Задача 3. Списки инцидентности

Источник:	базовая II
Имя входного файла:	--
Имя выходного файла:	--
Ограничение по времени:	3 секунды*
Ограничение по памяти:	128 мегабайт

Дан неориентированный граф из N вершин и M рёбер. Рёбра заданы одним списком, каждое ребро задаётся парой концевых вершин и весом. Возможны кратные рёбра и петли. Требуется за один проход по списку рёбер составить списки инцидентности для всех вершин.

От вас требуется отправить один файл `sol.c`. В нём следует подключить хедер `sol.h` следующего содержания:

```
#ifndef EDGELISTS_SOL_579846984654
#define EDGELISTS_SOL_579846984654

typedef struct {
    int from, to;
    int weight;
} Edge;

//=============================================
//call these functions to get the graph data:

//returns N -- the number of vertices in the graph
int getVerticesCount();
//reads the next edge from the edge list
//if there is next edge, returns 1 and stores that edge to *oEdge
//if there is no next edge, returns 0 without touching pointer
int readEdge(Edge *oEdge);

//=============================================
//implement these functions in your solution:

//this function is called first to initialize graph
//you are expected to read graph here and fill internal data structures
void init();

//returns number of edges indicent to vertex iVertex
int getEdgesCount(int iVertex);
//returns iIndex-th edge incident to the vertex iVertex
//it must have .from == Vertex and .to denoting the other end
Edge getIncidentEdge(int iVertex, int iIndex);

#endif
```

В `sol.c` нужно реализовать функции `init`, `getEdgesCount` и `getIncidentEdge`. Функции `getVerticesCount`, `readEdge` будут реализованы в другой единице трансляции и слинкованы вместе с вашим кодом. Для тестирования "у себя" вам желательно тоже их где-то реализовать, но отправлять в систему их **не** нужно.

Ограничения на размер графа: $N \leqslant 3 \cdot 10^5$, $M \leqslant 3 \cdot 10^5$. В данной задаче всё нумеруется

начиная с нуля.

Гарантируется, что при тестировании функция `init` будет вызвана один раз до всех остальных вызовов. Остальные две функции могут вызываться в произвольном порядке и объёме. Гарантируется, что все вызовы корректны: `iVertex` лежит в пределах от 0 до $(N-1)$, а `iIndex` в пределах от 0 до того, что вернула ваша функция `getEdgesCount`, минус один. Всего количество вызовов `getEdgesCount` и `getIncidentEdge` не превышает 10^6.

Обратите внимание, что каждое ребро должно входить в список инцидентности обеих своих концевых вершин. Петля должна входить дважды в список рёбер своей вершины. Порядок рёбер в каждом списке инцидентности значения не имеет.

Пример

Ниже схематично приведён порядок вызовов функций на первом тесте. Для кратности `getIncidentEdge` обозначается как `getIE`.

--	--

`5 = getVerticesCount()`
`1 = readEdge() : [0, 2, 178]`
`1 = readEdge() : [3, 4, 207]`
`1 = readEdge() : [1, 1, 356]`
`1 = readEdge() : [2, 0, 101]`
`1 = readEdge() : [4, 1, 286]`
`1 = readEdge() : [4, 1, 213]`
`0 = readEdge()`
`0 = readEdge()`
	`init()`
	`2 = getEdgesCount(0)`
	`[0, 2, 178] = getIE(0, 0)`
	`[0, 2, 101] = getIE(0, 1)`
	`4 = getEdgesCount(1)`
	`[1, 1, 356] = getIE(1, 0)`
	`[1, 4, 286] = getIE(1, 3)`
	`[1, 1, 356] = getIE(1, 2)`
	`[1, 4, 213] = getIE(1, 1)`
	`2 = getEdgesCount(2)`
	`[2, 0, 178] = getIE(2, 0)`
	`[2, 0, 101] = getIE(2, 1)`
	`1 = getEdgesCount(3)`
	`[3, 4, 207] = getIE(3, 0)`
	`3 = getEdgesCount(4)`
	`[4, 3, 207] = getIE(4, 2)`
	`[4, 1, 286] = getIE(4, 0)`
	`[4, 1, 213] = getIE(4, 1)`