Внимание: Согласно правилам олимпиады, каждая команда может использовать только **один** компьютер. Вы можете использовать дополнительные компьютеры **только** для связи с сокомандниками и для чтения условий задач.

В любой момент времени может быть только один активный компьютер, на котором вы можете писать, компилировать, запускать программы. Активный компьютер может переключаться во время тура, например, чтобы следующую задачу писал другой член команды. Однако нельзя проявлять активность больше чем на одном компьютере одновременно.

Использовать интернет разрешается.

Использовать предварительно написанный код разрешается.

Внимание: Отправляя исходный код решения в систему тестирования, вы соглашаетесь с тем, что представители компании Huawei после конца тура смогут его посмотреть и использовать в исследовательских целях.

Задача 1. Облако

Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Не секрет, что сегодня компании чаще арендуют виртуальные машины в облаке вроде HUAWEI CLOUD вместо того, чтобы использовать собственные физические машины на своей территории. В этом случае у облачного провайдера в дата-центре стоят мощные **серверы**, на каждом из которых работает одна или несколько **виртуальных машин**, возможно принадлежащие разным пользователям.

Запускать на одном сервере много виртуальных машин приходится потому, что пользователю чаще всего не нужны полные мощности сервера для решения его задач, а благодаря размещению нескольких виртуальных машин на сервер в разы снижается стоимость аренды. Разумеется, пользователи лишь выбирают **размер** виртуальной машины, который им нужен, а размещением их машин по серверам дата-центра занимается облачный провайдер.

Облачный провайдер хочет заполнить свои серверы виртуальными машинами как можно «плотнее», чтобы все вычислительные мощности сервера использовались. Набор виртуальных машин часто меняется, поэтому провайдеру приходится периодически **переносить** виртуальные машины между серверами, по возможности поддерживая плотное заполнение серверов. В данной задаче вам предлагается разработать эффективный алгоритм для этого переноса.

Облачный провайдер предоставляет виртуальные машины T размеров. Для каждого размера известно, сколько ядер процессора и сколько гигабайт оперативной памяти получает такая виртуальная машина. Кроме того, один размер выбран как **типовой**: это будет иметь значение позже.

В дата-центре имеется N серверов. Для каждого сервера известно, сколько у него ядер процессора, и сколько гигабайт оперативной памяти. На сервере можно разместить произвольный набор виртуальных машин при условии, что суммарное количество ядер по размещаемым виртуальным машинам не превышает количество ядер сервера, и суммарное требование виртуальных машин по оперативной памяти не превышает объём памяти сервера. Это условие называется **ограничением по ресурсам**.

Иногда пользователи хотят, чтобы их виртуальные машины были размещены на разных серверах. Чтобы это требование удовлетворить, задано G **групп**. Для каждой группы заданы виртуальные машины, которые в неё входят. Все машины одной группы должны быть размещены на разных серверах — назовём это **ограничением по группам**.

Изначально в дата-центре размещено M виртуальных машин: размер каждой машины известен, ограничения по ресурсам и по группам выполнены. Нужно сделать большую перестановку виртуальных машин, чтобы улучшить плотность заполнения серверов.

К сожалению, на перенос всего подряд нет времени. Для каждой виртуальной машины известен **штраф переноса** — некоторое неотрицательное число. Сумма штрафов по всем переносимым машинам не должна превышать заданный **бюджет переноса**. Следует заметить, что некоторые виртуальные машины вообще нельзя перемещать: в этом случае у них штраф переноса больше бюджета.

Насколько плотно заполнены сервера, определяется по тому, сколько виртуальных машин типового размера можно на них добавить. Напоминаем, что один размер выбран как

типовой (например, наиболее популярный у пользователей). **Потенциалом** сервера называется количество виртуальных машин типового размера, которое можно разместить на этом сервере дополнительно без нарушения ограничения по ресурсам.

Требуется найти план переноса виртуальных машин между серверами, при котором:

1. Суммарный штраф за перенос машин не превышает бюджета переноса.
2. После завершения плана: ограничения по ресурсам и по группам выполнены.
3. После завершения плана: сумма потенциалов серверов как можно больше.

