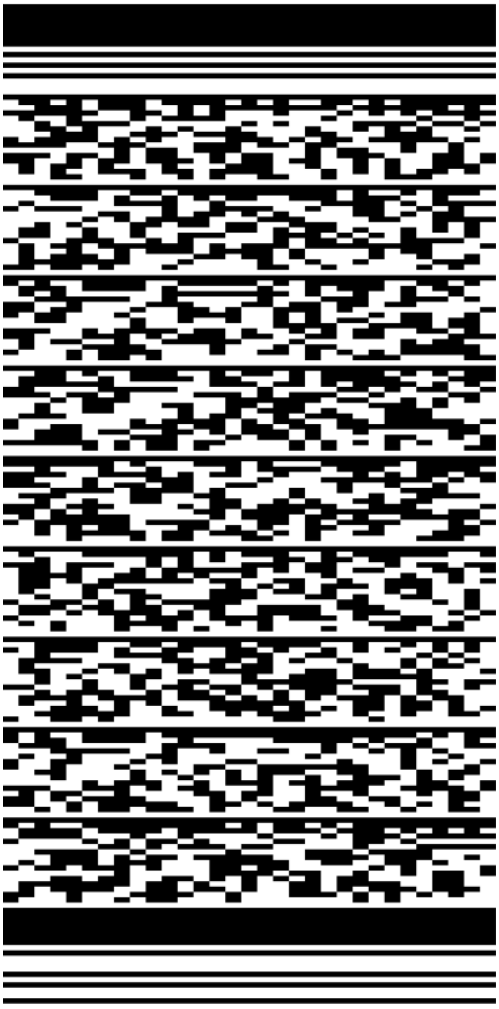
Задача 4 второго дня: Сохранение информации



Служба доставки «Скедеф» перевозит по воздуху посылки между несколькими городами. Какие-то из этих городов являются *узлами* и в них установлены специальные устройства для обработки посылок. Каждый из самолётов компании «Скедеф» летает туда и обратно между одной из пар городов, и перевозит посылки в любом из двух направлений, если это требуется.

Чтобы перевезти посылку из одного города в другой, её необходимо провезти, используя последовательные перелёты через города, при этом на каждом перелёте посылка перевозится между парой городов, обслуживаемой одним из самолётов. Более того, в этой последовательности городов должен быть хотя бы один узел.

Для облегчения подборки маршрута, компания «Скедеф» хочет закодировать длины кратчайших последовательностей перелётов из каждого города в каждый узел и записать коды на метке каждой из посылок. Длина кратчайшей последовательности перелётов из узла в себя равна нулю. Очевидно, что требуется компактное представление этой информации.

Вам необходимо реализовать две процедуры — **encode(N,H,P,A,B)** и **decode(N,H)**. Здесь **N** — количество городов, **H** — количество узлов. Города пронумерованы от 0 до (N-1), а узлами являются города с номерами от 0 до (H-1). Кроме того, $N \leq 1000$ и $H \leq 36$. **P** — количество пар городов, между которыми летает самолёт. Все пары городов различны как неупорядоченные пары. **A** и **B** — это массивы размера P, и (A[0], B[0]) — это первая пара городов, между которыми летает самолёт, (A[1], B[1]) — вторая пара и т.д.

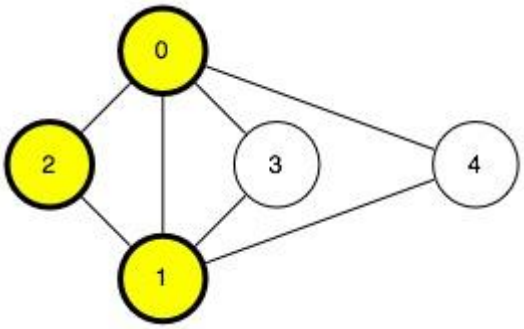
Процедура **encode** должна построить такую последовательность битов, по которой процедура **decode** сможет восстановить количество перелётов, необходимых для перевозки посылки от каждого из городов до каждого из узлов. Процедура **encode** будет передавать последовательность битов системе оценивания с помощью последовательных вызовов процедуры **encode_bit(b)**, где **b** равно 0 или 1. Процедура **decode** будет получать последовательность битов от системы оценивания с помощью вызовов процедуры **decode_bit**. При этом *i*-ый вызов процедуры **decode_bit** будет возвращать значение **b**, переданное *i*-му вызову **encode_bit(b)**. Заметим, что вы должны удостовериться в том, что количество вызовов процедурой **decode** процедуры **decode_bit** не будет превосходить количества сделанных процедурой **encode** вызовов процедуры **encode_bit(b)**.

