Тайная Вечеря

Леонардо был очень увлечен работой над Тайной Вечерей - его самой известной фреской. Одной из первоочередных его задач была - какие цвета красок использовать в течение рабочего дня. Ему нужны многие цвета красок, но количество мест на подставке для красок, которой пользовался художник, ограничено. Его помощник, кроме всего прочего, отвечал за подъем подставки для красок, чтобы доставлять их художнику, и за спуск подставки для красок, чтобы ставить краски обратно на соответствующую полку на полу.

В этой задаче вам нужно написать две отдельные программы для помощника. Первая получит инструкции Леонардо (последовательность красок, которые понадобятся Леонардо в течение дня), и создаст *короткую* версию плана, называемую *подсказкой*. В течение дня помощник не имеет доступа к списку запросов Леонардо, а только к той подсказке, которую сгенерировала ваша первая программа. Вторая программа получит вашу подсказку, и будет получать и обрабатывать запросы Леонардо в реальном времени (то есть по одному за раз). Эта программа должна понимать, что означает подсказка и использовать её для того, чтобы всегда делать оптимальные выборы. Далее это разъясняется более детально.

Перемещение красок между полкой на полу и подставкой для красок

Мы будем рассматривать упрощенную модель. Предположим, что есть N красок, пронумерованных от 0 до N - 1, и что каждый день Леонардо запрашивает у помощника новую краску ровно N раз. Пусть C - это последовательность красок, которые запросил Леонардо за день. Мы можем рассматривать C как последовательность из N чисел от 0 до N - 1 включительно. Заметьте, что некоторые из красок могут не встречаться в последовательности C, а некоторые - могут встречаться несколько раз.

Подставка для красок всегда заполнена и содержит ровно K мест для N красок, $K < N$. Изначально подставка для красок содержит краски от 0 до K - 1 включительно.

Помощник обрабатывает запросы Леонардо по одному. Если краска, о которой попросил Леонардо, уже установлена в подставку для красок, то помощник может отдохнуть. В противном случае, ему нужно взять на полке запрошенную художником краску и поставить ее на подставку для красок. Поскольку на подставке для красок не существует свободного места для новой краски, помощник должен выбрать на подставке одну из красок и переставить ее обратно на полку.

Оптимальная стратегия Леонардо

Помощник хочет отдыхать как можно больше раз. Количество запросов, во время которых

помощник может отдыхать, зависит от действий помощника. Точнее, каждый раз, когда помощник убирает краску с подставки, различные варианты выбора краски могут приводить к различным результатам в будущем. Леонардо объяснил ему, как он может добиться своей цели, зная последовательность запросов C. Наилучший выбор для краски, которую нужно убрать с подставки, можно узнать на основании того, какие краски сейчас находятся на подставке и какие запросы будут сделаны. Краску нужно выбрать из тех, которые находятся на подставке согласно следующим правилам:

- Если на подставке есть краска, которая больше не потребуется, то помощник должен забрать такую краску с подставки.
- В противном случае, с подставки надо убрать ту краску, *которая будет использована как можно позже*, то есть для каждой краски на подставке мы находим первое будущее использование. Краска, которую мы переместим на полку должна быть той, которая потребуется последней.

Можно доказать, что при использовании стратегии, предложенной Леонардо, помощник будет отдыхать максимальное число раз.

Пример 1

Пусть N = 4, то есть у нас есть 4 краски, пронумерованных от 0 до 3, и 4 запроса. Пусть последовательность запросов C = (2, 0, 3, 0). Пусть K = 2. Это значит, что у Леонардо есть подставка для красок, которая вмещает две краски. Как было указано выше, подставка для красок изначально содержит краски 0 и 1. Мы будем записывать набор красок, которые есть на подставке, как [0, 1]. Один возможный способ, по которому помощник может обрабатывать запросы художника, следует ниже.

Первая запрошенная краска (номер 2) отсутствует на подставке для красок. Помощник устанавливает её туда, решая убрать краску 1 с подставки. Текущий набор красок на доске для красок такой: [0, 2].