Следует заметить, что нас не интересуют промежуточные состояния по ходу выполнения плана: можно смело нарушать ограничения, лишь бы после завершения плана всё было корректно.

Пункты 1 и 2 задают жёсткие ограничения: если они не выполнены, то ответ вашей программы считается полностью некорректным. Пункт 3 определяет, как будет оцениваться корректный ответ: чем больше сумма потенциалов в вашем ответе, тем больше вы получаете баллов за тест, и выше оказываетесь в рейтинге.

Формат входных данных

В первой строке записано шесть целых чисел: T — количество размеров, разрешённых провайдером, t_0 — номер типового размера, M — количество виртуальных машин в дата-центре, N — количество серверов в дата-центре, G — количество групп, B — бюджет переноса ($1 \leqslant t_0 \leqslant T \leqslant 25$, $1 \leqslant M \leqslant 100\,000$, $1 \leqslant N \leqslant 2\,000$, $0 \leqslant G \leqslant 100$, $1 \leqslant B < 10\,000\,000$).

В следующих T строках описаны размеры виртуальных машин. Для каждого размера указано два целых числа: c_t — количество ядер процессора и m_t — количество гигабайт памяти ($1 \leqslant c_t \leqslant 128$, $1 \leqslant m_t \leqslant 1024$). Размер под номером t_0 является типовым.

В следующих N строках описаны серверы дата-центра. Для каждого сервера указано два целых числа: c_i — количество ядер процессора и m_i — количество гигабайт памяти ($1 \leqslant c_i \leqslant 128$, $1 \leqslant m_i \leqslant 1024$).

В следующих M строках описаны виртуальные машины и их начальное размещение в дата-центре. Для каждой машины указано три целых числа: t_j — её размер, i_j — на каком сервере размещена эта машина изначально, b_j — штраф переноса этой машины ($1 \leqslant t_j \leqslant T$, $1 \leqslant i_j \leqslant N$, $1 \leqslant b_j \leqslant 10\,000\,000$).

В оставшихся G строках описаны группы. Для каждой группы в отдельной строке указано сначала целое число q_g — размер группы, а затем q_g целых чисел j_{gl} — номера виртуальных машин, которые состоят в этой группе ($2 \leqslant q_g \leqslant N$, $1 \leqslant j_{gl} \leqslant M$). Гарантируется, что в каждой группе номера машин различные, и что никакая машина не входит в несколько групп одновременно.

Объекты каждого типа нумеруются в порядке перечисления во входном файле, начиная с единицы.

Гарантируется, что начальное размещение виртуальных машин удовлетворяет ограничениям по группам и по ресурсам. Между соседними блоками во входном файле добавлены пустые строки (см. пример).

Формат выходных данных

В первую строку требуется вывести три целых числа: K — сколько переносов нужно сделать, B_0 — суммарный штраф этих переносов, P — суммарный потенциал по всем серверам после выполнения переносов ($0 \leqslant K \leqslant M$, $0 \leqslant B_0 \leqslant B$, $0 \leqslant P$).

В оставшиеся K строк выведите переносы. Для каждого переноса выведите три целых числа: j_k — номер виртуальной машины, которую надо перенести, s_k — номер сервера, с которой надо машину убрать, e_k — номер сервера, на которую машину надо добавить ($1 \leqslant j_k \leqslant M$, $1 \leqslant s_k, e_k \leqslant N$). Все j_k должны быть различными.

Пример

input.txt	output.txt
6 6 8 4 2 1000 2 10 3 5 4 5 1 15 2 20 2 15 5 25 8 20 7 50 8 30 1 1 300 2 1 200 3 2 300 4 2 200 5 4 500 3 3 300 1 3 1100 1 4 400 4 1 3 5 7 4 2 4 6 8	3 900 1 4 2 3 6 3 4 8 4 2

Пояснение к примеру

В примере 4 сервера и 8 виртуальных машин. Типовой размер последний: 2 ядра и 15 ГБ памяти. Задано 2 группы, из-за которых нельзя разместить две машины с номерами одинаковой чётности на одном сервере. Машину номер 7 нельзя перемещать, т.к. штраф её переноса 1100 больше бюджета 1000.

