2 3
1 8 4
7 6 5

1 2 3
0 8 4
7 6 5

1 2 3
8 0 4
7 6 5

PS D:\AI\Lab\Assign2>
```

8-Puzzle problem (DFS):

```
PS D:\AI\Lab> cd "d:\AI\Lab\Assign2\" ;
Possible solution:

2 8 3
1 6 4
7 0 5

2 8 3
1 0 4
7 6 5

2 0 3
1 8 4
7 6 5

0 2 3
1 8 4
7 6 5

1 2 3
0 8 4
7 6 5

1 2 3
8 0 4
7 6 5

PS D:\AI\Lab\Assign2>
```