BFS:

```
#include<bits/stdc++.h>
#define X -1
#define G 100

using namespace::std;

vector<int> get_adjs(vector<int> graph, int node, int m, int n){
      vector<int> ans;
      if((node >= n) && (graph[node - n] != X))
      ans.push_back(node-n);           // up
      if((node + n < m * n) && (graph[node + n] != X))
      ans.push_back(node + n); // down
      if(((node % n) != 0)  && (graph[node - 1] != X))
      ans.push_back(node - 1);         // left
      if(((node + 1) % m != 0) && (graph[node + 1] != X))
      ans.push_back(node + 1); // right
      return ans;
}

void print_queue(queue<int> q){
      for(queue<int> tq = q; !tq.empty();){
      cout << tq.front() << ", ";
      tq.pop();
      }
}

void print_path(vector<int> path, string delimiter, int m, int n){
      int i;
      for(i=0; i < path.size() - 1; i++)
      printf("(%d, %d) -> ", path[i] / m, path[i] % m);
      printf("(%d, %d)", path[i] / m, path[i] % m);
}

void print_graph(vector<int> graph, int m, int n){
      int s = graph.size();
      for(int i=0; i < s; i++){
            if(graph[i] == X) cout << "X ";
            else if(graph[i] == G) cout << "G ";
            else cout << graph[i] << " ";
            if(i % m == (m-1))
                  cout << endl;
      }
}

vector<int> BFS(vector<int> graph, int s, int m, int n){
      // s: start node.
      // n: number of nodes in the graph.
      queue<int> q;
      vector<int> res;
      q.push(s);
      res.push_back(s);
      for(int i=0; !q.empty(); i++){
      int node = q.front();
```

```
        q.pop();
        for(int adj: get_adjs(graph, node, m, n)){
            if(find(res.begin(), res.end(), adj) == res.end()){
                q.push(adj);
                if(graph[adj] == G)
                return res;
                res.push_back(adj);
            }
        }
        return res;
}

int main(void){
        vector<int> graph = {
                    1, 1, X, X, X,
                    X, 1, X, G, X,
                    X, 1, X, 1, X,
                    X, 1, 1, 1, X
                    };
        cout << "Initial Graph:" << endl;
        print_graph(graph, 5, 4);
        vector<int> path = BFS(graph, 0, 4, 5);
        cout << "\nPath found: ";
        print_path(path, " -> ", 5, 4);
        cout << endl;
        return 0;
}

/*

OUTPUT:
Initial Graph:
1 1 X X X
X 1 X G X
X 1 X 1 X
X 1 1 1 X

Path found: (0, 0) -> (0, 1) -> (1, 1) -> (2, 1) -> (3, 1) -> (3, 2)
-> (3, 3) -> (2, 3)

*/
```

DFS:

```
#include<bits/stdc++.h>
#define X -1
#define G 100
using namespace::std;
vector<int> get_adjs(vector<int> graph, int node, int m, int n){
      vector<int> ans;
      if((node >= n) && (graph[node - n] != X))
      ans.push_back(node-n);           // up
      if((node + n < m * n) && (graph[node + n] != X))
      ans.push_back(node + n); // down
      if(((node % n) != 0)  && (graph[node - 1] != X))
      ans.push_back(node - 1);         // left
      if(((node + 1) % m != 0) && (graph[node + 1] != X))
      ans.push_back(node + 1); // right
      return ans;
}

void print_path(vector<int> path, string delimiter, int m, int n){
      int i;
      for(i=0; i < path.size() - 1; i++)
      printf("(%d, %d) -> ", path[i] / m, path[i] % m);
      printf("(%d, %d)", path[i] / m, path[i] % m);
}

void print_graph(vector<int> graph, int m, int n){
      int s = graph.size();
      for(int i=0; i < s; i++){
            if(graph[i] == X) cout << "X ";
            else if(graph[i] == G) cout << "G ";
            else cout << graph[i] << " ";
            if(i % m == (m-1))
                  cout << endl;
      }
}

vector<int> DFS(vector<int> graph, vector<bool> visited, vector<int>
res, int s, int m, int n){
      // s: start node.
      // n: # cols in the graph.
      // m: # rows in the graph
      visited[s] = true;

      vector<int> res1;
      res.push_back(s);
```

```
      for(int adj: get_adjs(graph, s, m, n)){
      if(!visited[adj]){
            if(graph[adj] == G)
                  return res;
            res1 = DFS(graph, visited, res, adj, m, n);
      }
      }
      return res1;
}

int main(void){
      vector<int> graph = {
                  1, 1, X, X, X,
                  X, 1, X, G, X,
                  X, 1, X, 1, X,
                  X, 1, 1, 1, X
                  };
      int m=4, n=5;
      cout << "Initial Graph:" << endl;
      print_graph(graph, n, m);
      vector<bool> visited(m * n, false);
      vector<int> res;
      vector<int> path = DFS(graph, visited, res, 0, m, n);
      cout << "\nPath found: ";
      print_path(path, " -> ", n, m);
      cout << endl;
      return 0;
}
```

OUTPUT:

```
Initial Graph:
1 1 X X X
X 1 X G X
X 1 X 1 X
X 1 1 1 X

Path found: (0, 0) -> (0, 1) -> (1, 1) -> (2, 1) -> (3, 1) -> (3, 2)
-> (3, 3) -> (2, 3)
```