Задача

Участникам предлагается реализовать многопоточную структуру данных с поддержкой пяти методов:

- **`void init(integer threadsNumber)`** — инициализация структуры данных, на вход подается количество потоков, вызывается первым;
- **`void put(integer key, integer value, integer timestamp)`** — добавить элемент с ключом `key` и значением `value` в структуру данных; если элемент с таким ключом уже находится в структуре данных, то необходимо обновить его значение ($0 \le$ **`key`** $\le 10^9$, $-10^{18} \le$ **`value`** $\le 10^{18}$, $0 \le$ **`timestamp`** $\le 10^9$);
- **`void remove(integer key, integer timestamp)`** — удалить элемент с ключом `key` из структуры данных;
- **`integer get(integer key, integer timestamp)`** — получить значение элемента с ключом `key`; если элемента с таким ключом нет в структуре данных, то вернуть -2^{63} (`-9223372036854775808`);
- **`void end(integer timestamp)`** — вызывается из каждого потока по окончании всех вызовов методов **`put`**, **`remove`** и **`get`** для этого потока.

Метод **`init`** вызывается первым. До окончания его работы не производится вызов никаких других методов. Время работы этого метода не учитывается при подсчете времени работы решения участника.

Методы **`put`**, **`remove`**, **`get`** и **`end`** могут вызываться из разных потоков в любом порядке, например, одновременно из трех потоков могут быть вызваны два метода **`remove`** и один **`get`**.

Методы должны применяться к структуре данных в порядке **`timestamp`**. Гарантируется, что **`timestamp`**'ы для каждого потока в отдельности строго возрастают, также все **`timestamp`**'ы различны для всех потоков и распределены более-менее (это неформальное наречное выражение использовано здесь намеренно) равномерно по потокам. Например, точно не будет так, что все **`timestamp`**'ы одного потока больше **`timestamp`**'ов другого.

Гарантируется, что вызовы методов **`remove`** корректны, т.е. не может быть вызовов, в которых пришлось бы удалять элемент, который ещё не добавлен в структуру данных в соответствии с **`timestamp`**.

Решению запрещается производить любые действия (переопределять, писать, закрывать и т.п.) со стандартным потоком ошибок (`standard error`).

Ограничения

- Общее количество вызовов методов `put`, `remove` и `get` не превышает 10^5.
- $3 \le$ `((#put + #remove) / #get)` ≤ 10, где `#` обозначает количество вызовов методов.
- Максимальное количество потоков, из которых могут вызываться методы **`put`**, **`remove`**, **`get`** и **`end`** – 32.
- Ограничение по времени на работу реализуемого модуля — 8 секунд (см. п. Тестирующая инфраструктура).
- Ограничение по памяти для решения и инфраструктуры вместе — 2 ГБ. Максимальный размер стека на тестирующих серверах установлен в 16 МБ.

Описание программного интерфейса для языков программирования

Java

Необходимо реализовать класс `Task` с интерфейсом

```
public class Task {
     public Task(final int threadsNumber);
     public  void  put(final  int  key,  final  long  value,  final  int
timestamp);
     public void remove(final int key, final int timestamp);
     public long get(final int key, final int timestamp);
     public void end(final int timestamp);
}
```

C/C++

Необходимо реализовать класс `Task` с интерфейсом

```
class Task {
     public:
          Task(const int threadsNumber);

          void put(const int key, const long long value, const int
timestamp);
          void remove(const int key, const int timestamp);
          long long get(const int key, const int timestamp);
          void end(const int timestamp);
};
```

Pascal

Необходимо реализовать модуль `Task` с интерфейсом

```
unit task;
interface

type
    TTask = class
       constructor init(const threadsNumber : longint);
       procedure put(const key : longint; const value : int64; const
timestamp : longint);
       procedure remove(const key : longint; const timestamp: longint);
       function get(const key : longint; const timestamp: longint) :
int64;
       procedure finish(const timestamp : longint);
     end;
```

Тестирующая инфраструктура

Для каждого языка программирования, доступного в системе тестирования, написана тестирующая инфраструктура, которая на вход принимает файл *input.txt* и вызывает методы решения участников в соответствии с данными в этом файле. В результате работы инфраструктура записывает в стандартный поток ошибок (`standard error`) число 0, если решение участника дало хотя бы один неправильный ответ на вызов **`get`**, иначе число 1 и через пробел время работы решения участника на данном тесте в миллисекундах. Код тестирующей инфраструктуры есть в прилагаемом архиве. Решения, написанные на Java, запускаются в тестирующей инфраструктуре дважды для прогрева Java-машины, при этом используется время второго запуска.

