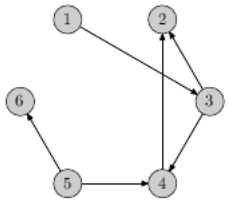
Графы: поиск достижимых вершин

На рисунке изображён ориентированный граф, на примере которого мы будем разбирать алгоритмы этого раздела. Во всех задачах этого раздела мы будем считать, что вершины графа — «города», рёбра — соединяющие их «дороги», имеющие направление (т. к. граф ориентирован).



36	Смежные вершины	2 балла

Выведите номера вершин, в которые можно перейти из вершины с данным номером за один шаг (т. е. пройдя по одному ребру). Пример: в графе на рисунке из вершины 1 за один шаг можно попать только в вершину 3, а из вершины 5 — в вершины 4 и 6.

37	Очередь	4 балла

Структура данных «очередь» позволяет запоминать некоторые данные (например, числа) и выдавать их в том же порядке, в котором они запоминались. Это напоминает настоящую очередь (например, в магазине), когда покупатели обслуживаются в том порядке, в котором приходят.
С точки зрения программирования очередь (англ. queue [kju:]) — некий «чёрный ящик», с которым можно делать три операции:

1. положить в него число (`enqueue`);
2. забирать из него очередное число (`dequeue`), при этом числа «достаются из ящика» в том же порядке, в котором они были туда положены;
3. проверить, пуст ли ящик (`empty`).

В этом задании вам нужно написать эти три функции (`enqueue`, `dequeue`, `empty`) для работы с очередью из целых чисел. Будем хранить числа в массиве `int Q[N]`. Работу с очередью организуем следующим образом:

переменная `int w` (от слова write) будет хранить номер элемента массива, в который функция `enqueue` запишет следующее поступившее в очередь число, а переменная `int r` (read) — номер элемента, из которого функция `dequeue` достанет очередное число, которое должно будет покинуть очередь. Таким образом, функция `enqueue` будет увеличивать переменную `w`, а функция `dequeue` — увеличивать переменную `r`.
В следующем примере показано состояние массива и значения переменных `r` и `w` после нескольких вызовов `enqueue` и `dequeue`. Текущее содержимое очереди на каждом шаге — в квадратных скобках.

Состояние до	**Команда**	**Состояние после**

`  w`
`[]_  _  _  _  _`
`  r`
	`enqueue(1);`	`     w`
		`[ 1 ]_  _  _  _`
		`  r`

`     w`
`[ 1 ]_  _  _  _`
`  r`
	`enqueue(2);`	`        w`
		`[ 1  2 ]_  _  _`
		`  r`

`        w`
`[ 1  2 ]_  _  _`
`  r`
	`x = dequeue();`
	(x равно 1)
		`        w`
		`  1[ 2 ]_  _  _`
		`     r`

`        w`
`  1[ 2 ]_  _  _`
`     r`
	`enqueue(3);`	`           w`
		`  1[ 2  3 ]_  _`
		`     r`

`           w`
`  1[ 2  3 ]_  _`
`     r`
	`y = dequeue();`
	(y равно 2)
		`           w`
		`  1  2[ 3 ]_  _`
		`        r`

`           w`
`  1  2[ 3 ]_  _`
`        r`
	`z = dequeue();`
	(z равно 3)
		`           w`
		`  1  2  3[]_  _`
		`           r`

`           w`
`  1  2  3[]_  _`
`           r`
	`enqueue(4);`	`              w`
		`  1  2  3[ 4 ]_`
		`           r`

Стоит заметить, что сами функции `enqueue`, `dequeue`, `empty` получатся намного короче, чем условие этой задачи.

38	Обход в ширину	7 баллов

Рассмотрим алгоритм перебора вершин графа, начиная с вершины номер 1. Нам понадобится очередь (функции `enqueue`, `dequeue` и `empty` из предыдущей задачи) и одномерный массив для отмечания пройденных вершин.
Сначала поместим вершину 1 в очередь и пометим её как пройденную. Затем будем повторять следующие действия, пока очередь не пуста:

1. взять номер следующей вершины из очереди (эта вершина станет текущей);
2. напечатать этот номер вершины;
3. перебрать все вершины графа; если вершина не помечена и в неё можно пройти из текущей — пометить её как пройденную и поместить в очередь.

39	Обход в глубину	6 баллов

Рассмотрим ещё один алгоритм перебора вершин графа, начиная с вершины номер 1. Реализуем функцию «посетить вершину номер `k`», которая выполнит следующие действия:

1. напечатает номер посещённой вершины (`k`);
2. отметит эту вершину как пройденную;
3. переберёт все вершины графа; если вершина `i` не помечена и в неё можно пройти из вершины `k` — вызовет функцию «посетить вершину номер `i`».