

Лабораторная работа 5.

Принципы оптимизации кода.

Задание 1. (1 балл)

В данной задаче необходимо оптимизировать код, использующийся для сортировки студентов по рейтингу. Данные каждого студента находятся в структуре Student, объявленной в файле student.h. Рейтинг студента записан в поле rating:

```
1 struct Student {
2     string first_name;
3     string last_name;
4     map<string, double> marks;
5     double rating;
6
7     bool operator < (const Student& other) const {
8         return GetName() < other.GetName();
9     }
10
11     bool Less(const Student& other) const {
12         return rating > other.rating;
13     }
14
15     string GetName() const {
16         return first_name + " " + last_name;
17     }
18 };
```

Для сортировки студентов по рейтингу используется функция сравнения, возвращающая true, если рейтинг студента first выше рейтинга студента second.

```
1 bool Compare(Student first, Student second) {
2     return first.Less(second);
3 }
```

Было выявлено, что эта функция является узким местом процесса сортировки, и именно её нужно оптимизировать.

Вам дан файл student.cpp в котором представлено некоторое решение, оптимизируйте его.

Задание 2. (1 балл)

Студента попросили написать класс Learner, помогающий изучать иностранный язык. В публичном интерфейсе класса должны быть две функции:

```
1 int Learn(const vector<string>& words);
2 vector<string> KnownWords();
```

Функция `Learn` должна получать порцию слов, "запоминать" их и возвращать количество различных новых слов. Функция `KnownWords` должна возвращать отсортированный по алфавиту список всех выученных слов. В списке не должно быть повторов.

Студент написал следующее решение этой задачи, однако оно почему-то работает очень медленно. Вам надо его ускорить. Вам дан файл `learner.cpp` с медленным решением задачи.

Не меняя публичный интерфейс класса `Learner`, найдите в нём узкие места, исправьте их.

Профилирование кода

Примите во внимание тот факт, что при профилировании с использованием потоков ввода/вывода, скорость работы потоков ввода/вывода намного меньше, чем скорость выполнения обычных инструкций. Это может существенно влиять на итоговые результаты профилирования кода. Поэтому старайтесь вынести ввод/вывод за пределы профилируемого кода.

Узкое место

Обратите внимание, сколько раз в тестирующем коде вызывается метод `KnownWords`, а сколько - `Learn`.

Тестирующий код, на основе которого у вас должна быть реализация:

Ваш код будет тестироваться так:

```
1  int main() {
2      Learner learner;
3      string line;
4      while (getline(cin, line)) {
5          vector<string> words;
6          stringstream ss(line);
7          string word;
8          while (ss >> word) {
9              words.push_back(word);
10         }
11         cout << learner.Learn(words) << "\n";
12     }
13     cout << "=== known words ===\n";
14     for (auto word : learner.KnownWords()) {
15         cout << word << "\n";
16     }
17 }
```

Задание 3. (1 балл)

Вам даны задача и её решение — верное, но не удовлетворяющее заданным ограничениям на время работы.

Условие

Разработайте простейшую систему маршрутизации экспрессов, курсирующих по одному железнодорожному направлению, представляющему собой прямую. Ваша программа должна уметь обрабатывать запросы двух типов:

ADD start finish — добавить в систему маршрутов экспресс, следующий со станции **start** до станции **finish** и обратно. Экспресс не делает промежуточных остановок. Станции задаются целыми числами, равными их расстоянию от вокзала (он имеет номер 0).

GO start finish — попытаться проложить беспересадочный маршрут от станции **start** до станции **finish**. Если существует экспресс между этими двумя станциями, в ответ на данный запрос выведите 0. В противном случае выведите положительное число — минимальное расстояние, на которое можно приблизиться к станции **finish**, стартовав строго на станции **start** и используя не более одного экспресса.

Формат входных данных

В первой строке вводится количество запросов Q — натуральное число, не превосходящее 10^5 . В следующих Q строках в соответствии с описанным выше форматом вводятся запросы. Гарантируется, что номера станций являются целыми числами, по модулю не превосходящими 10^9 .

Формат выходных данных

Для каждого запроса **GO** выведите единственное целое неотрицательное число — минимальное расстояние до конечной станции маршрута, вычисляемое в соответствии с описанными выше правилами.

Ограничения

1 секунда на выполнение всех запросов. Все описанные в условии гарантии действительно справедливы для всех тестов, на которых будет запускаться ваша программа. Проверять корректность тестов не нужно.

Пример

Ввод

```
1 7
2 ADD -2 5
3 ADD 10 4
4 ADD 5 8
5 GO 4 10
6 GO 4 -2
7 GO 5 0
8 GO 5 100
9
```

Вывод

```
1 0
2 6
3 2
4 92
5
```

Комментарий к примеру

В первом запросе GO выгодно воспользоваться экспрессом 10 4. Во втором выгоднее остаться на месте, чем пользоваться экспрессом. В третьем и четвёртом запросах GO необходимо воспользоваться экспрессами -2 5 и 5 8.