Изначально на первом сервере машины 1 и 2, а втором — 3 и 4, на третьем — 6 и 7, на четвёртом — 5 и 8. На втором и четвёртом серверах используется вся имеющаяся память, на первом — все имеющиеся ядра, а на третьем всего одно свободное ядро. Поэтому если ничего не переносить, то потенциал всех серверов нулевой. Учтите, что в предварительных и финальных тестах такой ситуации быть не может: суммарный потенциал изначально всегда положительный.

Приведённое решение предлагает переместить машины 4, 6 и 8, чтобы распределение стало: машины 1 и 2 на первом сервере, 3 и 8 на втором, 4 и 7 на третьем, 5 и 6 на четвертом. В результате на третьем сервере получается свободно 4 ядра и 25 ГБ памяти, и на это место можно потенциально уместить одну дополнительную машину типового размера.

В принципе, в данном тесте для получения суммарного потенциала 1 можно было просто поменять машины 4 и 6 местами. При этом суммарный штраф переноса был бы 500, а не 900.

Тесты

Есть три категории тестов:

- Пример (приведён выше).
- Предварительные тесты.
- Финальные тесты.

Пример написан вручную, все остальные тесты сгенерированы одним генератором, исходный код которого выдаётся в архиве с материалами.

Предварительных тестов 30 штук, они сгенерированы со следующими параметрами:

```
gen_random  10    10 10    0  0   0   0.25   >tests/1.in
gen_random  25    20 30    0  0   0.1 0.25   >tests/2.in
gen_random  25    30 99    3  5   0   0.2    >tests/3.in
gen_random  10    40  3    0  0   0.1 0.5    >tests/4.in
gen_random  25    50  1    5  5   0.1 0.3    >tests/5.in
gen_random  25    60  3    0  0   0   0.25   >tests/6.in
gen_random  10    70 10    0  0   0.1 0.2    >tests/7.in
gen_random  25    80  1    5 10   0   0.25   >tests/8.in
gen_random  10    90 99    0  0   0.1 0.3    >tests/9.in
gen_random  25   100 30    0  0   0   0.25   >tests/10.in
gen_random  25   200 99    0  0   0   0.25   >tests/11.in
gen_random  25   300  1   10 10   0.2 0.3    >tests/12.in
gen_random  25   400 30    0  0   0.1 0.5    >tests/13.in
gen_random  25   500  3    5 40   0   0.2    >tests/14.in
gen_random  25   600 10   30 10   0   0.25   >tests/15.in
gen_random  25   700  1    0  0   0.2 0.25   >tests/16.in
gen_random  25   800  3    0  0   0.1 0.3    >tests/17.in
gen_random  10   900 10    0  0   0   0.25   >tests/18.in
gen_random  25  1000 30   30 10   0.1 0.2    >tests/19.in
gen_random  25  1200 99    0  0   0   0.5    >tests/20.in
gen_random  25  1500 99    0  0   0.1 0.25   >tests/21.in
gen_random  10  2000  1   99 10   0   0.3    >tests/22.in
gen_random  25  1999  3    0  0   0.1 0.25   >tests/23.in
gen_random  25  1997 10    0  0   0.2 0.25   >tests/24.in
gen_random  25  1983 30    0  0   0   0.35   >tests/25.in
gen_random  25  2000 30   10 99   0.1 0.25   >tests/26.in
gen_random  25  2000  1    0  0   0.1 0.2    >tests/27.in
gen_random  25  2000 10    0  0   0   0.2    >tests/28.in
gen_random  10  2000 99    0  0   0.1 0.3    >tests/29.in
gen_random  25  2000  3   10 10   0   0.5    >tests/30.in
```

Финальные тесты сгенерированы этим же скриптом, запущенным Z раз с разными random seed-ами — всего получается $30Z$ финальных тестов. Таким образом, по размерам и по характеру финальные тесты похожи на предварительные. Число Z жюри определит в зависимости от длительности тестирования, но оно будет не меньше 3.

