Breadth-first search (bypassing width, breadth-first search) - this is one of the basic graph algorithms.

As a result, breadth-first search is the shortest path length in the unweighted graph, ie a path containing the smallest number of ribs.

The algorithm works for $O(n+m)$where n- number of vertices m- number of edges.

Description of the algorithm

Algorithm is applied to the input of the given graph (unweighted), and number of starting vertex s. Graph can be as focused and undirected, for the algorithm is not important.

The algorithm can be understood as a process of "ignition" of the graph: at zero step ignite only the tip s. At each step the fire already burning with each vertex spreads to all its neighbors, ie in one iteration of the algorithm is expanding "ring of fire" in width by one (hence the name of the algorithm).

More precisely, it is represented as follows. Create queue q, which will be placed in special tops and manages a boolean array $\text{used}[]$in which each vertex will celebrate, she had already lit or not (or in other words, whether she had).

Initially, only the tip fits all s, and $\text{used}[s] = \text{true}$, as for all the other vertices $\text{used}[] = \text{false}$. Then the algorithm is a loop: while queue is not empty, get out of her head one vertex, see all the edges emanating from that vertex, and if any of the vertices visited still off, then set them on fire and put in the queue.

As a result, when the queue is empty, bypassing wide bypass all accessible from s the top, and each comes to the shortest path. You can also calculate the length of the shortest paths (which just need to have an array of lengths of paths $d[]$), and compact to save enough information to restore all of the shortest paths (you have to make an array of "ancestors" $p[]$, in which each vertex store node number at which we got to the top of this).

Implementation

Implement the above algorithm in the language C + +.

Input:

```
vector < vector<int> > g; // граф
int n; // число вершин
```

```
int s; // стартовая вершина (вершины везде нумеруются с нуля)

// чтение графа
...
```

Circumvention itself:

```
queue<int> q;
q.push (s);
vector<bool> used (n);
vector<int> d (n), p (n);
used[s] = true;
p[s] = -1;
while (!q.empty()) {
	int v = q.front();
	q.pop();
	for (size_t i=0; i<g[v].size(); ++i) {
		int to = g[v][i];
		if (!used[to]) {
			used[to] = true;
			q.push (to);
			d[to] = d[v] + 1;
			p[to] = v;
		}
	}
}
```

If we now want to restore and display the shortest path to some vertex to, it can be done as follows:

```
if (!used[to])
	cout << "No path!";
else {
	vector<int> path;
	for (int v=to; v!=-1; v=p[v])
		path.push_back (v);
	reverse (path.begin(), path.end());
	cout << "Path: ";
	for (size_t i=0; i<path.size(); ++i)
		cout << path[i] + 1 << " ";
}
```

Application of the algorithm

- Find **the shortest path** in unweighted graph.
- Search **the connected components** in the graph for $O(n+m)$.
- To do this, we simply run wide detour from each vertex except vertices remaining visits ($\text{used} = \text{true}$) after previous launches. Thus, we perform normal start in width from each vertex, but do not reset each time the array $\text{used}[]$, whereby every time we'll get a new connected component, and the total time of the algorithm will continue $O(n+m)$(such multiple runs on a graph traversal without zeroing array usedcalled a series of detours in width).
- Finding a solution to a problem (the game) **with the smallest number of moves** , if each state of the system can be represented by a vertex of the graph, and the transitions from one state to another - edges of the graph.
- A classic example - a game where the robot moves along the field, while it can move boxes that are on the same field, and requires the least number of moves required to move boxes in position. Solved this preorder traversal through the graph, where the state (top) is a set of coordinates: the coordinates of the robot, and the coordinates of all the boxes.
- Finding the shortest path in a **0-1-graph** (ie, weighted graph, but with weights equal only 0 or 1) is sufficient to slightly modify the breadth-first search, if the current edge of zero weight, and there is some improvement in the distance to the top, then add this summit is not the end but the beginning of the queue.
- Finding **the shortest cycle** in a directed unweighted graph: search in width from each vertex; once while crawling we try to go from the current vertex on some edge to already visited vertices, then it means that we have found the shortest cycle and stop bypassing wide, and found among all such cycles (one from each run bypass) choose the shortest.
- Find all edges that lie **on any shortest path** between a given pair of vertices (a,b). To do this, run the breadth-first search 2: from a, to and from b. Denoted by $d_a[]$an array of the shortest distances resulting from the first crawl, and after $d_b[]$- in a second bypass. Now for any edge $(u,v)$is easy to check whether it is in any shortest path: the criterion is a condition $d_a[u]+1+d_b[v]=d_a[b]$.
- Find all vertices lying **on some shortest path** between a given pair of vertices (a,b). To do this, run the breadth-first search 2: from a, to and from b.Denoted by $d_a[]$an array of the shortest distances resulting from the first crawl, and after $d_b[]$- in a second bypass. Now for any vertex $v$is easy to check whether it is in any shortest path: the criterion is a condition $d_a[v]+d_b[v]=d_a[b]$.

- Find **the shortest path even** in a graph (ie, the path of even length). For this we need to build an auxiliary graph, whose vertices are states $(v, c)$where v - number of current peaks $c = 0 \dots 1$- the current parity. Each edge (a, b) of the original graph in this new column will turn into two edges $((u, 0), (v, 1))$ and$((u, 1), (v, 0))$. After that, on this graph should preorder traversal to find the shortest path from the starting vertex to the terminal, with parity of 0.

Problem in online judges

List of tasks that can be taken using a wide detour:

- SGU # 213 **"Strong Defence"** [Difficulty: Medium]