added: 10 Jun 2008 22:14
Edited: 10 Jun 2008 22:15

Finding Euler path in O (M)

Contents [hide]

Euler path - a path in the graph, passing through all of his ribs. Euler tour - is the Euler path is a cycle.

The challenge is to find an Euler path in **an undirected multigraph with loops** .

Algorithm

First check whether there is an Euler path. Then find all simple cycles and combine them into one - it will be an Euler tour. If the graph is such that the Euler path is not a cycle, then, add the missing edge, find an Euler tour, then remove the extra edge.

To test whether there is an Euler path, you must use the following theorem. Euler cycle exists if and only if the degrees of all vertices is even. Euler path exists if and only if the number of vertices with odd degree is two (or zero, in the case of the existence of an Euler tour).

In addition, of course, the graph must be sufficiently connected (ie if you remove all isolated vertices, it is supposed to be a connected graph).

View all cycles and will combine them a recursive procedure:

```
procedure FindEulerPath (V)
        1 to iterate over all the edges emanating from the vertex V;
                each such edge is removed from the graph, and
                FindEulerPath call from the second end of the rib;
        2 add a vertex V in response.
```

The complexity of this algorithm is obviously linear in the number of edges.

But the same algorithm can be written in **a non-recursive** version:

```
stack St;
in St we place any vertex (starting vertex);
while St is not empty
        Let V - value at the top of St;
        if the power (V) = 0, then
                add V to the response;
                remove V from the top of St;
        otherwise
                find any edge going from V;
                remove it from the graph;
                the other end of the rib put in a St;
```

It is easy to verify the equivalence of these two forms of the algorithm. However, the second shape obviously is faster, with a code is not anymore.

The problem of the domino

We give here a classic problem in the Euler tour - the problem of the domino.

There are N dominoes are known at both ends domino recorded one number (typically 1 to 6, but in this case is not important). You want to publish all the dominoes in a row so that any two adjacent dominoes numbers written on their common side match. Dominoes are allowed to turn.

Reformulate the problem. Let the numbers recorded on donimoshkah - vertices of the graph, and the dominoes - the edges of the graph (each domino with numbers (a, b) - is an edge (a, b) and (b, a)). Then our problem **reduces to** the problem of finding **an Euler path** in this graph.

Implementation

The following program searches for and displays an Euler cycle or path in the graph, or displays -1 if it does not exist.

First, the program checks the degrees of the vertices if the vertices with odd degree is not, then the graph has an Euler tour if there are two vertices with odd degree, then the graph is only an Euler path (Euler cycle is not), and if such heights greater than 2, then graph no Euler cycle or Euler path. To find an Euler path (not cycle), proceed as follows: if V1 and V2 - the two vertices of odd degree, then just add an edge (V1, V2), the resulting graph will find an Euler tour (he obviously will exist), and then remove from the response of "dummy" edge (V1, V2). Euler tour will look exactly as described above (non-recursive version), and at the same time at the end of this algorithm check was connected graph or not (if the graph is not connected, then at the end of the algorithm in the column will remain some of the edges, and in this case, we want to output 1). Finally, the program takes into account that in the graph can be isolated vertices.

```
int main () {

        int n;
        vector <vector <int>> g (n, vector <int> (n));
        Reading ... in the adjacency matrix of the graph ...
```

```
	vector <int> deg (n);
	for (int i = 0; i <n; ++ i)
		for (int j = 0; j <n; ++ j)
			deg [i] + = g [i] [j];

	int first = 0;
	while (! deg [first]) ++ first;

	int v1 = -1, v2 = -1;
	bool bad = false;
	for (int i = 0; i <n; ++ i)
		if (deg [i] & 1)
			if (v1 == -1)
				v1 = i;
			else if (v2 == -1)
				v2 = i;
			else
				bad = true;

	if (v1! = -1)
		++ G [v1] [v2], ++ g [v2] [v1];

	stack <int> st;
	st.push (first);
	vector <int> res;
	while (! st.empty ())
	{
		int v = st.top ();
		int i;
		for (i = 0; i <n; ++ i)
			if (g [v] [i])
				break;
		if (i == n)
		{
			res.push_back (v);
			st.pop ();
		}
		else
		{
			--g [v] [i];
			--g [i] [v];
			st.push (i);
		}
	}

	if (v1! = -1)
		for (size_t i = 0; i + 1 <res.size (); ++ i)
			if (res [i] == v1 && res [i + 1] == v2 || res [i] == v2 && res [i + 1] == v1)
			{
				vector <int> res2;
				for (size_t j = i + 1; j <res.size (); ++ j)
					res2.push_back (res [j]);
				for (size_t j = 1; j <= i; ++ j)
					res2.push_back (res [j]);
				res = res2;
				break;
			}

	for (int i = 0; i <n; ++ i)
		for (int j = 0; j <n; ++ j)
			if (g [i] [j])
				bad = true;

	if (bad)
		puts ("-1");
	else
		for (size_t i = 0; i <res.size (); ++ i)
			printf ("% d", res [i] +1);

}
```

Ramil6085 • 2 года назад

1) Определение эйлерова пути в начале статьи не совсем верно. Некоторые могут подумать, что эйлеров путь - это вообще всякий путь проходящий по всем ребрам, а это не так. Эйлеров путь - это путь, проходящий по всем рёбрам заданного графа и притом только по одному разу.
2) Обычно в статьях Вы приводите доказательство некоторых не совсем очевидных вещей. Лично мне здесь это очень нравится :) Здесь же предлагаю все-таки написать (хотя бы вкратце) доказательство утверждения "Эйлеров цикл существует тогда и только тогда, когда степени всех вершин
чётны. Эйлеров путь существует тогда и только тогда, когда количество
вершин с нечётными степенями равно двум (или нулю, в случае
существования эйлерова цикла)". Там ведь не слишком большое доказательство, но все-таки многим непросвещенным интересно было бы почитать, я думаю. Конечно, доказательство можно найти и в интернете, но когда все в одном месте - это всегда лучше.
3) Что делать если граф ориентирован? Многие ли изменится в решении и в критерии существования цикла? Как решать задачу о прохождении всех ребер по одному разу в этом случае? Заранее спасибо.

25 • Ответить • Поделиться ›

Iama → Ramil6085 • 2 года назад

Присоединюсь к вопросу насчет орграфа.

8 • Ответить • Поделиться ›

Break-Neck → Iama • год назад

В случае орграфа достаточно, чтобы все концы всех ребер принадлежали одной компоненте сильной связности, для цикла степень входа и выхода всех вершин четные, для пути может быть одна вершина с на 1 большей степенью выхода (начало) и на 1 большей степенью входа (конец); доказательства аналогичны неориентированному графу.

3 • Ответить • Поделиться ›

Break-Neck → Break-Neck • год назад

Извиняюсь, достаточно, чтобы степень входа во всех вершинах была равна степени исхода.

4 • Ответить • Поделиться ›

Павел → Ramil6085 • 9 месяцев назад

Доказательство абсолютно неинтересно. Главное, что предложили решение и оно работает

• Ответить • Поделиться ›

Андрей → Павел • 8 месяцев назад

Странная логика. Вы всегда слепо верите во все, что вам говорят?

• Ответить • Поделиться ›

Дмитрий • год назад

КАК это творение запустить??

• Ответить • Поделиться ›