

Initial Case 1:

Memory usage: 530.6171875KB

Runtime: 6 Milliseconds

```

1 import java.util.*;
2
3 public class bfs {
4
5     static String[] goalnode = {"1","2","3","4","5","6","7","8","9","10","11","12","13","14","15","0"};
6     public static void main(String args[]){
7         long startTime = System.currentTimeMillis();
8         String[] node;
9         Queue<String[]> frontier = new LinkedList<String[]>();
10        List<String[]> explored = new LinkedList<String[]>();
11        String[] rootnode = {"1","2","3","4","5","6","7","8","9","0","10","11","13","14","15","12"};
12        frontier.add(rootnode);
13        while(!frontier.isEmpty()){
14            node=frontier.poll();
15            explored.add(node);
16            String[] checkedNode = {};
17            for(int x=0;x<4;x++){
18                if(x==0){
19                    checkedNode = Arrays.copyOf(up(node), 16);
20                }
21                else if(x==1){
22

```

Console Output:

```

[1, 2, 3, 4, 5, 6, 7, 8, 0, 9, 10, 11, 13, 14, 15, 12]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 11, 13, 14, 15, 12]
[1, 2, 3, 4, 5, 6, 7, 0, 9, 10, 11, 8, 13, 14, 15, 12]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0]
Solution found.
Memory usage: 530.6171875KB
Runtime: 7 Milliseconds

```

Initial Case 2:

Memory usage: 9605.8515625KB

Runtime: 57523 Milliseconds

```

3 import java.util.List;
4 import java.util.Queue;
5
6 public class bfs2 {
7
8     static String[] goalnode = {"1","2","3","4","5","6","7","8","9","10","11","12","13","14","15","0"};
9     public static void main(String args[]){
10        long startTime = System.currentTimeMillis();
11        String[] node;
12        Queue<String[]> frontier = new LinkedList<String[]>();
13        List<String[]> explored = new LinkedList<String[]>();
14        String[] rootnode = {"1","2","3","4","5","6","7","8","0","9","10","15","13","14","12","11"};
15        frontier.add(rootnode);
16        while(!frontier.isEmpty()){
17            node=frontier.poll();
18            explored.add(node);
19            String[] checkedNode = {};
20            for(int x=0;x<4;x++){
21                if(x==0){
22                    checkedNode = Arrays.copyOf(up(node), 16);
23                }
24            }

```

Console Output:

```

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 12, 13, 14, 11, 15]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 0, 15]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 0, 14, 15]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0]
Solution found.
Memory usage: 9605.8515625KB
Runtime: 56647 Milliseconds

```

Initial Case 3:

Memory usage: ?

Runtime: ?

The screenshot shows the Eclipse IDE with the file `bfs3.java` open. The code is a Java program that implements a Breadth-First Search (BFS) algorithm. It starts with a goal node and explores nodes in a queue until it finds the goal. The console output shows the sequence of nodes explored, starting from the goal node and moving through various states of a 3x3 grid.

```

1 import java.util.Arrays;
2 import java.util.LinkedList;
3 import java.util.List;
4 import java.util.Queue;
5
6 public class bfs3 {
7
8     static String[] goalnode = {"1","2","3","4","5","6","7","8","9","10","11","12","13","14","15","0"};
9     public static void main(String args[]){
10         long startTime = System.currentTimeMillis();
11         String[] node;
12         Queue<String[]> frontier = new LinkedList<String[]>();
13         List<String[]> explored = new LinkedList<String[]>();
14         String[] rootnode = {"1","2","3","4","5","6","7","8","9","10","11","12","13","14","15","6"};
15         frontier.add(rootnode);
16         while(!frontier.isEmpty()){
17             node=frontier.poll();
18             explored.add(node);
19             String[] checkedNode = {};
20             for(int x=0;x<4;x++){
21                 if(x==0){

```

The console output shows the sequence of nodes explored, starting from the goal node and moving through various states of a 3x3 grid. The output is as follows:

```

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 6]
[1, 2, 3, 4, 5, 8, 12, 10, 9, 7, 15, 11, 13, 14, 0, 6]
[1, 2, 3, 4, 5, 8, 12, 10, 9, 0, 7, 11, 13, 14, 15, 6]
[1, 2, 3, 4, 5, 8, 12, 10, 9, 7, 11, 0, 13, 14, 15, 6]
[1, 2, 3, 4, 5, 8, 12, 10, 9, 7, 0, 11, 13, 14, 15, 6]
[1, 2, 3, 4, 5, 8, 12, 10, 9, 7, 15, 11, 13, 14, 0, 6]
[1, 2, 3, 4, 5, 8, 12, 10, 9, 7, 15, 11, 13, 0, 14, 6]
[1, 2, 3, 4, 5, 8, 12, 10, 9, 7, 15, 11, 13, 14, 6, 0]
[1, 2, 3, 4, 5, 8, 12, 10, 9, 0, 15, 11, 13, 7, 14, 6]

```

