Let a directed or undirected graph without loops and multiple edges. Want to check whether it is acyclic, and if not, then find any cycle.

Solve this problem by using depth-first search in O (M).

Algorithm

We carry out a series of searches in depth in the graph. Ie from each vertex, which we had never come, run dfs that at the entrance to the top will paint it gray, and on leaving - in black. And if dfs tries to go into a gray top, it means that we have found a cycle (if the graph is undirected, the cases where the depth-first search of some vertex tries to go to the parent are not considered).

The cycle itself can be restored over an array of ancestors.

Implementation

Here is an implementation for the case of a directed graph.

```
int n;
vector <vector <int>> g;
vector <char> cl;
vector <int> p;
int cycle_st, cycle_end;

bool dfs (int v) {
	cl [v] = 1;
	for (size_t i = 0; i <g [v]. size (); + + i) {
		int to = g [v] [i];
		if (cl [to] == 0) {
			p [to] = v;
			if (dfs (to)) return true;
		}
		else if (cl [to] == 1) {
			cycle_end = v;
			cycle_st = to;
			return true;
		}
	}
	cl [v] = 2;
	return false;
}
```

```
int main () {
	... Reading the graph ...

	p.assign (n, -1);
	cl.assign (n, 0);
	cycle_st = -1;
	for (int i = 0; i <n; + + i)
		if (dfs (i))
			break;

	if (cycle_st == -1)
		puts ("Acyclic");
	else {
		puts ("Cyclic");
		vector <int> cycle;
		cycle.push_back (cycle_st);
		for (int v = cycle_end; v! = cycle_st; v = p [v])
			cycle.push_back (v);
		cycle.push_back (cycle_st);
		reverse (cycle.begin (), cycle.end ());
		for (size_t i = 0; i <cycle.size (); + + i)
			printf ("% d", cycle [i] +1);
	}
}
```