include <bits/stdc++.h>

using namespace std;

/\*

\* A directed graph using adjacency list representation;

\* every vertex holds a list of all neighbouring vertices

\* that can be reached from it.

\*/

class Graph {

public:

// Construct the graph given the number of vertices...

Graph(int vertices);

// Specify an edge between two vertices

void add\_edge(int src, int dst);

// Call the recursive helper function to count all the

// paths

int count\_paths(int src, int dst, int vertices);

private:

int m\_vertices;

list<int>\* m\_neighbours;

void path\_counter(int src, int dst, int& path\_count,

vector<bool>& visited);

};

Graph::Graph(int vertices)

{

m\_vertices = vertices; // unused!!

/\* An array of linked lists - each element corresponds

to a vertex and will hold a list of neighbours...\*/

m\_neighbours = new list<int>[vertices];

}

void Graph::add\_edge(int src, int dst)

{

m\_neighbours[src].push\_back(dst);

}

int Graph::count\_paths(int src, int dst, int vertices)

{

int path\_count = 0;

vector<bool> visited(vertices, false);

path\_counter(src, dst, path\_count, visited);

return path\_count;

}

/\*

\* A recursive function that counts all paths from src to

\* dst. Keep track of the count in the parameter.

\*/

void Graph::path\_counter(int src, int dst, int& path\_count,

vector<bool>& visited)

{

// If we've reached the destination, then increment

// count...

visited[src] = true;

if (src == dst) {

path\_count++;

}

// ...otherwise recurse into all neighbours...

else {

for (auto neighbour : m\_neighbours[src]) {

if (!visited[neighbour])

path\_counter(neighbour, dst, path\_count,

visited);

}

}

visited[src] = false;

}

// Driver code

int main()

{

// Create a graph given in the above diagram - see link

Graph g(5);

g.add\_edge(0, 1);

g.add\_edge(0, 2);

g.add\_edge(0, 4);

g.add\_edge(1, 3);

g.add\_edge(1, 4);

g.add\_edge(2, 3);

g.add\_edge(2, 1);

g.add\_edge(3, 2);

// Function call

cout << g.count\_paths(0, 4, 5);

return 0;

}