После декодирования информации о количестве необходимых перелётов, процедура **decode** должна вызвать процедуру **hops(h,c,d)** для каждого из узлов **h** и каждого из

городов **c** (в частности, и для узлов, и в случае **c**=**h**), сообщая минимальное количество необходимых перелётов **d**, которые требуются для перевозки посылки между узлом **h** и городом **c**. Таким образом, вы должны сделать N×H вызовов процедуры **hops(h,c,d)**. Порядок этих вызовов не важен. Гарантируется, что всегда возможно перевезти посылку между любым узлом и любым городом.

*Внимание: процедуры **encode** и **decode** должны взаимодействовать только через описанный выше интерфейс. Совместное использование одних и тех же переменных, доступ к файлам и к сети запрещены. В языках C и C++ вы можете объявить глобальные переменные с использованием* `static`*, чтобы они были доступны и сохраняли значения для соответствующей процедуры, но не были видны из другой процедуры. В языке Pascal вы можете объявлять глобальные переменные в части* `implementation` *вашего модуля.*

Пример



Для примера рассмотрим рисунок справа. На нём показаны пять городов (N=5) связанных семью самолётами (P=7). Города с номерами 0, 1 и 2 являются узлами (H=3). Для доставки посылки между узлом с номером 0 и городом с номером 3 необходим один перелёт, а для доставки посылки между узлом с номером 2 и городом с номером 3 необходимо 2 перелёта. Данные этого примера доступны в файле `grader.in.1`.

Все значения **d**, которые процедура **decode** должна передать процедуре **hops(h,c,d)**, приведены в таблице ниже:

D		**Город номер c**				
		0	**1**	**2**	**3**	**4**
Узел номер h	**0**	0	1	1	1	1
	1	1	0	1	1	1
	2	1	1	0	2	2

Подзадача 1 [25 баллов]

Процедура **encode** должна сделать не более 16 000 000 вызовов процедуры **encode_bit(b)**.

Подзадача 2 [25 баллов]

Процедура **encode** должна сделать не более 360 000 вызовов процедуры **encode_bit(b)**.

Подзадача 3 [25 баллов]

Процедура **encode** должна сделать не более 80 000 вызовов процедуры **encode_bit(b)**.

Подзадача 4 [25 баллов]

Процедура **encode** должна сделать не более 70 000 вызовов процедуры **encode_bit(b)**.

Детали реализации

- Используйте среду программирования и тестирования RunC
- Папка для реализации: `/home/ioi2010-contestant/saveit/` (прототип: saveit.zip)
- Участник должен реализовать:
  - `encoder.c` или `encoder.cpp` или `encoder.pas`
  - `decoder.c` или `decoder.cpp` или `decoder.pas`
- Интерфейс участника:
  - `encoder.h` или `encoder.pas`
  - `decoder.h` или `decoder.pas`

- Интерфейс системы оценивания: `grader.h` и `graderlib.pas`
- Пример системы оценивания: `grader.c` или `grader.cpp` или `grader.pas` *и* `graderlib.pas`
- Пример ввода для оценивания: `grader.in.1` `grader.in.2` и т. д.
  Внимание: Первая строка входного файла содержит `N P H`*. Следующие* `P` *строк содержат пары чисел — номера городов* `A[0] B[0]`*,* `A[1] B[1]` *и т.д. Последующие* `H×N` *строк содержат количество перелётов, требуемое для перевозки посылки из каждого узла в каждый город, включая город, совпадающий с узлом, и остальные узлы. Так, количество перелётов, необходимое для перевозки посылки из узла* `i` *в город* `j` *находится в (*`i×N+j`*+1)-ой из этих строк.*
- Ожидаемый вывод на пример ввода:
  - Если программа корректно решает подзадачу 1, вывод будет содержать `OK 1`
  - Если программа корректно решает подзадачу 2, вывод будет содержать `OK 2`
  - Если программа корректно решает подзадачу 3, вывод будет содержать `OK 3`
  - Если программа корректно решает подзадачу 4, вывод будет содержать `OK 4`

- Компиляция и запуск (из командной строки): `runc grader.c` или `runc grader.cpp` или `runc grader.pas`
- Компиляция и запуск (модуль для gedit): *Control-R*, в момент редактирования файла решения.
- Посылка (из командной строки): `submit grader.c` или `submit grader.cpp` или `submit grader.pas`
- Посылка (модуль для gedit): *Control-J*, в момент редактирования файла решения или системы оценивания.