Система оценки

Правила определения рейтинга таковы. Эталонное решение — это решение, которое никогда не переносит никакие виртуальные машины. Пусть решение участника на тесте получило суммарный потенциал P, а эталонное решение получило суммарный потенциал $P_0 > 0$. Тогда за тест участнику начисляется $\max(P - P_0, 0)/P_0$ очков. Например, если P в три с половиной раза больше P_0, то начисляется 2.5 очков. Если решение отработало некорректно или вывело некорректный ответ, то за тест участник получает ноль очков. Участники ранжируются в рейтинге по сумме очков на всех тестах набора: чем больше очков, тем лучше место.

Когда вы отправляете решение в систему тестирования nsuts, оно проверяется на предварительных тестах. Во время тура в системе показывается текущий рейтинг по предварительным тестам. Этот предварительный рейтинг **не** имеет никакого значения с точки зрения определения результатов олимпиады.

После конца соревнования **последнее успешно скомпилированное** решение от каждого участника попадёт в финальное тестирование, которое будет проведено на финальных тестах. В зачёт первой номинации идёт рейтинг по финальный тестам.

Во время тура вы можете увидеть детальный отчёт о работе вашего решения. Для этого нужно зайти на вкладку «Результаты». Сумму очков на предварительных тестах можно увидеть в столбце «Суммарный балл». Кроме того, вы можете кликнуть по ссылке «открыть отчёт» в столбце «Отчёт о тестировании», и увидеть информацию по тестам. Для каждого теста в таблице показан полученный суммарный потенциал («Потенциал»), очки («Очки»), а также время работы.

Материалы

Вы можете скачать архив материалов в системе nsuts на вкладке «Новости» — там же, где вы скачали это условие.

Содержимое архива:

Файл или директория	Описание содержимого
`check.cpp`	Проверяющая программа.
`gen_random.cpp`	Программа для генерации тестов.
`gen.cmd`	Скрипт для генерации предварительных тестов (приведён выше в условии).
`tests/*.in`	Набор предварительных тестов, полученных запуском `gen.cmd`.
`statement.pdf`	Это условие задачи.
`problem.h`	Общая часть кода чекера и генератора.
`testlib.h`	Служебный файл, используемый для компиляции генератора и чекера.
`check.exe`	Предсобранный исполняемый файл для `check.cpp`.
`gen_random.exe`	Предсобранный исполняемый файл для `gen_random.cpp`.

Чекер и генератор

Проверяющую программу и генератор можно скомпилировать примерно так:

```
cl check.cpp /O2 /EHsc
cl gen_random.cpp /O2 /EHsc
g++ check.cpp -std=c++11 -O2 -o check
g++ gen_random.cpp -std=c++11 -O2 -o gen_random
clang++ check.cpp -std=c++11 -O2 -o check
```

```
clang++ gen_random.cpp -std=c++11 -O2 -o gen_random
```

Проверяющая программа запускается с тремя параметрами:

```
check  input.txt output.txt output.txt
```

Первые два параметра — входной и выходной файл, а третий параметр должен указывать на любой файл (его содержимое не используется). Результаты проверки выводятся в stderr, то есть в консоль. Также суммарный потенциал пишется в `score.txt`.

Генератор запускается с восемью параметрами:

```
gen_random  T N SzParam  G Qmax  Pimm Rmove  (seed)  >input.txt
```

Первые два параметры — это характерные размеры T, N из условия. `SzParam` влияет на то, сколько примерно виртуальных машин входит на один сервер: чем больше значение, тем больше будет машин. Параметр `G` определяет количество групп, а `Qmax` — максимальный размер каждой группы. Вещественное число `Pimm` задаёт примерно долю машин, которые нельзя переносить, а число `Pmove` — какую примерно долю из переносимых машин можно перенести в рамках бюджета.

Последний параметр `(seed)` опциональный, нужен для инициализации генератора случайных чисел. При повторных запусках с одинаковыми параметрами генератор должен выдавать одинаковые данные. Последний элемент `>input.txt` перенаправляет вывод в файл `input.txt` (без этого генератор пишет входные данные для теста в `stdout`, то есть в консоль).