Алгоритм 3

Задача поиска Эйлерова цикла в графе

Эйлеров путь - это путь в графе, проходящий через все его рёбра. Эйлеров ЦИКЛ - это эйлеров путь, являющийся циклом.

Сначала проверим, существует ли эйлеров путь. Затем найдём все простые циклы и объединим их в один - это и будет эйлеровым циклом.

Чтобы проверить, существует ли эйлеров путь, нужно воспользоваться следующей теоремой. 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).

Кроме того, конечно, граф должен быть достаточно связным (т.е. если удалить из него все изолированные вершины, то должен получиться связный граф).

procedure FindEulerPath (V)

1. перебрать все рёбра, выходящие из вершины V;

каждое такое ребро удаляем из графа, и

вызываем FindEulerPath из второго конца этого ребра;

2. добавляем вершину V в ответ.

Приведенная ниже программа ищет и выводит эйлеров цикл или путь в графе, или выводит -1, если его не существует.

Сначала программа проверяет степени вершин: если вершин с нечётной степенью нет, то в графе есть эйлеров цикл, если есть 2 вершины с нечётной степенью, то в графе есть только эйлеров путь (эйлерова цикла нет), если же таких вершин больше 2, то в графе нет ни эйлерова цикла, ни эйлерова пути. Чтобы найти эйлеров путь (не цикл), поступим таким образом: если V1 и V2 - это две вершины нечётной степени, то просто добавим ребро (V1,V2), в полученном графе найдём эйлеров цикл (он, очевидно, будет существовать), а затем удалим из ответа "фиктивное" ребро (V1,V2). Эйлеров цикл будем искать в точности так, как описано выше (нерекурсивной версией), и заодно по окончании этого алгоритма проверим, связный был граф или нет (если граф был не связный, то по окончании работы алгоритма в графе останутся некоторые рёбра, и в этом случае нам надо вывести -1). Наконец, программа учитывает, что в графе могут быть изолированные вершины.

int main() {

int n;

vector < vector<int> > g (n, vector<int> (n));

... чтение графа в матрицу смежности ...

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);

}