Медленное решение представлено в файле *slow.cpp*

Задание 4. (2 балла)

Вам даны задача и её решение — верное, но не удовлетворяющее заданным ограничениям на время работы.

Условие

Разработайте систему стимулирования чтения электронных книг. Для простоты будем считать, что книга всего одна, но её одновременно читают много людей. Необходимо следить за прогрессом чтения у всех пользователей и выводить мотивирующие уведомления. А именно, ваша программа должна обрабатывать следующие события:

READ user page — сохранить факт того, что пользователь под номером user дочитал книгу до страницы page. Если ранее такой пользователь не встречался, необходимо его добавить. Гарантируется, что в рамках одного пользователя номера страниц в соответствующих ему событиях возрастают.

CHEER user — сообщить пользователю user, какая доля существующих пользователей (не считая его самого) прочитала меньшую часть книги, чем он. Если этот пользователь на данный момент единственный, доля считается равной 1. Если для данного пользователя пока не было ни одного события READ, доля считается равной 0, а сам пользователь не учитывается при вычислении долей для других пользователей до тех пор, пока для него не случится событие READ.

Формат входных данных

В первой строке вводится количество запросов Q — натуральное число, не превосходящее 10^6 . В следующих Q строках в соответствии с описанным выше форматом вводятся запросы. Гарантируется, что все вводимые числа целые и положительные, при этом номера пользователей не превосходят 10^5 , а номера страниц не превосходят 1000.

Формат выходных данных

Для каждого запроса CHEER user выведите единственное вещественное число от 0 до 1 — долю пользователей, прочитавших меньше страниц, чем user. Формат вывода этого числа должен быть в точности таким же, как в опубликованном ниже медленном решении.

Ограничения

4 секунды на выполнение всех запросов. Все описанные в условии гарантии действительно справедливы для всех тестов, на которых будет запускаться ваша программа. Проверять корректность тестов не нужно.

Пример

Ввод

```
1 12
2 CHEER 5
3 READ 1 10
4 CHEER 1
5 READ 2 5
6 READ 3 7
7 CHEER 2
8 CHEER 3
9 READ 3 10
10 CHEER 3
11 READ 3 11
12 CHEER 3
13 CHEER 1
14
```

Вывод

```
1 0
2 1
3 0
4 0.5
5 0.5
6 1
7 0.5
8
```

Комментарии к примеру

Пользователь 5 не учитывается при вычислении долей, потому что для него не произошло ни одного события READ.

Пользователь 1 изначально был единственным, но в конце его обогнал 3-й, но не обогнал 2-й, поэтому он оказался продуктивнее 50 % пользователей.

Пользователь 3 изначально обгонял только 2-го и потому получал долю 50 %, но в конце обогнал 1-го и получил долю 100 %.

Медленное решение в файле slow4.cpp

Задание 5. (2 балла)

Условие

Разработайте систему бронирования отелей, позволяющую бронировать номера клиентами и контролировать спрос владельцами отелей.

Пусть система работает с некоторым набором событий, происходящий в некоторые моменты времени *time*.

Пусть подобные моменты времени *time* в системе измеряются в секундах, отсчитываемых от некоторого начального момента времени *time* = 0. Более того, пусть указанная система отсчета времени является **общей и единой** для **всех** определяемых далее в условии задачи событий.

Наконец, пусть *current_time* — время последнего события бронирования **BOOK** (см. ниже) в системе бронирования отелей относительно **всех** имеющихся в системе отелей. Тогда Ваша программа должна обрабатывать следующие события:

- **BOOK** *time hotel_name client_id room_count* — забронировать клиентом *client_id* *room_count* номеров в отеле *hotel_name* в момент времени *time*.
- **CLIENTS** *hotel_name* — вывести количество различных клиентов, бронировавших номера в отеле *hotel_name* за последние сутки относительно последнего события бронирования в системе: $current_time - 86400 < time \leq current_time$, где 86400 — количество секунд в сутках. Обратите внимание, бронирование с временной меткой $current_time - 86400$ учитываться не должно.
- **ROOMS** *hotel_name* — вывести количество номеров, забронированных в отеле *hotel_name* за последние сутки относительно последнего события бронирования в системе. Работа команды определяется образом, аналогичным определению работы команды **CLIENTS**.

Формат входных данных

В первой строке вводится количество запросов *Q* — натуральное число, не превосходящее 10^5 . В следующих *Q* строках в соответствии с описанным выше форматом вводятся запросы. Гарантируется, что:

- *time* — целое число в диапазоне от -10^{18} до 10^{18} и не убывает от события к событию.
- *hotel_name* — строка из латинских букв и цифр, имеющая длину не более 12 символов.
- *client_id* — натуральное число, не превосходящее 10^9 .