- Следующая запрошенная краска (номер 0) уже имеется на подставке для красок, так что помощник может отдохнуть.

- Для того, чтобы обработать третий запрос (краска номер 3), помощник убирает краску 0. Теперь на подставке для красок находятся краски [3, 2].

- Наконец, последнюю запрошенную краску (номер 0) нужно забрать с полки для установки на подставку для красок. Помощник решает для этого убрать с подставки краску с номером 2, и подставка теперь содержит набор красок [3, 0].

Заметьте, что в приведенном выше примере помощник не придерживался оптимальной стратегии Леонардо. Оптимальным было бы убрать краску с номером 2 при выполнение третьего запроса, при этом помощник мог бы отдохнуть на последнем шаге.

Стратегия помощника, когда его память ограничена

Утром помощник попросил Леонардо записать последовательность запросов C на листе бумаги, чтобы найти оптимальную стратегию и следовать ей. Но Леонардо хочет сохранить в секрете технику своей работы, поэтому он отказался отдать помощнику лист с

записями. Он разрешил ему только прочитать запись последовательности C и попытаться её запомнить.

К сожалению, у помощника плохая память. Он может запомнить не более M бит информации. Из-за этого он не всегда может запомнить всю последовательность запросов целиком. Но помощник может с умом подойти к вычислению той последовательности битов, которую он запомнит. Мы будем называть эту последовательность *последовательностью-подсказкой* и будем обозначать её A.

Пример 2

Утром помощник берет записку Леонардо с последовательностью C, читает эту последовательность и делает все необходимые выборы. Например, он может отслеживать состояния подставки для красок после каждого из запросов. Так, используя (неоптимальную) стратегию из примера 1, последовательность состояний подставки для красок будет [0, 2], [0, 2], [3, 2], [3, 0]. Заметим, что он знает, что начальное состояние подставки было [0, 1].

Допустим, что M = 16, то есть помощник может запомнить 16 бит информации. Так как N = 4, можно запоминать номер краски, используя 2 бита. По этой причине, 16 бит достаточно, чтобы запомнить описанную выше последовательность состояний подставки для красок. Таким образом, помощник вычисляет следующую последовательность A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`).

В течение дня, помощник сможет, расшифровывая последовательность-подсказку, делать свои выборы.

Очевидно, что при M = 16 помощник может просто запомнить последовательность C вместо этого, используя только 8 бит из доступных 16. Этим примером мы хотели показать, что он может действовать не так, как описано выше, не выдавая каких-либо хороших решений.

Постановка задачи

Требуется написать *две различные программы* на одном и том же языке программирования. Эти программы будут выполняться последовательно, без возможности их взаимодействия.

Первая программа будет использована помощником утром. Эта программа получит последовательность C и должна вычислить последовательность-подсказку A.

Вторая программа будет использована помощником в течение дня. Это программа получит последовательность-подсказку A и будет обрабатывать последовательность C запросов Леонардо. Заметим, что элементы последовательности C будут передаваться программе по одному, и каждый запрос должен быть обработан до получения следующего элемента.

Более строго, ваша первая программа должна реализовать процедуру

`ComputeAdvice`(C, N, K, M), которая получает в качестве аргументов массив C из N целых чисел (каждое в диапазоне 0, …, N - 1), число красок на подставке, и число бит M, которое помощник может запомнить. Программа должна вычислить последовательность-подсказку A, состоящую не более чем из M бит. Программа должна передать последовательность A системе, путем последовательного вызова следующей процедуры для каждого бита из A.

- `WriteAdvice`(B) — добавить в конец текущей последовательности-подсказки A бит B. Вы можете вызвать эту процедуру не более, чем M раз.

Во второй программе вам надо реализовать процедуру `Assist`(A, N, K, R). Этой процедуре будут переданы последовательность-подсказка A, целые числа N и K, описанные выше, и длина R последовательности-подсказки A в битах ($R \leq M$). В этой процедуре вы должны будете выполнить предложенную вами стратегию для помощника, используя следующие процедуры:

- `GetRequest`() — возвращает следующую краску, запрошенную Леонардо. Никакая информация о будущих запросах не разглашается.

