**BFS**

Обход:

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 (int 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;

}

}

}

*Если теперь надо восстановить и вывести кратчайший путь до какой-то вершины , это можно сделать следующим образом:*

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 (int i=0; i<path.size(); ++i)

cout<< path[i]+ 1<< " ";

}

**DFS**

*Наиболее простая реализация:*

vector < vector<int> > g; // граф

int n; // число вершин

vector<char> used;

void dfs (int v) {

used[v] = true;

for (vector<int>::iterator i=g[v].begin(); i!=g[v].end(); ++i)

if (!used[\*i])

dfs (\*i);

}

*Более подробная реализация:*

vector < vector<int> > g; // граф

int n; // число вершин

vector<int> color; // цвет вершины(0, 1, или2)

vector<int> time\_in, time\_out; // "времена" захода и выхода из вершины

int dfs\_timer = 0; // "таймер" для определения времён

void dfs (int v) {

time\_in[v] = dfs\_timer++;

color[v] = 1;

for (vector<int>::iterator i=g[v].begin(); i!=g[v].end(); ++i)

if (color[\*i] == 0)

dfs (\*i);

color[v] = 2;

time\_out[v] = dfs\_timer++;

}