- ***room_count*** — целое положительное число, не превосходящее 1000.

Формат выходных данных

Для каждого запроса **CLIENTS** и **ROOMS** выведите единственное целое число — ответ на запрос. Если указанный в запросе отель пока не имеет ни одного бронирования, выведите 0.

Ограничения

1 секунда на выполнение всех запросов. Все описанные в условии гарантии действительно справедливы для всех тестов, на которых будет запускаться ваша программа. Проверять корректность тестов не нужно.

Пример

Ввод

```
1 11
2 CLIENTS Marriott
3 ROOMS Marriott
4 BOOK 10 FourSeasons 1 2
5 BOOK 10 Marriott 1 1
6 BOOK 86409 FourSeasons 2 1
7 CLIENTS FourSeasons
8 ROOMS FourSeasons
9 CLIENTS Marriott
10 BOOK 86410 Marriott 2 10
11 ROOMS FourSeasons
12 ROOMS Marriott
13
```

Вывод

```
1 0
2 0
3 2
4 3
5 1
6 1
7 10
8
```

Комментарии к примеру

После бронирования, случившегося в момент времени 86410, в статистике перестают учитываться бронирования, случившиеся в момент времени 10.

Задание 6 (1 балл).

Словарь задан массивом отсортированных в лексикографическом порядке строк. Напишите программу эффективного поиска слова в словаре.

Входные данные

На вход программе сначала подается искомое слово, во второй строке — число n ($1 \leq n \leq 100000$) — количество слов в словаре. В следующих n строках расположены слова словаря, по одному слову в строке. Все слова состоят только из строчных латинских букв, слова упорядочены по алфавиту (расположены в лексикографическом порядке).

Длина слов не превосходит 20. Пустых слов нет.

Выходные данные

Выведите *YES* или *NO* в зависимости от того, есть искомое слово в словаре или нет.

Примеры

Входные данные
abba 4 a ab aba baba
Выходные данные
NO

Задание 7 (2 балла).

Ваша цель — реализовать симулятор обработки сетевых пакетов. Для i -го пакета известно время его поступления $arrival_i$, а также время $duration_i$, необходимое на его обработку. В вашем распоряжении имеется один процессор, который обрабатывает пакеты в порядке их поступления. Если процессор начинает обрабатывать пакет i (что занимает время $duration_i$), он не прерывается и не останавливается до тех пор, пока не обработает пакет.

У компьютера, обрабатывающего пакеты, имеется сетевой буфер размера $size$. До начала обработки пакеты хранятся в буфере. Если буфер полностью заполнен в момент поступления пакета (есть $size$ пакетов, поступивших ранее, которые до сих пор не обработаны), этот пакет отбрасывается и уже не будет обработан. Если несколько пакетов поступает в одно и то же время, они все будут сперва сохранены в буфер (несколько последних из них могут быть отброшены, если буфер заполнится).

Компьютер обрабатывает пакеты в порядке их поступления. Он начинает обрабатывать следующий пакет из буфера сразу после того, как обработает текущий пакет. Компьютер может простаивать, если все пакеты уже обработаны и в буфере нет пакетов. Пакет освобождает место в буфере сразу же, как компьютер заканчивает его обработку.

Формат ввода

Первая строка входа содержит размер буфера $size$ и число пакетов n .


Каждая из следующих n строк содержит два числа: время $arrival_i$ прибытия i -го пакета и время $duration_i$, необходимое на его обработку. Гарантируется, что $arrival_1 \leq arrival_2 \leq \dots \leq arrival_n$. При этом может оказаться, что $arrival_{i-1} = arrival_i$. В таком случае считаем, что пакет $i - 1$ поступил раньше пакета i .

Все числа во входе целые, $1 \leq size \leq 10^5$; $1 \leq n \leq 10^5$; $0 \leq arrival_i \leq 10^6$; $0 \leq duration_i \leq 10^3$; $arrival_i \leq arrival_{i+1}$ для всех $1 \leq i \leq n - 1$.


Формат вывода

Для каждого из n пакетов выведите время, когда процессор начал его обрабатывать, или -1 , если пакет был отброшен.

Пример 1


Ввод 

```
1 1
0 0
```


Вывод 

```
0
```

Пример 2


Ввод 

```
1 1
0 1
```


Вывод 

```
0
```

Пример 3


Ввод 

```
1 3
0 1
1 3
4 2
```


Вывод 

```
0
1
4
```

Пример 4

Ввод 

```
3 6
0 2
1 2
2 2
3 2
4 2
5 2
```

Вывод 

```
0
2
4
6
8
-1
```