- `PutBack`(T) — перемещат краску T с подставки обратно на полку. Вы можете вызвать эту процедуру только в том случае, если краска T находится на подставке.

Ваша процедура `Assist` должна вызывать `GetRequest` ровно N раз, каждый раз получая следующий по порядку запрос Леонардо. После каждого вызова `GetRequest`, если краска, которую запросил Леонардо, находится *не* на подставке для красок, то *должна* быть вызвана процедура `PutBack`(T) с выбранным вами T. В противном случае процедура `PutBack` вызываться *не должна*. Если это сделать не так как описано, то это будет расценено как ошибка и приведет к завершению работы вашей программы. Обратите внимание, что изначально на подставке стоят краски с номерами от 0 до K-1.

Тест будет считаться успешно пройденным, если обе ваши процедуры выполнили все вышеописанные ограничения и количество вызовов процедуры `PutBack` *в точности равно* тому, которое совершается при использовании оптимальной стратегии, описанной Леонардо. Если есть несколько оптимальных стратегий, которые совершают такое же количество вызовов процедуры `PutBack`, то можно выбрать любую из них. То есть не требуется в точности следовать стратегии Леонардо, если есть другая оптимальная стратегия.

Пример 3

Снова рассмотрим пример 2. Предположим, что ваша процедура `ComputeAdvice` вернула A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`). Для взаимодействия с системой вы должны сделать следующую последовательность вызовов процедуры `WriteAdvice`:
`WriteAdvice(0)`, `WriteAdvice(0)`, `WriteAdvice(1)`, `WriteAdvice(0)`, `WriteAdvice(0)`, `WriteAdvice(0)`, `WriteAdvice(1)`, `WriteAdvice(0)`, `WriteAdvice(1)`, `WriteAdvice(1)`, `WriteAdvice(1)`, `WriteAdvice(0)`, `WriteAdvice(1)`, `WriteAdvice(1)`, `WriteAdvice(0)`, `WriteAdvice(0)`.

Ваша вторая процедура `Assist` при запуске получит вышеописанную последовательность A и значения N = 4, K = 2 и R = 16. Процедура `Assist` должна сделать ровно N = 4 вызовов процедуры `GetRequest`. Кроме этого, вам нужно вызвать процедуру `PutBack`(T) после некоторых вызовов процедуры `GetRequest`

Таблица ниже иллюстрирует последовательность вызовов, которые соответствуют неоптимальным выборам помощника из примера 1. Дефис обозначает отсутствие вызова процедуры `PutBack`

`GetRequest()`	Действие
2	`PutBack`(1)
0	-
3	`PutBack`(0)
0	`PutBack`(2)

Подзадача 1 [8 баллов]

- $N \leq 5\,000$.
- Вы можете использовать не более M = 65 000 бит.

Подзадача 2 [9 баллов]

- $N \leq 100\,000$.
- Вы можете использовать не более M = 2 000 000 бит.

Подзадача 3 [9 баллов]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Вы можете использовать не более M = 1 500 000 бит.

Подзадача 4 [35 баллов]

- $N \leq 5\,000$.
- Вы можете использовать не более M = 10 000 бит.

Подзадача 5 [до 39 баллов]

- $N \leq 100\,000$.
- $K \leq 25\,000$.

- Вы можете использовать не более M = 1 800 000 бит.

Количество баллов за эту подзадачу зависит от длины R последовательности-подсказки, которую составила ваша программа. В частности, если R_{max} - это самое большое (по всем тестам) значение длины последовательности-подсказки созданной процедурой `ComputeAdvice`, количество балов будет равно:

- 39 баллам, если $R_{max} \leq 200\,000$;
- 39 $(1\,800\,000 - R_{max}) / 1\,600\,000$ баллов, если $200\,000 < R_{max} < 1\,800\,000$;
- 0 баллов, если $R_{max} \geq 1\,800\,000$.

Детали реализации

Необходимо послать на проверку ровно два файла, содержащих описанные выше процедуры, реализованные на одном и том же языке программирования.

