Given a directed weighted graph $G$with $n$vertices and $m$edges. You want to find in it any **cycle of negative weight** , if any.

When another formulation of the problem - you want to find **all pairs of vertices** such that there is a path between any number of small length.

These two variants of the problem can be conveniently solved by different algorithms so below will be considered both of them.

One of the common "life" performances of this problem - the following: there are **exchange rates** ie courses transfer from one currency to another. You want to know whether a certain sequence of exchanges to benefit, ie started with one unit of a currency, receive as a result of more than one unit of the same currency.

The decision by the algorithm of Bellman-Ford

Bellman-Ford algorithm allows to check the presence or absence of negative weight cycle in the graph, and if available - to find one of these cycles.

We will not go into details here (which are described in the article on the algorithm of Bellman-Ford), and give only the result - how the algorithm works.

Done $n$iterations of Bellman-Ford, and if at the last iteration has been no change - is a negative cycle in the graph no. Otherwise, take the vertex, the distance to which the change, and we will go from her ancestor, while not going into the cycle; this cycle will be the desired negative cycle.

Implementation :

```
struct edge {
	int a, b, cost;
};

int n, m;
vector<edge> e;
const int INF = 1000000000;

void solve() {
	vector<int> d (n);
	vector<int> p (n, -1);
	int x;
	for (int i=0; i<n; ++i) {
		x = -1;
		for (int j=0; j<m; ++j)
```

```
                if (d[e[j].b] > d[e[j].a] + e[j].cost) {
                    d[e[j].b] = max (-INF, d[e[j].a] +
e[j].cost);
                    p[e[j].b] = e[j].a;
                    x = e[j].b;
                }
        }

        if (x == -1)
            cout << "No negative cycle found.";
        else {
            int y = x;
            for (int i=0; i<n; ++i)
                y = p[y];

            vector<int> path;
            for (int cur=y; ; cur=p[cur]) {
                path.push_back (cur);
                if (cur == y && path.size() > 1)  break;
            }
            reverse (path.begin(), path.end());

            cout << "Negative cycle: ";
            for (size_t i=0; i<path.size(); ++i)
                cout << path[i] << ' ';
        }
}
```

The decision by the algorithm Floyd-Uorshella

Floyd's Algorithm-Uorshella solves the second formulation of the problem - when you have to find all pairs of vertices (i, j)between which the shortest path does not exist (ie, it has an infinitely small amount).

Again, more detailed explanations contained in the description of the algorithm Floyd-Uorshella , and here we present only the results.

After Floyd's algorithm-Uorshella work for the input graph, iterate over all pairs of vertices (i, j), and for each such pair check infinitesimal shortest path from $i$to $j$or not. To do this, let's look over the top of the third t, and if it turned out to be $d[t][t] < 0$(ie, it lies in a cycle of negative weight), and she is reachable from $i$and out of the attainable j

- the path (i, j)can have an infinitesimal length.

Implementation :

```
for (int i=0; i<n; ++i)
     for (int j=0; j<n; ++j)
          for (int t=0; t<n; ++t)
               if (d[i][t] < INF && d[t][t] < 0 && d[t][j] < INF)
                    d[i][j] = -INF;
```

Problem in online judges

List of tasks that require search cycle of negative weight:

- UVA # 499 **"Wormholes"** [Difficulty: Easy]
- UVA # 104 **"Arbitrage"** [Difficulty: Medium]
- UVA # 10557 **"XYZZY"** [Difficulty: Medium]