Запрещается влиять на работу тестирующей инфраструктуры или как-либо использовать данные об ответах и входных данных, которые в ней хранятся. В частности, для решений на Java нельзя использовать тот факт, что оно запускается дважды на одном и том же тесте, например, запоминая ответы.

Оценка решения

На каждом тесте решение участника запускается k раз (k нечётно).

Если хотя бы один из запусков завершился неуспешно (падение / неверный ответ / ...), то результат участника на тесте считается равным бесконечности.

В противном случае результат участника на тесте равняется медиане времени работы решения по всем запускам на тесте:

$$T = \underset{j=1}{\overset{k}{med}}\, t_j$$

Как во время тура, так и после конца тура, рейтинг считается по формуле:

$$R(t_1, \ldots, t_c) = \alpha \sum_{i=1}^{c} \frac{t_i^* + \delta}{t_i + \delta},$$

где:

t_i — результат участника на i-ом тесте,

t_i^* — результат калибровочного решения жюри на i-ом тесте,

α — нормировочный коэффициент,

δ = 30 мс — стабилизационная константа.

При подведении итогов нормировочный коэффициент вычисляется таким образом, чтобы у победителя (участника, имеющего максимальный рейтинг) рейтинг оказался равным 100:

$$max\ R = 100.$$

Во время тура нормировочный коэффициент постоянен. Он определяется таким образом, чтобы калибровочное решение жюри имело рейтинг 100:

$$\alpha = \frac{100}{c}.$$

При этом набор тестов во время тура отличается от финального набора.

Формат файла *input.txt*

В первой строке файла находится единственное целое положительное число N — количество потоков, из которых вызываются методы **`put`**, **`remove`**, **`get`** и **`end`**. Далее следует описание вызовов методов для каждого из N потоков. В первой строке находится общее количество вызываемых методов для каждого потока, далее в отдельной строке находится описание вызываемого метода: первая цифра соответствует методу, который будет вызван: 1 — **`put`**, 2 – **`remove`**, 3 – **`get`**, 4 – **`end`**.

Если вызываемый метод **`put`**, remove или get, то вторым числом будет значение **`key`**. Для команд **`put`** и **`get`** третьим числом будет значение **`value`** (для **`put`** – значение **`value`**, которое будет передано при вызове метода участника, для **`get`** – значение, которое должен вернуть участник). Последним числом для всех команд будет значение timestamp.

Пример

input.txt

```
2
3
3 1 1807 2
2 1 4
4 13
4
1 1 1807 1
3 1 -9223372036854775808 5
1 2 10 10
4 15
```

Комментарии

Из двух потоков параллельно вызываются методы решения участника в соответствии с таблицей:

Поток №1	Поток №2
`get(1, 2)`	`put(1, 1807, 1)`
`remove(1, 4)`	`get(1, 5)`
`end(13)`	`put(2, 10, 10)`
	`end(15)`

Поскольку операции над очередью должны производиться в порядке `timestamp`, последовательность должна быть такой:

```
put(1, 1807, 1)
get(1, 2)
remove(1, 4)
get(1, 5)
put(2, 10, 10)
```

Компиляция

gcc 4.4.3: `g++ -pthread -Wall -static -O2 Executable.cpp -o executable`
Java 1.6.0_37: компиляция – `javac Executable.java`,
запуск – `java -Xmx2g -Xms16m -Duser.language=en_US Executable`
Free Pascal 2.4.0: `fpc -XS -O2 Executable.pas -oexecutable`

Проверка решения

Участникам предоставляется доступ к виртуальной машине под управлением 64-битной ОС Ubuntu 10.04 LTS, на которой можно проверять решения.
Конфигурация компьютеров, на которых будет проводиться тестирование:

Intel® Xeon® Processor E5630

Модель	E5630
Количество ядер	4
Количество потоков	8
Тактовая частота	2.53 GHz
Объем кэша	12 MB
Максимальная пропускная способность памяти	25.6 GB/s

Материалы

Ссылка для скачивания архива с материалами задачи выложена в системе тестирования NSUts.

Пароль к архиву:

Hekatonkheires