Первый файл должен называться `advisor.c`, `advisor.cpp` или `advisor.pas`. Файл должен содержать реализацию процедуры `ComputeAdvice`, как описано выше, и вызывать процедуру `WriteAdvice`. Второй файл должен называться `assistant.c`, `assistant.cpp` или `assistant.pas`. Файл должен содержать реализацию процедуры `Assist`, как описано выше, и вызывать процедуры `GetRequest` и `PutBack`.

Прототипы всех необходимых процедур представлены ниже.

Программы на C/C++

```
void ComputeAdvice(int *C, int N, int K, int M);
void WriteAdvice(unsigned char a);
```

```
void Assist(unsigned char *A, int N, int K, int R);
void PutBack(int T);
int GetRequest();
```

Программы на Pascal

```
procedure ComputeAdvice(var C : array of LongInt; N, K, M : LongInt);
procedure WriteAdvice(a : Byte);
```

```
procedure Assist(var A : array of Byte; N, K, R : LongInt);
procedure PutBack(T : LongInt);
function GetRequest : LongInt;
```

Эти процедуры должны вести себя так, как описано выше. Конечно, можно реализовывать любые другие процедуры для внутреннего использования. Для программ на C/C++ ваши внутренние процедуры должны быть объявлены как статические `static`, так как они будут слинкованы в один файл. Другим решением будет избегать одинаковых

названий процедур в разных файлах. Нельзя взаимодействовать с файлами и со стандартным входом/выходом.

Также необходимо выполнить следующие инструкции (шаблоны, которые можно найти на компьютере, уже удовлетворяют указанным ниже требованиям).

Программы на C/C++

В начале решения необходимо подключить файлы `advisor.h` и `assistant.h`, в advisor и assistant, соответственно. Это делается вставкой в код строки:

```
#include "advisor.h"
```

или

```
#include "assistant.h"
```

Файлы `advisor.h` и `assistant.h` будут находится на компьютере. Также можно скачать их через веб-интерфейс. Скрипты для компиляции и запуска решения будут доступны аналогичным образом. В частности, нужно будет запустить скрипт `compile_c.sh` или `compile_cpp.sh` (в зависимости от языка программирования) в той папке, где находится решение.

Программы на Pascal

Используйте модули `advisorlib` и `assistantlib`, в файлах advisor и assistant, соответственно. Это делается вставкой в код строки:

```
uses advisorlib;
```

или

```
uses assistantlib;
```

На компьютере будут находится файлы `advisorlib.pas` и `assistantlib.pas`. Их также можно скачать через веб-интерфейс. Скрипты для компиляции и запуска решения будут доступны аналогичным образом. В частности, нужно будет запустить `compile_pas.sh` в той папке, где находится решение.

Пример проверяющего модуля (grader)

Пример проверяющего модуля принимает данные в следующем формате:

- Строка 1: N, K, M;
- Строки со 2 по N + 1: C[i].

Проверяющий модуль сначала вызовет процедуру `ComputeAdvice`. В результате этого

будет сгенерирован файл `advice.txt`, содержащий биты последовательности-подсказки, разделенные пробелом. Файл будет завершен цифрой 2.

После этого будет запущена процедура `Assist`, в результате запуска которой будет сгенерирован вывод, каждая строка которого будет либо в форме "`R [number]`", либо в форме "`P [number]`". Строки первого типа будут обозначать вызов процедуры `GetRequest()` и полученные ответы. Строки второго типа будут обозначать вызовы процедуры `PutBack` и краски, которые забираются с подставки. Вывод будет завершен строкой вида "`E`".

Необходимо обратите внимание, что с использованием официального проверяющего модуля время выполнения программы может отличаться от времени работы программы на локальном компьютере. Это различие не будет значительным. Тем не менее, можно использовать веб-интерфейс для того, чтобы убедиться, что решение укладывается в ограничение по времени.

Ограничения по времени и памяти

- Ограничение по времени: 7 секунд
- Ограничение по памяти: 256 мебибайт